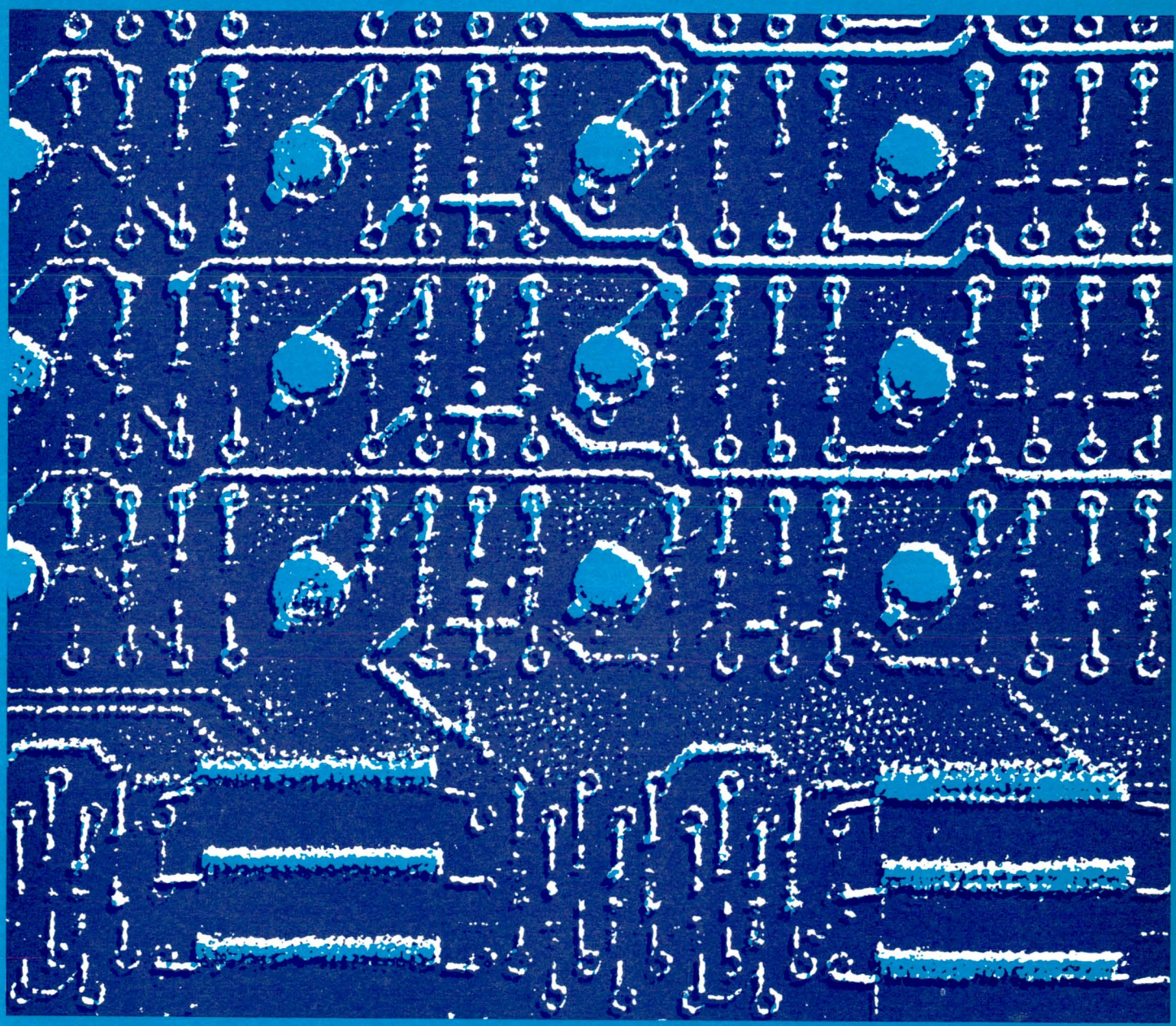
P800M Programmer's Guide 1

Volume IV: BRTM



PHILIPS Data Systems

PHILIPS

P800M Programmer's Guide 1
Vol. IV: BRTM

A Publication of

Philips Data Systems B.V.
Marketing Group Small Computers
Apeldoorn, The Netherlands

Publication number 5122 991 27342

June 1976

Copyright © by Philips Data Systems B.V., 1976
All rights strictly reserved. Reproduction or issue to third parties in any form whatever is not permitted without written authority from the publisher.

Printed in The Netherlands.

Preface

This is volume IV of a six-volume set dealing with the Basic Operating System (non-real time and real time) for the P800M series. It describes the Basic Real Time Monitor.

The other volumes of this set are:

I: Basic Operating Monitor
II: Instruction Set
III: Software Processors
V: Small Real Time Monitor
VI: Cassette Operating System

Other books pertaining to the P800M series are:

P852M System Handbook
P856M/P857M Handbook
P800M Operator's Guide
P800M Interface and Installation Manual
P800M Software Reference Data

Great care has been taken to ensure that the information contained in this manual is accurate and complete. However, should and errors or omissions be discovered, or should any user wish to make a suggestion for improving this manual, he is invited to send his comments, written on the sheet provided at the end of this book, to:

Manual Writing Small Computers

at the address on the opposite page.

PART 2: bRTM CONFIGURATION

PART 1 MONITOR USE

1 Principles of operation

The Basic Real Time Monitor has been designed to supervise the execution of pretested programs in a real time environment, on the basis of a priority system consisting of up to 48 hardware interrupt levels and 14 software priority levels.
Since there are no special provisions for program development (i.e. background) it is assumed that all programs used under the BRTM have been debugged and pretested.

The priority system incorporates hardware interrupt lines and software priority levels:

- 0 up to 47 are levels for hardware interrupt lines
- 48 and 49 are priority levels for the monitor
- 50 to 63 are software priority levels for user programs.

The highest priority level is 0, so hardware interrupts will always overrule software level programs.
The interrupt routines which service the internal and external hardware interrupts, are connected to the levels 0 to 47.
Some of the interrupt routines are standard; the user can easily include interrupt routines written by himself. Incoming interrupts are handled by hardware and receive control on the basis of the priority level to which they have been assigned. The dispatcher, a monitor module on level 48, divides central processor time between competing priorities, i.e. it allots C.P.U. time to the user programs. The highest level active program always gets control until it is interrupted. Registers are saved by hardware in a system stack (addressed by register A15) and by software in the same stack or in a save area, which the ~~Initial Program Loader~~ reserves in front of each user program. The hardware saves only P-register and PSW.
It is possible for user programs and subroutines, to obtain temporary memory space dynamically in a dynamic allocation area. This is very important for re-entrant subroutines, as they may not contain any work areas. The dynamic area is formed by the remaining part of memory behind the user programs. By means of special requests, a program can ask for an area of memory and, after use, deallocate it again for use by other programs.
To operate the system, the monitor and all necessary user programs must be loaded and then started. If new programs are to be added later, everything must be reloaded.
Loading is done by special program, the ~~Initial Program Loader (IPL)~~, which in turn is loaded by a general program loader, the IPL. It is

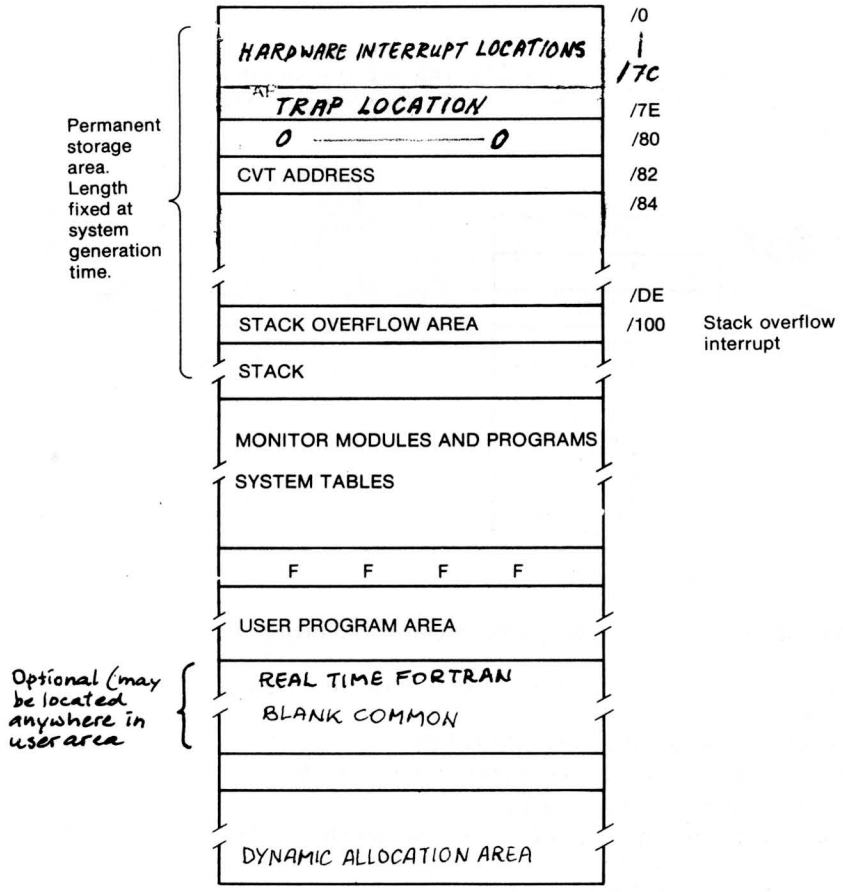
possible to activate and start programs at loading time, but this can also be done later on by program.
All I/O operations for user programs are initiated by monitor requests, i.e. LKM instructions followed by a DATA directive with a number as operand. Several I/O functions can be performed, such as binary, ASCII and object code I/O, with conversion and control facilities, if applicable. Peripheral devices are referenced through file codes, logical numbers of 2 hexadecimal digits.
Monitor requests are also used to ask the monitor to perform a variety of other functions:

- program activation
- waiting for the occurrence of an event
- program exit
- obtaining and releasing buffer space in the dynamic allocation area
- connecting programs to a priority level and disconnecting them
- connecting programs to timers and disconnecting them
- switching from one program to another inside a level to allow timeslicing, i.e. allotting processor time to several programs on one level by turns, on the basis of timer interrupts
- getting the time and date
- informing the monitor of the occurrence of an event
- directing the monitor to wait with activation of a program until a certain, specified time
- reserving a preripheral for exclusive use by one particular program; detaching that device from the program.

A special feature, enhancing the multiprogramming aspect of the system, is the scheduled label, which is used in conjunction with monitor requests.
A scheduled label is a subroutine which is started when the monitor request function has been completed, although it is specified at the same time as the monitor request, i.e. the main program can continue while the requested function is performed. For example, combined with an I/O request, the main program continues while the requested I/O function is performed and the branch to the scheduled label is not made until that operation has been terminated. This can be very useful, for example to analyze the results of a monitor request.

2 Memory Organization

The memory layout is as follows:

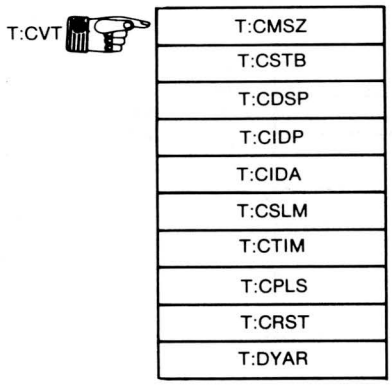


Locations /0 to /7C are *hardware interrupt locations.* They are hard-wired to internal and external interrupt lines. Each location contains the address of the interrupt routine required to service the interrupt connected to that location. The interrupt connected to location /0 has the highest priority (level 0). See chapter 4.

Location /7E contains the address of the trapping routine which handles simulation of certain instructions not included in the hardware (e.g. double add, double subtract, multiply, divide, multiple store, multiple load and double shift).

Location /80 contains 0.

Location /82 points to the Communication Vector Table. This is a system table which contains information which may be of use to the user program. The table has the following layout:



where:

T:CMSZ contains the machine memory size in characters. If the memory size is 32k words, the value is 0.

T:CSTB contains the system stack base address as defined at system generation time.

T:CDSP contains the address of the dispatcher (M:DISP).

T:CIDP contains the Program Status Word (PSW) of the Idle Task.

T:CIDA contains the start address of the Idle Task.

T:CSLM contains the maximum number of scheduled labels to be recorded in a table at any one time.

T:CTIM contains the address of the Timer Block.

T:CPLS contains the value of the timer pulse (PR): 1 for the 50Hz standard clock.

T:CRST contains the non-standard clock reset value.

T:DYAR contains the start address of the dynamic allocation area (see below).

Locations /84 to /DF are used for other system tables.

The area occupied by the *Stack* is defined at system generation time. When an interrupt occurs, P-register, PSW are stored here by hardware and a number of registers by software. The number of registers stored depends on whether the interrupt routine servicing the interrupt runs in inhibit mode (anywhere from 0 to 15 registers) or in enable mode or branches to the dispatcher (always 8 registers). The A15 register always points to the next free location in the stack (where all information is stored towards the **lower** memory addresses). When A15 reaches the value /100 or becomes lower, a stack overflow interrupt is given.

The area which remains after the user *program area,* is reserved for *dynamic memory allocation* after termination of all loading procedures. From this area, blocks of memory space can be requested by the system or by the user. The user must send a 'Get Buffer' monitor request for this purpose. When he does no longer need the buffer, he must send a 'Release Buffer' monitor request.

3 Interrupt System

Programs and routines under the BRTM run on the basis of an interrupt and priority system which consists of up to 64 levels. These levels are subdivided as follows:

0 - 47:	levels for interrupt routines connected to the hardware interrupt lines	hardware interrupt lines
48 :	interruptable monitor service routines	software priority levels
49 :	high priority system programs	
50 - 63:	user programs	
64 :	idle task (simulated level)	

Level 0 has the highest priority , 63 the lowest, so all hardware interrupts always have priority over the software levels.

HARDWARE INTERRUPT LINES

The interrupt lines are connected to memory locations /0 to /7C. These locations contain the addresses of the interrupt routines which service the internal and external interrupts.
For the interrupts the user can define the priority levels at system generation time.

The following priorities are strongly recommended for the various interrupt lines:

```
 0:   power failure                - (interrupt location /00)
 1:   LKM / stack overflow         - (interrupt location /02)
 2:   real time clock              - (interrupt location /04)
 3:   not used
 4:   punched tape reader                    etc.
 5:   tape punch
 6:   operator's typewriter
 7:   control panel
 8 to /F:  free
/10:  X1215 disc
/11:        disc
/12:  disc
/13:  magnetic tape
/14:  cassette tape
/15:  card reader
/16:
/17:  line printer
/18 to /1F:  free
```

T, with the other interruptable monitor service routines

DISPATCHER

The dispatcher is a monitor module (M:DISP) running on level 48 which divides central processor time by starting programs according to their priority. The dispatcher can be entered only from an interrupt routine, i.e. from a level below 48, such as the I/O interrupt or monitor request handlers.

or equal to

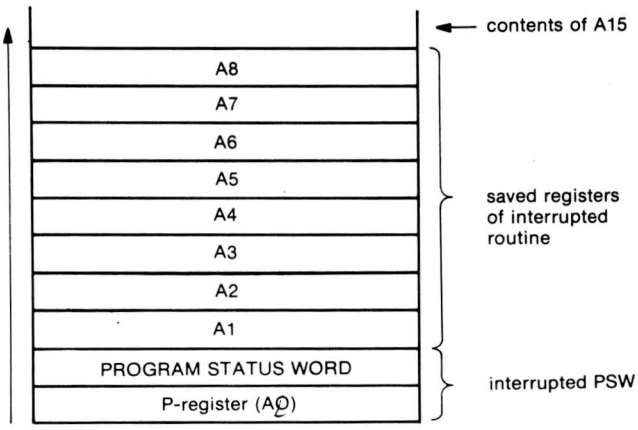
SOFTWARE PRIORITY LEVELS

Under the BRTM user programs may be connected to any of the levels 50 to 63, according to the priority which the user attaches to them. Here programs are activated by monitor requests or at monitor loading time. It is possible in this way for one program to activate another one
By means of the monitor request Switch Inside a Level it is possible to divide processor time among several programs on one level, on the basis of real time clock interrupts.
Requests for program activation are handled by the 'Activate' monitor module, then passed on to the dispatcher.

STACK

When an interrupt occurs, certain information about the interrupted program or routine must be saved before the interrupt can be serviced.
For user programs, this is done in the system stack pointed to by register A15 or, when there is a priority change from one program to another, in a 16-word save area reserved in front of each program, where P-register, PSW and registers A1 to A14 are stored.
For interrupt routines it is done in the system stack.
The start address of this stack is defined at system generation time and it is built in a downward direction in memory, i.e. towards the lower addresses. The A15 register always points to the first free location in the stack.
Upon interrupt, PSW and P-register are always saved in this stack and moreover a number of registers:

- any number if the interrupt routine runs in inhibit mode and takes care of restoring the registers itself;
- register A1 to A8 if the interrupt routine runs in enable mode or ends with a branch to the dispatcher, because the dispatcher always handles the stack on this basis:

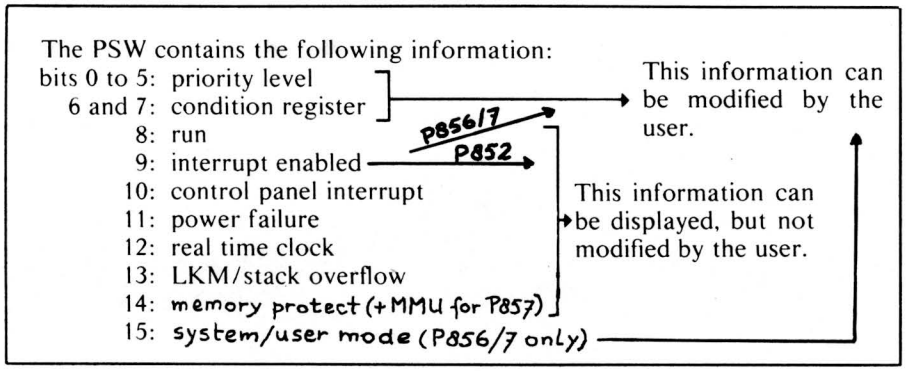


10

When the stack pointer (A15) reaches the value /100, the last location before the stadk overflow area, an interrupt will be given.

or when its value is below /100

The Program Status Word (PSW), stored in the stack upon interrupt, contains the following information:



The PSW can be displayed on the control panel.

4 Programs

There are three types of user-written programs:
- programs (user-written codings sequences), connected to any of the **software** levels 50 to 63
- subroutines, re-entrant or non-re-entrant; they are called through a CF (Call Function) instruction and run on the level of the calling program.
- interrupt routines, servicing one of the 63 possible hardware interrupts and connected to any of the **hardware** levels 0 to 47.

SOFTWARE LEVEL PROGRAMS

These are user programs connected to any of the software priority levels 50 to 63.

The programs required for a real time process may be used by one or more other programs. They are loaded into memory as separate programs and the connections between the programs which are related to each other are supplied by monitor requests in the individual programs.
A program is connected to a level either at loading time (system message 'L':, where the user may specify a level) or by a monitor request (**Connect a Program to a Level** – LKM 20). The level is registered in the PSW (bits 10-15). Program **activation** is also possible in two ways:

- by the Initial Program Loader (IPL), at loading time.
- by an **'Activate'** monitor request (LKM12) from another program.

After activation, the dispatcher will start the programs according to their priority.
Besides being connected to a level, programs can also be connected to a timer, so that they may be activated at a certain time or after regular intervals, according to parameters which the user specifies in a monitor request **'Connect a Program to a Timer'** (LKM10).
It is also possible to share central processor time between various levels and between programs of the same level by time-slicing, i.e. the monitor request **'Switch Inside a Software Level'** (LKM13). See below. The *scheduled label* is another feature which extends the possibilities for real-time programming. See below.

After a program has been loaded, all relevant information about it is stored in a Program Control Table in the PCT Pool *in* the monitor area in memory.
The program remains under monitor control until it is disconnected from its level, during which time it passes through various states, as recorded in the PCT:
- inactive: the program has been connected to a level, but it has not been called yet;
- active: the program has been called and it is not yet terminated;
- wait for execution: the program is ready to use central processor time when it has the current highest priority;
- wait for an event: the program has given up control voluntarily with a *Wait for an Event* monitor request (LKM2) to wait for the occurrence of a particular event.

Every Software Level Program is preceded by a 16-word save area reserved by INIMON. If any scheduled labels are used in the program, the user must reserve himself an additional save area, for interrupts occurring while the program is in a scheduled label sequence, to store the contents of its P-register, PSW and registers A1 to A14. The user must reserve 16 + (N + 1) words, where N is a system generation parameter specifying the maximum number of scheduled labels which may be queued in this table at any one time (see below):

```
        IDENT     PRNAME
        RES       16+ (N+1)   N+1=FILLAB table length
START       Coding
        LKM
        DATA      3
        END
```

SCHEDULED LABELS

The scheduled label is a feature which allows the user to do a sort of multi-tasking by attaching a routine to a monitor request.
To this end, the user specifies the monitor request as having the two's complement of the DATA number indicating the monitor function which is to be executed, followed by the label of the routine which must be executed upon completion of the request. For example:

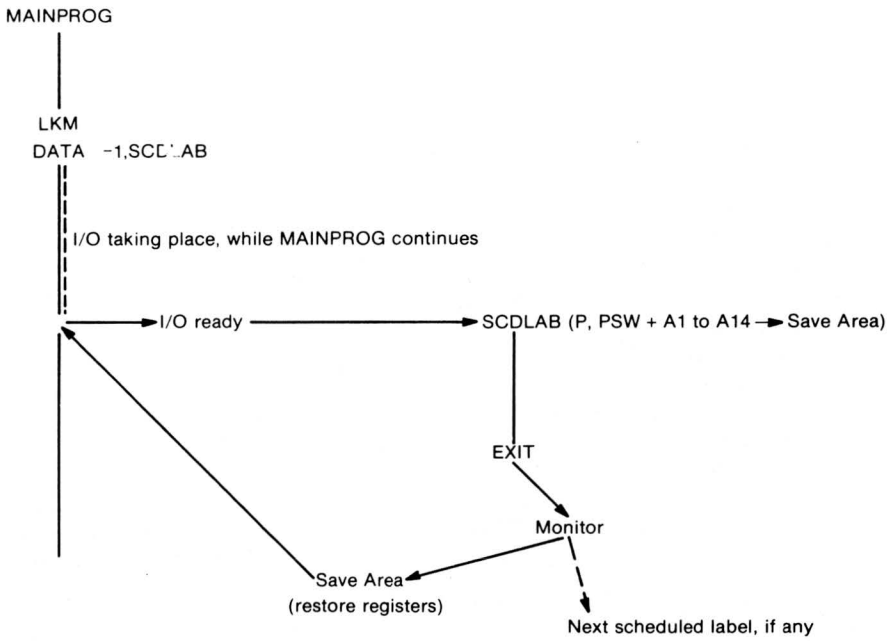
normal I/O request:

```
        LDK     A7,CODE
        LDKL    A8,ECBADR
        LKM
        DATA    1
```

with sched. label:

```
        LDK     A7,CODE
        LDKL    A8,ECBADR
        LKM
        DATA    -1,SCDLAB
```

In this case, the routine SCDLAB is to obtain control on completion of the I/O monitor request specified. When a scheduled label is attached to an I/O request, one should not set the wait bit, so that the program can continue concurrently with the I/O operation requested. When that operation is finished, control is passed to the SCDLAB routine, and the P-register, PSW and registers A1 to A14 of the main program are stored in the 16-word save area in front of the user program.
When entering the SCDLAB routine, no register value is significant, so it is not possible to pass parameters to the scheduled label sequence. When the routine is finished and exits, control is returned to the monitor, which passes control to a possible following scheduled label routine, or back to the main program by restoring the registers and PSW from the save area. This can be illustrated as follows:



Any number of scheduled labels may be given in a program.
However, it is possible that one scheduled label is blocking execution of another one, because it is active, i.e. is using central processor time. In such cases the address(es) of the queued scheduled label routines are temporarily stored in a table (FILLAB). The maximum number of scheduled label addresses which may be stored in this table at any one time is defined at system generation time.
Queued scheduled labels are treated on a first-in-first-out basis.

Note: Although it is possible to give a Wait monitor request within a scheduled label routine, this is not normally recommended, for it blocks the whole system.

INTERRUPT ROUTINES

User interrupt routines must
have been connected to one of the interrupt locations in memory (locations /0 to /7e. That is, the address of the interrupt routine must be placed in one of these locations, which at the same time determines the priority level of the interrupt routine, i.e. the interrupt routine whose address is loaded in location /0 has the highest priority (0).
The interrupt routine address may be loaded into this location in several ways. One of them is to link-edit and include the routine with the rest of the system modules at system generation time. Another method, providing a relocatable routine, is shown in the following example:
A routine, labeled INTRO is to be connected to interrupt level 10, so the

starting address of the routine has to be loaded into memory location /20

```
        IDENT   INTRO
          :
        LDKL    A1,ROUT
        ST      A1,/20
          :
ROUT      :     }     INTERRUPT ROUTINE
          :
        END
```

When an interrupt occurs, the P-register and PSW of the interrupted program are stored by hardware in the system stack pointed to by the A15 register (see Ch.3) and the system is put in inhibit mode.
Then the interrupt routine receives control and from within the routine the user may store any other registers by software, if he wishes. The interrupt routine may now continue in inhibit mode, or if the user decides that other interrupts must be able to overrule the current one, he may set the system to enable mode by giving an ENB instruction. This, however, entails a substantial difference in the handling of the system stack. If the routine runs in inhibit mode from beginning to end, any of the registers A1 and A14 can be used, provided the user first takes care of storing old contents in the A15 stack and restoring them at the end of the routine. This may, for example, be done as follows:

```
STR   A1,A15
STR   A2,A15
  :     :
STR   A8,A15
   coding
LDR   A8,A15
LDR   A7,A15
RTN   A15
```

For P856/7 or for P852 when the simulation routines have been selected at sysgen:

```
MSR   8,A15
        |
      coding
        |
MLR   8,A15
RTN   A15
```

This is the case of an interrupt routine in inhibit mode with a normal return via the A15 stack. The RTN via A15 results in an automatic enable.
However, if other interrupts are to be enabled during a routine or the user makes an absolute branch from the interrupt routine to the dispatcher (ABL M:DISP; for dispatcher address: see CVT), he must take care that before the ENB or ABL instruction is given, the A15 stack contains only P, PSW and **registers A1 to A8** inclusive, because on this basis the A15 stack is handled.

Conventions

- Interrupt routines must start by saving the old contents of the registers to be used in the routine
- Before returning via A15, the old register contents must be restored

  If a branch is made to the dispatcher, the stack must contain P, PSW and registers A1 to A8, so any other registers used, must have been restored before making the branch.
- In case of an interrupt routine for internal interrupts
  LKM/stack overflow real time clock, power failure, control panel), an RIT instruction must be given at the beginning of the routine, to reset the interrupt. See System Software Manual.
- Of monitor requests, only Activate (LKM12) should be used in interrupt routines. The others, if used, may be executed at the level of the interrupt, thus possibly producing unforeseen program delay.

(Note: If an INH instruction immediately follows the ENB instruction, a dummy instruction must be inserted, because external interrupts are scanned every two instruction. This dummy instruction may, for example, be another ENB, so that then the correct sequence becomes: ENB - ENB - INH)

DYNAMIC MEMORY ALLOCATION

The user can make temporary use of the dynamic allocation area of memory if his monitor contains the module with the two monitor requests:

- Get Buffer, to reserve a user buffer in the dynamic allocation area;
- Release Buffer, to release the area created by the Get Buffer request.

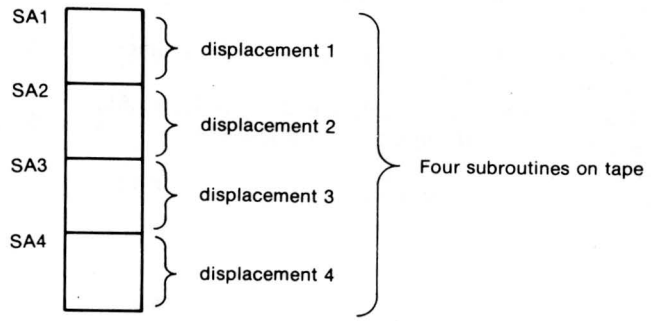
Memory allocation is in blocks, the block length being specified by the user. Preceding each block is a control block containing links and other parameters *which must not be destroyed.*
For the syntax of these requests, refer to the section Monitor Requests.

REENTRANT SUBROUTINES

Re-entrant subroutines are subroutines which can be used by more than one program. They can be interrupted and restarted for a higher priority program at any time.
These subroutines must be loaded into memory in such a way, that the programs which will use them, can find their start addresses.

Direct addressing is not possible, it is recommended to use the registers for this purpose.
When the subroutines are loaded, a dummy program may also be loaded, defining the entry points of the subroutines and their addresses. This dummy program must be linked with each program which is going to use any or one of these subroutines.

Example



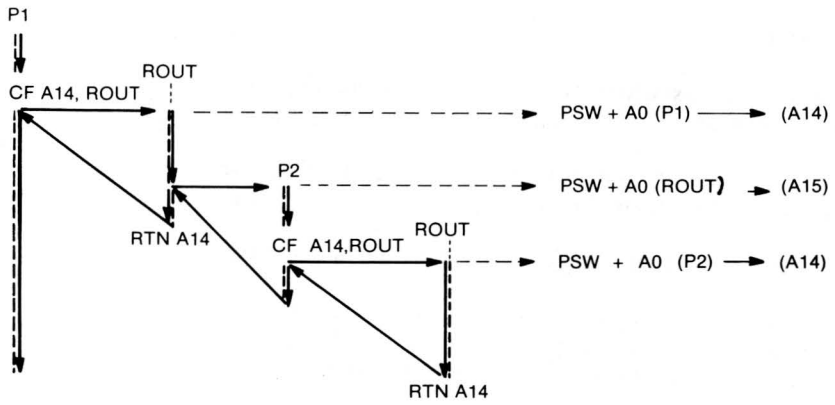
Dummy program:

```
        IDENT   DUPR
        ENTRY   SA1
        ENTRY   SA2
        ENTRY   SA3
        ENTRY   SA4
SA1     EQU     /2500
SA2     EQU     SA1+'displacement 1'
SA3     EQU     SA2+'displacement 2'
SA4     EQU     SA3+'displacement 3'
        END
```

This dummy program must be linked with each program using any of the four subroutines. The programs, for example, are as follows:

```
IDENT   PROG
  |
EXTRN   SA1
EXTRN   SA4
  |
  CF    A14,SA4
  |
  CF    A14,SA1
  |
 END
```

If a re-entrant subroutine wishes to call another re-entrant subroutine, it must reserve two words in the dynamic allocation area through the *Get Buffer* monitor request. The subroutine obtains any other memory space it requires, in the same manner.
Re-entrant subroutines cannot use scheduled label routines.
If a subroutine is interrupted for another program of a higher level, requiring the same subroutine, we get the following situation:



- Program P1 starts running.
- CF is given, PSW + P of P1 are stored in stack, of which address is given in A14 and subroutine ROUT is started.
- Interrupt for program P2, PSW and P of ROUT are stored in stack specified in A15 and P2 starts running.
- CF is given, PSW + P of P2 are added to the stack specified in A14 and subroutine ROUT is started and allowed to terminate with RTN A14.
- P2 is resumed and allowed to terminate normally.
- Via A15 subroutine ROUT is resumed and allowed to terminate with RTN A14.
- P1 is resumed.

Note: the stack indicated by A14 has to be reserved by the user; the stack indicated by A15 is reserved and handled by the monitor.

TIME-SLICING

For software level programs time-slicing is possible, i.e. allotting equal amounts of processor time to several programs connected to one level, under control of a timer and by turns. This functions as follows:
Let us assume that level 52 is connected to a timer, e.g. the real time clock, and that a program on level 52 contains the monitor request which activates the switching procedure:

Real Time Clock → 52 →

```
Switching program:
LDK A7, 55
LKM
DATA 13
                This calling sequence
                requests switching
                on level 55.
LKM
DATA 3          'Exit' monitor request
                to go to level 55.
```

Level 55 has three programs connected to it: A, B and C.
At an interrupt of the real time clock (every 20 milliseconds), the program connected to level 52 performs the switching request and exits. The dispatcher then activates one of the programs, A, B, or C on level 55, to which ever one it was pointing. This program then starts running:

- until an interrupt starts an interrupt routine (level 0 to 47) or until it is required to start level 53 or 54, e.g.
  because an I/O operation for which one of them was waiting is ready.
- until 20 milliseconds have passed. Then the real time clock gives another interrupt and control is returned to the program on level 52. Another request is performed, level 52 exits and, barring higher priority interrupts, the next program on level 55 is started, to run until another 20msecs. have passed or a higher priority interrupt occurs.

This sequence continues until the programs A, B and C have had enough processor time to be executed.
If, for any reason it is desired to limit the interrupts during switching as much as possible, the switching program can be connected to level 50 and the programs to be switched to level 51. In this case, only hardware interrupts can interfere.

Note: This request is also used by some of the monitor modules, i.e. those which handle the monitor requests Activate, Exit, Detach Device, Release Buffer and Input/Output.

5 Input/Output

All I/O operations are initiated by an I/O monitor request. At system generation time, the necessary modules and tables for fulfilling this request must have been filled. When the request is given, with an LKM instruction, register A7 must have been loaded with parameters about the particular type of I/O function, while register A8 must contain the address of an Event Control Block which holds the necessary information about the data to be transferred.

There are three types of I/O request (as specified in A7):

Basic I/O requests: for these requests the monitor will not do any character checking or data conversion, so they are used in case of binary I/O. The monitor handles only the control command initialization and signals the end of the I/O operation.

Standard ASCII I/O requests: these requests provide more monitor facilities, such as error control characters, data conversion from external code to internal ASCII code and vice versa, character checking for end of data. Characters are stored 8 by 8 bits, two to a word.

Standard Object I/O requests: by means of standard conventions facilities are provided for error control characters, checksum and data conversion from external 4+4+4+4 or 8+8 punched tape format to internal 16-bit format.

Moreover, a number of control functions can be performed through a monitor request, such as writing EOS or EOF records, skipping forward or backwards, rewinding, etc.
In the Event Control Block, pointed to by register A8, the user specifies the file code (logical address) of the device concerned with the I/O operation, and additional parameters such as buffer address and buffer length. At the end of the I/O operation, the monitor places information concerning the result of the operation in this ECB, so that it may be verified by the user program.

The **file codes** are a means of giving a logical address to a file or a peripheral unit. They are defined at system generation time and consist of 2 hexadecimal digits from /01 to /FF. The following file codes are standard:

/01:	Source Input
/02:	Listing Output
/03:	Punch Output
/04:	Object Input
/05:	Operator's Typewriter (input and output)

The following are **recommended** file codes:

/06: ASR tape reader
/07: ASR tape punch
/08: High-speed tape reader
/09: High-speed tape punch
/0A: Line printer
/0B: Card reader
/0C: Magnetic tape cassette
/0D: Magnetic tape cassette
/0E: Magnetic tape cassette
/0F: Magnetic tape cassette

These can also be re-assigned by the user, i.e. it is possible to re-assign file code 08 to, for example, the card reader.

The remaining file codes, up to /FF can be assigned by the user to any other files or devices.

The file codes are contained in a system table: the File Code Table (FCT), containing the addresses of the Device Work Tables (DWT's) of the peripheral devices. The place in the table determines the file code, i.e. the first address in the table gets file code /01, the second /02, and so on.

According to the above-mentioned user parameters, the I/O operations are handled by a varying number of monitor modules, including drivers.
Chapter 7 gives the necessary details for the I/O monitor requests (LKM1).

6 Monitor Requests

The user program can request the monitor to execute certain functions. A request takes the form of a LKM (Link to Monitor) instruction followed by a DATA directive.
The directive has a number as operand which specifies the function to be executed. If this number is negative, the user is scheduling a label on completion of the request.
Preceding a request, certain parameters may need to be loaded into the A7 and A8 registers.
After the monitor has processed the request it loads a return code in the A7 register. If the requested module is not available, A7 is always −1. The following requests are available:

Monitor Request	Register A7	Register A8	LKM DATA	page
I/O Requests	Order	ECB Address	1	1-24
Wait for an Event	—	ECB Address	2	1-28
Exit	—	—	3	1-29
Get Buffer	Buffer Length	—	4	1-30
Release Buffer	—	(A14: Buffer Address)	5	1-31
Connect Program to Timer	Prog. Name Address	Param. Block Address	10	1-32
Disconnect Program from Timer	Prog. Name Address	Timer Number	11	1-33
Activate Program	Prog. Name Address	ECB Address	12	1-34
Switch Inside a Level	Level	—	13	1-35
Attach Device	Wait Flag	Device Block Address	14	1-36
Detach Device	—	Device Block Address	15	1-37
Get Time	Binary Flag	Timer Block Address	17	1-38
Set Event	—	Event Block Address	18	1-39
Connect Program to Level	Level	Prog. Name Address	20	1-40
Disconnect Program from Level	Level	Prog. Name Address	21	1-41
Wait for a given Time	—	ECB Address	22	1-42

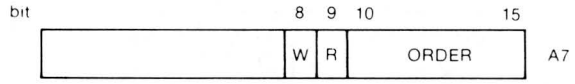
Note: It is possible to attach a scheduled label to a Monitor Request. For details, see Chapter 4.

I/O REQUEST

Calling sequence:

```
LDK   A7,CODE
LDKL  A8,ECBADR
LKM
DATA  1
```

Use:
The user can ask the system to start a particular I/O operation on a peripheral device.
Register A7 is loaded with a CODE that specifies the details of the I/O function, as follows:



W and R specify the mode of operation:

W = 1: Return to the calling program will be made after completion of the I/O operation.

W = 0: Return to the calling program will be made as soon as the transfer has been initiated. The program will give a Wait Request later, for synchronization.

R = 1: Any abnormal conditions will be processed by the user program. The system will return the Hardware Status in ECB word 4 (see under ECB). No retry is possible. Possible with Basic Read/Write only.

R = 0: Any abnormal conditions will be processed by the system and the Software Status will be returned in ECB word 4 (see under ECB).

ORDER consists of a hexadecimal value on six bits, identifying the I/O function:

01: Basic Read

05: Basic Write
For Basic I/O requests the system does not provide for character checking or data conversion, only for control command initialization and end of operation signals.

02: Standard Read

06: Standard Write
Standard (ASCII) I/O requests provide, by means of standard conversions, for special features such as error control characters, conversion from external code to internal ASCII and vice versa.

08: Object Write −8+8 format. A word is contained within 2 rows, 8 bits per row.
Object I/O requests provide, by means of standard conversions, for special features such as error control characters, checksum and data conversion from external 4+4+4+4 or 8+8 tape format to internal format.

30: Get information about a file code

Tape handling orders:

16: Skip forward to next EOF mark
22: Write EOF mark
24: Write EOV mark
26: Write EOS mark
31: Rewind to load point
33: Backspace one block

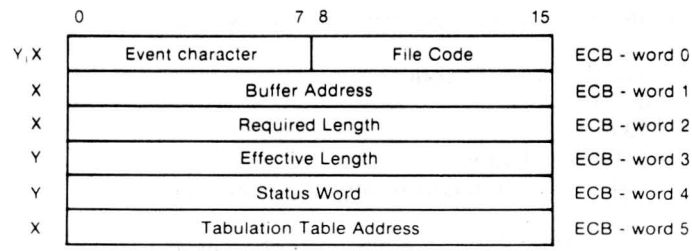
36: Skip backwards to EOF mark

38: Unlock

Refer to Appendix A for more information specific to the various peripheral devices.

ECBADR is the address of the Event Control Block containing the parameters for the I/O function:

Event Control Block
The ECB address must be loaded in the A8 register. ECB structure:



where: The words marked X must be filled by the user; those marked Y will be filled by the system.

word 0: event character:
bit 0 = 1: end of operation has occurred for this ECB.
file code: must be loaded before requesting the I/O. See Appendix C.

word 1: address of the user buffer.

word 2: requested length to be read or written (in words for basic read on card reader, in characters for other devices). The first character is always the character given by the buffer address.
For standard write on typewriter and line printer, two characters must be added, at the beginning of the buffer.

word 3: effective length which has been transmitted (in words for basic read on card reader, in characters for other devices). This information is stored here by the monitor upon completion of the I/O operation.

Note: If magnetic tape or cassette tape handling orders are given in this monitor request (/16 and /31 to /38), it may be useful to make a separate ECB for these orders, as in these cases only ECB word 0 is important (file code). The contents of ECB1 and 2 are then irrelevant, *but must be unequal to zero.*

word 4: Status word, stored by the monitor upon completion of the requested I/O operation.

- Request accepted: for Basic orders this word will be filled with the hardware status (bit 0 = 1). (See Appendix D). For Standard orders the status will be:
  Status 8000: no operation (after RD message) < bit 7=1: no data on CRC casseH
  positive: bit 8 = 1: End-Of-Volume } mag. or cass. tape
  9 = 1: End-Of-Tape }
  10 = 1: beginning of tape.
  11 = 1: end of input medium.
  12 = 1: incorrect length requested or checksum error.
  13 = 1: illegal character code
  14 = 1: EOS
  15 = 1: EOF
  zero: normal I/O completion.
- Request rejected: for Basic or Standard orders the status will be:
  status C000 and: bit 11 = 1: order unknown or not compatible with the device.
  12 = 1: illegal buffer size.
  13 = 1: illegal buffer address.
  14 = 1: device attached to another program.
  15 = 1: illegal file code.

word 5: for a Standard read on punched tape equipment or typewriter, the user may fill this word with the tabulation table address. This tabulation table gives the format in which the user wishes the printout to appear.

The table has the following format:

Number of Tackets	First Tacket
Second Tacket	Third Tacket

etc.

The tackets have an absolute position in the line. Characters up to the following tacket are filled with blanks.

Note: If word 5 is filled, the required length in word 2 must contain both the characters *and* the blanks in the tackets.

Example:

3	10
20	30

Input Line: LABEL\OPER\OPERAND\COMMENT

Line in Buffer:
LABEL␣␣␣␣OPER␣␣␣␣␣␣OPERAND␣␣␣COMMENT

1 10 20 30

At completion of input, the buffer is filled with spaces, but the returned length is the length effectively entered and stored, including the spaces replacing the tabulation code (\).

Upon completion of one of these I/O functions, the system responds as follows:
A7 > 0: request completed.
= −1: the corresponding request module is not in memory.

WAIT FOR AN EVENT REQUEST

Calling Sequence

```
LDKL    A8, ECBADR
LKM
DATA    2
```

where ECBADR gives the address of the Event Control Block (see I/O requests). The first character of the ECB is the event character. If bit 0 of this character is set to 1, the event has been completed.

Use

This request causes a program to stop and wait for the completion of an event which must take place in the same or another program (user or system). If the event has occurred, the dispatcher returns control to the requesting program. If the event has not occurred, the program is put in wait state, to be restarted when the event has occurred.

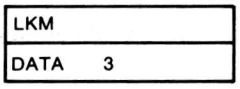
- Wait for the exit of a program:
  When a user program activates another program (see Activate), the first word pointed to by the A8 register is the address of the ECB which must be used to wait for the exit of the activated program. As the activated program and the calling program run concurrently, this provides a means of communication between the two programs. This synchronization must be done by the user program. See Exit request
- Wait for an I/O operation:
  The program waits for the end of an I/O operation it has requested.

Notes:

1. It is recommended not to use a Wait request inside a scheduled label routine, as this causes the whole program to be blocked temporarily.
2. Under BRTM, the user can create his own set of events. In this case, he must reset the event bit in the ECB and inform the system by giving a 'Set Event' monitor request.

EXIT REQUEST

Calling Sequence



Use

This request is used to indicate the end of a user program or scheduled label routine. The program exit is effected after completion of all I/O operations and after all labels have been scheduled.

A scheduled label routine exit passes control to the next scheduled label routine, if one is present, otherwise control passes to the main program.

The program becomes inactive, unless there is another activation request waiting for this program.

The program remains connected to its level. If it is connected to a shared level (Switch monitor request) a switch is implicit.

The ECB supplied at the activation of the program must be updated by giving a 'Set Event' monitor request (LKM 18), to set bit 0 of the event word to 1.

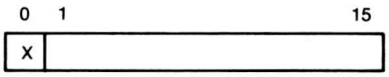
GET BUFFER REQUEST

Calling Sequence

```
LDK     A7,LENGTH
LKM
DATA    4
```

where:

LENGTH is the length, in characters, to be allocated to the buffer area (maximum 32k characters):

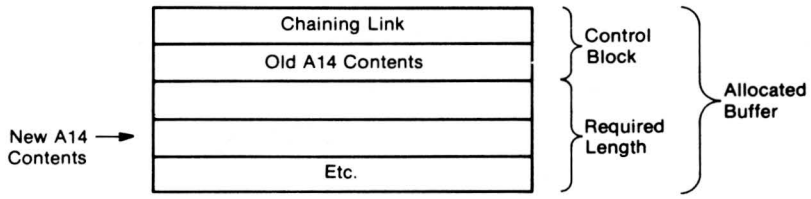


If bit 0 = 0: return to user in case of overflow.
If bit 0 = 1: implicit wait of the calling program in case of overflow.

If 0 is loaded into A7, the monitor will return the memory upper address in A7. If the memory size is 32k, 0 will be returned in A7.

Use

By means of this request, the user can allocate a memory area for temporary use, in the dynamic memory allocation area, behind the user program area.
When the allocation is made, a control block is created by the system at the beginning of the allocated area. This block will contain a chaining link and the old contents of the A14 register:



The user must not destroy this control block.
Upon completion of the request, the system responds as follows:

A7 = 0: the buffer is allocated
= 1: there is no memory space available (bit 0 in LENGTH = 0)
A14: contains the address of the fourth word of the allocated buffer, so that, as soon as the buffer is allocated, the user may give a Call Function instruction with the A14 register without having to update the A14 register first.
However, the user must then provide for stack handling.

RELEASE BUFFER REQUEST

Calling Sequence

```
LDKL    A14,BUFADR
LKM
DATA    5
```

where:

BUFADR points to the second word in the buffer as given in register A14 after the Get Buffer request.

Use
To release the memory space previously reserved by a Get Buffer request. The A14 register is reloaded with the value it contained before the Get Buffer request was made.
The system responds as follows:

A7 = 0: the memory space is released.

If the A14 pointer is incorrect, or if the buffer area has been destroyed, the system issues a Halt.
If the dynamic area was in overflow state, this request frees it again, and programs waiting because of buffer overflow are restarted, and their requests reinitialized.

CONNECT A PROGRAM TO A TIMER

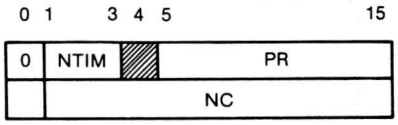
Calling Sequence

```
LDKL A8, PARAM
LDKL A7, PRNAME
LKM
DATA 10
```

where:

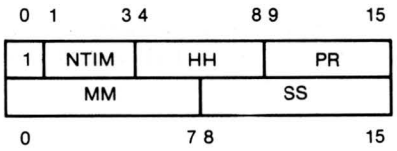
PARAM points to a two-word block, containing the necessary parameters. Two formats are possible for this block:

— format 1 (bit 0 = 0):



NTIM: the number of the timer (0 = real time clock). Timer numbering: see next page.
PR: requested pulse rate, i.e. the number of cycles of the timer between two activations: a number from 0 to 2047. If 0 is specified, the program is activated only once and then **automatically** disconnected.
NC: number of cycles of the timer before the first activation: a number from 0 to 32767.

— format 2 (bit 0 = 1):



where NC is replaced by an absolute time: HH (hour) — MM (minute) — SS (second), and PR is a number from 0 to 127.

Use

A program running at a software level can be connected to a timer, according to the parameters given in register A8.

The program must have been connected to a software level, otherwise it cannot be started by the dispatcher.
The system responds as follows:
A7 = 0: the connection has been made.
A7 ≠ 0: connection is impossible, because:
— the specified timer has not been defined: A7 = -2
— the program does not exist: A7 = -3
— dynamic area overflow: A7 = —4.

DISCONNECT A PROGRAM FROM A TIMER

Calling Sequence

```
LDK A8, NTIM
LDKL A7, PRNAME
LKM
DATA 11
```

where:

NTIM is a constant specifying the timer number.
PRNAME points to a 3-word block containing the program name.

Use

A program can, by means of this request, be disconnected from the timer specified in register A8.
The program remains connected to its software level.
The system responds as follows:

A7 = 0: the program is disconnected from the timer.
A7 = −3: it is impossible to disconnect the program, because the program does not exist, or was not connected to this timer.

Timer Numbering:

- with declaration of the 20 msec. clock:
  0: clock timer (20 msecs.)
  1: tenths of seconds
  2: seconds
  3: minutes
  4: hours
- with declaration of a non-standard clock:
  0: non-standard clock
  1: 20 milliseconds
  2: tenths of seconds
  3: seconds
  4: minutes
  5: hours

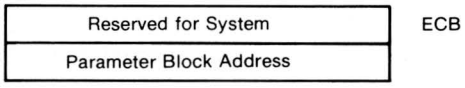
ACTIVATE A PROGRAM

Calling Sequence

```
LDKL A7, PRNAME
LDKL A8, BLOCK
LKM
DATA 12
```

where:

PRNAME points to a 3-word block containing the name of the program to be activated.
BLOCK points to a 2-word block of the following format:



The first word of this block may be updated by means of a 'Set Event' monitor request (LKM18) at the exit of the activated program, so that, if the calling program has requested a wait for the activated program, this word may be considered its Event Control Block (see Wait Request). The second word contains the address of a parameter block. The calling program may give a Get Buffer Request to build this block in the Buffer pool, before the Activate request is given. Register A4 will then contain the address of the parameter block when the activated program is started by the dispatcher and register A14 will contain zero. That is, A4 contains the value of ECB+2 and A8 points to the ECB for a possible Set Event request.

Note: When a user program is activated by an interrupt sequence, there is no parameter block and the contents of A4 is not significant. A14 is set to zero.

Use

This request can be made by a program running at any level, to activate a software level program.
Calling program and activated program may be processed concurrently, depending on their priority levels.
If the activated program is busy, the request is recorded in a stack, to be processed later.
The activated program must previously have been connected to a level. The system responds as follows:

A7 = 0: the request has been processed or recorded in a stack;
A7 ≠ 0: the request has not been taken into account, because:
- the program has not been connected to a level: A7 = −2;
- the program does not exist: A7 = −3.
- dynamic area overflow: A7 = −4.

SWITCH INSIDE A SOFTWARE LEVEL

Calling Sequence

LDK A7, LEVEL
LKM
DATA 13

where:

LEVEL is a constant specifying the number of the level in which the switch is to be made. If LEVEL is specified as 0, the level to be switched is equal to the requesting level plus one.

Use

By means of this request it is possible to have timeslicing inside a software level, by halting execution of the program running on that level (A7) and giving control to the next program on the same level. See also page 1-18.
In three cases a switch is implicit, i.e. this request is not necessary:

- when an Activate request is given, a switch is done on the level of the activated program, to find another program which can then be started.
- when a Wait or Exit request is given, a switch is done on the level of the program which is waiting or exits, so that an attempt can be made to give control to another program on the same level.
- for a Set an Event request, a switch is done on all levels where an event must be reset at that time.

The system response is:
A7=0: the level has been switched (if there is more than one PCT connected to this level.
A7=-2: the level is not connected:
- the level is not between 49 and 64.
- there is only one PCT connected to this level.

ATTACH A DEVICE TO A PROGRAM

Calling Sequence

```
LDK A7, FLAG
LDKL A8, DEVBLK
LKM
DATA 14
```

where:

FLAG is either zero or not zero. When it is zero, control is returned to the calling program with −3 in A7, if the device was already attached to another program.
When it is zero, the calling program is put in wait until the device is detached.
DEVBLK points to a one-word block containing:

```
FC
```

where:

FC is the file code assigned to the device concerned.

Use

By means of this request, a program running at any software level can obtain exclusive use of the device specified. Other programs will be unable to perform I/O operations on this device.
If the device has already been attached to another program, the requesting program is put in wait state until the device is detached by the other program (see following request).
This is checked via word 34 in the D.W.T. where bit 0 is 1 if the device is **not** attached. If it is attached, the wait is done on this bit.
In case several programs have given an attach request and are waiting for a device to be detached, the highest level program will receive control first.
The system responds as follows:

A7 = 0: the device has been attached;
A7 = −2: the device does not exist, or it is impossible to attach it to this program.

Note: The ASR is considered as 3 devices, so 3 requests must be given to attach the ASR keyboard, tape reader and tape punch.

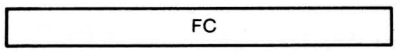
DETACH A DEVICE FROM A PROGRAM

Calling Sequence

```
LDKL A8, DEVBLK
LKM
DATA 15
```

where:

DEVBLK points to a one-word block of the following format:



where:

FC is the file code of the device concerned.

Use

By means of this request, a device which has been attached to a program by an Attach request (LKM 14), is detached from that program.
To prevent another program having to wait unnecessarily for this device, the Detach request must be made as soon as the device is no longer required.
The system responds as follows:

A7 = 0:	the device has been detached;
A7 = −2:	the device does not exist, or it is attached to another program.

GET TIME

Calling Sequence

```
LDK A7, FLAG
LDKL A8, TIMBLK
LKM
DATA 17
```

where:

FLAG is a value of either 0 or 1, to specify whether the time will be output in ASCII (0) or binary (1) format.
TIMBLK points to a six-word block into which the monitor will load the requested information.

Use
Upon receipt of this request, the monitor will return the time to the block specified in register A8, in the format indicated by the flag in register A7.
If A7 = 0, the information will be in the following format:

DAY	word 0
MONTH	1
YEAR	2
HOUR	3
MINUTE	4
SECOND	5

The values are given in ASCII.

If A7 = 1, the information will be in the following format:

HOUR	word 0
MINUTE	1
SECOND	2
Tenth of Second	3
Fiftieth of Second	4
C:TIME	5

The values are given in binary.
C: TIME is 0, if the standard clock is included. If a non-standard clock is included, it contains the pulse rate of this clock.

SET AN EVENT

Calling Sequence

```
LDKL A8, ECBADR
LKM
DATA 18
```

where:

ECBADR is the address of the ECB whose event bit must be reset, because the event for which it was waiting, has occurred.

Use

The user can create his own set of events. Each time such an event has occurred, the user must inform the system about it by giving this request, so that it can restart any programs which were waiting for this event. The system sets the event bit to 1.

For example, the system sends this request after monitor requests for I/O (twice: for ECB and for controller status), Release Buffer, Detach Device and Exit.

CONNECT A PROGRAM TO A SOFTWARE LEVEL

Calling Sequence

```
LDK A7, NUMBER
LDKL A8, PRNAME
LKM
DATA 20
```

where:

NUMBER is the number of the software level to which the program is to be connected.
PRNAME points to a 3-word block containing the name of the program.

Use

A program, running at any level can make this request for another program to be connected to a software level.
The system responds as follows:

A7 = 0: the connection has been accomplished.
A7 ≠ 0: it is impossible to make the connection, because:
- the program does not exist: A7 = −3;
- the program is not compatible with the requested level, i.e. it is an interrupt level: A7 = −3;
- the program has already been connected to a level: A7 = −2.

DISCONNECT A PROGRAM FROM A LEVEL

Calling Sequence

```
LDK A7, NUMBER
LDKL A8, PRNAME
LKM
DATA 21
```

where:

NUMBER is the number of the software level to which the program has been connected.
PRNAME points to a 3-word block containing the name of the program which is to be disconnected.

Use

The program specified is disconnected from the level given in register A7.
The system responds as follows:

A7 = 0: the program is disconnected.
A7 ≠ 0: it is impossible to disconnect the program, because:
- the program does not exist: A7 = −3;
- the program is busy: A7 = −2;

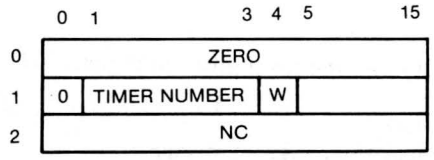
WAIT FOR A GIVEN TIME

Calling Sequence

```
LDKL A8, PARAM
LKM
DATA 22
```

where:

PARAM is the address of a 3-word block of the following format:



where word 0 contains zero, as the ECB on which the calling program will be put in wait state, must be equal to zero.

word 1: Bit 0 = 0
Bits 1-3: timer number of the timer for which the program waits. (see under LKM 10)
Bit 4: WGT flag: any value; destroyed by the system.
Bits 5-15: not significant.

word 2: Number of cycles, of the timer defined in word 1, to be made before the program is restarted.

Use

This request is given to put a program in wait state, until a certain time has passed, as indicated in word 2 of the PARAM block. The precision of the waiting time depends on the precision of the timer defined in word 1 of this block.

When the number of cycles specified has passed, the requesting program is restarted with the system response in A7: 0 = request performed correctly. If the response in A7 is −2, the wrong timer has been defined.

If it is −4, there has been dynamic area overflow.

7 System messages

System messages are printed on the operator's typewriter by the system. For systems without operator communication, no messages are printed. When abort and exit conditions arise, the P-register halts at specific addresses.

Abort
format: ABORT␣ <code> ␣ <address>
meaning: an error has been encoutered and the program is aborted.
<code>: may be:
01: simulation routine save area overflow.
02: illegal instruction.
04: buffer area destroyed or block was bigger than 32k.
05: label could not be scheduled.
06: operator abort.
<address>: the address at which the abort occurred.

Erroneous Cluster
format: EC
meaning: an erroneous cluster has been encountered on the tape. Program loading stops.

End of File
format: :EOF
meaning: the loader has read the "end-of-file" mark and stops loading.

End of Segment
format: :EOS␣ <address>
meaning: the loader has read the "end-of-segment" mark.
<address>: address of the first free location after the program.

Error in an Operator Control Message
format: ER
meaning: the operator message contains an error. The operator must press the INT button and retype the message.

Program Termination
format: EXIT
meaning: This message is printed when execution of the user program is completed.

Program Identification
format: IDENT␣ < prog. id. > ␣ < address >
meaning: the loader has read the program identification.
< prog. id. > : program name.
< address > : first address of the loaded program.

No Start Address
format: NS
meaning: No start address was specified in the END/START cluster (record type 7. See Appendix B). Program cannot be executed.

Overflow
format: OVL
meaning: Insufficient memory available. Program loading stops.

Peripheral Device Error
format: PU, < device name and address >, < hardware status > [RY]
meaning: a failure has been detected on a peripheral device. RY is printed if a retry is possible.
If the operator releases the operation on the device, the system will consider the I/O operation completed and control will be returned to the user.
< hardware status > : see Appendix D.

Input/Output Error
format: I/O ERROR XXXX YYYY
meaning: An error has occured during an I/O operation, e.g. throughput error, data fault, in connection with an MC operator command.
XXXX is device name and device address.
YYYY is the hardware status of the control unit.

8 Operation

The initial loading procedure is as follows:
the bootstrap is loaded, either through the toggle switches or by pushing the IPL button on the control panel, then the Initial Program Loader (IPL) is loaded into memory, followed by the monitor.

LOADING BOOTSTRAP AND IPL

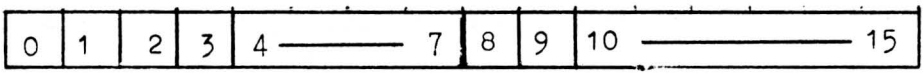
The bootstrap can be loaded in one of two ways, depending on whether the optional ROM bootstrap is included in the system or not:

If not, the procedure is as follows:
- switch on the CPU
- load the bootstrap into the first 64 memory locations manually, by means of the toggle switches and the Load Memory button on the control panel. The 64 bootstrap values can be found in Appendix E. Then check by reading these locations out.
- set up the device parameters on the toggle switches, as shown below, and load this value into the A15 register.
- push the MC button.
- put the IPL tape on the tape reader and make the reader operable.
- push the RUN button on the CPU control panel.
  Now the IPL is read and loaded into memory.

If the ROM bootstrap is included in the CPU, which is highly recommended, the operation is much simpler:

- switch on the CPU.
- put the IPL tape on the tape reader and make it operable.
- set up the device parameters on the toggle switches, as shown below.
- push the IPL button on the control panel. This loads the bootstrap into memory, which immediately starts reading and loading the IPL from the tape reader.

The device parameters on the data switches must be set as follows:



bit 0: =0: tape format used is 8+8
=1: tape format used is 4x4
bit 1: =0: the device used is not a disc
=1: the device used is a disc
bit 2: used only in case bit 1 = 1:
=1: the disc used is a moving head disc
=0: the disc used is a fixed head disc
bit 3: =0: the device is connected to the multiplex
=1: the device is connected to the programmed channel
bits 4 to 7: are used to qualify the CIO start command sent by the bootstrap and are transferred to the addressed control unit on lines BIO 12 to 15 when required
bit 8: =0: the control unit involved is a single device control unit
=1: the control unit involved is a multiple device control unit
bit 9: =1: the disc used is an X1215 disc (P824-001)
bits 10 to 15 contain the device address.

Examples:
9130: ASR tape reader, format 4x4, address 30
1020: punched tape reader, format 8+8, address 20
0244: magnetic tape unit, address 04.
0795: cassette tape unit, address 15

LOADING OBJECT TAPES

The Initial Program Loader (IPL) which has now been loaded into memory, is actually divided into two parts. The bootstrap, which was located at addresses 0 to /7F, loads the first part (80 characters) into memory from location /80 and transfer control to this IPL part with start address /84. This IPL part then computes the memory size and loads the rest of the IPL into the high end of memory, where it receives control from the first IPL part.
Now object tapes such as the monitor can be loaded and the bootstrap and first IPL part can be overwritten by them:

- when the IPL has been loaded it outputs the message
  OBJECT TAPE ON READER. THINK OF BASE!
  (If it is an IPL which is not on a separate tape but on the tape in front of the object program, e.g. in front of the Assembler or Stand Alone Linkage Editor, this message is not output, but the IPL carries on loading the rest of the tape immediately after it has itself been loaded.)
- now put the monitor tape on the tape reader, ensuring that the reader is operable. If other object programs than the monitor are loaded the user has, at this point, the opportunity of changing the base address of the object program which is to be loaded. This address is contained in register A9 and can be modified via the CPU control panel when the CPU is stopped. If this register is not modified, it is assumed to contain the value 0, as in the case of monitor loading.
- push the RUN button.
  This restarts the IPL which loads and starts the monitor.
- the following message is output
  IDENT XXXXXX
  identifying the loaded object tape. When loading is finished
  :EOS
  :EOF
  are typed out.
- the monitor is now ready for use and if the INT button on the control panel is pushed, it will type out
  M:
  to ask for a command. Note that the INT button must not be pushed if the OCOM module (operator communication) has not been included at SYSGEN for this will result in abort.

Loading the BRTM

The IPL is used to load the BRTM NUcleus (resident monitor) and the initialization program INIMON which overwrites the IPL and is erased at the end of initialization. IPL is on the system tape in front of the Nucleus and INIMON. This tape must be placed on the reader and the user must push the RUN button to load and start it. After initialization the space occupied by INIMON is released for dynamic memory allocation.

At the end of this loading process, ▽ asks for a number of messages, which it requires to initialize the Nucleus and be able to load the user programs. These messages are always output. If an option, to which the message refers, is not included in the system, the operator must type LF CR.

▽INIMON

Clock Initialization (if option included)

INIMON types: **SD:** (date)

Operator answers: **DD␣MM␣YY** LF CR

giving the day, month and year he wishes to initialize.

INIMON types: **SC:** (time of the day)

Operator answers: **HH␣MM␣SS** LF CR

giving the hour, minute and second he wishes to initialize

If the operator does not want to initialize any or one of these parameters, he must simply type **LF CR.**

Any syntax errors in the parameters typed by the operator cause *INIMON* to repeat the question, so that a new, correct answer can be typed in.

Note: the actual machine clock will not be started until *INIMON* has terminated. This should be taken into account when entering the time parameter.

Note: The control panel key should not be in the position ON/RTC during the INIMON process, but should be switched to this position only after the ST message (see below).

Loading User Programs

After the initialization phase *INIMON* enters the loading phase and types:

I:

The possible operator answers are:

– WM␣ <address> ␣ <value 1> [␣ <value 2> . . . ␣ <value n>]
This message is used if the operator wishes to alter the contents of one or more memory locations.
<address> is the first (or only) location to be altered, given as a hexadecimal number of up to 4 digits.
<value 1> is the value to be loaded into the location given by <address>
<value 2> is loaded into the location <address + 2> etc.

– LD [LF] [CR]

In this case the program is loaded from a tape and *INIMON* outputs program name and start address. When the program has been loaded, *INIMON* asks for the level to which the program must be connected:

L:

The operator types in

XX [LF] [CR]

where XX is a two-digit hexadecimal number specifying the level. If the operator does not yet wish to connect the program to a level he must type only [LF] [CR.]

After the level specification, *INIMON* outputs another message:

A:

asking whether the program loaded must be activated immediately when control is given to the BRTM Nucleus. If the operator wants immediate activation, he types:

A [LF] [CR]

If he wants the program activated later on, he just types:

[LF] [CR]

At this point *INIMON* returns to the beginning of the loading phase loop and again types:

I:

The operator may load a new program, repeating the sequence above, or he may specify the end of the loading phase:

–ST [LF] [CR]

is typed in by the operator to indicate the end of the loading phase and to request *INIMON* to transfer control to the BRTM Nucleus.
The parameters concerning the dynamic memory allocation area are set in the Nucleus tables.
At this point the control panel key may be switched to ON/RTC to start the clock, if it is included.
Control is given to the dispatcher so that the highest-priority active program can be started.

Errors

— If overflow occurs during loading, because there is not enough memory space available, *INIMON* types out the message:

OVL

— If an input error occurs during loading, *INIMON* types out:

EC

— If an overflow occurs because there is not enough memory space available in the PCT Pool to create a new entry, *INIMON* outputs the message:

OVT

— If the user wants to cancel a line, he can type a vertical arrow (↑). The line will then be skipped up to the next carriage return (CR).

Loading User Programs
After the initialization phase *INIMON* enters the loading phase and types:

I:

The possible operator answers are:

– WM␣ <address> ␣ <value 1> [␣ <value 2> . . . ␣ <value n>]
This message is used if the operator wishes to alter the contents of one or more memory locations.
<address> is the first (or only) location to be altered, given as a hexadecimal number of up to 4 digits.
<value 1> is the value to be loaded into the location given by <address>
<value 2> is loaded into the location <address + 2> etc.

– LD [LF] [CR]

In this case the program is loaded from a tape and *INIMON* outputs program name and start address. When the program has been loaded, *INIMON* asks for the level to which the program must be connected:

L:

The operator types in

XX [LF] [CR]

where XX is a two-digit hexadecimal number specifying the level. If the operator does not yet wish to connect the program to a level he must type only [LF] [CR.]

After the level specification, *INIMON* outputs another message:

A:

asking whether the program loaded must be activated immediately when control is given to the BRTM Nucleus. If the operator wants immediate activation, he types:

A [LF] [CR]

If he wants the program activated later on, he just types:

[LF] [CR]

At this point *INIMON* returns to the beginning of the loading phase loop and again types:

I:

The operator may load a new program, repeating the sequence above, or he may specify the end of the loading phase:

–ST [LF] [CR]

is typed in by the operator to indicate the end of the loading phase and to request *INIMON* to transfer control to the BRTM Nucleus.
The parameters concerning the dynamic memory allocation area are set in the Nucleus tables.
At this point the control panel key may be switched to ON/RTC to start the clock, if it is included.
Control is given to the dispatcher so that the highest-priority active program can be started.

Errors

— If overflow occurs during loading, because there is not enough memory space available, *INIMON* types out the message:

OVL

— If an input error occurs during loading, *INIMON* types out:

EC

— If an overflow occurs because there is not enough memory space available in the PCT Pool to create a new entry, *INIMON* outputs the message:

OVT

— If the user wants to cancel a line, he can type a vertical arrow (↑). The line will then be skipped up to the next carriage return (CR).

APPENDICES

Appendix A Peripheral input/output

This appendix refers to the I/O order which the user must specify in register A7, when he gives an I/O monitor request (LKM1).

BASIC READ (/01)

Operator's Typewriter:
All characters are entered on 8 bits until the requested length is reached.

ASR Tape Reader:
All characters are entered on 8 bits. The reader stops one character after an Xoff code has been read.

High-speed Tape Reader:
All characters are entered on 8 bits, without checking or special features, until the requested length is reached.

Card Reader:
All the words are entered and stored in Hollerith code on 12 bits (4 to 15). In each word the column image is right-justified. The words are stored until the requested length is reached. The length is given in words.

Magnetic Tape Cassette:
All Read/Write operations (Basic, Standard, Object) are the same, with the following characteristics:
- maximum record length: 256 characters.
- required length: block length.
- effective length: block length (without control character).
- all read/write operations are done on the requested length
- incorrect length after read operation: no error, if requested length is greater than block length and the returned status is correct.
- throughput error or data fault: retry is made automatically, up to five times.
  - after read : backspace - read
  - after write : backspace - erase - write.

Magnetic Tape:
Same as for cassette tape, with the following differences:
- maximum record length: 4095 characters; minimum 12 characters.
- required length: block length (2 dummy characters must be reserved behind the buffer).
- physical block length: required length + 2.
- effective length: block length.
- 12 characters are always transferred, in any case.
- incorrect length: see cassette tape above.

BASIC WRITE (/05)

Operator's Typewriter:
All characters are output without checking or special features. This order can be used to print something and have the answer on the same line.

ASR Tape Punch:
All characters are output without checking or special features.

Line Printer:
All characters are output without checking. There is no control character.

High-speed Tape Punch:
All characters are output without checking or special features.

Cassette and Magnetic Tape:
See under Basic Read (/01).

STANDARD WRITE (/06)

Operator's Typewriter:
All characters, except /0 to /1F (special code characters), are output without checking. At the end of a line, a carriage return and line feed are output. The first word in the buffer contains a control character (second character), as for the line printer (see below). If it equals /30 or /31, it is output as line feed; if it is different, it is not output.

ASR Tape Punch:
Same features as for the keyboard. At the end of a line, the following character sequence is output: LF - Xoff - CR - Rubout

High-speed Tape Punch:
Same as for ASR tape punch.

Line Printer:
All characters are output without checking, except for the control code. It must be stored in the right-hand character of the first word of the buffer. This control code may have one of the following three values:

+ (/2B): print the line without advancing the paper (superposition).
0 (/30): advance two lines before printing.
1 (/31): skip to top of page before printing.

All other control codes are used as normally: advance one line and print. At the end of the buffer, after the requested length, one character must follow to be used by the system for a print code.
If the requested length is more than one line, the system puts a print code after the maximum length and the buffer will be printed on two or more lines.

Cassette and Magnetic Tape:
See under Basic Read (/01).

OBJECT WRITE 4+4+4+4 TAPE FORMAT (/07)

ASR Tape Punch:
The first character is output on one row, converted from /0 - /7 to /18 - 1F. Each following character is output on two rows; to avoid special (ASCII) code each row is converted. The second character contains the length of the block in words, excluding first character and checksum. At the end an 8-bit checksum is performed and punched, followed by an Xoff code.

High-speed Tape Punch:
Same as for ASR tape punch.

OBJECT WRITE 8+8 TAPE FORMAT (/08)

High-speed Tape Punch:

The standard object code is output in 8+8 format, where the first character is a format character and is output on one row, converted as follows:

/0 – /10
/1 to /4 – /01 to /04
/5 to /7 – /15 to /17

The second character contains the length in words, excluding the first word and checksum. An 8-bit checksum is performed and punched.

Cassette and Magnetic Tape:
See under Basic Read (/01).

READ UP TO END-OF-SEGMENT (/14)

High-speed Tape Reader:
The tape is read until an:EOS statement has been read.

Card Reader:
The cards are read until an:EOS statement has been read.

READ UP TO END-OF-FILE (/16)

High-speed Tape Reader:
The tape is read until an :EOF statement has been read.

Card Reader:
The cards are read until an :EOF statement has been read.

WRITE EOF MARK (/22)

Operator's Typewriter:
An end-of-file mark is output as follows: :EOF LF Xoff CR Rub-out

ASR Tape Punch:
An end-of-file mark is output as follows: :EOF LF Xoff CR Rub-out

High-speed Tape Punch:
An end-of-file mark is output as follows: :EOF LF Xoff CR Rub-out

Line Printer:
An end-of-file mark is output as follows: :EOF

WRITE EOS MARK (/26)

Operator's Typewriter:
An end-of-segment mark is output as follows: :EOS LF Xoff CR Rub-out

ASR Tape Punch:
An end-of-segment mark is output as follows: :EOS LF Xoff CR Rub-out

High-speed Tape Punch:
An end-of-segment mark is output as follows: :EOS LF Xoff CR Rub-out

Line Printer:
An end-of-segment mark is output as follows: :EOS

Magnetic Tape Cassette:
An end-of-segment mark is written as /6F6F.

Magnetic Tape:
An end-of-segment mark is written as :EOS + 8 blank characters.

WRITE EOV (/24)

End-of-tape management for Magnetic and Cassette tapes is a user program responsibility.
When the physical end of a tape is encountered during a write operation, a status is returned in ECB word 4 with the EOT bit set.
The user may then issue a Write EOV request (/24; Write End-Of-Volume; see under I/O monitor requests), before requesting the operator to mount a new tape. When a new tape is mounted, for magnetic tape the unit must first be switched off by pressing the OFF LINE button, while for cassette tape a Manual Control (MC) operator command 'Unlock' must be given to enable the operator to remove the cassette.
Then the operator can mount a new tape reel or cassette and restart the program.
To ensure that all records will be retrieved when this file is read, the EOT (end-of-tape) status also returned in the status word of the ECB should be ingnored and only the EOV status must be taken into account.

Note: In case the EOT is detected when reading an EOV, only the EOV status is returned.

RETURN INFORMATION ABOUT A FILE CODE (/30)

By means of this order it is possible to find out the assignment of a file code. The information will be returned in the Event Control Block:

ECB - word 0:	File Code.
ECB - word 1:	Device Name (2 ASCII characters): TY = operator's typewriter (listing) TR = ASR tape reader TP = ASR tape punch PR = tape reader PP = tape punch LP = line printer CR = card reader MT = magnetic tape TK = cassette tape
ECB - word 2:	maximum record size.
ECB - word 3:	left character: unused. right character: device address.
ECB - word 4:	status=0. For line printer, this word contains the number of lines specified for this printer at system generation time.

If the file code in ECB word 0 is set to zero, the other words of the ECB will also contain zeros. This means no device has been assigned to the file code.

OFF LINE (/38)

Magnetic Tape:
This order switches the tape drive off-line.

Cassette Tape:
This order unlocks the cassette from the drive unit.

Appendix B Object code record types

Type 0: Block data record
1: Entry point names record
2: External reference names record
3: Code words record
4: Internal modification record
5: Entry point definition record
6: Common length definition record
7: End/Start record.

Appendix C File codes and device names

File codes enable files to be identified and addressed. The file codes are assigned to devices at SYSGEN. The device addresses are fixed by hardware. The size of the file code table is also established at SYSGEN so that dummy file codes should be included if future expansion is envisaged.

Standard File Codes	**File**
01	Source input
02	Listing output
03	Punch output
04	Object input
05	Operator's typewriter (input and output)

Device Names	**Device**
TR	ASR tape reader
TP	ASR tape punch
PR	High speed tape reader
PP	High speed tape punch
TY	Operator's typewriter
CR	Card reader
LP	Line printer
TK	Magnetic tape cassette
MT	Magnetic tape
NO	No device: an operation on this file will have no effect.

Appendix D Control unit status word configuration

This is the hardware status as returned in ECB word 4. See I/O monitor request.

Bit	Description	Control Unit						
		ASR	CR	LP	PTP	PTR	TK	MT
0								
1	has become ready						×	×
2	rewinding							×
3	tape mark read						×	×
4	no data						X	
5	Load point beginning of tape						X	X
6	write unable						×	×
7	A or B side (A=1, B=0)						×	
8	device address						×	×
9	device address						×	×
10	EOT tape low				×	×	×	×
11	program error						×	×
12	incorrect length		×				×	×
13	parity error						×	×
14	throughput error	×	×			×	×	×
15	not operable (only significant bit for TST)	×	×	×	×	×	×	×

```
00000                    IDENT    BEBOOT
00001          *
00002          *
00003          *         DISPLAY THE KEYS AS FOLLOWS:
00004          *
00005          *         BITS     MEANING
00006          *         0=1      IPL LOADED FROM ASR , FORMAT 4*4
00007          *         1=1      DISK,THEN
00008          *         2=1      MOVING HEADS
00009          *         2=0      FIXED HEADS
00010          *         3=1      PROGRAMMED CHANNEL
00011          *         3=0      I/O PROCESSUR
00012          *         4 TO 7   BOU LINES ( 4 RIGHTMOST BITS )
00013          *         8=1      MULTI DEVICE CONTROLLER
00014          *         8=0      SINGLE DEVICE CONTROLLER
00015          *         9=1      X1215 DISK
00016          *         10 TO 15 DEVICE ADDRESS
00017          *
00018          *
00019          *
00020          *         DESCRIPTION
00021          *
00022          *         BEBOOT LOADS ONE RECORD ONTO LOCATION /80 THEN START AT /84
00023          *         THE RECORD IS THE SECTOR # 1 IF DISK,OR 254 CHARACTERS OF THE
00024          *         INPUT DEVICE,LEADING NULL CHARACTERS IGNORED
00025          *
00026          *
00027          *
00028          *         USED REGISTERS :
00029          *
00030          *         A1 BOU LINES,NOT TO BE DESTROYED IF BOOT IS CALLED AGAIN
00031          *         A2 ADDR OF INR INSTRUCTION
00032          *         A3 ADDR OF CIO INSTRUCTION(WHICH IS DESTROYED AND NEEDS TO BE
00033          *                          RESTORED IF BOOT IS CALLED AGAIN)
00034          *         A4 ADDR OF SST INSTRUCTION
00035          *         A5 MULTIPLEX : CONTENTS OF 1ST WORD TO BE SENT TO EXT REGISTER
00036          *            IO BUS : CHARACTER COUNT,INITIALIZED AT 254 AND DECREMENTED
00037          *         A6 MULTIPLEX : CONTENTS OF 2ND WORD TO BE SENT TO EXT REGISTER
00038          *                          (LOADING ADDR)
00039          *            IO BUS : ADDR OF NEXT CHAR TO BE LOADED,INIT AT /80 AND
00040          *                          INCREMENTED
00041          *         A7,A8 WORK REGISTERS
00042          *         A9 WORK REGISTER
00043          *         A10 TO A14 NOT USED
00044          *         A15 CONTAINS THE KEYS'VALUE
00045          *
00046          *
00047          *
00048          *
00049          *
00050          *
00051          *
```

```
00052                                   EJECT
00053                                   AORG    0
00054                           *
00055                           *
00056                           BOOT    EQU     *
00057                           *       INITIALIZE REGISTERS
00058  0000  0200  F                    LDK     A2,INR          ADDR OF INR INSTRUCTION
00059  0002  0300  F                    LDK     A3,CIO          ADDR OF CIO INSTRUCTION
00060  0004  0400  F                    LDK     A4,SST          ADDR OF SST INSTRUCTION
00061                           *                               EXTRACT DEVICE ADDR AND INIT I/O COMMANDS
00062  0006  861E                       LDR     A6,A15
00063  0008  263F                       ANK     A6,/3F          DEVICE ADDR
00064  000A  9629                       ADRS    A6,A2           INITIALIZE I/O INSTRUCTIONS
00065  000C  962D                       ADRS    A6,A3
00066  000E  9641                       ADS     A6,HIO
       0010  0000  F
00067                           *                               EXTRACT CONTROLLER ADDR AND INIT WER INST
00068  0012  871E                       LDR     A7,A15
00069  0014  3FC8                       SLC     A7,8            MULTI OR SINGLE DEVICE CONTROLLER
00070  0016  5600  F                    RF(6)   INIT20          SINGLE ONE
00071  0018  260F                       ANK     A6,/F           MULTIPLE ONE
00072                           INIT20  EQU     *               INITIALIZE MULTIPLEX DBLE WORD:
00073  001A  9631                       ADRS    A6,A4           SST INSTRUCTION
00074  001C  3E41                       SLL     A6,1
00075  001E  9641                       ADS     A6,WER1         SET UP WER INSTRUCTIONS
       0020  0000  F
00076  0022  9641                       ADS     A6,WER2
       0024  0000  F
00077                           *                               LOAD A1 WITH BOU CONTENTS
00078  0026  811C                       LDR     A1,A7
00079  0028  0550                       LDK     A5,80           MULTIPLEX DOUBLEWORD: LOAD 80 CHAR INTO
00080                           *                               LOCATION /80
00081  002A  0680                       LDK     A6,/80
00082                           *                               CHECK IF DISK
00083  002C  3FE7                       SRC     A7,7
00084  002E  5600  F                    RF(6)   NODISK          NO
00085                           *                               FIXED HEADS ?
00086  0030  3FC1                       SLC     A7,1
00087  0032  5600  F                    RF(6)   NOSEEK          YES
00088  0034  0103                       LDK     A1,3            SEEK ZERO
00089  0036  41C0                CIO    CIO     A1,1,0          CIO SEEK ZERO
00090                           NOSEEK  EQU     *
00091  0038  811E                       LDR     A1,A15
00092  003A  3966                       SRL     A1,6            SECTOR NUMBER
00093  003C  213C                       ANK     A1,/3C
00094  003E  8520                       LDKL    A5,/80CD        1ST WORD OF MULTIPLEX FOR DISK DEVICE
       0040  80CD
00095                           NODISK  EQU     *
00096                           *                               EXECUTE WER,WHATEVER THE CHANNEL IS
00097                           WER1    EQU     *
```

```
00098  0042  7500                WER     A5,0
00099                    WER2    EQU     *
00100  0044  7601                WER     A6,1
00101  0046  F031                EXR*    A4          SST
00102  0048  F02D                EXR*    A3
00103  004A  5C06                RB(4)   *-4
00104  004C  871E                LDR     A7,A15
00105  004E  3F43                SLL     A7,3
00106  0050  5600  F             RF(6)   SST         MULTIPLEX
00107                    *
00108                    *                           IO BUS
00109                    *
00110  0052  8194                LDR     A9,A5       IF A9=A5 IGNORE LEADING CHAR.
00111                    INR     EQU     *
00112  0054  4F00                INR     A7,0,0      READ ONE CHAR
00113  0056  5C04                RB(4)   *-2
00114  0058  E994                CWR     A9,A5       LEADING CHAR?
00115  005A  5400  F             RF(4)   INR10       NO
00116  005C  27FF                ANK     A7,/FF      CHECK IF NULL
00117  005E  580C                RB(0)   INR         YES,IGNORE
00118                    *                           NO,CHECK IF 4*4
00119                    *
00120                    INR10   EQU     *
00121  0060  879E                LDR     A15,A15     4*4
00122  0062  5600  F             RF(6)   STORE       NO 8*8
00123  0064  3F44                SLL     A7,4        YES,4*4
00124  0066  809C                LDR     A8,A7       SAVE LEFT BITS
00125  0068  F029                EXR*    A2          READ NEXT CHARACTER
00126  006A  5C04                RB(4)   *-2
00127  006C  270F                ANK     A7,/F       GET 4 RIGHT MOST BITS
00128  006E  9702                ADR     A7,A8
00129                    STORE   EQU     *
00130  0070  E739                SCR     A7,A6       STORE CHAR
00131  0072  1601                ADK     A6,1        NEXT CHAR ADDR
00132  0074  1D01                SUK     A5,1        COUNT DONE ?
00133  0076  5924                RB(1)   INR         NO
00134                    *                           YES
00135                    *
00136  0078  4180                HIO     CIO     A1,0,0
00137                    *
00138                    STATUS  EQU     *
00139  007A  4FC0                SST     SST     A7,0
00140  007C  5C04                RB(4)   *-2
00141  007E  0F84                AB      /84
00142                    *
00143                    *
00144                    *
00145                            END     BOOT
```

```
          SYMBOL TABLE

  BOOT     0000  A    INR      0054  A    CIO      0036  A    SST      007A  A
  HIO      0078  A    INIT20   001A  A    WER1     0042  A    WER2     0044  A
  NODISK   0042  A    NOSEEK   0038  A    INR10    0060  A    STORE    0070  A
  STATUS   007A  A

          ASS,ERR,    00000
:EOF
PROG ELAPSED TIME:  00H-00M-00S-000MS-
BEA IPL52S
DATE     /   /     TIME 24H-60M-60S-
```

To standardize the system generation procedure for all systems, a set of system generation processors has been developed which provides great flexeibility, extensive logging of the process and improved efficiency.
The three steps inherent in any system generation process, i.e. monitor tables generation, monitor body generation and system medium generation are handled by a number of generation processors which are loaded and started successively.

For the generation of a Basic Real Time Monitor these are:

- GENMON, a generation monitor used only during the system generation process to run the generation processors.
- BRMGEN, which generates the monitor tables from the answers it receives from the user in a conversational process with a standard list of questions.
- IPLGEN, which generates the Initial Program Loader (IPL), the first module to be stored on the system medium. IPLGEN runs under any monitor.
- GENLKE, which runs under GENMON and scans the library of system modules, to select the ones requested during the BRMGEN phase and link them with the monitor tables generated during BRMGEN.

Depending on his configuration, the user may receive his sysgen tools, i.e. the above-mentioned processors, on punched tape or on cassette. In the first case, each processor is contained on a separate punched tape, in the second case all the processors are contained on one cassette.
In the description which follows in the paragraphs below, the cassette case is the basis, as this is the assumed standard for this sysgen process. It is very easy for the user, however, to redefine these standards (under GENMON) in case he works with punched tape. Apart from the redefinition of the standards, the main difference in the description is that from cassette the successive processors can be loaded and started without any manual

operation, whereas with punched tape, for each following step a new tape with the following sysgen processor must be put on the tape reader and then loaded and started.

In the following paragraphs, the whole set of operations necessary for generation of a BRTM is described in a number of sections corresponding to the system generation processors listed above. At the end of each section a number of notes and remarks is given, which the user must carefully read before starting the operation.

OPERATION

The minimum configuration required for generating a BRTM is:
- CPU with 16k word memory
- typewriter with address 10
- paper tape reader and punch, or:
  two magnetic tape cassette drives on 1 control unit

If the configuration is paper tape-oriented, the user receives 5 paper tapes, containing:
- IPL + GENMON
- BRMGEN
- IPLGEN
- GENLKE
- BRMLIB (BRTM Library)

If the configuration is cassette tape-oriented, the user receives one so-called generation cassette, containing:
- on side A: IPL
  GENMON
  BRMGEN
  IPLGEN
  GENLKE
- on side B: BRMLIB (BRTM Library)

If the user is going to generate his own system on cassette tape, he needs two cassettes of his own:
- one system cassette, referred to in the rest of this chapter as cassette S

- one working cassette, for temporary storage, referred to as cassette W.

If the system is going to be generated on paper tape or magnetic tape, these cassettes are not required, of course.
All this must be defined during the GENMON phase, where the user can accept the standard cassette case definitions or redefine file codes which he must do if he uses paper tape for the system generation process. During the GENMON phase the GENMON outputs questions in reply to which the user can make these file code definitions. Also the device addresses, configuration and interrupt levels are fixed during this phase.

Note:
Throughout the chapter, user replies typed in response to questions output by one of the sysgen processors, are underlined.

To start the process:
- switch on the CPU
- for cassette:
  - load the generation cassette, with side A up, in cassette drive TK05
  - load cassette W (working cassette) in cassette drive TK15;
  - set the data switches on the CPU control panel to allow the bootstrap to load from TK05:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	1	1	1	1	0	0	0	0	1	0	1

(hexa 0785)

(for the significance of the bits, see page 1-46 ; suffice it to mention here that bit 3 is 0 if the cassette drives are connected to the I/O processor and 1 if they are connected to the programmed channel, and bits 10 to 15 contain the device address).
- for paper tape:
  - put the tape containing IPL +GENMON on the tape reader and make it operable
  - switch on the paper tape punch and feed tape
  - set the data switches on the CPU control panel to allow the bootstrap to load from the tape reader:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	(hexa 1020)

(for the significance of the bits, see page 1-46 ; suffice it to mention here that bit 3 is 1 because the reader is connected to Programmed channel and that bits 10 to 15 contain the device address which is here assumed to be 20).

Then:
- press the MC button
- press the IPL button

Now the bootstrap is loaded which loads the first sysgen processor from either the cassette in drive TK05 or the paper tape reader into memory:

GENMON

When GENMON is loaded its identification is output on the typewriter:
GENMON
When loading is terminated,
:EOS
:EOF
is output on the typewriter, followed by the question
STANDARD CONFIGURATION?
The reply to this question can be Y[ES] or N[O].
If the user replies Y followed by (LF) (CR), GENMON assumes the following standard configuration definition:

- typewriter : TY10 at level /6
- tape reader : PR20 at level /4
- tape punch : PP30 at level /5
- line printer : LP07 at level /17
- card reader : CR06 at level /15
- cassette tape : TK05 at level /14
  TK15 at level /14
  TK25 at level /14
- magnetic tape : MT04 at level /13
  MT14 at level /13
  MT24 at level /13

If the user replies N , i.e. if one or more of the device addresses or levels is different from the standards above, GENMON outputs the following list on the typewriter, thereby allowing the user to define the configuration himself:

```
TY:
PP:
PR:
LP:
TK:
MT:
CR:
DK:    (answer NO, for no discs used in configuration)
LKM LEVEL:    (standard = 1)
RTC LEVEL:    (standard = 2)
PANEL INTERRUPT LEVEL:    (standard = 7)
```

For each of the devices listed, the user can reply as follows:
- (CR) if he wants the standard address and level (see above);
- <address>,<level> if one of these is different from the standards, followed by (CR)
- N or NO followed by (CR) if he wants the device excluded from the system.

When the user has terminated his reply to the first question, GENMON types out:
STANDARD FILE CODE ASSIGNMENT?

The procedure here is the same as for the first question: the reply may be either Y[ES] or N[O].
If the user replies Y or YES followed by (LF) (CR) GENMON assumes the following standard file code assignments:

- file code 1: TY10 (system keyboard)
- file code 2: LP07 (listing output)
- file code 3: TK15 (object output)
- file code 4: TK05 (object input)
- file code 5: TY10 (system typewriter)
- file code 6: TK05 (object input)
- file code 7: TK25 (object input)
- file code 8: PR20 (object input)
- file code A: TY10 (sysgen source input)
- file code B: TK15 (sysgen object input)

If the user replies N or NO to this second GENMON question, i.e. if one or more of his file code assignments is going to be different from the standard list above, which is the case when the user works with paper tape, GENMON outputs the following list on the typewriter, thereby allowing the user to give his own file code assignments (since the GENMON processor is the same for all monitors, some of the file codes given in this list are irrelevant to the generation of a BRTM and the user must type in NO after those):

```
LOAD INPUT DEV. AND MAIN LKE INPUT DEV.  F.C./4:  (standard = TK05)
SYSGEN INPUT DEV.                        F.C./A:  (standard = TY10)
SYSGEN OUTPUT DEV.                       F.C./B:  (standard = TK15)
AUX. LKE INPUT DEV 1                     F.C./6:  (standard = TK05)
AUX. LKE INPUT DEV 2                     F.C./7:  (standard = TK25)
AUX. LKE INPUT DEV 3                     F.C./8:  (standard = PR20)
AUX. LKE INPUT DEV 4                     F.C./9:  (type in NO)
IPLGEN/LKE/CASLOAD OUTPUT DEV.           F.C./3:  (standard = TK15)
LISTING OUTPUT DEV                       F.C./2:  (standard = LP07)
CASLOAD INPUT DEV.                       F.C./C:  (type in NO)
DISLOAD INPUT DEV.                       F.C./E2: (type in NO)
```

For each of the file codes listed, the user can reply as follows:

- (CR) if he wants the standard assignment (see above); it is also possible to type in the standard device name and address followed by (LF) (CR)
- <dev.name><dev. address> if one of these is different from the standards, followed by (LF) (CR)
- N or NO followed by (CR) if he does not want this file code taken into account.

When the user has terminated his reply to this question, GENMON types out:

END OF GENMON INITIALIZATION
READY TO LOAD PROGRAMS

Now the user can proceed to the next phase: BRMGEN.

Notes on GENMON:

For the question STANDARD CONFIGURATION:

- From the time the MC button has been pushed up to the end of GENMON initialization, no Ready interrupts (from cassette or magnetic tape especially) should occur.
- If the user has answered N or NO to this question, in the list typed out by GENMON, specification of a level is mandatory for LKM LEVEL. For RTC LEVEL it is mandatory if the CPU key is in ON/RTC or LOCK position. For PANEL LEVEL it is also mandatory.

For the question STANDARD FILE CODE ASSIGNMENTS:

- The standards imply sysgen from cassette tape. Therefore they must be redefined if sysgen is done from paper tape.
- file code/4: all programs will be loaded from this file code. During the Syslink phase it is used as GENLKE object input, so it must be cassette or punched tape.
- file code /A: from this file code the parameters for table generation (BRMGEN) will be read. This may be done in interactive mode (e.g. TY) or not (e.g. PTR, cassette, mag tape or card reader)
- file code /B: this may be cassette, punched tape or magnetic tape.

- AUX. INPUT DEV.: these may be assigned in advance, especially for Syslink if libraries are to be scanned on various devices.
- file code /3: this is the main output device (sequential)), i.e. cassette tape, punched tape or magnetic tape.
- file code /2: for logging of the sysgen operation: LP.
- file codes /4, /A, /B, /3 are mandatory;
  file codes /6 up to /9 and /2 are optional.

- When answers to GENMON questions are given on an ASR typewriter, they must not be typed in before the bell signal because of the low speed of the GENMON I/O module. This is not necessary for devices on V-24 interface such as the matrix typewriter P842, because they work in echo mode.

BRMGEN

When GENMON initialization is terminated and the message READY TO LOAD PROGRAMS has been output the user can start the second phase, BRMGEN, the building of the BRTM tables. This is done by typing in replies to questions output on the typewriter by BRMGEN. From these replies BRMGEN builds the tables and records them on the medium with file code /B, i.e. in standard cases cassette drive TK15.
When the questions and answers are handled via the typewriter, this phase is done in conversational mode. It is also possible however, to do it in non-conversational mode, for example by having the questions and answers pre-recorded on punched tape. In such a case file code /A must have been assigned to tape reader during the GENMON phase under the question STANDARD FILE CODE ASSIGNMENTS?

In any case, the BRMGEN phase is started as follows:
(when working with paper tape, take the IPL+GENMON tape from the reader, put the BRMGEN tape on it and make it operable)
- push the INT button on the CPU control panel
- on output of M: type in LD
- now the following sysgen processor, i.e. BRMGEN is loaded either from the tape reader or from the cassette tape and its identification is output:
  IDENT BRMGEN

- when loading is terminated,
  :EOS
  :EOF
  is output on the typewriter

(At this point, if file codes /A and/or /B have been redefined under GENMON, i.e. if they are not assigned to TY10 and to a cassette, prepare the relevant device. Normally, cassette drive TK15 should contain a working cassette now and be ready to receive the tables output by BRMGEN).

- push the INT button
- on output of M: type in ST
- Now BRMGEN is started and outputs the following messages on the typewriter:
  TABGEN INITIALIZATION
  IDENT SYSTEM DEFINITION
- then, if the user works in conversational mode, a series of questions is output, to which the user must type in the replies:

IDENT

Reply: Specify a character string of up to 6 alphanumeric characters, to be punched at the beginning of the module. This may be followed by a character string of up to 73 characters, containing comments. The comment field must be separated from the ident field by a blank.

Error Message: TG03: the first character is not alphabetic.

STACK SIZE

Reply: Note, that each device requires up to 20 words in the stack for each interrupt managed for that device. Specify up to 4 hexadecimal characters,

giving the size in words, of the stack which is used by the system to save registers when an interrupt occurs.

Error Message: TG03: the number specified is not hexadecimal.

USER INTERRUPT ROUTINES

Reply: Specify all user interrupt routines (usually drivers for special devices) which must be link-edited with the monitor for inclusion in the system, as follows:

<L>,<ENTRYi>
:
END

L,ENTRYi specifies the name (up to 6 characters) of a routine connected to level L.

END indicates that all routines have been declared or is used if none are entered.

Error Messages: TG03: syntax error
TG06: the first character of the name (ENTRYi/j) is not alphabetic
TG09: error in the level declaration.

POWER FAILURE

Reply: Specify the level of power failure option as follows:
<L> giving the level number in 1 or 2 hexadecimal digits.

Specify N if this feature is not available in the system.

Note: The standard system power failure routine performs only a HALT.

Error Message: TG03: parameter error.

REAL TIME CLOCK LEVEL

Reply: Specify the level for the real time clock, in one or two hexadecimal digits.

ABS TIME MANAGEMENT NEEDED

Reply: Specify Y if connection to absolute time is required, or N if it is not required.
This question will appear only if the answer to the question REAL TIME CLOCK LEVEL was neither N⌈0⌉or EN⌈D⌉.
If the reply is N, the connection to a timer remains avaliable, as well as the Get Date and Get Time features (monitor requests).

PANEL LEVEL

Reply: When operator communication (OCOM) modules are selected, the user must specify the level to which the control panel interrupt is connected. The level is specified as:
<L> a level number of 1 or 2 hexadecimal digits.
Specify N if no operator communication feature is used.

Error Message: TG03: parameter error.

LKM LEVEL

Reply: First specify the level to which the LKM interrupt is connected in the same manner as for power failure and real time clock. Then, define the standard monitor requests necessary for your

system. TABGEN prints the names of these monitor requests one by one, and after each one the user can answer:
Y if the monitor request is required
N if it is not wanted.
All monitor requests are optional.
The following list of standard monitor requests is printed by TABGEN:

IORM (Input/Output Requests)
WAIT (Wait for an Event)
EXIT (Exit)
BUFF (Get/Release Buffer)
CNTM (Connect Time)
DNTM (Disconnect Time)
ACT (Activate a Program)
SWTC (Switch Inside a Level)
ATDV (Attach a Device)
DTDV (Detach a Device)
GTIM (Get Time)
RSEV (Set an Event)
CNLV (Connect Level)
DNLV (Disconnect Level)
WGT (Wait for a Given Time)

Note: If the user issues a request for a non-existing function, later on in a program, the value-1 will be returned in register A7.

Error Message: TG06: the reply was neither Y nor N. The user must type the correct reply.

USER LKM

Reply: The user may specify his own set of monitor requests for inclusion in the monitor, as follows:

⟨N1⟩,⟨ENTRY1⟩

⟨Ni⟩,⟨ENTRYi⟩

⟨Nn⟩,⟨ENTRYn⟩

END

where Ni consits of two hexadecimal digits defining the DATA ⟨number⟩ which follows the LKM instruction, and ENTRYi is the symbolic entry point of the routine processing the LKM DATA Ni function.
The system will extend the monitor request table in which word i contains the address of ENTRYi. Therefore, during SYSLINK, the user must provide the Linkage Editor with the module containing all entry points ENTRYi specified here.
END indicates that all user LKM functions have been declared or that none are wanted.

Note: Ni must not be equal to any of the standard LKM DATA numbers.

Error Message: TG03: the first parameter is not hexadecimal
TG07: the first character of the second parameter is not alphabetic
TG08: syntax error.

PCT STANDARD LEVELS

Reply: BRMGEN will print a list of system programs, behind each of which the user may type Y or N to indicate whether he wants the standard level and assignations for that program or not.
If he types N, he must also type the level he wants, as follows: ⟨Level⟩
where level must be a software level between /31 and /3F exclusive.
The following program name is printed:

TIMER (Clock management: M:DCK2 module)

If Y is specified, BRMGEN prints and implements the following:

TIMER : 31

Note: The Idle Task (IDLTSK module) will always be on the level /31 (i.e. physically; logically it will operate on level /40).

NB OF PROG CONTROL TABLES

Reply: Specify the number of program control tables which must be reserved for your system. For each user program one entry in the program control table is allocated. Each entry occupies 18_{10} words.

The user need not reserve entries for the system programs.

Error Message: TG03: invalid reply.

DEVICES ON PROGRAMMED CHANNEL

(Note: devices on channels must be declared according to Release Bulletin)

Reply: Specify, which devices are connected to the programmed channel, as follows:

⟨DNDA⟩,⟨L⟩

:

END

where:

DN is the device name, in two ASCII characters.

DA is the device address, in two hexadecimal digits.

⟨L⟩ specifies the level to which the device is connected.

END indicates that all devices have been declared.

Note: For the opertor's typewriter, three devices (TY, TP, TR) must be declared with the same address, if they will all be used.
No check is made on device declaration.

Error Messages: TG01: the device specified is not supported by the system
TG02: device address error
TG09: level error
TG10: device not declared.
TG06: missing parameter.

DEVICES ON MULTIPLEX

Reply: Specify, which devices are connected to the multiplex in the same manner as for the devices on the programmed channel.

Note: For line printer there are additional parameters:
- Line printer:
  LPDA,L,LG,⟨number⟩
  where LG=⌊S⌋⌊L⌋ : S = 80-column printer
  L = 132-column printer.
  ⟨number⟩: a hexadecimal number specifying the number of lines per page wanted.

No check is made on the device declaration.

Device names used:

TR : ASR tape reader
TP : ASR tape punch
PR : high-speed reader
PP : high-speed punch
TY : operator's typewriter
CR : card reader
LP : line printer
TK : cassette tape unit
MT : magnetic tape unit

HIGHEST FILE CODE NUMBER

Reply: Specify the highest file code number (from /01 to /FF) used in any of the programs. This will indicate how many words the system must reserve for the File Code Table.

Error Message: TG03: syntax error.

SPECIFY FILE CODES

Reply: Specify all file codes used by the programs. If system processors are used, their standard file codes must be declared. The standard file codes are: 01:source input, 02:listing output, 03:punch output, 04:object input, 05:operator's typewriter. Declaration is done as follows:

```
⟨FC⟩,⟨DNDA⟩
      :
⟨FC⟩,⟨DNDA⟩
      :
    END
```

where FC is a file code (2 hexadecimal digits) assigned to the device indicated by DN with address DA.

Note: File code 05 is used by the system for input/output to/from the operator's typewriter. It cannot be assigned or reassigned by AS operator commands.
So, if used, it is necessary to declare at least this file code. All other file codes may be assigned at execution time by means of the operator command AS.

Error Message: TG08: syntax error
TG10: device has not been declared.

FILE CODES ASG TO USER DEVICES

Reply: The user may declare all file codes assigned to his non-standard devices. The related I/O drivers (including Device Work Tables) and the interrupt routines for these devices must be written by the user and be incorporated in the system during the SYSLINK phase. (The entry points for the interrupt routines must be declared under USER INTERRUPT ROUTINES). These file codes must be declared here, as they cannot be assigned by AS operator message. The reply must be as follows:

⟨FC⟩,⟨DWTi⟩

END

where:
FC is a two-digit hexadecimal file code which will be generated in the File Code Table.
DWTi is the entry point (a name of up to six characters) of the Device Work Table (DWT) associated with this device.
END indicates that all file codes have been declared.
Any number of file codes can be assigned to the same DWTi. The system checks only if the file code is a two-digit hexadecimal number, but not if it has already been used or is one of the standard file codes used by the system processors.

Error Messages: TG03: syntax error, e.g. the file code is not a hexadecimal number.
TG07: the first character of the DWT is not an alphabetical character.

SIMULATED INSTRUCTIONS

Reply: Y or N, depending on whether the simulation package must be included or not. If the reply was Y, the following list is output:

MULTIPLY:

DIVIDE:

D ADD:

D SUB:

D SHIFT:

MLR - MSR:

After each item, the user must type in Y or N to indicate whether he wants that instruction simulation routine included or not.
For P856 or P857 N must be typed, since these instructions are included in their instruction sets.

SIMULATED ROUTINES SAVING AREAS NB

Reply: This question is output only if the reply to the previous question was Y.
The user must type in the number of save areas required by the simulation package.

MAX NUMBER OF SCHEDULED LABELS

Reply: Type in a two-digit hexadecimal number, specifying the maximum number of scheduled labels which may be in queue at the same time. This will be the length of the FILLAB table described in the paragraph on Scheduled Labels.

Note: This is not the maximum number of scheduled labels used in the program, which may be a higher number.

TABGEN ENDED

IPLGEN

During this phase an Initial Program Loader for the user's system will be generated. This implies, that at this point, he must prepare his system medium.
In standard cases, the output from IPLGEN will be onto file code /3, i.e. the cassette in drive TK15, so the user must put his system cassette S into drive TK15 with side A up and wait for it to rewind.
If the user works with paper tape, i.e. when he has redefined file code /3 to the tape punch under GENMON, he must tear the tape with the tables generated during BRMGEN from the punch and feed new tape to prepare it for the punching of the IPL and, during the next phase, the BRTM.

Now IPLGEN can be loaded and started:
- when working with paper tape, take the BRMGEN tape from the reader, put the IPLGEN tape on it and make it operable)
- push the INT button on the CPU control panel
- on output of M: type in LD
- now the next sysgen processor is loaded either from the cassette tape or from the paper tape reader (file code/4) and its identification is typed out:
  IDENT IPLGEN
- when loading is terminated, :EOS :EOF is output on the typewriter
- push the INT button
- on output of M: type ST
- Now the IPLGEN processor is started and an Initial Program Loader (IPL) is written at the beginning of the user's system medium.

Note:
- When the IPL is generated onto cassette or magnetic tape, the positioning of this tape must remain stable until the user's monitor which must follow it is recorded onto it. Remember that a cassette is rewound when taken out of the drive and put back in again!
- The generated IPL can be used for any device.

GENLKE

During this phase the final user monitor is obtained by linking the tables generated under BRMGEN with the monitor modules required from the BRTM Library, and/or any User Library or Extension Libraries (see Note at the end of this section).
In any case, the GENLKE processor must now be loaded:

- when using paper tape, take the IPLGEN tape from the reader, put on the GENLKE tape and make the reader operable)
- push the INT button on the CPU control panel
- on output of M: type in LD
- now GENLKE is loaded from the cassette tape or the tape reader (see file code /4 under GENMON) and its identification output:
  IDENT GENLKE
  when loading is terminated,
  :EOS
  :EOF
  is output on the typewriter.

With cassettes:

On the basis of the availability of only two cassette drives, the cassettes are now handled as follows:

- take the generation cassette from TK05 after GENLKE has been loaded.
- put the working cassette W with the tables generated under BRMGEN in drive TK05 and wait for it to rewind.
- start GENLKEas follows:
  push the INT button
  on output of M: type in ST
- now GENLKE outputs
  L:
  and the user must type in the link-edit command as follows:
  E⌈:<decimal number>⌉,<module name>⌈,L8|4⌉⌉
  where <decimal number> consists of 3 digits:
  - the first is the file code for the object output device
  - the second is the file code for the listing device

- the third is the file code for the object input device.

<module name> is the name of the user's BRTM.

8 or 4 is used if the monitor is punched on paper tape to indicate whether it must be punched in 4-track or 8-track format.

If the standard file codes are used (see under GENMON, i.e. /3, /2 and /4 respectively), they need not be specified and the command can be given as:

E,<module name>

- then GENLKE outputs

L:

to which the user must reply with:

P

The tables generated during BRMGEN are now recorded from the cassette W in TK05 onto the user's system cassette in TK15.

- when this is finished take the working cassette W from TK05
- put the generation cassette with side B up into drive TK05 and wait for it to rewind. It is now positioned correctly for the scanning of the BRTM Library.
- in response to the

L:

output by GENLKE, now type in

L

upon which GENLKE will start scanning the BRTM LIbrary, select the required modules and record them onto the user's system cassette in TK15. The names of the selected modules are output on the listing device, together with their base addresses and any comments included in the identifiers.

- when this is finished, GENLKE again types out

L:

The user must now type in

U

to check if there are any unsatisfied references. The last module to be included must be INIMON, so if GENLKE types

INIMON

after the user has typed his first U, it is correct. The

GENLKE processor then types out

L:

and the user can type

L

to solve this last unsatisfied reference.

(If there were more unsatisfied references, the user must repeat this L:L process until INIMON is the last unsolved reference and then give his last L command).

- Now, after all modules have been included, GENLKE again types

L:

The user once more types

U

to make sure that all references have been solved. Then, on the next

L:

the user types

T

to indicate the end of the GENLKE phase.

GENLKE then outputs the symbol table of the generated BRTM on the listing device and on the typewriter it outputs monitor length (L=XXXX), the monitor start address (S=XXXX) and the first free location after the monitor in memory (E=XXXX).

Note:

If the user has 3 cassette drives and wants to use them all during this phase, the procedure is as follows:

- take the generation cassette from drive TK05 and put it back in with side B up (BRTM Library). Wait for it to rewind.
- put the working cassette W, containing the tables generated under BRMGEN in drive TK25 and wait for it to rewind.
- start the GENLKE processor:

  INT button

  M: ST

  and on output of

  L:

  type in the option message as follows:

E:327,<module name>,8

where 3 and 2 are the normal object output and listing file codes, but 7 is assigned to input file code, which normally is /4 (TK05), but now must be /7 (aux. LKE input device TK25), because TK25 contains the working cassette with the tables. See also under GENMON for the file code assignments.

- when this is accepted, GENLKE outputs

  L:

  and the user must type

  H

  Upon this command, GENLKE scans the input file (i.e. the working cassette with the tables) up to EOF and then goes into Pause state.
- now GENLKE is restarted with an RS command with a new input file code, switching it back from /7 to /4, the normal input file code and the one assigned to TK05, which contains the BRTM Library which must now be scanned. So:

  push INT button and on output of M: type

  RS 04

  GENLKE now starts scanning and selecting the required modules from the BRTM Library and the rest of the procedure is the same as described above , starting after the user's first L command.

With paper tape:

- in this case the procedure is basically the same as with cassettes (see description above), but input is done from the tape reader and output onto tape punch. On the tape punch an IPL has already been generated and the paper tape should be left as it is.
- first the GENLKE tape must be put on the reader and GENLKE is loaded into memory by LD command. Then the tape containing the tables generated under BRMGEN is put on the reader and GENLKE is started with an ST command. Having entered the E: option command, the user types P and the tables are processed.
- then the BRMLIB is put on the tape reader and the user types L after which this library is scanned and the selected modules are output on the punched tape.

- having checked if INIMON is the last unsolved reference with a U command and having typed L to solve it, the user then types T to terminate the process.
- on the tape punch, a BRTM has now been generated.

Note:

If modules from other (paper or cassette) tapes beside the BRMLIB tape must be link-edited during this phase, the tapes must be scanned in a defined order, which is:

- User Library tape(s), if any
- Extension tape(s)
- BRMLIB tape

When more references than INIMON remain unsatisfied, rescanning must start at the first step in this sequence.

This terminates the system generation process. The user now has a complete BRTM.

Programming Rules

With TABGEN, the user can combine his own routines with the standard system routines. This entails that the interfaces must not be modified and that the routines must be written in accordance with these interfaces.
With respect to this, some guidelines are given below.

- *Interrupt Routines*

- If the routine does not reset any event and does not have to run in enable mode, any registers can be saved in the A15 stack. They will have to be restored at the end of processing, before returning to the interrupted routine via A15.
  Example:
  An interrupt routine running in inhibit mode, using 3 registers:

Save 3 registers in A15 stack:	STR instructions	(For P856/7 or with simulation on P852: MSR 3,A15)
Reset interrupt:	RIT,INR,OTR....	
Process.		
Restore 3 registers from A15 stack:	LDR instructions	(or: MLR 3,A15)
Return to interrupted context:	RTN A15	

- If the routine resets an event or has to run in enable mode, it is mandatory to have stored in the A15 stack the P-register, PSW and the first 8 registers before switching to enable mode. Moreover, at the end of processing a branch must be made to the dispatcher without restoring the registers.

If no label is scheduled, A6 must be 0 and A5 contents is irrelevant.
If a label is scheduled, A6 must contain the scheduled label address and A5 the Program Control Table address.

For example:
An interrupt routine resetting an event:

Save 8 registers in the A15 stack:	STR instructions	(or for P856/7 and with simulation on P852: MSR 8,A15)
Reset interrupt:	RIT,INR....	
Process.		
	LDKL A6,SCLAB	
Do **not** restore registers.	LDKL A5,PCTAD	
Branch to dispatcher:	ABL DISPAT	

Note: - DISPAT,SCLAB,PCTAD must have been defined as an external. The dispatcher address can be found in the Communication Vector Table.
- It is always mandatory to reset the interrupt signal by executing a special instruction.

- *I/O Drivers*

When an I/O request is given, the monitor will call the IORM module to process the request. The control will be transferred to the driver, the start address of which is stored in the Device Work Table (DWT).
- The driver runs at level 48.
- Register A6 contains the DWT address.
  Register A7 the I/O order.
  Register A8 the ECB address.
- Having initialized the I/O, the driver must return to the dispatcher by means of:
  ABL C:WAIT
  with A7 containing the I/O order and A8 the ECB address.

- Regarding I/O interrupts, the driver must respect the rules laid down for interrupt routines.
- When the I/O has been completed (after the SST command), the driver can switch to level 48 as follows:
  A1 must contain the return address, then:
  AB L:VCH
- The driver can return to the ENDIO module by branching to the appropriate entry point:

  R:TUR2: first post-edit, then return to calling program.
  Status of the I/O operation can be found in register A2.

  R:TUR4: return to calling program.
  Status of the I/O operation can be found in register A2.

  R:TURN: return to the interrupted program.

  M:RETR: output an error message. If:
  A1=0: print the message with RY.
  A1≠0: print the message without RY.

- *User-written Monitor Requests*

The module processing the user monitor request must be included in the system at SYSLINK time; thus, the entry point is added to the monitor table. When the request is made, it is processed as follows:
- the monitor saves register A1 to A8 of the calling program in the A15 stack.
- the LKM interrupt is reset.
- the module processing the monitor request is called via a branch instruction.
- when the processing routine is called:
  A6=0: if no scheduled label is used.
  A6≠0: then it contains the address to be scheduled when the event occurs.
  A5 must then contain the PCT address.
- the processing routine has to return to the dispatcher by ABL DISPAT where DISPAT must have been declared as an external.

The processing routine is thus called by the system at the level of the LKM. If the user wants to switch to a lower level, i.e. that of the monitor, he must give the following instructions:

```
LDKL   A1,48 or 49 (monitor level)
CF     A15,CHLEV
```

Where CHLEV must have been declared as an external.

- *User-written Operator Commands*

Operator commands are activated by control panel interrupt. The control panel routine processes the interrupt, saves registers and decodes the first two characters of the command (i.e. the mnemonic). Then control is transferred to the processing routine, which must follow these rules:
- the routine runs at level 49.
- the message is read into buffer BUFCP, consisting of 80 characters.
- A5 contains the address of a flag, used as follows:
  If 0: accept new operator command.
  If 1: do not accept DM and MC commands.
  This flag is reset to zero when the routine is called, so the routine must set the flag to 1, by means of IMR A5.
- the routine must return to the label CPTRN which must have been declared as an external.

  Note: All input/output must be performed without wait.

PART 2 MONITOR CONFIGURATION

1 System Components

The Basic Real Time Monitor consists of a number of modules, from which the user can select those that are required for his application and configure his own monitor. The three main parts of the BRTM are:

- INIMON (monitor initialization program)
- Nucleus (system tables and monitor modules)
- System Interrupt Modules.

A number of modules are optional and can be added to the ones contained in these groups.
All modules are centered around the dispatcher, a monitor routine which determines which routine is to be executed next, on the basis of priority levels (see chapter 3).

INIMON

INIMON is a monitor initialization program. Among other things, it loads the Basic Real Time Monitor and all user programs. It also entails some initialization procedures:

- date and time are recorded;
- programs may be connected to priority levels and activated;
- the Real Time Clock is started if the control panel key is in ON/RTC position.
- the system stack (register A15) is initialized;
- the memory size is calculated.

Full details on this procedure are given in Part 1, chapter 9.
After completion of all these procedures, INIMON gives control to the dispatcher and execution can be started.

NUCLEUS

The Monitor nucleus consists of a set of mandatory monitor modules and the system tables.

Monitor Modules
These modules handle part of the monitor requests which are made through LKM instructions. The following are mandatory and handle:

– Wait for an Event monitor request (M:WAIT)
– Activate monitor request (M:ACT)
– Exit monitor request (M:EXIT)
– Connect a Program to a Level monitor request (M:CNLV)
– Switch inside a Level monitor request (M:SWTC)
– Set Event monitor request (M:RSEV)

Also included here is the Dynamic Memory Allocation handler, for handling system or user requests for temporary memory space:

– M:DMA for allocation of memory space
– M:DML for deallocation.

System Tables
The following tables are part of the BRTM nucleus:

– T:LKM, a table with the calling addresses of all the routines which handle the different monitor requests.
– T:CVT, a table containing parameters on the memory configuration, such as size, stack base address, dispatcher address, etc.
– PCT, program control table. This is a pool of tables, where each table consists of 16 words, containing all relevant information about the program to which it belongs.
– T:SLT, the software level table, consisting of 16 entries, corresponding to levels 49 to 64. It is used by the dispatcher to find the current PCT connected to a given level.
- T:FLAG is a table containing some system flags, such as current PCT pointer.

Some tables are included only if the corresponding options are present:

– FCT, the file code table, containing the links between the logical file codes and the actual peripheral devices.
– DWT, the device work table, one for each peripheral device, containing the necessary parameters on the device and its logical handling.
– Timer Management tables.

Optional monitor modules
The following monitor modules are optional:

Handlers for:

– Disconnect a Level monitor request (M:DNLV)
– Get Buffer monitor request (M:GBUF)
– Release Buffer monitor request (M:FBUF)
– I/O monitor requests (IORM)
– End of I/O module (ENDIO)
– Attach/Detach Device to/from Program monitor requests (M:ATDT)
– Connect a Program to a Timer monitor request (M:CNTM)

- Disconnect a Program from a Timer monitor request (M:DNTM)
- Get Time monitor request (M:GTIM)
- Wait for a Given Time monitor request (M:WGT).

The following monitor modules are optional as well:

- I/O drivers, which perform the input and output for a specific device, using service routines such as INPUT, OUTPUT, COMIO, etc.,
- A Timer handler (M:DCK)
- Routines for the simulation of non-wired instructions.

SYSTEM INTERRUPT MODULES

This set of modules consists of a number of routines which service certain hardware interrupts. Some of the routines are optional. The set can be extended with user-written interrupt routines.

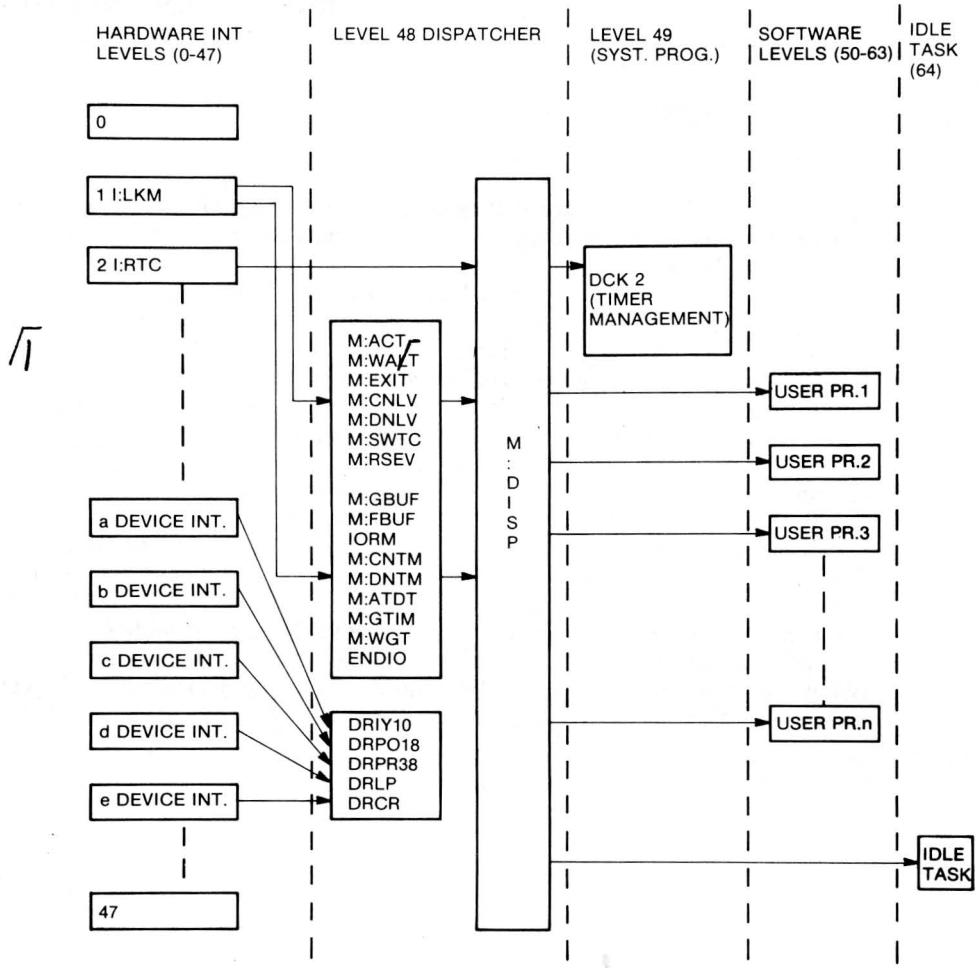
- I:PFAR and I:ITCP handle power failure and control panel interrupts.
- LKM Handler (I:LKM) handles the monitor requests given through LKM instructions and the interrupt which occurs when an 'illegal instruction code' is used. It normally operates on priority level 1.

The following system interrupt modules are optional:

- Real Time Clock Interrupt Routine (I:RTC), which handles the real time clock.
- I/O Interrupt routines (I:ASR, I:PR, I:PP, etc.) which handle the interrupts that come from the various standard peripheral devices.

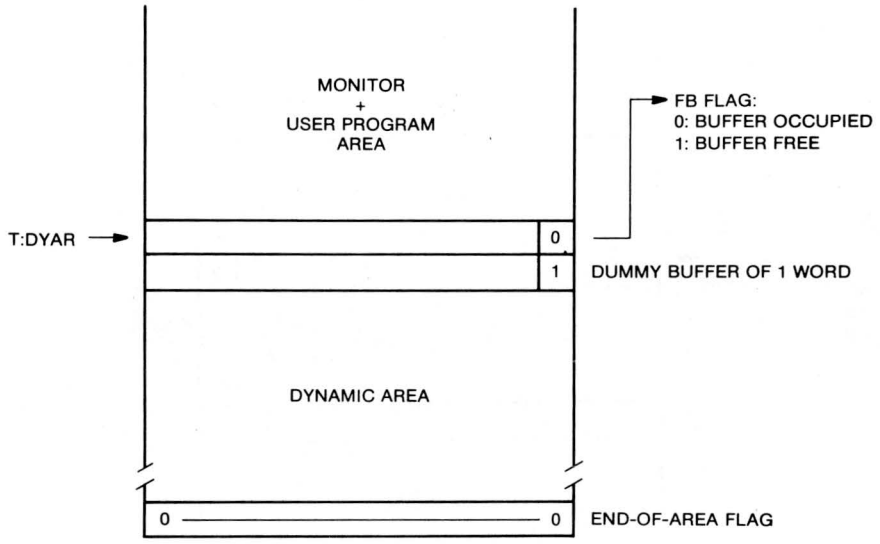
DISPATCHER

All monitor modules are centered around the dispatcher, a module which runs at level 48 and divides central processor time between programs according to their priority level.
When an interrupt has been handled, the dispatcher determines which program must be started and prepares its activation by loading its start address and register contents from the save area (a 16-word area in front of the program).
For hardware interrupt levels and level 48, control will be returned to the interrupted routine. For software levels (>48), the dispatcher will compare the priority levels of the programs which are active and not waiting for an event, in order to find the one with the highest priority. The dispatcher finds this information in the Program Control Tables, the addresses of which it looks up in the Software Level Table. If the highest priority program is not the interrupted program, it will receive control after the relevant data of the interrupted program (P-register, PSW, registers A1 to A14) have been stored in its save area.
If the interrupted program was also the one with the current highest priority, it will be restarted.
Below, an illustration is given of the relationship of the dispatcher to the other system parts in the priority structure:

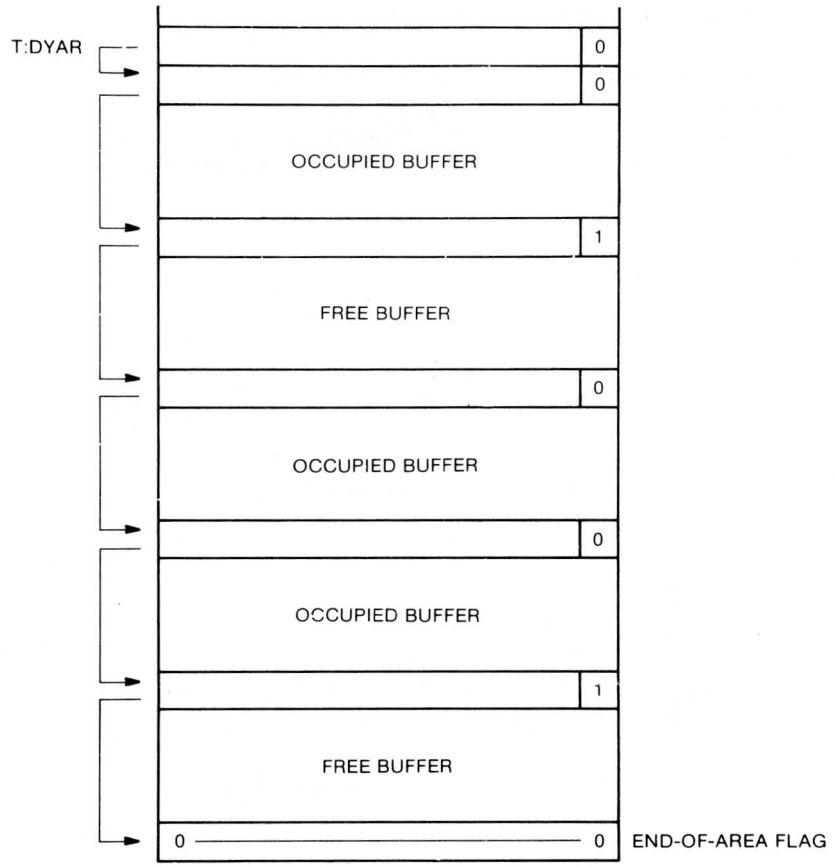

HARDWARE INT LEVELS (0-47)
LEVEL 48 DISPATCHER
LEVEL 49 (SYST. PROG.)
SOFTWARE LEVELS (50-63)
IDLE TASK (64)
0
1 I:LKM
2 I:RTC
a DEVICE INT.
b DEVICE INT.
c DEVICE INT.
d DEVICE INT.
e DEVICE INT.
47
M:ACT
M:WAIT
M:EXIT
M:CNLV
M:DNLV
M:SWTC
M:RSEV
M:GBUF
M:FBUF
IORM
M:CNTM
M:DNTM
M:ATDT
M:GTIM
M:WGT
ENDIO
DRIY10
DRPO18
DRPR38
DRLP
DRCR
M:DISP
DCK 2 (TIMER MANAGEMENT)
USER PR.1
USER PR.2
USER PR.3
USER PR.n
IDLE TASK

DYNAMIC MEMORY ALLOCATION AREA

The area of memory which remains after INIMON loaded the monitor and user programs, is formatted to be used for dynamically allocating buffer space to monitor or user programs. This area will have the following layout:

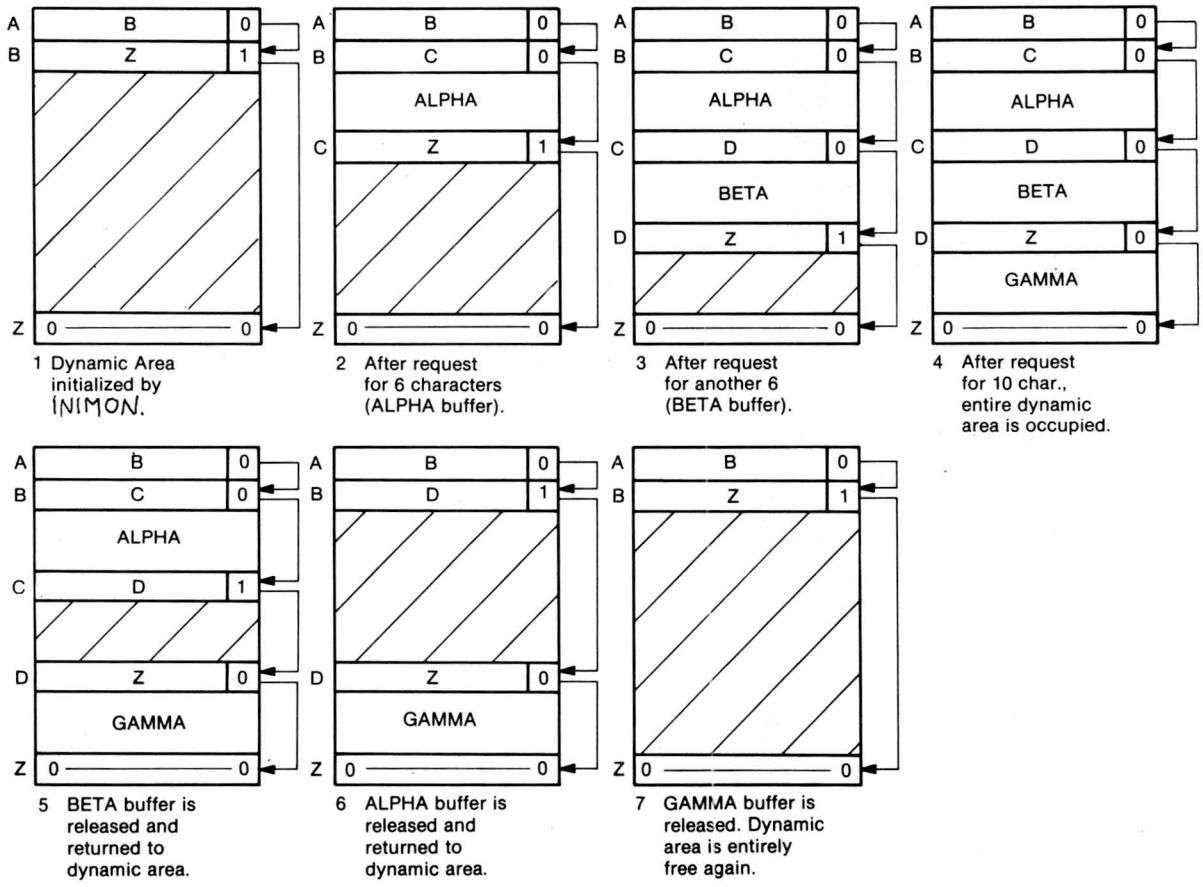


Blocks of memory space can be requested either by the system itself, or by the user through a *Get Buffer* monitor request. When a buffer is allocated, a buffer guide of one word is set in front of it, in which bit 15 is set to 0, to indicate that the buffer is allocated. If a request is sent for deallocation of this buffer, either by the system or by the user *(Release Buffer* monitor request), this bit is set to 1, to indicate that the buffer is free again. After a number of such requests, the dynamic area might look as follows:



Any new requests for allocation are handled in such a manner that the first free area encountered, which is large enough for the request, is allocated.
Allocation is done by the M:DMA module.
Deallocation is done by the M:DML module.
Below an example is given of a sequence of requests for allocation and deallocation of buffers in the dynamic area.

Note: If the user wants to reserve the upper part of memory for himself and make sure not to overwrite part of the dynamic area, e.g. in case of a stand-alone dump (which is done in the upper memory part), he may himself fix the limit of the dynamic memory allocation area, by setting it up in the word CVTMSZ in the Communication Vector Table (see Part 1, chapter 2).


A
B
Z
0
1
0
0
1 Dynamic Area initialized by INIMON.
A
B
C
Z
0
0
1
0
0
ALPHA
2 After request for 6 characters (ALPHA buffer).
A
B
C
D
Z
0
0
0
1
0
0
ALPHA
BETA
3 After request for another 6 (BETA buffer).
A
B
C
D
Z
0
0
0
0
1
0
0
ALPHA
BETA
GAMMA
4 After request for 10 char., entire dynamic area is occupied.
A
B
C
D
Z
0
0
0
1
0
0
ALPHA
GAMMA
5 BETA buffer is released and returned to dynamic area.
A
B
D
Z
0
0
1
0
0
GAMMA
6 ALPHA buffer is released and returned to dynamic area.
A
B
Z
0
1
0
0
7 GAMMA buffer is released. Dynamic area is entirely free again.

INTERNAL PROGRAM ORGANIZATION

Details about the coding of user programs and interrupt routines are given in Part 1, chapter 4.
Scheduled labels are handled as follows:
When a scheduled label is specified in a monitor request, this is recorded by the I:LKM module, which increments the event count in the PCT and transfers the following parameters to the corresponding monitor request handler:

A5: PCT address of requesting program
A6: Scheduled label address.

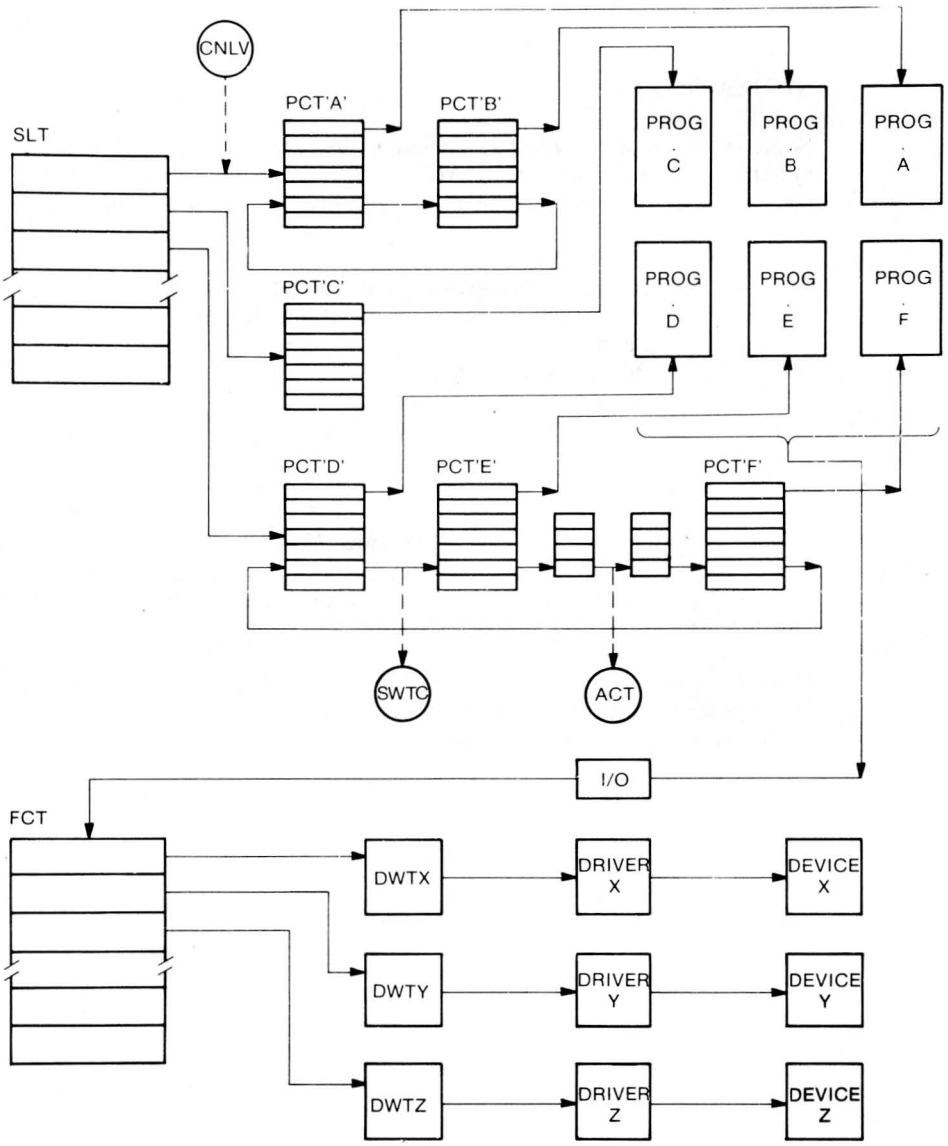
These parameters must be kept by the monitor request handler until it is necessary to pass them on to the dispatcher, e.g. when an event occurs. Then the dispatcher records the scheduled label in the FILLAB table (behind Save Area) of the related program, sets the SA bit in the word TIMAD of the PCT and decrements the event count. After this, the dispatcher transfers control on the basis of priority. If the program which is restarted has a scheduled label in its FILLAB table and is not already running in a scheduled label sequence, this scheduled label is taken from the table and started.

SYSTEM TABLES

Some of the monitor modules provide links between the user programs and system programs. These links are in the form of tables, some of fixed length, some of variable length. The tables provide the interface between user and system software;

- Each program has a **Program Control Table** (PCT) in the monitor area, where all the necessary parameters about the program are stored and updated by the monitor.
- The **Software Level Table** (SLT) holds the PCT addresses of the current programs for each software priority level (49-64), to enable the dispatcher to find the current highest priority program in case it has to give control to a program.
- When a program wants to perform an I/O operation, this is done through a file code for the desired device. In the **File Code Table** (FCT), the monitor will find the address of a **Device Work Table** (DWT), containing parameters about the device related to the file code specified. The DWT also contains the address of the I/O driver needed to perform the operation.

These four tables are the main interfaces to establish a working relationship between user programs, system programs and peripheral devices, as shown in the following illustration:


CNLV
SLT
PCT'A'
PCT'B'
PCT'C'
PCT'D'
PCT'E'
PCT'F'
PROG C
PROG B
PROG A
PROG D
PROG E
PROG F
SWTC
ACT
I/O
FCT
DWTX
DWTY
DWTZ
DRIVER X
DRIVER Y
DRIVER Z
DEVICE X
DEVICE Y
DEVICE Z

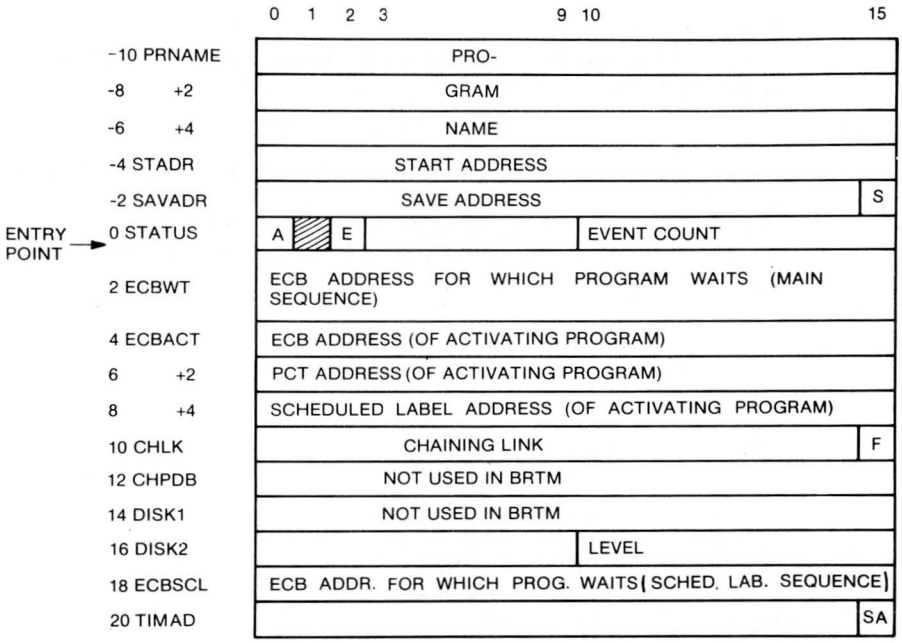
The following paragraphs show the layout of the tables and their contents:

Program Control Table
For each program loaded, the monitor builds a Program Control Table, (PCT) in the PCT Pool.

The address of each PCT is its Status word, the sixth word in the table.
The PCT length is defined in the monitor module T:SYS.
For each program loaded a table is created, in accordance with the illustration below:

```
STATUS    EQU   0
SAVADR    EQU   STATUS-2
STADR     EQU   SAVADR-2
PRNAME    EQU   STADR-6
ECBWT     EQU   STATUS+2
ECBACT    EQU   ECBWT+2
CHLK      EQU   ECBACT+6
CHPDB     EQU   CHLK+2
DISK1     EQU   CHPDB+2
DISK2     EQU   DISK1+2
ECBSCL    EQU   DISK2+2
TIMAD     EQU   ECBSCL+2
PCTLG     EQU   TIMAD-PRNAME+2      (PCT LENGTH)
```

The words in the table have the following meaning:



where:

PRNAME: is a 3-word block, containing the program name in ASCII.

STADR: is the start address of the program.

SAVADR: Adress of the save area of the program:
If S = 0, it is the address of the main program save area.
If S = 1, it is the address of the scheduled label save area.

STATUS: gives the status of the program. This word is the entry point of the PCT block.

- If A = 0: the program is in active state (set by Activate (M:ACT)).
- If A = 1: the program is in inactive state (set by Exit (M:EXIT), when Event Count is zero).
- If E = 0: the program has not made its exit.
- If E = 1: the program has made its exit (set by Exit (M:EXIT), when Event Count is not zero, reset when Event Count has become zero).

Event Count: This is a counter for the I/O requests and scheduled labels taking place. For I/O requests the counter is incremented by the module IORM and decremented by the module ENDIO. For scheduled labels, the counter is incremented by the module I:LKM and decremented by M:DISP.

The program cannot become inactive until the counter has become zero again (see bit E).

ECBWT: Address of the ECB for which the main program is waiting.

ECBACT: A block of three words to enable the **activating** program to synchronize itself with this program, i.e. the program activated by it. The block contains:
- ECB address
- PCT address
- scheduled label address.

The activating program can synchronize itself by giving a Wait monitor request after the Activate request or by giving a scheduled label at activation time. When the activated program is completed, the ECB should be updated by the user program before it exits and the label will be scheduled by the system.

CHLK: Chaining link (see below).

DISK 2: With the BRTM, this word contains only the level to which this PCT is connected (bits 10 to 15).

ECBSCL: Adress of an ECB for which a scheduled label is waiting.

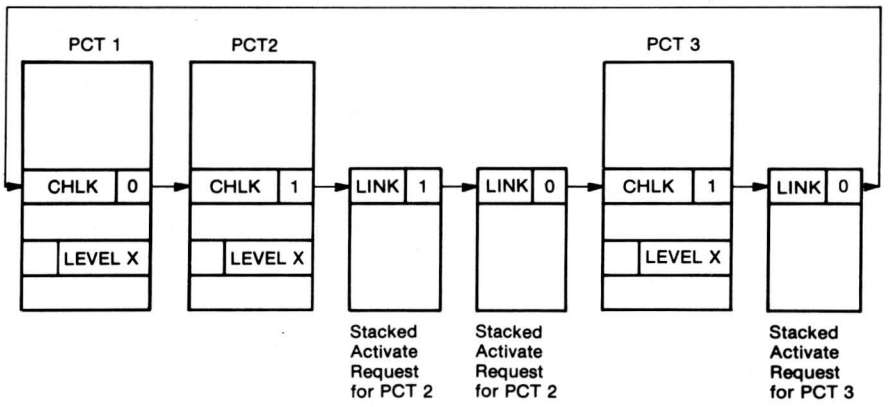
TIMAD: With the BRTM, this word contains only one relevant bit: SA.

It is used to ascertain the existance of a scheduled label save area for this program. It is set to one by the dispatcher when the first label is scheduled and reset when no more label is to be scheduled. This bit remains set as long as a program is running in scheduled label sequence or as long as a scheduled label is recorded in the FILLAB table behind the program save area.

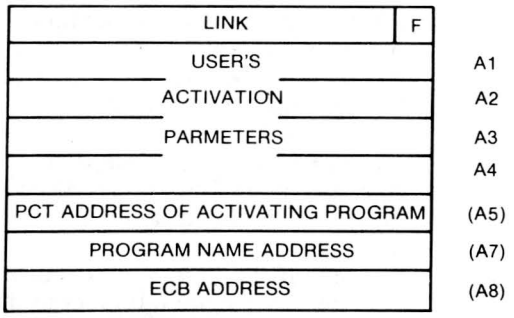
Chaining Link

If, in this word, F=0, it indicates a link to another PCT. If F=1, it is a link to a stacked Activate request.

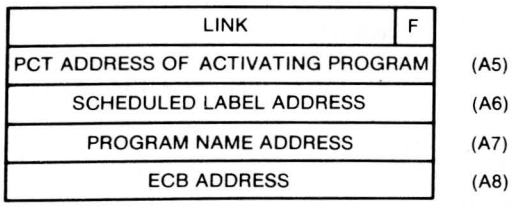
All PCT blocks connected to the same priority level and the stacked Activate requests for them are linked together in a chain as follows:



- PCT's are inserted into the chain each time a monitor request 'Connect to a Level' is given for a program on the same level. If there is only one PCT, the chain loops on itself.
- Activate requests are stacked each time an Activate request is given for a program which is already active: an entry is created (by dynamic memory allocation) and inserted into the PCT chain after the PCT block itself. The entries for these stacked Activate requests may have one of the following formats:
  - Level 49 stacked entry:



  - General stacked activate entry:



These blocks are created by the Activate monitor request handler (M:ACT) and erased by the Exit monitor request handler (M:EXIT).

Software Level Table

This table, for which the entry points and the format are defined in the monitor module T:SYS, is used by the dispatcher to find the current PCT for a given level.

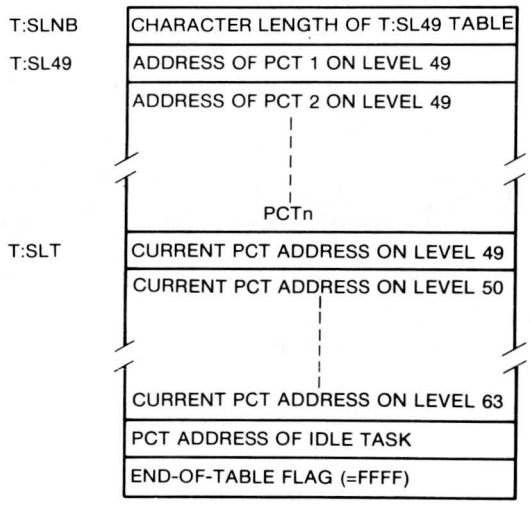
The priority level of a Software Level Program is specified by the position of its PCT-address word in the Software Level Table.

The SLT contains 16 entries:

- one for each of the levels 49 through 63
- one for level 64 (simulated level for the Idle Task).

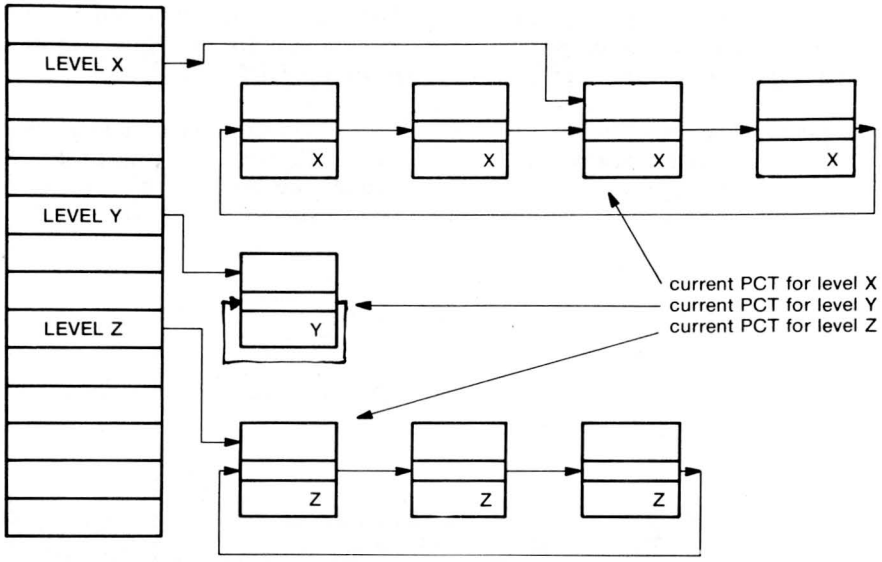
This table is immediately preceded by a table dedicated to level 49 only. It contains the addresses of the PCT's of the programs connected to level 49, listed in their order of priority. For the standard BRTM, only one program runs at level 49: the Timer Control Program M:DCK2.

Note: The multi-PCT chain of links, as defined above, under the PCT description, does not apply to level 49, because these programs run on the basis of their priority, which is defined by their position in the Software Level Table. The table is organized as follows:



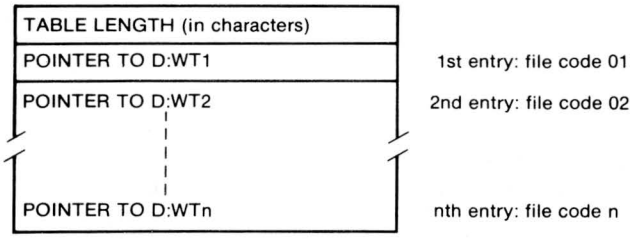
Example of PCT-SLT Organization

The links between SLT and PCT entries are created by the 'Connect a Level' monitor request handler (M:CNLV) and deleted by the 'Disconnect a Level' request handler (M:DNLV), when there is only one PCT in the chain. Within the chain itself the links are changed by the Switch Inside a Level request handler (M:SWTC).



File Code Table

The file code table (FCT module) establishes the connection between a logical file code as used in software, and a physical peripheral device. The table is built at system generation time.
Its layout is as follows:



D:WT1-n are entries in the Device Work Table. For each peripheral one such entry is built, to contain the necessary parameters for the system to handle this device. See below.

Note: All Basic processors (ASM, LKE, FRT, etc.) and the Control Panel Package (INTCP) use file code 05 to refer to the operator's typewriter.

Device Work Table

The DWT is a monitor module, which consists of fixed-length 20-word entries, one entry being built for each peripheral device.
Note that the ASR is considered as consisting of three different devices and, consequently, occupies three entries in the DWT: one each for the typewriter, the tape reader and the tape punch.

An entry in the DWT has the following layout:

ENTRY POINT	0	DEVICE NAME	*
	2	DEVICE ADDRESS	*
	4	BEST RECORD LENGTH	*
	6	DRIVER ADDRESS	*
	8	SOFTWARE STATUS	
	10	ECB ADDRESS	*
	12	CHARACTER OR BUFFER ADDRESS AT INITIALIZATION	
	14	REQUESTED LENGTH	
	16	EFFECTIVE LENGTH	
	18	ORDER	
	20	RETRY BIT (BASIC ORDER)	
	22	WORD TO BE OUTPUT (I/O BUS)	
	24	CHECKSUM (BASIC ORDER) /CONTROL CHAR. FOR LINEPR.	
	26	CHARACTER INDIC. FOR OBJ. WRITE/CONTROL CHAR. FOR LINEPR.	
	28	SAVED A5 (PCT ADDRESS OF REQUESTING PROGRAM)	
	30	SAVED A6: SCHEDULED LABEL (0, IF NONE)	
	32	CONTROLLER STATUS ADDRESS	
	34	ATTACH LOCATION (PCT ADDRESS OF ATTACHED PROGRAM)	*
	36	SST SEQUENCE ADDRESS	*

*: These words should not be modified by any user-written drivers; see chapter 3 of this part)

where:

Word 0 contains the device name, consisting of 2 ASCII characters.
Word 2 contains a binary value giving the physical address of the device.
Word 4 contains the recommended maximum record length for this device.

This number depends on physical device limitations, the multiplex transfer rate and on compatibility with other devices:
typewriter, tape punch, tape reader, card reader: 80.
cassette tape: 256. magnetic tape: 4094. Line printer: 80 or 136.

Word 6 contains the address of the I/O driver belonging to this device.

Word 8 Sofware status, used to record any I/O error during data transfers on the I/O bus or to register the specific basic I/O operation to be performed for an order for Tape Cassette or Magnetic Tape (see Part 1, Chapter 7).

Word 10 contains the address of the user's ECB.

Word 12 contains the user buffer address, or the character address of the next character to be input or output.

Word 14 Requested length of the I/O operation, given by the user.

Word 16 Effective length, used to count the number of characters actually transferred during an I/O operation.

Word 18 The order for the I/O operation, as specified by the user in the A7 register to define the particular I/O function to be performed.

Word 20 is used to record that the user wants the standard recovery feature, even for Basic I/O orders.

Word 22 Next character or word to be output, for devices on the I/O bus.

Word 24 For an object write order this word contains the checksum. For Line Printers, this word contains the last character, saved from the buffer.

Word 26 specifies, in case of 4+4+4+4 object input for devices on the I/O bus, whether the right or left character must be entered. In case of line printer output it is used to save the control code (first two characters of a buffer).

Word 28 is used to save the program PCT address as recorded at initialization time by the IORM module. It is returned to the dispatcher at completion of the I/O operation.

Word 30 is used to save a scheduled label address, as recorded at initialization time by the IORM module. It is returned to the dispatcher at completion of the I/O operation. This word contains 0, if no label has been scheduled.

Word 32 Address of the control unit status word. With every DWT entry, a control unit status word is associated, indicating whether an I/O operation can be initiated or not (busy-not busy).

Word 34 This word is used to record the PCT address of the program which has been attached to this device by means of a monitor request. If this device is not attached, this word contains the hexadecimal value /8000.

Word 36 This word is used to find the SST (sense status) sequence address relative to a device, in case of a throughput error on a device connected to the I/O bus.

T:LKM

This table contains the addresses of the routines which handle the different monitor requests. When a monitor request is made, control first goes to the I:LKM interrupt routine, which then finds the corresponding monitor request handler via this table, in accordance with the DATA number specified in the request.

A table entry set to zero indicates that the corresponding option is not present in the system.

T:LKM 

TABLE LENGTH IN WORDS
M:IORM (monitor request 1)
M:WAIT (monitor request 2)
M:EXIT (monitor request 3)
M:GBUF (monitor request 4)
M:FBUF (monitor request 5)
0
0
0
0
M:CNTM (monitor request 10)
M:DNTM (monitor request 11)
M:ACT (monitor request 12)
M:SWTC (monitor request 13)
M:ATDV (monitor request 14)
M:DTDV (monitor request 15)
0
M:GTIM (monitor request 17)
M:RSEV (monitor request 18)
0
M:CNLV (monitor request 20)
M:DNLV (monitor request 21)
M:WGT (monitor request 22)

REAL TIME CLOCK - TIMERS

The following tables are connected with the management of the real time clock and the software timers. (For timer numbering, see under LkM request 10 in Part 1).

Real Time Block

W:DAY	DAY (ASCII VALUE)
W:MONT	MONTH (ASCII VALUE)
W:YEAR	YEAR (ASCII VALUE)
H:TIME	HOURS (BINARY VALUE -24)
M:TIME	MINUTES (BINARY VALUE -60)
S:TIME	SECONDS (BINARY VALUE -60)
A:TIME	TENTHS OF A SECOND (BINARY VALUE -10)
B:TIME	FIFTIETH OF A SECOND (BINARY VALUE -5)
C:TIME	NON-STANDARD CLOCK

C:TIME contains, if a non-standard clock is included in the system, zero minus the number of non-standard clock cycles during 20 msec.

Chaining pointers for programs connected to timers

H:POIN	Address of first block connected to timer H:TIME
	Address of last block connected to timer H:TIME
M:POIN	Address of first block connected to timer M:TIME
	Address of last block connected to timer M:TIME
S:POIN	Address of first block connected to timer S:TIME
	Address of last block connected to timer S:TIME
A:POIN	Address of first block connected to timer A:TIME
	Address of last block connected to timer A:TIME
B:POIN	Address of first block connected to timer B:TIME
	Address of last block connected to timer B:TIME
C:POIN	Address of first block connected to timer C:TIME
	Address of last block connected to timer C:TIME
R:POIN	Address of first block connected to abs. time HH MM SS
	Address of last block connected to abs. time HH MM SS

V:FLAG
This is a table built by the I:RTC module.
When a chain of programs connected to a timer must be scanned, I:RTC sets a flag which is to be detected by the timer management module M:DCK2 and reset by it.
If one of the values in the table ≠ 0, the chain connected to that timer must be scanned:

V:FLAG	If not zero, scan chain for H:TIME
	M:TIME
	S:TIME
	A:TIME
	B:TIME
	C:TIME

V:RSET
This block contains the values necessary to reset the real time block values.

V:RSET	-24
	-60
	-60
	-10
	-5
	Negative non-standard clock pulse

Blocks built by Connect Program to Timer request (M:CNTM module)

Standard Block

Format **before** first activation of the connected program

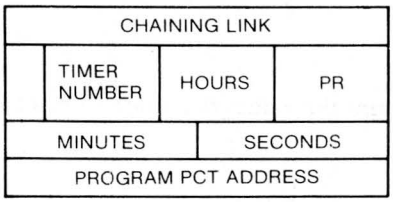
CHAINING LINK
-NC
+PR
PROGRAM PCT ADDRESS

Format **after** the first activation

CHAINING LINK
+PR
-PR
PROGRAM PCT ADDRESS

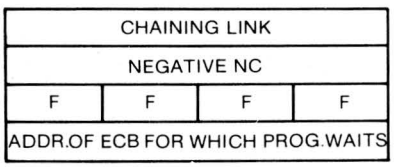
NC: number of timer cycles before the first program activation
PR: number of timer cycles between two activations of the same program.

Block for programs connected to absolute time



T.N. is the timer number.

Block for programs waiting for a given time (M:WGT module)



3 Input/Output

Input/output operations involve a number of monitor modules, aside from the Device Work Table and File Code Table already mentioned in the previous chapter.
The most important of these are the I/O drivers, one for each type of peripheral.
Every I/O monitor request is handled by a driver. The user can, if he wishes, write his own drivers and insert them into the system. Such user-written drivers must interface with certain tables and modules which make up the I/O system of the monitor. The necessary information which must be taken into account is described in the following paragraphs.

TABLES

Three tables are used by the I/O system to know the necessary details about the peripheral devices used:

- File Code Table, which enables the user to assign a file code (a logical number) to a device and use this file code in programming. See chapter 2 of this part.
- Device Work Table: contains all parameters about the peripherals which are necessary for the I/O system to know. See chapter 2 of this part.
- Controller Status Table: one for each device control unit. This is a free format table of unspecified length in which the user can put information for use by his I/O driver. The I/O system knows one word of this table (referred to by word DWT+32) which contains the status of the control unit. The first bit of this status word is reset to 0 (=busy) as soon as an I/O operation for a device is started and set to 1 when it is terminated (=free).

I/O SYSTEM

The I/O system can be divided into four main parts:

- The I/O request module (IORM)
- Driver
- End of I/O module (ENDIO)
- Service routines (COMIO and M:RETR)

IORM

When the user has given an I/O monitor request (LKM 1), this module will

receive control, via the I:LKM module and T:LKM table.
First this module

- checks the validity of the request
- increments the event count (PCT)
- computes the Device Work Table address
- checks whether the device control unit is busy or free (via DWT+32)
  If the unit is busy, the user is put in wait.
- checks whether the device is busy or not. If it is busy, the user is put in wait.

The IORM sets some parameters in the DWT and in the controller status table:

- controller is set to busy (bit 0=1)
- any scheduled label parameters are set in DWT+28, DWT+30. DWT+24 and DWT+26 are set to zero
- the user Event Control Block is initialized, i.e. the left character of ECB 0 and ECB+8 (status) are set to zero
- the I/O order (A7) is analyzed which may result in initialization of some DWT location.

At the end of this process a branch is made to the I/O driver concerned, the address of which is found in DWT+6.

Driver
On entry into a driver, these registers must contain the following parameters:

A7: User I/O order
A8: User ECB address
A6: DWT address. (A4: user order (without wait bit))
The DWT must contain:
Words 0 to 8: not modified.
Word 10: User ECB address
Word 12: User buffer address
Word 14: Requested length (may be non-significant)
Word 16: Zero
Word 18: I/O order (without wait or retry bit; may be non-significant, e.g. skip orders)
Word 20: Retry bit
Word 22: not significant.

The first part of the I/O driver performs the I/O initialization. An Exit to the C:WAIT module is made with the user order (with wait bit!) in A7 and the user ECB address in A8.

The second part of the driver is made up by the interrupt routine:

For P856/7 and with simulation, for P852:
MSR 8, A15

— interrupt sequence generated by SYSGEN:
for single unit controllers:

LDKL A6,DWT address

for multi-unit controllers, e.g. cassette and magnetic tape:

LDKL A6, controller status address

— the exit from the interrupt routine is made to R:TURN (in ENDIO module) if the I/O is not yet finished or to R:TUR4 (in ENDIO) if it is finished. The exit to R:TUR4 must be made with A2 containing the I/O status and A6 containing the DWT address.
— for Retry procedures a branch must be made:
ABL M:RETR
In this case, the calling sequence is:
A6: DWT address
A2: Status to be printed

ENDIO

The calling sequence to be used by user I/O drivers for this module is:
A6: DWT address
A2: I/O software status (see Part 1, chapter 7)
ABL R:TUR4
Moreover, ENDIO must find the following parameters set in the DWT:

— effective length (word 16)
— ECB address (word 10)
— Scheduled label parameters (words 28, 30)

Then, the following functions are performed by the ENDIO module:

— controller status is updated (bit 0 is set to 0:free)
— the event count is decremented
— the event character in the user ECB is updated (bit 0 = 0)
— the effective length is set in the user ECB (word 3)
— the software status is set in the user ECB (word 4)
— a branch is made to the dispatcher (M:DISP), with register A5 containing the PCT address and A6 the scheduled label address, if any.

Service Routines

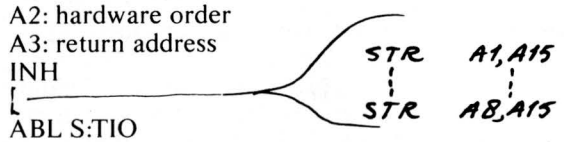
COMIO

This routine has two functions: executing I/O hardware instructions and putting a requesting program in Wait State. Below the calling sequences for the functions are listed.

Note, that in these sequences the instructions INH — STR — ABL must be given in the order specified:

— CIO Start:

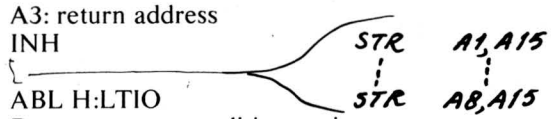
A6: DWT address
A2: hardware order
A3: return address
INH



ABL S:TIO
Return: — condition register set
— inhibit mode.

— CIO Stop:

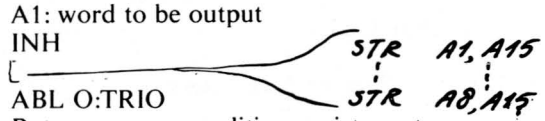
A6: DWT address
A3: return address
INH



ABL H:LTIO
Return: — condition register set
— inhibit mode.

— OTR:

A6: DWT address
A3: Return address
A1: word to be output
INH

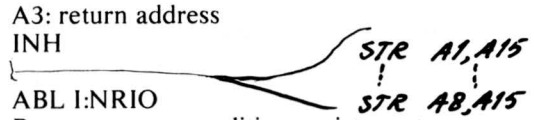


ABL O:TRIO
Return: — condition register set
— inhibit mode.

For P856/7 and, with simulation, for P852:
MSR 8,A15

— INR:

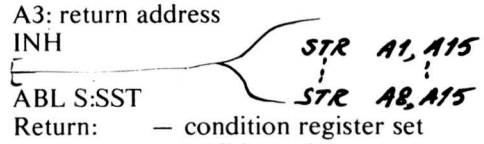
A6: DWT address
A3: return address
INH



ABL I:NRIO
Return: — condition register set
— inhibit mode
— word or character to be input: in A1.

— SST:

A6: DWT address
A3: return address
INH



ABL S:SST
Return: — condition register set
— inhibit mode
— status in A2.

The requesting program is put in Wait, if this is necessary, as follows:

A7: User I/O order
A8: User ECB address
ABL C:WAIT

M:RETR

This module is involved when the hardware status must be returned after an I/O operation. The calling sequence is:

A6: DWT address
A2: Status to be printed
ABL M:RETR

Bit 0 is set to 1 in the status and a branch is made to entry point R:TUR4 in the ENDIO module.

INIMON

Calling Sequence

The monitor initialization progr. is loaded from a tape device by the initial program loader. At the end of this operation, this loader transfers control to INIMON.

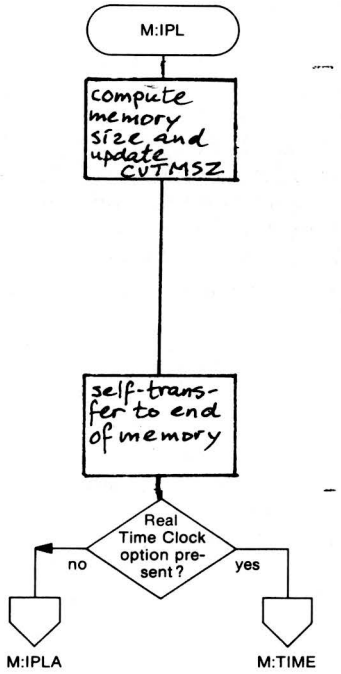
Work Areas and Tables

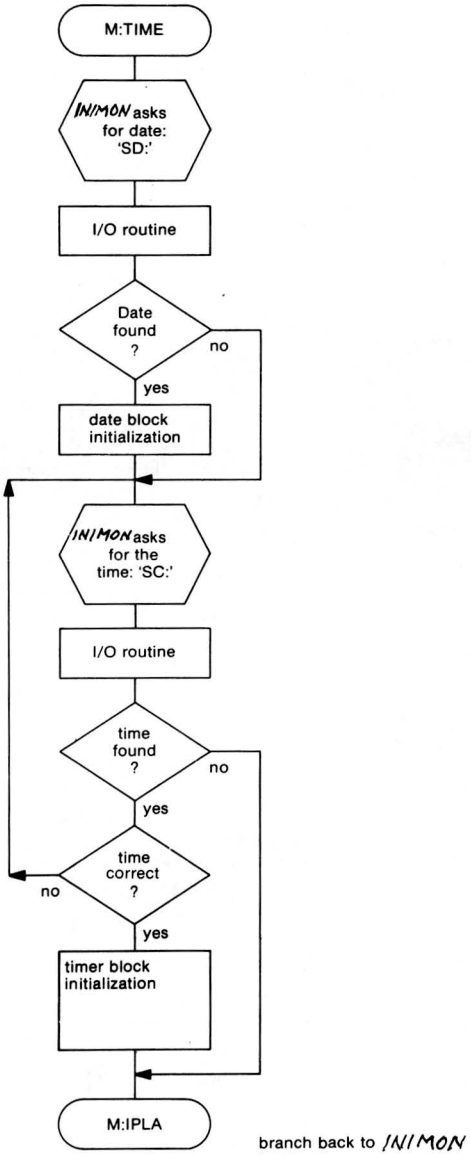
INIMON initializes the following tables and work areas:

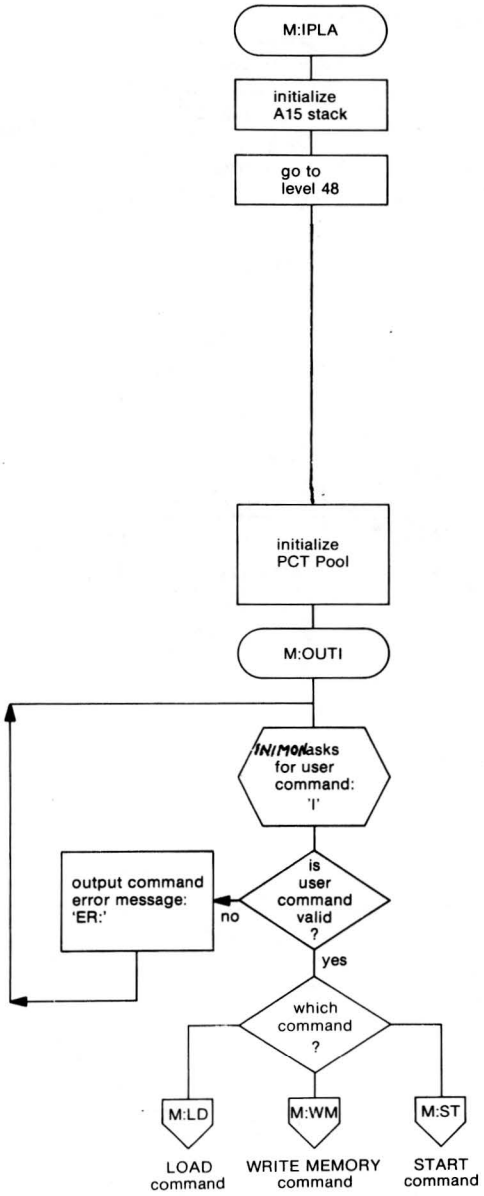
- Timer block (if option is included)
- PCT Pool
- A15 stack
- Dynamic Allocation Area.

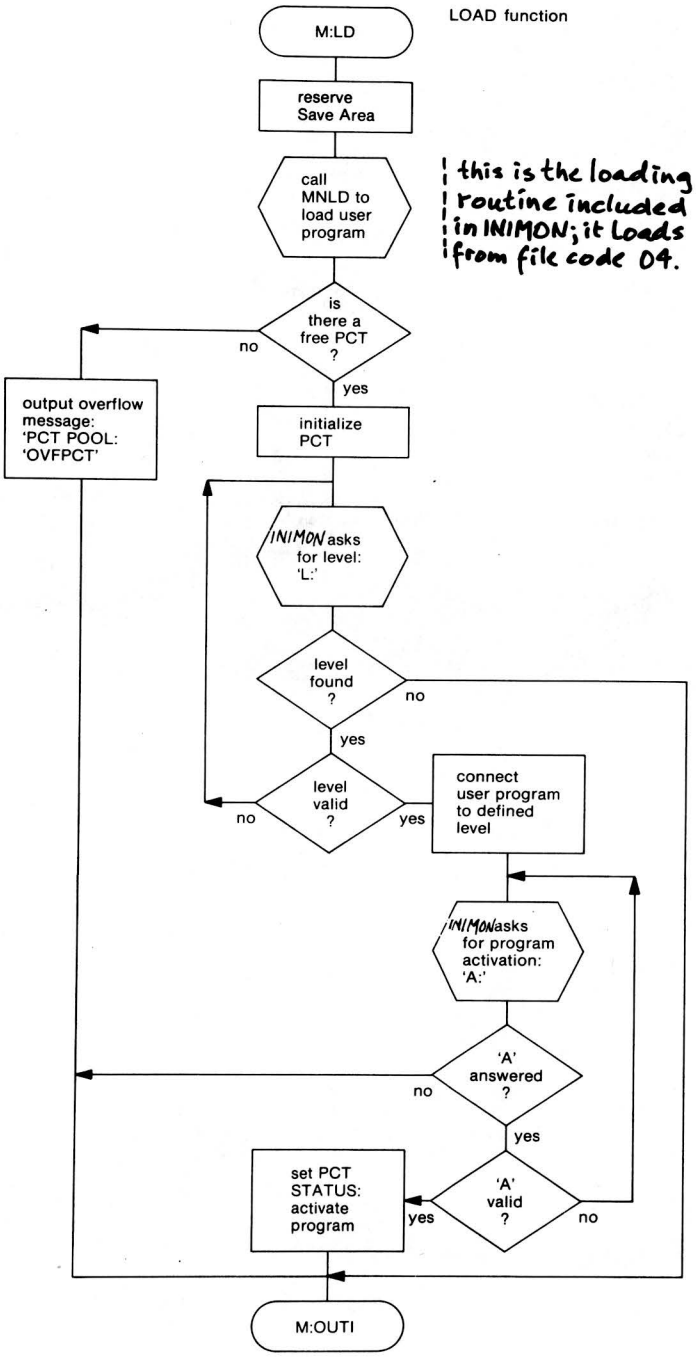
Functional Description

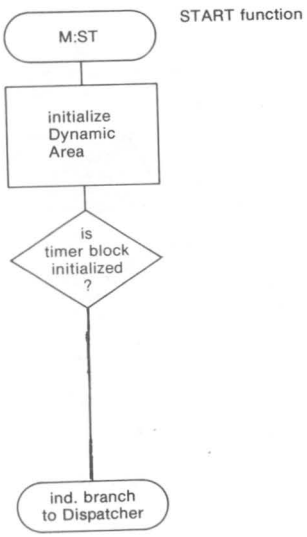
See Part 1, Chapter **9**.




M:TIME
INIMON asks for date: 'SD:'
I/O routine
Date found ?
no
yes
date block initialization
INIMON asks for the time: 'SC:'
I/O routine
time found ?
no
yes
time correct ?
no
yes
timer block initialization
M:IPLA
branch back to INIMON


M:IPLA
initialize A15 stack
go to level 48
initialize PCT Pool
M:OUTI
IN/MON asks for user command: 'I'
is user command valid ?
no
output command error message: 'ER:'
yes
which command ?
M:LD
M:WM
M:ST
LOAD command
WRITE MEMORY command
START command


LOAD function
M:LD
reserve Save Area
call MNLD to load user program
this is the loading routine included in INIMON; it loads from file code 04.
is there a free PCT ?
no
yes
output overflow message: 'PCT POOL: 'OVFPCT'
initialize PCT
INIMON asks for level: 'L:'
level found ?
no
yes
level valid ?
no
yes
connect user program to defined level
INIMON asks for program activation: 'A:'
'A' answered ?
no
yes
'A' valid ?
yes
no
set PCT STATUS: activate program
M:OUTI


START function
M:ST
initialize
Dynamic
Area
is
timer block
initialized
?
ind. branch
to Dispatcher

SYSTEM INTERRUPT ROUTINES

Calling Sequence
These routines are called by a hardware interrupt.

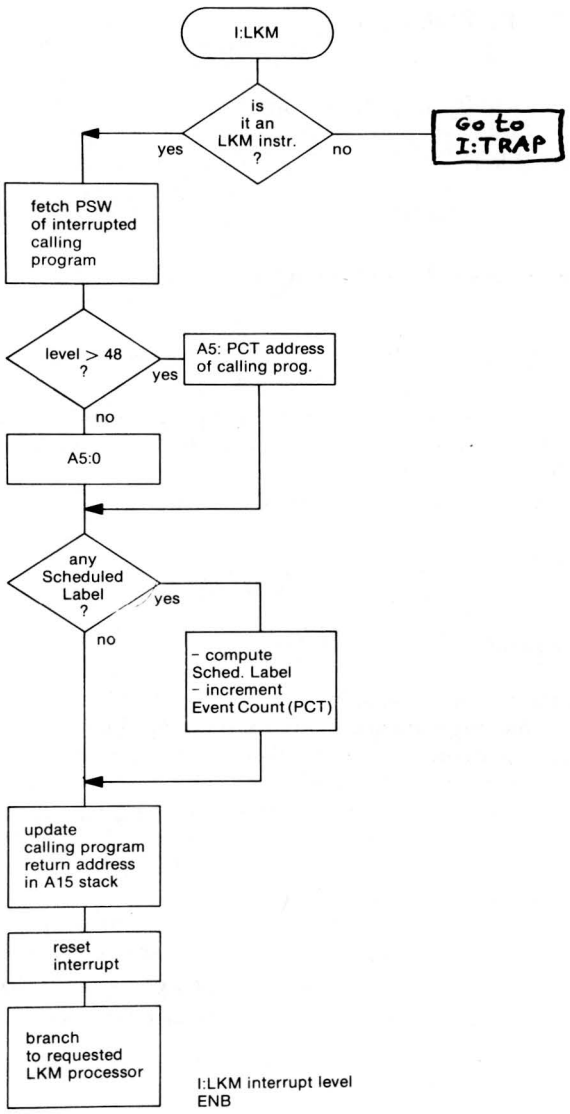
Work Areas and Tables
For:
— I:PFAR, I:ITCP: A15 stack.
— I:LKM: A15 stack
T:LKM (table of LKM modules)

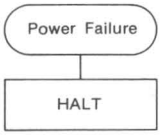
— I:RTC: I:CPLS: word in Communication Vector Table indicating the number of timer cycles during 20 msec.
V:FLAG: flag vector indicating the timer chain to be scanned.
V:RSET: a value vector to reset the timers.
H:TIME: start address of the timer block.
— I/O Routines: Device Work Table (DWT).

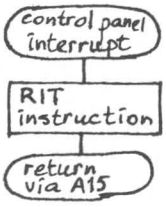
Input/Output Files
None.

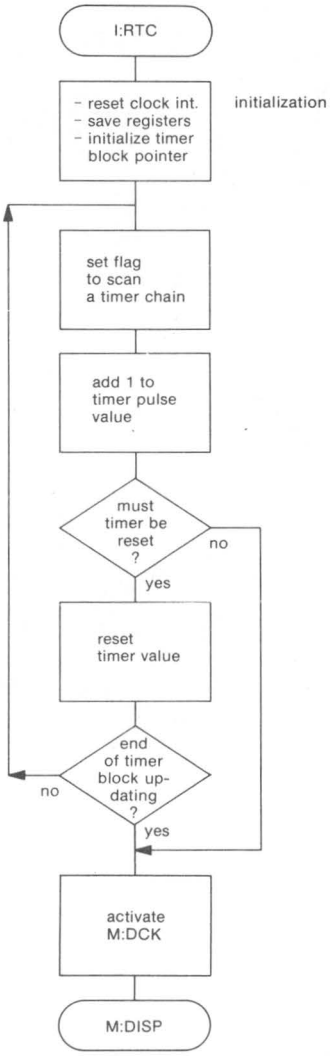
Functional Description

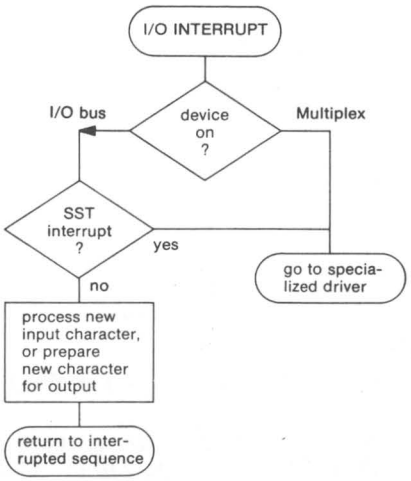
I:LKM processes the monitor requests.
I:MHDL processes interrupts on the common interrupt line.
I/O routines handle input/output routines for specific devices.
I:RTC is the real time clock driver. It is started by a real time clock interrupt and normally runs at level 2. It starts by updating the timer block and setting flags in the V:FLAG vector in case a chain of program blocks connected to a timer must be scanned, analyzed and updated. This task is managed by the module M:DCK which runs at level 49.
Then, I:RTC activates the module M:DCK and gives control to the dispatcher.
I:PFAR issues a HALT when power failure occurs.
I:ITCP handles a control panel interrupt (INT button) by resetting this interrupt and returning via A15.


I:LKM
is it an LKM instr. ?
yes
no
Go to I:TRAP
fetch PSW of interrupted calling program
level > 48 ?
yes
A5: PCT address of calling prog.
no
A5:0
any Scheduled Label ?
yes
no
- compute Sched. Label
- increment Event Count (PCT)
update calling program return address in A15 stack
reset interrupt
branch to requested LKM processor
I:LKM interrupt level ENB


Power Failure
HALT


control panel interrupt
RIT instruction
return via A15


I:RTC
- reset clock int.
- save registers
- initialize timer block pointer
initialization
set flag to scan a timer chain
add 1 to timer pulse value
must timer be reset ?
no
yes
reset timer value
end of timer block up-dating ?
no
yes
activate M:DCK
M:DISP


I/O INTERRUPT
device on ?
I/O bus
Multiplex
SST interrupt ?
yes
no
go to specia-lized driver
process new input character, or prepare new character for output
return to inter-rupted sequence

I/O DRIVERS

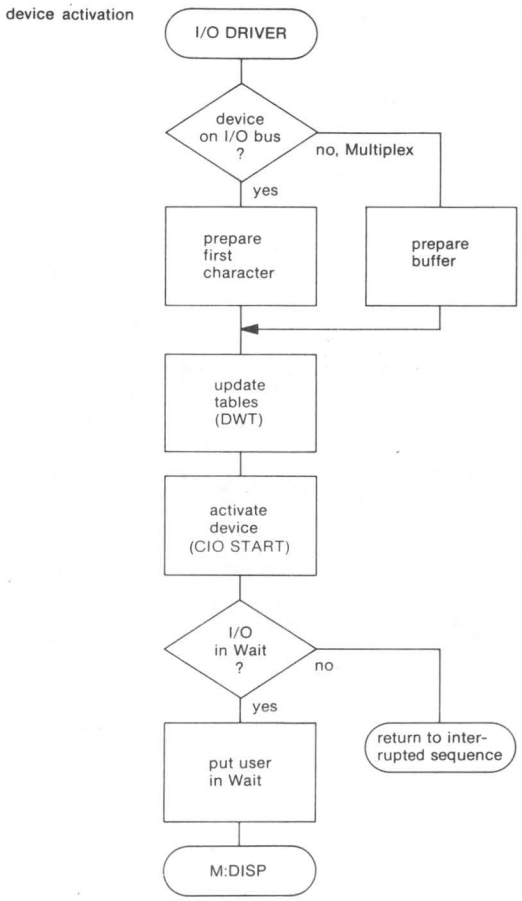
Calling Sequence
A6: DWT address
A4: I/O order

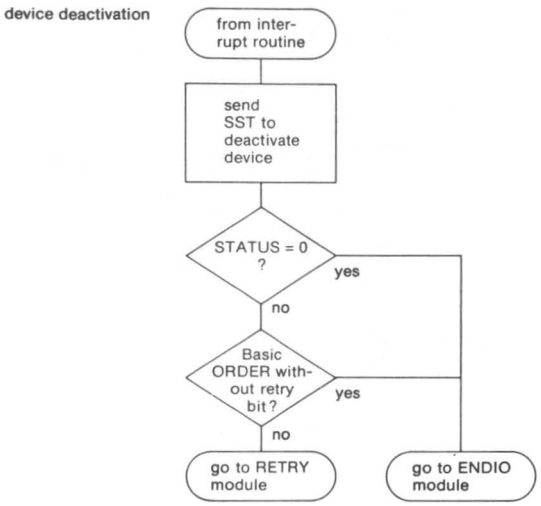
Work Areas and Tables
DWT: Device Work Table

Input/Output Files
None.

Functional Description
Details on the coding of I/O drivers are given in chapter 3 of this Part.

Note: The driver for card reader is a special case: post-editing (conversion from Hollerith to ASCII) is done within the driver after the card has been completely read into an internal buffer. Cards not punched in Hollerith code are not converted: data fault status is set in bit 13 of the Control Unit Status Word.


device activation
I/O DRIVER
device on I/O bus ?
no, Multiplex
yes
prepare first character
prepare buffer
update tables (DWT)
activate device (CIO START)
I/O in Wait ?
no
yes
return to interrupted sequence
put user in Wait
M:DISP


device deactivation
from inter-rupt routine
send SST to deactivate device
STATUS = 0 ?
yes
no
Basic ORDER with-out retry bit?
yes
no
go to RETRY module
go to ENDIO module

M:DISP (Dispatcher)

Calling Sequence

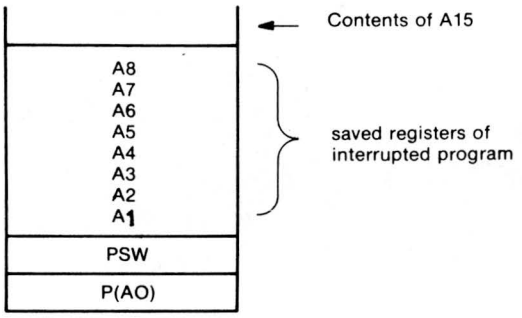
A6: Scheduled Label address, if a scheduled label is to be dispatched. If not: 0.

A5: Address of the PCT entry to which the scheduled label applies. (Irrelevant, if A6 = 0)

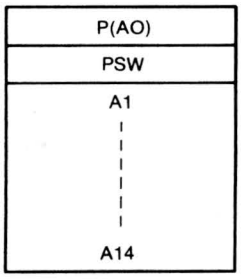
It is assumed that the A15 stack contains P-register, Program Status Word and registers A1 to A8 of the interrupted program.

Work Areas and Tables

A15 stack. Format:



User save area. Format:



User scheduled label save area and table (FILLAB) in case a label is scheduled.

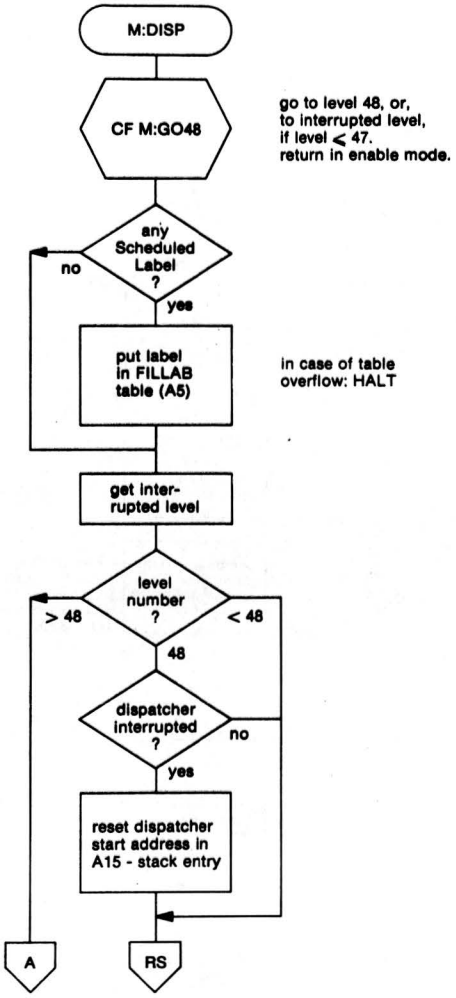
Input/Output Files

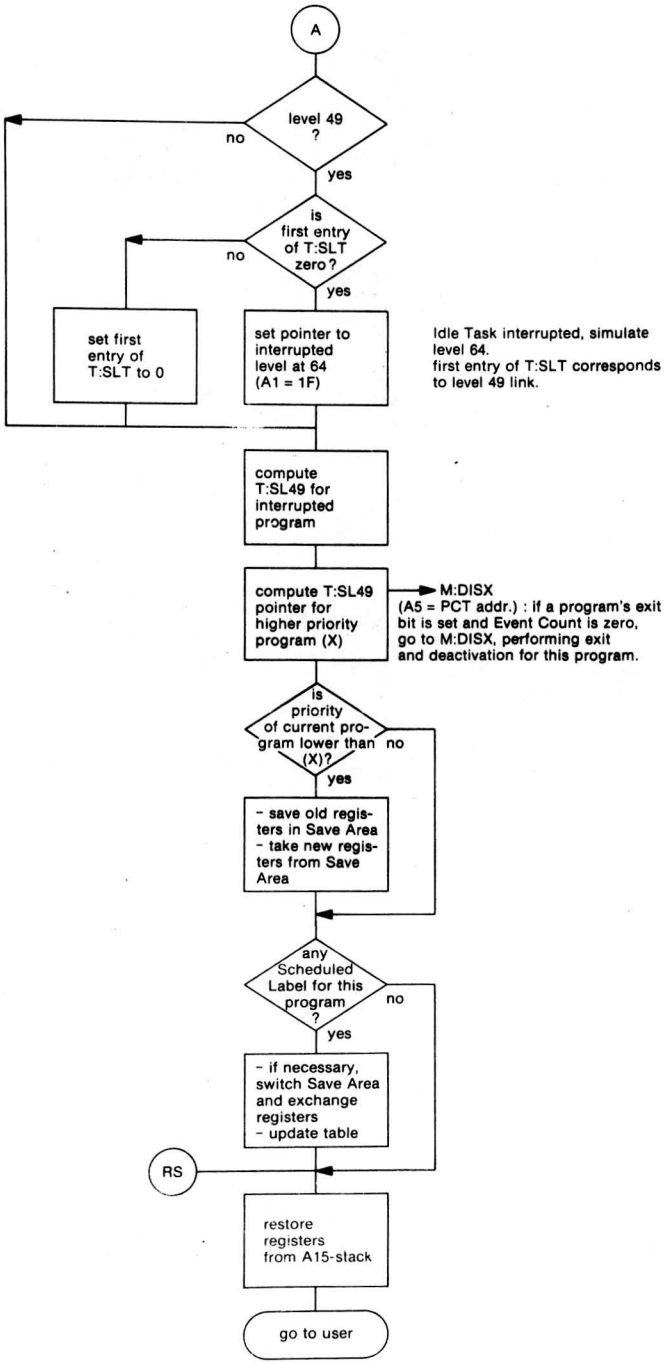
None.

Functional Description

Any system routine can call the dispatcher when it has completed its processing.

The dispatcher then determines which is the next program to be started, by comparing the status of the last interrupted program with the highest-priority program which is in **active** state.
The dispatcher saves the old contents of registers and restores new ones before starting the next program in either a scheduled label sequence or the main program sequence, depending on the state of the scheduled label table (FILLAB).
If no action is to be taken for a user program, control is automatically transferred to the Idle Task.


M:DISP
CF M:GO48
go to level 48, or,
to interrupted level,
if level < 47.
return in enable mode.
any
Scheduled
Label
?
no
yes
put label
in FILLAB
table (A5)
in case of table
overflow: HALT
get inter-
rupted level
level
number
?
> 48
< 48
48
dispatcher
interrupted
?
no
yes
reset dispatcher
start address in
A15 - stack entry
A
RS


A
level 49 ?
no
yes
is first entry of T:SLT zero ?
no
yes
set first entry of T:SLT to 0
set pointer to interrupted level at 64 (A1 = 1F)
Idle Task interrupted, simulate level 64.
first entry of T:SLT corresponds to level 49 link.
compute T:SL49 for interrupted program
compute T:SL49 pointer for higher priority program (X)
M:DISX
(A5 = PCT addr.) : if a program's exit bit is set and Event Count is zero, go to M:DISX, performing exit and deactivation for this program.
is priority of current program lower than (X)?
no
yes
- save old registers in Save Area
- take new registers from Save Area
any Scheduled Label for this program ?
no
yes
- if necessary, switch Save Area and exchange registers
- update table
RS
restore registers from A15-stack
go to user

IDLE TASK

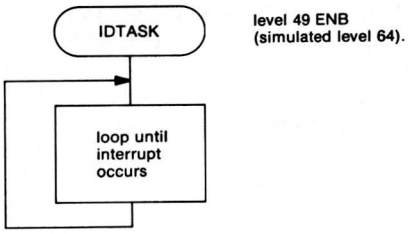
Calling Sequence
None. Register values not significant.

Entry Point: IDTASK

Work Areas and Tables
None.

Input/Output Files
None.

Functional Description
The idle task loop is executed when no other action needs to be taken by the system.
The register values never have any significance for the idle task, because in the Exit monitor request the idle task is simulated. **This loop can therefore not be used as a user program.**



M:IORM (Input/Output – LKM1)

Calling Sequence
A5: PCT address of calling program
A6: Scheduled Label
A7: I/O ORDER
A8: ECB address.

Entry Point: M:IORM

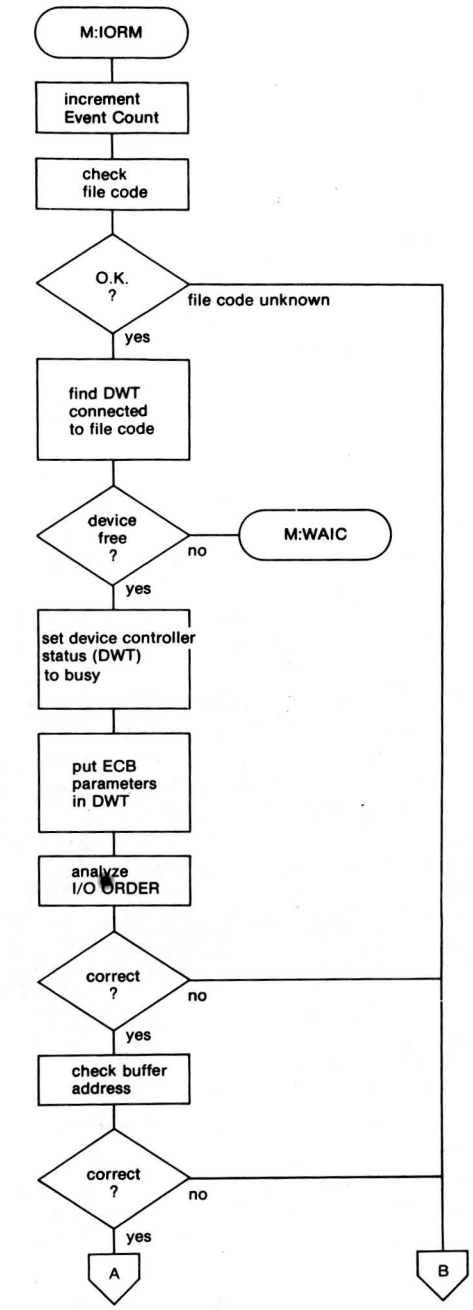
Work Areas and Tables
FCT (File Code Table)
DWT (Device Work Table)

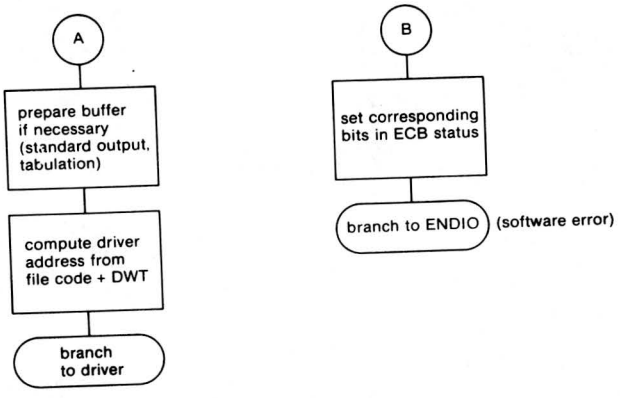
Input/Output Files
None.

Functional Description
This module, entered from the I:LKM module, performs all initializations for an I/O operation:

- it checks the validity of the request parameters;
- formats a buffer, according to the I/O order specified in register A7 (removal of blanks, tabulation check);
- computes the address of the Device Work Table (DWT) and stores the necessary parameters in this table.

Then it branches to the entry specific to the requested I/O operation.


M:IORM
increment Event Count
check file code
O.K. ?
file code unknown
yes
find DWT connected to file code
device free ?
no
M:WAIC
yes
set device controller status (DWT) to busy
put ECB parameters in DWT
analyze I/O ORDER
correct ?
no
yes
check buffer address
correct ?
no
yes
A
B


A
prepare buffer if necessary (standard output, tabulation)
compute driver address from file code + DWT
branch to driver
B
set corresponding bits in ECB status
branch to ENDIO
(software error)

ENDIO

Calling Sequence
No parameters.

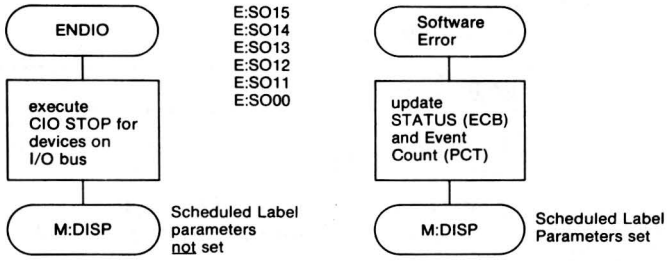
Entry Points: 11 entry points to this module, dependant on the function requested.
E:S015, E:S014, E:S013, E:S012, E:S011, E:S000 for return with error, to update the ECB status word and branch to the dispatcher.
E:NDIO, to execute a CIO stop of a device on the I/O bus and then return to the user.
R:TUR2, to enter a post-edit sequence at the end of I/O.
R:TUR4, to end the I/O operation without post-editing (A2: STATUS).

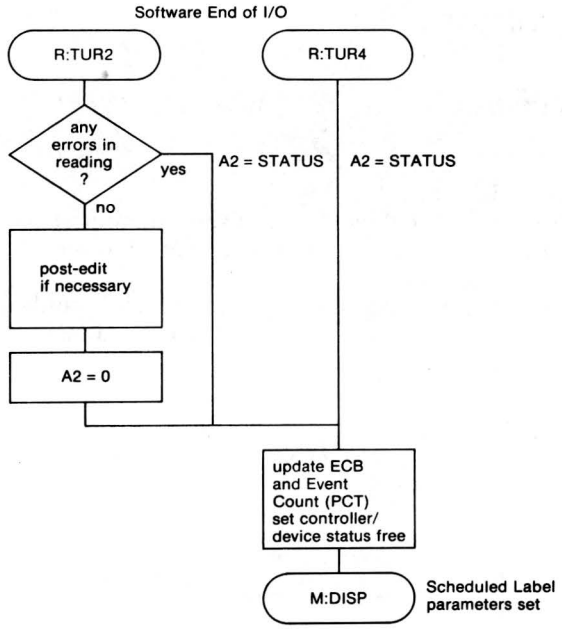
Work Areas and Tables
DWT: Device Work Table.

Input/Output Files
None.

Functional Description
This module consists of four parts:

- ENDIO performs a CIO stop for devices connected to the I/O bus and then returns to the interrupted program.
- Another part handles software errors, updating the ECB Status word and event character in accordance with the error code (C00X).
- The parts entered at entry points R:TUR2 and R:TUR4 perform a normal return:
  - via R:TUR2 to a post-edit process:
    blanks are filled in, in case of a standard input order;
    reading is continued in case of skip orders;
    the Status word in the ECB is set, according to any errors which may have occurred.
  - via R:TUR4 without post-editing:
    Status word, event character and event count are updated.
  - Return to the dispatcher, with scheduled label parameters set.


ENDIO
execute CIO STOP for devices on I/O bus
M:DISP
Scheduled Label parameters not set
E:SO15
E:SO14
E:SO13
E:SO12
E:SO11
E:SO00
Software Error
update STATUS (ECB) and Event Count (PCT)
M:DISP
Scheduled Label Parameters set


Software End of I/O
R:TUR2
R:TUR4
any errors in reading ?
yes
no
A2 = STATUS
A2 = STATUS
post-edit if necessary
A2 = 0
update ECB and Event Count (PCT) set controller/ device status free
M:DISP
Scheduled Label parameters set

M:WAIT (Wait for an Event — LKM2)

Calling Sequence
A5: PCT address
A6: Scheduled Label
A8: Event Control Block address

ECB format:

X

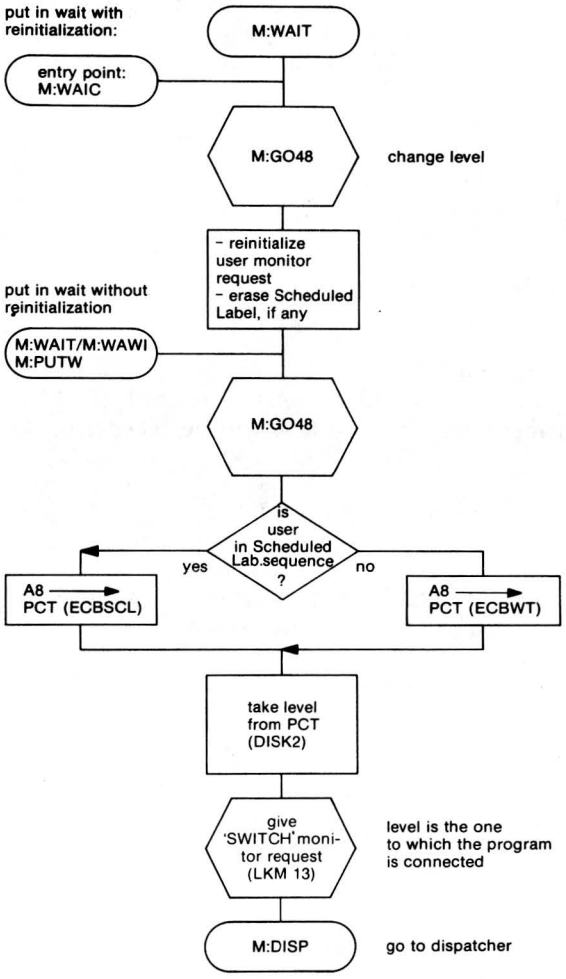
X = 1: event has occurred.

Entry Points:
- M:WAIT; M:PUTW; M:WAWI
  (wait without re-initialization of user request)
- M:WAIC
  (wait with reinitialization of user request)

Work Areas and Tables
None.

Input/Output Files
None.

Functional Description
This module processes the *Wait for an Event* monitor request (LKM2). There are different entry points to this module, depending on whether the request must or must not be reinitialized.
A wait is done on the occurrence of an event which is notified by the setting of bit 0 of the ECB word of the program or scheduled label in which the event occurs.


put in wait with reinitialization:
M:WAIT
entry point: M:WAIC
M:GO48
change level
- reinitialize user monitor request
- erase Scheduled Label, if any
put in wait without reinitialization
M:WAIT/M:WAWI M:PUTW
M:GO48
is user in Scheduled Lab.sequence ?
yes
no
A8 → PCT (ECBSCL)
A8 → PCT (ECBWT)
take level from PCT (DISK2)
give 'SWITCH' monitor request (LKM 13)
level is the one to which the program is connected
M:DISP
go to dispatcher

M:EXIT (Exit – LKM3)

Calling Sequence
A5: PCT address of program requesting exit.

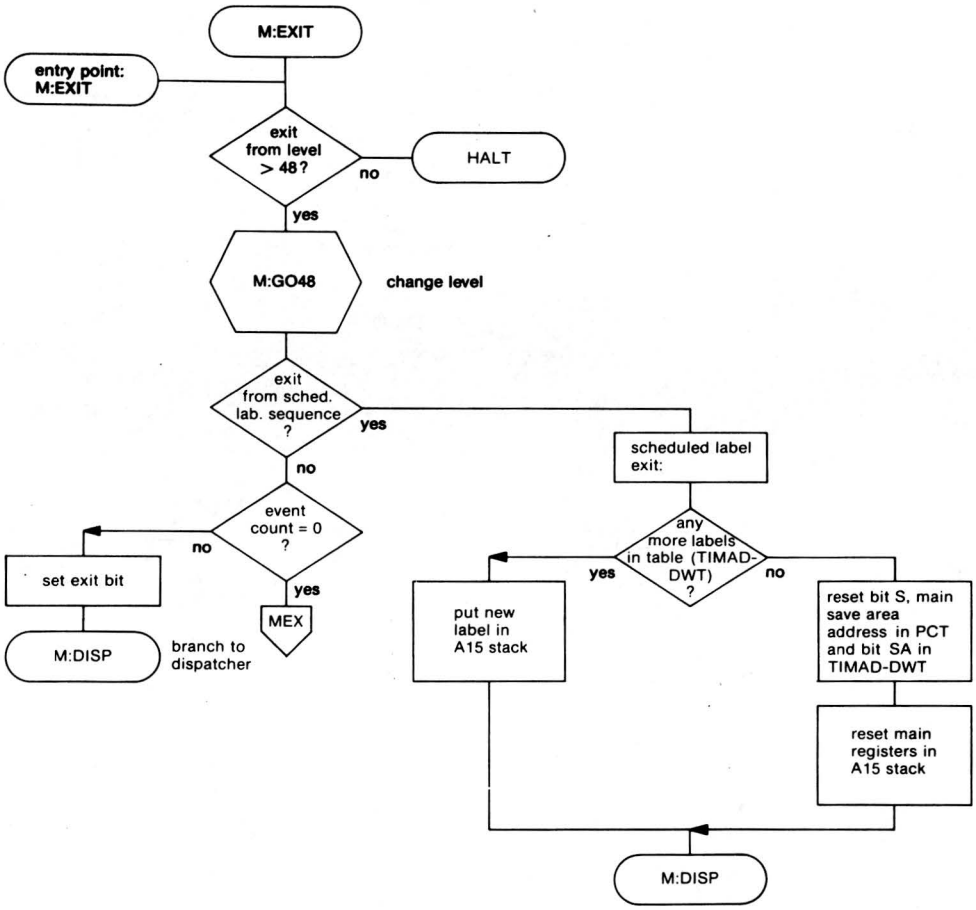
Entry Points:
- M:EXIT (entry point from I:LKM: user request)
- M:DISX (entry point from dispatcher; used when exit was delayed, because event count was not zero yet).

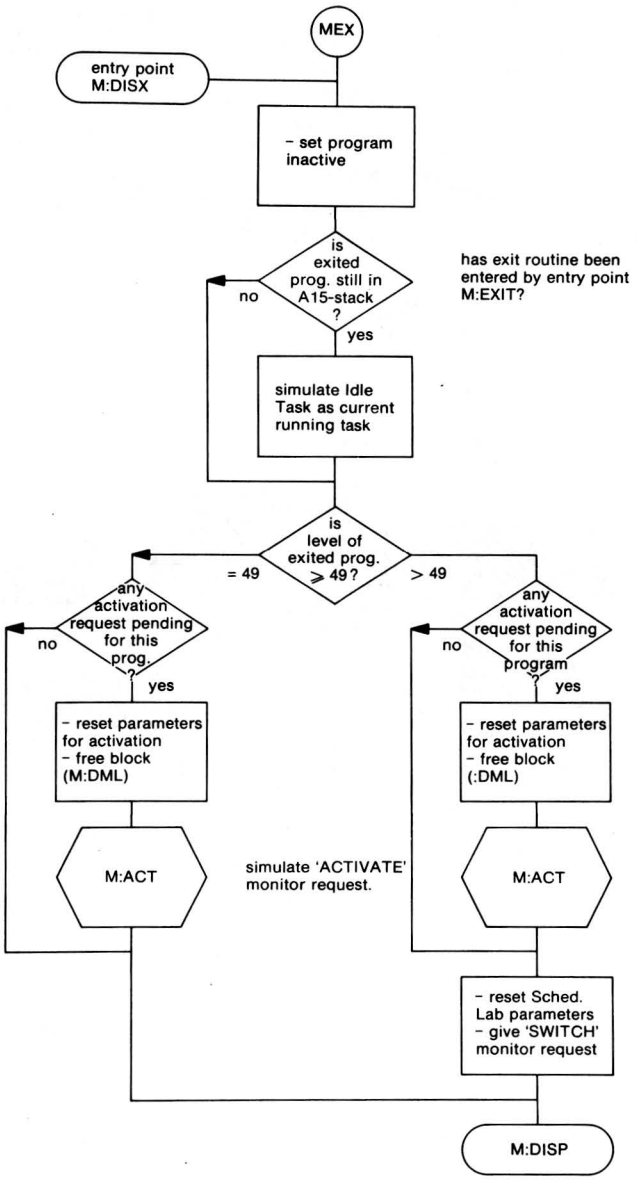
Work Areas and Tables
SLT (Software Level Table).
PCT entry of program requesting exit.
Save area of this program.
Dynamic memory allocation area.

Input/Output Files
None.

Functional Description
This module processes the *Exit* monitor request (LKM3). If all I/O is terminated and there are no more labels to be scheduled, the main program exits:

- the program is put in inactive state;
- the PCT chain is checked to see if there are any stacked activate requests for this program, in which case it is reactivated.
  If the program is not yet finished, the exit bit is set in PCT word 0 (Status) and control is given to the dispatcher. When the event count becomes zero, the dispatcher returns control to the M:EXIT module (via M:DISX) and the main program exit is performed.


M:EXIT
entry point:
M:EXIT
exit
from level
> 48?
no
HALT
yes
M:GO48
change level
exit
from sched.
lab. sequence
?
yes
no
scheduled label
exit:
event
count = 0
?
no
yes
any
more labels
in table (TIMAD-
DWT)
?
yes
no
set exit bit
MEX
put new
label in
A15 stack
reset bit S, main
save area
address in PCT
and bit SA in
TIMAD-DWT
M:DISP
branch to
dispatcher
reset main
registers in
A15 stack
M:DISP


MEX
entry point M:DISX
- set program inactive
is exited prog. still in A15-stack ?
no
yes
has exit routine been entered by entry point M:EXIT?
simulate Idle Task as current running task
is level of exited prog. ⩾ 49 ?
= 49
> 49
any activation request pending for this prog. ?
no
yes
- reset parameters for activation
- free block (M:DML)
M:ACT
simulate 'ACTIVATE' monitor request.
any activation request pending for this program ?
no
yes
- reset parameters for activation
- free block (:DML)
M:ACT
- reset Sched. Lab parameters
- give 'SWITCH' monitor request
M:DISP

M:GBUF, M:FBUF (Get Buffer – LKM4/Release Buffer – LKM5)

Calling Sequence

– M:GBUF:

A7: Length of requested buffer, in characters.
If zero is specified, the memory size in characters is returned in the A7 register (If memory size = 32k, 0 is returned).

Entry Point: M:GBUF

– M:FBUF:

A14: address of the buffer previously reserved for the calling program by a Get Buffer request.

Entry Point: M:FBUF

Work Areas and Tables

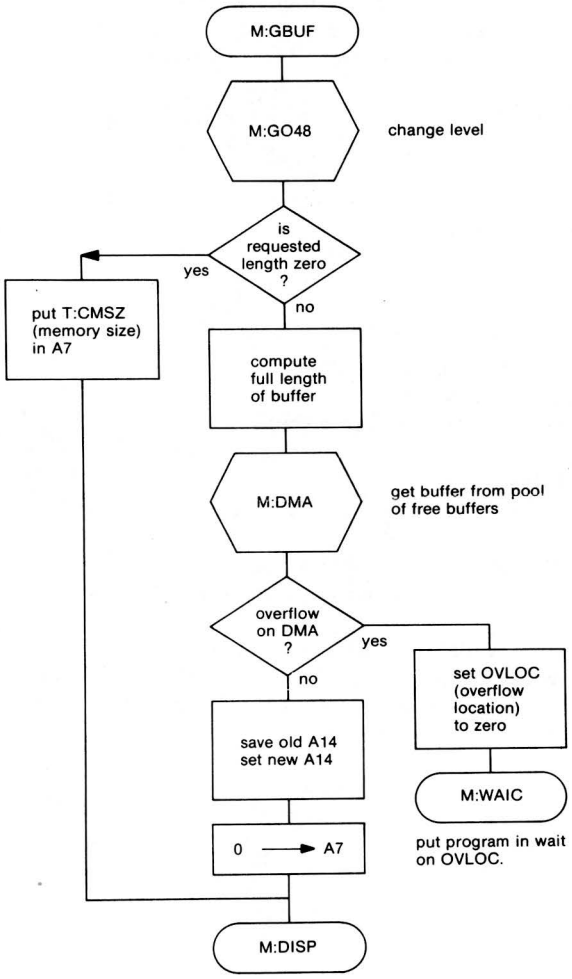
T:CVT: Communication Vector Table.
PCT entry of calling program.
Dynamically Allocated Memory Area (buffer).

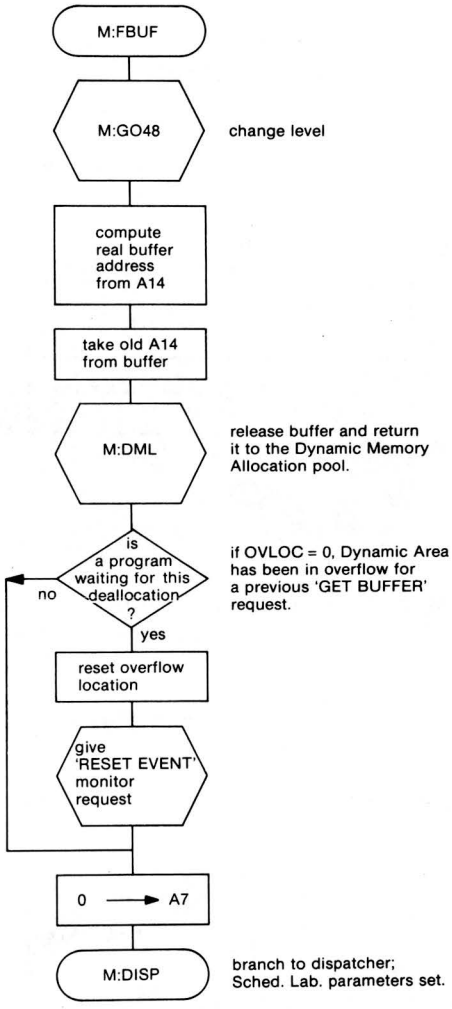
Input/Output Files

None.

Functional Description

These modules handle the dynamic allocation of memory space for user programs.
If there is no space available when a request is made, the user program is put in wait state (with reinitialization of the request). When memory space becomes available again, i.e. after a Release Buffer request has been given, the user program is restarted to repeat its request for a buffer.


M:GBUF
M:GO48
change level
is requested length zero ?
yes
no
put T:CMSZ (memory size) in A7
compute full length of buffer
M:DMA
get buffer from pool of free buffers
overflow on DMA ?
yes
no
set OVLOC (overflow location) to zero
M:WAIC
put program in wait on OVLOC.
save old A14 set new A14
0 → A7
M:DISP

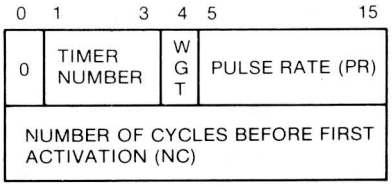

M:FBUF
M:GO48
change level
compute real buffer address from A14
take old A14 from buffer
M:DML
release buffer and return it to the Dynamic Memory Allocation pool.
is a program waiting for this deallocation ?
no
yes
if OVLOC = 0, Dynamic Area has been in overflow for a previous 'GET BUFFER' request.
reset overflow location
give 'RESET EVENT' monitor request
0 → A7
M:DISP
branch to dispatcher; Sched. Lab. parameters set.

M:CNTM (connect a Program to a Timer — LKM10)

Calling Sequence

Entry Point: M:CNTM
A7: address of program name block.
A8: address of 2-word parameter block, which may be of one of the following two formats:

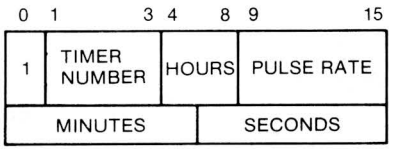
Standard Connection:



where

TIMER NUMBER is the timer to which the program specified via A7 must be connected.
PR is a value from 0 to 2047.
NC is a value from 0 to 32677.
Bit 4 is set to 0 by the system (calling level ≠ 48) and its use is reserved to the 'Wait for a given Time' module (M:WGT).

Absolute Time Connection:



where

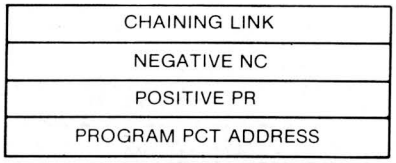
PR is a value from 0 to 127.
The program specified via A7 is connected to the absolute time chain. At the time defined by the user (HH MM SS), it is started, disconnected from this

chain and connected to the chain on the timer defined by the user in bits 1 to 3. This is managed by the M:DCK module.

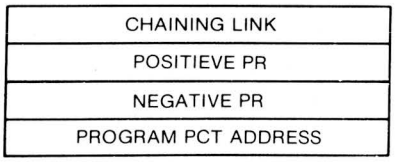
Note: When PR = 0, only one program activation takes place and the program is automatically disconnected from the timer. This is managed by the M:DCK module.

Work Areas and Tables
H:POIN Chain Pointer
When a connection is requested from M:CNTM, a 4-word block is automatically reserved in the dynamic allocation area. The format of such a block is as follows:

— *Standard Connection*

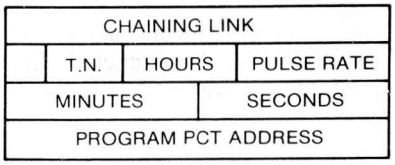


After the first program activation, the block format is as follows (unless PR was 0, in which case automatic disconnection will follow):



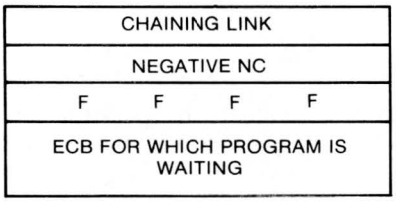
where the third word contains the PR value as updated by M:DCK.
The second word contains the PR value needed to reset the third word when it equals 0 (= program activation).

— *Absolute Time Connection*



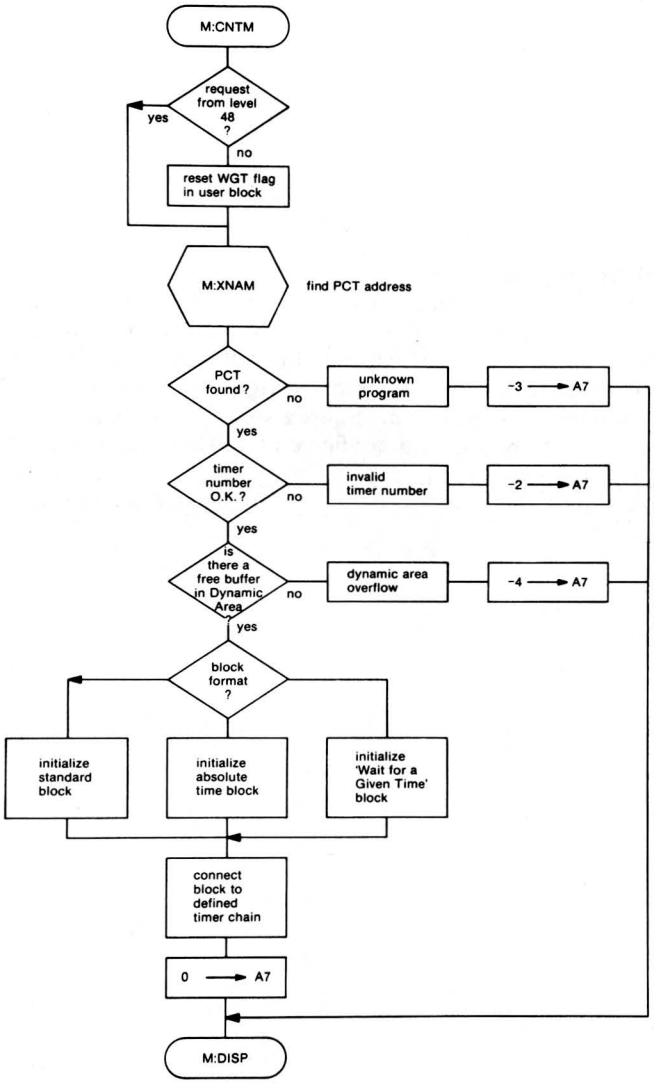
After the first program activation, this block is reinitialized in standard format.

— *'Wait for a Given Time' Connection*
The following block is built after a system request at level 48, by the M:WGT module:



Input/Output Files
None.

Functional Description
This request builds and initializes a timer block, to establish a link between a timer and the calling program.
First, the module checks the level of the request in order to set or reset the WGT flag in the two-word parameter block: for user levels, this flag is always zero.
Then the program name and timer number are checked.
If an error is detected at this point, it is set in the A7 register. M:CNTM then gives a request to the M:DMA module for a 4-word block in the dynamic allocation area. If this request is not accepted (dynamic area overflow), the value −4 is loaded into the A7 register. Otherwise, all conditions are fulfilled for building a timer block. This block is then initialized and connected to the timer defined by the calling program.


M:CNTM
request from level 48 ?
yes
no
reset WGT flag in user block
M:XNAM
find PCT address
PCT found?
no
unknown program
-3 → A7
yes
timer number O.K.?
no
invalid timer number
-2 → A7
yes
is there a free buffer in Dynamic Area
no
dynamic area overflow
-4 → A7
yes
block format ?
initialize standard block
initialize absolute time block
initialize 'Wait for a Given Time' block
connect block to defined timer chain
0 → A7
M:DISP

M:DNTM (Disconnect a Program from a timer — LKM11)

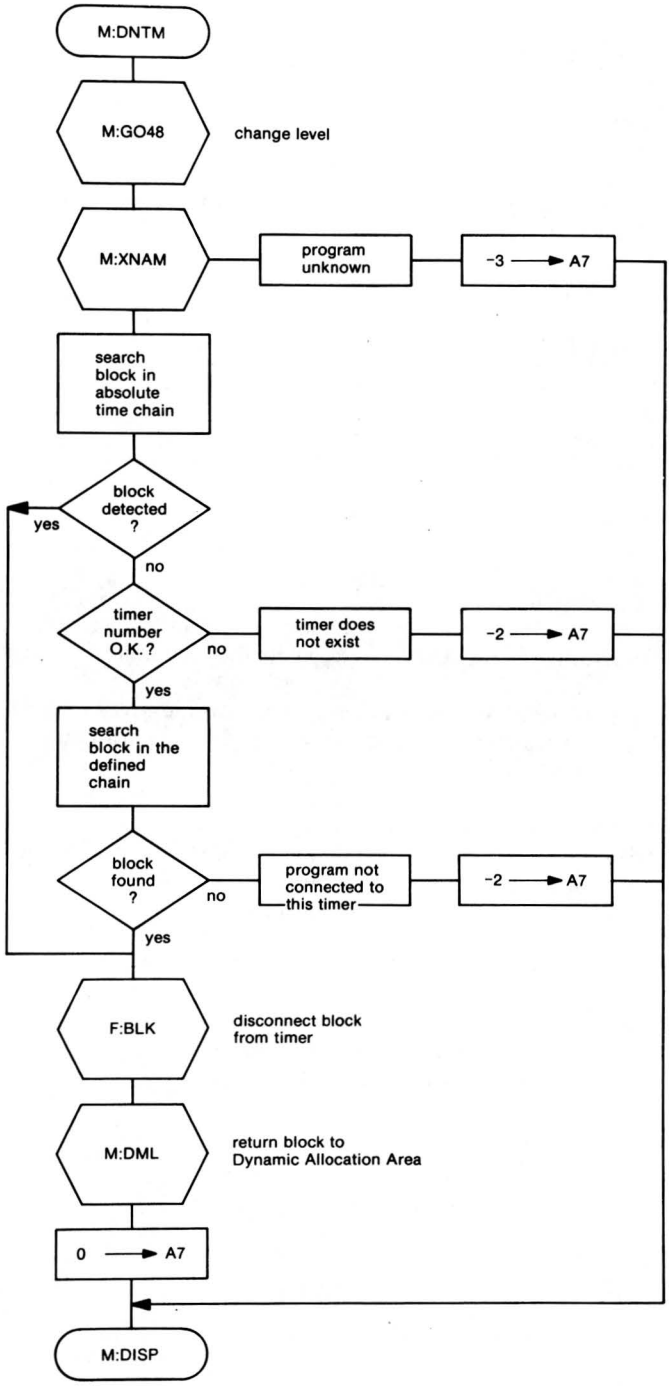
Calling Sequence
A7: address of program name block
A8: timer number.

Entry Point: M:DNTM

Work Areas and Tables
H:POIN: Chain pointer

Input/Output Files
None.

Functional Description
This module disconnects a timer block from a chain. First, the address is found of the PCT entry for this program. If the M:XNAM module cannot find this address, the program does not exist. If the address is found, M:DNTM scans the absolute time chain to look for the program's timer block. If no block is found here, the timer chain specified in register A8 is checked.
When the block is found, it is disconnected from the chain and returned to the dynamic allocation area.
If an error is found, this is indicated in the A7 register of the calling program.


M:DNTM
M:GO48
change level
M:XNAM
program unknown
-3 → A7
search block in absolute time chain
block detected ?
yes
no
timer number O.K. ?
no
timer does not exist
-2 → A7
yes
search block in the defined chain
block found ?
no
program not connected to this timer
-2 → A7
yes
F:BLK
disconnect block from timer
M:DML
return block to Dynamic Allocation Area
0 → A7
M:DISP

M:ACT (Activate a Program — LKM12)

Calling Sequence
A5: PCT-address of activating program
A6: Scheduled Label
A7: Address of program name block of program to be activated
A8: ECB-address of calling program.

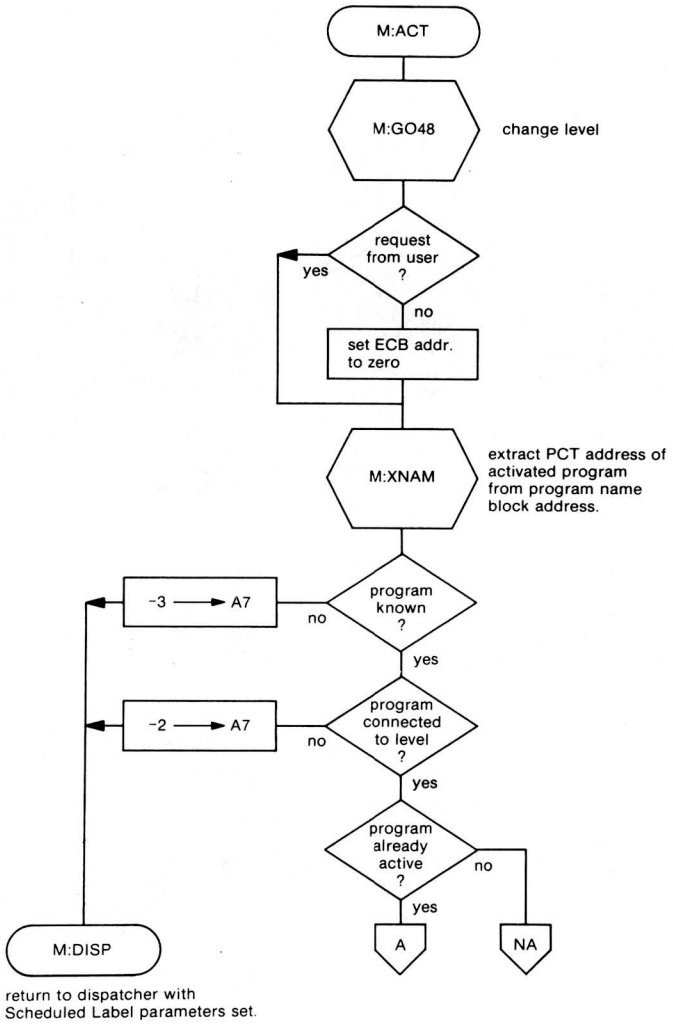
Entry Point: M:ACT

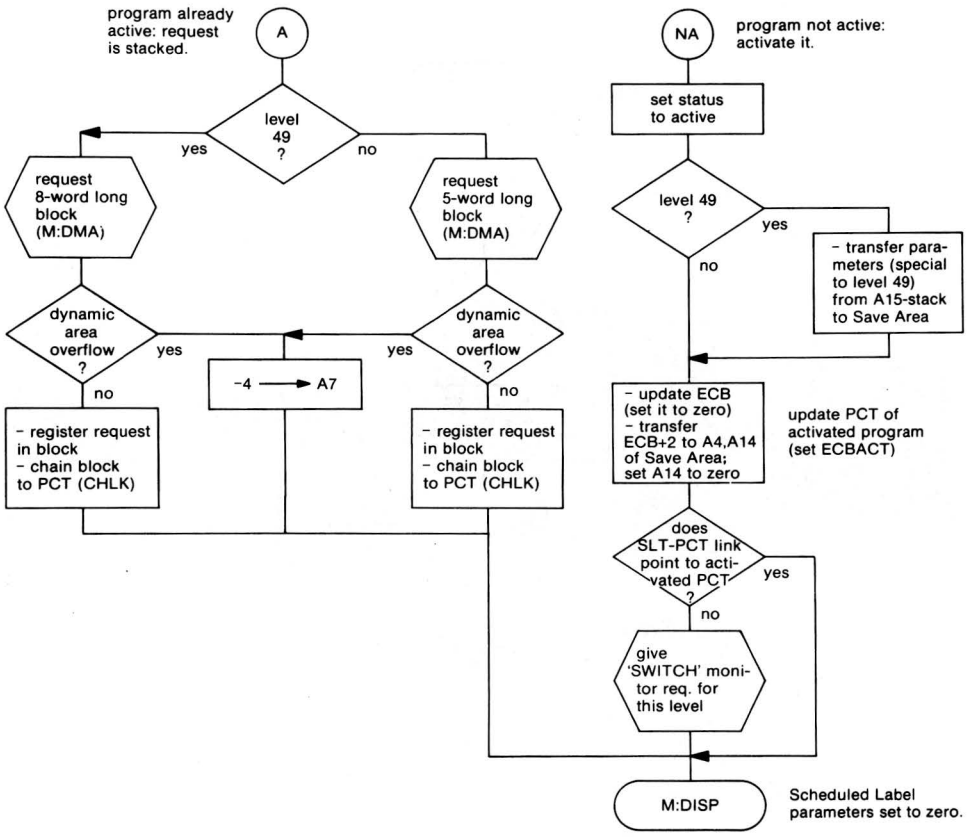
Work Areas and Tables
SLT (Software Level Table).
PCT entry of activated program.
Save area of activated program.
Dynamic memory allocation area.

Input/Output Files
None.

Functional Description
This module will activate a program, unless it is in active state already, in which case the Activate request is stacked in the dynamic memory allocation area.
If the request is for level 49, the user or system routine gives its first 3 registers to the level 49 program, when it is activated. These parameters are stacked together with the request.
If the request is for another level, only the parameters of the request are stacked (registers A5 to A8).
When the program has been activated, the request for level 49 is handled in the same way as a request for another level, after the special parameters for level 49 have been transmitted to the save area:

- user parameter (word 2 of ECB) is transmitted to registers A4 and A14 of the activated program.
- Status is set to active.
- 'Switch' request is given for the requested level, to ensure that the program will be started by the dispatcher when it gets priority.


M:ACT
M:GO48
change level
request from user ?
yes
no
set ECB addr. to zero
M:XNAM
extract PCT address of activated program from program name block address.
program known ?
-3 → A7
no
yes
program connected to level ?
-2 → A7
no
yes
program already active ?
no
yes
M:DISP
A
NA
return to dispatcher with Scheduled Label parameters set.


program already active: request is stacked.
A
level 49 ?
yes
no
request 8-word long block (M:DMA)
request 5-word long block (M:DMA)
dynamic area overflow ?
yes
no
-4 → A7
dynamic area overflow ?
yes
no
- register request in block
- chain block to PCT (CHLK)
- register request in block
- chain block to PCT (CHLK)
NA
program not active: activate it.
set status to active
level 49 ?
yes
no
- transfer parameters (special to level 49) from A15-stack to Save Area
- update ECB (set it to zero)
- transfer ECB+2 to A4,A14 of Save Area; set A14 to zero
update PCT of activated program (set ECBACT)
does SLT-PCT link point to activated PCT ?
yes
no
give 'SWITCH' monitor req. for this level
M:DISP
Scheduled Label parameters set to zero.

M:SWTC (Switch Inside a Software Level — LKM13)

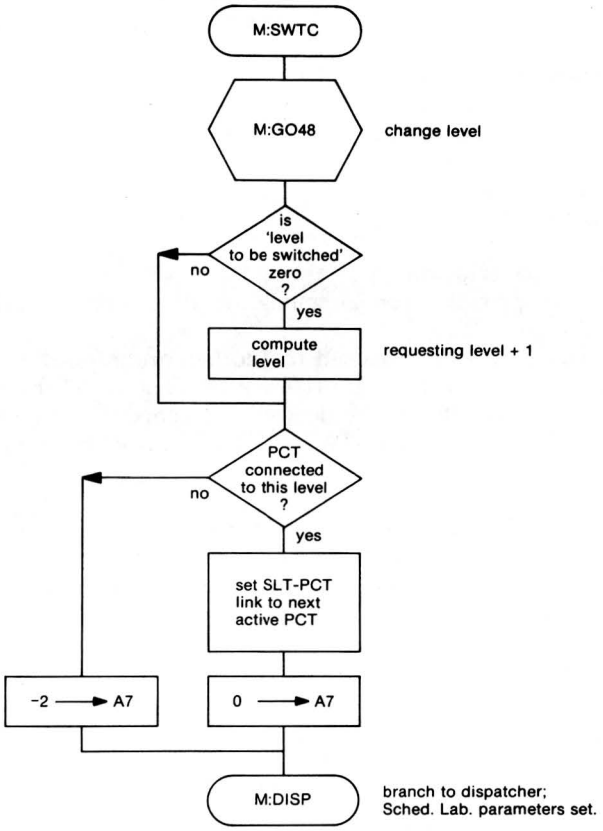
Calling Sequence
A5: PCT address of calling program
A6: Scheduled Label
A7: Level to be switched. If this is zero, the level to be switched is equal to the level of the calling program +1.

Entry Point: M:SWTC

Work Areas and Tables
SLT (Software Level Table).
PCT (Program Control Table).

Input/Output Files
None.

Functional Description
There may be several PCT entries connected to one level. A link in the Software Level Table points to one of them. When this request is given, the link will be changed to the next active PCT entry in the chain, which is not in wait state.
This request may be given by user as well as system programs. From the system, this request is given by the *Wait, Activate, Exit* and *Set an Event* modules.


M:SWTC
M:GO48
change level
is 'level to be switched' zero ?
no
yes
compute level
requesting level + 1
PCT connected to this level ?
no
yes
set SLT-PCT link to next active PCT
-2 → A7
0 → A7
M:DISP
branch to dispatcher; Sched. Lab. parameters set.

M:^TDT (Attach/Detach a Device to/from a Program — LKM 14+15)

Calling Sequence
A5: PCT-address of calling program
A6: Scheduled Label
A7: Wait Flag (=0: no wait; ≠ 0: wait)
A8: Address of ECB containing the related file code.

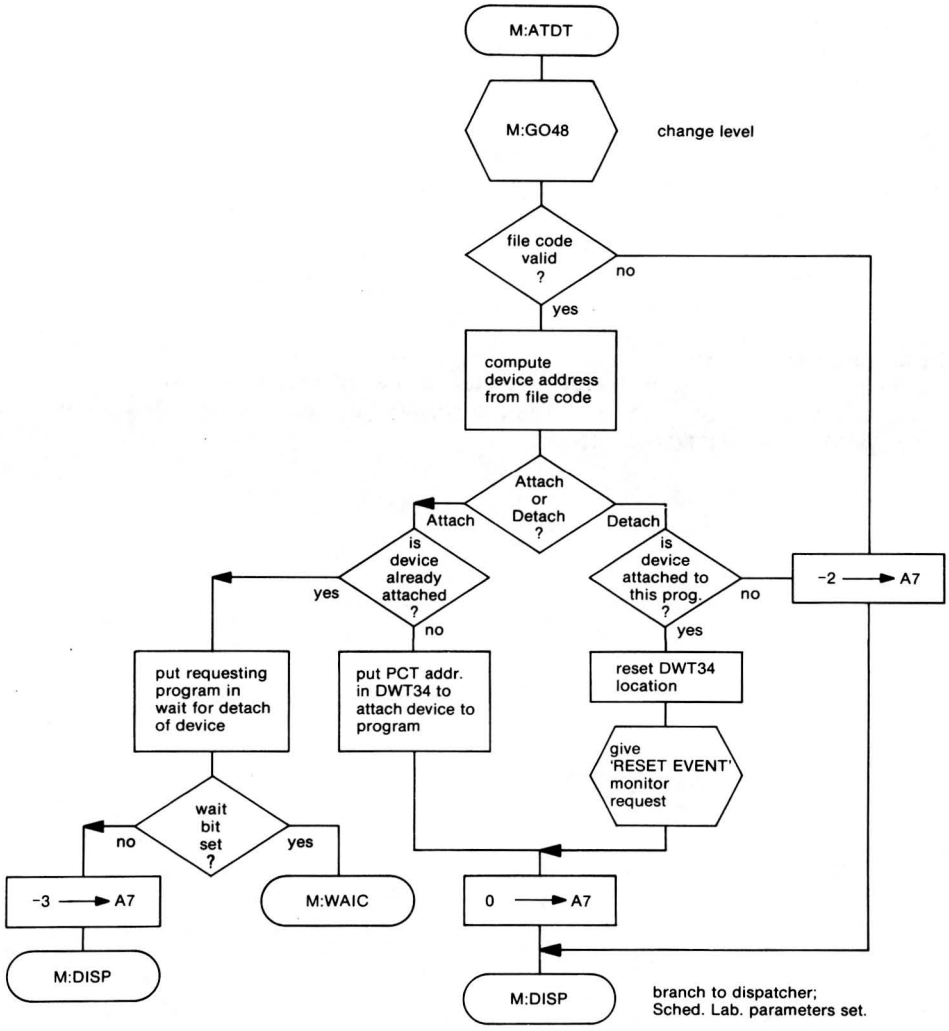
Entry Points: — M:ATDV (Attach function)
— M:DTDV (Detach function)

Work Areas and Tables
FCT (File Code Table)
DTW (Device Work Table).

Input/Output Files
None.

Functional Description
This module provides for attaching or detaching a device to or from a program.
If the device is already attached to another program, the requesting program may, depending on the value of the Wait Flag in A7, be put into wait state (with re-initialization) until the device is detached. Corresponding to the action to be taken, word 34 in the DWT (PCT address of program) is filled or set to /8000.

Note: The ASR is considered as 3 devices, so if the whole ASR is to be attached or detached, 3 requests must be given for the file codes corresponding to the ASR typewriter, ASR tape punch and ASR tape reader.


M:ATDT
M:GO48
change level
file code valid ?
no
yes
compute device address from file code
Attach or Detach ?
Attach
Detach
is device already attached ?
yes
no
is device attached to this prog. ?
no
yes
-2 → A7
put requesting program in wait for detach of device
put PCT addr. in DWT34 to attach device to program
reset DWT34 location
give 'RESET EVENT' monitor request
wait bit set ?
no
yes
-3 → A7
M:WAIC
0 → A7
M:DISP
M:DISP
branch to dispatcher; Sched. Lab. parameters set.

M:GTIM (Get Time — LKM17)

Calling Sequence

A7: contains a binary flag.
If this flag is zero, the date and time will be given in ASCII in the 6-word user block.
If it is not zero, they will be given in binary.

A8: address of a 6-word block, containing date and time.

Entry Point: M:GTIM

Work Areas and Tables

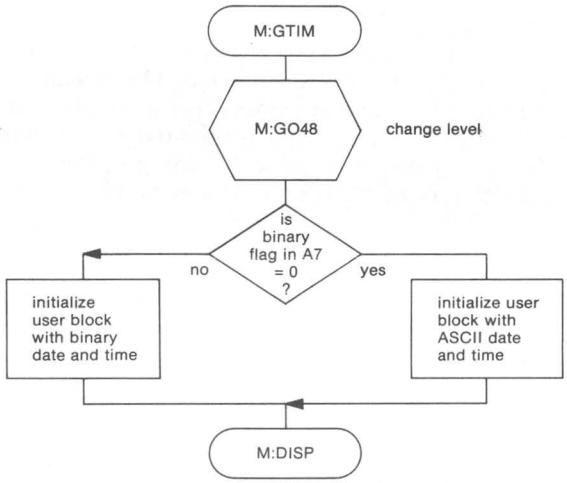
The timer block is read.

Input/Output Files

None.

Functional Description

Depending on the value in the A7 register, a 6-word user block will be filled with a binary or ASCII value specifying the date and time (DD␣MM␣YY␣HH␣MM␣SS).


M:GTIM
M:GO48
change level
is
binary
flag in A7
= 0
?
no
yes
initialize
user block
with binary
date and time
initialize user
block with
ASCII date
and time
M:DISP

M:RSEV (Set an Event – LKM18)

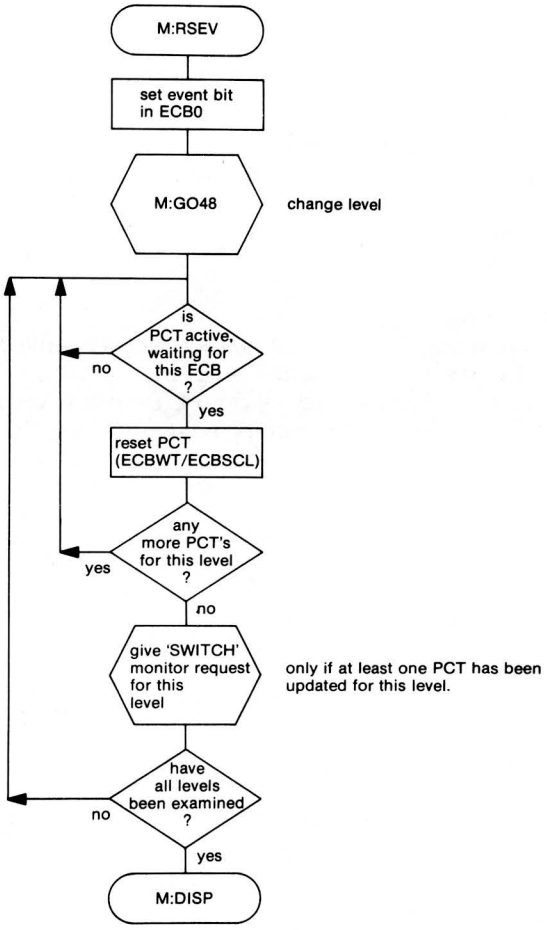
Calling Sequence
A5: PCT address of calling program
A6: Scheduled Label
A8: Event Control Block address.

Entry Point: M:RSEV

Work Areas and Tables
SLT (Software Level Table).
PCT (Program Control Table).

Input/Output Files
None.

Functional Description
When an event has occurred, the system may give this request so that the programs waiting for this event can be restarted. The M:RSEV module checks all PCT entries and updates them if necessary, i.e. the ECB addresses (ECBWT and ECBSCL in PCT). Each time an entry is updated, a *Switch* monitor request is given for this level, to make it possible for the program connected to that PCT entry, to regain control of the central processor as quickly as possible.


M:RSEV
set event bit in ECB0
M:GO48
change level
is PCT active, waiting for this ECB ?
no
yes
reset PCT (ECBWT/ECBSCL)
any more PCT's for this level ?
yes
no
give 'SWITCH' monitor request for this level
only if at least one PCT has been updated for this level.
have all levels been examined ?
no
yes
M:DISP

M:CNLV (Connect a Level to a Program — LKM20)

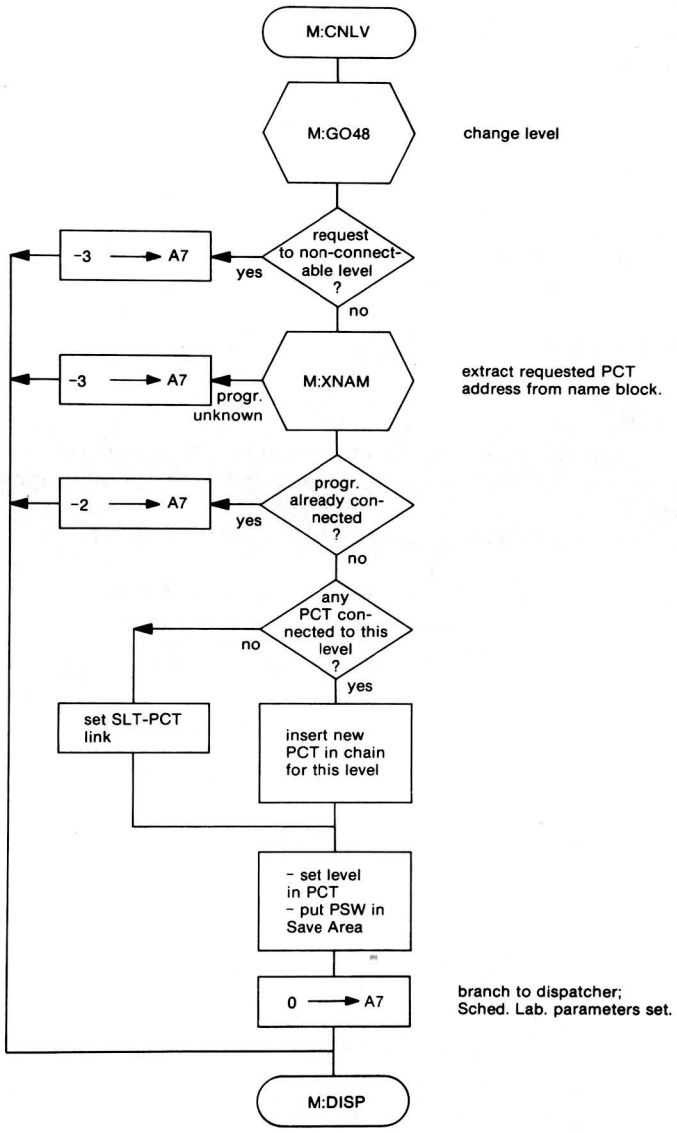
Calling Sequence
A5: PCT address of calling program
A6: Scheduled Label
A7: Level to which the program must be connected
A8: Address of name block of program which must be connected.

Entry Point: M:CNLV

Work Areas and Tables
SLT (Software Level Table)
PCT (Program Control Table)
Save area of program which must be connected.

Input/Output Files
None.

Functional Description
By means of this request a link is established between the Software Level Table and the PCT entry of the requested program.
This is simple in case the level is free, i.e. no PCT entry is yet connected to it. If the level is not free, the new PCT entry is inserted into the PCT chain for this level.


M:CNLV
M:GO48
change level
request to non-connect-able level ?
yes
-3 → A7
no
M:XNAM
extract requested PCT address from name block.
progr. unknown
-3 → A7
progr. already con-nected ?
yes
-2 → A7
no
any PCT con-nected to this level ?
no
yes
set SLT-PCT link
insert new PCT in chain for this level
- set level in PCT
- put PSW in Save Area
0 → A7
branch to dispatcher; Sched. Lab. parameters set.
M:DISP

M:DNLV (Disconnect Program from a Level — LKM21)

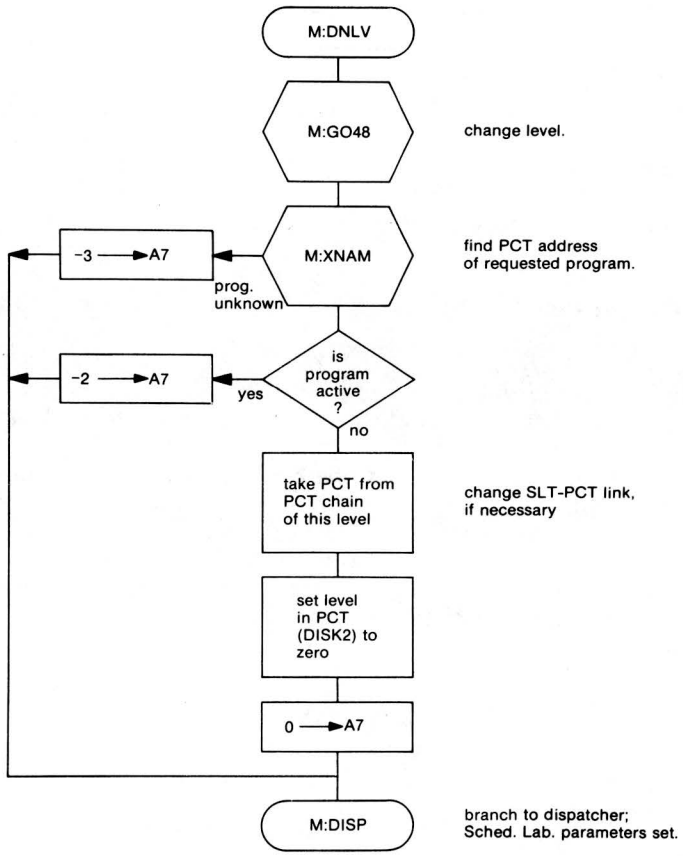
Calling Sequence
A5: PCT address of calling program
A6: Scheduled Label
A7: Level which must be disconnected
A8: Address of name block of program to be disconnected.

Entry Point: M:DNLV

Work Areas and Tables
SLT (Software Level Table).
PCT (Program Control Table).

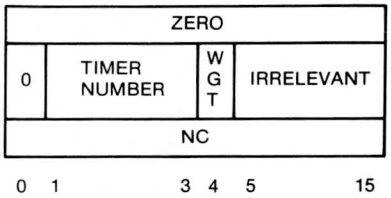
Input/Output Files
None.

Functional Description
With this request the PCT entry of the program which must be disconnected is taken out of the SLT-PCT chain, provided the corresponding program is in inactive state.
If there is only one PCT entry in the chain for this level, the SLT-PCT link is reset to zero.
If there are more entries in the chain, the requested PCT entry is removed.
In both cases the level specified in the PCT entry (DISK2), is reset to zero.


M:DNLV
M:GO48
change level.
M:XNAM
find PCT address
of requested program.
-3 → A7
prog.
unknown
is
program
active
?
-2 → A7
yes
no
take PCT from
PCT chain
of this level
change SLT-PCT link,
if necessary
set level
in PCT
(DISK2) to
zero
0 → A7
M:DISP
branch to dispatcher;
Sched. Lab. parameters set.

M:WGT (Wait for a Given Time — LKM22)

Calling Sequence
A8: address of an Event Control Block initialized as follows:



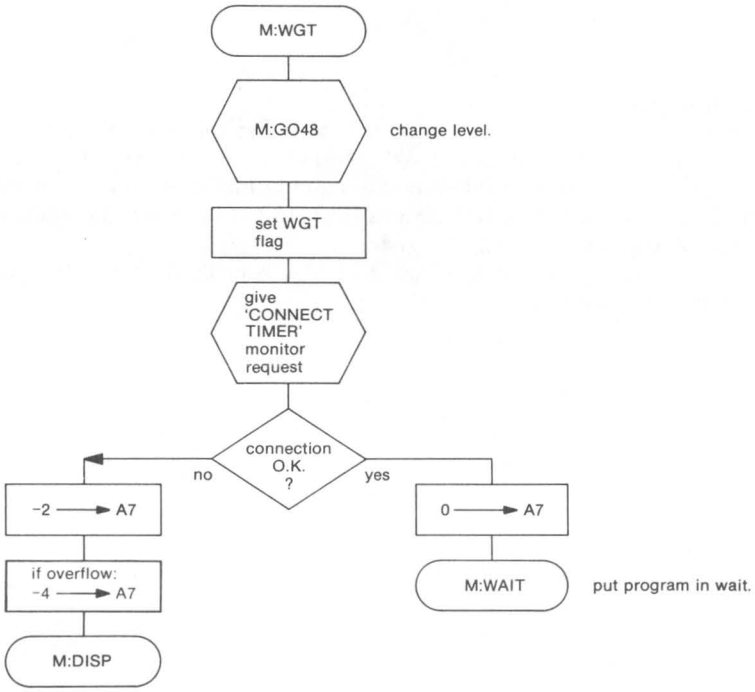
where NC is the number of timer cycles before the program is restarted.

Entry Point: M:WGT

Work Areas and Tables
None.

Input/Output Files
None.

Functional Description
This module processes a special timer connection, i.e. to a *Wait for a Given Time* block.
If the connection is accomplished, the M:WAIT module puts the program in wait state. If not, the error code −2 or −4 is set in the A7 register and control is returned to the dispatcher.
The user program is restarted by the M:DCK4 module after a number of cycles of a specified timer, as defined in a block pointed to by the A8 register. At the same time, the program is disconnected from that timer.


M:WGT
M:GO48
change level.
set WGT flag
give 'CONNECT TIMER' monitor request
connection O.K. ?
no
yes
-2 → A7
if overflow: -4 → A7
M:DISP
0 → A7
M:WAIT
put program in wait.

M:DMA (Dynamic Memory Allocation Handler)

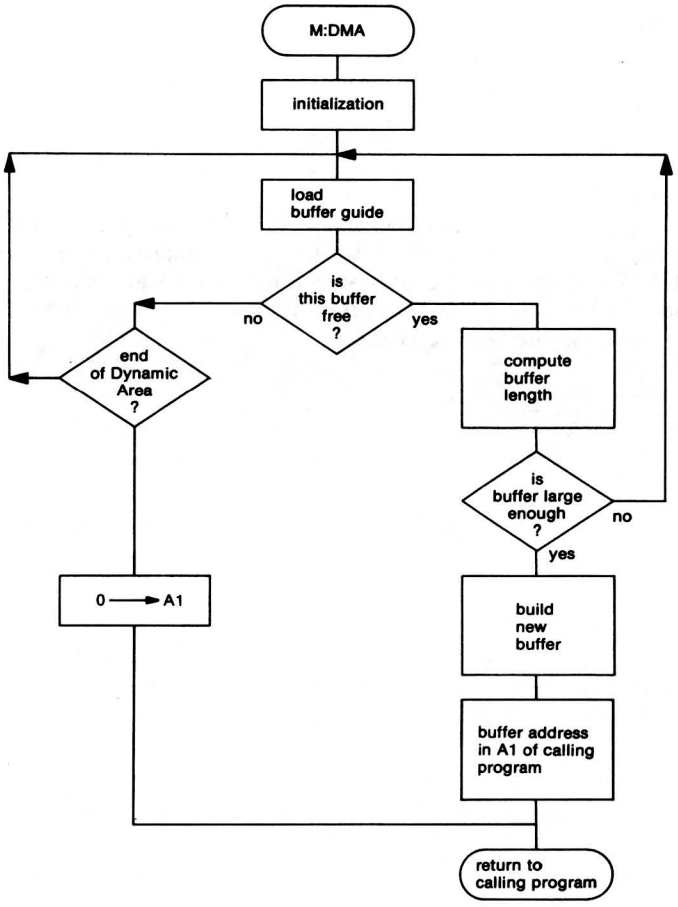
Calling Sequence
A1: Length of requested buffer, in characters (bits 1 to 15).
If bit 0 = 1: user request (from M:GBUF)
If bit 0 = 0: system request.

Entry Point: M:DMA

Work Areas and Tables
Dynamic Allocation Area (see Chapter 3).

Input/Output Files
None.

Functional Description
When a request is made to this module, it searches the dynamic allocation area for a free buffer. If no buffer can be found which is large enough for the request, the search is restarted. When a suitable buffer is found, it is initialized and the status flag in the buffer guide is reset to 0. On return the address of the requested block can be found in register A1.
In case a request cannot be satisfied, M:DMA returns A1 with the value 0, to indicate dynamic area overflow.


M:DMA
initialization
load
buffer guide
is
this buffer
free
?
no
yes
end
of Dynamic
Area
?
compute
buffer
length
is
buffer large
enough
?
no
yes
0 → A1
build
new
buffer
buffer address
in A1 of calling
program
return to
calling program

M:DML (Dynamic Memory Deallocation Handler)

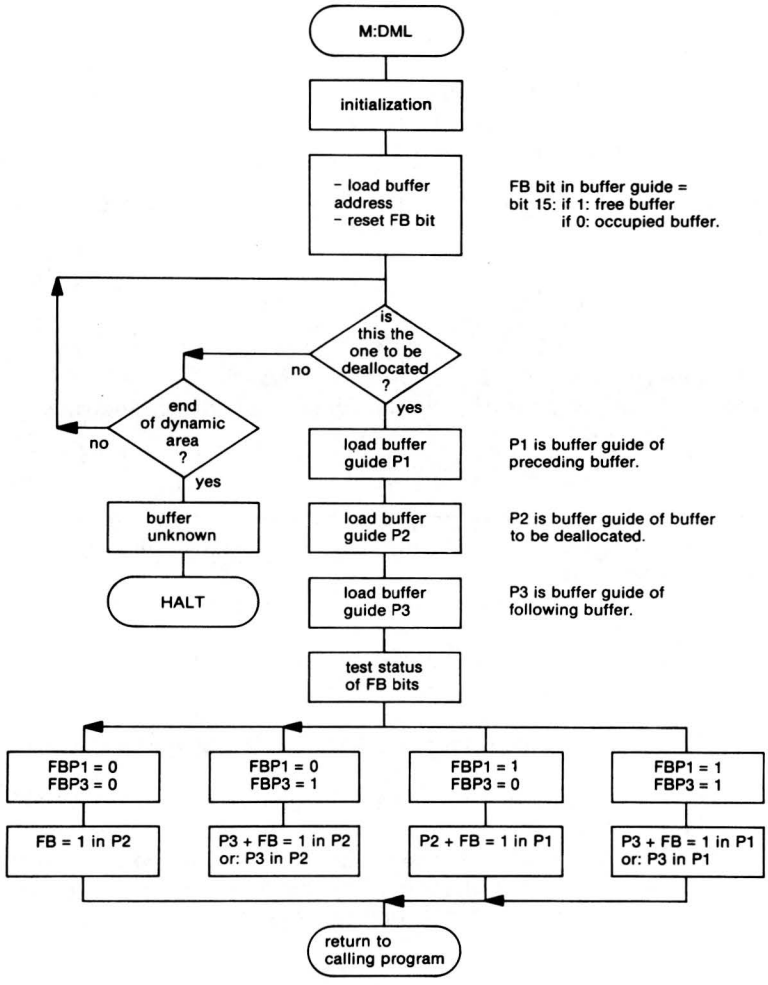
Calling Sequence
A1: address of the buffer which must be deallocated.

Entry Point: M:DML

Work Areas and Tables
Dynamic Allocation Area (see chapter 3).

Input/Output Files
None.

Functional Description
When a request is given to the M:DML module, it searches the buffer which must be deallocated in the dynamic allocation area (address in A1). If the buffer is found, M:DML updates the necessary buffer guides and the status flag in the deallocated buffer guide, thus returning the buffer for allocation.
If the buffer is unknown, the system halts.


M:DML
initialization
- load buffer address
- reset FB bit
FB bit in buffer guide =
bit 15: if 1: free buffer
if 0: occupied buffer.
is this the one to be deallocated ?
no
yes
end of dynamic area ?
no
yes
load buffer guide P1
P1 is buffer guide of preceding buffer.
buffer unknown
load buffer guide P2
P2 is buffer guide of buffer to be deallocated.
HALT
load buffer guide P3
P3 is buffer guide of following buffer.
test status of FB bits
FBP1 = 0 FBP3 = 0
FBP1 = 0 FBP3 = 1
FBP1 = 1 FBP3 = 0
FBP1 = 1 FBP3 = 1
FB = 1 in P2
P3 + FB = 1 in P2 or: P3 in P2
P2 + FB = 1 in P1
P3 + FB = 1 in P1 or: P3 in P1
return to calling program

M:DCK (Timer Handler)

Calling Sequence
This program is started by I:RTC activation.
On entry the parameters are contained in V:FLAG, which is set by the I:RTC module.

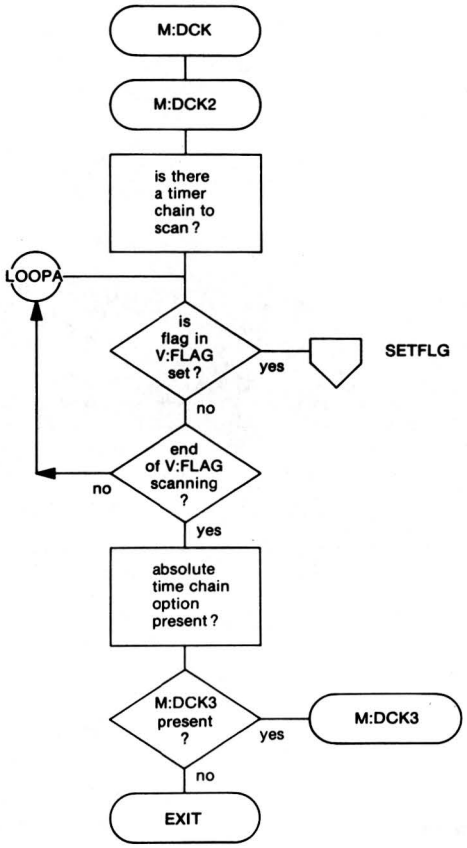
Work Areas and Tables
V:FLAG: flag vector indicating the timer chain to be scanned.
V:RSET: a value vector to reset the timers.
H:TIME: timer block start address.
H:POIN: chain pointer start address.
All programs connected to timer blocks (built by the M:CNTM module) are managed by M:DCK.

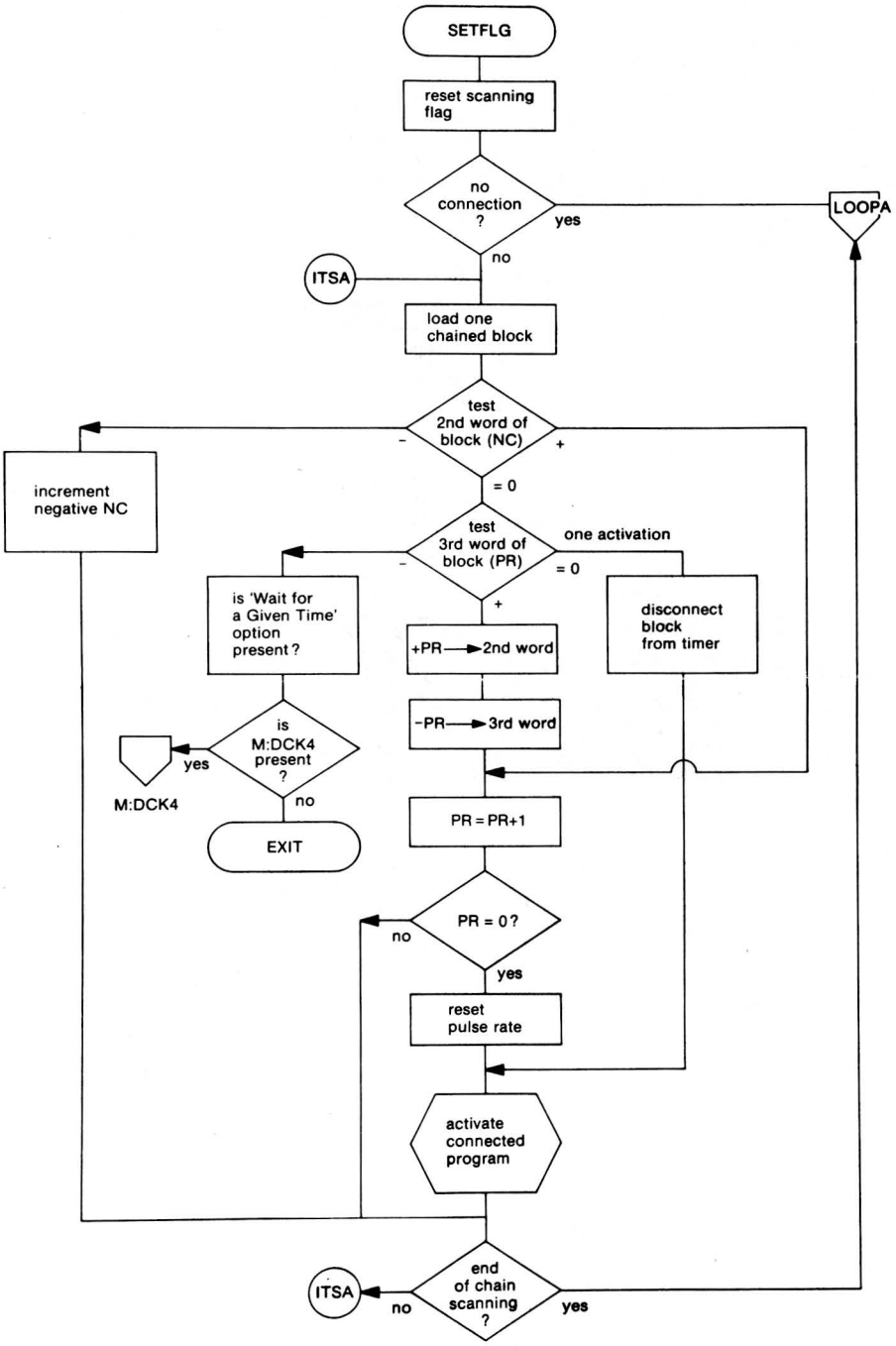
Input/Output Files
None.

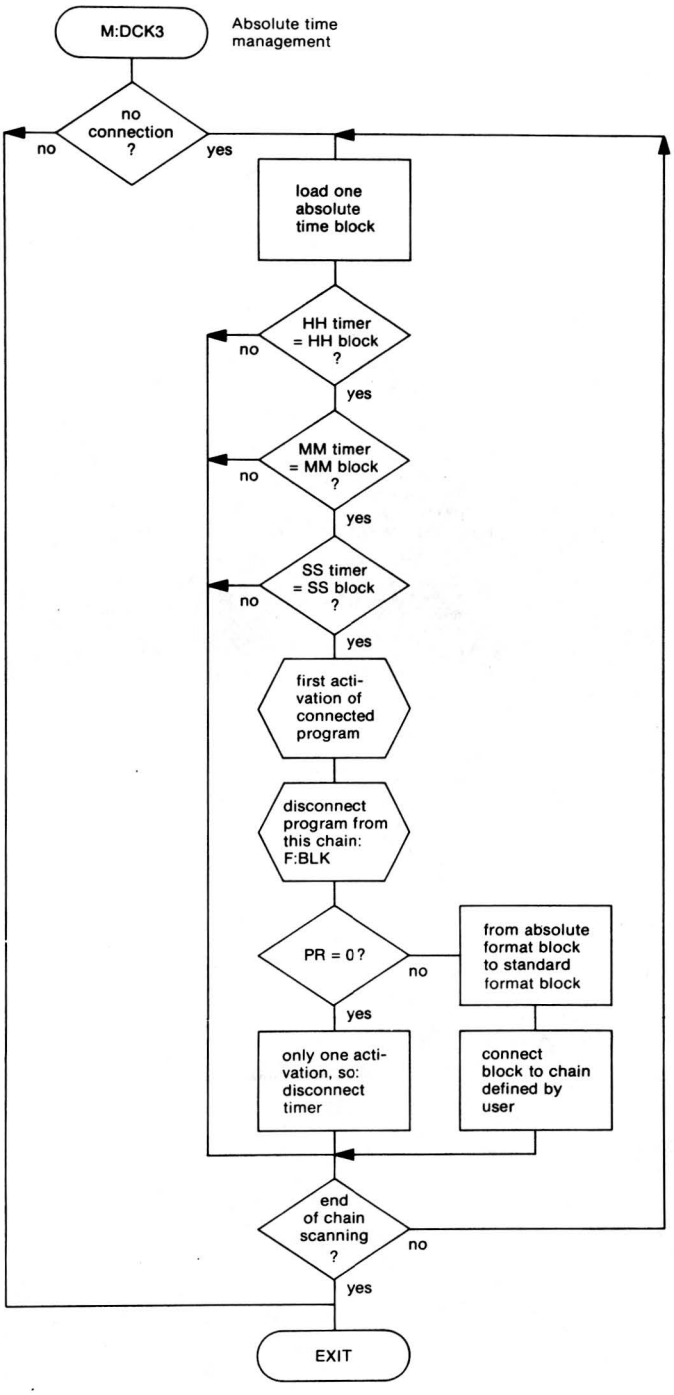
Functional Description
This module is started on activation by the I:RTC clock driver and runs at level 49.
It is composed of three modules:

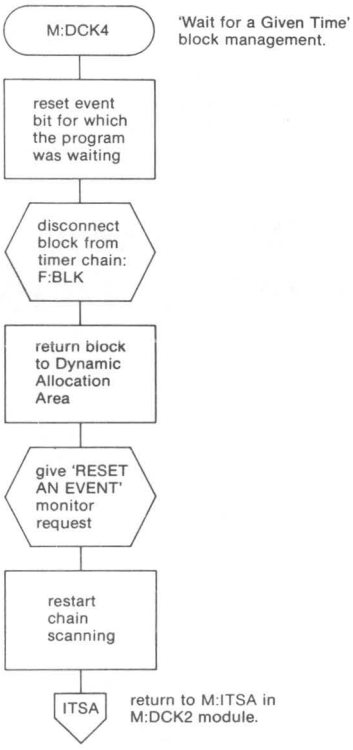
- M:DCK2 manages the chain of programs connected to standard timers.
- M:DCK3 manages the chain of programs connected to the absolute time (NC = HH␣MM␣SS).
- M:DCK4 manages the programs which are put in 'Wait for a Given Time'.

M:DCK2 first checks V:FLAG (initialized by I:RTC) to find out if a chain of programs connected to a timer must be analyzed.
At the end of M:DCK2, or in case no programs are connected to the scanned timer chains, the absolute time option is asked for. If it is present, M:DCK3 is started, otherwise M:DCK2 exits.
If there are programs connected to a chain flagged by I:RTC in V:FLAG, the timer blocks are updated and the programs are activated, if necessary. If a *Wait for a Given Time* block is detected, that option is asked for and if it is present, M:DCK4 is started. Otherwise, the block is ignored. At the end, control is returned to the dispatcher through an Exit request.


M:DCK
M:DCK2
is there a timer chain to scan ?
LOOPA
is flag in V:FLAG set ?
yes
SETFLG
no
end of V:FLAG scanning ?
no
yes
absolute time chain option present ?
M:DCK3 present ?
yes
M:DCK3
no
EXIT


SETFLG
reset scanning flag
no connection ?
yes
LOOPA
no
ITSA
load one chained block
test 2nd word of block (NC)
-
+
= 0
increment negative NC
test 3rd word of block (PR)
one activation
= 0
-
+
is 'Wait for a Given Time' option present ?
+PR → 2nd word
disconnect block from timer
-PR → 3rd word
is M:DCK4 present ?
yes
M:DCK4
no
EXIT
PR = PR+1
PR = 0 ?
no
yes
reset pulse rate
activate connected program
ITSA
end of chain scanning ?
no
yes


M:DCK3
Absolute time management
no connection ?
no
yes
load one absolute time block
HH timer = HH block ?
no
yes
MM timer = MM block ?
no
yes
SS timer = SS block ?
no
yes
first activation of connected program
disconnect program from this chain: F:BLK
PR = 0 ?
no
from absolute format block to standard format block
yes
only one activation, so: disconnect timer
connect block to chain defined by user
end of chain scanning ?
no
yes
EXIT


M:DCK4
'Wait for a Given Time' block management.
reset event bit for which the program was waiting
disconnect block from timer chain: F:BLK
return block to Dynamic Allocation Area
give 'RESET AN EVENT' monitor request
restart chain scanning
ITSA
return to M:ITSA in M:DCK2 module.

F:BLK (Release a block from a timer chain)

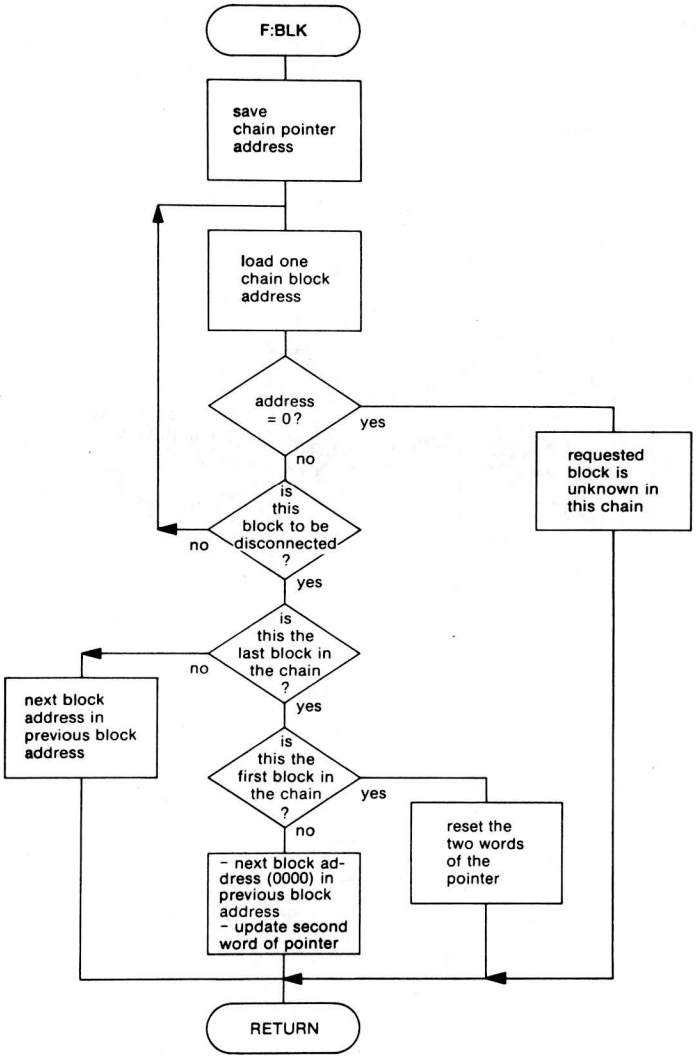
Calling Sequence
A1: Chain pointer address.
A2: Address of block which must be disconnected from the chain.

Entry Point: F:BLK.
This module is called only by:
– M:DNTM (Disconnect a Timer module)
– M:DCK3 (Absolute time management module)
– M:DCK4 ('Wait for a Given Time' management module).

Work Areas and Tables
H:POIN: chain pointer

Input/Output Files
None.

Functional Description
This module is a service routine by means of which a block can be disconnected from a timer chain.
The block which must be disconnected is looked up and the chain pointer updated.
At the end of the routine A1 will contain one of these values:
A1 $\neq -1$: disconnection performed.
A1 $= -1$: the block cannot be found in this chain.
A2 remains the same.
A3 and A4 are destroyed.
A6 must contain the return address.


F:BLK
save chain pointer address
load one chain block address
address = 0 ?
yes
no
requested block is unknown in this chain
is this block to be disconnected ?
no
yes
is this the last block in the chain ?
no
yes
next block address in previous block address
is this the first block in the chain ?
yes
no
reset the two words of the pointer
- next block address (0000) in previous block address
- update second word of pointer
RETURN

M:A00 (Non-wired instruction simulation routine)

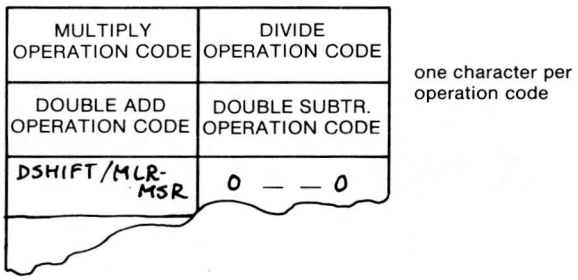
Calling Sequence

When an interrupt signal is generated as the result of the use of an illegal instruction code, the I:LKM module is started. First, this module checks whether the code specified is an LKM instruction. If not, the M:A00 routine is started and I:LKM transfers the following parameters to M:A00:

A1: user operation code
A2: address of user data
A3: user data.

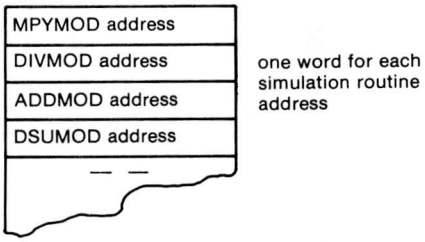
Work Areas and Tables

T:OPC (non-wired operation code table):

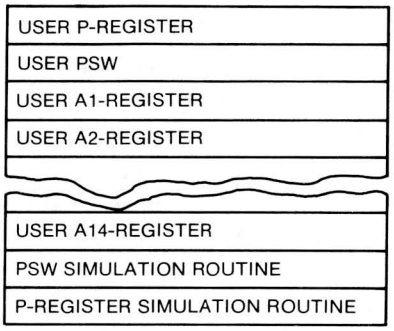


The scanning index is initialized by a parameter L:TOPC:
L:TOPC EQU 'non-wired op.code number — 1'

T:SRA (table of addresses of non-wired op.code simulation routines):

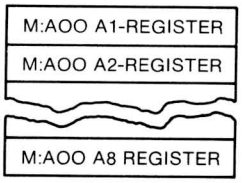


T:SVR (user information safeguard table)



Each entry (18 words) in this table contains P-register, PSW and registers A1 to A14 of the user program, plus the P-register and PSW of the simulation routine.
The number of entries is defined at system generation time.
The scanning index is initialized by a parameter L:TSVR:
L:TSVR EQU (number of entries — 1) ×36.

Z:SVR (M:A00 routine register save area)
The length of this area is 8 words:



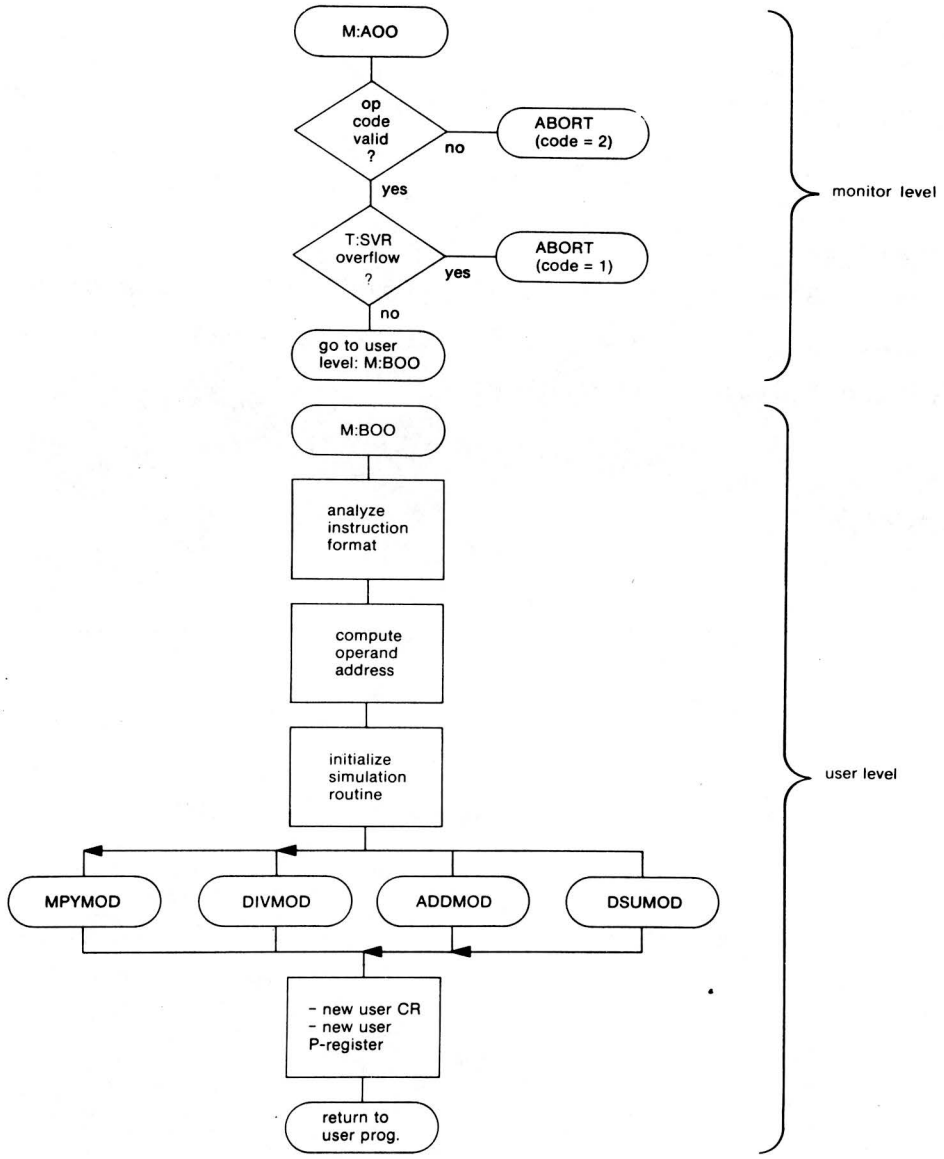
Input/Output Files
None.

Functional Description
First the M:A00 checks the validity of the given operation code. If it is not valid, the ABORT routine is started with code 2.
Then it checks the T:SVR table to assign a save area. If overflow occurs on this table, the ABORT routine is started with code 1.

Note: In this case the overflow is not a user error, but a system surrender (too many non-wired instructions encountered at the same time).

If a T:SVR entry is available, the user program must be restarted.

Then M:A00 gives control to the M:B00 which operates at user level.
This routine looks for the user operand and initializes an arithmetical routine to simulate the operation code given by the user.
At the end of simulation control is given back to the M:B00 routine which updates the user program's P-register, PSW and condition register before restarting the user program.


M:AOO
op code valid ?
no
ABORT (code = 2)
yes
T:SVR overflow ?
yes
ABORT (code = 1)
no
go to user level: M:BOO
monitor level
M:BOO
analyze instruction format
compute operand address
initialize simulation routine
MPYMOD
DIVMOD
ADDMOD
DSUMOD
- new user CR
- new user P-register
return to user prog.
user level

I:TRAP (TRAPPING ROUTINE)

Calling Sequence

Upon interrupt caused by use of an illegal instruction, a branch is made to I:TRAP via the trap location: memory address /7E.

Work Areas and Tables

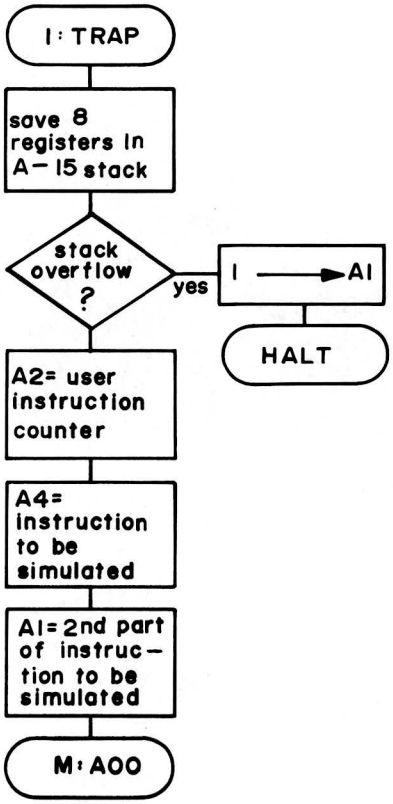
A-15 stack.

Input/Output Files

None.

Functional Description

After saving 8 registers in the A15 stack and a check on overflow, the parameters are prepared by I:TRAP for the M:A00 routine, which will simulate the illegal instruction.


I: TRAP
save 8 registers in A-15 stack
stack overflow ?
yes
I → AI
HALT
A2= user instruction counter
A4= instruction to be simulated
AI= 2nd part of instruction to be simulated
M: A00

Comment Sheet

P800M Programmer's Guide 1, Volume IV (5122 991 27342)

Name _____

Company _____

Department _____

Address _____

_____

Telephone Number _____ ext. _____

Comments or Suggestions:

PHILIPS DATA SYSTEMS B.V.

MARKETING GROUP SMALL COMPUTERS
P.O. Box 245, Apeldoorn, The Netherlands
Phone: 055-230123; telex: 49142

For further details contact the above address or:

EUROPE

Sweden
Svenska AB Philips
Data Systems
Minidatorer
Rissneleden 16
Fack
172 07 Sundbyberg
Tel. 08 830300

Denmark
Philips Data Systems A/S
Prags Boulevard 80
2300 København S
Tel. 0127 2222

Norway
Norsk A/S Philips
Data Systems Division
Nils Hansens vei 2
P.O. Box 5040
Oslo 6
Tel. 02 679380

Finland
OY Philips AB
Department Data Systems
Kaivokatu 8
P.O. Box 10255
Helsinki 10
Tel. 90 17271

Belgium
Philips Data Systems SA
Marketing Group Small Computers
Anspachlaan 1
1000 Brussel
Tel. 02 2193900

France
Philips Data Systems
Département Mini-ordinateurs
5 Square Max-Hymans
75015 Paris 15
Tel. 01 734 7759

Western Germany
Philips GmbH-Eiserfeld
Bereich Prozessrechner
Münsterstrasse 330
4 Düsseldorf 30
Tel. 0211 632087

Höhenstrasse 17
7012 Fellbach bei Stuttgart
Tel. 0711 523086
523088

Austria
Österreichische Philips GmbH
Industrie Elektronik
Breitenfurterstrasse 219
1230 Wien
Tel. 0222 831501

Italy
Philips S.p.A.
Sezione S and I
Viale Elvezia 2
20052 Monza
Tel. 039 361441

Switzerland
Philips AG
Data Systems
Binzstrasse 18
8027 Zürich
Tel. 01 442211

Great Britain
Philips Data Systems
Elektra House
2 Bergholt Road
Colchester
C04-5AA Essex
Tel. 206 5115

The Netherlands
Philips Data Systems B.V.
Bordewijkstraat 4
Rijswijk (Z-H)
Tel. 070 906720

Spain
Philips Ibérica S.A.E.
Grupo Instrumentación
Martinez Villergas 2
Madrid 27
Tel. 091 4042200

FAR EAST

Japan
Nihon Philips Corporation
P.O. Box 13
World Trade Centre
Hamamatsu-cho, Minato-ku
Tokyo 105
Tel. 03 435 5211

NORTH AMERICA

U.S.A.
North American Philips Corp.
Dept. 007
100 East 42nd Street
New York N.Y. 10017
Tel. 212 697 3600

Canada
Philips Electronics Industries Ltd.
Telecommunication Division
1001, Ellesmere Road
Scarborough 706
Ontario
Tel. 416 7521980