IBM Field Engineering

Theory of Operation

www.IBM360.info

2065 Processing Unit, Volume 1

Preface

This manual describes the operation of the 2065 Processing Unit. It is assumed that the reader has a knowledge of processors, of ALD interpretation, and of the basic circuits used in the 2065.

The EC levels of the ALD's and CLD's for the basic 2065, upon which this manual and its companion maintenance diagram manual are based, are:

ALD's: EC 705369 9/68
CLD's: EC 705340 3/68
Power: EC 711576 8/68

The manual consists of two volumes, and is divided into six chapters and three appendices. Volume 1, Form Y27-2036-0, contains:

Chapter 1, Introduction. Discusses system organization and data flow; character codes, instruction formats, and operands; program execution and control; and the CPU functional units and the Universal instruction set.

Chapter 2, Functional Units. Analyzes registers, adders, and counters individually, except for those units that work together to perform a specific function (for example, variable-field-length register and its associated byte counter).

Volume 2, Form Y27-2037-0, contains:

Chapter 3, Principles of Operation. Presents a detailed analysis of instruction fetching, and instruction by instruction class.

Chapter 4, Features. Discusses the features available for the 2065 CPU.

Chapter 5, Power Distribution and Control. Describes the power distribution and control within the CPU (making a distinction between 2065's and 2060's that have been converted to 2065's) and within the system.

Chapter 6, Console Controls and Maintenance Features. Discusses the controls on the system control panel and on the CE panel and their application, and the maintenance features available.

Appendix A, Special Circuits. Discusses the special circuits in the 2065.

Appendix B, World Trade Differences. Discusses the major difference between the World Trade version of the Model 65 and the domestic version.

Appendix C, Example of FLT Generation. Discusses FLT generation, using a simple four-block tree as an example.

Volume 2 also contains the index for the complete manual.

Following most paragraph heads are bullets (key statements preceded by •) which summarize significant points about the subject. The bullets serve two functions: (1) they provide the CE with the key points of the topic, and (2) they provide quick reference for review and recall for the CE who is familiar with the machine. Detailed text follows, providing the non-classroom student with the fill-in material necessary for self-instruction.

The diagrams supporting the text are divided into two groups: (1) purely instructional diagrams and (2) maintenance-oriented diagrams and diagrams that aid recall. Examples of the first group are high-level block diagrams and diagrams that show general data flow and timing considerations. These diagrams are generally not affected by engineering changes, and, if they include AND/OR logic blocks, the blocks are drawn in positive logic convention and do not maintain ALD lines or line names. The instruction diagrams, which are placed in this manual and called "Figures", are numbered consecutively within a chapter. (For example, 1-1 is the first figure in Chapter 1; 3-7 is the seventh figure in Chapter 3.)

Second Edition (November 1973)
This is a reprint of SY27-2306-0 incorporating changes released in the following Technical Newsletter:

SY27-2258 (dated October 16, 1969).

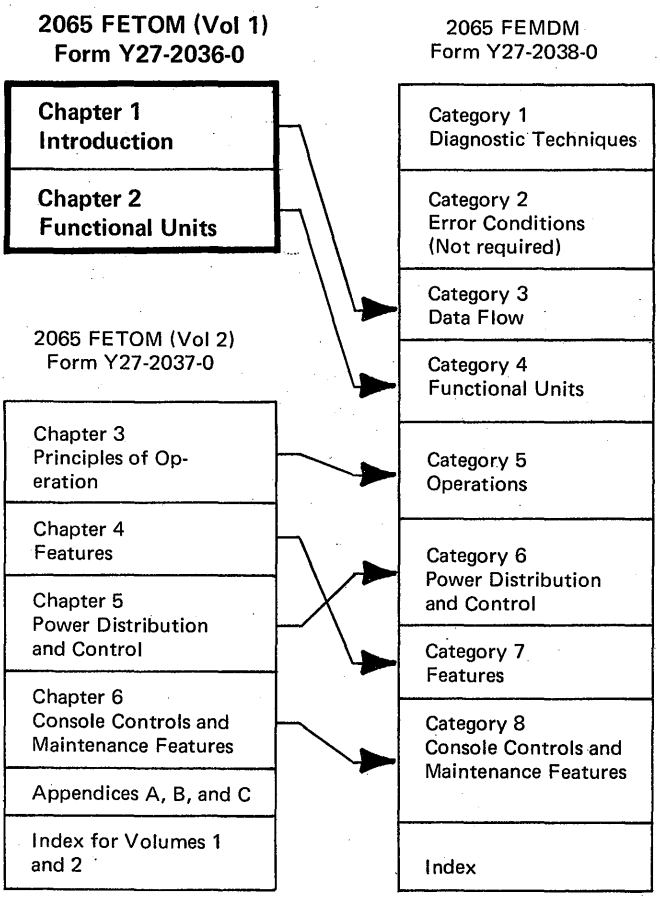
Changes are periodically made to the specifications herein; any such changes will be reported in subsequent revisions or Technical Newsletters.

Text for this manual has been prepared with the IBM SELECTRIC® Composer.

This manual has been prepared by the IBM System Products Division, Product Publications, Dept. B97, PO Box 390, Poughkeepsie, N.Y. 12602. A form for readers' comments is provided at the back of this publication. If the form has been removed, comments may be sent to the above address. Comments become the property of IBM.

The diagrams of the second group are referenced in this manual (for example, Diagram 5-30, FEMDM) but are located in the companion FE Maintenance Diagrams Manual to allow ready reference during maintenance and to facilitate updating the diagrams to new engineering levels. These diagrams are grouped by categories similar to the chapters of this manual.

The relationship of this manual to the FEMDM is shown below. (Arrows indicate cross-referencing between chapters in this manual and categories of diagrams in the FEMDM: for example, most references in Chapter 2 are made to Category 4 diagrams.)



Companion, related, and prerequisite manuals and standards are:

2065 Processing Unit
FEMDM, Form Y27-2038-0
FEMM, Form Y27-2270-0

IBM System/360 Principles of Operation, SRL, Form A22-6821-7.

2065 Processing Unit, 7070/7074 Compatibility Feature
FETOM, Form Y27-2106-0
FEDM, Form Y27-2107-0

2065 Processing Unit, 7080 Compatibility Feature
FETOM, Form Y27-2090-0
FEDM, Form Y27-2091-0

2065/2067 Processing Unit, 709/7040/7044/7090/7094/7094II Compatibility Feature
FETOM, Form Y27-2098-0
FEDM, Form Y27-2099-0

2365 Processor Storage
FETOM, Form Y22-6608-0
FEDM, Form Y22-6601-1
FEMM, Form Y22-6600-1

2361 Core Storage
FETOM, Form Y22-2897-0
FEDM, Form Y22-2895-0
FEMM, Form Y22-2894-0

2860 Selector Channel
FETOM, Form Y27-2220-0
FEMDM, Form Y27-2221-0
FEMM, Form Y22-2893-1

2870 Multiplexer Channel (70,000 Series)
FETOM, Form Y27-2152-0
FEDM, Form Y27-2153-0
FEMM, Y27-2154-0

1052 Adapter and 2150 Console, FETOM, Form Y22-2808

SLT Component Circuits, FEMI, Form Z22-2798 (IBM Confidential)

SLT Power Supplies, FEMI, Form 223-2799

SLT Packaging, FEMI, Form 223-2800

Control Automation System (CAS) Logic Diagram (CLD), IBM Corporate Engineering Standard, CES 0-1046-4

(THIS VOLUME)

Chapter 1 Introduction 1-1

SECTION 1 SYSTEM DESCRIPTION 1-1
Basic System 1-1
Features Available 1-2
System Data Flow 1-3
Main Storage 1-3
2365 Processor Storage 1-4
2361 Core Storage (Optional) 1-4
2065 Central Processing Unit 1-5
Control Section 1-5
Bus Control Unit 1-5
Instruction Fetching Section 1-5
Instruction Execution Section 1-6
Input/Output 1-6
Channel 1-7
Modes of Operation 1-7
Types of Channels 1-7
Control Units 1-8
I/O Devices 1-8
Multisystem Configurations 1-8

SECTION 2 SYSTEM CODING 1-12
Hexadecimal Number System 1-12
Eight-Bit Zoned Character Codes 1-12
Instruction Coding 1-13
Instruction Formats 1-13
Operand Addressing 1-14
Effectively Addressed Operands 1-14
Immediate Operands 1-15
Operands in Local Storage 1-15
Data Formats 1-15
Fixed-Point Data 1-16
Number Representation 1-16
Formats 1-16
Floating-Point Data 1-17
Number Representation 1-17
Formats 1-19
Normalization 1-19
Decimal Data 1-20
Number Representation 1-20
Formats 1-21
Logical Data 1-21

SECTION 3 PROGRAM EXECUTION AND CONTROL . 1-22
Control Program 1-22
Program States 1-23
Problem/Supervisor 1-23
Operating/Stopped 1-24
Running/Wait 1-24
Interruptable/Masked 1-24
Program Status Word 1-25
Interruptions and Exceptional Conditions 1-26
Interruptions 1-29
Interruption Masking 1-29
System Mask Field 1-32
Machine-Check Mask Bit 1-32
Program Mask Field 1-32
Instruction Address Determination 1-32
Machine-Check Interruption 1-33
Program Interruptions 1-33
Supervisor-Call Interruption 1-35
External Interruptions 1-35
I/O Interruptions 1-35
Exceptional Conditions 1-36
Timer Exceptional Condition 1-36
CPU Store In Progress Exceptional Condition 1-36
Manual Control Stop Exceptional Condition 1-36
Manual Control Wait Exceptional Condition 1-36
Manual Control Repeat Exceptional Condition 1-36
Program Store Compare Exceptional Condition . . . 1-36
Invalid Instruction Address Test Exceptional Condition 1-36
Q-Register Refill Exceptional Condition 1-36
Control of I/O Operations 1-36
Instructions, Commands, and Orders 1-37
I/O Control Words 1-37
Channel Address Word 1-37
Channel Command Word 1-37
Channel Status Word 1-38
I/O System Operation 1-38

SECTION 4 CPU DESCRIPTION 1-39
Control 1-39
CPU Timing 1-39
Data Transfer 1-39
Read-Only Storage 1-39
Relationship of ROS to Conventional Controls 1-40
ROS Word 1-41
ROS Addressing and Branching 1-42
No Branch Specified 1-43
Y- and/or Z-Branches 1-43
X-Branches 1-43
Overriding Branches 1-47
ROS Data Flow 1-47
ROS Control of CPU 1-48
PSW Register 1-50
Bus Control Unit 1-50
Major Interface Lines 1-51
BCU Clocks 1-53
BCU Operation 1-53
CPU Request 1-54
Channel Request 1-54
Operation with Main Storage 1-54
Instruction Fetching 1-55
Functional Units Used 1-55
Q-Register 1-55
R-Register 1-56
E-Register 1-56
Instruction Counter 1-56
D-Register 1-58
Instruction Path 1-58
Prefetching of Operands 1-60
Obtaining New Instructions from Main Storage 1-63
CPU Interruption and Exceptional Condition Recovery 1-64
Instruction Execution 1-64
Functional Units Used 1-64
AB Register 1-64
ST Register 1-65

AB and ST Byte Counters . . . 1-65
Mark Triggers . . . 1-65
F-Register . . . 1-66
G-Register . . . 1-66
Serial Adder . . . 1-66
Arithmetic Functions . . . 1-66
Logical Functions . . . 1-68
Parallel Adder . . . 1-68
Local Storage . . . 1-69
Status Triggers . . . 1-69
Fixed-Point Instructions . . . 1-70
Instruction Formats . . . 1-70
Data Flow . . . 1-70
Program Interruptions . . . 1-70
Condition Codes . . . 1-75
Floating-Point Instructions . . . 1-75
Instruction Formats . . . 1-76
Data Flow . . . 1-76
Program Interruptions . . . 1-81
Condition Codes . . . 1-82
Decimal Instructions . . . 1-82
Data Handling . . . 1-82
Instruction Format . . . 1-86
Data Flow . . . 1-86
Program Interruptions . . . 1-87
Condition Codes . . . 1-87
Logical Instructions . . . 1-87
Instruction Formats . . . 1-87
Data Flow . . . 1-90
Program Interruptions . . . 1-91
Condition Codes . . . 1-91
Branching Instructions . . . 1-91
Instruction Formats . . . 1-92
Data Flow . . . 1-94
Program Interruptions . . . 1-94
Condition Codes . . . 1-94
Status Switching Instructions . . . 1-94
Instruction Formats . . . 1-95
Data Flow . . . 1-95
Program Interruptions . . . 1-97
Condition Codes . . . 1-97
Input/Output Instructions . . . 1-98
Instruction Format . . . 1-98
Data Flow . . . 1-98
Program Interruption . . . 1-99
Condition Codes . . . 1-99
Power . . . 1-99

Chapter 2 Functional Units . . . 2-1

SECTION 1 TIMING AND CLOCK CONTROL . . . 2-1
Clock Signal Generators . . . 2-1
Model G65, H65, and I65 CPU Clock Signal Generator . . . 2-1
Model IH65 and J65 CPU Clock Signal Generator . . . 2-1
Clock Timing . . . 2-2
Clock Control and Signal Distribution . . . 2-3

SECTION 2 READ-ONLY STORAGE . . . 2-6
Capacitive Read-Only Storage Array . . . 2-6
CROS Electrical Theory . . . 2-6
CROS Planes . . . 2-6
Drive and Balance Lines (Bit Plates) . . . 2-6
Sense Lines . . . 2-9
Bit Capacitors . . . 2-10
Physical Package . . . 2-10
ROS Addressing . . . 2-10
Read-Only Storage Address Register . . . 2-12
ROSAR(0–5) . . . 2-13
ROSAR(6–9) . . . 2-13
ROSAR(10) . . . 2-13
ROSAR(11) . . . 2-13
ROSAR(0–10) Decoding . . . 2-13
Strobed Drive Lines . . . 2-14
Select Lines . . . 2-14
Array Drivers . . . 2-14
Sense Amplifiers . . . 2-14
ROSAR(11) Function . . . 2-14
ROS Data Flow . . . 2-14
ROS Sense Latches . . . 2-14
ROS Data Register and ROSDR Latches . . . 2-14
ROS Decoders . . . 2-15
ROS Timing . . . 2-15
Maintenance Aids . . . 2-16
ROSAR Latches . . . 2-16
ROS Previous Address Registers . . . 2-16
ROSPARA and ROSPARB Alternator . . . 2-16
ROS Back-Up Register . . . 2-18
ROS Error Checking . . . 2-18
Scan Mode Operations . . . 2-18

SECTION 3 BUS CONTROL UNIT . . . 2-20
General Description . . . 2-20
Basic Interface Considerations . . . 2-20
Basic Operating Considerations . . . 2-21
Operation with Processor . . . 2-21
Operation with I/O Channels . . . 2-22
Operation with Main Storage . . . 2-22
Operation with LCS (Optional Feature) . . . 2-24
Basic Control and Timing Considerations . . . 2-24
Basic Operational Sequence . . . 2-26
Detailed Analysis of BCU Functions . . . 2-28
Initial Handling of Requests . . . 2-28
Establishing Priority . . . 2-29
Gating the Address to SAB . . . 2-29
Stopping the CPU Clock . . . 2-29
Selecting the Storage Unit . . . 2-30
Converting SAB Parity . . . 2-34
Generating 'Select' Signal to Storage . . . 2-34
Generating 'Select' Signal if LCS Is Not in System . . . 2-34
Generating 'Select' Signal if LCS Is in System . . . 2-37
Detection of Invalid Address . . . 2-38
Recording of Error Indications from Storage . . . 2-38
Resetting of BCU Logic . . . 2-38
Detailed Analysis of BCU Operations . . . 2-40
CPU Storage Requests . . . 2-40
3- and 4-Cycle Fetch Operations . . . 2-40
Store Operation . . . 2-40
Insert-Key Operation . . . 2-41
Set-Key Operation . . . 2-41
Test-and-Set Operation . . . 2-41
Single-Cycle Operation . . . 2-41
Channel Storage Requests . . . 2-42

SECTION 4 DATA AND CONTROL REGISTERS . . . 2-44
Q-Register . . . 2-44
Input . . . 2-44
Op-Code Transfer . . . 2-44

B-Field and D-Field Transfer . . . 2-44
B-Field Transfer . . . 2-45
D-Field Transfer . . . 2-45
R-Register . . . 2-46
Input . . . 2-46
Output . . . 2-46
Predecoding . . . 2-46
E-Register . . . 2-46
Input . . . 2-46
Output . . . 2-46
Incrementers . . . 2-46
Instruction Counter . . . 2-49
Input . . . 2-49
Output . . . 2-49
Incrementing IC(0–20) . . . 2-50
Incrementing IC(21–23) . . . 2-50
D-Register . . . 2-50
Input . . . 2-50
Output . . . 2-50
Operational Functions . . . 2-51
Branch and Execute Operations . . . 2-51
Shift Operations . . . 2-52
VFL Operations . . . 2-52
Fixed-Point Operations . . . 2-52
Floating-Point Operations . . . 2-52
Manual-Control Operations . . . 2-52
I/O Operations . . . 2-52
Interruption Operations . . . 2-52
AB Register . . . 2-52
Input . . . 2-52
Output . . . 2-54
ST Register . . . 2-54
Input . . . 2-54
Output . . . 2-54
AB and ST Byte Counters . . . 2-54
AB Byte Counter . . . 2-54
ST Byte Counter . . . 2-57
Mark Triggers . . . 2-58
F-Register . . . 2-58
Input . . . 2-58
Output . . . 2-58
G-Register . . . 2-58
PSW Register . . . 2-58
MCW Register . . . 2-59

SECTION 5 LOCAL STORAGE . . . 2-61
Addressing and Data Flow . . . 2-61
Data Transfer Controls . . . 2-61
Read LS Operation . . . 2-61
Write LS Operation . . . 2-63
LS Timing . . . 2-63

SECTION 6 SERIAL AND PARALLEL ADDERS . . . 2-64
Serial Adder . . . 2-64
Input and Output . . . 2-64
Adder Operation . . . 2-64
Controls . . . 2-64
Functional Description . . . 2-66
Binary Add . . . 2-67
Decimal Operation . . . 2-67
Logical Functions . . . 2-68
Parity Correction . . . 2-68
Error Detection . . . 2-72
Parallel Adder . . . 2-73
Data Input . . . 2-73
Individual Bit-Position Logic . . . 2-73
Half-Adder . . . 2-76
Carry-Into-Bit Logic . . . 2-76
Full-Sum Logic . . . 2-76
Latch-Shifter Logic . . . 2-76
Carry Lookahead . . . 2-77
Group-Level Carry Logic . . . 2-77
Section-Level Carry Logic . . . 2-78
Section-Level Carry-Into Logic . . . 2-78
Group-Level Carry-Into Logic . . . 2-78
Bit-Level Carry-Into Logic . . . 2-78
Full-Sum Development . . . 2-78
Arithmetic Function Sequence . . . 2-81
Parity-Predict Logic . . . 2-82
Error Checking . . . 2-84
Half-Sum Checking . . . 2-84
Full-Sum Checking . . . 2-84
Convert-to-Decimal Operation . . . 2-85
Set Condition Code . . . 2-85

SECTION 7 STATUS AND CONTROL TRIGGERS . . . 2-86
STAT A . . . 2-86
STAT B . . . 2-86
STAT C . . . 2-87
STAT D . . . 2-88
STAT E . . . 2-88
STAT F . . . 2-88
STAT G . . . 2-88
STAT H . . . 2-89
Control Triggers . . . 2-89

(VOLUME 2)

Chapter 3 Principles of Operation . . . 3-1

SECTION 1 INSTRUCTION FETCHING . . . 3-1
Basic End-Op Cycle . . . 3-1
Prefetching of Operands During End Op . . . 3-2
Fetching of Instructions by End-Op Micro-Order . . . 3-3
Requests During End Op . . . 3-3
Requests During Early End Op . . . 3-3
Selection of I-Fetch Microprogram . . . 3-4
Basic RR I-Fetch . . . 3-5
Basic RX I-Fetch . . . 3-6
Basic RS and SI I-Fetch . . . 3-6
Basic SS I-Fetch . . . 3-7
Address Storage Compare (ASC) Test . . . 3-9
I-Fetch Microprogram . . . 3-9
I-Fetch Control If at End Op IC(21,22) = 00 . . . 3-11
I-Fetch Control If at End Op IC(21,22) = 01 . . . 3-12
I-Fetch Control If at End Op IC(21,22) = 10 . . . 3-12
I-Fetch Control If at End Op IC(21,22) = 11 . . . 3-12
Deviations from Basic End Op and I-Fetch . . . 3-12
I-Fetch Sequencers . . . 3-12
Block I-Fetch Trigger . . . 3-13
Timer Exceptional Condition . . . 3-14
CPU Store In Progress Exceptional Condition . . . 3-14
Machine Check Interruption . . . 3-14
Program Interruption . . . 3-15
Supervisor Call Interruption . . . 3-15
External Interruption . . . 3-16
I/O Interruption . . . 3-16
Common Interruption Routine . . . 3-17
Stop, Wait, and Repeat Exceptional Conditions . . . 3-17
Program Store Compare Exceptional Condition . . . 3-17

Invalid Instruction Address Test Exceptional Condition . . . 3-18
Specification Detection . . . 3-18
Invalid Address Detection . . . 3-19
Fetch Protection Detection . . . 3-21
Invalid Instruction Address Microprogram . . . 3-21
Q-Register Refill Exceptional Condition . . . 3-22
Two-Cycle RR I-Fetch . . . 3-23
Forced-Cycle RX I-Fetch . . . 3-24
Two-Cycle RS and SI I-Fetch . . . 3-24

SECTION 2 FIXED-POINT INSTRUCTIONS . . . 3-25
Load . . . 3-25
Load, LR (18) . . . 3-25
Load, L (58) . . . 3-25
Load Halfword, LH (48) . . . 3-26
Load and Test, LTR (12) . . . 3-27
Load Complement, LCR (13) . . . 3-27
Load Positive, LPR (10) . . . 3-28
Load Negative, LNR (11) . . . 3-28
Load Multiple, LM (98) . . . 3-29
Add-Type Instructions . . . 3-30
Add, AR (1A) . . . 3-31
Add, A (5A) . . . 3-31
Add Halfword, AH (4A) . . . 3-31
Add Logical, ALR (1E) . . . 3-32
Add Logical, AL (5E) . . . 3-33
Subtract, SR (1B) . . . 3-33
Subtract, S (5B) . . . 3-34
Subtract Halfword, SH (4B) . . . 3-34
Subtract Logical, SLR (1F) . . . 3-35
Subtract Logical, SL (5F) . . . 3-35
Compare, CR (19) . . . 3-36
Compare, C (59) . . . 3-36
Compare Halfword, CH (49) . . . 3-37
Multiply . . . 3-37
Multiply, MR (1C) . . . 3-38
Multiply, M (5C) . . . 3-42
Multiply Halfword, MH (4C) . . . 3-42
Divide . . . 3-42
Divide, DR (1D) . . . 3-43
General Discussion . . . 3-44
Detailed Discussion . . . 3-45
Divide, D (5D) . . . 3-47
Convert . . . 3-47
Convert to Binary, CVB (4F) . . . 3-51
Convert to Decimal, CVD (4E) . . . 3-51
Store . . . 3-53
Store, ST (50) . . . 3-53
Store Halfword, STH (40) . . . 3-54
Store Multiple, STM (90) . . . 3-55
Shift . . . 3-56
Shift Left Single, SLA (8B) . . . 3-56
Shift Left Double, SLDA (8F) . . . 3-58
Shift Right Single, SRA (8A) . . . 3-60
Shift Right Double, SRDA (8E) . . . 3-61

SECTION 3 FLOATING-POINT INSTRUCTIONS . . . 3-63
Exponent Overflow and Underflow . . . 3-63
Zero Results . . . 3-63
Conditions at Start of Execution . . . 3-64
Load . . . 3-64
Load, LER (38) – RR Short Operands . . . 3-64
Load, LE (78) – RX Short Operands . . . 3-64
Load, LDR (28) – RR Long Operands . . . 3-65
Load, LD (68) – RX Long Operands . . . 3-65
Load and Test, LTER (32) – RR Short Operands . . . 3-66
Load and Test, LTDR (22) – RR Long Operands . . . 3-66
Load Complement, LCER (33) – RR Short Operands . . . 3-67
Load Complement, LCDR (23) – RR Long Operands . . . 3-67
Load Positive, LPER (30) – RR Short Operands . . . 3-68
Load Positive, LPDR (20) – RR Long Operands . . . 3-68
Load Negative, LNER (31) – RR Short Operands . . . 3-68
Load Negative, LNDR (21) – RR Long Operands . . . 3-69
Add, Subtract, and Compare . . . 3-69
Add Normalized, AER (3A) – RR Short Operands . . . 3-71
Add Normalized, AE (7A) – RX Short Operands . . . 3-75
Add Normalized, ADR (2A) – RR Long Operands . . . 3-76
Add Normalized, AD (6A) – RX Long Operands . . . 3-77
Add Unnormalized, AUR (3E) – RR Short Operands . . . 3-78
Add Unnormalized, AU (7E) – RX Short Operands . . . 3-78
Add Unnormalized, AWR (2E) – RR Long Operands . . . 3-79
Add Unnormalized, AW (6E) – RX Long Operands . . . 3-79
Subtract Normalized, SER (3B) – RR Short Operands . . . 3-80
Subtract Normalized, SE (7B) – RX Short Operands . . . 3-81
Subtract Normalized, SDR (2B) – RR Long Operands . . . 3-81
Subtract Normalized, SD (6B) – RX Long Operands . . . 3-82
Subtract Unnormalized, SUR (3F) – RR Short Operands . . . 3-82
Subtract Unnormalized, SU (7F) – RX Short Operands . . . 3-83
Subtract Unnormalized, SWR (2F) – RR Long Operands . . . 3-83
Subtract Unnormalized, SW (6F) – RX Long Operand . . . 3-84
Compare, CER (39) – RR Short Operands . . . 3-84
Compare, CE (79) – RX Short Operands . . . 3-85
Compare, CDR (29) – RR Long Operands . . . 3-85
Compare, CD (69) – RX Long Operands . . . 3-86
Halve . . . 3-86
Halve, HER (34) – RR Short Operands . . . 3-86
Halve, HDR (24) – RR Long Operands . . . 3-87
Multiply . . . 3-87
Data Flow and Algorithm . . . 3-88
Multiply, MER (3C) – RR Short Operands . . . 3-91
Multiply, ME (7C) – RX Short Operands . . . 3-92
Multiply, MDR (2C) – RR Long Operands . . . 3-93
Multiply, MD (6C) – RX Long Operands . . . 3-93
Divide . . . 3-94
Characteristic Computation . . . 3-95
Normalization . . . 3-96
Fraction Division . . . 3-96
Data Flow and Algorithm . . . 3-98
Divide, DER (3D) – RR Short Operands . . . 3-100
Divide, DE (7D) – RX Short Operands . . . 3-101
Divide, DDR (2D) – RR Long Operands . . . 3-102
Divide, DD (6D) – RX Long Operands . . . 3-103
Store . . . 3-104
Store, STE (70) – RX Short Operands . . . 3-104
Store, STD (60) – RX Long Operands . . . 3-105

SECTION 4 DECIMAL INSTRUCTIONS . . . 3-106
Instruction Handling . . . 3-106
Word Overlap Condition . . . 3-107
General Initialization Sequence . . . 3-109
Add, Subtract, and Compare . . . 3-109
Add, AP (FA) and Subtract, SP (FB) . . . 3-109
GIS for Add and Subtract . . . 3-110

True Add Sequence . . . 3-110
Complement Add Sequence . . . 3-113
Compare, CP (F9) . . . 3-114
Zero and Add, ZAP (F8) . . . 3-115
Multiply, MP (FC) . . . 3-116
General Description . . . 3-118
Detailed Description . . . 3-124
Divide, DP (FD) . . . 3-125
General Description . . . 3-128
Detailed Description . . . 3-134
Pack, PACK (F2) . . . 3-137
Instruction Execution, Not Word Overlap . . . 3-138
Instruction Execution, Word Overlap . . . 3-139
Unpack, UNPK (F3) . . . 3-139
Instruction Execution, Not Word Overlap . . . 3-140
Instruction Execution, Word Overlap . . . 3-141
Move With Offset, MVO (F1) . . . 3-141
Instruction Execution, Not Word Overlap . . . 3-142
Instruction Execution, Word Overlap . . . 3-142

SECTION 5 LOGICAL INSTRUCTIONS . . . 3-144
General Initialization Sequence . . . 3-144
Move . . . 3-144
Move, MVI (92) . . . 3-144
Move, MVC (D2) . . . 3-144
Move Numerics, MVN (D1) . . . 3-145
Move Zones, MVZ (D3) . . . 3-146
Compare . . . 3-147
Compare Logical, CLR (15) . . . 3-147
Compare Logical, CL (55) . . . 3-147
Compare Logical, CLI (95) . . . 3-147
Compare Logical, CLC (D5) . . . 3-148
AND . . . 3-148
AND, NR (14) . . . 3-149
AND, N (54) . . . 3-149
AND, NI (94) . . . 3-150
AND, NC (D4) . . . 3-150
OR . . . 3-150
OR, OR (16) . . . 3-151
OR, O (56) . . . 3-151
OR, OI (96) . . . 3-151
OR, OC (D6) . . . 3-152
Exclusive-OR . . . 3-152
Exclusive-OR, XR (17) . . . 3-152
Exclusive-OR, X (57) . . . 3-153
Exclusive-OR, XI (97) . . . 3-153
Exclusive-OR, XC (D7) . . . 3-153
Test Under Mask, TM (91) . . . 3-154
Insert Character, IC (43) . . . 3-154
Store Character, STC (42) . . . 3-154
Load Address, LA (41) . . . 3-155
Translate, TR (DC) . . . 3-155
Translate and Test, TRT (DD) . . . 3-156
Edit and Edit and Mark, ED and EDMK (DE and DF) . . . 3-158
Introduction to Edit Operation . . . 3-158
Introduction to Edit and Mark Operation . . . 3-160
General Data Handling . . . 3-160
Microprogram Description . . . 3-161
First Cycle . . . 3-161
Second Cycle . . . 3-161
Exit Conditions . . . 3-161
Shift . . . 3-162
Shift Left Single, SLL (89) . . . 3-162
Shift Left Double, SLDL (8D) . . . 3-162
Shift Right Single, SRL (88) . . . 3-162
Shift Right Double, SRDL (8C) . . . 3-162

SECTION 6 BRANCHING INSTRUCTIONS . . . 3-164
Branch on Condition, BCR (07) . . . 3-164
Successful Branch . . . 3-164
Unsuccessful Branch . . . 3-165
Branch on Condition, BC (47) . . . 3-165
Branch and Link, BALR (05) . . . 3-166
Unsuccessful Branch . . . 3-166
Successful Branch . . . 3-167
Branch and Link, BAL (45) . . . 3-168
Branch on Count, BCTR (06) . . . 3-169
Successful Branch . . . 3-170
Unsuccessful Branch . . . 3-170
Branch on Count, BCT (46) . . . 3-170
Branch on Index High, BXH (86) . . . 3-170
Branch on Index Low or Equal, BXLE (87) . . . 3-172
Execute, EX (44) . . . 3-173

SECTION 7 STATUS SWITCHING INSTRUCTIONS . . . 3-176
Load PSW, LPSW (82) . . . 3-176
Set Program Mask, SPM (04) . . . 3-177
Set System Mask, SSM (80) . . . 3-177
Supervisor Call, SVC (0A) . . . 3-177
Set Storage Key, SSK (08) . . . 3-178
Insert Storage Key, ISK (09) . . . 3-179
Write Direct, WRD (84) . . . 3-180
Read Direct, RDD (85) . . . 3-181
Diagnose (83) . . . 3-182
Test and Set, TS (93) . . . 3-183

SECTION 8 INPUT/OUTPUT INSTRUCTIONS . . . 3-184
Start I/O, SIO (9C) . . . 3-184
Test I/O, TIO (9D) . . . 3-186
Halt I/O, HIO (9E) . . . 3-186
Test Channel, TCH (9F) . . . 3-187

Chapter 4 Features . . . 4-1

SECTION 1 FEATURE INDEX . . . 4-1

SECTION 2 MULTIPROCESSING FEATURES . . . 4-2
Multiprocessing System/360 Model 65 . . . 4-2
Main Storage . . . 4-2
Storage Allocation . . . 4-2
Floating Addressing . . . 4-2
Direct Address Relocation . . . 4-3
Input/Output . . . 4-4
Processing Units . . . 4-4
Multisystem Mode . . . 4-4
Model 65 Mode . . . 4-5
Partition Mode . . . 4-5
Multisystem Signals . . . 4-5
Summary of Multiprocessing System Advantages . . . 4-6
Functional Units . . . 4-6
Configuration Control Panel . . . 4-7
Storage Allocation Control . . . 4-7
Floating Address Control . . . 4-7
Direct Address Relocation Control . . . 4-8
Multiprocessing System Mode Control . . . 4-8
I/O Allocation Control . . . 4-8
BCU Modifications . . . 4-8
Storage Address Decoding with Prefixing Disabled . . . 4-8

Storage Address Decoding with Prefixing Enabled . . . 4-8
Invalid Storage Address . . . 4-9
BCU-Storage Operations . . . 4-9
Multisystem Timer . . . 4-9
Operation . . . 4-9
System Hang Timing . . . 4-10
External System Reset Timing . . . 4-10
Multisystem Operations . . . 4-10
Set System Mask Instruction (Multisystem Mode) . . . 4-10
Write Direct Instruction (Not Model 65 Mode) . . . 4-10
Read Direct Instruction (Not Model 65 Mode) . . . 4-11
Malfunction Alert . . . 4-11
Gated Load . . . 4-11
System Call . . . 4-11
Log I/O Interrupt . . . 4-12
External System Reset . . . 4-12
External Start . . . 4-12
Power Distribution and Control . . . 4-13
Console Controls and Maintenance Features . . . 4-13
Configuration Control Panel . . . 4-13
Storage Allocation Switches . . . 4-13
Floating Address Switches . . . 4-13
PREFIX Switches . . . 4-13
CPU Mode Switches . . . 4-13
I/O Allocation Switches . . . 4-14
VALID ADDRESS Indicators . . . 4-14
System Control Panel Modifications, Multisystem Feature . . . 4-14
EMERGENCY PULL Switch . . . 4-14
POWER ON Pushbutton . . . 4-14
Marginal Voltage Control . . . 4-14
DISABLE INTERVAL TIMER Switch . . . 4-14
DISABLE DIRECT CONTROL Switch . . . 4-14
Storage Switches . . . 4-14
Indicators . . . 4-14
System Control Panel Modifications, Additional Storage Attachment Features . . . 4-15
Marginal Voltage Control . . . 4-15
POWER CHECK Indicators . . . 4-15
STORAGE INDICATE SWITCH and Indicators . . . 4-15
Logout and Scan In . . . 4-15

Chapter 5 Power Distribution and Control . . . 5-1
AC Power Distribution . . . 5-1
60-Hz Units . . . 5-1
50-Hz Units . . . 5-1
Converter/Inverter . . . 5-1
DC Power Distribution . . . 5-4
High-Frequency Regulator Modules . . . 5-4
Marginal Adjustments . . . 5-6
Power-On Sequence . . . 5-6
Power-Off Sequence . . . 5-8
Normal Power-Off . . . 5-8
Emergency Power-Off . . . 5-8
Automatic Power-Off . . . 5-9
Overcurrent Protection . . . 5-9
Overvoltage Protection . . . 5-9
Positive Regulators, Converted Units . . . 5-10
Negative Regulators, Converted Units . . . 5-10
Undervoltage Protection . . . 5-11
Thermal Protection . . . 5-11
Indicators . . . 5-11
System Power-On Indicator . . . 5-11
Power Check Indicators . . . 5-11
1052 Printer-Keyboard Power . . . 5-12
Audible Alarm . . . 5-12
Dual 1052 Power Interface . . . 5-12
Direct Control Power . . . 5-12
Usage Meters and Key Switch . . . 5-13

Chapter 6 Console Controls and Maintenance Features . . . 6-1

SECTION 1 CONSOLE CONTROLS . . . 6-1
System Control Panel . . . 6-1
Manual Controls . . . 6-4
Stop Loop . . . 6-5
Power-On Reset . . . 6-5
SYSTEM RESET Pushbutton . . . 6-6
CHECK RESET Pushbutton . . . 6-6
STOP Pushbutton . . . 6-6
LOAD Pushbutton (IPL) . . . 6-6
DATA Switches . . . 6-9
ADDRESS Switches . . . 6-9
ADDRESS COMPARE STOP Switch . . . 6-9
STORAGE SELECT Switch . . . 6-9
DEFEAT INTERLEAVING Switch . . . 6-12
STORAGE INDICATE Switch . . . 6-12
STOP ON STORAGE CHECK Switch . . . 6-12
SET IC Pushbutton . . . 6-12
RATE Switch . . . 6-13
PROCESS Position . . . 6-13
INSN STEP Position . . . 6-13
SINGLE CYCLE Position . . . 6-13
SINGLE CYCLE STORAGE INHIBIT Position . . . 6-13
REPEAT INSN Switch . . . 6-13
Repeat Single Instruction . . . 6-14
Repeat Multiple Instructions . . . 6-14
STORE Pushbutton . . . 6-14
DISPLAY Pushbutton . . . 6-15
START Pushbutton . . . 6-15
ROS TRANSFER Pushbutton . . . 6-16
Storage-Ripple Microprogram . . . 6-16
Storage-Ripple-Store Function . . . 6-16
Storage-Ripple-Display Function . . . 6-16
REPEAT ROS ADDRESS Switch . . . 6-16
PSW RESTART Pushbutton and Wait State . . . 6-16
DISABLE DIRECT CONTROL Switch . . . 6-17
DISABLE INTERVAL TIMER Switch . . . 6-17
INTERRUPT Pushbutton . . . 6-17
CPU CHECK Switch . . . 6-17
PULSE MODE Switch . . . 6-17
PROC Position . . . 6-18
TIME Position . . . 6-18
COUNT Position . . . 6-18
LOG OUT Pushbutton . . . 6-18
TEST MODE, ROS/PROC/FLT Switch . . . 6-18
TEST MODE, REPEAT Switch . . . 6-19
RESTART FLT I/O Pushbutton . . . 6-19
CE Key Switch and Usage Meters . . . 6-19
FREQUENCY ALTERATION Switch . . . 6-19
Indicators . . . 6-19
CE Panel . . . 6-21

SECTION 2 MAINTENANCE FEATURES . . . 6-22
Diagnose Instruction and MCW's . . . 6-22
Diagnose Instruction MCW for CPU . . . 6-23
Diagnose Instruction MCW for Channel . . . 6-23
ROS Test MCW . . . 6-24
FLT MCW . . . 6-24
Logout, ROS Tests, and FLT's . . . 6-24
Introduction . . . 6-24

Logout 6-25
ROS Tests 6-26
FLT's 6-26
FLT Tapes 6-27
Tape Generation 6-27
FLT Hardcore Tests 6-28
Zero-Cycle Tests 6-28
One-Cycle Tests 6-28
FLT Format 6-29
Scan Logic Functional Units 6-29
Scan Timing 6-29
Scan Clock 6-30
FLT Clock 6-30
Scan Counter Latches and Decrementer 6-31
Input and Output 6-31
Scan Counter Decrementer 6-31
Address Sequencer 6-31
Address Sequencer Decoder 6-33
Storage Address Generator 6-34
Check Counter 6-34
Input and Output 6-35
Check Counter Decrementing 6-36
FLT Counter 6-36
Input 6-36
FLT Counter Decrementing 6-37
Cycle Counter 6-37
ROS Test Sequencer 6-38
Scan-Out Bus 6-38
Logout Controls 6-38
Scan Out S and T 6-39
Scan Stop-CPU-Clock Logic 6-39
Control Triggers 6-39
Scan Mode Control of ROS 6-40
Scan/Channel Interface 6-40
Operational Analysis 6-40
Logout 6-40
Hardware-Controlled Sequence 6-41
ROS-Controlled Sequence 6-42
ROS Tests 6-43
ROS Test Tape 6-43
ROS Test Setup 6-44
IPL 1 6-44
Loader 6-45
Hardcore Tests and IPL 2 6-45
Test 1, Record 3 6-46
IPL 2 6-47
Test 2, Record 5 6-47
Record 6 6-47
Test 1, Record 7 6-48
Test 1, Record 8 6-48
Test 2, Record 8 6-48
Test 3, Record 8 6-48
Summary of Hardcore Tests 6-48
ROS Bit Tests 6-49
ROS Test State 7 6-49
ROS Test State 6 6-49
ROS Test State 5 6-49
ROS Test State 4 6-49
ROS Test State 3 6-50
ROS Test State 2 6-50
ROS Test State 1 6-50
ROS Test State 0 6-50
Fault Locating Tests 6-51
FLT Tape 6-51
FLT Format 6-52
FLT Test Setup 6-53
IPL 1 6-53
Loader 6-53
Transmission Checks During FLT Read In 6-54
Hardcore Tests 6-54
Zero-Cycle and One-Cycle Tests 6-54
Scan In 6-55
Test Cycles 6-56
Scan Out 6-56
Result Comparison 6-56
Terminate or Continue 6-56
TN/ATN Comparison 6-57
Ripple Tests 6-57
Diagnostic Programs 6-57
Marginal Checking 6-58

Appendix A A-1

Appendix B B-1

Appendix C C-1

Index X-1

Illustrations

(THIS VOLUME)

Legend xv
Frontispiece System/360 Model I65 xvi
1-1 Basic System Configuration 1-1
1-2 System/360 Model 65 Layout 1-2
1-3 Functional Structure of a Simplex System 1-8
1-4 Channel-to-Channel Adapter as Multisystem Connector 1-9
1-5 Transmission Control Unit as Multisystem Connector 1-9
1-6 2-Channel Switch Feature as Multisystem Connector 1-9
1-7 Device Switching Unit as Multisystem Connector 1-10
1-8 Shared LCS Feature as Multisystem Connector 1-10
1-9 Direct Control Feature as Multisystem Connector 1-10
1-10 Multiprocessing System/360 Model 65 . . . 1-11
1-11 Instruction Formats 1-14
1-12 Main Storage Integral Boundaries 1-15
1-13 Examples of Control Program Functions . . 1-22
1-14 Action Taken When Single Interruption Occurs 1-30

1-15 Example of Need for Interruption Masking . 1-31
1-16 Data Transfer Scheme 1-40
1-17 ROS Addressing and Branching 1-42
1-18 ROS Addressing Block Diagram 1-44
1-19 ROS Data Flow 1-45
1-20 ROS Timing 1-48
1-21 ROS Control of CPU Operations 1-49
1-22 Status Information Contained in PSW Register 1-51
1-23 Basic BCU Interface 1-52
1-24 BCU Clock Logic 1-53
1-25 BCU Priority Logic 1-53
1-26 Start Storage Sequence Logic 1-54
1-27 Storage Selection 1-54
1-28 Q-Register Halfword Outgating per IC(21,22) 1-56
1-29 Instruction Addressing 1-57
1-30 Operand Data Byte Selection per IC(21–23) 1-58
1-31 Basic Instruction Path 1-59
1-32 Path Through Q-, R-, and E-Registers of Op-Code Halfword 1-61
1-33 Basic Scheme for Operand Prefetching . . . 1-62
1-34 Q-Register Refill Addressing Scheme 1-65
1-35 Decimal Format Serial-Adder Data Flow . . . 1-67
1-36 Parallel Adder Logical Functions 1-68
1-37 Parallel Adder Group/Section Breakdown . . 1-69
2-1 Trigger and Latch Data Relationship 2-2
2-2 Typical Clock Signals 2-3
2-3 Clock Signal Development and Distribution 2-5
2-4 Basic 4 X 4 CROS Matrix 2-7
2-5 Bit Plate 2-8
2-6 Sense Lines 2-9
2-7 Sense Line Layout 2-10
2-8 Bit Capacitors 2-11
2-9 CROS Plane Pressure Mounting Assembly . . 2-12
2-10 Control Field A Decoder 2-15
2-11 Detailed ROS Timing 2-16
2-12 ROSPARA and ROSPARB Alternator . . . 2-17
2-13 ROS Parity Checking 2-19
2-14 Primary BCU Interface Signals 2-21
2-15 Basic Organization of HSS Unit 2-23
2-16 Basic BCU Scheme for Processing Storage Requests 2-24
2-17 Basic Request Timing 2-25
2-18 Basic BCU Operational Sequence 2-27
2-19 Typical Timing for CPU Fetch Request to LCS 2-31
2-20 Gating of Storage Address from CPU & Channels to Address Decoders in BCU . . . 2-32
2-21 Selection of Correct Storage Unit 2-33
2-22 Basic SAB Decoding Circuits in BCU and HSS Unit 2-35
2-23 SAB Parity Conversion Logic 2-37
2-24 BCU Reset for LCS Operation 2-39
2-25 Typical CPU Clock Stopping Sequence . . . 2-40
2-26 Q-Register Data Flow 2-45
2-27 Q-Register Halfword Transfer per IC(21,22) 2-46
2-28 R-Register Data Flow 2-47
2-29 E-Register Data Flow 2-48
2-30 Instruction Counter Data Flow 2-49
2-31 D-Register Data Flow 2-51
2-32 AB Register Data Flow 2-53
2-33 ST Register Data Flow 2-55
2-34 PSW Input to S(16–31) 2-56
2-35 F-Register Data Flow 2-58
2-36 G-Register Data Flow 2-59
2-37 PSW Register Data Flow 2-60
2-38 PSW Register(0,6) Logic 2-60
2-39 MCW Register Data Flow 2-60
2-40 Local Storage Data Flow 2-62
2-41 Serial Adder Data Flow 2-65
2-42 True-Complement Data Entry 2-66
2-43 Serial Adder (Simplified) 2-66
2-44 Half-Sum and Full-Sum Logic 2-67
2-45 Carry Lookahead, Block Diagram 2-67
2-46 Serial Adder Gating Controls 2-69
2-47 Serial Adder Parity Predict Logic 2-71
2-48 Half-Sum and Full-Sum Error Logic 2-73
2-49 Parallel Adder Data Flow 2-74
2-50 Parallel Adder Function Breakdown 2-75
2-51 Parallel Adder Input Buses 2-76
2-52 Bit Position Block Diagram 2-76
2-53 Parallel Adder Carry Lookahead Data Flow 2-79
2-54 Actual and Predicted Carry Origin for PA(44) 2-81
2-55 Full-Sum Development Logic 2-82
2-56 Parallel Adder Logic Function Sequence . . 2-83
2-57 Convert-to-Decimal Data Flow to Parallel Adder 2-85
2-58 Summary of Setting of STAT's 2-87

(VOLUME 2)

3-1 Typical Microprogram Sequence 3-1
3-2 Basic Sequencing for SS Instructions 3-8
3-3 ASC Test for SS Instructions 3-10
3-4 Detection of Invalid Instruction Address . . 3-20
3-5 Detection of Fetch-Protected Instruction Address 3-22
3-6 Fixed-Point Multiply, Example No. 1 (RR Format) 3-40
3-7 Fixed-Point Multiply, Example No. 2 (RR Format) 3-41
3-8 Fixed-Point Divide, Example No. 1 3-48
3-9 Fixed-Point Divide, Example No. 2 3-48
3-10 Convert to Decimal Example 3-52
3-11 Restore and Non-Restore Division 3-97
3-12 Fraction Divide Example 3-98
3-13 Floating-Point Divide Example 3-102
3-14 Operand Specifications for Decimal Multiply Instruction 3-118
3-15 Typical Multiply Add Sequence 3-119
3-16 Typical Multiply Subtract Sequence 3-120
3-17 Data Handling During GIS of Decimal Multiply 3-121
3-18 Data Handling During Multiplier Left-Adjust Sequence 3-122
3-19 Data Flow for Right-4 Shift of ST to AB, Decimal Multiply 3-126
3-20 Operand Specifications for Decimal Divide 3-128
3-21 Example of a Typical Divide Sequence . . . 3-129
3-22 Data Handling During GIS of Decimal Divide 3-130

3-23 Data Handling During Divisor Left-Adjust Sequence 3-131
3-24 Data Handling During Dividend Fetch and Left-Adjust Sequence 3-133
3-25 Simplified Data Flow for AND, OR, and Exclusive-OR Instructions 3-149
3-26 Example of Use of Branch and Link Instruction 3-167
3-27 Storage Protection Key Assignments 3-179
4-1 Direct Address Relocation 4-3
4-2 Duplicate Storage Addressing Detection . . 4-7
5-1 Primary AC Power Distribution, 60-Hertz Units 5-2
5-2 Primary AC Power Distribution, 50-Hertz Units 5-3
5-3 Representative DC Distribution 5-5
5-4 Overcurrent Protection Loop 5-10
6-1 Normal IPL Operaiton 6-7
6-2 Data Switch Gating 6-10
6-3 Address Switch Gating 6-11
6-4 Scan Counter Latches and Decrementer Data Flow 6-32
6-5 Address Sequencer Data Flow 6-33
6-6 Address Sequencer Decoder 6-34
6-7 Check Counter Data Flow 6-35
6-8 FLT Counter Data Flow 6-36
6-9 Cycle Counter Data Flow 6-37
6-10 ROS Test Sequencer Data Flow 6-38
C-1 Four-Block Tree and FLT Pattern Generated C-1

Tables

(THIS VOLUME)

1-1 Permanent Main Storage Assignments . . . 1-4
1-2 Characteristic Notation 1-18
1-3 Program States 1-24
1-4 PSW Interruption Mask Bit Designation . . . 1-25
1-5 Interruptions 1-28
1-6 ROS Word Breakdown 1-41
1-7 Control Field V (Bits 97–99); E, Q Outgates to Parallel Adder B-Bus 1-42
1-8 Fixed-Point Instructions 1-71
1-9 Floating-Point Instructions 1-77
1-10 Decimal Instructions 1-83
1-11 Logical Instructions 1-88
1-12 Branching Instructions 1-93
1-13 Status Switching Instructions 1-96
1-14 I/O Instructions 1-98
2-1 Decimal Correct for Erroneous Numeric Characters 2-68
2-2 Control Triggers 2-89

(VOLUME 2)

3-1 Q-Register Refill Exceptional Conditions . . 3-23
3-2 Value of Multiple Determined by Multiple Selection Bits (Fixed-Point) 3-39
3-3 Divide Multiple Values, Fixed-Point 3-46
3-4 Conversion to Decimal (Excess-6) 3-51
3-5 Excess-6 Conversion, B(60–63) 3-53
3-6 Operand Bits Transferred, STH Instruction . 3-55
3-7 Left Shift Combinations 3-57
3-8 Right Shift Combinations 3-61
3-9 Examples of Branching on Characteristic Difference 3-73
3-10 Multiplier Bits Selected, Floating-Point Multiply 3-89
3-11 Value of Multiple Determined by Multiple Selection Bits (Floating-Point) 3-90
3-12 Condition Code Setting Per Hardware Conditions, Decimal Instructions 3-113
4-1 Feature Index 4-1
4-2 Floating Address Intervals 4-3
5-1 High-Frequency Regulator Modules 5-4
6-1 Logout Format 6-25
6-2 FLT Format 6-52
C-1 SCOPEX C-2

Abbreviations

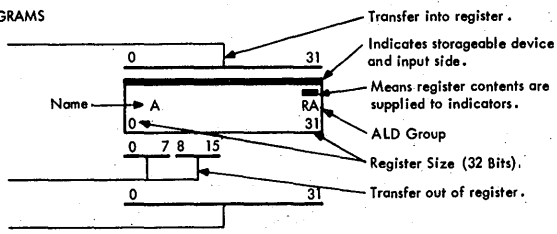
ABC	AB register byte counter
ac	alternating current
adr	address, addressed, addressing
ALD	automated logic diagram
amp	ampere
ASC	address store compare
ATN	alternate test number
BCD	binary-coded decimal
BCU	bus control unit
C	capacitor
CAW	channel address word
CB	circuit breaker
CC	condition code
CCW	channel command word
CE	customer engineer
charistic	characteristic
CLD	control automation system logic diagram
CPU	central processing unit
CR	diode

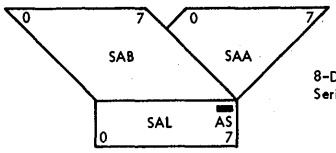
CROS	capacitive read-only storage
CSW	channel status word
CT	conditional terminate
dc	direct current
dec div	decimal divide
dec ovflo	decimal overflow
DX	first byte in a series of destination bytes
DX + 1	second byte in a series of destination bytes
DX + 2	third byte in a series of destination bytes
end op	end operation
EPO	emergency power off
ERSLT	expected result
exp ovflo	exponent overflow
exp unflo	exponent underflow
F	fuse
FEMDM	Field Engineering Maintenance Diagrams Manual
FEMI	Field Engineering Manual of Instruction
FEMM	Field Engineering Maintenance Manual
FETOM	Field Engineering Theory of Operation Manual
fix-pt div	fixed-point divide
fix-pt ovflo	fixed-point overflow
FLT	fault locating test
flt-pt div	floating-point divide
FLUT	Fault Locating Utility program
FPR	floating-point register
fract	fraction
GIS	general initialization sequence
GPR	general-purpose register
hex	hexadecimal
HSS	high-speed storage
Hz	Hertz
IC	instruction counter
I-Fetch	instruction fetching
ILC	instruction length code
I/O	input/output
IPL	initial program load
K	kilo
K	relay
kHz	kilohertz
LAL	local storage address latches
LAR	local storage address register
LCS	large capacity storage
LS	local storage
LSWR	local storage working register
MAR	memory address register
max	maximum
MCW	maintenance control word
mHz	megahertz
MMSC	maintenance mode stop clock
MPR	multiplier
ms	millisecond
multisys	multisystem
no op	no operation
ns	nanosecond

op code	operation code
oper	operation
opr	operand
P	parity
PAA	parallel adder A-side
PAB	parallel adder B-side
PAL	parallel adder latch
pf	picofarad
PK	power contactor
PP	partial product
PQ	partial quotient
priv oper	privileged operation
prot	protection
PS	power supply
PSW	program status word
R	resistor
ROS	read-only storage
ROSAR	read-only storage address register
ROSBR	read-only storage backup register
ROSDR	read-only storage data register
ROSPARA	read-only storage previous address register A
ROSPARB	read-only storage previous address register B
SAA	serial adder A-side
SAB	serial adder B-side
SAB	storage address bus
SAL	serial adder latch
SAR	storage address register
SBA	serial adder bus A
SBB	serial adder bus B
SCOPEX	scoping index
SCR	silicon-controlled rectifier
SDBI	storage data bus in
SDBO	storage data bus out
signif	significance
SLT	solid logic technology
SMS	standard modular system
SOROS	scan out read-only storage
spec	specification
SRL	Systems Reference Library
STAT	status trigger
STC	ST register byte counter
stg	storage
SW BD	switch board
sync	synchronizing
T	transformer
T(DX)	table byte specified by DX
T(DX + 1)	table byte specified by DX + 1
TIC	transfer in channel
TN	test number
uf	microfarad
usec	microsecond
UT	unconditional terminate
V	volt
VFL	variable-field length
$\geq$	greater than or equal to
$\geq$	greater than or equal to
$\leq$	less than or equal to
$\leq$	less than or equal to
=	equal to
$\neq$	not equal to

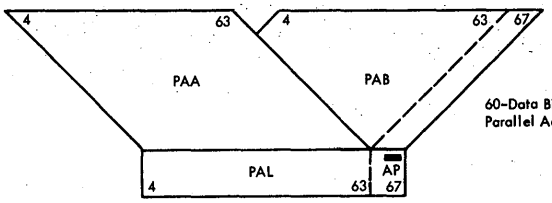
Legend

LOGIC DIAGRAMS

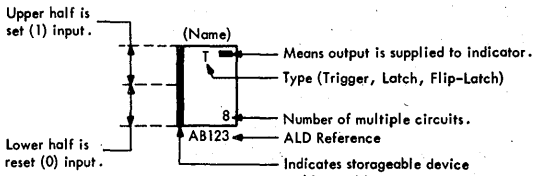


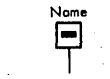


8-Data Bit (Plus Parity) Serial Adder



60-Data Bit (Plus Parity) Parallel Adder

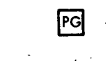




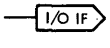
Indicator

Roller Switch Indicator

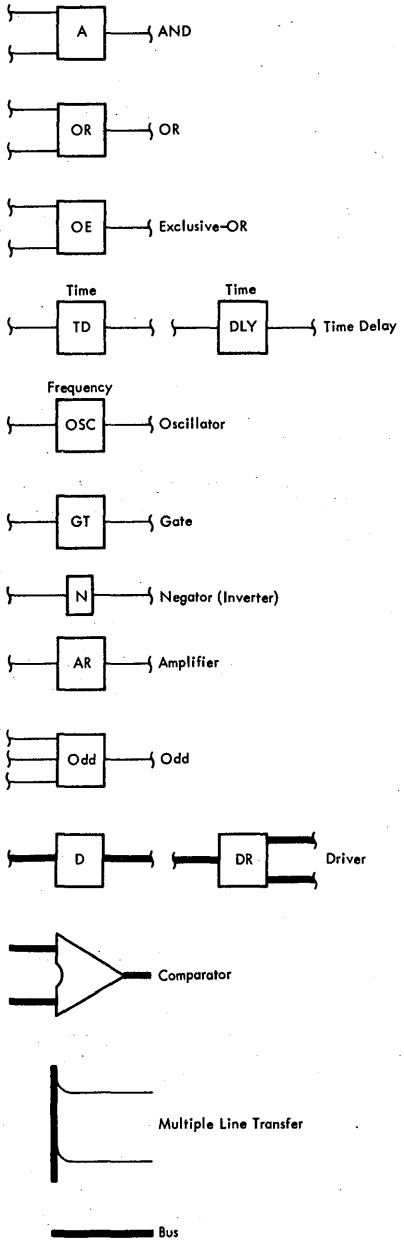
Parity Check Logic



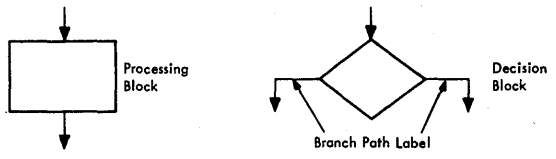
Parity Generate Logic



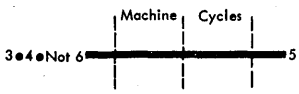
Interface
Denotes interface between two units.



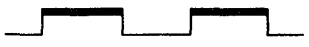
FLOWCHARTS



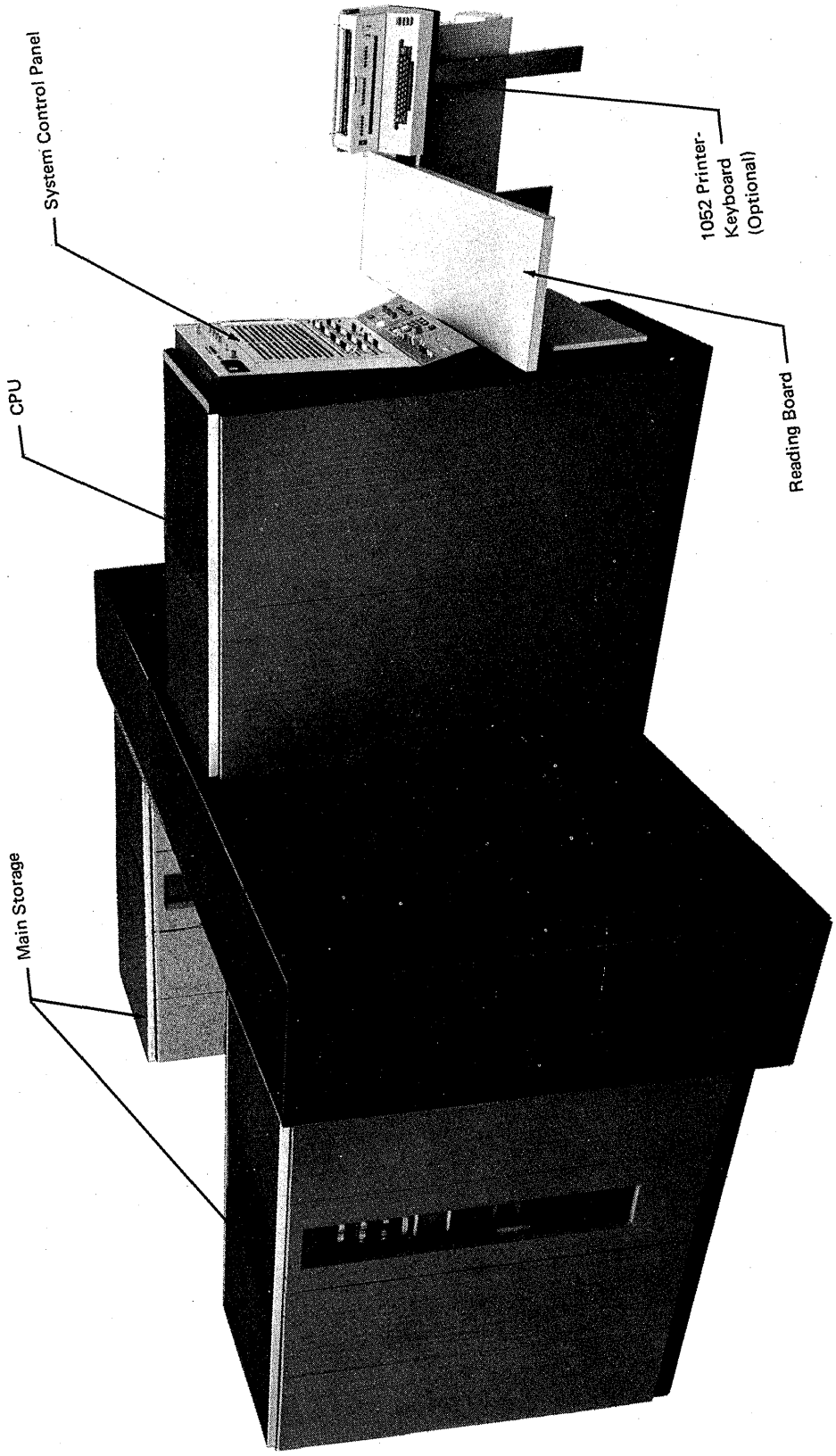
TIMING CHARTS



Heavy bar indicates active state. Number(s) at beginning and end of the bar identify the signal(s) (also on the same chart) that activate and deactivate this line. "Not" preceding a number means that the deactive signal conditions this line.



Waveshape
Heavy bar indicates active state.



System/360 Model 165

www.IBM360.info

Chapter 1. Introduction

This chapter introduces the 2065 Processing Unit (Central Processing Unit or CPU) of the System/360 Model 65. It is divided into four sections:

Section 1, System Description, describes the basic system in terms of main storage, CPU, and input/output (I/O), the features available, and the data flow. Also included is a discussion of multisystem configurations and the resources available to achieve a multisystem.

Section 2, System Coding, discusses the hexadecimal number system, character codes, instruction formats, and operands used in the System/360 Model 65.

Section 3, Program Execution and Control, discusses the role of the supervisor program, the eight program states, the make-up of the program status word (PSW), the purpose and implementation of interruptions and exceptional conditions, and the initiation and control of I/O operations.

Section 4, CPU Description, discusses the functional units of the CPU, Instruction Fetching (I-Fetch) and instruction execution, the Universal instruction set by instruction class, and power considerations.

Section 1. System Description

BASIC SYSTEM

A basic System/360 Model 65 is shown in Figure 1-1. This system consists of a 2065 CPU, one 2365 Processor Storage Model 1 or up to four Model 2's, and up to six 2860 Selector Channels and one 2870 Multiplexer channel. (The number and configuration of I/O control units and devices is flexible and is therefore not shown.) Five models are available:

1. Model G65. Uses one 2365 Processor Storage, Model 1 (131,072 bytes).
2. Model H65. Uses one 2365 Processor Storage, Model 2 (262,144 bytes).
3. Model I65. Uses two 2365 Processor Storage, Model 2's (524,288 bytes).
4. Model IH65. Uses three 2365 Processor Storage, Model 2's (786,432 bytes).
5. Model J65. Uses four 2365 Processor Storage, Model 2's (1,048,576 bytes).

As shown in Figure 1-1, the storage address bus (SAB) and the two storage data buses ['storage data bus in' (SDBI) and 'storage data bus out' (SDBO)] are common to the CPU and I/O channels. The Bus Control Unit (BCU), located in the CPU, monitors and controls the availability of these buses and selects the main storage area to be used. Storage requests (stores and fetches) generated by the CPU and I/O channels enter the BCU, which issues a response signal to the requesting unit having the highest priority. The response signal places main storage at the disposal of the unit for one storage cycle. All units must compete for the next available storage cycle in the same manner. I/O channel requests are given priority over CPU requests, and each channel has an assigned priority.

The 2065 CPU and main storage layout is shown in Figure 1-2. The system control panel is attached to the end of the CPU frame. One or two optional 1052 Printer-Keyboards may be placed adjacent to the CPU reading board to serve as operator consoles.

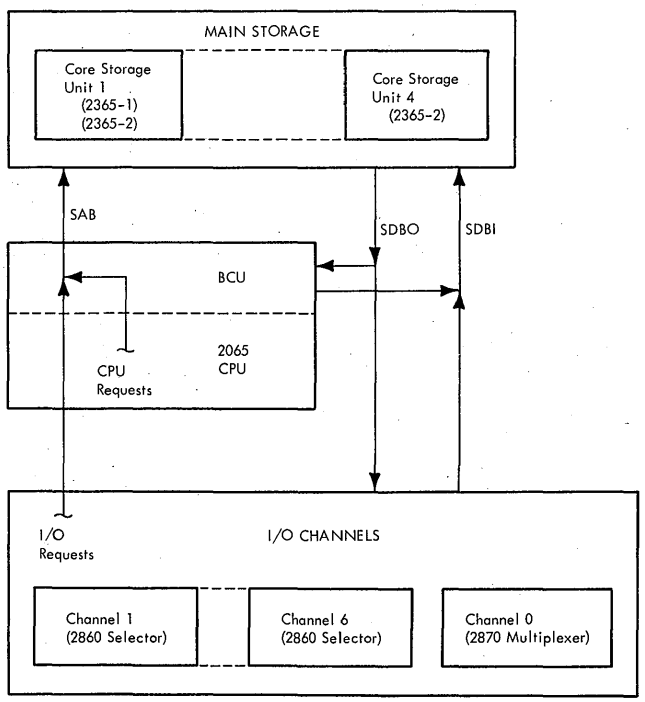


Figure 1-1. Basic System Configuration

FEATURES AVAILABLE

Features available for the 2065 CPU are:

1. Direct Control. Provides two instructions, Read Direct and Write Direct, and six external interruption lines which are independent of I/O channel operations. The instructions provide timing signals and transfer a byte of information between two CPU's or between the CPU and an external device.
2. 2361 Attachment. Provides bulk core storage of from 1,048,576 bytes to 8,388,608 bytes through the attachment of up to four 2361 Core Storage Units. Two models are available: Model 1, which has a storage capacity of 1,048,576 bytes; and Model 2, which has a capacity of 2,097,152 bytes. This feature provides two-way interleaving for pairs of 2361 units.
3. 1052 Adapter. Permits attachment of one 1052 Printer-Keyboard, Model 7, for system console I/O. Also includes a program-controlled alarm. Two 1052 Adapter features can be installed, allowing the attachment of a 1052 on each side of the CPU reading board.
4. Multisystem. Permits two 2065 CPU's to be joined to form a Multiprocessing System/360, Model 65. Both CPU's share a configuration control panel and two to four 2365 Model 13 Processor Storage units (262,144 bytes each). See Additional Storage Attachment feature also.

4A. Additional Storage Attachment (Model J only). Four features, each permitting the attachment of one additional 2365 Model 13 Processor Storage in a Multiprocessing System/360 Model 65.

5. Emergency Power-Off Control. Provides, in effect, a single Emergency Power-Off switch in a "room" or "area".
6. 7070/7074 Compatibility (Models H, I, IH, and J only). Allows the execution of 7070 and 7074 instructions.
7. 7080 Compatibility (Models H, I, IH, and J only). Enables the system to execute 705 and 7080 instructions.
8. 709/7040/7044/7090/7094/7094II Compatibility (Models I, IH, and J only). Enables the system to execute 709, 7040, 7044, 7090, 7094, and 7094II instructions.

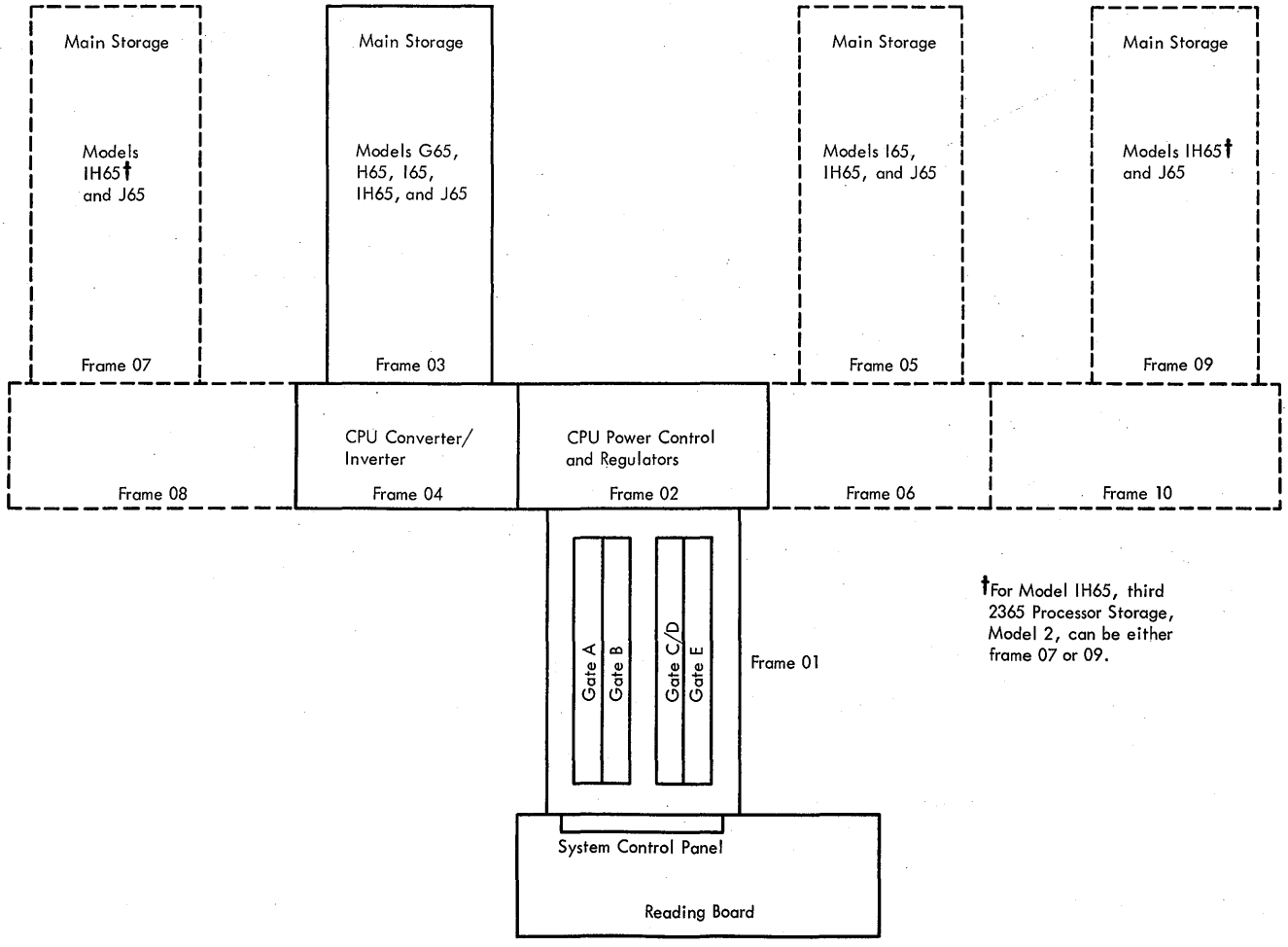


Figure 1-2. System/360 Model 65 Layout

SYSTEM DATA FLOW

The basic data flow in the System/360 Model 65 is shown in Diagram 3-1, FEMDM. The multiplexed SDBO and SDBI are the data paths between main storage and the CPU, and between main storage and the I/O channels. These buses transfer 64 data bits (8 bytes or a doubleword) plus 8 parity bits on each main storage cycle. The BCU determines which unit may access main storage according to a priority scheme. The multiplexed SAB carries 21 address bits plus 3 parity bits from the highest-priority requesting unit to the main storage addressing logic.

When storing data, the eight mark triggers determine which bytes of the data placed on the SDBI are to be stored into the doubleword addressed by SAB. When fetching data, a doubleword of data is transferred, via the SDBO, to the requesting unit from the main storage location addressed by SAB.

The bus-in and bus-out lines between the I/O channels and their attached control units are part of the Standard I/O Interface, and transfer one byte of data at a time.

Each I/O channel has buffer registers which can receive or transmit a doubleword of data between the I/O channel and main storage, and which can receive or transmit a byte of data at a time between the I/O channel and the attached I/O control units.

Parity checking throughout the system is at a byte level.

Main Storage

- Byte, eight bits, is format building block and basic addressable unit of information.
- Halfword is two consecutive bytes.
- Word is four consecutive bytes.
- Doubleword is eight consecutive bytes.
- Byte locations are numbered consecutively, starting with 0.
- Group of bytes is addressed by lowest-numbered (leftmost) byte.
- Data is fetched in doubleword lengths.
- Data is stored on byte basis.
- Store and fetch protection is provided for 2048-byte blocks.

The byte, which consists of eight bits, is the basic building block of all formats and the basic addressable unit of information. A ninth bit, the parity bit, is transmitted with each byte and establishes odd parity for each byte. The parity bit cannot be affected by the program; its only purpose is to cause a machine check interruption when a parity error is detected. References to the size of data fields and registers therefore exclude the associated parity bits. All storage capacities are expressed in number of bytes provided, regardless of the physical word size used.

Bytes may be handled separately or grouped together in fields. A ***halfword*** is a field of two consecutive bytes; a ***word***, a field of four consecutive bytes; a ***doubleword***, a field of eight consecutive bytes. The location of any field of bytes in main storage is specified by the address of its lowest-numbered (leftmost) byte.

Byte locations in main storage are numbered consecutively, starting with 0; each number is considered the address of the corresponding byte. A group of bytes in main storage is addressed by the lowest-numbered (leftmost) byte of the group. The CPU can accommodate binary addresses up to 24 bits long, thus providing addresses for up to 16,777,216 bytes. Addresses 0 through 4095 can be generated without a base address or an index. This property is important when the program status word (PSW) and general register contents must be preserved and restored during program switching. Thus this area has special significance to supervisor programs and contains all permanent storage assignments, such as old PSW's, new PSW's, channel address word, channel status word, and the interval timer value (Table 1-1).

The available storage is normally contiguously addressable, starting at address 0. An addressing program interruption occurs when addressing beyond the maximum available capacity of the installation. Except for a few instructions, the interruption occurs only when attempting to use the data and not before.

Each time a storage unit is accessed by the CPU or I/O channels, information is either stored into or fetched from that storage unit. When a storage request is received via the BCU, core storage performs a read/write storage cycle. If a fetch operation is to be performed ('store' latch reset), the addressed doubleword from the core array is gated to the memory data registers (MDR's), a pair of 32-bit registers which serves as a buffer. The output of the MDR's is gated to the SDBO and to the write circuits of the core array to be regenerated into the addressed doubleword location of main storage. The SDBI is blocked so that data is not entered into main storage.

If a store operation is to be performed ('store' latch set), the SDBO is blocked, and the data on the SDBI is accompanied by one or more mark signals on the mark bus. There is a mark line for each of the eight SDBI bytes. Presence of signals on these lines indicates which bytes are to be replaced in core storage. Thus data can be stored selectively by bytes; a single store operation can replace up to eight bytes in storage at the address designated by the SAB. Although the SDBO is blocked, the addressed doubleword from the core array is gated to the MDR's. There, those bytes that do not have a corresponding mark line active (those bytes to be regenerated) are combined with the bytes on the SDBI that have a corresponding mark line active (indicating new data). The contents of the MDR's are then transferred to the write circuits and placed into the addressed doubleword location of the core array.

Instructions that involve fetching and subsequent storing of data do not necessarily take storage accesses consecutively; it is possible for a channel to take one or more intervening accesses. Only the Test and Set instruction takes a combination fetch and store access without permitting any intervening accesses.

Table 1-1. Permanent Main Storage Assignments

Main Storage Address		Length	Information Stored
Dec	Hex		
0	0	Doubleword	Initial program loading PSW
8	8	Doubleword	Initial program loading Channel Command Word 1 (CCW 1)
16	10	Doubleword	Initial program loading CCW 2
24	18	Doubleword	External interruption, old PSW
32	20	Doubleword	Supervisor call interruption, old PSW
40	28	Doubleword	Program interruption, old PSW
48	30	Doubleword	Machine check interruption, old PSW
56	38	Doubleword	I/O interruption, old PSW
64	40	Doubleword	Channel Status Word (CSW)
72	48	Word	Channel Address Word (CAW)
76	4C	Word	Unassigned
80	50	Word	Timer
84	54	Word	Unassigned
88	58	Doubleword	External interruption, new PSW
96	60	Doubleword	Supervisor call interruption, new PSW
104	68	Doubleword	Program interruption, new PSW
112	70	Doubleword	Machine check interruption, new PSW
120	78	Doubleword	I/O interruption, new PSW
128	80	22 doublewords	Diagnostic log-out area

A storage protection capability is provided to protect the contents of certain areas of main storage from destruction (store protection) or misuse (fetch protection). Locations may be protected against store violations or against store and fetch violations but never against fetch violations only. Protection is achieved by dividing main storage into 2048-byte blocks and assigning a five-bit protection key to each block. The keys for the blocks are contained in a separate small core array in the associated storage unit. The low-order bit of the key in storage designates whether the block is protected against fetches. If this bit is a 0, no fetch protection is specified and only store protection is provided. If this bit is a 1, protection applies to both storing and fetching. The key may be assigned by the Set Storage Key instruction and inspected by the Insert Storage Key instruction.

When protection applies to a storage reference, the key in storage is compared with the protection key supplied by the CPU or channel. Access is permitted only when the four high-order bits of the key in storage match the supplied protection key or when the supplied protection key is zero.

2365 Processor Storage

Two models of the 2365 Processor Storage unit (also referred to as the High-Speed Storage or HSS) are available: Model 1, which provides 131,072 bytes; and Model 2, which provides 262,144 bytes. Both models have an internal cycle time of 750 ns.

Each 2365 Model 2 Processor Storage is organized into even- and odd-numbered doubleword storage areas. The access path to these areas is through a shared interface which allows the BCU to interleave even and odd requests; i.e., an odd request can be issued by the BCU halfway through an even cycle, and vice versa. By interleaving references to even-numbered and odd-numbered doublewords in main storage, the effective storage cycle approaches one half the cycle time for the unit. As a maintenance aid, the interleaved mode can be defeated to allow operation of a program solely in the odd or even area of storage.

2361 Core Storage (Optional)

The overall storage capacity of the system may be increased by means of the 2361 Attachment feature. This feature provides bulk, direct-access core storage through the attachment of up to four 2361 Core Storage Units (also referred to as Large Capacity Storage or LCS). Two models are available: Model 1, which has a storage capacity of 1,048,576 bytes; and Model 2, which has a capacity of 2,097,152 bytes. Either one 2361 Model 1 or up to four 2361 Model 2's may be attached if interleaving is ***not*** utilized; ***with*** interleaving, either two 2361 Model 1's, or two or four 2361 Model 2's may be attached.

Eight bytes are accessed per storage cycle. Read access time is 3 usec; a read/rewrite storage cycle requires 8 usec. If interleaving is utilized, the LCS units are attached to the system in pairs, and the addressing assignment is split so that one unit contains all odd addresses and the other all even. Thus, these units can be addressed on an interleaved bias to achieve a sequential access rate of 4 usec per doubleword.

2065 Central Processing Unit

The 2065 CPU performs arithmetic, logical, and control instructions specified by a stored program. It contains facilities for addressing main storage, processing fixed-point, floating-point, and decimal arithmetic, operating on logical data, sequencing instructions in the desired order, and initiating I/O operations. Also provided are facilities for character-handling, processing of fixed-length and variable-field-length (VFL) data, indexing, and indirect addressing. Functionally the CPU can be divided into four major sections: control, BCU, instruction fetching, and instruction execution.

Control Section

The basic CPU clock cycle period is 200 ns. A clock signal generator provides a 5-megaHertz (5-mHz) symmetrical (100-ns clock and 100-ns not-clock portions) signal. To provide additional time for CPU logic functions, the symmetrical clock signal is modified to give a 5-mHz unsymmetrical (80-ns clock and 120-ns not-clock portions) signal. Clock distribution logic distributes and synchronizes the clock signals to the logic gates, and stops distribution of clock signals to most of the CPU processing logic upon detection of a machine check during certain operations.

A significant feature of the CPU is the read-only storage (ROS) used to control operations. The ROS contains a permanently recorded microprogram holding information that remains fixed during machine operations. The information is in the form of 100-bit words, each word containing a unique, predetermined bit pattern. When decoded, the bits control gates to route data in the CPU. Word access time is approximately 95 ns. The information can be read out as required, but a physical modification is necessary to change the stored information. In general, a control word is read out from ROS at the end of each machine cycle (200 ns) and controls the CPU during the following machine cycle. Each ROS word contains the address of the ROS word to control the CPU during the following cycle. The number of control words (and machine cycles) required to perform a particular operation may vary because both the individual functions and the address of the next ROS word are modifiable by the (1) operation in progress, (2) data or control bit configuration, and (3) detection of interruptions or exceptional conditions. Used as a control device, ROS eliminates the need for most complex instruction decoders and sequencing networks.

The Program Status Word (PSW), a doubleword, contains the information required for proper program execution. Primarily, the PSW controls instruction sequencing and holds the system status in relation to the program being executed. By storing the PSW, the program can preserve the status of the CPU for subsequent analysis. By loading a new PSW or part of a PSW, the state of the CPU may be changed.

Bus Control Unit

The main storage associated with a system is shared by the CPU and I/O channel. A main storage address bus (SAB) and two main storage data buses (SDBI and SDBO) are also common to the CPU and I/O channel. The BCU, located in the CPU, monitors and controls the availability of these buses and selects the main storage area to be used. Storage requests (stores and fetches) generated by the CPU and I/O enter the BCU, are processed by means of a priority scheme, and are executed when the requesting unit is granted priority. When there is a conflict between the CPU and an I/O channel wanting to use main storage, the BCU gives priority to the channel. Priority between channels is also preassigned, with channel 1 having highest priority, followed by channels 2, 0, 3, 4, 5, and 6. When the CPU cannot access main storage because of a channel, the CPU clock is stopped until main storage is available to the CPU.

Each unit requiring access to main storage issues a request to the BCU via a separate (simplex) control line. The BCU examines the control lines from each unit, establishes which unit should be granted access to main storage for the particular cycle, and generates response signals for that unit. These response signals control the placing and sampling of data on the common bus. Access to main storage for a particular unit is granted by the BCU on a cycle-to-cycle basis, and all units again compete for the subsequent storage cycle.

The CPU operates on a basic cycle of 200 ns while the main storage cycle time is 750 ns. To increase processing speed, the CPU overlaps, whenever possible, storage requests for new data with processing of the existing data within its buffer registers.

Instruction Fetching Section

An instruction buffer, the Q-register, provides buffering for eight instruction bytes (four halfwords), thus reducing the number of storage requests that must be made to fetch instructions from main storage. Associated with the Q-register is the R-register. It holds the halfword (two bytes) containing the op code, received from the Q-register, of the instruction to be executed next. As a result, the R-register allows overlapping of instruction fetching with the refilling

of the Q-register. The E-register holds the halfword containing the op code of the instruction being presently executed. The instruction op code is used to address the ROS, which provides the required microprogram for execution of a specific instruction.

An interruption capability permits the CPU to change state automatically as a result of conditions arising outside the system, in I/O units, or in the CPU. An interruption switches the CPU from one program to another by changing the instruction address; the interruption cause and all essential machine status information are stored for analysis in a program status word. This information is available to the system as required. When a different program is requested, the status information about the current program is stored temporarily and is retrieved when that program is to be continued. This facility allows interrupted programs to resume at the point of interruption without the loss of control conditions.

Before an instruction is executed, it is tested for odd parity. The op-code halfword is tested in the E-register. The remaining halfwords, if any, are tested by the parallel adder half-sum checking circuits as the effective address is calculated.

Instruction Execution Section

The CPU operates on a basic internal cycle time of 200 ns. The data flow (Diagram 3-2, FEMDM) utilizes two major working registers (ST and AB) to give increased speed and simplified implementation of the instruction set. The 60-bit parallel adder is the focal point for most data transfers and facilitates handling the full long fraction in floating-point operations. An eight-bit serial adder provides simultaneous execution of the floating-point exponent as the fraction is operated on in the parallel adder, and has the capability of executing decimal arithmetic and VFL instructions.

The CPU extracts from the doubleword fetched from main storage the bytes on which it will operate. Thus, storage accesses are not required for every byte. As a result, processing speed is increased and system performance is improved. Data may be stored, however, on a byte basis; any number and combination of bytes up to 8 (doubleword) can be stored in one storage cycle. Addresses for data are formulated in the arithmetic section of the CPU and then placed in the D-register for addressing main storage. The D-register is also used for addressing I/O devices on I/O instructions.

The CPU incorporates a local storage of 25 word-length registers: 16 word-length general-purpose registers for fixed-point operations; 4 doubleword length (8 word-length) floating-point registers for floating-point operations; and 1 word-length register, called the working register, for miscellaneous operations. Local storage serves two functions: (1) it is used in generating operand addresses in main storage and (2) it holds operands and intermediate results of data operations, thus eliminating the need for special-purpose registers, such as accumulators. Local storage employs nondestructive readout, eliminating the need for regeneration, and operates on a 200-ns cycle with an 80-ns access time.

In the CPU, checking is facilitated by providing a parity bit for each byte of data. Odd parity is maintained. A parity check is made on data transferred to and from main storage. Most data transfers within the CPU are made via the parallel adder, in which the parity of each operand byte is checked against the half-sums. The parity of the sum, which is not a function of the parity of the operands but is generated logically within the adder, is checked against the latched sum.

As programs are executed, they are checked for correct instructions and data. This monitoring action identifies and separates program errors and machine errors. Because each type of error causes a unique interruption, program errors cannot create machine checks (errors).

Input/Output

- I/O section consists of devices, control units, channels.
- I/O operations are initiated by CPU and thereafter are performed independently of CPU.

Use of the I/O devices by the system is referred to as an I/O operation. The basic I/O operations are reading, which transfers data from an I/O device to main storage, and writing, which records at an I/O device data received from main storage.

The operation of a specific I/O device is governed by a control unit (Diagram 3-1, FEMDM) which provides control unique to its attached devices. The control unit may be a part of the I/O device or may be shared by a number of devices and be a separate logical entity. Besides controlling the operations of specific I/O devices, the control unit makes all I/O devices appear identical to the channel.

All I/O devices are connected, via their control units and the Standard I/O Interface, to the I/O channels. Each channel provides the logic necessary for synchronizing I/O data cycles with those of main storage and exerts programmed control over operations of the I/O devices connected to it. Logically, the channel can be considered as an autonomous entity provided for transferring information between I/O control units and main storage under control of the CPU.

In the System/360 Model 65, the channels interface directly with the CPU, providing a path for the exchange of system control information. This connection is used by the CPU to start and monitor a channel program, and by the channel to alert the CPU of the progress and termination of I/O operations. A BCU-controlled connection between the channel and main storage provides a path for the channel to fetch and store data, as well as channel command words that supplement the direct communication between the channel and the CPU.

Lengthy I/O operations (such as reading and writing) are executed in parallel with the CPU operations. The CPU need only initiate an I/O sequence by issuing an instruction to the channel. Thereafter, the channel establishes the address of the first channel command word for that sequence, establishes connection with the required I/O device, and verifies that the operation designated by the CPU can be executed. Once the device is started and the channel is set up to execute its commands, the CPU is released. Several commands can be chained, creating an I/O program for the device.

Main storage cycles are required during I/O operations to transfer data to or from main storage. These cycles do *not* interfere with CPU operations, except when both the CPU and the channel concurrently attempt to access main storage.

Channel

After the operation with the device is initiated, the channel assembles or disassembles data and synchronizes the transfer of data bytes with main-storage cycles. To transfer data, the channel maintains and updates an address and a count that describe the destination or source of data in main storage. Data is transferred between main storage and the channel eight bytes at a time, and between the channel and the control unit one byte at a time.

Modes of Operation. Data can be transmitted between main storage and an I/O device in two modes: burst mode and multiplex mode.

In the burst mode of operation, a device monopolizes the channel and stays logically connected to the channel for transmission of a burst of information. The burst can consist of a number of bytes, a block of data, or a sequence of blocks with associated chained commands. Only one I/O device can communicate with the channel during the time a burst is transmitted.

In the multiplex mode of operation, the channel facilities are shared by a number of concurrent I/O operations. All I/O operations are split into short intervals of time during which only a segment of information is transmitted over the interface. The intervals associated with different operations vary in response to demands from the I/O devices. The channel controls are occupied with any one operation only for the time required to transmit a segment of information. The segment can consist of a byte of data, a few bytes of data, or a control sequence such as the initiation of a new operation.

Operation in burst and multiplex modes is distinguished by the way the channels respond to I/O instructions. A channel operating in the burst mode appears busy to new I/O instructions. A channel operating in the multiplex mode appears available to new I/O instructions.

Types of Channels. Highlights:

- Model 65 can accommodate two types of channels: selector channel, which operates in burst mode; multiplexer channel, which can operate in either burst or multiplex mode.
- Model 65 can utilize up to six selector channels (addresses 1 through 6) and one multiplexer channel (address 0).

A model 65 can be equipped with two types of channels: 2860 Selector and 2870 Multiplexer. The channels are classified according to the modes of operation they can sustain.

The channel facilities required for sustaining an I/O operation are termed a ***subchannel***. The subchannel consists of the channel storage used for recording the addresses, count, and any status and control information associated with the I/O operation. The mode in which a channel can operate depends upon whether it has one or multiple subchannels.

The 2860 Selector channel has only one subchannel and operates only in the burst mode. The bursts extend over the whole block of data, or, when command chaining is specified, over the whole sequence of blocks. The selector channel cannot perform any multiplexing and, therefore, can be involved in only one data transmission operation at a time. Once a data transfer is initiated, a logical connection is established between the selected I/O device and the channel. This connection persists for the duration of the operation, and other I/O devices cannot communicate with the channel. In the meantime, however, other I/O devices attached to the channel can execute operations that do not involve communication with the channel. The selector channel keeps scanning I/O devices for interruption conditions when it is not executing an operation or a chain of operations. A maximum of six selector channels may be attached to a 2065; their addresses are 1 through 6. A 2860 Model 1 contains one selector channel; a 2860 Model 2 contains two; a 2860 Model 3 contains three.

The 2870 Multiplexer channel is intended primarily for operation with lower-speed devices. It contains multiple subchannels and can operate in either multiplex or burst mode. It can switch between the two modes of operations at any time, and an operation on any one subchannel can occur partially in the multiplex and partially in the burst mode.

When the multiplexer channel operates in the multiplex mode, it can concurrently sustain one I/O operation per subchannel, provided that the aggregate data rate does not exceed the capacity of the channel. Each subchannel appears to the program as an independent selector channel. When the multiplexer channel is not servicing an I/O device, it keeps scanning its devices for data and for interruption conditions.

When the multiplexer channel operates in the burst mode, the subchannel associated with the burst operation monopolizes all channel facilities and appears to the program as a single selector channel for the duration of the burst. The remaining subchannels on the multiplexer channel remain dormant and resume communicating with I/O devices when the burst is completed.

One 2870 Multiplexer channel may be attached to a 2065, and its address must be 0.

Control Units

The control unit is the buffer between the I/O device and the channel. It adapts the characteristics of the device to the standard form of control provided by the channel. A control unit may be housed separately or it may be physically and logically an integral part of the device. The control unit accepts control signals from the channel, controls the timing of data transfer to the channel, and provides indications of device status.

Except for the signal used to establish priority among control units, all communication to and from the channel occurs over a common bus, and any bus signals provided by the channel are available to all control units. At any instant, only one control unit is logically connected to the channel. Selection of a control unit for communication with the channel is controlled by a signal that passes serially through all control units and permits each control unit to respond sequentially to the signals provided by the channel. A control unit remains logically connected to the channel until it has transferred the information it needs or has, or until the channel signals it to disconnect.

Up to eight control units (limited by electrical considerations) can be connected to one channel.

I/O Devices

The I/O devices consist of auxiliary storage as well as equipment used to communicate with the system users. The auxiliary storage includes direct-access devices such as disks and drums, as well as sequential-access devices such as magnetic tape units and punched card equipment. The equipment used to communicate between the system and its users includes a wide range of devices: printers, visual display units, keyboards, and communication terminals. Up to 256 directly addressable devices (limited by addressing facilities) can be attached to one channel.

Multisystem Configurations

- Multisystem consists of two or more interconnecting systems, each with a CPU.
- Multisystem can be achieved by:
  1. Channel-to-Channel Adapter
  2. Transmission Control Unit
  3. Two-Channel Switch Feature
  4. Device Switching Unit
  5. Shared LCS Feature
  6. Direct Control Feature
  7. Multisystem Feature

A multisystem consists of two or more systems, each with a CPU, that can communicate with each other without manual intervention. Thus, in addition to their specialized jobs, the individual systems may pool resources to perform a common job.

A multisystem configuration consists of two CPU's with a complement of storage and I/O equipment. This equipment is shared by the CPU's, but is considered to be logically independent if the CPU's are interconnected by well-defined interfaces so that they can, upon reconfiguration, operate without communicating with each other. Examples of logically independent system components are storage units, CPU's, I/O control units, and I/O devices.

Although logically independent, these system components may still be physically dependent because they share common equipment. Communication between the CPU's of a multisystem may be achieved by transmitting information from one CPU to another through a connecting link or by giving them access to a shared storage medium.

Figure 1-3 shows, in simplified form, the major components of a single, or simplex, system. A multisystem configuration can be achieved by the following:

1. Channel-to-Channel Adapter. Allows connecting the I/O interfaces of two channels (Figure 1-4). The main purpose of the channel-to-channel adapter is to make each channel appear as a control unit to the other channel. Transmission of data between the two channels

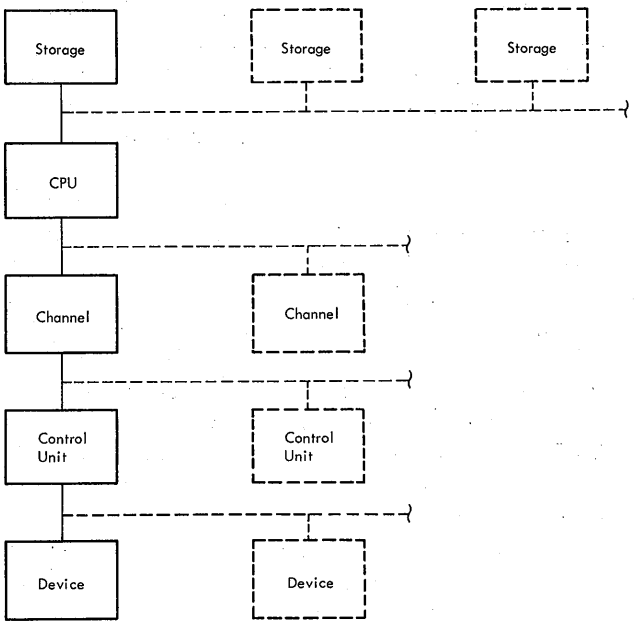


Figure 1-3. Functional Structure of a Simplex System

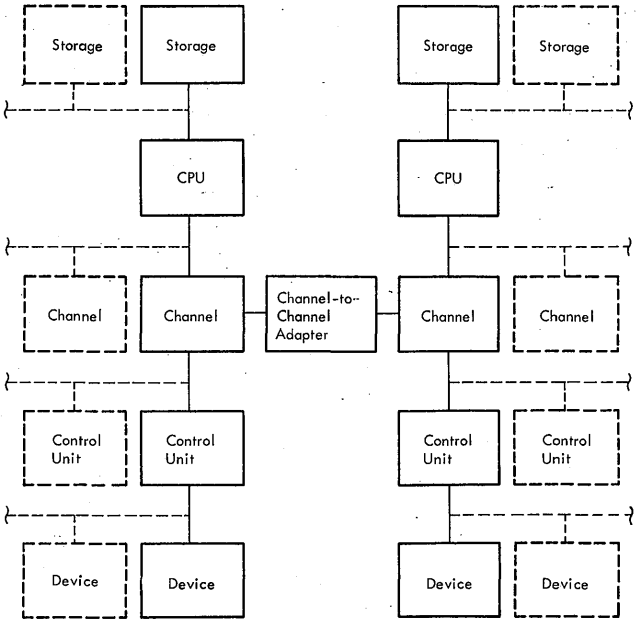


Figure 1-4. Channel-to-Channel Adapter as Multisystem Connector

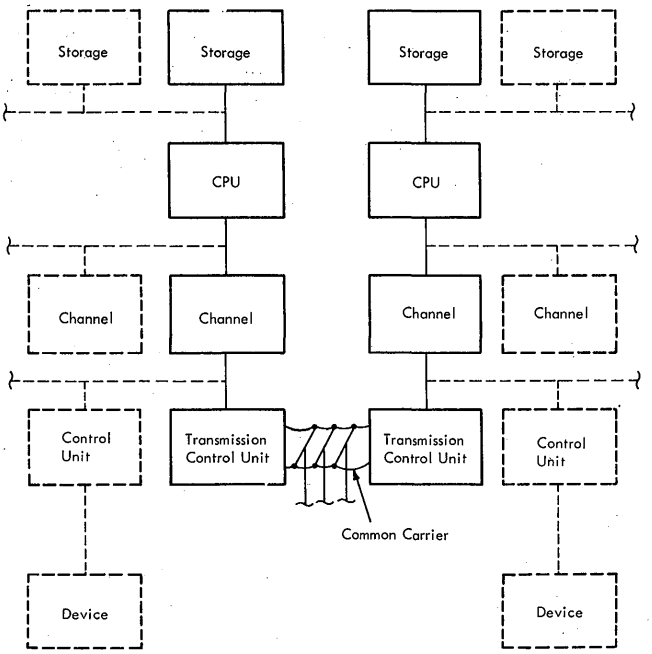


Figure 1-5. Transmission Control Unit as Multisystem Connector

is by byte at a rate established by the two channels. Because of the standardization of the I/O interface, this adapter may connect any model of the System/360 to any other model, and may use any type of channel on a given model. Any number of channel-to-channel adapters may be used in a multisystem, but their main function is in a multisystem emphasizing medium reconfiguration time or equipment specialization.

2. Transmission Control Unit. Permits communication by private line or common carrier. As indicated in Figure 1-5, communication is established by a specialized device interface rather than via the channel interface. The rate of data transmission is determined mainly by the line capacity. Any two models of System/360, as well as those of any other system, can be connected. The major multisystem application of the transmission control unit is for geographically separated computers.
3. Two-Channel Switch Feature. Permits sharing of control unit. When two or more CPU's have access to a common file, information placed into the common file by one CPU can be read by another. In contrast to transmission, sending and receiving are not simultaneous, and a one-to-one relation between recording and retrieval is not necessary. The choice of the shared media is determined by access time, capacity, and cost per bit. Shared devices are useful for program restarting information for job recovery upon reconfiguration. Disks, drums, and tape units may be pooled for storage of system programs as well as a means of communication between specialized CPU's to achieve improved turnaround time. Because a control unit normally controls several disk files, drums, or tape units, a switch between a channel and a control unit allows efficient sharing of a control unit by two CPU's (Figure 1-6).
4. Device Switching Unit. Tape units or other I/O devices are shared between control units (Figure 1-7), rather than control units being shared between channels as in Figure 1-6. This choice permits pooling of tape units between control units and permits simultaneous operation of any combination of tape units. This logical

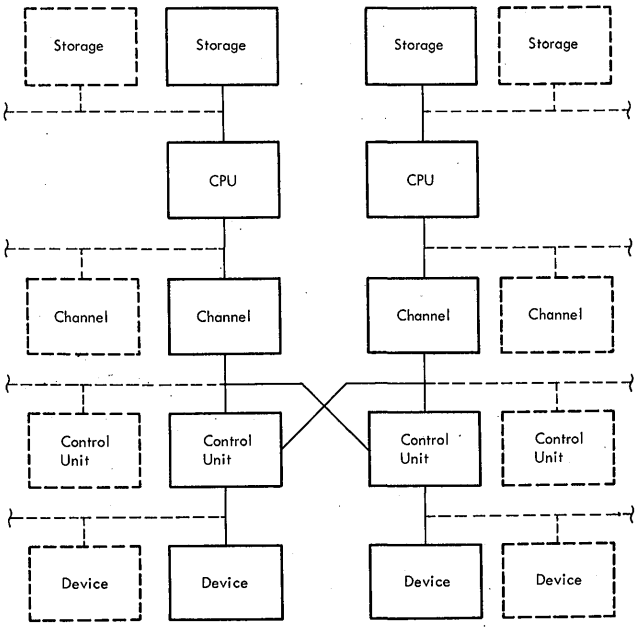


Figure 1-6. 2-Channel Switch Feature as Multisystem Connector

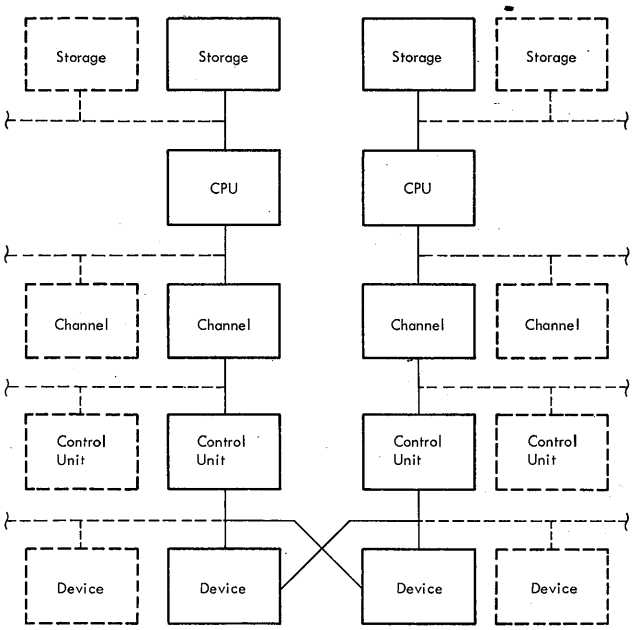


Figure 1-7. Device Switching Unit as Multisystem Connector

ability increases the thruput of a multisystem, as well as providing a common file. For example, the sharing of any pair of tape drives by two control units improves the sorting time significantly.

5. Shared LCS Feature. LCS can be shared by two CPU's (Figure 1-8). When one program is executed by different CPU's, it is desirable to have the locations of instructions and data located in identical addresses in every CPU. This addressing convention is adopted in System/360 for multisystem operation. This application of shared storage is mainly used in multisystems requiring a short reconfiguration time.
6. Direct Control Feature. Communications between CPU's may be direct (Figure 1-9). Using the Direct Control feature, control signals and one byte of control information are transferred without waiting for recognition. (The F- and G-registers in the 2065 are used by the Read Direct and Write Direct instructions, respectively.) This multisystem connection, unlike those previously described, does not lend itself to the transfer of large amounts of data.
7. Multisystem Feature. Any one or any combination of the foregoing interconnection methods results in a multisystem. However, the most sophisticated method of interconnecting two Model 65's utilizes the Multisystem feature to form the Multiprocessing System/360 Model 65 (Figure 1-10). 2365 Processor Storage Model 13 units are used, the CPU's are equipped with the Multisystem feature (with the Direct Control feature as a prerequisite), and most of the control units are equipped with the Two-Channel Switch feature. Any of the other interconnection methods, (except the Shared LCS feature) may also be incorporated. The Multiprocessing System/360 Model 65 and the Multisystem feature are described in Chapter 4, Section 2.

The interconnection methods described above are not always used to form a multisystem. A single system can also effectively make use of:

1. Channel-to-Channel Adapter feature.
2. Transmission control unit.
3. Two-Channel Switch feature.
4. Device-switching unit.
5. Direct Control feature (connected to an external device).

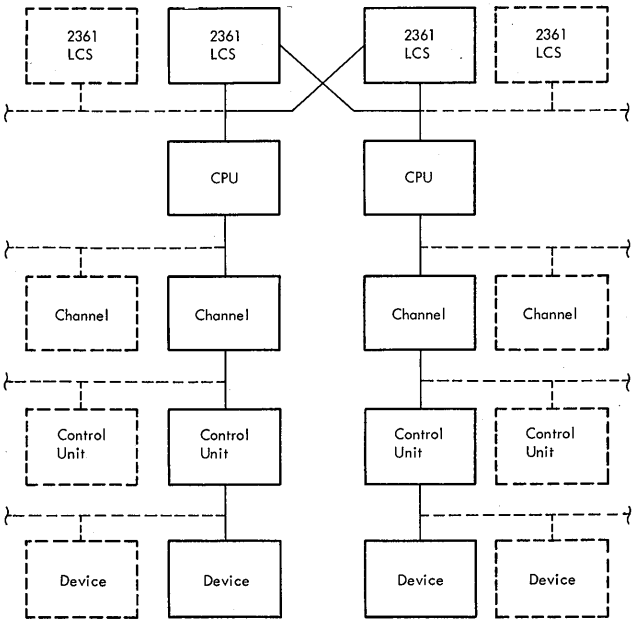


Figure 1-8. Shared LCS Feature as Multisystem Connector

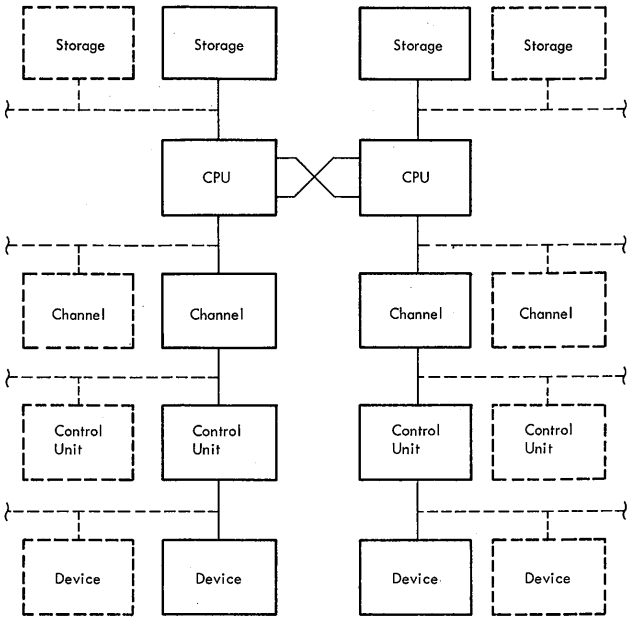


Figure 1-9. Direct Control Feature as Multisystem Connector

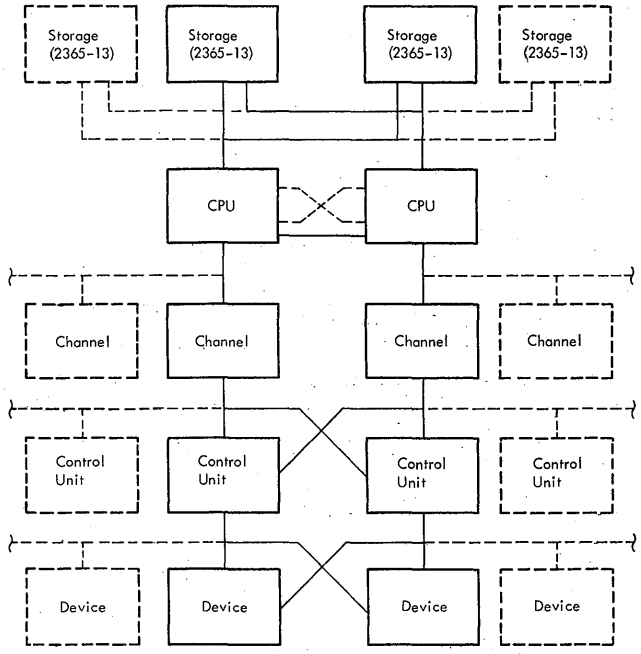


Figure 1-10. Multiprocessing System/360 Model 65

www.IBM360.info

To understand the operation of the CPU, it is necessary to become familiar with the system coding. Accordingly, this section discusses: (1) the hexadecimal (hex) number system, (2) the 8-bit zoned character codes, (3) the instruction formats and operand designations, and (4) the various data formats.

HEXADECIMAL NUMBER SYSTEM

- System uses 16 symbols: 0–9, A–F.
- Base of system is 16.
- System is shorthand notation for binary numbers.
- Four binary bits are represented by one hex symbol.
- Byte is represented by two hex symbols.

Binary numbers have approximately 3.3 times as many terms as their decimal counterparts. This increased length presents a problem when talking or writing about binary numbers. A long string of 1's and 0's cannot be effectively spoken or read. A shorthand system is necessary, one that has a simple relationship to the binary system and that is compatible with the basic eight-bit byte used in the CPU. The hexadecimal (hex) number system meets these requirements.

The hex system has 16 symbols: 0–9, A–F. Counting is performed as in the decimal and binary systems. When the last unique symbol (F) is reached, a 1 is placed in the next position to the left and counting resumes with a 0 in the original position, as follows:

0	10	20	A0
1	11	21	A1
2	12	22	A2
3	13	23	↓
4	14	↓	
5	15		↓
6	16		and so on
7	17		
8	18		
9	19		
A	1A	9A	
B	1B	9B	
C	1C	9C	
D	1D	9D	
E	1E	9E	
F	1F	9F	

One hex symbol can represent four binary bits. Thus the 8-bit binary byte, in turn, can be represented by two hex symbols. The relationship between the hex, binary, and decimal systems is as follows:

Hex	Binary	Decimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
A	1010	10
B	1011	11
C	1100	12
D	1101	13
E	1110	14
F	1111	15

The important relationship to remember is that four binary positions are equivalent to one hex position.

Hex numbers are represented in the same manner as decimal and binary numbers, except that the base is 16. The terms of the number represent the coefficients of the ascending powers of 16. For example, consider the hex number 257 (decimal equivalent equals 599):

$$257 = (2 \times 16^2) + (5 \times 16^1) + (7 \times 16^0)$$

$$= (2 \times 256) + (5 \times 16) + (7 \times 1)$$

$$= 512 + 80 + 7$$

$$= 599_{10}$$

EIGHT-BIT ZONED CHARACTER CODES

All I/O devices requiring a zoned character code use either the Extended Binary-Coded-Decimal Interchange Code (EBCDIC) or the USA Standard Code for Information Interchange extended to eight bits (USASCII-8). The EBCDIC and USASCII-8 codes for the hex characters 0–F are listed below. For charts showing the complete EBCDIC and USASCII-8 codes with associated graphic characters, refer to the SRL, *IBM System/360 Principles of Operation*, Form A22-6821-6. The codes do not have a symbol defined for all 256 eight-bit codes. To represent codes that do not

have a defined symbol, two hex terms (representing four bits each) may be used instead of the eight-bit code.

Hex Code	Printed Graphic	EBCDIC Code	USASCII-8 Code
0000	0	1111 0000	0101 0000
0001	1	1111 0001	0101 0001
0010	2	1111 0010	0101 0010
0011	3	1111 0011	0101 0011
0100	4	1111 0100	0101 0100
0101	5	1111 0101	0101 0101
0110	6	1111 0110	0101 0110
0111	7	1111 0111	0101 0111
1000	8	1111 1000	0101 1000
1001	9	1111 1001	0101 1001
1010	A	1100 0001	1010 0001
1011	B	1100 0010	1010 0010
1100	C	1100 0011	1010 0011
1101	D	1100 0100	1010 0100
1110	E	1100 0101	1010 0101
1111	F	1100 0110	1010 0110

INSTRUCTION CODING

The Model 65 uses the Universal instruction set, which enables the CPU to execute fixed-point, floating-point, decimal, logical, branching, status switching, and I/O instructions. Two more status switching instructions, Read Direct and Write Direct, may be implemented by installing the Direct Control feature.

The Universal instruction set uses five instruction formats: RR, RX, RS, SI, and SS. Operands are designated as first, second, or third operands. For addressing purposes, the operands are grouped into three classes:

1. Effectively addressed operands in main storage.
2. Immediate operands in the instruction format.
3. Operands in local storage (LS).

Instruction Formats

- Five instruction formats are available: RR, RX, RS, SI, and SS.
- Halfword is basic building block for instruction.
- Instructions are made up of 1, 2, or 3 halfwords.
- First halfword contains 8-bit op code and, depending on format:
  4-bit LS address for operand, or operand address component.
  4-bit mask.
  8-bit immediate operand.
  4-bit or 8-bit length fields.
- Second and third halfwords contain:
  4-bit LS address for operand address component.
  12-bit displacement.

Five instruction formats are available, denoted by the format codes RR, RX, RS, SI, and SS. The format codes express, in general terms, the operation to be performed. RR denotes a register-to-register operation; RX, a register-to-indexed-storage operation; RS, a register-to-storage operation; SI, a storage and immediate-operand operation; SS, a storage-to-storage operation. The Universal instruction set for the Model 65 may be divided into seven classes; the breakdown by format is as follows:

Instruction Class	Format
Fixed-Point	RR, RX, RS
Floating-Point	RR, RX
Decimal	SS
Logical	RR, RX, RS, SI, SS
Branching	RR, RX, RS
Status Switching	RR, SI
I/O	SI

The basic unit of length for instructions is the halfword, consisting of two bytes. The length of an instruction format can be 1, 2, or 3 halfwords. It is related to the number of initial main storage references necessary for the operation. An instruction making no reference to main storage (RR) consists of one halfword; an instruction making one reference (RX, RS, or SI) consists of two halfwords; an instruction making two references (SS) consists of three halfwords. All instructions must be located in main storage on an integral boundary for halfwords. The five formats are shown in Figure 1-11.

For purposes of describing the execution of instructions, operands are designated as first, second, or third operands, referring to the manner in which the operands participate in the operation. The operand to which a field in an instruction format pertains is denoted by the number following the letter designation of the field; for example, the R1 field is the address of an LS register containing the first operand; R2, the second operand.

As shown in Figure 1-11, the first halfword of each format consists of two parts. The first byte contains the operation code (op code), which identifies the operation to be performed. Bits 0 and 1 specify the format, bits 2 and 3 specify the class of instruction, and bits 4 through 7 identify the instruction within the class. The second byte of the first halfword is used as either two four-bit fields or a single eight-bit field. The fields and the information contained within the fields are as follows:

1. R1, R2, and R3: four-bit address of an LS register containing the first, second, and third operands, respectively.

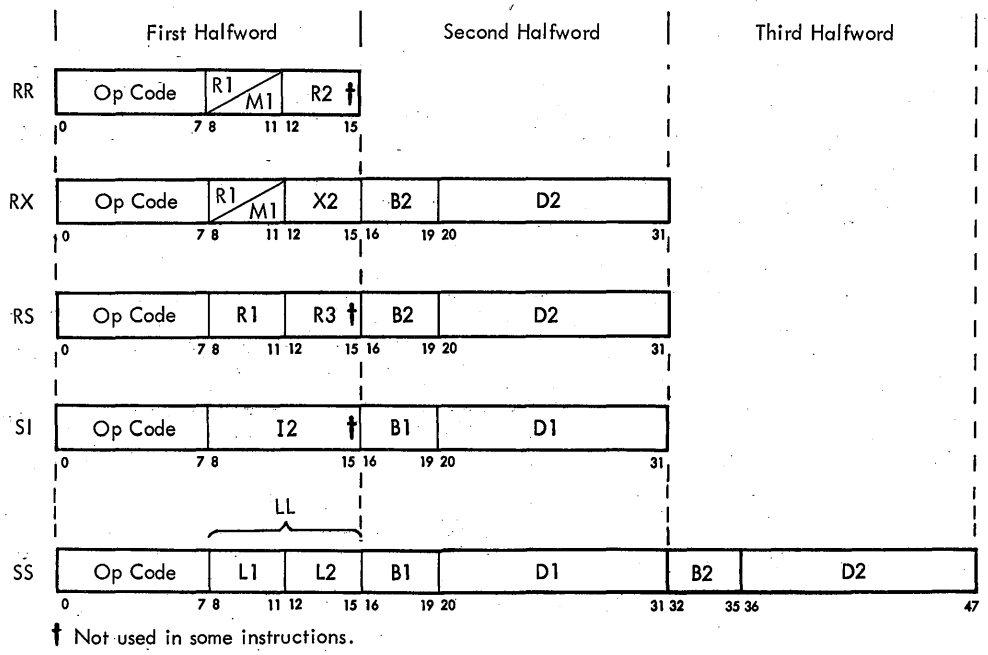


Figure 1-11. Instruction Formats

2. M1: four-bit mask used in some branching instructions.
3. X2: four-bit address of an LS register containing the index value used in generating the effective second operand address.
4. I2: eight-bit byte of immediate data (second operand).
5. L1 and L2: four-bit length (up to 16 bytes) of first and second decimal VFL operands, respectively.
6. LL: eight-bit length field (up to 256 bytes) for logical VFL operands.

The second and third halfwords always contain the same information: a four-bit address of an LS register containing the base value to be added to the following 12-bit displacement field.

Operand Addressing

For addressing purposes, operands are grouped into three classes: (1) effectively addressed operands in main storage, (2) immediate operands in the instruction format, and (3) operands in LS.

Effectively Addressed Operands

- Operands are either fixed-length or VFL.
- Fixed-length operands are located on integral boundary.
- Length of VFL operand is specified by L or LL field.
- L and LL fields denote number of bytes to right of addressed byte.
- Effective operand address is sum of 24-bit base address, 12-bit displacement, and 24-bit index value.
- Base address and index value are located in LS.
- Displacement is located in instruction format.

An effectively addressed operand is selected from a main storage location not related to the location of the instruction referring to it. It is specified by means of a main storage address. When the operand consists of more than one byte, the address gives the location of the first byte; subsequent bytes are located in the next-higher addressed byte locations. Both the first and second operands of an instruction can be effectively addressed.

Effectively addressed operands can be of either fixed length or variable field length. The length of VFL operands, in terms of the number of bytes to the right of the addressed byte, is specified by the L or LL field of the instruction. The LL field can be eight bits long and thus can specify a maximum operand field length of 256 bytes.

In the instruction format, an effectively addressed operand is specified by a base address, a displacement, and, in some cases, an index value. The base address and the index value are contained in LS general-purpose registers addressed by the B and X fields, respectively, of the instruction. The registers contain 32 bits, the low-order 24 of which constitute an unsigned address component (base address or index value). The high-order eight bits of the register are ignored. The 24-bit base address is included in every address computation. The 24-bit index value is included in the address computation as specified by the RX instruction format.

The displacement value is a 12-bit number contained in the D-field of the instruction. It is included in every address computation. The displacement provides for relative addressing up to 4095 bytes beyond the base address.

In computing the effective operand address, the base address and the index value are treated as 24-bit positive binary integers having no sign position. The displacement is

similarly treated as a 12-bit positive binary integer. The three numbers are added. Because every operand address includes a base address, the sum is always 24 bits long. Any overflow above the 24 low-order bits of the sum is ignored, causing a lower address to be generated. If this lower address is *above* the maximum available storage, an addressing program interruption occurs. If the lower address is available the CPU accesses that location.

An instruction may contain zeros in the B, X, or D field. In the case of the B and X fields, a zero does not denote the address of LS general-purpose register 0, but indicates that base and index values of zero are to be used in generating the effective operand address. Similarly, a D field of zero indicates a displacement of zero.

Fixed-length fields, halfwords, words, and doublewords, must be located in main storage on an integral boundary for that length field. A boundary is called "integral" for a field when its storage address is a multiple of the length of the field in bytes (Figure 1-12). For example, words (four bytes) must be located in main storage so that their address is a multiple of the number 4. A halfword (two bytes) must have an address that is a multiple of 2, and doublewords (eight bytes) must have an address that is a multiple of 8.

Main storage addresses are expressed in binary form. Therefore, integral boundaries for halfwords, words, and doublewords can be specified only by binary addresses in which 1, 2, or 3 of the low-order bits, respectively, are zero (Figure 1-12). Thus, integral boundaries for words are binary addresses in which the two low-order bit positions are zero; for example, 00000, 00100, 01000, and 01100.

VFL fields are not limited to integral boundaries, but may start on any byte location.

Immediate Operands

Immediate operands are contained in SI instructions for logical operations. They are one byte (bits 8–15) long, serve as the second operand, and are designated I2.

Operands in Local Storage

- LS registers are addressed by four-bit R-field in instruction format.
- LS GPR's are addressed 0–15.
- LS FPR's are addressed 0, 2, 4, and 6.
- For fixed-point doubleword operands, the register address implies use of a pair of adjacent registers.

Fixed-point and floating-point operands may be located in the 16 general-purpose registers (GPR's) and the 4 floating-point registers (FPR's), respectively, of LS. The registers are addressed by a four-bit field in the instruction, designated the R-field. The GPR's are designated by addresses 0–15, whereas the FPR's are identified by addresses 0, 2, 4, and 6. (When an FPR is designated by any other address, a specification program interruption occurs.) The op code of the instruction implies whether the GPR's or the FPR's are to be used.

The GPR's are 32 bits (one word) in length. Fixed-point operands normally have an implied length of one word. In some operations, one register address implies the use of a pair of adjacent GPR's, thus providing a doubleword. For these instructions, the addressed register (say R1) contains the high-order operand bits and must have an even address, and the implied register (R1 + 1) contains the low-order operand bits and has the next higher address.

The FPR's are 64 bits or a doubleword in length, and can contain either a short (one word) or a long (doubleword) floating-point operand. A short operand occupies the high-order bits of an FPR; the low-order bits are ignored.

DATA FORMATS

Data can be numeric, alphabetic, or logical, and fixed or variable in length. Numeric data is distinguished as fixed-

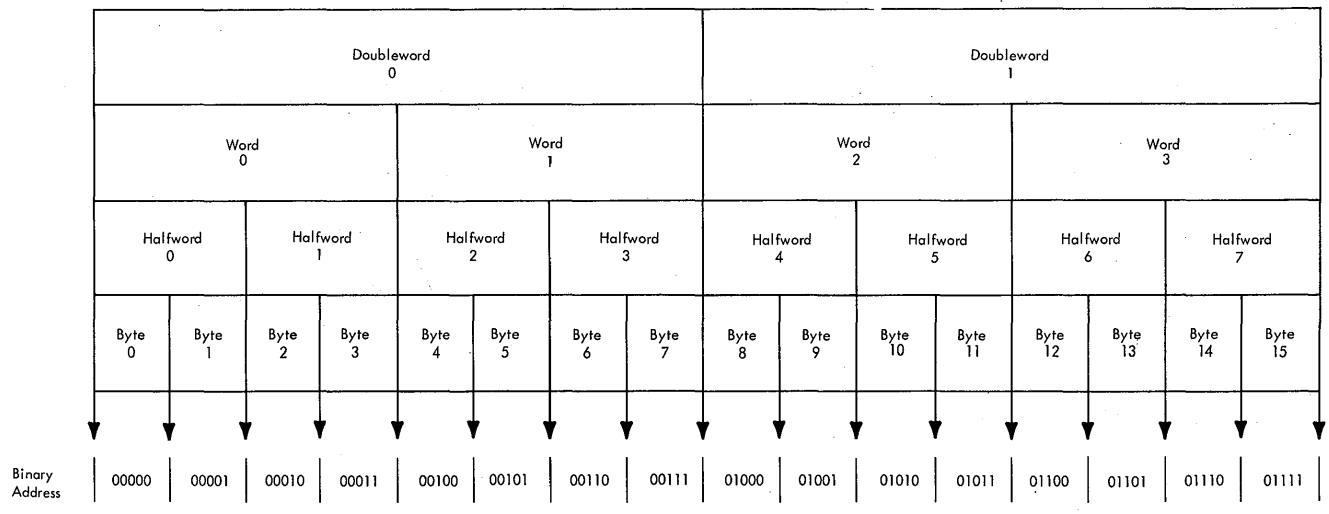


Figure 1-12. Main Storage Integral Boundaries

point, floating-point, or decimal. The data may be divided into four classifications:

1. Fixed-point numbers, having a binary radix and a fixed length, usually a word or a halfword.
2. Floating-point numbers, represented by a 7-bit characteristic and a signed hex fraction, occupying either a word or a doubleword.
3. Decimal numbers, represented by four-bit binary-coded-decimal (BCD) digits, usually packed two digits to a byte, and variable in length.
4. Logical information, represented by eight-bit zoned character codes, and fixed or variable in length.

Fixed-Point Data

Fixed-point instructions are available for loading, adding, subtracting, comparing, multiplying, and dividing. A pair of conversion instructions, Convert to Binary and Convert to Decimal, provides transition between decimal and binary radix without the use of tables.

The basic fixed-point operand is the 32-bit binary word. To improve performance and storage utilization, 16-bit halfword operands may be specified in most operations. In both lengths, bit position 0 holds the sign of the number, with the remaining bit positions designating the magnitude of the number. To preserve precision, some products and all dividends are 64 bits long.

Because a 24-bit address can be accommodated in the 32-bit word, fixed-point instructions can be used both for integer operand arithmetic and for address computation. This combined usage provides economy and permits the entire fixed-point instruction set to be used for address computation. Thus, multiplication, shifting, and logical manipulation of address components are possible.

Number Representation

- Positive numbers are represented in true binary form with sign of 0.
- Negative numbers are represented in 2's complement notation with sign of 1.

All fixed-point operands are treated as signed integers. Positive numbers are represented in true binary form with a sign bit of 0. Negative numbers are represented in 2's complement notation with a sign bit of 1. In all cases, the bits between the sign bit and the leftmost significant bit of the integer are the same as the sign bit; i.e., all 0's for positive numbers, all 1's for negative numbers. Therefore, when an operand must be extended with high-order bits, each nonsignificant bit is made equal to the sign bit.

Negative fixed-point numbers are formed in 2's complement notation by complementing the true binary representation of the number and adding 1. For example, to convert the decimal number +26 to 2's complement form (−26), proceed as follows:

	S 1 ← Integer → 31
Decimal +26 to true binary form:	0 0000000 00011010
Complement the binary number:	1 1111111 11100101
Add 1:	1
Result is -26 (2's complement form):	1 1111111 11100110

The result is equivalent to subtracting the number 00000000 00011010 from 1 00000000 00000000.

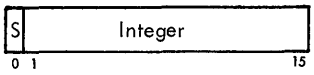
The largest positive number consists of a sign bit of 0 with all 1's in the integer field, whereas the largest negative number consists of a sign bit of 1 with all 0's in the integer field:

	S 1 ← Integer → 31	Decimal Number
Largest positive number:	0 1111111 11111111	= +2,147,483,647
Smallest positive number:	0 0000000 00000000	= 0
Smallest negative number:	1 1111111 11111111	= -1
Largest negative number:	1 0000000 00000000	= -2,147,483,648

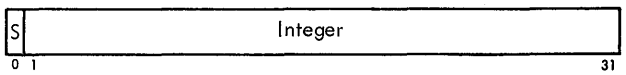
Formats

Fixed-point numbers occur in 16-bit halfword, 32-bit word, or 64-bit doubleword operands. Bit 0 is the sign bit, and the remaining bits make up the integer:

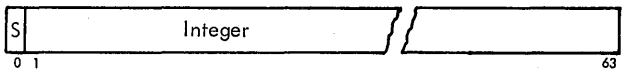
Halfword operand



Word operand



Doubleword operand



In LS, fixed-point operands are a word long. In some operations, such as multiply, divide, and shift, the operand may be a doubleword. The doubleword operands are located in a pair of adjacent 32-bit GPR's and are addressed by an even address that refers to the leftmost (lower-

addressed) register of the pair. In this case, the sign-bit position of the rightmost register *may* contain an operand bit instead of a sign bit. The sign-bit position of the leftmost register must *always* contain a sign bit.

In main storage, fixed-point data may be a halfword, a word, or a doubleword. This data must be located on integral storage boundaries for these units of information. When a halfword operand is fetched from main storage, it is extended to a full word. The original signed integer occupies bits 16 through 31, and is operated on as a full word. When the operand is extended to a full word, bits 0 through 15 assume the state of the original sign bit, now in bit 16.

Floating-Point Data

- Scientific and engineering calculations require that very small and very large numbers be represented.
- Scientific notation uses powers of 10 to simplify calculations with high and low magnitude numbers.
- Floating-point instructions operate upon data that uses powers of 16 to represent numerical quantities.
- Quantity expressed by floating-point number is product of signed hex fraction and 16 raised to power designated by exponent.

When performing calculations for scientific and engineering work, very small or very large numbers must sometimes be represented. Consider the problem of a nuclear physicist who wants to write an equation that contains the value for the mass of a subatomic particle. If he wrote the number as a decimal fraction, he would have to put down a decimal point followed by more than 20 zeros and a few numerals. At the opposite extreme, an astronomer may be calculating distances between objects that may be millions of millions of miles apart.

To overcome the pointless effort of having to write many zeros when working with such numbers, a mathematical method using powers of 10 is used. This method is called scientific notation. For example, in scientific notation, the mass of an electron can be written as 9.107 x 10^{-28} grams; and the astronomical distance of one light year can be written as 5.878 x 10^{12} miles.

Consider the parts of a number represented in scientific notation. The example of the mass of an electron, for instance, can be divided into three distinct parts: (1) a signed mixed number (+9.107) multiplied by, (2) 10 raised to the power designated by, (3) a signed exponent (−28). By changing the position of the decimal point and adjusting the exponent to compensate for the change, the number can also be written as a fraction times a power of 10 (+.9107 x 10^{-27}), or an integer times a power of 10 (+9107 x 10^{-31}).

For scientific and engineering applications where quantities may be of the magnitudes mentioned above, the CPU has instructions for handling them. These instructions, called floating-point instructions, manipulate data in a manner similar to scientific notation. However, because the CPU is primarily a binary machine in which numbers can be easily worked upon in hex (four-bit) units, a quantity is represented as a hex number times a power of 16 rather than as a decimal number times a power of 10. Except for this difference in base, floating-point notation is similar to scientific notation; the same rules of algebra apply to powers of 16 as to powers of 10.

Number Representation

- Fraction represents number expressed in hex digits.
- Characteristic specifies exponent to which 16 is raised.
- Characteristic is expressed in excess 64 notation; range is −64 to +63.
- Radix point is to left of high-order hex digit.
- True zero result yields positive sign.

A floating-point number contains the same components as a number written in scientific notation. However, due to the nature of the computer, the format is different and certain rules are imposed upon the way a floating-point number may be represented. The number to be multiplied by a power of 16 is a hex fraction with a fixed length, and the 16 is understood rather than shown. Therefore, as represented in the CPU, a floating-point number consists of a sign, which is the sign of the fraction, a signed exponent, called a characteristic, and a hex fraction. The quantity expressed by this number is the product of the fraction and 16 raised to the power designated by the characteristic (exponent).

The fraction of a floating-point number is expressed in hex digits. The radix point (representing the base 16) of the fraction is assumed to be immediately to the left of the high-order fraction digit. To provide the proper magnitude for a floating-point number, the fraction is considered to be multiplied by a power of 16 (fraction x 16^n power). The characteristic (bits 1–7) indicates the power (exponent). Bit 0 designates the sign of the fraction; it is a 0 if the fraction is a positive number and a 1 if the fraction is negative. Both positive and negative quantities are in true form, with the difference indicated by the sign.

The exponent may also be either a positive or negative number. For example, −.A8 x 16^{-2} is an example of a floating-point number with a negative fraction and a negative exponent. Therefore, to represent both positive and negative exponents, ***excess 64 notation*** is used. Excess 64 notation simply means that ***+64 (+40 hex) is added to the true exponent*** and the value obtained is used as the

characteristic. Therefore, the characteristic varies around a base of 64; i.e., an exponent of 0 is represented by a characteristic of 64, a positive exponent is represented by a characteristic greater than 64, and a negative exponent is represented by a characteristic less than 64. In the example just given, for instance, −2 is the exponent. Adding +64 to −2 yields +62, which is the value of the characteristic in excess 64 notation.

Performing the same calculation in binary gives the following results:

```
Bits   0  1  2  3  4  5  6  7

       S  1  1  1  1  1  1  0     2's complement of 2 (-2)
       S  1  0  0  0  0  0  0     +64 (+40 hex)
      ---------------------------------------
       S  0  1  1  1  1  1  0  = +62 (+3E hex)
       ↙
    Carry
```

If the exponent were +2 (positive exponent), adding +64 yields +66, the characteristic value placed into bits 1–7. In binary, the addition is as follows:

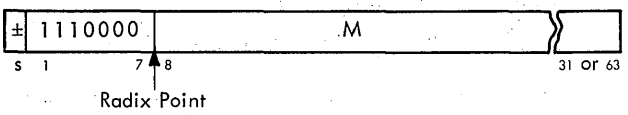
```
Bits   0  1  2  3  4  5  6  7

       S  0  0  0  0  0  1  0     +2 exponent
       S  1  0  0  0  0  0  0     +64 (40 hex)
      ---------------------------------------
       S  1  0  0  0  0  1  0  = +66 (42 hex)
```

Note in these examples that a negative exponent in excess 64 notation caused bit 1 (high-order bit of the characteristic) to be a 0, and a positive exponent caused bit 1 to be a 1. This rule holds true for the range of positive and negative exponents, as can be seen in Table 1-2. The table also shows that because only seven binary bits (1–7) are available to represent the characteristic in floating-point format, the most negative exponent that can be expressed is −64 and is represented by an all-zero characteristic. The most positive exponent is +63, and is represented by all 1's (7F hex). Midpoint between these two extremes is a 0 exponent, which is represented by a +64 (+40 hex) characteristic.

Table 1-2. Characteristic Notation

Excess 64 Notation			
Binary	Decimal	Hex	Exponent
0000000	0	0	-64
0000001	1	1	-63
↓	↓	↓	↓
0111111	63	3F	-1
1000000	64	40	0
1000001	65	41	+1
↓	↓	↓	↓
1111110	126	7E	62
1111111	127	7F	63

For another example of converting an exponent to an excess 64 characteristic, assume the value of $\pm M \times 16^{48}$ must be stated in excess 64 notation. The characteristic of the fraction then becomes 48 + 64 = 112 (0110000 + 1000000 = 1110000). The floating-point number thus takes the following form:



To illustrate how numbers are represented in floating-point format, assume that the decimal number 149.25 is to be converted to a floating-point short operand. This conversion is accomplished as follows:

1. The number is separated into a decimal integer and a decimal fraction:

   149.25 = 149 plus 0.25

2. The decimal integer is converted to its hex representation:

   $149._{10} = 95_{16}$

3. The decimal fraction is converted to its hex representation:

   $0.25_{10} = 0.4_{16}$

4. The integral and fractional parts are combined and expressed as a fraction times a power of 16 (exponent):

   $95.4_{16} = (0.954 \times 16^{2})_{16}$

5. The characteristic is developed from the exponent and converted to binary:

   Excess 64 + exponent = characteristic
   64 + 2 = 66 = 1000010

6. The fraction is converted to binary and grouped hexadecimally:

   $.954_{16} = 1001\ 0101\ 0100$

7. The characteristic and the fraction are placed in the short precision format; the sign position contains the sign of the fraction:

S	Characteristic	Fraction
0	1000010	1001 0101 0100 0000 0000 0000

Other examples follow:

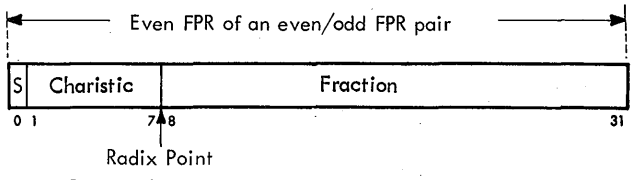
Number	Powers of 16	S	Charistic	Fraction
1.0	$= +1/16 \times 16^{1}$	= 0	1000001	0001 0000 ≀≀ 0000
0.5	$= +8/16 \times 16^{0}$	= 0	1000000	1000 0000 ≀≀ 0000
1/64	$= +4/16 \times 16^{-1}$	= 0	0111111	0100 0000 ≀≀ 0000
0.0	$= +0 \times 16^{-64}$	= 0	0000000	0000 0000 ≀≀ 0000
-15.0	$= -15/16 \times 16^{1}$	= 1	1000001	1111 0000 ≀≀ 0000
2×10^{-78}	$= +1/16 \times 16^{-64}$	= 0	0000000	0001 0000 ≀≀ 0000
7×10^{75}	$= (1-16^{-6}) \times 16^{63}$	= 0	1111111	1111 1111 ≀≀ 1111

Formats

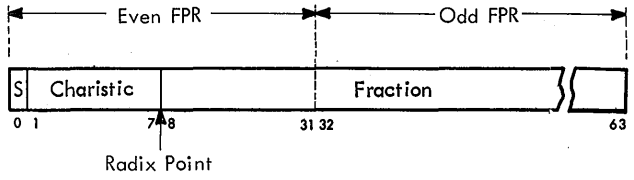
- Data format consists of 1-bit sign, 7-bit characteristic, and 24- or 56-bit fraction.
- Results are 32 bits (short operand) or 64 bits (long operand) long.
- Multiply product is always 64 bits.
- Guard digit is retained.

Floating-point data is represented in the CPU in one of two fixed-length formats, depending upon whether a full-word short operand or a doubleword long operand is desired. Both formats may be used in main storage and in the eight LS FPR's used exclusively by floating-point instructions. The data formats for short and long operands are:

Short Operand



Long Operand



For both formats, the first bit position is the sign bit and the subsequent seven bit positions constitute the characteristic. The following 24 or 56 bits represent the fraction; short operand fractions are 24 bits or 6 hex digits; long operand fractions are 56 bits or 14 hex digits.

When short operands are specified, the results are usually 32-bit floating-point words; the odd FPR of the even/odd pair of FPR's does not participate in the operation and remains unchanged. However, in multiply instructions, the product occupies two FPR's (64 bits).

When long operands are specified, all operands and results are 64-bit floating-point doublewords.

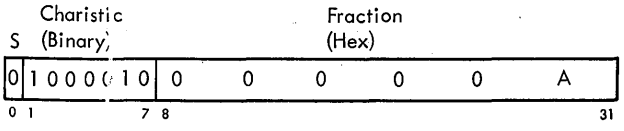
Although final results have 6 or 14 hex fraction digits, intermediate results in addition, subtraction, and compare operations may extend to 7 or 15 fraction digits. This extra digit, called the *guard digit*, occurs when one of the fractions is shifted right (as part of the characteristic equalization process that occurs during execution of add-type floating-point instructions; see Chapter 3, Section 3, *Add, Subtract, and Compare*). The guard digit increases the accuracy of the final result if normalization occurs. In normalization, the fraction is shifted left until a significant digit appears in the high-order digit position of the fraction; thus the guard digit becomes part of the 6 or 14 hex digits of the final result. This saving of the guard digit becomes especially significant where the high-order 6 or 14 digits of the intermediate result are all zeros.

Normalization

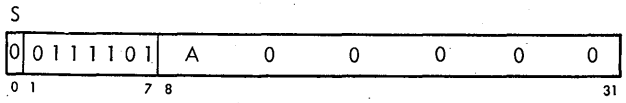
- Normalized fraction has nonzero, high-order hex digit; unnormalized fraction has one or more leading hex zeros.
- Characteristic is adjusted on normalization cycles.
- Postnormalization is normalization of final result.
- Prenormalization is normalization before result computation.
- Results are shifted right if fraction overflow occurs.

A quantity can be represented with the greatest precision by a floating-point number of a given fraction length when that number is normalized. A normalized floating-point number has a nonzero high-order hex fraction digit. If one or more high-order fraction digits are zero, the number is said to be unnormalized. Normalization consists of shifting the fraction left until the high-order hex digit is nonzero and reducing the characteristic by the number of hex digits shifted. A zero fraction cannot be normalized, and its associated characteristic therefore remains unchanged when normalization is called for.

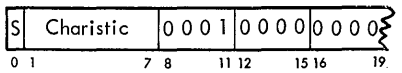
An example of an unnormalized floating-point number in numerical terms is $.00000A_{16} \times 16^{2}$. To convert this number to its normalized form, the number must be shifted five hex digits to the left and, because five shifts are necessary, five is subtracted from the exponent. The result is $.A00000_{16} \times 16^{-3}$. In the CPU, the original number would have the following format:



After normalization, the format would be:



Because normalization applies to hex digits, the three high-order bits of a normalized number may be zero. For example, if the high-order digit of a fraction is a hex 1, normalization will not occur although normalization is specified and bits 8–10 = 000:



Floating-point operations are performed with or without normalization. Addition and subtraction may be specified either way depending upon the instruction op code. The multiply, divide, and halve instructions always specify normalization. The load, compare, and store instructions specify unnormalized results. Normalization usually occurs when the intermediate arithmetic result is changed to the final result. This function is called *postnormalization*. In multiplication and division, the operands are normalized before the arithmetic process. This function is called *prenormalization*.

When an operation is performed without normalization, high-order zeros in the result fraction are not eliminated. In both normalized and unnormalized operations, the initial operands need not be in normalized form.

Decimal Data

- Operands and results are located in main storage.
- VFL is 1–16 bytes.
- Four-bit BCD digits are packed two to a byte for arithmetic.
- Unpacked (zoned) format is used for transmitting data to I/O devices.
- Pack and Unpack instructions are provided.

Decimal instructions are designed for operations requiring few computational steps between the source input and the documented output. Processing of this type is frequently found in commercial applications. Because of the limited number of arithmetic operations performed on each item of data, radix conversion from decimal to binary and back to decimal is not justified, and the use of registers for intermediate results yields no advantage over storage-to-storage processing. Hence, in the Model 65, decimal instructions are provided and both operands and results are located in main storage. Decimal instructions include addition, subtraction, multiplication, division, and comparison.

Decimal arithmetic operates on data in the packed format, in which two four-bit BCD digits are packed two to a byte. They appear in fields of variable length (from 1 to 16 bytes) and are accompanied by a sign in the rightmost four bits of the low-order byte. The use of packed digits within a byte and of variable-length fields within storage results in efficient use of storage and in increased arithmetic performance.

Decimal numbers may also appear in a zoned format for use with I/O devices operating in that format. The zoned format is not used in decimal arithmetic operations, but only for transmitting data to the I/O device. Instructions are provided for packing and unpacking decimal numbers so that they may be changed from the zoned (unpacked) to the packed format and vice versa.

Processing takes place right to left between main storage locations, except in the divide operation which is processed left to right. All decimal instructions use the two-address SS format. Each address specifies the leftmost byte of an operand. Associated with this address is a length field, indicating the number of additional bytes that the operand extends beyond the first byte.

Number Representation

Numbers are represented as right-aligned true integers with a plus or minus sign. Decimal digits 0–9 are represented in the four-bit BCD form by 0000 through 1001, respectively. Codes 1010–1111 (10–15) are not valid as digits and are reserved for sign codes: 1010, 1100, 1110 and 1111 represent a plus; 1011 and 1101 represent a minus.

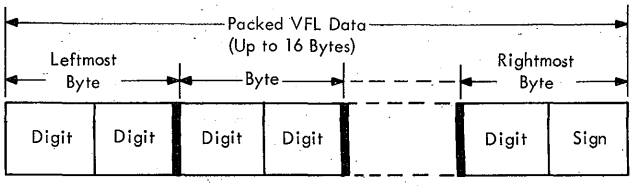
Digit	Code	Sign	Code
0	0000	+	1010
1	0001	–	1011
2	0010	+	1100
3	0011	–	1101
4	0100	+	1110
5	0101	+	1111
6	0110		
7	0111		
8	1000		
9	1001		

All valid sign codes are recognized in decimal operations; however, the appropriate sign codes (and zone codes for the Unpack instruction) generated during the operation depend on the character set specified by PSW(12). If PSW(12) = 0, EBCDIC is selected, and code 1100 is generated for a plus sign, code 1101 is generated for a minus sign, and code 1111 is generated for a zone. If PSW(12) = 1, USASCII-8 is selected, and code 1010 is generated for a plus sign, code 1011 is generated for a minus sign, and code 0101 is generated for a zone.

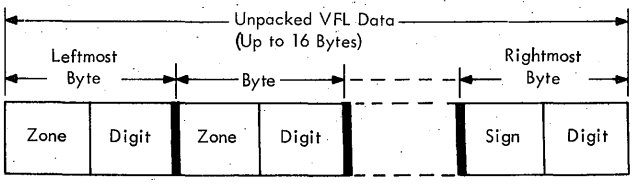
Formats

Decimal operands reside in main storage only. The operand field length may range from a minimum of one byte to a maximum of 16 bytes. The operands need not occupy the entire field length but are always right-aligned in the field; i.e., the sign of the operand is always in the rightmost byte of the specified field. This rightmost byte contains the lowest-order operand digit and the operand sign. All decimal instructions (except Divide) process the operands from low order to high order, or from right to left between main storage locations.

Data may be in the packed or unpacked (zoned) format. In the packed format, two four-bit BCD digits are placed adjacently in an eight-bit byte, except for the rightmost (low-order) byte of the field. In the low-order byte, a four-bit sign (sign of the decimal number) is placed to the right of the decimal digit.



In the unpacked or zoned format, a decimal digit normally occupies the four low-order bits of a byte, the *numeric*. The four high-order bits of a byte are called the *zone*. An exception is the rightmost byte in the field, where the sign of the decimal number occupies the zone position.



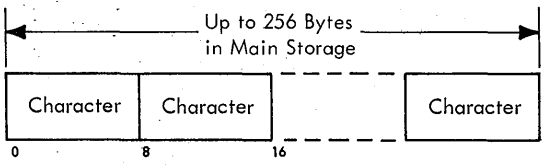
Logical Data

- Data is fixed-length or VFL.
- One byte of immediate data is held in some instruction formats.

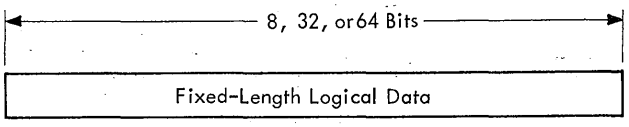
The logical instructions provide for moving, comparing, bit testing, bit connecting, translating, editing, and shifting operations. Except for the editing instructions, data is not treated as numbers. Editing converts packed decimal digits into alphanumeric characters; the digits, signs, and zones are recognized and generated as for decimal instructions.

Data resides in main storage or in LS, or is contained in the instruction format. The data may be a single byte, a word, a doubleword, or variable in length. When two operands participate in the operation, they have equal length, except in the editing instructions. The data format depends on the type of operation performed:

1. In storage-to-storage operations, data has a VFL format, starting at any byte address and continuing for a maximum of 256 bytes; it is processed left to right.



2. In storage-to-register operations, the main storage data may be either a word or a byte. The word must be located on a word boundary; that is, the low-order two bits of its address must be 0's. Data in GPR's normally occupies all 32 bits. Bits are treated uniformly, and no distinction is made between sign and numeric bits. In a few operations, only the low-order eight bits of the register participate, leaving the remaining 24 bits unchanged. In some shift operations, 64 bits of an even/odd pair of GPR's participate.



3. In operations which introduce data directly from the SI format instruction as an immediate operand, data is restricted to an eight-bit byte. Only one byte may be introduced per instruction, and only one byte from main storage takes part in the operation.

www.IBM360.info

This section discusses the supervisor program, the eight program states, the program status word, interruptions and exceptional conditions, and the initiation and control of I/O operations.

CONTROL PROGRAM

Because internal processing speeds of data processing systems have increased without a corresponding reduction in the time required by the operator to load programs and to manually insert data, the setup time has become a more significant factor in system operation. To reduce this setup time, during which the system is idle, control programs were devised to control execution of problem programs. The simplest control program, which shares main storage with the problem programs, controls the loading of problem programs; the problem programs handle their own I/O operations (Figure 1-13, A). Operation is as follows:

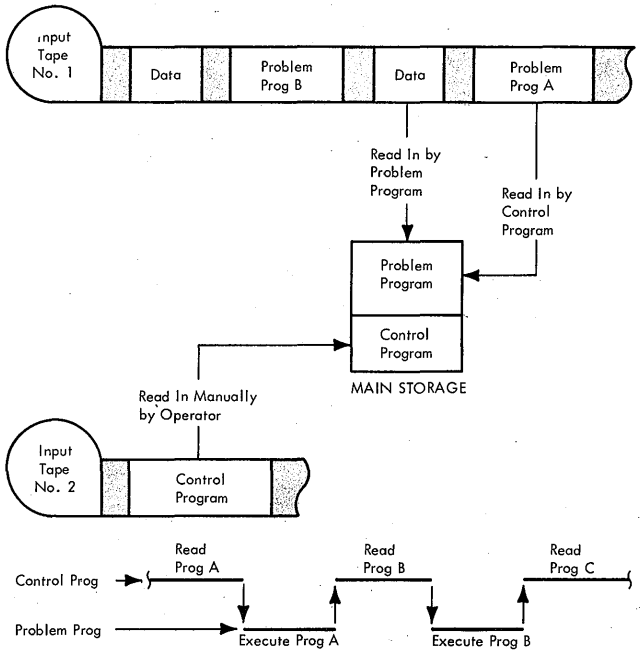
1. An input device is prepared containing the problem programs and their associated data.
2. The operator loads the control program into main storage.
3. The control program loads in the first problem program and then passes control to the problem program.
4. The problem program reads in its data and performs its assigned task.
5. When the problem program is finished, it passes control back to the control program.
6. The control program then loads in the next problem program and passes control to it.
7. This operation continues until all programs are executed.

Notice three things about the use of a control program in the preceding example:

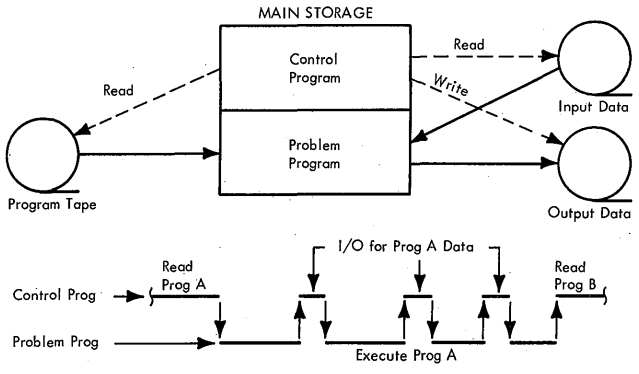
1. The system never halts between jobs.
2. The control program remains in main storage as the problem programs are executed.
3. The control program serves only as a link between jobs; its only function is to bring in a new problem program as each job is finished.

The simple control program discussed above has limited functions. Other functions can be included, such as initiation and control of I/O operations. Because the problem program is designed mainly to process data, the read and write operations necessary to transfer data between the I/O devices and main storage can be handled by the control program. Each operation to be handled by the control program may consist of many instructions. Besides telling the I/O device to start, for example, the instructions check for error conditions and I/O device status.

In this function, control passes back and forth between the problem and control programs *during* execution of the problem program (Figure 1-13, B). The control program not only reads in the problem programs but also handles



A. Simple Control Program - Loads Problem Programs



B. Expanded Control Program - Loads Problem Programs, and Handles I/O Operations for Problem Program Data

Figure 1-13. Examples of Control Program Functions

the I/O data operation during execution of the problem program. The *problem* program transfers control to the *control* program whenever an I/O operation is necessary.

The control program can be given other functions as well. However, the more functions a control program has, the more main storage space it requires. This problem is solved by placing into main storage only those key parts of the control program that ensure continuous, coordinated operation of the system. This portion is called the *supervisor*. The remaining parts of the control program are placed on a high-speed, direct-access I/O device, such as a disk storage unit, and are brought into main storage as they are required to perform a specific function.

In the System/360, the supervisor is the control center of the operating system. Its primary function is to perform a variety of services for other parts of the system including problem programs. It coordinates and controls the performance of these services to yield efficient and coordinated use of the physical and programming facilities of the system. The supervisor prevents programs and routines that are run on the system from interfering with one another and with operation of the control program. This control is accomplished, in part, through its use of *privileged* instructions, such as storage protection and I/O instructions, which can be executed only by the control program.

A service performed by the supervisor may be specifically requested by a program, such as a request for storage space, or it may be a service that is automatically provided when a contingency occurs, such as attempting to recover from an error condition. Among the services the supervisor may provide are:

1. Allocating main storage space required by programs during their execution.
2. Sharing areas of main storage among routines that need not be in main storage at the same time.
3. Loading programs into main storage.
4. Controlling the concurrent execution of programs and routines.
5. Scheduling and controlling I/O operations.
6. Providing standard procedures that assist in diagnosing exceptional conditions, such as underflow in floating-point arithmetic operations.
7. Keeping a running log of machine check and I/O errors for CE diagnostic use.

One of the reasons why a control program is used in the System/360 is to eliminate machine idle time. Realizing this, the designers of System/360 did not incorporate a halt instruction. Therefore, a problem program cannot issue a halt instruction when it is finished but passes control to the supervisor by means of a Supervisor Call instruction. Machine idle time is further reduced in another way. A machine check (such as an even number of bits in a byte) or a program check (such as locating a halfword operand on an odd-byte address) does *not* cause an error stop but causes an automatic branch to the supervisor, unless instructed by the programmer to be ignored. These automatic branches to the supervisor are called *interruptions*. That is, the current sequence of problem instructions is interrupted and an automatic branch (interruption) is taken to a new sequence of control instructions.

When an interruption occurs, the interruption-handling routine of the supervisor stores the status of the CPU and fetches information with which to control the CPU while it handles the interruption. The status of the CPU is contained in a doubleword called the *old* program status word (PSW). Bits of this old PSW show the cause of the interruption and the program state of the CPU at the time of the interruption. There are eight paired states, Problem/Supervisor, Operating/Stopped, Running/Wait, and Interruptable/Masked, all of which, except Stopped, are defined in the PSW. The following paragraphs define the program states and discuss the PSW before the discussion of interruptions is continued.

PROGRAM STATES

The eight program states which determine the overall CPU status differ in the way they affect the CPU functions and in the way their status is indicated. Refer to Table 1-3 for pertinent information about the program states.

Problem/Supervisor

In the Problem state, all I/O, protection, and direct control instructions are invalid as well as the Load PSW, Set System Mask, and Diagnose instructions. These instructions are called *privileged* instructions. A privileged instruction encountered in the Problem state constitutes a privileged-operation interruption and interrupts the operation. In the Supervisor state, all instructions are valid.

The CPU is switched between the Problem and Supervisor states by changing PSW(15). When PSW(15) is a 1, the CPU is in the Problem state; when a 0, the CPU is in the Supervisor state. This bit can be changed only by introducing a new PSW. Thus, the status switching for Problem/Supervisor state may be performed by an interruption operation or by a Load PSW instruction containing a new PSW with the desired value in bit 15. Because the Load PSW instruction is a privileged instruction, the CPU must be in the Supervisor state before the switch. The CPU status can also be changed between Problem and Supervisor states by issuing a Supervisor Call instruction or an initial program load (IPL). The Supervisor Call instruction causes an interruption which will load new PSW data. This new PSW data may change the state of the CPU. Similarly, the IPL introduces a new PSW. The new PSW may introduce the Problem or Supervisor state, regardless of the preceding CPU state.

Table 1-3. Program States

State	Comments
Problem	Load PSW, Set System Mask, and Diagnose instructions, and all I/O, storage protection, and direct control instructions are invalid. These instructions are termed privileged instructions.
Supervisor	All instructions are valid.
Operating	CPU processes instructions (if not in Wait state) and interruptions (if not masked off). Entered by: 1. Depressing START on system control panel. 2. Starting an IPL operation.
Stopped	Instructions and interruptions are not processed; interruptions remain pending. Execution of program is not affected by stopping CPU. Entered by: 1. Depressing STOP on system control panel. 2. Detecting equality on an address-compare-stop operation. 3. Completing one instruction when in instruction-step mode. 4. Turning power on or following a system reset.
Running	Instruction processing proceeds in normal manner.
Wait	No instructions are processed, and main storage is not addressed. The CPU waits for an I/O or external interruption to occur before executing further instructions, or for an IPL operation.
Interruptable	Interruptions are accepted.
Masked	System and machine-check interruptions remain pending, and program interruptions are ignored.

Operating/Stopped

When the CPU is in the Stopped state, instructions and interruptions are not executed. When the CPU is in the Operating state, instructions are executed as long as the CPU is not also in the Wait state. Interruptions are taken if they are not masked off. Manual operations, such as load PSW, can be used *only* when in the Stopped state. A change in the Stopped/Operating states can occur only by manual intervention or by machine malfunction. No instruction or interruption can start or stop the CPU. The CPU is placed in the Stopped state when STOP on the system control panel is depressed, detecting equality on an address-compare-stop operation, completing one instruction when in instruction-step mode, and after power is turned on or following a system reset, except during IPL. The CPU is placed in the Operating state when START on the system control panel is depressed and when an IPL is started.

Changing from the Operating state to the Stopped state occurs at the end of instruction execution and before the start of the next instruction execution. When the CPU is in the Wait state, the change from Operating to Stopped occurs immediately. All interruptions pending and not masked off are taken while the CPU is still in the Operating state. The interruptions cause an old PSW to be stored and a new PSW to be fetched before entering the Stopped state. Once the CPU is in the Stopped state, interruptions are no longer taken but remain pending.

Running/Wait

In the Wait state, no instructions are processed and main storage is not addressed. In the Running state, instruction fetching and execution proceed in the normal manner. The CPU status is switched between the Wait and Running states by PSW(14). When PSW(14) is a 1, the CPU is in the Wait state; when a 0, in the Running state. This bit can only be changed by introducing a new PSW. Thus, switching from the Running to the Wait state may be achieved by the privileged instruction Load PSW, by an interruption such as given by a Supervisor Call instruction, or by an IPL. Switching from the Wait to the Running state may be achieved by an I/O or external interruption or by an IPL. The new PSW may introduce the Wait or Running state regardless of the preceding CPU state.

Interruptable/Masked

The Masked/Interruptable state of the CPU is determined by the system mask bits [PSW(0–7)], the machine-check mask bit [PSW(13)], and the program mask bits [PSW(36–39)]. If a mask bit is a 1, the associated interruption is accepted; if it is a 0, system and machine-check interruptions remain pending and program interruptions are ignored. The PSW bits and interruptions that will occur if the bit is active are listed in Table 1-4.

The Masked/Interruptable state of the CPU is switched by changing the mask bits in the PSW. The program mask may be changed separately by the Set Program Mask instruction, and the system mask may be changed separately by the Set System Mask instruction. The machine-check mask bit can be changed only by introducing an entirely new PSW, as in the Problem/Supervisor and Wait/Running states. Thus, a change in the entire masked status may be achieved by the privileged instruction Load PSW, by an interruption such as for the Supervisor Call instruction, or by an IPL. Regardless of the preceding program state, the new PSW may introduce a new mask status.

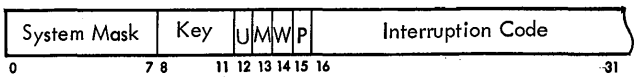
Table 1-4. PSW Interruption Mask Bit Designation

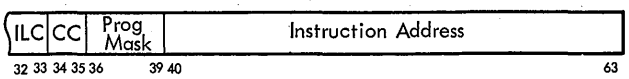
PSW Bit	Interruption
System mask	
0	Multiplexer channel (0)
1	Selector channel 1
2	Selector channel 2
3	Selector channel 3
4	Selector channel 4
5	Selector channel 5
6	Selector channel 6
7	Timer, INTERRUPT pushbutton, or external signals 2–7
Machine-check mask	
13	Machine check
Program mask	
36	Fixed-point overflow
37	Decimal overflow
38	Exponent underflow (floating-point)
39	Significance (floating-point)

PROGRAM STATUS WORD

- PSW provides program status and controls system operation.
- Active PSW is termed *current* PSW.
- Interruption causes current PSW to be stored into old PSW location.
- Load PSW instruction, interruption, PSW restart, and IPL introduce new PSW into CPU.
- Set Program Mask and Set System Mask instructions load new PSW mask fields into CPU.

A doubleword, the program status word (PSW), contains the information required for proper program execution. In general, the PSW controls instruction sequencing, and holds and indicates the status of the system in relation to the program being executed. The PSW has the following format:





Bits 0–7, System Mask. Associated with I/O and external interruptions as follows:

System Mask Bit	Interruption Source
0	Multiplexer channel (0)
1	Selector channel 1
2	Selector channel 2
3	Selector channel 3
4	Selector channel 4
5	Selector channel 5
6	Selector channel 6
7	Timer, INTERRUPT pushbutton, or external signals 2–7

When a system mask bit is a 1, the associated source can interrupt the CPU; when a 0, the interruption remains pending.

Bits 8–11, Key. Contain the CPU storage protection key. The key is matched with the storage key whenever data is stored, or whenever data is fetched from a location that is fetch-protected.

Bit 12, U (USASCII-8). Affects decimal operations only. When a 1 and in unpacked format, decimal digits are represented by USASCII-8. When a 0, EBCDIC is specified.

Bit 13, M (Machine-Check Mask). When a 1, a machine check causes the CPU to log out its status to main storage and to take a machine-check interruption. If the machine-check mask is a 0, the machine check remains pending. As a maintenance aid, the CPU CHECK switch on the system control panel can modify the machine-check mask bit functions.

Bit 14, W (Wait State). When a 1, CPU is in the Wait state; instructions are not executed until an external or I/O interruption or an IPL occurs. When a 0, the CPU is in the Running state.

Bit 15, P (Problem State). When a 1, the CPU is in the Problem state. When a 0, the CPU is in the Supervisor state.

Bits 16–31, Interruption Code. Identify the cause, purpose, or source of the interruption. (Has no meaning in a new PSW.)

Bits 32 and 33, ILC (Instruction Length Code). Indicate the length, in halfwords, of the last processed instruction. This code is predictable only for most program and supervisor call interruptions. For I/O and external interruptions, the interruption is not caused by the last interpreted instruction, and the code is not predictable for these instructions. For machine-check interruptions, the setting of the code may be affected by the

malfunction and, therefore, is unpredictable. A code of 0, used only for program interruptions, indicates that the instruction address in the PSW is *not* the location of the instruction following the instruction that caused the program interruption.

Bits 34 and 35, CC (Condition Code). Contain the condition code that reflects the result of most arithmetic, logical, or I/O instructions. Each of these operations can set the code to any one of four states, and the conditional-branch instructions can specify the state to be used as the criterion for branching. For example, the CC may reflect such conditions as nonzero, overflow, and underflow. Once set, the CC remains unchanged until modified by an instruction that reflects a different code. The two bits of the CC provide for four possible binary settings: 00, 01, 10, and 11. This manual refers to the CC's as 0, 1, 2, and 3. (The CC has no meaning in a new PSW.)

Bits 36–39, Program Mask. Each bit is associated with a program interruption as follows:

Program Mask Bit	Program Interruption
36	Fixed-point overflow
37	Decimal overflow
38	Exponent underflow (floating-point)
39	Significance (floating-point)

When a program mask bit is a 1, the associated program interruption results in an interruption; when 0, the interruption is lost.

Bits 40–63, Instruction Address. Specify the leftmost byte address of the next instruction.

The active or controlling PSW is called the *current* PSW. The information making up the current PSW is held in triggers and registers in the CPU and is constantly being updated as instructions are executed. (The instruction-address field of the PSW, for example, is held in the instruction counter and is updated to give the address of the next instruction to be executed.) When an interruption is taken, the PSW is assembled in the ST register and is transferred to a fixed location in main storage corresponding to the interruption. (The stored PSW is called the *old* PSW.) In this way, the program can preserve for subsequent analysis the status of the CPU at the time of interruption. To complete the interruption, a new PSW is introduced into the CPU from another unique main storage location, and the instruction at the specified location is fetched.

In certain circumstances, the entire PSW is loaded into the CPU; in others, only part of it. As explained in the preceding paragraph, when the CPU is interrupted, the entire current PSW is stored and an entire new PSW is loaded. The state of the CPU may be changed by executing the Load PSW instruction, which introduces a new PSW. New program mask and system mask bits may be specified by altering the corresponding fields in the PSW through the Set Program Mask and Set System Mask instructions, respectively. The Set Program Mask instruction also changes the condition code.

INTERRUPTIONS AND EXCEPTIONAL CONDITIONS

- Five classes of interruptions are recognized:
  Machine check
  Program
  Supervisor call
  External
  I/O
- Eight exceptional conditions are recognized:
  Timer
  CPU store in progress
  Manual control stop
  Manual control wait
  Manual control repeat
  Program store compare
  Invalid instruction address test
  Q-register refill
- Program interruption suppresses, terminates, or allows completion of instruction being processed.
- Machine-check interruption terminates instruction being processed.
- Other interruptions and exceptional conditions allow completion of instruction being executed.

In general terms, the purpose of the interrupt area in the CPU is twofold: (1) to recognize defined interruptions and exceptional conditions that may arise while the system is in operation, and (2) to control the action that is subsequently initiated. The distinction between interruptions and exceptional conditions is that interruptions always allow a new instruction flow to be entered whereas exceptional conditions do not, unless an associated interruption condition is also present.

Five classes of interruptions have been defined for the System/360:

1. Machine Check. Caused by the machine-checking circuits detecting a machine malfunction. The program automatically enters a diagnostic routine if the machine-check mask bit is a 1. The malfunction is indicated in the logout data.
2. Program. Caused by unusual conditions (such as incorrect operands or programming errors) encountered in a program. The exact error is indicated in the Interrupt Code triggers.
3. Supervisor Call. Caused by the program issuing an instruction to turn control over to the supervisor program. The reason for the call is shown in E(8–15).

4. External. Caused by the interval timer going from positive to negative, or by depressing the INTERRUPT pushbutton on the system control panel, or, if the Direct Control feature is installed, by an external device requiring attention.
5. I/O. Caused by an I/O device ending an operation or otherwise needing attention, or by operator intervention at an I/O device. Identification of the device and channel causing the interruption is signalled to the CPU. The status of the device and channel is stored in a fixed main storage location.

Table 1-5 lists for each interruption the interruption code, mask bits, ILC, and how the instruction execution is finished.

Unlike interruptions, exceptional conditions vary between models of the System/360. In the Model 65, eight exceptional conditions are recognized:

1. Timer. Caused by positive swing of line frequency.
2. CPU Store In Progress. Caused by a store operation in progress while an interruption or other exceptional condition is to be serviced or a Load PSW instruction is to be performed.
3. Manual Control Stop. Caused by a need to switch the program from Operating to Stopped state.
4. Manual Control Wait. Caused by the program switching from Running to Wait state.
5. Manual Control Repeat. Caused by the repeat instruction operation, a maintenance aid using manual controls.
6. Program Store Compare. Caused by possibly altering a main storage location already prefetched as an instruction, or by returning to the instruction flow following an Execute instruction.
7. Invalid Instruction Address Test. Caused when the next instruction to be performed is at an invalid, protected, or incorrectly specified main storage location.
8. Q-Register Refill. Caused by the need to delay processing of the next instruction when the Q-register is being refilled.

Interruptions and exceptional conditions are always processed after ending the current instruction and before starting the next instruction. The finishing of the current instruction is influenced by the cause of the interruption; the instruction may be completed, terminated, or suppressed.

In instruction completion, the results of the operation are stored and the condition code (CC) is set as for normal instruction operation, although the result may be altered by the interruption condition. In instruction termination, execution of the instruction has started and some of the data in the registers has changed. The results may therefore be incorrect. The CC may also be incorrect. The instruction may or may not be allowed to continue to completion; the final results should not be used for further computation. In instruction suppression, execution of the instruction is halted by forcing an end-op cycle. Results are not stored, and the CC is not changed.

In some cases, the instruction is almost finished before the interruption condition is detected. For these cases, the CPU blocks any change to the CC and prevents storing of the unpredictable final result by changing the store operation to a fetch operation.

A program interruption may suppress, terminate, or allow completion of the instruction being processed; the particular effect depends on the instruction being performed and the cause of the condition. A machine-check interruption terminates the instruction (or any other action being performed) immediately upon detection of the malfunction. The other interruptions and exceptional conditions detected during instruction execution allow the instruction to be completed.

As an instruction ends and before the next instruction starts, the CPU interrupt logic examines triggers to determine whether an interruption or exceptional condition has arisen. If an interruption operation is to be performed, an exit is made from the current instruction flow and, following the interruption operation, a new instruction flow can be entered. If an exceptional condition operation is to be performed, the current instruction flow is usually not left but only delayed for the time it takes to perform the operation.

During execution of an instruction, several interruptive events may occur. For example, the instruction may give rise to a program interruption, an external interruption signal may be received, a timer exceptional condition may occur, a machine malfunction may occur, and an I/O interruption request may be made. Instead of the program interruption, a supervisor call interruption might occur; these two interruptions cannot occur together, because they are mutually exclusive. When simultaneous interruptions occur, the competing interruptions are serviced in a fixed order of priority.

The priority of interruptions and exceptional conditions is:

1. Timer exceptional condition.
2. CPU store in progress exceptional condition.
3. Machine-check interruption.
4. Program interruption.
5. Supervisor call interruption.
6. External interruption.
7. I/O interruption.
8. Manual control stop exceptional condition.
9. Manual control wait exceptional condition.
10. Manual control repeat exceptional condition.
11. Program store compare exceptional condition.
12. Invalid instruction address test exceptional condition.
13. Q-register refill exceptional condition.

Table 1-5. Interruptions

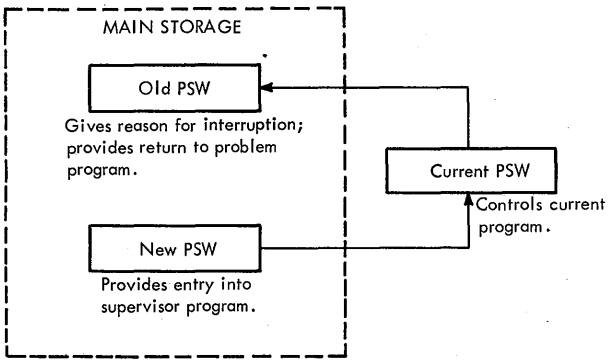
Interruption	Interruption Code (Old PSW Bits 16–31)	PSW Mask Bit	ILC	How Instruction Execution Is Finished
Machine Check	00000000 00000000	13	u	Terminated
Program				
Operation	00000000 00000001	–	1, 2, 3	Suppressed
Privileged operation	00000000 00000010	–	1, 2	Suppressed
Execute	00000000 00000011	–	2	Suppressed
Protection	00000000 00000100	–	0, 2, 3	Suppressed or Terminated
Addressing	00000000 00000101	–	0, 1, 2, 3	Suppressed or Terminated
Specification	00000000 00000110	–	1, 2, 3	Suppressed
Data	00000000 00000111	–	2, 3	Terminated
Fixed-point overflow	00000000 00001000	36	1, 2	Completed
Fixed-point divide	00000000 00001001	–	1, 2	Suppressed or Completed
Decimal overflow	00000000 00001010	37	3	Completed
Decimal divide	00000000 00001011	–	3	Suppressed
Exponent overflow	00000000 00001100	–	1, 2	Terminated
Exponent underflow	00000000 00001101	38	1, 2	Completed
Significance	00000000 00001110	39	1, 2	Completed
Floating-point divide	00000000 00001111	–	1, 2	Suppressed
Multisystem	00000000 00010010	–	2	Suppressed
Supervisor Call	00000000 rrrrrrrr	–	1	Completed
External				
External signal 7	00000000 ppppppp1	7	u	Completed
External signal 6	00000000 pppppp1 p	7	u	Completed
External signal 5	00000000 ppppp1 pp	7	u	Completed
External signal 4	00000000 pppp1 ppp	7	u	Completed
External signal 3	00000000 ppp1 pppp	7	u	Completed
External signal 2	00000000 pp1 ppppp	7	u	Completed
INTERRUPT pushbutton	00000000 p1 pppppp	7	u	Completed
Timer	00000000 1ppppppp	7	u	Completed
I/O				
Multiplexer channel (0)	00000000 aaaaaaaa	0	u	Completed
Selector channel 1	00000001 aaaaaaaa	1	u	Completed
Selector channel 2	00000010 aaaaaaaa	2	u	Completed
Selector channel 3	00000011 aaaaaaaa	3	u	Completed
Selector channel 4	00000100 aaaaaaaa	4	u	Completed
Selector channel 5	00000101 aaaaaaaa	5	u	Completed
Selector channel 6	00000110 aaaaaaaa	6	u	Completed

Notes:
u: Unpredictable; E(0,1)
r: R1 and R2 fields of Supervisor Call instruction
p: Set if pending before PSW(7) is set to a 1.
a: I/O device address

Interruptions

- Each class of interruption has two related PSW's: old and new.
- Old and new PSW's have fixed, unique main storage addresses.
- Current PSW is stored into old address.
- 64 is added to old PSW address to get new PSW address.
- New PSW becomes current PSW.
- There are 12 unconditional interruptions.
- 20 interruptions have associated PSW mask bits. If mask bit is 1, take interruption; if 0, ignore interruption.

An interruption replaces the entire *current* PSW. It is placed into a fixed location in main storage, and becomes the *old* PSW (Figure 1-14). This old PSW gives the reason for the interruption and also provides a return to the interrupted program. A *new* PSW is then fetched from a fixed location in main storage and becomes the current PSW. This new PSW provides an entry into the correct interruption-handling routine in the supervisor program:



Each of the five classes of interruptions has its own distinct locations for new and old PSW's, as follows:

Interruption	Decimal Address		Hex Address	
	Old PSW	New PSW	Old PSW	New PSW
External	24	88	18	58
Supervisor Call	32	96	20	60
Program	40	104	28	68
Machine Check	48	112	30	70
I/O	56	120	38	78

Thus, for example, a machine-check interruption causes the current PSW to be placed into location 48 and a new PSW to be brought out from location 112. Note that these locations are all divisible by 8 because they contain doublewords, and that the location of any new PSW is 64 higher than its corresponding old PSW location.

The five classes of interruptions tell the supervisor only the *general* reason for the interruption. For instance, the fact that the new PSW was brought out of location 104 tells the supervisor that the interruption was caused by a program check. The supervisor still needs to know what type of program check occurred. This is the function of the interruption code, which is set into the current PSW automatically by the CPU logic before the PSW is stored. By examining the interruption code in bits 16–31 of the old PSW, the program-check routine in the supervisor program can tell specifically whether it was a specification, addressing, or some other type of error. In the case of I/O interruptions, the interruption code tells the supervisor which channel and I/O unit caused the I/O interruption. In the case of a machine-check interruption, the supervisor must inspect the logout data to learn the specific malfunction.

After the interruption has been processed by the supervisor, the last instruction can be a Load PSW. This instruction causes the old PSW to once again become the current PSW, and the CPU is back in the problem program.

The load PSW instruction may also be used to: (1) allow the supervisor to change the current PSW, and (2) load the PSW for a new problem program after the program has been read into main storage by the supervisor.

Interruption Masking

Sometimes it is not desirable to allow an interruption. This condition becomes apparent when I/O interruptions are considered (Figure 1-15). Assume an I/O operation is completed, resulting in an I/O interruption. The current PSW is stored as the old PSW to give the supervisor the reason for (or the source of) the interruption. This old PSW also enables the supervisor to return to the interrupted problem program. A new PSW is then brought out of storage and becomes the current PSW which indicates the first instruction of the I/O interruption-handling routine. If at this time a second I/O interruption, perhaps caused by operator intervention at an I/O device of another channel, were allowed, the old PSW stored as a result of the first I/O interruption would be lost. The supervisor can prevent this second I/O interruption from being accepted until it has processed the first I/O interruption by means of mask bits in the new PSW.

Twenty of the 32 interruptions for the five classes can be masked off by associated mask bits in the PSW. (The

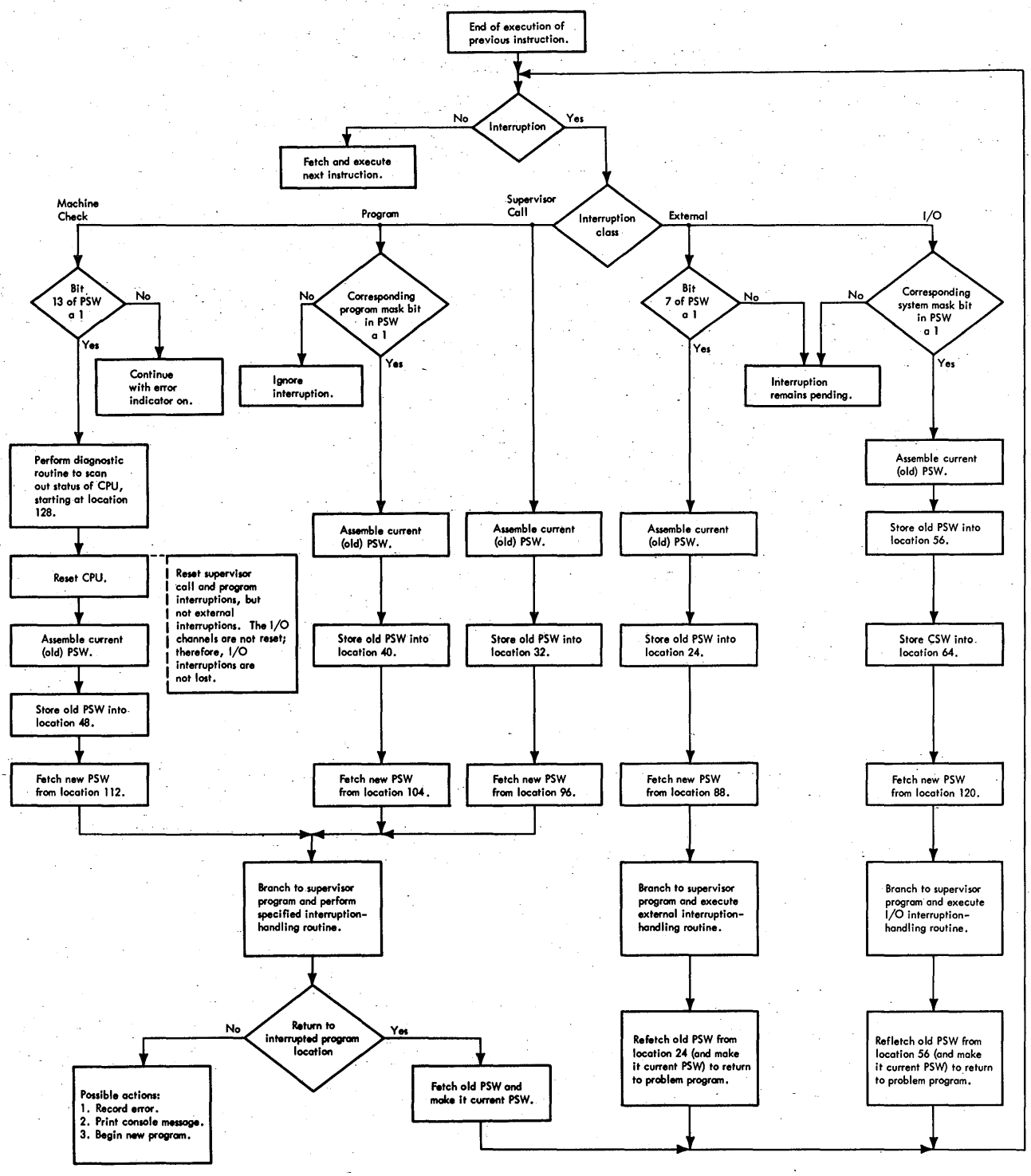


Figure 1-14. Action Taken When Single Interruption Occurs

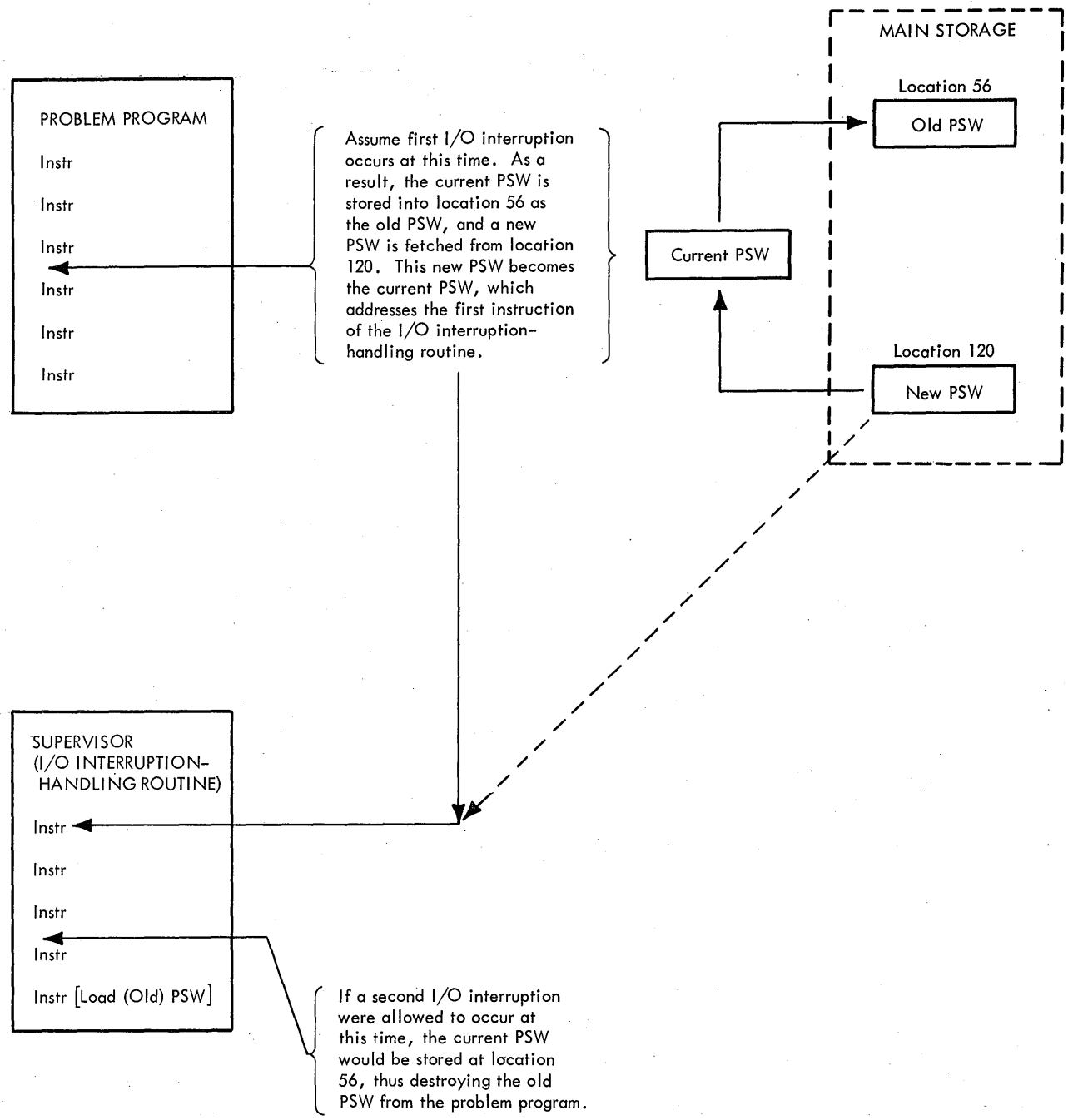


Figure 1-15. Example of Need for Interruption Masking

remaining 12 interruptions are unconditional; they are always taken.) If the corresponding mask bit is a 1, the interruption is taken; if a 0, the interruption is ignored or remains pending. External and I/O interruptions may be masked off by the system mask field of the PSW; machine-check interruptions may be masked off by the machine-check mask bit; 4 of the 15 program interruptions may be masked off by the program mask field.

System Mask Field. The system mask field consists of eight bits [PSW(0–7)] which can be used selectively or collectively to mask all I/O and external interruptions:

System Mask Bit	Interruption Source
0	Multiplexer channel (0)
1	Selector channel 1
2	Selector channel 2
3	Selector channel 3
4	Selector channel 4
5	Selector channel 5
6	Selector channel 6
7	Timer, INTERRUPT pushbutton, or external signals 2–7

To prevent an I/O or external interruption before the first interruption has been processed, the system mask of the new PSW should contain zeros. When a system mask bit is a 0, the associated I/O or external interruption remains pending.

The system mask field may be changed by introducing a new PSW, or it may be changed separately by the Set System Mask instruction.

Machine-Check Mask Bit. The machine-check mask bit [PSW(13)] controls the acceptance of a machine-check interruption. If this bit is a 0, machine-check interruptions are ignored and remain pending. If this bit is a 1, machine-check interruptions are taken, depending on the position of the CPU CHECK switch on the system control panel. If this switch is in the PROC (normal) position, the CPU stops and the status is logged into main storage; a machine-check interruption then takes place. If the CPU CHECK switch is in the DSBL (disable) position, the CPU does not stop upon detection of a machine check and no logout or interruption takes place. If the switch is in the STOP position, the CPU stops upon detection of a machine check, but there is no logout of data and no interruption takes place.

The usual mode of operation is to have the CPU CHECK switch set to the PROC position and PSW(13) set to a 1.

The machine check mask bit can be changed only by introducing a new PSW.

Program Mask Field. The program mask field consists of four bits [PSW(36–39)], each of which is associated with a program check:

Program Mask Bit	Program Interruption
36	Fixed-point overflow
37	Decimal overflow
38	Exponent underflow (floating-point)
39	Significance (floating-point)

When a program mask bit is a 1, the associated program check results in an interruption; when a 0, no interruption occurs and the condition does not remain pending.

The program mask field may be changed by introducing a new PSW, or it may be changed separately by the Set Program Mask instruction.

Instruction Address Determination

- PSW holds address of instruction to be executed next.
- Interruption, if any, occurs during instruction execution.
- For program and supervisor call interruptions, instruction address less ILC gives address of preceding instruction during which interruption occurred.

As stated earlier, the instruction address portion of the current PSW is used by the interruption operation to fetch an instruction. Once the instruction has been fetched, the instruction address portion of the PSW is updated to address the next instruction. Interruptions are serviced only *after* an instruction is finished. Therefore, the instruction address portion of the old PSW does not contain the address of the last instruction executed, but rather contains the address of the next instruction that *would* have been executed if the interruption had not occurred. For some interruptions, it is desirable to locate the instruction during which the interruption occurred. To obtain this location, the instruction address portion of the old PSW must be decremented by the supervisor. To do so, the supervisor must know the length of the last instruction executed. This

length, in halfwords, is given by the instruction length code (ILC) of the old PSW (bits 32 and 33). The ILC for the five formats is as follows:

ILC	PSW Bits 32 and 33	Op Code Positions 0 and 1	Instruction Length (in Halfwords)	Format
0	00	–	Not available	–
1	01	00	1	RR
2	10	01	2	RX
2	10	10	2	RS and SI
3	11	11	3	SS

The ILC is predictable only for program and supervisor-call interruptions. For I/O and external interruptions, the interruption is not caused by the preceding instruction, and the ILC is therefore not predictable for these instructions. For machine-check interruptions, the ILC setting may be affected by the malfunction and, therefore, may be incorrect. Therefore, the instruction causing these interruptions, if any, must be located by other means.

For supervisor-call interruptions, the ILC is 1, indicating the halfword length of the Supervisor Call instruction. For program interruptions, an ILC of 1, 2, or 3 indicates the length of the instruction in halfwords. The ILC of 0 is reserved for a storage protection condition detected after completion of the instruction that caused the violation; this condition is called a *late* storage protection check and results is an *indefinite* program interruption.

Note that for a program interruption caused by an incorrect branch address, the address determined from the instruction address and ILC is the branch address and not the address of the branch instruction.

Machine-Check Interruption

The machine-check interruption provides a means for recognizing a machine malfunction. The following malfunctions cause a machine check:

1. ROS word parity check.
2. Parallel adder full-sum parity check.
3. Parallel adder half-sum parity check.
4. Serial adder full-sum parity check.
5. Serial adder half-sum parity check.
6. E-register parity check.
7. Multiply/divide logic error.
8. Storage address check to CPU.
9. Storage data check to CPU.
10. System hang (Multisystem feature only).

Each malfunction sets a specific trigger which, in turn, sets the 'error' trigger and lights the PROC CHK indicator on the system control panel and the CHK SUMM indicator on roller switch 3 (position 1, bit 20).

Acceptance of a machine-check interruption depends upon the state of the machine-check mask bit, PSW(13), and upon the position of the CPU CHECK switch on the system control panel.

If the machine-check mask bit is a 1 and the CPU CHECK switch is in the PROC position, machine-check interruptions are taken. The current instruction is terminated, and a diagnostic routine called *logout* is initiated. The status of the CPU is logged out into the permanent main storage area starting at location 128 (decimal) and extending through location 295, a total of 22 doublewords. A machine-check interruption then takes place; the old PSW is stored at main storage location 48 (decimal) with an interruption code of 0 and a new PSW is fetched from main storage location 112.

If the machine-check mask bit is a 0 and the CPU CHECK switch is in the PROC position, the interruption remains pending and the CPU attempts to complete the current instruction and to proceed with the next instruction.

If the CPU CHECK switch is in the DSBL (disable) position, the interruption is ignored regardless of the state of the machine-check mask bit.

If the switch is in the STOP position, the CPU stops upon detection of the machine check regardless of the state of the mask bit.

Following an emergency power-off, power-on, or system-reset operation, incorrect parity may exist in storage and registers. Unless new information is loaded, a machine-check condition may occur erroneously. Once storage and registers are cleared, a machine-check interruption can be caused only by a machine malfunction and not by invalid data or instructions.

Program Interruptions

Program interruptions result from improper specifications or unusual conditions arising during the processing of data or instructions. There are 15 program interruptions, 4 of which may be masked off by associated bits in the PSW program mask field; the remaining 11 are unconditionally taken. If the associated mask bit is a 0, the interruption is ignored and does not remain pending.

The program interruption causes the old PSW to be stored into main storage location 40 (decimal) and a new PSW to be fetched from location 104. Interruption code bits 28–31 identify the cause of the interruption. Bits 16–27, the remainder of the interruption code, are made zeros.

Four Interrupt Code triggers determine the code to be set into bits 28–31 of the interruption code. The outputs of the four triggers are encoded to give the 15 possible codes. The specific trigger(s) to be set is determined by a combination of a micro-order, the op code, and data conditions. The 'program interrupt' latch is set by the

Interrupt Code triggers. A brief description of the 15 program interruptions follows:

1. Operation. An invalid op code is detected. The instruction is suppressed; the ILC is 1, 2, or 3.
2. Privileged Operation. A privileged instruction is encountered in the Problem state. The instruction is suppressed; the ILC is 1 or 2.
3. Execute. The subject instruction of an Execute instruction is another Execute instruction. The instruction is suppressed; the ILC is 2.
4. Protection. The storage key of a main storage location does not match the storage protection key in the PSW. For a store protection violation, the instruction is suppressed, except for Store Multiple, Read Direct, and Test and Set instructions and VFL operations, which are terminated. The ILC is 0, 2, or 3. For a fetch-protection violation, the instruction is terminated except for the Execute instruction which is suppressed; the protected information is not loaded into an addressable register or moved to another storage location. The ILC is 0, 2, or 3. In the case of a violation caused by an I/O operation, data transmission is terminated in such a way that the protected information is not recorded on an output medium. The violation is indicated in the channel status word stored as a result of the operation. The ILC is 2 or 3.
5. Addressing. An address specifies any part of data, an instruction, or a control word outside the available main storage. In most cases, the instruction is terminated. Data in main storage remains unchanged, except when designated by valid addresses. The instruction is suppressed for the Convert to Decimal, Diagnose, and Execute instructions, for certain SI-format instructions, and for certain store instructions. The ILC is 0, 1, 2, or 3.
6. Specification. A specification program interruption is caused by any one of six conditions:
   a. A data, instruction, or control-word address does not specify an integral boundary for the unit of information.
   b. The R1 field of an instruction specifies an odd LS register address instead of an even register address for a pair of GPR's that contains a doubleword operand.
   c. An FPR address other than 0, 2, 4, or 6 is specified.
   d. The multiplier or divisor in decimal arithmetic operations exceeds 15 digits and sign.
   e. The divisor in decimal division is equal to or greater than the dividend, or the multiplier in decimal multiplication is equal to or greater than the multiplicand.
   f. The block address specified in Set Storage Key or Insert Storage Key instructions does not have the four low-order bits set to zero.

   The instruction is suppressed; the ILC is 1, 2, or 3.
7. Data. A data interruption is caused by any one of three conditions:
   a. The sign or digit codes of operands in decimal arithmetic, editing, or convert-to-binary operations are incorrect.
   b. Fields in decimal arithmetic operations overlap incorrectly.
   c. The decimal multiplicand has too many high-order significant digits. (The number of high-order zeros in the multiplicand must at least equal the multiplier field.)

   The instruction is terminated; the ILC is 2 or 3.
8. Fixed-Point Overflow. A high-order carry occurs or high-order significant bits are lost in fixed-point add, subtract, shift, or load operations. The instruction is completed by ignoring the overflow; the ILC is 1 or 2. The interruption may be masked off by making the fixed-point overflow mask bit [PSW(36)] a 0; the interruption is then ignored.
9. Fixed-Point Divide. The quotient exceeds the register size in a fixed-point divide instruction, or the result of a Convert to Binary instruction exceeds 31 bits. The divide instruction is suppressed, and the Convert to Binary instruction is completed by ignoring the extra bits. The ILC is 1 or 2.
10. Decimal Overflow. The destination field is too small to contain the result field in a decimal arithmetic operation. The instruction is completed by ignoring the overflow information; the ILC is 3. The interruption may be masked off by making the decimal-overflow mask bit [PSW(37)] a 0; the interruption is then ignored.
11. Decimal Divide. The quotient exceeds the specified data field size in a decimal division. The instruction is suppressed; the ILC is 3.
12. Exponent Overflow. The result exponent (characteristic) of a floating-point addition, subtraction, multiplication, or division overflows, and the result fraction is not zero. The operation is completed by making the characteristic 128 smaller than the true result; the sign and fraction remain unchanged. The ILC is 1 or 2.
13. Exponent Underflow. The result of a floating-point addition, subtraction, multiplication, or division underflows, and the result fraction is not zero. A program interruption occurs if the exponent-underflow mask bit [PSW(38)] is a 1. The operation is completed by replacing the result with a true zero if the mask bit is 0. If the mask bit is 1, the characteristic is made 128 larger than the true result, and the sign and fraction remain unchanged. The ILC is 1 or 2.
14. Significance. The result of a floating-point addition or subtraction has an all-zero fraction. The instruction is completed; the ILC is 1 or 2. The interruption may be masked off by making the significance mask bit [PSW(39)] a 0; the interruption is then ignored.
15. Floating-Point Divide. A division is attempted by a

floating-point divisor with a zero fraction. The instruction is suppressed; the ILC is 1 or 2.

Note: If the Multisystem feature is installed, a fifth Interrupt Code trigger is added for interruption code bit 27. A multisystem program interruption code of 18 (decimal) occurs if the Set System Mask instruction is encountered when in Multisystem mode. The instruction is suppressed; the ILC is 2.

Supervisor-Call Interruption

The supervisor-call interruption is used by the problem program to pass control to the supervisor program. To do so, the problem program executes the Supervisor Call instruction which, in turn, causes a supervisor-call interruption. The interruption is unconditionally taken; there is no associated mask bit in the PSW. The old PSW is stored at main storage location 32 (decimal), and the new PSW is fetched from location 96. The R1 and R2 fields (bit positions 8–15) of the Supervisor Call instruction become the low-order half (bits 24–31) of the interruption code of the old PSW. These bits may be used to convey a message (for example, the reason for the call) from the calling program to the supervisor program. Bits 16–23 of the interruption code are made zero. The ILC is 1, indicating the one-halfword length of the Supervisor Call instruction.

External Interruptions

The external interruption enables the CPU to respond to signals from external units, from the INTERRUPT pushbutton on the system control panel, and from the interval timer. An external interruption may occur at any time, and interruptions from the different sources may occur simultaneously. Interruptions are kept pending until accepted by the CPU. When several interruptions from one or more sources are pending, only one interruption is taken servicing all pending external interruptions.

An external interruption can occur only when system mask bit 7 is a 1 and after the current instruction is executed. The interruption causes the old PSW to be stored at main storage location 24 (decimal) and a new PSW to be fetched from location 88. The source of the interruption is identified by bit positions 24–31 of the interruption code; bits 16–23 are made zero.

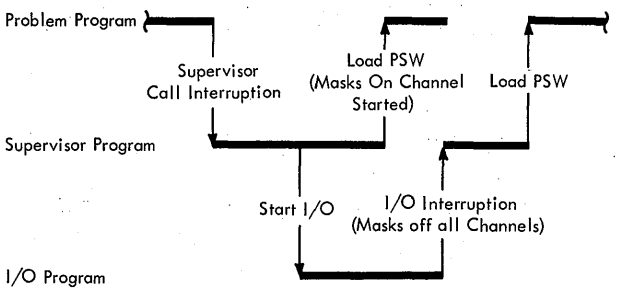
A brief discussion of the three external interruptions follows:

1. External Signals. Available only if the Direct Control feature is installed. May be masked off by depressing DISABLE DIRECT CONTROL on the system control panel or by setting bit 7 of the system mask field to a 0. Six signal-in lines, representing external signals 2–7, are connected to the CPU to cause an external interruption. The specific external signal causing the interruption is identified by a unique bit position in positions 26–31 of the interruption code. For example, external signal 7 is identified by a 1 in bit position 31, external signal 6 by a 1 in bit position 30, and so on. The Multisystem feature, if installed, assigns specific meanings to external signals 2 and 3. (Refer to Chapter 4, Section 2.)
2. INTERRUPT Pushbutton. If bit 7 of the system mask field is a 1, depressing INTERRUPT on the system control panel causes an external interruption. Bit position 25 of the interruption code is set to a 1.
3. Timer. If bit 7 of the system mask field is a 1 and the interval timer value changes from positive to negative, an external interruption occurs. Bit position 24 of the interruption code is set to a 1. The interruption is initiated as the timer count proceeds from a positive number, including zero, to a negative number. The interval timer is updated (decremented) 60 times a second or 50 times a second, depending on the line frequency. It is possible that, after an interruption is initiated, the timer may have been updated several times before the CPU is actually interrupted, depending on the instruction being executed and on the state of the mask bit. The operation of the timer is controlled by the DISABLE INTERVAL TIMER switch on the system control panel.

I/O Interruptions

The I/O interruption enables the CPU to respond to signals from I/O devices. A request for an I/O interruption may occur at any time, and more than one request may occur at the same time. The requests remain pending in the I/O area of the system until accepted by the CPU. Priority is established among requests so that only one interruption request is honored at a time. The order of priority is channel 0, then channels 1–6. Note that this priority is different from the priority the BCU establishes for servicing I/O storage requests.

I/O interruptions generally occur at the end of an I/O operation. Most I/O operations are overlapped with processing; an I/O interruption, therefore, is an efficient way of signalling the supervisor that the I/O operation is finished. After the I/O interruption-handling routine in the supervisor is finished, control is passed to the problem program:



An I/O interruption can occur only if the associated mask bit in the system mask field of the PSW is a 1. The

I/O interruption causes the old PSW to be stored at main storage location 56 (decimal) and the channel status word associated with the interruption to be stored at location 64. The new PSW is fetched from location 120. Bit positions 21–23 and 24–31 of the interruption code identify the channel and the I/O device, respectively, causing the interruption. Bit positions 16–20 are made zero.

Exceptional Conditions

Exceptional conditions are processed by the CPU after finishing the instruction in progress. After processing the exceptional condition, the instruction flow might be continued, stopped, repeated, or left, depending on the specific exceptional condition. An exceptional condition operation does not change the current PSW, nor does the PSW contain mask bits for exceptional conditions.

Timer Exceptional Condition

When the interval timer must be stepped (decremented), a timer exceptional condition is generated (recognized by a trigger that is set by a positive swing of the line frequency). When the instruction in progress is finished, the CPU generates main storage address 80 (decimal), the location of the timer value. The timer value is fetched, stepped, and returned to location 80. If the timer value is stepped from a positive value to a negative value, a timer external interruption is processed next. Otherwise, processing continues with the next instruction. The timer is controlled by the DISABLE INTERVAL TIMER switch on the system control panel.

CPU Store in Progress Exceptional Condition

When the next instruction must be delayed to prevent the subsequent program interruption from being indefinite, a CPU store in progress exceptional condition is generated. This condition is recognized if the storing of data into main storage would overlap the processing of the Load PSW instruction or the handling of an interruption or manual control or program store compare exceptional condition. A two-cycle CPU loop is entered to insure that any "late" storage protection check will not result in an indefinite program interruption under these conditions. If a protection check occurs, a program interruption is processed next. Otherwise, processing continues with the next instruction.

Manual Control Stop Exceptional Condition

The manual control stop exceptional condition arises when the program is to be switched from the Running to the Stopped state. The CPU enters a "stop loop", during which no instructions are processed and all interruptions are kept pending. Only the following pushbuttons are recognized: STORE, DISPLAY, SET IC, START, ROS TRANSFER, and PSW RESTART.

Manual Control Wait Exceptional Condition

The manual control wait exceptional condition arises when PSW(14) is set to a 1. CPU clock signals are inhibited. Processing continues when an external or I/O interruption or an IPL operation is initiated.

Manual Control Repeat Exceptional Condition

The manual control repeat exceptional condition arises when the repeat instruction operation (a maintenance aid) is begun and, if the REPEAT INSN (instruction) switch on the system control panel is in the SINGLE position, before each subsequent repetition of the specified instruction.

Program Store Compare Exceptional Condition

A program store compare exceptional condition results if the next instruction to be processed must again be fetched into the Q-, R- and E-registers. This need occurs after processing the Execute instruction and after some store operations. Although the instruction to be executed next is held in the instruction buffer (Q-register), it is modified in its main storage location only. Therefore, to have the correct version of the instruction in the Q-register, the next instruction is refetched before it is executed.

Invalid Instruction Address Test Exceptional Condition

The previously discussed exceptional conditions and interruptions result from unusual conditions occurring during the execution of an instruction. To identify the instruction, its address and op code are preserved by inhibiting the fetching of the next instruction until the microprogram routine that handles the interruption or exceptional condition is ended.

In an invalid instruction address test exceptional condition, however, it is the address of the *next* instruction that is invalid, protected, or incorrectly specified. Processing of the instruction is allowed to start; thus the address of the previous instruction is replaced with an erroneous address. The appropriate Interrupt Code trigger(s) is set and a program interruption is processed next.

Q-Register Refill Exceptional Condition

The Q-register refill exceptional condition arises when Q-register (instruction buffer) refilling conflicts with the start of the next instruction. The exceptional condition delays processing of the next instruction by one (or two) cycles.

CONTROL OF I/O OPERATIONS

The following paragraphs: define an I/O operation; discuss how I/O operations are controlled by instructions, commands, orders, and control words; and illustrate how the I/O system works, using the Start I/O instruction as an example.

Instructions, Commands, and Orders

- CPU executes instructions.
- Channels execute commands.
- Control units and devices execute orders.
- Five operations are available:
  Write
  Read
  Read backward
  Control
  Sense

Input/output operations are initiated and controlled by three types of information: instructions, commands, and orders. Instructions are decoded and executed by the CPU and are part of the CPU program. Commands are decoded and executed by the channel, and initiate I/O operations such as reading and writing. Instructions and commands are fetched from main storage and are common to all types of devices. Orders specify functions peculiar to an I/O device, such as rewinding tape or spacing a line on a printer. Orders are contained in the control command; they are decoded and executed by the device.

The action in an I/O device initiated by a command is termed an *I/O operation*. Five I/O operations are available: write, read, read backward, control, and sense. The channel initiates the operation by executing the associated command.

The write command initiates a write operation at the device. Data from main storage is fetched in an ascending order of addresses and transferred to the device.

The read command initiates a read operation at the device. Data is read from the device in the same sequence as it was written by a write command. Data is placed into main storage in an ascending order of addresses.

The read-backward command initiates a read-backward operation at the device. Data is read from the device in a sequence opposite to that in writing. Data is placed into main storage in a descending order of addresses.

The control command contains information, termed *orders*, that controls the selected device. Orders are unique to the particular device in use and specify such functions as backspacing or rewinding magnetic tape. Orders are fetched from main storage in an ascending order of addresses and transferred to the device.

The sense command initiates a sense operation at the device. Data transferred during a sense operation provides information about unusual conditions detected during the last operation and the status of the device. Data is placed into main storage in an ascending order of addresses.

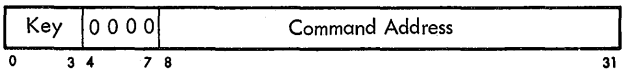
I/O Control Words

Three I/O control words are used during an I/O operation:

1. Channel Address Word (CAW), which initiates I/O sequencing.
2. Channel Command Word (CCW), which controls I/O operations and sequencing.
3. Channel Status Word (CSW), which indicates channel status.

Channel Address Word

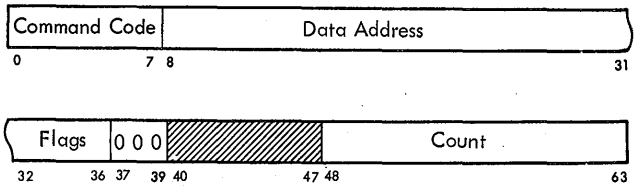
The CAW specifies the address of the first CCW associated with the Start I/O instruction. The CAW is assigned permanent main storage address 72 (decimal). The channel refers to the CAW only during execution of the Start I/O instruction. The pertinent information is stored in the channel, and the CPU program is free to change the contents of the CAW. The CAW has the following format:



Bits 0–3, Key. Specifies the storage protection key for all commands associated with the Start I/O instruction.
Bits 4–7. Must be all 0's.
Bits 8–31, Command Address. Designates location of the first CCW in main storage.

Channel Command Word

The CCW specifies the command to be executed and, for commands initiating I/O operations, designates the main-storage area associated with the operation and the action to be taken whenever data transfers to or from the main storage are completed. The CCW's can be located anywhere in main storage, and more than one can be associated with a Start I/O instruction. The channel refers to a CCW in main storage only once, whereupon the pertinent information is stored in the channel. The first CCW is fetched during execution of the Start I/O instruction. Each additional CCW is obtained when the operation has progressed to the point where the additional CCW is needed. The CCW has the following format:

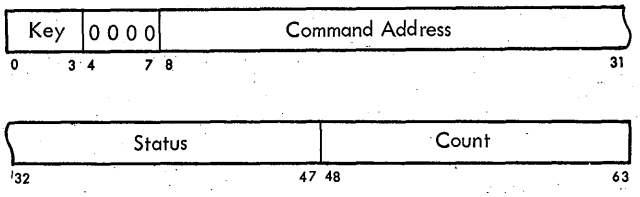


Bits 0–7, Command Code. Specifies I/O operation to be performed.
Bits 8–31, Data Address. Specifies location of an eight-bit byte in main storage; it is the first location referred to in the main storage area designated by the CCW.
Bits 32–36, Flags. Cause certain functions to be performed that modify the operation.

Bits 37–39. Must be all 0's for every CCW other than the CCW that specifies a transfer-in-channel operation.
Bits 40–47. Not used.
Bits 48–63, Count. Specifies the number of eight-bit byte locations in the main storage area designated by the data-address field in the CCW.

Channel Status Word

The CSW provides the program with the status of an I/O device or the condition under which an I/O operation has been finished. The CSW is formed, or parts of it are replaced, in the process of I/O interruptions and during execution of the Start I/O, Test I/O, and Halt I/O instructions. The CSW is placed into main storage location 64 (decimal) and is available to the program at this location until the next I/O interruption occurs or until another I/O instruction causes its contents to be replaced, whichever occurs first. The CSW has the following format:



Bits 0–3, Key. Contains the storage protection key that was used in the I/O operation initiated by the last Start I/O instruction.
Bits 4–7. Must be all 0's.
Bits 8–31, Command Address. Identifies the last CCW used.
Bits 32–47, Status. Identifies the conditions in the I/O device and channel that caused the CSW to be stored.
Bits 48–63, Count. Contains the residual count of the last CCW used.

I/O System Operation

- Start I/O instruction is given.
- Channel fetches CAW from main storage location 72 (decimal).
- CAW designates main storage address of first CCW.
- CCW specifies command and main storage area.
- Channel selects device. Start I/O instruction is terminated at this point.
- Channel controls operation and data transfers.
- Operation is terminated by two conditions: channel end and device end.

The CPU program initiates I/O operations by means of the Start I/O instruction. This instruction identifies the I/O device and causes the channel to fetch the CAW from main storage location 72 (decimal). The CAW designates the location in main storage from which the channel subsequently fetches the first CCW. The CCW specifies the command to be executed, the main storage area to be used, and the number of data bytes to be transferred, if any.

The channel attempts to select the device by sending the address of the device to all attached control units. Upon recognizing the address, the control unit associated with the addressed device connects itself logically to the channel. The channel subsequently sends the command code to the device, and the device responds by indicating whether it can execute the command.

At this time, execution of the Start I/O instruction is terminated, and the CPU continues with its program. The results of the attempt to initiate command execution are indicated in the PSW and, under certain conditions, by storing a portion of the CSW.

If the operation is initiated by the I/O device and its execution involves transfer of data, the channel responds to service requests from the device and assumes control of the operation. For operations that do not require transfer of data, the device signals the end of the operation immediately on receipt of the command code, and the channel is immediately available for a new I/O operation.

An I/O operation may involve transfer of data to or from one main-storage area, designated by a single CCW, or, when data chaining is specified, to or from a number of noncontiguous main-storage areas. In the latter case, a chain of CCW's is used in which each CCW designates an area in main storage for the continuation of the original command (operation).

Termination of the I/O operation normally is indicated by two conditions: channel end and device end. The channel-end condition indicates that the I/O device has received or provided all information associated with the operation and no longer needs channel facilities. The device-end condition indicates that the device has finished the operation.

Facilities are provided for the program to initiate execution of a chain of commands with a single Start I/O instruction. When command chaining is specified, the device-end condition causes the channel to fetch a new CCW that specifies a new operation at the device.

Conditions that initiate I/O interruptions are asynchronous with the activity in the CPU, and more than one interruption condition can occur at the same time. A priority has been established among the conditions so that only one interruption is processed at a time. The I/O interruption conditions are preserved in the I/O devices and channel until accepted by the CPU.

This section discusses: (1) the functional units (based on Diagram 3-2, FEMDM); and (2) instruction fetching and execution, the Universal instruction set by instruction class, and power considerations.

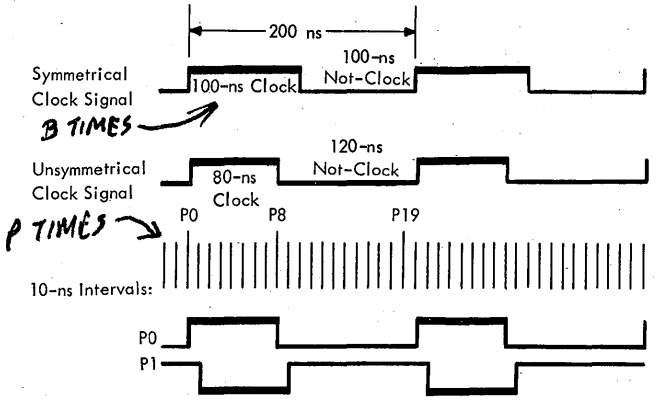
CONTROL

The 2065 CPU operates with a basic clock cycle period of 200 ns under control of ROS. The following paragraphs discuss: (1) CPU timing and data transfer; (2) ROS – what it is, how it controls the CPU, and its data flow; (3) the additional control provided by the PSW register.

CPU Timing

- Basic clock cycle period is 200 ns.
- Symmetrical clock signal consists of 100-ns clock portion and 100-ns not-clock portion.
- Unsymmetrical clock signal consists of 80-ns clock portion and 120-ns not-clock portion.

The basic CPU clock cycle period is 200 ns, divided into clock and not-clock portions. A clock signal generator provides a 5-megaHertz (5-mHz) symmetrical (100-ns/100-ns) clock signal. Two types of clock signal generators are used: a continuously running crystal-controlled oscillator in the Model G65, H65, and I65 CPU or a gated delay-line oscillator in the Model IH65 and J65 CPU. To provide additional time for CPU logic functions, the symmetrical clock signal is modified to give a 5-mHz unsymmetrical (80-ns/120-ns) clock signal. Finer intracycle control is obtained by dividing each of the two clock signals into 20 intervals of approximately 10 ns each. These intervals, named P0, P1, P2, . . . P19, are created by inverters which delay the signal by about 10 ns. Thus the notations P0, P1, P2, . . . P19 refer to signals which are inverted and are delayed 10 ns with respect to the previous signal:



The clock signals are distributed to logic gates A through E. Adjustable time delays within the logic gates synchronize the clock signals with a reference signal, thereby eliminating the various amounts of delay introduced by the distribution cables. The distribution of the clock signals to the CPU processing logic is stopped upon detection of an error or during scan, logout, single-cycle, and certain ROS operations. When the CPU clock signals are stopped, the BCU must continue to service the I/O channels and the scan logic must be operable. Therefore, the clock signals to these areas of the logic are not stopped, except under certain circumstances in the Model IH65 and J65 CPU.

Data Transfer

Data is transferred into a register, into an adder, and into and out of LS by gating signals controlled by ROS (Figure 1-16). Referring to Figure 1-16, note that data from PAL is always available at the A-register, B-register, and T-register, but is transferred only into the selected register by means of the corresponding gating signal from ROS. When gating data into an adder, timing considerations require the use of 'gate control' triggers; these triggers, which are set by the ROS decode logic, generate the required gating signal.

When transferring data into LS, the gating signal from ROS is combined with a signal from the LS addressing logic to develop a 'write LS' signal, which gates the data into LS. When transferring data from the LS, an address signal selects 1 of 24 LS registers, the contents of which are transferred to the LS bus. A second gating signal transfers the data from the LS bus to the S- or T-register.

Read-Only Storage

The CPU is controlled by ROS, a permanently recorded microprogram, supplemented by conventional control logic. A read-only storage is a storage device which contains information (1's and 0's) of a nondestructive nature. The 2065 CPU utilizes a capacitive read-only storage, in which bits are stored in the form of a capacitance between a fixed drive-plate pattern etched at right angles to a sense-plate pattern. Sense and drive plates are separated by a Mylar† film (approximately 1 mil thick), and the resulting sandwich is held together under pressure. A 1-signal is coupled from a drive line to one of a pair of sense lines, and a 0-signal is coupled to the other. Sense line outputs are detected in a differential amplifier which in turn feeds a latch. When decoded, the information in ROS controls gates to route data in the CPU. Access time is approximately 95 ns.

†Trademark of E. I. duPont de Nemours & Co. (Inc.)

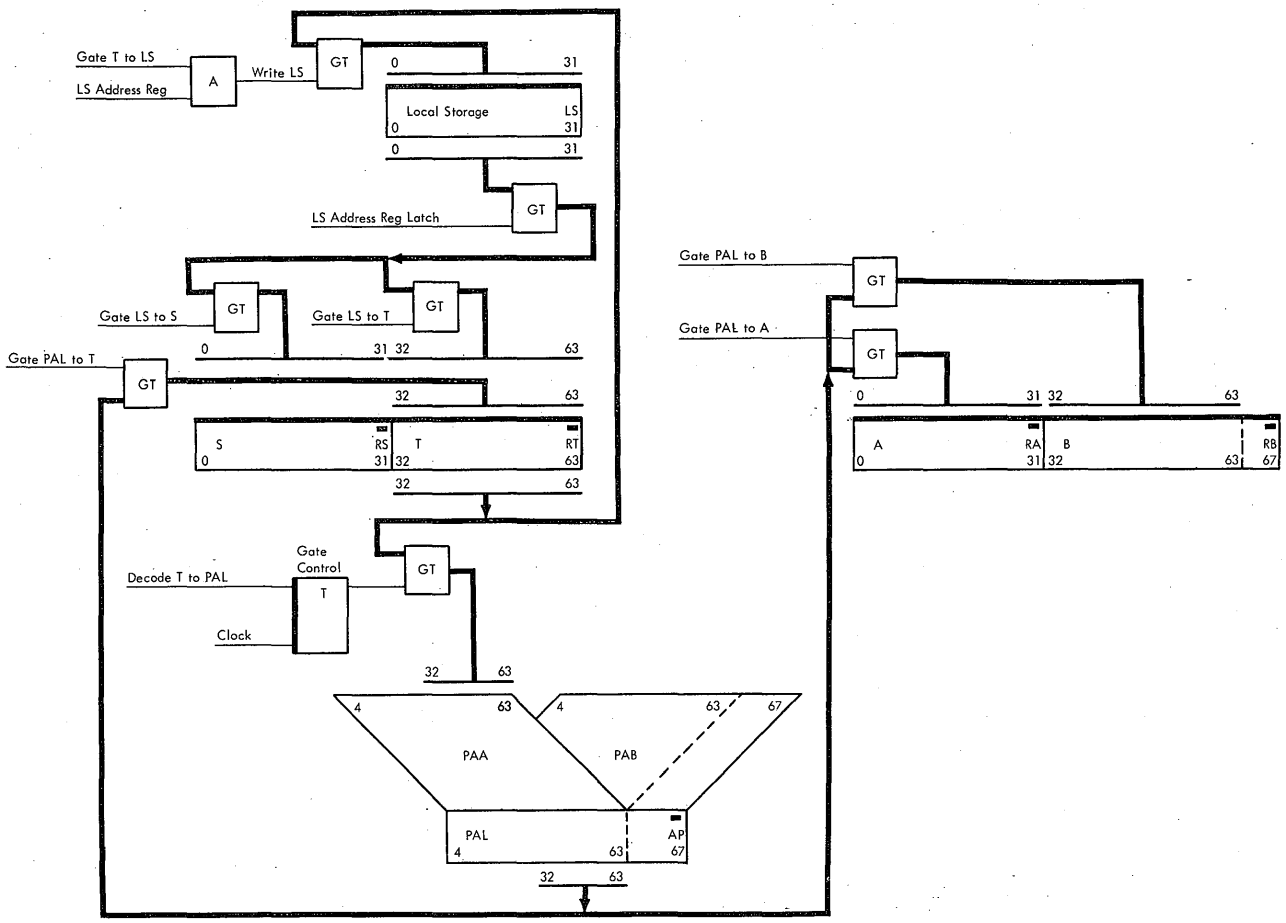


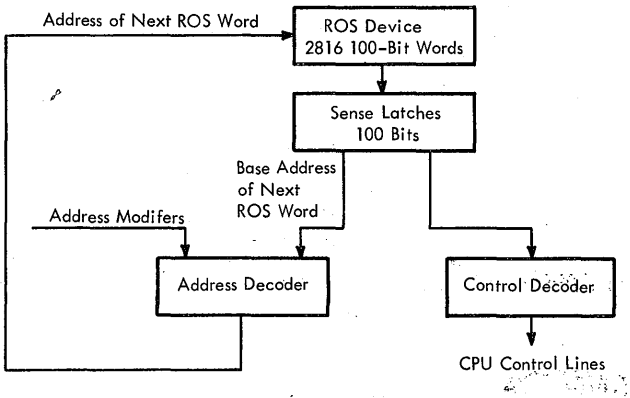
Figure 1-16. Data Transfer Scheme

Relationship of ROS to Conventional Controls

- ROS words replace most conventional sequence triggers and control lines.
- ROS word is a unique bit configuration and controls CPU during machine cycle it is in use.
- In addidion to control data, ROS word holds address of next ROS word and branch tests, if any.
- For branches, 1 ROS word is associated with each possible condition.

To understand ROS operation, it is helpful to note its relationship to conventional controls. Conventional controls may be characterized by sequence triggers, and by the control lines activated by the sequence triggers as a function of the operation to be performed and data conditions. Each cycle that the CPU may take represents a state of the CPU as defined by the control circuits. Each state, in turn, specifies which control lines are to be activated during that cycle and which state is to follow next. The defined state will cause the next sequence trigger to be set in the following cycle. In some cases, the next state may be contingent upon a branch condition in which one of two or more sequence triggers must be selected.

In ROS-controlled CPU's, the sequence triggers are replaced by micro-instructions or ROS words. Each ROS word consists of a predetermined bit pattern and represents a state of the CPU. A micro-instruction is read into the sense latches from the ROS device as follows:



Decoding of the bit pattern activates control lines which initiate operations or functions in the CPU under timing control of ROS decode logic. The base address of the next ROS word to be used is also included in each ROS word. Data conditions within the CPU may modify the address if

the bit pattern indicates a test for branching (e.g., branch if overflow occurs). One ROS word is associated with each possible condition; the base address is modified to address the ROS word which satisfies the data conditions. Thus ROS eliminates the need for most of the complex sequencing networks.

ROS Word

- ROS word is divided into 100 bits, grouped into 21 control fields.
- Number of bits within field determines number of unique control signals within field.
- Control signals are termed *micro-orders*.

The ROS is physically organized into 16 planes, each plane holding 88 200-bit words. Through addressing, the 200-bit word is further divided in half to yield 2816 100-bit words, hereafter referred to as *ROS words*. Each ROS word consists of a unique predetermined bit configuration grouped into 21 control fields (Table 1-6). The number of bits within a field determines the number of unique control signals (micro-orders) available within that field. (In a four-bit field, for example, 16 distinct micro-orders can be defined, only one of which can be activated at any one time.) The micro-orders are grouped functionally within the fields according to two rules:

1. Micro-orders that are functionally similar (such as micro-orders that control ingating to the AB register) are grouped in one field for ease of decoding.
2. All micro-orders grouped in a field must be mutually exclusive because only one micro-order within that field may be specified at a time.

Usually, rule 1 results in rule 2.

When decoded, each micro-order activates one or more control lines that condition gates to perform the function specified by the micro-order. Each micro-order is assigned a mnemonic code (up to 12 characters) that defines the control function performed. As an example, Table 1-7 lists the micro-orders and associated microcommands pertaining to control field V of the ROS word, referenced to the bit configurations of that field, and their function.

Each ROS word is represented by a block on a Control Automation System (CAS) Logic Diagram (CLD). The CLD is to the ROS microprogram what an ALD (Automated Logic Diagram) is to logic. The blocks are connected to show the logical sequence of ROS words to perform the specific function. Refer to ALD M7061 for a definition of the CLD format and content, and of the ROS block language and information contained within the block.

Table 1-6. ROS Word Breakdown

Bits	Control Field	Function Controlled
0–5	–	Spare
6–9	A	A-, B-, and IC-register ingating
10, 11	B	LS to S- and T-register ingating
12–16	C	PSW and S-, T-, D-, G-, and Q-register ingating
17–19	D†	F-register ingating, and end-op signals
20	–	Parity
21–24	E	E- and R-register ingating
25–30	F†	Status triggers and miscellaneous control lines
31–35	G†	Status triggers, IC, and miscellaneous control lines
36, 37	–	Spare
38–42	H	LS
43–46	L	Storage requests and setting of mark triggers
47–56	NA	Base address of next ROS word
57–61	K	Y-conditional branches
62–68	J	Z- (and X-) conditional branches
69–73	M	Serial adder A bus
74–77	N	Serial adder B bus
78–80	P	Parallel adder latches
81	–	Spare
82–84	Q	Hot 1's to parallel adder A-side
85	–	Parity
86	R	F- and AB-register outgating to serial adder A-bus
87–90	T	A-, B-, IC-, and F-register outgating to parallel adder B-bus
91	–	Parity
92–95	U	S-, T-, and D-register outgating to parallel adder A-bus
96	–	Spare
97–99	V	E- and Q-register outgating to parallel adder B-bus

† Control fields D, F, and G serve two functions. In the normal processing mode, they are decoded to yield standard CPU micro-orders; in the scan mode, they are identified as field S, and they yield special scan micro-orders (using common micro-order codes). The choice of modes is controlled by a 'scan mode' trigger.

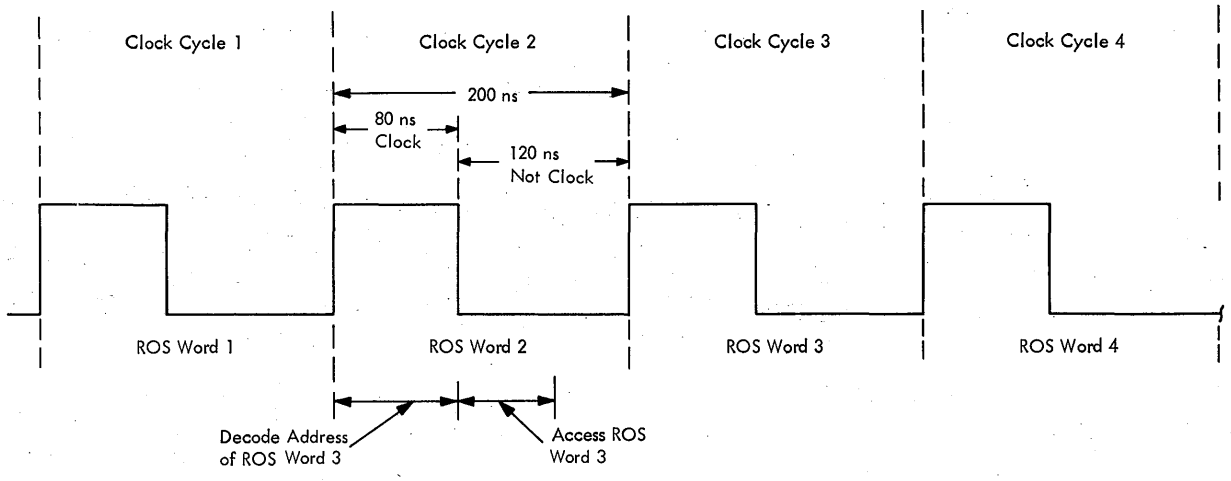
Table 1-7. Control Field V (Bits 97–99); E, Q outgates to Parallel Adder B-Bus

Bit Configuration			Micro-Order Mnemonic	Function
97	98	99		
0	0	0	0	Zero gated with parity
0	0	1	E3	E(12–15) to PB(60–63)
0	1	0	E2	E(8–11) to PB(60–63)
0	1	1	E23	E(8–15) to PB(56–63)
1	0	0	Q7	Q(52–63) to PB(52–63)
1	0	1	Q5	Q(36–47) to PB(52–63)
1	1	0	Q3	Q(20–31) to PB(52–63)
1	1	1	Q1	Q(4–15) to PB(52–63)

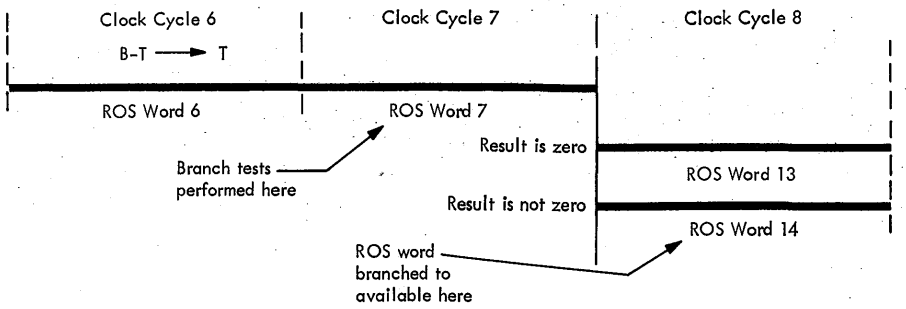
ROS Addressing and Branching

- Conditional branches are dependent on internal conditions of previous cycle.
- Word addressed as result of branch test is not available until 1 cycle later.
- ROS word is addressed by 12-bit binary address:
  0–9 is 10-bit base address.
  10 is Y-branch bit.
  11 is Z-branch bit.
- X-branch (functional branch) affects more than 1 bit.
- Overriding branch blocks 12-bit address and forces new 12-bit address into ROSAR.

As described earlier, the CPU cycle presently being executed is controlled by the ROS word addressed during the previous cycle. Referring to Figure 1-17, A, the normal sequence of ROS words is achieved by placing the address of ROS word 2 into ROS word 1, the address of ROS word 3 into ROS word 2, and so on. The address of the next ROS word is decoded during clock time of the cycle controlled by the present ROS word; the next ROS word is accessed during not-clock time of the cycle.



A. Sequential ROS Word Addressing



B. Conditional Branch ROS Word Addressing

Figure 1-17. ROS Addressing and Branching

Conditional branches are dependent on the internal conditions of the previous cycle. It is important to note that the ROS word addressed as a result of the branch test is not available until one cycle later. To explain this 1-cycle delay in addressing, assume that ROS word 6 contains the micro-orders necessary to subtract the contents of the T-register from the contents of the B-register, and to place the result into the T-register (Figure 1-17, B). Assume further that if the result is zero, a branch will be made to ROS word 13; if not zero, the next ROS word addressed will be 14. Contained in ROS word 6 is the address of ROS word 7, which defines the branch test and contains the associated branch addresses.

The results of the arithmetic operation performed in cycle 6 are tested during clock time of cycle 7. It is during this time that the address of ROS word 13 or 14 (depending on the results of the branch test) is decoded: the selected ROS word is accessed during not-clock time of cycle 7. Hence, the ROS word branched to as a result of the arithmetic operation performed during cycle 6 is not available until cycle 8.

ROS words are selected by means of a 12-bit binary address. The address is held in the ROS address register (ROSAR), whose bit positions are numbered 0 through 11, high order to low order. Bits 0 through 10 specify a 200-bit doubleword in ROS; bit 11 gates the proper 100-bit half to the ROS sense latches (Figure 1-18).

The 12-bit address is made up of three components:

1. A 10-bit base address, bits 0–9.
2. A conditional branch test, or an unconditional value of 0 or 1 applying to bit 10, designated Y-branch.
3. A conditional branch test, or an unconditional value of 0 or 1 applying to bit 11, designated Z-branch.

Included in the Z-branch field of micro-orders is a subset of branch micro-orders called X-branch or functional-branch micro-orders. The X-branch micro-orders affect more than one bit of the ROS address.

Included in the Y-branch field of micro-orders is a subset of overriding branch micro-orders. When the conditions tested by these micro-orders are satisfied, the full 12-bit address is blocked and a new 12-bit address is forced into ROSAR.

If branching conditions are to be tested, the address bits that may be affected by the branch must be set to 0's, except in the case of an overriding branch. If the branch is satisfied, 1's are forced into the ROSAR bit positions associated with that branch test; if the branch condition is not satisfied, the bits remain 0's. Thus the address is modified only if the branch is to be taken.

Addresses can be grouped into four categories: (1) no branch specified, (2) Y- and/or Z-branches, (3) X-branches, and (4) overriding branches. The following paragraphs discuss the addressing for each category. Refer to ALD M7061 for a definition of ROS addressing and branching terms used in the following paragraphs.

No Branch Specified. If no branch tests are to be made, there is only one possible ROS word that can follow, and hence only one possible next address. Accordingly, the 10-bit base address (bits 0–9) and absolute values of 1 or 0 for bits 10 and 11 are specified. The micro-orders that unconditionally set an absolute value into bits 10 and 11 are:

1. 0 in left of R-line (K0), which sets bit 10 to a 0.
2. 1 in left of R-line (K1), which sets bit 10 to a 1.
3. 0 in right of R-line (J0), which sets bit 11 to a 0.
4. 1 in right of R-line (J1), which sets bit 11 to a 1.

The appropriate Y- and Z-branch micro-orders are selected, and bits 10 and 11 are set correspondingly.

Y- and/or Z-Branches. Conditional branch addresses may be specified in which bits 10 and/or 11 are affected.

Only a Y-branch can be executed as follows. A 10-bit base address and an absolute value or X for bit 11 are specified. A branch test is defined in the Y-branch micro-order control field. If the branch condition is satisfied, bit 10 is set to a 1; if not, bit 10 remains a 0. For example, micro-order 'WCRY' sets bit 10 to a 1 if a carry is detected in the serial adder. If there is no carry, bit 10 remains a 0.

Conversely, only a Z-branch can be executed as follows. Here, a 10-bit base address and an absolute value for bit 10 are specified. If the branch test defined in the Z-branch micro-order control field is satisfied, bit 11 is set to a 1; if not, bit 11 remains a 0.

Certain situations require the use of both Y- and Z-conditional branches simultaneously. The 10-bit base address is specified, and bits 10 and 11 may assume one of the following values:

Bits 10	Bits 11	Branch Results
0	0	Y- and Z-branch conditions both unsatisfied.
0	1	Z-branch condition only satisfied.
1	0	Y-branch condition only satisfied.
1	1	Y- and Z-branch conditions both satisfied.

X-Branches. Where a branch to one of four or more possible addresses is required (as well as some special 64-way branch tests), an X-branch is used. The X-branch may affect bits 10 and 11 (four-way branch), bits 9–11 (eight-way branch), bits 8–11 (16-way branch), or bits 6–11 (64-way branch). An example of an X-branch is the 64-way branch, 'E(2–7)→ROA', made at the end of the I-Fetch sequence per the op code to enter the execution phase of the specific instruction.

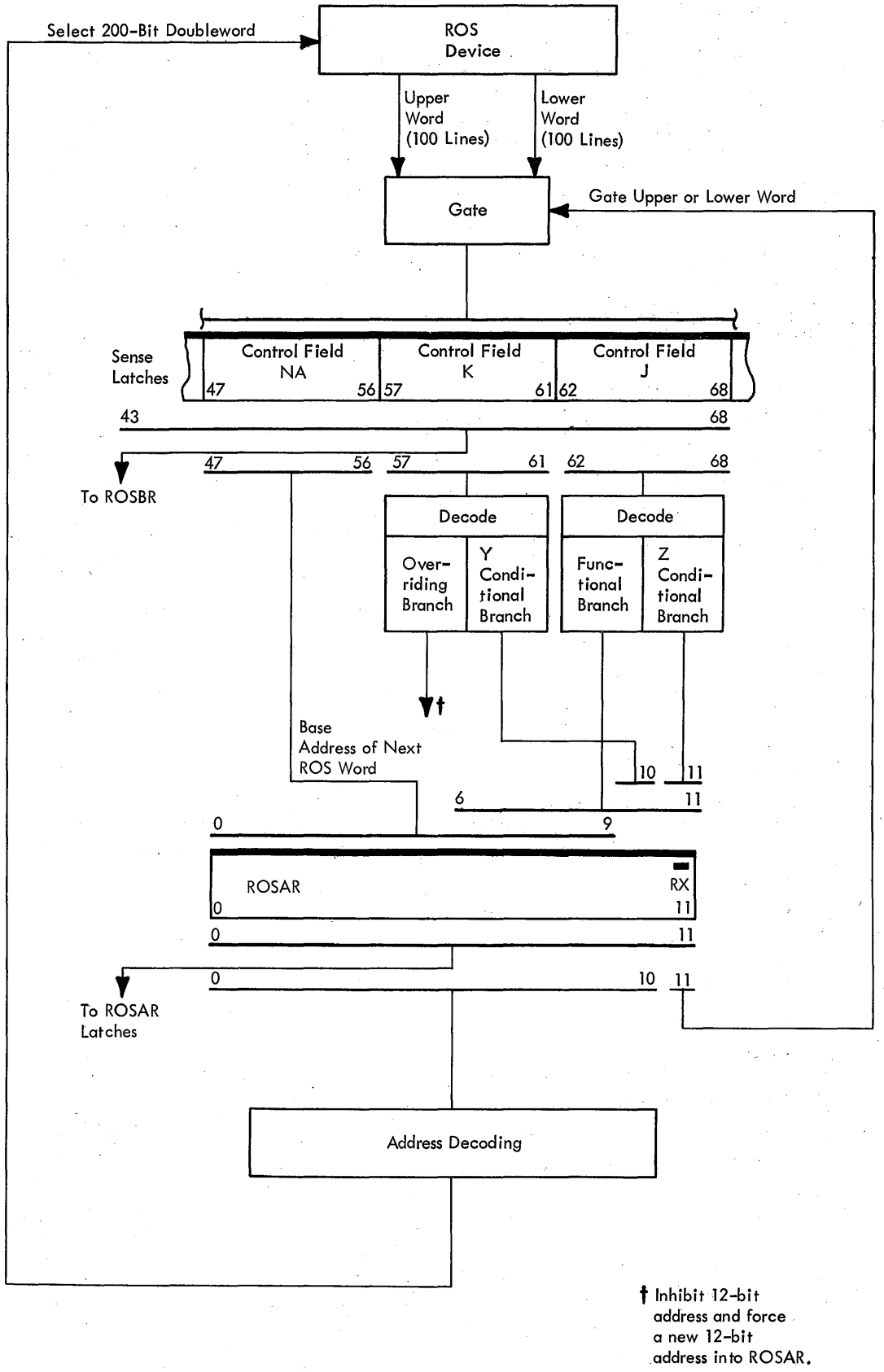


Figure 1-18. ROS Addressing Block Diagram

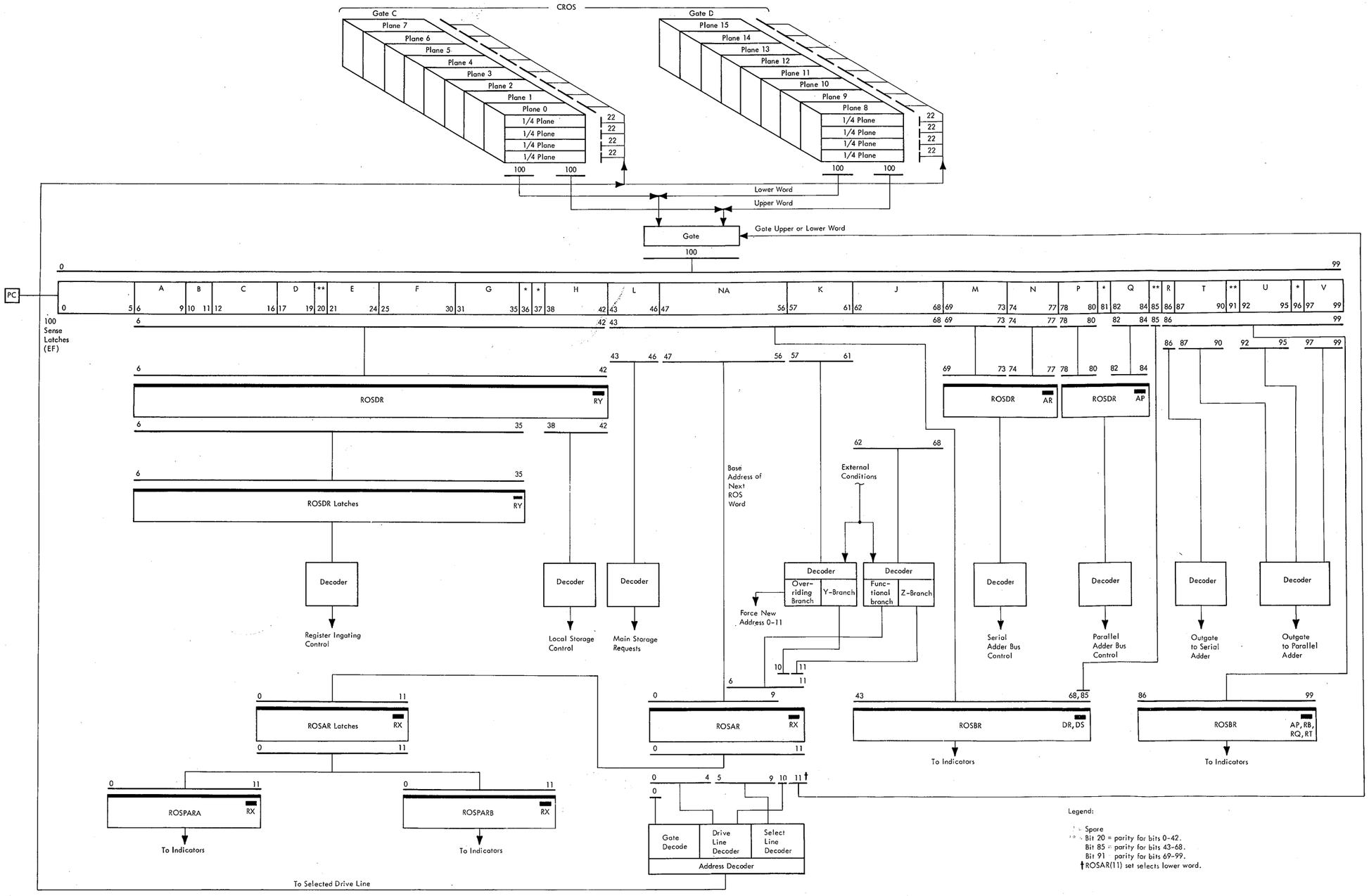


Figure 1-19. ROS Data Flow

For these multiway branches, one condition sets the associated address bit to a 1. To illustrate, assume conditions A, B, and C sets bits 9, 10, and 11, respectively, to a 1. The possible results are:

Bits			Branch Results
9	10	11	
0	0	0	None of the conditions is satisfied.
0	0	1	Condition C is satisfied.
0	1	0	Condition B is satisfied.
0	1	1	Conditions B and C are satisfied.
1	0	0	Condition A is satisfied.
1	0	1	Conditions A and C are satisfied.
1	1	0	Conditions A and B are satisfied.
1	1	1	All three conditions are satisfied.

The addressing is similar to that previously discussed. A 10-bit base address is specified, with those bits that may be affected by the X-branch set to 0. Thus, for the example given above, bit 9 of the base address is set to 0, the Y-branch micro-order control field contains micro-order 0 in left of R-line to set bit 10 to 0, and bit 11 is automatically set to 0 when the X-branch is specified. Subsequently, the bit(s) associated with successful condition(s) is set to 1. The ROS word addressed will be that ROS word whose address satisfies the branch conditions.

Overriding Branches. There are exceptional machine conditions (such as interruptions) for which the normal ROS word sequence must be stopped and a new sequence started. This change is accomplished by an overriding branch specified in the Y-branch micro-order control field.

The normal sequencing address is made up of (1) the 10-bit base address and (2) bit 11 set to 0 automatically because the overriding branch is a function of the Y-field. Because the overriding branch is specified in the Y-control field, no Y-branch can be specified.

If the overriding branch condition is satisfied, the normal full 12-bit address is blocked and a new address, as determined by the overriding branch condition, is forced into ROSAR.

ROS Data Flow

- ROS data flow units are:
  100 sense latches, 1 per ROS word bit
  ROSAR
  ROSAR latches
  ROSPARA and ROSPARB
  ROSDR
  ROSDR latches
  ROSBR
  Decode logic
- Control fields may be:
  Decoded directly from sense latches.
  Placed into ROSDR and subsequently decoded.
  Placed into ROSDR, sent to ROSDR latches, and then decoded.

The 100-bit ROS word is divided into 21 control fields. When read out from ROS, the ROS word is placed into 100 sense latches, one latch for each bit position. The control fields are handled according to the functions they control. They may be (Figure 1-19):

1. Decoded directly from the sense latches (control fields L, K, J, R, T, U, and V) or transferred directly to ROSAR (control field NA).
2. Placed into the ROS data register (ROSDR) and decoded (control fields H, M, N, P, and Q).
3. Transferred to ROSDR latches from ROSDR (control fields A–G).

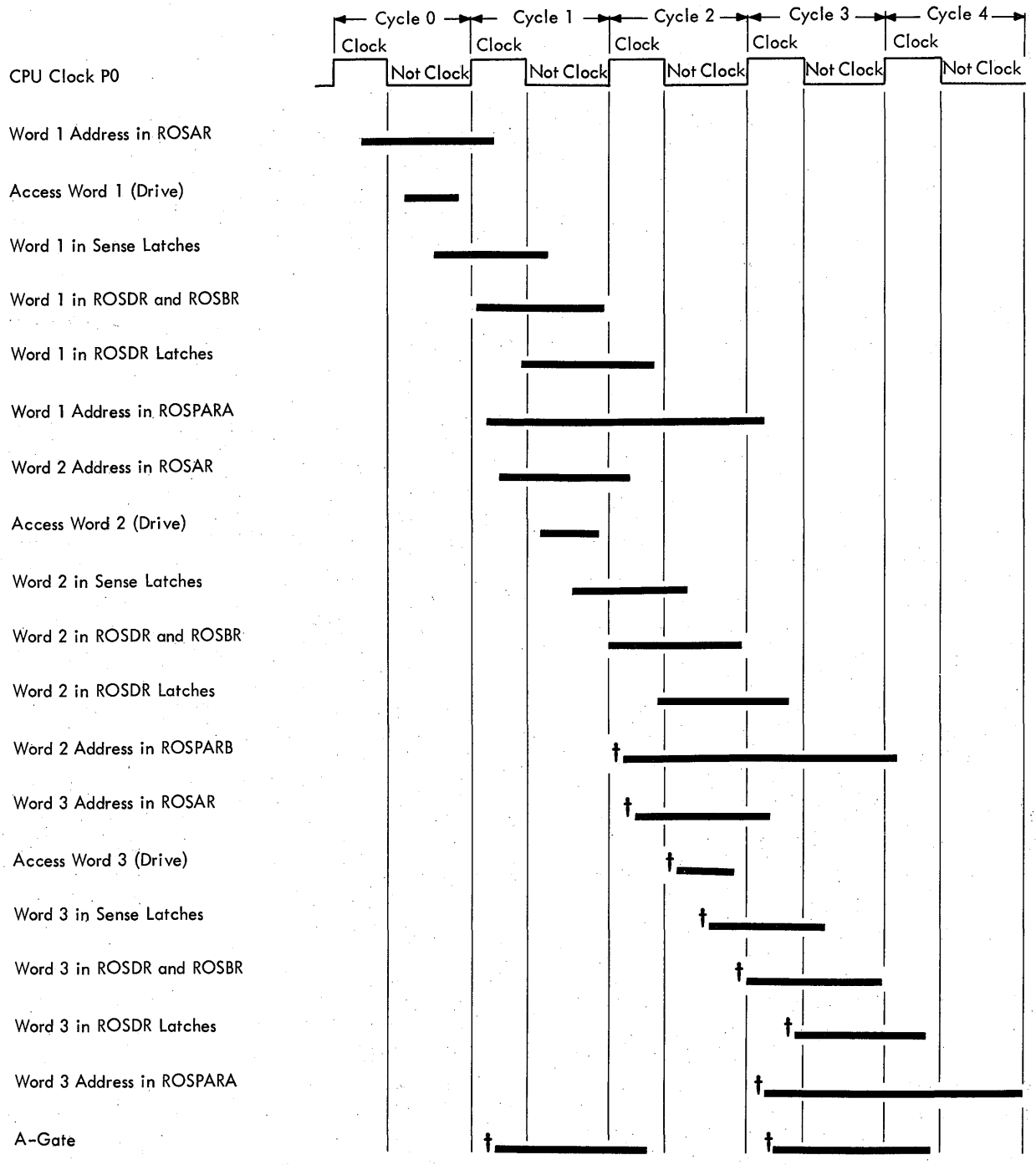
Assuming ROS words and the cycles they control are designated 1, 2, and 3, ROS word 1 is set into the sense latches during not-clock time of cycle 0 (Figure 1-20). The control fields used during clock time of cycle 1 are decoded directly from the sense latches (case 1 above). These control fields, which may be considered critical timing fields, control inputs to the adders, define the base address and branch tests for the next ROS word, and control the storage-request and mark triggers.

Although the sense latches are cleared at not-clock time of cycle 1, the control fields of ROS word 1 that are required during that time (case 2 above) are placed into ROSDR at clock time of cycle 1. These signals control the adders and LS. Note that both portions of the ROSDR associated with the adders are packaged physically with the adders they control. The balance of the ROSDR serves control fields A–H.

Control fields A–G control register inputs and triggers that are to be set during clock time of cycle 2 (case 3 above). Although the ROSDR is reset at the end of cycle 1, the ROSDR latches keep control fields A–G available for that additional 80 ns (Figure 1-20).

Control fields L, NA, K, J, R, T, U, and V are sent to the ROS backup register (ROSBR) from the sense latches. The ROSBR does not play a part in the ROS functions; it provides an indication of the subject fields during maintenance (test) mode. When the CPU stops on an error during test mode, the ROSBR contents can be used by the CE to help isolate malfunctions.

The NA control field, in addition to being stored in the ROSBR, is stored in ROSAR and provides the base address as previously explained. During each ROS cycle, the contents of ROSAR are sent to the ROSAR latches which, in turn, are alternately gated (by means of an 'A-gate' signal) to the ROS previous address A (ROSPARA) and ROS previous address B (ROSPARB) registers. These registers serve the same purpose as the ROSBR; i.e., provide the CE with an indication for maintenance use during test



† If an error is detected in cycle 1, these steps are not performed.

Figure 1-20. ROS Timing

mode. When an error causes the CPU to stop in test mode, ROSAR, ROSPARA, and ROSPARB provide the addresses of the next ROS word, current ROS word, and previous ROS word (Figure 1-20).

ROS Control of CPU

Efficient control of CPU operations is achieved by overlapping ROS words. Clock signals (P0–P19) time ROS sense latches, ROSDR, and ROSDR latches, thus allowing the processing of parts of more than one ROS word simultaneously To illustrate, Figure 1-21 shows the ROS word timing relationship for a hypothetical example.

The address of word 1 is gated into ROSAR from the ROS sense latches at P5 of word 0. [ROSAR(11) may be set as late as P7.] Information from word 1 enters the ROS sense latches at P0 + 160 ns (P16). (In normal operation, a

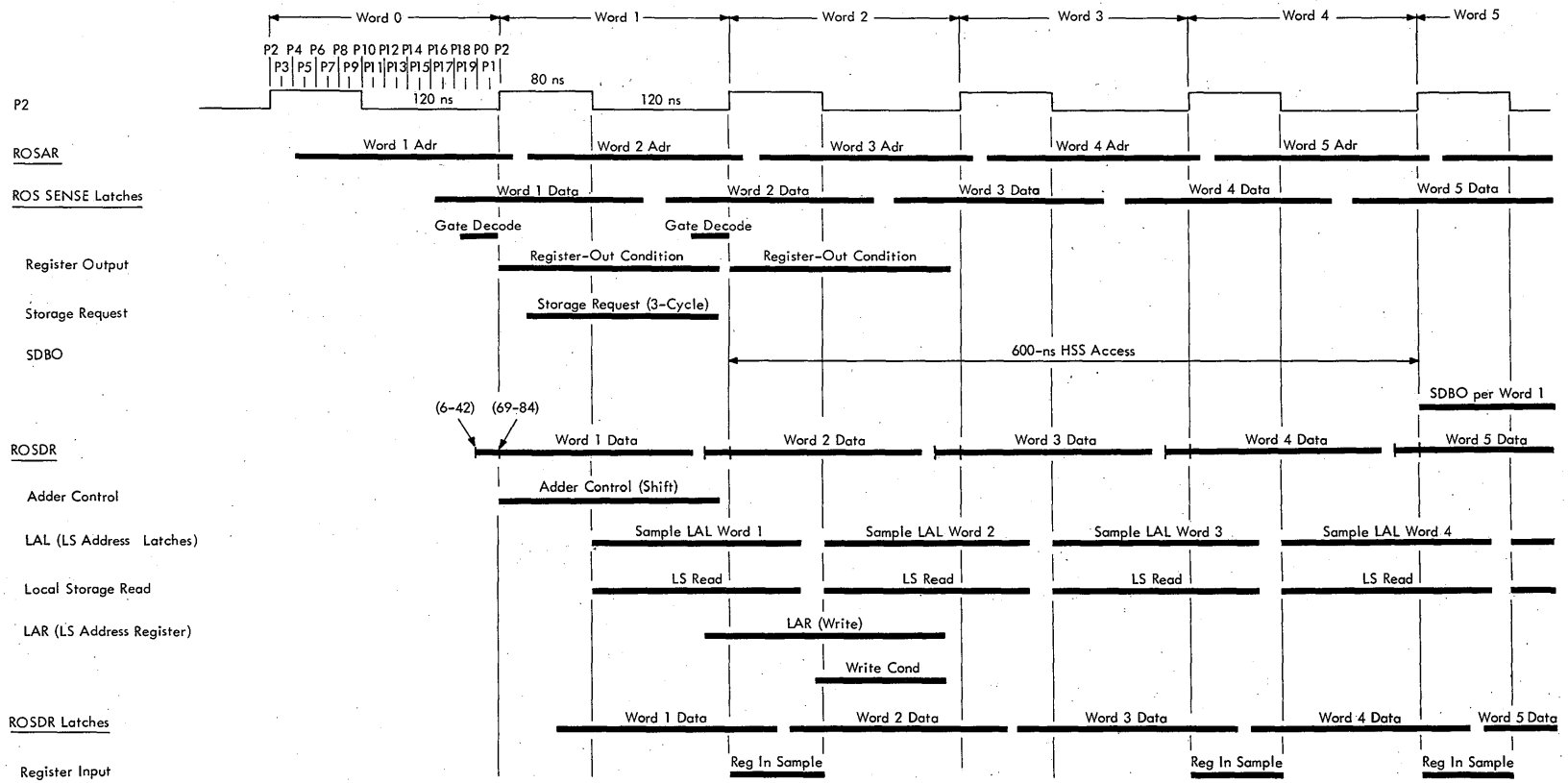


Figure 1-21. ROS Control of CPU Operations

new word enters the ROS sense latches every 200 ns.) In the example, word 1 controls: (1) register output, (2) main storage request, (3) adder shift, (4) local storage write, and (5) register input. A register output micro-order gates register data into an adder at P2 by means of 'gate control' triggers. A three-cycle main storage request is initiated at P4 to fetch information which is used three cycles later. Note that register output and the main storage request are decoded directly from the ROS sense latches.

Bits 6–35 and 38–42 enter ROSDR at P0, and bits 69–80 and 82–84 enter ROSDR at P2; they are decoded for adder control and LS operations. In the example, micro-orders generate an adder shift and a local storage write operation. The shift is performed immediately, but the local storage write operation is delayed because a local storage read operation is automatically set up first. The LS address is entered into the local storage address latches (LAL); a read operation is performed, but data is not gated into a register.

The address then is gated into the local storage address register (LAR) to perform the write operation. Note that the write condition, ordered by word 1, starts after word 2 has been transferred to the ROSDR. The figure shows an LS read operation for every word because this sequence happens even if it is not ordered. When no order is given to LS, a readout of LS address 0 is performed but the data is not used. Forcing the read operation saves time if it is needed.

ROSDR(6–35) is gated to the ROSDR latches at P7 to retain word 1 information at the same time word 2 is set into ROSDR. In the example, word 1 transfers data into a register at clock time (P2) of word 2. This action, along with an LS write operation, illustrates ROS word overlap because these operations are performed at the same time the register out condition is energized from the decoding of word 2.

Word 2 has only one micro-order, register output, but gates are conditioned to transfer word 2 data from the ROS sense latches to the ROSDR and then to the ROSDR latches. Note that the LS read micro-order is active, though not ordered. Word 3 operates in a similar manner, but the only micro-order is a register input which takes place during word 4.

As the ROS words are executed, the main storage request is processed and data is returned on the Storage Data Bus Out (SDBO); word 4 contains the micro-order to gate the data into a register for further processing.

PSW Register

Program status words (PSW's) contain detailed information pertaining to the particular mode in which the CPU is operating. These status words are composed of a system mask, storage key, program state, interruption code, instruction length code, condition code, program mask, and an instruction address that enables the interrupted program to resume at the correct location.

Status information concerning the current operating program is contained in several groups of triggers, from where it controls all system operations essential to that particular program mode. These groupings of control triggers, although not adjacently located in logic, are collectively referred to as the PSW register. Although a completely assembled PSW is 64 bits long, only 24 positions of status word data are contained in the PSW register. The remaining information (generated by the CPU at the time of the interruption) is not retained when a previously stored PSW is reloaded, because its function is only to identify the cause of the interruption and to return the CPU to the correct program location. (This information is gated directly from ST when the old PSW is recalled from main storage.) Figure 1-22 shows an assembled PSW and those areas of control information retained in the PSW register.

BUS CONTROL UNIT

In the 2065, the BCU responds to storage requests from the CPU and from up to seven I/O channels, all of which may be operating asynchronously with respect to each other. The flow of information between main storage and all units serviced by the BCU is effected through a single bus system. Because the bus system is shared by all units, the BCU must resolve conflicts between simultaneous storage requests from these units and ensure that the storage bus system is available to one unit at a time. Thus, the major function of the BCU is to provide for efficient time-sharing of the storage bus system by all units.

Each unit requiring access to storage communicates with the BCU through individual control lines. These lines are monitored in the BCU to establish priority between storage requests and to inform the requesting units of the bus system availability.

When priority is awarded to a requesting unit, the BCU decodes the high-order bits, and bit 20, of the storage address supplied by that unit. The BCU then sends a 'select' signal to the proper storage unit. The selected main storage unit initiates a storage cycle and decodes the low-order bits (6–19) of the address supplied on the storage address bus (SAB) to access the required doubleword location. When the BCU grants storage access to one unit, storage requests from all other units remain pending until the current data transfer operation is completed. At that time, the BCU performs a new priority test on all units (including the unit just serviced).

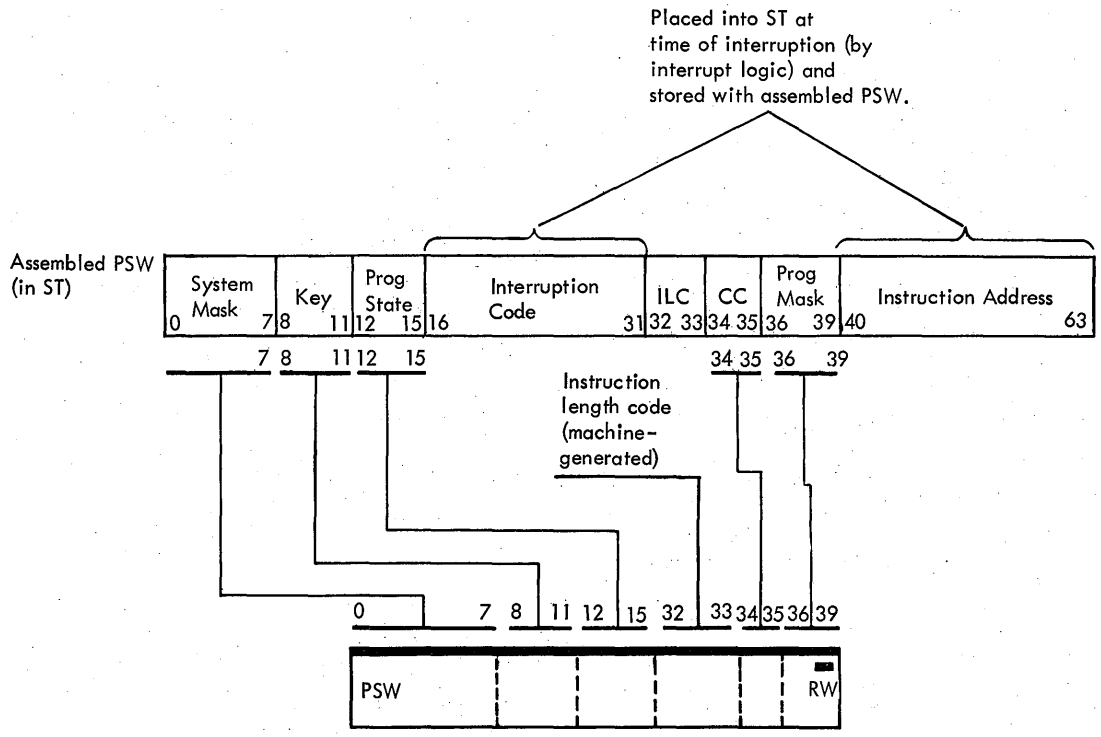


Figure 1-22. Status Information Contained in PSW Register

Major Interface Lines

A simplified diagram of the BCU interface with the Model 65 system is shown in Figure 1-23. This interface consists of multiplex data bus lines, and of simplex and multiplex control lines. By definition, a multiplex line is a line that is shared by *more than two* units of the system; for example the SAB, which connects the BCU and all the I/O channels to all storage units in the system. A simplex line is a line that is used exclusively between two units of the system. For example, individual 'select' lines are used to interconnect the BCU to *each* storage unit in the system.

The major interface lines shown in Figure 1-23 are defined as follows:

1. Address and data buses.
   a. Storage Address Bus (SAB). This multiplex bus connects the CPU and all I/O channels to all main storage units in the system. The SAB consists of 21 address lines (plus three parity lines) and specifies the address of a doubleword contained in one of the main storage units associated with the system. Note that the SAB path is different from the other buses. This difference is necessary because SAB(0–6, 20) from the CPU and from the channels must enter the BCU storage address decoder to select a storage unit.
   b. Storage Data Bus Out (SDBO). This multiplex bus connects the CPU and all I/O channels to all main storage units in the system. The SDBO consists of 64 data lines (plus eight parity lines) and carries data requested from main storage by the CPU or the I/O channels.
   c. Storage Data Bus In (SDBI). This multiplex bus connects the CPU and all I/O channels to all main storage units in the system. The SDBI consists of 64 data lines (plus eight parity lines) and carries data to be stored into main storage. This data may be supplied by the CPU or by the I/O channels.
   d. Mark Bus. This multiplex bus connects the CPU and all I/O channels to all main storage units in the system. The mark bus consists of eight mark lines (plus one parity line) and designates the bytes on the SDBI that are to be stored into main storage. (There is one mark line for each byte on the SDBI; complete absence of mark signals occurs only on a fetch operation.)
   e. Key-In Bus. This multiplex bus connects the CPU and all I/O channels to all main storage units in the system. The key-in bus consists of five key lines (plus one parity line) and transfers the storage protection key from the CPU or the I/O channels to the storage protection area in the selected main storage unit.
   f. Key-Out Bus. This multiplex bus connects the CPU to all main storage units in the system. The key-out bus, which consists of five key lines plus one parity line, is used only during execution of the insert-key operation by the CPU. This operation transfers the storage protection key from the selected main storage unit to the CPU.

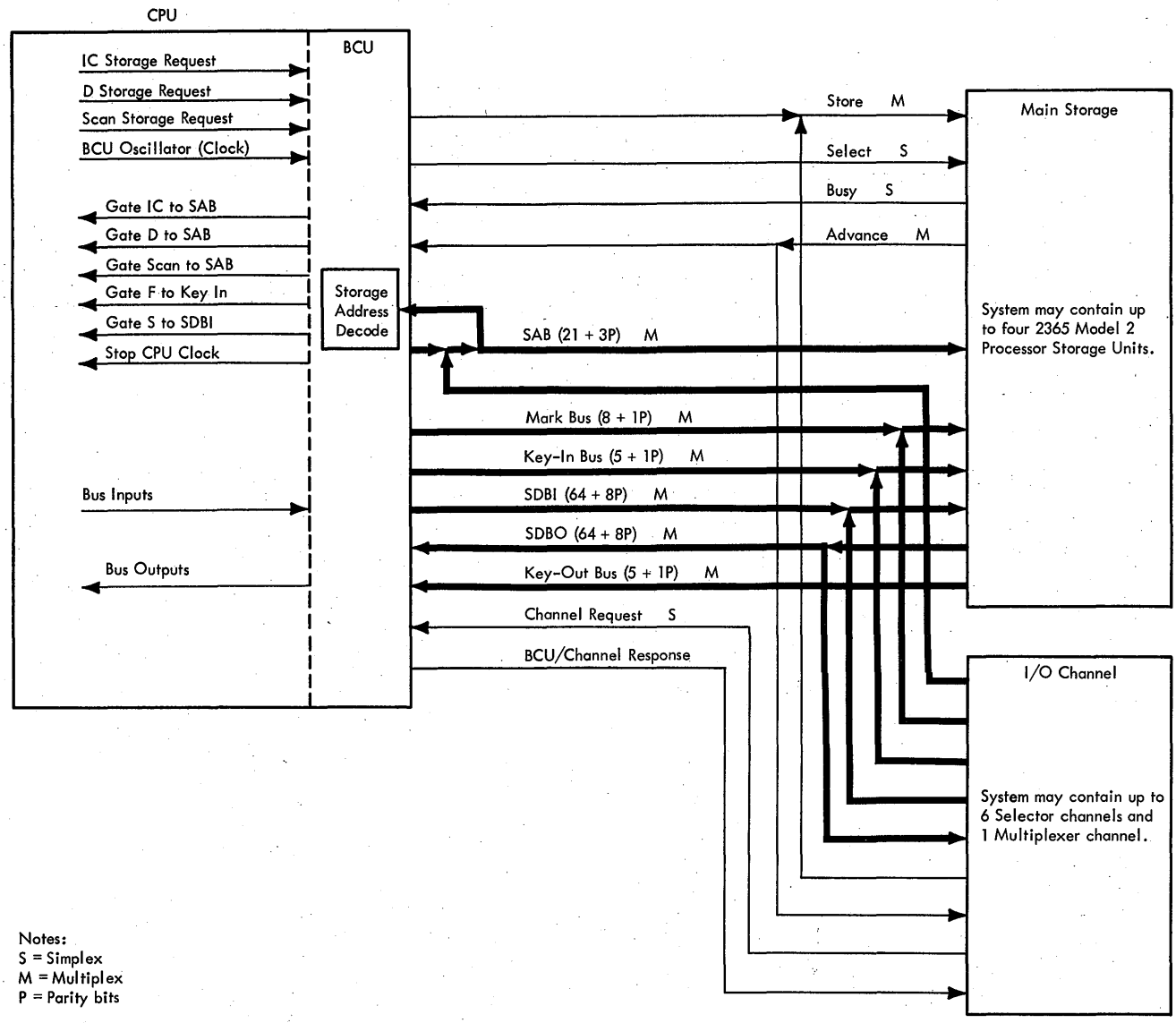


Figure 1-23. Basic BCU Interface

On channel operations, the mark bus, key-in bus, SDBI, and SDBO are independent of the BCU.

2. Major control signals.
   a. 'Select'. This simplex signal is issued by the BCU to the selected storage unit. This signal causes the selected unit to gate in the address from SAB and initiate a storage cycle.
   b. 'Store'. This multiplex signal is issued on all store operations performed by the CPU or by the channels to instruct the storage unit to store the contents of the SDBI as specified by the mark signals. The absence of a 'store' signal indicates a fetch operation.
   c. 'Busy'. This simplex signal is sent from a storage unit to the BCU to signify that the storage unit is in a store cycle.
   d. 'Advance'. This multiplex signal is issued by the storage unit to the BCU and all channels to indicate that the storage unit is about to gate data onto the SDBO. (The 'advance' signal is received approximately 200 ns before the data arrives on the SDBO.)
   e. 'Channel request'. This simplex signal is issued by a channel to the BCU when the channel needs access to main storage.
   f. 'BCU/channel response'. This simplex signal is issued by the BCU to the requesting channel when the BCU has awarded priority to that channel.

Within the CPU, storage requests are generated and sent to the BCU to develop a CPU request. The signals that perform this function are related to the source of the storage address, as follows:

1. IC storage request. The storage address is in the instruction counter.
2. D-storage request. The storage address is in the D-register.
3. Scan storage request. The storage address is developed in scan logic.

Representative signals developed by the BCU to control the buses are:

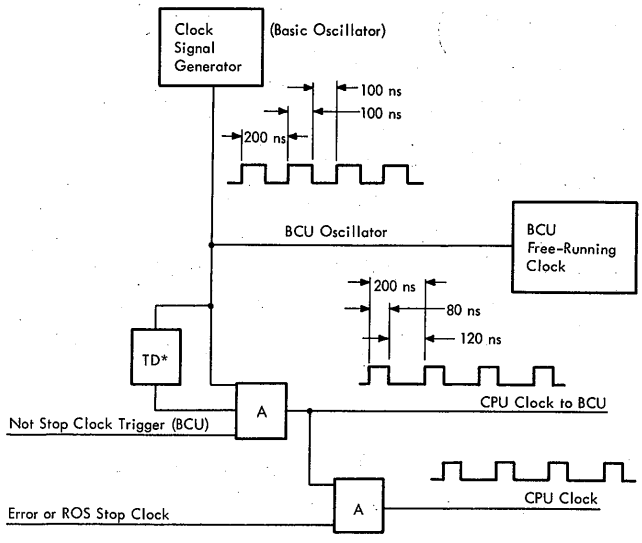
1. 'Gate IC to SAB'. Gates the contents of the instruction counter to SAB.
2. 'Gate D to SAB'. Gates the contents of the D-register to SAB.
3. 'Gate scan to SAB'. Gates the address developed by the scan logic to SAB.
4. 'Gate F to key-in'. Gates the contents of the F-register to the 'key in' bus.
5. 'Gate S to SDBI'. Gates the contents of the ST register to SDBI.
6. 'Stop CPU clock'. Stops the CPU clock when unable to grant priority to a CPU request.
7. 'Gate SDBO to AB'. Gates the data on the SDBO into the AB register.
8. 'Gate SDBO(0–31) to S(0–31)'. Gates SDBO(0–31) into the S-register.
9. 'Gate SDBO(32–63) to T(32–63)'. Gates SDBO (32–63) into the T-register.
10. 'Gate SDBO to Q(0–63)'. Gates SDBO(0–63) into the Q-register.

BCU Clocks

The timing of latches and triggers within the BCU is controlled by signals from the 2065 clock signal generator (Figure 1-24); thus the BCU operation is synchronized with the CPU. The BCU has two clocks: a free-running clock, which is active whenever power is applied to the 2065, and a 'CPU clock to BCU' signal. As shown in Figure 1-24, an error or the ROS microprogram can stop the CPU clock without interfering with the BCU clocks.

The free-running clock controls the timing of priority selection, storage unit selection, and channel signals. Additional controls are unnecessary for servicing channel requests because the BCU transfers control of the buses to the channel after the storage unit selection is made. Thus, channel storage requests are made independently of the CPU.

CPU requests, however, require additional gating controls. To control information to and from the buses, the BCU retains control of the buses and develops gating signals which are timed by the 'CPU clock to BCU' signal. When a CPU request cannot be processed because the BCU or storage unit is busy, the CPU clock and the BCU-developed gating signals are disabled by a 'stop clock trigger' signal to prevent transfer of the wrong information (Figure 1-24). The BCU restarts the clocks after it awards priority to the CPU.



* TD converts symmetrical clock signal to unsymmetrical clock signal.

Figure 1-24. BCU Clock Logic

BCU Operation

The BCU provides efficient time-sharing of the main storage by the I/O channels and the CPU. Storage requests are not honored on a first-come, first-served basis, but rather on a priority basis. Figure 1-25 shows the BCU priority logic; channel 1 has highest priority, followed by channels 2, 0, 3, 4, 5, 6, and CPU. When a channel or the CPU is granted

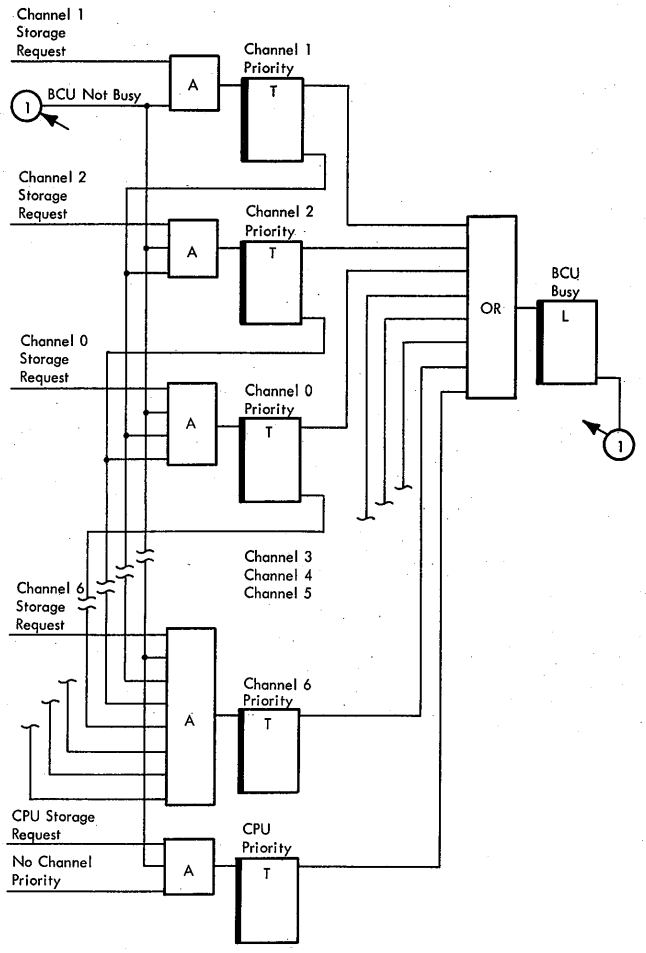


Figure 1-25. BCU Priority Logic

priority, the 'BCU busy' latch is set so that additional requests are not honored until the storage cycle is completed. For example, if after a channel 1 request a second channel 1 request is made before the 'BCU busy' latch is reset, channel 1 is selected again, even if other requests are pending. As another example, if a channel 0 request is made during a storage cycle and is followed by a channel 2 request before the end of the cycle, channel 2 is awarded priority.

CPU Request

A CPU request for storage is generated from one of three sources: (1) instruction fetch logic to refill the instruction buffer (Q-register), (2) microprogram to fetch or store data, and (3) scan logic. In each case, the request is made three or four cycles before the data is needed to allow time for storage access.

The address of the desired doubleword in storage is located in the instruction counter or the D-register, or is generated by the scan logic. BCU develops the gating signals to supply the address from the specified source to SAB. At the same time, BCU controls gating for SDBI, SDBO, 'mark' bus, and the storage address protection keys. When a CPU request for storage is sensed at the BCU, the 'CPU request' trigger is set (Figure 1-26). If the CPU is granted priority during the next priority test, the 'CPU priority' trigger is set, and the storage sequence is started immediately.

If the BCU is busy and cannot grant priority to the CPU, a 'stop clock trigger' signal is sent from the BCU to stop the CPU clock (Figure 1-24). This action stops all CPU activity until the BCU grants priority to the CPU and the storage cycle starts. When the BCU and storage are available, the CPU clock is restarted and the CPU sends or receives data over the buses. The clock signal from the CPU keeps the BCU synchronized with ROS and other CPU functions.

Channel Request

When a channel has been granted priority, the BCU sends a 'BCU response' signal to the channel so that the channel will put the storage address on SAB (Figure 1-26). After the address has been gated to SAB, the channel sends an 'address valid' signal to the BCU. This signal does *not* denote a correct address, but signifies that the available address has been gated to SAB. The BCU uses the 'address valid' signal to sample the SAB.

After the storage unit has been selected, the BCU turns control of the multiplex buses to the channel. Channel registers provide data, marks, and address protection keys as needed. When the CPU receives a 'release' signal from the channel, the CPU is free to continue processing except for conflicts in requesting main storage.

Operation with Main Storage

- Address is decoded to select storage unit.
- Invalid address results if storage capacity is exceeded or if power is off at the selected unit.
- Interleaving reduces storage access time.

A portion of SAB is decoded in the BCU to select the storage unit that contains the addressed doubleword. SAB(4,5) determines the high speed storage (HSS) unit, and SAB(6 or 20) specifies the odd or even half for interleaving. Figure 1-27 shows the data path for the address decoding for HSS and the optional large capacity storage (LCS). Decoding of a 'select' line sends a 'select' signal on a simplex line to the desired storage unit where the address on SAB is decoded to locate the addressed doubleword.

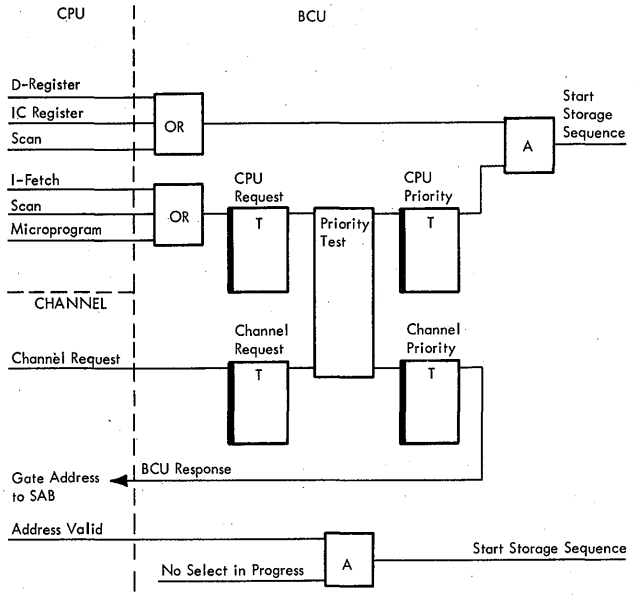


Figure 1-26. Start Storage Sequence Logic

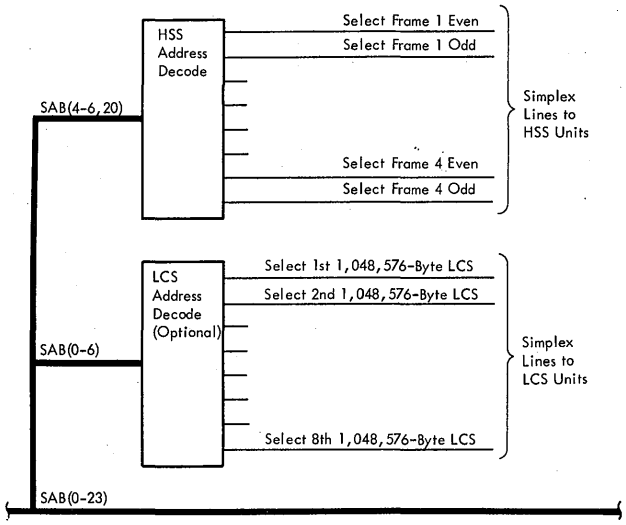


Figure 1-27. Storage Selection

Within the BCU decoder, a test is made for an invalid address condition. Jumper cards are set to define the limit of the storage attached. If the address specifies a value beyond the storage limit, an 'invalid address' signal is developed. The 'invalid address' signal is also developed if power is off at the specified storage unit. The unit originating the address is notified, and an interruption may occur.

The basic storage cycle of the 2365 Processor Storage is 750 ns, but interleaving between odd and even sections reduces the effective storage time to 400 ns. This reduction is possible because each area of storage contains its own bus arrangement, thus enabling independent operation. An address enters storage and is used during the first half cycle to set up the storage access. The second half cycle is used to bring out and store data. During the second half cycle, a new address is sent on the SAB to the other half (odd or even) of storage, thus allowing a storage access in approximately half the normal storage cycle. A delayed (by 80 ns) 'select' signal conditions the BCU reset logic. The delay holds the BCU busy through the cycle after the storage request to prevent a second storage request within 400 ns.

INSTRUCTION FETCHING

The processing of an instruction is divided into two phases: instruction fetching and execution. Instruction fetching, or I-Fetch, retrieves instructions from main storage and performs operations common to many instructions. For the most part, I-Fetch, which is controlled by ROS and conventional hardware:

1. Determines the address to be placed into the instruction counter (IC).
2. Fetches instructions from main storage.
3. Determines the instruction format (RR, RX, RS, SI, or SS).
4. Calculates the effective operand address (adds the D-field, the contents of the LS register designated by the B-field, and the index, if required) for those formats that require that function.
5. Places the operands specified by RX format instructions into the applicable registers (AB, ST, and D). For the other formats, I-Fetch issues a storage request for the second operand. The second operand is placed into the registers during the execution phase.
6. Passes control to the specific execution phase by means of a 64-way branch.

The transition from the execution phase of an instruction to the I-Fetch sequence of the next instruction is achieved by an end-op cycle, the last cycle of the execution phase. The end-op cycle completes the execution phase of the instruction being processed by:

1. Setting the condition code to reflect the result of the instruction, if applicable.
2. Detecting exceptional conditions and interruptions.

The end-op cycle initiates the I-Fetch sequence for the next instruction by:

1. Decoding the format of the next instruction.
2. Initiating operand fetches as required by that format.
3. Performing a 64-way branch to establish the correct I-Fetch sequence for that format.
4. Fetching more instruction halfwords, if required.

Functional Units Used

Five registers play vital roles in the I-Fetch sequence: Q, R, E, IC, and D. The following paragraphs discuss the functions generally performed by these five registers.

Q-Register

- Buffers four instruction halfwords received from main storage.
- Provides for overlap of I-Fetch and instruction execution.
- Transfer of B- and D-fields from Q reduces instruction processing time.

The Q-register is a 64-bit (plus eight parity bits) trigger register that buffers all instructions entering the CPU from main storage (Diagram 3-2, FEMDM). It is divided primarily into four halfword (16-bit) areas. This arrangement provides for the buffering of four instruction halfwords (eight bytes), thus increasing CPU efficiency and reducing the amount of main storage time required by the CPU. The Q-register is loaded directly from the SDBO; information is transferred to the LS address register [LAL (Read) and LAR (Write)], to the R-register, and to the parallel adder.

After being loaded with a doubleword from main storage, those Q-register halfwords containing instruction op codes are sequentially transferred to R (for subsequent execution in E). When the last op-code halfword has been transferred from Q, a new doubleword is again loaded into Q from main storage. This process of continuously refilling Q with instructions is overlapped with instruction execution whenever possible.

Additional Q-register information selects the instruction fields to be sent to LAR and to the parallel adder. Such information consists of four four-bit fields (B-fields) specifying LS registers and four 12-bit fields (D-fields) containing the displacement for main storage addresses. Transferring this information directly from Q instead of via R or E provides a lookahead capability by allowing both LS and effective addresses to be available before the execution time of the associated instruction. Transferring of these 4- and 12-bit fields is performed selectively so that the information is associated with the correct instruction.

Before an instruction is executed, it is tested for odd parity. The op-code halfword is tested in the E-register. The remaining halfwords, if any, are tested by the parallel adder half-sum checking circuits as the effective address is calculated.

R-Register

- Only op-code halfwords are transferred from Q to R.
- Selection of op-code halfword is determined by IC(21,22).

The R-register is a halfword (16 bits plus two parity bits) trigger register, providing intermediate buffering of op-code halfwords between Q and E (Diagram 3-2, FEMDM). This buffering extends the total instruction buffering capability to five halfwords (five instructions in the event of all RR formats), as Q is normally refilled after the last op-code halfword has been transferred to R. The use of two separate registers (Q and R) for containing op-code halfwords also provides double buffering. This scheme allows storage requests to be generated immediately upon transferring the last op-code halfword from Q, instead of having to wait until the instruction in E has been executed, as would be required if halfwords from Q were transferred directly to E.

Because op-code information is all that is required to initiate execution, only those halfwords in Q containing op codes are gated to R. Also, because RX, RS, SI, and SS instructions are composed of either two or three halfwords (only the first of which contains the op code), it is necessary to select the proper halfword to be transferred to R, rather than merely proceeding sequentially through the four halfwords. Selection of the halfword for transfer to R is determined by IC(21,22) as follows. Depending on the format, instructions may be 1, 2, or 3 halfwords long. The number of halfwords in an instruction is specified by the first two bits of the op code as follows:

Format	Op Code Positions 0 and 1	Instruction Length in Halfwords
RR	00	1
RX	01	2
RS and SI	10	2
SS	11	3

Because the op code of the next instruction to be executed is always in R, its format (positions 0 and 1) can be predecoded to determine the number of halfwords that compose that instruction and thus indicate which of the four Q-register halfwords contains the *next* sequential instruction op code. This predecoding occurs at end-op time of each instruction; the result (Q halfword number) is set into IC(21,22), which in turn selects a subsequent I-Fetch ROS word that specifies the next op-code Q-halfword to be transferred to R. The IC(21,22) values associated with each Q-register halfword are illustrated in Figure 1-28.

R(8–11) or R(12–15) is sent to LS address register to prefetch an operand for RR format instructions during end op of the preceding instruction. This transfer can be done

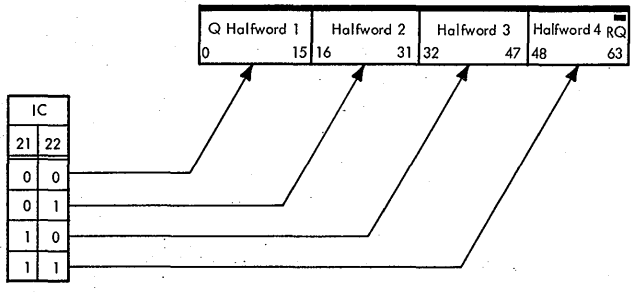


Figure 1-28. Q-Register Halfword Outgating per IC(21,22)

from R rather than from Q because RR instructions are completely contained in R; eight additional paths from Q to LAL are, therefore, not needed.

E-Register

The E-register (Diagram 3-2) is a halfword (16 bits plus two parity bits) trigger register that contains the first halfword (op-code halfword) of the instruction being executed. Portions of the op-code halfword in E are transferred to LAL, the op-code decoder, the parallel adder, the E-register incrementer, and, if the Direct Control feature is installed, an external device. The contents of the E-register are parity-checked.

Instruction Counter

- IC is divided into two sections: IC(0–20) and IC(21–23).
- IC(0–20) addresses a doubleword from main storage.
- IC(21) specifies left or right word within accessed doubleword: IC(21) = 1, select right word; IC(21) = 0, select left word.
- IC(22) selects left or right halfword from selected word: IC(22) = 1, select right halfword; IC(22) = 0, select left halfword.
- IC(21,22) specifies Q-register halfword that contains op code of next instruction to be executed.
- During VFL operations, IC(21–23) specifies addressed byte within doubleword addressed by IC(0–20).

The Instruction Counter (IC), Diagram 3-2, is a 24-bit trigger register used primarily for accessing the next sequential doubleword of instructions from main storage (excluding those specified by branch instructions, which are handled by the D-register). Source operand data is also addressed by the IC during VFL instructions.

The IC is divided into two logical sections: IC(0–20) and IC(21–23). These sections function in the following manner. The main storage area used with the CPU is addressable on a byte (eight-bit) basis, each address placed into the IC referring to a particular byte. However, because the Q-register is of doubleword (64-bit) length, instructions

are accessed from main storage in doubleword (eight-byte) groups. The address of the first byte of each doubleword is all that is required in accessing these doublewords from main storage, and this address is obtained from positions IC(0–20), regardless of the complete address. [Any address represented only by IC(0–20) is a multiple of 8 and lies on a doubleword integral boundary.]

Following the accessing of each doubleword from main storage, IC(0–20) is incremented by 8 (via the parallel adder) to develop the next sequential doubleword address in main storage (eight byte addresses ahead of the doubleword previously accessed).

Once the doubleword addressed by IC(0–20) is read into the CPU, the remaining portion of that complete address [IC(21–23)] selects either instruction halfwords or data bytes from the doubleword. When instructions are addressed by the IC, IC(21,22) only is used to extract the op-code halfword of the addressed instruction in the Q-register; IC(21) selects the right or left word within the doubleword, and IC(22) then selects the right or left halfword from the specified word. (In both cases, a 1 specifies the right portion and a 0 the left portion.) IC(21,22) values of 00, 01, 10, and 11 correspond respectively to the four (1–4) Q-register halfword portions. Figure 1-29 illustrates the Q-register halfword selection for a specified main storage instruction address of 468 (1D4 hex or 111010100 binary).

Note: Because instructions are restricted to even-numbered storage locations, IC(23) must always contain a 0 during instruction addressing. Detection of a 1 in IC(23) during instruction addressing produces an exceptional condition followed by a program interruption.

At end op of each instruction, the format of the instruction just transferred from Q and its location in Q [per IC(21,22)] are examined to determine the location in Q of the op-code halfword of the next sequential instruction.

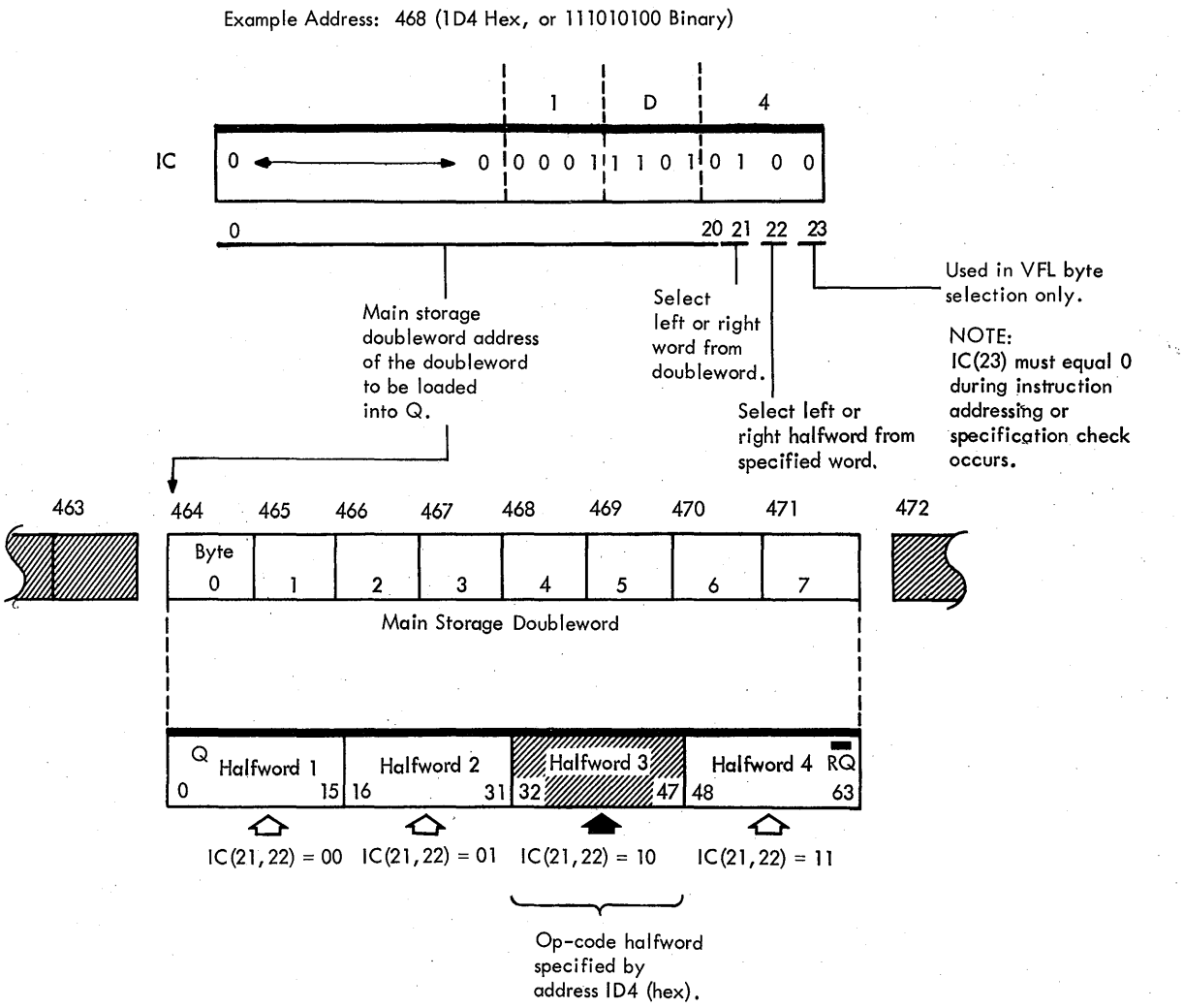


Figure 1-29. Instruction Addressing

Both format and Q-register location must be considered to avoid transferring the remaining non-op-code halfwords of a multi-halfword instruction (RX, RS, SI, or SS) to R.

When the IC is used for addressing source operands during VFL operations, doublewords containing the addressed byte(s) are referenced by IC(0–20) in the same manner as in instruction addressing. However, the accessed doubleword is read out to AB instead of to Q. IC(21–23) then specifies the addressed byte within this doubleword to be gated to the serial adder. [The initial IC(21–23) value is set into the AB counter, which is incremented or decremented, as required, to perform right-to-left or left-to-right processing of the data in AB.] Figure 1-30 illustrates the byte selection as determined by IC(21–23).

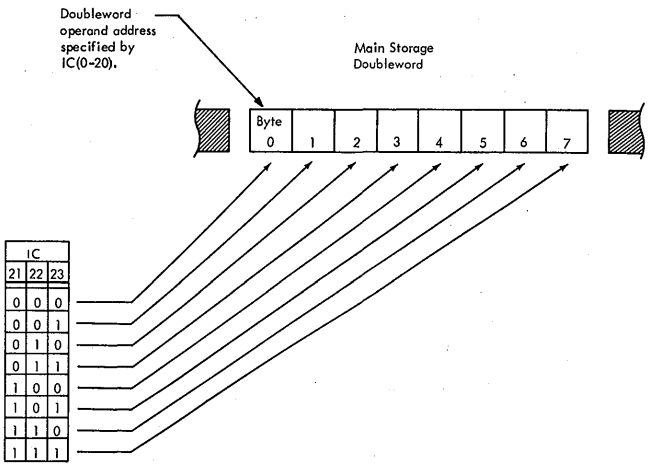


Figure 1-30. Operand Data Byte Selection per IC(21-23)

Note: During VFL operation, the instruction address contained in the IC is temporarily stored into the LS working register (LSWR).

Storage requests are generated to access the next doubleword whenever IC(21–23) indicates that all the information in the present doubleword (op-code halfword in Q-register or data bytes in AB) has been processed.

D-Register

The D-register (Diagram 3-2) is a 24-bit (plus three parity bits) trigger register that functions as a main storage address register during manual-control, branching, and certain arithmetic operations, and as a channel and unit address register during I/O instructions. When addressing main storage, D(0–20) references a doubleword; D(21,22) then extracts the desired instruction halfword and D(23) extracts the desired byte, depending on the operation.

For RS instructions, the I-Fetch routine adds the base and displacement values and places the result into D. Normally, this result is the effective second operand address. For shift instructions, however, this total specifies the number of bit positions to be shifted.

In the Stopped state, D contains the main storage address of the next instruction to be executed (address of instruction in R). (This address is generated and placed into D by the stop-loop microprogram that is in process whenever the CPU enters the Stopped state.) The stop-loop routine subtracts 8 or 16 (decimal) from the updated IC address and places the result into D. In this case, D(0–20) indicates the doubleword address of the instructions in Q, and D(21,22) specifies the location of the op-code halfword within that doubleword.

Instruction Path

- Instructions are fetched into Q from main storage four halfwords at a time.
- R contains first halfword of instruction to be executed next.
- E contains first halfword of instruction being executed.
- IC specifies location in main storage from which next instructions will be fetched and also instruction in Q to be executed next:
  IC(0–20) addresses main storage.
  IC(21,22) indicates which op-code halfword in Q has been transferred to R and is to be executed next.
  IC(23) must be 0 when addressing instructions.

The basic path for instructions entering the CPU is illustrated in Figure 1-31. The first register in the instruction path is the four-halfword instruction buffer called the Q-register. For each access, four instruction halfwords are fetched from a doubleword location in main storage (addressed by the IC) and loaded into the Q-register from the SDBO. Because instructions can vary from one to three halfwords in length, as many as four complete instructions (RR format) or as few as 1-1/3 instructions (SS format) may reside in the Q-register.

Instructions in the Q-register are sequentially selected for processing by means of IC(21,22), which indicates the first halfword (the halfword containing the op code) of the instruction to be executed next. The op-code halfword thus selected is transferred to the R-register, where format predecoding takes place during the end-op cycle. If the instruction is of the single halfword RR-format, the R-register contains the entire instruction. In the case of a two- or three-halfword instruction (RX, RS, SI, or SS format), the R-register contains only the first halfword; the balance (second or second and third halfwords) is not transferred but remains in the Q-register. For this reason, each halfword field of the Q-register is equipped with appropriate transfer paths for processing of the B and D fields of the instruction.

The format of the upcoming instruction (in R) is established by examining R(0,1). This predecoding groups

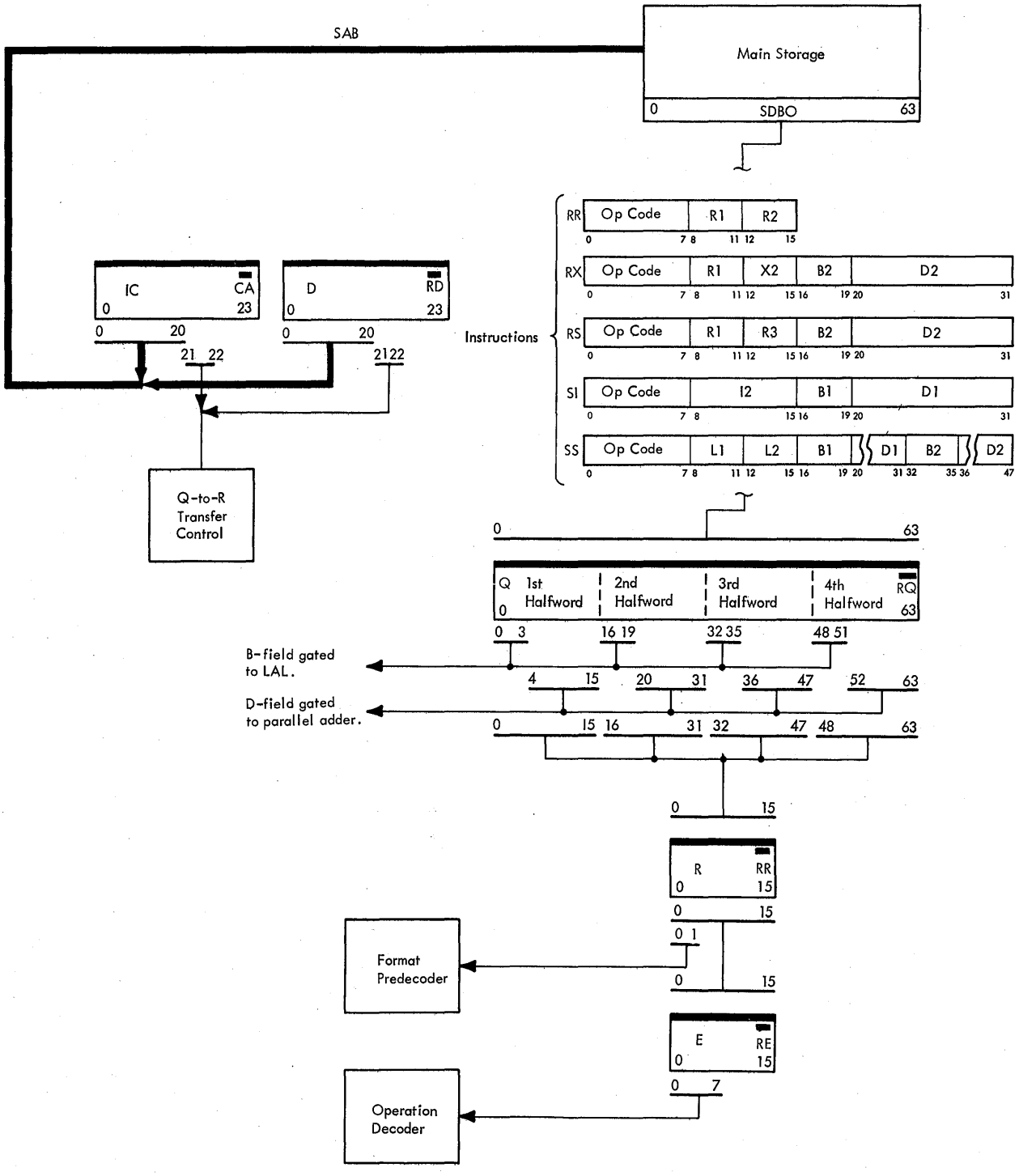


Figure 1-31. Basic Instruction Path

the instructions into four general categories (RR, RX, RS or SI, and SS) to allow loading of the appropriate data registers with operands and operand addresses. Thus operand prefetching is initiated before execution time.

The 16-bit E-register (Figure 1-31) contains the op-code halfword of the instruction presently being executed. This halfword remains in the E-register until the execution phase is completed, at which time it is replaced by the op-code halfword of the next instruction. During each execution phase, the instruction op code contained in E(0–7) is decoded and the specific operations necessary to execute the instruction are performed.

The functions performed during the end-op cycle and the I-Fetch sequence are implemented while the instruction halfwords are in the Q-, R-, and E-registers. The path and the movement of the op-code halfword between the registers for the five formats are shown in Figure 1-32. To illustrate, the following paragraphs trace an RR instruction through the Q-, R-, and E-registers (Figure 1-32, A). Note that an RR instruction is composed of only the op-code halfword; therefore, the complete instruction fits in the R- and E-registers. For the other formats, only the op-code halfword moves through the R- and E-registers; the other halfwords are not transferred from the Q-register. For the example, assume that:

1. All instructions have the RR format.
2. The instruction being executed is No. 4, the next instruction is No. 5, the following instruction is No. 6, and so on.
3. Instruction No. 5 is the one under consideration.

During the I-Fetch sequence of No. 4, instruction No. 5 is placed into the R-register. Here, during the end-op cycle of No. 4, the format of No. 5 is established, operand prefetching is initiated, No. 5 is transferred to the E-register, and the I-Fetch sequence for No. 5 is entered.

During the I-Fetch sequence for No. 5, prefetching of operands for No. 5 is completed, the execution routine for No. 5 is established, and No. 6 is transferred to the R-register. The CPU enters the execution phase for No. 5. During its end-op cycle, the condition code is set (if applicable) and any exceptional conditions and interruptions are detected. The remaining functions performed during the end-op cycle of No. 5 are devoted to initiating the I-Fetch sequence for No. 6.

Prefetching of Operands

- Operand prefetching starts before instruction execution.
- Depending on instruction format, operands are in LS or main storage:
  RR – both operands are in LS.
  RX, RS, SI – one operand is in LS, the other is in main storage.
  SS – both operands are in main storage.
- Address computation for main storage operands always starts first.

To increase the speed of instruction processing, the operands and operand addresses specified in the upcoming instruction are assembled into appropriate registers. For RR instructions, the operands are obtained directly from the LS. For instructions specifying operand addresses in main storage, address calculations take place, and the D-register prefetches an operand from main storage.

The major registers employed for operand prefetching are shown in Figure 1-33. Prefetching of operands begins when the op-code halfword of the instruction is in R and is completed after this halfword has been transferred to E. R(0,1) establishes the instruction format and, consequently, the type of operand fetch that must be performed.

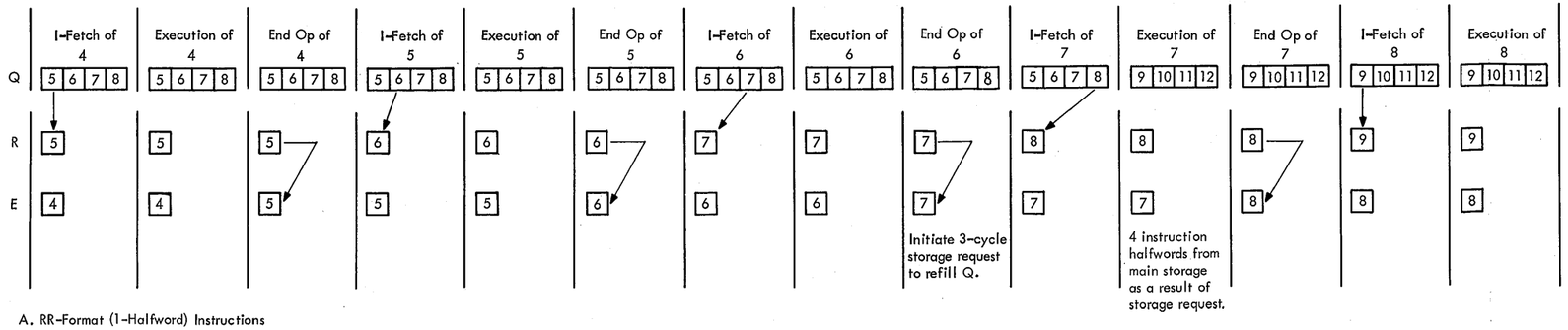
For one-halfword instructions (RR format), R contains the entire instruction. The first operand is fetched by transferring R(8–11) to LAL. After the first operand is retrieved from the LS, it is usually placed into A, B, and D. The second operand is usually fetched after the instruction is transferred to E by transferring E(12–15) to LAL. When the second operand is accessed, it is normally placed into S and T.

For two-halfword instructions (RX, RS, and SI formats), the first halfword is transferred to R while the second halfword is processed directly from Q. Address calculation for the operand in main storage is performed first so that this operand may be requested as soon as possible. This calculation is accomplished by transferring the appropriate B-field from Q to LAL. If the B-field is not zero, the contents of the LS register specified by the B-field are then routed to the parallel adder, where they are added to the D-field (transferred directly from Q). The sum constitutes the operand address specified by RS and SI instructions. This address is transferred to D, from which a storage request for the operand is made.

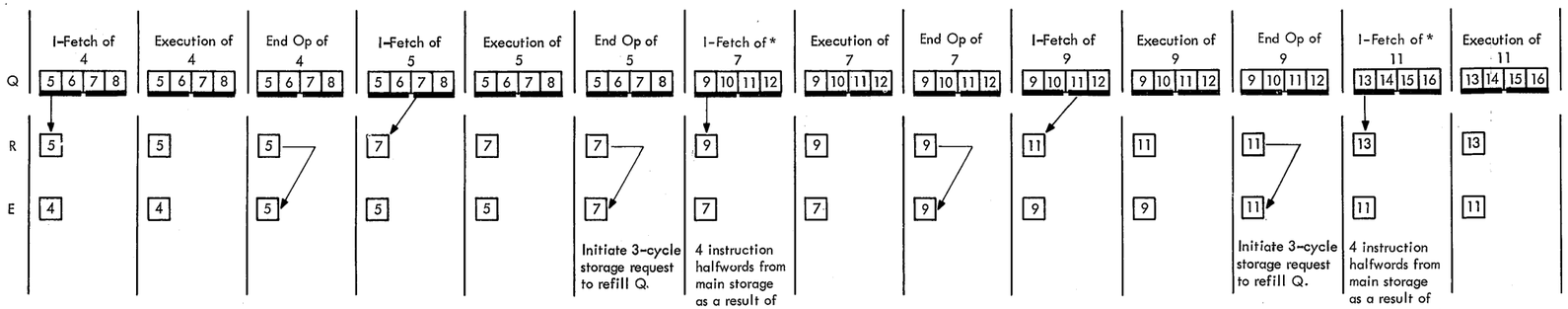
For indexed RX instructions, that is, when the X2 field is not zero, an additional step is required to derive the operand address. Consequently, the partial sum (LS contents per B-field, plus D-field) is temporarily stored into B. The LS is then addressed by the X2 field of the instruction. At this time, the instruction op-code halfword is in E, with E(12–15) containing the X2 field. The contents of the LS register accessed by the X2 field are then summed with the contents of B in the parallel adder to obtain the operand address. This address is transferred to D, and a storage request for the operand is initiated.

The operand setup for two-halfword instructions is completed by fetching the first operand from the LS (not used by SI instructions). This action is performed by transferring E(8–11) to LAL. The first operand is usually loaded into A and B. S and T are usually loaded with the second operand when it arrives on the SDBO during the execution phase.

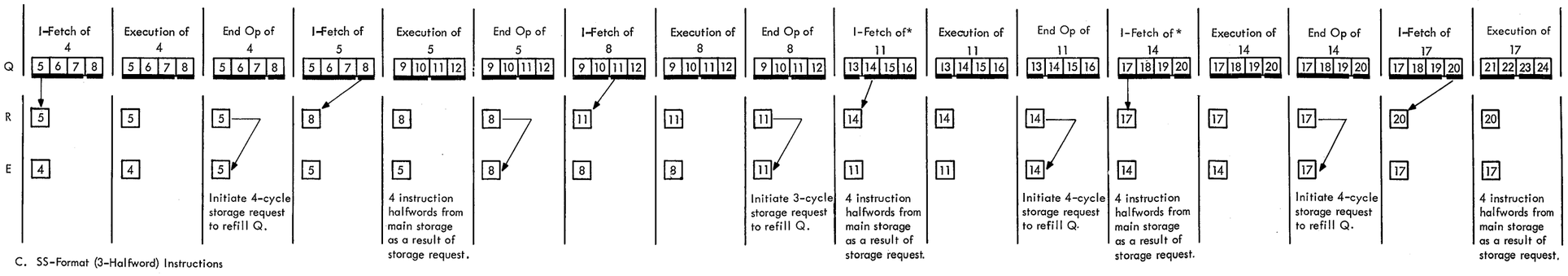
For three-halfword instructions (SS format), the first halfword is transferred to R and the remaining two halfwords are processed directly from Q. The main storage addresses for the first and second operands are calculated in a manner similar to that of two-halfword instructions. The first-operand address is computed first and loaded into D, and a storage request to prefetch the operand is initiated. The second-operand address is then computed while the contents of the IC are transferred (via the parallel adder) to the LSWR. When the second-operand address is computed, the address is loaded into the IC, from which a storage request for the operand is later made. Upon execution of an SS instruction, the instruction address is restored to the IC so that it again selects the next instruction.



A. RR-Format (1-Halfword) Instructions



B. RX-, RS-, and SI-Format (2-Halfword) Instructions



C. SS-Format (3-Halfword) Instructions

Figure 1-32. Path Through Q-, R-, and E-Registers of Op-Code Halfword

Note: Numbers indicate halfword sequence referenced to their location in Q.

For 2- and 3-halfword instructions, lines at bottom of Q group the 4 halfwords into instructions or portions of instructions.

Movement is referenced to associated ROS control word.

* Delay transferring to R until instruction halfwords arrive from main storage.

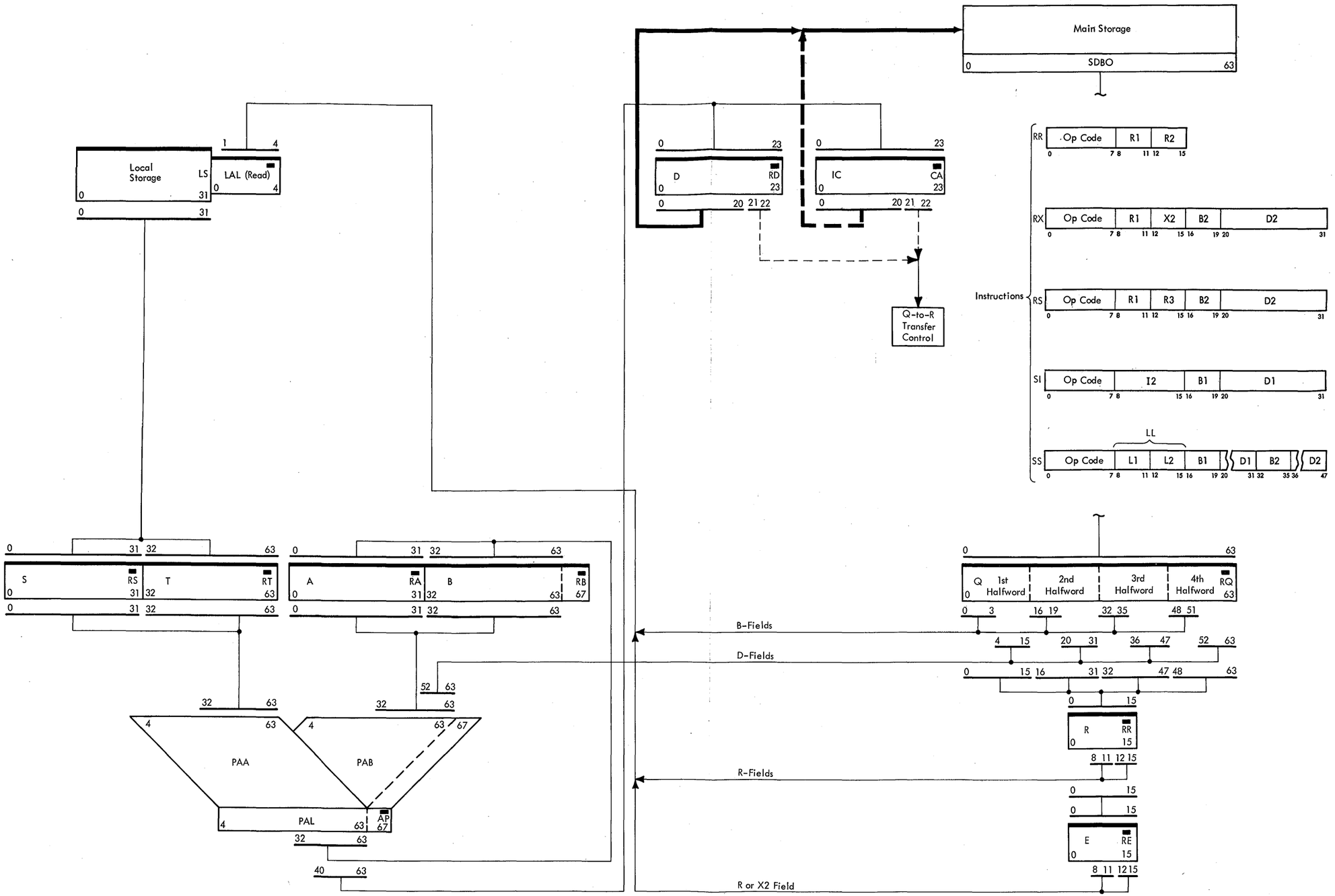


Figure 1-33. Basic Scheme for Operand Prefetching

Obtaining New Instructions from Main Storage

- Requests for new instructions are made before CPU exhausts instructions in Q.
- Usually, three- or four-cycle requests are made from IC.
- Settings of IC(21,22) and R(0,1) determine whether request is needed.
- IC is incremented by 8 after each request.
- For branch instructions, requests are made from D.
- If branch is unsuccessful, instructions accessed by request are ignored.
- If branch is successful, instructions are used and instruction address is transferred from D to IC.

Requests to refill Q with new instructions overlap most of the storage access time with processing of the remaining instructions in Q. The basic scheme used in requesting new instructions from main storage is shown in Figure 1-34.

During normal instruction sequencing, IC(21,22) is examined to establish the number of halfwords in Q that remain to be processed. These bits indicate the op-code halfword of the instruction that has been transferred to R and is to be executed next. Depending on the format of the upcoming instruction decoded from R(0,1), a request for new instructions may be initiated when 2, 3, or all 4 halfwords in Q remain to be processed. The time required to access new instructions from main storage is then used to process the remaining instruction(s) in Q. This access time is specified by the type of request generated by the CPU. Depending on its instruction status, the CPU can generate a three- or a four-cycle request. When a three-cycle request is issued, three CPU cycles must elapse before instructions arrive from main storage. Thus, new instructions are gated into the CPU on the fourth cycle following the request. Similarly, when a four-cycle request is generated, new instructions are gated into the CPU on the fifth cycle following the request. The difference in access times for a three- and four-cycle request is illustrated in Figure 1-34.

IC(0–20) normally specifies the address of new instructions to be accessed from main storage. When it is established that the CPU needs instructions, a request is made and IC(0–20) is transferred to the SAB. The IC is then incremented to address the next successive storage location from which subsequent instructions are to be fetched. (Because the BCU dictates that the address of each successive main storage location must be valid in the IC for at least two cycles, updating of the instruction address is not initiated until two cycles following the request.) Depending on the format of the upcoming instruction, incrementing of the IC is controlled by the CPU hardware (for all formats except SS) or by the ROS microprogram (for SS format). In either case, the IC is incremented by transferring its contents to the parallel adder, where a 1 is added to IC(20) (equivalent to advancing the IC address by 8). The updated address is then routed back to the IC so that a new storage request may be initiated immediately upon detecting the need to refill Q.

A departure from the normal sequencing described above occurs when the instruction being fetched is a branch instruction. To anticipate branch instructions, R (which always contains the first halfword of an upcoming instruction) is connected to a branch predecoder. Execution of a branch instruction may alter the main storage address from which new instructions are to be fetched. If the branch instruction is successful, the address specified by that branch becomes the new instruction address. If, however, the branch is not successful, the address specified by that branch must be ignored. Because it is assumed that branch instructions are successful (the only exception is the Branch on Condition instructions), a request for instructions is initiated as soon as the address specified by the branch is placed into D. Thus, a request is made and the contents of D are transferred to the SAB before establishing that the branch is indeed successful. If it is later found that the branch instruction is not successful, the instructions accessed by that branch are not transferred into Q and a new storage request is generated from the IC, if necessary. Otherwise, upon establishing a successful branch, the contents of D are transferred to the parallel adder, incremented by 8, and transferred to the IC. Normal sequencing is resumed by the IC until another branch instruction is encountered.

The Branch on Condition instructions must be treated differently from other branch instructions. For this reason, a separate detection circuit is provided to anticipate this branch. Whether a Branch on Condition instruction is successful depends on the condition code setting established by a previous instruction. Therefore, the outcome of this branch instruction is known at the time of the request. If the branch is unsuccessful, the request from D is inhibited and, if Q needs refilling, an IC request is generated instead.

Another instruction that falls in the branch category and requires unique treatment is the Execute instruction. This instruction designates a single instruction (subject instruction) to be inserted into the instruction sequence. Briefly, the Execute instruction initiates the following I-Fetch actions:

1. During I-Fetch of an Execute instruction (RX format), IC(21,22) is advanced to indicate the first halfword of the next instruction; that is, the instruction immediately following the Execute instruction in Q.
2. The address of the subject instruction specified by the Execute instruction is computed and placed into D. A storage request from D is then made.
3. When the four instruction halfwords accessed by this request are gated to Q, the subject instruction is selected from Q by examining D(21,22).

4. Upon executing the subject instruction, and if the subject instruction is not a successful branch, a storage request for the instructions previously contained in Q is made from the IC. This request is performed by the program store compare exceptional condition. (If the subject instruction is a successful branch, the request for new instructions is made from D and normal processing is resumed.)
5. When the instructions previously contained in Q are refetched, IC(21,22) is examined to select the instruction following the Execute instruction, and normal processing is resumed.

CPU Interruption and Exceptional Condition Recovery

I-Fetch recognizes the five classes of interruptions: external, supervisor call, program, machine check, and I/O. CPU recovery from these interruptions requires additional processing time for storing the old PSW into main storage and fetching a new PSW and new instructions from main storage into the CPU.

In addition, I-Fetch also recognizes eight exceptional conditions; these conditions require special handling by the CPU. The exceptional conditions and the corresponding actions performed by the CPU are:

Exceptional Condition	CPU Action
Timer	When updating is required, the CPU fetches the timer value from main storage, decrements this value (amount of decrement depends on the line frequency), and stores it back into main storage.
CPU Store in Progress	Extra cycles are added to I-Fetch.
Manual Control Stop	An address is forced into ROSAR that places the CPU into the stop loop.
Manual Control Wait	An address is forced into ROSAR that places the CPU into the Wait state.
Manual Control Repeat	An address is forced into ROSAR that places the CPU into the repeat instruction loop.
Program Store Compare	The CPU obtains the address for the next instruction and refetches it from main storage.
Invalid Instruction Address Test	A program interruption is forced when the CPU tries to use an erroneously addressed instruction.
Q-Register Refill	One extra I-Fetch cycle is added, delaying execution of the next instruction by one CPU cycle.

When one or more interruptions or exceptional conditions occur, processing of the next instruction is delayed until all such conditions are cleared in the order of their priority.

Processing of interruptions and exceptional conditions is initiated during the first cycle of I-Fetch. This cycle issues an 'EXCEP' micro-order to establish if any interruptions or exceptional conditions have resulted from execution of the preceding instruction. If one or more interruptions or exceptional conditions did occur, the 'EXCEP' micro-order overrides the basic I-Fetch actions and transfers control to an appropriate microprogram for clearing the condition that is assigned the highest priority. After processing this condition, the remaining interruptions and exceptional conditions, if any, are handled in the order of their priority. When all interruptions and exceptional conditions have been processed, the microprogram resumes the I-Fetch sequence for the next instruction. The invalid instruction address test and conditions do not override the basic I-Fetch actions but instead augment them.

INSTRUCTION EXECUTION

The execution phase of instruction processing manipulates the data to execute the instruction prepared by I-Fetch. The following paragraphs introduce the functional units that are used primarily in the execution phase and discuss the seven classes of instructions: fixed-point, floating-point, decimal, logical, branching, status switching, and I/O.

Functional Units Used

The following discussion of functional units is based on Diagram 3-2, FEMDM. For a detailed discussion, refer to Chapter 2.

AB Register

The AB register is a 64-bit (plus eight parity bits) trigger register that functions as a working register and also as a buffer for doubleword operands received from main storage.

The AB register is logically divided into two 32-bit (plus four parity bits) registers, A and B. A four-position extension, B(64–67), provides for retaining low-order significance during certain arithmetic and shifting operations.

Byte gating into and out of A and B facilitates their use with the serial adder for VFL operations. A three-position counter, the AB byte counter (ABC), controls the outgating selection of the eight bytes in the AB. AB information is also processed in the parallel adder. Both A and B outgating controls are capable of shifting data left two positions en route to the parallel adder. (Combined shifting capabilities of ST, AB, and the parallel adder thus enable any amount of shift in any direction.)

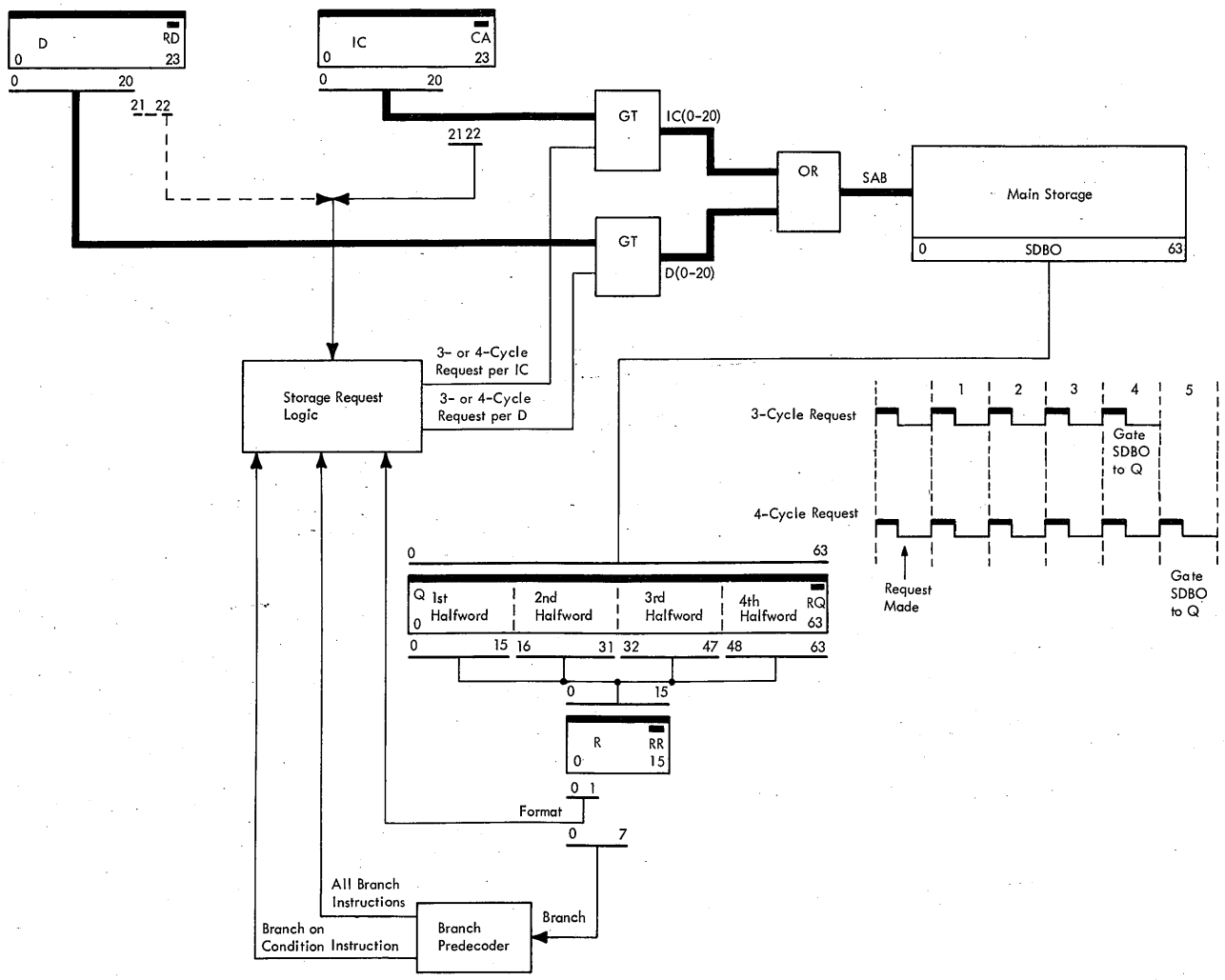


Figure 1-34. Q-Register Refill Addressing Scheme

ST Register

The ST register is a 64-bit (plus eight parity bits) trigger register that functions as an operand buffer between main storage, LS, and the CPU, and also as a working register for arithmetic and logical operations.

The ST register is logically divided into two 32-bit (plus four parity bits) registers, S and T, which serve as working registers for all operations. They also serve as a final assembly area for resultant data to be entered into either LS (T only) or main storage (S and T). Byte gating of ST inputs and outputs facilitates their use with the serial adder for VFL operations. A three-position ST byte counter (STC) and incrementer control the gating of the eight ST bytes.

ST information can also be processed in the parallel adder. T-data can be sent to the parallel adder in either true or complement form, and with a left 1 shift for certain operations. (Scan operations also utilize ST inputs from the parallel adder.) Multiply logic extracts data, in byte lengths, from S via the multiplier bus, and PSW information is sampled from ST(0–39) via the PSW bus.

AB and ST Byte Counters

For operations involving the serial adder, it must be possible to extract bytes from doublewords contained in AB and ST and to assemble bytes in ST for subsequent storage. These capabilities are provided by two byte counters: the ABC for controlling AB byte transfer and the STC for controlling ST byte input and output.

Mark Triggers

Eight mark triggers are contained within the CPU. During store-data operations, these triggers indicate to main storage which bytes of doubleword data placed on the SDBI are to enter storage. The mark triggers thus serve to store CPU data into main storage on a byte (eight-bit plus parity) basis.

An active mark bus line specifies the corresponding byte for storage entry: mark trigger 0 specifies byte 0, or SDBI(0–7); mark trigger 1 specifies byte 1, or SDBI (8–15); and so on, through mark trigger 7, which specifies byte 7 or SDBI(56–63). Any mark trigger not set causes its corresponding byte of original storage data to be regenerated into the addressed location. Complete absence of mark signals on the mark bus occurs only on a fetch operation.

F-Register

The F-register is a one-byte (eight-bit plus parity) trigger register used in certain arithmetic, logical, and data-transfer operations. Functions such as developing quotients, saving floating-point characteristics, converting routines, and processing storage-protection keys are performed in F, and, during direct-control read operations, F serves as a data entry buffer for the external device.

F-register parity circuits generate correct parity for all information received by the register.

G-Register

The G-register is an eight-bit trigger register used for buffering a byte of data between the CPU and an external device during direct-control write operations. G contains no parity bit.

Serial Adder

The serial adder processes information from ST, AB, and F on a byte basis, and is capable of performing binary and decimal arithmetic in addition to logical AND, OR, and Exclusive-OR operations. Other miscellaneous functions performed by the serial adder logic include sign insertion and correction, digit insertion, invalid character detection, zone correction (EBCDIC and USASCII-8), zero and nonzero recognition, and parity adjustment. Parity-predict logic and carry lookahead logic are employed to improve operational speeds, with checking performed on both a half-sum and full-sum basis.

Arithmetic Functions. Highlights:

- Operates on binary or decimal data.
- Processes decimal bytes as two four-bit groups.
- Each four-bit group represents one BCD digit.
- Employs excess-6 arithmetic when processing decimal data.
- Decimal arithmetic results are produced in BCD form.
- Multiply/divide results are assembled in serial adder.

Serial adder logic is capable of performing both binary and decimal arithmetic operations. For binary operations, the adder functions as an eight-bit (plus parity) binary adder, processing bytes from ST and either AB or F and developing eight-bit (plus parity) results. High-order carries resulting from arithmetic operations are stored in Status Trigger (STAT) H for use in processing the next two bytes of that operation.

For decimal operations, each byte of data from ST, AB, or F is treated as two individual four-bit groups, which are then processed with excess-6 arithmetic. This feature provides a programming advantage by enabling the adder to accept data in binary-coded decimal (BCD) form (packed digits) and to produce results which are also in decimal format.

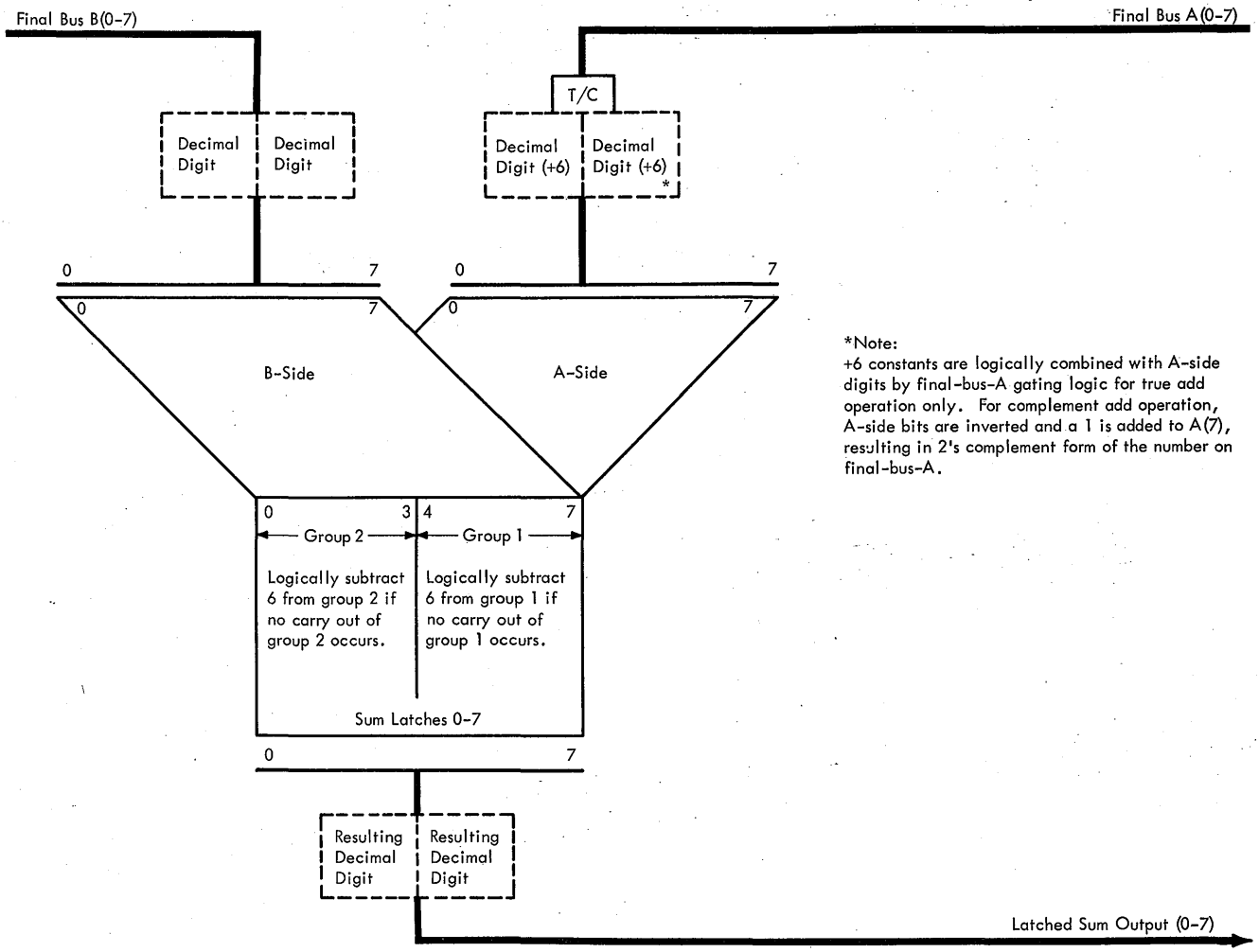
Excess-6 arithmetic involves adding 6 (under ROS control) to incoming first operands. This is necessary to preserve the decimal value in a four-bit binary character. Each binary character (four bits) has a maximum decimal value of 15 (1111 binary), which is 6 more than the maximum valid decimal character of 9 (1001 binary). When the second operand is added and the total exceeds 15 (1111 binary), the carry is a decimal value carry and leaves a correct decimal value in the four binary bits. If no carry occurs after the addition, the character is an erroneous decimal value (it is too large by 6) and the excess 6 is subtracted. The decimal values of the binary character under no-carry conditions are 6–15 (actual value of 0–9 after correction).

The following examples show excess-6 addition. Note that correction occurs when there is no carry irrespective of the decimal validity of the sum.

	1 + 1	3 + 5	6 + 9
Operand 1 (SBA)	0001	0011	0110
Excess-6	0110	0110	0110
SAA	0111	1001	1100
Operand 2 (SBB)	0001	0101	1001
Sum	1000	1110	←0101
Correction	0110	0110	----
Result	0010	1000	←0101
	(2)	(8)	(15)

In the first case (1 + 1) the sum is a valid decimal character (8), but the absence of a carry indicates an erroneous result calling for subtracting the excess 6. The second case (3 + 5) is similar except that the sum is not a decimal character. In the third case (6 + 9) there is a carry into the decimal tens position and the sum (5) does not need correction.

Figure 1-35 shows the excess-6 data paths through the serial adder and illustrates the adder operation by means of the decimal example: 46 + 28 = 74. Note that +6 constants



*Note:
+6 constants are logically combined with A-side digits by final-bus-A gating logic for true add operation only. For complement add operation, A-side bits are inverted and a 1 is added to A(7), resulting in 2's complement form of the number on final-bus-A.

Decimal Arithmetic Example: 46 + 28 = 74

A-side operand byte (46_{10})	0100	0110
Convert A-side digits to excess-6 (Logically add +6 to both groups at final bus.)	0110	0110
A-side digits in excess-6 value	1010	1100
B-side operand byte (28_{10})	0010	1000
	← Group 2 →	← Group 1 →
A-side adder entry	1010	1100
B-side adder entry	0010	1000
Logical sum	1101	0100
Decimal correction (Logically subtract 6 from group 2.)	0110	
Decimal result (74_{10})	0111	0100

Decimal Arithmetic Example: 46 - 28 = 18

A-side operand byte (28_{10})	0010	1000
Complement	1101	0111
+1 to A(7)		1
2's complement of A-side digit	1101	1000
B-side operand (46_{10})	0100	0110
	← Group 2 →	← Group 1 →
A-side adder entry	1101	1000
B-side adder entry	0100	0110
Logical sum	0001	1110
Decimal correction (Logically subtract 6 from group 1)		0110
Decimal result (18_{10})	0001	1000

Figure 1-35. Decimal Format Serial-Adder Data Flow

are logically added to the A-side digits that are in packed format before entry into the adder (by final-bus-A-gating logic), and that subtraction of the +6 constants (decimal-correct) is performed on each four-bit group in which a group carry (carry from high-order position of group) does *not* occur as a result of the arithmetic operation.

When a ROS micro-order calls for complement add, the data entering SAA is converted to 2's-complement form. This conversion is accomplished by complementing the bit configuration on the A-side entry and adding a hot carry to the input of SAL(7). For complement add decimal operation, +6 constants are not combined with SBA inputs. When complementing BCD, the excess-6 is effectively added because the resultant complement is a character based on 16 rather than 10. To illustrate, the 10's-complement of 7 is 3, but the 2's-complement of 7 (0111) is 9 (1001) or 6 more than the 10's-complement. The addition then occurs as in true +6 add, and the absence of a carry likewise forces decimal correction of the sum. A carry out of the high-order adder position [serial adder bit carry (0)] sets STAT H for use in processing the next data byte of that operation.

Figure 1-35 also shows the serial adder operation for a complement add example: 46 – 28 = 18. Note that +6 constants are not added to the A-side digits, but decimal correction is performed in the same manner as for a true add operation.

Logical Functions. The serial adder also performs logical AND, OR, and Exclusive-OR functions. To implement the logical functions, each bit position of the serial adder is, in effect, a separate unit. The logical functions are defined as follows:

1. AND. If *both* operand bits are 1's, the resulting bit is a 1; otherwise, the result is a 0. (Carries between bits are suppressed.)
2. OR. If *either* operand bit is a 1, the resulting bit is a 1; otherwise, the result is a 0. (Carries between bits are suppressed.)
3. Exclusive-OR. If *one and only one* of the operand bits is a 1, the resulting bit is a 1; otherwise, the result is a 0. (Carries between bits are suppressed.)

If the conditions for the corresponding function is met, the associated serial adder latch is set.

Parallel Adder

- 60-bit (plus parity) full-binary adder.
- Inputs are from S, T, D, A, B, Q, IC, E, and F.
- Inputs from T and D are 2's complemented for subtract and compare operations and address updating.
- Output data can be shifted left 4 or right 4 into the parallel adder latches; parity is adjusted accordingly.
- Adder employs carry-lookahead and parity-predict logic.
- Adder includes half-sum and full-sum error checking.

The parallel adder is a 60-bit (plus parity) full-binary arithmetic unit. In addition to arithmetic functions, the parallel adder performs certain logical operations (e.g., convert routines) and is involved in most intra-CPU data transfers. Correct parity (odd) is generated with all adder output data. Immediate left 4 and right 4 shifting capabilities are available at the adder output, with parity adjusted accordingly. Error-checking facilities within the adder provide for validity checking of both incoming operands and full-sum results.

The parallel adder has true-complement gating controls on adder entries from T and D. For subtract and compare instructions, operands from T are 2's complemented and presented to the A-side of the adder. (The binary bits are inverted and a hot-carry is added to position 63.) From this point, add and subtract operations are the same. Complement entries from D are used for address compare and address update operations and for floating-point operations.

The parallel adder has full-binary capabilities (half-adder and full-sum functions), with immediate left-4/right-4 shift logic included on its output to facilitate data shifting without the need of an additional machine cycle. Figure 1-36 illustrates the logical functions of the parallel adder. Note the four-position adder extension, PAB(64–67); it serves to retain low-order significance during certain right-shift operations.

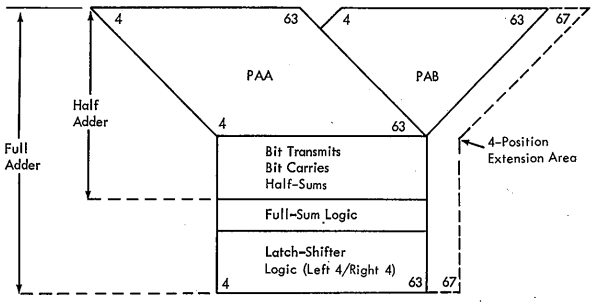


Figure 1-36. Parallel Adder Logical Functions

Half-adder functions supply information concerning incoming operands for use in both checking the validity of the operands and producing carry information for full-sum development. Full-sum logic combines half-adder outputs with carry information to produce the final or full-sum result. Parity information is also developed by logically combining half-adder and carry functions and is normally supplied on a byte basis, although certain adder areas (bits 4–8 and 64–67) require half-byte or four-bit group parity.

Carry-lookahead and parity-predict facilities are employed within the parallel adder. These features provide for the immediate development of full-sum results (and parity), without the need for additional cycles in which to incorporate carry information and generate parity. The lookahead and predict circuitry is implemented through logic in which the 60 bit positions are arranged in four-bit

groups, and these groups divided into four sections. Figure 1-37 illustrates the logical grouping of the 60 bit positions.

All adder results are checked using half-sum and full-sum error-checking logic. Half-sum checking determines the validity of incoming operands by comparing the odd/even bit count with the assigned parity. Full-sum checking involves comparing the full-sum resultant bit count with the independently generated full-sum parity; an inconsistency in either causes a full-sum error.

Local Storage

A high-speed transistor storage area, local storage (LS), is located within the CPU to reduce the number of main storage references required by the CPU during each operation. The LS consists of 25 registers for use in storing address information, fixed-point, logical, and floating-point operands, and the IC contents (IC contents stored in LS working register, LSWR, only). Local storage data is available to the CPU at 200-ns intervals. In addition to reducing the main storage reference, this access time increases operational speeds within the CPU.

The 25 LS registers are grouped as follows: 16 (0–15) general-purpose registers (GPR's), 8 (16–23) floating-point registers (FPR's), and 1 (24) working register called the LSWR. Each register contains 32 data (plus 4 parity) positions, and is directly addressable by the R1, R2, R3, B1, B2, and X2 fields of the instructions, with the exception of register 24. Register 24 (LSWR) is not available to the operational program. It is reserved for use by ROS microprograms in manipulating information during execution of certain instructions. (Such applications include temporary storage for the contents of the IC when the IC is to be used as an operand address register, and temporary storage for floating-point second operands while prenormalizing the first operand.)

The eight FPR's (16–23) function as four double-length (64-bit) registers. Each double-length register consists of two single-length (32-bit) registers coupled as follows: 16 and 17, 18 and 19, 20 and 21, and 22 and 23. Only the leftmost 32 bit positions are used in short-operand floating-point instructions, with all 64 positions participating in long-operand floating-point instructions. For either short or long operands, only the leftmost (even-numbered) registers must be addressed.

Data transferred from LS is checked only upon being processed in the adders or when entered into main storage.

Status Triggers

The CPU contains eight commonly available status triggers (STAT's) to record information that may be significant in the execution of present instruction operations. These eight triggers are designated as STAT's A–H, and retain such information as invalid digit-detection, overflow and carry conditions, and negatively signed operands. The STAT's are reset at each I-Fetch.

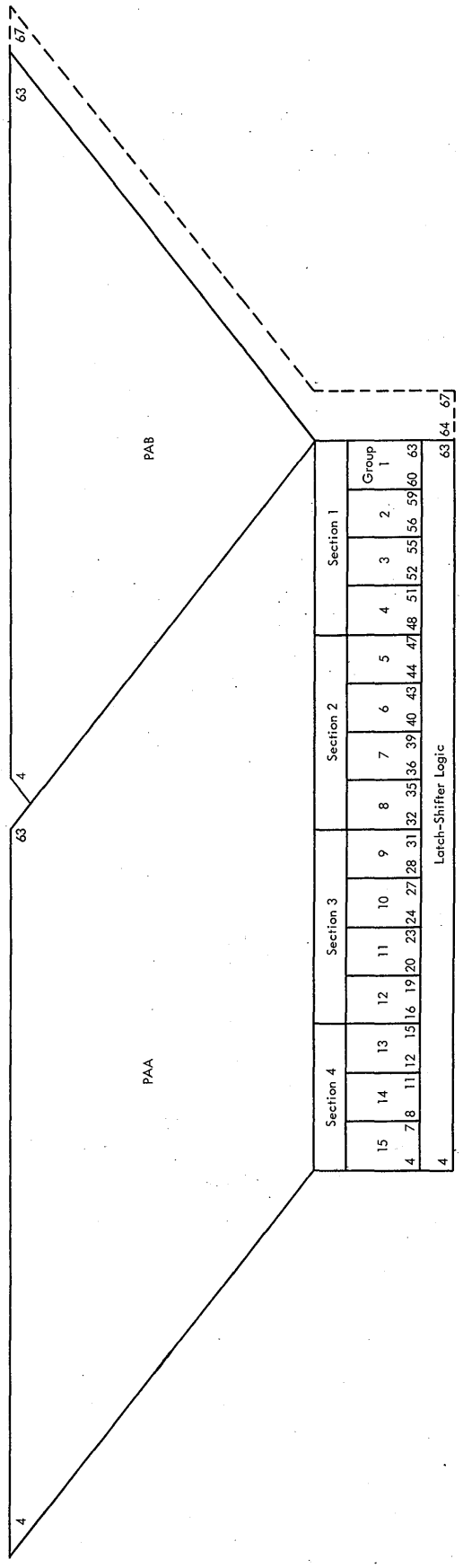


Figure 1-37. Parallel Adder Group/Section Breakdown

Certain STAT's serve multiple functions and are capable of receiving several types of information for use with different instructions. The outputs of these multiple-use triggers are distributed, via line-sense amplifiers, to the CPU areas requiring this information.

Scan operations test STAT's; during scan in, the eight STAT's are set to the state of T(38) and T(54–60).

All STAT's are reset by either 'system reset' or 'I-Fetch reset' signals, with certain STAT's containing additional individual resets.

In general, all STAT's are reset and set during clock time of the basic machine cycle. However, certain clock signals that control these triggers are delayed 180 ns. This delayed operation is necessary to prevent timing problems that could arise if the detected conditions occurred too late to set a STAT with normal clock signals. All clock signals that control the STAT's are inhibited during scan-in operations by an 'FLT inhibit clock' signal.

Fixed-Point Instructions

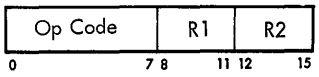
The fixed-point instruction set performs binary arithmetic on operands serving as data, addresses, index quantities, and counts. Instructions are provided for loading, adding, subtracting, comparing, multiplying, dividing, shifting, storing, and converting from binary to decimal and from decimal to binary. Table 1-8 lists the fixed-point instructions.

For a discussion of number representation, data formats, and operand addressing, refer to Section 2 of this chapter.

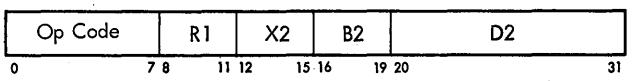
Instruction Formats

The fixed-point instruction set uses three instruction formats:

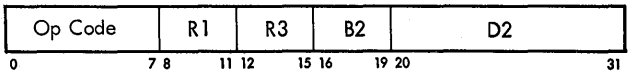
RR



RX



RS



In the RR format, R1 specifies the address of the GPR containing the first operand and R2 specifies the address of the GPR containing the second operand. Both the first and second operands may be specified by the same GPR.

In the RX format, R1 specifies the address of the GPR containing the first operand. The contents of the GPR specified by the X2 and B2 fields are added to the contents of the D2 field to form an address designating the main storage location of the second operand.

In the RS format, R1 specifies the address of the GPR containing the first operand. The contents of the GPR specified by the B2 field are added to the contents of the D2 field to form an address. This address designates the main storage location of the second operand for Load Multiple and Store Multiple instructions. In shift operations, the low-order six bits of the address specify the number of bit positions to be shifted. The R3 field specifies the address of GPR for Load Multiple and Store Multiple instruction and is ignored in the shift operations.

Data Flow

The data flow path for fixed-point operations is shown in Diagram 3-3, FEMDM. The functional units used to perform the major functions for fixed-point operations are:

1. ST. Holds the second operand and assembles data before it is sent to LS or main storage. (T is the only register that can transfer data to the LS.)
2. AB. Holds the first operand and assembles data during an operation.
3. F. Assembles product and quotient bits during multiply and divide operations; holds the binary bits to be converted during convert operations.
4. E. Controls product and quotient derivation; also contains instruction op code and number of shifts when performing shift instructions.
5. Parallel adder. Manipulates the operands to obtain the desired result. Is also the central point in the data path between ST and AB.
6. Serial adder. Calculates product and quotient bytes. Is also the central point in the data path between F and AB and between F and ST.
7. STC. Controls selection of data from and placement of data into ST, primarily during multiply, divide, and convert operations.
8. D. Addresses second operand located in main storage.

Program Interruptions

Six program interruptions can occur during execution of fixed-point instructions. Of the six, only fixed-point overflow can be masked off; the others are unconditionally taken. If the fixed-point overflow mask bit [PSW(36)] is a 0, the fixed-point overflow interruption is ignored; if a 1, it is taken.

The six interruptions and their causes are:

1. Protection. The storage key of a main storage location does not match the storage protection key in the PSW. The instruction is suppressed for a store violation, unless it is the Store Multiple instruction, which is terminated. For a fetch violation, the instruction is terminated.

Table 1-8. Fixed-Point Instructions

Instruction	Mnemonic	Op Code	Format	Operands	Function	Program Interruptions	Condition Code
Add	A	5A	RX	R1 D2(X2, B2)	Algebraically add 2nd opr (in stg) to 1st opr (in GPR per R1) & place result into 1st opr location. D(21) determines which word of doubleword from stg is 2nd opr: if 1, right word; if 0, left word.	Prot (F) Adr Spec Fix-Pt Ovflo	0 : Sum = 0 1 : Sum < 0 2 : Sum > 0 3 : Overflow
Add	AR	1A	RR	R1 R2	Algebraically add 2nd opr (in GPR per R2) to 1st opr (in GPR per R1) & place result into 1st opr location.	Fix-Pt Ovflo	0 : Sum = 0 1 : Sum < 0 2 : Sum > 0 3 : Overflow
Add Halfword	AH	4A	RX	R1 D2(X2, B2)	Algebraically add halfword 2nd opr (in stg) to 1st opr (in GPR per R1) & place result into 1st opr location. 1. D(21) determines which word of doubleword from stg contains halfword 2nd opr: If 1, right word; if 0, left word. 2. D(22) determines which half of word is halfword 2nd opr: If 1, right half; if 0, left half. 3. Halfword 2nd opr is expanded to full word before addition by propagating sign bit through 16 high-order bits.	Prot (F) Adr Spec Fix-Pt Ovflo	0 : Sum = 0 1 : Sum < 0 2 : Sum > 0 3 : Overflow
Add Logical	AL	5E	RX	R1 D2(X2, B2)	Algebraically add 2nd opr (in stg) to 1st opr (in GPR per R1) & place result into 1st opr location. 1. D(21) determines which word of doubleword from stg is 2nd opr: if 1, right word; if 0, left word. 2. Sign bit of result is treated as high-order integer & is tested for carry to determine CC.	Prot (F) Adr Spec	0 : Sum = 0 (no carry) 1 : Sum ≠ 0 (no carry) 2 : Sum = 0 (carry) 3 : Sum ≠ 0 (carry)
Add Logical	ALR	1E	RR	R1 R2	Algebraically add 2nd opr (in GPR per R2) to 1st opr (in GPR per R1) & place result into 1st opr location. Sign bit of result is treated as high-order integer & is tested for carry to determine CC.	None	0 : Sum = 0 (no carry) 1 : Sum ≠ 0 (no carry) 2 : Sum = 0 (carry) 3 : Sum ≠ 0 (carry)
Compare	C	59	RX	R1 D2(X2, B2)	Algebraically compare 1st opr (in GPR per R1) with 2nd opr (in stg) & set CC according to result. D(21) determines which word of doubleword from stg is 2nd opr: if 1, right word; if 0, left word.	Prot (F) Adr Spec	0 : Opr 1 = Opr 2 1 : Opr 1 < Opr 2 2 : Opr 1 > Opr 2
Compare	CR	19	RR	R1 R2	Algebraically compare 1st opr (in GPR per R1) with 2nd opr (in GPR per R2) & set CC according to result.	None	0 : Opr 1 = Opr 2 1 : Opr 1 < Opr 2 2 : Opr 1 > Opr 2
Compare Halfword	CH	49	RX	R1 D2(X2, B2)	Algebraically compare 1st opr (in GPR per R1) with halfword 2nd opr (in stg) & set CC according to result. 1. D(21) determines which word of doubleword from stg contains halfword 2nd opr: if 1, right word; if 0, left word. 2. D(22) determines which half of word is halfword 2nd opr: if 1, right half; if 0, left half. 3. Halfword 2nd opr is expanded to full word before comparison by propagating sign bit through 16 high-order bits.	Prot (F) Adr Spec	0 : Opr 1 = Opr 2 1 : Opr 1 < Opr 2 2 : Opr 1 > Opr 2
Convert to Binary	CVB	4F	RX	R1 D2(X2, B2)	Convert radix of 2nd opr (in stg) from decimal to binary & place result into 1st opr location (in GPR per R1). 1. 2nd opr is doubleword in packed format. 2. High-order word is converted first. 3. Max positive integer that can be converted is +2,147,483,647. 4. Max negative integer that can be converted is -2,147,483,648.	Prot (F) Adr Spec Data Fix-Pt Div	Unchanged

Table 1-8. Fixed-Point Instructions (Cont)

Instruction	Mnemonic	Op Code	Format	Operands	Function	Program Interruptions	Condition Code
Convert to Decimal	CVD	4E	RX	R1 D2(X2, B2)	Convert radix of 1st opr (in GPR per R1) from binary to decimal & place result into 2nd opr location (in stg). 1. Result is in packed format on doubleword boundary. 2. Low-order 4 bits of field are sign. 3. If PSW(12) = 1, use USASCII-8 code for sign; if PSW(12) = 0, use EBCDIC code.	Prot (S) Adr Spec	Unchanged
Divide	D	5D	RX	R1 D2(X2, B2)	Divide 1st opr (in GPR per R1 & R1 + 1) by 2nd opr (in stg) & place result into 1st opr location (remainder in GPR per R1; quotient in GPR per R1 + 1). 1. R1 must be even adr. 2. D(21) determines which word of doubleword from stg is divisor: if 1, right word; if 0, left word. 3. Relative value of opr's must result in quotient expressible in 32-bit signed integer. 4. Sign of quotient is determined algebraically, except 0 quotient is positive. 5. Sign of remainder is same as sign of dividend, except 0 remainder is positive.	Prot (F) Adr Spec Fix-Pt Div	Unchanged
Divide	DR	1D	RR	R1 R2	Divide 1st opr (in GPR per R1 & R1 + 1) by 2nd opr (in GPR per R2) & place result into 1st opr location (remainder in GPR per R1; quotient in GPR per R1 + 1). 1. R1 must be even adr. 2. Relative value of opr's must result in quotient expressible in 32-bit signed integer. 3. Sign of quotient is determined algebraically, except 0 quotient is positive. 4. Sign of remainder is same as sign of dividend, except 0 remainder is positive.	Spec Fix-Pt Div	Unchanged
Load	L	58	RX	R1 D2(X2, B2)	Load 2nd opr (in stg) into 1st opr location (in GPR per R1). 1. D(21) determines which word of doubleword from stg is to be stored: if 1, right word; if 0, left word. 2. 2nd opr is unchanged.	Prot (F) Adr Spec	Unchanged
Load	LR	18	RR	R1 R2	Load 2nd opr (in GPR per R2) into 1st opr location (in GPR per R1). 2nd opr is unchanged.	None	Unchanged
Load & Test	LTR	12	RR	R1 R2	Load 2nd opr (in GPR per R2) into 1st opr location (in GPR per R1) & set CC according to result. 2nd opr is unchanged.	None	0 : Result = 0 1 : Result <0 2 : Result >0
Load Complement	LCR	13	RR	R1 R2	Load 2's complement of 2nd opr (in GPR per R2) into 1st opr location (in GPR per R1) & set CC according to result. Overflow occurs only if max negative number is 2's complemented.	Fix-Pt Ovflo	0 : Result = 0 1 : Result <0 2 : Result >0 3 : Overflow
Load Halfword	LH	48	RX	R1 D2(X2, B2)	Load halfword 2nd opr (in stg) into 1st opr location (in GPR per R1). 1. D(21) determines which word of doubleword from stg contains halfword 2nd opr: if 1, right word; if 0, left word. 2. D(22) determines which half of word is halfword 2nd opr: if 1, right half; if 0, left half. 3. Halfword 2nd opr is expanded to full word before loading by propagating sign bit through 16 high-order bits.	Prot (F) Adr Spec	Unchanged

Table 1-8. Fixed-Point Instructions (Cont)

Instruction	Mnemonic	Op Code	Format	Operands	Function	Program Interruptions	Condition Code
Load Multiple	LM	98	RS	R1 R3 D2(B2)	Load 2nd opr (as many words as required; in stg) into GPR's, in ascending order, starting with 1st opr location (per R1) & ending with 3rd opr location (per R3). 1. 2nd opr is unchanged. 2. If R1 = R3, only 1 word is loaded. 3. If R3 < R1, GPR adr's wraparound from 15 to 0. 4. D(21) determines which word of doubleword from stg is to be loaded into LS: if 1, right word; if 0, left word.	Prot (F) Adr Spec	Unchanged
Load Negative	LNR	11	RR	R1 R2	Load 2nd opr (unchanged if negative, 2's complemented if positive; in GPR per R2) into 1st opr location (in GPR per R1). If 2nd opr = 0, unchanged with plus sign.	None	0 : Result = 0 1 : Result < 0
Load Positive	LPR	10	RR	R1 R2	Load 2nd opr (unchanged if positive, 2's complemented if negative; in GPR per R2) into 1st opr location (in GPR per R1). Overflow occurs only if max negative number is 2's complemented.	Fix-Pt Ovflo	0 : Result = 0 2 : Result > 0 3 : Overflow
Multiply	M	5C	RX	R1 D2(X2, B2)	Multiply 1st opr (in GPR per R1 + 1) & 2nd opr (in stg) & place 64-bit result into 1st opr location (in GPR per R1 & R1 + 1). 1. R1 must be even adr. 2. D(21) determines which word of doubleword from stg is 2nd opr: if 1, right word; if 0, left word.	Prot (F) Adr Spec	Unchanged
Multiply	MR	1C	RR	R1 R2	Multiply 1st opr (in GPR per R1 + 1) by 2nd opr (in GPR per R2) & place 64-bit result into 1st opr location (in GPR per R1 & R1 + 1). R1 must be even adr.	Spec	Unchanged
Multiply Halfword	MH	4C	RX	R1 D2(X2, B2)	Multiply 1st opr (in GPR per R1) & halfword 2nd opr (in stg) & place low-order 32 bits of result into 1st opr location. 1. D(21) determines which word of doubleword from stg contains halfword 2nd opr: if 1, right word; if 0, left word. 2. D(22) determines which half of word is halfword 2nd opr: if 1, right half; if 0, left half. 3. Halfword 2nd opr is expanded to full word before multiplication by propagating sign bit through 16 high-order bits.	Prot (F) Adr Spec	Unchanged
Shift Left Double	SLDA	8F	RS	R1 D2(B2)	Shift 1st opr (in GPR per R1 & R1 + 1) left number of bit positions specified by low-order 6 bits of 2nd opr adr & place result into 1st opr location. 1. R1 must be even adr. 2. High-order bits of 1st opr are shifted out & lost; low-order vacated bits are made 0's. 3. If bit unlike sign bit is shifted out of bit position 1 of even register, fixed-point overflow occurs.	Spec Fix-Pt Ovflo	0 : Result = 0 1 : Result < 0 2 : Result > 0 3 : Overflow
Shift Left Single	SLA	8B	RS	R1 D2(B2)	Shift 1st opr (in GPR per R1) left number of bit positions specified by low-order 6 bits of 2nd opr adr & place result into 1st opr location. 1. High-order bits of 1st opr are shifted out & lost; low-order vacated bits are made 0's. 2. If bit unlike sign bit is shifted out of bit position 1 of even register, fixed-point overflow occurs.	Fix-Pt Ovflo	0 : Result = 0 1 : Result < 0 2 : Result > 0 3 : Overflow

Table 1-8. Fixed-Point Instructions (Cont)

Instruction	Mnemonic	Op Code	Format	Operands	Function	Program Interruptions	Condition Code
Shift Right Double	SRDA	8E	RS	R1 D2(B2)	Shift 1st opr (in GPR per R1 & R1 + 1) right number of bit positions specified by low-order 6 bits of 2nd opr adr & place result into 1st opr location. 1. R1 must be even adr. 2. Low-order bits of 1st opr are shifted out & lost; high-order vacated bits are made equal to sign bit.	Spec	0 : Result = 0 1 : Result < 0 2 : Result > 0
Shift Right Single	SRA	8A	RS	R1 D2(B2)	Shift 1st opr (in GPR per R1) right number of bit positions specified by low-order 6 bits of 2nd opr adr & place result into 1st opr location. Low-order bits of 1st opr are shifted out & lost; high-order vacated bits are made equal to sign bit.	None	0 : Result = 0 1 : Result < 0 2 : Result > 0
Store	ST	50	RX	R1 D2(X2, B2)	Store 1st opr (in GPR per R1) into 2nd opr location (in stg). 1. PAL(61) determines into which word of doubleword in stg 1st opr is to be stored: if 1, right word; if 0, left word. 2. 1st opr is unchanged.	Prot (S) Adr Spec	Unchanged
Store Halfword	STH	40	RX	R1 D2(X2, B2)	Store halfword 1st opr (in GPR per R1) into 2nd opr location (in stg). 1. ABC selects 16 low-order bits of 1st opr for storage; high-order bits are ignored. 2. STC [D(21–23)] positions 16 low-order bits of 1st opr into doubleword 2nd opr location. 3. 1st opr is unchanged.	Prot (S) Adr Spec	Unchanged
Store Multiple	STM	90	RS	R1 R3 D2(B2)	Store into 2nd opr location (as many words as required; in stg) contents of GPR's, in ascending order, starting with 1st opr location (per R1) & ending with 3rd opr location (per R3). 1. GPR adr's wrap around from 15 to 0. 2. D(21) determines into which word of doubleword in stg contents of 1st GPR are to be stored: if 1, right word; if 0, left word. 3. If R1 = R3, 1 word is stored.	Prot (S) Adr Spec	Unchanged
Subtract	S	5B	RX	R1 D2(X2, B2)	Algebraically subtract 2nd opr (in stg) from 1st opr (in GPR per R1) & place result into 1st opr location. D(21) determines which word of doubleword from stg is 2nd opr: if 1, right word; if 0, left word.	Prot (F) Adr Spec Fix-Pt Ovflo	0 : Dif = 0 1 : Dif < 0 2 : Dif > 0 3 : Overflow
Subtract	SR	1B	RR	R1 R2	Algebraically subtract 2nd opr (in GPR per R2) from 1st opr (in GPR per R1) & place result into 1st opr location.	Fix-Pt Ovflo	0 : Dif = 0 1 : Dif < 0 2 : Dif > 0 3 : Overflow
Subtract Halfword	SH	4B	RX	R1 D2(X2, B2)	Algebraically subtract halfword 2nd opr (in stg) from 1st opr (in GPR per R1) & place result into 1st opr location. 1. D(21) determines which word of doubleword from stg contains halfword 2nd opr: if 1, right word; if 0, left word. 2. D(22) determines which half of word is halfword 2nd opr: if 1, right half; if 0, left half. 3. Halfword 2nd opr is expanded to full word before subtraction by propagating sign bit through 16 high-order bits.	Prot (F) Adr Spec Fix-Pt Ovflo	0 : Dif = 0 1 : Dif < 0 2 : Dif > 0 3 : Overflow

Table 1-8. Fixed-Point Instructions (Cont)

Instruction	Mnemonic	Op Code	Format	Operands	Function	Program Interruptions	Condition Code
Subtract Logical	SL	5F	RX	R1 D2(X2, B2)	Algebraically subtract 2nd opr (in stg) from 1st opr (in GPR per R1) & place result into 1st opr location. 1. D(21) determines which word of doubleword from stg is 2nd opr: if 1, right word; if 0, left word. 2. Sign bit of result is treated as high-order integer & is tested for carry to determine CC.	Prot (F) Adr Spec	1 : Dif ≠0 (no carry) 2 : Dif = 0 (carry) 3 : Dif ≠0 (carry)
Subtract Logical	SLR	1F	RR	R1 R2	Algebraically subtract 2nd opr (in GPR per R2) from 1st opr (in GPR per R1) & place result into 1st opr location. Sign bit of result is treated as high-order integer & is tested for carry to determine CC.	None	1 : Dif ≠0 (no carry) 2 : Dif = 0 (carry) 3 : Dif ≠ (carry)

2. Addressing. An address designates a location outside the available main storage capacity. The instruction is terminated except for the Store, Store Halfword, and Convert to Decimal instructions, which are suppressed. Operand addresses are tested only when used to address storage. Addresses used as a shift amount are not tested. The address restrictions do not apply to the D2 field or to the contents of the GPR's addressed by the X2 and B2 fields.
3. Specification. A data, instruction, or control word address does not specify an integral boundary for the unit of information, or the R1 field of an instruction specifies an odd register address for a pair of GPR's that contain a doubleword operand. The operation is suppressed.
4. Data. A sign or digit code of the decimal operand in the Convert to Binary instruction is incorrect. The operation is terminated.
5. Fixed-point overflow. A high-order carry occurs, or high-order significant bits are lost in load, add, subtract, or shift operations. The instruction is completed by ignoring the overflow. The interruption may be masked off by making the fixed-point overflow mask bit [PSW(36)] a 0. If the mask bit is a 1, the interruption is taken.
6. Fixed-point divide. The quotient of a division, including division by zero, exceeds the register size, or the result of the Convert to Binary instruction exceeds 31 bits. If the interruption occurs during division, the operation is suppressed. If the interruption occurs during the Convert to Binary instruction, the conversion is completed but only the low-order 32 bits of the converted data are placed into LS.

Condition Codes

The results of fixed-point load, add, subtract, compare, and shift instructions set the CC in the PSW (Table 1-8). All other fixed-point instructions leave the CC undisturbed.

For fixed-point arithmetic operations, the CC can be set to reflect three types of results:

1. For most operations, codes 0, 1, and 2 indicate the result is zero, less than zero, or greater than zero, respectively, and code 3 indicates fixed-point overflow.
2. For compare operations, codes 0, 1, and 2 indicate that the first operand is equal to, lower than, or higher than the second operand, respectively.
3. For Add Logical and Subtract Logical instructions, codes 0 and 1 indicate a zero and non-zero result, respectively, in the absence of a logical carry out of the sign position; codes 2 and 3 indicate a zero and nonzero result, respectively, with a carry out of the sign position.

Floating-Point Instructions

The floating-point instructions serve to load, add, subtract, compare, halve, multiply, divide, and store floating-point numbers. These instructions may occur in the RR format for register-to-register transfers or in the RX format for register-to-storage transfers. Eight 32-bit FPR's in LS are reserved exclusively for floating-point instructions. They are logically connected by pairs to form four 64-bit FPR's. At the end of the execution of the floating-point add, subtract, compare, and certain load instructions, a CC is set.

Operands may be either short or long. Short operands are a word long (32 bits) and long operands are a doubleword long (64 bits). Long operands provide greater precision; however, where great precision is not necessary, short operands reduce instruction execution time and the amount of storage required.

Operands and final arithmetic results are always in true form (as opposed to complement form). A 0 in the sign position indicates a positive fraction; a 1, a negative fraction. If intermediate results are in complement form, they are changed to true form before the final result is stored into the first operand location. For the add, subtract, multiply, and divide instructions, the result signs are determined algebraically.

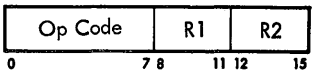
Table 1-9 lists the floating-point instructions. For a discussion of number representation, data formats, normalization, and operand addressing, refer to Section 2 of this chapter.

Instruction Formats

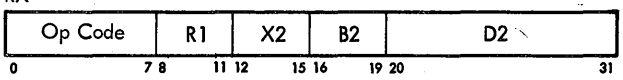
- Floating-point instructions use RR and RX formats.
- Programmer must specify even FPR of even/odd pair (0, 2, 4, or 6).
- Main storage address of second operand must designate word boundaries (bits 22 and 23 = 00) for short operands and doubleword boundaries (bits 21, 22, and 23 = 000) for long operands.

Floating-point instructions occur in the RR and RX formats:

RR



RX



In these formats, R1 is the address of an FPR that contains the first operand. The second operand location is defined differently for the two formats.

In the RR format, R2 is the address of an FPR containing the second operand. The same FPR may be specified for the first and second operands.

The R1 and R2 fields must specify 0, 2, 4, or 6, or a specification program interruption occurs. The specification check is made by testing E(8) and E(11) for zero; for RR instructions, E(12) and E(15) are tested for zero. If E(11) or E(15) does *not* equal zero, an odd address has been specified. If E(8) or E(12) does *not* equal zero, the specified FPR address is greater than 7. Thus, if any of the tested E bits equals 1, a specification program interruption is taken.

In the RX format, the contents of the GPR's specified by X2 and B2 are added to the contents of the D2 field to form an effective address designating the main storage location of the second operand. A zero in an X2 or B2 field indicates that no index or base component is to be used. The main storage address should designate word boundaries for short operands (bits 22 and 23 = 00) and doubleword boundaries (bits 21, 22, and 23 = 000) for long operands. Otherwise, a specification program interruption occurs.

The results replace the first operand except for store operations, where result replaces the second operand. Except for the storing of the final result, the contents of all LS registers and main storage locations participating in operand addressing or operation execution remain unchanged.

Data Flow

- Eight 32-bit LS registers are reserved for floating-point instructions.
- Micro-orders control low-order fraction fetch.
- LS FPR address specified must be even (0, 2, 4, or 6).
- Sign-handling is achieved via serial adder or STAT's.
- Characteristic-handling is performed via serial adder.
- Fraction-handling is performed via parallel adder.

Eight 32-bit LS registers (addresses 16–23) are reserved for floating-point instruction operands and results (Diagram 3-4, FEMDM). An even/odd pair of these registers functions as a double-length (64-bit) register with an assigned address of 0, 2, 4, or 6. A 0 in the R1 or R2 field of a floating-point instruction specifies LS locations 16 and 17; a 2 specifies locations 18 and 19; a 4 specifies locations 20 and 21; a 6 specifies locations 22 and 23.

In instructions other than floating-point, addressing is limited to 16 GPR's because the R1 and R2 fields contain four bits each. The LS address register (LAL), however, contains five bits; LAL(0) is used to address the FPR's. Because floating-point instructions must specify an LS address of 0, 2, 4, or 6 in the R1 and R2 (RR only) fields, and use only the FPR's for operands, a 1 is forced into LAL(0) when accessing LS during execution of a floating-point instruction. (Note that, for RX format instructions, the base and index register fields specify GPR's.) For example, if address 0 is specified by the R1 or R2 field, LS accesses LS register 16 (LAL = 10000). Short operand instructions fetch only 32 bits (single word) from the specified FPR. Because ingating and outgating of LS are limited to 32 bits each, long floating-point operands must be divided into two 32-bit words stored in an even/odd pair of FPR's. Under micro-order control, a 1 is forced into the low-order bit position of LAL [LAL(4)] to fetch or store the low-order 32 bits of a long operand from R1 plus 1 or R2 plus 1. For example, the 'RF*E3Ω1' micro-order specifies the FPR addressed by E(12–15) + 1. The R1 + 1 and R2 + 1 registers are the odd-numbered addresses of FPR's.

At the beginning of the execution phase of floating-point instructions, a specification test establishes that:

1. An even register is specified in the R1 and R2 (RR format only) fields.
2. A register address greater than 6 is not specified in the R1 and R2 (RR format only) fields.
3. The effective main storage address is on a doubleword boundary for long operands and on a word boundary for short operands.

Data flow may be divided into two paths: the fraction path and the sign and characteristic path. The fractions are transferred, added, or shifted via the parallel adder. The operands are located in DT, ST, and AB. For floating-point

Table 1-9. Floating-Point Instructions

Instruction	Mnemonic	Op Code	Format	Operands	Function	Program Interruptions	Condition Code
Add Normalized (long)	AD	6A	RX	R1 D2(X2, B2)	Algebraically add 2nd opr (in stg) to 1st opr (in FPR per R1 & R1 + 1) & place normalized result into 1st opr location. 1. Low-order fraction of 1st opr must be fetched from LS. 2. Set CC per result sign & magnitude.	Prot (F) Adr Spec Exp Ovflo Exp Unflo Signif	0 : Fract = 0 1 : Fract < 0 2 : Fract > 0
Add Normalized (long)	ADR	2A	RR	R1 R2	Algebraically add 2nd opr (in FPR per R2 & R2 + 1) to 1st opr (in FPR per R1 & R1 + 1) & place normalized result into 1st opr location. 1. Low-order fractions of 1st & 2nd opr's must be fetched from LS. 2. Set CC per result sign & magnitude.	Spec Exp Ovflo Exp Unflo Signif	0 : Fract = 0 1 : Fract < 0 2 : Fract > 0
Add Normalized (short)	AE	7A	RX	R1 D2(X2, B2)	Algebraically add 2nd opr (in stg) to 1st opr (in FPR per R1) & place normalized result into 1st opr location. 1. Low-order half of FPR is ignored & unchanged. 2. D(21) determines which half of doubleword from stg is 2nd opr; if 1, right half; if 0, left half. 3. Set CC per result sign & magnitude.	Prot (F) Adr Spec Exp Ovflo Exp Unflo Signif	0 : Fract = 0 1 : Fract < 0 2 : Fract > 0
Add Normalized (short)	AER	3A	RR	R1 R2	Algebraically add 2nd opr (in FPR per R2) to 1st opr (in FPR per R1) & place normalized result into 1st opr location. 1. Low-order halves of FPR's are ignored & unchanged. 2. Set CC per result sign & magnitude.	Spec Exp Ovflo Exp Unflo Signif	0 : Fract = 0 1 : Fract < 0 2 : Fract > 0
Add Unnormalized (long)	AW	6E	RX	R1 D2(X2, B2)	Algebraically add 2nd opr (in stg) to 1st opr (in FPR per R1 & R1 + 1) & place unnormalized result into 1st opr location. 1. Low-order fraction of 1st opr must be fetched from LS. 2. Set CC per result sign & magnitude.	Prot (F) Adr Spec Exp Ovflo Signif	0 : Fract = 0 1 : Fract < 0 2 : Fract > 0
Add Unnormalized (long)	AWR	2E	RR	R1 R2	Algebraically add 2nd opr (in FPR per R2 & R2 + 1) to 1st opr (in FPR per R1 & R1 + 1) & place unnormalized result into 1st opr location. 1. Low-order fractions of 1st & 2nd opr's must be fetched from LS. 2. Set CC per result sign & magnitude.	Spec Exp Ovflo Signif	0 : Fract = 0 1 : Fract < 0 2 : Fract > 0
Add Unnormalized (short)	AU	7E	RX	R1 D2(X2, B2)	Algebraically add 2nd opr (in stg) to 1st opr (in FPR per R1) & place unnormalized result into 1st opr location. 1. Low-order half of FPR is ignored & unchanged. 2. D(21) determines which half of doubleword from stg is 2nd opr: if 1, right half; if 0, left half. 3. Set CC per result sign & magnitude.	Prot (F) Adr Spec Exp Ovflo Signif	0 : Fract = 0 1 : Fract < 0 2 : Fract > 0
Add Unnormalized (short)	AUR	3E	RR	R1 R2	Algebraically add 2nd opr (in FPR per R2) to 1st opr (in FPR per R1) & place unnormalized result into 1st opr location. 1. Low-order halves of FPR's are ignored & unchanged. 2. Set CC per result sign & magnitude.	Spec Exp Ovflo Signif	0 : Fract = 0 1 : Fract < 0 2 : Fract > 0
Compare (long)	CD	69	RX	R1 D2(X2, B2)	Algebraically compare 1st opr (in FPR per R1 & R1 + 1) with 2nd opr (in stg); CC indicates result. 1. Low-order fraction of 1st opr must be fetched from LS. 2. Opr's remain unchanged.	Prot (F) Adr Spec	0 : Opr 1 = Opr 2 1 : Opr 1 < Opr 2 2 : Opr 1 > Opr 2

Table 1-9. Floating-Point Instructions (Cont)

Instruction	Mnemonic	Op Code	Format	Operands	Function	Program Interruptions	Condition Code
Compare (long)	CDR	29	RR	R1 R2	Algebraically compare 1st opr (in FPR per R1 & R1 + 1) with 2nd opr (in FPR per R2 & R2 + 1); CC indicates result. 1. Low-order fractions of 1st & 2nd opr's must be fetched from LS. 2. Opr's remain unchanged.	Spec	0 : Opr 1 = Opr 2 1 : Opr 1 < Opr 2 2 : Opr 1 > Opr 2
Compare (short)	CE	79	RX	R1 D2(X2, B2)	Algebraically compare 1st opr (in FPR per R1) with 2nd opr (in stg); CC indicates result. 1. Low-order half of FPR is ignored. 2. D(21) determines which half of doubleword from stg is 2nd opr: if 1, right half; if 0, left half. 3. Opr's remain unchanged.	Prot (F) Adr Spec	0 : Opr 1 = Opr 2 1 : Opr 1 < Opr 2 2 : Opr 1 > Opr 2
Compare (short)	CER	39	RR	R1 R2	Algebraically compare 1st opr (in FPR per R1) with 2nd opr (in FPR per R2); CC indicates result. 1. Low-order halves of FPR's are ignored. 2. Opr's remain unchanged.	Spec	0 : Opr 1 = Opr 2 1 : Opr 1 < Opr 2 2 : Opr 1 > Opr 2
Divide (long)	DD	6D	RX	R1 D2(X2, B2)	Divide 1st opr (in FPR per R1 & R1 + 1) by 2nd opr (in stg) & place normalized quotient into 1st opr location. 1. Low-order fraction of 1st opr must be fetched from LS. 2. Opr's are prenormalized. 3. Remainder is not saved.	Prot (F) Adr Spec Exp Ovflo Exp Unflo Flt-Pt Div	Unchanged
Divide (long)	DDR	2D	RR	R1 R2	Divide 1st opr (in FPR per R1 & R1 + 1) by 2nd opr (in FPR per R2 & R2 + 1) & place normalized quotient into 1st opr location. 1. Low-order fractions of 1st & 2nd opr's must be fetched from LS. 2. Opr's are prenormalized. 3. Remainder is not saved.	Spec Exp Ovflo Exp Unflo Flt-Pt Div	Unchanged
Divide (short)	DE	7D	RX	R1 D2(X2, B2)	Divide 1st opr (in FPR per R1) by 2nd opr (in stg) & place normalized quotient into 1st opr location. 1. Low-order half of FPR is ignored & unchanged. 2. D(21) determines which half of doubleword from stg is 2nd opr: if 1, right half; if 0, left half. 3. Opr's are prenormalized. 4. Remainder is not saved.	Prot (F) Adr Spec Exp Ovflo Exp Unflo Flt-Pt Div	Unchanged
Divide (short)	DER	3D	RR	R1 R2	Divide 1st opr (in FPR per R1) by 2nd opr (in FPR per R2) & place normalized quotient into 1st opr location. 1. Low-order halves of FPR's are ignored & unchanged. 2. Opr's are prenormalized. 3. Remainder is not saved.	Spec Exp Ovflo Exp Unflo Flt-Pt Div	Unchanged
Halve (long)	HDR	24	RR	R1 R2	Divide 2nd opr (in FPR per R2 & R2 + 1) by 2 & place normalized quotient into 1st opr location (in FPR per R1 & R1 + 1). Low-order fraction of 2nd opr must be fetched from LS.	Spec Exp Unflo	Unchanged
Halve (short)	HER	34	RR	R1 R2	Divide 2nd opr (in FPR per R2) by 2 & place normalized quotient into 1st opr location (in FPR per R1). Low-order halves of FPR's are ignored & unchanged.	Spec Exp Unflo	Unchanged
Load (long)	LD	68	RX	R1 D2(X2, B2)	Load 2nd opr (in stg) into 1st opr location (in FPR per R1 & R1 + 1).	Prot (F) Adr Spec	Unchanged

Table 1-9. Floating-Point Instructions (Cont)

Instruction	Mnemonic	Op Code	Format	Operands	Function	Program Interruptions	Condition Code
Load (long)	LDR	28	RR	R1 R2	Load 2nd opr (in FPR per R2 & R2 + 1) into 1st opr location (in FPR per R1 & R1 + 1). Low-order fraction of 2nd opr must be fetched from LS.	Spec	Unchanged
Load (short)	LE	78	RX	R1 D2(X2, B2)	Load 2nd opr (in stg) into 1st opr location (in FPR per R1). 1. D(21) determines which half of doubleword from stg is 2nd opr: if 1, right half; if 0, left half. 2. Low-order half of FPR is ignored & unchanged.	Prot (F) Adr Spec	Unchanged
Load (short)	LER	38	RR	R1 R2	Load 2nd opr (in FPR per R2) into 1st opr location (in FPR per R1). Low-order halves of FPR's are ignored & unchanged.	Spec	Unchanged
Load & Test (long)	LTDR	22	RR	R1 R2	Load 2nd opr (in FPR per R2 & R2 + 1) into 1st opr location (in FPR per R1 & R1 + 1). 1. Low-order fraction of 2nd opr must be fetched from LS. 2. Set CC according to sign & magnitude.	Spec	0 : 2nd opr fract = 0 1 : 2nd opr < 0 2 : 2nd opr > 0
Load & Test (short)	LTER	32	RR	R1 R2	Load 2nd opr (in FPR per R2) into 1st opr location (in FPR per R1). 1. Low-order halves of FPR's are ignored & unchanged. 2. Set CC according to sign & magnitude.	Spec	0 : 2nd opr fract = 0 1 : 2nd opr < 0 2 : 2nd opr > 0
Load Complement (long)	LCDR	23	RR	R1 R2	Load 2nd opr (in FPR per R2 & R2 + 1) into 1st opr location (in FPR per R1 & R1 + 1) with sign complemented. 1. Low-order fraction of 2nd opr must be fetched from LS. 2. Set CC according to original sign & magnitude.	Spec	0 : 2nd opr fract = 0 1 : Orig sign + 2 : Orig sign -
Load Complement (short)	LCER	33	RR	R1 R2	Load 2nd opr (in FPR per R2) into 1st opr location (in FPR per R1) with sign complemented. 1. Low-order halves of FPR's are ignored & unchanged. 2. Set CC according to original sign & magnitude.	Spec	0 : 2nd opr fract = 0 1 : Orig sign + 2 : Orig sign -
Load Negative (long)	LNDR	21	RR	R1 R2	Load 2nd opr (in FPR per R2 & R2 + 1) into 1st opr location (in FPR per R1 & R1 + 1) with sign made minus. 1. Low-order fraction of 2nd opr must be fetched from LS. 2. Set CC according to result sign & magnitude.	Spec	0 : 2nd opr fract = 0 1 : 2nd opr < 0
Load Negative (short)	LNER	31	RR	R1 R2	Load 2nd opr (in FPR per R2) into 1st opr location (in FPR per R1) with sign made minus. 1. Low-order halves of FPR's are ignored & unchanged. 2. Set CC according to result sign & magnitude.	Spec	0 : 2nd opr fract = 0 1 : 2nd opr < 0
Load Positive (long)	LPDR	20	RR	R1 R2	Load 2nd opr (in FPR per R2 & R2 + 1) into 1st opr location (in FPR per R1 & R1 + 1) with sign made plus. 1. Low-order fraction of 2nd opr must be fetched from LS. 2. Set CC according to result sign & magnitude.	Spec	0 : 2nd opr fract = 0 2 : 2nd opr > 0
Load Positive (short)	LPER	30	RR	R1 R2	Load 2nd opr (in FPR per R2) into 1st opr location (in FPR per R1) with sign made plus. 1. Low-order halves of FPR's are ignored & unchanged. 2. Set CC according to result sign & magnitude.	Spec	0 : 2nd opr fract = 0 2 : 2nd opr > 0

Table 1-9. Floating-Point Instructions (Cont)

Instruction	Mnemonic	Op Code	Format	Operands	Function	Program Interruptions	Condition Code
Multiply (long)	MD	6C	RX	R1 D2(X2, B2)	Multiply 1st opr (in FPR per R1 & R1 + 1) & 2nd opr (in stg) & place normalized product into 1st opr location (in FPR per R1 & R1 + 1). Opr's are prenormalized.	Prot (F) Adr Spec Exp Ovflo Exp Unflo	Unchanged
Multiply (long)	MDR	2C	RR	R1 R2	Multiply 1st opr (in FPR per R1 & R1 + 1) & 2nd opr (in FPR per R2 & R2 + 1) & place normalized product into 1st opr location (in FPR per R1 & R1 + 1). Opr's are prenormalized.	Spec Exp Ovflo Exp Unflo	Unchanged
Multiply (short)	ME	7C	RX	R1 D2(X2, B2)	Multiply 1st opr (in FPR per R1) & 2nd opr (in stg) & place normalized product into 1st opr location (in FPR per R1 & R1 + 1). 1. D(21) determines which half of doubleword from stg is 2nd opr: if 1, right half; if 0, left half. 2. Opr's are prenormalized.	Prot (F) Adr Spec Exp Ovflo Exp Unflo	Unchanged
Multiply (short)	MER	3C	RR	R1 R2	Multiply 1st opr (in FPR per R1) & 2nd opr (in FPR per R2) & place normalized product into 1st opr location (in FPR per R1 & R1 + 1). Opr's are prenormalized.	Spec Exp Ovflo Exp Unflo	Unchanged
Store (long)	STD	60	RX	R1 D2(X2, B2)	Store 1st opr (in FPR per R1 & R1 + 1) into 2nd opr location (in stg). 1st opr is unchanged.	Prot (S) Adr Spec	Unchanged
Store (short)	STE	70	RX	R1 D2(X2, B2)	Store 1st opr (in FPR per R1) into 2nd opr location (in stg). 1. PAL(61) determines into which half of doubleword in stg 1st opr is to be stored: if 1, right half; if 0, left half. 2. Low-order half of FPR is ignored. 3. 1st opr is unchanged.	Prot (S) Adr Spec	Unchanged
Subtract Normalized (long)	SD	6B	RX	R1 D2(X2, B2)	Algebraically subtract 2nd opr (in stg) from 1st opr (in FPR per R1 & R1 + 1) & place normalized result into 1st opr location. 1. Low-order fraction of 1st opr must be fetched from LS. 2. Set CC per result sign & magnitude.	Prot (F) Adr Spec Exp Ovflo Exp Unflo Signif	0 : Fract = 0 1 : Fract < 0 2 : Fract > 0
Subtract Normalized (long)	SDR	2B	RR	R1 R2	Algebraically subtract 2nd opr (in FPR per R2 & R2 + 1) from 1st opr (in FPR per R1 & R1 + 1) & place normalized result into 1st opr location. 1. Low-order fractions of 1st & 2nd opr's must be fetched from LS. 2. Set CC per result sign & magnitude.	Spec Exp Ovflo Exp Unflo Signif	0 : Fract = 0 1 : Fract < 0 2 : Fract > 0
Subtract Normalized (short)	SE	7B	RX	R1 D2(X2, B2)	Algebraically subtract 2nd opr (in stg) from 1st opr (in FPR per R1) & place normalized result into 1st opr location. 1. Low-order half of FPR is ignored & unchanged. 2. D(21) determines which half of doubleword from stg is 2nd opr: if 1, right half; if 0, left half. 3. Set CC per result sign & magnitude.	Prot (F) Adr Spec Exp Ovflo Exp Unflo Signif	0 : Fract = 0 1 : Fract < 0 2 : Fract > 0
Subtract Normalized (short)	SER	3B	RR	R1 R2	Algebraically subtract 2nd opr (in FPR per R2) from 1st opr (in FPR per R1) & place normalized result into 1st opr location. 1. Low-order halves of FPR's are ignored & unchanged. 2. Set CC per result sign & magnitude.	Spec Exp Ovflo Exp Unflo Signif	0 : Fract = 0 1 : Fract < 0 2 : Fract > 0

Table 1-9. Floating-Point Instructions (Cont)

Instruction	Mnemonic	Op Code	Format	Operands	Function	Program Interruptions	Condition Code
Subtract Unnormalized (long)	SW	6F	RX	R1 D2(X2, B2)	Algebraically subtract 2nd opr (in stg) from 1st opr (in FPR per R1 & R1 + 1) & place unnormalized result into 1st opr location. 1. Low-order fraction of 1st opr must be fetched from LS. 2. Set CC per result sign & magnitude.	Prot (F) Adr Spec Exp Ovflo Signif	0 : Fract = 0 1 : Fract < 0 2 : Fract > 0
Subtract Unnormalized (long)	SWR	2F	RR	R1 R2	Algebraically subtract 2nd opr (in FPR per R2 & R2 + 1) from 1st opr (in FPR per R1 & R1 + 1) & place unnormalized result into 1st opr location. 1. Low-order fractions of 1st & 2nd opr's must be fetched from LS. 2. Set CC per result sign & magnitude.	Spec Exp Ovflo Signif	0 : Fract = 0 1 : Fract < 0 2 : Fract > 0
Subtract Unnormalized (short)	SU	7F	RX	R1 D2(X2, B2)	Algebraically subtract 2nd opr (in stg) from 1st opr (in FPR per R1) & place unnormalized result into 1st opr location. 1. Low-order half of FPR is ignored & unchanged. 2. D(21) determines which half of doubleword from stg is 2nd opr: if 1, right half; if 0, left half. 3. Set CC per result sign & magnitude.	Prot (F) Adr Spec Exp Ovflo Signif	0 : Fract = 0 1 : Fract < 0 2 : Fract > 0
Subtract Unnormalized (short)	SUR	3F	RR	R1 R2	Algebraically subtract 2nd opr (in FPR per R2) from 1st opr (in FPR per R1) & place unnormalized result into 1st opr location. 1. Low-order halves of FPR's are ignored & unchanged. 2. Set CC per result sign & magnitude.	Spec Exp Ovflo Signif	0 : Fract = 0 1 : Fract < 0 2 : Fract > 0

instructions, the parallel adder shifts operands right or left four bit positions under micro-order control. For floating-point, the contents of PAL can be gated to T, D, A, and B.

The sign and characteristic path is from ST or AB to F, via the serial adder. The byte gated to the inputs of the serial adder depends upon the STC and the ABC values. For floating-point, the STC is normally set to 4 to specify the first byte of T, and the ABC is set to 0 to specify the first byte of A. The data from F is gated to the serial adder. From the serial adder, the data is transferred to ST per the STC. For floating-point operations, the serial adder adds 1 to, or subtracts 1 or 64 from, the characteristic at the inputs of the serial adder under micro-order control.

In floating-point instructions, the signs are saved in STAT's under micro-order control. When the 'SAVE SIGNS' micro-order is executed, bit 0 of the ST byte selected by the STC is gated in true or complement form, depending upon the instruction, to STAT C. Because the STC is set to 0 or 4 before issuing the 'SAVE SIGNS' micro-order, the contents of either S(0) or T(32), whichever contains the sign of the operand, is saved in STAT C via the serial adder. At the same time, the sign of the operand in AB is sent to STAT F. If the instruction is a multiply or divide and SAL(0) = 1 (indicating a carry resulted from the characteristic addition or subtraction), STAT D is set.

When the results of the execution of a floating-point instruction are to be stored, the STAT's are decoded, under micro-order control, to determine the sign of the result. If the sign is minus, the 'INSERT SIGN' micro-order forces a 1 to bit 0 of the FPR (on the LS bus in) addressed by R1 and inhibits gating of T(32) to LS. The result sign is minus under the following conditions:

1. Multiply or divide and signs (STAT's C and F) are unlike.
2. Load complement and STAT C equals 0.
3. Halve, load, or load and test, and STAT C equals 1.
4. Add, subtract, or compare and sign of the larger operand is minus.
5. Load negative.

Program Interruptions

Seven program interruptions can occur during execution of floating-point instructions. Of the seven, "exponent underflow" and "significance" can be masked off; the others are unconditionally taken. If the associated mask bit [PSW(38) and PSW(39), respectively] is a 0, the interruption is ignored; if a 1, it is taken.

The seven interruptions and their causes are:

1. Protection. The storage key does not match the protection key in the PSW for all RX instructions. When an

instruction causes a fetch-protection violation, instruction execution is terminated, the program execution is altered by a program interruption, and a protection program interruption is indicated in the old PSW. When an instruction causes a store-protection violation, the operation is suppressed.

2. Addressing. An address designates a location outside the available storage for the installation. The operation is terminated.
3. Specification. A short operand is not located on a word boundary, a long operand is not located on a double-word boundary, or an FPR address other than 0, 2, 4, or 6 is specified. The instruction is suppressed. The address restrictions do not apply to the components (contents of the D2 field and the contents of the LS registers specified by X2 and B2) from which an address is generated.
4. Exponent overflow. The result exponent (characteristic) of an addition, subtraction, multiplication, or division overflows, and the result fraction is not zero. The operation is completed by making the characteristic 128 smaller than the true result; the sign and fraction remain unchanged.
5. Exponent underflow. The result of an addition, subtraction, multiplication, or division underflows, and the result fraction is not zero. A program interruption occurs if the exponent-underflow mask [PSW(38)] is a 1. The operation is completed by replacing the result with a true zero, if the mask is off. If the mask is on, the characteristic is made 128 larger than the true result and the sign and fraction remain unchanged.
6. Significance. The result fraction of an addition or subtraction is zero. A program interruption occurs if the significance mask [PSW(39)] is a 1. The mask bit also affects the result of the operation. When the significance mask bit is a 0, the operation is completed by replacing the result with a true zero. When the significance mask bit is 1, the operation is completed without further change to the characteristic of the result. In either case, the CC is set to 0.
7. Floating-point divide. Division by a number with a zero fraction is attempted. The division is suppressed, but the CC and the data in storage remain unchanged.

Condition Codes

The results of floating-point add, subtract, compare, and certain load operations set the CC (Table 1-9). Multiplication, division, and storing leave the CC unchanged.

The CC can be set to reflect two types of results for floating-point arithmetic. For most operations, CC's of 0, 1, and 2 respectively indicate that the result register contains zero, less than zero, and more than zero. A zero result is indicated whenever the result fraction is zero, including a forced zero. A CC of 3 is never set by floating-point instructions.

For compare instructions, CC's of 0, 1, and 2 respectively indicate that the first operand is equal to, lower than, and higher than the second operand.

Decimal Instructions

The decimal instructions provide for addition, subtraction, comparison, multiplication, division, and format conversion of variable-field length (VFL) operands. The VFL data, which may range from 1 to 16 bytes in length, resides in main storage only. All decimal instructions are therefore in the SS format to provide for storage-to-storage operations. In general, most decimal instructions require fetching the operands from main storage, performing the operations specified by the instruction op code, and storing the results in main storage. A list of the decimal instructions is contained in Table 1-10.

For a discussion of number representation, data formats, and operand addressing, refer to Section 2 of this chapter.

Data Handling

- Decimal arithmetic is performed by either true add or complement add sequence, using excess-6 arithmetic.
- True add sequence adds 6 to each digit gated to A-side of serial adder.
- Complement add sequence gates 2's complement of each digit to A-side of serial adder.
- Inputs to B-side of serial adder are unchanged.
- Each digit which did not cause a carry at output of serial adder is reduced by 6 (decimal corrected).
- If no carry from high-order digit, result is in complement form and must be recomplemented.

Decimal arithmetic operations are performed in the serial adder on a byte basis. A true add or a complement add sequence is used depending upon the instruction and the operand signs. Because decimal digits are in BCD format, excess-6 arithmetic is used.

As stated previously, the decimal digits are represented by a binary code. Each digit consists of a four-bit field, bit combinations 0000–1001 corresponding to decimal digits 0–9. This system of decimal notation allows relatively simple binary techniques to be applied when operating with decimal data, and also facilitates direct reading of decimal results. However, two problems are encountered. One problem is that the four-bit field used to represent decimal

Table 1-10. Decimal Instructions

Instruction	Mnemonic	Op Code	Format	Operands	Function	Program Interruptions	Condition Code
Add Decimal	AP	FA	SS	D1(L1, B1) D2(L2, B2)	Algebraically add 2nd opr (in stg) to 1st opr (in stg) & place result into 1st opr location. 1. Opr's & result are in packed format. 2. Opr fields may overlap if low-order bytes coincide. 3. Right to left, byte by byte. 4. Shorter opr is extended with high-order 0's. 5. 1st opr field must be large enough to contain all 2nd opr significant digits.	Prot (S,F) Adr Data Dec Ovflo	0 : Sum = 0 1 : Sum < 0 2 : Sum > 0 3 : Overflow
Compare Decimal	CP	F9	SS	D1(L1, B1) D2(L2,B2)	Algebraically compare 1st opr (in stg) with 2nd opr (in stg) & set CC according to result. 1. Opr's are in packed format. 2. Shorter opr is extended with high-order 0's. 3. Opr fields may overlap if low-order bytes coincide. 4. Right to left, byte by byte. 5. Result is not stored & opr fields are unchanged.	Prot (F) Adr Data	0 : Opr 1 = Opr 2 1 : Opr 1 < Opr 2 2 : Opr 1 > Opr 2
Divide Decimal	DP	FD	SS	D1(L1, B1) D2(L2, B2)	Divide 1st opr (in stg) by 2nd opr (in stg) & place result into 1st opr location (quotient is leftmost in 1st opr location; remainder, rightmost). 1. Opr's are in packed format. 2. Dividend must contain at least 1 high-order 0. 3. Max dividend field = 16 bytes (31 digits & sign); L1 = 15. 4. Max divisor field = 8 bytes (15 digits & sign); L2 = 7. 5. Divisor field must be < dividend field (L2 < L1). 6. Max quotient field = 15 bytes. 7. Quotient field = dividend field minus remainder (divisor) field (L1 minus L2). 8. Remainder field = divisor field. 9. Opr fields may overlap if low-order bytes coincide. 10. Sign of quotient is determined algebraically, except 0 result is positive. 11. Sign of remainder is same as dividend sign.	Prot (S,F) Adr Spec Data Dec Div	Unchanged
Move with Offset	MVO	F1	SS	D1(L1, B1) D2(L2, B2)	Store 2nd opr (in stg) to left of and adjacent to low-order 4 bits of 1st opr (in stg). 1. Opr's are in packed or unpacked format. 2. If 2nd opr is shorter than 1st opr, fill 1st opr field with high-order 0's. 3. If 2nd opr is longer than 1st opr, ignore excess 2nd opr high-order digits. 4. Right to left, byte by byte.	Prot (S,F) Adr	Unchanged
Multiply Decimal	MP	FC	SS	D1(L1, B1) D2(L2, B2)	Multiply 1st opr (in stg) by 2nd opr (in stg) & place result into 1st opr location. 1. Opr's are in packed format. 2. Product must contain at least 1 high-order 0. 3. Max multiplicand field = 16 bytes (31 digits & sign); L1 = 15. 4. Max multiplier field = 8 bytes (15 digits & sign); L2 = 7. 5. Multiplier field must be < multiplicand field (L2 < L1); max value of L2 = 7. 6. Multiplicand field initially contains high-order 0-field equal in length to multiplier field. 7. Max product field = 16 bytes (31 digits & sign). 8. Sign of product is determined algebraically, except 0 result is positive.	Prot (S,F) Adr Spec Data	Unchanged

Table 1-10. Decimal Instructions (Cont)

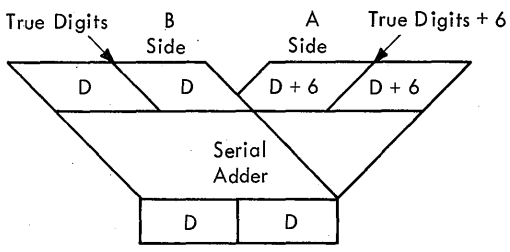
Instruction	Mnemonic	Op Code	Format	Operands	Function	Program Interruptions	Condition Code
Pack	PACK	F2	SS	D1(L1, B1) D2(L2, B2)	Convert format of 2nd opr (in stg) from zoned to packed & place result into 1st opr location (in stg). 1. 2nd opr is in zoned format. 2. No restriction on overlapping fields. 3. Extend 2nd opr with high-order 0's, if necessary. 4. If 1st opr field is too short to contain all significant digits of 2nd opr field, ignore excess 2nd opr high-order digits. 5. Right to left, byte by byte.	Prot (S,F) Adr	Unchanged
Subtract Decimal	SP	FB	SS	D1(L1, B1) D2(L2, B2)	Algebraically subtract 2nd opr (in stg) from 1st opr (in stg) & place result into 1st opr location. 1. Opr's & result are in packed format. 2. Opr fields may overlap if low-order bytes coincide. 3. 1st opr field must be large enough to contain all 2nd opr significant digits. 4. Shorter opr is extended with high-order 0's. 5. Right to left, byte by byte.	Prot (S,F) Adr Data Dec Ovflo	0 : Dif = 0 1 : Dif < 0 2 : Dif > 0 3 : Overflow
Unpack	UNPK	F3	SS	D1(L1, B1) D2(L2, B2)	Convert format of 2nd opr (in stg) from packed to zoned & place result into 1st opr location (in stg). 1. 2nd opr is in packed format. 2. No restriction on overlapping fields. 3. Extend 2nd opr with high-order 0's, if necessary. 4. If 1st opr field is too short to contain all significant digits of 2nd opr field, ignore excess 2nd opr high-order digits. 5. If PSW(12) = 1, use USASCII-8 code for zones; if PSW(12) = 0, use EBCDIC. 6. Right to left, byte by byte.	Prot (S,F) Adr	Unchanged
Zero & Add	ZAP	F8	SS	D1(L1, B1) D2(L2, B2)	Place 2nd opr (in stg) into 1st opr location (in stg). 1. 2nd opr is in packed format. 2. Opr fields may overlap if low-order byte of 1st opr coincides with or is to the right of low-order byte of 2nd opr. 3. 1st opr field must be large enough to contain all 2nd opr significant digits.	Prot (S,F) Adr Data Dec Ovflo	0 : Result = 0 1 : Result < 0 2 : Result > 0 3 : Overflow

digits has 16 possible codes, of which 6 (binary combinations for 10 through 15 inclusive) are invalid as decimal digits. Thus means must be provided to correct invalid results when they occur in an arithmetic operation. For example, the addition of decimal digits 0110 (six) and 0101 (five) must yield a decimal result of 0001 0001 (eleven). If, however, a pure binary addition is carried out, it will yield an unacceptable result:

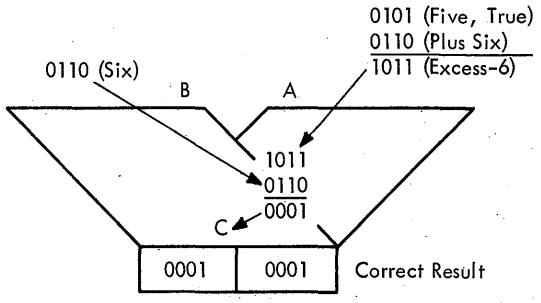
```
 0110   (decimal, or binary 6)
 0101   (decimal, or binary 5)
 1011   (invalid as decimal, but 11 in binary)
```

The second problem is in the generation of a decimal carry. When the sum of two decimal digits exceeds 9, a carry must be sent to the next high-order digit. However, a pure binary addition does not yield a carry unless the sum of the digits exceeds 1111 (15), which has the effect of a hex carry; i.e., carrying the order of 16 rather than 10.

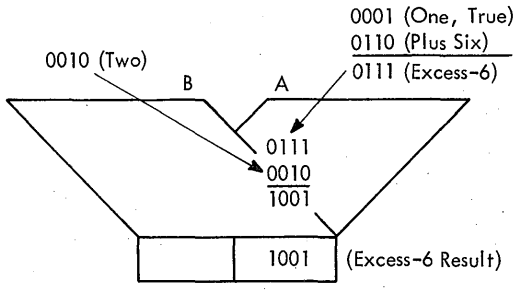
Both of the above problems are solved by the excess-6 arithmetic scheme and the decimal correction functions of the serial adder. In the excess-6 scheme, often referred to as true +6 arithmetic, a 6 is added to each digit as it is gated to the A-side of the adder, one byte (two digits) at a time from the second operand; the digits gated to the adder B-side, one byte at a time from the first operand, are not affected:



If the sum of the two digits to be added is 10 or greater, the true +6 scheme automatically eliminates the unwanted binary configuration and also supplies a decimal carry in terms of a hex carry. In true +6 arithmetic, the previous add example of digits 5 and 6 is executed as follows:

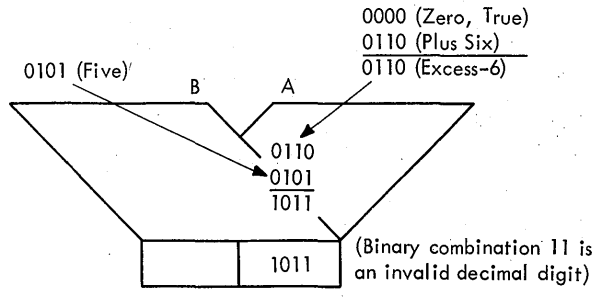


Addition of a 6 in all cases, however, may create an erroneous and sometimes invalid result. This occurs if the sum of the two digits to be added is less than 10. For example, consider the addition of decimal digits 1 and 2:



In the above case, the result (9) is clearly in excess-6 form; the digit 6 must be subtracted from the result to obtain the correct answer.

A further example illustrates how an excess-6 digit may generate an invalid result. Consider the addition of decimal digits 0 and 5:

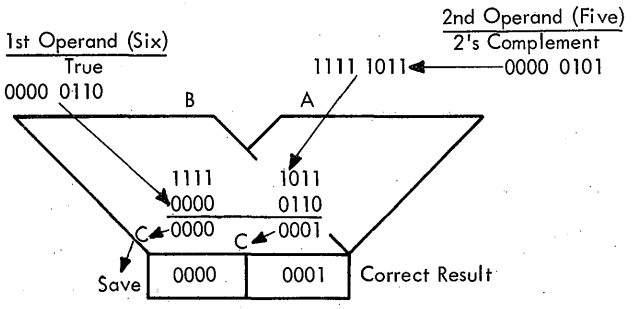


Note that both the erroneous and the invalid results are characterized by a no-carry to the next high-order digit. This condition holds true in all cases when incorrect data is generated, and is utilized by the decimal correction logic of the adder. When a no-carry condition is detected, this logic automatically deducts 6 from the result, thus supplying the correct digit to the adder output.

The decimal correct function of the adder is also used during complement add operations. The binary codes of the decimal digits at the adder A-side are gated in 2's complement form; excess 6's are ***not*** supplied. The digits at the B-side of the adder are gated in true form. The result of a complement add operation may be in true or complement form.

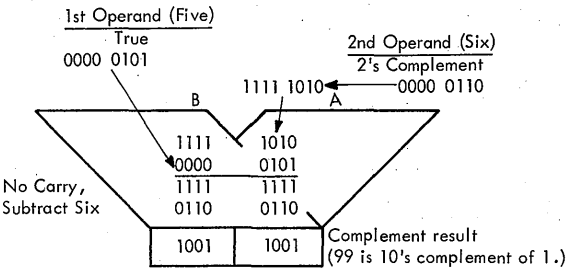
For clarity, the previous examples have shown operations that use only one digit. However, the serial adder normally handles one byte (two digits) at a time. To demonstrate the operation of the serial adder during decimal operations, the following examples deal with a byte of data.

If the first operand is larger than the second, the result is in true form. Consider complement addition of decimal digit 5 to 6; that is 6 minus 5:



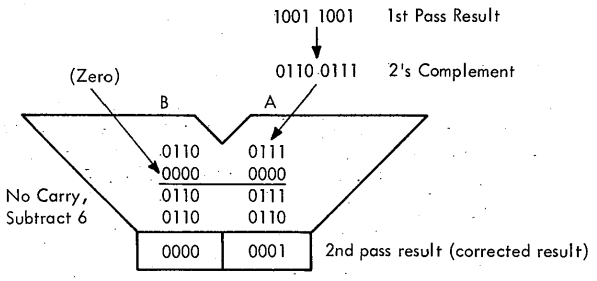
A true result during a complement add operation is always characterized by a carry from the last high-order digit. As in the case of the true add operation, a carry to the next digit indicates that no decimal correction of that digit is necessary.

If the first operand is smaller than the second, the result is in complement form. Because decimal data is always stored in true form, the result must be recomplemented. Consider complement addition of the decimal digit 6 to 5; that is, 5 minus 6:



Note that the decimal correction feature of the adder always subtracts 6 from each digit position which does not produce a carry. In a complement add operation, a no-carry condition from the last high-order digit also indicates that

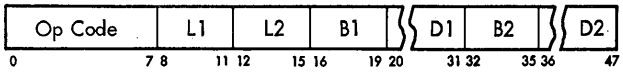
the result is in complement form and must be recomplemented. This requires a second pass through the adder:



Instruction Format

- Instructions specify two addresses.
- B1 (contents) + D1 + L1 specifies rightmost byte of 1st operand.
- B2 (contents) + D2 + L2 specifies rightmost byte of 2nd operand.
- Results are stored in true form at first operand location.

All decimal instructions use the SS format:



An SS instruction operates on two operands in main storage and stores the result into the same location from which the first operand was obtained. Therefore, the address of the first operand is also the destination address; the address of the second operand is commonly referred to as the *source address.*

The contents of the GPR specified by the B1 field are added to the D1 field to form an address. This address specifies the leftmost byte of the first operand. The number of operand bytes to the right of this byte is specified by the L1 field of the instruction. The L1 field may specify up to 16 bytes. Similarly, the address of the second operand is specified by the B2, D2, and L2 fields of the instruction. A zero in the B1 or B2 fields indicates the absence of the corresponding address component.

Normally, decimal operands are processed from right to left. Thus the address for the initial operand fetch is:

LS register per B-field + D-field + L-field.

Operands are fetched from main storage one doubleword, or eight bytes, at a time. Because the L-field may specify up to 16 bytes, several operand fetches may be required to completely access the operand. After each fetch, the operand address is decremented by 8 to access the next high-order eight bytes of the operand.

The results of decimal operations are placed into the first operand field and must be in true form. The result is never stored outside the first operand field specified by the instruction. If the first operand is longer than the second, the second operand is extended with high-order zeros up to the length of the first operand. Such extension does not modify the second operand in main storage, where it remains unchanged.

Data Flow

- All decimal instructions use serial adder.
- First operand is placed into ST; second operand into AB.
- STC specifies which ST byte is to be processed. ABC specifies which AB byte is to be processed.
- Destination bytes replace first operand bytes in ST.
- D contains first operand and destination address.
- IC contains second operand address.
- L1 and L2 specify number of first and second operand bytes, respectively, to be processed.

The data path used for decimal operations consists primarily of ST, AB, and the serial adder (Diagram 3-5, FEMDM). ST contains the first operand, and AB the second. The input byte to the adder A-side is selected from AB under control of the ABC. The input to the B-side of the adder is selected from ST under STC control. The selected bytes are gated to the adder simultaneously.

The serial adder handles the data at a rate of one byte per cycle; i.e., for each two input bytes, one output byte is generated at the SAL. The output byte is gated from SAL to ST under control of the STC, after which the ABC and the STC are decremented and a new cycle is started. Thus, as the operation progresses, the first operand bytes in ST are replaced by the destination bytes.

The number of first and second operand bytes processed depends upon length fields L1 and L2, respectively. The L1 count contained in E(8–11) is decremented once for each byte of first operand that is processed. Similarly, the L2 count in E(12–15) is decremented once for each second operand byte processed.

D contains the address of the first operand, which is also the address of the destination. The initial address in D specifies the doubleword containing the rightmost byte of the operand. When the STC is decremented to zero, indicating that all first operand bytes in ST have been processed, the contents of ST are stored into main storage. If additional first operand bytes remain in main storage (the L1 count has not stepped to zero), the D-address is decremented by 8, and a fetch of the next operand doubleword is made to ST.

Storage of the destination bytes in ST is controlled by the mark triggers. The mark triggers permit alteration of only those bytes in main storage that belong to the field being processed. There is one mark trigger for each of the eight bytes in ST. As a byte of processed data is gated to ST, the corresponding mark trigger is set, thus designating the byte for main storage.

The IC contains the address of the second operand. (The instruction address is held in the LSWR during execution of SS instructions.) The initial IC address specifies the doubleword containing the rightmost byte of the second operand. When the ABC is decremented to zero, all operand bytes in AB have been processed. If additional second operand bytes remain in main storage (L2 count has not stepped to zero), the IC address is decremented by 8 and a fetch is made of the next second operand doubleword to AB.

This pattern of fetching data, processing via the serial adder, assembling the results in ST, and storing the contents of ST into main storage is continued until either the first or the second operand length field (L1 or L2 count, respectively) has counted below zero. The operation at this point depends upon the individual instruction. If L2 has been exhausted but not L1, some instructions may require extension of the remaining first operand bytes with high-order zeros. On the other hand, if L1 is exhausted before L2, the instruction may test the remaining second operand bytes for presence of significant digits. This test is performed to detect a possible overflow condition and is accomplished by running the excess second operand bytes through the serial adder and sensing nonzeros. In all cases, if both L1 and L2 counts are exhausted, the instruction execution ends after the last destination word is stored into main storage.

Some decimal operations require use of the parallel adder to perform a right-4 or left-4 shift of the entire operand. The "spilled" bits generated during the shift are held in B(64–67).

Program Interruptions

Six program interruptions can occur during execution of decimal instructions. Of the six, only decimal overflow can be masked off; the others are unconditionally taken. If the decimal overflow mask bit [PSW(37)] is a 0, the decimal overflow interruption is ignored; if a 1, it is taken.

The six interruptions and their causes are:

1. Protection. The storage key does not match the protection key in the PSW. The operation is terminated for either a store or a fetch violation.
2. Addressing. An address designates a location outside the available storage for the installed system. The operation is terminated.
3. Specification. A multiplier or a divisor size exceeds 15 digits and sign, or a divisor is equal to or greater than the dividend, or a multiplier is equal to or greater than the multiplicand. The instruction is suppressed.
4. Data. A sign or digit code of an operand specified in the Add, Subtract, Compare, Zero and Add, Multiply, or Divide instruction is incorrect, a multiplicand has insufficient high-order zeros, or the operand fields in these instructions overlap. The operation is terminated before any original data is changed in main storage, except for an invalid digit code which is detected after the first store cycle.
5. Decimal overflow. Execution of the Add, Subtract, or Zero and Add instruction results in an overflow condition. The program interruption occurs only when the decimal-overflow mask [PSW(37)] is a 1. The operation is completed by placing the truncated low-order result into the result field and setting the CC to 3. The sign and low-order digits contained in the result field are the same as they would have been for an infinitely long result field.
6. Decimal divide. The quotient exceeds the specified data field, including division by zero. Division is suppressed. Therefore, the dividend and divisor remain unchanged in storage.

Condition Codes

The results of the Decimal Add, Subtract, Compare, and Zero and Add instructions set the CC as shown in Table 1-10.

Logical Instructions

The logical instructions provide for logical manipulation of data: moving, comparing, bit testing, bit connecting, translating, editing, and shifting. The logical instructions use all five instruction formats and work with both fixed- and variable-field length data. Table 1-11 lists the logical instructions.

For a discussion of the eight-bit zoned character codes, data formats, and operand addressing, refer to Section 2 of this chapter.

Instruction Formats

Logical instructions use the following five formats:

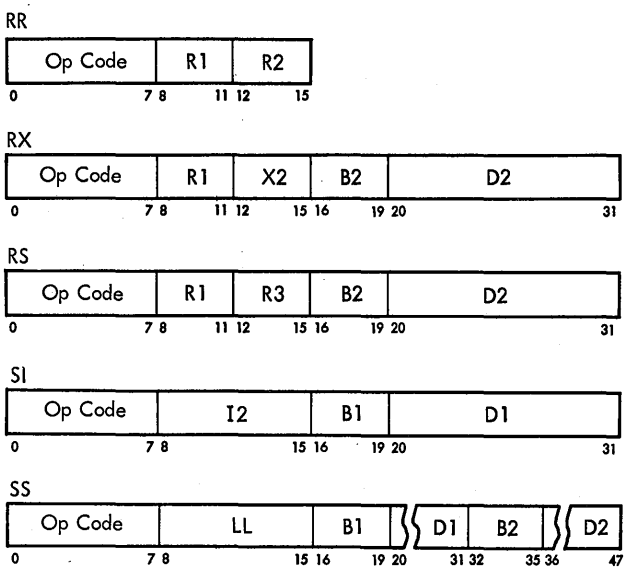


Table 1-11. Logical Instructions

Instruction	Mnemonic	Op Code	Format	Operands	Function	Program Interruptions	Condition Code
AND	N	54	RX	R1 D2(X2, B2)	AND 1st opr (in GPR per R1) with 2nd opr (in stg) & place result into 1st opr location. Left to right, byte by byte.	Prot (F) Adr Spec	0 : Result = 0 1 : Result ≠0
AND	NC	D4	SS	D1(L, B1) D2(B2)	AND 1st opr (in stg) with 2nd opr (in stg) & place result into 1st opr location. 1. Left to right, byte by byte. 2. Max number of bytes is 256.	Prot (S,F) Adr	0 : Result = 0 1 : Result ≠0
AND	NI	94	SI	D1(B1) I2	AND immediate opr (I2 of inst) with 1st opr (in stg) & place result into 1st opr location.	Prot (S) Adr	0 : Result = 0 1 : Result ≠0
AND	NR	14	RR	R1 R2	AND 1st opr (in GPR per R1) with 2nd opr (in GPR per R2) & place result into 1st opr location. Left to right, byte by byte.	None	0 : Result = 0 1 : Result ≠0
Compare Logical	CL	55	RX	R1 D2(X2, B2)	Binarily compare 1st opr (in GPR per R1) with 2nd opr (in stg) & set CC according to result. 1. Left to right, byte by byte. 2. Terminate on inequality or end of fields.	Prot (F) Adr Spec	0 : Opr 1 = Opr 2 1 : Opr 1 < Opr 2 2 : Opr 1 > Opr 2
Compare Logical	CLC	D5	SS	D1(L, B1) D2(B2)	Binarily compare 1st opr (in stg) with 2nd opr (in stg) & set CC according to result. 1. Left to right, byte by byte. 2. Max number of bytes is 256. 3. Terminate on inequality or end of fields.	Prot (F) Adr	0 : Opr 1 = Opr 2 1 : Opr 1 < Opr 2 2 : Opr 1 > Opr 2
Compare Logical	CLI	95	SI	D1(B1) I2	Binarily compare 1st opr (in stg) with immediate opr (I2 of inst) & set CC according to result. 1. Left to right. 2. Terminate on inequality or end of fields.	Prot (F) Adr	0 : Opr 1 = Opr 2 1 : Opr 1 < Opr 2 2 : Opr 1 > Opr 2
Compare Logical	CLR	15	RR	R1 R2	Binarily compare 1st opr (in GPR per R1) with 2nd opr (in GPR per R2) & set CC according to result. 1. Left to right, byte by byte. 2. Terminate on inequality or end of fields.	None	0 : Opr 1 = Opr 2 1 : Opr 1 < Opr 2 2 : Opr 1 > Opr 2
Edit	ED	DE	SS	D1(L, B1) D2(B2)	Change format of source (2nd opr; in stg) from packed to zoned, edit source under control of pattern (1st opr; in stg), & place result into 1st opr location. 1. Left to right, byte by byte. 2. Max number of bytes is 256.	Prot (S,F) Adr Data	0 : Result = 0 1 : Result < 0 2 : Result > 0
Edit & Mark	EDMK	DF	SS	D1(L, B1) D2(B2)	Change format of source (2nd opr; in stg) from packed to zoned, edit source under control of pattern (1st opr; in stg), place result into 1st opr location, & place location of each 1st significant result digit into GPR1. 1. Left to right, byte by byte. 2. Max number of bytes is 256.	Prot (S,F) Adr Data	0 : Result = 0 1 : Result < 0 2 : Result > 0
Exclusive OR	X	57	RX	R1 D2(X2, B2)	Exclusive-OR 1st opr (in GPR per R1) with 2nd opr (in stg) & place result into 1st opr location. Left to right, byte by byte.	Prot (F) Adr Spec	0 : Result = 0 1 : Result ≠0
Exclusive OR	XC	D7	SS	D1(L, B1) D2(B2)	Exclusive-OR 1st opr (in stg) with 2nd opr (in stg) & place result into 1st opr location. 1. Left to right, byte by byte. 2. Max number of bytes is 256.	Prot (S,F) Adr	0 : Result = 0 1 : Result ≠0
Exclusive OR	XI	97	SI	D1(B1) I2	Exclusive-OR immediate opr (I2 of inst) with 1st opr (in stg) & place result into 1st opr location.	Prot (S) Adr	0 : Result = 0 1 : Result ≠0
Exclusive OR	XR	17	RR	R1 R2	Exclusive-OR 1st opr (in GPR per R1) with 2nd opr (in GPR per R2) & place result into 1st opr location. Left to right, byte by byte.	None	0 : Result = 0 1 : Result ≠0

Table 1-11. Logical Instructions (Cont)

Instruction	Mnemonic	Op Code	Format	Operands	Function	Program Interruptions	Condition Code
Insert Character	IC	43	RX	R1 D2(X2, B2)	Insert 2nd opr (byte; in stg) into bits 24–31 of 1st opr location (in GPR per R1). Remaining bits in GPR are unchanged.	Prot (F) Adr	Unchanged
Load Address	LA	41	RX	R1 D2(X2, B2)	Insert 2nd opr adr into bits 8–31 of GPR specified by R1. 1. Bits 0–7 in GPR are made 0's. 2. 2nd opr is not fetched from stg.	None	Unchanged
Move	MVC	D2	SS	D1(L, B1) D2(B2)	Place 2nd opr (in stg) into 1st opr location (in stg). 1. Left to right, byte by byte. 2. Max number of bytes is 256. 3. Move operation can be high or low speed.	Prot (S,F) Adr	Unchanged
Move	MVI	92	SI	D1(B1) I2	Place immediate opr (I2 of inst) into 1st opr location (in stg).	Prot (S) Adr	Unchanged
Move Numerics	MVN	D1	SS	D1(L, B1) D2(B2)	Place numeric portion (low-order 4 bits) of each byte of 2nd opr (in stg) into low-order 4 bits of corresponding byte of 1st opr (in stg). 1. Left to right, byte by byte. 2. Max number of bytes is 256. 3. Zones (high-order 4 bits) in both opr's are unchanged. 4. No restriction on overlapping fields.	Prot (S,F) Adr	Unchanged
Move Zones	MVZ	D3	SS	D1(L, B1) D2(B2)	Place zone portion (high-order 4 bits) of each byte of 2nd opr (in stg) into high-order 4 bits of corresponding byte of 1st opr (in stg). 1. Left to right, byte by byte. 2. Max number of bytes is 256. 3. Numerics (low-order 4 bits) in both opr's are unchanged. 4. No restriction on overlapping fields.	Prot (S,F) Adr	Unchanged
OR	O	56	RX	R1 D2(X2, B2)	OR 1st opr (in GPR per R1) with 2nd opr (in stg) & place result into 1st opr location. Left to right, byte by byte.	Prot (F) Adr Spec	0 : Result = 0 1 : Result ≠ 0
OR	OC	D6	SS	D1(L, B1) D2(B2)	OR 1st opr (in stg) with 2nd opr (in stg) & place result into 1st opr location. 1. Left to right, byte by byte. 2. Max number of bytes is 256.	Prot (S,F) Adr	0 : Result = 0 1 : Result ≠ 0
OR	OI	96	SI	D1(B1) I2	OR immediate opr (I2 of inst) with 1st opr (in stg) & place result into 1st opr location.	Prot (S) Adr	0 : Result = 0 1 : Result ≠ 0
OR	OR	16	RR	R1 R2	OR 1st opr (in GPR per R1) with 2nd opr (in GPR per R2) & place result into 1st opr location. Left to right, byte by byte.	None	0 : Result = 0 1 : Result ≠ 0
Shift Left Double Logical	SLDL	8D	RS	R1 D2(B2)	Shift 1st opr (in GPR per R1 & R1 + 1) left number of bit positions specified by low-order 6 bits of 2nd opr adr. 1. R1 must be even adr. 2. High-order bits of 1st opr are shifted out & lost; vacated low-order bits are made 0's.	Spec	Unchanged
Shift Left Single Logical	SLL	89	RS	R1 D2(B2)	Shift 1st opr (in GPR per R1) left number of bit positions specified by low-order 6 bits of 2nd opr adr. High-order bits of 1st opr are shifted out & lost; vacated low-order bits are made 0's.	None	Unchanged
Shift Right Double Logical	SRDL	8C	RS	R1 D2(B2)	Shift 1st opr (in GPR per R1 & R1 + 1) right number of bit positions specified by low-order 6 bits of 2nd opr adr. 1. R1 must be even adr. 2. Low-order bits of 1st opr are shifted out & lost; vacated high-order bits are made 0's.	Spec	Unchanged

Table 1-11. Logical Instructions (Cont)

Instruction	Mnemonic	Op Code	Format	Operands	Function	Program Interruptions	Condition Code
Shift Right Single Logical	SRL	88	RS	R1 D2(B2)	Shift 1st opr (in GPR per R1) right number of bit positions specified by low-order 6 bits of 2nd opr adr. Low-order bits of 1st opr are shifted out & lost; vacated high-order bits are made 0's.	None	Unchanged
Store Character	STC	42	RX	R1 D2(X2, B2)	Store bits 24–31 of 1st opr (in GPR per R1) into 2nd opr location (in stg).	Prot (S) Adr	Unchanged
Test Under Mask	TM	91	SI	D1(B1) I2	Set CC according to state of 1st opr bits (in stg) selected by mask bits (12 of inst). 1. If mask bit = 1, test corresponding 1st opr bit; if mask bit = 0, ignore corresponding 1st opr bit. 2. Character in stg is unchanged.	Prot (F) Adr	0 : Selected bits all 0's (mask is all 0's) 1 : Selected bits mixed 0's & 1's 3 : Selected bits all 1's
Translate	TR	DC	SS	D1(L, B1) D2(B2)	Add 1st opr byte (argument; in stg) to effective 2nd opr adr, use result as stg adr, & place function byte from resulting stg adr into corresponding 1st opr byte location. 1. Effective 2nd opr adr = contents of GPR adr by B2, + D2. 2. LL = number of bytes to be translated. 3. 1st opr bytes are processed left to right.	Prot (S,F) Adr	Unchanged
Translate & Test	TRT	DD	SS	D1(L, B1) D2(B2)	Add 1st opr byte (argument; in stg) to effective 2nd opr adr, use result as stg adr, & test function byte from resulting stg adr. If 0, translate & test next argument byte; if non-0, complete operation by inserting related argument adr into GPR1 & function byte into GPR2. 1. Effective 2nd opr adr = contents of GPR adr by B2, + D2. 2. LL = number of bytes to be translated. 3. 1st opr bytes are processed left to right. 4. Set CC according to ending condition.	Prot (F) Adr	0 : All bytes tested are all 0's 1 : Non-0 byte found before last byte to be tested 2 : Non-0 byte found as last byte to be tested

In the RR, RX, and RS formats, the contents of the GPR specified by R1 are called the first operand. In the SI and SS formats, the contents of the GPR specified by B1 are added to the contents of the D1 field to form an address. This address designates the leftmost byte of the first operand field. The number of bytes to the right of this first byte is specified by the LL field in the SS instruction. In the SI format, the operand size is one byte.

In the RR format, the R2 field specifies the GPR containing the second operand. The same GPR may be specified for the first and second operands.

In the RX format, the contents of the GPR's specified by the X2 and B2 fields are added to the contents of the D2 field to form the address of the second operand in main storage.

In the RS format, used for shift operations, the contents of the GPR specified by the B2 field are added to the contents of the D2 field. This sum is not used as an address but the low-order six bits specify the number of bits of the shift. The R3 field is ignored in the shift operations.

In the SI format, the second operand is the eight-bit immediate data field, I2, of the instruction.

In the SS format, the contents of the GPR specified by B2 are added to the contents of the D2 field to form the address of the second operand. The second operand field has the same length as the first operand field.

A 0 in the X2, B1, or B2 field indicates the absence of the corresponding address or shift-amount components. An instruction can specify the same GPR both for address modification and for operand location. Address modification is always completed before operation execution.

Data Flow

Data paths used by the logical instructions are identical to those used by the decimal instructions, with one exception (Diagram 3-5, FEMDM). For decimal instructions, E(8–15) is divided into L1 and L2 fields. For logical instructions, E(8–15) is one field (LL).

The logical instructions operate on data which may range from 1 to 256 bytes in length. The operands are obtained either from the main storage or from a GPR. Sometimes, the operand may be contained in the instruction itself.

Processing of data in main storage proceeds from the high-order to the low-order address, or from left to right. The initial byte selected for processing may be at either an odd or even main storage address. As a rule, processing of

data in a GPR involves the complete register contents. Except for the editing instructions, data is not treated as numbers.

Generally, the operands are treated as eight-bit bytes. In a few cases, the left or right four bits of a byte are treated separately or operands are shifted a bit at a time.

Results replace the first operand, except in the Store Character instruction, where the result replaces the second operand. A variable-length result is never stored outside the field specified by the address and length.

The contents of all GPR's and storage locations participating in the addressing or execution of an operation generally remain unchanged. Exceptions are the move instructions, and the result locations, GPR1 in the Edit and Mark instruction, and GPR's 1 and 2 in the Translate and Test instruction.

Editing operations provide transformation from packed decimal digits to alphanumeric characters; i.e., editing requires a packed decimal field and generates zoned decimal digits. The digits, signs, and zones are recognized and generated as for decimal arithmetic; all bit configurations are considered valid.

The translating operations use a list of arbitrary values. A list provides a relation between an argument (the quantity used to reference the list) and the function (the contents of the location related to the argument). The purpose of the translation may be to convert data from one code to another code or to perform a control function. The list is specified by an initial address, the address designating the leftmost byte location of the list. The byte from the operand to be translated is the argument. The address used to address the list is obtained by adding the argument to the low-order positions of the initial address. As a consequence, the list contains 256 eight-bit function bytes. Where it is known that not all eight-bit argument values will occur, it may be possible to reduce the size of the list.

Use of GPR1 is implied in Edit and Mark and in Translate and Test instructions. A 24-bit address may be placed into this register during these operations. The Translate and Test instruction also implies GPR2. The low-order eight bits of GPR2 may be replaced by a function byte during a translate-and-test operation.

Program Interruptions

Four program interruptions can occur during execution of logical instructions:

1. Protection. The storage key of a result location in main storage does not match the protection key in the PSW. The operation is suppressed on a store violation. The only exceptions are the variable length storage-to-storage operations, which are terminated. The operation is terminated on a fetch violation.
2. Addressing. An address designates a location outside the available storage for the installed system. In most cases, the operation is terminated. The exceptions are the AND, Exclusive-OR, OR, and Move instructions that have the SI format, and the Store Character instruction. These instructions are suppressed. Operand addresses are tested only when used to address storage. Addresses used as a shift amount are not tested. Similarly, the address generated by the use of the Load Address instruction is not tested. The address restrictions do not apply to the contents of the D1 and D2 fields, or to the contents of the GPR's specified by X2, B1, and B2.
3. Specification. A full-word operand in a storage-to-register operation is not located on a 32-bit boundary, or an odd register address is specified for a pair of GPR's containing a 64-bit operand. The operation is suppressed.
4. Data. A digit code of the second operand in the Edit or Edit and Mark instruction is invalid. The operation is terminated.

Condition Codes

The results of most logical operations set the CC in the PSW (Table 1-11). The Load Address, Insert Character, Store Character, Translate, and the moving and shift instructions leave this code unchanged.

The CC can be set to reflect five types of results for logical operations. For the Compare Logical instructions, the 0, 1, and 2 states indicate that the first operand is equal to, less than, or greater than the second operand, respectively.

For the logical AND, OR, and Exclusive-OR instructions, the states 0 and 1 indicate a zero or nonzero result field, respectively.

For the Test under Mask instruction, the states 0, 1, and 3 indicate that the selected bits are all-zero, mixed zero and 1, or all-1, respectively.

For the Translate and Test instruction, the states 0, 1, and 2 indicate an all-zero function byte, a nonzero function byte with the operand incompletely tested, or a last function byte nonzero, respectively.

For editing, the states 0, 1, and 2 indicate a zero, less-than zero, or greater-than-zero content of the last result field, respectively.

Branching Instructions

- Branching causes departure from normal instruction sequencing.
- Branch address is introduced as next sequential address.
- Branch address is obtained from GPR or specified as 2nd operand address.
- Branch may be conditional or unconditional.

- Conditional branches (may or may not use branch address):
  Branch on condition
  Branch on count
  Branch on index
- Unconditional branches (always use branch address):
  Branch and link
  Execute
- On branch, normal storage request per IC to fill Q is blocked; branch logic will make request for IC if the branch is unsuccessful and Q needs to be refilled.
- If branch is unsuccessful, Q is refilled if required.

Normally, the CPU is controlled by instructions taken in sequential order. That is, an instruction is fetched from a main storage location specified by the instruction address in the IC. The address is then increased by the number of bytes needed to address the next instruction in sequence, and this updated address replaces the old address in the IC. The current instruction is executed, and the same steps are repeated using the updated instruction address to fetch the next instruction.

A departure from the normal instruction sequence occurs when branching is performed. A branch address is introduced as the next instruction address. This branch address may be obtained from one of the GPR's or it may be the second operand address specified by a particular instruction. Depending upon the format and the instruction, branching may be either conditional or unconditional. The conditional branches are branch on condition, branch on count, and branch on index. The unconditional branches are branch and link and execute. Conditional branches may or may not use the branch address. If the branch is successful (that is, the branch is taken), the branch address is used and the storage request issued per the IC during I-Fetch is blocked. If the branch is unsuccessful, the instruction address in the IC is used to fill Q. Unconditional branches are always taken and use the branch address.

Whether a conditional branch is successful depends upon the result of operations concurrent with the branch or preceding the branch. The first case is represented by the branch on count and branch on index instructions. The second case is represented by the branch on condition instructions, which inspect the CC that reflects the result of a previous arithmetic, logical, or I/O operation.

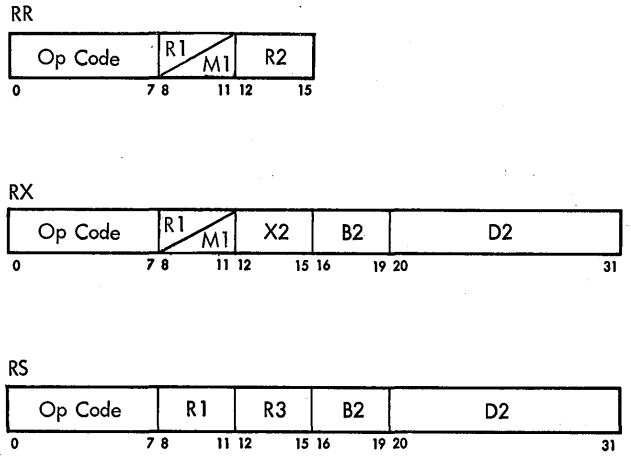
Branching is used to reference a subroutine, to resolve a two-way choice, or to repeat a portion of a program. To save time and increase the speed of the operating program, branching is always considered to be successful unless proven otherwise. (The branch conditions for branch on condition instructions is tested during I-Fetch for a successful or unsuccessful branch, and a D or IC request is issued dependent upon this test.) Therefore, whenever a branch instruction is decoded during I-Fetch, the next instruction address is the branch address located in D. If the branch is found to be unsuccessful (determined during execution of the branch- instruction), the instruction address from D is ignored, and the correct instruction address is obtained from the IC.

There are two methods of performing an end-op cycle in the branch operations: normal end op and branch end op. The normal end-op cycle allows decoding of the next instruction format from R and of the instruction address from the IC, and is normally used when ending an operation. Decoding off R is possible because the data placed into the register has become stable by the time the end-op cycle begins. The branch end-op cycle, on the other hand, allows decoding of the next instruction format from the SDBO and of the instruction address from D. This end-op cycle is used when the data, which has been placed into R, is not yet stable and is some halfword other than the last halfword of Q. Decoding from the SDBO saves the time it takes for the data to stabilize in R and the instruction address to stabilize in the IC.

Table 1-12 lists the branching instructions.

Instruction Formats

Branching instructions use the RR, RX, and RS formats:



In the formats shown above, bits 8–11 are normally the R1 field that specifies the address of a GPR containing the first operand. In the branch on condition instruction, however, bits 8–11 are designated as M1 and contain mask bits used in conjunction with the PSW CC to determine whether the branch is successful.

In the RR format, the R2 field specifies the address of a GPR that contains the branch address, except when R2 = 0, in which case no branching is to take place.

In the RX format, the contents of the GPR's specified by the X2 and B2 fields are added to the D2 field to form the branch address.

Table 1-12. Branching Instructions

Instruction	Mnemonic	Op Code	Format	Operands	Function	Program Interruptions	Condition Code
Branch & Link	BAL	45	RX	R1 D2(X2, B2)	Store PSW(32–63), link information, into GPR (adr by R1) & branch to location specified by 2nd opr adr. 1. Branch is unconditional. 2. Link information is stored whether or not branch is successful.	Prot (F)†	Unchanged
Branch & Link	BALR	05	RR	R1 R2	Store PSW(32–63), link information, into GPR (adr by R1) & branch to location specified by GPR (adr by R2). 1. Branch is unsuccessful if R2 = 0; use next sequential instr adr. 2. Link information is stored whether or not branch is successful.	Prot (F)†	Unchanged
Branch on Condition	BC	47	RX	M1 D2(X2, B2)	Branch to location specified by 2nd opr adr if state of CC is as specified by M1. 1. Branch is unconditional if M1 is all 1's. 2. Branch is unsuccessful if M1 is all 0's; use next sequential instr adr.	Prot (F)†	Unchanged
Branch on Condition	BCR	07	RR	M1 R2	Branch to location specified by GPR (adr by R2) if state of CC is as specified by M1. 1. Branch is unconditional if M1 is all 1's and R2 $\neq$ 0. 2. Branch is unsuccessful if R2 = 0 or if M1 is all 0's; use next sequential instr adr.	Prot (F)†	Unchanged
Branch on Count	BCT	46	RX	R1 D2(X2, B2)	Subtract 1 from 1st opr (in GPR per R1); if result $\neq$ 0, branch to location specified by 2nd opr adr. 1. Place result of subtraction into 1st opr location. 2. Branch is unsuccessful if result = 0; use next sequential instr adr. 3. If 1st opr = 1, no branching occurs.	Prot (F)†	Unchanged
Branch on Count	BCTR	06	RR	R1 R2	Subtract 1 from 1st opr (in GPR per R1); if result $\neq$ 0, branch to location specified by GPR (adr by R2). 1. Place result of subtraction into 1st opr location. 2. Branch is unsuccessful if result = 0 or if R2 = 0; use next sequential instr adr. 3. If 1st opr = 1, no branching occurs.	Prot (F)†	Unchanged
Branch on Index High	BXH	86	RS	R1 R3 D2(B2)	Add increment (3rd opr; in GPR per R3) to 1st opr (in GPR per R1), algebraically compare result (index) with comparand (in odd-adr GPR specified by R3 or R3 + 1); if index $>$ comparand, branch to location specified by 2nd opr adr. 1. Place index into 1st opr location. 2. Branch is unsuccessful if index = or $<$ comparand; use next sequential instr adr.	Prot (F)†	Unchanged
Branch on Index Low or Equal	BXLE	87	RS	R1 R3 D2(B2)	Add increment (3rd opr; in GPR per R3) to 1st opr (in GPR per R1), algebraically compare result (index) with comparand (in odd-adr GPR specified by R3 or R3 + 1); if index = or $<$ comparand, branch to location specified by 2nd opr adr. 1. Place index into 1st opr location. 2. Branch is unsuccessful if index $>$ comparand; use next sequential instr adr.	Prot (F)†	Unchanged
Execute	EX	44	RX	R1 D2(X2, B2)	Execute subject instr at location specified by 2nd opr adr. Subject instr may be modified by 1st opr (in GPR per R1) if E(8–11) $\neq$ 0. Modification is achieved by OR'ing bits 8–15 of subject instr with bits 24–31 of 1st opr; if R1 = 0, no modification takes place.	Execute Prot (F) Adr Spec	Set by subject instr

† Fetch protected: bit 4 of storage protect set.

In the RS format, which is used in branch on index operations, the contents of the GPR specified by the B2 field are added to the D2 field to form the branch address. The R3 field specifies the address in LS of an increment value (third operand) which is added to the first operand to determine the index value. R3, if odd, also is the comparand; if R3 is even, R3 + 1 is the comparand.

Data Flow

Diagram 3-6, FEMDM, is a diagram of the basic data flow for the branching instructions. The main functional units used to determine addresses and instructions in the branching operations are Q, R, E, D, and the IC. The secondary functional units, T, AB, parallel adder, STC, and ABC, determine whether the branch is successful when the branch being executed is a conditional branch. The purpose of each functional unit is as follows:

1. Q. Holds the doubleword that contains the instruction addressed by the branch instruction if the branch is successful.
2. R. Contains the instruction to be performed after execution of the branch instruction.
3. E. Contains the branch instruction presently being executed.
4. D. Holds the address of the doubleword which, if the branch is successful, contains the next instruction to be executed.
5. IC. Holds the address of the doubleword which, if the branch is unsuccessful, contains the next instruction to be executed.
6. T. Buffers the operand being tested and operated on.
7. AB. Holds the first operand when added to some other value to determine whether the branch is successful.
8. Parallel adder. Determines whether conditions have been met when a conditional branch is being executed.
9. STC. Allows transfer of last byte of T during an Execute instruction when modifying the subject instruction of the Execute instruction.
10. ABC. Selects data being modified in the subject instruction during an Execute instruction.

Program Interruptions

Four program interruptions can occur during execution of branching instructions:

1. Execute. The subject instruction of an Execute instruction is another Execute instruction. The operation is suppressed.
2. Protection. The branch address of an Execute instruction is protected. The branch-to address of any branch instruction may be fetch-protected. In this case, the PSW key must match the storage key or must be a master key of 0. The operation is suppressed.
3. Addressing. The branch address of an Execute instruction designates an instruction-halfword location outside the available storage area. The operation is suppressed.
4. Specification. The branch address of an Execute instruction is odd. The operation is suppressed.

Condition Codes

The branching instructions leave the CC unchanged, except for the Execute instruction. If the CC is set during the Execute instruction, it is set by the subject instruction.

Status Switching Instructions

- Load PSW, Set Program Mask, Set System Mask, and Supervisor Call instructions control status of CPU.
- Set Storage Key, Insert Storage Key, and Test and Set instructions control status of data in main storage.
- Write Direct and Read Direct instructions control status of external device (also transfer data bytes).
- Diagnose instruction controls status of CPU and channels.

The status switching instructions can change the status of the CPU, the channels, the external device, and the data in main storage. The status of a unit may also be changed by manual intervention and by interruptions (described elsewhere in this manual). The overall status of the CPU is determined by the current PSW and associated logic. (For a discussion of the PSW and of the eight CPU program states, refer to Section 3 of this chapter.) Any field in the current PSW may be changed directly by the Load PSW instruction, if the CPU is in the Supervisor state. Thus, the Load PSW instruction may be used to switch from the Supervisor state to the Problem state, between the Wait and Running states, and between the Masked and Interruptable states. At any time, the Set Program Mask instruction may be used to switch any of the four program mask bits between the Masked and Interruptable states. When in the Supervisor state, the Set System Mask instruction may be used to switch any of the eight system mask bits between the Masked and Interruptable states. The Supervisor Call instruction allows a problem program to switch the CPU from the Problem state to the Supervisor state; simultaneously, a byte of information is passed to the supervisor program via the interrupt code of the Supervisor Call old PSW.

Three instructions control the protection status of data in main storage. The Set Storage Key and Insert Storage Key instructions are privileged instructions for controlling the protection status of main storage data in 2048-byte blocks. The Set Storage Key instruction changes the storage protection keys in main storage. The Insert Storage Key instruction fetches the keys from main storage for inspection by the program. The Test and Set instruction, on the other hand, may be used in either the Supervisor or Problem state for protecting data in main storage in blocks of any length; this application is described in IBM Systems Reference Library, *IBM System/360 Principles of Operation,* Form A22-6821.

The Write Direct and Read Direct instructions, which are part of the Direct Control feature, may be used to switch the status of an external device by means of a code in the I2 field. If the external device is another System/360, the Write Direct and Read Direct instructions can be used to externally interrupt the receiving CPU. For example, assume CPU 1 and CPU 2 are both operating and have their Direct Control Features enabled (Diagram 5-607, Sheet 2, FEMDM). CPU 1 requests data from CPU 2 by executing a Read Direct instruction, sending an external interruption code to CPU 2 on the 'timing signal bus out' lines. If CPU 2 is masked on for external interruptions, if no other interruptions or exceptional conditions of higher priority are pending, and as soon as the instruction being processed is finished, CPU 2 is externally interrupted. After CPU 2's program decodes the interruption code and determines the operation to be performed, it executes a Write Direct instruction, sending its external interruption code and a 'direct control write out' signal to CPU 1 and putting the requested data on the 'direct control bus out' lines. CPU 1 can now be similarly interrupted and again execute a Read Direct instruction (but this time with its I2 field clear). CPU 1 waits until the 'direct control write out' signal (received as a 'direct control hold in' signal) is removed and then transfers the requested data from the 'direct control bus in' lines to main storage.

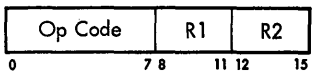
The Diagnose instruction controls the status of the CPU and the channels. Unlike the Load PSW, Set Program Mask, Set System Mask, and Supervisor Call instructions that switch the CPU's status by changing the current PSW, the Diagnose instruction switches the CPU's status by setting control triggers (such as 'defeat interleave', 'emulation mode', and 'diagnose FLT') through the use of a maintenance control word. The Diagnose instruction may also be used to switch the channels between a normal operating mode and a test mode.

Table 1-13 lists the status switching instructions.

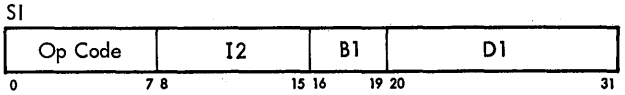
Instruction Formats

Status switching instructions have two formats:

RR



SI



In the RR format, the R1 and R2 fields specify GPR's except when used in the Supervisor Call instruction. The R1 and R2 fields in the Supervisor Call instruction are replaced by an I-field which contains an eight-bit interruption code. In the Set Program Mask instruction, the R2 field is ignored.

In the SI format, the I2 field is ignored for the Load PSW, Set System Mask, and Test and Set instructions. In the Write Direct and Read Direct instructions, the I2 field contains a timing signal code that is sent to an external device. In the Diagnose instruction, the I2 field contains a code for controlling certain maintenance aids and an optional Compatibility feature. The contents of the GPR specified by the B1 field are added to D1 to form a main storage address of an operand to be fetched by the instruction specified, except for Read Direct. The Read Direct instruction uses the address derived for storing data from an external device. Only one storage address is required in status switching operations. A 0 in the B1 field indicates the absence of the base address component.

Data Flow

- Each status switching instruction has different data flow.
- ST is used by most instructions as buffer before final data transfer.

The status switching instructions transfer data from one unit to another; except for the Insert Storage Key instruction, there is no intermediate processing. Depending on the instruction, the data is obtained from LS, main storage, or the 'direct control bus in' lines and is transferred to either LS, main storage, CPU control triggers, or the 'direct control bus out' lines. A generalized data flow is shown in Diagram 3-7, FEMDM. The following is a list of the functional units and their purposes.

1. BCU. Primarily used for 3-cycle fetches of storage operands per D. During the Load PSW instruction, a 3-cycle fetch per the IC is made for the next instruction after the new PSW has been loaded into the CPU. For the Set Storage Key, Insert Storage Key, and Test and Set instructions, the BCU performs a 4-cycle set-key operation per D, a 3-cycle insert-key operation per D, and a 3-cycle test-and-set operation per D, respectively. During the Read Direct instruction, a 4-cycle store operation per D is made.
2. Q. Holds the doubleword containing the instruction being executed. It may also hold the next sequential doubleword if a Q-refill operation occurred during I-Fetch. The Load PSW instruction refills Q with the next instruction regardless of its storage location.
3. R. Contains the instruction to be performed after execution of the status switching instruction.
4. E. Contains the status switching instruction (or the first 16 bits of the instruction) being executed. E(0–7) contains the R code for the Write Direct, Read Direct, and Diagnose instructions, and E(8–15) contains an immediate operand. For the Supervisor Call instruction, E(8–15) contains a supervisor call interruption code. For the Insert Storage Key instruction, E(8–11) contains the address of the GPR into which the protection key is to be inserted.

Table 1-13. Status Switching Instructions

Instruction	Mnemonic	Op Code	Format	Operands	Function	Program Interruptions	Condition Code
Diagnose	None	83	SI	D1(B1) I2	Load word designated by stg opr adr into MCW, set or reset certain control triggers, & branch to ROS adr specified by MCW.	Priv Oper Prot (S,F) Adr Spec	Unpredictable
Insert Storage Key	ISK	09	RR	R1 R2	Insert stg protection key for 2048-byte stg block, adr by bits 8–20 of 2nd opr (in GPR per R2), into bits 24–28 of 1st opr (in GPR per R1). 1. 1st opr: bits 0–23 are unchanged; bits 29–31 are cleared. 2. 2nd opr: bits 0–7 & 21–27 are ignored; bits 28–31 must = 0's. 3. Key is fetched twice because of 2-way interleaving.	Priv Oper Adr Spec	Unchanged
Load PSW	LPSW	82	SI	D1(B1)	Load doubleword stg opr (designated by stg opr adr) into CPU, thus replacing current PSW, & branch to new instr sequence. 1. Bits 0–15: system mask, protection key, program state. Bits 16–33: ignored. Bits 34–39: CC, program mask. Bits 40–63: instr adr. 2. If PSW(14) = 1, enter Wait state. 3. If PSW(15) = 1, enter Problem state. 4. Load PSW instr is only instr available for entering Problem or Wait state.	Priv Oper Prot (F) Adr Spec	Set by new PSW bits 34 & 35
Read Direct	RDD	85	SI	D1(B1) I2	Send 'direct control read out' signal & timing signal code (I2; in instr) to external device for about 0.6 usec; store 1 data byte from external device into stg (per stg opr adr) when 'direct control hold in' signal is absent.	Oper Priv Oper Prot (S) Adr	Unchanged
Set Program Mask	SPM	04	RR	R1	Replace CC & program mask (bits 34–39) of current PSW with bits 2–7 of 1st opr (in GPR per R1).	None	Set by opr 1 bits 2 & 3
Set Storage Key	SSK	08	RR	R1 R2	Set stg key (bits 24–28 of 1st opr; in GPR per R1) for 2048-byte stg block (adr by bits 8–20 of 2nd opr; in GPR per R2) into stg protection logic in main storage. 1. 1st opr: bits 0–23 & 29–31 are ignored. 2. 2nd opr: bits 0–7 & 21–27 are ignored; bits 28–31 must = 0's. 3. Key is set twice because of 2-way interleaving.	Priv Oper Adr Spec	Unchanged
Set System Mask	SSM	80	SI	D1(B1)	Replace system mask (bits 0–7) of current PSW with byte from location designated by stg opr adr.	Priv Oper Prot (F) Adr Multisys	Unchanged
Supervisor Call	SVC	0A	RR	I	Cause supervisor call interruption; replace old PSW(24–31) with I-field (bits 8–15) of instr, providing interruption code. 1. Clear PSW(16–23). 2. Store old PSW at stg location 32 (decimal). 3. Fetch new PSW from stg location 96 (decimal).	None	Unchanged
Test & Set	TS	93	SI	D1(B1)	Test high-order bit (bit 0) of stg opr byte (in stg), set CC according to state of tested bit, & set addressed byte back into stg as all 1's.	Prot (S,F) Adr	0 : High-order bit = 0 1 : High-order bit = 1
Write Direct	WRD	84	SI	D1(B1) I2	Send 'direct control write out' signal & timing signal code (I2; in instr) to external device for about 0.8 usec; make 1 data byte from stg (per stg opr adr) available to external device until next WRD is executed.	Oper Priv Oper Prot (F) Adr	Unchanged

5. D. Contains the main storage address for storage requests issued during execution of the Set Storage Key, Insert Storage Key, Read Direct, and Test and Set instructions. This register also selects the byte to be used in the Set System Mask and Write Direct instructions, and selects the halfword containing the instruction to be executed after the Load PSW instruction.
6. IC. Contains the main storage address of the next instruction during execution of the Load PSW instruction.
7. AB. Buffers operands for the serial adder and the parallel adder. During I-Fetch of the Set Program Mask, Set Storage Key, and Insert Storage Key instructions, the first operand is placed here. During the Set System Mask, Diagnose, and Test and Set instructions, doublewords from storage are received here.
8. ST. Buffers operands for the serial adder and the parallel adder. Data is received here from main storage for the Load PSW instruction and from LS for the Set Storage Key and Insert Storage Key instructions. Data is stored from here into main storage during the Read Direct instruction and from LS during the Insert Storage Key instruction. The Load PSW, Set Program Mask, Set System Mask, and Test and Set instructions cause all or part of the PSW register to be changed per ST. The Diagnose instruction causes the MCW register, scan counters, and ROSAR to be changed per ST.
9. ABC and STC. Controls selection of data from and placement of data into AB and ST, respectively. Also, during the Read Direct and Test and Set instructions, STC sets a mark trigger.
10. Mark. Identifies the byte to be used by main storage during the Read Direct and Test and Set instructions. All mark triggers are set during the Set Storage Key instruction by a ROS micro-order.
11. F. Buffers the storage key before it is placed into main storage during the Set Storage Key instruction and after it is taken from main storage during the Insert Storage Key instruction. This register also buffers data received from an external device during the Read Direct instruction.
12. G. Buffers a byte of data being sent to an external device when executing a Write Direct instruction.
13. PSW register. Contains a portion of the current PSW. All or part of the PSW register contents is changed directly by the Load PSW, Set Program Mask, Set System Mask, and Test and Set instructions. Because the Supervisor Call, Write Direct, Read Direct, and Diagnose instructions may cause an interruption after being executed, they may indirectly change all of the PSW register contents.
14. MCW register. Controls CPU or channel diagnostic functions during and after execution of the Diagnose instruction.
15. Parallel adder. Provides the data transfer path between AB, ST, D, and the IC. Adds 8 to the IC and D for address updating. Subtracts 8 from A during the Set Storage Key and Insert Storage Key instructions so that a re-entrant loop may be constructed. Calculates IC - D + 7 for the address store compare tests made during Read Direct, Diagnose, and Test and Set instructions.
16. Serial adder. Provides the data transfer path from AB to ST and G; also, during the Read Direct instruction, assigns odd parity to the data byte transferred from F to ST. During the Insert Storage Key instruction, the contents of F are logically OR'ed with the contents of T via the serial adder.
17. LS. Contains operands required by the Set Program Mask, Set Storage Key, and Insert Storage Key instructions. Only the Insert Storage Key instruction transfers data into LS.

Program Interruptions

Six program interruptions can occur during execution of status switching instructions:

1. Operation. Occurs if the Direct Control feature is not installed or not enabled and the instruction being executed is either Read Direct or Write Direct. The operation is suppressed.
2. Priviliged Operation. Occurs if a Load PSW, Set System Mask, Set Storage Key, Insert Storage Key, Write Direct, Read Direct, or Diagnose instruction is encountered while the CPU is in the Problem state. The operation is suppressed.
3. Protection. Occurs if the storage key of the location designated by the instruction does not match the protection key in the current PSW. The instruction is suppressed on a store violation, except for the Read Direct and Test and Set instructions, which are terminated. The operation is terminated on a fetch violation.
4. Addressing. Occurs if an address designates a location outside the available main storage. The operation is terminated, except for the Diagnose instruction, which is suppressed.
5. Specification. Occurs if (1) the operand address of a Load PSW or Diagnose instruction does not have 0's in the three low-order bit positions, or (2) the block address specified by the Set Storage Key or Insert Storage Key instruction does not have 0's in the four low-order bit positions. The operation is suppressed.
6. Multisystem (Multisystem feature only). Occurs if Set System Mask instruction is encountered when in Multisystem mode. The operation is suppressed.

Condition Codes

Three status switching instructions affect the condition code: Load PSW, in which the CC is set by new PSW(34,35); Set Program Mask, in which the CC is set by

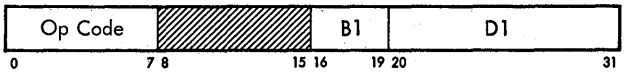
bits 2 and 3 of the first operand; and Test and Set, in which the CC is set to 0 if the high-order bit of the addressed byte in storage equals 0 or set to 1 if the high-order bit equals 1. The remaining instructions leave the CC unchanged, except for the Diagnose instruction in which the CC is unpredictable.

Input/Output Instructions

The 2065 has four I/O instructions: Start I/O, Test I/O, Halt I/O, and Test Channel (Table 1-14).

Instruction Format

The four I/O instructions use the SI format:



Bits 8–15 are ignored. The base plus the displacement determines the channel and I/O unit address: bits 16–23 of the sum are the channel address (of which only bits 21–23 are valid), and bits 24–31 of the sum are the I/O unit address.

Data Flow

Although the CPU operation is essentially the same for all I/O instructions, the Start I/O instruction is used to illustrate the data flow because the channel operation is more extensive. The Halt I/O, Test I/O, and Test Channel instructions do not fetch a CAW or a CCW, do not transfer data, and do not cause an I/O interruption of succeeding CPU operations.

To illustrate the channel operation, the IBM 2860 Selector Channel is used. The Start I/O instruction sets the 'timing gate' trigger, which gates a 'channel select' signal to select 1 of 7 channels (Diagram 3-8, FEMDM). With the 'timing gate' trigger set, the CPU cannot proceed to the next instruction. The I/O unit address is sent to all channels via the 'unit address bus out' (UABO) line, addressing the proper control unit and I/O unit as specified in the instruction. The channel requests the CAW from main storage address 72 (48, hex), and the first command address (address of the CCW) is set into the data address register via the SDBO lines. The channel requests the first CCW from the main storage address specified in the CAW (now stored in the data address register). Parts of the CCW are set into various channel registers to control the I/O operation. During the CCW fetch cycle, the first command address is incremented by eight bytes and is stored into the command address register to be used to fetch the next CCW, if chaining is specified in the first CCW.

Depending upon the availability of the control unit and I/O unit, the I/O operation may be initiated. In any case, the appropriate CC is generated and sent to the CPU together with a 'release' signal, thus resetting the 'timing gate' trigger. The CPU performs an end-op cycle and proceeds to fetch and execute the next instruction.

If the selected control unit and I/O unit are available, the I/O operation is initiated and the channel performs the operation specified by the CCW; the CPU and the channel

Table 1-14. I/O Instructions

Instruction	Mnemonic	Op Code	Format	Operands	Function	Program Interruptions	Condition Code
Halt I/O	HIO	9E	SI	D1(B1)	Terminate current I/O operation at selected channel & I/O unit. 1. D(13–15) is channel adr. 2. D(16–23) is I/O unit adr.	Priv Oper	0 : Interruption in channel 1 : CSW stored 2 : Halted 3 : Unavailable
Start I/O	SIO	9C	SI	D1(B1)	Select specified I/O unit & initiate channel command to that unit. 1. D(13–15) is channel adr. 2. D(16–23) is I/O unit adr. 3. CAW, which specifies address of 1st CCW, is fetched from location 72 (48, hex).	Priv Oper	0 : Available 1 : CSW stored 2 : Working 3 : Unavailable
Test Channel	TCH	9F	SI	D1(B1)	Test state of selected channel & set CC accordingly. 1. D(13–15) is channel adr. 2. D(16–23) is ignored. 3. State of channel is not affected.	Priv Oper	0 : Available 1 : CSW ready 2 : Working 3 : Unavailable
Test I/O	TIO	9D	SI	D1(B1)	Clear interruption condition in addressed channel or associated I/O units, & set CC according to status of addressed channel & I/O units. 1. D(13–15) is channel adr. 2. D(16–23) is I/O unit adr. 3. CSW is stored at location 64 (40, hex) if: a. I/O unit or control unit contains pending interruption. b. I/O unit or control unit is executing previous operation, or there is pending channel-end/control unit-end for another I/O unit. c. I/O unit or its control unit detects machine error.	Priv Oper	0 : Available 1 : CSW stored 2 : Working 3 : Unavailable

continue to share main storage under control of the priority function of the BCU. If an I/O operation was not initiated or was completed, the channel operation is ended when the CC is set into the CPU, and the channel is freed to perform further operations as initiated by the CPU.

For an I/O write operation, a doubleword is fetched from the main storage location specified by the data address in the CCW (now in the data address register), and is stored into the channel A-register. The contents of the channel A-register are transferred to the channel B-register, the data address register is updated, and another doubleword is fetched from main storage and stored into the channel A-register. The contents of the channel B-register are gated to the bus-out register, a byte at a time, and are transmitted over the bus-out lines to the control unit and to the I/O unit as required. When the channel B-register has transferred the last byte, the contents of the channel A-register are transferred to the channel B-register, and another doubleword is fetched from main storage and stored into the channel A-register.

For an I/O read operation, data is received over the bus-in lines, one byte at a time, from the I/O unit via the control unit, and is set into the bus-in register. The contents of the bus-in register are transferred to the appropriate byte location in the channel B-register; when B is full, its contents are transferred to the channel A-register. The contents of A are stored into main storage, a doubleword at a time, according to the address in the data address register.

When the read or write operation is completed or is terminated because of an error condition, the channel requests an I/O interruption of the CPU to present to the CPU the CC and status byte describing the condition of the channel and I/O unit. If the CPU accepts the interruption request, the channel stores a CSW into main storage address 64 (40, hex) and is freed for further operations. Until the CPU accepts the interruption request, the channel remains unavailable to the CPU.

Program Interruption

The only program interruption that may occur for an I/O instruction is the privileged-operation interruption. It occurs if the CPU is in any state other then Supervisor. The instruction is suppressed before the channel is selected. The CSW, the CC in the PSW, and the state of the addressed channel and of the I/O unit remain unchanged. The interruption code in the program old PSW(16–31) is 00000000 00000010.

Condition Codes

When the CPU is released from an I/O instruction, 1 of 4 CC's is set into the CC register of the CPU and becomes a part of the current PSW. This CC is the result of tests by the CPU, the channel, or the I/O unit, and indicates various conditions that exist in the channel, the control unit, or the I/O unit. The CC's for the four I/O instructions are summarized in Table 1-14. For a detailed discussion of the setting of the CC's, refer to the applicable I/O channel FETOM.

POWER

The CPU wall contains a 75-amp, 2.5-kHz converter/inverter and the necessary high-frequency regulator modules to provide the CPU with dc power. The power of just the CPU may be turned on and off by the CPU READY/OFF switch and the CPU ON pushbutton on the enclosed CE panel. Normally, however, CPU power is turned on and off together with the power of the system. The POWER ON and POWER OFF pushbuttons on the system control panel (and the 2150 Console if it is installed) control the power of the system.

Depressing the POWER ON pushbutton causes the units of the system to be turned on one at a time. This sequencing is done so that the higher turn-on current of all the units is not required at the same time. The CPU is turned on first, followed by the channels and, lastly, the main storage units. The units attached to one channel are turned on before the power of the next channel is turned on. The main storage units are turned on last, preventing inadvertent storing by the CPU or channels during their power-on transition before their control triggers have been reset. The system power-on sequence is completed with a system reset operation. The entire system power-on sequence is performed in seconds.

The system power-off sequence turns off the dc power in the main storage units first, followed by the rest of the system. Again, this sequence is followed to prevent inadvertent storing during the power transition while all control triggers may not be turned off. A switch on most of the units in the system (such as CPU READY/OFF or LOCAL/REMOTE) allows the unit to be bypassed during power-off and power-on sequences. In an emergency, the power of the complete system, regardless of the position of these switches, is turned off by pulling the EMERGENCY PULL switch on the system control panel.

To protect the CPU from thermal damage, it is forced-air-cooled by blowers whenever the CPU is turned on; if the internal temperature gets too high, the CPU is turned off automatically. The CPU is also automatically turned off in an orderly, interlocked manner if an overcurrent, overvoltage, or undervoltage condition is detected on the output of any of the CPU's regulators. Whenever the CPU's power is turned off by any means other than the POWER OFF pushbutton, two manual actions are needed to turn the power back on. The intention of this precaution is to prevent inadvertent restoration of power.

Indicators at the top of the CPU's system control panel warn of a power fault in the CPU, channels, and 2365 Processor Storage units. Also located there are controls and a voltmeter for margining the 6V dc supplies in those units and the 18V dc ROS supply in the CPU.

Chapter 2. Functional Units

This chapter discusses the functional units composing the 2065 CPU, and is divided into seven sections:

Section 1, Timing and Clock Control.
Section 2, Read-Only Storage.
Section 3, Bus Control Unit.
Section 4, Data and Control Registers.
Section 5, Local Storage.
Section 6, Serial and Parallel Adders.
Section 7, Status and Control Triggers.

Each functional unit is described separately, as concerns operation, operational timing, and functional application. Supporting the descriptions are simplified, positive-logic upper-level diagrams, flowcharts, and timing charts.

Section 1. Timing and Clock Control

The 2065 CPU operates with a basic CPU clock cycle period of 200 ns; i.e., a 5-megahertz (5-mHz) clock frequency. Each cycle is composed of clock and not-clock portions, used for data transfer and logic functions, respectively. The 200-ns clock cycle period is divided into twenty 10-ns intervals for intracycle timing.

CLOCK SIGNAL GENERATORS

Two types of clock signal generators are used in the CPU, depending on the system model. CPU's in a Model G65, H65, or I65 system, with a maximum of two high-speed storage (HSS) units, use a 10-mHz continuously running crystal-controlled oscillator and divide-by-two logic to provide the basic 5-mHz (200-ns) clock signal. CPU's in a Model IH65 system with three HSS units, and in a Model J65 system with four HSS units, use a 5-mHz gated delay-line oscillator to provide the basic clock signal. The longer cables to HSS units 3 and 4 cause an additional 10-ns to 15-ns storage response delay, which is accounted for by inhibiting the clock. The gated delay-line oscillator is also used if a 2361 Core Storage Unit (Large Capacity Storage or LCS) is attached or if the Multisystem feature is installed, regardless of the CPU model. The gated delay-line oscillator reduces the time required to restart the clock after the storage request sequence that required it to be inhibited. It is not necessary to wait for the next full oscillator cycle to restart the clock as would be the case if a continuously running oscillator were used (as much as the full 200 ns might be required).

Both types of clock signal generators may be operated with a higher output frequency (5.128 mHz), thereby shortening the clock cycle period 2.5% (from 200 ns to 195 ns) for test purposes. The higher frequency is obtained from the clock signal generators by setting the FREQUENCY ALTERATION switch to the down position and by turning the CE Key switch to the CE position. Both switches are located on the CPU system control panel.

Model G65, H65, and I65 CPU Clock Signal Generator

The Model G65, H65, and I65 CPU's have a continuously running crystal-controlled oscillator with two crystals: 10 mHz and 10.256 mHz. The crystal used is determined by a reed relay. The relay is operated by the FREQUENCY ALTERATION switch and the CE Key switch (Diagram 4-1, FEMDM). With either switch in the normal operating position, the relay is not picked and the 10-mHz crystal is used. With both switches set to the test mode position, the relay is picked and the 10.256-mHz crystal is used.

The output of the oscillator circuit drives the frequency dividing logic, consisting of six inverting AND's. The output of the dividing logic provides the basic 5-mHz (200-ns) or 5.128-mHz (195-ns) clock signal. This oscillator cannot be inhibited by the logic.

Model IH65 and J65 CPU Clock Signal Generator

The Model IH65 and J65 CPU has two gated delay-line oscillators operating at 5 mHz and 5.128 mHz. The oscillator used is determined by the FREQUENCY ALTERATION switch and by the CE Key switch (Diagram 4-2, FEMDM). With either switch in the normal operating position, the 5-mHz oscillator is enabled. With both switches set to the test mode position, the 5.128-mHz oscillator is enabled. The output of the oscillators provides the basic 5-mHz (200-ns) or 5.128-mHz (195-ns) clock signal.

The gated delay-line oscillator can be inhibited by the BCU during a storage request sequence to either HSS unit 3 or 4 (Diagram 4-2). The oscillator is inhibited at the

time the BCU would normally expect the data from the accessed HSS unit (600 ns after the 'select' signal was sent to the unit) and is automatically restarted at the time the data arrives from the unit by a fixed (pre-adjusted) time delay. The oscillator provides stable clock signals immediately; the negative-going signal into the oscillator not only starts the oscillator but becomes part of the first output cycle.

A continuously running crystal-controlled oscillator (identical to the clock signal generator in the Model G65, H65, or I65 CPU; see Diagram 4-1) provides a 5-mHz reference frequency for adjusting the 5-mHz delay-line oscillator frequency. A comparator circuit (Diagram 4-2) mixes the two signals and provides an output that is the absolute difference frequency between the two signals. When the two signals are within 1 kHz of each other, the output of the comparator circuit is a null or is a difference frequency of less than 1 kHz, indicating that the delay-line oscillator frequency is within 0.02% of the crystal-controlled oscillator frequency.

CLOCK TIMING

- Triggers are set and reset at clock time.
- Latches are set and reset at not-clock time.
- Logic operations normally occur at not-clock time; subsequent data transfers occur at following clock time.
- Symmetrical and unsymmetrical clock signals are used, depending on logic function.
- Twenty 10-ns delay intervals provide for intracycle timing.

Throughout the CPU, trigger and latch logic is used for all data-handling and control functions. Although implemented with the same logic components (AND's, OR's, and inverters), triggers (by definition) are set or reset at clock time, whereas latches (by definition) are set or reset at not-clock time. In general, the data registers consist of triggers, and all intermediate logic units (such as adders, incrementer/decrementers, and decoders) consist of latches. Control logic consists of both triggers and latches. The CPU design provides for all intra-CPU data manipulation to be done by register-to-latch-to-register (trigger-to-latch-to-trigger) sequences, in lieu of direct register-to-register (trigger-to-trigger) transfer. Continuous availability of stable data results from the overlapping of the trigger and latch set/reset states (Figure 2-1). Note that the data (e.g., "A" in the figure) is stable in either a trigger or a latch at any one time during the indicated two clock cycles, but that the trigger and latch are available for new data ("B") in the next clock cycle.

Each logic component within the CPU introduces some degree of signal delay which must be considered in the CPU operation. All logic blocks with inversion introduce between 3 ns and 20 ns of delay, with 10 ns as the average. (Some special circuits will introduce either 30 ns or 700 ns.) All logic blocks without inversion introduce less than 3 ns of delay. In addition, approximately 10-ns to 12-ns delay is created by every 6 feet of signal transmission line.

Because of these delays, the clock signal is converted from a 5-mHz symmetrical signal to a 5-mHz unsymmetrical signal, with an 80-ns clock time and a 120-ns not-clock time. This unsymmetrical signal provides the needed extra time for logic operations during not-clock time and still leaves sufficient time for trigger input at clock time.

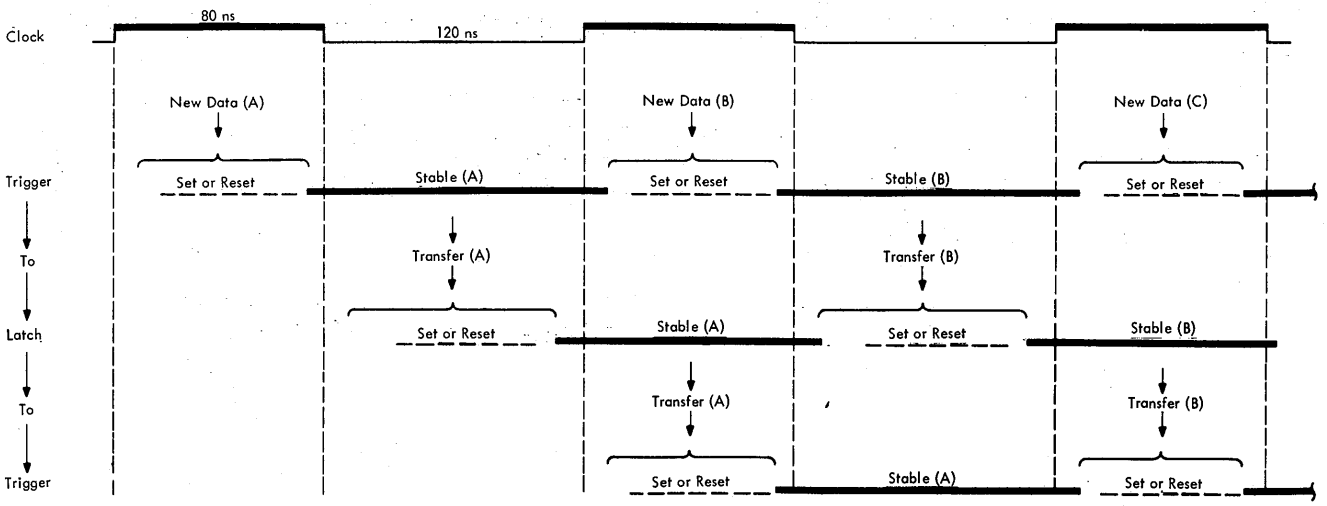


Figure 2-1. Trigger and Latch Data Relationship

Many logic blocks are used in the parallel adder at clock time, thus causing excess delay. To overcome this delay, the unsymmetrical clock signal is extended to a symmetrical clock signal with a 100-ns clock time and a 100-ns not-clock time; this conversion is made within the parallel adder. Some of the parity checking and sign propagating circuits are timed by the unsymmetrical clock signals. Primary concern in clock timing is to provide stable information at sampling time.

Oscillators A through E (Diagram 4-3, FEMDM) set up controls to stop, start, or control the clock. These signals are used in advance of clock time so that the controls are stable when they are needed.

With most of the CPU logic blocks introducing 10 ns of delay, the 200-ns clock cycle period is divided into twenty 10-ns delay intervals to provide timing within a clock cycle (Figure 2-2). These delay intervals are called "B" time or "P" time and are relative to the start of the clock cycle. "B" time refers to symmetrical clock signals; "P" time, unsymmetrical clock signals. The delayed clock signals are created by series inverters (each inverter introducing one 10-ns delay interval) or by time delays (e.g., B0, B1, B2, ... , and P0-1, P0, P1, P2, ...).

CLOCK CONTROL AND SIGNAL DISTRIBUTION

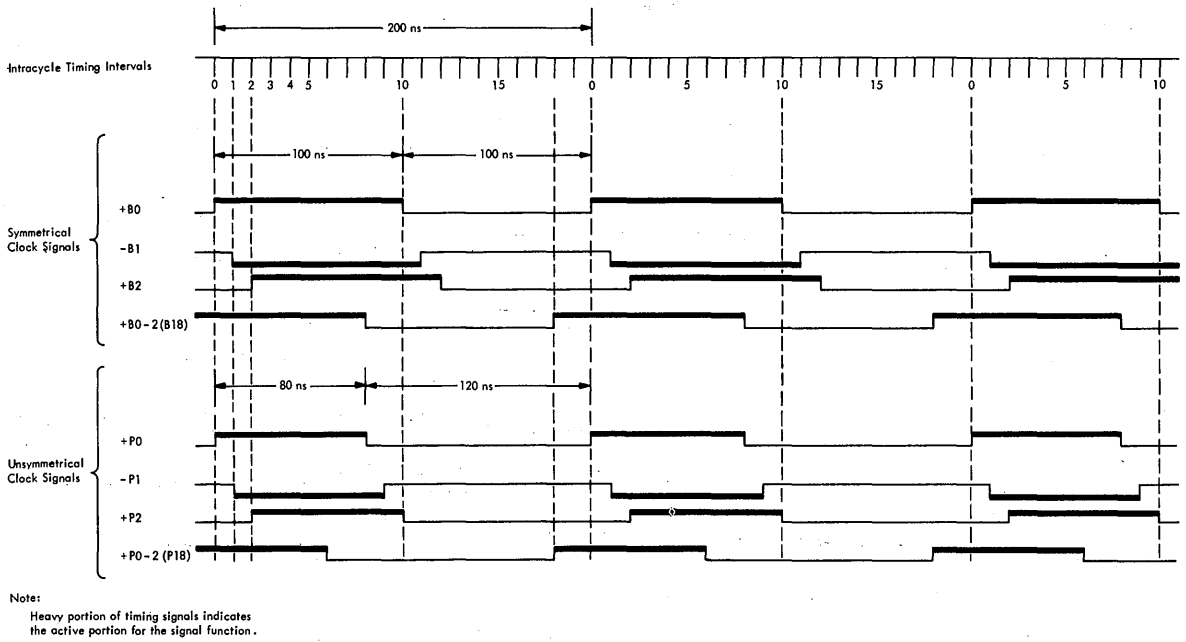
- Manual, BCU, and ROS operations, and errors, affect clock signal availability.
- One symmetrical and one unsymmetrical clock signal are distributed throughout logic gates.
- Time delays unskew and synchronize clock signals within and between sections of logic.

The availability of the clock signals to the CPU processing logic is controlled by the clock-stopping logic (Diagram 4-3). Usually, the clock signals to the BCU are not stopped because the BCU must continue to service the I/O channels even if the CPU is not operating. In the Model IH65 or J65 CPU, however, the BCU has the ability to inhibit the clock signal generator to account for the cabling delays to HSS units 3 and 4 (Diagram 4-2). During maintenance operations (such as scan, log-out, and single-cycle operations), the clock signals may be stopped or permitted to run intermittently.

The 'pass pulse' trigger (Diagram 4-3) provides clock signal distribution control during both normal (continual) and single-cycle operations. A start, load, or reset operation sets the 'pass pulse' trigger and permits the clock signals to be passed on to the logic. When in the single-cycle mode, the 'block' trigger (set by the same operations) resets the 'pass pulse' trigger and blocks the clock signals before the next clock cycle, unless the BCU holds the clock on.

The 'stop clock' trigger (Diagram 4-3) provides BCU control of the clock signal distribution. If the BCU cannot process a CPU storage request immediately or if a request to a large-capacity storage (LCS) unit has been made, the BCU sets the 'stop clock' trigger and blocks the clock signals. The 'BCU cleanup' signal for the HSS, the 'advance' signal for the LCS, or a start, load, or reset operation resets the 'stop clock' trigger.

During certain operations, the ROS microprogram may stop the CPU clock signal distribution for one or two cycles ('STOP1' or 'STOP2' micro-orders). The 'stop clock ROS' trigger provides this control from the bit configuration of ROS word bits 45 and 46 (Diagram 4-3).

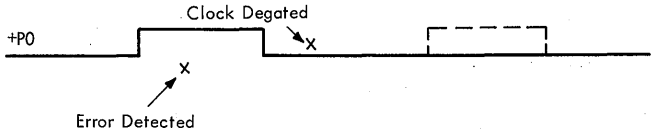


Note:
Heavy portion of timing signals indicates the active portion for the signal function.

Figure 2-2. Typical Clock Signals

The distribution of clock signals to portions of the ROS logic is stopped while the CPU is in the Wait state (Diagram 4-3). The Wait state is entered if PSW(14) = 1 at end op. Entering the stop loop or an interruption removes the wait state block of the ROS signals. The 'time clock step' trigger also removes the wait state block, allows the exceptional branch to step the time clock (Diagram 8-20, FEMDM), and then allows the CPU to re-enter the wait state, if there has been no change in PSW(14).

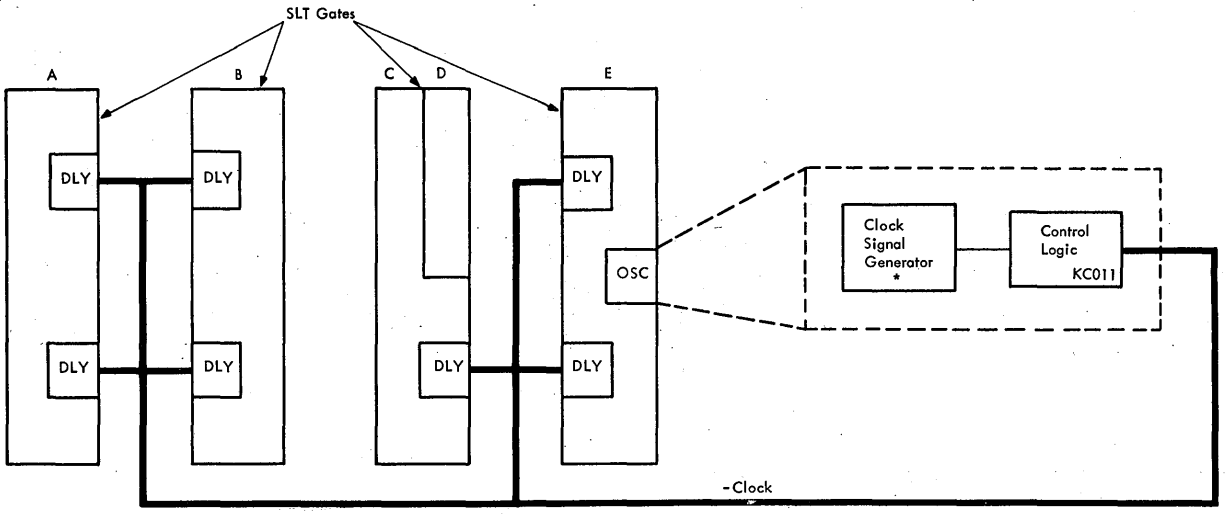
Clock signal distribution to the CPU processing logic is stopped when the 'error' trigger is set and the CPU CHECK switch is not in the DSBL (disable) position. Error detection occurs during clock time, and the clock signals cease with the next not-clock time:



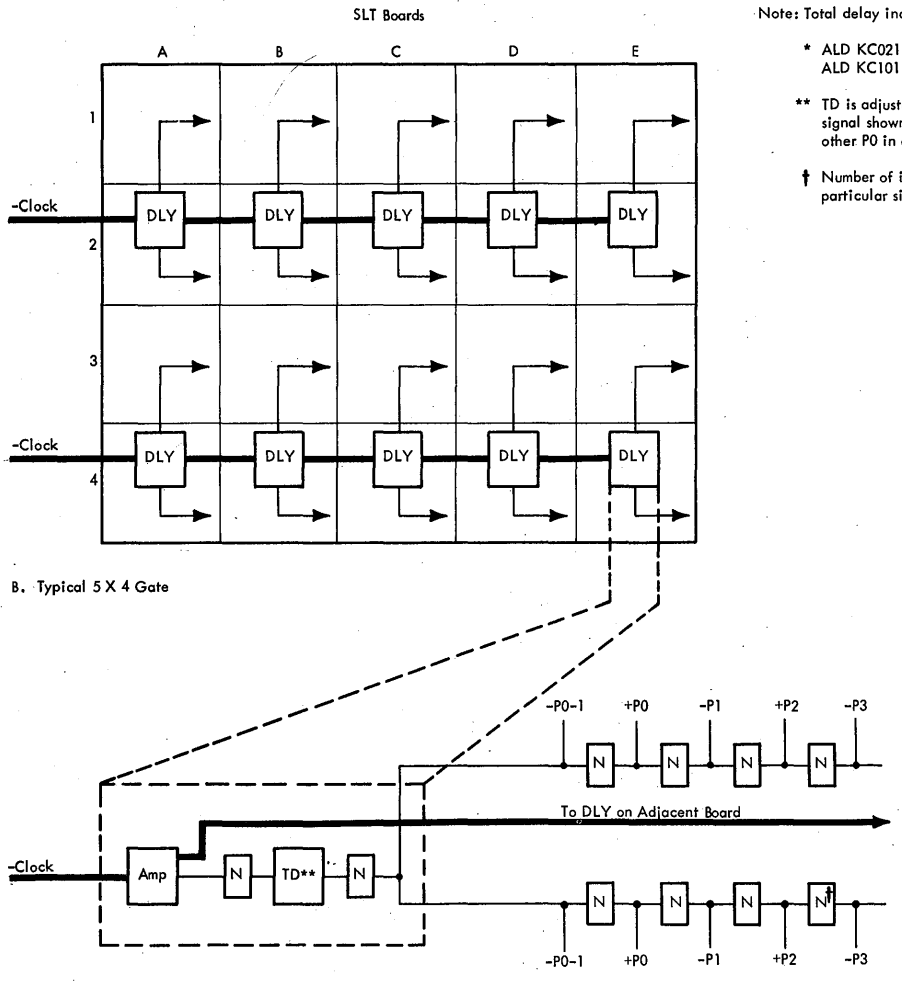
Necessary information for error analysis is thus held in latches and triggers to be examined directly or to be stored by a logout operation. Note that the DLY (delay) units allow the clock to finish the cycle started by the last clock signal. The clock may be restarted by the CPU CHECK switch, internal circuits, or by resetting the 'error' trigger (Diagram 4-3).

The clock signal development and distribution concept is shown in Figure 2-3; note that it is not representative of any logic gate. Figure 2-3 (A) shows how the master 'clock' signal is distributed to the gates. Figure 2-3 (B) shows distribution within a typical gate, with a separate delay logic for every two SLT boards. The adjustable time delay shown in Figure 2-3 (C) allows for unskewing of the clock signals; i.e., aligning all 'P0' signals. The 'special sync for P0 B0' signal is used to adjust all time delays so that all 'P0' signals occur simultaneously; see ALD M8001 for the timing procedure.

www.IBM360.info



A. Clock Distribution



B. Typical 5 X 4 Gate

C. Unskewing Delay Logic

Note: Total delay includes 10 ns per 6 feet of wire plus adjustable time delay.

* ALD KC021 for Models G65, H65, and I65.
ALD KC101 for Models IH65, and J65.

** TD is adjusted to synchronize each P0 with 'special sync for P0 B0' signal shown on ALD ZA001. Each P0 must coincide with every other P0 in any SLT gate.

† Number of inverters varies from gate to gate, depending on need for particular signal.

Figure 2-3. Clock Signal Development and Distribution

The read-only storage (ROS) is a device containing a permanently recorded microprogram used to control CPU operations. The microprogram is in the form of 100-bit micro-instructions (ROS words), each of which has a unique predetermined bit pattern. The ROS words can be read out as required, but a physical modification is necessary to change the stored information. When decoded, the bits of the ROS word condition gates whose outputs perform the necessary functions to execute an operation. Thus, ROS eliminates the need for most complex instruction decoders and sequencing networks, and introduces a flexibility to machine design not previously available in control hardware. This flexibility allows changes to be made to control circuits for special features by replacing printed circuit sheets in ROS.

CAPACITIVE READ-ONLY STORAGE ARRAY

The capacitive read-only storage (CROS) array consists of 2816 100-bit ROS words which are addressed by a 12-bit ROS address register (ROSAR). The array consists of 16 planes, each of which is divided into 4 quarter planes. Each quarter plane has one array driver energizing 1 of 22 select lines. Each select line causes two ROS words to be read out. To address a particular drive line from ROSAR, bits 0–3 select a plane, bits 4 and 10 select a quarter plane, and bits 5–9 energize one select line. Bit 11 of ROSAR selects one of the two ROS words (upper or lower) read out each cycle.

CROS Electrical Theory

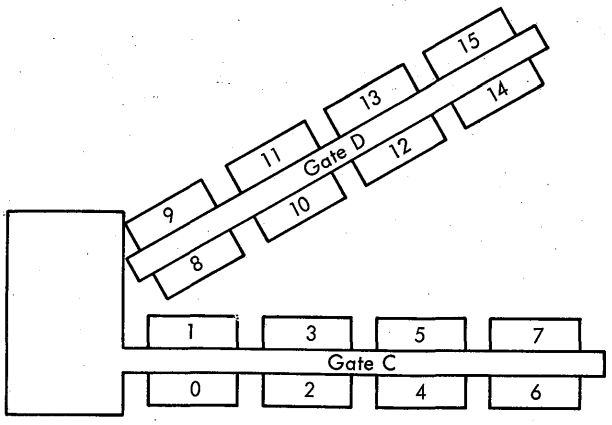
The CROS operates on the presence or absence of a capacitor between a drive line and a sense line. Only one driver at a time may be energized. In the example shown in Figure 2-4, when driver 1 is energized an impulse is coupled through the capacitor C1 to differential sense amplifier D, producing the "D" bit. The same drive results in inputs to differential sense amplifiers A, B, and C, but the polarity is reversed and no bits are generated. To equalize the capacitive load (impedance) to all sense amplifiers, a balance line is provided with each driver and is allowed to "float".

Note: Because the balance line function was found to be unnecessary, later machines do not use the balance line, although it is still printed on the ROS planes.

Some unwanted capacitive coupling exists in this type of matrix. In Figure 2-4, when driver 1 is energized, C1 couples the voltage shift to sense line D, C2 couples the voltage shift to drive line 2, and C3 couples the voltage shift to sense line B. This unwanted signal is very low because it passes through three elements in cascade. The threshold of the sense amplifier is designed so that the low signal is rejected while the desired signal is amplified.

CROS Planes

The 2816 words of ROS are stored in 16 planes. Each plane contains the capacitors, drive lines, balance lines, and sense lines for 176 100-bit ROS words. The drive and balance lines are independent, whereas the sense lines feed common sense amplifiers. Planes 0–7 are on gate C, and planes 8–15 are on gate D:



Drive and Balance Lines (Bit Plates)

The drive and balance lines are photo-etched from a sheet of copper that is bonded to epoxy glass (Figure 2-5). The resulting epoxy sheet with copper drive and balance lines is called a *bit plate*. A separate bit plate controls the bit configuration for each CROS plane.

Tabs at the top and bottom of the bit plate are used for electrical connections to the drive and balance lines. The top tabs connect the drive and balance lines to terminating resistors. The bottom tabs connect the drive lines to the drive circuits.

Four holes in the bit plate align the bit plate to the sense plane. The two outer holes snap over locating studs in the sense plane, and the inner two holes provide clearance for the center studs.

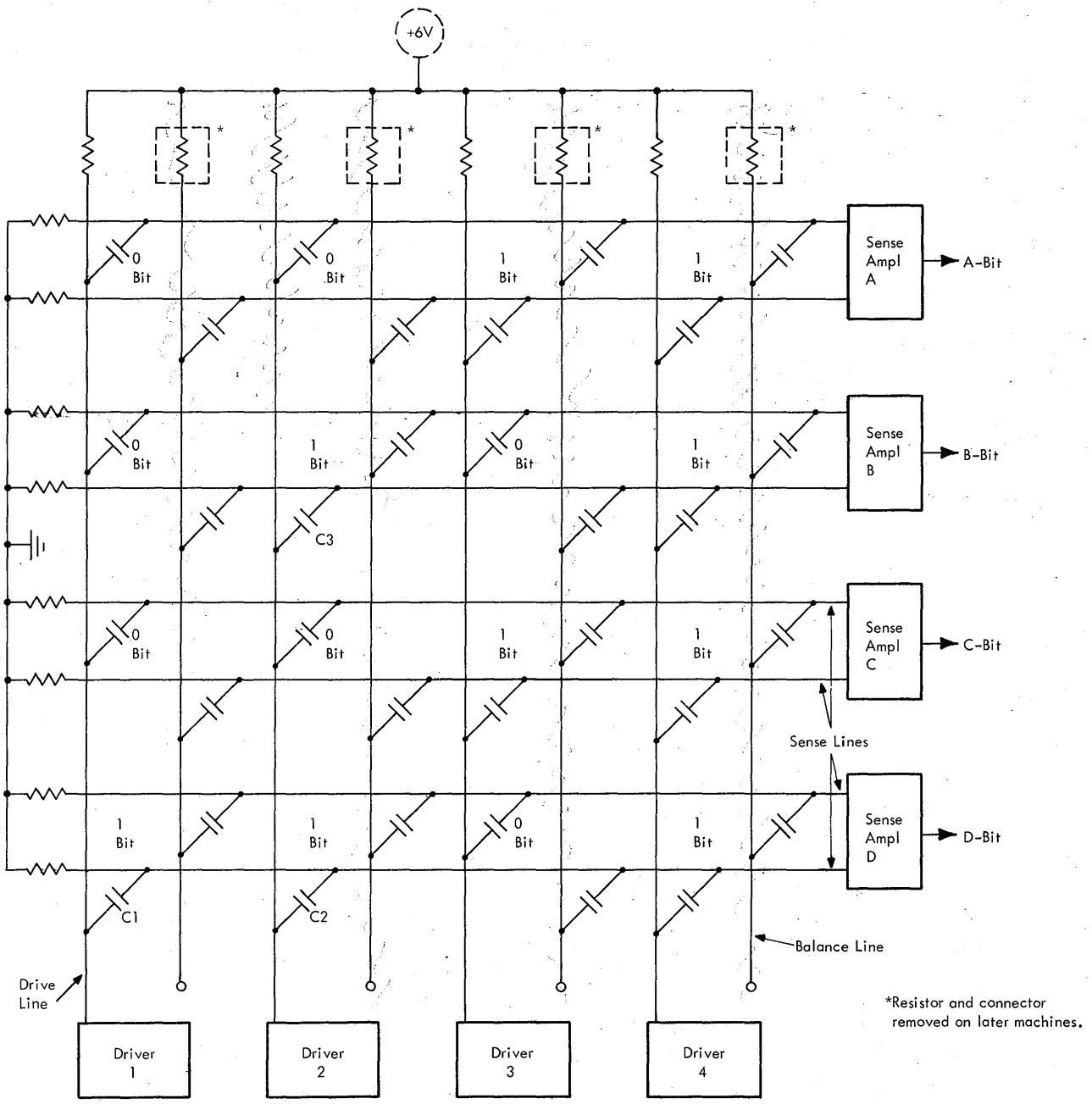


Figure 2-4. Basic 4 X 4 CROS Matrix

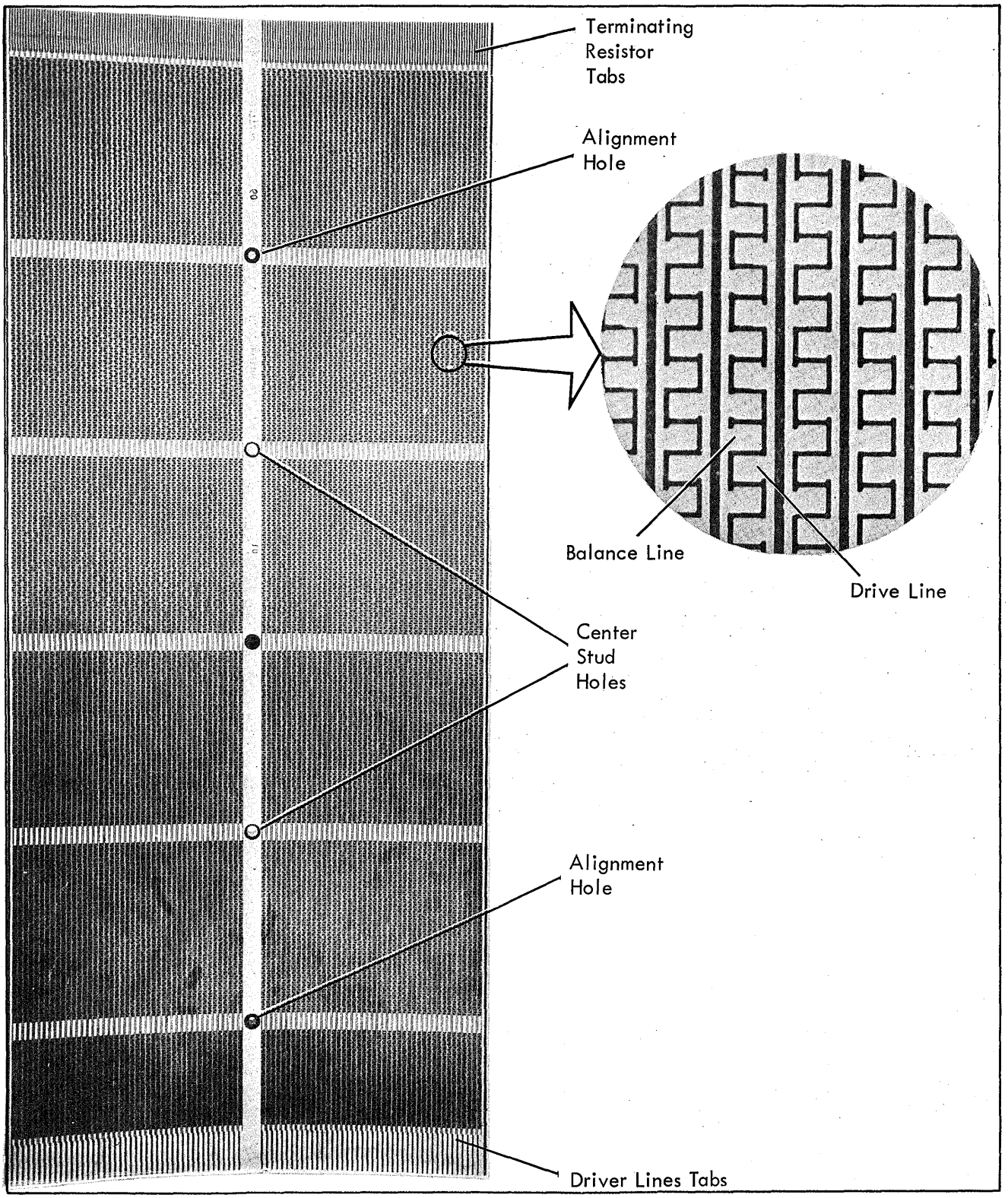


Figure 2-5. Bit Plate

Sense Lines

- 200 pairs of sense lines are in each CROS plane.
- Pair of sense lines carries signal for one ROS-word bit position.
- 200 pairs of sense lines read out two 100-bit ROS words simultaneously.

The sense lines are photo-etched into copper-covered epoxy-glass plates (Figure 2-6). The sense-line plates are permanently mounted to the array gates. Electrical connections from the sense lines to the terminating resistors and sense amplifiers are made with low-temperature solder.

There are 200 pairs of sense lines in each CROS plane. Two sense lines are required to read out one bit position of the ROS word. One drive line simultaneously reads out two 100-bit ROS words, which use 200 pairs of sense lines.

Figure 2-7 shows the layout of the sense lines in the ROS planes. The top pair of sense lines is bit 0 of the upper word. The next lower pair of sense lines is bit 0 of the lower word. This order continues to the bottom pair of sense lines, which is bit 99 of the lower word. The upper and lower words are read out simultaneously. Each sense line is terminated through a resistor to ground. Note the distribution of sense lines through the planes to the differential sense amplifiers. The sense lines through the planes on both sides of a gate are tied together for each bit. The pair of sense lines from each gate is then OR'ed in the sense amplifier for each bit. Because only one plane has an active drive line for a given ROS address, the sense amplifier receives only one input signal.

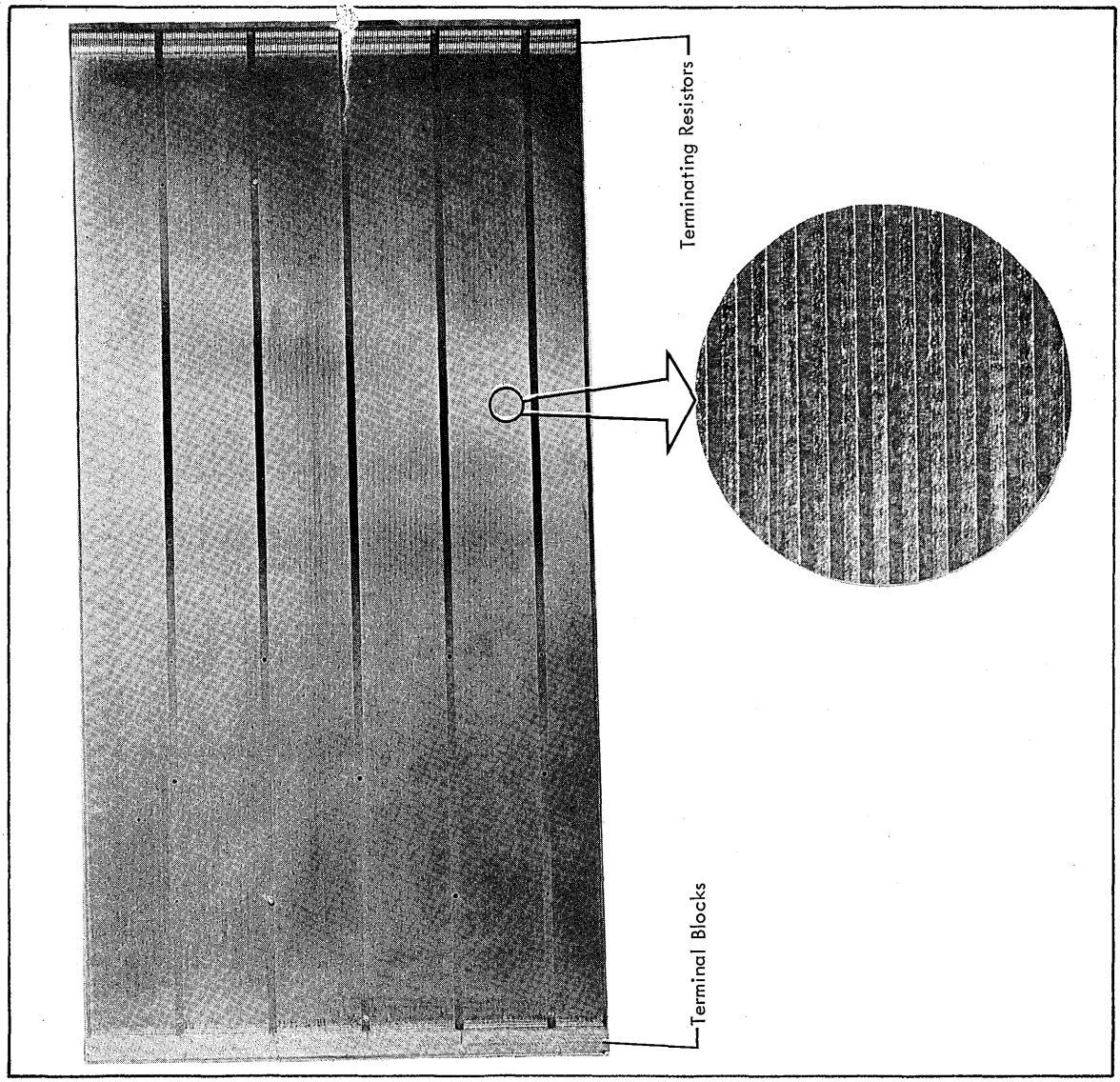


Figure 2-6. Sense Lines

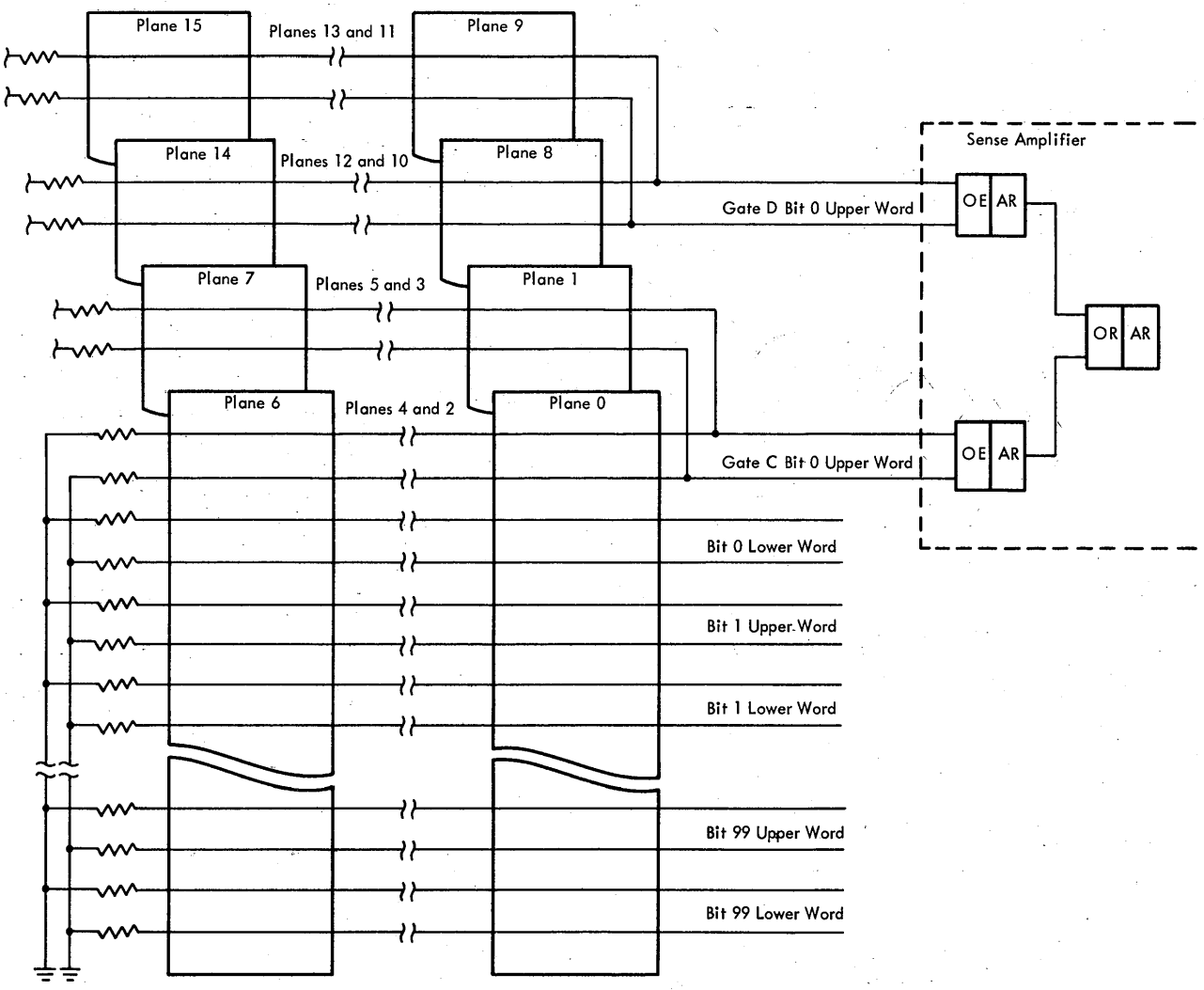


Figure 2-7. Sense Line Layout

Bit Capacitors

The bit capacitors are formed by sandwiching a sheet of Mylar† between the bit plate and the sense lines (Figure 2-8). Pressure plates hold these pieces firmly together. The Mylar is the dielectric, and the drive, balance, and sense lines become the plates of the capacitors.

Tabs on the drive and balance lines increase the size of the capacitors to form the bit configuration. The effective capacitive coupling of a drive line to a sense amplifier is equal to C1 minus C2. The size of this effective capacitor is approximately 0.5 pf.

The bit configuration within a CROS plane is controlled by the bit plate. Therefore, the ROS word can be changed by replacing the bit plate that contains the word.

Physical Package

A CROS plane consists of a sandwich composed of the sense line board, a dielectric sheet, and a bit plate. These pieces are held firmly together by pressure plates (Figure 2-9).

A pressure plate, with a neoprene pad, fits over each group of capacitors in the plane. The plates are loosely connected to a pressure frame that is bolted to the gate. Adjusting screws in the frame squeeze the pressure plate against the bit plate. Because the sense lines are on a rigidly mounted board, the pressure plate holds the bit-capacitor sandwich firmly together.

Electrical connections to the bit plate are also made through pressure connections.

ROS ADDRESSING

ROS word addresses are assembled in the 12-bit read-only storage address register (ROSAR). Each ROS word contains the basic address of the next ROS word. The basic address may be modified by machine operation or by error conditions.

†Trademark of E. I. duPont de Nemours & Co. (Inc.)

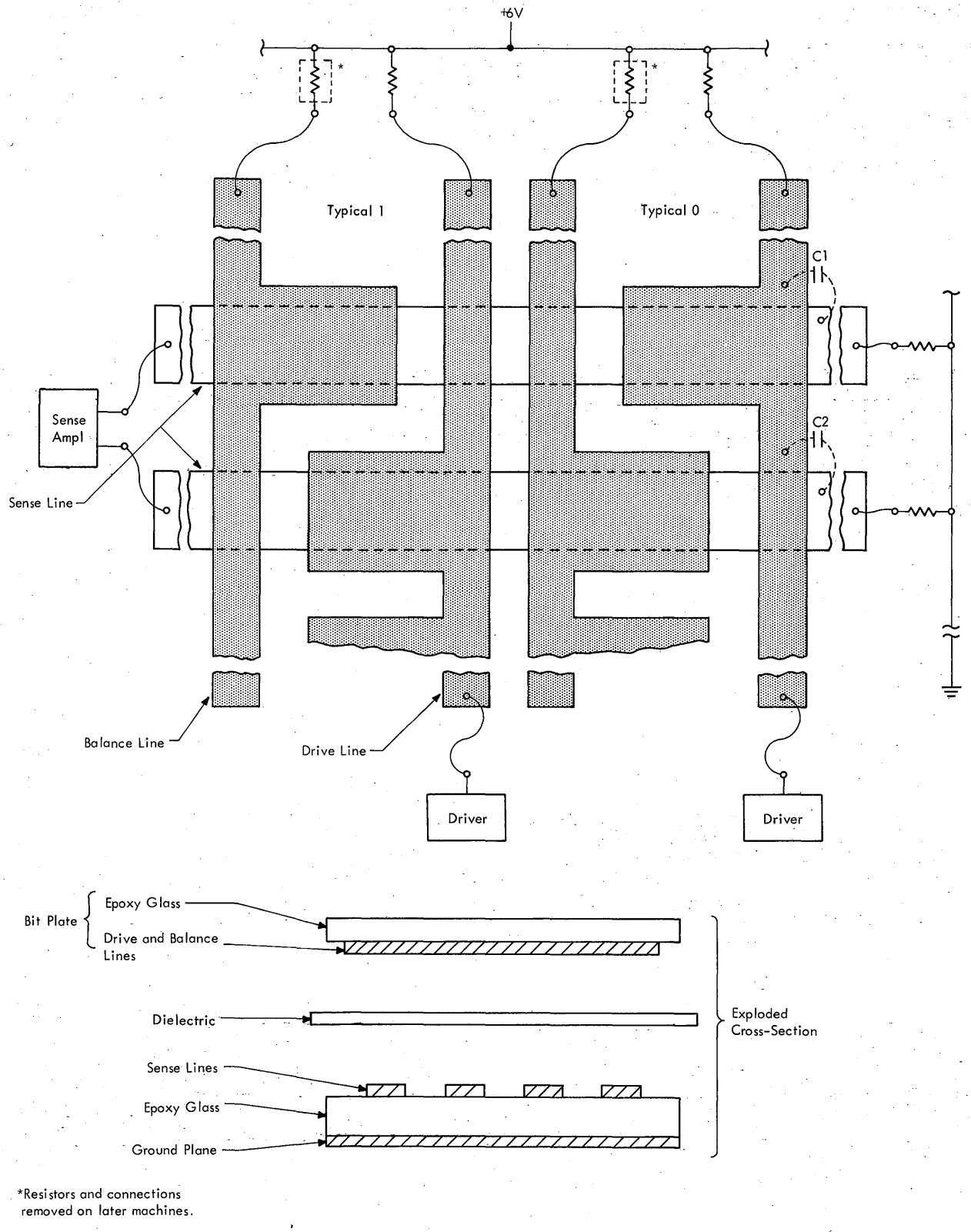


Figure 2-8. Bit Capacitors

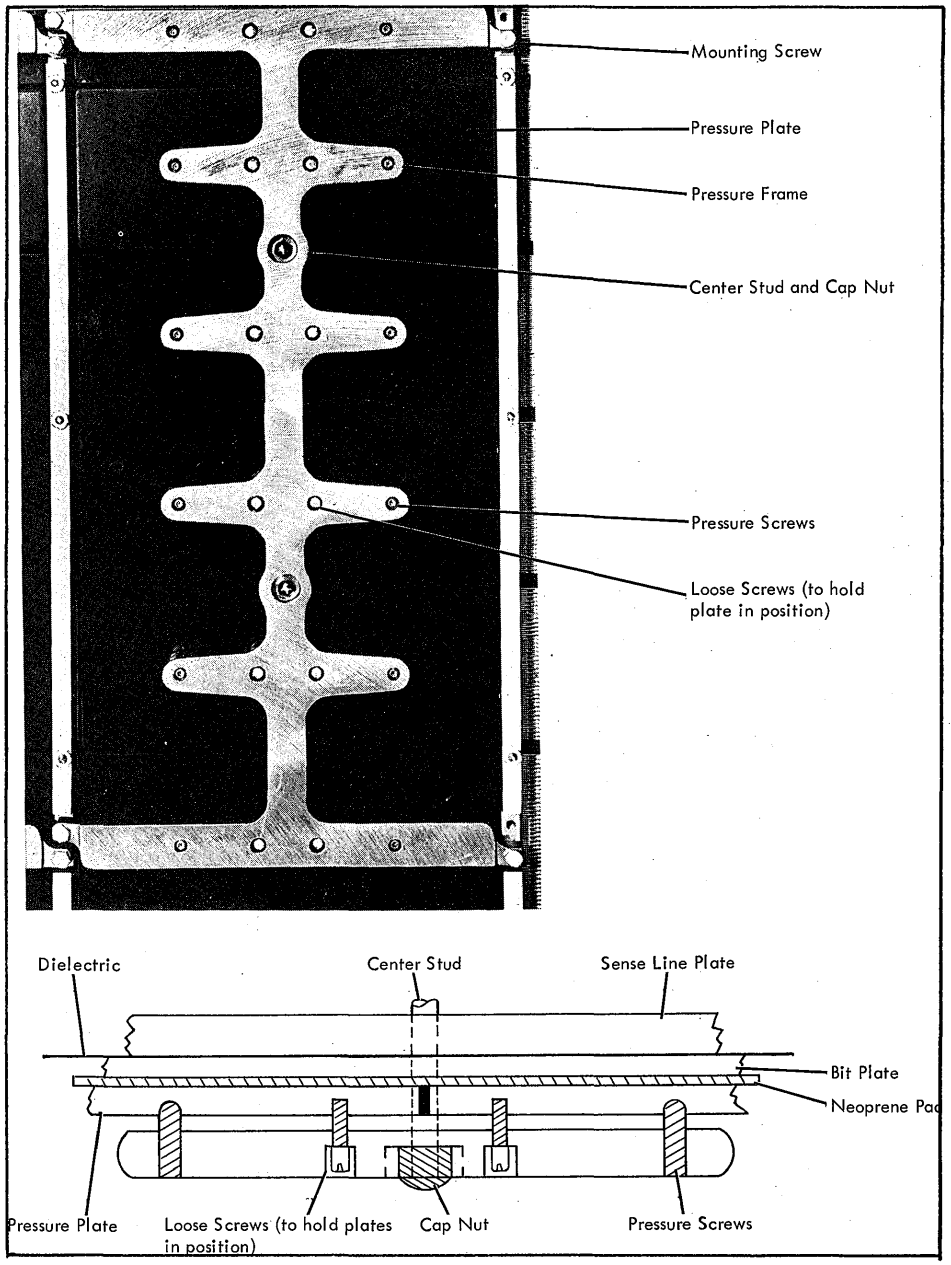


Figure 2-9. CROS Plane Pressure Mounting Assembly

Read-Only Storage Address Register

- ROSAR(0–11) supplies 12-bit address that selects next ROS word.
- Overriding branch and manual ROS operations force new address to ROSAR.

The ROSAR, a 12-position (labeled 0–11) latch register, supplies the address to select the next ROS word. The configuration of the ROSAR contents (address) is controlled by the NA, K, and J control fields of the present ROS word. A new address is available in ROSAR 50 ns after each P2 clock time. Although each ROS word contains the address of the next word to be accessed, address modifications can result by satisfying data-dependent branch conditions. These data conditions are stable at ROSAR(0–10) by P2 + 30 ns and at ROSAR(11) by P2 + 50 ns. The gate at ROSAR is a P4 clock pulse. To prevent late branching, the output of all ROSAR bits must be stable by P2 + 60 ns; at approximately P0 minus 30 ns, the ROS sense latches are sampled. This sequence is repeated every machine cycle.

The ROSAR bit positions can be arbitrarily divided into four groups according to their inputs. These groups are (1) ROSAR(0–5), which normally receives only the six high-order bits of the base address; (2) ROSAR(6–9), which can receive the four low-order bits of the base address and/or the output of the X-branch decoder; (3) ROSAR(10), which can receive data from either the X-branch decoder or the Y-branch decorder; and (4) ROSAR(11), which can receive data from either the X-branch decoder or the Z-branch decoder. [Note the overlap between the base address and X-branches on ROSAR(6–9).] An overriding branch, however, affects all positions of the ROSAR.

The following micro-orders can cause an overriding branch: 'T→RAR,' 'EXCEP', and 'SPEC'. If control field K contains the 'T→RAR' micro-order, the base address is inhibited and T(40–51) is transferred to ROSAR(0–11). This branch is unconditional. An 'EXCEP' micro-order inhibits the base address if an interruption is pending. The source of the interruption provides the branch address. When an 'SPEC' micro-order is specified, a specification program interruption forces the branch address into the ROSAR. In addition, a local store write operation is blocked.

Certain FLT operations force a new address into ROSAR (Diagram 4-101, FEMDM).

Two manual operations cause the contents of the ADDRESS switches to be forced into ROSAR: depressing the ROS TRANSFER pushbutton or activating the REPEAT ROS ADDRESS switch. The operation of these controls is described in Section 1 of Chapter 6.

ROSAR(0–5)

ROSAR(0–5) (Diagram 4-101, FEMDM) can be set from one of three sources: ROS sense latch bits 48–52, T(40–45), or ADDRESS switches 0–5. Normally, positions 0–5 receive the five high-order bits of the base address from the NA control field of the current ROS word contained in the ROS sense latches. If, however, control field K contains an overriding branch and the branch condition is met, or if an FLT operation is in progress and certain conditions are present, or if certain manual operations are being performed, the base address is inhibited from entering the ROSAR and ROSAR(0–5) is set from the ST bus or the ADDRESS switches.

ROSAR(6–9)

ROSAR(6–9) (Diagram 4-102, FEMDM), in addition to receiving the low-order bits of the base address and the overriding branch address, can be set individually by X-branches (functional branches) specified by control field J (bits 62–68) of the ROS word. The control field J micro-orders that specify X-branches are listed as J96 to J124 on ALD M7021, which also shows the ROSAR bits that are set under specific conditions for each X-branch micro-order. When an X-branch is executed, the high-order positions [ROSAR(0–5)] remain unchanged; i.e., still contain the high-order bits of the base address. The base address bit positions corresponding to the bits affected by the X-branch must be set to zero when an X-branch micro-order is given. For example, if ROSAR(6) can be set to a 1 by a certain micro-order, ROS sense latch bit 53 must be 0 in the ROS word that contains the branch.

ROSAR(10)

ROSAR(10) (Diagram 4-103, FEMDM) can be set by an X-branch, an overriding branch, and/or a Y-branch. The base address, however, has no affect on this bit. Also, because an overriding branch and a Y-branch are both decoded from bits in control field K (bits 57–61) of the ROS word, only one of the two branches can be executed at a time; they cannot be executed together. That is, if control field K specifies an overriding branch, a Y-branch cannot be specified, and vice versa.

However, a Y-branch and an X-branch can be executed together because they are functions of micro-orders in separate control fields (K and J). The result of ROSAR(10) is as follows: if neither the X-branch condition nor the Y-branch condition is met, ROSAR(10) remains a 0; if either or both of these conditions are met, ROSAR(10) is set to a 1. The Y-branch micro-orders that affect ROSAR(10) are listed in the K field on ALD M7031 in the same manner as that of X-branches.

ROSAR(11)

ROSAR(11) (Diagram 4-104, FEMDM) can be set by an X-branch, an overriding branch, or a Z-branch. The base address, just as for ROSAR(10), has no effect on ROSAR(11). Because an X-branch and a Z-branch are both decoded from bits in control field J (bits 62–68), only one of the two branches can be executed at one time. ALD M7021 lists the Z-branch micro-orders.

However, a Y-branch and a Z-branch can be executed together. The effects on ROSAR(10,11) are as follows: if neither condition is met, both ROSAR bits remain a 0; if either condition is met, the associated ROSAR bit is set to 1; if both conditions are met, both bits are set to 1's.

If control field K contains an overriding branch and control field J contains a Z-branch and the overriding-branch condition is met, the result of the Z-branch is inhibited and ROSAR(11) is set as specified by the overriding branch.

ROSAR(0–10) Decoding

ROSAR decode logic decodes the address in ROSAR(0–10) to select 1 of 1408 array drive lines. ROSAR(0–4,10) is decoded into 1 of 64 drive lines; ROSAR(5–9) is decoded into 1 of 22 select lines; and ROSAR(0) selects

gate C or D. One select line and one drive line then select 1 of the 1408 array drive lines. See Diagram 4-105, FEMDM, for decode flow. ROSAR(11) is decoded to select one of the two ROS words read out each cycle. (If bit 11 = 1, the lower word is selected; if bit 11 = 0, the upper word is selected.)

Strobed Drive Lines

An ROSAR address selects 1 of 64 strobed drive lines by decoding ROSAR(0–4,10), as shown in A of Diagram 4-105, FEMDM. The decoded address is a gate-drive signal to the array drivers. Each strobed drive line controls the array drivers for one CROS quarter plane.

Decoding is accomplished in two levels. In the first level, bits 3, 4, and 10 are decoded to activate 1 of 8 lines, and bits 0, 1, and 2 activate 1 of 8 lines. The outputs of the two first-level decoders are then combined with a gate-drive signal to activate 1 of 64 drive lines.

Select Lines

One of 44 select lines is activated by decoding ROSAR(0,5–9) (B of Diagram 4-105). Twenty-two of these select lines are connected to gate C, and 22 to gate D. ROSAR(0) is decoded to select the gates, and ROSAR(5–9) is decoded to activate a select line within a gate. Although ROSAR(5–9) can be decoded 32 different ways, only the first 22 combinations are considered valid addresses; the other 10 combinations are not tested. If an illegal bit combination is entered into these bit positions, no select line is activated. Illegal addresses are addresses in which ROSAR(5,6) = 11.

Decoding is accomplished in two levels. In the first level, bits 7, 8, and 9 are decoded to activate 1 of 7 lines, and bits 0, 5, and 6 are combined with a 'gate word select' signal (clock P2 delayed) to activate 1 of 6 lines. (Note that when bits 5 and 6 = 11, no signals are developed from the first-level decoder.) The outputs of the two first-level decoders are then combined in the second-level decoder to activate one of the 44 select lines.

Array Drivers

There are 1408 array drivers in ROS, 704 on gate C, planes 0–7, and 704 on gate D, planes 8–15. Diagram 4-105, C, shows how the drive-line signals are developed and distributed. Each array driver is an AND that AND's 1 of 64 drive lines with 1 of 22 select lines (details are shown in Diagram 4-106, FEMDM). The AND's are single transistors: the drive lines condition the emitters while the select lines control the bases. Voltage is supplied to the collectors through the array drive-line-terminating resistors.

Sense Amplifiers

The sense amplifiers increase the voltage difference between paired 1 and 0 sense lines. The first stage of the sense amplifiers (D of Diagram 4-105, FEMDM) consists of two differential amplifiers, one for gate C sense lines and one for gate D sense lines. Because only one array driver is active for a machine cycle, the sense lines of only one gate carry a signal during a machine cycle. The first-stage differential amplifier increases the voltage difference between paired sense lines and sends this signal to the second-stage amplifier. The second-stage further amplifies the signal and transmits it to the ROS sense latches.

ROSAR(11) Function

Each cycle, 200 bits are sent to the sense latches. ROSAR(11) divides this information into two 100-bit words. If ROSAR(11) = 1, the lower word is selected; if ROSAR(11) = 0, the upper word is selected.

ROS DATA FLOW

ROS word data is transferred from the sense amplifiers to the sense latches. A portion of the word is immediately decoded, while other portions step through registers and latches to provide a delay so that the data is available at the desired time. The ROS word data flow is shown in F of Diagram 4-105.

ROS Sense Latches

The 100 ROS sense latches hold the ROS word for decoding and for setting ROSDR at P0 of the next machine cycle. The sense latches are set by strobing either the upper or lower word sense amplifier outputs and are reset approximately 120 ns after P0 of the machine cycle (E of Diagram 4-105).

ROS Data Register and ROSDR Latches

The ROS data register (ROSDR) holds fields A through H, and M, N, P, and Q of the ROS word for use in the next machine cycle. Fields H, and M, N, P, and Q are decoded directly from the ROSDR to control LS and the adders, respectively. Fields A–G, however, are further delayed by holding them in the ROSDR latches. These fields are used for register ingating.

The ROSDR latches allow a ROS word to control certain gates during the register set time of the next cycle. For example, one ROS word may contain the micro-instruction: add the contents of T and A, and store the answer into A. The ROS word adds the contents of the

registers on one machine cycle and stores the sum from the parallel-adder-out bus at register set time of the next cycle.

Diagram 4-107, FEMDM, is a simplified diagram of ROSDR. In this diagram, each main division of ROSDR is represented by a single bit position. At clock P0-1 of each machine cycle, ROSDR is reset. At not-clock P0-1, the contents of the ROSDR (bits 6–36) are sent to the ROSDR latches. At clock P0, the contents of the sense latches are transferred to the ROSDR. The output of the ROSDR latches (containing the previous ROS word) and the output of ROSDR (38–42, 69–77, and 78–84) are then decoded to perform the selected micro-orders.

ROS Decoders

The ROS decoders use the bits from the ROS word to develop control lines. One micro-instruction may activate a number of control lines. Timing consideration governs the source of the lines; i.e., sense latches, ROSDR, ROSDR latches.

Field A (bits 6–9 of the ROS word) in Figure 2-10 is an example of a decoding network to develop control lines from ROS bits. Line A ['gate M1M2 to PAL(64, 65)'] is decoded from ROSDR because it updates PAL(64,65) before PAL is gated to AB. The 'B38M' micro-order (A7), shown on ALD M7001, uses line A to update PAL(64,65) during not-clock time. Then, lines B, F, and G are developed from ROSDR latches 6–9 to gate PAL(24–67) to AB(24–67) during the next clock time. Note that the 'B38M' micro-order results when bit 6 = 0 and bits 7, 8, and 9 = 111.

This example demonstrates the register-to-latch-to-register timing which controls the source of the decoded control lines. The other micro-order control fields are decoded in a similar manner to provide the control lines at the proper time.

ROS Timing

ROS timing is controlled by the master clock signals. At P0 + 160 ns, the ROS word is strobed (gated) into the sense latches, which are reset at P0 + 120 ns. Data from the sense latches is stable and available at P0-5 ns when it conditions ROSDR(6–42) for setting at clock P0 and ROSDR(69–84) for setting at clock P2. Gate controls from the sense latches (register data transfer) are activated at clock P2, and remain up for 190 ns. ROSDR latches are set at P7 ('not clock P0-1' signal) and initiate register inputs during the following 200 ns. Figure 2-11 shows the timing relationships of the registers and latches. These timings are theoretical and do not show the delays caused by the signals passing through inverters.

Note: Initially, ROSDR is set to all 1's; a 0 in a sense latch position resets the corresponding ROSDR position.

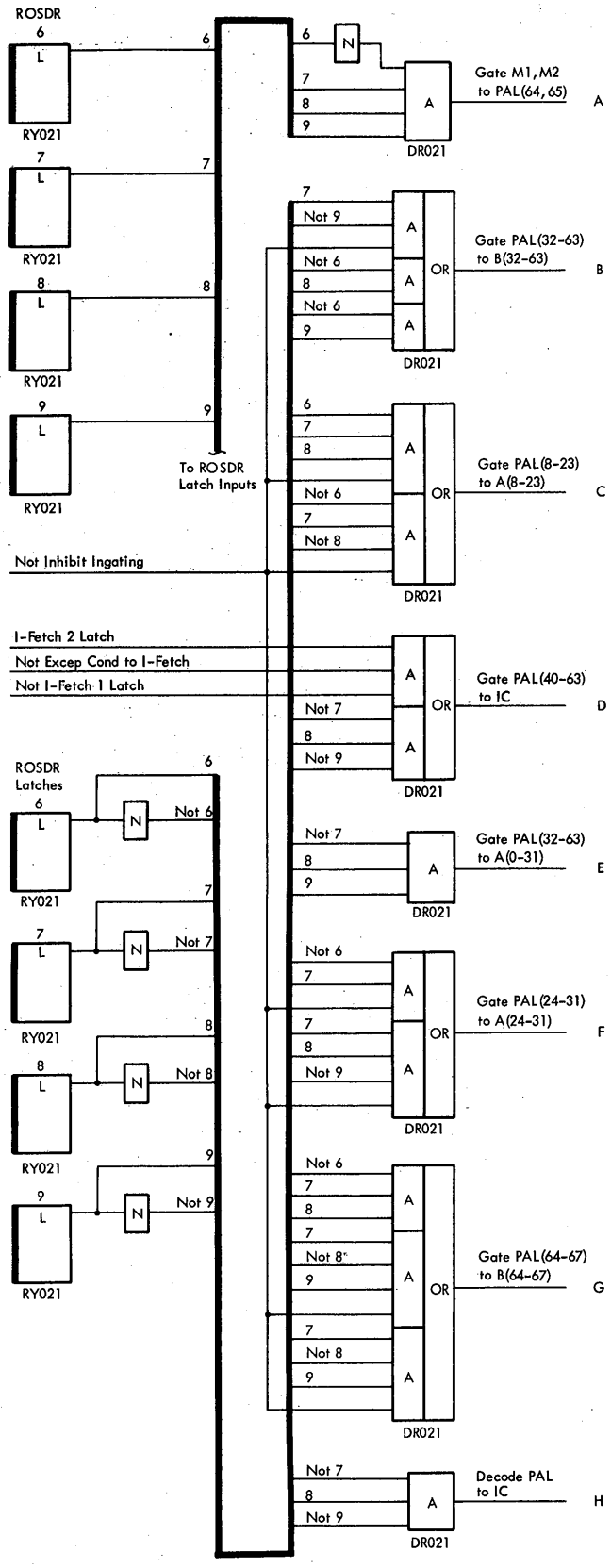


Figure 2-10. Control Field A Decoder

Maintenance Aids

When an error occurs, data leaves ROS registers and latches before the clock is stopped. To retain this information, which identifies the instruction which resulted in the error, secondary registers (which have no other purpose) are provided: ROSAR latches, ROS previous address registers A and B, and ROS backup register.

ROSAR Latches

The ROSAR latches are loaded from ROSAR at P11 time (not-clock P3 time) of each ROS cycle. At P4 time, the latch output is gated to the ROS previous address registers A and B (ROSPARA and ROSPARB) by an alternator. At the next P10 time (not-clock P2 time), the ROSAR latches are reset.

ROS Previous Address Registers

The contents of the ROSAR latches are alternately gated to ROSPARA and ROSPARB, which alternately contain the address of the current and previous ROS words. These registers, which are loaded at P4 time of the ROS cycle, comprise polarity-hold circuits and retain their values until gated into again. Thus, if ROSPARA is loaded on one cycle and ROSPARB on the next cycle, the contents of ROSPARA are maintained until the third cycle, at which time a new address is loaded. The contents of ROSPARA and ROSPARB are indicated on the roller switch indicators: roller 1, position 4, bits 12–23 and bits 24–35, respectively (Diagram 8-2, FEMDM).

ROSPARA and ROSPARB Alternator

The ROSPARA and ROSPARB alternator (Figure 2-12) causes the contents of the ROSAR latches to be sent alternately to the ROSPARA and ROSPARB indicators. Referring to Figure 2-12, assume that the latch is reset, the CPU is at clock P0 time, the A-gate is conditioned, and the B-gate is deconditioned.

At the following P3 time, AND 1 becomes conditioned, which in turn conditions AND 3. The output of AND 1 is also sent to AND 2, but AND 2 cannot be conditioned because the A-gate is set. The output of AND 3 gates the contents of the ROSAR latches to the ROSPARB indicator circuits.

Because the B-gate is deconditioned at P6 time, the latch is set. On the rise of the -P5 signal, the latch being set causes the B-gate to be conditioned and the A-gate to be deconditioned. With the A-gate deconditioned, AND 2 is activated at P3 time (via AND 1) of the following cycle. This action gates the contents of the ROSAR latches to the ROSPARA indicator circuits.

When the -P5 signal drops, the latch is reset. The gates remain in this condition (A deconditioned, B conditioned) until the rise of the -P5 signal. At that time, the A-gate is conditioned by the reset latch, and the B-gate is deconditioned by the A-gate.

During the next two cycles and each cycle thereafter, the operation described above is repeated until the CPU clock is stopped. At that time, the contents of the ROSAR latches are gated to the ROSPAR indicators associated with the deconditioned gate. To indicate which ROS address is in which set of indicators, the latch output

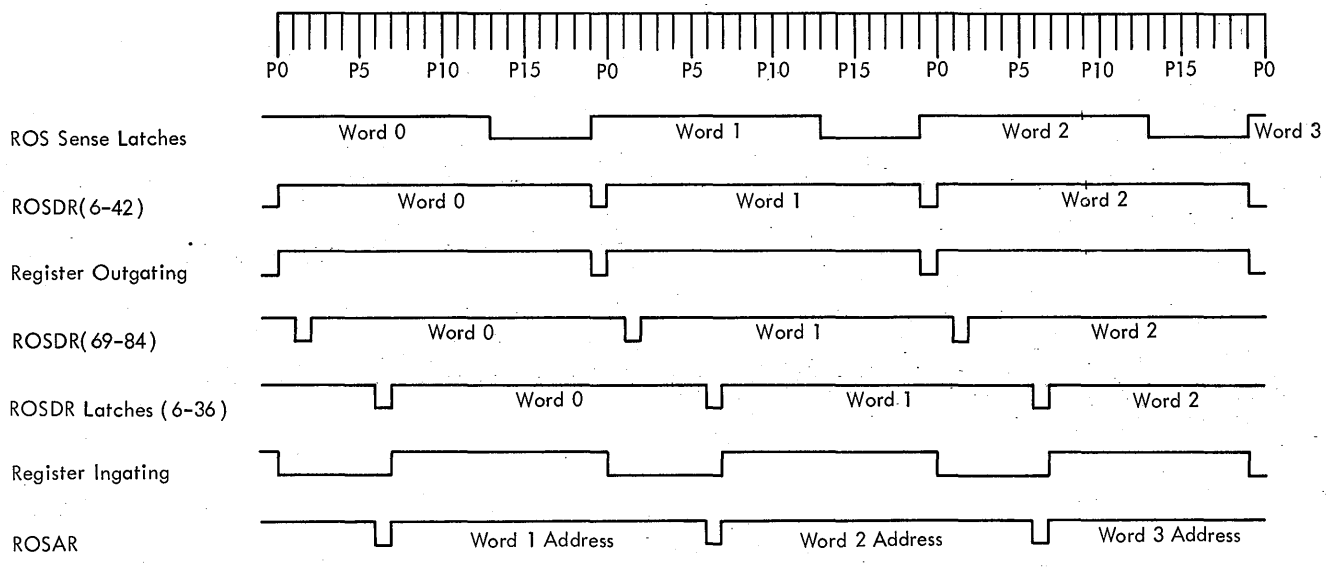
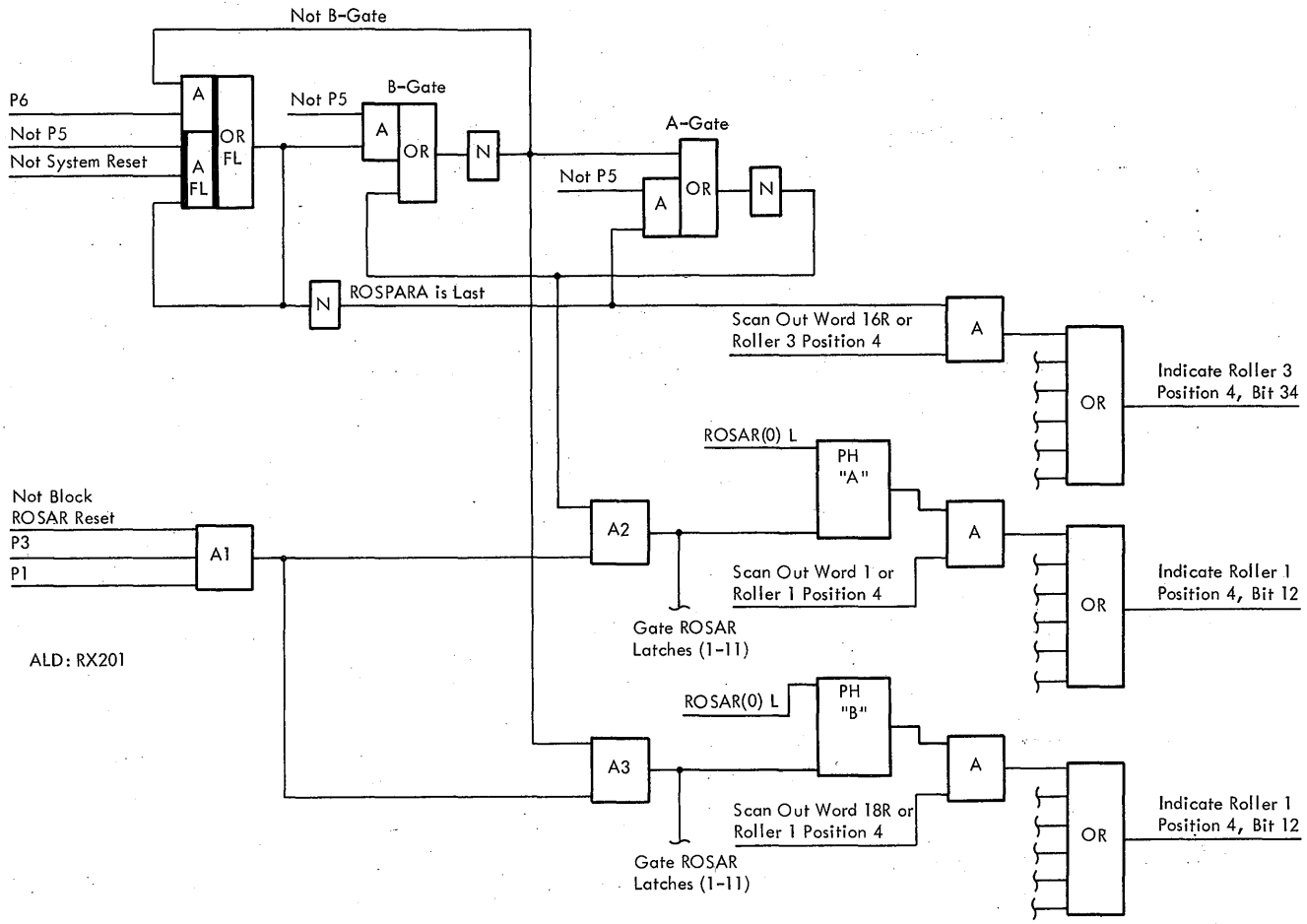


Figure 2-11. Detailed ROS Timing



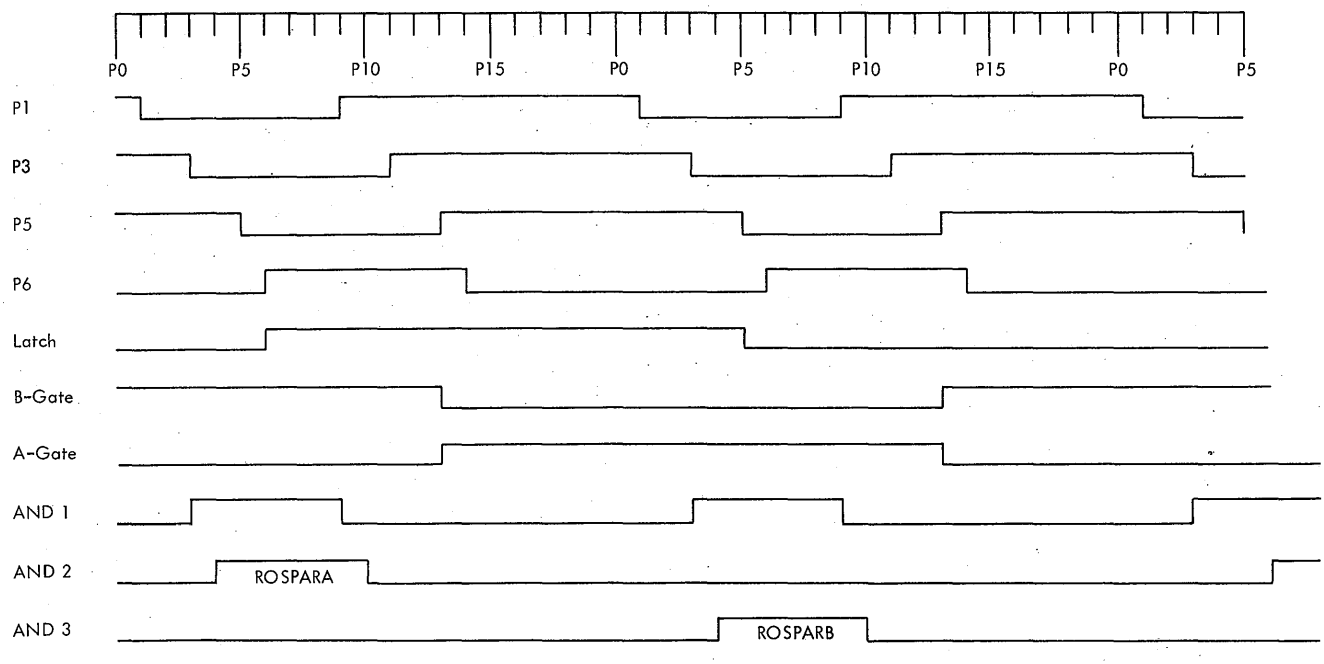


Figure 2-12. ROSPARA and ROSPARB Alternator

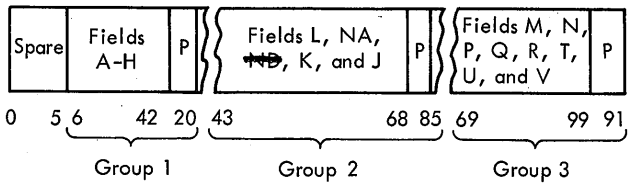
(inverted) is sent to the ROSPARA indicator (roller 3, position 4, bit 34). When the latch is reset, the indicator is on, indicating that ROSPARA contains the address of the current ROS word and that ROSPARB contains the address of the previous ROS word; if off, the contents of ROSPARA and ROSPARB are reversed.

ROS Back-Up Register

The ROS back-up register (ROSBR) holds fields L, NA, K, and J (which are indicated on roller 3, position 4, bits 7–32), and fields R, T, U, and V (which are indicated on roller 4, position 4, bits 17–30). These indicators combined with the ROSDR indicators provide the CE with a picture of ROS word contents when the CPU stops during maintenance (test) mode.

ROS Error Checking

Before each ROS word is decoded, it is checked for correct parity. Parity is checked in three groups from the ROSDR and the ROSBR:



Each group contains its own parity bit and must have an odd number of bits to result in correct parity. (There is also a parity bit for the entire ROS word, bit 0. At the present time, however, this bit is not checked.) Figure 2-13 illustrates ROS parity checking for the three groups: A of the figure shows how parity is checked from the ROSDR for group 1; B of the figure shows how parity is checked from the ROSBR for group 2; C of the figure shows how parity is checked from the ROSDR and ROSBR for group 3.

The clock reset is blocked in that part of the ROSDR or ROSBR containing the failing ROS word. The part or parts not in error are reset, and the next ROS word is gated to its respective register part(s). For example, if bits 43–68 of a ROS word contain an error, the bits are retained for observation. The other two groups not in error (bits 0–42 and 69–99) will change. The groups that change belong to the data word accessed by the failing word when the data register is examined. Thus, when a ROS parity error occurs in one part, the ROS bit indicators on the system control panel comprise bits from two different ROS words.

A ROS parity error also prevents stepping ROSAR, ROSPARA, and ROSPARB, thus enabling the operator to establish the address of the current ROS word, the address of the previous ROS word, and the address of the next ROS word. The address of the previous ROS word should be particularly helpful when parity errors are caused by late ROS branches.

Assume that ROS bit 40 fails. A parity error in bits 6–42 is indicated, and the ROS previous address register indicated the failing word. Reference to the ROS bit plane description shows the expected bit content of the failing word. The incorrect bit (bit 40) can be determined directly by comparing the bit plane description with the indicators.

To summarize, if a machine check is not disabled and a ROS parity error occurs, the parity group in error is not reset at CPU clock time of the next cycle. The CPU clock set-reset signal is blocked to the group that contains the parity error (Diagram 4-107, FEMDM). The new ROS word, however, is gated to the two groups not in error. The bit in error can be determined by displaying the ROS micro-instruction, noting which group of bits is in error, deciding which ROS address is in error, and referring to the listing of ROS micro-instructions. If a ROS parity error occurs and the machine check mask bit [PSW(13)] is set, a logout occurs.

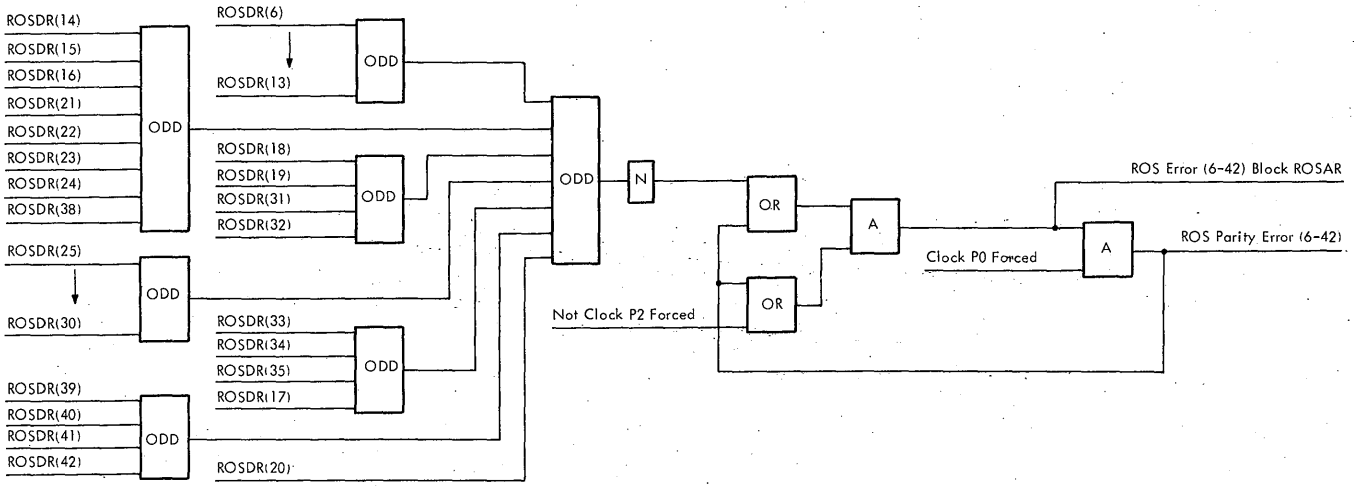
Scan Mode Operations

Scan mode operations affect three fields of ROSDR: field D (bits 17–20), field F (bits 25–30), and field G (bits 31–35). These fields serve dual functions. In the normal mode, they are decoded from the ROSDR latches as standard CPU control lines. In scan mode, they are decoded as special scan control lines and are referred to as field S.

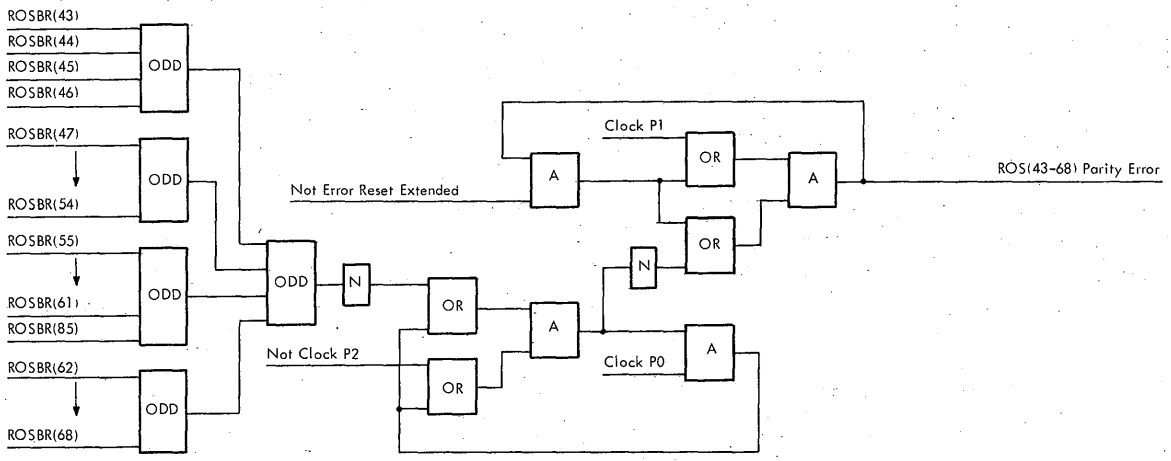
The scan mode is controlled by the 'scan mode' trigger. When the 'scan mode' trigger is reset, the standard decode path is used. When the 'scan mode' trigger is set, however, the standard control lines are blocked and scan control lines (using common CPU control line codes) are activated.

The 'scan mode' trigger can only be set in normal CPU mode and reset only in scan mode. The scan control logic generates an 'inhibit register ingating' signal which is sent to the ROSDR fields to block register inputs and to allow scan control use of the ROS in sequencing through its test operations.

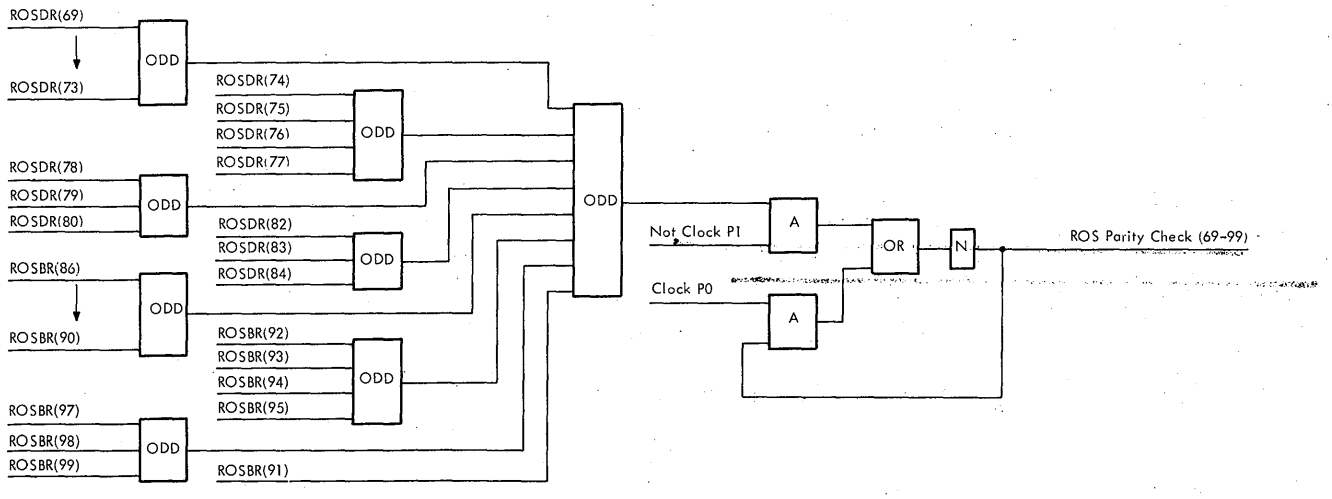
Scan also affects ROS microbranching. (See "Scan Mode Control of ROS" in Section 2 of Chapter 6.)



A. ROSDR(6-42)



B. ROSBR(43-68,85)



C. ROSDR(69-84) AND ROSBR(86-99)

Figure 2-13. ROS Parity Checking

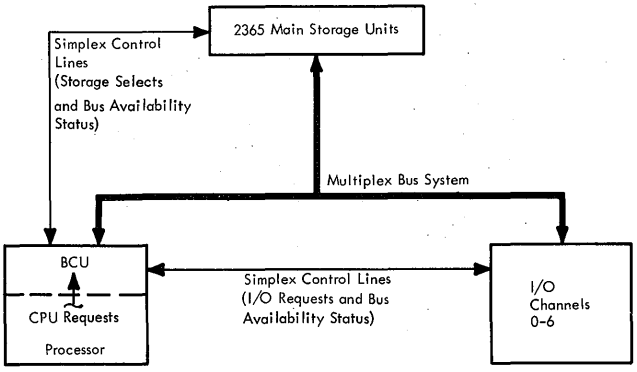
Section 3. Bus Control Unit

The Bus Control Unit (BCU) governs the flow of all information to and from main storage. This unit regulates the flow of addresses, data, key information, and other control signals associated with main storage. In the Model 65 system, the BCU responds to the CPU and to as many as seven I/O channels, all of which are operating asynchronously in respect to each other.

Note: If the Multisystem feature is installed, storage address decoding, 'select' signal generation, and BCU resetting is affected. Refer to Chapter 4, Section 2, for details.

GENERAL DESCRIPTION

The System/360 Model 65 uses a common, or multiplex, bus arrangement to transfer all information to and from main storage:



Each unit requiring access to storage is equipped with a set of receivers and drivers which tap into the multiplex bus. The major function of the BCU is to provide for efficient time-sharing of the multiplex bus by all units. To accomplish this task, each main storage unit and each unit requiring access to main storage communicates with the BCU through individual, or simplex, control lines. These lines are monitored by the BCU to establish priority between the requesting units and to inform the units of the bus availability status.

Basic Interface Considerations

Figure 2-14 is a simplified diagram of the BCU interface with the system. The functions of the major buses and control lines are explained below.

1. Multiplex Buses
   a. Storage Address Bus (SAB). This bus, 21 address lines and 3 parity lines, specifies the address of a doubleword in main storage.
   b. Storage Data Bus Out (SDBO). This bus, 64 data lines and 8 parity lines, carries data sent by the main storage unit.
   c. Storage Data Bus In (SDBI). This bus, 64 data lines and 8 parity lines, carries data sent by the CPU or the channel to main storage.
   d. Mark Bus. This bus, 8 control lines and 1 parity line, designates which data bytes on the SDBI are to be stored into main storage; there is a mark line corresponding to each byte on the SDBI. Complete absence of mark signals on the mark bus occurs only on a fetch operation.
   e. Key-In Bus. This bus, 5 key lines and 1 parity line, is used during storage operations and the set-key operation. These operations transfer the storage protection key from the CPU or channel to the storage protect area in the selected main storage unit.
   f. Key-Out Bus. This bus, 5 key lines and 1 parity line, is used only during the insert-key operation. This operation transfers the storage protection key from main storage to the CPU.
2. Control Lines
   a. Select. Upon selecting the required storage unit, the BCU issues a 'select' signal to that unit. This signal causes the selected unit to perform a storage cycle.
   b. Busy. Throughout the duration of the storage cycle, the storage unit generates a 'busy' signal to the BCU.
   c. CPU Request. Upon requiring access to main storage, the CPU issues a request signal to the BCU.
   d. Channel Request. Upon requiring access to main storage, the channel issues a request signal to the BCU.
   e. BCU-Channel Response. This signal is generated by the BCU to the requesting channel in response to the channel-request signal. This signal indicates that priority has been granted to the channel and that a storage address is now required.
   f. Advance. Indicates to the CPU or channel that the storage unit is about to place data on the SDBO.
   g. Store. This multiplex line indicates to the storage unit that a store operation is in progress.

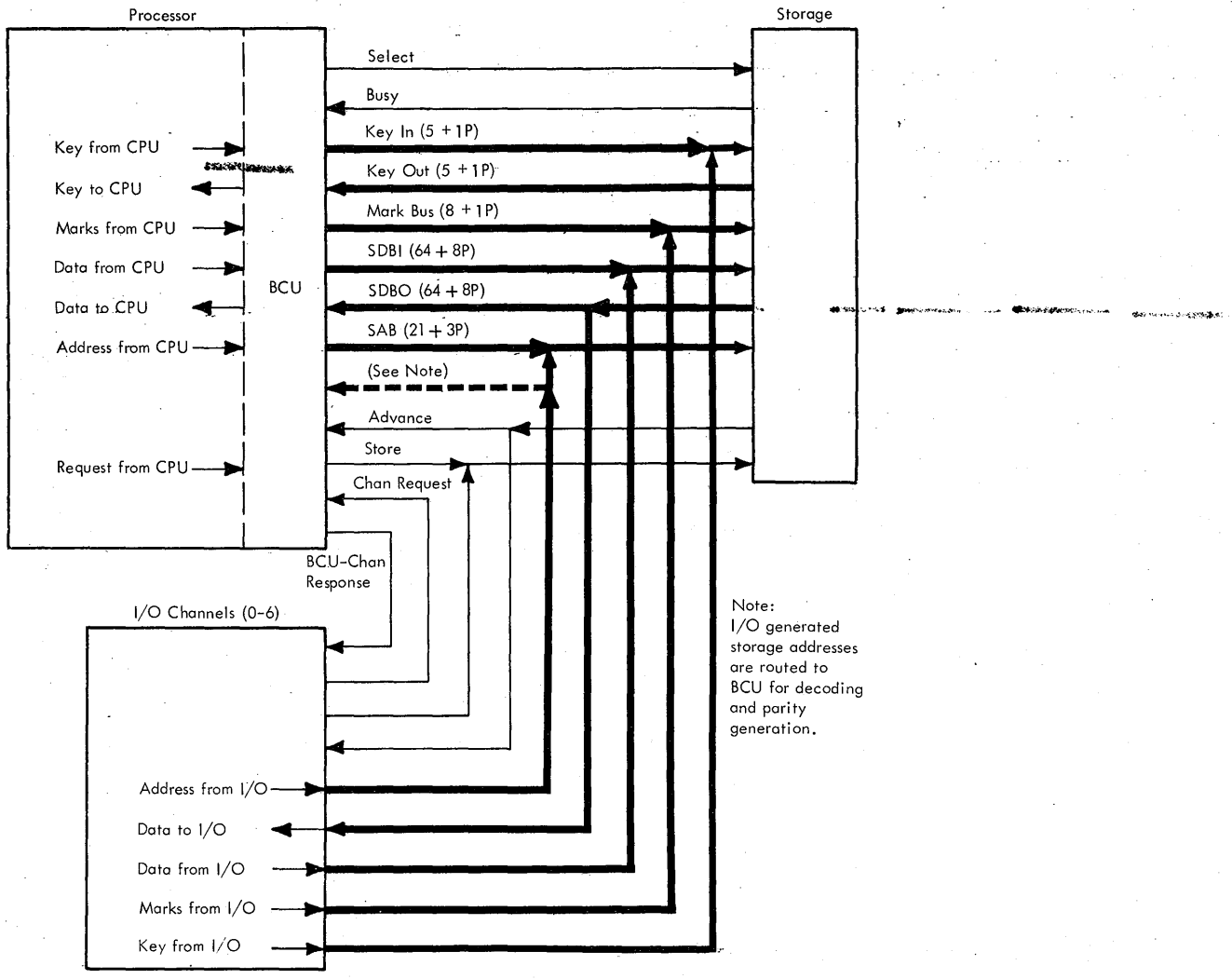


Figure 2-14. Primary BCU Interface Signals

Special multiplex lines for operation with the 2361 Core Storage Unit (Large Capacity Storage or LCS), not shown on Figure 2-14, include:

1. LCS Pre-Advance. Warns the BCU that the LCS is preparing to send data on the SDBO. This signal prevents the start of another request which might interfere with the incoming data.
2. LCS Advance. Precedes the data from the LCS so that a register will be ready to receive the data.
3. Set LCS Priority. Notifies the requesting channel that it should look for an 'LCS advance' signal and ignore the 'advance' signal from the 2365 Core Storage Unit (High-Speed Storage or HSS).

A simplex line, 'LCS operation', from each channel sets the 'channel X waiting' trigger to allow other channels to operate with the HSS while the LCS is busy.

Basic Operating Considerations

The following paragraphs briefly examine the major operational characteristics of the processor, channels, main storage units, and large-capacity storage (LCS) units from a BCU viewpoint.

Operation with Processor

- Requests are issued by (1) I-Fetch, (2) microprogram, (3) Scan.
- CPU clock is stopped if request is not serviced within 3 cycles.

The BCU resides in, and is a logical but independent part of, the CPU. To enable efficient handling of CPU storage requests, the BCU and CPU clocks are synchronized. The BCU is designed to accept storage requests from the CPU every 0.4 usec (every other CPU clock cycle). However, the speed with which these requests are serviced depends on the storage availability and on the amount of channel interference.

The CPU can issue five types of requests to the BCU: fetch data, store data, insert kev. set key, and test and set. Furthermore, the fetch data and store data requests can be issued by various functional areas of the CPU and are categorized accordingly. For example, the fetch data requests can be issued by the I-Fetch control hardware, by the microprogram, and by Scan controls.

Whenever possible, requests from the processor are handled by the BCU in a manner to provide for maximum CPU operation; i.e., processing of data in the CPU is overlapped with handling of the request by the BCU. However, the degree of overlap is limited, and the BCU must stop the CPU clock when a channel request has taken priority, when the storage unit requested by the CPU is busy, and when the data requested by the CPU cannot be retrieved within the required number of cycles.

A need to fetch new data is detected in the processor three or four cycles before this data is required. Accordingly, the CPU issues a 3- or 4-cycle fetch request indicating to the BCU that data will be required after three or four cycles have elapsed from the time of the request (i.e., the transfer of data into the CPU must take place on the fourth or fifth cycle following the request). It is the responsibility of the BCU to ensure that the requested data is present at the SDBO at the specified time. To do this, the BCU allows the CPU clock to cycle three times regardless of whether storage can be accessed within this specified period. If data is not available at the SDBO at the end of the third cycle, the BCU stops the CPU clock. CPU processing is thus halted until its storage request is completed, at which time the BCU restarts the CPU clock (see ALD M8251).

Requests to store data are initiated in the CPU by the microprogram or by the Scan controls; these requests are always accompanied by the 'mark' signals, which designate the bytes to be stored. Store requests are not categorized into the 3- and 4-cycle types because no critical transfer into the CPU is involved; once the storage unit is selected, the CPU no longer depends upon storage unit operations.

The insert-key, set-key, and test-and-set requests are issued by the CPU microprogram. Basically, the insert-key request is a fetch operation to obtain the protection key from main storage. The set-key request is a store operation which transfers the five-bit (plus parity) storage protection key from the CPU into a specified storage protect area of main storage. The test-and-set request is essentially a combined fetch/store operation effected in a single storage cycle.

Operation with I/O Channels

- Requests are issued by up to seven channels.
- Requests are handled in the following order of priority: channels 1, 2, 0, 3, 4, 5, 6; then CPU.

Up to seven channels (six selector plus one multiplexer) are available on the Model 65. All channels may operate at the same time. The channel sustains a maximum data rate when it is servicing a high-speed drum: 1.25 million bytes per second. Because data is transferred between the channels and main storage on an eight-byte basis, the channel makes a storage request every 6.4 usec at this speed. When servicing the 340-kHz and 90-kHz magnetic tape devices, the channel requires access to storage at 23- and 89-usec intervals, respectively.

The channel can issue two types of requests: fetch data or store data. Requests from channels are processed by the BCU on a priority basis. Where there is a conflict between two I/O channels or between an I/O channel and the CPU, the BCU gives priority to the highest-ranking channel. Priority is preassigned in the following manner: Channel 1 has the highest priority, followed by channels 2, 0, 3, 4, 5, and then 6 (channel 0 is the multiplexer channel); when no channel requests remain, a CPU request is honored. Priority is therefore not established on a first-in/first-out basis; instead, as each request is serviced, a priority test is performed again on all channels, including the channel just serviced.

Operation with Main Storage

- BCU allows interleaving of odd and even requests.
- Storage keys protect storage contents.
- Insert key operation fetches key to CPU.
- Set key operation replaces key in main storage.

The 2365 main storage units, also referred to as the high-speed storage (HSS), operate on a basic cycle of 750 ns. Each HSS is divided into an even- and an odd-address storage area equipped with its own SAB, SDBO, SDBI, and 'mark' bus. The access path to these areas is through a common interface (Figure 2-15). This arrangement (internal to the HSS) allows the BCU to interleave even and odd storage requests; i.e., an odd request can be issued by the BCU halfway through an even storage cycle, and vice versa. By interleaving references to even- and odd-address areas, the effective storage cycle approaches one-half the cycle time of the unit. (If desired, the interleaved mode of operation can be defeated to allow operation of a program solely in the odd or even area of storage.)

Each time a storage unit is accessed by the CPU or I/O channels, information is either stored into or fetched from

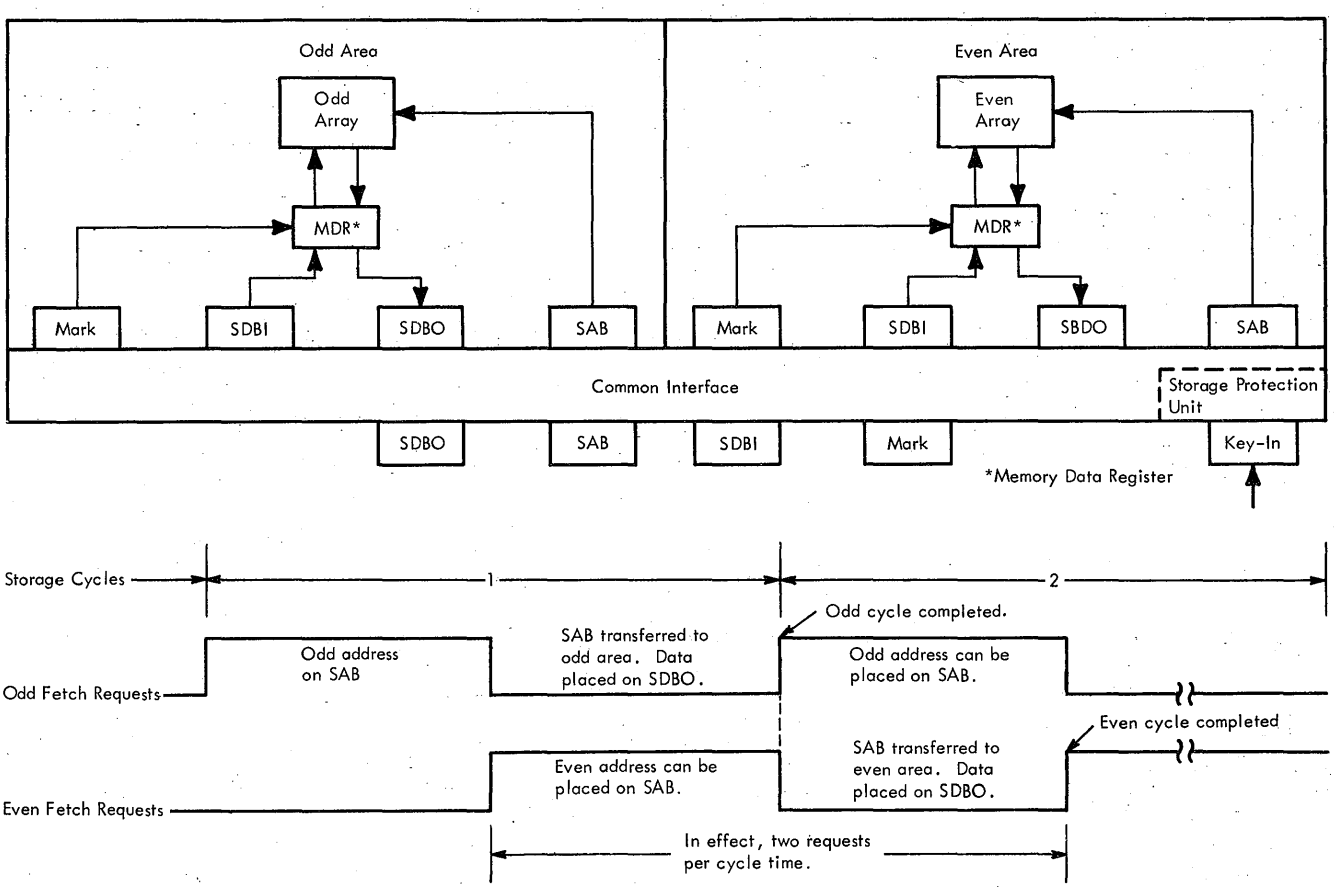


Figure 2-15. Basic Organization of HSS Unit

that unit. A store operation is performed when the data on the SDBI is accompanied by one or more 'mark' signals on the 'mark' bus. There is a 'mark' line for each of the eight bytes that can be placed on the SDBI. The presence of signals on these lines indicates which SDBI bytes are to be stored. Thus, data can be stored selectively by bytes, or, if all marks are active, the entire SDBI contents are stored at the doubleword address specified by the SAB. When the CPU or I/O channel requests a storage cycle and no signals are present on the 'mark' bus, a fetch operation is initiated. In a fetch operation, the doubleword addressed by the SAB is placed on the SDBO and is made available to the requesting unit.

A storage protection capability is provided to protect the contents of main storage from unauthorized use or destruction. Both fetch and store operations are subject to this protection. Protection is implemented by subdividing each storage unit into 2048-byte blocks and assigning a one-byte protection key pattern to each block. The assigned key patterns for the addresses for all 2048-byte blocks within a storage unit are recorded in the storage protection mechanism of that unit. During a storage request, the storage protection mechanism compares the key pattern for the addressed block with the key pattern supplied by the requesting CPU or channel. (Keys are gated from the requesting unit over the 'key in' bus at the beginning of each storage cycle.) A mismatch in keys results in a protection violation; the storage request is not honored, and a 'protect violation' signal is sent to the requesting unit.

The protection key for any 2048-byte block of storage can be fetched from the protection key mechanism and brought into the CPU for inspection. This operation is performed through execution of the Insert Storage Key (ISK) instruction, which issues a fetch request directly into the protection key area of the storage unit. Conversely, the protection key for any 2048-byte block of storage can be changed through execution of the Set Storage Key (SSK) instruction by the CPU. This instruction provides a new key pattern and issues a store request into the protection area.

The ISK and SSK instructions can be executed only in the Supervisor state and are not available to the problem programmer. A programmed protection of storage can be achieved by means of the Test and Set (TS) instruction. When the TS instruction is executed by the CPU, a doubleword is fetched from main storage, and the CPU inspects one byte in that doubleword. The main storage sets the byte inspected by the CPU to all 1's, while the remaining bytes in the doubleword are not changed. Thus

programmed storage protection is achieved in the sense that the CPU can later inspect the first byte of a particular storage block to establish whether this block has been previously processed.

A test-and-set storage request combines aspects of both fetch and store operations; therefore, it requires special handling by the BCU and the storage unit. From the BCU viewpoint, the TS instruction is the only one that generates a 'mark' signal during a fetch request. The BCU sends the 'mark' signal and the 'test and set' signal to the storage unit. During the storage write cycle, all bytes (excluding the byte specified by the 'mark' signal) are regenerated into the core array; all 1's are generated into the byte location specified by the 'mark' signal.

Operation with LCS (Optional Feature)

The LCS is an optional feature that extends the storage capacity of the system. One 2361 Model 1 (1, 048, 576 bytes), or up to four 2361 Model 2's (2, 097, 152 bytes each) may be installed without interleaving. By installing units in pairs, two 2361 Model 1's or two or four 2361 Model 2's, the addressing can be split so that one unit of a pair contains all even addresses and the other unit contains all odd addresses. When these units are addressed by the BCU on an interleaved basis, the normal 2361 internal cycle time of 8 usec is reduced to an effective cycle time of 4 usec.

Basic Control and Timing Considerations

- Requests are recorded by sync trigger/latches.
- Priority test is performed when BCU is not busy.
- Once priority is granted, BCU decodes storage address.
- 'Select' signal is sent to storage if unit is not busy.

The basic scheme for processing storage requests by the BCU is shown in Figure 2-16. Requests from the CPU and I/O channels are entered into the BCU request sensing logic and are recorded by the corresponding sync trigger/latch circuits of the BCU. If the BCU is not busy processing a previous request, it examines the status of the sync latches to perform a priority test. When priority is granted to a particular unit, the priority trigger for that unit is set, and the storage address from the unit is made available to the BCU for decoding. At the completion of the decoding, the BCU initiates a storage cycle by gating the storage address to the SAB and sending a 'select' signal to the even or odd storage area of the selected storage unit. Upon receipt of the 'select' signal, the storage unit proceeds to read out or write in the data at the addressed location.

On a fetch data request, the storage unit gates one doubleword of data onto the SDBO for sampling by the requesting CPU or channel. On a store data request, the storage unit replaces the contents of the addressed location with the data sent over the SDBI from the CPU

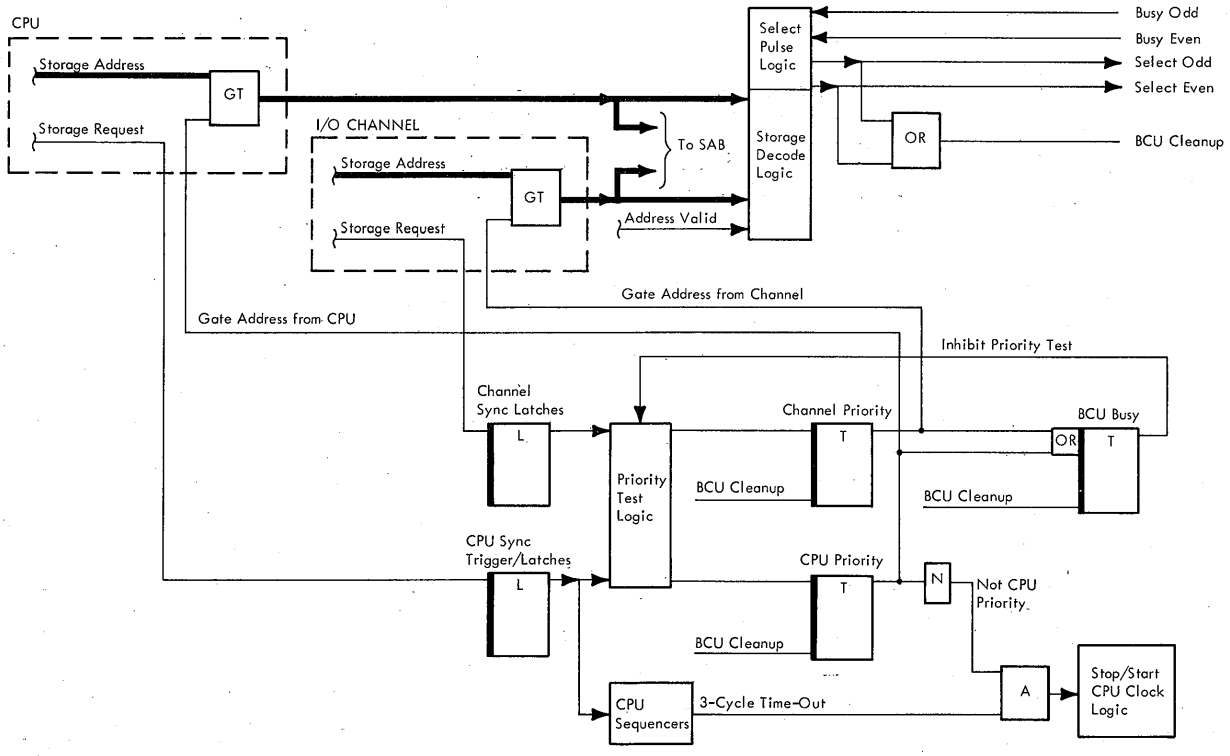


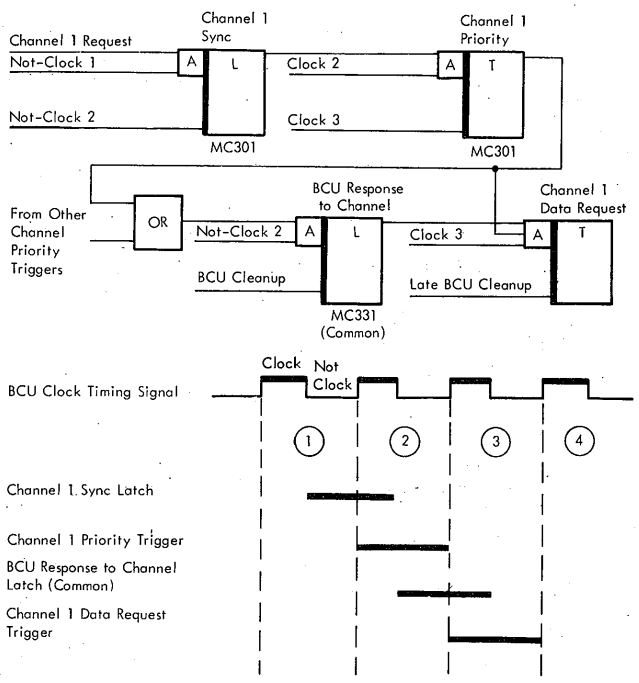
Figure 2-16. Basic BCU Scheme for Processing Storage Requests

or channel. Only those bytes designated by the 'mark' signals are placed into main storage; in the absence of 'marks', bytes already in storage remain unaltered.

During the time when the storage unit is actually performing a fetch or store cycle, it is not available for reselection. To prevent the BCU from doing so, a busy indication is presented to the BCU from either the even or odd storage area, depending on whether the storage unit is performing an even or odd request.

The access time to main storage is 600 ns; i.e., data becomes available from or is stored into main storage 600 ns after the request is issued. The basic timings for the fetch and store operations are shown in Figure 2-17.

To provide the necessary control signals at the correct time, the BCU makes extensive use of trigger/latch circuits. All BCU triggers are set at clock time of the machine timing signal and are reset at the following clock time. Conversely, the latch circuits are set at not-clock time of the timing signal and are reset at the following not-clock time. Thus, BCU timing sequences are implemented (essentially) by sequential shifting of status information through latch-to-trigger-to-latch circuits. A typical arrangement is shown in the adjacent column. Note that the state of a particular trigger or latch at any particular time is indicative of the progress made since the issue of the request:



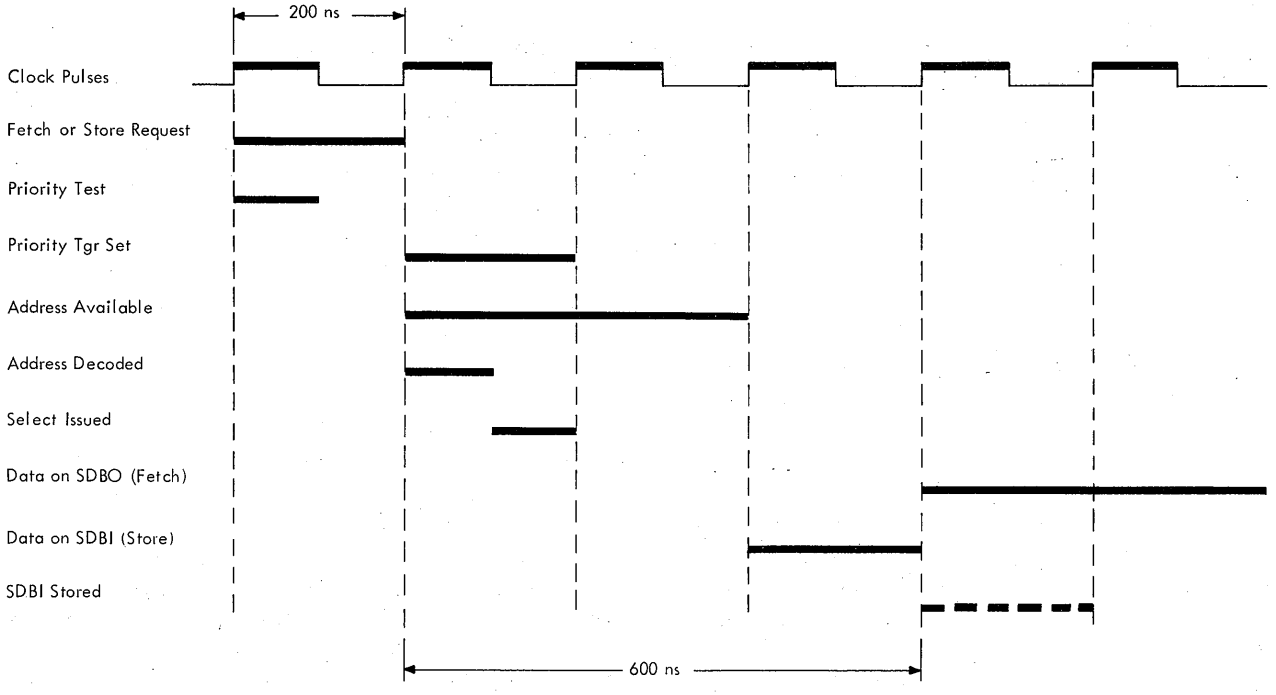
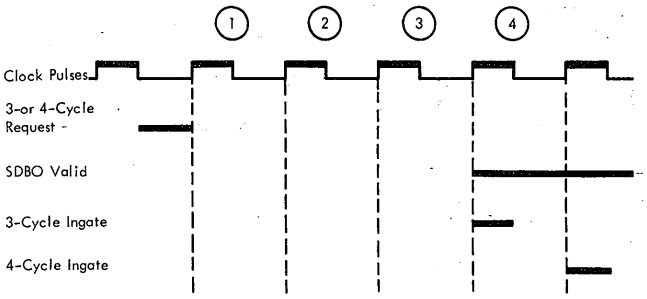


Figure 2-17. Basic Request Timing

As mentioned previously, the CPU can generate either a 3- or 4-cycle fetch request to specify the exact time at which the SDBO is to be gated into the processor. If the CPU is granted priority, and if a successful storage selection has been made, the SDBO becomes "valid" after 3 cycles have elapsed from the time of request. To meet the requirements of a 4-cycle request, the SDBO is held valid for two successive machine cycles. This provision ensures that data will also be present at the SDBO during the 4-cycle sample pulse:



LCS operation is similar to HSS operation, except for the time involved. After the LCS unit is selected, it takes 3 usec for data to return from the selected storage; for about 2 usec of that time the BCU is free to process channel requests. In the case of a CPU fetch request, the CPU clock is stopped until the 'LCS advance' signal restarts it to process the incoming data. In the case of channel requests, the channel waits for the 'LCS advance' signal, but does not hold up other operations.

Basic Operational Sequence

- CPU sequencers count CPU cycles following CPU request.
- Sending of 'select' signal to storage signifies successful completion of request.
- 'Invalid address' signal is generated if 'select' signal is not sent to storage.

The basic BCU operational sequence in handling storage requests from either the CPU or the I/O channels is shown in Figure 2-18. The handling of CPU requests is described first.

The CPU storage requests are issued by one of three functional areas within the processor: I-Fetch logic, ROS microprogram, or scan controls. Once the request is issued, a group of CPU sequencers is activated in the BCU. These sequencers count the CPU cycles that elapse after the request is issued. If the request cannot be completed within a specific time defined by the processor (3 or 4 cycles, depending on the type of request), the sequencers stop the CPU clock to prevent processing in the CPU from advancing beyond the specified cycle. The CPU clock remains stopped until the BCU is able to complete the request.

When the BCU is not busy processing a previous request, a priority test is performed on all the requests (CPU and channel) recorded by the BCU sync latches. After priority is granted to a specific request, the BCU is immediately placed into the busy status to inhibit further priority testing until the accepted request is completed. (During BCU busy status, requests are still received by the BCU and are stored into the latches for consideration on the next priority test.)

If the BCU is not busy, and if no channel requests are pending at the time, the CPU request is granted priority. This action allows the storage address supplied by the CPU to be gated to the SAB. The BCU then decodes the SAB address and generates a 'select' signal to send to the specified storage unit. Whether or not the signal is actually sent depends on the SAB specifying a unit that is within the addressing range of the physical storage and has its power turned on.

If the address supplied by the CPU is outside the physical address range of the installation or specifies a storage unit whose power is down, the 'select' signal is not sent to the storage unit. In this case an 'invalid address' signal is sent to the CPU, and the request is cancelled.

In a case where the unit address is correctly defined and its power turned on, the attempt to issue a 'select' signal will be made. However, the sending of the 'select' signal to the unit may be delayed if the unit is busy servicing a previous request. A busy storage unit causes another selection attempt to be made by the BCU on the following cycle. Continuous selection attempts are thus made until a 'select' signal is sent to storage.

The sending of a 'select' signal to a storage unit signifies to the BCU that a successful selection has taken place. A BCU cleanup operation is then initiated to reset the BCU control circuits and to allow a priority test to be made on pending requests.

Storage requests from the seven I/O channels are handled by the BCU in a sequential order of priority: the highest priority is assigned to channel 1, then to channel 2, then to channel 0 (the multiplexer channel), and then sequentially to channels 3, 4, 5, and 6. When the BCU grants priority to a requesting channel, a BCU response signal is sent to that channel. The channel then gates the storage address to the SAB and responds to the BCU with an 'address valid' signal. Upon receipt of the 'address valid' signal, the BCU decodes the SAB address and attempts to issue a 'select' signal to the specified storage unit. Basically, from this point on, the channel request is handled in the same manner as a CPU request. When the channel is performing a store operation, the 'mark' bus, SDBI, and 'store' signal are gated directly from the channel to the addressed storage unit. If an invalid address is decoded by the BCU during processing of a channel request, an invalid address indication is sent to the channel and to the I/O interrupt logic of the CPU.

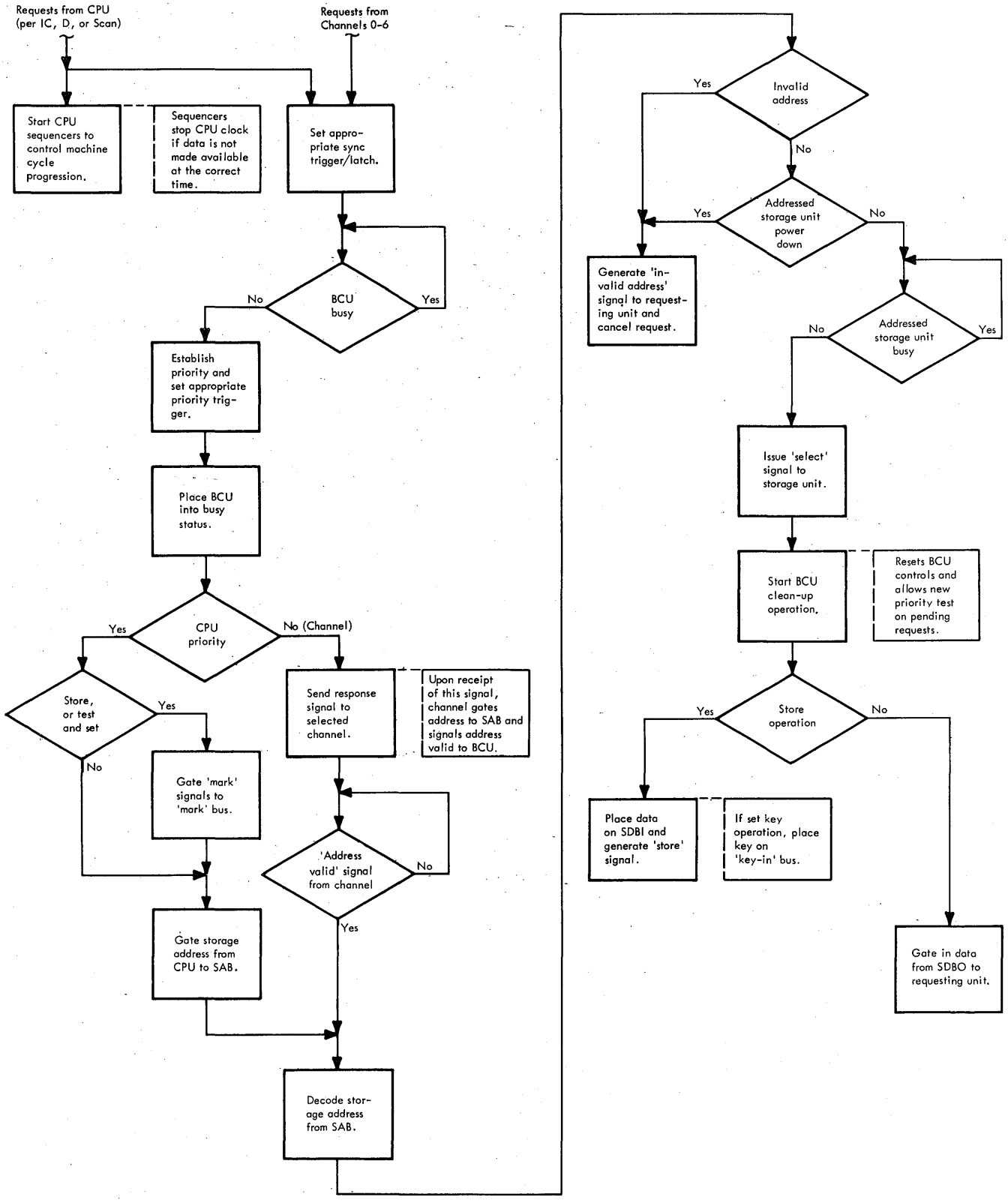


Figure 2-18. Basic BCU Operational Sequence

DETAILED ANALYSIS OF BCU FUNCTIONS

For purposes of discussion, the BCU is divided into a number of functionally distinct logic areas. The subsequent paragraphs describe the functions performed by each area and explain how these functions fit into the overall operational sequence of the BCU (Diagram 4-201, FEMDM).

Initial Handling of Requests

- CPU requests during clock time are recorded by sync triggers.
- CPU requests during not-clock time are recorded by sync latches.
- I/O requests are always recorded by sync latches.

All requests for main storage originate in either the CPU or the I/O channel. The BCU logic used for sensing and recording these requests is shown in Diagram 4-202, FEMDM.

Storage requests from the processor can be issued to the BCU either at clock or not-clock time of the machine cycle. To synchronize the clock and not-clock requests, the BCU employs a trigger/latch sync arrangement. Requests received at clock time are first entered into the BCU sync triggers; they are then propagated (at not-clock time) into the sync latches. Requests received at not-clock time are entered directly into the sync latches. Thus, at the completion of one machine cycle, all requests are reduced to a common time-reference frame.

From the BCU "viewpoint", the storage requests issued by the CPU can be placed into one of three general categories:

1. Requests Generated by Microprogram. These requests are decoded at clock time and, depending on the address source (IC or D), are entered into the corresponding BCU sync triggers. Furthermore, all fetch requests must be specified as being either 3 or 4 cycles in duration. This is to inform the BCU of the specific time at which the requested data must be gated into the processor. The presence or absence of the '3-cycle request' signal from the CPU indicates whether a 3- or 4-cycle fetch has been initiated; i.e., the '3-cycle sync' trigger (in the BCU) is set on all 3-cycle fetch requests and reset on all 4-cycle fetch requests. Upon the setting of the appropriate sync trigger in the BCU, the request is propagated (at not-clock time) into the corresponding sync latch.
2. Requests Generated by I-Fetch Hardware. These requests are decoded at not-clock time and, therefore, are entered directly into the corresponding IC or D latch and the '3-cycle sync' latch.
3. Requests Generated by Scan Hardware. These requests, generated during logout and ROS test operations, are decoded at clock time of the machine cycle. Accordingly, the requests are first entered into the scan-sync trigger and are then propagated into the scan-sync latch at not-clock time. Fetch requests initiated by scan operations are always specified as 4-cycle requests; i.e., the 3-cycle trigger is not set.

At clock time of the machine cycle following the requests, the signals from the sync latches are further propagated into the appropriate request triggers and into the 'CPU request' trigger. The output of the 'CPU request' trigger feeds the priority test logic, which enters the CPU request into priority contention with the channels; it also feeds the CPU sequencer and clock control logic, which stops the CPU clock if the request cannot be handled within the specified time. If the priority test establishes that the CPU request can be handled immediately, the outputs of the IC, D, or scan sync latches and request triggers are used to gate the storage address from the appropriate CPU source (IC, D, or scan address generator) into the BCU.

In conjunction with a request per D, the processor may issue an 'insert key', 'set key', or 'test and set' signal to the BCU. These signals are recorded into the appropriate BCU triggers and are later used to modify the handling of the storage request. Basically, this modification is as follows:

1. The 'insert key' trigger causes an 'insert key' signal to be sent to storage during the handling of the request.
2. The 'set key' trigger issues a 'set key' signal to storage and gates the key bus to storage.
3. The 'test and set' trigger causes a 'test and set' signal to be sent to storage during the handling of the request.

The store-data requests from the CPU are detected by the BCU whenever any mark triggers are set in the processor at the time of the request; a 'store' trigger is then set in the BCU. The output of the 'store' trigger activates a 'store' signal to the selected storage unit and also sets the 'BCU data gate' latch, which gates ST to the SDBI. In addition, the 'mark' signals are transmitted to the selected storage unit via the 'mark' bus. The 'mark' signals specify to the storage unit which bytes on the SDBI are to be entered into the addressed doubleword location.

Storage requests from the I/O channels are made to the BCU by means of channel-request signals. These request signals are detected at not-clock time and are entered into the corresponding BCU sync latches. The sync-latch outputs are in turn applied to the priority test logic to establish whether the particular channel can be granted priority at that time.

The BCU does not distinguish between fetch or store requests from the I/O channels: both types of requests are handled by a common BCU response sequence. When a store-data operation is performed by the channel, the 'mark' bus, SDBI, and 'store' signal are transmitted directly from the channel to the storage unit selected by the BCU.

A channel request to a busy LCS unit energizes a channel lockout circuit. This circuit allows selection of lower priority units while waiting for the completion of the LCS operation. The 'LCS busy' signal forces a priority test to reset the channel priority trigger. The 'channel X waiting' trigger is set, and the channel sync latch is deconditioned by the channel lockout (Diagram 4-202). The channel request remains pending until the 'LCS precomplete' signal resets the 'channel X waiting' trigger, at which time the channel sync latch is set and the channel request is processed.

Establishing Priority

- Priority test sets appropriate priority trigger.
- Once priority is established, BCU is placed in busy status.

A priority test is performed whenever the BCU finishes processing a storage request; it is repeated on each subsequent cycle until the BCU accepts a request and attempts to process it. The BCU is then placed in a busy status, inhibiting further priority tests until that request is completed. When the request is completed, the BCU is again placed in a not-busy state (by a BCU clean-up operation), and a priority test is forced at the beginning of the next machine cycle. The next request is then accepted from the sync latches, or, if no requests are present in the sync latches, a priority test takes place on each following machine cycle until a request is received.

Priority for a waiting storage request is established by transferring the particular request from its sync latch to its associated priority trigger. Setting a priority trigger causes the proper storage address (IC, D, or scan in the case of a CPU request; an I/O channel in the case of a channel request) to be gated to the SAB. On CPU storage requests, processing continues under control of the CPU sequencers, while the BCU selects the addressed storage unit. On channel requests, setting a priority trigger sets a response latch, which in turn initiates the controlling request/response sequence between the BCU and the I/O channel.

The priority triggers for the CPU and I/O channels (Diagram 4-203, FEMDM) are implemented so that there is a definite sequence in which they can be set on successive priority-test operations: the setting of a priority trigger is inhibited if a higher-order priority trigger is about to be set. Thus, only one priority trigger can be set at a given time, with channel 1 having highest priority and the CPU rating lowest priority. Note also that when a priority test is made all requests received up to the point of the test are considered, including requests for the unit just serviced. For example, if two requests are present at the time a priority test is made, the request having highest priority is serviced, and a priority test is again initiated. If, however, another request for the unit just serviced has been received during the interim, that unit is again serviced, and the lower-priority request is kept waiting.

Gating the Address to SAB

- Address gating is initiated by set priority trigger.

When priority is established for either the CPU or the I/O channel, the set state of the corresponding priority trigger initiates gating of the storage address into the BCU. The address gating logic is illustrated in Diagram 4-204, FEMDM.

When the 'CPU priority' trigger is set, gating of the storage address is under control of the IC, D, or scan sync latches. If the storage address is in the process of being transferred into IC or D at the time the CPU has been granted priority, the PAL gate trigger permits a direct transfer of the address from PAL to SAB.

When an I/O channel has been granted priority, the corresponding priority trigger is set for that channel. A 'BCU response' signal is then sent to the channel to initiate the gating of the storage address to the SAB. An 'address valid' signal is then returned to the BCU from that channel to indicate that the address is on the SAB and to start the storage sequence. A channel-data-request trigger for that channel is set on the next cycle, signalling the requesting channel to place data to be stored on the SDBI.

Stopping the CPU Clock

- CPU sequencers control distribution of clock timing within CPU during handling of CPU requests.
- CPU sequencers are started at clock time of cycle following request.
- Sequence in which sequencers are stepped varies with request being processed:
  1. 4-cycle fetch: 'CPU 2' trigger, 'CPU 2' latch, 'CPU 3' trigger, 'CPU 3' latch, 'CPU 4' trigger, 'CPU 4' latch, 'CPU 5' trigger, 'CPU 5' latch.
  2. 3-cycle fetch: 'CPU 2' trigger, 'CPU 2' latch, 'CPU 3' trigger, 'CPU 4' latch, 'CPU 5' trigger, 'CPU 5' latch.

3. Store or set key: 'CPU 2' trigger, 'CPU 2' latch, 'CPU 3' trigger.
4. Insert key: 'CPU 5' trigger, 'CPU 5' latch.

- Conditions that stop CPU clock:
  1. 'CPU 2' latch is set, and either CPU has not received priority or storage is busy.
  2. 'CPU 4' latch is set and CPU did not receive 'advance' signal from storage. (This function is disabled if LCS feature is not installed.)
  3. Insert-key operation.

A group of CPU sequencers (four trigger/latch combinations) is used in the BCU to control CPU cycle progression after each CPU storage request. These sequencers control the 'stop CPU clock' trigger in the BCU. The 'stop CPU clock' trigger has direct control of the CPU clock: setting this trigger stops the CPU (on the following cycle); resetting this trigger starts the CPU (on the following cycle). Diagram 4-205, FEMDM, shows the CPU sequencers and the control logic for the 'stop CPU clock' trigger. The sequencers are started on the CPU cycle following a storage request and are advanced by the subsequent CPU clock signals: the 'CPU 2' trigger and latch are set during the first CPU cycle following the request; the 'CPU 3' trigger and latch are set during the second CPU cycle following the request; and so on.

The 'stop CPU clock' trigger is so implemented that if its reset logic is active its set logic is prevented from setting the trigger. Because of this method of implementation (as seen in Diagram 4-205), if the 'CPU 2' latch is set (first cycle following a storage request), the 'stop CPU clock' trigger will be set on the next cycle (second cycle following the request), and the CPU clock will be inhibited from performing the third processing cycle. This clock stopping sequence occurs during both fetch and store operations, retaining the storage address in the IC, D, or scan controls until the BCU and the storage unit become available. When the BCU and storage unit become available, and when the CPU is awarded priority, the 'stop CPU clock' trigger is reset by the BCU 'cleanup' signal. Once the CPU clock is started, both CPU processing and further sequencer stepping is continued. (If LCS units are attached to the Model 65, the CPU clock is stopped again at latch-4 time provided the advance signal from storage is not received.)

The 'CPU 5' trigger and 'CPU 5' latch are held in the set condition when the 'LCS request' signal sets the 'stop CPU clock' trigger. The CPU is inhibited for about 3 usec until the 'LCS advance' signal resets the 'stop CPU clock' trigger (Figure 2-19). The 'LCS pre-advance' signal prepares to restart the CPU. First, the 'advance waiting' trigger is set to prevent honoring another request by diverting 'issue a select' signals to the 'storage 1' trigger. The 'storage 1' latch retains the 'select' signal until the 'LCS advance' signal resets the 'advance waiting' trigger, thus allowing the 'select' signal to set the 'storage 2' trigger. The 'LCS advance' signal also resets the 'stop CPU clock' trigger if the 'CPU 5' latch is set. The clock is restarted, data arrives on the SDBO, and requests are honored in the normal manner again.

For store data operations, the CPU sequencers are started in the normal manner. However, the output of the 'store' latch modifies the subsequent sequencer stepping as follows: 'CPU 2' trigger and 'CPU 2' latch are set on the first cycle following the request; 'CPU 3' trigger is set on the second cycle following the request. Further sequencer advance is inhibited during store operations because ingating is not required.

Selecting the Storage Unit

- Storage address must correspond to physical storage unit.
- Storage capacity is defined by pluggable jumper cards in CPU.

The capacity of main storage (also referred to as the high-speed storage, or HSS) varies with the model:

Model	Capacity	Description
G65	131,072 bytes	1 storage unit without interleaving capability
H65	262,144 bytes	1 storage unit with interleaving capability
I65	524,288 bytes	2 storage units with interleaving capability
IH65	786,432 bytes	3 storage units with interleaving capability
J65	1,048,576 bytes	4 storage units with interleaving capability

If the system is equipped with more than one storage unit (Models I65, IH65, or J65), the starting address for each unit varies in increments of 262,144 bytes; i.e., addresses 0–262,143 are assigned to unit 1, 262,144–524,287 to unit 2, and so on. Thus, when a storage address is supplied to the BCU, the BCU must select the physical storage unit referred to by this address. Furthermore, if interleaving is desired, the BCU must establish whether the address pertains to the odd or the even area of the selected unit. Once the physical storage unit and the odd or even area within it have been established, the BCU issues an 'odd-' or 'even-select' signal to that unit. This signal commands the storage unit to sample the address on the SAB.

Two identical address decoder circuits are provided in the BCU: one circuit decodes the addresses supplied by the CPU, and the other circuit decodes the addresses

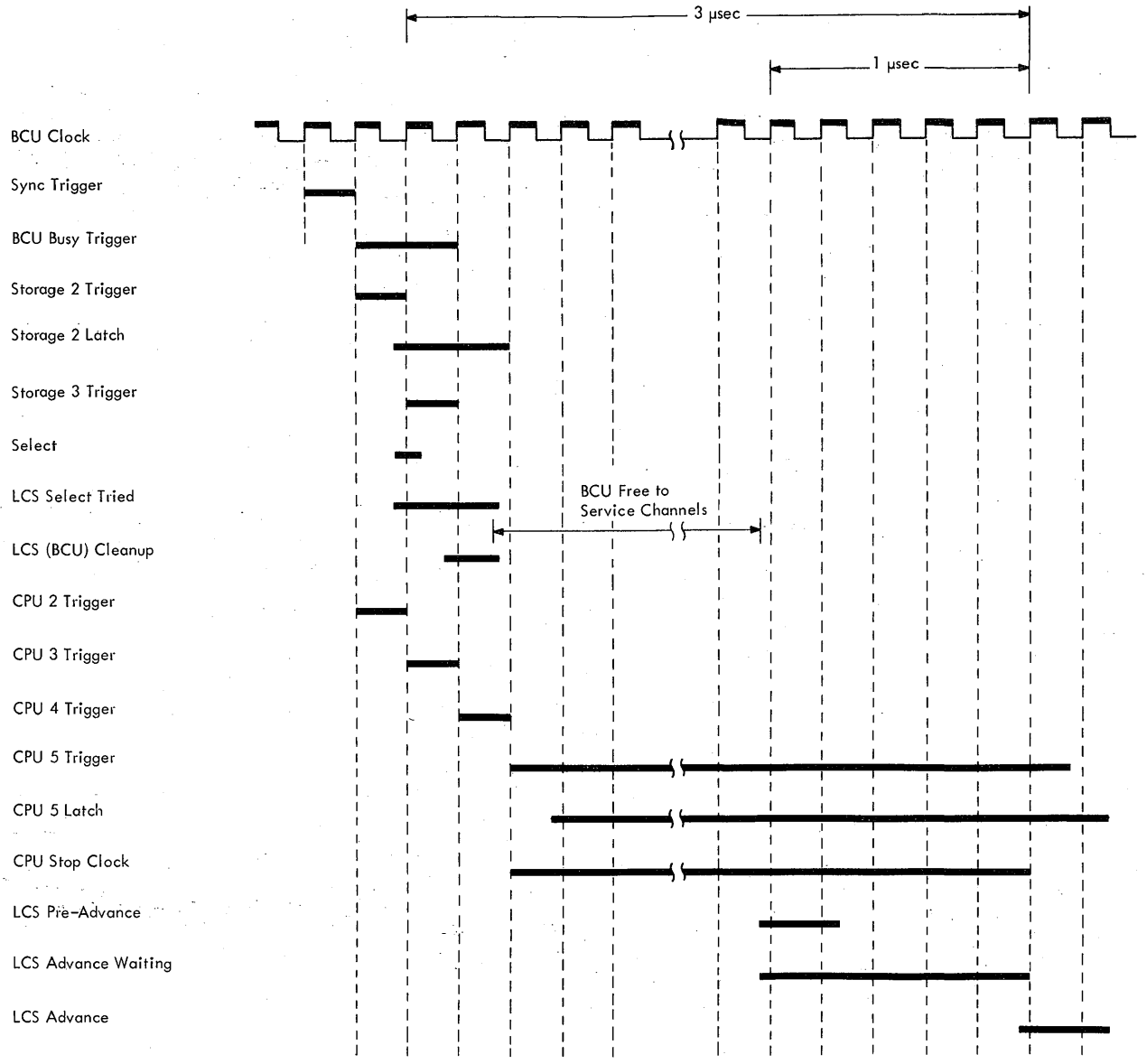


Figure 2-19. Typical Timing for CPU Fetch Request to LCS

supplied by the I/O channels. The need for two decoders (Figure 2-20) is due to a time lag between the CPU- and channel-supplied addresses. Note that a CPU-supplied address bypasses the driver/receivers of the multiplex bus and is applied directly to the address decoder; a channel-supplied address must pass through the channel drivers and through the BCU receivers before being applied to the address decoder. The time lag introduced by the driver/receivers in the multiplex bus is approximately 55 ns.

The optional LCS unit(s) can increase the storage capacity by from 1,048,576 bytes to 8,388,608 bytes. When one or more LCS units are used, it is the function of the BCU to establish whether the particular storage address refers to the HSS or the LCS and to select the correct unit within each type. To perform this function, separate HSS and LCS decoding logic is used in the BCU. Figure 2-21 shows the basic scheme used for selection of the correct storage unit.

At the time the system is set up, pluggable jumper cards are set and inserted into the CPU to define the capacity of the HSS and the LCS and to customize the address decoding logic to the available storage.

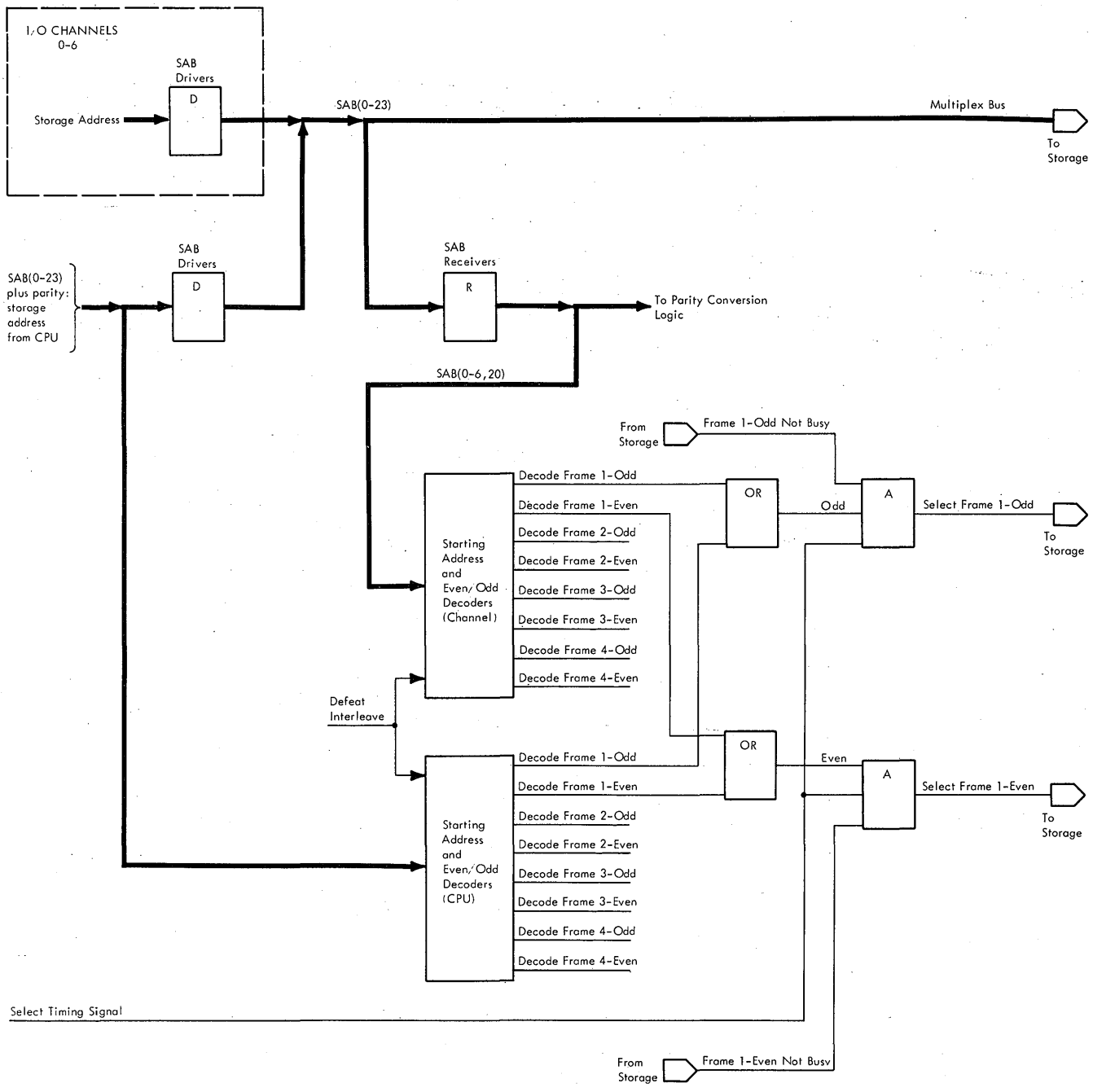
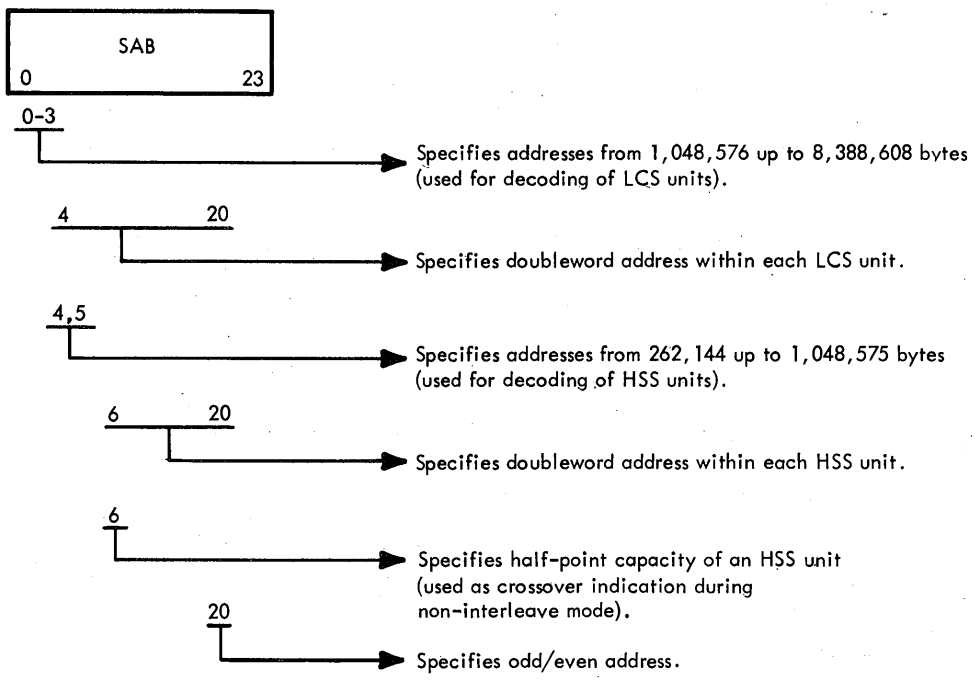


Figure 2-20. Gating of Storage Address From CPU and Channels to Address Decoders in BCU

Each HSS unit of 262,144 bytes is divided into two 131,072-byte areas: even and odd. In normal processing operations, HSS accesses are interleaved between the two storage areas, permitting a higher data transfer rate than is possible without interleaving. Selection of the even or odd storage area within the HSS unit is determined by the setting of the DEFEAT INTERLEAVING switch on the system control panel and SAB(6) or SAB(20).

LCS units may also be interleaved to reduce access time. When interleaving is used, the first LCS unit is accessed (about 3 usec) and the system is released although the LCS unit must be unavailable an additional 5 usec while it is completing its cycle. However, a different LCS unit may be addressed immediately after the 'LCS advance' signal. Because the access time is less than half the cycle time of the LCS unit, the effective access rate is reduced to 4 usec in interleaved mode.

The address decode logic used for decoding CPU- and channel-supplied addresses is shown in Diagram 4-206, FEMDM. When the DEFEAT INTERLEAVING switch is



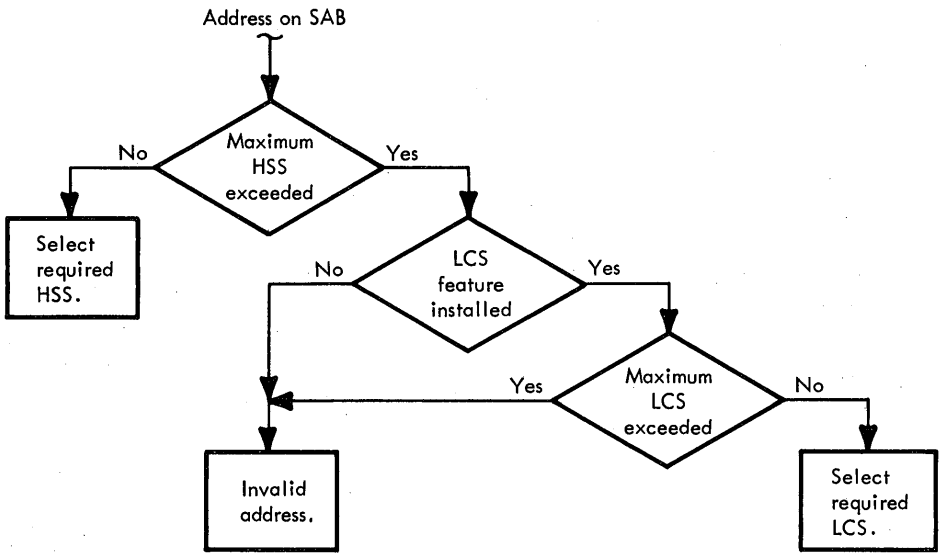


Figure 2-21. Selection of Correct Storage Unit

set to the normal or PROC position, SAB(20) determines the even or odd storage area: if SAB(20) = 0, then even; if SAB(20) = 1, then odd. When the switch is set to the NO REV position (defeat interleaving and no reversal of the even and odd storage areas), SAB(6) determines the even or odd storage area: if SAB(6) = 0, then even; if SAB(6) = 1, then odd. However, the function of SAB(6) is reversed if the switch is set to the REV position (defeat interleaving with reversal of storage addresses). In this case, SAB(6) = 0 specifies an odd storage area; SAB(6) = 1, an even storage area.

The LCS addresses are decoded in 1,048,576-byte groups, the first of which is contiguous with the last HSS address. If the available HSS capacity is exceeded, the 'select' signals to HSS units are blocked, and the LCS byte group is determined by SAB(0–3) and an above/below detector. The above/below detector is conditioned by pluggable jumper cards defining the HSS byte limit and, depending on the HSS byte limit, by SAB(4) or SAB(4,5).

The following examples show this operation. Assume a

524,288-byte HSS capacity and a 2,097,152-byte LCS capacity. Assume, also, an address of 1,179,648 is on the SAB [SAB(3) and SAB(6) = 1]. The HSS 'select' signals are blocked because the HSS has been exceeded and the 'below' signal is generated. The 'below' signal results from the 524,288-byte HSS limit signal being active (pluggable jumper card) and SAB(4) = 0. The LCS byte group gating finds SAB(0–3) = 0001 and the 'below' signal present; the only LCS byte group requiring these conditions is the first. Therefore, the address 1,179,648 is in the first 1,048,576 LCS byte group. (The address boundaries for this first group are 524,288 and 1,572,863.)

Converting SAB Parity

- Generate parity bit P(A) for SAB(6–12).
- Generate parity bit P(B) for SAB(13–19).

The SAB-parity-conversion logic subtracts bits 0–5 and bits 20–23 from SAB(0–23). On the basis of this subtraction, two new parity bits are generated: P(A) for SAB(6–12), and P(B) for SAB(13–19). In generating these new parity bits, the parity-conversion logic must take into account the setting of the DEFEAT INTERLEAVING switch on the system control panel. If this switch is not in the PROC position, it will cause the storage unit to reverse SAB(6) and SAB(20). Accordingly, the same bit reversal must be performed by the parity-conversion logic in generating P(A) and P(B) parity bits.

The basic decoding performed in the BCU and in the storage unit to select a specific core location is shown in Figure 2-22. Note that in the storage unit, only SAB(6 or 20) and SAB(7–19) are gated into the odd or even memory address register (MAR); i.e., the address of any doubleword location in the odd or even area of the storage unit is specified by 14 address bits.

During operation in the interleave mode, SAB(6–19) remains constant for two storage cycles, and two consecutive storage accesses are made per the same address: once from the even and once from the odd area of the storage unit. The selection of the odd/even area is made in the BCU by decoding SAB(20); this bit changes once for each storage access.

During operation in the defeat-interleave mode, SAB(6) and SAB(20) are reversed: the BCU now decodes SAB(6) to select the odd/even storage area; and the storage unit gates SAB(20) as the highest-order bit in the address of a core location.

From a system standpoint, SAB(6) designates the one-half capacity of a storage unit; or, the total capacity of an odd- or even-area in the storage unit. Accordingly, the BCU continues to issue 'select' signals to the *same* even, or the *same* odd, storage area until all core locations in that area are accessed; i.e., SAB(6) changes from 1 to 0, or vice versa.

From a system standpoint, SAB(20) designates the doubleword boundary and is updated for each storage request. Because the storage unit substitutes SAB(20) for SAB(6), the highest-order address bit (within the selected odd/even area) is changed on each storage cycle. This changing leads to interleaved accesses which are restricted *solely* to the odd or to even area of the storage unit; i.e., storage accesses alternate between the upper and lower halves of the area.

The SAB-parity-conversion logic is shown in Figure 2-23 and in Diagram 4-207, FEMDM. This logic utilizes Exclusive-OR circuits to generate, subtract, and add SAB parity bits until the required P(A) and P(B) results are obtained. Note that the output of an Exclusive-OR always excludes those bits which are applied simultaneously on *both* circuits inputs. Thus, depending on the input bits, an Exclusive-OR can be used as an adder or as a subtractor. When the parity bit of SAB(0–3) is combined with SAB P(0–3,6 or 20,7) at an Exclusive-OR, the result is SAB P(6 or 20,7) and SAB P(0–3) has been canceled or subtracted. SAB P(6 or 20,7) is then combined with SAB P(8–12) to produce the parity of SAB(6 or 20,7–12) by addition of parity. In this manner, the parity of SAB(0–23) is converted to parity of SAB(6 or 20,7–19) and then sent to the storage unit.

Generating 'Select' Signal to Storage

When priority is granted to either the CPU or the channel, the BCU attempts to issue a 'select' signal to storage (Diagram 4-208, FEMDM). The manner in which the 'select' signal is generated depends on whether the LCS feature is installed in the system.

Generating 'Select' Signal if LCS Is Not in System

- 'Select' signal is initiated by 'CPU priority' signal for CPU requests.
- 'Select' signal is initiated by 'address valid' signal for channel requests.

The 'select' signal logic is activated by the 'CPU priority' signal or, in the case of channel requests, by the 'address valid' signal. If no previous selection attempt has been made ('storage 2' latch not set), detection of the 'CPU priority' or 'address valid' signal results in the setting of the 'storage 2' trigger. The 'storage 2' trigger sets the 'storage 2' latch to gate a 135-ns 'select' timing signal to the final gating logic for HSS units 1 through 4. This 135-ns timing signal is sent to storage if the unit power is on, if a valid unit address is decoded, and if the odd/even area of the unit is not busy servicing a previous request.

The absence of storage power or the failure to decode a valid storage address results in an invalid address condition. (Refer to "Detection of Invalid Address.") However, if the 'select' signal is not sent due to a busy storage

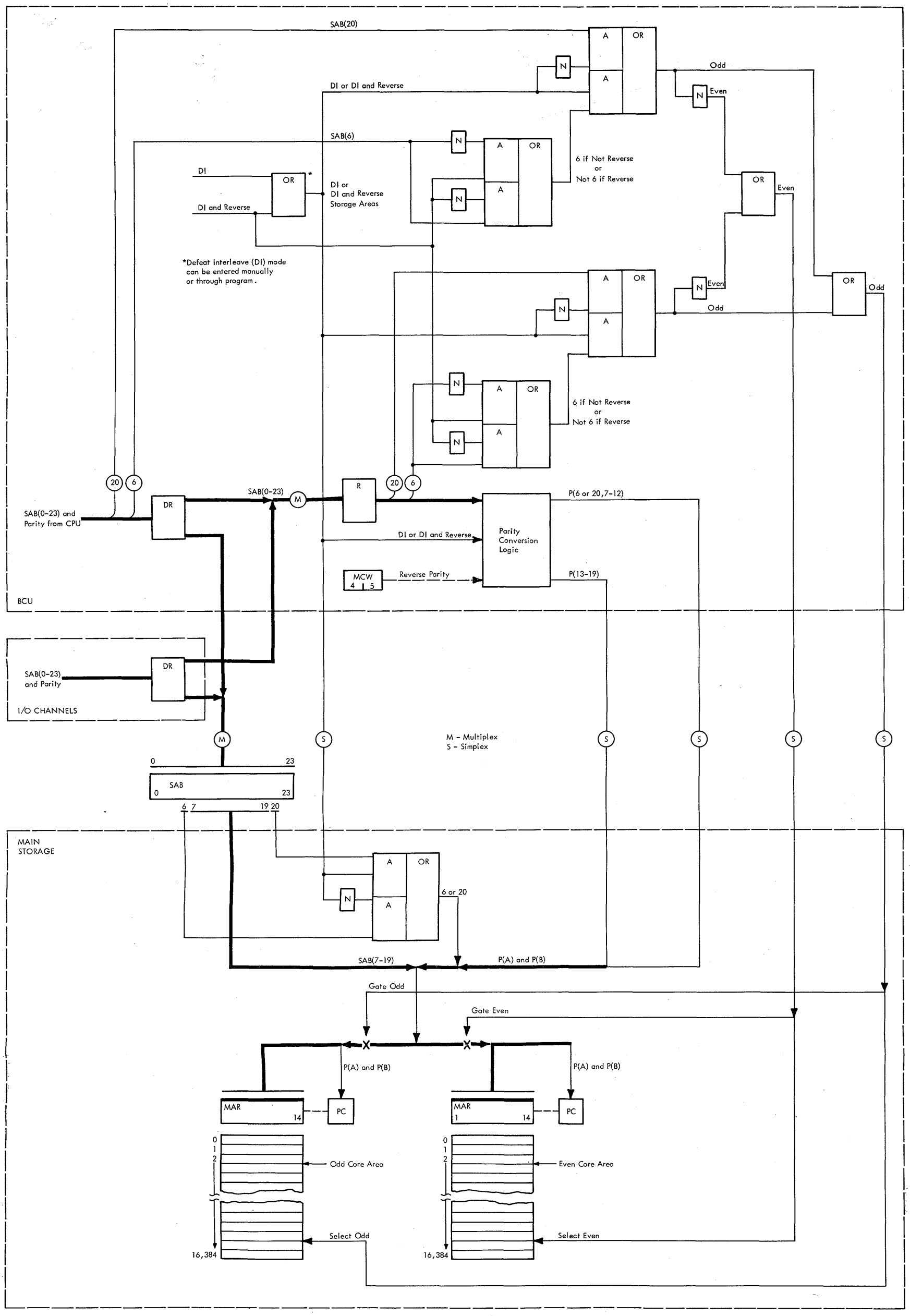


Figure 2-22. Basic SAB Decoding Circuits in BCU and HSS Unit

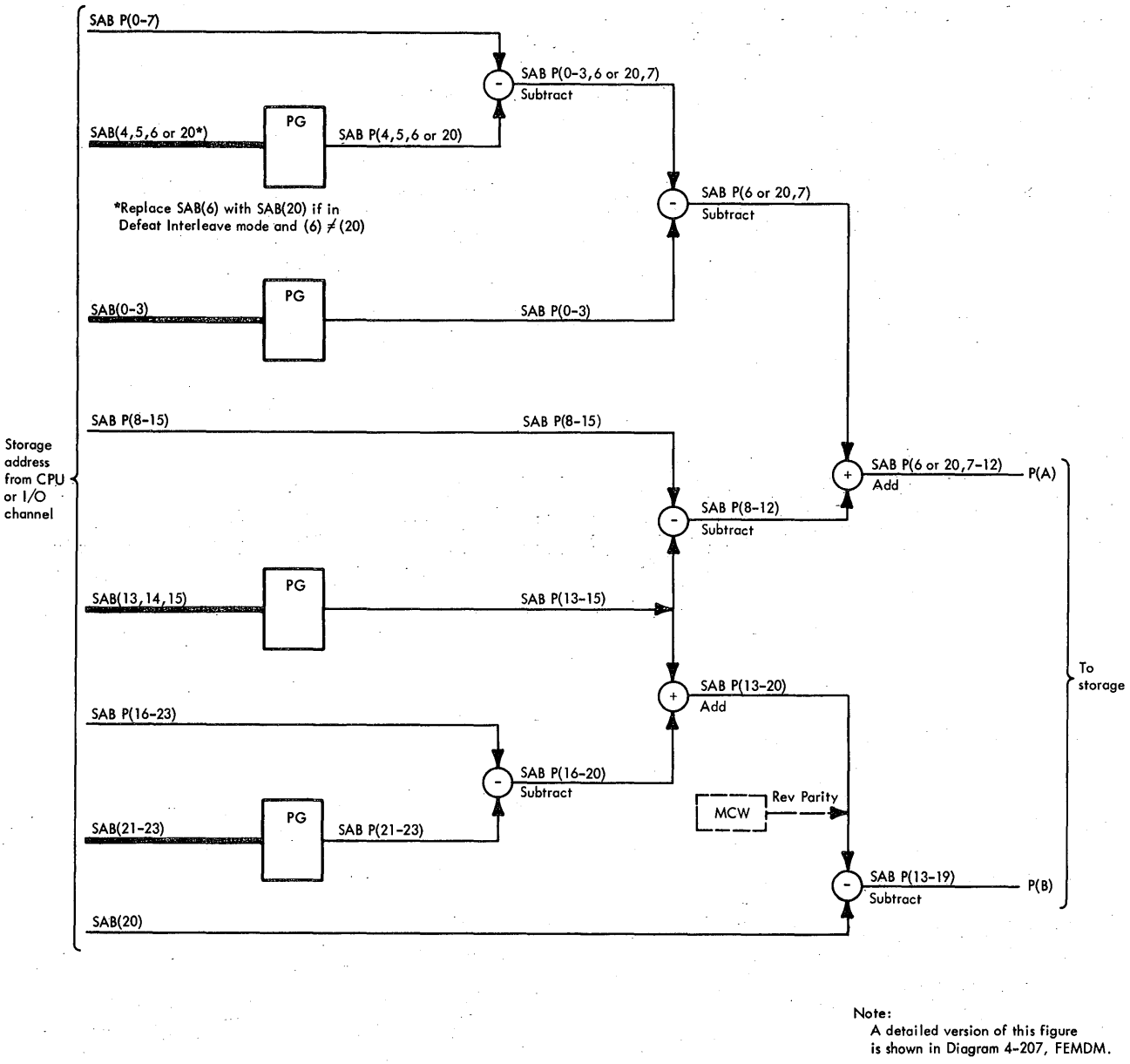


Figure 2-23. SAB Parity Conversion Logic

condition, the 'HSS select sent' trigger is not set. The reset state of this trigger and the set state of the 'storage 2' latch initiates another selection attempt. Successive attempts are thus made until the storage unit becomes not busy and a 'select' signal is sent to that unit. The sending of the 'select' signal to storage initiates a BCU cleanup operation which restarts the CPU clock and resets the BCU control circuits for handling of the next request.

Generating 'Select' Signal if LCS Is in System

- 'Storage 2' trigger is set only when storage not busy.

Detection of the 'CPU priority' or of the 'address valid' signal by the select logic results in setting of either 'storage 1' or 'storage 2' trigger, depending on the availability status of the storage data bus. The bus availability status is specified by the state of the 'advance waiting' trigger: if this trigger is reset (no storage advance), the bus is available; if set, unavailable.

If the storage bus is available, the 'storage 2' trigger is set; however, if the storage bus is not available, the 'storage 1' trigger is set. Note that once the 'storage 1' trigger is set, it forms a loop with the 'storage 1' latch to continue trying to set the 'storage 2' trigger. When the 'advance' signal is received from the storage using the data bus, the 'advance waiting' trigger is reset, allowing the 'storage 2' trigger to be set.

The set state of the 'storage 2' trigger results in sending the 'select' signal to the HSS or to the LCS. The conditions for LCS selection are similar to those described for HSS. However, one major difference should be noted: the 'storage 3' trigger maintains the 'storage 2' latch set for two successive cycles to compensate for a 1-cycle delay in starting the invalid address test. This delay is due to the need of sampling the state of the 'advance waiting' trigger during LCS selection.

Detection of Invalid Address

- Address is outside physical storage capacity.
- Power is down on storage unit.

An invalid address condition is detected in the BCU if power is off in the addressed storage unit or if the address supplied by the requesting unit exceeds the physical storage capacity of the system. There are two invalid address detection circuits: one to detect an invalid address from the CPU, and another to detect an invalid address from the channel. The 'invalid address' signal has a dual function: to detect an invalid address and to select an address decoder. For a channel request, an 'invalid address' signal disables the CPU decoder; for a CPU request, a similar signal disables the channel decoder. The invalid address logic is shown in Diagram 4-209, FEMDM.

The BCU detects an invalid address by examining the states of the 'test for invalid address' trigger and the 'select tried' latch: if both are set, the address is valid; if the 'test for invalid address' trigger is set but the 'select tried' latch is reset, the address is invalid. The subsequent paragraphs analyze the conditions that lead to detection of an invalid address and describe the BCU operation that follows.

The 'test for invalid address' trigger is set when a 'select' signal is issued to the final decode-gating logic of the BCU. The output of this trigger then monitors whether the 'select' signal is successfully passed by the gating logic. If the address of a storage unit has been defined correctly, and if its power is turned on, the 'select' signal is passed through the gating logic to set the 'select tried' latch.

The set state of the 'select tried' latch signifies that a successful storage selection has been made; i.e., a 'select' signal will be issued to storage even though the sending of the signal may be delayed if the unit is busy servicing a previous request. Thus, this condition indicates that a valid storage address has been decoded.

If the 'select' signal fails to pass through the final decode-gating logic, the 'select tried' latch is not set. This condition indicates to the BCU that a successful storage selection is not possible due to an incorrect address on the bus or a power failure in the storage unit.

Upon detection of an invalid address, the BCU alerts the requesting CPU or channel of the error condition and then proceeds to force a 'select' signal to storage. (Detection of an invalid address during I-Fetch does not necessarily result in a program interruption.)

The forced storage selection is made to generate a 'BCU cleanup' signal, which will allow the CPU to continue (CPU clock was stopped) or the channel to continue. The BCU forces a request to any storage unit that has power, in the ascending order: starting with unit 1 and, if unit 1 power is off, proceeding to units 2, 3, and 4. The storage unit accessed by a forced 'select' signal, however, is not allowed to complete its cycle because the BCU also issues a 'cancel' signal to that unit. The 'cancel' signal prevents storage data transfer to the SDBO; thus 0's are transferred to the selected register in the channel or the CPU.

Recording of Error Indications from Storage

- Storage errors are address check, data check, protection check.

If a storage request causes an error within the selected storage unit, an error indication is sent from that unit to the BCU. The storage-error-recording logic in the BCU is shown in Diagram 4-210, FEMDM. Note that the PROC CHK indicator is activated only if a CPU request is responsible for the storage error; the STOR CHK indicator is activated if *either* a processor or channel request has caused the error. (If a storage error is received when the CPU CHECK switch is set to the PROC position and the machine check mask bit (bit 13) is a 1, it causes a logout operation.)

The storage-error indicators (Stor Adr Chk and Stor Data Chk) are activated as a result of one of the following error conditions:

1. Storage Address Check. A parity error has been detected on either SAB or the 'mark' bus. (On store or set-key operations, the original data is regenerated into the addressed storage location to prevent loss of data.)
2. Storage Data Check. A parity error has been detected on the SDBI, the SDBO, or the storage-key bus.

In addition to the address-check and data-check error conditions, the storage unit also generates a protection check if the key supplied by the request either is not zero or does not match the protection key assigned to the storage area being addressed. (Upon detection of this condition, the storage unit cancels the request to prevent loss of data.) A protection-check indication from storage is routed via the BCU and applied directly to the CPU I-Fetch-and-interrupt logic.

Resetting of BCU Logic

- Sending of 'select' signal to storage initiates the following reset signals: 'channel accept', 'BCU cleanup', 'late BCU cleanup', 'late BCU cleanup for CPU request'.

The BCU-reset logic is shown in Diagram 4-211, FEMDM. After the BCU sends a 'select' signal to storage, this logic generates the following reset signals: 'channel accept', 'BCU cleanup', 'late BCU cleanup', and 'late BCU cleanup for CPU request'.

The reset signals are initiated as a result of the 'pseudo accept' signal; i.e., a signal which simulates an "accept" condition from storage. The start of the 'pseudo accept' signal is controlled by routing the 'select timing signal' signal through a number of inverters in the BCU which provide a total delay of approximately 80 ns. The delayed 'select timing signal' signal is then AND'ed with the 'select sent' signal to set the 'accept' latch. The output of the 'accept' latch initiates the 'channel accept' and 'BCU cleanup' signals.

The reason for delaying the 'BCU cleanup' signal (approximately 80 ns after the 'select timing signal' signal) is to prevent a new priority test from taking place on the same cycle in which the 'select' signal is issued to storage. (This condition could result in two successive 'select' signals being issued less than 400 ns apart; i.e., an early 'BCU cleanup' signal would reset the 'BCU busy' trigger before the 'BCU busy' latch had time to latch on; the resulting priority test could initiate another 'select' signal on the following BCU cycle.)

The time duration for which the 'accept' latch stays set must be such as to "extend" the 'BCU cleanup' signal past B2 time of the following clock cycle. This condition is necessary to provide the following functions:

1. Inhibit setting of the 'stop CPU clock' trigger, if the set state of the 'CPU 2' latch coincides with the issue of the 'select' signal. (The 'stop CPU clock' trigger is so implemented that if its reset logic is active, its set logic is prevented from setting the trigger; see Diagram 4-205, FEMDM.)
2. Inhibit setting of the 'test for invalid address' trigger; i.e., when a successful 'select' signal is generated, the 'BCU cleanup' signal blocks the invalid address test (Diagram 4-209, FEMDM).

The 'late BCU cleanup' and 'late BCU cleanup for CPU request' signals are generated approximately 130 ns after the 'accept' latch is set. These signals clear all BCU functional circuits associated with the previous request to allow servicing of a new request on the following cycle: the 'late BCU cleanup' signal resets all channel-priority and data-request triggers; the 'late BCU cleanup for CPU request' signal resets the 'CPU priority' trigger and removes the CPU address from the multiplex bus.

Resetting of BCU logic on an LCS operation is delayed one cycle (200 ns) because the 'accept' signal from the storage (LCS) is used instead of the 'pseudo accept' signal. This delay allows time to test the busy condition of the LCS unit. Figure 2-24 shows the logic to develop the 'LCS cleanup' signal.

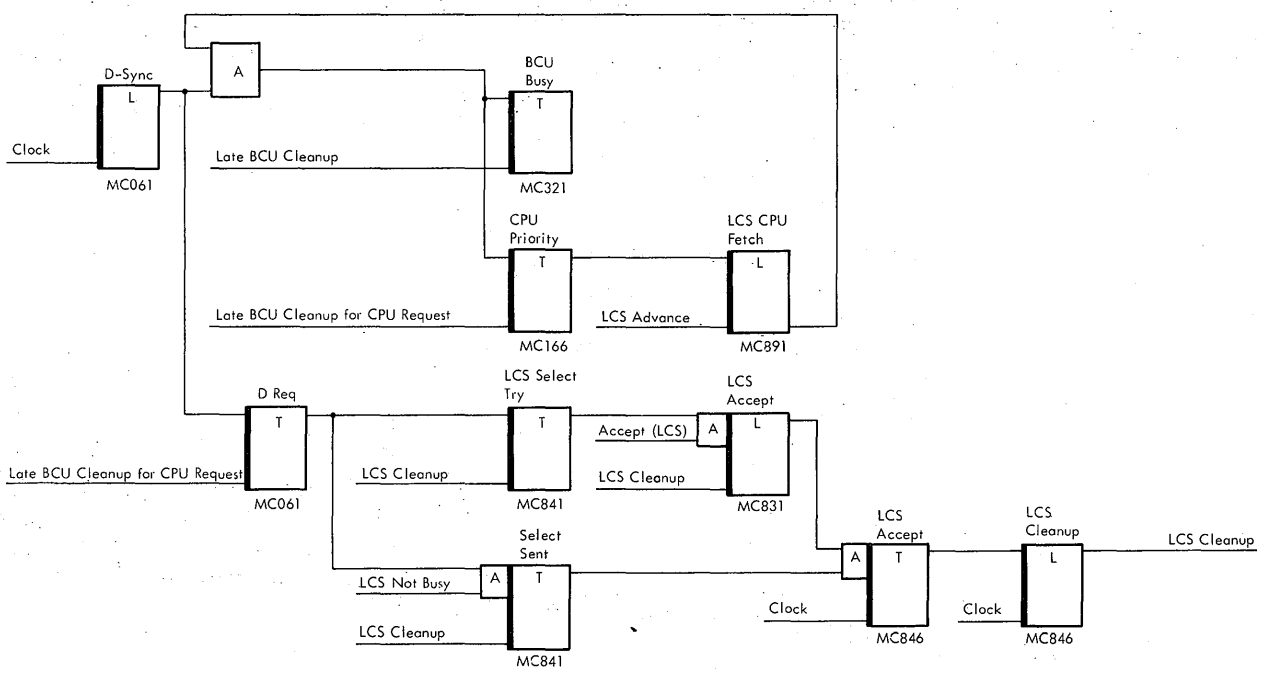


Figure 2-24. BCU Reset for LCS Operation

DETAILED ANALYSIS OF BCU OPERATIONS

The subsequent paragraphs describe the operational sequences performed in the BCU during processing of storage requests for the CPU or channel.

CPU Storage Requests

The functional sequence for CPU storage requests is illustrated in Diagram 4-201, FEMDM.

CPU storage requests are issued to the BCU from I-Fetch, ROS, and Scan logic, and both the storage request and the resulting data transfer are overlapped with processing. Although all CPU requests are handled by the BCU as basic fetch or store requests, the following variations exist: 3- and 4-cycle fetch, store, insert-key, set-key, test-and-set, and single-cycle operation. Processing of the request is interrupted when either the storage unit or the BCU is busy, or when the requested storage data is not available in the specified time.

3- and 4-Cycle Fetch Operations

- Ingating of requested storage data is specified at 3 or 4 cycles following storage request.

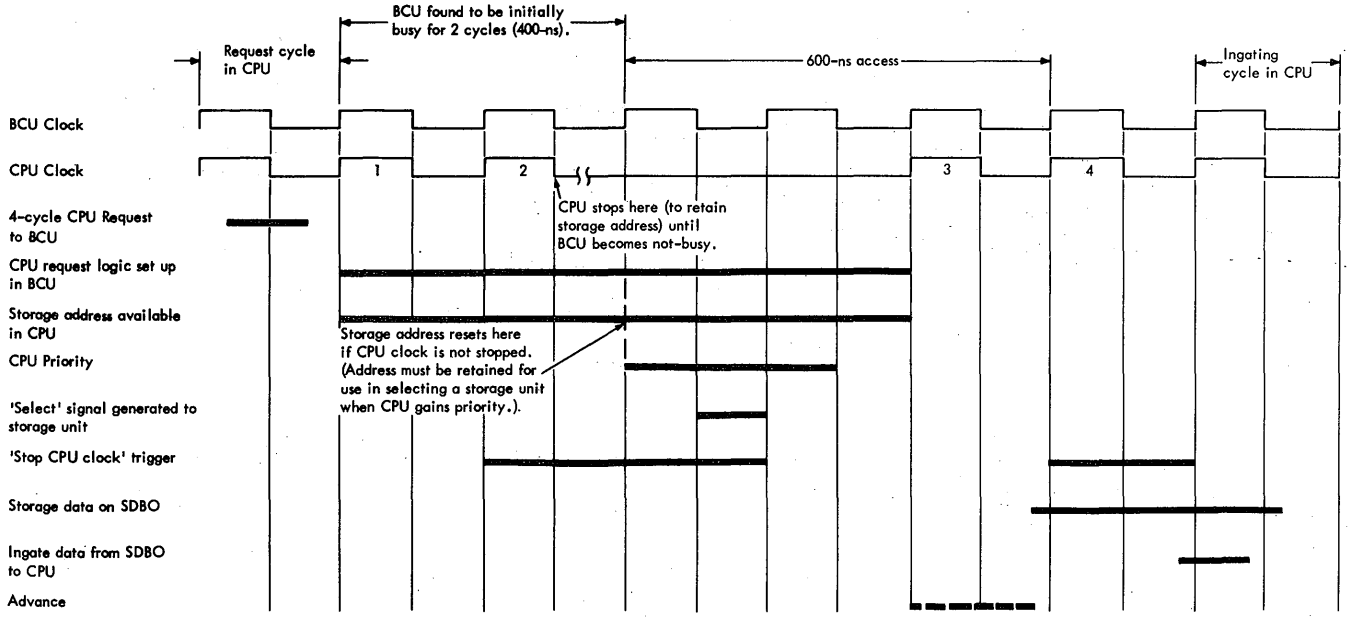
Because the BCU operates at the same machine cycle speed as the CPU, and because the access time to storage requires 3 cycles, the CPU is allowed to continue processing for 3 cycles following the request. The BCU must keep track of the CPU cycle progression so that the SDBO ingating is executed at the correct time. If data is not present at the SDBO at the end of 3 cycles, or if the SDBO is to be ingated following the 4th cycle, the CPU clock is stopped. The requested data may not be present on the SDBO for one of the following reasons:

1. A BCU-busy condition, wherein priority cannot be established immediately because of channel interference.
2. A storage-busy condition, wherein the addressed storage unit is still servicing a previous request.
3. Storage access time limitations, wherein a request has been made to an LCS unit (optional feature).

Figure 2-25 illustrates a typical clock control timing sequence for a 4-cycle fetch (with the BCU initially busy for 2 cycles). For detailed timing charts of 3- and 4-cycle fetch operations, refer to ALD's M8221, M8241, and M8251.

Store Operation

Store-data requests are always made per D. Data from ST is gated to the SDBI for transfer to the storage unit. Mark trigger settings are transferred to the storage unit (via the 'mark' bus) to specify which of the eight bytes of data are to be stored. Processing of the store request is interrupted



Notes:
1. 4-cycle fetch to 750-ns storage unit (600-ns access) with BCU initially busy for 2 cycles (400-ns).
2. CPU clock is shown stopped at the end of the 2nd cycle following the request cycle in CPU.
   Reason for stopping: to retain storage address for use in selecting a storage unit when BCU is found to be busy (store or fetch operations).

Figure 2-25. Typical CPU Clock Stopping Sequence

if the BCU or the storage unit is busy. For detailed timing of the store operation, refer to ALD M8211. Note that the 'store' signal prepares the storage unit for storing data.

Insert-Key Operation

- 'D-storage request' and 'insert key' signals are sent from ROS to BCU.
- Basic fetch operation is performed by BCU.
- Five-bit (plus parity) storage protection key is transferred from storage to CPU via 'key out' bus.

Insert-key operations are essentially fetch requests per D, in which a five-bit (plus parity) storage protection key is obtained from the specified storage protection area of main storage and inserted into F(0–4) of the CPU. These operations enable the CPU to examine the key patterns used by the storage protection mechanism.

An insert-key request sets the 'insert key' trigger in the BCU to modify the normal stepping of the CPU sequencers: the 'CPU 5' trigger is set on the first cycle following the insert-key request, and further sequencer stepping is not performed. The 'stop CPU clock' trigger is also set on the first cycle following the insert-key request. Thus, the CPU clock is stopped on the second cycle after the request.

When the BCU awards priority to the CPU, the contents of D are gated to the SAB. A 'select' signal is then generated and sent to the addressed storage unit together with an 'insert key' signal. In the storage unit, the two signals ('select' and 'insert key') initiate an insert-key operation. The seven high-order bits of the SAB are decoded to determine the protection key location, and the key pattern (five bits plus parity) is fetched from that location. The storage unit then generates an 'advance key' signal, which prepares the BCU for ingating of the 'key out' bus into the F-register of the CPU. A detailed timing chart for the insert-key operation is shown on ALD M8211.

Set-Key Operation

- 'D-storage request' and 'set key' signals are generated from ROS to BCU.
- Basic store operation is performed by BCU.
- Five-bit (plus parity) storage key is transferred from CPU to storage via 'key in' bus.

Set-key operations are essentially store requests per D in which a five-bit (plus parity) storage protection key is obtained from F(0–4) in the CPU and stored into the specified storage protection area of main storage. These operations enable the CPU to set new key patterns into the storage protection mechanism.

A set-key request sets the 'set key' trigger in the BCU. In addition, all CPU mark triggers are set during the set-key operation. Receipt of the D-storage request by the BCU then sets the 'store' trigger and starts the CPU sequencers in the normal manner. When the CPU is awarded priority, the contents of D are gated to the SAB, and the 'select' and 'set key' signals are sent to the addressed storage unit. (The sending of the 'store' signal to the storage unit is inhibited during the set-key operation.)

The 'select' and 'set key' signals initiate a set-key operation in the storage unit. The seven high-order bits of the SAB are decoded to determine the protection key location, and the 'key in' bus is gated into that location. For a detailed timing chart of the set-key operation, see ALD M8211.

Test-and-Set Operation

Although a test-and-set request combines aspects of both fetch and store operations, the basic handling of the request by the BCU is similar to a 3-cycle fetch per D. The major difference is that the BCU sends one mark and a 'test and set' signal to the storage unit specified by the D-address. The basic test-and-set timing sequence is as shown on ALD M8221.

Single-Cycle Operation

- START pushbutton provides for manual stepping through CPU cycles.
- RATE switch in SINGLE CYCLE STORAGE INHIBIT provides for manual stepping through all cycles within request sequence. (Storage unit is not selected; data transfer is inhibited.)
- RATE switch in SINGLE CYCLE enables CPU to run automatically from time 'select' signal is sent to storage until data transfer operation is completed.

When CPU operations are being tested in the single-cycle mode, the CPU clock is stepped manually; one CPU clock cycle results for each depression of START. The BCU clock, however, is not affected by the single-cycle mode and runs automatically, thus allowing the BCU to continue servicing storage requests from the CPU or from the I/O channels.

The single-cycle operation can be performed by the CPU with or without access to storage: if the RATE switch is placed in SINGLE CYCLE, the CPU must access storage whenever it steps through a cycle specifying a storage request; if the RATE switch is in SINGLE CYCLE STORAGE INHIBIT, all storage requests are ignored by the BCU.

When servicing storage requests from the CPU in the single-cycle mode, the BCU must ensure that the gating of data to or from the CPU is synchronized with the storage

unit operation. To accomplish this function, special single-cycle logic in the BCU controls the CPU clock and runs it automatically whenever synchronized ingating is required. Refer to Diagram 4-212, FEMDM, a functional flowchart of BCU operations during servicing of CPU requests in the single-cycle mode.

To enable manual stepping through as many CPU cycles as possible, the BCU delays sending the 'select' signal to storage until the time when the CPU sequencers stop the CPU clock. (This delay is implemented by blocking CPU priority, when in the single-cycle mode, until the CPU clock is stopped.) The stop-clock condition indicates to the BCU that the data transfer between CPU and storage must be executed on the next depression of START.

At this point, the nature of the request is of primary consideration. If a fetch-data request is in progress, the BCU must override the single-cycle controls and run the CPU clock automatically until the CPU executes the ROS word with the 'ingate SDBO' micro-order. If a store-data request is in progress, the BCU need not control the CPU clock because the CPU data is placed on the SDBI when CPU priority is established and a 'select' signal is sent to storage; i.e., as soon as the START pushbutton is depressed. Thus, on store-type requests, the operator can single-cycle through every ROS word of the CPU microprogram; on fetch-type requests, the operation automatically skips over one or two ROS words, depending on whether a 3- or 4-cycle request is specified. (Note that the time slice between two consecutive depressions of START does not enter into consideration; this time slice is much greater than the 600-ns time interval required to access storage.)

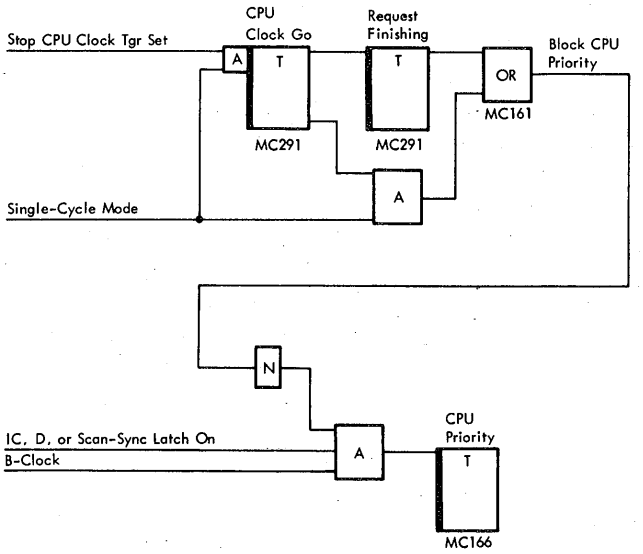
When START is depressed and a 'select' signal is sent to storage, the resulting 'BCU cleanup' signal restarts the CPU clock. The state of the CPU sequencers, after the first CPU clock signal is generated, indicates the type of request in progress and whether additional stepping of the CPU clock is required; this stepping is performed automatically under control of the 'CPU clock go' trigger. Note that the 'CPU clock go' trigger is always reset on the first clock signal after the 'CPU 5' latch is set.

If a store-data request is in progress, the 'CPU 5' latch is set before the CPU clock is restarted. Thus, as soon as the CPU clock is restarted, the 'CPU clock go' trigger is reset to indicate that no additional CPU clock cycles are required to complete the request.

If a 3-cycle fetch-data request is in progress, the 'CPU 4' latch is set before the CPU clock is restarted. When the CPU clock is restarted, the first clock signal sets the 'CPU 5' trigger/latch sequencers, and accesses the ROS word with the 'ingate SDBO' micro-order. Note, however, that the ingating of the SDBO into the CPU takes place on the following cycle. Thus, the CPU clock must be automatically stepped an additional cycle to perform the ingating. This function is performed by the 'CPU clock go' trigger, which is reset by the same clock signal that gates in the data into the CPU.

If a 4-cycle fetch-data request is in progress, the 'CPU 3' latch is set before the CPU clock is restarted. In this case, the 'CPU clock go' trigger is not reset until two cycles after clock-restart. Thus, the CPU clock is automatically stepped through two additional cycles to perform the required ingating.

As mentioned previously, sending the 'select' signal to storage is delayed when servicing requests in the single-cycle mode. This delay is accomplished as shown in the following illustration:



Note that the 'block CPU priority' signal is inactive when the CPU is not in the single-cycle mode. In this case, the 'CPU priority' trigger is set on the cycle following the setting of the appropriate IC, D, or scan-sync latch.

When servicing requests in the single-cycle mode, CPU priority is initially blocked by the reset state of the 'CPU clock go' trigger. When the BCU is ready to issue a 'select' signal, however, the 'CPU clock go' trigger is set and remains set until the request is completed. During the final stage of request servicing, the 'CPU priority' trigger is reset by the 'BCU cleanup' signal. The 'request finishing' trigger prevents immediate setting of the 'CPU priority' trigger if another sequential CPU request (in single-cycle mode) has been entered into the sync latch. After the current request has been serviced, the 'CPU clock go' and 'request finishing' triggers are both reset. The 'CPU clock go' trigger then blocks CPU priority until the pending request stops the CPU clock.

Channel Storage Requests

- Each channel storage request transfers 64 data (plus 8 parity) bits between channel and storage (via SDBI/SDBO).

- Data transfer is performed by asynchronous request/response sequence.

I/O channel data-transfer operations are initiated as a result of the CPU program selecting an I/O device and loading I/O control words into the associated channel. When data transfer is required, a series of requests is made by the channel to the BCU. After each request is accepted by the BCU, the proper storage unit is selected, and data is transferred either from storage to the channel (via the SDBO) or from the channel to storage (via the SDBI) on a 64-data-bit (plus 8 parity bits) basis. Thus, the function of the BCU is to accept channel requests, establish channel priority, decode the storage address supplied by the channel, select the proper storage unit, and execute the data transfer.

When priority is established for a particular channel, a 'BCU response' signal causes that channel to gate the storage address to the SAB. The storage address is then decoded by the BCU, and, if the addressed storage unit is available, a 'select' signal is generated and sent to the storage unit. If a store-data operation is required by the requesting channel, a 'store' signal is generated by that channel and is combined with the 'select' signal in the addressed storage unit to produce a store operation.

When the 'select' signal is sent from the BCU to the addressed storage unit, a 'channel accept' signal is sent to the requesting channel, verifying that the storage unit has been successfully selected. If a 'channel accept' signal does not follow the 'data request' signal, the requesting channel assumes that the selection attempt was unsuccessful. Detailed timing charts for channel requests are shown on ALD M8281.

www.IBM360.info

This section describes the registers employed for CPU data flow and control functions. For the overall register data flow, see Diagram 3-2, FEMDM.

Q-REGISTER

The Q-register is a doubleword (64 data bits plus 8 parity bits) buffer for instructions entering the CPU from main storage on the SDBO (Figure 2-26). Data is transferred to the local storage address latches (LAL), the parallel adder, or the R-register.

Input

Instructions are transferred from the SDBO into Q by means of a gate signal decoded from the ROSDR latches. The transfer of data is initiated by either the I-Fetch hardware or by the ROSDR latches at not-clock time; the transfer controls remain active for one cycle (200 ns). The instructions are transferred into Q at clock time.

Op-Code Transfer

- Only halfwords containing op codes are transferred to R.
- Selection of halfword containing op code is determined by IC(21,22).

Only those halfwords in Q containing op codes are transferred to R. Because RX, RS, and SI instructions are composed of two halfwords and SS instructions are composed of three halfwords (only the first of which contains the op code), it is necessary to select the proper halfword to be transferred to R. Note that because RR instructions are composed of only *one* halfword, the next halfword to be transferred to R after an RR instruction is completed is the next sequential halfword in Q.

Selection of the halfword for transfer to R is determined by IC(21,22), and transfer is performed either directly through hardware or as a result of an I-Fetch micro-order. Recall that during an I-Fetch operation the op code of the next instruction to be executed is transferred from R to E, with R then being refilled with the next sequential op code. Because the op code of the next instruction to be executed is always in R, its format (positions 0 and 1) can be predecoded to determine the number of halfwords that compose that instruction and thus indicate which of the four Q-register halfwords contains the next sequential instruction op code. This predecoding occurs at end-op time of each instruction; the result (Q halfword number) is set into IC(21,22), which in turn selects a subsequent I-Fetch ROS word that specifies the next op-code halfword to be transferred to R.

Note: IC(21,22) is not used in addressing main storage, but only specifies which of the four Q-register halfwords is to be transferred to R by the following I-Fetch sequence. The IC(21,22) values associated with each Q-register halfword are illustrated in Figure 2-27.

An exception to the normal I-Fetch ROS word method of transferring the Q-resister halfwords to R is as follows. Assume a condition whereby a four-byte (RX, RS, or SI format) instruction occupies the right half of Q, IC(21,22) = 10, and a storage request is generated to main storage. When the I-Fetch sequence loads the op code of this four-byte instruction into R, the predecode logic determines that the next doubleword being accessed from main storage contains, in its leftmost byte, the op code of the next sequential instruction. [IC(21,22) is also stepped to 00 during this particular I-Fetch.] When, during the following I-Fetch sequence, the contents of R are transferred to E for execution, R is not refilled from Q in the normal manner because the particular I-Fetch ROS word selected to control this operation does not contain a micro-order specifying refilling of R. When that doubleword being brought from main storage enters Q, however, Q(0–15) (the op-code halfword) is allowed to proceed directly on into R. Thus, R again contains the op code of the next instruction to be executed, even though the instruction was not present during the I-Fetch sequence. This function is accomplished solely through the use of hardware that constantly tests for the presence of two signals: 'I-Fetch latch 1 and 3 set' and 'IC(21,22) = 00'.

B-Field and D-Field Transfer

The instruction B-field, which specifies LS registers, and the D-field, which is the main storage address displacement, is transferred from Q to LAL and the parallel adder, respectively. To save time, this information is transferred directly from Q instead of from R or E, thus allowing LS and the address to be available before the execution time of the associated instruction. Transferring these fields must be performed selectively so that the information is associated with the correct instruction.

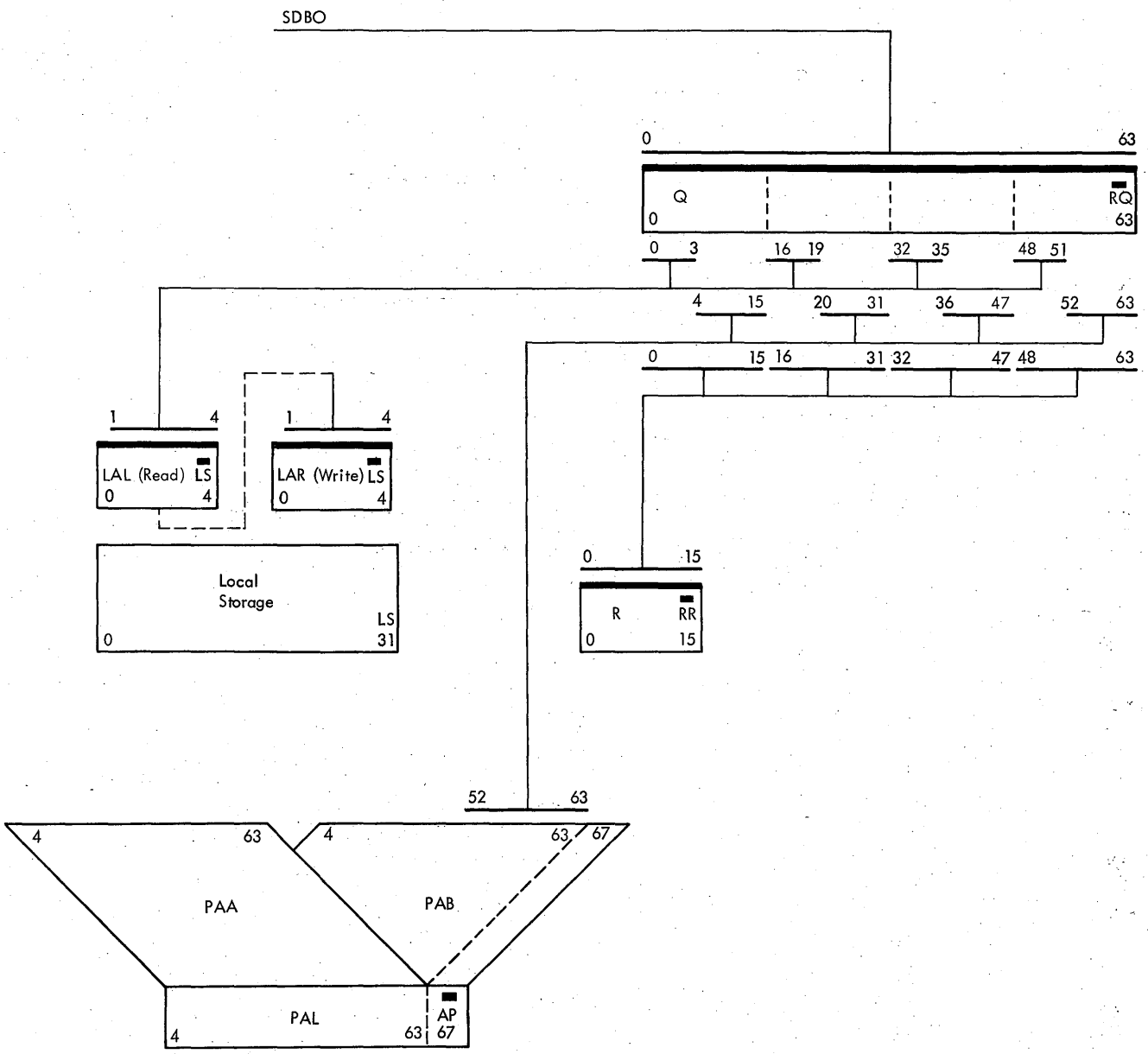


Figure 2-26. Q-Register Data Flow

B-Field Transfer

The four-bit Q-fields (B-field address data) are normally transferred to LAL at end-op time, under hardware control, per IC(21,22), or, for certain branch instructions, per D(21,22). (See Diagram 4-301, FEMDM.)

For SS instructions, however, two B-field values must be transferred to LAL. The first B-field is transferred to LAL at end-op time, per IC(21,22), under hardware control and in the normal manner. [D(21,22) is used for branch instructions.] The B-field of the second operand is then transferred to LAL (from the selected portion of Q) during I-Fetch of the SS instruction by a micro-order contained within one of the I-Fetch ROS words.

All transfer of data from Q to LAL takes place at not-clock time; the data is transferred into LS at clock time.

Note: Because an RR instruction can be contained within R and E, only halfword transfers from Q to R are required for RR instructions. All addresses for LAL can therefore be transferred directly from R or E.

D-Field Transfer

Selection of the Q-register D-field for transfer to the parallel adder (for use in address computation) is determined by the particular ROS word selected for use. The D-field transfer occurs at clock time.

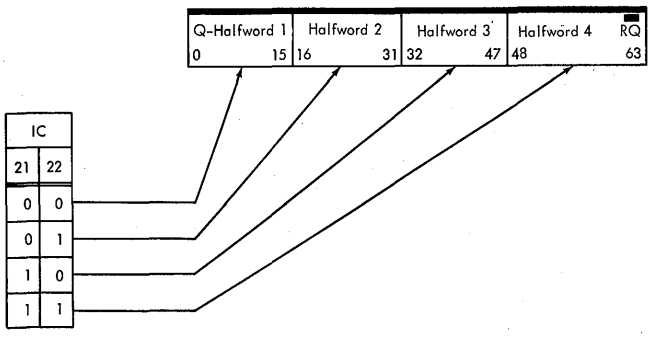


Figure 2-27. Q-Register Halfword Transfer Per IC(21, 22)

R-REGISTER

The R-register is a halfword (16 data bits plus 2 parity bits) register that provides intermediate buffering between Q and E for the halfword that contains the op code (Figure 2-28). This intermediate buffering speeds the refill of Q by allowing a storage request when the last op code has been transferred from Q but has not yet been executed.

Input

The R-register is loaded with one of the four halfwords in Q at I-Fetch time under ROS control. The contents of PAL(56–63) are also transferred to R (at I-Fetch time of the subject instruction of an Execute instruction).

Output

Whenever the instruction in R is predecoded as an RR non-branch instruction, R(8–11) is transferred to LAL at end-op time (Diagram 4-302, FEMDM). (The RR format indicates that R contains the entire instruction.)

Whenever the instruction in R is predecoded as an RR branch instruction, R(12–15) is transferred to LAL at end-op time. The contents of R are transferred to E at I-Fetch time under ROS control.

Predecoding

The R-register predecode logic samples R(0,1) at end-op time to determine the format of the next instruction. Time is saved because prefetching of operands per the format prepares data for use after the instruction is transferred to E. In addition, R(0,1) and IC(21,22) determine the need for storage requests to refill Q.

R(0–4) is tested for shift instructions. Because shifting does not require a storage request, time is saved if a shift instruction is decoded when a Q-refill request is generated 2 cycles before end op. The Q-register refill exceptional condition is eliminated because there is no interference between the shift instruction and the Q-refill storage request.

R(0–7) is sampled for branch instructions so that prefetching of the new instruction address can start immediately, thus saving time. R(12–15) is sampled for a zero condition which prevents the branch in the RR format.

On a branch end op, the instruction halfword is still in the process of being requested from storage. To save time in prefetching operands, the instruction format is predecoded from the SDBO rather than waiting until the instruction becomes stable in R.

E-REGISTER

The E-register is a halfword (16 data bits plus 2 parity bits) register which contains the op-code halfword of the instruction being executed (Figure 2-29).

Input

An op-code halfword (including two parity bits) is transferred from R to E during a normal I-fetch sequence under ROS control. On shift operations, D(18–21) is transferred to E(12–15) via the E-register incrementer. The data path from PAL(56–63) to E(8–15) is used in some SS format instructions to control the specified number of bytes.

Output

Op-code signals to control processing are decoded directly from E(0–7) without the use of gating logic.

LS is addressed by transferring E(8–11) or E(12–15) to LAL(1–4). E(8,11,12,15) is examined for an LS address specification error. When manual operations are performed using LS, E(11,12–15) is transferred to LAL(0,1–4) so that all registers may be addressed. Transferring E(8–11) to PAB(56–59), E(12–15) to PAB(60–63), and E(8–15) to PAB(56–63) provides for multiply and divide operand aligning, byte count control for SS format operation, and subsequent transfer to other registers. E(8–11), E(12–15), or E(8–15) may be sent to the E-register incrementer for alteration under ROS control.

E(8–15) is sent to an external device to provide external-control information during direct control operations. The data is transferred when the 'timing gate' trigger is set. E(8–15) is also sent to the PSW interruption code [S(24–31)] on a supervisor call interruption only, and to control triggers such as 'disable interleaving' and 'diagnose FLT'.

Incrementers

Two four-position incrementer registers are available, with ROS controls, for either treating E(8–11) and E(12–15) separately or treating E(8–15) as an entity. Positions E(8–11) and E(12–15) can be either incremented or

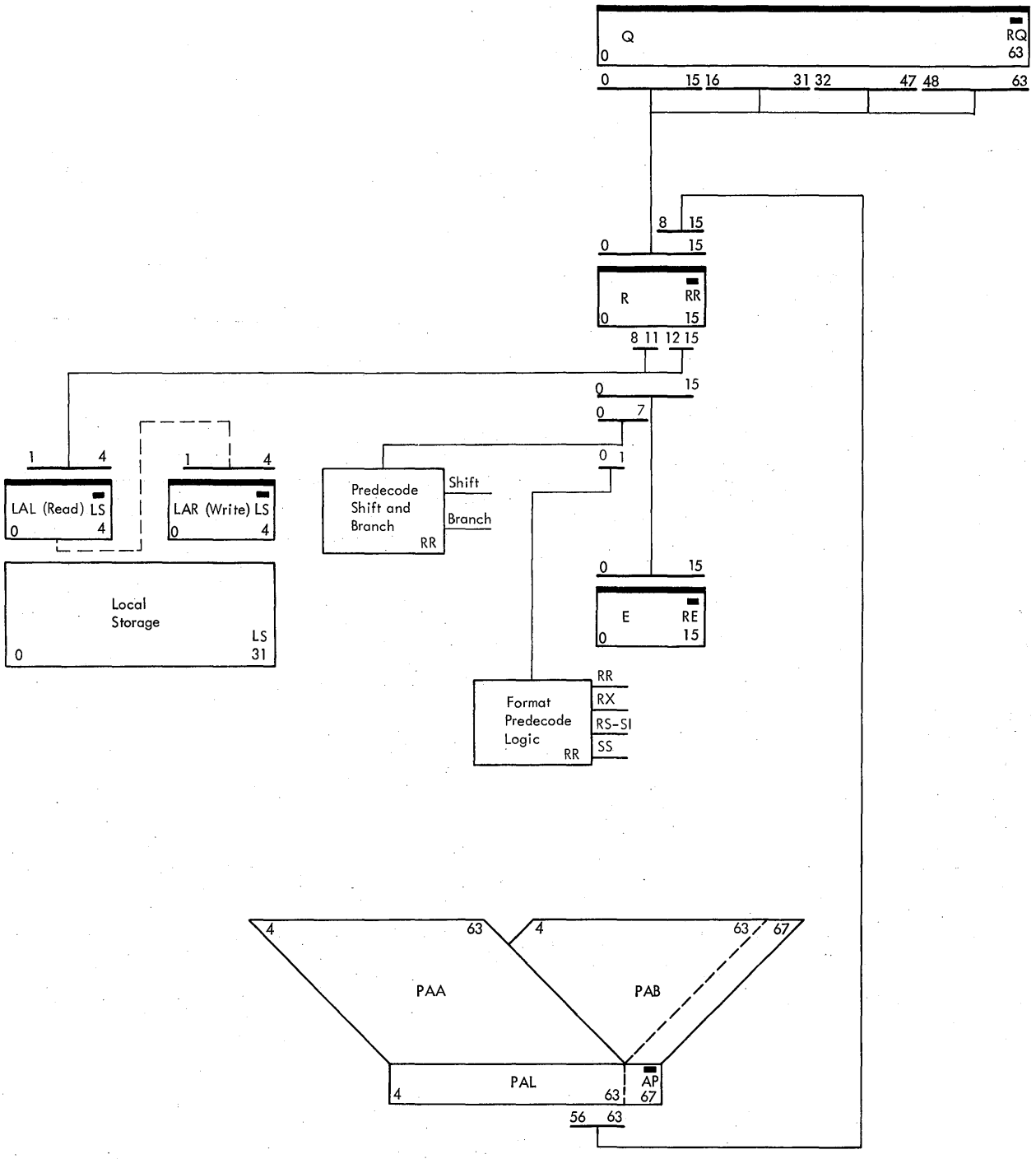


Figure 2-28. R-Register Data Flow

decremented by 1, but E(8–15) can only be *decremented* by 1 (for example, used for reducing length fields in logical VFL operations).

The E-register incrementers consist of latch circuits with logic decoder inputs (Diagram 4-303, FEMDM). The four-position incrementers are not capable of counting, but rather decode the binary information at their input, generate a binary value of 1 greater (or 1 less), and then set that value into the latches. Processing E(8–15) as an entity is accomplished by logically connecting the two four-position incrementers.

ROS controls also load constants into the E-register incrementers during execution of certain instructions (e.g., fixed-point multiply or divide) in order to select serial adder positions when developing products or quotients.

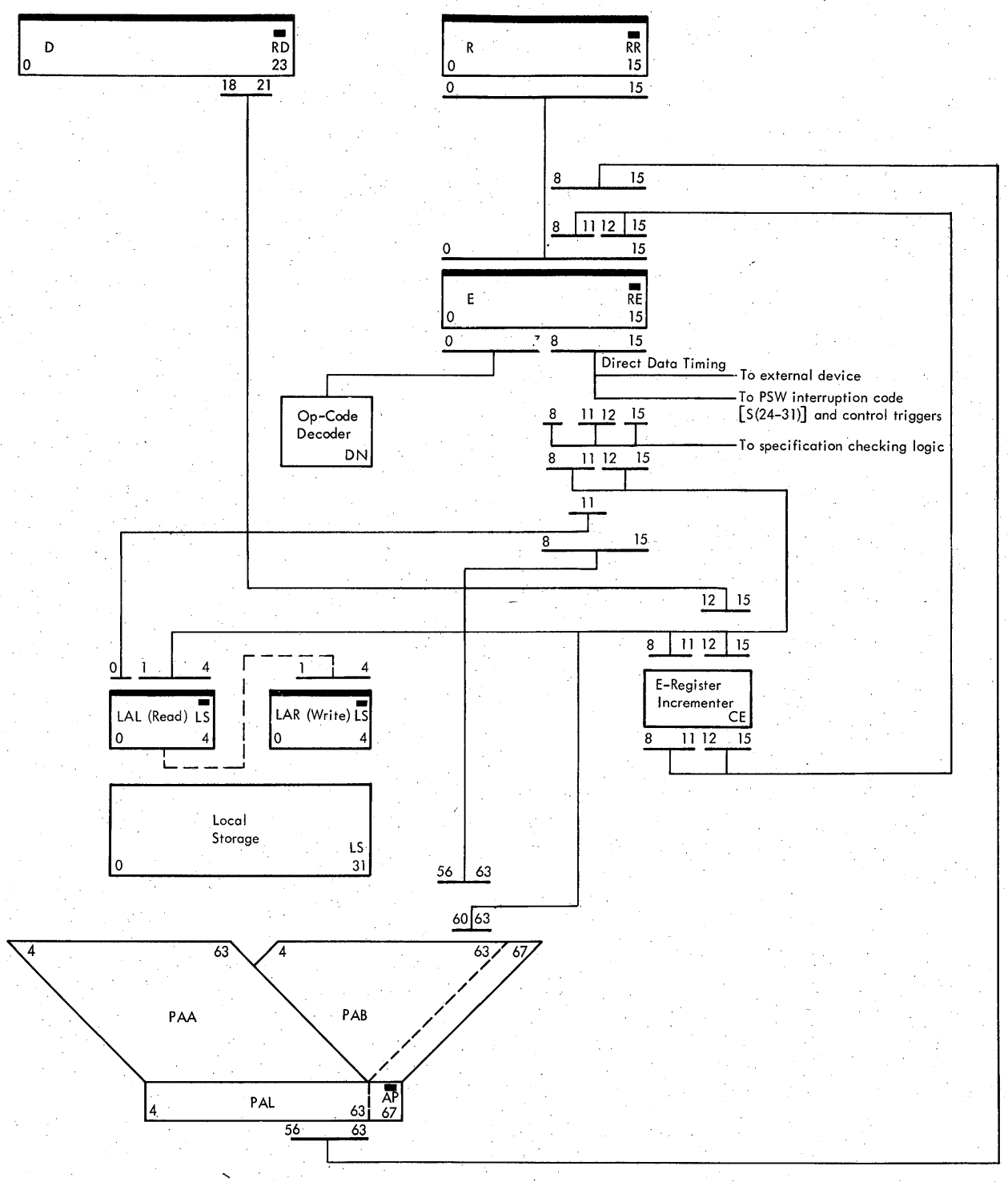


Figure 2-29. E-Register Data Flow

During shift instructions, D(18–21) is gated into the E(12–15) incrementer; decrementing functions reduce the specified shift amount as each shift operation is completed, and thus control the shift instruction.

When E(8–15) is modified in the E-register incrementer, E(8–15) parity may change. Diagram 4-304, FEMDM, shows the parity prediction logic to yield correct parity for E. If, for example, E(15) = 0 and the 'Add 1 to E(12–15)' signal is active, the 'change parity of E(12–15)' signal is developed. Assuming E(8–15) is odd, the 'INCR(8–15) bits even' latch is set which, in turn, sets the 'E(8–15) parity' trigger. Thus, parity is altered at the same time E is modified.

INSTRUCTION COUNTER

The instruction counter (IC) is a 24-bit (plus three parity bits) register used primarily in addressing doublewords of instructions from main storage (Figure 2-30).

Input

PAL(40–63) is transferred to IC(0–23) when incrementing the IC, or when entering a branch instruction as specified by D. Because IC(21,22) selects halfwords in Q, ROS controls the setting of IC(21,22) independently of the parallel adder.

Output

IC(0–23) is transferred to PAB(40–63) to be incremented by 8 so that the next sequential instruction address in main storage will be available in the IC. When called for, IC(0–20) is transferred to the storage address bus (SAB) to address the next instruction doubleword from main storage. IC(23) is transferred to the specifica-

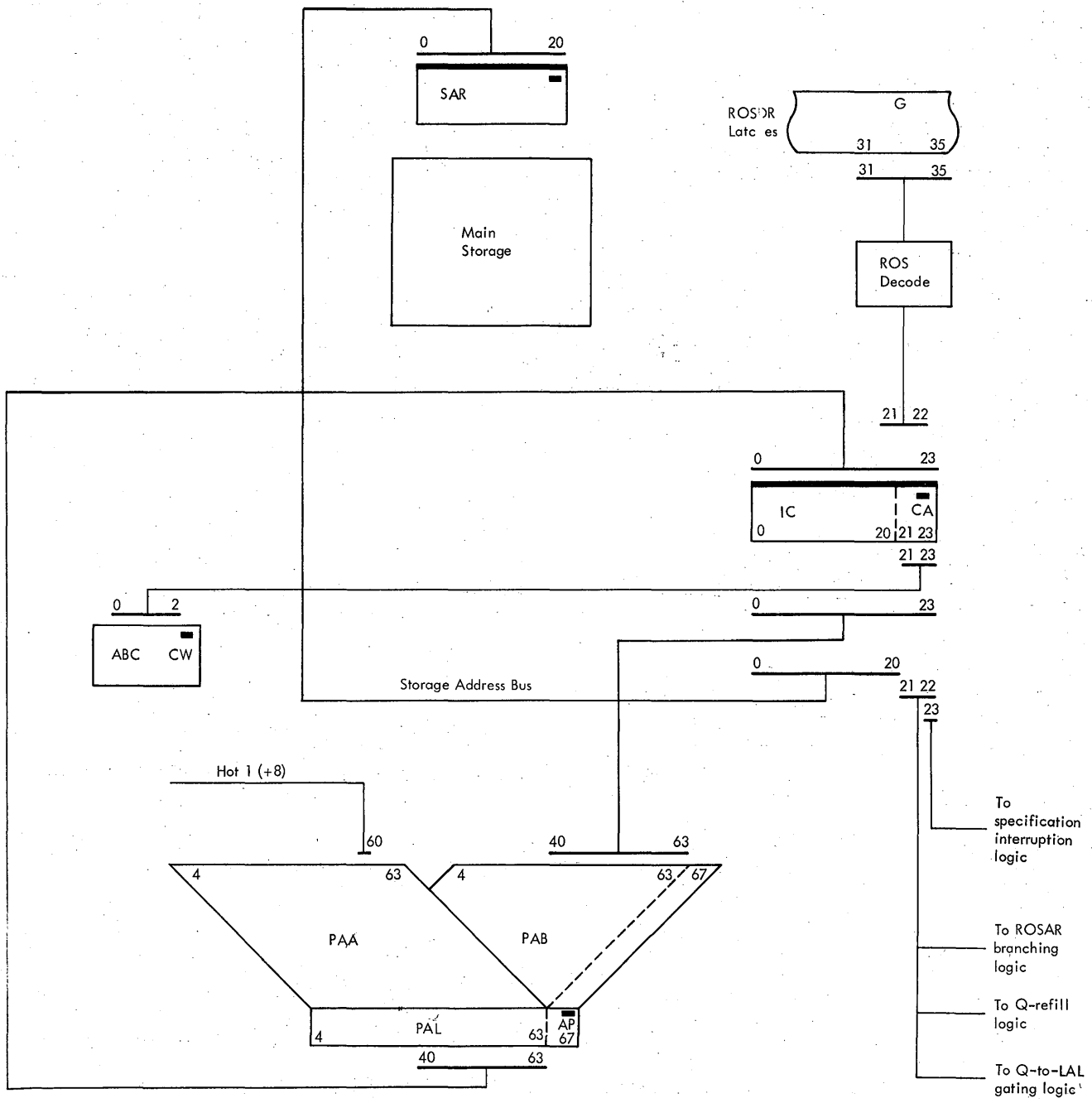


Figure 2-30. Instruction Counter Data Flow

tion logic to test for a 0-bit on instruction addressing; a 1-bit indicates a specification error.

In some instances, the address is for VFL data. Accordingly, IC(21–23) is transferred to ABC(0–2) on VFL operations to specify the desired data byte in AB. IC(21,22), through ROS branching, specifies the Q-halfword to be transferred to R; IC(23) determines the byte within that halfword.

Incrementing IC(0–20)

- After each storage request is generated, IC(0–20) is incremented by 8 to develop address of next sequential doubleword in main storage.

Incrementing (updating) IC(0–20) by 8 to develop the address of the next sequential doubleword in main storage is accomplished using the parallel adder. [IC(0–23) is gated to PAB, and a hot-1 bit is forced into PAA(60).]

At any given time, however, IC(0–20) may be either one or two doubleword addresses ahead of the doubleword in which the instruction being executed is located. These conditions occur as follows. When the instruction being executed is contained in a doubleword still present in Q, IC(0–20) has been updated and is one doubleword (8 byte addresses) ahead of the doubleword in Q. However, when the op-code halfword just transferred to R happens to occupy the last halfword portion of Q [IC(0–20) already being one doubleword address ahead], a storage request is generated to access the next doubleword and the IC is again updated by 8. [IC(0–20) is now two doubleword addresses ahead of the doubleword in which the instruction being executed is located.]

Incrementing IC(21–23)

- After each op-code halfword is transferred from Q to R, IC(21,22) is set to the value corresponding to the Q portion occupied by that halfword.

IC(21,22) values of 00, 01, 10, and 11 correspond respectively to the four (1–4) Q-register halfword portions. On the same cycle in which the op-code halfword is transferred from Q, IC(21,22) is set to the value corresponding to that halfword portion. [IC(21–23) trigger circuitry is not capable of accumulating, but only of receiving, input values.] Thus, these triggers are not stepped or incremented but, rather, set to values indicating the four Q-register halfword areas. IC(21,22) is controlled by ROS words, and, in the case of instruction sequencing, by the same ROS words that gate the Q-register op-code halfword to R.

In the event of a non-RR instruction, IC(21,22) must be changed by 2 or 3 to skip over the non-op-code halfwords remaining in the instruction. This skipping is accomplished as follows. Two factors involved in the 64-way ROS branch (NEXT-INST*IC) occurring at end-op time of each instruction are: (1) the format of the instruction op code previously transferred to R, indicating the number of halfwords composing that instruction; and (2) the contents of IC(21,22), indicating the Q-register area occupied by that instruction. These factors enable the end-op branch to access the proper I-Fetch ROS word for gating out the next sequential op-code halfword and also setting IC(21,22) to the value corresponding to that particular halfword area.

IC(23), set only during VFL operations in which an odd-numbered operand address is set into the IC, is not otherwise subject to change.

Note: IC(21–23) of the VFL operand addresses is placed into the AB counter, which then assumes the function of sequencing through the data-field bytes.

When IC(21,22) is set to new values, the parity of IC(16–23) may change. Parity adjust logic (Diagram 4-305, FEMDM) conditions the IC P(16–23) bit when the 'set IC(21,22)' micro-order is executed. IC(21,22) is set before the 'adjust parity' trigger is set, but circuit delays hold the parity adjust condition until the trigger is set. In effect, the parity adjust logic subtracts the parity of the old value of IC(21,22) and then adds the parity of the new value of IC(21,22), thus resulting in an updated IC P(16–23).

D-REGISTER

The D-register is a 24-bit (plus three parity bits) register which functions as a main storage address register for certain operations and as an I/O channel and unit address register for I/O instructions (Figure 2-31).

Input

Inputs to D are under ROS control. Address information (main storage or I/O) may come from either the parallel adder or the ADDRESS switches. Address information placed into D(17–20) is generated by the interruption logic.

Output

D(0–20) is transferred to the main storage address register (SAR) to provide storage addressing. On channel operations, D(8–15) is transferred to the channel address decoder to select a channel, and D(16–23) is transferred to the channel to provide the I/O unit address. D(18–21) is sent to the E-register, via the E-register incrementer. D(21–23) is sent to the ST byte counter, ROSAR branching logic, and specification checking logic. D(21,22) determines which halfword of Q to use to provide LS address information. The transfer of D(0–23)

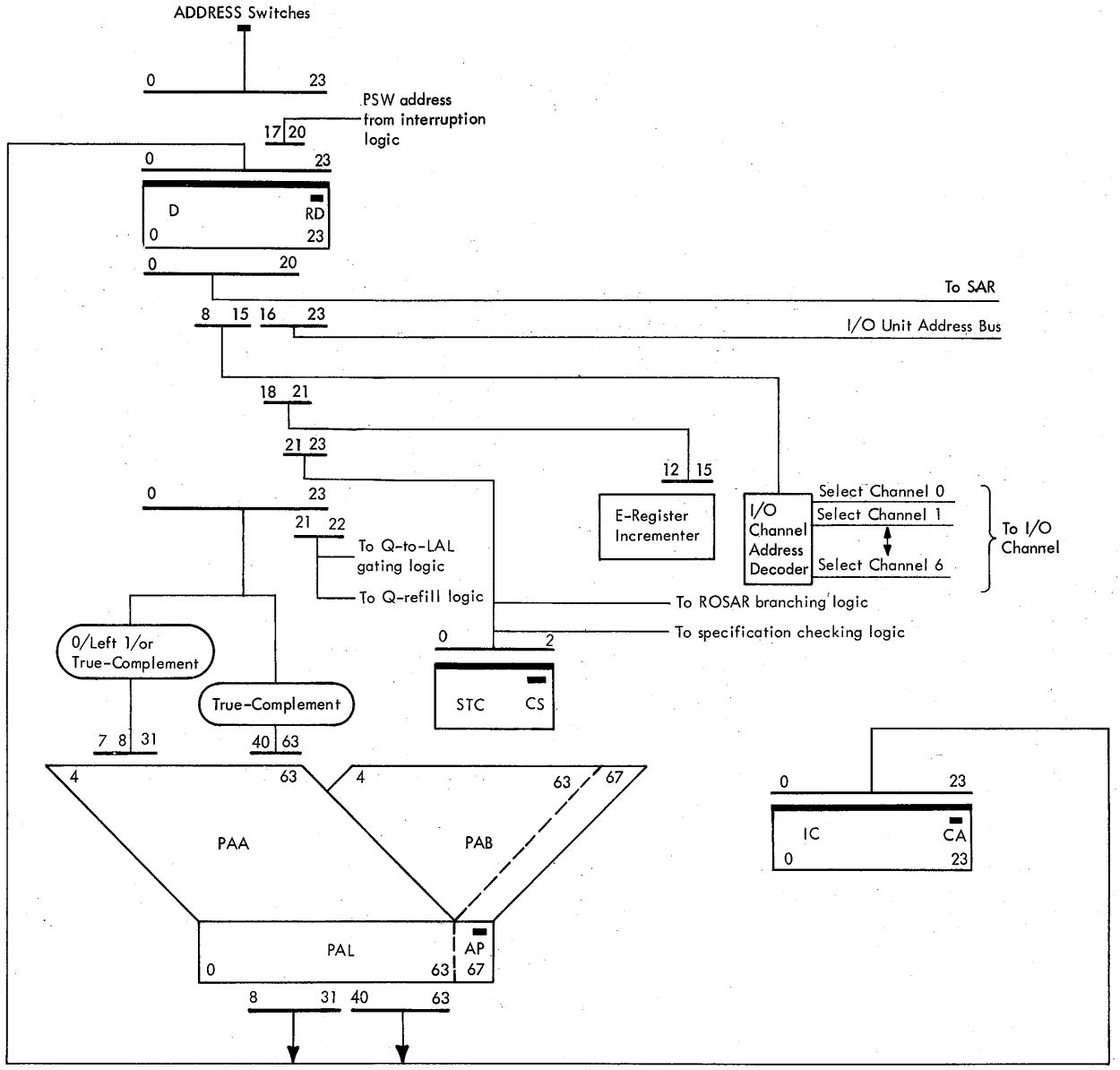


Figure 2-31. D-Register Data Flow

to PAA(8–31) or PAA(40–63) provides the path to alter or update the D-register.

Operational Functions

Operations in which D participates are: (1) branch and execute, (2) shift, (3) VFL, (4) fixed-point, (5) floating-point, (6) manual-control, (7) I/O, and (8) interruption.

Branch and Execute Operations

For branch and execute instructions, the specified successful branch address (all branches are assumed to be successful until otherwise determined) is placed into D by the normal I-Fetch sequence. A storage request is then issued to main storage (per D), and D(0–20) is gated onto SAB. D(21,22) specifies the particular op-code halfword within the doubleword in the same manner as IC(21,22) does for normal operation. If during execution of a branch instruction the branch is found to be successful (branch condition satisfied), the requested doubleword from main storage is gated into Q and the branch address in D is sent to the parallel adder and updated by 8. The result is placed into the IC (replacing the IC address), and the program proceeds in the normal manner.

If, however, the branch instruction is found to be unsuccessful, the doubleword requested from main storage (per D) is not gated into Q and the branch address

in D does not replace the IC address. The program then proceeds with the next sequential instruction.

Shift Operations

For RS instructions, I-Fetch adds the base and displacement values and places the result into D. Normally, this result is the second operand address. For shift instructions, however, this total specifies the number of bit positions to be shifted, and is used as follows. The number of shifts specified by D(22,23), a maximum of 3, are executed immediately upon being computed in the parallel adder and without the use of D. The number of shifts remaining is now specified by D(18–21), which indicates the number of shift operations necessary to complete the shift instruction, provided four shifts are accomplished by each shift operation.

Because left 4 and right 4 shifts are possible in the parallel adder, the binary number in D(18–21) is transferred to E(12–15), where the E-incrementer then controls the remaining number of left 4 or right 4 shift operations required to complete the instruction.

VFL Operations

For VFL operations, destination operand addresses are placed into D by the I-Fetch sequence. (Source operand addresses are placed into the IC.) Storage requests for destination operands are made per D(0–20), and the accessed doubleword is loaded into ST. D(21–23) is set into the ST byte counter to control ST byte transfer. The address in D is updated by 8 following each storage request, and, when the ST byte counter value reaches 7, another storage request is made per D to refill ST with destination operand data.

Fixed-Point Operations

For fixed-point operations, operand addresses are placed into D by the I-Fetch sequence. Operand storage requests are made per D(0–20), with D(21) determining which 32-bit word of the accessed doubleword is to be gated into ST. For halfword operation, D(22) determines which half of the 32-bit word specified by D(21) is to be gated into ST.

Floating-Point Operations

For floating-point operations using long operands, D and T provide for the handling of a 56-bit fraction. The high-order 24 bits of long fractions are contained in D.

Manual-Control Operations

In manual-control operations (manual mode), addresses entered into the ADDRESS switches are transferred to D. A storage request is then made per D to reference main storage for operations such as storing and displaying.

The address is entered into D as follows. Manual operation microprogram routines (for example, store or display) cause the parallel adder to generate all 1's, and then transfer them to D. Those ADDRESS switches not depressed (not set to 1) cause their associated D-register positions to be reset to 0; the resulting bit configuration in D is the address.

I/O Operations

The channel and unit address for an I/O instruction is placed into D(8–23) via the normal I-Fetch sequence, D(8–15) specifying the I/O channel and D(16–23) specifying an I/O device attached to that channel.

During the execution phase, D(8–15) is decoded in the CPU to determine the I/O channel. [Up to seven channels are currently available per system; D(8–12) must therefore always contain zeros, with the binary I/O channel address located in D(13–15).] Outputs from this channel decoder function to select one of seven possible I/O channels. D(16–23) is commonly routed to all available channels as a unit-address bus, and is decoded by the selected channel to select a particular I/O unit attached to that channel.

Interruption Operations

At end op of each instruction, ROS Y-branch (overriding branch) tests are made to check for the presence of any interruptions. Each interruption forces a unique bit configuration into D(17–20), which is generated by the interruption-decode and forced-address logic. (This logic also forces an address into ROSAR to access the first ROS word of the associated interruption-handling routine.) These four positions constitute the low-order bits of doubleword addresses in main storage that contain the new PSW for the various interruptions.

AB REGISTER

The AB register is a doubleword (64 data bits plus 8 parity bits) register that serves as a working register and as a buffer for doubleword operands received from main storage (Figure 2-32). Note that the AB register is logically divided into two 32-bit (plus four parity bits) registers, A and B, and has a four-bit extension, B(64–67), to retain low-order significance during certain shift and arithmetic operations.

Input

All AB positions are reset at clock P1 time of the cycle in which they are selected to receive information; data transfer then takes place at P2 time.

Main storage information (doubleword length) is transferred into AB under ROS control by transferring

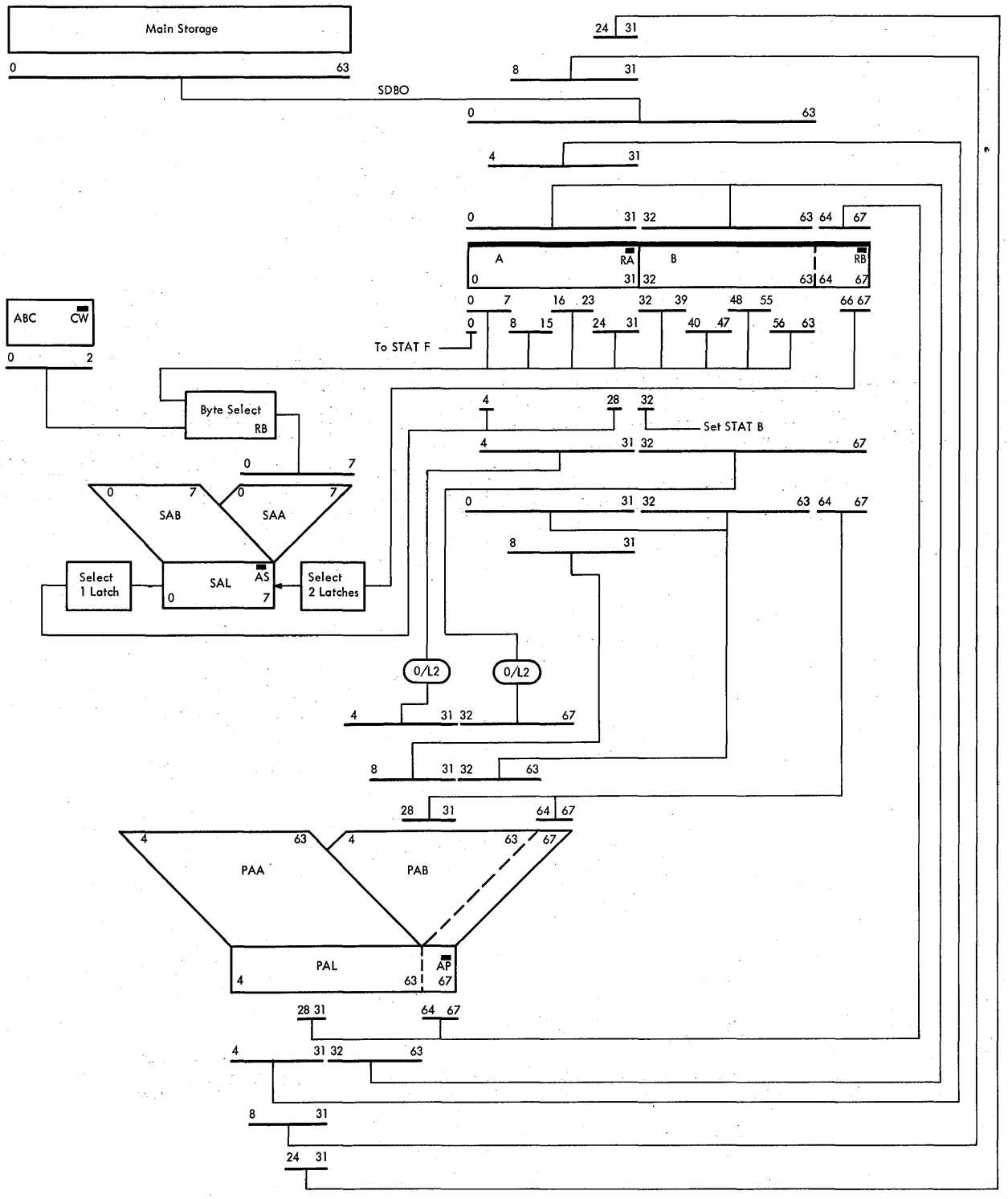


Figure 2-32. AB Register Data Flow

SDBO(0–31) and SDBO(32–63), plus parity, to A and B, respectively.

Parallel adder information (plus assigned parity) is also transferred to A and B under ROS control. Ingating of B(64–67) from PAL(64–67) or PAL(28–31) provides for maintaining high- and low-order significance during shift operations. Transferring PAL(24–31) to A(24–31) facilitates processing of fixed-point divide instructions.

Output

All AB transfer is under ROS control and is accomplished primarily through the use of gate-control triggers. All gate-control triggers are reset at P1 time of each machine cycle; the specific triggers selected for use are set at P2 time of the cycle in which they are to function. (One level of logic delay is incurred in transition; as a result, the respective transfer controls are activated at P3 time of that same cycle.) Selection of the gate-control triggers for use during any given cycle is determined either directly, through ROS decoding, or indirectly, through the AB byte counter. Parity information is transferred on a byte basis.

On multiply operations, the partial product, B(66,67), is placed directly into the serial adder latches (SAL) per E(14,15). Divide operations transfer A(4) or A(28) to one position of SAL per a ROS micro-order and E(14,15).

ST REGISTER

The ST register is a doubleword (64 data bits plus 8 parity bits) register that serves as a buffer between main storage, LS, and the CPU and also serves as a working register for arithmetic and logical operations (Figure 2-33). Note that the ST register is logically divided into two 32-bit (plus four parity bits) registers.

Input

- Inputs are from main storage, LS, parallel adder, serial adder, PSW register, interrupt logic, and DATA switches.

When new data is transferred into ST, only the bit positions involved are reset. Resetting occurs at P1 time and data transfer at P2 time. Reset signals are generated by the gating signals so that doublewords may be assembled (Diagram 4-306, FEMDM).

Main storage information is placed into ST by transferring SDBO(0–63) to ST(0–63), SDBO(0–31) to S(0–31) or T(32–63), or SDBO(32–63) to T(32–63). LS information (32 data bits plus 4 parity bits) is transferred to either S or T. All ST storage activity is controlled by ROS.

PAL(32–63) or PAL(40–63), plus parity, is transferred to T, and SAL(0–7), plus parity, is transferred to the ST byte per the ST byte counter and incrementer. ROS controls the transfer from the adders to the ST register.

PSW information is transferred from the PSW register to S(0–15) and T(32–39) under ROS control. The interruption code from the interrupt logic enters S(16–31) and is stored into the old PSW. Figure 2-34 shows S(16–31) input logic.

For manual control operations, information from the DATA switches is placed into ST for subsequent entry into main storage (or LS) in the following manner. All positions of ST are set to 1's by means of the parallel adder and LS ROS micro-orders. All DATA switches not set to 1's cause their respective ST positions to be reset to 0. Thus, ST reflects the information contained in the DATA switches.

Output

- Outputs are to main storage, LS, PSW register, serial adder, parallel adder, multiply/divide logic, and MCW.

All ST transfer is under ROS control. Transfers to the adders are performed by gate-control triggers. These triggers are reset at P1 clock time of every machine cycle; the specific triggers selected for use are set at P2 clock time of the cycle in which they are to function. (One level of logic delay is incurred in transition, and, as a result, each trigger activates its outgating circuitry at P3 clock time of that same cycle.)

Selection of gate-control triggers for use on any given cycle is controlled either directly by ROS or indirectly by the ST byte counter during ROS-controlled VFL operations. ST transfer to main storage takes place when a storage request is initiated with the 'store' trigger set; 64 data (plus eight parity) bits are then transferred to the SDBI. (Only T-register information can be gated to the LS.)

PSW information from S(0–15) and T(34–39) is transferred to the PSW register under control of ROS micro-orders.

The contents of ST, plus parity, can be transferred to PAA. In addition, T-to-PAA data transfer logic is capable of both true or complement and left 1 shift, and either T(32–47) or T(48–63) can be gated to PAA(48–63).

Byte transfer from ST (for product and quotient insertion during multiply and divide operations) is controlled by E(13–15). (The selected product/quotient bytes are transferred to the MPR bus.)

AB AND ST BYTE COUNTERS

For operations involving the serial adder, it must be possible to extract bytes from doublewords contained in AB and ST and to assemble bytes into ST for subsequent storage. These capabilities are provided by two byte counters: the ABC for controlling AB byte transfer, and the STC for controlling ST byte ingating and outgating.

AB Byte Counter

- Inputs are from PAL(61–63), T(57–59), E(13–15), and IC(21–23).
- ABC logic increments, decrements, or retains absolute values.

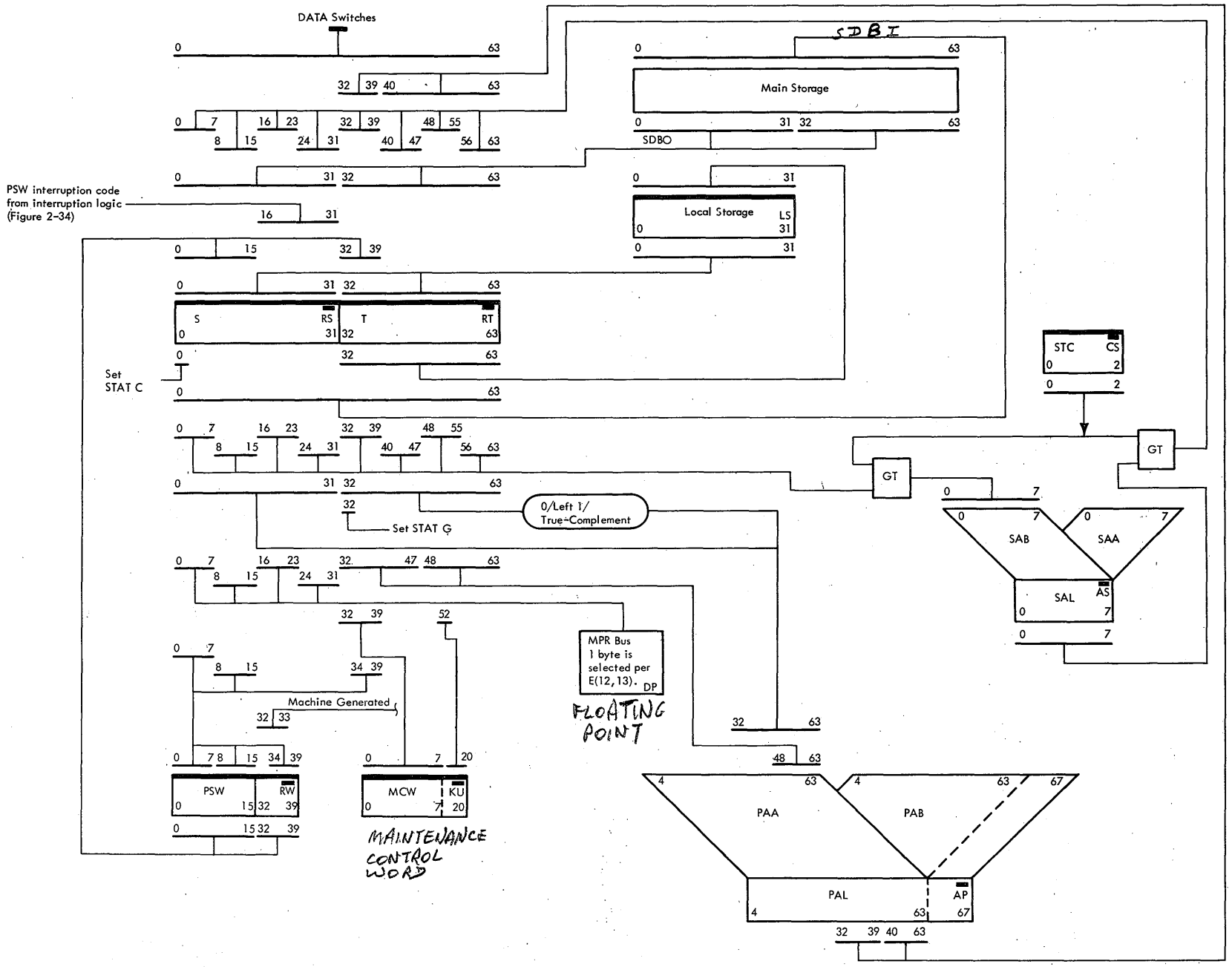


Figure 2-33. ST Register Data Flow

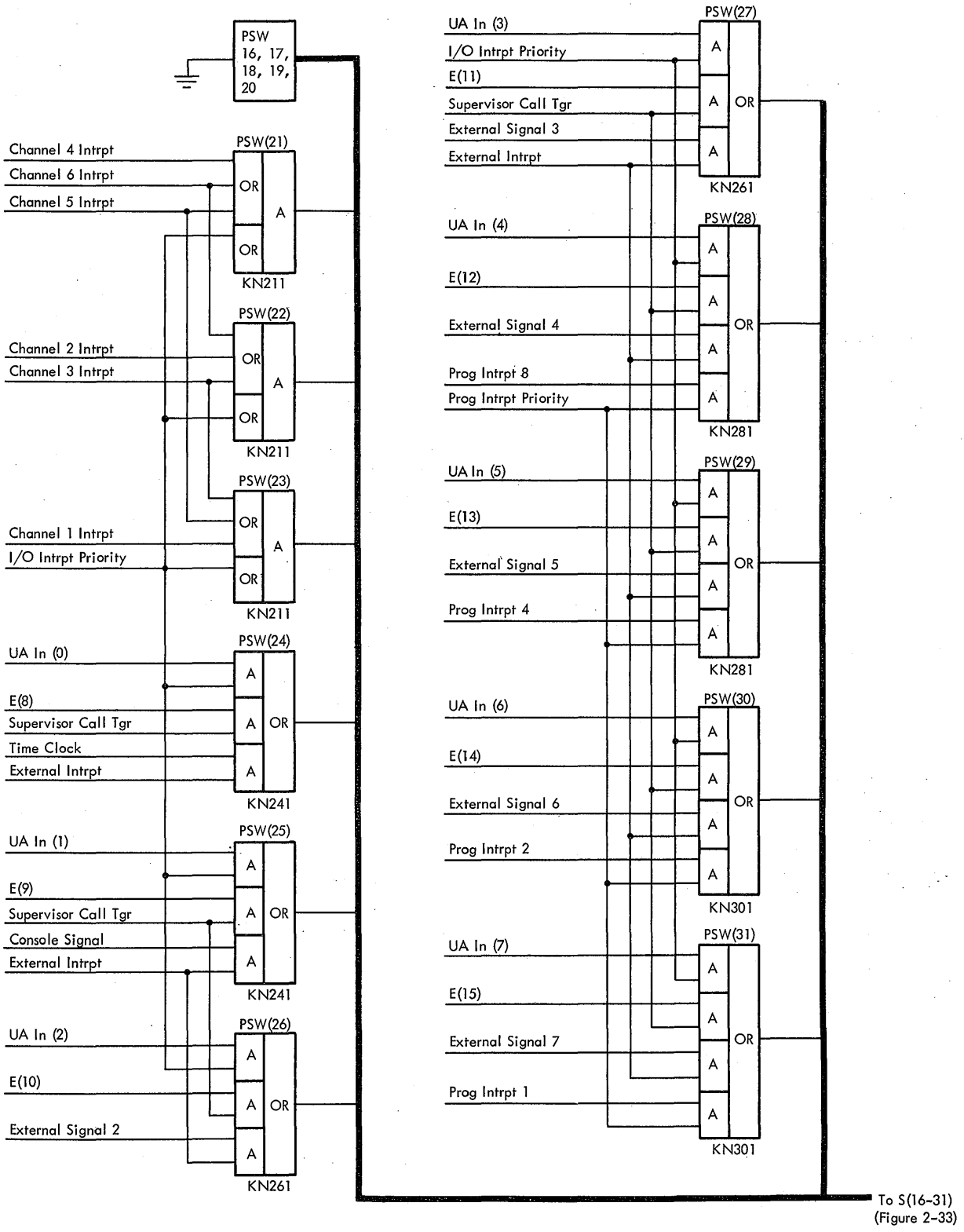


Figure 2-34. PSW Input to S(16-31)

The ABC (Diagram 4-307, FEMDM) consists of three triggers and three incrementer latches. These components are designated T0, T1, T2, and L0, L1, L2, in each group, and represent decimal values of 4, 2, and 1, respectively. Thus the ABC is capable of selecting any AB byte from 0 to 7. Both the trigger and latch groups are capable of receiving information (000–111 binary); modification (incrementing/decrementing) is performed through the use of incrementer-decoding logic on the input of the incrementer latches.

ROS controls the transfer of information into the ABC from PAL(61–63), E(13–15), and IC(21–23); data from T(57–59) is controlled by scan logic.

In operation, binary values of 000–111 (specifying AB bytes 0–7) are transferred into the ABC triggers at clock time under ROS control [ROSDR(25–30)]. The incrementer-decoding logic samples the ABC triggers and, under ROS control, sets that value, incremented by 1, decremented by 1, or absolute, into the incrementer latches at not-clock time. The incrementer latches are then sampled, and the outputs are decoded into eight lines (0–7) to select one of the eight AB bytes for transfer on the following machine cycle. In addition to controlling register transfer, ABC trigger outputs are utilized by scan operations and certain ROS-branch-decode functions.

Note: E(13–15) or IC(21–23) can also be entered directly into the incrementer latches, at not-clock time, under ROS control.

The ABC triggers are reset with a negative P1 clock pulse at clock time of each machine cycle and set with incoming data at P2 clock time. The incrementer latches are reset with a negative P2 not-clock pulse at not-clock time of each machine cycle, with data transfer occurring at P3 not-clock time.

The contents of the ABC triggers are transferred to the incrementer latches, and the same value is returned to the ABC triggers (regenerated), during each machine cycle in which no other ABC transfer controls are specified by ROS (provided that an 'I-Fetch reset' signal does not occur).

ROS control signals ('000 to ABC' and 'I-Fetch reset') reset the ABC and ABC incrementer to 0 by allowing both the triggers and the latches to reset on the following machine cycle.

ST Byte Counter

- Inputs are from PAL(61–63), T(54–56), E(13–15), and D(21–23).
- STC logic increments, decrements, or retains absolute value.

The STC (Diagram 4-308, FEMDM) consists of three triggers, three bipolar (polarity-hold) latches, and three incrementer latches. The triggers are designated as T0, T1, and T2 and the latches as L0, L1, and L2, representing decimal values of 4, 2, and 1, respectively. Thus the STC is capable of selecting any ST byte from 0 to 7. The STC triggers (with associated polarity-hold latches) and the STC incrementer latches are capable only of receiving information (000–111 binary); modification (incrementing/decrementing) is accomplished through the use of incrementer-decoding logic on the input of the incrementer latches. The polarity-hold latches retain each STC setting for one additional cycle, providing for a resultant data byte to be gated into ST at the same time the next sequential ST byte is being gated for processing.

ROS controls the transfer of information into the STC from E(13–15), D(21–23), and PAL(61–63). Entry of T(54–56) to the STC is controlled by scan logic.

In operation, binary values of 000–111 (specifying ST bytes 0–7) are transferred into the STC triggers at clock time, with the associated polarity-hold latches assuming the same value at not-clock time of that same cycle. The incrementer-decoding logic samples the contents of the STC triggers and sets that value (incremented, decremented, or absolute) into the incrementer latches. The incrementer latches are then sampled, and the outputs are decoded into eight lines (0–7) to select the ST bytes for transfer during a subsequent machine cycle.

Note: E(13–15), D(21–23), or a defined constant can also be entered into the incrementer latches under ROS control.

The polarity-hold latches are set (or reset) at not-clock time to the value of the STC triggers. This value (specifying an ST byte) is retained in the polarity-hold latches until not-clock time of the following cycle. Thus, at clock time of the following cycle, with the STC triggers having been set to a new value and the incrementer latches possibly containing a modification of this new value, the previous STC-trigger setting is still present in the polarity-hold latches. This retained value now allows information from the serial adder to be placed into the ST byte that was previously transferred out, at the same time that the incrementer latch output is transferring the next sequential ST byte to the serial adder for processing.

All incrementer latch decode lines are sent to the mark-trigger logic (specifying byte areas for main storage entry), and decode lines 0, 3, and 7 are sent to the branch logic controlling the ROSAR setting. An 'STC greater than 3' signal is also transferred to the branch logic controlling ROSAR whenever incrementer latch 0 (binary value of 4) is set.

Gating of the contents of the STC triggers into the incrementer latches and regeneration of that latch value to the triggers are performed each machine cycle in which no other STC ingating controls are specified by ROS (provided that an 'I-Fetch reset' signal does not occur).

The '000 to STC' and 'I-Fetch reset' signals reset all STC triggers and latches on the following machine cycle. The '1 to STC bit 0' and 'I-Fetch reset' (for RR instructions) signals cause the incrementer to assume a decimal value of 4; the '011 to STC' signal sets incrementer latches 1 and 2, thus setting the incrementer to a decimal value of 3.

MARK TRIGGERS

Eight mark triggers, contained in the CPU, indicate which bytes of the doubleword on SDBI are to be entered into main storage on a store operation.

Mark trigger logic (Diagram 4-309, FEMDM) is ROS controlled; operation is as follows. ROS control field L (ROS sense latch positions 43–46) is decoded to activate one or more of four mark trigger signal lines. These lines set the mark triggers as required: (1) individually, per the STC, (2) in groups of four (0–3 or 4–7), and (3) unconditionally, by setting *both* the 0–3 and 4–7 groups. ROS micro-orders to set mark triggers also set the 'store' latch to generate a 'store' signal which is sent to the selected storage unit.

F-REGISTER

The F-register is a one-byte (plus parity) trigger register that is used in certain arithmetic, logical and data-transfer operations (Figure 2-35).

Input

Inputs to F are under ROS control. All positions involved in an operation are reset at P1 clock time of the same cycle in which they are to receive information, with the ingating occurring at P2 clock time. Data and external control information is received during direct-control read operations, and serial adder outputs are received during VFL operations. F(0–3), F(0–4), and F(4–7) are utilized by Set Key and Insert Key instructions, logical instructions, and decimal multiply and divide instructions.

Output

All outputs are under ROS control. F(0–7) is transferred to the serial adder by means of a gate-control trigger at clock time, and F(4–7) is transferred to the parallel adder under control of the parallel adder input logic. F(0–3) is transferred to the storage protect area during set-key operations. F(4–7) is also used in ROS branching.

G-REGISTER

The G-register is a one-byte register that buffers one byte of data between the CPU and an external device (Figure 2-36). SAL(0–7) is the only input, and the external device is the only transfer path.

PSW REGISTER

- Inputs are from ST and interruption-control logic.
- Outputs are sent to ST and CPU control circuitry.

Although the PSW is 64 bits in length, the PSW register contains only 24 bits (Figure 2-37). The remaining information (generated by the CPU at the time of an interruption) is used to identify the cause of the interruption and to allow the CPU to return to the correct program address.

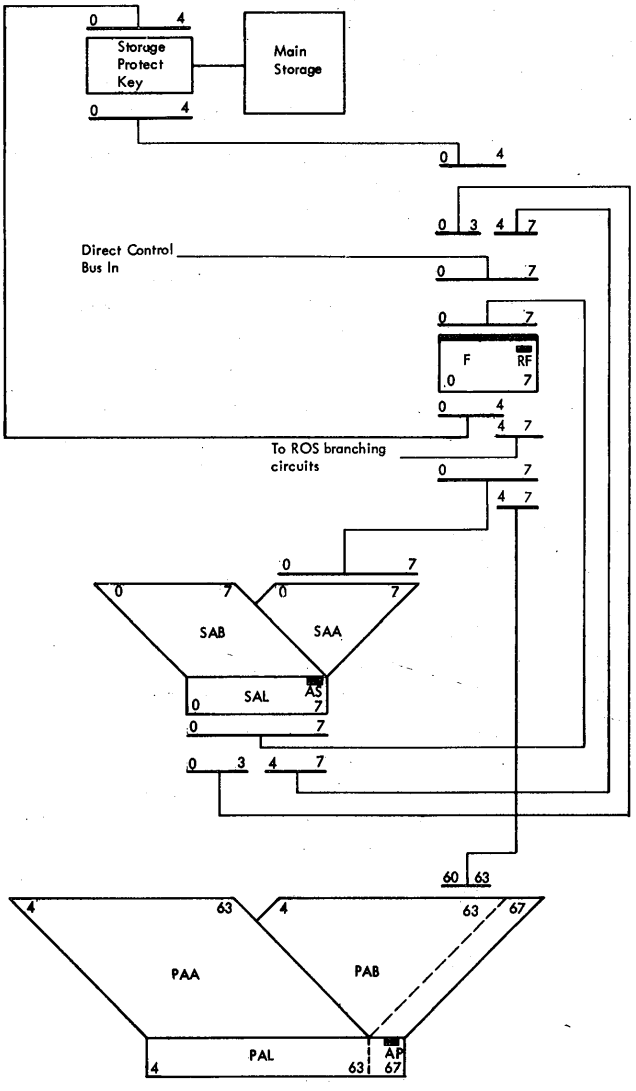


Figure 2-35. F-Register Data Flow

PSW register trigger logic is shown in Figure 2-38. The 'gate S(0–7) to PSW(0–7)' signal resets the triggers and, through the logic delay, provides the gating signal to allow the ST register information to enter the PSW register. Thus the information remains in the PSW register until it is replaced by a new PSW.

All PSW register input and transfer is initiated by ROS micro-orders. When an interruption occurs, a series of micro-orders in the accessed ROS word transfer the contents of the PSW register into ST for subsequent entry into main storage.

The format of the instruction in E (interrupted instruction) is decoded, thus providing the instruction-length code to be entered into the PSW register before transferring the contents of the PSW register to ST. Micro-orders also transfer the old PSW address (generated by the interruption control logic) for that particular interruption to D(17–20) to develop the old PSW address for that interruption. Either 8 or 16 (depending on the current instruction address) is subtracted from the IC and inserted into the instruction-address field of the assembled PSW.

Micro-orders executed by the Load PSW, Set System Mask, or Set Program Mask instructions control the transfer of PSW information from the SDBO to ST and the transfer of PSW data from ST to the PSW register and the IC. The old PSW address (contents of D + 64, decimal) is generated in the parallel adder, also under ROS control.

The PSW register does not contain data transfer logic; PSW information is constantly available throughout the CPU for use as required.

MCW REGISTER

The MCW register is a nine-bit trigger register that provides program control of scan operations (Figure 2-39). During execution of the Diagnose instruction, FLT's, or ROS tests, MCW(0–7,20) is gated from T(32–39,52) to the MCW register. The bits of the MCW are retained in the MCW register and decoded by the MCW decoder to perform the functions as specified in Chapter 6.

Note: Four MCW's use the same MCW register: (1) FLT, (2) ROS test, (3) Diagnose for CPU, and (4) Diagnose for channels. See Chapter 6, Section 2, for the format of each MCW.

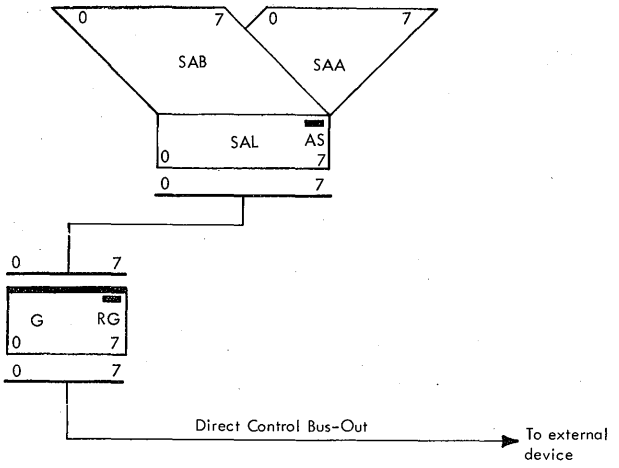


Figure 2-36. G-Register Data Flow

www.IBM360.info

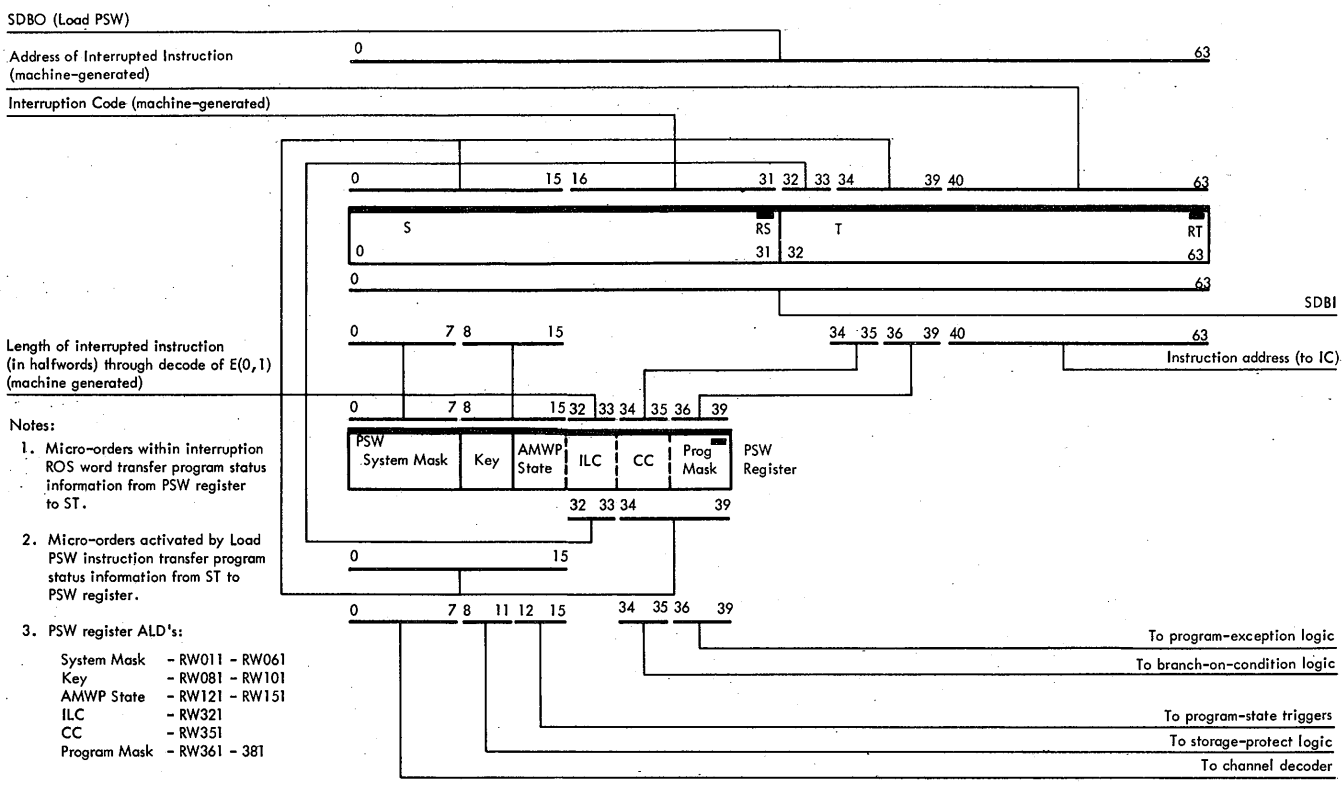


Figure 2-37. PSW Register Data Flow

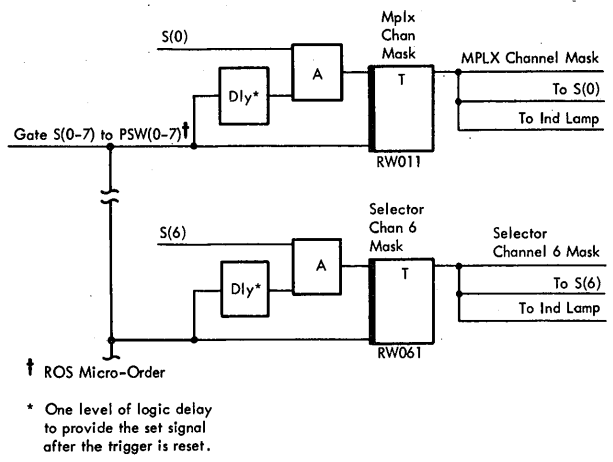


Figure 2-38. PSW Register(0,6) Logic

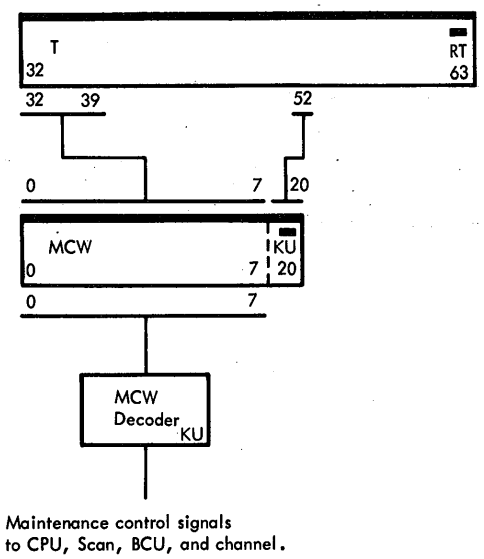


Figure 2-39. MCW Register Data Flow

This section describes the operation of the 25-register local storage (LS).

ADDRESSING AND DATA FLOW

- 5-position address registers [LAL (Read) and LAR (Write)] address LS.
- Input to LS is from T only; output is sent to S and/or T under ROS control.

Two five-position LS address registers [LAL (Read) and LAR (Write)] select the 25 individual LS registers (Fig. 2-40). The LS address is received from Q, R, or E under ROS control (or directly from ROS control words when addressing the LSWR). The particular register (Q, R, or E) and field within that register to be set into LAL is determined by decoding ROSDR(38–42). The four-bit LS addresses are gated into the four low-order positions of LAL from Q, R, or E fields. These four-bit addresses are capable of directly addressing registers 0–15 (general-purpose registers). For floating-point operations (requiring the use of registers 16–23), a 1-bit is forced into the high-order position of LAL upon decoding of the floating-point op code. This action increments the four-bit address from Q, R, or E by 16, thus forcing the use of floating-point registers 16–23.

Note: Floating-point instructions are restricted to the use of even LS addresses 0, 2, 4, and 6. Automatic incrementing of these values by 16 then generates LS addresses of 16, 18, 20, and 22.

Long-operand floating-point op codes also force a 1-bit into the low-order position of LAL, in addition to the high-order 1-bit forced by all floating-point op codes. This additional bit further increments the 16, 18, 20, and 22 floating-point addresses by 1, thus generating the second (R1 + 1) register address required for long-operand (64-bit) instructions.

For operations requiring use of the LSWR (register 24), the ROS words controlling these operations force 1-bits into the two high-order positions of LAL [LAL(0,1)]. This action generates a binary address of 24, and is the only means of selecting the LSWR.

Selection of the Q, R, or E field to be entered into LAL is determined by decoding ROSDR(38–42) of the controlling ROS word or by selecting 'NEOP' or 'BEOP' micro-orders, depending on the next programmed instruction. Regardless of the address source or of whether a read LS or write LS operation is indicated, the contents of the addressed register are always read out onto the LS data bus. If a read LS operation is indicated, decoding of ROSDR(10,11) of the controlling ROS word gates the contents of the LS data bus into S, T, or both S and T. When a write LS operation is indicated, the contents of the addressed LS registers are gated out onto the LS data bus in the same manner, but the output resulting from decoding ROSDR(10,11) remains inactive and does not condition the ST ingating controls. A 'write into LS' signal, resulting from the ROSDR(38–42) decoder, then gates the ST bus (T-data) into the addressed LS register.

DATA TRANSFER CONTROLS

The following paragraphs describe the LS logic involved in read LS and write LS operations. Diagram 4-401, FEMDM, illustrates the read/write logic of LS register 0 and also the common control circuit timings for each 200-ns LS cycle. (Registers 1–24 are identical with register 0.)

Local storage addressing and all LS register-operating logic are implemented in 10-ns circuitry; the polarity-hold latches and associated input logic are implemented in 30-ns circuitry.

Read LS Operation

- ROSDR(38–42) sets LAL from specified Q, R, or E field.
- LAL gates contents of selected LS register to LS data bus.
- ROSDR(10,11) gates LS data bus into ST.

Read LS operations are initiated at not-clock time when LAL (Read) is set with Q, R, or E information, as determined by ROSDR(38–42). The LSWR address and the floating-point register address bits are also entered into LAL at this time, depending on ROSDR(38–42). The contents of LAL (Read) are decoded, and the decoder outputs gate the contents of the selected register polarity-hold latches to the LS data bus. ROSDR(10,11) then activates the required ST input logic, and at clock time of the following cycle the LS data bus information is set into the ST triggers. Polarity-hold circuits provide nondestructive readout, eliminating the need for regeneration. (Refer to the timing chart in Diagram 4-401, for relative control timings.)

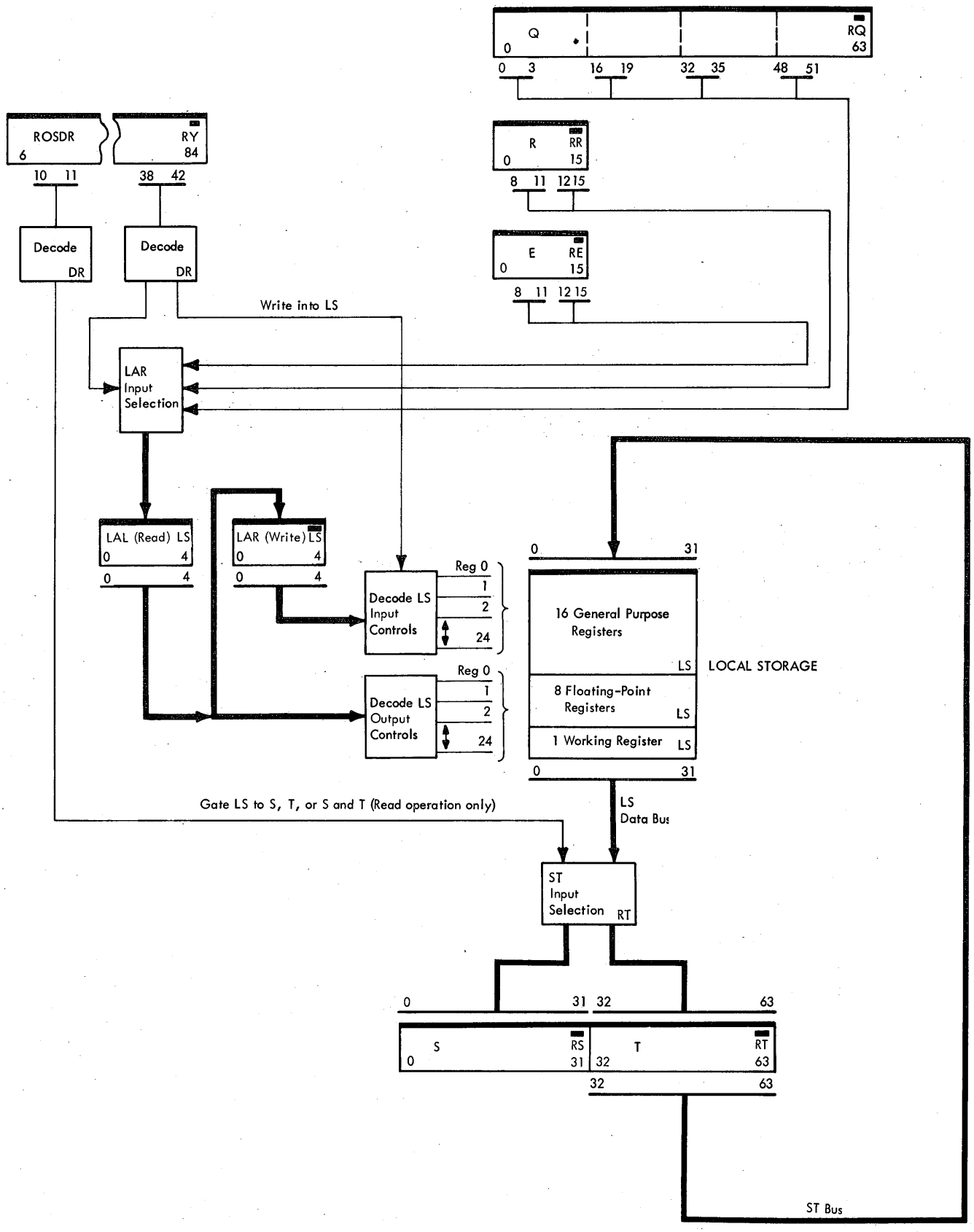


Figure 2-40. Local Storage Data Flow

Note: LAL (Read) is sampled every cycle, even if no command is given. Data from LS is thus placed on the LS data bus every cycle, but is not always gated into a register. The contents of LAL (Read) are transferred to LAR (Write) every cycle, but no addressing is performed unless a write LS micro-order is decoded in ROS.

Although LS data is available to the CPU approximately 100 ns after the setting of LAR, consecutive LS data readout is limited to 200 ns. LS cycles are therefore defined as being 200 ns long.

Write LS Operation

- ROSDR(38–42) sets LAL (Read) to specified Q, R, or E field.
- LAL (Read) contents are transferred to LAR (Write).
- LAR (Write) decoder selects specified LS register.
- ROSDR(38–42) gates ST bus data into selected LS register.

On write-LS operations, LAL (Read) latches are set at not-clock time with the specified Q, R, or E information. At the beginning of the following cycle, LAL (Read) is transferred into LAR (Write) in the same manner as for a read operation. The selected LS register is also gated to the LS data bus as in read operations; up to this point, read and write operations are identical. (On write LS operations, however, LS data bus information is not gated into the ST register.)

At the beginning of the following cycle, LAL (Read) is transferred into LAR (Write). Further decoding of ROSDR(38–42) generates a 'write into LS' signal that sets the 'write LS' trigger at P0 time. This trigger provides the signal to gate the ST bus (T-data) into the selected LS register at not-clock time of the following cycle. (Refer to the timing chart in Diagram 4-401 for relative write control timings.) Negative levels on the ST bus represent 1 bits and set the respective polarity-hold latches; positive levels represent 0's and reset the respective polarity-hold latches.

For an 'insert sign' micro-order when the result sign is minus, T(32) is forced to a 1. To preserve proper parity for this operation, the parity bit for T(32–39) is inverted before it is transferred to LS.

There are situations when writing into LS must be inhibited. When such a situation occurs, the 'SPEC' (K31) micro-order causes a set signal to the 'inhibit LS write' trigger so that the LS positions remain unchanged.

Note that a minimum of 40-ns coincidence must exist between stable ST bus data (1's or 0's) and an active 'gate T to LS' polarity-hold-latch control signal to give correct data entry. When the 'gate T to LS' signal is deactivated, polarity-hold latches of the selected register remain in their present state until new data is entered on a subsequent write LS operation.

Note, too, that LS data is not parity-checked until it enters an adder at a later time.

LS Timing

Separate address registers for reading and writing coordinate the LS timing to other CPU functions. The 'gate LS reg n' signal is generated by LAL (latch timing) to make the LS data available at clock time for entry into a register. LAR, at not-clock time, allows the data entered into T early in the cycle to be entered into LS late in the cycle. Refer to Diagram 4-401, which illustrates the entry of new data into T (reflected by the shift in the ST bus data line) and the subsequent storage of that data into LS.

www.IBM360.info

Section 6. Serial and Parallel Adders

This section describes the operation and application of the serial and parallel adders.

SERIAL ADDER

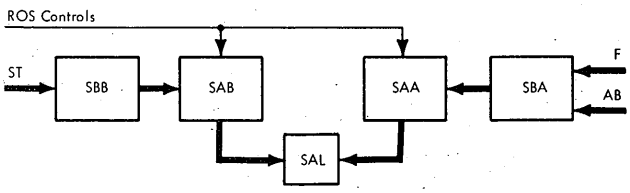
The serial adder (8 data bits plus 1 parity bit) processes data in binary or decimal format, performs logical AND, OR, and Exclusive-OR functions, and assembles multiply/divide results.

Data flow for the serial adder is illustrated in Figure 2-41. Note that data entered into the A-side of the adder (via final bus-A) comes from either AB or F under ROS control.

Data entering the serial adder is in true or complement form. For a true add operation, the data is entered directly; for a complement add operation, the input data to the serial adder A-side (SAA) is inverted (Fig. 2-42). Note that the input data may be in true or complement form. The two's complement value is achieved by forcing a hot-1 to the input logic for serial adder latch (SAL) position 7.

Input and Output

Inputs to the serial adder A-side (SAA) are the contents of F or a selected byte from AB (per the ABC); the input to the serial adder B-side (SAB) is a selected byte from ST (per the STC). A bus arrangement transfers data from the registers to the serial adder as follows:



During transfer from the serial adder bus A (SBA) to the SAA or from the serial adder bus B (SBB) to the SAB, ROS controls can alter the data being transferred. The SAA input can be altered by the following functions: decimal excess-6, complement add, shift, crossgating (interchanging of incoming bits 0–3 with bits 4–7), and zone and sign insertion. The SAB input may be altered by the following functions: sign insertion, special digit insertion, and special gating for changing destination (for example, placing bits 0–3 into bit positions 0–3 and 4–7).

After the sum has been developed and placed into SAL, gating signals from ROS allow the information to be transferred to F, to G, or to a selected byte in ST.

Adder Operation

A simplified summary of serial adder operation is shown in Figure 2-43. SAA and SAB are combined in the serial adder to produce a bit-carry, a bit-transmit, or a half-sum. A bit-carry is developed when both input bits are present, a bit-transmit when either input is present, and a half-sum when only one of the input bits is present (Figure 2-44). Carry-in and half-sum conditions combine to produce a full sum. The table in Figure 2-44 shows the conditions which produce a full-sum bit. For example, if SAA is a 1-bit and SAB is a 0-bit there will be no bit-carry, but a bit-transmit and a half-sum will be produced. If no 'carry in' signal is present, the full-sum is a 1.

The 'carry in' signal is developed by the carry lookahead logic. A test is made for a carry from the next lower position or for a carry developed from bit-transmits and a lower-order carry (Figure 2-45). The carry look-ahead logic saves time by providing an immediate carry rather than using another cycle to ripple a low-order carry through the adder.

Accurate results are achieved by parity checking and parity correction circuits. Tests for error conditions are made at half-sum and full-sum levels as well as on decimal input data.

Controls

- Selected data enters on SBA and SBB.
- Data is first modified on transfer from SBA to SAA and from SBB to SAB.
- Final modification occurs as data enters SAL.

There are three control areas: input bus, final bus, and SAL (Figure 2-46).

ROS sense latch 86, field R, selects F or AB as a data source for SBA. If the latch is set, F is selected; if it is reset, AB is selected. Gate control triggers provide the gating signals to transfer data to the buses. For SBA, if F is not selected, an AB gate control trigger is selected by the value in the ABC (Diagram 4-501, FEMDM). Similarly, the STC selects an ST gate control trigger. One byte of data is selected by each gate control trigger, and two gate control triggers are selected (one for SAA and one for SAB) each machine cycle whether or not the data will be used.

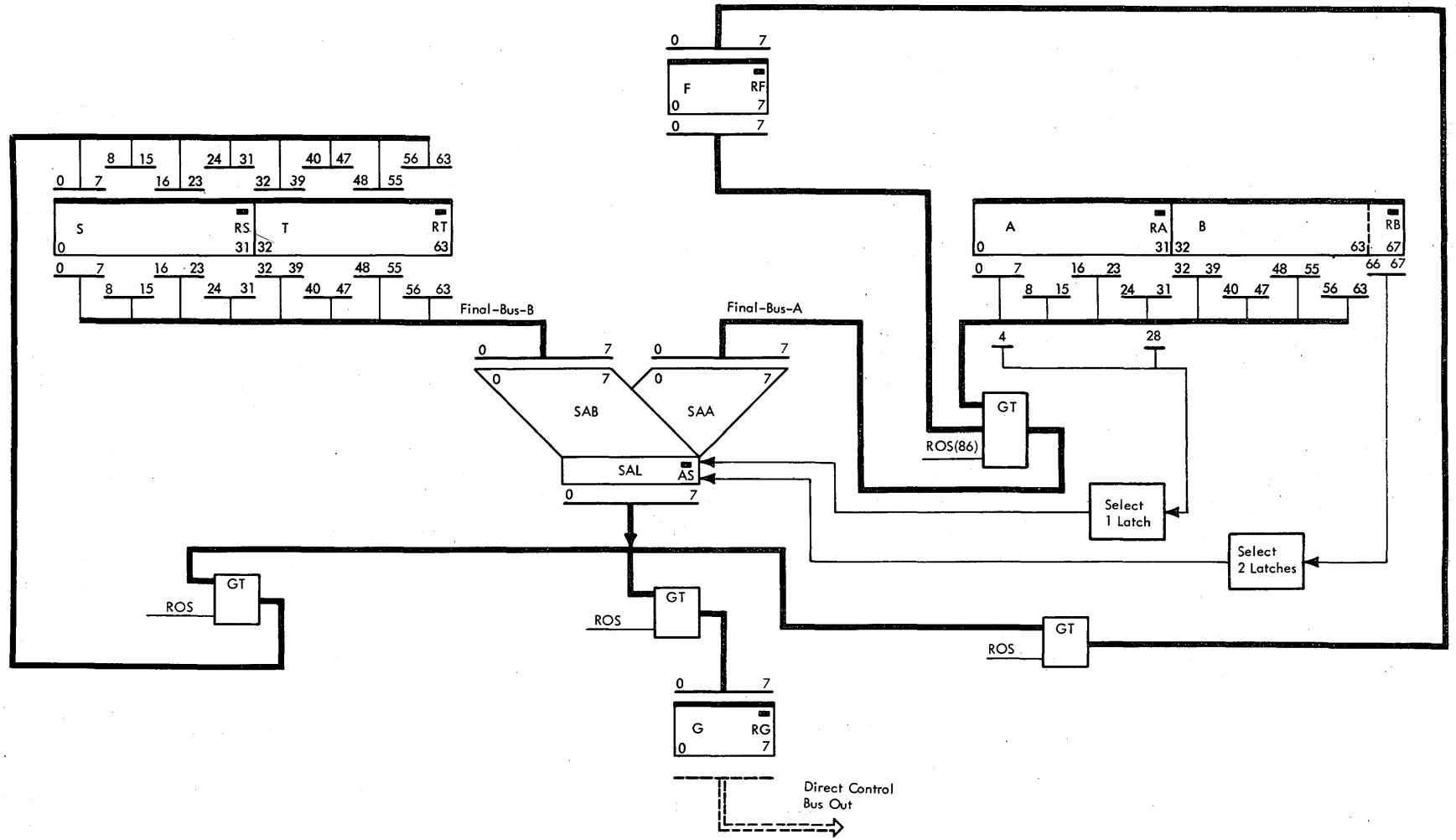


Figure 2-41. Serial Adder Data Flow

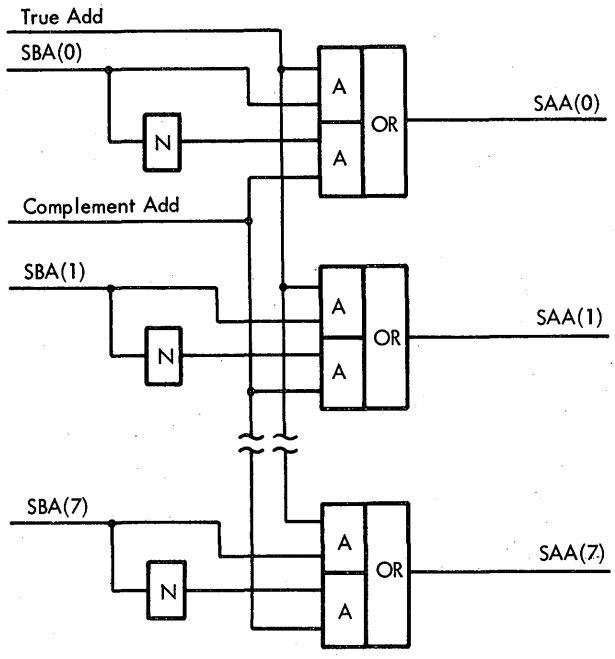


Figure 2-42. True-Complement Data Entry

ROS fields M (bits 69–73) and N (bits 74–77) govern the second area of data control; that is, the transfer of data from the input buses to the A- and B-sides of the adder. ROSDR(69–73) provides signals to control transfer from SBA to SAA. When no control is present, SAA(0–7) is, in effect, 0. Micro-orders allow true-complement transfer, crossgating, excess-6 adding for decimal operations, forcing of certain bits, and sign insertion (Diagram 4-501). ROSDR(74–77) provides similar, though less extensive, control for transfer from SBB to SAB. A list of the micro-orders generated by control fields M and N is contained in ALD M7031.

The third control area, SAL input, is also governed by ROS fields M and N. Controls include logical functions, decimal correction, product/quotient operations, and a hot-carry to SAL(3) or SAL(7) for complement add operations.

Functional Discription

Because the serial adder is used in many operations and is so versatile, a general discussion is not sufficient. Accordingly, the following paragraphs discuss the adder functions individually.

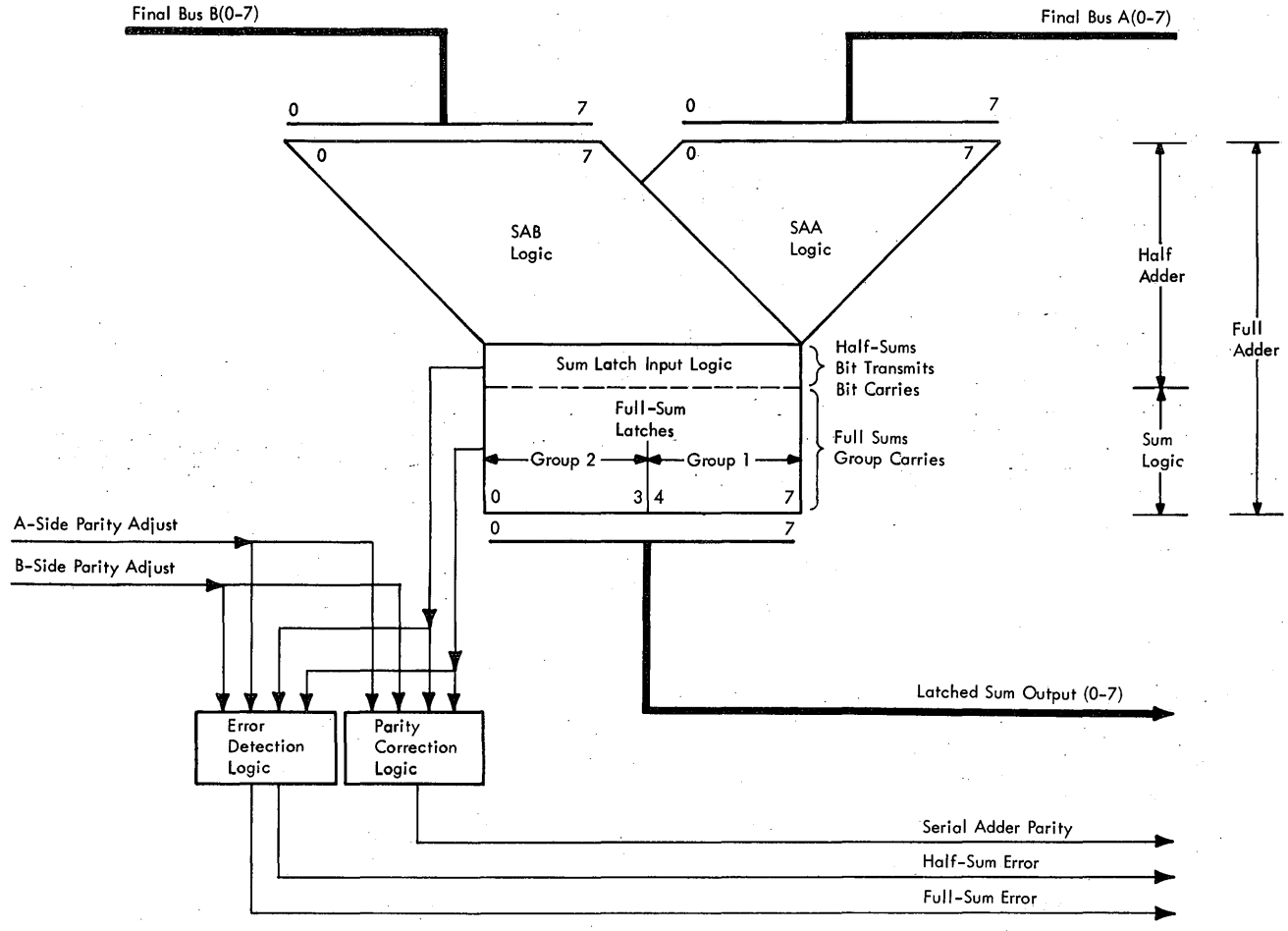
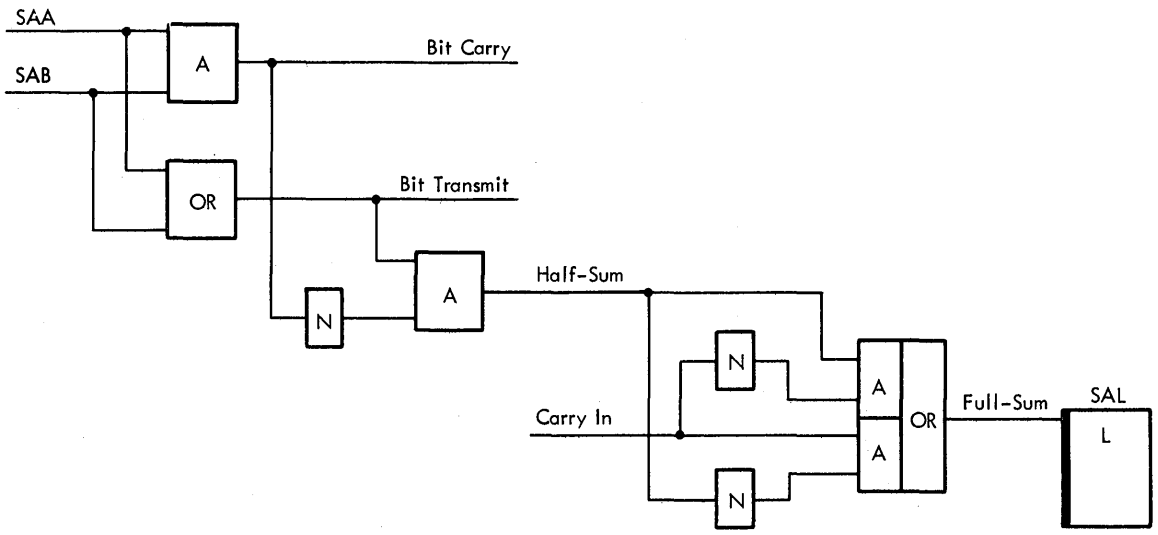


Figure 2-43. Serial Adder (Simplified)



SAA	0	1	0	1	0	1	0	1
SAB	0	0	1	1	0	0	1	1
Bit Carry	0	0	0	1	0	0	0	1
Bit Transmit	0	1	1	1	0	1	1	1
Half-Sum	0	1	1	0	0	1	1	0
Carry In	0	0	0	0	1	1	1	1
Full-Sum	0	1	1	0	1	0	0	1

Figure 2-44. Half-Sum and Full-Sum Logic

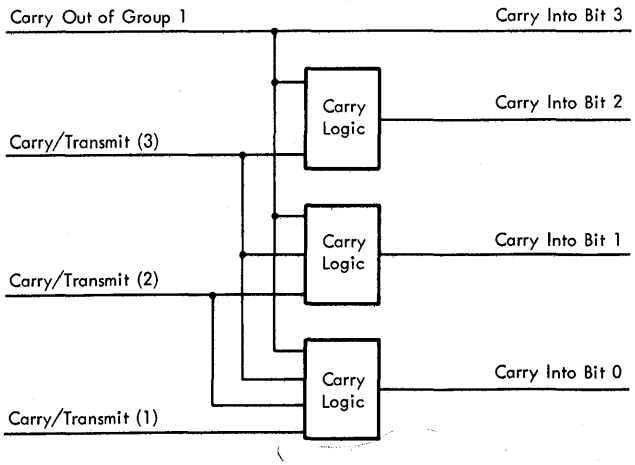


Figure 2-45. Carry Lookahead, Block Diagram

Binary Add

For binary add, data on SBA is entered in true or complement form and is combined with SBB data which may be 0's, forced bits, or data from ST. Combination takes place in the half-sum and full-sum logic with carry signals from the carry lookahead logic (Diagram 4-502, FEMDM).

Decimal Operation

- Excess-6 is provided in input logic to SAA.
- Decimal correction is made in set-SAL logic.
- Validity tests are made on input digits and signs.

The excess-6 operation for decimal instructions is implemented by logical circuits rather than by using extra adder cycles. The decimal character entering on SBA is increased by 6 as it is transferred to SAA (Diagram 4-503, FEMDM). Note that no time is lost in this operation; the circuits select the SAA positions which are 6 (0110) greater than the value on SBA.

The two operands are combined in the half-sum logic. If no group carry results, decimal correction is initiated by a ROS micro-order (Diagram 4-504, FEMDM). Decimal correction removes the excess-6 factor by using logical circuits to set SAL to a value 6 less than the full-sum value. Table 2-1 shows the decimal-corrected values for all possible erroneous characters. Again, because the circuits have been preconditioned, no cycles are lost and the decimal operation proceeds at full speed.

Table 2-1. Decimal Correction for Erroneous Numeric Characters

Group 1 Result					Group 1 Result Decimal-Corrected (Binary Position)			
Decimal	Binary Position 4	5	6	7	4	5	6	7
15*	1	1	1	1	1	0	0	1
14*	1	1	1	0	1	0	0	0
13**	1	1	0	1	0	1	1	1
12**	1	1	0	0	0	1	1	0
11***	1	0	1	1	0	1	0	1
10***	1	0	1	0	0	1	0	0
9****	1	0	0	1	0	0	1	1
8****	1	0	0	0	0	0	1	0
7*	0	1	1	1	0	0	0	1
6*	0	1	1	0	0	0	0	0
5 ↕ 0	Valid digits, no correction required							

* 1-bits in positions 5 and 6: reset positions 5 and 6.
** A 0 in position 6: set position 6 and reset position 4.
*** A 1-bit in position 6 and a 0 in position 5: set position 5 and reset position 4.
**** 0's in positions 5 and 6; set position 6.

The following is an example of decimal correction for SAL(0–3) using Diagram 4-504:

Uncorrected binary result to SAL(0–3) = 1011.
SAL(3): No correction is made. It is set to 1 per binary addition.
SAL(2): Requires carry plus half-sum, or no carry plus no half-sum (effective 0 result). Conditions are not met and SAL(2) is not set.
SAL(1): Requires effective 0 and full-sum (2). Conditions are met and SAL(1) is set.
SAL(0): Requires effective 1 plus full-sum (1 and 2). Conditions are not met and SAL(0) is not set.
Corrected result in SAL(0–3) = 0101.

Incoming data is examined for validity on decimal instructions. If either character on SBA or SBB exceeds a value of 9 (binary 1001), an 'invalid digit' signal is generated and STAT E is set (Diagram 4-505, FEMDM). At the same time, 1's are forced into SAL to yield correct parity for the number transferred to S.

The invalid digit logic is also used to test the sign character entering the serial adder. An 'invalid sign' signal is developed if the sign character does not have a value of 10–15 (binary 1010–1111).

For multiply operations, the product is sent from B(66,67) to selected pairs of SAL bits to accumulate a byte of data. On non-decimal divide operations bits are sent from A(4) and A(28) to selected SAL positions to accumulate a byte of data.

Logical Functions

Logical functions (AND, OR, and Exclusive-OR) are performed in the serial adder. These functions produce full-sum latch settings (carry information from adjacent positions is disregarded) as follows (Diagram 4-506, FEMDM):

1. AND. The combination of an active AND control signal and a bit-carry from the half-adder of that position.
2. OR. The combination of an active OR control signal with a bit-carry or a half-sum (in effect, a bit on either or both inputs) from the half-adder of that position.
3. Exclusive-OR. The combination of an active OER control signal with a half-sum (in effect, either input bit but not both) from the half-adder of that position.

Results are transferred to selected bytes in ST.

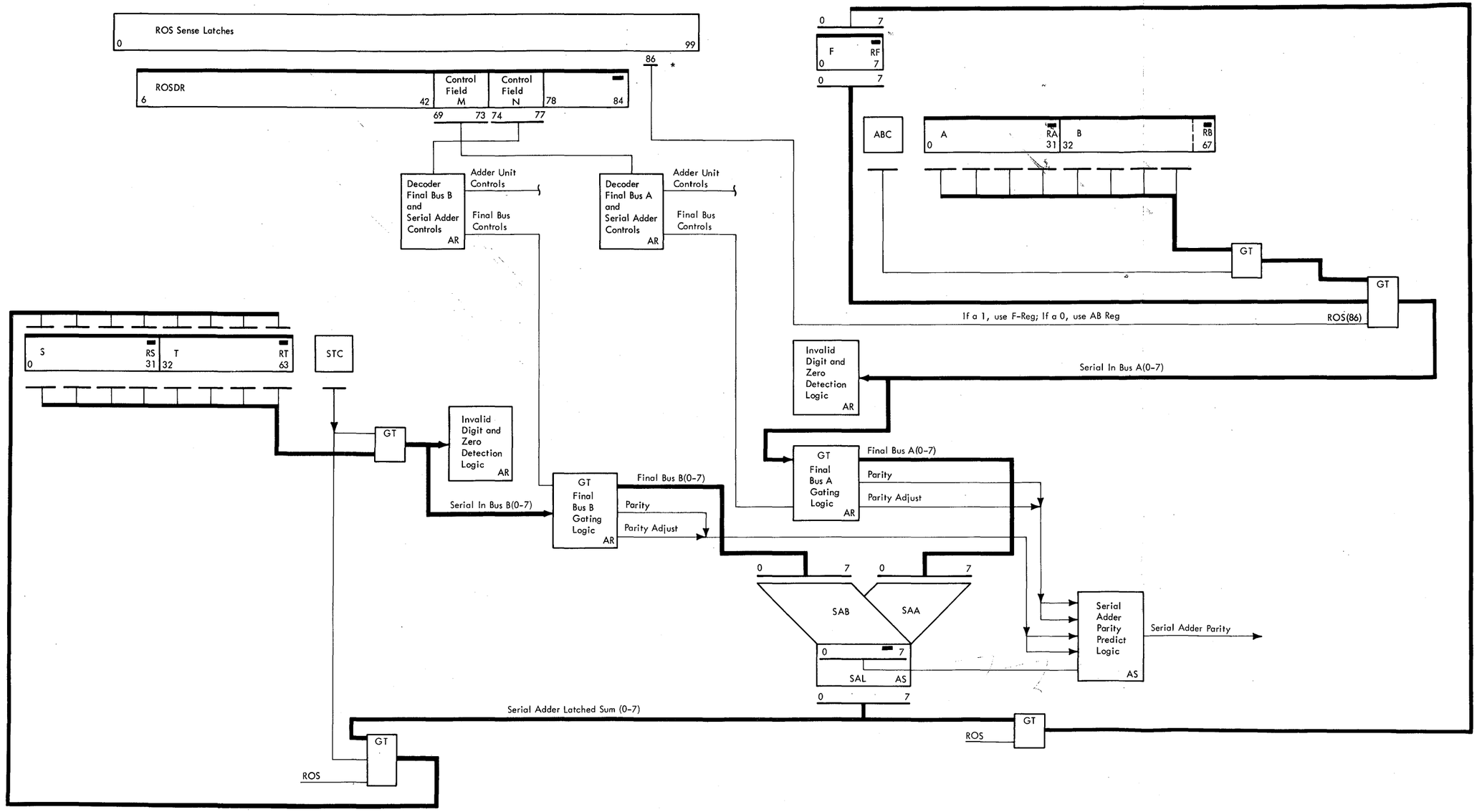
Parity Correction

- Parity bit is set if number of bits in SAL(0–3) and SAL(4–7) are both odd or both even.
- Additional logic predicts parity for decimal operations.
- Parity is reversed on multiply and divide operations if only one bit is sent to SAL.
- Logical operations develop parity through unique logic.

Correct (odd) parity for the serial adder outputs is generated in the parity predict logic (Diagram 4-507, FEMDM). There are two basic areas of parity generation: (1) arithmetic and (2) logical (AND, OR, and Exclusive-OR). The arithmetic parity generation is further divided into binary and decimal. See Figure 2-47, a block diagram of parity predict logic.

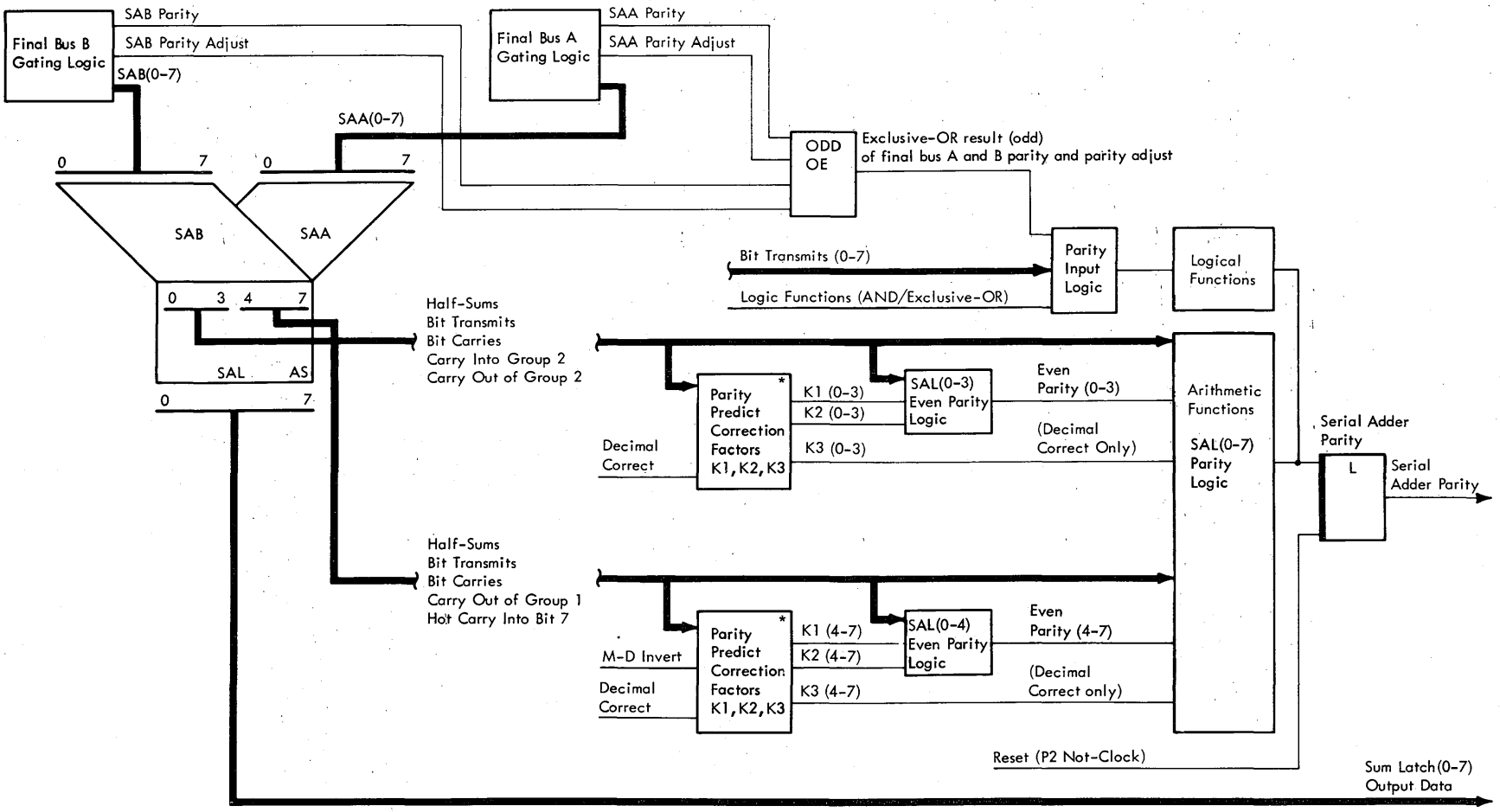
Binary parity predict logic includes factors K1 and K2, half-sum (0), and an odd or even number of transmit bits (1–3 and 5–7). K1 and K2 are established by carry and transmit bits (Diagram 4-507). The signal resulting from these factors reflects the odd or even number of bits in SAL(0–3) and in SAL(4–7). If both are odd or both are even, the serial adder parity latch is set. If only one is odd, the parity latch is not set.

The shaded area in Diagram 4-507 indicates the additional control (K3) used for decimal operations. A decimal correction must be set up (decimal operation and no group carry) to allow energizing of the K3 signal. An examination of carry and half-sum (1, 2, 5, 6) allows the prediction of parity after the excess-6 factor is subtracted from SAL.



* ROS control field R (bit 86) determines whether AB or F data is presented to final-bus-A gating logic.

Figure 2-46. Serial Adder Gating Controls



*K1, K2, and K3 are defined factors of serial adder half-sum, bit-transmit, bit-carry, group-carry, and carry-into-group functions. (K3 factor is activated during decimal-correct operations only.)

Figure 2-47. Serial Adder Parity Predict Logic

Multiply-divide operations present a different problem in that data is presented directly to SAL and is not processed through normal parity predict logic. A test is performed on the two partial product bits or the one quotient bit (Diagram 4-508, FEMDM). A 1-bit change causes an 'invert predicted parity' signal which energizes K2 (4–7). Because no carries are generated, K2 would not normally be energized; thus, the K2 signal inverts the predicted parity.

Logical operations disable arithmetic parity prediction and energize a different parity predict circuit. Three conditions are tested to set the SAL parity latch; (1) input parity, (2) parity adjust, and (3) odd-even transmit bits (Diagram 4-507). The development of each is as follows:

1. Parity. Normally presents the original parity (1 or 0) of the byte being entered on the input bus (one byte for SBA and one byte for SBB).
2. Parity-adjust. In effect, inverts the incoming parity regardless of the actual 1 or 0 parity. Parity-adjust is energized when a micro-order alters incoming data [for example, if SBA(0) enters as a 1 and a ROS micro-order forces SBA(0) to a 0, parity-adjust is energized].
3. Odd-even transmit bits. Exclusive-OR circuits analyze the developed transmit bits of all positions and produce signals denoting an odd or even number of transmit bits.

Parity generation for an OR function is based on the OR command and the OE transmit bits. Because only effective transmits are used to set SAL(0–7), parity generation needs only an even number of transmit bits to set the parity latch. The following is an example of parity generation for an OR function:

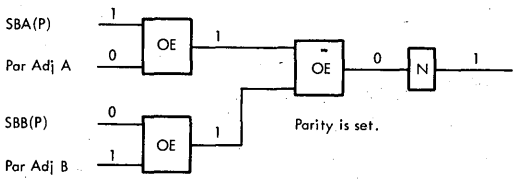
Bit	0	1	2	3	4	5	6	7	P	
SBA	1	0	1	0	0	0	1	0	0	
SBB	0	0	1	1	0	0	1	0	0	
Transmit	1	0	1	1	0	0	1	0		Even
SAL	1	0	1	1	0	0	1	0	1	

4 (even) transmit bits; Parity is set.

The Exclusive-OR function requires the parity and parity-adjust Exclusive-OR logic to generate correct parity. (Because the transmit bits do not reflect parity for Exclusive-OR, they are not used.) Regardless of the number of 1 bits, if the parity of the two input data bytes is different, the resultant will be an odd number of bits and the parity bit will not be set. An exception is caused by micro-order insertion of data over the byte data. This condition is corrected by means of the parity adjust circuits, as shown by the following example:

ROS Force 0

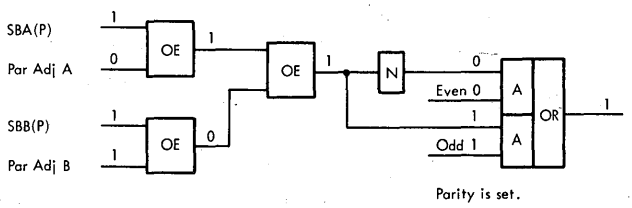
Bit	0	1	2	3	4	5	6	7	P
SBA	1	1	1	1	0	1	1	0	1
SBB	1	0	0	1	0	1	0	0	0
SAL	1	1	1	0	0	0	1	0	1



When the AND function is used, all three signals (SBA and SBB parity, SBA and SBB parity adjust, and odd-even transmit bits) are analyzed to generate parity. The serial adder parity latch is set when the result of the parity and parity-adjust signals matches the odd-even transmit bits signal, indicating an even number of bits have been generated; a parity bit is thus needed. An example of parity generation for a logical AND operation follows:

ROS Force 0

Bit	0	1	2	3	4	5	6	7	P	
SBA	0	0	1	0	1	1	0	1	1	
SBB	1	1	1	1	1	0	1	0	1	
Transmit	0	1	1	1	1	1	1	1		Odd
SAL	0	0	1	0	1	0	0	0	1	



Note: There are four special cases when the SBB input parity bit is set: (1) no-operation, when SBA is all 0's, (2) when -64 (1100 0000 binary) is forced, (3) when subtract 1 (1111 1111 binary) is forced, and (4) on partial product entry when SBB is zero. In addition, the SBB parity bit is held off on add 1 (0000 0001 binary) because it is incorrect parity.

When an invalid digit is detected on a decimal operation and all 1's are forced to SAL(0–7), the parity predict logic is bypassed, and the SAL parity latch is set to give proper parity for the byte sent to ST.

Error Detection

Serial adder data is parity-checked on both a half-sum and a full-sum basis. Error indications are retained in the half-sum error or full-sum error latches (Figure 2-48).

Half-sum error logic tests the incoming data for accuracy. The signals tested are (1) half-sum(0–7), and (2) input parity and parity-adjust for both A- and B-sides of the adder. When an input parity is in error (even parity), the combined signals produce an odd result and the 'half-sum error' trigger is set.

Full-sum error logic tests the accuracy of the final answer by combining the SAL outputs and the resulting state of the parity latch. An even result sets the 'full-sum error' trigger.

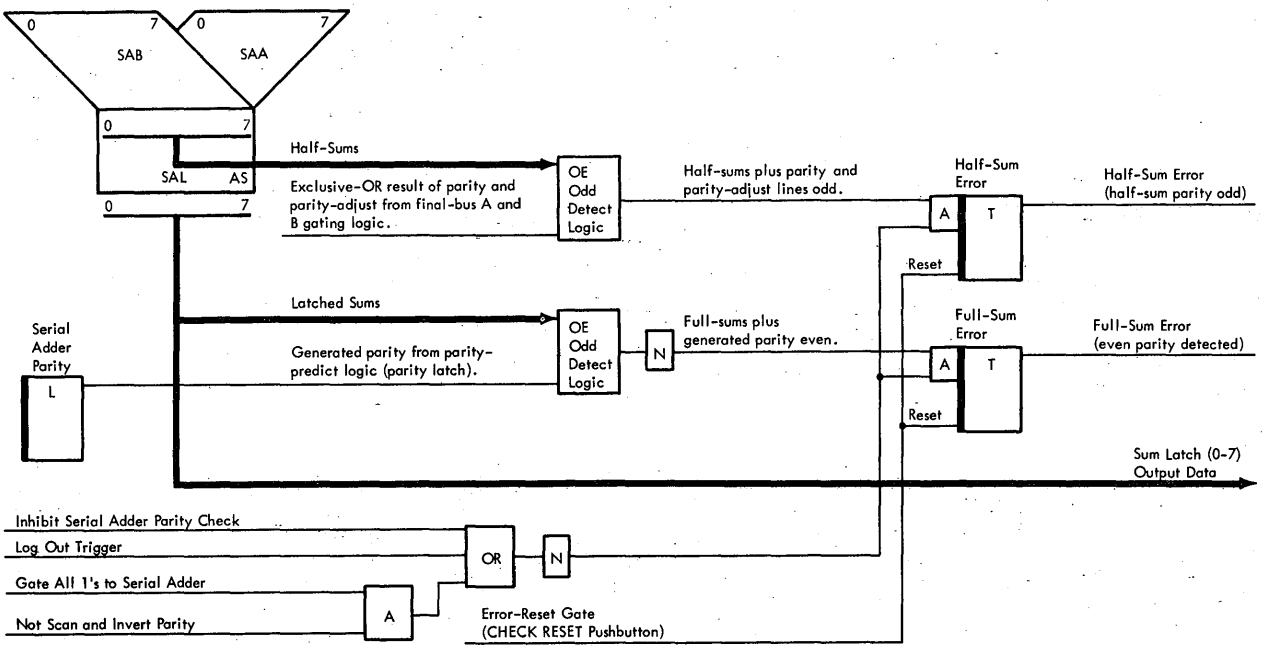


Figure 2-48. Half-Sum and Full-Sum Error Logic

The error triggers remain set until reset by the 'error reset gate' signal. To avoid meaningless error indications and subsequent logout operations, the set error trigger signals are blocked when:

1. The 'inhibit serial adder parity check' micro-order is active.
2. An invalid digit is detected during a decimal operation (all serial adder latches, including the parity latch, are set).
3. The 'logout' trigger is set.

PARALLEL ADDER

The parallel adder, 60 data bits plus parity, performs arithmetic and logical functions and is involved in most intra-CPU data transfers. Data flow for the parallel adder is shown in Figure 2-49.

The parallel adder bit positions are divided into sections and groups to implement carry lookahead and parity predict functions (Figure 2-50). Additional functions illustrated in the figure are half-sum, full-sum, and latch-shifter logic.

Data Input

Data is transferred to the parallel adder from various registers by means of input buses controlled by ROS micro-orders (Figure 2-51). More than one bus may enter the same side of the parallel adder, but only one bus is active at a given time.

ROS fields T and U control inputs to bus B and bus A, respectively. Gate control triggers for adding B and T are shown in Diagram 4-509, FEMDM. ROS sense latches are decoded at P2 time to select a gate control trigger; the trigger remains set until the following P1 time to make the data available as long as the parallel adder needs it. Note in Diagram 4-509 that only 3 of the 4 bits of the ROS fields are used; the state of the fourth bit does not affect the gate control triggers shown but does affect other gate control triggers.

Individual Bit-Position Logic

- Full-binary capabilities (half-adder and full-sum logic) are provided.
- Shift logic (latch-shifter) is included at the output.

The logic functions associated with each adder position are shown in block form in Figure 2-52. These functions (half-adder, carry-into-bit, full-sum, and latch-shifter) constitute the full-binary logic of each bit position; operation is as follows. The status of corresponding A- and B-side operand bits is entered into the half-adder, where they are combined to produce bit-transmits, bit-carries, and half-sums. The bit-transmit/bit-carry is sent to carry lookahead logic to produce predicted carry information, and the half-sum is sent to the full-sum logic. The carry-into-bit logic combines the immediately available bit-transmit/bit-carry from adjacent lower-order adder positions (representing an actual carry) with the somewhat later returning predicted carry. This predicted carry output (carry-into-bit) is also sent to the full-sum logic, where it is combined with the half-sum from the half-adder logic to generate a final full sum (1 or 0) for that adder position. This full

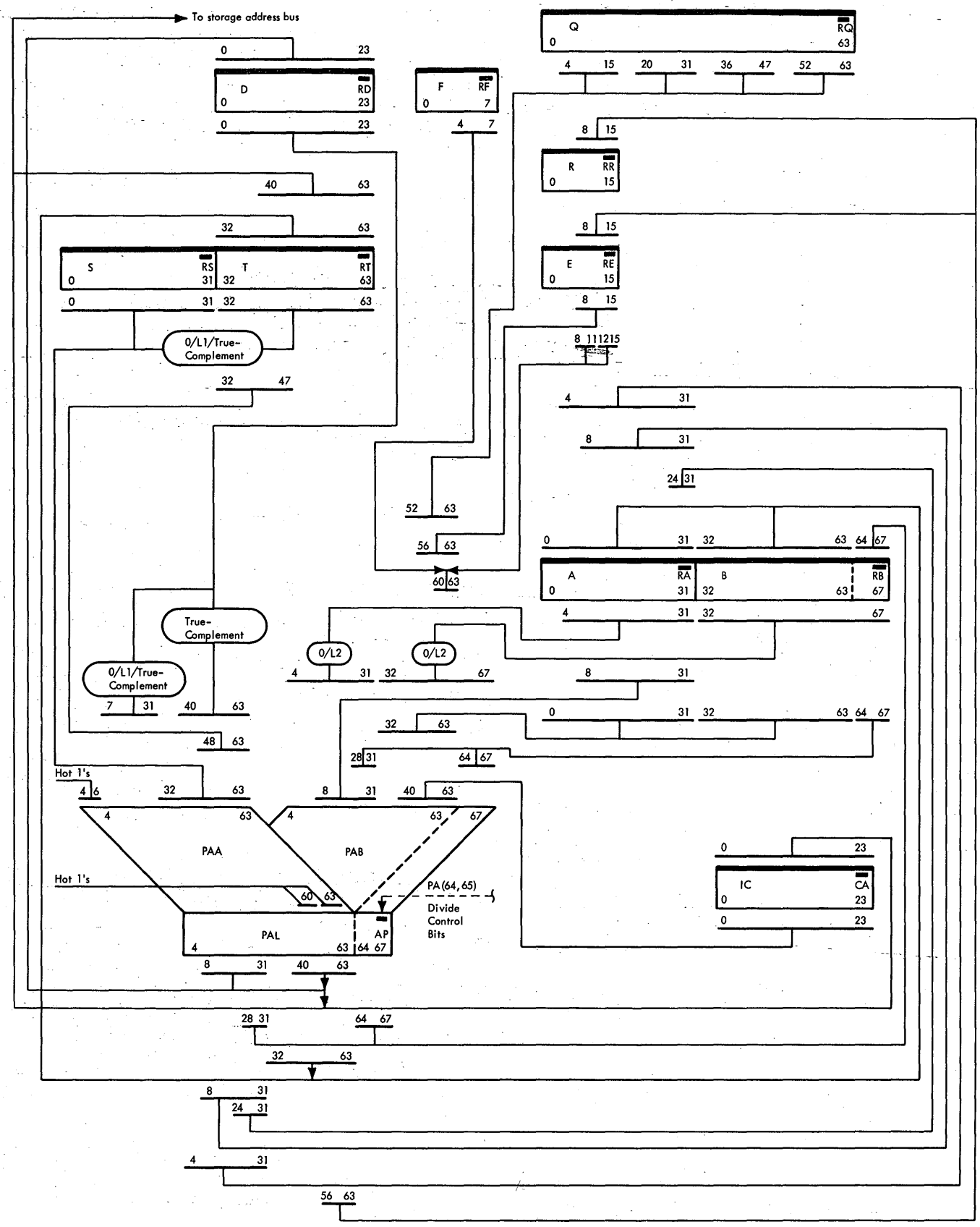


Figure 2-49. Parallel Adder Data Flow

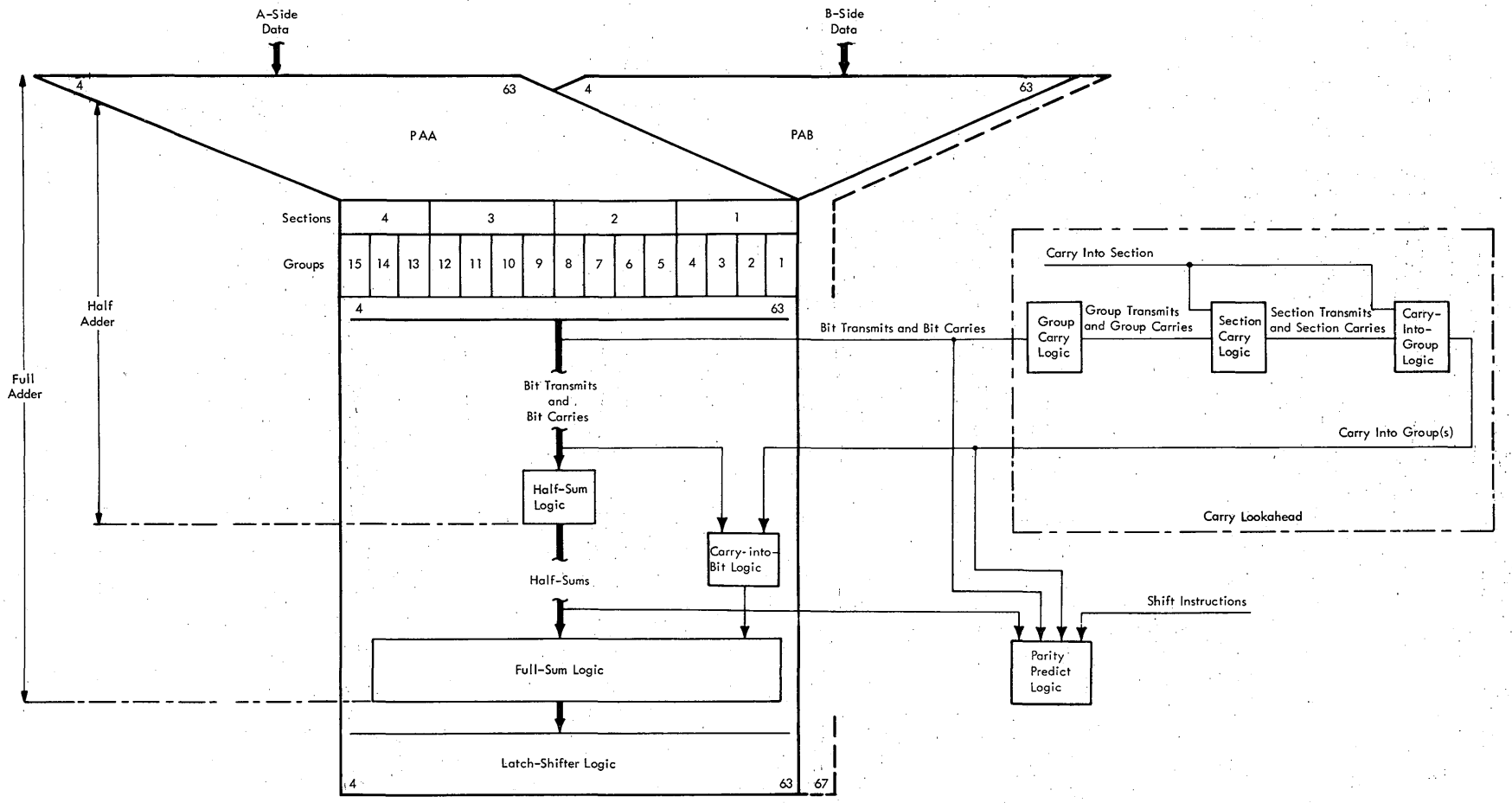


Figure 2-50. Parallel Adder Function Breakdown

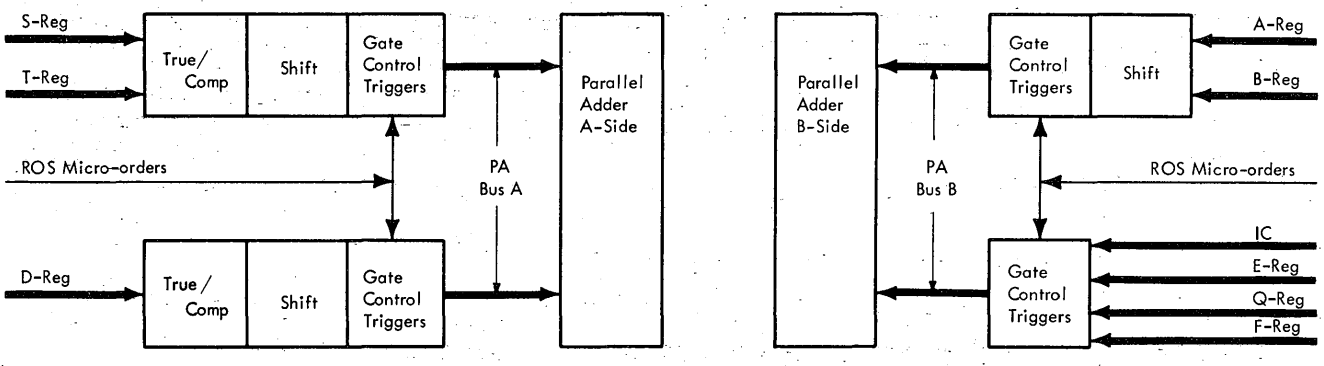


Figure 2-51. Parallel Adder Input Buses

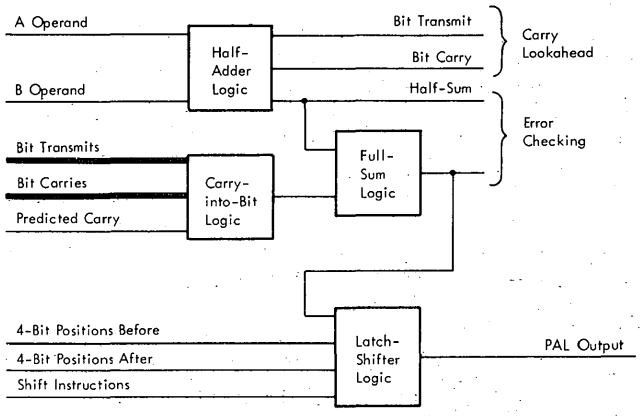


Figure 2-52. Bit Position Block Diagram

sum, or possibly the full sum from four positions to either the left or the right (depending on the particular shift control) is gated into the adder latch. Latched sum data is retained until the following cycle for sampling into the selected register(s). The following paragraphs discuss the logic involved in each function.

Half-Adder

The logic involved in bit-transmit, bit-carry, and half-sum functions are:

1. Bit-transmit. At least one *and possibly two* 1 bits are contained in the two corresponding A- and B-side operand positions.

   Note: For certain operations, the parallel adder is set to all 1's by a micro-order which forces all 1's into the A-side of the half-adder (Diagram 4-510, FEMDM).

2. Bit-carry. Two 1 bits are contained in the corresponding A- and B-side operand positions.
3. Half-sum. A single 1 bit, but not two, is contained in the corresponding A- and B-side operand positions.

Carry-into-Bit Logic

- Detects "carry-into" conditions affecting each particular bit position.

The carry-into-bit logic of each adder position detects whether a carry-into condition prevails, resulting from either an actual carry or a predicted carry. Carry-into-bit circuitry logically OR's the actual carry (prevailing carry conditions from immediately adjacent lower-order positions) with predicted carry (carry-into-group indications from lookahead logic, signifying that effective carry conditions exist in the more extreme lower-order areas).

The actual carry into any particular adder position is determined by logically testing all remaining lower-order bit positions within that same group or, if the particular adder position happens to be the low-order group position, testing all four bit positions of the next lower-order group. (Predicted carries are discussed in "Carry Lookahead".)

Note: Either an actual carry (from adjacent positions) or a predicted carry (from carry lookahead) is allowed to affect a particular position, but not both. Where both carries occur, conditions producing the actual carry also function to inhibit the predicted carry from entering the affected bit position(s). This inhibit logic is illustrated in Diagram 4-510, FEMDM, as follows: Carry conditions from positions 48–51 generate both a predicted carry to position 47 (carry-into-group 5), via carry lookahead logic, and an actual carry in the form of bit-transmit/bit-carry signals. Because group 4 positions (48–51) represent the actual carry source, the group-4-carry condition is then inverted to inhibit predicted carry entries into position 47.

Full-Sum Logic

The full-sum logic for any particular adder position combines (by means of an Exclusive-OR) the carry-into-bit output with the half-sum output of the half-adder, developing a 1 or 0 full-sum for that adder position.

Latch-Shifter Logic

Latch-shifter logic facilitates the left 4/right 4 shifting of the full sum during the same cycle in which it is

developed. (Logical and data-transfer operations also utilize this logic.) For any particular adder position, zero-shift, left-4 shift, and right-4 shift controls respectively gate the full sum into the latch associated with that position, into the latch associated with the position four places to the left, and into the latch associated with the position four places to the right. (Scan-out operations also utilize the latch-shifter logic but only for its data path facilities.) All latches retain the latched sum until the following cycle, and extended-clock signals delay resetting the latches long enough for the error-checking logic to function.

Note: An adder-hold ('→HOLD') micro-order, used during certain operations, blocks the extended-clock reset signal and causes the latches to retain their data for one additional cycle.

Carry Lookahead

- Predicts carry before full-sum development.
- Reduces time required to provide full sum.
- Lookahead logic divides 60-position adder into 15 four-bit groups, and these groups into four sections.
- Lookahead information is developed in form of bit-position carry, group carry, and section carry, and then fed back into individual positions as predicted carries.

The carry lookahead function provides the adder with the capability of entering full sums directly into the adder latches. Lookahead functions effectively predict the carry resulting from combining two operands, and use this predicted carry to convert half-sums to full sums before the entry of information into the adder latches. This sequence eliminates the additional time required by ripple operations, which would be necessary in converting half-sums to full sums if half-sums were entered directly into the latches.

For design reasons, the 60-position adder is divided into 15 four-position groups, and these groups are subdivided into four sections. This group/section arrangement reduces the logic decoding required in implementing the carry lookahead functions. Group/section arrangement and carry lookahead data flow are shown in Figure 2-53.

Lookahead logic is designed so that, for any particular position, the effective carry conditions in all lower-order positions (except for an adjacent few) are logically predicted for that position. Carry conditions which would later be produced in these same adjacent few positions as a result of propagated lower-order carries are predetermined by the lookahead functions and logically entered into that position as a predicted carry. Using this method, each bit position then requires only that logic necessary to detect prevailing carries (actual carry) in the adjacent positions, and to logically OR the actual and predicted carries when developing the full sum. Predicted carries are presented to the input logic of individual positions as 'carry into group' signals. Figure 2-54 illustrates the adder areas supplying source information for actual and predicted carry signals to adder position 44. Note that, although lower-order carry conditions exist in both examples, they are represented by an actual carry in example 1 and by a predicted carry (carry-into-group-5) in example 2. (Recall also, from the previous discussions, that where both actual and predicted carries are generated to a particular position, only the actual carry is entered; the predicted carry entry is blocked.)

In the lookahead logic, predicted carry information is developed by testing each adder group for bit-position carry conditions, combining these conditions to form group-carry conditions, and then similarly combining the group-carry indications to produce section-carry conditions. All lower-order carry conditions affecting any individual adder position (with the exception of the positions immediately adjacent to that position) are then collectively represented to the lookahead logic as section-level carry information. Section-carry information from each section is then (after being combined with lower-order section-carry indications) sent to higher-order sections, where it is combined with the group-carry conditions within these sections, to produce 'carry into group' signals. As previously described, these 'carry into group' (predicted-carry) signals are then logically OR'ed with actual carries within the carry-into-bit logic of each individual adder position, and combined with half-sum information to generate a full sum. (Recall also that, where both actual and predicted carries are generated to a particular position, only the actual carry is entered and the predicted carry entry is blocked.)

The following paragraphs give detailed descriptions of group-level and section-level carry-predict functions.

Group-Level Carry Logic

- Bit-position carry conditions are combined in four-bit groups to generate group-carry conditions.
- Group-carry logic outputs define effective status of all bit positions composing a group.
- Group-carry outputs are sent to section-carry logic.

The group-level carry information generated for any particular group is determined by logically combining the bit-transmit/bit-carry outputs of all four positions within that group. Group-transmit/group-carry signals are then

generated and sent to the section-level carry logic of the section in which the particular group is located. Group-level carry logic outputs indicate the following:

1. Group transmit. Signifies that all bit positions within that group have received at least one bit of operand data, i.e., bit-transmit conditions exist throughout the group.
2. Group carry. Signifies that bit-transmit/bit-carry conditions within that group are such that an effective carry condition exists from the high-order position of that group.

Group-level carry logic is illustrated in Diagram 4-511, FEMDM. Note in the diagram that group-transmit/group-carry outputs are determined solely by bit-transmit/bit-carry conditions, which represent incoming data only (without the use of any propagated carry information).

When group-carry conditions are sent to their associated section-level carry logic, they may also (at the same time) generate 'carry into group' signals to adjacent higher-order groups within that same section. This sequence results in the immediate propagation of group-carry information (within that same section). Carry-into-group circuits that are not activated at this particular time may be activated somewhat later by incoming section-carry signals from lower-order sections.

Because group-carry conditions are used in developing section-carry conditions, a time differential exists between the two logic functions. The timing relationships between bit-carry, group-carry, section-carry, and carry-into-group are discussed in "Arithmetic Function Sequence".

Section-Level Carry Logic

- Group-level carry conditions are combined to develop section-level carry conditions.
- Section-carry outputs define effective carry conditions of all bit positions within a particular section.
- Section-carry outputs are sent to higher-order sections as predicted carry information.

Section-level carry information is determined by logically combining the group-transmit/group-carry outputs of all groups within a section. Like group-carry generation, section-carry outputs are also determined solely by group-carry conditions (without the use of any carry propagation). Section-level carry logic outputs signify the following:

1. Section-transmit. Indicates that all bit positions of all groups within that section have received at least one bit of operand data (i.e., group-transmit conditions exist throughout the section).
2. Section-carry. Signifies that group-transmit/group-carry conditions within that section indicate that an effective carry condition exists in the high-order bit position of that section.

The section-level carry logic (Diagram 4-511) develops a 'section 1 transmit' signal from group transmits and a 'section 1 carry' signal from combinations of group-transmits and carries.

Section-Level Carry-Into Logic

A section-carry generates a carry into the next higher order section. The section-level-carry-into logic (Diagram 4-511) develops carry-into-section signals, starting with a 'carry into section 1' signal produced by a hot-carry. Carry into sections 2, 3, and 4 are developed by section-transmit and section-carry logic.

Group-Level Carry-Into Logic

Carry-into-section signals produce a carry-into-group signal for at least the low-order group of the section. Development of additional carry-into-group signals is dependent on group carry/transmit conditions (Diagram 4-511). Note that carry-into-group signals may be developed independently from the carry-into-section signals.

Bit-Level Carry-Into Logic

Carry lookahead conditions the bit-level carry-into logic of the low-order group position if no group carry is present from the next lower-order group. Other bits in the group are conditioned if intervening low-order transmit bits are present (Diagram 4-511). For example, if a bit 49 carry and a bit 48 transmit have been developed, the result is a group 4 carry that generates a section 1 carry. The section 1 carry and a section 2 transmit produce a 'carry into section 3' signal. A 'carry into group 9' signal results from the 'carry into section 3' signal. When bit-transmits 30 and 31 are present, a carry into bits 29 and 30 takes place to develop full sums.

Diagram 4-511 shows the timing relationships for carry lookahead. Note that although a direct carry occurs before a carry lookahead, this time difference does not affect the final sum development which takes place after all carry circuits have settled down.

Full-Sum Development

- Half-sums are combined with carry information (actual and predicted) to develop full sum.

The manner in which carry lookahead and half-sum functions are logically combined to produce a full-sum result is illustrated in Figure 2-55. Note that all group-level and section-level functions are arranged on a section (four-section) basis, whereas carry-into-bit and full-sum functions appear in each adder position.

The complete carry lookahead system is shown in Diagram 4-511; a summary of carry-predict operation is as follows. When operand data is presented to the A- and B-sides of the adder, the half-adders of all positions are

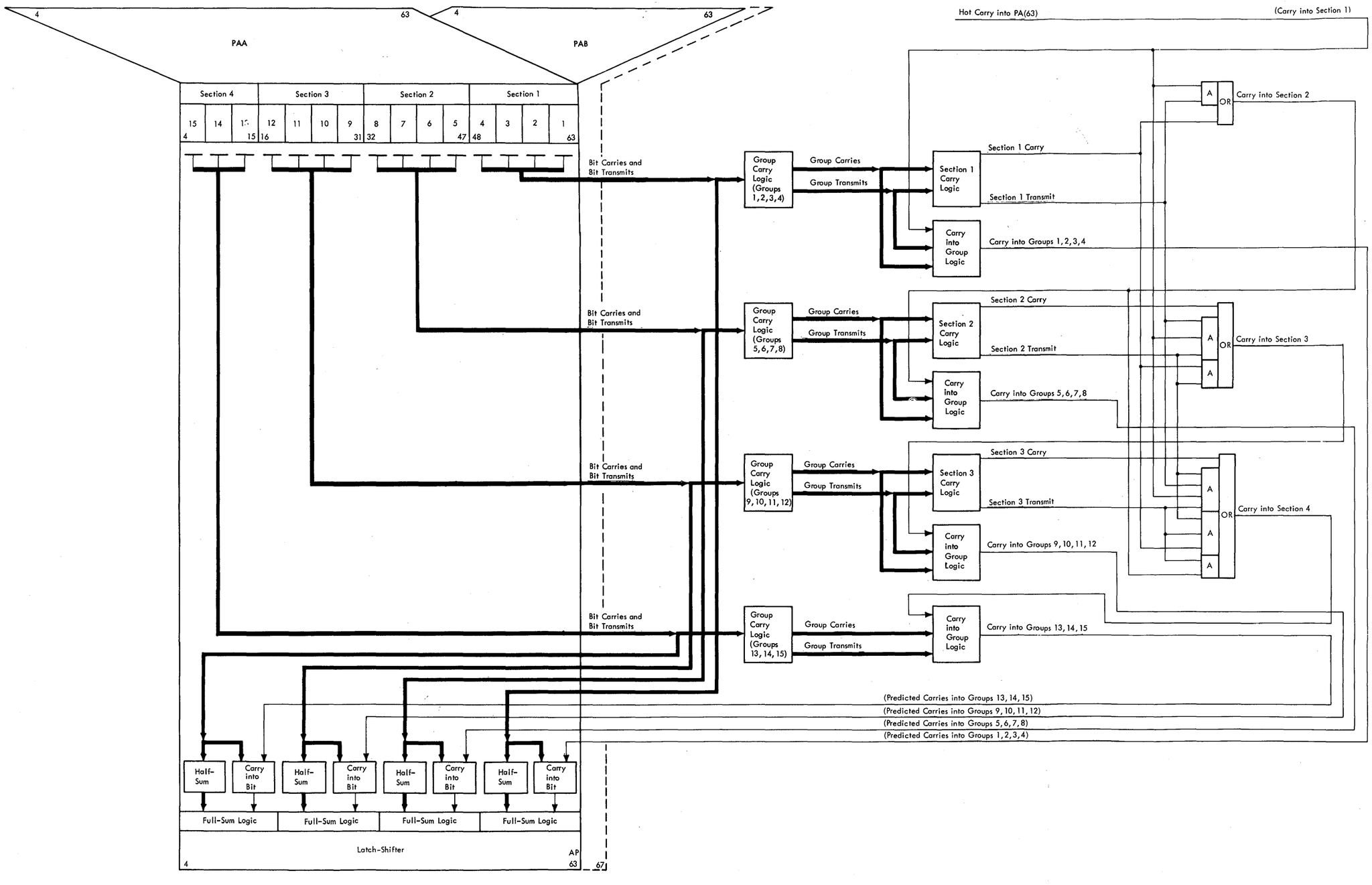


Figure 2-53. Parallel Adder Carry Lookahead Data Flow

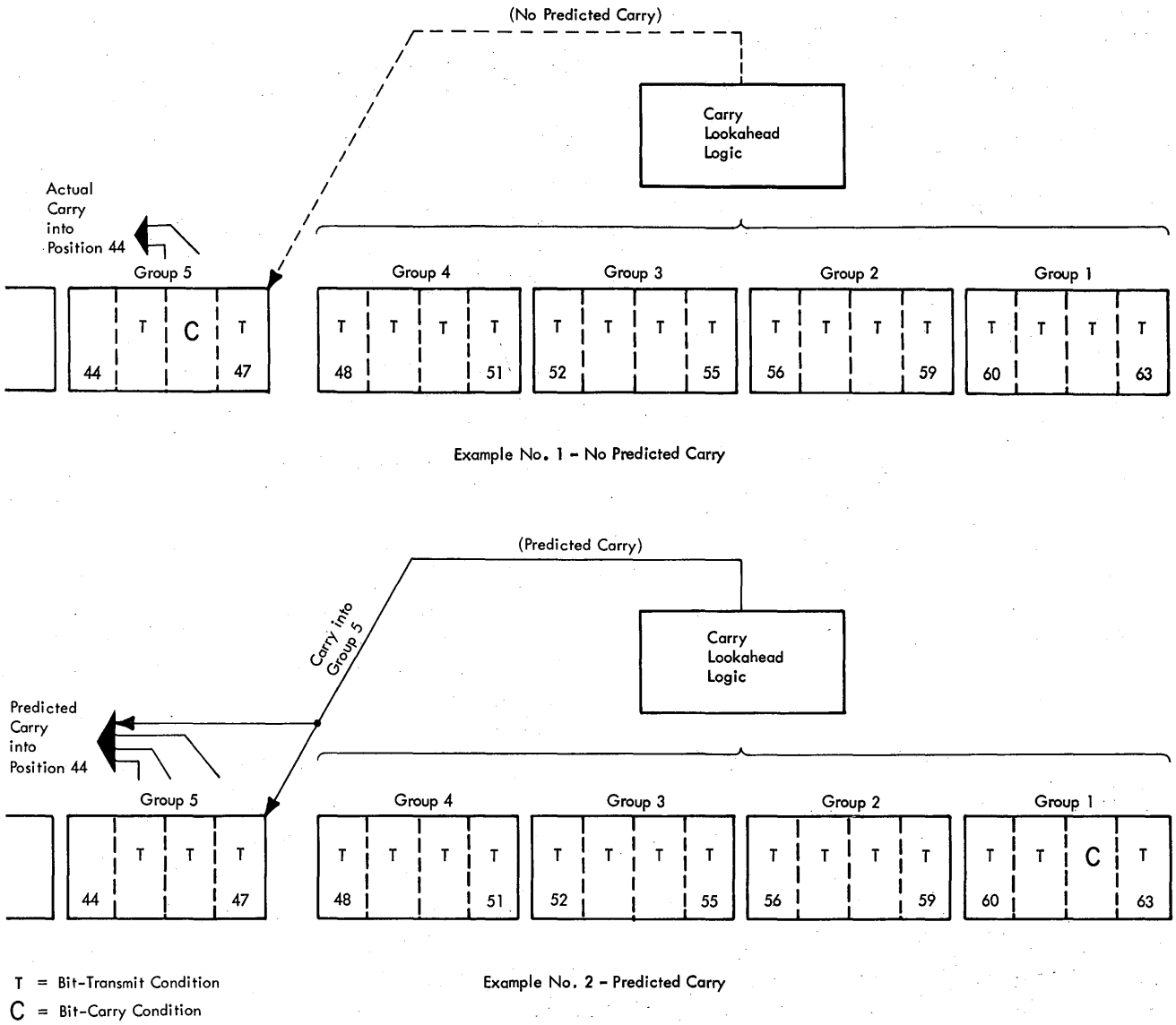


Figure 2-54. Actual and Predicted Carry Origin for PA(44)

sampled for bit-transmit/bit-carry information. (Half-sums are also generated from the half-adders and presented to the full-sum logic of each position at this time.) All bit-transmit/bit-carry information is sent to the associated group-carry logic, and all group-carry outputs are entered into their respective section-carry logic. Section-carry outputs now represent the carry status that logically prevails in the high-order position of each section (without any effects of carry propagation). All section-level carry outputs are then combined with lower-order section-carry information to determine whether a 'carry into section' (predicted-carry) signal is generated for the higher-order section(s). 'Carry into section' signals sent to higher-order sections combine with group-carry conditions within those sections to produce the carry-into-group conditions that represent predicted carries for the individual bit positions. The carry-into-bit logic of each individual position then logically OR's 'carry into group' (predicted-carry) signals with actual carry indications, and this output combines with the half-sum to produce the full-sum result.

Note that throughout the lookahead sequence no ripple operations are required. Definite cycle times, however, are associated with each predict function (bit-carry, group-carry, section-carry, carry-into-section, and carry-into-group); these times are discussed in the following paragraph.

Arithmetic Function Sequence

- Eight logical delay levels are required for arithmetic functions.
- Extended clock signals are used within adder.
- Full-sum results are latched (retained) for 1 cycle.

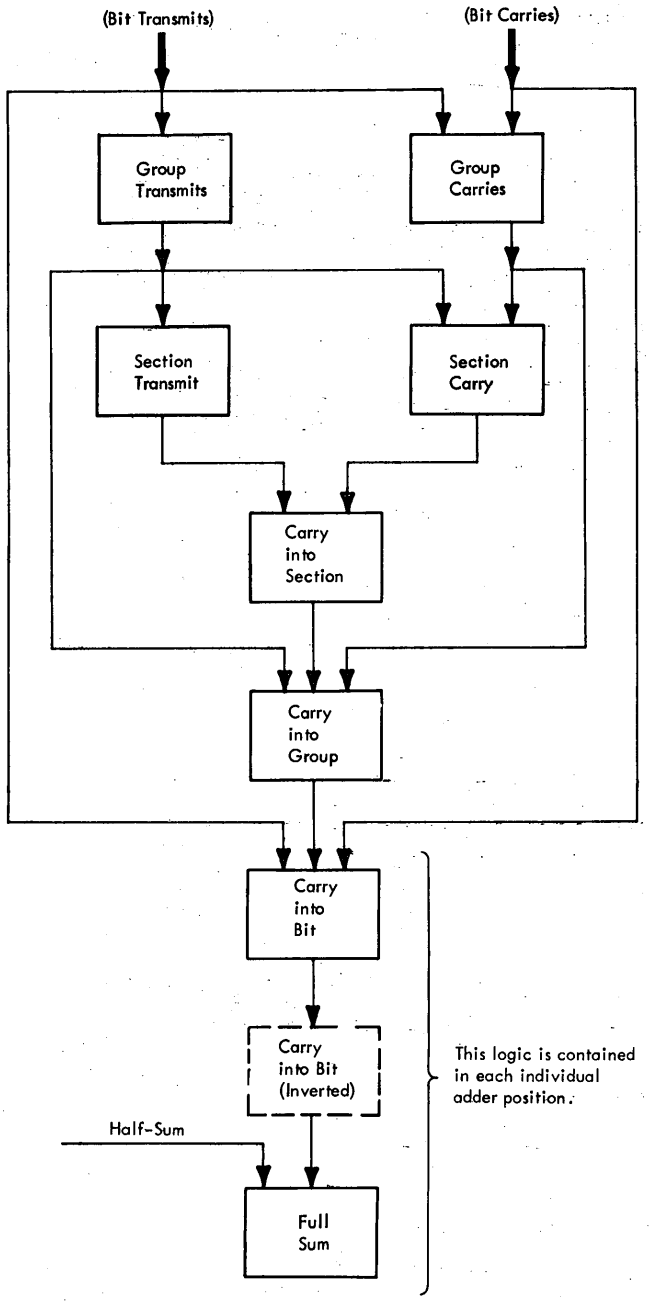


Figure 2-55. Full-Sum Development Logic

The timing sequence in which all adder logic operates to develop and check full-sum information is shown in Figure 2-56. Three delay levels (P4–P7) occur between the time at which data is placed on the adder input bus (by the associated gate-control triggers) and the time at which the same data enters the half-adders. (Two of these delay levels result from bus-gating delays; the third, from the signal cables.) Eight levels of signal delay, then, are required within the adder for the full-sum development process. As noted on the timing chart in Figure 2-56, the logic functions that require the eight delay times occur in the following sequence:

1. Bit-carry/transmit.
2. Group-carry/transmit; carry-into-bit (direct).
3. Section-carry/transmit.
4. Carry-into-section.
5. Carry-into-group.
6. Carry-into-bit (predicted).
7. Carry-into-bit (inverted).
8. Full-sum.

Note: A carry-into-bit can originate early from a direct carry or late from the predicted carry logic.

Extended clock signals are used within the parallel adder to control all latches. The clock portion of the normal CPU clock signal is extended two delay levels (approximately 20 ns), producing a symmetrical clock signal of 100-ns clock and not-clock times. These extended clock signals result in delaying both the setting and resetting of the adder latches. Delaying the setting of the full-sum latches provides additional time for carry-predict functions, and delaying the resetting of the latches retains latched sum information long enough for sampling by the error-checking logic.

Full-sum information contained in the adder latches is normally retained one cycle. For certain operations, however, an adder-hold ('→HOLD') micro-order inhibits the clock signal that resets the adder latches, thus retaining latched sum information for one additional cycle.

Parity-Predict Logic

- Odd parity is supplied with each byte (or half-byte) of adder output data.
- Parity generation is simultaneous with full-sum development.
- Parity-predict logic utilizes inputs from half-adders and carry lookahead logic.
- Parity generation is corrected accordingly for left-4/right-4 data shifting.

Odd parity is generated for each byte (or half-byte in the case of positions 4–7 and 64–67). Predict logic is employed, allowing parity information to be generated simultaneously with the development of full-sum data. (This scheme eliminates the time involved in analyzing the full-sum bit count to determine parity.) Parity is initially predicted for each four-bit group of adder output data. For the eight-bit byte outputs, the parity information predicted for the two adjacent four-bit groups that constitute a particular byte is combined (Exclusive-OR'ed) to determine the full-sum parity of that byte. Because the adder is also capable of shifting full-sum data

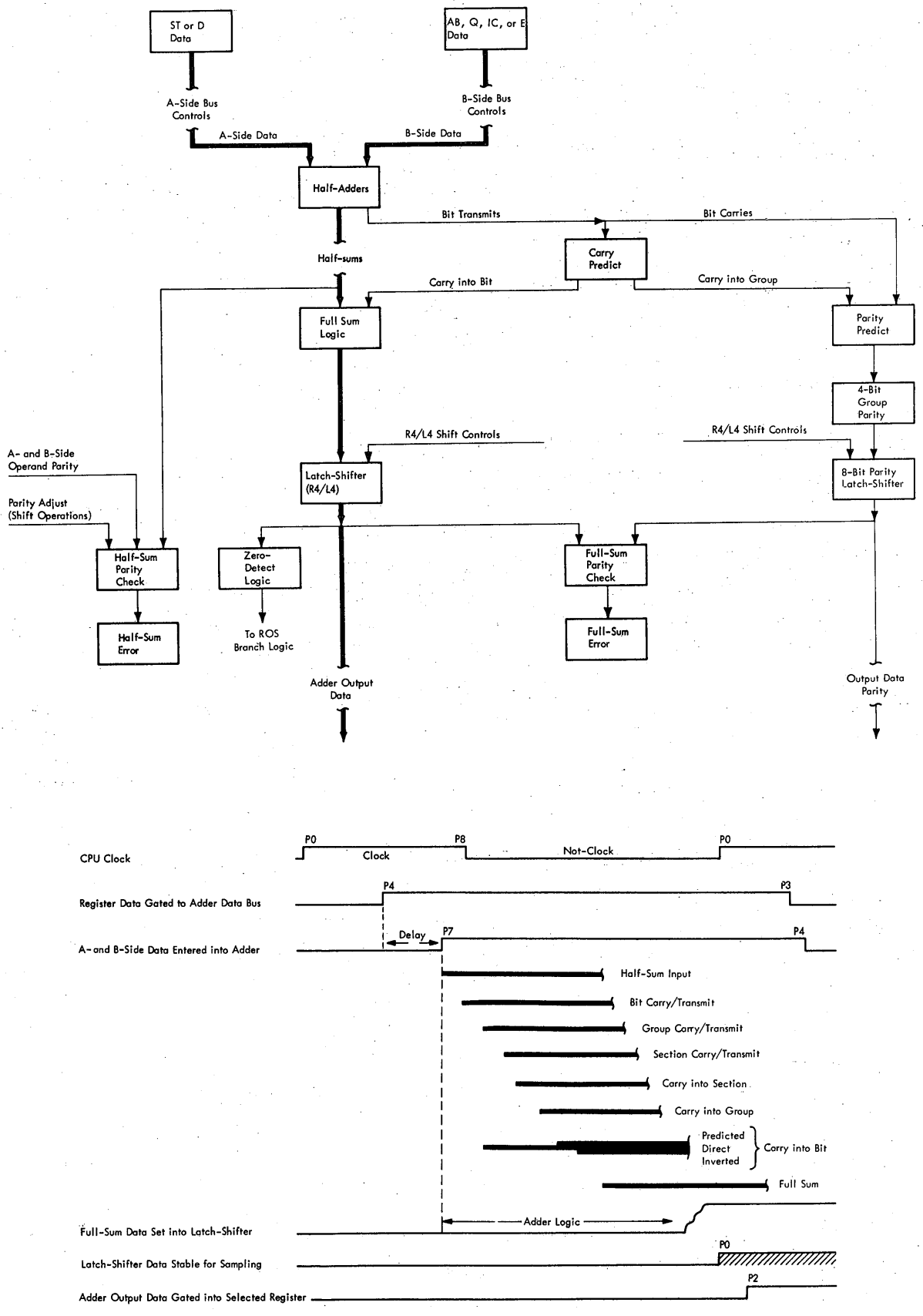


Figure 2-56. Parallel Adder Logic Function Sequence

left 4 and right 4 (before entry into the adder latches), generation of correct byte parity for left-4/right-4 operations then becomes a matter of selecting which two adjacent four-bit group parity outputs to combine when determining the parity of a particular output byte.

Parity is logically predicted through functions of the incoming operand data; operation is as follows. At the same time half-adder outputs are sent to the lookahead logic (to predict carry information), they are also sent to four-bit group parity-predict logic. A typical four-bit group parity-predict function is shown in Diagram 4-512, A, FEMDM. (Group 4 is used as an example; all groups are similar.) For each four-bit group, bit-transmit, bit-carry, and half-sum outputs from half adders and carry-into-group outputs from the lookahead logic are combined to logically predict whether the resultant full-sum bit count for that particular group will be odd or even. Note in the diagram that duplicate decoder logic is present in each four-bit group parity-predict circuit. This duplicate logic simultaneously produces the opposite polarity (out-of-phase) signals required for use in the eight-bit parity latch-shifter logic without the signal delay introduced if an additional inversion component were used.

Typical parity latch-shifter logic used in combining two adjacent four-bit group parities to determine eight-bit byte parity is shown in Diagram 4-512, B. (Adder output byte 48–55 is used as an example; all byte parity logic is similar.) Note in the diagram that the two four-bit group parity outputs to be combined (Exclusive-OR'ed) when determining byte parity are selected according to the type of shift operation in process; i.e., left 4, right 4, or no shift (straight transfer). The generated parity for each adder byte (or half-byte in the case of positions 4–7 and 64–67) is set into the corresponding parity latches for transfer with the data and sent to the full-sum error-checking logic. Because parity information is used in full-sum error checking and both parity and full-sum information are formed independently, an inconsistency in either will cause a full-sum error.

Error Checking

Parallel adder logic employs both half-sum and full-sum checking facilities. Half-sum checking verifies incoming data (in regard to assigned parity only); this test also results in verifying half-adder operations because half-sum outputs are used in half-sum checking logic. Full-sum checking logic compares the full-sum bit count (odd/even) with the generated parity information on a byte (or half-byte) basis. Because full-sum and parity information are formed independently, an inconsistency in either results in a full-sum error.

Half-Sum Checking

- Compares half-sums with incoming operand parity.

Half-sum checking logic combines the parity information assigned to incoming A- and B-side operand data with the half-sum generated when the same two operands are combined in the half-adders. This combining of parity and half-sums is performed on a byte (or half-byte) basis, with detected errors stopping the CPU clock and lighting indicators signifying the byte (or half-byte) in error.

Half-sum checking logic, illustrated in Diagram 4-513, FEMDM, operates as follows. A stage of precheck logic for each byte (or half-byte) combines half-sums with the corresponding A- and B-side parity information in odd-detect (Exclusive-OR) circuits. (The precheck logic shown in Diagram 4-513 monitors adder positions 48–55.) This logic functions so that, if the number of half-sums, plus the A- and B-side parity bits, results in an odd bit count, the half-sum precheck trigger for that associated adder area is set. Precheck outputs from all adder areas are then combined with left-shift logic at the input to the 'half-sum error' trigger. This left-shift logic determines whether an actual half-sum error exists or whether shifting the register data left 1 or left 2 positions (while en route to the adder) has forced a half-sum error.

Note: Left-shifting the adder input data left 1 or left 2 positions invalidates the assigned parity information, thus forcing half-sum errors. The number of half-sum errors created, however, should result in an even number; i.e., the half-sum errors forced into positions 4–31 should equal the number of half-sum errors forced into positions 32–63. Odd-detect logic, therefore, allows only an odd number of half-sum precheck indications to set the 'half-sum error' trigger during a left 1/left 2 shift operation.

If a valid half-sum error exists, the 'half-sum error' trigger is set, thus setting the 'final error' latch. Setting the 'final error' latch prevents the 'half-sum error' trigger from automatically resetting, which in turn prevents resetting the precheck logic for the area in which the half-sum error occurred. Inhibiting these resets causes the CPU program to stop on the following cycle (provided the CPU CHECK switch is in the STOP position), with the HALF SUM error indicator for the area incurring the error displayed on the roller switch indicators.

The half-sum error indications are reset by the 'error reset gate' signal (SYSTEM RESET or CHECK RESET pushbutton).

Full-Sum Checking

- Compares latched sum information with generated parity. ON A BYTE BASIS

Full-sum checking logic combines latched sum information with generated parity information on a byte (or half-byte) basis. (Full-sum checking for adder positions

48–55 is shown in Diagram 4-514, FEMDM.) Because the CPU operates with odd parity, combining full-sum bits with generated parity should always result in an odd bit count. Detecting an even latched-sum-plus-parity bit count sets the 'full-sum error' trigger for that particular adder byte area, which in turn sets the 'final error' latch. The error signal that sets the 'full-sum error' trigger stops the CPU program on the following cycle (provided the CPU CHECK switch is in the STOP position) and lights a FULL-SUM error indicator on the roller switch indicators, signifying the area incurring the full-sum error.

For practical reasons, combining full sums with parity is logically accomplished by first Exclusive-OR'ing the generated parity with a single latched sum position (Diagram 4-514), and then combining that result with the remaining latched sums of that particular byte. An odd result (signifying an *even* overall bit count) then sets the associated 'full-sum error' trigger.

Full-sum error conditions are reset by the 'error reset gate' signal (SYSTEM RESET or CHECK RESET push-button).

Convert-to-Decimal Operation

Special circuits, used only in the convert-to-decimal operation, provide excess-6 decimal correction when required. Excess-6 is forced on the PAB bus when a test of a four-bit group indicates a decimal value higher than 9. For this operation, parallel adder bit positions 28–63 are logically divided into four bit groups, each group representing a decimal digit in the packed format. Diagram 4-515, FEMDM, shows the development of excess-6 signals for PAB(28–31) and PAB(60–63). Note that these signals are activated only when ROS has developed the excess-6 gate and when the AB bits indicate the need for decimal correction.

For this operation, data is brought into the parallel adder one bit at a time by transferring one byte of data to the serial adder and sampling SAL(0). If SAL(0) = 1, the 'conv dec' trigger is set and a hot-carry sets PAL(63) (Fig. 2-57). The contents of the serial adder are then shifted left 1 position so that the next bit can be sent to the parallel adder (which is also shifted left one position). [For details, see Chapter 3, Section 2, "Convert to Decimal, CVD (4E)".] Because data is processed through normal parallel adder entry logic, parity generation takes place in a normal manner.

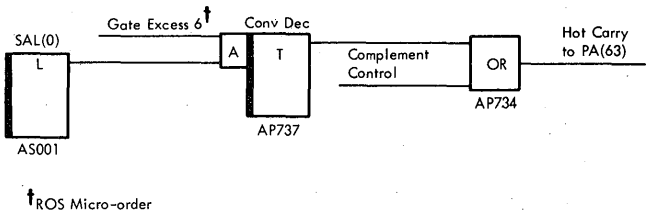


Figure 2-57. Convert-to-Decimal Data Flow to Parallel Adder

Set Condition Code

After an operation, PAL is analyzed to set the PSW condition code (CC). CC's are set in many ways with many variables for different instructions; Diagram 4-516, FEMDM, shows a typical example. Various sections of PAL are examined for a zero condition; combinations of PAL equal zero and micro-orders set STAT A, which is sampled by the instruction and the result sign to set the CC.

In the example, if PAL(32–63) is not equal to zero and the result is negative, a CC of 1 is set on a fixed-point operation. (This setting indicates a number less than zero.) Note that an overflow condition on a fixed-point instruction sets both CC bits, regardless of the condition of PAL.

For a floating-point operation PAL(7–67) is examined for zero. A not-zero condition and a plus result set a CC of 2.

www.IBM360.info

This section discusses the eight Status Triggers (STAT's A–H) and miscellaneous control triggers. A summary of the conditions that set STAT's A–H is shown in Figure 2-58, and a typical example of STAT logic (showing STAT B) is illustrated in Diagram 4-601, FEMDM.

STAT A

- STAT A indicates:
  Zero condition for parallel adder.
  Non-zero condition for serial adder.
  Digit condition on edit operations.

STAT A primarily indicates zero-detect conditions. Except during scan-in, when it is set directly to the value of T(54), STAT A is normally set at P2 clock time by one of the following conditions:

1. 'Set STAA if SAL(0–7) not equal to zero' signal, with SAL(0–7) not containing all zeros.
2. 'Edit set STAA' signal (edit operations).
3. 'Serial adder (0–3 or 4–7) not zero' signal, from the 'serial adder not zero' (SNZ) latch (indicating that the serial adder latched outputs do not contain all zeros).

   Note: The 'SNZ' latch is set at not-clock time by the 'decimal correct 0–3 set STAT's AE', or 'serial carry-7 STAT's AE decimal correct 4–7' signal.

4. 'Set STAA if PAL(7–63) equals zero' signal, with PAL(7–63) latched outputs containing all zeros.
5. 'Set STAA if PAL(32–63) equals zero' signal, with PAL(32–63) latched outputs containing all zeros.
6. 'Set STAA if PAL equals zero and insert sign' signal, with E(6) = 1.

The output of STAT A is entered directly into a polarity-hold latch, which unconditionally assumes the same binary state as STAT A at P1 not-clock time. (This latch retains its assumed state until not-clock time of the cycle in which STAT A is reset.)

STAT A is reset at P1 clock time if one of the following conditions is active:

1. 'STAT trigger reset' signal (conditioned by either a 'system reset' or an 'I-Fetch reset' signal).
2. 'Reset STAA' signal, which is conditioned when any one of the following is active:
   a. 'Edit reset STAA' signal.
   b. 'Set STAA if PAL(0–63) equals zero' signal.
   c. 'Set STAA if PAL(32–63) equals zero' and signal.
   d. 'Set STAA if PAL(32–63) equals zero and insert sign' signal, with E(6) = 1.
   e. 'Reset STAA if PAL(32–63) not equal zero' signal, with STAA polarity-hold latch set.

      Note: If PAL(32–63) contains all zeros and the STAA polarity-hold latch is set, the 'reset STAA if PAL(32–63) not equal zero' signal is inhibited from resetting STAT A.

   f. 'Set STAA if SAL(0–7) not equal zero' signal.

STAT B

- STAT B indicates:
  Zero-condition for serial adder.
  Overflow condition for decimal, fixed point, left-shift operations.
  Condition of PAL(31).
  Condition of B(32).

STAT B primarily indicates overflow conditions. Except during scan-in, when it is set directly to the value of T(55), STAT B is normally set at P2 clock time by one of the following conditions:

1. 'Set STAB if SAL(0–7) equals zero' signal, with SAL(0–7) containing all zeros.
2. 'Set STAB on decimal overflow' signal, with the 'decimal overflow' latch set.

   Note: The 'decimal overflow' latch is set at not-clock time of a decimal-compare cycle in which:
   a. The 'serial adder in bus A(7)' contains a 1 bit, and either STAT A, STAT D, or STAT H is reset.
   b. The 'serial adder in bus A(0–6)' is not equal to 0.
   c. STAT H is set, with STAT C and STAT F either both set or reset.

3. 'Set STAB if PAL(31) equals 1' signal, with PAL(31) = 1.
4. 'Gate fixed-point overflow to STAB' signal, with a fixed-point overflow condition prevailing.
5. 'Set STAB on left shift overflow' signal, with a left-shift overflow condition detected.
6. 'B(32) to STAB and T(32) to STAG' signal, with B(32) = 1. [STAT B is set at P2 + 140 ns under this condition to allow B(32) to become stable before it is sampled.]

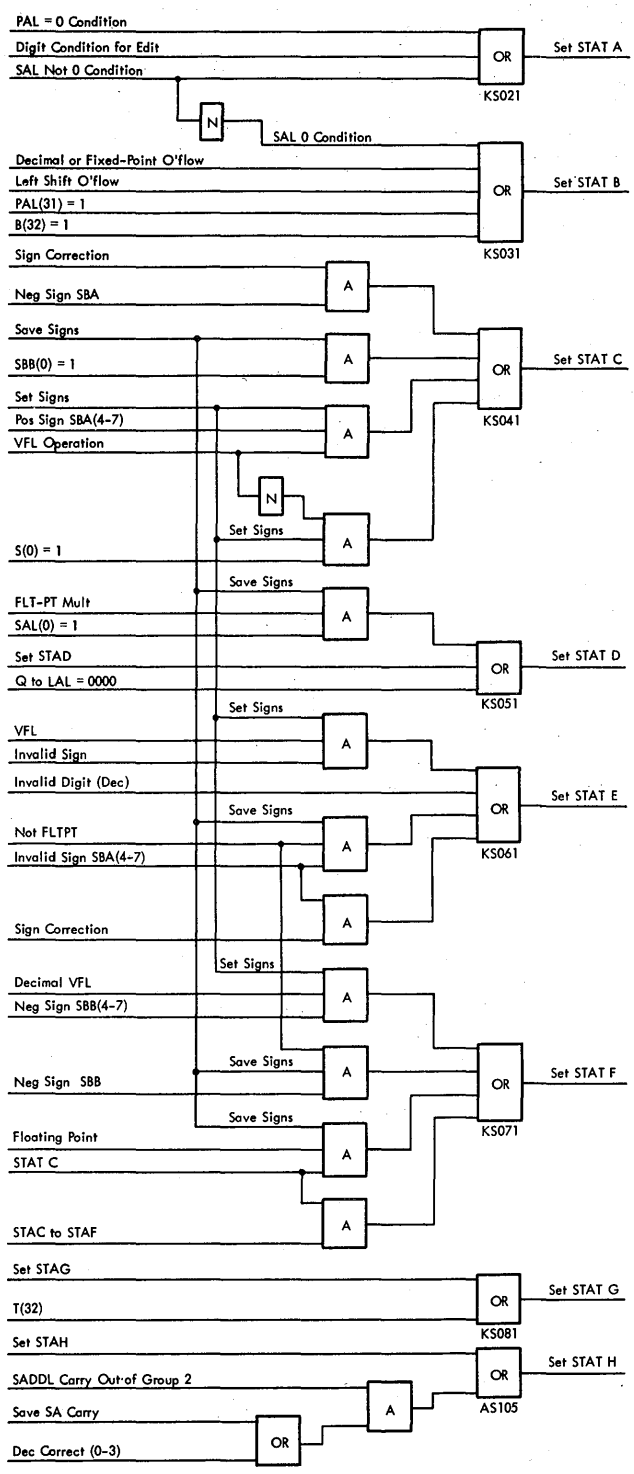


Figure 2-58. Summary of Setting of STAT's

The outputs of STAT B are sent directly to a polarity-hold latch. This latch unconditionally assumes the state of STAT B at not-clock time of the cycle in which STAT B is set, and retains this information until not-clock time of the cycle in which STAT B is reset. The output from the STAT B polarity-hold latch inhibits the resetting of STAT B whenever STAT B is set during the same cycle in which either a fixed-point overflow or a left-shift overflow is detected. (Either of these overflow conditions causes a program interruption requiring that STAT B remain set for interrogation.)

STAT B is normally reset at P1 clock time if one of the following conditions is active:

1. 'STAT reset' signal (conditioned by either a 'system reset' or an 'I-Fetch reset' signal).
2. 'Reset STAB' signal, which is conditioned if one of the following signals is active:
   a. 'Set STAB if PAL(31) equals 1'.
   b. 'Gate fixed-point overflow to STAB'.
   c. 'Set STAB on left shift overflow'.
   d. 'B(32) to STAB and T(32) to STAG' (resets STAT B at P1 + 140 ns).

STAT C

- STAT C holds:
  'Serial adder in bus A' sign on sign-correction VFL operations.
  'Serial adder in bus B' sign on save-signs VFL operations.
  Sign for set-signs on VFL and non-VFL operations.

STAT C primarily indicates the sign of a source operand. Except during scan-in, when it is set to the value of T(56), STAT C is set at P2 + 140 ns clock time by one of the following conditions:

1. 'Sign correct SA(4–7)' signal, with a negative sign detected on 'serial adder in bus A'.
2. 'Save signs' signal, with 'serial adder in bus B' position 0 containing a 1-bit during subtract or compare operations and a 0-bit during all others.
3. 'Set signs' signal during VFL instructions in which 'serial adder in bus A' positions 4–7 contain a positive sign during subtract or compare operations and a negative sign during all others.
4. 'Set signs' signal during any non-VFL operation in which S(0) = 1.

STAT C is normally reset at P1 clock time by the 'STAT trigger reset' signal (activated by either a 'system reset' or an 'I-Fetch reset' signal). STAT C is also reset at P1 + 140 ns whenever the 'set signs' signal is activated for operations other than VFL operations.

STAT D

- Used by microprogram to retain ROS branch information.
- Indicates sign for save-signs operation on floating-point multiply and divide.
- Indicates Q-to-LAL equals 0000.

STAT D stores a characteristic carry from SAL(0) during floating-point multiply and divide operations and indicates that the B1 or B2 field of an instruction equals zero. For operations other than floating-point multiply and divide and scan-in operations, STAT D is available for arbitrary microprogram use and can be unconditionally set or reset by the 'set STAT D' and 'reset STAT D' signals, respectively. (Such operations include storing of the dividend sign on fixed-point operations.) Except during scan-in operations, when it is set directly to the value of T(57), STAT D is normally set at P2 + 140 ns by one of the following conditions:

1. 'Set STAT D' signal.
2. 'Save signs' signal during floating-point multiply and divide operations in which SAL(0) = 1. (STAT D is set at P2 clock time under this condition.)
3. 'Gate Q to LAL 0000' signal.

Note: This signal is activated whenever the B1 or B2 field of an instruction is being gated from Q to LAL and is found to equal 0. Although STAT D is always set on this condition, its significance is of value only during an SS-format instruction when a B2 = 0000 indication must be retained for more than one cycle. (Used in setting ROSAR when selecting I-Fetch ROS words for SS-format instructions.)

STAT D is normally reset at P1 clock time by the 'STAT trigger reset' signal, which is activated by either a 'system reset' or an 'I-Fetch reset' signal. 'Reset STAD' and 'gate I-Fetch invalid address' signals reset STAT D at P0 + 140 ns. The 'reset STAD on decimal overflow' signal resets STAT D at P2 clock time.

STAT E

- Indicates invalid digits and signs.

STAT E primarily indicates the detection of invalid data during decimal operations. Except during scan-in, when it is set to the value of T(58), STAT E is normally set at clock P2 + 140 ns by one of the following conditions:

1. 'Set signs' signal during VFL operations in which an invalid sign is detected on either the 'serial adder in bus-A' or '-B'.
2. 'Save signs' signal for operations other than floating-point operations in which an invalid sign is detected on the 'serial adder in bus B(4–7)'.
3. 'Sign correct SA(4–7)' signal with the detection of an invalid sign on the 'serial adder in bus A(4–7)'.
4. Detection of an invalid digit on either side of the 'serial adder in bus'.
5. Detection of an invalid digit on the 'serial adder in bus A(0–3)', with the 'digit examine' latch set (edit operations).

STAT E is reset at P1 clock time by the 'STAT trigger reset' signal (activated by either a 'system reset' or an 'I-Fetch reset' signal).

STAT F

- STAT F holds:
  'Serial adder in bus B' sign on set-signs decimal operations.
  'Serial adder in bus B' sign on save-signs operations.
  Condition of STAT C.

STAT F primarily indicates the sign of VFL destination operands. Except during scan-in, when it is set to the value of T(59), STAT F is normally set at P2 + 140 ns by one of the following conditions:

1. 'Set signs' signal during a VFL decimal operation, with a negative sign detected on 'serial adder in bus B(4–7)'.
2. 'Save signs' signal during operations other than floating-point operations, with a negative sign detected on 'serial adder in bus B'.
3. 'Save signs' signal during a floating-point operation, with A(0) = 1.
4. 'STAC to STAF' signal, with STAT C set. (Sets STAT F at P2 clock time.)

STAT F is normally reset at P1 clock time by the 'STAT trigger reset' signal (activated by a 'system reset' or an 'I-Fetch reset' signal). The 'STAC to STAF' signal also resets STAT F at P0 clock time in preparation for setting STAT F again at P2.

STAT G

- Used by microprogram to retain ROS branching information.
- Indicates state of T(32).

STAT G is available for arbitrary microprogram use and for indicating the state of T(32). Except during scan-in, when it is set to the value of T(60), STAT G is normally set at P2 + 140 ns by one of the following conditions:

1. 'B(32) to STAB and T(32) to STAG' signal, with T(32) = 1.
2. 'Set STAG' signal.

STAT G is normally reset at P0 + 140 ns by the 'reset STAG' or 'B(32) to STAB and T(32) to STAG' signal. A 'system reset' signal and an 'I-Fetch reset' signal also reset STAT G.

STAT H

- Used for serial adder carry-control functions and ROS branching information.

STAT H indicates a serial-adder carry. Except during scan-in, when it is set to the value of T(38), STAT H is set by one of the following conditions:

1. 'Set STAH' signal and clock time P2 + 140 ns.
2. The output of a latch set at not-clock time by either a 'decimal correct 0–3 and set STAT's AE' or a 'save serial adder carry' signal in conjunction with a serial adder carry from group 2. The set condition is timed at P2 clock time.

STAT H is reset at P1 clock time whenever the latch referred to in item 2 is set. A 'system reset' signal and an 'I-Fetch reset' signal also reset STAT H at P1 clock time. It is also reset at P2 + 140 ns by the microprogram.

CONTROL TRIGGERS

A number of control triggers perform functions similar to STAT's. Table 2-2 lists the most significant triggers, summarizes their functions, and provides ALD and FETOM references.

Table 2-2. Control Triggers

Trigger	Function	ALD Reference	FETOM Reference	Roller Switch Indicator
Right Digit	Selects digit from AB byte on edit operations.	KZ321	Volume 2, Chapter 3, Section 5, "General Data Handling"	RT DIG Roller 4 Position 4 Bit 32
S	Indicates source character, rather than fill character, on edit character transfers.	KZ321	Volume 2, Chapter 3, Section 5, "Introduction to Edit Operation"	S Roller 4 Position 4 Bit 33
Leave	Controls 'serial adder bus B' on edit operations.	KZ201	—	LEAVE Roller 4 Position 4 Bit 34
Step ABC	Increments ABC on edit operations, if 'right digit' trigger is set.	KZ501	—	STEP ABC Roller 4 Position 4 Bit 35
Block I-Fetch	Prevents most I-Fetch functions when interruption or exceptional condition is to be processed.	KD501	Volume 2, Chapter 3, Section 1, "Block I-Fetch Trigger"	BLOCK Roller 4 Position 5 Bit 8
Branch Invalid Address	Indicates branch address of successful branch is invalid.	KD701	Volume 2, Chapter 3, Section 1, "Invalid Address Detection"	BR INVLD ADR Roller 4 Position 5 Bit 16
I-Fetch Invalid Address	Indicates Q has been refilled from invalid address.	KD711	Volume 2, Chapter 3, Section 1, "Invalid Address Detection"	INVLD ADR Roller 4 Position 5 Bit 17
Instruction Length Not Available	Resets ILC in old PSW and resets all interrupt code triggers except 'interrupt code 4' trigger. Set by "late" storage protection check.	KM851	Volume 2, Chapter 3, Section 1, "Fetch Protection Detection"	IL NOT AVAIL Roller 4 Position 5 Bit 18

Table 2-2. Control Triggers (Cont)

Trigger	Function	ALD Reference	FETOM Reference	Roller Switch Indicator
Time Clock at Limit	Indicates timer has been decremented past zero and requests external interruption	KM221	Volume 2, Chapter 3, Section 1, "Timer Exceptional Condition"	TC AT LIMIT Roller 4 Position 5 Bit 19
Timing Gate	Controls duration of I/O control or direct-control signals. Set and reset by microprogram.	KX311	Volume 2, Chapter 3, Section 7, "Write Direct, WRD (84)" and "Read Direct, RDD (85)"	TIME GATE TGR Roller 4 Position 5 Bit 35
No Retry†	Indicates to diagnostic programmer instruction retry may give unpredictable results. Set by (1) 'store per D' signal, (2) 'PAL to IC' signal and not SS format, and (3) 'write local stor' signal and LSWR not selected.	KS321	—	NO RETRY Roller 4 Position 1 Bit 18
IC in LSWR†	Indicates IC is saved in LSWR. Occurs only on SS format operations.	KS321	—	IC IN LSWR Roller 4 Position 1 Bit 19

† 'No retry' and 'IC in LSWR' triggers perform no control function but indicate machine conditions only.

www.IBM360.info

IBM Field Engineering

Theory of Operation

www.IBM360.info

2065 Processing Unit, Volume 2

Preface

This manual describes the operation of the 2065 Processing Unit. It is assumed that the reader has a knowledge of processors, of ALD interpretation, and of the basic circuits used in the 2065.

The EC levels of the ALD's and CLD's for the basic 2065, upon which this manual and its companion maintenance diagram manual are based, are:

ALD's: EC 705369 9/68
CLD's: EC 705340 3/68
Power: EC 711576 8/68

The manual consists of two volumes, and is divided into six chapters and three appendices. Volume 1, Form Y27-2036-0, contains:

Chapter 1, Introduction. Discusses system organization and data flow; character codes, instruction formats, and operands; program execution and control; and the CPU functional units and the Universal instruction set.

Chapter 2, Functional Units. Analyzes registers, adders, and counters individually, except for those units that work together to perform a specific function (for example, variable-field-length register and its associated byte counter).

Volume 2, Form Y27-2037-0, contains:

Chapter 3, Principles of Operation. Presents a detailed analysis of instruction fetching, and instruction execution by instruction class.

Chapter 4, Features. Discusses the features available for the 2065 CPU.

Chapter 5, Power Distribution and Control. Describes the power distribution and control within the CPU (making a distinction between 2065's and 2060's that have been converted to 2065's) and within the system.

Chapter 6, Console Controls and Maintenance Features. Discusses the controls on the system control panel and on the CE panel and their application, and the maintenance features available.

Appendix A, Special Circuits. Discusses the special circuits in the 2065.

Appendix B, World Trade Differences. Discusses the major difference between the World Trade version of the Model 65 and the domestic version.

Appendix C, Example of FLT Generation. Discusses FLT generation, using a simple four-block tree as an example.

Volume 2 also contains the index for the complete manual.

Following most paragraph heads are bullets (key statements preceded by •) which summarize significant points about the subject. The bullets serve two functions: (1) they provide the CE with the key points of the topic, and (2) they provide quick reference for review and recall for the CE who is familiar with the machine. Detailed text follows, providing the non-classroom student with the fill-in material necessary for self-instruction.

The diagrams supporting the text are divided into two groups: (1) purely instructional diagrams and (2) maintenance-oriented diagrams and diagrams that aid recall. Examples of the first group are high-level block diagrams and diagrams that show general data flow and timing considerations. These diagrams are generally not affected by engineering changes, and, if they include AND/OR logic blocks, the blocks are drawn in positive logic convention and do not maintain ALD lines or line names. The instruction diagrams, which are placed in this manual and called "Figures", are numbered consecutively within a chapter. (For example, 1-1 is the first figure in Chapter 1; 3-7 is the seventh figure in Chapter 3.)

The diagrams of the second group are referenced in this manual (for example, Diagram 5-30, FEMDM) but are located in the companion FE Maintenance Diagrams Manual to allow ready reference during maintenance and to facilitate updating the diagrams to new engineering levels. These diagrams are grouped by categories similar to the chapters of this manual.

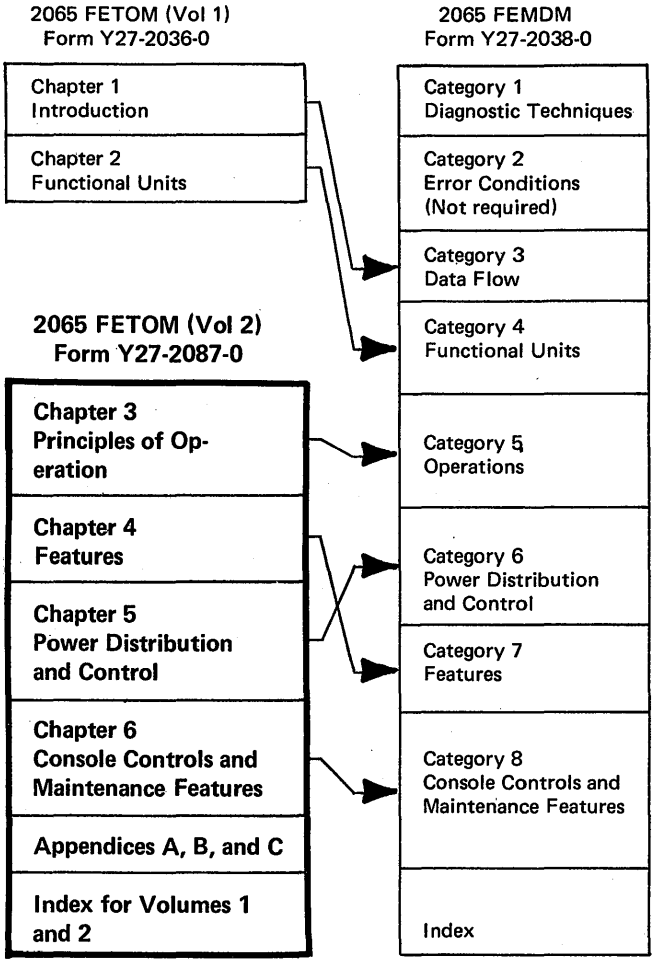
Second Edition (May 1974)
This edition, SY27-2037-1, is a reprint of SY27-2037-0, incorporating changes released in the following FE Supplement:
SY27-2259 (dated October 16, 1969)

Changes are periodically made to the information herein; any such changes will be reported in subsequent revisions or Technical Newsletters.

This manual has been prepared by the IBM System Products Division, Product Publications, Dept. B97, PO Box 390, Poughkeepsie, N.Y. 12602. A form for readers' comments is provided at the back of this publication. If the form has been removed, comments may be sent to the above address. Comments become the property of IBM.

©Copyright International Business Machines Corporation 1969

The relationship of this manual to the FEMDM is shown below. (Arrows indicate cross-referencing between chapters in this manual and categories of diagrams in the FEMDM: for example, most references in Chapter 3 are made to Category 5 diagrams.)



Companion, related, and prerequisite manuals and standards are:

2065 Processing Unit
FEMDM, Form Y27-2038-0
FEMM, Form Y27-2270-0

IBM System/360 Principles of Operation, SRL, Form A22-6821-7.

2065 Processing Unit, 7070/7074 Compatibility Feature
FETOM, Form Y27-2106-0
FEDM, Form Y27-2107-0

2065 Processing Unit, 7080 Compatibility Feature
FETOM, Form Y27-2090-0
FEDM, Form Y27-2091-0

2065/2067 Processing Unit, 709/7040/7044/7090/7094/7094II Compatibility Feature
FETOM, Form Y27-2098-0
FEDM, Form Y27-2099-0

2365 Processor Storage
FETOM, Form Y22-6608-0
FEDM, Form Y22-6601-1
FEMM, Form Y22-6600-1

2361 Core Storage
FETOM, Form Y22-2897-0
FEDM, Form Y22-2895-0
FEMM, Form Y22-2894-0

2860 Selector Channel
FETOM, Form Y27-2220-0
FEMDM, Form Y27-2221-0
FEMM, Form Y22-2893-1

2870 Multiplexer Channel (70,000 Series)
FETOM, Form Y27-2152-0
FEDM, Form Y27-2153-0
FEMM, Y27-2154-0

1052 Adapter and 2150 Console, FETOM, Form Y22-2808

SLT Component Circuits, FEMI, Form Z22-2798 (IBM Confidential)

SLT Power Supplies, FEMI, Form 223-2799

SLT Packaging, FEMI, Form 223-2800

Control Automation System (CAS) Logic Diagram (CLD), IBM Corporate Engineering Standard, CES 0-1046-4

Abbreviations

ABC	AB register byte counter
ac	alternating current
adr	address, addressed, addressing
ALD	automated logic diagram
amp	ampere
ASC	address store compare
ATN	alternate test number
BCD	binary-coded decimal
BCU	bus control unit
C	capacitor
CAW	channel address word
CB	circuit breaker
CC	condition code
CCW	channel command word
CE	customer engineer
charistic	characteristic
CLD	control automation system logic diagram
CPU	central processing unit
CR	diode

CROS	capacitive read-only storage
CSW	channel status word
CT	conditional terminate
dc	direct current
dec div	decimal divide
dec ovflo	decimal overflow
DX	first byte in a series of destination bytes
DX + 1	second byte in a series of destination bytes
DX + 2	third byte in a series of destination bytes
end op	end operation
EPO	emergency power off
ERSLT	expected result
exp ovflo	exponent overflow
exp unflo	exponent underflow
F	fuse
FEMDM	Field Engineering Maintenance Diagrams Manual
FEMI	Field Engineering Manual of Instruction
FEMM	Field Engineering Maintenance Manual
FETOM	Field Engineering Theory of Operation Manual
fix-pt div	fixed-point divide
fix-pt ovflo	fixed-point overflow
FLT	fault locating test
flt-pt div	floating-point divide
FLUT	Fault Locating Utility program
FPR	floating-point register
fract	fraction
GIS	general initialization sequence
GPR	general-purpose register
hex	hexadecimal
HSS	high-speed storage
Hz	Hertz
IC	instruction counter
I-Fetch	instruction fetching
ILC	instruction length code
I/O	input/output
IPL	initial program load
K	kilo
K	relay
kHz	kilohertz
LAL	local storage address latches
LAR	local storage address register
LCS	large capacity storage
LS	local storage
LSWR	local storage working register
MAR	memory address register
max	maximum
MCW	maintenance control word
mHz	megahertz
MMSC	maintenance mode stop clock
MPR	multiplier
ms	millisecond
multisys	multisystem
no op	no operation
ns	nanosecond

op code	operation code
oper	operation
opr	operand
P	parity
PAA	parallel adder A-side
PAB	parallel adder B-side
PAL	parallel adder latch
pf	picofarad
PK	power contactor
PP	partial product
PQ	partial quotient
priv oper	privileged operation
prot	protection
PS	power supply
PSW	program status word
R	resistor
ROS	read-only storage
ROSAR	read-only storage address register
ROSBR	read-only storage backup register
ROSDR	read-only storage data register
ROSPARA	read-only storage previous address register A
ROSPARB	read-only storage previous address register B
SAA	serial adder A-side
SAB	serial adder B-side
SAB	storage address bus
SAL	serial adder latch
SAR	storage address register
SBA	serial adder bus A
SBB	serial adder bus B
SCOPEX	scoping index
SCR	silicon-controlled rectifier
SDBI	storage data bus in
SDBO	storage data bus out
signif	significance
SLT	solid logic technology
SMS	standard modular system
SOROS	scan out read-only storage
spec	specification
SRL	Systems Reference Library
STAT	status trigger
STC	ST register byte counter
stg	storage
SW BD	switch board
sync	synchronizing
T	transformer
T(DX)	table byte specified by DX
T(DX + 1)	table byte specified by DX + 1
TIC	transfer in channel
TN	test number
uf	microfarad
usec	microsecond
UT	unconditional terminate
V	volt
VFL	variable-field length
$\geq$	greater than or equal to
$\geqslant$	greater than or equal to
$\leq$	less than or equal to
$\leqslant$	less than or equal to
=	equal to
$\neq$	not equal to

Chapter 3 Principles of Operation 3-1

SECTION 1 INSTRUCTION FETCHING 3-1
Basic End-Op Cycle 3-1
Prefetching of Operands During End Op 3-2
Fetching of Instructions by End-Op Micro-Order 3-3
Requests During End Op 3-3
Requests During Early End Op 3-3
Selection of I-Fetch Microprogram 3-4
Basic RR I-Fetch 3-5
Basic RX I-Fetch 3-6
Basic RS and SI I-Fetch 3-6
Basic SS I-Fetch 3-7
Address Storage Compare (ASC) Test 3-9
I-Fetch Microprogram 3-9
I-Fetch Control If at End Op IC(21,22) = 00 3-11
I-Fetch Control If at End Op IC(21,22) = 01 3-12
I-Fetch Control If at End Op IC(21,22) = 10 3-12
I-Fetch Control If at End Op IC(21,22) = 11 3-12
Deviations from Basic End Op and I-Fetch 3-12
I-Fetch Sequencers 3-12
Block I-Fetch Trigger 3-13
Timer Exceptional Condition 3-14
CPU Store In Progress Exceptional Condition 3-14
Machine Check Interruption 3-14
Program Interruption 3-15
Supervisor Call Interruption 3-15
External Interruption 3-16
I/O Interruption 3-16
Common Interruption Routine 3-17
Stop, Wait, and Repeat Exceptional Conditions 3-17
Program Store Compare Exceptional Condition 3-17
Invalid Instruction Address Test Exceptional Condition 3-18
Specification Detection 3-18
Invalid Address Detection 3-19
Fetch Protection Detection 3-21
Invalid Instruction Address Microprogram 3-21
Q-Register Refill Exceptional Condition 3-22
Two-Cycle RR I-Fetch 3-23
Forced-Cycle RX I-Fetch 3-24
Two-Cycle RS and SI I-Fetch 3-24

SECTION 2 FIXED-POINT INSTRUCTIONS 3-25
Load 3-25
Load, LR (18) 3-25
Load, L (58) 3-25
Load Halfword, LH (48) 3-26
Load and Test, LTR (12) 3-27
Load Complement, LCR (13) 3-27
Load Positive, LPR (10) 3-28
Load Negative, LNR (11) 3-28
Load Multiple, LM (98) 3-29
Add-Type Instructions 3-30
Add, AR (1A) 3-31
Add, A (5A) 3-31
Add Halfword, AH (4A) 3-31
Add Logical, ALR (1E) 3-32
Add Logical, AL (5E) 3-33
Subtract, SR (1B) 3-33
Subtract, S (5B) 3-34
Subtract Halfword, SH (4B) 3-34
Subtract Logical, SLR (1F) 3-35
Subtract Logical, SL (5F) 3-35
Compare, CR (19) 3-36
Compare, C (59) 3-36
Compare Halfword, CH (49) 3-37
Multiply 3-37
Multiply, MR (1C) 3-38
Multiply, M (5C) 3-42
Multiply Halfword, MH (4C) 3-42
Divide 3-42
Divide, DR (1D) 3-43
General Discussion 3-44
Detailed Discussion 3-45
Divide, D (5D) 3-47
Convert 3-47
Convert to Binary, CVB (4F) 3-51
Convert to Decimal, CVD (4E) 3-51
Store 3-53
Store, ST (50) 3-53
Store Halfword, STH (40) 3-54
Store Multiple, STM (90) 3-55
Shift 3-56
Shift Left Single, SLA (8B) 3-56
Shift Left Double, SLDA (8F) 3-58
Shift Right Single, SRA (8A) 3-60
Shift Right Double, SRDA (8E) 3-61

SECTION 3 FLOATING-POINT INSTRUCTIONS 3-63
Exponent Overflow and Underflow 3-63
Zero Results 3-63
Conditions at Start of Execution 3-64
Load 3-64
Load, LER (38) – RR Short Operands 3-64
Load, LE (78) – RX Short Operands 3-64
Load, LDR (28) – RR Long Operands 3-65
Load, LD (68) – RX Long Operands 3-65
Load and Test, LTER (32) – RR Short Operands 3-66
Load and Test, LTDR (22) – RR Long Operands 3-66
Load Complement, LCER (33) – RR Short Operands . . 3-67
Load Complement, LCDR (23) – RR Long Operands . . 3-67
Load Positive, LPER (30) – RR Short Operands 3-68
Load Positive, LPDR (20) – RR Long Operands 3-68
Load Negative, LNER (31) – RR Short Operands . . . 3-68
Load Negative, LNDR (21) – RR Long Operands 3-69
Add, Subtract, and Compare 3-69
Add Normalized, AER (3A) – RR Short Operands . . . 3-71
Add Normalized, AE (7A) – RX Short Operands 3-75
Add Normalized, ADR (2A) – RR Long Operands . . . 3-76
Add Normalized, AD (6A) – RX Long Operands 3-77
Add Unnormalized, AUR (3E) – RR Short Operands . . 3-78
Add Unnormalized, AU (7E) – RX Short Operands . . . 3-78
Add Unnormalized, AWR (2E) – RR Long Operands . . 3-79
Add Unnormalized, AW (6E) – RX Long Operands . . . 3-79
Subtract Normalized, SER (3B) – RR Short Operands 3-80
Subtract Normalized, SE (7B) – RX Short Operands 3-81

Subtract Normalized, SDR (2B) – RR Long Operands . . . 3-81
Subtract Normalized, SD (6B) – RX Long Operands . . . 3-82
Subtract Unnormalized, SUR (3F) – RR Short Operands . . . 3-82
Subtract Unnormalized, SU (7F) – RX Short Operands . . . 3-83
Subtract Unnormalized, SWR (2F) – RR Long Operands . . . 3-83
Subtract Unnormalized, SW (6F) – RX Long Operand . . . 3-84
Compare, CER (39) – RR Short Operands . . . 3-84
Compare, CE (79) – RX Short Operands . . . 3-85
Compare, CDR (29) – RR Long Operands . . . 3-85
Compare, CD (69) – RX Long Operands . . . 3-86
Halve . . . 3-86
Halve, HER (34) – RR Short Operands . . . 3-86
Halve, HDR (24) – RR Long Operands . . . 3-87
Multiply . . . 3-87
Data Flow and Algorithm . . . 3-88
Multiply, MER (3C) – RR Short Operands . . . 3-91
Multiply, ME (7C) – RX Short Operands . . . 3-92
Multiply, MDR (2C) – RR Long Operands . . . 3-93
Multiply, MD (6C) – RX Long Operands . . . 3-93
Divide . . . 3-94
Characteristic Computation . . . 3-95
Normalization . . . 3-96
Fraction Division . . . 3-96
Data Flow and Algorithm . . . 3-98
Divide, DER (3D) – RR Short Operands . . . 3-100
Divide, DE (7D) – RX Short Operands . . . 3-101
Divide, DDR (2D) – RR Long Operands . . . 3-102
Divide, DD (6D) – RX Long Operands . . . 3-103
Store . . . 3-104
Store, STE (70) – RX Short Operands . . . 3-104
Store, STD (60) – RX Long Operands . . . 3-105

SECTION 4 DECIMAL INSTRUCTIONS . . . 3-106
Instruction Handling . . . 3-106
Word Overlap Condition . . . 3-107
General Initialization Sequence . . . 3-109
Add, Subtract, and Compare . . . 3-109
Add, AP (FA) and Subtract, SP (FB) . . . 3-109
GIS for Add and Subtract . . . 3-110
True Add Sequence . . . 3-110
Complement Add Sequence . . . 3-113
Compare, CP (F9) . . . 3-114
Zero and Add, ZAP (F8) . . . 3-115
Multiply, MP (FC) . . . 3-116
General Description . . . 3-118
Detailed Description . . . 3-124
Divide, DP (FD) . . . 3-125
General Description . . . 3-128
Detailed Description . . . 3-134
Pack, PACK (F2) . . . 3-137
Instruction Execution, Not Word Overlap . . . 3-138
Instruction Execution, Word Overlap . . . 3-139
Unpack, UNPK (F3) . . . 3-139
Instruction Execution, Not Word Overlap . . . 3-140
Instruction Execution, Word Overlap . . . 3-141
Move With Offset, MVO (F1) . . . 3-141
Instruction Execution, Not Word Overlap . . . 3-142
Instruction Execution, Word Overlap . . . 3-142

SECTION 5 LOGICAL INSTRUCTIONS . . . 3-144
General Initialization Sequence . . . 3-144
Move . . . 3-144
Move, MVI (92) . . . 3-144
Move, MVC (D2) . . . 3-144
Move Numerics, MVN (D1) . . . 3-145
Move Zones, MVZ (D3) . . . 3-146
Compare . . . 3-147
Compare Logical, CLR (15) . . . 3-147
Compare Logical, CL (55) . . . 3-147
Compare Logical, CLI (95) . . . 3-147
Compare Logical, CLC (D5) . . . 3-148
AND . . . 3-148
AND, NR (14) . . . 3-149
AND, N (54) . . . 3-149
AND, NI (94) . . . 3-150
AND, NC (D4) . . . 3-150
OR . . . 3-150
OR, OR (16) . . . 3-151
OR, O (56) . . . 3-151
OR, OI (96) . . . 3-151
OR, OC (D6) . . . 3-152
Exclusive-OR . . . 3-152
Exclusive-OR, XR (17) . . . 3-152
Exclusive-OR, X (57) . . . 3-153
Exclusive-OR, XI (97) . . . 3-153
Exclusive-OR, XC (D7) . . . 3-153
Test Under Mask, TM (91) . . . 3-154
Insert Character, IC (43) . . . 3-154
Store Character, STC (42) . . . 3-154
Load Address, LA (41) . . . 3-155
Translate, TR (DC) . . . 3-155
Translate and Test, TRT (DD) . . . 3-156
Edit and Edit and Mark, ED and EDMK (DE and DF) . . . 3-158
Introduction to Edit Operation . . . 3-158
Introduction to Edit and Mark Operation . . . 3-160
General Data Handling . . . 3-160
Microprogram Description . . . 3-161
First Cycle . . . 3-161
Second Cycle . . . 3-161
Exit Conditions . . . 3-161
Shift . . . 3-162
Shift Left Single, SLL (89) . . . 3-162
Shift Left Double, SLDL (8D) . . . 3-162
Shift Right Single, SRL (88) . . . 3-162
Shift Right Double, SRDL (8C) . . . 3-162

SECTION 6 BRANCHING INSTRUCTIONS . . . 3-164
Branch on Condition, BCR (07) . . . 3-164
Successful Branch . . . 3-164
Unsuccessful Branch . . . 3-165
Branch on Condition, BC (47) . . . 3-165
Branch and Link, BALR (05) . . . 3-166
Unsuccessful Branch . . . 3-166
Successful Branch . . . 3-167
Branch and Link, BAL (45) . . . 3-168
Branch on Count, BCTR (06) . . . 3-169
Successful Branch . . . 3-170
Unsuccessful Branch . . . 3-170
Branch on Count, BCT (46) . . . 3-170
Branch on Index High, BXH (86) . . . 3-170
Branch on Index Low or Equal, BXLE (87) . . . 3-172
Execute, EX (44) . . . 3-173

SECTION 7 STATUS SWITCHING INSTRUCTIONS . . . 3-176
Load PSW, LPSW (82) . . . 3-176
Set Program Mask, SPM (04) . . . 3-177
Set System Mask, SSM (80) . . . 3-177
Supervisor Call, SVC (0A) . . . 3-177
Set Storage Key, SSK (08) . . . 3-178
Insert Storage Key, ISK (09) . . . 3-179
Write Direct, WRD (84) . . . 3-180
Read Direct, RDD (85) . . . 3-181
Diagnose (83) . . . 3-182
Test and Set, TS (93) . . . 3-183

SECTION 8 INPUT/OUTPUT INSTRUCTIONS . . . 3-184
Start I/O, SIO (9C) . . . 3-184
Test I/O, TIO (9D) . . . 3-186
Halt I/O, HIO (9E) . . . 3-186
Test Channel, TCH (9F) . . . 3-187

Chapter 4 Features . . . 4-1

SECTION 1 FEATURE INDEX . . . 4-1

SECTION 2 MULTIPROCESSING FEATURES . . . 4-2
Multiprocessing System/360 Model 65 . . . 4-2
Main Storage . . . 4-2
Storage Allocation . . . 4-2
Floating Addressing . . . 4-2
Direct Address Relocation . . . 4-3
Input/Output . . . 4-4
Processing Units . . . 4-4
Multisystem Mode . . . 4-4
Model 65 Mode . . . 4-5
Partition Mode . . . 4-5
Multisystem Signals . . . 4-5
Summary of Multiprocessing System Advantages . . . 4-6
Functional Units . . . 4-6
Configuration Control Panel . . . 4-7
Storage Allocation Control . . . 4-7
Floating Address Control . . . 4-7
Direct Address Relocation Control . . . 4-8
Multiprocessing System Mode Control . . . 4-8
I/O Allocation Control . . . 4-8
BCU Modifications . . . 4-8
Storage Address Decoding with Prefixing Disabled . . . 4-8
Storage Address Decoding with Prefixing Enabled . . . 4-8
Invalid Storage Address . . . 4-9
BCU-Storage Operations . . . 4-9
Multisystem Timer . . . 4-9
Operation . . . 4-9
System Hang Timing . . . 4-10
External System Reset Timing . . . 4-10
Multisystem Operations . . . 4-10
Set System Mask Instruction (Multisystem Mode) . . . 4-10
Write Direct Instruction (Not Model 65 Mode) . . . 4-10
Read Direct Instruction (Not Model 65 Mode) . . . 4-11
Malfunction Alert . . . 4-11
Gated Load . . . 4-11
System Call . . . 4-11
Log I/O Interrupt . . . 4-12
External System Reset . . . 4-12
External Start . . . 4-12
Power Distribution and Control . . . 4-13
Console Controls and Maintenance Features . . . 4-13
Configuration Control Panel . . . 4-13
Storage Allocation Switches . . . 4-13
Floating Address Switches . . . 4-13
PREFIX Switches . . . 4-13
CPU Mode Switches . . . 4-13
I/O Allocation Switches . . . 4-14
VALID ADDRESS Indicators . . . 4-14
System Control Panel Modifications, Multisystem Feature . . . 4-14
EMERGENCY PULL Switch . . . 4-14
POWER ON Pushbutton . . . 4-14
Marginal Voltage Control . . . 4-14
DISABLE INTERVAL TIMER Switch . . . 4-14
DISABLE DIRECT CONTROL Switch . . . 4-14
Storage Switches . . . 4-14
Indicators . . . 4-14
System Control Panel Modifications, Additional Storage Attachment Features . . . 4-15
Marginal Voltage Control . . . 4-15
POWER CHECK Indicators . . . 4-15
STORAGE INDICATE SWITCH and Indicators . . . 4-15
Logout and Scan In . . . 4-15

Chapter 5 Power Distribution and Control . . . 5-1
AC Power Distribution . . . 5-1
60-Hz Units . . . 5-1
50-Hz Units . . . 5-1
Converter/Inverter . . . 5-1
DC Power Distribution . . . 5-4
High-Frequency Regulator Modules . . . 5-4
Marginal Adjustments . . . 5-6
Power-On Sequence . . . 5-6
Power-Off Sequence . . . 5-8
Normal Power-Off . . . 5-8
Emergency Power-Off . . . 5-8
Automatic Power-Off . . . 5-9
Overcurrent Protection . . . 5-9
Overvoltage Protection . . . 5-9
Positive Regulators, Converted Units . . . 5-10
Negative Regulators, Converted Units . . . 5-10
Undervoltage Protection . . . 5-11
Thermal Protection . . . 5-11
Indicators . . . 5-11
System Power-On Indicator . . . 5-11
Power Check Indicators . . . 5-11
1052 Printer-Keyboard Power . . . 5-12
Audible Alarm . . . 5-12
Dual 1052 Power Interface . . . 5-12
Direct Control Power . . . 5-12
Usage Meters and Key Switch . . . 5-13

Chapter 6 Console Controls and Maintenance Features . . . 6-1

SECTION 1 CONSOLE CONTROLS . . . 6-1
System Control Panel . . . 6-1
Manual Controls . . . 6-4
Stop Loop . . . 6-5
Power-On Reset . . . 6-5
SYSTEM RESET Pushbutton . . . 6-6
CHECK RESET Pushbutton . . . 6-6
STOP Pushbutton . . . 6-6
LOAD Pushbutton (IPL) . . . 6-6
DATA Switches . . . 6-9
ADDRESS Switches . . . 6-9
ADDRESS COMPARE STOP Switch . . . 6-9

STORAGE SELECT Switch 6-9
DEFEAT INTERLEAVING Switch 6-12
STORAGE INDICATE Switch 6-12
STOP ON STORAGE CHECK Switch 6-12
SET IC Pushbutton 6-12
RATE Switch 6-13
PROCESS Position 6-13
INSN STEP Position 6-13
SINGLE CYCLE Position 6-13
SINGLE CYCLE STORAGE INHIBIT Position . . 6-13
REPEAT INSN Switch 6-13
Repeat Single Instruction 6-14
Repeat Multiple Instructions 6-14
STORE Pushbutton 6-14
DISPLAY Pushbutton 6-15
START Pushbutton 6-15
ROS TRANSFER Pushbutton 6-16
Storage-Ripple Microprogram 6-16
Storage-Ripple-Store Function 6-16
Storage-Ripple-Display Function 6-16
REPEAT ROS ADDRESS Switch 6-16
PSW RESTART Pushbutton and Wait State 6-16
DISABLE DIRECT CONTROL Switch 6-17
DISABLE INTERVAL TIMER Switch 6-17
INTERRUPT Pushbutton 6-17
CPU CHECK Switch 6-17
PULSE MODE Switch 6-17
PROC Position 6-18
TIME Position 6-18
COUNT Position 6-18
LOG OUT Pushbutton 6-18
TEST MODE, ROS/PROC/FLT Switch 6-18
TEST MODE, REPEAT Switch 6-19
RESTART FLT I/O Pushbutton 6-19
CE Key Switch and Usage Meters 6-19
FREQUENCY ALTERATION Switch 6-19
Indicators 6-19
CE Panel 6-21

SECTION 2 MAINTENANCE FEATURES 6-22
Diagnose Instruction and MCW's 6-22
Diagnose Instruction MCW for CPU 6-23
Diagnose Instruction MCW for Channel 6-23
ROS Test MCW 6-24
FLT MCW 6-24
Logout, ROS Tests, and FLT's 6-24
Introduction 6-24
Logout 6-25
ROS Tests 6-26
FLT's 6-26
FLT Tapes 6-27
Tape Generation 6-27
FLT Hardcore Tests 6-28
Zero-Cycle Tests 6-28
One-Cycle Tests 6-28
FLT Format 6-29
Scan Logic Functional Units 6-29
Scan Timing 6-29
Scan Clock 6-30
FLT Clock 6-30
Scan Counter Latches and Decrementer 6-31
Input and Output 6-31
Scan Counter Decrementer 6-31
Address Sequencer 6-31
Address Sequencer Decoder 6-33
Storage Address Generator 6-34
Check Counter 6-34
Input and Output 6-35
Check Counter Decrementing 6-36
FLT Counter 6-36
Input 6-36
FLT Counter Decrementing 6-37
Cycle Counter 6-37
ROS Test Sequencer 6-38
Scan-Out Bus 6-38
Logout Controls 6-38
Scan Out S and T 6-39
Scan Stop-CPU-Clock Logic 6-39
Control Triggers 6-39
Scan Mode Control of ROS 6-40
Scan/Channel Interface 6-40
Operational Analysis 6-40
Logout 6-40
Hardware-Controlled Sequence 6-41
ROS-Controlled Sequence 6-42
ROS Tests 6-43
ROS Test Tape 6-43
ROS Test Setup 6-44
IPL 1 6-44
Loader 6-45
Hardcore Tests and IPL 2 6-45
Test 1, Record 3 6-46
IPL 2 6-47
Test 2, Record 5 6-47
Record 6 6-47
Test 1, Record 7 6-48
Test 1, Record 8 6-48
Test 2, Record 8 6-48
Test 3, Record 8 6-48
Summary of Hardcore Tests 6-48
ROS Bit Tests 6-49
ROS Test State 7 6-49
ROS Test State 6 6-49
ROS Test State 5 6-49
ROS Test State 4 6-49
ROS Test State 3 6-50
ROS Test State 2 6-50
ROS Test State 1 6-50
ROS Test State 0 6-50
Fault Locating Tests 6-51
FLT Tape 6-51
FLT Format 6-52
FLT Test Setup 6-53
IPL 1 6-53
Loader 6-53
Transmission Checks During FLT Read In 6-54
Hardcore Tests 6-54
Zero-Cycle and One-Cycle Tests 6-54
Scan In 6-55
Test Cycles 6-56
Scan Out 6-56
Result Comparison 6-56
Terminate or Continue 6-56
TN/ATN Comparison 6-57
Ripple Tests 6-57
Diagnostic Programs 6-57
Marginal Checking 6-58

Appendix A A-1
Appendix B B-1
Appendix C C-1

Index X-1

Illustrations

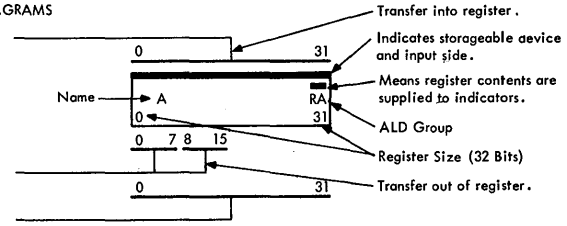
Legend x
3-1 Typical Microprogram Sequence 3-1
3-2 Basic Sequencing for SS Instructions 3-8
3-3 ASC Test for SS Instructions 3-10
3-4 Detection of Invalid Instruction Address . . 3-20
3-5 Detection of Fetch-Protected Instruction Address 3-22
3-6 Fixed-Point Multiply, Example No. 1 (RR Format) 3-40
3-7 Fixed-Point Multiply, Example No. 2 (RR Format) 3-41
3-8 Fixed-Point Divide, Example No. 1 3-48
3-9 Fixed-Point Divide, Example No. 2 3-48
3-10 Convert to Decimal Example 3-52
3-11 Restore and Non-Restore Division 3-97
3-12 Fraction Divide Example 3-98
3-13 Floating-Point Divide Example 3-102
3-14 Operand Specifications for Decimal Multiply Instruction 3-118
3-15 Typical Multiply Add Sequence 3-119
3-16 Typical Multiply Subtract Sequence 3-120
3-17 Data Handling During GIS of Decimal Multiply 3-121
3-18 Data Handling During Multiplier Left-Adjust Sequence 3-122
3-19 Data Flow for Right-4 Shift of ST to AB, Decimal Multiply 3-126
3-20 Operand Specifications for Decimal Divide 3-128
3-21 Example of a Typical Divide Sequence . . . 3-129
3-22 Data Handling During GIS of Decimal Divide 3-130
3-23 Data Handling During Divisor Left-Adjust Sequence 3-131
3-24 Data Handling During Dividend Fetch and Left-Adjust Sequence 3-133
3-25 Simplified Data Flow for AND, OR, and Exclusive-OR Instructions 3-149
3-26 Example of Use of Branch and Link Instruction 3-167
3-27 Storage Protection Key Assignments 3-179
4-1 Direct Address Relocation 4-3
4-2 Duplicate Storage Addressing Detection . . 4-7
5-1 Primary AC Power Distribution, 60-Hertz Units 5-2
5-2 Primary AC Power Distribution, 50-Hertz Units 5-3
5-3 Representative DC Distribution 5-5
5-4 Overcurrent Protection Loop 5-10
6-1 Normal IPL Operaiton 6-7
6-2 Data Switch Gating 6-10
6-3 Address Switch Gating 6-11
6-4 Scan Counter Latches and Decrementer Data Flow 6-32
6-5 Address Sequencer Data Flow 6-33
6-6 Address Sequencer Decoder 6-34
6-7 Check Counter Data Flow 6-35
6-8 FLT Counter Data Flow 6-36
6-9 Cycle Counter Data Flow 6-37
6-10 ROS Test Sequencer Data Flow 6-38
C-1 Four-Block Tree and FLT Pattern Generated C-1

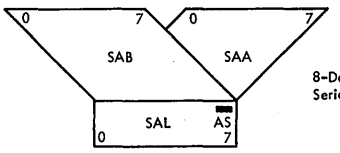
Tables

3-1 Q-Register Refill Exceptional Conditions . . 3-23
3-2 Value of Multiple Determined by Multiple Selection Bits (Fixed-Point) 3-39
3-3 Divide Multiple Values, Fixed-Point 3-46
3-4 Conversion to Decimal (Excess-6) 3-51
3-5 Excess-6 Conversion, B(60–63) 3-53
3-6 Operand Bits Transferred, STH Instruction . 3-55
3-7 Left Shift Combinations 3-57
3-8 Right Shift Combinations 3-61
3-9 Examples of Branching on Characteristic Difference 3-73
3-10 Multiplier Bits Selected, Floating-Point Multiply 3-89
3-11 Value of Multiple Determined by Multiple Selection Bits (Floating-Point) 3-90
3-12 Condition Code Setting Per Hardware Conditions, Decimal Instructions 3-113
4-1 Feature Index 4-1
4-2 Floating Address Intervals 4-3
5-1 High-Frequency Regulator Modules 5-4
6-1 Logout Format 6-25
6-2 FLT Format 6-52
C-1 SCOPEX C-2

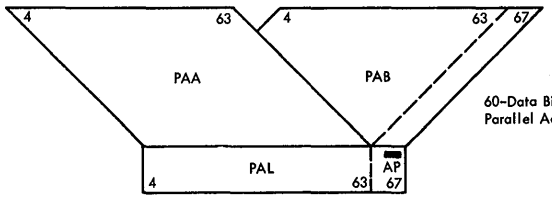
Legend

LOGIC DIAGRAMS

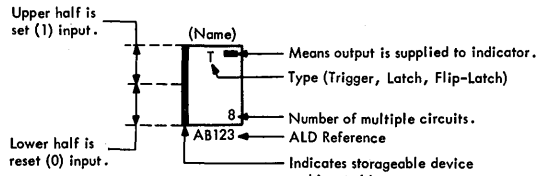


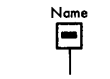


8-Data Bit (Plus Parity) Serial Adder



60-Data Bit (Plus Parity) Parallel Adder



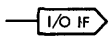


Indicator

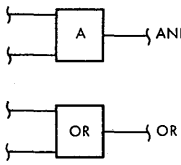
Roller Switch Indicator

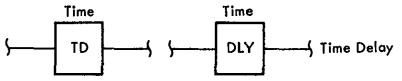
Parity Check Logic

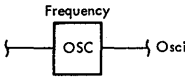
Parity Generate Logic

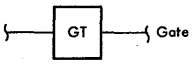


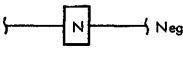
Interface
Denotes interface between two units.

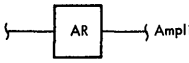


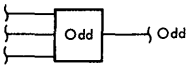


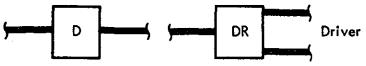


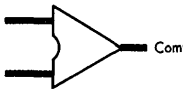




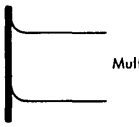








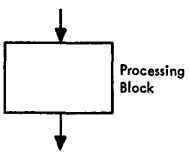
Comparator



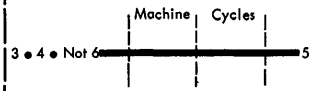
Multiple Line Transfer

Bus

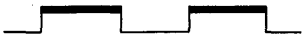
FLOWCHARTS



TIMING CHARTS



Heavy bar indicates active state. Number(s) at beginning and end of the bar identify the signal(s) (also on the same chart) that activate and deactivate this line. "Not" preceding a number means that the deactive signal conditions this line.



Waveshape
Heavy bar indicates active state.

This chapter, which discusses the 2065 CPU instructions, is divided into eight sections:

Section 1, Instruction Fetching.
Section 2, Fixed-Point Instructions.
Section 3, Floating-Point Instructions.
Section 4, Decimal Instructions.
Section 5, Logical Instructions.
Section 6, Branching Instructions.
Section 7, Status Switching Instructions.
Section 8, I/O Instructions.

Machine operation during instruction fetching and execution is controlled by ROS microprograms which are represented by CLD's. The discussions in the following sections are based upon simplified versions of the CLD's and upon upper-level, positive-logic diagrams located in the associated FEMDM.

Section 1. Instruction Fetching

Basic control for the instruction fetching (I-Fetch) operation is derived from one of four possible microprograms, depending on the format of the instruction being fetched. Each microprogram performs routines dictated by the instruction format (RR, RX, RS and SI, or SS) and is therefore common to many instructions. (The same microprogram governs the I-Fetch of RS and SI Instructions.) Subsequently, a branch is made to an appropriate microprogram for execution of a specific instruction. These individual execution sequences all terminate with a branch back to the I-Fetch microprogram to continue the sequence.

A typical microprogram sequence is shown in Figure 3-1. The correct I-Fetch microprogram to be entered upon completion of an instruction is dependent on the format of the instruction to be executed next. A test for the format of the upcoming instruction is made on the last cycle of the execution phase. The various actions performed during this last cycle (called the end operation or end-op cycle) must be thoroughly understood before undertaking a detailed analysis of each I-Fetch sequence.

BASIC END-OP CYCLE

- End-op cycle completes execution of instruction and initiates fetching of next instruction.
- End-op cycle is governed by normal end-op or branch end-op ROS word.
- Branch end op is used to speed execution of branch-type operations.

The end-op cycle is the last cycle in the execution phase. During this cycle, actions dictated by the execution phase of the instruction are completed and the fetching of the

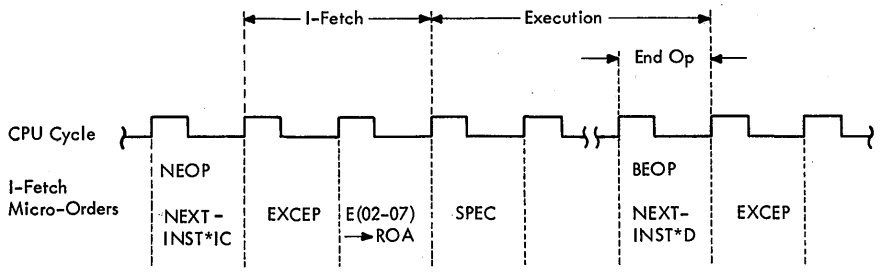


Figure 3-1. Typical Microprogram Sequence

next instruction begins. The execution phase is completed by setting the CC (if specified in the instruction) and by detecting interruptions or exceptional conditions that may have occurred during the execution phase. (The recovery microprograms are discussed after the basic end-op and I-Fetch sequences.)

The instruction fetching begins by:

1. Decoding the format of the upcoming instruction.
2. Initiating the operand fetch required by that format.
3. Establishing the correct I-Fetch sequence which is to follow.
4. Detecting the need for more instructions, and requesting new instructions from main storage when the need exists.

This discussion deals with those end-op actions that affect the subsequent I-Fetch sequence. Although instruction fetching begins during the end-op cycle, the next cycle is defined as the first I-Fetch cycle.

The setting of the CC affects the subsequent I-Fetch only when the upcoming instruction is a Branch on Condition instruction. Depending on the CC, new instructions may be requested from D (condition met) or from the IC (condition not met). The manner in which the CC is set is discussed in the specific execution sequences described in this chapter (Sections 2 through 8).

The actions performed during the end-op cycle are governed by two basic ROS end-op words: normal end-op and branch end-op. (Although they perform different functions during end-op, they perform the same functions for the subsequent I-Fetch sequence.) The normal end-op word is in control of the end operation if the address of the next instruction is specified by the IC. The next instruction is decoded from R. Conversely, the branch end-op word is in control if the address of the next instruction is specified by D. In this case, the next instruction is decoded from the SDBO (the effective R) at the start of the end operation.

The primary function of the branch end-op word is to fulfill specific timing requirements imposed upon execution of some branch instructions (see Section 6 of this chapter). Two conditions lead to a branch end-op micro-order:

1. Sometimes upon execution of a successful branch, end-op takes place before the address of new instructions (in D) has been transferred to the IC. In such cases, the branch end-op word is always in control. To establish the correct I-Fetch microprogram for the next instruction, the branch end-op word samples D(21,22) and the effective-R(0,1) bits; i.e., bits 0 and 1 of the op-code halfword to be transferred to R are sampled directly from the SDBO. Thus, the I-Fetch microprogram for the next instruction is established as soon as the instructions (specified by the branch) arrive from main storage.
2. Except for the Branch on Condition instructions, the CPU assumes that all branches are successful. Accordingly, upon predecoding a branch instruction, the CPU inhibits any IC request to refill Q and, instead, requests instructions per the branch address (in D). If, during execution, the branch proves to be unsuccessful, the instructions accessed by the D-request are not gated into Q, and the CPU must resume processing of the instructions specified by the IC. At this time it may be found that the unsuccessful branch was the last instruction in Q. Although a request per the IC is immediately generated, at least three cycles (main storage access time) must elapse before the CPU can resume normal processing. Also, because the format of the instruction is usually decoded from R(0,1), additional time would be lost if the first halfword (arriving from main storage) had to be gated to R before the I-Fetch microprogram for the instruction could be established. Under such conditions, use of the branch end-op word increases the speed in establishing the I-Fetch microprogram for the next instruction. The instruction address (in the IC) is temporarily transferred to D. Instead of sampling R(0,1) the branch end-op word samples the effective-R(0,1) to establish the correct I-Fetch microprogram immediately upon arrival of the instructions from main storage.

Prefetching of Operands During End Op

- For RR instructions, one LS register is accessed by R1 field if not a branch instruction; by R2 field if branch instruction.
- For RX, RS, SI, and SS instructions, one LS register is accessed by B-field.

During the end-op cycle, R contains the op-code halfword of the next instruction. The format of the instruction is established by sampling R(0,1), and the operand prefetch dictated by that format is initiated. The end-op cycle is completed with the 'R→E' micro-order, which transfers the op-code halfword to E at the start of the I-Fetch sequence.

The scheme for prefetching operands during end-op time is shown in Diagram 5-1, FEMDM. During this time, an LS register specified in the R or B field of the upcoming instruction is addressed and transferred to T. The desired LS register is addressed by gating the appropriate field of the instruction to LAL. Ingating to LAL is initiated by the 'NEOP' micro-order in the normal end-op word or by the 'BEOP' micro-order in the branch end-op word.

The format of the upcoming instruction is established by decoding R(0,1):

R(0,1)	Instruction Format
00	RR
01	RX
10	RS or SI
11	SS

When an RR format is decoded, a further test is performed to determine whether the upcoming instruction is a branch. If the instruction is not a branch, the R1 field [R(8–11)] is gated to LAL. For an RR branch, however, the R2 field [R(12–15)] is gated to LAL. This action is necessary because, for branch instructions, R2 specifies the LS register containing the branch address. Since in this case a storage request for new instructions must be made as soon as possible, R2 must be gated to LAL first.

When an RX, RS, SI, or SS format is decoded, a test is made to determine which of the four halfword positions in Q contains the second halfword of the upcoming instruction. The B-field of the selected halfword is then always gated to LAL. Selection of the correct halfword in Q depends upon the ROS word (branch or normal) in control of the end-op. The normal end-op word specifies that the address of the upcoming instruction is contained in the IC. In this case, IC(21,22) indicates the Q portion from which the first halfword of the instruction has been transferred to R. Consequently, these bits are decoded to select the second halfword of the instruction in Q. The branch end-op word is in control when the address of the upcoming instructions is in D. Because in this case D(21,22) points to the correct Q position, these bits are used to select the correct B-field in Q.

Prefetching of operands from LS is from locations 0–15 (decimal), unless an RR format, floating-point instruction has been predecoded. In this case, the FPR addressed by R1 is selected by forcing LAL(0) to 1. The contents of the LS register accessed during the end-op cycle are always transferred to T. This action is performed by the '→T' micro-order in the end-op word. Thus, at the start of an I-Fetch sequence, T always contains an operand (per R-field) or the base portion of an operand address (per B-field).

At the completion of an end-op cycle, the halfword containing the op code of the instruction is transferred to E (initiated by the 'R→E' micro-order in the end-op word). Thus, further operand prefetching (by the subsequent I-Fetch sequence) is performed with the op code in E.

Fetching of Instructions by End-Op Micro-Order

A test to establish whether new instructions are required is always performed during end op. If the upcoming instruction is not a branch and Q needs to be refilled, a request for new instructions is generated at end op. If the upcoming instruction is a branch, the storage request is blocked during end op.

Under certain conditions, it is possible to request new instructions from main storage one or two cycles before end-op. This action is initiated by the 'early end-op' (EEOP) micro-order, contained in the execution sequences of some instructions. All execution sequences, including those with the 'EEOP' micro-order, terminate with the end-op word.

A Q-register refill exceptional condition usually follows an end-op request for new instructions. This exceptional condition adds one cycle to the basic RR, RX and RS, and SI I-Fetch routines.

Requests During End Op

During the end-op cycle, a test is made to establish whether Q needs to be refilled with new instructions. The outcome of this test depends upon the format of the upcoming instruction, on its position in Q, and on whether it is a branch or the subject instruction of an Execute instruction.

As shown in Diagram 5-2, FEMDM, a test of the status of Q is initiated by the normal end-op (NEOP) or branch end-op (BEOP) micro-order contained in the normal or branch end-op word, respectively. Upon the decoding of the 'NEOP' micro-order, IC(21,22) is sampled to establish which halfword position in Q has been transferred to R. The same function is performed by the 'BEOP' micro-order when the address of the upcoming instruction is contained in D. In this case, D(21,22) is examined to establish which halfword in Q is to be processed next. Depending on the instruction format decoded from R(0,1), and if the upcoming instruction is neither a branch nor the subject of an Execute instruction, storage requests per the IC may be generated when the first, second, or third halfword position in Q is to be processed next.

Q-Position Transferred to R	Setting of IC(21,22) or D(21,22)	Instruction Format	Type of Request
1st	00	SS	4-cycle
2nd	01	SS	4-cycle
		RX, RS, or SI	3-cycle
3rd	10	All formats	3-cycle
4th	11	All formats	None

Q has already been refilled during the instruction being completed if bits 21 and 22 = 11; therefore, another refilling of Q is not necessary.

Requests During Early End Op

Execution sequences of some instructions contain the 'EEOP' micro-order. The function of this micro-order, which is given 1 or 2 cycles before the 'NEOP' micro-order, is to examine the instruction status in Q and

to initiate an early storage request if Q needs refilling. Requests initiated by the 'EEOP' micro-order are blocked if the next instruction to be executed is (1) a branch instruction, (2) an SS instruction, or (3) a subject of an Execute instruction.

Early requests to refill Q are generated according to conditions shown in Diagram 5-3, FEMDM. The normal end-op request is blocked when an early request is in progress. Note that the 'EEOP' micro-order can only initiate a 4-cycle request. The advantage of an early request is that the BCU will address main storage 1 or 2 cycles before end op. When initiated 2 cycles before end op, the refilling of Q does not force the Q-register refill exceptional condition if the instruction being fetched is of the RR or indexed RX format or is a shift instruction.

Selection of I-Fetch Microprogram

- Selection of I-Fetch sequence is controlled by 'NEXT-INST*IC' micro-order during normal end op or by 'NEXT-INST*D' micro-order during branch end op.
- 'NEXT-INST*IC' micro-order specifies functional ROS branch per R(0,1), IC(21,22), B = 0, and X2 = 0.
- 'NEXT-INST*D' micro-order specifies functional ROS branch per effective-R(0,1), D(21,22), B = 0, and X2 = 0.

The correct I-Fetch sequence is entered by establishing the address of the first ROS word in that sequence. This address is then placed into ROSAR so that the desired ROS control word may be obtained on the following cycle.

ROSAR(0–5) is furnished directly by the end-op word as 001000. These bits designate the address of a general I-Fetch operation about to take place. To arrive at the specific I-Fetch sequence (RR, RX, RS and SI, or SS), the bit configuration of ROSAR(6–11) must be established. The manner in which ROSAR(6–11) is established is determined by the ROS word (normal or branch) in control of the end op.

The normal end-op word contains the 'NEXT-INST*IC' micro-order specifying a 64-way functional branch. This micro-order sets ROSAR(6–11) according to the following conditions:

ROSAR Bit	Condition
6	Set if R(0) = 1
7	Set if R(1) = 1
8	Set if instruction X2 field = 0, and RX format
9	Set if instruction B field = 0, and not RR format
10	Set if IC(21) = 1
11	Set if IC(22) = 1

The above actions specify the format of the upcoming instruction, the type of further operand fetch required, and the number of counts by which IC(21,22) must be increased to select the first halfword of the instruction following it in main storage.

The registers affected by the 'NEXT-INST*IC' micro-order are shown in Diagram 5-5, FEMDM. The format of the upcoming instruction is decoded from R(0,1). For non-RR instructions, a test is made to determine whether the B-field of the instruction is equal to zero and, in the case of RX instructions, whether the X2 field is also zero.

The zero test for the B and X2 fields is necessary to establish a correct address computation by the subsequent I-Fetch routine. To increase the speed of operand prefetching, the B-field is always gated to LAL during the end-op cycle. A zero address to LAL accesses LS register 0, the contents of which may not necessarily be zero. However, the condition of B-field being zero requires that the base portion of the operand address be zero. Thus, the subsequent I-Fetch sequence selected must ignore the contents of LS register 0 (accessed by a zero B field). Similarly, in the case of the X2 field being zero, the I-Fetch sequence selected must not address LS per the X2 field. The manner in which the correct I-Fetch sequence is selected is decribed below.

Four 4-way AND's simultaneously sample the four possible B-field locations in Q. Each AND is conditioned if its corresponding four-bit input consists of all zeros. As explained previously, IC(21,22) selects the first halfword of the instruction that has been transferred from Q to R. Therefore, these bits are used as gates to select the second halfword of the instruction in Q. If, for example, the first halfword position of Q has been transferred to R and decoding of R(0,1) shows that the instruction is not of the RR format, Q(16–19) must be selected to obtain the correct B-field. When the first halfword position in Q is transferred to R, IC(21,22) is set to 00. This setting (coupled with the absence of an 'RR block' signal) selects the output of the AND that samples the correct B-field; i.e., Q(16–19). When the B-field of the instruction is found to be zero, ROSAR(9) is set to 1. This action addresses an I-Fetch microprogram that ignores the contents of the LS register accessed by the B-field.

When the upcoming instruction is of an RX format, a similar test is performed to establish whether the X2 field of the instruction, R(12–15), is equal to zero. Upon detecting a zero X2 field, ROSAR(8) is set to 1. This action dictates that the subsequent I-Fetch microprogram does not address LS per X2; i.e., E(12–15).

The ROS branch specified by the 'NEXT-INST*IC' micro-order is completed by forcing IC(21,22) into ROSAR(10,11). This action allows the first I-Fetch microinstruction to correctly update IC(21,22) and R without further testing.

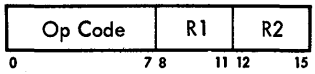
The 'NEXT-INST*D' micro-order in the branch end-op word sets ROSAR(6–11) according to the following conditions:

ROSAR Bit	Condition
6	Set if effective-R(0) = 1
7	Set if effective-R(1) = 1
8	Set if instruction X2 field = 0, and RX format
9	Set if instruction B field = 0, and not RR format
10	Set if D(21) = 1
11	Set if D(22) = 1

Note that ROSAR(6,7) is set from the effective-R rather than from R, and that ROSAR(10,11) is set from D(21,22) rather than from IC(21,22). This is done because R and IC are either still invalid or are just being set by the branch operation in progress.

BASIC RR I-FETCH

- RR format:



- Purpose:
  1. For nonbranch instructions, load 1st operand into A, B, and D. Load 2nd operand into S and T.
  2. For branch instructions, load 2nd operand into A, B, and D. Load 1st operand into S and T. Request new instructions, if needed.
  3. Set STC to 100 and ABC to 000.
- Conditions at start of I-Fetch:
  1. Instruction is transferred to E.
  2. If instruction is not a branch, 1st operand is in T; for a branch, 2nd operand is in T.

The following paragraphs describe the basic actions initiated by the ROS microprogram during I-Fetch of RR instructions. It is assumed that no interruptions or exceptional conditions were detected in the preceding end-op cycle.

The RR instructions basically require a 1-cycle I-Fetch. The actions initiated during this cycle are governed by 1 of 4 possible ROS control words selected at end-op time. Selection of the specific ROS word depends on the original position of the RR instruction in Q. This word contains the appropriate micro-orders for incrementing IC(21,22) and for transferring the first halfword of the next instruction to R. Except for these actions, the functions performed by the four ROS words are identical. Diagram 5-6, FEMDM, is a simplified flowchart of an RR I-Fetch; Diagram 5-7 shows the data registers used.

If no interruption or exceptional condition is detected, the entire RR instruction is transferred to E at the start of the I-Fetch cycle (by the 'R→E' micro-order at end op). The operand prefetching, initiated at end op, is then continued. The order in which operands are prefetched depends on whether the instruction is a branch:

1. For nonbranch instructions, the first operand (accessed during end op) is transferred from T via the parallel adder to A, B, and D. The second operand is then addressed by gating E(12–15) to LAL. When the second operand is accessed, it is loaded into S and T.
2. The above order is reversed for branch instructions; i.e., the second operand (accessed during end op) is placed into A, B, and D while the first operand is placed into S and T. A storage request per the branch address is generated subject to the conditions shown in Diagram 5-6.

The correct execution sequence is entered by establishing the address of the first ROS word in that sequence. This address is determined by sampling the instruction op code from E(2–7) by means of the 'E(02–07)→ROA' micro-order. As stated earlier, this description of RR I-Fetch applies only when no exceptional conditions or interruptions are present. The ROS word governing the I-Fetch cycle always contains the 'EXCEP' micro-order, which can override the functional branch per the instruction op code. Therefore, the branch to the first execution cycle occurs only when there are no interruptions or exceptional conditions.

In addition to prefetching the operands, the I-Fetch ROS word contains appropriate micro-orders to increment IC(21,22) and to transfer the first halfword of the next instruction to R. IC(21,22) is set one count higher ('X→IC' micro-order) to point at the next instruction. The first halfword of the next instruction is transferred to R by the 'QXX→R' micro-order.

Included in the first RR I-Fetch word is the 'RESET' micro-order, which causes the following actions:

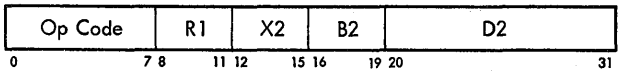
1. During I-Fetch of branch instructions, initiates the request for new instructions (see Diagram 5-4) and gates E(8–11), instead of E(12–15), to LAL.
2. Resets all STAT's and Edit-instruction controls.
3. Sets STC to 100 and ABC to 000.
4. Forces LAL(0) to 1 for floating-point instructions, causing the FPR's to be addressed.
5. Sets the 'stop' trigger if operating at the instruction-step rate.

Note that IC(21,22) is not advanced, the Q-to-R transfer is not effected, and unsuccessful branch-on-condition requests are not generated if the 'execute in progress' trigger is set. The set state of this trigger indicates that the current I-Fetch is for a subject

instruction of an Execute instruction. Therefore, the address of the RR instruction is specified by D and not by the IC.

BASIC RX I-FETCH

- RX format:



- Purpose:
  1. Compute address of 2nd operand and transfer to D; request 2nd operand from main storage, if necessary.
  2. Transfer 1st operand to S and T.
- Conditions at start of I-Fetch:
  1. 1st halfword of instruction is transferred to E; 2nd halfword is in Q.
  2. Contents of LS register specified by B2 are transferred to T.

The following paragraphs describe the basic actions initiated by the ROS microprogram during I-Fetch of an RX instruction. It is assumed that no interruption or exceptional conditions were detected in the preceding end-op cycle.

The RX instructions basically require a 1- or 2-cycle I-Fetch. The actions initiated during the first I-Fetch cycle are governed by 1 of 16 possible ROS control words selected at end-op time. This selection depends on whether the B2 and/or X2 fields of the instruction are zero, and on the original position of the RX instruction in Q. The zero test establishes four separate cases for the I-Fetch routine: (1) B2 = 0 and X2 = 0, (2) B2 ≠ 0 and X2 = 0, (3) B2 = 0 and X2 ≠ 0, (4) B2 ≠ 0 and X2 ≠ 0. The first two cases require a 1-cycle I-Fetch; the last two, a 2-cycle I-Fetch. Each of the above four routines contains appropriate micro-orders for incrementing IC(21,22) and transferring the first halfword of the next instruction to R. Consequently, a four-way branch is inherent in each routine, depending on the previous IC(21,22) setting; i.e., 00, 01, 10, or 11.

Diagram 5-9, FEMDM, is a simplified flowchart of an RX I-Fetch; Diagram 5-10 shows the data registers used. If no interruptions or exceptional conditions are detected, the op-code halfword of the RX instruction is transferred to E at the start of I-Fetch (initiated by the 'R→E' micro-order at end op). The operand prefetch routine is then continued with the first halfword of the instruction in E and the second halfword in Q. The I-Fetch of non-indexed RX instructions (X2 = 0) is described first.

The 'X→IC' micro-order issued by the first I-Fetch word sets IC(21,22) two counts higher. The first halfword of the next instruction is transferred to R by the 'QXX→R' micro-order if it is now in Q; that is, if IC(21,22) did not equal 10 at the start of I–Fetch. If the instruction is not indexed, the address of the second operand is obtained by adding D2 to the base address. The contents of the LS register per B2 are placed into T at the start of I-Fetch. If B2 was found to be zero during end op, the contents of T are ignored, and the appropriate D2 field in Q is selected and routed to D via the parallel adder. However, if B2 ≠ 0, the contents of T and the D2 field are gated simultaneously to the parallel adder, and the resultant sum is transferred to D. A 3-cycle storage request for the second operand is made from D if the following conditions do not exist:

1. A Q-register refill exceptional condition is in progress.
2. The instruction is Store Halfword, Store Character, or Load Address.
3. The instruction is an unsuccessful Branch on Condition. (A request for new instructions is issued from the IC, if necessary.)

The first operand is obtained from LS per R1 and transferred to S and T. At the completion of the I-Fetch cycle, a branch is made to a specific execution sequence as determined by the 'E(02–07)→ROA' micro-order.

In the case of indexed RX instructions (X2 ≠ 0), two cycles are required to complete the I-Fetch routine (Diagram 5-11, FEMDM). During the first cycle, D2 is added to the contents of T (if B2 ≠ 0) and the result is temporarily stored into B. The LS register specified by X2 is then accessed, and its contents are placed into T. The contents of T and B are added during the second cycle, and the sum (second operand address) is transferred to D. The conditional storage request is now made. After the first operand is obtained from LS and placed into S and T, a branch per the instruction op-code is made to enter the correct execution sequence.

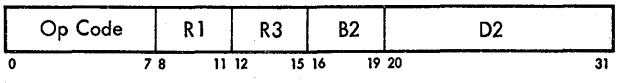
The 'RESET' micro-order:

1. Resets all STAT's and Edit instruction controls.
2. Resets STC and ABC to 000.
3. Initiates any necessary storage requests for branch instructions and for subject instructions of the Execute instruction.
4. Forces LAL(0) to 1 for floating-point instructions, causing the FPR's to be addressed.
5. Sets the 'stop' trigger if operating at the instruction-step rate.

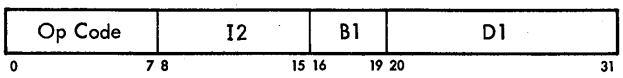
If the instruction being fetched is the subject of an Execute instruction ('execute in progress' trigger is set), the incrementing of IC, the Q-to-R transfer, and the unsuccessful Branch on Condition requests to refill Q are inhibited.

BASIC RS AND SI I-FETCH

- RS format:



- SI format:



- Purpose:
  1. Add contents of LS register specified by B-field to D-field; place result into D.
  2. Request operand from main storage, if necessary.
  3. For RS instructions, load 1st operand into S and T. (Contents of S and T are ignored for SI instructions.)
- Conditions at start of I-Fetch:
  1. 1st halfword of instruction is transferred to E; 2nd halfword is in Q.
  2. Contents of LS register specified by B-field are transferred to T.

The following paragraphs describe the basic actions initiated by the ROS microprogram during I-Fetch of RS and SI instructions. It is assumed that no interruptions or exceptional conditions were detected in the preceding end-op cycle.

The RS and SI instructions basically require a 1-cycle I-Fetch. The actions initiated during this cycle are governed by 1 of 8 possible ROS control words selected at end-op time. This selection depends on whether the B-field of the instruction is zero, and on the original position of the instruction in Q. The zero test establishes two distinct I-Fetch routines: (1) B = 0 and (2) B $\neq$ 0. A four-way branch is inherent in each routine, depending on the previous IC(21,22) setting; i.e., 00, 01, 10, or 11.

Diagram 5-13, FEMDM, is a simplified flowchart of RS and SI I-Fetch. If no interruptions or exceptional conditions are detected, the halfword containing the op-code of the RS or SI instruction is transferred to E at the start of I-Fetch (initiated by the 'R→E' micro-order at end op). The operand prefetch routine is then continued with the first halfword of the instruction in E and the second halfword in Q.

The LS register specified by the B-field (B1 or B2) is accessed during end op, and its contents are placed into T at the start of I-Fetch. If the B-field was found to be zero during end op, the I-Fetch routine ignores the contents of T, selects the appropriate D-field (D1 or D2) in Q and routes it to D via the parallel adder. If the B-field is not zero, the contents of T and the D-field are gated simultaneously to the parallel adder and the sum is then transferred to D. A 3-cycle storage request for the second operand is then made from D if the following conditions do not exist:

1. A request to refill Q was generated during the previous execution segment; i.e., IC(21,22) = 01 or 10. For this case, the ROS micro-order is not contained in the I-Fetch word.
2. The E-register contains a shift, Store Multiple, Move (MVI), Test and Set, or I/O instruction. For this case, the 'D sync' latch is prevented from being set.
3. The E-register contains a branch-on-index (BXH, BXLE) instruction. For this case, the 'RESET' micro-order resets the '3-cycle request' trigger, causing a 4-cycle storage request to be made from D regardless of IC(21,22).

Upon loading D with the second operand address, the I-Fetch routine proceeds to set IC(21,22) two counts higher, to transfer the first halfword of the next instruction to R if it is in Q, and to establish the first ROS control word for the execution phase. Fetching of the first operand per E(8–11) is meaningful only for RS instructions. For SI instructions, E(8–11) contains a portion of the immediate operand. Since a common ROS control word governs the I-Fetch of both formats, E(8–11) is always gated to LAL; the contents of the LS register thus accessed are placed into S and T. However, the subsequent execution sequences for SI instructions ignore the contents of S and T.

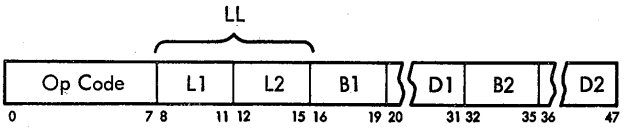
In addition to causing a 4-cycle storage request during the I-Fetch of a branch-on-index instruction, the 'RESET' micro-order:

1. Resets all STAT's and Edit-instruction controls.
2. Resets STC and ABC to 000.
3. Sets the 'stop' trigger if operating at the instruction-step rate.

If the instruction being fetched is the subject of an Execute instruction ('execute in progress' trigger is set), the incrementing of IC and the Q-to-R transfer is inhibited.

BASIC SS I-FETCH

- SS format:



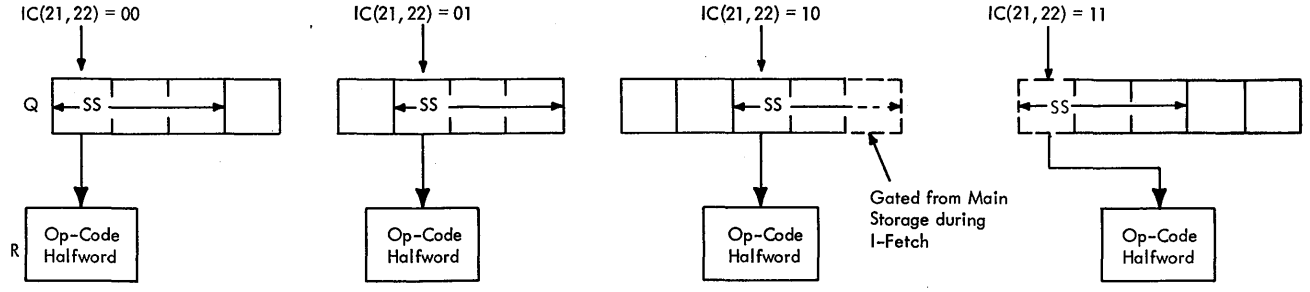
- Purpose:
  1. Transfer op-code halfword of next instruction to R; update IC and place into LSWR.
  2. Transfer computed address of 1st operand (destination) per instruction class to D; request destination operand from main storage (gated into CPU at start of 2nd execution cycle).
     a. Lowest destination address for logical instructions = base address (per B1) + D1.
     b. Highest destination address for decimal instructions = base address (per B1) + D1 + L1.

3. Transfer computed address of 2nd operand (source) to IC and T. Lowest source address = base address (per B2) + D2.
4. Perform ASC test (and invalid instruction address test if complete instruction is in Q).

- Conditions at start of I-Fetch:
  1. 1st halfword of instruction is transferring to E, 2nd halfword is in Q, 3rd halfword is in Q if IC(21,22) ≠ 10 (otherwise 3rd halfword is gated to Q during 4th cycle of I-Fetch).
  2. Base address (per B1) is in T.
  3. Q refill is not in progress if IC(21,22) = 11.

The I-Fetch of SS instructions differs considerably from the I-Fetch routines described thus far (RR, RX, RS, and SI). Differences arise from three characteristics of the SS format: (1) the SS format is three halfwords long, (2) an SS instruction always stores the results into main storage, and (3) two main storage addresses are specified.

As previously stated, requests to refill Q are generated before the CPU runs out of instructions. In describing the I-Fetch microprograms used for RR, RX, RS, and SI formats, it was assumed that the instruction to be executed was contained in Q. Because of the manner in which storage requests for instructions are generated, the assumption is valid for all 1- and 2-halfword instructions. For SS instructions, however, the I-Fetch routine may sometimes begin while the last halfword of the instruction is still in main storage. Figure 3-2 shows all possible locations that the SS instruction may assume in Q and the manner in which storage requests are generated for more instructions. Storage requests for SS instructions are generated (at end-op time) when IC(21,22) is set to 00, 01, or 10. If IC(21,22) = 00 or 01, the entire instruction is in Q at the start of I-Fetch. However, if IC(21,22) = 10, the last halfword of the instruction will arrive from main storage on the fourth cycle of I-Fetch. Consequently, processing of the third halfword of the instruction cannot



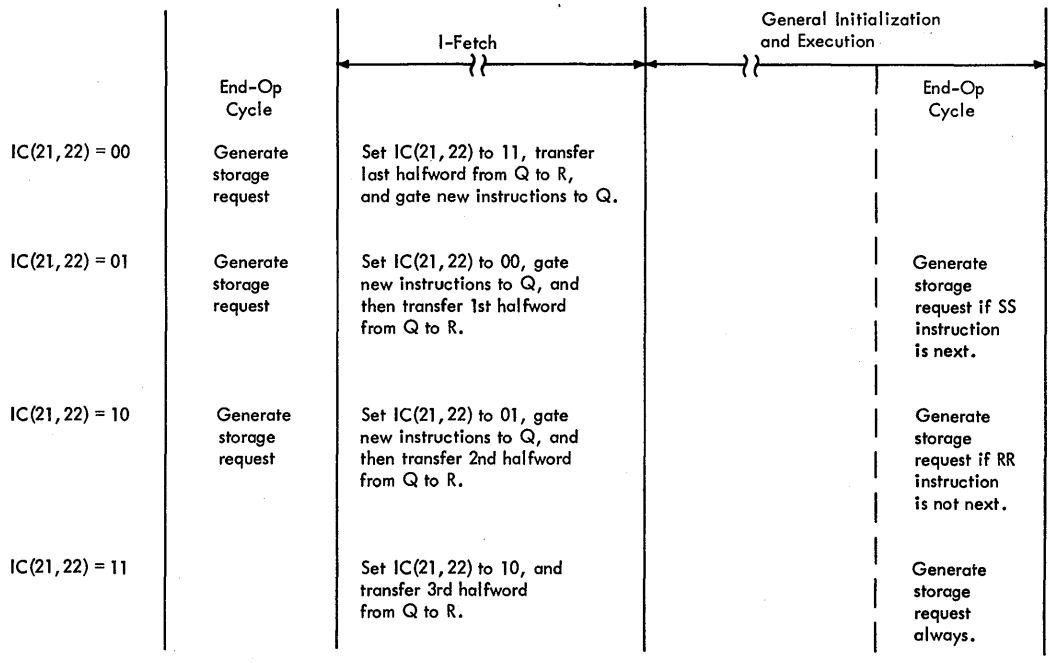


Figure 3-2. Basic Sequencing for SS Instruction

start until Q is refilled. Finally, if IC(21,22) = 11, at end op, Q has been refilled as a result of a previous end-op cycle and Q(0–31) contains the balance of the upcoming SS instruction.

An SS instruction operates on two operands obtained from main storage and stores the result into the same location from which the first operand was obtained. Therefore, the address of the first operand is also the destination address; the address of the second operand is commonly referred to as the source address. The first and second operand addresses are calculated in a manner similar to that of two-halfword instructions. The address of the first operand is computed first and loaded into D, and a storage request for the operand is made. The partial address of the second operand is then computed while the contents of the IC are transferred to the LSWR. The partial second operand address is loaded into the IC. After completing the I-Fetch routine, a General Initialization Sequence, GIS, is performed, after which control is transferred to the execution phase. (During GIS, the calculation of the second operand address is completed, and a storage request issued, if necessary.) Upon execution of the instruction, results are stored into main storage per the address in D (first operand or destination address).

Address Store Compare (ASC) Test

- Main storage address where data is to be stored is compared with address of current instructions.
- Comparison is made whenever data is stored into main storage.
- If data is stored at instruction address, 'PSC' trigger is set to indicate that instructions in Q must be refetched.
- For SS instructions, ASC test is performed during I-Fetch. Lower and upper limits of destination address are compared with instruction address.

An ASC test must be made each time the CPU stores data into main storage. This test compares the destination address of the data with the current instruction address. If it is found that both addresses are the same, the 'program store compare' (PSC) trigger is set, indicating a need to refetch instructions; i.e., the instructions currently in Q must again be obtained from main storage because the next instruction to be executed may have been modified by the data just stored.† For all *but* SS instructions, the ASC test is made during the execution phase whenever a store operation is performed. Because, in the case of SS instructions, a store operation is always implied, an ASC test has been incorporated in the SS I-Fetch microprogram.

†The refetch routine is initiated if the result of a comparison of the destination and IC addresses falls within a 16-byte safety margin: Destination address (in D) = IC address ± 16 bytes. Thus, instructions in Q may not necessarily be modified by the store operation.

For SS instructions, the ASC test must determine that instructions (currently in Q) were not obtained from a region defined by the upper and lower limits of the destination address for data. This test is made in two steps, as illustrated in Figure 3-3. The first step determines whether the lower limit of the destination address is above the instruction address in the IC. When the lower limit is above, the upper limit must also be above the IC, and the 'PSC' trigger is not set. This condition indicates that current instructions (in Q) cannot be affected by the subsequent store operations. However, if the lower limit of the destination address is found to be below the IC, the 'PSC' trigger is set and a further test must be made to establish that data will not be stored in the instruction path. The last step compares the upper limit of the destination address with the IC. If the IC is found to be above the upper limit, the 'PSC' trigger is reset. In such cases the subsequent store operations will not extend to the IC location. On the other hand, if the IC points below the upper limit, the 'PSC' trigger remains set, indicating that the subsequent store operations may affect the next instruction. Consequently, after execution of the instruction that caused the PSC condition, an exceptional condition microprogram is initiated to refetch the instructions in Q. The details of the refetch microprogram are described under "Program Store Compare Exceptional Condition".

I-Fetch Microprogram

- If request for new instructions has been generated at end op, I-Fetch routine requires 7 cycles; if not, I-Fetch requires 6 cycles.
- Setting of IC(21,22) at end op determines manner in which I-Fetch is performed.

The following paragraphs describe the basic actions initiated by the ROS microprogram during I-Fetch of SS instructions. It is assumed that no interruptions or exceptional conditions were detected in the preceding end-op cycle.

The first halfword of the SS instruction is transferred from R to E at the start of I-Fetch. The operand prefetch (initiated at end op) is then continued with the first halfword in E, the second halfword in Q, and the third halfword in Q [or in main storage if IC(21,22) = 10], as shown in Diagram 5-14, FEMDM.

The SS instructions require a 7- or 6-cycle I-Fetch. The actions initiated during the first I-Fetch cycle are governed by 1 of 8 possible ROS control words selected at end-op time. This selection depends on whether the B1 field of the instruction is zero and on the setting of IC(21,22). The setting of IC(21,22) establishes four distinct cases for the SS I-Fetch microprogram:

1. When IC(21,22) = 00, a 4-cycle storage request to refill Q is generated at end op. Because Q(48–63) contains

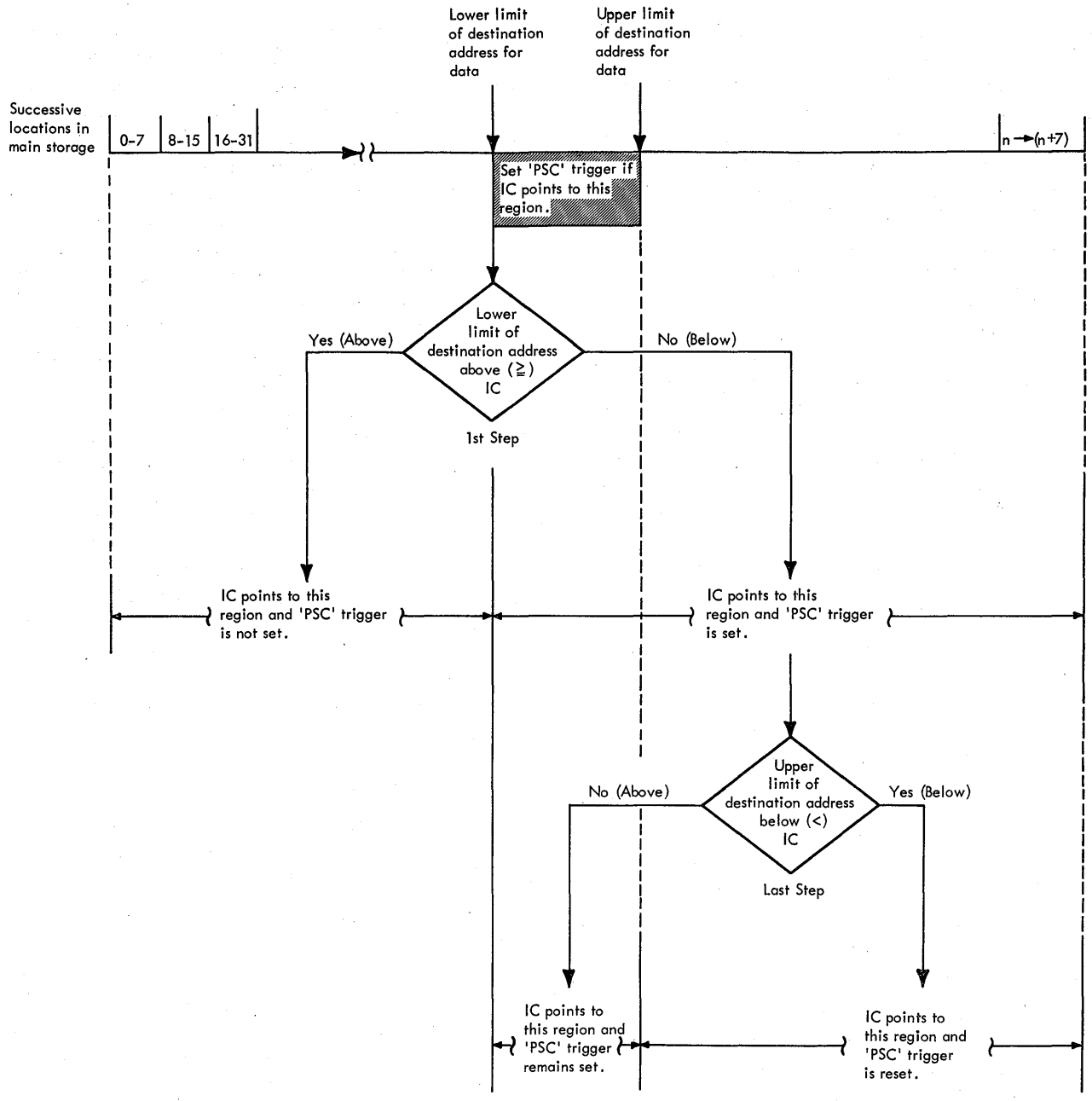


Figure 3-3. ASC Test for SS Instructions

the op-code halfword of the next instruction, the I-Fetch routine must transfer this halfword to R before the next doubleword arrives from main storage. Also, the I-Fetch routine must gate the new instructions to Q at the correct time.

2. When IC(21,22) = 01, a 4-cycle request to refill Q is generated. Because Q contains no new instructions, the I-Fetch routine must wait until Q is refilled and then transfer the first halfword of the next instruction from Q to R. New instructions must be transferred to Q at the correct time.
3. When IC(21,22) = 10, a 3-cycle request to refill Q is generated. Because the third halfword of the SS instruction is in main storage, processing of this halfword is delayed until Q is refilled. New instructions must be transferred to Q at the correct time, after which the next instruction may be gated to R.
4. When IC(21,22) = 11, a storage request is not generated because Q has been refilled as a result of the previous end op. Thus the I-Fetch routine is not concerned with transferring new instructions to Q.

During the first I-Fetch cycle of all SS I-Fetch routines, the lower limit of the destination address is calculated and placed into D, and IC(21,22) is advanced to indicate the first halfword of the next instruction. At the start of I-Fetch, T contains the base portion of the destination address (LS contents per B1). If B1 ≠ 0, the contents of T are added to the appropriate D1 field and the sum is transferred to D. If, however, B1 = 0, the contents of T are ignored and only the D1 field is routed via the parallel adder to D. After the first I-Fetch cycle, D contains the lower limit of the destination address.

The I-Fetch sequence after the first cycle differs for each setting of IC(21,22). The following paragraphs describe the I-Fetch control for each setting.

I-Fetch Control If at End Op IC(21,22) = 00

- STAT D is set to indicate B2 = 0.
- Branch per instruction is made to establish starting destination address.
- If 'execute in progress' trigger is reset, IC(20) is advanced by 1; if trigger is set, IC is not incremented.
- Instruction address is stored into LSWR.
- I-Fetch requires 7 cycles.

If at end op IC(21,22) = 00, Q contains the remaining two halfwords of the SS instruction and Q(48–63) contains the halfword of a new instruction. Q(48–63) is transferred to R by the second ROS word in the I-Fetch sequence. This word also accesses the LS register specified by B2 and initiates the ASC test. The LS register is accessed by gating the B2 field [Q(32–35)] to LAR and transferring the LS contents to S. (STAT D records the condition when B2 = 0 and is interrogated when the source address is computed.) The ASC test is initiated by subtracting the lower limit of the destination address (contained in D) from the IC. If the difference is equal to or greater than zero, the 'PSC' trigger is set to indicate that the destination address may overlap the instruction path.

A branch is made per the instruction op code to the third I-Fetch cycle. This branch establishes the manner in which the upper limit of the destination address must be obtained:

1. For decimal instructions, the L1 field [E(8–11)] is transferred to the parallel adder, where it is added to the contents of D.
2. For logical instructions, the LL field [E(8–15)] is added to the contents of D.

Also, depending on the instruction, either the upper or the lower limit of the destination address becomes the starting point from which operands are to be processed. For all decimal instructions, operand processing starts from the upper limit of the destination address and proceeds toward the lower limit. Conversely, for all logical instructions, operand processing starts from the lower limit and proceeds toward the upper limit. Thus, when a decimal instruction is decoded, the upper limit of the destination address is transferred to D and becomes the starting address from which the first operand will be requested. In the case of logical instructions, the original contents of D (lower limit of destination address) are not changed. Because the upper limit of the destination address is required to complete the ASC test, this address is temporarily stored into T.

The fourth ROS word in the I-Fetch sequence initiates calculation of the source address and gating of new instructions to Q, and requests the first operand from main storage (per the destination address in D). At the start of the fourth cycle, STAT D is tested to establish whether the B2 field of the instruction is zero. If B2 is not zero (STAT D not set), the contents of S (where LS contents per B2 have been placed) and the appropriate D2 field in Q are simultaneously gated to the parallel adder, and the sum is temporarily stored into B. If B2 = 0 (STAT D set), however, the contents of S are ignored and only the D2 field is placed into B. With the completion of the above actions, all halfwords in Q have been processed and Q is refilled with new instructions. The fourth ROS word also initiates a 4-cycle request per D for the destination operand. A ROS micro-order in the first GIS word gates the destination operand into the CPU.

The fifth ROS word in the I-Fetch sequence completes the ASC test. The upper limit of the destination address (contained in T) is subtracted from the IC contents. If the difference is greater than zero, the 'PSC' trigger is reset. This condition indicates that the instruction address is above the highest main storage location into which data is to be stored. However, if the difference (IC minus T) is less than zero, the 'PSC' trigger remains set. The set state of the 'PSC' trigger initiates the program store compare exceptional condition after execution of the current SS instruction.

The sixth ROS word in the I-Fetch sequence usually increments the IC by 8 and then stores the updated address into the LSWR. Following each storage request from the IC, the contents of the IC must be updated by 8 to obtain the address of the next doubleword location from which subsequent instructions will be requested. An exception to this rule occurs if the SS instruction currently being processed is the subject of an Execute instruction. Because, upon execution of the subject instruction, the instructions previously contained in Q must be refetched from the main storage address specified in the IC, the contents of the IC are not updated. Thus, before storing the IC into the LSWR, a test must be made to determine whether an Execute instruction is in progress. This test is accomplished by examining the status of the 'execute in progress' trigger.

6. Main storage requests. Requests for new instructions because of a predecoded branch instruction are inhibited by blocking the decoding of the 'RESET' micro-order. Storage requests for operands are inhibited by blocking the decoding of the 'MS-REQ*D-3' micro-order.
7. Initiation of the invalid instruction address test.

The 'block I-Fetch' trigger is not set by detection of the invalid instruction address test or Q-register refill exceptional conditions. In the first case, because addressing is at fault, the next address must be computed and retained for subsequent evaluation. In the second case, the trigger is not set because this action would inhibit the purpose of the Q-register refill exceptional condition.

The 'block I-Fetch' trigger is reset by the '0→STAT D' micro-order issued by the first word in the recovery microprogram.

Timer Exceptional Condition

- Initiated by 'time clock step' trigger.
- Decrements timer in location 50 (hex) per power-line frequency.
- Sets 'time clock at limit' trigger if timer value is less than zero, thus causing external interruption.

The timer value is stored in permanent main storage location 80, decimal (50, hex). It is stepped at a rate determined by the input power frequency. If the input power frequency is 60 Hz, the timer is stepped each 16.67 ms; if 50 Hz, the timer is stepped each 20 ms. The timer exceptional condition detection scheme is shown in Diagram 5-17, FEMDM.

Each time the input power swings to a positive peak, the 300-ns singleshot† generates a signal to set the 'sample pulse' trigger if the DISABLE INTERVAL TIMER switch is not activated. The output of this trigger sets the 'time clock step' trigger, provided that the CPU is not in the end-op cycle. Once set, the 'time clock step' trigger initiates the timer update sequence.

The need to inhibit setting of the 'time clock step' trigger at end op is twofold: (1) the timer exceptional condition is asynchronous with respect to program execution, and (2) it has the highest priority. Because priority is established at end op, sampling the timer exceptional conditions at this time could result in a priority conflict with a pending interruption or exceptional condition. Thus, use of two triggers ensures that timer priority is present before entry into end op; i.e., if the need to update the timer arises at end op, this event is recorded by the 'sample pulse' trigger, and the timer update sequence is initiated on the next end-op cycle.

Once the 'time clock step' trigger is set, it inhibits the priority circuits of all other interruptions and exceptional conditions. When end op occurs, the trigger output is gated to alter the subsequent I-Fetch by inhibiting the loading of E from R. D is set to 50 (hex) to obtain the timer value from main storage and, if an Execute instruction is not in progress, the 'block I-Fetch' trigger is set. The 'EXCEP' micro-order in the first I-Fetch cycle detects timer priority and forces address 014 (hex) into ROSAR.

The address forced into ROSAR by the timer exceptional condition is used to enter a timer update microprogram. A flowchart of the hardware operations just explained and of the timer update microprogram is shown in Diagram 5-17, FEMDM. The microprogram resets the 'block I-Fetch' trigger and issues a 3-cycle storage request per D to fetch the timer value. While the fetch is in progress, any protection checks from storage are ignored. After the fetch, the 32-bit timer value is loaded into A and then decremented by 5 or 6, depending on 60- or 50-Hz power input frequency, respectively.† The updated timer value is placed into S and T, and then stored per the D-address (location 50, hex). Before storing, however, the timer value is sampled to see if it has been decremented to less than 0. If this condition exists, the 'time clock at limit' latch is set to request an external interruption on the following end op.

CPU Store In Progress Exceptional Condition

Regarding the protection violation, a unique situation occurs if at end op a store operation is also in progress. In this case, detection of an interruption or exceptional condition could cause a failure to detect a "late" storage protect violation if it occurs. To prevent this situation, a special circuit is provided to test for a store-in-progress condition at end op (Diagram 5-18, FEMDM). If an exceptional condition to I-Fetch or a Load PSW instruction has been detected while a store operation is in progress, this circuit forces a microprogram that provides a 2-cycle delay to allow recording of a possible protection check and establishing the correct priority for the subsequent program interruption.

Machine Check Interruption

- Follows log-out microprogram.
- Clears interruption code, resets STAT H, enters common interruption routine.

†The singleshot output is sent to the I/O channel to provide an automatic restart; it is also used by the pulse-mode controls (see Chapter 6, Section 1). Also, if the Multisystem feature is installed, it is used for timing external resets and for detecting system inactivity.

†The variation of decrement value is determined by a pluggable card. This card (type 3388) is inserted into position 01BC2F2, AP811, when 60-Hz input power is used.

A machine check interruption is initiated only by a log-out operation. The machine check interruption detection scheme is shown in Diagram 5-19, FEMDM.

When the log-out operation (initiated by the Diagnose instruction, or a machine check error coupled with the machine check mask-bit being on) is about to be concluded, the '1→MCH-CK-TRP' micro-order causes the 'machine check interrupt' trigger to be set. Once set, the trigger blocks any new machine checks from initiating another log-out operation. This trigger also establishes machine check priority, and the operation then waits for the logout to finish, signified by an end op, before continuing. At end-op time, the set state of the 'machine check interrupt' trigger forces D to 30 (hex) and, if an Execute instruction is not being concluded, sets the 'block I-Fetch' trigger, inhibiting most of the I-Fetch actions.

The 'EXCEP' micro-order in the first I-Fetch cycle detects machine check priority and forces address 00C (hex) into ROSAR. The ROSAR address causes a branch to an interruption microprogram that stores the old PSW per the D-address. Subsequently, the microprogram loads the new PSW per the address in D + 40 (hex). The operation then proceeds as directed by the new PSW.

A flowchart of the hardware operations just explained and of the beginning of the machine check microprogram is shown in Diagram 5-19. The microprogram starts by gating the interruption code to S(16–31) to begin forming the old PSW in ST. This is done by setting S(16–31) to all 0's, because the machine check interruption code, PSW(16–31), is always 0. The 'block I-Fetch' trigger and STAT H are reset. (STAT H is reset to modify the subsequent common interruption routine for specific machine check actions.) At this point, the machine check interruption microprogram enters the common interruption routine.

Program Interruption

- Initiated by 'program interrupt' latch.
- Value in interrupt code triggers is transferred to S(16–31) by 'priority 1' latch.
- Sets STAT H, enters common interruption routine.

A program interruption results from improper conditions arising during the processing of data or instructions. Generally, these improper conditions can be described as errors in programming. When any of these conditions are detected, they cause a value to be placed into the four Interrupt Code triggers. The value inserted reflects the condition responsible for the interruption. The conditions that cause a program interruption and their corresponding Interrupt Code trigger settings are shown in FEMDM Diagrams 5-20, 5-21, and 5-22.

Once any Interrupt Code trigger has been set, the 'program interrupt' latch shown in Diagram 5-22 is set. (The 'INTRP X-branch' micro-order samples this latch to modify the execution sequence of instructions and the I-Fetch of SS format instructions. Once this latch is set, further program violations are lost with the exception of a "late" protection check.) This latch establishes priority for the program interruption by blocking the priority circuits of lower-priority interruptions and exceptional conditions.

At end op, the output of the 'program interrupt' latch is gated to force D to 28 (hex), to set the 'priority 1' latch, and, if an Execute instruction is not in progress, to set the 'block I-Fetch' trigger. The 'EXCEP' micro-order in the first I-Fetch cycle detects program interruption priority and forces address 00A (hex) into ROSAR. This ROSAR address causes a branch to an interruption microprogram to store the old PSW per D. The 'priority 1' latch causes the values in the Interrupt Code triggers (program-interruption code) to be gated to S as part of the old PSW. Subsequently, the microprogram loads the new PSW per the address in D + 40 (hex). CPU operation then proceeds as dictated by the new PSW.

Diagram 5-22 is a flowchart of the hardware operations just described and of the beginning of the program interruption microprogram. The microprogram starts by gating the interruption code from PSW(16–31) to S(16–31). The 'block I-Fetch' trigger and STAT D are reset, and STAT H is set. (STAT H modifies the subsequent common interruption routine for specific program interruption actions.) At this point, the program interruption microprogram enters the common interruption routine.

Supervisor Call Interruption

- Initiated by 'supervisor call' trigger, which is set by preceding Supervisor Call instruction.
- E(8–15), which contains interruption code, is transferred to S(24–31).
- Sets STAT H, enters common interruption routine.

The supervisor call interruption results from execution of the Supervisor Call instruction. Its basic purpose is to initiate a branch to the supervisor program. When the priority of the interruption is established, an address is forced into ROSAR and into D. The address in ROSAR causes the operation to branch to a microprogram which stores the old PSW in the address forced into D and fetches a new PSW from the address in D + 40 (hex). This new PSW places the CPU into the Supervisor state.

Diagram 5-23, FEMDM, shows how the '1→ INTREQ-TGR' micro-order tests for a Supervisor Call instruction and sets the 'supervisor call' trigger. This trigger is reset if the 'interrupt code 4' trigger is set. Because all program interruptions would have been handled before executing the Supervisor Call instruction, the 'interrupt code 4'

trigger can be set now only by a "late" protection check. Therefore, performance of the supervisor call interruption is suppressed and a program interruption occurs in its place. If the 'interrupt code 4' trigger is not set, then the 'supervisor call' trigger is not reset.

The 'supervisor call' trigger gates E(8–15) to S(24–31) to begin assembling the old PSW. (The 'supervisor call' trigger also sets the 'priority 1' latch; because the supervisor call and program interruptions cannot be pending at the same time, no conflict results from both setting an interruption priority code of 01.)

Diagram 5-23 is a flowchart of the hardware operations just described and of the beginning of the supervisor call interruption microprogram. The microprogram starts by gating the interruption code from PSW(16–31) to S(16–31). The 'block I-Fetch' trigger and STAT D are reset, and STAT H is set. At this point, the supervisor call interruption microprogram enters the common interruption microprogram routine.

External Interruption

- Remains pending if external mask bit, PSW(7), is not set.
- Initiated by setting of 'time clock at limit' latch, depression of INTERRUPT pushbutton, or recognition of 'external signal in' bus.
- 'Time clock at limit', 'console signal', and 'external signal-2' through '-7' triggers are transferred to S(24–31).
- Sets STAT H, enters common interruption routine.

An external interruption is caused by one of the following if the external bit of the PSW system mask is a 1:

1. The set state of the 'time clock at limit' latch (refer to "Timer Exceptional Condition").
2. The depression of the INTERRUPT pushbutton on the system control panel.
3. The recognition of any signal on the 'external signal in' bus of the Direct Control feature. The external interruption circuits and a flowchart of the initiation of the external interruption microprogram are shown in Diagram 5-24, FEMDM.

Once priority for the external interruption is established during end op, D is forced to 28 (hex), the 'priority 2' trigger is set, and, if an Execute instruction is not in progress, the 'block I-Fetch' trigger is set. The 'EXCEP' micro-order in the first I-Fetch cycle detects external interruption priority and forces address 006 (hex) into ROSAR. The ROSAR address causes a branch to an interruption microprogram to store the old PSW per D. The 'priority 2' trigger causes the contents of the eight signal triggers (the interruption code) to be gated to S(24–31) as part of the old PSW. Subsequently, the microprogram loads the new PSW per the address in D + 40 (hex). CPU operation then proceeds as dictated by the new PSW.

The external interruption microprogram starts by gating status of the 'time clock at limit', 'console signal' and External Signal triggers to S(24–31). S(16–23) is reset to zero, and correct (odd) parity is assigned to S(16–31). The 'block I-Fetch' trigger and STAT D are reset, and STAT H is set. (STAT H is set so that the common interruption routine skips micro-orders pertaining to the machine check interruption.) At this point the microprogram enters the common interruption routine.

I/O Interruption

- Remains pending if associated channel mask bit, in PSW(0–7), is not set.
- Channel 0 has highest interruption priority, followed in order by channels 1–6.
- 3-bit channel address and 8-bit unit address are transferred to S(21–31).
- Sets STAT H, enters common interruption routine.

An I/O interruption results from the reception of a simplexed 'interruption request' signal from a channel if the system mask bit for that channel is a 1. All I/O channels in the system compete for the interruption priority. The multiplexer channel (channel 0) is assigned the highest priority followed in order by selector channels 1 through 6 (maximum). The I/O interruption circuits and a flowchart of the initiation of the I/O interruption microprogram are shown in Diagram 5-25, FEMDM. The 'channel X interrupt request' triggers are set at the start of every non-end-op cycle. At the start of end op, the highest-priority trigger that is set resets all the others.

Once priority for an I/O interruption is established during end op, D is forced to 38 (hex), the 'priority 1' and 'priority 2' triggers are set, and, if an Execute instruction is not in progress, the 'block I-Fetch' trigger is set. The 'EXCEP' micro-order in the first I-Fetch cycle detects I/O interruption priority and forces address 006 (hex) into ROSAR. This ROSAR address causes a branch to an interruption microprogram to store the old PSW per D. The 'priority 1' and 'priority 2' triggers (which have already caused all but the highest-priority interrupt request trigger to be reset) causes the interruption code (the 3-bit channel address and the 8-bit unit address) to be gated to S(21–31) as part of the old PSW. Subsequently, the microprogram loads the new PSW per the address in D + 40 (hex). CPU operation then proceeds as dictated by the new PSW.

The I/O interruption microprogram starts by setting the 'timing gate' trigger, thus gating a response back to the

highest-priority channel to reset the interruption request controls. While waiting for the channel to return a 'release' signal, the 'block I-Fetch' trigger is reset and the interruption code is gated to S(16–31). When the microprogram finds the 'release CPU' latch set, signifying that the 'release' signal from channel has been received, the 'timing gate' trigger is reset. STAT H is set so that the common interruption routine, which follows next, skips micro-orders pertaining to the machine check interruption.

Common Interruption Routine

- Program status is assembled in ST and is then stored into old PSW location per interruption cause.
- System is reset if STAT H is reset (machine check interruption).
- Applicable new PSW is fetched per D.
- Processing resumes after new instructions have been fetched and placed into Q.

The common interruption routine (Diagram 5-26, FEMDM) stores the old PSW into main storage and loads a new PSW into the CPU. This routine is entered by all five interruption microprograms.

The IC is reduced by 8 or 16 to reflect the doubleword address of the instruction that caused the interruption. This address is placed into T(40–63) as part of the old PSW. Next, the contents of the PSW register are gated to S(0–15) and T(34–39). E(0,1) is gated to T(32,33) unless the 'instruction length not available' trigger is set, indicating a program interruption is in progress because of a "late" protection check. For that case only, the instruction-length code is set to 0. This action completes the old PSW transfer to ST. The routine inhibits storage protection, sets Marks 0–7, and initiates a 4-cycle storage request to store the old PSW per the D address. An interruption reset clears the CPU of the condition which initiated the access to the interruption microprogram. The 'invalid branch' trigger, the 'invalid instruction address' trigger, and STAT G are reset.

At this point, the common interruption routine checks STAT H to see what class of interruption initiated the operation. If STAT H is reset, the operation is due to a machine check interruption, and the CPU is placed in the scan mode. Three no-op cycles are taken to allow the CPU and main storage to become quiescent. Then a 'system reset' signal clears all control triggers. The 'scan mode' trigger is reset, and the routine prepares to load the new PSW.

STAT H is set if the initiating interruption is other than a machine check. In this case, the CPU is not placed in the scan mode and the system is not reset.

To generate the address for the new PSW, 10 (hex) is placed into B, setting B(59) to 1. Next, the value in B is shifted left twice and gated to PAB as an effective value of 40 (hex). Simultaneously, the old PSW address is gated from D to PAA. The sum, the address of the new PSW, is gated from PAL to D. Storage protection is then inhibited, and a 3-cycle fetch per D is initiated.

The interruption microprogram has now finished the common interruption routine and enters the Load PSW microprogram (Diagram 5-601, FEMDM). When received from main storage, the new PSW is loaded into ST. Because the new PSW will require fetching of instructions from a new storage location, the 'I-Fetch invalid address' trigger is set to enable recording of any invalid address that may result on the subsequent fetch. Portions of the new PSW are loaded into the PSW register and into the IC and D. A 3-cycle storage request per the IC is initiated, and the IC is incremented by 8. At this point, the program shifts to a common branch microprogram. The instruction address in D is incremented by 8 and transferred to the IC in anticipation of a branch instruction. When the first doubleword arrives from main storage it is loaded into Q, and the op-code halfword of the first instruction is transferred to R. D(21,22) is sampled to see if Q needs refilling; if so, a second request per the IC is initiated. If Q does not need refilling, the program generates an end op.

Stop, Wait, and Repeat Exceptional Conditions

The stop exceptional condition is caused by (1)depressing STOP, (2) detecting an address-compare condition when ADDRESS COMPARE STOP is in the stop position, and (3) operating at the instruction-step rate. (If the Multisystem feature is installed, an external-start operation also causes a stop exceptional condition.) The wait exceptional condition is caused by the wait mask bit, PSW(14), being set to a 1. The repeat exceptional condition is caused by activating the REPEAT INSN switch. The scheme for detecting a stop, wait, or repeat instruction exceptional condition is shown in Diagram 5-27, FEMDM.

When any one of these exceptional conditions has priority during end op and an Execute instruction is not in progress, the 'block I-Fetch' trigger is set. During the next cycle, the first I-Fetch cycle, the 'EXCEP' micro-order forces an address into ROSAR: 026, 02A, and 028 (hex) for the stop, wait, and repeat exceptional condition, respectively. This address causes a functional branch to a loop microprogram. Because each of these exceptional conditions may be caused by manual intervention, their microprograms are discussed in Chapter 6, Section 1.

Program Store Compare Exceptional Condition

- Instruction refetch is performed when 'PSC' trigger is set by ASC test or Execute instruction.
- Refetch routine decrements instruction address by 8 or 16 and issues request to refill Q.

An instruction refetch routine is performed if the instructions previously obtained by the CPU must be refetched from main storage. The need to refetch instructions occurs when:

1. The CPU stores data at the main storage address that corresponds to the address currently specified by the IC. In this case, the instructions presently contained in Q may have been modified by the store operation and, therefore, must be refetched.
2. The CPU completes the subject instruction of an Execute instruction and must resume processing the instructions previously contained in Q.

An instruction refetch is initiated by the 'PSC' trigger which is set for either of the above cases. In the first case, the 'PSC' trigger is set as described under "Address Store Compare (ASC) Test." In the second case, the 'PSC' trigger is set by the Execute instruction. The scheme for detecting a program store compare exceptional condition and the instruction refetch microprogram flowchart are shown in Diagram 5-28, FEMDM.

A need to refetch instructions is treated as an exceptional condition by the CPU. When this condition is detected, the 'block I-Fetch' trigger changes the normal I-Fetch routine, and a branch to an instruction refetch microprogram is performed by the first I-Fetch word.

The first ROS word in the refetch microprogram resets the 'block I-Fetch' trigger so that normal I-Fetch can be resumed after Q is refilled. This word also establishes whether the address currently specified in the IC is one or two doublewords ahead of the current instruction. The address in the IC is always at least one doubleword ahead of the address for the instructions in Q. If Q was not refilled before the refetch routine, the IC is one doubleword (8 bytes) ahead of the current instruction; if Q was just refilled, IC is two doublewords or 16 bytes ahead.

IC(21,22) indicates whether Q was refilled before the refetch routine. If IC(21,22) is not set to 11, a request to refill Q (if generated) was blocked by the exceptional condition in progress (i.e., the need for instruction refetch) and the IC is 8 bytes ahead of the current instruction. If, however, IC(21,22) = 11, the need for an instruction refetch occurred after Q was refilled; IC(20) has been incremented, and the IC is 16 bytes ahead. Accordingly, the second ROS word in the refetch microprogram subtracts 8 or 16 from the IC and issues a 3-cycle request per the decremented address. This word also resets the 'PSC' and 'execute in progress' triggers and then causes the Load PSW microprogram to be entered (as shown in Diagram 5-601, FEMDM). Entry corresponds to a point after the new PSW has been loaded but before the successful branch routine. IC is incremented by 8, the next instruction is transferred to R, and Q is refilled, if necessary, before completing the program store compare exceptional condition microprogram with an end op.

Invalid Instruction Address Test Exceptional Condition

- Determines interrupt code triggers to be set if program check was detected while addressing instruction.

When addressing instructions in main storage, three requirements must be met:

1. Because instructions are specified on a 2-byte basis, the least significant bit of the instruction address must always be a zero. Failure to meet this requirement results in a specification program interruption.
2. The instruction address cannot exceed the storage capacity used with a given installation. (The sizes of main storage available for the 2065 CPU are listed in Chapter 1, Section 1.) In addition, the storage unit containing the instruction must be available to the CPU. An attempt by the CPU to execute an instruction from an unavailable or non-existent location results in an addressing program interruption.
3. The instruction address cannot specify an area in main storage that is fetch-protected. An attempt by the CPU to execute instructions from a fetch-protected location results in a protection program interruption.

If any of these three requirements is not met, the CPU hardware forces a new address into ROSAR. The microprogram accessed by this address sets the appropriate program interruption code (specification, addressing, or protection) into the CPU. This microprogram is then followed by the program interruption microprogram previously described. The following paragraphs describe the methods used to detect each violation and to set the appropriate interrupt code triggers.

Specification Detection

All storage requests for instructions do not result in Q being refilled. For example, end-op requests are ignored if the 'block I-Fetch' trigger is set. Also, branch requests made during I-Fetch are ignored if the conditions for a successful branch are not found met during the following execution (non-branch on condition instructions only). For this reason, the least significant address bit of the instruction, IC(23)[or D(23) if preceded by the 'BEOP' micro-order], is detected at the start of I-Fetch. However, because the specification interruption code is not yet set, the program interruption microprogram cannot be immediately entered. Instead, the invalid instruction address test exceptional condition microprogram is entered after processing all interruptions and higher-priority exceptional conditions. If, however, during this forced microprogram, an invalid or fetch-protected address is requested, the specification interruption code is not set because, in either case, the address is outside of "fetchable" storage.

Invalid Address Detection

- 'I-Fetch request' trigger prevents setting of addressing interruption code while refilling Q.
- 'I-Fetch invalid address' trigger indicates IC request is invalid.
- 'Branch invalid address' trigger indicates branch address of successful branch instruction is invalid.

Following a request to refill Q, the IC is incremented by 8 to obtain the instruction address for the next request. The scheme of incrementing the IC ahead of time allows greater speed in requesting instructions from main storage. However, with the IC one doubleword ahead of the instructions in Q, a unique case occurs if the instructions in Q are obtained from the last available location in main storage. In this case, the incremented IC specifies an invalid address; i.e., an address that is in excess of the main storage capacity. Because the Q-register refill routine is initiated before the CPU runs out of instructions, a request per the IC refills Q with instructions from an invalid address.† Even though Q contains invalid instructions, an addressing program interruption must not occur until the CPU attempts to process these instructions. This condition arises because the last valid instruction being processed by the CPU may result in a successful branch to a valid storage location.

A similar situation may occur following an unsuccessful branch instruction that specifies a branch to the last available main storage location. Excluding the Branch on Condition instructions, the CPU assumes that the branch instruction is successful and, accordingly, issues a request per D. Following the request, D is updated by 8 and specifies an invalid address. In this case, an addressing program interruption must not occur because, upon establishing that the branch is unsuccessful, the CPU resumes normal addressing per the IC. For branch instructions, an addressing interruption must occur only when the address specified by a successful branch is above the available main storage capacity. This situation may also exist after any load-PSW operation or after the program store compare exceptional condition.

An invalid-address test is performed each time the CPU issues a request to refill Q. Because a request for invalid instructions will not necessarily cause an interruption, setting of the interrupt code triggers must be blocked while Q is being refilled.† The scheme used for detecting a "true" invalid instruction address error is shown in Figure 3-4.

The 'I-Fetch request' trigger prevents the invalid-address condition from causing an interruption while Q is being refilled. This trigger is set when a need to refill Q is detected; depending on the current instruction status in the CPU, the trigger is set as follows:

1. For non-branch 1- and 2-halfword instructions, the trigger is set by the I-Fetch sequencers.
2. For branch instructions, the trigger is set by the '1→INST-MSREQ' micro-order given at the start of the branch execution. Note that if an unsuccessful Branch on Condition instruction occurs and IC(21,22) = 00, the 'I-Fetch request' trigger is not set because Q will not be refilled.
3. For SS instructions, the trigger is set if IC(21), or D(21) for the 'BEOP' micro-order, is equal to 0 during end op. This condition indicates that the complete SS instruction is already in Q and succeeding instructions are being requested.

The output of the 'I-Fetch request' trigger prevents the interrupt code triggers from being set by the 'invalid address' signal from the BCU. In addition, the output of the 'I-Fetch request' trigger serves as one of the conditioning inputs for the 'I-Fetch invalid address' and 'branch invalid address' triggers. One of these triggers is set whenever the BCU indicates that the address of the storage request exceeds the main storage capacity. The 'invalid address' signal sets the 'I-Fetch invalid address' trigger if the invalid address is due to the CPU fetching ahead. Conversely, this signal sets the 'branch invalid address' trigger when the invalid address is the result of a successful branch instruction.

The 'gate I-Fetch invalid address' trigger dictates whether the 'I-Fetch invalid address' or 'branch invalid address' trigger is to be set. When set, this trigger conditions the 'I-Fetch invalid address' trigger; when reset, the 'branch invalid address' trigger. Depending on the current instruction status in the CPU, the 'gate I-Fetch invalid address' trigger is set as follows:

1. For non-branch 1- and 2-halfword instructions, the trigger is set by the I-Fetch sequencers.
2. For SS instructions, the trigger is set by the 'IF-INV→TGR' micro-order given at the start of the SS I-Fetch routine; i.e., the presence of the 'IF-INV→TGR' micro-order and the absence of the

†The instruction address is considered invalid by the BCU upon detection of a carry from the most significant bit position in the IC. This bit position is defined by the size of the main storage in the particular installation.

†An exception to this rule occurs if the last one or two halfwords of an SS instruction are requested from an invalid address while the first halfword is contained in a valid storage location. In this case, the entire SS instruction is considered to have an invalid address and, because the CPU has attempted to process the instruction, the interrupt code triggers are set as soon as the request is generated.

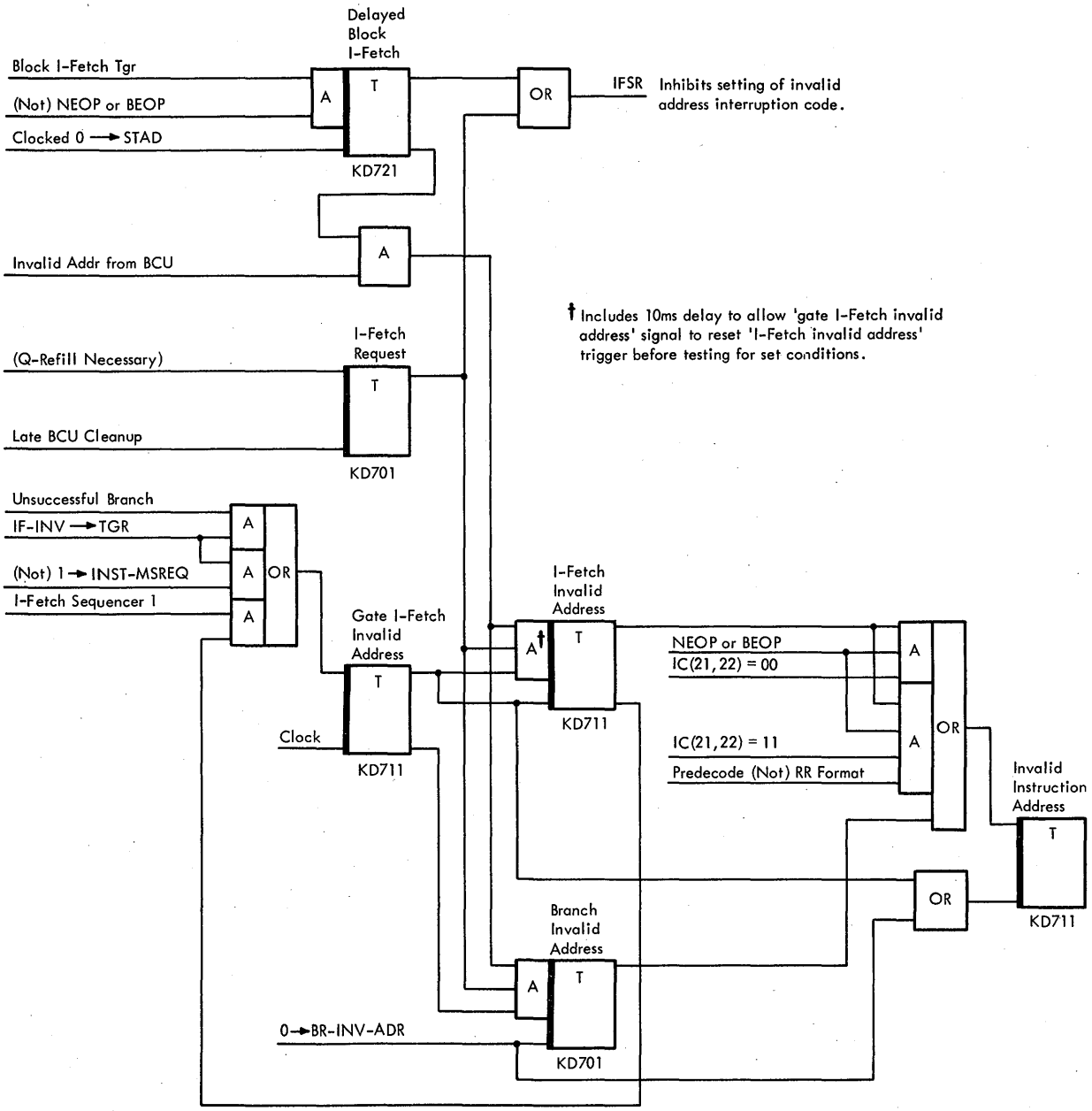


Figure 3-4. Detection of Invalid Instruction Address

'1→INST-MRSEQ' micro-order when the complete SS instruction is in Q.

3. For unsuccessful branch instructions (except Branch on Condition), the trigger is set by the presence of the 'IF-INV→TGR' micro-order and the absence of the '1→INST-MSREQ' micro-order; for an unsuccessful Branch on Condition instruction, the trigger is set by the simultaneous presence of the '1→INST-MSREQ' and 'IF-INV→TGR' micro-orders at the start of execution. (The various branching conditions that may arise are described in Section 6 of this Chapter.)

When the 'gate I-Fetch invalid address' trigger is set, the 'invalid address' signal is allowed to set the 'I-Fetch invalid address' trigger, indicating that Q has been refilled with instructions from an invalid address. However, because R may still contain a valid RR instruction, further testing is required to establish that a true interruption condition exists. The setting of IC(21,22) during end op indicates whether a valid or an invalid instruction is contained in R. If IC(21,22) = 11 and an RR instruction is predecoded, R contains a valid instruction. When IC(21,22) = 11 but the instruction is not of the RR

format, the balance of the instruction has been obtained from an invalid location and the 'invalid instruction address' trigger is set. If IC(21,22) = 00, the 'invalid instruction address' trigger is set regardless of the instruction format; this condition indicates that R contains the first halfword of an invalid instruction.

When the 'gate I-Fetch invalid address' trigger is *not* set, the 'invalid address' signal sets the 'branch invalid address' trigger. Because, in this case, the invalid address is the result of a successful branch instruction, the 'invalid instruction address' trigger is set without further testing being necessary.

Fetch Protection Detection

- 'Delayed I-Fetch storage request' trigger prevents setting of protection interruption code while Q is being refilled.
- 'Delayed I-Fetch protect gate' trigger is set if request is due to normal sequencing.
- 'Protected branch address' trigger is set if branch is made to protected location.

The CPU cannot execute instructions from a fetch-protected area in main storage. Because the IC is always one doubleword ahead of the instructions in Q, a unique situation occurs if the instructions in Q are obtained from a main storage location adjacent to a protected area. In this case, the incremented IC specifies a protected address and, because the Q-register refill routine is initiated before the CPU runs out of instructions, the request per the IC refills Q with instructions from a protected address. A protection interruption, however, does not occur until the CPU attempts to execute the protected instructions. This condition arises because the last valid instruction being processed by the CPU may result in a successful branch to a valid storage location.

A protection test is performed each time the CPU issues a request to refill Q. Because a request for protected instructions will not necessarily cause an interruption, the setting of the 'protection check (to CPU)' and 'instruction length not available' triggers is blocked while Q is being refilled. The scheme for detecting a "true" protection violation, shown in simplified form in Figure 3-5, is closely related to the invalid addressing detection scheme previously described. The major difference between the invalid-addressing and fetch-protection schemes is in timing: the 'invalid address' signal arrives at the CPU 1 cycle after the request, while the 'protection check' signal arrives 2 cycles after the request. For this reason a separate circuit is used for detecting a protection violation.

The 'delayed I-Fetch storage request' trigger prevents the 'protection check' signal from causing an interruption while Q is being refilled. This trigger is set one cycle after the 'I-Fetch request' trigger is set by a request to refill Q. The output of the 'delayed I-Fetch storage request' trigger serves as one of the conditioning inputs for the 'delayed I-Fetch protect gate' trigger. This trigger is set if the current storage request is due to a nonbranch or unsuccessful branch request and is set by the same conditions that set the 'gate I-Fetch invalid address' trigger. When the 'delayed I-Fetch storage request' trigger and the 'delayed I-Fetch protect gate' trigger are both set, the 'protection check' signal sets the 'I-Fetch invalid address' trigger. The action at this point is identical to that described for detection of invalid addressing; i.e., a test is made to establish if the CPU has attempted to execute instructions from a protected area, and the 'invalid instruction address' trigger is set if a protect violation has occurred.

If the request to refill Q is a result of a branch instruction (as indicated by the reset state of the 'gate I-Fetch invalid address' trigger), the 'delayed I-Fetch protect gate' trigger is not set. In this case, the 'protection check' signal sets the 'protected branch address' trigger, which in turn sets the 'invalid instruction address' trigger.

Invalid Instruction Address Microprogram

- Issues second request for instruction.
- Interrupt code triggers are set per highest-priority error detected: (1) addressing = 101, (2) protection = 100, (3) specification = 110.

The detection of a specification, addressing, or protection exception and the associated microprogram are shown in Diagram 5-29, FEMDM. Because all three exceptions access the same microprogram, the microprogram must re-establish the nature of the exception to set the proper interruption code. To establish which exception is currently in effect, the microprogram issues a second request for instructions. This time, however, the setting of the interrupt code triggers is not blocked; i.e., the appropriate interruption code is set immediately upon detection of a specification, addressing, or protection exception.

Before generating a second request, the IC must be decremented to the address that caused the exception. The status of IC(21,22) indicates whether the current IC count is one or two doublewords ahead of the required address: if IC(21,22) is not set to 00, the IC is one doubleword ahead; if IC(21,22) = 00, the IC is two doublewords ahead. (The other recovery microprograms test IC(21,22) for a setting of 11; because the invalid instruction address test exceptional condition does not set the 'block I-Fetch' trigger, IC(21,22) is updated and tested for a setting of 00.)

Accordingly, the microprogram subtracts 8 or 16, decimal, from the IC, and loads the decremented address into D. The STC is then set to zero and a 3-cycle request is issued per D. After the request, the STC is incremented

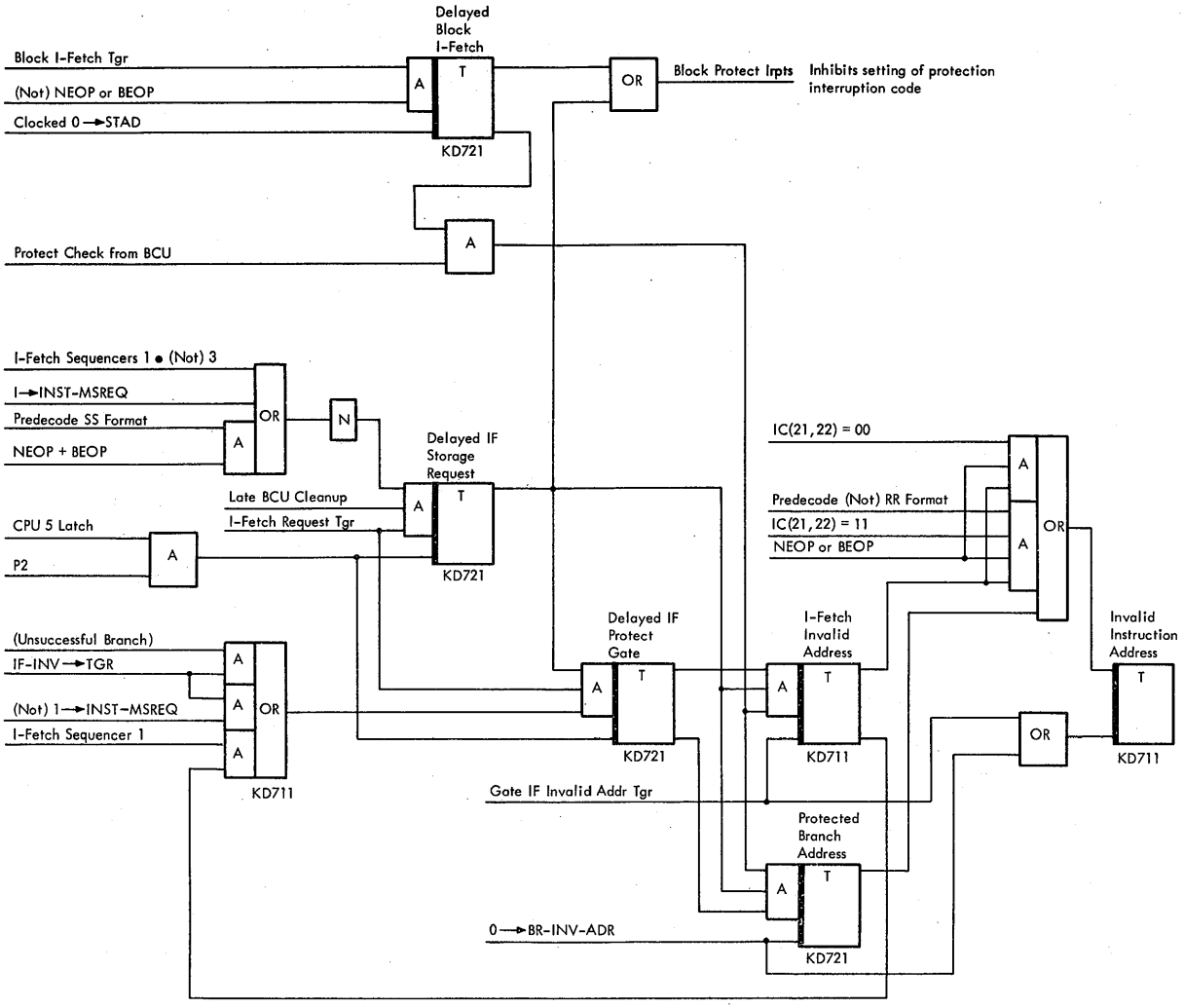


Figure 3-5. Detection of Fetch-Protected Instruction Address

once during each subsequent cycle to provide the required wait interval between the request and what would be the cycle for transferring the instructions to Q. The appropriate interrupt code trigger(s) is then set upon receipt of a specification, addressing, or protection exception. Because the low-order bits of the addressing interruption code equal 101, there is no need to block the protection interruption code (100) from also being set if it is received. However, if either of these conditions is set, the setting of the specification interruption code, 110, is blocked. The specification interruption code need only be set if IC(23) = 1; because the invalid instruction address test exceptional condition does not set the 'block I-Fetch' trigger, IC(23) = 1 if D(23) equalled 1 during the preceding branch end op.

The invalid instruction address test exceptional condition microprogram terminates with a normal end op. If no higher-priority exceptional condition or interruption is detected, the program interruption microprogram is entered next.

Q-Register Refill Exceptional Condition

- One extra I-Fetch cycle is performed to allow refilling of Q without conflicting with execution sequence of next instruction.
- I-Fetch sequencers are always activated.

Following each storage request for instructions, the IC is incremented by 8 to obtain the address from which instructions will be fetched by the next request. This updating is accomplished by gating the contents of the IC to the parallel adder, adding a 1 to IC(20), and gating the incremented address back to the IC. The need to update the IC usually adds another cycle to the I-Fetch of RR, RX, RS, and SI instructions. There are two reasons for this extra cycle:

1. The BCU requires that each main storage address be retained for at least two cycles. Therefore, main storage requests during the first I-Fetch cycle would interfere with end-op requests.

2. Because the parallel adder is used to increment the IC, and the first cycle in the execution phase may also require the use of the parallel adder, the execution phase must be delayed until the IC is incremented.

The case when I-Fetch requires a second cycle (third if an indexed RX instruction) is treated as an exceptional condition in the CPU. The 'EXCEP' micro-order in the first ROS word of the I-Fetch microprogram overrides the functional branch micro-order per the instruction op code [E(02–07)→ROA] and forces a new address into ROSAR. This forced address is determined by the format of the upcoming instruction and the status of the I-Fetch sequencers (Diagram 5-30, FEMDM). Operation of the I-Fetch sequencers is initiated at end op when the need to refill Q exists and the next instruction to be executed is not in the SS format. If the request was generated two cycles before end op, sequencer 2 is latched at the start of I-Fetch; otherwise, sequencer 1 is being set.

At the start of I-Fetch, The 'EXCEP' micro-order samples sequencer 1 to see if it is being set. If it is, a new address is always forced into ROSAR, causing one extra I-Fetch word to be added to the basic I-Fetch.

If sequencer 2 is found latched, the parallel adder is available for use on the next cycle because IC(20) has already been incremented. I-Fetch of RR and shift instructions does not require the fetching of an operand from main storage; also, storage operands for indexed RX instructions are not requested until the second basic I-Fetch cycle. For these reasons, sequencer 2 forces a new ROSAR address only if an RR, indexed RX, or shift instruction is *not* being fetched.

The forced ROSAR addresses as a result of the Q-register refill exceptional condition are shown in Table 3-1. The SS format is included for completeness. However, because the I-Fetch sequencers are not activated, a Q-register refill exceptional condition is never detected during the SS I-Fetch microprogram. Instead, the functions of the sequencers are initiated by micro-orders in that microprogram.

Two-Cycle RR I-Fetch

The actions of the first I-Fetch cycle are unchanged except for the overriding of the 'E(02–07)→ROA' micro-order by the 'EXCEP' micro-order (Diagram 5-6). During

Table 3-1. Q-Register Refill Exceptional Conditions

Instruction Being Fetched	IC(21,22) at End Op	Request Issued During Preceding: EEOP (2 Cycles Early)	Request Issued During Preceding: NEOP, BEOP, or EEOP (1 Cycle Early)	Forced ROSAR Address (Hex)
RR	00, 01, or 11	Never	Never	None
	10	Yes	Never	None
	10	No	Yes	030
RX, RS, or SI	00 or 11	Never	Never	None
Indexed RX, or shift RS†	01 or 10	Yes	Never	None
Indexed RX	01 or 10	No	Yes	03A
Non-indexed RX	01 or 10	Yes	Never	022
Non-indexed RX	01 or 10	No	Yes	032
Shift RS†	01 or 10	No	Yes	020
Non-shift RS, or SI	01 or 10	Yes	Never	024
Non-shift RS, or SI	01 or 10	No	Yes	034
SS	11	Never	Never	None
	00, 01, or 10	Never	Yes	None

†All shift instructions are of the RS format with an op code of 1000 1XXX.

the second RR I-Fetch cycle, sequencer 2 is set and sequencer 1 is reset. This status increases the IC by 8 and returns the new instruction address to the IC at the start of the next cycle, thus completing the updating of IC(20). During the next cycle (first execution cycle), sequencers 3 and 1 are both set by the 'RASCR' micro-order in the forced word. This condition indicates that new instructions are to be gated to Q at the start of the second execution cycle. The major registers and timing applicable to this sequence are shown in Diagram 5-8, FEMDM.

Forced-Cycle RX I-Fetch

If the request to refill Q was not issued 2 cycles before end op, the actions of the second RX I-Fetch cycle include the same actions as the second RR I-Fetch cycle. Otherwise, IC(20) has already been incremented and sequencers 1 and 3 are automatically set during the second cycle. If the RX instruction is indexed, then a second forced word is now performed; this word contains the same micro-orders as the second word of the basic RX I-Fetch. Otherwise, the first forced word completes the I-Fetch routine after issuing any request inhibited during the first I-Fetch cycle. This action is performed by the 'MS-REQ*D-3' micro-order. D is also transferred to PAL if the request for new instructions was issued 2 cycles before end op. Because in this case sequencer 2 is reset during the first execution cycle, a 'SPEC' micro-order in the first execution word tests the storage address from PAL, not D. The new instructions are gated into Q [and the next op-code word to R from Q(0–15) if IC(21,22) = 10 at end op] at the start of the first execution cycle (second execution cycle if the request was not made two cycles early and the RX instruction is not indexed). The major registers and timing applicable to the non-indexed case are shown in Diagram 5-12, FEMDM.

Two-Cycle RS and SI I-Fetch

The major registers and timing applicable to the 2-cycle RS and SI I-Fetch are the same as that for the 2-cycle, non-indexed RX I-Fetch (Diagram 5-12). If the request to refill Q was not issued two cycles before end op, the actions of the second I-Fetch cycle include the same actions as the second RR I-Fetch cycle. Otherwise, IC(20) has already been incremented and sequencers 1 and 3 are automatically set during this forced cycle. This action results in the refill of Q at the start of the next cycle (first execution cycle). Whichever cycle Q is refilled, Q(0–15) is also transferred to R if IC(21,22) was set to 10 during the preceding end op. This transferring of the op-code halfword is otherwise performed by an appropriate micro-order in the first I-Fetch word. If an MVI, STM, TS, I/O, or shift instruction is being fetched, no further I-Fetch actions are necessary. However, fetching of other RS or SI instructions causes the storage request omitted by the first I-Fetch cycle to be issued now. This request is performed by the 'MS-REQ*D-3' micro-order. Also, D is transferred to PAL if the Q-register refill request was issued two cycles before end op. Because in this case sequencer 2 is reset during the first execution cycle, any 'SPEC' micro-order tests the storage address from PAL, not D.

It was previously stated that the last I-Fetch word always includes the 'E(02–07)→ROA' micro-order. However, there is one exception to this statement: the forced I-Fetch cycle for shift instructions includes the 'E(04–07)→ROA' micro-order, which, in turn, forces the first execution cycle to branch to the second cycle per D rather than per PAL.

www.IBM360.info

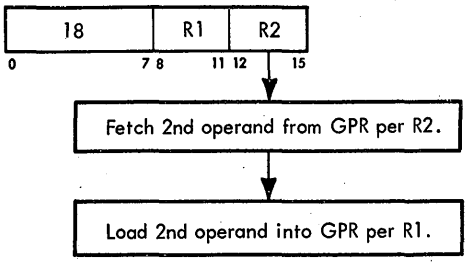
This section discusses the 35 instructions of the fixed-point instruction set. These instructions use the RR, RX, and RS formats. Positive fixed-point numbers are expressed in true binary form, whereas negative numbers are expressed in complement binary form (2's complement form). One operand is always in 1 of the 16 GPR's; the other operand may be in either a GPR or in main storage. For a discussion of number representation, data formats, operand addressing, instruction formats, data flow, program interruptions, and condition codes, see Chapter 1.

LOAD

The fixed-point load instructions provide a means of loading operands into the LS GPR's. The load operation may be register-to-register (RR format) or storage-to-register (RX and RS formats). In any case, the instruction loads the second operand into the first operand location, and the second operand location remains unchanged. In addition, certain load instructions can test the second operand before loading it and can load the second operand in complement, positive, or negative form.

Load, LR (18)

- Load 2nd operand (in GPR per R2) into 1st operand location (in GPR per R1).
- RR format:

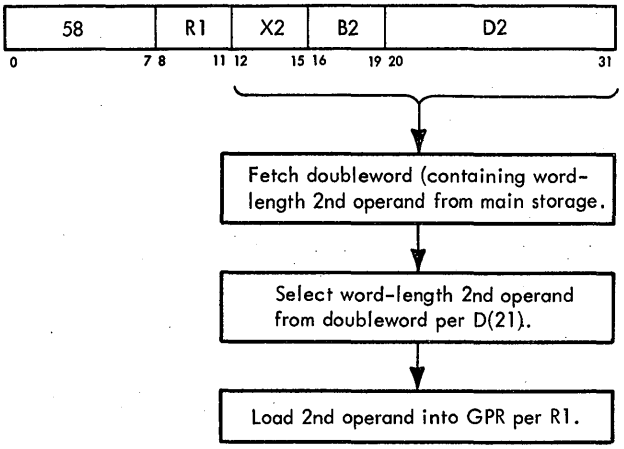


- Conditions at start of execution:
  Instruction is in E.
  1st operand is in A, B, and D (not used).
  2nd operand is in S and T.

The Load, LR, instruction loads the second operand from the GPR per R2 into the GPR per R1. At the start of execution, the word-length second operand is in S and T. Because both operands are in GPR's, no specification test is performed. The second operand is loaded into the GPR specified by R1, and an end-op cycle is taken.

Load, L (58)

- Load 2nd operand (in storage) into 1st operand location (in GPR per R1).
- RX format:

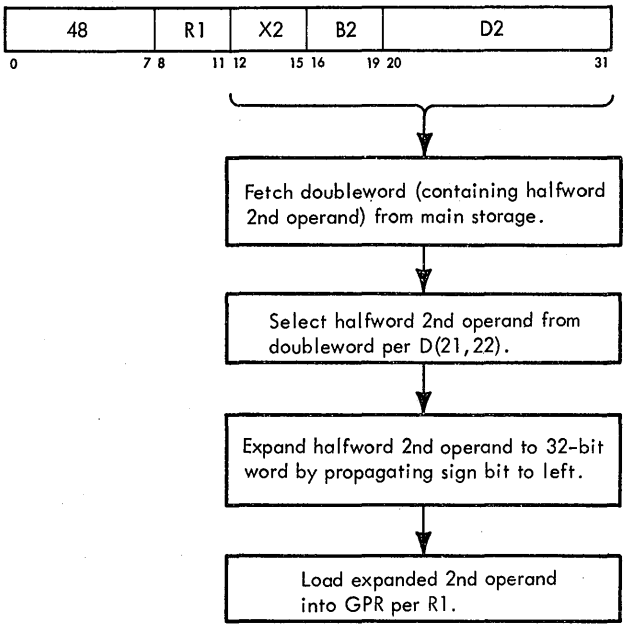


- Conditions at start of execution:
  First 16 bits of instruction are in E.
  1st operand is in S and T (not used).
  2nd operand address is in D.
  Main storage request for 2nd operand has been issued per D.
- D(21) determines which word of doubleword is to be stored: if a 1, right word; if a 0, left word.

The Load, L, instruction loads the second operand from main storage into the GPR per R1. Instruction execution starts with a specification test. If a program specification interruption occurs, an end op is forced and the instruction is suppressed. If no specification check exists, D(21) is tested to determine which word of the doubleword fetched from main storage will be gated from the SDBO to T. If D(21) = 1, the right word is gated; if D(21) = 0, the left word is gated. The contents of T are then loaded into the GPR specified by R1, and an end-op cycle is taken.

Load Halfword, LH (48)

- Load halfword 2nd operand (in storage) into 1st operand location (in GPR per R1).
- RX format:



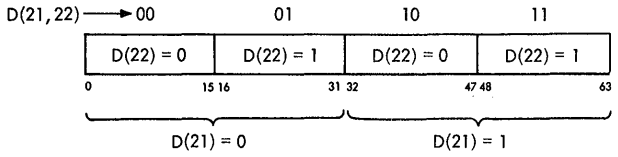
- Conditions at start of execution:
  First 16 bits of instruction are in E.
  1st operand is in S and T (not used).
  2nd operand address is in D.
  Main storage request for 2nd operand has been issued per D.
- Halfword operand is expanded to word by propagating sign of halfword into 16 high-order bits of word-length register.
- D(21) determines whicn word of doubleword contains halfword 2nd operand: if a 1, right word; if a 0, left word.
- D(22) determines which half of word contains halfword 2nd operand: if a 1, right half; if a 0, left half.

The Load Halfword, LH, instruction loads the halfword second operand (located in main storage) into the GPR specified by R1. The halfword obtained from main storage consists of 16 bits. Before loading the operand into LS, it is expanded to a 32-bit word by propagating the sign bit of the halfword through the 16 high-order bits of a word-length register.

Diagram 5-102, FEMDM, is a flowchart of the Load Halfword instruction. At the start of execution, the first 16 bits of the instruction are in E, the second operand address is in D, and the first operand is in S and T. (The first operand plays no part in the instruction; it is subsequently destroyed when data is loaded into T.) During I-Fetch, a storage request was made to obtain the doubleword containing the halfword second operand.

The instruction first tests for a specification check condition. If a program specification interruption occurs, an end op is forced and the instruction is suppressed. If no specification check exists, the execution continues and may be divided into three general steps:

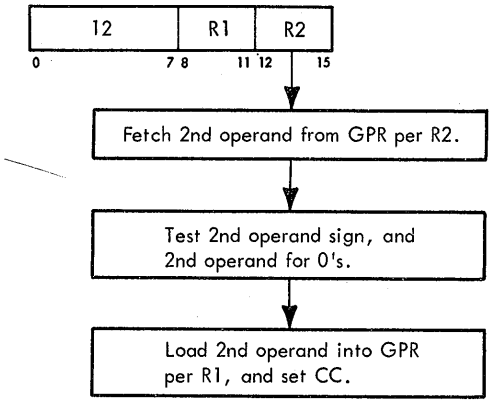
1. Set up A to propagate the sign:
   a. Load 1's into PAL(32–59), shift left four positions, and gate the result to A. ("A" now contains FFFF FF00.) The 1's are generated (ALD AP811) as a result of ROS word 0108 (CLD QB011) containing 100 in bit positions 82, 83, and 84 (CLD QZ011).
   b. Gate the contents of A to PAB, shift left four positions, and gate the result to A. ("A" now contains FFFF F000.)
   c. Gate the contents of A to PAB, shift left four positions, and gate the result to A. ("A" now contains FFFF 0000.)
2. Examine D(21) and D(22) to determine which halfword of the doubleword operand brought out from main storage is to be used as the halfword operand. The specified halfword may be in any one of four possible positions of the doubleword specified by D(21) and D(22):



   a. If D(21) = 0, gate SDBO(0–31) to T; if D(21) = 1, gate SDBO(32–63) to T.
   b. If D(22) = 0, gate T(32–47) to PAA(48–63), and gate a 1 to PAA(47) if T(32) = 0. If D(22) = 1, gate T(48–63) to PAA(48–63), and gate a 1 to PAA(47) if T(48) = 0.
   c. Gate the contents of A to PAB(32–63).
3. Load the selected word:
   a. Gate PAL(32–63) to T.
   b. Gate the contents of T to the GPR specified by the R1 field in E(8–11).
   c. Take an end-op cycle.

Load and Test, LTR (12)

- Load 2nd operand (in GPR per R2) into 1st operand location (in GPR per R1) and set CC according to result.
- RR format:



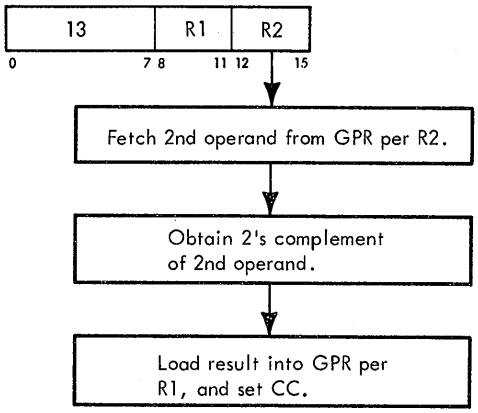
- Conditions at start of execution:
  Instruction is in E.
  1st operand is in A, B, and D (not used).
  2nd operand is in S and T.
- Set STAT A if PAL is all 0's.
- Test T(32) for plus or minus sign.
- STAT A and T(32) determine CC.
- CC setting:
  Result in PAL is zero: CC = 0.
  Result in PAL is less than zero: CC = 1.
  Result in PAL is greater than zero: CC = 2.

The Load and Test, LTR, instruction tests the second operand, from the GPR per R2, for all zeros and loads it into the GPR per R1. (If R1 and R2 specify the same GPR, the operation is equivalent to a test of the data without movement of the data.)

Diagram 5-103, FEMDM, is a flowchart of the instruction. The contents of T (second operand) are gated to PAA(32–63), and STAT A is set if PAL equals zero. The contents of T are then gated to the GPR specified by R1, and the CC is set as follows. If STAT A is set, the CC is set to 0. If STAT A is not set, the sign bit [T(32)] determines the CC: if the sign is minus [T(32) = 1], the CC is set to 1; if the sign is plus, the CC is set to 2. An end op completes instruction execution.

Load Complement, LCR (13)

- Load 2's complement of 2nd operand (in GPR per R2) into 1st operand location (in GPR per R1) and set CC according to result.
- RR format:



- Conditions at start of execution:
  Instruction is in E.
  1st operand is in A, B, and D (not used).
  2nd operand is in S and T.
- Set STAT B if overflow occurred.
- Set STAT A if PAL is all 0's.
- Test T(32) for plus or minus sign.
- STAT's A and B, and T(32) determine CC.
- CC setting:
  Result in PAL is zero: CC = 0.
  Result in PAL is less than zero: CC = 1.
  Result in PAL is greater than zero: CC = 2.
  Overflow: CC = 3.

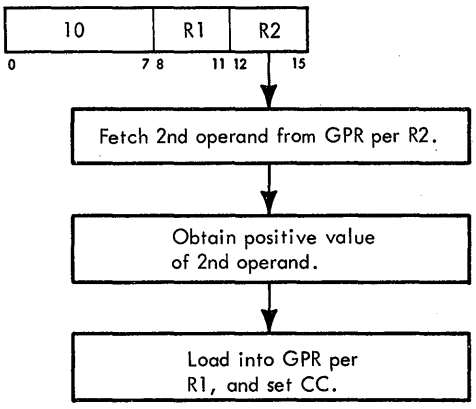
The Load Complement, LCR, instruction loads the 2's complement of the second operand from the GPR per R2 into the GPR per R1.

See Diagram 5-104, FEMDM, a flowchart of the instruction. The contents of T (second operand) are gated in 2's complement form to PAA(32–63). STAT A is set if PAL equals zero, and STAT B is set if a fixed-point overflow occurs. (Overflow occurs if the maximum negative number is 2's complemented.) The contents of PAL are transferred (via T) to the GPR specified by R1, and the CC is set as follows.

If STAT B is set, the CC is set to 3. If STAT A is set, the CC is set to 0. If neither STAT is set, the sign bit [T(32)] determines the CC: if the sign is minus [T(32) = 1], the CC is set to 1; if the sign is plus, the CC is set to 2. An end op completes instruction execution.

Load Positive, LPR (10)

- Load 2nd operand (unchanged if positive, 2's complemented if negative; in GPR per R2) into 1st operand location (in GPR per R1).
- RR format:



- Conditions at start of execution:
  Instruction is in E.
  1st operand is in A, B, and D (not used).
  2nd operand is in S and T.
- Set STAT A if PAL is all 0's.
- T(32) determines whether operand loaded is positive.
- If T(32) = 1, 2's complement operand.
- Set STAT B if overflow occurs.
- STAT's A and B and T(32) determine CC.
- CC setting:
  Result in PAL is zero: CC = 0.
  Result in PAL is greater than zero: CC = 2.
  Overflow: CC = 3.

The Load Positive, LPR, instruction loads the absolute value of the contents of the GPR specified by R2 into the GPR specified by R1. The instruction also tests for an all-zero result and for an overflow condition. (Overflow occurs only when the maximum negative number is 2's complemented.) The results of the tests are indicated by the CC.

Diagram 5-105, FEMDM, is a flowchart of the Load Positive instruction. At the start of execution, the instruction is in E, the first operand is in A, B, and D, and the second operand is in S and T. The first cycle of the instruction places the contents of T into PAL(32–63) and tests PAL for all 0's. If PAL equals zero, STAT A is set. The data in T is then loaded into the GPR per E(8–11) (R1).

Because the purpose of the instruction is to load only positive numbers, a test for negative numbers is made by examining T(32). If T(32) = 1, the data loaded in LS was a negative number. In this case, the contents of T must be converted to a positive number (true form) and reloaded into the GPR per E(8–11), thus destroying the negative number in that location.

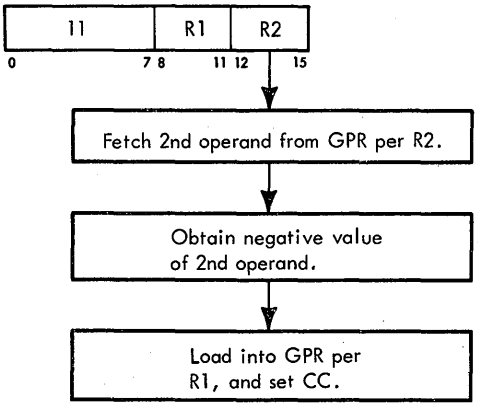
While in PAL, the 2's complement form of the data is tested to see whether overflow occurred when the number was converted. If overflow occurred, STAT B is set.

If T(32) = 0, the data loaded in LS was positive and need not be changed.

STAT's A and B, and T(32) determine the CC as follows. If STAT B is set, the CC is set to 3. If STAT A is set, the CC is set to 0. If neither STAT is set, the sign bit [T(32)] determines the CC; however, the sign can only be plus [T(32) = 0] and the CC is set to 2. An end-op cycle completes instruction execution.

Load Negative, LNR (11)

- Load 2nd operand (unchanged if negative, 2's complemented if positive; in GPR per R2) into 1st operand location (in GPR per R1).
- RR format:



- Conditions at start of execution:
  Instruction is in E.
  1st operand is in A, B, and D (not used).
  2nd operand is in S and T.
- Set STAT A if PAL is all 0's.
- T(32) determines whether operand loaded is negative.
- If T(32) = 0, 2's complement operand.
- STAT A and T(32) determine CC.
- CC setting:
  Result in PAL is zero: CC = 0.
  Result in PAL is less than zero: CC = 1.

The Load Negative, LNR, instruction loads the negative value of the contents of the GPR specified by R2 into the GPR specified by R1. The LNR instruction also tests the operand for all zeros and indicates the result in the CC.

See Diagram 5-106, FEMDM, a flowchart of the instruction. At the start of execution, the instruction is in E, the first operand is in A, B, and D, and the second operand is in S and T. The first cycle of the instruction

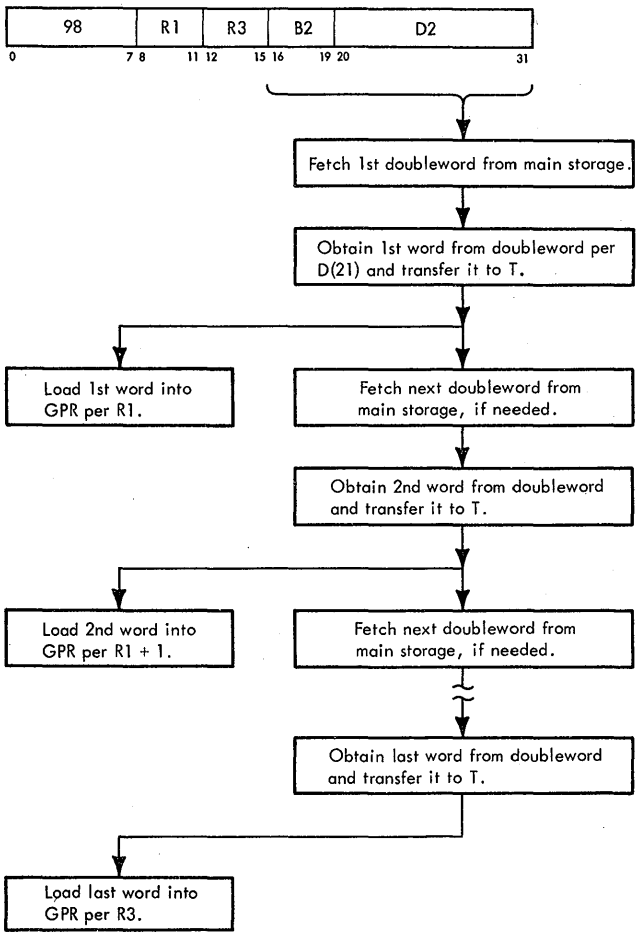
places the contents of T into PAA(32–63) and tests PAL for all 0's. If PAL contains all 0's, STAT A is set. The data in T, regardless of whether the result equals zero, is loaded into the GPR per E(8–11) (R1).

Because the purpose of the LNR instruction is to load only negative numbers, a test for positive numbers is made by examining T(32). If T(32) = 0, the data loaded into LS was a positive number. In this case, the contents of T must be converted to a negative number (2's complement form) and reloaded into the GPR per E(8–11), thus destroying the positive number in that location. If T(32) = 1, the data loaded into LS was a negative number and need not be changed.

STAT A and T(32) determine the CC as follows. If STAT A is set, the CC is set to 0. If STAT A is not set, the sign bit [T(32)] determines the CC; however, the sign can only be minus [T(32) = 1] and the CC is set to 1. An end-op cycle completes instruction execution.

Load Multiple, LM (98)

- Load 2nd operand (as many words as required; in storage) into GPR's, in ascending order, starting with 1st operand location (per R1) and ending with 3rd operand location (per R3).
- RS format:



- Conditions at start of execution:
  First 16 bits of instruction are in E.
  1st operand is in S and T (not used).
  Starting operand address is in D.
  Main storage request for 2nd operand has been issued per D.
- R1 and R3 are compared to determine whether one or more words are to be loaded.
- If R1 = R3, only one word is to be loaded.
- If R3 is less than R1, addresses wraparound from GPR 15 to GPR 0.
- D(21) determines which word of doubleword is to be loaded into LS: if a 1, right word; if a 0, left word.
- 8 is added to D when more than one word is to be loaded.
- R1 is incremented by 1 each time a new word is to be loaded.

The Load Multiple, LM, instruction loads 32-bit words from main storage into LS. The GPR's are loaded in the ascending order of their addresses, starting with the GPR addressed by R1 and continuing up to and including the GPR addressed by R3. All combinations of GPR addresses specified by R1 and R3 are valid. When R3 is less than R1, the addresses wrap around from GPR 15 to GPR 0.

Diagram 5-107, FEMDM is a flowchart of the Load Multiple instruction. At the start of execution, the first 16 bits of the instruction are in E, the first operand is in S and T, and the starting operand address is in D. During I-Fetch, a storage request was made to fetch the operand addressed by D.

The instruction first tests for a specification check condition. If there is a specification check, a program specification interruption occurs and the operation is suppressed. Assuming there is no specification check, R1 [E(8–11)] and R3 [E(12–15)] are compared to determine whether one or more words are to be loaded. If R1 equals R3, only one word is to be loaded; if unequal, more than one word is to be loaded.

Assume one word is to be loaded (R1 = R3). D(21) is tested to determine which word of the doubleword from main storage is to be loaded. If D(21) is a 0, the left word is selected; if a 1, the right word is selected. The selected word is loaded into the GPR specified by R1, and an end-op cycle is taken, completing the instruction.

If more than one word is to be loaded (R1 and R3 are unequal), the main storage address in D is incremented by 8 to address the next doubleword in main storage, if it should be needed. D(21) is tested to determine which word of the doubleword from SDBO will be loaded first. R3 is decremented by 1, and R1 and R3 are again compared. As a result of these two tests, four possible conditions may exist, resulting in four different branches as follows:

1. If D(21) = 0 and R1 = R3, two words are to be loaded starting with the left word of the doubleword on the

SDBO. In this case, no further main storage request is necessary. SDBO(0–31) is gated to T, SDBO(0–63) is gated to AB, and the contents of T are loaded into the GPR specified by R1. E(8–11) (R1) is incremented by 1 to address the next sequential GPR, and the contents of B are transferred (via T) to the GPR specified by E(8–11) (R1 + 1).

2. If D(21) = 0 and R1 does not equal R3, more than two words are to be loaded starting with the left word. A main storage request per D is initiated to fetch the next doubleword from main storage. The microprogram enters the basic loop, which loads a doubleword, one word at a time, in consecutive GPR's and initiates another main storage request. An exit from the basic loop is made when one or two words remain to be loaded.
3. If D(21) = 1 and R1 = R3, two words are to be loaded starting with the right word. A main storage request per D is initiated because the second word to be loaded is contained in the next doubleword in main storage. SDBO(32–63) is transferred (via T) to the GPR specified by E(8–11) (R1). E(8–11) is incremented by 1, and, when the next doubleword is available, SDBO(0–31) is transferred (via T) to the GPR specified by E(8–11) (R1 + 1).
4. If D(21) = 1 and R1 does not equal R3, more than two words are to be loaded starting with the right word. A main storage request per D is initiated to fetch the next doubleword from main storage, and SDBO(32–63) is transferred (via T) to the GPR specified by E(8–11) (R1). E(8–11) is incremented by 1 to address the next sequential GPR, and D is incremented by 4 to address the next sequential doubleword in main storage. R1 and R3 are again compared; if equal, only two more words are to be loaded. When the requested doubleword is available, it is loaded, one word at a time, into the two sequential GPR's specified. If R1 does not equal R3, the microprogram enters the basic loop.

   Note that the last time D was incremented, it was incremented by 4 rather than by 8. At the start of this sequence, D(21) equalled 1, which is equivalent to a value of 4 in D. Adding 4 to D increases the value to 8, which will address the next sequential doubleword in main storage. D(21) has also been changed to a 0, which allows the microprogram to remain in the basic loop as long as is required. Note that a branch on D(21) is performed in the basic loop.

   The 4 that is added to D is developed in F. At the start of the instruction, F was set to -64 (1100 0000). F(0) is then set to 0, establishing a value of 0100 0000 in F. F(0–3) and F(4–7) are transposed, giving a value of 0000 0100 (4) in F. When F(4–7) is added to D, D is incremented by 4.

   When the last word has been loaded into the GPR specified by R3, an end-op cycle is taken, completing the instruction.

ADD-TYPE INSTRUCTIONS

- Fixed-point add-type instructions use RR and RX formats.
- 2nd operand is algebraically added to 1st operand.
- For subtract and compare instructions, 2nd operand is in 2's complement form.
- Except for compare instructions, result is stored into 1st operand location.
- CC is determined by op code and hardware conditions.

The fixed-point add-type instructions use the RR format with word-length operands, the RX format with word-length operands, and the RX format with a halfword second operand.

At the start of execution of RR format fixed-point instructions, the first operand is in B (also in A and D) and the second operand is in T (also in S).

At the start of execution of RX format fixed-point instructions, the first operand is in S and T, and a main storage request for the second operand has been issued per D. Because the second operand is fetched from main storage, a specification test is performed (Diagram 5-108, Sheet 1, FEMDM), and a program specification interruption is taken if the second operand address does not specify integral boundaries. If a program specification interruption is taken, the instruction is suppressed. If no specification check occurs, the first operand is transferred from T to B, and the specified word of the doubleword requested from main storage is selected per D(21) and is gated to T. The first operand is now in B and the second operand is in T, which is the same condition which would exist after an RR I-Fetch.

If the RX format instruction specifies a halfword second operand, two additional functions must be performed. The desired halfword [selected per D(22)] of the word in T [selected per D(21)] must be loaded into the low-order halfword of T, and the sign bit must be propagated left to fill the high-order halfword of T.

The fixed-point add-type instructions may be divided into three functional groups: add, subtract, and compare. All add-type instructions set a CC, and all except compare instructions store the result into the first operand location.

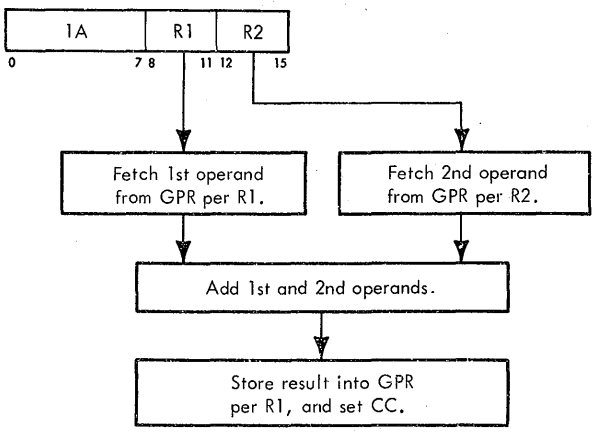
The CPU performs fixed-point add-type instructions as follows (Diagram 5-108, Sheet 2):

1. The second operand is algebraically added to the first operand and the result is stored (except for compare instructions) into the first operand location. For subtract and compare instructions, the second operand (sign bit and integer) is 2's complemented, which, in effect, inverts the sign.
2. Because of the sign notation used, and because positive numbers exist in true binary form and negative numbers exist in 2's complement form, the operand signs are treated as high-order extensions of the integers.

3. Except for Add Logical and Subtract Logical instructions, the sign bit of the result [T(32)] is used as one factor in determining the CC; carry conditions from the high-order digit and from the sign bit are tested for a fixed-point overflow condition (recorded in STAT B).
4. For all fixed-point add-type instructions, a zero result is indicated by setting STAT A.
5. For Add Logical and Subtract Logical instructions, the sign bit of the result is treated as a high-order extension of the integer, and is tested for a carry condition to determine the CC. The result of Add Logical or Subtract Logical instructions is the same as for the corresponding add or subtract instruction, except that the result is not tested for a fixed-point overflow condition and the significance of the CC is different. (See Table in Sheet 2 of Diagram 5-108.)

Add, AR (1A)

- Algebraically add 2nd operand (in GPR, per R2) to 1st operand (in GPR, per R1) and place result into 1st operand location.
- RR format:

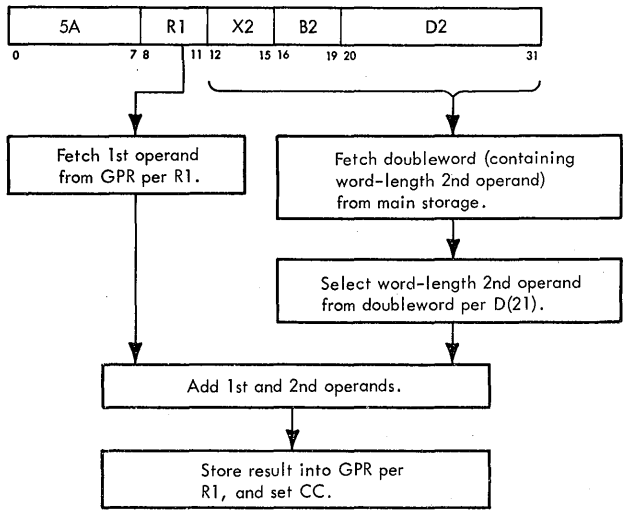


- Conditions at start of execution:
  Instruction is in E.
  1st operand is in A, B, and D.
  2nd operand is in S and T.
- CC setting:
  Result is zero (STAT A is set): CC = 0.
  Result is less than zero [T(32) = 1, and STAT's A and B are reset]: CC = 1.
  Result is greater than zero [T(32) = 0, and STAT's A and B are reset]: CC = 2.
  Overflow (STAT B is set): CC = 3.

The Add, AR, instruction algebraically adds the second operand (from the GPR per R2) to the first operand (from the GPR per R1) and stores the result into the first operand location. For the instruction execution, refer to "Add-Type Instructions" and Diagram 5-108.

Add, A (5A)

- Algebraically add 2nd operand (in storage) to 1st operand (in GPR, per R1) and place result into 1st operand location.
- RX format:



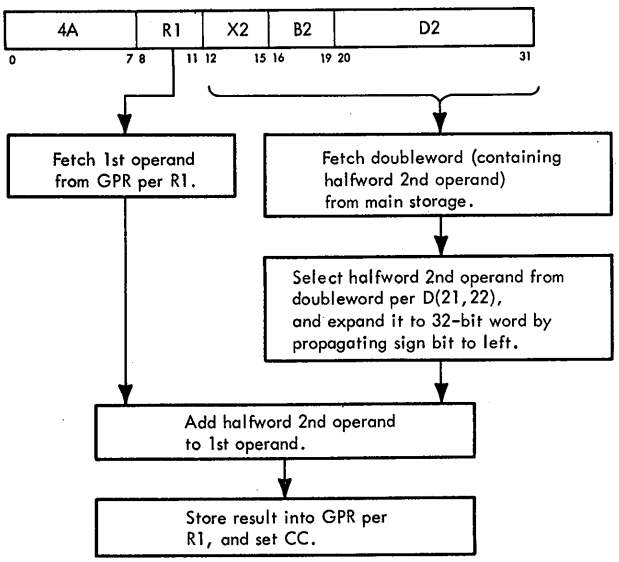
- Conditions at start of execution:
  First 16 bits of instruction are in E.
  1st operand is in S and T.
  2nd operand address is in D.
  Main storage request for 2nd operand has been issued per D.
- D(21) determines which word of doubleword contains 2nd operand: if a 1, right word; if a 0, left word.
- CC setting:
  Result is zero (STAT A is set): CC = 0.
  Result is less than zero [T(32) = 1, and STAT's A and B are reset]: CC = 1.
  Result is greater than zero [T(32) = 0, and STAT's A and B are reset]: CC = 2.
  Overflow (STAT B is set): CC = 3.

The Add, A, instruction algebraically adds the second operand (from storage) to the first operand (from the GPR per R1) and stores the result into the first operand location. D(21) determines which word of the doubleword fetched from main storage contains the word-length second operand: if D(21) = 1, the right word; if D(21) = 0, the left word. For the instruction execution, refer to "Add-Type Instructions" and Diagram 5-108.

Add Halfword, AH (4A)

- Algebraically add halfword 2nd operand (in storage) to 1st operand (in GPR per R1) and place result into 1st operand location.

- RX format:



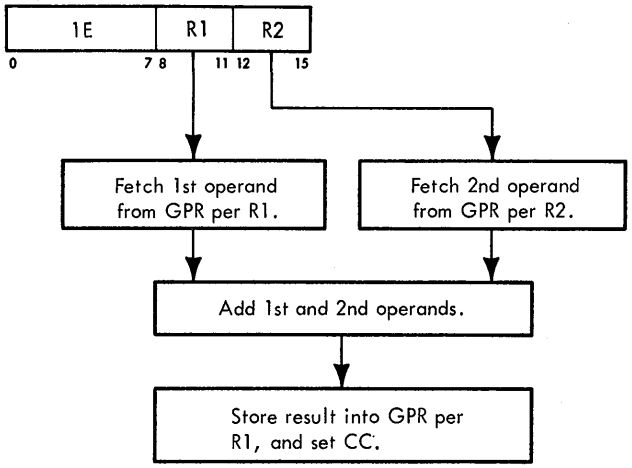
- Conditions at start of execution:
  First 16 bits of instruction are in E.
  1st operand is in S and T.
  2nd operand address is in D.
  Main storage request for 2nd operand has been issued per D.
- D(21) determines which word of doubleword contains halfword 2nd operand: if a 1, right word; if a 0, left word.
- D(22) determines which half of word contains halfword 2nd operand: if a 1, right half; if a 0, left half.
- CC setting:
  Result is zero (STAT A is set): CC = 0.
  Result is less than zero [T(32) = 1, and STAT's A and B are reset]: CC = 1.
  Result is greater than zero [T(32) = 0 and STAT's A and B are reset]: CC = 2.
  Overflow (STAT B is set): CC = 3.

The Add Halfword, AH, instruction algebraically adds the halfword second operand (from storage) to the first operand (from the GPR per R1) and stores the result into the first operand location. D(21) determines which word of the doubleword fetched from main storage contains the halfword second operand, and D(22) determines which halfword of that word contains the second operand, as follows: D(21) = 0, left word; D(21) = 1, right word; D(22) = 0, left halfword; D(22) = 1, right halfword. When the halfword second operand is selected, it is expanded to a word by propagating the sign bit through the 16 high-order bit positions of T.

For the instruction execution, refer to "Add-Type Instructions" and Diagram 5-108.

Add Logical, ALR (1E)

- Algebraically add 2nd operand (in GPR per R2) to 1st operand (in GPR per R1) and place result into 1st operand location.
- RR format:



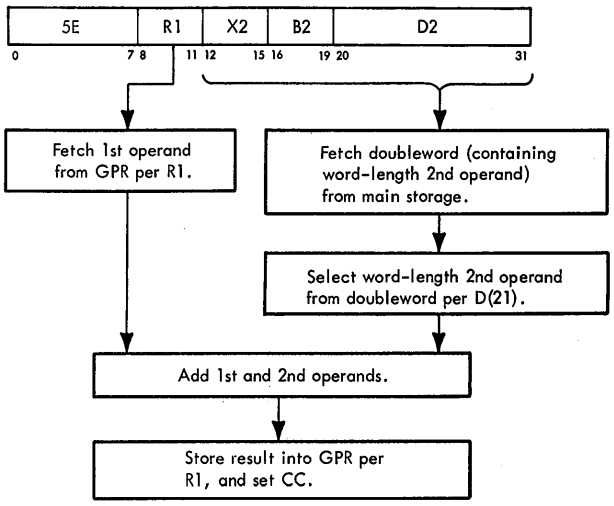
- Conditions at start of execution:
  Instruction is in E.
  1st operand is in A, B, and D.
  2nd operand is in S and T.
- CC setting:
  Result is zero and no carry from PAL(32) [STAT A is set and A(31) = 0]: CC = 0.
  Result is not zero and no carry from PAL(32) [STAT A is reset and A(31) = 0]: CC = 1.
  Result is zero and carry from PAL(32) [STAT A is set and A(31) = 1]: CC = 2.
  Result is not zero and carry from PAL(32) [STAT A is reset and A(31) = 1]: CC = 3.

The Add Logical, ALR, instruction algebraically adds the second operand (from the GPR per R2) to the first operand (from the GPR per R1) and stores the result into the first operand location. The sign bit of the sum is treated as a high-order extension of the integer, and is tested for a carry condition [A(31) = 1] to determine the CC. The sum is the same as for the AR instruction; the only difference in execution is that the sum is not tested for a fixed-point overflow condition, and that the significance of the CC's is different.

For the instruction execution, refer to "Add-Type Instructions" and Diagram 5-108.

Add Logical, AL (5E)

- Algebraically add 2nd operand (in storage) to 1st operand (in GPR per R1) and place result into 1st operand location.
- RX format:



- Conditions at start of execution:
  First 16 bits of instruction are in E.
  1st operand is in S and T.
  2nd operand address is in D.
  Main storage request for 2nd operand has been issued per D.
- D(21) determines which word of doubleword contains 2nd operand: if a 1, right word; if a 0, left word.
- CC setting:
  Result is zero and no carry from PAL(32) [STAT A is set and A(31) = 0] : CC = 0.
  Result is not zero and no carry from PAL(32) [STAT A is reset and A(31) = 0] : CC = 1.
  Result is zero and carry from PAL(32) [STAT A is set and A(31) = 1] : CC = 2.
  Result is not zero and carry from PAL(32) [STAT A is reset and A(31) = 1] : CC = 3.

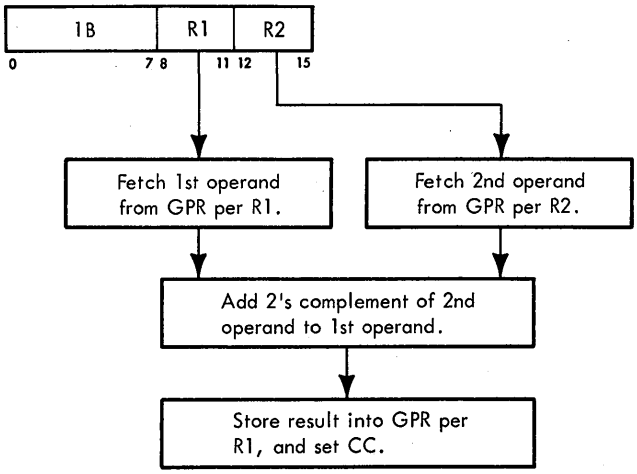
The Add Logical, AL, instruction algebraically adds the second operand (from storage) to the first operand (from the GPR per R1) and stores the result into the first operand location. D(21) determines which word of the doubleword fetched from main storage contains the word-length second operand: if D(21) = 1, the right word; if D(21) = 0, the left word.

The sign bit of the sum is treated as a high-order extension of the integer, and is tested for a carry condition [A(31) = 1] to determine the CC. The sum is the same as for the A instruction; the only difference in execution is that the sum is not tested for a fixed-point overflow condition, and that the significance of the CC's is different.

For the instruction execution, refer to "Add-Type Instructions" and Diagram 5-108.

Subtract, SR (1B)

- Algebraically subtract 2nd operand (in GPR, per R2) from 1st operand (in GPR per R1) and place result into 1st operand location.
- RR format:



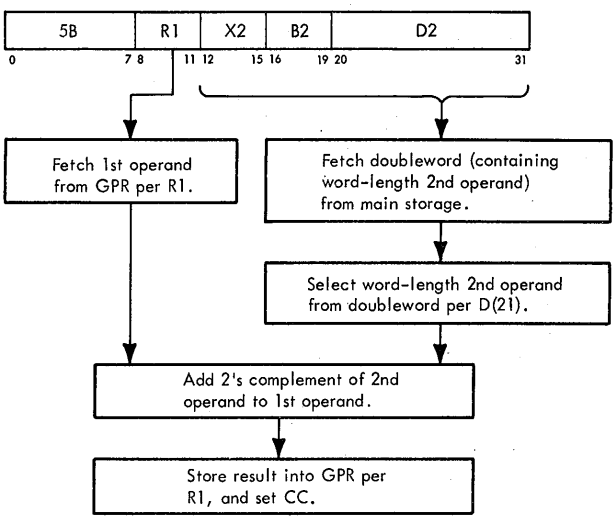
- Conditions at start of execution:
  Instruction is in E.
  1st operand is in A, B, and D.
  2nd operand is in S and T.
- CC setting:
  Result is zero (STAT A is set): CC = 0.
  Result is less than zero [T(32) = 1 and STAT's A and B are reset] : CC = 1.
  Result is greater than zero [T(32) = 0 and STAT's A and B are reset] : CC = 2.
  Overflow (STAT B is set): CC = 3.

The Subtract, SR, instruction adds the 2's complement of the second operand (from the GPR per R2) to the first operand (from the GPR per R1) and stores the result into the first operand location. The only difference between the SR and AR instructions is that the second operand is 2's complemented.

For the instruction execution, refer to "Add-Type Instructions" and Diagram 5-108.

Subtract, S (5B)

- Algebraically subtract 2nd operand (in storage) from 1st operand (in GPR per R1) and place result into 1st operand location.
- RX format:



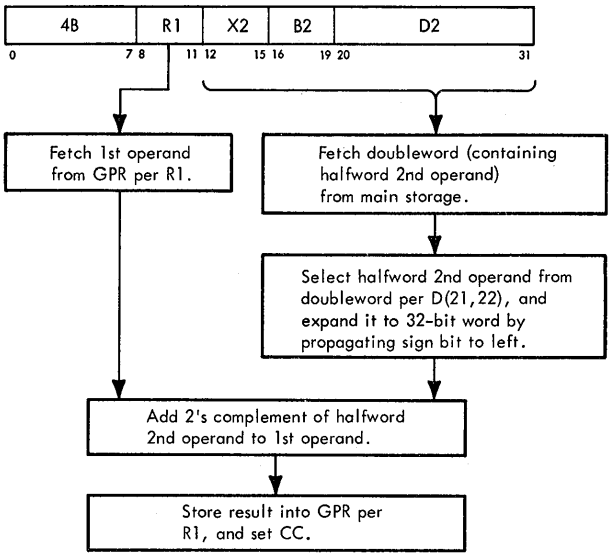
- Conditions at start of execution:
  First 16 bits of instruction are in E.
  1st operand is in S and T.
  2nd operand address is in D.
  Main storage request for 2nd operand has been issued per D.
- D(21) determines which word of doubleword contains 2nd operand: if a 1, right word; if a 0, left word.
- CC setting:
  Result is zero (STAT A is set): CC = 0.
  Result is less than zero [T(32) = 1 and STAT's A and B are reset]: CC = 1.
  Result is greater than zero [T(32) = 0 and STAT's A and B are reset]: CC = 2.
  Overflow (STAT B is set): CC = 3.

The Subtract, S, instruction adds the 2's complement of the second operand (from storage) to the first operand (from the GPR per R1) and stores the result into the first operand location. The only difference between the S and A instructions is that the second operand is 2's complemented.

For the instruction execution, refer to "Add-Type Instructions" and Diagram 5-108.

Subtract Halfword, SH (4B)

- Algebraically subtract halfword 2nd operand (in storage) from 1st operand (in GPR per R1) and place result into 1st operand location.
- RX format:



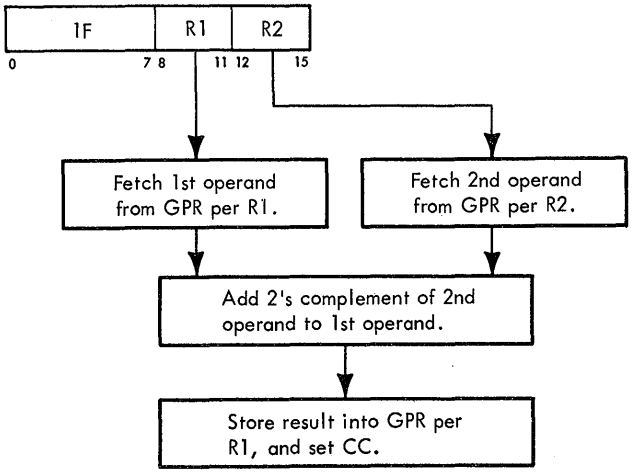
- Conditions at start of execution:
  First 16 bits of instruction are in E.
  1st operand is in S and T.
  2nd operand address is in D.
  Main storage request for 2nd operand has been issued per D.
- D(21) determines which word of doubleword contains halfword 2nd operand: if a 1, right word; if a 0, left word.
- D(22) determines which half of word contains halfword 2nd operand: if a 1, right half; if a 0, left half.
- CC setting:
  Result is zero (STAT A is set): CC = 0.
  Result is less than zero [T(32) = 1 and STAT's A and B are reset]: CC = 1.
  Result is greater than zero [T(32) = 0 and STAT's A and B are reset]: CC = 2.
  Overflow (STAT B is set): CC = 3.

The Subtract Halfword, SH, instruction adds the 2's complement of the halfword second operand (from storage) to the first operand (from the GPR per R1) and stores the result into the first operand location. The only difference between the SH and AH instructions is that the second operand is 2's complemented.

For the instruction execution, refer to "Add-Type Instructions" and Diagram 5-108.

Subtract Logical, SLR (1F)

- Algebraically subtract 2nd operand (in GPR per R2) from 1st operand (in GPR per R1) and place result into 1st operand location.
- RR format:



- Conditions at start of execution:
  Instruction is in E.
  1st operand is in A, B, and D.
  2nd operand is in S and T.

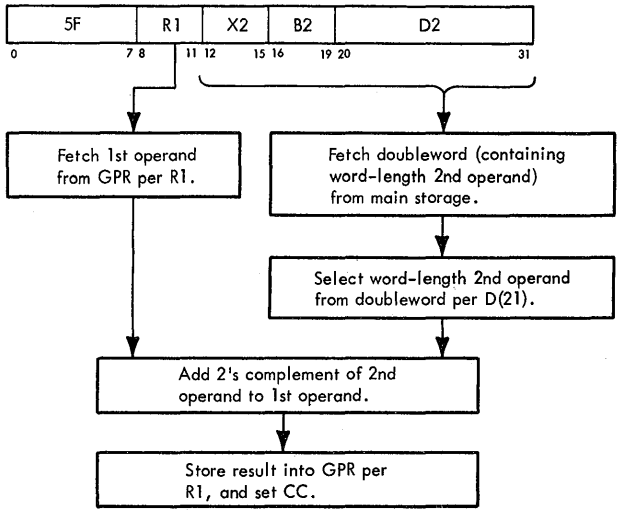
- CC setting:
  Result is not zero and no carry from PAL(32) [STAT A is reset and A(31) = 0] : CC = 1.
  Result is zero and carry from PAL(32) [STAT A is set and A(31) = 1] : CC = 2.
  Result is not zero and carry from PAL(32) [STAT A is reset and A(31) = 1] : CC = 3.

The Subtract Logical, SLR, instruction adds the 2's complement of the second operand (from the GPR per R2) to the first operand (from the GPR per R1) and stores the result into the first operand location. The sign bit of the result is treated as a high-order extension of the integer, and is tested for a carry condition [A(31) = 1] to determine the CC. The result is the same as for the ALR instruction; the difference in execution is that the second operand is 2's complemented, the result is not tested for a fixed-point overflow condition, and the significance of the CC's is different.

For the instruction execution, refer to "Add-Type Instructions" and Diagram 5-108.

Subtract Logical, SL (5F)

- Algebraically subtract 2nd operand (in storage) from 1st operand (in GPR per R1) and place result into 1st operand location.
- RX format:



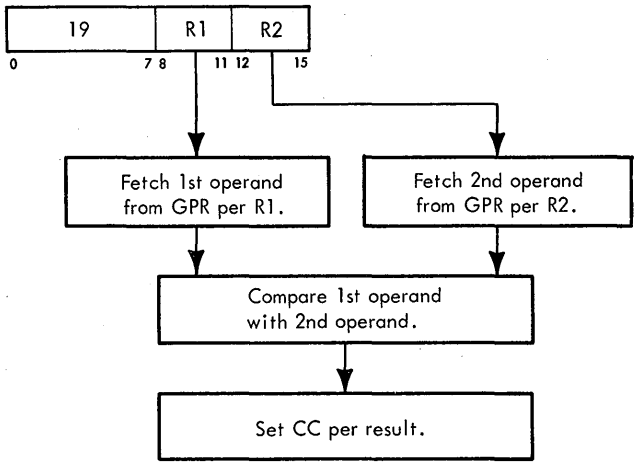
- Conditions at start of execution:
  First 16 bits of instruction are in E.
  1st operand is in S and T.
  2nd operand address is in D.
  Main storage request for 2nd operand has been issued per D.
- D(21) determines which word of doubleword contains 2nd operand: if a 1, right word; if a 0, left word.
- CC setting:
  Result is not zero and no carry from PAL(32) [STAT A is reset and A(31) = 0] : CC = 1.
  Result is zero and carry from PAL(32) [STAT A is set and A(31) = 1] : CC = 2.
  Result is not zero and carry from PAL(32) [STAT A is reset and A(31) = 1] : CC = 3.

The Subtract Logical, SL, instruction algebraically adds the 2's complement of the second operand (from storage) to the first operand (from the GPR per R1) and stores the result into the first operand location. The sign bit of the result is treated as a high-order extension of the integer, and is tested for a carry condition [A(31) = 1] to determine the CC. The result is the same as for the AL instruction; the difference in execution is that the second operand is 2's complemented, the result is not tested for a fixed-point overflow condition, and the significance of the CC's is different.

For the instruction execution, refer to "Add-Type Instructions" and Diagram 5-108.

Compare, CR (19)

- Algebraically compare 1st operand (in GPR, per R1) with 2nd operand (in GPR, per R2) and set CC according to result.
- RR format:

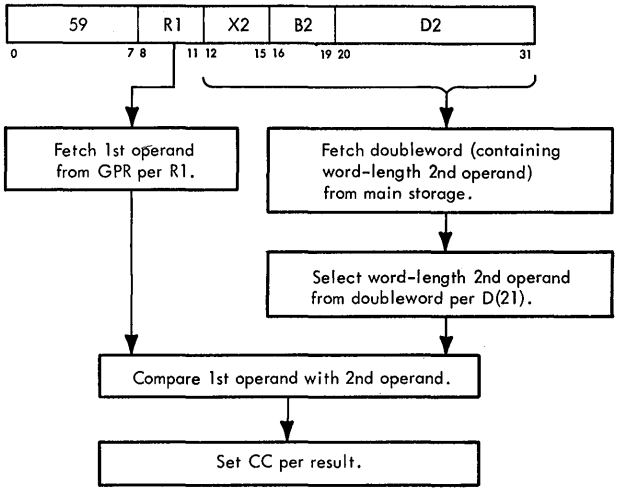


- Conditions at start of execution:
  Instruction is in E.
  1st operand is in A, B, and D.
  2nd operand is in S and T.
- CC setting:
  Operands are equal (STAT A is set): CC = 0.
  1st operand is less than 2nd operand [STAT B is set or T(32) = 1]: CC = 1.
  1st operand is greater than 2nd operand [STAT B is set and T(32) = 1, or STAT B is reset and T(32) = 0]: CC = 2.

The Compare, CR, instruction algebraically compares the first operand (from the GPR per R1) with the second operand (from the GPR per R2) and sets the CC according to the result. The compare operation is accomplished by adding the 2's complement of the second operand to the first operand and setting the CC according to the result. The result is not stored. For the instruction execution, refer to "Add-Type Instructions" and Diagram 5-108.

Compare, C (59)

- Algebraically compare 1st operand (in GPR per R1) with 2nd operand (in storage) and set CC according to result.
- RX format:



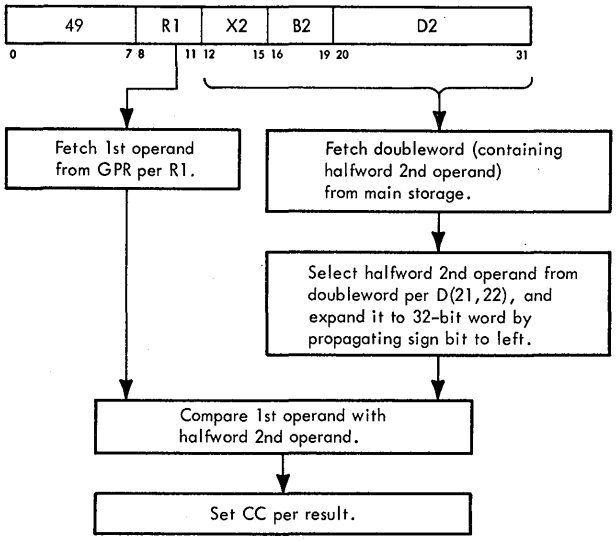
- Conditions at start of execution:
  First 16 bits of instruction are in E.
  1st operand is in S and T.
  2nd operand address is in D.
  Main storage request for 2nd operand has been issued per D.
- D(21) determines which word of doubleword contains 2nd operand: if a 1, right word; if a 0, left word.
- CC setting:
  Operands are equal (STAT A is set): CC = 0.
  1st operand is less than 2nd operand [STAT B is set or T(32) = 1]: CC = 1.
  1st operand is greater than 2nd operand [STAT B is set and T(32) = 1, or STAT B is reset and T(32) = 0]: CC = 2.

The Compare, C, instruction algebraically compares the first operand (from the GPR per R1) with the second operand (from storage) and sets the CC according to the result.

Because the word-length second operand is in main storage, D(21) determines which word of the doubleword fetched from main storage contains the second operand: if a 1, right word; if a 0, left word. The compare operation is accomplished by adding the 2's complement of the second operand to the first operand and setting the CC according to the result. The result is not stored. For the instruction execution, refer to "Add-Type Instructions" and Diagram 5-108.

Compare Halfword, CH (49)

- Algebraically compare 1st operand (in GPR per R1) with halfword 2nd operand (in storage) and set CC according to result.
- RX format:



- Conditions at start of execution:
  First 16 bits of instruction are in E.
  1st operand is in S and T.
  2nd operand address is in D.
  Main storage request for 2nd operand has been issued per D.
- D(21) determines which word of doubleword contains halfword 2nd operand: if a 1, right word; if a 0, left word.
- D(22) determines which half of word contains halfword 2nd operand: if a 1, right half; if a 0, left half.
- CC setting:
  Operands are equal (STAT A is set): CC = 0.
  1st operand is less than 2nd operand [STAT B is set or T(32) = 1] : CC = 1.
  1st operand is greater than 2nd operand [STAT B is set and T(32) = 1, or STAT B is reset and T(32) = 0] : CC = 2.

The Compare Halfword, CH, instruction algebraically compares the first operand (from the GPR per R1) with the halfword second operand (from storage) and sets the CC according to the result.

Because the halfword second operand is in main storage, D(21) determines which word of the doubleword fetched from main storage contains the halfword second operand: if a 1, right word; if a 0, left word. D(22) determines which half of that word contains the second operand: if a 1, right half; if a 0, left half. The halfword second operand is expanded to word-length by propagating the sign bit through the high-order 16-bit positions of T. The compare operation is accomplished by adding the 2's complement of the halfword second operand to the first operand and setting the CC according to the result. For the instruction execution, refer to "Add-Type Instructions" and Diagram 5-108.

MULTIPLY

There are three fixed-point multiply instructions:

1. Multiply, MR, RR format – which uses a 32-bit multiplier and multiplicand, and produces a 64-bit product.
2. Multiply, M, RX format – which uses a 32-bit multiplier and multiplicand, and produces a 64-bit product.
3. Multiply Halfword, MH, RX format – which uses a 16-bit multiplier and a 16-bit multiplicand, and produces a 32-bit product.

Note: In the Multiply, M, and Multiply Halfword, MH, instructions, the roles of the first and second operands are reversed from the roles defined in the *System/360 Principles of Operation,* SRL, Form A22-6821-6. That is, the second operand is the multiplicand and the first operand is the multiplier. (Interchanging the operand roles does not affect the product.) The result, however, still replaces the first operand.

Each of the three fixed-point multiply instructions has a unique initialization routine. The initialization routines (Diagram 5-109, Sheet 1, FEMDM) perform a specification test, set E(12–15) to 15, set the STC to 3, and establish the operands in S and T as follows:

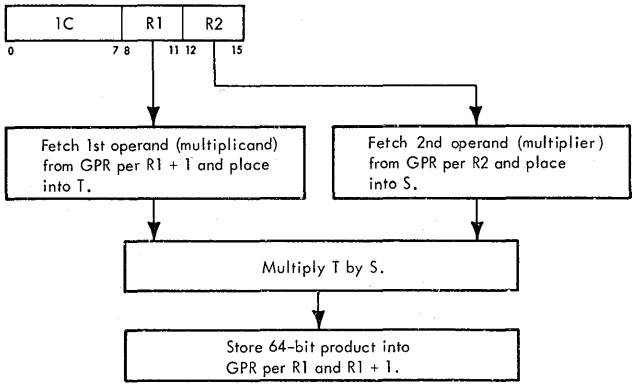
1. Multiply, MR, RR format – Transfers the multiplier (second operand from GPR per R2) to S and the multiplicand (first operand from GPR per R1 + 1) to T.
2. Multiply, M, RX format – Transfers the multiplicand (first operand from GPR per R1 + 1) to S and the multiplier (second operand from main storage) to T.
3. Multiply Halfword, MH, RX format – Transfers the multiplicand (first operand from GPR per R1) to S and the multiplier (halfword second operand from main storage, expanded to word length by propagating the sign bit through the 16 high-order bit positions) to T.

A common multiply microprogram is then entered. Multiples of T are selected per bit-pairs from S and are added to a partial product in B to form a new partial product. Low-order partial product bit-pairs are accumulated in F and SAL. When SAL has accumulated a

partial product byte, it is stored into S, replacing the byte of S already used. Sixteen multiply cycles are taken until a word (four bytes) of product is accumulated in S. PAL now contains the high-order word of the product and S contains the low-order word of the product. The product is stored in an even/odd pair of GPR's specified by R1 and R1 + 1, and an end-op cycle is taken to terminate the operation. (For the Multiply Halfword instruction, only the low-order 32 bits of the product are stored into the GPR per R1.)

Multiply, MR (1C)

- Multiply 1st operand (in GPR per R1 + 1) by 2nd operand (in GPR per R2) and place 64-bit product into 1st operand location (in GPR per R1 and R1 + 1).
- RR format:



- Conditions at start of execution:
  Instruction is in E.
  Contents of even-address GPR specified by R1 are in A, B, and D (not used).
  Multiplicand (1st operand) is in odd-address GPR specified by R1 + 1.
  2nd operand (multiplier) is in S and T.
- Multiple selection bits (M1,M2) are selected from multiplier (in S) per E(12–15).
- Multiples of multiplicand (in T) are selected by M1,M2 bits and 'TX' trigger.
- Multiples of multiplicand are added to partial product in B.
- Partial product bits from B(66,67) are accumulated in SAL and F per E(14,15).
- SAL contains byte of partial product when filled.
- SAL is transferred to correct byte in S per STC.
- When last (4th) byte is transferred to S, multiplier in S is replaced by low-order half of product; high-order bits are in PAL.

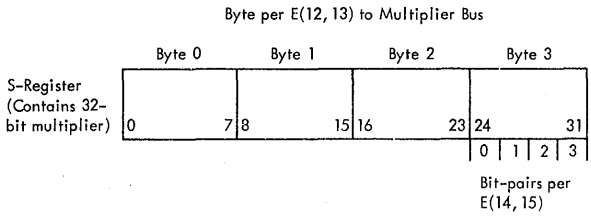
The Multiply, MR, instruction multiplies the contents of T (multiplicand) by the contents of S (multiplier). Because both the multiplier and the multiplicand are 32-bit signed integers, the product is a 64-bit signed integer and must be stored into an even/odd pair of GPR's.

A flowchart of the operation is shown in Diagram 5-109, FEMDM. To correctly specify the even/odd GPR pair, the R1 field of the instruction must refer to an even-address GPR or a program specification interruption occurs. After R1 is tested to see whether it is even, 15 and 3 are placed into E(12–15) and the STC, respectively. The value in E(12–15) selects the correct multiple selection bits (M1,M2) from the multiplier in S, and the value in the STC correctly positions the partial product byte in S. Each value is sequentially reduced during the operation. The value in T is now destroyed, and the multiplicand is transferred from the GPR per R1 + 1 to T. At this point, S contains the multiplier and T contains the multiplicand.

Execution of the MR instruction occurs in three iterative steps.

1. Selection of multiplicand multiples.
2. Addition of multiplicand multiples to partial product to form a new partial product.
3. Extraction of partial product bits to form a product.

Multiple selection bits (M1,M2) are selected from S per E(12–15), which is initially set to 15 (decimal) and is decremented by 1 each time multiple selection bits are selected. E(12,13) determines which byte of S is gated to the multiplier (MPR) bus, and E(14,15) determines which bit-pair of the selected byte is used to set M1,M2:



Bits M1,M2, which have the same bit configuration as the bit-pair selected from S by E(12–15), are used with the 'TX' trigger to select a multiple of the multiplicand which will be added to a partial product to develop a new partial product. The multiple selected for all combinations of M1,M2 bits and the state of the 'TX' trigger is listed in Table 3-2.

Four bit configurations of M1,M2 are possible, representing decimal values of 0, 1, 2, and 3. Five multiplicand multiples can be selected: 0 x T, 1 x T, -1 x T, 2 x T and -2 x T. Multiplicand multiples are developed and applied as follows:

1. 0 x T: Zero's are added to the partial product.

2. 1 x T: The multiplicand (in T) is added to the partial product.
3. -1 x T: The multiplicand is added in 2's complement form to the partial product.
4. 2 x T: The multiplicand is gated to PAA, shifted left one bit position (in effect, doubles its value), and added to the partial product.
5. -2 x T: The multiplicand, shifted left one bit position, is added in 2's complement form to the partial product.

No provision has been made to develop a multiple of the multiplicand of 3 x T. Therefore, when M1,M2 has a decimal value of 3, a multiple of -1 x T is selected and the 'TX' trigger is set and remains set into the next multiply cycle. Note in Table 3-2 that when the 'TX' trigger is set during the selection of a multiple, it has the effect of increasing the value of the multiple by 1 for the corresponding value of M1,M2. Because the partial product is shifted right two positions before each multiple is added, the value of the multiple is increased by a factor of 4. The effect of the 'TX' trigger's being set is to increase the value of the multiple (defined by M1,M2) by 4. Thus the multiplicand is, in effect, multiplied by -1 and +4 (plus the multiple which would have been selected if the 'TX' trigger were not set).

When the partial product is shifted right two positions (right 4 and left 2) after each addition of the selected multiple of the multiplicand, the low-order bit-pair of the partial product is gated to SAL per E(14,15) and is added to the contents of F. When a byte of the partial product is accumulated in F, the byte is transferred to S per the STC (via SAL), replacing the byte of multiplier which has been used.

When the low-order partial product bit-pair is gated to SAL, E(12–15) has been decremented twice and the low-order bit pair is stored into F(6,7) per the value of 01 in E(14,15) (C of Sheet 3, Diagram 5-109, FEMDM).

As previously mentioned, the partial product bits are transferred via SAL to F. This register accumulates a partial product byte (eight bits). When the last two partial product bits, required to complete a byte, are selected [E(14,15) = 10], the contents of F are transferred to SAL, where the two partial product bits are positioned correctly. This byte is then transferred to S and positioned according to the value of the STC. At this point, the STC is equal to 011, thus placing the partial product byte into S(24–31). The STC and E(12–15) are then reduced by 1, and selection of partial product bits is continued. The microprogram remains in the multiple selection loop until E(12,13) = 00 when tested. At this time, E(12–15) contains 0001 and is decremented to 0000; and the microprogram enters the multiply termination routine.

During the multiply termination routine, five events take place:

1. A multiple is selected for the high-order bit-pair in S.
2. Because there is a 2-cycle lag between the selection of a multiple and the storage of the corresponding partial product bit-pair into F, E(12–15) wraps around to 1110 to control the storage of the last two bit-pairs of the partial product in F (via SAL).
3. The high-order partial product byte is transferred from F to S per the STC. (The last bit-pair, however, does not go to F, because the last byte is gated directly from SAL to S.)
4. The high-order word of the partial product is transferred from PAL to T, and from T to the GPR specified by R1.
5. The low-order word of the partial product in S is transferred to T, and from T to the GPR specified by R1 + 1.

This sequence places the complete product into the even/odd GPR pair in LS. An end-op cycle is then taken, and the operation is finished.

The product sign follows the rules of algebra (except that the sign of a zero product is plus); however, the sign bits are manipulated as though they were high-order extensions of the integer throughout the multiply operation. Two multiply examples follow; the first, in Figure 3-6, uses two positive operands, and the second, in Figure 3-7, uses the same operands with minus signs. Note that the product is the same in both cases with no special handling of the sign bits required.

Table 3-2. Value of Multiple Determined by Multiple Selection Bits (Fixed-Point)

Multiple Selection Bits		'TX' Trigger	T-Register Times Value Indicated	Set 'TX' Trigger
M1	M2			
0	0	0	0 x T	No
0	1	0	1 x T	No
1	0	0	2 x T	No
1	0	0	-2 x T	No†
1	1	0	-1 x T (2's Complement)	Yes
0	0	1	1 x T	No
0	1	1	2 x T	No
1	0	1	-1 x T (2's Complement)	Yes
1	1	1	0 x T	Yes

† Used on last multiple selection if multiplicand is negative.

	Hex	Binary	Decimal
Multiplicand (in T)	001A	0000 0000 0001 1010	+26
Multiplier (in S)	00A3	0000 0000 1010 0011	x +163
Product (to S via F)	108E	0001 0000 1000 1110	+4238

1. M1, M2 bits are derived from S per E(12-15).
2. See Table 3-2 for multiple selection.

M1, M2	'TX' Tgr	Multiple	E(12-15)	Remarks	Contents in PAL	Accumulate Bit-Pairs in F; Transfer Bytes to S
	0		1111	Initial partial product	0000 0000 0000 0000	
1 1	0	-1 x T	1111	+ T (2's complement)	1111 1111 1110 0110	
	1		1110	New partial product	1111 1111 1110 0110	
	1		1110	Right 2 positions	1111 1111 1111 1001	10
0 0	1	1 x T	1110	+ T (true)	0000 0000 0001 1010	10
	0		1101	New partial product	0000 0000 0001 0011	
	0		1101	Right 2 positions	0000 0000 0000 0100	11
1 0	0	2 x T	1101	+ 2T (true, left 1)	0000 0000 0011 0100	1110
	0		1100	New partial product	0000 0000 0011 1000	
	0		1100	Right 2 positions	0000 0000 0000 1110	00
1 0	0	2 x T	1100	+ 2T (true, left 1)	0000 0000 0011 0100	00 1110
	0		1011	New partial product	0000 0000 0100 0010	
	0		1011	Right 2 positions	0000 0000 0001 0000	10
0 0	0	0 x T	1011	Add 0's	0000 0000 0000 0000	1000 1110
	0		1010	New partial product	0000 0000 0001 0000	
	0		1010	Right 2 positions	0000 0000 0000 0100	00
0 0	0	0 x T	1010	Add 0's	0000 0000 0000 0000	00
	0		1001	New partial product	0000 0000 0000 0100	
	0		1001	Right 2 positions	0000 0000 0000 0001	00
0 0	0	0 x T	1001	Add 0's	0000 0000 0000 0000	0000
	0		1000	New partial product	0000 0000 0000 0001	
	0		1000	Right 2 positions	0000 0000 0000 0000	01
0 0	0	0 x T	1000	Add 0's	0000 0000 0000 0000	01 0000
	0		0111	New partial product	0000 0000 0000 0000	
	0		0111	Right 2 positions	0000 0000 0000 0000	00
0 0	0	0 x T	0111	Add 0's	0000 0000 0000 0000	0001 0000
						S ← 0001 0000 1000 1110
						1 0 8 E

and so on

Notes:
1. T = T-register.
2. For RX format instruction, reverse multiplier and multiplicand.

Figure 3-6. Fixed-Point Multiply, Example No. 1 (RR Format)

	Hex	Binary	Decimal
Multiplicand (in T)	FFE6	1111 1111 1110 0110	-26
Multiplier (in S)	FF5D	1111 1111 0101 1101	x -163
Product (to S via F)	108E	0001 0000 1000 1110	+4238

1. M1, M2 bits are derived from S per E(12-15).

2. See Table 3-2 for multiple selection.

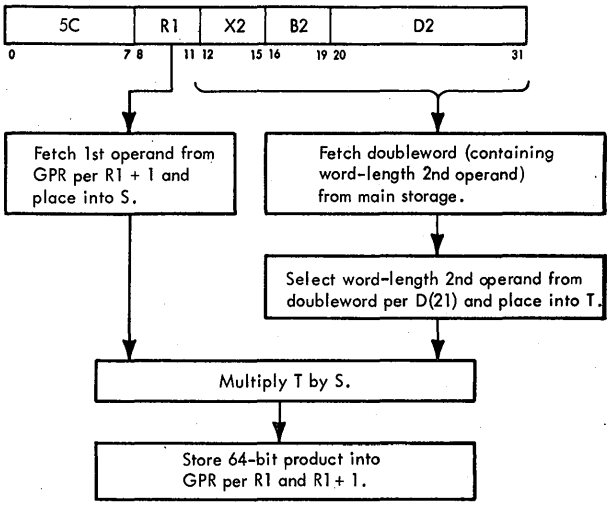
M1, M2	'TX' Tgr	Multiple	E(12-15)	Remarks	Contents in PAL	Accumulate Bit-Pairs in F; Transfer Bytes to S
	0		1111	Initial partial product	0000 0000 0000 0000	
0 1	0	1 x T	1111	+ T (true)	1111 1111 1110 0110	
	0		1110	New partial product	1111 1111 1110 0110	
	0		1110	Right 2 positions	1111 1111 1111 1001	10
1 1	0	-1 x T	1110	+ T (2's complement)	0000 0000 0001 1010	10
	1		1101	New partial product	0000 0000 0001 0011	
	1		1101	Right 2 positions	0000 0000 0000 0100	11
0 1	1	2 x T	1101	+ 2T (true, left 1)	1111 1111 1100 1100	1110
	0		1100	New partial product	1111 1111 1101 0000	
	0		1100	Right 2 positions	1111 1111 1111 0100	00
0 1	0	1 x T	1100	+ T (true)	1111 1111 1110 0110	00 1110
	0		1011	New partial product	1111 1111 1101 1010	
	0		1011	Right 2 positions	1111 1111 1111 0110	10
1 1	0	-1 x T	1011	+ T (2's complement)	0000 0000 0001 1010	1000 1110
	1		1010	New partial product	0000 0000 0001 0000	
	1		1010	Right 2 positions	0000 0000 0000 0100	00
1 1	1	0 x T	1010	Add 0's	0000 0000 0000 0000	00
	1		1001	New partial product	0000 0000 0000 0100	
	1		1001	Right 2 positions	0000 0000 0000 0001	00
1 1	1	0 x T	1001	Add 0's	0000 0000 0000 0000	0000
	1		1000	New partial product	0000 0000 0000 0001	
	1		1000	Right 2 positions	0000 0000 0000 0000	01
1 1	1	0 x T	1000	Add 0's	0000 0000 0000 0000	01 0000
	1		0111	New partial product	0000 0000 0000 0000	
	1		0111	Right 2 positions	0000 0000 0000 0000	00
1 1	1	0 x T	0111	Add 0's	0000 0000 0000 0000	0001 0000
						S ← 0001 0000 1000 1110
			and so on			1 0 8 E

Notes:
1. T = T-register.
2. For RX format instructions, reverse multiplier and multiplicand.

Figure 3-7. Fixed-Point Multiply, Example No. 2 (RR Format)

Multiply, M (5C)

- Multiply 1st operand (in GPR per R1 + 1) and 2nd operand (in storage) and place 64-bit result into 1st operand location (in GPR per R1 and R1 + 1).
- See Note under "Multiply."
- RX format:

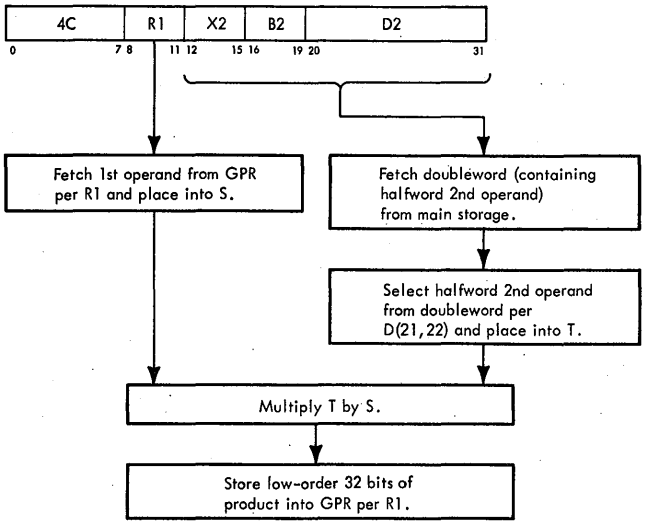


- Conditions at start of execution:
  First 16 bits of instruction are in E.
  Contents of even-address GPR per R1 are in S and T (not used).
  1st operand is in odd-address GPR per R1 + 1.
  2nd operand address is in D.
  Main storage request for 2nd operand has been issued per D.
- D(21) determines which word of doubleword contains 2nd operand: if a 1, right word; if a 0, left word.

The Multiply, M, instruction multiplies the contents of T (second operand from storage) and the contents of S (first operand, from GPR per R1 + 1), and stores the 64-bit product into the GPR's per R1 and R1 + 1. Once the operands have been obtained, the operation is identical to that of the Multiply, MR, instruction, except that the roles of the multiplier and multiplicand are reversed. (See Note under "Multiply" and Diagram 5-109.)

Multiply Halfword, MH (4C)

- Multiply 1st operand (in GPR per R1) and halfword 2nd operand (in storage) and place low-order 32 bits of result into 1st operand location.
- See Note under "Multiply".
- RX format:



- Conditions at start of execution:
  First 16 bits of instruction are in E.
  1st operand is in S and T.
  2nd operand address is in D.
  Main storage request for 2nd operand has been issued per D.
- D(21) determines which word of doubleword contains halfword 2nd operand: if a 1, right word; if a 0, left word.
- D(22) determines which half of word contains halfword 2nd operand: if a 1, right half; if a 0, left half.

The Multiply Halfword, MH, instruction multiplies the contents of T (expanded halfword second operand from main storage) and the contents of S (first operand from GPR per R1) and stores the 32 low-order bits of the product into the first operand location. D(21,22) determines the location of the second operand within the doubleword obtained from main storage. The second operand is then expanded to a word-length operand by propagating the sign bit through the high-order 16 bit positions of T.

From this point, the operation is identical to that of the Multiply, MR, instruction, except that the roles of the multiplier and multiplicand are reversed. (See Note under "Multiply.") For a flowchart of the operation, see Diagram 5-109.

DIVIDE

Fixed point division is performed by repetitive reduction of the dividend by multiples of the divisor to obtain a

remainder whose value is less than that of the divisor, and to accumulate partial quotient (PQ) bits derived from the partial remainders to form a quotient. The basic nonrestoring method is used, with the dividend in true form. Nonrestoring division means that if a negative remainder is obtained during the reduction cycles (an overdraw has been made), the remainder is not corrected, but instead the next divisor multiples are made positive until the remainder becomes positive again. (In most valid divide operations, the first reduction cycle is an intentional overdraw.)

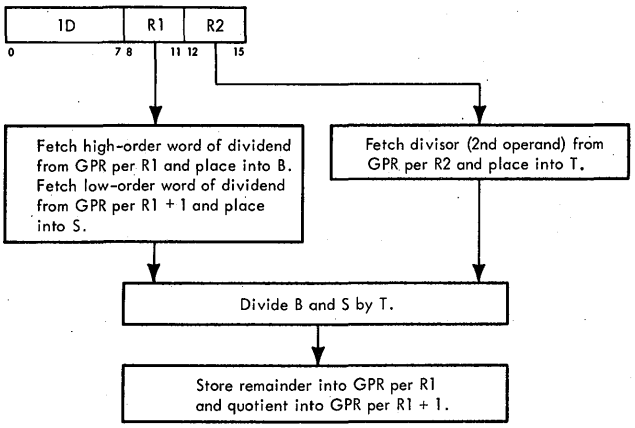
There are two fixed-point divide instructions: Divide, DR, RR format, and Divide, D, RX format. Each has a unique initialization routine. The initialization routines: provide a specification test; place the low-order half of the dividend into S and the high-order half of the dividend into B, in true form regardless of the sign; place the divisor into T; store the signs for later use; and set E(12–15) and the STC to 0.

A common divide microprogram is then entered. Two bits of the low-order dividend are appended to the high-order dividend. A multiple of the divisor is selected to reduce the dividend to a partial remainder. The inverted sign of the partial remainder is stored into F as a partial quotient (PQ) bit. Thirty-two such reduction cycles are taken, accumulating PQ bits in F until a byte of quotient is obtained. The quotient byte is stored into S, replacing the byte of the low-order dividend which has been used. When four quotient bytes have been stored into S, S contains the quotient and B contains the remainder.

The microprogram now enters one of four termination routines, determined by the sign of the divisor and the form (true or 2's complement) of the quotient. The termination routines establish the proper sign and form of the remainder and quotient, according to the convention of fixed-point arithmetic and the rules of algebra. The remainder and quotient are then stored in an even/odd pair of GPR's specified by R1 and R1 + 1, and an end-op cycle is taken to terminate the operation.

Divide, DR (1D)

- Divide 1st operand (in GPR per R1 and R1 + 1) by 2nd operand (in GPR per R2) and place result into 1st operand location (remainder in GPR per R1; quotient in GPR per R1 + 1).
- RR format: (See adjoining column.)
- Conditions at start of execution:
  Instruction is in E.
  High-order half of dividend (1st operand) is in A, B, and D.
  Low-order half of dividend is in GPR per R1 + 1.
  Divisor (2nd operand) is in S and T.



The Divide, DR, instruction divides the contents of B (high-order bits of dividend) and S (low-order bits of dividend) by the contents of T (divisor).

The dividend is a 64-bit signed integer occupying an even/odd pair of GPR's addressed by R1 and R1 + 1, respectively. To correctly specify the even/odd pair of GPR's, R1 must refer to an even-numbered GPR or a program specification interruption occurs. A 32-bit signed remainder and a 32-bit signed quotient replace the dividend in the even-numbered and odd-numbered GPR, respectively, of LS. The divisor is also a 32-bit integer.

Two bits of the low-order half of the dividend are first placed into the high-order half of the dividend. A multiple of the divisor is then selected and subtracted from the high-order half of the dividend to form a partial remainder. The resultant value determines the partial quotient (PQ) bit which is placed into F to accumulate the bits until a PQ byte is available. This PQ byte is transferred to S, which contains the low-order half of the dividend, and replaces those bits that have already been used in the operation. This action continues until a complete quotient and remainder are available, at which time they are stored into LS and an end-op cycle is taken.

The sign of the quotient is determined algebraically; four possible combinations of signs can occur:

1. + dividend, + divisor, + quotient, + remainder.
2. - dividend, - divisor, + quotient, - remainder.
3. + dividend, - divisor, - quotient, + remainder.
4. - dividend, + divisor, - quotient, - remainder.

Note that if the dividend and divisor signs are alike the quotient is positive; if unlike the quotient is negative. Note also that the sign of the remainder is the same as the sign of the dividend, except for a zero result, which is always positive.

When the relative magnitude of the dividend and divisor is such that the quotient cannot be expressed by a 32-bit signed integer, a program fixed-point divide interruption occurs. When this happens, the instruction is suppressed, leaving the dividend unchanged in local storage.

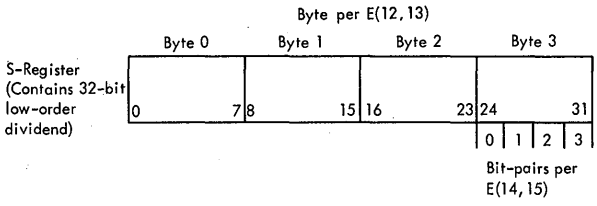
General Discussion

- Multiple selection bits are determined by E(12–15) and S bits.
- Multiples of divisor are determined by 'DVDL1' or 'DVDL0' micro-order, carry from PAL(28), and T(32).
- PQ bits are transferred and accumulated in F per E(14,15), and 'DVDL1' or 'DVDL0' micro-order.
- F contains byte of partial quotient when filled.
- Contents of F are transferred to correct byte in S per STC.

Execution of the DR instruction occurs in three iterative steps:

1. Transfer of bits from low-order half of dividend to high-order half of dividend.
2. Selection of divisor multiple.
3. Determination of quotient bits.

Selection of the two bits from the low-order half of the dividend is shown in B of Sheet 4, Diagram 5-110, FEMDM. Multiple selection bits (M1,M2) are selected from S per E(12–15), which is initially set to 0 and is incremented by 1 after the selection of each pair of multiple selection bits. E(12,13) determines which byte of S is selected, and E(14,15) determines which bit-pair of the selected byte will be used to set M1, M2:



M1,M2 has the same bit configuration as the bit-pair selected from S by E(12–15), and is inserted into B(64,65) [via PAL(64,65)] where it extends the high-order portion of the dividend in B.

Selection of the divisor multiple is shown in A of Sheet 4, Diagram 5-110. The factors which determine the divisor multiple are:

1. The carry condition from PAL(28) from the previous add cycle.
2. The state of T(32), which is the divisor sign bit.
3. The 'DVDL0' or 'DVDL1' micro-order.

Four divisor multiples are developed and applied as follows:

1. TL0 (+1 x T): The divisor in T is added to the partial remainder in AB.
2. TCL0 (-1 x T): The 2's complement of the divisor in T is added to the partial remainder in AB.
3. TL1 (+2 x T): The divisor in T is shifted left one bit position (in effect, doubled in value) and is added to the dividend or partial remainder in AB.
4. TCL1 (-2 x T): The 2's complement of the divisor in T is shifted left one bit position and is added to the dividend or partial remainder in AB.

The first divisor multiple is arbitrarily set to +2 x T if the divisor is negative (STAT G set) or to -2 x T if the divisor is positive (STAT G reset). This selection is done for two reasons:

1. The carry condition from PAL(28), which is normally a factor in selecting a divisor multiple, is meaningless at this time, because no previous reduction cycle has taken place.
2. If the relative magnitude of the dividend and divisor allows, the first reduction must be an overdraw. (Except for one special case, if the first reduction cycle does not result in an overdraw, the quotient and remainder will be invalid.)

Determination of the PQ bit is shown in C of Sheet 4, Diagram 5-110. The result of adding the selected divisor multiple to the dividend or partial remainder is stored into AB as a new partial remainder. The PQ bit is decoded as the inverse of A(28), and is gated to F (via SAL) per E(14,15) and the 'DVDL0' or 'DVDL1' micro-order in effect at that time. When a PQ byte has been accumulated in F, it is gated to S per the STC, where it replaces the byte of S which has already been used. When four PQ bytes have been stored into S, the complete quotient is in S and the remainder is in B.

At this point, STAT G is tested. (Recall that STAT G was set to the sign of the divisor.) If set, the divisor is negative and is in 2's complement form. Because the first reduction cycle is an attempt to overdraw the dividend, the negative divisor is shifted left 1 bit position (in effect, doubled in value) to PAA(31–62). If STAT G is reset, the divisor is positive and must be 2's complemented and shifted left one bit position to reduce the dividend. During the addition, the sign of the divisor is propagated into PAL(24–31), and the result (remainder) is gated to AB(24–67). A(28) is then tested to determine the PQ bit. A remainder in true form [A(28) = 0] causes a 1-bit to be selected for the PQ bit; a 2's complement remainder [A(28) = 1] causes a 0-bit to be selected for the PQ bit.

Two micro-orders, 'DVDL0' and 'DVDL1', are alternately used in the divide algorithm. Each micro-order has two functions: (1) to determine the location of the PQ bit in SAL (and F) from the bit-pair selected by E(14,15), and (2) to determine the shifting of the divisor multiple to PAA. [The carry condition from PAL(28) and the state of T(32) determine whether the multiple will be in true or 2's complement form.] The

'DVDL0' micro-order causes the selected PQ bit to be placed into the odd SAL bit position of the bit-pair selected by E(14,15), thus locating the PQ bit in the PQ byte being accumulated in F. This micro-order also determines that the divisor multiple will be gated to PAA(32–63) (no shift). The 'DVDL1' micro-order causes the selected PQ bit to be placed into the even SAL bit position, and also determines that the divisor multiple will be gated to PAA(31–62) (shifted left 1 bit position).

Detailed Discussion

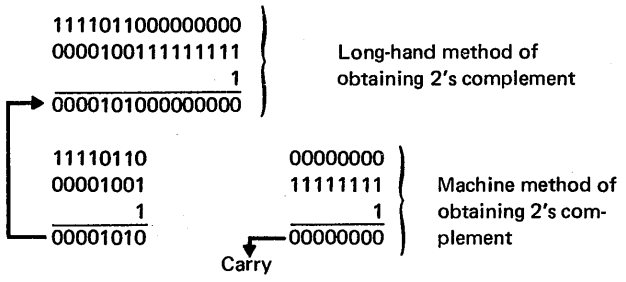
- Select M1,M2 bits from S per E(12–15).
- Insert M1,M2 bit-pair as low-order extension of high-order dividend in B.
- Select divisor multiple per: carry condition from PAL(28), divisor sign [T(32)], and 'DVDL0' or 'DVDL1' micro-order.
- Reduce dividend (or remainder) in AB by divisor multiple selected.
- Determine PQ bits per A(28).
- Accumulate PQ bits in F to form PQ byte.
- Accumulate PQ bytes in S to form quotient.
- Determine validity of quotient and remainder.

A flowchart of the DR instruction is shown in Diagram 5-110. At the start of execution, the instruction is in E, the high-order half of the dividend is in A, B, and D, and the second operand (divisor) is in S and T. (As previously mentioned, the dividend occupies an even-odd pair of GPR's.) To correctly specify the even-odd pair of GPR's, the R1 field of the instruction must refer to an even-numbered GPR, or a program specification interruption occurs.

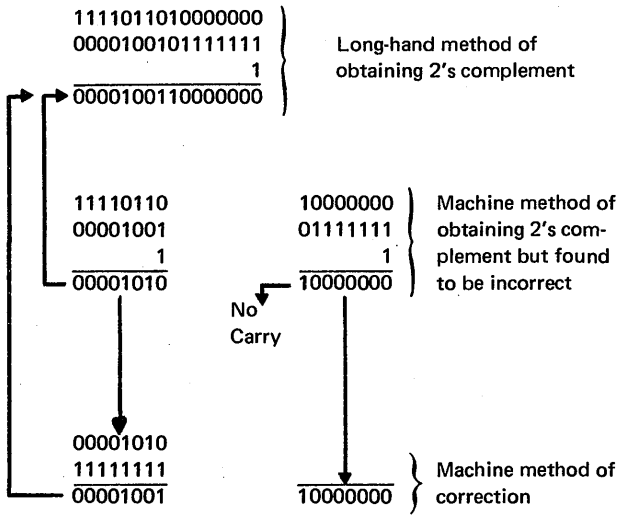
The states of B(32) and T(32) are first tested. B, at this time, contains the high-order half of the dividend, and T contains the divisor. If B(32) = 1, STAT B is set; if T(32) = 1, STAT G is set. These STAT's are used in later operations to determine the correct sign of the quotient and remainder, and to obtain a dividend in true form in B and S.

Recall that the dividend must be in true form. Because the dividend is a 64-bit operand and the maximum operable word length is 32 bits, the high-order half and the low-order half of the dividend must be treated separately when determining the true value of the dividend. To obtain a true value of a negative dividend, the dividend must be 2's complemented. Because the complementation process is always accomplished on a 2's complement basis during fixed-point operations, and because the high-order and low-order halves of the dividend are treated separately, the value of the high-order bits may be incorrect when in the true form. To prevent an incorrect high-order dividend from being operated on, the value of the low-order dividend bits is tested by checking for a carry out of the high-order bit location when 2's complemented. If a carry occurs, the high-order dividend bits are in the correct form. If a carry does not occur, the high-order bits are incorrect and a minus 1 must be added to the bits to obtain the desired dividend value. The following two examples illustrate this method (using only a 16-bit double word):

Example 1: Correct Value Obtained.



Example 2: Incorrect Value Obtained and How It Is Corrected.



Sheet 2 of Diagram 5-110 illustrates the method used by the CPU to obtain a positive dividend value in B and S.

With the correct values of the dividend in B and S and the divisor in T, E(12–15) is set to 0000 to allow selection of the first byte of the low-order half of the dividend. The first partial remainder is obtained by transferring the contents of B (high-order half of dividend) to PAB(32–63) and placing the divide multiple selection bits, M1,M2, into PAL(64,65). The divide multiple selection bits are determined by decoding E(12–15) and the S bits, as previously described.

After selection of the M1,M2 bits, E(12–15) is incremented by 1, setting up conditions for selection of

the next M1,M2 bits. The resultant value of the addition of M1,M2 to the high-order half of the dividend is transferred to AB(24–67), from where it is shifted left 2 into the parallel adder, introducing the M1,M2 bits into the partial remainder.

Recall that STAT G was set to the sign value of the divisor. At this point, STAT G is tested (Sheet 3 of Diagram 5-110.) If set, the divisor is negative and is in 2's complement form, and is transferred to the parallel adder, shifting left 1 to PAA(31–62). If STAT G is reset, the divisor is positive and must be 2's complemented before being transferred to the parallel adder so that it can reduce the dividend. During the addition, the sign of the divisor is propagated into PAL(24–31) and is checked after the addition to determine the PQ-bit setting.

At this point in the operation, the 'DVDL0' micro-order is in effect and causes the first PQ bit to be gated to the odd SAL bit position of the high-order bit-pair of SAL [E(14,15) = 01] (C of Sheet 4, Diagram 5-110). The first PQ bit is extraneous, because it reflects the condition of A(28) before the first reduction cycle, and it is replaced by a valid bit 2 cycles later. The result of the first reduction is placed into AB.

A 'DVDL1' micro-order is issued next. This micro-order causes selection of a PQ bit per A(28). (AB presently contains the result of the first reduction.) If the result of the first reduction is negative [A(28) = 1], a 0 is the selected PQ bit; if positive [A(28) = 0], a 1 is the selected PQ bit. Because the 'DVDL1' micro-order is now in effect, the PQ bit is gated to the even SAL bit position of the high-order bit-pair of SAL [E(14,15) = 01], and SAL is gated to F. The contents of AB are transferred to PAB, and the divisor multiple [selected by the carry condition of PAL(28), the state of T(32), and the 'DVDL0' micro-order] is gated to PAA. The result of the addition, together with a new pair of multiple selection bits, is gated to AB as a new partial remainder. The next divisor multiple is selected per the carry condition of PAL(28), the status of T(32) (which remains the same for the entire divide operation), and the 'DVDL1' micro-order. Refer to Table 3-3 for the value of the divisor multiple for all conditions.

Selection of the PQ bits and divisor multiples occurs as just described until a complete byte of the PQ is available: (1) the 'DVDL0' micro-order is issued, and the PQ bit obtained from the last addition is stored into the odd register location determined by E(14,15), (2) the next divisor multiple is selected, placed in the adder, and added to the partial remainder, and (3) a 'DVDL1' micro-order is issued. The 'DVDL1' micro-order accomplishes the same operation as the 'DVDL0' micro-order, except that it places the PQ bit into the even GPR location per E(14,15) and shifts the selected multiple left 1. (Refer to Sheet 3, Diagram 5-110.)

When a byte of the PQ is obtained, it is transferred to S per the STC, replacing those bits of the low-order half of the dividend that have already been used. Operations continue in the same manner; i.e., PQ bits are selected to form a PQ byte, this byte is transferred to S, and the used dividend bits are replaced until a complete quotient is developed.

Thirty-two reduction cycles are provided by the fixed-point divide microprogram. To obtain a valid quotient and remainder, the absolute value of the dividend and divisor must be so related that in 32 reduction cycles the dividend can be reduced to a remainder whose value is less than that of the divisor. If the highest-order significant bit of the dividend is less than 31 bit positions to the left of the highest-order significant bit of the divisor, the quotient is in true form and a valid result is obtained. If the highest-order significant bit of the dividend is more than 30 bit positions to the left of the highest-order significant bit of the divisor, the quotient is in 2's complement form and an invalid result is obtained unless the quotient is the maximum negative number (10000 . . . . . 000). Valid results are stored, but invalid results cause a program fixed-point divide interruption and the original operands remain unchanged in storage.

When E(14,15) = 11 and the STC = 11 (when tested), E(12–15) is incremented to 0000, the next-to-last PQ bit

Table 3-3. Divide Multiple Values, Fixed-Point

Carry from PAL(28)		T(32)		Divide Multiple Micro-Order	
Yes	No	1	0	'DVDL0'	'DVDL1'
X			X	2's complement of T.	2's complement of T shifted left 1.
X		X		T.	T shifted left 1.
	X	X		2's complement of T.	2's complement of T shifted left 1.
	X		X	T.	T shifted left 1.

is stored into F(6), and the last reduction cycle is taken (Sheet 3, Diagram 5-110). The last PQ bit is taken from the remainder in AB [A(28)] and is gated to SAL(7) [per E(14,15) = 00 and the 'DVDL0' micro-order]. This action completes the last byte in SAL, which is gated to S per the STC.

A branch on divisor sign (STAT G) and quotient form (true or 2's complement, STAT C,) causes the microprogram to enter 1 of 4 termination routines. The purpose of the termination routines is to test for valid results, to establish the quotient and remainder in the proper form according to the proper sign, to store the corrected remainder and quotient, and to end the operation.

Sheet 6 of Diagram 5-110 is a flowchart of the fixed-point divide termination routine if the quotient is in true form. In this case, the quotient and remainder are valid, and it is only necessary to store the results in the proper form. (Positive results must be stored in true form and negative results must be stored in 2's complement form, according to the convention of fixed-point arithmetic and the rules of algebra.) Because the dividend was stored in B and S in true form regardless of sign, the results may or may not be in the correct form. It is therefore necessary to branch on the original signs of the operands and on the form of the results to achieve the proper form for the results. If the remainder is in 2's complement form, it is the result of an overdraw, and must be corrected by adding the divisor in true form before storing or complementing the remainder according to the sign of the dividend.

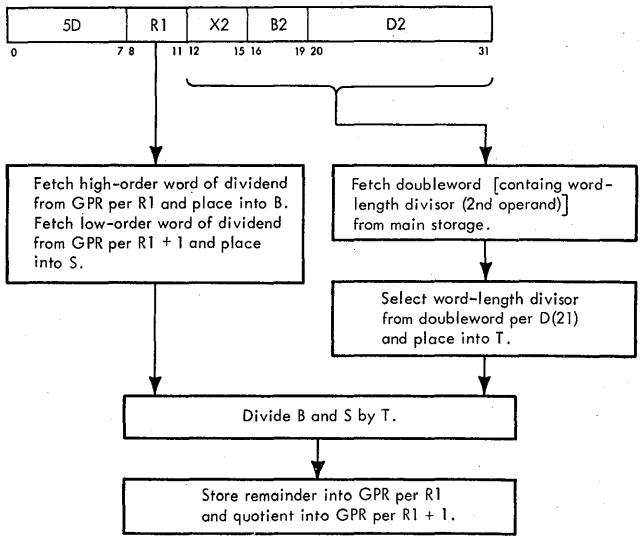
Sheet 5 of Diagram 5-110 is a flowchart of the fixed-point divide termination routine if the quotient is in 2's complement form. If the quotient is in 2's complement form and it is the maximum negative number (10000 . . . . . 000), the results are valid and the termination routine must accomplish the same tasks as for the true form termination. If the quotient is not the maximum negative number, a program fixed-point divide interruption is taken and the original operands remain unchanged in storage.

When the quotient and remainder are valid and have been converted to the proper form according to their algebraic sign, the remainder is stored into the GPR per E(8–11) and the quotient is stored into the GPR per E(8–11) + 1, replacing the high-order and low-order halves of the dividend respectively, and leaving the divisor unchanged in storage. An end-op cycle is taken, finishing the operation.

Two examples of the fixed-point divide operation are presented in Figures 3-8 and 3-9. These examples are the inverse of the two fixed-point multiply examples (Figures 3-6 and 3-7).

Divide, D (5D)

- Divide 1st operand (in GPR per R1 and R1 + 1) by 2nd operand (in storage) and place result into 1st operand location (remainder in GPR per R1; quotient in GPR per R1 + 1).
- RX format:



- Conditions at start of execution:
  First 16 bits of instruction are in E.
  High-order half of dividend (1st operand) is in S and T.
  Low-order half of dividend, is in GPR per R1 + 1.
  2nd operand address is in D.
  Main storage request for 2nd operand has been issued per D.
- D(21) determines which word of doubleword contains divisor: if a 1, right word; if a 0, left word.

The Divide, D, instruction divides the contents of B (high-order bits of dividend) and S (low-order bits of dividend) by the contents of T (divisor). D(21) determines which word of the doubleword fetched from main storage contains the divisor: if a 1, right word; if a 0, left word. Once the divisor is obtained from main storage, the operation is identical to the operation of the DR instruction (Diagram 5-110).

CONVERT

There are two fixed-point convert instructions, both in the RX format: Convert to Binary and Convert to Decimal. These instructions convert the radix of an operand from decimal to binary and binary to decimal,

the second operand is tested to see whether the proper integral boundary has been specified. If the address is located on an incorrect (non-doubleword) integral boundary, a program specification interruption occurs. The CVB operation is suppressed, and the data in LS and in main storage remains unchanged. If no program specification interruption occurs, the operation continues.

Recall that, at the start of execution, a storage request for the second operand had been issued. At this time, the data (doubleword operand) is present at the SDBO and is gated into ST, destroying the first operand. The STC, which selects the correct byte to be converted from ST, is set to 000, thus selecting S(0–7).

The contents of T (low-order half of the doubleword operand) are transferred to the LSWR. (This data is converted at a later time.) Because the converted data is to be stored into T and D, they are now cleared. The first byte (bits 0–7) of S is transferred to the serial adder B bus. Bits 0–3 are transferred to SAL(0–3) and SAL(4–7) and on to F. F now contains the first decimal digit to be converted.

As the decimal data from S is passed through the serial adder to F, it is tested for invalid digits. If the digits are invalid, STAT E is set and later, when tested, causes a program data interruption, which terminates the operation. If the digits are valid, the first decimal digit is transferred from F(4–7) to PAB(60–63). The contents of D and T are then transferred and shifted left 1 to PAA(7–30) and PAA(31–62), respectively, and added to the decimal digit. At this time, D and T contain zero.

The result of the addition, which is the decimal digit, is transferred from PAL(8–63) to DT and from PAL(32–63) to B(32–63). The contents of DT are then transferred to PAA(8–63). The contents of B are now shifted left 2, placed into PAB(4–65), and added to PAA. This action, in effect, multiplies the original decimal digit by 5. The result of the addition is then transferred from PAL(8–63) to DT. A byte from S is transferred to the serial adder B bus per the STC. At this time, the byte transferred to the serial adder is the first byte in S (STC = 000). This action allows the second decimal digit to be placed into the serial adder. From the serial adder, the data is sent to F. The STC is increased by 1 so that the next byte from S can be transferred when selected.

The second decimal digit, F(4–7), is transferred to PAB(60–63). The contents of DT are then transferred to PAA(7–62). This transfer shifts the converted data left 1, in effect multiplying the original decimal digit by 10 (x5 and x2). PAA and PAB are added, and the result in PAL(8–63) is transferred to DT. The contents of PAL(32–63) are transferred to B(32–63).

The next byte in S (bits 8–15) is now transferred to F via SAL. STAT D is then tested. If STAT D is set, it indicates the low-order word of the doubleword operand is being converted; if reset, the high-order word is being converted. At this time, STAT D is reset. The STC is now tested to see which byte of S is being worked on; the value presently in the STC is 001. Because STAT D is reset and the STC does not equal 011, operations continue in the same manner as previously described; i.e., a decimal digit is brought in and added to the sum of the converted digits, and the result is multiplied by 10. This procedure continues until all digits have been converted.

While the last two decimal digits of the high-order word are being processed, STAT D is set to indicate the low-order word. When the last decimal digit of the high-order word has been transferred to F, the STC is set to 000; the low-order word is transferred from the LSWR (where it was stored at the beginning of the operation) to S and is converted in the same manner as the high-order word.

When the low-order byte of the low-order word is transferred from SAL to F, the sign is tested for validity, setting STAT E if invalid. The state of STAT E is then tested. If STAT E is set, a data-check condition exists and an end-op cycle is taken, leaving the contents of LS unaltered. If STAT E is not set, the sign of the number is determined by [F(4–7)], and the last decimal digit is converted. If F(4–7) is a plus sign, the converted data is transferred from T to the GPR per E(8–11). If F(4–7) is a minus sign, the converted data is first 2's complemented and then transferred from T to the GPR per E(8–11). STAT G is then set if T(32) = 1. PAL(32–63) is tested for all zero's (T = 0), and a branch is made on the result of this test.

The contents of D (overflow bits) are then transferred to PAL(40–63), and PAL(32–63) is again tested for all zero's (D = 0). Note that D is not 2's complemented when T is 2's complemented for a negative sign, and should always equal zero unless an overflow occurred.

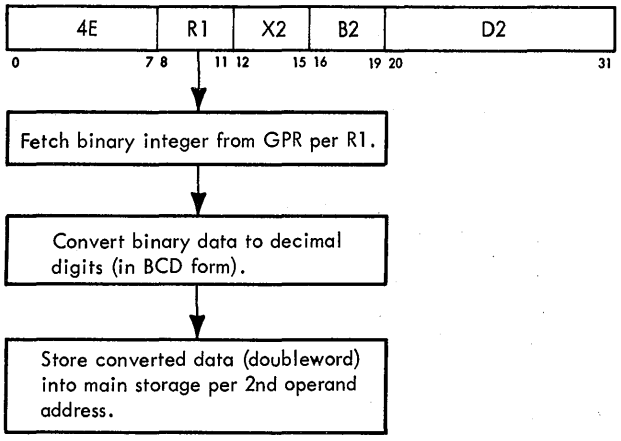
If T(32–63) = 0 and D(0–23) = 0, the result is zero and a normal end-op cycle is taken. In all cases, if D(0–23) does not equal 0, an overflow has occurred, a fixed-point divide check condition exists and an end-op cycle is taken. If D(0–23) = 0 and T(32–63) does not equal 0, a further test is made to determine if the maximum positive or negative number has been exceeded. If the decimal sign [F(4–7)] was positive and T(32) = 1 (STAT G set), the maximum positive number has been exceeded. If the decimal sign [F(4–7)] was negative and T(32) = 0 (STAT G reset), the maximum negative number has been exceeded. (For a negative sign, the contents of T have previously been changed to 2's complement form.) In both of the above cases, a fixed-point divide check condition exists and an end-op cycle is taken. If the maximum number has not been exceeded, an end-op cycle is taken, completing the operation.

Note that even if an overflow condition is detected or if the maximum positive or negative number has been exceeded, the low-order 32 bits of the converted integer are stored into the GPR per E(8–11) and the only indication is

the fixed-point divide check condition. If any decimal digit or the sign is invalid, a data check condition exists and the operation is terminated without storing any data.

Convert to Decimal, CVD (4E)

- Convert radix of 1st operand (in GPR per R1) from binary to decimal and place result into 2nd operand location (in storage).
- RX format:



- Conditions at start of execution:
  First 16 bits of instruction are in E.
  1st operand is in S and T.
  2nd operand address is in D.
- Operand to be converted is 32-bit signed binary integer.
- Converted data is in packed decimal format.
- Positive sign is encoded as 1100 or 1010.
- Minus sign is encoded as 1101 or 1011.

The Convert to Decimal (CVD) instruction converts the radix of the first operand (from GPR per R1) from binary to decimal, storing the result into the second operand location (in main storage). The number to be converted is a 32-bit signed binary integer; the 31 binary bits yield 15 decimal digits in the packed format. The sign bit may be encoded in two forms for both positive and negative numbers. A positive sign may be 1100 or 1010; a minus sign may be 1101 or 1011. The choice between the two sign representations is determined by PSW(12).

A binary number is converted to decimal by parallel decimal correct function, which operates from AB to the parallel adder. The binary number being converted is extracted one bit at a time from the most significant bit to the least significant bit (left to right) and is added to a previous partially converted decimal number. The resultant number is first multiplied by 2, by shifting left 1, and then decimal-corrected. The parallel decimal-correct function extracts 6 or 0 from the partially converted number and adds the 6 or 0 to twice the partially converted number so that the resulting decimal number does not exceed the maximum decimal number of 9. This function is performed for each half-byte of AB(28–63) for every add cycle. Table 3-4 lists the AB bits and parallel adder bits used in the decimal-correct function. The process of conversion is repeated until all bits of the binary number have been examined. At the completion of the conversion, the converted number is shifted left 4, and the correct sign is placed into the low-order bit positions. The example shown in Figure 3-10 illustrates the method of converting from binary to decimal.

Table 3-4. Conversion to Decimal (Excess-6)

AB Bits Set	Set PAB Bus Bits (+6)
A(28)	PAB(29,30)
A(29,30)	PAB(29,30)
A(29) and A(31)	PAB(29,30)
B(32)	PAB(33,34)
B(33,34)	PAB(33,34)
B(33) and B(35)	PAB(33,34)
B(36)	PAB(37,38)
B(37,38)	PAB(37,38)
B(37) and B(39)	PAB(37,38)
B(40)	PAB(41,42)
B(41,42)	PAB(41,42)
B(41) and B(43)	PAB(41,42)
B(44)	PAB(45,46)
B(45,46)	PAB(45,46)
B(45) and B(47)	PAB(45,46)
B(48)	PAB(49,50)
B(49,50)	PAB(49,50)
B(49) and B(51)	PAB(49,50)
B(52)	PAB(53,54)
B(53,54)	PAB(53,54)
B(53) and B(55)	PAB(53,54)
B(56)	PAB(57,58)
B(57,58)	PAB(57,58)
B(57) and B(59)	PAB(57,58)
B(60)	PAB(61,62)
B(61,62)	PAB(61,62)
B(61) and B(63)	PAB(61,62)

Diagram 5-112, FEMDM, is a flowchart of the CVD instruction. At the start of execution, the first 16 bits of the instruction are in E, the first operand is in S and T, and the second operand address is in D. The first portion of the operation is devoted to testing for specification-check and address-store-compare conditions. If a specification check is present, a program specification interruption occurs and the operation is suppressed. If an address-store-compare condition occurs, the 'PSC' trigger is set and the operation continues. The value of S(0) (sign of original binary number) is set into STAT C. At the end of the operation, STAT C is examined to determine the sign of the converted number.

Convert +146 Binary (92 Hex) to +146 BCD

1. SAL = 1001 0010 (shift left 1 after each addition).
2. Gate 6 or 0 to each half-byte of PAB per AB bits (see Table 3-4).

AB												Remarks
52	53	54	55	56	57	58	59	60	61	62	63	
0	0	0	0	0	0	0	0	0	0	0	0	Gate 0 to each half-byte of PAB.
											1	Hot carry to PAA(63) [SAL(0) = 1].
0	0	0	0	0	0	0	0	0	0	0	1	Sum to AB and DT.
0	0	0	0	0	0	0	0	0	0	1	0	Gate DT left 1 to PAA.
0	0	0	0	0	0	0	0	0	0	0	0	Gate 0 to each half-byte of PAB.
0	0	0	0	0	0	0	0	0	0	1	0	Sum to AB and DT.
0	0	0	0	0	0	0	0	0	1	0	0	Gate DT left 1 to PAA.
0	0	0	0	0	0	0	0	0	0	0	0	Gate 0 to each half-byte of PAB.
0	0	0	0	0	0	0	0	0	1	0	0	Sum to AB and DT.
0	0	0	0	0	0	0	0	1	0	0	0	Gate DT left 1 to PAA.
0	0	0	0	0	0	0	0	0	0	0	0	Gate 0 to each half-byte of PAB.
											1	Hot carry to PAA(63) [SAL(0) = 1].
0	0	0	0	0	0	0	0	1	0	0	1	Sum to AB and DT.
0	0	0	0	0	0	0	1	0	0	1	0	Gate DT left 1 to PAA.
0	0	0	0	0	0	0	0	0	1	1	0	Gate 0 or 6 to each half-byte of PAB.
0	0	0	0	0	0	0	1	1	0	0	0	Sum to AB and DT.
0	0	0	0	0	0	1	1	0	0	0	0	Gate DT left 1 to PAA.
0	0	0	0	0	0	0	0	0	1	1	0	Gate 0 or 6 to each half-byte of PAB.
0	0	0	0	0	0	1	1	0	1	1	0	Sum to AB and DT.
0	0	0	0	0	1	1	0	1	1	0	0	Gate DT left 1 to PAA.
0	0	0	0	0	0	0	0	0	1	1	0	Gate 0 or 6 to each half-byte of PAB.
											1	Hot carry to PAA(63) [SAL(0) = 1].
0	0	0	0	0	1	1	1	0	0	1	1	Sum to AB and DT.
0	0	0	0	1	1	1	0	0	1	1	0	Gate DT left 1 to PAA.
0	0	0	0	0	1	1	0	0	0	0	0	Gate 0 or 6 to each half-byte of PAB.
0	0	0	1	0	1	0	0	0	1	1	0	Result [place sign of 1100 into B(64-67) per STAT C].

0001 0100 0110 1100 = +146 in BCD packed format

Figure 3-10. Convert to Decimal Example

Because conversion is done on a positive operand basis, the sign of the operand is determined by testing T(32). If T(32) is a 1, the data to be converted is negative and its 2's complement form must be derived; if a 0, the data is positive. The contents of D are then shifted left 4 and transferred to the LSWR. Because the high-order converted data is stored in D, D is cleared. The first byte of data is now sent from S to SAL per the STC (STC = 0).

The first decimal convert value (0 or 6) is obtained by examining the contents of AB. Because AB is cleared at this time, the first decimal convert value is all 0's. (See Table 3-4 for conversion values.) The decimal convert value is placed into PAB(28–63), and a hot carry is generated if SAL(0) = 1. If SAL(0) = 0, a hot carry is not generated and the value in PAB is transferred directly to PAL. The contents in SAL are shifted left 1, thus placing the next bit from the byte into SAL(0). Next, PAL(8–63) is transferred to AB(8–63), PAL(8–31) is transferred to D(0–23), and PAL(32–63) is transferred to T(32–63). This action places the first converted bit into AB and DT. The value in DT is then shifted left 1 to PAA(7–63). A decimal convert value is again obtained from B and placed into PAB. SAL(0) is now checked. If it is a 1, a hot carry is generated; if a 0, no carry is generated. In either case, the numbers are added. The result of the addition is then transferred to DT and AB. The contents of SAL are shifted left 1, bringing in the next bit for conversion. The ABC is increased by 1.

The contents of SAL (a byte of S per the STC) are shifted left one digit position as follows: F is gated to SAA, the byte of S (per the STC) is gated to SAB, and SAL is gated to F and S (per the STC). This operation is equivalent to adding the contents of SAL to itself, which doubles the value of SAL and, in effect, shifts the bits one digit position to the left. After seven add cycles, every bit in SAL has been presented in sequence to PAA(63) via SAL(0), the original low-order bit of F (and of the byte of S) has been shifted to the high-order bit position of F and of the byte of S, and the STC has been incremented by 1 to present the next byte of S to SAL and F. To assure that the new byte of S presented to SAB will be added to all zeros, F(4–7) is gated to SAA(0–3) and all zeros are gated to SAA(4–7).

The ABC indicates the progress of the microprogram in converting bits of the byte from S (now located in SAL, F, and S). When the ABC = 4, five bits of the byte have been converted, the microprogram has branched (per ABC = 3) to a routine to convert the last three bits of the byte, the ABC is set to 0, and the STC is incremented by 1 to present the next byte of S to SAL and F. One conversion cycle is taken and the microprogram re-enters the conversion loop, converting the next byte of S in the same manner as the first. This routine continues until, at the time the microprogram branches from the conversion loop, the STC = 3. The microprogram then enters a termination routine, during which the last three bits of the low-order byte of S are converted and the sign is set per STAT C. (Recall that STAT C was set to the value of the sign of the binary number.) If STAT C is set, the last decimal convert value (converting bit 7 of the last byte) is obtained and the hot carry is generated according to the value of SAL(0); the result is placed into AB and DT. A minus sign (1101 or 1011) is then placed into B(64–67). AB is shifted left 4 and transferred to ST(0–63). Mark triggers 0 through 7 are set, and the converted data is transferred to main storage. An end-op cycle is taken, completing the operation. If STAT C is reset, a positive sign is placed into B(64–67), shifted into the low-order bits of the converted operand, and stored into main storage.

The CVD instruction is the only instruction which uses the excess-6 gates from B to PAB. (For a discussion of the excess-6 gates, refer to "Parallel Adder" in Chapter 2.) Note in the example in Figure 3-10 that the result of each addition is gated from PAL to AB and to DT; then the contents of DT are gated left 1 to PAA (in effect, doubled in value). By examining the bit configuration of each half-byte of AB, a decision is made whether to add 6 to the corresponding half-byte of DT after being shifted left 1 to PAA. This decision is a prediction whether or not each half-byte of AB, if doubled in value, will exceed 9. Each half-byte of AB which has a value of 5 or greater will, if doubled, exceed 9. Therefore, for decimal correction purposes, 6 must be added to the corresponding half-bytes of DT.

For simplicity, the low-order half-byte of B is taken as an example in Table 3-5; however, each half-byte of AB (28–63) is simultaneously examined. Note in the table that if B(60–63) contains a value of 0 through 4, the value, when doubled, will *not* exceed 9 and a 6 is *not* added. If B(60–63) contains a value of 5 through 15, however, its value, when doubled, *will* exceed 9 and a 6 *must* be added. Note that for values 5 through 15 B(60–63) contains one or more of the following bit combinations: B(60) = 1, B(61,62) = 11, B(61) and B(63) = 1.

STORE

There are three fixed-point store instructions: Store, ST, RX format; Store Halfword, STH, RX format; and Store Multiple, STM, RS format. The function of the store instructions is to store the contents of specified GPR(s) into main storage.

Store, ST (50)

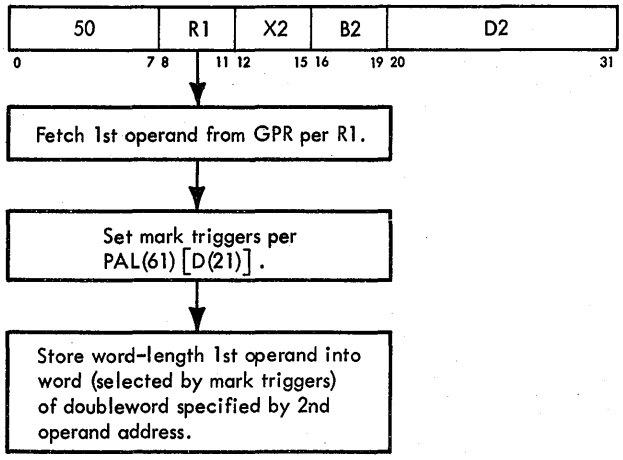
- Store 1st operand (in GPR per R1) into 2nd operand location (in storage).

Table 3-5. Excess-6 Conversion, B(60–63)

Decision Making Factors (ALD RB753)	B-bits 60	61	62	63	Decimal Value	Decimal Value if Doubled	Add to PAB(60–63)
None of the below conditions	0	0	0	0	0	0	↑
	0	0	0	1	1	2	
	0	0	1	0	2	4	0
	0	0	1	1	3	6	
	0	1	0	0	4	8	↓
B(61) and B(63) = 1	0	1	0	1	5	10	↑
B(61,62) = 11	0	1	1	0	6	12	
	0	1	1	1	7	14	
B(60) = 1	1	0	0	0	8	16	
	1	0	0	1	9	18	6
	1	0	1	0	10	20	[1 to
	1	0	1	1	11	22	PAB(61,62)]
	1	1	0	0	12	24	
	1	1	0	1	13	26	
	1	1	1	0	14	28	
	1	1	1	1	15	30	↓

Refer to Table 3-4 for remaining half-bytes of AB.

- RX format:

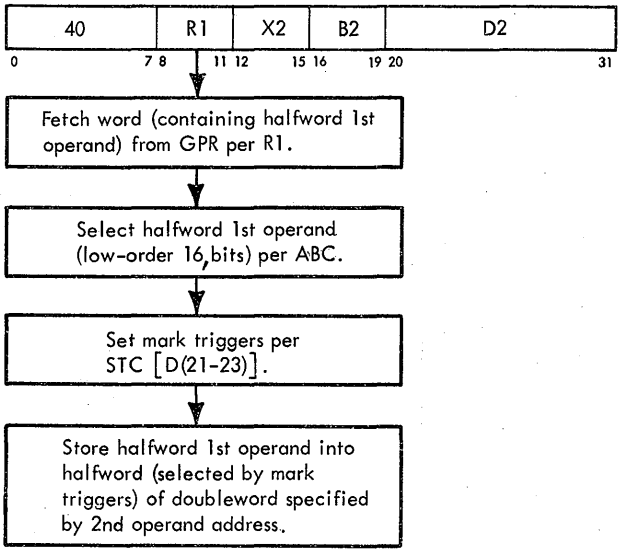


- Conditions at start of execution:
  First 16 bits of instruction are in E.
  1st operand is in S and T.
  2nd operand address is in D.
- PAL(61) determines into which word of doubleword 1st operand is to be stored: if a 1, right word; if a 0, left word.

The Store, ST, instruction stores the first operand (from the GPR per R1) into the main storage address specified by the second operand address. Diagram 5-113, FEMDM, is a flowchart of the ST instruction. PAL(61) determines which word (right or left) of the doubleword addressed by D will receive the first operand; mark triggers 0–3 or 4–7 are set accordingly.

Store Halfword, STH (40)

- Store halfword 1st operand (in GPR per R1) into 2nd operand location (in storage).
- RX format:



- Conditions at start of execution:
  First 16 bits of instruction are in E.
  1st operand is in S and T.
  2nd operand address is in D.
- ABC selects 16 low-order bits of 1st operand for storage; high-order bits are ignored.
- STC [D(21–23)] positions 16 low-order bytes for storage.

The Store Halfword (STH) instruction stores the 16 low-order bits of the GPR specified by R1 into the main storage location specified by the second operand address. The high-order bits are not used.

The selected halfword is stored through the use of mark triggers, which reflect the value of D(21–23). This value, plus 1, signifies into which portion of the doubleword the halfword is to be stored. Diagram 5-114, FEMDM, is a flowchart of the STH instruction. At the start of execution, the first 16 bits of the instruction are in E, the first operand is in S and T, and the second operand address (into which the halfword operand is to be stored) is in D.

The second operand address is first tested to see that it is on an integral boundary; if not, a program specification interruption occurs and the operation is suppressed. Assuming no specification error, D(21–23) is transferred to the STC to select the correct portion of the main storage doubleword into which the 16 low-order bits of the first operand are to be stored. Next, the first operand (located in T) is transferred to B. The ABC is then set to 6 by placing all 1's into the ABC and subtracting 1 from this value. (The ABC selects the two low-order bytes of the operand presently located in B.) The mark trigger, which transfers the selected high-order byte of the halfword from ST to the main storage location, is selected per the STC. (See Table 3-6 for the STC and mark trigger settings and the corresponding operand bits transferred.) The eight high-order bits of the halfword are now transferred from B(48–55) to the correct position via the serial adder and the STC. A 3-cycle storage request is given. (Three cycles later, the data in ST is stored into main storage.) Also at this time, the ABC and STC are increased by 1 to select the next byte of data and to position the byte into ST by the time the 3-cycle storage request has elapsed.

To determine whether this store operation modified the instruction to be executed next, an address-store-compare test is made by comparing the IC with the main storage address used in the store operation. The test is made by transferring the 2's complement of D (address of main storage doubleword into which the halfword operand is to be stored) to PAA(40–63). The contents of the IC are transferred to PAB(40–63); PAA and PAB are added, and the result is shifted right 4 in PAL. The PAL is tested for 0; if 0, an address-store-compare condition exists and the 'PSC' trigger is set. This trigger is tested

Table 3-6. Operand Bits Transferred, STH Instruction

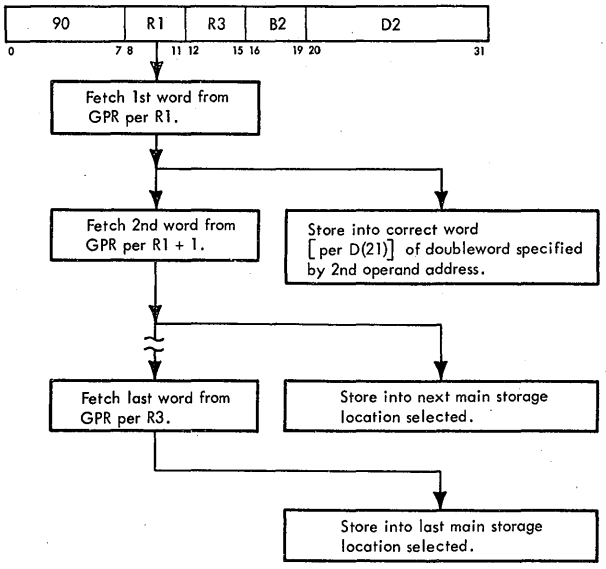
STC			Mark Trigger	Operand Bits Transferred
D(21)	D(22)	D(23)		
0	0	0	0	0–7
0	1	0	2	16–23
1	0	0	4	32–39
1	1	0	6	48–55
STC +1				
0	0	1	1	8–15
0	1	1	3	24–31
1	0	1	5	40–47
1	1	1	7	56–63

during the I-Fetch sequence of the next instruction and, if set, causes the modified instruction to be fetched from main storage and reloaded in Q. (Refer to "ASC Test" in Section 1 of this Chapter for an explanation of the address-store-compare test during I-Fetch.)

At this point, the eight low-order bits of the halfword operand are transferred from B(56–63) to ST via the SAL and STC. The associated mark triggers are also set at this time per the STC. An end-op cycle is taken to complete the operation. If a protection check condition occurs, the operation is suppressed, because no data storage takes place. Instead, the next instruction is fetched and executed, followed by a program interruption due to the "late" protection check.

Store Multiple, STM (90)

- Store into 2nd operand location (as many words as required, in storage) contents of GPR's, in ascending order, starting with 1st operand location (per R1) and ending with 3rd operand location (per R3).
- RS format: (See adjoining column.)
- Conditions at start of execution:
  First 16 bits of instruction are in E.
  1st operand is in S and T.
  2nd operand address is in D.
- Number of words to be stored is determined by E(8–11) and E(12–15).
- Addressed GPR's wrap around from 15 to 0.
- D(21) determines into which word of doubleword the first word is to be stored: if a 1, right word; if a 0, left word.



The Store Multiple (STM) instruction stores one or more 32-bit words from LS, starting with the GPR specified by R1 and ending with the GPR specified by R3, into main storage. The area in main storage where the contents of the GPR's are placed starts at the location designated by the second operand address and continues through as many words as needed in an ascending order.

The number of words to be stored is determined by E(8–11) and E(12–15). If the contents of these bit locations are equal, only one word is to be stored. If the contents of the bit locations are not equal, storage of words is continued; E(8–11) is updated by 1 for each word stored until the bit locations are equal; then, one more word is stored, and the operation is completed. Once it has been decided whether one or more words are to be stored, it must be determined into which word of the doubleword, in main storage, the first word is to be stored; D(21) serves this function. If D(21) = 0, the first word is to be placed into the left word of the doubleword; if D(21) = 1, into the right word.

See Diagram 5-115, FEMDM, a flowchart of the STM instruction. At the start of execution, the first 16 bits of the instruction are in E, the first operand is in ST, and the second operand address is in D. The instruction first tests for a specification-check condition. If one exists, a program specification interruption occurs and the operation is suppressed. Assuming there is no specification check, the contents of D are transferred to PAA(40–63). A 3-cycle storage request is given. To determine whether one or more words are to be stored, E(8–11), R1, is compared with E(12–15), R3. If E(8–11) equals E(12–15), only one word is to be stored; if it does not, more than one word is to be stored.

Assume one word is to be stored [E(8–11) equals E(12–15)]. D(21) is tested to determine whether the LS operand is to be stored into the left or right word of the

main storage doubleword. If D(21) = 0, the left word is selected; if D(21) = 1, the right word is selected. Assume D(21) = 1. Mark triggers 4 through 7 are set to gate T(32–63) to bits 32–63 of the main storage doubleword when data is stored. An address-store-compare test is then made. This test involves transferring the 2's complement of D to PAA(40–63) and 7 to PAA(61–63), and transferring the contents of the IC to PAB(40–63); they are added and shifted right 4 to PAL. The PAL is then tested for zero. If zero, the 'PSC' trigger is set. This trigger is tested during end op and, if set, indicates that the next instruction to be executed has been modified. The modified instruction must then be refetched into Q during I-Fetch. (For information about the address-store-compare test, refer to "ASC Test" in Section 1 of this chapter.)

A protection test is also made by main storage while the next instruction is being fetched. The protection key in the PSW is compared with the storage key for the location. If the keys agree, storage is permitted. If the keys do not agree, storage is not permitted, the instruction is terminated, and a "late" protection interruption is taken after the execution of the next instruction.

Now assume D(21) = 0 and E(8–11) equals E(12–15). Again only one word is to be stored into main storage. In this case, however, the word is to be stored in the left word of the doubleword location in main storage. Accordingly, mark triggers 0 through 3 are set to gate S(0–31) to bits 0–31 in main storage.

Now assume E(8–11) is not equal to E(12–15). If D(21) = 1, the first word is to be stored into the right word of the doubleword (mark triggers 4 through 7 are set). Because more than one word is to be loaded, the next sequentially addressed word from LS is transferred to S. A storage request is then given. The contents of D are increased by 8. E(8–11) and E(12–15) are again compared. If equal, mark triggers 0 through 3 are set and the data is gated into main storage. If they are not equal, more than two words are to be stored into main storage. Because the second operand to be gated from LS is already in S, and because D contains the address where both the second and third operands are to be stored, the third operand is now transferred out of LS and mark triggers 0 through 7 are set. The data is stored into main storage. Address-store-compare and protection-key tests are made for each new main storage address. Only the last storage request issued can cause a "late" protection interruption.

If more than two words are to be stored, the operation continues as just described. E(8–11) is incremented, a word is loaded into ST, the address in D is increased by 8, and storage requests are generated when needed. When E(8–11) and E(12–15) are equal, the last word is loaded into main storage and an end-op cycle is taken.

If D(21) = 0, and if E(8–11) and E(12–15) are not equal, the first word is to be placed into the left word of the doubleword location in main storage and the second word into the right word.

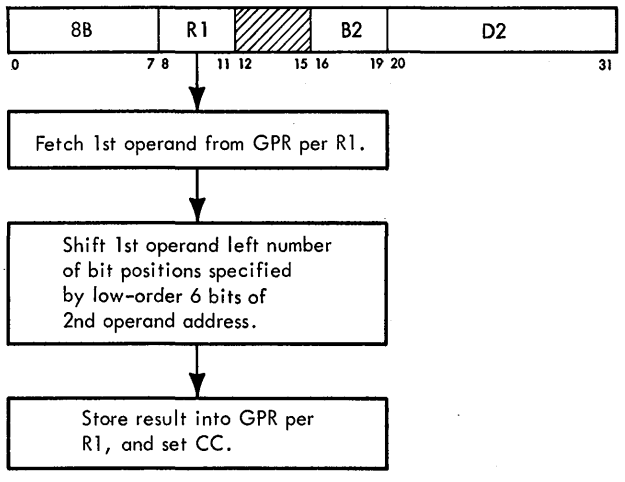
SHIFT

There are four fixed-point shift instructions: Shift Left Single, SLA; Shift Left Double, SLDA; Shift Right Single, SRA; and Shift Right Double, SRDA. Their function is to shift the first operand (right or left) and to store the result into the first operand address. The first operand may be a word or a doubleword in length and is shifted the amount specified by the low-order six bits of the second operand address. The specified amount of shift is accomplished in increments of left 1, left 2, left 4, and right 4 shifts. To expedite the operation, combinations of these increments are used whenever possible, and a maximum number of left 4 and right 4 shifts are used.

The CC is set to indicate the status of the result. A left-shifted result is tested for an overflow condition to determine if significant high-order digits were lost. During a right-shift, significant low-order digits may be lost with no indication.

Shift Left Single, SLA (8B)

- Shift 1st operand (in GPR per R1) left number of bit positions specified by low-order 6 bits of 2nd operand address and place result into 1st operand location.
- RS format:



- Conditions at start of execution:
  First 16 bits of instruction are in E.
  1st operand is in S and T.
  Amount of shift is in D and PAL.
- D(18–23) indicates total amount of shift.

- Methods of shifting:
  Left 1 from T to PAA.
  Left 2 from AB to PAB.
  Left 4 from PAA or PAB to PAL.
- E(12–15) indicates number of left 4 shifts to be performed.
- E(12–15) is reduced by 1 after each multiple of four shifts occurs.
- 0's are supplied to vacated bit positions.
- Overflow occurs if data is shifted out of bit position 1.
- CC setting:
  Result in PAL is zero: CC = 0.
  Result in PAL is less than zero: CC = 1.
  Result in PAL is greater than zero: CC = 2.
  Overflow: CC = 3.

The Shift Left Single, SLA, instruction shifts the first operand left the number of bit positions specified by the low-order six bits of the second operand address. (The remainder of the second operand address is ignored.) The second operand request per D, normally initiated during RS I-Fetch, is blocked during I-Fetch of a shift instruction because D does not contain a main storage address. Refer to "Basic RS and SI I-Fetch" in Section 1 of this chapter.

The sign of the first operand remains unchanged. All 31 bits of the operand participate in the left shift. Zeros are transferred into the vacated low-order register positions. If a bit unlike the sign bit is shifted out of position 1, an overflow occurs, causing a program fixed-point overflow interruption during end op if the fixed-point overflow mask bit is a 1.

Left-shifting can be accomplished by three methods: (1) shifting left 1 bit position from T to the parallel adder, (2) shifting left 2 bit positions from AB to the parallel adder, and (3) shifting left 4 bit positions from the parallel adder to PAL. In the interest of speed, the desired amount of left shift is accomplished using a maximum number of left 4 shifts and a minimum number of left 1 and left 2 shifts (not more than one of each). Also, whenever possible, a left 1 or a left 2 shift is combined with a left 4 shift because these combinations can be accomplished in one pass through the parallel adder. Table 3-7 shows how left-shifting is accomplished for any amount of shift desired.

See Diagram 5-116, FEMDM, a flowchart of the SLA instruction. At the start of execution, the first 16 bits of the instruction are in E, the first operand is in S and T, and the number of bit positions to be shifted is in D and PAL.

In the SLA instruction, PAL(58–63) is first tested to determine the amount of shift. (PAL contains the same value as D.) If PAL(58–61) = 0, a shift of less than 4 is needed; PAL(62) is then tested. If PAL(62) = 1, a shift of either 3 or 2 is to be performed; if PAL(62) = 0, either a shift of 1 or no shift is to be performed.

As an example, assume that PAL(58–61) does not equal 0 and that PAL(62,63) = 11. A shift of left 7, left 11, or more is indicated. STAT C is set to S(0), the sign of the

Table 3-7. Left Shift Combinations

PAL(58–61)	PAL(62,63)	Total Shift Desired	Incremental Shifting Sequence		
0000	00	None	None		
	01	Left 1	Left 1		
	10	Left 2	Left 2		
	11	Left 3	Left 1	Left 2	
Not 0000	00	Left 4, 8, 12 and so on	Left 4	Left 4 until E(12–15) = 1	
	01	Left 5, 9, 13 and so on	Left 1 and left 4	Left 4 until E(12–15) = 1	
	10	Left 6, 10, 14 and so on	Left 2 and left 4	Left 4 until E(12–15) = 1	
	11	Left 7, 11, 15 and so on	Left 1 and left 4	Left 2 if E(12–15) = 1	
				Left 2 and left 4 if E(12–15) ≠ 1	Left 4 until E(12–15) = 1

first operand that will be transferred to LS at the end of the operation. D(18–21), which is equal to PAL(58–61), is transferred to E(12–15). This value is reduced sequentially by 1 after every left 4 shift until it equals 0001, at which time no more shifting of data is necessary. The contents of T are transferred to PAA(31–62), thus achieving a left 1 shift. T(32) is propagated into PAL(26–31). Because a left 4 shift takes place only when transferring data from PAA or PAB into PAL, PAA(8–67) is shifted left 4 positions into PAL(4–63). PAL(26–32) is now tested to see that no important data has been lost during the shifting. If these bits are not all 0's or all 1's, STAT B is set, indicating overflow occured; if found set during end op, a program fixed-point overflow interruption occurs if the associated overflow mask bit in the PSW is a 1. PAL(24–67) is transferred to AB(24–67). E(12–15) is tested to see whether it contains a value of 0001. If it does, only a left 7 shift is to be performed; otherwise, a shift of left 11 or more is to be made.

Assume E(12–15) = 0001. A 1 is subtracted from E(12–15). The contents of AB(6–67) are transferred to PAB(4–65). This transfer accomplishes a left 2 shift, making a total left shift of 7 positions. PAL(26–32) is tested for overflow, and PAL(32–63) is tested for all 0's. The data is transferred to LS, and an end-op cycle is taken to complete the operation.

Now assume E(12–15) does not equal 0001, indicating a total left shift of at least 11 positions or more is called for by the instruction. Because at this point the data has already been shifted left 5 positions, a left 6 shift is necessary to achieve the minimum left 11 shift. A 1 is subtracted from E(12–15). AB(6–67) is transferred to PAB(4–65), yielding a left 2 shift. A left 4 shift is achieved by shifting the contents of PAB(4–65) to PAL(0–61). PAL(26–32) is tested for overflow, and PAL(32–63) is tested for all 0's. PAL(32–63) is transferred to T(32–63), and T(32–63) is transferred to the GPR per E(8–11). The value of STAT C is placed into the sign position of the GPR. E(12–15) is again checked for a value of 0001. If the bits now contain this value, the data has been shifted the correct number of times. A 1 is subtracted from E(12–15), and an end-op cycle is taken, completing the operation. If E(12–15) does not equal 0001, a left shift of more than 11 is required for this instruction.

If E(12–15) still does not equal 0001, 1 is again subtracted from E(12–15). T(32–63) is transferred to PAA(32–63), and the data is shifted left 4 positions into PAL(28–63). After testing PAL(26–32) and PAL(32–63), the contents of PAL(32–63) are transferred to T, and the data in T(33–63) is transferred into the GPR per E(8–11). The value of STAT C is placed into the GPR sign position per E(8–11).

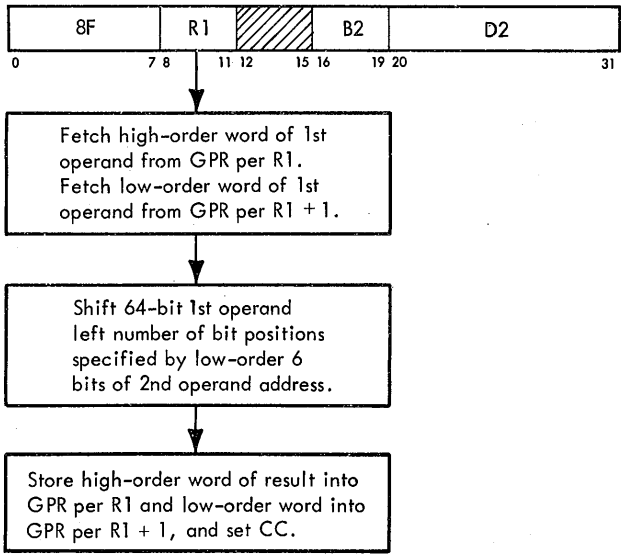
Note: The result of the first 11 shifts is transferred to LS before E(12–15) is tested to see whether a shift of more than 11 places is required. If a total shift greater than 11 is specified, the data is shifted left an additional four places and transferred to LS, where it destroys the 11-place shifted operand stored earlier. E(12–15) is tested; if a shift greater than 15 in specified, the data is again shifted left 4 and transferred to LS. This procedure of testing E(12–15), shifting left 4, and transferring the result to LS continues until E(12–15) equals 0001.

E(12–15) is again checked for 0001. If E(12–15) = 0001 at this time, 1 is subtracted from it and an end-op cycle is taken, completing the operation. If E(12–15) does not equal 0001, the operand continues to be shifted in multiples of 4 until E(12–15) equals 0001. The CC is then set per STAT A, STAT B, and the result sign [T(32)], and an end-op cycle is taken.

Left shifts of other amounts are performed in a similar manner. (Refer to Table 3-7 and Diagram 5-116, FEMDM.)

Shift Left Double, SLDA (8F)

- Shift 1st operand (in GPR per R1 and R1 + 1) left number of bit positions specified by low-order six bits of 2nd operand address and place result into 1st operand location.
- RS format:



- Conditions at start of execution:
  First 16 bits of instruction are in E.
  1st operand is in S and T.
  Amount of shift is in D and PAL.
- D(18–23) indicates total amount of shift.
- E(12–15) indicates number of left 4 shifts to be performed.
- E(12–15) is reduced by 1 after each multiple of four shifts occurs.

- High-order bits of low-order word of doubleword operand are saved and placed into high-order word operand.
- 0's are supplied to vacated bit positions.
- Overflow occurs if data is shifted out of bit position 1 of high-order word.
- CC setting:
  Result in PAL is zero: CC = 0.
  Result in PAL is less than zero: CC = 1.
  Result in PAL is greater than zero: CC = 2.
  Overflow: CC = 3.

The Shift Left Double, SLDA, instruction shifts the doubleword first operand left the number of bit positions specified by the low-order six bits of the second operand address. The R1 field of the instruction is the address of the GPR containing the high-order 32 bits of the doubleword operand. The low-order word of the doubleword operand is in the GPR per R1 + 1. The R1 field of the instruction must specify an even register. When R1 is odd, a program specification interruption occurs.

The sign of the doubleword operand is the sign of the even GPR. The high-order bit (sign) of the odd GPR is treated as an integer bit. As the information is shifted, 0's are supplied to the vacated low-order positions. If a bit unlike the sign bit is shifted out of bit position 1 of the high-order word of the doubleword operand, an overflow occurs. The overflow causes a program fixed-point overflow interruption if the associated mask bit in the PSW is a 1.

The SLDA instruction is similar to the SLA instruction in that a left 1 shift occurs when transferring data from T to PAA, a left 2 shift occurs when transferring data from AB to PAB, and a left 4 shift occurs when data is transferred from the inputs of the parallel adder to PAL. Also, the total shift specified is accomplished using a maximum number of left 4 shifts and a minimum number of left 1 and left 2 shifts. The differences are a result of handling a doubleword, one word at a time, in the SLDA instruction.

B(64–67), which contains the overflow bits from the low-order word, may be thought of as a four-bit register inserted between the high-order word and the low-order word. It is therefore necessary to shift these bits four bit positions to the left to position them correctly into the high-order word. This positioning is accomplished in different ways according to the shift being performed; the result, however, is always that of shifting the overflow bits left four bit positions.

Refer to Diagram 5-117, FEMDM, a flowchart of the SLDA instruction. At the start of execution, the first 16 bits of the instruction are in E, the high-order word of the first operand is in S and T, and the number of bit positions to be shifted is in D and PAL.

In the beginning of the operation, the sign of the operand (which will be transferred to LS at the end of the operation) is stored into STAT C. Because the low-order word of the doubleword operand is treated first, that word is now transferred from LS. (Recall that S and T presently contain the high-order word of the doubleword.) Once the low-order word of the doubleword is obtained, the data is shifted as for the SLA instruction. The high-order bits of the low-order word of the doubleword are not lost when shifted out but are transferred to B(64–67). When the high-order word of the doubleword is obtained, these bits are shifted into the high-order word of the operand.

The low-order word is shifted the amount specified by PAL(62,63), and the low-order 32 bits of the result are stored in the GPR specified by R1 + 1 (odd register). A zero test is performed, and STAT A is set if the low-order word is all zeros. The high-order bits that were shifted out of the low-order word are gated from PAL(28–31) to B(64–67) where they are stored, shifted left four positions and appended to the high-order word. The high-order word is transferred from the even GPR specified by R1, is shifted the same amount as the low-order word per PAL(62,63), and is added to the overflow bits from the low-order word. An overflow test is performed to determine if significant bits were lost from the high-order word, and STAT B is set if an overflow occurred. A zero test is performed, and STAT A is reset if the high-order result is not all zeros.

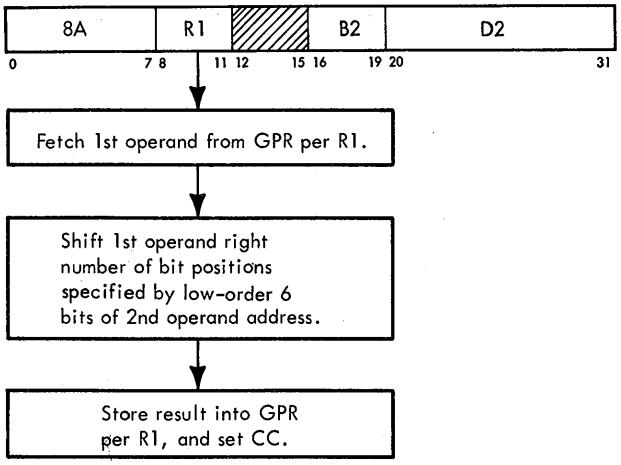
If PAL(58–61) was zero at the end of I-Fetch, the last shift has been performed and the low-order 31 bits of the high-order word, plus the inserted sign bit (per STAT C), are stored into the even GPR specified by R1. The CC is set per hardware conditions, and an end-op cycle is taken.

If PAL(58–61) was not zero, the low-order word has been transferred from the odd GPR to S, and one or more left 4 shifts remain to be performed. The low-order word in S is shifted left 4 and stored into S. The four overflow bits are retained in B(64–67), the high-order word in T is gated to PAA, B(64–67) is gated to PAB, and the result is shifted left 4 to PAL. This sequence shifts the high-order word, plus the overflow bits from the low-order word, left 4. The low-order 31 bits of the high-order word, plus the inserted sign bit (per STAT C), are gated to T and stored into the even GPR per R1. The number of left 4 shifts to be performed is determined by E(12–15), which was set per D(18–21) at the start of execution. E(12–15) is decremented after each left 4 shift is performed. The microprogram remains in the shift left 4 loop until E(12–15) is reduced to 0001, at which time the low-order word is stored into the odd GPR specified by R1 + 1. The CC is set per STAT A, STAT B, and the result sign [A(0)], and an end-op cycle is taken.

When the low-order word is shifted, a zero test is performed and STAT A is set if the low-order result is all zeros. When the high-order word is shifted, a zero test is performed and STAT A is reset if the high-order result is not all zeros. When the high-order word is shifted, an overflow test is performed; STAT B is set if an overflow occurred and a program fixed-point overflow interruption is taken if the associated mask bit in the PSW is a 1.

Shift Right Single, SRA (8A)

- Shift 1st operand (in GPR per R1) right number of bit positions specified by low-order six bits of 2nd operand address and place result into 1st operand location.
- RS format:



- Conditions at start of execution:
  First 16 bits of instruction are in E.
  1st operand is in S and T.
  Amount of shift is in D and PAL.
- D(18–23) indicates total amount of shift.
- Method of shifting: right 4 from PAA or PAB to PAL.
- Shifts of right 3 or less are obtained by combining left 1, left 2, or left 3 shifts with right 4 shift.
- E(12–15) indicates number of shifts to be performed in multiples of 4.
- E(12–15) is reduced by 1 after each shift of 4.
- CC setting:
  Result in PAL is zero: CC = 0.
  Result in PAL is less than zero: CC = 1.
  Result in PAL is greater than zero: CC = 2.

The Shift Right Single, SRA, instruction shifts the first operand right the number of bit positions specified by the low-order six bits of the second operand address. (The remainder of the address is ignored.)

The sign of the first operand remains unchanged. All 31 bits of the operand participate in the shift. Bits equal to the sign are supplied to the vacated high-order bit positions. Low-order bits are shifted without inspection and are lost.

Right-shifting is accomplished only be shifting right 4 bit positions at a time from the parallel adder to the PAL. Right shifts of less than 4 are obtained by combining left 1, left 2, or left 3 shifts with the right 4 shift. In the interest of speed, the desired amount of right shift is accomplished using a maximum number of right 4 shifts and a minimum number of left 1 and left 2 shifts (not more than one of each). Also, wherever possible, a left 1 or left 2 shift is combined with a right 4 shift because these combinations can be accomplished in one pass through the parallel adder. Table 3-8 shows how right-shifting is accomplished for any amount of shift desired.

Diagram 5-118, FEMDM, is a flowchart of the SRA operation. At the start of execution, the first 16 bits of the instruction are in E, the first operand is in S and T, and the number of bit positions to be shifted is in D and PAL.

In the SRA operation, PAL(58–63) initially determines the amount of shift. PAL(58–61) determines whether a shift of more than right 3 is to occur. If PAL(58–61) = 0, a shift of right 3 or less is to be performed; if PAL(58–61) does not equal 0, then shifts of right 4 or more are to occur.

As an example, assume that PAL(58–61) does not equal 0 and that PAL(62,63) = 01. A shift of right 5, right 9, or more is indicated. D(18–21) is transferred to E(12–15) to determine the number of right 4 shifts to be used if a shift of more than right 5 is to occur. PAL(32–63) is now tested for all 0's. If this condition is present, STAT A is set. The first operand in T(32–63) is transferred to AB, with T(32) propagated into A(26–31).

The first operand in T(32–63) is transferred to PAA(31–62), causing a left 1 shift of the data, and T(32) is propagated into PAL(26–31). PAA(31–62) and PAL(26–31) are transferred right 4 positions to PAL(35–66) and PAL(30–35), respectively, giving, in effect, a shift of right 3 positions. The data in PAL is transferred to AB(24–67), from where it is transferred to PAB(4–65), causing a left 2 shift. The total effective shift at this time is a right 1 shift. PAL(32–63) is now tested for all 0's, and STAT A is set if all 0's are present. PAL(32–63) is transferred to T(32–63), from where the data is transferred into the GPR per E(8–11). E(12–15) is tested for 0001. If this value exists, an end-op cycle is taken to complete the operation. If E(12–15) does not equal 0001, then the data continues to be shifted in multiples of 4, and E(12–15) is reduced by 1 for each right 4 shift until it equals 0001. At this time the CC is set per STAT A and the result sign [T(32)], and an end-op cycle is taken.

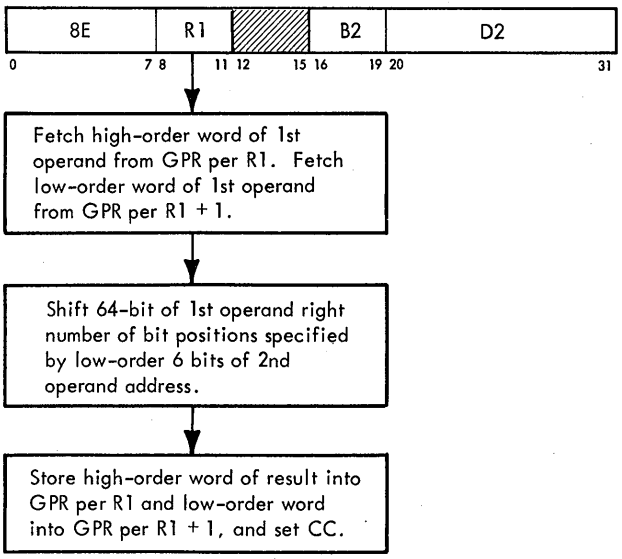
Right shifts of other amounts are performed in a similar manner. (Refer to Table 3-8 and Diagram 5-118, FEMDM.)

Table 3-8. Right Shift Combinations

PAL(58–61)	PAL(62,63)	Total Shift Desired	Incremental Shifting Sequence		
0000	00	None	None		
	01	Right 1	Left 1 and right 4	Left 2	
	10	Right 2	Left 2 and right 4		
	11	Right 3	Left 1 and right 4		
Not 0000	00	Right 4, 8, 12 and so on	Right 4	Right 4 until E(12–15) = 1	
	01	Right 5, 9, 13 and so on	Left 1 and right 4	Left 2 and right 4	Right 4 until E(12–15) = 1
	10	Right 6, 10, 14 and so on	Left 2 and right 4	Right 4	Right 4 until E(12–15) = 1
	11	Right 7, 11, 15 and so on	Left 1 and right 4	Right 4	Right 4 until E(12–15) = 1

Shift Right Double, SRDA (8E)

- Shift 1st operand (in GPR per R1 and R1 + 1) right number of bit positions specified by low-order six bits of 2nd operand address and place result into 1st operand location.
- RS format:



- Conditions at start of execution:
  First 16 bits of instruction are in E.
  1st operand is in S and T.
  Amount of shift is in D and PAL.
- D(18–23) indicates total amount of shift.
- E(12–15) indicates number of shifts to be performed in multiples of 4.
- E(12–15) is reduced by 1 after each shift of 4.
- Low-order bits of high-order word of doubleword operand are saved and placed into low-order word of operand.
- Value of operand sign is supplied to vacated bit positions.
- CC settings:
  Result in PAL equals zero: CC = 0.
  Result in PAL is less than zero: CC = 1.
  Result in PAL is greater than zero: CC = 2.

The Shift Right Double, SRDA, instruction right-shifts the doubleword first operand the number of bit positions specified by the low-order six bits of the second operand address. (The remainder of the address is ignored.) The R1 field of the instruction addresses the high-order 32 bits of the doubleword operand. The low-order word of the operand is in the GPR per R1 + 1. The R1 field must specify an even GPR; a program specification interruption occurs when R1 is odd.

The sign of the doubleword operand is the sign of the even GPR (R1). The sign bit (high-order bit) of the odd GPR (R1 + 1) is treated as an integer bit. Bits equal to the sign of the doubleword operand are supplied to the vacated high-order positions as the information is shifted. Bits shifted out of the low-order positions are lost.

Diagram 5-119, FEMDM, is a flowchart of the SRDA instruction. At the start of execution, the first 16 bits of the instruction are in E, the first operand is in S and T, and the number of bit positions to be shifted is in D and PAL.

The SRDA instruction execution is similar to that of the SRA instruction. The differences are a result of shifting a doubleword, one word at a time, in the SRDA instruction. Because the high-order word is shifted first, it is necessary to retain the underflow bits from the high-order word, to shift them right 4, and to append them to the low-order word.

Any specified amount of right shift involves at least one shift of right 4 for both the high-order word and the low-order word. Therefore, the underflow bits from the high-order word are appended to the low-order word [by transferring B(64–67) to PAB(28–31)] before the right 4 shift. This action results in the underflow bits and the low-order word being shifted right 4 as a unit.

Each time the shifted high-order word is stored, a zero test is performed and STAT A is set if the result is 0. When the low-order word has been shifted the same amount, STAT A is reset if the result is not 0. STAT A and the result sign [A(0)] determine the CC.

www.IBM360.info

This section discusses the 44 instructions making up the floating-point instruction set. Before analyzing the instructions, however, the following paragraphs discuss exponent overflow and underflow and zero results, and list the conditions at the start of execution. (For a discussion of number representation, data formats, normalization, operand addressing, instruction formats, data flow, program interruptions, and condition codes, see Chapter 1.)

EXPONENT OVERFLOW AND UNDERFLOW

- Exponent overflow occurs if two positive characteristics are added, or if positive number is added to positive characteristic, and final result is negative characteristic.
- Exponent underflow occurs if two negative characteristics are added, or if quantity is subtracted (complement added) from negative characteristic, and final result is positive characteristic.
- Exponent underflow program interruption occurs if PSW(38) = 1.

During floating-point operations, values may be chosen that cause the CPU to yield invalid results. For example, the largest positive exponent that can be expressed as a floating-point characteristic is +63, and is represented in excess-64 notation as 111 1111 (7F, hex). Assume that +63 is the characteristic of a floating-point operand and that a 1 is added to it:

```
0  1  2  3  4  5  6  7
S  1  1  1  1  1  1  1   +63 (Excess-64 Notation)
S  0  0  0  0  0  0  1   + 1
S  0  0  0  0  0  0  0
Carry
```

Note that the sum in bits 1–7, instead of indicating an exponent of +64, indicates an exponent of -64, 128 less than the true exponent. Thus, *exponent overflow* occurred. The rule for exponent overflow is: if two positive characteristics are added, or if a positive number is added to a positive characteristic, and the final result is a negative characteristic, exponent overflow occurred. This rule does not hold for intermediate result characteristics which may exceed the highest expressible exponent.

If exponent overflow occurs, an interruption is forced and cannot be masked off (refer to Chapter 1). The resulting invalid characteristic is not altered and remains in the result register for examination by the interruption-handling microprogram.

The largest negative exponent that can be expressed as a floating-point characteristic is -64, and is represented in excess-64 notation as a characteristic of all zeros. Assume that -64 is the characteristic of a floating-point operand and that a 1 is subtracted from it:

```
0  1  2  3  4  5  6  7
S  0  0  0  0  0  0  0   –64 (Excess-64 Notation)
S  1  1  1  1  1  1  1   2's Complement of 1
S  1  1  1  1  1  1  1
```

The difference in bits 1–7, instead of indicating an exponent of -65, indicates an exponent of +63, 128 more than the true exponent. This is known as *exponent underflow.* The rule for exponent underflow is: if two negative characteristics are added, or if a quantity is subtracted (complement-added) from a negative characteristic, and the final result is a positive characteristic, exponent underflow occurred. This rule does not hold for intermediate characteristics which may exceed the most negative expressible exponent.

If exponent underflow occurs, an interruption takes place only if the exponent-underflow mask bit [PSW(38)] is a 1 (refer to Chapter 1). If the interruption is taken, the resulting invalid characteristic is not altered and remains in the result register for examination by the interruption-handling microprogram. However, if the underflow interruption is masked off, the entire result (sign, characteristic, and fraction) is converted to a true zero (see "Zero Results"). This value can be a valid result for some calculations because exponent underflow indicates that the result was very small (less than 16^{-64}) and therefore close to zero.

Referring to the two examples given above, the CPU determines if exponent overflow or underflow occurred by examining bit 1 of the final characteristic and testing for a carry out of bit 1. If a carry occurred during a characteristic addition and bit 1 is a 0, exponent overflow occurred; if a carry did *not* occur during a characteristic subtraction and bit 1 is a 1, exponent underflow occurred.

ZERO RESULTS

A zero result is normally stored into the result register as a true zero; that is, a zero characteristic, a zero fraction, and a plus sign. A true zero may occur because of the magnitude of the operands or it may be forced. A true

zero is forced if exponent underflow occurs during add, subtract, multiply, or divide instructions and the exponent-underflow mask is off [PSW(38) = 0]. A true zero is also forced when a result fraction is zero and the program interruption for significance is masked off during add or subtract instructions. "Significance" means that an add or subtract instruction resulted in a zero fraction. This condition causes a significance program interruption if the significance mask is on [PSW(39) = 1]. When a significance condition occurs with the mask on, the result characteristic and sign remain unchanged and are stored with the zero fraction. True zero is *never* forced when a zero fraction occurs during a load, store, or halve instruction. Whenever a result has a zero fraction, an exponent overflow or exponent underflow condition is ignored.

CONDITIONS AT START OF EXECUTION

The conditions at the start of execution for the RR and RX instructions, short and long operands, are:

1. RR, Short Operands:
   a. First operand is in A, B, and D (24-bit fraction only).
   b. Second operand is in S and T.
   c. Instruction is in E.
2. RR, Long Operands:
   a. 32 bits of first operand are in A, B, and D (24-bit fraction only).
   b. 32 bits of second operand are in S and T.
   c. Low-order fractions of first and second operands are in LS.
   d. Instruction is in E.
3. RX, Short Operands:
   a. First operand is in S and T.
   b. Main storage request for second operand has been issued per D.
   c. First 16 bits of instruction are in E.
4. RX, Long Operands:
   a. 32 bits of first operand are in S and T.
   b. Low-order fraction of first operand is in LS.
   c. Main storage request for 2nd operand has been issued per D.
   d. First 16 bits of instruction are in E.

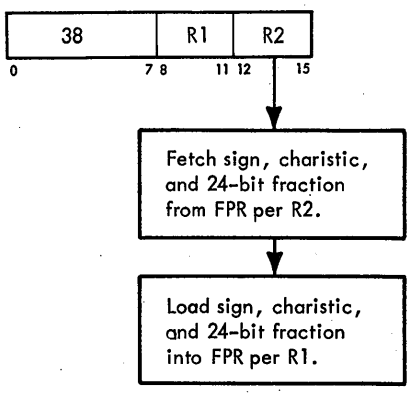
LOAD

The floating-point load instructions provide a means of loading operands into the LS FPR's. The load operation may be register-to-register (RR format) or storage-to-register (RX format) and may use short or long operands. In any case, the instruction loads the second operand into the first operand location, and the second operand location remains unchanged.

In addition, certain floating-point load instructions can test or modify the sign of the second operand before loading it into LS. The second operand may be also complemented before loading.

Load, LER (38) – RR Short Operands

- Load 2nd operand (in FPR, per R2) into 1st operand location (in FPR, per R1).
- RR format:

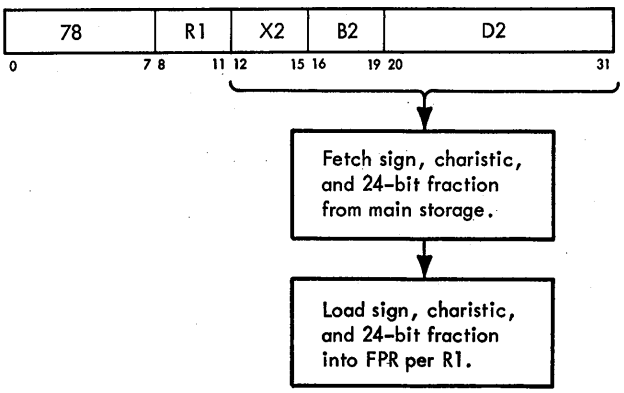


- Conditions at start of execution:
  1st operand is in A, B, and D (24-bit fraction only).
  2nd operand is in S and T.
  Instruction is in E.

The Load, LER, instruction (Diagram 5-202, FEMDM) loads the 32-bit second operand from the LS FPR specified by the R2 field into the first operand location specified by the R1 field. During the RR I-Fetch, the second operand is gated to S and T. At the beginning of the execution phase, a specification test is initiated. If no specification check occurred, the contents of T (second operand) are gated to the LS FPR specified by R1. If a specification check did occur, the operation is suppressed and a specification program interruption occurs.

Load, LE (78) – RX Short Operands

- Load 2nd operand (in storage) into 1st operand location (in FPR, per R1).
- RX format:



- Conditions at start of execution:
  1st operand is in S and T.
  Main storage request for 2nd operand has been issued per D.
  First 16 bits of instruction are in E.

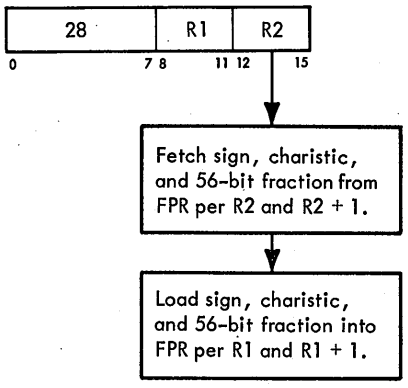
The Load, LE, instruction (Diagram 5-203, FEMDM) loads the 32-bit second operand from the main storage location specified by the effective address into the first operand location specified by the R1 field. The effective address of the second operand must be on a word boundary or a specification program interruption occurs. During the RX I-Fetch, the effective address is computed and placed into D. A main storage request for the second operand is then initiated per D.

At the beginning of the execution phase, a specification test is initiated. If no specification check occurred, either SDBO(0–31) or SDBO(32–63) is gated to T. Because a main storage request is always for a doubleword and only a word operand is desired, D(21) determines which word of the main storage doubleword is used; if D(21) = 1, the right word is gated to T; if D(21) = 0, the left word. From T, the second operand is gated to the LS FPR specified by R1.

If a specification check occured at the start of the execution phase, the operation is suppressed and a specification program interruption occurs.

Load, LDR (28) – RR Long Operands

- Load 2nd operand (in FPR, per R2 and R2 + 1) into 1st operand location (in FPR, per R1 and R1 + 1).
- RR format:



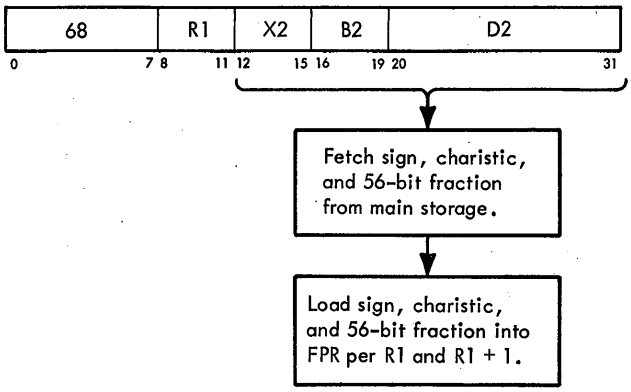
- Conditions at start of execution:
  32 bits of 1st operand are in A, B, and D (24-bit fraction only).
  32 bits of 2nd operand are in S and T.
  Low-order fractions of 1st and 2nd operands are in LS.
  Instruction is in E.

The load, LDR, instruction (Diagram 5-202, FEMDM) loads the doubleword second operand from the LS FPR specified by R2 and R2 + 1 into the first operand location specified by R1 and R1 + 1. During the RR I-Fetch, the high-order 32 bits of the second operand are placed into S and T. At the beginning of the execution phase, a specification test is initiated. If no specification check occurred, the low-order 32 bits of the second operand are fetched from the odd register of the even/odd pair of FPR's specified by the R2 field. From T, the low-order 32 bits of the second operand are loaded into the odd register of the even/odd pair of FPR's specified by the R1 field. The high-order 32 bits of the second operand are then gated from S to T. From T, they are loaded into the even register of the even/odd pair of FPR's specified by the R1 field.

If a specification check occurred at the start of the execution phase, the operation is suppressed and a specification program interruption occurs.

Load, LD (68) – RX Long Operands

- Load 2nd operand (in storage) into 1st operand location (in FPR, per R1 and R1 + 1).
- RX format:



- Conditions at start of execution:
  32 bits of 1st operand are in S and T.
  Low order fraction of 1st operand is in LS.
  Main storage request for 2nd operand has been issued per D.
  First 16 bits of instruction are in E.

The Load, LD, instruction (Diagram 5-203, FEMDM) loads the doubleword second operand from the main storage location specified by the effective address into the first operand location specified by R1 and R1 + 1. The effective address of the second operand must be on a doubleword boundary or a specification program interruption occurs. During the RX I-Fetch, the effective address is computed and placed into D. A main storage request for the second operand is then initiated per D.

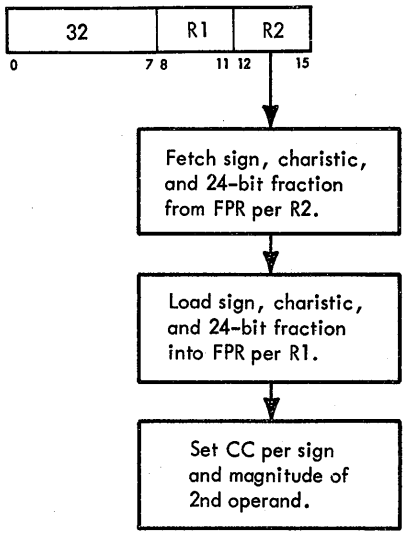
At the beginning of the execution phase, a specification test is initiated. If no specification check occurred, the doubleword second operand is gated from SDBO to ST. The low-order 32 bits are then loaded into the odd

register of an even/odd pair of FPR's specified by the R1 field. The high-order 32-bits of the second operand are then gated from S to T. From T, they are loaded into the even register of the even/odd pair of FPR's specified by the R1 field.

If a specification check occurred at the start of the execution phase, the operation is suppressed, and a specification program interruption occurs.

Load and Test, LTER (32) – RR Short Operands

- Load 2nd operand (in FPR, per R2) into 1st operand location (in FPR, per R1).
- RR format:



- Conditions at start of execution:
  1st operand is in A, B, and D (24-bit fraction only).
  2nd operand is in S and T.
  Instruction is in E.
- CC setting:
  2nd operand fraction equals zero: CC = 0.
  2nd operand is less than zero: CC = 1.
  2nd operand is greater than zero: CC = 2.

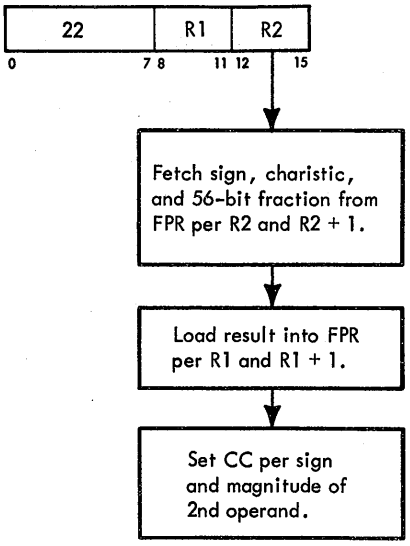
The Load and Test, LTER, instruction (Diagram 5-204, FEMDM) loads the 32-bit second operand from the FPR specified by the R2 field into the first operand location specified by the R1 field. The sign and magnitude of the second operand determine the CC, as follows:

1. If the fraction of the second operand equals zero, the CC is set to 0.
2. If the second operand (sign, characteristic, and fraction) is less than zero, the CC is set to 1.
3. If the second operand (sign, characteristic, and fraction) is greater than zero, the CC is set to 2.

The LTER execution is similar to LER execution. However, the sign of the second operand is saved in STAT C, and STAT A is set if the fraction equals zero. The CC is determined during the normal end-op cycle. If STAT A is set, indicating that the fraction equals zero, the CC is set to 0. The sign and characteristic are not considered when the fraction equals zero. If the second operand fraction is not equal to zero, the sign (STAT C) determines a greater-than-zero or less-than-zero condition. If the sign is minus (STAT C set), the second operand is less than zero and the CC is set to 1. If the sign is plus (STAT C reset), the second operand is greater than zero and the CC is set to 2. Setting the CC depends upon the 'Set-CR' micro-order, the instruction op-code, and the hardware conditions specified above.

Load and Test, LTDR (22) – RR Long Operands

- Load 2nd operand (in FPR, per R2 and R2 + 1) into 1st operand location (in FPR, per R1 and R1 + 1).
- RR format:



- Conditions at start of execution:
  32 bits of 1st operand are in A, B, and D (24-bit fraction only).
  32 bits of 2nd operand are in S and T.
  Low-order fractions of 1st and 2nd operands are in LS.
  Instruction is in E.
- CC setting:
  2nd operand fraction equals zero: CC = 0.
  2nd operand is less than zero: CC = 1.
  2nd operand is greater than zero: CC = 2.

The Load and Test, LTDR, instruction (Diagram 5-205, FEMDM) loads the doubleword second operand from the FPR specified by R2 and R2 + 1 into the first operand location specified by R1 and R1 + 1. The sign and magnitude of the second operand determine the CC, as follows:

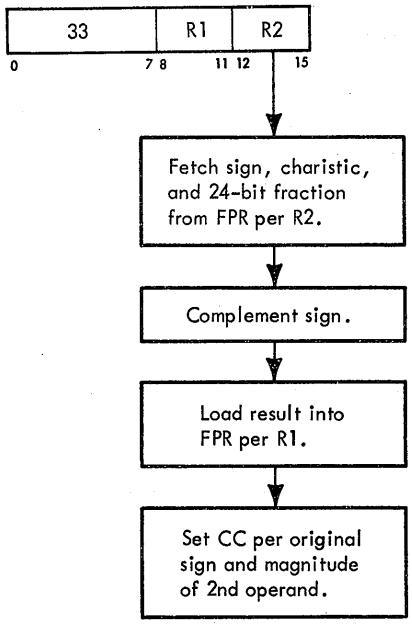
1. If the fraction of the second operand equals zero, the CC is set to 0.

2. If the second operand (sign, characteristic, and fraction) is less than zero, the CC is set to 1.
3. If the second operand (sign, characteristic, and fraction) is greater than zero, the CC is set to 2.

The LTDR execution is similar to LDR execution. However, the sign of the second operand is saved in STAT C, and STAT A is set if the fraction equals zero. The CC is determined during the normal end-op cycle in the same manner as explained for the LTER instruction.

Load Complement, LCER (33) – RR Short Operands

- Load 2nd operand (in FPR, per R2) into 1st operand location (in FPR, per R1) with sign complemented.
- RR format:



- Conditions at start of execution:
  1st operand is in A, B, and D (24-bit fraction only).
  2nd operand is in S and T.
  Instruction is in E.
- CC setting:
  2nd operand fraction equals zero: CC = 0.
  Original sign is plus: CC = 1.
  Original sign is minus: CC = 2.

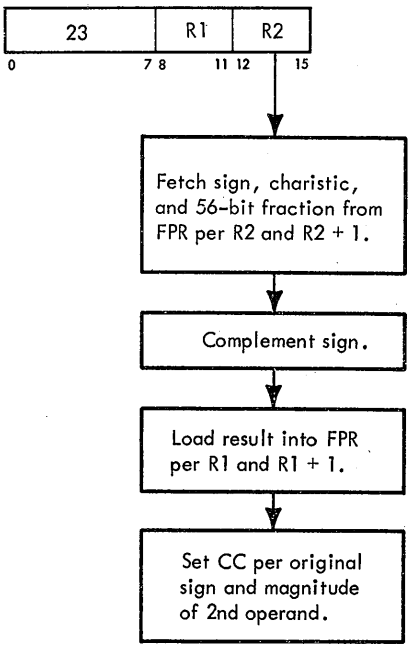
The Load Complement, LCER, instruction (Diagram 5-204) loads the 32-bit second operand from the FPR specified by the R2 field into the first operand location specified by the R1 field. During the loading, the sign is changed to the opposite value (complemented). The original sign and magnitude of the second operand determine the CC, as follows:

1. If the fraction of the second operand equals zero, the CC is set to 0.
2. If the original sign of the second operand is plus, the CC is set to 1.
3. If the original sign of the second operand is minus, the CC is set to 2.

Except for complementing the sign of the second operand, LCER execution is identical to LTER execution.

Load Complement, LCDR (23) – RR Long Operands

- Load 2nd operand (in FPR, per R2 and R2 + 1) into 1st operand location (in FPR, per R1 and R1 + 1) with sign complemented.
- RR format:



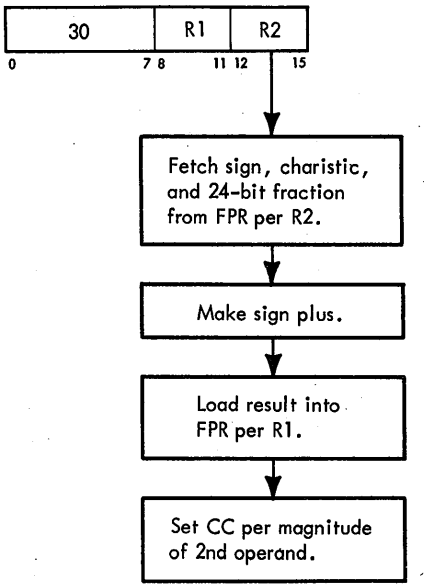
- Conditions at start of execution:
  32 bits of 1st operand are in A, B, and D (24-bit fraction only).
  32 bits of 2nd operand are in S and T.
  Low-order fractions of 1st and 2nd operands are in LS.
  Instruction is in E.
- CC setting:
  2nd operand fraction equals zero: CC = 0.
  Original sign is plus: CC = 1.
  Original sign is minus: CC = 2.

The Load Complement, LCDR, instruction (Diagram 5-205) loads the doubleword second operand from the FPR specified by R2 and R2 + 1 into the first operand location specified by R1 and R1 + 1. During the loading, the sign is complemented. The original sign and magnitude of the second operand determine the CC in the same manner as explained for the LCER instruction.

Except for complementing the sign of the second operand, LCDR execution is identical to LTDR execution.

Load Positive, LPER (30) — RR Short Operands

- Load 2nd operand (in FPR, per R2) into 1st operand location (in FPR, per R1) with sign made plus.
- RR format:



- Conditions at start of execution:
  1st operand is in A, B, and D (24-bit fraction only).
  2nd operand is in S and T.
  Instruction is in E.

- CC setting:
  2nd operand fraction equals zero: CC = 0.
  2nd operand is greater than zero: CC = 2.

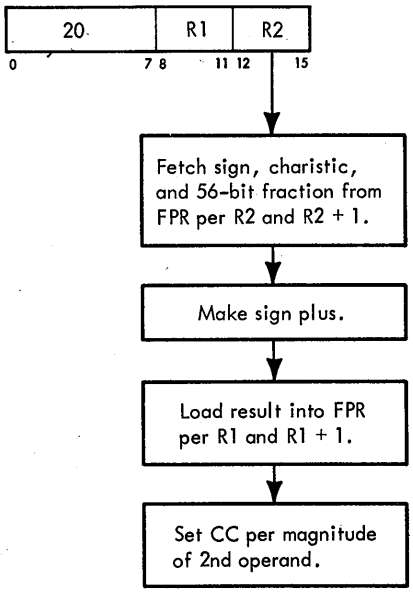
The Load Positive, LPER, instruction (Diagram 5-204) loads the 32-bit second operand from the FPR specified by the R2 field into the first operand location specified by the R1 field. During the loading, the sign is made plus. Thus, the result stored is always zero or greater. STAT A is set if the second operand fraction equals zero. The magnitude of the second operand determines the CC, as follows:

1. If the second operand fraction equals zero, the CC is set to 0.
2. If the second operand is greater than zero, the CC is set to 2.

Except for making the sign of the second operand plus, LPER execution is identical to LTER execution.

Load Positive, LPDR (20) — RR Long Operands

- Load 2nd operand (in FPR, per R2 and R2 + 1) into 1st operand location (in FPR, per R1 and R1 + 1) with sign made plus.
- RR format:



- Conditions at start of execution:
  32 bits of 1st operand are in A, B, and D (24-bit fraction only).
  32 bits of 2nd operand are in S and T.
  Low-order fractions of 1st and 2nd operands are in LS.
  Instruction is in E.
- CC setting:
  2nd operand fraction equals zero: CC = 0.
  2nd operand is greater than zero: CC = 2.

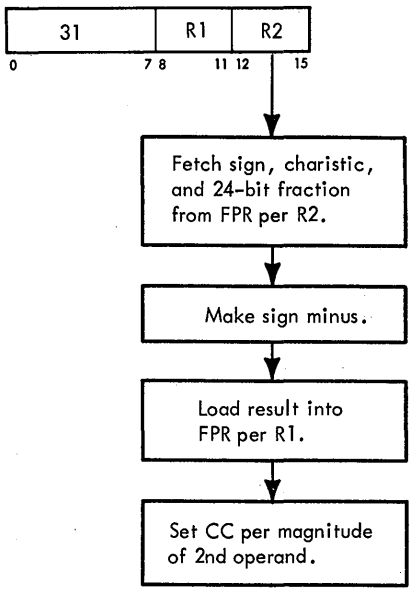
The Load Positive, LPDR, instruction (Diagram 5-205) loads the doubleword second operand from the FPR specified by R2 and R2 + 1 into the first operand location specified by R1 and R1 + 1. During the loading, the sign is made plus. Thus, the result stored is always zero or greater. STAT A is set if the second operand fraction equals zero. The magnitude of the second operand determines the CC in the same manner as explained for the LPER instruction.

Except for making the sign of the second operand plus, LPDR execution is identical to LTDR execution.

Load Negative, LNER (31) — RR Short Operands

- Load 2nd operand (in FPR, per R2) into 1st operand location (in FPR, per R1) with sign made minus.

- RR format:



- Conditions at start of execution:
  1st operand is in A, B, and D (24-bit fraction only).
  2nd operand is in S and T.
  Instruction is in E.

- CC setting:
  2nd operand fraction equals zero: CC = 0.
  2nd operand is less than zero: CC = 1.

The Load Negative, LNER, instruction (Diagram 5-204) loads the 32-bit second operand from the FPR specified by the R2 field into the first operand location specified by the R1 field. During the loading, the sign is made minus. Thus, the result stored is always zero or less. STAT A is set if the second operand fraction equals zero. The magnitude of the second operand determines the CC, as follows:

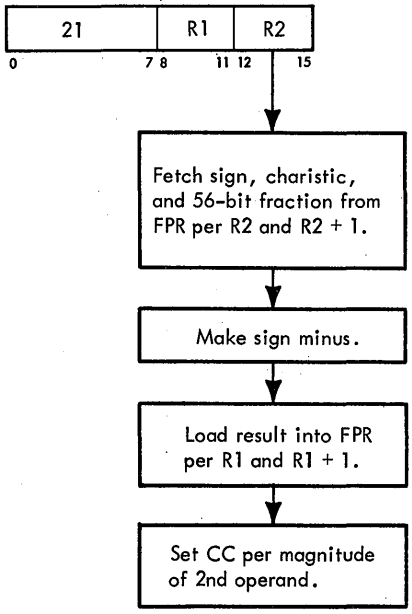
1. If the second operand fraction equals zero, the CC is set to 0.
2. If the second operand is less than zero, the CC is set to 1.

Except for making the sign of the second operand minus, LNER execution is identical to LTER execution.

Load Negative, LNDR (21) — RR Long Operands

- Load 2nd operand (in FPR, per R2 and R2 + 1) into 1st operand location (in FPR, per R1 and R1 + 1) with sign made minus.

- RR format:



- Conditions at start of execution:
  32 bits of 1st operand are in A, B, and D (24-bit fraction only).
  32 bits of 2nd operand are in S and T.
  Low-order fractions of 1st and 2nd operands are in LS.
  Instruction is in E.

- CC setting:
  2nd operand fraction equals zero: CC = 0.
  2nd operand is less than zero: CC = 1.

The Load Negative, LNDR, instruction (Diagram 5-205) loads the doubleword second operand from the FPR specified by R2 and R2 + 1 into the first operand location specified by R1 and R1 + 1. During the loading, the sign is made minus. Thus, the result stored is always zero or less. STAT A is set if the second operand fraction equals zero. The magnitude of the second operand determines the CC in the same manner as explained for the LNER instruction.

Except for making the sign of the second operand minus, LNDR execution is identical to LTDR execution.

ADD, SUBTRACT, AND COMPARE

- 20 add, subtract, and compare instructions.
- Short and long operands are available in both formats.
- Add and subtract instruction results may be normalized or unnormalized.
- Results are in true form with plus or minus values.

The 20 floating-point add-type instructions are divided into three major groups: add, subtract, and compare. RR and RX formats using short and long operands are available in each group. The results of add and subtract instruction may be normalized or unnormalized, depending upon the instruction being executed. A CC is set on all add-type instructions; the compare instructions cause a CC to be set with no result stored.

The CPU computes the sum of floating-point numbers as follows (assume that $0.004_{16} \times 16^5$ is to be added to $0.502_{16} \times 16^4$.):

1. Equalizes the characteristics.
   a. If the characteristics are unequal, the operand with the smallest characteristic is shifted right the number of hex digits necessary to equalize the characteristics. In the example, $0.502_{16} \times 16^4$ has the smallest characteristic (exponent); it is therefore changed to $0.0502_{16} \times 16^5$, thus making the characteristics equal.
   b. If the number of shifts exceeds the number of hex digits available, the operand with the largest characteristic becomes the intermediate result.
2. When the characteristics are equal, algebraically adds the first and second operand fractions.
   a. If the signs are alike, adds the first operand fraction to the second operand fraction. In the example, the signs are alike; therefore, the fractions are added giving a sum of $0.0542_{16} \times 16^5$.
   b. If the signs are unlike, subtracts the second operand fraction from the first operand fraction (adds the 2's complement of the second operand fraction to the first operand fraction).
3. If the intermediate result fraction is in complement form, recomplements it (takes 2's complement) to obtain the true fraction value.
4. Normalizes the intermediate result, if normalization is called for. Assume a 3-digit machine. If normalization is specified, the final result of the example (2a above) becomes $0.542_{16} \times 16^4$. If normalization is *not* specified, the low-order digit (guard digit) is truncated, and the final result is $0.054_{16} \times 16^5$.
5. Determines the sign and characteristic value.
6. Stores the sign, characteristic, and fraction into LS as specified by the R1 field.
7. Sets the CC per hardware conditions.

For subtraction of floating-point numbers, the algebraic rule applies: to subtract two numbers, change the sign of the subtrahend and proceed as in addition. When subtracting floating-point numbers, the sign of the second operand is complemented. The rules of addition apply as outlined in the previous paragraph. To illustrate, assume the same numbers as used above and that the signs are unlike. Thus, a 2's complement add (step 2b above) is performed:

$$\begin{array}{ll} .0502_{16} & \\ \underline{.FFC0_{16}} & \text{(2's complement of } .0040_{16}\text{)} \\ .04C2_{16} & \end{array}$$

The difference, then, is $0.04C_{16} \times 16^5$ if unnormalized; or $0.4C2_{16} \times 16^4$, if normalized.

The compare instructions are similar to the subtract instructions; the results, however, are not stored. The compare instructions algebraically compare the first operand with the second operand and set the CC accordingly. These objectives are accomplished by complementing the sign, algebraically adding the fractions, determining a high, low, or equal condition, and setting the CC.

The basic objectives of the add-type instructions are shown in Sheet 1 of Diagrams 5-206 and 5-207, FEMDM (short and long operands, respectively). After the RR or RX I-Fetch and the specification test, the remaining operand and/or low-order fraction(s) must be fetched or the low-order fractions reset to zeros. The signs are saved in STAT's. For short operand instructions (Diagram 5-206), zeros are gated to the low-order fractions of the 64-bit operands.

The characteristics are then compared. Preshifting occurs, if necessary, followed by the addition or subtraction of the fractions. Because the characteristics must be equal before algebraically adding the operands, the characteristics are subtracted to determine whether they are equal and whether preshifting is meaningful. For short operands, the characteristic difference must be 7 or less; for long operands, 15 or less.

If the characteristic difference is greater than 7 (short operands) or 15 (long operands) the fraction resulting after right-shifting equals zero. Therefore, preshifting is not performed, and the operand with the largest characteristic becomes the result. If preshifting is meaningless, the value in AB or ST is the result. If the characteristics are within range, the smallest fraction is right-shifted until the characteristics are equal; the fractions are then added algebraically to form an intermediate result. If a high-order carry occurs as a result of the addition (overflow), the intermediate result is right-shifted one hex digit and the characteristic is increased by 1. If this increase causes a characteristic overflow, an exponent-overflow program interruption occurs.

The intermediate result consists of 7 or 15 hex digits and a possible carry. The low-order digit is a guard digit retained from the fraction which is shifted right. Only one

guard digit participates in the fraction addition. The guard digit is zero if no shift occurs.

After the addition or subtraction, a test is made for compare instructions, normalized instructions, or unnormalized instructions. Postnormalization, recomplementation, and/or fraction overflow correction is accomplished during this phase. The final result is stored (except on compare instructions), and the CC is set according to the computed results. An end op completes instruction execution.

For normalized instructions the intermediate result fraction is left-shifted as necessary to form a normalized fraction. Vacated low-order digit positions are filled with zeros, and the characteristic is reduced by the amount of the shift. If normalization causes the characteristic to underflow, an exponent-underflow program interruption condition exists; the sign, characteristic, and fraction are made zero if the underflow mask bit [PSW(38)] is a 0. If PSW(38) is a 1, a program interruption occurs, and the characteristic is made 128 larger than the true result; the sign and fraction remain unchanged. If no left shift takes place, the guard digit is removed to obtain the proper fraction length.

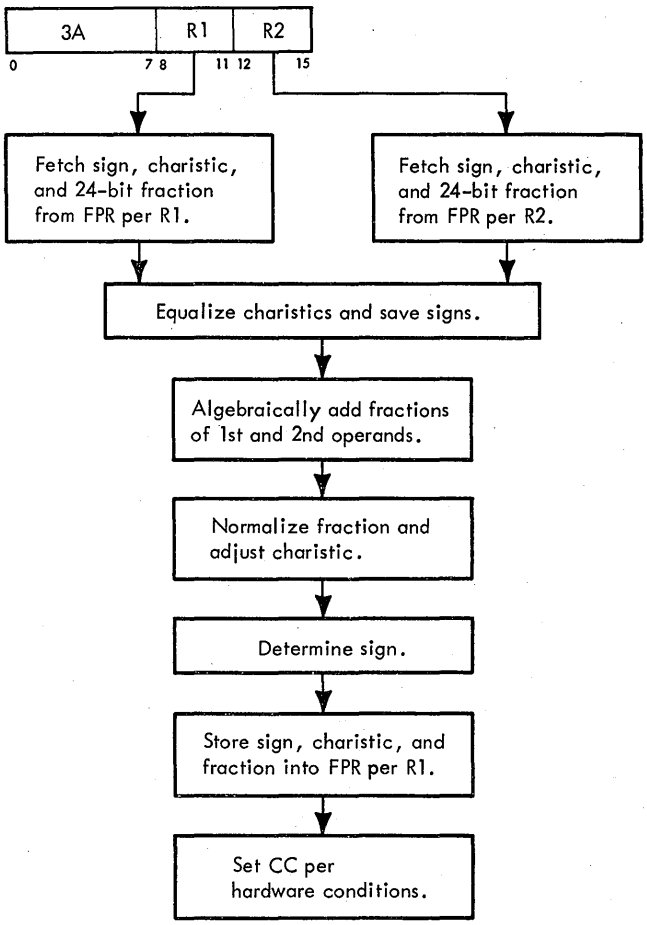
When the intermediate result fraction is zero and the significance mask bit is a 1, a significance program interruption takes place. No normalization occurs, and the intermediate result characteristic remains unchanged. When the intermediate result is zero and the significance mask bit is a 0, a significance program interruption does not occur; rather, the characteristic and the sign are made zero, yielding a true zero result. Exponent underflow does not occur for a zero fraction.

The sign of the result is derived algebraically. However, the sign of a zero result fraction is always positive.

Add Normalized, AER (3A) — RR Short Operands

- Algebraically add 2nd operand (in FPR, per R2) to 1st operand (in FPR, per R1) and place normalized sum into 1st operand location.
- RR format: (See adjoining column.)
- Conditions at start of execution:
  1st operand is in A, B, and D (24-bit fraction only).
  2nd operand is in S and T.
  Instruction is in E.
- CC setting:
  Result fraction equals zero: CC = 0.
  Result fraction is less than zero: CC = 1.
  Result fraction is greater than zero: CC = 2.

The Add Normalized, AER, instruction (Diagram 5-206) algebraically adds the second operand (specified by R2) to the first operand (specified by R1), and places the normalized sum into the first operand location. The CC is set according to hardware conditions. The AER instruction uses 32-bit operands; therefore, the contents of the low-order halves of the FPR's in LS remain unchanged.



At the beginning of the execution phase:

1. The first operand is in A, B, and D (24-bit fraction only).
2. The second operand is in S and T.
3. The STC was set to 4 during I-Fetch.
4. The AER instruction is in E.

A specification test is made at the beginning of the execution phase. If a specification check exists, instruction execution is suppressed, and a specification program interruption occurs. Assume that no specification check exists.

Because the AER instruction uses short operands (32 bits) in the RR format, no operand fetch during instruction execution is necessary. The sign of the first operand is saved in STAT F and the sign of the second operand is saved in STAT C. The first operand characteristic is subtracted from the second operand characteristic to determine the characteristic difference. Next, the second operand is gated to D and S. Because the AER instruction operates on short operands, B and T are reset, and the operand is treated as a 56-bit fraction with the low-order bits set to zero. The characteristic difference and the signs

determine the next operation to be performed by means of a 10-way 'FLR' micro-order branch.

The 10-way 'FLR' branch on characteristic difference and signs occurs for all add-type instructions. When this branch is encountered, the conditions are as follows:

1. The first operand is in AB (B equals zero for short operands).
2. The second operand is in DT (T equals zero for short operands) and S.
3. The sign of the first operand is in STAT F.
4. The sign of the second operand is in STAT C.
5. The characteristic difference is in SAL and F.

To subtract the first operand characteristic from the second operand characteristic, the 2's complement of the first operand characteristic is added to the second operand characteristic. Because the serial adder consists of 8 binary bit positions, a 1 is forced into bit position 0 on the A-bus (first operand side) of the serial adder, and a 0 is forced into bit position 0 on the B-bus (second operand side) of the serial adder. The characteristic difference is then routed to SAL and F.

The 10-way 'FLR' branch is determined by the result of the characteristic subtraction and by the signs in STAT F and STAT C. The 10-way 'FLR' branch affects bits 8–11 of ROSAR as defined in Sheet 2 of Diagram 5-206. ROSAR(8–11) is set as follows:

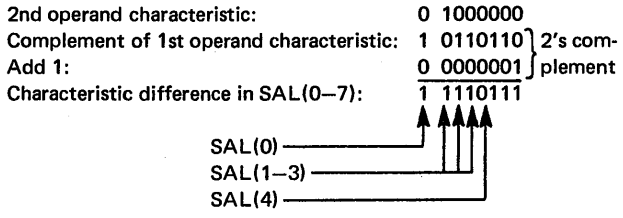
1. ROSAR(8) = 1 when a serial adder carry indicates that the second operand is greater than or equal to the first operand.
2. ROSAR(9) = 1 when the signs are alike. If the signs are alike, the fractions are added; if unlike, the fractions are subtracted.
3. ROSAR(10) = 1 when the characteristic difference is within range. ROSAR(10) equaling 1 implies that the characteristic difference is small enough so that equalizing the characteristics is meaningful. (A zero fraction may occur as a result of characteristic equalization.)
4. ROSAR(11) = 1 when the result in SAL is zero. This condition indicates that the characteristics are equal. A serial adder carry also occurs; thus ROSAR(8) will also equal 1.

Assume that the following two characteristics are to be compared to determine the ROSAR value:

2nd operand characteristic: $64_{10} = 1000000_2$

1st operand characteristic: $73_{10} = 1001001_2$

The first operand characteristic is subtracted from the second operand characteristic shown below:



Because no SAL(0) carry occurred, ROSAR(8) = 0. ROSAR(9) depends upon the signs assigned to the fractions of the two operands. The table in Sheet 2 of Diagram 5-206 shows the bit positions tested in SAL to determine the value of ROSAR(10) and whether preshifting is meaningful. In this example, SAL(0–3) = 1111 (binary). Because a short operand instruction is being executed, ROSAR(10) = 0; therefore, the value is in AB because no carry from SAL(0) indicated that the first operand is greater than the second operand. ROSAR(11) = 0 because SAL(0–7) is not all 0's.

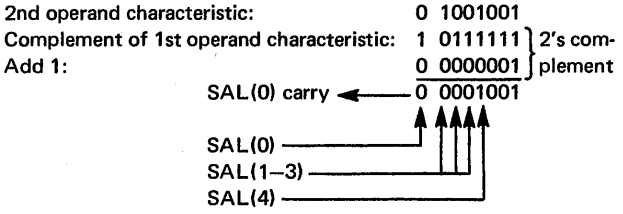
If all positions of SAL are 0, ROSAR(11) = 1, indicating that the characteristics of the two operands are equal. ROSAR(9) then determines the next operation (signs alike, add; signs unlike, subtract). In this case [SAL(0–7) = 0], ROSAR(8) also equals a 1 because a SAL(0) carry occurred.

For the next example, assume that the two characteristics in the previous example are interchanged. The characteristics are compared as follows:

2nd operand characteristic: 1001001

1st operand characteristic: 1000000

The first operand characteristic is subtracted from the second operand characteristic shown below:



In this example, ROSAR(8) = 1 because a SAL(0) carry occurred. ROSAR(10) = 0. Because preshifting in this example is meaningless, the operand in ST is the result fraction because the SAL(0) carry indicates that the second operand is the largest.

The two examples just discussed illustrate the determination of the ROSAR(8–11) values. Additional examples are shown in Table 3-9. ROSAR(8–11) determines the ROS branch that performs the next steps in executing the AER instruction.

The examples of determining the ROSAR(8–11) values as shown in Table 3-9 indicate that the fraction of the operand with the smallest characteristic is shifted right when the characteristic difference is seven or less.

Four possible ROSAR(8- '1) values (0010, 0110, 1010, and 1110) cause characteristic equalization and then an algebraic addition or subtraction of fractions. For example, assume that the AER instruction requires characteristic equalization, fraction subtraction, recomplementation (second operand fraction is greater than first operand fraction), and normalization. Further, assume a ROSAR(8–11) value of 0010. The 0010 branch causes one right shift of the second operand to occur, and a 1 is

Table 3-9. Examples of Branching on Characteristic Difference

Description	Example No. 1	Example No. 2	Example No. 3	Example No. 4	Example No. 5
2nd operand characteristic	1000000	1000000	1001000	1000111	1000000
1st operand characteristic	1001001	1001000	1000000	1000000	1000000
2nd operand characteristic	0 1000000	0 1000000	0 1001000	0 1000111	0 1000000
Complement of 1st operand characteristic	1 0110110	1 0110111	1 0111111	1 0111111	1 0111111
Add 1	0 0000001	0 0000001	0 0000001	0 0000001	0 0000001
Difference in SAL(0–7)	1 1110111	1 1111000	← 0 0001000	← 0 0000111	← 0 0000000
			SAL(0) carry	SAL(0) carry	SAL(0) carry
Within Range?* Short	No	Yes	No	Yes	Yes
Within Range?* Long	Yes	Yes	Yes	Yes	Yes
ROSAR(8–11) value	Sub 0000 Add 0100	Sub 0010 Add 0110	Sub 1000 Add 1100	Sub 1010 Add 1110	Sub 1011 Add 1111
Comments	Result in AB.'	Equalize fraction in ST.	Result in ST.	Equalize fraction in AB.	Add or sub. No shift necessary.

Notes:

1. ROSAR(8) = 1 when there is a serial adder carry. A carry indicates that R2 ≧ R1.
2. ROSAR(9) = 1 when the signs are alike.
*3. ROSAR(10) = 1 when the characteristics are within range. ROSAR(10) = 1 on SAL results as follows:
   a. SAL carry and SAL(0–3) = 0 and long operands.
   b. SAL carry and SAL(0–4) = 0.
   c. No SAL carry and SAL(0–3) = 1's and long operands.
   d. No SAL carry and SAL(0–4) = 1's.
4. ROSAR(11) = 1 when the SAL outputs equal 0.

added to F. Note that one guard digit is retained. SAL(4–7) is checked for 1111 (binary). When SAL(4–7) = 1111, the characteristics are equal. Because the test for a branch is made one machine cycle before the ROS branch occurs, the SAL value is one machine cycle behind the actual shift count. For this reason, a test is made for 1111 in SAL(4–7) instead of for 0000.

Once the characteristics are equal, the second operand is subtracted from the first operand (signs unlike). To subtract fractions (signs unlike), the 2's complement of the second operand fraction in DT is added to the first operand fraction in AB with the intermediate fraction result placed into AB and DT. The intermediate fraction result may be in true form or in complement form, or it may be equal to zero. If a zero fraction results, STAT A is set.

Because the larger characteristic is used as the characteristic of the result, it is gated to F(1–7); F(0) is reset.

If the second operand fraction is greater than the first operand fraction, the result is in complement form. Conversely, the result is in true form if the first operand fraction is greater than the second operand fraction. If A(7) = 1, the intermediate result is in complement form. If A(7) = 0, the intermediate result is in true form. When the intermediate result is in complement form, the result must be recomplemented; that is, the 2's complement of the intermediate result is taken after the algebraic addition.

When the result is in true form, and if the fraction is not equal to zero, the fraction must be normalized and stored and the CC set. Because the AER instruction is a normalized instruction, the intermediate result is normalized, if necessary. After normalization, the sign and the characteristic are inserted and stored with the fraction into the first operand location (specified by R1). Assuming no errors or zero fraction, the CC is set. An end-op cycle completes instruction execution.

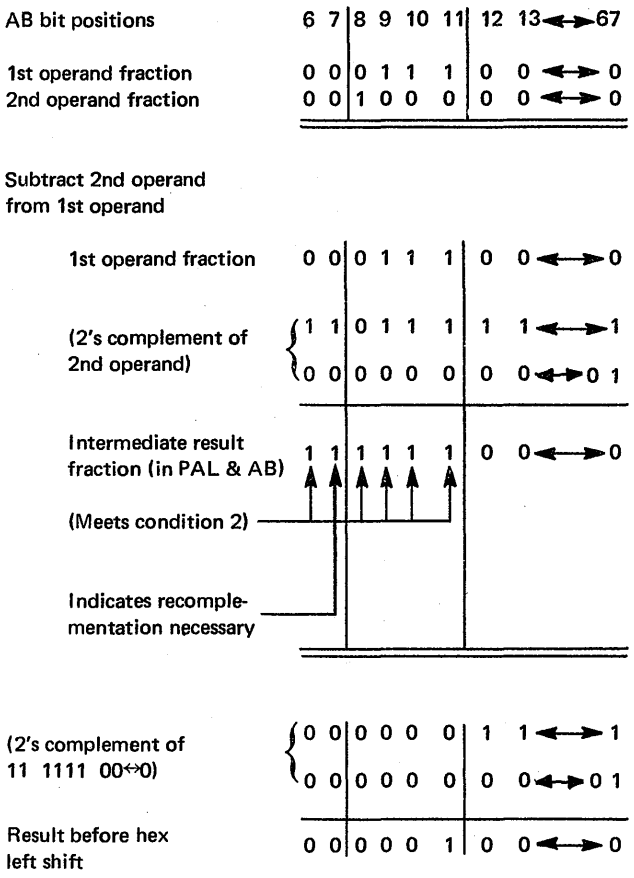
When in the normalizing loop after subtraction, the intermediate fraction result can be left-shifted out of the high-order hex digit position if the intermediate fraction is 0001. This left shift results in a zero fraction. The zero fraction, in this case, is not a true zero result or a significance condition; therefore, the true value must be restored.

Because the test for ROS branches is made one machine cycle before the ROS branch occurs, a test for

normalization is made before the recomplementation is performed. Therefore, the test for normalization is determined by the following conditions:

1. PAL(7–11) is 0's and PAL(7–67) is not 0's.
2. PAL(6,8–11) is 1's and PAL(7–67) is not 0's.

If one of these two conditions is met, at least one normalization cycle is performed after the recomplementation machine cycle. This action is not always necessary, however. For example, if the following fractions are subtracted, the assumed normalization cycle is not necessary:



The intermediate result fraction above shows that PAL(6,8–11) equals 1's and PAL(7–67) does not equal 0's. This condition causes a branch to the ROS normalization routine. Because A(7) = 1, the intermediate result fraction is in complement form, and the 2's complement of the intermediate fraction must be performed to obtain the true result fraction. As shown in the example, the true result fraction is .0001 0↔0. The one hex left shift that occurs yields a zero fraction result. Therefore, the result fraction located in DT is the true result fraction. The contents of D are transferred to T, and the sign, characteristic, and high-order fraction are stored into the FPR per the R1 field. An end op completes instruction execution.

If the signs were alike and the characteristic difference was within range (i.e., seven or less), the fractions are added [ROSAR(8–11) = 0110, 1110, or 1111]. First, however, the characteristics are equalized, if necessary, by right-shifting the fraction of the operand with the smallest characteristic one hex digit at a time and subtracting one from the characteristic difference. When the characteristic difference goes to zero, the characteristics are equal. Note that, if the first operand characteristic is larger, a 1 is *added* to the characteristic difference because the characteristic difference is in complement form.

The fractions are then added and the results are gated to DT and AB. A guard digit is retained in B(64–67). The characteristic of the larger operand, which is the characteristic of the intermediate result, is placed into F. When adding fractions, the possibility of a fraction overflow exists. A fraction overflow is indicated by a carry out of the high-order position, PA(8). After the two fractions are added, the intermediate result is placed into DT and AB. If A(7) = 1, a fraction overflow occurred. Therefore, the fraction is right-shifted one hex digit, and 1 is added to the intermediate result characteristic. Whether or not a fraction overflow occurred, the microprogram now determines whether the intermediate result fraction is normalized, not normalized, or equal to zero (the guard digit is included in the test for zero). If the fraction is normalized, the contents of STAT C are gated to the sign position of the LS bus, the result characteristic is transferred from F to T(32–39), and the result fraction is gated from D to T(40–63). This result (sign, characteristic, and fraction) is stored into the FPR specified by the R1 field. If the intermediate result fraction is not normalized, it is normalized by shifting left one hex digit at a time and subtracting one from the characteristic until a significant hex digit appears in the high-order position. After normalizing, the operation continues in the same manner as a normalized intermediate fraction. If the intermediate result is zero, a true zero is stored into the FPR specified by the R1 field, and the significance mask bit [PSW(39)] is tested. If the mask bit is a 1, the characteristic is stored with the zero fraction, and a program interruption is initiated. Regardless of the setting of the mask bit, the CC is set to 0, and the instruction is terminated by an end op.

If the intermediate result fraction is shifted right, the possibility of an exponent overflow exists. During normalization of the fraction, the possibility of an exponent underflow exists. When SAL(0) = 1 after a fraction shift (right or left), an exponent overflow or exponent underflow condition exists. The proper sign and fraction are stored. Because bit 0 of the characteristic (characteristic carry) is inhibited from entering LS by the 'RSLT-SIGN→LS' micro-order, the characteristic stored is 128 larger than the true result for underflow and 128 smaller on overflow.

Whenever an exponent overflow or exponent underflow condition exists, SAL(1) and PSW(38) are examined to determine whether a program interruption is to be executed. If SAL(1) = 0, an exponent overflow exists, and an interruption request is unconditionally generated. A program interruption occurs on all exponent overflows. If SAL(1) = 1, an exponent underflow exists; if PSW(38) = 1, an exponent underflow program interruption request is generated. If PSW(38) = 0, exponent underflow is masked off, and a true zero is stored with no program interruption occurring.

Note that an interruption occurs on all exponent overflows. This overflow indicates that the value of the absolute result exceeds the limits of the machine; therefore, further action is necessary. In some scientific computations, very small numbers may be eliminated from an equation without serious error. An exponent underflow means that the computed result approaches zero. Therefore, the programmer may find that a program interruption is unnecessary, and a true zero result is desirable.

Significance and specification program interruption conditions may also exist during execution of an AER instruction. The action that occurs is shown in Diagram 5-206, and is discussed earlier in this section.

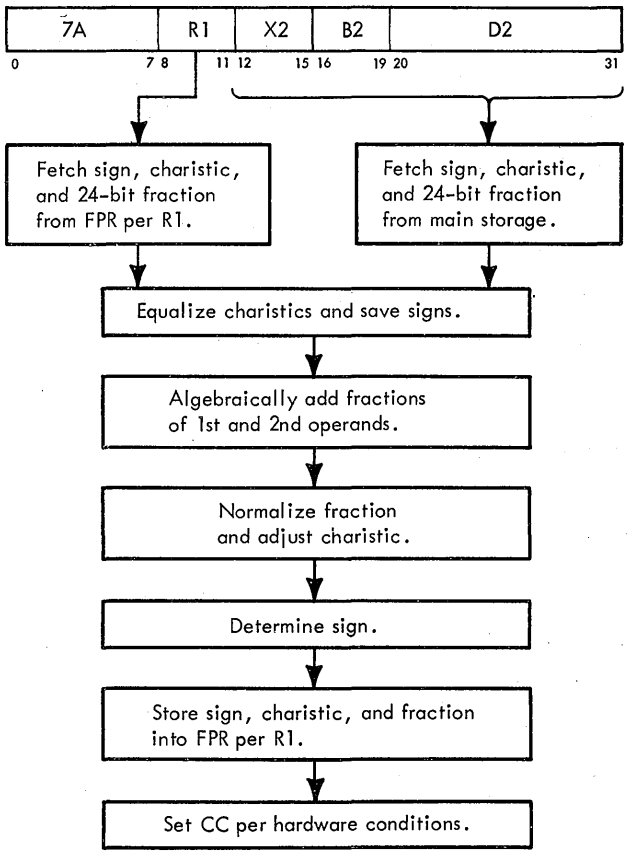
Tests for zero intermediate results are made at several points during instruction execution. If the result is zero, a program interruption occurs if PSW(39) = 1. The positive sign, the result characteristic, and a zero fraction are stored into LS. A program interruption occurs, and the program interruption routine determines the action to be taken. If PSW(39) = 0, a true zero result is stored into LS.

If the characteristics are not within limits when executing the AER instruction [ROSAR(8–11) = 0000, 0100, 1000, or 1100 on 'FLR' branch], the fraction with the largest characteristic is normalized and stored along with the sign into LS per the R1 field.

This discussion of the AER instruction is referred to in the discussions of Add-type instructions that follow. If the AER instruction is understood and the instruction differences noted, any short operand add-type instruction execution path can be followed by referring to Diagram 5-206.

Add Normalized, AE (7A) — RX Short Operands

- Algebraically add 2nd operand (in storage) to 1st operand (in FPR, per R1) and place normalized sum into 1st operand location.
- RX format: (See adjoining column.)
- Conditions at start of execution:
  1st operand is in S and T.
  Main storage request for 2nd operand has been issued per D.
  First 16 bits of instruction are in E.



- CC setting:
  Result fraction equals zero: CC = 0.
  Result fraction is less than zero: CC = 1.
  Result fraction is greater than zero: CC = 2.

The Add Normalized, AE, instruction (Diagram 5-206) algebraically adds the second operand (specified by the effective address) to the first operand (specified by R1), and places the normalized sum into the first operand location. The CC is set according to hardware conditions. The AE instruction uses 32-bit operands; therefore the contents of the low-order halves of the FPR's remain unchanged.

The conditions at the beginning of the execution phase are:

1. The first operand is in S and T.
2. A main storage request for the second operand has been issued per the effective address in D.
3. The contents of A and B are unknown.
4. The first 16 bits of the instruction are in E.

A specification test is made at the beginning of the execution phase. If a specification check exists, instruction execution is suppressed, and a specification program interruption occurs. Assume that no specification check exists.

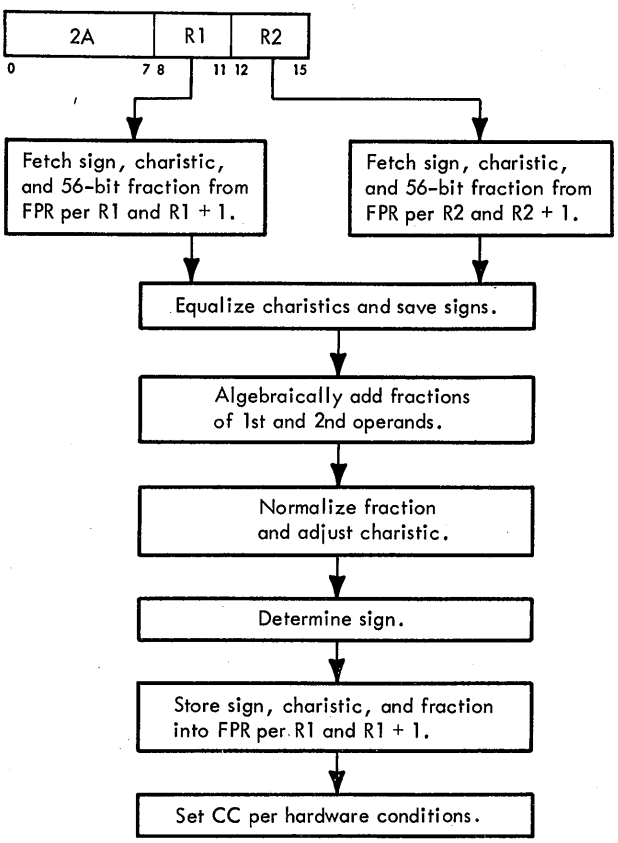
Because the AE instruction uses short operands (32 bits) in the RX format, no low-order fractions need to be

fetched. The first operand is moved from T to A. The second operand arrives from main storage and is placed into T. D(21) determines which word from the SDBO is gated to T. (Note that main storage is addressed on doubleword boundaries.) If D(21) = 0, SDBO(0–31) is gated to T; if D(21) = 1, SDBO(32–63) is gated to T.

The remainder of the AE instruction execution is identical to the execution of the AER instruction.

Add Normalized, ADR (2A) – RR Long Operands

- Algebraically add 2nd operand (in FPR, per R2 and R2 + 1) to 1st operand (in FPR, per R1 and R1 + 1) and place normalized sum into 1st operand location.
- RR format:



- Conditions at start of execution:
  32 bits of 1st operand are in A, B, and D (24-bit fraction only).
  32 bits of 2nd operand are in S and T.
  Low-order fractions of 1st and 2nd operands are in LS.
  Instruction is in E.
- CC setting:
  Result fraction equals zero: CC = 0.
  Result fraction is less than zero: CC = 1.
  Result fraction is greater than zero: CC = 2.

The Add Normalized, ADR, instruction (Diagram 5-207, FEMDM) algebraically adds the second operand (specified by R2 and R2 + 1) to the first operand (specified by R1 and R1 + 1), and places the normalized sum into the first operand location. The CC is set according to hardware conditions. The ADR instruction uses 64-bit operands.

The conditions at the beginning of the execution phase are:

1. 32 bits of the first operand (sign, characteristic, and high-order fraction) are in A and B, and D contains the high-order 24 bits of the fraction.
2. 32 bits of the second operand (sign, characteristic, and high-order fraction) are in S and T.
3. The STC contains a count of 4.
4. The instruction is in E.

Because the ADR instruction uses long operands (64 bits) in the RR format, the low-order fractions of the first and second operands must be fetched from LS. The low-order fraction of the first operand is fetched from LS per E(8–11) + 1 and is placed into B via T and the parallel adder. The low-order fraction of the second operand is fetched from LS per E(12–15) + 1 and placed into T; the high-order fraction is placed into D.

The sign of the first operand is saved in STAT F and the sign of the second operand is saved in STAT C. The first operand characteristic is subtracted from the second operand characteristic, and the characteristic difference and the signs determine the next operation by means of a 10-way 'FLR' branch.

Note that with long operands, the characteristics can be equalized if the characteristic difference is less than or equal to 15. Assume that the following two characteristics are to be compared to determine the ROSAR value:

2nd operand characteristic: $64_{10} = 1000000_2$
1st operand characteristic: $73_{10} = 1001001_2$

The first operand characteristic is subtracted from the second operand characteristic as follows:

2nd operand characteristic:	0 1000000	
Complement of 1st operand characteristic:	1 0110110	2's com-
Add 1:	0 0000001	plement
Characteristic difference in SAL(0–7):	1 1110111	

Because no SAL(0) carry occurred, ROSAR(8) = 0. ROSAR(9) depends upon the signs assigned to the fractions of the two operands. The table in Sheet 2 of Diagram 5-207 shows the bit positions tested in SAL to determine the value of ROSAR(10) and whether preshifting is meaningful. In this example, SAL(0–3) = 1111. Therefore, because ADR is a long operand instruction, ROSAR(10) is set to 1 and preshifting is meaningful. ROSAR(11) = 0 because SAL(0–7) is not all 0's. For

other examples of ROSAR setting on an 'FLR' branch, refer to the description of the AER instruction and Table 3-9. The examples shown in Table 3-9 indicate that the fraction of the operand with the smallest characteristic is shifted right when the characteristic difference is 15 or less for long operand instructions.

Four possible ROSAR(8–11) values (0010, 0110, 1010, and 1110) cause characteristic equalization and then an algebraic addition or subtraction of fractions. For example, ROSAR(8–11) is set to 0110, if an ADR instruction is being executed with operands having the following parameters: (1) the characteristic difference is less than 15, (2) the signs of the operands are alike, and (3) the second operand characteristic is the smaller of the two. The 0110 branch right-shifts the second operand in DT one hex digit and adds 1 to the characteristic difference in F until the characteristics are equal [SAL(4–7) = 1111].

After the characteristics are equalized, the fractions of the operands are added and the sum is placed into AB and DT. A plus sign is set into F(0) (in case of a zero fraction result), and the characteristic of the first operand is placed into F(1–7). STAT A is set if the fraction of the result equals zero.

Because ADR is a normalized instruction, the intermediate result is normalized, if necessary. First, however, a test determines whether the fraction overflowed due to the addition. (A carry into bit 7, the low-order bit of the characteristic, is considered a fraction overflow.) If the fraction did overflow, it is shifted right one hex digit, and a 1 is added to the characteristic.

The low-order fraction is then stored into the FPR specified by R1 + 1, and normalization proceeds if the result was not zero and *not* normalized. The low-order fraction is stored after each left-shift.

After normalization, the sign and the characteristic are inserted and stored with the high-order fraction into the first operand location (specified by R1). Assuming no error conditions or zero fraction, the CC is set, and an end-op cycle completes the execution.

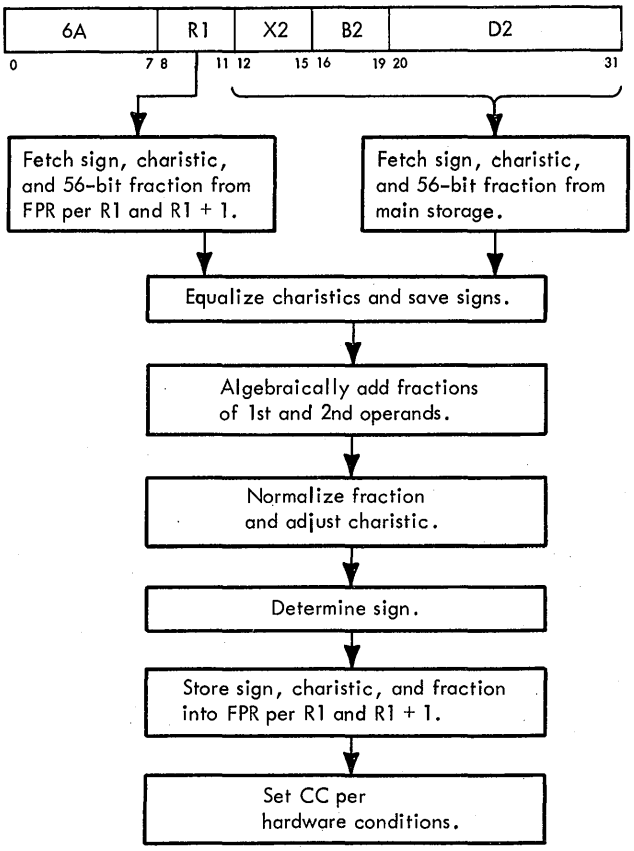
If signs are unlike, when in the normalizing loop, the intermediate result fraction can be left-shifted out of the high-order hex digit position if the intermediate fraction is 0001. Refer to the description of the AER instruction for an explanation of how this problem is handled.

All other operations that may occur during the execution of an ADR instruction are handled as described for an AER instruction, with the exception that the low-order fraction must be considered in all calculations. The major differences are:

1. An additional operand fetch is needed.
2. The low-order halves of the FPR's are used.

Add Normalized, AD (6A) – RX Long Operands

- Algebraically add 2nd operand (in storage) to 1st operand (in FPR, per R1 and R1 + 1) and place normalized sum into 1st operand location.
- RX format:



- Conditions at start of execution:
  32 bits of 1st operand are in S and T.
  Low-order fraction of 1st operand is in LS.
  Main storage request for 2nd operand has been issued per D.
  First 16 bits of instruction are in E.
- CC setting:
  Result fraction equals zero: CC = 0.
  Result fraction is less than zero: CC = 1.
  Result fraction is greater than zero: CC = 2.

The Add Normalized, AD, instruction (Diagram 5-207) algebraically adds the second operand (specified by the effective address) to the first operand (specified by R1 and R1 + 1), and places the normalized sum into the first operand location. The CC is set according to hardware conditions. The AD instruction uses 64-bit operands.

The conditions at the beginning of the execution phase are:

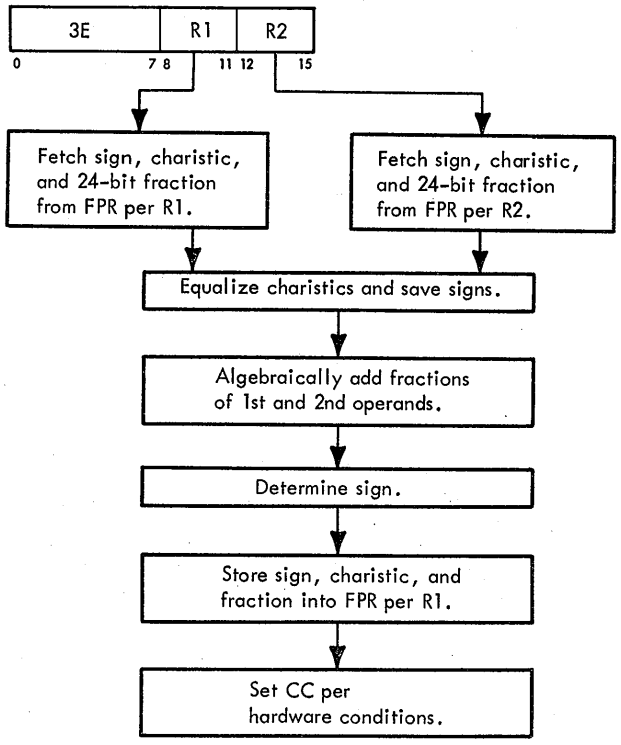
1. 32 bits of the first operand (sign, characteristic, and high-order fraction) are in S and T.
2. A main storage request for the second operand has been issued per the effective address in D.
3. The contents of A and B are unknown.
4. The first 16 bits of the instruction are in E.

Because the AD instruction uses long operands (64 bits) in the RX format, the low-order fraction of the first operand must be fetched from LS. Accordingly, the sign, characteristic, and high-order fraction of the first operand are placed into A. The low-order fraction is then fetched from LS per E(8–11) and routed to B via T and the parallel adder. The 64-bit second operand is fetched from main storage per D and placed into ST; the high-order fraction is also placed into D. The sign of the first operand is saved in STAT F and the sign of the second operand is saved in STAT C.

The first operand characteristic is then subtracted from the second operand characteristic, and the characteristic difference and the signs determine the next operation by means of a 10-way 'FLR' branch. The remainder of the execution of the AD instruction is identical to that of the ADR instruction.

Add Unnormalized, AUR (3E) – RR Short Operands

- Algebraically add 2nd operand (in FPR, per R2) to 1st operand (in FPR, per R1) and place unnormalized sum into 1st operand location.
- RR format:



- Conditions at start of execution:
  1st operand is in A, B, and D (24-bit fraction only).
  2nd operand is in S and T.
  Instruction is in E.
- CC setting:
  Result fraction equals zero: CC = 0.
  Result fraction is less than zero: CC = 1.
  Result fraction is greater than zero: CC = 2.

The Add Unnormalized, AUR, instruction (Diagram 5-206) algebraically adds the second operand (specified by R2) to the first operand (specified by R1), and places the unnormalized sum into the first operand location. The CC is set according to hardware conditions. Note that the guard digit is not examined to determine the CC setting or checked for a significance condition. Also, because the characteristic is not reduced for normalization, exponent underflow cannot occur.

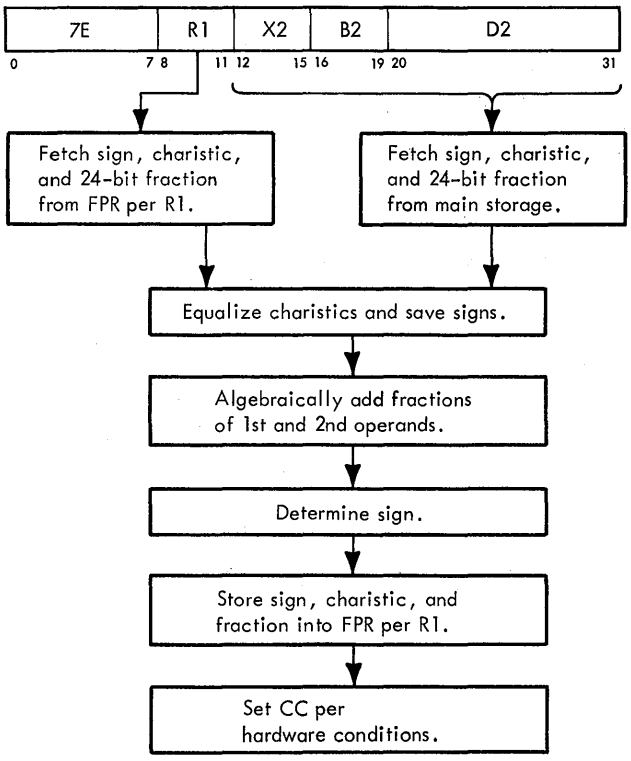
The execution of the AUR instruction is identical to that of the AER instruction until the branch on the type of instruction is performed. After the fraction is tested for overflow and shifted right (if necessary), the result is checked to see if it is zero. If the result is not zero, the result is immediately stored, along with the sign and characteristic, into the FPR specified by R1.

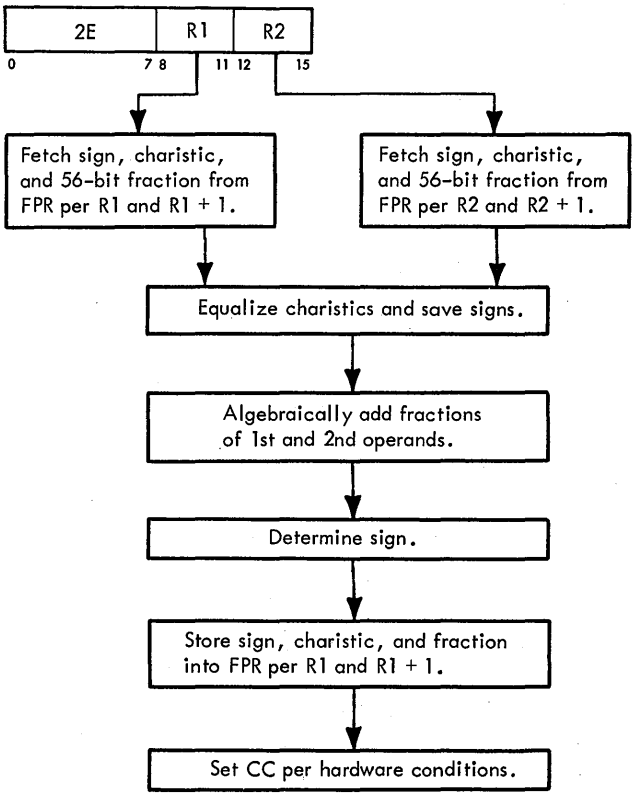
A test then determines whether equalization had occurred (STAT D set). If it did, a possibility exists that the only significant digit may be the guard digit. Therefore, if equalization did occur and no exponent overflow condition exists, the guard digit is removed from the result and the fraction is tested to see if it is zero. If the fraction equals zero, a true zero is stored and a significance program interruption is initiated. Otherwise, the instruction terminates normally.

Add Unnormalized, AU (7E) – RX Short Operands

- Algebraically add 2nd operand (in storage) to 1st operand (in FPR, per R1) and place unnormalized sum into 1st operand location.
- RX format: (See left column of next page.)
- Conditions at start of execution:
  1st operand is in S and T.
  Main storage request for 2nd operand has been issued per D.
  First 16 bits of instruction are in E.
- CC setting:
  Result fraction equals zero: CC = 0.
  Result fraction is less than zero: CC = 1.
  Result fraction is greater than zero: CC = 2.

The Add Unnormalized, AU, instruction (Diagram 5-206) algebraically adds the second operand (specified by the effective address) to the first operand (specified by R1), and places the unnormalized sum into the first operand location. Exponent underflow cannot occur. The CC is set





according to hardware conditions. The execution is identical to that of the AE instruction until the branch to determine an unnormalized, normalized, or compare instruction. From this point, the execution is identical to that of the AUR instruction.

Add Unnormalized, AWR (2E) — RR Long Operands

- Algebraically add 2nd operand (in FPR, per R2 and R2 + 1) to 1st operand (in FPR, per R1 and R1 + 1) and place unnormalized sum into 1st operand location.
- RR format: (See adjoining column.)
- Conditions at start of execution:
  32 bits of 1st operand are in A, B, and D (24-bit fraction only).
  32 bits of 2nd operand are in S and T.
  Low-order fractions of 1st and 2nd operands are in LS.
  Instruction is in E.
- CC setting:
  Result fraction equals zero: CC = 0.
  Result fraction is less than zero: CC = 1.
  Result fraction is greater than zero: CC = 2.

The Add Unnormalized, AWR, instruction (Diagram 5-207) algebraically adds the second operand (specified by R2 and R2 + 1) to the first operand (specified by R1 and R1 + 1) and places the unnormalized sum into the first operand location. Exponent underflow cannot occur.

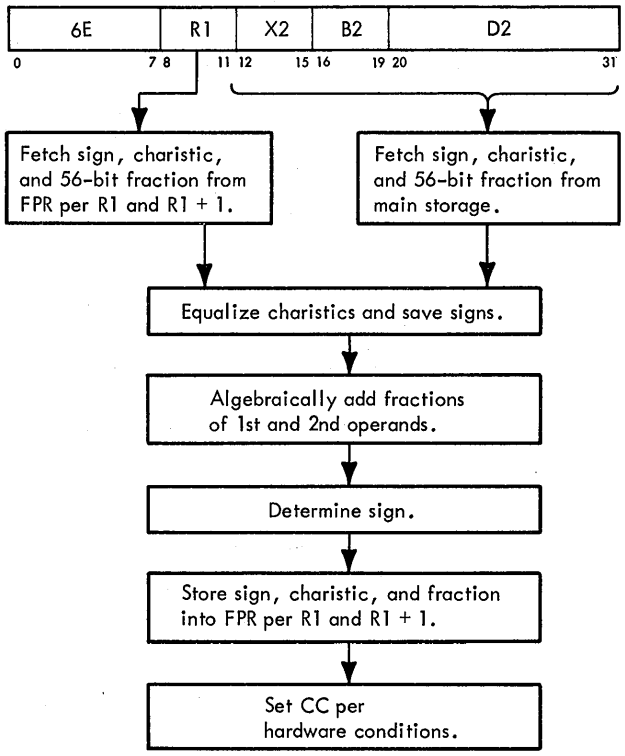
The execution phase is identical to that of the ADR instruction until the branch to determine an unnormalized, normalized, or compare instruction. Assume that the operand signs are alike and that the characteristic difference is less than 15. On Sheet 3 of Diagram 5-207, after the branch, a test determines whether the fraction of the sum overflowed (indicated by a carry into bit 7). If an overflow occurred, the fraction is shifted right one digit, and a 1 is added to the characteristic.

The low-order position of the sum (bits 32–63) is then stored into the FPR designated by R1 + 1. The result fraction is then tested to see if it equals zero. If it does not equal zero, the sign, characteristic, and high-order fraction are stored into the FPR specified by R1. If no exponent overflow occurred, the CC is set and the instruction is terminated by an end op.

Add Unnormalized, AW (6E) — RX Long Operands

- Algebraically add 2nd operand (in storage) to 1st operand (in FPR, per R1 and R1 + 1) and place unnormalized sum into 1st operand location.

- RX format:



- Conditions at start of execution:
  32 bits of 1st operand are in S and T.
  Low-order fraction of 1st operand is in LS.
  Main storage request for 2nd operand has been issued per D.
  First 16 bits of instruction are in E.

- CC setting:
  Result fraction equals zero: CC = 0.
  Result fraction is less than zero: CC = 1.
  Result fraction is greater than zero: CC = 2.

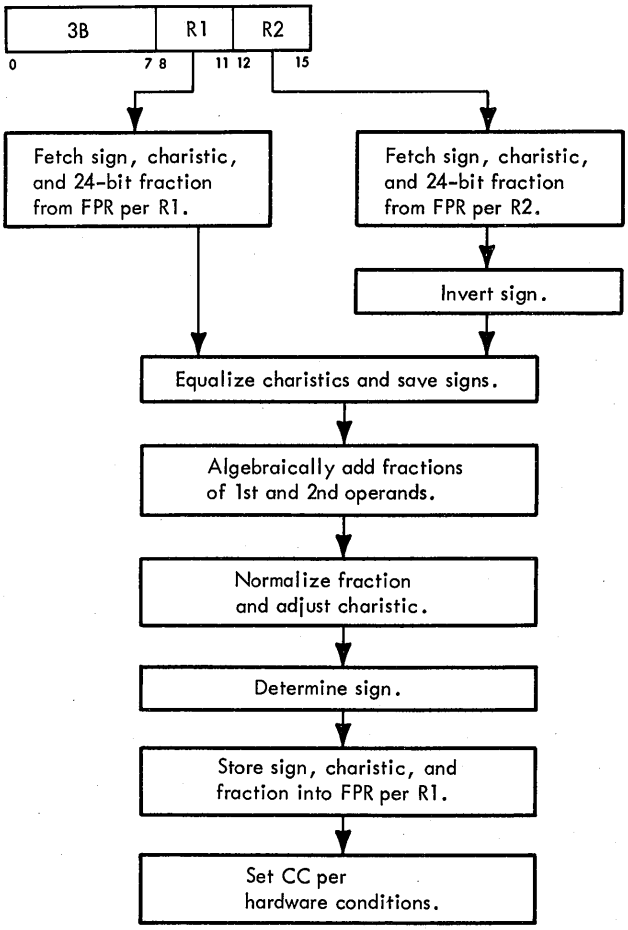
The Add Unnormalized, AW, instruction (Diagram 5-207) algebraically adds the second operand (specified by the effective address) to the first operand (specified by R1 and R1 + 1) and places the unnormalized sum into the first operand location. Exponent underflow cannot occur. The CC is set according to hardware conditions.

The execution phase is identical to that of the AD instruction until the branch to determine an unnormalized, normalized, or compare instruction. From this point, execution is identical to execution of the AWR instruction.

Subtract Normalized, SER (3B) — RR Short Operands

- Algebraically subtract 2nd operand (in FPR per R2) from 1st operand (in FPR, per R1) and place normalized difference into 1st operand location.

- RR format:



- Conditions at start of execution:
  1st operand is in A, B, and D (24-bit fraction only).
  2nd operand is in S and T.
  Instruction is in E.

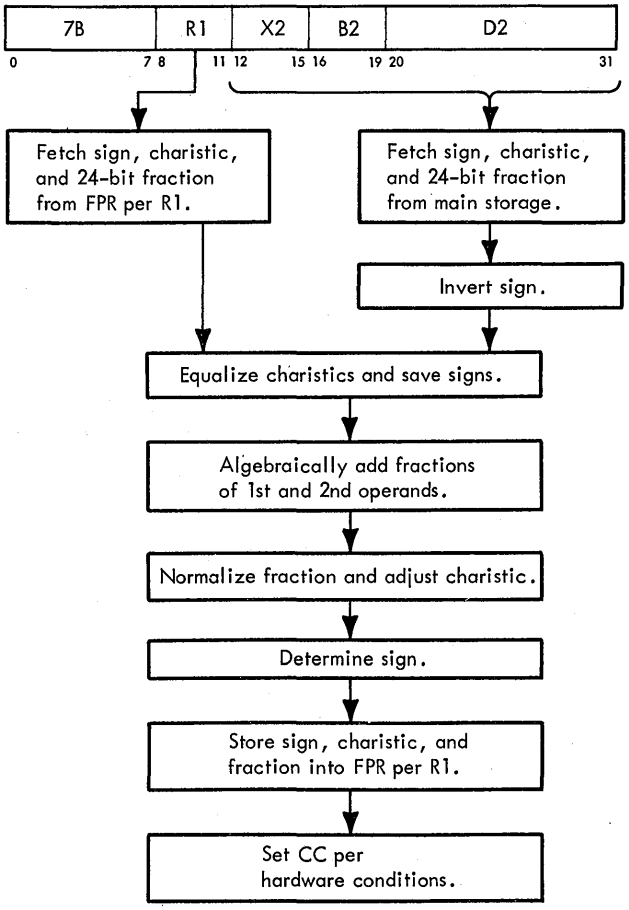
- CC setting:
  Result fraction equals zero: CC = 0.
  Result fraction is less than zero: CC = 1.
  Result fraction is greater than zero: CC = 2.

The Subtract Normalized, SER, instruction (Diagram 5-206) algebraically subtracts the second operand (specified by R2) from the first operand (specified by R1), and places the normalized difference into the first operand location. The second operand location remains unchanged. The CC is set according to hardware conditions.

For subtract instructions, the sign of the second operand is complemented, after which the algebraic subtraction is treated as an algebraic addition. Therefore, execution of the SER instruction is identical to that of the AER instruction except that the complement of the second operand sign rather than the true sign is gated to STAT C.

Subtract Normalized, SE (7B) — RX Short Operands

- Algebraically subtract 2nd operand (in storage) from 1st operand (in FPR, per R1) and place normalized difference into 1st operand location.
- RX format:



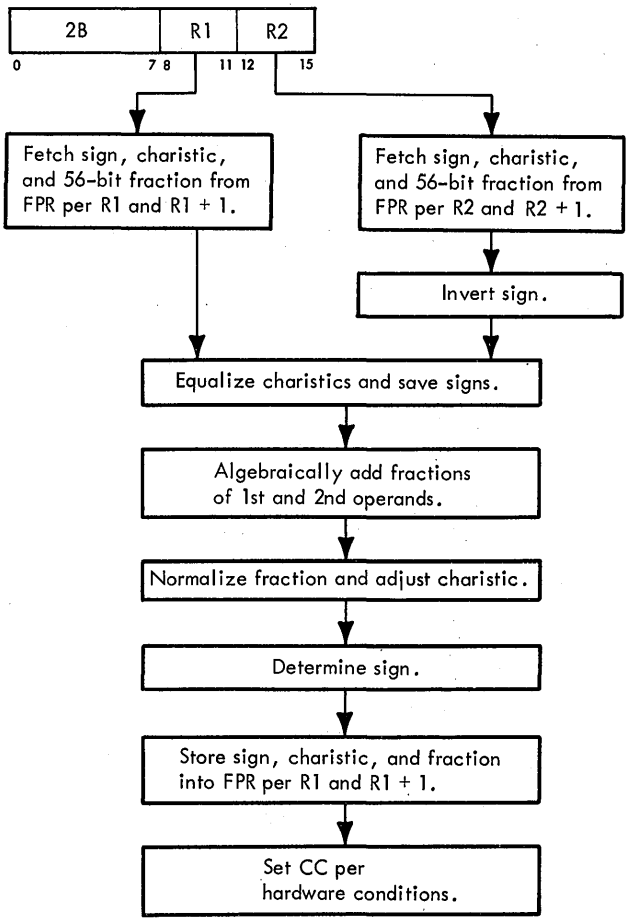
- Conditions at start of execution:
  1st operand is in S and T.
  Main storage request for 2nd operand has been issued per D.
  First 16 bits of instruction are in E.
- CC setting:
  Result fraction equals zero: CC = 0.
  Result fraction is less than zero: CC = 1.
  Result fraction is greater than zero: CC = 2.

The Subtract Normalized, SE, instruction (Diagram 5-206) algebraically subtracts the second operand (specified by the effective address) from the first operand (specified by R1) and places the normalized difference into the first operand location. The CC is set according to hardware conditions.

Execution of the SE instruction is identical to that of the AE instruction except that the complement of the second operand sign instead of the true sign is gated to STAT C.

Subtract Normalized, SDR (2B) — RR Long Operands

- Algebraically subtract 2nd operand (in FPR, per R2 and R2 + 1) from 1st operand (in FPR, per R1 and R1 + 1) and place normalized difference into 1st operand location.
- RR format:



- Conditions at start of execution:
  32 bits of 1st operand are in A, B, and D (24-bit fraction only).
  32 bits of 2nd operand are in S and T.
  Low-order fractions of 1st and 2nd operands are in LS.
  Instruction is in E.

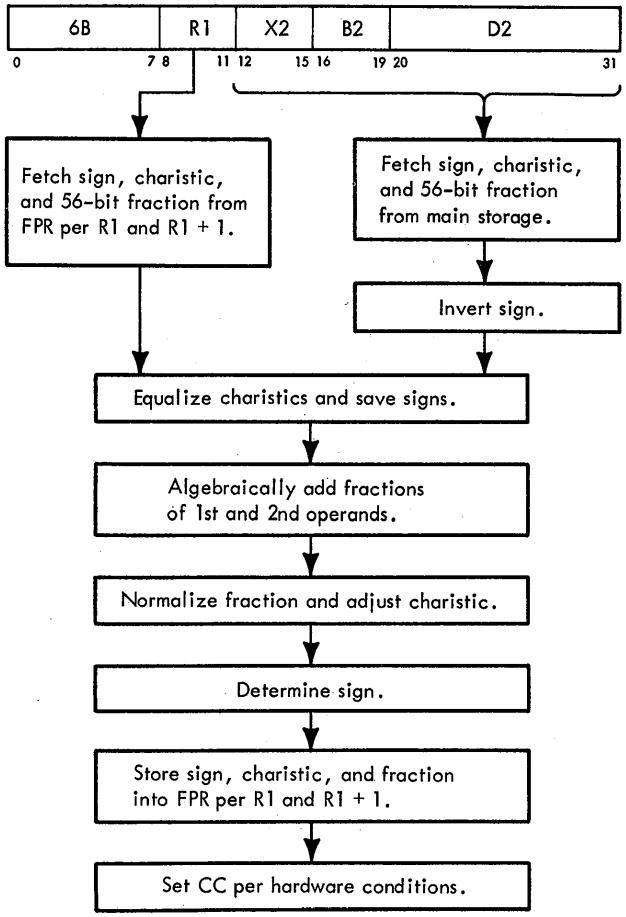
- CC setting:
  Result fraction equals zero: CC = 0.
  Result fraction is less than zero: CC = 1.
  Result fraction is greater than zero: CC = 2.

The Subtract Normalized, SDR, instruction (Diagram 5-207) algebraically subtracts the second operand (specified by R2 and R2 + 1) from the first operand (specified by R1 and R1 + 1) and places the normalized difference into the first operand location. The CC is set according to hardware conditions.

Execution of the SDR instruction is identical to that of the ADR instruction except that the complement of the second operand sign instead of the true sign is gated to STAT C.

Subtract Normalized, SD (6B) — RX Long Operands

- Algebraically subtract 2nd operand (in storage) from 1st operand (in FPR, per R1 and R1 + 1) and place normalized difference into 1st operand location.
- RX format:



- Conditions at start of execution:
  32 bits of 1st operand are in S and T.
  Low-order fraction of 1st operand is in LS.
  Main storage request for 2nd operand has been issued per D.
  First 16 bits of instruction are in E.

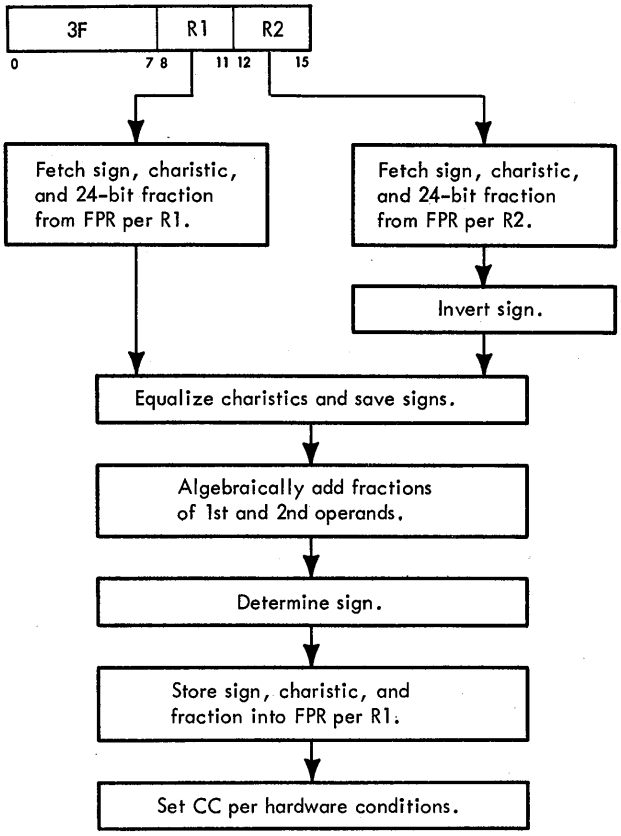
- CC setting:
  Result fraction equals zero: CC = 0.
  Result fraction is less than zero: CC = 1.
  Result fraction is greater than zero: CC = 2.

The Subtract Normalized, SD, instruction (Diagram 5-207) algebraically subtracts the second operand (specified by the effective address) from the first operand (specified by R1 and R1 + 1) and places the normalized difference into the first operand location. The CC is set according to hardware conditions.

Execution of the SD instruction is identical to the execution of the AD instruction except that the complement of the second operand sign instead of the true sign is gated to STAT C.

Subtract Unnormalized, SUR (3F) — RR Short Operands

- Algebraically subtract 2nd operand (in FPR, per R2) from 1st operand (in FPR, per R1) and place unnormalized difference into 1st operand location.
- RR format:



- Conditions at start of execution:
  1st operand is in A, B, and D (24-bit fraction only).
  2nd operand is in S and T.
  Instruction is in E.
- CC setting:
  Result fraction equals zero: CC = 0.
  Result fraction is less than zero: CC = 1.
  Result fraction is greater than zero: CC = 2.

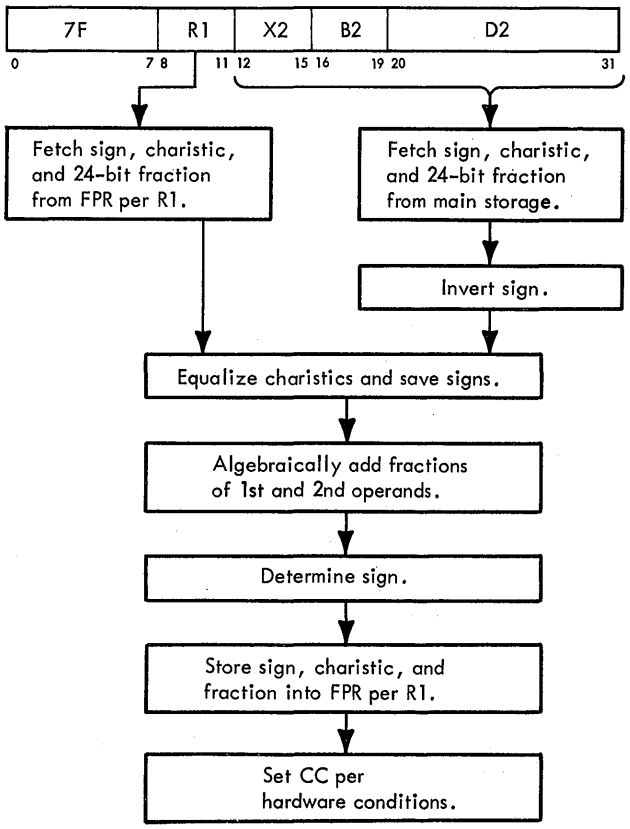
The Subtract Unnormalized, SUR, instruction (Diagram 5-206) algebraically subtracts the second operand (specified by R2) from the first operand (specified by R1) and places the unnormalized difference into the first operand location. The CC is set according to hardware conditions.

Except that the intermediate results of the Subtract Unnormalized instructions are not normalized, the operation is the same as that of the Subtract Normalized instructions. That is, the second operand sign is inverted (and saved in STAT C) and the fractions are algebraically added.

The SUR instruction is executed in the same manner as the AUR instruction except that the complement of the second operand sign instead of the true sign is gated to STAT C. Note that, when executing Subtract Unnormalized instructions, the guard digit is not examined to determine the CC setting or checked for a significance condition. Also, as in Unnormalized Add instructions, exponent underflow cannot occur.

Subtract Unnormalized, SU (7F) — RX Short Operands

- Algebraically subtract 2nd operand (in storage) from 1st operand (in FPR, per R1) and place unnormalized difference into 1st operand location.
- RX format:



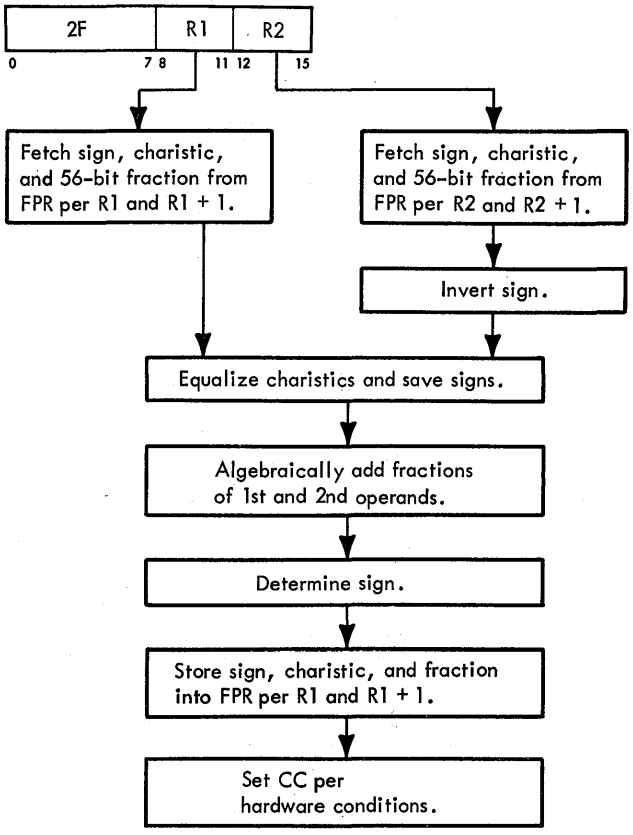
- Conditions at start of execution:
  1st operand is in S and T.
  Main storage request for 2nd operand has been issued per D.
  First 16 bits of instruction are in E.
- CC setting:
  Result fraction equals zero: CC = 0.
  Result fraction is less than zero: CC = 1.
  Result fraction is greater than zero: CC = 2.

The Subtract Unnormalized, SU, instruction (Diagram 5-206) algebraically subtracts the second operand (specified by the effective address) from the first operand (specified by R1) and places the unnormalized difference into the first operand location. The CC is set according to hardware conditions.

Execution of the SU instruction is identical to that of the AU instruction except that the complement of the second operand sign instead of the true sign is gated to STAT C.

Subtract Unnormalized, SWR (2F) — RR Long Operands

- Algebraically subtract 2nd operand (in FPR, per R2 and R2 + 1) from 1st operand (in FPR, per R1 and R1 + 1) and place unnormalized difference into 1st operand location.
- RR format:



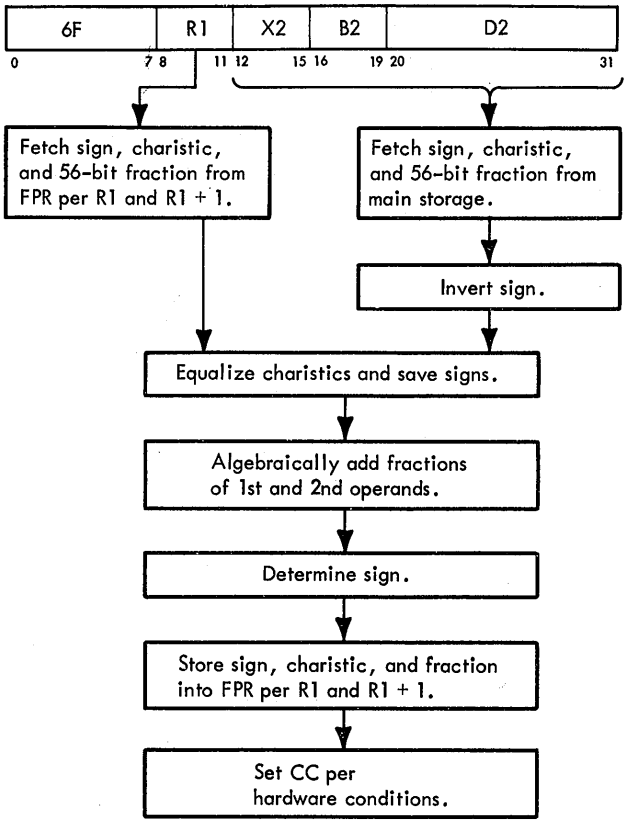
- Conditions at start of execution:
  32 bits of 1st operand are in A, B, and D (24-bit fraction only).
  32 bits of 2nd operand are in S and T.
  Low-order fractions of 1st and 2nd operands are in LS.
  Instruction is in E.
- CC setting:
  Result fraction equals zero: CC = 0.
  Result fraction is less than zero: CC = 1.
  Result fraction is greater than zero: CC = 2.

The Subtract Unnormalized, SWR, instruction (Diagram 5-207) algebraically subtracts the second operand (specified by R2 and R2 + 1) from the first operand (specified by R1 and R1 + 1) and places the unnormalized difference into the first operand location. The CC is set according to hardware conditions.

Execution of the SWR instruction is identical to that of the AWR instruction except that the complement of the second operand sign instead of the true sign is gated to STAT C.

Subtract Unnormalized, SW (6F) — RX Long Operands

- Algebraically subtract 2nd operand (in storage) from 1st operand (in FPR, per R1 and R1 + 1) and place unnormalized difference into 1st operand location.
- RX format:



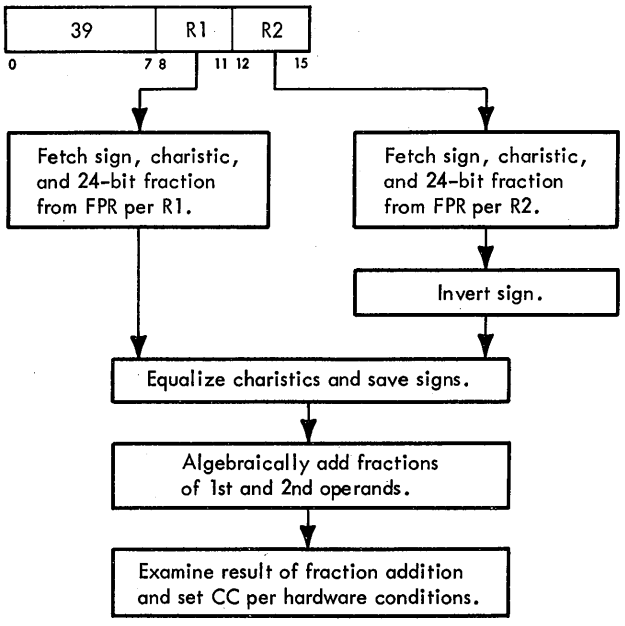
- Conditions at start of execution:
  32 bits of 1st operand are in S and T.
  Low-order fraction of 1st operand is in LS.
  Main storage request for 2nd operand has been issued per D.
  First 16 bits of instruction are in E.
- CC setting:
  Result fraction equals zero: CC = 0.
  Result fraction is less than zero: CC = 1.
  Result fraction is greater than zero: CC = 2.

The Subtract Unnormalized, SW, instruction (Diagram 5-207) algebraically subtracts the second operand (specified by the effective address) from the first operand (specified by R1 and R1 + 1) and places the unnormalized difference into the first operand location. The CC is set according to hardware conditions.

Execution of the SW instruction is identical to that of the AW instruction except that the complement of the second operand sign instead of the true sign is gated to STAT C.

Compare, CER (39) — RR Short Operands

- Algebraically compare 1st operand (in FPR, per R1) with 2nd operand (in FPR, per R2); CC indicates result.
- RR format:



- Conditions at start of execution:
  1st operand is in A, B, and D (24-bit fraction only).
  2nd operand is in S and T.
  Instruction is in E.

- CC setting:
  Operands are equal: CC = 0.
  1st operand is less than 2nd operand: CC = 1.
  1st operand is greater than 2nd operand: CC = 2.

The Compare, CER, instruction (Diagram 5-206) algebraically compares the first operand (specified by R1) with the second operand (specified by R2); the CC indicates that the first operand is equal to, less than, or greater than the second operand.

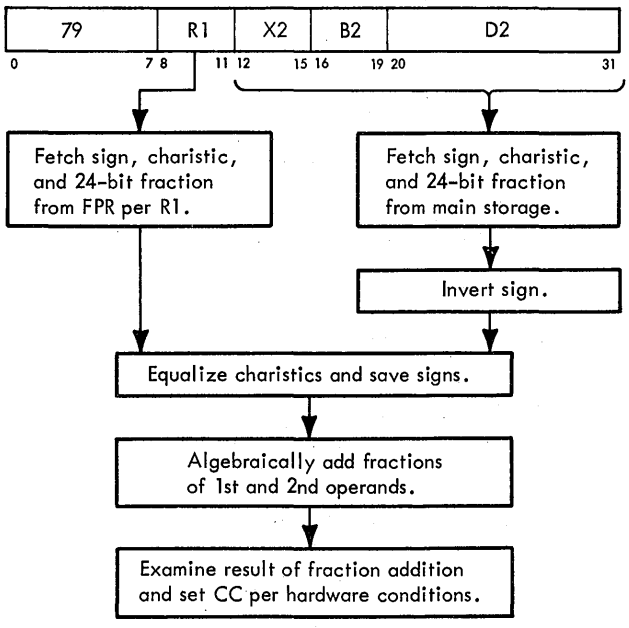
Comparison is algebraic, taking into account the sign, fraction, and characteristic of each operand. An exponent inequality is not decisive for magnitude determination because the fractions may have different numbers of leading zeros. Equality is established by following the rules for floating-point subtraction. The intermediate result is not normalized or stored. When the intermediate result, including a possible guard digit, is zero, the operands are equal. Numbers with zero fractions compare equal even when they differ in sign or characteristic. Exponent overflow, exponent underflow, or significance check cannot occur. The CC is set per hardware conditions at end-op time.

In the CER instruction, the contents of the low-order halves of the FPR's are ignored. (Neither operand location is changed as a result of any compare instructions.)

Execution of the CER instruction is identical to that of the SER instruction until the branch to determine an unnormalized, normalized, or compare instruction. At that point, the CC is set per hardware conditions, and the instruction is terminated by an end op.

Compare, CE (79) — RX Short Operands

- Algebraically compare 1st operand (in FPR, per R1) with 2nd operand (in storage); CC indicates result.
- RX format:



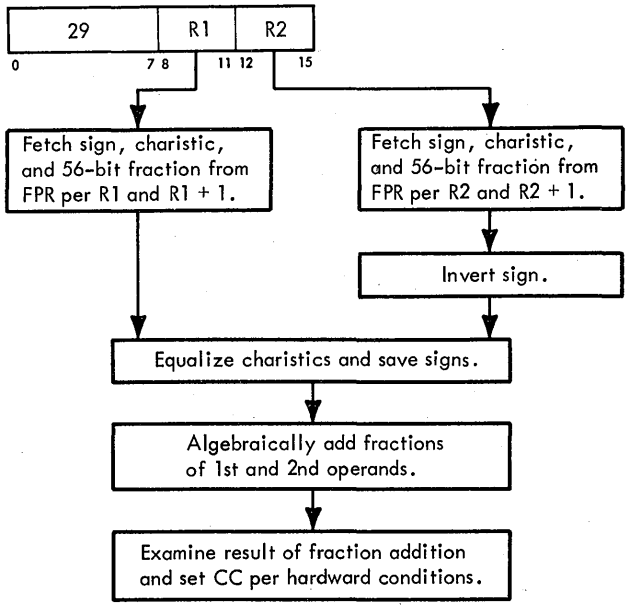
- Conditions at start of execution:
  1st operand is in S and T.
  Main storage request for 2nd operand has been issued per D.
  1st 16 bits of instruction are in E.
- CC setting:
  Operands are equal: CC = 0.
  1st operand is less than 2nd operand: CC = 1.
  1st operand is greater than 2nd operand: CC = 2.

The Compare, CE, instruction (Diagram 5-206) algebraically compares the first operand (specified by R1) with the second operand (specified by the effective address); the CC indicates the result. Exponent underflow, exponent overflow, or significance check cannot occur.

Execution of the CE instruction is identical to that of the SE instruction until the branch to determine an unnormalized, normalized, or compare instruction. At that point, the CC is set per hardware conditions, and the instruction is terminated by an end op.

Compare, CDR (29) — RR Long Operands

- Algebraically compare 1st operand (in FPR, per R1 and R1 + 1) with 2nd operand (in FPR, per R2 and R2 + 1); CC indicates result.
- RR format:



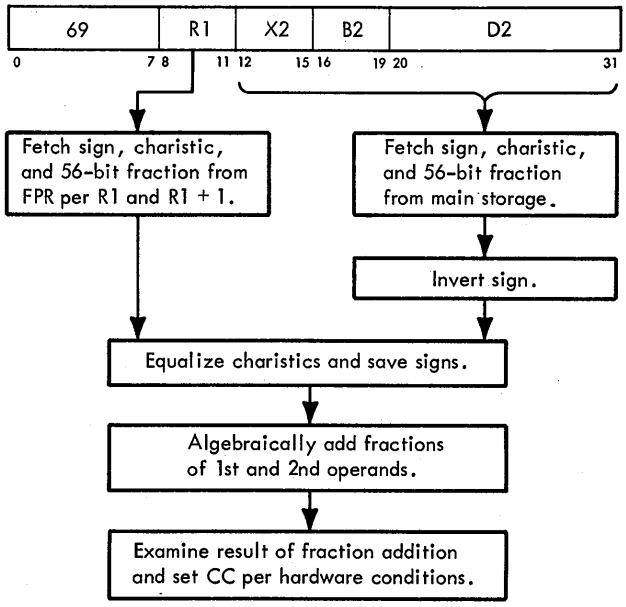
- Conditions at start of execution:
  32 bits of 1st operand are in A, B, and D (24-bit fraction only).
  32 bits of 2nd operand are in S and T.
  Low-order fractions of 1st and 2nd operands are in LS.
  Instruction is in E.
- CC setting:
  Operands are equal: CC = 0.
  1st operand is less than 2nd operand: CC = 1.
  1st operand is greater than 2nd operand: CC = 2.

The Compare, CDR, instruction (Diagram 5-207) algebraically compares the first operand (specified by R1 and R1 + 1) with the second operand (specified by R2 and R2 + 1); the CC indicates the result. Exponent underflow, exponent overflow, or significance check cannot occur.

Execution of the CDR instruction is identical to that of the SDR instruction until the branch to determine an unnormalized, normalized, or compare instruction. At that point, the CC is set per hardware conditions, and the instruction is terminated by an end op.

Compare, CD (69) – RX Long Operands

- Algebraically compare 1st operand (in FPR, per R1 and R1 + 1) with 2nd operand (in storage); CC indicates result.
- RX format:



- Conditions at start of execution:
  32 bits of 1st operand are in S and T.
  Low-order fraction of 1st operand is in LS.
  Main storage request for 2nd operand has been issued per D.
  First 16 bits of instruction are in E.
- CC setting:
  Operands are equal: CC = 0.
  1st operand is less than 2nd operand: CC = 1.
  1st operand is greater than 2nd operand: CC = 2.

The Compare, CD, instruction (Diagram 5-207) algebraically compares the first operand (specified by R1 and R1 + 1) with the second operand (specified by the effective address); the CC indicates the result. Exponent underflow, exponent overflow, or significance check cannot occur.

Execution of the CD instruction is identical to that of the SD instruction until the branch to determine an unnormalized, normalized, or compare instruction. At that point, the CC is set per hardware conditions, and the instruction is terminated by an end op.

HALVE

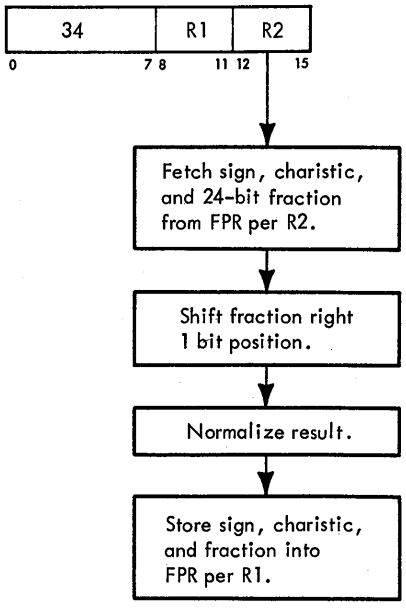
The Halve instructions (HER and HDR) divide the second operand by 2 and place the normalized quotient into the first operand location. The Halve instructions are in the RR format with short and long operand options available. In the HER instruction, the low-order half of the result register remains unchanged.

After the second operand is in ST, the sign and the characteristic are saved in F, and the high-order fraction (24 bits) is placed into D (long operands). Shifting the fraction one bit position to the right divides the operand by 2. Because the data in PAL cannot be shifted right 1 directly, two machine cycles are necessary. First the fraction is shifted left 1 from D to the parallel adder and then shifted right 4 to PAL, thus yielding an effective right 3 shift. Next, the fraction is placed into AB. A left 2 shift occurs when the fraction is routed to DT via the parallel adder, thus resulting in a right 1 shift and thereby dividing the fraction by 2. Guard digits are saved and used if normalization is necessary. After the fraction is normalized, the sign, characteristic, and fraction are stored into LS per R1, completing instruction execution.

The halve operation differs from the divide operation in that 2 is the only divisor.

Halve, HER (34) – RR Short Operands

- Divide 2nd operand (in FPR, per R2) by 2 and place normalized quotient into 1st operand location (in FPR, per R1).
- RR format:



- Conditions at start of execution:
  1st operand is in A, B, and D (24-bit fraction only).
  2nd operand is in S and T.
  Instruction is in E.

The Halve, HER, instruction (Diagram 5-208, FEMDM) divides the second operand (specified by R2) by 2 and places the normalized quotient into the FPR specified by R1. To divide by 2, the fraction is shifted right one bit position.

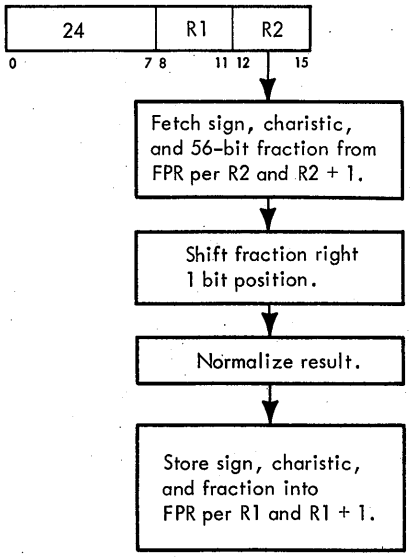
At the start of execution, the second operand is in S and T. After a specification test, the fraction is placed into D, the sign is saved in STAT C, and the characteristic is saved in F. The fraction in D is then shifted right one bit position. A test is initiated to determine whether the fraction is zero and normalized. If the fraction equals zero, the FPR specified by R1 is set to a true zero and the operation terminates with an end op. If the fraction contains leading zero's, it is normalized by shifting left one hex digit at a time until a significant hex digit appears in the high-order position. A 1 is subtracted from the characteristic for each shift.

After normalization, the characteristic and fraction are gated to T from F and D, respectively. The result is then stored into the FPR specified by R1. The sign position is set by forcing the contents of STAT C onto the LS bus as the fraction and characteristic are being stored.

The characteristic is then tested for a possible exponent underflow. If no underflow occurred, the instruction is terminated. If an underflow did occur, the remainder of the operation is determined by the state of the underflow mask bit.

Halve, HDR (24) – RR Long Operands

- Divide 2nd operand (in FPR, per R2 and R2 + 1) by 2 and place normalized quotient into 1st operand location (in FPR, per R1 and R1 + 1).
- RR format:



- Conditions at start of execution:
  32 bits of 1st operand are in A, B, and D (24-bit fraction only).
  32 bits of 2nd operand are in S and T.
  Low-order fractions of 1st and 2nd operands are in LS.
  Instruction is in E.

The Halve, HDR, instruction (Diagram 5-209, FEMDM) divides the second operand, (specified by R2 and R2 + 1) by 2 and places the normalized quotient into the FPR specified by R1 and R1 + 1. To divide by 2, the fraction is shifted right one bit position.

At the start of the execution phase, the sign, characteristic, and high-order fraction are in S and T. After a specification test, the high-order fraction is placed into D and the low-order fraction is fetched from LS per R1 + 1 and gated to T. The sign is saved in STAT C, and the characteristic is saved in F. The fraction in DT is then shifted right one bit position, and a test determines whether the fraction is equal to zero and normalized. If the fraction equals zero, the FPR specified by R1 and R1 + 1 is set to a true zero and the operation terminates with an end op. If the fraction contains leading zero's, it is normalized by shifting left one hex digit at a time until a significant hex digit appears in the high-order position. A 1 is subtracted from the characteristic for each shift.

After normalization, the low-order fraction is stored into the FPR specified by R1 + 1. The characteristic and high-order fraction are then gated to T from F and D, respectively. The result is stored into the FPR specified by R1. The sign position is set by forcing the contents of STAT C onto the LS bus as the fraction and characteristic are being stored.

The characteristic is then tested for a possible exponent underflow. If no underflow occurred, the instruction is terminated. If an underflow did occur, the remainder of the operation is determined by the state of the underflow mask bit.

MULTIPLY

- Multiplies 1st operand and 2nd operand and places normalized product into 1st operand location.
- *Note:* In 2065 floating-point multiply operations, roles of 1st and 2nd operands are reversed from roles defined in *System/360 Principles of Operation*, SRL, Form A22-6821-6. That is, 2nd operand is multiplicand and 1st operand is multiplier. (Interchanging operand roles does not affect product.) Result, however, still replaces 1st operand.
- Product is 64 bits for both short and long operand instructions.
- Characteristics are added and 64 is subtracted to obtain intermediate characteristic.
- Operands are prenormalized before multiplying.

- Product is normalized before storing.
- Sign of product is determined algebraically.

Two floating-point numbers are multiplied by adding their characteristics and multiplying their fractions. For example, if 1250 is to be multiplied by 5, converting these numbers to hex notation yields the equation

$$(.4E2_{16} \times 16^3) \times (.5 \times 16^1).$$

The product is obtained, as follows:

$$\begin{array}{rl} & .4E2_{16} \times 16^3 \\ \times & \quad .5_{16} \times 16^1 \\ \hline \text{Product of fractions} \longrightarrow & .186A_{16} \times 16^4 \leftarrow \text{Exponents are added} \end{array}$$

The product ($.186A_{16} \times 16^4$) equals 6250 (decimal), which is the product of 1250 x 5.

When two floating-point numbers are multiplied, the characteristics are added to yield the final characteristic value of the product, as shown above. Because excess-64 notation is used, 64 must be subtracted from the characteristic sum because the characteristic value is in excess-128 [(C1 + 64) + (C2 + 64) = C1 + C2 + 128] after characteristic addition. When 64 is subtracted, the result is returned to excess-64 notation (C1 + C2 + 128 - 64 = C1 + C2 + 64). For instance, in floating-point format, the exponents used in the example above yield characteristics of 67 (3 + 64) and 65 (1 + 64). The sum of these characteristics is 132. Subtracting 64 from 132 leaves 68, which is equivalent to an exponent of 4 in excess-64 notation.

If one or both operand fractions contain leading zeros, the unnormalized operand(s) is prenormalized. That is, the operands are normalized before multiplication begins. Prenormalization increases product precision. By prenormalizing the operands, a maximum of one postnormalization cycle is necessary. Postnormalization is the process of normalizing the product after fraction multiplication. Prenormalization and postnormalization are accomplished by shifting the fraction left one hex digit and subtracting 1 from the characteristic value for each left shift until a significant hex digit appears in the high-order position of the fraction.

The product for both short and long operand multiply instructions is 64 bits long. Note that, if the fraction is not prenormalized, dropping the low-order bits of the product in excess of 64 may result in a false zero product without prenormalization. This result would occur often in long operand instructions because 56 low-order bits of the product are lost when executing the multiply algorithm. Thus, to prevent a false zero, the product for long operand instructions would have to be 120 bits long. A false zero is prevented, however, by prenormalizing the operands and postnormalizing the intermediate product. During postnormalization, the intermediate product characteristic is reduced by the number of left shifts. For long operands, the low-order bits of the intermediate product are dropped before left-shifting. For short operands (six-digit fractions), the product fraction has the full 14 hex digits of the long format, and the two low-order hex fraction digits are accordingly always zeros. The two low-order hex fraction digits are zeros in short operand instructions because a maximum of 12 nonzero hex digits is possible when multiplying two six-digit numbers. In multiplication, the number of digits in the product cannot exceed the sum of the available operand digits. Therefore, because 14 product digits are available, the two low-order hex digits of short operand products are always zeros.

The sign of the product is determined algebraically; that is, if the signs of the operands are alike (both plus or both minus), the product is assigned a plus sign; if the signs are unlike, the product is made negative.

Exponent overflow occurs if the final product characteristic exceeds 127. The operation is terminated, and a program interruption occurs. The overflow interruption condition does not occur for a partial product characteristic exceeding 127 when the final characteristic is brought within range through normalization.

When exponent underflow occurs, the final product characteristic is less than zero. The sign, characteristic, and fraction are made zero, and a program interruption occurs if the corresponding mask bit is a 1. Underflow is not signalled when the characteristic of an operand becomes less than zero during prenormalization, and the correct characteristic and fraction value are used in the multiplication.

When all 14 result fraction digits are zero, the product sign and characteristic are made zero, yielding a true zero result, exponent underflow is not signalled, and no interruption is taken. The program interruption for significance is never taken for multiplication.

Data Flow and Algorithm

- See Note under "Multiply".
- Signs are saved in STAT's C and F.
- Characteristics are added in serial adder; carry is saved in STAT D.
- Fraction multiplication is performed by multiply/divide logic.
- E(12–15) selects two multiplier bits from S.
- S bits determine multiple value to be added to partial product.

The data paths of the floating-point operands during multiplication are shown in Diagram 5-210, FEMDM. The characteristic is computed in the serial adder (A of the

diagram). The signs are saved in STAT C and STAT F. To add the characteristics, the first operand characteristic is gated to SAA(1–7) from AB per the ABC, and the second is gated to SAB(1–7) from ST per the STC. The characteristic sum is routed to F and the characteristic carry is saved in STAT D and F(0). 64 is subtracted from the characteristic by adding the 2's complement of 64 to the sum in F.

Note: For an RX instruction with a normalized first operand, the first operand characteristic and sign are in S or T and the second operand characteristic and sign are in A.

Fraction multiplication is performed by the multiply/divide logic (B of Diagram 5-210) and is similar to fixed-point multiplication (Section 2 of this Chapter) except that the operands are shorter. After the signs are saved, the characteristic is determined and the operands are prenormalized; the multiplicand is in DT, the multiplier is in S, and a count, representing the number of repetitive operations necessary to perform the multiplication, is in E(12–15). Multiplication is performed two bits at a time; that is, the multiplicand in DT is multiplied by two bits of the multiplier in S using the multiple-selection decoder and the parallel adder. The count in E(12–15), in addition to keeping track of the number of operations, determines which multiplier bits are to be used, starting with the low-order bits and moving two bits to the left for each multiplication. The result of each two-bit multiplication is a multiple of the multiplicand which is then added to the partial product formed by previous two-bit multiplications. Thus, a new partial product is obtained. The two-bit multiplications continue until E(12–15) indicates that all bits of the multiplier have been used. At that time, the intermediate product is contained in AB(4–67).

The steps of the fraction multiplication are:

1. Place the constant F (hex) into E(12–15).
2. Using the value in E(12–15), select two multiplier bits (M1 and M2) from S. E(12,13) selects the byte and E(14,15) selects the two bits within a byte (see B of Diagram 5-210). For example, with the original value in E(12–15), E(12,13) = 11 selecting the third byte in S (bits 24–31), and E(14,15) = 11 selecting bits 6 and 7 within that byte. Thus, the first M1, M2 values used are S(30,31). Table 3-10 shows which S bits are selected for all values of E(12–15).
3. The value of the M1, M2 bits, in conjunction with the 'TX' trigger, gates the correct multiple of the multiplicand to the PAA. Considering the M1, M2 bits as a two-bit multiplier, the multiplicand in DT can be multiplied by 0 (M1, M2 = 00), by 1 (M1, M2 = 01), by 2 (M1, M2 = 10), or by 3 (M1, M2 = 11), as follows:
   a. M1, M2 = 00 and 'TX' Trigger Is Reset: Because zero times the multiplicand is zero, nothing is added to the partial product. However, the partial product is shifted right two bit positions.
   b. M1, M2 = 01 and 'TX' Trigger Is Reset: One times the multiplicand equals the multiplicand. Thus, the multiplicand is added to the partial product, and the partial product is shifted right two bit positions.
   c. M1, M2 = 10 and 'TX' Trigger Is Reset: The multiplicand is multiplied by 2 by shifting it left one bit position. The result is then added to the partial product, and the partial product is shifted right two bit positions.
   d. M1, M2 = 11 and 'TX' Trigger Is Reset: Because the facilities for multiplying the multiplicand by 3 are not directly available, an effective multiplication by 3 is accomplished by multiplying by 4 and subtracting one times the multiplicand (4X - 1X = 3X). The minus 1X occurs first by adding the 2's complement of the multiplicand to the partial product, and 4X occurs by adding 1 to the next two multiplier bits. The 'TX' trigger is set to remember that an additional 1 is required in the next cycle. After adding the 2's complement of the multiplicand to the partial product, the partial product is shifted right two bit positions.
   e. M1, M2 = 00 and 'TX' Trigger Is Set: One times the multiplicand is added to the partial product, which is then shifted right two bit positions.
   f. M1, M2 = 01 and 'TX' Trigger Is Set: Two times (left 1 shift) the multiplicand is added to the partial product, which is then shifted right two bit positions.
   g. M1, M2 = 10 and 'TX' Trigger Is Set: The 2's complement of the multiplicand is added to the partial product, and the 'TX' trigger is again set. The resulting partial product is shifted right two bit positions.
   h. M1, M2 = 11 and 'TX' Trigger is Set: Zero is added to the partial product, and the 'TX' trigger is set. The partial product is shifted right two bit positions.

Note that in each case the new partial product is shifted right two bit positions. Thus, each higher-order

Table 3-10. Multiplier Bits Selected, Floating-Point Multiply

	E(14,15)							
	00		01		10		11	
E(12,13)	M1	M2	M1	M2	M1	M2	M1	M2
00	0	1	2	3	4	5	6	7
01	8	9	10	11	12	13	14	15
10	16	17	18	19	20	21	22	23
11	24	25	26	27	28	29	30	31

multiplicand multiple is displaced two positions to the left when added to the partial product. This right 2 shift is accomplished by shifting the partial product right 4 as it enters PAL, and shifting it left 2 when it is gated to PAB for addition to the next multiplicand multiple. The two low-order bits [B(66,67)] are shifted out, and therefore lost on each cycle.

Note: Steps 2 and 3 are performed by the 'SELMPL*E3' micro-order and hardware conditions.

4. Reduce E(12–15) by 1.
5. Determine, by the count in E(12–15), whether all bits of the multiplier fraction have been used to select a multiplicand multiple. If not, repeat steps 2–4; if they have all been used, proceed to step 6.
6. After decoding the last S bits, the 'TX' trigger is checked. If the 'TX' trigger is set, one additional termination cycle is necessary to obtain the final intermediate product. If the 'TX' trigger is reset, no extra cycle is necessary. After the fraction intermediate product is obtained, the fraction is normalized (post-normalization), the characteristic is adjusted, the sign is determined, and the final 64-bit product is stored into the FPR specified by R1 and R1 + 1 [located in E(8–11)].

To illustrate the multiply operation, assume that the following fractions are to be multiplied:

$$0.24_{10} \times 0.15_{10} = 0.0360_{10} \text{ or } 0.18_{16} \times .F_{16} = 0.168_{16}$$

The operands in machine language become:

0 1000000. 0001 1000 0↔0 X 0 1000000. 1111 0000 0↔0

In hex notation, the example becomes:

+40.18 X +40.F = +40.168

Further, assume that a short operand instruction in the RR format is to be executed. For this discussion, assume that 0.15 (decimal) is the multiplier (first operand) and 0.24 (decimal) is the multiplicand (second operand). At the start of execution, the instruction is in E; the first operand is in A, B, and D (this value is not used and is subsequently destroyed), and the second operand is in S and T.

The signs are saved, the characteristics are determined, and the fractions are prenormalized before beginning the multiply algorithm. The value in E(12–15) is set to 15 and sequentially reduced by 1 during the operation. Before fraction multiplication begins, the first operand fraction is transferred to S, the second operand is transferred to D, and B and T are reset.

The first multiple of the multiplicand is determined by checking E(12–15), which initially contains 1111 (binary). Using Table 3-10 to determine the M1 and M2 bits, the first bits selected are S(30,31). At this time, S(30,31) = 00. Thus, the first partial product placed into AB equals zero. The sequential reduction of E(12–15) continues until the value equals 0101, at which time the partial product in AB equals zero.

Referring to Table 3-10, when E(12–15) = 0101, S(10,11) is selected. These selected bits determine the multiple (M1, M2) of the multiplicand to be added to the partial product in AB. Because S(10,11) = 11 (binary), the 2's complement of DT is gated to PAA. The contents of AB are shifted left 2 at this time (AB equals zero) and gated to PAB. The output of the parallel adder is shifted right 4 to the PAL's. The contents of the PAL's are gated to AB, forming a new partial product. AB(4–67) now contains 1111.1111 1110 1000 0↔0. PA(4) is propagated into PAL(4–7) by the 'R→' micro-order. Because S(10,11) = 11, the 'TX' trigger is set (Table 3-11). Because E(12-15) is decremented after each multiple selection, zeros are added to PAA on the next multiple selection (0 X DT), as shown in Table 3-11. During this select multiple, the contents of AB are shifted left 2 to PAB, and the next partial product is shifted right 4 to the PAL's and AB(4–67), thus yielding an effective right 2 shift. The new partial product in AB(4–67) becomes 1111.1111 1111 1010 0↔0.

Table 3-11. Value of Multiple Determined by Multiple Selection Bits (Floating-Point)

Multiple Selection Bits		TX Trigger	DT Register Times Value Indicated (Add to Partial Product in AB)	Set TX Trigger
M1	M2			
0	0	0	0 X DT	No
0	1	0	1 X DT	No
1	0	0	2 X DT	No
1	1	0	-1 x DT (2's Complement)	Yes
0	0	1	1 X DT	No*
0	1	1	2 X DT	No
1	0	1	-1 x DT (2's Complement)	Yes
1	1	1	0 X DT	Yes

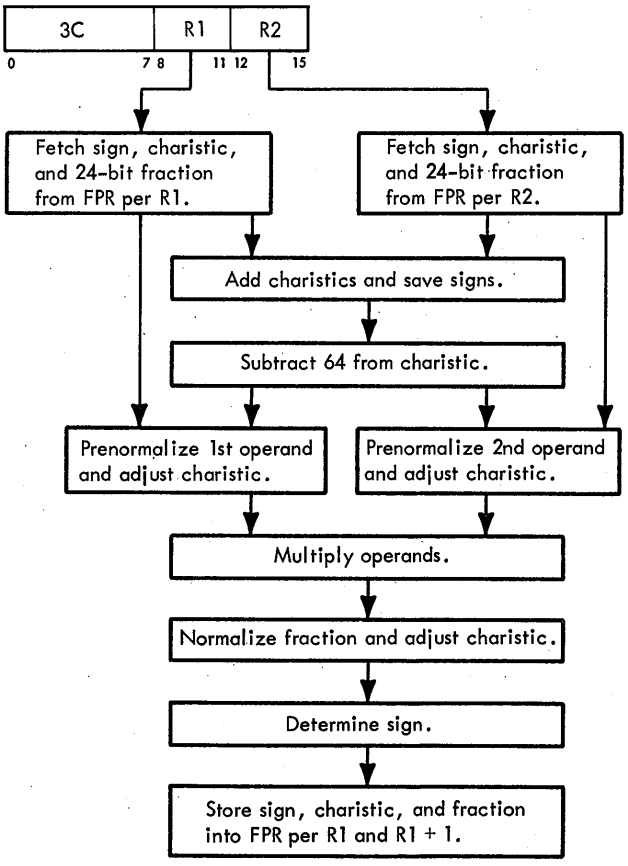
* Used on last multiple selection if 'TX' trigger is set.

At this point, all multiples of the multiplicand have been selected. If the 'TX' trigger is not set, the partial product in AB becomes the intermediate result. In this example, however, the 'TX' trigger was set because the multiplier bits equalled 11 and the 'TX' trigger was previously set (Table 3-11). Therefore, DT must be added to the partial product in AB. The contents of AB are shifted left 2 to PAB and added to DT. No right 4 shift from the parallel adder to the PAL's occurs at this time. The intermediate product is transferred from the PAL's to AB(4–67) and DT. The intermediate product is .0001 0110 1000 0↔0 (0.168_{16}). In this example, normalization is not necessary. The sign, characteristic, and fraction are stored into the FPR specified by R1 and R1 + 1. An end-op cycle completes the operation.

If the integers were preceded by zeros in this example, prenormalization of the operands would occur before executing the multiple algorithm.

Multiply, MER (3C) – RR Short Operands

- Multiply 1st operand (in FPR, per R1) and 2nd operand (in FPR, per R2) and place normalized product into 1st operand location (in FPR, per R1 and R1 + 1).
- See Note under "Multiply".
- RR format:



- Conditions at start of execution:
  1st operand is in A, B, and D (24-bit fraction only).
  2nd operand is in S and T.
  Instruction is in E.

The Multiply, MER, instruction (Diagram 5-211, FEMDM) multiplies the second operand (specified by R1) by the first operand (specified by R2) and places the normalized product into the first operand location (per R1 and R1 + 1).

The conditions at the beginning of the execution phase are:

1. The first operand is in A, B, and D (24-bit fraction only).
2. The second operand is in S and T.
3. The STC contains a value of 4.
4. The instruction is in E.

The second operand fraction (multiplicand) is transferred from T to D. The first operand sign is saved in STAT F and the second is saved in STAT C. The characteristics are added, yielding an excess-128 characteristic, and the sum is placed into F. SA(0) is saved in STAT D and placed into F(0). B and T are reset by transferring zeros from PAL(32–63) to B and T. The first operand (multiplier) is fetched from LS and placed into S to be used for the select multiple function. A constant of 15 is placed into E(12–15) to be used for selecting the two multiple bits from S. The operands are now in position so that multiplying may begin. The ROS microprogram assumes that both operands are normalized. However, the operands are tested, via a four-way branch, to determine whether prenormalization is necessary. The four-way branch tests for the following conditions:

1. First and second operands are normalized.
2. First operand is normalized and the second is unnormalized.
3. First operand is unnormalized and the second is normalized.
4. First and second operands are unnormalized.

Assume that both operands need normalizing. The second operand is normalized by left-shifting the fraction in DT one hex digit and subtracting 1 from the characteristic. Left-shifting continues until the second operand fraction is normalized.

After the fraction of the second operand is normalized, the first operand (multiplier) is transferred from S to B. The contents of T (0's for short operands) are saved in the LSWR. Normalization is accomplished by left-shifting the contents of AB one hex digit and subtracting 1 from the characteristic. (B is reset during the first shift.) Left-shifting continues until the fraction of the first operand is normalized. On each left-shift, the shifted low-order fraction (0's) is stored into the FPR specified by R1

[E(8–11)]. S is then loaded with the short operand multiplier. T is reset, and DT becomes a 56-bit multiplicand (second operand).

Because the characteristic is in excess-128 notation, 64 is subtracted from F so that the excess-64 rule applies. AB is reset, and the multiply algorithm begins. A 'SELMPL*E3' micro-order is executed, and 1 is subtracted from E(12–15) for each machine cycle. When E(12–15) = 0100, all 12 pairs of multiples have been selected. Because the 'TX' trigger may have been set on the previous multiple selection, a last multiple selection is necessary to add in the multiplicand to obtain the correct product.

Because the operands were normalized before multiplying, a maximum of one left-shift is necessary to normalize the intermediate product fraction. If A(8–11) = 0, one left shift of the intermediate product fraction is necessary. When the left shift occurs, a 1 is subtracted from the characteristic sum. The characteristic of the final product is located in SAL(1–7) and F(1–7). The sign is determined algebraically, and then the sign, characteristic, and 56-bit fraction are stored into the FPR specified by R1 and R1 + 1.

If SAL(0) = 1, an exponent overflow or exponent underflow condition exists and the product is incorrect. Zeros are stored into the first operand location if an exponent underflow has occurred. A program interruption occurs on all exponent overflows, and on exponent underflows if masked on. If SAL(0) = 0, the stored product is correct. An end-op cycle completes instruction execution.

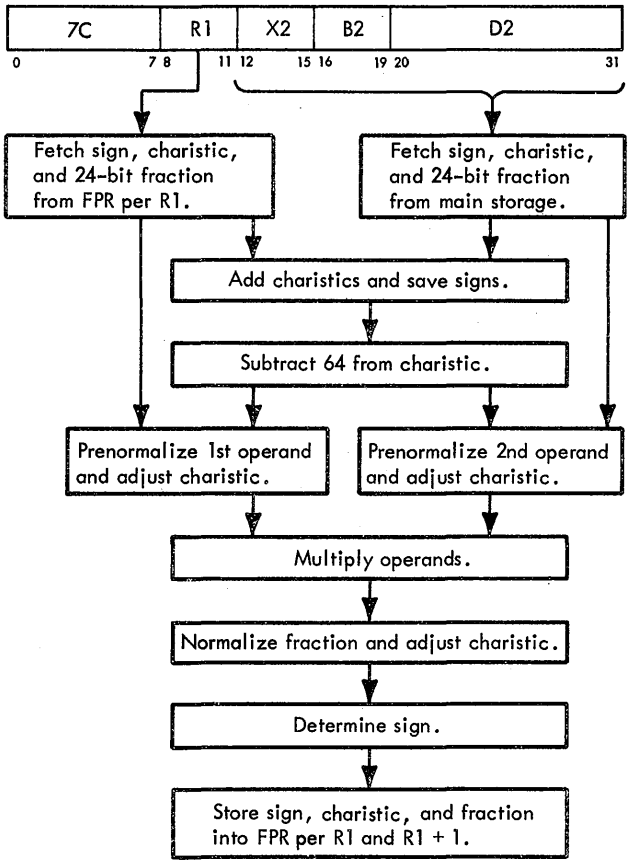
Multiply, ME (7C) – RX Short Operands

- Multiply 1st operand (in FPR, per R1) and 2nd operand (in storage) and place normalized product into 1st operand location (in FPR, per R1 and R1 + 1).
- See Note under "Multiply".
- RX format: (See adjoining column.)
- Conditions at start of execution:
  1st operand is in S and T.
  Main storage request for 2nd operand has been issued per D.
  First 16 bits of instruction are in E.

The Multiply, ME, instruction (Diagram 5-211) multiplies the second operand (specified by the effective address) by the first operand (specified by R1) and places the normalized product into the first operand location (per R1 and R1 + 1)

The conditions at the beginning of the execution phase are:

1. The first operand is in S and T.
2. A main storage request for the second operand has been issued per the effective address in D.
3. The first 16 bits of the instruction are in E.



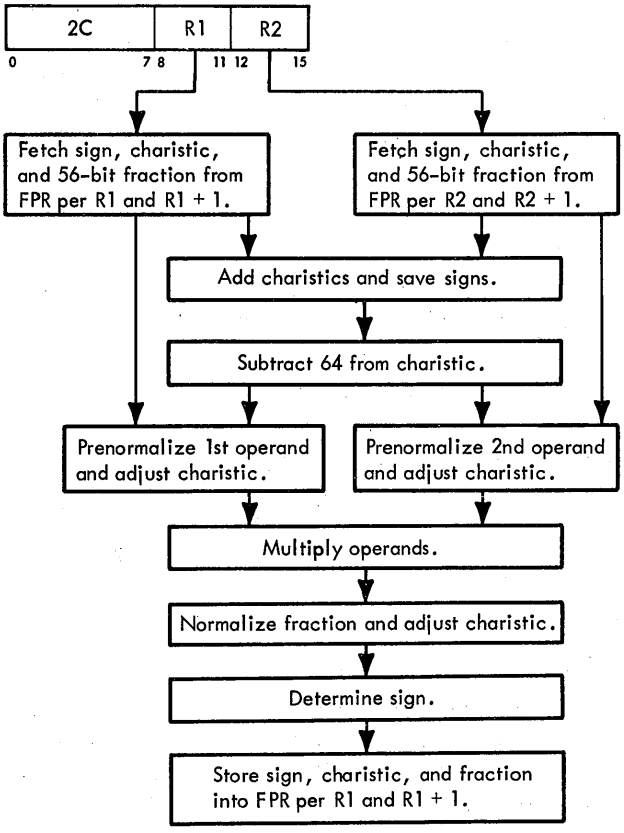
The first operand (multiplier) is placed into A, a constant of 15 is placed into E(12–15), and the STC is set to 4. Assume that the first operand is not normalized and that the second is normalized. The second operand is fetched from main storage (per the effective address in D) and placed into ST. If D(21) = 1, the second operand (multiplicand) is in T; conversely, if D(21) = 0, the second operand is in S and must be placed into T. The sign of the first operand is saved in STAT F and the sign of the second is saved in STAT C. The characteristics are added, and the sum is placed into F. SAL(0) is saved in STAT D and F(0). The fraction of the second operand is placed into D. B and T are reset, the first operand is placed into S, and 15 is placed into E(12–15). A four-way branch determines the next operation. From this point, operation is similar to that of the MER instruction.

If the first operand was normalized, the second operand (multiplicand) from main storage is placed into AB. T and the STC are reset. The transfer of the second operand fraction to D is determined by D(21). If D(21) = 1, the second operand from B is transferred to A and D. If D(21) = 0, the second operand in A is transferred to D. Note that the sign of the first operand is saved in STAT C and that of the second in STAT F. The characteristics are added, and the sum is saved in F. The characteristic carry is saved in STAT D and F(0).

Because the first operand was initially normalized, the ROS microprogram assumes that the second operand is also normalized. Therefore, the first partial product is computed. If the second operand needs normalizing, however, the operands and the constant 15 in E(12–15) are restored, and the second operand is normalized before multiplying. Once both operands are normalized, the operands are multiplied and the results stored.

Multiply, MDR (2C) – RR Long Operands

- Multiply 1st operand (in FPR, per R1 and R1 + 1) and 2nd operand (in FPR, per R2 and R2 + 1) and place normalized product into 1st operand location (in FPR, per R1 and R1 + 1).
- See Note under "Multiply".
- RR format:



- Conditions at start of execution:
  32 bits of 1st operand are in A, B, and D (24-bit fraction only).
  32 bits of 2nd operand are in S and T.
  Low-order fractions of 1st and 2nd operands are in LS.
  Instruction is in E.

The Multiply, MDR, instruction (Diagram 5-212, FEMDM) multiplies the second operand (specified by R2 and R2 + 1) by the first operand (specified by R1 and R1 + 1) and places the normalized product into the first operand location (per R1 and R1 + 1).

The conditions at the beginning of the execution phase are:

1. 32 bits of the first operand are in A, B, and D (24-bit fraction only).
2. 32 bits of the second operand are in S and T.
3. The STC contains a value of 4.
4. The instruction is in E.

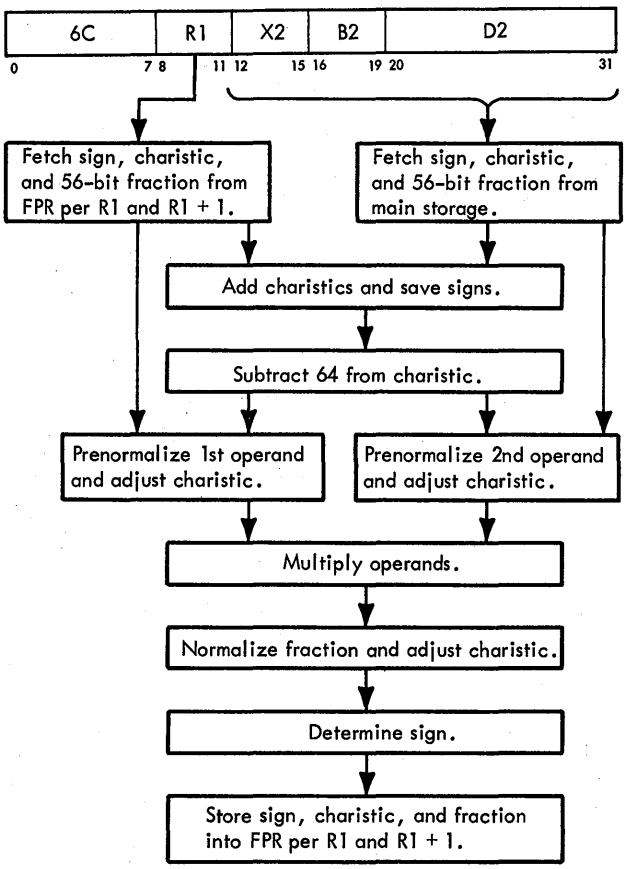
The high-order fraction of the second operand (multiplicand) is transferred from T to D. The low-order fraction of the first operand (multiplier) is placed into S, and that of the second operand is placed into T. DT contains the multiplicand fraction, and S contains the low-order fraction of the multiplier. The signs are saved in STAT C and STAT F. The characteristics are added, and the sum is placed into F. The characteristic carry is saved in STAT D and also placed into F(0). A constant of 15 is placed into E(12–15) to be used for selecting the two multiple bits located in S.

The operands are now in position so that multiplying may begin. The ROS microprogram assumes that both operands are normalized. However, the operands are tested, via a four-way branch, to determine whether prenormalization is necessary. Assume that the first operand is normalized and that the second needs normalizing. The second operand is normalized by left-shifting the contents of DT one hex digit and subtracting 1 from the characteristic on each shift. Left-shifting continues until the fraction is normalized. Because the characteristic sum is in excess-128 notation, 64 is subtracted from the characteristic in F. AB is reset, and the first multiple is selected. The multiples are selected per E(12–15) until E(12–15) = 0001, indicating that the multiples must be selected from the high-order fraction located in LS. Accordingly, the high-order fraction of the first operand (multiplier) is fetched from LS per R1 [E(8–11)], and placed into S. From this point, the multiply execution is the same as that for the MER instruction.

Multiply, MD (6C) – RX Long Operands

- Multiply 1st operand (in FPR, per R1 and R1 + 1) and 2nd operand (in storage) and place normalized product into 1st operand location (in FPR, per R1 and R1 + 1).
- See Note under "Multiply".

- RX format:



- Conditions at start of execution:
  32 bits of 1st operand are in S and T.
  Low-order fraction of 1st operand is in LS.
  Main storage request for 2nd operand has been issued per D.
  First 16 bits of instruction are in E.

The Multiply, MD, instruction (Diagram 5-212) multiplies the second operand (in storage) by the first operand (specified by R1 and R1 + 1) and places the normalized product into the first operand location (per R1 and R1 + 1).

The conditions at the beginning of the execution phase are:

1. 32 bits of the first operand are in S and T.
2. The low-order fraction of the first operand is in LS.
3. A main storage request for the second operand has been issued per the effective address in D.
4. The first 16 bits of the instruction are in E.

The first operand (multiplier) sign, characteristic, and high-order fraction are transferred from T to A. The low-order fraction of the first operand is fetched from LS and placed into S. If the first operand is not normalized, the STC is reset, the low-order fraction is transferred from S to B, the second operand (multiplicand) is fetched from main storage and placed into ST and D (high-order fraction in D), the characteristics are added, and the signs are saved (first operand sign in STAT F and second in STAT C).

The low-order fraction of the first operand is again placed into S, and a constant of 15 is placed into E(12–15). The four-way branch determines the next operation. The remainder of the operation is identical to that of the MDR instruction.

If the first operand was normalized, the second operand is fetched from main storage and placed into AB. The second operand fraction (multiplicand) is transferred from AB to DT. The sign of the first operand is saved in STAT C and that of the second in STAT F. The characteristics of the first and second operands are added; the result is placed into F, and the characteristic carry is saved in STAT D. [The carry is also transferred to F(0).] Because the first operand is normalized, the ROS microprogram assumes that the second operand is also normalized; therefore, the first multiple is selected. If the second operand needs to be normalized, the initial conditions are restored and the ROS microprogram proceeds with the normalization of the second operand.

DIVIDE

- Divides 1st operand (dividend) by 2nd operand (divisor) and places normalized quotient into 1st operand location.
- Characteristics are subtracted, and 64 is added to the characteristic difference.
- Operands are prenormalized before dividing.
- Quotient is 32 bits for short operands, 64 bits for long operands.
- Quotient is normalized.
- Sign of quotient is determined algebraically.
- No remainder is retained.

The Divide instruction divides the first operand (dividend) by the second operand (divisor) and places the normalized quotient into the first operand location. In short operand instructions, the low-order halves of the FPR's are ignored and remain unchanged.

A floating-point division consists of a characteristic subtraction and a fraction division. The difference between the dividend and divisor characteristics, plus 64, is used as an intermediate quotient characteristic.

The quotient fraction is normalized by prenormalizing the operands. Postnormalizing the intermediate quotient is never necessary, but a right-shift of one hex digit may be necessary if the normalized dividend fraction is larger

than the normalized divisor fraction. The intermediate quotient characteristic is adjusted for the shifts. Low-order digits of the quotient fraction are removed to obtain the desired number of digits.

The sign of the quotient is determined algebraically. That is, if the signs of the operands are alike, a plus sign is assigned to the quotient; if the signs are unlike, a minus sign is set into the quotient.

A program interruption for exponent overflow occurs when the final quotient characteristic exceeds 127; the operation is terminated.

A program interruption for exponent underflow occurs when the final quotient characteristic is less than zero and the corresponding mask bit is a 1. Underflow is not signalled for the intermediate quotient or for the operand characteristics during prenormalization.

If division by a divisor with a zero fraction is attempted, the divide operation is suppressed. The dividend remains unchanged, and a program interruption for floating-point divide occurs. When the dividend fraction is zero, the quotient fraction will be zero. The quotient sign and characteristic are made zero, yielding a true zero result without taking the program interruption for exponent underflow or exponent overflow. The program interruption for significance is never taken for division. The CC remains unchanged.

Characteristic Computation

After the first and second operands are fetched and placed into the proper registers, the characteristics are subtracted. Because the complement gates to the serial adder are on the SAA bus, the first operand characteristic (C1) is subtracted from the second operand characteristic (C2). Therefore, the characteristic computations differ from what might be expected. (Normally, C1 - C2 would be expected.)

The CPU takes the following steps in computing the quotient characteristic:

1. Subtracts C1 from C2 (dividend characteristic from divisor characteristic).
2. Subtracts 64 from the characteristic difference.
3. Normalizes the first operand and adds a 1 to the intermediate characteristic for each digit position that the fraction is shifted.
4. Normalizes the second operand and subtracts a 1 from the intermediate characteristic for each digit position that the fraction is shifted.
5. Takes the 2's complement of the intermediate characteristic.
6. Checks for a divisor fraction greater than a dividend fraction. If the dividend is the larger number, right-shifts the dividend one hex digit and adds 1 to the characteristic.
7. Saves the final characteristic.
8. Checks the final characteristic for exponent overflow or exponent underflow.

As an example of this computation, assume that two hex numbers are to be divided, .004 by .02:

$$\frac{\text{1st operand dividend}}{\text{2nd operand divisor}} \pm \frac{.004 \times 16^5}{.02 \times 16^2} = \pm .2 \times 16^3 = \pm 3.2$$

Fraction

Characteristic

Convert the above characteristics to excess 64 notation:

C1

$$\frac{69.004}{66.02}$$

C2

Convert the above characteristics to binary form:

$$\frac{1000101.004}{1000010.02} = \frac{69.004}{66.02} = \pm 67.2 \text{ or } .2 \times 16^3$$

after 64 is subtracted from the characteristic.

Step 1. The machine subtracts the characteristics (C2 - C1):

```
1000010    C2
0111010 }  2's complement of C1
      1 }
-------
1111101                          .004
                              1111101.02
```

Step 2. 64 is subtracted from the characteristic to maintain excess-64 notation:

```
1111101
0111111 }  2's complement of 64
      1 }
-------
0111101                          .004
                              0111101.02
```

Step 3. Note that the first operand hex fraction requires two left-shifts to prenormalize. Shift left 2 and add 2 to the characteristic:

```
0111101
0000010
-------
0111111                          .4
                              0111111.02
```

Step 4. The second operand hex fraction requires one left-shift. Shift left 1 and subtract 1 from the characteristic:

```
0111111
1111110 }  2's complement of 1
      1 }
-------
0111110                          .4
                              0111110.2
```

Step 5. Take the 2's complement of the characteristic:

```
1000001 }  2's complement of 0111110
      1 }
-------                      1000010.4
1000010                           .2
```

Step 6. Because the dividend fraction is greater than the divisor fraction, the dividend is shifted right 1 and 1 is added to the characteristic before dividing fractions:

```
1000010
0000001                                   1000011.04
1000011    = 67 = final characteristic          .2
```

Step 7. Save 67, which is the final characteristic.

Step 8. Divide fractions and store quotient:

```
1000011.04
      .2    =±  67.2
                  |  |
                  |  |_____ Result fraction
                  |________ Result characteristic
```

Subtracting the first operand characteristic from the second effectively makes the characteristic difference part of the divisor (dividend/divisor); to add to the characteristic, therefore, the value must be subtracted. For example, excess-64 notion is used in the CPU. Subtracting (C1 + 64 from C2 + 64) equals C2 - C1 + 0; therefore, 64 must be added to the characteristic difference to maintain excess-64 notation. Because the C2 minus C1 difference is 2's-complemented later in the operation, 64 must be subtracted (2's complement and add) from the characteristic that is part of the divisor. The characteristic must be part of the dividend to obtain the final quotient characteristic.

The intermediate characteristic is 2's-complemented to obtain the correct characteristic of the quotient because the initial characteristic subtraction places the intermediate characteristic in the divisor. The intermediate characteristic is not considered to be in the 2's complement form.

Normalization

In the divide operation, both fractions must be normalized before dividing the fractions. Also, the divisor must be larger than the dividend. If the divisor is less than the dividend, the dividend is divided by 16 by right-shifting the dividend four binary bit positions. Prenormalization and making the divisor larger than the dividend make postnormalization unnecessary.

Fraction Division

- Is performed as follows:
  1. Gate dividend to adder, shifted left 2; gate divisor to adder, shifted left 1.
  2. Add; result is partial dividend.
  3. Develop quotient bit: if partial dividend is in true form, place a 1 into quotient; if partial dividend is in 2's complement form, place a 0 into quotient.
  4. Gate partial dividend and divisor to adder (*no* shift). If partial dividend is in true form, gate divisor in 2's complement form; if partial dividend is in 2's complement form, gate divisor in true form.
  5. Perform steps 2 and 3.
  6. Perform steps 1–5.
  7. Continue until count signals end of algorithm.

The basic algorithm for the floating-point fraction divide operation is similar to the algorithm used in fixed-point divide. The characteristics of the two operands are subtracted, and 64 is added to maintain excess-64 notation. The divisor fraction is subtracted from the dividend fraction. A carry indicates that the dividend is greater than the divisor. The dividend must be less than the divisor; if not, a right 4 shift of the dividend is required. Division is accomplished by successive subtractions and storing of quotient bits as determined by the carry. Successive subtractions are performed, and the divisor is, in effect, shifted right one position with respect to the dividend for each subtraction (actually, the dividend is shifted left as explained below).

In binary arithmetic, to divide one number (dividend) by another (divisor), the dividend is repeatedly reduced by subtracting the divisor. The number of times this reduction can be done is the solution (quotient). There are two methods of performing binary division: restore and nonrestore (Figure 3-11).

In restore division, the result of a reduction of the dividend by the divisor is retained only if the result is the true difference (as opposed to a 2's complement difference); a carry indicates that the result is in true form. This result, called the partial dividend, is used in the next reduction. However, if the result is the 2's complement of the difference (no carry), the result is discarded and the old partial dividend is doubled in relation to the divisor to participate in the next reduction. A 1 is inserted into the quotient when the result is true (carry), and a 0 is inserted when the result is in 2's complement form (no carry).

In the nonrestoring method of division, the result of a reduction is retained as the new partial dividend whether it is in true or 2's complement form. When a partial dividend is in true form, the 2's complement of the divisor is added to it; when the partial dividend is in 2's complement form, the true divisor is added to it. In each reduction, the partial dividend is shifted left one bit in relation to the divisor; also, a 1 is inserted into the quotient when the result is true (carry) and a 0 is inserted into the quotient when the result is in 2's complement form (no carry).

Shifting the dividend left one position doubles its value and is equivalent to halving the divisor. Similarly, shifting the dividend left two positions quadruples it and is equivalent to reducing the divisor to ¼ of its value. Note

Problem: 45 ÷ 7 = 6 3/7

```
Dividend: 0 0 1 0  1 1 0 1 = (2 x 16) + (13 x 1) = 45
Divisor:          0 1 1 1 =          ( 7 x 1) =  7
Quotient:         0 1 1 0 =          ( 6 x 1) =  6
Remainder:        0 0 1 1 =          ( 3 x 1) =  3
```

n/c = no carry
c = carry

A. Restore

```
                          00110
                 0111 | 00101101
2's Complement Divisor:     1001          1st Reduction
2's Complement Result (Discarded):
                  n/c    1011

True Partial Dividend:      0101
2's Complement Divisor:     1001          2nd Reduction
2's Complement Result (Discarded):
                  n/c    1110

True Partial Dividend:      1011
2's Complement Divisor:     1001          3rd Reduction
True Result:              c 0100

True Partial Dividend:      1000
2's Complement Divisor:     1001          4th Reduction
True Result:              c 0001

True Partial Dividend (Remainder):  0011
2's Complement Divisor:             1001  5th Reduction
2's Complement Result (Discarded):
                          n/c       1100
```

B. Non-Restore

```
                          00110
                 0111 | 00101101
2's Complement Divisor:     1001          1st Reduction
2's Complement Result:
                  n/c    1011

2's Complement Partial Dividend:  0111
True Divisor:                     0111    2nd Reduction
2's Complement Result:
                          n/c     1110

2's Complement Partial Dividend:  1101
True Divisor:                     0111    3rd Reduction
True Result:                    c 0100

True Partial Dividend:      1000
2's Complement Divisor:     1001          4th Reduction
True Result:              c 0001

True Partial Dividend (Remainder):  0011
2's Complement Divisor:             1001  5th Reduction
2's Complement Result:
                          n/c       1100

2's Complement Partial Dividend (Remainder-Divisor):  1100
True Divisor:                                         0111  Correction Cycle
True Result (Remainder):                            c 0011
```

Figure 3-11. Restore and Non-Restore Division

the similarity between the restore and nonrestore methods of division (the first successful reduction of the dividend is made by one-quarter of the divisor):

(Restore) Dividend minus ¼ Divisor, is equivalent to (Non-Restore) Dividend minus Divisor + ½ Divisor + ¼ Divisor.

In the 2065, the nonrestoring method of division is used, and the dividend is shifted left rather than shifting the divisor right. The development of the quotient and the left-shifting of the dividend is performed as follows:

1. The dividend is gated to the adder shifted left 2. The divisor is gated to the adder shifted left 1, thus yielding an equivalent right 1 displacement with respect to the dividend. (The divisor may be in true or 2's complement form, depending on the partial dividend.)
2. The two numbers are added, and the result (partial dividend) is placed back into the dividend register.

3. A quotient bit is developed from the partial dividend obtained in step 2. If the partial dividend is in true form, a high-order carry occurred and the high-order bit is a 0; a 1 is therefore placed into the quotient. Conversely, if the partial dividend is in 2's complement form, a high-order carry did *not* occur and the high-order bit is a 1; a 0 is therefore placed into the quotient.
4. The partial dividend and the divisor are gated to the adder. Because the partial dividend was shifted left 2 in step 1, the correct displacement between them exists, and they are not shifted now. If the partial dividend obtained in step 2 is in true form, the divisor is gated in 2's complement form; if the partial dividend is in 2's complement form, the divisor is gated in true form.
5. Steps 2 and 3 are repeated.
6. Steps 1 through 5 are repeated.

The operation continues developing quotient bits, developing new partial dividends, gating the partial dividend to the adder (shifted left 2 every other time), and gating the divisor to the adder in true or 2's complement form (shifted left 1 every other time) until a count, which is reduced every time the partial dividend is shifted, signals the end of the algorithm.

Figure 3-12 is an example of the divide operation as it is performed in the 2065 CPU. Note that the first subtraction does *not* result in a quotient bit but is used to decide whether the divisor is to be true- or 2's complement-added on the next cycle.

Data Flow and Algorithm

- Signs are saved in STAT's C and F.
- Characteristic computation is performed in serial adder.
- Characteristic difference is saved in F.
- 'DVDL0' and 'DVDL1' micro-orders are unique to divide algorithm.

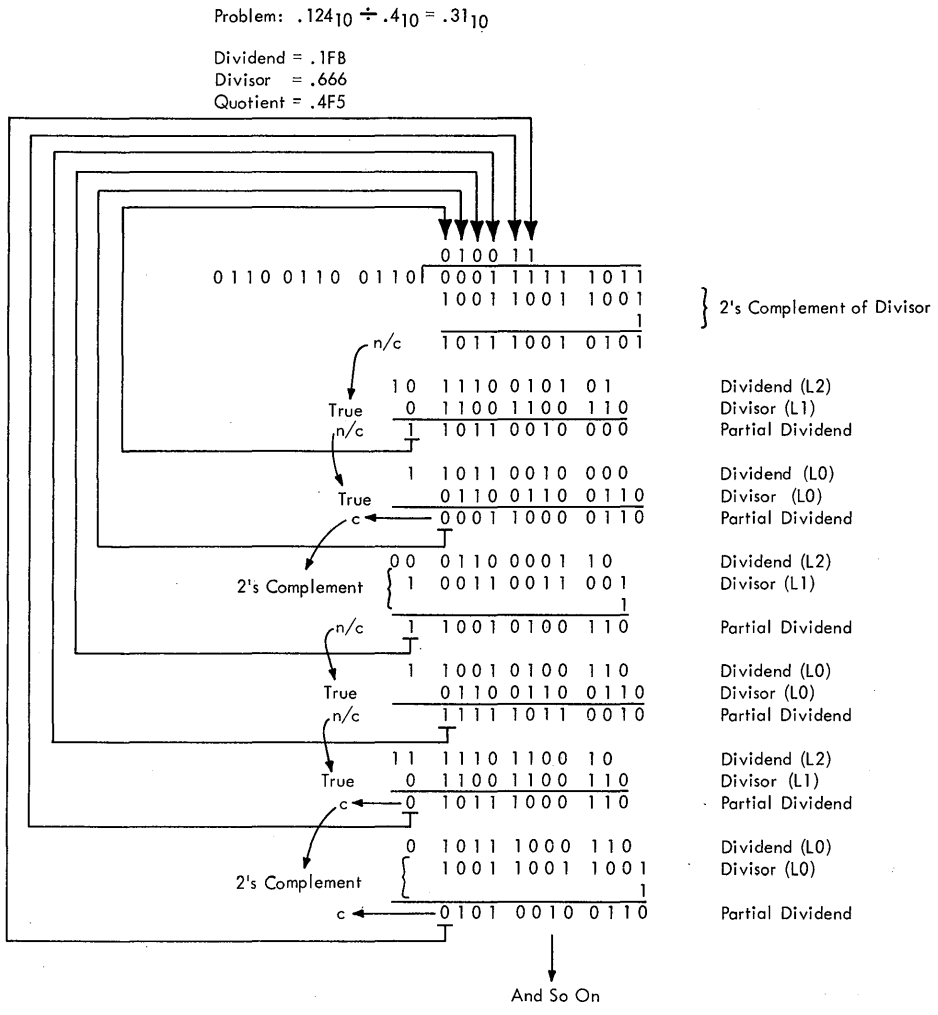


Figure 3-12. Fraction Divide Example

- Fraction division uses parallel adder.
- Divisor is in DT; dividend is in AB; quotient is developed in S.

The first operation that occurs is the computation of the final characteristic (For an example, see "Characteristic Computation"). The data paths for the signs and characteristics are shown in A of Diagram 5-213, FEMDM. The signs are saved in STAT C and STAT F. The first and second operand characteristics are gated to the SAA and SAB per the ABC and STC, respectively. To subtract the characteristics, the 2's complement of the first operand characteristic is added to the second operand characteristic. The characteristic difference is stored into F(0–7), and the characteristic carry [SA(0)] is saved in STAT D. Other inputs to the SAB bus allow subtracting 64, subtracting 1, gating the 2's complement of F, or adding 1 to the value in F. After the final characteristic is computed, the result is stored into S(0–7) per the STC.

The data path for the derivation of the divide multiple is shown in B of Diagram 5-213. When the divide algorithm begins, the divisor (first operand) is in DT and the dividend is in AB.

Two micro-orders ('DVDL0' and 'DVDL1') are used specifically during the divide algorithm. These micro-orders have three functions: (1) to gate the true or 2's complement of DT (divisor) to the PAA; (2) to determine the amount of shift (L0 = no shift, L1 = left 1 shift) of the divisor (contents of DT) to the PAA; and (3) to determine the partial quotient (PQ) bit and the PQ bit location after the subtraction of the divisor and the partial dividend has taken place.

Whether the true or 2's complement form of the divisor is sent to the PAA is determined by the PA(4) carry from the previous algebraic subtraction of the divisor and partial dividend, and the amount of shift (L0 or L1) as determined by the 'DVDL0' or the 'DVDL1' micro-order. If a PA(4) carry occurred, the 2's complement is gated (L0 or L1) to the parallel adder. If a PA(4) carry did not occur, DT is gated (L0 or L1) in true form to the parallel adder. The data in AB is gated to PAB with no shift or a left 2 shift under micro-order control ('AB' and 'ABL2').

As previously noted, the 'DVDL0' and the 'DVDL1' micro-orders determine the PQ bit and the location of the bit. The PQ bit is determined by testing AB(4) for a 0 or a 1. If AB(4) = 1, the partial dividend is in 2's complement form and a 0 is placed into the selected PQ SAL location. If AB(4) = 0, the partial dividend is in true form, and a 1 is placed into the selected PQ SAL location.

As shown in B of Diagram 5-213, the PQ location in SAL is determined by E(14,15) and by the 'DVDL0' or 'DVDL1' micro-order. E(14,15) selects the pair of SAL bits into which the PQ bit is to be placed. The 'DVDL0' micro-order selects the odd bit of the selected pair; the 'DVDL1' micro-order selects the even bit. At the same time that the PQ bit is gated into SAL, the contents of F are added to the PQ bit and saved in F. After a PQ byte (eight bits) is available, the contents of F(0–7) are gated to S per the STC. After S is filled with the quotient (or PQ), the contents of S are stored into LS per E(8–11).

For a discussion of the divide algorithm, assume that the final characteristic is in S(0–7) and that the normalized fractions are in DT (divisor) and AB (dividend). By definition, the CPU requires that floating-point numbers consist of a sign, a characteristic, and a fraction. Because no provisions are made in the CPU to handle integers in floating-point instructions, the divisor must be larger than the dividend to retain a fraction quotient. After both fractions are normalized, therefore, the contents of DT are subtracted from the contents of AB. A carry from PA(4) indicates that the dividend is larger than the divisor. Whenever the dividend is larger than the divisor, the contents of AB must be restored and shifted right 4 (divided by 16) before proceeding with the divide algorithm, and a 1 must be added to the characteristic. If no carry occurred from PA(4), the dividend is less than the divisor, and the CPU proceeds with the divide algorithm.

When the divisor (d) is subtracted from the dividend (D), the difference is placed into AB (D - d in AB). If a right 4 shift was necessary, the divisor (d) is restored and divided by 16 (d in DT). AB now contains the dividend (D). At the beginning of the divide algorithm, the 2's complement of DT is shifted left 1 and added to the contents of AB shifted left 2; the result is placed into AB, thus yielding an effective left 1 shift of AB (dividend). The contents of AB may be expressed by the equation 4D - 2d = contents of AB. The value 4D - 2d is in AB after the first machine cycle of the divide algorithm.

If the dividend was less than the divisor, D - d is in AB. The CPU proceeds to add the contents of DT shifted left 1 to the contents of AB shifted left 2, with the result placed into AB. This addition results in the equation 4(D - d) + 2d = contents of AB. Simplifying the equation yields 4D - 4d + 2d = 4D - 2d. At the end of the first cycle of the divide algorithm, the same result (4D - 2d) is obtained as when the dividend was larger than the divisor. The CPU continues with the divide algorithm.

During the first machine cycle of the divide algorithm, the 'DVDL0' micro-order also selects the DT gating to the parallel adder per the PA(4) carry. The subtraction resulting from the 'DVDL0' micro-order is accomplished during the next machine cycle. The PQ bit, however, is determined by the A(4) value that was computed during the previous machine cycle. On the first cycle of the divide algorithm, the contents of AB (dividend) are shifted left 2 by a micro-order and added to the contents of DT (divisor) shifted left 1 per a micro-order. Thus the divisor is shifted right 1 with respect to the dividend, but

the resulting partial dividend is displaced left 2 in AB. On the next divide select multiple subtraction, the dividend and the divisor are subtracted without shifting, yielding the correct right 1 displacement. The following cycle causes AB and DT to shift again. Note that, as the dividend is shifted left, the low-order bit positions of AB are filled with 0's.

As previously noted, the PQ bit is gated to SAL per E(14,15) and the 'DVDL0' or 'DVDL1' micro-order. A 1 is added to the ABC after each pair of PQ bits is gated to F via SAL. When the ABC equals 3, F contains eight PQ bits (one per byte). The PQ byte is gated to S per the STC.

After each byte is gated to S, a 1 is added to the STC. When the STC equals 3, S contains the characteristic and fraction (or high-order fraction). The contents of S are stored into the LSWR.

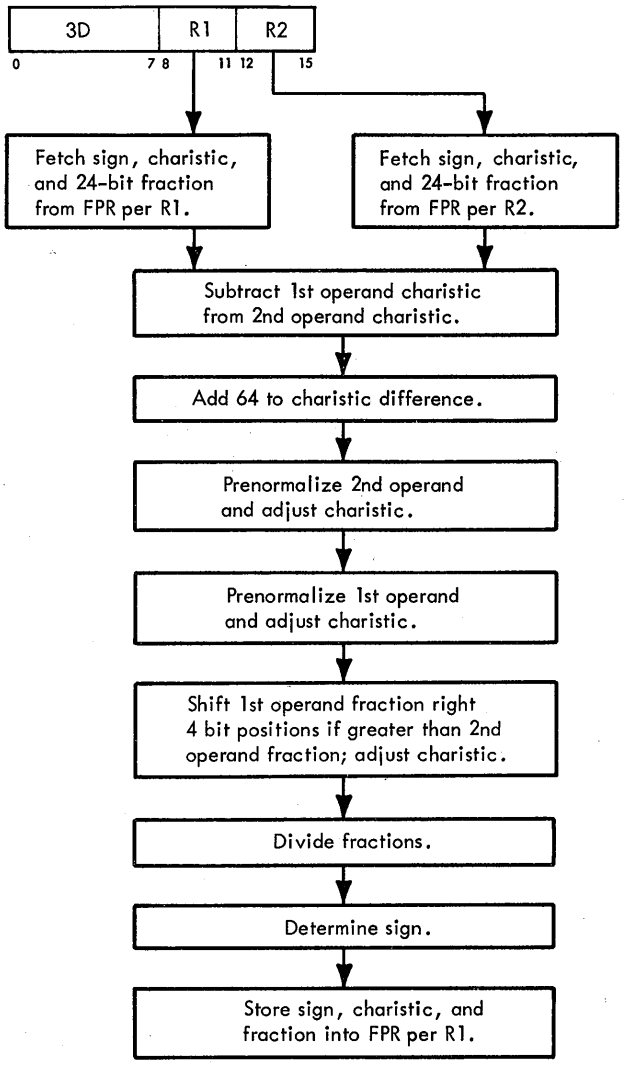
Before initiating the divide algorithm, STAT D was reset to indicate the first pass at loading PQ bytes into S. After the sign, characteristic, and high-order fraction are stored into the LSWR, the instruction and STAT D determine the next operation. If a short operand instruction is being executed, the sign is inserted and stored with the characteristic and fraction into the FPR per E(8–11). An end-op cycle completes instruction execution.

If the instruction was a long operand instruction, the sign, characteristic, and high-order fraction are stored into the FPR per E(8–11). STAT D is set, and the divide algorithm continues. The contents of the LSWR are returned to T. The same operations as described above are performed to obtain the remaining low-order fraction part of the quotient, and the same three-way branch is encountered. This time the divide algorithm is completed, and the low-order fraction is stored into the FPR per E(8–11) + 1. An end-op cycle completes instruction execution. The remainder in AB is not stored.

Divide, DER (3D) – RR Short Operands

- Divide 1st operand (in FPR, per R1) by 2nd operand (in FPR, per R2) and place normalized quotient into 1st operand location.
- RR format: (See adjoining column.)
- Conditions at start of execution:
  1st operand is in A, B, and D (24-bit fraction only).
  2nd operand is in S and T.
  Instruction is in E.

The Divide, DER, instruction (Diagram 5-214, FEMDM) divides the first operand (specified by R1) by the second operand (specified by R2) and places the normalized quotient into the first operand location. No remainder is retained.



The conditions at the beginning of the execution phase are:

1. The first operand is in A, B, and D (24-bit fraction only).
2. The second operand is in S and T.
3. The STC contains a value of 4.
4. The instruction is in E.

If no specification check occurred, the second operand fraction is transferred from T to D. The characteristics are subtracted, and 64 is algebraically added to the characteristic difference to maintain excess 64-notation. The sign of the first operand is saved in STAT F and the sign of the second operand is saved in STAT C. B and T are reset, and the contents of AB and ST are treated as 56-bit fractions.

In the divide instructions, both operands are prenormalized before the divide algorithm begins. A four-way branch determines the prenormalization path by testing A(8–11), the dividend, and PAL(40–43), the divisor, for the normalized conditions:

1. The first and second operands are normalized.

2. The first operand is normalized, and the second is unnormalized.
3. The first operand is unnormalized, and the second is normalized.
4. The first and second operands are unnormalized.

Assume that both operands are unnormalized. The second operand (divisor in DT) is shifted left 4 until the operand is normalized. A 1 is subtracted from the intermediate quotient characteristic for each shift.

After the second operand is normalized, the first operand is normalized by left-shifting until the fraction contains a hex digit [A(8–11) not equal to zero]. On each left-shift, a 1 is added to the intermediate quotient characteristic in F.

After the operands are normalized, the second operand fraction (divisor) is subtracted (take 2's complement of second operand and add) from the first operand fraction. Before branching on the PAL(4) carry, the 2's complement of the intermediate characteristic is computed and placed into F. Also, the constant 5 is placed into E(12–15) for controlling the divide algorithm. A carry from PAL(4) indicates that the dividend is larger than the divisor. If the dividend is larger than the divisor, the dividend is restored and is divided by 16 by a right-shift of one hex digit. A 1 is added to the characteristic value, which is the final characteristic of the quotient. The final characteristic is placed into S(0–7).

No carry from PAL(4) indicates that the dividend is less than the divisor, at which time the first machine cycle of the divide algorithm is executed. A test is made to determine an overflow or underflow condition. Assume that no overflow or underflow condition exists.

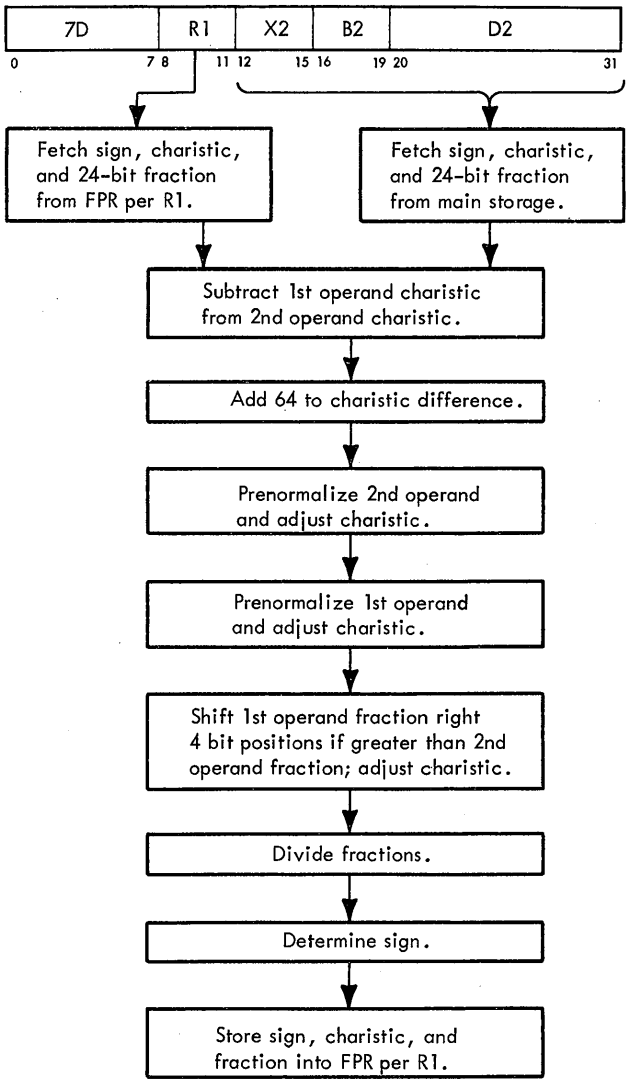
Fraction division begins as shown in Sheet 4 of Diagram 5-214. Figure 3-13 is an example of the action that occurs in parallel adder bits 4–11 (bits 12–31 being considered to equal 0's).

During the normalization routine, tests for zero fractions are made. If the second operand fraction (divisor) equals zero, the divide operation is suppressed and a floating-point divide program interruption occurs. If the first operand fraction (dividend) equals zero, a true zero quotient results. A true zero is stored into the first operand location, and an end-op cycle completes instruction execution.

Divide, DE (7D) – RX Short Operands

- Divide 1st operand (in FPR, per R1) by 2nd operand (in storage) and place normalized quotient into 1st operand location.

- RX format:



- Conditions at start of execution:
  1st operand is in S and T.
  Main storage request for 2nd operand has been issued per D.
  First 16 bits of instruction are in E.

The Divide, DE, instruction (Diagram 5-214) divides the first operand (specified by R1) by the second operand (from main storage) and places the normalized quotient into the first operand location. No remainder is retained.

The conditions at the beginning of the execution phase are:

1. The first operand is in S and T.

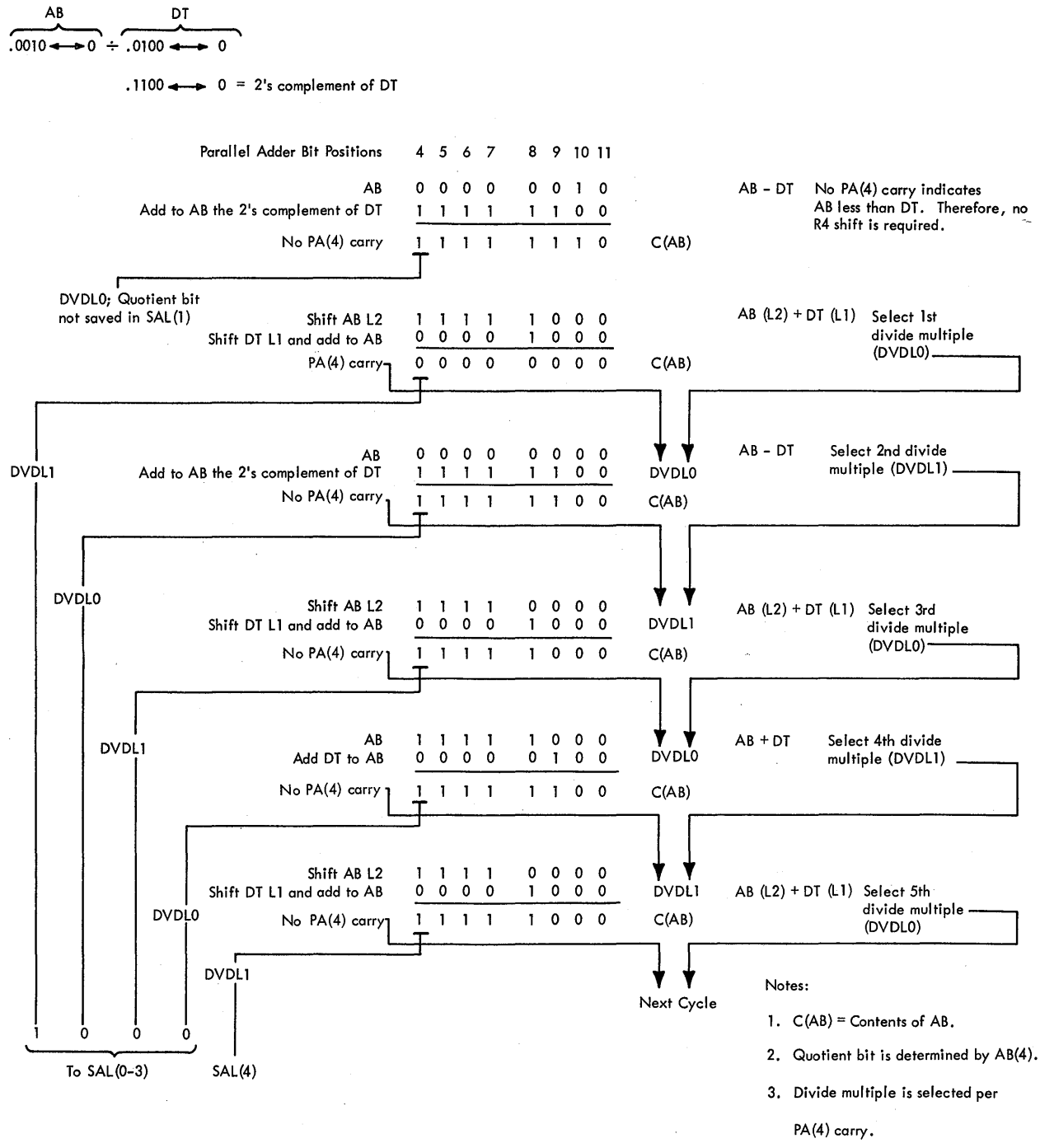


Figure 3-13. Floating-Point Divide Example

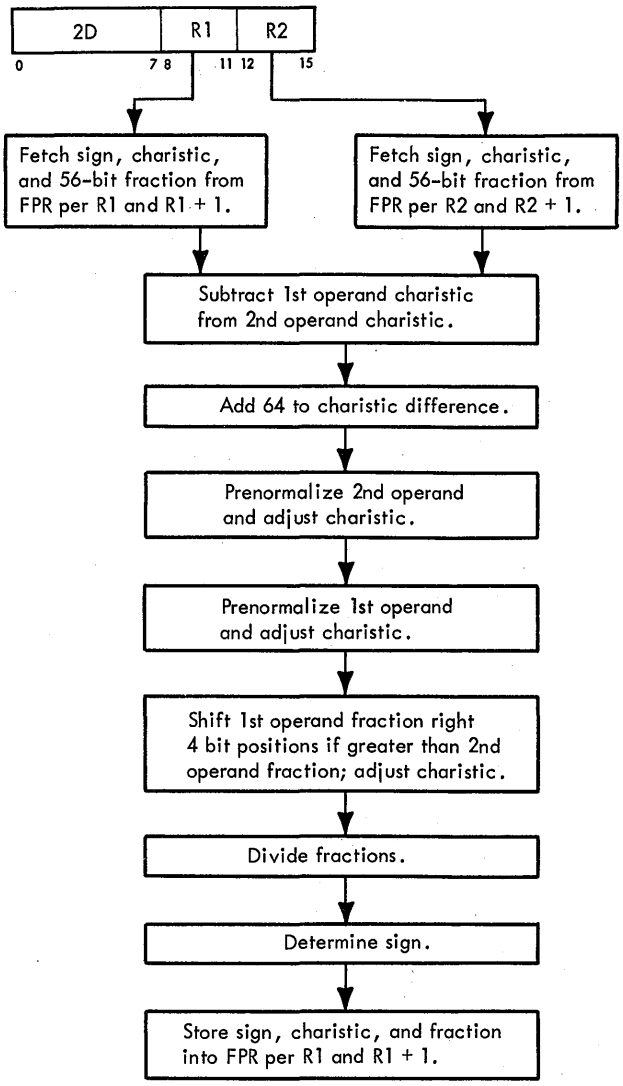
2. A main storage request for the second operand has been issued per the effective address in D.
3. The first 16 bits of the instruction are in E.

The first operand is transferred from T to A, and the STC is set to 4. The second operand is fetched from main storage per D. D(21) determines which 32 bits of the 64-bit doubleword are gated to T (Sheet 2 of Diagram 5-214). From this point, instruction execution is the same as that of the DER instruction.

Divide, DDR (2D) — RR Long Operands

- Divide 1st operand (in FPR, per R1 and R1 + 1) by 2nd operand (in FPR, per R2 and R2 + 1) and place normalized quotient into 1st operand location.

- RR format:



- Conditions at start of execution:
  32 bits of 1st operand are in A, B, and D (24-bit fraction only).
  32 bits of 2nd operand are in S and T.
  Low-order fractions of 1st and 2nd operands are in LS.
  Instruction is in E.

The Divide, DDR, instruction (Diagram 5-215, FEMDM) divides the first operand (specified by R1 and R1 + 1) by the second operand (specified by R2 and R2 + 1) and places the normalized quotient into the first operand location. No remainder is retained.

The conditions at the beginning of the execution phase are:

1. 32 bits of the first operand are in A, B, and D (24-bit fraction only).
2. 32 bits of the second operand are in S and T.
3. The STC contains a value of 4.
4. The instruction is in E.

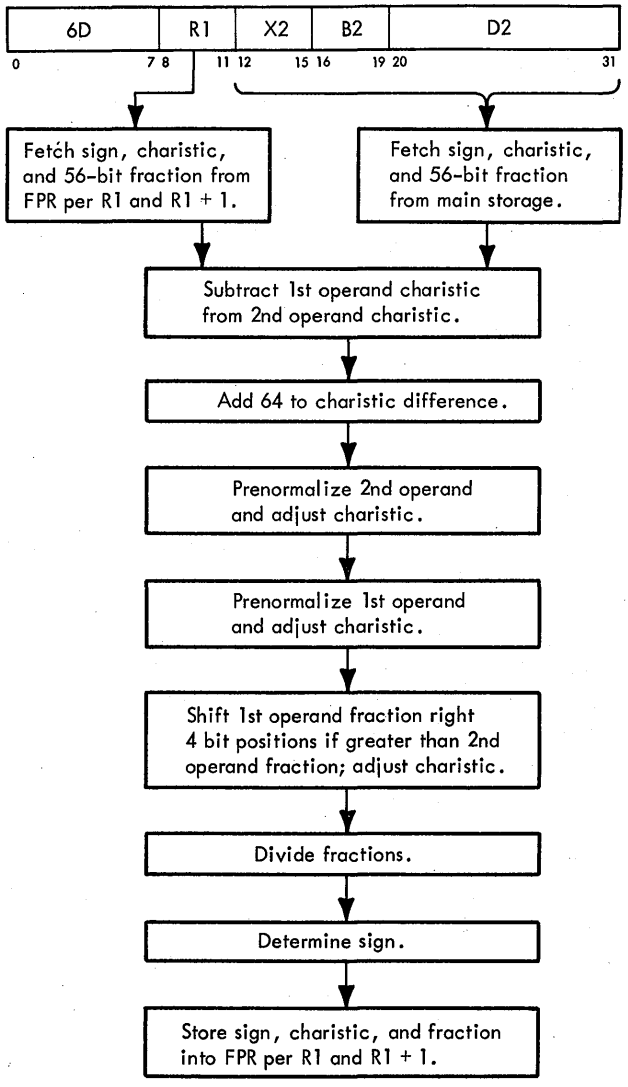
If no specification check occurred, the high-order fraction of the divisor is gated from T to D. The low-order fractions of the first and second operands are placed into B and T, respectively. As a result, the dividend fraction is in AB and the divisor fraction is in DT.

The signs are saved in STAT C and STAT F. The characteristics are subtracted, and excess-64 notation is maintained. The ABC is reset.

The four-way branch (Sheet 3 of Diagram 5-215) determines the next operation. The normalization routine, divide algorithm, and end ops are explained in the DER instruction discussion.

Divide, DD (6D) — RX Long Operands

- Divide 1st operand (in FPR, per R1 and R1 + 1) by 2nd operand (in storage) and place normalized quotient into 1st operand location.
- RX format:



- Conditions at start of execution:
  32 bits of 1st operand are in S and T.
  Low-order fraction of 1st operand is in LS.
  Main storage request for 2nd operand has been issued per D.
  First 16 bits of instruction are in E.

The Divide, DD, instruction (Diagram 5-215) divides the first operand (specified by R1 and R1 + 1) by the second operand (from main storage) and places the normalized quotient into the first operand location. No remainder is retained.

The conditions at the beginning of the execution phase are:

1. 32 bits of the first operand are in S and T.
2. The low-order fraction of the first operand is in LS.
3. A main storage request for the second operand has been issued per the effective address in D.
4. The first 16 bits of the instruction are in E.

The sign, characteristic, and high-order fraction of the first operand are transferred from T to A. The low-order fraction of the first operand is fetched from LS and placed into B via T and the parallel adder. The second operand (64 bits) is fetched from main storage and placed into ST. The high-order fraction is transferred from S to D, and the contents of DT become the 56-bit fraction divisor. The signs are saved in STAT C and STAT F. The characteristics are subtracted, and 64 is added to the characteristic difference to maintain excess-64 notation.

The first operand (dividend) is in AB, and the second operand (divisor) is in DT. The next step is to check for the prenormalization of the dividend and divisor fractions (Sheet 3 of Diagram 5-215). The remainder of the operation is identical to that of the DDR instruction.

STORE

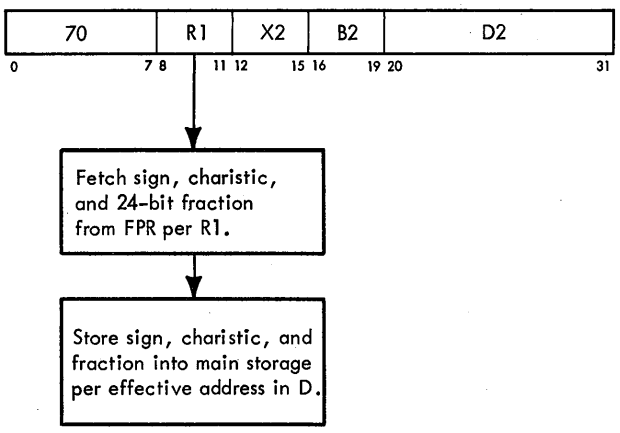
The Store instructions (STE and STD) store the first operand from an FPR in LS into the second operand location in main storage. The Store instructions are in the RX format with short and long operand options available. In the STE instruction, the low-order half of the first operand register is ignored. The first operand location remains unchanged.

Storing must be on word boundaries for the STE instruction and on doubleword boundaries for the STD instruction.

For all Store instructions, an address store compare test is made because the instructions that are in Q may be modified in main storage by the Store instruction. If an instruction is modified in main storage and is not corrected in Q, the program may not be properly executed; therefore, Q must be reloaded. The address store compare test compares the main storage address, where data is to be stored, with the effective address indicated by the Store instruction. The comparison is made by subtracting the contents of D (effective address) from the contents of the IC, shifting the difference right 4, and testing for a zero result. If the difference equals zero, the difference is less than 16; therefore, the 'program store compare' trigger is set to indicate that the instructions in Q must be refetched. The address store compare test is discussed in Section 1 of this Chapter.

Store, STE (70) – RX Short Operands

- Store 1st operand (in FPR, per R1) into 2nd operand location (in storage).
- RX format:

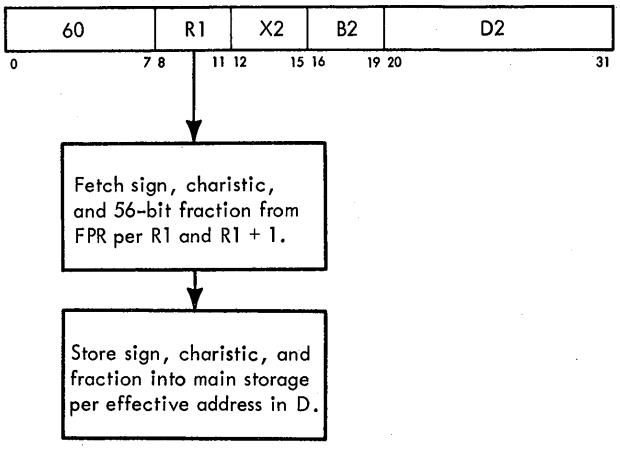


- Conditions at start of execution:
  1st operand is in S and T.
  Main storage request has been issued per effective address in D.
  First 16 bits of instruction are in E.

The Store, STE, instruction (Diagram 5-216, FEMDM) stores the first operand from the FPR specified by R1 into the main storage location specified by the effective address (second operand location). During the RX I-Fetch, the effective address is computed and placed into D, and a main storage request is initiated. The effective address must be on a word boundary or a specification program interruption is taken. Bit 21 of the effective address [PAL(61)] is tested to determine which mark triggers are set. If PAL(61) = 1, mark triggers 4–7 are set; if PAL(61) = 0, Mark triggers 0–3 are set. An address store compare test is then performed, and the instruction is terminated by an end-op cycle.

Store, STD (60) — RX Long Operands

- Store 1st operand (in FPR, per R1 and R1 + 1) into 2nd operand location (in storage).
- RX format:



- Conditions at start of execution:
  32 bits of 1st operand are in S and T.
  Low-order fraction of 1st operand is in LS.
  Main storage request has been issued per effective address in D.
  First 16 bits of instruction are in E.

The Store, STD, instruction (Diagram 5-216) stores the first operand from the even/odd pair of FPR's specified by R1 and R1 + 1 into the main storage location specified by the effective address (second operand location). During the RX I-Fetch, the effective address is computed and placed into D and a main storage request is initiated. The effective address must be on a doubleword boundary or a specification program interruption is taken. The low-order half of the first operand is gated to T from the FPR specified by R1 + 1, and mark triggers 0–7 are set. After an address store compare test is performed, the 64-bit operand is stored per the effective address, and the instruction is terminated by an end-op cycle.

www.IBM360.info

Section 4. Decimal Instructions

This section discusses the nine instructions that operate on decimal data. The instructions use the SS format and assume packed operands and results except for Pack, which has a zoned operand, and Unpack, which has a zoned result. For a discussion of the number representation, data formats, excess-6 arithmetic, operand addressing, instruction formats, data flow, program interruptions, and condition codes, refer to Chapter 1.

INSTRUCTION HANDLING

- Depending on instruction, processing of 1 or 2 operands may be specified.
- Pack, Unpack, Move with Offset, and Zero and Add instructions operate on 1 operand.
- Add, Subtract, Compare, Multiply, and Divide instructions operate on 2 operands.
- All add-type instructions set CC.
- Major serial adder functions used by decimal instructions are:
  Excess-6 translation.
  Decimal correction.
  Complement gating.
  Cross-gating.
  Zone or sign insertion.
  Invalid digit and sign detection.
  Zero detection.

Decimal instructions may be classified into the general categories of 1- and 2-operand instructions. The 1-operand instructions are Pack, Unpack, Move with Offset, and Zero and Add. The 2-operand instructions are Add, Subtract, Compare, Multiply, and Divide.

In the 1-operand instructions, the first operand is not processed but its address is used as the destination address; the second operand is processed, and the results are placed into the first operand location. The 1-operand instructions are handled by fetching the second operand to AB. Successive AB bytes are selected per the ABC and are processed in the serial adder, and the resultant bytes are entered into ST per the STC. After all second operand bytes have been processed, the contents of ST are stored into main storage at the first operand address.

The 2-operand Add, Subtract, and Compare instructions are executed by fetching the first operand to ST and the second operand to AB. A true add or complement add operation is then performed in the serial adder one byte at a time, with the resultant bytes replacing the first operand bytes in ST as they are processed. For the Add and Subtract instructions, the results are stored into main storage at the first operand address. The Compare instruction does not store the result, but performs a test to determine the high, low, or equal relationship of the first operand to the second operand and sets the CC accordingly.

For the 2-operand Multiply and Divide instructions, the operands must be properly aligned in the registers prior to entering execution. This function is performed by the appropriate right- and left-adjust sequences incorporated in the individual microprogram of the instruction.

Basically, the multiply operation is performed by repetitive addition. The product bytes are developed one byte at a time, starting with the low-order byte. Each time one byte of product is developed, it is stored into main storage under control of the corresponding mark trigger. The instruction then proceeds to develop the next higher-order product byte. Upon execution of the instruction, the first operand is completely replaced by the product.

The Divide instruction is performed by repetitive subtraction. It is the only decimal instruction that processes the operands starting with the high-order bytes. The full divisor and a sufficient number of high-order dividend bytes are fetched to perform the first successful subtraction. Then by repeatedly subtracting the divisor from the dividend and counting the number of successful subtractions, the high-order quotient byte is developed. This byte is stored into main storage, and the instruction proceeds to develop the next lower-order quotient byte. Upon execution of the instruction, the first operand is completely replaced by the quotient and the remainder. The remainder occupies the low-order portion of the destination field.

The results of the Add, Subtract, and Compare instructions are used to set the CC. All other decimal instructions leave the code unchanged. The Add and Subtract instructions set the CC to 0, 1, or 2 to indicate a zero, less-than-zero, or greater-than-zero result; the CC is set to 3 if the result of the operation overflows. The Compare instruction sets the CC to 0, 1, or 2 to indicate that the first operand was equal to, less than, or greater than the second operand.

The serial adder performs many functions on its input data. The functions of excess-6 translation, decimal correction, and complement gating are discussed in "Data

Handling" in Chapter 1, Section 4. Additional serial adder functions used by the decimal operations are:

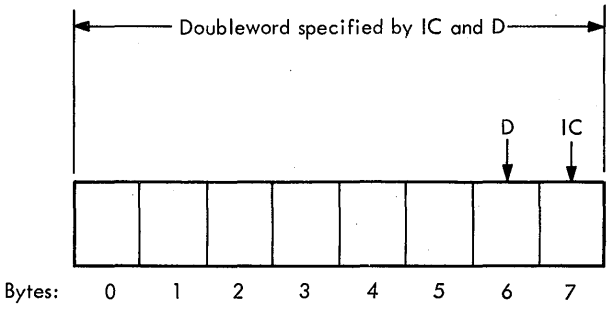
1. Cross-gating. The two-digit input to the A-side of the adder is swapped upon gating to SAL; the digit at adder A-side (0–3) is gated to SAL(4–7), and A-side (4–7) is gated to SAL(0–3). This function is used mainly by the Pack and Unpack instructions to interchange the sign and digit positions.
2. Zone or sign insertion. The correct zone or sign code (USASCII-8 or EBCDIC) is applied from the ROSDR to the adder A-side. The zone or sign may be merged with the digit in any combination.
3. Invalid digit and sign detection. The inputs to the A and B sides of the adder are tested for invalid digits or signs. An appropriate interrupt trigger is set upon detection of an invalid code.
4. Zero detection. This function is used to sense overflow conditions and also to detect all-zero results. An all-zero result placed into main storage must carry a positive sign. Consequently, arithmetic instructions such as Add and Subtract specify testing of each SAL byte for zeros. If upon execution of the instruction it is found that an all-zero result has been stored, the instruction forces storing of a plus sign at the low-order byte address.

WORD OVERLAP CONDITION

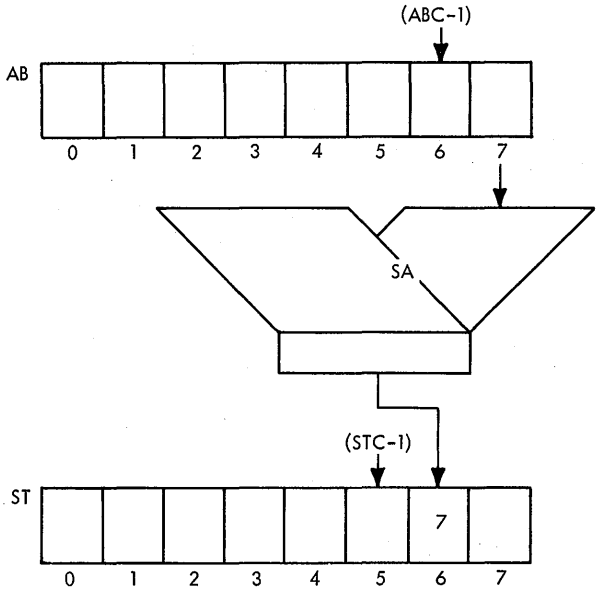
- Word overlap condition exists when: IC(0–20) = D(0–20) and IC(21–23) > D(21–23).
- Test for word overlap is performed by GIS of all one-operand instructions.
- Execution of one-operand instructions provides separate microprogram to handle word overlap conditions.
- No special action is taken to detect word overlap in two-operand instructions.
- Word overlap in two-operand instructions causes data program interruption.

Data is fetched from and placed into main storage one doubleword at a time. However, program compatibility of the 2065 CPU with smaller models in the System/360 requires that all results placed into main storage must be considered to be stored one byte at a time as they are processed. There are some cases where this compatibility would not be maintained unless special actions were taken. The condition that requires special handling is called "word overlap" and occurs when the fields of the first and second operands specified by the instruction overlap.

The operand addresses and field lengths may be such that one or more bytes in main storage are specified as part of both the first and the second operands. For example, consider the case in which the IC and D specify the same doubleword in storage; the IC specifies byte 7 as the starting second operand byte to be processed in this doubleword, and D specifies byte 6 as the starting first operand address. At least two operand bytes are to be processed.



This doubleword is fetched from main storage and placed into AB and ST. A one-operand instruction, such as Move with Offset, will process the first AB byte (byte 7) in the serial adder and place the result into the designated first operand byte; i.e., byte 6 in ST. Then, ABC and STC are reduced one count to designate the next AB byte to be processed and the ST location into which the results must be placed.



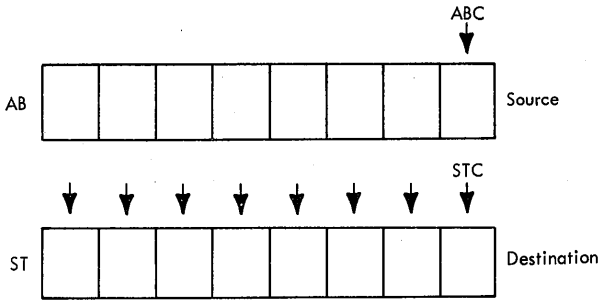
Note, however, that the preceding move operation has replaced the original contents of byte 6 in ST with the contents of byte 7. Thus, the next AB byte to be processed (original byte 6) is no longer valid and must be updated; i.e., the equivalent of storing ST byte 6 and then refetching this byte to AB must be performed.

As seen from the preceding example, the word-overlap condition may require special handling of data. Execution of all one-operand decimal instructions (Pack, Unpack, Move with Offset, and Zero and Add) provides for two

alternate microprograms. One microprogram is for the normal, or not-word-overlap, case; the other handles the word-overlap condition. Selection of the appropriate microprogram is dependent on the outcome of the word-overlap test, which is performed in the General Initialization Sequence (GIS) of all one-operand instructions.

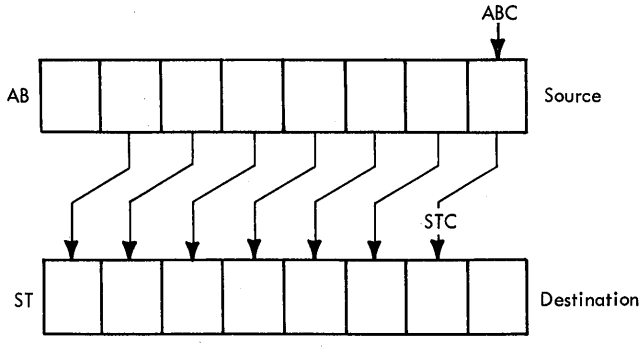
A word-overlap condition exists when both operands have the same doubleword address. The manner in which the first and second operand bytes are specified within this doubleword determines whether special data handling is required. When the word-overlap condition exists, three cases of byte specification may be distinguished:

1. The first and second operand bytes are the same; no special data handling is required.†



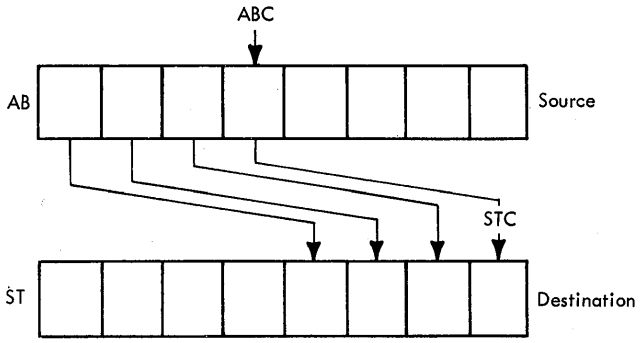
In this case, the destination bytes are placed into the same locations from which the source bytes are obtained. Because processing of any source byte does not affect the contents of the next source byte, no updating of source bytes is necessary.

2. The first operand bytes are specified "ahead" of the second operand bytes; special data handling is required.



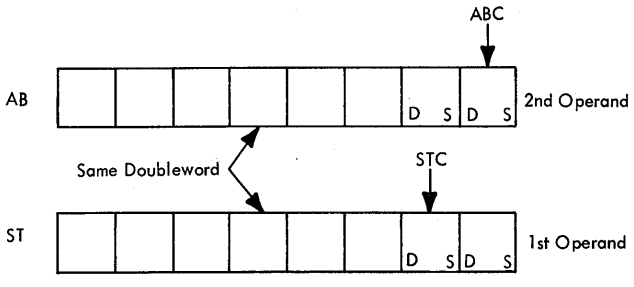
In this case, the destination bytes are placed into the locations from which the next source bytes will be processed. The data in AB becomes "obsolete" after processing of one or more source bytes. (The crossover point at which data becomes obsolete depends on the amount of skew between ABC and STC.) This word overlap case requires special data-handling techniques.

3. The first operand bytes are specified "behind" the second operand bytes; no special data handling is required.



In this case, the destination bytes are placed behind the source bytes as they are processed. Thus processing of any source byte cannot affect the contents of the next source byte, and no updating is necessary.

In two-operand arithmetic instructions, no special action is taken to detect word overlap. Word overlap is ignored during execution of a Compare instruction, because this instruction does not store the results. The operand fields specified for Add, Subtract, Multiply, and Divide instructions either should not overlap at all or should have coincident rightmost bytes. The GIS for these instructions does not perform the word overlap test, because improper overlap of the operands causes an invalid data condition to be detected in the execution phase. In two-operand instructions, the operand fields are correctly specified when the rightmost byte of each operand contains the operand sign; all bytes to the left of the sign byte must contain only digits. This requirement cannot be fulfilled when both operands in the instruction specify the same doubleword with different byte addresses. The following example shows that the sign byte of the first operand is also the "digit" byte of the second operand.



During execution of the instruction, all operand digits are checked for validity. Detection of a sign code in the digit position forces a data program interruption.

† Except for Unpack instruction. This instruction generates two bytes of destination for each byte of source and requires special data handling in all cases of word overlap.

GENERAL INITIALIZATION SEQUENCE

At the completion of the SS I-Fetch, the CPU is in the following status:

1. A main storage request for the doubleword containing the low-order byte of the first operand has been issued per D.
2. D contains the low-order byte address (contents of GPR addressed by B1, + D1, + L1) of the first operand.
3. The IC contains the high-order byte address (contents of GPR addressed by B2, + D2) of the second operand.
4. The next instruction address has been transferred from the IC to the LSWR.
5. E(0–7) contains the instruction op code.
6. The 'PSC' trigger has been set, if appropriate.

Following the SS I-Fetch, a branch is made per the instruction op code to the appropriate General Initialization Sequence (GIS). The general functions of the GIS for decimal instructions are described below. Functions peculiar to a specific instruction are covered in subsequent paragraphs dealing with the execution of that instruction.

The function of the GIS microprogram is to set up initial conditions for the execution phase. These include:

1. Gating the first operand from the SDBO to ST. A D request for the doubleword containing the low-order byte of the first operand was issued during SS I-Fetch.
2. Adding of L2 field to IC. At the end of SS I-Fetch, the IC contains the address of the high-order byte of the second operand. To address the low-order byte of the second operand, the IC must be incremented by the L2 field during GIS.
3. Initiating a storage request per the IC for the second operand. GIS initiates a storage request for the doubleword containing the low-order byte of the second operand.
4. Setting of STC and ABC. The STC is set to the rightmost first operand byte in ST, the byte to be processed first. Because the address of the rightmost byte is specified by D(21–23), the STC is set per these bits. Similarly, the rightmost second operand byte is selected in AB by gating IC(21–23) to the ABC.
5. Gating the second operand from the SDBO to AB. An IC request for the second operand is issued during GIS. Subsequently, the GIS gates the second operand from the SDBO to AB.
6. Performing a sign handling function. For add-type instructions, the sign of the result is tentatively set to agree with the sign of the first operand before the execution phase. The GIS examines the signs in the rightmost bytes of both operands and establishes whether to perform a true add or a complement add operation. For multiply and divide operations, the sign of the result is determined during the execution phase by examining the appropriate STAT's, which have been previously set according to the signs of the two operands. Both operands are tested for an invalid sign.
7. Performing the word overlap test. A word overlap test is performed during the GIS for Pack, Unpack, Move with Offset, and Zero and Add instructions.

ADD, SUBTRACT, AND COMPARE

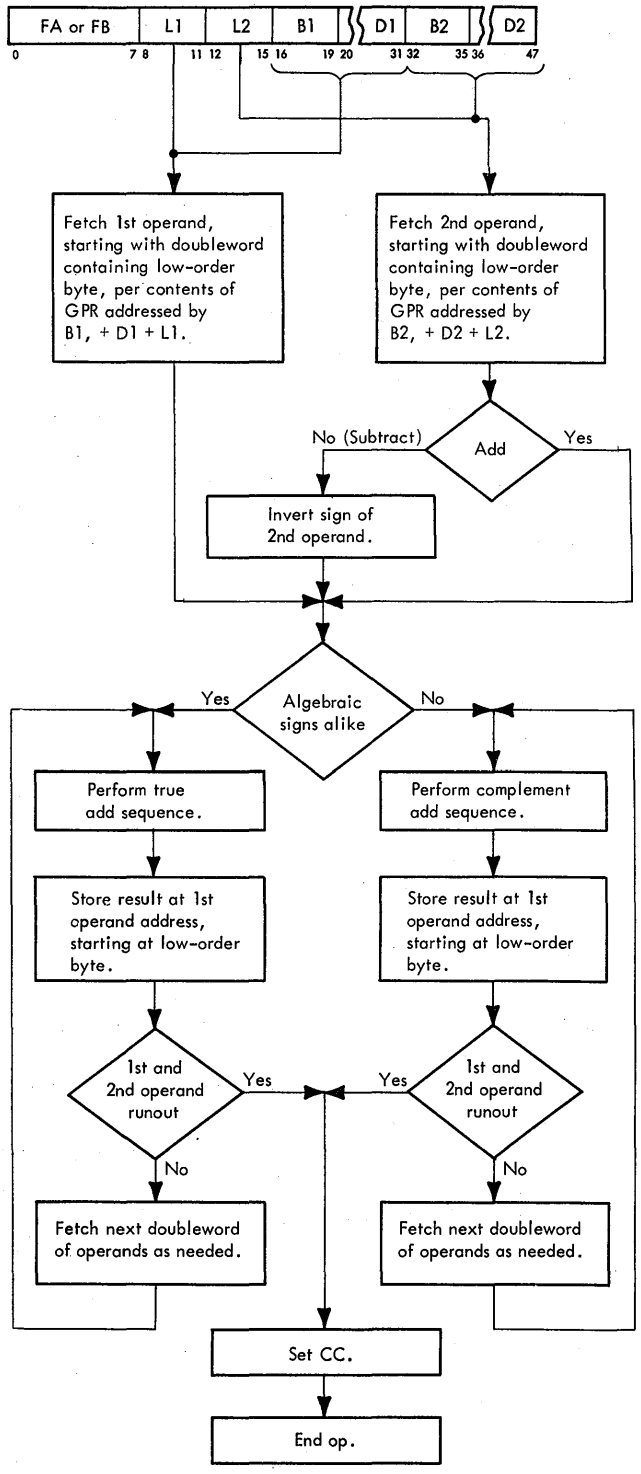
During the GIS, separate sign handling is performed for the Add, Subtract, and Compare instructions; the sign of the second operand is, in effect, inverted for Subtract and Compare instructions. After exit from the GIS, the three instructions share a common true add or complement add routine, depending on the operand signs and the instruction. Because the result of a Compare instruction is not stored into main storage, the setting of the mark triggers and overflow detection are inhibited in hardware during execution of this instruction.

Add, AP (FA) and Subtract, SP (FB)

- Algebraically add (subtract) 2nd operand (in storage) to (from) 1st operand (in storage) and place result into 1st operand location.
- SS format: (See following page.)
- CC setting:
  Result is zero: CC = 0.
  Result is less than zero: CC = 1.
  Result is greater than zero: CC = 2.
  Overflow: CC = 3.

The Decimal Add and Subtract instructions specify an algebraic addition with the sign of the second operand inverted for the subtract instruction. The sign of the result is tentatively set to agree with the sign of the first operand. Then, if the algebraic signs of the two operands are alike, a true add sequence is performed; if unlike, a complement add sequence is performed. The tentative result sign is correct in all cases except for a complement add operation where the magnitude of the second operand is equal to or greater than the magnitude of the first operand. If the magnitudes of the two operands are equal, the result is zero and a positive sign is stored into the result field. If the magnitude of the second operand is greater than that of the first, a complement result is formed and the sign of the result is inverted by setting it to the algebraic sign of the second operand when the result is recomplemented. All signs and digits are tested for validity. The operand fields may overlap when their low-order bytes coincide; therefore, it is possible to add a number to itself.

The result is stored at the first operand address. If the first operand field is too short to contain all significant digits of the result, a decimal overflow occurs.



GIS for Add and Subtract

The GIS microprogram for the Add and Subtract instructions is shown in Diagram 5-301, FEMDM. It performs the following functions:

1. Loads into ST the doubleword containing the low-order byte of the first operand.
2. Adds L2 to the IC, after which it requests, per the IC, the doubleword containing the low-order byte of the second operand. When this doubleword arrives, loads it into AB.
3. Transfers the L1 count to F(0–3).
4. Assigns a result sign that agrees with the sign of the first operand.
5. Performs a branch on the algebraic signs of the operands (contained in STAT's C and F) to enter the true add or complement add sequence.

At the start of GIS, the IC contains the address of the high-order byte of the second operand (contents of GPR addressed by B2, + D2). To obtain the address of the low-order byte, L2 is added to the IC, and a main storage request is issued per the result.

The L1 count in E(8–11) is destroyed during subsequent execution and must be preserved in F(0–3). This action is necessary because, upon execution of the instruction, it may be found that results were placed in main storage in complement form. Because the final result must be true, the destination field is refetched and recomplemented, and the sign is inverted. In such cases, the L1 count in F(0–3) is used to refetch the correct number of destination bytes.

The result is arbitrarily assigned the sign of the first operand by performing a branch of STAT's F and C. STAT F is set if the first operand is minus. Note, however, that the sign of the second operand is not known at this time and STAT C will always be in the reset state. Thus, when STAT's F and C are alike, it indicates that STAT F is not set (first operand plus); when the STAT's are not alike, it indicates that STAT F has been set and the first operand is negative.

A second branch on STAT's F and C is performed after the sign of the second operand has been sensed, and STAT C set accordingly. STAT C is set for a minus sign for an Add instruction and for a plus sign for a Subtract instruction. This is the only difference in the execution of an Add or Subtract instruction. If the STAT's are alike, a true add sequence is entered. Upon entry into this sequence, the result always carries the correct sign. If the STAT's are not alike, a complement add sequence is entered. In this case, the algebraic sign of the result cannot be known at the start of the operation because it is dependent on the relative magnitude of the operands. If the sign has been assigned incorrectly, the result of the complement add operation will be in complement form. This condition will be detected at the completion of the instruction, in which case the result will be recomplemented and the sign inverted by setting the result sign to the algebraic sign of the second operand (per STAT C).

True Add Sequence

- True +6 add operation exits on one or more of the following conditions:
  L1 or STC = 0.
  L2 = 0.
  ABC = 0.

- STAT A is set if result is not zero.
- STAT B is set if overflow occurs.
- STAT E is set if operand digit or sign is invalid.
- STAT G is set if Compare instruction.
- STAT H is set if carry to next byte occurs.

An overall flowchart of the true add sequence and the data path used for its execution are shown in Sheet 1 of Diagram 5-302, FEMDM. The flowchart outlines the major functional steps and sequences used in the Add, Subtract, and Compare microprogram.

Upon entry into the true add sequence, the signs of both operands have been examined (by the GIS) and the correct sign has been entered into the low-order destination byte in ST. The first step in the microprogram is to true-add the digits contained in the sign bytes of the operands. The result is then placed into the digit portion of the destination byte. At this point, one complete byte of the result has been developed. The operand length codes (L1 and L2) and the status of the byte counters (STC and ABC) are examined for one or more of the following exit conditions from the true add loop:

1. STC and L1 $\neq 0$, L2 = 0.
   The second operand field has run out.
2. STC and L1 $\neq 0$, L2 $\neq 0$, ABC = 0.
   More second operand bytes are needed.
3. STC or L1 = 0, L2 $\neq 0$, ABC = 0.
   The first operand field has run out, or ST is full and more first operand bytes are needed. In either case, ST must be stored into the destination address per D. More second operand bytes are needed.
4. STC or L1 = 0, L2 $\neq 0$, ABC $\neq 0$.
   The same conditions exist as in item 3 except that more second operand bytes are not needed.
5. STC or L1 = 0, L2 = 0.
   The first operand field has run out, or ST is full and more first operand bytes are needed. In either case, ST must be stored into the destination address per D. The second operand field has run out.

If none of the above exit conditions exist, the microprogram re-enters the true add loop to generate the next destination byte. L1, L2, STC, and ABC are decremented one count, the selected AB and ST bytes are added in the serial adder, and the result replaces the selected ST byte. After this, the status of all counters is again sensed for exit conditions.

Upon establishing one or more exit conditions, the operations dictated by the conditions are performed, and, if L1 is not zero, the true add loop is re-entered. When L1 is zero, all destination bytes have been processed. The microprogram then performs an overflow test and a test for all-zero result.† If an all-zero result has been obtained, the address of the low-order destination byte is restored in D and a plus is stored at this address. Restoration is necessary, because D is decremented by 8 for each doubleword of first operand that is fetched. If, for example, two doublewords of first operand have been fetched, the address of the low-order destination byte is obtained by adding 16 to D.

A detailed flowchart of the true add sequence is shown in Sheet 2 of Diagram 5-302. It is an expanded version of the overall flowchart, showing the data handling used in the various subroutines of the true add operation. This data handling is straightforward for the most part and requires no explanation. Those areas in need of clarification are discussed in the following subparagraphs.

True Add Operation. The selected AB byte is gated (true +6) to the serial adder and added as a binary number to the selected ST byte. The adder output is decimal-corrected at the input to SAL and gated back to the selected ST byte. For the first (or sign) byte, only bits 0–3 of the selected AB byte are gated to the adder. The decimal correction involves examining the carry from each digit and logically subtracting 6 from each result digit that did not have a carry. As each byte is processed, ABC, STC, L1 and L2 are decremented by 1 and the selected mark trigger is set (except for a Compare instruction). The carry from each byte is saved in STAT H and, if set, results in a carry to bit 7 of the next byte processed. STAT A is set if any nonzero result digit is detected. STAT E is set if any invalid digit is detected at the inputs to the serial adder.††

Exit Conditions. An exit is made when one or more of the following conditions exist as determined by a functional branch micro-order ('Decimal' micro-order):

1. L1 or STC = 0.
2. L2 = 0.
3. ABC = 0.

Five possible exit conditions exist:

1. STC and L1 $\neq 0$, L2 = 0.
   The second operand field has been completely processed. AB and ABC are cleared, and the high-order source extend routine is started to process the remaining destination field.

† The Add, Subtract, and Compare instructions have the same microprogram. The Compare instruction does not store the result into main storage and, upon exit from the true add loop, enters the end-op sequence.

†† When an invalid digit is detected at the serial adder in-buses, the adder forces 1's into its sum and parity output latches. This action insures that valid parity is always gated to ST from the serial adder.

2. STC and L1 ≠ 0, L2 ≠ 0, ABC = 0.
   A second operand fetch sequence is started to fetch the next doubleword of second operand to AB.
3. STC or L1 = 0, L2 ≠ 0, ABC = 0.
   The first operand has either run out, or ST is full and the next doubleword of the first operand is needed. In either case, ST must be stored if an Add or Subtract instruction is being executed. (If a Compare instruction is being executed, no mark triggers are set and ST is not stored.) The next doubleword of the second operand is needed.

   After the destination store cycle, L1 is tested for all 1's to determine whether the L1 field has run out. If L1 equals all 1's, a second operand fetch is initiated before entering the first operand runout sequence; if L1 does not equal all 1's, the first operand fetch sequence is performed, followed by the second operand fetch sequence and resumption of the true add loop.
4. STC or L1 = 0, L2 ≠ 0, ABC ≠ 0.
   The same conditions exist as in item 3 except that a second operand fetch is not needed and is not performed.
5. STC or L1 = 0, L2 = 0.
   A storage request per D is issued to store ST into the destination field (unless a Compare instruction is being executed). AB and ABC are cleared to start the high-order source extend routine. A further test is required to determine whether L1 was zero.

   If L1 is now all 1's, all destination bytes have been processed. A carry from the last destination byte indicates an overflow condition, and STAT B is set.

   If L1 is not all 1's, a first operand fetch sequence is started, after which the high-order source extend routine is resumed.

First Operand Fetch. A separate entry is made into this routine, per STAT G, for a Compare instruction. In a compare operation, a D request for the next doubleword of first operand has already been given. D is decremented by 8, and the doubleword arriving at the SDBO is gated to ST.

For Add or Subtract instructions, D is decremented by 8, and a D request is made for the next doubleword of first operand. F is incremented by 1 to record the number of fetches made. This information will be required to restore the low-order address in D in case an all-zero result is obtained. If ABC equals 111, a second operand fetch routine is started. If ABC does not equal 111, the appropriate addition or high-order source extend is started.

Second Operand Fetch. The IC is decremented by 8, and the next doubleword of second operand is fetched to AB. After this, the appropriate addition or overflow routine is started as determined by the L1 count.

Second Operand Runout. An all-zeros AB byte is gated true +6 to the serial adder and added to the selected ST byte. The result is decimal-corrected and gated back to the selected ST byte. STAT's A, E, and H and the mark triggers are set as previously explained.

L1 and STC are decremented by 1 as each byte is processed; ABC and L2 are not stepped. The sequence is repeated until L1 is stepped to zero, with an exit to the destination store and first operand fetch sequences whenever STC equals 7. A carry from the last destination byte is an overflow condition and sets STAT B. The end-op sequence is started when L1 equals zero.

First Operand Runout and Overflow Test. An overflow condition exists whenever a carry results as the last destination byte is processed or whenever a nonzero digit is detected in the source field after the destination field has been processed. STAT B is set if STAT H is set when entering this routine. Next, the remaining second operand bytes are gated true +6 to the serial adder with STAT B being set if any nonzero bytes are detected.

ABC and STC are decremented by 1 as each byte is processed. The next source doubleword is fetched to AB whenever ABC is stepped to zero unless L2 equals zero. When L2 is stepped to zero, the end-op sequence is started.

Zero Result. If at the completion of the true add operation STAT A is not set, an all-zero result has been obtained. In this case, the Add and Subtract instructions always force a positive sign into the low-order byte of the destination field. (If STAT G is set, an exit is made to the end-op sequence because no correction of the result is required for a Compare instruction.)

The low-order destination address is regenerated by adding 8 to D the number of times indicated in F(4–7). STC is set per D(21–23), and the selected ST byte is cleared.

STAT B is examined to determine an overflow condition. For a zero result and no overflow, a plus sign is inserted via the serial adder into the low-order destination byte with the selected mark trigger being set. A storage request is given to store the sign into the destination field, and the end-op sequence is started. For a zero result and overflow, the destination sign is not corrected. The end-op sequence is started immediately.

End-Op Sequence. The instruction address (original content of the IC) is restored from the LSWR to the IC, and STAT G is reset. A data program interruption occurs if STAT E is set. A decimal overflow program interruption occurs if STAT B is set and STAT E is not set. The CC is set per hardware conditions as shown in Table 3-12.

Table 3-12. Condition Code Setting Per Hardware Conditions, Decimal Instructions

Hardware Conditions	Setting
Not STAT B • Not STAT A • (Add, or Subtract, or Zero and Add)	0
Not STAT A • Not STAT H • Compare	0
Not STAT A • STAT H • STAT F • Not STAT C • Compare	0
Not STAT A • STAT H • Not STAT F • STAT C • Compare	0
Not STAT B • STAT A • STAT F • (Add, or Subtract, or Zero and Add)	1
STAT A • STAT F • STAT H • Compare	1
STAT A • STAT C • Not STAT H • Compare	1
STAT F • STAT C • STAT H • Compare	1
Not STAT B • STAT A • Not STAT F • (Add, or Subtract, or Zero and Add)	2
STAT A • Not STAT F • STAT H • Compare	2
STAT A • Not STAT C • Not STAT H • Compare	2
STAT H • Not STAT F • Not STAT C • Compare	2
STAT B • (Add, or Subtract, or Zero and Add)	3

Note: • Designates logical AND connective.

Complement Add Sequence

- Complement add operation with exits on one or more of the following conditions:
  1. L1 or STC = 0.
  2. L2 = 0.
  3. ABC = 0.
- STAT A is set if result is not zero.
- STAT B is set if overflow occurs.
- STAT D is set if result must be recomplemented.
- STAT E is set if operand digit or sign is invalid.
- STAT G is set if Compare instruction.
- STAT H is set if carry to next byte occurs.
- Carry out of last destination byte indicates result is in true form; no carry condition indicates result is in complement form.

Diagram 5-303, Sheet 1, FEMDM, is an overall flowchart of the complement add sequence. This flowchart outlines the major functional steps and sequences used in the Add, Subtract, and Compare microprogram.

Upon entry into this sequence, the signs of both operands have been examined (by the GIS) and the sign of the first operand has been inserted as the sign of the result. This sign may or may not turn out to be the correct sign: if the first operand is larger than the second, the result carries the correct sign; if the reverse is true, the sign of the result must be inverted.

Basically, the complement add microprogram is similar to the true add sequence previously described. The first step in the microprogram is to complement add the digits contained in the sign bytes of the operands. The result is then placed into the digit portion of the destination sign byte. At this point, one complete byte of result has been developed. The operand length codes (L1 and L2) and the status of the byte counters (STC and ABC) are examined for one or more of the following conditions:

1. The result byte is contained in the last byte of ST and must be stored (STC = 0).
2. Additional first operand bytes must be fetched from main storage (STC = 0 and L1 $\neq$ 0).
3. Additional second operand bytes must be fetched from main storage (ABC = 0 and L2 $\neq$ 0).
4. The second operand has run out; i.e., all second operand bytes have been processed, and zeros must be added to the first operand bytes (L2 = 0 but L1 $\neq$ 0).
5. The first operand has run out; i.e., the destination field has been completely processed (L1 = 0).

If none of the above conditions exist, the microprogram enters the complement add loop to generate the next destination byte. L1, L2, STC, and ABC are decremented one count, the selected AB byte is complement-added to the selected ST byte, and the result replaces the selected ST byte. After this, the status of all counters is again sensed for exit conditions.

Upon establishing one or more exit conditions, the operations dictated by the conditions are performed and, if L1 is not zero, the complement add loop is re-entered. When L1 is zero, all destination bytes have been processed. The microprogram then performs an overflow test, a zero-result test, and a complement result test.† If an

† The Compare instruction does not store the result into main storage and, upon exit from the subtract loop, enters the end-op sequence.

all-zero or complement result has been generated, the address of the low-order destination byte is restored to D. Then, the result is either recomplemented and the sign inverted or a plus is stored into the low-order destination byte.

Sheet 2 of Diagram 5-303 is a detailed flowchart of the complement add sequence. It is an expanded version of the overall flowchart, showing the data handling in the various subroutines of the complement add operation. The complement add operation is similar to the true add sequence described previously. For this reason, only the differences are discussed below.

Complement Add Operation. The selected AB byte is converted to 2's complement form at the input to the serial adder, and is then added to the selected ST byte. For the first (or sign) byte, bits 0–3 only of the selected AB byte are gated in inverted binary form to the adder, with a hot carry supplied to bit 3 to convert to 2's complement.

Second Operand Runout. An all-zeros AB byte is gated in complement form to the serial adder and added to the selected ST byte. Thus the second operand is extended with high-order binary 1's (decimal 9's + 6).

Overflow Test. Generally, an overflow condition exists, if, upon processing all destination bytes, a non-zero source byte is detected. One exception occurs when the first source byte sensed, after the first operand has run out, equals 1 and a "carry" condition exists. A carry condition is determined by STAT's A and H being set; i.e., a nonzero result and a carry from the previous byte.

When L2 has been stepped to zero, a carry from the last destination byte is examined. A carry condition indicates a true result, and the end-op sequence is started. No carry indicates a complement result, and a recomplement sequence is started for Add and Subtract instructions. For a Compare instruction (STAT G set), the end-op sequence is started immediately.

Zero Result and Recomplement Setup. STAT D is set when an entry is made to this routine because of the result's being in complement form. STAT D is not set when an entry is made because of a zero result. If STAT G is set when entering this routine, an exit is made to the end-op sequence because no corrections of the result are required for the Compare instruction.

The low-order destination address is regenerated by adding 8 to D the number of times indicated in F(4–7). STC is set per D(21–23), and the selected ST byte is cleared. If an overflow condition exists (STAT B set), the results need not be corrected and the end-op sequence is started immediately.

Recomplement Sequence. The original L1 count was saved in F(0–3) by the GIS. This count is now placed into the L2 location in E; i.e., E(12–15). The L1 location in E(8–11) is set to zero and then decremented one count to provide an exit from the first operand fetch routine to the recomplement sequence. The complement result is gated from the SDBO to AB, and the recomplement sequence is started.

The sign byte is processed by gating bits 0–3 of the selected AB byte in inverted binary form to the serial adder, with a hot carry supplied to bit 3. The sign is inverted by inserting the algebraic sign of the second operand into serial adder bits 4–7, as determined by STAT C. Bits 0–3 are decimal-corrected, and the adder output is gated to ST.

All bytes following the sign byte are processed by gating the selected AB byte complement to the serial adder, where it is added to an all-zero ST byte. The adder output is decimal-corrected and gated back to the selected ST byte.

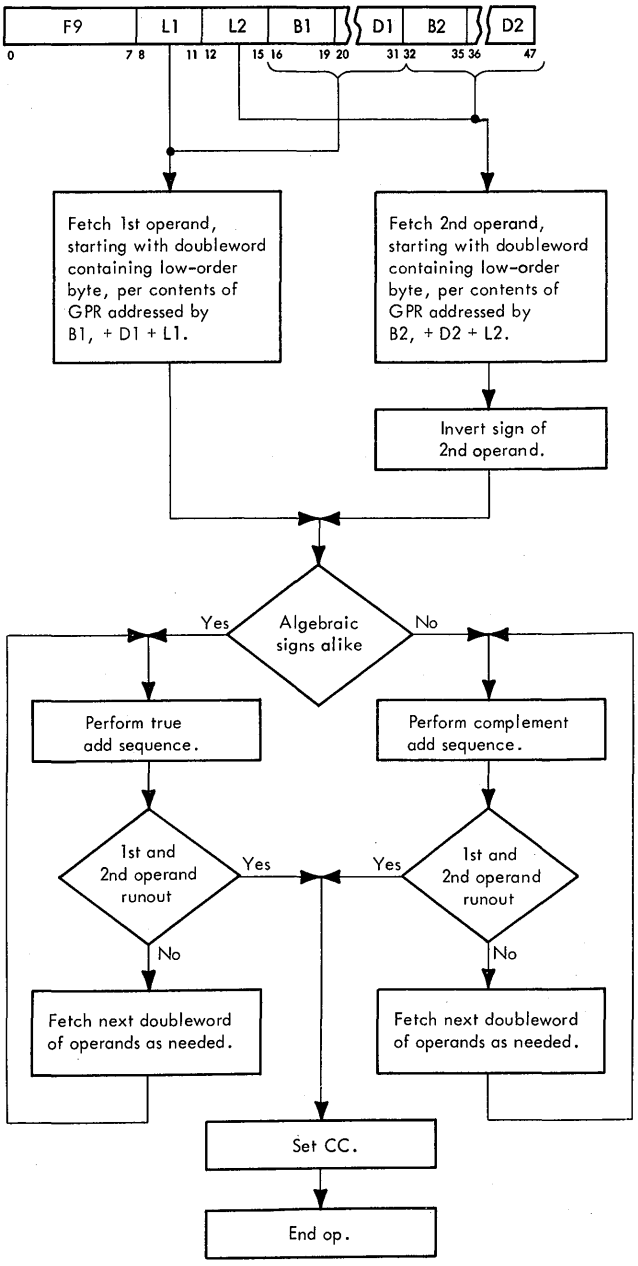
As each byte (including the sign byte) is processed, the ABC, STC, and L2 counts are decremented. The mark trigger selected by the STC is set. The serial adder carry is saved in STAT H. STAT A is set on nonzero digits, and STAT E is set on invalid digits.

Recomplementation is continued until the L1 in E(12–15) is stepped to zero. If ABC steps to zero and L1 is not zero, ST is stored and the next doubleword of destination is fetched to AB. When L1 steps to zero, ST is stored into the destination field, AB is cleared, and STAT F is set or reset per STAT C. The CC is set per hardware conditions (see Table 3-12), and the instruction is ended.

Compare, CP (F9)

- Algebraically compare 1st operand (in storage) with 2nd operand (in storage) and set CC according to result.
- SS format: (See left column of next page.)
- CC setting:
  Operands are equal: CC = 0.
  1st operand is less than 2nd operand: CC = 1.
  1st operand is greater than 2nd operand: CC = 2.

The CP instruction shares the same true add and complement add routines used by the Add and Subtract instructions. The GIS microprogram for the Compare instruction is shown in Diagram 5-301. As in the Subtract instruction, this microprogram effectively inverts the sign of the second operand by setting STAT C on a positive sign. The result of the compare operation is not placed

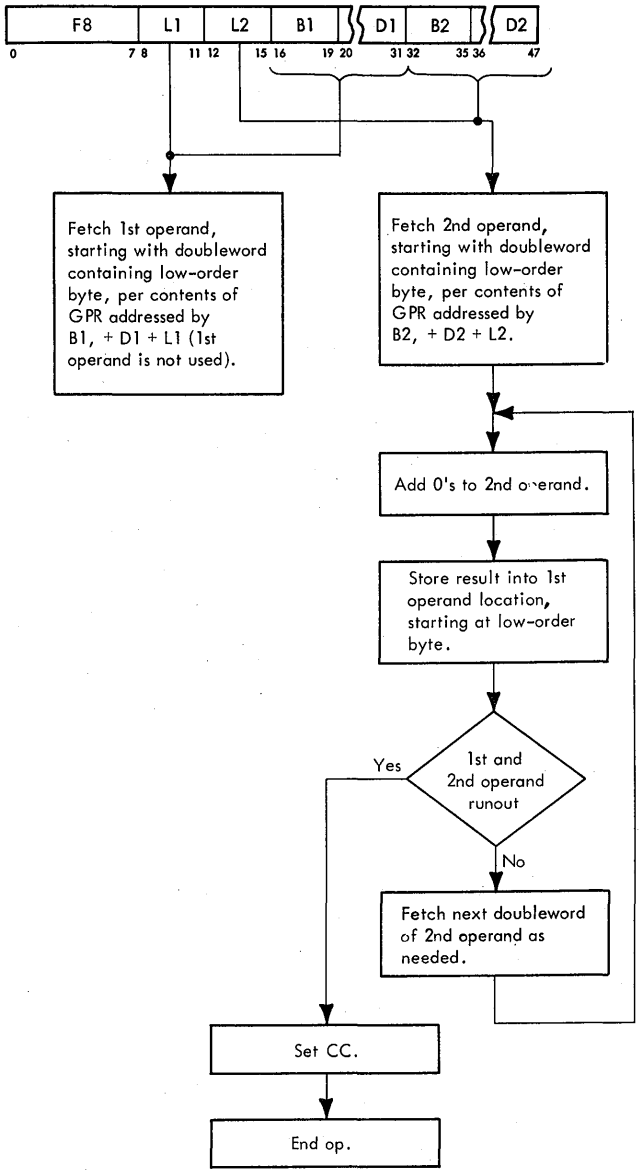


into main storage. STAT G is set to provide a means of taking special action, where required for the Compare instruction, during execution of the common true add or complement add sequences.

ZERO AND ADD, ZAP (F8)

- Place 2nd operand (in storage) into 1st operand location (in storage).

- SS format:



- CC setting:
  2nd operand is zero: CC = 0.
  2nd operand is less than zero: CC = 1.
  2nd operand is greater than zero: CC = 2.
  2nd operand cannot fit into destination field: C = 3.

The operation specified by the ZAP instruction is equivalent to addition to zero. A zero result is always made positive. When high-order digits are lost because of overflow, a zero result has the sign of the second operand.

Only the second operand is checked for valid sign and digit codes. Extra high-order zeros are supplied if needed. When the first operand field is too short to contain all significant digits of the second operand, a decimal overflow occurs and results in a decimal overflow program interruption, provided that the decimal overflow mask bit, PSW(37), is 1. The first and second operand fields may overlap when the rightmost byte of the first operand field is coincident with or to the right of the rightmost byte of the second operand. A flowchart of the GIS and execution of the Zero and Add instruction is shown in Diagram 5-304, FEMDM.

At the start of the GIS, the following actions have been performed by SS I-Fetch: (1) a D request has been issued for the doubleword containing the low-order byte of the first operand; (2) the low-order byte address of the first operand (contents of GPR addressed by B1, + D1 + L1) is in D; and (3) the high-order byte address of the second operand (contents of GPR addressed by B2, +D2) is in the IC. During the GIS, the doubleword containing the low-order byte of the first operand is gated from the SDBO to ST, the IC is incremented by the L2 count to address the low-order byte of the second operand, the STC is set to the low-order destination byte, an IC request for the second operand is issued, and a word overlap test is performed. If a word-overlap condition is predicted by this test, the instruction address is restored to the IC, the 'invalid data interrupt' trigger is set, and the instruction is ended. If no word-overlap condition is detected, the doubleword containing the low-order byte of the second operand is gated from the SDBO to AB.

The sign byte is processed by gating bits 0–3 of the selected AB byte to the serial adder. Bits 4–7 of the AB byte are decoded for a positive, negative, or invalid sign. The approved plus or minus sign is inserted into SAL(4–7) and gated, with the digit, to the selected ST byte.

All bytes following the sign byte are processed by gating the selected AB byte true +6 to the serial adder. The selected ST byte is not gated to the adder, and the validity check at the adder B-side is inhibited in hardware. The adder output is decimal-corrected and gated to the selected ST byte.

As each byte is processed, including the first byte, the ABC, STC, L1, and L2 are decremented, the selected mark trigger is set, STAT E is set for invalid data, and STAT A is set for a nonzero digit.

The byte-by-byte transfer from AB to ST is continued until one or more of the following exit conditions are detected via a ROS branch ('Decimal' micro-order):

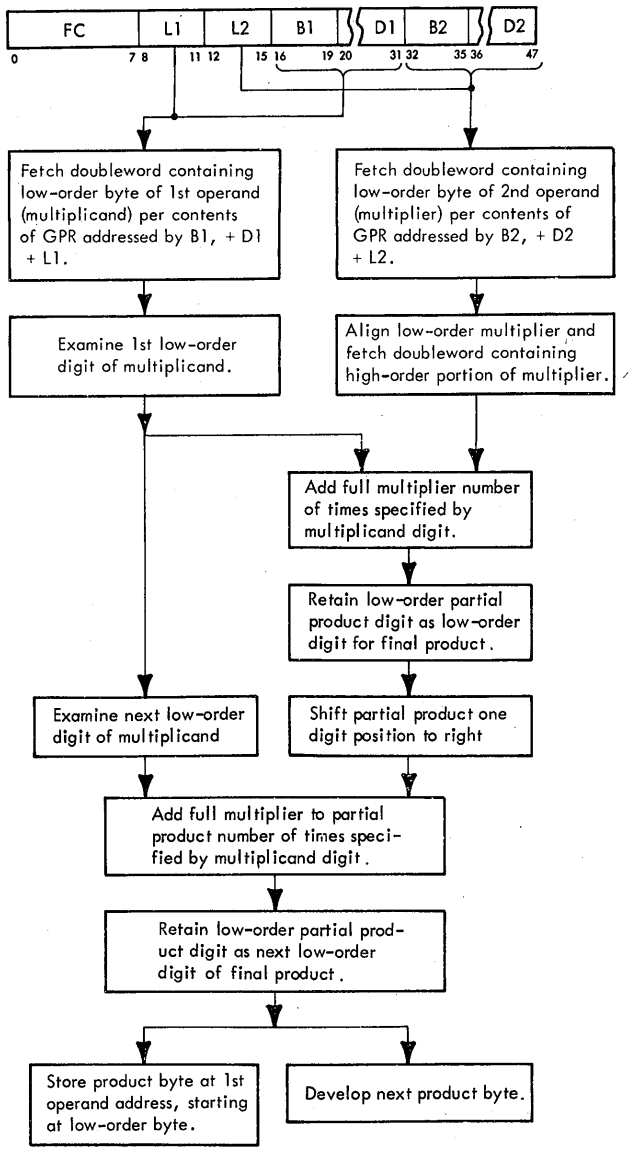
1. STC and L1 $\neq$ 0, L2 = 0.
   The second operand field has run out. AB and ABC are cleared and zeros are gated to ST per the STC until L1 or STC equals zero. If the STC is reduced to zero before L1, ST is stored per the D address, and zeros are gated to ST per the STC until L1 is reduced to zero.
2. STC and L1 $\neq$ 0, L2 $\neq$ 0, ABC = 0.
   More second operand bytes are needed. The next doubleword of the second operand is gated to AB and the zero and add loop is resumed.
3. STC or L1 = 0, L2 $\neq$ 0, ABC = 0.
   The destination field has run out or ST is full, requiring a destination store cycle. More second operand bytes are needed. ST is stored into the destination address per D, and the next doubleword of the second operand is fetched per the IC and gated to AB. If L1 equals all 1's (the destination field has run out), AB is tested for nonzero digits to determine if a decimal overflow condition exists. If L1 does not equal all 1's, the zero and add loop is resumed.
4. STC or L1 = 0, L2 $\neq$ 0, ABC $\neq$ 0.
   The same conditions exist as in item 3, except that a second operand fetch is not needed and is not performed.
5. STC or L1 = 0, L2 = 0.
   The destination field has run out or ST is full, requiring a destination store cycle. The second operand field has run out. ST is stored in the destination address per D. If L1 equals all 1's, both operands have run out. If L1 does not equal all 1's, the second operand runout sequence is performed.

MULTIPLY, MP (FC)

- Multiply 1st operand (in storage) by 2nd operand (in storage) and place result into 1st operand location.
- SS format: (See following page.)
- Maximum multiplicand field (1st operand) is 16 bytes.
- Maximum multiplier field (2nd operand) is 8 bytes.
- Multiplicand field initially contains high-order zero field equal in length to multiplier field.
- L2 $>$ 7 or L2 $\geq$ L1 causes specification program interruption.
- Multiplication accomplished by repetitive addition or subtraction:

Multiplicand Digit	Sequence Selected
0	–
1–4	Addition
5–9	Subtraction

The Decimal Multiply instruction replaces the multiplicand (1st operand) with the product of the multiplicand and the multiplier (2nd operand). To be able



to store the product in the multiplicand field at all times, several restrictions are imposed on both the multiplicand and the multiplier:

1. In any multiply operation, the maximum number of product digits that can be obtained is equal to the sum of the digits in the two operands. Because the product is stored in the multiplicand field, this field must initially contain high-order zero digits for at least a field size equal to that of the multiplier. Thus the multiplicand field is initially split into two parts; the high-order zero field of length equal to the multiplier, and the low-order field containing the effective multiplicand digits. This arrangement of the multiplicand ensures that product overflow will not occur (Figure 3-14).
2. By definition, the multiplier field must be at least one digit less than the multiplicand. Because the multiplicand must initially contain a zero field equal in size to the multiplier digits, the multiplier size is limited to 8 bytes (15 digits and sign). A specification program interruption occurs if the multiplier length code designates more than 8 bytes (L2 is greater than 7), or if L2 is greater than or equal to L1.
3. The maximum product size is 31 digits and sign (16 multiplicand digits plus 15 multiplier digits). The sign is determined algebraically from the multiplier and the multiplicand signs, even if one or both operands are zero. Because during sign resolution two sign positions are merged into one, at least one high-order digit of the product field is zero.

The multiply operation is executed in much the same manner as in manual arithmetic.† The multiplicand is examined one digit at a time, starting with the low-order digit, and the entire multiplier is added the number of times specified by the multiplicand digit. After the first multiplicand digit has been processed, the low-order digit of the resulting partial product (PP) is saved as the low-order product digit. The PP is then shifted one digit position to the right and brought into computation of the next product digit (one order higher than before). This time, the multiplier is added to the PP the number of times specified by the next digit of the multiplicand, and the low-order digit of the new PP thus formed becomes the next product digit. The PP is again shifted to the right, and the sequence is continued until all digits of the multiplicand have been processed. The PP resulting after the last multiplicand digit has been processed becomes the high-order product.

Figure 3-15 illustrates a typical repetitive addition sequence used for multiplication. As each multiplicand byte is processed, the multiplicand length code (L1) is reduced by one count and compared with the multiplier length code (L2). When L1 = L2, all effective multiplicand digits have been processed and the operation is completed.

To reduce the number of computations in the multiply operation, either a repetitive add or a repetitive subtract sequence may be performed. Selection of the sequence is dependent on the magnitude of the multiplicand digit under consideration. An add sequence is selected if the magnitude of the digit is in the range of 1 through 4. For multiplicand digits of magnitude 5 or greater, a subtract sequence is selected. This sequence deducts the multiplier from the PP the number of times specified by the 10's complement of the multiplicand digit and then adds 1 to the next digit of the multiplicand; increasing the next high-order digit of the multiplicand has the effect of

† The major difference is that the roles of the multiplicand and the multiplier are reversed. Because of its size (up to 16 bytes), the entire multiplicand cannot be held in the CPU at one time. For this reason, the full multiplier (up to 8 bytes) is fetched to the CPU and multiplied by the individual digits of the multiplicand, which is fetched from main storage 1 byte at a time.

adding the multiplier 10 times. For example, the equivalent of a multiplication by 7 is subtracting the operand 3 times to obtain a negative PP and then effectively adding the operand to the PP 10 times.

An example of a typical subtract sequence used for multiplication is shown in Figure 3-16. Note that the PP resulting from a subtract operation is in 10's complement form. When the 10's complement PP is shifted right, its high-order digit position must be extended with a 9.

Following are general and detailed descriptions of the multiply microprogram. The general description outlines the overall structure of the microprogram, enumerates its major functional steps and sequences, and explains their relationship to the overall operation. The detailed description analyzes each sequence individually, making specific references to the register-to-register data transfer in the CPU.

General Description

Upon entering the multiply microprogram, the following actions have been performed by SS I-Fetch:

1. A D request has been issued for the doubleword containing the low-order multiplicand byte.
2. The low-order multiplicand address has been placed into D.
3. The IC contents have been transferred to the LSWR, and the high-order multiplier address has been placed into the IC.

An overall flowchart of the multiply microprogram and the general data path used for its execution are shown in Sheet 1, of Diagram 5-305, FEMDM. The major subroutines and functional steps, shown in the figure, are explained below. Additional simplified diagrams are provided as an aid in visualizing the data handling performed. For the most part, these diagrams do *not* show the gates and data paths used in the CPU, but are intended solely to convey how the multiply algorithm is implemented. For purposes of illustration, a seven-byte multiplicand and a four-byte multiplier are assumed in these diagrams.

General Initialization Sequence

This sequence gates the multiplicand from the SDBO to ST and sets the STC to the low-order multiplicand byte. It increments the address in the IC to the address of the low-order byte of the multiplier by adding L2 to the IC contents. An IC request is issued for the multiplier (second operand), starting at the low-order address. The contents of D are transferred to the STC.

The GIS gates the multiplier from the SDBO to AB and sets the ABC to the low-order multiplier byte. It also performs several actions relating to the subsequent left-adjust sequence of the multiplier:

1. The low-order digit of the multiplicand (in ST) is transferred to F(0–3).
2. STAT F is set if the sign of the multiplicand is negative and STAT E is set if the sign is invalid.
3. The multiplier length code, L2, is transferred to F(4–7).

The functions performed by the GIS are illustrated in Figure 3-17.

Specification Test

This test verifies that the length codes for both operands in the instruction are correctly specified; i.e., L2 specifies eight bytes or less and is smaller than L1.

Incorrect Specification

Detection of an invalid specification forces a specification program interruption. The instruction address is restored from the IC to the LSWR, and the instruction is ended.

Multiplier Left-Adjust Sequence

The multiplier bytes are transferred from AB to ST in such a manner that the high-order multiplier byte occupies the leftmost byte in ST. STAT C is set if the multiplier sign is negative and STAT E is set if the multiplier sign is invalid.

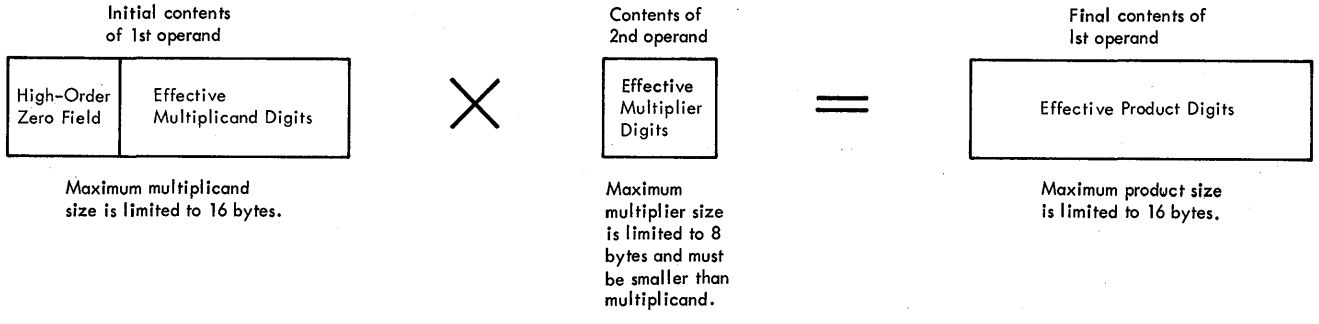


Figure 3-14. Operand Specifications for Decimal Multiply Instruction

Multiply (+204) by (-32) to obtain a product of (-6,528).

Execution:

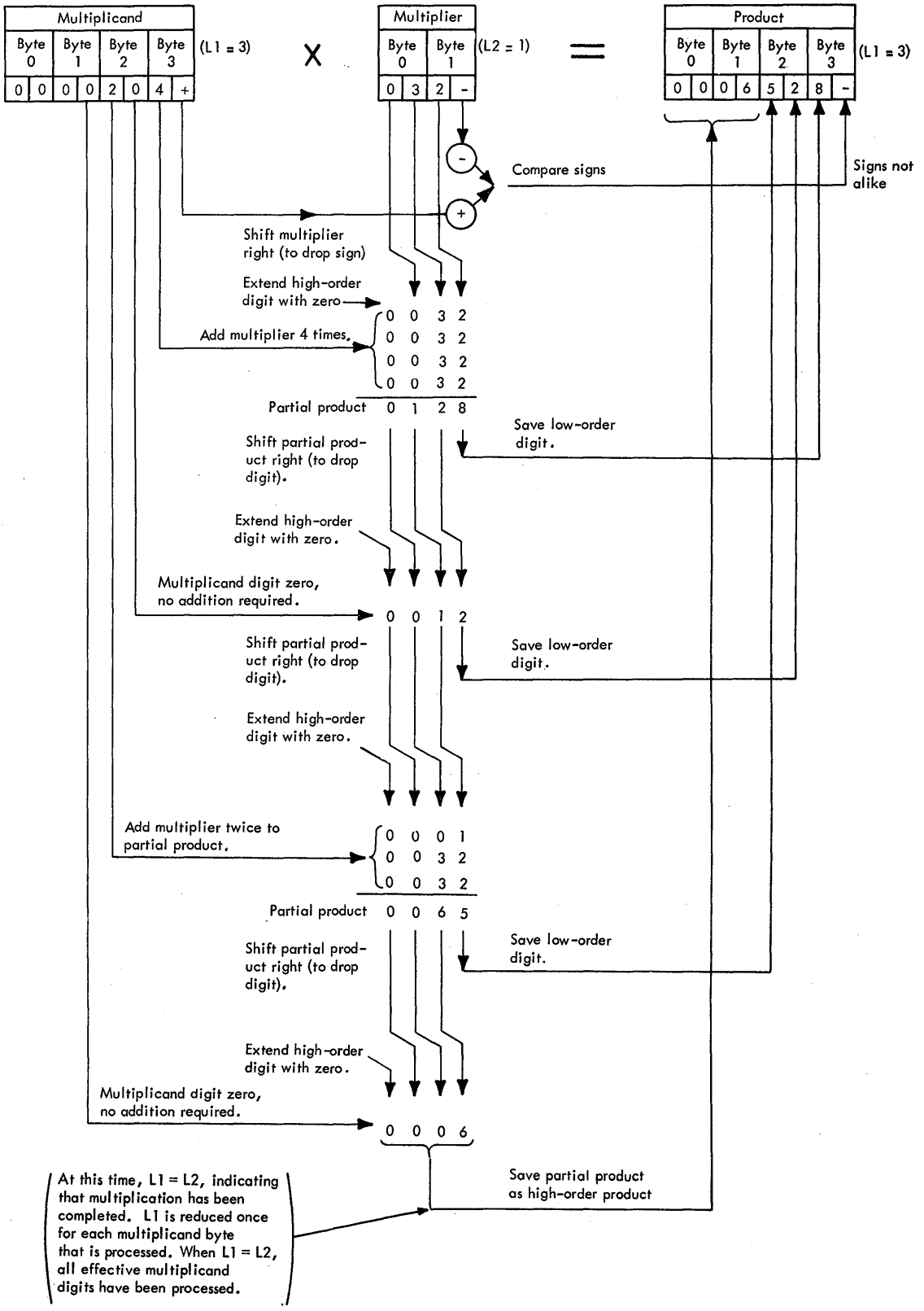


Figure 3-15. Typical Multiply Add Sequence

Multiply (+827) by (+25) to obtain a product of (+20,675).

Execution:

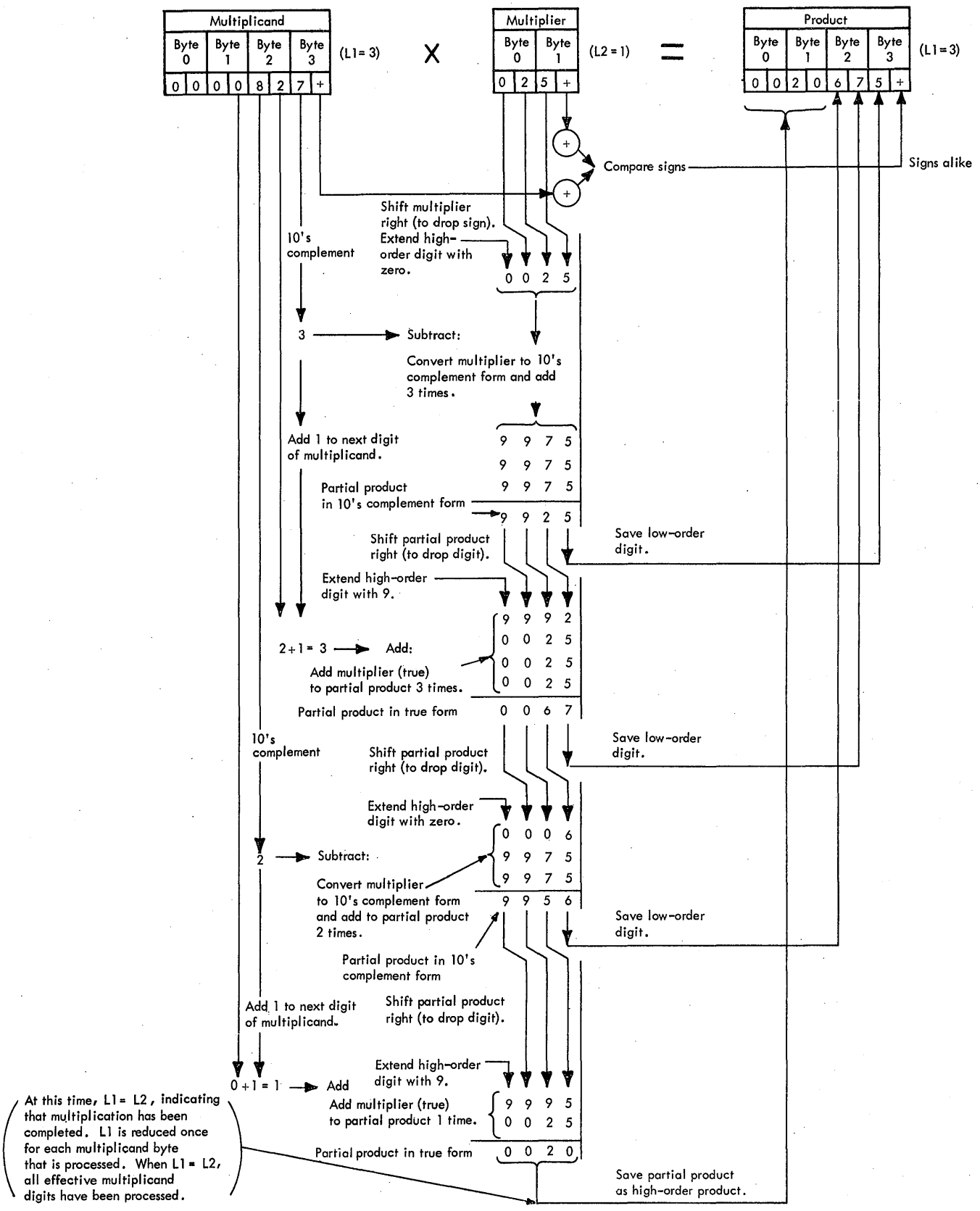


Figure 3-16. Typical Multiply Subtract Sequence

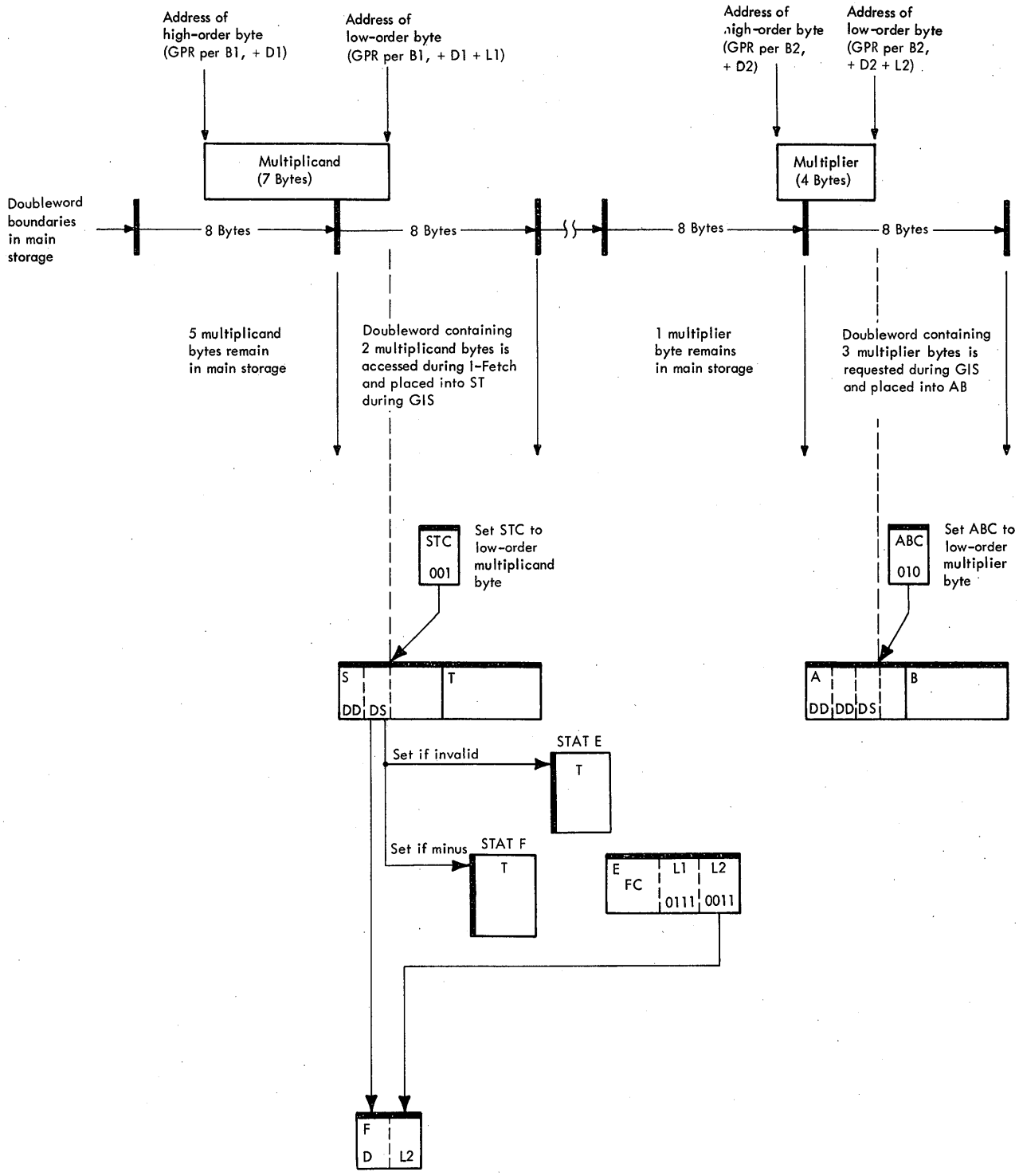


Figure 3-17. Data Handling During GIS of Decimal Multiply

The left-adjust transfer is initiated by setting the STC per L2 (Figure 3-18). Because the maximum multiplier length is limited to eight bytes, only three of the four bit positions in L2 are needed to specify the length code; i.e., the count in L2 may range from a minimum of 0000 (for 1 byte) to a maximum of 0111 (for eight bytes). Setting the STC per L2 automatically selects, according to the multiplier size, the correct ST position for the low-order byte of the multiplier; the number of bytes to the left of the selected ST position corresponds to the length field of the full multiplier.

The transfer is performed one byte at a time, through the serial adder, starting with the low-order multiplier byte. ABC, STC, and L2 are decremented by 1 for each byte transferred. The multiplier is completely transferred when the L2 count is decremented to zero. Because the first IC request (during I-Fetch) does not necessarily gate the full multiplier to AB, it may be necessary to fetch the balance of the multiplier from main storage. (This fetch occurs if the ABC steps to zero before L2 steps to zero.)

After exit from the left-adjust sequence, the full multiplier has been fetched and left-adjusted to ST. Note that the original ST contents (multiplicand) have been destroyed except for the low-order multiplicand byte, which is saved during the GIS; i.e., digit placed into F(0–3) and sign-recorded by STAT F. The destroyed multiplicand bytes are later refetched from main storage, one byte at a time, as required by the multiply operation.

L2 Restoration

During transfer of the multiplier bytes from AB to ST, the L2 count in E(12–15) is decremented by 1 for each byte transferred. At the completion of the left-adjust sequence, L2 has been decremented to zero. The initial L2 count, saved in F(4–7) during the GIS, is now restored to E(12–15). The L2 count will be required by the subsequent multiply sequence.

Multiplier Right-4 Shift to Drop Sign

In the multiply operation to follow, product bytes are developed by adding the entire multiplier the number of times specified by successive digits of the multiplicand. The sign of the multiplier does not enter into the

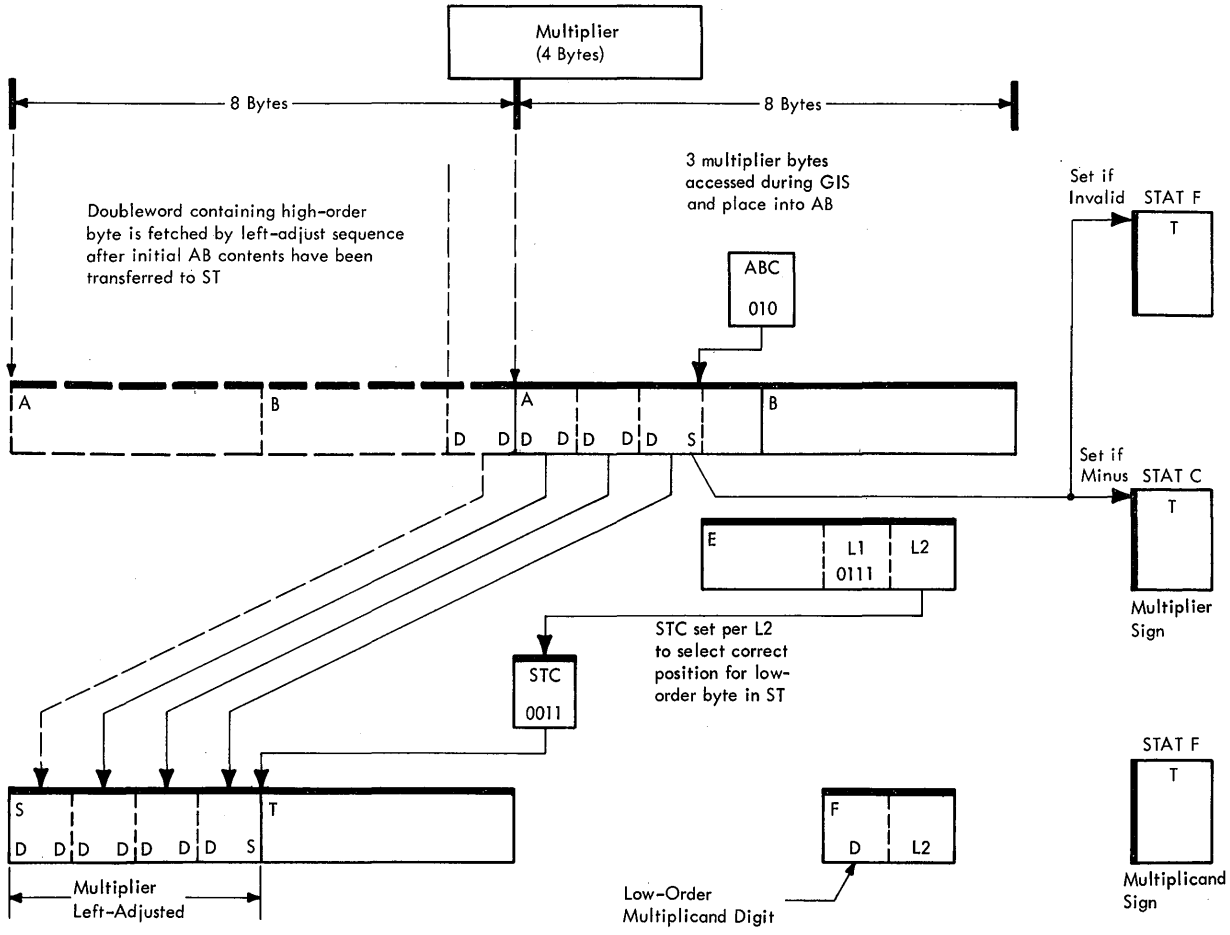
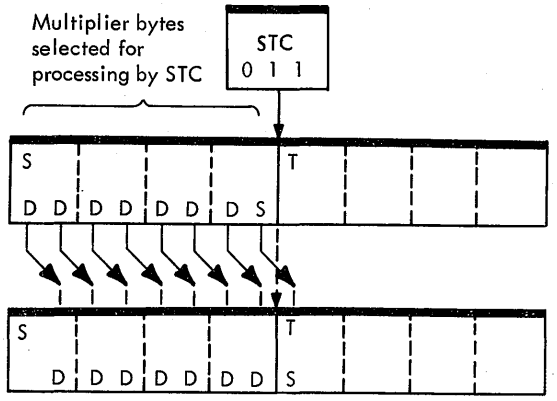


Figure 3-18. Data Handling During Multiplier Left-Adjust Sequence

repetitive addition sequence and must be discarded. The sign is discarded by shifting the multiplier in ST 4 bit positions (one digit) to the right as illustrated below. This action places the sign beyond the rightmost multiplier byte selected by the STC for subsequent computation.



Sign Handling

A test of STAT F and STAT C is made to establish the product sign algebraically:

1. Signs alike (both STATS set or reset) – set sign plus.
2. Signs not alike (one STAT set and the other reset) – set sign minus.

Upon establishing the correct product sign, it is placed into F(4–7).

Basic Multiply Add or Subtract Sequence for Left Digit

This sequence processes the digit in the left portion of the multiplicand byte. (The low-order byte of the multiplicand always contains the digit in the left portion and the sign in the right portion.) The entire multiplier (in ST) is added or subtracted the number of times specified by the left digit of the multiplicand saved in F(0–3). An add sequence is performed if the digit in F(0–3) is 4 or less; a subtract sequence, if 5 or greater. A data program interruption occurs prior to a storage cycle if an invalid multiplicand or multiplier digit or sign is detected. The PP resulting from the add or subtract sequence replaces the multiplicand in ST.

Product Byte Store

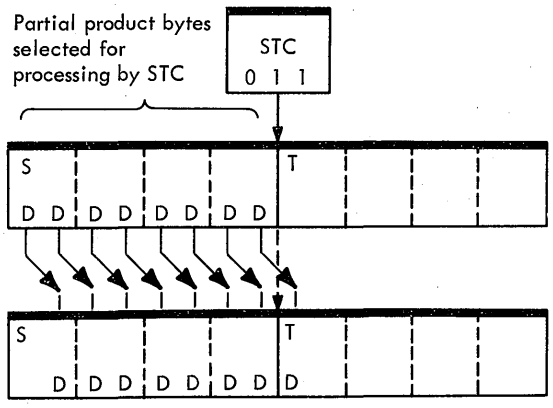
The product is stored into main storage one byte at a time. After exit from the left-digit sequence, one complete byte of product has been developed and must be stored. If the exit is made for the first time, this byte consists of the product sign (in F) and the low-order digit of PP (in ST). All product bytes generated thereafter consist of two digits: one (in F) has been saved from a previous PP developed in the right digit sequence, and the second is the low-order digit of a new PP (in ST) obtained in the left-digit sequence.

Multiplicand Request

A request from D is issued for the next byte of the multiplicand in main storage.

Partial Product Right-4 Shift to Drop Digit

The low-order digit of PP has been stored as a product digit and must not enter into subsequent computation. The digit is discarded by shifting the PP in ST four bit positions to the right. This action places the digit beyond the rightmost PP byte selected by the STC for computation of the next product digit.



L1 = L2

This test establishes whether all digits of the multiplicand have been processed. At the start of the multiply operation, the total field length specified by L1 includes a zero field equal in size to the multiplier plus the effective field of the multiplicand:

L1 = L2 + number of effective multiplicand bytes.

Because L1 is decremented once after each effective multiplicand byte is processed, all the effective multiplicand bytes have been processed when L1 equals L2.

Complete Multiplicand Byte Fetch

If L1 is not equal to L2, the multiply sequence is continued. The next byte of the multiplicand (requested earlier) is selected from the SDBO and placed into F. Control is then transferred to the add or subtract sequence for the right digit of the multiplicand.

Basic Multiply Add or Subtract Sequence for Right Digit, and Shift Right-4 Sequence

This sequence processes the digit in the right portion of the multiplicand byte. The entire multiplier is added to

(or subtracted from) the PP in ST. The number of add or subtract operations is controlled by the right digit of the multiplicand contained in F(4–7). After a new PP has been developed in ST, its low-order digit replaces the right digit of the multiplicand in F(4–7). The PP is then shifted four bit positions to the right to drop the low-order digit, and an entry is made to the left-digit sequence to process the next multiplicand digit contained in F(0–3).

Multiplicand Zero Test and Partial Product Store

When L1 equals L2, all the effective digits of the multiplicand have been processed. The remaining multiplicand bytes are fetched from main storage and tested for zero. Detection of a nonzero digit results in an interruption. After the zero test is completed, the PP is stored as the high-order product into main storage and the instruction is ended.

Detailed Description

- STAT E is set if digit or sign is invalid.
- STAT A is set if digit is not zero.
- STAT G is set if multiplier is zero.
- STAT H is set to generate hot carry.
- STAT D is set to add 1 to next digit.

Sheet 2 of Diagram 5-305 is a detailed flowchart of the multiply microprogram. This flowchart is an expanded version of the overall flowchart, showing the data handling used in the various subroutines of the Multiply instruction. For the most part, this data handling is straightforward and requires no explanation. Those areas in need of clarification are discussed in the following subparagraphs.

General Initialization Sequence

This sequence shares a common microprogram with the Divide instruction. An appropriate branch is taken to enter either the divide or the multiply sequence.

Multiplier Left-Adjust Sequence

The ABC has been set to select the low-order multiplier byte in AB. The STC is now set per L2, E(12–15). The transfer is performed one byte at a time via the serial adder. As each byte is gated to the serial adder, it is tested for nonzero value and for invalid digits. STAT E is set upon detection of an invalid digit or sign, and STAT A is set upon detection of a nonzero digit. If upon completion of the left-adjust transfer STAT A remains reset, the multiplier value is zero.

Multiplier Right-4 Shift and L2 Restoration

The multiplier is shifted four bit positions to the right and transferred from ST to AB. L2 is transferred from F(4–7) to E(12–15). Both actions are performed in parallel. As the high-order multiplier bytes are gated from S to PAA and the right-4 shift is initiated, the L2 count is transferred from F(4–7), via the serial adder, to S(28–31). After the right-4 shift has been performed through the parallel adder, the L2 count is gated from S to PAA and the zero count in E(12–15) is gated to PAB. The net result (original L2 count) is gated from PAL to E(12–15).

Sign Handling

STAT G is set if STAT A has not been set during the preceding sequence. This step is taken to indicate a zero multiplier condition which requires special action.

Basic Multiply Add or Subtract Sequence

To perform a branch on the value of the multiplicand digit, the digit must be in SAL(4–7). This requirement is dictated by the 'W=(1–15)' micro-order which samples SAL(4–7). For this reason, the contents of F are cross-gated through the serial adder and placed back into F. SAL(4–7) is then examined for the following values:

1. SAL(4–7) = 0
   No addition cycles are required.
2. SAL(4–7) = 1 through 4
   The multiplier in AB is added to the PP in ST the number of times specified by the digit value.
3. SAL(4–7) = 5 through 9
   The multiplier in AB is subtracted from the PP in ST the number of times specified by the 10's complement of the digit value (10 minus the digit value). STAT H is set to supply a hot carry for the subtract sequence. STAT D is set to add a 1 to the next digit of the multiplicand (equivalent to adding the multiplicand 10 times).
4. SAL(4–7) = invalid digit
   The definition of an invalid digit is dependent on whether the digit to be processed is the first digit of the multiplicand; i.e., the digit immediately following the sign. If it is the first digit, then any value in the range of 10 through 15 is considered invalid and sets the interrupt trigger. After the first digit has been processed, a value of 10 is permissible in SAL(4–7), provided that it was formed by an original value of 9 to which a 1 has been added because STAT D was set. Under these conditions, the value of 10 does not set the interrupt trigger, no addition cycles are required, and a carry is propagated to the next digit by setting STAT D.

The multiplier-to-PP addition or subtraction is done one byte at a time in the serial adder, with the AB byte gated true +6 if adding and complement if subtracting. The ABC and STC are both initially set to the L2 value and are decremented by 1 each time a byte is processed. When the ABC count is stepped to 000, F(4–7) is examined to determine whether further additions or subtractions are necessary. If so, the STC and ABC are again set to the L2 value, F(4–7) is incremented if subtracting or decremented if adding, and the multiplier is again added to or subtracted from the PP. The micro-order which steps the digit in F(4–7) is executed after the digit has been examined to determine whether further add or subtract cycles are required. For this reason, when a branch on F(4–7) is being made, a value of 1 when adding or of 9 when subtracting indicates that the multiplicand digit is completely processed. The low-order digit of the PP in ST is the product digit developed.

Product Byte Store, PP Right-4 Shift to Drop Digit, Multiplicand Request

These three functions are accomplished in parallel fashion. After initiating the store operation, control is transferred to the shift-right-4 sequence. When the ST contents have been temporarily transferred, the store operation is resumed; the product byte is cross-gated, transferred to ST, and stored into main storage per the D-address. Thereafter, the microprogram requests the next multiplicand byte from main storage and simultaneously completes the right-4 shift.

As illustrated in Figure 3-19, the PP is shifted right-4 via the parallel adder. This shifting is done in several steps, with the LSWR being used as temporary storage for the operand. Upon completion of the right-4 shift, B(64–67) is normally inserted as the high-order S digit. B(64–67) was previously set to 0 if the value in F(4–7) was less than 5, or to 9 if F(4–7) was 5 or greater, for then the PP was in 10's complement form. An exception is made when STAT G is set, indicating an all-zero multiplier. In this case, B(64–67) is not inserted into high-order PP because it should always be zero.

Complete Multiplicand Byte Fetch

If there are more multiplicand digits to be processed (L1 $\neq$ L2), the contents of T are temporarily transferred to the LSWR and either the left or the right half of the operand is gated from the SDBO to T. The next byte of the multiplicand is then selected per the STC and transferred to F. (Note that if the left half word has been gated to T, the high-order STC bit is forced to 1, because, otherwise, the STC would select a byte from S.)

Basic Multiply Add or Subtract Sequence for Right Digit, and Shift Right-4 Sequence

The next digit of the multiplicand in F(4–7) is examined, STAT's D and H are set or reset as required, a 0 or 9 is placed into B(64–67), and the appropriate add or subtract loop is entered. After exit from the add or subtract loop, the low-order digit of the resulting PP is saved in F(4–7). A right-4 shift is then performed on the PP in ST so that the low-order digit is dropped. At the completion of the right-4 shift, the left-digit sequence is resumed. As explained previously, the contents of F are cross-gated and the next digit of the multiplicand is sampled from SAL(4–7).

Multiplicand Zero Test and Partial Product Store

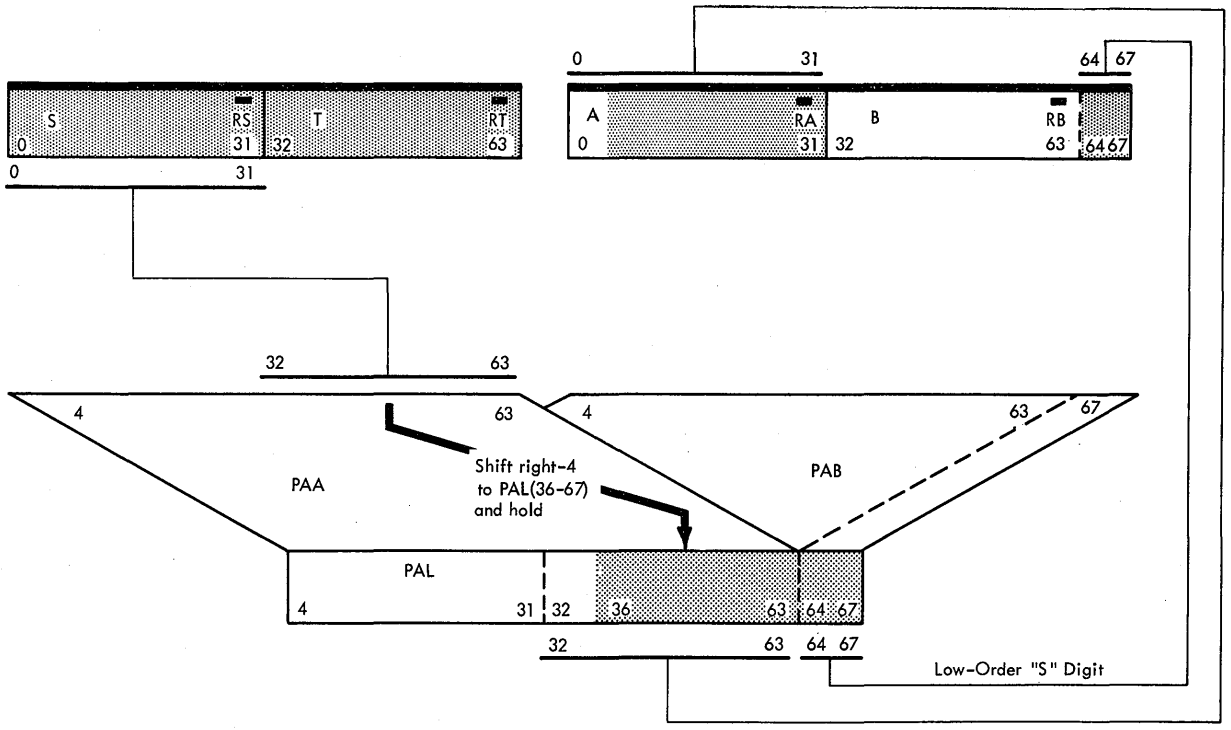
An entry to this routine is made from the left-digit sequence. By operand definition, the remaining high-order multiplicand bytes should all be zeros. The PP in ST is the high-order product and must be stored into the high-order portion of the initial multiplicand field. However, if STAT D is set at this time, the multiplier must be added to the PP once more. After this has been done, the contents of ST are transferred to AB and the high-order multiplicand bytes are fetched to ST (per the D-address). The STC is set per D(21–23) to designate the first high-order multiplicand byte to be tested for zero. The ABC is set per L2 to designate the first high-order PP byte in AB.

The selected multiplicand byte in ST is tested for zero; then the selected PP byte in AB is transferred via the serial adder to ST, and the corresponding mark trigger is set. ABC, STC, and L1 are decremented by 1 for each byte transferred. If a nonzero byte is detected in the high-order field of the multiplicand, the interrupt trigger is set and the instruction is ended.

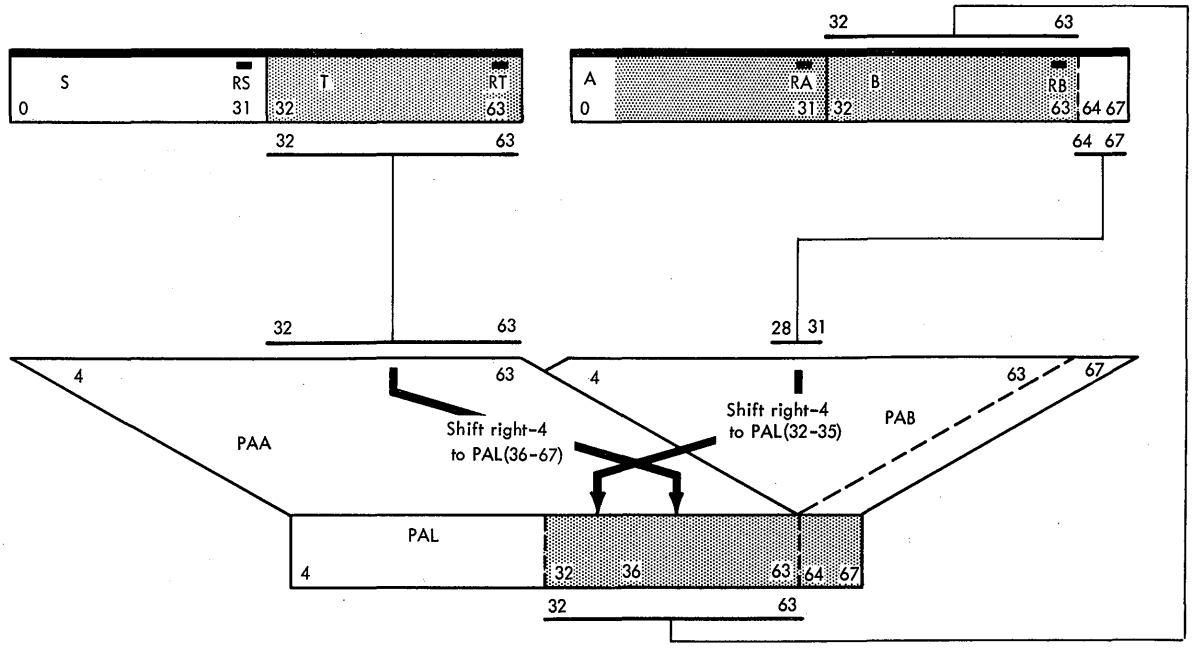
The AB-to-ST byte transfer is continued until L1 or STC is stepped to zero, at which time the ST contents are stored into main storage. If the STC has been stepped to zero (L1 $\neq$ 1111), the next high-order bytes of the multiplicand are fetched to ST and the sequence is resumed. If L1 has been stepped to zero (L1 = 1111), the instruction is ended.

DIVIDE, DP (FD)

- Divide 1st operand (in storage) by 2nd operand (in storage) and place result into 1st operand location (quotient is leftmost in 1st operand location; remainder, rightmost).



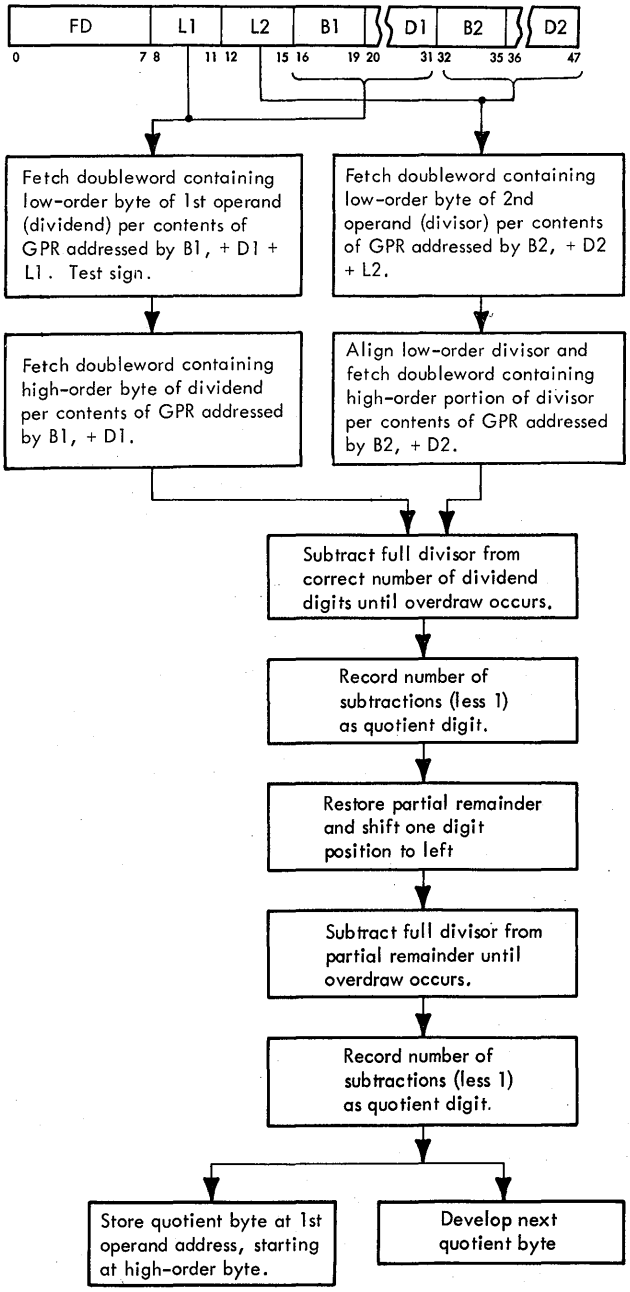
A. Step 1



B. Step 2

Figure 3-19. Data Flow for Right-4 Shift of ST to AB, Decimal Multiply

- SS format:



- Maximum dividend field (1st operand) is 16 bytes.
- Maximum divisor field (2nd operand) is 8 bytes.
- L2 specifies byte length for divisor and remainder.
- L2 $>$ 7 or L2 $\geq$ L1 causes specification program interruption.
- Division accomplished by repetitive subtraction.
- Dividend field must initially contain sufficient number of high-order zeros to make possible storing of quotient and remainder.

The Decimal Divide instruction replaces the dividend (1st operand) with the quotient and the remainder. To be able to store the quotient and the remainder into the dividend field at all times, several restrictions are imposed on the initial size of the dividend and the divisor (Figure 3–20).

The maximum dividend field is 16 bytes long. It is eventually replaced by the quotient, which is stored leftmost in the field, and by the remainder, which is stored rightmost. The size of the remainder is equal to the initial divisor size and is therefore predefined by length code L2. Because the minimum remainder size is 1 byte (L2 = 0), the maximum quotient size is limited to 15 bytes. By definition, the size of the divisor (and remainder) cannot exceed 8 bytes. A divisor greater than 8 bytes, or in excess of the dividend, is recognized as a specification error; the instruction is suppressed and a specification program interruption occurs.

The operand signs are tested for validity before instruction execution, and, if either sign is invalid, a data program interruption is taken before the contents of main storage are altered.

To make sure that the quotient and remainder will fit into the destination field, the magnitudes of the dividend and the divisor are compared before entering the divide sequence. This comparison, called "divide check" or "trial subtraction," yields the number of quotient digits that will result if division is carried out. If the predicted quotient is larger than that allowed, the instruction is suppressed and a decimal divide program interruption occurs. For this reason, an overflow condition cannot exist upon execution of a Divide instruction.

The dividend, divisor, quotient, and remainder are all signed integers, right-aligned in their fields. The sign of the quotient is determined algebraically from the dividend and divisor signs. The sign of the remainder is the same as the sign of the dividend. These rules hold true even when the quotient or the remainder is zero.

The divide operation is executed in much the same manner as in manual arithmetic. First, the divisor is properly aligned with the high-order dividend; then, by repeatedly subtracting the divisor from the dividend and counting the number of successful subtractions, the high-order quotient digit is determined. The partial remainder resulting from the last successful subtraction is shifted one digit position to the left, and the next lower-order dividend digit is inserted at the low-order end of the partial remainder. To obtain the next quotient digit, the divisor is again subtracted from the partial remainder. This sequence is repeated until all dividend

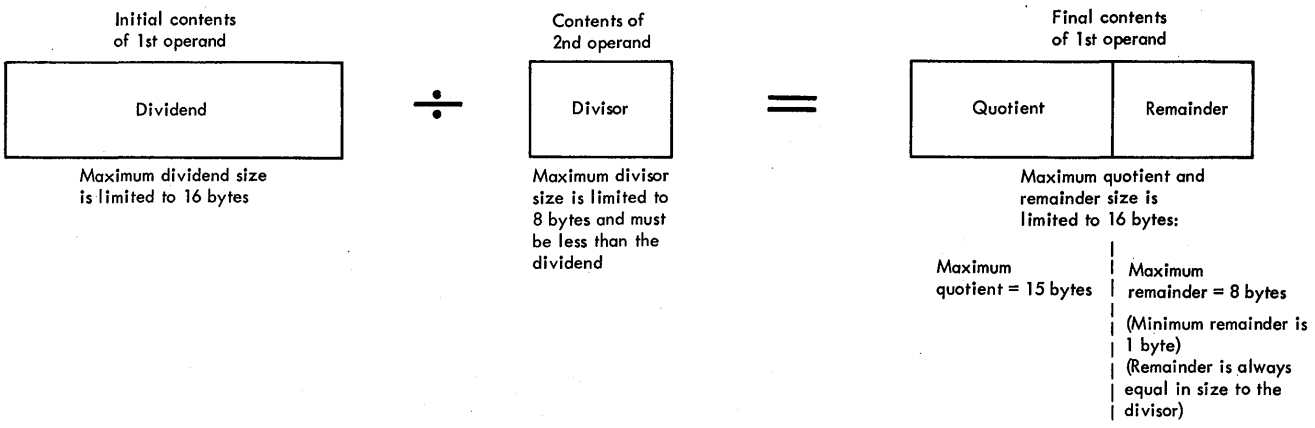


Figure 3-20. Operand Specifications for Decimal Divide

digits have been processed. The remainder resulting from the final successful subtraction is given the sign of the dividend and stored into the low-order end of the dividend field.

Figure 3-21 illustrates a typical repetitive subtract sequence used to accomplish division. Initially, a sufficient number of high-order dividend digits must be selected to perform the first successful subtraction. Successful subtractions of the divisor from the dividend occur until the partial remainder is overdrawn. The divisor is then added back once to restore the correct partial remainder. At the same time, the quotient digit is decremented by 1 to compensate for the overdraw. As each dividend byte is processed, the length code of the dividend (L1) is reduced by 1 and compared with the length code of the divisor (L2). Because the size of the remainder is also defined by length code L2, the condition of L1 equal to L2 indicates that all the effective bytes of the dividend have been processed, and the remainder is to be stored into the rest of the destination field. Note that, to be able to fit the quotient and the remainder into the destination field, this field must initially contain high-order zeros. A data program interruption occurs if the dividend does not have at least one leading zero.

Following are general and detailed descriptions of the divide microprogram. The general description outlines the overall structure of the microprogram, enumerates its major functional steps and subroutines, and explains their relationship to the overall operation. The detailed description analyzes each sequence individually, making specific references to the register-to-register data transfer in the CPU.

General Description

Upon entering the divide microprogram, the following actions have been performed by SS I-Fetch:

1. A D-request has been issued for the doubleword containing the low-order dividend byte.
2. The low-order dividend address has been placed into D.
3. The contents of the IC have been transferred to the LSWR, and the high-order divisor address has been placed into the IC.

An overall flowchart of the divide microprogram and the general data path used for its execution are shown in Sheet 1 of Diagram 5-306, FEMDM. The major subroutines and functional steps, shown in the figure, are explained in the following subparagraphs. Additional simplified diagrams are provided as an aid in visualizing the data handling performed. For the most part, these diagrams do not show the gates and data paths used in the CPU, but are intended solely to convey how the divide algorithm is implemented. For purposes of illustration, a nine-byte dividend and a three-byte divisor are assumed in these diagrams.

General Initialization Sequence

This sequence shares a common microprogram with the Multiply instruction. An appropriate branch is taken to enter either the divide or the multiply sequence. GIS gates the low-order dividend from the SDBO to ST, increments the IC by L2 to address the low-order byte of the divisor, issues an IC request for the divisor, and gates the divisor to AB. The ABC is set to the low-order divisor byte in AB (Figure 3–22), and the STC is set per L2. STAT E is set if the dividend sign is invalid, and D is decremented by L1 to the address of the high-order byte of the dividend.

Specification Test

This test verifies that the length codes for both operands in the instruction are correctly specified; i.e., L2 specifies eight bytes or less and is smaller than L1.

Incorrect Specification

Detection of an invalid specification forces a specification program interruption. The instruction address is restored to the LSWR, and the instruction is ended.

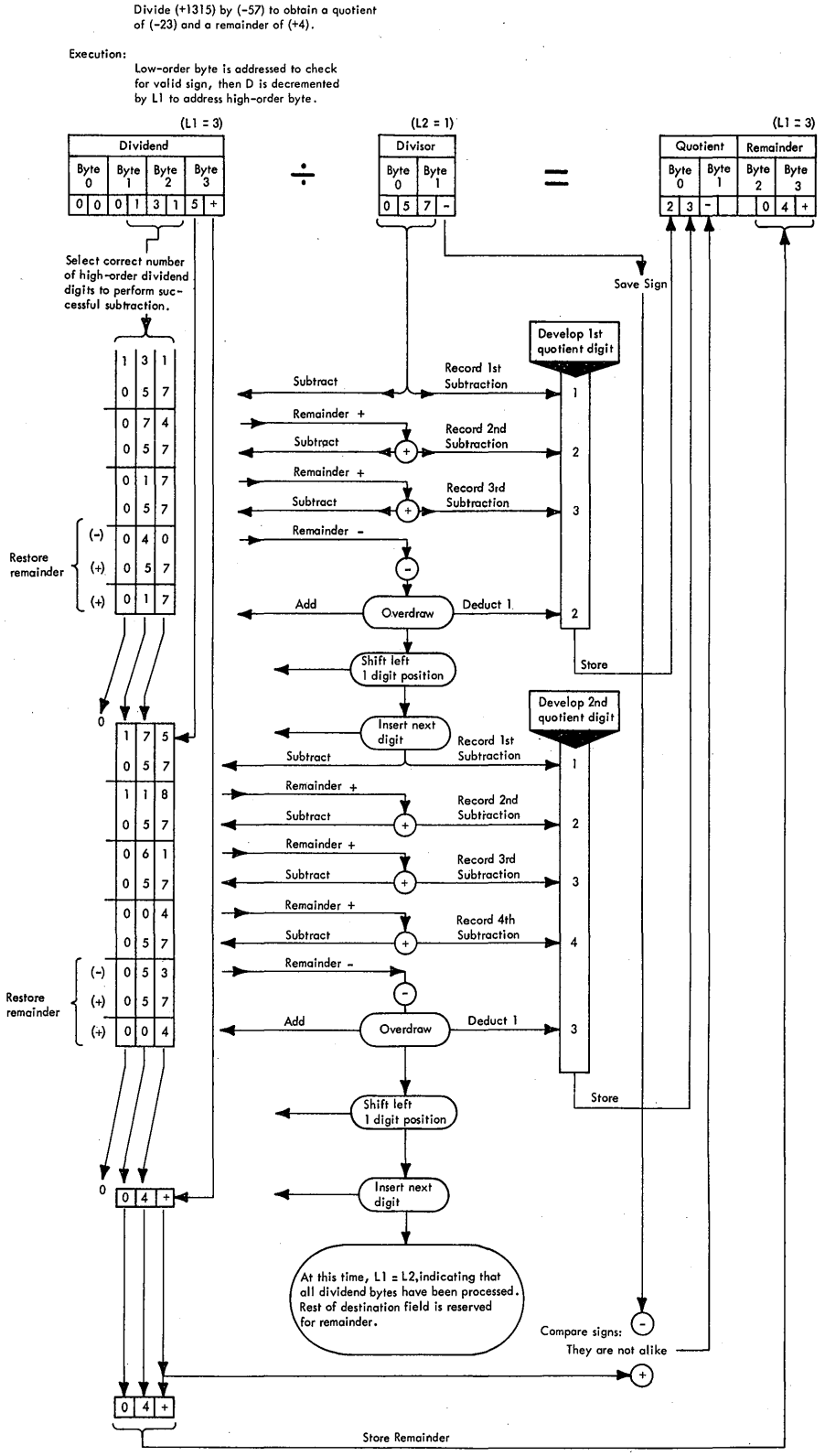


Figure 3-21. Example of a Typical Divide Sequence

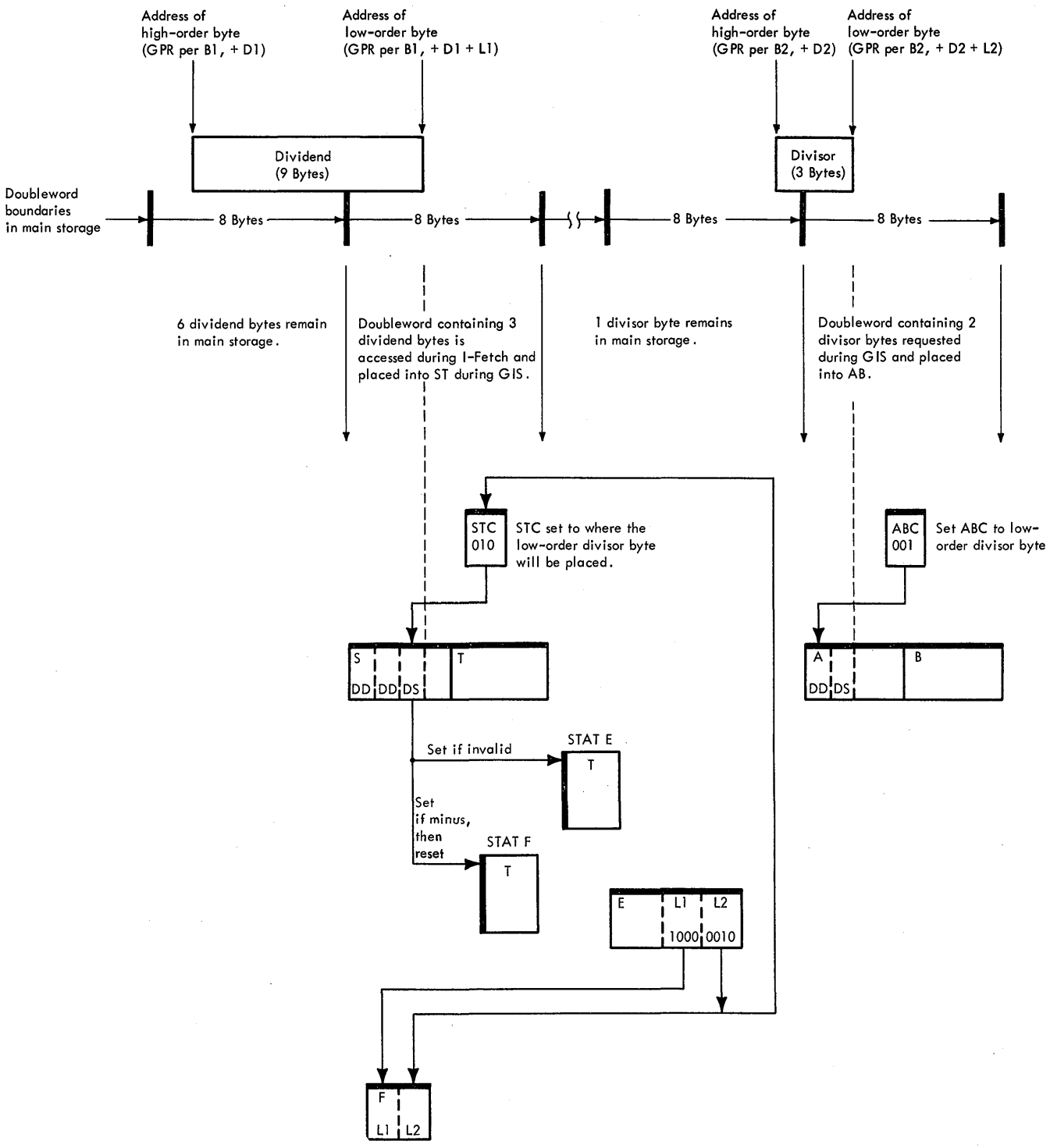


Figure 3-22. Data Handling During GIS of Decimal Divide

Divisor Left-Adjust Sequence

The divisor bytes are transferred from AB to ST in such a manner that the high-order divisor byte occupies the leftmost byte in ST. STAT C is set if the divisor sign is negative.

The left-adjust transfer is initiated by setting the STC per L2 (a function performed during the GIS). Because the maximum divisor length is limited to eight bytes, only three of the four bit positions in L2 are needed to effectively specify the length code; i.e., the count in L2 may range from a minimum of 0000 (for one byte) to a maximum of 0111 (for eight bytes). Setting the STC per L2 automatically selects, according to the divisor size, the correct ST position for the low-order byte of the divisor; the number of bytes to the left of the selected ST position corresponds to the length field of the full divisor (Figure 3–23). The actual transfer is performed one byte at a time, through the serial adder, starting with the low-order divisor byte. ABC, STC, and L2 are decremented by 1 for each byte transferred. The divisor is completely transferred when L2 is decremented to zero. Because the first IC request (during I-Fetch) does not necessarily access the full divisor to AB, it may be necessary to fetch the balance of the divisor from main storage. (This fetch occurs if the ABC steps to zero before L2 steps to zero.)

After exit from the Divisor Left-Adjust sequence, the full divisor has been fetched and left-adjusted to ST.

Dividend Fetch and Left-Adjust Sequence

This sequence fetches a sufficient number of high-order dividend bytes to perform a trial subtraction of the divisor from the dividend. The full divisor is subtracted once from the high-order dividend. Because the maximum divisor size is eight bytes, eight high-order dividend bytes are required for trial subtraction. If the dividend is eight bytes or less, it is completely fetched during this

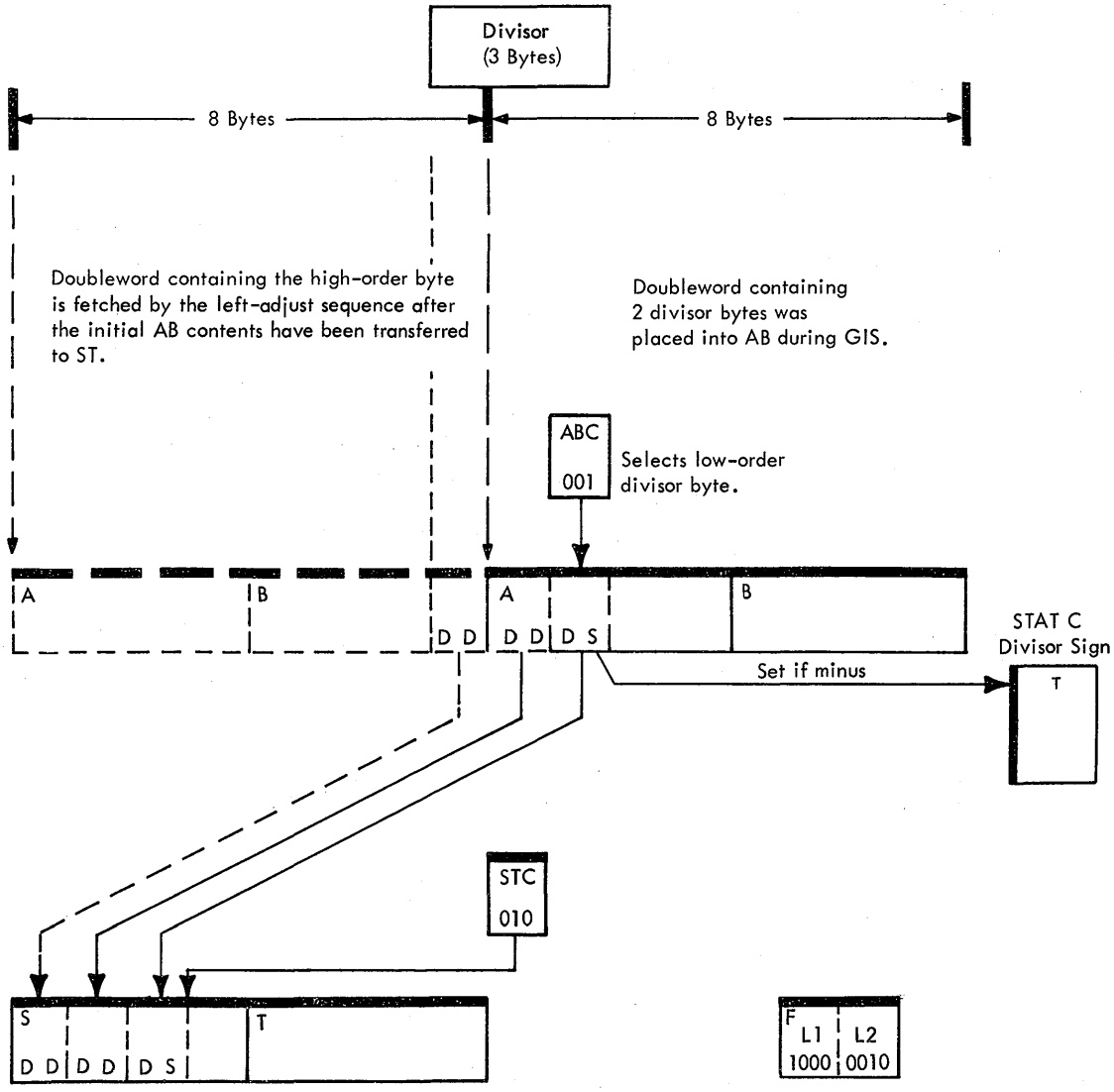


Figure 3-23. Data Handling During Divisor Left-Adjust Sequence

sequence; if greater than eight bytes, only the first eight high-order bytes are fetched. The dividend is fetched to AB and then transferred to ST in such a manner that the high-order byte occupies the leftmost byte in ST.

Upon entry into this sequence, ST is assumed to be completely occupied by the divisor. (If the divisor is four bytes or less, it is confined solely to S; if greater than four bytes, the divisor extends into T.) Because left-adjustment of the dividend requires the use of ST, the divisor must be transferred from ST; S is gated to the parallel adder and held in PAL, and T is stored into the LSWR (Figure 3–24).†

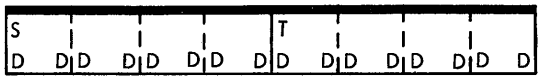
A request per the high-order dividend address is issued from D. Upon arrival of the dividend from main storage, the SDBO is gated to AB. The ABC is set per D(21–23) to select the high-order dividend byte. The left-adjust transfer is initiated by setting STC to 000, thus selecting the leftmost byte in ST. The dividend bytes are then transferred to ST, starting with the high-order byte. (The actual transfer is performed one byte at a time through the serial adder.) The ABC and STC are incremented, and L1 is decremented by 1 for each byte transferred. If L1 steps to zero before the ABC or STC steps to 7, the full dividend has been fetched and left-adjusted to ST. Because the first request does not necessarily access eight bytes of dividend to AB, it may be necessary to fetch additional dividend bytes from main storage. This fetch occurs if the ABC steps to 7 before the STC steps to 7 or L1 steps to zero.

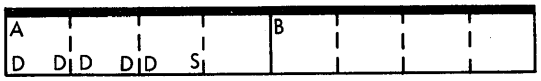
Restore L1 and L2 to E

Left-adjustment of the divisor and dividend has decremented L2 and L1 to zero. The initial L2 and L1 counts, saved in F during GIS, are now restored to E(8–15). These counts are required by the subsequent divide sequence.

Assemble Divisor in AB and Dividend in ST

The divisor in PAL and LSWR is restored to AB. Upon completion of this function, both operands are left-aligned: the dividend is in ST, and the divisor is in AB.





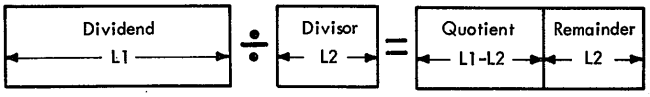
† The '→ HOLD' micro-order is issued on each cycle of the left-adjust sequence to hold the S contents in PAL.

Trial Subtraction

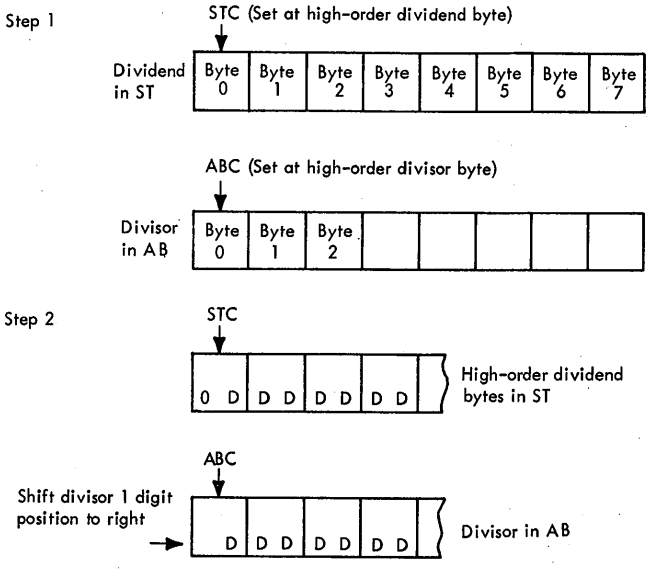
The divisor bytes in AB are subtracted from an equivalent number of high-order dividend bytes in ST. The remainder is then examined to establish whether the divide operation to follow will generate a result (quotient plus remainder) that will fit into the destination field. A negative remainder indicates that the destination field specified in the instruction is sufficiently large to accommodate the result. A positive remainder, however, indicates that the result cannot fit into the destination field, and a decimal divide program interruption occurs.

How prediction by trial subtraction is possible may be understood from the following considerations:

1. By definition, the dividend is at least one order higher than the divisor. The high-order digit position in the dividend is always zero.
2. The length code of the divisor (L2) is also the length code for the remainder. Consequently, the maximum number of quotient bytes that will fit into the destination field is equal to L1 minus L2. By operand definition, the difference of L1 minus L2 may range from a minimum of one byte to a maximum of eight bytes.



3. To perform the trial subtraction, the high-order divisor digit is aligned with the high-order digit of the dividend. This is performed in two steps: (1) the high-order divisor byte is aligned with the high-order byte of the dividend, and (2) because by definition the high-order digit position in the dividend is always zero, the divisor is shifted right one digit position to align the significant digits in both operands.



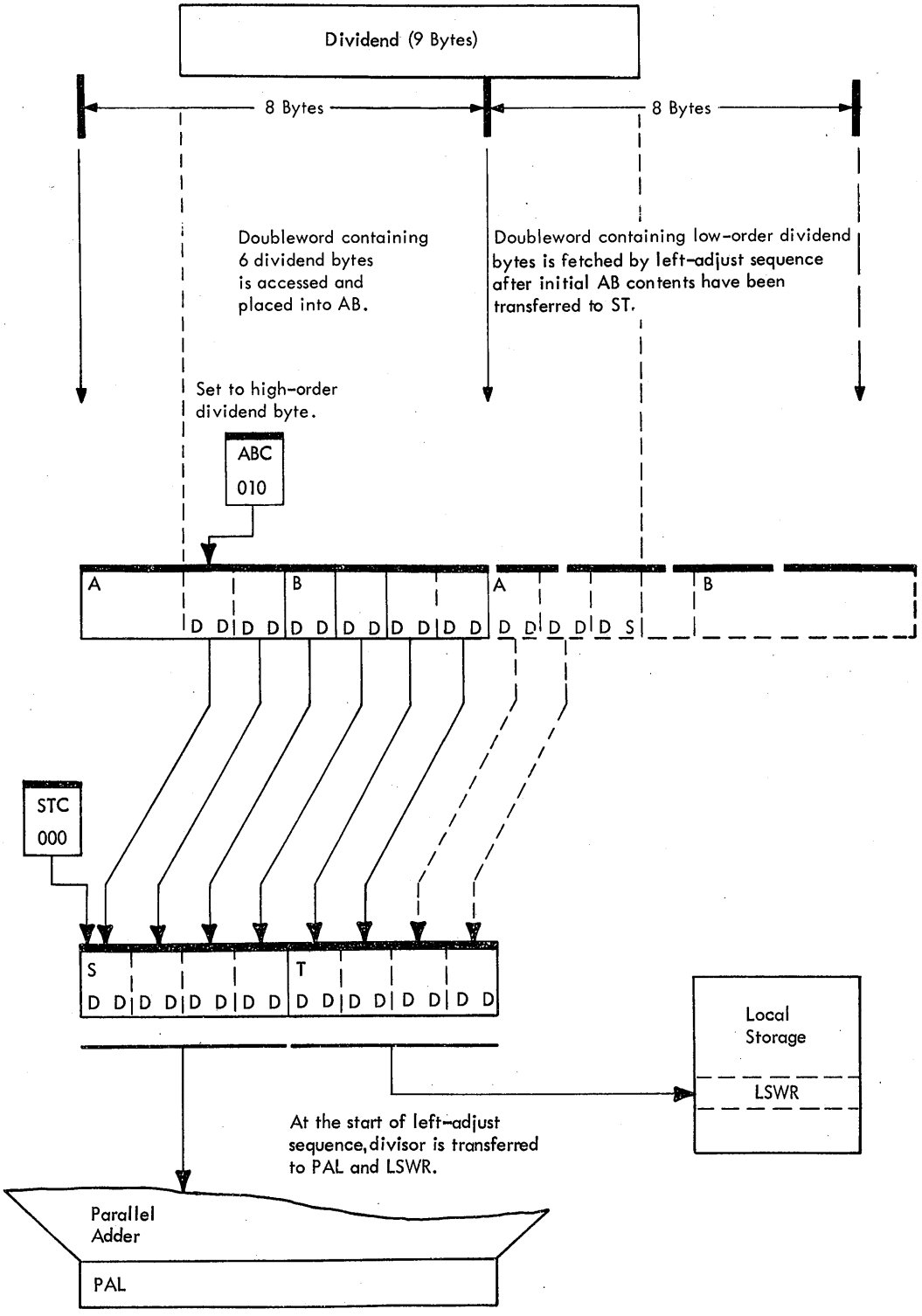
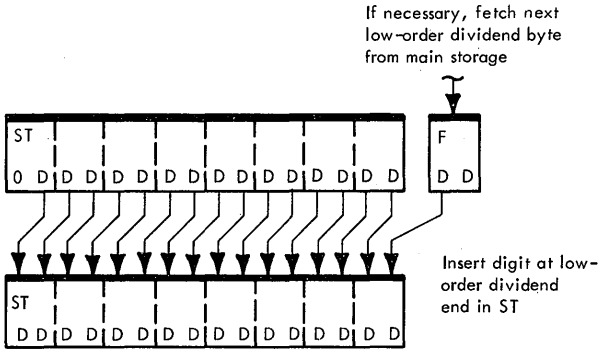


Figure 3-24. Data Handling During Dividend Fetch and Left-Adjust Sequence

4. Because the dividend is at least one order higher than the divisor, alignment of the high-order divisor digit with that of the dividend is equivalent to multiplying the divisor at least 10 times; if the dividend is one order higher, the divisor is multiplied 10 times; if two orders higher, 100 times; if three orders higher, 1000 times; and so on. Thus, during the trial subtraction, a quantity at least 10 times that of the divisor is subtracted from the dividend.
5. Because the maximum number of quotient digits allowed (L1 minus L2) corresponds to the difference between the orders of magnitude in the two operands, the result of the trial subtraction must always yield a negative remainder; otherwise, the number of quotient digits that would be generated would not fit into the destination field.

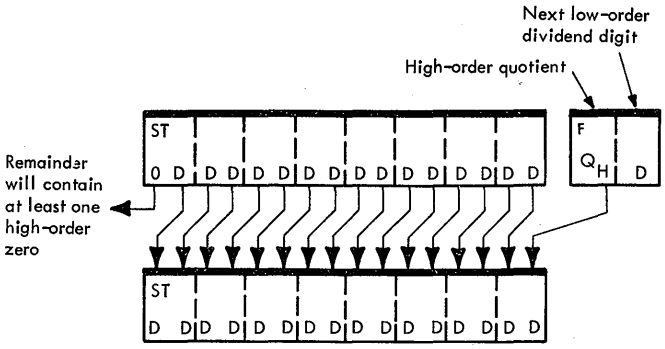
Shift Dividend One Digit to Left

The dividend is shifted one digit to the left to allow a successful subtraction of the divisor from the dividend. (To develop the quotient digit, the divisor must be repeatedly subtracted from the dividend until a negative remainder occurs.) Upon initiating the left-4 shift, a test is made to establish whether an additional low-order dividend digit is required for generation of the first quotient digit. If required, the next low-order dividend byte is fetched from main storage and placed into F. The digit is selected from F(0–3) and inserted at the low-order end of the dividend in ST.



Generate Quotient and Left-Digit Sequence

The divisor is repeatedly subtracted from the dividend until an overdraw occurs; i.e., a negative remainder is obtained. The number of successful subtractions is recorded and, after the last successful subtraction, becomes the high-order quotient digit. This digit is tested for validity and then inserted into F(0–3) by the left-digit sequence. This sequence also shifts the partial remainder (in ST) one digit to the left and inserts the next low-order dividend digit into the low-order end of ST.



Correct Low-Order Remainder Byte

In certain cases, the low-order remainder byte in ST must be corrected. The need for correction will become apparent when the Divide instruction is analyzed in detail. (See "Detailed Description" below.)

Generate Next Quotient Digit and Right Digit Sequence

After exit from the left-digit sequence, the operand length codes (L1 and L2) are compared to establish whether the last byte of the quotient is being processed. If L1 equals L2, the correct quotient sign is inserted into F(4–7). The last quotient byte (in F) is stored; then, the partial remainder (in ST) is stored into the low-order destination field as the final remainder.

If L1 does not equal L2, the next quotient digit is generated and placed into F(4–7) by the right-digit sequence. At the completion of this sequence, one complete byte of quotient is contained in F. This byte is stored into main storage, and the sequence for the left digit of the next quotient byte is started.

Detailed Description

- STAT A is set to indicate nonzero divisor.
- STAT C is set if divisor is negative.
- STAT D is set if dividend is less than eight bytes.
- STAT E is set if digit or sign is invalid.
- STAT F is set if dividend is negative.
- STAT G is first set if divisor is five bytes or greater. STAT G is then set again to enter left-digit sequence.
- STAT H is set to generate hot carry for subtract sequence.

Sheet 2 of Diagram 5-306 is a detailed flowchart of the divide microprogram. This figure is an expanded version of the overall flowchart, showing the data handling used in the various subroutines of the Divide instruction. The major subroutines and those areas in need of clarification are explained in the following subparagraphs.

General Initialization Sequence

This sequence shares a common microprogram with the Multiply instruction. An appropriate branch is taken to enter either the divide or the multiply sequence.

To test for an invalid dividend sign before the dividend is altered in main storage, the low-order dividend is accessed by a D request during SS I-Fetch. At the start of GIS, the doubleword containing the low-order byte of the dividend is gated to ST. The IC is incremented by L2 to address the low-order divisor byte, and the low-order divisor is accessed by an IC request. The contents of D are transferred to the STC, length codes L1 and L2 are transferred to F, the ABC is set to 7, and the IC is decremented by 8 to address the next doubleword of the divisor.

The dividend sign is tested, and STAT E is set if the sign is invalid. STAT F is set if the dividend sign is negative, but is immediately reset. D contains the address of the low-order byte of the dividend (contents of GPR addressed by B1, + D1 + L1). Because the divide operation begins at the high-order byte of the dividend, D is decremented by L1 to address the high-order byte of the dividend (contents of GPR addressed by B1, + D1). The STC is set per L2.

A test of L2 (contained in the STC) is performed to establish the byte size of the divisor. STAT G is set if the divisor is equal to or greater than five bytes. This function increases the execution speed when assembling the divisor in AB (see "Assemble Divisor in AB and Dividend in ST"); i.e., if the divisor is four bytes or smaller, the LSWR need not be restored to B.

Divisor Left-Adjust Sequence

1. The initial STC setting selects the rightmost ST byte that contains the low-order divisor byte.
2. STAT C is set if the divisor sign is negative.
3. If the ABC steps to zero before L2 steps to zero, the remaining low-order divisor bytes are fetched from main storage.
4. The divisor digits are checked for validity, and STAT E is set if an invalid digit is detected. STAT A is set to indicate a nonzero divisor. Division by zero results in a decimal divide program interruption during trial subtraction.
5. Upon fetching the full divisor, the divisor address is no longer needed, and the instruction address is restored to the IC.

Dividend Fetch and Left Adjust Sequence

1. The divisor is shifted one digit position to the right so that its high-order digit will be aligned with that of the dividend. (The dividend is not yet available at this time.)
2. The low-order divisor word is transferred from T to the LSWR. The high-order divisor word is gated to the parallel adder and held in PAL by the '- → HOLD' micro-order.
3. The STC is set to zero to select the high-order ST byte (where the high-order dividend byte will be placed).
4. A test of the high-order L1 bit is performed to establish the byte size of the dividend. STAT D is set if the dividend is less than eight bytes. This function increased the execution speed upon exit from the trial subtraction. A branch per STAT D is made to determine whether the complete dividend has been fetched. (If STAT D is set, the full dividend has been fetched, because at least eight dividend bytes are fetched to perform the trial subtraction.)
5. If the ABC steps to zero before L1 or STC steps to zero, the D-address is incremented by 8 and a fetch of the next doubleword of the dividend is made. The destination address for the subsequent quotient bytes is then restored by subtracting 8 from D.

Assemble Divisor in AB and Dividend in ST

This function is performed in parallel with the L1 and L2 restoration sequence. The high-order divisor word is transferred from PAL to A, after which restoration of the L1 and L2 counts is started. During the restoration sequence, STAT G is tested to establish whether the full divisor has been assembled in AB. If STAT G is not set, the divisor is four bytes or less. Therefore, the full divisor was contained in PAL and has been placed into AB. In this case, the restoration sequence is completed and an exit is made to the trial subtraction routine.

If STAT G is set, the low-order portion of the divisor is contained in the LSWR and must be transferred to B before entering the trial subtraction. The LSWR contents must be transferred to B via ST, which contains the left-aligned dividend. Several execution cycles are used to transfer the LSWR contents to B without destroying the ST contents.

Trial Subtraction

The divisor is subtracted from the dividend one byte at a time. After the last subtract cycle, a branch is made on a carry condition from SAL(0). Presence of a carry indicates that the remainder is positive, and the instruction is ended. Absence of a carry indicates a negative remainder, and that the result of the divide operation will fit into the destination field.

Dividend (or Partial Remainder) Left-4 Shift

The dividend is shifted left one digit position to perform the first successful subtraction of the divisor from the dividend. Upon initiating the left-4 shift, a test (per STAT D) is made to establish whether an additional dividend digit must be inserted into the low-order end of ST. If STAT D is set (see "Dividend Fetch and Left-Adjust Sequence," step 4), all dividend bytes have been fetched from main storage and the left-4 shift is completed. If STAT D is not set, the following actions take place:

1. The D-address is incremented by 8, and the next doubleword of the dividend is requested from main storage.
2. The contents of T are temporarily transferred to the LSWR. (Upon arrival of the dividend doubleword from main storage, T is loaded with the dividend word containing the next digit to be inserted.)
3. The STC is set per D(21–23) to select the correct dividend byte in the requested doubleword.
4. The left-4 shift of the dividend is completed. The high-order dividend word is in S, and the low-order word is in the LSWR.
5. A branch per D(21) is made to establish which word in the SDBO contains the next dividend byte. The correct word is then gated from SDBO to T. [Note that, if the left SDBO word is gated to T, STC(0) is forced to 1 to select the correct byte in T.]
6. The selected dividend byte is transferred from T to F. The shifted low-order dividend word is then restored from the LSWR to T.
7. The destination address is restored by subtracting 8 from D.
8. The high-order L1 bit is tested to establish the byte size of the dividend, and STAT D is set if the dividend is less than eight bytes. This function increases the execution speed upon exit from the right-digit sequence.

Generate Quotient Sequence

1. The ABC and STC are set per L2 to select the low-order operand bytes. STAT H is set to provide a hot carry to the serial adder.
2. The selected AB byte is subtracted from the selected ST byte via the serial adder. The result is gated back to the selected ST byte, with the carry being saved in STAT H. Any invalid digit detected in the serial adder sets STAT E.
3. The ABC and STC are decremented as each byte is processed. When the ABC is stepped to zero, a 1 is added to F(4–7) and the ABC and STC are again set per L2.
4. If a serial adder carry results upon processing the high-order byte, the partial remainder in ST is positive and the divisor is again subtracted from the dividend. F(4–7) is incremented by 1 each time a complete subtraction is made.
5. If there is no carry upon processing the high-order byte, an exit is made to the appropriate left- or right-digit sequence, as determined per STAT G.
6. Note that, before starting each subtract sequence, the partial remainder resulting from the previous subtraction is saved in the LSWR and PAL. The saving is done because, upon exit on a no-carry condition, an overdraw has occurred and the remainder in ST cannot be used for computation of the next quotient digit. Instead, the partial remainder resulting from the last successful subtraction is used for subsequent computation.

Left-Digit Sequence

1. The quotient digit in F(4–7) is the left digit of a quotient byte. This digit is reduced one count, to compensate for the overdraw, and then cross-gated via the serial adder to F(0–3).
2. The partial remainder resulting from the last successful subtraction and saved in the LSWR and PAL is shifted one digit to the left and restored to ST. The next low-order dividend digit, in B(64–67), is inserted into the low-order end of ST.
3. A test on STC equal to or greater than 4 is made to establish whether the low-order byte of the partial remainder in the LSWR has been overdrawn. In the generate quotient sequence, the contents of T are stored into the LSWR at the same time that the first subtract cycle is performed. Thus, if the partial remainder extends into T (which occurs if the STC is 4 or greater), the low-order divisor byte is subtracted from the low-order partial remainder byte once too often. In such cases, the low-order byte of the partial remainder is corrected by adding it to the low-order divisor byte. After performing the correction, the left-digit sequence is re-entered.
4. If L1 equals L2, the quotient sign byte is processed. Otherwise, the quotient digit generation routine is resumed to develop the next digit.

Correct Low-Order Remainder Byte

This routine is entered from the left- or right-digit sequence if the low-order divisor byte has been subtracted once too often from the low-order byte of the partial remainder. Correction is performed as follows:

1. STAT H is reset to initiate a true add cycle.
2. The low-order partial remainder word is placed into T. The STC is set per L2 to select the low-order byte in T.
3. The low-order divisor byte (per the ABC) is added once to the low-order partial remainder byte (per the STC), and the result is gated to T per the STC.
4. The left- or right-digit sequence is re-entered, as applicable.

Right-Digit Sequence

This sequence is entered when two quotient digits have been generated and placed into F. The following actions are performed:

1. The STC is set per D(21–23), and F is transferred to the selected ST byte. The corresponding mark trigger is set per the STC.
2. A storage request is issued to store the quotient byte per the D address.
3. A left-4 shift of the partial remainder (in PAL and LSWR) is initiated.
4. If STAT D is set, indicating that a dividend byte fetch is not required, the left-4 shift is completed and the partial remainder is restored to ST.
5. If STAT D is not set, the dividend byte fetch sequence is entered.
6. D is decremented by 1 to obtain the destination address for the next quotient byte.
7. F is cleared and STAT G is set to enter the left-digit sequence, after the first quotient digit is generated.
8. If STAT E is set, the 'invalid data interrupt' trigger is set and the instruction is ended.
9. If STAT E is not set, the generate-quotient sequence is entered.

Process Quotient Sign Byte

This routine is entered from the left-digit sequence when L1 equals L2. At this time, all dividend digits have been processed: the low-order quotient digit is in F(0–3), the byte selected by the STC is the dividend sign byte, and the remaining high-order contents of ST are the final remainder. The following actions take place:

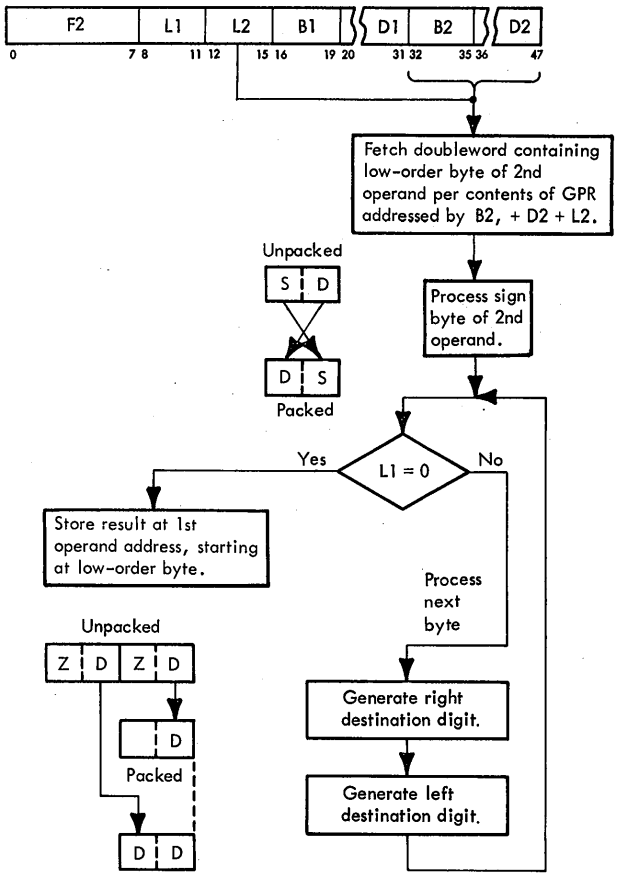
1. The STC and ABC are set per the L2 count. STAT F is set if bits 4–7 of the selected ST byte indicate a negative sign. STAT E is set if the sign is invalid; however, if an invalid sign existed, STAT E would have been set and a data program interruption would have occurred earlier.
2. The correct negative or positive sign is put into F(4–7) as determined by a comparison of STAT's C and F.
3. The ST contents are transferred to AB via the parallel adder.
4. The STC is set per D(21–23), and F is gated to the selected ST byte. The corresponding mark trigger is set, and the selected ST byte is stored into main storage.

Store Remainder Routine

1. The byte selected by the ABC, which is the low-order remainder byte, is saved in F. If necessary, the remainder sign is corrected in the serial adder before gating to F.
2. The remainder is transferred from AB to ST one byte at a time. As each byte is transferred, the corresponding mark trigger is set, the ABC and STC are incremented by 1, and L2 is decremented by 1.
3. When the STC steps to 7, ST contents are stored per the D-address. D is then incremented by 8, and the byte transfer is resumed.
4. When L2 steps to 0, the STC is decremented by 1, and the remainder sign byte is gated from F to ST. The contents of ST are then stored into the low-order destination field, and the instruction is ended.
5. If STAT E is set, an exit is made to the program interruption microprogram.

PACK, PACK (F2)

- Convert format of 2nd operand (in storage) from zoned to packed and place result into 1st operand location (in storage).
- SS format:



- Separate microprogram is used during word overlap.

The Pack instruction assumes source data in the unpacked format. The low-order source byte consists of a sign (bits 0–3) and a digit (bits 4–7). These two characters are swapped as they are gated to the low-order destination byte. All other source bytes consist of a zone (bits 0–3) and a digit (bits 4–7). Only the digits are gated to the destination field, with two bytes of source being processed for each byte of destination.

The sign and digits of the second operand are moved unchanged to the first operand field and are not checked for valid codes. A separate microprogram is provided for byte processing when a word-overlap condition exists. A test for word overlap is performed in the GIS of the instruction and also each time that a new doubleword of source is fetched from main storage.

The GIS microprogram for the Pack instruction is shown in Diagram 5-307, FEMDM. This microprogram gates the low-order first operand from the SDBO to ST, increments the IC by L2 to address the low-order byte of the second operand, gates the low-order second operand from the SDBO to AB, and performs the word-overlap test.

The word-overlap test is performed in two steps. First, the doubleword addresses for the destination and source are compared by subtracting D from the IC. The difference is then shifted four bit positions to the right and gated to PAL, and PAL(40–64) is sensed for an all-zero result to detect a possible word overlap. [The right-4 shift is made to avoid comparison of byte addresses within the doubleword; i.e., the difference for the byte addresses is shifted to PAL(65–67), which is not sensed by the branch.] If the addresses for the doublewords of source and destination are different, no word-overlap condition exists. Thus, if PAL(40–64) is not zero, a branch is made to the appropriate not-word-overlap execution sequence of the instruction.

If PAL(40–63) equals zero, indicating that the same doubleword address has been specified for the source and destination, a second test must be made to verify whether special data handling is required. The contents of D are again subtracted from the IC, but this time a right-4 shift on the difference is not performed and the byte addresses within the same doubleword are compared. If PAL(40–63) equals zero, an identical address has been specified for both source and destination. Because this case of word overlap does not require special data handling, a branch is made to the not-word-overlap microprogram. If, however, PAL(40–63) is not zero, the source and destination bytes are skewed; special data handling is required in the execution phase and, accordingly, a branch is made to the appropriate program.

Instruction Execution, Not Word Overlap

- Basic execution is as follows:
  1. Process sign byte and test for exit conditions.
  2. If no exit conditions, process right destination digit.
  3. Process left destination digit and test for exit conditions.

A flowchart of the execution of the Pack instruction without word overlap is shown in Diagram 5-308, FEMDM. The major functional steps in the microprogram are described in the following subparagraphs.

Process Sign Byte

1. The selected AB byte is gated via the serial adder cross-gates to the selected ST byte. The mark trigger selected by the STC is set.
2. ABC, STC, L1, and L2 are decremented by 1.
3. An exit is made to the appropriate routine if one or more of the counters (ABC, STC, L1, L2) was equal to zero before being stepped.
4. If no exit is made, the next source byte is processed to obtain the right destination digit.

Generate Right Destination Digit

1. Bits 4–7 of the selected AB byte are gated to SAA(4–7); no data is gated to SAA(0–3). The serial adder output is gated from SAL(0–7) to the selected ST byte.
2. The ABC and L2 are decremented by one count.
3. If L2 equals zero before stepping, the remaining source bytes are extended with high-order zeros. (See "Extension of Source Bytes with High-Order Zeros.")
4. If the ABC equals zero before stepping, an exit is made to the source fetch routine. STAT G is set to cause a return to the Generate Left Destination Digit routine after the source fetch.

Generate Left Destination Digit

1. Bits 4–7 of the selected AB byte are gated to SAA(0–3). Bits 4–7 of the selected ST byte are gated to SAB(4–7). The serial adder output is gated back to the selected ST byte, and the mark trigger selected by the STC is set.
2. ABC, STC, L1, and L2 are decremented by 1. If none of these counters equalled zero before stepping, the right digit for the next destination byte is generated. (Generate Right Destination Digit sequence is entered.)

Exit Conditions

An exit is made from the sign byte routine or from the left-digit routine when one or more of the following conditions are detected by the 'DECIMAL' (functional branch) micro-order:

1. L1 or STC = 0.
2. L2 = 0.
3. ABC = 0.

When the exit is on L1 or STC equals zero, a second test on L1-equal-all-1's is required to determine whether an end-op condition exists.

Extension of Source Bytes with High-Order Zeros

This routine is entered when L2 has stepped to zero before L1 has stepped to zero.

The serial adder output (zeros) is gated to the selected ST byte with the selected mark trigger being set. L1 and STC are decremented as each byte is processed. When L1 equals zero, the contents of ST are stored per the

D-address and the common end-op routine is started. When the STC equals zero, the contents of ST are stored, and D is decremented by 8. STAT H is set to cause a return to this routine after storing the contents of ST.

Source Fetch Routine

This routine is shared with the Move with Offset instruction. STAT D is set to cause a return to the pack microprogram.

The second operand is requested from main storage, and the IC is decremented by 8. A word-overlap test is performed. If no word-overlap condition exists, the next doubleword of the second operand is gated from the SDBO to AB. Processing of the left or right destination digit is resumed as determined by STAT G.

Instruction Execution, Word Overlap

- Basic execution is as follows:
  1. Process sign byte. Update AB and test for exit conditions.
  2. If no exit conditions, process right destination digit.
  3. Process left destination digit, update AB, and test for exit conditions.

A flowchart of the Pack instruction execution under word-overlap conditions is shown in Diagram 5-309, FEMDM. This microprogram is entered when a word-overlap condition is detected in the GIS or during a source fetch. The major functional steps in the microprogram are described in the following subparagraphs:

Process Sign Byte

The sign byte of the second operand in AB is processed in the same manner as in the not-word-overlap microprogram.

Update AB from ST

The data in AB is updated by transferring the contents of S to A or the contents of T to B, depending on the STC setting. ABC, STC, L1, and L2 are decremented by 1, and the mark trigger selected by the STC is set.

If any counter equalled zero before decrementing, an exit is made to the proper store, fetch, or extend-with-zeros routine as explained for the not-word-overlap sequence. If no exit conditions exist, processing of the right destination digit is started.

Generate Right Destination Digit

This routine is the same as in the not-word-overlap sequence.

Generate Left Destination Digit

This routine is the same as in the not-word-overlap sequence and is always followed by the update routine.

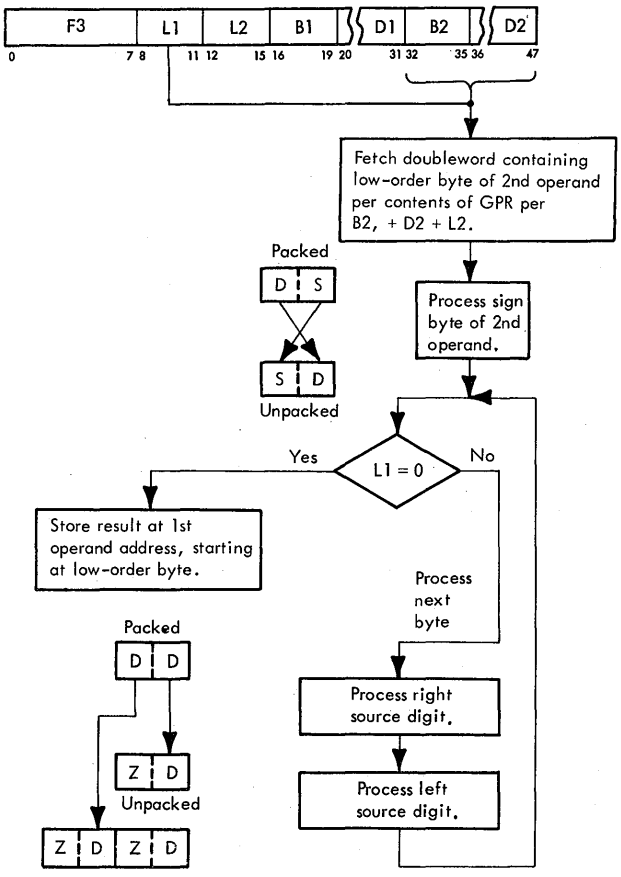
Source Fetch Routine

The next doubleword of source is requested from main storage, after which the IC is decremented by 8. Upon detection of a word-overlap condition, however, this doubleword is not used, because AB must be updated from ST. If, upon entering the source fetch routine, only the right destination digit has been placed into the selected ST byte, this byte is not transferred to AB. Instead, the following action takes place:

1. The portion of ST that has been processed (as determined by the mark triggers) is stored into the destination field, refetched from storage, and gated to both AB and ST.
2. If STAT G is set, indicating that only the right digit of the selected ST byte has been processed, the selected ST byte is transferred to F before the SDBO is gated to ST. After the SDBO is gated to ST, F is reinserted into the selected ST byte, and processing of the left digit is started.
3. If STAT G is not set, indicating that a complete ST byte has been processed, it is not necessary to save the selected ST byte. Processing of the right digit is started immediately.

UNPACK, UNPK (F3)

- Convert format of 2nd operand (in storage) from packed to zoned and place result into 1st operand location (in storage).
- SS format:



- Separate microprogram is used during word overlap.
- Word-overlap test is performed during GIS and in destination store and source fetch routines.

The Unpack instruction assumes data in the packed format. The low-order source byte consists of a sign (bits 4–7) and a digit (bits 0–3). These two characters are swapped as they are gated to the low-order destination byte. All other source bytes contain a pair of binary-coded-decimal digits. Each digit is transferred to the low-order portion (bits 4–7) of the corresponding destination byte, and a zone character is inserted into the high-order portion byte (bits 0–3). During this transfer, the digits are not tested for validity.

A separate microprogram is provided for byte processing when a word-overlap condition exists. A test for a word-overlap condition is performed in the GIS of the instruction and also each time that a doubleword of data is fetched from or stored into main storage.

The Unpack instruction generates two bytes of destination for each byte of source. Therefore, the condition when the destination bytes are processed "ahead" of the source always exists if the operand fields overlap. When the same doubleword address is specified, special data handling is required regardless of how the operand bytes are arranged in this doubleword. Special handling is necessary each time that source data is fetched from main storage; also, upon storing unpacked data into the destination field, a word-overlap test must be made to determine whether the source data in the CPU must be updated from storage.

The GIS microprogram for the Unpack instruction is shown in Diagram 5-307. When the first overlap indication occurs, the byte addresses are not checked. Instead, a branch is forced into the word-overlap sequence by supplying a hot carry to PAA(60), so that a test of PAL(40–63) always yields a nonzero result.

Instruction Execution, Not Word Overlap

- Basic execution is as follows:
  1. Process sign byte and test for exit conditions.
  2. If no exit conditions, process right source digit.
  3. Process left source digit, and test exit conditions.

A flowchart of Unpack instruction execution without word overlap is shown in Diagram 5-310, FEMDM. The major functional steps in the microprogram are described in the following paragraphs.

Process Sign Byte

The sign byte, selected by the ABC, is gated via the serial adder cross-gates to the selected ST byte, and the corresponding mark trigger is set. ABC, STC, L1, and L2 are decremented by 1, and an exit is made if any counter equalled zero before stepping.

Process Right Source Digit

1. Bits 4–7 of the selected AB byte are gated to SAA(4–7). The approved zone character is inserted into SAA(0–3). The serial adder output is gated to the selected ST byte, and the selected mark trigger is set.
2. L1 and STC are decremented by 1.
3. If L1 equalled zero before stepping, the contents of ST are stored and the common end-op sequence is started.
4. If STC equalled zero before stepping (and L1 was not zero), the destination store routine is started. STAT G is set to record an exit from the right digit routine.

Process Left Source Digit

1. Bits 0–3 of the selected AB byte are gated to SAA(4–7), and the zone character is inserted into SAA(0–3).
2. The adder output is gated to the selected ST byte, and the selected mark trigger is set.
3. ABC, STC, L1, and L2 are decremented by 1. An exit is made to the appropriate routine if any of the above counters equalled zero before stepping. If no exit condition exists, the right source digit in the next source byte is processed.

Exit Conditions

An exit is made from the byte processing routine whenever it is detected that L1, L2, ABC, or STC is equal to zero. Although a separate exit is provided for each possible combination of these conditions, they may be considered to be examined in the following order of priority:

1. L1 = 0
   The contents of ST are stored per the D-address, and the common end-op routine is started.
2. L2 = 0
   AB is cleared, the ABC is set per L2 (which is 7), and STAT H is set to record the end of the source field. If the STC was also zero, the destination store routine is started. If the STC was not zero, the high-order zeros routine is entered per STAT H.
3. STC = 0
   The destination store routine is started. If the ABC was also zero, STAT D is set to cause a source fetch after the destination store.
4. ABC = 0.
   The source fetch routine is started.

Extension of Source Bytes with High-Order Zeros

AB is cleared, and bits 4–7 of the selected AB byte (zeros) are gated to SAA(4–7); the approved zone character is inserted into SAA(0–3). The adder output is gated to the selected ST byte, and the corresponding mark trigger is set. L1 and STC are decremented by 1 for each

byte that is processed. An exit is made to the destination store routine when the STC steps to zero, and to end-op when L1 steps to zero.

Source Fetch Routine

A request is made per the IC address, after which the IC is decremented by 8. A word-overlap test is made. If there is no word-overlap condition, the next source word is gated to AB, and the right digit of the next source byte is processed.

Destination Store Routine

1. The contents of ST are stored into the destination field per the D-address, and D is decremented by 8.
2. An exit is made to the source fetch routine if STAT D is set.
3. An exit is made to the high-order zeros routine if STAT H is set.
4. If neither STAT D nor STAT H is set, a word-overlap test is made by comparing the IC and D addresses. If no word overlap exists, the left or right digit is processed as determined by STAT G.

Instruction Execution, Word Overlap

- Basic execution is as follows:
  1. Process sign byte. Update AB and test for exit conditions.
  2. If no exit conditions, process right source digit.
  3. Process left source digit, and test for exit conditions.
- Word-overlap test is performed during source fetch and destination store routines.

A flowchart of Unpack instruction execution under word-overlap conditions is shown in Diagram 5-311, FEMDM. The steps in which this microprogram differs from that for not-word-overlap are explained in the following paragraphs.

Process Sign Byte

This step is the same as in the not-word-overlap sequence except that it is always followed by the update routine.

Update AB from ST

If the STC is less than 4, the contents of S are transferred to A; the contents of T are always transferred to B. The mark trigger selected by the STC is set. ABC, STC, L1, and L2 are decremented by 1. An exit is made to the appropriate routine if any of the above counters equalled zero before their being stepped. If no exit conditions exist, the right digit in the next source byte is processed.

Process Right Source Digit

The right source digit is processed in the same manner as for not-word-overlap. If upon processing the right digit an exit is made on STC equal zero, and ABC is not zero, the contents of S are transferred to A. In this manner, the source is correctly updated before storing the contents of ST.

Process Left Source Digit

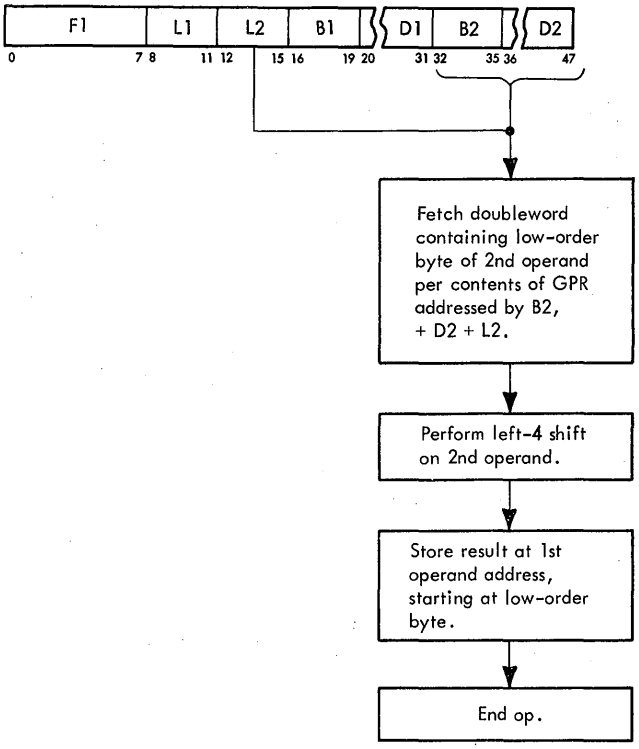
This step is the same as in the not-word-overlap sequence except that it is always followed by the update routine.

Source Fetch Routine

1. The source is requested per the IC address, after which the IC is decremented by 8.
2. The contents of D are subtracted from the IC to prepare for the word-overlap test; also, a test on STC equals 7 is made to establish how the source is to be updated in case of an overlap condition.
3. The condition when STC equals 7 indicates that the STC was zero before entering the source fetch routine. In this case, the destination has been stored into main storage. Thus, to update the source, the doubleword at the SDBO is gated to AB and ST, and processing of the left source digit is started.
4. If the STC is not 7, AB must be updated from ST. After transfer of the contents of ST to AB, processing of the right source digit is started.

MOVE WITH OFFSET, MVO (F1)

- Store 2nd operand (in storage) to left of and adjacent to low-order 4 bits of 1st operand (in storage).
- SS format:



- Separate microprogram is used during word overlap.

The MVO instruction performs a left-4 shift on the second operand and transfers the result to the first operand location. Thus, the four low-order bits of the first operand are preserved as the lowest-order character of the second operand. During execution of the instruction, the operand signs and digits are not tested for valid codes.

No decimal shift instruction is provided, because the equivalent of a shift can be obtained by programming. Programs for right or left shift, and for an even or an odd shift amount, are written with Move with Offset instruction and the logical move instructions described in Section 5 of this Chapter.

A separate microprogram is provided for byte processing when a word-overlap condition exists. A test for word overlap is performed in the GIS of the instruction, and also each time that a new doubleword of source is fetched from main storage.

The GIS for the Move with Offset instruction is shown in Diagram 5-307. This microprogram is identical with the GIS microprogram of the Pack instruction.

Instruction Execution, Not Word Overlap

- Basic execution is as follows:
  1. Transfer bits 4–7 of selected AB byte to bits 0–3 of selected ST byte. Decrement counters.
  2. Transfer bits 0–3 of selected AB byte to bits 4–7 of selected ST byte. Repeat first step.
  3. Exit on L1 or STC = 0, L2 = 0, or ABC = 0.

A flowchart of the execution of the Move with Offset instruction when no word-overlap condition exists is shown in Diagram 5-312, FEMDM. Basically, this microprogram specifies a 2-cycle loop with appropriate exits to source fetch, destination store, high-order-zero extend, and end-op routines.

Cycle 1

1. Bits 4–7 of the selected AB byte are gated to SAA(0–3).
2. Bits 4–7 of the selected ST byte are gated to SAB(4–7).
3. The serial adder output is gated back to the selected ST byte, and the corresponding mark trigger is set.
4. L1 and STC are decremented by 1. An exit is made to the destination store routine if L1 or STC equalled zero before stepping.

Cycle 2

1. Bits 0–3 of the selected AB byte are gated to SAA(4–7). No data is gated to serial adder bits 0–3.
2. The serial adder output is gated to the selected ST byte.
3. L2 and ABC are decremented by 1. If L2 was zero before stepping, an exit is made to the high-order zero extend routine. If L2 was not zero but ABC equalled zero, an exit is made to the source fetch routine.
4. If L2 or ABC is not equal to zero, cycle 1 is repeated.

High-Order Zero Extend Routine

An entry is made into this routine when the last source byte has been processed. The selected ST byte contains the high-order source digit in bits 4–7; bits 0–3 are zeros.

The following actions are performed upon entry into the routine:

1. STAT H is set.
2. The selected mark trigger is set.
3. L1 and STC are decremented by 1.
4. If L1 or STC equals zero before stepping, an exit is made to the destination store routine.

If L1 or STC is not zero, a 1-cycle loop is started, which:

1. Gates the serial adder output (zeros) to the selected ST byte.
2. Sets the mark trigger selected by the STC.
3. Decrements L1 and STC by 1.
4. Exits to the destination store routine when L1 or STC equals zero. (STAT H is set to cause re-entry into the high-order zeros routine after the destination is stored.)

Destination Store Routine

1. The contents of ST are stored into the destination field per the D-address.
2. A test is made for the end of the destination field. If the L1 count now equals all 1's, an exit is made to the common end-op sequence.
3. If L1 is not all 1's, D is decremented by 8.
4. If STAT H is set, the high-order zeros routine is resumed. If STAT H is not set, the byte processing loop is started at cycle 2.

Source Fetch Routine†

1. The source is requested from storage, and the IC is decremented by 8.
2. A word-overlap test is made by comparing the IC and D addresses.
3. If no word-overlap condition exists, the doubleword arriving from storage is gated to AB, and byte processing is resumed.

Instruction Execution, Word Overlap

- Basic execution is as follows:
  1. Transfer bits 4–7 of selected AB byte to bits 0–3 of selected ST byte.

† This routine is shared with the Pack instruction. Return to the appropriate microprogram is effected per STAT D.

2. Transfer bits 0–3 of selected AB byte to bits 4–7 of selected ST byte.
3. Update AB from ST, and repeat first step.
4. Exit on L1 or STC = 0, L2 = 0, or ABC = 0.

A flowchart of the execution of the Move with Offset instruction when a word-overlap condition exists is shown in Diagram 5-313, FEMDM. Basically, this microprogram specifies a 3-cycle loop with appropriate exits to source fetch, destination store, high-order-zero extend, and end-op routines.

Cycle 1

This cycle is identical with cycle 1 in the not-word-overlap microprogram.

Cycle 2

1. Bits 0–3 of the selected AB byte are gated to SAA(4–7).
2. The serial adder output is gated to the selected ST byte.

Cycle 3

1. If the STC is less than 4, the contents of S are transferred to A.
2. The contents of T are transferred to B via the parallel adder.
3. L2 and ABC are decremented by 1.
4. An exit is made to the high-order zeros routine if L2 was equal to zero before stepping. An exit is made to the source fetch routine if the ABC was equal to zero and L2 was not zero.
5. If no exit conditions exist, cycle 1 is repeated for the next byte.

High-Order Zero, Destination Store, and Source Fetch Routines

The high-order zero and destination store routines are the same as in the not-word-overlap sequence. The source fetch routine, however, is different.

Upon detecting a word-overlap condition, the source from main storage is not used. Instead, AB is updated from ST: if the STC is equal to 7, the contents of T are transferred to B; if the STC is not 7, the contents of S are transferred to A and the contents of T to B.

www.IBM360.info

This section discusses the 32 logical instructions. The instructions use all five formats and operate on fixed- and variable-field length data. For a discussion of data formats, operand addressing, instruction formats, data flow, program interruptions, and condition codes, see Chapter 1.

GENERAL INITIALIZATION SEQUENCE

Before execution of SS logical instructions, a General Initialization Sequence (GIS) is performed (Diagram 5-401, FEMDM). The general function of the GIS is to set up initial conditions for the execution phase. These include:

1. Setting of STC and ABC. The STC is set to the rightmost first operand byte in ST, the byte to be processed first. Because the address of the rightmost byte is specified by D(21–23), the STC is set per these bits. Similarly, the rightmost second operand byte is selected in AB by transferring IC(21–23) to the ABC.
2. Transferring the first operand to ST during the first cycle of GIS.
3. Transferring the second operand to AB. An IC request for the second operand is issued on the first cycle of GIS; subsequently, GIS transfers the operand from the SDBO to AB.
4. Performing a word overlap test. For the purpose of this test, refer to "Word Overlap Condition", Section 4 of this Chapter.

At the completion of SS I-Fetch, a branch is made per the instruction op-code to the appropriate GIS microprogram. Note that, because of similarities in the GIS microprograms, the SS logical instructions are divided into two groups. One group consists of the Translate and Translate and Test instructions; the remaining SS instructions form the second group.

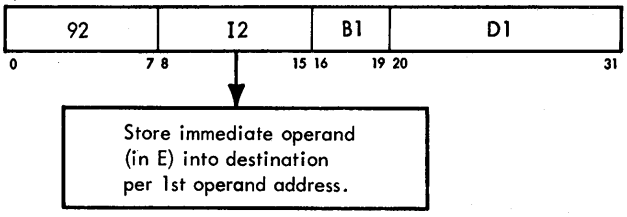
MOVE

Four logical move instructions are available:

1. Move, MVI, SI format. Places an immediate operand into the first operand location.
2. Move, MVC, SS format. Places the second operand into the first operand location.
3. Move Numerics, MVN, SS format. Places the numerics of the second operand bytes into the corresponding positions of the first operand bytes.
4. Move Zones, MVZ, SS format. Places the zones of the second operand bytes into the corresponding positions of the first operand bytes.

Move, MVI (92)

- Place immediate operand (I2 of instruction) into 1st operand location (in storage).
- SI format:

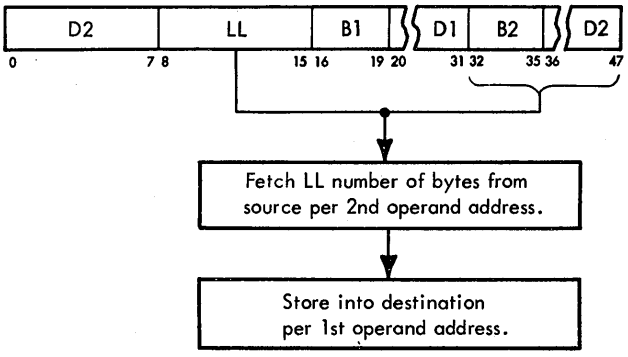


- Conditions at start of execution:
  First 16 bits of instruction, containing immediate operand, are in E.
  Main storage request for 1st operand has been issued per D.

The Move, MVI, instruction places the immediate operand into the first operand location. The immediate operand (I2 of instruction) is in E.

Move, MVC (D2)

- Place 2nd operand (in storage) into 1st operand location (in storage).
- SS format:



- Conditions at end of GIS:
  1st operand is in ST.
  2nd operand is in AB.
  ABC is set to select byte to be routed through serial adder.
  First 16 bits of instruction are in E.

- Move operation can be high or low speed.
- Three separate microprograms are provided:
  High-speed move.
  Word overlap.
  Low-speed move.

Three separate sequences are provided for the MVC instruction. The high-speed move sequence is used when it is possible to transfer a doubleword of data at a time. This condition exists when the high-order bytes of the source and destination are specified on doubleword boundaries and a full doubleword of data remains to be processed; i.e., both the ABC and STC are equal to zero, and the LL count is greater than 6. The word-overlap sequence is used when a word-overlap condition exists. The second operand in AB is updated after each AB byte is processed. The low-speed move sequence is used when the high-speed or word-overlap condition does not exist. (The high-speed and word-overlap conditions are detected in the GIS of the instruction.)

1. Low-Speed Move Sequence
   This sequence is basically a 1-cycle operation in which the AB byte selected by the ABC is transferred through the serial adder to the ST byte selected by the STC, and the mark trigger selected by the STC is set.
   The STC and ABC are incremented, the LL count in E(8–15) is decremented, and the cycle is repeated for the next byte, unless an exit condition exists.
2. Word-Overlap Move Sequence
   This sequence is a 2-cycle sequence in which the first cycle transfers the AB byte, selected by the ABC, to the ST byte selected by the STC. The second cycle updates the source operand in AB by transferring S to A, or T to B, as determined by the value of the STC. The mark trigger selected by the STC is set. The STC and ABC are incremented, the LL count is decremented, and the sequence is repeated for the next byte, unless an exit condition exists.
3. High-Speed Move Sequence
   This routine is entered from the GIS or from the low-speed move routine.
   a. When the entrance is made from the GIS, the source operand has been transferred to ST. The contents of ST are stored by setting mark triggers 0–7 and issuing a storage request per D.
   b. The LL count in E(8–15) is decremented by 8 via the parallel adder and is then tested for all 1's. If this condition exists, an end-op sequence is started. If no end-op condition exists, the IC is incremented by 8 via the parallel adder and a source fetch request is given.
   c. When the entrance is made from the low-speed routine, D is incremented by 8 and the source doubleword from main storage is gated to both AB and ST. If at least 8 bytes remain to be processed, as determined by a ROS branch on LL count being greater than 6, the high-speed move sequence is repeated (starting at step a). If fewer than 8 bytes remain to be processed, the low-speed move sequence is started to process the remaining data.

Exit is made from the low-speed or word-overlap move routines is one of the following conditions exists: (1) LL = 0, or STC = 7 and ABC ≠ 7; (2) LL = 0, or STC = 7 and ABC = 7; (3) only ABC = 7. A separate sequence is entered for each of these conditions, as explained below:

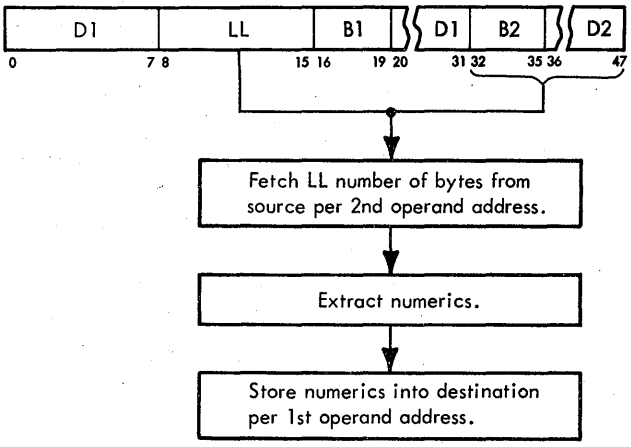
1. LL = 0, or STC = 7 and ABC ≠ 7
   A destination store is initiated, and a test for an end-op condition is made. If the LL count now equals all 1's, an entry is made into a common end-op sequence. If an end-op condition does not exist, D is incremented by 8 via the parallel adder and the low-speed move sequence is continued.
2. LL = 0, or STC = 7 and ABC = 7
   A destination store is initiated, and a test for end-op is made (LL = all 1's). A further test for a high-speed move condition is made. If at this time the LL count is 7 or greater, the IC and D are incremented by 8, a source fetch is initiated, and an entry is made into the high-speed move sequence. If neither an end-op nor a high-speed move condition exists, D is incremented by 8 and a common source fetch routine is entered which increments the IC by 8, fetches the next doubleword of source to AB, and tests for a word-overlap condition. Because there is no word-overlap at this time (ABC = STC), the low-speed move sequence is continued.
3. ABC = 7
   The IC is incremented by 8 through the parallel adder, and a fetch request is given to fetch the next doubleword of source operand. The common source fetch sequence is entered, which tests for word overlap. In this case, word overlap may exist: if it is detected, the source operand from main storage is not gated to AB, but instead ST is gated to AB and a branch is made to the move-word-overlap sequence. If no word overlap exists, the low-speed move sequence is continued after the source operand from main storage is gated to AB.

The common end-op routine is entered when the LL field has been decremented to zero. This routine restores the instruction address from the LSWR to the IC and resets STAT G (because it may have been used during the GIS).

Move Numerics, MVN (D1)

- Place numeric portion (low-order 4 bits) of each byte of 2nd operand (in storage) into low-order 4 bits of corresponding byte of 1st operand (in storage).

- SS format:



- Conditions at end of GIS:
  1st operand is in ST.
  2nd operand is in AB.
  ABC and STC are set to select byte(s) to be routed through serial adder.
  First 16 bits of instruction are in E.
- Separate microprogram is used for word overlap.

The MVN instruction is executed as follows:

1. Bits 4–7 of the selected AB byte are gated to SAA(4–7).
2. Bits 0–3 of the selected ST byte are gated to SAB(0–3).
3. Adder output is gated back to the selected ST byte.

Data is processed one byte at a time, and the fields may overlap in any way. Separate sequences are used for the not-word-overlap and the word-overlap conditions:

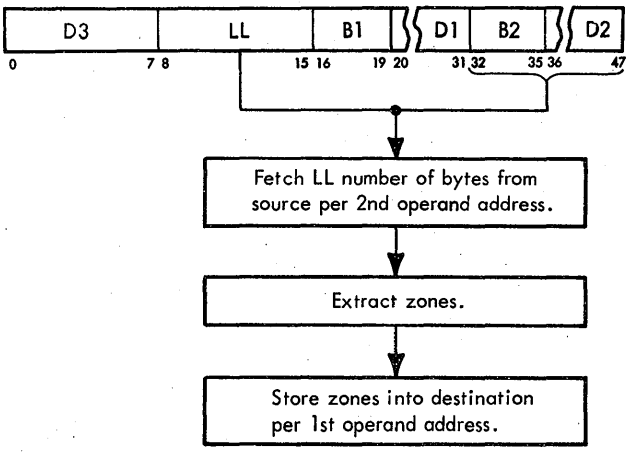
1. Not-Word-Overlap Sequence
   This sequence consists of a 1-cycle loop with an exit when LL = 0, STC = 7, or ABC = 7. As each byte is processed, the corresponding mark trigger is set per the STC; ABC and STC are incremented by 1 and LL is decremented by 1.
2. Word-Overlap Sequence
   This sequence consists of a 2-cycle loop with an exit when ABC or STC = 7, or when LL = 0.
   a. Cycle 1
      Numeric (bits 4–7) is moved from AB to ST.
   b. Cycle 2
      The contents of S are transferred to A, or the contents of T are transferred to B as determined by the STC value. The mark trigger selected by the STC is set; STC and ABC are incremented by 1, and LL is decremented by 1.

An exit from the byte processing sequence is made when LL = 0, STC = 7, or ABC = 7. A separate sequence is entered for each of these conditions, as explained below:

1. LL = 0
   The contents of ST are stored per D into the destination field. The common end-op sequence is started.
2. STC = 7
   The common destination store-fetch routine is started. If the ABC also equals 7, STAT D is set to cause a source fetch before resuming the byte processing loop.
3. ABC = 7
   The common source fetch routine is started, which includes a word-overlap test, which causes the appropriate instruction word-overlap or not-word-overlap loop to be continued.

Move Zones, MVZ (D3)

- Place zone portion (high-order 4 bits) of each byte of 2nd operand (in storage) into high-order 4 bits of corresponding byte of 1st operand (in storage).
- SS format:



- Conditions at end of GIS:
  1st operand is in ST.
  2nd operand is in AB.
  STC and ABC are set to select byte(s) to be routed through serial adder.
  First 16 bits of instruction are in E.
- Separate microprogram is used for word overlap.

The MVZ instruction specifies the following actions:

1. Bits 0–3 of the selected AB byte are gated to SAA(0–3).
2. Bits 4–7 of the selected ST byte are gated to SAB(4–7).
3. The adder output is gated back to the selected ST byte.

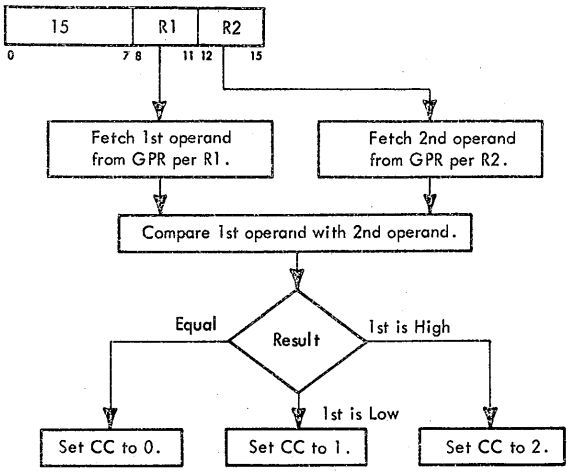
Except for the above actions, the byte processing sequence is the same as that for the MVN instruction.

COMPARE

Four Compare Logical instructions are provided, in the RR, RX, SI, and SS formats. Comparison is binary, and all codes are valid. Operation is terminated when an inequality is found.

Compare Logical, CLR (15)

- Binarily compare 1st operand (in GPR, per R1) with 2nd operand (in GPR, per R2) and set CC according to result.
- RR format:

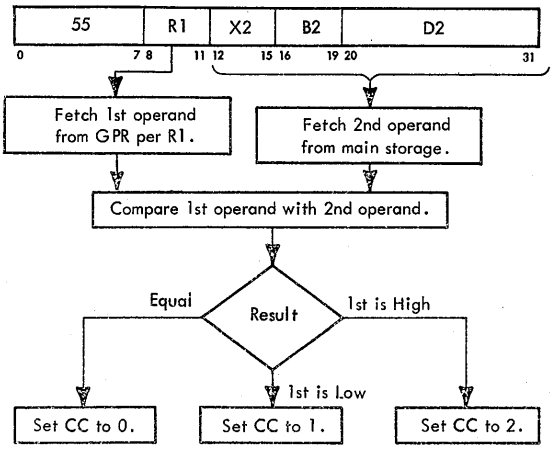


- Conditions at start of execution:
  1st operand is in S and T.
  2nd operand is in A and B.
  Instruction is in E.
- CC setting:
  Operands are equal: CC = 0.
  1st operand is less than 2nd operand: CC = 1.
  1st operand is greater than 2nd operand: CC = 2.

The Compare Logical, CLR, instruction, which is in the RR format, compares the first operand with the second operand. Comparison in binary, and is performed left to right, byte by byte. The CC is set according to the result.

Compare Logical, CL (55)

- Binarily compare 1st operand (in GPR, per R1) with 2nd operand (in storage) and set CC according to result.
- RX format:

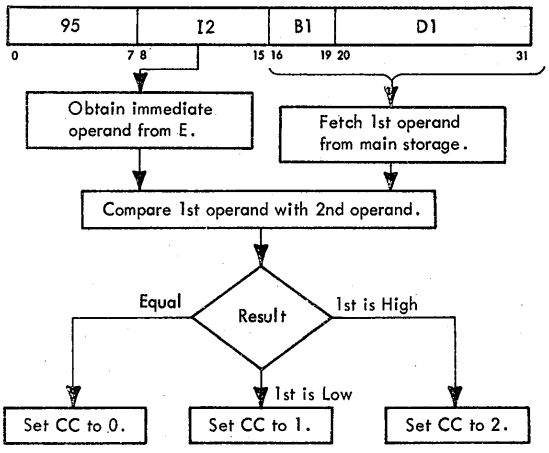


- Conditions at start of execution:
  1st operand is in S and T.
  Main storage request for 2nd operand has been issued per D.
  First 16 bits of instruction are in E.
- CC setting:
  Operands are equal: CC = 0.
  1st operand is less than 2nd operand: CC = 1.
  1st operand is greater than 2nd operand: CC = 2.

The Compare Logical, CL, instruction, which is in the RX format, compares the first operand with the second operand. Comparison is binary, and is performed left to right, byte by byte. The CC is set according to the result.

Compare Logical, CLI (95)

- Binarily compare 1st operand (in storage) with immediate operand (I2 of instruction) and set CC according to result.
- SI format:

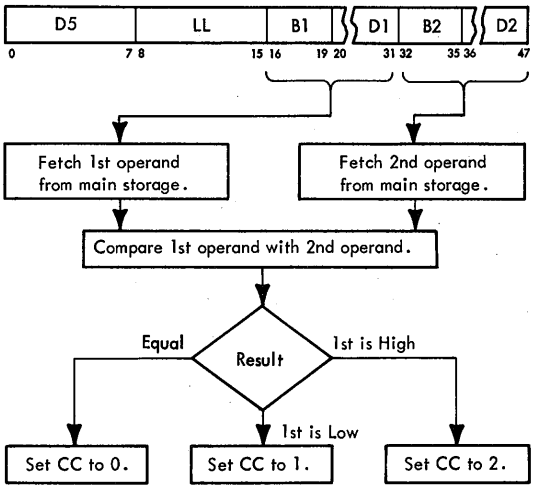


- Conditions at start of execution:
  Main storage request for 1st operand has been issued per D.
  1st 16 bits of instruction, containing immediate operand, are in E.
- CC setting:
  Operands are equal: CC = 0.
  1st operand is less than 2nd operand: CC = 1.
  1st operand is greater than 2nd operand: CC = 2.

The Compare Logical, CLI, instruction, which is in the SI format, compares the first operand with the immediate second operand. Comparison is binary, and is performed left to right. The CC is set according to the result.

Compare Logical, CLC (D5)

- Binarily compare 1st operand (in storage) with 2nd operand (in storage) and set CC according to result.
- SS format:



- Conditions at end of GIS:
  1st operand is in ST.
  2nd operand is in AB.
  ABC and STC are set to select byte(s) to be routed through serial adder.
  First 16 bits of instruction are in E.
- CC setting:
  Operands are equal: CC = 0.
  1st operand is less than 2nd operand: CC = 1.
  1st operand is greater than 2nd operand: CC = 2.
- Because results of operation are not stored into main storage, no special action is required during word overlap.

The CLC instruction is sequenced as follows:

1. The selected AB byte is gated complement to the serial adder with a hot carry to bit 7.
2. The selected ST byte is gated true to the serial adder.
3. The serial adder carry is saved in STAT H.
4. STAT A is set if a nonzero result byte is detected.
5. As each byte is processed, the LL count is decremented and the ABC and STC are incremented.
6. The above routine is continued until a nonzero result is detected in the serial adder, or until the LL count is stepped to zero, with exits for operand fetches when the STC or ABC is stepped to 7.
7. If an exit is made because a nonzero byte is detected, one additional byte will have been gated to the serial adder before the exit is made via the ROS branch. Therefore, STAT H will reflect the carry of the nonzero result byte plus 1. Because STAT H is used to determine the setting of the CC, it is set or reset per the carry of the first nonzero byte encountered.
8. The common end-op routine is used, which sets the CC per the following hardware conditions:

Hardware Conditions	CC Setting
STAT A is reset and equal compare	0
STAT A is set and STAT H is reset	1
STAT A and STAT H are set	2

AND

The AND instruction mixes two operands on a logical AND basis. An AND operation is defined as follows: if *both* operand bits are 1's, the resulting bit is 1; otherwise, the result is a 0. The following example illustrates the AND'ing of two bytes:

Bit positions	0	1	2	3	4	5	6	7
1st operand	1	0	1	0	1	0	1	0
2nd operand	1	0	0	1	1	1	0	0
Result	1	0	0	0	1	0	0	0

Note that only in bit positions 0 and 4 are both operand bits set to 1. Therefore, only bits 0 and 4 of the result are set to 1.

A simplified data flow path for AND, OR, and Exclusive-OR instructions is shown in Figure 3-25. All logical AND's, OR's, and Exclusive-OR's are performed in the serial adder, a byte at a time. The first operand is placed into ST and the second operand is placed into AB. Upon completion of the instruction, the result is placed into ST.

The sequencing of individual bytes from ST and AB through the serial adder is controlled by the STC and the ABC. In the fixed-format RR and RX instructions, the STC and ABC are preset to 4 and incremented to 7 as the four data bytes from ST and AB are routed through the serial adder. For SS instructions, the STC and ABC are

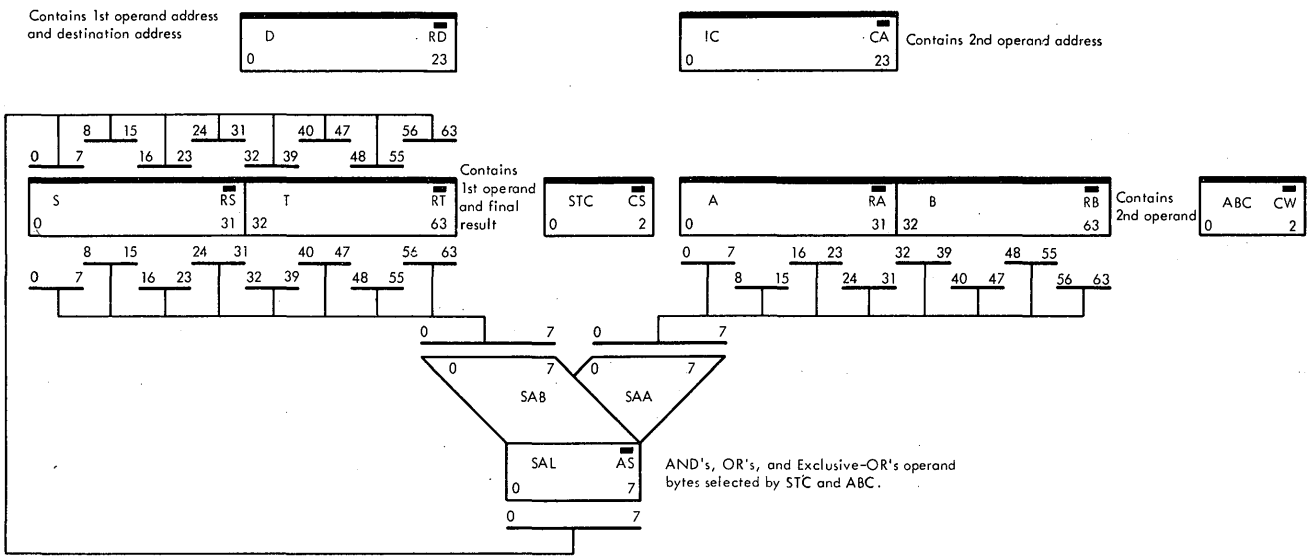
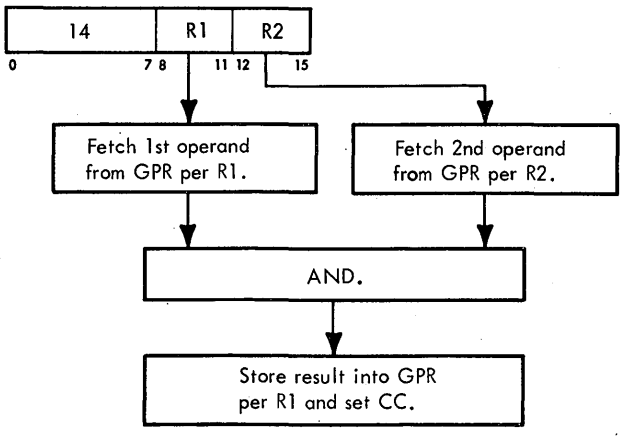


Figure 3-25. Simplified Data Flow for AND, OR, and Exclusive-OR Instructions

preset to values that point to the bytes where the data starts in their respective registers. For SI instructions, in which the second operand is one byte long (I2 field), the ABC points to the second operand location in AB, and the STC points to the first operand in ST.

AND, NR (14)

- AND 1st operand (in GPR, per R1) with 2nd operand (in GPR, per R2) and place result into 1st operand location.
- RR format:

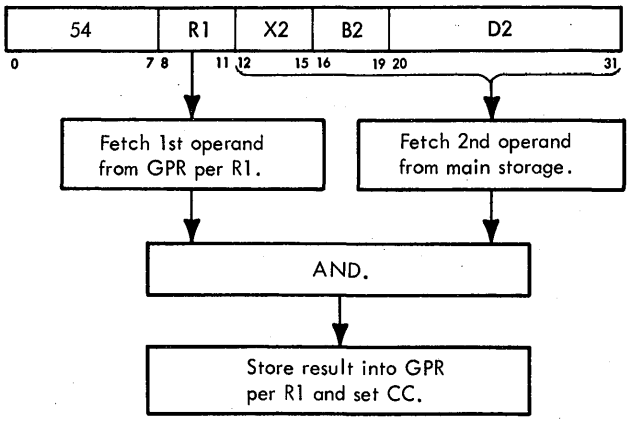


- Conditions at start of execution:
  1st operand is in S and T.
  2nd operand is in A and B.
  Instruction is in E.
- CC setting:
  Result is zero: CC = 0.
  Result is not zero: CC = 1.

The AND, NR, instruction, which is in the RR format, AND's the first operand with the second operand. The AND function is applied left to right, byte by byte.

AND, N (54)

- AND 1st operand (in GPR, per R1) with 2nd operand (in storage) and place result into 1st operand location.
- RX format:



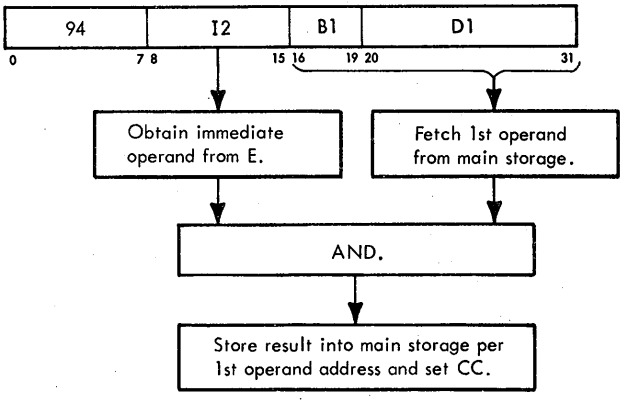
- Conditions at start of execution:
  1st operand is in S and T.
  Main storage request for 2nd operand has been issued per D.
  First 16 bits of instruction are in E.

- CC setting:
  Result is zero: CC = 0.
  Result is not zero: CC = 1.

The AND, N, instruction, which is in the RX format, AND's the first operand with the second operand from main storage. The AND function is applied left to right, byte by byte.

AND, NI (94)

- AND immediate operand (I2 of instruction) with 1st operand (in storage) and place result into 1st operand location.
- SI format:

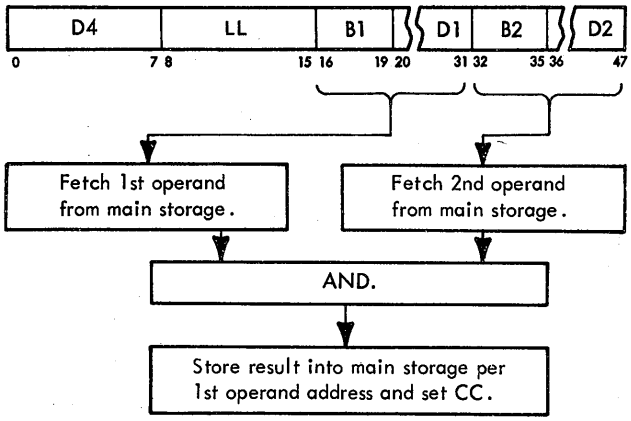


- Conditions at start of execution:
  Main storage request for 1st operand has been issued per D.
  First 16 bits of instruction, containing immediate operand, are in E.
- CC setting:
  Result is zero: CC = 0.
  Result is not zero: CC = 1.

The AND, NI, instruction, which is in the SI format, AND's the first operand with the immediate second operand. The AND function is applied left to right.

AND, NC (D4)

- AND 1st operand (in storage) with 2nd operand (in storage) and place result into 1st operand location.
- SS format:



- Conditions at end of GIS:
  1st operand is in ST.
  2nd operand is in AB.
  ABC and STC are set to select byte(s) to be routed through serial adder.
  First 16 bits of instruction are in E.
- CC setting:
  Result is zero: CC = 0.
  Result is not zero: CC = 1.
- Maximum number of bytes is 256.

The NC instruction specifies the following actions:

1. The selected AB byte is gated to SAA(0–7).
2. The selected ST byte is gated to SAB(0–7).
3. Each AB and ST bit is combined using the serial adder AND function.
4. The adder output is gated back to ST. (STAT A is set if the result byte is not zero.)

Except for the above actions, the byte processing sequence is the same as for the Move Numerics instruction.

OR

The OR instruction mixes two operands on a logical OR basis. An OR operation is defined as follows: if *either* operand bit is a 1, the resulting bit is a 1: otherwise, the

result is a 0. The following example illustrates the OR'ing of two bytes.

Bit positions	0	1	2	3	4	5	6	7
1st operand	1	0	1	0	1	0	1	0
2nd operand	1	0	0	1	1	1	0	0
Result	1	0	1	1	1	1	1	0

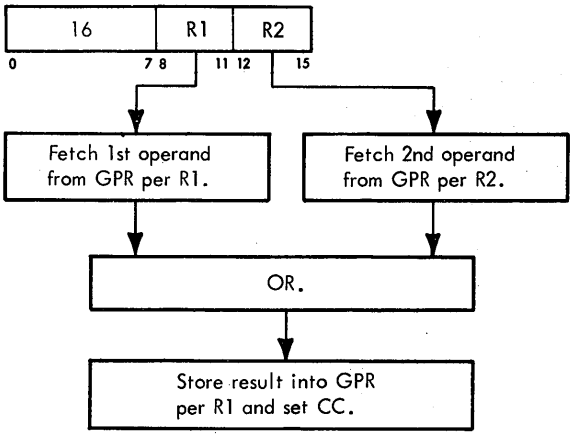
Note that only in bit positions 1 and 7 is neither bit a 1. Thus, only bits 1 and 7 of the result are set to 0, and the remaining bits are set to 1.

The sequencing of operands through the serial adder is similar to the sequencing of the AND instructions. The major difference is that the serial adder applies the OR function.

The OR operation may be executed by an instruction in the RR, RX, SI, or SS format.

OR, OR (16)

- OR 1st operand (in GPR, per R1) with 2nd operand (in GPR, per R2) and place result into 1st operand location.
- RR format:

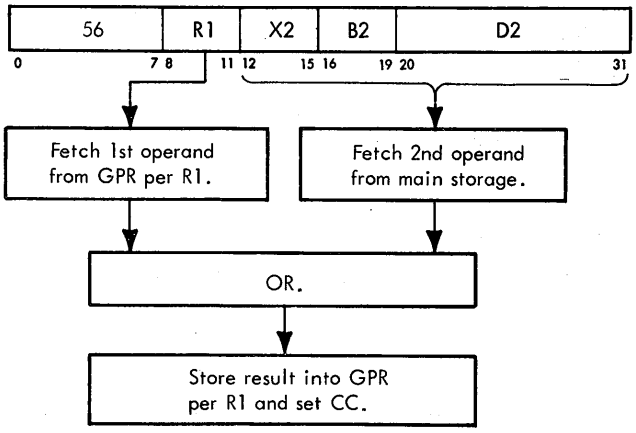


- Conditions at start of execution:
  1st operand is in S and T.
  2nd operand is in A and B.
  Instruction is in E.
- CC setting:
  Result is zero: CC = 0.
  Result is not zero: CC = 1.

The OR, OR, instruction, which is in the RR format, OR's the first operand with the second operand. The OR function is applied left to right, byte by byte.

OR, O (56)

- OR 1st operand (in GPR, per R1) with 2nd operand (in storage) and place result into 1st operand location.
- RX format:

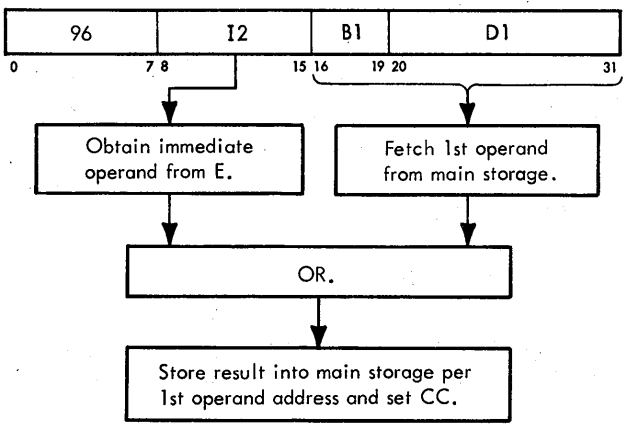


- Conditions at start of execution:
  1st operand is in S and T.
  Main storage request for 2nd operand has been issued per D.
  First 16 bits of instruction are in E.
- CC setting:
  Result is zero: CC = 0.
  Result is not zero: CC = 1.

The OR, O, instruction, which is in the RX format, OR's the first operand with the second operand from main storage. The OR function is applied left to right, byte by byte.

OR, OI (96)

- OR immediate operand (I2 of instruction) with 1st operand (in storage) and place result into 1st operand location.
- SI format:

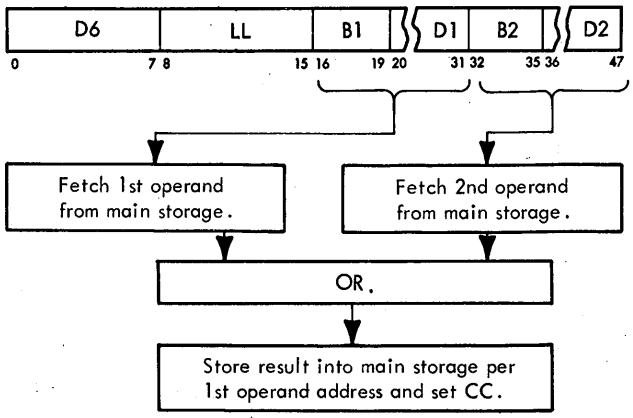


- Conditions at start of execution:
  Main storage request for 1st operand has been issued per D.
  First 16 bits of instruction, containing immediate operand, are in E.
- CC setting:
  Result is zero: CC = 0.
  Result is not zero: CC = 1.

The OR, OI, instruction, which is in the SI format, OR's the first operand with the immediate second operand. The OR function is applied left to right.

OR, OC (D6)

- OR 1st operand (in storage) with 2nd operand (in storage) and place result into 1st operand location.
- SS format:



- Conditions at end of GIS:
  1st operand is in ST.
  2nd operand is in AB.
  ABC and STC are set to select byte(s) to be routed through serial adder.
  First 16 bits of instruction are in E.
- CC setting:
  Result is zero: CC = 0.
  Result is not zero: CC = 1.
- Maximum number of bytes is 256.

The OC instruction specifies the following actions:

1. The selected AB byte and the selected ST byte are gated to the serial adder, where they are combined per the serial adder OR function.
2. The adder output is gated back to the selected ST byte, and the selected mark trigger is set per the STC.
3. STAT A is set if the result is not zero.

Except for the above actions, the byte processing sequence is the same as that for the Move Numerics instruction.

EXCLUSIVE-OR

The Exclusive-OR instruction mixes two operands on a logical Exclusive-OR basis. An Exclusive-OR operation is defined as follows: *if one and only one* of the operand bits is a 1, the resulting bit is a 1; otherwise, the result is a 0. The following example illustrates the Exclusive-OR'ing of two bytes.

Bit position	0	1	2	3	4	5	6	7
1st operand	1	0	1	0	1	0	1	0
2nd operand	1	0	0	1	1	1	0	0
Result	0	0	1	1	0	1	1	0

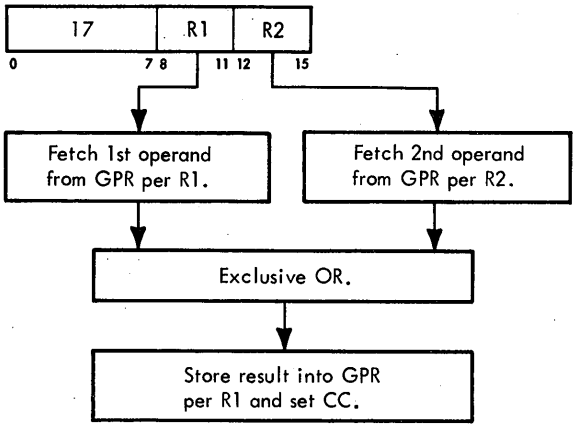
Note that in bit positions 2, 3, 5, and 6 *one and only one* of the operand bits is a 1, and that the corresponding bit positions of the result are set to 1. In bit position 0, both operand bits are 1 and the corresponding result bit is 0. In bit position 1, both bits are 0 and the result is 0.

The sequencing of operands through the serial adder is similar to the sequencing of the AND instructions. The major difference is that the serial adder applies the Exclusive-OR function.

The Exclusive-OR operation may be executed by an instruction in the RR, RX, SI, or SS format.

Exclusive-OR, XR (17)

- Exclusive-OR 1st operand (in GPR, per R1) with 2nd operand (in GPR, per R2) and place result into 1st operand location.
- RR format:



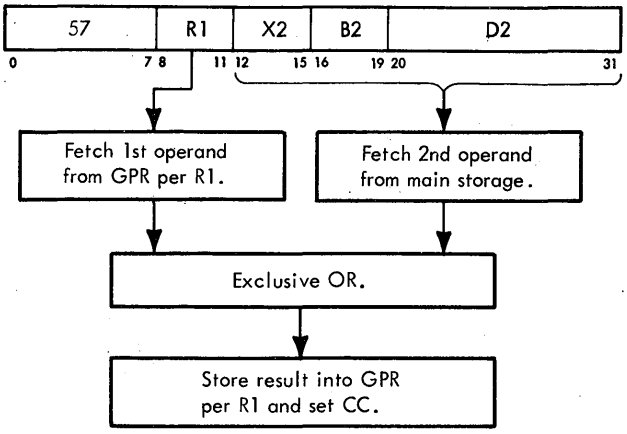
- Conditions at start of execution:
  1st operand is in S and T.
  2nd operand is in A and B.
  Instruction is in E.

- CC setting:
  Result is zero: CC = 0.
  Result is not zero: CC = 1.

The Exclusive-OR, XR, instruction, which is in the RR format, exclusive-OR's the first operand with the second operand. The exclusive-OR function is applied left to right, byte by byte.

Exclusive-OR, X (57)

- Exclusive-OR 1st operand (in GPR, per R1) with 2nd operand (in storage) and place result into 1st operand location.
- RX format:

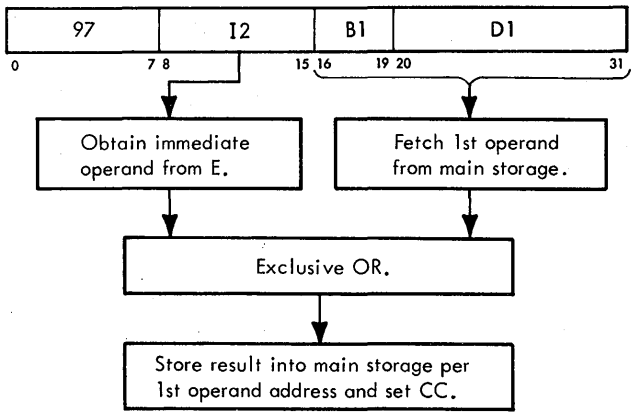


- Conditions at start of execution:
  1st operand is in S and T.
  Main storage request for 2nd operand has been issued per D.
  First 16 bits of instruction are in E.
- CC setting:
  Result is zero: CC = 0.
  Result is not zero: CC = 1.

The Exclusive-OR, X, instruction, which is in the RX format, exclusive-OR's the first operand with the second operand from main storage. The exclusive-OR function is applied left to right, byte by byte.

Exclusive-OR, XI (97)

- Exclusive-OR immediate operand (I2 of instruction) with 1st operand (in storage) and place result into 1st operand location.
- SI format:

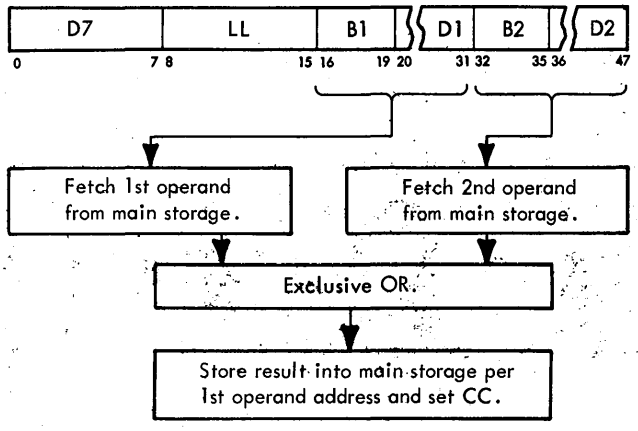


- Conditions at start of execution:
  Main storage request for 1st operand has been issued per D.
  First 16 bits of instruction, containing immediate operand, are in E.
- CC setting:
  Result is zero: CC = 0.
  Result is not zero: CC = 1.

The Exclusive-OR, XI, instruction, which is in the SI format, exclusive-OR's the first operand with the immediate second operand. The exclusive-OR function is applied left to right.

Exclusive-OR, XC (D7)

- Exclusive-OR 1st operand (in storage) with 2nd operand (in storage) and place result into 1st operand location.
- SS format:



- Conditions at end of GIS:
  1st operand is in ST.
  2nd operand is in AB.
  ABC and STC are set to select byte(s) to be routed through serial adder.
  First 16 bits of instruction are in E.

- CC setting:
  Result is zero: CC = 0.
  Result is not zero: CC = 1.

- Maximum number of bytes is 256.

The XC instruction specifies the following actions:

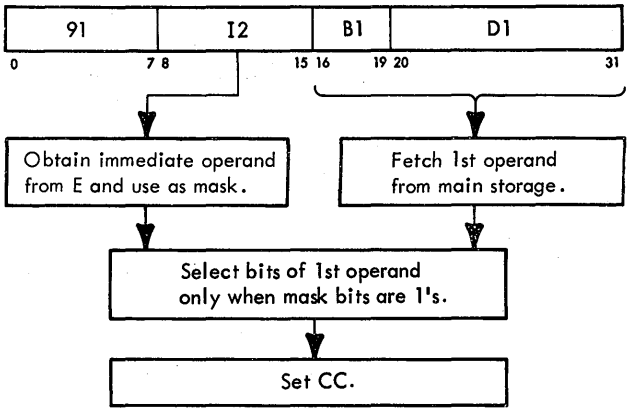
1. The selected AB byte and the selected ST byte are gated to the serial adder, where they are combined per the serial adder Exclusive-OR function.
2. The adder output is gated back to the selected ST byte, and the selected mark trigger is set per the STC.
3. STAT A is set if the result is not zero.

Except for the above actions, the byte processing sequence is the same as that for the Move Numerics instruction.

TEST UNDER MASK, TM (91)

- Set CC according to state of 1st operand bits (in storage) selected by mask bits (I2 of instruction).

- SI format:



- Conditions at start of execution:
  Main storage request for 1st operand has been issued per D.
  First 16 bits of instruction, containing immediate operand, are in E.

- CC setting:
  Selected bits all zero; mask is all zero: CC = 0.
  Selected bits mixed zero and 1: CC = 1.
  Selected bits all 1: CC = 3.

- Storage contents are not changed.

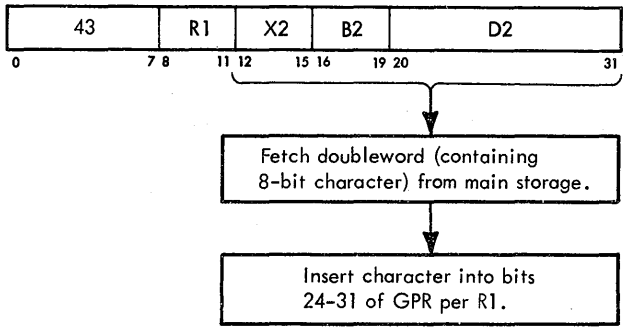
The byte of immediate data, I2, is used as an eight-bit mask. The bits of the mask are made to correspond one for one with the bits of the character in main storage specified by the first operand address.

A mask bit of 1 indicates that the storage bit is selected. When the mask bit is 0, the storage bit is ignored. When all storage bits thus selected are zero, the CC is made 0. The CC is also made 0 when the mask is all-zero. When the selected bits are all-1, the CC is made 3; otherwise, the CC is made 1. The character in storage is not changed.

INSERT CHARACTER, IC (43)

- Insert 2nd operand (byte; in storage) into bits 24–31 of 1st operand location (in GPR, per R1).

- RX format:



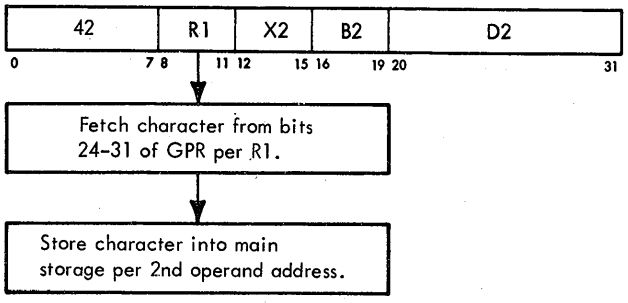
- Conditions at start of execution:
  1st operand is in S and T.
  Main storage request for 2nd operand has been issued per D.
  First 16 bits of instruction are in E.

The Insert Character instruction, which is in the RX format, inserts the byte of second operand into bits 24–31 of the GPR specified by R1. The remaining bits in the GPR are unchanged.

STORE CHARACTER, STC (42)

- Store bits 24–31 of 1st operand (in GPR, per R1) into 2nd operand location (in storage).

- RX format:

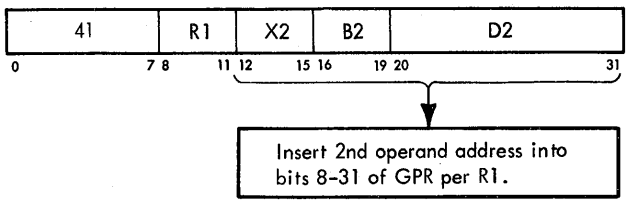


- Conditions at start of execution:
  1st operand is in S and T.
  Main storage request for 2nd operand has been issued per D.
  First 16 bits of instruction are in E.

The Store Character instruction, which is in the RX format, stores the byte (bits 24–31) of first operand into main storage per the second operand address.

LOAD ADDRESS, LA (41)

- Insert 2nd operand address into bits 8–31 of GPR specified by R1.
- RX format:



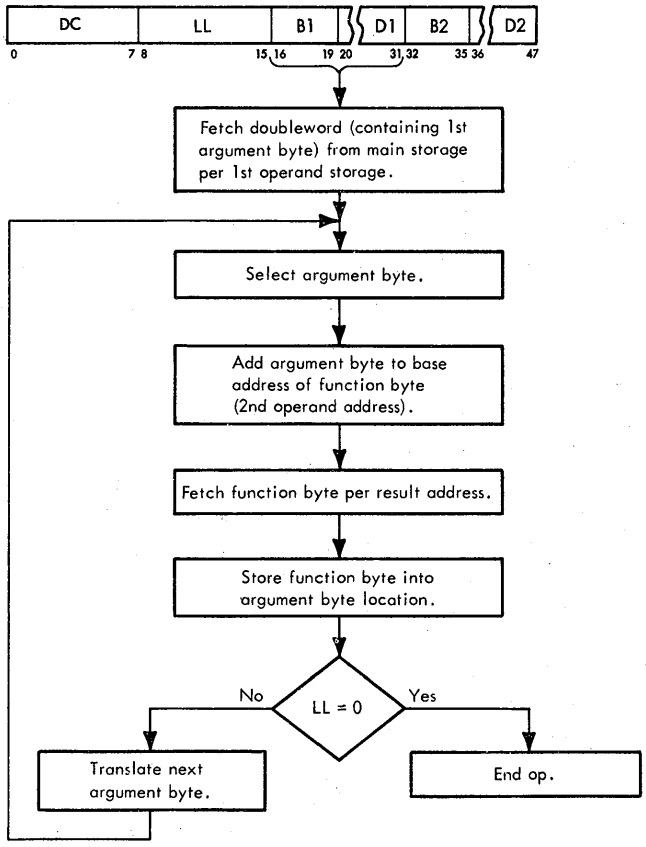
- Conditions at start of execution:
  1st operand is in S and T.
  Main storage request for 2nd operand has been issued per D.
  First 16 bits of instruction are in E.

The address specified by the X2, B2 and D2 fields is inserted into bits 8–31 of the GPR specified by R1; bits 0–7 are made 0's. The address is not inspected for availability, protection, or resolution.

The address computation follows the rules for address arithmetic. Any carries beyond the 24th bit are ignored. The same GPR may be specified by the R1, X2, and B2 instruction field, except that GPR0 can be specified only by the R1 field. In this manner, it is possible to increment the low-order 24 bits of a GPR, other than 0, by the contents of the D2 field of the instruction. The GPR to be incremented should be specified by R1 and by either X2 (with B2 set to zero) or B2 (with X2 set to zero).

TRANSLATE TR (DC)

- Add 1st operand byte (argument; in storage) to effective 2nd operand address, use result as storage address, and place function byte from resulting storage address into corresponding 1st operand byte location.
- SS format:



- Conditions at end of GIS:
  1st operand (destination) is in ST.
  Destination address is in D.
  Source address (contents of GPR per B2, + D2) is in IC.

The Translate instruction selects the first operand bytes for translation one byte at a time, proceeding from left to right. Each argument byte is added to the entire initial address, the second operand address, in the low-order bit positions. The sum is used as the address of the function byte, which then replaces the original argument byte. All

data is valid. The operation proceeds until the first operand field is exhausted. The table is not altered unless an overlap occurs.

At the start of the execution sequence, the first operand has been fetched to ST. A request per the IC has been made for the second operand, but this doubleword from main storage is not used.

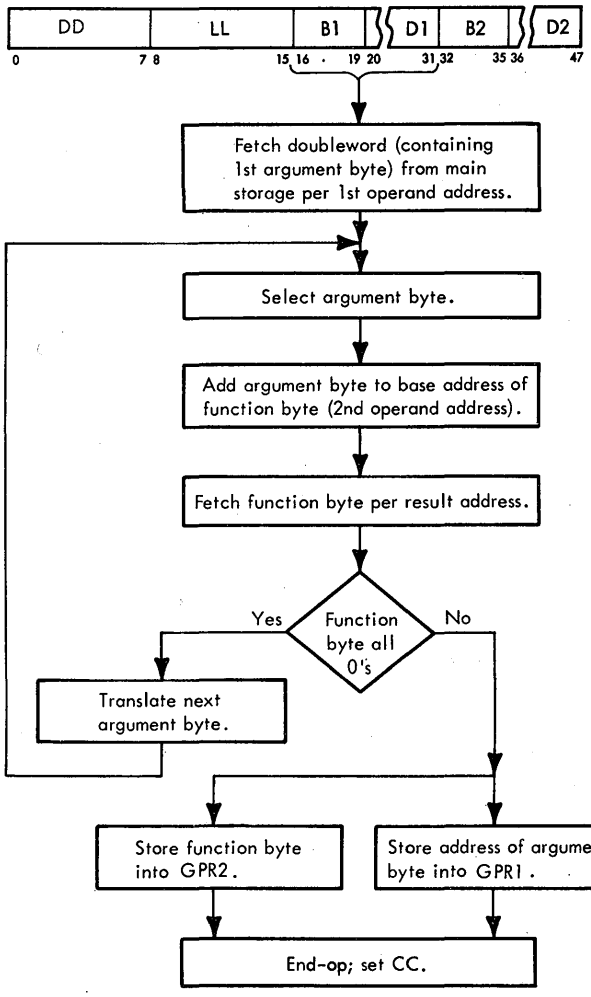
The execution sequence is as follows:

1. The selected ST byte is saved in F. The contents of T are saved in B. (The contents of the IC are saved in A.)
2. T is cleared, the STC is set to 111, and the contents of F (selected destination byte) are placed into T(56–63) via the serial adder.
3. The contents of T are added to the contents of the IC in the parallel adder, and the result is gated back to the IC.
4. A request for the second operand is issued per the IC.
5. The ABC is set per IC(21–23), and the STC is set per D(21–23).
6. The original source address is restored to the IC from A. The destination word is restored to T from B.
7. A word-overlap test was made before the source address was restored to the IC. If no word overlap exists, the table doubleword fetched from main storage is gated to AB. If word overlap is detected, the contents of S are transferred to A (B is already identical with T) and the doubleword from main storage is not used.
8. The selected AB byte is gated via the serial adder to the selected ST byte, and the selected mark trigger is set. The STC and D are incremented by 1. The LL count is decremented by 1.
9. Unless the STC was 7 or LL was zero before stepping, the sequence is repeated for the next destination byte.
10. If LL was zero, the contents of ST are stored and the common end-op sequence is started.
11. If the STC was 7 and LL not equal to zero, the contents of ST are stored and the next destination word is fetched by the common destination fetch sequence, after which the translate sequence is resumed.

TRANSLATE AND TEST, TRT (DD)

- Add 1st operand byte (argument; in storage) to effective 2nd operand address, use result as storage address, and test function byte from resulting storage address. If 0, translate and test next argument byte; if non-0, complete operation by inserting related argument address into GPR1 and function byte into GPR2.

- SS format:



- Conditions at end of GIS:
  1st operand (destination) is in ST.
  Destination address is in D.
  Source address (contents of GPR per B2, + D2) is in IC.

- CC setting:
  All function bytes are zero: CC = 0.
  Nonzero function byte encountered before operand is exhausted: CC = 1.
  Last function byte is nonzero: CC = 2.

The Translate and Test instruction fetches the function bytes in the same manner as the Translate instruction. Each function byte retrieved from the table is inspected for an all-zero combination.

When the function byte is zero, the operation proceeds with the next operand byte. When the first operand field is exhausted before a nonzero function byte is encountered, the operation is completed by setting the CC to 0. The contents of GPR's 1 and 2 remain unchanged.

When the function byte is nonzero, the related argument address is inserted into the low-order 24 bits of GPR1. This address indicates the argument last translated. The high-order eight bits of GPR1 remain unchanged. The function byte is inserted into the low-order eight bits of GPR2. Bits 0–23 of GPR2 remain unchanged. The CC is set to 1 when one or more argument bytes have not been translated. The CC is set to 2 if the last function byte is nonzero.

The following abbreviations are used in this discussion of the Translate and Test execution sequence:

DX: first byte in series of destination bytes.
T(DX): table byte specified by DX.
DX + 1: second byte in series of destination bytes.
T(DX + 1): table byte specified by DX + 1.'
DX + 2: third byte in series of destination bytes.

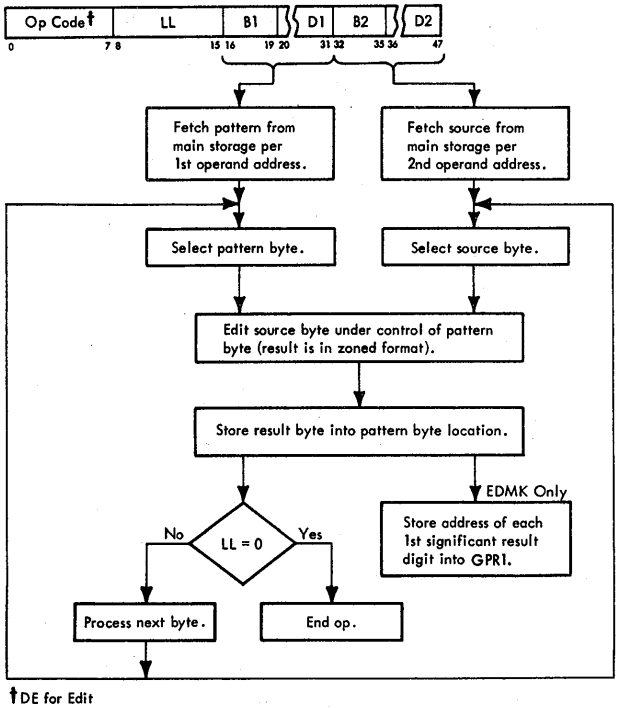
The Translate and Test instruction uses the following execution sequence:

1. First Byte Sequence
   a. The selected ST byte is saved in F.
   b. The contents of ST are transferred to AB.
   c. The STC is set to 3, and the contents of F (DX) are gated, via the serial adder, to byte 3 in S.
   d. Bytes 0, 1, and 2 in S are cleared by gating the contents of SAL to ST and successively decrementing the STC by 1.
   e. The ABC is set per D(21–23), and the STC is set to 3.
   f. The DX in S is added to the contents of the IC, and an IC request is made for T(DX).
   g. A branch per STAT G is made to the T(DX + 1) address generation routine. (STAT G is used to indicate that a table byte has been fetched and is ready for test.)
2. T(DX + 1) Address Generation
   a. The ABC is incremented by 1.
   b. DX is transferred from S to T.
   c. STAT G is set.
   d. The selected AB byte (DX + 1) is gated via the serial adder to byte 3 in S.
   e. The STC is set per IC(21–23).
   f. The T(DX) ingate and T(DX + 1) fetch sequence is started.
3. T(DX) Ingate and T(DX + 1) Fetch Sequence
   a. The table word which contains byte T(DX) is available from main storage, and either the left- or right-half word is gated to T as determined by IC(21). STC(0) is set to 1 to select the correct byte in T. Simultaneously, the contents of T(DX) are subtracted from the contents of IC to restore the table base address.
   b. If LL equals zero, an exit is made to the T(DX) test sequence.
   c. If LL is not zero, DX + 1 (in S) is added to the contents of IC and a fetch request is made for T(DX + 1).
   d. A branch per STAT G starts the T(DX) test sequence.
4. T(DX) Test Sequence and T(DX + 2) Address Generation
   a. The selected byte in T, T(DX), is gated to the serial adder for zero detection and is saved in F.
   b. STAT H is set if the ABC equals zero.
   c. DX + 1 is transferred from S to T.
   d. The STC is set to 3, and the ABC is incremented by 1 (selecting byte to DX + 2).
   e. An exit is made to the LS mark sequence (step 6) if a nonzero result is detected in the serial adder.
   f. An exit is made to the common end-op sequence if the serial adder result is zero and the LL count is zero.
   g. The LL count is decremented and the address in D is incremented by 1.
   h. If STAT H is set, an exit is made to the destination fetch routine.
   i. If no exit conditions are detected, the selected AB byte (DX + 2) is gated via the serial adder to S, and the STC is set per IC(21–23).
   j. The T(DX) ingating and T(DX + 1) fetch sequence is started. The table byte previously referred to as T(DX) has been tested. The table byte previously referred to as T(DX + 1) is now considered T(DX), and the processing loop is resumed.
5. Destination Fetch Routine
   a. Before entering this routine, the ABC has been stepped from 7 to 0 and a fetch request was made for a table byte using byte 0 of the present destination word to generate the table byte address. Because this was an erroneous address, the resulting word from main storage is not used.
   b. STAT's G and H are reset, and the IC is restored to the table base address by subtracting DX + 1.
   c. A fetch request is made per the D-address. The requested doubleword is gated to AB.
   d. The ABC was previously stepped from 0 to 1. It is now decremented to select byte 0 of the new destination doubleword (considered byte DX).
   e. The selected AB byte (DX) is gated via the serial adder to byte 3 in S.
   f. The DX (in S) is added to the contents of IC, and a fetch request is made for T(DX).
   g. Because STAT G is reset, the T(DX + 1) address generation routine is started.

6. LS Mark Sequence
   a. This routine is entered when a nonzero table byte is detected in the serial adder or when the LL count equals zero. (The last table byte tested is in F.)
   b. If the table byte was nonzero, STAT G is reset.
   c. E(8–15) is cleared and used for LAR addressing.
   d. GPR1 is accessed per E(8–15) + 1 and its contents transferred to T.
   e. The STC is incremented to 4, and byte 4 of ST is gated via the serial adder back to ST. Simultaneously, the contents of D are gated to T via the parallel adder.
   f. The contents of T are stored into GPR1; E(8–11) is incremented twice, and the STC is set to 7.
   g. The contents of GPR2 are transferred to T. The contents of F are gated via the serial adder to T(56–63), and the contents of T are stored into GPR2.
   h. STAT A is set if the byte in F was not zero.
   i. The common end-op sequence is started, which sets the CC per STAT's A and G.

EDIT AND EDIT AND MARK, ED AND EDMK (DE AND DF)

- Edit: change format of source (2nd operand; in storage) from packed to zoned, edit source under control of pattern (1st operand; in storage), and place result into 1st operand location.
- Edit and Mark: same as Edit, but in addition place location of each 1st significant digit into GPR1.
- SS format:



†DE for Edit
DF for Edit and Mark

- Conditions at end of GIS:
  Pattern (destination operand) is in ST.
  Pattern address is in D.
  Source address (contents of GPR per B2, + D2) is in IC.
  First 16 bits of instruction are in E.
- CC setting:
  Result is zero: CC = 0.
  Result is less than zero: CC = 1.
  Result is greater than zero: CC = 2.

The Edit instruction changes the format of the source (second operand) from packed to zoned, edits the source under control of a pattern (first operand), and places the result into the first operand location. The Edit and Mark instruction performs the same functions and, in addition, places the location of each first significant digit into GPR1. Both instructions are in the SS format, and share a common ROS microprogram with an exit to a separate mark routine for the Edit and Mark instruction. Because the results of a word-overlap condition are unpredictable, no special action is taken when this condition occurs.

Introduction to Edit Operation

- Edit instruction is used to:
  Eliminate high-order zeros.
  Provide asterisk protection.
  Handle sign control (CR).
  Provide punctuation.
  Blank out an all-zero field.
  Protect decimal point by use of significance start character. (This character can also be used to retain high-order zeros when desired.)
  Edit multiple adjacent fields via field separator character.

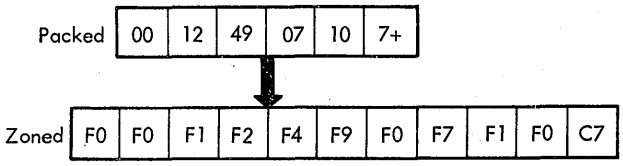
The edit operation is used to produce easy-to-read documents by inserting proper punctuation into a data record. The data to be edited (second operand) is called the "source" and must be in the packed BCD format. Consider the following source field:

00	12	49	07	10	7+

For the above field to be printed in a document, it must first be converted into the zoned format (USASCII-8 or EBCDIC). One function of the edit operation is to change the source field from packed to zoned format.† If

† Each time the digit from the source field replaces a digit select character, the four-bit digit has the proper EBCDIC or USASCII-8 zone bits inserted. PSW(12) determines whether the EBCDIC or USASCII-8 zone is inserted. For this discussion, it is assumed that the system is in EBCDIC mode.

changing from packed to zoned format were all that was necessary to produce a legible report, the Edit instruction would not be necessary, because the Unpack instruction would be sufficient. For instance, if the above packed BCD operand were changed to the EBCDIC zoned format, it would look like this:



If the above zoned BCD field were printed, it would look like this:

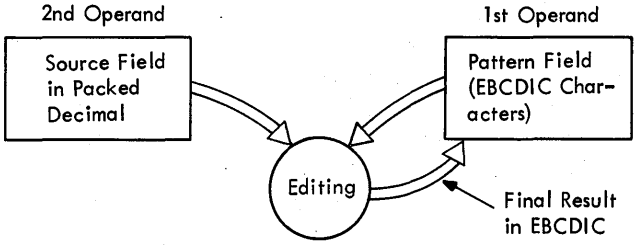
0 0 1 2 4 9 0 7 1 0 7 +

By examining the printed document, one could tell that it was a positive number with a low-order digit of 7. However, the printed document is still not legible. If, for instance, the number represents money, it would be desirable to obtain the following printed result:

$1,249,071.07

This result would require insertion of the commas and decimal points in the right place, as well as other editing. This is the main function of the edit operation.

The edit operation involves moving the source field (second operand) into the pattern field (first operand). The pattern field is initially made up of EBCDIC characters that control the editing. The final edited result replaces the pattern field:



As a rule, the second operand is shorter than the first because one source byte yields two result bytes.

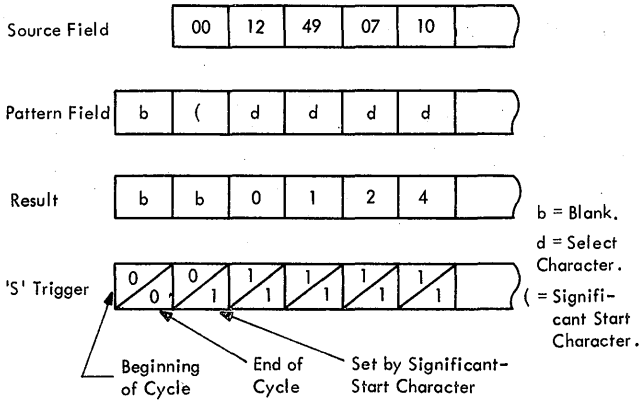
The characters in the pattern field determine the editing that takes place. The high-order (leftmost) character in the pattern field is known as the "fill" character. Any of the 256 possible EBCDIC combinations can be used as the fill character. In many edit operations, however, the fill character consists of an EBCDIC blank (0100 0000). The blank character (represented by "b" in the discussion that follows) is not printed out and facilitates programmed blanking of high-order zero fields.

Besides the fill character, three more control characters in the pattern field have special meaning: the digit select character, the significant start character, and the field separator character. These characters can appear anywhere in the pattern field.

For purposes of discussion, the digit select character is represented by "d." (The binary code for the digit select character is 0010 0000, or a hex 20.) When a digit select character is encountered in a pattern field, it is usually replaced with a digit from the source field. If the digit in the source field is a high-order zero, however, the digit select character is replaced by the fill character. By using a blank as the fill character, high-order zeros can be blanked out. If an asterisk is used as the fill character, asterisk protection for paychecks can be achieved.

Because the digit select character may be replaced by either a source digit or the fill character, the system needs some way of knowing which of the two to choose. This function is provided by a special control trigger, known as the 'S' trigger. When the 'S' trigger is set, it indicates that significant source digits are being processed. Consequently, the digit select characters in the pattern field are replaced with the digits from the source field. At the beginning of the edit operation, the 'S' trigger is always reset. As long as the 'S' trigger is reset, the digit select characters in the pattern field are replaced with the fill character.

As stated previously, the 'S' trigger is set when a nonzero digit is detected in the source field. The 'S' trigger is also set if a significant start character is detected in the pattern field. The significant start character has a bit code of 0010 0001 (hex 21). In this discussion, the symbol for the left parenthesis is used to represent the significant start character. When a significant start character is detected in the pattern field, it is replaced by either a digit from the source field or the fill character. A typical edit operation using the b, d, and (characters is illustrated and explained below.



The edit operation begins by examining the fill character (which is b in the above case). If it is not a

digit select or a significant start character, it is left in place in the pattern field. Then, the next pattern character is examined. Because this is a significant start character, the next high-order source digit is examined. Because this source digit is zero and the 'S' trigger is reset (at this time), the significant start character is replaced with the fill character. However, the significant start character sets the 'S' trigger so that all subsequent source digits are significant. The remaining pattern characters in the above example are digit select characters, which are replaced with source digits.

Once significance is started, the 'S' trigger remains set until the sign of the source operand is examined. If a plus sign is detected, the 'S' trigger is reset; if the source has a negative sign, the 'S' trigger remains set because the usual method of indicating a negative quantity in a printed report is with the letters "CR". The following example illustrates how the state of the 'S' trigger identifies the number as a positive or negative quantity:

Source Field (6 Bytes)	00	12	49	07	10	7+								
Pattern Field (14 Bytes)	b	d	d	d	d	d	d	d	d	d	d	d	C	R
Result	b	b	b	1	2	4	9	0	7	1	0	7	b	b

When a pattern character is not one of the three special control characters and the 'S' trigger is set, the character is not changed. If the 'S' trigger is reset, the character is replaced by the fill character. Because detection of a positive sign resets the 'S' trigger, the remaining pattern characters (CR) are replaced by the fill character. If the sign of the source field had been minus, the 'S' trigger would have remained set and characters CR would have been left in the pattern field.

As stated previously, the 'S' trigger is reset when a plus sign is detected in the source field. The 'S' trigger is also reset if a field separator character is detected in the pattern field. The field separator character has a bit code of 0010 0010 (hex 22). In this discussion, the symbol for the right parenthesis represents the field separator character.

The field separator character is used when two or more packed BCD source fields are to be edited with one instruction into a single pattern field. The following edit example illustrates the use of the field separator character.

Source Field (6 Bytes)	01	77	6+	00	00	0+															
Pattern Field (20 Bytes)	b	d	d	(	·	d	d	b	C	R	)	d	d	d	·	d	d	b	C	R	
Result	b	b	1	7	·	7	6	b	b	b	b	b	b	b	b	b	b	b	b	b	
Prints Out	17.76																				

Note that after the field separator character resets the 'S' trigger, the source field does not contain any significant digits. As a result, the pattern characters are replaced by the fill character (blank).

Introduction to Edit and Mark Operation

The Edit and Mark operation is identical with that of the Edit instruction, except for the additional function of inserting a byte address into bits 8–31 of GPR1. The byte address is inserted each time the 'S' trigger is reset and a nonzero digit is inserted in the result field. The address is not inserted when significance is forced by the significant start character of the pattern. Bits 0–7 of GPR1 are not changed. The Edit and Mark instruction facilitates the programming of floating currency-symbol insertion. The character address inserted into GPR1 is 1 more than the address where a floating currency-sign would be inserted. (The Branch on Count instruction, with zero in the R2 field, may be used to reduce the inserted address by 1.)

The character address is not stored when significance is forced. Therefore, the address of the character following the significant start character should be placed into GPR1 before the Edit and Mark instruction is executed.

When a single instruction is used to edit several numbers, the address of the first significant digit of each number is inserted into GPR1. Only the last address will be available after the instruction is completed.

General Data Handling

Special circuits are packaged in the serial adder for use in the Edit and Edit and Mark instructions. These circuits consist of:

1. A decoder of serial adder bus B (SBB) to detect a digit select, significant start, or field separator character in the selected ST byte.
2. 'Right digit' trigger for AB digit selection.
3. Controls for stepping the ABC.
4. Controls for determining which data (i.e., ST byte, F, or AB digit with zone) is to be used as the result byte, and controls for gating this data to the serial adder.
5. Zero detection of the selected AB digit.
6. Sign detection of the low-order digit of the selected AB byte.
7. Detection of a mark condition.
8. The 'S' trigger with associated set-reset controls.
9. Controls for setting or resetting STAT's.

The destination field is considered a pattern field and is processed one byte at a time, from left to right, under control of the STC. Each ST byte is gated to SBB for decoding and is replaced by a byte of data which, depending on decoded conditions, may be:

1. Original data of ST byte.
2. A selected digit of AB with a zone inserted into the high-order four bits.
3. A fill character, which is contained in F.

The source field is processed, one digit at a time, from left to right, under control of the ABC and 'S' trigger, which selects which digit of a byte is to be used. The selected AB digit is examined only if a digit select or significant start character appears in the selected ST byte. The selected AB digit is not necessarily used as part of the result byte, but the next digit to be processed is selected after the digit has been examined.

Microprogram Description

The flowchart for the Edit and Edit and Mark microprogram is shown in Diagram 5-411, FEMDM. At the start of the execution sequence, the fill character is gated from ST (per the STC) through the serial adder to F. A 2-cycle data-processing sequence is then started and is repeated until all destination operand bytes have been processed. Exits from this sequence are made when required for operand fetching or marking, after which this sequence is continued. The microprogram may be divided into three parts: (1) first cycle, (2) second cycle, and (3) exit conditions.

First Cycle

The first cycle is a decode cycle; no data is transferred. The selected ST byte is gated to SBB, and the selected AB byte is gated to serial adder bus A (SBA) with the digit to be examined determined by the 'right digit' trigger. The decode circuits are activated by ROS. Decoding of SBB, SBA, and the 'S' trigger governs the selection of appropriate inputs to the serial adder, and also whether the 'S' trigger is set or reset. STAT A is set if the selected source digit (in AB) is a nonzero digit. However, if a field separator character is decoded at SBB, STAT A is reset.

STAT E is set if an invalid digit is decoded in SBA(0–3). A 1 is added to D (except for the first entry from another sequence) to keep the byte address in D at the same value as the STC for use in the marking sequence. A mark condition is detected and latched for a branch condition of the Edit and Mark instruction.

Second Cycle

At the start of this cycle, data is gated to the serial adder by hardware controls as explained in the first cycle. The second cycle performs the following control functions:

1. The serial adder output is gated back to the selected ST byte, and the appropriate mark trigger is set.
2. The STC is incremented, and the LL count in E(8–15) is decremented by ROS control.
3. The ABC is incremented by hardware controls.
4. If required, the 'digit select' trigger is complemented. This action is conditional on the following:
   a. The digit selection of AB is changed only if a significant start or digit select character was decoded during the first cycle.
   b. When a sign code is decoded in SBA(4–7) at the time bits 0–3 are selected for examination, the low-order digit (sign) is skipped by stepping the ABC and leaving the 'right digit' trigger reset.
5. If required, exit to a separate routine is made via an eight-way ROS branch, for end-op, operand fetching, or marking. If no exit conditions exist, the execution sequence is repeated.

Exit Conditions

Exits from the data processing sequence are made when one or more of the following conditions exist:

1. Edit and Mark instruction is being executed and a mark condition is detected.
2. LL = 0 or STC = 7.
3. ABC = 7.

Where more than one of the above conditions exists, a branch is made to the proper sequence in the order they are listed above. An explanation of each sequence is given below:

1. Exit on Detection of Mark Condition
   Exit to the mark sequence is made regardless of other branch conditions. Special action is taken to return counter values to what they were before entering the mark sequence, so they can be retested. STAT H is set if the ABC has just stepped from 7 to 0, to record this condition. The contents of AB are destroyed by gating E(8–15) + 1 via the parallel adder to A, and the contents of T via the parallel adder to B. E(8–15) is cleared, and GPR1 is transferred to T using E(12–15) + 1 as the LAR address. The contents of D, the byte address of the last byte processed, are placed into T(40–63). T(32–39) is retained by gating it through the serial adder and back to T at the same time the D-PAL-T transfer occurs. The contents of T are now transferred back to GPR1. Registers and counters are restored to their original contents. The source operand is replaced in AB by refetching it from main storage, and a test is made, via a ROS branch, for any other exit condition which may have been present at the time the mark sequence was started. If no other exit condition exists, the data-processing sequence is resumed.
2. Exit on LL = 0 or STC = 7 (End-Op or Destination Fetch)
   STAT D is set if the ABC also equals 7, and a destination store is started and a test is made for invalid data. If STAT E has been set, an interruption code trigger is set and an end-op sequence is started. If STAT E is not set, a test is made for an end-op condition via a ROS branch. If the LL count has been stepped to all 1's, an end-op sequence is started which sets the CC, restores the instruction address to the IC, and resets STAT G. If an end-op condition does not exist, D is incremented and a fetch request is initiated for the next doubleword of destination operand. A test

is made to see whether a source fetch is also required (ABC = 0 and STAT D set). If not, the data-processing sequence is resumed.

3. Exit on ABC = 7 (Possible Source Fetch)
   A further test must be made to determine whether the last byte of AB has been processed. This is determined by testing the ABC for an all-zero count (i.e., the ABC was stepped from 7 to 0 in the previous cycle). If the ABC is not zero, the data-processing routine is restarted; otherwise, the IC is incremented by 8, and a fetch is initiated for the next doubleword of source operand. This source fetch sequence is common to all VFL logical instructions and incorporates the word-overlap test. However, this test does not affect the edit operation. The source doubleword from main storage is gated from the SDBO to AB, and the data-processing sequence is resumed.

SHIFT

Four logical shift instructions are available:

1. Shift Left Single. Shifts a 32-bit operand left.
2. Shift Left Double. Shifts a 64-bit operand left.
3. Shift Right Single. Shifts a 32-bit operand right.
4. Shift Right Double. Shifts a 64-bit operand right.

The second operand address is not used to address data. Rather, its low-order six bits indicate the number of bit positions to be shifted; the rest of the address is ignored.

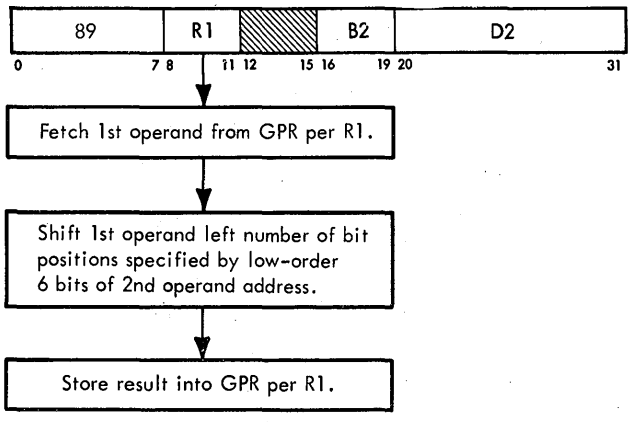
Shifting is accomplished as follows:

1. Left 1 from T to PAA.
2. Left 2 from AB to PAB.
3. Left 4 from PAA or PAB to PAL.
4. Right 4 from PAA or PAB to PAL.

Shifts of right 3 or less are obtained by combining left 1, left 2, or left 3 shifts with a right 4 shift.

Shift Left Single, SLL (89)

- Shift 1st operand (in GPR, per R1) left number of bit positions specified by low-order 6 bits of 2nd operand address.
- RS format:

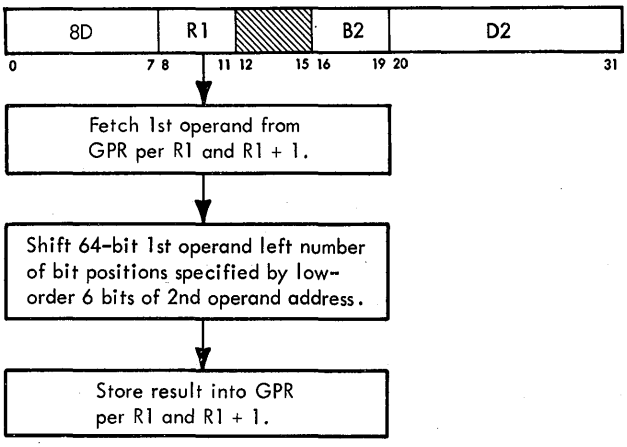


- Conditions at start of execution:
  1st operand is in S and T.
  D(18–23) specifies amount of shift.
  First 16 bits of instruction are in E.

The Shift Left Single, SLL, instruction shifts the 32-bit first operand left the number of bit positions specified by the low-order six bits of the second operand address. All 32 bits of the GPR participate in the shift. High-order bits are shifted out without inspection and are lost. Zeros are supplied to vacated low-order GPR positions. This instruction shares the same microprogram as the fixed-point Shift Left Single, SLA, instruction (Section 2 of this chapter).

Shift Left Double, SLDL (8D)

- Shift 1st operand (in GPR, per R1 and R1 + 1) left number of bit positions specified by low-order 6 bits of 2nd operand address.
- RS format:

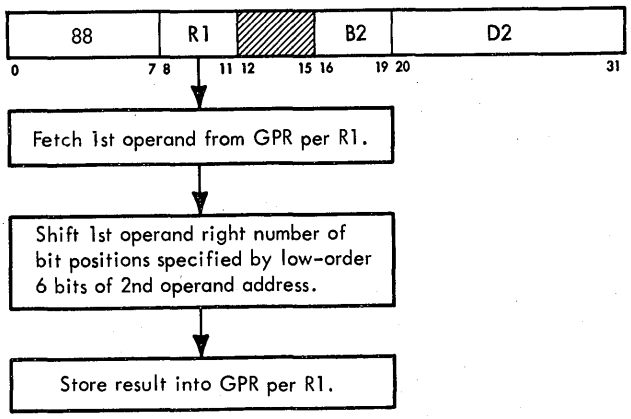


- Conditions at start of execution:
  1st operand is in ST.
  D(18–23) specifies amount of shift.
  First 16 bits of instruction are in E.

The Shift Left Double, SLDL, instruction shifts the 64-bit first operand left the number of bit positions specified by the low-order six bits of the second operand address. The R1 field of the instruction specifies an even/odd pair of GPR's and must contain an even GPR address. An odd value for R1 is a specification exception and causes a specification program interruption. All 64 bits of the even/odd GPR pair participate in the shift. High-order bits are shifted out of the even-numbered GPR without inspection and are lost. Zeros are supplied to vacated low-order positions of the odd-numbered GPR. This instruction shares the same microprogram as the fixed-point Shift Left Double, SLDA, instruction (Section 2 of this chapter).

Shift Right Single, SRL (88)

- Shift 1st operand (in GPR, per R1) right number of bit positions specified by low-order 6 bits of 2nd operand address.
- RS format:

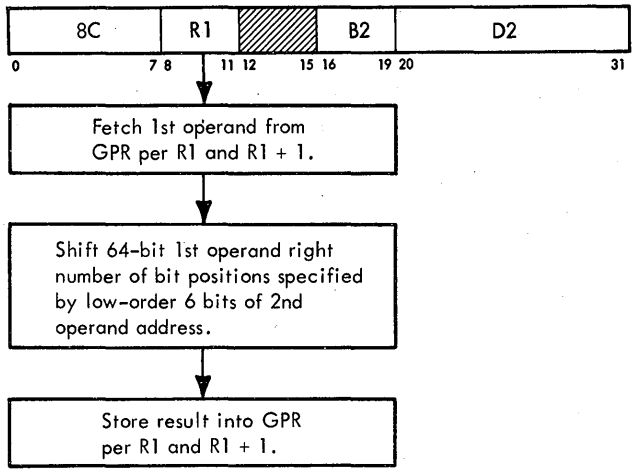


- Conditions at start of execution:
  1st operand is in S and T.
  D(18–23) specifies amount of shift.
  First 16 bits of instruction are in E.

The Shift Right Single, SRL, instruction shifts the 32-bit first operand right the number of bit positions specified by the low-order six bits of the second operand address. All 32 bits of the GPR participate in the shift. Low-order bits are shifted out without inspection and are lost. Zeros are supplied to vacated high-order GPR positions. This instruction shares the same microprogram as the fixed-point Shift Right Single, SRA, instruction (Section 2 of this chapter).

Shift Right Double, SRDL (8C)

- Shift 1st operand (in GPR, per R1 and R1 + 1) right number of bit positions specified by low-order 6 bits of 2nd operand address.
- RS format:



- Conditions at start of execution:
  1st operand is in ST.
  D(18–23) specifies amount of shift.
  First 16 bits of instruction are in E.

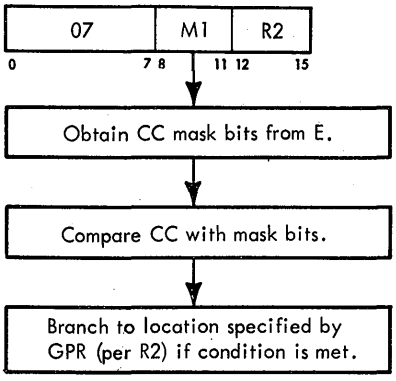
The Shift Right Double, SRDL, instruction shifts the 64-bit first operand right the number of bit positions specified by the low-order six bits of the second operand address. The R1 field of the instruction specifies an even/odd pair of GPR's and must contain an even GPR address. An odd value for R1 is a specification exception and causes a specification program interruption. All 64 bits of the even/odd GPR pair participate in the shift. Low-order bits are shifted out of the odd-numbered GPR without inspection and are lost. Zeros are supplied to vacated high-order positions of the even-numbered GPR. This instruction shares the same microprogram as the fixed-point Shift Right Double, SRDA, instruction (Section 2 of this chapter).

www.IBM360.info

This section discusses the nine branching instructions. The instructions use the RR, RX, and RS formats. For a discussion of branching, operand addressing, instruction formats, data flow, and program interruptions, see Chapter 1.

BRANCH ON CONDITION, BCR (07)

- Branch to location specified by GPR (addressed by R2) if state of CC is as specified by M1.
- RR format:



- Conditions at start of execution:
  Branch address is in D.
  3-cycle storage request for branch-to instruction has been issued per D if branch is successful.
  Instruction is in E.
- If branch is unsuccessful, 3-cycle storage request to refill Q will be issued per IC, if required.
- Branch is unsuccessful if R2 = 0 or if condition is not met.

The Branch on Condition, BCR, instruction, which has an RR format with an op code of 07, replaces the next sequential instruction address with the branch address located in the GPR specified by R2 if the CC agrees with the corresponding mask bit(s) in the M1 field. The M1 field is used as a four-bit mask. The four bits of the mask correspond, left to right, with the four CC's (0, 1, 2, and 3) as follows:

M1 Field	Mask Position Value	CC
8	8	0
9	4	1
10	2	2
11	1	3

The branch is successful whenever the CC has a corresponding mask bit(s) of 1.

When a branch is to be made on more than one CC, the pertinent CC's are specified in the mask as the sum of their mask position values. A mask of 12, for example, specifies that a branch is to be made on CC's 0 and 1.

When all four mask bits are 1's, that is, the mask position value is 15, the branch is unconditional. When all four mask bits are 0 or when R2 = 0, the branch instruction is equivalent to a No-Operation.

At the start of execution, the instruction is in E and the branch address is in D. For a BCR instruction, the storage request can be generated from two possible places, depending upon whether the branch is successful (Diagram 5-501, FEMDM). If the branch is successful, the storage request is generated per D. If the branch is unsuccessful, the storage request is generated per the IC if Q needs to be refilled. Because the CC's, which have to be compared with the mask bits, are set during the execution phase of a previous instruction and are tested during I-Fetch of the branch instruction, success of the branch can be determined beforehand. Therefore, the BCR instruction knows whether it is successful or unsuccessful before instruction execution.

The branch-to address is always placed in D by the I-Fetch sequence. If the branch is successful, a request is made per D by means of the 'I-Fetch reset' micro-order. The correct halfword within the doubleword from main storage is then gated into Q, and from Q to R per D(21,22). The contents of D are updated by 8 and placed into the IC to address the next sequential doubleword from main storage. If the branch is unsuccessful, the storage request is issued per the IC, if Q needs to be refilled, during I-Fetch (by means of the 'I-Fetch reset' micro-order), and the data from main storage is gated to Q during the execution of the unsuccessful branch.

Successful Branch

The 'execute' and 'PSC' triggers are reset if the branch is successful. The triggers are set if the branch instruction is the subject instruction of an Execute instruction.

The contents of D are transferred to PAA(40–63). Then 8 is added to PAA, and the result (address of next doubleword to be operated on) is transferred to the IC. D(21,22) is now tested. If D(21,22) = 11, it signifies that the next instruction to be executed, when the data is gated into Q from the SDBO, occupies the last halfword of Q, and a request must be made to obtain additional instructions for Q. If D(21,22) equals a value other than

11, then the next instruction to be executed is in some halfword other than the last halfword of Q.

Assume that D(21,22) = 11. In this case, a storage request per the updated instruction address in the IC is given to refill Q. At this time, the data (branch-to instruction) that was requested during I-Fetch of the branch instruction is present at the SDBO and is gated into Q. From Q, the data is gated to R per D(21,22), thus placing the last halfword of Q into R.

The contents of the IC are transferred to the parallel adder, where they are updated by 8 and replaced into the IC to address the next doubleword from main storage. After the IC has been updated, the next sequential doubleword (requested during execution of the branch instruction) is gated from the SDBO into Q. A normal end-op cycle completes the operation.

Now assume that D(21,22) equals a value other than 11 on a successful branch. This condition means that the next instruction to be executed is contained in either the first, second, or third halfword of Q when the data from storage is gated into Q. The data which was requested during I-Fetch of the branch instruction is now present at the SDBO and is gated into Q. The halfword that contains the next instruction to be executed is then gated into R per D(21,22). Format decoding is normally accomplished from R and instruction address decoding from the IC. Because the data to be decoded has just been placed into R and the IC, it is not yet stable and therefore cannot be decoded during this cycle. Rather than delaying a cycle until the information is stable, a branch end-op cycle is taken. This cycle allows decoding of the halfword (containing the next instruction) from the SDBO as the data is transferred from the SDBO to Q and decoding of the instruction address from D.

Unsuccessful Branch

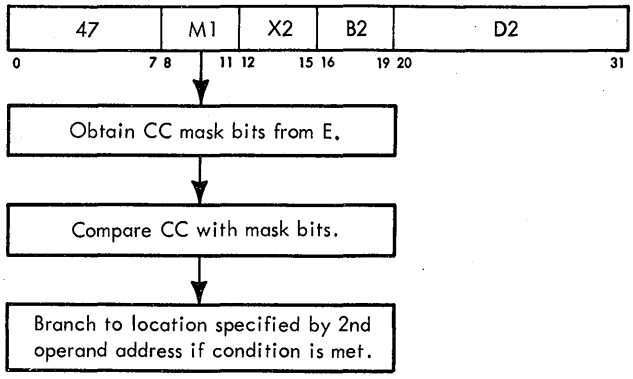
Assume, now, that the branch had been found to be unsuccessful (point B, Diagram 5-501, Sheet 2). Because the storage request was generated per the IC, IC(21,22) is now tested. Assume that IC(21,22) = 11. This value means that the next instruction to be executed is located in the last halfword of the doubleword from main storage that contains the branch instruction. This instruction is in R. Because a request to refill Q per the IC was made during I-Fetch, the IC must be updated. Accordingly, the contents of the IC are now transferred to PAB(40–63). Then 8 is added to PAB, and the result is transferred to the IC and D. At this time, the data that was requested during I-Fetch of the branch instruction is present at the SDBO and is gated into Q. The 'execute' trigger is now tested. If the trigger is set, the IC is reduced by 8 (because 8 had been added to it and the I-Fetch request was blocked by STAT G being set by the Execute instruction), a normal end-op cycle is taken, and an address-store-compare refill of Q will be performed. If the 'execute' trigger is reset, a branch end-op cycle is taken.

Now assume that IC(21,22) = 00, 01, or 10 and that the branch instruction was unsuccessful (point C, Diagram 5-501, Sheet 2). In this case, the next instruction to be executed is in either the first, second, or third halfword of Q, and Q does not need to be refilled. Therefore, a normal end-op cycle is taken, the next instruction format is decoded from R, and the instruction address is decoded from the IC.

A unique situation occurs for the BCR instruction when the 'PSC' trigger is set by an Execute instruction. This situation causes a Q-register refill following the Execute instruction. A branch is taken to the address-store-compare ROS microprogram, 8 or 16 is subtracted from the IC to select the doubleword that contains the instruction following the Execute instruction, and a storage request per the IC is made for that doubleword.

BRANCH ON CONDITION, BC (47)

- Branch to location specified by 2nd operand address if state of CC is as specified by M1.
- RX format:



- Conditions at start of execution:
  Branch address is in D.
  3-cycle storage request for branch-to instruction has been issued per D, if branch is successful.
  First 16 bits of instruction are in E.
- If branch is unsuccessful, 3-cycle storage request to refill Q will be issued per IC, if required.

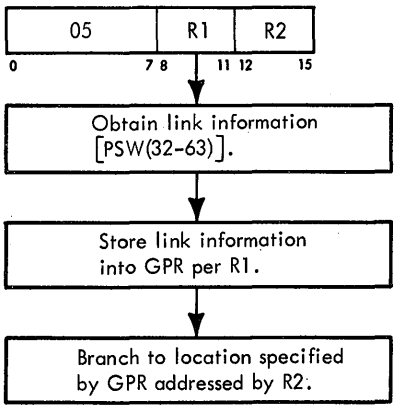
The BC instruction is similar to the BCR instruction, differing only when IC(21,22) = 00 when the branch is unsuccessful (point B, Diagram 5-501, Sheet 2). This value means that the next instruction to be executed is to come from the first halfword of the doubleword which has been requested during I-Fetch. The contents of the IC are updated by 8 and placed into the IC and D. The doubleword from main storage requested during I-Fetch is

now gated from the SDBO to Q. The first halfword from Q is then transferred to R.

The 'execute' trigger is now tested. If reset, the branch instruction ends in a branch end op, and format decoding of the next instruction takes place off the SDBO because R is not stable. If the 'execute' trigger is set, the branch instruction was the subject instruction of an Execute instruction, and the present contents of the IC are incorrect because the IC was increased by 8 during this instruction. The contents of the IC are thus reduced by 8 and replaced in the IC. A normal end op completes the operation. Because the BC instruction was the subject of an Execute instruction, the return to the proper next instruction occurs during I-Fetch, as discussed in the BCR instruction.

BRANCH AND LINK, BALR (05)

- Store PSW(32–63), link information, into GPR (addressed by R1) and branch to location specified by GPR (addressed by R2).
- RR format:



- Conditions at start of execution:
  2nd operand is in A, B, and D.
  3-cycle storage request has been issued per D for branch-to instruction.
  Instruction is in E.
- Link information consists of:
  Instruction length code.
  CC.
  Program mask bits.
  Address of next sequential instruction (link address).
- Link information is stored whether branch is successful or unsuccessful.
- If 'execute' trigger is set, link address is address of instruction following Execute instruction.
- Branch is unsuccessful if R2 = 0.

The Branch and Link, BALR, instruction, which has an RR format with an op code of 05, stores the address of the next sequential instruction. Stored with the address is link information containing the instruction length code, the CC, and the program mask bits. The instruction length code stored will be either 1 or 2. If the instruction length code stored is 2, the BALR instruction is the subject of an Execute instruction. If during a BALR operation the R2 field is equal to zero, the branch is considered unsuccessful.

The purpose of the BALR instruction is to branch to a subroutine and provide a means of returning from the subroutine to the main flow of instructions in a program. How this is accomplished is shown in Figure 3-26. When processing the main instruction flow and a BALR instruction is encountered, the address of the instruction which sequentially follows the BALR in the main instruction flow is stored in LS. For the example illustrated in Figure 3-26, the address of the next sequential instruction is 14 and is stored in GPR9. (If the BALR instruction was in address 14 of the main program, then the address stored would be 16.) Once the instruction address is stored, the branch to the subroutine occurs. The subroutine is performed and, when completed, a branch instruction could be issued using the address that was stored during the BALR instruction as the branch address to return to the main flow of instructions. After returning to the main flow of instructions, the program will continue in its normal manner, processing the remaining instructions.

In determining the address which is to be stored as the link address, IC(21,22) and the 'execute' trigger must be tested (Diagram 5-502, Sheet 2, FEMDM). If IC(21,22) = 11 and the 'execute' trigger is set (indicating the BALR is the subject of the Execute instruction and the Execute instruction is located in the second and third halfwords of its doubleword), 16 is subtracted from the IC and placed into T. Thus, if it is necessary to return to the main flow of instructions, the instruction which will be performed next is that instruction sequentially following the Execute instruction. If IC(21,22) does not equal 11 or the 'execute' trigger is reset, 8 is subtracted from the IC and the value is placed into T.

Unsuccessful Branch

E(12–15), which contains the address of the GPR which has the branch address, is examined. If E(12–15) = 0, branching is not to take place and a No-Operation occurs. If E(12–15) equals anything other than zero, branching occurs unconditionally. First assume that the branch is unsuccessful [E(12–15) equals zero]. IC(21,22) is tested. If IC(21,22) = 11, it indicates that the next instruction to be executed is in R (this instruction is the last halfword of the doubleword in Q) and a new doubleword must be placed into Q. If IC(21,22) equals any other value, the

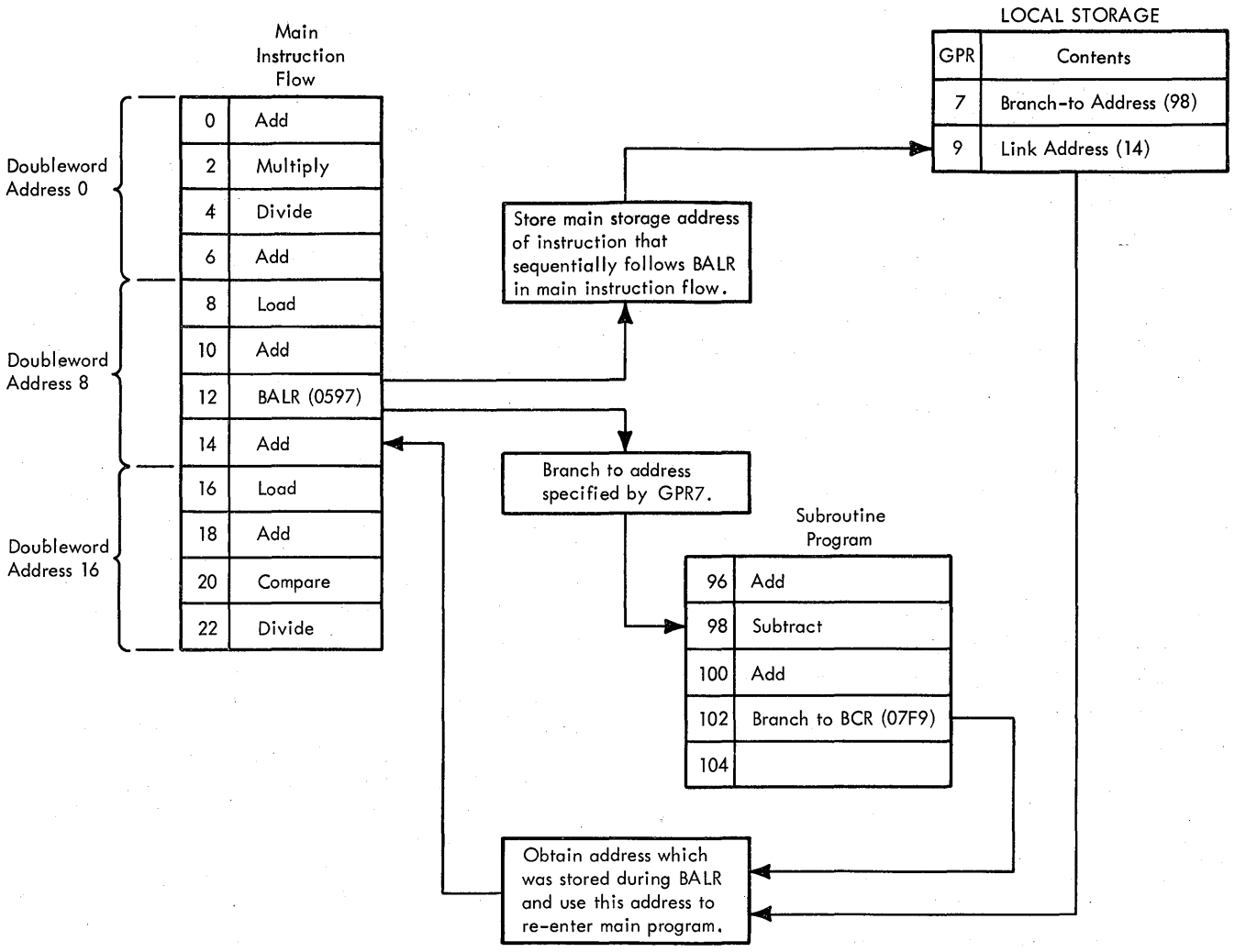


Figure 3-26. Example of Use of Branch and Link Instruction

next sequential instruction is also located in R but Q contains data which is still correct and may be operated on. In either case, the remainder of the link data [PSW(32–39)] is placed into T(32–39) and from there transferred to the GPR per E(8–11). If IC(21,22) equals a value other than 11, a normal end op is taken after storing the data and the next instruction is decoded from R. If IC(21,22) = 11, after storing the link information, a 3-cycle storage request is issued per the IC to obtain the next sequential doubleword to refill Q. The 'execute' trigger is again tested. If set, a normal end op is immediately taken and the next instruction to be executed is the instruction which sequentially follows the Execute instruction (of which the BALR instruction was the subject instruction). (This action is accomplished during the next I-Fetch by means of the ASC microprogram branch which will refetch the instruction following the Execute instruction.) If the 'execute' trigger is reset, the next instruction to be executed is in R. The IC is updated by 8 to select the next sequential doubleword after the one just requested. IC(21,22) is again tested; if it equals 11, the data on the SDBO is gated to Q and a branch end-op cycle is taken.

Successful Branch

Now assume that E(12–15) does not equal zero (indicating a successful branch). The 'PSC' and 'execute' triggers are reset by a combination of the 'T6' and 'M4' micro-orders. PSW(32–39) is transferred to T(32–39). The doubleword containing the branch-to instruction (requested during I-Fetch of the branch instruction) is now present at the SDBO and is gated to Q.

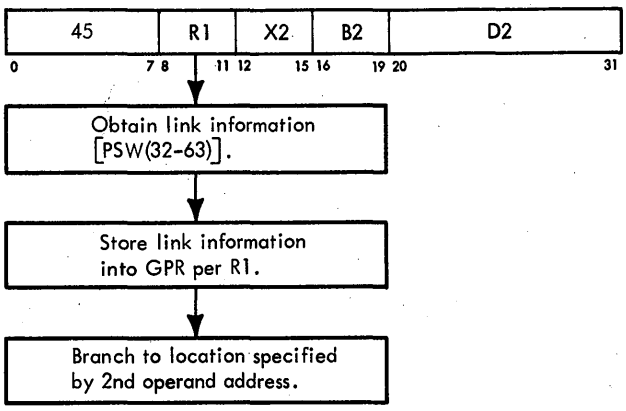
From Q, the correct halfword is transferred to R per D(21,22). Then 8 is added to D, and the result is placed into the IC to address the next sequential doubleword from main storage. The data in T is now transferred to the GPR per E(8–11).

D(21,22) is tested for a value of 11. If it does not equal 11, the data in R and the IC is not yet stable, thus preventing decoding of the next instruction from R or

decoding of the instruction address from the IC. A branch end-op cycle is taken, and the next instruction is decoded off the SDBO which, at this time, is stable. If D(21,22) = 11, the next instruction is located in the last halfword of the doubleword requested during I-Fetch. From Q, the last halfword is transferred to R per D(21,22). Because the halfword that contains the next instruction to be executed is the last halfword of the doubleword, Q must be filled with a new doubleword to allow continuous operation. Then 8 is added to D, and the result is transferred to the IC. A 3-cycle storage request is issued per the IC to obtain the next doubleword. The address of the doubleword is tested for validity; if the address is invalid, the 'I-Fetch invalid address' trigger is set. The IC is updated by 8. By this time, the requested doubleword is present at the SDBO and is gated to Q. A normal end-op cycle is taken, and the next instruction to be executed is decoded off R.

BRANCH AND LINK, BAL (45)

- Store PSW(32–63), link information, into GPR (addressed by R1) and branch to location specified by 2nd operand address.
- RX format:



- Conditions at start of execution:
  Branch address is in D.
  3-cycle storage request has been issued per D for branch-to instruction.
  First 16 bits of instruction are in E.
- Link information consists of:
  Instruction length code.
  CC.
  Program mask bits.
  Address of next sequential instruction (link address).
- Link information is stored whether branch is successful or unsuccessful.
- BAL is unconditional branch.
- If 'execute' trigger is reset and ABC = 0, IC reflects correct address and is stored as link address.
- If 'execute' trigger is reset and ABC does not equal 0, IC is reduced by 8 and then stored as link address.
- If 'execute' trigger is set, link address is address of instruction following Execute instruction.

The Branch and Link, BAL, instruction stores the address of the instruction which, if the branch were unsuccessful, would be the next sequential instruction address. Stored with the address is link information consisting of the instruction length code, the CC, and the program mask bits. The instruction length code stored is 2. The BAL instruction is an unconditional branch with an RX format and an op code of 45.

At the start of execution, the first 16 bits of the instruction are in E, the branch address is in D, and a 3-cycle storage request has been generated per D for the branch-to instruction (Diagram 5-503, FEMDM). At the beginning of the operation, the last three bits of the IC are transferred to the ABC. This value will be used to determine the correct value of the IC before it is stored into LS as link address information. The contents of the IC are transferred to the parallel adder and reduced by 8. This value is then transferred to T, from where it and the remainder of the link data [PSW(32–39)] will be transferred to LS. The 'execute' trigger is then tested; if it is set, the branch operation is the subject instruction of an Execute instruction.

First assume the 'execute' trigger is reset. The ABC is now checked for all zeros. If equal to zero, it indicates that the branch instruction now being executed was located in the third and fourth halfwords of Q. Normally, when an instruction with an RX format occupies the last two halfwords of Q, a storage request is generated during I-Fetch per the IC and the IC is updated by 8. Because this is a branch instruction, however, the storage request from the IC is prevented and the IC is not increased by 8. Therefore, the address presently in T (after being reduced by 8) is incorrect and 8 must be added to it before it is stored into LS.

Assume that the ABC did not equal zero. PSW(32–39) is transferred to T(32–39). Because the ABC was not equal to zero, the link address is correct and can be stored into the GPR per E(8–11). The contents of D are then transferred to the parallel adder, increased by 8, and transferred to the IC. The IC now contains the address of the doubleword in main storage which follows the doubleword containing the branch-to instruction. At this time, a storage request for the next doubleword is issued if D(21,22) = 11. The data requested per D during I-Fetch is at the SDBO and is gated to Q. From Q, the correct halfword is transferred to R per D(21,22).

D(21,22) is now checked for a value of 11. If it is equal to 11, the next instruction to be executed is in the last

halfword of Q. The IC is then updated by 8 to address the next doubleword in main storage. By this time, the doubleword that was requested during the branch instruction is present at the SDBO and can be gated into Q. A normal end-op cycle is then taken to complete the operation. If D(21,22) did not equal 11, a branch end-op cycle is taken and the next instruction is decoded off the SDBO.

Now assume that the 'execute' trigger is set, indicating that the branch instruction is the subject instruction of an Execute instruction. IC(21,22) is tested for a value of 11. If it equals 11, the Execute instruction was located in the second and third halfword of its doubleword and, when in Q, I-Fetch issued a storage request and increased the IC by 8. This increase results in an address in the IC that is 16 bytes higher than the doubleword address containing the Execute instruction. Because the address that is stored as link information is the address of the doubleword containing the instruction immediately following the Execute instruction, the address has to be reduced by 16. The address in T, however, has already been reduced by 8; therefore, only 8 must be subtracted from it. The remainder of the link information is now transferred from PSW(32–39) to T(32–39).

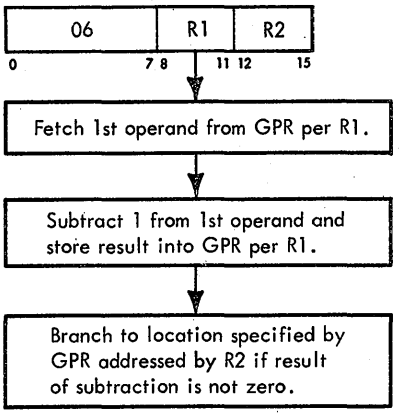
The contents of D are transferred to the parallel adder, updated by 8, and then transferred to the IC to address the next doubleword. The 'execute' trigger is now reset. A storage request for that doubleword whose address was just placed into the IC is issued if D(21,22) = 11. At this time, the data requested during I-Fetch of the branch instruction is present on the SDBO and is gated to Q. The correct halfword in Q is then transferred to R per D(21,22). The link address located in T is transferred to the parallel adder, where it is decreased by 8. This value is now equal to the address of the doubleword that contains the Execute instruction and is transferred to T. From T, the link information is transferred to the GPR per E(8–11).

D(21,22) is now checked for a value of 11. If it equals 11, the next instruction to be executed is in the last halfword of Q. The IC is then updated by 8 to address the next doubleword in main storage. By this time, the doubleword that was requested during the branch instruction is present at the SDBO and can be gated into Q. A normal end-op cycle is then taken to complete the operation. If D(21,22) did not equal 11, a branch end-op cycle is taken and the next instruction is decoded off the SDBO.

Now assume that IC(21,22) did not equal 11. This condition indicates that the Execute instruction was not in the second halfword and a storage request was not automatically generated. The link information presently in T is therefore correct and can be stored into the GPR per E(8–11). The contents of D are increased by 8 and placed into the IC. Again a storage request is generated if D(21,22) = 11. At this time, the doubleword containing the branch-to instruction is located in the SDBO and is gated to Q. The correct halfword in Q is then transferred to R per D(21,22). D(21,22) is tested for a value of 11, and operations continue as previously described, ending with a branch end op or a normal end op.

BRANCH ON COUNT, BCTR (06)

- Subtract 1 from 1st operand (in GPR, per R1) and, if result is not 0, branch to address specified by GPR (addressed by R2).
- RR format:



- Conditions at start of execution:
  Branch address is in D.
  3-cycle storage request has been issued per D for branch-to instruction.
  First operand is in S and T.
  Instruction is in E.

The Branch on Count, BCTR, instruction subtracts 1 from the first operand (contents of the GPR specified by R1) and, if the result does not equal zero or R2 does not equal zero, branches to the address specified by the contents of the GPR designated by R2. The result of the subtraction is stored into the first operand location. If the result of the subtraction equals zero, the next sequential instruction is executed. If E(12–15) = 0, the branch is automatically unsuccessful. The BCTR instruction has an RR format with an op code of 06.

At the start of execution, the instruction is in E, the first operand is in S and T, the branch address is in D, and a 3-cycle storage request has been issued per D for the branch-to instruction (Diagram 5-504, FEMDM). The first operand is transferred from T to B and from B to the parallel adder, where 1 is subtracted from the operand to determine whether the branch is successful. Before subtracting 1, E(12–15) is tested for zeros. As previously stated, if E(12–15) = 0, the branch is unsuccessful. Assume that E(12–15) does not equal zero. The contents

of B are transferred to the parallel adder, where 1 is subtracted from the operand; the result is transferred via T into LS. The result of the subtraction is tested for all zeros; if zero, the branch is unsuccessful; if not zero, the branch is successful.

Successful Branch

First assume that the branch is successful. The 'PSC' and 'execute' triggers are reset. Because D(21,22) indicates in which halfword the branch-to instruction is located, it is examined. If D(21,22) = 11, the instruction is located in the last halfword of the doubleword requested during I-Fetch of the branch instruction. The contents of D, therefore, are updated by 8 and transferred to the IC. By this time, the data requested during I-Fetch is present at the SDBO and can be gated into Q. The last halfword of Q is then transferred to R per D(21,22). A 3-cycle storage request for the next doubleword is now issued per the IC. The contents of the IC are then transferred to the parallel adder, updated by 8, and transferred back to the IC to select the next doubleword from main storage. At this time, the doubleword which sequentially follows the doubleword containing the branch-to instruction is present at the SDBO and is gated into Q. A normal end-op cycle is taken, and the next instruction to be executed is decoded from R.

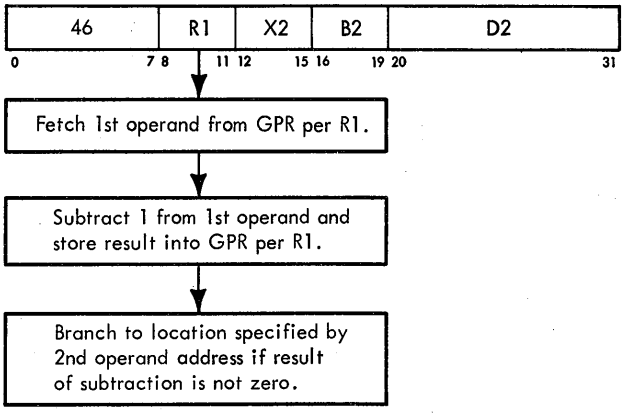
If D(21,22) did not equal 11, the branch-to instruction is located in some halfword other than the last. In this case, the contents of D are transferred to the parallel adder, updated by 8, and then transferred to the IC. At this time, the doubleword containing the branch-to instruction is present at the SDBO and can be gated into Q. From Q, the correct halfword is transferred to R per D(21,22). A branch end-op cycle is taken, and the next instruction is decoded off the SDBO.

Unsuccessful Branch

Now assume that the branch is unsuccessful. If IC(21,22) = 11 or 00, a storage request per the IC must be given during the branch execution phase to obtain the next sequential doubleword from main storage (because this action was inhibited during the end-op cycle by the branch decoder). Once the storage request is issued, the 'execute' trigger is tested. If set, it indicates that the branch instruction is the subject instruction of an Execute instruction. Therefore, a normal end-op cycle is taken to complete the operation. The data requested in this case is not gated into Q. If the 'execute' trigger is reset, IC(21,22) is tested to see whether it contains 11. If the value is 11, the IC is updated by 8 (to select another doubleword) and placed into the IC and D. By this time, the data requested by the storage request given during the execution phase of the branch instruction is present at the SDBO and can be gated into Q. A normal end-op cycle is then taken, and the next instruction to be executed is decoded off R. If IC(21,22) does not equal 11, then it equals 00, and the next instruction is located in the first halfword of the doubleword requested during the execution phase of the branch instruction. The IC is updated by 8, and the result is placed into D and the IC. At this time, the data requested during the execution phase is present at the SDBO. This doubleword is gated to Q; the correct halfword from Q(0–15) is transferred to R. Because the format is normally decoded off R and the data just placed in R is not yet stable, a branch end-op cycle is taken. This cycle allows the next instruction to be decoded off the SDBO, which at this time is stable.

BRANCH ON COUNT, BCT (46)

- Subtract 1 from 1st operand (in GPR, per R1) and, if result is not 0, branch to location specified by 2nd operand address.
- RX format:



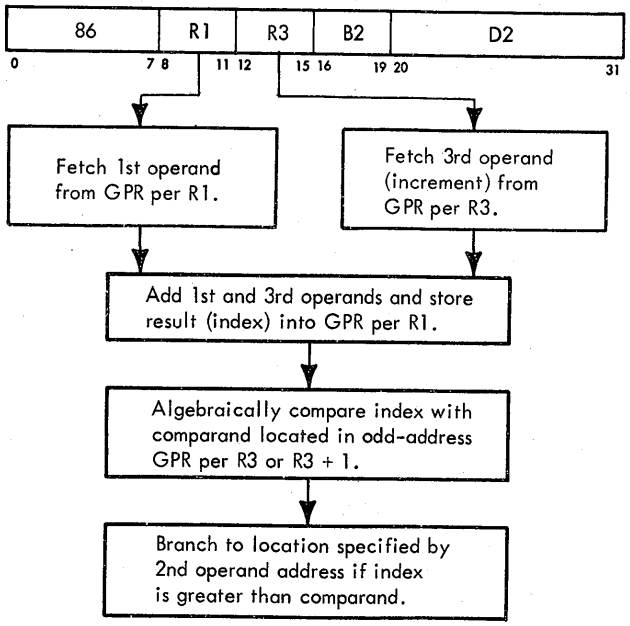
- Conditions at start of execution:
  Branch address is in D.
  3-cycle storage request has been issued per D for branch-to instruction.
  1st operand is in S and T.
  First 16 bits of instruction are in E.

The Branch On Count, BCT, instruction is similar to the BCTR instruction except that E(12–15) is not tested for zero. Refer to Diagram 5-504 for a flowchart of the BCT instruction.

BRANCH ON INDEX HIGH, BXH (86)

- Add increment (3rd operand; in GPR, per R3) to 1st operand (in GPR, per R1), algebraically compare result (index) with comparand (in odd-address GPR specified by R3 or R3 + 1), and, if index is greater than comparand, branch to location specified by 2nd operand address.

- RS format:



- Conditions at start of execution:
  Branch address is in D.
  3-cycle storage request has been issued per D for branch-to instruction.
  1st operand is in S and T.
  First 16 bits of instruction are in E.
- Sum of 1st and 3rd operands is always stored whether branch is successful or not.
- Comparand address (R3 or R3 + 1) must be odd.

The Branch on Index High, BXH, instruction, which has an RS format with an op code of 86:

1. Adds the third operand to the first operand.
2. Stores the result (index) into the GPR addressed by R1.
3. Compares the index with a *comparand* obtained from a GPR addressed by R3 or R3 + 1.
4. Branches if the sum is greater than the comparand.

At the start of execution, the first 16 bits of the instruction are in E, the first operand is in S and T, the branch address is in D, and a 3-cycle storage request has been issued per D for the branch-to instruction (Diagram 5-505, FEMDM). To allow the third operand to be placed into T without destroying the first operand, the first operand is transferred to B. The third operand is then transferred from the GPR per E(12–15) and placed into T. The two operands are added, and the result is transferred to B. The comparand, the value that the sum of the two operands (index) is compared with, is now transferred from LS and placed into T. The contents of B and the 2's-complement of T are transferred to the parallel adder and added to determine whether the branch is successful. IC(21,22) is tested for either 11 or 00. If the branch is unsuccessful, this test sets up conditions to issue a storage request per the IC to obtain the next sequential doubleword after the doubleword containing the branch instruction.

Assume that IC(21,22) = 11 or 00. D(21,22) is now tested to determine where in the doubleword the next instruction is located. If D(21,22) = 11, the next instruction to be executed is contained in the last halfword of the doubleword requested during I-Fetch of the branch instruction. The PAL's and E(7) are now tested (by means of the 'J47 $\neq$ 0' micro-order) to determine whether the branch is successful. First assume the branch is successful; that is, the PAL's are positive (index greater than the comparand) and E(7) = 0. The contents of B (index) are transferred to T, and from there to the GPR per E(8–11). The 'PSC' and 'execute' triggers are reset by means of the 'TIF' micro-order. At this time, the doubleword requested during I-Fetch is present at the SDBO and is gated into Q. The halfword containing the next instruction to be executed is then transferred to R per D(21,22). A test is made to re-establish that D(21,22) = 11. Assuming the original conditions still exist, the contents of D are now updated by 8 and placed into the IC. This action allows the selection of the next doubleword in main storage. A 3-cycle storage request is then issued per the new value in the IC. The IC is then updated by 8 to allow selection of the next doubleword from main storage when needed. By this time, the data requested during the execution phase of the branch instruction is available at the SDBO and can be gated into Q. A normal end-op cycle is then taken to complete the operation. During the end-op cycle, the next instruction executed is decoded off R.

Now assume the branch is unsuccessful. That is, the PAL's are not positive and E(7) = 0. The contents of B (index) are transferred to T, and from there to the GPR per E(8–11). Because the branch is unsuccessful and the contents of R have not been changed, R still contains the instruction that is located in the halfword following the last halfword of the branch instruction. A normal end-op cycle, therefore, can be taken and the instruction decoded off R. If IC(21,22) = 11 or 00, a 3-cycle storage request is issued per the IC, which at this time contains the address of the doubleword that sequentially follows the doubleword containing the branch instruction. The 'execute' trigger is now tested. If set, it indicates that the branch instruction

was the subject instruction of an Execute instruction and a normal end-op cycle is taken, completing the operation. If the 'execute' trigger is reset, IC(21,22) is tested for a value of 11. Recall that, when IC(21,22) was previously tested, it was checked for a value of either 11 or 00. To proceed sequentially in the program without any delay, it is now necessary to determine which value the IC contains. First, assume that IC(21,22) = 11. This value indicates that the next instruction occupies the last halfword of the doubleword in which the branch instruction is located and is presently in R. Recall that R is where format decoding of an instruction occurs. This situation being the case, the IC is updated by 8 to address the doubleword that is 16 bytes from the doubleword address containing the branch instruction. The data that was requested when it was found that the branch was unsuccessful is now present at the SDBO (Diagram 5-505, Sheet 3), and can be gated into Q. A branch end-op cycle is taken, completing the operation. The next instruction is decoded off the SDBO when the data is transferred to Q.

Now assume that IC(21,22) = 00. In this case, the data to be executed is located in the first halfword of the doubleword requested during the execution of the branch when the branch instruction was found to be unsuccessful. The IC is updated by 8. At this time, the data is present at the SDBO and can be gated into Q. The first halfword in Q is transferred to R. Recall that the format for the next instruction is normally decoded off R. Because the next instruction to be executed has just been transferred into R, this data is not yet stable and cannot be decoded. A branch end-op cycle is therefore taken. This cycle allows the next instruction format to be decoded off the SDBO. The SDBO is stable at this time and therefore can be used.

Now assume that when D(21,22) was tested for 11 some other value was found. Again, conditions are tested to see whether the branch is successful (Diagram 5-505, Sheet 2). If successful, the data in B is transferred to T and from there to the GPR per E(8–11). Also, the data that was requested during I-Fetch of the branch instruction is present at the SDBO and can now be gated into Q. From Q, the halfword containing the next instruction is transferred to R per D(21,22). If D(21,22) = 11, the contents of D are now updated by 8 and placed into the IC. This action allows selection of the next doubleword in main storage. A 3-cycle storage request is then issued per the new value in the IC. The IC is then updated by 8 to allow selection of the next doubleword from main storage when needed. By this time, the data requested during the execution phase of the branch instruction is available at the SDBO and can be gated into Q. A normal end-op cycle is then taken to complete the operation. During the end-op cycle, the next instruction executed is decoded off R.

If D(21,22) = 11 and the branch is unsuccessful, the contents of B are transferred to T, from where they are transferred to the GPR per E(8–11). If IC(21,22) = 11 or 00, a 3-cycle storage request is issued per the IC. [If IC(21,22) equals a value other than 11 or 00, a normal end cycle is taken.] This request is for the doubleword that sequentially follows the doubleword in main storage containing the branch instruction. The 'execute' trigger is tested next. From this point on, the operation is the same as that previously discussed for an unsuccessful branch.

Assume, now, that when IC(21,22) was initially tested for either 11 or 00, neither of these values was present (Diagram 5-505, Sheet 2). Again D(21,22), the PAL's, and E(7) are tested to determine whether the operation is a successful branch and where the next instruction is located in the doubleword address being branched to. Assume that D(21,22) = 11 and the branch is successful (Diagram 5-505, Sheet 3). Operation from this point on is identical with a successful branch, as previously described, when D(21,22) = 11.

Now assume that D(21,22) = 11 and that the branch is unsuccessful. The contents of B are transferred to T, from where they are transferred to the GPR per E(8–11). Because the branch is unsuccessful and the contents of R have not been changed, R still contains the instruction that is located in the halfword following the last halfword of the branch instruction. A normal end-op cycle, therefore, can be taken and the instruction decoded off R.

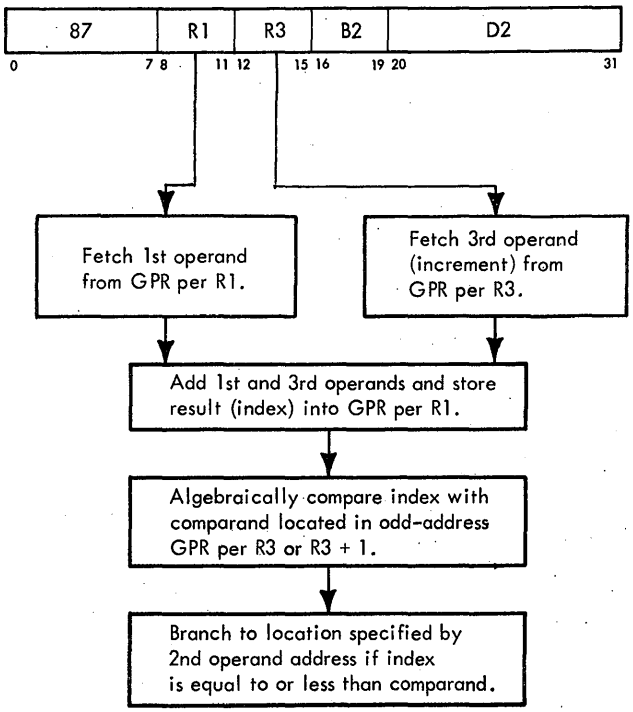
Now assume that D(21,22) did not equal 11 and the branch is successful (Diagram 5-505, Sheet 3). The contents of B are transferred to T, from where they are transferred to the GPR per E(8–11). At this time, the data requested during I-Fetch of the branch instruction is present at the SDBO and is gated to Q. From Q, the correct halfword containing the branch-to instruction is gated into R per D(21,22). The contents of D are then updated by 8 and transferred to the IC to select the next doubleword from main storage. Because the instruction to be executed has just been placed into R and is not yet stable, a branch end-op cycle is taken and the instruction format is decoded off the SDBO.

If the branch is unsuccessful and D(21,22) does not equal 11, the operation is identical with the case where D(21,22) = 11 and the branch is unsuccessful. A normal end-op cycle is taken.

BRANCH ON INDEX LOW OR EQUAL, BXLE (87)

- Add increment (3rd operand; in GPR, per R3) to 1st operand (in GPR, per R1), algebraically compare result (index) with comparand (in odd-address GPR specified by R3 or R3 + 1), and, if index is equal to or is less than comparand, branch to location specified by 2nd operand address.

- RS format:

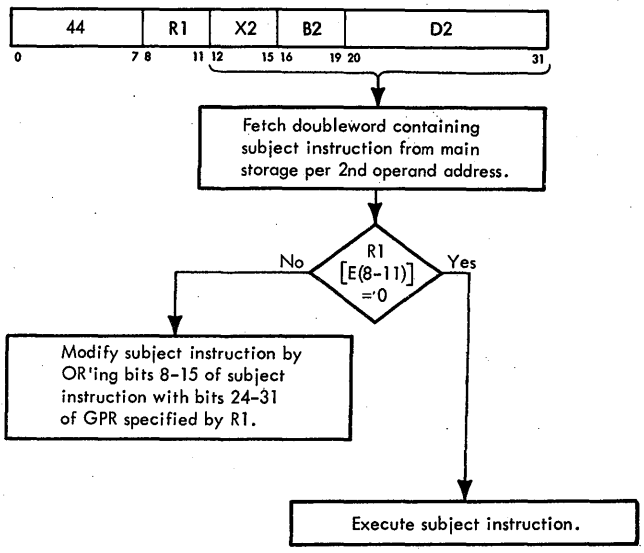


- Conditions at start of execution:
  Branch address is in D.
  3-cycle storage request has been issued per D for branch-to instruction.
  1st operand is in S and T.
  First 16 bits of instruction are in E.
- Sum of 1st and 3rd operands is always stored whether branch is successful or not.
- Comparand address (R3 or R3 + 1) must be odd.

The Branch on Index Low or Equal, BXLE, instruction is similar to the BXH instruction except that branching occurs on a low or equal result (Diagram 5-505).

EXECUTE, EX (44)

- Execute subject instruction at location specified by 2nd operand address. Subject instruction may be modified by 1st operand (in GPR, per R1) if E(8–11) is not equal to zero.
- RX format: (See adjoining column.)
- Conditions at start of execution:
  Address of subject instruction is in D.
  3-cycle storage request for subject instruction has been issued per D.
  1st operand is in S and T.
  First 16 bits of instruction are in E.



- Modification of subject instruction is accomplished by OR'ing bits 8–15 of subject instruction with bits 24–31 of 1st operand.
- If subject instruction is an Execute instruction, a program execute interruption occurs.
- If effective address of Execute instruction is odd, a program specification interruption occurs.
- 'Execute' trigger is set to indicate next instruction is subject of Execute instruction.
- 'PSC' trigger is set to indicate that Q data is not valid and needs to be refilled.
- If program interruptions are pending, normal end-op cycle is taken; if not, branch end-op cycle is taken.

The Execute, EX, instruction, which has an RX format with an op code of 44, executes a designated instruction whose address in main storage is the second operand address. This designated instruction is referred to as a subject instruction and can be modified by the contents of the first operand located in the GPR register specified by R1. Modification of the subject instruction is accomplished by OR'ing bits 8–15 of the subject instruction with bits 24–31 of the first operand. If R1 = 0, no modification takes place. The subject instruction may be 16, 32, or 48 bits long. If the subject instruction is another Execute instruction, a program execute interruption occurs and operation is suppressed. If the effective address of the EX is odd, a program specification interruption occurs.

At the start of execution, the first 16 bits of the instruction are in E, the first operand is in S and T, the address of the subject instruction is in D, and a 3-cycle storage request for the subject instruction has been issued per D (Diagram 5-506, FEMDM). At the beginning of execution, a test for program specification and execute

interruptions is made. If the program specification interruption is present (effective address of Execute instruction is odd), a program interruption occurs and the operation is suppressed. If a program execute interruption is present (the Execute instruction is the subject instruction of an Execute instruction), a program execute interruption occurs and the operation is suppressed. If no interruptions are present, the operation continues. The STC is loaded to 111, allowing the transfer of T(56–63) to the serial adder for modification of the subject instruction if modification is to be accomplished. The contents of D are now transferred to the parallel adder and updated by 8 to address the doubleword that follows the doubleword containing the subject instruction of the Execute instruction. This value is then transferred to D. PAL(61–63) is now transferred to the ABC to select the correct byte for modification of the subject instruction.

D(21,22) is tested to determine whether the subject instruction is contained in the last halfword of the doubleword that was requested during I-Fetch, or in some halfword other than the last. If D(21,22) = 11, the subject instruction is in the last halfword; if any other value, the subject instruction is in some other halfword. First assume that D(21,22) = 11. Because the subject instruction is located in the last halfword of the doubleword addressed during I-Fetch, there is a possibility that part of the instruction is contained in the next doubleword to be addressed. This possibility exists if the subject instruction has a format other than RR. Therefore, the next few operations determine the format of the subject instruction. By doing these tests, an extra request can be prevented, one that can cause an invalid address or protection check if the instruction has an RR format.

At this time, the data requested during I-Fetch is present at the SDBO and can be gated into Q and AB. From Q, the last halfword is transferred to R. The sixth byte in AB is then transferred to the serial adder per the ABC. Minus 64 (1100 0000) is sent to the serial adder, where it is logically AND'ed with the op code of the subject instruction. If the op code denotes an RR format, SAL should now equal zero. 1 is then added to the ABC to transfer the last halfword of AB to the serial adder if the instruction is to be modified.

E(8–11) is now tested to see whether the subject instruction is to be modified. If E(8–11) = 0000, the subject instruction is not to be modified; if any other value, the instruction is to be modified. First assume that E(8–11) = 0000. SAL(0–7) is now tested. Recall that the value in SAL indicates whether the subject instruction has an RR format, and recall that it has already been determined that the subject instruction is contained in the last halfword of Q. Therefore, assume that SAL(0–7) = 0. Because this value indicates that the subject instruction is an RR instruction, there is no need to issue a storage request for the next instruction because there is no information in that doubleword that will affect the operation of the RR instruction. The request for the next doubleword occurs during I-Fetch of the RR instruction.

The last byte in AB is then transferred to T via the serial adder per the ABC and STC. From T, the data is transferred to R. The 'PSC' and 'execute' triggers are set. A test is made for a pending program interruption. If one exists, a program interruption cycle is taken; if there is no interruption, the contents of R are transferred to E and STAT G is set. The setting of STAT G prevents the occurrence of interruptions and the premature prefetching of the next instruction from interfering with the execution of the subject instruction. A branch end-op cycle completes the operation.

Now assume that SAL(0–7) does not equal zero, indicating that the subject instruction has some format other than RR. So that the complete word may be decoded before instruction execution, the doubleword that sequentially follows the subject instruction must be requested. The last byte in AB is transferred to T via the serial adder per the ABC and STC. The 'PSC' trigger is set (insuring return to the main instruction flow upon execution of the subject instruction). This action causes an ASC exceptional condition branch to occur during the I-Fetch following execution of the subject instruction. The ROS microprogram subtracts the correct amount from the IC to select the doubleword containing the instruction following the Execute instruction, and issues a request for that doubleword. A 3-cycle storage request is issued for the next sequential doubleword. The microprogram waits (two storage cycles) until the data requested is present at the SDBO, at which time the data is gated to Q. The 'execute' trigger is set, and a test is made for a pending interruption. If an interruption is present, a program interruption cycle is taken; if not, the data in R is transferred to E and STAT G is set. A branch end-op cycle completes the operation.

If when E(8–11) is tested some value other than 0000 is found, the subject instruction is to be modified. SAL(0–7) is tested to determine whether the subject instruction has an RR format. Assume an RR format. The last byte in AB is then transferred to the serial adder per the ABC. This byte is then OR'ed with the last byte of ST which was transferred to the serial adder per the STC. The results of this OR'ing are then transferred to ST per the STC. The data in T is transferred to R. The 'execute' trigger is set, and the operation continues in the same manner described previously.

Now assume that SAL(0–7) contains some value other than zero. In this case, after the subject instruction is modified, a storage request for the next doubleword must be made. The microprogram waits (two storage cycles) until the data is present at the SDBO, after which the data is transferred to Q. From this point on, the operation is identical with that described for an RR instruction.

Now return to the point where D(21,22) is tested to see whether the subject instruction occupies the last halfword of the doubleword requested during I-Fetch. If D(21,22) equals some value other than 11, a storage request is not issued during execution of the Execute instruction. E(8–11) is now tested to determine whether the subject instruction is to be modified. If E(8–11) does not equal 0000, the instruction must be modified. After the data is placed into Q and AB, and the correct data is placed into the serial adder per the ABC, operation is identical with that of an RR instruction that is to be modified.

If E(8–11) = 0000, however, the subject instruction is not to be modified. Therefore, the correct halfword in Q need only be transferred to R per D(21,22). The 'PSC' and 'execute' triggers are set, and the test for a pending interruption is made. The operation continues in the same manner described previously.

www.IBM360.info

Section 7. Status Switching Instructions

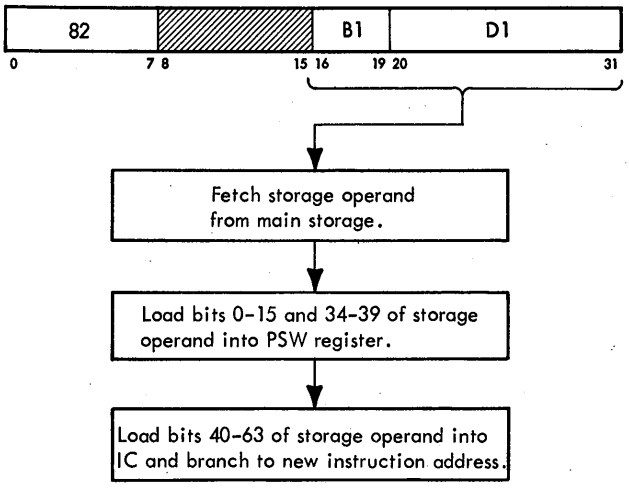
The 10 status switching instructions can change the status of the CPU, the channels, the external units, and the data in main storage:

1. The Load PSW, Set Program Mask, Set System Mask, and Supervisor Call instructions control the status of the CPU.
2. The Set Storage Key, Insert Storage Key, and Test and Set instructions control the status of the data in main storage.
3. The Write Direct and Read Direct instructions control the status of external units (and also transfer data bytes).
4. The Diagnose instruction controls the status of the CPU and channels.

For a discussion of operand addressing, program states, PSW, status switching, instruction formats, data flow, program interruptions, and CC's, see Chapter 1.

LOAD PSW, LPSW (82)

- Load doubleword storage operand (designated by storage operand address) into CPU, thus replacing current PSW, and branch to new instruction sequence.
- SI format:



- Use same microprogram as IPL, PSW RESTART, and interruption operations.
- Conditions at start of execution:
  First 16 bits of instruction are in E.
  Storage operand address is in D.
  Storage request has been issued per D.

The Load PSW, LPSW, instruction loads into the CPU the doubleword from main storage designated by the storage operand address. The doubleword (a new PSW) becomes the current PSW for the next instruction. Bits 40–63 of the doubleword become the address of the next instruction. The new PSW will not allow interruptions until after the LPSW instruction is executed. When the doubleword being loaded has a 1 in position 14 or 15, the CPU enters the Wait state or the Problem state, respectively; this is the only instruction available for entering these states.

Diagram 5-601, FEMDM, is a flowchart of the LPSW instruction. At the start of execution, the first 16 bits of the instruction are in E, the storage operand address is in D, and a storage request for the storage operand has been issued per D.

Tests for a privileged operation check and a specification check are made at the beginning of the execution. The CPU must be in the Supervisor state, and the address of the LPSW instruction must have three low-order 0's; otherwise, a program interruption results.

When the data (new PSW) requested during I-Fetch is available on the SDBO, it is gated to ST. The new PSW is then assembled by transferring S(0–15) and T(34–39) to the PSW register, transferring T(40–63) to the IC, and resetting the interruption request triggers. The ILC remains unchanged until an interruption occurs.

The instruction next makes a 3-cycle storage request per the IC to fill Q with the next instruction. Although the I-Fetch checking circuits have been activated previously, interruptions are inhibited by the setting of the 'I-Fetch request' trigger. The contents of T are now transferred to D so that, when the data is available, D will be able to select the correct halfword in Q to transfer to R. The IC is incremented by 8 to select the next doubleword to be fetched to refill Q. The 'PSC' and 'execute' triggers are reset. These triggers are set if the LPSW instruction is the subject instruction of an Execute instruction. Because the next instruction to be performed is determined by the new PSW being loaded, the triggers must be reset to prevent the instruction following the Execute instruction from being performed after the LPSW instruction.

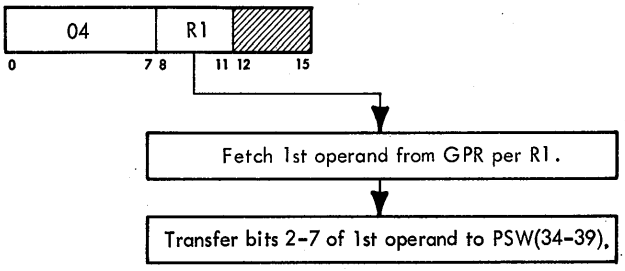
From this point on, the LPSW instruction execution is the same as for the Branch on Count instruction following a

successful branch. When the requested data is available, it is gated to Q, and then transferred to R per D. A decision is now made to refill Q under control of this instruction if D(21,22) = 11. If Q does not need refilling, the instruction terminates with a branch end op. If, however, Q is refilled, a branch end op is not necessary and the instruction terminates with a normal end op.

The LPSW microprogram used by the LPSW instruction is also entered by the interruption, PSW RESTART, and IPL microprograms. The storage operand address is forced to zero by the IPL microprogram and by the depression of the PSW RESTART pushbutton. One of the five new PSW addresses in permanent storage is generated by the interruption microprogram.

SET PROGRAM MASK, SPM (04)

- Replace CC and program mask (bits 34–39) of current PSW with bits 2–7 of 1st operand (in GPR per R1).
- RR format:

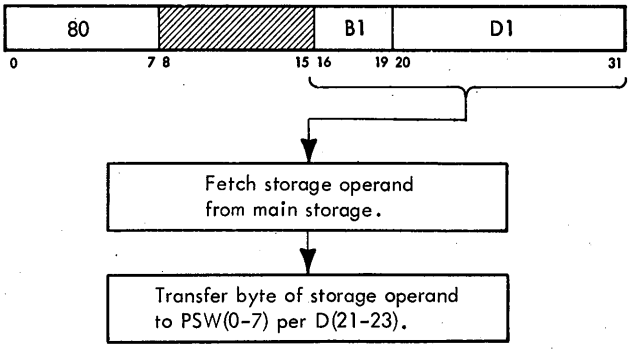


- Conditions at start of execution:
  Instruction is in E.
  1st operand is in A, B, and D.
  2nd operand is not used.

The Set Program Mask, SPM, instruction replaces the CC and the program mask of the current PSW with the contents of the GPR addressed by R1 (Diagram 5-602, FEMDM). Bits 2 and 3 of the GPR become the new CC in the current PSW, and bits 4–7 become the new program mask. Bits 2–7 of the first operand may have been loaded from the PSW register by a previous Branch and Link instruction.

SET SYSTEM MASK, SSM (80)

- Replace system mask (bits 0–7) of current PSW with byte from location designated by storage operand address.
- SI format:

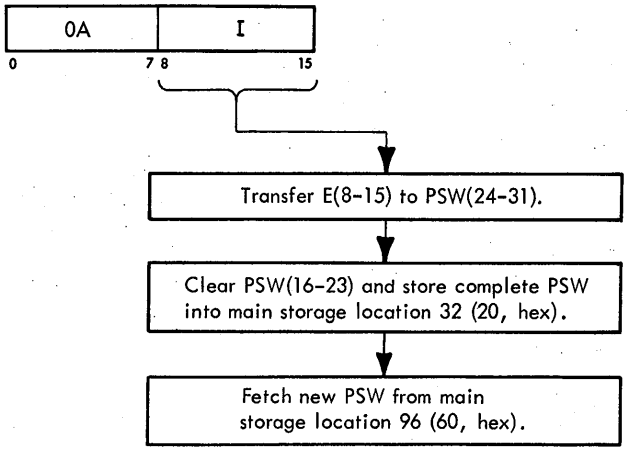


- Conditions at start of execution:
  First 16 bits of instruction are in E.
  Storage operand address is in D.
  Storage request has been issued per D.

The Set System Mask, SSM, instruction replaces the system mask of the current PSW with the addressed storage operand byte (Diagram 5-603, FEMDM). The addressed storage operand byte is fetched from main storage and placed into PSW(0–7) via AB, the serial adder, and S(0–7). The ABC is set per D(21–23) to select the correct byte in AB to gate to the serial adder.

SUPERVISOR CALL, SVC (0A)

- Cause supervisor call interruption; replace old PSW(24–31) with I-field (bits 8–15) of instruction, providing interruption code.
- RR format:

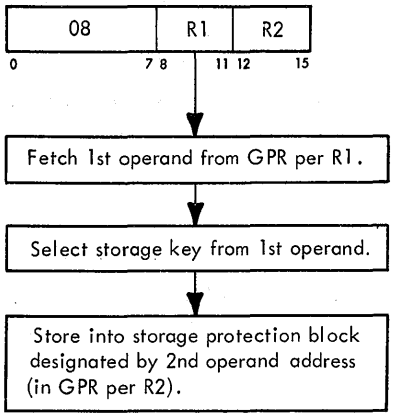


- Conditions at start of execution:
  Instruction is in E.
  E(8–15) is interruption code.

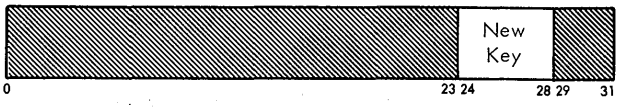
The Supervisor Call, SVC, instruction causes a supervisor call interruption at end-op (Diagram 5-604, FEMDM). The 'supervisor call' trigger is set and, if a timer exceptional condition, a machine check interruption, or a program protection interruption is not pending, the CPU takes a supervisor call interruption. During that operation, bits 8–15 of the instruction, still in E(8–15), are stored as the interruption code into the supervisor call old PSW. The new PSW usually switches the CPU to the Supervisor state. Refer to Section 1 of this Chapter for a discussion of the supervisor call interruption.

SET STORAGE KEY, SSK (08)

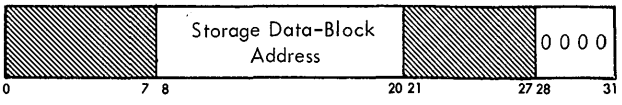
- Set storage key (bits 24–28 of 1st operand; in GPR per R1) for 2048-byte storage block (addressed by bits 8–20 of 2nd operand; in GPR per R2) into storage protection logic in main storage.
- RR format:



- Conditions at start of execution:
  Instruction is in E.
  1st operand is in A, B, and D.
  2nd operand is in S and T.
- Format of LS word addressed by R1:



- Format of LS word addressed by R2:



- New key is set twice because of two-way interleaving.

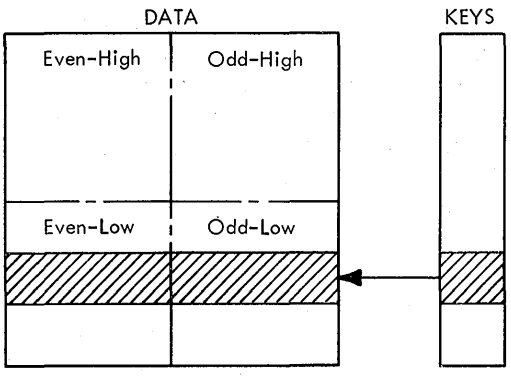
The Set Storage Key, SSK, instruction sets the key of the storage block addressed by the second operand according to the key in the GPR designated by R1. Bits 8–20 of the second operand address a block of 2048 storage bytes. (It is not necessary for the second operand to address the first byte in the block.) During the SSK instruction, bits 21–27 of the second operand, which address doublewords in the storage block, are ignored. Bits 28–31 of the second operand, however, must be 0's, or a program specification interruption occurs. The new storage key is obtained from bits 24–28 of the first operand; the rest of the operand is ignored.

Diagram 5-605, FEMDM, is a flowchart of the SSK instruction. At the start of execution, the instruction is in E, the first operand is in A, B, and D, and the second operand is in S and T. The ABC is set to 7 (111) to allow selection of the new storage key byte from B. The STC is set to 3 (011) to select the low-order byte of the second operand in S, the byte to be tested for a specification check. D is set to the address of the storage block minus 8; this step allows construction of a loop later in the microprogram. If the system is in the Problem state, a program privileged operation interruption occurs. If the system is in the Supervisor state and the address of the storage block does not specify an even doubleword boundary [S(28–31) = 0], a program specification interruption occurs. Otherwise, the execution of the instruction continues by transferring the new key from B to F, via the serial adder.

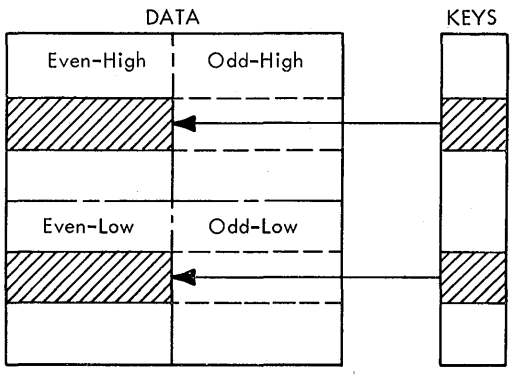
The new key is now in position to be set into the storage unit. D is incremented by 8, and a 4-cycle storage request is issued per the block address in D. The 'set key' trigger is set, causing a 'set key' signal to be sent to main storage with the 'storage select' signal, and all of the mark triggers are set. As a result, the selected storage unit recognizes its selection as being a request to change one of the keys stored in its storage protection area. The new key is gated from F to the 'key in' bus for the use of the selected storage unit.

Execution of this instruction may change the protection status of unprocessed instructions already in the CPU. Therefore, the 'PSC' trigger is set to force a program store compare exceptional condition during the next I-Fetch to refill Q.

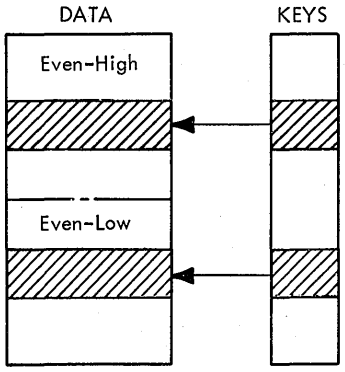
At this point, the setting of one key into main storage has taken place. When operating normally with a 2365 Processor Storage, Model 2, one setting of a new key for 2048 contiguous bytes of data is sufficient, as shown in Figure 3-27(A). However, the SSK instruction in the 2065 always sets the key twice, first for an even doubleword address and then again for the succeeding odd doubleword address. This duplicated setting of the key is done because the System/360 Model 65 has two-way interleaving of



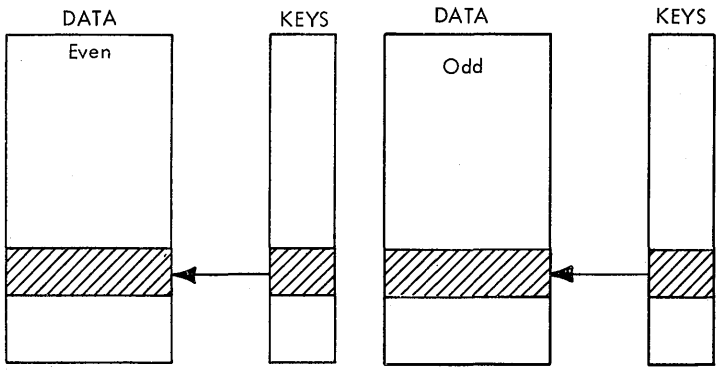
A. 2365-2 Normal Operation.
One Key per 2048-Byte Block.



B. 2365-2 Defeat Interleave Operation.
Two Keys per 2048-Byte Block.



C. 2365-1 Normal Operation.
Two Keys per 2048-Byte Block.



D. 2361 Paired Operation.
Two Keys per 2048-Byte Block.

Figure 3-27. Storage Protection Key Assignments

'storage select' signals based on even and odd storage addresses. There are three cases when contiguous even and odd storage addresses are not in the same storage-protection block in main storage:

1. When the defeat interleave maintenance aid is used, contiguous even and odd storage addresses in the 2365-2 are in two different protected blocks within the same unit, as shown in Figure 3-27(B).
2. When operating normally with a 2365-1, interleaving is always defeated because the 2365-1 has only half the core size of the 2365-2, as shown in Figure 3-27(C).
3. When pairs of 2361 Core Storage (LCS) are attached, even and odd addresses reside in two different storage units, as shown in Figure 3-27(D).

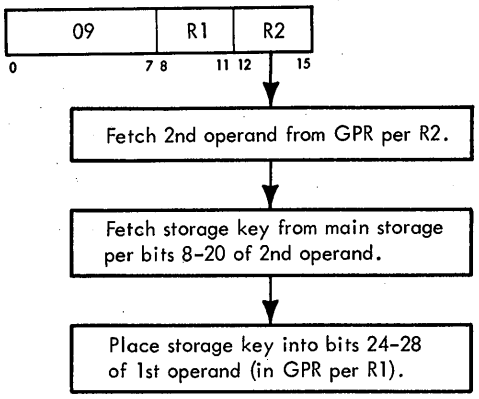
Therefore, after setting the first key, the SSK instruction again increments D by 8 and another 'set key' signal is issued to storage. For the case shown in Figure 3-27(A), the same key location is set again; otherwise, a different key location is set.

Although the System/360 Model 65 always uses two-way interleaving, there is provision in the microprogram for future four-way interleaving which would necessitate setting the key four times. After the key is processed twice, STAT D would be tested. If reset, it would be set, and the new key would be processed twice again into succeeding even and odd storage addresses. However, STAT D now is found set when tested; it is set when the 'set key' trigger is first set and is not reset until after the SSK instruction is completed.

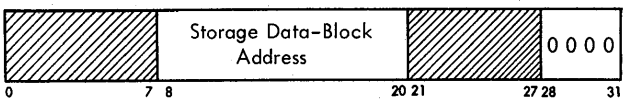
INSERT STORAGE KEY, ISK (09)

- Insert storage protection key for 2048-byte storage block, addressed by bits 8–20 of 2nd operand (in GPR per R2), into bits 24–28 of 1st operand (in GPR per R1).

- RR format:



- Conditions at start of execution:
  Instruction is in E.
  1st operand is in A, B, and D.
  2nd operand is in S and T.
- Format of LS word addressed by R2:



- Storage key is inserted into bits 24–28 of 1st operand.
- Bits 0–23 of 1st operand remain unchanged; bits 29–31 are cleared.
- Key is fetched twice because of two-way interleaving.

The Insert Storage Key, ISK, instruction inserts the storage key addressed by the second operand into bits 24–28 of the first operand (in GPR, per R1). Bits 8–20 of the second operand address a block of 2048 bytes in main storage. Bits 0–7 and 21–27 of the second operand are ignored, whereas bits 28–31 must be 0's or a program specification interruption occurs. The five-bit storage key is set into bits 24–28 of the first operand; bits 29–31 are cleared.

Diagram 5-606, FEMDM, is a flowchart of the ISK instruction. At the start of execution, the instruction is in E, the first operand is in A, B, and D, and the second operand is in S and T. The STC is set to 3 (011) in preparation of setting D to the address of the storage block minus 8, thus allowing the construction of a loop later in the microprogram. The first operand is transferred from B to T via the parallel adder, and the STC is set to 7 (111). The contents of T, with the fetched key inserted into byte 7, will later be transferred back into the GPR designated by R1. Before fetching the key, however, program tests are made. If the system is in the Problem state, a program privileged-operation interruption occurs. If the system is in the Supervisor state and the address of the storage block does not specify an even doubleword boundary [S(28–31) = 0], a program specification interruption occurs. Otherwise, the execution continues. F and the last byte of T are set to 0's. The fetching of the key can now begin.

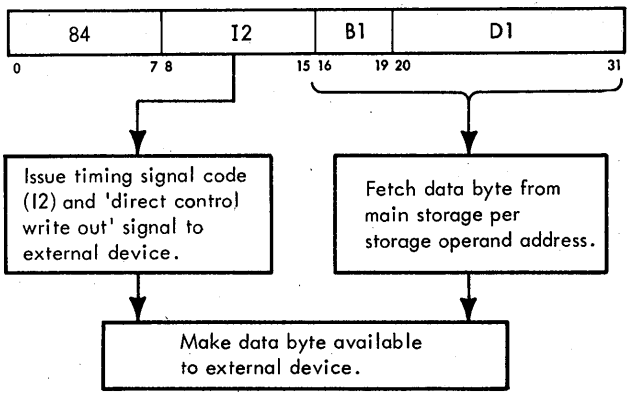
The contents of F are logically OR'ed into the last byte of T via the serial adder, again setting 0's. (This step would be important if the 2065 used four-way interleaving of 'storage select' signals; as with the SSK instruction, the ISK instruction has an unused loop that would result in the fetching of the key four times.) D is incremented by 8 and a 3-cycle storage request is issued per the block address in D. The 'insert key' trigger is set, causing an 'insert key' signal to be sent with the 'storage select' signal. As a result, the selected storage unit recognizes its selection as being a fetch request for one of the keys stored in its storage protection area. The CPU waits until the 'key advance' signal from storage is received and then gates the key from the 'key out' bus into F.

Because the key can be in two different locations, as explained for the Set Storage Key instruction, it is always fetched twice. This scheme ensures that odd addresses can be successfully accessed. After the fetched key is logically OR'ed into T from F, the key for the next doubleword is fetched exactly as before.

When the key is again available, this time from an odd doubleword address, it is logically OR'ed with the key first fetched. The thus modified first operand is then returned to the GPR from which it came. A normal end op completes the execution of the instruction.

WRITE DIRECT, WRD (84)

- Send 'direct control write out' signal and timing signal code (I2 in instruction) to external device for about 0.8usec; make 1 data byte from storage (per storage operand address) available to external device until next WRD is executed.
- SI format:



- Conditions at start of execution:
  First 16 bits of instruction are in E.
  Storage operand address is in D.
  Storage request has been issued per D.
- 8 data bits remain available in G until another WRD instruction is issued.
- 'Direct control write out' signal and timing signal code may be considered instruction for external device.

Note: The operation of the Write Direct instruction is modified if the Multisystem Feature is installed and if the CPU is not in Model 65 Mode. Refer to Chapter 4, Section 2, Multisystem Feature, "Write Direct Instruction (Not Model 65 Mode)".

The Write Direct, WRD, instruction sends the eight bits of the immediate operand plus a 'direct control write out' signal to an external device for about 0.8usec. During that time, the storage operand becomes available to the external device and remains available until the next WRD instruction is executed. No parity is presented with either the immediate or storage operands when sent to the external device.

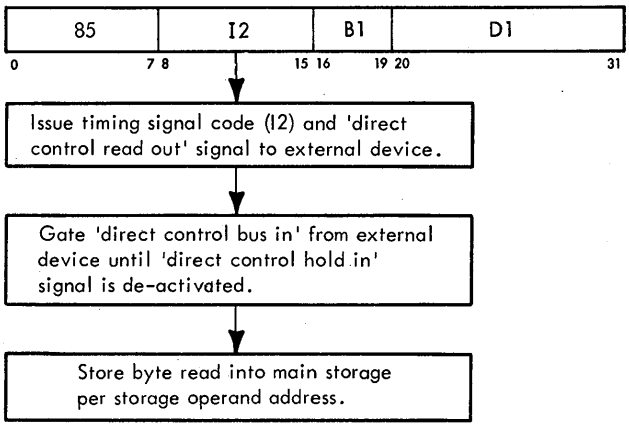
Diagram 5-607, Sheet 1, A, FEMDM, is a flowchart of the WRD instruction. At the start of execution, the first 16 bits of the instruction are in E, the storage operand address is in D, and a storage request for the storage operand has been issued per D. A privileged operation test is first made. If the system is in the Problem state, a program privileged-operation interruption occurs. Also, the validity of the instruction is tested. If the Direct Control Feature is not installed or if the DISABLE DIRECT CONTROL switch is in the active (down) position, the instruction is invalid and a program operation interruption occurs. Otherwise, instruction execution continues.

The 'timing gate' trigger is set for about 0.8usec. During this time, the immediate operand, containing the timing signal code, is sent from E to the external device via the 'timing signal bus out' lines; a 'direct control write out' timing signal is also sent to the external device. These nine timing signals may be considered an instruction for the external device.

While the nine timing signals are being sent, the storage operand is transferred from the SDBO to G via ST and the serial adder. Note that the 'direct control write out' signal is sent when the data in G is being changed. There is no outgating from G to the external device; the contents of G are always available on the 'direct control bus out' lines. G is reset only by the next WRD instruction. After the 'timing gate' trigger is reset, the instruction ends with a normal end op.

READ DIRECT, RDD (85)

- Send 'direct control read out' signal and timing signal code (I2; in instruction) to external device for about 0.6 usec; store 1 data byte from external device into storage (per storage operand address) when 'direct control hold in' signal is absent.
- SI format:



- Conditions at start of execution:
  First 16 bits of instruction are in E.
  Storage operand address is in D.
  Storage request has been issued per D.
- Timing signal code and 'direct control read out' signal may be considered instruction for external device.

Note: The operation of the Read Direct instruction is modified if the Multisystem Feature is installed and if the CPU is not in Model 65 Mode. Refer to Chapter 4, Section 2, Multisystem Feature, "Read Direct Instruction (Not Model 65 Mode)".

During the Read Direct, RDD, instruction, nine timing signals are sent to an external device. Next, a byte of data is read from an external device until the 'direct control hold in' signal is de-activated; the byte of data is then stored into main storage.

Diagram 5-607, Sheet 1, B, FEMDM, is a flowchart of the RDD instruction. At the start of execution, the first 16 bits of the instruction are in E, the storage operand address is in D, and a storage request has been issued per D. A privileged operation test is first made. If the system is in the Problem state, a program privileged-operation interruption occurs. Also, the validity of the instruction is tested. If the Direct Control Feature is not installed or if the DISABLE DIRECT CONTROL switch is in the active (down)

position, the instruction is invalid and a program operation interruption occurs. Otherwise, instruction execution continues. The STC is set per D(21–23), indicating the location in ST that the byte of read-direct data will occupy.

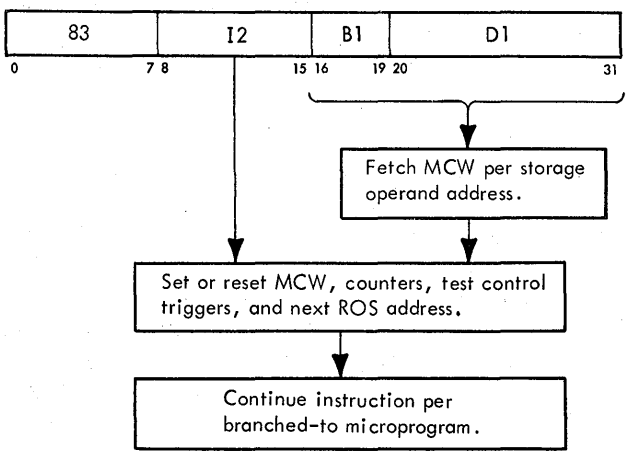
The 'timing gate' trigger is set for about 0.6 usec. During this time, the immediate operand, containing the timing signal code, is sent from E to the external device via the 'timing signal bus out' lines; a 'direct control read out' signal is also sent to the external device. These timing signals may be considered an instruction for the external device. While the 'timing gate' trigger is set, an address store compare test is made; its purpose is to cause Q to be refilled during the next I-Fetch if the byte being changed in main storage is also in Q.

The 'direct control bus in' lines, containing the byte of data to be read from the external device, are gated to F, and a mark trigger is set per the STC. These two actions continuously cycle as long as the 'direct control hold in' signal is being received from the external device. Otherwise, instruction execution continues by storing the contents of F into main storage (via the serial adder, ST per STC, and the SDBI), and finishes with a normal end op.

Neither the 'direct control bus in' lines nor the 'external signal in' lines include a parity bit. However, the serial adder generates odd parity for the data byte before it is stored.

DIAGNOSE (83)

- Load word designated by storage operand address into MCW, set or reset certain control triggers, and branch to ROS address specified by MCW.
- SI format:



- Conditions at start of execution:
  First 16 bits of instruction are in E.
  Storage operand address is in D.
  Storage request has been issued per D.
- Enable and disable various system maintenance aids.
- Available for entering and leaving emulator mode.

The Diagnose instruction has two purposes: it is available to the diagnostic programmer as a maintenance aid and to the system programmer for emulator mode operations. Whichever instruction application is made, the beginning of its execution is basically the same. The immediate operand is used to set or reset control triggers. The storage operand (a doubleword termed the maintenance control word, MCW) is used to set the MCW register, counters, and the address of the next ROS word. The remainder of the execution is determined by the ROS microprogram branched to and the control triggers.

Diagram 5-608, FEMDM, is a flowchart of the Diagnose instruction. For a description of the MCW format, see Section 2 of Chapter 6. For a description of the branches to emulator microprograms, see the applicable Compatibility Feature FETOM.

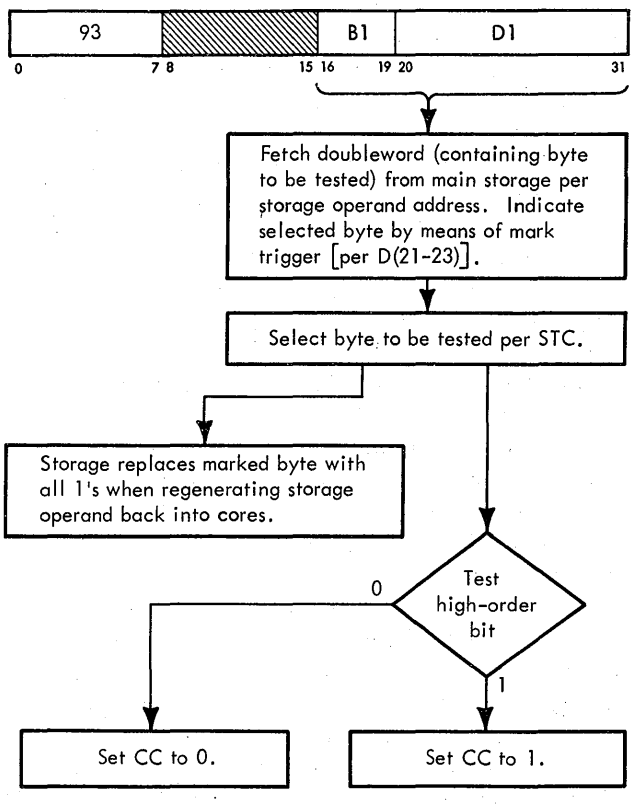
At the start of execution, the first 16 bits of the instruction are in E, the address of the storage operand is in D, and a storage request for the storage operand has been issued per D. If the system is in the Problem state, a program privileged-operation interruption occurs. If the system is in the Supervisor state and the address of the storage operand does not specify a doubleword boundary, a program specification interruption occurs. Otherwise, instruction execution continues by making an address store compare test similar to the test performed during the SS I-Fetch operation. Here it is important for 2067 operations but not for 2065 operations. At this point, the address of the storage operand is no longer needed and is incremented by 8.

The CPU is placed into the scan mode, and the MCW is gated from the SDBO to T (and also to A). The control triggers are now set or reset: T(32–39, 52) is transferred to the MCW register, T(53–57) to the address sequencer, T(58–61) to the FLT counter, T(62, 63) to the FLT clock, E(8, 9) to the two interleave control triggers, and E(10) to the 'diagnose FLT' trigger. The 'scan counter control' and 'diagnose' triggers are set. They remain set depending on the application. The CPU is now taken out of the scan mode and a ROS branch is taken.

The ROS branch address is transferred from T(40–51) to ROSAR(0–11). If the instruction is being used to reset a previously set trigger, the address of a normal end op, 010 (hex), is usually specified and the execution is completed. However, any ROS address may be specified, depending on the application.

TEST AND SET, TS (93)

- Test high-order bit (bit 0) of storage operand byte (in storage), set CC according to state of tested bit, and set addressed byte back into storage as all 1's.
- SI format:



- Conditions at start of execution:
  First 16 bits of instruction are in E.
  Storage operand address is in D.
- CC setting:
  High-order bit = 0: CC = 0.
  High-order bit = 1: CC = 1.

The Test and Set, TS, instruction tests the high-order bit of a single byte in main storage and then sets the byte tested to all 1's. The byte to be tested and set to 1's is specified in the storage operand address. The result of the test of the high-order bit is recorded in the CC: if the high-order bit is a 0, the CC is set to 0; if a 1, the CC is set to 1.

Diagram 5-609, FEMDM, is a flowchart of the execution of the Test and Set instruction. The first 16 bits of the instruction are in E, and the storage operand address is in D. The immediate operand and the storage operand requested during I-Fetch are not used. Instead, a storage request per D is made during execution accompanied by a 'test and set' signal. This action allows no other access to this storage location between the fetching and regeneration of the byte. Before issuing the storage request, a mark trigger is set per D(21–23), via the STC, for use later by the storage unit.

The storage unit performs a unique regeneration operation for the Test and Set instruction. The addressed storage doubleword is fetched and set unaltered onto the SDBO exactly as during a fetch operation. Unlike a normal fetch operation, however, the storage unit uses the mark bit supplied by the CPU to designate the byte to be changed. When regenerating the 72-bit word, the storage unit sets the designated byte in core storage to all 1's. Thus, the storage unit does a combination store and fetch operation. The storage protection facility operates as normal for a store operation.

While generating the request for the storage operand, an address store compare test is started, similar to the test performed during the SS I-Fetch operation. Its purpose is to detect if the storage operand is buffered in Q. If it is, the 'PSC' trigger is set, forcing an exceptional condition during the next I-Fetch, thus refilling Q.

The bit to be tested must be placed into T(32). Therefore, the second operand is gated from the SDBO to AB where the proper byte is selected and transferred to T(32–29) via the serial adder and F. An early end op is taken, overlapping the setting of the CC.

www.IBM360.info

The four I/O instructions are executed by the same ROS sequence (CLD QK021). An I/O instruction may be executed only when the CPU is in the Supervisor state. The first operation of an I/O instruction, therefore, is to determine the state of the CPU. If the CPU is not in the Supervisor state, a privileged-operation check occurs, causing a program privileged-operation interruption. If the CPU is in the Supervisor state, execution of an I/O instruction begins by setting the 'timing gate' trigger (Diagram 3-8, FEMDM). The 'timing gate' trigger gates a 'channel select' signal to select 1 of 7 channels, as determined by bits 16–23 of the operand address (base plus displacement). As long as the 'timing gate' trigger remains set, the CPU waits for a 'release' signal from the channel before initiating the I-Fetch of the next instruction.

At the start of execution, the first 16 bits of the instruction are in E; the first operand is not applicable. The base plus displacement produces a 32-bit result, the high-order 8 bits of which are not significant for storage addressing; hence only the low-order 24 bits are stored into D. Because this is an I/O instruction, the address in D is the address of the channel and I/O unit and is not to be interpreted as a main storage address.

For an I/O instruction, only the low-order 11 bits are significant and valid. Although an 8-bit byte has been reserved for the channel address, D(8–15), only seven channels can be attached to a System/360, Model 65. Hence only D(13–15) is necessary to address all possible channels, and all channel addresses above 6 are considered invalid. Therefore, a 1 in any position of D(8–12) indicates an invalid channel address, the 'channel select' signal is blocked, and the operation is terminated with a condition code of 3.

A properly addressed channel may also be unavailable because the channel:

1. Is physically disconnected from the CPU.
2. Power is down.
3. Meter is disabled.
4. Is in the test mode (the CE AUTO/TEST switch is in the TEST position).

If the addressed channel is unavailable for any of the above reasons, the CPU generates a CC of 3 and a 'release' signal so that the CPU may proceed.

If the channel is available, the I/O unit address, D(16–23), is gated to the unit address register in the channel. (The I/O unit address is ignored by the channel for the Test Channel instruction.)

At this point, the CPU is awaiting a 'release' signal from the channel, and the channel is initiating the operation determined by the I/O instruction. When the channel operation has been initiated, a 'release' signal is generated for the CPU. The 'release' signal allows the microprogram to advance and reset the 'timing gate' trigger, thus dropping the 'channel select' signal. The CPU is now free to initiate the next instruction, while the channel may continue to execute the specified operation. The CPU and the channel may continue to share main storage under control of the priority function of the BCU. Priority is, from highest to lowest, channels 1, 2, 0, 3, 4, 5, 6, and CPU.

When the I/O operation is completed after the CPU is released, the channel initiates an I/O interruption to store the status and condition code. If the CPU accepts the interruption, the channel is freed for further I/O operations initiated by the CPU; otherwise, the channel is unavailable because of the pending I/O interruption.

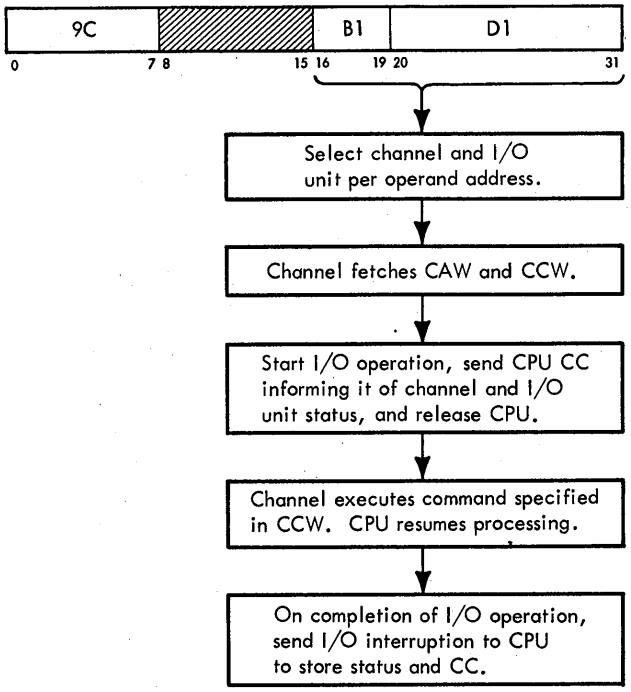
When the 'release' signal enters the CPU, it becomes the 'release CPU' signal. For normal I/O operations, when the 'timing gate' trigger has been set, the ROS microprogram is looping (CLD QK021), and the 'channel select' line is active. The 'release CPU' signal sets the 'release CPU' latch (KX411), the output of which sets ROSAR(11) (DS151). As a result, the microprogram branches and resets the 'timing gate' trigger and proceeds to an end op. The reset 'timing gate' trigger drops the 'channel select' signal and resets the 'release CPU' latch.

When selection of an unavailable channel is attempted, a 'release' signal is not available from the channel. The 'release CPU' latch and a CC of 3 are set by the inactive 'channel available' line. The 'release CPU' latch sets ROSAR(11) as before.

START I/O, SIO (9C)

- Select specified I/O unit and initiate channel command to that unit.

- SI format:



- Conditions at start of execution:
  1st 16 bits of instruction are in E.
  1st operand is not applicable.
  Operand address (address of channel and I/O unit) is in D.
- D(13–15) is channel address.
- D(16–23) is I/O unit address.
- If available, channel fetches CAW, specifying CCW location.
- Channel stores status byte if errors in CAW or unit address.
- CC's specify status of channel and I/O unit.
- CC setting:
  Available: CC = 0.
  CSW stored: CC = 1.
  Working: CC = 2.
  Unavailable: CC = 3.

The Start I/O, SIO, instruction selects a specified I/O unit and initiates a channel command to that unit. The channel commands associated with the SIO instruction are: read, read backward, write, sense, control, and transfer in channel.

Diagram 5-701, FEMDM, is a flowchart of the SIO instruction. At the start of execution, the first 16 bits of the instruction are in E and the operand address is in D; the first operand is not applicable. If the channel is available, the unit address is gated to the unit address register in the channel. The channel then fetches the CAW from main storage address 72 (48, hex). The CAW specifies the address of the first CCW and the storage protection key for all channel commands associated with the SIO instruction.

Two operations, fetching the CCW and selecting the I/O unit per the unit address, are now started simultaneously. Fetching of the CCW is initiated by issuing a storage request from the channel to main storage. When the BCU response is received by the channel, the channel places the command address on the SAB and waits for the 'advance' signal. The 'advance' signal is used to place the command information (command code, data address, flags, and counts) into the proper registers in the channel. The 'CCW valid' trigger is set, if there were no errors, to show that the CCW has been received. The CCW information is then checked for correct parity. During the storage operation, the command address was incremented by 1; the updated quantity is now gated back to the command address register in the channel, and may be used to fetch another CCW if chaining was indicated in the first CCW.

The second operation, selection of the proper I/O unit, is started at the same time as fetching of the CCW. Selecting the proper I/O unit is accomplished by placing the unit address on the bus-out to the control unit and issuing an 'address out' signal followed approximately 400ns later by a 'select out' signal. The control unit responds with an 'operation in' signal, which causes the channel to deactivate the 'address out' signal. When the control unit senses the deactivation of the 'address out' signal, it places the address of the selected I/O unit on the bus-in and activates its 'address in' signal. The channel then compares the address it received from the control unit with the address it sent to the control unit to determine that proper selection has been made.

If the addresses are identical, the 'CCW valid' trigger is set, and the operation continues. The CC is placed on the bus-out, and the 'command out' signal is sent to the control unit. The control unit responds with zero status if it can accept the command. The channel then sends a CC of 0 and a 'release' signal to the CPU. When the CPU receives the 'release' signal and the CC is set, the 'timing gate' trigger is reset, dropping the 'channel select' signal. At this point, the CPU and the channel may resume independent operation.

If any errors occurred up to the point where the channel and control unit compare addresses or if the control unit responded with anything other than zero status, a hardware-generated test-I/O code would be placed on the bus-out to the control unit instead of the command code, and the device status would be cleared. The channel would then disconnect from the I/O unit and request a storage cycle. When the BCU response is received, the status information is stored into the CSW, and a CC of 1 and a

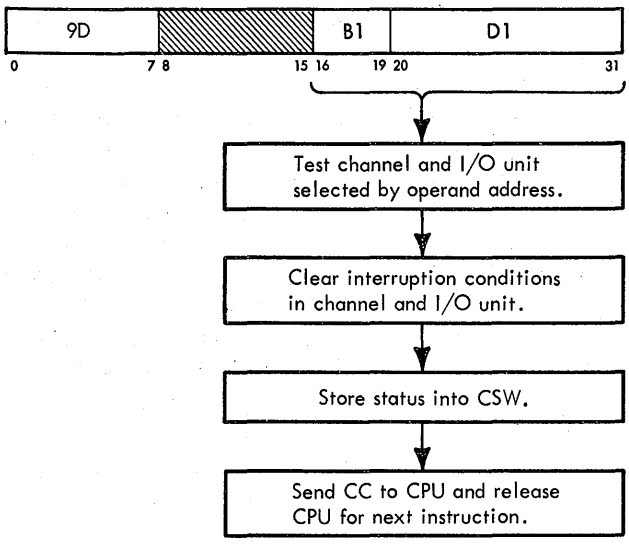
'release' signal are sent to the CPU. This action leaves the channel clear and ready to receive another instruction.

If, when the channel is selected, the selected channel is working or unavailable, a CC of 2 or 3, respectively, is sent to the CPU. A 'release' signal is also sent to the CPU, releasing it for execution of other instructions. If any errors are discovered in the CAW or in the unit address, a status byte is stored into the CSW and a CC of 1 is sent to the CPU. A 'release' signal is also sent to the CPU, releasing it for execution of other instructions.

If, when the control unit and the I/O unit are selected, either the control unit or the I/O unit is unavailable, a CC of 3 is sent to the CPU, together with a 'release' signal, thus releasing the CPU for execution of other instructions.

TEST I/O, TIO (9D)

- Clear interruption condition in addressed channel or associated I/O units, and set CC according to status of addressed channel and I/O units.
- SI format:



- Conditions at start of execution:
  First 16 bits of instruction are in E.
  1st operand is not applicable.
  Operand address (address of channel and I/O unit) is in D.
- D(13–15) is channel address.
- D(16–23) is I/O unit address.
- CC setting:
  Available: CC = 0.
  CSW stored: CC = 1.
  Working: CC = 2.
  Unavailable: CC = 3.

The Test I/O, TIO, instruction clears interruption conditions existing in the addressed channel or addressed I/O unit and stores a CSW in main storage location 64 (40, hex). The CSW is also stored when the channel or I/O unit detects an error during execution of the TIO instruction. The status bits of the CSW identify the error conditions that occurred in the channel or I/O unit. The contents of the CSW pertain to the I/O unit which is addressed by the operand address (base plus displacement) determined during I-Fetch of the TIO instruction.

Diagram 5-702, FEMDM, is a flowchart of the TIO instruction. At the start of execution, the first 16 bits of the instruction are in E and the operand address is in D; the first operand is not applicable.

If the CPU is in the Supervisor state, execution of the TIO instruction begins by setting the 'timing gate' trigger. This trigger sends a 'select' signal to the proper channel as determined by D(13–15). At this time, the unit address [D(16–23)] is sent to the channel. If an interruption is pending, the channel compares this unit address with the unit address it is holding in its unit address register. If they are equal, the interruption condition in the channel is stored into the CSW in main storage location 64 (40, hex), and a CC of 1 is sent to the CPU together with a 'release' signal, releasing the CPU for execution of other instructions. If the addresses are not equal, a CC of 2 is sent to the CPU along with a 'release' signal, allowing the CPU to start processing the next instruction. When the CPU receives the 'release' signal, it resets the 'timing gate' trigger and takes an end-op cycle, completing the operation.

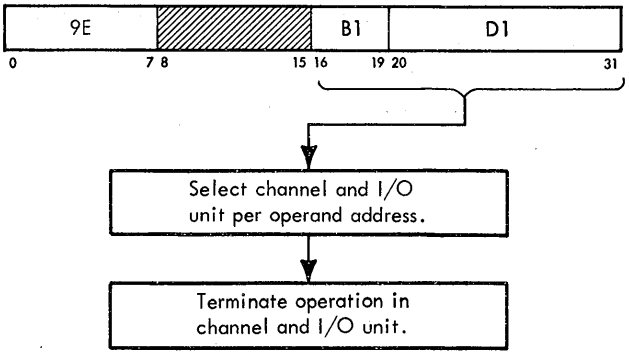
If an interruption is not pending, the unit address sent from the CPU is placed into the unit address register in the channel to select the specified I/O unit. Because this is the TIO instruction, only status is required from the selected I/O unit. If the status of the I/O unit is other than all 0's, a CSW is stored into main storage location 64 (40, hex) and a CC of 1 is sent to the CPU. If the I/O unit status is all 0's, a CC of 0 is sent to the CPU.

A 'release' signal is then sent to the CPU. When the CPU receives the 'release' signal, the 'timing gate' trigger is reset and an end-op cycle is taken, completing the operation. If the status returned to the channel when the test I/O command is issued is all 0's, a CC of 0 is sent to the CPU. This value indicates that the I/O unit is available. The CPU is then released for further instruction execution.

HALT I/O, HIO (9E)

- Terminate current I/O operation at selected channel and I/O unit.

- SI format:



- Conditions at start of execution:
  First 16 bits of instruction are in E.
  1st operand is not applicable.
  Operand address (address of channel and I/O unit) is in D.
- Channel to be halted is determined by D(13–15).
- I/O unit to be halted is determined by D(16–23).
- Channel sends status to CPU via CC and by storing status byte into CSW.
- CC setting:
  Interruption in channel: CC = 0.
  CSW stored: CC = 1.
  Halted: CC = 2.
  Unavailable: CC = 3.

The Halt I/O, HIO, instruction terminates the current I/O operation at the selected channel. The status of the channel is sent, and the status of the I/O unit may be sent, to the CPU by setting the CC and by storing the CSW status byte. The operand address (base plus displacement) determined during I-Fetch of the HIO instruction is used to address the channel and I/O unit.

Diagram 5-703, FEMDM, is a flowchart of the HIO instruction. At the start of execution, the first 16 bits of the instruction are in E and the operand address is in D; the first operand is not applicable. If the CPU is in the Supervisor state, execution of the HIO instruction begins by setting the 'timing gate' trigger. This trigger sends a 'select' signal to the proper channel as determined by D(13–15). At this time, the unit address, D(16–23), is also gated to the channel.

When the specified channel receives the 'channel select' signal, the channel is tested for a pending interruption. If an interruption is pending, a CC of 0 is sent to the CPU together with a 'release' signal. (The interruption is not cleared and remains pending.)

If an interruption is not pending, the channel is tested to determine if it is free (not performing an operation). If the channel is not free ('command out' signal), the I/O unit is disconnected from the channel with an incorrect-length indication, the 'interrupt request' trigger is set, and a CC of 2 is sent to the CPU together with a 'release' signal. (The requested interruption remains pending in the channel.)

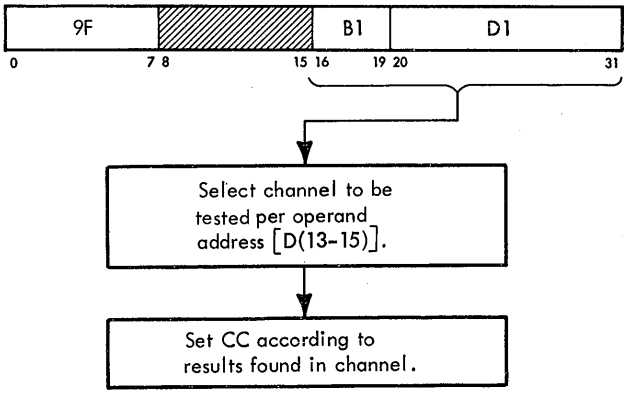
If the channel is free, an attempt is made to select the specified I/O unit. If an I/O unit is not selected (control unit or I/O unit is unavailable), a CC of 3 is sent to the CPU together with a 'release' signal.

If an I/O unit is selected, an address compare test is performed to determine if the correct I/O unit was selected. If the specified I/O unit was selected, it is disconnected from the channel; if some other I/O unit was selected, the 'interface control check' latch is set and an interface reset operation occurs. In either case, an interruption is initiated to store the status bits into the CSW, and a CC of 1 is sent to the CPU together with a 'release' signal.

When a 'release' signal is received by the CPU, the 'timing gate' trigger is reset, dropping the 'channel select' signal. An end-op cycle is taken, completing the operation, and the CPU is free to resume processing.

TEST CHANNEL, TCH (9F)

- Test state of selected channel and set CC accordingly.
- SI format:



- Conditions at start of execution:
  First 16 bits of instruction are in E.
  1st operand is not applicable.
  Operand address (address of channel and I/O unit) is in D.
- Channel to be tested is determined by D(13–15).
- D(16–23), which normally selects I/O unit, is ignored.
- CC setting:
  Available: CC = 0.
  CSW ready: CC = 1.
  Working: CC = 2.
  Unavailable: CC = 3.

The Test Channel, TCH, instruction is used to obtain a CC describing the current state of the channel addressed by D(13–15). The channel state is not affected by the TCH instruction. The appropriate CC is generated and set into the CPU CC register, and the CPU is released to perform further instructions (Diagram 5-704, FEMDM).

If an interruption is pending, a CC of 1 is generated; if an interruption is not pending, the channel is tested to determine if it is working. If the channel is working, a CC of 2 is generated; if not working, a CC of 0 is generated. The generated CC is sent to the CPU, together with a 'release' signal. When the CC is set into the CPU CC register, the 'timing gate' trigger is reset and the CPU takes an end-op cycle, after which it is free to initiate the next instruction.

www.IBM360.info

Section 1. Feature Index

The features available for the 2065 are listed in Table 4-1. Those features discussed in this manual are indicated by a page or chapter and section reference; only the major references are listed. Some of the features are discussed in separate Field Engineering manuals; the title and form number of these manuals are referenced.

Table 4-1. Feature Index

Name and Number	Reference
Additional Storage Attachment 1305 (5th Storage Unit) 1306 (6th Storage Unit) 1307 (7th Storage Unit) 1308 (8th Storage Unit)	Chapter 4, Section 2: Multiprocessing Features
Direct Control 3274	DISABLE DIRECT CONTROL switch: p6-17 Discussion: p1-2 G-register: p2-58 Operation: p3-180, 3-181, 4-4 Power: p5-12
Emergency Power-Off Control 3621 (2 Systems) 3622 (Up to 12 Systems)	Discussion: p5-8
Multisystem 4951 (CPU 1) 4952 (CPU 2)	Chapter 4, Section 2: Multiprocessing Features
1052 Adapter 7920 (1st Adapter) 7921 (2nd Adapter) 7922 (Dual Adapter)	Power: p5-12 1052 Adapter and 2150 Console FETOM: Form Y22-2808
2361 Attachment 8080	Discussion: p2-4, 2-31, 2-37
7070/7074 Compatibility 7117	2065 Processing Unit, 7070/7074 Compatibility Feature FETOM: Form Y27-2106 FEDM: Form Y27-2107
7080 Compatibility 7118	2065 Processing Unit, 7080 Compatibility Feature FETOM: Form Y27-2090 FEDM: Form Y27-2091
709/7040/7044/7090/7094/7094 II Compatibility 7119	2065/2067 Processing Unit, 709/7040/7044/7090/7094/7094 II Compatibility Feature FETOM: Form Y27-2098 FEDM: Form Y27-2099

Section 2. Multiprocessing Features

MULTIPROCESSING SYSTEM/360 MODEL 65

The Multiprocessing System/360 Model 65 (Diagram 7-2, FEMDM) is a particular multiprocessor configuration. It contains two System/360 Model 65's with the Multisystem feature added to each CPU. The Multisystem feature can only be understood when described in the context of the Multiprocessing System/360 Model 65.

The Multiprocessing System/360 Model 65, also called here the multiprocessing system, can operate in one of three ways. Firstly, it can operate as one system, both CPU's controlled by the same supervisor program stored in the completely shared main storage units; because most I/O units are also shared by the CPU's, almost any I/O task can be assigned by the supervisor program to either CPU. Operating two CPU's as resources of one system is termed multiprocessor operation or simply multiprocessing. Secondly, it can be operated as two separate systems, each with its own supervisor program, main storage, CPU, and I/O units. Thirdly, it can be operated with only some of the storage units shared; this case is of limited interest and is not considered here.

The multiprocessing system has a configuration control panel (Diagram 7-11, FEMDM) mounted on the frame section connecting the two CPU's. The following discussion of each of the three major machine areas (main storage, I/O, and CPU):

1. Describes the concepts of multiprocessing.
2. Lists features added to the basic System/360 Model 65's to achieve the multiprocessing system.
3. Describes how the multiprocessing system is tailored by the configuration control panel switches.

Main Storage

In general, a multiprocessing system is characterized by multiple CPU's, each having access to all of main storage. In this fully shared main storage resides a supervisor program that regards the CPU's as a pool of processing resources. The supervisor program can then use the CPU's alternately or simultaneously on one task or separately on two tasks to achieve maximum thruput. Although the supervisor program is necessarily long and complex, it need be placed in only one area of main storage to operate the CPU's; thus, given a fixed number of main storage units and CPU's, more of main storage is available for use by problem programs.

In the Multiprocessing System/360 Model 65, main storage consists of at least two and up to eight 2365-13 Processor Storage units. Each storage unit added after the fourth requires an Additional Wall Section feature and, installed in each CPU, an Additional Storage Attachment feature. Each storage unit has a double BCU-storage interface. Priority circuits in the interface determine which BCU last had access to storage and, in the event of simultaneous requests, give access to the other BCU. This scheme ensures that neither CPU is allowed two successive storage references to a particular storage unit if the other CPU is waiting. Each storage unit has a CE panel allowing off-line maintenance.

Storage Allocation

Each of the main storage units can be allocated to both, either, or none of the CPU's; Storage Allocation switches on the configuration control panel determine which BCU-storage interfaces are enabled. Therefore, the multiprocessing system need not always be operated in a fully shared multiprocessing environment. For example, if multiprocessing is inappropriate for a certain workload, one group of storage units can be allocated to CPU 1 and the remaining storage units to CPU 2. The multiprocessing system can then be operated as two conventional System/360 Model 65's, each CPU run by a different supervisor program.

When allocated to neither CPU, a faulty storage unit can usually be repaired off-line using its CE panel.

Floating Addressing

Because any group of storage units can be allocated to the CPU's, a flexible means of arranging the main storage addresses must be provided. This arrangement is done by Floating Address rotary switches on the configuration control panel, one switch per storage unit. Any of the storage units in a group can contain any of the possible address intervals. This is what is meant by ***floating addressing***.

In round decimal numbers, 262 kilobytes of data can be stored in one 2365-13 Processor Storage, each byte individually addressable; it contains, then, an address interval of 262 kilobytes. The assignment of address intervals to the allocated storage units is therefore in multiples of 262, (i.e., 0 to 262; 262 to 524; and so on). Once the assignments for a group are made, they remain unchanged at least until the next IPL. Exact address ranges are listed in Table 4-2.

Whether the group of storage units is assigned to one CPU or shared by both, the assignment of floating addresses has one restriction: the first address interval of 0 to 262 ***must*** be assigned. For example, assume storage units 1, 3, and 4 have been allocated to CPU 1. The

Table 4-2. Floating Address Intervals

Rotary Switch Label	Decimal	Hex
0 TO 262K	0 to 262,143	0 to 3 FF FF
262K TO 524K	262,144 to 524,287	4 00 00 to 7 FF FF
524K TO 786K	524,288 to 786,431	8 00 00 to B FF FF
786K TO 1048K	786,432 to 1,048,575	C 00 00 to F FF FF
1048K TO 1310K	1,048,576 to 1,310,719	10 00 00 to 13 FF FF
1310K TO 1572K	1,310,720 to 1,572,863	14 00 00 to 17 FF FF
1572K TO 1834K	1,572,864 to 1,835,007	18 00 00 to 1B FF FF
1834K TO 2096K	1,835,008 to 2,097,151	1C 00 00 to 1F FF FF

highest possible address is 786K because 262K X 3 = 786K. Then the contiguous address intervals to be assigned are 0 to 262, 262 to 524, and 524 to 786. They can be assigned to the three storage units in any combination such as storage 3 = 0 to 262, storage 1 = 262 to 524, and storage 4 = 524 to 786. Thus in hex notation, storage 1 contains locations 4 00 00 – 7 FF FF, storage 3 contains locations 0 – 3 FF FF, and storage 4 contains locations 8 00 00 – B FF FF. Presumably, storage 2 has been allocated to CPU 2. In that case, storage 2 also has the address interval of 0 to 262 assigned to it.

Direct Address Relocation

Addresses 0–4095 (decimal) are generated without a base address or index and so are called the ***direct addresses***. Direct address relocation causes CPU and attached-channel accesses to the lowest 4096 data bytes in allocated main storage to be made to the highest 4096 bytes; in addition, CPU accesses to the highest 4096 bytes are made to the lowest 4096 bytes. Only one CPU operating in the fully shared multiprocessing environment requires this ability. The assignment is made by a PREFIX switch on the configuration control panel, one switch per CPU. Refer to Figure 4-1 for the relocated addresses. The highest area is relocated to by inserting a 12-bit prefix into each address which has the high order 12 bits set to zero, SAB(0–11) = 0, and hence pertains to locations 0–4095.

The need for direct address relocation by one of the two CPU's sharing all of main storage is understood when the permanent storage assignment addresses are considered. Main storage bytes 0–383 (decimal) constitute the ***permanent storage area*** of System/360. It is these locations that are referenced by the programs, CPU, and channels for such data as the old and new PSW's, the interval timer value, the CAW, the CSW, the CPU logout, and the channel logout. It is impossible for multiple CPU's to operate normally sharing the same physical locations in main storage for this purpose. For example, old PSW's stored by CPU 1 would be destroyed by old PSW's stored by CPU 2; CPU 1, therefore, would re-enter the interrupted programs of CPU 2. Clearly, each CPU requires a separate permanent storage area. When operating separately with unshared storage allocated to each CPU, no conflict results. When operating together in the fully shared multiprocessing environment, however, the conflict would result if not for direct-address-relocation logic

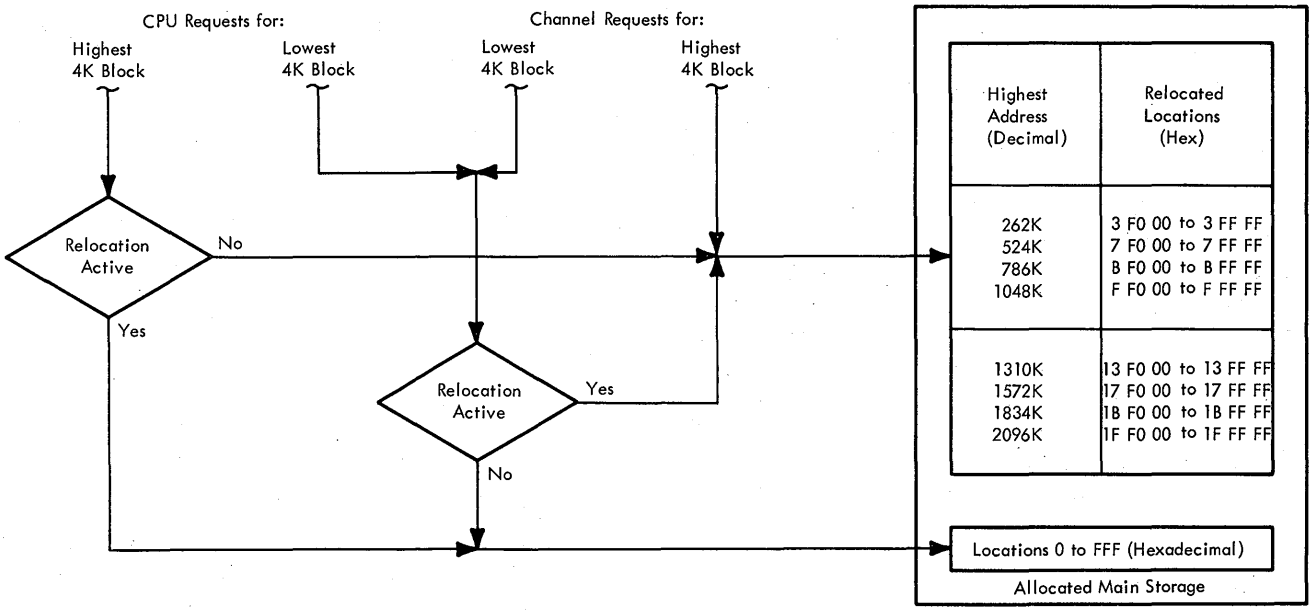


Figure 4-1. Direct Address Relocation

added to the BCU by the Multisystem feature. Just as the CPU uses main storage addresses up to 383 (decimal) for its operations, the supervisor program uses up to address 4095, generally, for its operations. The tables, pointers, and control blocks associated with only one CPU are stored in that area or conflicts result. Therefore, the first 4096 bytes (0 to FFF, hex) are relocated.

Notice in Figure 4-1 that both permanent storage areas can be made accessible to both CPU's. This function is important for the supervisor program and error recovery routines. Notice also that the channels attached to the *relocating* CPU do not have access to the permanent storage area of the other CPU. This hardware-imposed restriction is inconsequential because the channels never need access to the other CPU's permanent storage area.

Input/Output

Not every I/O device in a multiprocessing system is connected to both CPU's. The 1052 Printer-Keyboard is an example. However, most of the devices are connected to both CPU's. The path of connection is from separate channels, through a double channel-control unit interface, to the control unit of the device. The double channel-control unit interface might be a separate standalone unit or a special feature (Two-Channel Switch feature) of the control unit, depending on the device. In either case, the Remote Switch Attachment feature is added to this two-channel switch interface placing I/O Allocation switches on the configuration control panel, one switch associated with each interface. When operating as a multiprocessing system, all the I/O Allocation switches are enabled. The supervisor program can then issue I/O commands from either CPU to most of the I/O devices. If one CPU is busy, the other can perform the I/O instruction. The CPU that starts an I/O operation is generally the CPU that is interrupted by the I/O when the operation finishes. The two-channel switch interface in the selected path directs the interruption request to the correct CPU.

When operating the two CPU's independently, the I/O devices are usually allocated to only one of the two CPU's. Remember that with or without the Multisystem feature, a System/360 Model 65 can operate as part of a multisystem by a shared-control-unit connection. Other I/O multisystem connections can exist through Channel-to-Channel Adapter feature, shared-device, or communication-line connections. The channels, control units, and devices (and their features) attachable to a basic System/360 Model 65 are attachable to a Multiprocessing System/360 Model 65.

Processing Units

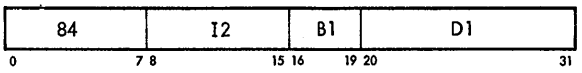
In some multiprocessing systems, one of the CPU's is designed primarily, or even solely, for I/O processing, thus allowing economies in that CPU's capabilities. For instance, a floating-point instruction set is not needed in that CPU. However, when this assignment is done in a multiprocessing system of two CPU's, versatility is lost. In the Multiprocessing System/360 Model 65, the CPU's have equal capabilities. If both CPU's are busy processing, the next task – I/O, logical, or arithmetic – can be processed by the first CPU to become idle.

There are two three-position rotary switches on the configuration control panel, one per CPU. These rotary switches control the mode of system operation of the two CPU's. When multiprocessing together (duplexed), both rotary switches are placed in the MS position (Multisystem mode), enabling use of six special-purpose, inter-CPU signals, the multisystem signals. Also only in this mode, the Set System Mask instruction is modified. When operating independently (simplexed), the CPU is placed in either Model 65 (65 position) or Partition (PTN position) mode. Operation of the Direct Control feature (a prerequisite for installation of the Multisystem feature) is modified in Multisystem and Partition modes. The mode of operation does not affect storage allocation, floating addressing, direct address relocation or I/O allocation.

One Emergency Power-Off Control feature is required in the system.

Multisystem Mode

When multiprocessing in a fully shared environment, both CPU's are manually placed into Multisystem mode. To enhance multiprocessor operations, five multisystem signals are enabled between the CPU's in this mode. One of the signals, 'malfunction alert', is machine-generated. Another, 'gated load pb', is manually generated when LOAD is depressed. The remaining four signals are issued by execution of the Write Direct instruction as shown below:



I2 Field	Multisystem Signal
Bits 8 and 9 = 01	'system reset'
Bits 8 and 9 = 10	'external start'
Bits 8 and 9 = 11	'log I/O interrupt'
Bit 10 = 1	(reserved)
Bit 11 = 1	'system call'

The Write Direct instruction is performed and the multisystem signals are issued regardless of the position of the DISABLE DIRECT CONTROL switch on the CPU system control panel. 'System reset', 'external start', and 'log I/O interrupt' signals are recognized by the receiving

CPU even if its direct control is disabled. Only the 'system call' and 'malfunction alert' signals can cause an external interruption of the receiving CPU.

Direct control might be interconnected between the CPU's or might be connected to external devices, depending on the system configuration. This can be done because the multisystem signals are separate from the direct control interface. However, direct control operations are modified when operating in Multisystem mode. Because the four high-order bits of the I2 field are assigned for special functions, their use in the Read Direct instruction, as well as external signals 0–3, are ignored. Use of bits 8–11 in the Write Direct instruction is restricted to the multiprocessing functions. Also, although direct control operations are still controlled by the DISABLE DIRECT CONTROL switch, no operation program interruption occurs if the switch is in the active (depressed) position. All other direct control operations, including external signals 4–7, are unchanged.

The Set System Mask instruction is modified in Multisystem mode only. The current system mask bits can not be changed by execution of the Set System Mask instruction when operating in Multisystem mode. If operating in the Supervisor state and Multisystem mode, the Set System Mask instruction is suppressed and causes a multisystem program interruption. The interruption code of 12 (hex) stored in the old PSW is used by the supervisor program to identify this source of interruption.

Model 65 Mode

Usually, both CPU's are switched to Model 65 mode when independent operation is desired. The Model 65 mode allows each 2065 to function as if the Multisystem features were not installed. Storage allocation, floating addressing, direct address relocation, and I/O allocation are active regardless of the operating mode. However, the multisystem signals and the multisystem program interruption are not needed when a non-multiprocessing supervisor program is operating one CPU. Therefore, the Direct Control feature and the Set System Mask instruction are as described for a System/360 Model 65 without the Multisystem feature.

The supervisor programs applicable to the conventional System/360 Model 65 are run in Model 65 mode. The emulator program for a Compatibility feature, for example, is run in Model 65 mode. (Any of the three Compatibility features might be installed in the two 2065's, depending on the system requirements.) Before initially loading the emulator program, a simplex system must be manually selected from the multiprocessing system. This selection is done by switching the CPU with the desired Compatibility feature to Model 65 mode and allocating to it the required storage and I/O units; the Floating Address switches are dialed so that storage addressing is continuous from location 0. Direct address relocation is not enabled because the IBM-supplied emulator program cannot be shared by the other CPU.

Partition Mode

Partition mode allows a multiprocessing supervisor program to run with one CPU if the other CPU in the multiprocessing system is unavailable. The other CPU may be unavailable to the multiprocessing program because of a changing workload or because of a machine malfunction. The configuration control panel is used to partition a simplex system from the multiprocessing system to perform other work or system maintenance.

Multiprocessing is simplified in Partition mode. Because the multiprocessing program handles I/O interruptions from only one CPU when running in Partition mode, the Set System Mask instruction operates as described for the 2065 without the Multisystem feature. Direct control operations in Partition mode, however, remain modified as in Multisystem mode; the multiprocessing program can then perform direct control operations the same way in both modes. Because the CPU is operating independently, the multisystem signals are neither issued nor recognized by a CPU in Partition mode.

Multisystem Signals

The six multisystem signals are issued and recognized only in Multisystem mode. Therefore, they are meaningful only if both CPU's are in Multisystem mode:

1. Malfunction Alert. The 'malfunction alert' signal is issued when CPU power is turned off or when a CPU machine check requires a logout operation. (As in a 2065 without the Multisystem feature, a CPU machine check requires a logout followed by a machine check interruption if the CPU CHECK switch is in the PROC position and the machine check mask bit is set to 1.) The receiving CPU is externally interrupted if the external mask bit is set to 1; bit 26 is set to 1 in the external-interruption old PSW. Thus, even if the malfunction of the sending CPU prevents it from performing a reliable logout and machine check interruption, the supervisor program can be notified via the external interruption of the receiving CPU that a malfunction has occurred. An additional CPU machine check condition can be detected in Multisystem mode to enhance the programmed recovery from "hang"-type malfunctions; a system hang machine check is caused if the enabled interval timer is not updated within eight cycles of the line frequency (116.7 to 133.3 ms in 60-Hz installations, 140 to 160 ms in 50-Hz installations). A multisystem timer, added by the Multisystem feature, counts the line-frequency cycles. The 'malfunction alert' signal remains pending in the receiving CPU if the external mask bit is set to 0.

2. Gated Load Pushbutton. The 'gated load pb' signal is issued when LOAD is depressed. All the units available to the receiving CPU, as well as the receiving CPU, are immediately reset: storage units, channels, control units, and devices. After being reset, the receiving CPU is placed in manual mode.
3. System Call. The 'system call' signal is issued by the Write Direct instruction if bit 11 of the I2 field is set to 1. Like the 'malfunction alert' signal, the 'system call' signal causes an external interruption of the receiving CPU if its external mask is set to 1. Bit 27 is set in the external-interruption old PSW. Unless direct control is interconnected between the CPU's, this is the only programmable means of externally interrupting the receiving CPU. The 'system call' signal remains pending in the receiving CPU if the external mask bit is set to 0.

The remaining three signals are mutually exclusive for any one Write Direct instruction because bits 8 and 9 of the I2 field can be coded for only one at a time. The signals do not externally interrupt the receiving CPU. Rather, they cause specific operations used by the multiprocessing program for recovery from program and machine malfunctions. These operations cannot be disabled by PSW mask bits or by the DISABLE DIRECT CONTROL switch.

4. Log I/O Interrupt. The 'log I/O interrupt' signal is issued by the Write Direct instruction if bits 8 and 9 of the I2 field are set to 11. If an I/O interruption is pending in the receiving CPU, the signal causes the CSW and interruption code of the waiting channel to be stored, regardless of its system mask bit. The multiprocessing program can thus recover I/O interruptions left pending at a malfunctioning CPU. Because the status of the I/O operation is not lost, the I/O operation does not have to be repeated.

   The microprogram in process when the 'log I/O interrupt' signal is received is immediately discontinued and the receiving CPU is forced to the manual mode. The operation of the system remains consistent if the CPU is in the Wait state and has all channels masked off when the 'log I/O interrupt' signal is received. If multiple I/O interruptions are pending, only the channel with highest priority is serviced. CPU machine checks are caused and ignored by the log I/O interrupt operation. After log I/O interrupt operations, the receiving CPU must be reset and restarted to continue normal processing.
5. System Reset. The 'system reset' signal is issued by the Write Direct instruction if bits 8 and 9 of the I2 field are set to 01. All the units available to the receiving CPU, as well as the receiving CPU, are reset: storage units, channels, control units, and devices. After being reset, the receiving CPU is placed in manual mode. Unlike the actions started by the 'gated load pb' signal, the system reset operation is delayed for two counts of the line frequency, allowing any storage of data in progress to finish. The multisystem timer is used for this counting function also, regardless of the position of the DISABLE INTERVAL TIMER switch.
6. External Start. The 'external start' signal is issued by the Write Direct instruction if bits 8 and 9 of the I2 field are set to 10. The receiving CPU is started (or restarted) by the 'external start' signal as if the PSW RESTART pushbutton were depressed. That is, the reception of the 'external start' signal causes processing to resume per the PSW in main storage address 0. The external-start operation begins immediately if the receiving CPU is in the Wait or Stopped state; otherwise it begins at the next end op. (If an external system-reset operation is in progress, the external-start operation begins when the reset is finished.) No system reset is performed, no I/O operation is initiated, and no old PSW is stored during the external-start operation.

Summary of Multiprocessing System Advantages

The key advantages of the multiprocessing system may be summarized as follows:

Increased Utilization: In installations having two independent CPU's, certain jobs possibly cannot be processed on one system because sufficient main storage or I/O devices are lacking. The other system, at the same time, may have main storage or I/O devices not being used. Resource sharing by the multiprocessing system minimizes these shortcomings.

Increased Availability: Resource sharing leads to another advantage. The multiprocessing system can continue to process in the event of most single machine malfunctions, even CPU malfunctions. This increases system availability. Concurrent repair of the faulty unit is possible in most cases.

Simplified Management: In an installation with two independent systems, manual balancing of the daily workload between two job queues is needed. Also, the installation data base might be split between the two systems, requiring duplication of records. The multiprocessing system alleviates these two problems when operated as a single system. The need for manually balancing the workload is minimized, and efficient operation is possible for applications needing large online data bases and higher file access rates.

Flexibility: The multiprocessing system can also be reconfigured into two independent systems when required by the workload.

FUNCTIONAL UNITS

Two functional units, the configuration control panel and the multisystem timer, are added; one functional unit, the BCU, is modified by the Multisystem feature.

Configuration Control Panel

The configuration control panel (Diagram 7-11, FEMDM) is mounted on the frame section connecting the two CPU's. The manual switches on this panel, the 'system X available' latches in the storage units, and the 'enable prefix' and 'multisystem mode' triggers in the CPU control the operation of the multiprocessing system. These controls are shown simplified in Diagram 7-3, FEMDM, and are described in the following paragraphs.

Storage Allocation Control

There is one double-pole, two-position toggle switch for each CPU-storage interface. When in the enabled position, CPU dc ground through one pole and -3V dc through the other pole are connected to the floating address switch of the same storage unit; this gates the setting of the floating address switch to the CPU to which that storage unit is allocated. The enabled dc ground is also sent to the designated CPU-storage interface in the storage unit, setting the 'system X available' latch when a 'not clock out' level is received from the same CPU. A 'power on storage frame X' level is then returned to the CPU if the storage unit is ready; that is, if it has power on and is not in test mode. The 'system X available' latch is reset when the CPU or storage unit power is turned off or when the 'not clock out' level from the CPU detects the absence of the enabled dc ground. This interlocking prevents selecting an unavailable storage unit and manually disabling a busy CPU-storage interface.

Floating Address Control

- One switch per storage unit.
- Logical address of each storage unit is encoded into three binary bits.
- 'Floating prefix bit' signals are encoded from allocated Floating Address switch set in highest position.
- If set storage addresses allocated to one CPU are duplicated or are higher than 1048K (without Additional Storage Attachment feature installed), then storage address decoding is blocked and VALID ADDRESS indicator is not lit.

There is one seven-deck, eight-position rotary switch for each storage unit. Decks A, B, and C of each switch encode the position into a three-bit binary value. For example, position 1 = 000, position 2 = 001, position 3 = 010, and so on. Both CPU's receive the encoded value regardless of the storage unit's allocation.

Deck E is used by CPU 1 (deck D by CPU 2) for encoding a three-bit *floating prefix*. The 'floating prefix bit' signals define the storage unit containing the highest 4K block of storage. They correspond to SAB(3–5) and are important during direct address relocation operations. The highest address range set into the allocated Floating Address switches is encoded into a three-bit binary code.

Deck G is used by CPU 1 (deck F by CPU 2) to detect when more than one of the storage units allocated to CPU 1 are assigned the same interval of addresses. The CPU detecting that conflict is blocked from decoding the storage address until the Floating Address switches are correctly positioned. Storage address decoding is also disabled when the rotary switch for any allocated storage unit is assigned an address interval above 1048K (without an Additional Storage Attachment feature installed). If the floating addresses of the allocated storage units are not duplicated and within the correct storage bounds, the VALID ADDRESS indicator lights.

The scheme for detecting duplicate addressing is shown in Figure 4-2, using CPU 1, storage 1, and storage 2 as examples. When the input to the receiver is greater than 1.9V dc, as when no duplicate addresses are assigned, a positive logic level is recognized and inverted to a negative logic level. However, both storage units are shown assigned the address interval of 000 to 262K. Because both storage units are allocated to CPU 1, two 270-ohm resistors are in parallel, lowering the resistance in series with the 200-ohm resistor. This action, in turn, lowers the receiver's input voltage to near ground potential (logical 0 level) instead of +2.4V dc (logical 1 level). Because the input to the receiver is now less than 1.0V dc, a negative logic level is recognized and inverted to a positive voltage level. A positive output voltage from the receiver, therefore, is indicative of a duplicate address assignment. Any further duplication of addresses lowers the input voltage even more; the receiver's output, therefore, remains at the positive logic level.

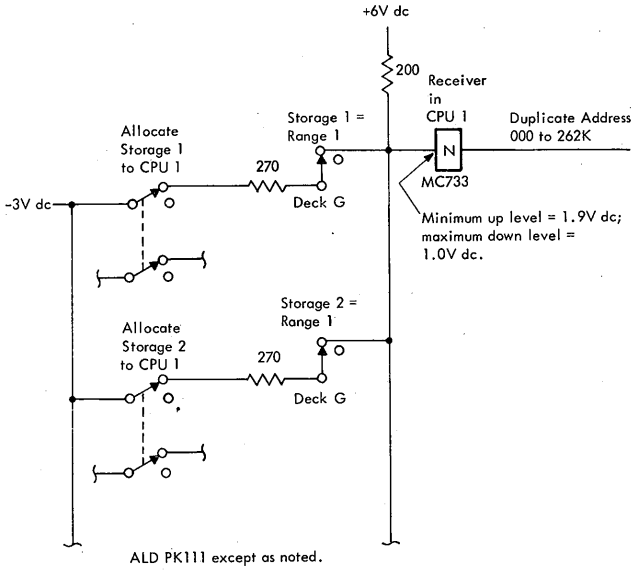


Figure 4-2. Duplicate Storage Addressing Detection

Direct Address Relocation Control

The relocation of direct addresses (prefixing) may be enabled manually for either or both CPU's, or may be disabled for both CPU's. There is a two-position toggle switch, labelled PREFIX, for each CPU. The position of the switch is sampled when initially loading programs into the system; that is, when the LOAD pushbutton is depressed or when an external restart operation is performed. The switch position is also sampled when the CPU's power is turned on. If the PREFIX switch is in the ENABLE position, the 'enable prefix' trigger for that CPU is set. Otherwise, the CPU dc ground is returned to the CPU through the PREFIX switch and loading resets the 'enable prefix' trigger. Only when the 'enable prefix' trigger is set will storage requests for direct addresses be relocated.

Multiprocessing System Mode Control

The operating modes of the CPU's are separately controlled; either CPU can be placed in Multisystem, Partition, or Model 65 mode by its own Mode switch. Both of these switches are three-position, normally closed rotary switches, with the common terminal connected to the dc ground of the associate CPU. The Multisystem mode is entered and left only when the CPU's usage meter is stopped, such as when in the Stopped state. For this reason, the 'CPU clock stopped' signal samples the level returned from the MS position of the rotary switch. If the level is floating, a logical 1 is sensed and the 'multisystem mode' trigger is set; if CPU dc ground is returned, a logical 0 is sensed and the 'multisystem mode' trigger is reset. Switching between Partition and Model 65 modes is not synchronized with the operation of the CPU.

I/O Allocation Control

There are two switches on the configuration control panel for each Two-Channel Switch feature. Unlike all the other switches, they do not interface with the CPU logic. Refer to the FETOM of the applicable control unit for the operational description of the Remote Switch Attachment feature.

BCU Modifications

The following discussion explains the modifications to the BCU to accommodate up to 8 shared main storage units.

Storage Address Decoding with Prefixing Disabled

In a 2065 without the Multisystem feature, SAB(4,5) is decoded to select one of up to four 2365 Processor Storage units. Because Additional Storage Attachment features might be installed, the Multiprocessing system must provide for selecting up to eight storage units. Therefore, storage address decoding is concerned with SAB(3–5). Because of floating addressing, SAB(3–5) is not directly decoded but rather compared to the three binary-encoded lines returned from the Floating Address switches of each storage unit. If an equal comparison is made and the 262K block of addresses is available to the CPU, then the 'select' signal is issued to that storage unit.

Diagram 7-4, FEMDM, shows the storage address decoding. The circuits shown in the lower part of the diagram relocate direct addresses; if the CPU is not enabled for prefixing, these circuits are inactive. When the CPU has priority to access main storage, the address of the desired storage location is gated from D, IC, or PAL to SAB via a 24-bit OR in the BCU. Bits 3, 4, and 5 are gated from the OR by the 'not channel priority' signal to floating address comparison circuits, one for each storage unit. The desired storage unit is decoded if the storage unit with the equal comparison is available to the CPU and if the Floating Address switches are properly set. As in the 2065 without the Multisystem feature, bits 6 and 20 of the storage address are decoded in parallel to the decoding of bits 3–5; this decoding causes an even or odd interleaved storage selection to be made by the 'select timing pulse' signal. Unlike storage requests made by the CPU, the storage address decoding is done directly from the SAB when the storage request is made by an attached channel. Otherwise, the same decoding circuits are used for both CPU- and channel-initiated storage requests.

Storage Address Decoding with Prefixing Enabled

Three storage address detectors, in parallel with the decoding of the storage address and activated by the 'enable prefix' trigger, determine when prefixing is needed (Diagram 7-4, FEMDM). One detector determines if the CPU is attempting to access a location within its first 4K block of main storage addresses (prefix compare CPU). Another detector determines if an attached channel is attempting to access that first 4K block (prefix compare channel). The third detector determines if the CPU is attempting to access its last 4K block (reverse prefix compare). If such is not the case, prefixing is not needed and the 'select timing pulse' signal is issued to the decoded storage unit as a selection signal, just as if prefixing were not enabled. If, however, one of the three detectors determines that prefixing is needed, the 'select timing pulse' signal is blocked from being sent to main storage as a selection signal. Instead, the 'select timing pulse' signal sets the 'normal prefix' latch if relocation to its highest 4K block is needed (prefix compare CPU or channel) or sets the 'reverse prefix' latch if relocation to its lowest 4K block is needed (reverse prefix compare).

With either the 'normal prefix' or 'reverse prefix' latch set, the SAB is changed and a different storage unit is decoded to reflect this change. The 'normal prefix' latch changes SAB(3–5) to the 'floating prefix bit' signals and SAB(6–11) to all 1's. Thus, instead of the storage unit with

the lowest addresses being decoded, the storage unit with the highest addresses is now decoded. The 'reverse prefix' latch changes SAB(0–11) to all 0's by degating the IC and D inputs to the SAB OR in the BCU. Because the BCU cannot degate bits put on the SAB by the channel, reverse prefixing is limited to CPU storage requests for the highest 4K block. The 'normal prefix' or 'reverse prefix' latch sets the 'any prefix' trigger, which deactivates the three storage address detectors; this action allows the second 'select timing pulse' signal, generated because a selection has been unsuccessfully tried, to be issued as an even or odd selection signal to the correct storage unit.

Invalid Storage Address

Addressing is invalid if:

1. More than one allocated storage unit is assigned the same addresses, whether or not requested.
2. Any storage unit, whether or not requested, is manually assigned an address interval greater than 1048K and an additional Storage Attachment feature is not installed.
3. The storage unit containing the requested address is not allocated to the CPU, is in test mode, or does not have dc power on.
4. No storage unit is assigned the requested address.
5. The requested address is higher than 2,097,151, decimal; that is, SAB(0, 1, or 2) = 1.

Specific circuits in the BCU are needed to detect all invalid addresses except for the fourth case listed: no storage unit assigned the requested address. In this case, a successful comparison of the three 'combined SAB' lines to the Floating Address switches does not occur. Therefore, none of the 'decode frame X select' lines are activated at the time of the first 'select timing pulse' signal. No plug card is needed to specify the highest valid address as in the 2065 without the Multisystem feature.

Just as in the 2065 without the Multisystem feature, a second 'select timing pulse' signal is generated when a selection of an invalid address is tried but a 'select' signal is not sent. The second signal is issued to the first available storage unit as a selection signal together with a 'cancel' signal. The first available storage unit is identified by the lowest-numbered physical frame returning a 'power on frame' level to the CPU. Although prefixing also blocks the first 'select timing pulse' signal and causes a second one to be generated, the requested address is not considered invalid and a 'cancel' signal is not sent.

BCU-Storage Operations

The BCU-storage operations differ from the 2065 without the Multisystem feature in two major respects: (1) the storage selection signal is issued even if the decoded storage unit is busy, and (2) an extra decoding cycle is added by the prefixing operation.

Because the selection signal is issued even if the decoded storage unit is busy, the BCU-cleanup operation is delayed until the 'accept' signal is returned from the storage unit. A non-busy storage unit immediately returns an 'accept' signal to the BCU. Even though not busy, the response time from different storage units varies because of different cable lengths to the CPU. A busy storage unit records the 'select' signal until all higher-priority requests are serviced. For 'select' signals arriving at a busy storage unit, the time necessary to establish priority is overlapped with the storage cycle in progress.

A CPU-initiated storage request is shown in Diagram 7-5, FEMDM. The 'accept' signal is received 300 ns after the earliest expected time. Because the 'accept' signal is not detected at the start of the cycle following the issuance of the selection signal, an 'inhibit oscillator' signal is generated (line 25 in the diagram). The BCU and CPU clocks are inhibited at not-clock time by turning off the gated oscillator pulses. The gated oscillator is similar to that used in the Model J65; see Chapter 2, Section 1. The 'accept' signal forms the not-clock-time pulses needed to complete the stopped cycle and turns on the gated oscillator that forms the succeeding timing pulses (lines 2 and 3). The 'BCU cleanup' signal (line 30) is generated by the setting of the 'HSS accept' latch (line 29). Therefore, not until then are the CPU priority circuits reset. However, to speed up the servicing of pending channel requests, the 'accept' signal is delayed 25 ns to form the 'timed accept pulse' signal (not shown). The 'timed accept pulse' signal scans the channel sync latch outputs, setting the channel priority trigger of the highest-priority request regardless of the BCU cleanup.

The extra cycle needed for the normal prefixing of a CPU storage request is shown in Diagram 7-6, FEMDM. The BCU timing of a channel request needing normal prefixing or a CPU request needing reverse prefixing is the same as that shown. Notice that the CPU clock is stopped for one cycle, even though in the example shown there is no delay receiving the 'accept' signal from the selected storage unit. This ensures that the data is gated at the correct time. When prefixing a channel request, the CPU clock is not stopped.

Multisystem Timer

The multisystem timer is a four-bit counter made up of polarity-hold circuits. It is added by the Multisystem feature to detect CPU inactivity and to delay the external system reset operation. Counting occurs only in Multisystem mode. The interval timer must be enabled for CPU inactivity to be detected and identified as a machine malfunction.

Operation

Diagram 7-7, FEMDM, shows the logic, timing, and counting scheme of the multisystem timer. 'Time clock

step' pulses are counted if the CPU is in the Multisystem mode and either the interval timer is enabled or an external system reset operation is in progress. These 300-ns 'time clock step' pulses arrive at input-power line frequency (every 16 2/3 ms in 60-Hz systems; 20 ms in 50-Hz systems). A 100-ns time delay forms two 100-ns pulses; the first pulse, 'update overflow counter', is followed 300 ns later by the second pulse, 'transfer counter to latch'. Until the conditions for a counter reset are met, the counter triggers are set per the latches by the 'update overflow counter' pulse; following this, the triggers are set directly into the latches by the 'transfer counter to latch' pulse. The counter then waits for the next 'time clock step' pulse.

The counter is reset to zero by either an end op or an 'external system reset' signal. Both of these conditions degate the inputs to the triggers, update the triggers to zero, and then transfer the zero state of the triggers to the latches. The 'end op' trigger is set for 200 ns; therefore, in this case the second 100-ns pulse follows the first by 200-ns. The 'external system reset' signal is about 80 ns long, so in this case the two pulses are about 70 ns long and the second follows the first by 100 ns.

System Hang Timing

The timer, when in the Multisystem mode, counts the number of input-power frequency cycles occurring from the latest end-op cycle. While the CPU is operating normally, the counter may count one cycle, two at the most, before the next end op occurs. When the high-order bit of the counter, the 'system hang' trigger, is set, eight cycles are counted and it is assumed the CPU has ceased to operate, or is "hung", due to a machine malfunction. This assumption need not always be true. For example, a slow external device can cause the Read Direct instruction to be excessively long. Performing a manual storage ripple operation (as when clearing storage) with the interval timer enabled also sets the 'system hang' trigger; therefore, the interval timer should be manually disabled when clearing storage in Multisystem mode. The malfunction need not occur in the CPU either. For example, as previously discussed, the CPU and BCU clocks are inhibited until an 'accept' signal is received from the selected storage unit. If a machine malfunction in storage prevents the 'accept' signal from being issued, the CPU is "hung" and the 'system hang' trigger is set.

The eight 'time clock step' pulses are counted within 116.7 to 133.3 ms because the end op is independent from the counting and the period of a 60-Hz pulse is 16-2/3 ms. (140 to 160 ms are needed if the input-power frequency is 50 Hz.) The 'system hang' trigger sets the 'system hang error' latch and, if set, resets the gated oscillator control latches: 'select sent' and 'inhibit'. The 'system hang error' latch sets the 'error' trigger in the machine check interruption logic. The resetting of the oscillator control latches ensures that the BCU and CPU clocks are running; without these timing pulses the logout and machine check interruption microprograms cannot be performed.

External System Reset Timing

The timer delays the resetting of the system by the 'external system reset' signal for two counts of the 'time clock step' pulse. The 'external system reset' signal sets the 'external system reset' trigger and resets the counter. The 'external system reset' trigger then allows 'time clock step' pulses to be counted. The second 'time clock step' pulse updates the counter, setting the 'overflow 1' trigger. This action requires 16.7–33.3 ms for 60-Hz power (20–40 ms for 50-Hz power). At that time, then, the 'external reset delayed' trigger is set.

MULTISYSTEM OPERATIONS

The Multisystem feature modifies three instructions (Set System Mask, Write Direct, and Read Direct), and adds six operations (one per multisystem signal).

Set System Mask Instruction (Multisystem Mode)

The Set System Mask instruction operates as described in Chapter 3, Section 7, if the CPU is not in Multisystem mode. In Multisystem mode, the instruction is always suppressed and a program interruption occurs: in the Problem state, an interruption code of 2 (privileged operation) is stored into the old PSW; in the Supervisor state, an interruption code of 18, decimal (multisystem), is stored into the old PSW.

To set an interruption code of 18, the 'interrupt code 16' trigger is added by the Multisystem feature (Diagram 5-22, FEMDM). The 'SPEC' micro-order in the first ROS word sets 'interrupt code 2' and '16' triggers and forces an end op if the 'multisystem mode' trigger is set in the Supervisor state.

Write Direct Instruction (Not Model 65 Mode)

The Multisystem feature modifies the direct control circuits for the Write Direct instruction (Diagram 7-8, FEMDM) but does not change the write-direct microprogram. If the CPU is in Model 65 mode, the Write Direct instruction operates as described in Chapter 3, Section 7.

In the Partition mode, the instruction begins with a privileged operation test. If PSW(15) = 1, 'interrupt code 2' trigger is set and the instruction is suppressed by forcing an end op and is followed by a privileged operation program interruption. If PSW(15) = 0, the CPU is in the Supervisor state and execution continues. The 'partition mode' signal disables the invalid op code detector in case the DISABLE DIRECT CONTROL switch is in the active (down) position, preventing an operation program interruption. The

'partition mode' signal also prevents the issuing of E(8–11) to the external device when the 'timing gate' trigger is set. If the DISABLE DIRECT CONTROL switch is active, the 'direct control write out' signal and E(8–15) are blocked when the 'timing gate' trigger is set. The data byte is fetched and placed into the G-register as in Model 65 mode, and the instruction is completed with an end op regardless of the position of the DISABLE DIRECT CONTROL switch.

In Multisystem mode, the instruction, beginning with the same privileged operation test, continues if PSW(15) = 0. The 'multisystem mode' trigger disables the invalid op code detector, preventing an operation program interruption if the DISABLE DIRECT CONTROL switch is active. The 'multisystem mode' trigger also prevents the issuing of E(8–11) to the external device when the 'timing gate' trigger is set. If the DISABLE DIRECT CONTROL switch is active, the 'direct control write out' signal and E(8–15) are blocked when the 'timing gate' trigger is set. Regardless of the position of the DISABLE DIRECT CONTROL switch, the multisystem signals, E(8–11), are decoded and issued to the other CPU when the 'timing gate' trigger is set; 'system reset', 'external start', 'log I/O interrupt', and 'system call' signals can be issued by the Write Direct instruction only when executed in Multisystem mode. The data byte is fetched and placed into G as in Model 65 mode, and the instruction is completed with an end op.

Read Direct Instruction (Not Model 65 Mode)

The Multisystem feature modifies the direct control circuits but does not change the read-direct microprogram. The Read Direct instruction (Diagram 7-8, FEMDM) operates as described in Chapter 3, Section 7, if the CPU is in the Model 65 mode.

In Partition or Multisystem mode, the instruction begins by testing whether PSW(15) = 0. If it does not equal 0, 'interrupt code 2' trigger is set and the instruction is suppressed by forcing an end op and is followed by a privileged operation program interruption. If PSW(15) = 0, the CPU is in the Supervisor state and the execution continues. The 'partition mode' signal or the 'multisystem mode' trigger disables the invalid op code detector in case the DISABLE DIRECT CONTROL switch is in the active (down) position, preventing an operation program interruption. The 'partition mode' signal or 'multisystem mode' trigger also prevents the issuing of E(8–11) to the external device when the 'timing gate' trigger is set. If the DISABLE DIRECT CONTROL switch is active, the 'direct control read out' signal and E(8–15) are blocked when the 'timing gate' trigger is set. The data byte received from the external device is stored as in Model 65 mode, and the instruction is completed with an end op.

Malfunction Alert

The 'malfunction alert' signal, generated only in Multisystem mode, results from either a CPU machine check or a loss of CPU power. A CPU machine check is detected when the 'error' trigger is set and the CPU CHECK switch is in the PROC (process) position. A loss of CPU power is detected by the dropping of the 'enable direct control' relay, K50; K50 is one of the first relays to drop during a manual or automatic power-off operation. Either of these conditions triggers a singleshot, generating a 825-ns pulse (Diagram 7-9, FEMDM). This pulse is issued to the other CPU via a gated driver. All the direct control signals use a gated driver to prevent the issuing of spurious signals during normal power-up and power-down operations. However, the gating of the 'MS malfunction alert' signal is handled slightly differently to assure its issuance when the CPU power is removed, whether accidentally or purposely. A reed relay provides an electromechanical delay of the 'power off' signal. The reed relay is on a standard SLT card and is not a special circuit. The 5-second pick delay of the 'power on' relay, K47, prevents the 'MS malfunction alert' signal from being issued during the power-on sequence.

The 'MS malfunction alert' signal sets the 'external signal 2' latch in the receiving CPU if that CPU also is in Multisystem mode. If no higher-priority exceptional conditions or interruptions are pending at the next end op and the external mask bit is set, the receiving CPU is then externally interrupted.

Gated Load

Depressing the LOAD pushbutton causes a normal initial program load operation and, if the CPU Mode switch is in the MS position, the issuance of an 'MS gated load pb' signal (Diagram 7-9, FEMDM) to the other CPU. The driver for this multisystem signal is gated by the same electromechanically delayed 'power off' signal as the 'malfunction alert' signal.

The receiving CPU forms the 'MS gated load pb' signal into a 30-usec pulse if it also has its CPU Mode switch in the MS position. This pulse resets the receiving CPU and its attached units at the same time the sending CPU and its attached units are being reset. The receiving system is reset as if its SYSTEM RESET pushbutton were depressed, as shown in Diagram 8-7, FEMDM. At the end of the reset operation, the CPU is in the stop loop.

System Call

An 'MS system call' signal is issued when a Write Direct instruction with bit 11 specified is executed in Multisystem mode; it sets the 'external signal 3' latch in the receiving CPU if that CPU also is in Multisystem mode. If no higher-priority exceptional conditions or interruptions are pending at the next end op and the external mask bit is set, the receiving CPU is then externally interrupted.

Log I/O Interrupt

- Three latches control receiving CPU, not ROS.
- D is forced to address of I/O old PSW, and 'priority 1' and 'priority 2' triggers are set.
- 'Timing gate' trigger gates 'interrupt response' signal to highest-priority channel requesting I/O interruption.
- After channel stores CSW, CPU issues D-request to store interruption code.
- CPU is reset before and after log I/O interrupt operation.

A 'log I/O interrupt' signal is issued when a Write Direct instruction with bits 8 and 9 of the I2 field set to 11 is executed in Multisystem mode. If the receiving CPU is also in Multisystem mode, it is reset and then its 'manual' trigger and 'log I/O preparation' latch are set, initiating the log I/O interrupt operation (Diagram 7-10, FEMDM). Setting the 'manual' trigger lights the MANUAL indicator on the system control panel, disables the interval and multisystem timers, and stops the selected usage meter on the CPU; because the 'stop' trigger is not set, the stop loop is not entered. Setting of the 'log I/O preparation' latch prevents further ROS control of the CPU by holding the ROS sense latches reset. The receiving CPU is now controlled by the three sequence latches: 'log I/O preparation', 'log I/O execute', and 'log I/O terminate'. The operation of the channels, main storage, and the BCU is not affected. Direct address relocation remains active if the CPU's PREFIX switch is set to ENABLE.

Once the 'log I/O preparation' latch is set, up to five machine cycles might elapse before FLT-2 time when the 'log I/O execute' latch is set. With both of these latches set, the interrupt request trigger of the highest-priority channel requesting an I/O interruption is set, regardless of the channel mask bits in PSW(0–6). Next, at FLT-1 time plus 80 ns, a 'branch or normal end op' signal is generated that (1) forces D to 38 (hex), the location of the I/O old PSW, and (2) sets the 'priority 1' and 'priority 2' triggers, preventing any further change of the 'channel X interrupt request' triggers and setting the 'timing gate' trigger. If any 'channel X interrupt request' trigger is set, the setting of the 'priority 1' and 'priority 2' triggers is ensured by either the resetting or the inhibiting of all higher-priority exceptional conditions and interruptions.

The 'timing gate' trigger gates the 'interrupt response' signal to the channel determined by the 'channel X interrupt request' trigger set. The channel then proceeds to store a CSW into direct-address bytes 40–47. The channel's interruption operation is unchanged by the log I/O interrupt operation. When the channel finishes storing the CSW, it sets the 'release CPU' latch in the CPU. Waiting until FLT-2 time, the 'log I/O terminate' latch is set. At FLT-3 time, the 'log I/O store interrupt' signal is generated, causing:

1. The 'log I/O preparation' latch to be reset.
2. The I/O interruption code to be transferred to S(21–31).
3. The 'D sync' trigger to be set.
4. The 'mark 2' and 'mark 3' triggers to be set.

During the next machine cycle a 'log I/O terminate and 0 time' signal is generated, resetting the CPU. The CPU is now ready to receive another 'log I/O interrupt' signal from the other CPU. Meanwhile, the BCU, if not busy with a channel storage request, immediately services the D-request. D still contains the direct address of the I/O old PSW, 38 (hex). Because only 'mark 2' and '3' signals are issued, only the interruption code field is altered.

The log I/O interrupt operation ends with the 'log I/O terminate' latch set. The latch is reset while another log I/O interrupt operation is executed or if a system reset is performed. Until a system reset is performed, either the 'log I/O preparation' or 'log I/O terminate' latch is set, holding the ROS sense latches reset.

If there is no I/O interruption request pending when a 'log I/O interrupt' signal is received, the CPU waits with both the 'log I/O preparation' and 'log I/O execute' latches set. Once every four cycles the forced 'branch or normal end op' signal tests whether any 'channel X interrupt request' trigger is set.

External System Reset

A 'system reset' signal is issued when a Write Direct instruction with bits 8 and 9 set to 01 is executed in Multisystem mode. The 'system reset' signal sets the 'external system reset' trigger in the receiving CPU if that CPU's Mode switch is in the MS position. When set, this trigger performs five functions:

1. Inhibits further setting of the channel sync latches, preventing the honoring of channel storage requests.
2. Blocks the setting of the 'storage 2' trigger, preventing selection of main storage.
3. Blocks the issuance of any 'malfunction alert' signals.
4. Enables the multisystem timer.
5. Keeps external starts pending until the system is reset.

The multisystem timer delays the system reset operation until all processing has stopped. When the multisystem timer counts two line frequency cycles, the receiving system is reset as if the SYSTEM RESET pushbutton on its system control panel were depressed. At the end of the reset operation, the CPU is in the stop loop.

External Start

An 'external start' signal is issued when a Write Direct instruction with bits 8 and 9 set to 10 is executed in Multisystem mode; it sets the 'external start' trigger in the receiving CPU if that CPU also is in the Multisystem mode.

This trigger sets the 'external start' latch, waiting first for any end-op or external system reset operations in progress to finish. The 'external start' latch puts the CPU into the stop loop by setting the 'stop' trigger. When the stop loop tests whether the PSW RESTART pushbutton has been depressed (ROS micro-order J30), the 'external start' latch forces ROSAR(11) to set, causing a branch to the load PSW routine. Also at this time the 'enable prefix' trigger is set per the PREFIX switch on the configuration control panel.

POWER DISTRIBUTION AND CONTROL

The Multisystem and Additional Storage Attachment features do not change the power distribution of the CPU except that in CPU 2 the DC return bus is not connected to the frame bond (logic YA023).

With the installation of the Multisystem feature, relay K6 in the CPU is no longer held picked by the last active 'storage driver on' signal (logic YA021). Therefore, when the POWER OFF pushbutton is depressed on the system control panel, K6 immediately drops regardless of the status of the storage units. The dropping of K6 does not allow the system power-off sequence to wait until the core driver power of the last storage unit is off but immediately turns off CPU power, channel power, and (only if the POWER OFF pushbutton is being depressed on the last CPU with power on) main storage power. To protect the contents of main storage, each 2365 Model 13 contains gated line drivers similar to those used in the Direct Control feature in the CPU. The gated line drivers are the first circuits to be turned off during the storage power-off sequence and the last circuits to be turned on during the storage power-on sequence. Refer to the FETOM of the 2065 Model 13 storage for a complete discussion of its power controls.

Several of the power controls are modified by both the Multisystem feature and the first Additional Storage Attachment feature. The changes are discussed next under "Console Controls and Maintenance Features".

CONSOLE CONTROLS AND MAINTENANCE FEATURES

The following paragraphs describe the configuration control panel and the modifications made by the Multisystem feature to the system control panel and to the logout and scan-in operations.

Configuration Control Panel

The configuration control panel is added by the Multisystem feature and is shared by the two CPU's. For the following discussion of the configuration control panel, refer to Diagram 7-11, FEMDM.

Storage Allocation Switches

A Storage Allocation toggle switch for each CPU is associated with each storage unit. Only with the switch set to ENABLE is the storage unit available to that CPU. At least one storage unit is allocated to each operating CPU. The same storage unit may be allocated to both CPU's. A change in storage allocation becomes effective when the associated CPU enters the Stopped or Wait state.

Floating Address Switches

One Floating Address switch is associated with each storage unit. These rotary switches control the address range of each storage unit in 262 kilobyte intervals. The first four positions are: 0 TO 262K, 262K TO 524K, 524K TO 786K, and 786K TO 1048K. If the first Additional Storage Attachment feature is installed, these switches may also be placed in one of the next four positions: 1048K TO 1310K, 1310 TO 1572K, 1572K TO 1834K, and 1834K TO 2096K.

Contiguous addresses are usually assigned for each CPU, starting with address 0. Two storage units may not be assigned the same address interval if either is allocated to both CPU's. To provide maximum thruput, the direct-address main storage unit(s) should be as close to the CPU as possible; refer to Diagram 7-2, FEMDM. The Floating Address switches are always active.

PREFIX Switches

One PREFIX toggle switch is associated with each CPU. With the switch set to ENABLE, CPU references to the low-order 4096 (decimal) bytes and the high-order 4096 bytes of main storage are swapped, or relocated; also, channel references to the low-order 4096 bytes are relocated to the high-order 4096 bytes. Relocation does not occur when the PREFIX siwtch is set to DISABLE. Normally, neither of these two switches is set to ENABLE when no storage units are shared; only one is enabled when all storage units are shared. A change in position of a PREFIX switch becomes effective when the associated CPU (1) performs a power-on sequence, (2) has its LOAD pushbutton depressed, or (3) performs an external start operation.

CPU Mode Switches

One CPU Mode switch is associated with each CPU and determines the operating mode of that CPU. Multisystem mode (MS) is entered or exited when the CPU clock is stopped, according to the position of the switch. Switching between Partition (PTN) and Model 65 (65) modes is immediately effective.

I/O Allocation Switches

These switches enable or disable communications between the associated CPU and the control unit attached to two channels. Switches for up to 24 control units can be accommodated on the configuration control panel. These switches are labeled as a group, *I/O CONTROL*, and individually according to the control unit type. Refer to the applicable control unit FETOM for a further description of these switches.

VALID ADDRESS Indicators

One VALID ADDRESS indicator is associated with each CPU. When not lit, an incorrect manual assignment of an allocated storage unit has been made: either the same floating addresses have been assigned more than once or a Floating Address switch is in one of the last four positions without an Additional Storage Attachment feature being installed. The indicator lights after the needed correction is made. Not until then can the associated CPU be operated.

System Control Panel Modifications, Multisystem Feature

EMERGENCY PULL Switch

The pulling of an EMERGENCY PULL, EPO, switch on either CPU removes power beyond the entry point from all units in the multiprocessing system. The two EPO switches of the CPU's are directly interconnected by Emergency Power-Off Control feature No. 3621 or, if there are other EPO switches in the multiprocessing system, by Emergency Power-Off Control feature No. 3622.

POWER ON Pushbutton

The POWER ON pushbutton on the system control panel of each CPU turns on the power of that CPU and all the units, shared and nonshared, controlled by that CPU. Power is applied to all storage units. The POWER OFF pushbutton removes power from that CPU and all the nonshared units controlled by that CPU; however, if the power of the other CPU has previously been turned off, the power of the shared units and all storage units is also turned off. The power of units having their LOCAL/REMOTE switch in LOCAL is not controlled.

Marginal Voltage Control

The +6VM power supplies of the channels attached to the CPU can be varied from the system control panel. Only the storage units with the 6VMC switch (on each 2365's power control panel) in the proper position can be margined from the system control panel: SYSA for CPU 1 and SYSB for CPU 2.

DISABLE INTERVAL TIMER Switch

The activation of the DISABLE INTERVAL TIMER switch (N13) prevents decrementing of the interval timer and incrementing of the multisystem timer.

DISABLE DIRECT CONTROL Switch

The activation (depression) of the DISABLE DIRECT CONTROL toggle switch (N27) still disables the 'direct control timing signal out' bus and the external signals received from the external device. However, only in Model 65 mode does a Read Direct or Write Direct instruction cause a program interruption when the switch is active. In Multisystem and Partition modes, the instructions are executed but no direct control timing signals are issued to the external device. The DISABLE DIRECT CONTROL switch does not affect the issuing or receiving of the six multisystem signals.

Storage Switches

The storage ADDRESS switches must be set to the relocated storage address, when applicable, for scoping or stop-on-storage-address-compare functions. Store and Display manual operations are not affected.

When the CPU's share main storage, the DEFEAT INTERLEAVING switches (N9) on both system control panels must be in the same position (unless specified otherwise during a diagnostic program).

When a storage unit available to both CPU's detects an error while the requesting CPU's STOP ON STORAGE CHECK switch (N11) is active, the operation of that storage unit immediately stops. Therefore, if the requesting CPU has not already received the 'accept' signal, its operation is also stopped. Eventually, when both CPU's are waiting for the 'accept' signal from the stopped storage unit, the entire multiprocessing system enters the Stopped state. The CPU's do not automatically enter the stop loop.

Indicators

The indicators added by the Multisystem feature are shown in Diagram 7-12, FEMDM. The STORAGE FRAME X ENAB indicator shows the storage unit's status with respect to the CPU; regardless of that status, the STORAGE FRAME X 1048K, 524K, and 262K indicators show the binary-encoded position of the Floating Address switch and, hence, the storage unit's lowest address. The CPU identification indicators 1 and 2 are wired into the respective CPU's during system installation. The remaining indicators are self-explanatory.

System Control Panel Modifications, Additional Storage Attachment Features

The installation of the first Additional Storage Attachment feature adds a section to the configuration control panel (as shown in Diagram 7-11, FEMDM) and further modifies the system control panel as next described.

Marginal Voltage Control

Marginal voltage control and metering for storage units 1 through 4 can only be done from CPU1. For storage units 5 through 8, marginal checking can only be done from CPU2.

POWER CHECK Indicators

POWER CHECK indicators for storage units 1 through 4 are on CPU1; those for storage units 5 through 8 are on CPU2.

STORAGE INDICATE Switch and Indicators

A four-position rotary switch is added to panel D, replacing the two-position STORAGE INDICATE lever switch on panel F. The four positions are 1–4, 5–8, 9–12, and 13–16 for the basic storage modules (BSM's) in storage units 1 and 2, 3 and 4, 5 and 6, and 7 and 8, respectively. The display, shown in Diagram 7-123, FEMDM, is made regardless of storage allocations.

Logout and Scan In

The indicators added by the Multisystem feature are included in the logout data. Only in Multisystem and Partition modes are the 16 GPR's also included in the logout data (Diagram 8-115, FEMDM). Three feature triggers can be set by a scan-in operation: 'enable prefix', 'external start', and 'external system reset'.

www.IBM360.info

When the 2065 CPU was introduced, the 2060 CPU's already produced were converted to 2065's. Although the functions of the two resulting power systems are the same, it is occasionally necessary to differentiate between the converted units and those originally manufactured as 2065's. Converted and original units may be identified by comparing the CE panel with Diagram 8-29, FEMDM. Additional variations in the power system are described for 50-Hz and 60-Hz line frequency and the installed optional features.

AC POWER DISTRIBUTION

The primary ac power distribution is shown in Figure 5-1 for 60-Hz units and in Figure 5-2 for 50-Hz units.

60-Hz Units

Main power from circuit breaker 1 (CB1) is applied to transformer T1 via fuses F1 and F2 (Fig. 5-1). T1 provides 28V ac for the channel's and 2365's remote margin power, the alarm circuit, and a full-wave bridge rectifier. This bridge rectifier supplies 24V dc to the Emergency Power-Off (EPO) loop and, via PK1 contacts, to the 24V dc bus. (The 24V dc bus energizes the thermal protection circuits and the relay gate.)

Main power is applied by PK1 contacts to:

1. T2 via CB12. T2 provides 115V ac to convenience outlets in the CPU and the 2365's via CB11, and to the 1052 Printer-Keyboard via F4 and PK2 contacts.
2. T3 via F3 and CB2. T3 provides 40V ac to the usage meters, 12.6V ac to the interval timer logic and the 1052 adapter logic, and low-voltage ac to another full-wave bridge rectifier, which in turn provides ±20V dc to the undervoltage (and overvoltage in the converted units) protection circuits.
3. T4 via F3 and CB7. T4 provides 28V ac to the converter/inverter.
4. PK2 via CB9.

Main power is applied by PK2 contacts to:

1. Gate and power supply blowers via CB8 (and CB3 through CB6 in the converted units).
2. Converter/inverter.

50-Hz Units

Main power is taken across the three phases or across each phase and neutral, depending on the voltages at the location (Figure 5-2). Transformer taps and switches are adjusted accordingly.

Main power from CB1 is applied to T1 via F1 and the phase/neutral terminals (Switchboard 1, SW BD1).

T1 provides 28V ac for the channel's and 2365's remote margin power, the alarm circuit, and a full-wave bridge rectifier. This bridge rectifier supplies 24V dc to the EPO loop, and, via PK1 contacts, to the 24V dc bus.

Main power is applied by PK1 contacts to:

1. Convenience outlets in the CPU and 2365's via CB5 and the phase/neutral terminals (SW BD2), and to 1052 Printer-Keyboard via CB5, PK2 contacts, and the phase/neutral terminals (SW BD2).
2. T3 via F2, CB2, and the phase/neutral terminals (SW BD2). T3 provides 40V ac to the usage meters, 12.6V ac to the interval timer logic and the 1052 adapter logic, and low-voltage ac to another full-wave bridge rectifier, which in turn provides ±20V dc to the undervoltage protection circuits.
3. T4 via F2, CB3, and phase/neutral terminals (SW BD2). T4 provides 28V ac to the converter/inverter.
4. PK2.

Main power is applied by PK2 contacts to:

1. Gate and power supply blowers via CB4, T2, and phase/neutral terminals (SW BD 2).
2. Converter/inverter via a three-phase autotransformer.

Converter/Inverter

The converter/inverter converts the main 50-Hz or 60-Hz, 3-phase ac to dc and inverts the dc to 140V, 2500-Hz, 1-phase ac for the regulators. A detailed description of the operation is given in the *SLT Power Supply,* FEMI, Form 223-2799. Diagram 6-1, FEMDM, is a simplified schematic.

The 3-phase wall power is converted to dc by the 3-phase bridge rectifier. The output of the rectifier is filtered by capacitors C0, C1, and C2. In addition to filtering, C1 and C2 provide a split source for the inverter. Resistors R1 and R2 help balance this split source and provide bleeder loading to discharge the capacitors when power is turned off.

The inverter is basically two silicon-controlled rectifiers (SCR's) that alternately switch the load across Edc at a 2500-Hz rate. A 2500-Hz square wave is formed at load points A and B. Switching is performed by the two load SCR's, SCR1 and SCR2, and the two commutation SCR's, SCR3 and SCR4. Assume SCR1 and SCR3 are gated on. Load current I1 enters the load at point A. Capacitor C3 charges to Edc. When C3 reaches full Edc, SCR3 turns off. At this point, SCR4 is gated on. C3 discharges against

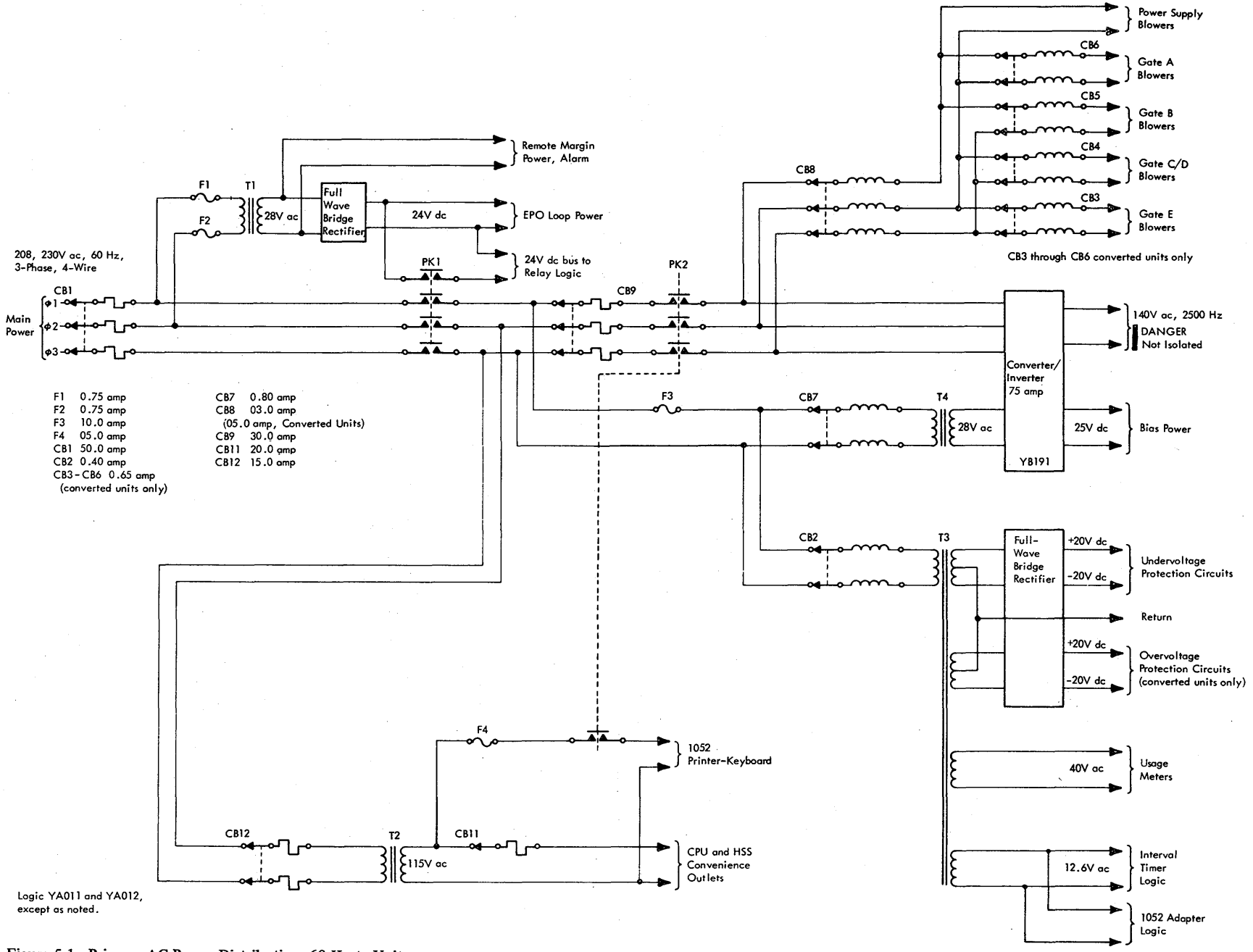


Figure 5-1. Primary AC Power Distribution, 60-Hertz Units

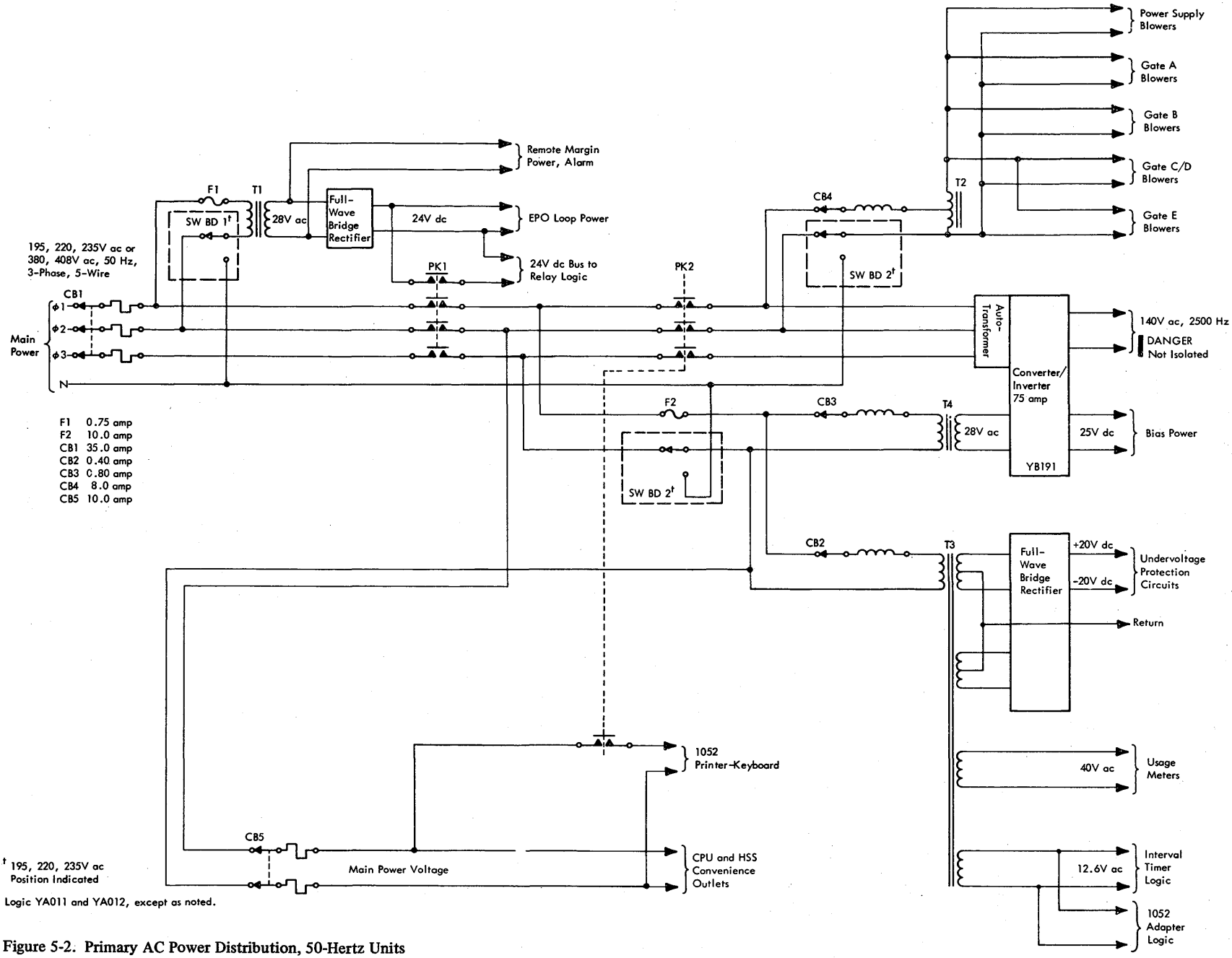


Figure 5-2. Primary AC Power Distribution, 50-Hertz Units

SCR1, turning it off. At this point, SCR2 is gated on. Load current I2 enters the load at point B; C3 charges to Edc. When C3 reaches full Edc, SCR4 turns off. At this point, SCR3 is gated on. C3 discharges against SCR2, turning it off. The cycle then repeats with SCR1 gated on.

DANGER
The output to the regulators is not isolated from the wall power, presenting a lethal potential to ground.

DC POWER DISTRIBUTION

The various dc voltages required by the CPU logic are provided by 15 regulators. A 16th regulator, the 48V power supply (PS9), is not installed in the CPU when the 2150 Console option is used. The 2150 Console has a multivoltage power supply which provides the 48V dc. Table 5-1 lists the output voltage levels and the major load for each regulator. Figure 5-3 is representative of their distribution.

High-Frequency Regulator Modules

- CPU regulators:

Volts, dc	Amps, dc	Type	Qty
+6	25	SCR Control	1
+6	40		4
+3	40		7
-3	40		2
-18	11	Magnetic amplifier control	1

Table 5-1. High-Frequency Regulator Modules

Power Wall Location	Nominal Output	Logic	Load	
			CPU Gate	Other
PS1	+6V 25 amp	YA061	C and D upper	—
PS2	+3V 40 amp	YA062	B lower	—
PS3	+3V 40 amp	YA061	E lower	—
PS4	-3V 40 amp	YA062	C and D	—
PS5	+3V 40 amp	YA061	B and E upper	—
PS6	+3V 40 amp	YA062	C and D	—
PS7*	+6V 40 amp	YA061	E	—
PS8*	+6V 40 amp	YA062	C and D lower	—
PS9†	+48V 2 amp	YA141	—	Keyboard, printer, and audible alarm of 1052
PS10	-3V 40 amp	YA072	A, B, and E	—
PS11*	-18V 11 amp	YA071	C and D upper	—
PS12	+3V 40 amp	YA072	A lower	Upper roller switch indicators
PS13	+3V 40 amp	YA071	B upper	—
PS14	+3V 40 amp	YA072	A upper	Lower roller switch indicators, usage meters
PS15*	+6V 40 amp	YA071	B	—
PS16*	+6V 40 amp	YA072	A	—

* Marginable regulator.
† PS9 is added by 1052 Adapter Feature if 2150 Console is not included in system.

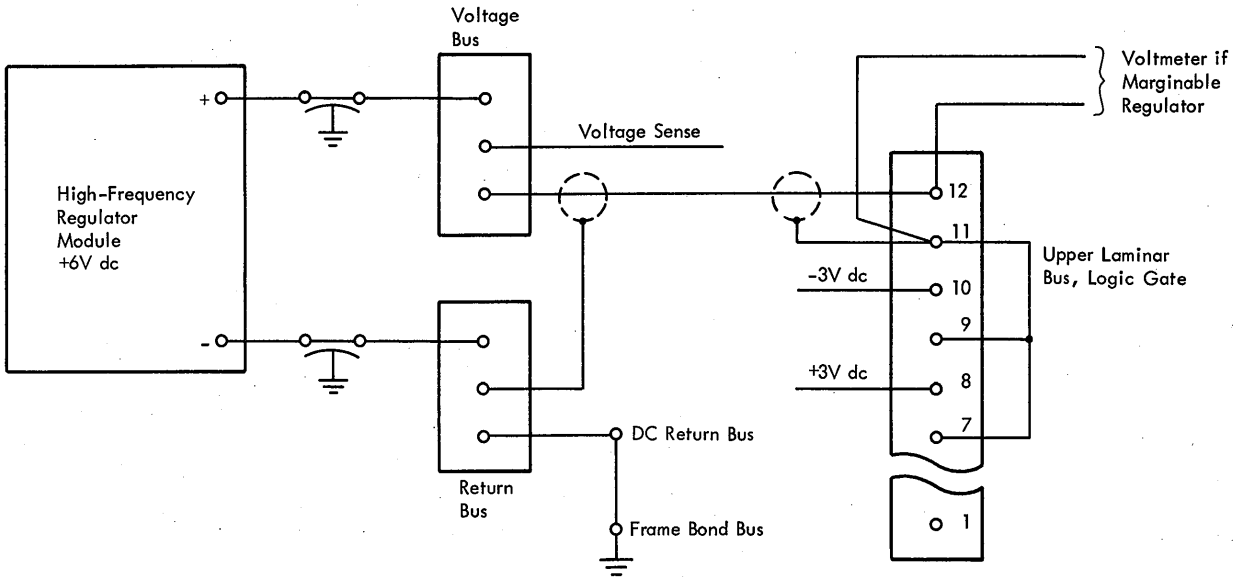


Figure 5-3. Representative DC Distribution

The high-frequency regulators rectify, control, and filter the 2500-Hz ac from the converter/inverter to the necessary dc voltage levels. There is transformer isolation from the wall power. A detailed description of the operation is given in the *SLT Power Supply,* FEMI, Form 223-2799.

DANGER
The input to the regulators from the converter/inverter is not isolated from the wall power, presenting a lethal potential to ground.

Two types of regulators are used in the CPU: the SCR-controlled output type and the magnetic-amplifier-controlled output type. The 18V dc ROS regulator (PS11) is of the first type, the others are of the second type.

The SCR-controlled output regulator utilizes the SCR's ability not to conduct until it is gated on to control the output voltage level. The duty cycle (time on versus time off in each half of the input ac cycle) of the SCR is determined by an adjustable voltage-sentitive feedback loop from the output terminals. The time that the SCR's are gated on determines the output voltage level (after averaging by the filters) at the immediate load current.

The magnetic-amplifier-controlled output regulator uses a bridge magnetic amplifier to control the output voltage level. The magnetic amplifier is a "square loop" toroidal core with three windings: gate, bias, and control (Diagram 6-2, FEMDM). The gate winding passes power to the output relative to the degree of saturation in the core. The core is set at a point on the slope of the saturation curve below saturation by the relative sum of the currents in the bias and control windings. The fixed current in the bias winding is counteracted by the variable out-of-phase current in the control winding. The amount of counteraction (and, therefore, the degree of saturation) determined by the two windings is adjusted according to the amount of error detected by the voltage feedback sensed from the output terminals. The remote sensing feature of these regulators is not used.

At the start of the gate cycle (the half of the input ac cycle determined by the diodes in the particular gate winding current path), the inductance of the gate winding is high (because the core is not saturated), causing a slow current rise. At some point in the gate cycle, the current in the gate winding rises (because the saturation level is rising with the additional current) to a point where the relative total of the currents in all three windings is sufficient to saturate the core. At saturation, the gate-winding current jumps to full current. When the current cycle passes through zero into the reset cycle, the saturation level returns to the point determined by the bias and control-winding currents.

The dc return levels of the regulators (logic or system ground) are made common with the wall power ground, the convenience outlet ground, the relay ground, and the frame at one point (frame bond bus). Voltage transients of high potential can exist in these other circuits at any time; usually these transients are caused by the switching of relays or contactors within the system. No ground circuit is completely without resistance. Therefore, if more than one low-resistance path were allowed between the logic ground circuits and the other ground circuits, transient currents could circulate in the "ground loop" thus formed, possibly high enough to induce erroneous switching of the semi-conductor components.

Marginal Adjustments

Several power supplies in the CPU and in the attached storage units and channels may have their output level varied from the nominal output. This feature allows critical circuits to be tested with nonstandard voltages as an aid in predicting failures.

When a power supply or attached unit is margined, an 'active margin' 24V dc level is generated. Each 24V dc level is fed to an OR to light the ACTIVE MARGIN indicator, showing the system has an active marginal adjustment. The MARGIN/METER SEL rotary switch may be turned to successive positions to locate the margined supply or unit. The LOCATE MARGIN indicator lights when the switch is at the position of a margined supply or unit. The voltage level is shown on the meter. (See logic YA081, YA082, and YA121 for the circuits.)

Within the CPU, four of the 6V power supplies (PS7, PS8, PS15, and PS16) and the 18V ROS supply (PS11) have their output levels adjusted by individual panel controls. These marginal controls have a cam and switch for the 24V dc 'active margin' level.

In the channels and 2365's, the margined supplies are adjusted by a motor drive controlled by the MARGIN CHANNEL/STOR lever switch and selected by the MARGIN/METER SEL switch. 28V ac through the raise and lower motor in each unit are returned through the rotary switch to select the unit and the lever switch to actuate the motor. The RAISE position causes the motor to adjust the marginal voltages to a higher level. The LOWER position adjusts the marginal voltages to a lower level. The 'active margin' 24V dc level is sent to the CPU by each unit with a margined supply.

POWER-ON SEQUENCE

- Close all CPU CB's.
- Move CPU READY/OFF switch to READY position.
- Close service CB.
- Depress CPU ON or POWER ON pushbutton after waiting 5 seconds.

DC power in the CPU may be brought up in two ways. It may be initiated at the CE panel if only CPU power-on is desired, or it may be initiated at the system control panel or at the 2150 Console if full system power is desired. Diagram 6-3, FEMDM, shows the primary power-on sequence; Diagram 6-4, FEMDM, shows the secondary power-on sequence; and Diagram 6-5, FEMDM, is a simplified diagram of the CPU power control circuits.

The primary power is sequenced on as follows:

1. Close all CB's: CB1, CB2, CB7, CB9 (60-Hz units only), CB11, and CB12 (also CB3–CB6 in converted units).
2. Move CPU READY/OFF switch on CE panel to READY.
3. Close service CB to apply main power to CB1. As a result, the following occurs:
   a. K4 and K28, the EPO relays, are picked by 24V dc through the EPO switch(es).
   b. PK1, the EPO contactor, is picked by 24V dc through K4 contacts.
   c. The 24V dc bus, source of all other 24V dc used in the CPU, is energized through PK1 contacts.
   d. If the stepping switch is not at the start position (position 26) when the 24V dc bus is energized, it is advanced to the start position by 24V dc through K32 and the start-interlock contacts. These contacts remain closed until the stepping switch reaches the start position. Refer to *SLT Power Supply,* FEMI, Form 223-2799, for a description of the stepping switch.
   e. Main power is applied to PK2, the CPU power contactor, through PK1 contacts and CB9. (CB9 is in 60-Hz units only.)
   f. K47 (K25 in converted units) is picked by 24V dc through the power-off pushbuttons, K12 contacts, CPU READY/OFF switch, and protection-relay contacts. K47 (K25) has a 5-second RC time delay before it transfers. Secondary power is usually sequenced on by depressing a POWER ON pushbutton (step 4) but is also sequenced on when the CPU ON pushbutton is depressed (step 5). Because of the delaying action of K47 (K25), at least 5 seconds must elapse before either pushbutton has any effect. (The 5-second delay ensures the delay of the RC turn-on circuits in the regulators. Otherwise, rapid alternations between automatic power-offs and attempted power-ons could cause overcurrents in the regulators.)
4. Depress POWER ON on system control panel or 2150 Console.
   a. The stepping switch drive coil is pulsed by 24V dc through stepping switch contacts A-26 to A-COM and the interrupter contacts; the stepping switch advances to position 1.
   b. K7 and K32 are picked through CR9; K6 through CR9 and CR8.
   c. K6, K7, and K32 are held by 24V dc through the power-off pushbuttons and K7 contacts.
   d. K12 is picked by 24V dc through K6 contacts.
   e. K5 is picked by 24V dc through the POWER ON pushbutton, stepping switch contacts C-1 to B-1, and K47 (K25) and K46 (K26) contacts.
   f. K2 is picked through CR11.
   g. K2 is held by 24V dc through K6 contacts, CPU READY/OFF switch, thermal and overcurrent sense relay contacts, CR7, and K2 contacts.

h. K5 is momentarily held by 24V dc through the same protection-relay contacts, CR10, and K5 and K46 (K26) contacts.
i. K46 (K26) is energized by 24V dc through K5 contacts and transfers after a 5-second RC time delay.
j. PK2 is picked by 24V dc through K5 contacts.
k. Main power is applied to the converter/inverter and the blowers through PK2 contacts.
l. The converter/inverter supplies 140V, 2500-Hz ac to the regulators.
m. As the regulators develop the dc power for the CPU logic, undervoltage sense relays K10 and K11 are picked.
n. K5 is now held by 24V dc through the same protection-relay contacts as before, CR10, and K5, K10, and K11 contacts.

Note: If K10 and K11 are not picked by the time K46 (K26) transfers, K5 is not held, thus dropping PK2 and K46 (K26).

o. K47 (K25) drops as K10 and K11 pick.
p. If PS9 is in the CPU, it supplies 48V dc to bus II BB 9 through relay II-K1 contacts. II-K1 is picked by 24V dc through K5, K10, and K11 contacts.
q. Momentary continuity through K35 contacts starts the power-on reset operation described in Chapter 6, Section 1. K35, which is shown in Diagram 6-6, A, FEMDM, is in series with CR15, a 100-uf capacitor, and a set of normally open contacts of K46 (K26) to the 24V dc bus. The initial charging current of the capacitor is sufficient to pick K35. As the charge on the capacitor builds up, the current becomes insufficient to hold K35, causing it to drop.
r. The stepping switch is advanced to position 2 by 24V dc through K10, K11, and stepping switch contacts A-1 to A-COM.

Full system power is achieved in a sequential and interlocked manner by the power control interface circuitry. After initiating power on in the CPU, the CPU stepping switch waits for CPU power-up to be confirmed, then advances to an attached stand-alone unit, directs it to bring its power up, waits for power-up to be confirmed, and advances to the next unit, repeating this procedure until power is up in all attached units. Each stand-alone unit has its own internal power-sequencing control. Diagram 6-7, FEMDM, shows a typical power control interface connection, using channel 1 as an example.

With the stepping switch at position 2, the system power-on sequence continues automatically.

s. 24V dc (supplied by channel 1) through K4, K6, and stepping switch contacts C-2 to B-2 provide channel 1 with a 'power pick' level. This level initiates internal power-on sequencing of channel 1. When channel 1 has completed power-on sequencing, 24V dc (from the CPU) are returned as a 'power complete' signal through stepping switch contacts A-2 to A-COM to advance stepping switch to position 3. K19 is picked and held at the same time.
t. Channel 2 is picked using position 3, K4, and K6 contacts. The 'power complete' signal advances the stepping switch to position 4 and picks and holds K21.
u. Channel 3 is picked using position 4, K4, and K6 contacts. The 'power complete' signal advances the stepping switch to position 5 and picks and holds K31.
v. The stepping switch is advanced to position 11 by 24V dc at stepping switch contacts A-5 through A-10 to A-COM.
w. Storage unit 1 is picked using position 11, K4, and K7 contacts. The 'power complete' signal advances the stepping switch to position 12 and picks and holds K29.
x. Storage unit 2 is picked using position 12, K4, and K7 contacts. The 'power complete' signal advances the stepping switch to position 13 and picks and holds K30.
y. Storage unit 3 is picked using position 13, K28, and K7 contacts. The 'power complete' signal advances the stepping switch to position 14 and picks and holds K20.
z. Storage unit 4 is picked using position 14, K28, and K7 contacts. The 'power complete' signal advances the stepping switch to position 15 and picks and holds K22.
aa. LCS unit 1 is picked using position 15, K28, and K32 contacts. The 'power complete' signal advances the stepping switch to position 16.
ab. LCS unit 2 is picked using position 16, K28, and K32 contacts. The 'power complete' signal advances the stepping switch to position 17.
ac. LCS unit 3 is picked using position 17, K28, and K32 contacts. The 'power complete' signal advances the stepping switch to position 18.
ad. LCS unit 4 is picked using position 18, K28, and K32 contacts. The 'power complete' signal advances the stepping switch to position 19.

Note: If any of these units are not attached to the CPU, a jumper must be installed to provide a simulated 'power complete' level return to advance the stepping switch to the next position. If any unit is attached to the CPU but is not

being used in the system, a LOCAL/REMOTE switch in the unit allows it to be bypassed in the sequencing. In LOCAL, power control is at the unit, except for EPO. In REMOTE, the unit is sequenced on by the stepping switch.

ae. In converted units, the stepping switch is advanced to position 26, the start position, by 24V dc at stepping switch contacts A-19 through A-25 to A-COM. This action completes the system power-on sequence of the converted units.

af. In original units, the stepping switch is advanced to position 25 by 24V dc at stepping switch contacts A-19 through A-24 to A-COM.

ag. K35 is momentarily picked through stepping switch contacts B-25 and C-25, CR16, and another 100-uf capacitor. In large system configurations, the 'power-on reset' signal follows the one issued when K46 was picked. This action completes the system power-on sequence of the original units.

5. Depress CPU ON pushbutton on CE panel.
   a. K5 is picked by 24V dc through K12 contacts, CPU ON pushbutton, and K47 contacts.
   b. K2 is picked through CR11.
   c. K2 is held by 24V dc through K12 contacts, CPU READY switch, thermal and overcurrent sense relay contacts, CR7, and K2 contacts.
   d. K5 is momentarily held by 24V dc through the same protection-relay contacts, CR10, and K5 and K46 contacts.
   e. The sequencing continues as described in steps 4i through 4q. (The holding 24V dc, however, is through K12 contacts rather than K6 contacts.)

POWER-OFF SEQUENCE

CPU power-off can be initiated manually or automatically. Secondary power in the CPU is dropped if the CPU READY/OFF switch is placed in OFF, if an automatic power-off sequence is initiated, or if a system POWER OFF pushbutton is depressed (as well as all other remotely controlled units). Primary and secondary power of all units is dropped if an EPO switch is pulled.

Normal Power-Off

- Depress POWER OFF to drop system secondary power.
- Move CPU READY/OFF switch to OFF to drop only CPU secondary power.

System secondary power-off is initiated at the system control panel or at the 2150 Console; the CPU and channel power goes off after the storage units to protect the stored data. CPU secondary power can be separately controlled at the CE panel; however, if any attached storage unit is on, data in storage may be lost. Diagram 6-8, FEMDM, shows the power-off sequence, which is as follows:

1. Depress POWER OFF on either system control panel or 2150 Console. As a result:
   a. The 24V dc holding line to K7 and K32 is opened, causing them to drop.
   b. The stepping switch is advanced to position 26 when the step relay driver is energized through the normally closed contacts of K32. (In converted units, the stepping switch is already in position 26.)
   c. The 'power hold' line to the storage units, through K7 and K32 contacts, is opened, causing them to cycle down unless they are shared with another CPU that is still on.
   d. As the last storage unit goes down, the 24V dc holding K6 drops, causing K6 to drop.
   e. The 24V dc holding line to K12, K5, and K2, through K6 contacts, is opened, causing them to drop.
   f. The 24V dc holding line to PK2, through K5 contacts, is opened, causing it to drop.
   g. Main power to the converter/inverter, through PK2 contacts, is opened, causing CPU dc to go off.
   h. AC power through PK2 contacts is opened to the blowers and to the 1052 Printer.
   i. The 'power hold' line to the channels, through K6 contacts, is opened, causing them to cycle down. (They do not cycle down if in local-control mode or if equipped with a Two-Channel Switch feature and the other interface has its power on.)
   j. K47 (K25 in converted units) is again picked through the path described in the power-on sequence, step 3f, allowing dc power to be turned on again.
2. Move CPU READY/OFF switch on CE panel to OFF.

Note: Under these conditions, data may be lost in the storage units as the CPU dc goes down.

   a. The 24V dc holding line to K5 and K2 is opened, causing them to drop.
   b. Same as steps 1f through 1h of the normal power-off sequence.
   c. If system power was up, it remains up because K6, K7, and K32 did not drop.
   d. The CPU dc power cannot be turned on again until K47 (K25) is again picked by returning the CPU READY/OFF switch to CPU READY and waiting 5 seconds. Both the system and CPU power-on pick paths are open when K47 (K25) is dropped.

Emergency Power-Off

- Pull any EMERGENCY PULL (EPO) switch.

Emergency power-off in the system is initiated by pulling the EMERGENCY PULL (EPO) switch at either the system control panel or at the 2150 Console, or by losing continuity through the Emergency Power-Off Control Feature (Diagram 6-7, FEMDM).

Note: All EMERGENCY PULL switches latch mechanically and must be reset by the CE.

For safety, the Emergency Power-Off Control Feature may be installed whenever more than one system is in an area or room; the feature is *always* installed when the systems are interconnected by other than communication lines. Feature No. 3621 is shared by two systems; feature No. 3622 can be shared by up to 12 systems. If the feature is added to the CPU, it is located in frame 4. Any EPO switch can initiate an emergency power-off in all systems sharing the feature. While correcting the power fault in one system, the power of the remaining system(s) can be returned by means of the NORMAL/BY-PASS switch located on the feature. The switch should be placed in BY-PASS by a CE only after the cause of the EPO event has been identified; the switch should be returned to NORMAL as soon as the cause has been corrected.

Diagram 6-7 shows the EPO circuits in the stand-alone units using the CPU and channel 1 as an example. Within the CPU, the following takes place during an emergency power-off:

1. The 24V holding line to K4, through the EPO switches and the Emergency Power-Off Control Feature, is opened, causing K4 to drop.
2. The 24V dc holding line to PK1, through K4 contacts, is opened, causing PK1 to drop.
3. Main power to the CPU, except to T1, through PK1 contacts, is opened, causing a complete power-off. T1 provides the low-voltage ac to a bridge rectifier for the dc to establish the EPO loop.
4. The EPO-hold line to the stand-alone units, through K4 or K28 contacts, is opened, causing them to drop complete power, even if they are in the local-control mode.

Automatic Power-Off

Secondary CPU power is sequenced down if an overcurrent (K8), overvoltage (also K8), undervoltage (K10 or K11), or thermal (K9, K14–K17, or K24) condition is detected. K8, K9, K14–K17, and K24 contacts are in series with the CPU READY/OFF switch; transfer of any of these relays causes a CPU power-off sequence similar to the sequence when the CPU READY/OFF switch is moved to OFF. Refer to the discussion of normal power-off, steps 2a through 2c. Because K47 (K25 in converted units) remains dropped, the cause of the power-off must first be corrected before CPU dc power can again be turned on.

The dropping of K10 or K11 results in a similar power-off sequence, except K2 remains held and K47 picks after a 5-second delay. Because K47 is picked, the CPU dc power may be turned on again; however, unless the faulty regulator voltage to the undervoltage sense circuit is corrected or turned off, the CPU power is again automatically removed.

Overcurrent Protection

The overcurrent sensing circuits are internal to each regulator. (Refer to *SLT Power Supply,* FEMI, Form 223-2799, for circuit details.) Any fault that draws excessive current from any regulator, except PS9, causes the CPU to drop power.

Figure 5-4, A, shows the overcurrent sense loop for original units. When an overcurrent condition exists, continuity is provided between terminals 8 and 9. 24V dc, through auxiliary switch contacts on CB2, CB7, or CB8 in 60-Hz units (or CB2, CB3, and CB4 in 50-Hz units) or through terminals 8 and 9 of the regulators except PS9, pick K8, the overcurrent sense relay.

Figure 5-4, B, shows the overcurrent sense loop for converted units. When an overcurrent condition exists, continuity between terminals 8 and 9 is broken. 24V dc, through auxiliary switch contacts on CB2 through CB8 and through terminals 8 and 9 of the regulators except PS9, drop K8, the overcurrent sense relay.

Terminals 8 and 9 of PS9 are not in the overcurrent sense loop because PS9 is used only by the 1052 Adapter feature. 24V dc, through terminals 8 and 9 of PS9, pick K18, the 48V power-check relay. (K18 is dropped in converted units.)

Overvoltage Protection

- Excessively high regulator output voltage causes CPU dc power to drop via overcurrent sense loop.
- Original units have overcurrent/overvoltage SMS card in each regulator.
- Converted units have positive and negative overcurrent circuits external to regulators.

Any fault that raises the output voltage level of any regulator except PS9 above a preset level causes the CPU to drop power via the overcurrent sense loop. (Because PS9 is not required for CPU operation, an overvoltage condition sensed in that regulator only lights power fault indicators.) An SCR, in series with a low resistance, is connected across the output terminals of each regulator except PS9. When gated on, the SCR shorts the regulator output through the low resistance, causing an overcurrent condition within the regulator. The internal overcurrent protection circuit initiates a CPU power-off sequence.

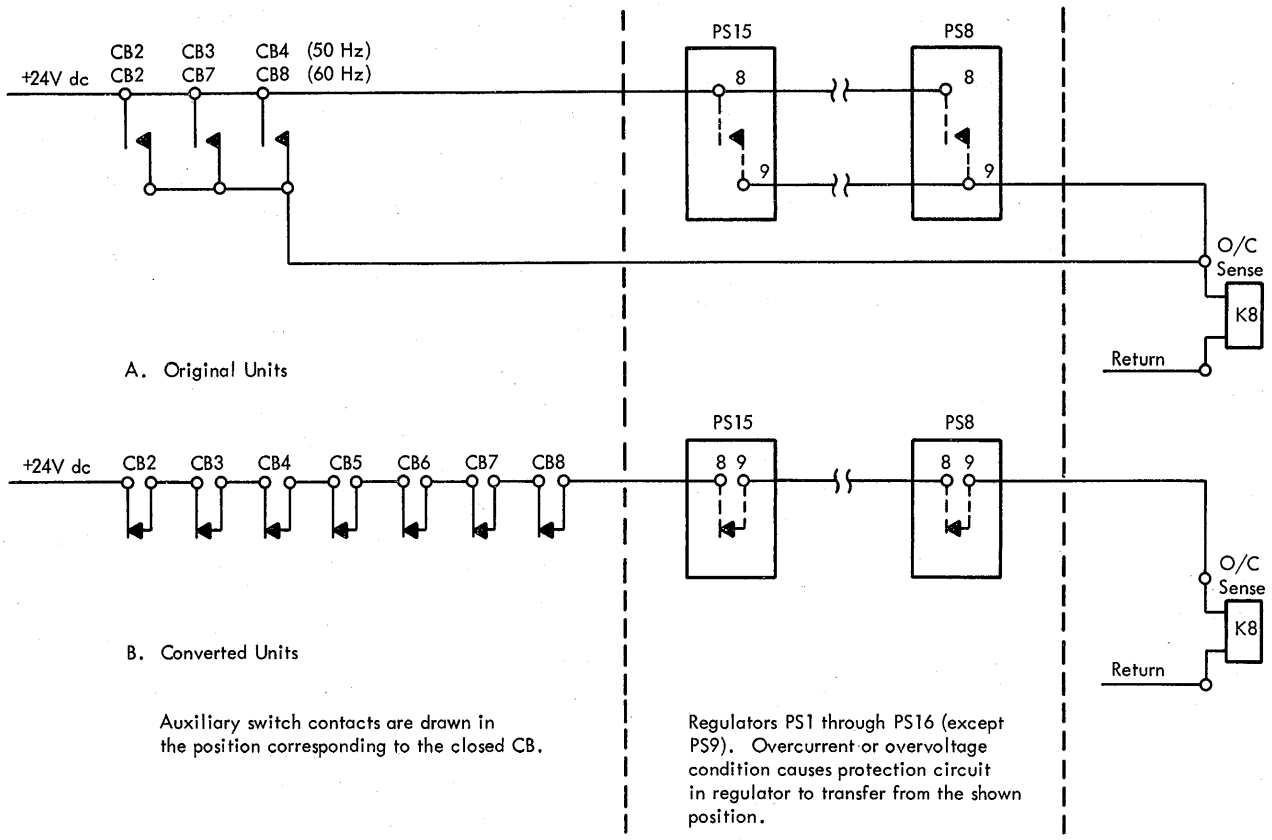


Figure 5-4. Overcurrent Protection Loop

The original units have separate overvoltage protection circuits for each regulator. An external assembly mounted on the output bus senses the output voltage and provides the low-resistance path through an SCR; the internal overcurrent/overvoltage SMS card provides the gating voltage to turn on the SCR. The overvoltage protection circuits in converted units have some logic in common with all the regulators (see logic YA091). Also, the overvoltage protection circuits in converted units are completely external to the regulators. Their operation is discussed in the following paragraphs.

Positive Regulators, Converted Units. A line from each positive regulator except PS9 is fed through a metering jack to the emitter of a sensing transistor. The base of this transistor is held at a preset upper limit voltage by Zener diode Z1 and potentiometer R4 (3V supplies) or R9 (6V supplies). The collector series load resistors are connected to ground or 0V. When its emitter is held negative with respect to its base by the regulator, the transistor is biased off, holding the collector circuit and the SCR gate at 0V. When the regulator output rises above the upper limit, causing the emitter to become positive with respect to the base, the transistor starts to conduct. The current drawn through the collector load resistors raises the collector circuit and the SCR gate to a positive level with respect to the regulator output level, gating the SCR on.

Negative Regulators, Converted Units. A line from each negative regulator except PS11 provides, through a metering jack, the return path for the collector circuit of an SCR gating transistor and the emitter of a sensing transistor. The base of the sensing transistor is held at a preset negative voltage limit by Z2 and R7. The collector load resistor is connected to ground or 0V. The base of the SCR gating transistor is fed by a resistor from the collector of the sensing transistor. The emitter of the SCR gating transistor is connected to ground or 0V. The sensing transistor, when its emitter is held positive with respect to its base by the regulator, is biased off, and its collector circuit and the base of the SCR gating transistor are held at 0V. As a result, the SCR gating transistor is biased off, and its collector circuit and the SCR gate are held at the regulator output level. When the regulator output level increases beyond the negative voltage limit, causing the emitter of the sensing transistor to become negative with respect to its base, the transistor starts to conduct. The current drawn through the collector load resistor lowers its collector circuit and the base of the SCR gating transistor to a negative level. With its base made negative with respect to its emitter, the SCR gating transistor starts to conduct. The current drawn through the collector load resistor raises the collector circuit and the SCR gate to a positive level with respect to the regulator output level; the SCR is gated on.

Regulator PS11 is sensed in the same way as the other negative supplies, except that the emitter voltage of the sensing transistor is established through an adjustable voltage-dividing network in the line from the regulator. This dividing network permits the sensing transistor to use the same voltage reference (Z2 and R7) as the other negative supplies.

Undervoltage Protection

The logic voltage sensing circuits provide an indication that all CPU dc is above minimum voltage levels (2.4V). Any fault that lowers the output voltage level of any regulator except PS9 below the minimum causes CPU power to drop. (See logic YA111 for the sensing circuits.)

A line from each positive regulator except PS9 is fed through isolation switches, labeled UNDER VOLTAGE CHECK, to an AND. The output level of the AND is determined by the lowest input level; that is, the one nearest zero or ground. Thus, if any supply not isolated is off, the output level of the AND is near zero. However, if all the supplies are on, the output level of the AND is about 3V. The output of the AND is transferred directly to the base of transistor Q1, which shares an emitter load resistor with Q2. The base of Q2 is held at 2.4V by Zener diode CR59 and resistor R8. Q2 now carries the full emitter load current and holds the common emitter circuit to 2.4V. With its emitter held positive with respect to its base, Q1 is biased off. As the output level of the AND raises the base of Q1 to 2.4V, Q1 begins to conduct and starts to share with Q2 the current drawn by the emitter load resistor. As the output level of the AND raises the base of Q1 above 2.4V, Q1 raises the emitter circuit above 2.4V. With its emitter raised to a level positive with respect to its base, Q2 is biased off. Q1 now carries the full emitter load current, which is sufficient to pick K10.

A similar circuit, of opposite polarity, checks the negative regulators and picks K11.

The UNDER VOLTAGE CHECK switches permit CPU dc power to be brought up for servicing purposes, with a low output-voltage level from a regulator.

CAUTION

Under these conditions, CPU logic may be damaged by the nonstandard voltages. Refer to the ***2065 Processing Unit*** FEMM, Form Y27-2039-2, Chapter 5, for undervoltage troubleshooting.

Thermal Protection

- Thermal relays are picked during power-on sequence.
- Overtemperature condition drops thermal relay, causing CPU power to drop and indicators to light.
- THERMAL RESET pushbutton on CE panel must be depressed before CPU power can be restored.

Thermal protection is provided by sensing elements (thermostats) in the return path of each thermal relay. If an overtemperature condition exists, the return path is opened, dropping the associated relay (causing CPU dc to drop) and lighting an indicator. The thermostats are at the top of the following areas: logic gate A (K9), gate B (K14), gates C and D (K15), gate E (K16), high-frequency regulator modules (K17), and the converter/inverter (K24). Logic YA022 shows the relay and indicator circuits.

The thermal relays are automatically reset when the 24V dc bus is energized by PK1 contacts, and are manually reset when the THERMAL RESET pushbutton on the CE panel is depressed.

Relay K3, which provides the 24V dc pulse to reset the thermal relays, is in series with a 1500-uf capacitor and a set of normally closed contacts of K46 (K26 in converted units) and the 24V dc bus. The initial charging current of the capacitor is sufficient to pick K3. As the charge on the capacitor builds up, the current becomes insufficient to hold K3, causing it to drop. [When K46 (K26) transfers, the capacitor is discharged through a 12-ohm resistor. Relay K46 (K26) transfers about 5 seconds after CPU ON or POWER ON are depressed.] The manual path for picking K3 is through the THERMAL RESET pushbutton to the 24V dc bus.

24V dc, through the transferred contacts of K3 and through diodes CR1–CR6, cause the thermal relays to pick, providing normal temperature conditions exist (all thermostats closed). Each thermal relay holds through its own contacts as K3 drops. An open thermostat causes the relay to drop, and its contacts light the associated indicator on the CE panel.

INDICATORS

Indicators on the system control panel show an incomplete power-up status in the CPU or in the attached HSS units and channels. The CPU and the units are individually indicated, as is the system status. (See logic YA026 and YA083, and Diagram 8-1, FEMDM.) The system status is shown by the POWER ON pushbutton. This pushbutton is normally white but glows red when system power is incomplete. This condition is also indicated on the 2150 Console.

System Power-On Indicator

24V dc, through the CPU voltage-sense relay contacts (K10 and K11) and the HSS units and channels power-complete relay contacts (K19–K22 and K29–K31), light the System Power-On indicator (clear or

white lamp backlighting the POWER ON pushbutton). If any of these relays is not picked, the indicator remains off and the dual lighted POWER ON pushbutton glows red as described below.

Power Check Indicators

There are nine power check indicators:

1. CPU POWER CHECK. 24V dc through K2 contacts are fed through parallel-connected contacts on voltage-sense relays K10 and K11 to the CPU POWER CHECK indicator. If either of these two relays is not picked after K2 is picked, this indicator lights. 24V dc are also fed through parallel-connected contacts on overcurrent relay K8 and thermal-sense relays K9, K14–17, and K24 to the CPU POWER CHECK indicator. 24V dc, through 48V power check relay contacts K18 or through the CPU READY/OFF switch in OFF, also light this indicator.
2. STOR FRAME 1 POWER CHECK. 24V dc, via K32 contacts, are fed through K29 contacts to this indicator. If K29 is not picked after K32 is picked, this indicator lights.
3. STOR FRAME 2 POWER CHECK. Similarly, through K30 contacts, this indicator lights.
4. STOR FRAME 3 POWER CHECK. Similarly, through K20 contacts, this indicator lights.
5. STOR FRAME 4 POWER CHECK. Similarly, through K22 contacts, this indicator lights.
6. CHAN FRAME 1 POWER CHECK. Similarly, through K19 contacts, this indicator lights.
7. CHAN FRAME 2 POWER CHECK. Similarly, through K21 contacts, this indicator lights.
8. CHAN FRAME 3 POWER CHECK. Similarly, through K31 contacts, this indicator lights.
9. System Power Check. A line from the CPU POWER CHECK indicator and a line from each unit power check indicator are combined at an OR. The OR turns on the System Power Check indicator. The System Power Check indicator also lights when any of the unit power check indicators is on. Because the bulb of the System Power Check indicator is red and backlights the translucent POWER ON pushbutton, the pushbutton glows red.

1052 PRINTER-KEYBOARD POWER

The CPU contains the +48V dc power supply (PS9) and the audible alarm used by the 1052 Adapter feature if a 2150 Console is not attached to the system. The power circuits added to the CPU for the 1052 Printer-Keyboard and the audible alarm are discussed here and are shown in Diagram 6-9, FEMDM. The manner of sequencing on the +48V dc has been described in step 4p of the power-on sequence.

Audible Alarm

The audible alarm provides the operator with a signal when directed by the program. It may be used, for example, to signal a program hangup or the end of a program.

The 48V dc through relay A-K2 have a return path through the 1052 Adapter logic. When logic grounds the return path, A-K2 is picked. A-K2 is then held by A-K1 and A-K2 contacts. 28V ac, through A-K1 and A-K2 contacts, starts the bell ringing. The 24V dc, through A-K2 contacts, picks A-K1. A-K1 has a short delay before it transfers; A-K2 drops and the bell stops ringing as A-K1 transfers.

Dual 1052 Power Interface

The CPU provides +48V dc for the keyboard of the 1052 via one connector and +48V dc and ac power to the printer via another connector. When a second 1052 is added, a Dual 1052 Adapter junction box is plugged into this second connector. The adapter box has two sets of connectors, one set for the 1052 on the right side of the system console, the other set for the 1052 on the left side. Four switches on the junction box allow the power in any cable to be turned off so that a faulty 1052 may be removed for maintenance without removing CPU power.

DIRECT CONTROL POWER

- Spurious direct control signals are avoided during normal CPU power transitions.

Two precautions are taken to ensure that spurious signals are not issued via the direct control feature during normal CPU power transitions: (1) the stepping switch is prevented from generating a 'power-on-reset' signal until K46 (K26 in converted units) has picked, and (2) signal line drivers with gated power are used. Diagram 6-6, B, FEMDM, shows the relay logic.

24V dc go through K46 normally-open contacts to position 25 contacts of the stepping switch, ensuring that the direct control circuits are reset after all regulators have reached their nominal voltage levels. Also, K47 (K25 in converted units) is again energized by 24V dc through the same K46 and stepping-switch position 25 contacts, CR17, K5 contacts, and CR18. After the 5-second transfer delay of K47, the power and logic of the system is quiescent. K50 is then picked through K47 contacts, turning on the +6V dc to the G-register and timing-signal line drivers. Until then, no direct control signals may be issued to the external device. (If only CPU power is turned on, the stepping switch remains in position 26. In this case, only one power-on reset occurs; K46 is picked by 24V dc through K46, K7, and K5 contacts and CR18.)

Although K50 picks last during a power-on sequence, it drops first during a normal power-off sequence. If a

system POWER OFF pushbutton is depressed, the stepping switch is immediately advanced to position 26, removing the 24V dc holding K50; if the CPU READY/OFF switch is placed in OFF, K5 immediately drops, removing the 24V dc holding K50. With K50 de-energized, the direct control interface line drivers are grounded, disabling them from transmitting a spurious signal.

USAGE METERS AND KEY SWITCH

- Process (left) meter records time customer used system.
- System maintenance time is recorded on CE (right) meter by key-switch control.

Two meters on the system control panel show the system running time while it is processing customer data (process meter) and while it is being operated by the CE (CE meter). (The process meter is also known as the cluster, central processing complex, system, and customer meter.) A key-operated switch selects the meter to be driven. The normal position of this switch allows power to be applied to the process meter. The CE, using a key, switches to the CE meter. The key cannot be removed when the switch is in the CE position.

The meter selected by the key switch records time as long as the 'pass pulse' trigger is set if (1) not in the Wait state and the 'manual' trigger is not set or (2) a 'meter in' signal is received from the channels. The selected meter also runs momentarily when any pushbutton on the system control panel is depressed (except STOP while already in the manual mode). The 'pass pulse' trigger is set by a start, load, or reset operation and, when in single-cycle-mode operation, is reset after gating one pulse (refer to "Clock Control and Signal Distribution", Chapter 2, Section 1). In general, the recording of time stops while a CPU machine check condition exists; however, the recording continues if, during an IPL operation, the CPU CHECK switch is in the DSBL position when the check condition occurs or if, during any other operation, the STOP ON STORAGE CHECK switch is depressed (activated) when the check condition occurs.

The multiplexed 'meter in' signal from the channels is received whenever an I/O operation initiated by the CPU is still in progress and requires recording. The CPU sends two signals to the channels. The 'CPU clock not stopped' signal is sent unless the CPU is in the Wait state or manual mode or is not recording time due to a CPU machine check. The 'meter out' signal is sent whenever time is being recorded on the process meter. These two signals are used by the channels and their attached units in running their own usage meters and, in turn, in generating the 'meter in' signal.

Diagram 8-28, FEMDM, shows the CPU metering circuits. The selected meter runs for a minimum of 400 ms. Relay RR1 on the usage meter card is picked by a signal from the CPU logic. 40V ac through RR1 contacts drive the meter.

www.IBM360.info

Maintenance aids available to the CE for the 2065 CPU fall into two categories: (1) those used for error detection during normal operation, such as error detection logic and interruptions (discussed in Chapter 1, Section 3), and (2) those aids used for diagnosing the cause of failures and for preventive maintenance. This second category includes:

1. System control panel: Contains the controls necessary for initiating any operation, for manual testing, and for performing various maintenance tasks. In addition, indicators allow monitoring of the CPU operation by displaying the status of important registers and control triggers.
2. CE panel: Contains the controls necessary for initiating power on and off sequences. In addition, indicators show the power status.
3. Diagnose instruction: Allows certain diagnostic functions to be performed on the CPU or the channel. It is used in conjunction with the maintenance control word (MCW) to allow such diagnostic functions as reversing parity and suppressing data checks. In addition, it can be used to initiate the logout and FLT functions.
4. MCW: Used in conjunction with the Diagnose instruction, FLT's, and ROS tests.
5. Logout, ROS tests, and FLT's: Logout stores the status of the console indicators into fixed positions of main storage when a trouble symptom occurs; the data logged out may be subsequently recalled for analysis although the status of the indicated logic is changed from what it was when the symptom appeared. ROS tests check each bit position of every ROS word. FLT's check the CPU at the logic-block level.
6. Ripple tests: Provide the capability of (1) storing data from the DATA switches into all addresses in LS or main storage and inserting 1's with correct parity into the storage-protect keys, and (2) reading out all locations of LS or main storage and displaying the data.
7. Diagnostic programs: Check the CPU on a functional basis by programming the CPU to perform one or more instructions or sets of instructions.
8. Marginal checking: Allows operation of critical circuits with nonstandard voltages to detect if any are approaching failure.

The maintenance aids provide the CE with a wide choice of troubleshooting techniques when isolating a fault. Thus, the CE can tailor his troubleshooting procedure to suit the particular problem and his own experience.

The maintenance aids have interrelated functions, dependent upon the troubleshooting technique used. For example, scan logic, which provides the control necessary to perform ROS tests and FLT's, is also used in logout, a system control panel function. On the other hand, switches on the system control panel are used to initiate all maintenance programs (i.e., FLT's, ROS tests, and diagnostics).

This chapter is divided into two sections. Section 1, Console Controls, discusses the manual controls and indicators on the system control panel and on the CE panel, and their application. Section 2, Maintenance Features, discusses the Diagnose instruction, MCW's, logout, ROS tests, FLT's, ripple tests, diagnostic programs, and marginal checking.

Section 1. Console Controls

SYSTEM CONTROL PANEL

The system control panel, in addition to its main function as the operating and monitoring center of the system, is one of the prime maintenance aids available to the CE. Using this panel, a CE can duplicate many program operations or portions of operations manually and can repeatedly exercise portions of the machine logic at a normal or a reduced rate of operation.

Some of the maintenance routines that can be performed from the system control panel include storage ripple, marginal checking, and frequency bias. Other controls allow the CE to stop the CPU at the end of the current instruction, to display main storage or LS, to store into main storage or LS, and to log out indicator status to fixed positions in main storage. (The latter is a function of the scan logic and is described in Section 2 of this chapter.)

The system control panel is divided into seven separate panels (A to G), as shown in Diagram 8-1, FEMDM. The operator control section (panel G) is identical to those used on the other models of the System/360, thus providing the operators with compatibility between models. If the 2150 Console or the 2250-1 Display Unit is included in the Model 65 System, the operator control section and the EMERGENCY PULL switch may be duplicated on one of these units to provide monitor control operation.

The functions of the controls and indicators, by panel, on the system control panel are defined briefly in the following listing:

PANEL A

1. DC voltmeter. Indicates the voltage levels of the marginable supplies. The particular supply indicated is determined by the MARGIN/METER SEL switch.
2. MARGIN/METER SEL switch. Has 12 positions to select the power supply to be indicated by the meter and to determine which of the attached stand-alone units may be marginally checked:
   a. STORE FRAME 1: Selects HSS unit 1.
   b. STORE FRAME 2: Selects HSS unit 2.
   c. STORE FRAME 3: Selects HSS unit 3.
   d. STORE FRAME 4: Selects HSS unit 4.
   e. CHAN FRAME 1: Selects channel 1.
   f. CHAN FRAME 2: Selects channel 2.
   g. CHAN FRAME 3: Selects channel 3.
   h. ROS LOCATE: Selects the 18V ROS bias power supply (gate D in the CPU).
   i. CPU A: Selects gate A in the CPU.
   j. CPU B: Selects gate B in the CPU.
   k. CPU C: Selects gate C in the CPU.
   l. CPU E: Selects gate E in the CPU.

   Note: When an Additional Storage Attachment feature is installed in CPU2, STORE FRAME 1, 2, 3, and 4 select HSS units 5, 6, 7, and 8, respectively.

3. MARGIN indicators:
   a. ACTIVE: Indicates that an internal power supply or an attached HSS unit or channel is being marginally checked.
   b. LOCATE: Indicates when the MARGIN/METER SEL switch is set to the position corresponding to the margined power supply, attached HSS unit, or channel.
4. POWER CHECK indicators. These eight indicators (CPU, STOR FRAME 1, 2, 3, and 4, and CHAN FRAME 1, 2, and 3) indicate an incomplete power-up status in the CPU, HSS units 1, 2, 3, and 4, and channels 1, 2, and 3, respectively.

   Note: When an Additional Storage Attachment feature is installed in CPU2, STOR FRAME 1, 2, 3 and 4 indicate an incomplete powerup status in HSS units 5, 6, 7, and 8, respectively.

5. MARGIN CHANNEL/STOR switch. Applies power to a motor in the channel or HSS unit selected by the MARGIN/METER SEL switch to lower or raise the output voltage levels from the marginable supplies in that unit or channel. The channel or storage unit must be in remote operation.

PANEL B

The five potentiometers on this panel (ROS, +6M A GT, +6M B GT, +6M C GT, and +6M E GT) raise or lower the output voltage levels from the 18V ROS and the 6V gate A, B, C, and E supplies, respectively.

PANEL C

The pull switch on this panel, EMERGENCY PULL, when pulled, initiates emergency power-off in the system. The switch latches in the out position and must be manually restored by the CE.

PANEL D

This panel is blank in the basic 2065. When an Additional Storage Attachment feature is installed, a four-position STORAGE INDICATE rotary switch is added:

1. STOR 1–4: Allows the contents of the storage address registers for HSS units 1 and 2 to be displayed on the roller switches.
2. STOR 5–8: Indicates HSS units 3 and 4.
3. STOR 9–12: Indicates HSS units 5 and 6.
4. STOR 13–16: Indicates HSS units 7 and 8.

PANEL E

1. Roller switches and indicators. This section of the panel contains six six-position roller switches, with 36 indicators (implicitly numbered 0–35) associated with each roller switch. A roll chart associated with each roller switch rotates with the roller switch to correspond with the roller position, thus showing the information being displayed for each indicator. Diagram 8-2, FEMDM, identifies the indicators for the six positions of each roller switch. These indicators are tested between positions of the switch. Position 6 of roller 6 is used to test the remaining indicators on the system control panel and on the 2150 Console.
2. DATA 0–31 and DATA 32–63 switches. These 64 switches, in hexadecimal groups, permit data to be entered manually. Correct parity is automatically generated by contacts on the switches.
3. ADDRESS switches. These 24 switches, in hexadecimal groups, select an addressable location in ROS, LS, or main storage. Correct parity is automatically generated by contacts on the switches.
4. STOR CHK (storage check) indicator: Indicates an error in the storage units.
5. PROC CHK (processor check) indicator: Indicates an error in the CPU.

PANEL F

1. TEST MODE, REPEAT switch: Repeats the ROS test or FLT in main storage continuously.

2. TEST MODE, ROS/PROC/FLT switch. This switch has three positions:
   a. ROS: Places the CPU into ROS test mode and removes program control.
   b. PROC (process) – normal position for CPU operation.
   c. FLT: Places the CPU in the FLT mode and removes program control.
3. STORAGE INDICATE switch. This switch has two positions:
   a. STOR 1–4 - normal position: Allows the contents of the storage address registers for HSS units 1 and 2 to be displayed on the roller switches.
   b. STOR 5–8: When in this position, the operation is the same as in the normal position for all Model 65 configurations except Models IH and J. For these models, the contents of the storage address registers for HSS units 3 and 4 are displayed on the roller switches.

Note: This switch is spare if an Additional Storage Attachment feature is installed.

4. FREQUENCY ALTERATION switch: Decreases the CPU clock period from 200 ns to 195 ns. Operates only with the CE key switch in the CE position.
5. DEFEAT INTERLEAVING switch. This switch has three positions:
   a. PROC (process) – normal position: Addressing is interleaved with no even/odd storage area reversal, unless changed by the Diagnose instruction.
   b. REV (reverse): Interleaving is disabled and the even and odd storage areas are reversed. No reversal occurs on Model G65.
   c. NO REV (no reverse): Interleaving is disabled and no even/odd storage area reversal takes place.
6. STOP ON STORAGE CHECK switch: Inhibits operation of and maintains the environment of the main storage basic storage module in which an error was detected. Other basic storage modules are not affected.
7. DISABLE INTERVAL TIMER switch: Prevents the interval timer from being decremented.

   Note: When the Multisystem feature is installed, disabling the interval timer when operating in the multisystem mode also disables the detection of CPU inactivity.

8. STORAGE SELECT switch. This switch has three positions:
   a. MAIN – normal position: Selects main storage for manually storing or displaying data.
   b. LOCAL: Selects LS for manually storing or displaying data.
   c. MAIN BYTE: Same as MAIN except that the byte selected by ADDRESS switches 21–23 is the only byte affected by a manual store operation.
9. ADDRESS COMPARE STOP switch: Stops processing if the main storage address being requested agrees with bits 2 through 20 of the ADDRESS switches.
10. CPU CHECK switch. This switch has three positions:
    a. PROC (process) – normal position: If the PSW machine check mask [PSW(13)] is a 1, the CPU stops on detection of a CPU check and the status is logged into main storage. A machine check interruption then takes place. If the mask is a 0, the result is the same as if the switch is in the DSBL position.
    b. DSBL (disable): The CPU does not stop on detection of a machine check, but the check trigger is set. No logout or interruption takes place.
    c. STOP: The CPU stops on detection of a machine check, but there is no logout of data. The check trigger is set.
11. PULSE MODE switch. This switch has three positions:
    a. PROC (process) – normal position: Does not affect CPU operation.
    b. COUNT: Provides a means of looping through a selected number of machine cycles (maximum of 2047). The number of cycles is entered into DATA switches 53–63. Each loop starts at the address contained in main storage location 0. The RATE switch must be in the PROCESS position.
    c. TIME: Provides looping when the interval timer is decremented. Each loop starts at the address contained in main storage location 0. The RATE switch must be in the PROCESS position.
12. REPEAT INSN (instruction) switch. This switch has three positions:
    a. PROC (process) – normal position: Does not affect CPU operation.
    b. SINGLE: Allows the first instruction in the DATA switches to be repeated continuously.
    c. MPLE (multiple): Allows continuous looping through the four instruction halfwords in the DATA switches.
13. REPEAT ROS ADDRESS switch: Continuously reads out the ROS address specified by ADDRESS switches 0–11. ROS TRANSFER must be depressed to start this loop.
14. DISABLE DIRECT CONTROL switch. Causes Write Direct and Read Direct instructions to become invalid instructions.

Note: The DISABLE DIRECT CONTROL switch is added by the Direct Control feature. Refer to Chapter 4 if the Multisystem feature is also installed.

15. RATE switch. This switch has four positions:
    a. INSN STEP (instruction step): CPU executes one machine instruction for each time START is depressed.
    b. PROCESS: Does not affect CPU operation; CPU operates at normal clock speed.
    c. SINGLE CYCLE: CPU advances by its minimum clock amount for each depression of START; all CPU operations are the same as for the PROCESS position.
    d. SINGLE CYCLE STORAGE INHIBIT: Same as SINGLE CYCLE without storage references.
16. SYSTEM RESET pushbutton: Resets main storage check triggers, on-line channels and control units, and CPU controls and check triggers to their initial state.
17. CHECK RESET pushbutton: Resets all CPU check triggers and latches to the non-error state.
18. PSW RESTART pushbutton: Loads a new PSW from main storage location 0 and starts processing if the RATE switch is in PROCESS.
19. ROS TRANSFER pushbutton: Serves to display the contents of any ROS location or to begin processing from any ROS address.
20. SET IC (instruction counter) pushbutton: Enters an address from the ADDRESS switches into bits 40–63 of the active (current) PSW. The data referenced will be loaded into Q, the first instruction will be transferred to R, and the IC will be updated by +8 bytes.
21. STORE pushbutton: Enters data from the DATA switches into the storage location specified by the STORAGE SELECT and ADDRESS switches.
22. DISPLAY pushbutton: Displays data from LS or main storage specified by the STORAGE SELECT and ADDRESS switches.

    Note: If the Multisystem feature is installed and prefixing is active when the SET IC, STORE, or DISPLAY pushbutton is depressed, the main storage address set into the ADDRESS switches is relocated.

23. START pushbutton: Starts the CPU operating in the mode selected by the RATE switch.
24. STOP pushbutton: Terminates CPU operation without changing the environment.
25. RESTART FLT I/O pushbutton: Backspaces one tape record and starts reading during ROS test or FLT operations.
26. LOG OUT pushbutton: Stores CPU status into fixed locations in main storage when the CPU is in the Stopped state.
27. Usage meters and CE Key switch: The usage meters indicate elapsed CPU running time: the Process (left) meter shows elapsed customer usage time; the CE (right) meter shows elapsed system maintenance time. The CE Key switch, when turned to the right, disables the Process meter and enables the CE meter and the FREQUENCY ALTERATION switch.

PANEL G

1. POWER ON pushbutton: Initiates power-on sequence in the CPU and the system units. The pushbutton is back-lighted by two indicator lights. When system power is on, the pushbutton glows white; when system power is off or not correctly on, the pushbutton glows red.
2. POWER OFF pushbutton: Initiates power-off sequence in the CPU and the system units.
3. LOAD UNIT switches: These three switches select the I/O unit to be used by a load operation.
4. INTERRUPT pushbutton: Causes an external interruption in the system and sets bit 25 of the interruption code to a 1.
5. SYSTEM indicator: Indicates that a CPU usage meter is running.
6. MANUAL indicator: Indicates the CPU is in the Stopped state.
7. WAIT indicator: Indicates the CPU is in the Wait state.
8. TEST indicator: Indicates that a switch on panel F is not in the normal operating position or that a channel is in the test mode.
9. LOAD indicator: Indicates a CPU load operation. A successful load turns this indicator off.
10. LOAD pushbutton: Resets the system and starts a load operation.

In the following paragraphs, the system control panel is presented in two parts: (1) manual controls, which discusses the function of each control on panels E, F, and G, and describes any interrelationship between the controls; and (2) indicators, which lists the conditions under which each indicator is turned on. Each control and indicator is located by an alphameric designation, referenced to the system control panel co-ordinates. For example, the SYSTEM RESET pushbutton is located at P30; the roller switches are located at D–J, 13–49.

Manual Controls

Manual operations are initiated and controlled by switches and pushbuttons. Pushbutton operation is synchronized with CPU clock signals as shown in Diagram 8-3, FEMDM. Depressing any of the pushbuttons (or performing a power-on reset) generates 'switch SS', 'delayed SS', and 'short SS' signals, which are used during manual control functions.

Functionally, manual controls consist of special logic and ROS microprograms. The CPU must be in the Stopped state before most manual operations can be initiated. The CPU is placed in the Stopped state by:

1. Performing a power-on-reset operation.
2. Depressing SYSTEM RESET (P30).
3. Depressing STOP (S30).
4. Detecting an address-compare condition when the ADDRESS COMPARE STOP switch (N17) is in the stop position.
5. Initiating an instruction-step-mode operation.
6. Initiating a scan operation.

When the CPU is in the Stopped state, it is in a ROS microprogram loop (stop loop) and the MANUAL indicator (S46) is lit.

Note: When the Multisystem feature is installed, the external reset operation generates the pushbutton single-shot signals and simulates depression of SYSTEM RESET, thus placing the CPU into the Stopped state. The external start operation also momentarily places the CPU into the Stopped state.

Stop Loop

- ROS microprogram loop.
- 'Stop' and 'manual' triggers control stop loop.
- Address of next instruction is in D.
- Interruptions and I/O operations are completed before entering stop loop.
- Six pushbuttons are sampled by stop loop: STORE, DISPLAY, SET IC, START, ROS TRANSFER, PSW RESTART.

When the CPU is in the Stopped state, a ROS microprogram, called the "stop loop", is continuously executed. The stop loop is shown by the heavy lines in Diagram 8-4, FEMDM; Stop loop pushbutton gating is shown in Diagram 8-5, FEMDM.

When in the stop loop, no program instructions are executed, the interval timer [location 80, decimal (50, hex), of main storage] is not stepped, and the usage meters are stopped unless a channel is running. Also, the MANUAL indicator (S46) is lit, and the stop loop determines the starting location of the next instruction to be executed and transfers the result to D (contents of IC minus 8 or 16). The contents of D are displayed in roller switch 1, position 2; the contents of the IC are displayed in roller switch 6, position 3.

Two triggers control the stop loop: the 'stop' trigger and the 'manual' trigger; Diagram 8-6, FEMDM. The 'stop' trigger, which forces the CPU to the stop loop, is set by:

1. Depressing STOP (S30).
2. Setting the 'instruction step' trigger in conjunction with an 'I-Fetch reset' micro-order or a 'reset interrupt triggers' micro-order.
3. Detecting an address-compare condition when the ADDRESS COMPARE STOP switch (N17) is in the stop position.
4. Being in scan operation.

Note: When the Multisystem feature is installed, the 'stop' trigger is also set at the initiation of an external start operation.

At end op, the 'stop' trigger's being set indicates an exceptional condition. A branch is forced to a count delay microprogram, provided all pending interruptions and I/O operations are completed. (All interruptions not masked off and all I/O operations are completed before entering the count delay and stop loop microprograms.) The 'stop' trigger forces address 026 (hex) into ROSAR to enter the count delay microprogram, which allows time to recognize that a pushbutton has been depressed before the stop loop is entered.

After the count delay microprogram is executed, the 'manual' trigger is set by the 'set stop loop trigger' micro-order (Diagram 8-6, FEMDM), and the stop loop is entered. The 'manual' trigger allows manual operations only when the CPU is in the Stopped state; however, the SYSTEM RESET (P30), CHECK RESET (P35), and LOAD (S51) pushbuttons operate in all modes. The 'manual' trigger also makes the RATE switch (Q25) inoperative. Whenever the CPU is in the stop loop or in one of the microprogram routines entered from the stop loop, the CPU operates at normal machine speed regardless of the position of the RATE switch.

The 'stop' trigger is not set during a system reset or power-on reset; therefore, the stop loop is entered directly (Diagram 8-4).

The stop loop continually samples six pushbuttons: STORE (R32), DISPLAY (R35), SET IC (R30), START (S25), ROS TRANSFER (Q35), and PSW RESTART (Q30). When the stop loop senses that one of these pushbuttons has been depressed, a branch is made to a new microprogram to perform the pushbutton function (Diagram 8-5). The microprograms for the six pushbuttons are shown in Diagram 8-4. The microprogram just executed either branches to the count delay microprogram and re-enters the stop loop or continues other preselected operations.

Note: When the Multisystem feature is installed, the stop loop also senses the state of the 'external start' latch, branching to the PSW-restart microprogram when it is set.

Power-On Reset

Depressing POWER ON (N43) initiates the power-on sequence. After power is applied to the system, a

system-reset occurs. The 'power-on reset' signal resets the system to an initial state and generates the 'switch SS', 'delayed SS', and 'short SS' signals. A ROS address of 00B (hex) is forced to execute the power-on reset microprogram to clear all LS locations (Diagram 8-7); main storage locations remain unchanged. The stop loop microprogram is then entered.

The 'system reset' signal performs the same reset functions as the SYSTEM RESET pushbutton (P30) (Diagram 8-7). After the system is reset, the ROS microprogram is entered by setting ROSAR(8,10,11) (address 00B, hex). LS is reset, and the stop loop is then entered.

SYSTEM RESET Pushbutton

Activating the SYSTEM RESET pushbutton (P30) resets all main storage check triggers, on-line channels and control units, and CPU controls and check triggers to their initial state. Data-flow registers remain unchanged. A system reset does not affect equipment in off-line channel operation. The SYSTEM RESET pushbutton is active in all modes of operation. When it is depressed, the system is reset to an initial state and ROSAR (10,11) is set (address 003). The CPU enters the stop loop. Because a system reset may occur in the middle of an operation, the contents of the PSW and of result registers or storage locations are unpredictable. If the CPU is in the pulse mode, the manual controls are not reset when a system reset is initiated (Diagram 8-7).

The lines used during a system reset are also activated for a power-on reset and IPL reset. Depressing LOAD (S51) initiates a system reset before entering the IPL microprogram. This system reset, however, is not propagated to the channels. During the IPL, the channel latches the 'initial program load' signal from the CPU, and thus initiates a channel reset and a control unit and I/O device reset.

A system reset micro-order is provided to reset the CPU when in the scan mode. The micro-order resets all check conditions but does not reset the channels or storage units and does not force a ROS address.

CHECK RESET Pushbutton

The CHECK RESET pushbutton (P35) serves to reset all CPU check triggers and latches to the nonerror state (Diagram 8-7). All logic check indicators on the system control panel are turned off. The CHECK RESET pushbutton is active in all modes of operation; depressing it does not change the mode of operation. The operation continues as though no error conditions existed; the results, however, may be unpredictable.

STOP Pushbutton

The STOP pushbutton (S30) terminates machine operations while retaining the machine environment. The CPU proceeds to the end of the instruction being executed at the time STOP is depressed; all I/O operations in process and all pending interruptions not masked off are completed before entering the stop loop. The operator may continue normal program operation by depressing START (S25), or he may execute certain manual operations (e.g., instruction-step operation).

If the ROS microprogram is not performing an end-op micro-order, depressing STOP sets the 'stop' trigger (Diagram 8-6). At end-op, an exceptional condition is signaled to I-Fetch. The 'EXCEP' micro-order and the 'stop' trigger set ROSAR(6,9,10) (address 026 hex); the count-delay microprogram is executed and the stop loop is entered.

LOAD Pushbutton (IPL)

- LOAD pushbutton is active in all modes of operation.
- Loads program from preselected I/O unit per LOAD UNIT switches.
- 24 bytes of data are read automatically into main storage locations 0, 8, 10 (hex).
- CPU initiates channel operation, which loads program and passes control back to CPU for execution of program.
- Channel does not release CPU until program is loaded and channel and I/O unit addresses have been stored into bits 21–31 of IPL PSW in main storage location 0.
- CPU starts execution of program by loading doubleword from main storage location 0 as current PSW.

The purpose of the initial program load (IPL) operation is to allow the channel to load a new program, or a series of programs, into main storage from a preselected I/O unit and to initiate the execution of the program by the CPU. The basic function of the IPL operation is to read in from a specified I/O unit and to store into main storage 24 bytes of data (three doublewords) which will control the operation while the desired programs are being loaded into main storage. (The IPL procedure is common to all models of the IBM System/360.) The first doubleword, starting at address 0, is the IPL PSW, which is used by the CPU to initiate the program once it has been loaded (Figure 6-1). The next two doublewords, starting at addresses 8 and 10 (hex), are the IPL CCW1 and IPL CCW2, respectively, which control the channel during the IPL operation.

The IPL operation is initiated manually from the system control panel. The input channel and I/O unit are selected by the LOAD UNIT switches (Q, 43–51). The left switch, numbered 0 through 7, selects 1 of 7 allowable channels (channels 0–6). The other two switches, which are numbered 0 through F (hex), are set to the unit address of the selected I/O unit. When the selected I/O unit is ready, depression of LOAD (S51)

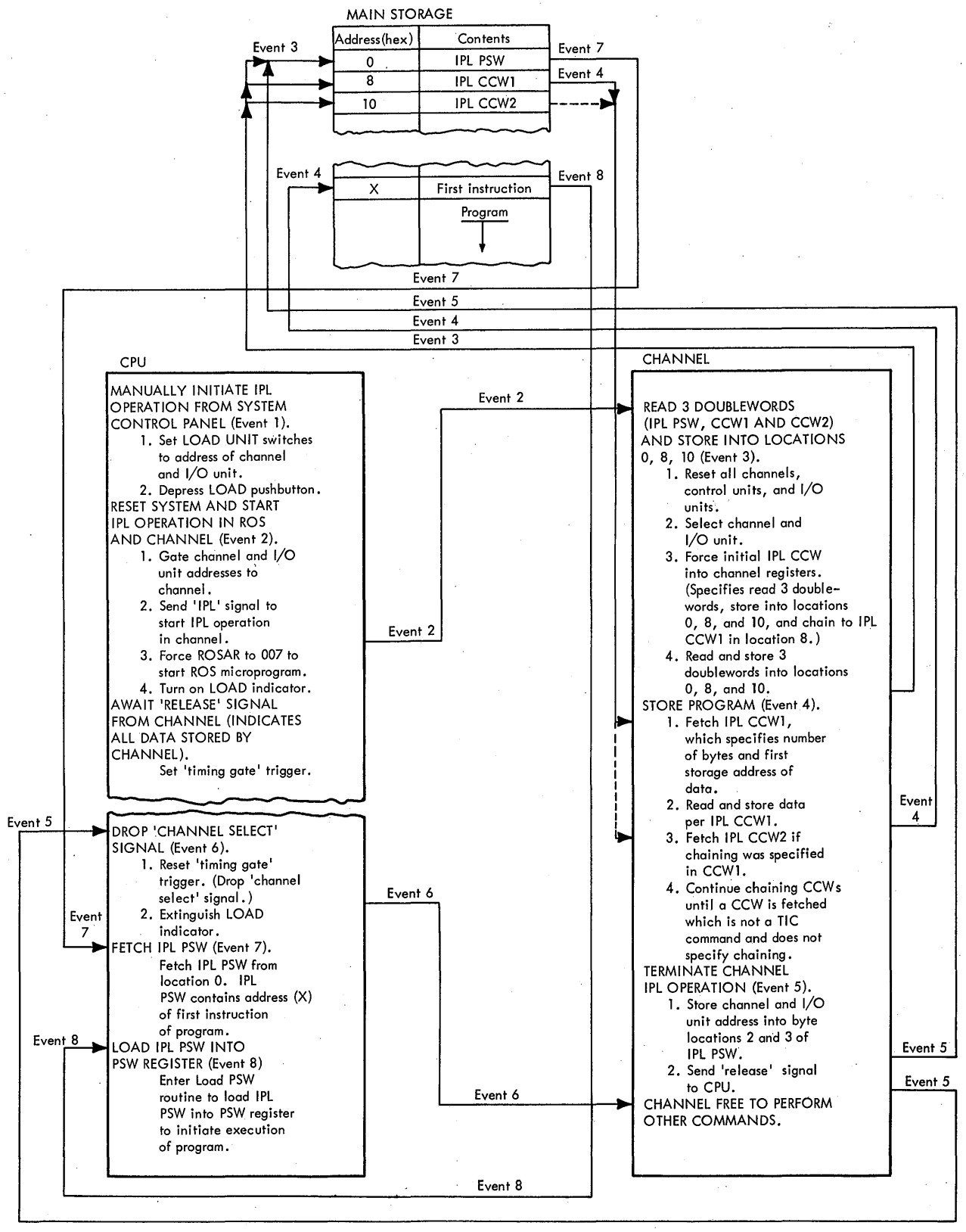


Figure 6-1. Normal IPL Operation

initiates the IPL operation. The LOAD pushbutton causes a system reset, turns on the LOAD indicator (S49), turns off the MANUAL indicator (S46) (if on), selects the channel, sends an eight-bit (+ P) unit address to the channel, sends an 'IPL' signal to the channel, and causes the CPU to enter the IPL microprogram. Diagram 8-8, FEMDM, is a flow chart of the IPL operation.

The 3-usec IPL 'system reset' signal resets the CPU by suspending all instruction processing, interruptions, and timer updating. The contents of LS remain unchanged. The IPL 'system reset' signal is not propagated to the channel. Instead, the channel receives the 'IPL' signal from the CPU, thus initiating a channel reset and dropping the 'operational out' signal to all on-line nonshared control units for 6 usec. Thus, all control units and their I/O units are reset.

After the system reset, ROSAR(9–11) is set, thus forcing a ROS address of 007 (Diagram 8-9, FEMDM). ROSAR(9,10) is set directly from the LOAD pushbutton. ROSAR(11) is set by the 'force address' trigger, which is set by the LOAD pushbutton. The 'timing gate' trigger is then set and the channel is tested for a 'release' signal. The LOAD pushbutton also sends an 'IPL' signal to the channel. The 'IPL status' trigger and the 'hold maintenance console IPL' latch gate the channel address and the I/O unit address to the channel. The selected channel gates the unit address to its unit address register and selects the specified I/O unit. The channel forces a hardware-generated initial CCW for a Read command, with a byte count of 24, and sets a 'chain command' bit in its flag register. The 'CCW valid' trigger is set, a command address of 8 (the address of IPL CCW1 in main storage) is set, and the data address is set to 0. This initial CCW indicates that 24 bytes of data are to be read into locations 0, 8, and 10 (hex) of main storage and that chaining will take place to IPL CCW1.

After the I/O unit has been selected, the IPL-forced CCW controls the transfer of the first three doublewords of data into main storage locations 0, 8, and 10 (hex); it then initiates a chain operation to IPL CCW1 at location 8. The data just stored governs further operation. IPL CCW1 may specify any valid operation for the selected I/O unit, and may be chained to IPL CCW2. Normally, IPL CCW1 initiates the read-in of another series of CCW's, then chains to IPL CCW2, which is a 'transfer in channel' (TIC) command, placing the channel under control of the new CCW's just read in under control of IPL CCW1. When the load-routine command chain is completed, the entire program has been loaded from the I/O unit and stored into main storage; the CPU can thus start executing the program just loaded.

The CPU, which is awaiting a 'release' signal from the channel, requires the identity of the channel and I/O unit involved in the IPL operation. When the first three doublewords of data were stored, the IPL PSW was stored into location 0. Because the interruption code of the IPL PSW (bits 21–31) has no other function, the channel stores the channel and I/O unit address into the IPL PSW interruption code, and sets bits 16–20 to all 0's as part of the IPL termination. (Bits 40–63 contain the starting instruction address of the program.) A 'release' signal is then sent to the CPU as the last function of the channel IPL routine.

During the channel operation, the 'select channel' signal is active. Normally, the channel time-out circuits detect the absence of a 'release' signal to the CPU after about 100 ms, and the channel and I/O interface are reset. During an IPL operation, however, the time-out circuits are deactivated because the channel must hold the CPU long enough to load the complete program; channel detection of interruptions is also blocked. (For a detailed analysis of the channel portion of an IPL operation, refer to the applicable channel FETOM.)

When the 'release' signal is sent to the CPU, the 'timing gate' trigger is reset and the 'channel select' signal is deactivated. The CPU starts an "IPL load PSW" routine, which loads the CPU PSW-register with the doubleword stored at main storage location 0 (IPL PSW). The CPU and the channel now resume normal operation. The CPU initiates the program just loaded; the instruction address in the current PSW is the address of the first instruction of the new program.

When the IPL operation was started by depressing LOAD, the LOAD indicator was turned on. When the IPL operation is successfully completed, the LOAD indicator is turned off. If a check condition or a selection failure occurs during the IPL operation, or if the I/O unit specified is not ready or available, the CPU continues to await a 'release' signal from the channel and the LOAD indicator remains on.

If the program to be loaded is a ROS test, the TEST MODE, ROS/PROC/FLT switch (N3) is set to ROS, the CPU CHECK switch (N19) is set to DSBL (disable), and SYSTEM RESET (P30) is depressed before LOAD is depressed. This procedure causes the ROS test to be loaded similar to a normal IPL, except that logic hardware circuits, rather than the normal ROS microprogram, control the CPU operation. A ROS test sets the 'maintenance mode stop clock' trigger, which stops the CPU clock and prevents a ROS cycle (Diagram 8-8). Thus, the ROS test is loaded even though ROS may be failing. The operating procedure for the ROS test is given in ALD M8005.

If an FLT is to be performed, the TEST MODE, ROS/PROC/FLT switch is set to FLT, the CPU CHECK switch is set to DSBL, and SYSTEM RESET is depressed

before LOAD is depressed. This procedure causes the FLT to be loaded similar to a normal IPL, except that the setup operation for execution of the FLT is different (Diagram 8-8). The operating procedure for the FLT is given in ALD M8005.

DATA Switches

The 64 DATA switches (K, L, 14–49) (Diagram 8-1, Sheet 2 of 2) allow the operator to enter data manually into the system. They are alternately colored black and white in groups of four to facilitate entering hex data into the CPU. Each switch is a two-position toggle switch, with the up position equalling a 0 and the down position equalling a 1. When using the DATA switches, the selected microprogram first places 1's into ST. Then if a DATA switch equals a 1, the corresponding bit in ST is unchanged (remains set); if a DATA switch equals a 0, however, the corresponding bit is reset to 0.

The DATA switches are used during the following manual operations (DATA switch gating is shown in Figure 6-2):

1. Store.
2. Storage ripple store.
3. Repeat instruction.
4. Pulse mode count function (switches 53–63).

The DATA switches are used during the following programmed operations:

1. Diagnose, with MCW(8–19) = 5B7.
2. FLT test number search after a stop.

When data is entered into the CPU, correct parity is automatically generated. If the switches are altered during an operation, such as repeat instruction or storage ripple, an error will probably occur.

ADDRESS Switches

The 24 ADDRESS switches (M, 14–40) allow the operator to manually select any address in ROS, LS, or main storage. They are alternately colored black and white in groups of four to facilitate entering hex addresses into the CPU. Each is a two-position toggle switch, with the up position equalling a 0 and the down position equalling a 1. During manual operations, the selected manual microprogram places 1's into D. If an ADDRESS switch equals a 1, the corresponding bit in D is unchanged. If an ADDRESS switch equals a 0, however, the corresponding bit in D is reset.

The ADDRESS switches are used during the following manual operations (ADDRESS switch gating to D, ROSAR, and comparison circuits is shown in Figure 6-3):

1. Store.
2. Display.
3. Set IC.
4. ROS transfer.
5. ROS repeat address.
6. Main storage address-compare stop or sync.
7. ROS address-compare sync.

To address main storage or LS, the ADDRESS switches are used with the STORAGE SELECT switch (N15). ADDRESS switches 0–20 select a doubleword in main storage; ADDRESS switches 2–20 may be used in conjunction with the ADDRESS COMPARE STOP switch (N17) when selecting an address for an address-compare stop or sync. ADDRESS switches 19–23 select an LS address.

ADDRESS switches 0–11 select a ROS address to be used to obtain a ROS address sync (a sync pulse is generated when the ROS address and the ADDRESS switches agree), or contain a ROS address for use with the ROS TRANSFER pushbutton (Q35).

When an address is entered, correct parity is automatically generated when gated to the CPU.

ADDRESS COMPARE STOP Switch

The ADDRESS COMPARE STOP switch (N17) enables the operator to stop at a predetermined address. To do so, he enters the address into the ADDRESS switches and places the ADDRESS COMPARE STOP switch in the stop (down) position. When ADDRESS switches 2–20 match the address sent to the BCU, the 'address compare' trigger is set (address comparison is performed on doubleword boundaries). The CPU will stop at the end of the instruction in progress. The 'stop' trigger is set, and the count delay microprogram is entered; entry to the stop loop is identical with depressing STOP. The TEST indicator (S48) is lit whenever this switch is in the down position.

In the center or normal position, a sync pulse is generated whenever SAB(2–20) matches ADDRESS switches 2–20 (Figure 6-3).

STORAGE SELECT Switch

The STORAGE SELECT switch (N15) is a three-position toggle switch that selects LS or main storage. It is used in conjunction with the ADDRESS switches and the STORE (R32) or DISPLAY (R35) pushbutton. The positions and corresponding functions are:

1. MAIN position – Selects the doubleword main storage location specified by ADDRESS switches 0–20 for both storing and displaying data.
2. MAIN BYTE position – For storing, selects the doubleword location (per ADDRESS switches 0–20) and byte (per ADDRESS switches 21–23) within the doubleword. For displaying, selects the doubleword in main storage to be displayed (same as MAIN).

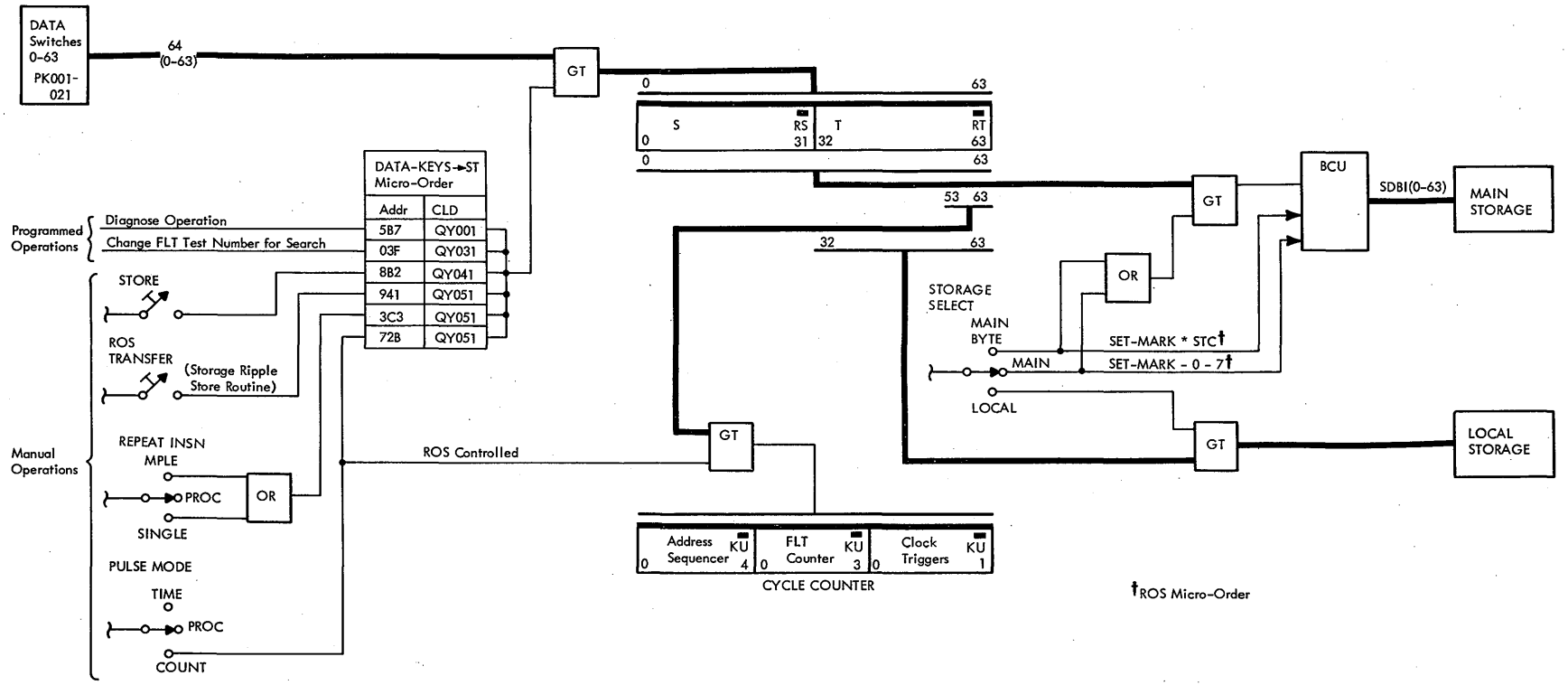


Figure 6-2. Data Switch Gating

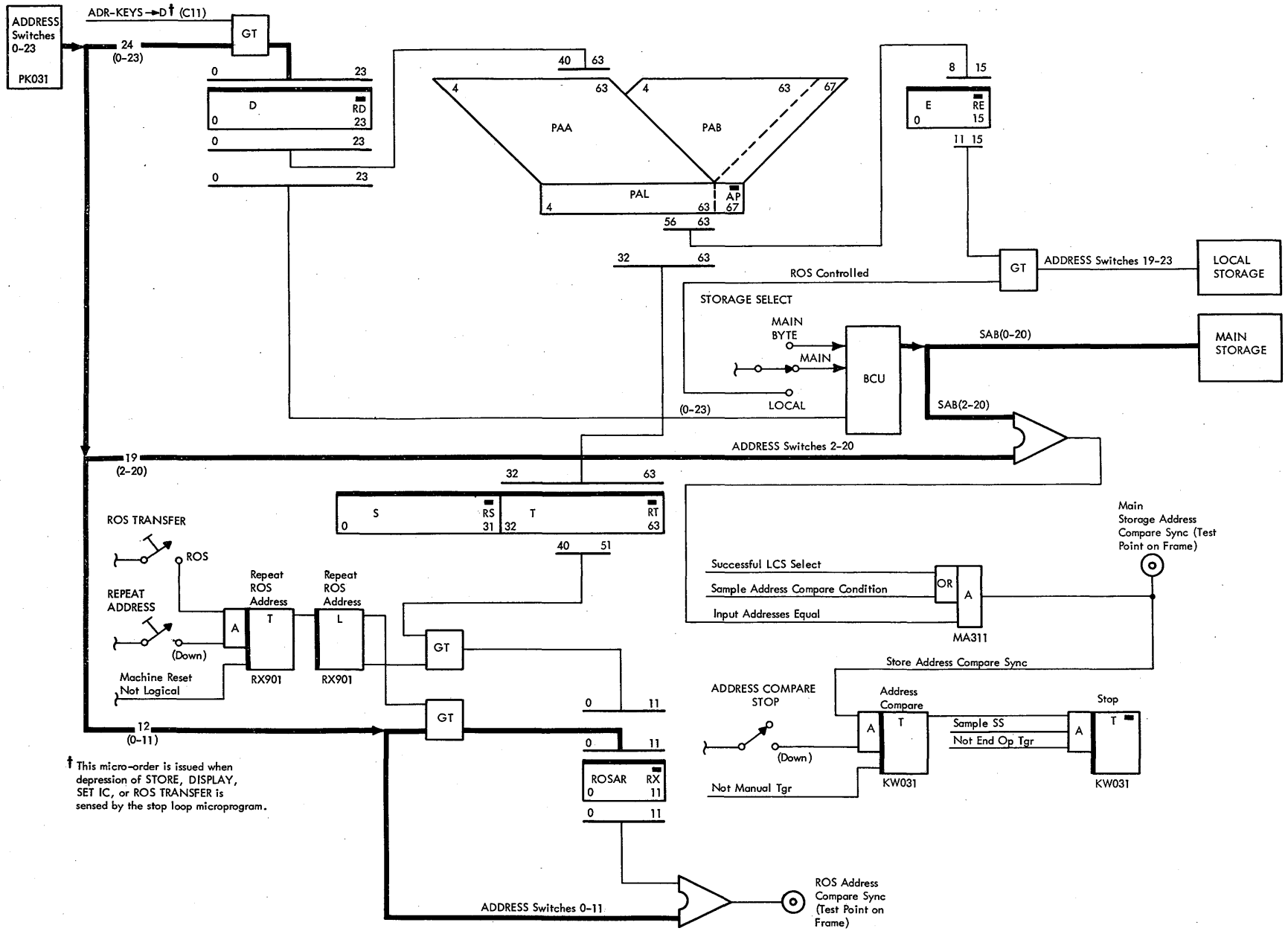


Figure 6-3. Address Switch Gating

3. LOCAL position – Selects an LS location (per ADDRESS switches 19–23) for both storing and displaying data.

The STORAGE SELECT switch conditions hardware so that a ROS branch may occur by setting or inhibiting ROSAR(11). The 'SB-PB' and 'LS-PB' micro-orders allow the stop loop microprogram to perform the storage selection specified by the STORAGE SELECT switch. STORAGE SELECT switch gating is shown in Diagram 8-10, FEMDM.

The use of the STORAGE SELECT switch is discussed in the following paragraphs whenever it is involved in a specific manual operation.

DEFEAT INTERLEAVING Switch

The DEFEAT INTERLEAVING switch (N10) permits the CE to choose which halves of main storage are the high-order and low-order portions for maintenance use. It is a three-position switch that performs the following functions (Diagram 8-11, FEMDM):

1. NO REV (up) position – Interleaving of main storage addresses is disabled, and locations in each storage unit are addressed consecutively.
2. REV (down) position – Interleaving of main storage addresses is disabled, and locations in each storage unit are addressed consecutively. Also, the high and low halves of each storage unit are reversed. No reversal occurs on Model G65.
3. PROC (center) position – Normal position of the switch. Addresses are interleaved in the normal manner with no reversal of storage addresses, unless changed by the Diagnose instruction.

By using this switch, the CE may address the first 16,383 doublewords consecutively rather than in an interleaved manner. When used for this purpose and the switch is in NO REV, the first 16,383 doubleword addresses to be selected will be even and the next 16,383 doubleword addresses will be odd. In REV, the first 16,383 doubleword addresses will be odd and the next 16,383 doubleword addresses will be even.

The signals generated by this switch are sent to the BCU and to main storage. When in NO REV or REV, the TEST indicator (S48) is lit.

STORAGE INDICATE Switch

The STORAGE INDICATE switch (N5) selects one of two pairs of storage units so that the contents of their storage address registers may be displayed on the roller switches. This switch has two positions:

1. STOR 1–4 (normal position). Storage address register contents for storage units 1 and 2 are displayed on position 5 of roller switches 1, 2, 5, and 6.
2. STOR 5–8. The operation is the same as for the normal position for all Model 65 configurations except Models IH and J. For these models, Storage address register contents for storage units 3 and 4 are displayed on position 5 of roller switches 1, 2, 5, and 6.

STOP ON STORAGE CHECK Switch

The STOP ON STORAGE CHECK switch (N11) inhibits further storage accesses in the main storage unit in which a storage error occurred so that the indicators will not be changed. The STOR CHK indicator (K52) lights to show that a storage error occurred and requires attention. The roller switches must be examined to determine the error and the address of the failing main storage word. The STOP ON STORAGE CHECK switch provides a storage stop upon encountering a storage check; this machine stop must not be confused with the Stopped state or the stop loop. The STOP ON STORAGE CHECK switch may be used in conjunction with the CPU CHECK switch (N19).

The 'stop on storage check switch' signal is sent to main storage, where the check signals are generated and sent back to the CPU. Storage checks caused by accesses by channels and by other CPU's also cause a stop. The TEST indicator (S48) is lit when this switch is not in the normal position.

SET IC Pushbutton

The SET IC pushbutton (R30) provides a means of entering an address into the current PSW. The pushbutton sets the instruction-address portion (bits 40–63) of the PSW [D(0–23)] to the value specified by the ADDRESS switches. The CPU is reset to the start of an I-Fetch at that address. Four instruction halfwords are loaded into Q per the address in D (contents of the ADDRESS switches); 8 is then added to D, and the sum is placed into the IC.

The first addressed instruction halfword is loaded into R per D(21,22). If D(21,22) = 11, Q is loaded with the next group of four instruction halfwords per the IC, and 8 is added to the IC (Diagram 8-4). If D(21,22) $\neq$ 11, Q is loaded only once.

After the instruction halfwords are fetched and loaded into Q and R, the count delay microprogram is entered. After the count delay, the stop loop is re-entered. Further manual intervention is required to start program execution. [Normally, START (S25) is depressed.] Note that the CPU must be in the Stopped state (stop loop) for this pushbutton to function. Note, too, that when the CPU is in the Stopped state, the instruction address is contained in D. After depressing the SET IC pushbutton, the new address contained in the IC is one or two doublewords more than the address contained in the ADDRESS switches.

RATE Switch

- Controls rate of instruction execution.
- CPU must be in stop loop before switch is activated.

The RATE switch (Q25) selects the rate at which instructions are to be executed. This rotary switch has four positions: PROCESS, INSN (instruction) STEP, SINGLE CYCLE, and SINGLE CYCLE STORAGE INHIBIT.

Two latches and two triggers control the RATE switch operation: 'instruction step' and 'single cycle' latches, and 'pass pulse' and 'block' triggers (Diagram 8-12, FEMDM).

The 'instruction step' latch performs two functions: (1) it allows setting the 'stop' trigger so that only one instruction is executed with each depression of START (S25); (2) it disables the stepping of the interval timer.

The 'single cycle' latch allows single-cycle operation. One machine cycle is allowed with each depression of START, unless the CPU requests additional machine cycles. The interval timer is disabled by the 'single cycle' latch.

When the 'pass pulse' trigger is set, CPU machine cycles are allowed. This trigger blocks the CPU machine cycles when in the single-cycle mode (RATE switch in SINGLE CYCLE or SINGLE CYCLE STORAGE INHIBIT position).

The 'block' trigger is used in single-cycle operation to allow one clock signal to be gated to the CPU each time START is depressed. The clock signal is blocked by resetting the 'pass pulse' trigger.

The TEST indicator is lit when the RATE switch is in any position other then PROCESS.

PROCESS Position. When the RATE switch is in PROCESS, the system operates at the normal clock speed of 200 ns; this is the position for normal program execution.

INSN STEP Position. The INSN STEP position allows the execution of one machine instruction for each depression of START; any instruction may be executed. All I/O operations and interruptions (not masked off) are executed after the instruction is completed. The CPU then enters the stop loop at the same point as the STOP pushbutton. The TEST indicator (S48) is lit when the RATE switch is in INSN STEP.

Instruction-step operation is shown in Diagram 8-13, FEMDM. The CPU must be in the stop loop before entering or leaving the instruction-step mode; the interval timer is disabled during the instruction-step mode.

When the RATE switch is moved to INSN STEP, the 'instruction step' latch is set if the CPU is in the stop loop or if the 'pass pulse' trigger is reset (Diagram 8-12). The interval timer is disabled after the 'instruction step' latch is set. No further action occurs until START (S25) is depressed, at which time one instruction is executed. At end-op, the 'stop' trigger is set by an 'I-Fetch reset' micro-order or a 'reset interrupt triggers' micro-order, thus forcing the CPU to enter the stop loop. The CPU remains in the stop loop until further action is taken by the operator.

SINGLE CYCLE Position. The SINGLE CYCLE position allows the CPU to advance one machine cycle (200 ns) each time START (S25) is depressed (Diagram 8-14, FEMDM). The CPU must be in the stop loop before entering or leaving the single-cycle mode and remains in the stop loop until START is depressed. The CPU begins executing instructions one machine cycle at a time for each depression of START. In Diagram 8-14, it is assumed that no CPU requests are generated; if an asynchronous device is used or if a storage request is given, however, more than one machine cycle is required. The single-cycle mode continues through all CPU functions of the instruction to the point of initiation of the asynchronous operation. The asynchronous operation begins on the next depression of START and runs to the completion point in a normal manner.

If the asynchronous device initiates an interruption request during single-cycle operation, the interruption is broken into single machine cycles. More than one depression of START is therefore required. The CPU runs at normal machine speed in the stop loop.

When the RATE switch is in SINGLE CYCLE, the 'single cycle' latch is set (Diagram 8-12). The CPU remains in the stop loop until START is depressed, at which time the CPU advances one machine cycle. The 'block' trigger is set as shown in Diagram 8-12. If there is no 'BCU hold on clock' signal, the 'pass pulse' trigger is reset after the 'block' trigger is set, thus inhibiting CPU clock signal distribution. START must be depressed for each CPU machine cycle advance. The TEST indicator (S48) is lit when the RATE switch is in SINGLE CYCLE.

SINGLE CYCLE STORAGE INHIBIT Position. The SINGLE CYCLE STORAGE INHIBIT position allows the CPU to advance one machine cycle (200 ns) each time START is depressed. All CPU requests are ignored, and asynchronous operations are suppressed.

Except for the inhibit signals sent to the BCU and inhibiting the 'stop 1' and 'stop 2' triggers from affecting the CPU clock, the single-cycle-storage-inhibit function is the same as the single-cycle function, Diagrams 8-12 and 8-14.

REPEAT INSN Switch

The REPEAT INSN (instruction) switch (N23) provides a means of repeating a single instruction or of repeating up to four instruction halfwords. The REPEAT INSN switch has three positions (Diagram 8-15, FEMDM): PROC, normal CPU operation; SINGLE; and MPLE (multiple).

A trigger-latch combination controls the repeat-instruction functions: 'repeat instruction adjust' trigger and 'repeat instruction initialization' latch.

The 'repeat instruction adjust' trigger forces a branch to the manual control repeat exceptional condition microprogram (ROS address 028, hex) at end-op of the start microprogram and sets the 'repeat instruction initialization' latch. STAT G is set to block re-entering the repeat-instruction microprogram at microprogram end-op. The trigger is set when the CPU is in the Stopped state and the REPEAT INSN switch is in SINGLE or MPLE; it is reset when the CPU is in the Stopped state and the REPEAT INSN switch is placed in PROC.

The 'repeat instruction initialization' latch blocks transfer to Q and stepping of the interval timer when in repeat-single-instruction mode. The latch is reset when in the stop loop and the RATE switch (Q25) is placed in PROCESS. The CPU must be in the Stopped state before entering or leaving the repeat-instruction mode. See Diagram 8-16, FEMDM, for the repeat-instruction operations.

The TEST indicator (S48) is lit when the REPEAT INSN switch is in any position other then PROC.

Repeat Single Instruction. When the REPEAT INSN switch is in SINGLE, one instruction is continuously executed. The instruction to be repeated is entered into the DATA switches, beginning with byte 0. If the CPU is in the stop loop, the 'repeat instruction adjust' trigger is set when the REPEAT INSN switch is placed in either MPLE or SINGLE. To begin the instruction, depress START (S25).

In the repeat-single-instruction mode, the repeat-instruction microprogram is executed to set up initial conditions before entering I-Fetch of the instruction to be executed (Diagram 8-16). The objectives of the microprogram are to load the contents of the DATA switches into Q, set IC(21,22) to 00, inhibit updating of IC(20) or above, gate the first instruction halfword from Q to R, set STAT G, and set the 'repeat instruction initialization' latch.

When the CPU is in the stop loop and the REPEAT INSN switch is in SINGLE, the 'repeat instruction adjust' trigger is set. The START pushbutton initiates the repeat-instruction function. A ROS address of 028 (hex) is forced into ROSAR to enter the repeat-instruction microprogram (Diagram 8-15). During this microprogram, STAT G is set which, in turn, sets the 'repeat instruction initialization' latch. This action prevents the loading of new instructions into Q. Note that STAT G is reset by the 'reset' micro-order that performs all the resets necessary before the next I-Fetch. The instruction that was loaded into Q from the DATA switches is executed.

Because the 'repeat instruction adjust' trigger was not reset during the initial setup routine, ROS address 028 (hex) is forced at end-op of the instruction to enter the repeat-instruction microprogram. Re-entering the repeat-instruction microprogram on each instruction resets IC(21,22) to 00, thus causing the first instruction to be repeated. Because the 'repeat instruction initialization' latch is set, Q is not loaded after the initial loading. The instruction is continuously executed until the CPU is manually stopped. The TEST indicator (S48) is lit while the REPEAT INSN switch is in SINGLE.

Repeat Multiple Instructions. When the REPEAT INSN switch is placed in MPLE, the four instruction halfwords loaded into Q are continuously executed per IC(21,22). The 'repeat instruction initialization' latch inhibits data from being transferred from the SDBO to Q. Once instruction execution begins, the repeat-instruction microprogram is not entered because the 'repeat instruction adjust' trigger is reset (Diagram 8-16).

The repeat-multiple-instructions function is similar in operation to the repeat-single-instruction function except for the following:

1. The interval timer is allowed to step.
2. The 'repeat instruction adjust' trigger is reset.
3. Interruptions are executed.

Resetting the 'repeat instruction adjust' trigger prevents re-entry to the repeat-instruction microprogram. Four instruction halfwords are continuously executed per IC(21,22), until the CPU is manually stopped. The TEST indicator (S48) is lit when the REPEAT INSN switch is in MPLE.

STORE Pushbutton

- Allows storing data into main storage or LS from DATA switches per STORAGE SELECT switch and ADDRESS switches.

The STORE pushbutton (R32) provides a means of storing information into any address of LS or main storage. The CPU must be in the Stopped state for this pushbutton to function.

The contents of the DATA switches are placed into the location specified by the ADDRESS switches and the STORAGE SELECT switch (N15) (Diagram 8-4). If the STORAGE SELECT switch is in LOCAL, the five low-order ADDRESS switches (19–23) specify the LS location into which the contents (32 bits plus 4 parity bits) of the right-half of the DATA switches (32–63) are to be stored. ADDRESS switch 19 in the 0 position permits storing into the general-purpose registers and in the 1 (down) position permits storing into the floating-point registers. ADDRESS switches 19 and 20, when set to 1's, address the working register.

If the STORAGE SELECT switch is in MAIN, the contents (64 data bits plus 8 parity bits) of the DATA switches are stored into main storage on a doubleword boundary per ADDRESS switches 0–20.

If the STORAGE SELECT switch is in MAIN BYTE, one byte is stored into main storage per ADDRESS switches 21–23. ADDRESS switches 0–20 specify the doubleword boundary in main storage. The value contained in the ADDRESS switches is placed into D for storing in main storage. The contents of the ADDRESS switches are placed into E (via D) for storing in LS.

For all store operations, the original contents of D, S, and T are destroyed. Correct parity is automatically generated before storing into either main storage or LS. After the data is stored, the microprogram enters the count delay routine and the CPU re-enters the stop loop (Diagram 8-4).

When STORE is depressed, the pushbutton signal is AND'ed with the 'STO-PB' micro-order. When the stop loop microprogram senses that STORE has been depressed, ROSAR(11) is set, causing entry into the store microprogram routine. The stop loop is re-entered after the store operation is executed.

DISPLAY Pushbutton

- Allows displaying of data from main storage into ST and AB or from LS into T per STORAGE SELECT switch and ADDRESS switches.

The DISPLAY pushbutton (R35) serves to display the contents of any location in LS or main storage. The CPU must be in the Stopped state for this pushbutton to function. The address and the storage to be used are determined by the ADDRESS switches and the position of the STORAGE SELECT switch (N15), respectively (Diagram 8-4). Data from main storage (64 data bits plus 8 parity bits) is displayed in ST and AB. (Set roller switches 1 and 2 to position 3 to display contents of ST, and set roller switches 3 and 4 to position 3 to display the contents of AB.) Data from LS (32 data bits plus 4 parity bits) is displayed in T. (Set roller switch 2 to position 3 to display contents of T.)

When DISPLAY is depressed, the pushbutton signal is AND'ed with the 'DIS-PB' micro-order. When the stop loop microprogram senses that DISPLAY has been depressed, ROSAR (11) is set, causing entry into the display microprogram. The original contents of S, T, and D are destroyed. After the selected data has been displayed, the count delay microprogram is executed and the stop loop is re-entered.

START Pushbutton

- Starts CPU processing.
- Initiates selected manual functions.

The START pushbutton (S25) serves to start the CPU in the process, instruction step, single-cycle, or single-cycle storage inhibit mode, depending on the position of the RATE switch (Q25). In Diagram 8-4, it is assumed that the CPU is in the process mode.

When START is depressed, a 'reset stop and manual triggers' micro-order resets the 'stop' and 'manual' triggers. An end-op occurs, and processing begins with I-Fetch of the next instruction. The 'interrupts' latch blocks interruptions at end-op of the start microprogram (Diagram 8-17, FEMDM). Therefore, interruptions are blocked until end-op of the first instruction. This latch does not block interruptions when the CPU is in the Wait state.

When START is depressed, the pushbutton signal is AND'ed with the 'STT-PB' micro-order. When the stop loop microprogram senses that START has been depressed, ROSAR(11) is set, causing entry to the start microprogram. Normal program execution continues.

If START is depressed after a normal halt, instruction processing continues as if no halt had occurred; pending interruptions are taken after execution of the first instruction. If START is depressed after an abnormal stop-loop entry or after a system reset, the results are unpredictable.

ROS TRANSFER Pushbutton

- ROS TRANSFER pushbutton allows ROS microprogram branch to any ROS location.

The ROS TRANSFER pushbutton (Q35) allows entry into a ROS word. Depressing ROS TRANSFER places the contents of the 12 high-order ADDRESS switches into ROSAR (Diagram 8-18, FEMDM). The next microinstruction is taken from ROS and placed into the ROS sense latches. Further action now depends upon the position of the RATE switch.

If the RATE switch (Q25) is in PROCESS, the CPU continues executing ROS words from the entry point. If the RATE switch is in INSN STEP, the CPU continues until an end-op is reached.

If the RATE switch is in SINGLE CYCLE or SINGLE CYCLE STORAGE INHIBIT, the CPU stops with the ROS word specified by the ADDRESS switches contained in the sense latches. (This ROS word may be displayed by means of the appropriate indicators.) Depressing START (S25) advances ROS one cycle, and the contents of the ROS data register may then be displayed. Further depressions of START advances ROS as in the single-cycle mode.

Regardless of the position of the RATE switch, checks may occur as a result of storage data bus transfer to registers and from the registers through the parallel adder. To prevent the CPU from stopping on these checks, place the CPU CHECK switch (N19) in DSBL. (See "CPU CHECK Switch".)

When ROS TRANSFER is depressed, the pushbutton signal is AND'ed with the 'ROS-PB' micro-order. When

the stop loop microprogram senses that ROS TRANSFER has been depressed, ROSAR(11) is set, causing entry into the ROS transfer microprogram. Instruction execution continues from the ROS address entered into the 12 high-order ADDRESS switches.

Storage-Ripple Microprogram

The storage-ripple microprogram (Diagram 8-19, FEMDM) is capable of (1) storing data from the DATA switches in all addresses in LS or main storage and putting correct parity into the storage-protect keys, or (2) reading all locations of LS or main storage and displaying the data. Main storage or LS is chosen by means of the STORAGE SELECT switch (N15).

If main storage is selected, the storage-ripple microprogram begins at address 0 and continues until an invalid address is detected, at which point a restart beginning at address 0 occurs. Diagram 8-19 shows the restart on an interruption request that resulted from detecting an invalid address. If LS is selected, the storage-ripple microprogram begins at address 0 and loops through all addresses in LS. Manual intervention (e.g., system reset or IPL) is required to exit from the storage-ripple microprogram.

The storage-ripple microprogram may also be used in troubleshooting by loading all storage locations with a predetermined value and then reading back the data. This microprogram may be used in conjunction with the STOP ON STORAGE CHECK switch (N11).

Storage-Ripple-Store Function. This function allows storing data into all locations of LS or main storage. To store data into LS or main storage, the CPU must be in the Stopped state. Enter the data to be stored into the DATA switches, and position the STORAGE SELECT switch (N15) to select LS or main storage. Enter 800006 (hex) into ADDRESS switches 0–23, and depress ROS TRANSFER. The data previously entered into the DATA switches is continually stored into all locations in the selected storage. Also, correct parity is placed into the storage protect keys on main storage ripple. Incorrect data may be stored if the DATA switches are changed when in the storage-ripple-store routine. To terminate the routine, depress SYSTEM RESET (P30). The ROS microprogram for the storage-ripple-store function is shown in Diagram 8-19.

Because there is no automatic means of clearing main storage and LS, the storage-ripple-store microprogram may be used for this purpose.

Storage-Ripple-Display Function. This function allows reading and displaying data from all locations in LS or main storage. The storage-ripple-display microprogram continuously reads all locations in LS or main storage as determined by the setting of the STORAGE SELECT switch (N15). The CPU must be in the Stopped state before ROS TRANSFER is depressed. To execute the storage-ripple-display routine, enter 800000 (hex) into the ADDRESS switches, select main storage or LS, and depress ROS TRANSFER (Diagram 8-19). If LS is selected by means of the STORAGE SELECT switch, the data is contained in S and PAL(32–63). If main storage is selected, the data is contained in AB and ST. The data may then be displayed using roller switches. Data is checked in PAL for correct parity.

To terminate the routine, depress SYSTEM RESET (P30).

REPEAT ROS ADDRESS Switch

The REPEAT ROS ADDRESS switch (N25) provides a means of continuously reading out a specified ROS address (Diagram 8-18). To do so, enter the address of the ROS word into the ADDRESS switches, activate the REPEAT ROS ADDRESS switch, and depress ROS TRANSFER (Q35). The same ROS address is accessed on each machine cycle. All storage requests are blocked. Changing the ADDRESS switches while in the repeat ROS loop may result in ROS parity checks.

The TEST indicator (S48) is lit when the REPEAT ROS ADDRESS switch is in the down position.

PSW RESTART Pushbutton and Wait State

- Allows loading new PSW from main storage location 0.
- If PSW(14) = 1, enter Wait state; if 0, enter Running state.

The PSW RESTART pushbutton (Q30) allows the operator to load a new PSW from main storage location 0 into the CPU. After the new PSW is fetched, the CPU continues processing if the RATE switch (Q25) is in PROCESS.

The 'stop' and 'manual' triggers are reset at the beginning of the PSW-restart microprogram (Diagram 8-4). The PSW RESTART pushbutton causes entry to the normal load PSW instruction routine, which refills Q, R, and E.

At every end-op, PSW(14) is tested. If PSW(14) = 1, the Wait state is entered (Diagram 8-20, FEMDM). [If PSW(14) = 0, the CPU is placed in the Running state.] The wait-state routine is depicted in Diagram 8-20, B, FEMDM. If the interval timer is to be stepped, the wait loop is re-entered after stepping the timer. If STOP (S30) is depressed, the stop loop is entered. When a restart from the stop loop is executed, the Wait state is re-entered if PSW(14) = 1.

An interruption causes a new PSW to be loaded into the CPU. PSW(14) is again tested, and the Wait state is re-entered if PSW(14) = 1; otherwise, the CPU is placed in the Running state. The usage meter is stopped when in the Wait state or stop loop unless a channel is operating.

When PSW RESTART is depressed, the pushbutton signal is AND'ed with the 'PSW-PB' micro-order. When the stop loop microprogram senses that PSW RESTART has been depressed, ROSAR(11) is set, causing entry to the PSW restart microprogram. A new PSW is loaded, and program execution continues.

DISABLE DIRECT CONTROL Switch

The DISABLE DIRECT CONTROL switch (N27) is added by the Direct Control feature. It is used to manually inhibit (partition) communications between the CPU and the external unit. When the switch is placed in the down position:

1. Write Direct and Read Direct instructions become invalid instructions, causing an operation program interruption after their I-Fetch.
2. Generation of 'direct control write out' and 'direct control read out' timing signals is inhibited.
3. The setting of the 'external interruption' latches for signals 2–7 is inhibited.

Note: When the Multisystem feature is installed, the functions of the DISABLE DIRECT CONTROL switch are altered when operating in the partitioned or multisystem mode. Refer to the discussion of this switch in the Multisystem feature write-up in Chapter 4.

DISABLE INTERVAL TIMER Switch

The DISABLE INTERVAL TIMER switch (N13), when placed in the down position, prevents the interval timer [main storage location 80, decimal (50, hex)] from being advanced (Diagram 8-21, FEMDM). In the center position, the timer is stepped at regular predetermined intervals.

In addition to the switch, the timer is disabled when the Diagnose instruction sets MCW(0) (disable timer bit) or when operating in the:

1. Stop-loop routine.
2. Single-cycle mode.
3. Instruction-step mode.
4. Repeat-instruction mode.
5. Scan mode.

The DISABLE INTERVAL TIMER switch is inactive when the PULSE MODE switch (N21) is in the TIME position and the 'pulse mode initialization' trigger is set. When this switch is in the down position, the TEST indicator (S48) is lit.

INTERRUPT Pushbutton

The INTERRUPT pushbutton (S43) initiates an external interruption by setting the 'console signal' trigger. If PSW(7) = 1, an interruption is taken after the current instruction, and interruptions of higher priority are completed. If PSW(7) = 0, the interruption request remains pending.

During the interruption, PSW(25) is made a 1, indicating that the INTERRUPT pushbutton is the source of the interruption. This pushbutton is effective while power is up for the system.

CPU CHECK Switch

The CPU CHECK switch (N19), a three-position toggle switch, is used to control the system when a machine check is encountered. The machine checks that set the error trigger and the logic controlled by the CPU CHECK switch are shown in Diagram 8-22, FEMDM. The TEST indicator (S48) is lit when this switch is in any position other than PROC.

When the CPU CHECK switch is in PROC and a machine check is detected, and PSW(13) = 1 (machine check mask bit), the CPU is stopped and the machine status is logged out to main storage. A machine check interruption is then executed. If PSW(13) = 0, the check is ignored, except that the error trigger is set, and no logout or interruption occurs.

When CPU CHECK is in STOP and a machine check is detected, the CPU clock is stopped and no logout occurs. The error trigger is set; the type of error may be determined by examining the roller switches. If CHECK RESET (P35) is depressed, operation is resumed, but the results may be unpredictable.

When CPU CHECK is in DSBL and a machine check is detected, the error trigger is set. Logout and interruptions do not occur, and the operation is not terminated. Program execution continues, ignoring machine checks. The error trigger may be reset by depressing CHECK RESET or SYSTEM RESET (P30).

PULSE MODE Switch

The PULSE MODE switch (N21) provides a means of looping through a number of machine cycles, starting at a selected address, or of looping each time the interval timer is advanced. The PULSE MODE switch has three positions: PROC (process), normal CPU operation; TIME; and COUNT.

The CPU must be in the stop loop before entering or leaving pulse mode operation. The PULSE MODE switch is inoperative during repeat-instruction mode.

Two triggers control pulse mode operation: 'pulse mode adjust' trigger and 'pulse mode initialization' trigger (Diagram 8-23, FEMDM). The 'pulse mode adjust' trigger determines when to force an overriding branch to the pulse-mode-initialization microprogram and when to reset the system. The 'pulse mode initialization' trigger is set by depressing START (S25) with the 'pulse mode adjust' trigger set. As a result, pulse mode operation begins.

The TEST indicator (S48) is lit when the PULSE MODE switch is in any position other then PROC.

PROC Position. The PROC position is used during normal program execution.

TIME Position. Highlights:

- Load program into main storage.
- Place starting address of program into main storage byte locations 5–7 by means of ADDRESS switches.
- Enter stop loop.
- Place PULSE MODE in TIME position.
- Depress START.

When the PULSE MODE switch is in the TIME position, instruction execution begins at the address specified in the address portion of the doubleword located in location 0 of main storage. Therefore, the starting address must be loaded (by means of the ADDRESS switches) into bits 40–63 of the doubleword located at location 0 before depressing START (S25). Entering data manually into main storage (from the DATA switches) must be done with the CPU in the stop loop and with STORE (R32) depressed. The RATE switch (Q25) must be in the PROCESS position.

The initial setup conditions are:

1. The program is in main storage.
2. The starting address is in main storage byte locations 5 through 7.
3. The CPU is in the stop loop.
4. The PULSE MODE switch is in the TIME position.

A flowchart of the pulse mode operation is shown in Diagram 8-24, FEMDM. When the CPU is in the stop loop and the PULSE MODE switch is set to TIME, the 'pulse mode adjust' trigger is set (Diagram 8-23). Depressing START sets the 'pulse mode setup' latch. This action fires a 350-ns singleshot that initiates the pulse-mode function. The 'pulse mode initialization' and the 'force address' triggers are set. This action, coupled with the set 'pulse mode adjust' trigger, sets ROSAR(9,11) to force an address of 005 into ROSAR. A system reset is generated. Note that the 'reset manual control' signal is inhibited (Diagram 8-23). The CPU then enters the pulse-mode microprogram. The 'pulse mode adjust' trigger is reset by the pulse-mode microprogram. The loading of the count into the MCW is meaningless. Instructions are then executed until the interval timer is stepped, at which time the 'pulse mode adjust' trigger is set. Because the 'pulse mode initialization' trigger is set, a system reset occurs, an address of 005 is forced into ROSAR, and the pulse-mode microprogram is re-entered. This action results in executing the program from clock step to clock step. Looping through the pulse-mode microprogram and the main storage program continues until manually stopped.

COUNT Position. Highlights:

- Load program into main storage.
- Place starting address of program into main storage byte locations 5–7 by means of ADDRESS switches.
- Enter stop loop.
- Place PULSE MODE in COUNT position.
- Enter number of machine cycles (up to 2047) to be executed into DATA switches 53–63.
- Depress START.

When the PULSE MODE switch is in the COUNT position, instruction execution begins at the address specified in the address portion of the doubleword located in location 0 of main storage, and proceeds through the number of machine cycles entered into DATA switches 53–63. Each time the cycle counter is reduced to 0, a machine reset occurs and the program is re-entered.

Except for PULSE MODE being in the COUNT position and the count entered into DATA switches 53–63, the initial setup conditions are similar to the time-mode routine. As shown in Diagram 8-24, the same microprogram is executed for both COUNT and TIME positions. The program in main storage is entered at end-op of the microprogram. The cycle counter is reduced by 1 on each machine cycle. When the counter equals 0, a machine reset occurs, and the pulse-mode microprogram is again executed. The looping through the microprogram and the storage program continues until manually stopped.

LOG OUT Pushbutton

The LOG OUT pushbutton (S35) serves to log the machine status into fixed locations of main storage. This pushbutton is inactive during normal processing. The logic associated with the LOG OUT pushbutton is shown in Diagram 8-25, FEMDM.

TEST MODE, ROS/PROC/FLT Switch

The TEST MODE, ROS/PROC/FLT switch (N3) serves to take the CPU out of program control for FLT's or ROS tests. It has three positions:

1. ROS: When in this position and LOAD (S51) is depressed, ROS tests are read into storage from the I/O unit selected by the LOAD UNIT switches. Signals from the selected channels cause the tests to be executed in the CPU.
2. PROC – normal position: Normal CPU processing takes place.
3. FLT: When in this position and LOAD is depressed, FLT's are read into storage from the I/O unit selected by the LOAD UNIT switches. Signals from the selected channels cause the tests to be executed in the CPU.

Logic associated with this switch is shown in Diagram 8-26, FEMDM. The interval timer is disabled and the TEST indicator (S48) is lit when this switch is in the ROS or FLT position.

TEST MODE, REPEAT Switch

The TEST MODE, REPEAT switch (N1) is used to continuously repeat the test in main storage or the new test being sought. When this switch is in the Repeat (down) position, depressing START (S25) causes the ROS test or FLT in main storage to be executed repeatedly. If RESTART FLT I/O (S32) is depressed, a new FLT is sought, according to the alternate test number in T, and executed repeatedly.

RESTART FLT I/O Pushbutton

The RESTART FLT I/O pushbutton (S32) provides a means of backspacing tape one record and of starting a read operation during an FLT or a ROS test. When the TEST MODE, ROS/PROC/FLT switch (N3) is in the ROS or FLT position, depressing this pushbutton signals the channel to start with the backspace-CCW at byte location 128, decimal (80, hex). The associated logic is shown in Diagram 8-27, FEMDM.

CE Key Switch and Usage Meters

- Selected usage meter records time CPU or channel is operating.
- 'Meter in' signal indicates channel is operating.
- 'Meter out' signal indicates Process meter is recording.
- 'Not clock out' signal indicates CPU is not processing.

Two meters on the system control panel show the system running time while it is processing customer data (Process meter, R5) and while it is being operated by the CE (CE meter, R13). (The Process meter is also known as the cluster, central processing complex, system, and customer meter.) A key-operated switch (R9) selects the meter to be activated. The normal position of this switch allows power to be applied to the Process meter. The CE, using a key, switches to the CE meter. When in the CE position, the key cannot be removed and the TEST indicator (S48) is lit.

The meter selected by the key switch records time as long as the 'pass pulse' trigger is set if: (1) not in the Wait state and the 'manual' trigger is not set or (2) a 'meter in' signal is received from the channels. The selected meter also runs momentarily when any pushbutton on the system control panel is depressed (except the STOP pushbutton while already in the manual state). The 'pass pulse' trigger is set by a start, load, or reset operation and, when in single-cycle-mode operation, is reset after gating one pulse (refer to Chapter 2, Section 1). The recording of time is inhibited as soon as a stop-on-CPU-check condition is detected, without waiting for the 'manual' trigger to be set during the ensuing stop-loop operation.

The 'meter in' signal, multiplexed from the channels, is received whenever an I/O operation initiated by the CPU still is in progress and requires recording. The CPU sends two signals to the channels. The 'meter out' signal is sent whenever time is being recorded on the Process meter. The 'not clock out' signal is sent when the CPU is in the Wait or manual states or is not recording time due to a CPU machine check. These two signals are used by the channels and their attached units in running their own usage meters and, in turn, generating the 'meter in' signal.

Diagram 8-28, FEMDM, shows the CPU metering circuits. The selected meter runs for a minimum of 400 ms. Relay RR1 on the use meter card is picked by a signal from the CPU logic. 40V ac through RR1 contacts drive the meter.

FREQUENCY ALTERATION Switch

The FREQUENCY ALTERATION switch (N7) is used to increase the CPU clock frequency. When this switch is in the center position, each machine cycle is 200 ns (normal speed). With the CE Key switch in the CE position and the FREQUENCY ALTERATION switch in the down position, the CPU clock cycle is decreased to 195 ns ±½ ns (Diagram 8-28).

Indicators

On panel E of the system control panel, there are six rows of indicators, with 36 indicators (implicitly numbered 0–35) in each row (Diagram 8-1). Associated with each row of indicators is a six-position roller switch. The operator may display the contents of a register or the status of a trigger or latch by placing the proper roller switch in the correct position. A roll chart above each row of indicators shows the information being displayed for each indicator. As the roller position is changed, the roll chart rotates to correspond with the roller position. The roller switch, the positions, and the information displayed for each row of indicators are shown in Diagram 8-2.

A lamp test for the roller switches is provided by positioning the roller switches between detent positions. The sixth position of roller 6 is a lamp test for the panel E and G indicators on the system control panel and on the remote operator's console.

Other indicators on the system control panel indicate machine status, check conditions, and power status, as follows:

1. SYSTEM (S45) – Lights when the Process meter or CE meter is running.
2. MANUAL (S46) – Lights when the CPU is in the Stopped state. The CPU is executing the stop loop microprogram.

3. WAIT (S47) – Lights when the CPU is in the Wait state. The CPU is executing the wait-loop microprogram, if the MANUAL indicator is not lit.
4. TEST (S48) – Lights when a manual control is not in a position for normal processing or a maintenance function is being applied, as follows:
   a. The RATE switch (Q25) is in a position other than PROCESS.
   b. The CPU CHECK switch (N19) is in STOP or DSBL.
   c. The DISABLE INTERVAL TIMER switch (N13) is down.
   d. The ADDRESS COMPARE STOP switch (N17) is down.
   e. The PULSE MODE switch (N21) is in TIME or COUNT.
   f. The TEST MODE, ROS/PROC/FLT switch (N3) is in ROS or FLT.
   g. The STOP ON STORAGE CHECK switch (N11) is down.
   h. The REPEAT INSN switch (N23) is in MPLE or SINGLE.
   i. The DEFEAT INTERLEAVING switch (N9) is in NO REV or REV.
   j. The REPEAT ROS ADDRESS switch (N25) is down.
   k. A channel is in test mode.
   l. The Diagnose instruction is active.
   m. The "lamp test" position (position 6 of roller 6) is selected.
   n. The 'FLT TON test light' signal is active.
   o. The CE Key switch is in CE.
5. LOAD (S49) – Lights when the CPU is in a load state (IPL microprogram). The LOAD indicator is turned off after a successful load.
6. STOR CHK (K51) – Lights on all storage errors associated with the CPU or channel. The roller switches must be examined to determine the specific error.
7. PROC CHK (M51) – Lights on all CPU errors. The roller switches must be examined to determine the specific error.
8. MARGIN indicators:
   a. ACTIVE (A15): Lights when an internal power supply or an attached storage unit or channel is being marginally checked.
   b. LOCATE (A17): Lights when the MARGIN/METER SEL switch is set to the position corresponding to the margined power supply, attached storage unit, or channel.
9. Power Check indicators – Indicators on the system control panel show an incomplete power-up status in the CPU or in the attached storage units and channels. The CPU and the units are individually indicated, as is the system status. There are nine power check indicators:
   a. CPU POWER CHECK (A19): 24V dc through relay K2 contacts are fed through parallel-connected contacts on the two voltage-sense relays to the CPU POWER CHECK indicator. If either of these relays is not picked after K2 is picked, this indicator lights (ALD YA021). 24V dc are also fed through parallel-connected contacts on the overcurrent and thermal-sense relays to the CPU POWER CHECK indicator (ALD YA022). 24V dc through 48V power-check relay contacts, or through the CPU READY/OFF switch (on the CE panel) in the OFF position, also light this indicator (ALD YA021).
   b. STOR FRAME 1 POWER CHECK (A21): 24V dc, via relay K32 contacts are fed through relay K29 contacts to this indicator. If K29 is not picked after K32 is picked, this indicator lights (ALD YA026).
   c. STOR FRAME 2 POWER CHECK (A22): Similarly, through relay K30 contacts, this indicator lights.
   d. STOR FRAME 3 POWER CHECK (A23): Similarly, through relay K20 contacts, this indicator lights.
   e. STOR FRAME 4 POWER CHECK (A24): Similarly, through relay K22 contacts, this indicator lights.
   f. CHAN FRAME 1 POWER CHECK (A26): Similarly, through relay K19 contacts, this indicator lights.
   g. CHAN FRAME 2 POWER CHECK (A27): Similarly, through relay K21 contacts, this indicator lights.
   h. CHAN FRAME 3 POWER CHECK (A28): Similarly, through relay K31 contacts, this indicator lights.
   i. System Power Check: A line from the CPU POWER CHECK indicator and a line from each unit Power Check indicator are combined at an OR. The OR turns on the System Power Check indicator, which is a red lamp backlighting the POWER ON pushbutton (N43). If a Power Check indicator is on, the System Power Check indicator also lights.
10. System Power-On Indicator: An indicator on the system control panel shows complete power-up status in the CPU and in the attached storage units and channels (ALD's YA026 and YA083). This System Power On indicator consists of a clear or white lamp

backlighting the POWER ON pushbutton (N43). This condition is also indicated on the remote operator's console. The operation of the System Power On indicator is described in Chapter 5.

CE PANEL

The CE panel is shown in Diagram 8-29, FEMDM. The controls and indicators function as follows (see Chapter 5 for details):

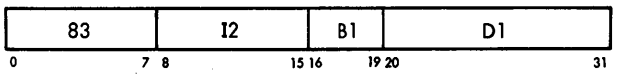
1. THERMAL RESET pushbutton: Resets the thermal sense relays in the CPU.
2. CPU READY/OFF switch. This switch has two positions:
   a. READY – normal position: Allows CPU power-up sequencing to continue if the thermal and overcurrent conditions are normal.
   b. OFF: Drops CPU power without affecting system power. The CPU is bypassed on a system power-on sequence.
3. CPU ON pushbutton: Starts CPU power-on sequencing if the CPU READY/OFF switch is in the READY position. Does not affect system power.
4. THERMAL TRIP indicators: These six indicators show the location of the overtemperature condition that dropped CPU power. The temperature sensors are located in gates A, B, C/D, and E, the converter/inverter, and the power supply tubs.
5. UNDER VOLTAGE CHECK switches (located on the relay gate below the CE panel in the converted units): These 15 switches isolate the power supplies from the undervoltage sensing circuits.

www.IBM360.info

This section discusses the 2065 maintenance features available to the CE: (1) Diagnose instruction and associated MCW's; (2) Logout, ROS tests, and FLT's; (3) ripple tests; (4) diagnostic programs; and (5) marginal checking. For a discussion of the system control panel and the CE panel, refer to Section 1 of this chapter.

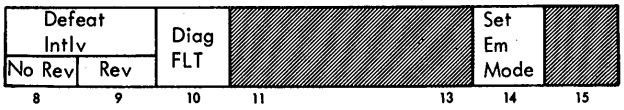
DIAGNOSE INSTRUCTION AND MCW'S

The Diagnose instruction (Section 7 of Chapter 3) has two purposes: it is available to the diagnostic programmer as a maintenance aid and to the system programmer for emulator mode operations. The Diagnose instruction has an SI format with an op code of 83:



The operations that the Diagnose instruction can do are selected by the I2 field of the instruction and by the bit configuration of the doubleword addressed by the storage operand address (contents of GPR addressed by B1, + D1). The right half of this doubleword is the MCW, and bears the same relation to the Diagnose instruction as the CCW does to an I/O instruction. That is, the Diagnose instruction addresses the location of the MCW, and the MCW specifies the machine function. Note that an exception to this analogy is the I2 field, which can also designate certain diagnostic functions.

The bits of the I2 field have the following meaning:



1. Bit 8, Defeat Interleaving and No Reversal of Storage Addresses. Interleaving is disabled and no even/odd storage area reversal takes place.
2. Bit 9, Defeat Interleaving and Reverse Storage Addresses. Interleaving is disabled and the even and odd storage areas are reversed.
3. Bit 10, Diagnose FLT. Allows portions of FLT's to be executed under control of the Diagnose instruction. While the FLT's are being executed, special CPU functions are generated, and storage requests and clock are inhibited. This bit, which is used in conjunction with the setting of address 6B0 in ROSAR, simulates the FLT position of the TEST MODE, ROS/PROC/FLT switch on the system control panel.
4. Bits 11–13. Spares.
5. Bit 14, Set Emulator Mode. Allows the CPU to enter the emulator mode. (See Compatibility Feature FETOM's for a discussion of emulator mode operations.)
6. Bit 15. Spare.

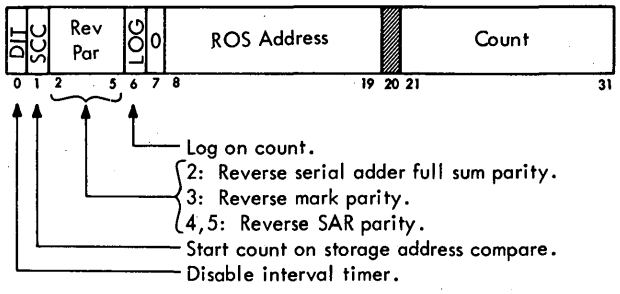
Besides the functions that can be selected by the I2 field, the Diagnose instruction can:

1. Perform channel diagnostic functions. When a channel is selected by the MCW, it is disconnected from its control units and connected to an internal (to the channel) interface simulator which consists of a one-byte register and associated controls. This simulator allows the diagnostic program to test the channel regardless of the type of attached control units that may be available.
2. Reverse the state of certain parity bits. Incorrect parity can thus be placed into the CPU or into the channel to test the error detection logic.
3. Cause a logout after a specified number of cycles (CPU only).
4. Branch into the ROS microprogram at any point.
5. Perform portions of the LS ripple tests.
6. Branch to an end-op micro-order and thus terminate normally.
7. Force PAL full-sum errors.
8. Perform scan-in operations. (Refer to *2065 Processing Unit* FEMM, Form Y27-2039-2 for the procedure.)
9. Stop the CPU clock.
10. Read the contents of the system control panel DATA switches into storage.

The MCW is a 32-bit word used to control Diagnose-instruction and scan operations. When the Diagnose instruction is executed, the MCW is in the main storage location specified by the storage operand address and designates the diagnostic function(s) to be performed. For ROS tests and FLT's, the MCW is contained in word 1 of each test on the ROS or FLT test tape and specifies the control conditions necessary for the test, such as the expected result and the ROS word to be used for gating control. A different format is used for each MCW.

Diagnose Instruction MCW for CPU

When executing the Diagnose instruction and MCW(7) = 0, the MCW applies to the CPU, as follows:

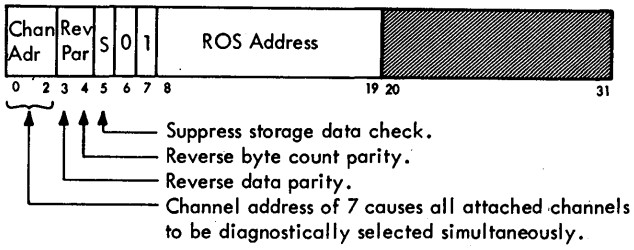


1. Bit 0, Disable Interval Timer. Disables the interval timer.
2. Bit 1, Start Count on Storage Address Compare. When used with bit 6, the FLT counter does not begin decrementing until the BCU addresses the same location as set into the MAIN STOR ADDRESS COMPARE (ADDRESS switches 2–20) switches.
3. Bit 2, Reverse Serial Adder Full-Sum Parity. Reverses the parity bit in the full-sum latch of the serial adder, thus allowing testing of the parity-checking circuits.
4. Bit 3, Reverse Mark Parity. Reverses the parity of the mark bits being sent from the BCU to main storage, thus allowing testing of the mark parity-checking circuits in main storage.
5. Bits 4 and 5, Reverse SAR Parity. Cause the parity bits which are sent to the storage address register to be reversed as follows:
   00 – No parity reversal.
   01 – Reverse low-order parity bit.
   10 – Reverse next-higher parity bit.
   11 – Reverse high-order parity bit.
6. Bit 6, Log On Count. Causes a logout to main storage when the FLT counter reaches 0. At the conclusion of the logging operation, the CPU performs a machine check interruption.
7. Bit 7. Must be a 0.
8. Bits 8–19, ROS Address. When the Diagnose instruction has completed its execution phase, these address bits are placed into ROSAR, and the operation branches to this location. The address placed into ROSAR can specify any location in ROS. Refer to Diagram 5-608, Sheet 2, FEMDM, for the most frequently used ROS addresses.
9. Bit 20. Spare.
10. Bits 21–31, Count. Specify the number of cycles (200 ns) that are to occur before the CPU enters a logout routine. The count field is made up as follows: MCW(21–25) is loaded into the address sequencer; MCW(26–29) is loaded into the FLT counter; MCW(30,31) is loaded into the FLT clock. By combining the address sequencer, the FLT counter, and the FLT clock, a maximum count of 2047 (11 bits) can be obtained. (These three counters are combined only when the MCW is used with the Diagnose instruction.) The FLT clock controls the decrementing of the FLT counter and address sequencer by 1; when the three counters combined equal 0, the logout routine is started.

Diagnose Instruction MCW for Channel

When executing the Diagnose instruction and MCW(7) = 1, the MCW applies to the channel. When a channel is selected by the MCW, the channel is disconnected from its control units and is connected to an internal interface simulator. The interface simulator consists of a one-byte register and associated controls. This simulator allows the diagnostic program to test the channel regardless of the type of control units that may be attached and available.

The MCW format for the channel diagnostic function is:

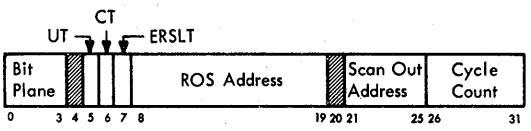


1. Bits 0–2, Channel Address. Select the channel to be diagnosed according to the following bit configuration:
   000 – Select channel 0 (multiplexer channel).
   001 – Select channel 1.
   010 – Select channel 2.
   011 – Select channel 3.
   100 – Select channel 4.
   101 – Select channel 5.
   110 – Select channel 6.
   111 – Select all channels simultaneously.
2. Bit 3, Reverse Data Parity. Causes all bytes that are read from the interface simulator to have reversed parity. This action allows testing of the storage bus-in parity check circuit.
3. Bit 4, Reverse Byte Count Parity. Provides a means of testing the byte control check circuits.
4. Bit 5, Suppress Storage Data Check. Prevents a storage data check from causing a channel data check and prevents a channel control check on a CCW fetch operation. Preventing the channel control check allows invalid CCW's to be brought into the channel to test sections of the channel check circuits.
5. Bit 6. Must be a 0.
6. Bit 7. Must be a 1.

7. Bits 8–19, ROS Address. When the Diagnose instruction has completed its execution phase, these address bits are placed into ROSAR and the operation branches to this location. The address placed into ROSAR can specify any location in ROS. Refer to Diagram 5-608, Sheet 2, FEMDM, for the most frequently used ROS addresses.
8. Bits 20–31. Spares.

ROS Test MCW

The MCW for ROS tests, the right half of word 1 of each test, contains the following control information about the test:

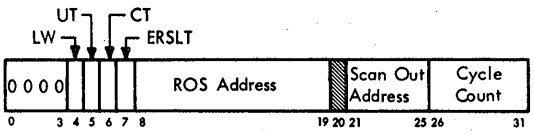


1. Bits 0–3, Bit Plane. Contain the number of the ROS bit plane tested. These bits are for display only; the ROS word is selected by the ROS address in bits 8–19, and the bit to be tested is selected by the mask.
2. Bit 4. Spare.
3. Bit 5, Unconditional Terminate (UT). If this bit equals 1, the test always causes a stop.
4. Bit 6, Conditional Terminate (CT). If this bit equals 1 and an error is encountered, testing is terminated. If, however, it equals 1 and the test does not detect an error, the CPU continues with the next ROS test. If this bit equals 0, termination is dependent upon the status of MCW(5).
5. Bit 7, Expected Result (ERSLT). Defines what the status of the ROS bit being tested should be. Comparing the ROS bit with the 1 or 0 in this bit determines whether the test passed or failed.
6. Bits 8–19, ROS Address. Contain the ROS address of the micro-instruction that contains the bit to be tested.
7. Bit 20. Spare.
8. Bits 21–25, Scan Out Address. Specify the scan-out address (roller switch position) of the portion of the ROSDR that contains the bit to be tested. This address will be either 15, 16, or 17. (See ALD M3051.)
9. Bits 26–31, Cycle Count. Determine the number of CPU clock cycles ROS must cycle to fetch the desired ROS word into ROSDR.

FLT MCW

The MCW in the right half of word 1 of every FLT contains control data about the test being performed. Pertinent bits of the MCW are retained in the MCW register during the running of an FLT. Other bits set the address sequencer, determine the ROS starting address, and fix the number of clock cycles the CPU will take following scan in.

Following is the format of an MCW during an FLT:



1. Bits 0–3. Contain 0's.
2. Bit 4, LW. Defines whether the left half or right half of the word to be scanned out contains the exit trigger status. If bit 4 equals 1, the left half is the desired word. If bit 4 equals 0, the right half word contains the trigger status.
3. Bit 5, UT. Always causes a stop when it equals 1.
4. Bit 6, CT. May cause a stop, depending on the outcome of the test. If this bit equals 1 and an error is encountered, testing is terminated. This bit is always set in zero-cycle and one-cycle FLT's currently in use.
5. Bit 7, ERSLT. Defines what the status of the exit trigger should be. Comparing the exit trigger state with the 1 or 0 in this bit determines whether the test passed or failed.
6. Bits 8–19, ROS Address. Contain the ROS address of the ROS word to be used for gating control.
7. Bit 20. Spare.
8. Bits 21–25, Scan Out Address. Specify the scan-out address of the scan-out word containing the status of the exit trigger.
9. Bits 26–31, Cycle Count. Fix the number of times the CPU is to cycle following scan in.

LOGOUT, ROS TESTS, AND FLT'S

Logout stores the status of the system control panel indicators into fixed positions of main storage when a trouble symptom occurs; the data logged out may be subsequently recalled for analysis although the status of the indicated logic is changed from what it was when the symptom appeared. ROS tests check each bit position of every ROS word. FLT's check the CPU at the logic block level.

Introduction

Logout, ROS tests, and FLT's are implemented by special hardware called scan logic. Therefore, when employing these maintenance aids, the CPU is said to be in the scan mode.

The scan logic performs the following functions:

1. Controls the operation of FLT's.
2. Records the state of the CPU when a machine malfunction is detected (logout).
3. Executes the Diagnose instruction.
4. Controls the operation of the ROS tests.

In the discussion that follows, certain terms are used which refer only to scan operations. These terms are defined as follows:

1. Scan mode. The CPU is said to be in "scan mode" during those operations that use the scan logic; i.e., FLT, ROS tests, and logout.
2. Scan in. The process of loading CPU register positions and control triggers with a predetermined bit pattern from storage.
3. Scan out. The sampling of the status of certain triggers via a special data path from the indicator logic to PAL.
4. Logout. The function that transfers status indicators and register contents to storage when signaled by the system.

Logout

The logout function of the scan logic stores the status of various triggers and registers, reflecting the state of the CPU, into predesignated locations of main storage. The 22 doublewords logged out (Table 6-1) are stored into main storage locations 80 through 128 (hex). (For a detailed list of the 22 doublewords, refer to ALD's M3021–M3061.)

Note: The Multisystem feature adds 16 log words (22–37, in main storage locations 180–1F8); see Chapter 4, Section 2.

The status of each trigger logged out is represented by a 1 if it is set, or by a 0 if it is reset. Thus a record of the machine state, at the time that a CPU or storage error occurs, is stored unchanged in predetermined locations of main storage with a fixed format. This record can then be accessed by a program or by manual controls for analysis, printout, or display.

A logout operation can be initiated by:

1. Manually depressing the LOG OUT pushbutton on the system control panel. The system must be in manual mode.
2. Executing the Diagnose instruction when a log-on-count function is specified. Logout occurs after a predetermined number of CPU clock cycles (preset by the programmer).
3. Detecting a machine check during normal CPU operation if the CPU CHECK switch is in the PROC position and the PSW machine check mask bit is on. When a storage error is detected and the CPU is enabled for machine checks, an additional word (log word 20) is logged out. This word, from the roller switch associated with the storage in error (position 5 on rollers 1, 2, 5, or 6), contains the information, excluding storage protect indicators, that is displayed in the roller switch indicators.

Table 6-1. Logout Format

Main Storage Byte Address (Hex)	Log Word	Left Half	Right Half
080	0	Parity for D-, F-, and Q-registers	Parity for ST-register
088	1	Miscellaneous controls	Parity for AB-, E-, and R-registers
090	2	G-register, LCS, and segmented clock	Storage checks
098	3	-----	Interruption, I-Fetch, and STAT's
0A0	4	PSW	BCU, FLT, and manual controls
0A8	5	-----	FLT controls, MCW, and counters
0B0	6	-----	ABC, STC, LAR, B(64–67), mark triggers, and PAL(64–67)
0B8	7	A-register	B-register
0C0	8	-----	Features
0C8	9	-----	IC and SAL
0D0	10	R- and E-registers	D- and F-registers
0D8	11	Q(0–31)	Q(32–63)
0E0	12	-----	CPU checks
0E8	13	Parity for LSWR, PSW, and IC; LSWR, PSW, IC, and checks	LSWR
0F0	14	-----	Gate control triggers
0F8	15	-----	ROSDR(70–99) and edit triggers
100	16	-----	ROSDR(37–69)
108	17	-----	ROSDR(1–35)
110	18	-----	ROSAR (3 registers)
118	19	-----	ROSAR, parity for ROSDR, LS address register, and gate control triggers
120	20	-----	Storage address, cycle, and data check(s)*
128	21	S-register	T-register

Note: PAL(64–67) and SAL(0–7) should always = 0 on logout, and the parity bits should always = 1.

*Present only when a storage check occurs and machine checks are enabled.

ROS Tests

- Test each bit of each ROS word.
- Tests are on tapes or disks.

ROS tests are the principal means of testing the validity of the ROS bit planes. These tests, generated by a computer program from the tapes used in the manufacture of the ROS bit planes, are stored on magnetic tape or disks. When testing ROS, the tests are read into two main storage buffer areas, starting at locations 8000 and 8080 (hex). Under scan logic control, the ROS tests compare the value of a particular bit in a selected ROS word with its expected value as specified on the ROS test tape. Each ROS test is read into the CPU and checks one bit of a ROS word. At the same time that the test from one buffer area is being executed in the CPU, the other buffer area is being filled from the test tape via the channel.

The ROS test format consists of two doublewords: word 0 and word 1. Word 1 contains the mask and MCW. The mask, a 32-bit field, in conjunction with the scan-out address field of the MCW, selects the ROS bit to be tested from the word read out of ROS. A particular bit is tested by making all mask bits 1's except the bit that corresponds to the test bit. Word 0 contains the test number (TN) and alternate test number (ATN). The TN, a four-byte field, contains two two-byte numbers that identify the test pattern. The lower-order two bytes are the complement of the TN, and the high-order two bytes are the TN. The ATN, which refers to another ROS test, is also represented in true and complement form, with the complement and true numbers reversed from that of the TN format. The TN refers to the test being executed. The ATN refers to the test that will be executed if the tests are restarted after a failure stop (generally, it refers to the next test).

The first part of each ROS test tape contains hardcore tests to establish that the CPU is able to run ROS tests. Testing should not proceed beyond the hardcore tests if failures are encountered.

Following the hardcore tests are the actual ROS tests. During these tests, the ROS word to be tested is selected by the ROS address in the MCW. The CPU clock is allowed to generate clock signals to cycle ROS so that the bit under test is placed into the ROSDR. The word containing this bit is defined by the scan-out address in the MCW and is transferred (scanned out) via the indicator driver logic and PAL to the T-register.

The mask is transferred from main storage to the S-register, and then the status of the bit under test is determined by comparing S with T. The result of this comparison is compared with the ERSLT bit to determine whether the test passed or failed. Pass, fail, and intermittent-fail are the three possible results of the comparison. The CT and UT bits in the MCW then determine whether to proceed with testing or to terminate.

If testing is terminated on a failure, the last two hex digits of the ROS address of the failing word are displayed in S(0–15) (roller 1, position 3), and the bit plane is displayed as the CPU TEST ADDRESS on roller 5, position 2, bits 0–3. To further define the S-register contents, S(0–7) indicates the failing word in hex, and S(8–15) indicates the failing bit in decimal.

The test is repeated until a transfer in channel (TIC) pulse notifies the CPU that the alternate buffer is filled and new test data is available.

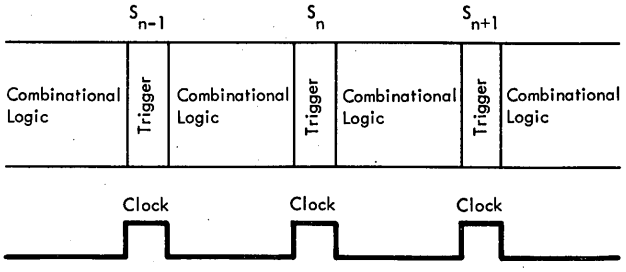
Because FLT's cannot be run if malfunctions exist in ROS, the ROS tests should be run first, followed by the FLT's.

FLT's

- FLT's check CPU logic at block level.
- Exit trigger determines whether test passed or failed.
- FLT's have three basic phases: scan in, clock advance, scan out.
- Three categories of FLT's are hardcore tests, zero-cycle tests, and one-cycle tests.

FLT's are a unique maintenance concept in that CPU logic is checked without executing CPU instructions (i.e., without executing a program in the ordinary sense). In an FLT, fault detection is performed at the logic level. That is, FLT's are concerned with the logical function (OR, AND, INVERT) of a block rather than with its operational function (e.g., as an adder, counter, control) in the CPU.

The tests are automatically produced by an IBM 7090 System which uses as its source the logic description of the CPU. Because the FLT's are not generated from an actual machine but are made from a mathematical model stored in 7090 storage, the FLT generator program treats the CPU as a set of triggers which can take on a new state (S) every time a clock signal is generated:



Thus, FLT's are generated for groups of logic blocks. A group consists of an exit trigger (a temporary storage

element taking on new data at clock time and holding the information during not-clock time) and associated logic that can affect the trigger during an advance of a predetermined number of cycles.

The FLT generating programs analyze the group of logic blocks and determine, from a set of input values, an expected response at the output of the exit trigger. (Note that, although the exit trigger determines whether the test passed or failed, it is not being tested per se; the FLT tests the logic that precedes it.) Using the information gained by the analysis, the program generates a test pattern which it places on tape. Each test pattern is an FLT. At the same time, the program generates the documentation necessary to troubleshoot a failure detected by the FLT.

The application of an FLT can be broken into three phases:

1. Scan in. Triggers are set to the desired state by a scan-in microprogram. Triggers that cannot be set are assumed to be at a predetermined value.
2. Clock advance. The number of clock signals specified by the FLT MCW are allowed to be generated. These signals advance the CPU from one trigger level to the next. The exit trigger assumes a value that is a function of the state of the CPU during the previous cycle.
3. Scan out. The new value of the exit trigger is compared with the expected result. If the machine is operating properly, the two values should agree. If the new value and the expected result differ, a fault has been located, and testing terminates† with reference to the FLT documentation.

FLT's can be divided into three categories:

1. Hardcore tests. Check the scan and normal CPU logic necessary to run FLT's.
2. Zero-cycle tests. Determine whether a trigger value can be changed by scan in and also whether the new value can be sensed. Zero-cycle tests establish the machine capability to scan in and scan out before running one-cycle FLT's.
3. One-cycle tests. During these FLT's, data is scanned into the CPU, the clock is allowed to run, and the exit trigger is scanned out and compared with a known value. One-cycle tests check combinational†† logic within the CPU.

Note that the terms "zero-cycle" and "one-cycle" do not refer to the number of clock cycles allowed after scan in. For example, if during a zero-cycle test the exit trigger requires a clock signal to set it, the clock must run for one cycle. These terms refer to test techniques rather than to any time element.

† Termination is conditional upon the contents of the MCW.

†† Combinational logic is all the logic required to pass the state of one trigger to the next by executing a specified number of clock cycles.

FLT Tapes. Highlights:

- FLT's are stored on magnetic tape or disks in the following order: hardcore tests, zero-cycle tests, one-cycle tests.

The FLT's are stored on magnetic tape or disks. Each FLT tape consists of thousands of tests, each concerned with a single sensitive path. Also contained on the tapes are tests preliminary to FLT's; i.e., tests designed to ensure that the CPU is capable of performing FLT's and that the triggers to be tested can be set or reset and the change sensed.

Each tape is divided into three sections: hardcore tests, zero-cycle tests, and one-cycle tests. The one-cycle tests are considered to be the true FLT's because these test combinational logic. A brief description of the three kinds of tests on the FLT tape follows:

1. Hardcore tests. The test tape begins with hardcore tests to check out the CPU hardware necessary for operating the FLT's. Hardcore tests determine that the S- and T-registers are functioning properly, that their bit content can be correctly sensed at PAL, and that the CPU can make decisions based on the outcome of a test and then act on that decision.
2. Zero-cycle tests. Next in the testing sequence are the zero-cycle tests. These tests verify that the exit trigger status can be changed and that the change can then be observed or sensed. Zero-cycle tests set the trigger, using either a special scan input or a normal machine path, and then verify the change in status of the trigger being tested. Because clock signals are needed to set most triggers, the CPU clock is allowed to cycle just enough times to set the trigger. Upon completion of the zero-cycle tests, the CPU's ability to run FLT's has been verified.
3. One-cycle tests. These tests make up the bulk of the FLT tape. They vary in the amount of logic checked, and run in sequence until a failing test is encountered, whereupon testing is terminated and the failing test number is displayed in the S-register. One-cycle tests require two or three clock cycles to set a trigger.

Tape Generation. Highlights:

- FLT tape is computer-generated from ALD data.
- Program develops sensitive tree with entry points and terminating in an exit trigger.
- Tests are printed out in two formats: test, including entry pattern, and listing of sensitive points (SCOPEX) which is used in troubleshooting.
- New tests are added to existing tape using FLUT.

The FLT tape is computer-generated from ALD data. Using this data, a special program develops tests that, when executed, affect triggers indicatable on the system control panel (designated exit triggers). The program works back from all entries into each of these triggers,

searching the logic path for "sensitive" points; i.e., points at which an error or a failure would propagate to the exit trigger. Only one fault is assumed for each sensitive point.

The logic feeding these sensitive points makes up a sensitive net, and, as the number of nets grows, the combinational logic involved resembles a tree; the exit trigger forms the tip of the tree, and the sensitive nets make up the body of the tree. The search continues until the program encounters another trigger that can be used as an entry point into the sensitive tree. Usually, several entry points into a particular tree are available. However, entry points are selected on the basis that data injected at the entry points will propagate through to the exit trigger. The program also selects micro-orders that will move data from the entry points to the exit trigger. Upon completion of the search, the program has developed a sensitive tree with entry points and terminating in an exit trigger. The program also has developed an entry pattern that results in a predictable status of the sensitive nets and a predictable change at the exit trigger. Once this pattern has been developed, an evaluator program verifies its correctness.

Upon verification, the test is printed out in two formats: the test itself, including the entry pattern, and a listing of the sensitive points within the tree. The first of these becomes the taped FLT, and the latter is the scoping documentation (scoping index, or SCOPEX), used in troubleshooting a failing test.

To put the FLT's in CPU language, the entry pattern is scanned into the CPU, micro-instructions are selected to move the data through the sensitive tree during a given number of clock cycles, and the exit trigger is then observed (scanned out) to determine whether it is at the predicted value for the test. If the trigger is not at the predicted value, the test has failed, and the CE may then repeat the failing test continuously and scope the sensitive points on the tree to find the net with the failure in it. Certain sensitive points appear in more than one test, and the same exit trigger may be the observation point for more than a single sensitive path. Thus, newly tested points are indicated as being newly tested on SCOPEX to indicate to the CE that this is the first time these sensitive nets have been encountered and tested.

By verifying that AND's, OR's, and INVERT's are functioning as designed, FLT's verify the operating capability of the CPU although specific computer functions are not performed. FLT's ascertain that the CPU is operating according to design specifications and, therefore, should be capable of functioning as a CPU.

After the FLT has been evaluated as a valid test, it becomes a part of the FLT tape and is distributed for field use. New FLT's are continually being generated, some to test new logic that results from engineering changes and other to test areas of the CPU that are not now being checked. New tests are incorporated into existing tapes with the Fault Locating Utility (FLUT) program. Normally, however, a complete tape is sent to the field incorporating both old and new tests. An example of how one FLT is generated appears in Appendix C.

FLT Hardcore Tests. The first eight records on the FLT tape contain tests that ensure that the hardcore logic necessary to run zero-cycle and one-cycle tests is operational. Hardcore logic is defined as all the logic, scan and normal, necessary to load the FLT's into storage, to scan in, to scan out, to make decisions based on the outcome of a test, and to act on those decisions.

The CPU hardcore hardware testing is subject to the following limitations:

1. Scan hardware is not tested directly; it is, however, exercised in the hardcore portion with tests designed to isolate the trouble.
2. The BCU is not tested in the FLT's, but the BCU must be functioning properly to run FLT's. Also, main storage and storage buses must be operating. Main storage may be checked using ripple tests.
3. The channel and tape drive (or disk) used to read in the FLT's are not tested. These units must be tested using manual controls on the channel.
4. Local storage is not needed to run FLT's, and, therefore, is not tested. Local storage is checked by ripple tests.
5. The ROS microprogram and ROS must be fault-free to run FLT's. Therefore, because hardcore tests do *not* check ROS, ROS tests should be run before FLT's.

Zero-Cycle Tests. To further check the operation of the scan hardware, zero-cycle tests determine whether the scan-in and/or scan-out paths are operative. Zero-cycle tests check only those triggers displayed on the system control panel. In these tests, a pattern is scanned into the machine, the clock is not advanced, and the exit trigger is observed. If the trigger has a scan-in path, three tests are performed: one for the reset state, one for the set state, and one for the reset state again. If the trigger has no scan-in path, only one test is performed for the reset state. While one trigger is being tested, other triggers can assume various states. Whenever possible, random states are used to simulate the combinations that may be used in a normal test and to reveal interaction between triggers.

The functions checked with zero-cycle tests include:

1. Reset to triggers (this is, in effect, a scan-in-zero operation).
2. Scan-in path to triggers.
3. Ability of a trigger to hold its value in the absence of a clock signal.
4. That scan-out bus gating signals can be generated.

One-Cycle Tests. One-cycle tests are the true FLT's. The input, produced by the FLT generating system, is a test that detects and locates at least one fault. The input pattern is scanned into selected triggers, and the CPU is

allowed to advance a given number of cycles. The result, which is in the exit trigger, is compared with the value expected for a correctly operating machine.

If the value in the trigger does not agree with the expected result, testing is terminated and the failing TN is displayed in S(0–15) for reference to the SCOPEX.

The SCOPEX is a series of lists, one for each test. Each list is headed by the test number in hex, followed by a row of asterisks, and consists of several lines, each line referring to a pin in the machine. If a pin in the machine is contained in a list, the net which feeds that pin is sensitive for the test pattern applied; a failure on the card, which can be observed with an oscilloscope at that pin, would cause the test to fail.

FLT Format. Each FLT on the test tape consists of nine doublewords, numbered 0 through 8 (10 doublewords for machines with the 7080 Compatibility Feature or the 709/7040/7044/7090/7094/7094II Compatibility Feature). This is one complete test. Each test occupies one of two areas in storage. Buffer 1 begins at main storage location 8000 (hex) and contains nine doublewords. Buffer 2 begins at main storage location 8080 (hex) and also contains nine doublewords.

While one test is being executed from one buffer, the other buffer is being filled from the channel. Filling the buffers is a sequential process, and the processing of the FLT's is in the same sequence, without further addressing.

The contents of the nine or ten doublewords is as follows:

1. Word 0 and words 2 through 7 or 8 contain bit patterns that are scanned into the CPU.
2. Word 1 contains a mask and the FLT MCW. The mask defines the exit trigger; the MCW, in the right half, contains control information about the test.
3. Word 8 (or 9) contains the TN in both true and complement form and the ATN in the same format. Word 8 (or 9) is not read into the CPU during test sequencing unless a fault is encountered or an ATN search is performed.

At the satisfactory completion of a test, the 'pass' trigger is set, the address sequencer is set to 7 (or 8), and word 7 (or 8) of the test is the first word scanned in again. When a test fails, the 'fail' trigger is set. After a 'TIC' pulse is received from the channel, the last thing the CPU does is to fetch word 8 (or 9) from storage and leave it in the S- and T-registers. If the CE desires to skip this test and continue with the remainder of the tests on the tape, he may depress the RESTART FLT I/O pushbutton. The signal generated by depressing this pushbutton causes the I/O device to backspace to the beginning of the record and to start reading the tests into storage again. As each buffer is filled, the TN in the left half of word 8 (or 9) is put into the S-register, and a comparison is made between the new TN in the S-register and the ATN in the T-register by using the scan out S and T facility. The result is 0 for the test immediately following that test in which a failure occurred. Upon detection of this 0 result in PAL, the CPU again begins testing and continues until the next failure. This procedure is the only way of getting past a failing test and continuing the test tape sequence.

Scan Logic Functional Units

The scan logic functional units (Diagram 8-101, FEMDM), in conjunction with the CPU operational hardware and the ROS microprogram, control the CPU during FLT, ROS tests, logout, and Diagnose instruction execution. During these operations, the scan logic supplements and sometimes overrides the operational CPU logic. In addition, the scan logic provides direct data flow paths (scan in) into triggers not ordinarily having direct entry paths and from indicatable storage devices to PAL (scan out).

During scan operations, CPU timing is controlled by two clocks (scan clock and FLT clock), and sequential operation is determined by four counters: address sequencer, check counter, FLT counter, and ROS test sequencer. These counters time-share the scan counter latches and decrementer. That is, although each counter is used for a specific function at a specific time, their contents are all decremented by the scan counter decrementer. For certain operations, the address sequencer and the FLT counter can be logically joined to triggers in the FLT clock to form an 11-bit cycle counter.

The five-bit address sequencer contains the low-order bits of storage word addresses, and its decoded value controls the gating signals to the scan-in or scan-out logic. The address sequencer also forms the high-order five bits of an 11-bit counter (as mentioned previously) for the Diagnose instruction and pulse-mode functions. The six-bit FLT counter counts CPU clock signals when required; the low-order two bits of the FLT counter, which are contained in the FLT clock, count oscillator signals reaching FLT controls. The check counter permits a tape record to be read a maximum of 32 times in case of tape errors or storage errors before determining that a test cannot be performed. The three-bit ROS test sequencer sequences the scan hardware during ROS tests.

Each ROS test or FLT has an associated MCW which allows the test to be controlled by a test pattern on tape. MCW's, including the Diagnose instruction MCW, are stored in the scan functional units and decoded by scan hardware.

Scan Timing

Scan timing is controlled by two clocks, the scan clock and the FLT clock. The scan clock is synchronized to the CPU clock and produces clock and not-clock signals similar to the CPU clock. Normally, the scan clock runs continuously whether the CPU is in scan mode or

functional operation, and can be made to run with the CPU clock off. The output of the scan clock steps the FLT clock.

The FLT clock is used in conjunction with the scan clock to provide scan control timing. It is synchronized to the scan and CPU clocks, and provides four output signals, FLT-time 0–3. The two FLT clock triggers can be logically combined with the address sequencer and the FLT counter to form an 11-bit cycle counter.

Scan Clock. Highlights:

- Consists of inverters which delay 'gated oscillator D' signal from CPU clock.
- Distributes clock and not-clock signals with delay levels of P0-3 to P2.
- Runs continuously unless blocked by BCU.
- During scan operations, CPU clock is turned off, leaving scan clock in control.

The scan clock (Diagram 8-102, FEMDM) consists of a number of inverters, each of which delays a 'gated oscillator D' signal from the CPU clock approximately 10 ns. The output of each delaying inverter is distributed as a clock or not-clock signal with a delay level of from P0-3 to P2. (See Chapter 2, Section 1, for an explanation of clock, not-clock, and delay levels.) The scan clock runs continuously unless one of the following actions stops it:

1. The 'pass pulse' trigger is reset (which also stops the CPU clock). This trigger is set when any operation is initiated from the system control panel, and normally remains set throughout most CPU operations. However, it can be reset by one of the following conditions, if the BCU is not holding the CPU clock on:
   a. 32 attempts to read in an FLT or ROS test were unsuccessful because of channel or storage errors.
   b. The UT bit was on at the completion of an FLT or ROS test.
   c. The CT bit was on and the 'fail' trigger was set during an FLT or ROS test.

   Note: The above three conditions reset the 'pass pulse' trigger at FLT-clock-2 time if the ROS test sequencer is at 0.

   d. The 'block' trigger is set. This trigger is set if one of the pushbuttons listed in Diagram 8-102 is depressed in single-cycle mode.
2. The 'stop clock' trigger is set with the RATE switch *not* in the SINGLE CYCLE STORAGE INHIBIT position and with the TEST MODE, REPEAT switch not set. The 'stop clock' trigger is set by the 'CPU 2' latch, the 'insert key' trigger, or the 'LCS interleave' latch.
3. The 'STOP 1' or 'STOP 2' micro-orders are executed.

Because the scan clock is driven by a 'gated oscillator D' signal (basic clock signal delayed; Diagram 4-3, FEMDM), it is always in synchronization with the CPU clock. The scan clock signals, P0-3, P0-2, P0-1, P0, P1, and P2 (Diagram 8-102) are developed by passing the 'gated oscillator D' signal through inverters. Each inverter delays the signal approximately 10 ns. The P0-3 signal is the first scan clock output and is developed after being delayed by an adjustable time delay. (The time delay should be adjusted so that P0 from the scan clock coincides with P0 from the CPU clock; See ALD M8001.) The P0-3 signal is then inverted and delayed 10 ns to produce the P0-2 signal. The P0-2 signal, in turn, is inverted and delayed to generate P0-1. The remaining scan clock signals are generated in the same manner as P0-2 and P0-1. These clock signals are distributed throughout the scan logic to time scan operations. Clock and not-clock signals have the same relation to scan triggers and latches as CPU clock and not-clock signals have to functional logic (see Chapter 2, Section 1).

The advantage of having a separate scan clock for scan logic is that during scan operations the CPU clock can be turned off, leaving the scan clock in control. This function is under control of the 'maintenance mode stop clock' (MMSC) trigger. (The operation of the MMSC trigger is discussed in "Scan Stop-CPU-Clock Logic".)

With both the scan and CPU clocks running, scan hardware is controlled by the scan clock, and normal CPU functions are controlled by the CPU clock.

FLT Clock. Highlights:

- Provides scan control timing and counts machine cycles.
- Consists of 'FLT clock-0' and '-1' triggers and 'FLT time-0' to '-3' latches.
- Stepped by scan clock.

The FLT clock (Diagram 8-103, FEMDM) is the prime scan operation sequencer which, in conjunction with the scan clock, provides scan control timing and counts machine cycles. It consists of two triggers, 'FLT clock-0' and '-1', and four latches, 'FLT time-0' to '-3'. Clock and not-clock signals from the scan clock step the FLT clock to provide the outputs shown in the timing chart. Six signals are generated, 'clock-0' and '-1', and 'FLT time-0' to '-3'. The clock steps once for each machine cycle. Thus, in four machine cycles, the FLT clock steps from 'FLT time-0' to '-3' and then recycles.

Two triggers, 'FLT clock 0' and 'FLT clock 1', are stepped by scan clock signals and by the conditions of the four latches. 'FLT time-0', '-1', '-2', and '-3'. The FLT clock triggers function as a reverse binary counter of scan clock cycles. The FLT time latches record the count indicated in the FLT clock triggers.

Assume that the 'FLT time 0' latch is set and both FLT clock triggers are reset (00). Because 'FLT time-1' to '-3' latches are reset, the rise of scan clock P0 sets both FLT clock triggers (11). At not-clock P0 time, the 'FLT time 0' latch is reset, and at not-clock P1 time, the 'FLT time 1' latch is set. Then, at clock P0-1 time, both FLT clock triggers are reset.

Because the 'FLT time 1' latch is set, only the 'FLT clock 0' trigger is set at clock P0 time, at which time the triggers equal 10. This operation continues with the triggers counting down binarily from 11 to 00 during clock time and the latches stepping up to 11 during not-clock time. When the trigger count is 00, the next not-clock P1 signal sets the 'FLT time 0' latch, and the cycle is repeated.

The FLT clock triggers are also used as the low-order two bits of the FLT counter and the cycle counter. As shown in Diagram 8-103, T(62,63), containing MCW(30,31), can be transferred to these triggers during cycle counter operation. Operation of the triggers as part of the FLT counter and the cycle counter is discussed in "FLT Counter" and "Cycle Counter", respectively.

Scan Counter Latches and Decrementer

Three counters and two sequencers (Figure 6-4) control the sequential operation of scan associated functions: the address sequencer, the check counter, the FLT counter, the ROS test sequencer, and the cycle counter. (The latter is an 11-bit counter composed of the address sequencer, the FLT counter, and the two FLT clock triggers.) Each sequencer or counter is decremented by routing the contents of the sequencer or counter through the scan counter latches and back to the source via the scan counter decrementer. The scan counter decrementer subtracts 1 from the value gated through it. However, although all counters are decremented by the scan counter decrementer, only one counter may be stepped during any one clock cycle.

Input and Output. Inputs to the scan counter latches are from the address sequencer, the check counter, the FLT counter, and the ROS test sequencer. Input is accomplished at not-clock time. The scan counter latch output automatically conditions the scan counter decrementer and, at the following clock time, the decrementer output is transferred into the source counter or sequencer. Inputs to the scan counter latches and to the source counter or sequencer are under control of the same 'not-clock P1' signal as shown in Figure 6-4. This signal resets the counter or sequencer before the decremented value is entered.

Scan Counter Decrementer. The scan counter decrementer (Diagram 8-104, FEMDM) is a logic network that subtracts 1 from any value routed through it. For example, assume that the address sequencer contains 21 (decimal) or 10101. When the address sequencer contents are to be decremented, they are transferred to the scan counter latches. The value is then routed through the decrementer, where it is reduced to 20 (10100) and sent back to the address sequencer.

Address Sequencer

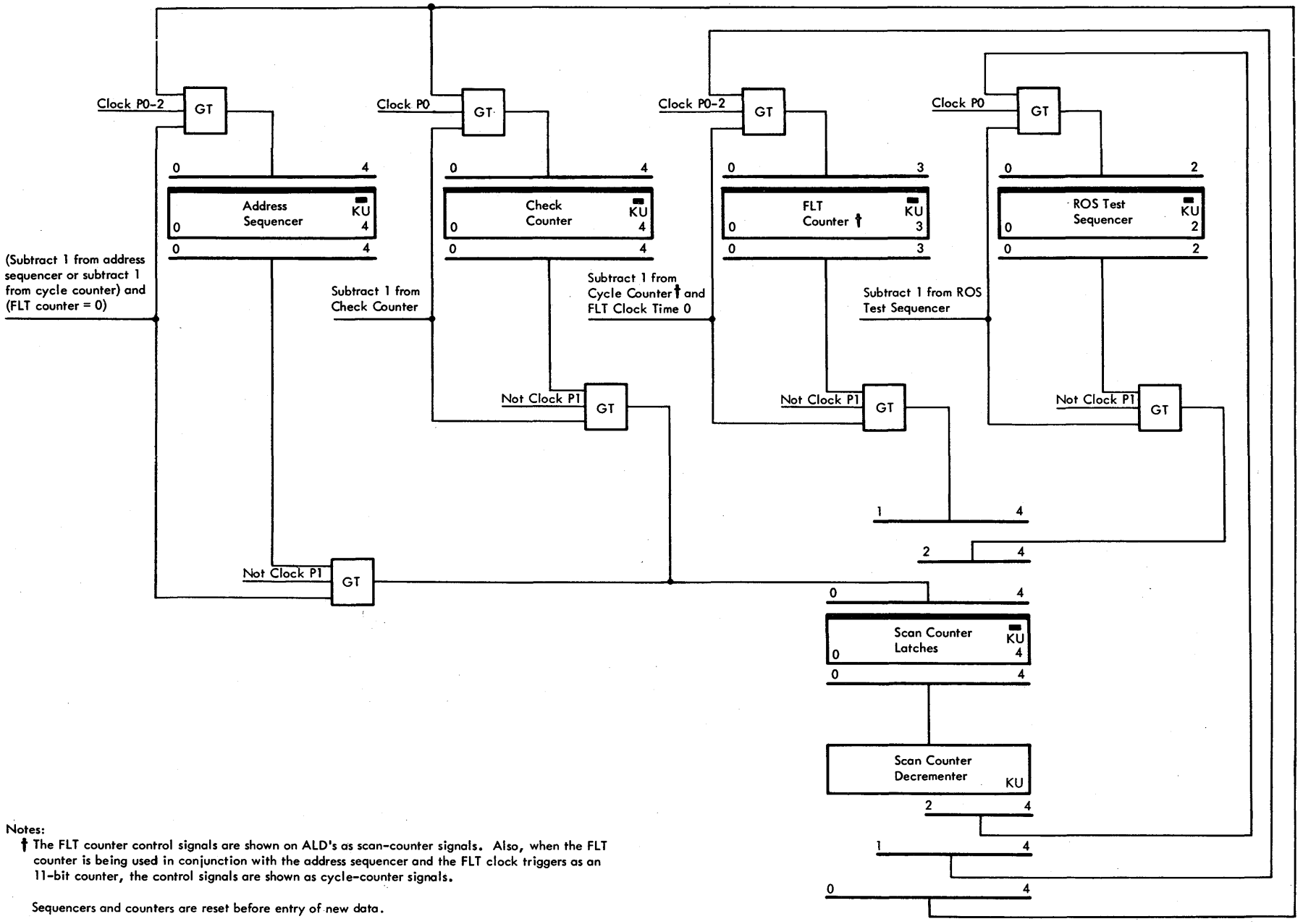
- Provides data used to generate main storage addresses; generates signals to select scannable triggers and latches on scan operations; functions as five high-order positions of cycle counter.
- Inputs are from T(53–57), hardware (set to 21 or 23), ROS micro-orders (set to 7, 8, or 21).

The address sequencer, a five-position trigger register, has two main functions: to sequence through a predetermined number of main storage addresses, and to select the scan address of the scannable triggers and latches to be scanned out to PAL. When the address sequencer is performing the first function, either a fixed address or a portion of the MCW is placed in it, and its output is transferred to a storage address generator which adds the necessary bits to make up the main storage address. The address sequencer contents are then decremented and transferred to the storage address generator to select the next address, and so forth. For the second function, the sequence is generally the same, except that the contents are transferred to the address sequencer decoder, which brings up 1 of 21 gating signals to scan out the proper scannable triggers and latches. Decrementing is accomplished by transferring the address sequencer contents to the scan counter latches and back again via the scan counter decrementer.

The address sequencer is also used as the five high-order positions of the 11-bit cycle counter during log-on-count or pulse-mode operation. This function of the address sequencer is discussed in "Cycle Counter".

The address sequencer (Figure 6-5) can be loaded with preassigned values by micro-orders or hardware, or T(53–57) can be transferred to it depending upon the operation. Its contents can be transferred to the address sequencer decoder for selection of scannable triggers and latches, to the storage address generator for addressing storage, or to the scan counter latches and decrementer.

During the scan-in portion of FLT's, the address sequencer is set to a count of 8 and then decremented by 1 for each test word fetched from main storage. The eighth word fetched (address sequencer equals 1) is the MCW part of the FLT test pattern and contains a new setting for the address sequencer. This new setting selects the scannable triggers and latches to be scanned out for the test comparison. The scan-out word is selected by decoding the address sequencer contents and the LW bit of the MCW. This action selects the roller switch indicating the exit trigger.



Notes:

† The FLT counter control signals are shown on ALD's as scan-counter signals. Also, when the FLT counter is being used in conjunction with the address sequencer and the FLT clock triggers as an 11-bit counter, the control signals are shown as cycle-counter signals.

Sequencers and counters are reset before entry of new data.

Figure 6-4. Scan Counter Latches and Decrementer Data Flow

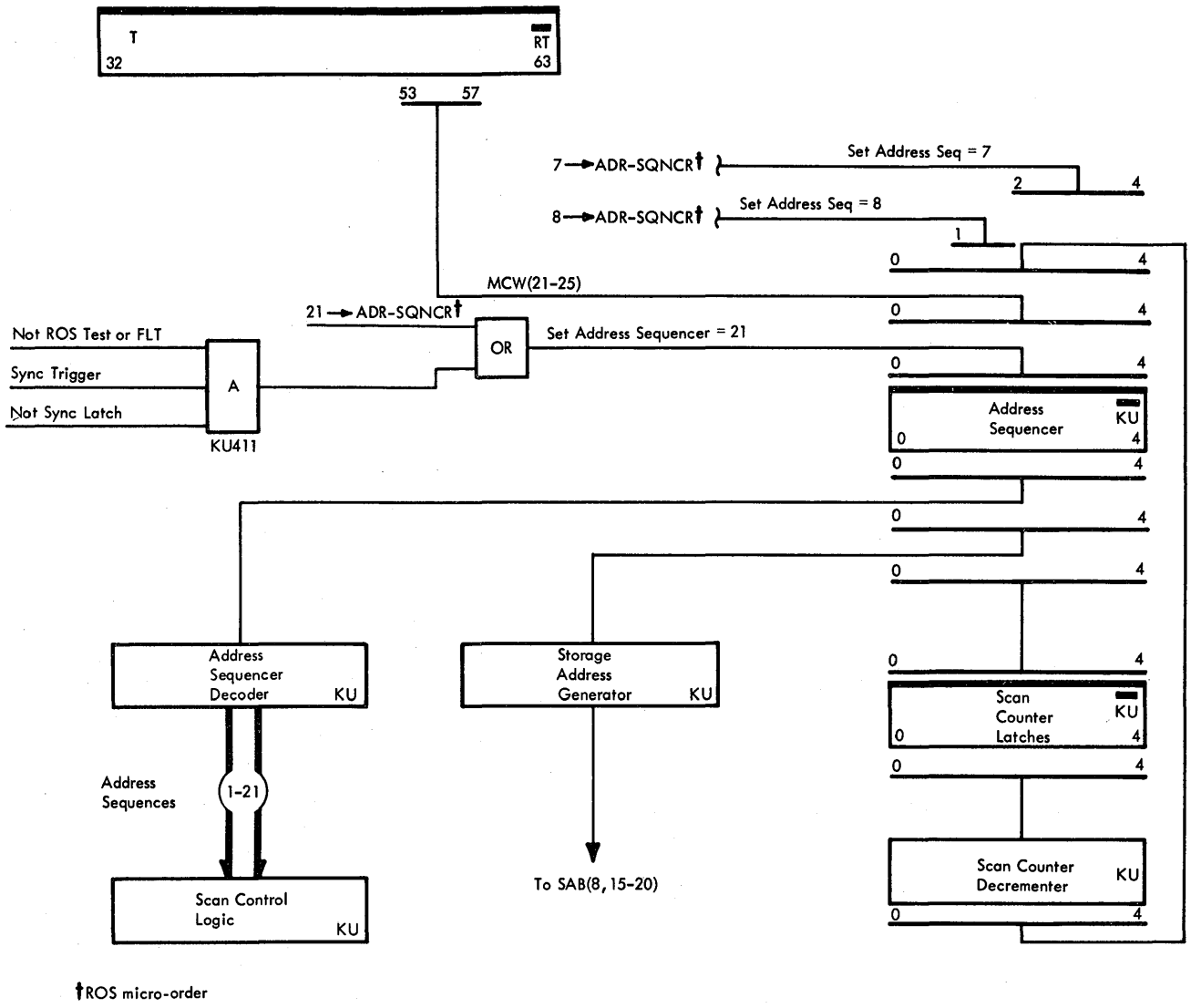


Figure 6-5. Address Sequencer Data Flow

The action taking place while the address sequencer steps from 8 to 0 is under control of the microprogram. When the count in the address sequencer is 0, the scan-in is complete for FLT's.

During a logout operation, the address sequencer is initially set to a count of 21 or 23 (decimal) and then decremented by 1 for each doubleword logged out. Again, the count in the address sequencer is decoded by the address sequencer decoder, and the logout word is selected using the indicator drivers as the source point. The address sequencer count is also decoded by the storage address generator to yield a main storage address for the logout word. This address is then put on the SAB before storing the logout word into main storage. While the sequencer steps from 21 (or 23) to 14, the CPU clock is turned off to prevent register operation. At a count of 13 (decimal), the CPU clock is again allowed to run, and the logout operation continues under ROS control. At count 0, logout is complete.

During logout, the address sequencer goes to 0 twice. The first time the count goes to 0, the sequencer is forced to a count of 21. This action recalls the first word logged out (word 21, ST contents) to correct possible incorrect parity. The word is then restored, and the sequencer is set to a count of 7 and again allowed to count down to 0, at which time the original parity bits are stored and the operation is completed. (Because no micro-order exists to set the address sequencer to 0, it is set to 7 and allowed to decrement to 0.)

Address Sequencer Decoder

The address sequencer decoder (Figure 6-6) interprets the output of the address sequencer during scan operations. The decoder consists of a high-order and a low-order bit section. The high-order bit section decodes the two high-order bits to yield, for example, 10XXX. The three low-order bits are then decoded to give, for example, XX101. The combination of the two, in this instance,

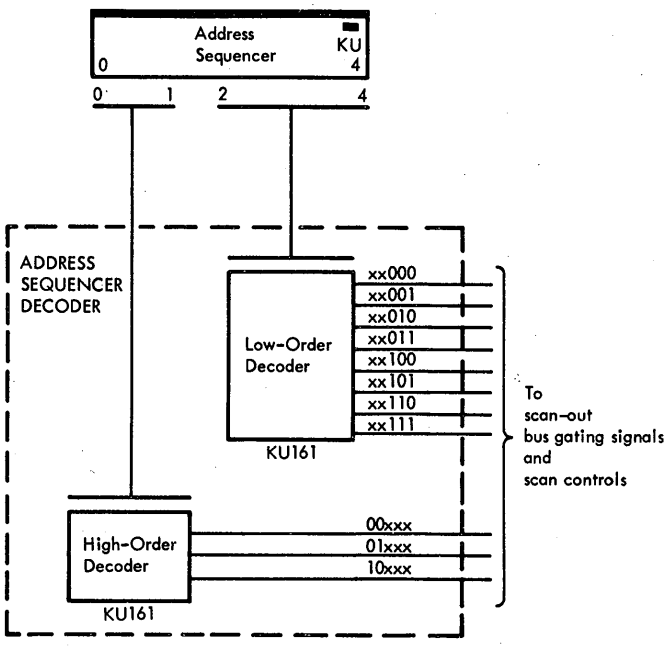


Figure 6-6. Address Sequencer Decoder

shows the address sequencer count to be 21 (10101). Thus, the word logged out or scanned out would be word 21, the contents of S and T.

The scan logic uses the indicator drivers, normally used for system control panel display, as scan-out source points. Therefore, the address sequencer selects the roller switch that contains the desired information. In the case where the count of the address sequencer is 21, the desired information is found in rollers 1 and 2, position 3 (S and T) (see "Scan-Out Bus").

Storage Address Generator

- Scan operations use three areas of main storage: logout area (80–128, hex), buffer 1 (8000–8048, hex), buffer 2 (8080–80C8, hex).
- SAB values to address location within the three areas are derived from address sequencer contents.

Three areas of main storage are used for scan operations: locations 80 through 128 (hex) is the logout area, which contains the log words after a log operation; and locations 8000 through 8048 (hex) and 8080 through 80C8 (hex) are the two areas that contain the FLT's or ROS tests. The SAB values necessary to address any location within these three areas are basically derived from the contents of the address sequencer. Forming a 24-bit address from the five bits of the address sequencer is accomplished by the storage address generator (Diagram 8-105, FEMDM).

The chart in Diagram 8-105 shows how the storage address is encoded. For all scan operations, bits 0–7, 9–14, and 21–23 of the storage address are set to 0's (bits 21–23 are 0's because logout words, FLT words, and ROS-test words are doublewords which are always on doubleword boundaries). Address sequencer bits 0 and 1–4 are transferred to SAB(15,17–20) to select the location within a scan area of storage.

In logout operations, bit 16 of the address is set to the complement of bit 0 of the address sequencer. Thus, for logout area addresses 80 through F8 (hex), SAB(15) is set to 0 and SAB(16) to 1; for addresses 100 through 128 (hex), SAB(15) is set to 1 and SAB(16) to 0. SAB(8) is always set to 0.

FLT's or ROS tests are always contained in two areas of storage called buffers. Buffer 1 is from 8000 through 8048 (hex), and buffer 2 is from 8080 through 80C8 (hex). When ROS tests or FLT's are being performed, each test, as it is read in, goes to the opposite buffer. For example, if the first test read from tape is placed into buffer 1, the next test is placed into buffer 2, the third into buffer 1, and the fourth into buffer 2, and so on. This scheme allows one test to be performed while the next test is being read into storage.

SAB(16) controls which buffer is addressed. When SAB(16) = 0, buffer 1 is addressed; when SAB(16) = 1, buffer 2 is addressed. During FLT's and ROS tests, SAB(16) is under control of the 'buffer 1' trigger. When this trigger is set, indicating that buffer 1 is to be addressed, SAB(16) is forced to 0. When the 'buffer 1' trigger is reset, indicating that buffer 2 is to be addressed, SAB(16) is set to 1.

During ROS tests, the 'buffer 1' trigger is complemented every time the ROS test sequencer equals 5. During FLT's, the 'buffer 1' trigger is under control of the 'ROS 31→CHK-CNTR' micro-order. If this micro-order is decoded and a test is in storage ('TIC' pulse received), the 'buffer 1' trigger is complemented.

For FLT's and ROS tests, SAB(8) is always set to 1 to address the portion of main storage starting at 8000 (hex).

SAB gating signals during scan operations are controlled by the 'scan sync' trigger. When this trigger is set, the contents of the address sequencer are transferred onto SAB. During FLT's and logout operations, the trigger is set by the 'ROS MS-REQ*SCAN4' micro-order. During ROS tests, it is set when the ROS test sequencer equals 1, 3, and 6.

Check Counter

The check counter, a five-position polarity-hold register, records errors encountered while reading FLT or ROS test tapes into storage and during scan in. At the start of the read-in operation, the check counter is set to 31 (decimal) (maximum count for a 5-bit register). It is then decremented by 1 each time an error is encountered. At the same time that the check counter is decremented, another attempt is made to read in the test that produced the error. After 32 errors (attempts at reading in the test), the check counter is stepped to 0, and the operation is terminated.

Decrementing is accomplished by transferring the check counter contents to the scan counter latches and back again via the scan decrementer.

Input and Output. The check counter (Figure 6-7) can only be set to 31 (decimal) or decremented via the scan counter decrementer; it has no other data inputs or outputs. It is set to 31 (decimal) if any of the following conditions occur:

1. Execution of the '31→CHK-CNTR' micro-order with the 'gap' trigger set.
2. ROS test sequencer equals 0 or 6.
3. ROS test sequencer equals 1 and the 'repeat' latch is set.
4. ROS test sequencer equals 4 and the 'pass' trigger is set, unless one of the following occurs:
   a. An input error is detected.
   b. The UT bit is set.

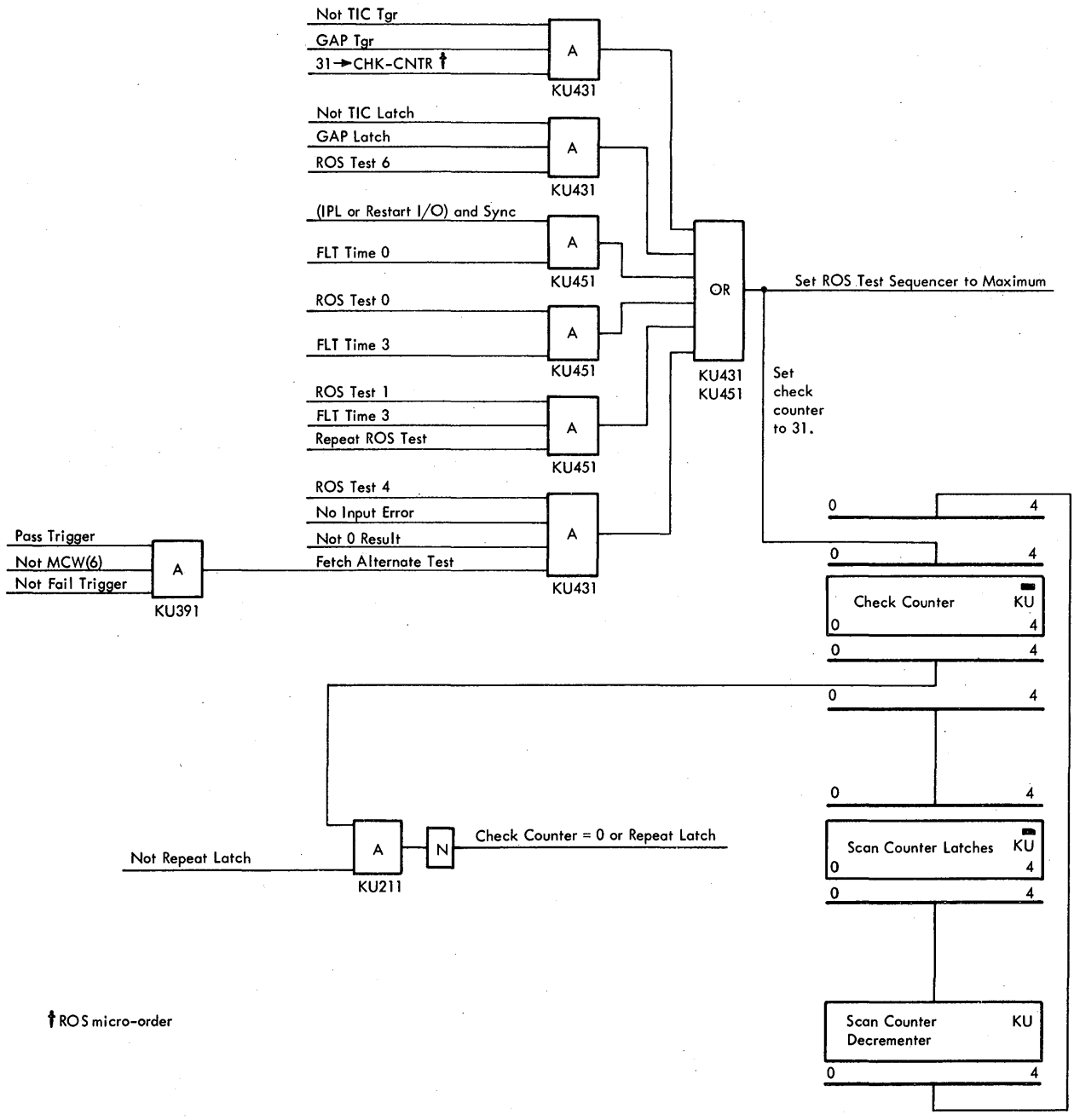


Figure 6-7. Check Counter Data Flow

c. The 'fail' trigger is set.
d. The test comparison in PAL resulted in 0.

5. An IPL is initiated or the test is restarted.

Check Counter Decrementing. The check counter decrementing controls, when activated, transfer the contents of the check counter to the scan counter latches and decrementer and back to the check counter; thus, the check counter is reduced by 1. During FLT's, the check counter is decremented by the 'ROS CHK CNTR-1' micro-order. During ROS tests, the check counter is decremented whenever an input error is detected.

FLT Counter

The FLT counter is a four-latch register/counter which is coupled with the two triggers of the FLT clock to make a six-position counter. Its main purpose is to count the number of cycles that the CPU is allowed to advance after scan in and before scan out. During FLT's and ROS tests, the FLT counter is set to the desired count by the cycle count field [MCW(26–31)]. During log-on-count and pulse-mode operations, the FLT counter (four bits) is combined with the address sequencer (five bits) and the two triggers of the FLT clock to make up an 11-bit cycle counter.

The two parts of the FLT counter are decremented differently. The low-order bits, FLT-clock 1 and 2, are a reverse binary counter that counts down from 3 to 0. Decrementing is accomplished by signals from the scan clock. Each time the low-order bits reach 0, the high-order bits, FLT-counter 0–3, are decremented via the scan counter decrementer in the same manner as the address sequencer and the check counter. The following scan clock signal resets the FLT-clock 1 and 2 to 3.

Input. The FLT counter (Figure 6-8) is loaded from T(58–61) or forced to the maximum count. Except during certain portions of the ROS tests, the MCW cycle

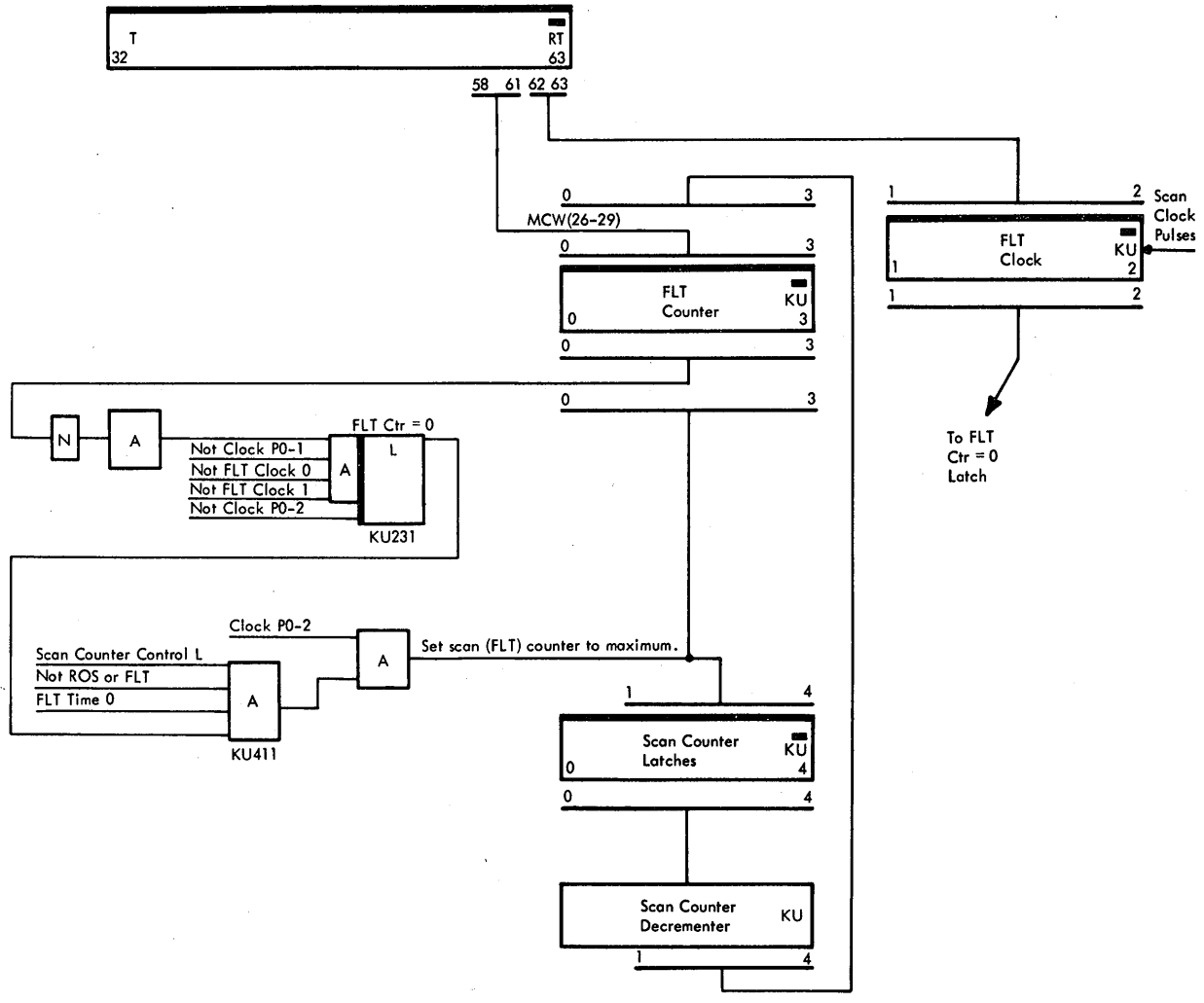


Figure 6-8. FLT Counter Data Flow

count field is gated from T(58–63) to the FLT counter and FLT clock by the 'ROS T→MCW' micro-order. During ROS tests, the cycle count field is gated from T to the FLT counter when the ROS test sequencer equals 2.

FLT Counter Decrementing. The FLT counter (Diagram 8-106, FEMDM) is decremented under control of two latches: the 'scan counter control' latch and the 'FLT counter equal 0' latch. Initially, the 'FLT counter equal 0' latch is reset, conditioning one leg of AND 1, the 'FLT clock-0' and '-1' triggers are set to some value (normally 3; see "FLT Clock"), and the 'FLT counter-0' to '-3' triggers are set to some value (either to maximum or to the value contained in the MCW cycle count field). When the FLT counter is to be decremented, the 'scan counter control' trigger is set which, in turn, sets the 'scan counter control' latch to bring up a second leg of AND 1.

For FLT operations, the 'scan counter control' trigger is set by the 'ROS 1→CTR CTL TGR' micro-order. For ROS tests, the trigger is set when the ROS test sequencer equals 2. A log-on-count-with-address-compare operation [Diagnose instruction with MCW(1) set] also sets the trigger if the storage address agrees with the ROS address field of the MCW.

With three conditions on AND 1 met (i.e., 'scan counter control' latch set, not SOROS and sync latch, and FLT counter not equal to 0), as soon as the FLT clock is decremented to 0 (decremented once per cycle), the FLT counter contents are decremented by transferring them to the scan counter decrementer and back again. At the same time, the FLT clock is reset to 3. When the FLT clock is again decremented to 0, the FLT counter is again decremented. This operation continues until the FLT counter, including the 'FLT clock-0' and '-1' triggers, are at 0 (reset). At that time, the 'FLT counter equal 0' latch is set, which inhibits further decrementing of the FLT counter.

Cycle Counter

During pulse-mode and log-on-count operations, the address sequencer is joined with the FLT counter and the two triggers of the FLT clock to form an 11-bit cycle counter (Figure 6-9). The low-order positions of the cycle counter are decremented as described in "FLT Counter Decrementing". When the FLT counter is decremented to 0 ('FLT counter equal 0' latch set), the address sequencer contents are decremented (gated to scan counter decre-

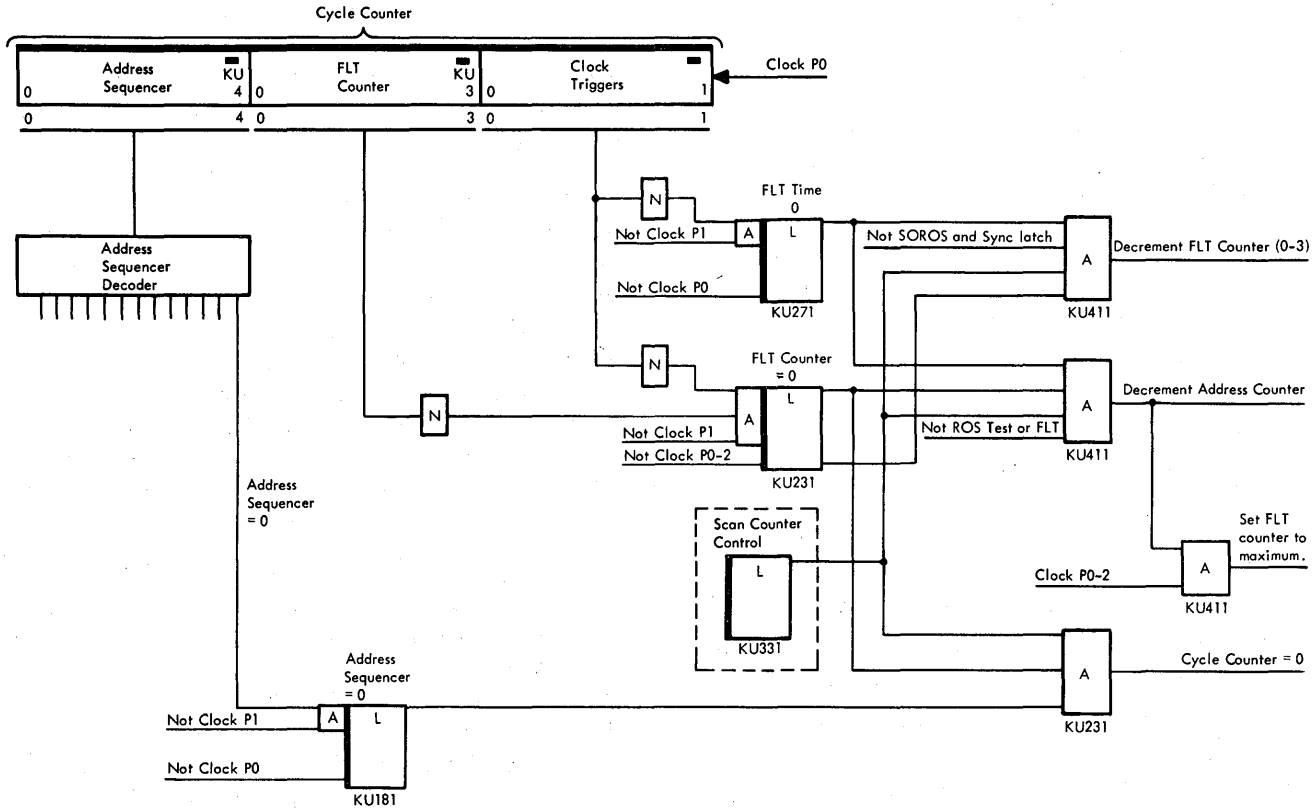


Figure 6-9. Cycle Counter Data Flow

menter and returned to address sequencer). At the same time, the 'FLT clock-0' and '-1' triggers are set, and the FLT counter is set to the maximum (all 1's). These counters are decremented as before until they reach 0 again; the address sequencer is then decremented, and the operation is repeated.

When all three counters equal 0, the 'cycle counter equals zero' signal is generated to stop the clock (pulse-mode) or to perform a logout operation.

ROS Test Sequencer

The ROS test sequencer (Figure 6-10), three polarity-hold circuits, controls the scan logic during a ROS test. At the start of the ROS test routine, the ROS test sequencer is set to a count of 7; it is decremented at FLT-time 3 by having its contents transferred to the scan counter latches and decrementer. The output is decoded by the ROS test decoder, which generates one of seven signals, depending upon the present value of the sequencer. These signals control the sequential operation of a ROS test.

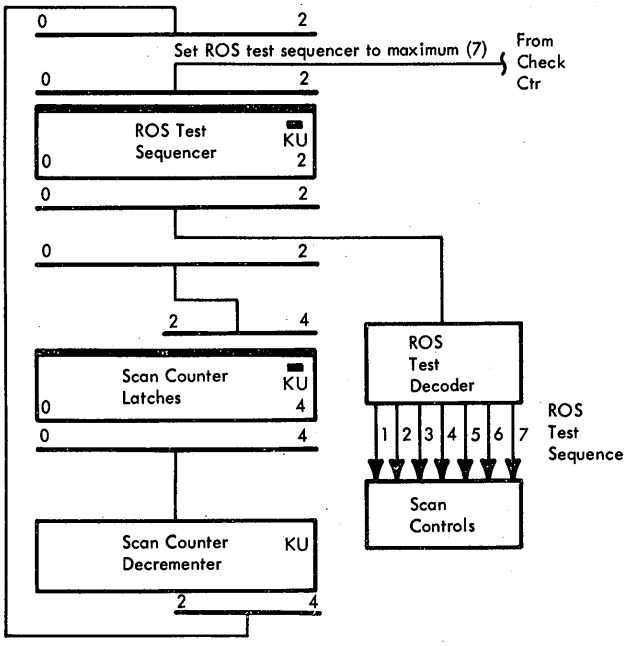


Figure 6-10. ROS Test Sequencer Data Flow

Scan-Out Bus

- Scan-out bus is data path from indicator logic to PAL.
- Status of 64 indicatable storage devices (scan-out word) is scanned out to PAL by one scan-out address.
- One half of scan-out word is transferred to PAL during one cycle.

The scan-out bus (Diagram 8-107, FEMDM) is a special data path used in scan operations that allows direct transfer into PAL from storage devices (triggers, latches, polarity-holds) which ordinarily have no direct input into PAL. The operation that performs this function is referred to as "scan out". Scan out makes use of the indicator roller switch position logic to perform this transfer. Thus, the first requirement necessary to scan out a storage device (except for the LSWR) is that the device is indicatable; that is, that is has an indicator on the system control panel roller switches.

During a scan out, the status of 64 indicatable storage devices are scanned out using one scan address (address sequencer contents); the 64 bits transferred to PAL are referred to as a scan-out word. The 22 scan-out words are listed on ALD's M3021–M3061 as logout words, because in a logout operation, the scan-out words are logged out (placed into the logout locations of main storage). Note that there is no direct correlation between the scan-out address and the roller switch position of the indicator. For example, in Diagram 8-107, scan-out address 21 is shown as selecting the same storage device as the 'roller 1 position 3' signal from the system control panel.

The indicatable storage devices to be scanned out at any one time are selected by the output of the address sequencer decoder and by a signal from the scan controls. Only one half of the scan word is transferred to PAL during one cycle; therefore the signal from the scan controls will be either 'scan out left' (selects bits 0–31 of a scan word) or 'scan out right' (selects bits 32–63). The output of the address sequencer decoder and the 'scan out left' or 'scan out right' signal places a signal on the same line as does the system control panel roller switch (Diagram 8-107). This signal allows the storage device output signal to light an indicator on the system control panel, or to be transferred to PAL via the scan logic.

At the same time, to prevent the roller switch setting from affecting the scan out, the 'not block indicator switches' signal is activated. Thus, the only indicatable storage devices whose outputs are sent to the indicator drivers are the ones selected by the address sequencer decoder output and the scan-out signal.

Gating signals route the indicator signals from the storage devices to two places: to indicators on the system control panel and to the scan-out bus. From the scan-out bus, they are transferred by the 'enable bus' signal to PAL(32–63).

Logout Controls. The controls for a logout operation are shown in Diagram 8-108, FEMDM. A logout is started by an 'error log required' signal, by depressing the LOG OUT pushbutton (in manual mode only), or by executing the Diagnose instruction. The 'SOROS' trigger sets the address sequencer to 21. The 'SOROS' trigger and the 'sync' latch request a storage cycle to store the logged out word. The FLT clock provides timing signals to control the logout operation until the address sequencer reaches 14; the CPU clock is then started and ROSAR is forced to 019 (hex) to control the rest of the logout operation.

Diagram 8-109, FEMDM, illustrates the scan-out path to PAL(54) on a logout operation. Assume that the address sequencer equals 10 (decimal). A scan out is initiated to transfer the contents of D(22) to PAL(54) if the 'scan out right' signal is generated. Normally, the 'scan out right' signal is generated by the 'SCAN OUT-RTWD' micro-order. However, on a scan-out ROS operation, it is generated by the scan controls. At the same time that the 'scan out right' signal is generated, the 'enable scan bypass' signal is produced. This signal is generated by the 'SCAN BYPASS' micro-order or, in the case of a scan-out ROS operation, by the scan controls. In addition to allowing transfer into PAL, the 'enable scan bypass' signal inhibits transfer to the system control panel indicators by the switchable indicator logic.

With the 'scan out right' signal active and the address sequencer equal to 10 (decimal), D(22) is transferred to indicator 25 of the row of indicators normally selected by roller 1 and to the scan out bus. Because the 'scan out S and T' signal is *not* up at this time, D(22) is transferred to PAL(54) by the 'enable scan bypass' signal. At the same time, the remainder of the right half of scan word 10 is transferred to the appropriate bits of PAL.

Because there are only four storage indicator roller areas and one logout area for a possible eight storage units, special provisions are made to log out the roller indicators of the failing unit. Diagram 8-110, FEMDM, shows the selection of the roller indicators of the failing unit. When an error signal is sent from storage 1, AND 1 is satisified, AND 2 is not satisfied, and the 'gate storage (1–4) indicators' signal is generated. Storage units 1–4 are conditioned to be indicated, but the storage unit check (in this case the 'storage 1 check' signal) generates the scan-out signal for the failing unit only. The storage frame indicators (always logged out) identify the particular failing unit and type of error (data or address).

Scan Out S and T. The scan out S and T operation is the method by which the mask in a ROS test or FLT selects the particular bit of the scan word to be tested. During a ROS test or FLT, the mask, which is set to all 1's except for the bit corresponding to the bit of the scan word to be tested, is placed into S(0–31), and the scan word containing the bit to be tested is placed into T(32–63). Using the scan-out bus (Diagram 8-109), the S and T contents are transferred by the 'scan out S and T' signal to a network composed of two AND's and an OR with an inverted output for each bit of S and T.

As can be seen from Diagram 8-109, the PAL bit positions remain 0's except for the position where both the input from S and the input from T are 0's. For example, if T(54) = 0 and S(22) = 0, PAL(54) is set. For any other combination of T(54) and S(22), PAL(54) remains reset. Because S contains a mask in which all bits are set to 1's except the bit that corresponds to the bit to be tested in T, all bits of PAL are always set to 0's except the bit to be tested, and that bit is also a 0 unless the scan word in T contains a 0 in the tested position.

The 'scan out S and T' signal is a result of the 'SCAN OUT S-REG' and 'SCAN OUT T-REG' micro-orders or the ROS test sequencer equalling 1 or 4.

Scan Stop-CPU-Clock Logic

During certain scan operations, the CPU clock must be stopped while the scan clock is allowed to run. This operation is controlled by the 'MMSC' trigger (Diagram 8-111, FEMDM). When this trigger is set, the unsymmetrical clock signal that controls CPU trigger and latch setting and resetting is inhibited; thus all CPU functions controlled by CPU clock signals are stopped, except as allowed by scan clock signals.

The 'MMSC' trigger is set by the following conditions:

1. Depressing LOAD with the TEST MODE, ROS/PROC/FLT switch in the ROS position.
2. Depressing RESTART FLT I/O.
3. Depressing LOAD. The system reset initiated by the IPL function resets the 'MMSC' trigger; thus, the CPU clock is restarted after the system has been reset.
4. Depressing LOG OUT.
5. Certain errors occur with the CPU CHECK switch in the STOP position.
6. Cycle counter equals 0 on a pulse-mode or log-on-count operation.
7. FLT counter steps to 0 during a ROS test or FLT.

Control Triggers

The scan logic contains many triggers which are used for status and control. The most important of these, with their function, are:

1. 'Restart I/O' (Diagram 8-112, FEMDM). Set whenever RESTART FLT I/O is depressed with the TEST MODE, ROS/PROC/FLT switch in either the ROS or FLT position. When set, this trigger causes the ROS or FLT tape to be backspaced to the beginning of the record. The buffers are then refilled by reading in each test until the test specified by the ATN is scanned into the CPU.
2. 'Scan mode'. Set when the scan controls are under microprogram control. The ROS fields used by the scan microprogram are shared by the non-scan functions of the CPU. They are interpreted as scan micro-orders if the 'scan mode' trigger is set.
3. 'Sync' (Diagram 8-112). Starts a ROS test or an FLT after an IPL or restart-I/O operation by synchronizing these operations with the scan controls.
4. 'Repeat'. Set by the RESTART FLT I/O pushbutton. When set, the ROS test or FLT currently running is repeated continuously.

5. 'FLT test' (Diagram 8-112). Set by the TEST MODE, ROS/PROC/FLT switch being in the FLT position. It puts the CPU into FLT mode.
6. 'Scan out ROS' (SOROS) (Diagram 8-108). Set by a machine error when in log-on-error mode, by the LOG OUT pushbutton, or by the 'cycle counter = 0' signal when performing a log-on-count operation. Initiates the scan-out portion of a logout operation.
7. 'Pass', 'fail', 'intermittent' (Diagram 8-112). Store the results of a ROS test or FLT. MCW(7) is the ERSLT bit, and PAL equals 0 if the bit to be tested in the scan word is a 1. Thus, if MCW(7) equals 1 and PAL equals 0, or if MCW(7) equals 0 and PAL is not equal to 0, the 'pass' trigger is set. If MCW(7) equals 1 and PAL is not equal to 0, or if MCW(7) equals 0 and PAL is equal to 0, the 'fail' trigger is set. For FLT's, the test is performed under microprogram control. For ROS tests, the test is performed when the ROS test sequencer equals 1. In either case, if both the 'pass' and 'fail' triggers are set, the 'intermittent' trigger is set.

Scan Mode Control of ROS

Scan mode operations affect three fields of ROSDR: field D (bits 17–19), field F (bits 25–30), and field G (bits 31–35). These fields serve dual functions. In the normal mode, they are decoded from the ROSDR latches as standard CPU control lines. In scan mode, they are decoded as special scan control lines and are referred to as field S.

The scan mode is controlled by the 'scan mode' trigger. When the 'scan mode' trigger is reset, the standard decode path is used. When the 'scan mode' trigger is set, however, the standard control lines are blocked and scan control lines (using common CPU control line codes) are activated.

The 'scan mode' trigger can only be set in normal CPU mode and reset only in scan mode. The scan control logic generates blocking signals to inhibit register gating signals at the ROSDR decode logic and to allow scan control use of ROS in sequencing through its test operations.

Scan logic also affects ROS microbranching (Diagram 8-113, FEMDM). The J-field micro-orders shown in the diagram are listed on ALD M7021.

Scan/Channel Interface

Because ROS tests and FLT's are read in from an I/O device (tape or disk), special interface lines (Diagram 8-114, FEMDM) are necessary for communication between the channel and the scan logic. A brief description of these lines follows:

1. 'TIC pulse'. A multiplex line from the channels to the scan controls that carries a 250 ±30-ns signal when a 'TIC' command is encountered after the 'IPL' latch is reset. This signal indicates a buffer has been filled.
2. 'Gap pulse'. A multiplex line from the channels to the scan controls that carries a minimum 1-usec signal whenever an end of record is passed. This signal does not occur if an error was detected in the record.
3. 'Unit data error'. A multiplex line from the channels to the scan controls that signals a data error from the time of discovery until a restart is initiated.
4. 'Channel control error'. A multiplex line from the channels to the scan controls that rises on a control error, command reject, or exceptional condition, and signals that the channel is unable to proceed.
5. 'Stop scan'. A multiplex line from the scan controls to the channels. This line carries a 250 ±50-ns signal that commands the operating channel to stop transmission to storage and wait. The tape or disk proceeds to the end of record and is deselected.
6. 'Start scan'. A multiplex line from the scan controls to the channels. This line carries a 250 ±50-ns signal that commands the selected channel to restart transmission of data to storage. The channel fetches the command at location 80 (hex) and continues. This action causes the restart to begin always at the front of the block it was transmitting at the time the 'stop scan' signal was received. If the 'stop scan' signal is received during an error backspace, the backspace is concluded and the ensuing 'start scan' signal initiates the read command that loads buffer 1.
7. 'Scan mode'. A multiplex line that indicates to the channel that the CPU scan controls are operative. It causes the channel to go into FLT mode when it receives the 'IPL' signal.

Operational Analysis

The following paragraphs present a detailed operational analysis of logout, ROS tests, and FLT's.

Logout

- Logout stores 22 doublewords, which reflect CPU status, into main storage locations 80–128 (hex).
- Operation is controlled by scan logic and ROS microprogram.
- Address sequencer is set to 21 and decremented by 1 for each log word stored.

The logout function, whether initiated by depressing the LOG OUT pushbutton, by executing the Diagnose instruction, or by detecting a machine check, stores 22 doublewords (log words), which reflect the CPU status, into main storage locations 80–128 (hex). The information presented in these log words is determined by the scan logic and is therefore fixed at a specified engineering level for any particular 2065 CPU. A list of the log-word locations and their contents is presented in ALD's M3021–M3061.

The words are logged out in a fixed sequence, as defined by the scan logic, with log word 21 being stored into location 128 (hex), log word 20 into location 120, and so on in reverse order until log word 0 is stored into location 80 (hex).

The logout operation is both hardware- and ROS-controlled. When a logout operation is initiated, for example by depressing the LOG OUT pushbutton†, the address sequencer is set to 21 (23 on a storage logout) and is decremented by 1 for each log word stored. From the instant that LOG OUT is depressed until the address sequencer is reduced to 14, the scan hardware controls the operation. When the address sequencer equals 13, control is switched to a ROS microprogram which completes the logout.

Although the CPU is under ROS control for the last 14 words logged out, the address sequencer is still decremented for each log word. This decrementing occurs because the address sequencer, in addition to controlling the logout sequence, defines the storage address for each log word and the indicatable storage devices to be logged out.

The log words compose the status of most indicatable storage devices (i.e., triggers, latches, and registers that have an indicator on the roller switches). For this reason, the scan-out bus, which is a data path from the indicator drivers to PAL, is used to transfer the status of these devices to PAL (the path used is identical to that used in an FLT or ROS test during a scan out; see "Scan-Out Bus"). From PAL, the log words are gated to ST, and then to main storage via the SDBI. Correct parity is assigned in PAL.

Because cycling of the CPU clock could change the contents of some of the storage devices logged out, the CPU clock is turned off (blocked) while log words 21 through 14 are being logged out, and timing is controlled by the scan and FLT clocks. Scan/FLT clock signals are distributed throughout the scan logic to control the logout function during the period that the CPU clock is turned off. For example, the address sequencer is decremented at FLT-time 3, a latched output of 190-ns duration.

At the end of a logout, the CPU performs an end op, and the 'machine check interrupt' trigger is set.

If the logout is caused by a storage error, all storage units are disabled because the CPU cannot determine which storage had the error. Therefore, no storage unit can be recycled until the contents of its indicators have been saved. As the first step in saving the indicators, the address sequencer is set to 23. The two extra counts of the address sequencer provide a delay which is necessary because of the cable delay and of the relatively slow indicator drivers in storage and the CPU. When the sequencer reaches 21, the storage units are again made available, and the contents of the indicators displaying information pertaining to the faulty main storage are transferred to PAL and held in PAL(32–63). The remainder of the logout sequence is the same as for a machine check logout.

Hardware-Controlled Sequence. Highlights:

- Logout is initiated by Diagnose instruction, LOG OUT pushbutton, or machine error.
- 'SOROS' trigger initiates hardware-controlled portion of logout sequence.
- For each word logged out: (1) address sequencer is decremented, (2) scan-out logic places right half of log word into T, and (3) log word is stored into main storage location addressed by storage address generator.
- Only right half of log words 20–14 contain data.

As stated previously, a logout (Diagram 8-115, FEMDM) may be initiated by executing the Diagnose instruction, by depressing LOG OUT, or by detecting a machine error when in log-on-error mode. The operational differences between the three methods occur before the actual logout sequence and are as follows:

1. The Diagnose instruction initiates a logout operation if MCW(6) = 1, thus specifying a log-on-count operation. In this case, MCW(21–31) is sent to the cycle counter, which is decremented by 1 each machine cycle. When the cycle counter equals 0, the 'SOROS' trigger is set, initiating a logout sequence.
2. If the logout operation is initiated by depressing LOG OUT with the CPU in manual mode, the 'console logout' latch and the 'pass pulse' trigger are set. Normally, the 'pass pulse' trigger will already be set. However, if the RATE switch is in the SINGLE CYCLE position, the 'pass pulse' trigger remains set only for the duration of one CPU clock signal (Diagram 4-3, FEMDM). This action allows the logout operation to be stepped through one cycle at a time by depressing START for each cycle. With the 'pass pulse' trigger and 'console logout' latch both set, the 'SOROS' trigger is set.
3. When the CPU CHECK switch is in the PROC position and the PSW machine check mask bit is a 1, a machine check ('error' trigger set) initiates a logout. Any of the following errors sets the 'error' trigger (which, in turn, sets the 'SOROS' trigger):
   a. CPU/storage error.
   b. Serial adder half-sum or full-sum error.
   c. Parallel adder half-sum or full-sum error.
   d. ROS error.
   e. E(0–15) parity error.
   f. MPR bus parity error.

† Note that the LOG OUT pushbutton can initiate a logout only when the CPU is in the manual mode.

When a storage check is detected by the processor, the following sequences occur:

a. The BCU resets any CPU request triggers that might have just been set. If the request triggers were set early enough to have sent a 'select' signal to storage, they are reset in a normal fashion and are, therefore, not reset by the storage check sequence.
b. All storages are made to look busy by blocking the set input to the 'not busy' triggers. This action prevents any channel from selecting the faulty storage unit.
c. The CPU 'stop clock' trigger is held reset. Because the storage check is asynchronous to any other BCU activity, this trigger might have been set by some other CPU 'select' signal.

All of these conditions persist until the scan logic signals release to the BCU. At the same time, the address sequencer is set to 23 instead of the normal 21. This setting allows a delay of approximately 1.6 usec until the count is reduced to 21. A 'scan-out storage X' line is activated to transfer the contents of the storage indicators to PAL(32–63). During this delay, the address or data check indicator stabilizes (worst case is 1 usec) and provides the proper gating signal level for its respective roller switch; e.g., storage 1 is roller 1, position 5; storage 4 is roller 6, position 5. When the address sequencer equals 21, the contents of the proper roller switch is transferred to PAL. Just before dropping the gating signal level, a 'hold paddl 32–63' line is activated to allow the input gates to be deactivated without losing the storage indicator information. At this point, the BCU is released by the scan logic. Any channel requests that were held pending in the BCU start being serviced in a normal manner, and a scan storage request for storing ST is issued. The remainder of the logout sequence is identical to a machine check logout sequence, except for the gate PAL to T and the set marks functions when the address sequencer equals 20. (Normally, these signals are blocked.)

Note that, in each case, the 'SOROS' trigger is set, thus starting the hardware portion of the logout sequence (Diagram 8-115). After the 'SOROS' trigger is set, the 'sync' trigger is set at FLT-time 3 to synchronize the logout operation to the FLT clock. The output of the 'sync' trigger sets the address sequencer to 21 (first log word), and sets the 'sync' latch at not-clock time. (The 'sync' latch being set causes the address sequencer to be decremented every FLT-time 3.)

With the 'sync' latch set, a storage request to the BCU is initiated at FLT-time 0. This request gates the address generated by the storage address generator to SAB. Because the first word to be logged out reflects the contents of ST (log word 21), the contents of ST are now stored (gated to SDBI).

When the 'sync' trigger was set, the 'MMSC' trigger was also set. This action keeps the CPU clock off (the 'error' trigger inhibited CPU clock signals) but allows the scan clock to run. Therefore, the status of all operational registers, except the registers used for the logout (S, T, and PAL), is preserved during the logout sequence.

Because the contents of ST have been stored, the address sequencer is decremented to 20 at FLT-time 3. A scan-out operation is performed, which gates the right half of log word 20 (selected by the address sequencer decoder) to PAL(32–63) from the indicatable storage devices via the scan-out bus. From PAL, log word 20 is gated to T for transfer to main storage. At FLT-time 0, another storage request is issued to store the contents of ST. This operation continues for log words 19 through 14; that is, for each word logged out: (1) the address sequencer is decremented, (2) a scan out places the right half of the log word into PAL, which is subsequently gated to T, and (3) the log word is stored into the main storage location addressed by the storage address generator. Note that only the right half of log words 20 through 14 is stored; the left half in storage remains unchanged and could contain anything.

When the address sequencer has been decremented to 14 and the log word has been stored, the 'MMSC' trigger is reset to start the CPU clock, address 19 is forced into ROSAR, and the address sequencer is reduced to 13. (Note that the ROSAR contents have been already logged out.) At this point, control is transferred to a ROS microprogram.

ROS-Controlled Sequence. Highlights:

- ROS microprogram directs logout from time address sequencer equals 13 until end of logout sequence.
- Log word 13 is formed by: (1) scanning out to T, (2) transferring T to LSWR, (3) transferring S to T, (4) transferring LSWR to S.
- Remaining words are formed by: (1) decrementing address sequencer by 1, (2) scanning out to T, (3) transferring T to LSWR, (4) transferring S to T, (5) transferring LSWR to S, (6) storing log word into storage.
- When address sequencer equals 0, parity is corrected on log word 21 (ST).

The ROS microprogram directs the logout operation from the time the address sequencer equals 13 until the end of the logout sequence (Diagram 8-115). The first micro-order in the logout microprogram transfers the contents of the LSWR to S, thus preserving the LSWR contents in a log word because the LSWR is used during the remainder of the operation. S now contains half of log word 13. The

complete log word is formed by: (1) scanning to T (scan out to PAL whose contents are transferred to T), (2) transferring the data in T to the LSWR, (3) transferring the contents of S to T, and (4) transferring the contents of the LSWR to S. As a result, ST contains the 13th log word. The operation continues in the following cycle:

1. Decrement address sequencer by 1.
2. Scan out to T (left half).
3. Transfer contents of T to LSWR.
4. Scan out to T (right half).
5. Transfer contents of LSWR to S.
6. Store word into main storage.

This cycle is repeated until the address sequencer equals 0, at which time 22 log words have been stored into main storage locations 80–128 (hex).

When the logout operation was started, the first word stored into main storage (log word 21) was composed of the contents of ST. The parity associated with this data is not always correct. Therefore, to insure that correct parity is stored with log word 21, the address sequencer is set to 21, and a scan storage request is initiated to fetch the original contents of ST in log word 21. When available on the SDBO, the word is gated to AB. The address sequencer is then set to 7, and AB (scan address 7) is scanned out. When AB is scanned out, correct parity is inserted for the data (original ST contents). The address sequencer is again set to 21, and a scan storage request is initiated to put log word 21 back into main storage with correct parity. Next, the address sequencer is reset to 7 and repeatedly decremented by 1 until it equals 0, at which time the original ST parity bits are stored. The scan system is then reset, the 'machine check interrupt' trigger is set, and the operation concludes at end op.

ROS Tests

- ROS tests consist of two doublewords: word 0 has TN and ATN; word 1 has mask and MCW.
- All bits of ROS bit planes are checked for 1 or 0.
- Each ROS test is repeated until receipt of 'TIC' pulse.

ROS tests are scan-controlled tests of the ROS microprogram. Each ROS test consists of two doublewords, designated words 0 and 1, which are read into the CPU from the ROS test tape. Word 0 contains the TN and ATN, and word 1 contains a mask and the MCW. A single ROS test checks one bit position of one ROS word and the cycling of data from ROS to the ROSDR.

Each ROS test tape begins with hardcore tests to check out the hardware that controls subsequent ROS tests. Upon successful completion of these hardcore tests, the true ROS tests are begun and continue until all bits of the ROS bit planes have been checked for a 1 or 0, or until an error is encountered. If an error occurs, the CPU stops and the failing bit number is displayed in S. Thus, no documentation is required.

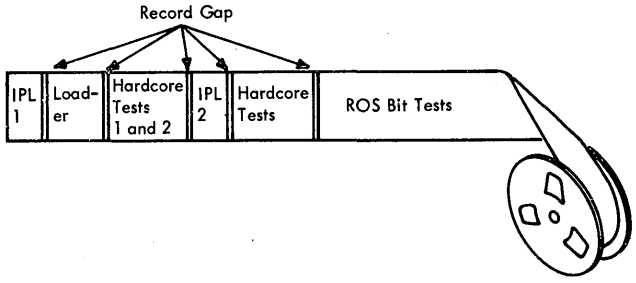
The tests are loaded into buffer areas 1 and 2, and 'TIC' pulses are generated as the buffers are filled. Each ROS test is continuously repeated until the CPU receives the 'TIC' pulse.

A single test consists of fetching the MCW from storage, permitting the CPU clock to advance a given number of cycles, and comparing one of the ROSDR triggers with an expected result. If the state of the trigger matches its predicted value, the 'pass' trigger is set and testing proceeds to the next sequential test. If the actual and predicted values disagree, the 'fail' trigger is set and testing is terminated.

The ROS tests are controlled by the ROS test sequencer, which is stepped by the scan clock. At the start of the ROS test, the ROS test sequencer is set to 7 and is decremented to 0 as the test progresses to conclusion. For each count of the test sequencer, a certain part of the ROS test is performed.

The number of clock signals required to move the selected ROS word into the ROSDR is specified in the cycle count field of the MCW. This count is set into the FLT counter; when the counter equals 0, the CPU clock is stopped and the bit comparison begins.

ROS Test Tape. The ROS test tape contains the following records:



1. Record 1, IPL 1. Contains the 24-byte "bootstrap" program necessary for any IPL operation. When LOAD is depressed, the three doublewords of this record are read into storage locations, as follows:

Storage Location (Hex)	Word	Contents
0	PSW	Backspace (not used).
8	CCW 1	Read command to read 40 bytes to location 80, and chain command to location 10.
10	CCW 2	TIC command to location 88.

2. Record 2, Loader. Contains the "loader" program that reads in the ROS tests. The IPL program in record 1 reads this record into storage locations as follows:

Storage Location (Hex)	Word	Contents
80	CCW	Backspace command.
88	CCW	Read command to read 16 (decimal) bytes to buffer 1 (location 8000, hex), and a chain data tag to location 90.
90	CCW	TIC command to location 98.
98	CCW	Read command to read 16 bytes to buffer 2 (location 8080), and a chain data tag to location A0.
A0	CCW	TIC command to location 88.

3. Record 3, Hardcore Test 1 and 2. Contains the first ROS hardcore test, two doublewords. Hardcore tests check the scan and CPU hardware required to do the actual ROS testing. Any failure encountered during the hardcore tests must be corrected before the actual tests have validity.
4. Record 4, IPL 2. The first hardcore test causes a stop. When LOAD is depressed, the following IPL program replaces the IPL 1 program:

Storage Location (Hex)	Word	Contents
0	PSW	Backspace record.
8	CCW	TIC command to location 88 (loader program).

5. Records 5–8, Hardcore Tests. Contain the remaining hardcore tests.
6. ROS Bit Tests. The remaining records on the tape contain the true ROS tests. Each test pattern (two doublewords) tests one bit of one ROS word.

ROS Test Setup. Several controls on the system control panel must be operated to initiate the ROS tests. The procedure to run a ROS test appears in *2065 Processing Unit FEMM*, Form Y27-2039-2 and in ALD M8005. However, a short discussion of the setup is included here because it affects the operation.

Diagram 8-116, Sheet 2, FEMDM, shows the start of a ROS test. The ROS test tape is mounted first. The LOAD UNIT switches are set to the address of the tape unit holding the test tape, the TEST MODE, ROS/PROC/FLT switch is set to the ROS position, and the CPU CHECK switch is set to DSBL. Going into ROS test mode causes the 'ROS test initiated' latch to be set and a 'scan mode' signal to be sent to the channel to prepare for a read operation. The manual control operations necessary to get started are concluded by depressing SYSTEM RESET, loading all DATA switches to 1's, depressing STORE to transfer the DATA switches to ST, and depressing LOAD.

IPL 1. Highlights:

- IPL is under hardware control.
- Channel operations are same as normal IPL.
- CPU clock is stopped until release is received from channel.
- 40 bytes are read into storage, starting at location 80 (hex).

Depressing LOAD with the TEST MODE, ROS/PROC/FLT switch set to ROS initiates an IPL operation that is different from the normal program load in that the operation is under hardware control (Diagram 8-8, FEMDM). Channel operations are identical to a normal IPL operation; 24 bytes are read from the selected device into the first three doubleword locations of main storage. However, when the 'IPL status' trigger is set, the 'MMSC' trigger is also set, which stops the CPU clock. Because the CPU clock is stopped, the IPL microprogram is not initiated and the remainder of the IPL is under scan and channel control.

At this point, the CPU is idle, waiting for a 'release CPU' signal from the channel; only the scan clock is running. Meanwhile, the channel IPL operation reads in record 1 of the ROS test tape, and executes the channel program specified by record 1. As a result, 40 bytes (record 2) are read into storage starting at location 80 (hex). Record 2 contains the loader program that reads each ROS test into the proper buffer area in storage.

After record 2 has been read in, command chaining causes CCW 2 in record 1 to be executed. CCW 2 is a TIC command to location 88, which now contains a read CCW. When the channel has finished executing the TIC command, it sends a 'release' signal to the CPU.

Because the CPU is in the ROS test condition, receipt of the 'release CPU' signal from the channel causes the 'timing gate' trigger to be reset. With the 'timing gate' trigger reset, the 'release CPU' latch is reset and the 'IPL status' latch is set. The 'sync' trigger and 'sync' latch are set to synchronize the operation to the FLT clock. In addition, the 'sync' trigger output resets the ROS sense latches and inhibits register ingating. At this point, channel selection is dropped and a scan reset is initiated. The scan controls are set up for operation by resetting the 'MMSC' trigger, setting the 'pass' trigger, resetting the 'fail' trigger, and setting the check counter to 31 and the ROS test sequencer to 7. The CPU is now ready to run the first hardcore tests.

Loader. While the scan controls are being set up to run the first hardcore test, the TIC command in CCW 2 of the IPL 1 program causes the channel program in record 2 of the ROS test tape [now in the 40 byte locations of storage beginning at 80 (hex)] to be executed. The read CCW's in locations 88 and 98 cause the channel to read the hardcore tests in record 3 into buffers 1 and 2 (8000 and 8080, respectively).

Meanwhile, scan control operations in the CPU begin when the ROS test sequencer is set to maximum, thus placing the CPU into ROS test state 7. In state 7, the 'buffer 1' trigger is reset, the 'ROS test' trigger is set, and a scan system reset clears the scan IPL controls. The ROS test sequencer is decremented by 1. During ROS test state 6, the 'sync' latch is reset, an address is forced to storage controls, and a scan storage request for the ROS test word containing the TN and the ATN word is initiated. The operation then waits for a 'TIC' pulse before progressing. This 'TIC' pulse results when the first hardcore test in record 3 has been read into storage from the channel (execution of TIC CCW in location 90). After the 'TIC' pulse, the ROS test sequencer is decremented by 1.

At this time, an ATN search is made. The test number, all 1's, from the test buffer is transferred to the S-register, and a successful search should result.

Hardcore Tests and IPL 2. Highlights:

- Eight records of hardcore tests at beginning of FLT and ROS test tapes determine that CPU hardware used to run tests is functioning.
- Failing test stops, with failing bit pattern in S.
- If hardcore tests run successfully, testing terminates on correct stop; when testing is resumed, CPU enters true ROS tests or FLT zero-cycle tests.

Every FLT and ROS test tape begins with hardcore tests (see ALD M8006) to determine that the CPU hardware used in running these tests is functioning properly. These tests are almost identical for ROS tests and FLT's; differences will be pointed out. Because the CPU hardware most involved in running FLT's or ROS tests is S, T, PAL, and the connecting paths between these points, these are the logic areas checked by the hardcore tests.

Eight records of tests are involved with hardcore testing. Records 1, 2, and 4 load the CPU for testing; the others check the CPU for various functions or capabilities. These functions include sensing for 1's or 0's in S and T, verifying the CPU's ability to stop when the stop conditions are met, and verifying the ability of the CPU to conduct an ATN search and sequential testing. A failing test stops testing, with the failing test bit identification pattern displayed in S and T. If all hardcore tests are run successfully, testing terminates on a hardcore stop (correct stop); when testing is restarted, the CPU enters the true ROS tests or, in the case of FLT's, the zero-cycle tests.

Because the ROS hardcore tests are sequenced by the same controls that sequence the ROS tests and because the FLT hardcore tests are executed in the same manner as the regular FLT's, the hardcore tests have a different format for ROS tests and FLT's. ROS hardcore tests have the following format:

		Buffer 1	Buffer 2
TN	ATN	8000	8080
Mask	MCW	8008	8088

Each FLT hardcore test contains the nine or ten words used in an FLT; however, only two of the words are significant to the test. These are words 8 or 9, and 1, the TN/ATN and the MCW, respectively. The TN/ATN causes the CPU to progress through the tape records. The MCW causes the CPU to make decisions. The other seven words read in during a hardcore test contain 0's and contribute nothing to the test.

Hardcore tests check S, T, and PAL by performing the TN/ATN comparison. The TN is in S, and the ATN from the previously executed test is in T. The corresponding bits in each register are compared at the scan-out-bus OR. If either bit is a 1, the inverted output of the OR is sensed as a 0.

For the result comparison, the mask is brought to S and compared with the value scanned into T during execution of the test. In hardcore tests, the mask is all 1's or all 0's. This condition forces a pass or fail condition regardless of the value scanned into T during execution. A 0 output is sensed if the mask is all 1's; a positive output is sensed if the mask is all 0's. Using the mask in this manner, in conjunction with the MCW, causes the CPU to decide whether to take the next test, an alternate test, or to terminate testing.

The significant bits of the MCW used during hardcore testing are 5, 6, and 7. MCW(5) is the UT bit, and, if set, causes the CPU to stop after the test, regardless of the outcome of the test. MCW(6) is the CT bit, and, if set, causes the CPU to stop if the test fails or to take the next test if the test passes. MCW(7) is the ERSLT bit, which specifies whether a 0 or a 1 should be sensed at PAL following the result-comparison portion of the test. The combinations of the ERSLT and the output at PAL that determine the setting of the 'pass' or 'fail' trigger are:

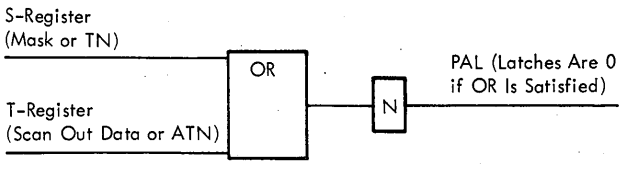
ERSLT Bit	PAL = 0	Set 'Pass' Tgr	Set 'Fail' Tgr
0	Yes	–	Yes
0	No	Yes	–
1	Yes	Yes	
1	No	–	Yes

The setting of the 'pass' or 'fail' trigger is then compared with the setting of MCW(5, 6) to determine the next operation, as follows:

Trigger Output MCW(5) (UT)	MCW(6) (CT)	'Pass' Tgr	'Fail' Tgr	Action
0	0	1	0	Continue – alternate test
0	1	1	0	Continue – next test
0	0	0	1	Continue – next test
0	1	0	1	Stop – gate alternate test on restart
1	0	1	0	Stop – gate alternate test on restart
1	0	0	1	Stop – gate alternate test on restart

Whenever the CPU stops during hardcore tests (whether an error stop or an unconditional stop), the ATN is left in T. When the test is restarted, the tape backspaces to the beginning of the test record and begins reading into storage. As each test is brought in, the CPU brings the TN of each sequential test into S. S and T are then compared, and, when PAL equals 0, the CPU performs the test that is in the buffer at that time.

One of the preliminary steps in preparing the CPU to perform ROS tests or FLT's is to set S and T to all 1's by setting the DATA switches to the down position and depressing STORE (see ALD M8005-1). This step is required because it is not known initially that these registers are functioning. By forcing 1's into all positions, the CPU is forced to take the first test, which is test 1 on record 3. The rationale is that a failing bit in one register is compensated for by the bit in the other register, so that when S and T are compared, the result is 0 and the test is taken. Shown below is a single position of the scan-out bus; a similar position exists for each bit in S and T.



Note that a 1 input to either side of the OR satisfies the OR condition, and the inverted output is sensed as a 0. If neither input is a 1, the OR is not satisfied, and the inverted output is sensed as a 1.

Test 1, Record 3. Highlights:

- Determines that 1's in S can be sensed as 0's at PAL.
- If test passes, CPU stops with 'pass' trigger set.
- If test fails, CPU stops with 'fail' trigger set or does not stop.
- Depressing RESTART FLT I/O on passing test repeats test. If restart is initiated on failing test, "runaway" tape may occur.

Note: This test must be taken because of setting S and T manually (assuming data failures only).

This test is the first of the hardcore series and has the following format:

Mask (S)	MCW (T)	TN (S)	ATN (T)
FF FF FF FF	05 00 00 00	FF FF FF FF	00 00 00 00

The MCW defines the conditions of the test, and in this instance the MCW has the configuration, in the first eight bits, of 0000 0101. With bits 5 and 7 set to 1's, the conditions of the test are that the test must terminate unconditionally (bit 5) and the ERSLT bit (bit 7) is a 1; therefore, the output of PAL, following the result-comparison, should be sensed as 0. This result means that the mask of all 1's in S should be sensed as all 0's if S and the paths through the PAL are functioning properly. This is the purpose of this first test – to determine that 1's in S can be sensed as 0's at PAL, and that an unconditional termination stops testing.

If this first test passes, the CPU stops with the 'pass' trigger set. Should the test fail, the CPU stops with the 'fail' trigger set. If the CPU does not stop, a malfunction exists in some other portion of the machine, an ATN was not performed successfully (could not sense 1's in S *or* T), or the CPU failed to act on the UT bit.

The correct indications for the successful passing of test 1 are:

1. S all 1's (roller 1, position 3).
2. T all 0's (roller 2, position 3).
3. 'Pass' trigger set (roller 5, position 2, bit 29).
4. UT bit set (roller 5, position 2, bit 5).
5. ERSLT bit set (roller 5, position 2, bit 7).
6. 'Buffer 1' trigger set (roller 5, position 2, bit 35).

When RESTART FLT I/O is depressed after a passing test, the test is repeated and again stops because the TN is all 1's. If a restart is initiated after a failing test, a "runaway" tape may occur. (A runaway tape condition occurs because the TN search routine keeps searching the tape for a nonexistent TN.) The indication of this condition is that S continues incrementing as test numbers are brought in for comparison. The failure is in S, in that one bit is being sensed as a 1 (0 in S). This runaway condition should result anytime the restart sequence is initiated following a failure indication on test 1.

On a failure indication or tape runaway on restart, T can be used to compensate for the failing bit in S by performing the following procedure:

Note: This procedure applies to the FLT tape only.

1. Reset (place in up position) DATA switches 32–63 (T), one at a time, keeping all other switches set (down). Depress STORE and RESTART FLT I/O in that order.
2. Repeat step 1 until resetting one of the DATA switches causes the tape to run away. The bit in S corresponding to the up switch in T is the failing bit, and is not being recognized as a 0 in PAL.

Analysis of the above procedure reveals that the DATA switches, by setting T to 1's, is compensating for the bit in S that is not being sensed as a 0 at PAL. The test fails, but the CPU stops until the bit in T, corresponding to the failing bit in S, is 0, at which time both inputs to the OR are 0 and the tape runs away. Assume the test fails when DATA switch 37 is up (reset). This condition means that bit 5 of S is failing.

Because the TN/ATN comparison and the result-comparison tests in this hardcore test are identical, a failing test may be repeated for scoping during the result-comparison portion of the test. The recommended procedure after the failing S-bit has been determined follows:

1. Restart from the beginning with all DATA switches set. At the first stop, set the TEST MODE, REPEAT switch and depress START. Scope the suspected S-bit at the time it is used for the result-comparison test (not TN/ATN compare). S should contain an all 1's mask for the result-comparison.
2. While synchronizing the scope on the 'scan out S and T' signal, scope the OR's shown on the left side of ALD AP691 for the suspected S-bit.

A tape runaway can continue because of failure of the UT bit. If the tape runaway continues when the above sequence is performed, the single-cycle routine detailed in *2065 Processing Unit FEMM*, form Y27-2039-2, should be followed.

If test 1 passes, test 2 must be taken on the second IPL.

IPL 2. Depressing LOAD after the stop at the end of the hardcore tests in record 3 reads in record 4, which contains the IPL 2 channel program. (See "ROS Test Tape".) The IPL 2 channel program replaces IPL 1 and contains a TIC CCW at location 8. Executing this TIC causes a branch to the read CCW in location 88 (loader), which, in turn, refills the buffer 1 area of storage from the next tape record.

The next CCW is a TIC to location 98, which contains a read CCW that fills buffer 2. At the same time that buffer 2 is being filled, the hardcore test (test 2, record 5) in buffer 1 is executed.

Test 2, Record 5. Test number 2 in the hardcore sequence determines whether 1's in T can be sensed as 0's at PAL. This test also leads into record 6. The format of test 2 is:

Mask (S)	MCW (T)	TN (S)	ATN (T)
FF FF FF FF	01 00 00 00	FF FF FF FF	FF FF FF FF

The test is taken because the TN is all 1's (S equals 0 at PAL). The MCW sets up the condition that the ERSLT bit is a 1, and, because there is no termination bit, an ATN search is performed. The mask of all 1's causes the test to pass unconditionally. (Test 1 determined that S is functioning properly.)

Following the test, the CPU immediately begins execution of the first test of record 6, as determined by the TN/ATN search (all 1's in T and all 0's in S). Recall that on a TN search, the 32-bit TN of each test in sequence is put into S to be compared with the 32 bits left in T by the previously executed test.

If the next test to be executed is not test 1 in record 6, a 1-bit in T is not being sensed at PAL. Should this occur, the failing bit results in a configuration in S that causes the CPU to do a successful TN search. The CPU stops at the test in record 6 bearing the bit configuration that agrees with the failing bit in T. The possible configurations are listed in ALD M8006-1.

To summarize, test 2 should cause the CPU to execute the first test in record 6. A failing bit in T, however, causes the CPU to execute an alternate test in record 6 and then stop, displaying a bit in S. The corresponding bit in T is the bit that is failing. The recommended procedure following a hardcore stop at this point is discussed next.

Record 6. The CPU stops in record 6 only if the ATN from test 2 cannot be sensed as all 0's. If no failures occurred in T and in the scan-out path from T to PAL, the first test in record 6 is executed and has the following format:

Mask (S)	MCW (T)	TN (S)	ATN (T)
FF FF FF FF	01 00 00 00	00 00 00 00	80 00 00 01

This test passes unconditionally because of 1's in the mask. The CPU does a TN search because the 'pass' trigger is set and no termination bit exists. The search leads the CPU to the first test in record 7 because the TN of test 1 in record 7 has the configuration 7F FF FF FE, which is the complement of the ATN of the first test in record 6.

If the ATN from test 2 cannot be sensed as all 0's, a successful test number comparison is *not* performed for test 1 of record 6. The test is *not* executed, and record 6 is searched until a test is found whose test number contains a bit corresponding to the failing bit in T. This test is executed and the CPU stops because the UT bit is set. For example, if the CPU executes the test with the TN configuration of 00 00 80 00, bit 48 in T is being sensed as a 1 at PAL. Scoping this error is difficult because the CPU does not refetch the TN during repeat. To pin-point the malfunction to a test that can be repeated and scoped, use the following procedure (FLT's only):

1. Put all 0's into DATA switches 0–63 on the system control panel. Depress STORE and RESTART FLT I/O. This procedure should bring the CPU to the second correct hardcore stop in record 8 (test 3).
2. Depress RESTART FLT I/O again. This action should cause the CPU to begin the zero-cycle tests. A failing test should be encountered in the zero-cycle test sequence that tests the T-bit that failed when running record 6. This zero-cycle test may be repeated and the failing points scoped to isolate the malfunction.

Note: If tape runaway is encountered during this routine, bit 63 or 55 in T is in error, and the TN search (which involves these bit positions) to start the zero-cycle tests is unsuccessful. Should this event happen, return the CPU to the second hardcore stop and repeat the procedure, changing the ATN of the second hardcore stop to 102 or 201.

If no error is encountered, the first test of record 6 will lead into the first test of record 7.

Test 1, Record 7. The first test in record 7 normally follows the successful testing of test 1 in record 6. The record 7 test places a 0 into each bit position in T and verifies that the 0 can be sensed as a 1 at PAL; the procedure is then repeated for S. Each test checks one bit position only. Error indications are the same for either register. Error stops are listed in ALD M8006-2. If all tests in record 7 are run successfully, the CPU begins the tests on record 8.

Test 1, Record 8. Test 1 in record 8 determines the ability of the CPU to take the next test rather than do a TN search, as was done in the previous records. The format of test 1 is:

Mask (S)	MCW (T)	TN (S)	ATN (T)
FF FF FF FF	03 00 00 00	22 22 22 22	44 44 44 44

The MCW, with a configuration of 0000 0011 (bits 6 and 7 set), determines the ability of the CPU to take the next test in sequence rather than search for the alternate test. The 'pass' trigger is set by the mask being all 1's and the ERSLT bit equal to 1. With the 'pass' trigger set and the CT bit equal to 1, the next test is executed without a TN search. If the CPU cannot force the next test under these conditions, it stops when it reaches test 5 of record 8. Assuming that test 1 passes, the CPU executes the next test, which is test 2 in record 8.

Test 2, Record 8. Test 2 in record 8 tests the ability of the CPU to take the next test on a failing condition. The format of test 2 is:

Mask (S)	MCW (T)	TN (S)	ATN (T)
00 00 00 00	01 00 00 00	55 55 55 55	33 33 33 33

This test must fail because the mask is all 0's and the ERSLT bit equals 1. With no termination bit set, the conditions dictate that the CPU is to execute the next test. If an error is encountered in this test, the CPU executes test 6, and stops. If test 2 is translated correctly, the CPU executes test 3, the second correct hardcore stop.

Test 3, Record 8. Test 3 in record 8 tests the ability of the CPU to stop on a failure. The format of test 3 is:

Mask (S)	MCW (T)	TN (S)	ATN (T)
FF FF FF FF	02 00 00 00	DD DD DD DD	FE FE 01 01

The MCW in this test has a format of 0000 0010, with the CT bit set to a 1. Because the ERSLT bit equals 0 and all 1's are in the mask, this test must fail. With the CT bit set and a failing test, the CPU should stop with the ATN in T. The complement of this ATN is the first true ROS test or, if the FLT tape is being run, the first test in the zero-cycle series. Therefore, when RESTART FLT I/O is depressed, testing should start with the first ROS test or the first zero-cycle test.

The correct indications for the successful passing of test 3 of record 8 are:

1. S: all D's (roller 1, position 3).
2. T: FE FE 01 01 (FFFF 0000 for ROS tests) (roller 2, position 3).
3. 'Fail' trigger set (roller 5, position 2, bit 30).
4. CT bit set (roller 5, position 2, bit 6).
5. UT bit set (set by hardware) (roller 5, position 2, bit 5).
6. ERSLT bit reset (roller 5, position 2, bit 7).
7. 'Buffer 1' trigger set (roller 5, position 2, bit 35).

If the CPU does not stop on this test, it will take the next test, 4, which has the UT bit set. The CPU must then stop because this bit has already been successfully tested.

Summary of Hardcore Tests. The hardcore tests have checked the CPU for the following:

1. Ability to sense 1's in S and T as 0's at PAL.
2. Ability to sense 0's in S and T as 1's at PAL.

3. Ability to take next test.
4. Ability to perform a TN search.
5. Ability to stop on a failing test.
6. Ability to stop on an UT signal.
7. Ability to make a result-comparison and decide on next step.
8. That all data paths connected with the above functions are operating properly.

ROS Bit Tests. Highlights:

- ROS tests, two doublewords, are read from tape one at a time, alternately into buffers 1 and 2.
- Test sequencing is controlled by ROS test sequencer.
- A test consists of: (1) fetching MCW from storage, (2) advancing CPU clock, (3) comparing state of one ROSDR trigger with predetermined result.
- If 'pass' trigger is set, operation proceeds to next test; if 'fail' trigger is set, testing is terminated.

The remaining records on the ROS test tape contain the test patterns used to check the ROS bit planes. Each ROS test pattern consists of two doublewords in the same configuration as the hardcore tests. (See "Hardcore Tests and IPL 2".) These tests are read from the tape one at a time, alternately into buffer 1 and buffer 2. At the completion of each read-in cycle, a 'TIC' pulse is generated (by the channel) which initiates the ROS test sequence.

Test sequencing is controlled by the ROS test sequencer. When the 'TIC' pulse is received, the ROS test sequencer is set to maximum (7). Then, as each portion of the test is executed, it is decremented by 1. For each count of the sequencer, the CPU is said to be in a certain "ROS state"; for example, if the ROS test sequencer equals 5, the CPU is in ROS test state 5. When the count reaches 0, the test is complete and the CPU enters the Wait state until another 'TIC' pulse is received.

A test consists of fetching the MCW from storage, permitting the CPU clock to advance a given number of cycles, and comparing the state of one of the ROSDR triggers with a predetermined result. If the state of the trigger matches its predicted value, the 'pass' trigger is set and the operation proceeds to the next test. If the actual and predicted values disagree, the 'fail' trigger is set and testing is terminated.

When the 'TIC' pulse is received, a TN comparison takes place. If the TN comparison is successful, the operation proceeds to scan in the MCW; otherwise, the operation reverts to the Wait state. CPU clock signals are distributed as long as the FLT counter does not equal 0. When the FLT counter equals 0, the CPU clock is stopped and the expected result comparison is started. The address sequencer governs the loading of a portion of the ROSDR word (32 bits) into T. The mask is in S (the other half of the word that was loaded into the MCW), and the result comparison takes place. If a 'TIC' or 'gap' pulse is not received, the operation returns to scan in and repeats the test.

For the following discussion of the ROS test, refer to Diagram 8-116, FEMDM.

ROS Test State 7. A ROS test is started when the ROS test sequencer and the check counter are set to maximum. This action occurs because one of the following conditions is present:

1. A 'gap' pulse is received from the channel, indicating an end of record. This condition is tested for during ROS test state 6.
2. The ROS test sequencer stepped to 0. This condition is the successful end of one ROS test; therefore the next test is initiated automatically.
3. The TN comparison failed during ROS test state 4; therefore the next test is brought in.
4. LOAD or RESTART FLT I/O was depressed.

During ROS test state 7, the only operation is the decrementation of the ROS test sequencer.

ROS Test State 6. During ROS test state 6, the 'sync' latch is reset, the TN/ATN address is forced to storage controls, and a scan storage request for the TN/ATN word is initiated. The operation then waits for a 'TIC' pulse before progressing. When the 'TIC' pulse is received, ensuring that a test is in storage, the ROS test sequencer is decremented by 1.

ROS Test State 5. During ROS test state 5, the state of the 'buffer 1' trigger is inverted so that the next ROS test is read into the other buffer. The TN of the incoming word is gated to S, and the 'TIC' latch is reset. The ROS test sequencer is then decremented by 1.

ROS Test State 4. During ROS test state 4, a 'scan out S and T' signal determines whether this is the test to be executed. This determination is accomplished by comparing the contents of S and T, via the negative-OR inputs to PAL (scan-out bus). S contains the TN of the current test obtained from the ROS test just read into storage during ROS test state 5. For the first test (hardcore test in record 3), T contains all 1's loaded from the DATA switches. For all subsequent tests, T contains the ATN of the previous test. This number should be the complement of the current TN. Thus, a successful comparison results in all 0's being sent to PAL.

PAL is then checked for an all 0 result. If PAL does *not* contain all 0's, indicating that the test currently in storage is not the one searched for, the ROS test sequencer is again set to maximum and decremented by 1. This action causes the CPU to be again in ROS test state 6, and the next sequential test on the tape is initiated. This operation continues until either the entire tape has been searched or the correct TN has been found. If a valid TN cannot be found, this condition is known as a tape runaway.

If PAL does contain all 0's, indicating a successful comparison, the 'pass' and 'fail' triggers are reset and the ROS test sequencer is decremented by 1.

ROS Test State 3. During ROS test state 3, a storage request is made for the mask/MCW word, and the ROS test sequencer is decremented by 1.

ROS Test State 2. Highlights:

- MCW fetched from storage is transferred to ST and subsequently distributed to address sequencer (bits 21–25), MCW-register (bits 0–7,20), ROSAR (bits 8–19), FLT counter (bits 26–31).
- CPU clock signals cycle ROS until FLT counter is reduced to 0.
- When FLT counter equals 0, CPU clock is stopped and result in ROSDR is scanned out to T.
- ROS test sequencer is decremented by 1.

During ROS test state 2, the doubleword fetched from storage is transferred to ST, and the address sequencer, the FLT counter, and the FLT clock are reset. T now contains the MCW which is subsequently distributed as follows:

1. T(32–39,52), which contains the ROS plane number [MCW(0–3)], the UT bit [MCW(5)], the CT bit [MCW(6)], and the ERSLT bit [MCW(7)], is transferred to the MCW register. Note that MCW(4) and MCW(20) are not used and therefore contain 0's. The ROS plane number is not used for the test but is displayed in case of a failure as a guide to the CE. For most ROS tests, except the hardcore tests, the UT bit is a 0 and the CT bit is a 1. The ERSLT bit is a 1 or a 0, depending upon the design of the ROS plane being tested.
2. T(40–51), which contains the ROS address of the plane to be tested, is set into ROSAR.
3. T(53–57), which contains the scan word address of the ROSDR bit to be tested, is transferred to the address sequencer.
4. T(58–63), which contains a count of the number of clock cycles needed to read out one ROS word, is transferred to the FLT counter and FLT clock.

At the same time, the 'scan counter control' trigger is set. The trigger output deactivates the ROS sense latch reset (up until this time the ROS sense latches have been held reset; therefore no microprogram operations have been taking place) and sets the 'scan counter control' latch. This latch causes the ROS test sequencer output to be blocked, thus taking the operation out of ROS test sequencer control, and inhibits stepping the ROS test sequencer.

At this time, the bit test begins ROS functions using the address in ROSAR. The CPU clock cycles ROS until the FLT counter (loaded from the MCW word and decremented by 1 in synchronism with the CPU clock cycles) is reduced to 0. When the FLT counter equals 0, the 'cycle counter equals zero' latch is set and the 'MMSC' trigger is set to stop the CPU clock and the test. The scan-out bus is now used to scan out the results. The 'SOROS' and 'sync' triggers are set, and the 'sync' latch is set to complete synchronization of the controls with the FLT clock. The ROS sense latches are reset, and register ingating is inhibited. The 'MMSC' trigger is then reset. The output of the scan-out bus is transferred to PAL and subsequently to T. The 'scan counter control' latch is reset, thus activating the 'reset ROS sense latch' signal and restoring control to ROS test state 2. A scan system reset resets the latches and triggers used to control scan out. The ROS test sequencer is decremented by 1.

ROS Test State 1. During ROS test state 1, the results in T and the mask in S are negative-OR'ed into PAL by the 'scan out S and T' signal. A scan storage request for the TN/ATN word is initiated and, depending on the ERSLT bit value and the PAL zero-result check, the 'pass' or 'fail' trigger is set as follows:

PAL	ERSLT Bit	'Pass' Tgr	'Fail' Tgr
= 0	1	set	–
≠0	0	set	–
= 0	0	–	set
≠0	1	–	set

If a 'TIC' signal has not arrived to indicate that the alternate buffer is full, and a scan input error did not occur, the operation returns to ROS test state 3 to repeat the test. If an error or a 'TIC' or 'gap' signal occurred, the ROS test sequencer is decremented by 1, thereby going to ROS test state 0.

ROS Test State 0. During ROS test state 0, the word fetched in ROS test state 1 is loaded into ST, and a stop or continue decision is made. If the test is to be continued, a TN or an ATN is specified, and the ROS test sequencer is set to 7 and decremented by 1. If the test is to be stopped, the 'pass pulse' trigger is reset to stop the FLT and scan clocks, the UT bit, MCW(5), is set, and a 'stop scan' signal stops the tape at the end of the current record. The ROS test sequencer is set to 7, and the scan controls stop. To continue testing ROS, either RESTART FLT I/O or LOAD must be depressed.

If the ROS tests are restarted by depressing RESTART FLT I/O, the same conditions are set up as for an IPL, except that instead of an 'IPL' signal, a 'start scan' signal is raised to the channel. This signal causes the backspace CCW in location 80 to be performed.

When the ROS test stops because of a failure, the following information is displayed in the system control panel indicators:

1. Failing plane (first digit of ROS address), in hex [roller 5, position 2, bits 0–3 (CPU TEST ADDRESS)].
2. Failing ROS address (last two digits of ROS address), in hex [roller 1, position 3, S(0–7)].
3. Failing bit in decimal [roller 1, position 3, S(8–15)].

Fault Locating Tests

- Each FLT checks small portion of logic by setting up conditions that affect exit trigger which can then be sensed for proper output.
- If error occurs, test is terminated and CE uses SCOPEX card with failing TN identification to determine test points to scope.
- FLT tapes consist of hardcore, zero-cycle, and one-cycle tests.
- FLT test consists of 9 or 10 doublewords.

As discussed earlier, each FLT checks a small portion of the logic by setting up certain conditions that affect a specific trigger (known as an exit trigger) which can then be sensed for the proper output. Each test on the tape, therefore, sets up the CPU to the proper status (certain triggers set) that results in a particular output of the logic under test. After the CPU has been set up for a test (scan in), the CPU clock is advanced a sufficient number of cycles to change the status of the exit trigger. The indicator associated with the exit trigger is then selected (scan out) and compared with an ERSLT bit in the FLT MCW, set as part of the scan-in routine. If the values are equal, the test passes; if unequal, the test fails. Because each test is repeated a number of times, both the pass and fail indicators may be on, which is interpreted as an intermittent failure.

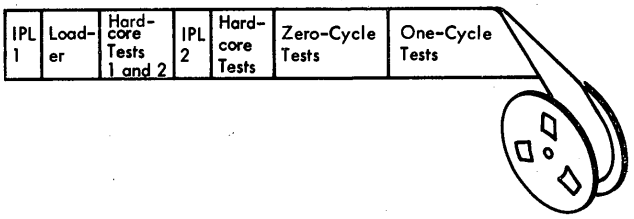
A summary of an FLT includes:

1. Load test into storage.
2. Scan into CPU triggers.
3. Advance CPU clock.
4. Stop CPU clock.
5. Scan out to T (scans desired indicator to T).
6. Load mask into S.
7. Scan out S and T (compares exit trigger with expected result).
8. Go to next test, or terminate and display failing TN.

CE action following this sequence (assuming an error or fault) requires locating the SCOPEX card with the failing TN identification, scoping the test points listed, and replacing the failing card(s).

FLT's are rarely run singly. So little time is involved that it is easier to run the entire series on a tape or until a failing test is encountered. The failing test may then be repeated until the fault is isolated.

FLT Tape. The make-up of the FLT tape is similar to that of the ROS test tape and contains the following records:



1. Record 1, IPL 1. This record contains the 24-byte "bootstrap" program necessary for any IPL operation. When LOAD is depressed, the three doublewords of this record are read into storage locations, as follows:

Storage Location (Hex)	Word	Contents
0	PSW	Backspace (not used).
8	CCW 1	Read command to read 40 bytes to location 80, and chain command to location 10.
10	CCW 2	TIC command to location 88.

2. Record 2, Loader. This record contains the loader program that reads in the FLT's. The IPL program in record 1 reads this record into storage locations, as follows:

Storage Location (Hex)	Word	Contents
80	CCW	Backspace command.
88	CCW	Read command to read 72 bytes to buffer 1 (location 8000), and a chain data tag to location 90.
90	CCW	TIC command to location 98.
98	CCW	Read command to read 72 bytes to buffer 2 (location 8080), and a chain data tag to location A0.
A0	CCW	TIC command to location 88.

3. Record 3, Hardcore Tests 1 and 2. This record contains the first FLT hardcore test. The hardcore tests for FLT's are identical to the ROS test hardcore tests except for their format. The FLT hardcore tests have the following format:

Storage Location (Hex)	FLT Word	Left Word	Right Word
8000	0	0	0
8008	1	Mask	MCW
8010	2	0	0
8018	3	0	0
8020	4	0	0
8028	5	0	0
8030	6	0	0
8038	7	0	0
8040	8	TN	ATN

Note that all locations except those containing FLT words 1 and 8 are blank. These blank locations are not used in hardcore tests.

4. Record 4, IPL 2. The first hardcore test causes a stop. When LOAD is depressed, the following IPL program replaces the IPL 1 program:

Storage Location (Hex)	Word	Contents
0	PSW	Backspace record.
8	CCW	TIC command to location 88 (loader program).

5. Records 5–8, Hardcore Tests. These records contain the remaining hardcore tests.
6. Zero-Cycle Tests. The next series of records on the FLT tape are the zero-cycle tests. These tests further check the scan logic necessary to run the main FLT's (one-cycle tests). They determine whether the scan-in and scan-out paths are operative. Scan-in and scan-out tests can exist only for triggers which have a scan-out path. In these tests, a pattern is scanned into the machine, the CPU clock is allowed to advance, and the output trigger is observed. If the trigger has a scan-in path, three tests are performed: for a reset state, for a set state, and again for a reset state. If the trigger has no scan-in path, only one test is performed, and this test ascertains that the trigger is at its reset state. While one trigger is being tested, other triggers can take on various states. Whenever possible, random states simulate the combinations which may be used in a one-cycle test and reveal interaction between triggers.
7. One-Cycle Tests. The remainder of the tape contains the one-cycle tests which are the true FLT's. The input has been generated by the FLT generating system and found to be a test that will detect and locate at least a single fault. An input pattern is scanned into the predetermined triggers, and the CPU clock is allowed to advance; the result, which is in the exit trigger, is compared with the value expected for a machine operating correctly.

The last test in each record is a dummy test, which keeps the channel from reading across the interrecord gap before the result of the last one-cycle test is known. Thus, if the last one-cycle test fails, a stop-scan sequence can be executed before the interrecord gap is reached. The dummy test is also used by the FLUT program. When used with FLUT, the dummy test is identified by a TN of all 1's and is forced to pass by having a mask of all 1's and an expected result bit of 1. The dummy test also provides a successful restart if the last test in a segment should fail.

FLT Format. Zero-cycle and one-cycle FLT's have the same format (Table 6-2), consisting of 9 doublewords (10 if a 7080 Compatibility Feature or a 709/7040/7044/7090/7094/7094II Compatibility Feature is attached). The doublewords (scan-in words) are used as follows (see ALD's M3071 and M3081 for the FLT format if a Compatibility Feature is attached):

Table 6-2. FLT Format

Storage Location (Hex)	Scan-In Word*	Left Word	Right Word
8000 and 8080	0	S	T
8008 and 8088	1	Mask	MCW
8010 and 8090	2	Q	Q
8018 and 8098	3	A	Miscellaneous triggers
8020 and 80A0	4	B	Miscellaneous triggers
8028 and 80A8	5	D	Miscellaneous triggers
8030 and 80B0	6	PSW	IC
8038 and 80B8	7	E, R	LAR
8040 and 80C0	8	TN	ATN

*Refer to ALD's M3071 and M3081 for bit assignments.

1. Scan-In Words 0 and 2–7. Contain the scan-in test pattern, which is distributed to registers and triggers as shown in Table 6-2. The data establishes a CPU state before performing the FLT.
2. Scan-In Word 1, Left Half (Mask). Specifies the exit trigger to be sensed. This is accomplished in the scan-out bus by OR'ing the scan-out word containing the exit trigger status with the mask bits. The mask contains all 1's except for a 0 in the position corresponding to the trigger being sensed. The output of the OR is inverted so that if the exit trigger is set, the value sent to PAL is 0.
3. Scan-In Word 1, Right Half (MCW). Contains the control information necessary to run the test:

MCW Bits	Scan-In Word 1 Position	Contents
0–3	32–35	0's
4	36	LW bit
5	37	UT bit
6	38	CT bit
7	39	ERSLT bit
8–19	40–51	ROS address (bits 73–84 of input pattern)
20	52	0
21–25	53–57	Scan out address
26–31	58–63	Cycle count

4. Scan-In Word 8. Contains the TN and ATN as follows: bits 0–15, TN; bits 16–31, complement of TN; bits 32–47, complement of ATN; bits 48–63, ATN.

FLT Test Setup. Several controls on the system control panel must be operated to initiate FLT's. The procedure for running FLT's appears in *2065 Processing Unit FEMM*, form Y27-2039-2 and ALD M8005. However, a short description of the setup is included here because it affects the operation.

Diagram 8-117, Sheet 2, FEMDM, shows the start of an FLT. The FLT tape is mounted first. The LOAD UNIT switches are set to the address of the tape unit holding the test tape, the TEST MODE, ROS/PROC/FLT switch is set to FLT, and the CPU CHECK switch is set to DSBL. Going into FLT test mode causes the 'FLT test' trigger to be set and a 'scan mode' signal to be sent to the channel. SYSTEM RESET is now depressed to place the CPU in a stop loop with the 'manual' trigger set. The DATA switches are set to all 1's, and STORE is depressed. This action places all 1's into ST so that T now contains an ATN for the first hardcore test. Lastly, LOAD is depressed, initiating a normal IPL operation which reads in IPL 1.

IPL 1. Depressing LOAD with the CPU in FLT test mode initiates operations that are identical to a normal IPL until a 'release' signal is received from the channel; i.e., 24 bytes are read from the selected device into the first three doubleword locations of main storage under ROS microprogram and channel control. At this point the CPU is idle, waiting for a 'release' signal from the channel. Meanwhile, the channel IPL operation reads in record 1 of the FLT tape and executes the channel program specified by record 1. (See "FLT Tape".) As a result, 40 bytes (record 2) are read into storage, starting at location 80 (hex). Record 2 contains the loader program that will read each FLT into the proper buffer area in storage.

After record 2 has been read in, command chaining causes CCW 2 in record 1 to be executed. This CCW is a TIC command to location 88, which now contains a read CCW. When the channel has finished executing the TIC command, it sends a 'release CPU' signal to the CPU. As soon as the 'release CPU' signal is received, the IPL microprogram is continued and the CPU proceeds as though performing a normal IPL. However, when the 'timing gate' trigger is reset (Diagram 8-117, Sheet 3), the 'IPL status' latch is set, which in turn sets the 'MMSC' trigger. With the 'MMSC' trigger set, the CPU clock is stopped, thus halting all ROS operations.

The 'sync' trigger and 'sync' latch are set to synchronize the operation to the FLT clock. In addition, the 'sync' trigger output resets the ROS sense latches and inhibits register input. Address 001 is forced into ROSAR, and the 'MMSC' trigger is reset to release the CPU clock. Thus, the FLT microprogram is initiated. ROS control now places the CPU in scan mode and sets the address sequencer to 7. The CPU is now ready to run the first hardcore tests.

Loader. Highlights:

- FLT's are read into two buffer areas (8000 and 8080, hex) in main storage.
- Each time a buffer is filled, 'TIC' pulse informs CPU that test can be performed.
- Read-in operation continuously checks for errors.
- After 32 errors, test is halted.
- Test is repeated until 'TIC' pulse is received.

The loader channel program read in by IPL 1 is used to transfer all FLT's, whether hardcore, zero-cycle, or one-cycle tests, to main storage. The test data is read into two buffer areas in main storage [starting at locations 8000 (hex), designated buffer 1, and 8080 (hex), designated buffer 2]. Each time a buffer is filled with a single test, a 'TIC' pulse informs the CPU that the buffer is filled and the test can be used by the CPU.

The tests are read into the buffer areas on a sequential basis. For example, test 1 is read into buffer 1. As test 1 is being processed, buffer 2 is being filled with test 2. As test 2 is being performed, buffer 1 is being refilled with test 3, and so on.

A continuing check for errors is made during read in and during transfer of data to the CPU. If an error is encountered, the tape is backspaced and another attempt is made to read the data into storage. The check counter, set to 31 by a micro-order, keeps track of the number of errors encountered and allows 32 attempts to read in before the test is halted. Detection of an error decrements the check counter by 1 and causes a backspace of the tape so that the read in is retried. Manual intervention is required following a stop under these conditions. Channel-control checks may also occur which halt the test procedure immediately.

Once a test is begun, it is repeated until a 'TIC' pulse is received, indicating the next test is ready for processing. No count is made of the number of times a test is run as this is a function of the data rate of the I/O device involved.

During normal sequential processing of FLT's (no faults encountered), the address sequencer is set to 7 and the eight doublewords composing a single FLT are scanned into the CPU. The following is a summary of the read-in and test sequence:

1. While the FLT is being scanned into the CPU, a continuing check for storage and/or input errors is made. Storage errors might occur between storage and the CPU. Input errors might occur between channel and storage while the next FLT is being read into the alternate buffer. For each error sensed, the check counter is decremented by 1. Detection of an error results in a backspace of the tape and a retry.
2. When scan in is complete, the 'scan mode' trigger is reset, and an unconditional branch to the ROS word addressed by MCW(8–19) is performed.

3. The 'scan counter control' trigger is set, and the FLT counter keeps track of the number of CPU cycles specified by MCW(26–31). The CPU clock is stopped when the FLT counter equals 0.
4. Following a successful scan in and clock advance, a scan-out operation places the test result into T. The combination of MCW(4) and MCW(21–25) determines the scan-out word that is transferred to T.
5. At this point, the mask is fetched from storage and transferred to S. A result comparison between the mask and the scanned-out word is made, and the 'pass' or 'fail' trigger is set. The next operation to be performed is determined by pass, fail, unconditional-, or conditional-terminate conditions.
6. Execution of this FLT is repeated until a 'TIC' or 'end of record' ('gap') signal is received. When either signal is received, the decision to stop or continue is made, and appropriate controls are set.

Transmission Checks During FLT Read In. When the channel discovers a data check, it sends a 'unit data error' signal to the FLT controls. The channel then automatically backspaces one record and starts reading into buffer 1 again.

The FLT controls, upon receipt of the 'unit data error' signal, decrement the check counter, reset the CPU, and wait for a 'TIC' pulse from the channel. When the 'TIC' pulse is received, the FLT controls start the testing from storage buffer 1, repeating the tests from the record that had the data check. If the check persists, the record is retried up to 32 times. If 32 attempts to complete the record are unsuccessful, the machine stops with the UDC (Unit Data Check) indicator (roller 5, position 2, bit 32) on.

If a control check occurs in the channel, the channel stops transmission and sends the 'channel control error' signal to the FLT controls. The signal stops the testing immediately and turns on the CCC (Channel Control Check) indicator (roller 5, position 2, bit 33).

If the BCU recognizes an address check, data check, or invalid address check, this information is sent to the FLT controls. On any of these checks, a 'stop scan' signal backspaces the record and starts reading into buffer 1 again. The CPU and FLT controls are reset, and the check counter is decremented. The FLT controls wait for a 'TIC' pulse from the channel. When this pulse is received, the FLT controls start testing from buffer 1, repeating the tests from the record involved. After 32 unsuccessful attempts to complete the test record, automatic testing stops with the error indicator on.

Hardcore Tests. Functionally, the hardcore tests at the beginning of the FLT tape are identical to the ROS hardcore tests. The differences are in the format and control of the tests. FLT hardcore tests have the same format as zero-cycle and one-cycle tests except that all scan-in words except words 1 and 8 are 0's. Word 1 contains the mask and MCW, and word 8 contains the TN/ATN. These words are used in the same manner as in the ROS hardcore tests.

FLT hardcore tests are controlled by the FLT ROS microprogram instead of by hardware as in the ROS hardcore tests. Successful testing terminates on a hardcore stop (a correct stop), and, upon restart, the CPU enters zero-cycle tests. The CE may attempt isolation and repair on an error stop encountered during hardcore tests according to the procedure in ALD M8006.

Zero-Cycle and One-Cycle Tests. Highlights:

- Scan in loads CPU with test data.
- Test cycle allows CPU to act on scan-in data.
- Scan out collects data after clock advance.
- Mask is compared with scan-out data.
- Testing ends or continues, depending on result comparison.

Each FLT follows a similar 6-step routine under ROS control (with the exception of scan out, which may be either under ROS or hardware control). The FLT sequence may be summarized as follows:

1. Scan In. When a storage buffer has been filled with an FLT and the CPU receives a 'TIC' pulse, scan in begins. The address sequencer is set to 7 by a micro-order, and words 0 through 7 are read into the CPU. These words enter the CPU via Q or ST, and are scanned into various registers and triggers throughout the CPU, using normal data paths, to set up the machine environment for a particular test. During scan in, certain micro-orders are interpreted for scan control rather than for functional operations. From T, the MCW is transferred to the address sequencer, the FLT counter, and the MCW register when the address sequencer equals 1. When the address sequencer equals 0, scan in is completed except for ST, and the test cycle phase is entered. A micro-order transfers T to ROSAR, and on the next cycle ST is scanned in.
2. Test Cycles. The count in the FLT counter, set by the MCW, determines the number of cycles taken by the CPU. A micro-order, specified by MCW(8–19), controls this portion of the test. When the FLT counter equals 0, the CPU clock is stopped and scan out begins.
3. Scan Out. The status of the exit trigger is placed into T. The scan-out word containing the status of the exit trigger is specified by the count in the address sequencer. MCW(4) determines whether the right or left word contains the exit trigger status. If the count in the address sequencer is 14 or greater, scan out is controlled by hardware. If the count is 13 or less, scan out is controlled by ROS.

4. Result Comparison. The exit trigger is compared with a known value, one that is predicted on the basis of the information scanned in. This comparison is accomplished by fetching the mask from storage, placing it into S, then OR'ing the scan-out word and the mask in the scan-out bus. The mask contains all 1's except for the position corresponding to the exit trigger, which is a 0. Assuming the exit trigger is set when it is OR'ed with the 0 in the mask, the output is transferred to PAL as a 0. Next, the CPU compares the output of the OR with MCW(7), the ERSLT bit. With all 0's in PAL and MCW(7) = 1, the CPU determines that the test has passed and sets the 'pass' trigger. If the comparison is unfavorable, the 'fail' trigger is set. Because the test is continually repeated until the next 'TIC' pulse is received, both the 'pass' and the 'fail' triggers can be set, indicating an intermittent failure, in which case the 'intermittent' trigger is also set. After the result-comparison, the CPU must decide whether to terminate testing or to continue.
5. Terminate or Continue. This decision is a major point in the FLT sequence. Four triggers determine what the CPU will do next: 'pass', 'fail', 'UT' [MCW(5)], and 'CT' [MCW(6)]. The CT bit is always set in current FLT's (except for certain hardcore tests). Depending upon the setting of these triggers, the decision that will be made is as follows:

Trigger Output 'UT'	'CT'	'Pass'	'Fail'	Action
0	0	1	0	Continue – alternate test
0	1	1	0	Continue – next test
0	0	0	1	Continue – next test
0	1	0	1	Stop – gate alternate test on restart
1	0	1	0	Stop – gate alternate test on restart
1	0	0	1	Stop – gate alternate test on restart

The CPU repeats the current test until the 'TIC' pulse is received from the channel. At that time, the decision to stop or continue is made. If the CPU is to continue, the test in the opposite buffer is scanned in. No count is made of the number of times the test is repeated.
6. TN/ATN Comparison. This comparison is accomplished in hardcore and whenever RESTART FLT I/O is depressed. The TN located in word 8 (word 9 for a 7080 Compatibility Feature or a 709/7040/7044/7090/7094/7094II Compatibility Feature) is compared with the ATN left in T by the last test to be executed. Also, a specific TN may be entered via the DATA switches, and the CPU searches for this number to the exclusion of all other tests. As FLT's are now set up, the ATN in each test is the number of the next FLT on the tape (with the exception of hardcore tests).

Scan-In. Highlights:

- Test pattern establishes trigger status before test.
- Test words are read into S, T, or Q, then distributed throughout CPU under microprogram control.

The scan-in portion of an FLT consists of fetching a test pattern to establish a trigger status before a test. Scan-in test words are read into S, T, or Q from the FLT buffers. From these registers, the data is transferred throughout the CPU, under microprogram control, via special scan circuits and normal CPU paths. Transfer paths are determined by the microprogram, which is repeated without variation for each FLT. At the end of scan in, the CPU clock is allowed to cycle the number of times required to condition the exit trigger. Scan in ordinarily fetches eight words from storage into the CPU but provision has been made for a ninth word to be used with the 7080 Compatibility Feature or the 709/7040/7044/7090/7094/7094II Compatibility Feature.

A scan in is started with the address sequencer set to 7 (Diagram 8-117, Sheet 3, FEMDM). A scan storage request fetches scan-in word 7 of the record. This word is transferred to ST, the address sequencer is reduced by 1, and a check is made for a scan input error. If an error is encountered, the check counter is reduced by 1, a 'start scan' signal causes the tape to backspace, and the operation begins again where scan-mode is set into the CPU. If no error occurred, another scan storage request is made for word 6, which is held in the CPU pending a check for a 'TIC' or 'gap' pulse. A 'TIC' or 'gap' pulse indicates the alternate storage buffer is full. Because the scan in was just started and the CPU is much faster than a tape drive, it can be assumed that no 'TIC' or 'gap' pulse is received. The address sequencer is reduced by 1, and a scan input error sample is made. If no scan input error occurred, another scan storage request is made. Anytime a scan input error occurs, the check counter is reduced by 1, the 'start scan' signal causes the tape drive to backspace one record, and the operation repeats from the time scan-mode is set.

Scan in continues to operate for words 5, 4, 3, and 2 exactly as it did for word 6, except that no check for a 'TIC' or 'gap' pulse is made when the address sequencer equals 2. Word 1 (mask and MCW) is loaded into ST, and a check is made for a 'TIC' or 'gap' pulse. Before word 0 is transferred into the CPU, the mask and MCW (word 1) are transferred out of ST. Diagram 8-117, Sheet 4, shows that T(32–39) is transferred to the MCW, T(53–57) is transferred to the address sequencer, T(58–61) is transferred to the FLT counter and FLT clock, and the 'scan counter control' trigger is set. If no scan input error occurred, word 0 is loaded into ST after T(40–51) is loaded into ROSAR. The 'scan counter control' latch is then set to allow the test to be performed.

Test Cycles. The CPU clock is allowed to advance a number of cycles as specified by MCW(26–31) (Diagram 8-117, Sheet 5). A portion of the microprogram starting at the address specified by MCW(8–19) is performed. Note that at this point the CPU seems to be running a program. However, this is not the case. The micro-orders being performed merely allow the scan-in test pattern to be transferred through the logic being tested to change the state of the exit trigger. If setting these triggers at scattered points causes three legs of an AND to be tested, then 1's are directed to those triggers when testing the AND. If the three triggers happen to be those normally set during, for example, I-Fetch, this fact is incidental to FLT testing.

As soon as the CPU clock cycles begin, the FLT counter is stepped in synchronism with the clock. When the proper number of cycles has been taken, the FLT counter steps to 0, which causes the 'cycle counter equals zero' latch to be set. The 'SOROS' and 'MMSC' triggers and the 'sync' latch are then set. The 'MMSC' trigger stops the CPU clock. As soon as the 'sync' latch is set and the operation is synchronized to the scan and FLT clocks, the 'MMSC' trigger is reset. The operation proceeds under ROS or scan logic control, depending on the value in the address sequencer.

Scan Out. During scan out (Diagram 8-117, Sheet 5), the exit trigger status is transferred from the indicator driver circuits, through PAL, to T. This path is the same path used for logout operations.

The word scanned out is determined by the scan-out address in the address sequencer. The setting of the address sequencer determines which roller switch on the system control panel contains the desired information, and MCW(4) determines whether the right or left half of the word should be scanned out. Scan out is first controlled by scan hardware and then by the micro-program. The hardware-controlled portion of scan out is always performed first regardless of the value in the address sequencer, although only those fields addressed by a value of 14 or greater are scanned out. If the exit trigger is in a scan-out word whose address is 13 or less, the field is scanned out under microprogram control.

ROS control is resumed by forcing one address if the address sequencer is greater than 13, and another address if it is less than 14. The microprogram started by the former is ready to perform a result comparison because the exit trigger value is in T. The latter program first branches on MCW(4) to determine if the left or the right half of the doubleword addressed by the address sequencer should be scanned out. The LSWR is gated to T by a normal CPU gating signal; therefore a branch must be made on address sequencer equal 13. Because the LSWR is contained in the right half of word 13, a branch on address sequencer equal 13 is unnecessary if MCW(4) equals 1. By means of these branches, the microprogram transfers the field to be tested to T, and thus begins the result comparison.

Result Comparison. FLT's check logic by moving data through a group of logic to an exit trigger and causing a change in the state of the exit trigger. To determine the success or failure of a test, the change in the exit trigger must be sensed on the basis of the data entered into the logic. This change is predictable, and the data on the test tape designates whether the exit trigger is to be a 1 or a 0 at the conclusion of each test. The expected state of the trigger is contained in the MCW as the ERSLT bit.

The scan out S-and-T function (Diagram 8-117, Sheet 5) determines the setting of the exit trigger by comparing a mask with the scan-out word containing the status of the exit trigger. Scan out places the selected scan-out word into T. A mask word, fetched from storage, is put into S. The mask contains all 1's except for the bit position corresponding to the exit trigger in T. This bit position is a 0, and therefore the exit trigger determines the setting of PAL.

If the exit trigger is set, a 0 is transferred to the corresponding PAL bit; if reset, a 1 is transferred to PAL. Thus, PAL equals 0 only if the exit trigger equals 1. If the ERSLT bit [MCW(7)] also equals 1, or if PAL does not equal 0 (indicating that the exit trigger is reset) and the ERSLT bit equals 0, the 'pass' trigger is set. If MCW(7) equals 1 and PAL does not equal 0, or if MCW(7) equals 0 and PAL equals 0, the 'fail' trigger is set. In any case, if both the 'pass' and 'fail' triggers are set, the 'intermittent' trigger is also set.

Terminate or Continue. When the mask was loaded into S, the address sequencer was set to 8 and a scan storage request was initiated to fetch scan-in word 8 (TN/ATN). At the same time that the mask is compared with the scan-out word, scan-in word 8 arrives from storage and is gated to ST. The result in PAL and the ERSLT bit are used to set the 'pass' or 'fail' trigger, and thus record the result. The address sequencer is then decremented, resulting in a repeat of the same test just executed.

Assuming the test was successful and no errors were found, the operation described is repeated until the buffer is filled with the next test or until an end-of-record gap is signalled. These conditions are repeatedly sampled for during scan in. If a 'TIC' or 'gap' pulse is received, scan-in execution is abandoned. The address sequencer is set to 8 and the TN/ATN of the test in progress is fetched and loaded into ST. After a system reset which clears the CPU registers, a stop or continue check is then made (Diagram 8-117, Sheet 5). If a continue condition results, the operation proceeds to the setting of scan mode to start the next or alternate test as determined by the stop or continue controls. In case of a stop conclusion, the 'stop scan' signal goes to the channel and the UT bit [MCW(5)]

is set. The clocks are then stopped by resetting the 'pass pulse' trigger, and the CPU enters a microprogram loop.

After a stop condition, manual intervention is necessary. The operation then depends upon the pushbutton depressed, as follows:

1. START (with TEST MODE, REPEAT switch off). Runs the test that causes the stop one time and causes another stop.
2. START [with TEST MODE, REPEAT switch in the Repeat (down) position]. Continues to run the test that caused the stop until TEST MODE, REPEAT is turned off.
3. STORE. May be used to set an ATN into T. (May be used in conjunction with RESTART FLT I/O to perform an ATN search for a known test.)
4. RESTART FLT I/O. Sends a 'start FLT' signal to channel and resets the UT and CT bits and the 'fail' and 'buffer 1' triggers. The 'pass' trigger is set. (This combination represents an ATN search condition.)
5. LOAD. Initiates a system reset and begins at the start of the FLT test tape.

TN/ATN Comparison. Every FLT contains the TN and the ATN of the test which the CPU is to execute upon completion of the current test. This information is contained in word 8 (word 9 for the 7080 Compatibility Feature or the 709/7040/7044/7090/7094/7094II Compatibility Feature) and consists of the TN and ATN in both true and complement form.

The hardcore tests verify the ability of the CPU to conduct a TN search by comparing the TN (in S) with the ATN (in T) using the 'scan out S and T' signal. A 0 output indicates a favorable comparison, and the CPU then executes the test. If the inverted-OR output is not sensed as all 0's, the CPU rejects the test and continues searching. The bit configuration in hardcore tests cannot strictly be called a TN, but the functions of comparison, acceptance, or rejection of the test are valid.

During zero-cycle and one-cycle tests, the TN search is rarely used. If the CPU does not encounter a failing test once it has entered the FLT sequence, word 8 is not even used. The address sequencer is set to 7, and scan in begins immediately.

However, if a failing test is encountered, the last action the CPU takes is to put the ATN into T. The test immediately following the failing test has a TN that is the complement of the ATN. When this TN is brought into S and compared with the ATN, a 0 output results, and the CPU begins testing again with this test. This action allows the CE to get past a failing test when he desires to do so. This situation could happen, for example, when an FLT is failing because of an engineering change and the CE is aware that failure is not due to a malfunction. Each FLT has the TN of the next sequential FLT as its ATN.

For example, assume the computer has just run TN 01 09 and the test has failed. The next test is TN 01 0A. RESTART FLT I/O is depressed to get past the failing test. Upon restart the tape is backspaced to the beginning of the record containing the failing test. The TN of each test in the record is brought into S for comparison with the ATN left in T by the failing test. The comparison is favorable when the test immediately following the failing test is encountered, and the CPU resumes testing at that point. TN 01 09 leaves the following configuration in T (right half of word 8):

ATN Complement	ATN
FE F5	01 0A

TN 01 0A, the next test, has the following configuration in the left half of word 8 which was brought into S:

TN	TN Complement
01 0A	FE F5

The following binary bit configuration is scanned out:

T:	1111	1110	1111	0101	0000	0001	0000	1010
S:	0000	0001	0000	1010	1111	1110	1111	0101

The CPU resumes testing with TN 01 0A and continues until another failure is encountered or to the end of the testing sequence.

RIPPLE TESTS

The ripple tests may be used to exercise either main storage or local storage and several functional units within the CPU. The tests use only the internal hardware of the CPU, including several ROS words, and storage. The procedure stores the contents of the DATA switches into all locations of storage or displays the contents of all storage locations.

The ripple tests may be used as a quick confidence test of CPU and storage operation or as a means of identifying, with the parity check indicators, a failing major functional unit within the CPU. The overall ripple test routine is contained in *2065 Processing Unit*, FEMM, form Y27-2039-2.

DIAGNOSTIC PROGRAMS

For a list and discussion of the diagnostic programs available for the Model 65, refer to the *2065 Processing Unit*, FEMM, form Y27-2039-2.

MARGINAL CHECKING

The marginal checking facility enables several units in the system to be operated with nonstandard voltage conditions, thus providing a means of detecting critical circuits that are deteriorating to a critical voltage-sensitive operating point. In addition, the CPU may have its clock period decreased from 200 ns to 195 ns, thus providing a means of detecting circuits that have developed a slower switching speed.

Several of the power supplies in the CPU, and several power supplies in the storage units and channels whose power control is interconnected with the CPU, may be varied from the nominal output voltage by controls at the CPU. The CPU power supplies are varied directly from the system control panel. (The adjustment procedure for these controls is contained in *2065 Processing Unit*, FEMM, form Y27-2039-2.) The storage units and channels are varied by motor drives controlled from the system control panel. (The adjustments of the remote control assemblies are contained in their respective FEMM's.)

The marginal checks are performed by running the ROS tests and FLT's with all 6V marginable power supplies in the CPU reduced to 5.5V dc and the ROS power supply reduced to 80 percent of nominal. If the tests passed, rerun them with normal voltages, but with the FREQUENCY ALTERATION switch set. After a successful run of the ROS tests and FLT's, run selected diagnostic programs with the marginable power supplies in the CPU, storage units, and channels set from nominal to higher or lower output.

www.IBM360.info

Appendix A. Special Circuits

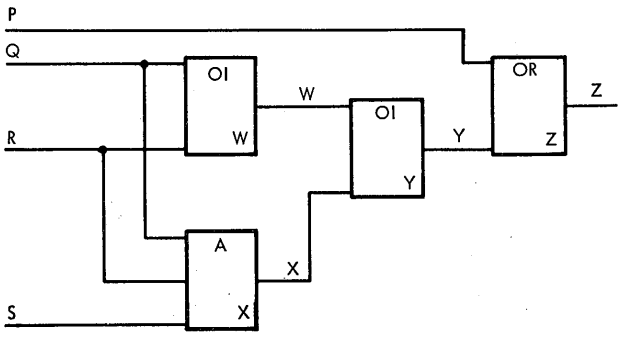
The only special circuits in the 2065 are in ROS for the differential amplifiers, latches, and driver circuits; they are not, however, field repairable. For a discussion of the standard SLT circuits, refer to the *SLT Component Circuits,* FEMI, Form Z22-2798 *(IBM Confidential, for release only to authorized persons).*

Appendix B. World Trade Differences

The major difference between World Trade versions of the Model 65 and the domestic version is that 50-Hz power is used in World Trade machines. This change primarily affects: (1) power distribution and control (Chapter 5), (2) decrementing of the interval timer (Chapter 3, Section 1: "Timer Exceptional Condition"), and (3) decrementing of the Multisystem timer, if the Multisystem feature is installed (Chapter 4, "Multisystem Timer").

Consider the simple four-block tree shown in Figure C-1. Assume that output Z is observable and that inputs P, Q, R, and S can be set directly. The FLT generating programs try to develop tests for each block to show that each input can, by itself, control the output and that the output can take both values. For example, the tests for a three-input OR would be 100, 101, 001, which check the inputs, and 000, which checks for a 0 output.

If tests are to be generated for the logic shown in Figure C-1, the first test pattern generated would put 10 as the inputs to block Z, which then yields a 1 output and is sensitive. The 1 input (P) is also sensitive, but the 0 input is not. To make the output of Y 0, a 1 is required on at least one input. Assume the program chooses W, which requires Q and R to be 0. This action forces X to 0; S can be any value, and is arbitrarily set to 0. Thus, the first test pattern is 1000 and tests only two nets. (See line 1 of the table in Figure C-1.)



Test #	P	Q	R	S	W	X	Y	Z
1	1$	0	0	0	1	0	0	1$
2	0	1$	0	0	0$	0$	1$	1$
3	0$	0$	0$	0	1$	0	0$	0$
4	0$	1$	1$	1$	0	1$	0$	0$
5	0	0	1$	0	0$	0$	1$	1$
6	0	0$	1	1	0$	0$	1$	1$
7	0	1	0$	1	0$	0$	1$	1$
8	0	1	1	0$	0$	0$	1$	1$

Legend: $ means that this net is sensitive.

Figure C-1. Four-Block Tree and FLT Pattern Generated

To test the other input of Z, a pattern 01 is needed. For the Y output to be 1, W and X must both be 0 and, because Y is now sensitive, so are W and X. To make the output of X 0, at least one input must be 0, and R is arbitrarily chosen. Now to make the output of W 0, Q must be 1 and is sensitive. Again, S is set to 0.

The next test pattern generated puts 00 as inputs to Z and checks the output for 0. The same procedure that set Y to 0 in test 1 is followed. However, Y is now sensitive and, as shown in line 3 of the table, so are Q, R, and W.

Block Z has now been fully tested and so the program steps back a level to check block Y. The first test for this sequence already exists in test 3 and so is not redeveloped. The second test requires 01 as the Y inputs. For the X output to be a 1 means that Q, R, and S must all be 1's. This condition automatically causes the output of W to be 0, which is correct. Because Y cannot be observed directly, its output must be propagated to Z as a sensitive path. This propagation is done by making P 0. Now Z, Y, X, and all of the inputs to X are sensitive. Because the output test for block Y is performed by the second test, testing of block Y is now complete.

The first input test for block W is already included in test 2. The second test makes Q 0 and R 1, which causes the output W to be 0. To propagate this condition sensitively through Y requires that X be 0, which it is. Again, to propagate through Z, P must be 0. S is arbitrarily set to 0. This condition is line 5 of the table in Figure C-1. The output test for block W is included in test 3.

The input test patterns for AND X should be 011, 101, and 110. None of these has been set up so far. Because these are all primary inputs, no real test development is needed. The only other important input is P, which must be 0 for Y to be sensitive. These three patterns are shown in lines 6, 7, and 8 of the table in Figure C-1. The output test for block X is already in pattern 4.

A test now exists for every possible logic failure in the network, and the process ends. The test patterns are formatted for the machine, termination bits are added, and index numbers are assigned. These numbers are those in the table in Figure C-1, with a constant added to define the segment.

The failure data from the table is now used to produce a SCOPEX. In a SCOPEX, all sensitive points are listed for each test, together with indications showing which are

newly tested. Table C-1 is an abbreviated version of SCOPEX for the first five tests. Note that the "Fed By" column refers only to those sensitive items feeding the given net. Thus, in test 2, although the output of block X is sensitive, no inputs are sensitive and so no input is listed. The list for test 4 shows an example of the G/FI notation. In this test, all three inputs to X are 1 and are sensitive. However, entry Q has been tested at 1 through block W before this. Therefore, the entry is now newly tested, but is newly tested as an input to X. For this reason, the G/FI is put on the line of X.

Table C-1. SCOPEX Example

	V	Net	Ref	Fed By
	***1			
G/F	1	Z	AA	A1
G/F	1	P	A1	Entry
	***2			
	1	Z	AA	AB
G/F	1	Y	AB	AC, AD
G/F	0	X	AC	–
G/F	0	W	AD	A1
G/F	1	Q	A1	Entry
	***3			
G/F	0	A	AA	AB, A1
G/F	0	Y	AB	AC
G/F	1	W	AC	A2, A3
G/F	0	P	A1	Entry
G/F	0	Q	A2	Entry
G/F	0	R	A3	Entry
	***4			
	0	Z	AA	AB, A1
	0	Y	AB	AC
G/FI	1	X	AC	A2, A3, A4
	0	P	A1	Entry
	1	Q	A2	Entry
G/F	1	R	A3	Entry
G/F	1	S	A4	Entry
	***5			
	1	Z	AA	AB
	1	Y	AB	AC, AD
	0	W	AC	–
	0	X	AD	A1
G/F	1	R	A1	Entry

Note: All index entries for Chapters 1 and 2 are in Volume 1; all other entries are in Volume 2.

A-Register (*see* AB Register)
AB Byte Counter (*see* ABC)
AB Register:
  Data Flow 2-53 (Fig. 2-32)
  Input 2-52
  Introduction 1-64
  Output 2-54
AB Register Byte Counter (*see* ABC)
ABC:
  Discussion 2-54
  Introduction 1-65
AC Power Distribution 5-1
Add A, (5A); Fix Pt, RX 3-31
Add, AP (FA); Dec, SS:
  Complement Add Sequence 3-113
  Discussion 3-109
  GIS 3-110
  True Add Sequence 3-110
Add, AR (1A); Fix Pt, RR 3-31
Add Halfword, AH (4A); Fix Pt, RX 3-32
Add Logical, AL (5E); Fix Pt, RX 3-33
Add Logical, ALR (1E); Fix Pt, RR 3-32
Add Normalized, AD (6A); Fl Pt, RX (Long) 3-77
Add Normalized, ADR (2A); Fl Pt, RR (Long) 3-76
Add Normalized, AE (7A); Fl Pt, RX (Short) 3-75
Add Normalized, AER (3A); Fl Pt, RR (Short) 3-71
Add-Type Instructions, Floating-Point 3-69
Add Unnormalized, AU (7E); Fl Pt, RX (Short) 3-78
Add Unnormalized, AUR (3E); Fl Pt, RR (Short) 3-78
Add Unnormalized, AW (6E); Fl Pt, RX (Long) 3-79
Add Unnormalized, AWR (2E); Fl Pt, RR (Long) 3-79
Addition, Binary 2-67
Additional Storage Attachment Feature:
  Description 1-2
  Floating Address Switches 4-13
  Invalid Storage Addressing 4-8
  Multiprocessing Feature 4-2
  Number 4-1 (Table 4-1)
Address Buses, BCU 1-51
ADDRESS COMPARE STOP Switch 6-9
Address Decoding, Storage; Multiprocessing System 4-8
Address, Invalid Storage; Multiprocessing System 4-8
Address Relocation, Multiprocessing System 4-3, 4-8
Address Sequencer, Scan Logic 6-31
Address Sequencer Decoder, Scan Logic 6-33
Address Store Compare (*see* ASC)
ADDRESS Switches:
  Basic System 6-9
  Multiprocessing System 4-13
Addressing, Floating; Multiprocessing System 4-2, 4-6
Addressing, Instruction:
  Incrementing IC 2-50
  Introduction 1-56
  Invalid Address Detection 3-19
  Invalid Instruction Address Microprogram 3-21
  Specification Detection 3-18
Addressing, Local Storage; Manually 6-9
Addressing, Main Storage:
  Manually 6-9
  SET IC Pushbutton 6-12
Addressing, Operand 1-56
Addressing Program Interruption 1-34
Addressing, ROS:
  Introduction 1-42
  Manually 6-9
  REPEAT ROS ADDRESS Switch 6-16
  ROSAR 2-12
  ROSAR(0–10) Decoding 2-13
  ROSAR(11) Function 2-13
Alternate Test Number (*see* ATN)
AND, N (54); Lgic, RX 3-149
AND, NC (D4); Lgic, SS 3-150
AND, NI (94); Lgic, SI 3-150
AND, NR (14); Lgic, RR 3-149
AND Function:
  Introduction 1-68
  Logical Instructions 3-148
  Parity Generation 2-72
  Serial Adder Operation 2-68
Array Drivers, ROS 2-14
ASC Test 3-9
ATN, ROS Tests 6-26
Audible Alarm 5-12

B-Register (*see* AB Register)
Balance Lines, CROS 2-6
Base Address 1-14
BCU:
  Channel Request 1-54
  Clocks 1-53
  Control Considerations, Basic 2-24
  Converting SAB Parity 2-34
  CPU Clock, Stopping of 2-29
  CPU Request 1-54
  CPU Sequencers 2-29
  Establishing Priority 2-29
  Gating Address to SAB 2-29
  General Description 2-20
  Initial Handling of Requests 2-28
  Interface, Basic Considerations 2-20
  Interface Lines, Major 1-51
  Invalid Address Detection 2-38
  Modifications to, Multiprocessing System 4-8
  Operation with Channels 2-22
  Operation with CPU 2-21
  Operation with LCS 2-24
  Operation with Main Storage 2-22
  Operational Sequence, Basic 2-26
  Priority Logic 1-53
  Resetting of 2-38
  Select Signal, Generation of 2-34
  Storage Errors 2-38
  Storage Operation, Multiprocessing System 4-9
  Storage Unit, Selection of 2-30
  Timing Considerations, Basic 2-24
Bit Capacitors, CROS 2-10
Bit-Level Carry-Into Logic, Parallel Adder 2-78
Bit Plates, CROS 2-6
Bit Position Logic, Parallel Adder 2-73

Bit Tests, ROS Tests:
Discussion 6-49
ROS Test State 0 6-50
ROS Test State 1 6-50
ROS Test State 2 6-50
ROS Test State 3 6-50
ROS Test State 4 6-49
ROS Test State 5 6-49
ROS Test State 6 6-49
ROS Test State 7 6-49
Block I-Fetch Trigger:
Discussion 3-13
Function 2-89 (Table 2-2)
Branch and Link, BAL (45); Br, RX 3-168
Branch and Link, BALR (05); Br, RR 3-166
Branch Invalid Address Trigger 2-89 (Table 2-2)
Branch on Condition, BC (47); Br, RX 3-165
Branch on Condition, BCR (07); Br, RR 3-164
Branch on Count, BCT (46); Br, RX 3-170
Branch on Count, BCTR (06); Br, RR 3-169
Branch on Index High, BXH (86); Br, RS 3-170
Branch on Index Low or Equal, BXLE (87); Br, RS 3-172
Branch Instructions:
Branch and Link, BAL (45); RX 3-168
Branch and Link, BALR (05); RR:
Discussion 3-166
Successful Branch 3-167
Unsuccessful Branch 3-166
Branch on Condition, BC (47); RX 3-165
Branch on Condition, BCR (07); RR:
Discussion 3-164
Successful Branch 3-164
Unsuccessful Branch 3-165
Branch on Count, BCT (46); RX 3-170
Branch on Count, BCTR (06); RR:
Discussion 3-169
Successful Branch 3-170
Unsuccessful Branch 3-170
Branch on Index High, BXH (86); RS 3-170
Branch on Index Low or Equal, BXLE (87); RS 3-172
CC 1-94
Data Flow 1-94
Discussion 1-91
Execute, EX (44); RX 3-173
Fetching 1-63
Format 1-92
List of 1-93 (Table 1-12)
Program Interruptions 1-94
Branching, ROS:
Discussion 1-42
No Branch 1-43
Overriding Branch 1-47
X-Branch 1-43
Y-Branch 1-43
Z-Branch 1-43
Bus Control Unit (*see* BCU)
Buses, BCU 1-51, 2-20

Capacitive Read-Only Storage (*see* CROS)
Carry-Into-Bit Logic, Parallel Adder 2-76
Carry Lookahead, Parallel Adder 2-77
Carry, Parallel Adder:
Bit-Level Carry-Into Logic 2-78
Carry-Into-Bit Logic 2-76
Group-Level Carry 2-77
Group-Level Carry-Into Logic 2-78
Lookahead 2-77
Section-Level Carry 2-78
Section-Level Carry-Into Logic 2-78
CAW 1-37
CC:
Branching Instructions 1-94
Decimal Instructions 1-87
Fixed-Point Instructions 1-75
Floating-Point Instructions 1-82
I/O Instructions 1-99
Logical Instructions 1-91
Parallel Adder Function 2-85
PSW 1-25
Status Switching Instructions 1-97
CCW 1-37
CE Key Switch 5-13, 6-19
CE Panel:
CPU ON Pushbutton 6-21
CPU READY/OFF Switch 6-21
THERMAL RESET Pushbutton 6-21
THERMAL TRIP Indicators 6-21
UNDER VOLTAGE CHECK Switches 6-21
CHAN FRAME 1 POWER CHECK Indicator 5-12, 6-20
CHAN FRAME 2 POWER CHECK Indicator 5-12, 6-20
CHAN FRAME 3 POWER CHECK Indicator 5-12, 6-20
Channel:
Introduction 1-7
Modes of Operation 1-7
Operation with BCU 2-22
2860 Selector 1-7
2870 Multiplexer 1-7
Channel Address Word (*see* CAW)
Channel Command Word (*see* CCW)
Channel, MCW Application 6-23
Channel/Scan Interface 6-40
Channel Status Word (*see* CSW)
Channel-to-Channel Adapter 1-8
Check Counter, Scan Logic 6-34
Check Indicators, Power 5-12
CHECK RESET Pushbutton 6-6
Circuits, Special A-1
Clock:
BCU 1-53
Control 2-3
CPU 1-39
FLT 6-30
FREQUENCY ALTERATION Switch 6-19
Scan 6-30
Scan Stop-CPU-Clock Logic 6-39
Signal Distribution 2-3
Signal Generators 2-1
Timing 2-2
Commands, I/O 1-37
Compare, C (59); Fix Pt, RX 3-36
Compare, CD (69); Fl Pt, RX (Long) 3-86
Compare, CDR (29); Fl Pt, RR (Long) 3-85
Compare, CE (79); Fl Pt, RX (Short) 3-85
Compare, CER (39); Fl Pt, RR (Short) 3-84
Compare, CP (F9); Dec, SS 3-114
Compare, CR (19); Fix Pt, RR 3-36
Compare Halfword, CH (49); Fix Pt, RX 3-37
Compare Logical, CL (55); Lgic, RX 3-147
Compare Logical, CLC (D5); Lgic, SS 3-148
Compare Logical, CLI (95); Lgic, SI 3-147
Compare Logical, CLR (15); Lgic, RR 3-147

Configuration Control Panel, Multiprocessing System:
Additional Storage Attach Feature 4-14, 4-15
CPU Mode Switches 4-13
Discussion 4-7
Floating Address Switches 4-13
I/O Allocation Switches 4-13
PREFIX Switches 4-13
Storage Allocation Switches 4-13
VALID ADDRESS Indicators 4-14
Continue or Terminate, FLT 6-55, 6-56
Control Lines, BCU 2-20
Control Program 1-22
Control Signals, BCU 1-52
Control Triggers 2-89
Control Unit, I/O 1-8
Convert to Binary, CVB (4F); Fix Pt, RX 3-49
Convert to Decimal, CVD (4E); Fix Pt, RX 3-51
Convert to Decimal Operation, Parallel Adder 2-85
Converter/Inverter 5-1
CPU, Basic Description:
BCU 1-5
Control Section 1-5
Instruction Execution Section 1-6
Instruction Fetching Section 1-5
CPU CHECK Switch 6-17
CPU Clock:
Scan Stop-CPU-Clock Logic 6-39
Stopping of, by BCU 2-29
CPU, MCW Application 6-23
CPU Mode Switches, Multiprocessing System 4-13
CPU ON Pushbutton 6-21
CPU POWER CHECK Indicator 5-12, 6-20
CPU READY/OFF Switch 6-21
CPU Sequencers 2-29
CPU Store in Progress Exceptional Condition:
Discussion 1-36
Effect on I-Fetch 3-14
CROS:
Electrical Theory 2-6
Planes 2-6
CSW 1-38
Current PSW 1-26, 1-29
Cycle Counter, Scan Logic 6-37

D-Register:
Branch Function 2-51
Data Flow 2-51 (Fig. 2-31)
Fixed-Point Function 2-52
Floating-Point Function 2-52
Input 2-50
Interruption Function 2-52
Introduction 1-58
I/O Function 2-52
Manual Control Function 2-52
Output 2-50
Shift Function 2-52
VFL Function 2-52
Data Buses, BCU 1-51
Data Entry, Manually 6-9
Data Flow:
AB Register 2-53 (Fig. 2-32)
Branching Instructions 1-94
D-Register 2-51 (Fig. 2-31)
Decimal Instructions 1-86
E-Register 2-48 (Fig. 2-29)
F-Register 2-58 (Fig. 2-35)
Fixed-Point Instructions 1-70
Floating-Point Instructions 1-76
G-Register 2-59 (Fig. 2-36)
IC 2-49 (Fig. 2-30)
I/O Instructions 1-98
Local Storage 2-62 (Fig. 2-40)
Logical Instructions 1-90
MCW Register 2-60 (Fig. 2-39)
Parallel Adder 2-74 (Fig. 2-49)
PSW Register 2-60 (Fig. 2-37)
Q-Register 2-45 (Fig. 2-26)
R-Register 2-47 (Fig. 2-28)
ROS 1-45 (Fig. 1-19), 1-47, 2-14
Serial Adder 2-65 (Fig. 2-41)
ST Register 2-55 (Fig. 2-33)
Status Switching Instructions 1-95
System 1-3
Data Formats:
Decimal 1-21
Fixed-Point 1-16
Floating-Point 1-19
Logical 1-21
Data Program Interruption 1-34
DATA Switches 6-9
Data Transfer 1-39
DC Power Distribution 5-4
Decimal Correction, Serial Adder 2-67
Decimal Data:
Discussion 1-20
Formats 1-21
Number Representation 1-20
Decimal Divide Program Interruption 1-34
Decimal Instructions:
Add, AP (FA); SS:
Complement Add Sequence 3-113
Discussion 3-109
GIS 3-110
True Add Sequence 3-110
CC 1-87
Compare, CP (F9); SS 3-114
Data Flow 1-86
Data Handling 1-82
Divide, DP (FD); SS:
Detailed Description 3-134
Discussion 3-125
General Description 3-128
Operand Specifications 3-127
Typical Sequence 3-128
Format 1-86
GIS 3-109, 3-110
Instruction Handling 3-106
List of 1-83 (Table 1-10)
Move with Offset, MVO (F1); SS:
Discussion 3-141
Not Word Overlap 3-142
Word Overlap 3-142
Multiply, MP (FC); SS:
Add Sequence 3-117
Detailed Description 3-124
Discussion 3-116
General Description 3-118
Operand Specifications 3-117
Subtract Sequence 3-117
Pack, PACK (F2); SS:
Discussion 3-137
Not Word Overlap 3-138
Word Overlap 3-139

Program Interruptions 1-87
Serial Adder Functions 3-106
Subtract, SP (FB); SS:
Complement Add Sequence 3-113
Discussion 3-109
GIS 3-110
True Add Sequence 3-110
Unpack, UNPK (F3); SS:
Discussion 3-139
Not Word Overlap 3-140
Word Overlap 3-141
Word Overlap 3-107
Zero and Add, ZAP (F8); SS 3-115
Decimal Overflow Program Interruption 1-34
DEFEAT INTERLEAVING Switch:
Basic System 6-12
Multiprocessing System 4-14
Device Switching Unit 1-9
Diagnose (83); Stat Sw, SI:
I2 Field Format 6-22
MCW, Role of 6-22, 6-23
Operation 3-182
Diagnostic Programs 6-57
Direct Control, Multiprocessing System 4-4
Direct Control Feature:
DISABLE DIRECT CONTROL Switch 6-17
Introduction 1-2
Multisystem Configuration 1-8
Power 5-12
Read Direct, RDD (85); Stat Sw, SI 3-181
Write Direct, WRD (84); Stat Sw, SI 3-180
DISABLE DIRECT CONTROL Switch:
Multiprocessing System Operation 4-14
Operation 6-17
DISABLE INTERVAL TIMER Switch 6-17
Displacement 1-14
DISPLAY Pushbutton 6-15
Divide, D (5D); Fix Pt, RX 3-47
Divide, DD (6D); Fl Pt, RX (Long) 3-103
Divide, DDR (2D); Fl Pt, RR (Long) 3-102
Divide, DE (7D); Fl Pt, RX (Short) 3-101
Divide, DER (3D); Fl Pt, RR (Short) 3-100
Divide, DP (FD); Dec, SS 3-125
Divide, DR (1D); Fix Pt, RR 3-43
Divide, Fixed-Point:
Divisor Multiple Selection 3-44
DVDL0 Micro-Order Function 3-44
DVDL1 Micro-Order Function 3-44
Examples 3-48 (Figs. 3-8 and 3-9)
Partial Quotient Bits, Determination of 3-44
Divide, Floating-Point:
Characteristic Computation 3-95
Data Flow and Algorithm 3-98
DVDL0 Micro-Order Function 3-99
DVDL1 Micro-Order Function 3-99
Example 3-102
Fraction Division 3-96
Normalization 3-96
Division, Non-Restoring 3-96
Division, Restore 3-96
Divisor Multiple Selection, Fixed-Point 3-44
Drive Lines, CROS 2-6, 2-14
E-Register:
Data Flow 2-48 (Fig. 2-29)
Incrementers 2-46
Input 2-46
Introduction 1-56
Output 2-46
Early End Op:
Discussion 3-3
Instruction Fetching 3-4
EBCDIC 1-12
Edit, ED (DE); Lgic, SS 3-158
Edit and Mark, EDMK (DF); Lgic, SS 3-158
Emergency Power-Off Control Feature 1-2
EMERGENCY PULL Switch 5-8
End Op:
Branch 3-2
Discussion 3-1
Function 1-55
Instruction Fetching 3-3
Normal 3-1
Normal, Deviations from 3-12
Operand Prefetching 3-2
End-Operation (*see* End Op)
EPO Switch 5-8
Error Checking:
Parallel Adder 2-84
ROS 2-18
Error Detection:
CPU CHECK Switch 6-17
PROC CHK Indicator 6-20
Serial Adder 2-72
STOP ON STORAGE CHECK Switch 6-12
STOR CHK Indicator 6-20
Exceptional Conditions:
CPU Store in Progress:
Discussion 1-36
Effect on I-Fetch 3-14
Discussion 1-26
Invalid Instruction Address Test:
Discussion 1-36
Effect on I-Fetch 3-18
Manual Control Repeat:
Discussion 1-36
Effect on I-Fetch 3-17
Manual Control Stop:
Discussion 1-36
Effect on I-Fetch 3-17
Manual Control Wait:
Discussion 1-36
Effect on I-Fetch 3-17
Priority 1-27
Program Store Compare:
Discussion 1-36
Effect on I-Fetch 3-17
Q-Register Refill:
Discussion 1-36
Effect on I-Fetch 3-22
Recovery from 1-64
Timer:
Discussion 1-36
Effect on I-Fetch 3-14
Excess-6 Arithmetic:
Complement Add Examples 1-85
Introduction 1-66
Serial Adder Operation 2-67
True Add Examples 1-85

Exclusive-OR, X (57); Lgic, RX 3-153
Exclusive-OR, XC (D7); Lgic, SS 3-153
Exclusive-OR, XI (97); Lgic, SI 3-153
Exclusive-OR, XR (17); Lgic, RR 3-152
Exclusive-OR Function:
  Discussion 3-152
  Parity Generation 2-72
  Serial Adder Operation 2-68
Execute, EX (44); Br, RX 3-173
Execute Program Interruption 1-34
Exponent Overflow 3-63
Exponent Overflow Program Interruption 1-34
Exponent Underflow 3-63
Exponent Underflow Program Interruption 1-34
Extended Binary-Coded-Decimal Interchange Code (*see* EBCDIC)
External Interruption:
  Discussion 1-35
  Effect on I-Fetch 3-16
External Start, Multiprocessing System 4-12
External Start Signal, Multiprocessing System 4-6, 4-12
External System Reset, Multiprocessing System 4-12
External System Reset Timing, Multiprocessing System 4-10

F-Register:
  Data Flow 2-58 (Fig. 2-35)
  Discussion 1-66
  Input 2-58
  Output 2-58
Fail Trigger, Scan Logic 6-40
Feature Index 4-1 (Table 4-1)
Features:
  Index 4-1 (Table 4-1)
  List of 1-2
  Multiprocessing 4-2
  Multisystem (*see* Multiprocessing System)
Fetch Protection, Detection of 3-21
Fetch Requests 2-22, 2-24, 2-40
50-Hz Power Distribution 5-1
Fixed-Point Data:
  Discussion 1-16
  Formats 1-16
  Number Representation 1-16
Fixed-Point Divide Program Interruption 1-34
Fixed-Point Instructions:
  Add, A (5A); RX 3-31
  Add, AR (1A); RR 3-31
  Add Halfword, AH (4A); RX 3-31
  Add Logical, AL (5E); RX 3-33
  Add Logical, ALR (1E); RR 3-32
  Add-Type, Discussion 3-30
  CC 1-75
  Compare, C (59); RX 3-36
  Compare, CR (19); RR 3-36
  Compare Halfword, CH (49); RX 3-37
  Convert to Binary, CVB (4F); RX 3-49
  Convert to Decimal, CVD (4E); RX 3-51
  Data Flow 1-70
  Divide, D (5D); RX 3-47
  Divide, Discussion 3-42
  Divide, DR (1D); RR 3-43
  Formats 1-70
  List of 1-71 (Table 1-8)
  Load, L (58); RX 3-25
  Load, LR (18); RR 3-25
  Load and Test, LTR (12); RR 3-27
  Load Complement, LCR (13); RR 3-27
  Load Halfword, LH (48); RX 3-26
  Load Multiple, LM (98); RS 3-29
  Load Negative, LNR (11); RR 3-28
  Load Positive, LPR (10); RR 3-28
  Multiply, Discussion 3-37
  Multiply, M (5C); RX 3-42
  Multiply, MR (1C); RR 3-38
  Multiply Halfword, MH (4C); RX 3-42
  Program Interruptions 1-70
  Shift Left Double, SLDA (8F); RS 3-58
  Shift Left Single, SLA (8B); RS 3-56
  Shift Right Double, SRDA (8E); RS 3-61
  Shift Right Single, SRA (8A); RS 3-60
  Store, ST (50); RX 3-53
  Store Halfword, STH (40); RX 3-54
  Store Multiple, STM (90); RS 3-55
  Subtract, S (5B); RX 3-34
  Subtract, SR (1B); RR 3-33
  Subtract Halfword, SH (4B); RX 3-34
  Subtract Logical, SL (5F); RX 3-35
  Subtract Logical, SLR (1F); RR 3-35
Fixed-Point Overflow Program Interruption 1-34
Floating Address Switches, Multiprocessing System 4-13
Floating Addressing, Multiprocessing System 4-2, 4-7
Floating-Point Data:
  Discussion 1-17
  Formats 1-19
  Normalization 1-19
  Number Representation 1-17
Floating-Point Divide Program Interruption 1-34
Floating-Point Instructions:
  Add Normalized, AD (6A); RX (Long) 3-77
  Add Normalized, ADR (2A); RR (Long) 3-76
  Add Normalized, AE (7A); RX (Short) 3-75
  Add Normalized, AER (3A); RR (Short) 3-71
  Add-Type, Discussion 3-69
  Add Unnormalized, AU (7E); RX (Short) 3-78
  Add Unnormalized, AUR (3E); RR (Short) 3-78
  Add Unnormalized, AW (6E); RX (Long) 3-79
  Add Unnormalized, AWR (2E); RR (Long) 3-79
  CC 1-82
  Compare, CD (69); RX (Long) 3-86
  Compare, CDR (29); RR (Long) 3-85
  Compare, CE (79); RX (Short) 3-85
  Compare, CER (39); RR (Short) 3-84
  Conditions at Start of Execution 3-64
  Data Flow 1-76
  Discussion 1-75
  Divide:
    Characteristic Computation 3-95
    Data Flow and Algorithm 3-98
    Discussion 3-94
    Fraction Division 3-96
    Normalization 3-96
  Divide, DD (6D); RX (Long) 3-103
  Divide, DDR (2D); RR (Long)) 3-102
  Divide, DE (7D); RX (Short) 3-101
  Divide, DER (3D); RR (Short) 3-100
  Exponent Overflow 3-63
  Exponent Underflow 3-63
  Formats 1-76
  Halve, HDR (24); RR (Long) 3-87
  Halve, HER (34); RR (Short) 3-86
  List of 1-77 (Table 1-9)
  Load, LD (68); RX (Long) 3-65
  Load, LDR (28); RR (Long) 3-65

Load, LE (78); RX (Short) 3-64
Load, LER (38); RR (Short) 3-64
Load and Test, LTDR (22); RR (Long) 3-66
Load and Test, LTER (32); RR (Short) 3-66
Load Complement, LCDR (23); RR (Long) 3-67
Load Complement, LCER (33); RR (Short) 3-67
Load Negative, LNDR (21); RR (Long) 3-69
Load Negative, LNER (31); RR (Short) 3-68
Load Positive, LPDR (20); RR (Long) 3-68
Load Positive, LPER (30); RR (Short) 3-68
Multiply:
Data Flow and Algorithm 3-88
Discussion 3-87
Multiply, MD (6C); RX (Long) 3-93
Multiply, MDR (2C); RR (Long) 3-93
Multiply, ME (7C); RX (Short) 3-92
Multiply, MER (3C); RR (Short) 3-91
Program Interruptions 1-81
Store, STD (60); RX (Long) 3-105
Store, STE (70); RX (Short) 3-104
Subtract Normalized, SD (6B); RX (Long) 3-82
Subtract Normalized, SDR (2B); RR (Long) 3-81
Subtract Normalized, SE (7B); RX (Short) 3-81
Subtract Normalized, SER (3B); RR (Short) 3-80
Subtract Unnormalized, SU (7F); RX (Short) 3-83
Subtract Unnormalized, SUR (3F); RR (Short) 3-82
Subtract Unnormalized, SW (6F); RX (Long) 3-84
Subtract Unnormalized, SWR (2F); RR (Long) 3-83
Zero Results 3-63
FLT:
Clock 6-30
Clock Advance 6-27
Counter 6-36
Discussion 6-51
Format 6-29, 6-52
Generation of, Example C-1
Hardcore Tests 6-27, 6-28, 6-51, 6-54
Hardcore Tests 1 and 2 6-51
Introduction 6-26
IPL 1 6-51, 6-53
IPL 2 6-52
Loader 6-51, 6-53
MCW Application 6-24
One-Cycle Tests:
Discussion 6-54
Introduction 6-52
Result Comparison 6-55, 6-56
Scan In 6-52, 6-55
Scan Out 6-54, 6-56
Terminate or Continue 6-55, 6-56
Test Cycles 6-54, 6-56
TN/ATN Comparison 6-55, 6-57
RESTART FLT I/O Pushbutton 6-19
Scan In 6-27
Scan-In Words 6-52
Scan Out 6-27
Setup 6-53
Tape Generation 6-27
Tape Make-Up 6-51
TEST MODE, REPEAT Switch 6-19
TEST MODE, ROS/PROC/FLT Switch 6-18
Transmission Checks 6-54
Zero-Cycle Tests:
Discussion 6-54
Introduction 6-52
Result Comparison 6-55, 6-56
Scan In 6-54, 6-55
Scan Out 6-54, 6-56
Terminate or Continue 6-55, 6-56
Test Cycles 6-54, 6-56
TN/ATN Comparison 6-55, 6-57
FLT Test Trigger 6-50
Formats:
Data 1-15
Instruction 1-13, 1-14 (Fig. 1-11)
4-Cycle Fetch Request 2-22, 2-26, 2-40
FREQUENCY ALTERATION Switch 6-19
Full Sum:
Checking, Parallel Adder 2-84
Development, Parallel Adder 2-78
Logic, Parallel Adder 2-76
Functional Units:
AB Register 2-52
ABC 2-54
Address Sequencer, Scan Logic 6-31
Address Sequencer Decoder, Scan Logic 6-33
BCU 2-20
BCU, Modification; Multiprocessing System 4-8
Check Counter, Scan Logic 6-34
Clock Signal Generator 2-1
Cycle Counter, Scan Logic 6-37
D-Register 2-50
E-Register 2-46
F-Register 2-58
FLT Clock 6-30
FLT Counter 6-36
G-Register 2-58
IC 2-49
Local Storage 2-61
Mark Triggers 2-58
MCW Register 2-59
Multiprocessing System 4-6
Multisystem Timer, Multiprocessing System 4-9
Parallel Adder 2-73
PSW Register 2-58
Q-Register 2-44
R-Register 2-46
ROS 2-6
ROS Test Sequencer 6-38
Scan Counter Latches and Decrementer 6-31
Scan Logic 6-29
Scan-Out Bus 6-38
Serial Adder 2-64
ST Register 2-54
STAT's 2-86
STC 2-54, 2-57
Storage Address Generator, Scan Logic 6-34

G-Register:
Data Flow 2-59 (Fig. 2-36)
Discussion 2-58
Introduction 1-66
Gated Load Pb Signal, Multiprocessing System:
Function 4-6
Operation 4-11
General Initialization Sequence (*see* GIS)
GIS:
Decimal Instructions 3-109, 3-110
Logical Instructions 3-144
Group-Level Carry-Into Logic, Parallel Adder 2-78
Group-Level Carry Logic, Parallel Adder 2-77

Half Adder, Parallel Adder 2-76
Half-Sum Checking, Parallel Adder 2-84
Halt I/O, HIO (9E); I/O, SI 3-186
Halve, HDR (24); Fl Pt, RR (Long) 3-87
Halve, HER (34); Fl Pt, RR (Short) 3-86
Hardcore Tests, FLT 6-27, 6-28, 6-51, 6-54
Hardcore Tests 1 and 2, FLT's 6-51
Hardcore Tests, ROS Tests:
  Discussion 6-45
  Format 6-45
  IPL 2 6-45, 6-47
  Record 6 6-47
  Summary of 6-48
  Tape Runaway 6-48
  Test 1, Record 3 6-46
  Test 1, Record 7 6-48
  Test 1, Record 8 6-48
  Test 2, Record 5 6-47
  Test 2, Record 8 6-48
  Test 3, Record 8 6-48
Hexadecimal Number System 1-12
High-Frequency Regulator Modules 5-4
High-Speed Storage (*see* Main Storage)

I-Fetch:
  ASC Test 3-9
  Basic, Deviations from 3-12
  Block I-Fetch Trigger 3-13
  Branching Instructions 1-63
  CPU Store in Progress Exceptional Condition 3-14
  Early End Op 3-3
  End Op, Branch 3-2
  End-Op Cycle 1-55
  End Op, Early 3-3
  End Op, Normal 3-1
  Exceptional Conditions, Recovery from 1-64
  External Interruption 3-16
  Fetch Protection, Detection of 3-21
  Functional Units Used 1-55
  General 3-1
  Instruction Address Specification, Detection of 3-18
  Instruction Addressing 1-56
  Instruction Path 1-58
  Interruption Routine, Common 3-17
  Interruptions, Recovery from 1-64
  Introduction 1-55
  Invalid Address Detection 3-19
  Invalid Instruction Address Microprogram 3-21
  Invalid Instruction Address Test Exceptional Condition 3-18
  I/O Interruption 3-16
  Machine Check Interruption 3-14
  Manual Control Repeat Exceptional Condition 3-17
  Manual Control Stop Exceptional Condition 3-17
  Manual Control Wait Exceptional Condition 3-17
  Microprogram Selection 3-4
  Obtaining New Instructions 1-63
  Op-Code Halfword, Transfer to R-Register 1-56
  Operand Addressing 1-56
  Operand Prefetching 1-60, 3-2
  Program Interruption 3-15
  Program Store Compare Exceptional Condition 3-17
  Q-Register Refill Exceptional Condition 3-22
  Refilling of Q-Register 1-55, 1-63
  RR, Basic 3-5
  RR, 2-Cycle 3-23
  RS, Basic 3-6
  RS, 2-Cycle 3-24
  RX, Basic 3-6
  RX, Forced Cycle 3-24
  Sequencers 3-12
  SI, Basic 3-6
  SI, 2-Cycle 3-24
  SS:
    ASC Test 3-9
    Detailed Description 3-9
    General 3-7
  Storage Requests 2-28
  Supervisor Call Interruption 3-15
  Timer Exceptional Condition 3-14
I-Fetch Invalid Address Trigger 2-89 (Table 2-2)
IC:
  Data Flow 2-49 (Fig. 2-30)
  Incrementing IC(0–20) 2-50
  Incrementing IC(21–23) 2-50
  Input 2-49
  Introduction 1-56
  Op-Code Halfword Selection 2-44, 2-50
  Output 2-49
  Parity Adjust Logic 2-50
  SET IC Pushbutton 6-12
IC in LSWR Trigger 2-90 (Table 2-2)
ILC, PSW 1-25, 1-32
Incrementers, E-Register 2-46
Index Value 1-14
Indicators (*see* Specific Indicator or Panel)
Initial Program Load (*see* IPL)
Input/Output (*see* I/O)
Insert Character, IC (43); Lgic, RX 3-154
Insert Storage Key 2-23, 2-41
Insert Storage Key, ISK (09); Stat Sw, RR 3-179
Instruction Address, PSW 1-26
Instruction Addressing:
  B-Field Transfer to LAL 2-44
  D-Field Transfer to Parallel Adder 2-44
  Doubleword 1-56
  Incrementing IC 2-50
  Op-Code Halfword Selection 1-57, 2-50
Instruction Counter (*see* IC)
Instruction Execution, Effect on by RATE Switch:
  Discussion 6-13
  INSN STEP Position 6-13
  PROCESS Position 6-13
  SINGLE CYCLE Position 6-13
  SINGLE CYCLE STORAGE INHIBIT Position 6-13
Instruction Execution, Effect on by REPEAT INSN Switch:
  Discussion 6-13
  Repeat Multiple Instructions 6-14
  Repeat Single Instruction 6-14
Instruction Fetching (*see* I-Fetch)
Instruction Format:
  Branching 1-92
  Decimal 1-86
  Fixed-Point 1-70
  Floating-Point 1-76
  Introduction 1-13
  I/O 1-98
  Logical 1-87
  Status Switching 1-95
Instruction Length Not Available Trigger 2-89 (Table 2-2)
Instruction Predecoding 2-46
Instructions (*see* Specific Instruction; or Class of Instructions, e.g., Fixed-Point Instructions)

Integral Boundaries, Main Storage 1-13
Interface, BCU 1-51, 2-20
Interleaving:
DEFEAT INTERLEAVING Switch 6-12
Main Storage 2-22
Protection Key Assignments, Effect on 3-178
Intermittent Trigger, Scan Logic 6-40
INTERRUPT Pushbutton 6-17
Interruptable State 1-24
Interruption Code, PSW 1-25
Interruptions:
Associated PSW's 1-29
Common Routine 3-17
D-Register Function 2-52
Discussion 1-26, 1-29
Effect on PSW 1-29
External:
Discussion 1-35
Effect on I-Fetch 3-16
ILC 1-32
Instruction Completion 1-27
Instruction Suppression 1-27
Instruction Termination 1-27
INTERRUPT Pushbutton 6-17
I/O:
Discussion 1-35
Effect on I-Fetch 3-16
Log I/O, Multiprocessing System 4-11
Machine Check:
Discussion 1-33
Effect on I-Fetch 3-14
Masking 1-29
Priority 1-27
Program:
Discussion 1-33
Effect on I-Fetch 3-15
Recovery from 1-64
Supervisor Call:
Discussion 1-35
Effect on I-Fetch 3-15
Interval Timer:
DISABLE INTERVAL TIMER Switch 6-17
Disabling 6-17
Invalid Address Detection 2-38
Invalid Instruction Address Test Exceptional Condition:
Discussion 1-36
Effect on I-Fetch 3-18
Invalid Storage Address, Multiprocessing System 4-9
I/O:
Burst Mode of Operation 1-7
CAW 1-37
CCW 1-37
Channel 1-7
Commands 1-37
Control Unit 1-8
CSW 1-38
D-Register Function 2-52
Devices 1-8
Interruption:
Discussion 1-35
Effect on I-Fetch 3-16
Introduction 1-6
Multiplex Mode of Operation 1-7
Multiprocessing System 4-4
Operations 1-37
Orders 1-37
System Operation 1-38
I/O Allocation, Multiprocessing System 4-4, 4-7
I/O Allocation Switches, Multiprocessing System 4-14
I/O Instructions:
CC 1-99
Data Flow 1-98
Discussion 3-184
Halt I/O, HIO (9E); SI 3-186
Instruction Format 1-98
List of 1-98 (Table 1-14)
Program Interruption 1-99
Start I/O, SIO (9C); SI 3-184
Test Channel, TCH (9F); SI 3-187
Test I/O, TIO (9D); SI 3-186
IPL 6-6
IPL 1, FLT's 6-51, 6-53
IPL 1, ROS Tests 6-44
IPL 2, FLT's 6-52
IPL 2, ROS Tests 6-45, 6-47

Key, PSW 1-25
Key Switch, CE 5-13

Large-Capacity Storage (*see* LCS)
Latch, Definition 2-2
Latch-Shifter Logic, Parallel Adder 2-76
LCS:
Introduction 1-4
Operation with BCU 2-24
Select Signal, Generation of, by BCU 2-37
Leave Trigger 2-89 (Table 2-2)
Load, L (58); Fix Pt, RX 3-25
Load, LD (68); Fl Pt, RX (Long) 3-65
Load, LDR (28); Fl Pt, RR (Long) 3-65
Load, LE (78); Fl Pt, RX (Short) 3-64
Load, LER (38); Fl Pt, RR (Short) 3-64
Load, LR (18); Fix Pt, RR 3-25
Load Address, LA (41); Lgic, RX 3-155
Load and Test, LTDR (22); Fl Pt, RR (Long) 3-66
Load and Test, LTER (32); Fl Pt, RR (Short) 3-66
Load and Test, LTR (12); Fix Pt, RR 3-27
Load Complement, LCDR (23); Fl Pt, RR (Long) 3-67
Load Complement, LCER (33); Fl Pt, RR (Short) 3-67
Load Complement, LCR (13); Fix Pt, RR 3-27
Load Halfword, LH (48); Fix Pt, RX 3-26
LOAD Indicator 6-20
Load Multiple, LM (98); Fix Pt, RS 3-29
Load Negative, LNDR (21); Fl Pt, RR (Long) 3-68
Load Negative, LNER (31); Fl Pt, RR (Short) 3-68
Load Negative, LNR (11); Fix Pt, RR 3-28
Load Positive, LPDR (20); Fl Pt, RR (Long) 3-68
Load Positive, LPER (30); Fl Pt, RR (Short) 3-68
Load Positive, LPR (10); Fix Pt, RR 3-28
Load PSW, LPSW (82); Stat Sw, SI 3-176
LOAD Pushbutton:
Basic System 6-6
Gated Load Pb Signal, Multiprocessing System:
Function 4-6
Operation 4-11
Loader:
FLT 6-51, 6-53
ROS Tests 6-45
Local Storage:
Addressing 2-61
Addressing, Manually 6-9
Data Flow 2-62 (Fig. 2-40)

DISPLAY Pushbutton 6-15
Introduction 1-69
Read Operation 2-61
Register Addresses 1-69
Ripple Tests 6-57
Selection Manually 6-9
Storage-Ripple Microprogram 6-16
STORE Pushbutton 6-14
Timing 2-63
Write Operation 2-63
Log I/O Interrupt Signal, Multiprocessing System 4-6, 4-12
Log I/O Interruption, Multiprocessing System 4-12
LOG OUT Pushbutton 6-18
Logical Data 1-21
Logical Instructions:
AND, Discussion 3-148
AND, N (54); RX 3-149
AND, NC (D4); SS 3-150
AND, NI (94); SI 3-150
AND, NR (14); RR 3-149
CC 1-91
Compare, CL (55); RX 3-147
Compare, CLC (D5); SS 3-148
Compare, CLI (95); SI 3-147
Compare, CLR (15); RR 3-147
Data Flow 1-90
Edit, ED (DE); SS 3-158
Edit and Mark, EDMK (DF); SS 3-158
Exclusive-OR, Discussion 3-152
Exclusive-OR, X (57); RX 3-153
Exclusive-OR, XC (D7); SS 3-153
Exclusive-OR, XI (97); SI 3-153
Exclusive-OR, XR (17); RR 3-152
Formats 1-87
GIS 3-144
Insert Character, IC (43); RX 3-154
List of 1-88 (Table 1-11)
Load Address, LA (41); RX 3-155
Move, MVC (D2); SS 3-144
Move, MVI (92); SI 3-144
Move Numerics, MVN (D1); SS 3-145
Move Zones, MVZ (D3); SS 3-146
OR, O (56); RX 3-151
OR, OC (D6); SS 3-152
OR, OI (96); SI 3-151
OR, OR (16); RR 3-151
OR, Discussion 3-150
Program Interruptions 1-91
Shift, Discussion 3-162
Shift Left Double, SLDL (8D); RS 3-162
Shift Left Single, SLL (89); RS 3-162
Shift Right Double, SRDL (8C); RS 3-163
Shift Right Single, SRL (88); RS 3-163
Store Character, STC (42); RX 3-154
Test Under Mask, TM (91); SI 3-154
Translate, TR (DC); SS 3-155
Translate and Test, TRT (DD); SS 3-156
Logout:
Controls 6-38
Discussion 6-40
Format 6-25 (Table 6-1)
Hardware-Controlled Sequence 6-41
Introduction 6-25
LOG OUT Pushbutton 6-18
Multiprocessing System 4-15
ROS-Controlled Sequence 6-42
Lookahead, Parallel Adder Carry 2-77
LS (*see* Local storage)

M-Bit, PSW 1-25, 1-32
Machine Check Interruption:
Discussion 1-33
Effect on I-Fetch 3-14
Magnetic-Amplifier-Controlled Regulator 5-5
Main Storage:
Addressing, Manually 6-9
Channel Request 1-54
CPU Request 1-54
Direct Address Relocation, Multiprocessing System 4-3, 4-8
Discussion 1-3
DISPLAY Pushbutton 6-15
Floating Addressing, Multiprocessing System 4-2, 4-7
Integral Boundaries 1-13
Interleaving:
DEFEAT INTERLEAVING Switch 6-12
Discussion 2-22
Multiprocessing System 4-2
Operation with BCU 1-54, 2-22
Permanent Address Assignments 1-4 (Table 1-1)
Protection 1-4, 2-23
Ripple Tests 6-57
Selection, Manually 6-9
STOP ON STORAGE CHECK Switch 6-12
STOR CHK Indicator 6-20
Storage Allocation, Multiprocessing System 4-2, 4-7
STORAGE INDICATE Switch 6-12
Storage-Ripple Microprogram 6-16
STORE Pushbutton 6-14
2361 Core Storage 1-4
2365-1, -2 Processor Storage 1-4
2365-13 Processor Storage 4-2
Maintenance Aids, ROS 2-16
Maintenance Features:
Diagnose (83); Stat Sw, SI 3-182, 6-22
Diagnostic Programs 6-57
FLT's:
Detailed Discussion 6-51
Introduction 6-26
Logout:
Detailed Discussion 6-40
Introduction 6-25
Marginal Checking 6-58
MCW's 6-22
Ripple Tests 6-57
ROS Tests:
Detailed Discussion 6-43
Introduction 6-26
Scan Logic:
Functional Units 6-29
Introduction 6-24
Malfunction Alert, Multiprocessing System 4-11
Malfunction Alert Signal, Multiprocessing System 4-5, 4-11
Manual Control, D-Register Function 2-52
Manual Control Repeat Exceptional Condition:
Discussion 1-36
Effect on I-Fetch 3-17
Manual Control Stop Exceptional Condition:
Discussion 1-36
Effect on I-Fetch 3-17
Manual Control Wait Exceptional Condition:
Discussion 1-36
Effect on I-Fetch 3-17

MANUAL Indicator 6-19
MARGIN Indicators:
Additional Storage Attachment Feature 4-15
Basic Unit 6-20
Multisystem Feature 4-14
Marginal Checking:
Discussion 5-6, 6-58
MARGIN Indicators 6-20
Mark Triggers:
Discussion 2-58
Introduction 1-65
Masked State 1-24
MCW:
Channel 6-23
CPU 6-23
Diagnose (83); Stat Sw, SI 3-182, 6-22
FLT 6-24
Role of 6-22
ROS Test 6-24
MCW Register:
Data Flow 2-60 (Fig. 2-39)
Discussion 2-59
Meters, Usage 5-13
Micro-Orders 1-41
Microprogram-Generated Storage Requests 2-28
Mode Control, Multiprocessing System 4-7
Model 65 Mode, Multiprocessing System 4-5
Move, MVC (D2); Lgic, SS 3-144
Move, MVI (92); Lgic, SI 3-144
Move Numerics, MVN (D1); Lgic, SS 3-145
Move with Offset, MVO (F1); Dec, SS 3-141
Move Zones, MVZ (D3); Lgic, SS 3-146
Multiple Selection Fixed-Point 3-38
Multiply, Fixed-Point:
Multiple Selection 3-38
Partial Product, Formation of 3-39
Partial Product Bits, Extraction of 3-39
RR Format Examples 3-40 (Fig. 3-6), 3-41 (Fig. 3-7)
Termination 3-39
Multiply, Floating-Point:
Data Flow and Algorithm 3-88
Example 3-90
Multiply, M (5C); Fix Pt, RX 3-42
Multiply, MD (6C); Fl Pt, RX (Long) 3-93
Multiply, MDR (2C); Fl Pt, RR (Long) 3-93
Multiply, ME (7C); Fl Pt, RX (Short) 3-92
Multiply, MER (3C); Fl Pt, RR (Short) 3-91
Multiply, MP (FC); Dec, SS 3-116
Multiply, MR (1C); Fix Pt, RR 3-38
Multiply Halfword, MH (4C); Fix Pt, RX 3-42
Multiprocessing System:
BCU Modifications 4-8
BCU-Storage Operation 4-9
Configuration Control Panel 4-2, 4-6, 4-13
CPU Mode Switches 4-13
DEFEAT INTERLEAVING Switch 4-13
Direct Address Relocation 4-3
Direct Address Relocation Control 4-8
Direct Control 4-4
DISABLE DIRECT CONTROL Switch 4-13
Discussion 4-2
External Start 4-12
External Start Signal 4-6, 4-12
External System Reset 4-12
External System Reset Timing 4-10
Floating Address Control 4-7
Floating Address Switches 4-13
Floating Addressing 4-2
Functional Units 4-6
Gated Load Pb Signal 4-6, 4-11
Indicators, Roller Switch 4-14
Invalid Storage Address 4-9
I/O 4-4
I/O Allocation Control 4-7
I/O Allocation Switches 4-13
Log I/O Interrupt Signal 4-6, 4-12
Log I/O Interruption 4-12
Logout 4-15
Main Storage 4-2
Malfunction Alert 4-11
Malfunction Alert Signal 4-5, 4-11
Mode Control 4-8
Multisystem Configuration 1-10
Multisystem Feature 1-2, 4-1
Multisystem Timer 4-9
Power Distribution and Control 4-13
PREFIX Switches 4-13
Processing Units:
Discussion 4-4
Model 65 Mode 4-5
Multisystem Mode 4-4
Multisystem Signals 4-5
Partition Mode 4-5
Read Direct, RDD (85); Stat Sw, SI 4-11
Scan In 4-15
Set System Mask, SSM (80); Stat Sw, SI 4-10
STOP ON STORAGE CHECK Switch 4-14
Storage Address Decoding 4-8
Storage ADDRESS Switches 4-14
Storage Allocation 4-2
Storage Allocation Control 4-7
Storage Allocation Switches 4-14
System Call 4-11
System Call Signal 4-6, 4-11
System Control Panel, Modifications 4-14
System Hang Timing 4-10
System Reset Signal 4-6, 4-12
VALID ADDRESS Indicators 4-14
Write Direct, WRD (84); Stat Sw, SI 4-10
Multiprocessing System/360 Model 65 4-2
Multisystem Configuration:
Channel-to-Channel Adapter 1-8
Device Switching Unit 1-9
Direct Control Feature 1-10
Discussion 1-8
Multisystem Feature 1-10
Shared LCS Feature 1-10
Transmission Control Unit 1-9
2-Channel Switch Feature 1-9
Multisystem Feature (*see* Multiprocessing System)
Multisystem Mode, Multiprocessing System 4-4
Multisystem Program Interruption 1-35
Multisystem Signals:
External Start 4-6, 4-12
Gated Load Pb 4-6, 4-11
Log I/O Interrupt 4-6, 4-12
Malfunction Alert 4-5, 4-11
System Call 4-6, 4-11
System Reset 4-6, 4-12
Multisystem Timer 4-9

New PSW 1-26, 1-29
No Retry Trigger 2-90 (Table 2-2)
Normalization 1-19
Number Representation:
  Decimal 1-20
  Fixed-Point 1-16
  Floating-Point 1-17

Old PSW 1-26, 1-29
One-Cycle Tests:
  Discussion 6-54
  FLT 6-27, 6-28
  Introduction 6-52
  Result Comparison 6-55, 6-56
  Scan In 6-54, 6-55
  Scan Out 6-54, 6-56
  Terminate or Continue 6-55, 6-56
  Test Cycles 6-54, 6-56
  TN/ATN Comparison 6-55, 6-57
Op-Code Halfword Transfer 2-44, 2-50
Operand Addressing:
  Byte Selection 1-57
  Doubleword 1-56
  Effectively Addressed Operands 1-14
  Immediate Operands 1-15
  Operands in Local Storage 1-15
Operand Prefetching:
  Discussion 3-2
  Introduction 1-60
Operand Specifications:
  Decimal Divide 3-127
  Decimal Multiply 3-117
Operands:
  Effectively Addressed 1-14
  Immediate 1-15
  In Local Storage 1-15
Operating State 1-24
Operation Program Interruption 1-34
OR, O (56); Lgic, RX 3-151
OR, OC (D6); Lgic, SS 3-152
OR, OI (96); Lgic, SI 3-151
OR, OR (16); Lgic, RR 3-151
OR Function:
  Discussion 3-150
  Parity Generation 2-72
  Serial Adder Operation 2-68
Orders, I/O 1-37
Oscillator:
  Crystal-Controlled, Continuously Running 2-1, 2-2
  Gated Delay Line 2-1
Overcurrent Protection 5-9
Overriding Branch, ROS 1-47
Overvoltage Protection 5-9

P-Bit, PSW 1-25
Pack, PACK (F2); Dec, SS 3-137
Parallel Adder:
  Arithmetic Function Sequence 2-81
  Bit-Level Carry-Into Logic 2-78
  Bit Position Logic 2-73
  Carry-Into-Bit Logic 2-76
  Carry Lookahead 2-77
  Convert to Decimal Operation 2-85
  Data Flow 2-74 (Fig. 2-49)
  Data Input 2-73
  Error Checking 2-84
  Full-Sum Checking 2-84
  Full-Sum Development 2-78
  Full-Sum Logic 2-76
  Functional Breakdown 2-73
  Group-Level Carry-Into Logic 2-78
  Group-Level Carry Logic 2-77
  Half Adder 2-76
  Half-Sum Checking 2-84
  Introduction 1-68
  Latch-Shifter Logic 2-76
  Parity Predict Logic 2-82
  Section-Level Carry-Into Logic 2-78
  Section-Level Carry Logic 2-78
  Set CC 2-85
Parity Adjust Logic, IC 2-50
Parity Checking 1-56
Parity Correction, Serial Adder 2-68
Parity Generation:
  AND Function 2-72
  Exclusive-OR Function 2-72
  OR Function 2-72
Parity Predict Logic, Parallel Adder 2-82
Parity Predict Logic, Serial Adder 2-68
Parity, SAB; Conversion of 2-34
Partial Product Bit Extraction, Fixed-Point 3-39
Partial Product Formation, Fixed-Point 3-39
Partial Quotient Bit Determination, Fixed-Point 3-46
Partition Mode, Multiprocessing System 4-5
Pass Trigger, Scan Logic 6-40
Physical Layout, CPU and Main Storage 1-2 (Fig. 1-2)
Plane, CROS:
  Balance Lines 2-6
  Bit Capacitors 2-10
  Bit Plates 2-6
  Drive Lines 2-6
  Physical Package 2-10
  Sense Lines 2-9
Postnormalization 1-20
Power:
  Audible Alarm 5-12
  CE Key Switch 5-13
  CE Panel 6-21
  Check Indicators 5-12, 6-20
  Converter/Inverter 5-1
  CPU ON Pushbutton 6-21
  CPU READY/OFF Switch 6-21
  Direct Control 5-12
  Distribution:
    DC 5-4
    50-Hz 5-1
    Multiprocessing System 4-13
    60-Hz 5-1
  EMERGENCY PULL Switch 5-8
  Introduction 1-99
  MARGIN Indicators 6-20
  Marginal Checking 5-6
  Overcurrent Protection 5-9
  Overvoltage Protection 5-9
  Power-Off Sequence:
    Automatic 5-9
    Emergency 5-8
    Multisystem Feature 4-13
    Normal 5-8
  Power-On Sequence 5-6
  Regulators 5-4

System Power-On Indicator 5-11, 6-20
1052 Power Interface 5-12
1052 Printer-Keyboard 5-12
Thermal Protection 5-11
THERMAL RESET Pushbutton 6-21
THERMAL TRIP Indicators 6-21
UNDER VOLTAGE CHECK Switches 6-21
Undervoltage Protection 5-11
Usage Meters 5-13
Power Check Indicators:
Additional Storage Attachment Feature 4-15
Basic Unit 5-12
Power-On Indicator, System 5-11
Power-On Reset 6-5
Predecoding, Instruction 2-46
PREFIX Switches, Multiprocessing System 4-13
Prenormalization 1-20
Priority Logic, BCU 1-53
Privileged Operation Program Interruption 1-33
Problem State 1-23
PROC CHK Indicator 6-20
Program Interruptions:
Branching Instructions 1-94
Decimal Instructions 1-87
Discussion 1-33
Effect on I-Fetch 3-15
Fixed-Point Instructions 1-70
Floating-Point Instructions 1-81
I/O Instructions 1-99
Logical Instructions 1-91
Status Switching Instructions 1-97
Program Mask, PSW 1-26, 1-32
Program States:
Interruptable 1-24
Masked 1-24
Operating 1-24
Problem 1-23
Running 1-24
Stopped 1-24
Supervisor 1-23
Wait 1-24
Program Status Word (*see* PSW)
Program Store Compare Exceptional Condition:
Discussion 1-36
Effect on I-Fetch 3-17
Protection Key Assignments, Storage 3-179 (Fig. 3-27)
Protection Program Interruption 1-34
PSW:
Current 1-26, 1-29
Discussion 1-25
Effect on by Interruption 1-29
Main Storage Addresses 1-29
New 1-26, 1-29
Old 1-26, 1-29
PSW RESTART Pushbutton 6-16
PSW Register:
Data Flow 2-60 (Fig. 2-37)
Discussion 2-58
Introduction 1-50
PSW RESTART Pushbutton 6-16
Pulse Mode Operation 6-17
PULSE MODE Switch 6-17
Pushbuttons (*see* Specific Pushbutton or Panel)

Q-Register:
B-Field Transfer 2-44
D-Field Transfer 2-44
Data Flow 2-45 (Fig. 2-26)
Input 2-44
Introduction 1-55
Op-Code Halfword Transfer 2-44
Refilling of 1-63
Q-Register Refill Exceptional Condition:
Discussion 1-36
Effect on I-Fetch 3-22

R-Register:
Data Flow 2-47 (Fig. 2-28)
Input 2-46
Instruction Predecoding 2-46
Introduction 1-56
Output 2-46
RATE Switch 6-13
Read Direct, RDD (85); Stat Sw, SI:
Basic System 3-181
Multiprocessing System 4-5, 4-11
Read-Only Storage (*see* ROS)
Read-Only Storage Address Register (*see* ROSAR)
Read-Only Storage Backup Register (*see* ROSBR)
Read-Only Storage Data Register (*see* ROSDR)
Read-Only Storage Previous Address Register A and B (*see* ROSPARA and ROSPARB)
Regulators, High-Frequency:
List of 5-4 (Table 5-1)
Magnetic-Amplifier Controlled 5-5
SCR-Controlled Output 5-5
REPEAT INSN Switch 6-13
REPEAT ROS ADDRESS Switch 6-16
Repeat Trigger, Scan Logic 6-39
Reset:
External System, Multiprocessing System 4-12
Power-On 6-5
System 6-6
RESTART FLT I/O Pushbutton 6-19
Restart I/O Trigger, Scan Logic 6-39
Result Comparison, FLT 6-55, 6-56
Right Digit Trigger 2-89 (Table 2-2)
Ripple Tests 6-57
Roller Switches:
Basic System 6-19
Multiprocessing System 4-14
ROS:
Addressing:
Discussion 2-10
Introduction 1-42
Manually 6-9
Branching 1-42
Control of, by Scan Logic 6-40
Control of CPU 1-48
CROS 2-6
Data Flow:
Decoders 2-15
Introduction 1-45 (Fig. 1-19), 1-47
ROSDR 2-14
ROSDR Latches 2-14
Sense Latches 2-14
Timing 2-15

Decoders 2-15
Error Checking 2-18
Introduction 1-39
Logout, Control of 6-42
Maintenance Aids:
ROSAR Latches 2-16
ROSBR 2-18
ROSPARA and ROSPARB 2-16
ROSPARA and ROSPARB Alternator 2-16
Micro-Instruction (*see* ROS Word)
Micro-Orders 1-41
Relationship to Conventional Controls 1-40
REPEAT ROS ADDRESS Switch 6-16
ROS TRANSFER Pushbutton 6-15
Scan Mode Operation 2-18
Word 1-41 (Table 1-6)
ROS Test Sequencer 6-38
ROS Test State:
0 6-50
1 6-50
2 6-50
3 6-50
4 6-49
5 6-49
6 6-49
7 6-49
ROS Tests:
ATN 6-26
Bit Tests:
Discussion 6-49
ROS Test State 0 6-50
ROS Test State 1 6-50
ROS Test State 2 6-50
ROS Test State 3 6-50
ROS Test State 4 6-49
ROS Test State 5 6-49
ROS Test State 6 6-49
ROS Test State 7 6-49
Format 6-26
General Discussion 6-43
Hardcore Tests:
Discussion 6-43
Format 6-43
Introduction 6-26
IPL 2 6-45, 6-47
Record 6 6-47
Summary of 6-48
Tape Runaway 6-46
Test 1, Record 3 6-46
Test 1, Record 7 6-48
Test 1, Record 8 6-48
Test 2, Record 5 6-47
Test 2, Record 8 6-48
Test 3, Record 8 6-48
Introduction 6-26
IPL 1 6-44
IPL 2 6-45
Loader 6-45
MCW Application 6-24
Setup 6-44
TEST MODE, REPEAT Switch 6-19
TEST MODE, ROS/PROC/FLT Switch 6-18
Test Tape 6-43
TN 6-26
ROS TRANSFER Pushbutton 6-15
ROSAR:
Introduction 2-10, 2-12
Latches 2-16
ROSAR(0–5) 2-13
ROSAR(6–9) 2-13
ROSAR(10) 2-13
ROSAR(11) 2-13
ROSAR(0–10) Decoding:
Array Drivers 2-14
Select Lines 2-14
Strobed Drive Lines 2-14
ROSBR 2-18
ROSDR 2-14
ROSDR Latches 2-14
ROSPARA and ROSPARB 2-16
ROSPARA and ROSPARB Alternator 2-16
RR I-Fetch:
Basic 3-5
2-Cycle 3-23
RS I-Fetch:
Basic 3-6
2-Cycle 3-24
Running State 1-24
RX I-Fetch:
Basic 3-6
Forced-Cycle 3-24

S-Register (*see* ST Register)
S-Trigger 2-89 (Table 2-2)
SAB Parity, Conversion of 2-34
Scan/Channel Interface 6-40
Scan Clock 6-30
Scan Counter Latches and Decrementer 6-31
Scan Hardware-Generated Storage Requests 2-28
Scan In:
FLT 6-27, 6-54, 6-55
Multiprocessing System 4-14
Words 6-52
Scan Logic:
Address Sequencer 6-31
Address Sequencer Decoder 6-33
Check Counter 6-34
Control of ROS 6-40
Cycle Counter 6-37
Fail Trigger 6-40
FLT Clock 6-30
FLT Counter 6-36
FLT Test Trigger 6-40
General 6-29
Intermittent Trigger 6-40
Introduction 6-24
Logout Controls 6-38
Pass Trigger 6-40
Repeat Trigger 6-39
Restart I/O Trigger 6-39
ROS Test Sequencer 6-38
Scan/Channel Interface 6-40
Scan Clock 6-30
Scan Counter Latches and Decrementer 6-31
Scan Mode Trigger 6-39
Scan-Out Bus 6-38
Scan Out S and T Operation 6-39
Scan Stop-CPU-Clock Logic 6-39
SOROS Trigger 6-40
Storage Address Generator 6-34
Sync Trigger 6-39

Scan Mode Operation, ROS 2-18
Scan Mode Trigger 6-39
Scan-Out Bus 6-38
Scan Out, FLT 6-27, 6-54, 6-56
Scan Out ROS Trigger (*see* SOROS Trigger)
Scan Out S and T Operation 6-39
Scan Stop-CPU-Clock Logic 6-39
SCOPEX, Example C-2 (Table C-1)
SCR-Controlled-Output Regulator 5-5
Section-Level Carry-Into Logic, Parallel Adder 2-78
Section-Level Carry Logic, Parallel Adder 2-78
Select Lines, ROS 2-14
Select Signal, Generation of, by BCU 2-34
Sense Amplifiers, ROS 2-14
Sense Latches, ROS 2-14
Sense Lines, CROS 2-9
Sequencers, I-Fetch 3-12
Serial Adder:
- AND Function 2-68
- Arithmetic Functions 1-66
- Binary Addition 2-67
- Controls 2-64
- Data Flow 2-65 (Fig. 2-41)
- Data Form 2-64
- Decimal Instruction Functions 3-106
- Error Detection 2-72
- Excess-6 Arithmetic 2-67
- Exclusive-OR Function 2-68
- Input 2-64
- Logical Functions 1-68, 2-68
- Operation, Simplified 2-64
- OR Function 2-68
- Output 2-64
- Parity Correction 2-68
- Parity Predict Logic 2-68

SET IC Pushbutton 6-12
Set Program Mask, SPM (04); Stat Sw, RR 3-177
Set Storage Key 2-22, 2-41
Set Storage Key, SSK (08); Stat Sw, RR 3-178
Set System Mask, SSM (80); Stat Sw, SI:
- Basic System 3-177
- Multiprocessing System 4-10

709/7040/7044/7090/7094/7094II Compatibility Feature 1-2
7070/7074 Compatibility Feature 1-2
7080 Compatibility Feature 1-2
Shared LCS Feature 1-10
Shift Left Double, SLDA (8F); Fix Pt, RS 3-58
Shift Left Double, SLDL (8D); Lgic, RS 3-162
Shift Left Single, SLA (8B); Fix Pt, RS 3-56
Shift Left Single, SLL (89); Lgic, RS 3-162
Shift, Logical Instructions; Discussion 3-162
Shift Right Double, SRDA (8E); Fix Pt, RS 3-61
Shift Right Double, SRDL (8C); Lgic, RS 3-163
Shift Right Single, SRA (8A); Fix Pt, RS 3-60
Shift Right Single, SRL (88); Lgic, RS 3-163
SI I-Fetch:
- Basic 3-6
- 2-Cycle 3-24

Significance Program Interruption 1-34
Single-Cycle Operation 2-41
60-Hz Power Distribution 5-1
SOROS Trigger 6-40
Special Circuits A-1
Specification Program Interruption 1-34
SS I-Fetch:
- ASC Test 3-9
- Detailed Description 3-9
- General 3-7
- IC(21,22) = 00 at End Op 3-11
- IC(21,22) = 01 at End Op 3-12
- IC(21,22) = 10 at End Op 3-12
- IC(21,22) = 11 at End Op 3-12

ST Byte Counter (*see* STC)
ST Register:
- Data Flow 2-55 (Fig. 2-33)
- Input 2-54
- Introduction 1-65
- Output 2-54

ST Register Byte Counter (*see* STC)
Start I/O, SIO (9C) - I/O, SI 3-184
START Pushbutton 6-15
States, Program:
- Interruptable 1-24
- Masked 1-24
- Operating 1-24
- Problem 1-23
- Running 1-24
- Stopped 1-24
- Supervisor 1-23
- Wait 1-24, 6-16

STAT's:
- General 2-86
- Introduction 1-69
- Setting, Summary of 2-87 (Fig. 2-58)
- STAT A 2-86
- STAT B 2-86
- STAT C 2-87
- STAT D 2-88
- STAT E 2-88
- STAT F 2-88
- STAT G 2-88
- STAT H 2-89

Status Switching Instructions:
- CC 1-97
- Data Flow 1-95
- Diagnose (83); SI 3-182
- Discussion 1-94
- Formats 1-95
- Insert Storage Key, ISK (09); RR 3-179
- List of 1-96 (Table 1-13)
- Load PSW, LPSW (82); SI 3-176
- Program Interruptions 1-97
- Read Direct, RDD (85); SI 3-181
- Set Program Mask, SPM (04); RR 3-177
- Set Storage Key, SSK (08); RR 3-178
- Set System Mask, SSM (80); SI 3-177
- Supervisor Call, SVC (0A); RR 3-177
- Test and Set, TS (93); SI 3-183
- Write Direct, WRD (84); SI 3-180

Status Triggers (*see* STAT's)
STC:
- Discussion 2-57
- Introduction 1-65

Step ABC Trigger 2-89 (Table 2-2)
Stop Loop 6-5
STOP ON STORAGE CHECK Switch:
- Basic System 6-12
- Multiprocessing System 4-14

STOP Pushbutton 6-6
Stopped State 1-24
STOR CHK Indicator 6-20
STOR FRAME 1 POWER CHECK Indicator 5-12, 6-20
STOR FRAME 2 POWER CHECK Indicator 5-12, 6-20
STOR FRAME 3 POWER CHECK Indicator 5-12, 6-20
STOR FRAME 4 POWER CHECK Indicator 5-12, 6-20
Storage Address Decoding, Multiprocessing System 4-8
Storage Address Generator, Scan Logic 6-34
Storage ADDRESS Switches:
  Basic System 6-9
  Multiprocessing System 4-14
Storage Allocation, Multiprocessing System 4-2, 4-7
Storage Allocation Switches, Multiprocessing System 4-13
Storage-BCU Operation, Multiprocessing System 4-9
Storage Errors 2-38
STORAGE INDICATE Switch:
  Additional Storage Attachment Feature 6-2
  Basic Unit 6-3, 6-12
Storage Protection 1-4
Storage Protection Key Assignments 3-179 (Fig. 3-27)
Storage Requests:
  Channel 2-22, 2-28, 2-42
  CPU 2-22, 2-28, 2-40
  Priority, Establishment of 2-29
Storage-Ripple Microprogram:
  Discussion 6-16
  Display Function 6-16
  Store Function 6-16
STORAGE SELECT Switch 6-9
Storage Selection:
  BCU 2-30
  Manually 6-9
Store, ST (50); Fix Pt, RX 3-53
Store, STD (60); Fl Pt, RX (Long) 3-105
Store, STE (70); Fl Pt, RX (Short) 3-104
Store Character, STC (42); Lgic, RX 3-154
Store Halfword, STH (40); Fix Pt, RX 3-54
Store Multiple, STM (90); Fix Pt, RS 3-55
STORE Pushbutton 6-14
Store Requests 2-22, 2-40, 2-42
Subtract, S (5B); Fix Pt, RX 3-34
Subtract, SP (FB); Dec, SS:
  Complement Add Sequence 3-113
  Discussion 3-109
  GIS 3-110
  True Add Sequence 3-110
Subtract, SR (1B); Fix Pt, RR 3-33
Subtract Halfword, SH (4B); Fix Pt, RX 3-34
Subtract Logical, SL (5F); Fix Pt, RX 3-35
Subtract Logical, SLR (1F); Fix Pt, RR 3-35
Subtract Normalized, SD (6B); Fl Pt, RX (Long) 3-82
Subtract Normalized, SDR (2B); Fl Pt, RR (Long) 3-81
Subtract Normalized, SE (7B); Fl Pt, RX (Short) 3-81
Subtract Normalized, SER (3B); Fl Pt, RR (Short) 3-80
Subtract Unnormalized, SU (7F); Fl Pt, RX (Short) 3-83
Subtract Unnormalized, SUR (3F); Fl Pt, RR (Short) 3-82
Subtract Unnormalized, SW (6F); Fl Pt, RX (Long) 3-84
Subtract Unnormalized, SWR (2F); Fl Pt, RR (Long) 3-83
Supervisor Call, SVC (0A); Stat Sw, RR 3-177
Supervisor Call Interruption:
  Discussion 1-35
  Effect on I-Fetch 3-15
Supervisor Program 1-23
Supervisor State 1-23
Switches (*see* Specific Switch or Panel)
Symmetrical Clock Signal 2-2
Sync Trigger, Scan Logic 6-39
System Call, Multiprocessing System 4-11
System Call Signal, Multiprocessing System 4-6, 4-11
System Coding 1-12
System Control Panel:
  ADDRESS COMPARE STOP Switch 6-9
  ADDRESS Switches:
    Basic System 6-9
    Multiprocessing System 4-14
  CE Key Switch 6-19
  CHECK RESET Pushbutton 6-6
  CPU CHECK Switch 6-17
  DATA Switches 6-9
  DEFEAT INTERLEAVING Switch:
    Basic System 6-12
    Multiprocessing System 4-14
  DISABLE DIRECT CONTROL Switch:
    Basic System 6-17
    Multiprocessing System 4-14
  DISABLE INTERVAL TIMER Switch 6-17
  DISPLAY Pushbutton 6-15
  FREQUENCY ALTERATION Switch 6-19
  INTERRUPT Pushbutton 6-17
  LOAD Indicator 6-20
  LOAD Pushbutton 6-6
  LOG OUT Pushbutton 6-18
  MANUAL Indicator 6-19
  MARGIN Indicators 6-20
  Modifications, Multiprocessing System 4-13
  Panel A 6-2
  Panel B 6-2
  Panel C 6-2
  Panel D 6-2
  Panel E 6-2
  Panel F 6-2
  Panel G 6-4
  Power Check Indicators 6-20
  PROC CHK Indicator 6-20
  PSW RESTART Pushbutton 6-16
  PULSE MODE Switch 6-17
  RATE Switch 6-13
  REPEAT INSN Switch 6-13
  REPEAT ROS ADDRESS Switch 6-16
  RESTART FLT I/O Pushbutton 6-19
  Roller Switches 6-19
  ROS TRANSFER Pushbutton 6-15
  SET IC Pushbutton 6-12
  START Pushbutton 6-15
  STOP ON STORAGE CHECK Switch:
    Basic System 6-12
    Multiprocessing System 4-13
  STOP Pushbutton 6-6
  STOR CHK Indicator 6-20
  STORAGE INDICATE Switch 6-12
  STORAGE SELECT Switch 6-9
  STORE Pushbutton 6-14
  SYSTEM Indicator 6-19
  System Power On Indicator 6-20
  SYSTEM RESET Pushbutton 6-6
  TEST Indicator 6-20
  TEST MODE, REPEAT Switch 6-19
  TEST MODE, ROS/PROC/FLT Switch 6-18
  WAIT Indicator 6-20
System Data Flow 1-3
System Description 1-1

System Hang Timing, Multiprocessing System 4-10
SYSTEM Indicator 6-19
System Mask, PSW 1-25, 1-32
System Power Check Indicator 5-12, 6-20
System Power On Indicator 5-11, 6-20
System Reset 6-6
System Reset, External; Multiprocessing System 4-12
System Reset, Multiprocessing System 4-10
SYSTEM RESET Pushbutton 6-6
System Reset Signal, Multiprocessing System 4-6, 4-12

T-Register (*see* ST Register)
Tape Runaway, ROS Tests 6-46
1052 Adapter Feature 1-2
1052 Power Interface 5-12
1052 Printer-Keyboard, Power 5-12
Terminate or Continue, FLT 6-55, 6-56
Test and Set, TS (93); Stat Sw, SI 3-183
Test and Set Storage Request 2-23, 2-41
Test Channel, TCH (9F); I/O, SI 3-187
Test Cycles, FLT 6-54, 6-56
TEST Indicator 6-20
Test I/O, TIO (9D); I/O, SI 3-186
TEST MODE, REPEAT Switch 6-19
TEST MODE, ROS/PROC/FLT Switch 6-18
Test Number (*see* TN)
Test Under Mask, TM (91); Lgic, SI 3-154
Thermal Protection 5-11
THERMAL RESET Pushbutton 6-21
THERMAL TRIP Indicators 6-21
3-Cycle Fetch Request 2-22, 2-26, 2-40
Time Clock at Limit Trigger 2-90 (Table 2-2)
Timer Exceptional Condition:
  Discussion 1-36
  Effect on I-Fetch 3-14
Timer, Multisystem 4-9
Timing, CPU 1-39
Timing, External System Reset; Multiprocessing System 4-10
Timing Gate Trigger 2-90 (Table 2-2)
Timing, ROS Data Flow 2-15
Timing, Scan 6-29
Timing, System Hang; Multiprocessing System 4-9
TN, ROS Tests 6-26
TN/ATN Comparison, FLT 6-55, 6-57
Translate, TR (DC); Lgic, SS 3-155
Translate and Test, TRT (DD); Lgic, SS 3-156
Transmission Checks, FLT 6-54
Transmission Control Unit 1-9
Trigger, Definition 2-2
True +6 Arithmetic (*see* Excess-6 Arithmetic)
2-Channel Switch Feature 1-9
2065 Models 1-1
2361 Attachment Feature 1-2
2361 Core Storage 1-4
2365-1, -2 Processor Storage 1-4
2365-13 Processor Storage 4-2
2860 Selector Channel 1-7
2870 Multiplexer Channel 1-7

U-Bit, PSW 1-25
UNDER VOLTAGE CHECK Switches 6-21
Undervoltage Protection 5-11
Universal Instruction Set 1-13
Unpack, UNPK (F3); Dec, SS 3-139
Unsymmetrical Clock Signal 2-2
USA Standard Code for Information Interchange, Extended to 8 Bits (*see* USASCII-8)
Usage Meters:
  CE Key Switch 6-19
  Power 5-13
USASCII-8 1-12

VALID ADDRESS Indicators, Multiprocessing System 4-14

W-Bit, PSW 1-25
WAIT Indicator 6-20
Wait State 1-24, 6-16
Word Overlap:
  Decimal Instructions, General 3-107
  Move With Offset, MVO (F1); Dec, SS 3-142
  Pack, PACK (F2); Dec, SS 3-139
  Unpack, UNPK (F3); Dec, SS 3-141
World Trade Differences B-1
Write Direct, WRD (84); Stat Sw, SI:
  Basic System 3-180
  Multiprocessing System 4-4, 4-10

X-Branch, ROS 1-43

Y-Branch, ROS 1-43

Z-Branch, ROS 1-43
Zero and Add, ZAP (F8); Dec, SS 3-115
Zero Cycle Tests:
  Discussion 6-54
  Introduction 6-52
  Result Comparison 6-55, 6-56
  Scan In 6-54, 6-55
  Scan Out 6-54, 6-56
  Terminate or Continue 6-55, 6-56
  Test Cycles 6-54, 6-56
  TN/ATN Comparison 6-55, 6-57

www.IBM360.info