Fri 10/14/88
@ 2:00pm
MC Classroom 4

CONVEX C3 VP Functional Specification

Document No. 000-000000-000

Revision 1.0
September 19, 1988

CONVEX Computer Corporation

Table of Contents

List of Tables

List of Figures

1

Introduction

1.1 Scope of Document

This document will describe the function and design of the C3 Vector Processor (VP).

This document is organised into two chapters. Chapter 1 describes the scope and organization of the rest of the document and compares some of the features of the C2 and C3 vector processors. Chapter 2 is a functional description of the C3 VP hardware.

1.2 C3 vs C2

The following is a comparison of some of the major architectural features of the C3 and C2 vector processors.

- The C3 vector processor, like C1, will run 32-bit (and shorter) operations at twice the rate of 64-bit operations. Throughout the document this feature is referred to as "rate 2x". The C2 vector processor was prohibited from running any operations faster than the one operation per clock rate due to the choice of CMOS vector register files.

- Unlike either C1 or C2, the C3 vector processor will have the ability to perform under mask operations *only on those elements who's VM bit is set, entirely skipping those that are not set.* For example, if VL=128 and only half of the bits in the VM are set, an under-mask instruction will only take (64+overhead) cycles on C3 rather than (128+overhead) cycles on C2. This method is variously referred to as "operate from list" and "VM accelerate".

- Add pipe functionality will be provided on the Multiply pipe as well as the Add pipe, and edit functions will operate on either the Add or Multiply pipes. Hence, the Add and Multiply pipes will be almost homogeneous with the exceptions of multiply, divide, and square root. This redundant functionality could be imperiled by cost or space factors.

- The C3 vector register files (VRF) will be "stacked" in the mode of C1 and C2 where registers are paired v0,v4 v1,v5 v2,v6 v3,v7. Stacking is required by gate-array space limitations. Two read accesses and one write access to each register pair on every cycle are provided by triple-cycling the gate-array RAM.

- The C3 vector processor will not have the even/odd structure that the C2 vector unit had. 64-bit operands will all be sent to a single set of function units. However, in order perform 32-bit operations at rate 2x, it is necessary to send the operands out at twice the rate of 64-bit operands. This is done by splitting the 64-bit operand busses into two 32-bit busses for the rate 2x operations. This appears to be an even/odd split from the perspective of the function pipes, although a single function unit may be receiving both busses.

- The C3 vector processor carries on the C2 tradition of having the entire vector processor operate in lock step by extending the clocks when any part of the VP is unable to produce or consume data on a cycle.

- The C3 vector unit may be provided with a foreplane port which will allow external connection of custom function devices to the vector processor. This would allow special devices to have the full bandwidth memory access that the VP itself has without physical modification of the CPU.

2

Functional Description

2.1 Overview

The C3 Vector Processor (VP) participates in the execution of all vector instructions executed by a processor. It contains the vector registers, vector mask register, and vector function units for execution of vector instructions. There are three pipes within the Vector Processor: the "Add" pipe, the "Multiply" pipe and the "Load" pipe. The Add pipe can perform all integer arithmetic/compare, floating point arithmetic/compare, logical and vector edit operations with the exception of multiplies and divides. The Multiply pipe can perform all multiply, divide, logical and vector edit operations (integer and floating point arithmetic/compare operations may also be added, space and cost permitting). The Load pipe performs all vector/VM/VS load/store operations.

A block diagram of the C3 Vector Processor is shown in Figure 2-1. The scalar processor dispatches instructions to the Vector Dispatch (VD) body of the VP. The VD body selects the pipe to execute the instruction, determines whether the vector register file ports and other required resources are available, and determines whether chaining and/or accelerated execution may occur. The VD body also waits for a seed from the scalar unit if it is required. When all the required conditions are met, the VD body dispatches the instruction to the selected pipe controller. This dispatch includes an entrypoint microaddress, vector register and port selects, source and destination data sizes, execution rate, and the scalar seed. The pipe controller then may execute the instruction without performing any resource checks.

Each pipe has it's own controller (UA/VM/ABD, UM/VM/MBD and UL/VM/LBD) which contains a microsequencer, a copy of the VM register, a control queue and a backdoor controller. Each pipe controller is responsible for reading vector elements from the Vector Register File (VRF) body, sending these elements through it's operation pipe (memory for the Load pipe), and writting the results back into the vector register file.

The Add Function Pipe (AFP) and Multiply Function Pipe (MFP) bodies perform all the "processing" within the Vector Processor, and are controlled by the UA and UM controllers respectively. The Input Staging (IS/ISCTL) body receives and queues all data from the scalar processor and memory. Control of the IS body is shared between the VD and UL bodies. All three pipe controllers may send data to the scalar processor and memory. The Output Staging Control (OSCTL) body handles the handshakes for these transfers, while the VP-AS.DATA bus is used for the data transfer.

2.1.1 Pipe Nomenclature

In order to keep the many stages of pipelining in the VP organized, a convention for naming pipes and pipeline levels was defined and strictly followed. Signals within a pipe are prefixed by "a", "m" or "l" for the add, multiply and load pipes respectively. This prefix is followed by a string indicating the pipeline level: "_", "1_", "2_", "3_", "q_", and "bd_". The first three strings refer to 0, 1, 2, and 3 levels after the control store and dispatch registers. The "q_" refers the the output of each pipes control queue, while "bd_" is one level after the queue level (the backdoor level). Thus "a_active" is the active bit for the add pipe microsequencer, "a3_active" is delayed three clocks, "aq_active" exits the control queue, and "abd_active" is in the backdoor registers.

Figure 2–1: C3 Vector Processor

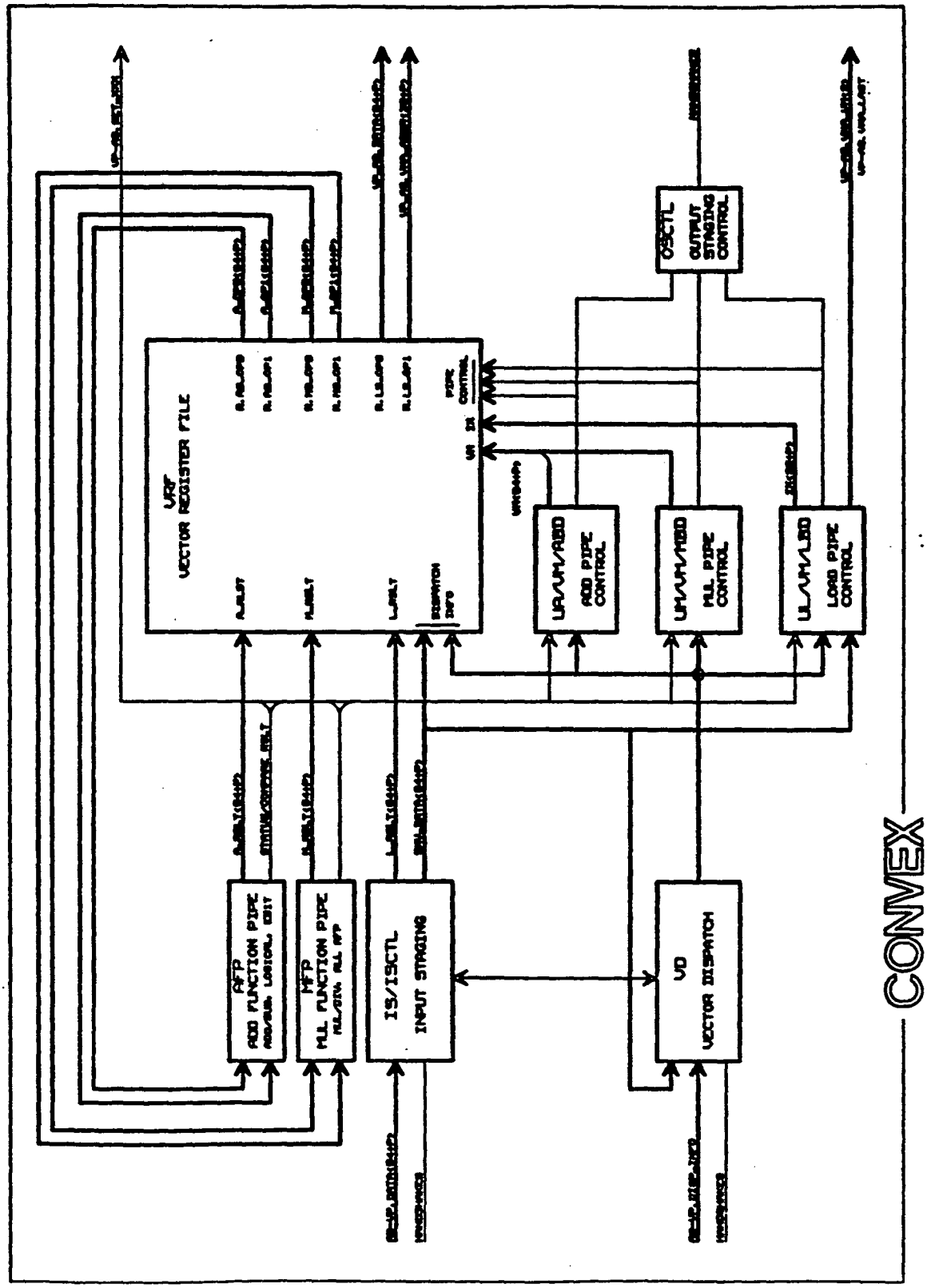


Figure 2-1
C3 VECTOR PROCESSOR

The VP is packaged as a single 18" by 20" multilayer printed circuit board. All board internal and interface signals are 100k ECL compatible.

The following sections describe the each of the bodies in more detail.

2.2 Vector Dispatch

A block diagram of the C3 Vector Dispatch interfaces is shown in Figure 2-2. The Vector Dispatch (VD) logic receives instructions from the instruction processor on the scalar unit via the IPCTL controller. These instructions are dispatched to the function pipe microcontrollers on the vector processor when all conflicts for resources are resolved. Resources primarily are: VRF read ports, VRF write ports, VM write port, function pipes, the scalar-to-vector interface, or the vector-to-scalar interface. For those instructions that are capable of executing on either the ADD or MUL pipe, the vector dispatch logic determines which function pipe an instruction will execute on.

The IPCTL controller performs all of the required handshaking with the instruction processor and attempts to maintain a queue of instructions such that an instruction will always be available as soon as the vector dispatch logic is ready.

When the IPCTL has an instruction available and the VD is ready, the VD begins a dispatch cycle. A block diagram of the C3 Vector Dispatch control is shown in Figure 2-3. Dispatch requires at least two clock cycles and is not pipelined. In those two cycles it reads the lookup table to determine the required resources for the instruction, checks the availability of all of those resources and prepares to allocate the resources that the instruction will require. If the resources are not available, the dispatch will be held off until they become available.

Resource availability checks fall into 7 main categories:

- Read port check/allocation
- Write port check/allocation
- Function pipe check
- Chaining checks
- Scalar port checks
- Rate 2x and VM accelerate checks
- Miscellaneous hazard checks

Read port check and allocation logic ascertains whether read ports on the requested registers are available, and assigns them if they are. Since there are two read ports on each register pair, either one can be assigned to any instruction. If an instruction requests the same register twice (e.g., *or.w v0,v0,v1* to copy a register), only one read port is assigned to the instruction and the VRF is commanded to use the same port for both leaving the other port free for reuse of the register.

Write port check and allocation logic determines whether the requested write port is available, and assigns it if it is. The requested write port may already be allocated to any of the three pipes, and either the front-door or back-door of any of those pipes may be using the write port. If a front-door is using the write port, dispatch is blocked. If a back-door is using the write port, dispatch is blocked *unless* the instruction in dispatch has a pipe length greater than or equal to the instruction that is currently hogging the write port.

Function pipe check logic determines whether the requested function pipe is in use. If the pipe is in use it blocks dispatch unless the dispatching instruction has a pipe length greater than the *remaining* pipe length of the executing instruction.

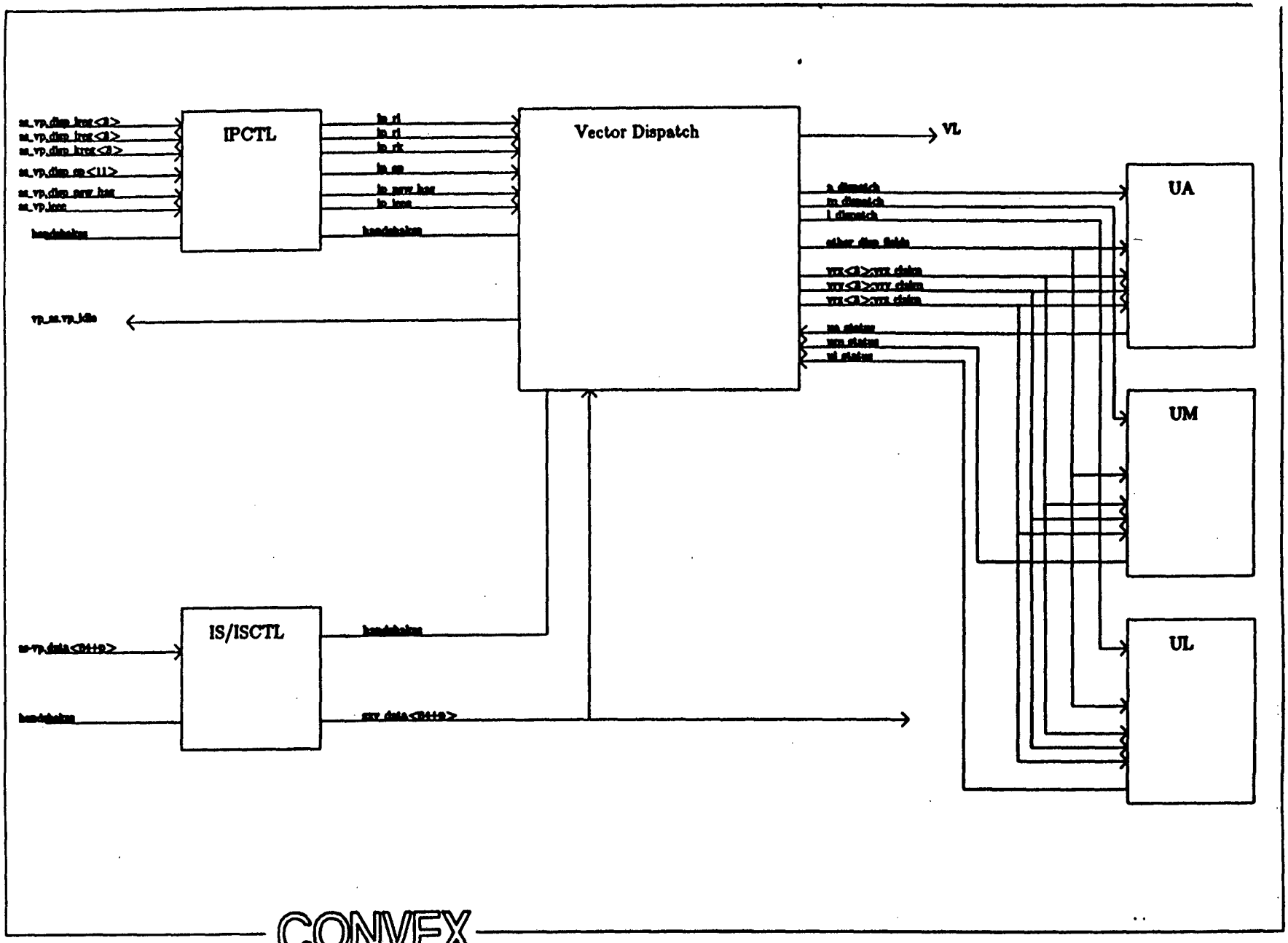


Figure 2_2
Vector Dispatch Interfaces

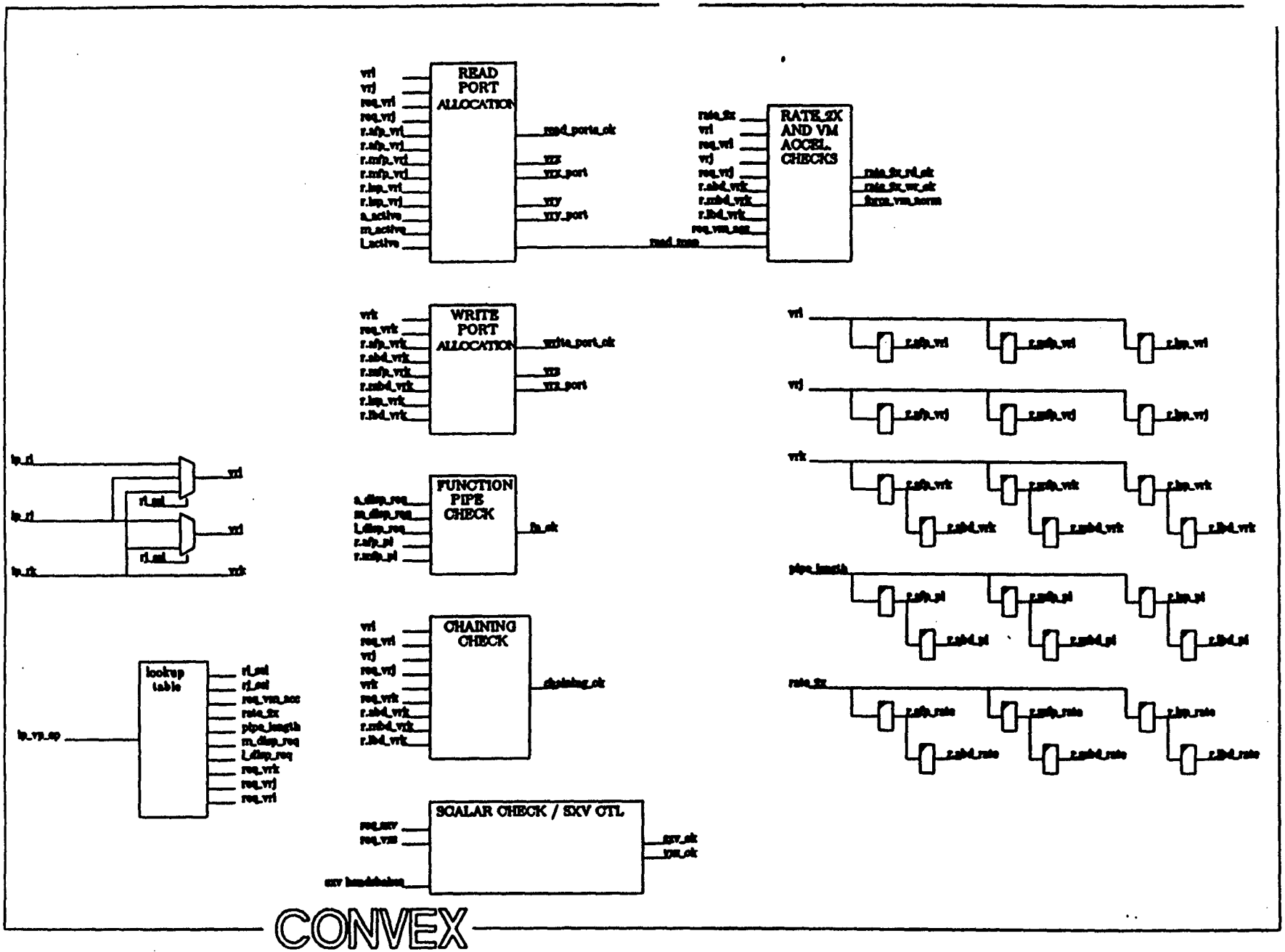


Figure 2-3

Vector Dispatch Control

Chaining check logic determines whether an executing instruction is writing to one of the registers (VRF or VM) that the dispatching instruction needs to read, and blocks dispatch until that executing instruction has written at least one result into that register. This approach assumes that all of the C3 function pipes run in lock-step. That is, that either all of the function pipes advance one step or all of the function pipes advance zero steps on each clock.

Scalar port checks control access to the scalar-to-vector (SXV) and vector-to-scalar (VXS) interfaces. Only one instruction which requires the SXV interface is allowed to execute at any time. This alleviates scalar data ordering problems. Only one instruction which requires the VXS/VXM (vector-to-memory) interface is allowed to execute at any time. This alleviates scalar and memory data ordering problems. Instructions which would violate the above conditions are delayed in dispatch until the resource becomes available. This body also interfaces with the Input Staging body to get scalars for those instructions that require them and to load the VL register.

Rate 2x and VM accelerate checks determine whether a dispatching instruction may cause problems as a result of consuming or producing data at an accelerated rate. Rate 2x and VM accelerated instructions may consume data and produce results at a rate that is faster than other instructions executing on the vector unit. These fast instructions can read data more rapidly than it is being produced by a normal instruction. In a chaining situation this could result in the fast instruction reading past the data that is being written by the normal instruction and it may begin consuming from vector elements that have not yet been written to. This is prohibited by this check. In this case a rate 2x instruction is prohibited from executing until the offending instruction completes; the under mask instruction would be forced to go at the normal rate rather than at the accelerated rate, but is dispatched immediately.

The other hazard is the reverse of the above. The fast instruction can produce results that overwrite elements that have yet to be read by a preceding instruction (e.g., *add.l v0,v1,v1* followed by *mul.w.t v2,v3,v0* where the *add.l* goes at the normal rate and the *mul.w* runs at rate 2x.) This is prohibited by the check. The dispatch acts in the same way as the above condition.

Miscellaneous checks are a set of small checks that do not fall into one of the above categories. These checks are as follows:

- VM write port checking and allocation. Both the MUL and ADD pipes can write VM bits serially on C3. The LOAD pipe can parallel write the VM register. The VM must be allocated properly.
- Simultaneous serial and parallel accesses to the VM register are prohibited.
- Edit instructions that do not produce or consume data at the single cycle rate must not interfere with instructions that do.

2.3 Vector Register File

A block diagram for the Vector Register File body is shown is Figure 2-4. The VRF contains the memories for the vector registers, along with the data selection and staging required to allow the three microcontrollers to perform their operations as independently as possible. There are four 256 location by 72 bit vector RAM banks: V0/V4, V1/V5, V2/V6, and V3/V7. Each bank is divided into two 256x36 bit halves (macros) which are separately addressed, as shown if Figure 2-5. The upper and lower words of each even vector element are swapped between the two halves. This allows pairs of byte/halfword/word vector elements to be read from the bank each clock, thus providing the bandwidth to run "single 2X double". Even longword elements are word swapped on writing and reading. Each RAM macro has internal registers on the address, write enable, read data and write data lines, and is triple cycled to allow two reads and a write each clock.

There are three sets of dispatch registers within the VRF, one for each pipe. When a microsequencer is dispatched an instruction, the "frontdoor" dispatch registers in the VRF for that pipe are loaded with selects and controls for the instruction. These dispatch registers route

Figure 2–4: Vector Register File

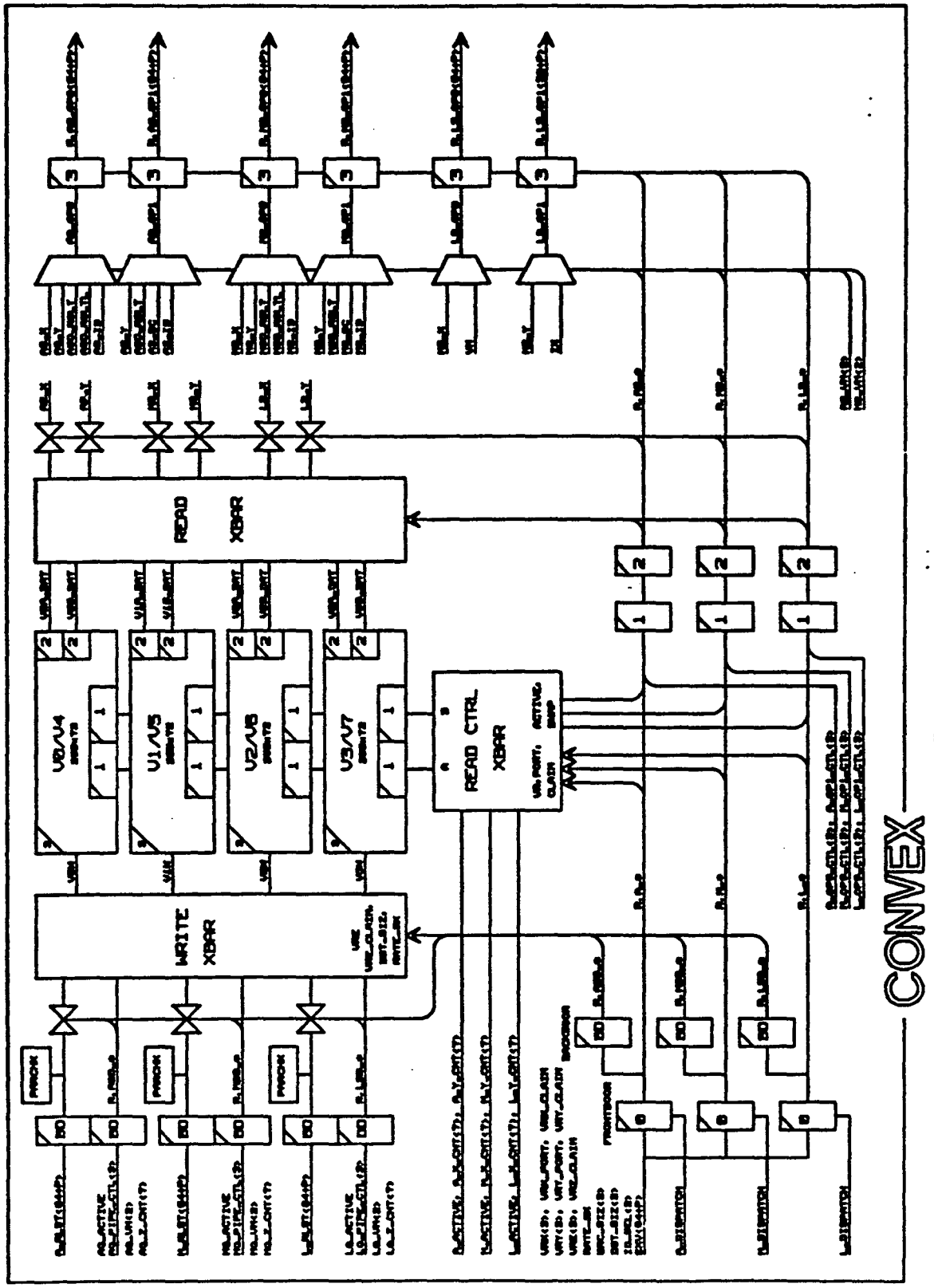
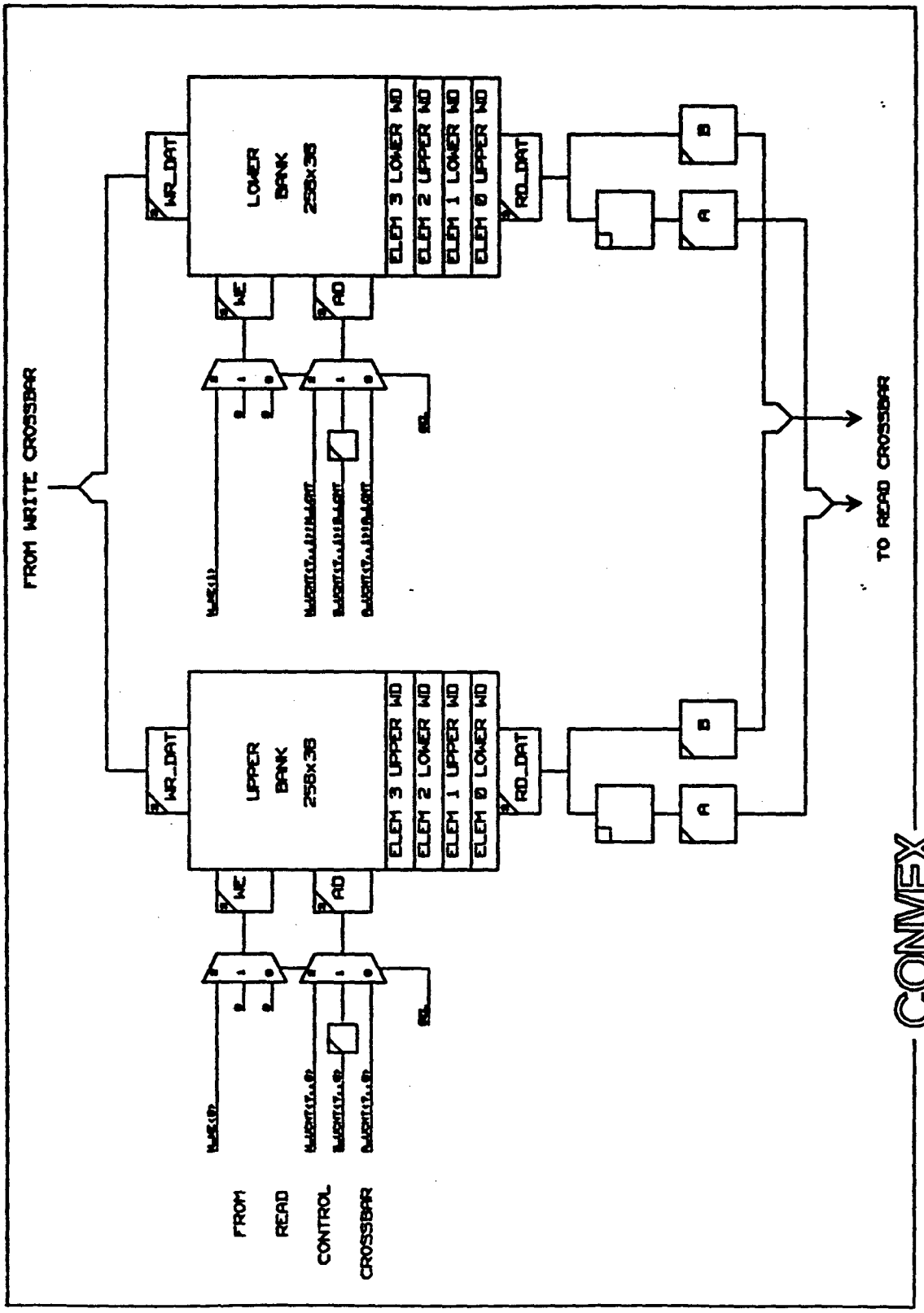


Figure 2–5: VRF Bank Organization



read addresses and controls from the microsequencer to the vector RAM banks via the read control crossbar. The dispatch registers are also piped two register levels to route read data from the vector RAM banks to the pipe data path via the read crossbar. These piped dispatch registers also control the swappers, operand selection muxes and data registers of the pipe.

A Xq_pipe_ctl code causes the "backdoor" dispatch registers for a pipe to be loaded. These backdoor dispatch registers route pipe results and controls to the vector RAM banks via the write crossbar. Parity is checked on pipe results while they are being written into the RAM banks.

The VRF will be bit sliced into eight 20K gate arrays, each array implementing 4 bits of the upper word and 4 bits of the lower word of every longword.

2.4 Function Pipes

The Function Pipes accept opcodes and data and perform the requested operations. The Add and Multiply pipes constitute the function pipes on the C3 vector processor. The function pipes are composed of multiple functional gate arrays and a function pipe (FP) controller for each pipe ("AFP controller" and "MFP controller"). The FP pipe controllers start the appropriate function unit and enable the appropriate function unit to drive the bus when its data is available. A block diagram of the vector processor function pipe architecture is shown in Figure 2-6.

The function pipes use a "split" bus structure to support the rate 2x mode for data types shorter than 64 bits. When 64-bit data types are being processed, an entire 64-bit bus contains a single piece of data. However, when 32-bit (or shorter) data types are being processed, a 64-bit bus is used to transfer two 32-bit pieces of data on a single cycle. The even data goes on the lower half of the bus and odd data goes on the upper half of the bus. This split bus structure permits the basic 64-bit busses to support the rate 2x mode without extra pins, busses or function units.

At present it has not been determined exactly how the function units will be implemented. Some assumptions are made here about the performance of the function units. The function units, when developed, must conform to the split bus structure used on the vector processor.

2.4.1 Add pipe

The add pipe will probably be composed of two gate arrays and a FP controller. One of the gate arrays will implement the floating point add/subtract operations. The other will implement all of the integer, logical, type convert and other miscellaneous instructions. Each of these function units will be required to receive and process a single 64-bit operand set or a pair of 32-bit operand

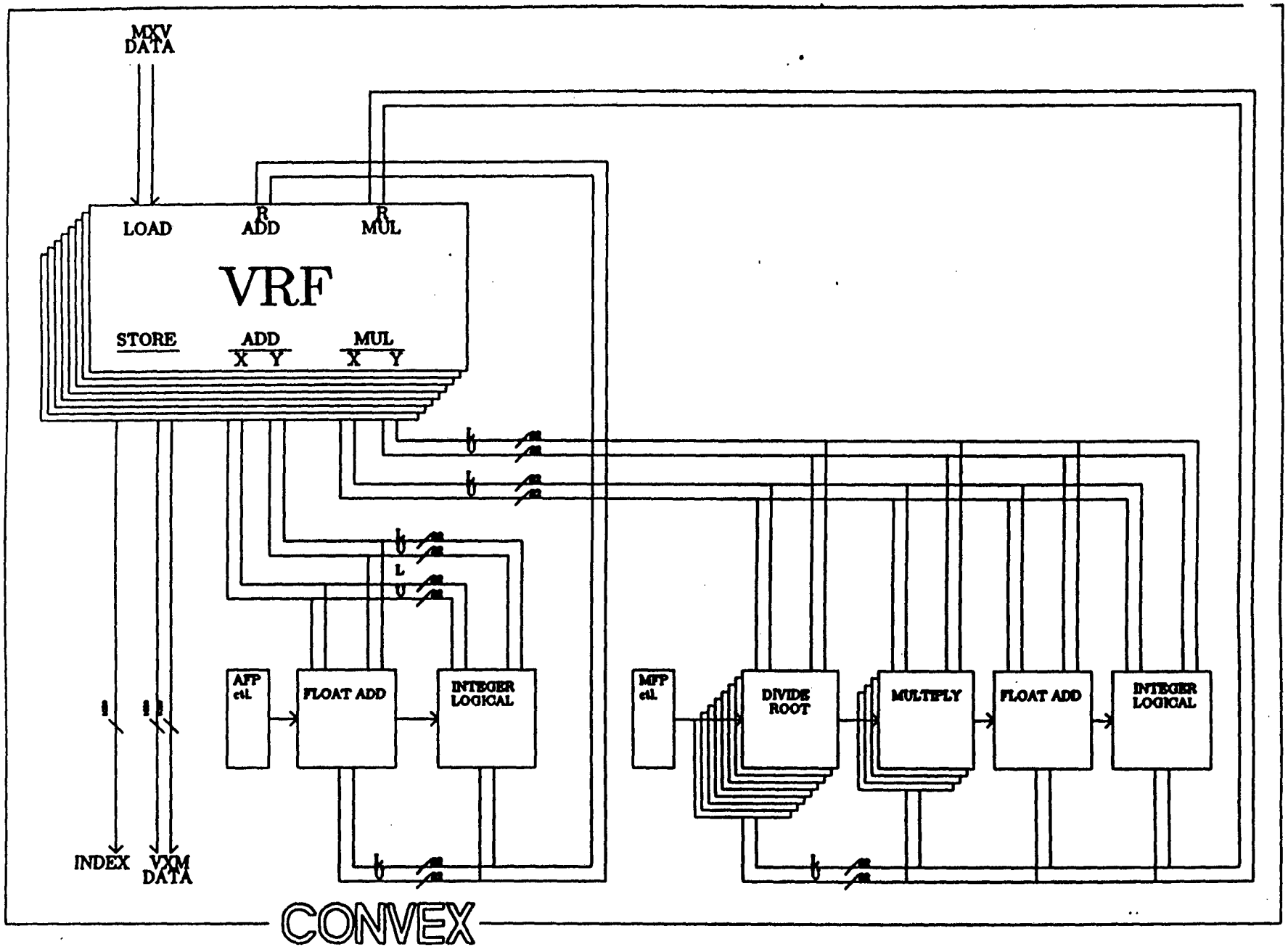


Figure 2-6
Function Pipe Architecture

sets on every cycle.

Table 2-1: Required functions for add pipe function units

Operation	Sizes
add	bhwlsd
sub	bhwlsd
le/lt/eq	bhwlsd
cvt	bhwlsd
min/max	bhwlsd
frint	sd
and	l
or	l
xor	l
not	l
shf	l
tsc	l
lop	l

Table 2-1 shows the required functions for the add pipe function units. The functions are described using the Convex assembler mnemonics. The min/max instructions require a bit that tells which of the inputs was the min/max, and the result data port must contain the actual data input which was min/max.

The AFP controller starts the appropriate function unit and enables it to drive the result bus at the appropriate time.

2.4.2 Multiply pipe

The Multiply pipe will probably be composed of 8 divide/square root ASICs, 4 multiply ASICs, and the two Add pipe gate arrays plus the MFP controller. It is assumed at this point that an ASIC divide unit with two dividers each capable of producing one bit per cycle will be available. Eight of these ASICs will provide 16 result bits per cycle. This rate produces a double precision result or two single precision results every four cycles. This is the same result rate that C1 and C2 have. Other configurations can easily be supported as long as they provide an aggregate rate of 16 bits/cycle and support the split bus structure.

Table 2-2: Required functions for multiply pipe function units

Operation	Sizes
mul	bhwlsd
div	bhwlsd
sqrt	sd

Table 2-2 shows the required functions for the multiply pipe function units. The functions are described using the Convex assembler mnemonics.

The number of multipliers is predicated on a function unit architecture based on a 16x54 bit multiplier core. This architecture requires four passes to complete a double precision or long integer multiply. Hence, four function units are required to produce one result every cycle. The multiply function units must receive 32-bit data from either the upper or lower half of the bus and drive the result onto the half that it got the data from.

The MFP controller starts the appropriate function unit and enables it to drive the result bus at the appropriate time. It also provides select signals to each function unit to allow that unit to operate on 32-bit data from either the upper or lower bus in the split bus configuration. When the multiply pipe is performing an operation that produces less than one piece of data per cycle (i.e. *div* and *sqrt*), the MFP controller extends the clocks on the rest of the VP to maintain lock step.

2.5 Input Staging

The Input Staging (IS) body receives and queues data from the scalar processor and memory. A block diagram for the IS body is shown is Figure 2-7. Memory (MXV) data and scalar (SXV) data are queued separately. Each queue is three levels deep, which provides enough overrun after an internal clock extend to absorb incoming data until the REQ_NEXT signals can be negated (plus one more transfer). Parity is checked on the top entry in the queue, so although a parity error may not be detected immediately, the bad data will always be retained.

Scalar data is driven onto the SXV bus. When data is available it may be popped by either the load microsequencer, the load backdoor controller, or the vector dispatch unit. Memory data or scalar data may be driven onto the load pipe result bus. Load pipe result data may only be popped by the load backdoor controller.

The IS may either be implemented on one 20K gate array, or be bit sliced into two 10K gate arrays each array implementing 36 bits. The possible problem with the 20K array is that the IS has 144 simultaneously switching outputs.

2.6 Pipe Controllers

Each VP pipe controller has a microsequencer, a set of vector element counters, a copy of VM, a control queue and a backdoor controller. A block diagram for the microcontroller is shown is Figure 2-8.

2.6.1 Microsequencer

The microsequencer controls the X, Y, Z and VM counters, selects operands from the VRF, generates a code to control operations farther down the pipe, and determines several "last element" conditions. The microsequencer is in the lower left part of Figure 2-8.

The microsequencer can accept a dispatch when it is idle or going idle (rdy). When accepting a dispatch, the six bit entrypoint is shifted left 3 bits to form an initial microcode address. The vector length and other parameters are registered, and the vector length is munged into a case vector for reduction operations. An active bit is set whenever the microsequencer is executing an instruction. The microaddress generated is applied to the control store RAMS to fetch the next microinstruction, and is also registered to facilitate single instruction micro-loops. Parity is checked across the microinstruction, and the microword and micro-pc are held if a parity error is detected. The microinstruction may select a condition to test. The result of the test is registered, and may be used on the following microinstruction for conditional micro-jumps.

2.6.2 Vector Element Counters

There are four element counter for each pipe. The X and Y counters address the vector source registers, the Z counter addresses the vector destination register, and the VM counter addresses the VM register for the selection of mask bits. All counters are cleared at the start of each instruction, and may increment at different rates for different instructions. The VM counter is

Figure 2–7: Input Staging

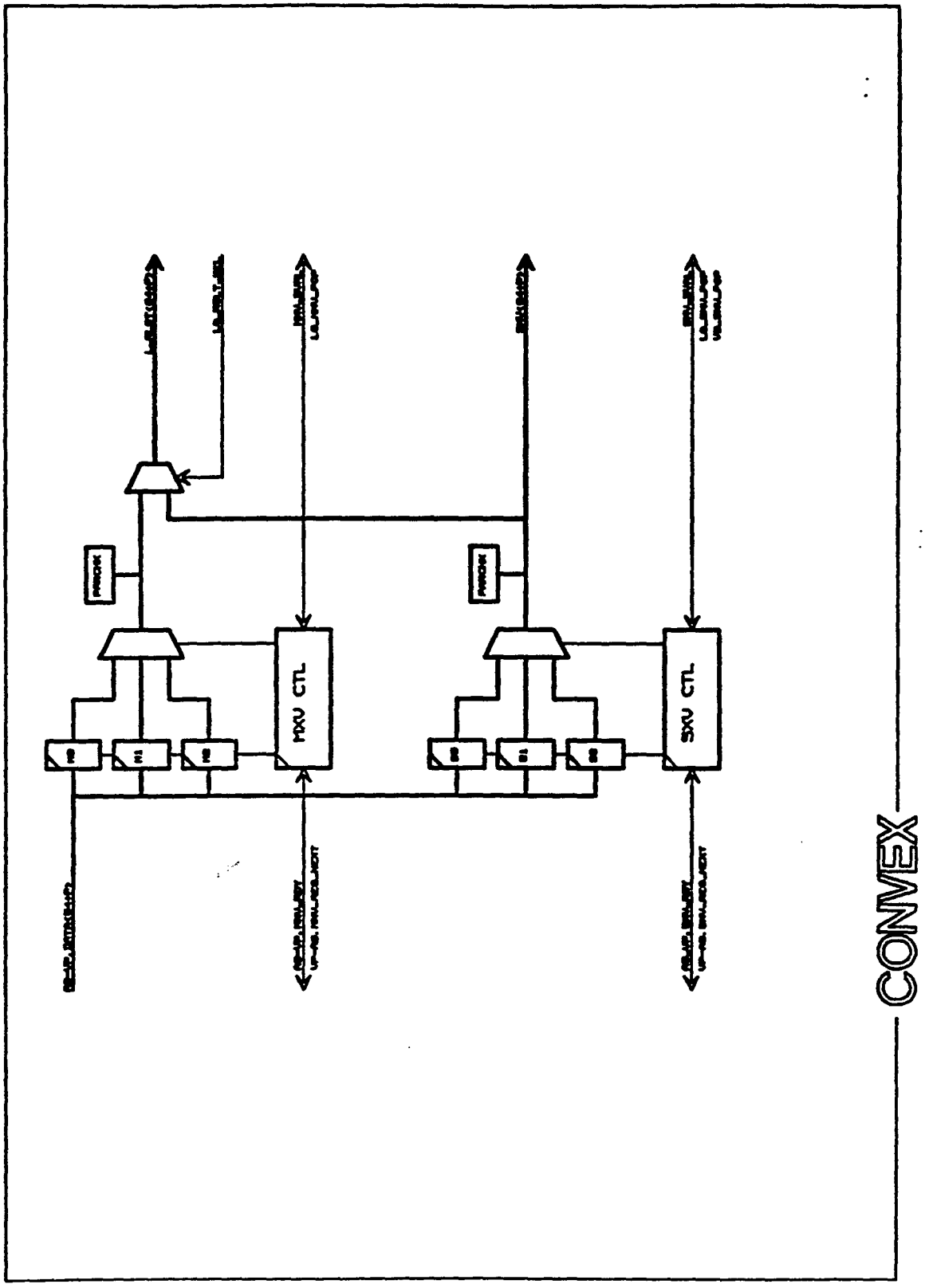


Figure 2–8: Pipe Controller

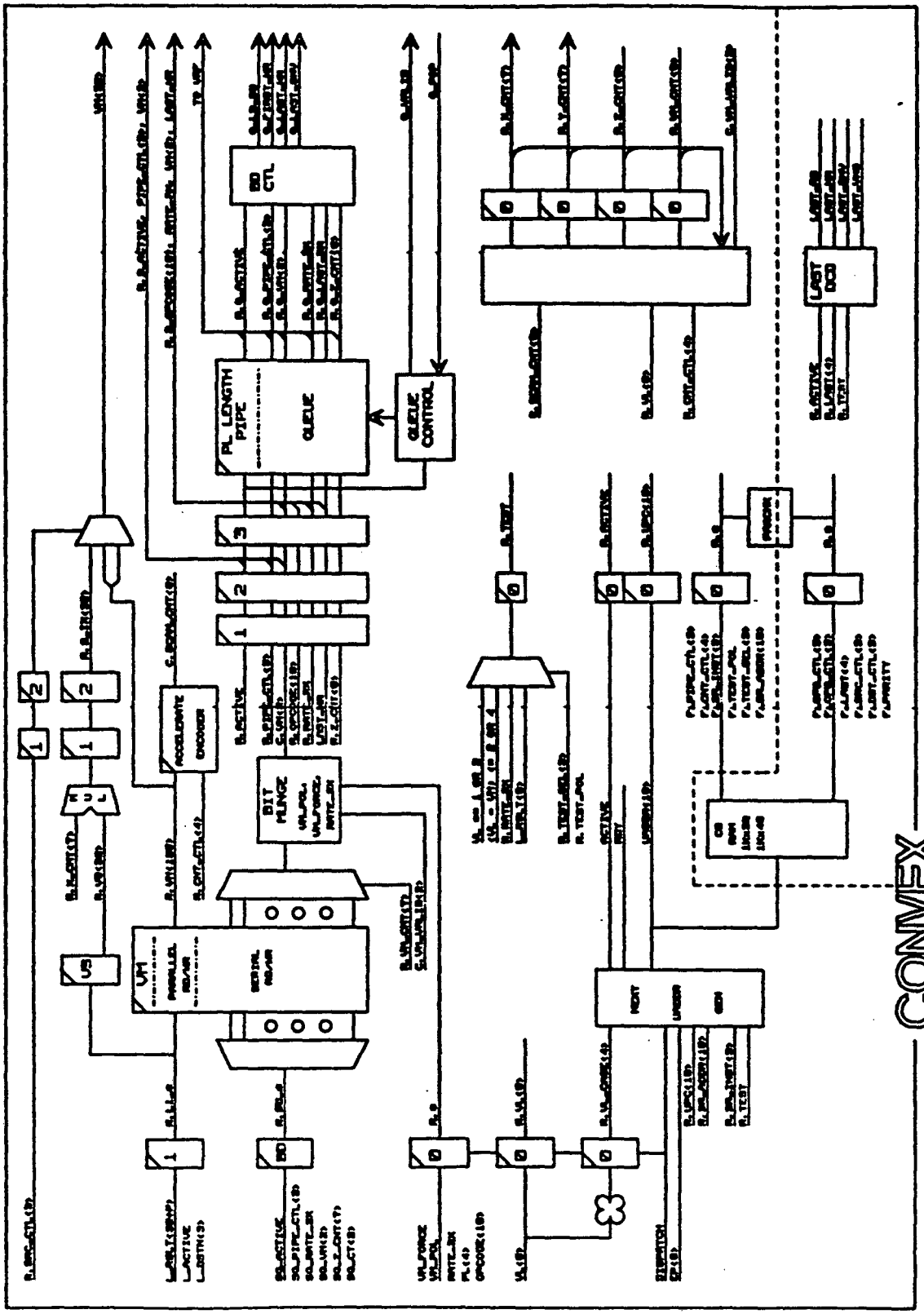


Figure 2-8
MICROCONTROLLER

compared against VL to determine when the instruction is complete, and when vm bits are valid. The VL register and the VM and Z counters are 8 bits wide. This allows using counts beyond 127 to occur without "wrapping" the address space, thus simplifying end condition detection. All counters may be loaded with the output of the VM accelerate encoder for accelerated operations under mask.

2.6.3 VM

Each pipe controller has its own VM register. Serial writes, parallel writes, and zeros of the VM register must write all three pipe's copies of VM. Each pipe reads the VM bit(s) selected by the VM counter and munges them into "active element" VM bits that are pushed through the control queue. Each pipe has the ability to preform "operate under mask" instructions by reading and writing only the vector elements whose corresponding VM bit active (one or zero). This is performed using the Accelerate Encoder to search the VM register for active bits and transfer their positions as counts to the vector element counters. The Accelerate Encoder requires two clocks to generate the address of the first active bit in the VM register.

The load microsequencer may parallel read VM and transfer it to the VRF and on to memory or the scalar unit.

2.6.4 Control Queue

Count and control information needed at different levels of the pipes are pushed through a control queue. Information propagates through the first three levels of the queue, controlling operand selection and function pipe operations along the way. The rest of the queue for most information is a FIFO that is pushed whenever r.2_active is asserted, and popped whenever Xn_pop is asserted. The active bit, however, propagates through a variable length pipe whose length is selected by the dispatched pipe length. In the add and multiply pipes the r.Xq_active bit is just connected to the Xn_pop signal, thus popping information from the queue whenever an active bit falls out of the queue. In the load pipe, r.Xq_active is unused. Instead, the backdoor controller generates lq_pop whenever data is ready from memory.

2.6.5 Backdoor Controller

All three backdoor controllers decode Xq_pipe_ctl to generate Xq_ld_bd, and to determine first and last write and last scalar to vector transfer. The load pipe backdoor controller (LBD) also matches data from scalar and memory with pipe_ctl codes being popped from the queue, and extends clocks until matches are made.

2.6.6 Load Pipe Functions

The load pipe controller has a copy of the vector stride (VS) register. The X vector element counter is multiplied by VS to generate indices to memory for vector loads, stores and store scalar extended under accelerated mask.

2.6.7 Implementation

All of the logic shown in Figure 2-8 that lies outside of the two dashed areas will be implemented in one 20K gate array. Three of the gate arrays will be used on the VP board, one for each pipe. Thus the gate array must implement the superset of the function needed on the three pipes. The control store RAMs, backdoor controllers and miscellaneous logic shown will be implemented in

discrete parts.

2.7 Clock Logic

All of the VP except for the external interfaces operate in lock step, with one common clock extend (clock hold). Three events can cause the clock to be extended.

1 The OSCTL body has data to transfer to the scalar processor or memory, and the required request is not asserted.
2 The LBD body needs non-first element data from memory or any data from the scalar processor and it is not ready.
3 The dividers within the MFP are being presented data and are unable to accept it.

110C TOTAL

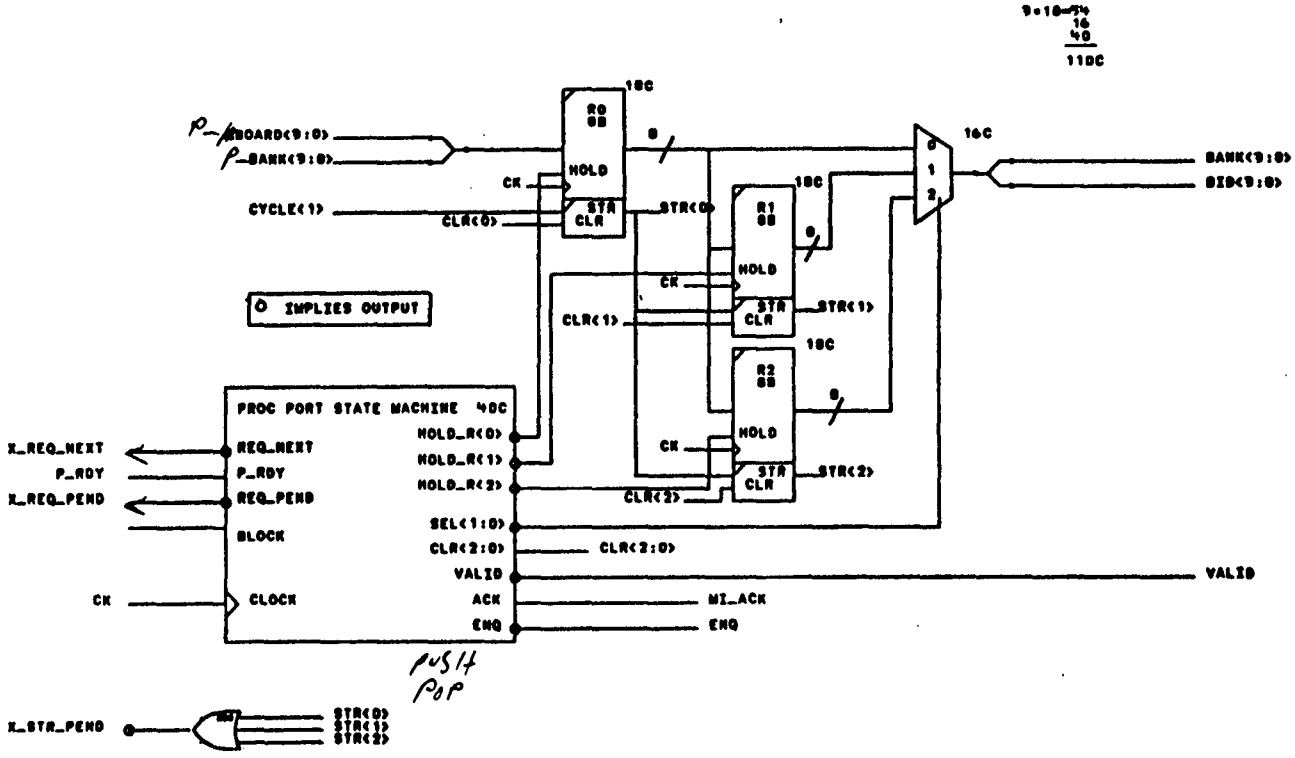


FIGURE 1-13

PROCESSOR INTERFACE

CONVEX

TITLE: PROCINT
DRAWING: 000-000000-000e n
REVISED: Mon Sep 12 18:59:52 1988
PAGE: 1 OF 1
ABBR: PROC INTERFACE
ENGR: GELSEY

54C TOTAL

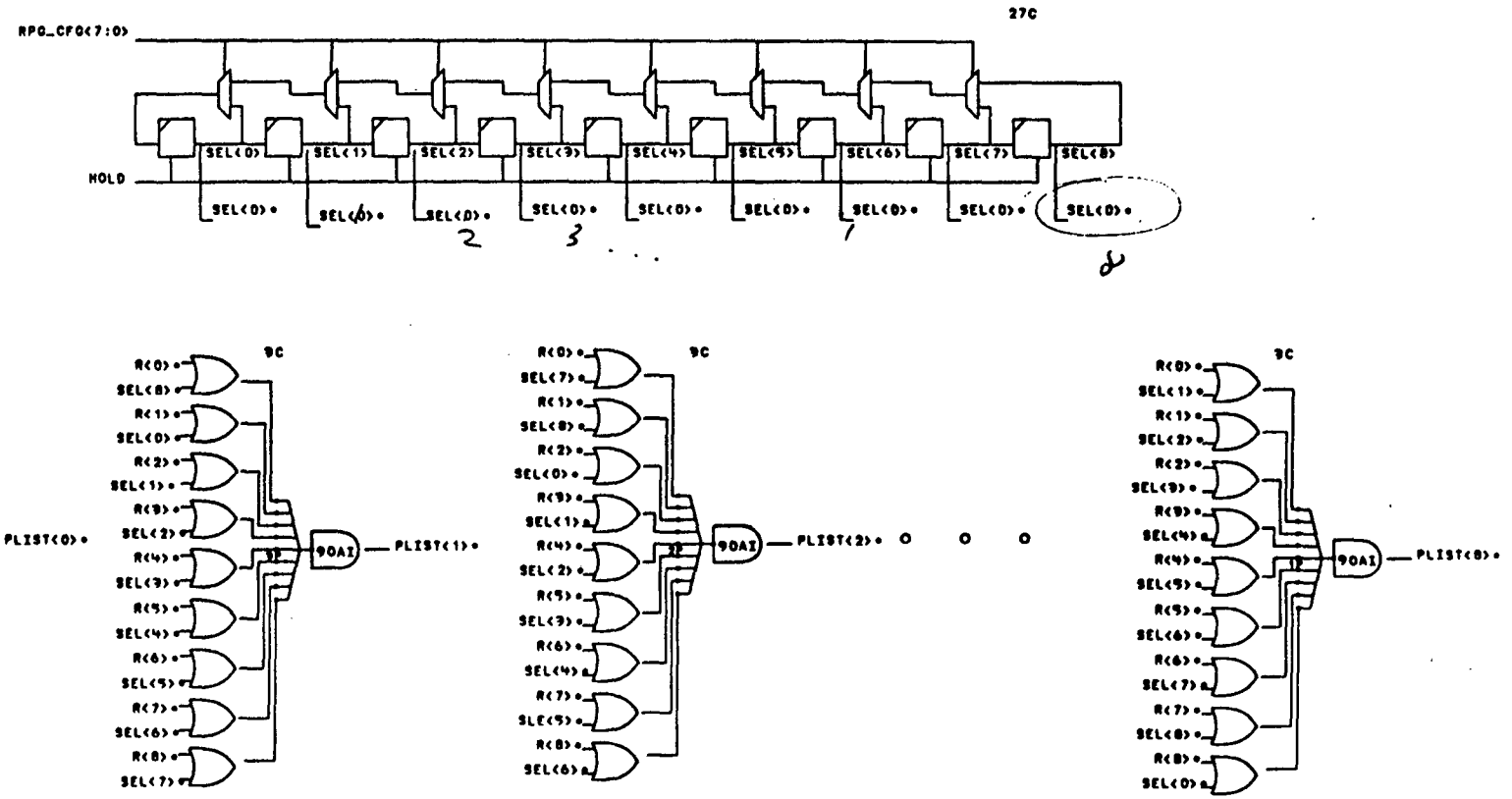


FIGURE 1-17

REQUEST PRIORITY GENERATOR

CONVEX

TITLE: REQPB			ABBR: REQ. PRIOR. QUE
DRAWING: 000-000000-000e A			ENGR: GELSEY
REVISED: Mon Sep 12 12:16:19 1988		PAGE:	1 OF 1