MicroVAX CPU CHIP DESIGN SPECIFICATION
DC 333 (21-20887-01)

Rev. 3.02 (WORKING DRAFT)

C O M P A N Y C O N F I D E N T I A L

REVISION HISTORY

REV	DATE	REASON
3.02	WORKING DRAFT	Pass 3 PG/LR updates.
3.01	Jul, 1984	Pass 2 PG updates.
3.00	Jan, 1984	Pass 1 PG updates.
2.00	May, 1983	New memory management.
1.02	Jan, 1983	Updates.
1.01	Dec, 1982	Updates.
1.00	Oct, 1982	Initial release.

1.0 INTRODUCTION

1.1 Scope

This document describes the MicroVAX CPU chip, a MOS/VLSI chip that implements a subset VAX compatible central processor. This specification describes the internal organization and operation of the chip. It does not describe the external interface and behavior of the chip. For further information, the applicable documents should be consulted.

1.2 Applicable Documents

VAX Architecture Standard (DEC Standard 032)
MicroVAX CPU Chip Engineering Specification
MicroVAX FPU Chip Engineering Specification
MicroVAX FPU Chip Design Specification

1.3 MicroVAX CPU Chip Features

The MicroVAX CPU chip is a 32-bit, virtual memory microprocessor. Implemented in ZMOS (double metal NMOS), the MicroVAX CPU chip is a high performance, low cost CPU for single board computers, single user workstations, low end systems, and other applications that do not need the flexibility of, or cannot afford the complexity of, the full VAX architecture. Its key features are:

1. Subset VAX data types. The MicroVAX CPU chip supports the following subset of the VAX data types: byte, word, longword, quadword, character string, and variable length bit field. Support for f_floating, d_floating, and g_floating is available via an external floating point unit. Support for the remaining VAX data types can be provided by macrocode emulation.

2. Subset VAX instruction set. The MicroVAX CPU chip implements the following subset of the VAX instruction set: integer and logical, address, variable length bit field, control, procedure call, miscellaneous, queue, MOVC3/MOVC5, and operating system support. Floating point is implemented via an external floating point unit. The remaining VAX instructions can be implemented via macrocode emulation (the MicroVAX CPU chip provides microcode assists for the emulation of the character string, decimal string, EDITPC, and CRC instructions).

3. Full VAX memory management. The MicroVAX CPU chip includes a demand paged memory management unit which is fully compatible with VAX memory management. System space addresses are virtually mapped through single level page tables, process space addresses through double level page tables.

4. Industry standard external interface. The MicroVAX CPU chip's external interface is a 32-bit extension of the industry standard microprocessor interface. The MicroVAX CPU chip can be easily interfaced to industry peripheral chips from Motorola, National, and other vendors.

5. Large virtual and physical address space. The MicroVAX CPU chip supports four gigabytes (2**32) of virtual memory, and one gigabyte (2**30) of physical memory.

6. High performance. At its maximum frequency, the MicroVAX CPU chip achieves a 200 nsec microcycle and a 400 nsec I/O cycle.

7. Single package. The MicroVAX CPU chip is packaged in a standard 68-pin surface mounted chip carrier and requires no special clock generator or support chips.

1.4 MicroVAX CPU Chip Sections

The MicroVAX CPU chip is composed of the following sections.

1.4.1 Instruction (I) Box -

The Instruction (I) Box contains the instruction buffer, initial decode PLA, and associated logic. It prefetches the instruction stream, generates microprogram addresses, and provides instruction data to the E Box.

1.4.2 Execution (E) Box -

The Execution (E) Box contains the VAX general registers, microcode working registers, ALU, shifter, and other facilities for the efficient emulation of the MicroVAX instruction set.

1.4.3 Memory (M) Box -

The Memory (M) Box contains the translation buffer, length check registers, and associated logic. It converts program generated virtual addresses to physical addresses, when necessary, and handles translation exceptions.

1.4.4 DAL Interface -

The DAL Interface connects the active chip logic to the external environment. It includes the incoming and outgoing data latches, rotators, and swappers, the pin signal generation logic, and the external operation control sequencer.

1.4.5 Microsequencer -

The Microsequencer determines the address of the next microword to be fetched and executed from the Control Store. It also oversees the generation and execution of microtraps.

1.4.6 Interrupts -

The Interrupt logic mediates hardware interrupt requests against the current IPL and generates an interrupt request to the I Box, if necessary.

1.4.7 Control Store -

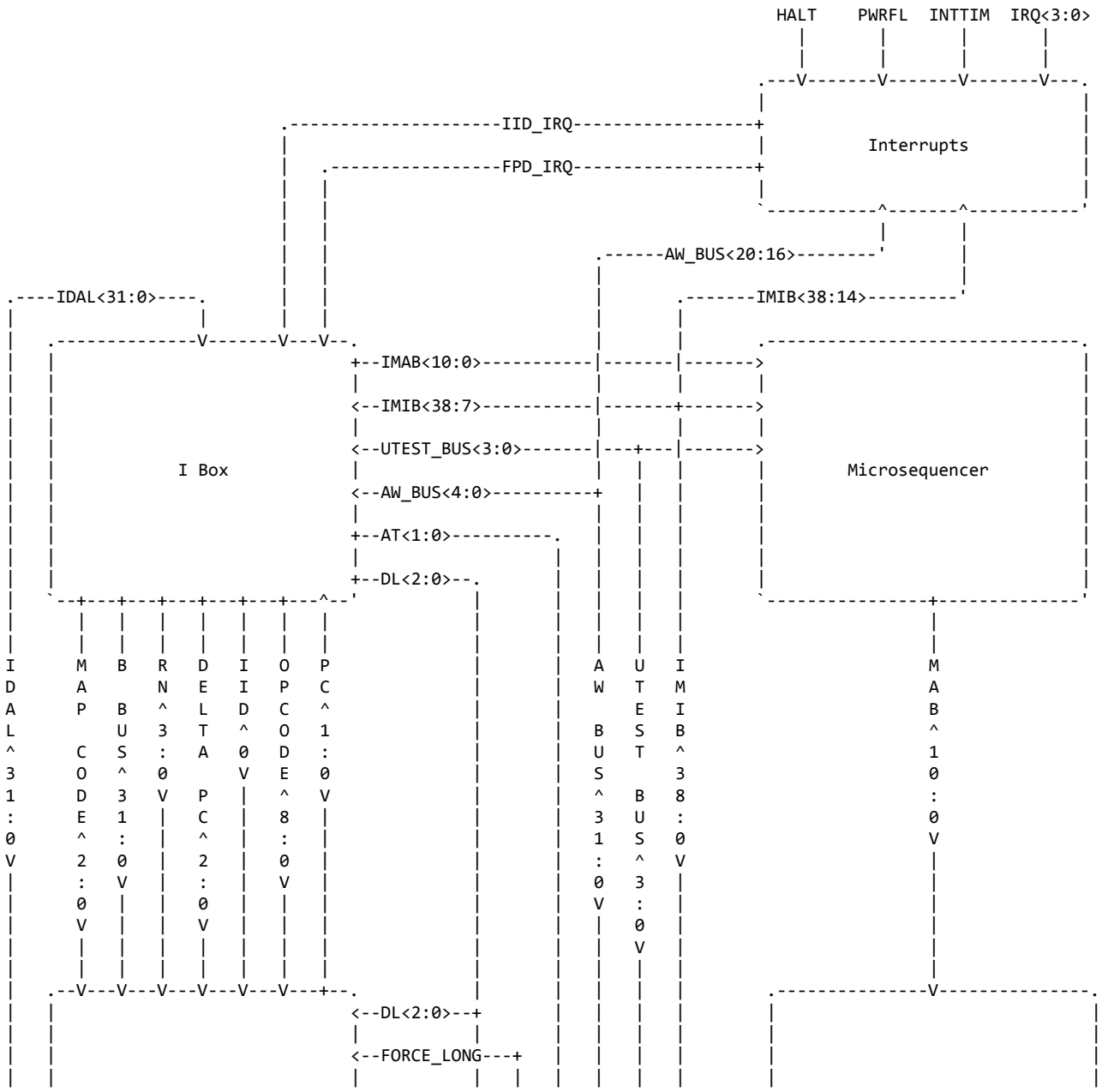
The Control Store contains 1600 x 39 words of microcode which direct all operations in the chip.

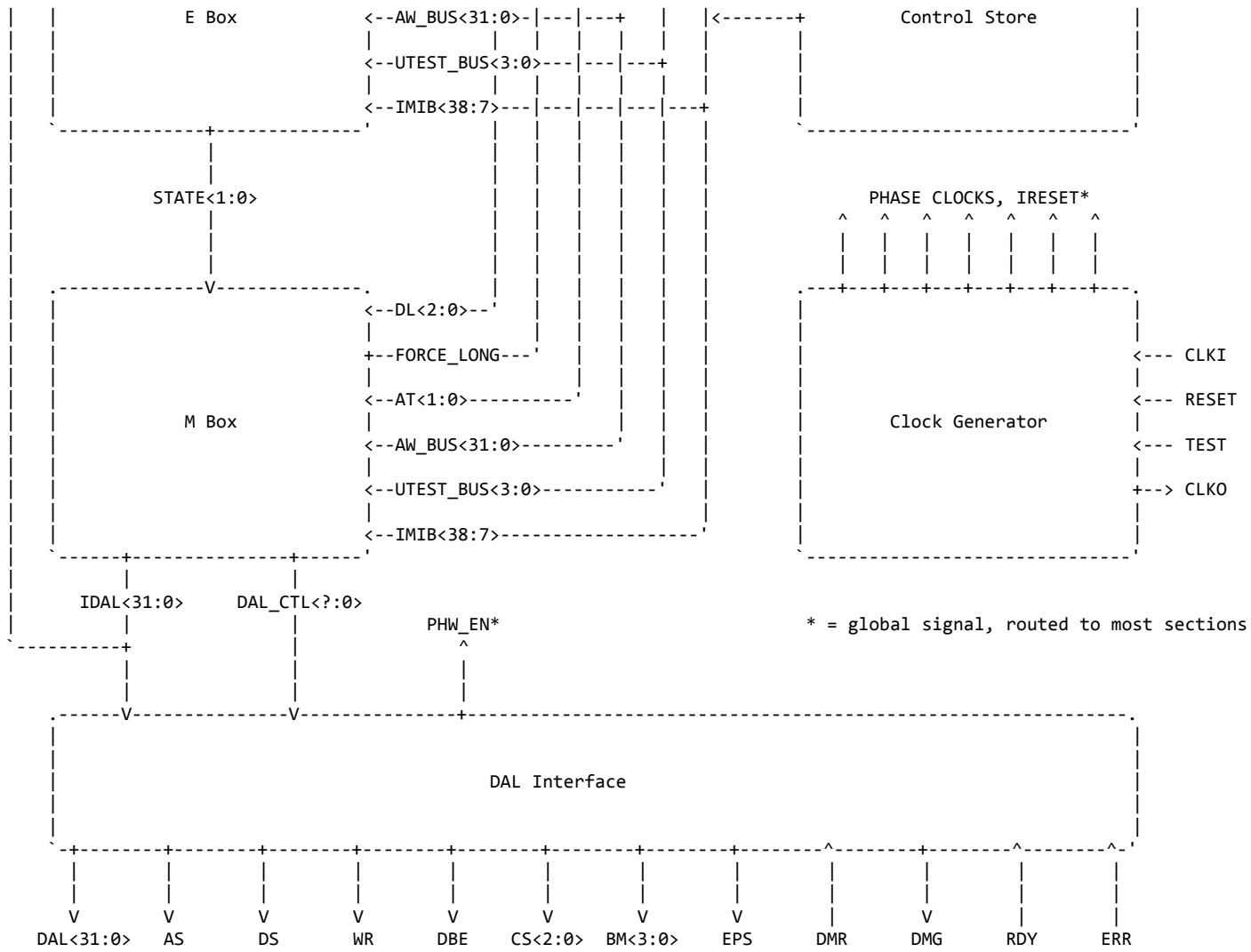
1.4.8 Clock Logic -

The clock logic converts an external, double frequency clock reference into the internal phase clocks required by the chip logic.

1.5 Block Diagram

MicroVAX CPU Chip Block Diagram




E Box
<--AW_BUS<31:0>
<--UTEST_BUS<3:0>
<--IMIB<38:7>
Control Store
STATE<1:0>
PHASE CLOCKS, IRESET*
<--DL<2:0>
+--FORCE_LONG
<--AT<1:0>
M Box
<--AW_BUS<31:0>
<--UTEST_BUS<3:0>
<--IMIB<38:7>
Clock Generator
<--- CLKI
<--- RESET
<--- TEST
+--> CLKO
IDAL<31:0>
DAL_CTL<?:0>
PHW_EN*
* = global signal, routed to most sections
DAL Interface
DAL<31:0>
AS
DS
WR
DBE
CS<2:0>
BM<3:0>
EPS
DMR
DMG
RDY
ERR

2.0 INSTRUCTION (I) BOX

The I Box handles instruction prefetching and the sequencing of instruction execution. The sequencing begins when the Initial Instruction Decode (IID) signal is given in the microinstruction (specifically, by the IID codes in the Branch Condition Select (BCS) field). This signal causes the I Box to examine the opcode and first specifier of the new macroinstruction and drive the Next Address Bus (NAB) with a microaddress to start the microprogram for the instruction. If the instruction has specifiers, the microaddress is based on the mode of the first specifier, and will take the microcode directly to the First Specifier Decode (FSD) routines. If there are no specifiers, execution flows for the instruction are entered immediately.

The FSD flows end with a microinstruction that issues a Next Specifier Decode (NSD) command, again with codes in the BCS field. The I Box again forces an address that starts up microprograms for decoding the second specifier or for executing the instruction. Following Second Specifier Decode (SSD), the I Box provides the microaddress for the instruction's execution flows.

If there are further specifiers to be decoded after the execution flows for an instruction have been entered, a SPEC DECODE command is issued in the MISC field of a microinstruction. In response to this, the I Box drives NAB<4:1> with a value based on the mode of the next specifier, and the rest of the microaddress is provided by the same microinstruction's JUMP address field. The microcode thus addressed decodes the next specifier and ends with a RETURN (in the BCS field) to the execution flows from which it was called. When the execution of an instruction is completed, its final microinstruction issues an IID to start the next instruction.

While the chip is executing one instruction, the I Box tries to fetch the next ones using free cycles on the DAL. If there are branches in the program flow, the prefetched instructions are thrown away. If for any reason the data has not come in by the time the I Box needs it, the I Box forces a branch in the microprogram flow. The microcode then loops until the data comes in, or goes off to memory management flows.

The I Box has four main parts: the Prefetcher, the IPLA, the Microaddress Generator, and miscellaneous registers and control.

2.1 Prefetcher

The Prefetcher has two sections, the Data Path and the Controller.

2.1.1 Prefetcher Data Path -

The I Box Prefetcher Data Path handles the actual macroinstructions. It buffers the prefetched longwords from memory, rotates the instructions to bring the opcode to the front, and stores literals and displacements for the E Box. It is controlled by the Prefetch Controller and consists of

the following blocks.

2.1.1.1 Instruction Prefetch Stack -

The Instruction Prefetch Stack consists of two longword registers, each corresponding to an aligned longword in memory. The registers comprise a two-entry FIFO stack or queue. Data is loaded into the end of the queue from the the IDAL and stored in whichever empty register is closest to the head of the queue. If the entire queue is empty the data is loaded into the first register; if the first register contains valid data, the data on the IDAL goes into the second register. When a full longword has been removed from the head of the queue, the contents of the second register move into the first.

2.1.1.2 Instruction Byte Rotator -

The Instruction Byte Rotator can select up to four contiguous bytes in the Prefetch Stack (for example, an opcode, a specifier, and two bytes of data) starting at any byte in the lowest longword. The position of the starting byte is specified by the IB Pointer.

2.1.1.3 Instruction Data (ID) Register -

This four-byte register is the mechanism by which the E Box data path is passed data from the instruction stream (displacements, literals, etc.). Data is loaded from the Byte Rotator, either automatically by the I Box or explicitly by the microinstruction MISC field, and sign extended to longword in the same cycle it is loaded.

The ID register is automatically loaded by the I Box when:

- The I Box Microaddress Generator detects that the opcode is a branch instruction. The ID register is loaded with the byte or word branch displacement.
- The Microaddress Generator detects that the specifier is a short literal (i.e. the specifier mode nibble is 0-3). The specifier itself, containing the operand, is loaded.
- The Microaddress Generator detects that the specifier mode is byte or word displacement (specifier mode A, B, C, or D). The following bytes, containing the byte or word displacement, are loaded.

Explicit loads of the ID register are accomplished by the microinstruction MISC field commands LOAD ID (with the data length based on the Data Length register) and LOAD ID LONG (with the data length forced to LONG). Longword displacements (for specifier modes E and F) are always loaded using an explicit MISC field command, because the Byte Rotator cannot provide the specifier byte and four bytes of data in the same cycle.

The I Box drives two bits on the Next Address Bus (NAB) to indicate whether an ID register load was successful.

NAB<2>	NAB<1>	
0	0	load was successful
1	0	not enough bytes in stack, prefetching not halted
1	1	not enough bytes in stack, prefetching halted

The other bits of the microaddress are unaffected. Jumps and branches can be done by the same microinstruction as a LOAD ID, but the final addresses must be allocated allowing for the I Box to drive NAB<2:1>.

The ID register is read by the E Box as Miscellaneous Register MR[3].

2.1.2 Prefetch Controller -

The DAL Interface must decide, based on information from the I Box, whether to fetch I Stream data for the Prefetch Stack when there is no D Stream access in progress. If prefetching has not been halted and at least one longword is invalid or expected to become invalid during the current microcycle, the Prefetch Controller will signal the DAL Interface that it wants I Stream data by asserting the IB_REQ line.

If the I Stream read results in a TB miss or there is some other problem with the PTE, the Hardware Prefetch Halt Bit is set and no more prefetching is done. This is done to prevent prefetching from interfering with memory management operations. When the Prefetch Stack eventually runs out of valid data, a special microaddress is forced onto the NAB; a microcode routine then handles the problem. The I Box does not react immediately to a prefetching error because the prefetched data may not be used. The Hardware Prefetch Halt Bit is cleared by the RESTART PREFETCH MISC field, and as a side effect of the BCS fields that load VIBA and PC.

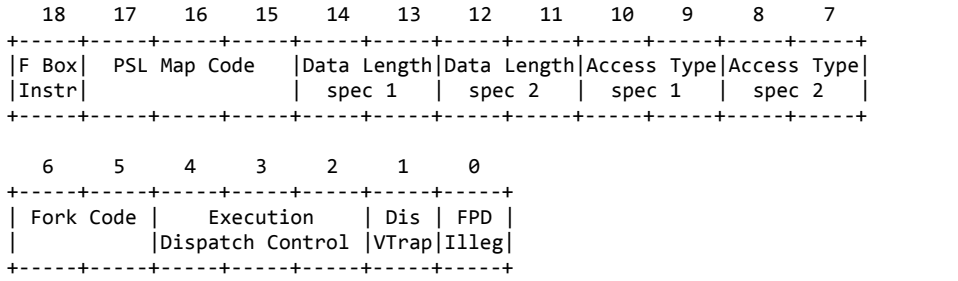
The microcode can also explicitly stop prefetching by setting the Microcode Prefetch Halt Bit in the I Box. This is a different bit from the Hardware Prefetch Halt Bit. This microcode-controllable bit can be set and cleared by the MISC field commands DISABLE PREFETCH and ENABLE PREFETCH. It is also cleared by the RESTART PREFETCH MISC field, and as a side effect of the BCS fields that load VIBA and PC.

A microinstruction with DISABLE PREFETCH in the MISC field and LOAD VIBA AND PC in the BCS field results in the microcode halt bit being set and the hardware halt bit being cleared.

A two bit register called the IB Pointer is kept to point to the next valid byte in the lowest longword of the Prefetch Stack. As data is drawn out of the stack by the Microaddress Generator, this pointer is incremented by Delta PC. When the macroinstruction stream branches, nothing in the stack is usable. It is flushed as a side effect of the BCS branches that load VIBA and PC. Prefetching is started up again, and the IB Pointer is set pointing to the start of the new instruction (i.e. bits <1:0> of the PC).

2.2 The Instruction PLA (IPLA)

The IPLA parses the opcode. It has as inputs the eight-bit opcode and a ninth bit called XFD, set by hardware if the previous opcode was FD (meaning that this is a two byte opcode). From this information, the IPLA generates the following 23 bits of data:



<18> The F Box Instruction bit is true if the opcode is for an F, D, or G floating point instruction. If this bit is true and there is no FPU present, an illegal opcode trap is taken. The microcode also tests this bit during the decode of short literal mode specifiers and, if set, rearranges the literal into floating point format.

<17:15> The PSL Map Code bits tell the E Box how to set the PSL condition code bits for the current opcode.

<14:7> The IPLA has a multiplexor on some of its outputs. This mux selects one of the two Data Length fields and one of the two Access Type fields, depending upon which specifier is being evaluated. This data is loaded into the AT/DL Register during IID and NSD microcycles.

<6:5> The Fork Code is an encoding of the Fork Types, which control the next microaddress sent out at NSD. (The codes are detailed in the Tables section.) If the Fork Type is S then the microcode will go to routines that decode the second specifier. If O then the hardware will detect if the second specifier is register mode and if so will go directly to execution flows (optimize), since no specifier decoding needs to be done. This saves a cycle in many instructions. If E then the microcode goes unconditionally to execution flows. The Fork Type I is used to denote an illegal opcode. Since many illegal opcodes do not have IPLA terms, the Fork Code bits default to the code for Fork Type I.

<4:2> The Execution Dispatch Control outputs from the IPLA are sent to the Microaddress Generator where they are used to form a mask that is ANDed with the opcode to produce a microaddress for the instruction's execution flows. Through the use of these masks, different macroinstructions can share the same microcode. If the instructions need to be separated further down the flow, there is a microbranch provided that cases on

the bottom four bits of the opcode. See the Tables section for the specific masks used.

<1:0> The last two bits are for the two microtraps asserted by the I Box: optimized integer overflow microtrap and trap on illegal opcode. The traps are asserted at the start of a macroinstruction, in the cycle following a executed IID (ie, IID AND PHI_WRITE).

The integer overflow trap exists so that the microcode does not have to waste two cycles checking to see if an arithmetic operation has overflowed. Instead a trap is taken the cycle after the end of the instruction if:

- the V bit (overflow) in the PSL is set, and
- the IV bit (overflow trap enable) in the PSL is set, and
- the IPLA disable overflow trap bit is not asserted for this opcode.

The disable overflow trap bit defaults to being set to prevent undecoded (i.e. illegal) opcodes from taking overflow traps.

The end of an instruction execution is indicated by IID AND PHI_WRITE. If the IID takes an exceptional branch like an interrupt the integer overflow trap will still be asserted. However, if an exception like a page fault occurs during the instruction, then microcode can clear any pending overflow traps with the SPECIAL MISC3 instruction CLEAR VAX TRAP REQUEST.

An illegal opcode trap occurs if the instruction is a floating point type and there is no FPU present in the system, if the Fork Code bits from the IPLA indicate an illegal opcode (Fork Type I), or if the FPD Illegal bit from the IPLA is asserted and the First Part Done bit in the PSL is also set. Unlike the overflow trap, the illegal opcode trap will not be asserted if IID takes an exceptional branch like an interrupt.

The IPLA has 72 terms and an access time of 100 ns.

2.2.1 IPLA Reducer -

For testability, there is a reducer at the outputs of the IPLA. The reducer uses an XOR and shift combination to checksum the IPLA during test mode. The shift out is routed to the M Box and conditionally driven on Control Status pin CS<2>.

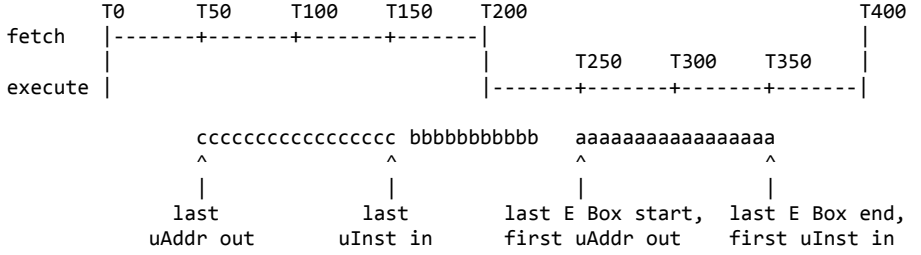
2.3 Microaddress Generator

The Microaddress Generator produces microaddresses for entering the microcode flows at various stages of a macroinstruction. The microaddresses may be generated at four times: during IID, during NSD, during SPEC DECODE, and during LOAD ID operations.

2.3.1 Initial Instruction Decode (IID) Branch -

The IID branch is described in three parts: exceptions/traps/interrupts pending, opcode dispatch, and specifier dispatch.

An IID command is given at the end of an instruction routine. It causes a microcode dispatch based on the next opcode. It is encoded into several of the microbranch commands, and may be conditional. Its use spans two microcycles, the microinstruction fetch cycle and the execute cycle:



ccc - Control Store fetch, 100 ns, T50 to T150 of fetch cycle
bbb - IID branch decision, 50 ns, T160 of fetch cycle to T25 of execute cycle
aaa - E Box execution, 100 ns, T50 to T150 of execute cycle

At time T50 the address for the last microinstruction of a macroinstruction execution sequence is sent to the Control Store. The microinstruction, which includes the code for the final E Box operation as well as the IID branch code, is driven by the Control Store beginning at T150 and is valid at the I Box Decoder at about T160. The IID branch decision is made at time bbb above, before the E Box operation is executed at time aaa. If the branch condition is true (that is, this actually is the last E Box operation required for the current macroinstruction), the I Box produces the microaddress for the first microinstruction of the routine for the next macroinstruction, which is sent to the Control Store at T250. From T250 to T350 the last E Box operation of the previous macroinstruction is being executed while the first microinstruction of the next macroinstruction is being fetched. In subsequent microcycles, other branches (NSD to SSD or Execute, or SPEC DECODE) can be made based on bits in the IPLA. The IPLA bits arrive at about T375, too late to control the IID branch calculation. Illegal opcodes are detected by having no IPLA row selected, or by the IPLA F Box Instruction output being true when the FPU Present bit is clear, or by the IPLA FPD output being true when PSL<FPD> is set. Since these are not determined until T375 (after IID), illegal opcodes generate a microtrap during the execute cycle after IID.

2.3.1.1 IID Exception Dispatch -

These exceptional conditions are examined at IID. They force special microaddresses and cause the opcode to be ignored.

VAX ARITH TRAP REQ (MISC3)	INTERRUPT PENDING	PSL<TP>	Prefetch Halted	IB DRY	uADDR (hex)	
1	x	x	x	x	114	arith trap
0	1	x	x	x	10E	interrupt
0	0	1	x	x	10C	trace trap pending
0	0	0	1	x	106	prefetch halted, not enough bytes in stack
0	0	0	0	1	104	prefetch not halted, not enough bytes in stack

The arithmetic trap bit is set and cleared by SPECIAL microinstructions.

MICROCODE NOTE

An IID and SET VAX TRAP may not be done in the same microinstruction; nor may an IID and CLEAR VAX TRAP. Also, a SET VAX TRAP or CLEAR VAX TRAP microinstruction may not be followed immediately by an IID.

Interrupts are signalled by the Interrupt Logic over the IID_IRQ line.

Trace trap pending is a bit in the PSL indicating that a debugging trap is to be taken.

Prefetching is halted by problems that prevent determining if an instruction read from a page is allowed, e.g., TB miss or TB hit with Access Violation or Translation Not Valid. Any of the exceptions will disable the illegal opcode microtrap.

There are four IB.DRY locations reachable from IID: 104 and 106, and 144 and 146 within FSD. 104 and 106 are the only ones used at IID, while 144 and 146 are used when the FSD flows are called by SPEC DECODE.

2.3.1.2 IID Opcode Dispatch -

If there are no exceptions pending at IID, an address based on the opcode is driven out:

PSL <FPD>	IB[PC] <7:0>	uADDR (hex)	
0	0000 0xxx	130..13E	zero specifiers HALT..SVPCTX
0	00x1 0000	124	BSBB..BSBW
0	00x1 0001	120	BRB..BRW
0	0001 001x	120	BNEQ..BEQL
0	0001 010x	120	BGTR..BLEQ
0	0001 1xxx	120	BGEQ..BCS
0	1111 1100	128	XFC
x	1111 1101	12A	XFD
1	not FD (hex)	12C	
0	xxxx xxxx	140..15E	all others go to FSD, using SPEC MAP(IB[PC+1])

The SPEC MAP bits come from logic that operates on the specifier. The offsets are given in the section on Specifier Dispatches.

If the opcode loaded at IID is FD (hex), the microcode performs a second IID, which picks up the second byte of the opcode and dispatches either to FSD or to one of the IB.DRY locations. The first IID (when IB[PC]=FD) sets the 9-bit opcode register to 0FD and adds 1 to the PC. The second IID sets the 9-bit opcode register to 1xx, where xx is the second opcode byte and increments PC by 1 to 4, if it does not go to IB.DRY. If the second IID does go to IB.DRY, it leaves the opcode at 0FD, and leaves the PC unchanged. The first IID sets the Backup PC (BPC) equal to the PC; the second one does not change it. If a page fault occurs after parsing the FD, but before picking up the second opcode byte and first specifier, the microcode backs out of the instruction in the normal way, backing up PC to be BPC, i.e. pointing at the FD again.

Instructions without specifiers go immediately to execute flows, while those with specifiers go to First Specifier Decode (FSD).

2.3.1.3 Specifier Dispatches -

Specifier dispatches cause 16-way cases in the microcode flow. Some of them also do a jump to fixed locations. FSD does its case starting at a base address of 140; Second Specifier Decode (an NSD with a fork type of S) cases at 180. The MISC field SPEC DECODE does its case at whatever base address the JUMP field specifies. The same mapping for microaddress bits <4:1> is used with all bases.

SPEC MAP

Specifier<7:4> (hex)	NAB<4:1> (hex) R=0..E	NAB<4:1> (hex) R=F	Delta PC (add one for IID.FSD to skip over opcode too)	
0,1,2,3	0	0	1	S^#
4	1	7	1 *	[R] [PC] illegal
dry, not halted	2	2	0	stack dry, prefetching not halted
dry and halted	3	3	0	stack dry, prefetching halted
5	5	7	1	R PC illegal
6	6	7	1	(R) (PC) illegal
7	8	7	1	-(R) -(PC) illegal
8	9	4	1	(R)+ (PC)+
9	A	B	1	@(R)+ @(PC)+
A,C	C	C	2,3	D(R) D(PC)
E	D	D	1 **	D(R) D(R) longword
B,D	E	E	2,3	@D(R) @D(PC)
F	F	F	1 **	@D(R) @D(R) longword

* The index mode prefix is treated as a one-byte specifier. The embedded specifier is parsed (and PC incremented, etc.) by a microcode SPEC DECODE.

** The Byte Rotator cannot pull off a longword displacement and a specifier at the same time. The displacement for the E and F specifiers must be obtained with a subsequent LOAD ID command.

2.3.2 Next Specifier Decode -

The NSD signal comes from a number of codes in the Branch Condition Select field of the microinstruction. In executing these forks, the Microaddress Generator examines the Fork Type information from the IPLA and the current specifier.

Fork Type	FT Label	Next Address Bus 10	9	8	7	6	5	4	3	2	1	0	Address Range
SSD	S	0	0	1	1	0	0	SPEC MAP				0	180 - 19E
Optimizations	O	1	0	EXECUTION ADDRESS								0	400 - 5FE
Execute	E	0	1	EXECUTION ADDRESS								0	200 - 3FE

The Execution Address is derived from the nine-bit opcode and the three-bit Execution Dispatch Control code from the IPLA. The Execution Dispatch Control code is decoded by a simple logic network to generate an eight-bit mask. Opcode<8> is OR'ed into Opcode<5> and the eight-bit result is AND'ed with the mask to form the eight execution dispatch address bits.

The Optimizations are for instructions with register mode specifiers. Because these do not need to go through a specifier decode microroutine,

they go directly to execution flows.

2.3.3 Specifer Decode -

If the MISC field of a microinstruction contains the SPEC DECODE signal, the I Box drives bits <4:1> of the Next Address Bus with the same bit patterns as in the NSD SSD dispatch. (Bit <0> is always 0.) This gives the microcode a case branch on the specifier type.

2.3.4 Delta PC Logic -

The Delta PC logic is integrated with the microaddress generation. It generates the three bit value to be added to the PC when the next piece of instruction data is used. Whenever data is not being requested from the Prefetch Stack, it must be zero. Delta PC is used by the PC function of the E Box data path, and by the Prefetch Controller. The controller uses it to increment the IB Pointer that specifies the next byte to be used in the Prefetch Stack.

Condition	Delta PC
IID forks:	
Exceptions	0
FPD	1
Opc = Branch byte disp.	2
Opc = Branch word disp.	3
Opc = FD (hex)	1
Zero operand opc.	1
FSD	see note
NSD forks:	
SSD	see note
Optimize	1
Execute	0
SPEC DECODE	see note
LOAD ID:	
DL=0	1
DL=1	2
DL=2	4
DL=other	4
LOAD ID LONG:	4

NOTE:
In these cases the amount of data taken out of the Prefetch Stack depends on the specifier mode.

Spec Mode	Spec Code	Delta PC
byte disp.	A	2
byte disp. deferred	B	2
word disp.	C	3
word disp. deferred	D	3
longword disp.	E	1
longword disp. def.	F	1
other	0-9	1

If this is an IID fork, delta PC is increased by 1 to account for the opcode.

2.4 Miscellaneous Functions

2.4.1 Opcode And FD Bit Register -

The nine-bit Opcode Register holds the opcode for the duration of the macroinstruction execution. It consists of the eight-bit opcode and a ninth bit (called XFD) that indicates an extended opcode (first byte is FD). The Opcode Register is loaded on every IID, and is the opcode input to the IPLA. The XFD bit is loaded from the previous opcode byte on every IID, and cleared as a side effect of the BCS fields that load VIBA and PC. A case branch is provided on the bottom four bits of the opcode.

The nine-bit Opcode Register may be read (but not written) when addressed as T[E] by a BASIC microinstruction with Source/Destination code equal to C or E, or by a MEM REQ or FPU XFER microinstruction with Source/Destination code equal to 1.

MICROCODE NOTE

A read T[E] and an IID may not be done in the same microinstruction.

2.4.2 Register Number Latch (RN) -

The Register Number Latch holds the GPR address of the specifier last decoded. It can be operated upon under control of the MISC field or the SPECIAL MISC1 field. It is also used with the RLOG facility in the E Box data path section. When RLOG is pushed, RN supplies four bits of the data that is pushed. When RLOG is popped, RN is loaded from that facility.

RN is also loaded during IID, non-execute NSD forks, and SPEC DECODE, with bits <3:0> of the specifier.

MISC	Operation
1	RN is incremented by 1
2	RN is decremented by 1
3	RN is loaded from SC
1E	RN <- SPEC<3:0>

MISC1	Operation
1	RN is loaded from RLOG

BCS	
IID	RN <- SPEC<3:0>
NSD and	(FT=SSD or FT=Optimize) RN <- SPEC<3:0>

MICROCODE NOTE

During execution, it is normal for conflicting commands to be sent from the I Box and microcode. When these cases occur, the I Box-requested operation is executed and the microcode-requested operation is ignored. If, for example, the MISC field says to increment RN, but the Branch Condition Select field says do a Second Specifer Dispatch, then RN will be loaded with the number of the second specifier.

2.4.3 RMODE -

This bit indicates that the addressing mode of the specifier under decode is register. It is used at the end of instructions like INC to determine if the destination is a GPR. It is loaded during IID, during NSD if the fork type is not Execute, and during SPEC DECODE.

2.4.4 FPU Present -

During power up initialization, the microcode must find out whether the system contains a Floating Point Unit. If so, it will set this bit, and floating point instructions will cause a normal FSD dispatch at IID. If this bit is clear and the IPLA entry for an instruction specifies F Box execution, an illegal opcode microtrap occurs. This bit is also tested by the microcode in CVT Integer to Floating instructions, because these are not identified by the IPLA as floating point instructions but still must trap if no FPU is present.

2.4.5 VAX Trap Request -

When the microcode detects an arithmetic exception like integer overflow it can set this bit with the SPECIAL MISC3 microinstruction SET VAX TRAP REQUEST. Then on the next IID a special branch will be forced by the I Box. The handler routine can clear the bit with the SPECIAL MISC3 microinstruction CLEAR VAX TRAP REQUEST.

2.4.6 Specifier Counter -

This facility indicates which specifier is presently under decode. It is cleared by IID, and incremented by NSD and SPEC DECODE. It is only used to count the first two specifiers, since the IPLA contains the data lengths, access types, and fork codes for only two specifers.

2.4.7 Data Length And Access Type -

The data length and access type of the specifier presently being worked on are kept in the AT/DL register. This register may be read as PR[B] with the access type in bits <4:3> and the data length in bits <2:0>. The access type portion of the AT/DL register may be written in the following ways:

- Automatically with a value from the IPLA one cycle after IID or on NSD (for the first two specifiers only).
- Explicitly from the AW_Bus, addressed as PR[B]<4:3>.

The codes for the access type are:

0	Read Source
1	Modify Source
2	Address Source
3	Field Source

The data length portion of the AT/DL register may be written in the following ways:

- Automatically with a value from the IPLA one cycle after IID or on NSD (for the first two specifiers only).
- Explicitly from the AW_Bus, addressed as PR[B]<2:0>.
- Forced by the MISC field.

The codes for the data lengths are (bit <2> is always 0):

0	BYTE
1	WORD
2	LONG
3	QUAD

2.4.8 PSL Bits -

The following PSL bits are stored in the I Box:

< 4>	T
< 5>	IV
<27>	FPD
<30>	TP

All four bits are loaded by an MXPR write of PSL.HWRE. The T and TP bits may be read as part of PR[C].

MICROCODE NOTE

An MXPR write of PSL.HWRE can not occur in the same microinstruction as an IID, nor, if <TP> or <T> were changed, can it be immediately followed by an IID microinstruction.

2.4.9 Execution_Started -

The Execution_Started flip-flop is set by the I Box when an NSD dispatch to execution flows is made. It is tested by several of the Branch Conditions to cause either a microsubroutine RETURN (if set) or NSD (if clear). This permits the same specifier flows (FSD flows) to be used in decoding an instruction's first specifier and in decoding specifiers encountered after execution flows are entered.

The Execution_Started flip-flop is cleared at IID.

2.5 I Box Related Microinstructions

In the Source/Destination fields:

- The ID register can be read onto the B_Bus as MR[3].
- The AT/DL register can be read to or written from the AW_Bus as PR[B] with AT in bits <4:3> and DL in bits <2:0>.
- The PSL TP and T bits can be read to the AW_Bus as PR[C] with TP in bit <30> and T in bit <4>.
- The opcode register can be read onto the AW_Bus as T[E].

In the MISC field:

Code	Operation	Comment
1	RN <- RN+1	increment RN
2	RN <- RN-1	decrement RN
3	RN <- SC	load RN with the bottom four bits of SC
B	ENABLE PREFETCH	clear the microcode Prefetch Halt bit
C	DISABLE PREFETCH	set the microcode Prefetch Halt bit
E	DL <-- BYTE	force data length to BYTE
F	DL <-- WORD	force data length to WORD
14	SPEC DECODE	parse a new specifier
15	DL <-- LONG	force data length to LONG
16	LOAD ID LONG CASE	load the ID register with four bytes from the Prefetch Stack, case if not enough data
17	LOAD ID CASE	load ID from the Prefetch Stack, data length based on the DL register, case if not enough data
1B	RESTART PREFETCH	clear both the hardware and microcode Prefetch Halt bits

In the SPECIAL MISC1 field:

Code	Operation	Comment
1	POP RLOG	pop the top of the RLOG stack into RN and STATE
4	SET FPNT	set FPU Present
5	CLR FPNT	clear FPU Present

In the SPECIAL MISC3 field:

Code	Operation	Comment
6	SET VAX TRAP REQUEST	set the trap bit for the IID branch
7	CLEAR VAX TRAP REQUEST	clear the trap bit for the IID branch

In the MXPR ADDR field:

Code	Operation	Comment
28	PSL.HWRE	load the PSL bits stored in the I Box (FPD, IV, T, TP) from the AW_Bus

In the Branch Condition Select (BCS) field:

Code	Operation	Comment
1	NO FPD INTERRUPTS PENDING	branch if no FPD interrupts pending
5	IF AVMF GOTO IF BWL NSD/RET	branch if AT is not read or if instruction is floating point, else if DL is byte, word, or long then if not Execution_Started NSD, else RET
9	NOT FPU	branch if FPU Present is clear
1A	RMODE	branch if last specifier was register mode
24	IF A GOTO IF V OR BWL NSD/RET	branch if AT is address, else if AT is variable or DL is byte, word, or long then if not Execution_Started NSD, else RET
25	IF AVM GOTO IF BWL NSD/RET	branch if AT is not read, else if DL is byte, word, or long then if not Execution_Started NSD, else RET
26	IF BWL IID ELSE GOTO	branch if DL is quad, else IID
29	IF R OR M GOTO	branch if AT is read or modify
2A	IF BCOND LOAD VIBA&PC ELSE IID	branch if BCOND is true and invalidate the Prefetch Stack, else IID
2B	SUCCESSIVE IID	unconditional IID, ignore exceptions other than IB_DRY
2C	NSD/RET	NSD if not Execution_Started, else RET
2D	IF AV NSD/RET ELSE GOTO	branch if AT is read or modify, else if not Execution_Started NSD, else RET
2E	IID	unconditional IID
2F	LOAD VIBA&PC	unconditional branch, invalidate Prefetch Stack
36	OPCODE 3-0	case on bottom four bits of opcode
37	DL.MBOX STATUS	case on DL and the M Box status bits, with DL in uTest<3:2> and the M Box status in uTest<1:0>
3D	RETURN + BO	unconditional return plus branch offset

2.6 Tables

2.6.1 Fork Type Decode -

This logic derives the Fork Type from the Fork Code and the Specifier Counter, according to the following table.

	Specifier #		
Fork Code	0	1	
0	S	E	S = Second Spec Decode
1	O	E	O = Optimize if RMODE, else SSD
2	E	X	E = Execute
3	I	X	I = Illegal Opcode Trap

2.6.2 Execution Dispatch Masks -

Opcode<8> is OR'd into Opcode<5> and the result is ANDed with one of these masks to produce eight bits of the dispatch address for the NSD Execute fork.

Code	Mask	Comment
0	1111 1001	used to force BBCx to BBC, BBSx to BBS
1	1001 1111	used to force ADD{B,W,L}x to ADDBx, SUB{B,W,L}x to SUBBx, etc.
2	1101 1001	used to force ADD{F,D,G}x, SUB{F,D,G}x, MUL{F,D,G}x, DIV{F,D,G}x to ADDFx
3	0001 0000	used to force MOVx, MOVAx, MOVZxL to 10 (hex)
4	0001 0101	used to force PUSHAx, PUSHL to 15 (hex), CLRx to 14 (hex)
5	1111 1111	used for unmodified dispatch
6	1111 1100	used to force CVT{F,D,G}{B,W,L} to CVT{F,D}B, CVT{B,W,L}{F,D,G} to CVTB{F,D}
7	1111 1110	used to consolidate even-odd pairs

2.6.3 Karnaugh Maps Of Opcodes -

KARNAUGH MAP OF MicroVAX ONE-BYTE OPCODES

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	HALT	NOP	REI	BPT	RET	RSB	LDPCTX	SVPCTX	CVTPS	CVTSP	INDEX	CRC	PROBER	PROBEW	INSQUE	REMQUE
1	BSBB	BRB	BNEQ	BEQL	BGTR	BLEQ	JSB	JMP	BGEQ	BLSS	BGTRU	BLEQU	BVC	BVS	BCC	BCS
2	ADDP4	ADDP6	SUBP4	SUBP6	CVTPT	MULP	CVTTP	DIVP	MOVC3	CMPC3	SCANC	SPANC	MOVC5	CMPC5	MOVTC	MOVTUC
3	BSBW	BRW	CVTWL	CVTWB	MOVP	CMPP3	CVTPL	CMPP4	EDITPC	MATCHC	LOCC	SKPC	MOVZWL	ACBW	MOVAW	PUSHAW
4	ADDF2	ADDF3	SUBF2	SUBF3	MULF2	MULF3	DIVF2	DIVF3	CVTFB	CVTFW	CVTFL	CVTRFL	CVTBF	CVTWF	CVTLF	ACBF
5	MOVF	CMPF	MNEGF	TSTF	EMODF	POLYF	CVTFD		ADAWI				INSQHI	INSQTI	REMQHI	REMQTI
6	ADDD2	ADDD3	SUBD2	SUBD3	MULD2	MULD3	DIVD2	DIVD3	CVTDB	CVTDW	CVTDL	CVTRDL	CVTBD	CVTWD	CVTLD	ACBD
7	MOVD	CMPD	MNEGD	TSTD	EMODD	POLYD	CVTDF		ASHL	ASHQ	EMUL	EDIV	CLRQ	MOVQ	MOVAQ	PUSHAQ
8	ADDB2	ADDB3	SUBB2	SUBB3	MULB2	MULB3	DIVB2	DIVB3	BISB2	BISB3	BICB2	BICB3	XORB2	XORB3	MNEGB	CASEB
9	MOVB	CMPB	MCOMB	BITB	CLRB	TSTB	INCB	DECB	CVTBL	CVTBW	MOVZBL	MOVZBW	ROTL	ACBB	MOVAB	PUSHAB
A	ADDW2	ADDW3	SUBW2	SUBW3	MULW2	MULW3	DIVW2	DIVW3	BISW2	BISW3	BICW2	BICW3	XORW2	XORW3	MNEGW	CASEW
B	MOVW	CMPW	MCOMW	BITW	CLRW	TSTW	INCW	DECW	BISPSW	BICPSW	POPR	PUSHR	CHMK	CHME	CHMS	CHMU
C	ADDL2	ADDL3	SUBL2	SUBL3	MULL2	MULL3	DIVL2	DIVL3	BISL2	BISL3	BICL2	BICL3	XORL2	XORL3	MNEGL	CASEL
D	MOVL	CMPL	MCOML	BITL	CLRL	TSTL	INCL	DECL	ADWC	SBWC	MTPR	MFPR	MOVPSL	PUSHL	MOVAL	PUSHAL
E	BBS	BBC	BBSS	BBCS	BBSC	BBCC	BBSSI	BBCCI	BLBS	BLBC	FFS	FFC	CMPV	CMPZV	EXTV	EXTZV
F	INSV	ACBL	AOBLSS	AOBLEQ	SOBGEQ	SOBGTR	CVTLB	CVTLW	ASHP	CVTLP	CALLG	CALLS	XFC	**		

** TWO-BYTE OPCODE PREFIX

KARNAUGH MAP OF MicroVAX TWO-BYTE OPCODES

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0																
1																
2																
3				CVTGF												
4	ADDG2	ADDG3	SUBG2	SUBG3	MULG2	MULG3	DIVG2	DIVG3	CVTGB	CVTGW	CVTGL	CVTRGL	CVTBG	CVTWG	CVTLG	ACBG
5	MOVG	CMPG	MNEGG	TSTG	EMODG	POLYG										
6																
7																
8																
9										CVTFG						
A																
B																
C																
D																
E																
F																

IID BRANCH, NO EXCEPTIONS PENDING, ONE-BYTE OPCODE: OPCODE<8> = 0, PSL<FPD>=0

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	exe	exe	exe	exe	exe	exe	exe	exe	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd
1	bsub	br	br	br	br	br	fsd	fsd	br	br	br	br	br	br	br	br
2	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd
3	bsub	br	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd
4	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd
5	fsd	fsd	fsd	fsd	fsd	fsd	fsd		fsd				fsd	fsd	fsd	fsd
6	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd
7	fsd	fsd	fsd	fsd	fsd	fsd	fsd		fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd
8	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd
9	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd
A	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd
B	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd
C	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd
D	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd
E	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd
F	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	FC	FD		

exe = 130..13E, 8-way opcode branch based on opcode<2:0>
br = 120
bsub = 124
FC = 128
FD = 12A
fsd = 140..15E, 16-way specifier branch based on IB[PC+1]
___ = don't care. These illegal opcodes will microtrap on the cycle following IID.

IID BRANCH, NO EXCEPTIONS PENDING, TWO-BYTE OPCODE: OPCODE<8> = 1, PSL<FPD>=0

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0																
1																
2																
3				fsd												
5	fsd	fsd	fsd	fsd	fsd	fsd										
4	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd	fsd
6																
7																
8																
9										fsd						
A																
B																
C																
D																
E																
F																

fsd = 140..15E, 16-way specifier branch based on IB[PC+1]
___ = don't care. These illegal opcodes will microtrap on the cycle following IID.

IID BRANCH, NO EXCEPTIONS PENDING, ONE-BYTE OPCODE: OPCODE<8> = 0, PSL<FPD>=1

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0									fpdx	fpdx		fpdx				
1																
2	fpdx	fpdx	fpdx	fpdx	fpdx	fpdx	fpdx	fpdx	fpdx	fpdx	fpdx	fpdx	fpdx	fpdx	fpdx	fpdx
3					fpdx	fpdx	fpdx	fpdx	fpdx	fpdx	fpdx	fpdx				
4																
5						fpdx										
6																
7						fpdx										
8																
9																
A																
B																
C																
D																
E																
F									fpdx	fpdx			fpdx	FD		

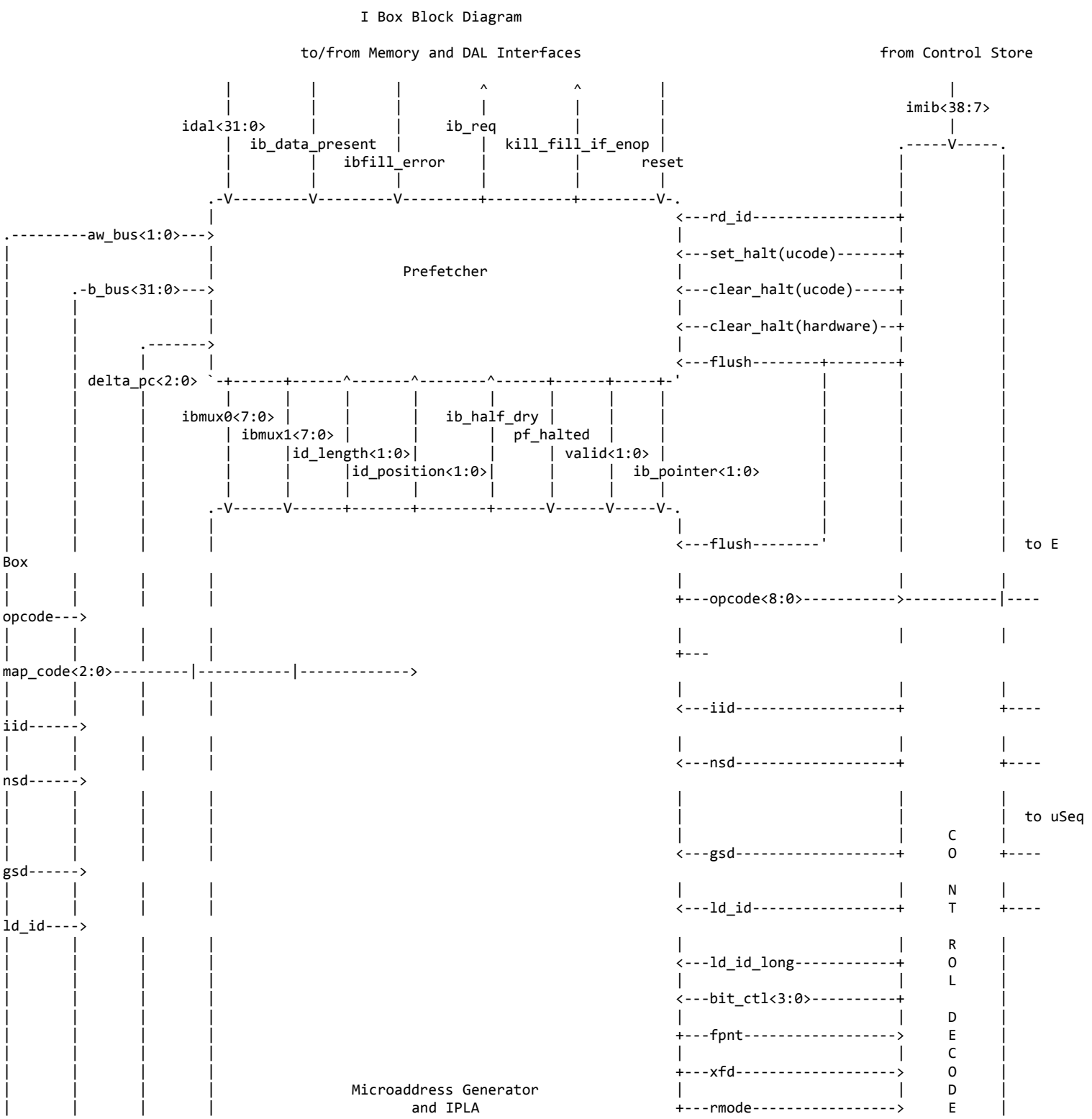
fpdx = 12C
FD = 12A
___ = don't care. These illegal opcodes will microtrap on the cycle following IID.

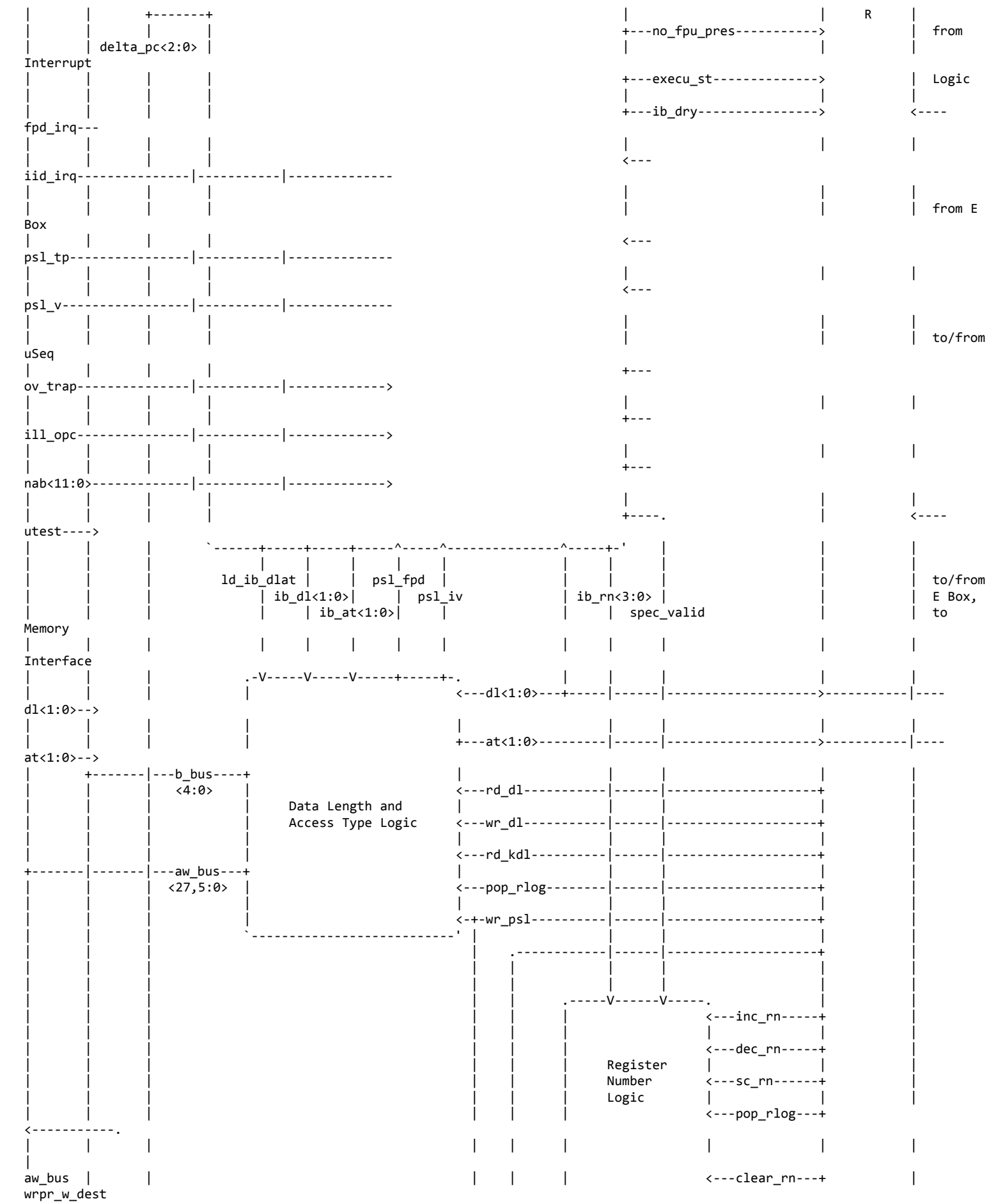
IID BRANCH, NO EXCEPTIONS PENDING, ONE-BYTE OPCODE: OPCODE<8> = 1, PSL<FPD>=1

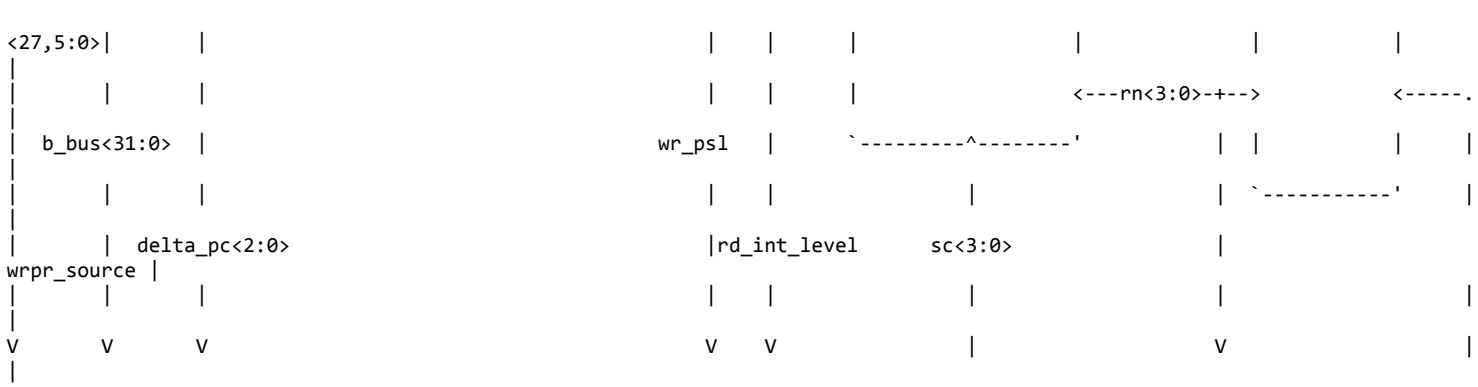
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0																
1																
2																
3																
4																
5						fpdx										
6																
7																
8																
9																
A																
B																
C																
D																
E																
F																

fpdx = 12C
___ = don't care. These illegal opcodes will microtrap on the cycle following IID.

2.7 Block Diagram




delta_pc<2:0>
fpd_irq
iid_irq
psl_tp
psl_v
ov_trap
ill_opc
nab<11:0>
utest
ld_ib_dlat
ib_dl<1:0>
ib_at<1:0>
psl_fpd
psl_iv
ib_rn<3:0>
spec_valid
dl<1:0>
at<1:0>
b_bus<4:0>
aw_bus<27,5:0>
Data Length and Access Type Logic
rd_dl
wr_dl
rd_kdl
pop_rlog
wr_psl
Register Number Logic
inc_rn
dec_rn
sc_rn
pop_rlog
clear_rn
no_fpu_pres
execu_st
ib_dry
R
from Interrupt Logic
from E Box
to/from uSeq
to/from E Box, to Memory Interface
aw_bus wrpr_w_dest


<27,5:0>
b_bus<31:0>
delta_pc<2:0>
wrpr_source
wr_psl
rd_int_level
sc<3:0>
rn<3:0>
to/from E Box
to Interrupt Logic
to/from E Box

3.0 EXECUTION (E) BOX

The E Box contains the main execution data path. It consists of the following major functions:

- General purpose registers
- Temporary storage for 19 longwords of data or addresses
- Program Counter (PC)
- Constant generator
- Shift Counter (SC)
- Arithmetic logical unit (ALU)
- Shifter
- ALU, shifter, and PSL condition code logic
- State logic
- Register logging (RLOG) stack

The E Box is connected to the IDAL. It does address and data transfers to and from the other subsystems over this bus. The Data Path receives instruction data from the I Box. It is controlled by the microinstruction and a limited amount of internal state.

3.1 Register File

The Register File consists of three sub-files: the general purpose registers, the temporary registers, and the working registers.

3.1.1 General Purpose Registers (GPR[0:E]) -

Fifteen of the locations in the register file are the General Purpose Registers, GPR[0] - GPR[E]. GPR[0] - GPR[D] are the user visible registers R0 - R13. GPR[E] is the user visible SP. GPR[F] is the PC and is located in the PC block. The address to the GPR's is either the A field of the microinstruction, the RN register, or the output of logic that adds one to the RN register. This is determined by the micro opcode and the associated Source/Destination field.

This file is read onto the A_Bus and can be written from either the AW_Bus or the D_Bus. The D_Bus is the data path from the DAL. The AW_Bus is the path from the ALU and Shifter outputs. Writes are data length dependent: if FORCE_LONG%I_M%P24_H is asserted, the length is long; otherwise, the length is specified by the DL register.

3.1.2 Temporary Registers (T[0:B]) -

Twelve of the locations in the register file are the temporary registers, T[0] - T[B]. These may be used as general purpose temporary storage for intermediate results of microcode routines, or to hold architecturally specified registers or data. They are addressed by the A field of the microinstruction. This file is read onto the A_Bus and can be written from either the AW_Bus or the D_Bus. Transfers to and from the temporary registers are longwords only.

T[0] through T[B] are general purpose temporary registers and reside in the E Box register file. T[C] is the VA' register and T[D] is the VIBA register, both of which reside in the M Box. T[E] is the opcode register, which resides in the I Box. T[F] is the Memory Management Status register, which resides in the M Box. T[C:F] may only be used with a limited subset of the microinstructions. T[C:E] are read only; T[F] is write only.

3.1.3 Working Registers (WR[0:6]) -

Seven of the locations in the register file are the dual-ported working registers, WR[0] - WR[6]. These are used as general purpose storage of intermediate microcode routine results.

WR[0] through WR[5] are general-purpose temporaries. WR[6] can be used as a destination register for three address arithmetic. It is loadable as WR[6] or can be written with the MISC field. It is always read as WR[6].

WR[7] is the SC register. While SC is usable as a normal working register, it has other functions. For a description of these functions, see the SC Logic section.

Working registers are always addressed by the A and B fields; the register addressed by the A field appears on the A_Bus when selected, and the one addressed by the B field appears on the B_Bus.

The AW_Bus data can be loaded into any of the working registers under control of the microword. WR[6,7] can be loaded in parallel with the other working registers or other temporaries under control of the MISC field.

3.1.4 Functional Summary -

The following tables summarize the read/write operation of the register file as a function of the microinstruction.

Note: Register file writes may be from one of two sources: the AW_Bus or the D_Bus, dependent on the microinstruction type. The source is the D_Bus for MXPR, FBOX XFER, and MEM REQ, and the AW_Bus otherwise.

Note: During the MEM REQ READ microinstruction (data transfer from memory to E Box), the DAL Interface always zero extends the incoming data. On

transfers of this type, the destination register will be written with this zero extended data.

Note: PC and SC are also written from the register file bit lines, so these rules apply to the writing of these registers as well.

3.1.4.1 BASIC Source And Destination Control Field, CS<32:28> -

	A_Bus	B_Bus	Dest
0	WR/PR[A]	WR[B]	WR/PR[A]
1	WR/PR[A]	WR[B]	None
2	WR/PR[A]	WR[B]	WR[B]
3	GPR[A]	WR[B]	None
4	WR/PR[A]	MR[B]	WR/PR[A]
5	WR/PR[A]	MR[B]	None
6	T[A]	WR[B]	None
7			
8	GPR[A]	MR[B]	None
9	GPR[A]	MR[B]	GPR[A]
A	GPR[A]	WR[B]	WR[B]
B	GPR[A]	WR[B]	GPR[A]
C	T[A]	MR[B]	None
D	T[A]	MR[B]	T[A]
E	T[A]	WR[B]	WR[B]
F	T[A]	WR[B]	T[A]
10	GPR[RN]	WR[B]	WR/PR[A]
11	GPR[RN]	WR[B]	GPR[RN] & WR/PR[A]
12	GPR[RN+1]	WR[B]	WR/PR[A]
13	WR/PR[A]	WR[B]	GPR[RN+1]
14			
15	GPR[RN]	MR[B]	GPR[RN] & WR/PR[A]
16			
17			
18	GPR[RN]	MR[B]	WR/PR[A]
19	WR/PR[A]	MR[B]	GPR[RN]
1A			
1B			
1C			
1D			
1E	T[RN]	WR[B]	WR/PR[A]
1F			

3.1.4.2 CONSTANT Source And Destination Control Field, CS<31:29> -

	A_Bus	Dest
0	WR/PR[A]	WR/PR[A]
1	T[A]	T[A]
2	GPR[A]	GPR[A]
3	GPR[A]	VA
4	WR/PR[A]	None
5	WR/PR[A]	WR6
6	WR/PR[A]	SC
7	WR/PR[A]	VA

3.1.4.3 SHIFT Source And Destination Control Field, CS<30:27> -

	A_Bus	B_Bus	Dest
0	WR/PR[A]	#0	WR[B]
1	WR/PR[A]	#0	WR[B]
2	WR/PR[A]	WR[B]	None
3	WR/PR[A]	WR[B]	WR[B]
4	WR/PR[A]	#0	WR/PR[A]
5			
6	#0	WR[B]	WR/PR[A]
7	#0	WR[B]	WR/PR[A]
8	WR/PR[A]	#0	None
9			
A	WR/PR[A]	WR[B]	WR/PR[A]
B	WR/PR[A]	WR[B]	None
C	GPR[RN]	#0	WR/PR[A]
D			
E			
F			

3.1.4.4 MXPR Source And Destination Control Field, CS<25:23> -

	Source	Dest
0	WR/PR[A]	WR[A]
1	T[A]	T[A]
2	GPR[A]	GPR[A]
3	GPR[RN]	GPR[RN]
4		
5		
6		
7		

Note: The direction of data transfer is determined by the MXPR RD bit, CS<28>. This bit is SET for transfers into the E Box (E Box register writes), and CLEAR for transfers out of the E Box (E Box register reads).

3.1.4.5 MEM REQ Source And Destination Control Field, CS<25:23> -

	Source	Dest
0	WR/PR[A]	WR[A]
1	T[A]	T[A]
2	GPR[A]	GPR[A]
3	GPR[RN]	GPR[RN]
4		
5		
6		
7		

Note: The memory function determines the direction of the data transfer. For memory reads, transfer into the E Box (E Box register writes) is indicated. For memory writes, transfer out of the E Box (E Box register reads) is indicated.

3.1.4.6 FBOX XFER Source And Destination Control Field, CS<24:23> -

	Source	Dest
0	WR/PR[A]	WR[A]
1	T[A]	T[A]
2	GPR[A]	GPR[A]
3	GPR[RN]	GPR[RN]

Note: The direction of data transfer is determined by the FBOX XFER RD bit, CS<25>. This bit is SET for transfers into the E Box (E Box register writes), and CLEAR for transfers out of the E Box (E Box register reads).

3.1.4.7 SPECIAL MISC2 Control Field, CS<27:23> -

MISC2	Operation
8	WRITE T[A]

3.1.4.8 SPARE Function Control Field, CS<32:28> -

Note that the SPARE functions specify the Source/Dest information; SPARE microinstructions do not have an explicit S/D field.

	A_Bus	B_Bus	Dest
0	WR/PR[A]	WR[B]	WR[B]
1	WR/PR[A]	WR[B]	WR[B]
2	WR/PR[A]	WR[B]	WR[B]

3.1.4.9 MISC Control Field, CS<22:18> -

In addition to the specific microinstruction control listed above, Register File operations are initiated by the MISC field as well.

MISC	Operation
A	WRITE WR[6]

This operation is defined as a function of the microinstruction type as follows:

- MXPR, FBOX XFER, MEM REQ: Register is loaded from the D_Bus.

  Note that for outgoing transfers (writes), WR[6] will NOT catch the outgoing data; data written must be considered unpredictable.

- BASIC, SHIFT, SPARE: Register is loaded from the AW_Bus.

This function has the following restrictions:

- When a GPR is written in parallel, the data length must be LONG.

3.1.5 Microcode Restrictions -

1. When a GPR and a WR are written in parallel, and the effective data length is not LONG, then the contents of the WR must be considered unpredictable. This condition may occur through either an explicit BASIC dual register write S/D field, or with a normal GPR write destination field with a MISC/WRITE WR[6] or MISC/WRITE SC field.

2. PR[9:F] can not be specified as source or destination in MXPR, FBOX XFER, or MEM REQ microinstructions.

3. T[B] can not be used with the SPECIAL MISC2 function WRITE T[A].

4. T[C:D] can only be used as a source, and only in BASIC microinstructions with source/dest codes = C or E.

5. T[E] can only be used as a source, and only in BASIC microinstructions with source/dest codes = C or E or in FBOX XFER and MEM REQ microinstructions with source/dest code = 1.

6. T[F] can only be used as a destination, and only in BASIC microinstructions with source/dest codes = D or F.

7. T[C:E] are read only; T[F] is write only.

3.2 PC Logic

The PC logic function consists of the PC register, the PC adder, and the BPC register.

3.2.1 PC Register -

This longword register is used to hold the (virtual) Program Counter. It is read onto the A_Bus when addressed as GPR[F] or as GPR[RN] and RN = F (it CANNOT be addressed as GPR[RN+1]). It is loaded from the AW_Bus or D_Bus via two BCS fields:

BCS	Operation
2A	IF BCOND LOAD VIBA&PC ELSE IID
2F	LOAD VIBA & PC & BRANCH ALWAYS

(The first function is conditional on the selected branch condition, the second is unconditional.) The value loaded depends on the microinstruction being executed. For all but MXPR, FBOX XFER, and MEM REQ, the PC is loaded from the AW_Bus. For these types of microinstructions, it is loaded from the D_Bus (incoming DAL data), regardless of the direction of transfer. Note that PC will NOT catch outgoing data for these microinstruction types.

Note: For this type of load, while PC may be loaded with incoming DAL data, VIBA will not; VIBA will be loaded from the AW_Bus regardless.

The PC is loaded from the BPC register when the SPECIAL microinstruction is executed and MISC2 = 2.

Whenever it is not addressed as a destination, the PC is loaded from the PC adder, which adds delta PC, a value furnished by the I Box.

3.2.2 PC Adder -

The PC adder is used for incrementing the PC as macroinstructions are parsed. Its inputs are the 32-bit PC register, and the 3-bit value, delta PC, which can have the values 0, 1, 2, 3, 4, 5, or 6. Delta PC is provided by the I Box as it processes the instruction stream. Whenever there is nothing to add to the PC, delta PC is made equal to zero by the I Box.

3.2.3 BPC Register -

This longword register is used to snapshot the PC during IID in case the instruction faults or an interrupt is to be taken. It is loaded during normal IID. BPC will not be loaded on successive IID (BCS = 2B).

3.2.4 Microcode Restrictions -

1. The PC can not be loaded as GPR[F] or GPR[RN].

2. The PC can not be addressed as GPR[RN+1].

3. All PC loads write all 32 bits of the PC.

3.3 K Mux

The K Mux gates data onto the A_ and B_Busses.

3.3.1 K(DL) -

A decoded version of DL called K(DL) is available in miscellaneous register MR[6]:

K(DL) = 1, BYTE
= 2, WORD
= 4, LONG and FLOAT
= 8, QUAD and DOUBLE and GRAND

3.3.2 Constant Generator -

The constant generator creates a 32-bit constant from twelve bits in the microinstruction. The constant is put on the B_Bus during CONSTANT microinstructions, as specified by the KFMT and KPOS fields:

KFMT	KPOS	B<31:24>	B<23:16>	B<15:8>	B<7:0>
0	x	constant	constant	constant	constant
1	0	00000000	00000000	00000000	constant
1	1	00000000	00000000	constant	00000000
1	2	00000000	constant	00000000	00000000
1	3	constant	00000000	00000000	00000000
3	0	11111111	11111111	11111111	constant
3	1	11111111	11111111	constant	00000000
3	2	11111111	constant	00000000	00000000
3	3	constant	00000000	00000000	00000000

Note that KFMT = 2 is unused.

3.3.3 Zero Extension -

Under control of the FUNCTION field of the BASIC microinstruction and the data length, zeroes are inserted instead of the upper bits of either bus. Zero extension of a BYTE causes bits <31:8> to be cleared; for a WORD, <31:16> are cleared. ZEXT has no effect with LONGWORD data.

Zero extension is based on the DL Register.

BASIC Microinstruction, FUNCTION field:

code	function
0A	zero extend A_Bus
1D	zero extend B_Bus
1F	zero extend B_Bus
all others	no zero extension

3.3.4 Addressing/Other Constants -

Some of the K Mux functions are addressed using MR decodes:

MR	Data to B_Bus
0	#0
1	#4
2	#2

Other K Mux functions are addressed using PR decodes:

PR	Data to A_Bus
8	#1
E	SEXT(ALU.N)

Certain SHIFT functions require a zero on the A_ or B_Bus. This is generated by the K Mux.

S/D	A_Bus	B_Bus
0	WR/PR[A]	#0
1	WR/PR[A]	#0
4	WR/PR[A]	#0
6	#0	WR[B]
7	#0	WR[B]
8	WR/PR[A]	#0
C	GPR[RN]	#0

3.3.5 Microcode Restrictions -

No known restrictions.

3.4 SC (Shift Counter)

The SC is a 32-bit register located in the data path. It may be used as a general purpose 32-bit register when addressed as a working register. In addition, it is used in conjunction with the ALU to perform double precision shift operations.

The SC register/counter is general purpose in nature and is used for the following:

- Shift value
- Source/destination of the RN register
- General purpose counter
- Q register for multiply and divide operations

3.4.1 Functional Summary -

The SC may be addressed as WR[7] and as such may be read/written as a working register would be. When addressed as a WR[7], SC is read onto the AW_Bus. When written during BASIC, CONSTANT, SHIFT, and SPARE microinstructions, it is written from the AW_Bus. When written during MXPR, FBOX XFER, and MEM REQ microinstructions, it is written from the D_Bus, with incoming DAL data. (See Register File Section for addressing details.)

In addition to addressing as WR[7], SC is loadable under control of the MISC field, as follows:

- MISC 4#16: SC <-- RN (SC<3:0> <-- RN, SC<31:4> <--#0)
- MISC 6#16: SC<15:0> <-- SC<15:0> + 1 \Use only to set SC<0>!\
- MISC 7#16: WRITE SC

WRITE SC is defined as a function of the microinstruction type as follows:

- MXPR, FBOX XFER, MEM REQ: Register is loaded from the D_Bus.

  Note that for outgoing transfers (writes), SC will NOT catch the outgoing data; data written must be considered unpredictable.

- BASIC, SHIFT, SPARE: Register is loaded from the AW_Bus.

This function has the following restrictions:

- When a GPR is written in parallel, the data length must be LONG.
- If SC was loaded from the D_Bus, it must not be tested on the next microinstruction.

Used in conjunction with the ALU, SC may be used to accomplish double precision single bit left/right shifts. These functions are under control of the SPARE microinstruction. The summary is as follows:

SPARE FUNCTION	SC SHIFT FUNCTION
0	SHIFT SC RIGHT
1	SHIFT SC RIGHT
2	SHIFT SC LEFT
all others	none

All of these shifts are done in conjunction with the ALU data path. On all RIGHT shifts, the SC shift INPUT (the MSB) is the shift OUTPUT of the ALU data path. In addition, on all LEFT shifts, the SC shifter passes a shift OUTPUT to the ALU data path to be used as its shift input. On all LEFT shifts, a zero is the shift input. (See ALU section for details of entire operation.)

The SC register is testable via the Microtest Bus, as specified by the BCS field, as follows:

BCS	Branch Command
31#16	SC<3:0> case
33#16	SC<7:4> case

3.4.2 Microcode Restrictions -

1. SC can not be tested (via Branch Condition Select) immediately following a load from the D_Bus. This includes writes via microinstructions MXPR, FBOX XFER, or MEM REQ.

2. When executing a MISC/WRITE SC, the derived data length must be LONG when a GPR is being written in parallel.

3. The following SC operations must be mutually exclusive:

   - Load as WR[7]
   - Load via MISC/WRITE SC
   - Increment by one, MISC operation
   - Load of SC from RN, MISC operation
   - Any SPARE function

3.5 ALU

The ALU is a general purpose Arithmetic Logic Unit, used for microcode specified operations. The ALU operates upon the data presented over the A_Bus and B_Bus and puts its results on the AW_Bus, as specified by the micro opcode and various ALU control fields.

3.5.1 Functional Summary -

The ALU data path consists of a full 32-bit block capable of addition of any combination of A, .NOT.A, B, and .NOT.B with the carry input programmable as well as any logical function of A and B.

The ALU always generates 32 bits of valid data; the results are not data length dependent.

The ALU also generates condition codes based on the results of its operation. These immediate ALU CC's are data length dependent. If FORCE_LONG%I_M%P24_H is asserted, the length in longword; otherwise, the length is specified by the DL register.

The ALU is controlled by the microinstruction. In the BASIC and CONSTANT microinstructions, specific fields specify the function the ALU performs. "A" refers to the A_Bus, and "B" refers to the B_Bus. The result is driven onto the AW_Bus.

3.5.1.1 Operation Control -

For the following functional description, the notation is as follows:

- ALU_SUM<n> is the nth order bit of the generated ALU result.
- ALU_CY<n> is the nth order carry bit from the ALU data path. Note that it is the carry bit INTO the nth bit slice. (This is consistent with the full adder description S(i) = A(i).XOR.B(i).XOR.C(i)). Under this convention, the low order carry input to the ALU would actually be ALU_CY<0> and the high order carry out of the ALU would be ALU_CY<32>.

3.5.1.1.1 BASIC ALU Control -

In the BASIC microinstruction, the ALU operation is defined by the bits <37:33> of the microinstruction.

(Quantities in braces indicate use of these codes by other logic.)

Code	Function	Code	Function
0	A.MINUS.B	10	A.MINUS.1
1	B.MINUS.A	11	A.PLUS.B.PLUS.(PSL.C)
2	A.MINUS.B.PLUS.(.NOT.PSL.C)	12	A.PLUS.B.PLUS.1
3	A.MINUS.B {RLOG}	13	A.PLUS.B {RLOG}
4	.NEG.B	14	A.PLUS.B
5	A.PLUS.1	15	A.PLUS.B.PLUS.(ALU.C)
6	B.MINUS.1	16	A.XOR.B
7	A.AND.NOT.B	17	.NOT.A
8	A.AND.B	18	B.PLUS.1
9	PASS A	19	A.OR.B
A	PASS A {ZEXT}	1A	PASS B
B		1B	
C		1C	
D		1D	PASS B {ZEXT}
E		1E	
F	.NOT.B	1F	PASS B {SLIT}

NOTE: Zero extension of passed data is based on the DL Register contents.

Note: The Pass B short literal function requires that the following NOT be selected as the A_Bus source:

- o T[C] (VA')
- o T[D] (VIBA)
- o PR[A] (VA)
- o PR[D] (ALU CC's and STATE flags)

3.5.1.1.2 CONSTANT ALU Control -

In the CONSTANT microinstruction, the ALU function is controlled by bits <36:34> of the microinstruction.

Code	Function
0	A.PLUS.B
1	A.MINUS.B
2	B.MINUS.A
3	A.AND.B
4	A.OR.B
5	PASS B
6	A.AND.NOT.B
7	A.XOR.B

3.5.1.1.3 SPARE ALU Control -

The SPARE microinstruction function field controls a set of ALU functions used for multiply and divide operations. These functions are defined as follows:

1. SPARE FUNCTION = 0, CON ADD/PASS, SHIFT INTO SC, MSB = C

   If SC<0> = 1, then A PLUS B, ELSE PASS B. Shift the result one bit right into SC, with a shift in of ALU.C.

2. SPARE FUNCTION = 1, CON ADD/PASS, SHIFT INTO SC, MSB = N XOR V

   If SC<0> = 1, then A PLUS B, ELSE PASS B. Shift the result one bit right into SC, with a shift in of ALU.N XOR ALU.V.

3. SPARE FUNCTION = 2, SHIFT WR[A]&SC LEFT, LSB = 0

   PASS A and shift the result left one bit into the bit bucket. The shift in is the shift out of SC.

3.5.1.1.4 SHIFT ALU Control -

The ALU does not drive the AW_bus during a shift.

3.5.1.1.5 MEM REQ, SPECIAL, FBOX XFER, MXPR ALU Control -

The ALU function is PASS A.

3.5.1.2 Immediate ALU Condition Codes -

In addition to the results generated by the operation specified by the microinstruction, the ALU also generates a set of condition codes based on the results of that operation, Imm_ALU.N,Z,V,C. (Note that the "Imm_" prefix indicates that they are the immediate codes generated by the ALU during the current cycle. This is to avoid confusion with the ALU.N,Z,V,C codes that are stored in the ALU CC register in the CC logic.) These condition codes are defined as follows:

1. Imm_ALU.N - The sign bit of the result, that is, the most significant bit of the result.

2. Imm_ALU.Z - The zero condition bit of the result. This is true if the result is exactly zero.

3. Imm_ALU.C - The carry-out of the operation. This only has meaning during ADD and SUBTRACT operations. It is exactly zero for PASS and all logical functions.

4. Imm_ALU.V - The integer overflow bit. This bit indicates an integer overflow of the operation, including two's complement operations. It is defined as the exclusive-OR of the carry-in to the last stage and the carry-out of the last stage.

These conditions are dependent on the prevailing data length, as follows:

- DL register = BYTE and FORCE_LONG%I_M%P24_H not asserted:
  Imm_ALU.N = ALU_SUM<7>
  Imm_ALU.Z = ALU_SUM<7:0> EQL 0
  Imm_ALU.V = (ALU_CY<7>).XOR.(ALU_CY<8>)
  Imm_ALU.C = ALU_CY<8>

- DL register = WORD and FORCE_LONG%I_M%P24_H not asserted:
  Imm_ALU.N = ALU_SUM<15>
  Imm_ALU.Z = ALU_SUM<15:0> EQL 0
  Imm_ALU.V = (ALU_CY<15>).XOR.(ALU_CY<16>)
  Imm_ALU.C = ALU_CY<16>

- DL register = LONG or QUAD or FORCE_LONG%I_M%P24_H asserted:
  Imm_ALU.N = ALU_SUM<31>
  Imm_ALU.Z = ALU_SUM<31:0> EQL 0
  Imm_ALU.V = (ALU_CY<31>).XOR.(ALU_CY<32>)
  Imm_ALU.C = ALU_CY<32>

NOTE: For all logical and PASS operations, the carry chain K terms will always be zero. Thus, for these operations, all carries will be zero, and Imm_ALU_C and Imm_ALU_V will be zero.

The condition codes are stored by the CC Logic.

3.5.2 Microcode Restrictions -

1. During BASIC PASS B short literal, the following must NOT be selected as the A_Bus source:

   - T[C] (VA')
   - T[D] (VIBA)
   - PR[A] (VA)
   - PR[D] (ALU CC's and STATE flags)

3.6 Shifter

The Shifter is a full 64-bit in, 32-bit out combinatorial network. It is used for shift operations as specified in the macro instruction.

Its inputs are the A_Bus and B_Bus, with the A_Bus being more significant. Its output is the AW_Bus. It can do 0 to 31 bit right shifts, where a zero-bit shift selects the B_Bus. It can also do a 0 to 31 bit left shift (32-SV), where a shift value (SV) of 0 selects the A_Bus. Note that the full A_Bus may not be selected during a right shift, and the full B_Bus cannot be selected during a left shift.

The shift operations to be done are specified by the source/destination field of the SHIFT microinstruction. Depending upon that field, the shift amount is given by either SV or 32-SV. SV is the SHF_VAL field of the microword if nonzero, otherwise either the low five bits of the SC or the zero-extended DL register. When using SC, both zero and 32-bit shifts work; for an SC right shift of zero, the result is the B_Bus; for an SC right shift of 32, the A_Bus.

3.6.1 Functional Summary -

	A_Bus	B_Bus	Dest	Shift	Reg.
0	WR/PR[A]	#0	WR[B]	32-SV	SC
1	WR/PR[A]	#0	WR[B]	SV	SC
2	WR/PR[A]	WR[B]	None	32-SV	SC
3	WR/PR[A]	WR[B]	WR[B]	SV	SC
4	WR/PR[A]	#0	WR/PR[A]	32-SV	SC
5					
6	#0	WR[B]	WR/PR[A]	32-SV	SC
7	#0	WR[B]	WR/PR[A]	SV	SC
8	WR/PR[A]	#0	None	32-SV	SC
9					
A	WR/PR[A]	WR[B]	WR/PR[A]	32-SV	SC
B	WR/PR[A]	WR[B]	None	SV	SC
C	GPR[RN]	#0	WR/PR[A]	32-SV	DL
D					
E					
F					

Note: When the Shift Value is shown as SV, the shift is right, and when it is (32-SV), the shift is left.

The shifter generates Imm_SHFT.NZ, condition codes to be used in the CC Logic block. These condition codes are based on the full 32-bit result (DL is always LONG for SHIFT microinstructions). Imm_SHFT.N is the MSB of the shift result; Imm_SHFT.Z is asserted if the shift result is zero.

3.6.2 Microcode Restrictions -

1. DL can not be used as a shift value immediately after being loaded as a PR.

3.7 Condition Code Logic

On each data path operation, new condition codes, Imm_ALU.NZVC or Imm_SHFT.NZ, are generated. These condition codes can be loaded into the ALU CC Register or presented to the CC.MAP to be mapped for input to the PSL CC register. Loading is done per the various condition code control fields.

The condition codes stored in these registers are used to control certain conditional branches under control of the Branch Condition Select (BCS) field of the microinstruction.

The condition inputs are Imm_ALU.NZVC and Imm_SHFT.NZ. These are the condition codes generated every cycle.

Note that the ALU handles data length dependency, such that the CC logic need only load, as defined by the CC and MISC fields of the microinstruction.

The Condition Code function is partitioned into the following blocks.

3.7.1 ALU CC Register -

This four bit register holds the ALU CC's for use on subsequent cycles by microcode. It is loaded from different sources depending upon the following operations:

Microinstruction			Function
BASIC,	<27:26> =	00	NOP
SPARE:		01	Load from ALU
		10	Load from ALU
		11	Load from ALU **
			(Load PSL CC's from ALU)
FBOX XFER:	<27:26> =	00	NOP
		01	Load from FPU
		10	Load from FPU
		11	(Load PSL CC's from FPU)
CONSTANT,	<26> =	0	NOP
MXPR:		1	Load from ALU
SHIFT:	<26> =	0	NOP
		1	Load from ALU

** Only when PSL CC's are being loaded from Imm_ALU.NZVC, as determined by the MAPCODE signal (see section on PSL CC register).

In addition to being directly loaded from the ALU, Shifter, and FPU, the ALU CC register may be read and written using PR[D]. ALU.NZVC appears as bits <3:0> of this PR. (See Register File for detailed PR addressing information.)

The operation of the Z bit is changed when MISC = USE OLD Z. Here, the new Z bit is SET only if the new Z value supplied as input and the old Z bit are both SET. This applies to both ALU and PSL CC's. Note that this MISC field cannot be used with FPU XFER microinstructions.

For SHIFT microinstructions, the ALU CC's include only N and Z bits; V and C are cleared.

3.7.2 PSL CC Logic -

This logic includes the four bit PSL CC register and is used to hold the VAX condition codes. It is used to control conditional branches under control of the microinstruction Branch Condition Select (BCS) field by gating conditions onto the Microtest Bus. It may also be read and written as PR[C].

The PSL CC register may be loaded in several ways:

1. Through the PSL CC map. The PSL CC register is loaded through the CC MAP when the various CC fields specify that the PSL CC's be loaded. As noted above, they are loaded during the BASIC, FBOX XFER, and SPARE microinstructions when the CC field = 11. Note also that is is loaded during the MEM REQ microinstruction when the CC field = 1 (CC field is one bit in that case).

2. From the AW_Bus when addressed as PR[C].

3. From the AW_Bus when an MXPR write to the PSL is executed.

3.7.3 CC Map -

The CC map converts the immediate ALU condition code data into the actual PSL CC's that should be generated for the macroinstruction being executed. It can be used during ALU operations or when results are being stored in memory. Its data source is either the current Imm_ALU.NZVC, the ALU CC Register, or the received FPU condition codes (N and Z only, C = V = 0). It also examines the current PSL CC register.

The PSL map input conditions are called MAP.NZVC. In general, the opcode and microinstruction type determine the MAP.NZVC data source.

The IPLA controls the PSL CC map with a three bit code that "configures" the map to the proper functions. This three bit code is called MAP_CODE<2:0>.

MAP_CODE<2:0> affects PSL CC map loading as follows:

map_code						
000	PSL.NZVC	<--	IMM_ALU.NZVC			
001	PSL.NZV	<--	IMM_ALU.NZV,	PSL.C	<--	~IMM_ALU.C
010	PSL.NZV	<--	IMM_ALU.NZV,	PSL.C	<--	PSL.C
011	PSL.N	<--	IMM_ALU.N xor IMM_ALU.V,	PSL.Z	<--	IMM_ALU.Z
	PSL.V	<--	0,	PSL.C	<--	~IMM_ALU.C
100	PSL.NZVC	<--	ALU.NZVC			
101	PSL.NZV	<--	ALU.NZV,	PSL.C	<--	~ALU.C
110	PSL.NZV	<--	ALU.NZV,	PSL.C	<--	PSL.C
111	not used					

Note: The following PSL loading configuration is NOT possible:

PSL.N	<--	ALU.N xor ALU.V,	PSL.Z	<--	ALU.Z
PSL.V	<--	0,	PSL.C	<--	~ALU.C

3.7.4 Branch Test Logic -

This logic network examines the current PSL and ALU CC's, and the macro opcode. It generates one bit, BRANCH CONDITION MET, which the microcode tests via the Microtest Bus. This bit indicates if a branch should not be taken. This condition is tested when the BCS field is 2A#16. The following table summarizes the opcode dependency of this logic.

Branch Condition	Macro Opcode	
PSL.N	BLSS	19
PSL.Z	BEQL	13
PSL.V	BVS	1D
PSL.C	BCS	1F
.NOT.(PSL.N)	BGEQ	18
.NOT.(PSL.Z)	BNEQ	12
.NOT.(PSL.V)	BVC	1C
.NOT.(PSL.C)	BCC	1E
(PSL.C).OR.(PSL.Z)	BLEQU	1B
(PSL.N).OR.(PSL.Z)	BLEQ	15
.NOT.((PSL.C).OR.(PSL.Z))	BGTRU	1A
.NOT.((PSL.N).OR.(PSL.Z))	BGTR	14
always true	BRB	11
	BRW	31

In addition to the opcode dependent branches listed above, the CC Logic generates branch decision information for several BCS specified branches, as follows:

BCS	Branch Command
7	.not.(ALU.Z)
A	ALU.N
B	ALU.Z
E	ALU.V
F	ALU.C
30	ALU.NZVC case

3.7.5 Microcode Restrictions -

1. The ALU CC register must not be loaded as PR[C] AND via CC field in same microinstruction.

2. The PSL CC register must not be loaded as PR[D] AND via CC field in same microinstruction.

3. The PSL CC register must not be loaded from ALU CC register at the same time the ALU CC register is loaded.

4. An MXPR to PSL is not allowed during IID (TP load conflict).

5. Neither the ALU CC's nor the PSL CC's can be tested via a Branch Condition Select immediately following a load as a PR. This implies, in particular, that the PSL CC's can not be loaded as a PR during IID.

6. MISC/USE OLD Z can not be used with FPU XFER microinstructions.

3.8 RLOG

RLOG is a six entry LIFO stack onto which a GPR's address and the amount it is being changed are pushed. It is used to restore GPR contents under certain exception conditions. It is used primarily with auto-increment/decrement mode instructions:

```
RLOG<6:4> <- encoded incr/decr amount (contents of the DL reg if
             K(DL) is selected to B_Bus, otherwise #2)
RLOG<7>   <- .NOT.(CS<37>)  (1 for add, 0 for subtract)
RLOG<3:0> <- RN<3:0>
```

The output of this stack is automatically loaded into RN and the STATE registers when it is popped (see STATE logic).

The stack is pushed when executing a BASIC microinstruction and the FUNCTION field = 3 or 13. It is popped when executing a SPECIAL microinstruction and the MISC1 field = 2.

The stack is cleared by IID.

3.8.1 Microcode Restrictions -

1. Push/pop of RLOG is not allowed during IID.

2. Pop of RLOG is not allowed during NSD.

3.9 STATE

STATE<7:0> is a general purpose state register whose bits are set/cleared in a variety of ways. It can be tested via the Microtest Bus for branch control.

STATE<3:0> may be read and written over the AW_Bus as PR[D]. STATE<3:0> appear as bits <7:4> (see Register File for PR addressing details).

Five of the STATE bits are also loaded when the RLOG stack is popped. During this operation, STATE<4> is SET if the stack is empty, and CLEARED otherwise. STATE<3> is SET if the GPR was incremented, and CLEARED if it was decremented (STATE<3> <- RLOG[0]<7>). STATE<2:0> are SET to the encoded length value (STATE<2:0> <- RLOG[0]<6:4>).

All STATE bits may be selectively set/cleared using MISC and SPECIAL MISC2 fields, as follows:

MISC	Operation
11#16	CLEAR STATE<3:0>
12#16	SET STATE<0>
13#16	SET STATE<1>

SPECIAL MISC2	Operation
3#16	SET STATE<2>
4#16	SET STATE<3>
10#16	SET STATE<4>
11#16	SET STATE<5>
12#16	SET STATE<6>
13#16	SET STATE<7>
14#16	CLEAR STATE<4>
15#16	CLEAR STATE<5>
16#16	CLEAR STATE<6>
17#16	CLEAR STATE<7>

During Initial Instruction Decode (IID), STATE<3:0> are CLEARED.

In addition to being directly readable on the AW_Bus, STATE bits are testable on the Microtest Bus, via the BCS field, as follows:

BCS	Branch Condition
34#16	STATE <3:0> case
38#16	STATE <7:4> case

In addition to the STATE<7:0> register, the STATE logic contains the VAX Restart Bit. This bit is cleared by IID. It is set by:

1. Any write to memory that was not aborted by memory management.

2. Any write to a GPR that did not invoke RLOG.

Note: The other VAX-visible resources are some of the MXPRs and the PSL. Any changes to these will be tracked explicitly by the microcode; copying

a GPR to itself will set the VAX Restart Bit.

This bit is read using PR[D] and appears as bit <30>. It is NOT writable using this PR.

3.9.1 Microcode Restrictions -

1. The following operations must be mutually exclusive:

   - Load of STATE<3:0> as PR[D]
   - Set/clear of any of STATE<3:0> via MISC field

2. STATE<3:0> can not be tested immediately following loading as PR[D].

3.10 Summary Of E Box Microcode Restrictions

3.10.1 Register File -

1. When a GPR and a WR are written in parallel, and the effective data length is not LONG, then the contents of the WR must be considered unpredictable. This condition may occur through either an explicit BASIC dual register write S/D field, or with a normal GPR write destination field with a MISC/WRITE WR[6] or MISC/WRITE SC field.

2. PR[9:F] can not be specified as source or destination in MXPR, FBOX XFER, or MEM REQ microinstructions.

3. T[B] can not be used with the SPECIAL MISC2 function WRITE T[A].

4. T[C:D] can only be used as a source, and only in BASIC microinstructions with source/dest codes = C or E.

5. T[E] can only be used as a source, and only in BASIC microinstructions with source/dest codes = C or E or in FBOX XFER and MEM REQ microinstructions with source/dest code = 1.

6. T[F] can only be used as a destination, and only in BASIC microinstructions with source/dest codes = D or F.

7. T[C:E] are read only; T[F] is write only.

3.10.2 PC Logic -

1. The PC can not be loaded as GPR[F] or GPR[RN].

2. The PC can not be addressed as GPR[RN+1].

3. All PC loads write all 32 bits of the PC.

3.10.3 K Mux -

No known restrictions.

3.10.4 SC Logic -

1. SC can not be tested (via Branch Condition Select) immediately following a load from the D_Bus. This includes writes via microinstructions MXPR, FBOX XFER, or MEM REQ.

2. When executing a MISC/WRITE SC, the derived data length must be LONG when a GPR is being written in parallel.

3. The following SC operations must be mutually exclusive:

   - Load as WR[7]
   - Load via MISC/WRITE SC
   - Increment by one, MISC operation
   - Load of SC from RN, MISC operation
   - Any SPARE function

3.10.5 ALU -

1. During BASIC PASS B short literal, the following must NOT be selected as the A_Bus source:

   - T[C] (VA')
   - T[D] (VIBA)
   - PR[A] (VA)
   - PR[D] (ALU CC's and STATE flags)

3.10.6 Shifter -

1. DL can not be used as a shift value immediately after being loaded as a PR.

3.10.7 CC Logic -

1. The ALU CC register must not be loaded as PR[C] AND via CC field in same microinstruction.

2. The PSL CC register must not be loaded as PR[D] AND via CC field in same microinstruction.

3. The PSL CC register must not be loaded from ALU CC register at the same time the ALU CC register is loaded.

4. An MXPR to PSL is not allowed during IID (TP load conflict).

5. Neither the ALU CC's nor the PSL CC's can be tested via a Branch Condition Select immediately following a load as a PR. This implies, in particular, that the PSL CC's can not be loaded as a PR during IID.

6. MISC/USE OLD Z can not be used with FPU XFER microinstructions.

3.10.8 RLOG -

1. Push/pop of RLOG is not allowed during IID.

2. Pop of RLOG is not allowed during NSD.

3.10.9 State -

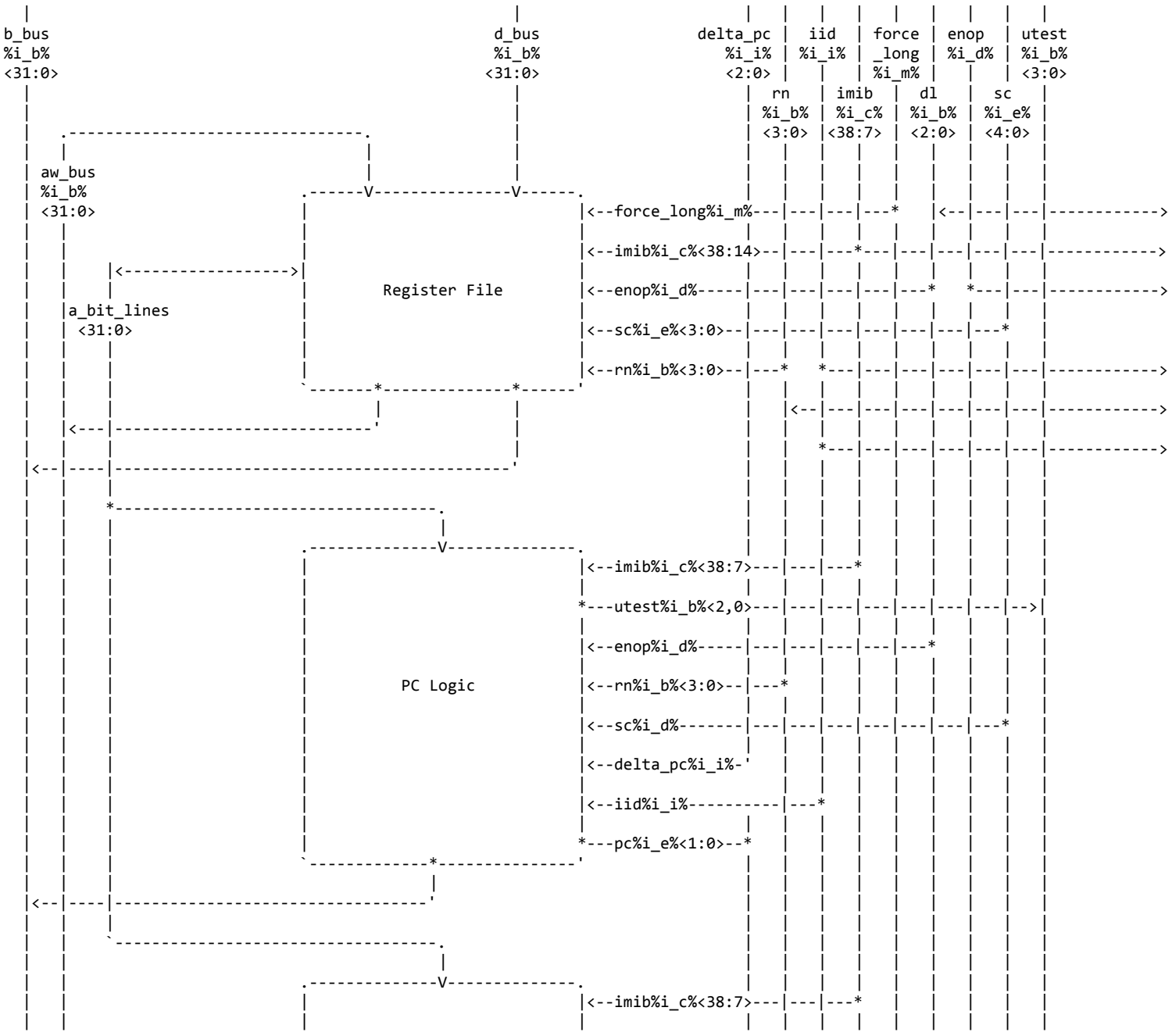
1. The following operations must be mutually exclusive:

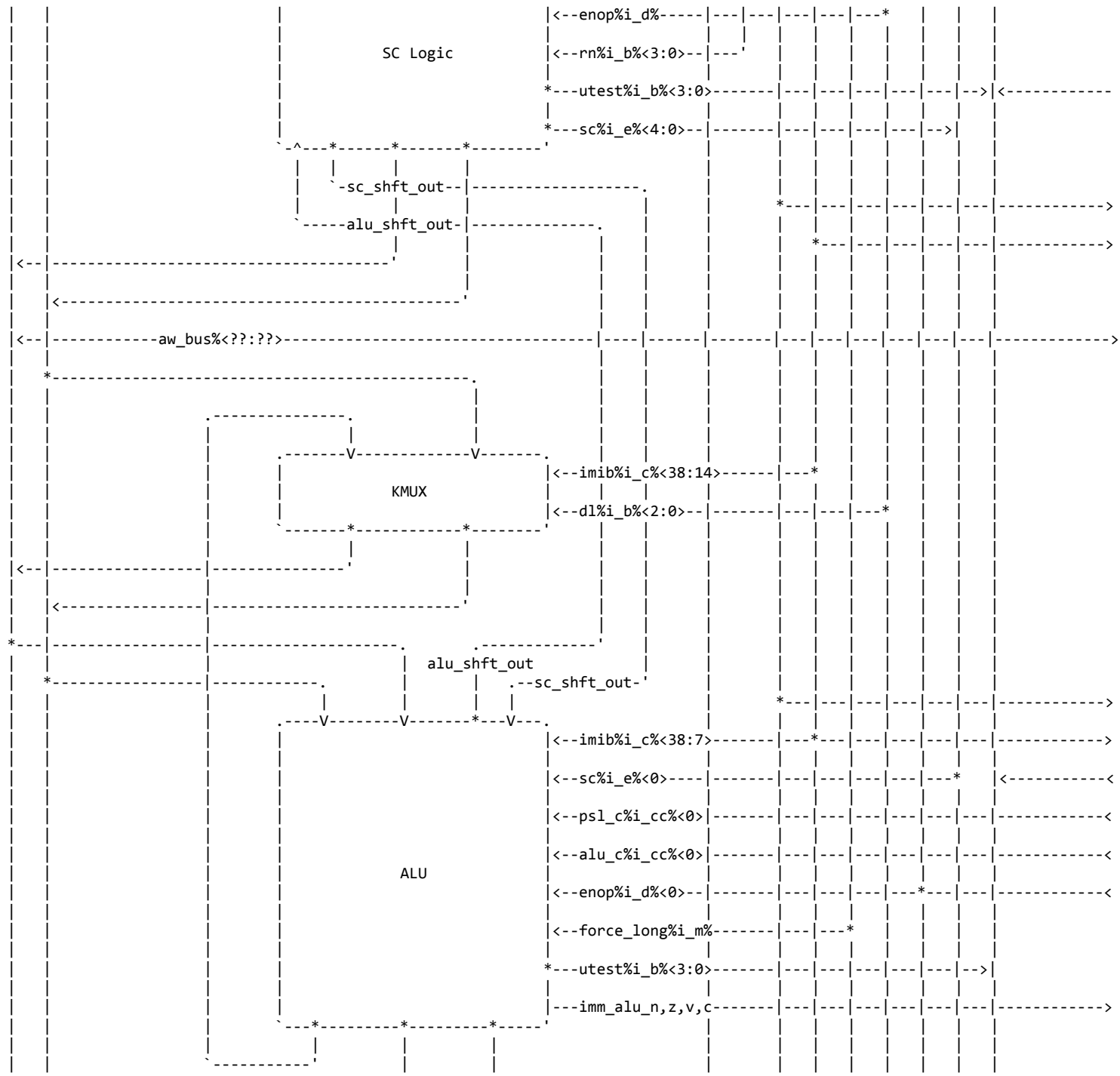
   - Load of STATE<3:0> as PR[D]
   - Set/clear of any of STATE<3:0> via MISC field

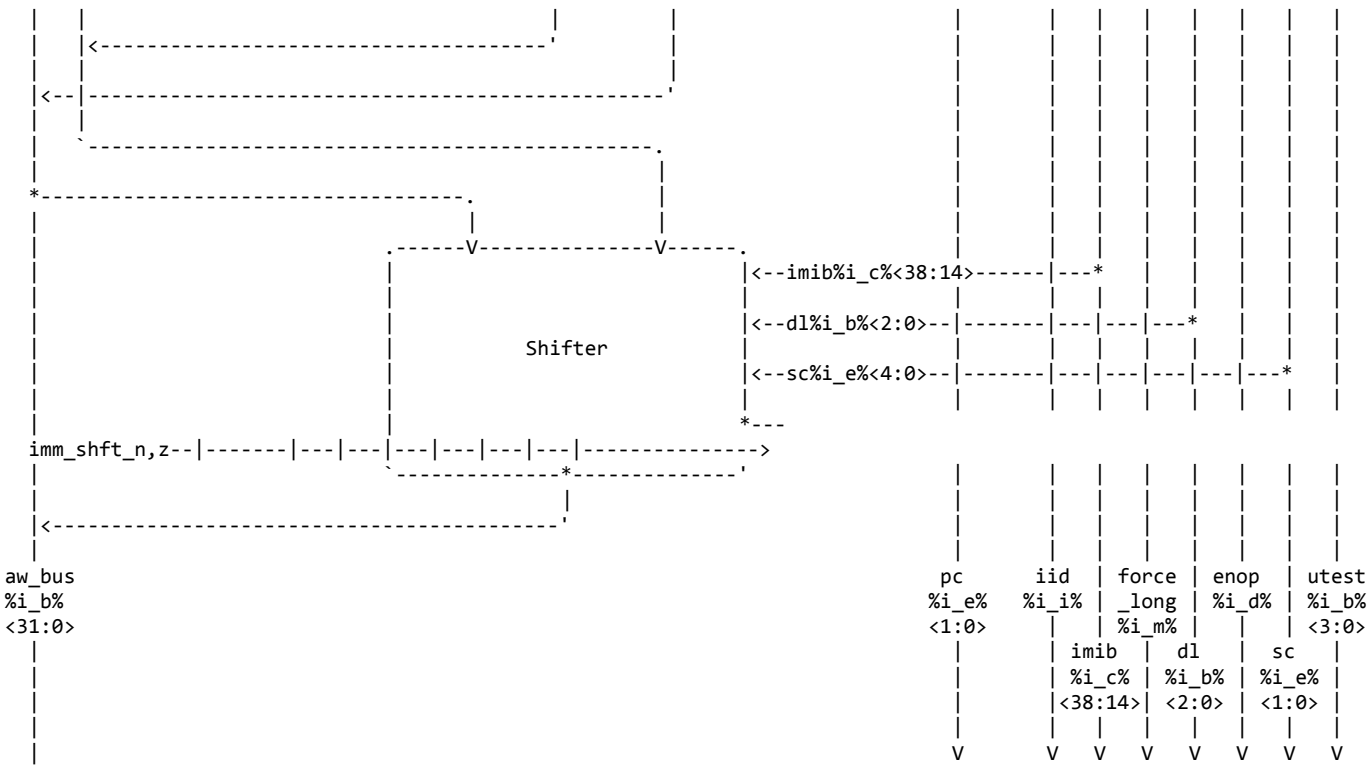
2. STATE<3:0> may not be tested immediately following loading as PR[D].

3.11 Block Diagrams

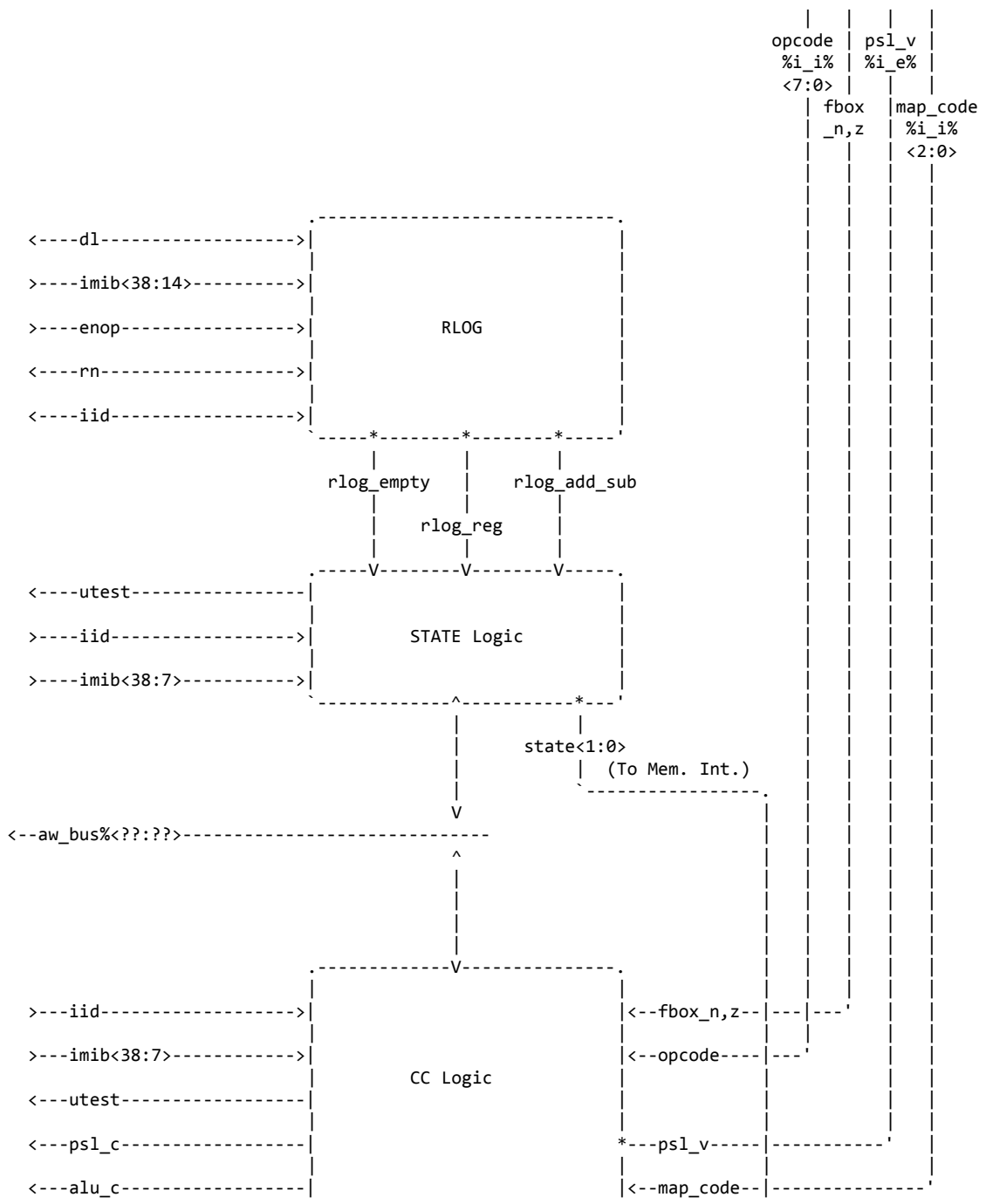
E Box Data Path Block Diagram

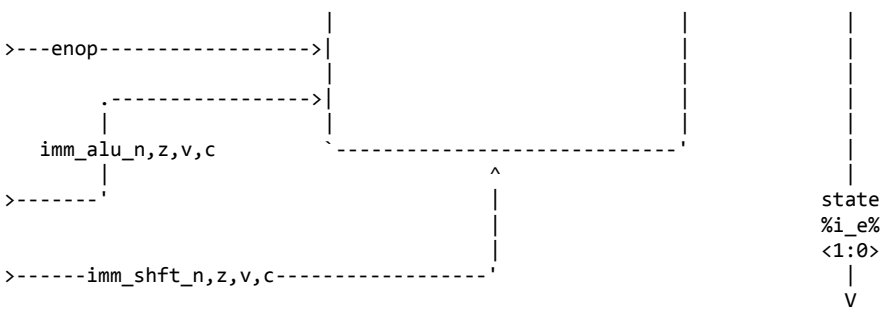



SC Logic
sc_shft_out
alu_shft_out
KMUX
ALU
aw_bus%<??:??>
enop%i_d%
rn%i_b%<3:0>
utest%i_b%<3:0>
sc%i_e%<4:0>
imib%i_c%<38:14>
dl%i_b%<2:0>
imib%i_c%<38:7>
sc%i_e%<0>
psl_c%i_cc%<0>
alu_c%i_cc%<0>
enop%i_d%<0>
force_long%i_m%
utest%i_b%<3:0>
imm_alu_n,z,v,c


Shifter
imm_shft_n,z
aw_bus
%i_b%
<31:0>
<38:14>
<2:0>
<4:0>
pc
%i_e%
<1:0>
iid
%i_i%
force
_long
%i_m%
enop
%i_d%
utest
%i_b%
<3:0>
imib
%i_c%
dl
%i_b%
sc
%i_e%

E Box Non-Data Path Functions Block Diagram




>---enop---------->
imm_alu_n,z,v,c
>-------'
>------imm_shft_n,z,v,c----------
state
%i_e%
<1:0>

4.0 MEMORY (M) BOX

The Memory (M) Box is responsible for memory management in MicroVAX. The M Box description is divided into two sections: an M Box overview and a detailed M Box functional description.

4.1 M Box Overview

The M Box Overview provides a general discussion of the operation of the M Box. It describes the individual microinstructions which directly control the M Box. A flow chart of the various events that can occur during an address translation is presented. Also included is a chart of all the M Box registers.

4.1.1 Introduction -

The M Box is responsible for memory management. The M Box's primary function is to translate virtual addresses into physical addresses. In addition to translating the virtual address it must determine if the process has the required privilege to reference the requested page, if the page is swapped out, if a page is being modified for the first time, if there is a length violation, and if the reference crosses page boundaries.

The M Box informs the microcode of various exceptions by generating a microtrap and/or driving specified bits on the Microtest bus. During certain M Box generated microtraps, the hardware asserts the memory management trap disable bit (MMGT_TD).

The DAL Interface is controlled by the M Box. Therefore the M Box must determine if the memory reference requires more than one DAL transaction. The microcode can never request more than one longword per microinstruction. For any given longword reference it is possible for the address to be unaligned. Whenever this occurs the M Box informs the DAL Interface of this condition by asserting REQ_2ND_REF. The M Box will then use VA' for addressing the second longword of data. When this occurs a single MEMORY REQUEST (MREQ) microinstruction causes two bus cycles to occur.

4.1.2 Microinstruction Control Of The M Box -

There are many types of microinstructions that control the M Box. The microinstruction type that generates the most control of the M Box is the MEMORY REQUEST type of microinstruction which causes memory requests to occur. Other types of microinstructions such as BASIC, MXPR, and SPECIAL, also control the M Box and are used for reading and writing registers in the M Box and for controlling the Microtest bus.

4.1.2.1 Memory Requests -

The MEMORY REQUEST (MREQ) microinstructions cause various functions to be executed in the M Box. The MEM FUNC (Memory Function) CS<32:28> field of the MREQ microinstruction specifies the type of function to be performed. The table below lists all the memory request functions. The privilege check column indicates the type of access check requested by the particular memory reference. The access mode column indicates the mode, or the source of the mode, for use in privilege checking the memory request's access to a given page. Pages can have access restrictions placed on them such as read only in kernel mode. If the access requested by the memory reference (privilege check/access mode) is not allowed by the protection code associated with the page, then an access control violation will occur. This will cause a microtrap if it is a virtual request, if it is not a probe, and if traps are not disabled.

MEMORY REQUEST FUNCTION FIELD DEFINITION

	Function	Reference Type	Privilege, M=0 Check	Access Mode	Data Length	Address Source	VA' after Reference	
VIRTUAL								
00	Read	Virtual	Read	PSL	DL	VA	VA+4	
08	Read	Virtual	Read	PSL	LONG	VA	VA+4	
01	Read	Virtual	(AT)	PSL	DL	VA	VA+4	
02	Read	Virtual	Write	PSL	DL	VA	VA+4	
03	Read Lock	Virtual	Write	PSL	DL	VA	VA+4	
18	Read	Virtual	None	-	LONG	VA'	VA'+4	
04	Write	Virtual	Write	PSL	DL	VA	VA+4	
09	Write	Virtual	Write	PSL	LONG	VA	VA+4	
05	Write Unlock	Virtual	Write	PSL	DL	VA	VA+4	
19	Write	Virtual	None	-	LONG	VA'	VA'+4	
11	Write	Virtual	None	-	LONG	VA'	VA'	
PHYSICAL								
0A	Read	Physical	None	-	LONG	VA	VA+4	
0B	Write	Physical	None	-	LONG	VA	VA+4	
1A	Read	Physical	None	-	LONG	VA'	VA'+4	
1B	Write	Physical	None	-	LONG	VA'	VA'+4	
KERNEL								
10	Read	Virtual	Read	Kernel	LONG	VA'	VA'	
PROBE								
06	Probe	Virtual	Read	PSL	-	VA	VA'	Aligned only
07	Probe	Virtual	Write	PSL	-	VA	VA'	Aligned only
0E	Probe	Virtual	Read	STATE	-	VA	VA'	Aligned only
0F	Probe	Virtual	Write	STATE	-	VA	VA'	Aligned only
16	Probe	Virtual	Read	Kernel	-	VA'	VA'	Aligned only
RD PTE								
12	Read SPTE	Physical	None	-	LONG	VA'	VA'	Aligned only
13	Read PPTE	Virtual	Read	Kernel	LONG	VA'	VA'	Aligned only
RD INT								
0C	Read Interrupt Vector	Physical	None	-	LONG	VA=LEVEL	VA+4	Aligned only

4.1.2.1.1 Microcode Restrictions -

The following is a list of microcode restrictions and general facts concerning the use of the M Box.

1. Writes must not result in bus errors if an explicit write to VA' is possible before the write completes.

2. A memory cycle immediately following a RD PTE must use the same PTE that was just written. If an MREQ microinstruction does not follow a RD PTE, then prefetching must be disabled during the microcycle following a RD PTE to avoid an attempt to access a PTE different than was just written. An MREQ physical also cannot follow a RD PTE because the physical address will conflict with the PTE being read.

3. The SPECIAL microinstruction's MISC3 field must not use encodings 8, 9, D, E, F, 10, or 15.

4. A MISC load VA or load VA' must not occur during an MXPR because the E box is not driving at this time.

5. A BCS MEM REF OK or BCS NOT VA MEM REF OK should not occur during an MREQ that loads the MMCSR.

6. If the response to an FBOX XFER microinstruction with an F BOX OP field equal to FPU RESPONSE ENABLE is sampled, then the immediately following microinstruction's branch field must be NOT.FPU.SIGNAL.

4.1.2.1.2 Virtual References -

4.1.2.1.2.1 Read Virtual -

Execute a READ function. Use VA as address. Translate using TB. Check for read privilege. Data length is given by the microinstruction and is either DL or LONG. DL is the data length received from the I Box.

4.1.2.1.2.2 Read Virtual (AT) Check -

The operation is READ but check for read privilege if Access Type (AT) = Read Source, Address Source, or Field Source; or for write privilege if Access Type (AT) = Modify Source.

4.1.2.1.2.3 Read Virtual Write Check -

The operation is READ but check privilege as if it were a WRITE.

4.1.2.1.2.4 Read Virtual Write Check Lock -

Same as Read Virtual Write Check but issue a READ LOCK function on the system bus.

4.1.2.1.2.5 Read (VA') -

Same as Read Virtual but use VA' as address. No privilege checking is done; data length is LONG. Note that on virtual references with no privilege checking, the TB.V/ TAG, PTE.V bit, and the PTE.M bit are still examined. Cross page checking only occurs on VA references.

4.1.2.1.2.6 Write Virtual -

Execute a WRITE function. Use VA as address. Translate using TB. Check for write privilege. Data length is given by the microinstruction and is either by DL or LONG.

4.1.2.1.2.7 Write Virtual Unlock -

Same as Write Virtual but issue a WRITE UNLOCK command on the system bus.

4.1.2.1.2.8 Write (VA') -

Same as Write Virtual but use VA' as address. No privilege checking is done; data length is LONG. Note that the TB.V/ TAG, PTE.V bit, and the PTE.M bit are still examined. Cross page checking only occurs on VA references.

4.1.2.1.3 Physical References -

4.1.2.1.3.1 Read Physical -

Execute a READ function. Use VA as address. Do not translate; bypass TB. Data length is LONG.

4.1.2.1.3.2 Write Physical -

Execute a WRITE function. Use VA as address. Do not translate; bypass TB. Data length is LONG.

4.1.2.1.3.3 Read Physical (VA') -

Same as Write Physical but use VA' as the address. Data length is LONG.

4.1.2.1.3.4 Write Physical (VA') -

Same as Write Physical but use VA' as the address. Data length is LONG.

4.1.2.1.4 Kernel References -

Execute a READ function but check privilege as if a read in kernel access mode. Use VA' as address. Data length is LONG.

4.1.2.1.5 Probe References -

Probe references are used to determine if a memory data transfer will be successful, ie, whether memory management will allow it. No data transfers are done under any circumstances. The probes test reads and writes of bytes using the PSL current mode, STATE<1:0>, or kernel as the mode for access checking. VA' is unchanged. Probes can only take TB miss microtraps. Probes do not take ACV/TNV microtraps, because this would result in an exception, and a probe is only a test of the success of a memory reference. Probes do not take M = 0 microtraps, because a probe does not actually modify memory. Probes are forced aligned. Cross page traps cannot occur on probes because probes are forced aligned. Probes do not take M = 0 microtraps. The MMCSR is loaded during probe microinstructions. This allows the microcode to test whether there has been an access violation or a translation not valid.

4.1.2.1.6 Read PTE References -

Do a physical access of memory for Read SPTE and a virtual access of memory for Read PPTE. Put the rotated data into the least recently used entry of the TB. Also write the data to a destination register, if specified. The Least Recently Used (LRU) pointer indicates the least recently used entry of the eight entries in the TB. VA' is unchanged. The tags are unaffected by a Read SPTE but will be written on a Read PPTE with a miss. An aligned reference always takes place. Even if VA<1:0> do not equal zero, no second reference takes place, and the DAL Interface read rotator does not do a rotation.

4.1.2.1.7 Read Interrupt Vector -

Execute an interrupt identify sequence. The decoded level is passed to the bus via the VA. The TB is bypassed as in physical reads. In response to this command, the memory system responds with the interrupt vector received from the interrupting device. An aligned reference always takes place. Even if VA<1:0> do not equal zero, no second reference takes place, and the DAL Interface read rotator does not do a rotation.

4.1.2.2 Non Memory Request Microinstructions -

Besides the Memory Request type of microinstruction, a number of other types of microinstructions and fields of microinstructions control the M Box. The MXPR microinstruction is used for reading and writing the three length registers and the MAPEN register. It is also used for writing the partial PSL register. The BCS field of the microinstruction is used to control driving the Microtest bus. The A field of microinstructions that contain an A field is used to read VA, VA', VIBA, and write MMCSR. The MISC field is used to clear MMGT Trap Disable, set REXE, and write VA and VA'. The SPECIAL microinstruction is used to read VA, invalidate all TB entries, clear REXE, and set REPROBE/invalidate a single entry. The CONSTANT microinstruction is used to write VA and disable MISC decodes. Other microinstructions or groups of microinstructions are used to generate signals for the DAL Interface and the E Box.

4.1.3 Translation Buffer Description -

The Translation Buffer is an eight entry, fully associative cache of virtual to physical address translations. Virtual address translations can result in either a TB hit or a TB miss as follows:

TB Hit - There is a TB hit if bits <31:9> of the virtual address undergoing translation matches one of the tags in the TB and the TB.V stored in the corresponding PTE is asserted. When there is a TB hit, TB_MISS remains de-asserted. The TB drives the physical address out onto the IDALs and DALs.

TB Miss - There is a TB miss if the REPROBE flag is not set and bits <31:9> of the virtual address undergoing translation does not match any tag, or if it matches a tag and the TB.V bit stored in the corresponding PTE is de-asserted. If there is a miss during a D stream translation, then a TB miss microtrap is taken if the trap disable flag is not set. If there is a miss during an I stream prefetch, then no microtrap is taken, and the IBFILL_ERROR signal is asserted. On A D stream TB miss, the TAG pointed to by the LRU circuit is written, and the corresponding TB.V bit is cleared.

4.1.4 Microcode Flows -

The following tables present an overview of various memory reference sequences. The first table presents longword references, the second presents references that use DL.

The microcode flows is presented in tabular form. Each line in the table represents one machine cycle. First the MREQ microinstruction mnemonic is given. "----" in the microinstruction entry then that indicates that the previous microinstruction is still being executed. Then the source and destination fields are given. The following tables assume that all bus cycles complete in two microcycles (which is the fastest allowed by MicroVAX). The term REGn indicates any register allowed in the source/destination field. REQ_2ND_REF, MMGT_ERROR, and TB_MISS are the state of these signals for that given cycle. The entries under the VA, VA', and DL columns are defined as the state of these registers at the beginning of the cycle.

The terms WL and RL refer to the write latch and the read latch located in the DAL Interface, which are used to hold the first part of an unaligned reference. BM refers to the byte mask; the four bits following it refer to byte3, byte2, byte1, byte0. The term 'xxxx' indicates a don't care. Note that VA<7:0> and VA'<7:0> are given in hex.

4.1.4.1 Longword References -

microinstruction mnemonic	S/D	REQ	2ND_REF	MMGT_ERROR	TB_MISS	VA<7:0>	VA'<7:0>	DL	comments
ALIGNED LONGWORD READ									
READ	REG1	0	0	0	10	xx	LONG	DAL<-Address ; BM<-1111;	
----	----	0	0	0	10	xx	LONG	REG1<-DAL(data);	
ALIGNED LONGWORD WRITE									
WRITE	REG1	0	0	0	10	xx	LONG	DAL<-Address ; WL<-data ; BM<-1111;	
----	----	0	0	0	10	xx	LONG	Memory<-WL;	
UNALIGNED LONGWORD READ									
READ	REG1	1	0	0	12	xx	LONG	DAL<--Address; BM<-1100;	
----	----	1	0	0	12	xx	LONG	RL<- ROT(DAL);	
----	----	0	0	0	12	16	LONG	DAL<--Address; BM<-0011;	
----	----	0	0	0	12	16	LONG	REG1<-RL & ROT(DAL);	
UNALIGNED LONGWORD WRITE									
WRITE	REG1	1	0	0	12	xx	LONG	DAL<--Address; WL<-ROT(REG1); BM<-1100;	
----	----	1	0	0	12	xx	LONG	DAL<-WL;	
----	----	0	0	0	12	16	LONG	DAL<--Address; BM<-0011;	
----	----	0	0	0	12	16	LONG	DAL<--WL;	
ALIGNED LONGWORD READ WITH TB_MISS									
READ	REG1	0	1	1	12	xx	LONG	microtrap is forced; Uinstr. is aborted;	
UNALIGNED LONGWORD READ WITH TB_MISS									
READ	REG1	1	1	1	12	xx	LONG	microtrap is forced; Uinstr. is aborted;	

4.1.4.2 Memory References That Use DL -

All of the above examples assumed that the data length was equal to LONG. The following cases show what happens when DL is equal to a WORD and when DL is equal to a QUADWORD. Note that when a byte or word is read from memory, the upper bytes are zero extended before the destination register is loaded. Note that when a QUADWORD transaction takes place, it is convenient to use a MREQ using VA for the first reference and a MREQ using VA' for the second reference.

microinstruction mnemonic	S/D	REQ_2ND_REF	MMGT_ERROR	TB_MISS	VA<7:0>	VA'<7:0>	DL	comments
ALIGNED READ, DL = WORD								
READ	REG1	0	0	0	10	xx	WORD	DAL<-Address; BM<--0011;
----	----	0	0	0	10	xx	WORD	REG1<-- 00 & DAL;
ALIGNED WRITE, DL = WORD								
WRITE	REG1	0	0	0	10	xx	WORD	DAL<-Address; WL<-REG1; BM<--0011;
----	----	0	0	0	10	xx	WORD	DAL<-WL;
UNALIGNED READ, DL = WORD *								
READ	REG1	0	0	0	12	xx	WORD	DAL<--Address; BM <- 1100;
----	----	0	0	0	12	xx	WORD	REG1<- 00 & ROT(DAL);
UNALIGNED WRITE, DL = WORD *								
WRITE	REG1	0	0	0	12	xx	WORD	DAL<-Address; WL<-ROT(REG1); BM <- 1100;
-----	----	0	0	0	12	xx	WORD	DAL<-WL ;
ALIGNED READ USING DL WHERE DL = QUADWORD								
READ	REG1	0	0	0	10	xx	QUAD	DAL<-Address ;
----	----	0	0	0	10	xx	QUAD	REG1<-DAL;
READ VA'	REG2	0	0	0	10	14	QUAD	DAL<-Address;
----	----	0	0	0	10	14	QUAD	REG2<-DAL;
UNALIGNED READ USING DL WHERE DL = QUADWORD								
READ VA	REG1	1	0	0	12	xx	LONG	DAL<--Address;
----	----	1	0	0	12	xx	LONG	RL<- ROT(DAL);
----	----	0	0	0	12	16	LONG	DAL<--Address;
----	----	0	0	0	12	16	LONG	REG1<-RL & ROT(DAL);
READ VA'	REG2	1	0	0	12	16	LONG	DAL<--Address;
----	----	1	0	0	12	16	LONG	RL<- ROT(DAL);
----	----	0	0	0	12	1A	LONG	DAL<--Address;
----	----	0	0	0	12	1A	LONG	REG2<-RL & ROT(DAL);
ALIGNED READ THAT CROSSES PAGE BOUNDARY, DL = QUADWORD								
READ	REG1	0	1	0	FC	xx	QUAD	microtrap is forced; Uinstr. is aborted
UNALIGNED READ WITH TB_MISS, DL = QUADWORD								
READ	REG1	1	1	1	12	xx	QUAD	microtrap is forced; Uinstr. is aborted

* This word is unaligned in the sense that the lowest two address bits do not equal 00. It is not unaligned in the sense that the word crosses a longword boundary.

4.1.5 Registers -

This section is a summary of the registers and latches located within the M Box and the methods available for reading and writing these storage elements.

Register or latch	Bits	Primary Addressing Method	Access	Alternative Addressing Method
VA	<31:0>	WR/PR[A]	RW	MISC field = Load VA Destination of CONSTANT uinstr = VA Read during all SPECIAL uinstr
VA'	<31:0>	T[C] *	RW	MISC field = Load VA'
VIBA	<31:0>	T[D] *	RW	BCS=IF BCOND LOAD VIBA & PC ELSE IID BCS=LOAD VIBA & PC AND BRANCH ALWAYS Bits <1:0> read only zero
MMCSR	<3:0>	T[F] **	WO	Written with status during all MREQ's unless REPROBE set, however bits <4,2> are unchanged if MMGT_TD is set
P0LR	<31:9>	MXPR P0LR	RW	---
P1LR	<31:9>	MXPR P1LR	RW	---
SLR	<31:9>	MXPR SLR	RW	---
PSL (interrupt stack)	<26>	MXPR PSL.HWRE	WO	---
PSL (current mode)	<25:24>	MXPR PSL.HWRE	WO	---
MAPEN	<0>	MXPR MAPEN	RW	---
MBOXCSR	<1:0>			Written during MREQ microinstruction
MMGT_TD	<0>			Cleared by: (a) MISC field=CLEAR MMGT_TD (b) BCS:(IF MEM REF OK RETURN+BO) & MMCSR=1111 Set by hardware if a mmgt microtrap occurs during MREQ
MMGT_REXE	<0>			Cleared by: (a) IID (b) MREQ (c) SPECIAL MISC2=INVALIDATE_ALL (d) SPECIAL MISC1=CLR_REEXECUTE Set by MISC field=SET MMGT_REXE
REPROBE	<0>			Cleared by: (a) IID (b) MREQ (c) SPECIAL MISC2=INVALIDATE_ALL Set by SPECIAL MISC1=SET_REPROBE

Register or latch	Bits	Primary Addressing Method	Access	Alternative Addressing Method
TAG[7:0]	<31:9>			TAG[LRU] written with CMP Bus data during a D stream TB Miss
PTE[7:0]	<29:9,7:3,1>,TB.V			PTE[LRU] written from IDAL during MREQ PTE TB.V[LRU] set by MREQ PTE TB.V[7:0] cleared by SPECIAL MISC1=INVALIDATE ALL TB.V[LRU] cleared by: (a) SPECIAL MISC1= SET REPROBE (b) a TB miss TB.V[hit entry] cleared by: (a) hit with PTE.V=0;TNV does not have to occur (to kill TB.V on probes) (b) hit with M = 0 utrap

* Read only with BASIC microinstruction with source and destination field C or E
** Write with BASIC microinstruction with source and destination field D or F;
MMCSR<3,1:0> <-- AW_Bus<26:24>

4.2 Function Descriptions

The major functions of the M Box include the central microinstruction decoding logic, the memory address logic for producing the starting address for memory transactions, the TB for translating virtual addresses into physical addresses, the access logic for assuring that memory is protected and access is allowed to translated addresses, the microtrap logic for flagging memory management conditions requiring trap routines to correct them, unaligned/multiword reference logic for controlling DAL data formatting, and DAL second cycles and length checking logic.

The M Box passes E Box data as well as virtual or physical addresses to the DAL. Whenever addresses are delivered the data size logic transmits a two bit size code on the DAL as well.

4.2.1 Memory Address Logic -

The Memory Address Logic consists of the memory address registers, a longword incrementer, and associated control for supplying virtual addresses to the TB or physical addresses to the DAL. The Memory Address Logic consists of the following blocks.

4.2.1.1 VA (Virtual Address) Register -

The VA is a longword register used to hold the initial address for memory references. The address may be virtual or physical.

The VA register is never incremented.

The VA register is explicitly read or written as WR/PR[A]. It may also be written when the MISC field = LOAD VA or when the destination of a CONSTANT uinstr = VA.

The VA register is also read during all SPECIAL microinstructions. ENOP inhibits the writing of VA.

4.2.1.2 VA' (VA Prime) Register -

The VA' is a longword register used during unaligned memory references, multi-word memory sequences, and Memory Management microtrap processing.

The M Box selects the VA' register during MREQ microinstructions if the request uses VA' as the address source or during the second half of an unaligned data reference.

The VA' register is incremented during MREQ microinstructions, except during MREQ KERNEL, MREQ PROBE, MREQ PTE, the second bus cycle of an unaligned reference (to prevent two increments in an unaligned), or a

microtrap request (to preserve VA' for when the microinstruction is retried after the microtrap). The +4 adder requires two microcycles to add; VA' is loaded at the end of the second microcycle.

The VA' register is explicitly read as T[C] during BASIC microinstructions with the source and destination field equal to C or E. It is written when the MISC field = LOAD VA'.

4.2.1.3 VIBA (Virtual Instruction Buffer Address) Register -

The VIBA is a longword register used to hold the address of the instruction stream that is being pre-fetched. IB Fills use the VIBA register for the instruction address. The VIBA register is selected if the current microinstruction is not a MREQ.

The VIBA register is incremented by four at the completion of successful instruction prefetches. During a prefetch, incrementing the VIBA register is inhibited unless the IB data is available and the IB prefetcher is ready to accept it. Furthermore, since a load from the VA Adder and a microcode write to VIBA may occur at the same time in the cycle, the explicit write always takes precedence. The signal IB_DATA_PRS is generated by the DAL Interface when the above conditions are true and is used by the M Box to load VIBA from the adder. The M Box determines when VIBA will be written and sends WILL_LOAD_VIBA to the DAL Interface. If the DAL Interface receives WILL_LOAD_VIBA then IB_DATA_PRS is inhibited. The potential conflict occurs during a BRANCH microinstruction.

The VIBA register is explicitly read as T[D] during BASIC microinstructions with the source and destination field equal to C or E. It is written via a BRANCH microinstruction when BCS = IF BCOND LOAD VIBA&PC ELSE IID and the specified branch condition is met, or when BCS = LOAD VIBA & PC AND BRANCH ALWAYS.

4.2.1.4 VA Adder -

The 32-bit VA adder is used to increment longword addresses from VA, VA' or VIBA by four. The result can be loaded into the VA' or VIBA register. If a MREQ microinstruction is not executing, then VIBA + 4 is calculated by the adder.

4.2.2 TB -

The Translation Buffer is an eight entry fully associative cache for fast virtual to physical address translation and supplying information to the Access Logic for page protection.

The TB is utilized in the following three ways:

1. TB access - This is the normal virtual to physical address translation process which results in the driving of the physical address onto the DALs. In the case of an TB hit, a portion of the PTE is sent to the Access Logic to determine whether the access should be allowed. This process is described in the TB Data Path.

2. TB Fill from Memory - This operation loads a new PTE into the TB from Memory as a result of a Read PTE. This process is described in the TB Read/Write Control block.

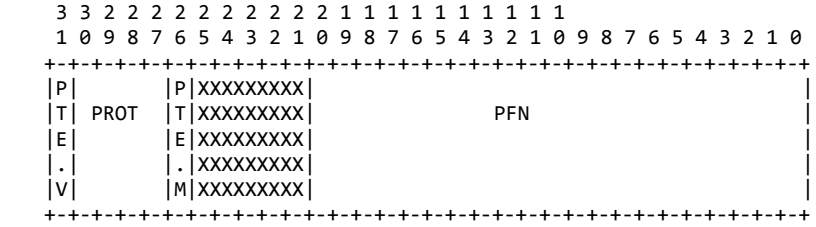
3. Invalidate - This operation invalidates one or all of the PTEs. It is described in the TB Invalidate Logic block. Single PTEs are invalidated by the SPECIAL MISC1= SET REPROBE microinstruction which sets TB.V[LRU] = 0. If TB.V is de-asserted the state of the rest of the PTE is irrelevant. The SPECIAL MISC2= INVALIDATE ALL microinstruction causes all TB.V bits to be cleared.

4.2.2.1 PTEs -

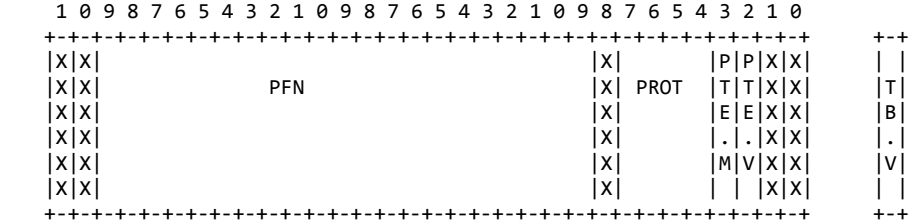
The TB_PTEs are stored in the TB in rotated format. The unrotated PTEs, corresponding to the format in memory, are called VAX_PTEs because the field assignments correspond to those defined in the SRM.

The unrotated and rotated PTE's are shown below. Note that all bits are not rotated by the same amount. The PTE.V bit is rotated left by 3 bits, all the other bits are rotated left by 9 bits.

VAX_PTE (Unrotated PTE)



TB_PTE (Rotated PTE)



Note that the X's indicate unused bits. These bits are read only ones.

PTE.V is the PTE Valid bit. It governs the validity of the Modify (M) bit and the Page Frame Number (PFN). When PTE.V = 1, M and PFN are valid; when PTE.V = 0, M and PFN are reserved.

PROT is the PTE Protection code. This field is always valid in the VAX_PTE but is only valid in the TB_PTE if the TB.V is valid. The PROT field describes the accessibility (read, write, no access) of the given page for each mode (kernel, executive, supervisor, user). Refer to the VAX Architecture Standard for a table of protection codes.

PTE.M is the PTE Modify bit. If PTE.M = 1, the page has been recorded as modified; if PTE.M = 0 the page has not been recorded as modified. It is used only if PTE.V = 1. It is SET by microcode on a valid, access-allowed modify or write memory access.

PFN is the Page Frame Number which is the upper 21 bits of the physical address. This page address is valid only if PTE.V = 1.

TB.V is the TB Valid bit. There is a TB.V bit associated with each PTE. The TB.V bit is not part of the PTE or TAG but is in the control logic. When TB.V = 1, the TB_PTE entry is valid and corresponds to a VAX_PTE entry. When TB.V = 0, the entry is invalid. A SPECIAL MISC1 = INVALIDATE ALL microinstruction may be used to invalidate the entire TB. A SPECIAL MISC1 = SET REPROBE or a TB miss will clear TB.V[LRU]. There is a TB hit when there is a tag match and the TB.V bit associated with the PTE corresponding to that tag is asserted. When there is an TB hit but PTE.V

= 0 or an M = 0 microtrap, hardware clears the TB.V bit associated with the accessed PTE. When a PTE is read from memory into the TB via a MREQ PTE, TB.V is set by the M Box.

4.2.2.2 LRU -

When a D stream TB miss occurs, a new tag must be written with the PFN of the virtual address that resulted in a miss. This new tag must be written over an existing entry in the TB. Statistics show that the least recently used entry is the best one to replace. The LRU circuit determines which is the least recently used entry. The LRU circuit consists of an 8 entry stack. Whenever there is a hit, the number of the PTE/TAG pair that hit is encoded and is pushed into the most recently used position in the stack. At the same time, all entries between the most recently used position and the previous position in the stack of the entry that hit are pushed. This removes the hit entry from its previous position and moves it to the most recently used position. Both D stream and I stream hits cause the LRU to cycle.

4.2.2.3 TB Data Path -

The TB data path consists of eight TB entries (which can be used for either I or D stream), plus logic to facilitate data formatting and transfers described below. Each TB entry consists of a 23-bit Tag/Comparator register (CAM) and a 32-bit PTE register (of which <31,30,8,2,0> are unused).

If the Memory Management Enable (MME) bit of the MAPEN register is CLEAR, or if the reference is physical, the TB is bypassed, and the virtual address is transmitted as the physical address. Otherwise, a TB access is performed as described below.

Note that an TB hit requires in general that two conditions be met: first that the TAG and the appropriate field of the virtual address agree, and second that the TB.V bit of the TB_PTE pointed to by the TAG is asserted.

4.2.2.3.1 I Stream Access -

Whenever the microinstruction is not a MREQ, and IB_REQ is asserted, and IB_CYC is asserted, the TB attempts an address translation using VIBA. IB_CYC is generated by the DAL Interface whenever the arbitration logic has determined that the bus will be available for a prefetch. IB_REQ is generated by the I Box whenever prefetch data is desired. Even though the TB attempts a translation, a prefetch will not occur (address strobe will be inhibited) if the translation results in MMGT_ERROR being asserted.

4.2.2.3.1.1 I Stream TB Hit -

During a virtual I stream access, the fully associative TB_TAGS are compared to bits <31:9> of the VIBA register. If the comparison is successful and TB.V is asserted, and IB_CYC and IB_REQ are asserted, there is a TB hit. The LRU is cycled. A physical address composed from the TB_PTE and the VIBA is passed to the DAL. The low nine bits, TB_PTE<8:0>, are examined by the Access Logic to determine if the access is allowed or whether the IBFILL_ERR signal should be asserted. No microtraps are ever taken, and the Memory Management Case Status Register in the Access Logic is not altered.

4.2.2.3.1.2 I Stream TB Miss -

If the comparison was unsuccessful, or if the tag comparison was successful but TB.V is not asserted (the TB_PTE may not correspond to the VAX_PTE), or if an access violation occurs, then the signal IBFILL_ERR is sent to the I Box to disable prefetching. No microtraps occur because the data being prefetched may not be used. Only when the prefetcher eventually runs out of valid data does the microcode force an I stream related memory management fault.

4.2.2.3.2 D Stream Access -

Besides I stream accesses described above, the TB is also activated whenever the microinstruction is a virtual MREQ, and MME is set, and the bus is not busy.

4.2.2.3.2.1 D Stream TB Hit -

On a D stream translation, the fully associative TB_TAGS are compared to bits <31:9> of either the VA or the VA' register, depending on the type of memory request. If the comparison is successful and TB.V is asserted, there is a TB hit. The LRU is cycled. For a TB hit, a physical address is passed to the DAL as for the I stream case. The low nine bits, TB_PTE<8:0> are examined by the Access Logic to determine if the access is allowed or whether a MMGT_ERROR should be issued and a Memory Management microtrap taken.

4.2.2.3.2.2 TB Miss -

If the comparison was unsuccessful, or if there was a successful comparison but TB.V was not asserted (the TB_PTE may not correspond to the VAX_PTE), then a TB_MISS occurs. A TB_MISS also occurs if the reference is a MREQ PPTE and the address is in process space. This is used to detect the error condition of a PPTE in process space. A tag is written

when a D stream miss occurs. If MMGT_TD = 0, and REPROBE = 0, then a TB MISS microtrap occurs.

4.2.2.3.3 TB Data Formatting -

This logic properly formats addresses delivered to the DAL.

4.2.2.3.3.1 TB Bypasses -

If the MME bit of the MAPEN register is CLEAR, or if the MREQ is physical, then the contents of one of the address registers (VIBA, VA', or VA), bits <29:0>, are passed to the DAL. The data size bits are placed on <31:30> by the Memory Data Size Control.

4.2.2.3.3.2 TB Accesses -

Whenever there is a TB hit, the physical address must be constructed and driven onto the DAL during the address portion of the cycle. The physical address is built as follows. The upper two bits of the DAL (the data size of the request) are driven by the Memory Data Size Control. The next 21 bits are the physical page number and are driven by bits <29:9> of the TB_PTE. The low nine bits of the DAL are driven from the selected address register, bits <8:0>. The low nine bits of the TB_PTE, the Access Control bits, go to the Access Logic for validation.

4.2.2.4 TB Fills From Memory -

Memory fills to the TB are performed during Read PTE microinstructions. The TB fill happens during the execution of that microinstruction.

During a Read PTE, the LRU stack identifies the TB_PTE to be written. The associated tag was written when the TB miss occurred. The TB_PTE is written from the DAL via a hardwired left rotation using the LRU as a pointer.

Whenever the TB is loaded from memory, the TB.V bit is ASSERTED by hardware, indicating that the entry is valid.

4.2.2.5 TB Invalidate Logic -

The TB entries may be invalidated, the TB.V bits cleared, either collectively or individually. The reason for marking entries invalid is to prevent TB hits.

The SPECIAL microinstruction is used to invalidate the entire TB (via MISC2 = INVALIDATE_ALL), while the individual entry pointed to by the LRU may be invalidated by SPECIAL MISC1 = SET_REPROBE. When TB entries are loaded from memory, they are marked valid by the M Box.

When there is an TB Hit but PTE.V is not asserted, hardware clears the TB.V bit in the accessed PTE to prevent future incorrect TNV microtraps. Since the software can validate a PTE without notifying the hardware, unless the TB entry is replaced with an updated version of the PTE, TB hits with TNV microtraps will continue to occur (at that TB location) even after the PTE in memory has become valid. Thus TB.V is cleared by hardware to cause a TB Miss to occur and a new entry to be loaded in the TB. Note that I stream references cannot cause TNV's (or any other microtrap).

4.2.2.6 TB Miss Logic -

If the microinstruction is a virtual MREQ, and memory management is enabled, and there is no TB hit, then TB_MISS is asserted. If a memory reference using VIBA is attempted, and memory management is enabled, and there is no TB hit, then TB_MISS is asserted.

TB_MISS is asserted only if there is no match in the TB. A match means that the Tag comparison was successful and that the selected PTE's TB.V bit was asserted.

4.2.2.7 TB Summary -

```
! No TB translation attempted

If NOT(MME) OR PHYSICAL MREQ
    Then
        If VIBA is the selected address source
            Then
                DAL <== PA[<31:30>=DATA SIZE<1:0>, <29:0>=VIBA<29:0>]

        If VA' is the selected address source
            Then
                DAL <== PA[<31:30>=DATA SIZE<1:0>, <29:0>=VA'<29:0>]

        If VA is the selected address source
            Then
                DAL <== PA[<31:30>=DATA SIZE<1:0>, <29:0>=VA<29:0>]


! I Stream TB translation attempted

If MME AND (NOT(MREQ) AND IB_CYC AND IB_REQ)
    Then
        If  TB Hit                                      ! TB Hit
```

```
            Then
                DAL <== PA[<31:30>=DATA SIZE<1:0>, <29:9>=TB_PTE<29:9>, <8:0>=VIBA<8:0>];

        If no TB Hit                                    ! TB Miss
            Then
                Assert IBFILL_ERROR

! D Stream TB translation attempted using VA' as the address source

If MME AND (MREQ using VA' OR second half of unaligned reference)

    Then
        If TB Hit                                       ! TB Hit
            Then
                DAL <== PA[<31:30>=DATA SIZE<1:0>, <29:9>=TB_PTE<29:9>, <8:0>=VA'<8:0>];

        If no TB Hit                                    ! TB Miss
            Then
                TB_MISS, TB MISS MICROTRAP

! D Stream TB translation attempted using VA as the address source

If MME AND MREQ using VA
    Then
        If TB Hit                                       ! TB Hit
            Then
                DAL <== PA[<31:30>=DATA SIZE<1:0>, <29:9>=D_STREAM_TB_PTE<29:9>, <8:0>=VA<8:0>];

        If no TB Hit                                    ! TB Miss
            Then
                TB_MISS, TB MISS MICROTRAP

! TB Fill from Memory, D Stream only

If Read PTE                                             ! Read SPTE or PPTE
    Then
        TB_PTE[LRU] <== [<29:9>=DAL<20:0>, <7:3>=DAL<30:26>, <2>=DAL<31>];

! TB Invalidate

If SPECIAL AND (MISC2 = INVALIDATE ALL)                 ! All entries invalidated
    Then
        TB_PTE[0:7]TB.V = 0

If SPECIAL AND (MISC1 = SET REPROBE)
   Then
        TB_PTE[LRU]TB.V = 0
```

4.2.3 Access Logic -

The Access Logic function determines if the memory reference request produces an access violation. It is made up of the following blocks.

4.2.3.1 Privilege Check Logic -

The Privilege Check Logic compares the TB PTE protection code bits to the mode bits specified by the MREQ function field and determines whether or not there is a privilege violation.

The Privilege Check Logic compares the PSL.CURM (the two CURRENT MODE bits from the PSL), STATE<1:0>, or kernel mode (00) with the protection code (PROT, TB_PTE<7:4>) to determine whether Read or Write Access is allowed to the addressed page for the given memory request. Read Access is granted whenever Write Access is granted.

Refer to the VAX Architecture Standard for a table of Read/Write Access for the four privilege modes as a function of the 16 protection codes.

A privilege violation occurs if a Read Access Check is requested and no Read Access is allowed, or if a Write Access Check is requested and no Write Access is allowed.

Note that Write Access is independent of PTE.M. However if the PTE.M is not set when a Write Access is determined, then the memory management microtrap M = 0 is taken to set PTE.M in the PTE stored in memory and in the rotated version in the TB.

A privilege violation during IB Fills can be determined by checking for a Read Access when fills are taking place.

If the MME bit in the MAPEN register (bit 0) is clear then privilege checking is disabled. If MME is set and the reference is to reserved address space (VA<31:30>=11), then a length violation occurs.

4.2.3.2 Length Check Logic -

The length check logic consists of three registers and a subtractor. The three registers are the System Length Register (SLR), Process Space 0 Length Register (P0LR), and Process Space 1 Length Register (P1LR). During a virtual memory request the upper two bits of the virtual address select one of the three length registers. The selected register is compared against the page frame number (PFN) of the virtual address. This comparison is performed by the length subtractor. For P0 and System address space, lenth violations occur if the PFN is greater than or equal to the associated length register. For P1 space, length violations occur if the PFN is less than the P1LR. For reserved system space (VA<31:30>=11), length violations always occur.

4.2.3.3 Inhibit IB Fill Logic -

When prefetching is halted and data is needed by the instruction buffer, microroutines are entered which load VIBA into VA and do a MREQ to bring the PTE into the TB if it is not already there and check for ACVs and TNVs. Microtraps are enabled. The result of this MREQ is not stored. Prefetching is restarted only if there is no ACV or TNV.

No ACV or TNV will occur when prefetching is started. However, there are other cases where an ACV or TNV may occur when VIBA is used. For example, the I stream may prefetch into a page whose PTE was already brought into the TB by the D stream. If mapping VIBA results in a TB Miss, ACV, or TNV, IBFILL_ERROR is asserted, but the microinstruction is allowed to continue. No memory reference is started.

4.2.3.4 Memory Management Case Status Register Logic -

The Memory Management Case Status Register (MMCSR) contains four status bits. MMCSR is clocked on every MREQ, including PROBE's, unless REPROBE is set; however, MMCSR<2> is not changed if MMGT_TD is set.

```
MMCSR<3>  =       0 for ACV, 1 for TNV or TB miss or reference ok
MMCSR<2>  =       0 for read, 1 for write or modify intent
MMCSR<1>  =       0 for ACV or TNV or TB miss, 1 for reference ok
MMCSR<0>  =       0 for ACV or TNV, 1 for TB miss or reference ok
```

With the above encoding, ALL legitimate values of MMCSR are generated automatically by hardware or within the memory management flows and are unique:

```
MMCSR<3:0> =      0X00 for process ACV (hardware)
                  0X01 for process length violation (microcode)
                  0X10 for ppte ACV (microcode)
                  0X11 for ppte length violation (microcode)
                  1X00 for process TNV (hardware)
                  1X01 for TB miss (hardware)
                  1X10 for ppte TNV (microcode)
                  1X11 for reference ok (hardware)
```

In all status situations, bit <2> denotes read vs write or modify intent.

MMCSR can be accessed as a case branch or written as a temporary register. When accessed as a case branch, its four bits are mapped directly onto the Microtest Bus. When written via T[F], MMCSR<3,1:0> are written from AW_Bus<26:24>. (MMCSR<2> is, in effect, read only.)

The MM Case Status Register is set to 1X11 if MME = 0 or if the reference is physical.

The definition of "memory reference ok" is MMCSR<3:0> = 1X11. This code is used for testing certain branch conditions. They are the following:

```
VA.MEM.REF.NOT.OK               ; for PROBE testing

IF.MEM.REF.OK.RET+BO            ; used in Read/Write Crossing Page Boundary
```

If BCS = IF MEM REF OK RETURN +BO and MMCSR is 1X11, the Microtest Bus is driven, UTEST<3:0> = RETurn + BO, to inform the microsequencer of the completion. If MMCSR is not 1X11, then UTEST<3:0> = NSI (next sequential instruction). This logic will also drive the Microtest Bus if the branch condition BCS = VA MEM REF NOT OK is detected. If BCS = VA MEM REF NOT OK and MMCSR is 1X11, then UTEST<3:0> = NSI and if MMCSR is not 1X11, then UTEST<3:0> = BRanch.

PROBE microinstructions must exercise the TB and Access logic to determine if the PTE is present and access is permitted. PROBE microinstructions take TB miss but do not take ACV/TNV, cross page, or M = 0 microtraps. The result of a probe is determined using the IF MEM REF OK RET+BO Branch microinstruction, the VA MEM REF NOT OK Branch microinstruction, or by examining MMCSR.

4.2.3.5 MBOX Case Status Register -

This two bit register is loaded during all MREQ microinstructions.

```
00 P0 Space Reference with no length violation
01 P1 Space Reference with no length violation
10 System Space Reference with no length violation
11 Length Violation
```

The contents of this register are tested by microcode by CASE[MBOX.STATUS].

4.2.3.6 Bus Error Case Status Register -

This register stores four bits of information during the last microcycle of a memory transaction. This register is needed to augment the MMCSR because write bus errors can take place long after the original MREQ was issued, resulting in the loss of needed information. The Bus Error Case Status Register feeds the microtrap logic to generate the microtrap vector.

4.2.4 Memory Management Microtrap Logic -

4.2.4.1 Partial PSL Logic -

The following processor status longword register bits are stored locally:

PSL.IS - INTERRUPT STACK flag, PSL<26>, indicates use of the interrupt stack. Although PSL.IS is only used for Microtest Bus testing, the bit is located in this block.

PSL.CURM - CURRENT MODE Bits, PSL<25:24>, indicates the privilege level of the currently executing program. PSL.CURM bits are used for access checking (see Privilege Check Logic). Testing by the Microtest Bus is also done on these bits.

These bits are loaded when an MXPR Write to the PSL is done. They may be tested via Branch microinstructions with BCS = NOT INT STACK, or BCS = KERNEL MODE. Depending upon the value of the PSL bits either the next sequential instruction or a branch is taken.

4.2.4.2 Cross Page Detection Logic -

The Cross Page logic examines the VA and the data length to determine if the reference will cross a page boundary. The Cross Page logic examines VA and data length only during references that use VA as the address source. The Cross Page conditions are summarized in the following table:

DATA LENGTH	VA<8:0>	IF	CROSS PAGE ?
BYTE	-------	--	NO
WORD	11111111x	x.NE.0	YES
LONG	1111111xx	xx.NE.00	YES
QUAD	111111xxx	xxx.NE.000	YES

MMGT_REXE disables the Cross Page Detection Logic. MMGT_REXE may be set and reset by microcode, and is reset when a MREQ microinstruction is executed with MMGT_REXE set. MMGT_REXE is also reset by IID.

After a cross page microtrap is taken, the microcode routine probes or reads on both sides of the page boundary. If everything is ok, then MMGT_REXE is set, and the original memory reference is re-tried. Setting MMGT_REXE inhibits the cross page microtrap during the retry. The trailing edge of the retry resets MMGT_REXE so that ensuing references will microtrap if they cross a page boundary.

4.2.4.3 Microtrap And Abort Determination Logic -

Before an access is deemed valid or not, one or more microtraps may occur. There are nine memory management microtraps initiated by the Microsequencer as a result of trap requests by the M Box or the external Error pin. The microtraps, in the order of their priority and with their

associated microtrap vectors are:

highest:	Cross Page	0
	TB Miss	1
	ACV/TNV	2
lowest:	M = 0	3
	Bus Error Read Virtual*	4
	Bus Error Read Physical*	5
	Bus Error Write Virtaul*	6
	Bus Error Write Physical*	7
	Bus Error Read Interrupt Vector*	D

* Bus Error microtraps do not occur with the other microtraps and are mutually exclusive between themselves.

The following describes when these microtraps are generated. If MMGT_TD is set, or if MME is clear, or if an MREQ PHYSICAL, then only bus error microtraps are enabled.

1. Cross page microtraps occurs as a result of a cross page condition (described above), and MMGT_REXE = 0, and MREQ VA, and not MREQ PROBE, and MMGT_TD = 0.

2. The TB miss microtrap occurs because of a miss in the TB, and REPROBE = 0, and MMGT_TD = 0.

3. The ACV/TNV microtrap is taken if not MREQ PROBE, and MMGT_TD = 0, and (the memory request has insufficient privilege or REPROBE is set or the page is not in memory). A page is not in memory when PTE.V is clear an there is a hit. When a page is not in memory (TNV), reading the PTE from memory to guarantee a fresh PTE is not necessary, since a TNV can only take place after a TB miss/fill routine and retry have taken place. Therefore, if a TNV takes place the PTE in the TB is fresh.

4. The M = 0 microtrap occurs if there is a TB hit, and the PTE.M bit is 0, and not MREQ PROBE, and there is a write check, and MMGT_TD = 0.

5. If the external Error pin is asserted during a D stream read or write, the transaction is terminated and one of five bus error microtraps is taken. If the Error pin is asserted during an I Stream read, then the transaction is terminated and IBFILL_ERROR is asserted, but a microtrap is not taken. The Bus Error microtrap cannot be disabled.

The I Box prefetcher needs to know when microtrap conditions occur so that it can halt loading of its instruction buffer stack. The memory management microtrap logic notifies the I Box of an IB fill error if abort conditions are detected and IB_FILL is asserted. Note that no microtraps are taken and ABORT is not asserted during attempted IB fills. If mapping VIBA results in a TB miss then the IBFILL_ERROR is asserted. If mapping VIBA results in a bus error, then prefetching is halted and a microtrap is NOT taken.

No memory management microtraps take place when VIBA is mapped. This is true because when IB is dry and halted, IIDs, SSDs, and NSDs branch to locations where VIBA is loaded into VA and a MREQ VA with microtraps enabled is used to bring the PTE into the TB and do validity, protection, and length checks. Bus error microtraps can also occur at this point. MicroVAX has no provision for memory management microtraps using VIBA.

If the IB is dry and prefetching is not halted, then IID or the appropriate specifier microinstructions branch on self until the IB is loaded.

4.2.5 Memory Management Controller -

4.2.5.1 Trap Disable Logic -

The Memory Management Trap (and Prefetch) Disable (MMGT_TD) flop is set by a TB miss, ACV/TNV, or M = 0 memory management microtrap (NOT, however, by a cross page or bus error microtrap), and cleared by a MISC/CLEAR MMGT_TD field in an executed microinstruction or by a BCS/IF MEM REF OK THEN RET+BO that actually executes a return. When set, MMGT_TD disables prefetching, inhibits further TB miss, ACV/TNV, or M = 0 microtraps (NOT, however, bus error microtraps), and inhibits changes to MMGT.STATUS<2> (see below). In particular, with memory management traps disabled, an M = 0 microtrap situation is ignored completely, and the memory access completes normally.

Memory Management Trap Disable, MMGT_TD, enables and disables I Box prefetching during Memory Management exceptions because it is necessary to preserve the LRU pointer to allow writing to the correct PTE. Note that probes may result in an ACV or TNV, and since no microtrap occurs, MMGT_TD is not set; if prefetching is desired to be disabled in this case it must be disabled using the MISC field= DISABLE.IB.PREFETCH (this does not set MMGT_TD).

4.2.5.2 Reexecute Reference Logic -

MMGT_REXE is used after the Memory Management microcode has made sure that a reference that trapped will now execute properly, and that cross page microtraps will not occur. MMGT_REXE is CLEARed after one memory reference by hardware. It is also CLEARed by IID and by SPECIAL microinstruction with MISC1 = CLEAR REEXECUTE or MISC2 = INVALIDATE ALL. This bit is SET with MISC = SET REEXECUTE.

4.2.5.3 REPROBE Flag -

REPROBE is set via SPECIAL MISC1 = SET REPROBE, which also clears TB.V in the currently selected TB entry. REPROBE is cleared by any MREQ, by IID, or by a MISC2 = INVALIDATE ALL microinstruction. When set REPROBE suppresses the execution of the next MREQ. No TB lookup occurs; MMCSR, and MBOXCSR are not updated. Instead, an ACV/TNV condition is declared. If the MREQ was not a probe, a microtrap occurs; if it was a probe, execution proceeds normally.

4.2.5.4 Memory Management Enable Logic -

The TB function is enabled by the MME bit being set. This bit is written by the microcode by writing to bit zero of REG ADDR = MAPEN with the MXPR microinstruction. It is read by reading bit zero of REG ADDR = MAPEN with the MXPR microinstruction. This function controls all memory management operations.

When MME is clear, the entire memory management mechanism is turned off. No traps will occur for any reason related to Memory Management, including during unaligned or cross page references. Addresses are not translated and are considered physical. Access privilege is not checked. When MME is set, address translation is enabled.

4.2.6 Second DAL Cycle Detection Logic -

This logic determines if the data transfer requested will require more than one cycle per four bytes (or less) of data. During normal, kernel mode, or physical MREQ's, the data length and VA (or VA') are examined. If no extra cycle is required, based upon the address and data length, REQ_2ND_REF is not asserted and control goes on to the next microinstruction. If an extra cycle is required, REQ_2ND_REF is asserted.

Probe, Read PTE, and Read Interrupt Vector memory request functions are always aligned. Using VIBA inhibits REQ_2ND_REF since all instruction reads are aligned.

The memory request microinstructions use the data length (DL) from the I Box or assume the length is long.

Once the correct data length is determined, the low two bits of VA (or VA') are examined to decide whether the (first/last) register-worth of data will take one or two DAL cycles to transfer from/to memory. When the microinstruction is determined to require two references, REQ_2ND_REF is asserted. The MREQ microinstruction is re-executed in the sense that another memory request is done. The M Box will use VA' on the second reference.

4.2.6.1 REQ_2ND_REF Logic -

The following table indicates when REQ_2ND_REF is asserted.

DATA LENGTH	VA<1:0>	REQ_2ND_REF
BYTE	--	NO...bytes are never unaligned
WORD	11	YES
LONG	01,10,11	YES
QUAD	01,10,11	YES

4.2.7 Memory Data Size Control Logic -

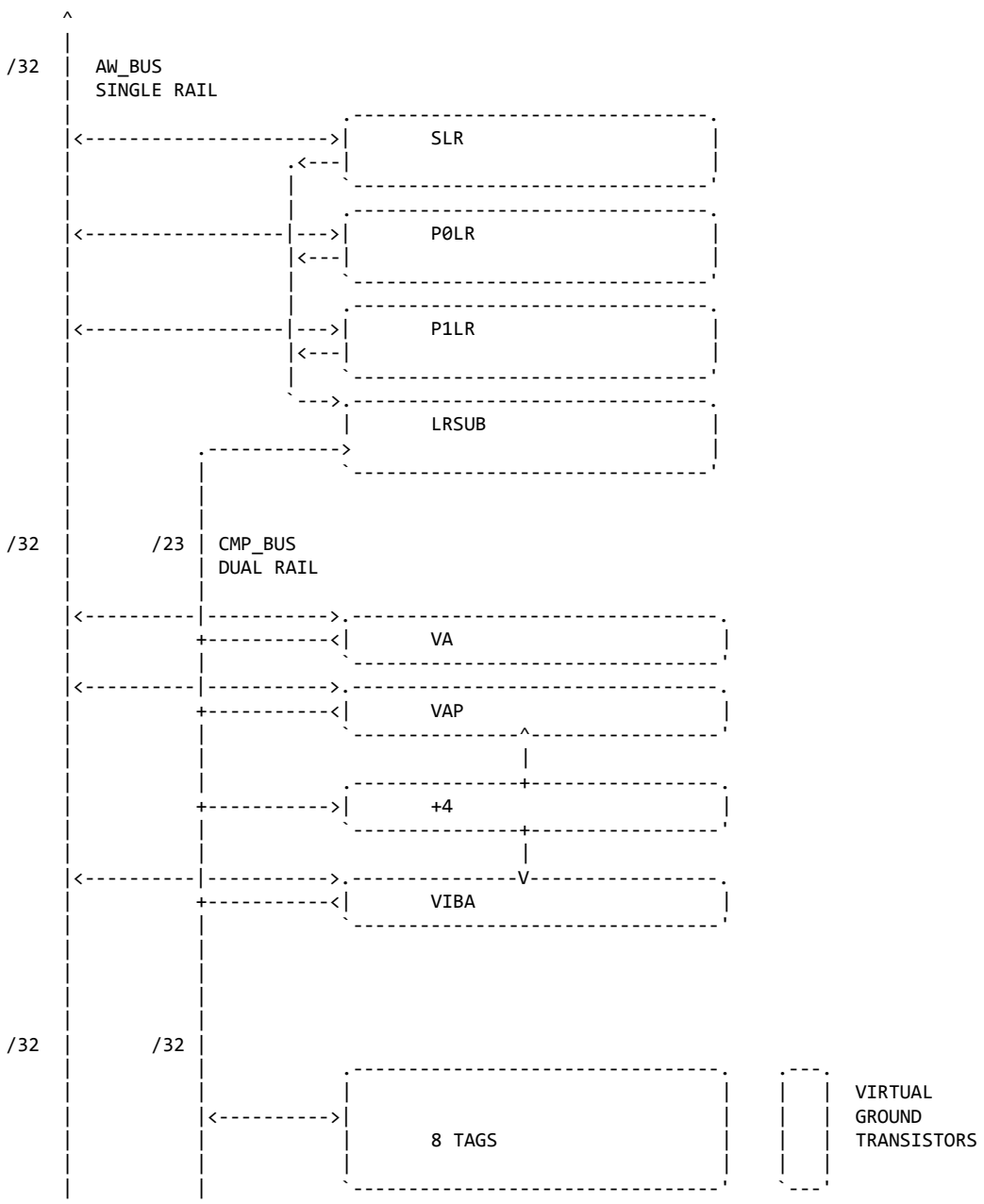
This logic examines the memory request function and the data type of the operation to produce a two-bit data size code for memory references. IB Fills also require a size code with their physical address. The data size codes are given in the table below for MREQ microinstructions as a function of the data type and for IB Fills.

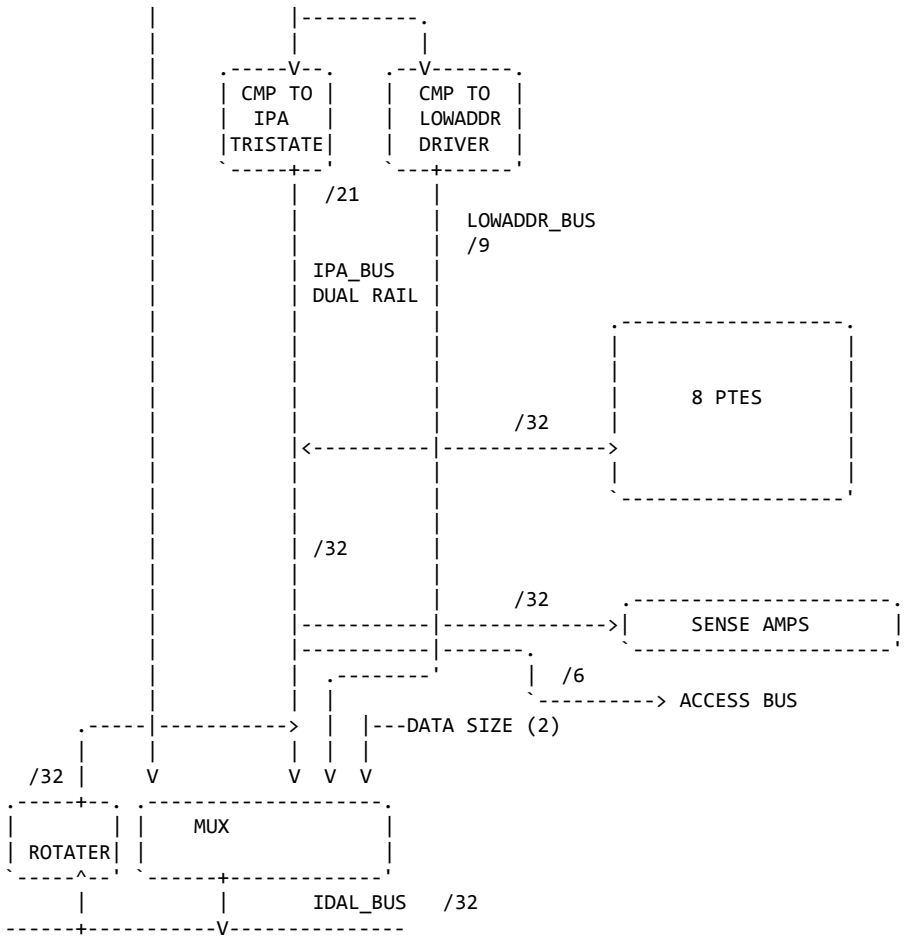
Operation	Data Length	Data Size Code
MREQ	BYTE	00
MREQ	WORD	01
MREQ	LONG	10
MREQ	QUAD	11
IB FILL	LONG	10

The output of this logic network is put on the upper two bits of the DAL during the address portion of the cycle.

4.3 Block Diagram

ADDRESS TRANSLATION UNIT BLOCK DIAGRAM




CMP TO IPA TRISTATE
CMP TO LOWADDR DRIVER
/21
LOWADDR_BUS
/9
IPA_BUS
DUAL RAIL
/32
8 PTES
/32
/32
SENSE AMPS
/6
ACCESS BUS
DATA SIZE (2)
/32
MUX
ROTATER
IDAL_BUS /32

5.0 DAL INTERFACE

The DAL Interface section connects the external DAL (data address lines) to the I Box or E Box sections during DAL reads and to the M Box section during DAL writes. The DAL Interface can be separated into three parts. First the DAL Interface contains all the control circuitry and most of the data path circuitry for the rotating and latching of data going to or coming from the DALs. Secondly, the DAL Interface contains a State Sequencer which goes through a state per microcycle, whether or not the main micromachine is stalled, and generates signals used throughout the chip. The Sequencer generates the PHW_EN signal which prevents loading of registers when the machine is stalled or is taking a microtrap. In many cases the DAL State Sequencer stalls the main micromachine so that the M Box has time to complete a memory transaction. Thirdly, the DAL State Sequencer, the M Box, the bus arbiter circuitry and the external environment feed information to set/reset flip flops and other circuitry which generate the off-chip address and data strobes, the read/write signals, the data transceiver control signal, the control status codes, and the byte masks.

5.1 DAL State Sequencer

The DAL State Sequencer monitors the external DAL control signals and address and instruction information to determine what types of transactions should take place on the DALs and when they should start and finish. The DAL State Sequencer is also responsible for determining if the current microinstruction should be executed and if the next microinstruction should be latched (i.e. should the current microinstruction be re-executed). The chip never stalls the clocks; therefore, if the execution of a microinstruction must be stalled, the DAL Sequencer prevents the loading of the of the next microinstruction by de-asserting CS_UPDATE.

The DAL State Sequencer must also arbitrate between the microcode and the prefetcher. If both the prefetcher and the microcode want to use the DAL in the same cycle, the microcode is given higher priority. If the prefetcher has control of the DAL and is waiting for data, or bus control has been relinquished to another device, then the microcode that is requesting the use of the DAL is stalled. The DAL State Sequencer is responsible for informing the I Box that the data on the DAL is the instruction data that it requested. It does this by asserting IB_DATA_PRS.

Microinstructions that require the IDALs and DALs include MREQ reads and writes and FBOX reads and writes. MXPR read and write microinstructions require the IDALs; therefore they also participate in the arbitration process.

5.1.1 States Of The State Sequencer -

A state diagram for the DAL State Sequencer is included in the diagrams at the end of this section.

The Sequencer has only two states: HOME_LE and XFER_TE. The Sequencer is always in the home state when a memory reference is started. If the memory reference requires more than one microcycle, that is, if the memory reference is an MREQ or IB fill, then the Sequencer makes a transition to XFER_TE.

The suffix LE refers to the Leading Edge of the control signals sent from the CPU chip to the external environment. The Sequencer is always in the HOME_LE state in phase one of a microcycle if the Leading Edges of the control signals are asserted during that microcycle. The suffix TE refers to the Trailing Edge of these control signals. The Sequencer is always in the XFER_TE state in phase one of a microcycle if the Trailing Edges of the control signals are asserted during that microcycle.

HOME_LE is the idle state. The Sequencer remains here if no memory transaction is taking place. The Sequencer remains in the home state if an IB fill or microcode memory request is made, but the bus is unavailable because a DMA request has been granted. The Sequencer remains in the home state if memory management is enabled and an address translation is unsuccessful. The Sequencer also remains in the home state during an FPU XFER or MXPR microinstruction. The Sequencer exits the home state if an IB_REQ or non-probe MREQ is made, the bus is available, and the address is physical or the translation is successful.

When the Sequencer leaves HOME_LE it goes to XFER_TE. It remains in this state until RDY_NO_ERR or QUAL_BUS_ERR or RESET are asserted, at which time it returns to HOME_LE.

Sequencer input IB_CYC allows differentiation between I stream and D stream memory accesses.

5.1.2 DAL State Sequencer Outputs -

The DAL State Sequencer has six outputs: XFER_TE, CS_UPDATE, PHW_EN, E_FIRST_OK, IB_DATA_PRS, and IB_IN_PROG. XFER_TE and E_FIRST_OK are fed back to the Sequencer as inputs.

5.1.2.1 CS_UPDATE And PHW_EN -

ENOP is an intermediate output of the DAL State Sequencer which is asserted when the chip stalls or a microtrap takes place. CS_UPDATE and PHW_EN are asserted if and only if ENOP is de-asserted. The microinstruction latch is loaded if and only if CS_UPDATE is asserted. PHW is asserted if and only if PHW_EN is asserted.

The following is a list of cases when CS_UPDATE and PHW_EN are not asserted:

1. During the first microcycle of a MREQ READ with a successful address translation or a MREQ READ that uses a physical address.

2. During all microcycles except the first microcycle of a MREQ READ when RDY_NO_ERR or QUAL_BUS_ERR or Reset is not asserted or when QUAL_BUS_ERR is asserted for the first time.

3. During all microcycles of the first memory reference of an unaligned reference.

4. Whenever there is a MREQ, or FPU XFER, or MXPR and the bus is unavailable because a DMA request has been granted or there is an IB fill in progress.

5. Whenever VIBA is loaded while an IB fill is in progress and RDY_NO_ERR or QUAL_BUS_ERR or Reset is not asserted.

6. Whenever a microtrap takes place.

The timing portion of this section has detailed information on the assertion of ENOP during aligned and unaligned reads and writes.

5.1.2.2 I Stream -

The DAL State Sequencer outputs two signals which provide information about an I stream reference.

IB_IN_PROG indicates to the M Box that an IB fill read bus cycle is executing.

IB_DAT_PRS indicates to the I Box and the M Box that the data on the DAL is valid instruction data so the prefetcher may be loaded.

5.1.2.3 Unaligned References -

The DAL State Sequencer outputs one signal which gives information about unaligned references. The signal E_FIRST_OK is asserted when the first transaction of an unaligned transaction pair has completed successfully. It is de-asserted when the second bus cycle concludes. If a DMA grant is issued between the two bus cycles of an unaligned reference, then E_FIRST_OK is asserted throughout the grant.

5.2 BUSARB Circuitry

The BUS ARBiter Circuitry (BUSARB) receives inputs from the DAL State Sequencer,the M Box, and the DMA control logic and outputs signals which begin memory, IB fill, FPU XFER, and MXPR cycles.

BUSARB inputs requests for bus use. When the DAL State Sequencer is in the home state and the DALs are free, BUSARB picks the highest priority request, if any, for use of the bus and starts the appropiate bus activity by asserting IB_CYC, DAT_CYC, FPU_CYC or MXPR_CYC. MREQ, FPU XFER, and MXPR microinstructions have higher priority than IB fills.

If the DAL State Sequencer is in the home state, IDMGA_BUF is not asserted, there is no MREQ, FPU XFER, or MXPR request, and MMGT_TD is not asserted, then the BUSARB logic starts an IB fill cycle. The Qualifier Logic, which is described in the next section, will kill the cycle before AS L is asserted, if the conditions described in the Qualifier Logic Section are not met.

If F_XFER_WR or MREQ_WR are asserted and IDMGA_BUF is not asserted then WR_CYC is asserted.

The outputs of the BUSARB logic are latched during ph12 if the DAL State Sequencer is in the home state. The outputs of the BUSARB latches are de-asserted when IRESET is asserted.

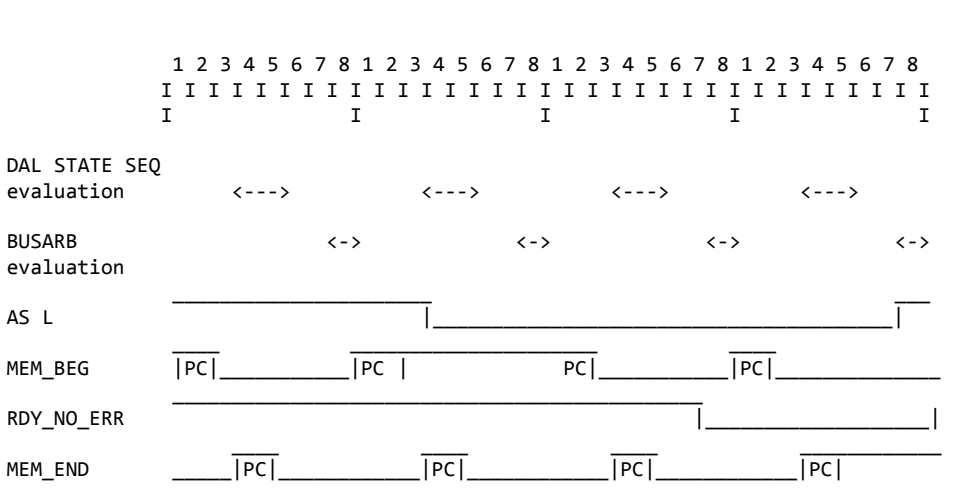
MEM_BEG is asserted if IB_CYC is asserted by BUSARB, MMGT_ERR is not asserted by the M Box, WILL_LOAD_VIBA is not asserted by the M Box, and IB_REQ is asserted by the Ibox. MEM_BEG is asserted if DAT_CYC is asserted by BUSARB and MMGT_ERR is not asserted by the M Box.

When MEM_BEG is asserted, AS L, DS L, DBE L are asserted during the appropiate phases.

MEM_END is asserted when RDY_NO_ERR or QUAL_BUS_ERR are asserted.

MEM_BEG and MEM_END are never asserted during the same microcycle.

The next chart shows timing for the DAL State Sequencer, BUSARB, MEM_BEG and MEM_END during an MREQ. AS L is shown so that it is apparent when the memory access takes place. PC is an abbreviation for pre-charge.



5.3 Off-Chip Control

The DAL Interface generates the external bus interface signals specified in the MicroVAX CPU Chip Specification. The output signals fall into the following categories:

Output Signal Categories	Signals in this Category
data, address, and status strobes	AS L, EPS L, DS L
read/write indicator	WR L
byte indicators	BM<3:0>
type of transaction	CS<2:0>
transceiver control	DBE L
bus ownership	DMG L

The DAL Interface inputs three external bus interface signals.

Input Signal Categories	Signals in this Category
transfer completion indicator	RDY L
memory error	ERR L
bus ownership request	DMR L

Input/Output Signal Category

Output Meaning	Input Meaning	Signals in this Category
type of transaction	FPU is finished	CS<2>

5.3.1 Byte Mask Logic -

The byte mask logic is responsible for generating the four byte mask bits presented on the mask pins. The byte mask only has significance during MREQS and IB fills. The byte mask signals are asserted low. A new byte mask is loaded during ph2 if the DAL State Sequencer is in the home state.

The byte mask depends upon the data length and the alignment (from VA<1:0> or VA'<1:0>). Unaligned references may require two cycles to transfer the data, and the byte masks will be different for the first and second cycle. BM<3:0> are pulled up before a DMA grant is given and are put in the high impedence state during the grant. BM<3:0> are all asserted during an IB fill, MREQ PTE, or MREQ INT VEC.

When the test pin signal is asserted, BM<3:0> are put in the high impedance state and serve as inputs for test microaddress bits.

5.3.2 Control Status Signals -

The control status signals specify what type of bus cycle, if any, is taking place. These signals are valid when AS L or EPS L is asserted by the MicroVAX CPU chip. AS L and EPS L are never asserted simultaneously.

MicroVAX pulls CS<2> high before it asserts EPS L, and CS<2> is tri-stated while EPS L is asserted. CS<2> is a bi-directional signal which is latched by MicroVAX with a transparent latch clocked by not ph67. If MicroVAX is executing a FPU XFER response enable microinstruction, and the sampled CS<2> is asserted low, then EP_DONE L is asserted during ph67. EP_DONE L is de-asserted during ph67 of any microinstruction other than a response enable microinstruction. EP_DONE L is mapped onto the utest bus by the M Box when the BCS field equals NOT_FPU_SIGNAL.

When REDUCER_OUT is asserted, CS<2:0> serve as outputs for the reducers. The assignments are shown in the DAL Interface test section.

The MicroVAX CPU chip pulls the control status lines up before they are tri-stated during DMA grants. The lines are also pulled up when RESET is asserted.

The CS codes are specified in the following chart.

CS<2:0>	State of WR L	Meaning when qualified by MicroVAX assertion of AS L
111	H	read D stream
	L	write D stream
110	H	read D stream (modify)
101	H	read lock
	L	write unlock
100	H	read I stream
011	H	interrupt acknowledge

CS<1:0>	State of WR L	Meaning when qualified by assertion of EPS L
11	H	EP response enable
10	L	EP command (alternate)
01	H	EP transfer to MicroVAX
	L	EP transfer from MicroVAX
00	L	EP command

5.3.3 Asynchronous Inputs -

RDY L, ERR L, and DMR L are asynchronous inputs. They are all sampled by transparent latches clocked by ph81 and synchronized during ph234. The latched RDY and ERR inputs are forced to the unasserted state if HOME_LE is asserted. If HOME_LE is not asserted, RDY L is asserted, and ERR L is not asserted, then RDY_NO_ERR is asserted. RDY_NO_ERR contributes to the generation of MEM_END.

After ERR L is synchronized, it feeds a latch clocked by ph45, and this latch in turn feeds a latch clocked by ph81. The output of this final latch is the signal QUAL_BUS_ERR. QUAL_BUS_ERR contributes to the generation of MEM_END, is an input to the DAL State Sequencer, and is an output that goes to the M Box.

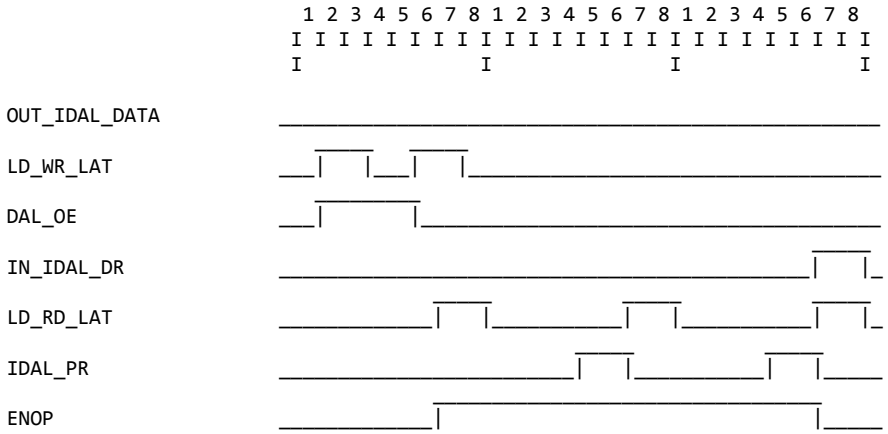
After DMR L is synchronized, it feeds a latch clocked by ph56. If the output of this latch is asserted, and XFER_LE, BUS_LOCK, and RESET are not asserted, then IDMGA_BUF is asserted. This causes a DMA grant in the next microcycle. IDMGA_BUF forces CS<2:0> to 111 and BM<3:0> to 1111 before DMG L is asserted. DMG_START is asserted at the beginning of a DMA grant and DMG_END is asserted at the termination of the grant. DMG_START results in the assertion of DMG L and the tri-stating of AS L, DS L, DBE L, WR L, CS<2:0>, and BM<3:0>. DMG_END results in the de-assertion of DMG L, AS L, DBE L, and DS L and the enabling of WR L, CS<2:0>, and BM<3:0>.

5.3.4 Timing Of Internal Signals -

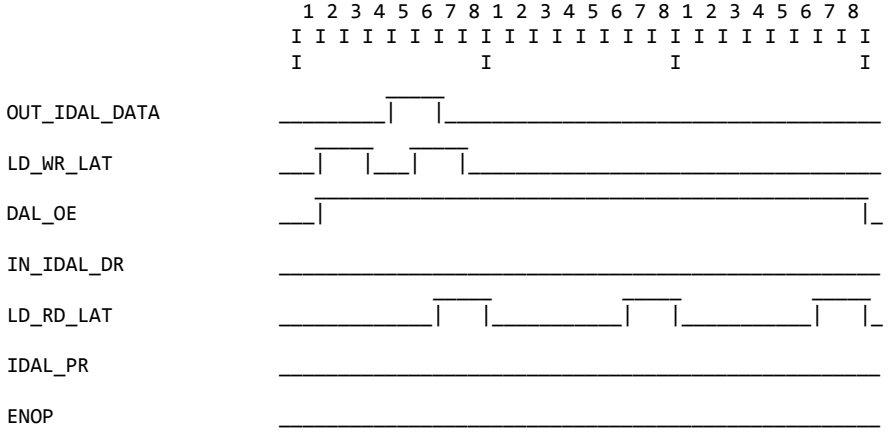
5.3.4.1 CPU Read And Write Cycles -

Each timing diagram has three microcycles. The first microcycle shows the beginning of the bus cycle, the second microcycle shows what happens if the RDY L or ERR L is not asserted, and the third microcycle shows what happens if RDY L is asserted and the bus cycle concludes. If ERR L were asserted, then there would be one more middle microcyle before the bus cycle concludes. All the signals are shown with high assertions. The direction of assertion depends on the implementation.

5.3.4.1.1 Aligned Read -



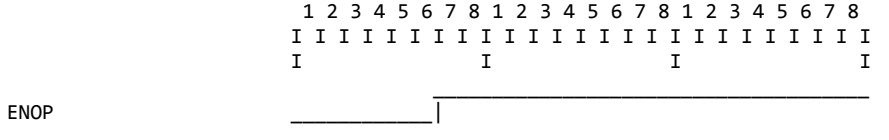
5.3.4.1.2 Aligned Write -



5.3.4.1.3 Unaligned Read -

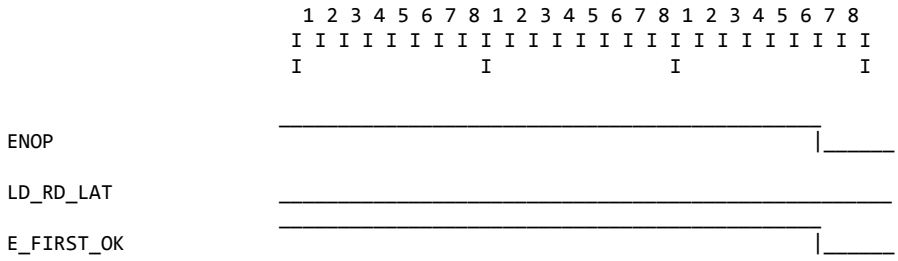
5.3.4.1.3.1 First Bus Cycle -

All of the signals shown in these diagrams except ENOP have the same timing for an aligned read and the first bus cycle of an unaligned read. The signal E_FIRST_OK, which informs the M Box that the first bus cycle of an unaligned reference has completed successfully, is included in the timing diagrams for unaligned references.



5.3.4.1.3.2 Second Bus Cycle -

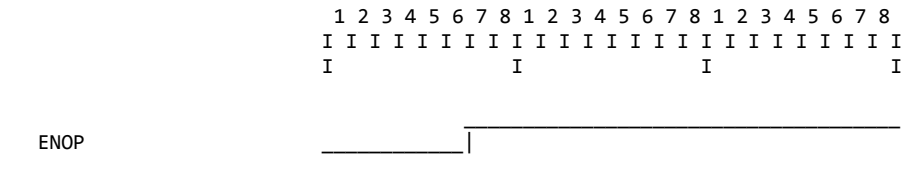
ENOP and LD_RD_LAT have different timing for an aligned read and for the second bus cycle of an unaligned read.



5.3.4.1.4 Unaligned Write -

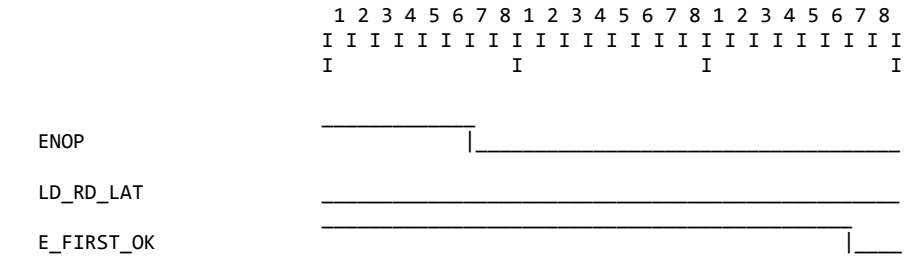
5.3.4.1.4.1 First Bus Cycle -

All of the signals shown in these diagrams except for ENOP have the same timing for an aligned write and the first bus cycle of an unaligned write.



5.3.4.1.4.2 Second Bus Cycle -

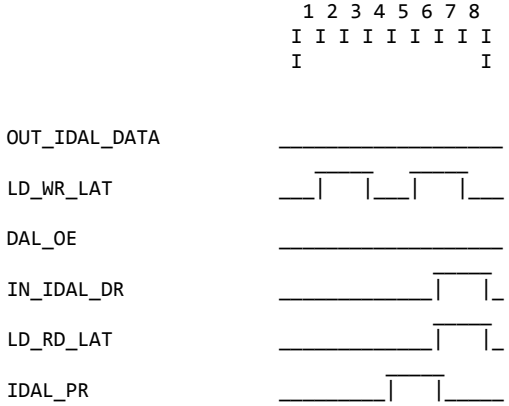
All of the signals shown in these diagrams except for ENOP and LD_RD_LAT have the same timing for an aligned write and the second bus cycle of an unaligned write.



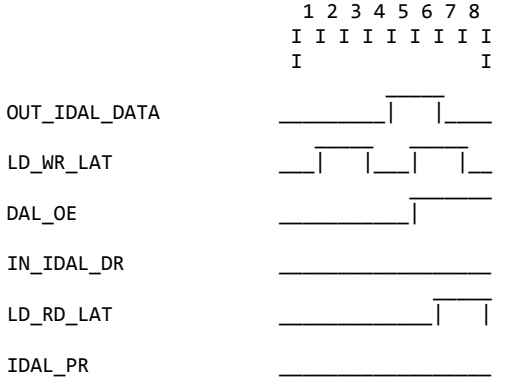
5.3.4.2 FPU XFER Chip Signals -

If the DALs are not busy then a FPU XFER always completes in 1 microcycle.

5.3.4.2.1 FPU XFER Read -

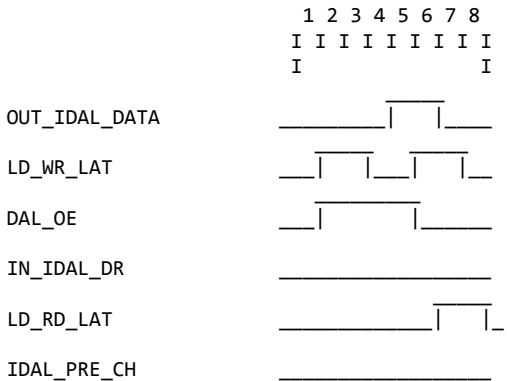


5.3.4.2.2 FPU XFER Write -



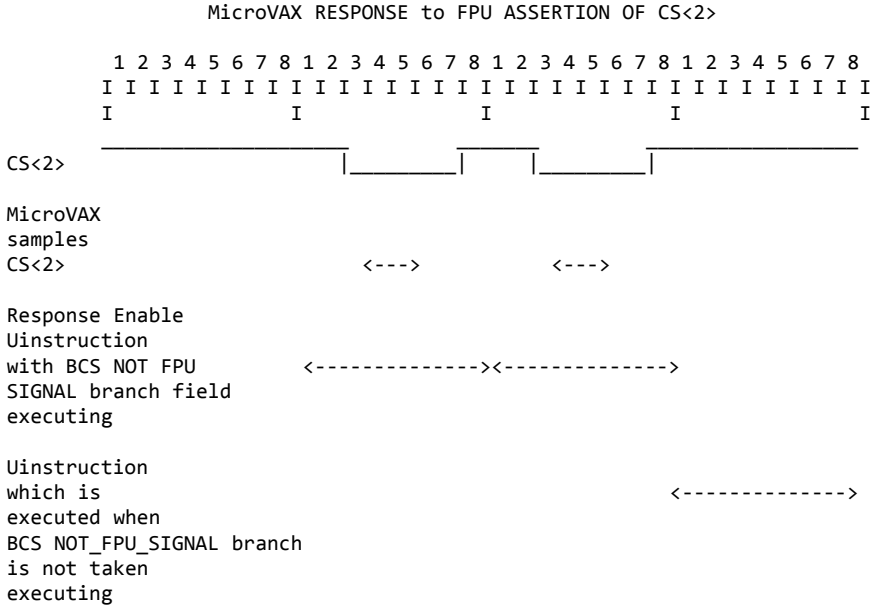
5.3.4.3 MXPR Chip Signals -

OUT_IDAL_DATA is asserted only if MXPR_DIS_DRIVE is not asserted.



5.3.4.4 MicroVAX Response To FPU Assertion Of CS<2> L -

The next diagram shows the timing when the MicroVAX is waiting for the FPU to finish an operation. MicroVAX checks for FPU operation completion by executing an FPU XFER response enable microinstruction with BCS field equal to NOT FPU SIGNAL.



5.4 On-Chip Control

The DAL Interface generates signals for control of the IDAL drivers, write latch, DAL drivers, rotators, D_BUS mux, zero extender, and the Byte Mask. Control circuitry for rotation, the D_BUS multiplexor, the zero extender, and the Byte Mask is physically located in the Memrory Interface Control section.

5.4.1 IDAL And DAL Pad Control -

The OUT IDAL driver drives from the M Box to the IDALs when it is enabled. During ph12, the OUT IDAL Driver drives from IPA<31:9> concatenated with LOWADDR_BUS<8:0> to IDAL<31:0> unless XFER_TE or IRESET is asserted. If an MREQ write, FPU XFER write or command, MXPR read or write with MXPR_DIS_DRIVE de-asserted is taking place, then the OUT IDAL driver drives from MTB_AW_BUS<31:0> to IDAL<31:0> during ph56 and the IDALs are not driven during ph78. IDAL capacitance maintains the state of the IDALs through the end of ph8. If an MREQ read, FPU XFER read or response enable, or IB fill is taking place, then the IDALs are pre-charged high during ph56, and the pad drivers drive the incoming data onto the IDALs during ph78.

During MXPR microinstructions when MXPR_DIS_DRIVE is not asserted, data is driven from the MTB_AW_BUS to the IDALs during ph56. In some cases this data is handled like read data by the main data path at the top of the chip. However, the IDALs are not pre-charged before this data is driven onto the IDALs as is the case with other read data.

If MXPR_DIS_DRIVE is asserted, then the OUT IDAL drivers are in the high impedance state for ph56.

During MREQ PTE microinstructions, the data on the IDALs is driven onto the IPA bus during ph78. The drivers that accomplish this function are located in the M Box.

FBOX_N is a buffered version of IDAL<5>. FBOX_Z is a buffered version of IDAL<4>. These signals permit the loading of the N and Z condition codes directly from the IDALs. The signals are valid at the beginning of ph8.

> Microcode Restriction:
> The ALU_CC and PSL_CC condition codes may not be used for branching in the microinstruction immediately after the microinstruction that loads them via the FBOX_N and FBOX_Z lines.

The DAL Interface control signals are described in the following list:

1. OUT_IDAL_DATA — Asserted when MTB_AW_BUS is selected by mux and drives IDAL during ph56.
2. OUT_IDAL_ADDR — Asserted when IPA<31:9>, LOWADDR_BUS<8:0> is selected by mux and drives IDAL during ph12.

3. LD_WR_LAT Asserted when DAL buffer write latch is loaded from the IDALs.

4. DAL_OE Asserted when DAL output buffer drives DALs.

5. IN_IDAL_DR Asserted when DAL input buffer drives the IDALs.

6. LD_RD_LAT Asserted when data path read latch which is located at top of data path is loaded from IDALs.

7. IDAL_PR IDAL pre-charge.

The IGEN bus runs through the IGEN section. This is a control bus containing signals specifying what type of bus cycle, if any, is taking place, whether the cycle is beginning or ending, and what state the DAL State Sequencer is in. This information comes from the BUSARB logic, the DMA logic, the M Box, and the DAL State Sequencer. The bus also contains clock signals. Decoders located under and adjacent to this bus generate signals which are used by the IGEN logic.

The IGEN logic consists primarily of decoders under the IGEN bus and associated circuitry which generate control signals for the IDALs and the DS L output.

SIGNALS GENERATED BY IGEN LOGIC

OUT_IDAL_ADDR
OUT_IDAL_DATA
LD_WR_LAT
DAL_OE
IDAL_PR
IDAL_PR_KILL

5.4.2 Data Latches Control -

The Read Data Latch is loaded during ph78 of MREQ Reads if LD_RD_LAT is asserted. The Write Latch is loaded during ph23 and ph67 if the DAL State Sequencer was home during ph1 of the microcycle.

5.4.3 Rotators And Data Multiplexer Control -

During reads, data is received from the DAL via a read rotator, latch, and mux. During writes, data is passed to the DAL via a write rotator and latch. Note that addresses are not rotated. FPU and MXPR data is never rotated and thus is never sourced from the read latch.

Read rotation is 0,1,2, or 3 bytes right and write rotation is 0,1,2, or 3 bytes left. The read mux selects bytes from the current microinstruction's data (ROT) or from the previous bus cycles data (LAT) for unaligned references. The write rotators are set to pass except during the data portion of an MREQ. The rotators' and muxes' actions are summarized in the following table as a function of the low two bits of VA

(or VA').

VA<1:0>	READ BYTE ROTATOR ACTION	READ MUX ACTION			
		B<3>	B<2>	B<1>	B<0>
00	Pass, no rotation	ROT	ROT	ROT	ROT
01	Rotate 1 byte right	ROT	LAT	LAT	LAT
10	Rotate 2 bytes right	ROT	ROT	LAT	LAT
11	Rotate 3 bytes right	ROT	ROT	ROT	LAT

VA<1:0>	WRITE BYTE ROTATOR ACTION
00	Pass, no rotation
01	Rotate 1 byte left
10	Rotate 2 bytes left
11	Rotate 3 bytes left

On an MREQ PTE, no read rotation occurs no matter what VA<1:0> is. This is necessary because the address of the PTE may be generated from an unaligned address. No rotation occurs because MREQ PTE data does not pass through the read rotators. On an MREQ INT VEC, no rotation occurs no matter what VA<1:0> is.

5.4.4 Zero Extender Control -

The zero extender is used to zero out the high order bytes being loaded into chip registers when the data is not a longword. The value of the zero extend mask is a function of data length.

Only MEM REQ read byte and word and MEM INT VEC data is zero extended.

5.5 Test Control

When the REDUCER_OUT input is asserted, CS<2:0> serve as outputs for the reducers. The assignments are show in the next chart. REDUCER_OUT is controlled by a set/reset latch that is reset by IRESET and set when MISC1 = EN_REDUCER_OUT.

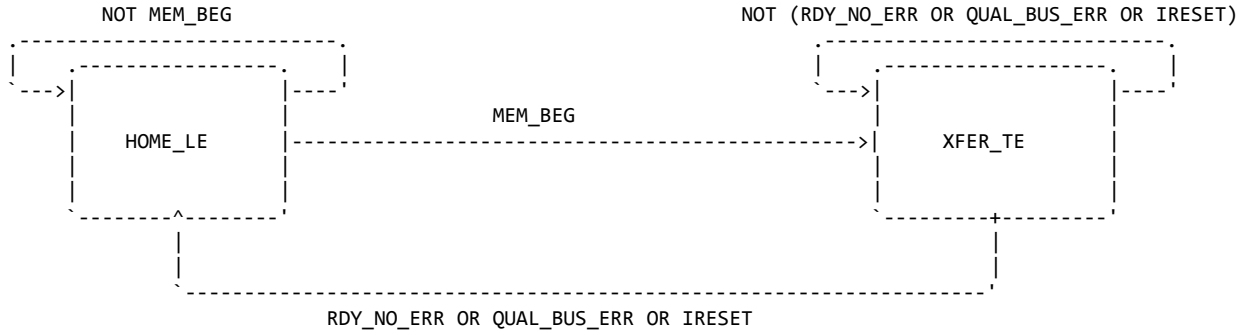
When FORCE_UADDR is asserted, ENOP and LCSLIT are de-asserted. FORCE_UADDR is controlled by a set/reset latch that is reset by IRESET and set when MISC1 = EN_FORCE_UADDR.

CS<2:0> ASSIGNMENTS WHEN TEST PIN IS ASSERTED

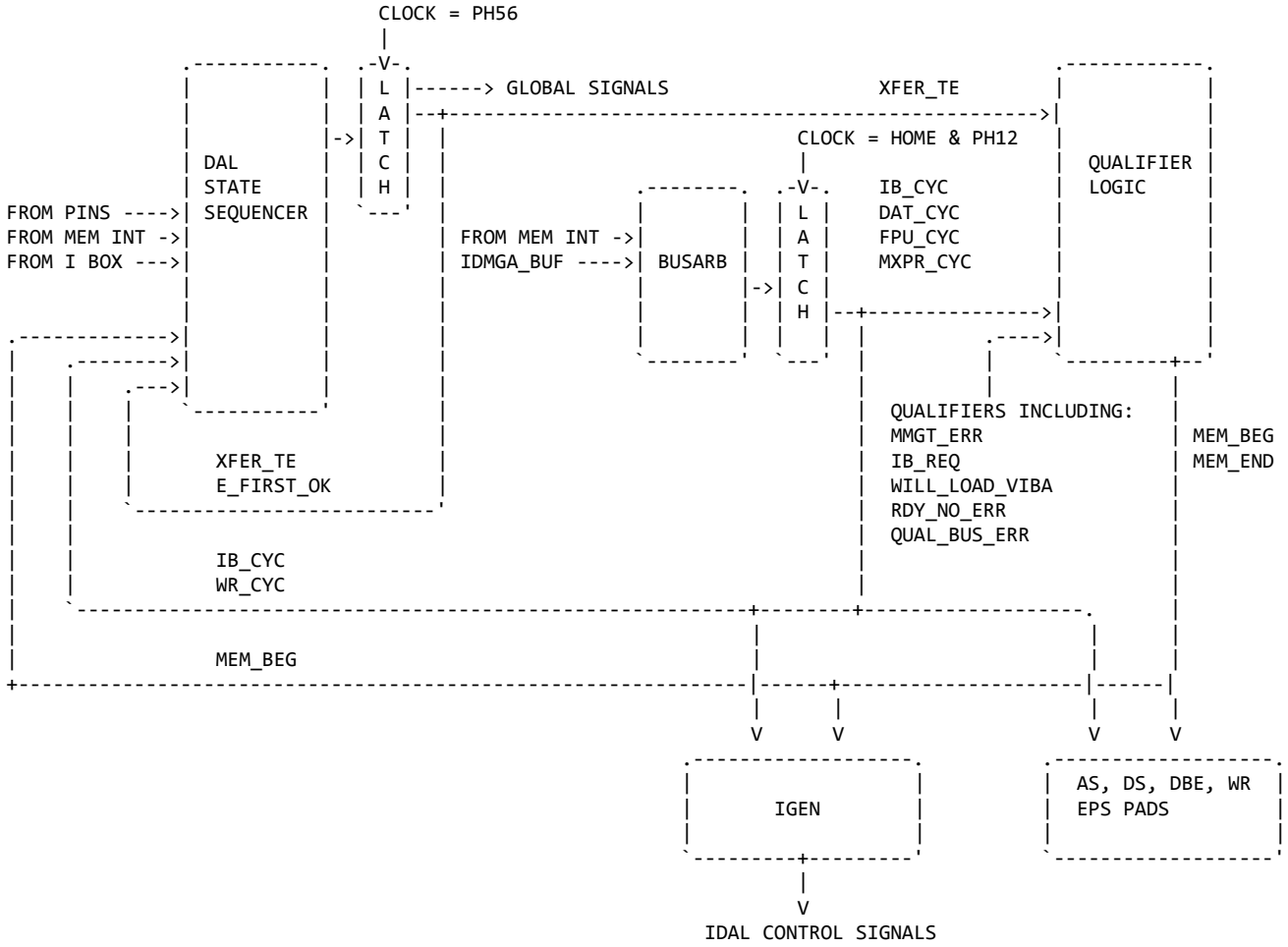
IPLA reducer	CS<2>
Microsequencer reducer	CS<1>
Control Store reducer	CS<0>

When TEST_PIN is asserted, BM<3:0> are tri-stated and serve as inputs for test microaddresses.

STATES OF THE DAL STATE SEQUENCER

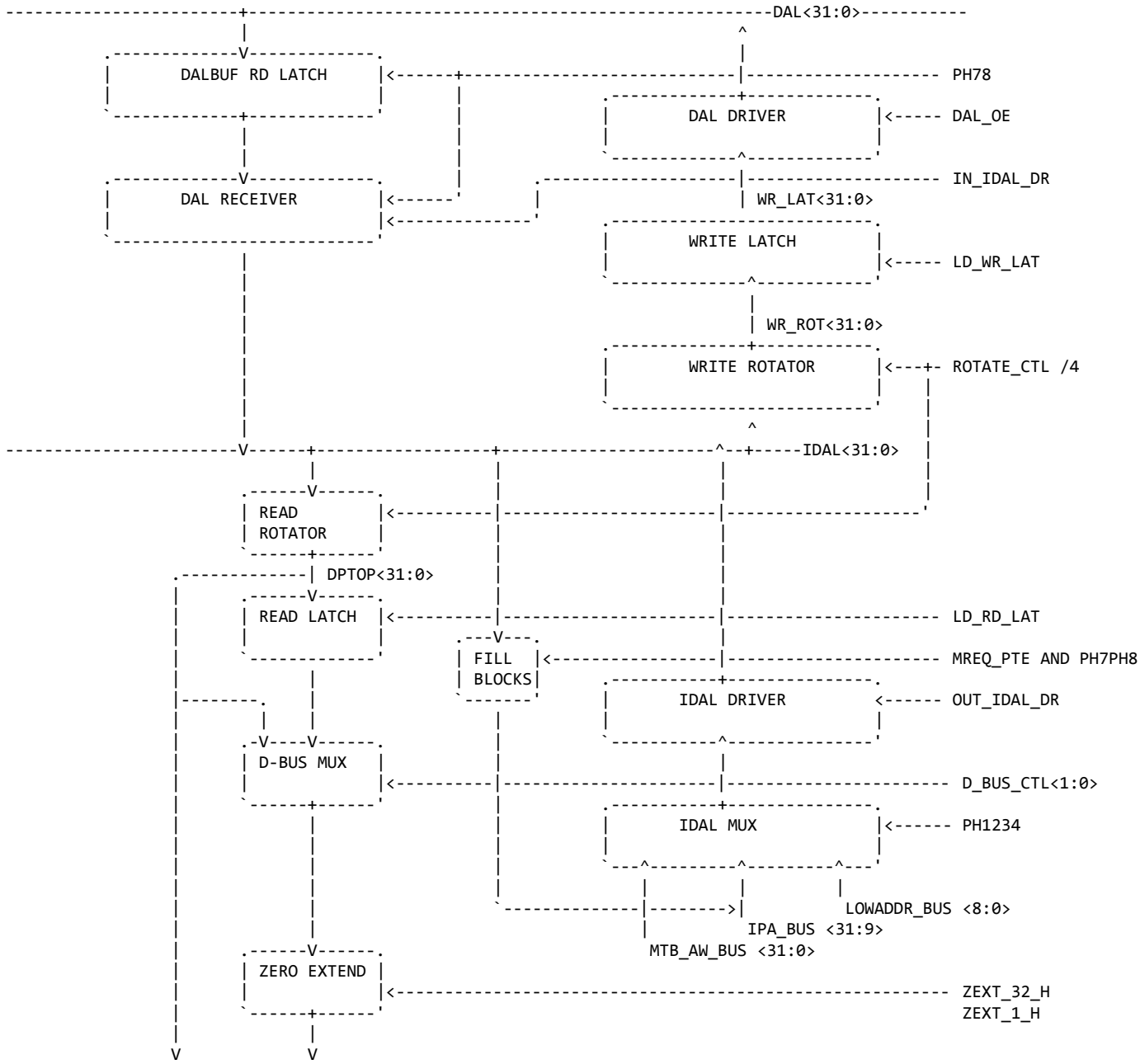


DAL AND IDAL CONTROL LOGIC
BLOCK DIAGRAM



5.6 Block Diagram

DAL INTERFACE BLOCK DIAGRAM



TO IBOX D-BUS<31:0>

6.0 MICROSEQUENCER

The Microsequencer produces the address of the next microinstruction to be executed. The address could be:

- The next sequential address
- The next sequential address plus an offset
- The calculated next address OR'ed with data
- An address specified by the microinstruction
- The top of the microstack
- The top of the microstack plus an offset
- A microtrap routine address
- An address generated by the I Box
- An externally generated test address

This description of the Microsequencer is divided into two parts: an overview describing the busses, major components, and gross timing, and a functional description at the microcode level.

6.1 Overview

Please refer to the block diagram at the end of this section.

6.1.1 Busses -

6.1.1.1 Microaddress Bus (MAB) <10:0> -

The Microaddress Bus is the Microsequencer's major output bus. Connected to the Control Store's row and column decoders, it contains the address of the next microinstruction to be executed.

Because of the circuit complexities of the Control Store, bits of the MAB are implemented differently. MAB<10:4> are dual rail. The true side is preconditioned low, and complement side high, starting in phase 8. Starting in phase 2, MAB<10:4> take on their final values for that cycle. There are both true and complement versions of MAB<3>. One or the other is pulsed high during phases 34; both are pulled high during phases 67; then both are discharged during phases 812. MAB<2:0> are single rail with the same timing as MAB<10:4>.

6.1.1.2 Internal Microaddress Bus (IMAB) <10:0> -

The Internal Microaddress Bus is the Microsequencer's major internal bus. It is a unidirectional bus which receives data each cycle from one of four sources and/or the OR BOX.

The Internal Microaddress Bus is precharged during phases 78, and conditionally discharged during phases 23.

6.1.1.3 Next Address Bus (NAB) <10:1> -

The Next Address Bus is the I Box's major output bus. It contains valid microaddress information during cycles which parse new information from the I stream. The least significant bit of all microaddresses from the I Box is always zero and hence not routed.

The Next Address Bus is unconditionally precharged during phases 78. It is conditionally discharged starting as early as phase 1.

6.1.1.4 Jump Next Address Bus (JMPNA) <10:0> -

Jump Next Address is a secondary internal bus. This bus serves to translate data which is routed horizontally into the vertical pitch of the microsequencer datapath. It receives an externally generated test address if the test control pin is asserted; otherwise, it receives bits<10:0> of the Internal Microinstruction Bus (IMIB).

Jump Next Address is precharged during phases 56. When the test control pin is not asserted, JMPNA receives flow through IMIB data during phase 7. When test is asserted, JMPNA receives the test address on phase 1.

6.1.1.5 Microtest Bus (uTest) <3:0> -

Microtest is a global control bus for the Microsequencer. In conjunction with the Internal Microinstruction Bus, Microtest controls most major functions of the Microsequencer.

The Microsequencer precharges Microtest during phases 78. Sources to Microtest start discharging it in phase 1.

6.1.1.6 Internal Microinstruction Bus (IMIB) <38:0> -

Buffered AND decodes of the Internal Microinstruction Bus provide the usual form of microcode control to the Microsequencer. These decodes are physically located in the I Box.

Both true and complement versions of the IMIB are pre-discharged during phases 56. They become valid during phase 7.

6.1.2 Logic Components -

6.1.2.1 Microsubroutine Stack (uStack) [0:7] <10:0> -

The Microsubroutine Stack is a true LIFO stack of eight entries used for storing return microaddresses. When pushed, all entries move down one, and the top of the Microsubroutine Stack is loaded from the Microprogram Counter. When popped, all entries move up one, and the top of the stack is lost. The contents of the top entry is always presented to the Adder. It may also be enabled onto the IMAB.

Pushes and pops occur during PHI_WRITE, the global register write clock, such that the stack state will not change during stretched cycles. The control strobes PUSH_UST and POP_UST perform these operations. UST_SEL controls the gating onto the IMAB.

6.1.2.2 Adder -

The Adder adds sign-extended JMPNA<6:0> to either the Microsubroutine Stack or the Microprogram Counter. The result of the addition can be enabled onto the IMAB by the strobe ADDER_SEL.

The add operation occurs during phases 81 under control of the strobes ADD_UST and ADD_UPC. This allows one phase for the newly updated register information to propagate to the Adder.

6.1.2.3 Microprogram Counter (uPC) <10:0> -

The Microprogram Counter is an eleven bit register/incrementer combination. Its contents can be enabled onto the IMAB by the strobe UPC_SEL. During every cycle, the incrementer adds one to the microaddress in the MAB Latch. The result is then loaded into the register with PHI_WRITE timing. Note that the Microprogram Counter must incorporate a slave register so that the master can be loaded at the same time that the slave's contents is being pushed onto the Microsubroutine Stack.

6.1.2.4 Jump Control -

Jump Control controls the source of the Jump Next Address Bus and its enabling onto the IMAB by the strobe IMIB_SEL.

6.1.2.5 OR Box -

The OR Box consists of only five transistors, each of which can be selected separately to discharge a corresponding bit of the IMAB. Because the IMAB is asserted active low, this discharging represents an OR function.

6.1.2.6 MAB Mux -

The MAB Mux is a simple pass transistor network which can select either the IMAB or the NAB as the source of the MAB Latch and hence the MAB. Furthermore, different combinations of bits can be selected. In particular, bits<10:5,0> can be selected as a group from either the NAB or IMAB. Likewise, bits<4:3> can be selected as a group and bits<2:1> can be selected as a group.

6.1.2.7 MAB Latch -

The MAB Latch is opened every cycle during phase 8 while the busses are still being precharged. It is closed at the end of phase 3.

6.1.2.8 MAB Reducer -

The MAB Reducer uses an XOR and shift combination to checksum the MAB during test mode. The shift out is routed to the DAL Interface and conditionally driven on a Control Status output pin.

6.1.2.9 Control Logic -

The Control Logic, although definitely random, can be thought of as containing four major blocks: strobe logic, mux control logic, microtrap logic, and test logic.

The strobe logic combines the microinstruction decodes, Microtest bus value, and clock edges to generate the strobes necessary for controlling the datapath.

The mux control logic combines two IMIB decodes, uTest<3>, and IRESET to control the MAB Mux pass transistor network. When IRESET is asserted, the

output of the mux is forced to zero.

The microtrap logic combines the four microtrap request inputs into a priority encoding which is used as a microcode routine base address. It also generates TRAP_PND, the inclusive-OR of the four requests. TRAP_PND, routed to the DAL Interface, is precharged phases 23 and conditionally discharged phase 4. It is valid at the beginning of phase 5.

The test logic disables all functions which are microinstruction dependent and guarantees that the test address will be written into the MAB Latch.

6.1.3 Timing Summary -

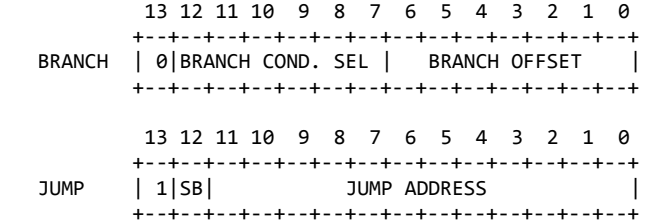
Phase 1	Microtest becomes valid Gate test address to JMPNA
Phase 2	Enable source to IMAB MAB becomes valid Precharge TRAP_PND
Phase 3	Close MAB Latch Latch decodes
Phase 4	Discharge TRAP_PND Latch microtrap requests
Phase 5	TRAP_PND valid Precharge JMPNA Precharge decodes
Phase 6	IRESET becomes valid
Phase_Write	Update uPC, uStack
Phase 7	Evaluate decodes JMPNA receives IMIB Precharge IMAB Precharge uTEST
Phase 8	Start addition

6.2 Functional Description

This section describes the microinstruction control of the Microsequencer, the microtrap mechanism, and finally, test mode.

6.2.1 Microsequencer Control Interpretation -

For all microinstructions, the lower 14 bits are used to control the Microsequencer. Bit<13> selects between one of two formats:



6.2.1.1 Branch Format -

The branch format is used when microinstruction bit<13> = 0. It allows relative branching on condition, return from subroutine, and case branching.

6.2.1.1.1 Branch Offset (BO) <6:0> -

The Branch Offset field contains a signed offset that is presented as one operand to the Adder. The second operand is either the Microprogram Counter or the Microsubroutine Stack.

6.2.1.1.2 Branch Condition Select (BCS) <12:7> -

To other areas of the chip, the BCS field addresses information to be put on the Microtest Bus. To the Microsequencer, the Branch Condition Select field selects one of the following four classes of branch control:

BCS Value	Mnemonic	Name
00 - 2F, except 21	RB	Relative Branch
21	MT	Microtrap
30 - 3B	CS	Case
3C - 3F	SB	Stack Branch

6.2.1.1.2.1 Relative Branch -

When the BCS field selects a relative branch, Microtest defines the source of the MAB as follows:

uTest Bits 3	2	1	0	SOURCE =
H	H	H	L	Adder
H	H	L	H	uStack
H	L	H	H	uPC
L	H	H	H	NAB
All else				UNPREDICTABLE

During these branches, the Microprogram Counter is selected as the second operand to the Adder. Thus, if Microtest selects the Adder, the resulting microaddress is relative to the current microaddress.

6.2.1.1.2.2 Microtrap -

The BCS field of 21 is used to select a code representing the type of Memory Management microtrap from the M Box. See the section entitled Microtrap Mechanism for details on how this is used.

6.2.1.1.2.3 Case -

When the BCS field selects a case, the Microprogram Counter is selected as the second operand to the Adder, and the Adder is selected as the source of the IMAB. In addition, the incoming data on Microtest is OR'ed into the result by the OR Box. This allows the microcode to specify the base address of a table, and the data on which to index into it.

6.2.1.1.2.4 Stack Branch -

These branches are identical to the relative branches except that the Microsubroutine Stack is selected as the second operand to the Adder. Thus, if Microtest selects the Adder, the resulting microaddress is relative to the most active microsubroutine return address.

6.2.1.1.3 Branch Condition Chart Summary -

Although the specific type of branch or case data which is selected by the BCS field is of no importance to the Microsequencer, for completeness, the following chart specifies the different possible branch conditions in detail. For the two types of branches, it specifies which Microtest bits could be asserted. For cases and microtraps, it specifies the data which is put on the bus.

BCS	CONDITION	uTest bits 3	2	1	0	uTest Driven by	TYPE
0	KERNEL MODE		X		X	M Box	RB
1	NO FPD INTERRUPTS PENDING		X		X	I Box	RB
2							
3							
4							
5	IF AVMF GOTO IF BWL NSD/RET	X	X	X	X	I Box	RB
6							
7	NOT ALU Z		X		X	E Box	RB
8							
9	NOT FPU		X		X	I Box	RB
A	ALU N		X		X	E Box	RB
B	ALU Z		X		X	E Box	RB
C							
D							
E	ALU V		X		X	E Box	RB
F	ALU C		X		X	E Box	RB
10							
11							
12							
13							
14							

15						
16						
17						
18						
19	VA MEM REF NOT OK		X	X	M Box	RB
1A	RMODE		X	X	I Box	RB
1B						
1C						
1D						
1E						
1F	NOT INT STACK		X	X	M Box	RB
20	NOT FPU SIGNAL		X	X	M Box	RB
21	MICROTRAP ADDRESS REQUEST	MA<4>	MA<3>	MA<2>	MA<1>	M Box	MT
22						
23						
24	IF A GOTO IF V OR BWL NSD/RET	X	X	X	X	I Box	RB
25	IF AVM GOTO IF BWL NSD/RET	X	X	X	X	I Box	RB
26	IF BWL IID ELSE GOTO	X			X	I Box	RB
27						
28						
29	IF R OR M GOTO		X		X	I Box	RB
2A	IF BCOND LOAD VIBA & PC (AND BR) ELSE IID	X			X	E Box	RB
2B	IID NO EXCEPTIONS	X				I Box	RB
2C	NSD/RET	X		X		I Box	RB
2D	IF AV NSD/RET ELSE GOTO	X		X	X	I Box	RB
2E	IID	X				I Box	RB
2F	LOAD VIBA & PC AND BRANCH				X		RB

30	ALU NZVC	ALU.N	ALU.Z	ALU.V	ALU.C	E Box	CS
31	SC 3-0	SC<3>	SC<2>	SC<1>	SC<0>	E Box	CS
32							
33	SC 7-4	SC<7>	SC<6>	SC<5>	SC<4>	E Box	CS
34	STATE 3-0	ST<3>	ST<2>	ST<1>	ST<0>	E Box	CS
35	MMGT STATUS	MST<3>	MST<2>	MST<1>	MST<0>	M Box	CS
36	OPCODE 3-0	OPC<3>	OPC<2>	OPC<1>	OPC<0>	I Box	CS
37	DL.MBOX STATUS	DL<1>	DL<0>	MB<1>	MB<0>	IBox,MBox	CS
38	STATE 7-4	ST<7>	ST<6>	ST<5>	ST<4>	E Box	CS
39							
3A							
3B							
3C	IF MEM REF OK RETURN+BO		X		X	M Box	SB
3D	RETURN+BO				X	I Box	SB
3E							
3F							

6.2.1.2 Jump Format -

The jump format is used when microinstruction bit<13> = 1. It allows unconditional jumps and subroutine calls to anywhere in the Control Store.

6.2.1.2.1 Subroutine Control Bit (SB) <12> -

The Subroutine Control Bit controls pushing the Microsubroutine Stack. If it is set, the uPC is pushed onto the Microsubroutine Stack and a jump to subroutine is done. If not, the stack is unaffected.

6.2.1.2.2 Jump Address Field <10:0> -

The Jump Address field specifies the eleven microaddress bits directly. This allows freedom to jump anywhere in the Control Store.

6.2.1.3 Miscellaneous Field (MISC) <22:18> -

The Miscellaneous Field is bits<22:18> of the microinstruction. It controls two additional Microsequencer functions. If the MISC field is 16 or 17 (hex), LOAD_ID_LEN(DL) or LOAD_ID_LONG, then NAB<2:1> is unconditionally muxed into the MAB Latch. If the MISC field is 14 (hex), SPECIFIER_DECODE, then NAB<4:1> is muxed into the MAB Latch. These MISC field values allows the microcode to do inline decodes of the prefetch state and current specifier, respectively.

6.2.2 Microtrap Mechanism -

The microtrap mechanism provides a means of diverting the normal microcode flow in response to a hardware event.

During each microcycle, the Microsequencer examines four microtrap request inputs. If a request is pending, the Microsequencer asserts TRAP_PND indicating that a microtrap is in progress. During the next phase 2, the IMAB is driven through the OR Box with the address of the first microinstruction of the microtrap routine.

The microaddress which is generated is based on a priority encoding of the four microtrap requests. In particular, the priority encoded value is microaddress<4:2>; microaddress<6> is forced to a 1; and, in the event of a Memory Management request, uTest is OR'ed into microaddress<4:1>. The following table summarizes the results:

Priority	Microtrap Request Input	Trap	Priority Encoding	Resulting Microaddress
4	MM_TRAP_REQ	Memory Management	000	040 + uTest<3:0>'0
3	OV_TRAP	Optimized Integer Overflow	100	050
2	ILL_OPC	Illegal Opcode	110	058
1	FPU_ERR	Floating Point Error	111	05C

TRAP_PND causes the DAL Interface to withhold PHW_EN, thereby suppressing PHI_WRITE, such that the microinstruction which was executing at the time of the microtrap is aborted. In addition, TRAP_PND causes the Control Store to disable latching the microinstruction currently being accessed and to force a literal into the IMIB latch. The literal is a SPECIAL microinstruction with all fields of NOP, and a BCS field of 21 (hex). The BCS field signals the M Box to gate a code representing the last type of Memory Management microtrap onto Microtest. During execution of the NOP, the first microinstruction of the microtrap routine is being accessed.

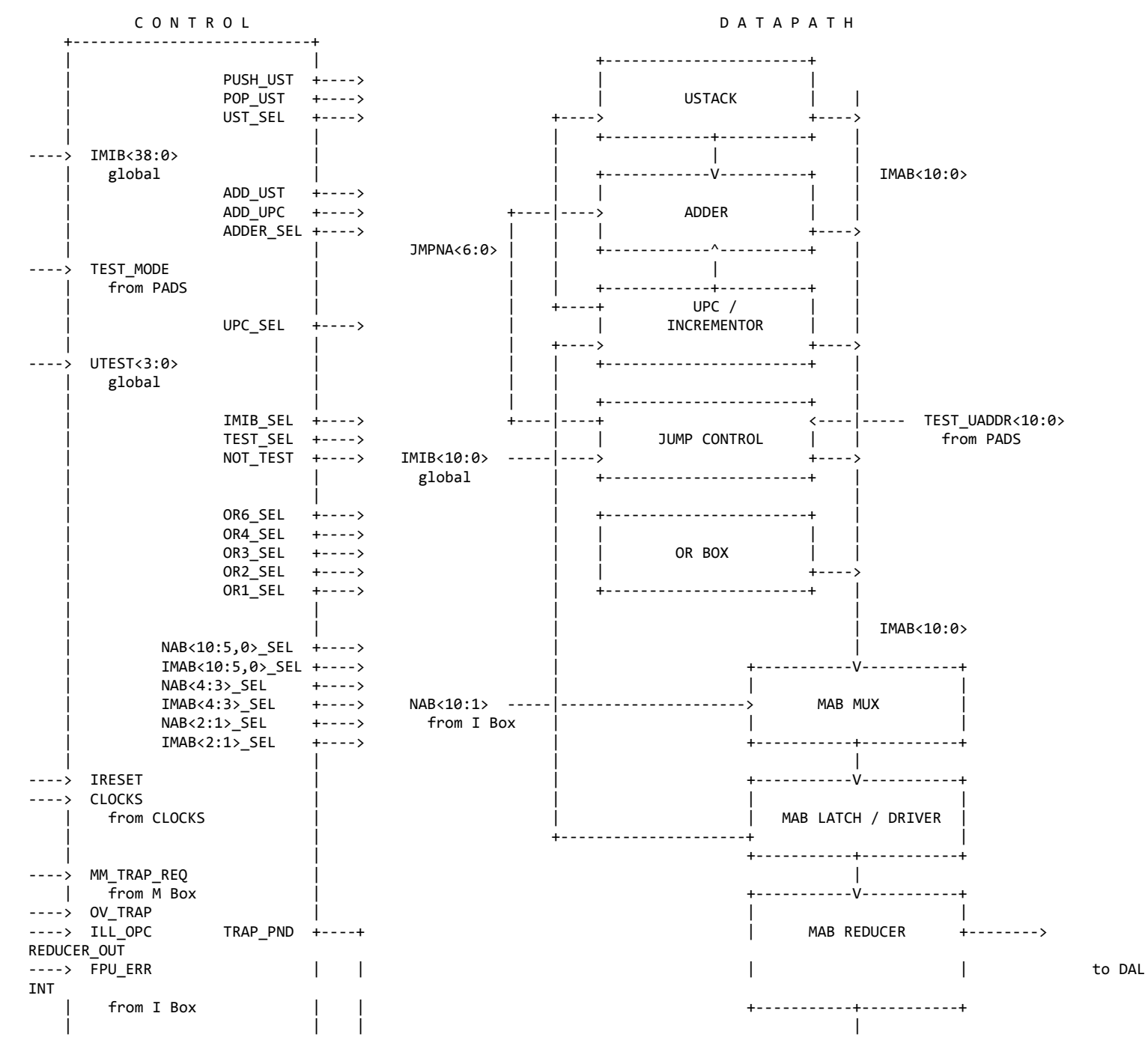
6.2.3 Test Mode -

When the test mode pin is asserted, the Microsequencer accepts an externally supplied value as the next microaddress. In particular, the Byte Mask, Interrupt Request, Power-Fail, Interrupt Timer, and Halt pins become the source to the MAB as follows:

MAB<10:0> <= BM<3:0>'IRQ<3:0>'PWRFL'INTTIM'HALT

Regardless of test mode, the Microsubroutine Stack behaves according to the microinstruction being executed. The exception to this is that if uTest<1> is asserted during Relative Branch microinstructions, the stack does not pop.

6.3 Block Diagram




TRAP_PND
to DAL INT
MAB<10:0>
to CONTROL STORE

7.0 INTERRUPT LOGIC

The Interrupt Logic mediates hardware interrupt requests against the current IPL and generates an interrupt request to the I Box, if necessary. It also identifies the highest priority outstanding interrupt to the microcode.

The Interrupt Logic also contains the logic to support a minimal Interval Clock Control register (ICCS - processor register #24). External logic provides a free running clock to the INTTIM L pin (this clock should be a 100Hz clock for software compatibility with VAX operating systems); the ICCS register enables and disables clock interrupts. The microcode supports the rest of the clock processor registers as read as zero, write is nop for software compatibility with other VAX systems. These include the Next Interval Count (NICR #25), the Interval Count (ICR #26), and the Time of Year (TODR #27).

The Interrupt Logic consists of the interrupt latches and edge detectors, the interrupt priority encoder, and the interrupt IPL and comparator.

NOTE

From the microcode perspective the MXPR's reads and writes to the Interrupt Logic registers look like transactions with the AW_Bus. In reality the Interrupt Logic interfaces with the IDALs which are driven from or to the AW_Bus as needed. The M Box provides a single decode of the two MXPR reads of the Interrupt Logic registers for the DAL to disable the normal IDAL DATA drive for the current cycle. The DAL precharges the IDALs and the Interrupt Logic only provides pulldowns.

7.1 Interrupt Latches

The interrupt latches are double-buffered latches which sample and synchronize the interrupt inputs (HALT L, PWRFL L, INTTIM L, and IRQ<3:0>). The IRQ lines are level sensitive. If asserted and then deasserted before the interrupt is acknowledged the interrupt is ignored. The other three interrupt inputs (HALT L, PWRFL L, and INTTIM L) are pseudo edge sensitive. They are sampled, synchronized, and input to flip-flops. These flip-flops are set when the input is asserted and cleared when IRESET_H is asserted or when the MXPR[INT.ID] register is read and matches one of the request levels corresponding to these flows. The PWRFL interrupt is enabled whenever IPL is less than 30. The HALT interrupt is always enabled but will only be arbitrated for between instructions (HALT will not cause a FPD instruction to packup). The

INTTIM flop is enabled/disabled with MACRO MTPR to ICCS (#24) that sets/clears bit <6> (IE). The remaining bits in the MACRO ICCS register read as zero and are ignored on write. In the micro machine the ICCS bit <6> is read and written by an MXPR[INT.ICCS]. The interrupt enable bit is bit <16> of the INT.ICCS register. The remaining bits are undefined on read and don't care on write. The interval timer is a level 16 interrupt.

7.2 Interrupt Priority Encoder

The latched interrupt inputs are prioritized to generate the highest level outstanding interrupt. That is compared to the current IPL. If it is greater than the IPL, the interrupt identification reads in accordance with the following table. If it is less than or equal to the IPL, the interrupt identification reads as 0. The interrupt identification MXPR[INT.ID] is read onto bits<20:16> of the AW_Bus by an MXPR READ microinstruction.

IPL	HALT	PWRFL	IRQ<3>	INTTIM	IRQ<2>	IRQ<1>	IRQ<0>	int ident (hex)
XXX	L	x	x	x	x	x	x	1F
<1E	H	L	x	x	x	x	x	1E
<17	H	H	L	x	x	x	x	17
<16	H	H	H	L	x	x	x	1C
<16	H	H	H	H	L	x	x	16
<15	H	H	H	H	H	L	x	15
<14	H	H	H	H	H	H	L	14
XXX	H	H	H	H	H	H	H	00

7.3 Interrupt IPL And Comparator

The Interrupt Logic includes a write-only copy of the PSL<IPL> field. This copy is loaded from bits<20:16> of the AW_Bus by an MXPR[PSL.HWRE] microinstruction. The Interrupt Logic also includes an IPL comparator. During every microcycle, the output of the interrupt priority encoder is compared to the stored copy of the IPL. If the output of the priority encoder is greater than the IPL, then signal FPD_IRQ%N%_H is asserted for the I Box. If either FPD_IRQ%N%_H is asserted, or HALT is asserted, then signal IID_IRQ%N%_H is asserted to the I Box.

The decode of the IMIBs for the MXPR read INT.ID and INT.ICCS and the MXPR write PSL.HWRE and INT.ICCS is done in the I Box, and the signals MXPR_READ_INT_ID, MXPR_READ_INT_ICCS, MXPR_WRITE_PSL_HWRE, and MXPR_WRITE_INT_ICCS are driven to the Interrupt Logic. This eliminates having to route the IMIBs across the I Box to the Interrupt Logic.

7.4 Microcode Notes

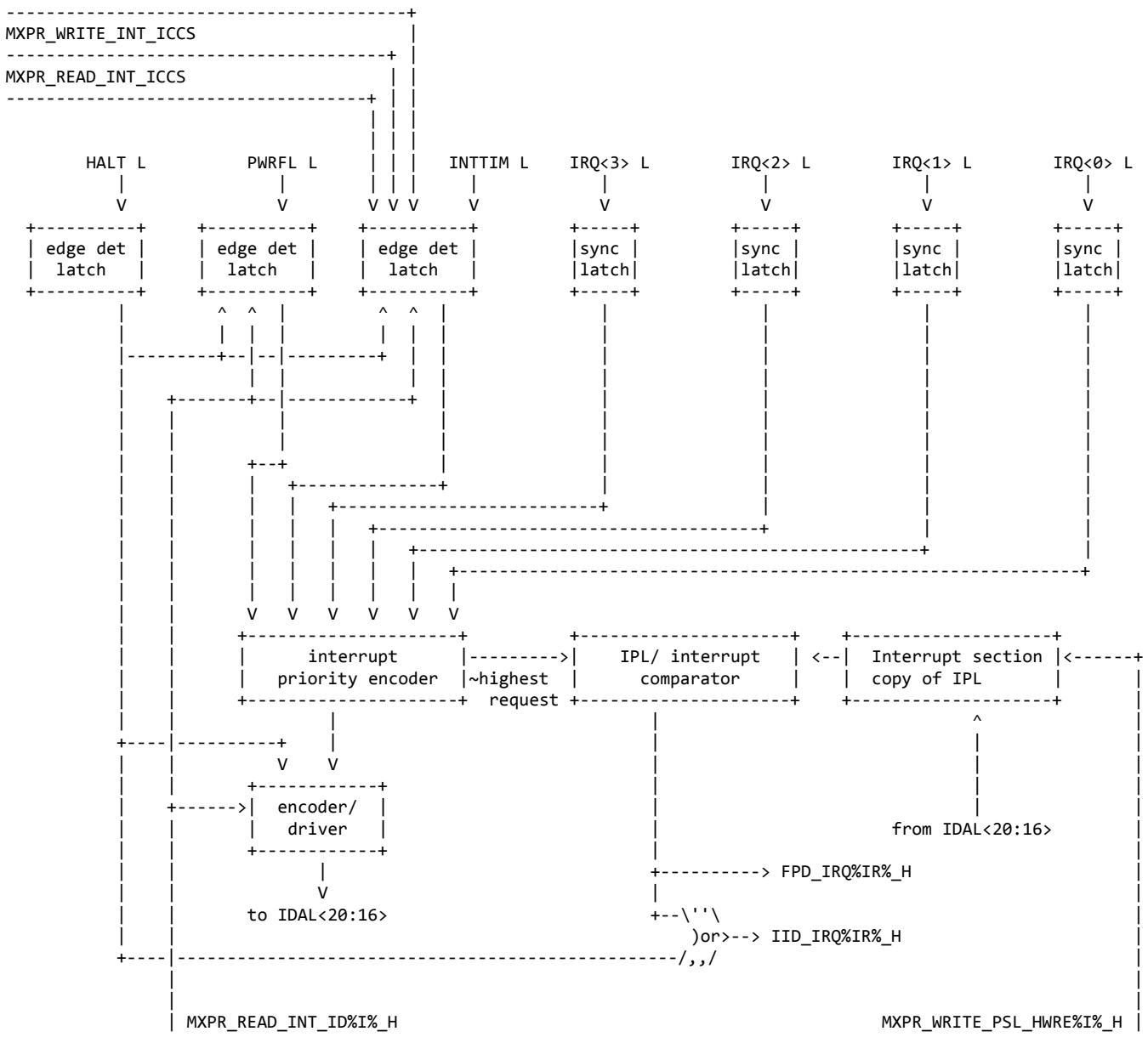
The Interrupt Logic takes four phases to compare the IPL to the interrupt request and drive new branch control signals. This means that microcode cannot take a branch that uses the FPD_IRQ%N%_H or IID_IRQ%N%_H signals the cycle after a MXPR WRITE to PSL.HWRE. This means that a MXPR write to PSL.HWRE that changes IPL should not be followed by an BRANCH IID.

MXPR Register	MXPR Address (hex)	Access
INT.ID	20	Read Only
INT.ICCS	22	Read Write
PSL.HWRE	28	Write Only

The flops for HALT, PWRFL, and INTTIM interrupts are cleared by an MXPR[INT.ID] and the interrupt level being 1F(hex) for HALT, 1E(hex) for PWRFL, and 1C(hex) for INTTIM. The microcode should not do a MREQ_RD_INT for these three interrupts.

7.5 Block Diagram

Interrupt Logic Block Diagram



8.0 CONTROL STORE

The Control Store consists of 1600 39-bit words of read only memory (ROM). Each 39-bit word is divided into a 25-bit portion which controls the data path, and a 14-bit portion which controls the Microsequencer. For details on the microinstruction format, see the last section, "Control Fields Summary".

8.1 Functional Summary

Access to the Control Store is under the control of the Microsequencer. The Microsequencer drives the address of the ROM word to be accessed onto the Microaddress Bus (MAB<10:0>). The Control Store accesses the specified ROM location and places the contents on the Internal Microinstruction Bus (IMIB<38:0>). Note that the Control Store access is pipelined; the Control Store is accessing the next microinstruction to be executed while the current microinstruction is being executed.

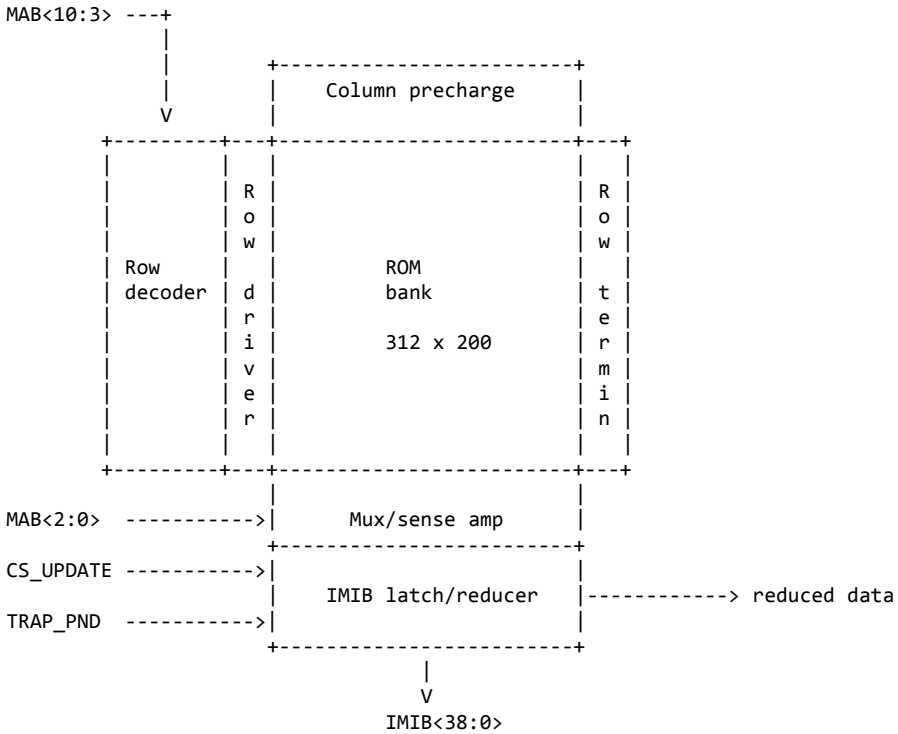
Physically, the Control Store is organized as a ROM array of 312 x 200 bits. Eight microwords are interleaved per row of the array. The output of the 312 x 200 array is fed into local multiplexors/sense amps to create the 39-bit microinstruction output. The new microinstruction is latched in the 39-bit IMIB latch, unless signal CS_UPDATE is not asserted by the DAL Interface. The IMIB latch is overriden to an effective NOP if the Microsequencer asserts signal TRAP_PND.

The IMIB latch includes a data reducer. This reducer uses an XOR and shift combination to checksum the control store output during test mode. The shift out is routed to the DAL Interface and conditionally driven onto Control Status output pin CS<0>.

The new microinstruction address is valid on the MAB during Phase 2. The Control Store access takes place during the next four phases. The new microinstruction is valid on the IMIB during Phase 7.

8.2 Block Diagram

Control Store Block Diagram



9.0 CLOCK LOGIC

The MicroVAX CPU chip receives a 40 Mhz TTL input from the Clock Input (CLKI) pin. This input feeds divide-by-two circuitry which generates a 20 MHz clock. The 20 MHz clock feeds a phase generator which creates four internal chip clocks: PHI1256%C%, PHI2367%C%, PHI3478%C%, and PHI4581%C%. These clocks drive the pass device phase generators which produce the 8 two-tick phase clocks plus PhiWrite and several extended phase clocks. Loading on the phase clocks ranges from 70pf to 240pf.

The clock logic is also responsible for reset synchronization using the external RESET L pin as an input. To provide on-chip initialization, the clock logic generates a reset signal, SYNC_RESET%C%, which will remain active for exactly 16 microcycles from the time the chip clocks start running after deassertion of RESET L. During reset time, the clocks will be in a quiescent state. Any circuitry which must be reset during this time must utilize static logic derived from SYNC_RESET%C% (eg., external pin tristate controls). The first valid clock phase will be PHI1%C%.

To provided system wide synchronization, the clock generator provides a 20MHz Clock Out (CLKO) signal from the MicroVAX chip. This signal remains low continuously while RESET L is asserted. It begins cycling a minimum of 50ns after RESET L is deasserted. The initial phase signals the start of internal phase 1 in the MicroVAX.

It is assumed that the control logic will provide no DC loading on the clock phases. This means that care must also be taken to avoid loading the clock lines due to local phase to phase overlap. Further, it is important that the majority of the capacitive load on the clock be present during the '0' to '1' transition (ie, no charge dumping in the middle of a phase).

9.1 Clock Input Buffer

This circuit consists of a TTL level converter followed by a buffer capable of driving the 40MHz signal to its destinations in the clock generator logic.

9.2 RESET L Input Buffer

This circuit consists of a TTL level converter and driver.

9.3 Reset Synchronizer

This logic consists of three cascaded D flop shift registers clocked with 40MHz. The output of the third flop is guaranteed to be synchronized with 40MHz.

9.4 Divide-By-Two Logic

This logic consists of a toggle flop, clocked by 40MHz and reset by the synchronized output of the reset synchronizer.

9.5 20MHz Clock Drivers

This circuitry takes the output of the divide-by-two logic as input and generates the non-overlapping high drive signals which are input to the phase generators. All the Vdd and Vss current supplied by the phase generators comes from this circuitry.

9.6 Phase Generators

The drivers are basically boot-strapped pass devices enabled by a Johnson Counter which shifts every 25ns. There are active pull-downs when the pass devices are disabled.

9.7 Reset Logic

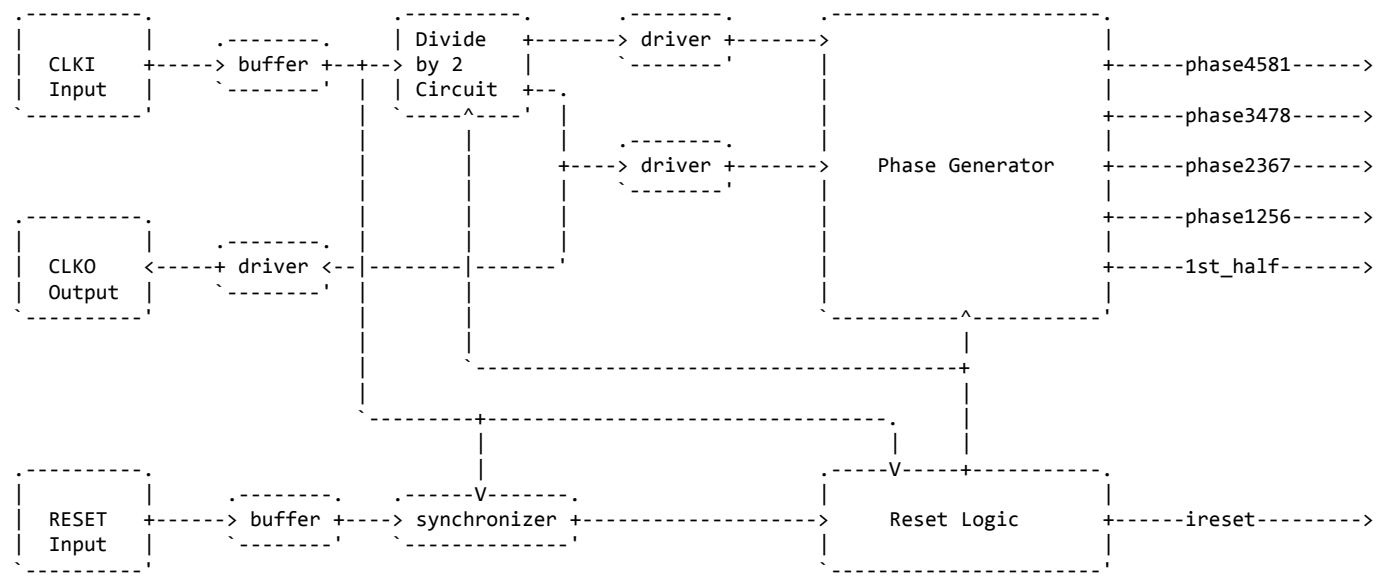
This logic consists of a 4 stage binary counter, which counts once per microcyle. It is released by the reset synchronizer logic; when the count is complete, SYNC_RESET%C% is released to allow chip activation.

9.8 20 MHz Clock Out Circuitry

This circuitry provides a buffered version of the 20MHz clock to an external pin. This buffer is designed to match the internal phase generator delay as well as possible, so that external timing can be accurately specified relative to this signal. This output is held low by reset synchronization such that the first low-high transition on Clock Out marks the initiation of PHI1%C% internally.

9.9 Block Diagram

Clock Logic Block Diagram



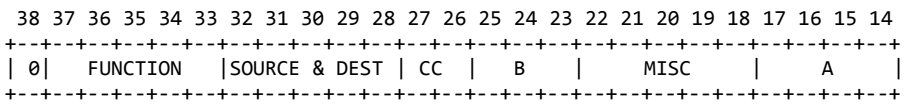
10.0 CONTROL FIELDS SUMMARY

The 39-bit microinstruction is divided into two major sections. The first 25 bits, Data Path Control, control the I Box, E Box, and M Box. This portion is encoded into nine microinstruction formats. The next 14 bits control the Microsequencer. This portion is encoded in two formats.

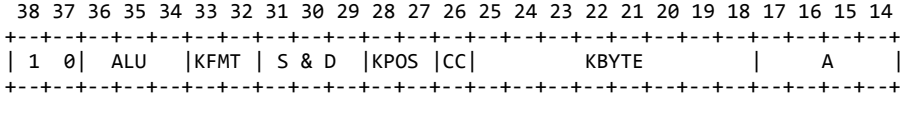
The different microinstruction formats have variable-sized opcode fields.

10.1 Data Path Control Formats

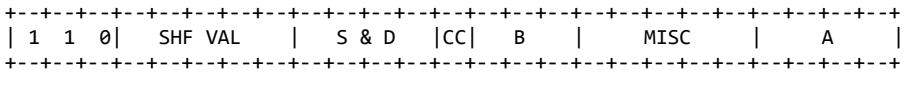
BASIC



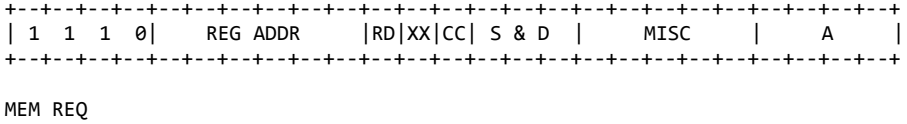
CONSTANT



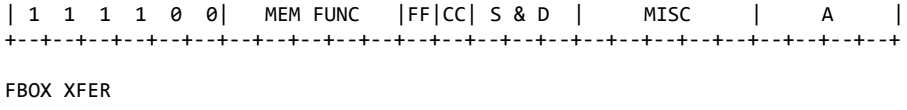
SHIFT



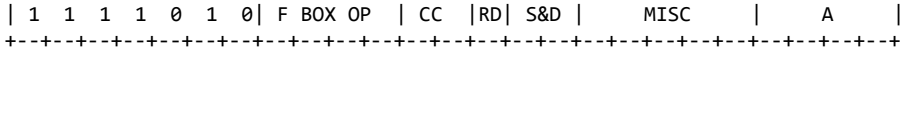
MXPR



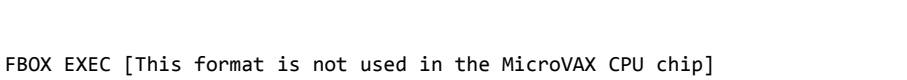
MEM REQ



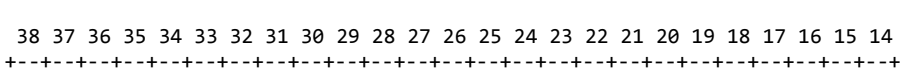
FBOX XFER



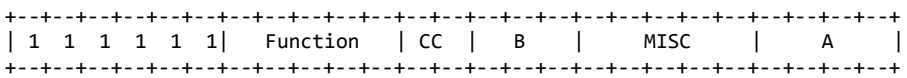
FBOX EXEC [This format is not used in the MicroVAX CPU chip]



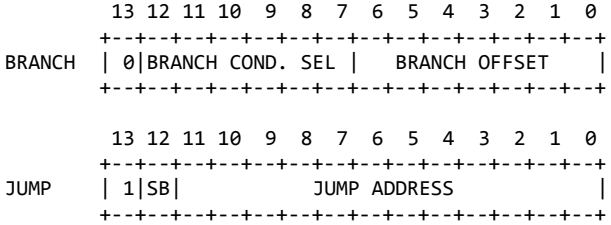
SPECIAL



SPARE



10.2 Microsequencer Control Formats

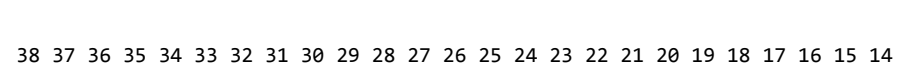


10.3 Data Path Register Addressing

```
GPR[A]          Use A field to address GPR file, PC = GPR[F]

GPR[RN]         Use GPR number from I Box to address GPR file

GPR[RN+1]       Add one to RN before addressing the GPR file

T[A]            A = 0:B   select T[0:11]
                    C     select VA'            (BASIC only)
                    D     select VIBA           (BASIC only)
                    E     select OPCODE         (BASIC, FPU XFER, MEM REQ only)
                    F     select MMGT.STATUS    (BASIC only, write only)

                Note:  T[C:F] can only be used with a restricted subset
                of microinstruction source/destination fields.  T[B:F]
                can not be used with SPECIAL microinstructions.

T[RN]           Use GPR number from I Box to address RN file

                Note:  T[RN] can only be used with T[0:B]

WR[ ]           Dual ported working registers:

                WR[0]:WR[6] - General purpose registers
                WR[7]       - SC (Shift Counter) register

WR[B]           Use B field to address the WR file

WR/PR[A]        A = 0:6 - WR[0:6]
                A = 7   - SC (Shift Counter)
                A = 8   - #1
                    9   - unused
                    A   - VA (Virtual Address) register
                    B   - AT/DL                 (AT in bits <4:3>,
                                                 DL in bits <2:0>, zeroes elswhere)
                    C   - PSL CC's and TP       (PSL.CC in bits <3:0>,
                                                 TP in bit <30>
                                                 T in bit <4>, zeroes elsewhere,
                                                 bits<31:4> are read only)
                    D   - ALU CC's, State       (ALU.CC in bits <3:0>,
                                                 STATE<3:0> in bits <7:4>,
                                                 VAX Restart in bit <30>, zeroes elsewhere,
                                                 bits<31:8> are read only)
                    E   - Sign extension     =   0 if ALU.N = 0
                                                -1 if ALU.N = 1
                    F   - unused

MR[B]           Miscellaneous registers:

                B = 0   - #0
                    1   - #4
                    2   - #2
                    3   - Instr. Data Reg  (ID)
```

```
4   - unused
5   - unused
6   - K(DL)  = 1 if DL = BYTE
               2 if DL = WORD
               4 if DL = LONG or FLOAT
               8 if DL = QUAD or DOUBLE or GRAND
7   - unused
```

10.4 BASIC Microinstruction



This microinstruction format is used for most of the specifier evaluation and computation flows. The micro opcode is CS<38> = 0.

10.4.1 BASIC Function Field CS<37:33> -

This field selects which ALU and/or miscellaneous function to perform on one or two variables. The miscellaneous functions that may be performed are recording specifier data in the RLOG stack, and zero extending the A_ or B_Bus.

Code	Function	Code	Function
0	A.MINUS.B	10	A.MINUS.1
1	B.MINUS.A	11	A.PLUS.B.PLUS.(PSL.C)
2	A.MINUS.B.PLUS(.NOT.PSL.C)	12	A.PLUS.B.PLUS.1
3	A.MINUS.B {RLOG}	13	A.PLUS.B {RLOG}
4	NEG.B	14	A.PLUS.B
5	A.PLUS.1	15	A.PLUS.B.PLUS.(ALU.C)
6	B.MINUS.1	16	A.XOR.B
7	A.AND.NOT.B	17	.NOT.A
8	A.AND.B	18	B.PLUS.1
9	PASS A	19	A.OR.B
A	PASS A {ZEXT}	1A	PASS B
B		1B	
C		1C	
D		1D	PASS B {ZEXT}
E		1E	
F	.NOT.B	1F	PASS B {SLIT}

10.4.2 BASIC Source And Destination Control Field CS<32:28> -

This field determines the sources of data for the A_ and B_Busses, and the destination(s) of the ALU result.

Code	A_Bus	B_Bus	Dest	
0	WR/PR[A]	WR[B]	WR/PR[A]	
1	WR/PR[A]	WR[B]	None	
2	WR/PR[A]	WR[B]	WR[B]	
3	GPR[A]	WR[B]	None	
4	WR/PR[A]	MR[B]	WR/PR[A]	
5	WR/PR[A]	MR[B]	None	
6	T[A]	WR[B]	None	\Do not use T[C:F]\
7				
8	GPR[A]	MR[B]	None	
9	GPR[A]	MR[B]	GPR[A]	
A	GPR[A]	WR[B]	WR[B]	
B	GPR[A]	WR[B]	GPR[A]	
C	T[A]	MR[B]	None	
D	T[A]	MR[B]	T[A]	\Do not use T[C:E]\
E	T[A]	WR[B]	WR[B]	
F	T[A]	WR[B]	T[A]	\Do not use T[C:E]\
10	GPR[RN]	WR[B]	WR/PR[A]	
11	GPR[RN]	WR[B]	GPR[RN] & WR/PR[A]	
12	GPR[RN+1]	WR[B]	WR/PR[A]	
13	WR/PR[A]	WR[B]	GPR[RN+1]	
14				
15	GPR[RN]	MR[B]	GPR[RN] & WR/PR[A]	
16				
17				
18	GPR[RN]	MR[B]	WR/PR[A]	
19	WR/PR[A]	MR[B]	GPR[RN]	
1A				
1B				
1C				
1D				
1E	T[RN]	WR[B]	WR/PR[A]	\Do not use T[C:F]\
1F				

10.5 General Fields

The following fields appear in several of the microinstruction formats. When they appear, they have the following meanings.

10.5.1 CC Field CS<27:26> -

This field gives limited condition code and data type control. When the data type is hardware dependent, the DL register determines it.

Code	Function
0	Hold ALU CC's, DL = LONG
1	Load ALU CC's, DL = DL Register
2	Load ALU CC's, DL = LONG
3	Load PSL CC's, DL = DL Register

This field appears in the BASIC, FBOX XFER, and SPARE microinstructions. For BASIC and SPARE the CC's loaded are the Immediate ALU CC's from the ALU. For FBOX XFER, the CC's loaded are the FPU CC's (N and Z are valid, V = C = 0).

10.5.2 B Field (B_Bus Address) CS<25:23> -

This field addresses the WR file and the MR's.

10.5.3 Miscellaneous Field CS<22:18> -

This field allows operations to happen in parallel with another data path operation.

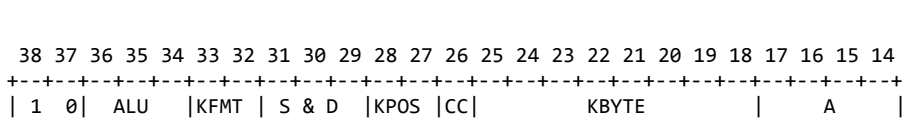
Code	Function	Code	Function
0	NOP	10	SET REEXECUTE FLAG
1	RN <- RN+1	11	CLEAR STATE<3:0>
2	RN <- RN-1	12	SET STATE<0>
3	RN <- SC	13	SET STATE<1>
4	SC <- RN	14	SPECIFIER DECODE
5		15	DL <-- LONG
6	SC <- SC+1 \Use only as OR\	16	LOAD ID LONG CASE
7	WRITE SC	17	LOAD ID LEN(DL) CASE
8	WRITE VA	18	
9	WRITE VA'	19	USE OLD Z
A	WRITE WR6	1A	
B	ENABLE IB PREFETCH	1B	CLEAR IB HALT BITS
C	DISABLE IB PREFETCH	1C	
D	CLEAR MMGT.TD	1D	

E	DL <-- BYTE	1E
F	DL <-- WORD	1F

10.5.4 A Field (A_Bus Address) CS<17:14> -

This field addresses the WR, T, and GPR files. It can also select one of the PR's. GPR[F] is the PC register.

10.6 CONSTANT Microinstruction



This microinstruction allows operations on registers by 32-bit constants. The micro opcode is CS<38:37> = 10. Note: Data length is forced to LONG in this microinstruction.

10.6.1 ALU Control Field CS<36:34> -

This field selects the ALU function that operates on the contents of the A_Bus and the constant.

Code	Function
0	A.PLUS.B
1	A.MINUS.B
2	B.MINUS.A
3	A.AND.B
4	A.OR.B
5	PASS B
6	A.AND.NOT.B
7	A.XOR.B

10.6.2 KFMT, KPOS, And KBYTE Fields CS<33:32,28:27,25:18> -

The KFMT (Constant Format), KPOS (Constant Position), and KBYTE (Constant Byte) fields are decoded to produce a 32 bit constant. KBYTE always provides eight bits of data. KFMT and KPOS indicate the format and position, respectively, of KBYTE within the 32 bit constant, as follows:

KFMT	KPOS	<31:24>	<23:16>	<15:8>	<7:0>
0	x	KBYTE	KBYTE	KBYTE	KBYTE
1	0	00000000	00000000	00000000	KBYTE
1	1	00000000	00000000	KBYTE	00000000
1	2	00000000	KBYTE	00000000	00000000
1	3	KBYTE	00000000	00000000	00000000
2	x	unused			
3	0	11111111	11111111	11111111	KBYTE
3	1	11111111	11111111	KBYTE	00000000
3	2	11111111	KBYTE	00000000	00000000
3	3	KBYTE	00000000	00000000	00000000

10.6.3 CONSTANT Source And Destination Field CS<31:29> -

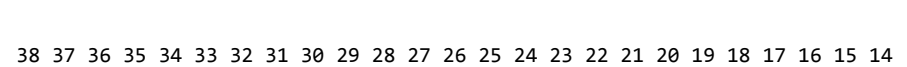
This field controls the source of data to the A_Bus and the destination. The B_Bus is always driven by the constant generator.

Code	A_Bus	Dest	
0	WR/PR[A]	WR/PR[A]	
1	T[A]	T[A]	\Do not use T[C:F]\
2	GPR[A]	GPR[A]	
3	GPR[A]	VA	
4	WR/PR[A]	None	
5	WR/PR[A]	WR6	
6	WR/PR[A]	SC	
7	WR/PR[A]	VA	

10.6.4 CC (Condition Code Control) Field CS<26> -

This field controls the ALU Condition Codes. If CLEAR, the CC's are unaffected. If SET, they are loaded. The CC's loaded are the Immediate ALU CC's.

10.7 SHIFT Microinstruction



This microinstruction, unlike all others, disables the ALU from driving the AW_Bus and enables the Shifter. The micro opcode is CS<38:36> = 110.
Note: Data length for this microinstruction is LONG.

10.7.1 SHF VAL (Shift Value) CS<35:31> -

This field, when used, is the number of bits to shift.

10.7.2 SHIFT Source And Destination CS<31:29> -

This field determines the drivers of the A_ and B_Busses and the write destination. It also controls the shift amount, selecting the SHF VAL field, the DL register, or SC<4:0> for the number of bits to shift right.

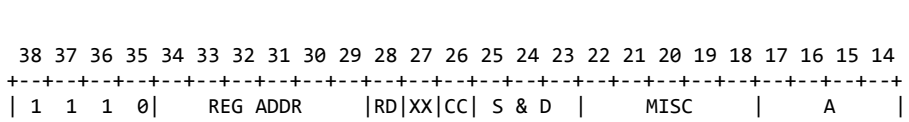
Code	A_Bus	B_Bus	Dest	Shift	Reg.
0	WR/PR[A]	#0	WR[B]	32-SV	SC
1	WR/PR[A]	#0	WR[B]	SV	SC
2	WR/PR[A]	WR[B]	None	32-SV	SC
3	WR/PR[A]	WR[B]	WR[B]	SV	SC
4	WR/PR[A]	#0	WR/PR[A]	32-SV	SC
5					
6	#0	WR[B]	WR/PR[A]	32-SV	SC
7	#0	WR[B]	WR/PR[A]	SV	SC
8	WR/PR[A]	#0	None	32-SV	SC
9					
A	WR/PR[A]	WR[B]	WR/PR[A]	32-SV	SC
B	WR/PR[A]	WR[B]	None	SV	SC
C	GPR[RN]	#0	WR/PR[A]	32-SV	DL
D					
E					
F					

When the Shift Value field is non-zero, SV is the Shift Value field. Otherwise, SV is DL or SC.

10.7.3 CC (Condition Code Control) Field CS<26> -

This field controls the ALU Condition Codes. If CLEAR, the CC's are unaffected. If SET, they are loaded. The ALU CC's are loaded from the Immediate Shift CC's (N and Z are valid, V and C are zero).

10.8 MXPR (Move To/From Processor Register) Microinstruction



This microinstruction allows the E Box to transfer data to or from the other parts of the chip. The micro opcode is CS<39:36> = 1110. Note: Data length for this microinstruction is LONG.

10.8.1 REG ADDR (Register Address) CS<34:29> -

This field selects the register. Note that most of the entries are unused, as the MicroVAX CPU chip keeps some of the architectural registers (P0BR, P1BR, SBR, SCB, PCB, SISR, IS, PSL) in the temporary file, and some (KSP, ESP, SSP, USP) in the Process Control Block.

ADDR	Register	ADDR	Register
0		20	INT.ID
1		21	
2		22	ICCS
3		23	
4		24	
5		25	
6		26	
7		27	
8		28	PSL.HWRE
9	P0LR	29	
A		2A	
B	P1LR	29	
C		2C	
D	SLR	2D	
E		2E	
F	MAPEN	2F	
10 - 1F	unused	30 - 3F	unused

10.8.2 RD (Read/Write Control) CS<28> -

This bit controls the direction of the move. If SET, the E Box is the destination. If CLEAR, the E Box is the source.

10.8.3 MXPR Source And Destination CS<25:23> -

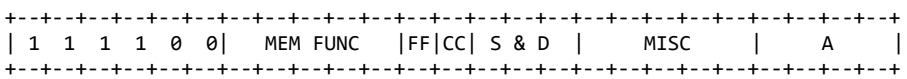
This field controls which file is the source/destination during MXPR microinstructions. Read/write control is microinstruction context dependent.

Code	Source	Dest	
0	WR/PR[A]	WR[A]	; Note: PR[9:F] cannot be source or destination
1	T[A]	T[A]	; Note: T[C:F] cannot be source or destination
2	GPR[A]	GPR[A]	
3	GPR[RN]	GPR[RN]	
4			
5			
6			
7			

10.8.4 MXPR CC Field CS<26> -

This field causes loading of the ALU CC register. If clear, it is not loaded. If set, it is loaded from the Immediate ALU CC's.

10.9 MEM REQ (Memory Request) Microinstruction



This microinstruction performs memory reads and writes. Its micro opcode is CS<38:33> = 111100. The data length of the request is either explicitly specified or is given by the DL register.

10.9.1 MEM FUNC (Memory/Bus Function) CS<32:28> -

This field selects which memory/translation/bus function to perform.

MEMORY REQUEST FUNCTION FIELD DEFINITION

	Code	Function	Reference Type	Privilege Check	Access Mode	Data Length	Address Source	VA' after Reference	
NORMAL									
	00	Read	Virtual	Read	PSL	DL	VA	VA+4	
	01	Read	Virtual	(AT)	PSL	DL	VA	VA+4	
	02	Read	Virtual	Write	PSL	DL	VA	VA+4	
	03	Read Lock	Virtual	Write	PSL	DL	VA	VA+4	
	08	Read	Virtual	Read	PSL	LONG	VA	VA+4	
	18	Read	Virtual	None	-	LONG	VA'	VA'+4	
	04	Write	Virtual	Write	PSL	DL	VA	VA+4	
	05	Write Unlock	Virtual	Write	PSL	DL	VA	VA+4	
	09	Write	Virtual	Write	PSL	LONG	VA	VA+4	
	19	Write	Virtual	None	-	LONG	VA'	VA'+4	
PHYSICAL									
	0A	Read	Physical	None	-	LONG	VA	VA+4	
	0B	Write	Physical	None	-	LONG	VA	VA+4	
	1A	Read	Physical	None	-	LONG	VA'	VA'+4	
	1B	Write	Physical	None	-	LONG	VA'	VA'+4	
KERNEL									
	10	Read	Virtual	Read	KERNEL	LONG	VA'	unchanged	
PROBE									
	06	Probe	Virtual	Read	PSL	-	VA	unchanged	aligned only
	07	Probe	Virtual	Write	PSL	-	VA	unchanged	aligned only
	0E	Probe	Virtual	Read	STATE	-	VA	unchanged	aligned only
	0F	Probe	Virtual	Write	STATE	-	VA	unchanged	aligned only
	16	Probe	Virtual	Read	KERNEL	-	VAP'	unchanged	aligned only
RD PTE									
	12	Read SPTE	Physical	None	-	LONG	VA'	unchanged	aligned only
	13	Read PPTE	Virtual	Read	KERNEL	LONG	VA'	unchanged	aligned only
RD INT									
	0C	Read Interrupt Vector	Physical	None	-	LONG	VA=LEVEL	VA+4	aligned only

10.9.2 FF (F Box Flag) Field CS<27> -

This bit is not used in the MicroVAX CPU chip.

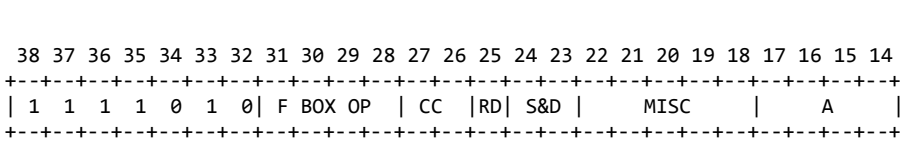
10.9.3 CC (PSL Condition Code Control) Field CS<26> -

This bit, when SET, causes the PSL CC's to be loaded conditionally on the (macro) opcode. When CLEAR, the CC's are unaffected.

10.9.4 MEM REQ Source And Destination CS<25:23> -

Code	Source	Dest	
0	WR/PR[A]	WR[A]	; Note: PR[9:F] cannot be source or destination
1	T[A]	T[A]	; Note: T[C:D,F] cannot be source or destination
2	GPR[A]	GPR[A]	
3	GPR[RN]	GPR[RN]	
4			
5			
6			
7			

10.10 FBOX XFER Microinstruction



This microinstruction transfers data between the E Box and the Floating Point Unit (FPU). It also controls the status signals sent out to the FPU. The micro opcode is CS<38:32> = 1111010.

10.10.1 FBOX OP (FPU Operation) Field CS<31:28> -

This field determines the status code sent out to the FPU. The implemented codes are:

Code	Function
0	FPU command
1	FPU data
2	FPU command alternate
3	FPU response
4-6	unused
7	FPU response, disable error trapping
8-F	unused

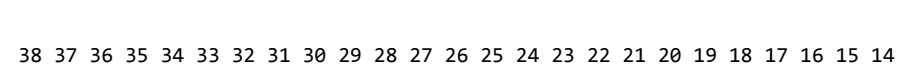
10.10.2 RD Field CS<25> -

This field determines the direction of the transfer. If SET, the E Box is the destination (transfer from FPU to CPU). If CLEAR, the E Box is the source (transfer from CPU to FPU).

10.10.3 FBOX XFER Source And Destination Control Field CS<24:23> -

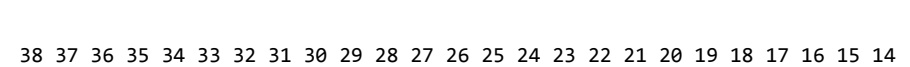
Code	Source	Dest	
0	WR/PR[A]	WR[A]	; Note: PR[9:F] cannot be source or destination
1	T[A]	T[A]	; Note: T[C:D,F] cannot be source or destination
2	GPR[A]	GPR[A]	
3	GPR[RN]	GPR[RN]	

10.11 FBOX EXEC Microinstruction



This microinstruction is not used in the MicroVAX CPU chip.

10.12 SPECIAL Microinstruction

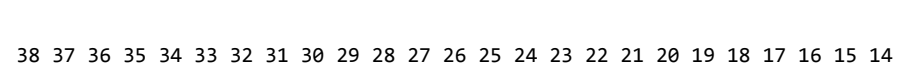


This microinstruction allows certain non-time critical functions. The microword is broken up into three encoded fields. The micro opcode is CS<38:33> = 111110.

Note: During the execution of this microinstruction, VA drives the A_Bus and ALU is in PASS A mode (W_Bus <-- A_Bus). A must be [0:A].

MISC1 (CS<32:28>)	MISC2 (CS<27:23>)	MISC3 (CS<22:18>)
0 NOP	0 NOP	0 NOP
1 POP RLOG	1 INVALIDATE MTB	1
2	2 PC<--BPC	2
3	3 SET STATE<2>	3
4 SET FPNT	4 SET STATE<3>	4
5 CLR FPNT	5	5
6	6	6 SET VAX TRAP REQUEST
7	7	7 CLEAR VAX TRAP REQUEST
8	8 WRITE W BUS TO T[A]	8
9	9	9
A	A	A
B	B	B
C	C	C
D	D	D
E	E	E
F	F	F
10 CLEAR REEXECUTE	10 SET STATE<4>	10
11	11 SET STATE<5>	11
12	12 SET STATE<6>	12
13	13 SET STATE<7>	13
14	14 CLEAR STATE<4>	14
15	15 CLEAR STATE<5>	15
16	16 CLEAR STATE<6>	16
17	17 CLEAR STATE<7>	17
18	18	18
19	19	19
1A	1A	1A
1B ZAP TB(LRU) SET REPROBE	1B	1B
1C ENABLE REDUCERS	1C	1C
1D ENABLE CSTORE READOUT	1D	1D
1E	1E	1E
1F	1F	1F

10.13 SPARE Microinstruction



This microinstruction implements additional ALU functions for integer multiply and divide. Its micro opcode is CS<38:33> = 111111.

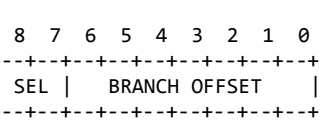
10.13.1 SPARE Function Field CS<32:28> -

This field provides both source/destination control and ALU control.

Code	A_Bus	B_Bus	Dest	Function
0	WR/PR[A]	WR[B]	WR[B]	CON ADD/PASS, SHIFT INTO SC, MSB = C
1	WR/PR[A]	WR[B]	WR[B]	CON ADD/PASS, SHIFT INTO SC, MSB = N XOR V
2	WR/PR[A]	WR[B]	WR[B]	SHIFT WR[A]&SC LEFT, LSB = 0
3-1F				

Detailed descriptions of these functions are included in the E Box section.

10.14 BRANCH Microinstruction



This format (of the Microsequencer Control portion of the microword) is used when CS<13> = 0. It allows relative branching on condition, return-from-subroutine, and case branching.

10.14.1 BCS (Branch Condition Select) Field CS<12:7> -

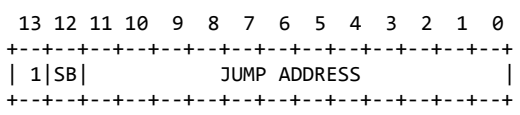
This field selects the branch condition or special function to be used in generating the next microaddress.

Code	Branch Cond	Code	Branch Cond
0	KERNEL MODE	20	NOT FPU SIGNAL
1	NO FPD INTERRUPTS PENDING	21	MICROTRAP ADDRESS REQUEST (NOP)
2		22	
3		23	
4		24	IF A GOTO IF V OR BWL NSD
5	IF AVMF GOTO IF BWL NSD	25	IF AVM GOTO IF BWL NSD
6		26	IF BWL IID ELSE GOTO
7	NOT ALU Z	27	
8		28	
9	NOT FPU	29	IF R OR M GOTO
A	ALU N	2A	IF BCOND LOAD VIBA & PC ELSE IID
B	ALU Z	2B	SUCCESSIVE IID
C		2C	NSD
D		2D	IF AV NSD ELSE GOTO
E	ALU V	2E	IID
F	ALU C	2F	LOAD VIBA & PC & BRANCH ALWAYS
10		30	ALU NZVC ; CASE BRANCH
11		31	SC 3-0 ; CASE BRANCH
12		32	
13		33	SC 7-4 ; CASE BRANCH
14		34	STATE 3-0 ; CASE BRANCH
15		35	MMGT STATUS ; CASE BRANCH
16		36	OPCODE 3-0 ; CASE BRANCH
17		37	DL.MBOX STATUS; CASE BRANCH
18		38	STATE 7-4 ; CASE BRANCH
19	VA MEM REF NOT OK	39	
1A	RMODE	3A	
1B		3B	
1C		3C	IF MEM REF OK RETURN+BO
1D		3D	RETURN+BO
1E		3E	
1F	NOT INT STACK	3F	

10.14.2 BO (Branch Offset) Field CS<6:0> -

This field contains the signed offset that is added to the uPC when a branch is taken or a return is done.

10.15 JUMP Microinstruction



This format (of the Microsequencer Control portion of the microword) is used when CS<13> = 1. It allows absolute unconditional branching and subroutine calling.

10.15.1 SB (SUBROUTINE) Control Bit CS<12> -

This bit controls pushing the microstack. If it is SET, the address of the microword plus one is pushed onto the microstack (a jump to subroutine is done). If not, the microstack is unaffected.

10.15.2 JUMP ADDRESS Field CS<11:0> -

This field is used when JUMP microinstructions are being done. It provides the 12 bits of the jump microaddress. This allows freedom to jump anywhere in the 4K microcode space.