VAX 11/780

DATA PATH DESCRIPTION

AA-H307A-TE

February 1979

digital equipment corporation - maynard, massachusetts

First Printing, February 1979

The postage prepaid READER'S COMMENTS form on the last page of this document requests the user's critical evaluation to assist us in preparing future documentation.

The following are trademarks of Digital Equipment Corporation:

DIGITAL	DECsystem-10	MASSBUS
DEC	DECtape	OMNIBUS
PDP	DIBOL	OS/8
DECUS	EDUSYSTEM	PHA
UNIBUS	FLIP CHIP	RSTS
COMPUTER LABS	FOCAL	RSX
COMTEX	INDAC	TYPESET-8
DDT	LAB-8	TYPESET-10
DECCOMM	DECsystem-20	TYPESET-11
ASSIST-11	RTS-8	ITPS-10
TMS-11		

CHAPTER 1 DATA PATH SPECIFICATION

CHAPTER 2 MICRO SEQUENCER SPECIFICATION

CHAPTER 4 INSTRUCTION BUFFER

CHAPTER 5 INTERRUPTS & EXCEPTIONS

CHAPTER 1

DATA PATH SPECIFICATION

The VAX 11/780 CPU DATA PATH consists of four sections for processing data and addresses as specified in the VAX and compatibility mode instruction sets. The EXPONENT, ADDRESS, ARITHMETIC, and DATA sections each operate as independent units which are capable of processing data or addresses in parallel with the operations taking place in the other sections.

The block diagram for the VAX 11/780 CPU Data Path is given in Figure 1-1.

1.1 ARITHMETIC SECTION

The ARITHMETIC SECTION of the DATA PATH contains the GENERAL REGISTERS, a bit mask generator, a constant generator, shifter, temporary storage registers, and register inputs from the ADDRESS and DATA SECTIONS so that the necessary arithmetic and logical operations can be performed on data and addresses.

Three data types are processed in the ARITHMETIC SECTION which consist of 8 bit bytes, 16 bit words, and 32 bit long words.

The data type is controlled by the UDT control field which selects the desired sign or zero extension, index constants, index shifting, and GENERAL REGISTER storage control.

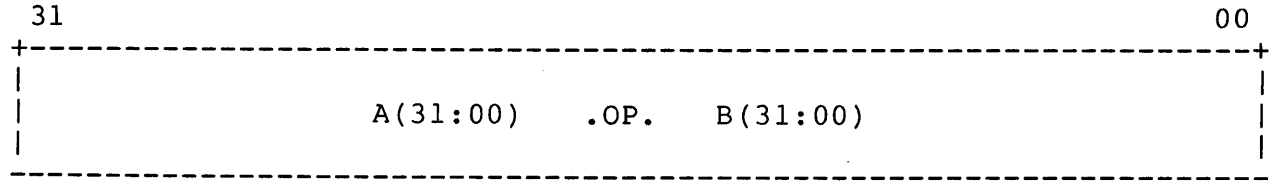
1.1.1 ALU

Arithmetic and Logic Unit

The ALU provides the main processing power of the ARITHMETIC SECTION by performing 32 bit arithmetic with fast carry look ahead logic or 32 bit logical operations.

ALU data types:

32 bit arithmetic or logical operation (.OP.).



ALU CONTROL

The ALU operation is controlled by the UALU field of the UWORD which may define the function explicitly or allow the function to be controlled by the instruction decode logic. The command codes are grouped into two classes, arithmetic and logic modes. The arithmetic mode operation requires more processing time and puts restrictions on the source data used in the case of slow constants (section 1.3.5), register set contents (section 1.3.7), Temporary scratch pad contents (section 1.3.8), packed floating format (section 1.3.3) or the conditional BMX selection of PC in the state immediately following the loading of the first specifier (section 1.3.3). There are no similar restricitons on logic mode functions.

UALU

Uword Arithmetic and Logic Unit control field, 4 bits.

	Uword hex code	Function	Mode
(D)	0	A.MINUS.B	A
	1	A.MINUS.B (RLOG)	A
	2	A.MINUS.B.MINUS.1	A
	3	INSTRUCTION DEP.	A&L
	4	A.PLUS.B.PLUS.1	A
	5	A.PLUS.B	A
	6	A.PLUS.B.(RLOG) A.ORNOT.B	A
	7	A.OR.B	L
	8	A.XOR.B A.ANDNOT.B	L
	9	A.AND.B	L
	A	.NOT.A	L
	B	A.PLUS.B.PLUS.C	A
	C	A.OR.B	L
	D	A.AND.B	L
	E	B	L
	F	A	L

(D)

When the UALU field = 3 the ALU functions are the result of the instruction being executed. Under Instruction Dependent mode the full 32 logical and arithmetic functions of the ALU (74S181) are available.

(D)

The C input to the ALU can be forced for either PSL C or NOT PSLC. This will provide the useful ALU functions of:

A.PLUS.B.PLUS.C	,	A.PLUS.B.PLUS.NOT.C
A.MINUS.B.MINUS.C	,	A.MINUS.B.MINUS.NOT.C
A.PLUS.C	,	A.PLUS.NOT.C
A.MINUS.C	,	A.MINUS.NOT.C
A.PLUS.A.PLUS.C	,	A.PLUS.A.PLUS.NOT.C

Independent of the ALU function selected, the AMX can be forced to zero giving: B+1 etc.

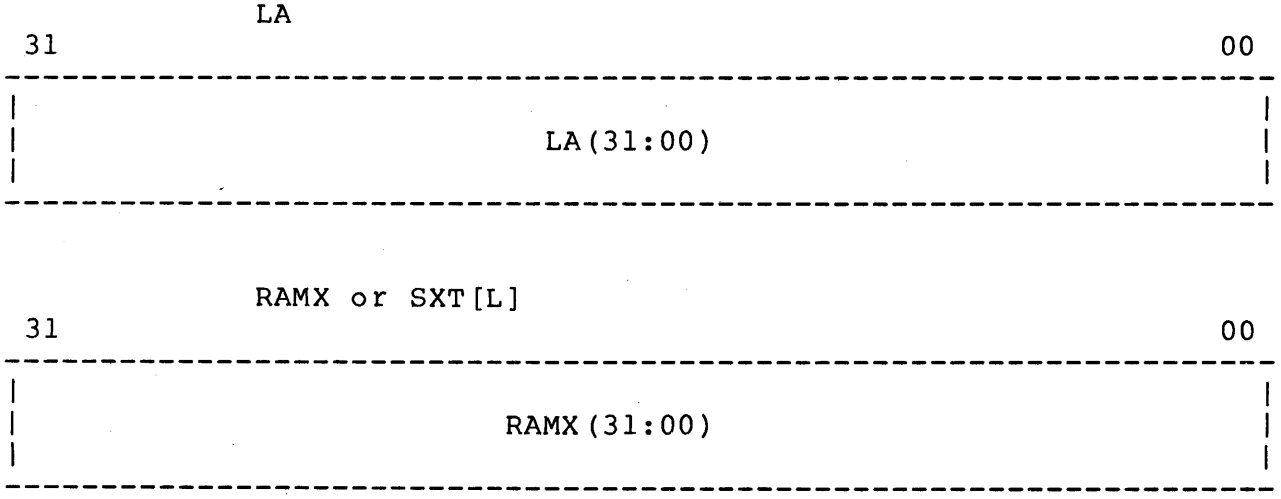
When UALU field = 1 or 6 the RLOG stack is updated with the general register (RA) address, the lower four bits of the KMX and a bit to determine if an add or subtract is requested.

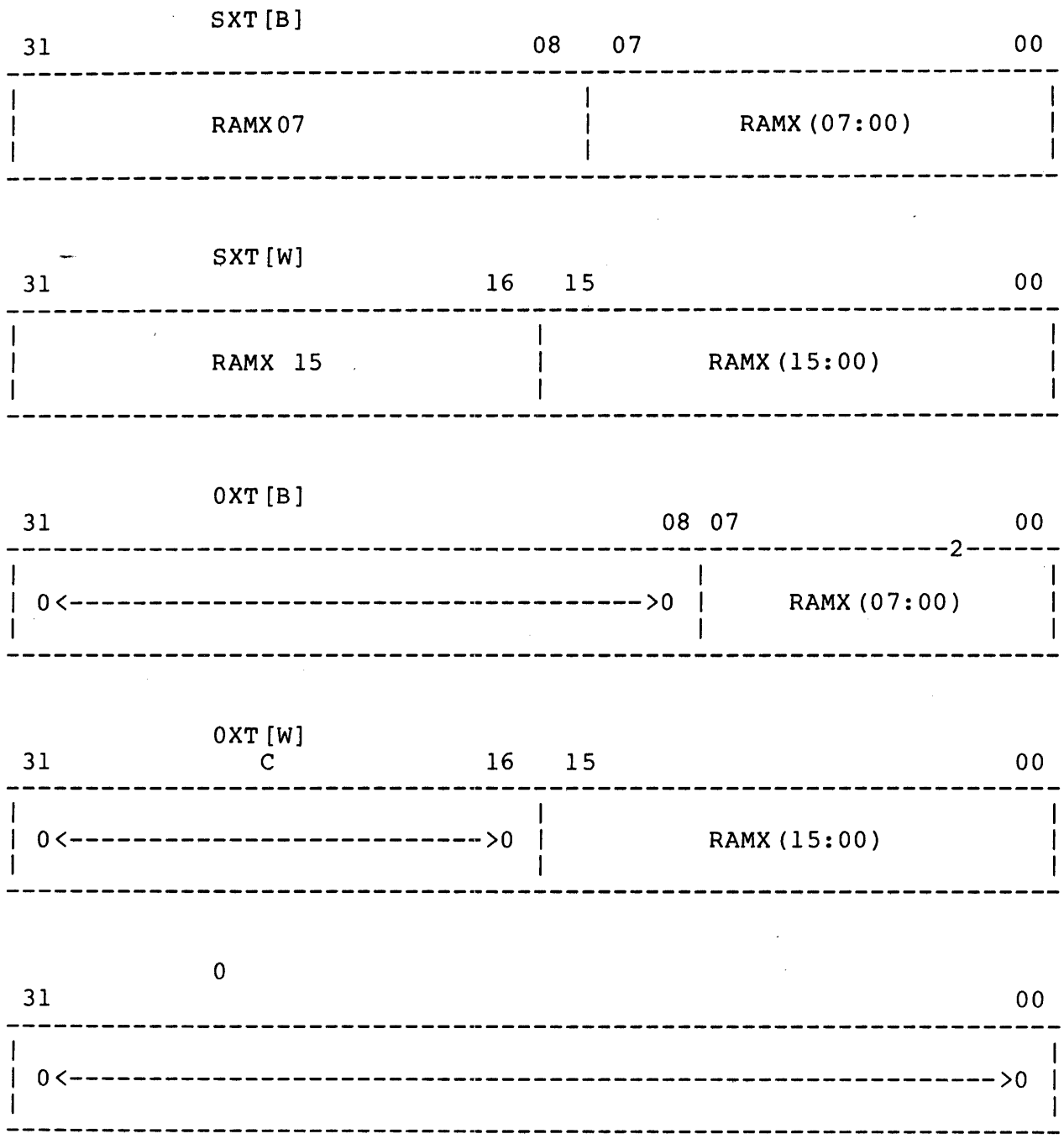
1.1.2 AMX

alu A input MultipleXor

The AMX provides the data source for the A input to the ALU. Sign or zero extension of the data type from either the D or Q register in the DATA SECTION is provided since the ALU operates on 32 bit data structures only.

AMX data types:




SXT[B]
31
08
07
00
RAMX07
RAMX(07:00)
SXT[W]
31
16
15
00
RAMX 15
RAMX(15:00)
0XT[B]
31
08
07
00
2
0
0
RAMX(07:00)
0XT[W]
31
C
16
15
00
0
0
RAMX(15:00)
0
31
00
0
0

AMX CONTROL

The data format of the RMX is selected by the UAMX field of the UWORD, while the sign or zero extension position is chosen by the UDT field. The zero extension of a long word data type is a special case and will force all zeroes at the output of AMX.

UAMX

Uword alu A input MultipleXor control field, 2 bits.

	LA	
1	RAMX	
2	RAMX SXT [UDT]	(Sign eXTension)
3	RAMX 0XT [UDT]	(. eXTension)

UDT

Uword Data Type select field, 2 bits.

0	LONGWORD	: SXT[L] or 0
1	WORD	: SXT[W] or 0XT[W]
2	BYTE	: SXT[B] or 0XT[B]
3	INSTRUCTION DEPENDENT	: Any of above

When UDT = 3 the instruction decode logic determines the data type to be used in the ARITHMETIC SECTION. This provides data type information for instruction execution and operand specifier evaluation. For instructions requesting Float Quad or Double Float context will be a LONGWORD data type.

1.1.3 BMX

alu B input MultipleXor

The BMX provides the data source for the B input to the ALU. The GENERAL REGISTERS and the D register are routed to the ALU B input so the instruction executions requiring the A.MINUS.B function can be performed without swapping the operand from one ALU input to the other.

The PC input is provided to route the address information in the ADDRESS SECTION back into the ARITHMETIC SECTION. This also allows PC displacement addressing modes to be calculated with the AMX selecting the sign extended displacement value.

(D)

The PC or LB input is conditionally selected where source mode R.EQL.PC selects the PC otherwise LB input is chosen. The BMX uword field must be set to 1.

(D)

The RLOG, PCSV input is selected by providing the signal UMSC Read RLog and setting the BMX uword.= 0.

(D)

The packed floating format puts back together the fraction (D), the exponent (EALU), and the sign (SD) of the result of the Data path floating point operations into VAX floating point data format.

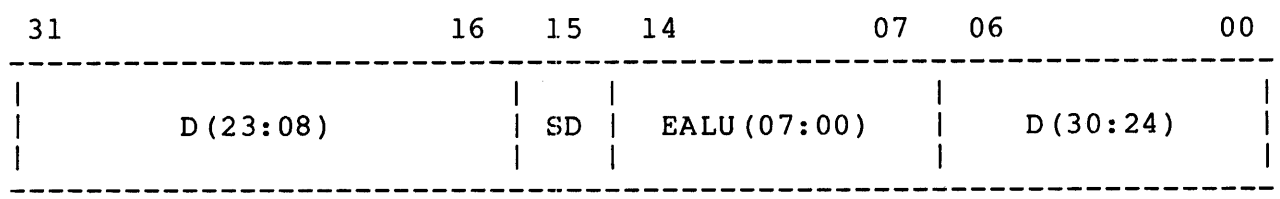
The LC input allows temporary storage locations to be routed into the ALU for ARITHMETIC SECTION operations.

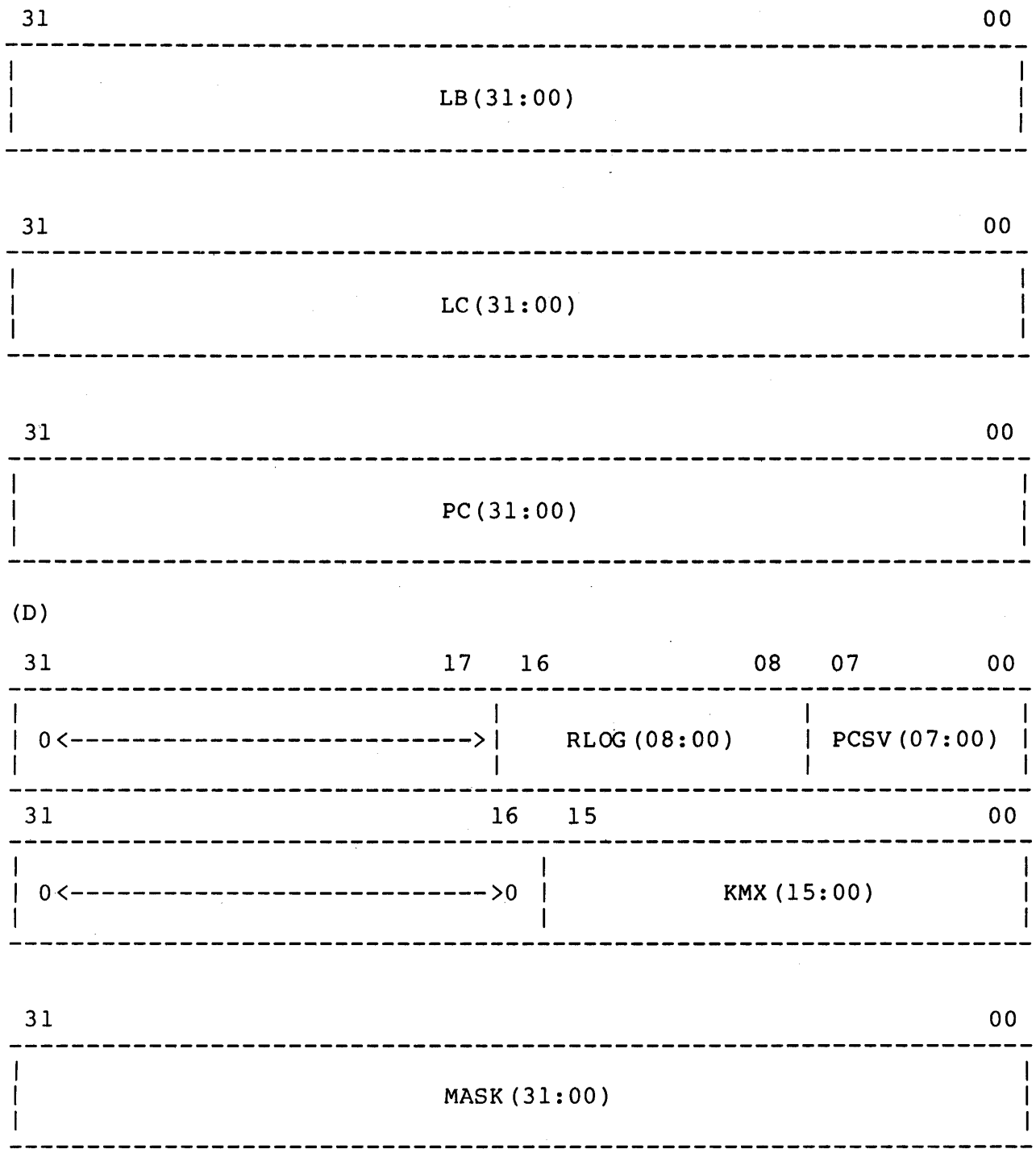
The KMX input supplies the constants necessary in the execution of instructions and evaluation of operand specifiers.

The MASK input routes the output of the bit MASK generator to the ALU for logical operations. BMX data types:

(D)

(D)




31
00
LB(31:00)
31
00
LC(31:00)
31
00
PC(31:00)
(D)
31
17
16
08
07
00
0<
RLOG(08:00)
PCSV(07:00)
31
16
15
00
0<
>0
KMX(15:00)
31
00
MASK(31:00)

BMX CONTROL

The BMX input is selected by the UBMX field of the UWORD.

UBMX

Uword alu B input MultipleXor control field, 3 bits.

0	MASK
1	COND
2	Packed Floating
3	LB
4	LC
5	PC
6	KMX
7	RBMX

(D)

When UBMX = 2, a packed floating point data type is assembled by taking the fraction position from the DREG, the sign from control logic, and the exponent from the EALU. Due to the routing delays involved both the EALU and the ALU must be selected for logic mode to insure that data is available in the ARITHMETIC SECTION. The SD bit contains the sign of the destination fraction in floating point operations. SD had been loaded and controlled by the USGN field of the UWORD during execution of floating point instructions. Conditional selection of PC and LB is provided when UBMX=1. The selection of Rlog must be made with UBMX=0 and the presents of the signal UMSC Read Rlog.

USGN

Uword SiGN control field, 3 bits.

(D)

0 NOP

1. SS<--ALU15
   SD<--SD

2. SS<--SD
   SD<--SD

3. SS<--$\overline{SS}$
   SD<--$\overline{SD}$

4. SS<--SS
   SD<--SS

5. SS<--ALU15.XOR.SS
   SD<--ALU15

6. SS<--.NOT.ALU15.XOR.SS
   IF IR[1] ELSE, SS<--ALU15.XOR.SS
   SD<--ALU15

7. SS<--0
   SD<--0

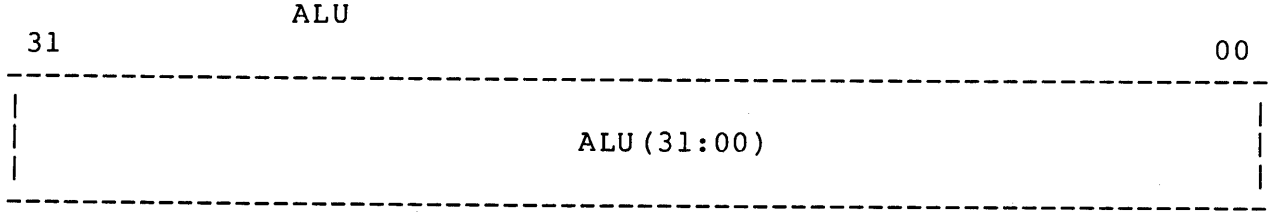
When UMSC = Read Log the RLOG stack and PCSV data is supplied to the B input of the ALU. This selection causes a pointer into the RLOG stack to be decremented at the end of the Micro-instruction allowing the next element of the RLOG stack to be read in a subsequent Micro-instruction. The PCSV data remains constant until a new instruction is begun.

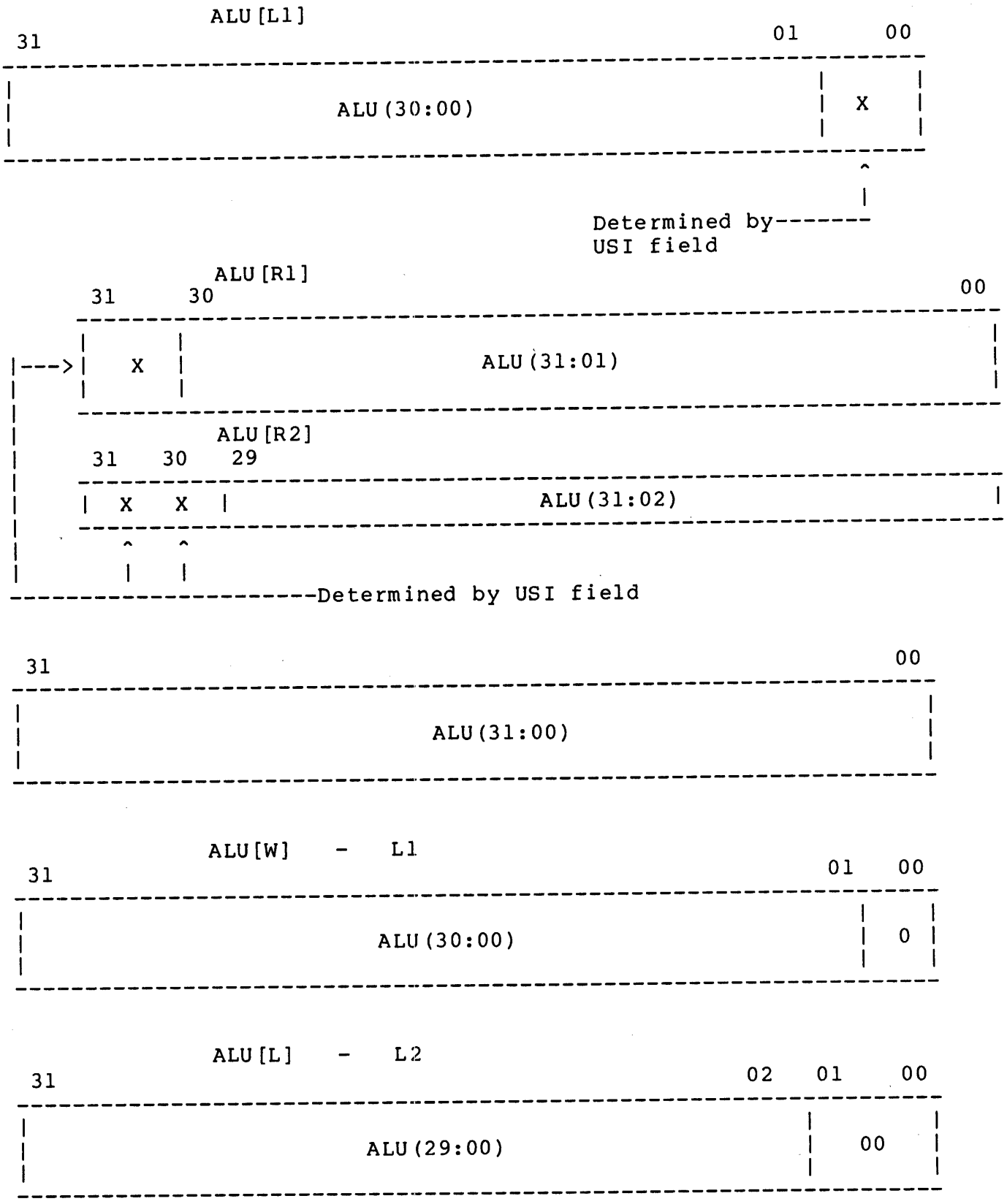
1.1.4 SHF

SHiFter

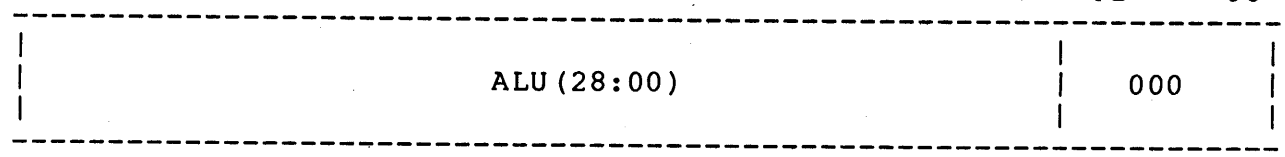
The SHF is used as a multiplier to create the correct index values for address caculations in INDEX mode specifier evaluations. The index value is multiplied by the appropriate number to index tables of byte, word, long word, or quad word data entries. For byte organized tables the index value is multiplied by 1 (ALU directly with no left shift - L0), word tables the index value is multiplied by 2 (left shift of 1 - L1), long word tables by 4 (left shift of 2 - L2), and quad word tables by 8 (left shift of 3 - L3). The data type information is either explicitly determined to be BYTE, WORD, or LONG WORD or is determined by the instruction decode logic and is controlled by the UDT field.

The SHF is also used in the execution of multiply and divide and compatibility mode rotate and shift instructions. In these cases the SHF has the capability of either a left shift by 1 (L1) or a right shift by 1 (R1) or by 2 (R2) with the shift input controlled by the USI field. SHF data types:




ALU[L1]
31
01
00
ALU(30:00)
X
Determined by
USI field
ALU[R1]
31
30
00
X
ALU(31:01)
ALU[R2]
31
30
29
X
X
ALU(31:02)
Determined by USI field
31
00
ALU(31:00)
ALU[W] - L1
31
01
00
ALU(30:00)
0
ALU[L] - L2
31
02
01
00
ALU(29:00)
00

ALU[Q] or ALU[L3] - L3



SHF CONTROL

The data type of the SHF is selected by the USHF field of the UWORD. In three of the shifted data types the shift input is determined by the USI field. The UDT field determines which of the remaining shifted data types is chosen.

USHF

Uword SHiFter control field, 3 bits.

(D)	0	ALU
	1	ALU[L1]
	2	ALU[R1]
	3	ALU[UDT]
	4	ALU[R2]
	5	ALU[L3]
	6	DO NOT USE
	7	DO NOT USE

For USHF = 1, 2, or 4 the USI field determines the shift input.
For USHF = 3 or 5, the shift input is zeroes.

USI

Uword Shift Input control field, 3 bits.

(D)	0	PSL[N]
	1	ALU31 (Do NOT use when writing RA, RB or RC)
	2	0
	3	0
	4	0
	5	Q31
	6	0
	7	1

For USHF = 3 the UDT field determines the shift amount.

UDT

Uword Data Type select field, 2 bits.

0	LONG WORD	:	ALU[L] - L2
1	WORD	:	ALU[W] - L1
2	BYTE	:	ALU[B] - L0
3	INSTRUCTION DEPENDENT	:	Any of above and ALU[Q] - L3

(D)

When UDT = 3 the instruction decode logic, SP1CON(02:00) determines the data type which will control the shifting.

1.1.5 KMX

Constant (K) MultipleXor

FK

Fast constant (K) multiplexor

SK

Slow constant (K) rom

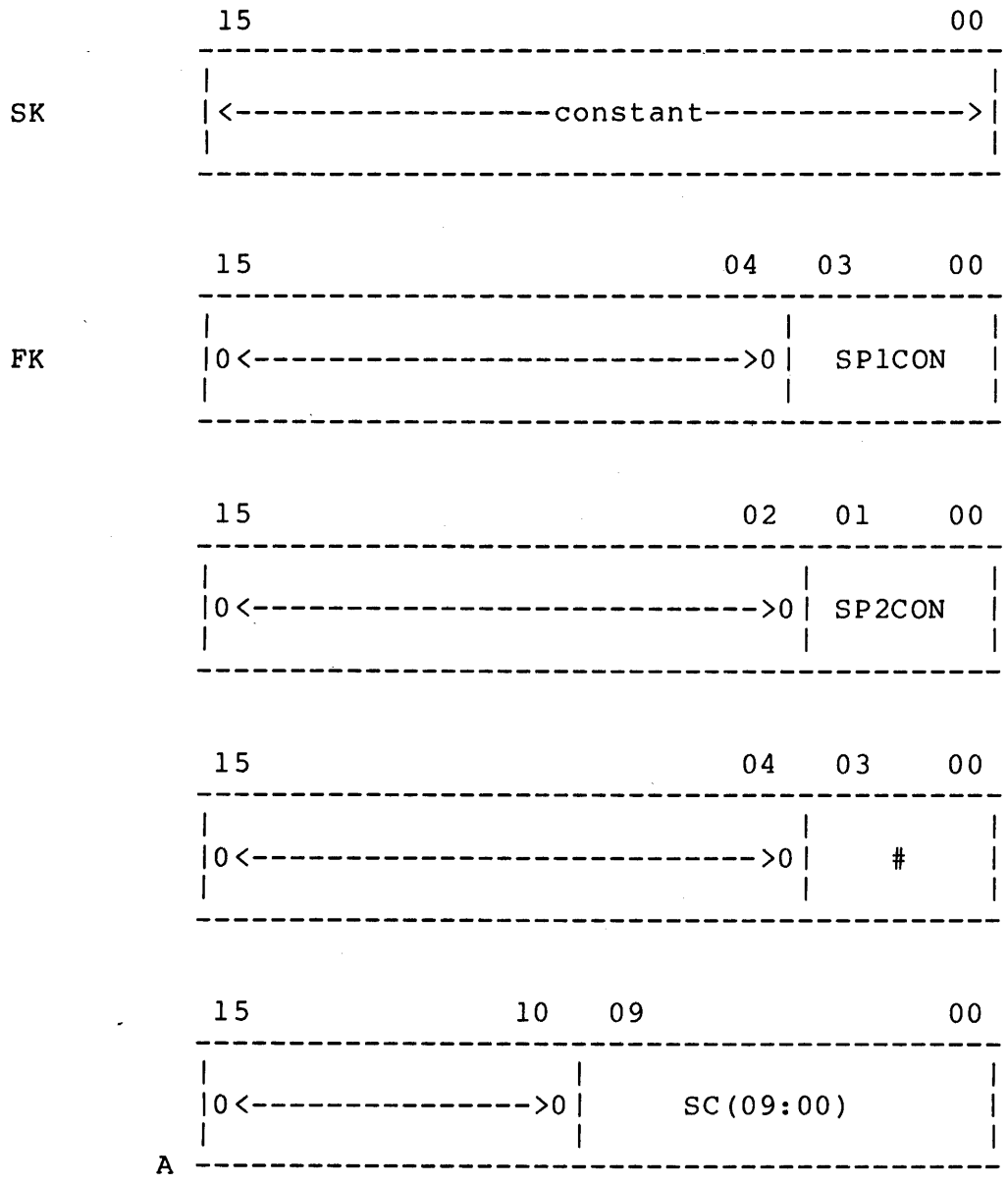
The KMX is used to select a constant explicitly specified by the micro instruction or to select an index constant dependent on the data type and register type of the operand specifier being evaluated. In VAX mode of operation, SP1CON (Specifier 1 CONstant) is a number determined from the data type of the operand specifier being evaluated. In 11 Compatibility mode SP1CON is a number determined from the data type and register type of the source mode/reg. field of the instruction. SP2CON (Specifier 2 CONstant) in 11 Compatibility mode is the number 1 or 2 determined from the data type and register number of the destination mode/reg. field of the instruction, and in Vax mode is the number 0. SP1CON may be the number 1,2,4, or 8. Both SP1CON and SP2CON are generated by the instruction decode logic.

The SC input to FK provides a path for the 10 bit data in the EXPONENT SECTION to enter the ARITHMETIC SECTION and is also used as a constant register in arithmetic operations.

The fast constants from FK are generally used to increment or decrement data and in the evaluation of auto-increment and auto-decrement addressing modes.

The SK rom provides the remainder of micro program constants used to execute instructions, isolate bits or bit fields, provide exponent biasing and select shift constants.

KMX data types:



KMX, SK, FK CONTROL

The constants are selected by the UKMX field of the UWORD. Constants selected from the SK ROM take additional lookup time and therefore must only be used in the ARITHMETIC SECTION when the ALU is selected for logic mode. This means that constants from SK must be stored in a register before being used in arithmetic operations in the ALU. Any register to which a load path exists may be used (including t.e SC and then selected from FK in the subsequent micro instruction).

An alternate method for using the SK would be to select a constant in one micro instruction and then (selecting the same constant) in the next micro instruction, use it in an arithmetic operation. Care must be taken to insure that a micro trap cannot occur between the two micro instructions so that the access time of the slow constant is violated if the second micro instruction is restarted.

(D)

The only restriction in using the slow constants in the EXPONENT SECTION is that the NABS (A.MINUS.B) function not be selected.

UKMX

Uword constant (K) MultipleXor select field, 6 bits.

00	#8
01	#1
02	#2
03	#3
04	#4
05	SP1CON
06	SP2CON/#0
07	SC
08	(TBD)
09	(TBD)
0A	(TBD)
0B	(TBD)
0C	(TBD)
0D	(TBD)
0E	(TBD)
0F	(TBD)
10	(TBD)
11	(TBD)
12	(TBD)
13	(TBD)
14	(TBD)
15	(TBD)
16	(TBD)
17	(TBD)
18	(TBD)
19	(TBD)
1A	(TBD)
1B	(TBD)
1C	(TBD)
1D	(TBD)
1E	(TBD)
1F	(TBD)
20	(TBD)
21	(TBD)
22	(TBD)
23	(TBD)
24	(TBD)
25	(TBD)

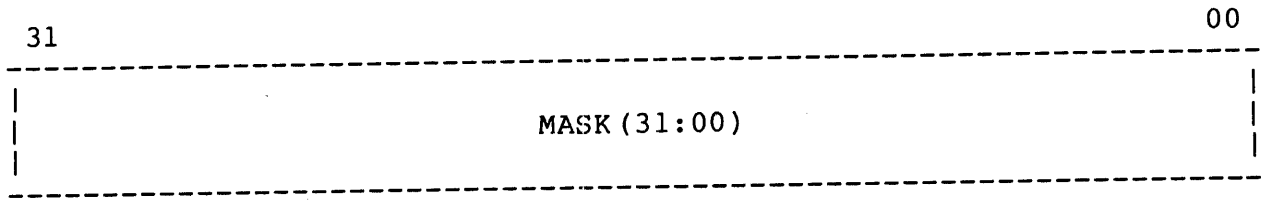
26	(TBD)
27	(TBD)
28	(TBD)
29	(TBD)
2A	(TBD)
2B	(TBD)
2C	(TBD)
2D	(TBD)
2E	(TBD)
2F	(TBD)
30	(TBD)
31	(TBD)
32	(TBD)
33	(TBD)
34	(TBD)
35	(TBD)
36	(TBD)
37	(TBD)
38	(TBD)
39	(TBD)
3A	(TBD)
3B	(TBD)
3C	(TBD)
3D	(TBD)
3E	(TBD)
3F	(TBD)

1.1.6 MASK

Bit MASK generator

The MASK is used to generate a bit pattern which can be used to isolate fields of bits thru use of the logical functions of the ALU. This occurs in the execution of bit field instructions and in the memory management process of translating virtual, to physical addresses when they are not already translated in the TBUF.

MASK data type:



MASK = single 0 bit in a field of ones

MASK CONTROL

The MASK generator is controlled by SC(04:00) which is used to address a bit position in a long word of ones and insert a zero in that position.

The procedure to generate a mask to retain all bits from bit position P and above would involve setting AMX=0, BMX=MASK, SC=P and performing an A.PLUS.B.PLUS.1 ALU operation.

To generate a mask to retain 5 bits from position P the settup would be: AMX=0, BMX=MASK, SC=P.PLUS.S and ALU=A.PLUS.B.PLUS.1. The resultant mask could then be added to the previously acquired mask to isolate the required bit field.

1.1.7 LC and RC

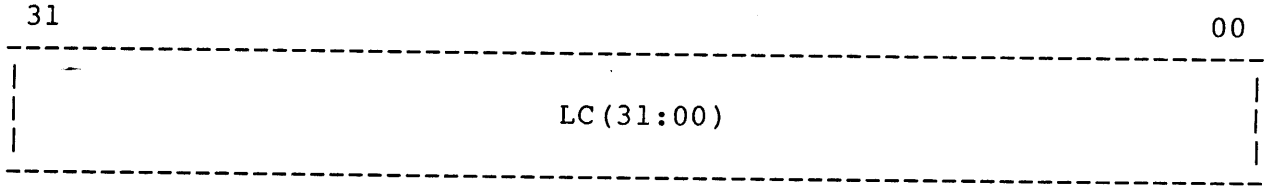
LC - Latch C

RC - Register set C

RC is used as a temporary storage area for addresses and operands generated during the execution of the micro program. There are 16, 32 bit temporary registers available and 1, 32 bit storage latch. The latch, LC, is used to hold the contents of a previously fetched temporary register in RC for use in the ARITHMETIC SECTION.

Generally, the contents of RC are fetched in one micro instruction into LC and then used by the ALU in the next micro instruction. RC may however, be read into LC and used in the same micro instruction if the ALU is selected for logical mode.

LC, RC data type:



LC, RC CONTROL

The loading of LC, the writing of RC, and the RC address are controlled by the USPO field of the Uword. Refer to section 1.3.8 for command code control information.

1.1.8 LA and LB

Latch A and Latch B

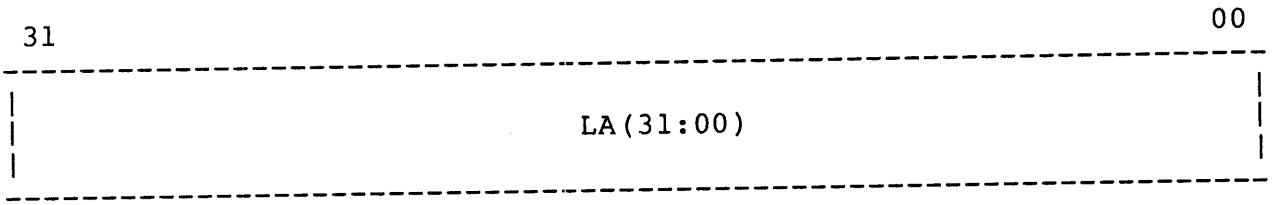
RA and RB

Register set A and Register set B

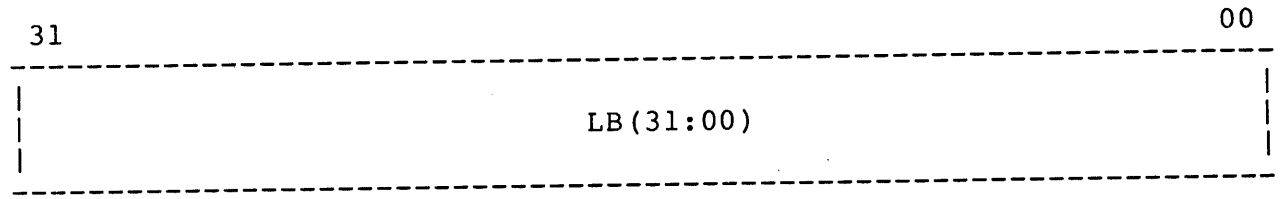
RA and RB combine to form a two address port storage location for the 16 processor GENERAL REGISTERS. These registers are used in the addressing mode evaluations and as fast memory storage by the instruction set. The two port feature allows fast access to both the source register from RB and the destination register from RA in the execution of register to register mode instruction.

LA and LB are used to hold the contents of a general register which had been fetched previously for use in the ARITHMETIC SECTION. Generally the contents of RA and RB are fetched in one micro instruction into LA and LB and then used by the ALU in the next micro instruction. RA and RB may however, be read into LA and LB and used in the same micro instruction if the ALU is selected for logical mode.

LA data type:



LB data type:



SCRATCH PAD CONTROL

The three register sets, RC, RB, and RA, and their associated latches, LC, LB and LA are controlled by a seven bit opcode field of the UWORD designated USPO. This field controls the writing of the scratch pads, the loading of the latches, and the address source of the register.

The address supplied to the RC register set can come from two sources. It can be generated explicitly as a register number (RN) in the USPO field or it may come from the SC register bits 03:00.

The RA and RB sets can be addressed explicitly as a register number (RN) in the USPO field or by an Address Code Number (ACN). The ACN number selects the address for RA and RB from several register fields of the instruction operand specifiers. In 11 compatibility mode the register address comes from either the register field for the source mode/register or destination mode register codes in the instruction opcode.

The ACN for the RA and RB sets also selects SC(03:00) as the address source. This allows the GENERAL REGISTERS to be sequentially indexed.

The USPO field also allows individual control of RC from that of RA and RB. In these cases when one may be read and the other written the contents of RC cannot be interchanged with the contents of RA, RB or vice versa in the same micro-instruction. Those USPO codes are 60 to 7F.

(D)

RC register write operations are a.ways Longword data types.

The RA and RB register write operations are context dependent and is controlled by the UDT uword. In operations requiring Float, Quad or Double Float context Longword data type will be used.

UDT

Uword Data type select field, 2 bits.

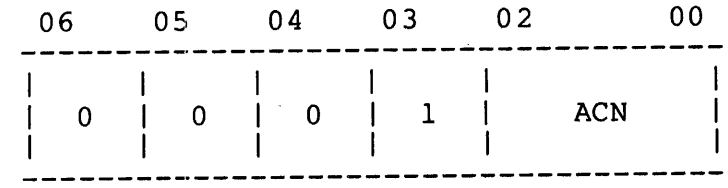
```
0  LONGWORD  :  32 BITS
1  WORD      :  16 BITS
2  BYTE      :  8 BITS
3  -INSTRUCTION DEPENDENT:  Any of above
```

When UDT=3, The Instruction decode logic determines to be used in RA and RB register write operations.

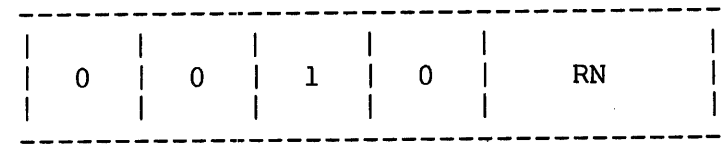
USPO

Uword Scratch Pad Opcode, 7 bits.

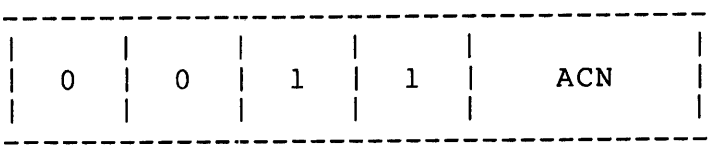
00 to 05	NOOP
06	LOAD LC[SC(03:00)]
07	WRITE RC[SC(03:00)]
08 to 0F	LOAD LA,LB[ACN]



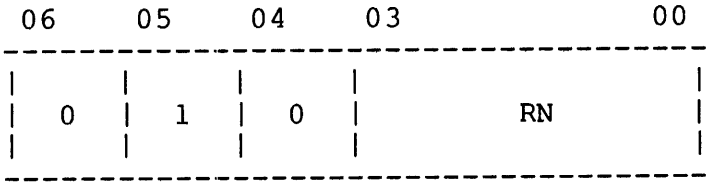
10 to 17 LOAD LA[RN] RN=(7:0)



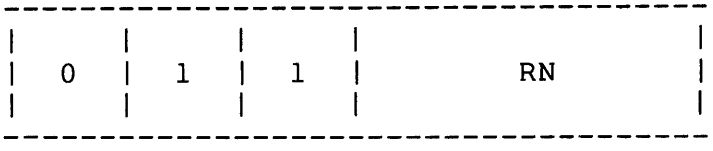
(D) 18 to 1F WRITE RA,RB[ACN]



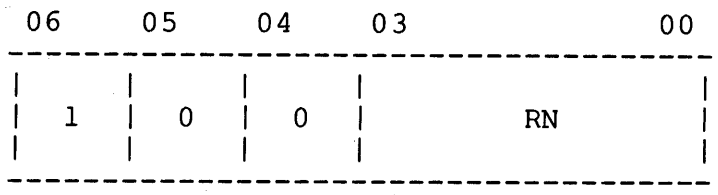
20 to 2F LOAD LC[RN]



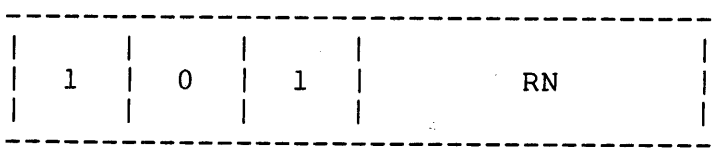
30 to 3F WRITE RC[RN]



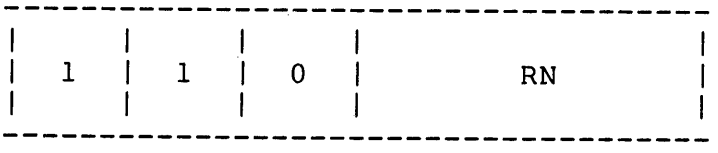
40 to 4F LOAD LA,LB[RN]



50 to 5F WRITE RA,RB[RN]



60 to 6F LOAD LA,LB[R1]
and WRITE RC[RN]



70 to 7F LOAD LC[RN]
and WRITE RA,RB[R1]

The ACN field of USPO has two interpretations, one for VAX mode and one for 11 compatibility mode. In VAX mode there are three register fields in the instruction which are examined. These are SP1R (SPecifier 1 Register), SP2R (SPecifier 2 Register), and PRN (Previous Register Number). SP1R is the register number from the operand specifier currently being examined by the IBUF control logic. SP2R is the register number from the byte following the operand specifier in IBUF and PRN is the register number from the last operand specifier in IBUF.

(D)

In 11 compatibility mode there are two register fields in the instruction which are examined. These are the SRC R (SouRCe Register) and the DST R (DeSTination Register) numbers found in the source and destination mode/register fields. The FPA, Floating Point Accelerator, will keep a copy of the RA register. The FPA will receive the RA address and a two bit encoded write enable field describing the RA write operation data type.

FPA RA Write Control field

0	WORD
1	LONGWORD
2	BYTE
3	NO WRITE

ACN - VAX mode
Address Code Number

(D)	RA address	RB address
0	SP1 R	SP1 R
1	SP2 R	SP2 R
2	SP2 R	SP1 R
3	PRN	PRN
4	PRN.PLUS.1	PRN.PLUS.1
5	SC(03:00)	SC(03:00)
6	SP1 R.PLUS.1	SP1 R.PLUS.1
7	0	0

ACN - 11 compatibility mode

(D)	RA address	RB address
0	SRC R	SRC R
1	DST R	DST R
2	DST R	SRC R
3	SRC R	SRC R
4	SRCR.OR.1	SRCR.OR.1
5	SC(03:00)	SC(03:00)
6	SRC R.PLUS.1	SRC R.PLUS.1
7	0	0

1.1.9 RLOG and PCSV

RLOG - Register LOG stack

PCSV - Program Counter SaVe register

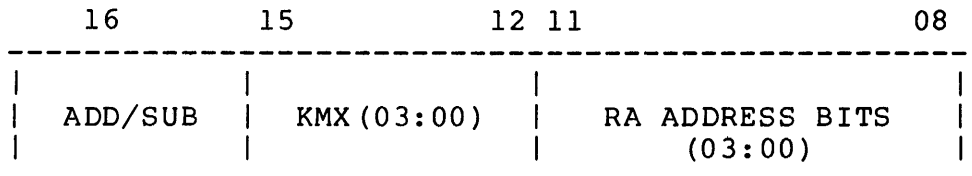
The RLOG stack and PCSV register provide sufficient information to the microcode so that the contents of the general registers may be restored to their original values in order that an instruction may be restarted. If a memory management fault occurs which requires a macro-level trap routine to be run it is necessary to back-up the general registers that had been auto-incremented and auto-decremented during the execution of the instruction causing the fault.

The RLOG stack keeps track of the register numbers and constant values used to update the registers in specifier evaluations. There are 16 locations in the RLOG stack and at each instruction fetch a pointer into the stack is initialized and an RLOG empty flag asserted. When the micro-code fault routine reads the RLOG the pointer is decremented and the next entry in the stack becomes available.

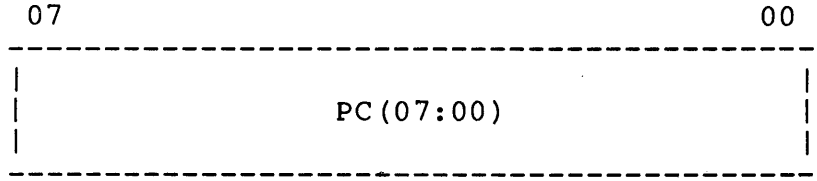
The PCSV register saves the lower 8 bits of the PC at the time an instruction is fetched. Since no instruction is longer than 256 bytes, the entire 32 bit starting PC may be reconstructed if a fault occurs.

RLOG data type:

(D)



PCSV data type:



RLOG CONTROL

The RLOG stack is written when the UALU field specifies an RLOG update operation. If the operation is A.PLUS.B (RLOG UPDATE), RLOG08 is set to a one, otherwise it is set to zero.

Whenever the UMSC field selects the Read RLOG function, the current value is read out of the stack and the pointer is incremented at the end of the micro-instruction.

PCSV CONTROL

Each time an instruction is fetched, the PCSV register gets the low 8 bits of the PC.

1.2 EXPONENT SECTION

The EXPONENT SECTION is used to perform exponent processing in parallel with fraction processing in the ARITHMETIC and DATA SECTIONS of the Data Path when floating point instructions are being executed. When processing exponents the 10 bit exponent path is interpreted as an 8 bit exponent and a 2 bit overflow/underflow code.

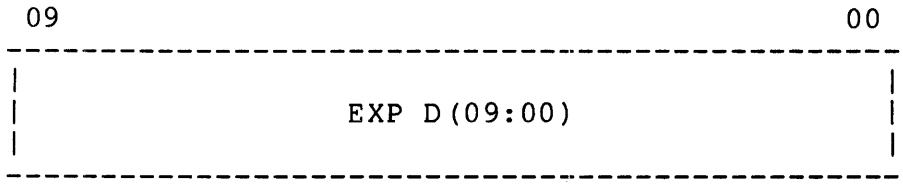
The EXPONENT SECTION is also used for controlling the SC register when it is used to generate masks, to address RA, RB, or RC, to address processor registers, or to be used as a shift value in the DATA SECTION.

1.2.1 EALU

Exponent Arithmetic and Logic Unit

The EALU performs the processing of data in the EXPONENT SECTION. It consists of an ALU circuit with fast carry look ahead, a negative absolute value look up rom and a multiplexor. The rom and multiplexor are used in NABS (A.MINUS.B) mode to provide a shift value in floating point arithmetic alignment.

EALU data type:



EALU CONTROL

The EALU function is controlled by the UEALU field of the UWORD and is defined as arithmetic or logical. The only restriction on EALU source data is that the NABS (A.MINUS.B) function not be used when a slow constant is being used from KMX unless the proper set-up time has been met as described in section 1.3.5.

UEALU

Uword Exp. Arithmetic and Logic Unit control field, 3 bits.

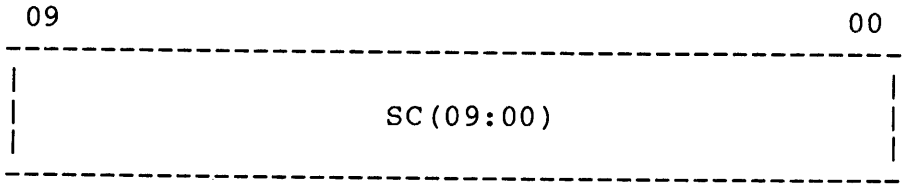
(D)	0	A
	1	A.OR.B
	2	A.AND.B
	3	B
	4	A.PLUS.B
	5	A.MINUS.B
	6	A.PLUS.1
	7	NABS(A.MINUS.B)

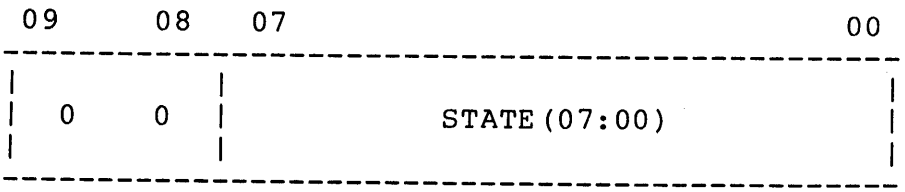
1.2.2 EAMX

Ealu A input MultipleXor

The EAMX provides the data source for the A input to the EALU. Whenever the STATE register is selected as the data source, the STATE register is loaded in the same micro instruction.

EAMX data types:





EAMX CONTROL

The EAMX input is selected by the UMSC field of the Uword. Whenever the UMSC field selects the LOAD STATE REG code the EAMX is switched so that the STATE REG is fed to the input of the EALU. At the end of that micro instruction, the STATE register is loaded. At all other times the SC register is selected at the EAMX.

1.2.3 EBMX

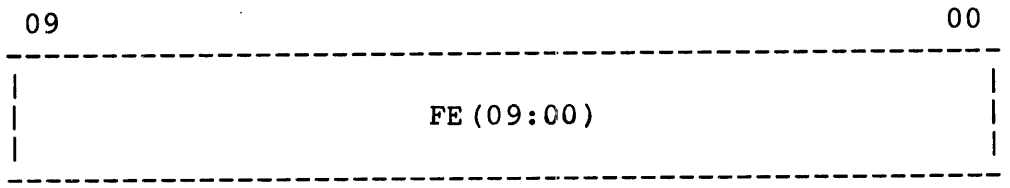
Ealu B input MultipleXor

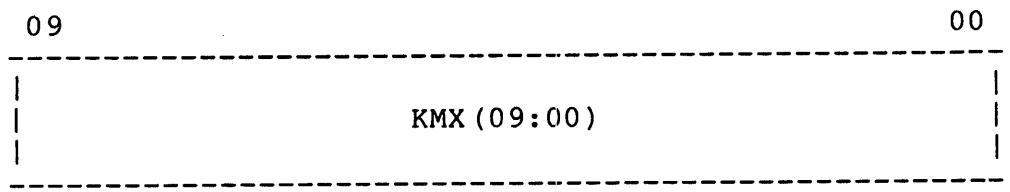
The EBMX provides the data source for the B input to the EALU. The EBMX receives either the exponent from the FE register or the exponent field from the AMX when the EXPONENT SECTION is used for exponents processing.

Constants from the KMX or shift values from the control logic of the DAL are used as data sources for the EBMX so that a shift count in SC may be operated on. Constants may also be selected to set or clear flags in STATE.

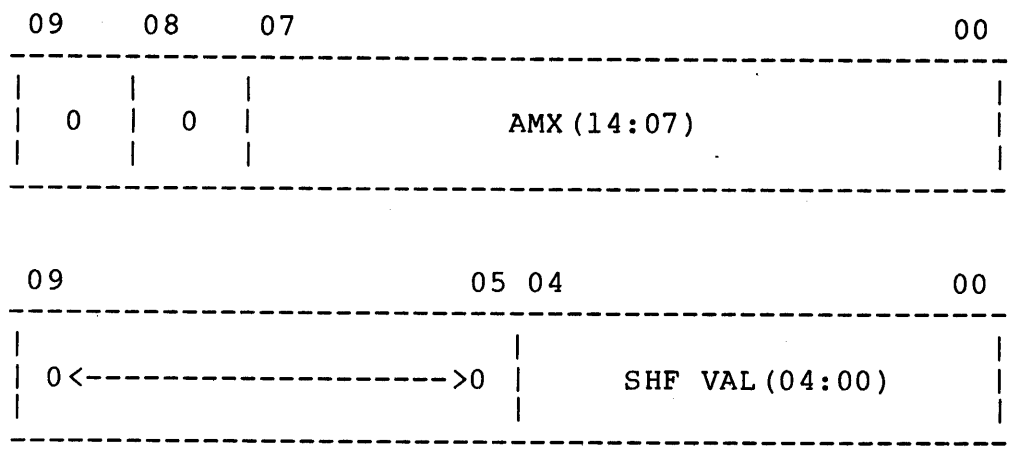
The SHF VAL is a hardware generated number of left shifts necessary to normalize the contents of the D register. The normalized number is obtained by shifting to the left of the most significant "1" in the D register to the BIT 31 position. This process can be accomplished in one machine cycle with the UDK uword = D enabling the D register to be updated with the normalized number.

EBMX data types:





(D)



EBMX CONTROL

The EBMX is controlled by the UEBMX field of the UWORD.

UEBMX

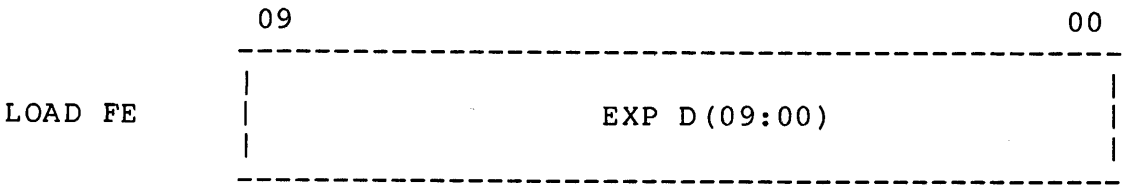
Uword Ealu B input Multiplexor control field, 2 bits.
(D) 0 FE
1 KMX
2 AMX EXP
3 SHF VAL

1.2.4 FE

Floating Exponent register

The FE is used to hold exponents or temporary values for processing in the EXPONENT SECTION.

FE data type:



FE CONTROL

The loading of the FE register is controlled by the UFEK field of the UWORD.

UFEK

Uword Floating Exp. control (K) field, 1 bit.

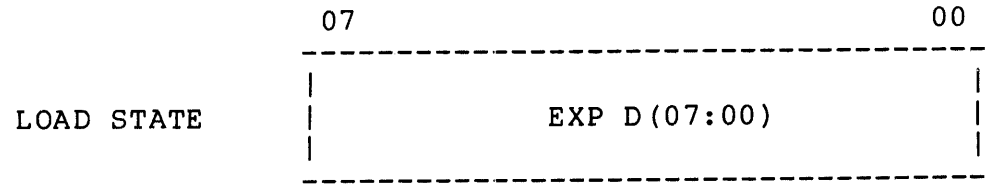
0	HOLD
1	LOAD

1.2.5 STATE

STATE register

The STATE register contains 8 flag bits which are generated by the micro program to be used in program flow control. Each four bit group of the STATE register is used as a 16 way branch condition in the micro sequencer section of the CPU. By using the logical operations of the EALU and the constants from KMX a micro instruction may set or clear individual bits in the STATE register.

STATE data type:



STATE CONTROL

The loading of the STATE register is controlled by selecting the LOAD STATE REG of the UMSC field of the UWORD.

1.2.6 SMX

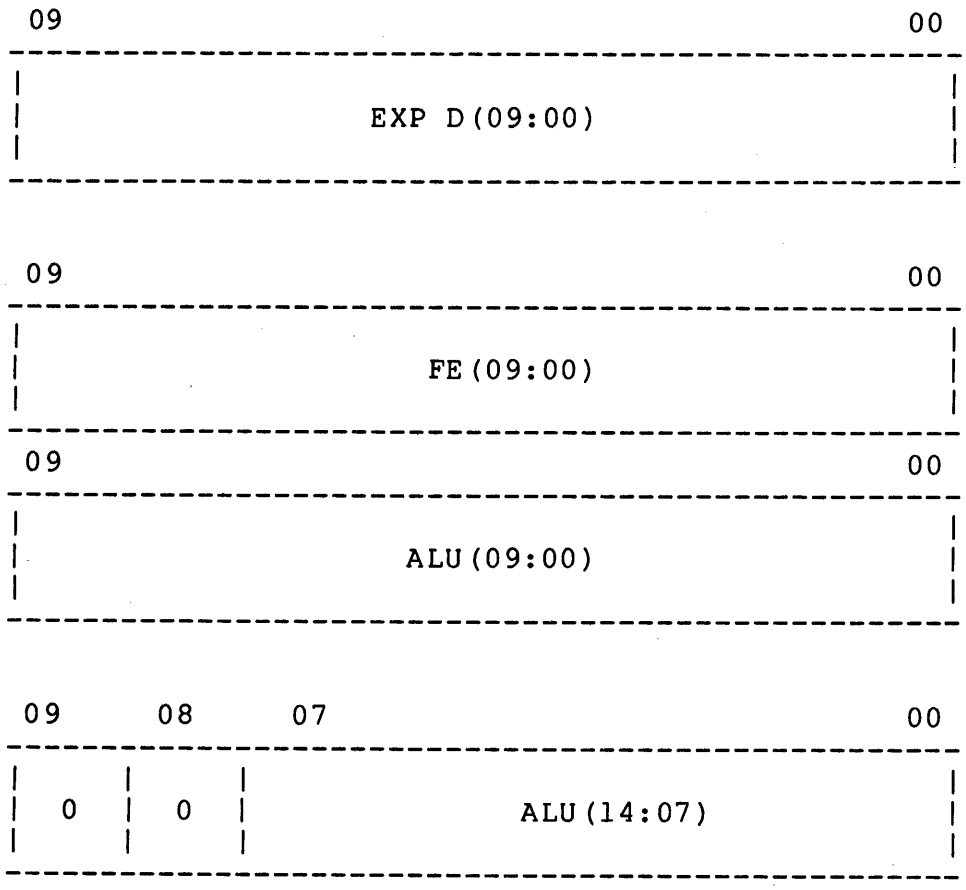
Shift count MultipleXor

The SMX provides a data link from the ARITHMETIC SECTION into the EXPONENT SECTION and allows either a 10 bit data field or an 8 bit exponent to be routed to the SC register from the ALU.

The SMX also has FE as a data source so that the values in FE and SC can be swapped in a single micro instruction.

The EALU data source of the SMX allows processing of the SC register in the EALU for incrementing or decrementing values in the SC.

SMX data types:



SMX CONTROL

The SMX data type is selected by the USMX field of the UWORD.

USMX

Uword Shift count MultipleXor control field, 2 bits.

0	EALU
1	FE
2	ALU
3	ALU EXP

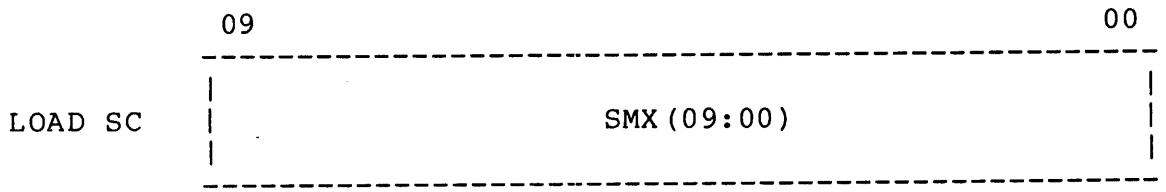
1.2.7 SC

Shift Count register

The SC is used extensively in the CPU for different control functions. The ID BUS control section uses SC(05:00) to address an internal processor register word. The DATA SECTION of the Data Path uses SC(09,04:00) to control the shift amount in the DAL. The ARITHMETIC SECTION uses SC(04:00) to generate a bit mask and SC(03:00) to form a register address in the RA, RB, and RC register sets.

The SC is also used to store exponents in the EXPONENT SECTION.

SC data type:



SC CONTROL

The loading of SC is controlled by the USCK field of the UWORD.

USCK

Uword Shift Count control (K) field, 1 bit.

0	HOLD
1	LOAD

1.3 DATA SECTION

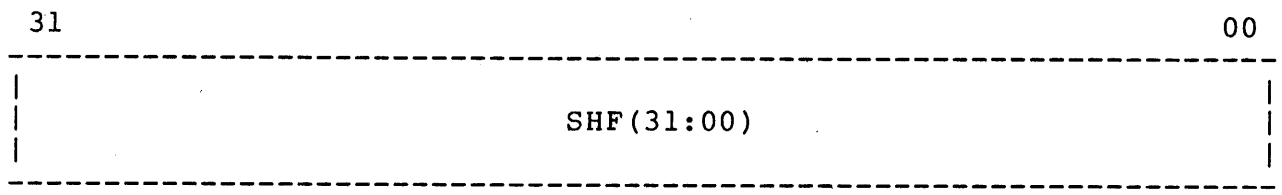
The DATA SECTION controls the shifting, the byte alignment, the unpacking of floating data types, and the moving of data to and from the DATA CACHE or ID BUS interface.

1.3.1 DFMX

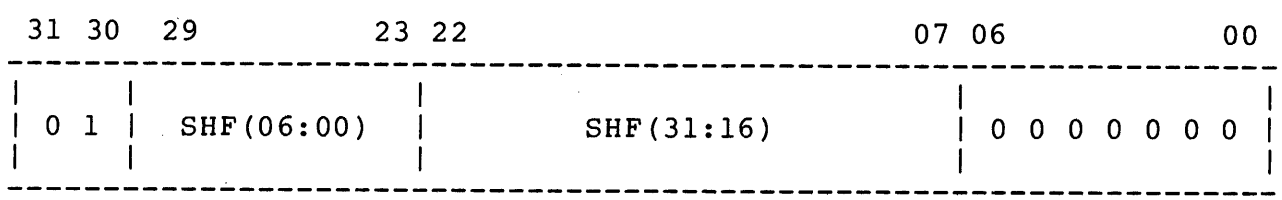
Data Format MultipleXor

The DFMX is the input link from the ARITHMETIC to the DATA section and generates data types in either integer or unpacked floating formats. In unpacked floating format the sign and exponent fields are stripped off and the fraction bits assembled in the correct sequence with zero guard bits and the hidden one inserted.

Integer format:



Unpacked floating format:



DFMX CONTROL

The DFMX formats are controlled by the UQK and UDK fields of the UWORD. If either control field calls for unpacked floating data the DFMX will select that format, otherwise integer format will be used. Refer to sections 1.5.5 and 1.5.6 for control codes.

1.3.1.1 BUS DFMX -

This is a tri-state bus that provides a data link from the Arithmetic Section, DFMX, or from the Floating Point Accelerator to the Data Section.

The Accelerator uses this path to sample RA scratch pad write data or route data into the Datapath via the D register or Q register.

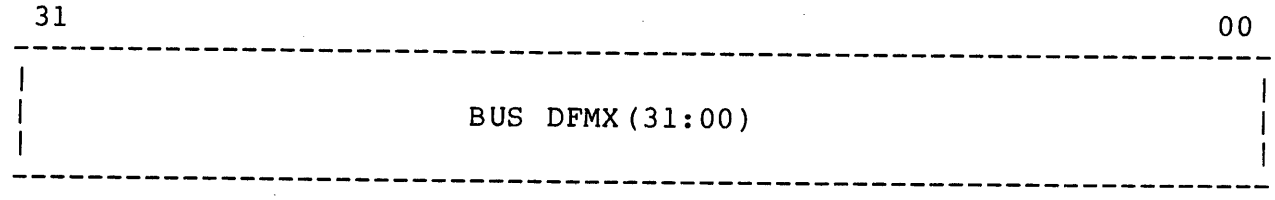
Tri-state control of the Bus DFMX is accomplished by programming the UDK or UQK uword fields. Accelerator control is made by selecting UDK=A or UQK=B. Arithmetic section control, DFMX, is the default programming of the above uword codes.

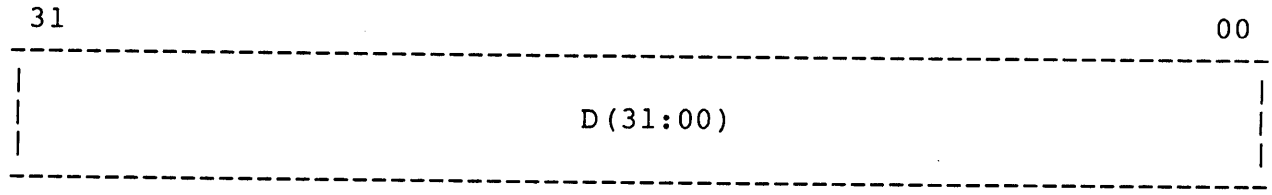
1.3.2 QMX

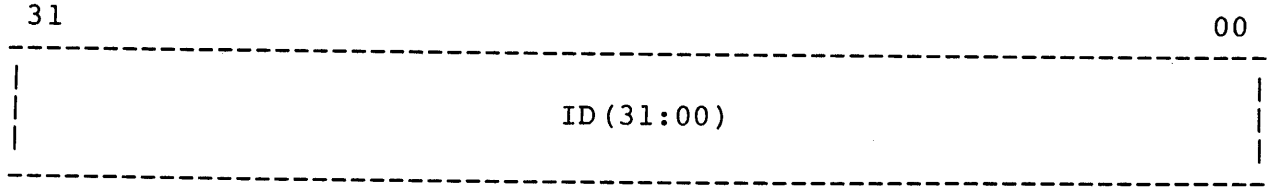
Q register MultipleXor

The QMX provides paths for loading the Q register from either the ARITHMETIC SECTION via the DFMX, from the Floating Point Accelerator, from the D register such that the ARITHMETIC SECTION may be simultaneously used for other operations; from the ID Bus or with the generation of a constant of six in each 4 Bit NIBBLE for use in decimal arithmetic. Each NIBBLE X constant is generated by the lack of an ALU byte X carry.

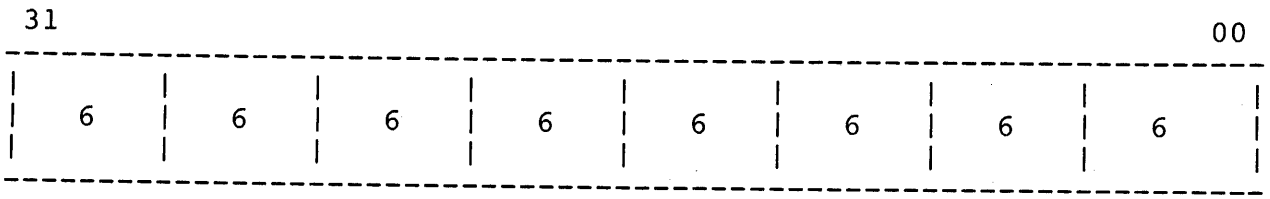
QMX data types:







(D)



QMX CONTROL

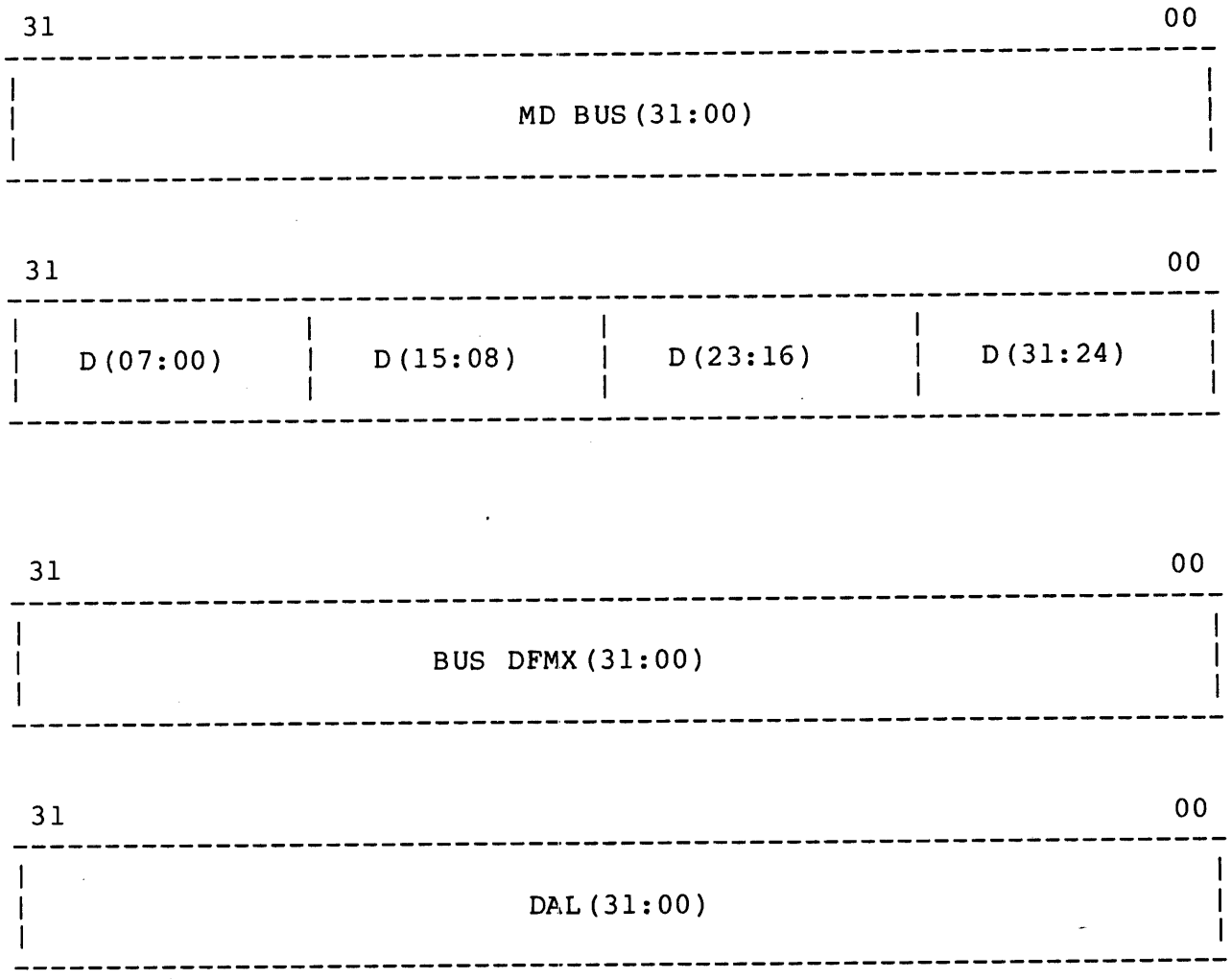
The QMX data type is selected from the UQK field of the UWORD.

1.3.3 DMX

D register MultipleXor

The DMX provides the necessary paths for loading the D register from the ARITHMETIC section via the DFMX, from the Floating Point Accelerator, the CACHE MEMORY interface via the MD BUS, distinct sections of the total CPU and from the DAL.

DMX data types:



DMX CONTROL

The DMX data types are selected from the UDK field of the UWORD. Refer to section 1.5.6 for control codes.

1.3.4 DAL

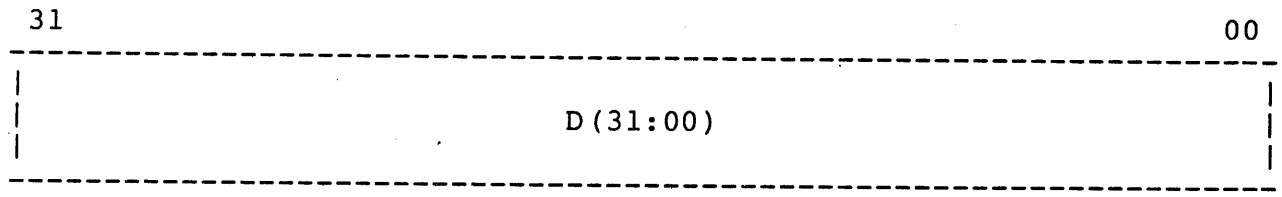
Data ALignment

The DAL allows the D register contents to be shifted a maximum of 32 positions to the right with the least significant end of the Q register shifted in or 31 positions to the left with the most significant end of Q shifted in. Positive shift numbers will cause left shifting, negative shift numbers (two's complement notation) will cause right shifting, negative zero value will cause a 32 bit shift (Q register), and positive zero will result in D unshifted.

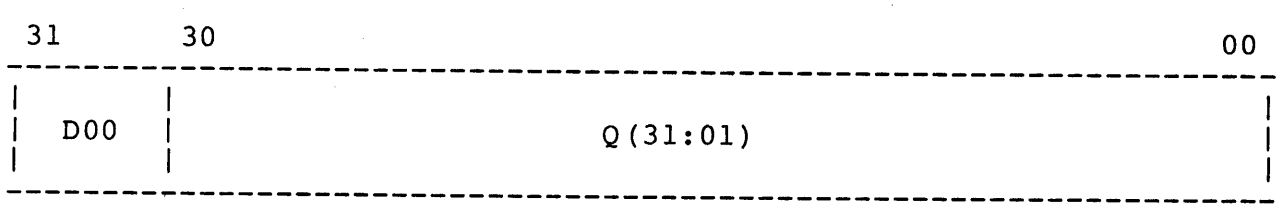
The DAL is used to perform the shifting operations required in the execution of bit field, multiply, divide, shift and decimal arithmetic instructions. It is also used to isolate bit fields in the virtual to physical address translation process.

DAL shift range:

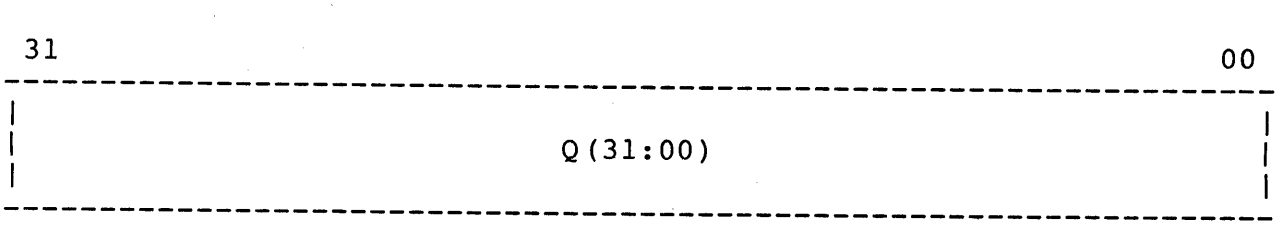
Shift amount = 000000 (NO SHIFT)



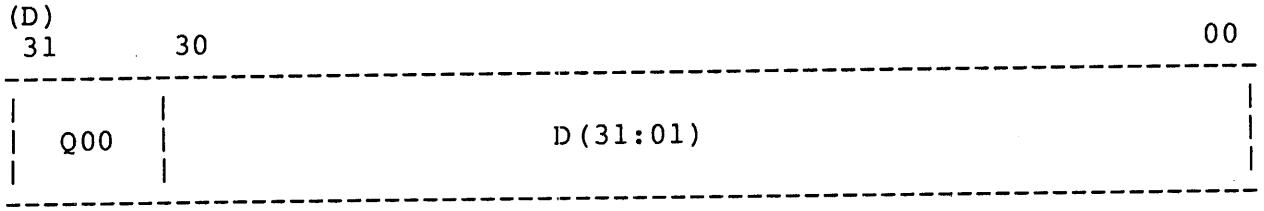
Shift amount = 011111 (LEFT 31)



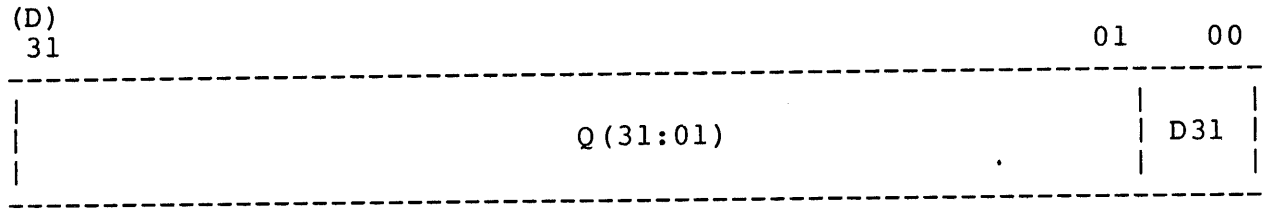
Shift amount - 100000 (RIGHT32, Q)



Shift amount = 111111 (RIGHT 1)



Shift amount = 100001 (RIGHT 31)



DAL CONTROL

The DAL with the shift amount selected by either the contents of SC or by a shift value determined by a hardware look-up may be specified by the UDK field of the UWORD.

When the UDK field specifies that the D register be loaded with the contents of the Q register the DAL is selected for a RIGHT shift of 32 and the DMX selects the DAL. Refer to section 1.5.6 for control codes.

1.3.5 Q

Q register

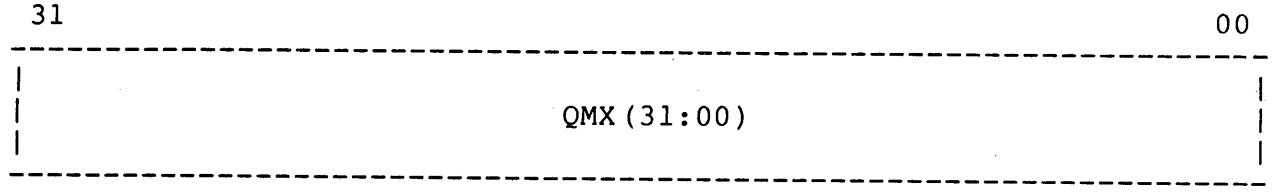
The major function of the Q register is to be used with the D register to hold data structures which are larger than 32 bits. This occurs in the execution of field and double floating instructions.

The Q register also is used to hold the multiplier and quotient bits in the execution of the multiply and divide instructions.

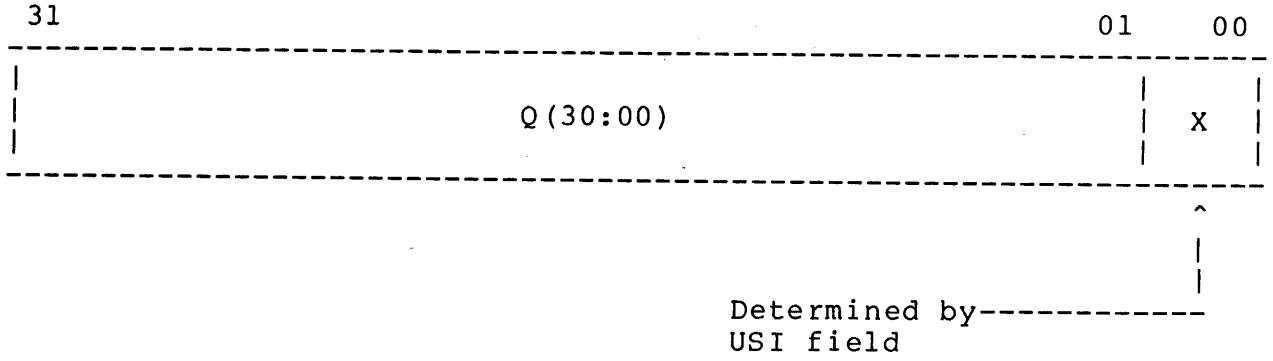
During evaluation of instruction operands the Q register is used to hold the first operand while the next one is being evaluated. In general, Q is used as a temporary storage location for data generated in the ARITHMETIC SECTION.

Q register data types:

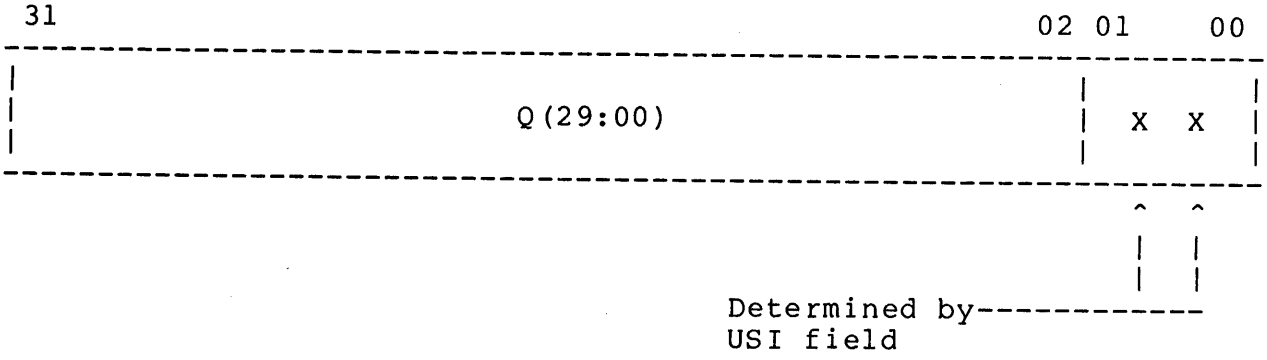
Load Q

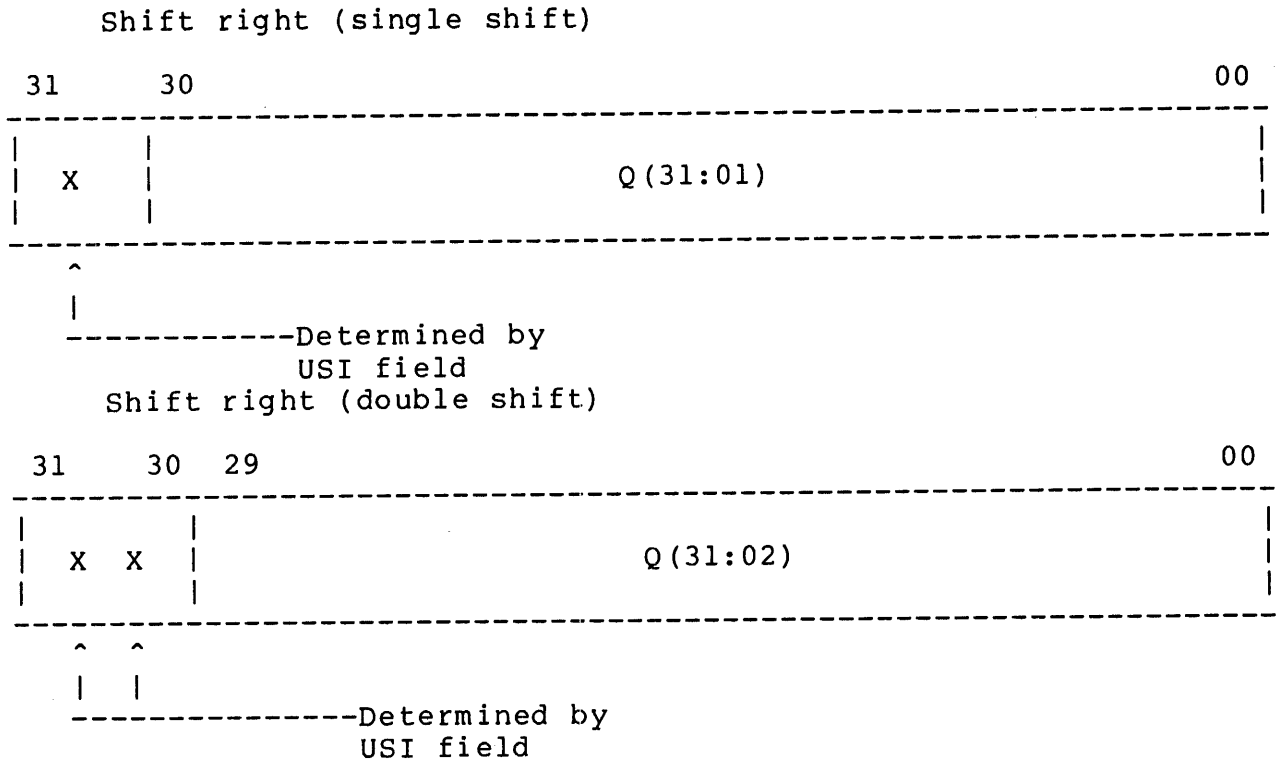


Shift left (single shift)



Shift left (double shift)





Q CONTROL

The Q register loading and shifting is controlled from the UQK field of the UWORD with shift inputs selected by the USI field.

UQK

Uword Q register control (K) field, 4 bits.

0	HOLD
1	DOUBLE SHIFT LEFT
2	DOUBLE SHIFT RIGHT
3	Reserved
4	Reserved
5	SINGLE SHIFT LEFT
6	SINGLE SHIFT RIGHT
7	Reserved
8	LOAD SHF (Integer format)
9	LOAD SHF (Unpacked floating format)
A	LOAD Decimal Const (NIBBLE ALU carry dep)
B	LOAD ACC DATA
C	LOAD D
D	Reserved
E	LOAD ID BUS
F	LOAD ZEROES

The shift input to the Q register is selected by the USI field.

USI - Q input

Uword Shift Input control field, 3 bits.

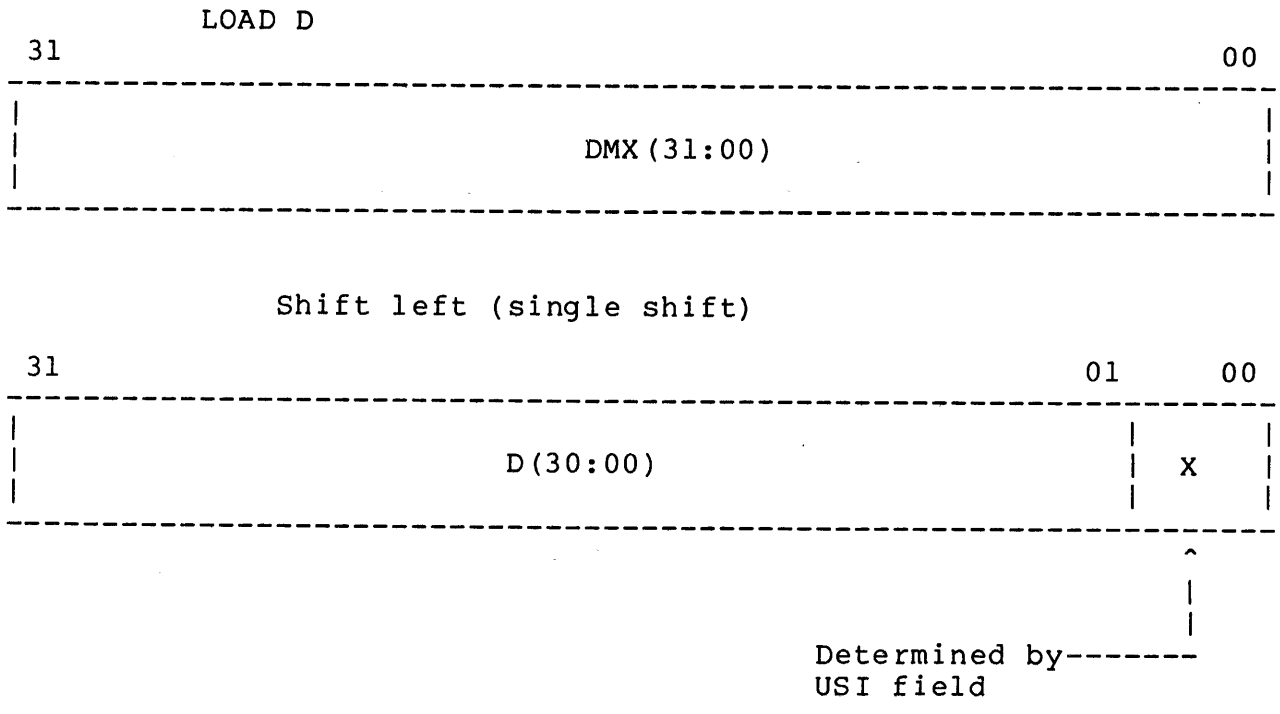
(D)	0	ALU CARRY 31 (to be used for single shift only)
	1	Q31
	2	D31
	3	0
	4	0
	5	ALU CARRY 31 (To be used for single shift only)
	6	0
	7	1

1.3.6 D

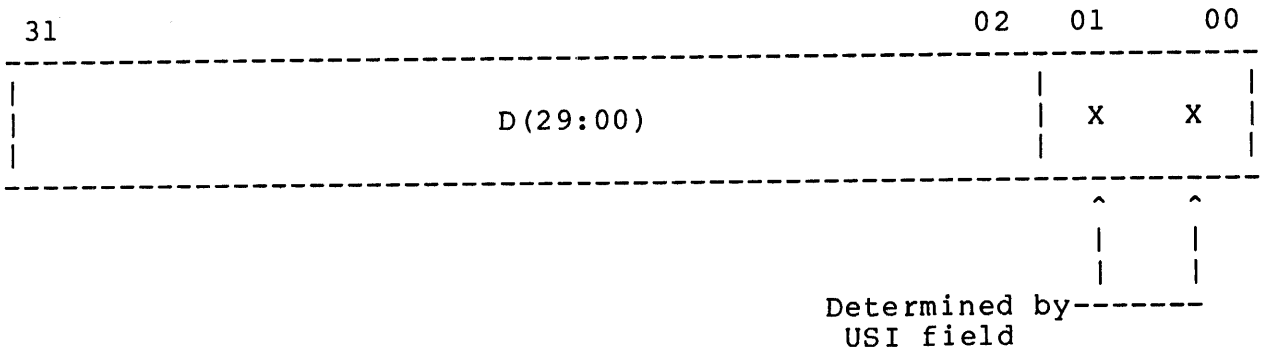
D register

The D register acts primarily as the data interface connection between the DATA PATH and Memory or the DATA PATH and remotely located sections of the CPU and FPA. When used for ID BUS write transfers, odd parity is generated on a per byte basis. Parity on data received from the ID is not checked.

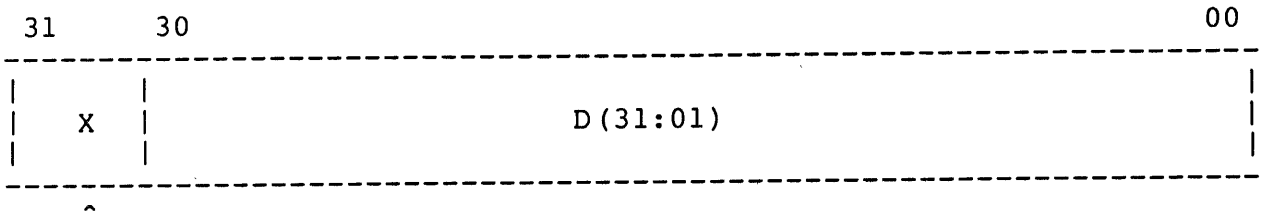
The D register is used in conjunction with the Q register to hold data structures larger than 32 bits. It may also be used as a temporary storage location for data generated in the ARITHMETIC SECTIO.. D register data types:



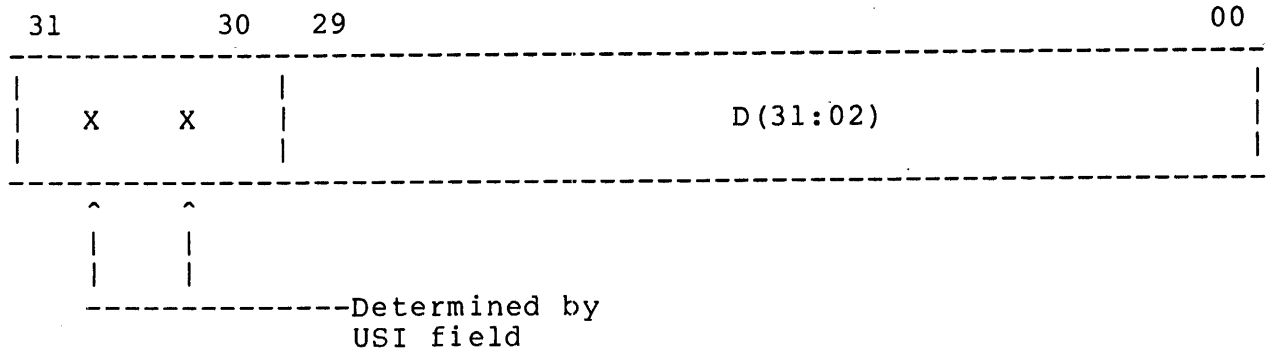
Shift left (double shift)



Shift right (single shift)



Shift right (double shift)



D CONTROL

The D register loading and shifting is controlled from the UDK field of the UWORD with shift inputs selected by the USI field.

UDK

Uword D register control (K) field, 4 bits.

0	HOLD
1	DOUBLE SHIFT LEFT
2	DOUBLE SHIFT RIGHT
3	Reserved
4	SINGLE SHIFT LEFT if .NOT, ALU CARRY.ELSE LOAD SHF(INT FORM).
5	SINGLE SHIFT LEFT
6	SINGLE SHIFT RIGHT
7	Reserved
8	LOAD SHF (Integer format)
9	LOAD SHF (Unpacked floating format)
A	LOAD ACC DATA
B	LOAD D NIBBLE SWAP
C	LOAD Q
D	LOAD D (Shifted by SC (09, 04:00))
E	LOAD D (Shifted by SHF VAL)
F	LOAD ZEROES

(D)

The shift input to the D register is selected by the USI field. The CACHE and SBI Subsystem can load the D register only when the UDK uword = 0.

USI - D input

Uword Shift Input control field, 3 bits.

0	Q31
1	Q00
2	0
3	0
4	0
5	Q31
6	SAVED ALU01/ALU00
7	SAVED ALU01/ALU00

If UDK=D the D register is shifted in the DAL by the value specified by SC09 and SC(04:00).

For UDK=E the D register is shifted in the DAL by a value determined by a hardware lookup table. This is used for the normalization of fractions and is completed in 1 machine cycle.

When UDK=C, the DAL is selected for a shift of 32 which produces the Q register on its outputs.

When USI=6 or 7 and the D register is selected for a double shift, ALU00 is shifted into D on the first shift and ALU01 on the second shift of the machine cycle. If a single shift has been selected, ALU01 will be shifted in.

If a memory read operation is specified by the UDK fields of the UWORD the D or Q register is loaded with the contents of the Memory Data Bus. Data is stored in memory or byte boundaries but accessed on the MD BUS on a long word boundary. To load the D register with the correct data alignment a right shifting procedure is done by the MDBAL.

1.3.6.1 MDBAL -

Memory Data Byte Alignment

The MDBAL provides the correct data alignment from the MD BUS for use in the Data Path.

MDBAL CONTROL

The right shifting process is controlled by two bits of the Virtual Address register, VA(01:00). These two bits specify the least significant byte position of a memory read access and the number of right shifting of bytes required for loading into the D Register.

VA(01:00)	MD Bus shift right positions
0 0	SHF[R0]*
0 1	SHF[R8]
1 0	SHF[R16]
1 1	SHF[R24]

If the memory control field of the UWORD specifies a memory read operation, the D register is loaded with the contents of the MD BUS. If the data length requires a second memory reference due to the starting byte of the Longword address being read, a mask is generated to selectively enable the loading of bytes into the register in order to assemble the entire data type being read. Data on the MD BUS will be rotated according to the byte address used for the reference.

---

* [RX] x being equal to the bit positions shifted right.

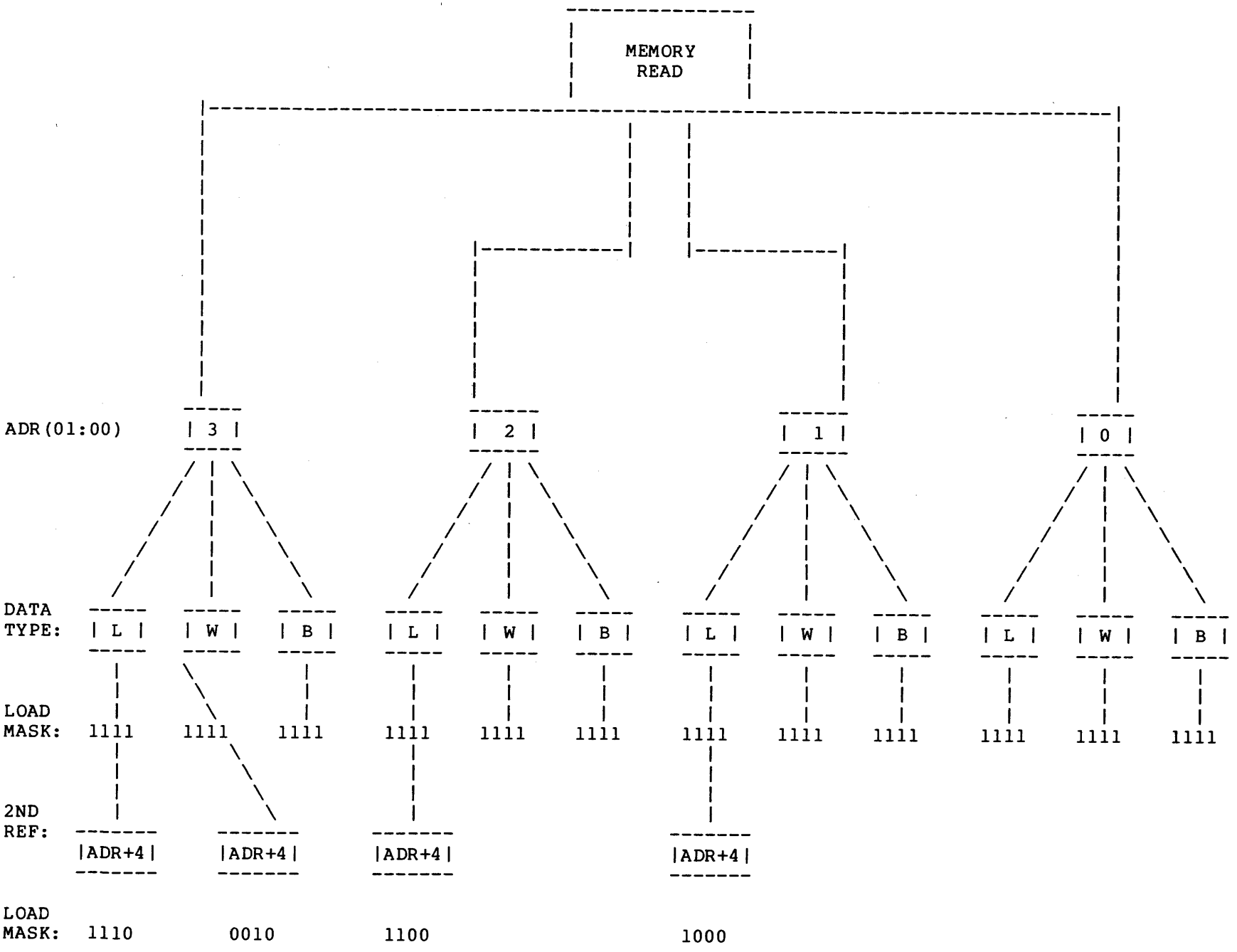

MEMORY
READ
ADR(01:00)
3
2
1
0
DATA
TYPE:
L
W
B
LOAD
MASK:
1111
2ND
REF:
ADR+4
LOAD
MASK:
1110
0010
1100
1000

Figure 1-2

	ADR 01	00	
MD BUS DATA:	0	0	BYTE 3/7 | BYTE 2/6 | BYTE 1/5 | BYTE 0/4
	0	1	BYTE 0/4 | BYTE 3/7 | BYTE 2/6 | BYTE 1/5
	1	0	BYTE 1/5 | BYTE 0/4 | BYTE 3/7 | BYTE 2/6
	1	1	BYTE 2/6 | BYTE 1/5 | BYTE 0/4 | BYTE 3/7

1.3.7 D PGEN

D reg Parity GENerator

The D PCGEN circuit generates one parity bit for each byte in the D register to be transmitted along with data on the IB BUS. Odd parity is generated.

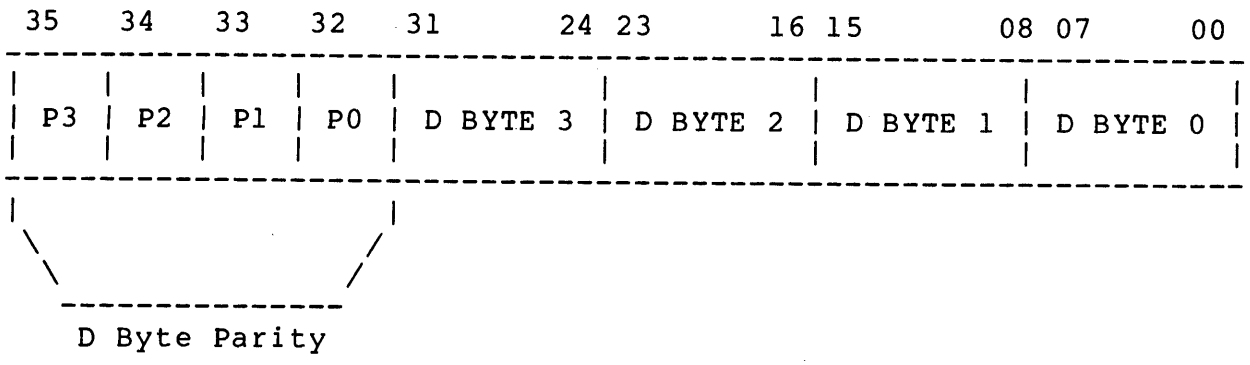
1.3.8 BAL

Byte ALignment

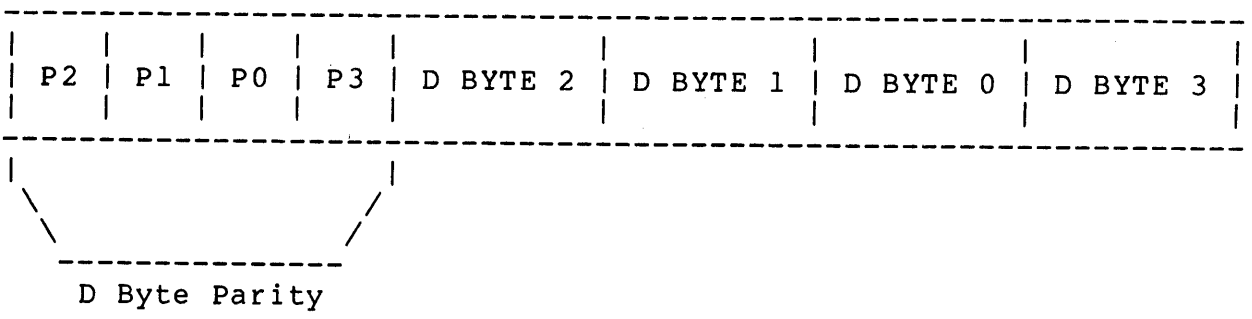
The BAL is used to rotate the contents of the D register so that the bytes are aligned with the byte address in which data is to be written. A byte mask is generated by the hardware to control which bytes in a long word address are to be written into memory or as an indication as to what bytes are being read.

BAL data types:

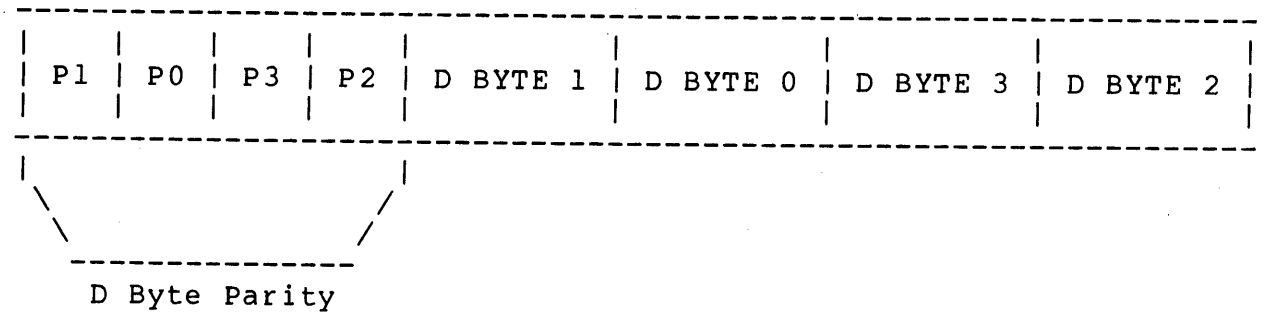
ADR(01,00) = 0



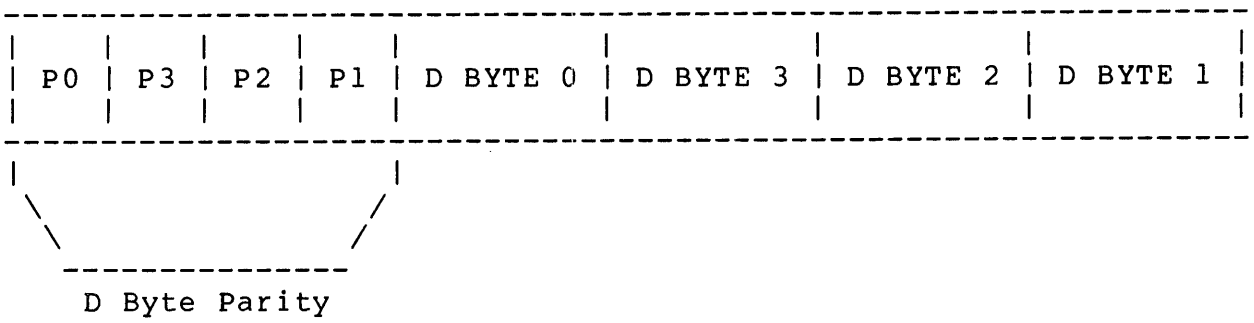
ADR(01,00) = 1



ADR(01,00) = 2



ADR(01,00) = 3



BAL CONTROL

The byte shift value is selected by the ADR address bits 01 and 00, and the BAL output drivers are turned on by the memory control logic in the Data Cache control.

A four bit byte mask is generated to enable the writing of individual bytes into memory. For long word data types at ADR(01:00) = 1, 2, or 3 or word data types of ADR(01:00) = 2 or 3 a second memory reference must be performed to complete the data write operation, in which case a second byte mask must be generated. A code in the UMSC field of the UWORD specifies the second reference of this data alignment procedure.

BYTE MASK field is also generated during Memory read operations.

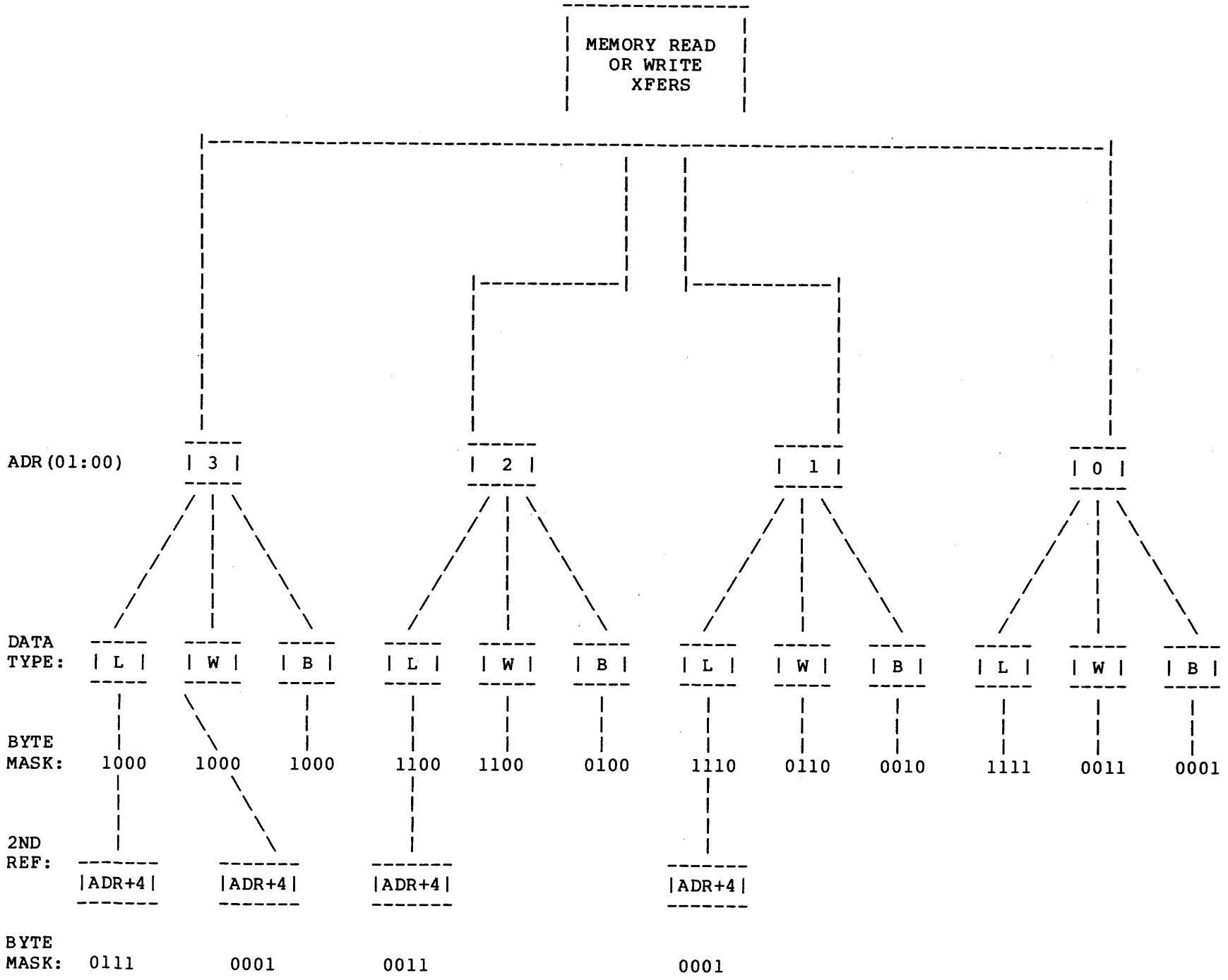

MEMORY READ OR WRITE XFERS
ADR(01:00)
3
2
1
0
DATA TYPE:
L
W
B
BYTE MASK:
1000
1000
1000
1100
1100
0100
1110
0110
0010
1111
0011
0001
2ND REF:
ADR+4
ADR+4
ADR+4
ADR+4
BYTE MASK:
0111
0001
0011
0001

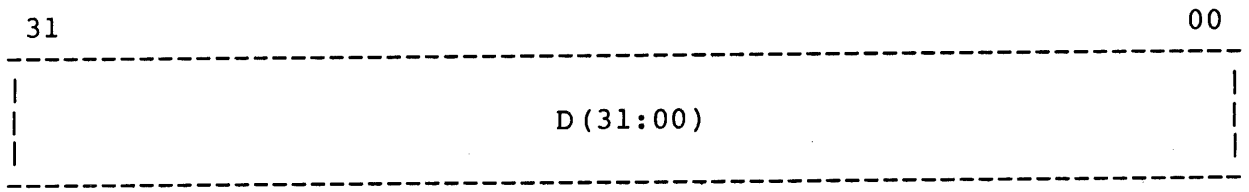
Figure 1-3

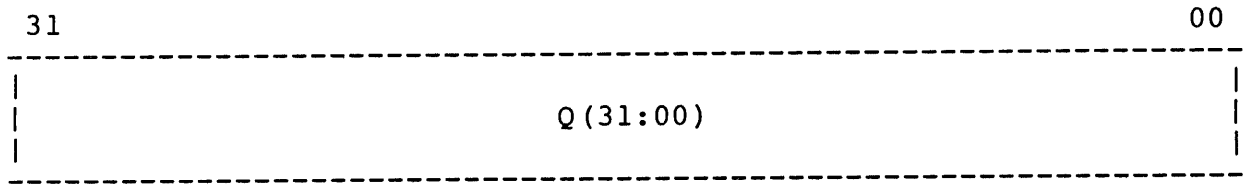
1.3.9 RAMX

Register Amx MultipleXor

The RAMX is a link for the DATA SECTION to provide data to the ALU A input of the ARITHMETIC SECTION.

RAMX data types:



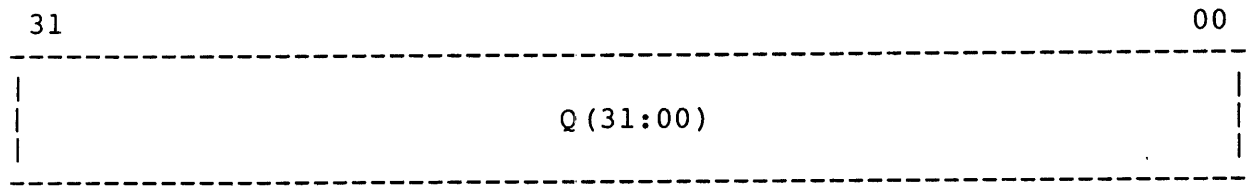


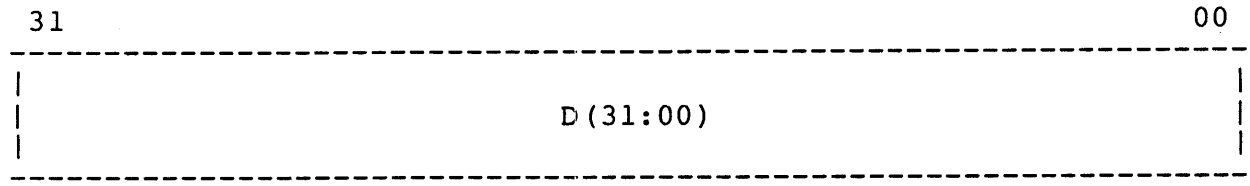
1.3.9.1 RBMX -

Register Bmx MultipleXor.

The RBMX is a link for the DATA SECTION to provide data to the ALU B input of the ARITHMETIC SECTION.

RBMX data types:





RAMX and RBMX Control

The data type of the RAMX and RBMX is selected by the URMX field of the UWORD.

URMX

Uword Register MultipleXor control field, 1 bit.

	RAMX	RBMX
0	D	Q
1	Q	D

1.4 ADDRESS SECTION

The ADDRESS SECTION of the Data Path is used to keep the virtual address for operand references, the program counter, and the instruction buffer address.

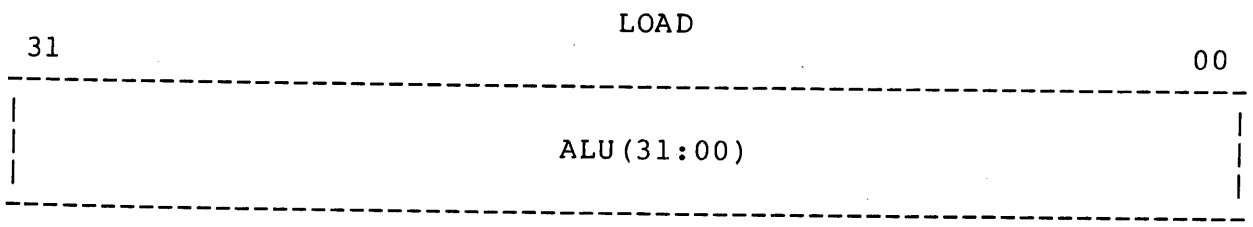
Each address register has counting ability so that the commonly used address arithmetic performed does not require use of the ARITHMETIC SECTION.

1.4.1 VIBA

Virtual Instruction Buffer Address counter

The VIBA holds the address for the instruction stream data being fetched by the instruction buffer (IBUF) control logic.

VIBA data type:



VIBA CONTROL

The loading of the VIBA is selected by a control code in the UIBC field of the UWORD. This takes place whenever the instruction execution changes sequence such as in the case of JUMP and successful

BRANCH instructions or as in the initiation of a trap or interrupt routine.

The IBUF control logic will increment the VIBA (by 4) whenever it has successfully fetched instruction data.

1.4.2 VA

Virtual Address counter

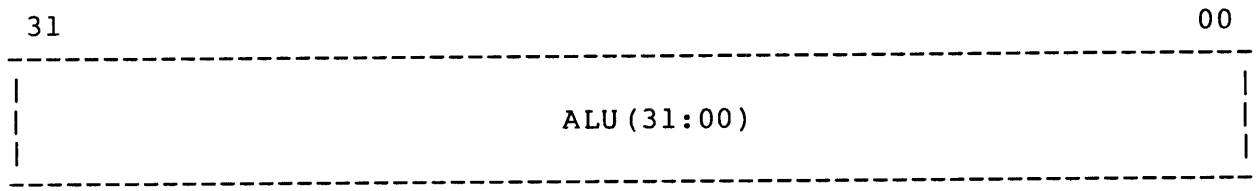
The VA holds the address which the micro program generates in the data path to read or write a memory location. The VA will primarily contain a virtual address which must be converted to a physical memory address by the TRANSLATION BUFFER.

At other times the VA may hold a physical memory address which has been generated by the micro program during the translation process or when the memory management mechanisms are turned off.

The VA may also be used to index into the TBUF whenever the TBUF is being updated or invalidated by the micro program. During the execution of the PROBE instruction the indexing is used to determine if an access violation would occur if the memory reference to that virtual address was actually performed.

The VA also provides a load path from the ARITHMETIC SECTION into the PC.

VA data type:



VA CONTROL

The loading of the VA is controlled by the UVAK field of the UWORD. A command code in the UACK field specifies that the VA counter be incremented by four. The load operation will override the incrementation if both functions are selected simultaneously.

UVAK

Uword Virtual Address control (K) field, 1 bit.

0	HOLD
1	LOAD

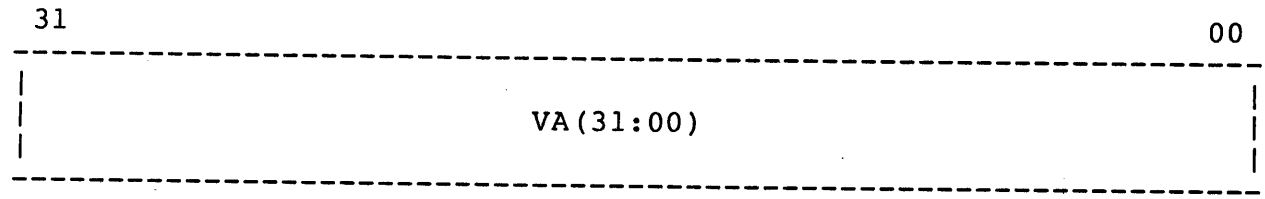
1.4.3 VAMUX

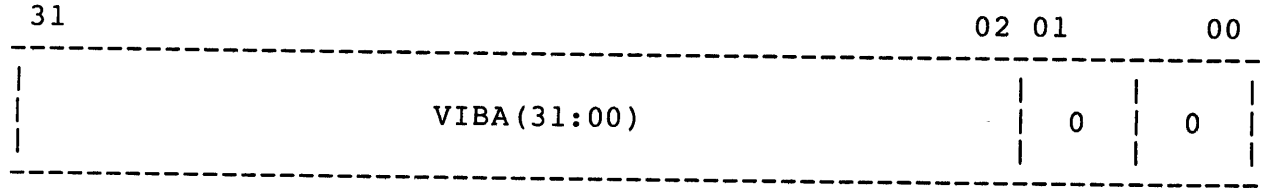
Virtual Address MUltipleXor

The VAMUX address interface logic formats the address into either the vax address format or the 11 compatibility format and selects either the VA or the VIBA as the address source.

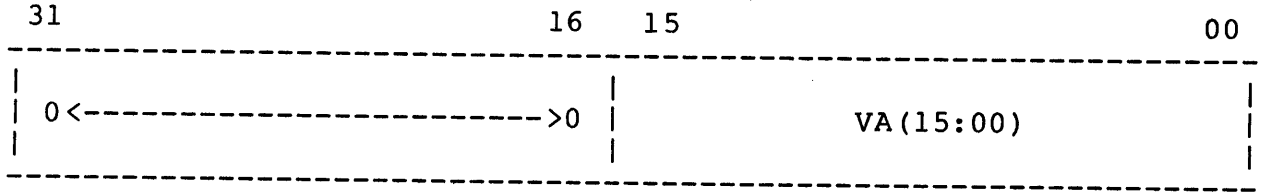
VAMUX formats:

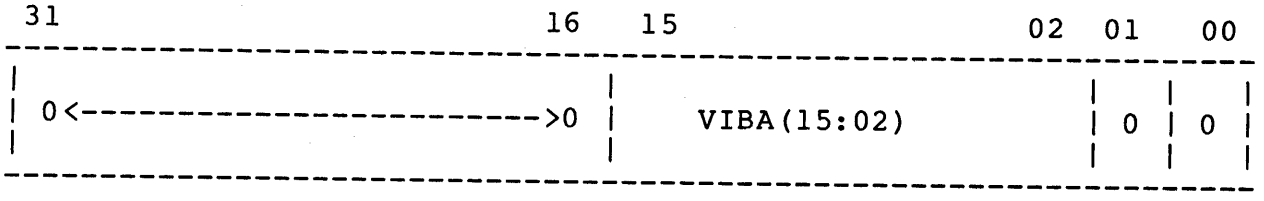
VAX MODE





11 COMPATIBILITY MODE





VAMX CONTROL

The Compatibility Mode bit in the PSL register controls which address format is selected in the VAMX multiplexor.

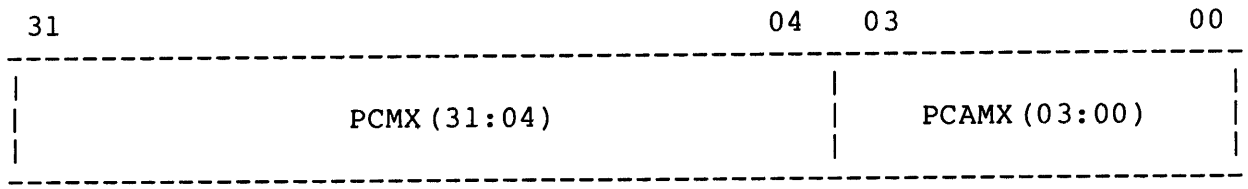
The address select is provided by a signal from the TBM. The VA is selected as the address source whenever the micro program is requesting a memory reference. At all other times, the micro program selects the VIBA as the address source and the IBUF control logic is allowed to use the cycle to request a memory transfer.

1.4.4 PC

Program Counter

The PC is used to hold the address of the opcode each time a new instruction execution is started, and is incremented as each of the operand specifiers are being evaluated.

PC data type:



PC CONTROL

The loading of the PC is controlled by the UPCK field of the UWORD. Command codes in the UPCK field specify incrementation of the PC by 1, 2, 4 or N. The value N is determined by the instruction buffer control logic and is used to increment the PC beyond instruction stream bytes associated with each specifier.

UPCK

Uword Program Counter control (K) field, 3 bit.

0	NO-OP	4	PC<--PC+1
1	PC<--VA	5	PC<--PC+2
2	PC<--IBA	6	PC<--PC+4
3	VA<--VA+4	7	PC<--PC+N

1.4.5 PCADD

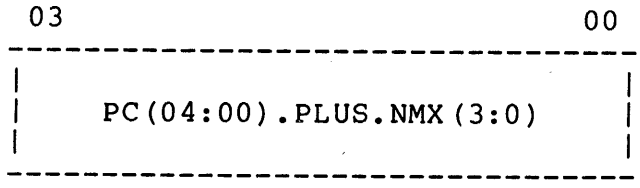
Program Counter ADDer

NMX

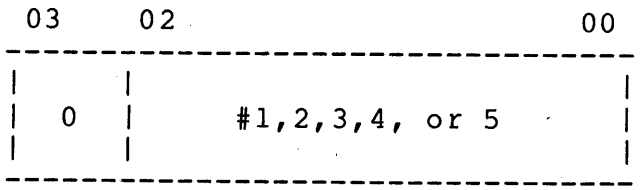
Number MultipleXor

The PCADD and NMX allow the numbers 1, 2, 4 and N to be added to the PC register. The number N comes from the Instruction buffer control and may be the numbers 1, 2, 3, and 5. The four bit output of the PCADD is loaded into the lower four bits of the PC register and if a carry results the upper 24 bits of the register is incremented.

PCADD data type:



NMX data type:



PCADD, NMX CONTROL

The PCADD and NMX are controlled by the UPCK field of the UWORD.

1.4.6 PCMX

Program Counter MultipleXor

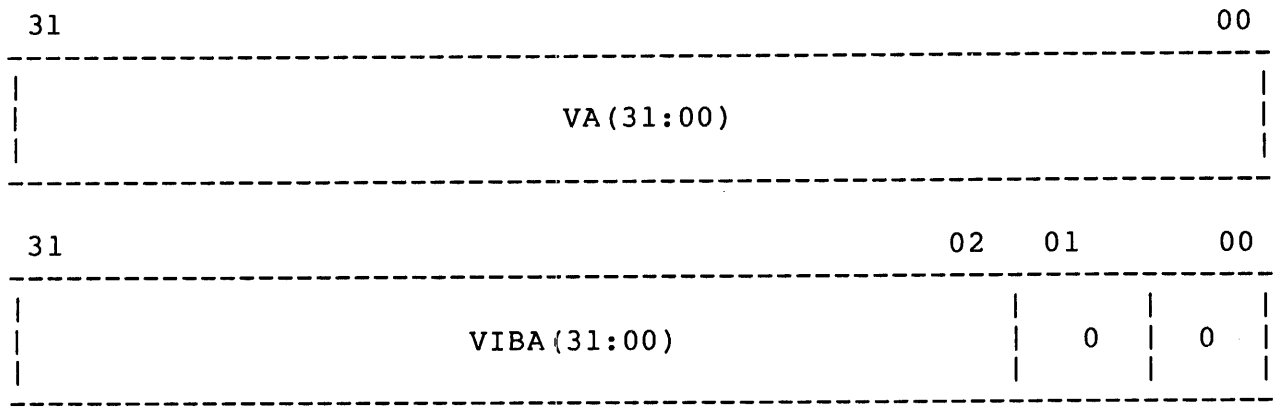
PCAMX

PC Adder MultipleXor

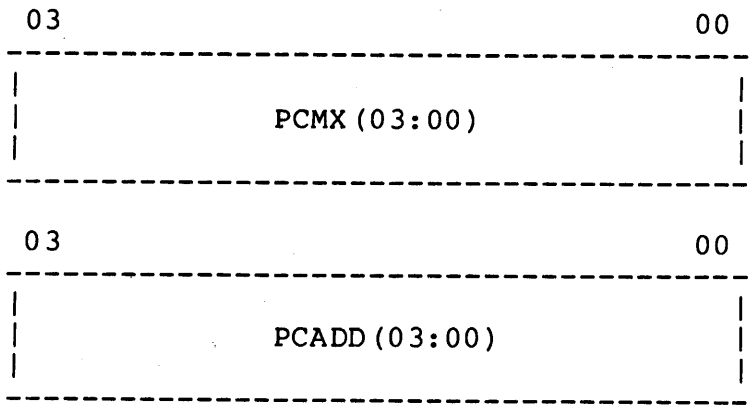
The PCMX and PCAMX provide the data input to the PC register. Whenever the PC register is being loaded the PCAMX selects the PCMX for the low 4 bits and the PCMX selects either the VA register or the VIBA register.

Whenever the PC register is counted the PCAMX selects the PCADD to be loaded into the low 4 bits of the register while the upper 24 bits of PC may be incremented.

PCMX data types:



PCAMX data types:



PCMX, PCAMX CONTROL

The PCMX and PCAMX are controlled by the UPCK field of the UWORD field.

CHAPTER 2

MICRO SEQUENCER SPECIFICATION

The purpose of the Micro Sequencer is to control CPU operations and functions and control the Micro Word Fields to ensure known operation during Powering Up, Powering Down, Micro Traps, Stalls, Micro Word ECOs, and console operations.

2.1 NORMAL MODE

The Micro Word is controlled by the Micro Program Counter address lines. (UPC) The Branch Enable field (BEN) and/or the Jump field (J Field) control the next Micro Address (NUA) which forms the UPC.

2.2 MICRO ECO CONTROL (UECO) MODE

The purpose of the UECO logic is to allow Micro words that need changes to be updated and written into Writeable control store (WCS). This greatly reduces the requirements for changing many Prom Chips by programming only one FPLA per CPU.

When a uword ECO is needed, the UPC address of the Prom uword is put into an FPLA. In addition, a new address to be used is put into the FPLA which references WCS.

During CPU operation and when the UPC matches an address in the FPLA, the FPLA outputs a signal to the stall control and Pico sequencer. Upon receipt of an ECO Dispatch signal a clear uword fields signal and an abort cycle signal are generated. These signals create in effect a NO-OP cycle in which to allow a new uword from WCS to be accessed.

The FPLA output is clocked into the UECO register at CPT 0 of the NO-OP cycle. This register holds the lower six bits of the WCS address which contains the program patch. The UMX select lines then allows the UECO register to be the source for the UPC address lines. WCS is then accessed and the new uword is clocked into the uword registers at CPT0 of the next cycle.

Control is returned to the normal Prom control store sequence by the Ben and J Fields of the uword from WCS.

The eight bit output from the FPLA is used as follows:

07= not presently used
06= signals the ECO sequence to begin
<05:00>= the UECO address in WCS to be accessed.

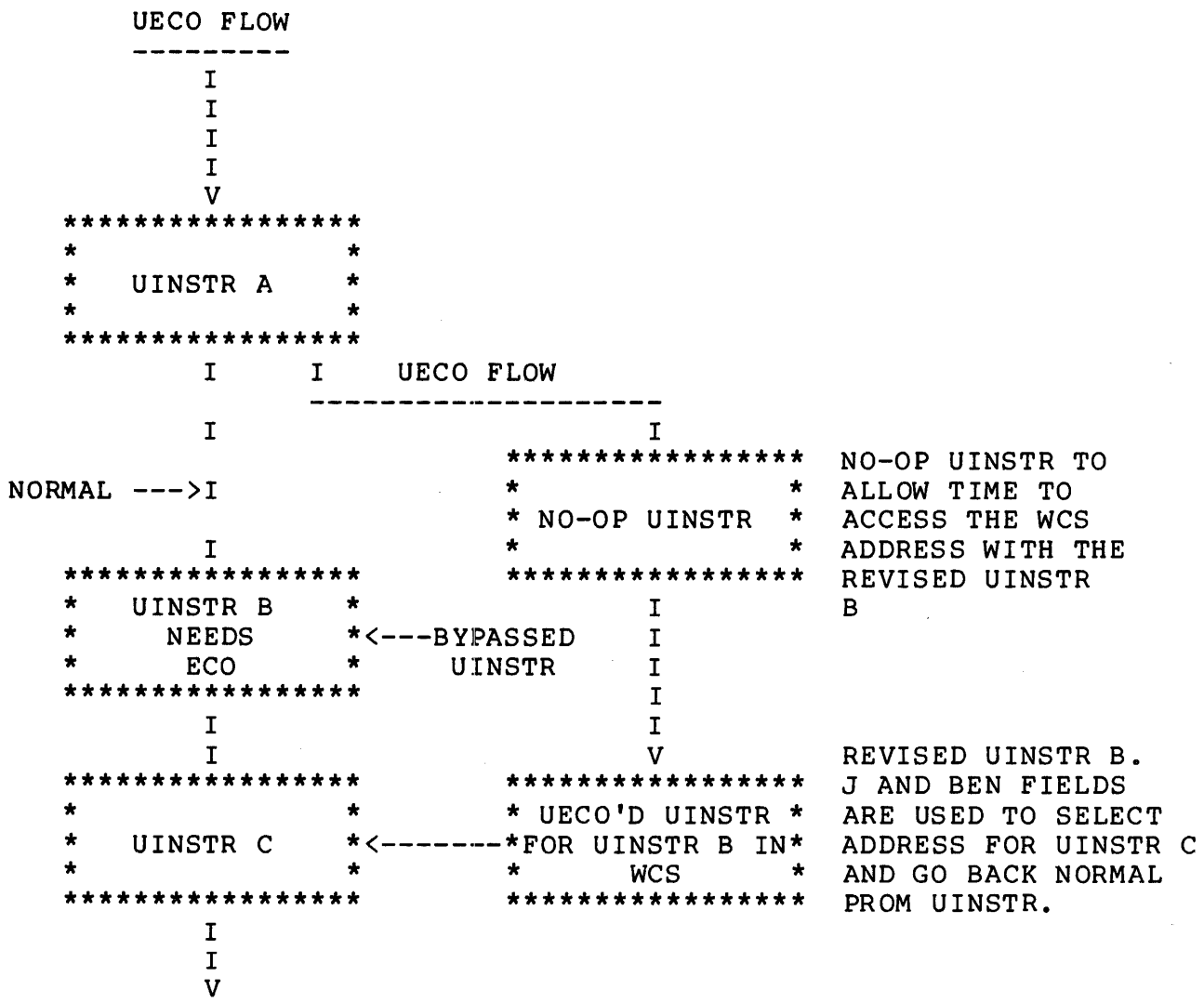


Figure 2-1

2.3 MICRO TRAP (UTRAP) MODE

Utraps are due to faults or errors in the CPU. Upon receiving a utrap signal the Micro sequencer control (USC) sends out the signals clear uword and abort cycle to other CPU subsystems. These signals create NO-OPs and allow time to direct the CPU to error handling micro code.

The utrap signal is clocked into the Pico sequencer at CPT0. The utrap cycle then selects the UMX to make the vector address the UPC address for the new uword to be clocked into the users registers at the next CPT0.

The utrap cycle clears the Ben,USUB, and JField registers on the USC. Ben 10 (HEX) is then enabled to allow the CEH module to control the NUA bits <03:00> as per the type of utrap.

The utrap vector address areas are:

Prom	0100	upto	010F	(HEX)
WCS	1100	upto	110F	(HEX)

The console can direct the utrap vector to WCS when the CIBN UPC 12(1) H bit is set in the console.

The UTRAP vectors are in HEX and their functions are:

X100	System Init
X101	Unaligned Data Trap
X102	Page Trap
X103	M Bit
X104	Protection Violation
X105	Translation Buffer Miss
X106	Reserved Floating Operand
X107	Trans. Buffer Parity Error
X108	Cache Parity Error
X109	Reserved
X10A	Reserved
X10B	Reserved

X10C	RDS Error
X10D	Time Out
X10E	Odd Address Error
X10F	Control Store Parity Error

When a utrap is iniated by the hardware, the Micro Program counter save register (UPCSV) contents are pushed onto the Micro Stack memory (USTACK) to specify the address of the NEXT normal uword to be used when returning from the utrap routine. The UPCSV is clocked at CPT 0 and therefore holds the UPC of the current executable uInstr. During utrap sequences the current executable uInstr is NO-OP'd while the ustack push is done.

The ustack pointer (USP) address is decremented prior to writing the information onto the ustack.

When in a utrap cycle a signal for clearing utraps is sent to the CEH module at CPT 75 to acknowledge a utrap cycle (uword no-op to set up the utrap vector address) being in progress.

2.4 CONTROL STORE PARITY ERROR MICRO TRAP MODE

This utrap differs from the other utraps in that the clocking of the UPCSV is inhibited at CPT 0 . The UPCSV contents which are pushed onto the ustack are the address of the control store uword which failed parity checking instead of the updated address as in any other utrap.

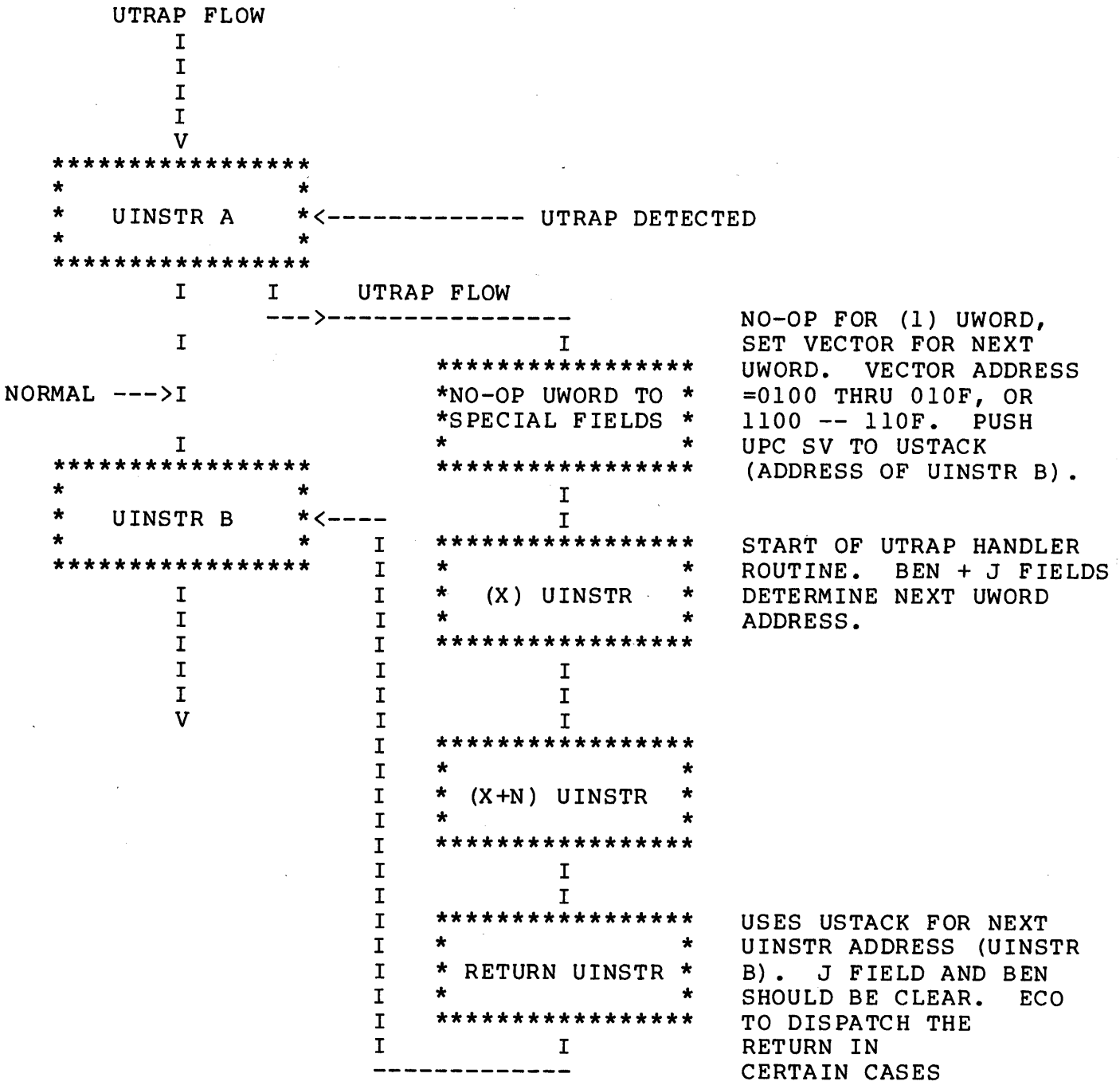

UTRAP FLOW
UINSTR A
UTRAP DETECTED
UTRAP FLOW
NO-OP FOR (1) UWORD, SET VECTOR FOR NEXT UWORD. VECTOR ADDRESS =0100 THRU 010F, OR 1100 -- 110F. PUSH UPC SV TO USTACK (ADDRESS OF UINSTR B).
NORMAL
NO-OP UWORD TO SPECIAL FIELDS
UINSTR B
START OF UTRAP HANDLER ROUTINE. BEN + J FIELDS DETERMINE NEXT UWORD ADDRESS.
(X) UINSTR
(X+N) UINSTR
RETURN UINSTR
USES USTACK FOR NEXT UINSTR ADDRESS (UINSTR B). J FIELD AND BEN SHOULD BE CLEAR. ECO TO DISPATCH THE RETURN IN CERTAIN CASES

Figure 2-2

2.5 CACHE STALLS

Cache Stalls are basically due to the Cache data not present during reads, and the Cache waiting on I/O subsystems during writes.

For a complete definition of conditions causing a Cache stall refer to the Cache Subsystem Specification. During Cache Stalls other uword functions are temporarily put in a NO-OP state by sending out the clear uword signal to other subsystem units. This NO-OP condition may last for several Micro cycles until Cache is finished and negates the stall signal.

Upon completion of Cache Stall the next uInstr to be performed would be the uInstr that would have been executed if the stall had not occured.

The UPCSV contents are selected thru the UMX to be the UPC to be used to address the uword when the stall signal is asserted and when the stall signal becomes false.

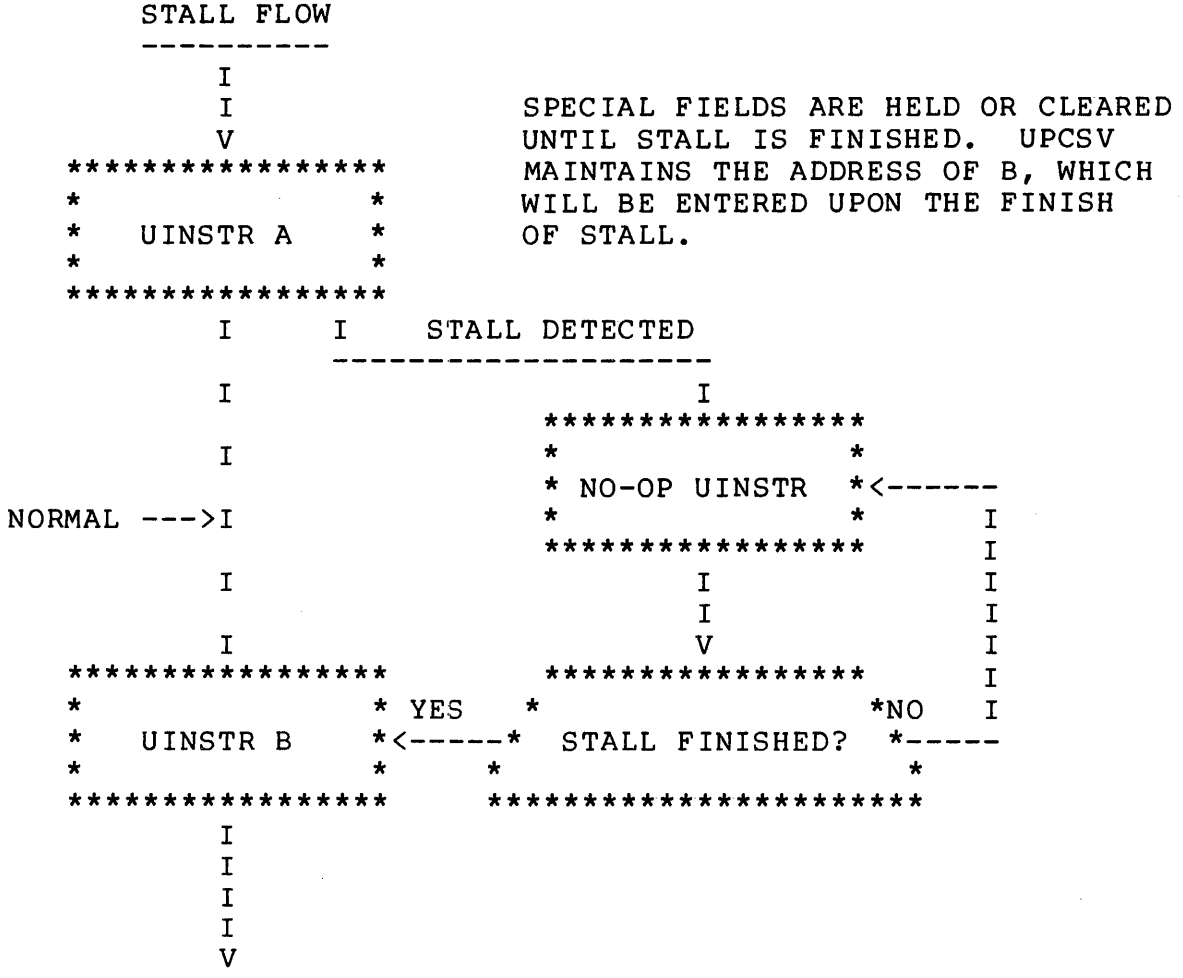


Figure 2-3

2.6 SYSTEM INITIALIZE

During Init the following conditions are held:

- All USC registers are held clear except the UECO UPCSV, UPC BREAK, and the V-Bus registers.
- The UMX selects are held so that a constant utrap vector is the UPC address and the decision point branch is inhibited.
- The Init state is held for one Micro cycle after the system Init level goes way to ensure all CPU subsystems start operations synchronously.
- For all conditions that generate initialize refer to the CEH and ICL modules specification.

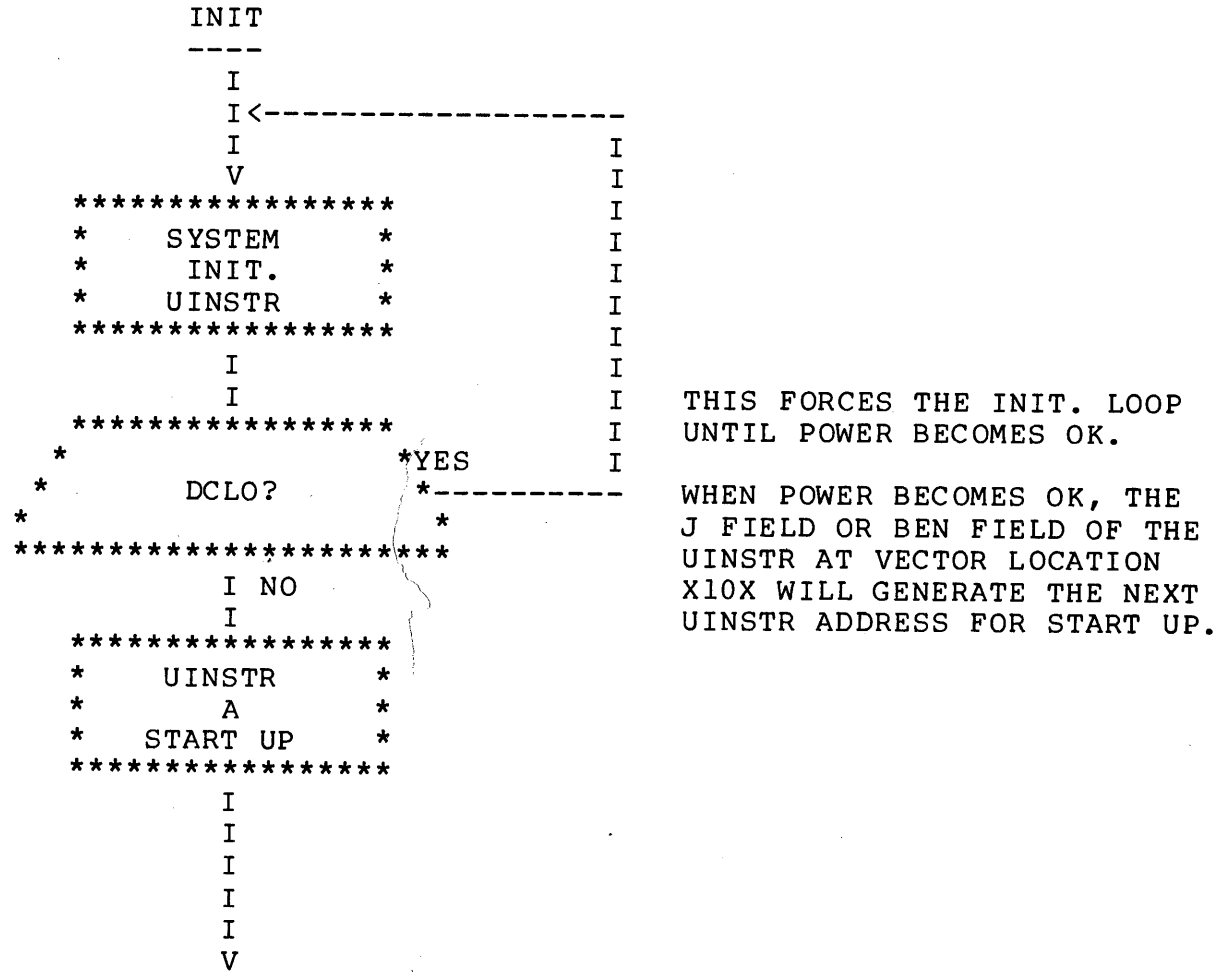


Figure 2-4

2.7 MICRO SUBROUTINE FIELD (USUB)

The uSub field is a two bit field and the operations are:

CODE	FUNCTION	OPERATION
00	NO-OP	No effect
01	Call	USP Decrements uStack Word written with UPCSV Data uPC<---(JField orBen)
10	Return	uStack word UPC<---(JField or Ben) or(uStack return address-13 bits) USP Increments
11	Decision Point Branch (DPB)	F.P.A to control UPC <12> JField to control UPC <11:08> I Buff to control UPC <07:00>

Control store bus bits <65:64> make the uSub Field.

2.8 JUMP FIELD (JFIELD OR UJMP)

Jump Field <12:00> 13 bits forms the base address for the next Micro address. (NUA) Control store bus bits <12:00> make the Jfield.

2.9 BRANCH ENABLE FIELD (UBEN)

UBen is a 5 bit field and the type of branches enabled are:

FUNCTION	OPERATION
Ben 00	No Operation
Ben 0F-01	1 of 15-(8) way Branch conditions
Ben 1B-10	1 of 12-(16) way Branch conditions
Ben 1F-1C	1 of 4-(32) way Branch conditions

Control store bus bits <76:72> make the uBen Field.

2.10 OTHER FIELDS - UBS+UBCT (NOT ON USC)

(See ID Bus specification)

FUNCTION	OPERATION
Push uStack	USP Decrements ID Bus Data Bits <15:00> written Uses: 1. Pass Parameters from D register to a subroutine 2. Put a return address on uStack for console control
Pop uStack	ID Bus (uStack word) USP Increments Uses: 1. Adjust uStack 2. Read the stored UPCSV to determine error locations

2.11 CALL SUBROUTINE

If the uInstr specifies a call subroutine function the UPCSV is pushed onto the uStack and forms the return address to be used later. The Ben, JField, and/or Decision point branch then determine the start of the subroutine UPC.

The Micro stack pointer (USP) is decremented prior to the push.

2.12 RETURN SUBROUTINE

The push of the call subroutine puts the UPC of the uInstr that has the CALL onto the uStack. When a return function is specified the uStack is "OR" D with the J field and/or Ben field of the return uInstr to make t return to the correct next uInstr past the call uInstr. After the return address is popped off the uStack, the USP is incremented.

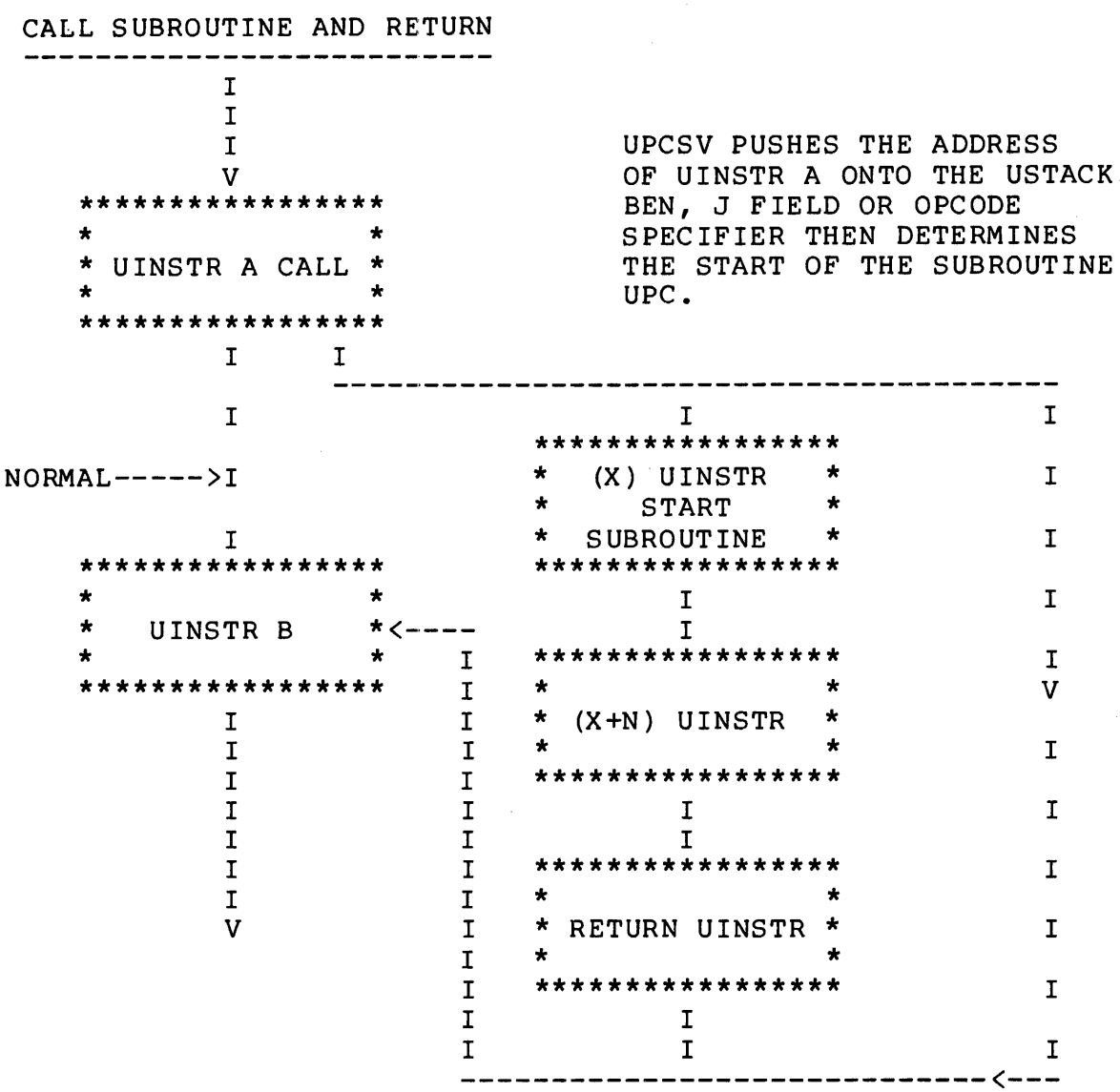


Figure 2-5

2.13 POWER UP OR DOWN

These conditions force the initialize signal which keeps the USC in an initialize state until after power becomes good. A constant uTrap vector is put onto the UPC lines during bad power.

2.14 CONSOLE CONTROLLED OPERATIONS

The following registers can be accessed by the console:

The uStack can be written from the ID Bus by a push function. This causes the USP to decrement and then write the ID Bus data onto the uStack.

The uStack can be read from the ID Bus by a POP function. The uStack data is given to the ID Bus and then the USP is incremented.

Maintenance return can select the uStack as a source for the NUA. This causes the NUA to get the uStack and then the uSP is incremented. The console signal to generate maintenance return is CIBN D MAINT RTN L.

The uPC break register can be written from or read by the ID Bus.

The WCS address register can be written from or read by the ID Bus.

The WCS Data can be written from the ID Bus in 32 data bit groups. When the WCS Data is read by the ID Bus, bits <07:00> reflect the eight possible WCS 1K address assignments and the presence of the WCS control store modules. All zeroes will be read back on ID Bus bits <31:08>.

The uPCSV register, the branch input lines and many other USC signals can be read by the V-Bus to the console.

The Break match signal goes to the clock control and will enable the CP clock to be stopped during CPT0 of a desired Micro word address if the console enable bit is set which also goes to the clock control.

The Break match signal becomes true when the UPC break register contents equals the uPC address of the NEXT uword to be used. This signal also is gated to the backpanel during CPT 150 time for an oscilloscope sync and to TP1 by the module handle.

The console can direct any uTrap handling to WCS instead of the normal Prom CS by forcing uPC <12> to a `ONE' during uTraps. This is done by the signal CIBN UPC 12(1) H.

The console can direct a micro word no-op cycle by the signal CIBN ROM NOPL. This forces the CLR uword and abort cycle signals to be true.

The timing of ID Bus read/writes of USC registers is done in accordance with the ID Bus spec.

The size of the registers accessed under ID bus operations are:

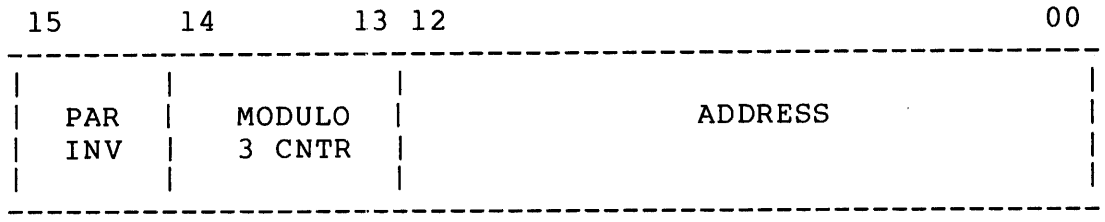
uStack = 16 Bits
UPC Break = 13 Bits
WCS address = 16 Bits
WCS Data = 32 Bits Write, 8Bits read

uSC ID Bus address assignments are in Hex as follows:

ADDRESS	REGISTER
20	Micro stack
21	uPC Break
22	WCS address
23	WCS Data

When writing or reading the uSC registers by way of the ID Bus, the LEAST significant bits will be used to pass the data.

WCS address register format

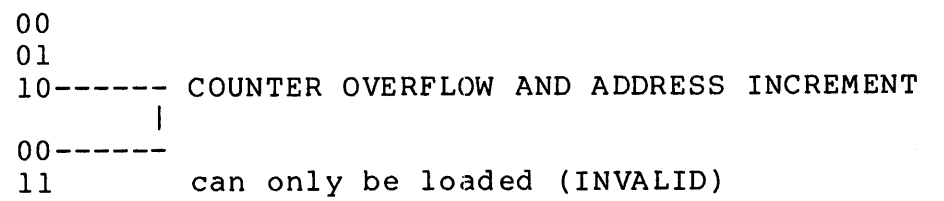


<15> = 0 = NORmal Parity
1 = Inverted Parity

<14:13> = 00 = WCS Data Bits <31:00>
01 = <63:32>
10 = <95:32>
11 = No Group Selected

<12:00> = WCS address

The modulo-3 counter will count as follows in binary:



The count of (11) can only be loaded and when loaded no further WCS data writes or address increments can take place until the modulo-3 counter and entire register are reloaded with a valid count.

Overflow from the Modulo-3 counter occurs when the count goes from (10) to (00) and increment the binary address in bits <12:00>.

The Modulo-3 counter increments after each WCS write. This allows the data to be written in successive higher order 32 bit groups without having to perform a WCS address write each time before writing data.

To write WCS, the WCS address register must be loaded and then a WCS write data must be performed. These cycles may be either back to back or spaced in time.

Microstack output data is clocked into a register at CPT 0 and held until the next CPT0.

V-Bus operation allows certain signals on the Micro sequencer to be read by the console in a transparent mode in respect to the micro sequencer.

The ID Bus address and read/write decoding are in accordance with the ID Bus specification.

2.15 PICO SEQUENCER AND PRIORITY DECODING

The Pico sequencer consists of flip-flops clocked at CPT0 to hold the conditions as exist at the beginning of a micro cycle. The conditions held are:

Initialize
Maintenance return
Cache Stall
Micro Trap
UECO

Priority of levels is as follows:

Highest Priority	Initialize
	Maintenance Return
	Cache Stall
	Micro Trap
	uECO
Lowest Priority	Normal

When maintenance return is true from the console, the CPU clock must be stopped and under control of the console. Maint. return keeps the NUA lines as the selected source of the UPC lines.

Maintenance return also inhibits writing of the ustack if a utrap or push operation occurs during the maint. utrap.

2.16 UPC ADDRESS LATCHING

The UPC latches hold the UPC lines stable to ensure that the control store bus lines are not changing. This is to prevent a false uword parity error. Control of the USC UMX tri-stating and decision point branch UMX tri-stating and UPC latching keeps the UPC valid.

At CPT 50 plus gate delays the USC UMX or decision point branch UMX are tri-state enabled. The UPC lines <09:00> are valid at CPT 90 plus 10ns of clock skew to the control store address inputs. The UPC lines <12:10> are valid at CPT 94 + 10ns of clock skew at the control store chip enable inputs. The inverter and open collector nand gate latches are enabled at CPT 125. At CPT 150 time plus gate delays the UMX tri-states are disabled with the latches holding the UPC levels. At CPT 50 the latches are opened and the next UPC address cycle can begin.

The present address in the UPCSV register will be displayed in 13 leds mounted near the module handle. A separate LED by the E-F handle section will be used to display the stall condition.

CHAPTER 3

INTERNAL DATA BUS SPECIFICATION

The Internal Data Bus is a high speed data path connection between the major functional areas of the CPU. It has four purposes:

1. Allows data to be transferred to/from the internal status and control registers and translation buffer.

2. Allows data in the form of displacements, and short & long literals to be transferred from the Instruction Buffer to the CPU and ACC data paths.

3. Allows data transfers between memory (via the D register) and the ACC data paths to take place.

4. Allows access of the internal registers from the console under console control in a maintenance operation mode.

3.1 FUNCTIONAL OPERATION

The ID BUS control is derived from a control field in the UWORD in normal operation and from the console interface logic in maintenance operation.

3.1.1 Normal Operation

Data transfers over the ID BUS are always directed to the Q and from the D register in the CPU data paths. The ACC will, however, snapshot the data on the bus when a transfer is being made from the IBUF to the Q register.

3.1.1.1 ID BUS Addresses - The ID BUS signal lines contain a six bit field to specify which internal register has been designated as the source or destination of data on the bus.

The address assignments are as follows:

00	IBUF DATA	20	USTACK
01	TIME OF DAY	21	UBREAK
02	NOT USED	22	WCS ADDRESS
03	SYSTEM ID	23	WCS DATA/STATUS
04	CNSL RXCS	24	P0BR
05	CNSL RXDB (TO ID)	25	P1BR
06	CNSL TXCS	26	SBR
07	CNSL TXDB (FROM ID)	27	RSVD FOR SYS SPACE
08	DQ (ID MAINT ONLY)	28	KSP
09	NEXT INTERVAL REG.	29	ESP
0A	CLOCK CS	2A	SSP
0B	INTERVAL COUNTER	2B	USP
0C	CES	2C	ISP
0D	VECT	2D	FPDA
0E	SIR	2E	D.SV
0F	PSL	2F	Q.SV
10	TBUF DATA	30	T0
11	-RSVD-	31	T1
12	TBUF REG0	32	T2
13	TBUF REG1	33	T3
14	ACC REG0	34	T4
15	ACC REG1	35	T5
16	ACC MAINT REG.	36	T6
17	ACC CONTROL/STATUS	37	T7
18	SBI SILO	38	T8
19	SBI ERR REG	39	T9
1A	SBI TIMEOUT ADDRESS	3A	PCBB
1B	SBI FAULT/STATUS	3B	SCBB
1C	SBI SILO COMPARATOR	3C	P0LR
1D	MAINTENANCE	3D	P1LR
1E	CACHE PARITY	3E	SLR
1F	-RSVD-	3F	RSVD FOR SYS SPACE

3.1.1.2 ID BUS Directional Control - One of the signal lines on the ID BUS will specify whether an internal register is to be read onto the bus or if data is to be clocked from the bus into the internal register. The D register in the Data Paths always acts as the source and the Q register as the destination.

ID WRITE L

H - ID BUS DATA<--(ADDRESSED REG.)
L - (ADDRESSES REG.)<--ID BUS DATA

3.1.1.3 ID BUS Data - There are 32 data signals on the ID BUS.

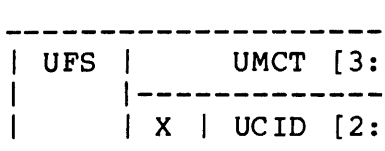
3.1.1.4 Signal Summary - There are 39 signals on the ID BUS:

NO. OF LINES		SIGNAL NAME
32		BUS ID D[31:00] L
6		ID LEFT ADDR (5:0) H
	or	ID RIGHT ADDR (5:0) H
1		ID LEFT WRITE L
	or	ID RIGHT WRITE L

3.1.1.5 ID BUS Control - The ID BUS signals are controlled by a field in the UWORD. The same UWORD field also controls MD BUS operation and therefore requires a 1-bit field in the UWORD for BUS control definition.

It is important to understand that the ID BUS is always active. That is, when the BUS control field defines MD BUS operations the ID address and write signals are zero and thus controlling the IBUF DATA to be gated onto the ID BUS data lines. The UQK field of the UWORD will select the appropriate time to clock the ID BUS data into the Q register.

The UWORD fields which control ID and MD BUS operations are as follows.



UFS - UWORD Function Select, 1 bit

0 - UMCT [3:0] = MD BUS CONTROL; ID BUS <-- IBUF DATA
1 - UCID [2:0] = ID BUS CONTROL & Console Control

No MD BUS function selected by micro instruction.

UMCT - UWORD Memory Control Field, 4 bits

UFS=0: SEE CACHE-SBI-TB SUBSYSTEM SPEC.

UFS=1: UCID - UWORD Console and ID bus control field

4 - ID DATA <-- (Internal Reg [SC 5:0])
5 - ID DATA <-- (Internal Reg [UKMX 5:0])
6 - (Internal Reg [SC 5:0]) <-- ID DATA
7 - (Internal Reg [UKMX 5:0]) <-- ID DATA

Internal Register addresses are generated either from the SC data path register or from the UKMX field of the micro-instruction.

& Console Control:
0 - NO-OP
1 - UNUSED SEE UWORD SPEC.
2 - CNSL ACK
3 - CNSL CONT

3.1.2 Maintenance Operation

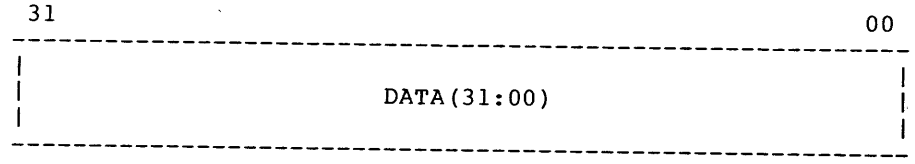
The ID BUS may also be controlled from the console interface logic in a maintenance mode operation. This allows access to writable control store, the USTACK, and visibility of internal registers from the console without the need of main microcode running. In maintenance mode operation only, the Data Paths D & Q registers may be addressed as an internal register over the ID BUS.

3.1.2.1 Console Control of ID BUS - Control of the ID BUS is accomplished via an interface signal generated by the console interface logic, ID MAINT.

When this signal indicates a maintenance operation the console will assert ID BUS address and write signals and may assert data.

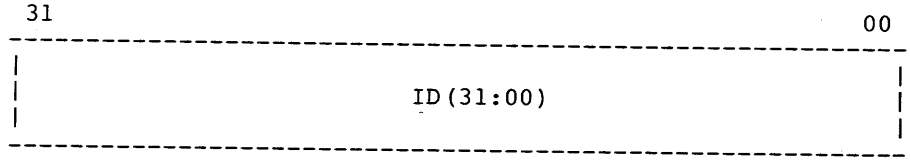
3.2 ID BUS REGISTER DESCRIPTION

3.2.1 IBUF DATA



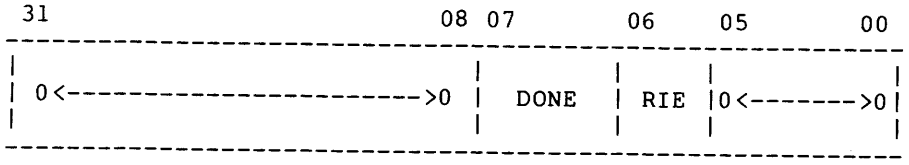
-READ ONLY REGISTER-
SEE IBUF SPEC

3.2.2 SYSTEM ID



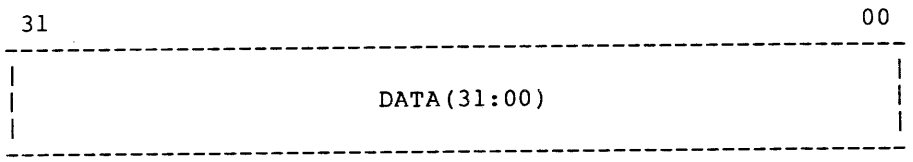
-READ ONLY REGISTER-
SEE CONSOLE SPEC

3.2.3 CNSL RXCS



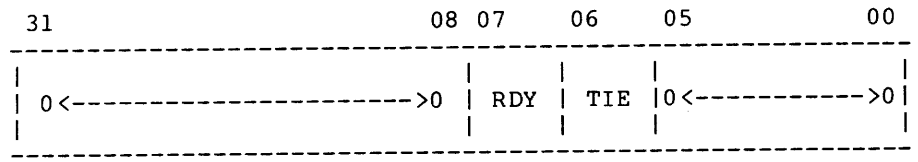
-READ/WRITE REGISTER-
SEE CONSOLE SPEC

3.2.4 CNSL RXDB



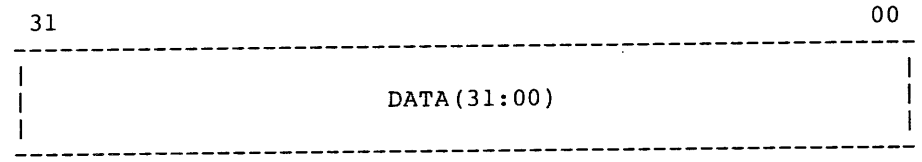
-READ ONLY REGISTER-
SEE CONSOLE SPEC

3.2.5 CNSL TXCS



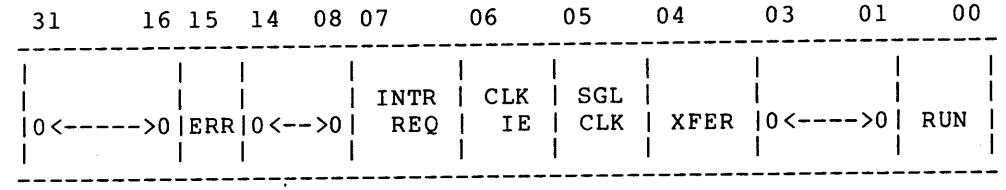
-READ/WRITE REGISTER-
SEE CONSOLE SPEC

3.2.6 CNSL TXDB



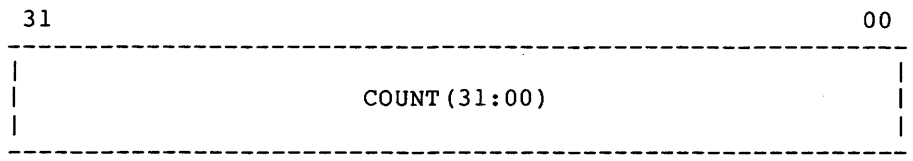
-WRITE ONLY REGISTER-
SEE CONSOLE SPEC

3.2.7 CLOCK CONTROL/STATUS



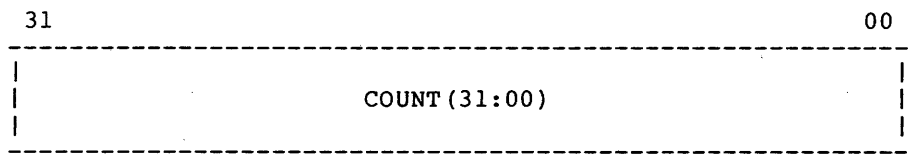
-READ/WRITE REGISTER-
SEE INTERVAL TIME CLOCK SPEC

3.2.8 NEXT INTERVAL COUNT



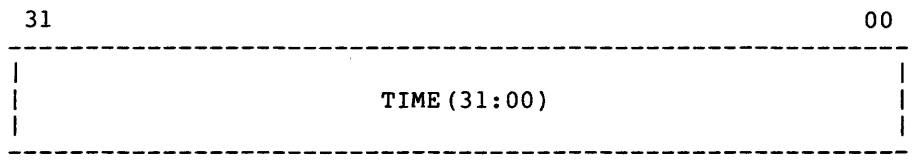
-WRITE ONLY-
SEE INTERVAL TIME CLOCK SPEC

3.2.9 INTERVAL COUNT



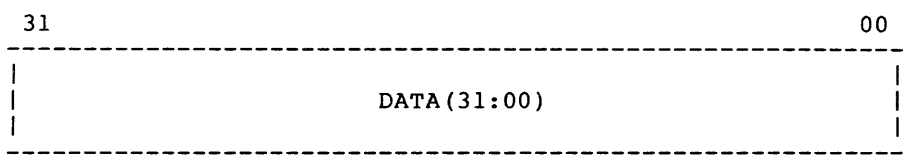
-READ ONLY REGISTER-
SEE INTERVAL TIME CLOCK SPEC

3.2.10 TIME OF DAY



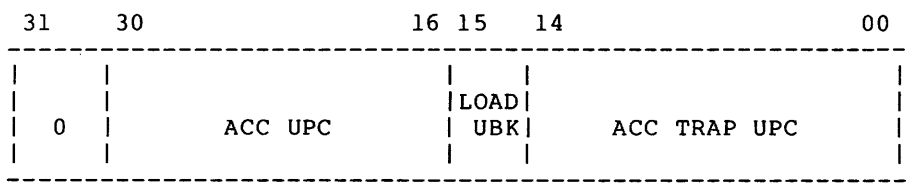
-READ/WRITE REGISTER-
SEE TIME OF DAY SPEC

3.2.11 ACC REG #0 THRU #1



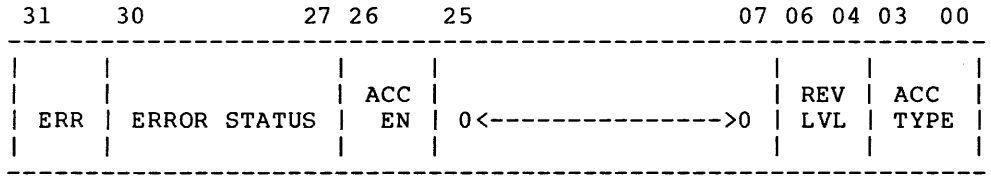
SEE ACCELERATOR SPEC

3.2.12 ACC MAINT



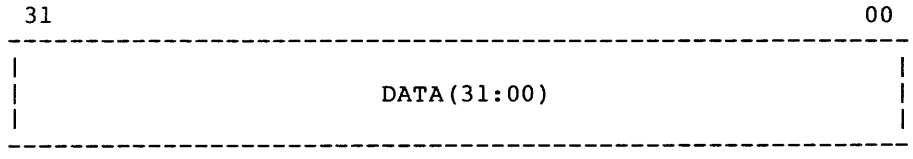
-READ/WRITE REGISTER--SEE ACCELERATOR SPEC

3.2.13 ACC CONTROL/STATUS



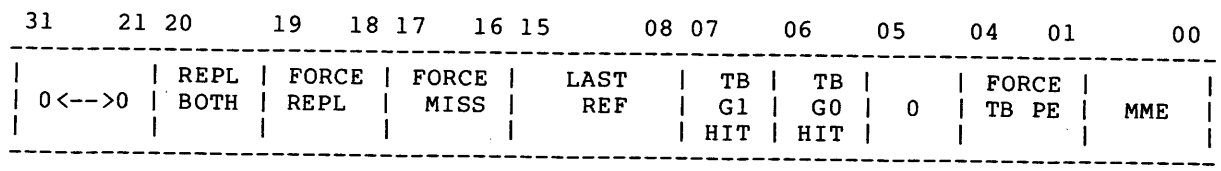
-READ/WRITE REGISTER--SEE ACCELERATOR SPEC

3.2.14 TBUF DATA



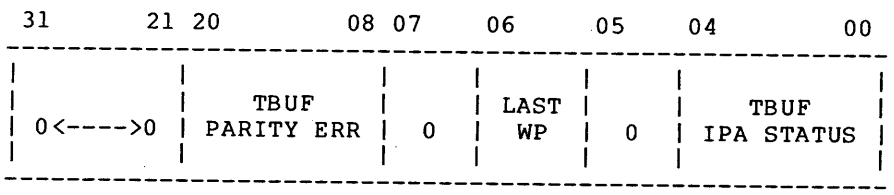
-WRITE ONLY--SEE CACHE SUBSYSTEM SPEC

3.2.15 TBUF REG0



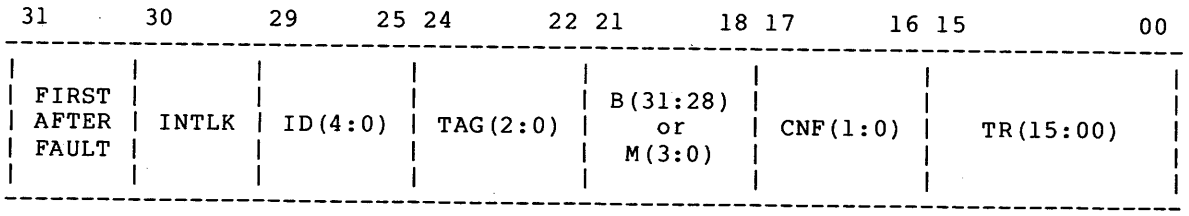
-READ/WRITE REGISTER-
SEE CACHE SUBSYSTEM SPEC

3.2.16 TBUF REG1



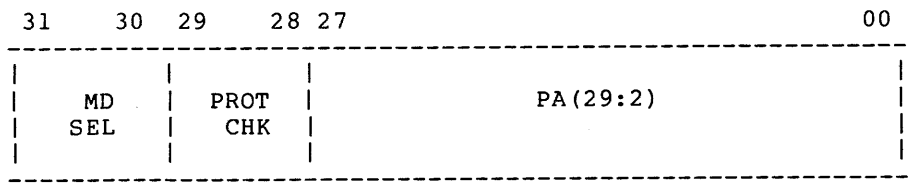
-READ/WRITE REGISTER-
SEE CACHE SUBSYSTEM SPEC

3.2.17 SBI SILO



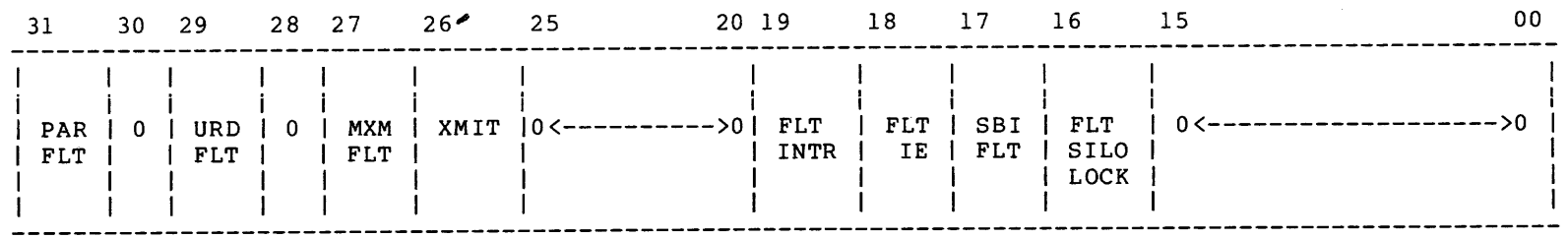
-READ ONLY-
SEE SBI REGISTER DEFINITION

3.2.18 SBI TIMEOUT ADDRESS



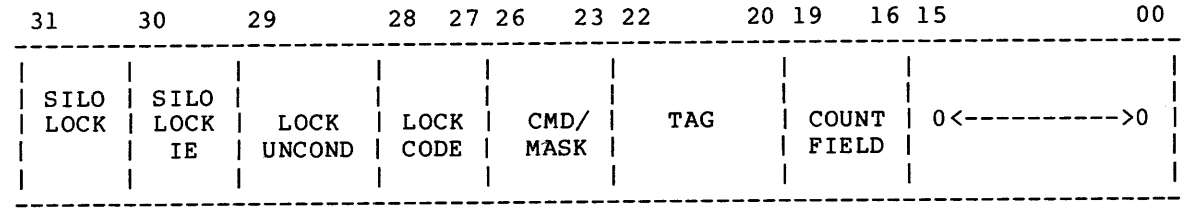
-READ ONLY-
SEE SBI REGISTER DEFINITION

3.2.19 SBI FAULT/STATUS



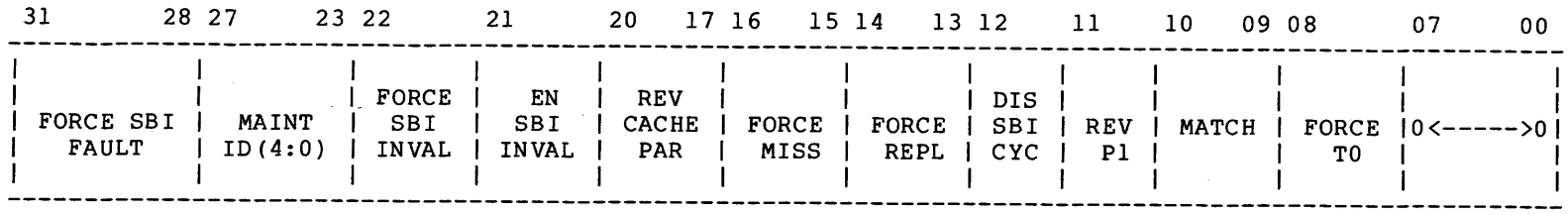
-READ/WRITE REGISTER-
SEE SBI REGISTER DEFINITION

3.2.20 SBI SILO COMPARATOR



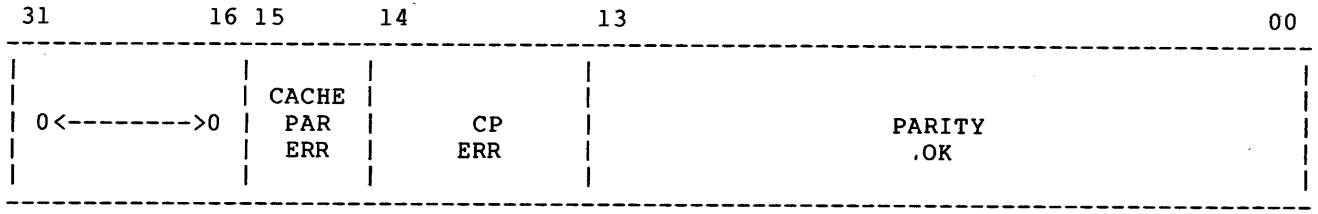
-READ/WRITE REGISTER-
SEE SBI REGISTER DEFINITION

3.2.21 SBI MAINTENANCE



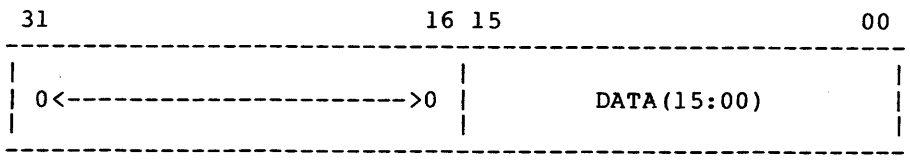
-READ/WRITE REGISTER-
SEE SBI REGISTER DEFINITION

3.2.22 SBI CACHE PARITY



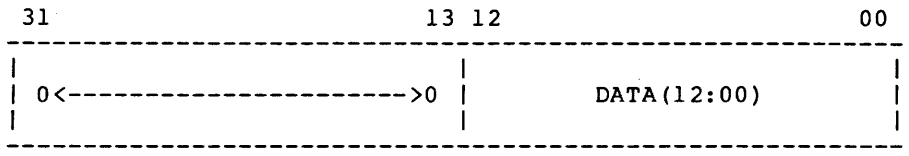
-READ ONLY-
SEE SBI REGISTER DEFINITION

3.2.23 USTACK



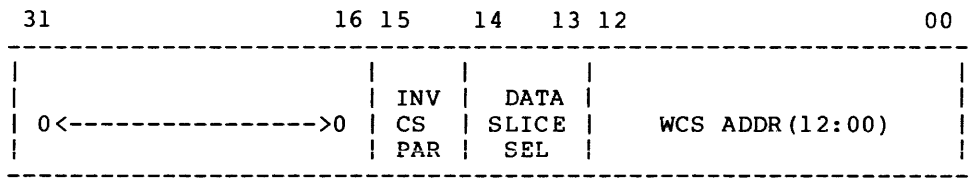
-READ/WRITE REGISTER-
SEE MICRO SEQUENCER SPEC

3.2.24 UBREAK



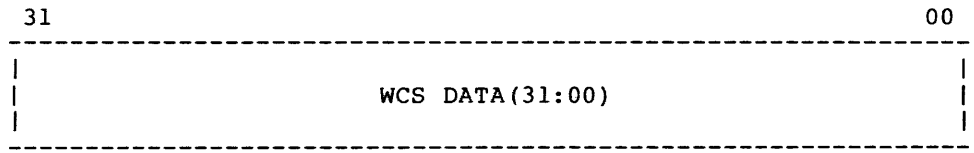
-READ/WRITE REGISTER-
SEE MICRO SEQUENCER SPEC

3.2.25 WCS ADDRESS

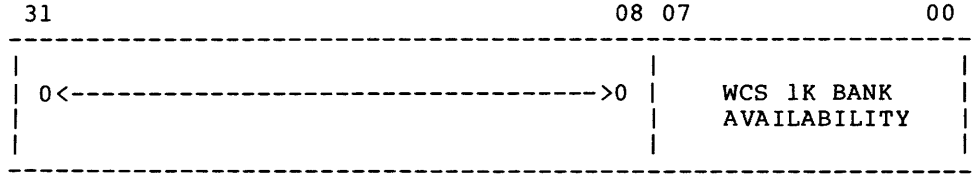


-READ/WRITE REGISTER-
SEE MICRO SEQUENCER SPEC

3.2.26 WCS DATA/STATUS

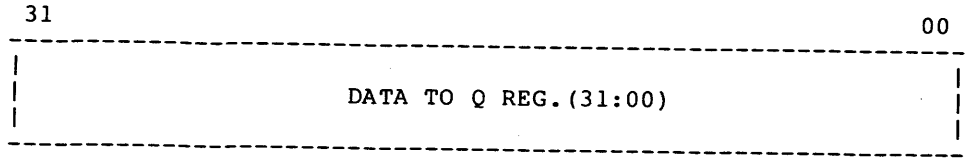


-WRITE ONLY BITS-

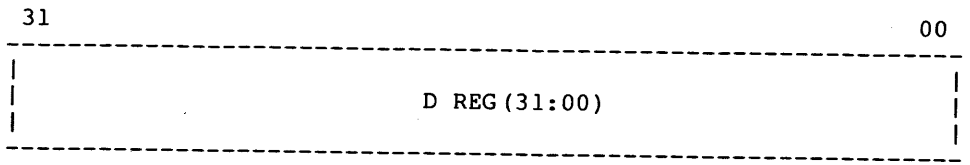


-READ ONLY BITS
SEE MICRO SEQUENCER SPEC

3.2.27 D,Q (MAINT MODE ONLY)

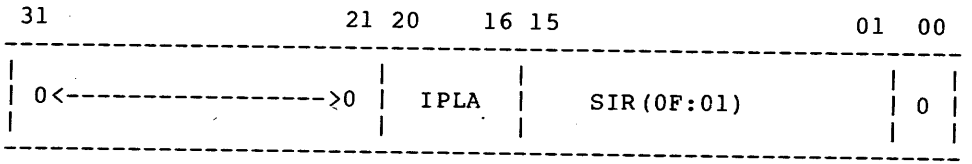


-WRITE ONLY BITS-



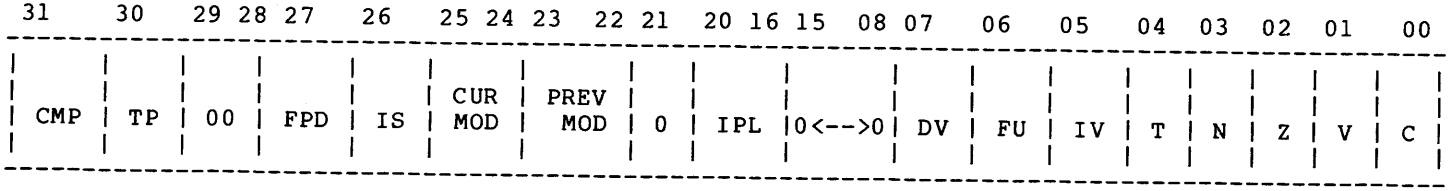
-READ ONLY BITS-

3.2.28 SIR



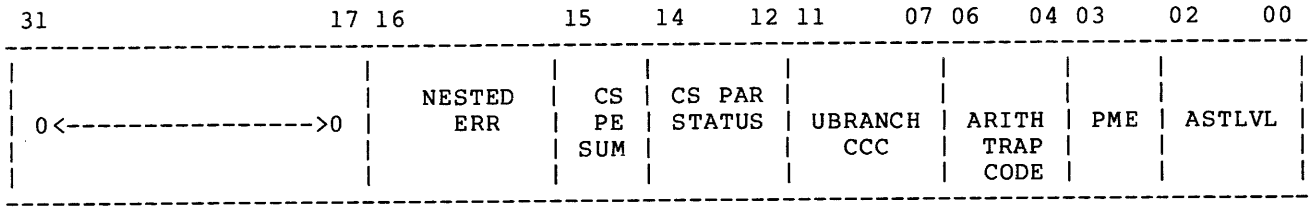
-READ/WRITE REGISTER-
SEE INTERRUPTS $ EXCEPTIONS SPEC

3.2.29 PSL



-READ/WRITE REGISTER-

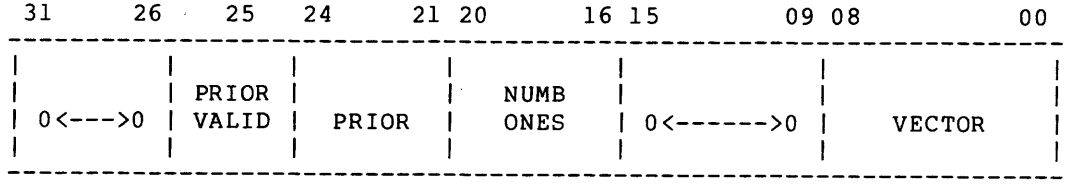
3.2.30 CPU ERROR/STATUS



FIELD	DESCRIPTION
ASTLVL (Asynchronous System Trap LeVeL)	Loaded or read by micro-code in MTPR Src, @#ASTR or MFPR @#ASTR, dst. Used in execution of REI instruction for AST delivery. Read/Write. Cleared on power up.
PME (Performance Monitor Enable)	Loaded or read by micro-code in MTPR Src, @#PMR or MFPR @#PMR, dst. Cleared on power up. Read/Write.
ARITH TRAP CODE (ARITHmetic TRAP CODE)	Loaded by hardware for enabled integer overflow traps when the V bit is set using the CCL7 function of the UCCK field of the UWORD. Must be loaded by microcode for arithmetic traps detected by other means. If non-zero an ARITHMETIC TRAP occurs. Read/Write. Cleared on power up. NOTE: The write is disabled if the appropriate Trap enable in PSL is 0 for the code being loaded. The codes have the following meaning: 0 - NO ARITHMETIC TRAP PENDING 1 - INTEGER OVERFLOW 2 - INTEGER DIV by ZERO 3 - FLOATING OVERFLOW 4 - FLOATING DIV by ZERO 5 - FLOATING UNDERFLOW 6 - DECIMAL OVERFLOW 7 - DECIMAL DIV by ZERO

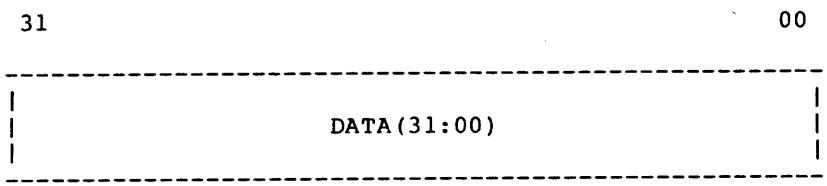
UBRANCH CC (Micro BRANCH Condition Codes)	Loaded by the micro-code or by the CCL1 function of the UCCK field of the UWORD. Read/Write. Cleared on power up. BIT --- 7 - ALUC 8 - ALUZ 9 - ALUN 10 - EALUZ 11 - EALUN
CS PE(2:0) (Control Store Parity Error)	Status indicators for Control Store Parity errors as follows: CS PE0 - CS DATA(31:00) CS PE1 - CS DATA(63:32) CS PE2 - CS DATA(95:64) Cleared on power up or by writing a one into CS PE SUM bit position. Read Only.
CS PE SUM (Control Store Parity Error SUMmary)	The 'or' of CS PE(2:0) Read/Write.
NESTED ERR	Read/only bit used by micro-code in memory management flows. Cleared on power up.

3.2.31 VECTOR



-READ/WRITE REGISTER-
SEE INTERRUPTS & EXCEPTIONS SPEC

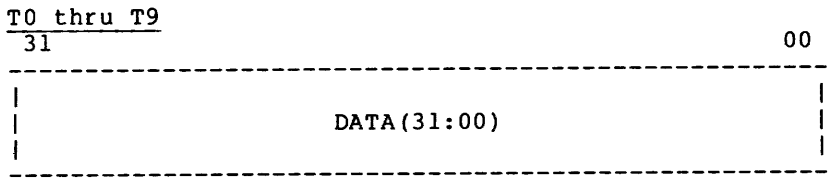
3.2.32 FPDA, D.SV, Q.SV



DESCRIPTION

Read/write registers used by micro-code in memory management flows.

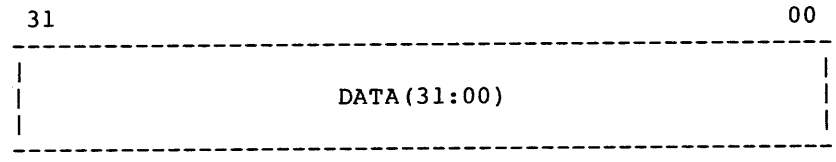
T0 thru T9



DESCRIPTION

Read/write registers used by micro-code as temporary holding registers.

3.2.33 P0BR P1BR SBR P0LR P1LR SLR PCBB SCBB KSP ESP SSP USP ISP



-Read/Write System registers-

CHAPTER 4

INSTRUCTION BUFFER

The instruction buffer is a eight byte array that fetches and decodes instruction stream data to decode opcodes and operand address information. It is basically made of a fifo type byte buffer with decode logic to generate micro addresses and branch conditions.

The major sections of the instruction buffer are:

1. Buffer Data Path
2. I-Stream Data Mux
3. IR Decode Logic

4.1 BUFFER DATA PATH

The instruction buffer data path consists of four major components:

1. A buffer register for storage of information.
2. A multiplexer shift network for advancing data thru the buffer register.
3. An input multiplexer for selecting data from memory or from the shift network.
4. A byte rotator used to convert the long word aligned data from the cache into byte addresses positions.

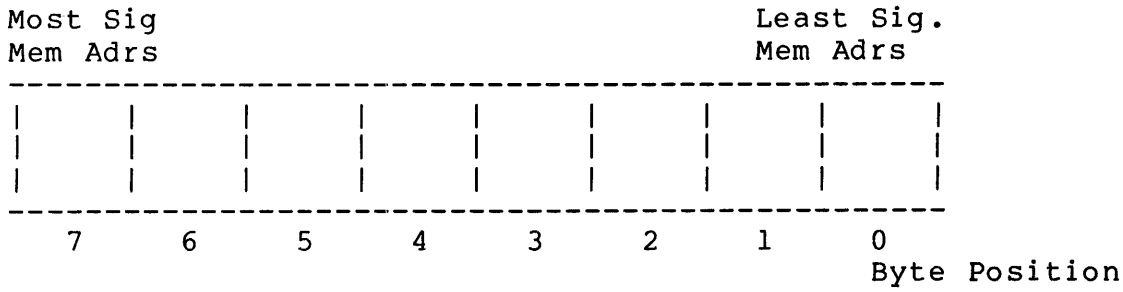
4.1.1 Buffer Register

This register is divided into eight sections of nine bits each. The nine bit sections contain:

8 Bits of data

1 Bit to indicate valid data

The bytes within the register are designated as bytes 7 thru 0. Byte 0 corresponds to data with the lowest value memory addresses. Likewise, the memory address increases with higher number byte positions.



The chip types for this register are 74S174's for bytes 7 thru 2. For bytes 1,0 will be 74S175's.

The entire register will be clocked at CPU T0 with no gating conditions.

4.2 SHIFT NETWORK

4.2.1 Multiplexer Shift Network

The shifting scheme for the instruction buffer is a four input multiplexer that is arranged to shift nine bit sections by 0, 1, 2 or 4 positions right at each micro cycle. The control of the shifter will be from a decode of 4 bits of the micro word. The micro controls are:

UIBC
- 0 - NOP
- 1 - STOP
- 2 - FLUSH AND LD VIBA
- 3 - START
- 4 - CLR BYTES 0,1
- 5 - CLR BYTES 2,3
- 6 - NOT USED
- 7 - B DEST
- 8 - NOT USED
- 9 - NOT USED
- A - NOT USED
- B - NOT USED
- C - CLR BYTE 0
- D - CLR BYTE 1
- E - CLR BYTES 0,1,2,3
- F - CLR BYTES 1-5 CONDITIONAL

4.2.1.1 MICRO Control Use -

0 - NOP - Default State. Has no effect on the instruction buffer.

1 - STOP - The assertion of this code will disable the instruction buffer cache requests. Current use is for micro code fault routines.

2 - FLUSH - This code is used to clear all valid bits of the IB. This will be used for jumps or branches and other conditions that change the PC. The same micro STATE will load the virtural IBA.

3 - START - The assertion of this code will resume prefetch operations to the cache.

4 - CLR 0,1 - This is used for compatibility mode to advance to the next OP code. It is also used in VAX mode for optimized fast execute (S , R or R, R) or 16 bit branch destinations.

5 - CLR 2,3 - This is used only in compatibility mode for discarding 16 bit displacements and literals after they have been used.

7 - B DEST - This code is used by the microcode to extract branch destinations from the instruction stream.

C - CLR 0 - This is used in VAX mode to advance to the next opcode.

D - CLR 1 - This is used in VAX mode for discarding the specifier from the IB. It is only used for displacement mode addressing, absolute addressing or long literals. In these modes, the last thing removed from the instruction buffer is the specifier. In all other modes, the specifier will be removed by hardware.

E - CLR 0,1,2,3 - This is used in compatibility mode for optimizing literal to register instructions.

F - CLR 1-5 Conditional - This is used in VAX mode for discarding long literals or displacements from the IB. The hardware will decode the specifier and/or opcode to determine 8,16, or 32 bits of data. In the modes that do not have I-stream data other than the specifier, the specifier itself will be discarded. This code is used at all FORK entries.

4.3 INPUT MULTIPLEXER

The input MUX is used to determine if data is to be loaded from the shift multiplexer or the cache data. The select of this MUX is controlled individually for each 9 bit section. It is selected for the shifter if the 9 bit section being shifted in is val d. Otherwise it is selected for cache data.

4.4 BYTE ROTATOR

This hardware is located as one of the sources to the input MUX. It is controlled by the IB by the low order two bits of the IB address register. These two bits will rotate the long word data such that the byte address will specify which byte is in the low order 8 bits.

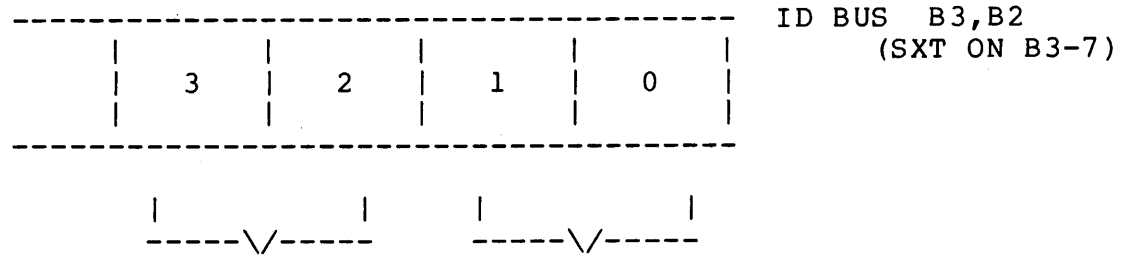
IBA	Byte Positions			
00	3	2	1	0
01	0	3	2	1
10	1	0	3	2
11	2	1	0	3

4.5 I-STREAM DATA MUX

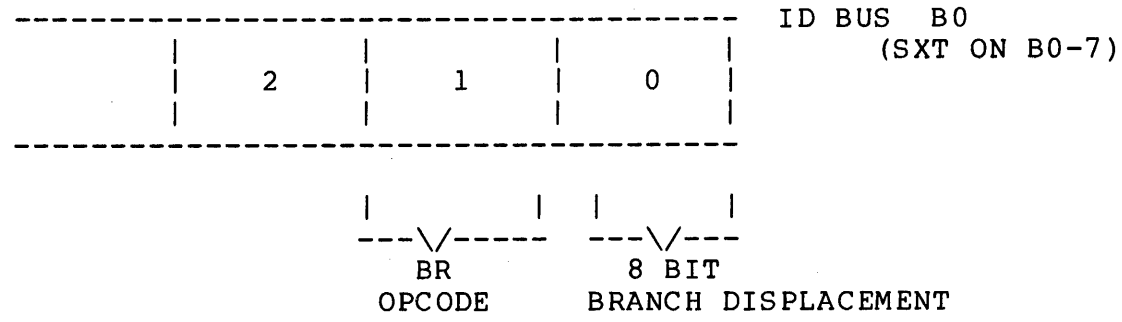
One of the functions of the instruction buffer is to sort out displacements, literals and other information and present it to the CPU data path. The IB will do as much as possible to make the data be in a useable form. This implies sign extension and shifting where necessary. To do this requires a decode of the mode, specifier, context lookup and opcode.

For compatibility mode, the IDMX will be selected for either bytes 3 and 2 or byte 0 shifted for branch displacements.

1.



2.

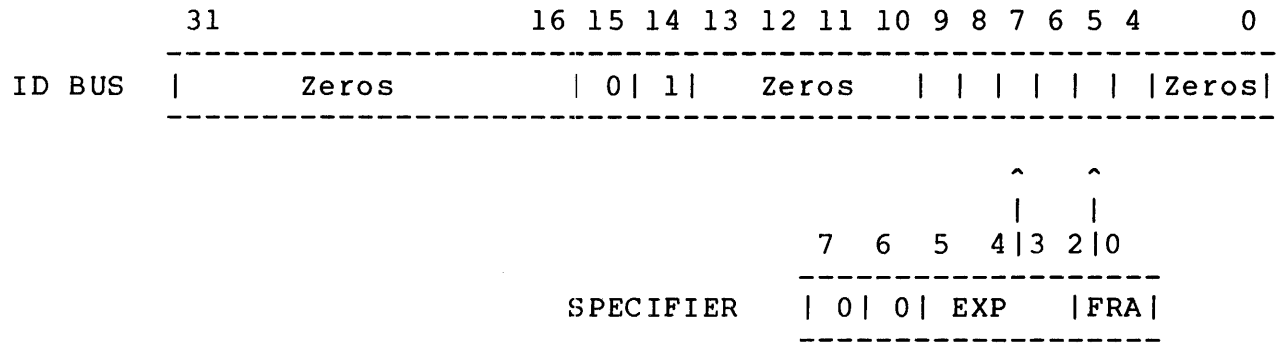


The branch displacement will be left shifted by one and sign extended on bit 7. For VAX Mode The IDMX Will Be As Follows:

Mode		Data	Comment
0	S#	7:0	Zero extended
1	S#	7:0	Zero extended
2	S#	7:0	Zero extended
3	S#	7:0	Zero extended
4	(E)	X	
5	R	X	
6	(R)	X	
7	-(R)	X	
8	(R)+	31:00	1,2, or 4 bytes depending on context.

			Sign extended.
9	@(R)+	31:00	
A	D8	7:0	Sign extend on bit 7
B	@ D8	7:0	Sign extend on bit 7
C	D16	15:00	Sign extend on bit 15
D	@ D16	15:00	Sign extend on bit 15
E	D32	31:00	
F	@ D32	31:00	
8 Bit BDest		7:0	Sign extend on bit 7
16 Bit BDest		15:0	Sign extend on bit 15

Floating Short Literal



4.6 PC UPDATES

PC Delta is a three bit number that is added to the low order bits of the PC register. The PDP-11 architecture has defined that each time a length of I-stream is fetched, that the PC is updated to reflect it. This means that during displacement mode addressing off of PC, that the PC is pointing at the end of the displacement. In order to eliminate an extra micro instruction on this flow, the hardware will determine the length of I-stream data to be used and create a value to be added to the PC. The end result is that displacement mode addressing can be done very quickly with the instruction buffer.

The PC update value will reflect the specifier (if any) and the length of I-stream required. Addressing modes that require additional data are:

Mode	Length
8- (R)+ if R=PC	Context Dependent
9- @(R)+ if R=PC	4 Bytes
A- D8	1 Byte
B- @ D8	1 Byte
C- D16	2 Bytes
D- @ D16	2 Bytes

E- D32 4 Bytes
F- @ D32 4 Bytes

The complete number to be added to PC will include the length number for the addressing mode plus one for the specifier. Note that the update for the opcode must be handled seperately by micro code.

Compatibility mode updates will work in a similar manner. The hardware will determine if any address calculation is required and the mode. If the following I-stream data is required, the number two will be added to the PC. If not, a zero will be added.

If a fault is detected such as error, TB miss or stall, the hardware will force the PC update to be zero.

If the opcode is a single byte instruction, the update number will be zero.

If the decision point entry is to an execute flow, the update number will be zero.

If first part done is set, the PC update will be zero.

4.7 IR DECODE

The IR decode will be done by the use of Roms. There are presently eight major execution points that are using a hardware generated address. The evaluation of additional operands will be done by a micro subroutine which is directed by a special subroutine call on the specifier.

Additional information such as context information and register addresses are obtained from the IR decode. The register numbers will be:

4 bits SRC Reg from byte 1 (SP1 ADR)
4 bits DST Reg from byte 2 (SP2 ADR)
4 bits SRC Reg latched from byte 1 (PRN ADR)

Note that the SRC register number will always be valid and the destination register is an assumption of S or R specifier for byte 1 followed by R specifier for the destination. The register numbers will be multiplexed for compatibility mode versus VAX mode. The multiplexer will be A 74S158 with (L)=1 polarity. The outputs will be stable at CPUT0 plus 27 N.S. (Clock skew not considered.)

4.7.1 Register Latched Number

The SRC register latched outputs will be clocked at CPU T0 during the MICRO state that does a execution point entry or a MICRO subroutine call to a specifier evaluation. This will physically be done by the MICRO control that bumps the specifier from byte 1.

4.7.1.1 Context Lookup - The context lookup information will also come from a Rom. This implies that the context information will not become available until CPU T0 plus 100 NS from the time the execution point, specifier or opcode has been changed. This information will define:

byte
Word
Long
Float
Double
Quad
ASRC
VSRC

4.7.1.1.1 Specifier 1 Constant - This constant will be used by MICRO code to do register updates for auto increment or auto decrement modes. This will be done with 4 lines connected to the fast constants multiplexer on the data path boards. The number generated for VAX mode will be:

1 - Byte
2 - Word
4 - Long/Float
8 - Quad/Double

This will be generated from the context lookup table for each execution point entry.

During compatability mode, this number will be derived from a decode of the instruction. This constant is only selected during the evaluation of the source operand. All single operand instructions and register class instructions use SP2 constants.

The number generated will either be 1 or 2. A one is generated if it is a byte instruction and the register field is not register six or seven. A two is generated if it is a word instruction or byte instruction with register six or seven.

4.7.1.1.2 Specifier 2 Constant - This constant will be used by micro code during compatibility mode flows. It is used for auto increment or auto decrement address mode updates. The number generated will be either one or two. A one is generated if it is a byte instruction and the destination register is not register six or seven. A two is generate if it is a word instruction with destination register six or seven.

During VAX mode, this number will be zero.

4.7.1.2 Data Length Field - A three bit field is generated from the operand context table to control the UDT, scratch pad control or AMX sign extention. The following numbers are generated.

1 - 8 bit Data Type (Byte)
2 - 16 bit Data Type (Word)
4 - 32 bit Data Type (Long, Float, Quad, Double)

Also generated is a signal that indicates the instruction opcode is either floating or double floating type. This will be used to determine condition code setting.

4.8 EXECUTION POINTS

Execution points are places in the micro flows where the IR decode logic directs instruction execution. Execution point entries are of two basic forms. The entry point will either evaluate an operand specifier, or enter an execution flow unique to the present instruction.

The mechenism for doing a execution point entry is by making the USUB field equal to three. In order to make the instruction buffer do the proper operation, the UIBC field must be set to F. In order to get the PC at the correct valve, the UPCK field must be set to F. The UMCT field must be set up to allow IB data to the ID BUS and the Q register must be clocked.

This micro control selection will perform the necessary functions to evaluate an operand specifier. The Q register will receive any literal, address or displacement data. The PC will be updated to point to the beginning of the next specifier or opcode. The instruction buffer will advance over the data transferred into the Q register.

If the entry was to an execution flow, the hardware will force the PC update to be zero and the instruction buffer to hold.

4.9 FIRST PART DONE

This is a flag that is set by micro code that indicates an interrupt was taken during the middle of an instruction. At the completion of the interrupt service routine, the instruction is fetched a second time. Instead of evaluating specifiers again, the IR decode will enter an execution flow.

This done by setting the execution point count to 7 with First Part Done = 1. This implies that interruptable instructions have a maximum of 7 execution points.

In order to get to the next instruction, the micro code must load a new address into the IBA and flush the IB. Before doing so, it must clear First Part Done.

4.9.1 IB Addressing

The address registers for the instruction buffer memory cycles are located in the data path and translation buffer. The data path holds the virtual address of the IB. The translation buffer has the physical address for the IB.

The virtural address is used for two reasons. First is for error reporting. The other use is when a page boundary is crossed. The virtural address is used to check the translation buffer for the new page.

The physical address register is loaded from the output of the translation buffer. After each reference, the address is incremented. If a page boundary is not crossed, the address must still be valid. If a page boundary is crossed, the hardware forces a translation to occur and the physical address is loaded.

If the translation fails, the data locations are flagged in the instruction buffer. When the code requests this data, a TB fault is taken. The code then interperts the cause.

Both the virtural and physical addresses count by 4 bytes at a time. The control of this counting is done by the IB.

4.10 CACHE INTERFACE

The interface to the cache is done with the MD BUS and several control signals.

1. IB request H - Asserted when there is room to put data in the instruction buffer and the address registers are not counting.

2. IB READ DATA L - Asserted when data is to be loaded into the IB. It is inhibited during flush states.

3. Count H - Asserted during the micro state following a read operation from cache to IB. It is used to count both the virtural and physical address register.

4.11 ACCELERATOR INTERFACE

The interface to the floating point accelerator is done by tracking the IR decode. The instruction opcode and the three bits of execution point count are sent to the FPA. The FPA will decode opcodes that it wishes to operate on. The execution point count allows it to decide when to jam the MICRO program into WCS. The restriction to this interface is that the FPA must follow the same combinations of specifier evaluations and execution entries as the CPU. Also, no new instructions that require specifier calculations can be implemented without modification to the IR decode logic.

In order to do optimizations of register to register and short literal to register, some additional information is passed:

1. DST R Mode H - Indicates that specifier two (if any) is R mode and R is not PC.

2. IMMED L - Indicates that specifier one is (PC)+.

3. VAX SL L - Indicates that specifier one is a short liTeral.

4. R Mode H - Indicates that specifier one is R mode and R is not PC.

5. B0 VAL (1) H - Indicates that the eight bits of opcode contain valid information.

6. B1 VAL (1) H - Indicates that the specifier position in the instruction buffer contains valid data. specifier are valid.

CHAPTER 5

INTERRUPTS & EXCEPTIONS

This chapter discusses interrupts, exceptions, machine halts and the UTRAP function.

5.1 INTERRUPTS

Interrupts are the notification of events in the system which require a change in the flow of control and are generally independent of the currently executing process. They are characterized by the following:

1. Interrupts always occur at the end of instructions or during well defined points of long iterative instructions.

2. Interrupts always push two long-words of state on either the kernel or interrupt stacks consisting of the PSL and PC of the next instruction.

3. Interrupts always cause the processor to raise its Interrupt Priority Level (IPL) to that of the highest Interrupt Priority Request (IPR).

5.1.1 Interrupt Priority Level (IPL)

There are 32 IPL levels defined within the processor and for an IPR to be serviced it must be greater than the IPL level of the current process. The IPL is a five bit field in the Processor Status Longword register (PSL).

There are several methods for changing the current IPL level:

1. An MTPR instruction is executed in kernel mode to the IPL register. This will load a 5 bit number defined in the instruction operand into the IPL field of the PSL.

2. The IPL field of the PSL is loaded from the stack in the execution of the REI instruction.

3. The IPL field is set to one if current IPL is zero and the process is not executing on the interrupt stack (PSLIS=0) in the execution of the SVPCTX instruction.

4. The IPL field is set to the highest Interrupt priority request active (IPRA) level during the execution of the interrupt sequence.

5. The IPL field is set to IPL 1F (highest priority) in the execution of the exception sequence for Kernel Stack Not Valid and Machine Check Faults.

5.1.2 System Control Block

The system control block is a page in memory containing the vectors by which Interrupts and Exceptions are dispatched to the appropriate service routines. The system control block page is pointed to by the System Control Block Base reg (SCBB) located in the internal processor register space and accessed with the MTPR and MFPR instructions.

The processor micro code insures that bit 31 and bits 8 thru 0 of the SCBB are always loaded with zeroes.

5.1.3 Vectors

A vector is a longword in the system control block which is used to point to the interrupt or exception service routine and which describes how the event is to be serviced. The vector is formed by adding the contents of the SCBB register to a nine bit hardware generated vector which is dependent on the event being serviced.

The low two bits of the contents of the vector fetched from the system control block contain a code which indicates how the event is to be serviced as follows:

Code	Operation
0	Service event on the kernel stack unless already on the Interrupt stack (PSL IS=1)
1	Service event on the Interrupt stack
2	Service event in WCS. Bits 15:02 are a parameter to service routine
3	HALT

5.1.4 Interrupt Requests and their Vectors

Interrupt requests are sampled by the micro code during the execution of each instruction and if the IPR level is greater than the IPL and no exceptions occur during the instruction a branch at FORK A in the flows is taken to the interrupt micro-code service routine. Each interrupt request signal is a level which is sampled at CPT150 time in certain micro-instructions and is then prioritized and the highest priority request level is compared with the PSL IPL level.

The hardware will generate the nine bit interrupt dependent vector in the Vector register which is available to micro-code but not to macro-code software. For SBI or Unibus interrupts the micro-code polls devices on the IPR level being serviced using the Interrupt Summary Read command. A sub-level bit mask of devices with pending interrupts at that IPR level is returned and written into the Vector register. Bits 31 thru 16 are prioritized in the Vector register according to lowest bit set has highest priority and are used to generate the 9 bit vector when a device interrupt is being serviced. Refer to section 1.7 , Registers used for Interrupt Servicing.

Interrupt priority request occur on 31 levels with IPR 1F the highest priority to IPR01 the lowest. IPR00 does not exist since the priority request must be greater than the IPL level of the processor to be serviced. The following is a list of interrupt conditions and their assigned vectors from highest priority to lowest priority.

INTERRUPT PRIORITY REQUESTS

Level	Condition	Vector
IPR 1F	NONE ASSIGNED	---
IPR 1E	CPU POWER FAIL	0C
IPR 1D	CPU TIMEOUT	60
IPR 1C	SBI FAULT	5C
IPR 1B	SBI ALERT	58
IPR 1A	CRD/RDS	54
IPR 19	SBI SILO COMPARE	50
IPR 18	INTERVAL TIMER	C0
IPR 17	SBI REQ7/UNIBUS BR7	1C0-1FC
IPR 16	SBI REQ6/UNIBUS BR6	180-1BC
IPR 15	SBI REQ5/UNIBUS BR5	140-17C
IPR 14	SBI REQ4/UNIBUS BR4	100-13C
	CONSOLE TERM. REC.	F8
	CONSOLE TERM. XMIT.	FC
IPR 0F	SOFTWARE REQ 0F	BC
IPR 0E	" 0E	B8
IPR 0D	" 0D	B4
IPR 0C	" 0C	B0
IPR 0B	" 0B	AC
IPR 0A	" 0A	A8
IPR 09	" 09	A4
IPR 08	" 08	A0
IPR 07	" 07	9C
IPR 06	" 06	98
IPR 05	" 05	94
IPR 04	" 04	90
IPR 03	" 03	8C
IPR 02	" 02 or AST DEL.	88
IPR 01	" 01	84

5.1.5 Description of Interrupt Conditions

5.1.5.1 CPU Power Fail - This interrupt occurs if a power fail warning is received for the processor (AC LO) or from a critical system element (SBI FAIL). Critical system elements include the SBI Clock circuitry, SBI terminators, Bootstrap Memories or Main Memory (in the standard configuration). There will be a guaranteed 5 millisecs of good power after the assertion of this interrupt. This interrupt may occur immediately after a power up sequence has begun; however, at least 5 millisecs is guaranteed from the assertion of the Power Fail until the possibility of a power up sequence is allowed.

This interrupt is cleared by hardware/micro-code and requires no macro-code intervention.

5.1.5.2 CPU Timeout - This interrupt occurs if the processor attempts to write data into a non-existing physical address or receives an ERR confirmation on the SBI Bus for the second longword during an extended read operation. This may be caused by software if the memory mapping is incorrectly set up or by a hardware failure.

NOTE

CPU timeout interrupts do not necessarily occur during the instruction which caused them since the processor is allowed to continue execution while an SBI write cycle is pending.

No additional state information is saved on the stack other than the PC and PSL at the time of the interrupt. Error status will be latched in the SBI ERROR and SBI TIMEOUT ADDRESS registers. This interrupt is cleared by macro-level software by a write ones to clear operation on the CP TIMEOUT or IB TIMEOUT bits in the SBI ERROR register.

5.1.5.3 SBI Fault - The SBI Fault interrupt occurs if an SBI bus error was detected by any device on the bus including the processor. If the processor detects a fault condition which prevents the completion of a read cycle for the CPU an exception condition is also generated. Generally this appears as a Read Timeout Machine Check exception.

This interrupt is cleared by macro-level software by clearing the Fault Interrupt bit in the processor's FAULT/STATUS register which also unlocks the Fault status in each device. This interrupt will occur only if enabled in that register.

5.1.5.4 SBI Alert - This interrupt occurs when a device which does not contain SBI Request sequencing logic wishes to interrupt the processor and may be caused by device power fail, device power up, or dangerous environmental conditions in the device. Currently main memories in non-standard configurations use this interrupt to report changes in it's power status.

This interrupt is cleared by macro-level software by clearing the Alert status bits in each device's configuration register.

5.1.5.5 CRD/RDS - The Corrected Read Data (CRD) interrupt is asserted if the processor received read data which had been corrected by main memory. The Read Data Substitute (RDS) interrupt is asserted if the processor received uncorrected read date on a read cycle to the Instruction Buffer. If during the execution of instructions from the Instruction Buffer a change in program flow is encountered (branch, jump, etc.) before the bad data is used this interrupt will remain asserted. However, if an attempt is made to use the bad data an exception will occur and the RDS interrupt removed.

These interrupts occur only if enabled in the processor's SBI ERROR register and must be cleared by macro-level software by writing ones to clear to the CRD,RDS, or IB RDS bits in that register.

Additionally the software must clear the error condition in the memory configuration register if continued logging of errors and recording of address information is desired.

5.1.5.6 SBI SILO Compare - This interrupt occurs when a match is detected on particular signal fields of the SBI bus. The signal field being checked can be program selected by control bits in the SILO COMPARATOR register. The previous 15 cycles on the SBI bus will be latched in the SILO register for interrogation by software.

This interrupt will occur only if enabled in the SILO COMPARATOR register and must be cleared by macro-level software by writing a one to clear to the Silo Lock bit in that register.

5.1.5.7 Interval Timer - This interrupt will occur when the Interval Count register overflows. This interrupt will occur only if enabled in the Clock Control Status register and must be cleared by macro-level software by writing the Interrupt Request bit in that register.

5.1.5.8 External Device Interrupts - External device interrupts occur at IPR 17 to IPR 14 and correspond to SBI REQ7/UNIBUS BR7 to SBI REQ4/UNIBUS BR4 levels respectively. These interrupts result from device completion, device errors, and important device status changes.

When an external device interrupt is being serviced in micro-code the processor will issue an Interrupt Summary Read command at the serviced level to poll devices with pending interrupts at that level. A bit pair mask is returned on bits 31, 15 to bits 17, 01 indicating devices needing service.

The hardware/micro-code will prioritize the returned data according to lowest bit set is highest priority and generate an interrupt vector for that level and highest priority device.

Generally, device interrupts must be enabled and always require macro-level software intervention to clear the interrupt.

5.1.5.9 Console Terminal Interrupts - The console terminal receive interrupt occurs when the Done bit in the RXCS register is set. The interrupt enable bit must be set in that register for the interrupt to occur.

The console terminal transmit interrupt occurs when the RDY bit in the TXCS register is set. Likewise, the interrupt enable bit must be set in that register for the interrupt to occur. The receive interrupts has higher priority than the transmit interrupt.

These interrupts are cleared by hardware/micro-code and requires no macro-code intervention.

5.1.5.10 Software Interrupts - There are 15 interrupt priority requests for use by the software, IPR 0F to IPR 01. The Software Interrupt Summary Register (SISR) contains 1's in bit positions 15 thru 01 corresponding to levels IPR 0F to IPR 01 respectively with pending interrupts. Bits 20 thru 16 of the SISR contain the level of the highest interrupt priority request active (IPRA) of both the hardware and software levels.

Macro-level software may book a request by using the MTPR instruction and writing a bit per request desired to the SISR or by writing the level number of a request to the Software Interrupt Request Register (SIRR). Using the SIRR register to book a software request is preferred since it will not inadvertently clear other requests pending. Writing the SISR mainly occurs when restoring state information after a power fail.

Micro-code will interpret writes to the SIRR as a bit set operation to the SISR. The mask generator in the CPU data paths can be used to decode the request level desired.

During the execution of the REI instruction the micro-code compares the current mode bits in the new PSL image with the asynchronous system trap level (ASTLVL) in the ASTR register. If the two bit current mode is greater than the three bit ASTLVL an AST is delivered at IPR02. This is performed by the micro-code doing a bit set to the SISR before completing the execution of REI.

The micro-code/hardware will clear the software interrupt level being serviced by reading the SISR to determine the priority (IPRA), decoding the level in the mask generator of the data paths, and performing a bit clear operation in the SISR. No macro-level software intervention is required.

5.1.6 UWORD Control for Interrupts

There is a two bit Uword field designated Uword Interrupt and Exception control (UIEK) which is used to monitor and acknowledge interrupt conditions. The functions available are as follows:

UIEK	
0	NO-OP
1	Interrupt Strobe (ISTR)
2	Interrupt Acknowledge (IACK)
3	Exception Acknowledge (EACK)

5.1.6.1 Interrupt Strobe - The ISTR function is used to clock the state of interrupts at IPR 1F to IPR10 and cause a new priority arbitration to occur. This is genrally done in the state prior to returning to the IRD state so that the Interrupt branch can be performed at Fork A.

During the execution of long iterative instructions the ISTR function is used to periodically monitor Interrupt activity and to allow a subsequent micro branch to be performed on the Interrupt signal.

5.1.6.2 Interrupt Acknowledge - The IACK function is used to clear the Power Fail, Console Term. Rec, and Console Term Xmit interrupts when they are being serviced by the micro-code routine. In addition the PSL is set to a predetermined state as follows:

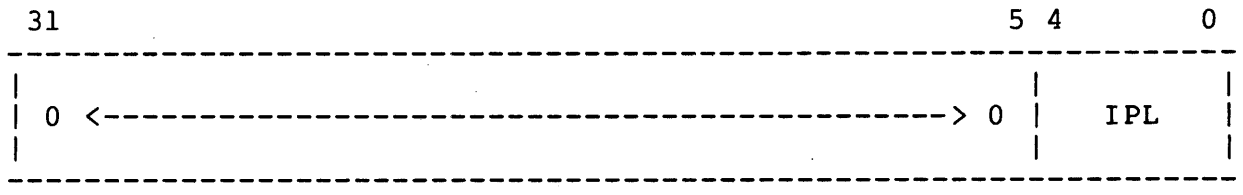
PSL:	CMP	<-- 0
	TP	<-- 0
	FPD	<-- 0
	IS	<-- IS

```
CUR MOD   <-- 0 (KERNEL)
PRV MOD   <-- 0 (KERNEL)
IPL       <-- IPRA (IPR being serviced)
DV        <-- 0
FU        <-- 0
IV        <-- 0
T         <-- 0
N         <-- 0
Z         <-- 0
V         <-- 0
C         <-- 0
```

5.1.7 Registers used for interrupt servicing

5.1.7.1 Interrupt priority level register - IPLR -

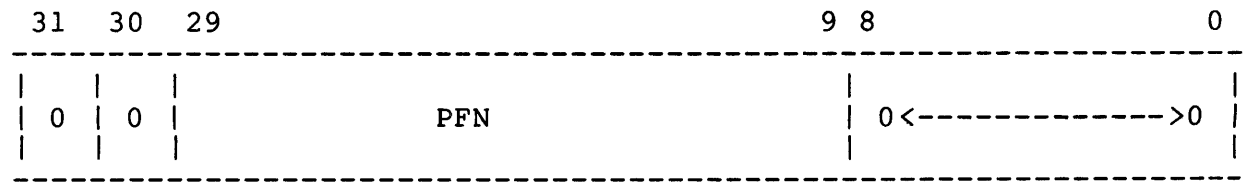
Processor Reg (PR) Address - 12
Internal Data bus (ID) Address - not an ID reg.



BITS	NAME	COMMENTS
IPL	Interrupt Priority Level	IPL field of PSL read/write

5.1.7.2 System control block base register - SCBB -

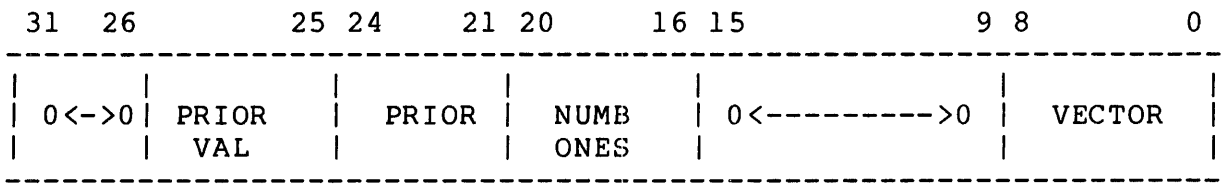
PR Address - 11
ID Address - 3B



BITS	NAME	COMMENTS
PFN	Page Frame Number	Holds the base physical address of the system control block page. Zero bits are always forced to zero when written. read/write

5.1.7.3 Vector register, VECTOR -

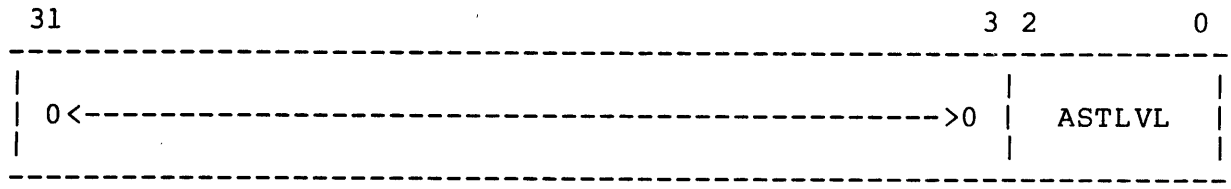
PR Address - Not available to software
ID Address - D



BITS	NAME	COMMENTS
PRIOR VAL	Priority Valid	Indicates at least one bit was set when the last PRIOR field was determined. Read only.
PRIOR	Priority Encode	The priority encoded value of the last bit mask written into bits 31 to 16 of the VECTOR register. Bit 31 represents the lowest priority (PRIOR=F) and bit 16 the highest (PRIOR=0). Used to form the Vector, bits 8:0 when IPRA indicates an external interrupt. Read Only.
NUMB ONES	Number of Ones	The number of ones in the data last written into the Vector register bits 31 to 16. Read only.
VECTOR	Interrupt Vector	A hardware generated number determined by IPRA. Read Only.

5.1.7.4 Asynchronous system trap level reg. - ASTR -

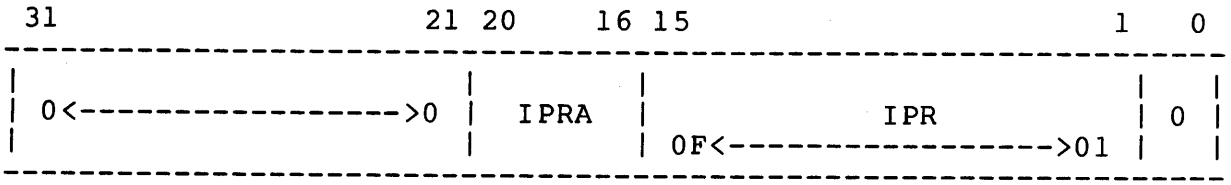
PR Address - 13
ID Address - Not an ID address



BITS	NAME	COMMENTS
ASTLVL	Asynchronous System Trap Level	Used to deliver AST interrupts in REI execution. Read/Write

5.1.7.5 Software interrupt summary register - SISR -

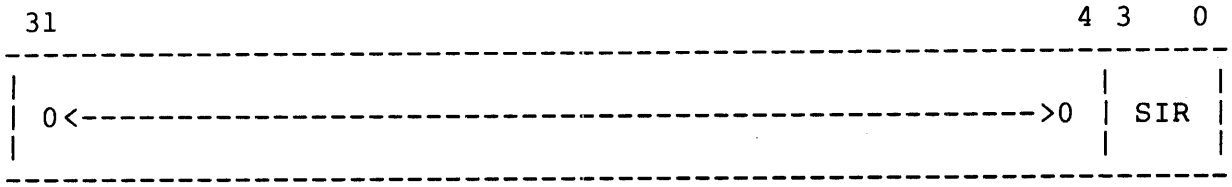
PR Address - 15
ID Address - E



BITS	NAME	COMMENTS
IPRA	Interrupt Priority Request Active	The level of the highest priority interrupt pending condition of the last Interrupt strobe time or write to SISR. Read Only
IPR 0F to 01	Interrupt Priority Request 0F to 01	Software interrupt request pending flags. Read/Write.

5.1.7.6 Software interrupt request register - SIRR -

PR Address - 14
ID Address - Not an ID register



BITS	NAME	COMMENTS
SIR	Software Interrupt Request	The level of the request t.at software wishes to book in the SISR. An SIR=0 books no request. Write Only.

5.2 EXCEPTIONS

Exceptions are the notification of events which force a change in the flow of control for the currently executing process. Their characteristics are as follows:

1. Exceptions occur in the middle or at -he end of an instruction during which the exception condition had been detected.

2. Exceptions always push two long-words of state on either the kernel or interrupt stacks consisting of the PSL and PC of the instruction (sometimes the PC of the next instruction). Additionally up to 16 longwords of exception parameter information may be pushed.

3. Exceptions generally do not change the processor's IPL level. Only for Kernel Stack Not Valid and Machine Check Faults is the IPL altered in which case it is forced to IPL 1F (highest priority).

4. Exceptions which occur while disabled do not cause an exception when subsequently enabled. Arithmetic traps are the only exceptions which can be disabled.

5.2.1 Classes of exceptions

Exceptions may fall into three categories depending upon when they occur and how they leave the machine state. The three classes are Traps, Faults, and Aborts.

5.2.1.1 Traps - A trap is an exception condition which occurs at the end of the instruction. The PC saved on the stack is that of the next instruction.

5.2.1.2 Faults - A fault is an exception which occurs in the middle of an instruction, but which leaves memory and the general registers in the same state as at the beginning of the instruction. The PC saved on the stack is that of the instruction in which the fault was detected so that the instruction may be restarted when the fault condition is eliminated.

5.2.1.3 Aborts - An Abort is an exception which occurs in the middle of an instruction but potentially leaves the registers and memory indeterminate, such that the instruction cannot be correctly restarted. The PC saved on the stack does not necessarily point to the beginning of the next instruction.

5.2.2 Exception conditions and their vectors

Exception conditions are detected thru the use of two mechanisms in the micro sequencing control:

1. ubranches - a unique micro state is reached by performing a test of an exception condition using the UBEN field of the Uword or by using the special micro call function to branch on a decision point (DPT).

2. utraps - a hardware detected micro trap occurs which alters the normal machine flow and causes unique micro service routines to be executed.

Each of the micro service routines will manufacture the exception vector called for using a generator set of constants from the UKMX function of the Uword. Likewise, the micro service routines will put together the necessary parameters to be saved on the stack from state information stored within the processor registers. Exception codes will be furnished by the UKMX function.

The following is a list of exception conditions, their class, and their assigned vectors. In most cases the exception conditions are mutually exclusive; however, in cases of conflict the exception condition detected by the utrap function will have higher priority.

EXCEPTION CONDITIONS

Condition	Vector	Class
MACHINE CHECK	04	FAULT/ABORT
KERNEL STACK NOT VALID	08	ABORT
RESERVED DEC OPCODES& PRIVILEGED INSTRUCTIONS	10	FAULT
RESERVED CUSTOMER OPCODES	14	FAULT
RESERVED OPERANDS	18	FAULT/ABORT
RESERVED ADDRESSING MODES	1C	FAULT
ACCESS CONTROL VIOLATION	20	FAULT
TRANSLATION NOT VALID	24	FAULT
TRACE TRAP	28	FAULT
BPT OPCODE	2C	FAULT
COMPATABILITY MODE TRAP	30	TRAP/ABORT
ARITHMETIC TRAP	34	TRAP
CHMK OPCODE	40	TRAP
CHME OPCODE	44	TRAP
CHMS OPCODE	48	TRAP
CHMU OPCODE	4C	TRAP

5.2.3 Description of exception conditions

5.2.3.1 Machine check - Raises IPL to 1F - Machine check exceptions push additional parameters onto the stack to assist in the evaluation of the condition causing the abort. Refer to Chuck Mathis' MACHINE CHECK DESCRIPTION AND SPECIFICATION for a list of the parameters.

5.2.3.1.1 Read timeout - Read timeouts occur when the processor is performing a read or interlock read command on the SBI bus.

This exception is detected by the utrap function for data path cycles and by the ubranch function for IBUF cycles.

5.2.3.1.2 Read data substitute - Read Data Substitute errors occur when the processor is performing a read or interlock read on the SBI bus and the memory has returned uncorrected read data.

This exception is detected by the utrap function for data path cycles and by the ubranch function for IBUF cycles.

5.2.3.1.3 Translation buffer parity error - TBUF parity errors occur when the processor is performing a virtual address reference and memory mapping is enabled and a parity check on the translation buffer indicates an error.

This exception is detected by the utrap function for data path cycles and by the ubranch function for IBUF cycles.

5.2.3.1.4 Cache parity error - Cache Parity errors occur when the processor is performing a read memory reference and a parity check of the cache indicates an error.

This exception is detected by the utrap function for data path cycles and by the ubranch function for IBUF cycles.

5.2.3.1.5 Control store parity error - This error condition occurs when the micro machine detects a parity error in the next micro-instruction and may cause a Machine Check Abort at any time (including during the halt state while in console wait).

This exception is detected by the utrap function.

5.2.3.1.6 Illegal Machine Sequence Error - This error condition occurs if an illegal micro-instruction is reached and will generally indicate a hardware failure in the instruction decode circuitry.

This exception is detected by the ubranch function.

5.2.3.2 Kernel stack not valid - Raises IPL to 1F - This exception is reported if a Translation Not Valid or Access Control Violation Fault would have occurred while pushing onto the Kernel stack in the exception, interrupt, or CHMX micro flows. The vector for this exception should define servicing on the interrupt stack.

No additional parameters are pushed for this exception. This exception is detected by the ubranch function.

5.2.3.3 Reserved DEC opcodes & priv. instr - Reserved Dec Opcodes are the following: 36, 37, 57 to 5F, 77, EF, FD to FF. Privileged Instructions are the following: Not Kernel Mode and HALT, MTPR, MFPR, LDPCTX, SVPCTX.

No additional parameters are pushed for this exception. This exception is detected by the ubranch function.

5.2.3.4 Reserved cust opcodes - Reserved Customer Opcodes are FC. No additional parameters are pushed for this exception. This exception is detected by the ubranch function.

5.2.3.5 Reserved operands - No additional parameters are pushed onto the stack. The RSVD OPERAND type can be determined by the OPCODE pointed to by the pushed PC.

5.2.3.5.1 Illegal floating number - Fault - A floating operand with sign=1 and exponent=0.

Occurs in: MOVF, MOVD, MNEGF, MNEGD, CVTFX, CVTDX, CVTRFL, CVTRDL, CMPF, CMPD, TSTF, TSTD, ADDF(2,3), ADDD(2,3), SUBF(2,3), SBUD(2,3), MULF(2,3), MULD(2,3), DIVF(2,3), DIVD(2,3), EMODF, EMODD, POLYF, POLYD, ACBF, ACBD.

This exception is detected by the utrap function when the UMSC field calls for - Check Float Operand.

5.2.3.5.2 Bit field too wide - Fault - Size operand is greater than 32 or less than 0 or when the bit field is located in a register with position operand greater than 31 or less than 0.

Occurs in: EXTV, EXTZV, INSV, CMPV, CMPZV, FFC, FFS, BBS, BBC, BBSS, BBSC, BBCS, BBCC, BBSSI, BBCCI.

This exception is detected by the ubranch function.

5.2.3.5.3 Illegal entry mask - Fault - Unspecified

Occurs in: CALLG, CALLS

This exception is detected by the ubranch function.

5.2.3.5.4 PSW MBZ FIELD not zero - Fault - Bits 15:08 of the new PSW value is non-zero.

Occurs in: RET, BISPSW, BICPSW

This exception is detected by the ubranch function.

5.2.3.5.5 Illegal PCB entry - Abort - The MBZ fields of the PCB+84 and PCB+92 are non-zero.

Occurs in: LDPCTX

This exception is detected by the ubranch function.

The PC pushed on the stack points to the opcode.

5.2.3.5.6 Illegal PSL image - Fault - The new PSL from the stack did not have the correct format.

Occurs in: REI

This exception is detected by the ubranch function.

5.2.3.5.7 Illegal processor reg - Fault - The internal processor register address does not exist.

Occurs in: MTPR, MFPR

This exception is detected by the ubranch function.

5.2.3.5.8 Decimal string too long - Fault - Length operand is greater than 31 or less than 0.

Occurs in: MOVP, CMPP(3,4), ADDP(4,6), SUBP(4,6), MULP, DIVP, CVTLP, CVTPL, CVTPN, CVTNP, ASHP.

This exception also occurs if an invalid numeric digit (an ASCII character other than 0 thru 9) is encountered.

Occurs in: CVTNP

This exception is detected by the ubranch function.

5.2.3.5.9 Reserved pattern operator - Fault - More input digits or less input digits are requested by the pattern than are specified or an unimplemented or reserved pattern operator is encountered.

Occurs in: EDITPC

This exception is detected by the ubranch function.

The PC pushed on the stack points to the opcode.

5.2.3.6 Reserved addressing modes - Fault - One of the following addressing modes was encountered during the evaluation of an operand specifier:

Specifier	Situation Illegal
Short Literal Mode	Modify, write, address source, or within index mode.
Register Mode	Address source, or within index mode.
Index Mode	Within index mode, or with PC as index.

No additional parameters are pushed for this exception.

This exception is detected by the ubranch function.

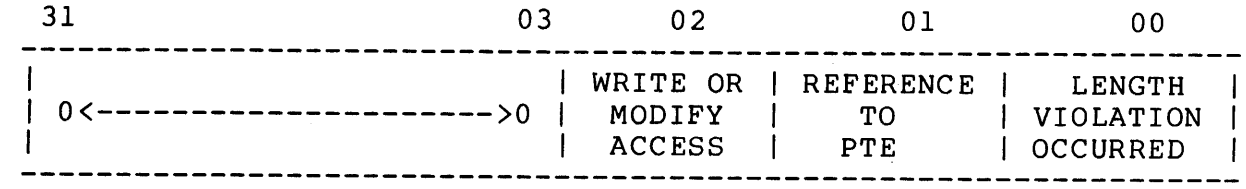
5.2.3.7 Access control violation - Fault - The exception occurs when the processor is performing a virtual reference and the protection code for that page found in the translation buffer does not allow access for the type of reference being performed (read or write). In the current mode (Kernel, Exec, Super., or User).

An access control violation also occurs if during the translation process the micro-code discovers that the page frame number of the Virtual Address (VA(29:09) for per process address space and VA(30:09) for system address space) is outside the bounds as specified in that address space's length register.

Additional parameters pushed include:

1. Virtual Address - VA reg. for data path cycles & VIBA for IBUF cycles.

2. Fault Parameter - location in microflows & ubranch.



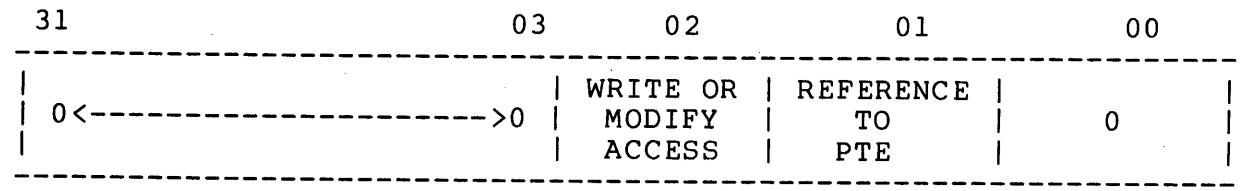
The protection violation is detected by the utrap function for data path cycles and the ubranch function for IBUF cycles. The length violation is detected by the ubranch function and the virtual address is always located in VA.

5.2.3.8 Translation not valid - Fault - The exception occurs when the processor is performing a virtual reference and if during the translation process (due to a TBUF miss utrap) an invalid page table entry (VALID bit=0 in PTE) is encountered.

Additional parameters pushed include:

1. Virtual Address - VA register.

2. Fault Parameter - location in microflows & ubranch.



The translation process is begun by the utrap function for data path cycles and by the ubranch function for IBUF cycles. If an IBUF cycle needs translation then the VIBA is placed in the VA register and the process begun. If during the translation process this exception is detected by the ubranch function the VA register always contains the correct virtual address.

5.2.3.9 Trace trap - TRAP - This exception occurs at the end of every instruction which has the T bit in PSL set at the beginning of the instruction. If enabled this trap must occur even if exception or interrupts occur for the instruction being executed.

The following instructions handle Trace Traps in special ways:

RET, CHMX, REI, BISPSW, & BICPSW, CALL

The trap occurs at Fork A using the ubranch function if the trap pending TP bit in PSL was set upon entering the IRD micro-state. At the end of the IRD micro-state the T bit is sampled and if set the TP bit is set. Microcode can set or clear TP and T during the execution of instruction which handle Trace Traps in a special way to get the proper results.

No additional parameters are pushed for this exception.

5.2.3.10 BPT opcode - FAULT - This trap occurs when the BPT opcode is encountered in the instruction stream.

No additional parameters are pushed.

This condition is detected by the ubranch function.

5.2.3.11 Compatability mode trap - TRAP/ABORT - This trap occurs when a reserved opcode or an illegal instruction is encountered when executing instructions in compatability mode. The following is a list of opcodes (in octal) which trap:

HALT, WAIT, RESET, SPL, MARK, FADD, FSBU, FMUL, FDIV, 17XXXX, 000007, 000077, 000210 to 000227, 007000 to 007777, 075040 to 076777, 1064000 to 106477, 106700 to 107777, IOT, BPT, EMT, TRAP, JMP*DM0, JSR*DM0.

In addition a Compatability mode Abort may occur if an odd address error is detected during the following memory references when in Comp. Mode:

1. Any reference with VA00=1 and not a byte instruction being executed.

2. Any opcode fetched from an unaligned word address.

3. The address fetch in the evaluation of addressing modes 3, 5, and 7.

4. The index word fetch in the evaluation of addressing modes 6 and 7.

Additional parameters pushed include:

1. Trap Code: - location in micro-flows

#0	RSVD or Illegal Opcode
#1	IOT opcode
#2	BPT opcode
#3	EMT opcode
#4	TRAP opcode
#5	ODD ADDRESS ERROR

Detection of the special opcodes for this exception is accomplished with the ubranch function and will result in a Trap.

Odd address errors are detected using the utrap function and will cause an ABORT since the PC pushed onto the stack is not necessarily that of the next instruction and that the instruction cannot be restarted.

5.2.3.12 Arithmetic trap - TRAP - This trap occurs when an overflow or underflow condition is detected during the execution of instructions and the particular trap condition for that instruction has been enabled by the DV, FU, and IV enable bits in the PSL. The instruction is always completed.

The following trap conditions are detected for the instructions indicated:

1. Integer Overflow - Enabled by IV

   Occurs in: MNEG B,W,L; CVT WB,LB,LW; ADD B(2,3), W(2,3), L(2,3); INC B,W,L; ADWC; SVB B(2,3), W(2,3), L(2,3); DEC B,W,L; SBWC; MUL B(2,3), W(2,3), L(2,3); DIV B(2,3), W(2,3), L(2,3); EDIV; ASHL; ASHQ; CVTF B,W,L; CVTD B,W,L; CVTRFL; CVTRDL; EMODF; EMODD; ACB B,W,L; AOBLEQ; AOBLSS; SOBGEQ; SOBGTR; CVTPL.

2. Integer Divide by Zero - Always Enabled

   Occurs in: DIV B(2,3), W(2,3), L(2,3); EDIV

3. Floating Overflow - Always Enabled

   Occurs in: CVTDF; ADD F(2,3), D(2,3); SUB F(2,3), D(2,3); MUL F(2,3), D(2,3); DIV F(2,3), D(2,3); PLOY F,D; ACB F,D.

4. Floating Divide by Zero - Always Enabled

   Occurs in: DIV F(2,3), D(2,3).

5. Floating Underflow - ENabled by FU

   Occurs in: ADD F(2,3), D(2,3); SUB F(2,3), D(2,3); MUL F(2,3), D(2,3); DIV F(2,3), D(2,3); EMOD F,D; PLOY F,D; ACB F,D.

6. Decimal Overflow - Enabled by DV

   Occurs in: EDITPC; ADDP(4,6); SUBP(4,6); MULP; DIVP; CVTLP; CVTPN; CVTNP; ASHP;

7. Decimal Divide by Zero - Always Enabled

   Occurs in: DIVP

Additional parameters pushed include:

1. Trap Code: - CPU ERROR/STATUS REG

#1	Integer Overflow
#2	Integer Divide by Zero
#3	Floating Overflow
#4	Floating Divide by Zero
#5	Floating Underflow
#6	Decimal Overflow
#7	Decimal Divide by Zero

This trap is detected by the ubranch function.

5.2.3.13 CHMX opcodes - At the completion of execution of the CHMK, CHME, CHMS, and CHMU instructions a trap is performed.

Additional parameters pushed include:

1. Sign extended operand - D register.

This trap is detected by the ubranch function.

5.2.4 Acknowledging exceptions

Most of the exception conditions do not require special action by the micro-code to complete the exception service code. Certain errors, arithmetic traps, and trace traps do, however, require special servicing.

5.2.4.1 Error acknowledging - The micro-code must clear the following error status bits in the indicated register for error logging to continue.

1. Read Timeout - SBI ERROR REG.
2. READ DATA SUBSTITUTE - SBI ERROR REG.
3. TBUF PARITY ERROR - TB1 REG.
4. CACHE PARITY ERROR - CACHE PARITY ERR REG.
5. CS PARITY ERROR - CPU ERR/STATUS REG.

5.2.4.2 Arithmetic trap acknowledging - The micro-code must clear pending arithmetic traps after they are serviced or when other exceptions occur, particularly those which restart the instruction.

The method for clearing these conditions will be by writing zeroes to the trap code in the CES register.

5.2.4.3 Trace trap acknowledging - The micro-code must clear the TP bit in the PSL to allow the next instruction to be executed without another Trace Trap occurring first.

In order that exception conditions can properly be serialized the TP bits must be set to the proper state when pushing the PSL on the stack if a Trace Trap is pending while servicing other exceptions.

5.2.4.4 UWORD control for exceptions - The UIEK field of the UWORD provides an exception acknowledge function (EACK) which sets the PSL into the following state:

```
PSL:        CMP         <-- 0
            TP          <-- 0
            FPD         <-- 0
            IS          <-- IS
            CUR MOD     <-- KERN
            PRV MOD     <-- CUR MOD
            IPL         <-- IPL
            DV          <-- 0
            FU          <-- 0
            IV          <-- 0
            T           <-- 0
            N           <-- 0
            Z           <-- 0
            V           <-- 0
            C           <-- 0
```

5.3 MACHINE HALTS

The halt state of the machine is defined as the micro-machine being in the CONSOLE WAIT state and responding only to console commands. If an exception occurs while executing any console command or while in the Console Wait state an exception routine will be initiated in which a branch is performed on a CONSOLE COMMAND MODE flag to prevent pushes to the stack.

5.3.1 Halt conditions

5.3.1.1 Halt Instruction - If a HALT instruction is executed while in Kernel Mode (PSL CURMOD=0) the machine halts.

5.3.1.2 CNSL halt - At any time during the execution of instructions the console may request that the machine come to a HALT. This is performed by a CNSL HALT REQUEST being asserted and serviced as an interrupt which does not push onto the stack nor change the IPL level. A CNSL HALT is serviced below Exceptions and above all other Interrupts and only at the end of instructions.

5.3.1.3 CHMX instructions - If a CHMK, CHME, CHMS, or CHMU instruction is encountered and the IS bit in PSL=1 then the machine comes to a halt before any execution of the instruction occurs.

5.3.1.4 Interrupt stack not valid - If a Translation Not Valid or Access Control Violation Fault would have occurred while pushing onto the Interrupt stack in the exception or interrupt micro code service flows a machine halt occurs with INTR STACK NOT VALID reported.

5.3.1.5 Halt code from vector - If an Interrupt or Exception occurs and the halt code is found in the Vector location then the CPU will halt after pushing the appropriate parameters onto the stack.

5.4 UTRAP FUNCTION

During the execution of micro-instructions the hardware detects certain error conditions and initiates a trap in micro-code to one of several service flows. The micro-PC is pushed onto the micro stack so that certain micro processes can be continued.

5.4.1 UTRAP conditons and their vectors

The following is a list of conditions and their relative priority (first is highest) which cause a utrap:

	CONDITION	VECTOR
1.	SYSTEM INIT	100
2.	ERRORS:	
	CS PARITY	10F
	ODD ADDRESS	10E
	TIMEOUT	10D
	READ DATA SUBSTITUTE	10C
	CACHE PARITY ERROR	108
	TBUF PARITY ERROR	107
3.	RESERVED FLOATING OPERAND	106
4.	TBUF MISS	105
5.	PROTECTION VIOLATION	104
6.	MBIT	103
7.	PAGE BOUNDARY	102
8.	UNALIGNED DATA	101

5.4.2 Description of utrap conditions

5.4.2.1 System Init - DC LO is asserted for the CPU or DEAD is received from the SBI bus.

5.4.2.2 Errors - Description of errors can be found in the sections on Exceptions.

5.4.2.3 Reserved Floating Operand - This condition occurs when the UMSC field of the Uword has the CHECK FLOAT OPERAND function and ALU15=1 with ALU(14:07)=0.

5.4.2.4 TBUF Miss - This occurs when the processor is doing a virtual reference with memory mapping enabled (MME=1) and there is no address translation in the TBUF.

5.4.2.5 Protection Violation - This occurs when the processor is doing a virtual reference with memory mapping enabled (MME=1) and the TBUF entry indicates the page being referenced is protected from the mode and reference type being used.

5.4.2.6 MBIT - This occurs when the processor is doing a virtual write reference with memory mapping enabled (MME=1) and the TBUF entry indicates the the page should be marked as being modified.

5.4.2.7 Page Boundary - This occurs when the processor is doing a virtual reference with memory mapping enabled (MME=1) and the data type will cause a reference across a page boundary. The micro-code will then probe the next page to insure that a Translation Not Valid or Access Violation Fault will not occur.

5.4.2.8 Unaligned Data - This occurs when the processor is doing any memory reference and the data type indicates a second reference is required. Quad and Double data types are treated as two sequential longword references so that two UNALIGNED DATA uTRAPS will occur if the data is not on a longword boundary.

5.5 SERIALIZATION OF EVENTS AT FORK A

The first decision point branch after the IRD state is used to perform Arithmetic and Trace Traps occurring for the previous instruction. In addition console halt requests and Interrupts are sampled. If one of these conditions are to be serviced the PC must be backed up since it was advanced in anticipation of executing the next instruction.

The following list is the priority of branches taken at Fork A so that the proper trap and interrupt sequencing can occur.

Highest listed first:

1. ARITHMETIC TRAP
2. CNSL HALT REQUEST
3. INTERRUPT
4. TRACE TRAP
5. IBUF STALL
6. IBUF ERRORS
7. OPCODES & SPECIFIERS

CHAPTER 6

MACHINE CHECK ABORT/FAULT/HALT

6.1 MACHINE CHECKS

A machine check function can be initiated by a hardware forced micro-trap, by the micro-code's testing for an I-buffer error on a memory reference, or by the firmware detecting a sequencing error.

A machine check can be caused by any one of the following conditions:

CONTROL STORE PARITY ERROR -

This condition occurs when the hardware detects a control store parity error while reading a micro-word. A micro-trap is used to initiate the error handling micro-code.

READ DATA SUBSTITUTE -

Read data substitute (RDS) errors occur when the processor is performing a read or interlock read on the SBI bus and the memory has returned uncorrected read data.

TRANSLATION BUFFER PARITY ERROR -

TBUF parity errors occur when the processor is translating a virtual address for a virtual memory reference (MME=1), and a parity error occurs on data read-out of the translation buffer.

CACHE PARITY ERROR -

CACHE parity errors occur when the processor is performing a read memory reference and a parity check is detected on data read out of the cache.

READ TIMEOUT/SBI ERROR CONFIRMATION -

Read timeouts and Error Confirmations occur when the processor is performing a read or interlock read command on the SBI bus and an SBI Error is detected as there is no response to the processor's read memory or I/O read reference.

NOT SUPPOSE-TO-BE-HERE -

When micro-code detects it has arrived at an illegal micro-address, it pushes that address onto the micro-stack and transfers control to the error handling micro-code.

The IPL is raised to "1F" when any one of the above conditions occur.

On CP memory reference errors, micro-traps are used to initiate the error handling micro-code. On I-Buffer memory reference errors, the micro-code detects on specifier evaluations the error and transfers control to the error handling micro-code.

See the "INTERRUPTS & EXCEPTIONS SPECIFICATION", Rev. B for addition details on the hardware implementation of the above micro-traps and "IBUF SPECIFICATION" for hardware specs for the micro-branch on IBUF memory reference errors.

6.2 INSTRUCTION ABORTS

Instructions are aborted on the following cases:

1. Control store parity error micro-traps

2. Error occurs during memory management micro-code

3. Error occurs during interrupt or exception micro-code

4. "Not suppose to be here", detected by micro-code.

The "NESTED ERROR" flag is used by users of the memory management micro-routines to tell the error handling micro-code to do an abort sequence.

6.3 INSTRUCTION FAULTS

On RDS, TBUF parity error, CACHE parity error, or a READ TIME OUT/ERROR Confirmation; the instruction being executed (or setup) will be faulted if an abort or halt situation does not exist.

This will allow selected instructions to be retryed by the operating software. For the instructions that are retryable, see Appendix A.

The "EFP" (Error First Pass) flag is left set on faults and must be reset by the operating software before continuing. Manually writing zeros to ID[SBI FAULT] clears it, and software can reset it using the MTPR (---0) instruction. It is also cleared by system initialization. (If it is not cleared, the CPU will be halted on the next machine check).

6.4 INSTRUCTION HALTS

The instruction being executed (or setup) will be halted if,

1. Processor is in Console Mode.

2. "EFP" (Error First Pass) flag is set on entry to the error handling micro-code.

3. A memory error occurs while attempting to push parameters on the kernel or interrupt stack. (Write timeout or Error Confirmation)

The "EFP" (Error First Pass) flag is set and tested by the error handling micro-code to determine the halt situation. This flag is a R/W bit in the "SBI FAULT" status register, bit 25.

6.5 ERROR LOGOUT

On Machine Checks, the Error Handling micro-code will attempt to log out the relevant status registers and the VA Register as parameters on the stack selected by the Machjine Check Exception Vector. (Suggest that software always use Interrupt stack to keep from losing error information if stack is not resident or valid.)

Also two additional longword parameters are pushed on the stack which are:

SUMMARY PARAMETER:

BYTE#0, Identification of the machine check that initiated the fault/abort sequence.

CODE

00 - CP READ TIMEOUT/SBI ERROR CONFIRMATION FAULT

02 - CP TBUF PARITY ERROR FAULT
03 - CP CACHE PARITY ERROR FAULT
05 - CP RDS FAULT
0A - IB TBUF PARITY ERROR FAULT
0C - IB RDS FAULT
0D - IB READ TIMEOUT/SBI ERROR CONFIRMATION FAULT
0F - IB CACHE PARITY ERROR FAULT
F1 - CS PARITY ERROR ABORT
F2 - CP TBUF PARITY ABORT
F3 - CP CACHE PARITY ERROR ABORT
F4 - CP READ TIMEOUT/SBI ERROR CONFORMATION ABORT
F5 - CP RDS ABORT
F6 - CP "NOT-SUPPOSE-TO BE HERE" ABORT

BYTE#1, Flag noting that a CP timeout or CP Error Confirmation interrupt was pending.

This flag will be set to a non-zero value if this interrupt was pending.

The operating software must examine the previously logged out parameters to determine and handle these error interrupts. (This possible pending interrupt has been cleared in order to handle the machine check sequence. The logged-out information is the only record that it was pending).

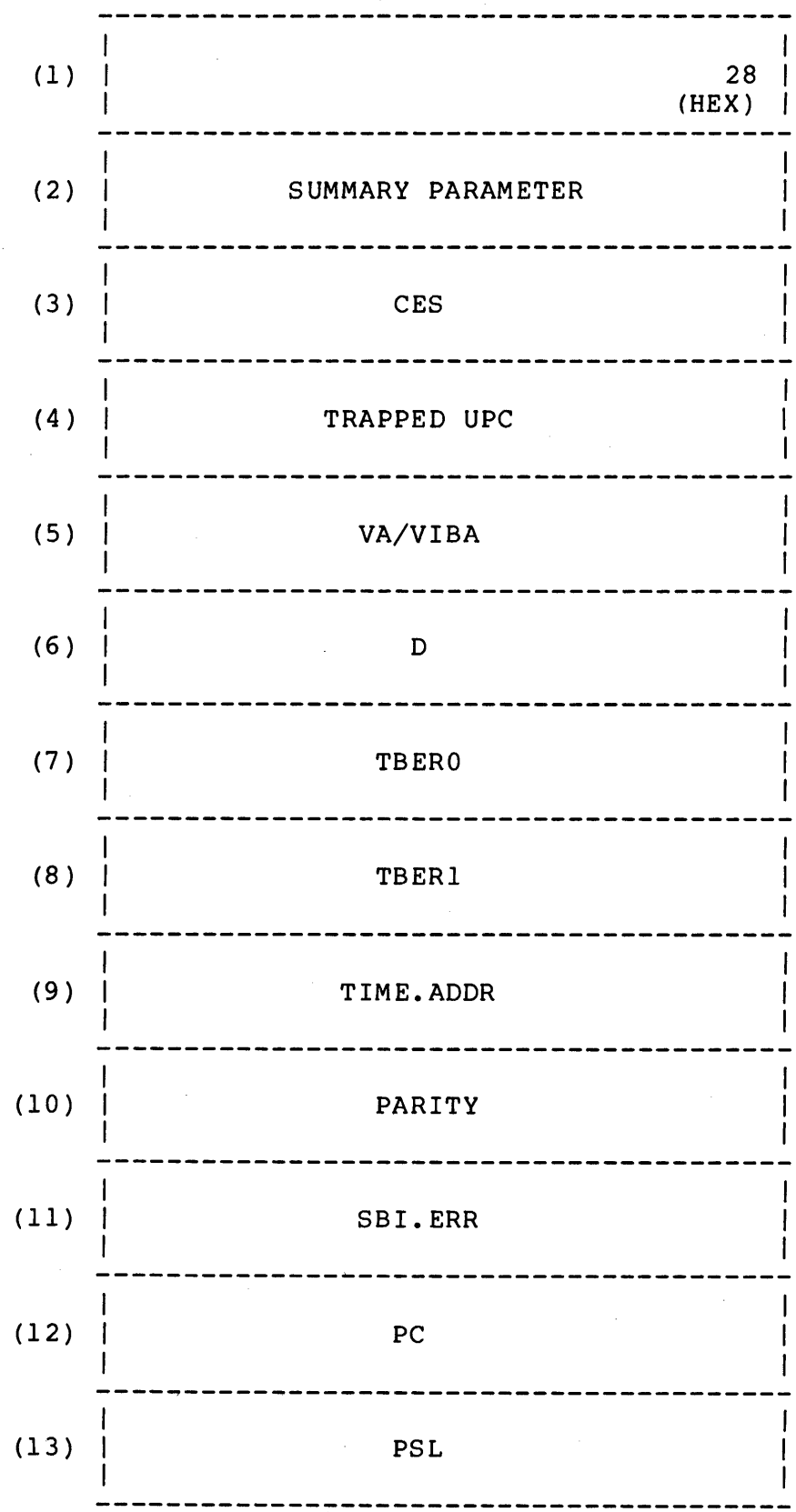
BYTES 3&4 - MBZ

LENGTH PARAMETER

BYTE#0, Number of bytes logged out exclusive of this parameter

BYTES #1-3 MBZ

The layout and contents of the logout area on the stack as follows:



ERROR LOGOUT AREA MAP/SP

6.6 INITIALIZATION OF CP, TBUF, CACHE, & SBI STATUS REGISTERS

The following status registers are initialized by the Error Handling micro-code.

TBUF ERROR REG1 (TBER1) -

Register is written to clear accumulated TBUF parity error information & FLUSH IB microrder is used to clear IB errors.

SBI ERROR REG (SBI.ERR) -

A one is written to CP Timeout also to clear CP Timeout and CPU Error Confirmation.

FLUSH.IB is used to clear I-Buffer errors.

Writing to CP timeout also unlocks the timeout address register.

CACHE PARITY REG (PARITY) -

Contents of this register are written to itself to clear the cache parity error bit. (Write one to clear type bit.)

6.7 CPU/CONSOLE INTERFACE STATE

Console Mode:

The cpu will halt after leaving the error halt code in ID[D.SV]. The information on the machine check is their respective error/status registers. "Note, Console Responsible for logging & clearing these registers".

Double Error Halt:

The cpu will halt if it finds on entry to the Error handling micro-code that "EFP" is set.

The information on the first error will be in ID[T0-T9] and U-STACK (trapped micro-addresses). See layout of ID[T0-T9]. Unpredictable on CS Parity errors.

The information on the 2nd error will be in the associated error/status registers.

The cpu will be halted after leaving a double error halt code in ID[D.SV].

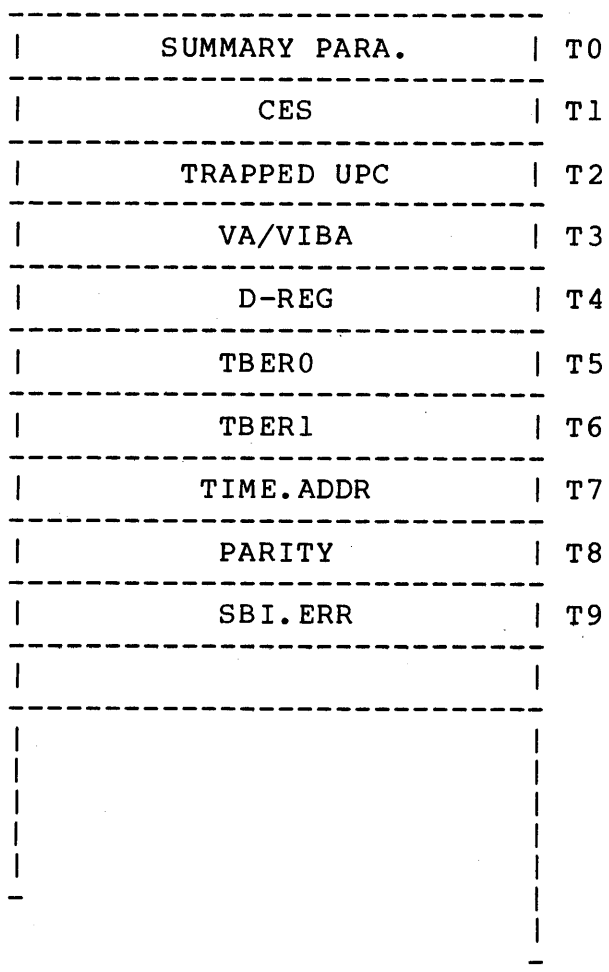


Figure 7.1 First Error Info on Double Error Halts in ID-Registers

6.8 HALT IDENTIFICATION CODES

Values left in ID[D.SV] for printout by Console whenever the CPU halts.

0 = Operation requested by console, completed successfully.

1 = Memory management fault, see <31-8> for specifics (on Console Request).

2 = Error occurred on Console Request.

3 = Warm/Cold Start Power-up sequence completed (occurs on console "INIT" also).

4 = Interrupt Stack not valid.

5 = CPU Double Error Halt (see Machine Mheck Abort/Fault/Halt Spec for details)!!! Sys Has Crashed!

6 = Halt instruction.

7 = Illegal I/E Vector Code/<1:0>

8 = No user WCS (I/E vector specs user WCS).

9 = Error interrupt(s) pending on "HALT" command.

A = CHM halt.

B = (open)

NOTE

4,5,7,8 codes represent system crashes and require a Sys Rebott to continue!!!

6.9 RETRYABLE INSTRUCTION LIST

See the attached list for instructions that are retryable and the specific information and conditions. A summary and interpretation is as follows: [applies only to CP Errors, all instructions faultable on IB Errors]

INSTRUCTIONS THAT ARE NOT RETRYABLE ON CP ERRORS:

ADDN4	CVTNL?	REI
ADDN6	CVTNP	REMQUE
ASHN	CVTPN	SUBN4
CHME?	EDITPC	SUBN6
CHMK?	INSQUE	SVPCTX
CHMS?	LDPCTX	XFC
CHMU?	MOVN	
CMPN3	MULN	
CMPN4	POPR	
CVTLN	PUSHR	

There are also restrictions on retrying some instructions that are retryable. Restrictions are noted using notes on the attached list.

[x] Means instruction is conditionally retryable.

Some restrictions are:

1. Cannot retry unaligned writes.

The Error handling micro-code aborts all such cases.

2. Instructions Referencing I/O Space

Software required to decide if retryable or not.

In addition these restrictions, the instructions with the following notes also have the added restrictions as noted.

[2] Cannot retry any write.

On the instructions noted, the operating software must examine the logged out information on the stack to determine if they were doing a write. If so they are not retryable.

[3] Can cause a SBI Fault.

On the instructions so noted, SBI fault will occur if the interlocked write is aborted because of a TB parity error. (Interlock timeout in unit receiving the previous interlock read). Operating software must determine that this case exists.

[4] Machine check while pushing information on stack (Kernel).

The Error Handling micro-code aborts all such cases.

[?] Means retryability to be determined later.

[NR] Instruction is not retryable.

9D	ACBB	ADD COMPARE AND BRANCH BYTE	x
6F	ACBD	ADD COMPARE AND BRANCH DOUBLE	x
4F	ACBF	ADD COMPARE AND BRANCH FLOATING	x
F1	ACBL	ADD COMPARE AND BRANCH LONG	x
3D	ACBW	ADD COMPARE AND BRANCH WORD	x
80	ADDB2	ADD BYTE 2 OPERAND	x
81	ADDB3	ADD BYTE 3 OPERAND	x
60	ADDD2	ADD DOUBLE 2 OPERAND	x
61	ADDD3	ADD DOUBLE 3 OPERAND	x
40	ADDF2	ADD FLOATING 2 OPERAND	x
41	ADDF3	ADD FLOATING 3 OPERAND	x
C0	ADDL2	ADD LONG 2 OPERAND	x

C1	ADDL3	ADD LONG 3 OPERAND | x |
20	ADDN4	ADD NUMERIC 4 OPERAND | NR|
21	ADDN6	ADD NUMERIC 6 OPERAND | NR|
A0	ADDW2	ADD WORD 2 OPERAND | x |
A1	ADDW3	ADD WORD 3 OPERAND | x |
D8	ADWC	ADD WITH CARRY |[2]|
F3	AOBLEQ	ADD ONE AND BRANCH ON LESS OR EQUAL | x |
F2	AOBLSS	ADD ONE AND BRANCH ON LES | x |
78	ASHL	ARITHMETIC SHIFT LONG | x |
F8	ASHN	ARTHMETIC SHIFT NUMERIC | NR|
79	ASHQ	ARITHMETIC SHIFT QUAD |[2]|
E1	BBC	BRANCH ON BIT CLEAR | x |
E5	BBCC	BRANCH ON BIT CLEAR AND CLEAR | x |
E7	BBCCI	BRANCH ON BIT CLEAR AND CLEAR INTERLOCKED |[3]|
E3	BBCS	BRANCH ON BIT CLEAR AND SET | x |
1E	BCC	BRANCH ON CARRY CLEAR | x |
1F	BCS	BRANCH ON CARRY SET | x |
E0	BBS	BRANCH ON BIT SET | x |
E4	BBSC	BRANCH ON BIT SET AND CLEAR | x |
E2	BBSS	BRANCH ON BIT SET AND SET | x |
E6	BBSSI	BRANCH ON BIT SET AND SET INTERLOCKED |[3]|
13	BEQL	BRANCH ON EQUAL | x |
13	BEQLU	BRANCH ON EQUAL UNSIGNED | x |
18	BGEQ	BRANCH ON GREATER OR EQUAL | x |
1E	BGEQU	BRANCH ON GREATER OR EQUAL UNSIGNED | x |
14	BGTR	BRANCH ON GREATER | x |
1A	BGTRU	BRANCH ON GREATER UNSIGNED | x |
8A	BICB2	BIT CLEAR BYTE 2 OPERAND | x |
8B	BICB3	BIT CLEAR BYTE 3 OPERAND | x |
CA	BICL2	BIT CLEAR LONG 2 OPERAND | x |
CR	BICL3	BIT CLEAR LONG 3 OPERAND | x |
B9	BICPSW	BIT CLEAR PROGRAM STATUS WORD | x |
AA	BICW2	BIT CLEAR WORD 2 OPERAND | x |
AB	BICW3	BIT CLEAR WORD 3 OPERAND | x |
88	BISB2	BIT SET BYTE 2 OPERAND | x |
89	BISB3	BIT SET BYTE 3 OPERAND | x |
C8	BISL2	BIT SET LONG 2 OPERAND | x |
C9	BISL3	BIT SET LONG 3 OPERAND | x |
B8	BISPSW	BIT SET PROGRAM STATUS WORD | x 1
A8	BISW2	BIT SET WORD 2 OPERAND | x |
A9	BISW3	BIT SET WORD 3 OPERAND | x |
93	BITB	BIT TEST BYTE | x |
D3	BITL	BIT TEST LONG | x |
B3	BITW	BIT TEST WORD | x |
E9	BLBC	BRANCH ON LOW BIT CLEAR | x |
E8	BLBS	BRANCH ON LOW BIT SET | x |
15	BLEQ	BRANCH ON LESS OR EQUAL | x |
1B	BLEQU	BRANCH ON LESS OR EQUAL UNSIGNED | x |
19	BLSS	BRANCH ON LESS | x |
1F	BLSSU	BRANCH ON LESS UNSIGNED | x |
12	BNEQ	BRANCH ON NOT EQUAL | x |
12	BNEQU	BRANCH ON NOT EQUAL UNSIGNED | x |
03	BPT	BREAK POINT TRAP |[4]|
11	BRB	BRANCH WITH BYTE DISPLACEMENT | x |
31	BRW	BRANCH WITH WORD DISPLACEMENT | x |

10	BSBB	BRANCH TO SUBROUTINE WITH BYTE DISPLACEMENT | x |
30	BSBW	BRANCH TO SUBROUTINE WITH WORD DISPLACEMENT | x |
1C	BVC	BRANCH ON OVERFLOW CLEAR | x |
1D	BVS	BRANCH OF OVERFLOW SET | x |
FA	CALLG	CALL WITH GENERAL ARGUMENT LIST |[2]|
FB	CALLS	CALL WITH STACK |[2]|
8F	CASEB	CASE BYTE | x |
CF	CASEL	CASE LONG | x |
AF	CASEW	CASE WORD | x |
BD	CHME	CHANGE MODE TO EXECUTIVE | ? |
BC	CHMK	CHANGE MODE TO KERNAL | ? |
BE	CHMS	CHANGE MODE TO SUPERVISOR | ? |
BF	CHMU	CHANGE MODE TO USER | ? |
94	CLRB	CLEAR BYTE | x |
7C	CLRD	CLEAR DOUBLE |[2]|
D4	CLRF	CLEAR FLOAT | x |
D4	CLRL	CLEAR LONG | x |
7C	CLRQ	CLEAR QUAD |[2]|
B4	CLRW	CLEAR WORD | x |
91	CMPB	COMPARE BYTE | x |
29	CMPC3	COMPARE CHARACTER 3 OPERAND | x |
2D	CMPC5	COMPARE CHARACTER 5 OPERAND | x |
71	CMPD	COMPARE DOUBLE | x |
51	CMPF	COMPARE FLOATING | x |
D1	CMPL	COMPARE LONG | x |
35	CMPN3	COMPARE NUMBERIC 3 OPERAND | ? |
37	CMPN4	COMPARE NUMBERIC 4 OPERAND | ? |
EC	CMPV	COMPARE VIELD | x |
B1	CMPW	COMPARE WORD | x |
ED	CMPZV	COMPARE ZERO-EXTENDED VIELD | x |
0B	CRC	CALCULATE CYCLIC REDUNDANCY CHECK | x |
6C	CVTBD	CONVERT BYTE TO DOUBLE |[2]|
4C	CVTBF	CONVERT BYTE TO FLOAT | x |
98	CVTBL	CONVERT BYTE TO LONG | x |
99	CVTBW	CONVERT BYTE TO WORD | x |
68	CVTDB	CONVERT DOUBLE TO BYTE | x |
76	CVTDF	CONVERT DOUBLE TO FLOAT | x |
6A	CVTDL	CONVERT DOUBLE TO LONG | x |
69	CVTDW	CONVERT DOUBLE TO WORD | x |
48	CVTFB	CONVERT FLOAT TO BYTE | x |
56	CVTFD	CONVERT FLOAT TO DOUBLE |[2]|
4A	CVTFL	CONVERT FLOAT TO LONG | x |
49	CVTFW	CONVERT FLOAT TO WORD | x |
F6	CVTLB	CONVERT LONG TO BYTE | x |
6E	CVTLD	CONVERT LONG TO DOUBLE |[2]|
4E	CVTLF	CONVERT LONG TO FLOAT | x |
F9	CVTLN	CONVERT LONG TO NUMERIC | NR|
F7	CVTLW	CONVERT LONG TO WORD | x |
36	CVTNL	CONVERT NUMERIC TO LONG | ? |
26	CVTNP	CONVERT NUMERIC TO PACKED | NR|
24	CVTPN	CONVERT PACKED TO NUMERIC | NR|
6B	CVTRDL	CONVERT ROUNDED DOUBLE TO LONG | x |
4B	CVTRFL	CONVERT ROUNDED FLOAT TO LONG | x |
33	CVTWB	CONVERT WORD TO BYTE | x |
6D	CVTWD	CONVERT WORD TO DOUBLE |[2]|

4D	CVTWF	CONVERT WORD TO FLOAT | x |
32	CVTWL	CONVERT WORD TO LONG | x |
97	DECB	DECREMENT BYTE | x |
D7	DECL	DECREMENT LONG | x |
B7	DECW	DECREMENT WORD | x |
86	DIVB2	DIVIDE BYTE 2 OPERAND | x |
87	DIVB3	DIVIDE BYTE 3 OPERAND | x |
66	DIVD2	DIVIDE DOUBLE 2 OPERAND |[2]|
67	DIVD3	DIVIDE DOUBLE 3 OPERAND |[2]|
46	DIVF2	DIVIDE FLOATING 2 OPERAND | x |
47	DIVF3	DIVIDE FLOATING 3 OPERAND | x |
C6	DIVL2	DIVIDE LONG 2 OPERAND | x |
C7	DIVL3	DIVIDE LONG 3 OPERAND | x |
27	DIVN	DIVIDE NUMERIC | NR|
A6	DIVW2	DIVIDE WORD 2 OPERAND | x |
A7	DIVW3	DIVIDE WORD 3 OPERAND | x |
38	EDITPC	EDIT PACKED TO CHARACTER | NR|
7B	EDIV	EXTENDED DIVIDE |[2]|
74	EMODD	EXTENDED MODULUS DOUBLE |[2]|
54	EMODF	EXTENDED MODULUS FLOATING |[2]|
7A	EMUL	EXTENDED MULTIPLY |[2]|
EE	EXTV	EXTRACT VIELD | x |
EF	EXTZV	EXTRACT ZERO-EXTENDED VIELD | x |
EB	FFC	FIND FIRST CLEAR BIT | x |
EA	FFS	FIND FIRST SET BIT | x |
00	HALT	HALT |[4]|
96	INCB	INCREMENT BYTE | x |
D6	INCL	INCREMENT LONG | x |
B6	INCW	INCREMENT WORD | x |
F0	INSV	INSERT VIELD | x |
0E	INSQUE	INSERT INTO QUEUE | NR|
17	JMP	JUMP | x |
16	JSB	JUMP TO SUBROUTINE | x |
06	LDPCTX	LOAD PROGRAM CONTEXT | NR|
3A	LOCC	LOCATE CHARACTER | x |
39	MATCHC	MATCH CHARACTERS | x |
92	MCOMB	MOVE COMPLEMENTED BYTE | x |
D2	MCOML	MOVE COMPLEMENTED LONG | x |
B2	MCOMW	MOVE COMPLEMENTED WORD | x |
DB	MFPR	MOVE FROM PRECESSOR REGISTER |[2]|
8E	MNEGB	MOVE NEGATED BYTE | x |
72	MNEGD	MOVE NEGATED DOUBLE |[2]|
52	MNEGF	MOVE NEGATED FLOATING | x |
CE	MNEGL	MOVE NEGATED LONG | x |
AE	MNEGW	MOVE NEGATED WORD | x |
9E	MOVAB	MOVE ADDRESS OF BYTE | x |
7E	MOVAD	MOVE ADDRESS OF DOUBLE | x |
DE	MOVAF	MOVE ADDRESS OF FLOAT | x |
DE	MOVAL	MOVE ADDRESS OF LONG | x |
7E	MOVAQ	MOVE ADDRESS OF QUAD | x |
3E	MOVAW	MOVE ADDRESS OF WORD | x |
90	MOVB	MOVE BYTE | x |
28	MOVC3	MOVE CHARACTER 3 OPERAND | x |
2C	MOVC5	MOVE CHARACTER 5 OPERAND | x |
70	MOVD	MOVE DOUBLE |[2]|

50	MOVF	MOVE FLOAT | x |
D0	MOVL	MOVE LONG | x |
34	MOVN	MOVE NUMERIC | ? |
DC	MOVPSL	MOVE PROGRAM STATUS LONGWORD | x |
7D	MOVQ	MOVE QUAD |[2]|
2E	MOVTC	MOVE TRANSLATED CHARACTERS | x |
2F	MOVTUC	MOVE TRANSLATED UNTIL CHARACTER | x |
B0	MOVW	MOVE WORD | x |
9A	MOVZBL	MOVE ZERO-EXTENDED BYTE TO LONG | x |
9B	MOVZBW	MOVE ZERO-EXTENDED BYTE TO WORD | x |
3C	MOVZWL	MOVE ZERO-EXTENDED WORD TO LONG | x |
DA	MTPR	MOVE TO PROCESSOR REGISTER | x |
84	MULB2	MULTIPLY BYTE 2 OPERAND | x |
85	MULB3	MULTIPLY BYTE 3 OPERAND | x |
64	MULD2	MULTIPLY DOUBLE 2 OPERAND |[2]|
65	MULD3	MULTIPLY DOUBLE 3 OPERAND |[2]|
44	MULF2	MULTIPLY FLOATING 2 OPERAND | x |
45	MULF3	MULTIPLY FLOATING 3 OPERAND | x |
C4	MULL2	MULTIPLY LONG 2 OPERAND | x |
C5	MULL3	MULTIPLY LONG 3 OPERAND | x |
25	MULN	MULTIPLY NUMERIC | NR|
A4	MULW2	MULTIPLY WORD 2 OPERAND | x |
A5	MULW3	MULTIPLY WORD 3 OPERAND | x |
01	NOP	NO OPERATION | x |
75	POLYD	EVALUATE POLYNOMIAL DOUBLE |[2]|
55	POLYF	EVALUATE POLYNOMIAL FLOATING | x |
BA	POPR	POP REGISTERS | NR|
0C	PROBER	PROBE READ ACCESS | x |
0D	PROBEW	PROBE WRITE ACCESS | x |
9F	PUSHAB	PUSH ADDRESS OF BYTE | x |
7F	PUSHAD	PUSH ADDRESS OF DOUBLE | x |
DF	PUSHAF	PUSH ADDRESS OF FLOAT | x |
DF	PUSHAL	PUSH ADDRESS OF LONG | x |
7F	PUSHAQ	PUSH ADDRESS OF QUAD | x |
3F	PUSHAW	PUSH ADDRESS OF WORD | x |
DD	PUSHL	PUSH LONG | x |
BB	PUSHR	PUSH REGISTERS | NR|
02	REI	RETURN FROM EXCEPTION OR INTERRUPT | NR|
0F	REMQUE	REMOVE FROM QUEUE | NR|
04	RET	RETURN FROM CALLED PROCEDURE | x |
9C	ROTL	ROTATE LONG | x |
05	RSB	RETURN FROM SUBROUTINE | x |
D9	SBWC	SUBTRACT WITH CARRY |[2]|
2A	SCANC	SCAN FOR CHARACTER | x |
3B	SKPC	SKIP CHARACTER | x |
F4	SOBGEQ	SUBTRACT ONE AND BRANCH ON GREATER OR EQUAL | x |
F5	SOBGTR	SUBTRACT ONE AND BRANCH ON GREATER | x |
2B	SPANC	SPAN CHARACTERS | x |
82	SUBB2	SUBTRACT BYTE 2 OPERAND | x |
83	SUBB3	SUBTRACT BYTE 3 OPERAND | x |
62	SUBD2	SUBTRACT DOUBLE 2 OPERAND |[2]|
63	SUBD3	SUBTRACT DOUBLE 3 OPERAND |[2]|
42	SUBF2	SUBTRACT FLOATING 2 OPERAND | x |
43	SUBF3	SUBTRACT FLOATING 3 OPERAND | x |
C2	SUBL2	SUBTRACT LONG 2 OPERAND | x |

C3	SUBL3	SUBTRACT LONG 3 OPERAND	x
22	SUBN4	SUBTRACT NUMERIC 4 OPERAND	NR
23	SUBN6	SUBTRACT NUMERIC 6 OPERAND	NR
A2	SUBW2	SUBTRACT WORD 2 OPERAND	x
A3	SUBW3	SUBTRACT WORD 3 OPERAND	x
07	SVPCTX	SAVE PROCESS CONTEXT	NR
95	TSTB	TEST BYTE	x
73	TSTD	TEST DOUBLE	x
53	TSTF	TEST FLOAT	x
D5	TSTL	TEST LONG	x
B5	TSTW	TEST WORD	x
FC	XFC	EXTENDED FUNCTION CALL	NR
8C	XORB2	EXCLUSIVE-OR BYTE 2 OPERAND	x
8D	XORB3	EXCLUSIVE-OR BYTE 3 OPERAND	x
CC	XORL2	EXCLUSIVE-OR LONG 2 OPERAND	x
CD	XOPL3	EXCLUSIVE-OR LONG 3 OPERAND	x
AC	XORW2	EXCLUSIVE-OR WORD 2 OPERAND	x
AD	XORW3	EXCLUSIVE-OR WORD 3 OPERAND	x
08		* RESERVED TO DEC *	
09		* RESERVED TO DEC *	
0A		* RESERVED TO DEC *	
57		* RESERVED TO DEC *	
58		* RESERVED TO DEC *	
59		* RESERVED TO DEC *	
5A		* RESERVED TO DEC *	
5B		* RESERVED TO DEC *	
5C		* RESERVED TO DEC *	
5D		* RESERVED TO DEC *	
5E		* RESERVED TO DEC *	
5F		* RESERVED TO CEC *	
77		* RESERVED TO DEC *	
FD	ESCD	* RESERVED TO DEC *	
FE	ESCE	* RESERVED TO DEC *	
FF	ESCF	* RESERVED TO DEC *	

CHAPTER 7

CACHE-SBI-TB SUBSYSTEM

7.1 MD BUS

MD Bus transfers longword aligned data amongst the cache, SBI interface, data path, and instruction buffer. Signals are:

```
BUS  MD    XX  H                 where XX runs 00 to 31

BUS  MD  BYTE  X  PARITY  H      where X runs 0 to 3

BUS  MD  BYTE  X  MASK    H      where X runs 0 to 3

TBMD     D  TO  MD  L

TBMD     MASK  TO  MD  L
```

Parity is computed over each 8 data bits, such that if the 8 data bits are low, the parity bit will be high. The mask is not checked. Byte mask high means write on a write cycle, and means this byte is wanted on a read cycle. The data, parity, and mask are all long word aligned.

The D TO MD signal directly drives the enables of the data and parity drivers on the data path. The MASK TO MD signal directly drives the enable of the mask driver on the data path.

7.2 CS BUS

This subsystem uses 6 CS bits:

```
BUS   CS   42   H  |  UFS
BUS   CS   47   H  |  UADS
BUS   CS   46   H  |  UMCT3
BUS   CS   45   H  |  UMCT2
BUS   CS   44   H  |  UMCT1
BUS   CS   43   H  |  UMCT0
```

These bits will be received by 74S194 chips on the TBM board.

7.3 V BUS

This subsystem will meet the V bus spec. Available signals TBS.

7.4 CLOCK BUS

Clock signal loading is:

7.5 ADDRESS BUS

The address is received in three sections;

VA REG < 8:2> H

VA MUX <15:9> L

VA MUX <31:16> L

The first group of seven bits is the low bits of the CPU virtual address register, unbuffered. During a subsystem activity using a virtual or physical address in the VA register, these bits will be put on the PA bus, the subsystem internal physical address bus. During a microcode requested load of the Instruction Physical Address Register (IPA) the low seven bits will be copied from the VA bits.

The next set of seven bits is the output of multiplexers. One set of data inputs is attached to the VA register, the other set is attached to the IA register. These multiplexer bits are constantly enabled.

The upper sixteen bits are driven the same way as the middle seven, except that the enables of the multiplexers are driven by a CPU generated signal to provide zeros for compatibility mode. All microcode-specified memory operations use the address from the VA. The IA is used only for automatic reloading of the IPA.

7.6 FROM IB

A. IDPJ COUNT H means that the IPA should be incremented. Only present one cycle after the IB receives data. Not present during FLUSH.

B. IDPJ FLUSH L means that the old IPA is no longer valid. Microcode should not do FLUSH and READ.V.NEWPC in the same state.

C. IRCH IB WRITE CHK H means that if the microcode does a READ.V.IBCHK the check should be for write access. This signal is clocked at the same time as the CS bits.

D. IDPJ IB REQ H means that if the microcode allows an IBREAD, the IB would like to use it, or if miss data comes back, the IB would like to receive it.

7.7 TO IB

A. SBLR IB READ DATA L means that data for the IB is on te MD bus this microcycle.

B. TBMX IB MISS L means that on the most recent load of the IPA, no entry was found in the translation buffer.

C. TBMX IB ERR L means that either

1. on the most recent load of the IPA, an entry was found in the TB but protection code was bad or
2. on the most recent load of the IPA, a parity error occured or
3. the SBI interface detected an error during a cycle being done for the IB which resulted in data never being delivered.

7.8 FROM MICROSEQUENCER

A. USCB ABORT CYCLE H means that the word coming from the control store should not be used because of a microtrap or ECO or console crock.

7.9 TO MICROSEQUENCER

A. SBLT STALL L means that the next microword should be temporarily prevented from executing.

B. TBMD LAST REF CODE 1 H
TBMD LAST REF CODE 0 H

are microbranch conditions. They are the output of a register clocked on any state which saves context (not inhibited by UMISC field). The codes are:

CODE 1	0	means
0	0	READ with RCHK
0	1	READ with WCHK
1	1	WRITE with WCHK
1	0	INTLK READ

C. TBMX BRANCH CODE 1 H
TBMX BRANCH CODE 0 H

are microbranch conditions. They are the output of a register.

CODE 1	0	means
0	0	WONDERFUL
0	1	TBHIT and PROTECTION OK and MBIT ERROR
1	1	TBMISS
1	0	TBHIT and PROTECTION VIOLATION

D. TBMB KERNEL MODE H is a microbranch condition.

7.10 FROM TRAPS AND INTERRUPTS

A. (TEMP) ISR CODE 1 H and 0 H specify the level at which an interrupt summary read SBI transaction should be executed if the microcode orders one. Stable from before the microcommand until after it.

B. (TEMP) CURRENT MODE 1 H and 0 H are the current mode bits of the PSL used for checking protection for certain virtual references and for auto-refill of the IPA. Stable all during any of them.

C. (TEMP) CSPAR ERR H

D. (TEMP) CMODDADRS TRAP L

E. (TEMP) PAGE TRAP H indicates data crossing page boundary.

7.11 TO TRAPS AND INTERRUPTS

A. Several signals which cause interrupts.

1. SBLM CRD RDS INTR L requests an interrupt if the CPU receives a CRD or an RDS from the SBI.

2. SBLM TIMO CNF INT L requests an interrupt if the CPU times out or receives an ERR confirmation on the SBI.

3. SBHE SBI REQ 7 R H
   SBHE SBI REQ 6 R H
   SBHE SBI REQ 5 R H
   SBHE SBI REQ 4 R H

   These are the SBI interrupt requests.

4. SBHE SBI ALERT R H is the SBI ALERT signal.

5. SBHK COMP INTR H requests an interrupt when the SBI silo comparator matches.

6. SBHL FAULT INTR H requests an interrupt because of the assertion of FAULT on the SBI.

B. Several signals which cause microtraps.

1. SBLM TIMEOUT TRAP L requests a microtrap on CPU timeouts which prevent further progress.

2. SBLP PAR ERR TRAP L requests a microtrap on CPU read cycles which encounter a cache parity error.

3. SBLR RDS TRAP L requests microtrap if a CPU READ cycle receives an RDS from the SBI.

4. TBMU PROT UTRAP L requests a microtrap if a translation for the CPU causes protection violation.

5. TBMW TB PAR UTRAP L requests a microtrap on TB parity errors during translation for the CPU.

6. TBMW MBIT UTRAP L requests a microtrap if a translation for the CPU doing a writecheck uses a TB entry with the MBIT not set.

7. TBMW MISS UTRAP L requests a microtrap if a translation for the CPU does not find an entry in the TB.

C. Several microtrap enable signals.

1. TBMN PAGE EDGE H indicates that the data crossing page boundary microtrap is enabled and VAREG <8:3> are high.

2. TBMW EN CMODDADRS H indicates that this type of memory cycle should enable the compatibility mode odd address trap.

3. TBMW EN UNALIGN TRAP H indicates that this type of cycle should microtrap if the data is not aligned.

D. TBMW SAVE CONTEXT H means that this type of memory cycle requires that certain context information be saved. This signal is overridden in the data path by certain UMISC field codes.

7.12 FROM DATA PATH - NONE

7.13 TO DATA PATH

A. SBLP MD to D L means that the D register should be loaded from the MD bus.

B. TBMD D TO MD L turns on the MD bus drivers in the data path.

C. TBMD MASK TO MD L turns on the MD bus mask drivers in the data path.

D. TBMC ENABLE IA H is the input which switch the VAMUX.

7.14 SELECTED INTERNAL SUBSYSTEM SIGNALS

A. SBHM SET SBI CYCLE H means that the MD bus cycle coming up will be used by the SBI interface, usually to transfer data from a read miss or to invalidate a location in the cache that was written on the SBI.

B. SBLK BUFFER FULL H means that the register in the SBI interface used to hold addresses for SBI cycles is full, because the previous write has not been acknowledged, the expected read data has not arrived, etc.

C. SBLR VALID H is the input to the valid bit in the data cache tag.

D. SBLR SET FORCE SBI L is the output of a circuit which will grab the first opportunity after a CPU write miss parity error to clear out the entry with the error.

E. TBMU CANCEL L indicates to the SBI interface that the cycle requested by the microcode should not be completed this cycle because of some error condition known on TBM or because an auto-refill of te IPA is in progress.

7.15 MICROBRANCHES

(See "VAX 11/780 Microcode" for up to date information).

A. BEN15 - LAST REFERENCE

This subsystem provides bits one and zero for this BEN. The code is:

UPC		RETRY
1	0	MICROORDER
0	0	8
0	1	14
1	1	6
1	0	10

B. BEN 1D - TRANSLATION TEST

This subsystem provides bits one and zero for this BEN. The code is given under microorders 0,2 below.

7.16 MICROORDERS

The available microorders are shown in the chart. Descriptions follow, indexed by the order number (decimal).

0,2 These are used to get the translation buffer's attention to load set of microbranch codes which can be tested in the next microinstruction.

	MSB	LSB	
The code is	1	1	TBMISS
	1	0	PROTECTION VIOLATION.NOT TBMISS
	0	1	NOT PROT VIOLATION.NOT TBMISS.WCHK.MBIT
	0	0	NO PROBLEM

6 This is the normal virtual write. This is retryable. Retry involves

1. using the previous cycle type microbranch to find out which of the four retryable cycles was being done.
2. sending the proper microorder again in combination with the proper saved context code in the miscellaneous field.

5 This code does a virtual write without a protection check or modify bit check. It is used for cycles that are prechecked by microcode, such as writing page table entries.

7	This is the virtual interlock write.
8	Normal virtual read. Retryable.
9	Nocheck virtual read for page tables, etc.
10	Virtual read with write check for modifying accesses. Retryable.
11	Virtual read with protection check read or write specified by the instruction buffer. The retry branch will indicate whether to retry as code 8 or 10.
12	This cycle is used whenever the microcode wishes to reload the IPA, whether because of a macroprogram transfer of control or to restart instruction prefetching after loading the TB with a translation of a previously missing page. It causes a virtual read cycle with data to the instruction buffer. All errors are handled with the IB error mechanism.
14	Virtual interlock read, used for interlock instruction. Retryable.
13	Nocheck virtual interlock read. No known use at present, but may be needed if MBIT update is respecified as interlocked.
25	Physical Read, for LDPCTX etc.
21	Physical Write, for STPCTX etc.
29,23	Physical Interlock Read and Write. No known use at present.
27	Causes a Read Interrupt Summary transaction on the SBI.
20	Causes an Extended Write on the SBI. The data written is unpredictable. Used to clear out double ECC errors in MOS. The location in cache will be invalidated.
16,17	Asserts Hold and Unjam on the SBI. For use in the console UNJAM command sequence.
18	Writes good parity non-valid data and tag in both groups at the index position specified. No SBI cycle. Used on all index positions at power up. Also used by microdiagnostics. May also be used by certain error routines.
19	Causes a write of the specified address and data, with good parity, marked valid, in the group specified by the force replacement bits. Used by microdiagnostics. No SBI cycle.

31,48-63 Allows initiation of read cycles by the instruction buffer while performing the ID bus operation specified. This code should be used in most places where an explicit memory operation is not required in order to allow the IB to acquire I stream bytes. Should not be used during IB FLUSH states.

32-47 ID bus operation only, IB cycle initiation is blocked. This code should appear in all locations of error trap routines until the IB can be turned off to prevent multiple errors.

1 NO OPERATION

3,4,15,22,24,26,27,28,30 RESERVED, UNPREDICTABLE, DO NOT USE EFFECT OF TRAPS

	READ	WRITE
TBMISS	A	A
PROTECTION	A	A
DATA CROSS PAGE	A	A
DATA NOT ALIGNED	N	N
WCHK.TBMBIT	A	A
CM.ODD ADRS	A	A
TB PARITY	A	A
CACHE PARITY	A	No Trap
TIMEOUT	B	B
CRD	*Not a Trap	-
URD (RDS)	A*	-
CS PARITY	A	A

A: No good read data
memory not written
cache not written
cache sideeffected
memory not sideeffected

B: No good read data
memory may be written
cache may be written
cache sideeffected
memory may be sideeffected

N: all normal effects

*: memory sideeffected

sideeffected includes changes in status bits

Memory Control Functions October 11, 1976

A D S	MCT	F/ P	F U N	C H E	S A V	trap on: T B S	A C S	X P E	A L N	T B M	O D A	T B R	S C R	B I E	
0	0000	0	V		R	N	N	N	N	N	N	Y	N	N	TEST.RCHK
0	0001	0	V			N	N	N	N	N	N	N	N	N	MEM.NOP
0	0010	0	V		W	N	N	N	N	N	N	Y	N	N	TEST.WCHK
0	0011	0													
0	0100	0													
0	0101	0	V	W		N	Y	N	-	-	N	-	Y	N	WRITE.V.NOCHK
0	0110	0	V	W	W	Y	Y	Y	*	*	Y	Y	Y	N	WRITE.V.WCHK
0	0111	0	V	IW		N	-	-	-	-	-	-	Y	N	LOCKWRITE.V.XCHK
0	1000	0	V	R	R	Y	Y	Y	*	*	N	Y	Y	Y	READ.V.RCHK
0	1001	0	V	R		N	Y	N	-	-	N	-	Y	Y	READ.V.NOCHK
0	1010	0	V	R	W	Y	Y	Y	*	*	Y	Y	Y	Y	READ.V.WCHK
0	1011	0	V	R	IB	Y	Y	Y	Y	Y	Y	Y	Y	Y	READ.V.IBCHK
0	1100	0	V	R	R	N	N	N	N	N	N	Y	N	N	READ.V.NEWPC
0	1101	0	V	IR		N	Y	N	-	-	N	-	Y	Y	LOCKREAD.V.NOCHK
0	1110	0	V	IR	W	Y	Y	Y	-	-	Y	-	Y	Y	LOCKREAD.V.WCHK
0	1111	0													
1	0000	0		HOLD		N	N	N	N	N	N	N	N	N	SBI.HOLD
1	0001	0		UNJAM		N	N	N	N	N	N	N	N	N	SBI.HOLD+UNJAM
1	0010	0	P	INVAL		N	N	N	N	N	N	N	N	N	INVALIDATE
1	0011	0	P	VAL		N	N	N	N	N	N	N	N	N	VALIDATE
1	0100	0	P	EXTWR		N	N	N	N	N	N	N	N	Y	EXTWRITE.P
1	0101	0	P	W		N	N	N	N	N	N	N	N	Y	WRITE.P
1	0110	0													
1	0111	0	P	IW		N	N	N	N	N	N	N	N	Y	LOCKWRITE.P
1	1000	0													
1	1001	0	P	R		N	N	N	N	N	N	N	Y	Y	READ.P
1	1010	0													
1	1011	0	P	ISR		N	N	N	N	N	N	N	N	Y	READ.INT.SUM
1	1100	1													
1	1101	0	P	IR		N	N	N	N	N	N	N	Y	Y	LOCKREAD.P
1	1110	0													
1	1111	0	I	R		N	N	N	N	N	N	N	N	N	ALLOW.IB.READ
0	XXXX	1				N	N	N	N	N	N	N	N	N	NO MEMORY OPERATION
1	XXXX	1	I	R		N	N	N	N	N	N	N	N	N	DEFAULT: ALLOW IB READ

Y = utrap on condition
* = utrap on condition unless MSC/SECOND.REF or RETRY.NO.TRAP
N = do not utrap on condition
\- = hardware behaviour undefined. uncode must prevent condition

Abort Ref on Trap? A A A A A A R A (A=any, R=read)

7.17 REGISTERS

HEX ID ADRS	REGISTER NAME
10	TRANSLATION BUFFER DATA REGISTER
11	NOT USED AT PRESENT
12	TB REGISTER 0
13	TB REGISTER 1
14 to 17	NOT USED BY THIS SUBSYSTEM
18	SILO DATA REGISTER
19	SBI ERROR REGISTER
1A	TIMEOUT ADDRESS REGISTER
1B	SBI FAULT-STATUS REGISTER
1C	SILO COMPARATOR REGISTER
1D	MAINTENANCE REGISTER
1E	CACHE PARITY REGISTER
1F	NOT USED AT PRESENT

10 TRANSLATION BUFFER DATA REGISTER

This register is write only from the ID bus. It is used to write the translation buffer. The virtual address to be translated must be in VA, the PTE must be in D (the MBZ bits must be zero). Which group is written is selected as follows:

1. if the REPLACE BOTH bit is on in TB REGISTER 0, both groups are written. Otherwise,
2. if one of the groups has a good entry for the address being translated, that entry will be updated. Otherwise,
3. if one of the FORCE REPLACE bits in TB REGISTER 0 is on, the selected group will be written. Otherwise,
4. a randomly chosen group will be written.

12 TB REGISTER 0

bit#	name and use
0	MME - read/write bit. If not set, the TB parity, miss, protection, Mbit, and page boundary microtraps are disabled and any address which would normally be translated is used directly from the VA or VIBA as appropriate.

1-4 FORCE TB PARITY ERROR CODE - read/write.

bit code 4 3 2 1	force error is GROUP	ADRS/DATA byte
0 0 0 0		
0 0 0 1		
0 0 1 0	0	D0
0 0 1 1	0	D1
0 1 0 0	0	D2
0 1 0 1	1	D0
0 1 1 0	1	D1
0 1 1 1	1	D2
1 0 0 0	0	A0
1 0 0 1	0	A1
1 0 1 0	0	A2
1 0 1 1	1	A0
1 1 0 0	1	A1
1 1 0 1	1	A2
1 1 1 0		
1 1 1 1		

5 NOT USED

6 TBG0 HIT - read only. This is the latched output of the group zero address checker. For diagnostics.

7 TBG1 HIT - as above for group 1

8-15 LAST REF - read only. These bits contain the following information about the most recent non-NOP memory request by microcode.

15 UFS
14 UADS
13 UMCT3
12 UMCT2
11 UMCT1
10 UMCT0
09 IB WCHK from the instruction buffer
08 This cycle was delayed one cycle by an auto-reload of the IPA

16 FORCE TBG0 MISS - read/write. Causes the group 0 address checker to say no match.

17 FORCE TBG1 MISS - same for group 1. (NOTE: The force miss bits also disable TB parity checking on that group.)

18,19 TB FORCE REPLACE CODE - read/write.

Affects which group is written by the Translation Buffer Data Register as follows:

19	18	result
0	0	RANDOM GROUP
0	1	GROUP 0
1	0	GROUP 1
1	1	DO NOT USE

20 REPLACE BOTH - read/write. See Translation Buffer Data Register for effect. Normally used when clearing the TB.

13 TB REGISTER 1

bit# name and use

0-3 IPA INFO - read only. These bits contain information about the most recent load of IPA as follows:

bit
0 AUTO LOAD - this load was automatic, not the result of READ.V.NEWPC
1 PROTECT - there was a protection violation on this load (or miss).
2 PARITY - there was a parity error on this load.
3 MISS - there was a TB miss on this load.

4 BAD IPA - read only. The information in IPA and IPA INFO is not useful. Set by counting across a page boundary or by FLUSH, cleared by loading the IPA.

5 NOT USED

6 LAST TB WRITE PULSE - read only.
This bit indicates which group was most recently written. It is indeterminate if both groups were written.

7 NOT USED

8 CP TB PAR ERR - cleared by any write to this register. This bit indicates that the TB has requested a TB PAR ERR microtrap.

9-20 TB PAR BITS - cleared by any write to this register. This group of bits will all be loaded if there is a TB parity error and either TB parity traps are enabled or the IPA is being loaded. Each bit gets a one if its corresponding checker detects an error, and a zero otherwise.

bit	group	ADRS/DATA	byte #
9	0	A	0
10	0	A	1
11	0	A	2
12	1	A	0
13	1	A	1
14	1	A	2
15	0	D	0
16	0	D	1
17	0	D	2
18	1	D	0
19	1	D	1
20	1	D	2

18 SILO DATA REGISTER

Listed below are the bit assignments for the SBI Silo.

BIT	NAME
31	First entry after FAULT cleared
30	SBI INTLK
29	SBI ID4
28	SBI ID3
27	SBI ID2
26	SBI ID1
25	SBI ID0
24	SBI TAG2
23	SBI TAG1
22	SBI TAG0
21	SBI M3 or SBI B31
20	SBI M2 or SBI B30
19	SBI M1 or SBI B29
18	SBI M0 or SBI B28
17	SBI CNF1
16	SBI CNF0
15	SBI TR15
14	SBI TR14
13	SBI TR13
12	SBI TR12
11	SBI TR11
10	SBI TR10
9	SBI TR09
8	SBI TR08
7	SBI TR07
6	SBI TR06
5	SBI TR05
4	SBI TR04
3	SBI TR03
2	SBI TR02
1	SBI TR01
0	SBI TR00

NOTE:
Silo bits 21-18 are written with SBI B31-B28 when the SBI TAG FIELD specifies command address TAG. Otherwise, SBI M3-M0 are written in these bit positions.

The SBI Silo is a history register of the state of the indicated SBI signals for the past 16 SBI cycles. While FAULT is not asserted on the SBI, the silo is written and its 4 bit address counter is updated every cycle. When FAULT is asserted, writes into the silo are prevented, and the normal update of the counter is inhibited. The silo address counter will increment only when the silo is read over the ID bus while FAULT is asserted. When FAULT is deasserted, the first location written has Bit 31=1. This register is read only.

19 SBI ERROR REGISTER

BIT #	DESCRIPTION	TYPE
31-16	Not used	R
15	RDS/CRD Interrupt Enable	RW
14	CRD	RWCL
13	RDS	RWCL
12	CP timeout	RWCL
11	CP timeout status 1	R
10	CP timeout status 0	R
9	Not used	R
8	CP SBI Error Confirmation	R
7	IB RDS	RWCL
6	IB Timeout	RWCL
5	IB Timeout status 1	R
4	IB Timeout status 0	R
3	IB SBI Error Confirmation	R
2	Multiple CP Error	R
1	SBI Interface not busy	R
0	Not used	R

Bit 15 & 14

Bit 14, CRD (Corrected Read Data) sets whenever CRD is returned to the CPU. The AND condition of bit 14 and bit 15 CRD/RDS interrupt enable causes an interrupt to be requested.

Bit 13

This bit set whenever RDS (Read Data Substitute) is returned to the CPU. The AND condition of this bit and bit 15 causes an interrupt to be requested.

Bit 12 CP Timeout

This bit will set any time there is a timeout for a CP requested cycle. While this bit is set an interrupt is requested.

Bits 11-10 CP Timeout Status

These bits described the type of timeout that has occurred.

BITS 11	10	
0	0	DEVICE NO RESPONSE
0	1	DEVICE WAS BUSY
1	0	WAITING FOR READ DATA
1	1	IMPOSSIBLE CODE

If notification of timeout was by interrupt (not microtrap) and the code is 10 the cycle was a read, if codes 00 or 01 the cycle was a write. If notification is by microtrap the type of cycle can be found in TB REG0 bits 15 to 8.

Bit 8

This bit sets whenever a CP requested cycle receives an error confirmation to a command address transfer. While this bit is set an interrupt will be requested.

NOTE

Writing a "1" in bit 12 clears bit positions 12-10 & 8 & 2.

Bit 7

Bit 7, IB RDS will set any time for any RDS sent to the CPU while the SBI interface is fetching data for the instructuion buffer on the SBI. This bit is write one to clear. It is also cleared by flushing the instruction buffer.

Bits 6-3

These bits take on similar meaning to bits 12-10, & 8, except these bits set only for instruction buffer initiated requests. Writing a "1" in bit 6 clears bits 6-3. Bits 6-3 clear with the FLUSHING of the instruction buffer.

Notification is by IB error microbranch. The cycle type is always READ.

Bit 2 Multiple CP Error

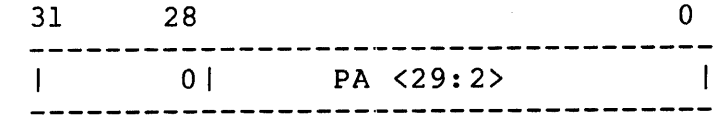
This bit will set when a second CP timeout or CP SBI error confirmation occurs with a previous CP timeout or CP SBI error confirmation set in the register. This bit is reset in the same manner as bits 12-10, & 8, by writing a "1" in bit 12.

Bit 1

This bit is a "1" whenever the SBI interface is not busy doing something on the SBI. Bits 14 & 13 (CRD & RDS) should not be reset by microcode when this bit is "0" (SBI BUSY).

1A TIMEOUT ADDRESS

This register is a holding register for the Physical Address sent on the SBI. Its format is listed below:



Bit 31 = MODE 1
30 = MODE 0
29 = Protection Checked Reference

PA = Physical Address

When a timeout occurs on the SBI, this register will latch up with the physical address of the timeout. This register remains latched until the timeout error bit (located in the SBI error register, bit 12) is written as one. This register will not latch up when a timeout occurs while the SBI is getting data for the Instruction Buffer. This register is read only.

Bits 31-29 provide the mode of the reference that resulted in the timeout. Bit 29 is 0 for references not subject to hardware protection check.

1B SBI FAULT - STATUS REGISTER

Listed below are the bit assignments for the FAULT/STATUS register, also included is a description of the type of bit. R read only, RW read-write, and RWCL read write one to clear.

BIT #	NAME	TYPE
31	Parity FAULT	R
30	Reserved	R
29	Unexpected Read Data FAULT	R
28	Reserved	R
27	Multiple Xmitter FAULT	R
26	Transmitter during FAULT cycle	R
25	Spare 0	RW
24	Spare 1	RW
23	Reserved	R
22	Reserved	R
21	Reserved	R
20	Spare 2	RW
19	FAULT LATCH	RWCL
18	FAULT Interrupt Enable	RW
17	SBI FAULT Signal	R
16	FAULT Silo Lock	R
15-00	Not used	

Bits 31-26 are the FAULT bits defined in the SBI spec. Bits 25, 24 & 20 are spare read/write bits.

Bits 19-17

Bit 19, FAULT latch sets on the leading edge of the SBI FAULT signal. While this bit is set, the CPU will assert FAULT on the SBI. This bit is write one to clear. When this bit is set and bit 18 (FAULT Interrupt Enable)=1 an interrupt will be requested. Bit 17, the SBI FAULT signal provides the ability to detect the FAULT signal being continuously asserted on the SBI.

Bit 16

The SBI Silo may lock due to two conditions (1) the SBI FAULT Signal or (2) the SBI Silo comparator finding a compare. If the SBI FAULT Signal was the reason for locking the Silo then Bit 16, FAULT Silo LOCK, will be set. If the comparator locked the silo, a bit in the comparator register will set. If both mechanisms occur simultaneously, both bits will set. Bit 16 will clear when a 1 is written into bit 19. The SBI FAULT signal must be deasserted for the silo to unlock.

1C SILO COMPARATOR REGISTER

The Silo Comparator allows the SBI Silo to become locked under another mechanism besides the assertion of FAULT on the SBI.

The Silo Comparator may lock the silo under 2 modes of operation. The first mode is unconditional lock. This allows the SBI Silo to become locked anywhere from 0 to 15 cycles after this register is written. The number is determined by the count field contained in this register. The count field is always set in 1's complement, that is to lock unconditionally after one cycle, the number E would be set in the count field. The count field is incremented until it is equal to all 1's.

The second mode of operation is conditional lock. When certain conditions exist on the SBI, a compare signal will be generated. This compare signal is latched. The output of this latch allows the counter to be incremented. When the count fild is equal to all 1's the SBI Silo will lock.

In both cases the Silo will be unlocked by writing a number other than F in the count field. The SBI FAULT signal must be deasserted for the silo to unlock.

The compare modes that are provided are:

1. SBI ID = Maintenance ID

2. SBI ID = Maintenance ID
   and
   SBI TAG = Comparator TAG

3. SBI ID = Maintenance ID
   and
   SBI TAG = Comparator TAG
   and
   SBI B<31:28> or SBI M<3:0> = Comparator Command/Mask Field

* When the comparator tag is equal to command/address the comparator command/mask field will be compared against B<31:28>, the command function, otherwise the compare will be against SBI M<3:0>.

The Maintenance ID bits are located in the Maintenance register.

The AND condition of Comp Silo lock and Silo Lock Interrupt Enable cause an interrupt to be requested. Listed below are the bit definitions for the comparator register.

BIT	NAME	TYPE	
31	Comp Silo Lock	R	*
30	Silo Lock Interrupt Enable	RW	
29	Lock Unconditional	RW	
28	Conditional lock codes	RW	
27	Conditional lock codes	RW	
26	Compare Command or Mask 3	RW	
25	Compare Command or Mask 2	RW	
24	Compare Command or Mask 1	RW	
23	Compare Command or Mask 0	RW	
22	Compare TAG 2	RW	
21	Compare TAG 1	RW	
20	Compare TAG 0	RW	
19	Count Field 3	RW	
18	Count Field 2	RW	
17	Count Field 1	RW	
16	Count Field 0	RW	

Conditional Lock Codes

BIT 28	BIT 27	
0	0	No compare
0	1	ID only
1	0	ID. TAG
1	1	ID. TAG. command function or mask

RW - Read Write
RWCL - Read, Write one to clear

* Any write to this register with bits 19 to 16 not all ones will clear this bit.

1D MAINTENANCE REGISTER

Listed below are the bit assignments for the maintenance register.

BIT	NAME	TYPE
31	Force P0 Reversal on SBI	RW
30	Force Write SEQuence FAULT	RW
29	Force Unexpected Read Data FAULT	RW
28	Force Multiple Xmitter FAULT	RW
27	Maintenance ID4	RW
26	Maintenance ID3	RW
25	Maintenance ID2	RW
24	Maintenance ID1	RW
23	Maintenance ID0	RW
22	Force SBI Invalidate	RW
21	Enable SBI Invalidate	RW
20	Reverse Cache Parity Field	RW
19	Reverse Cache Parity Field	RW
18	Reverse Cache Parity Field	RW
17	Reverse Cache Parity Field	RW
16	Force Miss Group 0	RW
15	Force Miss Group 1	RW
14	Force Replacement Group 0	RW
13	Force Replacement Group 1	RW
12	Disable SBI cycles	RW
11	Force P1 Reversal on SBI	RW
10	Group 1 Match	R
9	Group 0 Match	R
8	Force Timeout	RW
7-0	Not Used	R

R - Read Only
RW - Read Write

REVERSE CACHE PARITY FIELD DEFINITION

20	19	18	17	DESCRIPTION
0	0	0	0	NOP
0	0	0	1	Reverse Parity Group1, Byte A, Address
0	0	1	0	Reverse Parity Group1, Byte B, Address
0	0	1	1	Reverse Parity Group1, Byte C, Address
0	1	0	0	Reverse Parity Group0, Byte A, Address
0	1	0	1	Reverse Parity Group0, Byte B, Address
0	1	1	0	Reverse Parity Group0, Byte C, Address
0	1	1	1	Unused
1	0	0	0	Reverse Parity Group1, Byte 3 Data
1	0	0	1	Reverse Parity Group1, Byte 2 Data
1	0	1	0	Reverse Parity Group1, Byte 1 Data
1	0	1	1	Reverse Parity Group1, Byte 0 Data
1	1	0	0	Reverse Parity Group0, Byte 3 Data
1	1	0	1	Reverse Parity Group0, Byte 2 Data
1	1	1	0	Reverse Parity Group0, Byte 1 Data
1	1	1	1	Reverse Parity Group0, Byte 0 Data

Bits 31 & 11

While these bits are set, the appropriate parity generator in the SBI interface is reversed. The SBI interface must be forced to transmit to force this FAULT on the SBI.

Bit 30

While this bit is set, all writes done by the SBI interface will cause a write sequence FAULT. This is done by changing the WRITE DATA TAG to the TAG reserved for diagnostic use.

Bit 29

While this bit is set, the SBI interface will transmit TAG=0 (read data), ID=Maintenance ID, DATA=undefined, parity ok. This will cause unexpected read data FAULT in the NEXUS selected by the maintenance ID.

Bit 28

While this bit is set, multiple transmitter FAULT can be forced in any NEXUS. For NEXUS other than the CPU, the error is forced by reading the NEXUS configuration register. After the command address transfer specifying read, the CPU transmits the conditions specified under bit 29, except the TAG which is transmitted as 111, the reserved TAG. When the NEXUS returns the read data, with ID=CPU ID, that NEXUS will have a multiple transmitter FAULT (provided that maintenance ID was set to a value that would cause an ID mismatch to occur).

The CPU has this error forced by doing a write command. The CPU ID will be transmitted for the command address transfer. The maintenance ID will be transmitted with the write data. When the received ID is compared against the CPU ID, after the write data transmission, a mismatch will occur resulting in a multiple xmitter FAULT.

Bits 21-23

These are the maintenance ID bits. They are used for forcing unexpected read data FAULTS, forcing multiple transmitter FAULTS and as a compare field for the Silo Comparator.

Bit 22

Setting this bit forces writes done by the CPU on the SBI to become a write invalidate to the CACHE.

Bit 21

When this bit is set write invalidates from the SBI are allowed. When this bit is "0" write invalidates from the SBI are ignored. This bit must be on for normal system operation.

Bit 17-20

These bits are the reverse CACHE Parity Field. Setting this field to a specific code causes the indicated byte to have its parity bit continously asserted. To force the error trap the appropriate CACHE operation should be initiated.

Bits 16-15

These bits provide a method of forcing misses in the CACHE. Misses are not forced for write operations. This prevents the CACHE data from becoming stale. The bits have the following meaning.

BIT		FUNCTION
16	15	
0	0	No misses forced
0	1	Force miss on Group 1
1	0	Force miss on Group 0
1	1	Force miss on Group 1 and Group 0

Misses are forced only for read requests for the Data Path or instruction buffer. Misses are not forced for write or invalidate operations. Setting these bits will also cause parity errors to be ignored.

Bits 14-13

Normally, replacement in the CACHE is random. These bits provide for overriding the random bit.

BIT		CACHE REPLACEMENT
14	13	
0	0	Random
0	1	Group 1 always
1	0	Group 0 always
1	1	Undefined

Bit 12

While this bit is set, no SBI cycles will be started. For read operations with a CACHE miss, the data in the D register will be unpredictable.

Bits 10-9

These two bits are clocked every time there is a read request for the Data Path or the instruction buffer that results in (1) a CACHE Hit or (2) an SBI cycle started due to a CACHE Miss. (approximate)

Bit 8

This bit provides a mechanism for forcing timeouts on a read operation. This is done by loading the timeout counter with 'FF' after the read command has been accepted.

1E CACHE PARITY ERROR REG

The bit definitions for this register are listed below:

BIT #	DESCRIPTION	TYPE
31-16	Not used	R
15	Any Parity Error	RWCL
14	CP Parity Error	R
13	Parity OK CDM Group1 Byte 0	R
12	Parity OK CDM Group1 Byte 1	R
11	Parity OK CDM Group1 Byte 2	R
10	Parity OK CDM Group1 Byte 3	R
9	Parity OK CDM Group0 Byte 0	R
8	Parity OK CDM Group0 Byte 1	R
7	Parity OK CDM Group0 Byte 2	R
6	Parity OK CDM Group0 Byte 3	R
5	Parity OK CAM Group0 Byte 0	R
4	Parity OK CAM Group0 Byte 1	R
3	Parity OK CAM Group0 Byte 2	R
2	Parity OK CAM Group1 Byte 0	R
1	Parity OK CAM Group1 Byte 1	R
0	Parity OK CAM Group1 Byte 2	R

Bit 15

This bit will set any time a CACHE parity error is detected on a IB or CP read operation. Writing a "1" to this bit position forces 0's in all positions of the register (same state as on power up).

Bit 14

When bit 15 is set, this bit signifies whether the CACHE parity error was for the CP, Bit 14=1, or the instruction buffer, bit 14=0.

Bits 13-0

When bit 15 is set, these bits identify the CACHE bytes which did not have an error.

NOTE

This register will clear if it is holding a parity error for the instruction buffer and the instruction buffer is flushed.

ID REGISTERS "ON" TBM BOARD

ADRS HEX

10 TB DATA WRITE ONLY

31	30	27 26	25	21 20	0
VALID	PROTECTION CODE	M	MUST BE ZERO	PHYSICAL ADDRESS PAGE FRAME NUMBER	

SEE VAX 11/780 ARCHITECTURE HANDBOOK

12 TB REG 0

31	21 20	19	18 17	16 15	14	13	12	11	10	9	8	7	6	5	4	1	0
0	REPLACE BOTH	FORCE TB REPLACE CODE G1	FORCE TB MISS G1	FORCE TB MISS G0	UFS	UADS	UMCT3	UMCT2	UMCT1	UMCT0	IBWCHK	AR	TB G1 HIT	TB G0 HIT	0	FORCE TB PARITY ERROR CODE	MME

SEE CACHE SUBSYSTEM SPEC

	R/W	R/W	R/W	R/W	R/W	R0									R0	R0	R/W	R/W	R/W	R/W	R/W	
"NORMAL" USE -------->	0	0	0	0	0	X	X	X	X	X	X	X	X	X	X	0	0	0	0	0	1	(ON)
INIT ----->	0	0	0	0	0	X	X	X	X	X	X	X	X	X	X	0	0	0	0	0	0	

13 TB REG 1

31	21 20	9	8	7	6	5	4	3	2	1	0
0	GROUP--> / A/D --> / BYTE--> TB PARITY ERROR BITS: 1D2, 1D1, 1D0, 0D2, 0D1, 0D0, 1A2, 1A1, 1A0, 0A2, 0A1, 0A0	CP TB PAR ERR	0	LAST TB WP	0	BAD IPA	IPA INFO: MIS, PAR ITY, PERR OR, PERR OR T, ALUO, LOAD				

SEE CACHE SUBSYSTEM SPEC

READ ONLY EXCEPT ANY WRITE TO THIS REGISTER CLEARS --> THESE BITS

INIT -------> 0 0 0 0 0 0 0 0 0 0 0 0 0 0 X 0 1 1 1 0 0

ID REGISTERS ON SBI BOARD

18 SBI SILO
INIT UNPREDICTABLE

31	30	29	25 24	22 21	18 17	16 15	0
AFTER FAULT	SBI INTLK	SBI ID	SBI TAG	SBI M3-M0 OR B31 - B28	SBI CNF1-0	SBI TR<15:00>	

19 SBI ERROR

31 16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NOT USED, ZERO	RDS INT EN	C R D	R D S	CP TO	TO ST1	TO ST0	ZERO	CP ERR CNF	IB RDS	IB TO	IB TO ST1	IB TO ST0	IB ERR CNF	MULT ERR	NOT BSY	ZERO

INIT = 0<----------0----------> 0 1 0

1A TIMEOUT ADDRESS

31	30	29	28	27	0
MODE 1	MODE 0	PROT CHK	0	PHYSICAL ADDRESS<29:2>	

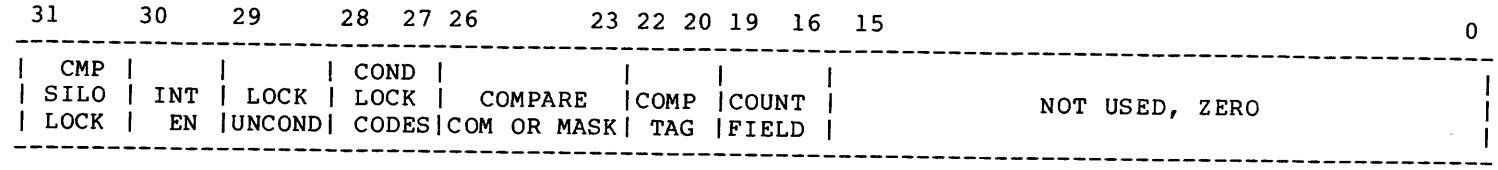
INIT UNPREDICTABLE

1B FAULT/ STATUS

31	30	29	28	27	26	25	19	18	17	16	15	0
PTY FLT	0	UNEX RD	0	MLT XMIT	XMIT FLT	0	0	FLT LTH	FLT INT EN	FLT SIG	FLT SILO LCK	NOT USED, ZERO

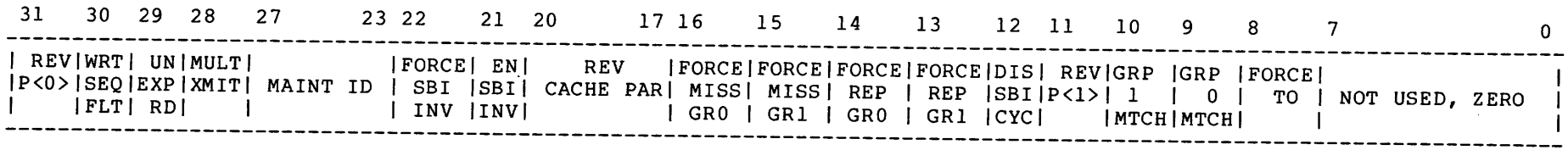
CLEAR ON INIT

1C COMPARATOR



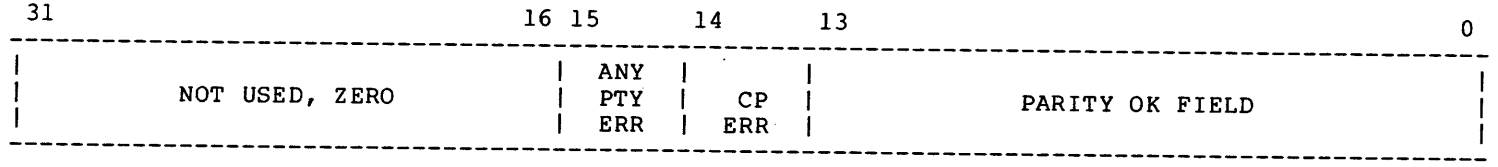
CLEAR ON INIT

1D SBI MAINTENANCE



CLEAR ON INIT

1E CACHE PARITY



CLEAR ON INIT

7.18 GENERAL DESCRIPTION

The Translation Buffer is 64 entries deep, two way associative. Virtual address bits 14 thru 9 are used as the index. This cache translates addresses in the VA register for microcode requested cycles, and translates addresses in the VIBA for automatic reloads of the IPA (Instruction Physical Address) register. This cache is bypassed for physical address references or when MME bit is off.

The IPA keeps a pretranslated copy of the VIBA (Virtual Instruction Buffer Lookahead Address) to prevent the waste of repeated translation. The IPA and VIBA are counted in step. When the IPA counts accross a page boundary, prefetching is prevented until the IPA is automatically reloaded. The reload is controlled by hardware sequencing logic which will start the reload during any ALLOW.IB.READ microcycle. The reload is completed in the following cycle. If this cycle requested a memory operation it is stalled for 200 nanoseconds. Once the reload is completed, prefetching will start again, unless a miss or other error occurred during the reload. In this case the IB will eventually run out of data and notify the microcode of the error. Note that the reload sequence occurs even if MME is not on.

The IPA can be loaded by microcode using the READ.V.NEWPC command. This is normally used when macrocode transfer of control occurs or when prefetching has stopped because of a translation miss and a new entry has been put in the TB.

When a memory cycle is to be performed, the appropriate address (VA, IPA, or translation) is selected by the PAMUX onto the PA BUS. The data cache uses addresses on the PA BUS to search for data.

The Data Cache is two-way associative. Each group contains 1024 long-word entries. One address is stored for each two long-words (one quad-word). If a hit occurs, the appropriate long-word is transmitted on the MD BUS. If a miss occurs, the quad-word containing the requested data is brought in from memory and placed in the cache (the appropriate long-word is also transmitted to the requestor). If the requested data is in I/O space instead of memory, only the appropriate long-word is read in, and it is not placed in the cache.

The replacement strategy is RANDOM. This means that when a new quad-word block is brought into the cache, it is placed in one group or the other as a random selection, not according to the previous entry's age or other characteristic.

The write strategy is write-thru, and not-write-allocate. Write-thru means that any write data transmitted from the CPU to the cache is immediately passed along to main memory. Not-write-allocate means that if the CPU transmits write data and a corresponding entry is not present in the cache the data is simply sent to memory with no new entry being made in the cache.

7.19 MICROCODING SUGGESTIONS

A. To prevent auto-reloading of the IPA:

1. An auto-reload will not start if an ALLOW.IB.READ (either type) is not used.

2. If a page boundary has not been crossed already, it will not be crossed if the IB cannot count the IPA, which will not happen if the IB is stopped.

3. A FLUSH of the IB will stop any auto-reload.

B. Do not meddle with the VIBA anytime that an auto-reload may happen.

C. After a new entry is placed in the TB as a result of an IB miss, a READ.V.NEWPC is required to load the IPA.

D. Do not do FLUSH and READ.V.NEWPC in the same microstate.

E. Writing any of this subsystem's ID registers while a memory operation is in progress is strongly unrecommended.

F. The microcommands in this subsystem's hardware error microtraps should contain NOP until the IB can be turned off, to reduce the likelihood of multiple errors, and until ID REG 12 is saved, to prevent loss of information.

G. Cache error registers should be read out, saved, and cleared before doing any more memory references on errors.

H. Remember that FLUSH clears out certain IB error information.

I. Remember that all microcommands for explicit memory operations use the address in VA, including READ.V.NEWPC.

J. Do not change the VA register in a memory reference microstate. Do not change D register in a write microstate. There may be restrictions on the D register clock control field of read microstates, see the data path spec.

K. When using the INVALIDATE microcommand to clear the cache, use long-word context. Both groups are cleared at once. All 1024 combinations of address bits 11 thru 2 must be cleared.

L. When clearing the TB, set the Replace Both bit and send an all zero word to the TB. For a complete clear, use all 64 combinations of address bits 14 thru 9.

M. The MBZ bits in entries for the TB must be zero.

N. Many of the force error bits in this subsystem are meant for microdiagnostics only and will hang the system if set by macrocode.

O. Remember that finding one error bit set somewhere does not mean there are no other errors. Due to the independent asynchronous operation of the IB, many interesting error combinations are possible.

P. The system will not run if misses are being forced on both groups of the TB. It will run with misses forced on both groups of the data cache.

Q. Make sure that an autoreload is not in progress and does not start when reading or writing any TB ID register.

R. Do not reset the CRD or RDS bits if an SBI cycle is still in progress for the IB.

S. Do not read silo data when the silo is not locked.

T. Do not change the current mode bits in the PSL while any memory reference is being requested.

U. Do not request a READ.V.NEWPC unless prefetching is stopped.

V. Do not reset timeout bits if an SBI cycle is still in progress.

W. Because of the UNIBUS DATIP problem, the first WRITE.V following a READ.V with write check must be the corresponding write unless the read is aborted.

X. Do not set or clear MME without preventing AUTO-RELOAD.

CHAPTER 8

VAX 11/780 CONSOLE SUBSYSTEM

The VAX 11/780 console subsystem consists of four major components: an LSI-11 microprocessor (KD-11F), which includes 4K of RAM; a single floppy disk and controller; a terminal and two serial interfaces, one for remote capability; and a CPU/console interface (CIB) which includes 2K or 4K of ROM.

The console system provides three major functions:

1. Traditional "lights and switches" functions such as examine, deposit, halt, start, single instruction, etc.

2. Diagnostic and maintenance functions, including the capability to load diagnostic microcode into WCS, control execution, and monitor results; control single step clock functions; and examine key system points via a serial diagnostic bus.

3. Materialize the terminal I/O registers in the processor register space. In the VAX 11/780 system, these register are on the ID bus and are in reality a mechanism by which VAX 11/780 microcode and LSI-11 software can communicate. Therefore, this same port is also used for floppy I/O transfers and any other (software defined) communication. (See Appendix)

The functions in items 1 and 2 are implemented at the user level by a set of keyboard commands and responses at the terminal. The LSI-11 in turn controls the VAX 11/780 CPU through a set of control/status and data registers in the Q-bus I/O space, which connect to the ID bus, the V-bus (the serial diagnostic bus) and many internal points in the CPU.

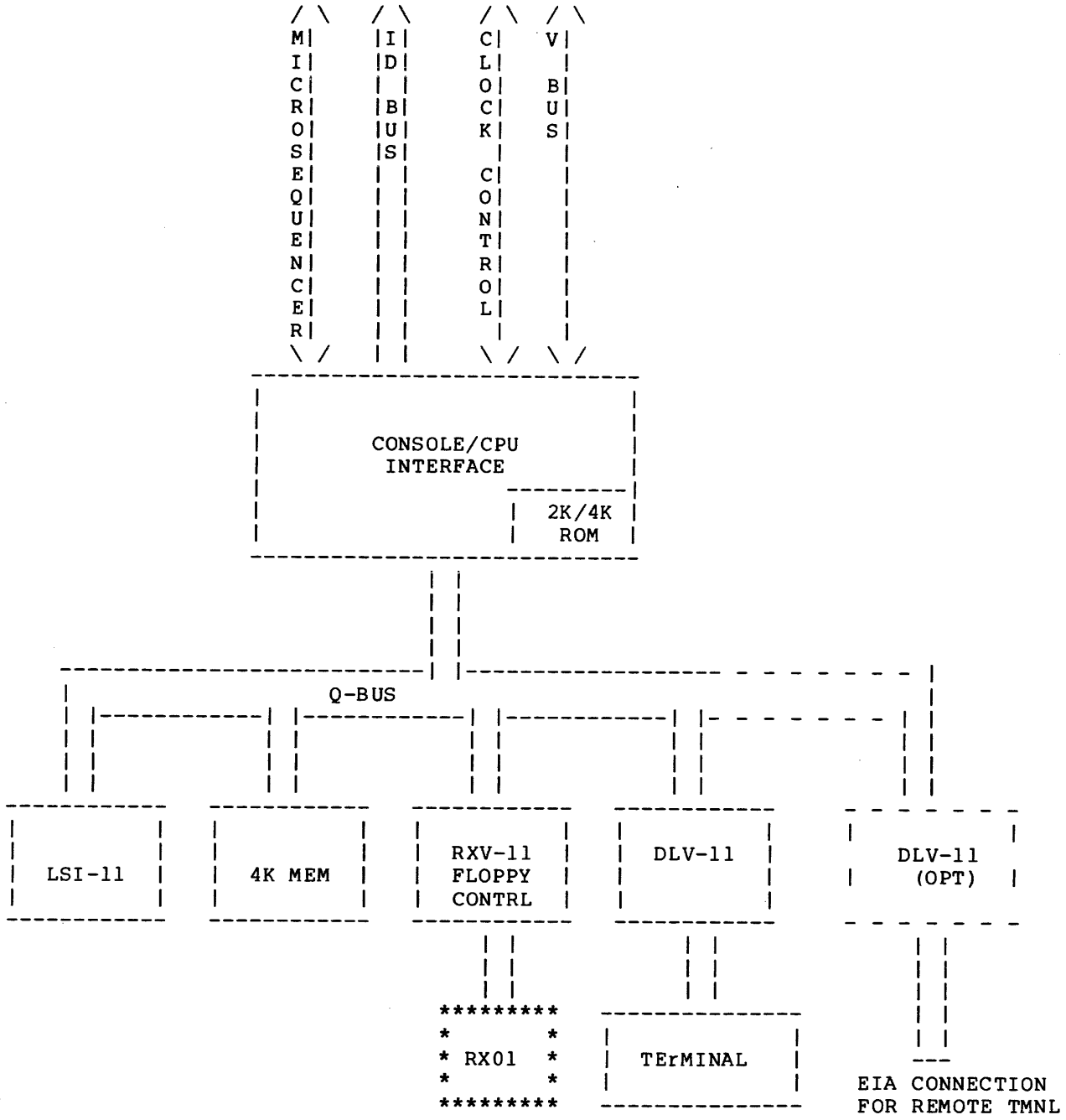

MICROSEQUENCER
IDBUS
CLOCK CONTROL
VBUS
CONSOLE/CPU INTERFACE
2K/4K ROM
Q-BUS
LSI-11
4K MEM
RXV-11 FLOPPY CONTRL
DLV-11
DLV-11 (OPT)
RX01
TErMINAL
EIA CONNECTION FOR REMOTE TMNL

Figure 8-1

8.1 THE CONSOLE/CPU INTERFACE

The Console Interface Board (CIB) is a board in the CPU backplane. It contains a Q-bus interface, an ID-bus interface and termination, and all the necessary hardware to implement the various control functions needed. In addition a Read-only memory of either 2K X 16 or 4K X 16 is provided for the "core" of the console LSI-11 software. There is, of course, a method by which the LSI-11 can force microcode execution at any microaddress. This mechanism is used to call various console service microroutines. Also, physical memory references are accomplished by using the virtual reference mechanism, but with the Memory Mapping Enable (MME) bit turned off.

FUNCTIONS IMPLEMENTED BY CIB

FUNCTIONS IMPLEMENTED BY MICROROUTINES

Virtual examine byte, word, longword

Virtual deposit byte, word, longword

Examine general registers

Deposit general registers

Examine processor register

Deposit processor register

Continue

Initialize TB, cache, etc. (result of CPU initialize signal)

Quad clear

SBI unjam

FUNCTIONS IMPLEMENTED BY HARDWARE

Stop clock

Start clock

Step one time state

Step one SBI cycle (stops in CPU T0)

Select one of four clock frequencies

Assert CPU initialization signal

Interrupt VAX CPU (terminal registers)

Halt at end of current instruction

Step single instruction

Stop clock upon microbreak match (in CPT0 of match state)

Force UPC<12> to WCS on microtrap

Force NOP on selected ROM fields

Clock VBUS

Assert VBUS loopback bit

Load VBUS registers

Read VBUS serial channels

Sense positions of auto-restart, boot, lock, and remote switches

Time-out off VAX 11/780 interrupt strobe signal, use to assert run light

Provide a write-only register on the ID bus (FM ID) (as a responder)

Provide a read-only register on the ID bus (TO ID) (as a responder)

Write to any ID bus address (requires console control and clock running)

Read any ID bus address (when clock is stopped or running)

Synchronize use of FM ID and TO ID via ready & done bits

Read clock states

Sense assertion of console acknowledge (reply to halt request)

Sense when system clock is stopped

Maintenance return (forced jump to UPC off top of microstack)

Turn floppy disk power on or off

Read ID address and direction lines (clock stopped only)

Materialize JMP into ROM at 173000 and 173002

The capability to read/write at any ID address permits register accesses to implement the following functions (and others).

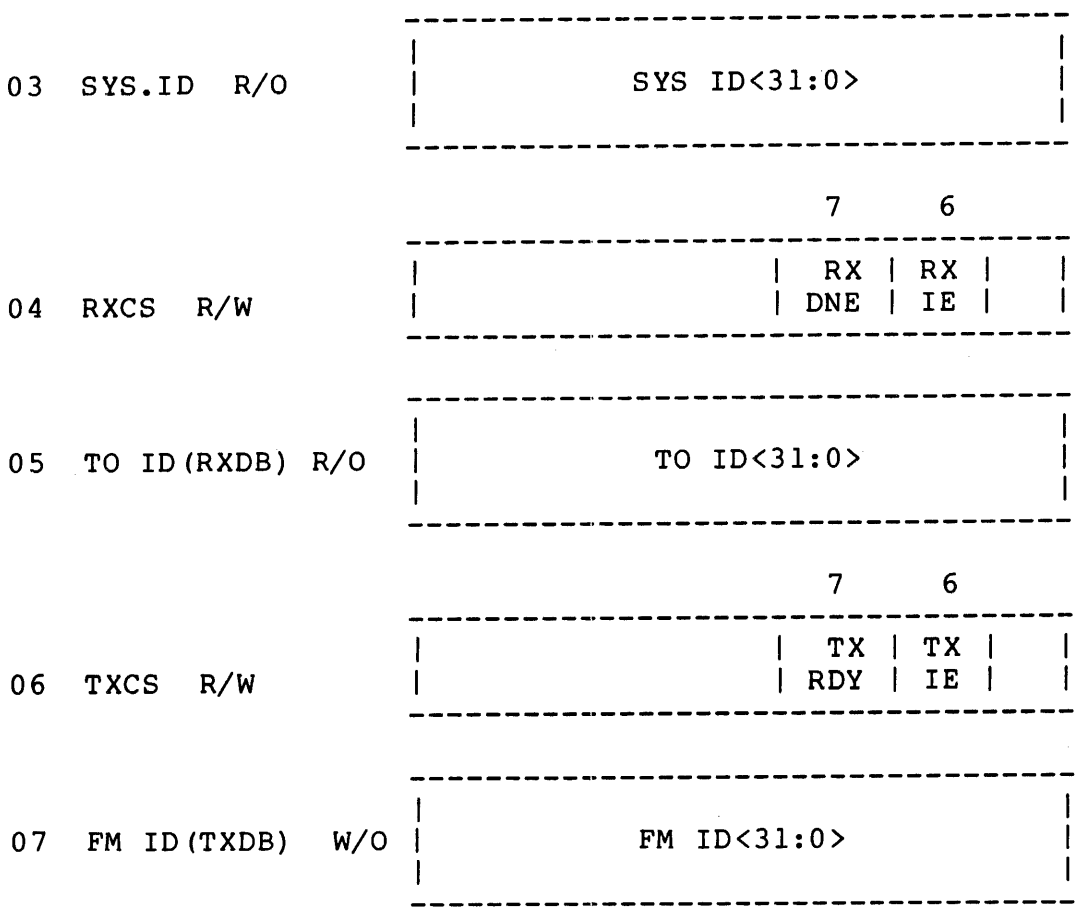
Push microstack
Pop microstack
Write microbreak
Read microbreak
Read WCS address
Write WCS address
Write WCS data
Read WCS status

8.2 ID BUS REGISTERS ON CIB

All of these registers except the SYS.ID register are essentially dual-ported between the ID bus (for VAX 11/780 access) and the Q-bus (for LSI-11 access).

This section describes the appearance of the ID bus registers on the VAX 11/780 side of the interface, i.e., as seen by VAX 11/780 microcode. The appearance of these registers to the LSI-11 is described in section 4.0.

The CIB as 5 ID bus addresses assigned, from 3(16) to 7(16).



ID registers as seen by VAX 11/780 microcode

Figure 8-2

ID03 - System Identification -
SYS ID is a read only register used to materialize the system ID register in the procreg space. (Exact format is TBS). The 32 bits come out to pins for backpanel switches. Pull-up resistors are provided on the board.

ID04 - Receiver Control/Status register
The RXCS is used to materialize the RXCS for the terminal receiver data buffer in the procreg space, and to syncronize data transfers from the LSI-11 to microcode through the TO ID register.

RXCS<7> - Receiver Done (RX DNE) - R/O
Set by the LSI-11 to indicate to microcode that valid data is available in the FM ID register. Cleared automatically by the hardware when the TO ID register is read on the ID bus. Cleared by system initialization.

RXCS<6> - Receiver Interrupt Enable (RXIE) - R/W
When set, enables an interrupt at IPL 13 and vector CC to the VAX 11/780 CPU. (See Interrupts and Exceptions specification), each time RX DNE makes a transition from 0 to 1. Only one interrupt for each transition is generated. If RX DNE is already set and RXIE goes from a 0 to a 1, the interrupt will also occur. Cleared by system initialize.

ID05 - To ID register (TO ID) - R/O

(Note: "To" and "From" are with respect to the LSI-11).

Contains up to 32 bits of data from the LSI-11, to be read by microcode. Valid only when RX DNE is set. Reading TO ID on the ID bus automatically clears RX DNE.

ID06 - Transmit Control/Status (TXCS) - R/W
The TXCS is used to materialize the TXCS for the terminal transmit data buffer (FM ID) in the space, and to synchronize data transfers from microcode to the LSI-11 through the FM ID register.

TXCS<7> - Transmitter Ready (TX RDY) - R/O
Set by the LSI-11 to indicate it is ready to accept another character in the TXDB (FM ID) register. Cleared automatically by a write to the FM ID on the ID bus; cleared by system initialize.

TXCS<6> - Transmit Interrupt Enable (TXIE) - R/W
When set, enables an interrupt at IPL 13 and vector C4 to the VAX 11/780 CPU, each time TX RDY goes from a 0 to a 1. Only one interrupt occurs for each 0-1 transition. If TX RDY is already set and TXIE makes a 0-1 transition, the interrupt will also occur. Cleared by system initialize.

ID07 - From ID register (FM ID) - W/O
Loaded by microcode with up to 32 bits of data to be passed to the LSI-11. Should be loaded only when TX RDY is a "one". Writing to FM ID on the ID bus will automatically clear TX RDY.

Use of the TO ID and FM ID registers

Under normal circumstances, when the VAX 11/780 CPU is running, only the 16 low bits of the FM ID and TO ID are used, and all references to the RXCS, TO ID, TXCS, and FM ID are the result of microcode interpretation of MFPR's and MTPR's. For MTPR and MFPR references to TO ID and FM ID, microcode does not test the state of the corresponding READY or DONE bits prior to referencing the data register; to do this would affect interrupt latency time. It is assumed under these circumstances that macro level instructions have already tested the synchronizing bits.

When the CPU is halted (i.e., in the console wait loop) microcode uses the same two registers (this time all 32 bits) to pass parameters to/from LSI-11 software. For any references other than MFPR's and MTPR's, it is microcode's responsibility to test the appropriate synchronizing bit prior to referencing the register. The bit must be a "one" before the read or write can take place.

Since the LSI-11 has no knowledge of the state of the TXIE and RXIE bits, a mechanism is provided to disable these interrupts to the VAX 11/780 and inhibit any change in the state of the "interrupt pending" flops while the console is in control and using the TO ID and FM ID registers for examines, etc.

It should also be understood that the TXCS and RXCS bits are totally divorced from the corresponding bits in the DLV-11. In program I/O mode, the LSI-11 simply passes data from/to the CIB, to/from the DLV-11.

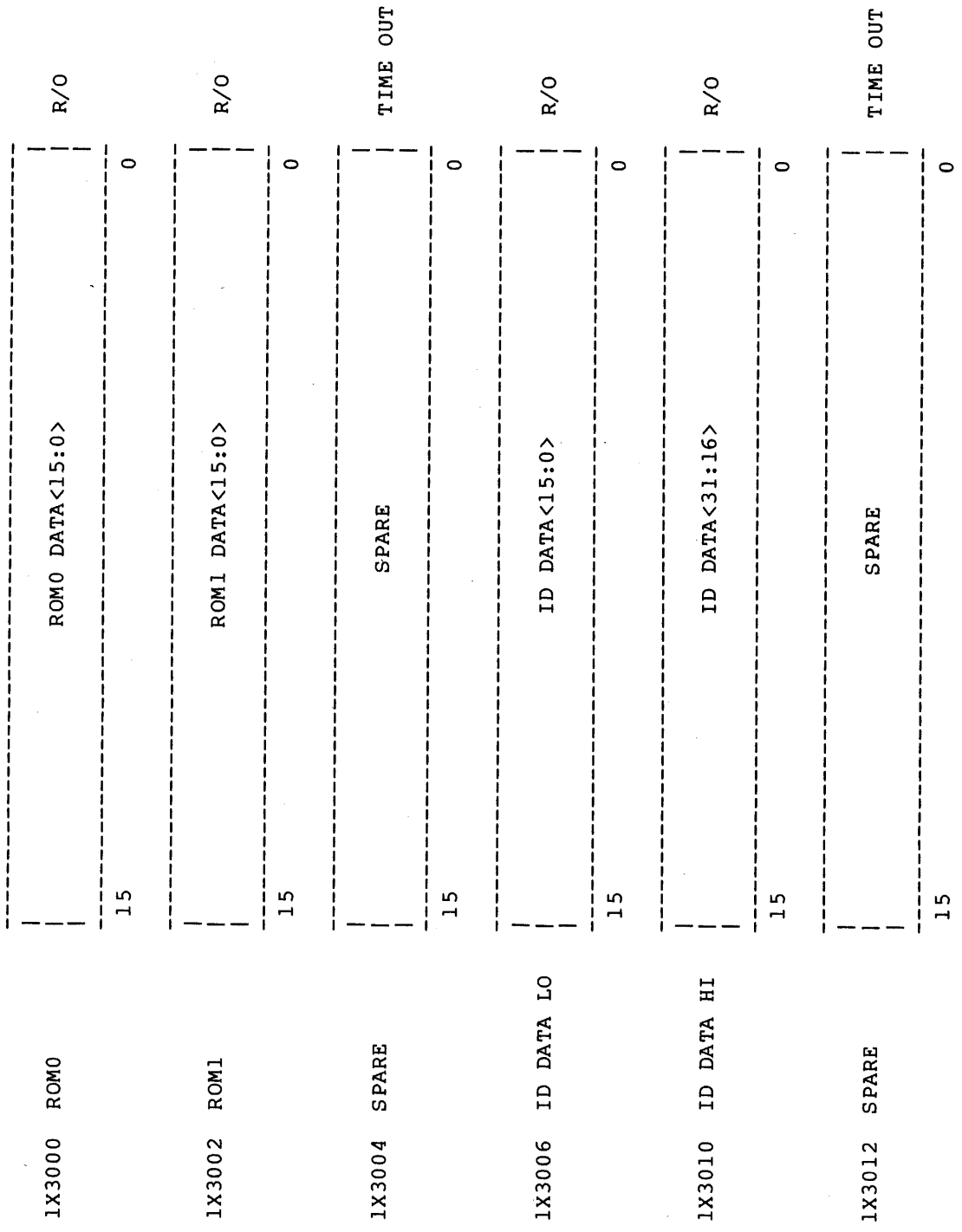
8.3 THE Q-BUS REGISTERS

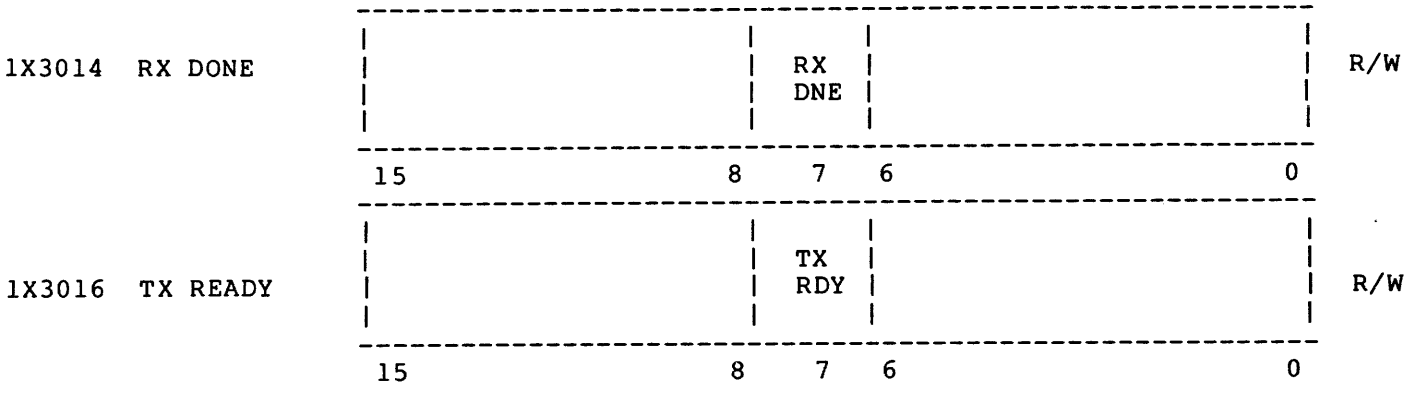
This section describes the functions and appearance of the registers on the Q-bus which the LSI-11 uses to control and monitor the CPU and handle data transfers and control of the ID bus.

For the breadboard, this block of addresses starts at location 173000(8). However, a block of 16 addresses has been assigned to the CIB, starting at 163000(8). A wire jumper will permit selecting which address group is used.

Read-only memory

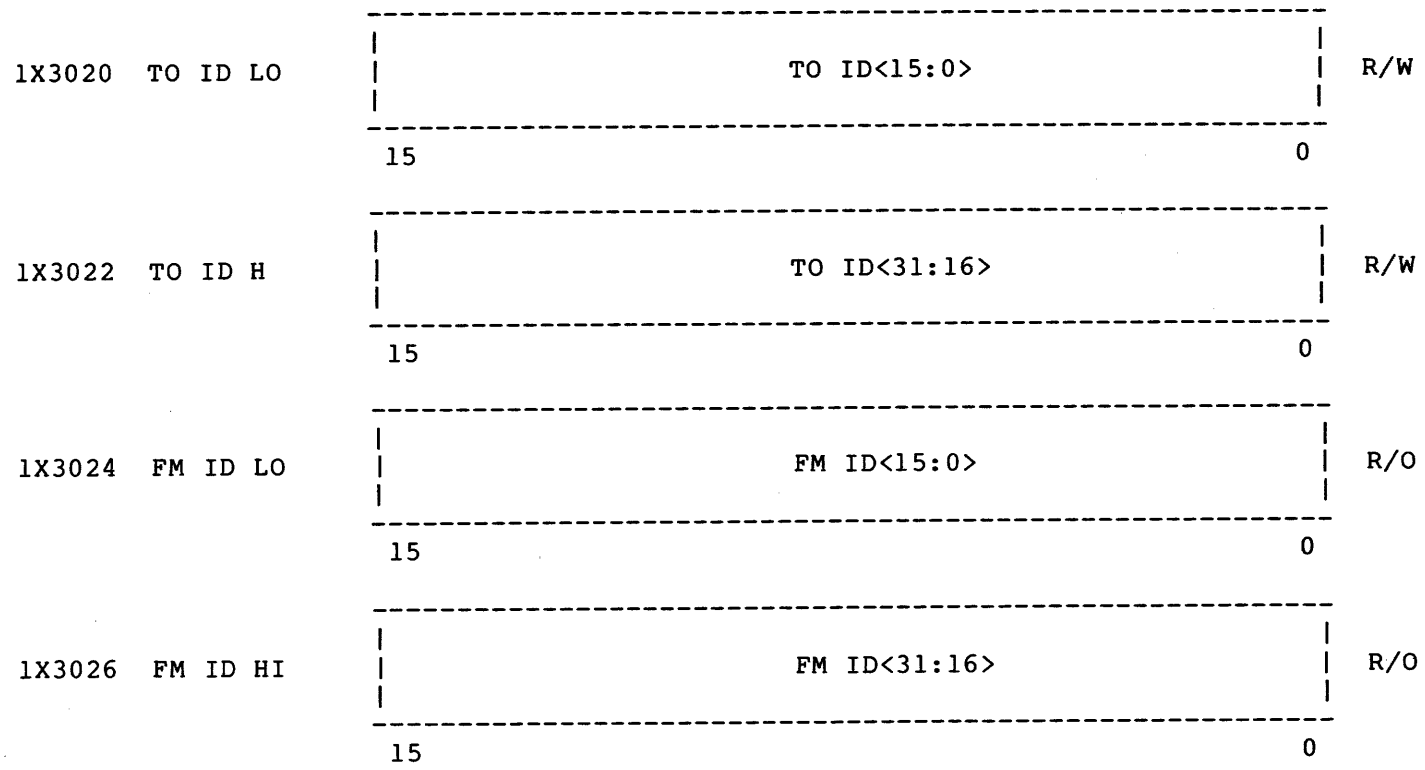
There will be either 2K words or 4K words of ROM starting at Q-bus address 140000(8) and running to either 147776(8) or 157776(8). This memory will contain the "core" of the console operating system, which includes power up routines, drivers, look-up tables, and basic areas of functionality such as examine, deposit, halt, etc.





Q-bus Registers, low 8. X=6 or 7

Figure 8-3



X3030 ID C/S

15	14	13 8	7	6	5 0	
ID CYCLE	REC WRITE	REC ID ADDRS<5:0>	ID MAINT	ID WRITE	ID ADDRS<5:0>	R/W

X3032 MCR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
HLT REQ	\	\	CPU RESET	\	MAINT RTN ENABLE	TRAP TO WCS	STAR INTR DISAB	ROM NOP	SOMM	CLK STPD	FREQ <1>	FREQ <0>	STS	SBC	PRO-CEED	R/W

X3034 MCS

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
\	\	\	FLPY OFF	BOOT	\	CNSL CMND MODE	RUN	HALT STATE	RDY IE	DNE IE	\	\	AUTO RST.	RE-MOTE	LOCK

R/W

X3036 V-BUS

15 8	7	6	5	4	3	2	1	0	
VBUS SER CHNL<7:0>	CPT 0	CPT 1	CPT 2	CPT 3		V SLFTST	V LOAD	V CLK	R/W

Q-bus Registers, high 8. X=6 or 7

Figure 8-4

Control/Data registers (refer to Figures 8-3 and 8-4)

00. ROM0 - Read only memory 0 - R/O
02. ROM1 - Read only memory 1 - R/O

These two registers, at 173000(8) and 173002(8), will contain a JMP X instruction to the LSI-11, where X is the starting location of the power-up code in the ROM. These two words are physically ROM locations 0 and 2, which therefore also appear at addresses 140000 and 140002. The LSI-11 will be configured to fetch at 173000 (a jumper option) upon power up, to execute the jump into the power-up code.

04. spare - Unused, will time-out if referenced.

06. ID bus data<15:0> - ID DATA LO - R/O
10. ID bus data<31:15> - ID DATA HI - R/O

These two registers provide visibility directly to the received data on the ID bus, and are valid only when the clock is stopped.

12. Spare - Unused, will time-out if referenced.

14: Receiver Done - RX DONE - R/W

RX DONE<7> is the "backside" of the RX DNE bit in the RXCS register. It is set by the LSI-11 to indicate to the microcode that valid data has been placed in the TO ID register. It is cleared automatically by a read reference to TO ID on the ID bus, or by system initialize. The 1-->0 transition of RX DNE will interrupt the LSI-11, if enabled.

15: Transmitter Ready - TX READY - R/W

TX READY<7> is the "backside" of the TX RDY bit in the TXCS register on the ID bus. When set by the LSI-11 it indicates to VAX 11/780 microcode that the LSI-11 is ready to accept another longword in the FM ID register. It is cleared automatically by a write reference to FM ID on the ID bus, or by system initialize. The 1-->0 transition of TX RDY will interrupt the LSI-11, if enabled.

NOTE

1) The LSI-11 has no visibility to the VAX 11/780 interrupt enables, TXIE and RXIE.

2) If the clock is running, clocking RX DNE and TX RDY from Q-bus data is done at CPT60-CPT90. THus, these bits will be stable at CPT200 when read on the ID bus. This is so microcode can be in a tight loop on RDY or DNE setting and the LSI-11 can set the bit from the Q-bus during reference on the ID bus to the same bit.

3) TX RDY and RX DNE are not automatically set by a reference on the Q-bus to FM ID or TO ID; they must be explicitly set by the LSI-11.

20: To ID register, low half - TO ID LO - R/W
22: To ID register, high half - TO ID HI - R/W

These two registers contain the 32 bits which microcode will read (into the Q-register in the CPU) when doing a read reference from ID bus address 05(16). See the TO ID register. The data in TO ID LO and TO ID HI is also the data placed on the ID bus during an ID bus write cycle invoked by the ID C/S register. They are readable for diagnostic purposes.

24: From ID register, low half - FM ID LO - R/O
26: From ID register, high half - FM ID HI - R/O

These two registers permit reading data loaded into the FM ID register on the ID bus as a result of a write reference to ID address 07(16). In addition the FM ID LO and FM ID HI registers are loaded with the ID bus data during an ID bus read cycle invoked by the ID C/S register.

30. ID Control/Status - IDC/S - R/W

The IDC/S is used to monitor the ID address and direction lines, (see ID bus specification) and control console-generated ID bus cycles directly. Combined with the ID DATA registers, the TO ID registers, and the FM ID registers, it permits reading or writing any ID address while the clock is running or in single step mode, and reading any ID address when the clock is stopped. Note that writing ID registers still requires stepping the clock if it is not running.

IDCS<15> - ID CYCLE - R/W

When written as a 1, ID CYCLE will cause ID MAINT to be asserted for one clock cycle (CPT0 to CPT0) if the clock is running, or asserted at the next CPT0 if single stepping. ID CYCLE is cleared automatically at the end of the ID cycle; therefore, the LSI-11 will normally read it as a zero, unless stepping in single time state mode. See ID MAINT description.

IDCS<14> - ID REC WRITE - R/O
IDCS<13:8> - ID REC ADDRS<5:0> - R/O

These 7 bits allow reading the state of the ID bus left address and direction lines, and are valid only when the clock is stopped. NOTE: Due to receiver inversions, these bits will be read as the complement of the logical state of the corresponding bus wire. On these lines, logical 1=+3V, which will be read as a 0.

IDCS<7> - ID MAINT - R/W

ID MAINT will assert automatically at the next available CPT0 following writing a 1 to ID CYCLE, and remain asserted until the following CPT0, when it is cleared. This bit steers the MUX in the CPU which selects the source of the ID bus address and WRITE lines from the CPU or the console. During the time ID MAINT is asserted, the ID bus address and WRITE lines will be sourced from IDCS<6:0>.

If the CPU clock is running, ID MAINT is read only, and writing to it has no effect. Note that if the clock is running, it would be read as a 0 by the LSI-11 since it is asserted only for a 200 nanosecond cycle.

If the CPU clock is not running (i.e., CLK STOPPED is set - see MCR description), then the LSI-11 may set and clear ID MAINT by writing a one or zero to it. Note that this permits statically reading ID registers via the ID DATA registers.

The address and WRITE fields may be loaded in the same instruction that writes a "one" to ID CYCLE or ID MAINT.

Both ID CYCLE and ID MAINT are cleared by LSI-11 system initialize.

IDCS<6> - ID WRITE - R/W
IDCS<5:0> - ID ADDRS - R/W

These bits are loaded with the address and direction to be used during an ID cycle invoked by ID CYCLE, or while ID MAINT is asserted. IDC/S<6>, the WRITE bit, should be set to invoke a write cycle, and cleared to invoke a read. The ID ADDRS bits are loaded in true form (unlike the REC ADDRS bits, which are read inverted). In addition, these bits are cleared by LSI-11 initialize.

For writes, data is sourced from the TO ID registers. For reads (static) the data is read in the ID DATA registers, while for dynamic reads the data is available in the FM ID registers following the cycle.

See details on use of the IDC/S.

32: Machine Control Register - MCR - R/W

NOTE

R/W1 indicates a bit that is readable; to clear it, a "one" is written to that position.

MCR15 - Halt Request - HLT REQ - R/W

Set by the LSI-11 to force microcode to the console wait loop. The VAX 11/780 CPU will recognize this request and set a HALT PENDING flop. Upon passing through the IRD state, if the HALT PENDING is set, microcode will set a Console Command Mode (CNSL CMND MODE) bit and enter the console wait loop, indicated by the assertion of HALT STATE (see MCS bit 7). A microcode CONTINUE function, invoked by the console, will reset both the HALT PENDING flag and CNSL CMND MODE, and enter IRD. If HALT REQ is still set, HALT PENDING will again set, but not until leaving IRD. Hence, a CONTINUE with HALT REQ set results in a single instruction execution. To resume normal instruction execution, the LSI-11 must first clear HLT REQ, then issue a CONTINUE function. Cleared by system initialize.

NOTE

Microcode can branch on the state of the CNSL CMND MODE bit. This is so that microcode can tell, in various error routines, whether the routine was entered as a result of some console-requested function or normal machine execution.

MCR14 - Reserved
MCR13 - Reserved
MCR12 - CPU reset - CPU RESET - R/W

When set, CPU RESET will force assertion of the (DC LO equivalent) internal initialization signal. Upon de-assertion, microcode will enter the power-up initialization microcode. Cleared by LSI-11 system initialize.

MCR11 - Reserved

MCR10 - Maintenance return enable - MAINT RET ENABLE - R/W

Writing a "one" to this bit causes a maintenance return. Specifically, the top element of the microstack is popped and J-field inputs are disabled. The result is a forced jump to the address on the top of the microstack. The actual signal to the microsequencer is asserted from approximately CPT150 to CPT50. If in single time state mode, MAINT RTN ENABLE will remain asserted from the time it was written as a "one" until the next CPT150 state is entered, when it will clear.

NOTE

The console should not attempt a MAINT RET function unless the CPU is in the console wait loop.

MCR9 - Trap to writable control store - TRAP TO WCS - R/W

When set, TRAP TO WCS will force bit 12 of the UPC to a "one" whenever a microtrap occurs, to force the trap into WCS. Should be set or cleared only when in the halt state or clock is stopped. Cleared by LSI-11 initialize.

MCR8 - VAX 11/780 Interrupt Disable - VAX 11/780 INTR DISAB - R/W

This bit, when set, disables the TX RDY and RX DNE interrupts to the VAX 11/780 CPU, regardless of the state of the TXIE and RXIE bits. Furthermore, it inhibits any change of state of the interrupt pending flops (on the CIB) in the terminal interrupt control logic. This bit is necessary to allow use of the ID registers and RDY and DNE bits for console functions without causing extraneous interrupts to the VAX 11/780 CPU.

MCR7 - Rom no-op - ROM NOP - R/W

This bit, when set, generates CLR UWORD and ABORT CYCLE in the microsequencer. This has the same effect as a STALL, to force NOP's on the various subsystem control fields so that random patterns from WCS will not produce undesired side effects during testing. If set while the clock is stopped, it is necessary to step to a CPT0 before the effect of ROM NOP will be felt. Cleared by LSI-11 initialize.

MCR<6> - Stop on microbreak match - SOMM - R/W

When a match is detected between the microbreak register and the UPC, if SOMM=1, the clock is stopped in CPT0 of the cycle in which the match is occurring, and CLK STPD will be asserted. Cleared by LSI-11 initialize.

MCR<5> - Clock Stopped - CLK STPD - R/O

This signal originates in the clock control logic and is set when the clock is not running. The signal will be negated for one clock step or one cycle when either single time state or single cycle is stepped, but the LSI-11 is not fast enough to see this. However, the signal is also used in several places on the CIB to determine whether synchronizing to the clock is necessary or not. Cleared by LSI-11 system initialize.

MCR<4:3> - Frequency select<1:0> - FR1 and FR0 - R/W

These two bits determine the clock frequency source as follows:

FR1	FR0		
0	0	-	10.0 Mhz, (normal)
0	1	-	10.525 Mhz, 5% short
1	0	-	8.925 Mhz, 12% long
1	1	-	external source

FR1 and FR0 should not be changed if the clock is running. Both are cleared by LSI-11 initialize.

MCR<2> - Single Time State - STS - R/W

When the clock is running, setting STS will cause the clock to stop, in any of the four time states. As long as STS is set, and regardless of the state of SBC, writing a "one" to PROCEED will step the clock one time state (e.g., from CPT100 to CPT150). Cleared by LSI-11 initialize.

MCR<1> - Single Bus Cycle - SBC - R/N

If asserted (and STS=0) while the clock is running, the clock will stop in CPT0. As long as SBC is set (and STS=0), writing a "one" to PROCEED will step the clock to the next CPT0 (i.e., one ROM/SBI cycle). Cleared by LSI-11 initialize.

MCR0 - Proceed - PROCEED - W/O

Writing a "one" to PROCEED will either step the clock one time state (if STS=1) or one cycle (if SBC=1 and STS=0) or, if both STS=0 and SBC=0, will start the clock running continuously, Writing a "one" to PROCEED when the clock is running has no effect. Read as a zero, cleared by LSI-11 initialize.

NOTE

1) The proper method to stop the clock is via the SBC bit, so that the stopped state is known (CPT0).

2) CPT0=SBIT1, i.e., SBI time states lead the CPU by one time state.

3) Clearing STS and SBC will not start the clock. It is necessary to write a "1" to PROCEED, after STS and SBC are cleared.

34: The Miscellaneous Control and Status register (MCS)

MCS<15> - Reserved

MCS<14> - Reserved

MCS<13> - Reserved

MCS<12> - Floppy on - FLPY ON - R/W

When set, applies power to the floppy disk drive via a relay. When cleared, removes power from the floppy. Set by LSI-11 initialize.

MCS<11> - Boot - BOOT - R/W1

Set by a 0-->1 transition of the boot signal from the control panel BOOT button. Cleared by writing a "one" or LSI-11 system initialize.

MCL<10> - Reserved

MCS<9> - Console Command Mode - CNSL CMND MODE - R/O

Permits reading the state of the CNSL CMND MODE bit. This bit is set by the microcode assertion of HALT STATE, and remains set until cleared by microcode while executing a CONTINUE function.

MCS<8> - run - RUN - R/O

This bit is the "1" side of a retriggerable one-shot clocked by the VAX 11/780 interrupt strobe signal, and will remain a "1" as long as the CPU is strobing interrupts at least every (TBS) microseconds. While the CPU is running and executing programs, negation of RUN generally indicates some type of problem; e.g., hung in a microcode loop or a hardware failure. This one-shot is also used to light the "RUN" indicator on the front panel.

MCS<7> - Halt State - HALT STATE - R/O

This is the raw decoded output of a ROM field, which is asserted if and only if the microcode is in the console wait loop. This is required because CNSL CMND MODE, which is set upon entry into the console wait loop, will remain set even if microcode leaves the loop. HALT STATE therefore is used to ascertain that microcode did return to the wait loop following a console microcode function (other than CONTINUE), and that microcode is in the wait loop prior to doing a console ID cycle.

MCS<6> - Transmit Ready Interrupt Enable - RDY IE - R/W

MCS<5> - Receiver Done Interrupt Enable - DNE IE - R/W

These two bits enable the corresponding interrupt to the LSI-11 upon a 1-0 transition of the TX RDY or RX DNE bits. If TX RDY or RX DNE is already 0, and the corresponding IE is set, the interrupt will occur. Cleared by LSI-11 initialize.

MCS<4> - Reserved

MCS<3> - Reserved

MCS<2:0> - Panel switch sense - R/O

These 3 bits sense the position of the control panel switches as follows:

MCS<2> - Auto restart switch - AUTO RST - R/O
MCS<1> - Remote mode - REMOTE - R/O
MCS<0> - Lock - LOCK - R/O

The V-bus register

In addition to the V-bus functions, V-bus <7:4> permit reading the state of the CP clock as follows:

V-bus<7> - CPT0
V-bus<6> - CPT1
V-bus<5> - CPT2
V-bus<4> - CPT3

These bits are read-only and meaningful only when the clock is stopped.

LSI-11 interrupts

Vector	Priority	What
300(8)	higher	TX RDY
304(8)	lower	RX DNE

8.4 USE OF THE Q-BUS REGISTERS

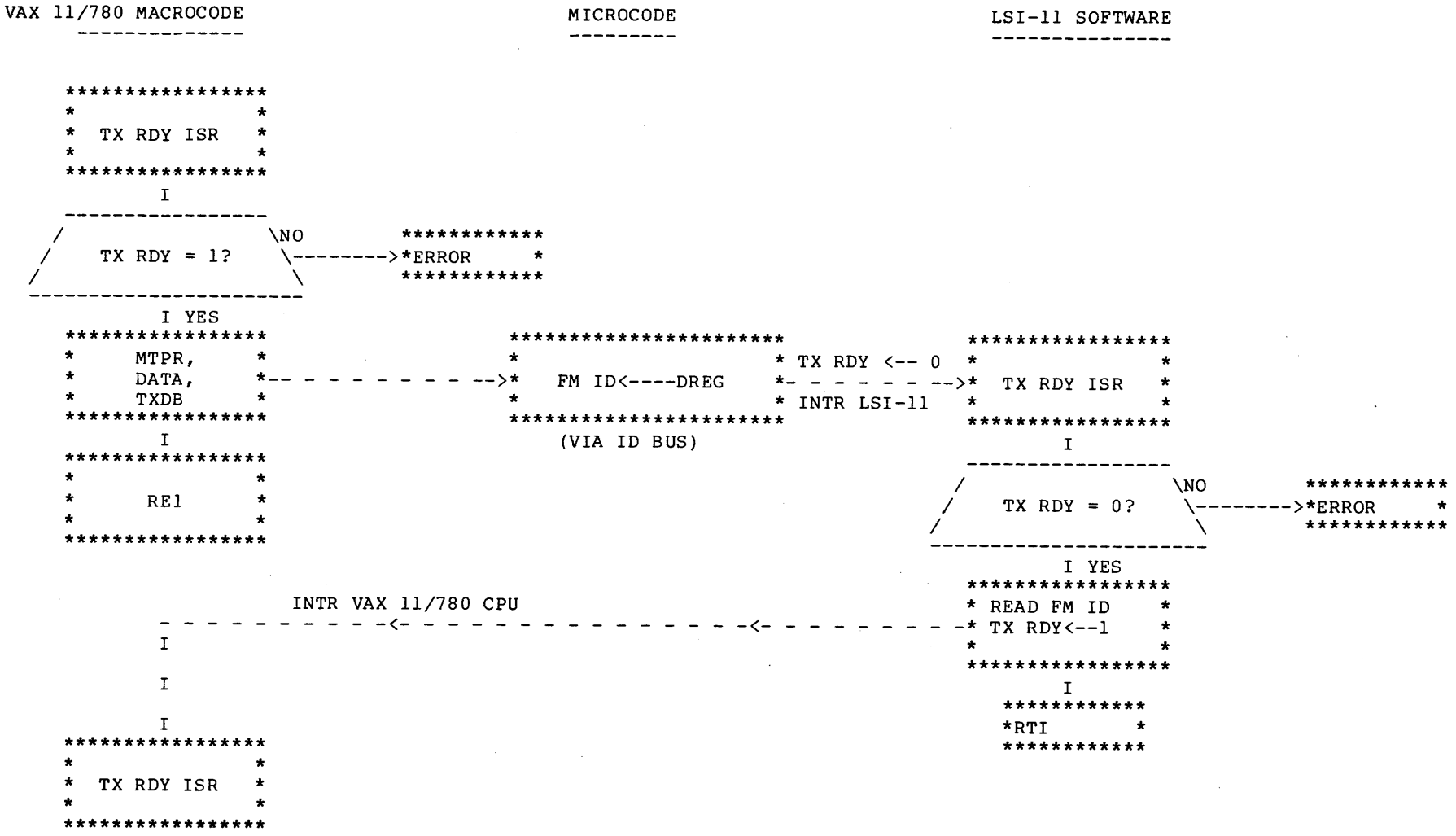
Program I/O

Program I/O mode is when the FM ID and TO ID registers are used as the TXDB and RXDB respectively when VAX 11/780 software wishes to communicate with the LSI-11 or the operator terminal, via MTPR's and MFPR's. Figure 8-5 shows the interaction between VAX 11/780 macrocode, microcode, and LSI-11 software. (ISR:=interrupt service routine).

Console microcode routines

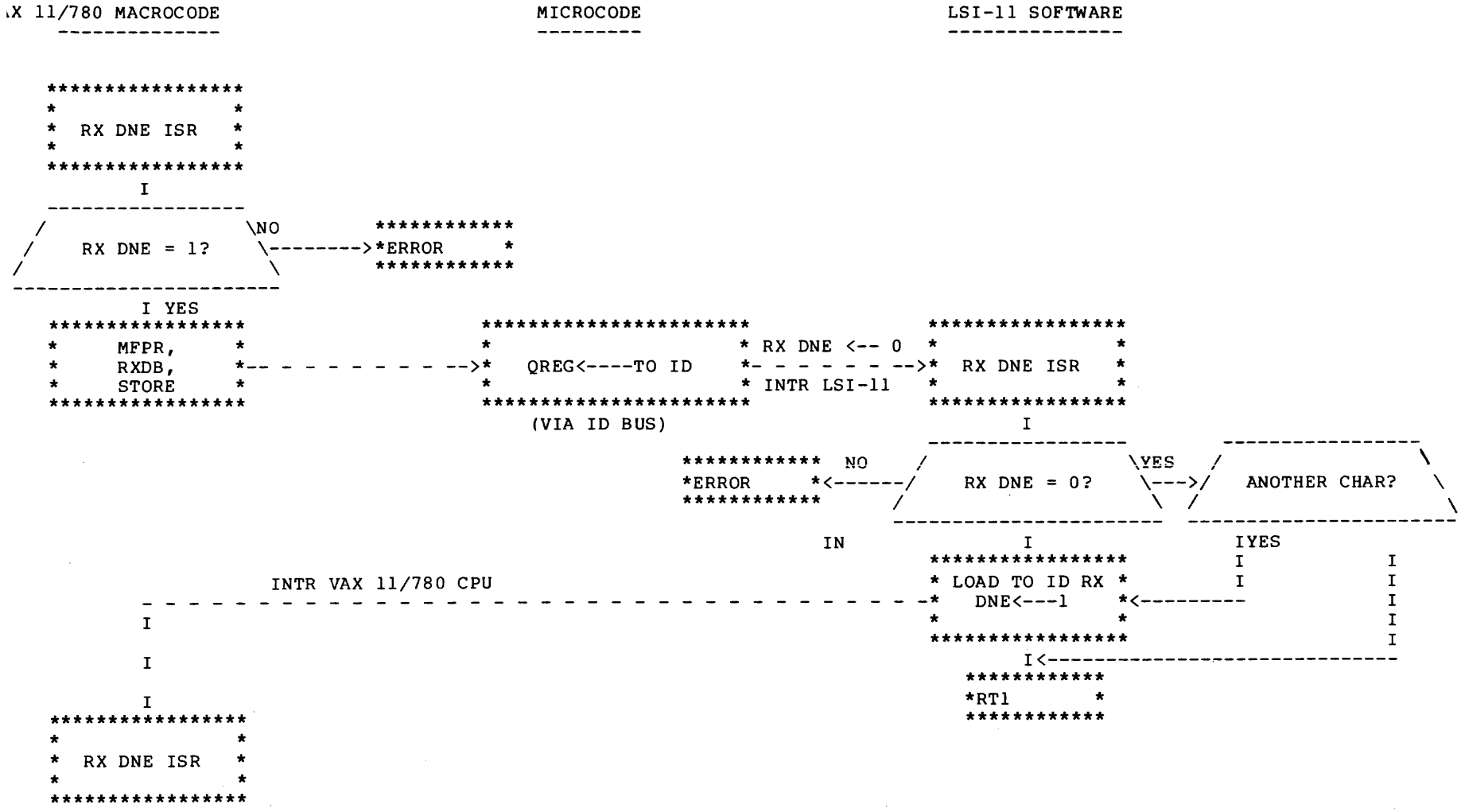
When the CPU is in the console wait loop, the console may request microcode routines to perform various functions, such as an examine virtual address. The TO ID and FM ID registers are used to pass parameters needed or supplied by these routines, and the transfers are interlocked in a manner similar to Figure 8-5, except that instead of the setting of TX RDY or RX DNE interrupting the VAX 11/780 CPU, it is VAX 11/780 microcode's responsibility to test the appropriate bit for being a "one" before reading or loading the TO ID or FM ID register over the ID bus.

The LSI-11 forces entry to those microroutines by writing to the microstack via the ID bus (see next section), which pushes the address on the microstack, then asserting MAINT RTN in the MCR. All console microroutines except CONTINUE must exit back to the console wait loop.



FOR TRANSMIT DATA

FIGURE 8-5A



FOR RECEIVE DATA

FIGURE 8-5B

There is of course the possibility that the FM ID and/or the TO ID register was in use for terminal I/O at the time of the entry to the console wait loop, either as a result of a halt instruction or an LSI-11 halt request such as for single instruction step. In order not to loose terminal data, the LSI-11 must do the following: (Figure 8-6 will help follow this).

1. If, at the time of the halt, the TX RDY bit=0, the LSI-11 must first read the FM ID register (and presumably either act on the data or at least save it) and set the TX RDY bit. Note that this will set the TX RDY interrupt pending bit (if TXIE=1). If TX RDY=1, indicating no new data, go directly to 2.

2. If, at the time of the halt, the RX DNE bit=1, indicating the TO ID register has not yet been read by microcode, the LSI-11 must save the content of TO ID and the state of the RX DNE bit. If RX DNE=0, indicating the buffer has been read, proceed to 3.

3. Set VAX 11/780 INTR DISABLE. This now "freezes" the state of the VAX 11/780 interrupt control logic, and the TO ID and FM ID, and synchronizing bits, may then be used freely for other functions.

When ready to continue, the LSI-11 must do the following:

1. Set TX RDY.

2. If RX DNE=0 upon entry (halt), then clear the VAX 11/780 INTR DISABLE and go to 3. If RX DNE=1 upon entry, then restore data to TO ID and set RX DNE, then go to 3, else restore old data to TO ID, set RX DNE, and issue the continue.

3. Re-enable VAX 11/780 interrupts; enable LSI-11 interrupts; issue CONTINUE.

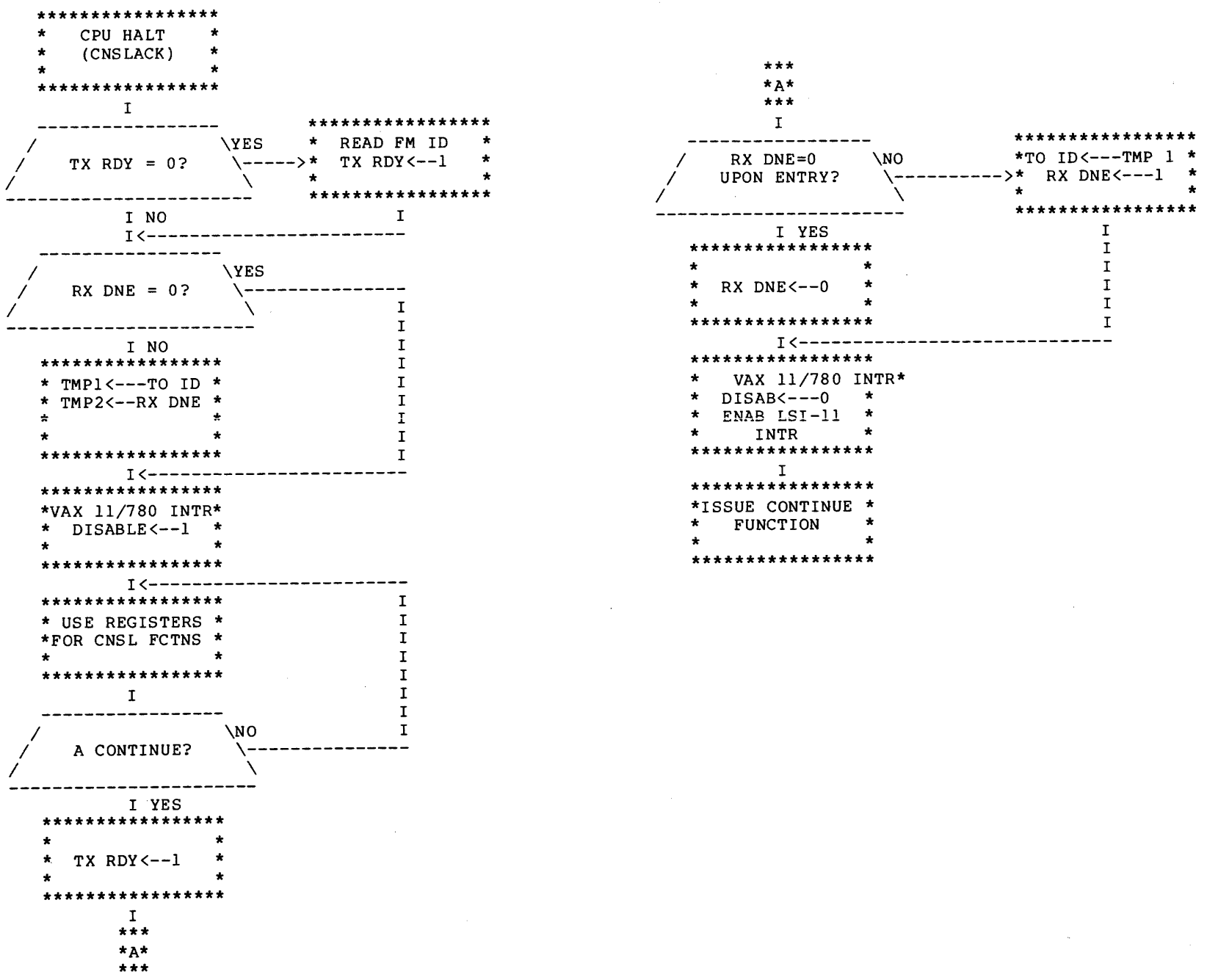

CPU HALT
(CNSLACK)
TX RDY = 0?
YES
READ FM ID
TX RDY<--1
I NO
RX DNE = 0?
YES
I NO
TMP1<---TO ID
TMP2<--RX DNE
VAX 11/780 INTR
DISABLE<--1
USE REGISTERS
FOR CNSL FCTNS
A CONTINUE?
I YES
TX RDY<--1
A
A
A
RX DNE=0
UPON ENTRY?
NO
TO ID<---TMP 1
RX DNE<---1
I YES
RX DNE<--0
VAX 11/780 INTR
DISAB<---0
ENAB LSI-11
INTR
ISSUE CONTINUE
FUNCTION

FIGURE 8-6

This sequence is necessary for two reasons: to prevent causing spurious interrupts to the VAX 11/780 CPU, and to ensure the interrupts do occur during single instruction stepping.

Direct ID bus references
---

1. Clock running

   Writing to ID registers is accomplished by first loading the data to be written into the TO ID LO and TO ID HI registers, then writing the register address, with the WRITE bit=1 and ID CYCLE=1, into ID C/S. ID CYCLE may be set by a separate instruction, but in either case the data from the TO ID register will be written into the addressed register.

   Reading is accomplished similarly, except that the ID WRITE bit=0 (i.e, read), and the data from the specified register is available in the FM ID LO and FM ID HI following the cycle.

2. Clock stopped

   Reading ID registers with the clock stopped (presumably in CPT0) is done by setting ID MAINT in the ID C/S register, and placing the desired address in the address field, with WRITE=0. Since all ID register strobes are disabled in CPT0, it is necessary to step the clock to whichever state, other than CPT0, the desired register is gated onto the ID bus. As long as ID MAINT is set, the clock is in the correct time state, and WRITE=0, the addressed register may be read through the ID DATA LO and ID DATA HI registers, and the address may be changed while ID MAINT is set. ID MAINT should be cleared prior to starting the clock again.

   It is impossible to write to ID registers when the clock is stopped without stepping through time states. However, ID writes may be accomplished by first ensuring the clock is in CPT0, then loading the desired ID address into ID C/S, along with ID MAINT=1, and the data in TO ID LO and TO ID HI. Invoking a single cycle via PROCEED will then write the TO ID data to the selected register, and clear ID MAINT. Reads may be done in a similar manner.

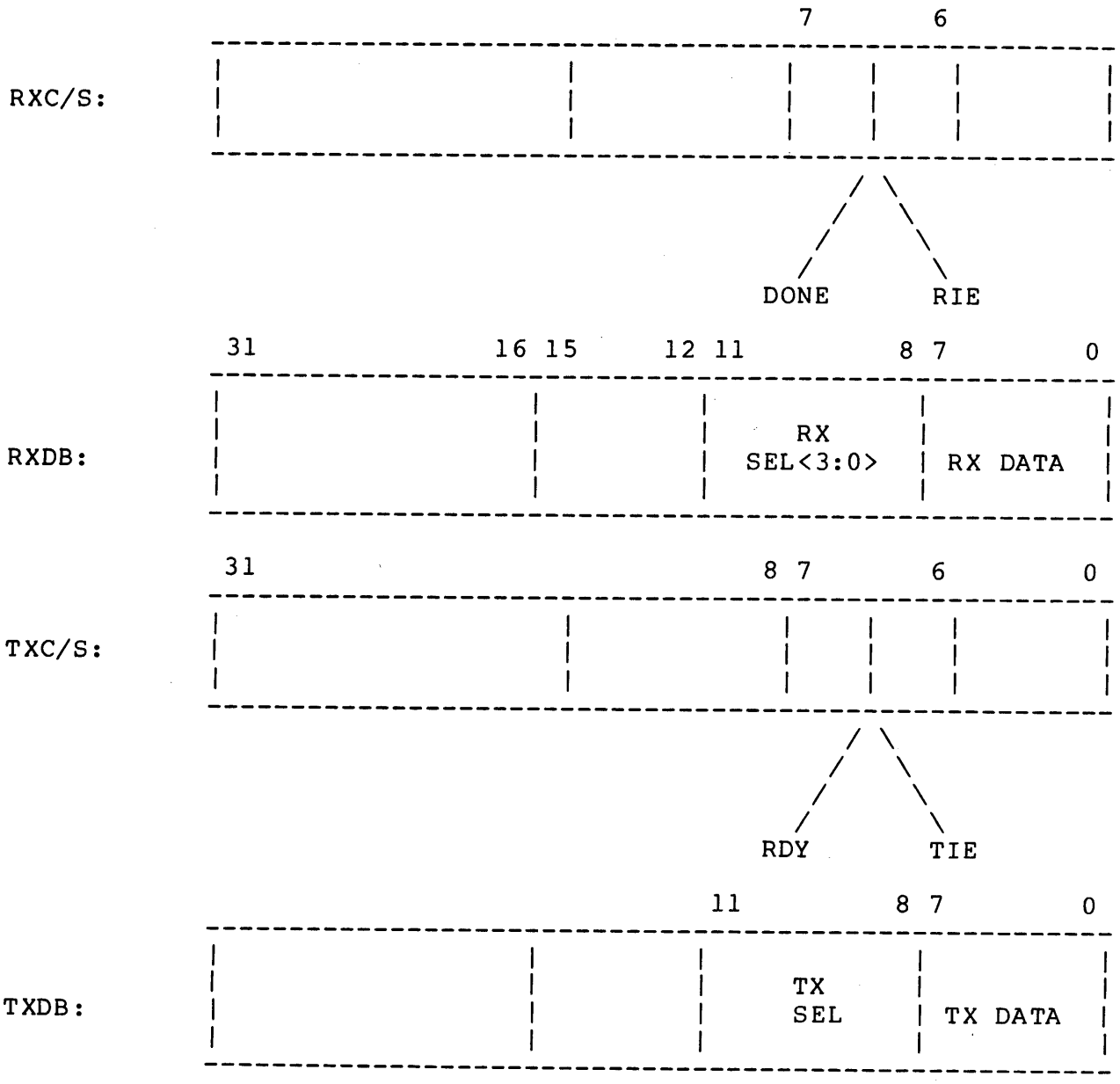
8.5 TERMINAL CONTROL REGISTERS IN THE PROCREG SPACE

The following is a proposal for the terminal communication registers which must appear in the processor register space. The interface basically consists of a transmitter and a receiver data buffer register 16 bits in width, and appropriate control and status information for each.

Note that bits <11:8> of the data buffers are used as a "unit code", with unit code 00(8) assigned to the operator terminal. This is so that the interface may be "subsetted" to a simple terminal interface on those implementations which have only that basic functionality. There may be a bit which indicates that the interface can do more than simple terminal functions; if so, that bit belongs in a CPU configuration register (TBS in Chapter 9), not in the control/status registers defined below.

Please note that this appendix describes the terminal registers as seen by macro level software via MTPR's and MFPR's, independent of specific implementations. Section 3.0, on the other hand, describes a set of registers on the ID bus as seen by microcode, and which will be used in the VAX 11/780 implementation to materialize these terminal registers (among other things).

This proposal, in some form, will be ECO'd into Chapter 9 to provide the TBS's for the terminal registers.


7
6
RXC/S:
DONE
RIE
31
16 15
12 11
8 7
0
RXDB:
RX
SEL<3:0>
RX DATA
31
8 7
6
0
TXC/S:
RDY
TIE
11
8 7
0
TXDB:
TX
SEL
TX DATA

RXC/S - Receiver control/status

RXCS<7> - receiver done - read only
Set when the data in RXDB is valid and ready to be read. Cleared by reading RXDB or by system initialize.

RXC/S<6> - receiver interrupt enable - read/write
When set, enables an interrupt to the CPU whenever receiver done becomes set. Cleared by program control or by a CPU reset.

RXDB - Receiver data buffer - read only

RXDB<7:0> - receive data
Contains one byte of data from the unit specified in RXDB<11:8>.

RXDB<11:8> - receive unit select
RXDB<11:8> specify from which logical unit the data in RXDB<7:0> originated. Logical unit 00 is reserved for the operator terminal.

Reading RXDB automatically clears receiver done in the RXC/S longword. Reading RXDB when receiver done is zero has UNPREDICTABLE results.

TXC/S - Transmitter control/status register

TXC/S<7> - transmitter ready - read only
When set, indicates that TXDB is ready to accept another character for transmission. Cleared by a write to TXDB; set by a CPU reset.

TXC/S<6> - transmitter interrupt enable - read/write
When set, enables an interrupt which occurs whenever transmitter ready becomes set. Cleared by program control or by a CPU reset.

TXDB - transmit data buffer - write only

TXDB<7:0> - transmit data
Written by program with the character to be transmitted to the logical unit specified by TXDB<11:8>.

TXDB<11:8> - transmit unit select
Written along with TXDB<7:0> to specify the logical destination of TXDB<7:0>. Logical unit 00 is reserved for output to the operator terminal.

Writing to TXDB clears transmitter ready in TXC/S. Writing to TXDB when transmitter ready is zero has UNPREDICTABLE results.

Note that when this interface is used for the operator terminal, the logical unit select fields are both all zeros and the interface appears much like a minimal DL-11. In the VAX 11/780 implementation, non-zero unit select fields may be used to initiate I/O with other devices (namely, the floppy), or general software communications with the LSI-11. Should software attempt communications to/from units other than unit zero on those implementations which have only the terminal, bits <15:8> are ignored and the characters still go to the terminal, which possibly prints garbage. This is probably acceptable, since it is the result of a clear-cut software error.

CHAPTER 9

VAX 11/780 ACCELERATOR INTERFACE

The VAX 11/780 Accelerator interface provides mechanisms for:

1. Main Machine/Accelerator synchronization.

2. Explicit control of the Accelerator machine state by the main machine.

3. Transferring of main machine control to Internal WCS space by the Accelerator at any arbitrary Decision Point.

4. Instruction stream interpretation by the Accelerator.

5. Transfer of 32 bit data words from the ISB to the Accelerator, from the Accelerator to the main machine Data Path or from the main machine Data Path to the Accelerator. Also, provision is made for incorporating within the accelerator single or multiple copies of the processor general register set.

6. Transfer of Accelerator status flags into the main machine micro control machine branch logic.

7. Transfer of condition code information into the main machine condition code logic.

Each of these mechanisms is defined in such a way that the use may be defined to suit a general Accelerator.

Each of these functions will be described in more detail in succeeding sections of this specification.

In summary, the Accelerator interface allows the Accelerator to examine the instruction stream and "pick out" instruction operation codes which it determines belong to it and then transfer control of the main machine to an "Accelerator handler" in main machine WCS while it processes the instruction.

9.1 DEFINITIONS

INSTRUCTION STREAM BUFFER (ISB) - The Instruction pre processing hardware which pre fetches and decodes operation codes and operand specifiers from the process Instruction Space.

CENTRAL PROCESSING UNIT (CPU) - The processing hardware consisting of control store, control machine, registers and data processing hardware which comprises the basic VAX 11/780 computer.

ACCELERATOR - An optional, autonomous, data processing machine which operates in conjunction with the CPU to increase processing speed for specific instructions within the VAX-11 architecture. This is accomplished by transparently (at the macro level) overriding emulation of those instructions by the CPU and instead causing the required processing to be accomplished in the Accelerator, whose processing elements are optimized to perform the required operations.

9.2 INTERFACE SPECIFICATION

Figure 9-1 depicts a block diagram of a VAX 11/780 ACCELERATOR interfaced to the VAX 11/780 Computer System. This diagram will be referred to throughout the remainder of this specification.

9.3 GLOSSARY OF INTERFACE SIGNALS

1. OP CODE: Eight bits of information from the ISB which are a copy of the data contained in byte 0 (OP CODE byte) of the ISB.

2. EXECUTION POINT COUNTER - Three bits from the ISB which denote where in the process of instruction execution of the CPU is. The case of execution point counter = 0 and an enable signal from the ISB is used to determine that the CPU is presently in the IR DECODE state.

   The Accelerator also uses the Execution Point Counter to determine when to force CPU control to WCS.

3. SRC1 SPECIFIER TYPE: Three signals from the ISB which determine the location of data referenced by the Operand Specifier in byte 1 of the ISB. The encoded locations are:

   CODE

   011 Data is in General Register
   111 Data is in Memory
   110 Data is Short Literal
   101 Data is Immediate

   All other codes cannot occur.

4. SRC2 SPECIFIER = REGISTER: One bit which when asserted indicates that the byte of Data in byte 2 of the ISB would evaluate to Register if it is an operand specifier. The only time this is meaningful is if SRC1 specifier is one byte in length.

5. VALID: Four signals from the ISB which indicate to the accelerator whether or not the information on the OP CODE, SRC1 SPECIFIER TYPE and SRC2 SPEC = REGISTER lines is valid. The signals are as follows:

   BYTE 0 VALID (OPCODE VALID)
   BYTE 1 VALID (SPECIFIER 1 VALID)
   BYTE 2 VALID (SPECIFIER 2 VALID)
   STALL + SVC (ALL INFORMATION VALID)

6. REGISTER NUMBERS: Two four bit quantities which designate the registers denoted by SRC1 and SRC2 operand specifiers if they are indeed Register Operand Specifiers. These may be used by the accelerator to pre-fetch register contents at the beginning of each IRD to expedite operations involving register data.

   Note that these register numbers will be undefined for operand specifiers other than SRC1 = SRC2 = Register.

7. DECISION POINT OVERRIDE: A signal from the ACCELERATOR to the CPU control machine which, when asserted, will unconditionally assert Address Bit 12 on input of the micro sequencer address mux. The effect of this is to cause transfer of control to the internal WCS module at the current execution.

8. ID BUS: The 32 bit bidirectional bus which is the main mechanism whereby data is entered into the Accelerator.

9. ACCELERATOR CONTROL FIELD: This is a two bit field used to command the operation of the Accelerator. Three of these codes are fixed use and the fourth is accelerator dependent. The codes are:

```
ACF   00 - NO OPERATION
      01 - CPU SYNC
      10 - ACCELERATOR TRAP
      11 - ACCELERATOR SPECIFIC CODE
```

The CPU trap and CPU sync codes will be used by the power up or abort micro routines. Therefore, all accelerators must use these codes.

CPU SYNC: One signal, derived from the ACF field, which is used to synchronize CPU and Accelerator functions. It functions as a binary semaphore or flag which is tested by the Accelerator when synchronism is required.

ACCELERATOR TRAP: A signal from the ACF Field which when asserted will cause a transfer of control within the Accelerator to the micro address specified by the TRAP ADDRESS word in the next micro cycle.

10. TRAP ADDRESS: 3 bits from the CPU control word which specify explictly the micro address in the ACCELERATOR Control space to which ACCELERATOR Control is to be transferred. These 3 bits are formed by redefining the (USI) field in the CPU control word.

11. ACCELERATOR STATUS FLAGS: Three bits which are passed from the ACCELERATOR to the CPU control machine branch logic. These bits can be tested by the CPU by selecting BEN 6. The meaning of these bits varies with the state of the CPU/Accelerator machine combination.

Once convention is established, however, in cases where synchronization is required. ACC<00> is defined AD HOC to be the signal Accelerator Sync. This signal is tested by the CPU at synchronizing points and together with CPU sync forms a bidirectional synchronization interlock. NOTE: That ACC<02:01> remain free of definition even during synchronization. Therefore, information may be passed to the CPU and interpreted in context at each synchronization point by the ACCELERATOR if so desired.

12. ACCELERATOR CONDITION CODES: The Accelerator has access to the PSL condition codes from the CPU. For each instruction implemented by the Accelerator, the PSL condition codes may want to be changed. Therefore, the Accelerator takes the PSL condition codes at the start of the instruction and returns four new condition code bits based on the result of the operation and the previous condition code states. NOTE: That these return lines are different than the condition codes coming to the accelerator. The returned condition codes are latched by the CPU. This information is then transfered to the PSL condition codes in the next CPU micro state by setting the UMSC field to a 6.

13. GENERAL REGISTER ADDRESS: Four signals from the CPU control machine which are latched copies of the SPA address (latched in the CPU at T150). During the second half of each micro cycle (T100-->T0) the address lines of the internal ACCELERATOR General Register sets are forced to agree with this address. It is during this time that any updates to the CPU general register are copied into the ACCELERATOR General Registers Copies.

14. GENERAL REGISTER WRITE ENABLE: Two bits which encode how much, if any, of the general register addressed by the General Register ADDRESS lines is being written in the CPU. The codes are as follows:

WREN	01	00	
	1	1	No Write
	1	0	Write Byte (Byte 0)
	0	0	Write Word (Bytes 0, 1)
	0	1	Write Long Word (Bytes 0, 1, 2, 3)

These bits are latched in the CPU at T150 of each micro cycle.

15. GENERAL REGISTER UPDATE BUS: 32 data lines from the CPU to the Accelerator. These lines are a buffered copy of the data inputs to the CPU general register sets. This is the data written into the ACCELERATOR General Register Copies when an update is indicated by WREN<00:01>. This bus is also used to return data back to the CPU. It is the task of the Accelerator to decide which data to return. Control of the direction of transfer on this bus is done by a control code in either UQK or the UDK fields in the CPU.

16. ISB CALL: A signal from the ISB to the ACCELERATOR which when asserted indicates that I Stream Data (either short literal or Immediate Data) is being driven onto the ID Bus by the ISB during the current cycle.

9.4 ACCELERATOR INTERFACE OPERATION

CPU<-->ACCELERATOR INTERFACE

The main function of the CPU/ACCELERATOR Interface is to provide synchronization between the two cooperating processes and to pass status information for the purpose of altering control flow in either or both machines.

Synchronization is obtained by the use of semaphores. When synchronization is required, the following sequence of events transpires (refer to figure 9-1). For the purpose of illustration synchronization of a data transfer from the CPU to the ACCELERATOR is discussed. NOTE: That this transfer of condition codes must be done at a syncronization point defined by accelerator and CPU micro code.

9.4.1 Data transfer

The example of Figure 9-2 shows a transaction in which a data word is being fetched from memory by the CPU. In this example, the ACCELERATOR is waiting for the Data and indicates such by asserting ACCEL SYNC. In each micro cycle while it is waiting for CPU SYNC the ACCELERATOR reads data from the ID bus and treats it as if it were the data it was expecting. In some cases, the ACCELERATOR may even begin processing this data if it can safely do this without destroying any internal information required at a later time.

Meanwhile, the CPU generates the Virtual Address of the required data and completes the memory reference.

Note that any faults encountered in this memory reference have no effect on the ACCELERATOR state - hence servicing of TB faults, cache parity errors, unaligned data traps, etc., can occur transparently.

Once the memory reference has been successfully completed and the data safely stored in the CPU D register, the CPU drives it onto the ID bus and asserts CPU SYNC. In this state, it tests for ACCEL SYNC and finding it asserted, continues.

The ACCELERATOR, meanwhile, has finally received CPU SYNC indicating that the data which it has received was indeed the data it required and it continues processing.

9.4.1.1 Initial data transfer - The bi-directional interlock shown in Figure 9-2 is not always necessary. In some cases, (noteably during initial argument loading or whenever the ACCELERATOR is guaranteed to be idle (i.e., waiting for data input) the test for ACCEL SYNC by the CPU may be safely bypassed as in Figure 9-3.

Note also in Figure 9-3 that the ACCELERATOR is designed in such a way that it is, after receipt of the first argument, continuously using ID bus data for processing as if it were valid data and storing the result in a scratch register while waiting for CPU SYNC. When the CPU SYNC is received, indicating that the data is on the ID BUS in the current cycle, the ACCELERATOR has already successfully completed one micro cycle of its required execution.

9.4.2 Accelerator control

In addition to synchronization of the ACCELERATOR and the CPU, the interface provides two mechanisms whereby the CPU may explicitly alter control flow in the ACCELERATOR.

9.4.2.1 Accelerator trap - The first is the ACCELERATOR TRAP function. When the CPU wishes to force the ACCELERATOR control program to a specific uaddress it asserts this uaddr on TRAP ADDRESS<03:00> and asserts ACCEL TRAP. (See Figure 9-4).

In the following ustate, ACCEL control will be unconditionally transferred to this micro address. Figure 9-4 illustrates how the CPU would use the ACCEL TRAP function to cause a transfer of control within the ACCELERATOR control machine to uAddress 6. This mechanism would be used primarily for initialization of the ACCELERATOR and perhaps in some catastrophic fault conditions.

9.4.2.2 Alternate trap function - Each implementation of Accelerator will likely require a means of subroutine calls. This can be implemented in a few different ways. One approach would be the complex scheme of a micro stack with call and return capabilities. Another means could be a trap to a specific address. This may be done by a field of bits located in the Accelerator maintenance register. This would allow routines to be entered where there is enough overhead available to allow a constant to be written into the status register. These traps could also be defined to imply syncronization points.

The most useful need for this appears to be for micro diagnostics.

9.4.2.3 CPU branches - As has previously been discussed, the primary mechanism for passing control/status information from the ACCELERATOR to the CPU is the three bit ACCEL STATUS FLAG Field.

One of these three bits (i.e., bit 0) is predefined to be ACCEL SYNC. The remaining two however may be used to convey different information at different times. Within the CPU these three bits form one branch

enable input into the control machine. Thus, at any synchronization point, an eight way branch within the CPU can be used to test internal Accelerator conditions (ACC<00> is predefined as ACCEL SYNC.

The ACCELERATOR control machine may, at different sync points within a given execution, switch these status flag inputs to various internal hardware outputs such as "overflow detected", "undefined variable", etc., or drive the bits directly. The only requirement is, of course, that the CPU recognize what significance these bits have in current context.

9.4.3 System clock

One important item which has not been mentioned yet but rather assumed is the SYSTEM CLOCK input to the ACCELERATOR. Both the CPU and ACCELERATOR use this system clock for timing. Furthermore, both control machines utilize synchronous 200 nsec u states (i.e., ACCELERATOR T0 = CPU T0 etc.) as far as the control interface is concerned.

Clock inputs are differential ECL and consist of decoded T0, T50, T100, T150 as well as Tph and the two phase clocks. (See VAX 11/780 Clock Spec.) Thus, with some care 25 nsec intervals within the 200 usec ROM state can be established within the data prcoessing section of the ACCELERATOR.

9.5 DATA INTERFACE

Connecting the ACCELERATOR and the CPU are two 32 bit data busses, the System ID Bus and the ACCELERATOR GEN REG Bus.

9.5.1 Data to accelerator

Data is transfered to the Accelerator by means of the system ID BUS. This transfer is done by the CPU asserting data onto the ID BUS and issuing CP SYNC. This implies that the transfer is done at a sync point. Note that no ID bus address is required for this transfer. The Accelerator simply accepts the current ID bus data.

9.5.1.1 Data from accelerator - Results returned from the accelerator is done via the Accelerator General Register Bus. This transfer is done by the CPU selecting the Accelerator data onto the 3-state general register bus. This implies that this transfer is done at a syncronization point where the data is meaningful.

9.5.2 Alternate data transfers

The data returned from the Accelerator could also be accomplished over the system ID bus. There are two drawbacks to this scheme. First is the timing consideration of when the result data is stable it must be stable sooner for an ID bus transfer. The second drawback is that the ID bus data can only go into the CPU Q Register. This implies that if the data is to be written to memory, an additional state of overhead is added.

9.5.3 Accelerator status registers

In addition to serving as a path for input and output data, the System ID bus provides READ/WRITE Access to the ACCELERATOR STATUS and MAINTENANCE registers. The ACCELERATOR STATUS register is accessible to macro level software via MTPR and MFPR instructions. Both registers are accessible to the console processor.

By examining this register (bits 0 - 3) the console (or macro level software) can determine what type (if any) ACCELERATOR is installed.

By writing bit 15 of the ACCEL STATUS register the console or macro level software may at any time turn the ACCELERATOR on or off. Turning the unit off will disable all control output signals which effectively removes the ACCELERATOR from the system. This allows the system to continue operation in degraded mode for most ACCELERATOR failures.

Bits 30 thru 27 contain error information based on the particular accelerator type.

Bit 31 is an error summary flag which is set when any of the error bits (30:27) are set.

9.5.3.1 Accelerator maintenance register - This register contains information useful to micro diagnostic programs. (Accelerator Dependent)

9.5.4 General register updates

A second method of passing data to the ACCELERATOR would be through the processor General Register set. Provision for maintaining "n" copies of the processor General Register set within the ACCELERATOR is provided. The ACCELERATOR may Read data from these internal copies from T0 to T100 of any micro state. It then relinquishes control of these copies to the CPU via the General Register Address, Data and Write enable lines of the CPU/ACCELERATOR interface.

During this portion of the cycle the CPU will cause the ACCELERATOR General Register copies to be written (updated) in an identical manner to the CPU general register sets.

Note that the ACCELERATOR General Register copies are WRITE ONLY Memory to the CPU and READ ONLY Memory to the ACCELERATOR. Operations with destinations within the general register set processed by the ACCELERATOR still require passing of the result through the CPU D register. The purpose of this section of the DATA INTERFACE is to provide rapid access to data contained in the General Register sets by the ACCELERATOR.

9.6 ISB INTERFACE

A key to the operation of the ACCELERATOR is enough visibility of the I stream and CPU machine state to be able to determine:

1. When to begin processing an instruction (i.e., when the CPU is in IRD).

2. When to force the CPU control flow into the WCS Accelerator handler package.

This is accomplished by, in the ACCELERATOR quiescent state (WAIT) examining the output of an "ON BOARD" instruction decode network. All u address targets for the ACCEL control machine are the WAIT state except for the instructions for which the unit was designed. Furthermore, this target uaddress is forced to the WAIT STATE address except when the CPU is in IRD, no interrupts or exceptional conditions are pending, and the ISB data being supplied is valid.

When the ACCELERATOR has determined that it should begin execution, it leaves WAIT and the on board decode has no further function until the next IRD state.

At some point in the CPU control program, the ACCELERATOR will determine that the general flows provided in the CPU micro code do not efficiently serve its requirements and it will assert EXECUTION POINT OVERRIDE. This will force a one to micro address bit 12. The net effect of this action is to force transfer of control in the CPU control machine to the WCS module at the next Decision Point.

Note that the 8 bits being asserted by the Instruction Buffer are unmodified by this action. Therefore, each target uaddress which might occur in the normal flow must be duplicated in the WCS handler. Worst Case this would be 256 locations, but this number should be considerably smaller.

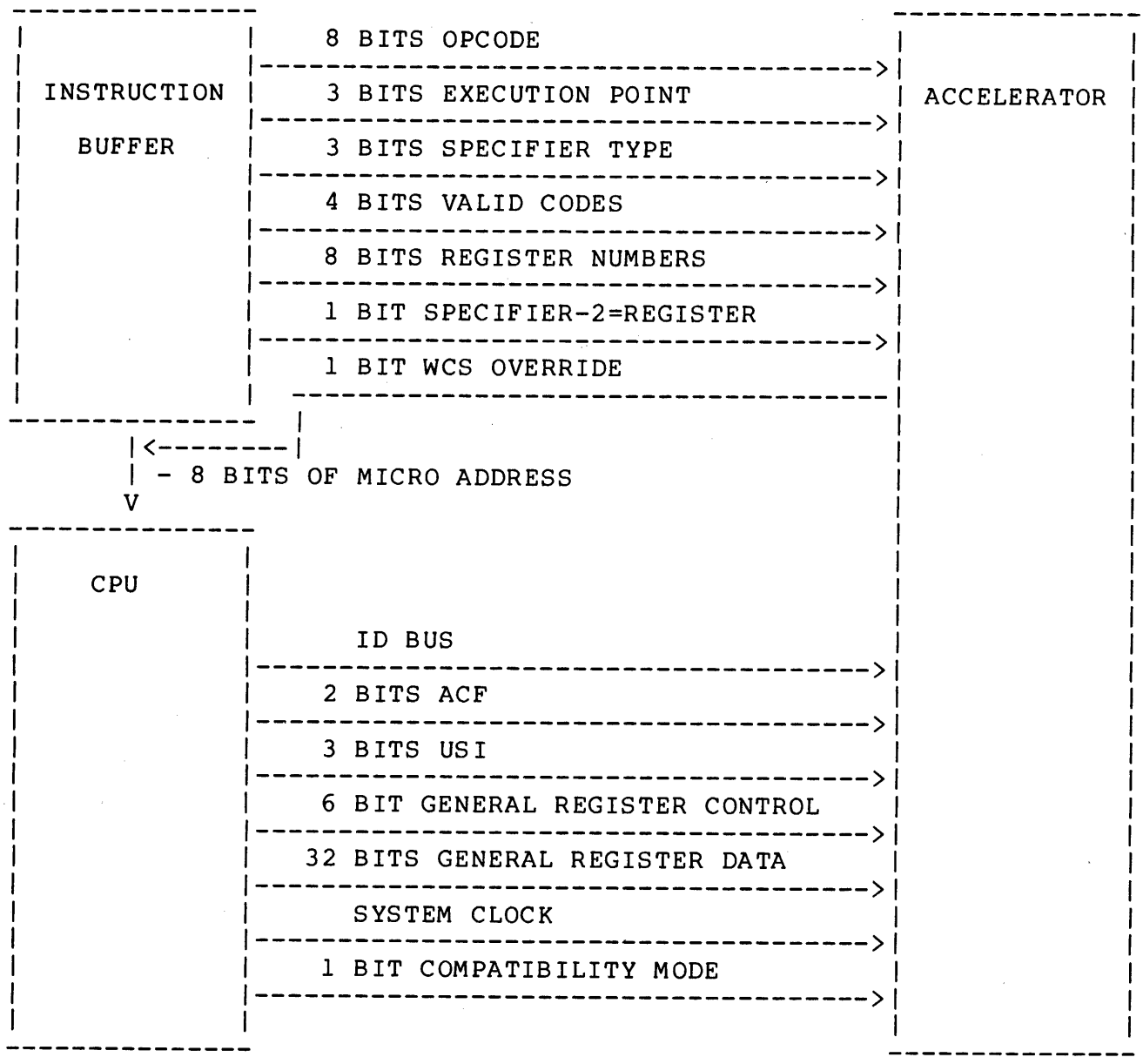


FIGURE 9-1

BLOCK DIAGRAM OF VAX 11/780 ACCELERATOR SUBSYSTEM INTERFACE

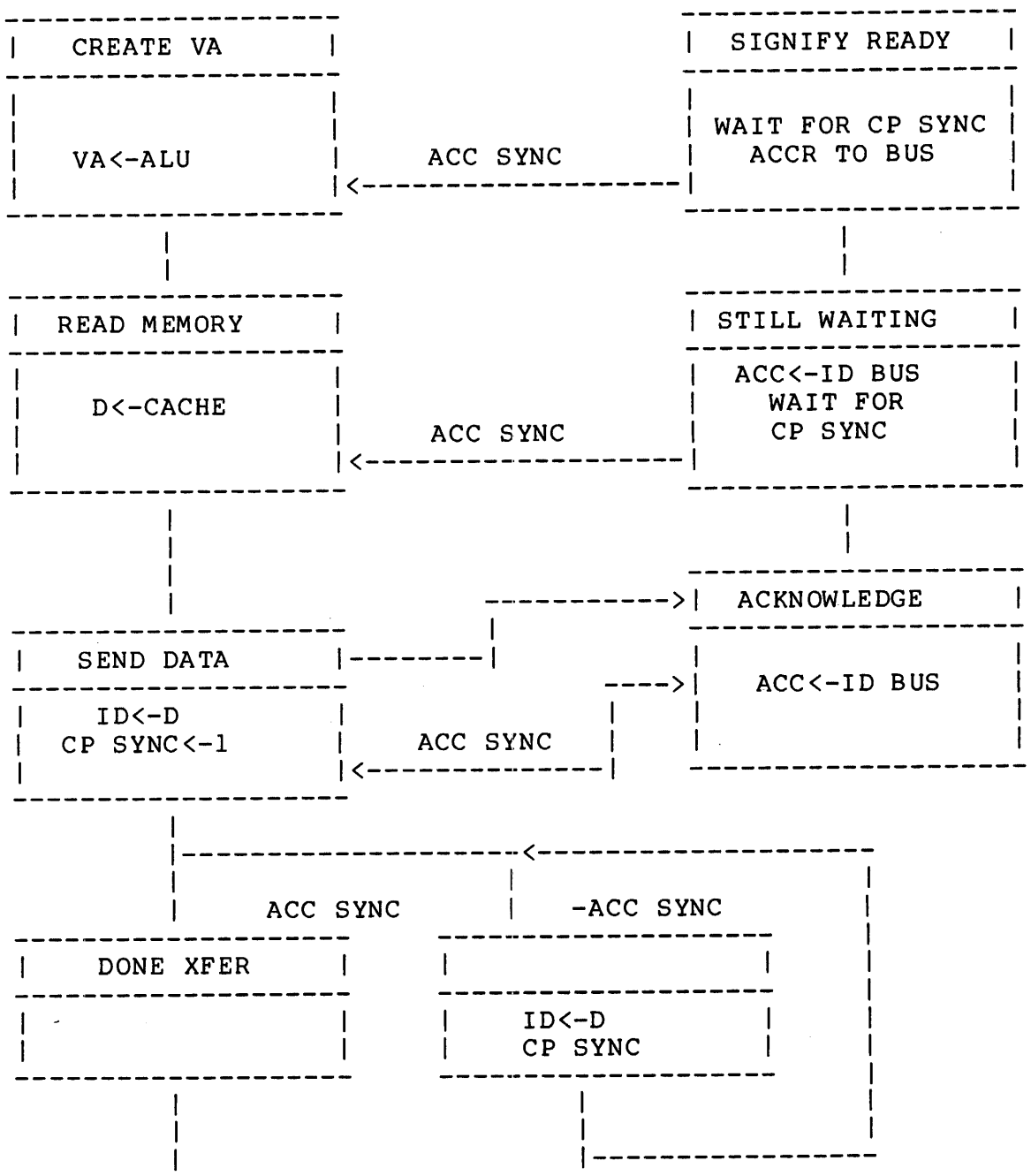

CREATE VA
VA<-ALU
ACC SYNC
SIGNIFY READY
WAIT FOR CP SYNC
ACCR TO BUS
READ MEMORY
D<-CACHE
ACC SYNC
STILL WAITING
ACC<-ID BUS
WAIT FOR
CP SYNC
ACKNOWLEDGE
SEND DATA
ID<-D
CP SYNC<-1
ACC SYNC
ACC<-ID BUS
ACC SYNC
-ACC SYNC
DONE XFER
ID<-D
CP SYNC

FIGURE 9-2

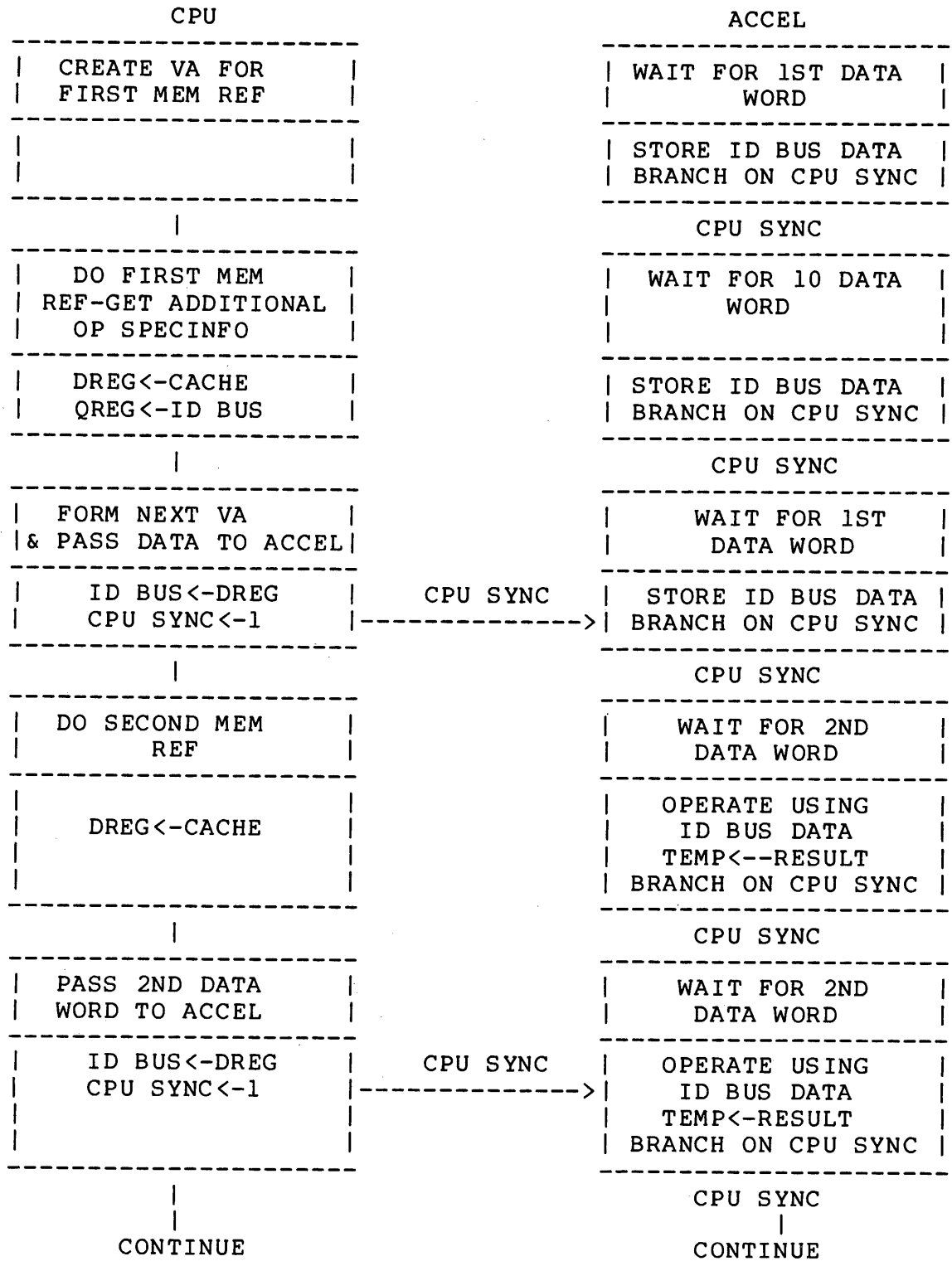

CPU
ACCEL
CREATE VA FOR FIRST MEM REF
WAIT FOR 1ST DATA WORD
STORE ID BUS DATA
BRANCH ON CPU SYNC
CPU SYNC
DO FIRST MEM REF-GET ADDITIONAL OP SPECINFO
WAIT FOR 10 DATA WORD
DREG<-CACHE
QREG<-ID BUS
STORE ID BUS DATA
BRANCH ON CPU SYNC
CPU SYNC
FORM NEXT VA & PASS DATA TO ACCEL
WAIT FOR 1ST DATA WORD
ID BUS<-DREG
CPU SYNC<-1
CPU SYNC
STORE ID BUS DATA
BRANCH ON CPU SYNC
CPU SYNC
DO SECOND MEM REF
WAIT FOR 2ND DATA WORD
DREG<-CACHE
OPERATE USING ID BUS DATA
TEMP<--RESULT
BRANCH ON CPU SYNC
CPU SYNC
PASS 2ND DATA WORD TO ACCEL
WAIT FOR 2ND DATA WORD
ID BUS<-DREG
CPU SYNC<-1
CPU SYNC
OPERATE USING ID BUS DATA
TEMP<-RESULT
BRANCH ON CPU SYNC
CPU SYNC
CONTINUE
CONTINUE

Figure 9-3

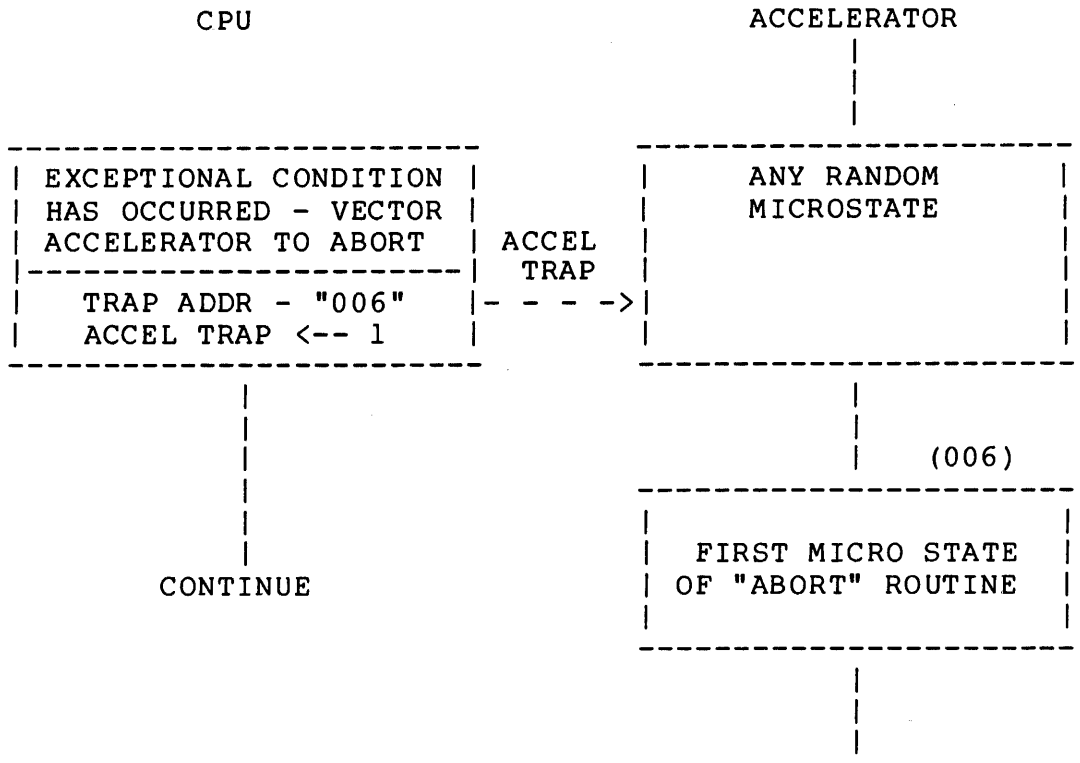


FIGURE 9-4

USE OF TRAP FUNCTION BY CPU TO UNCONDITIONALLY MODIFY ACCELERATOR CONTROL FLOW

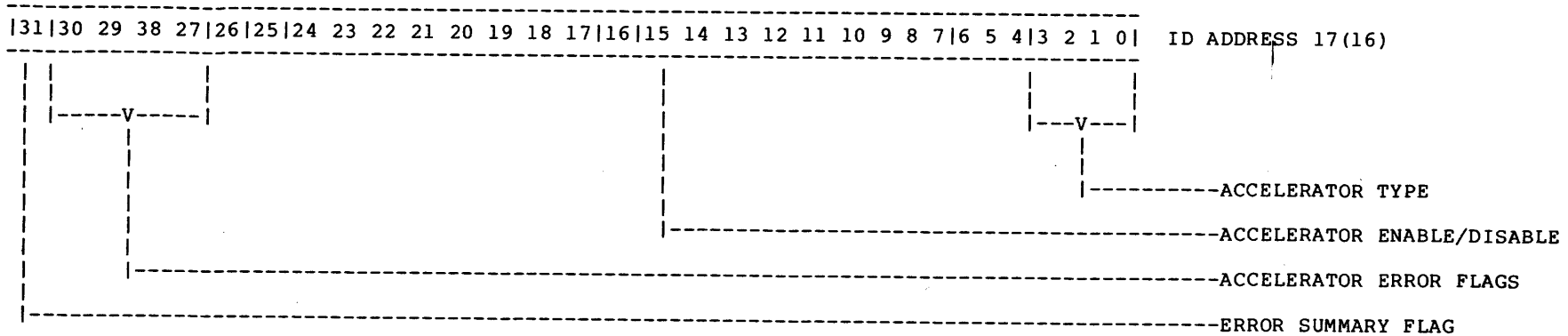


FIGURE 9-5

ACCELERATOR STATUS REGISTER

APPENDIX A

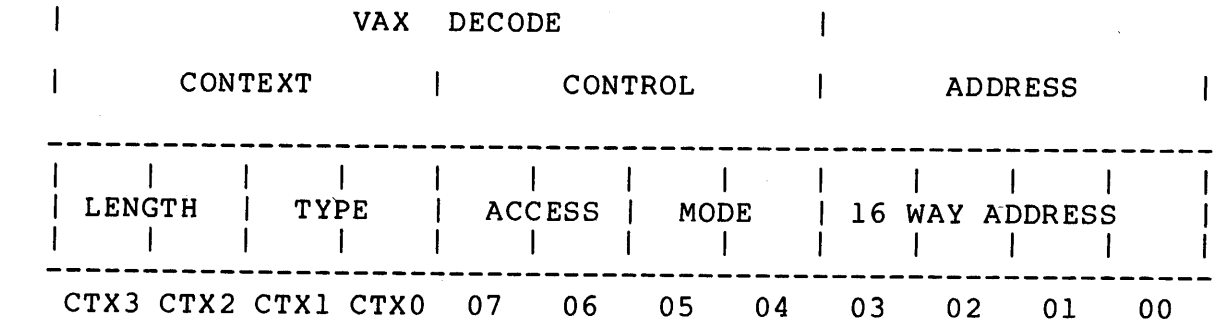
CONTROL WORD

THE SIX (6) 1K * 4 ROMS USED FOR VAX MODE ON THE IRC MODULE (M8224) MAKE UP A 12 BIT CONTROL WORD WHICH IS USED TO DETERMINE THE MAJOR EXECUTION POINTS OF THE INSTRUCTIONS.THE 12 BIT FIELD IS SPLIT INTO THREE (3),FOUR (4) BIT FIELDS WHICH ARE:

BITS	FIELD	DESCRIPTION
3:0	ADR	ADDRESS
7:4	CTL	CONTROL
11:8	CTX	CONTEXT

A.1 THE CONTROL WORD

THE CONTROL WORD IS SHOWN BELOW.



ADDRESS: THE ADDRESS FIELD IS USED TO PROVIDE A 16 WAY LITERAL DISPLACEMENT WHEN GENERATING AN EXECUTION ADDRESS.THE ADDRESS IS ONES COMPLEMENTED BEFORE GOING TO THE MICRO SEQUENCER.

EXAMPLE:IF BITS 03:00 = 5, THEN THE ADDRESS FIELD ON THE INPUT TO THE MICRO SEQUENCER WOULD BE AN "A".IN THIS CASE ALL EIGHT (8) VAX DECODE BITS ARE ONES COMPLEMENTED.

CONTROL: THE CONTROL FIELD IS SPLIT INTO TWO (2) ,TWO (2) BIT FIELDS WHICH ARE THE "MODE" FIELD (BITS 05:04) AND THE "ACCESS" FIELD (BITS 07:06).THESE FIELDS ARE DECODED AS FOLLOWS:

MODE:	05	04	OPERATION
	0	0	SELECT SPECIFIER
	0	1	EXECUTE IF R MODE (ONE OPERAND)
	1	0	OPTIMIZED (TWO OPERANDS)
	1	1	SELECT EXECUTE

ACCESS:	07	06	OPERATION
	0	0	BRANCH
	0	1	READ
	1	0	WRITE
	1	1	MODIFY

CONTEXT: THE CONTEXT FIELD IS SPLIT INTO TWO (2),TWO (2) BIT FIELDS WHICH ARE THE "TYPE" FIELD (BITS CTX1:CTX0) AND THE "LENGTH" FIELD (BITS CTX3:CTX2).THESE FIELDS ARE DECODED AS FOLLOWS:

TYPE:	CTX1	CTX0	OPERATION
	0	0	INTEGER
	0	1	FLOAT
	1	0	VSRC
	1	1	ASRC

LENGTH:	CTX3	CTX2	OPERATION
	0	0	BYTE
	0	1	WORD
	1	0	LONG
	1	1	QUAD

DEFINITIONS:

SELECT SPECIFIER: THIS CODE IS USED TO SELECT THE SPECIFIER DECODE LOGIC.THE EXECUTION ADDRESS WILL BE DETERMINED BY THE ADDRESSING MODE OF THE SPECIFIER.THE ADDRESSES THAT CAN BE GENERATED BY

THIS LOGIC ARE SHOWN IN THE SPECIFIER DECODE TABLE.

EXECUTE IF R MODE: THIS CODE IS USED TO CHECK IF THE SPECIFIER IN BYTE ONE IS REGISTER MODE.IF IT IS REGISTER MODE,THEN THE ONES COMPLEMENT OF THE VAX DECODE BITS 07:00 ARE USED AS THE EXECUTION ADDRESS.IF THIS CONDITION IS NOT MEET THEN THE SPECIFIER DECODE LOGIC IS SELECTED BY HARDWARE.

OPTIMIZED: THIS CODE CHECKS FOR SHORT LITERAL TO REGISTER (S^#,REG) OR REGISTER TO REGISTER (REG,REG). IF EITHER OF THESE CONDITIONS ARE MEET,THEN THE HARDWARE WILL MODIFY THE SPECIFIER ADDRESS. IF THESE CONDITIONS ARE NOT MEET,THEN THE SPECIFIER ADDRESS IS NOT MODIFIED.

SELECT EXECUTE: WHEN THIS CODE IS USED,THE VAX DECODE BITS 07:00 ARE ONES COMPLEMENTED AND USED AS AN EXECUTION ADDRESS.

BRANCH: THE BRANCH CODE WILL SELECT THE BRANCH DECODE LOGIC ONLY AT EXECUTION POINT 0. AT ANY OTHER EXECUTION POINT THIS CODE CAN ONLY BE USED AS A LITERAL FOR AN EXECUTION ADDRESS.

READ: THIS CODE IS USED WHEN A SPECIFIER EVALUATION WANTS TO READ DATA.THIS CODE CAN ALSO BE USED AS A LITERAL FOR AN EXECUTION ADDRESS WHEN THE SELECT EXECUTE MODE IS USED.

WRITE: THIS CODE IS USED WHEN A SPECIFIER EVALUATION WANTS TO WRITE DATA.THIS CODE CAN ALSO BE USED AS A LITERAL FOR AN EXECUTION ADDRESS WHEN THE SELECT EXECUTE MODE IS USED.

MODIFY: THIS CODE IS USED DURING A READ-MODIFY-WRITE. THIS INFORMS THE TRANSLATION BUFFER THAT THE OPERATION IS A READ WITH A WRITE CHECK.THIS CODE CAN ALSO BE USED AS A LITERAL FOR AN EXECUTION ADDRESS WHEN THE SELECT EXECUTE MODE IS USED.

INTEGER: THIS CODE IS USED WITH INTEGER DATA TYPES. REFER TO THE CONTEXT TABLE FOR COMMON USES OF THIS CODE.

FLOAT: THIS CODE IS USED WITH FLOATING DATA TYPES. REFER TO THE CONTEXT TABLE FOR COMMON USES OF THIS CODE.

VSRC: THIS CODE IS ONLY USED WITH FIELD INSTRUCTIONS. IT IS USED WHEN THE CALCULATED EFFECTIVE ADDRESS IS USED AS THE OPERAND.THIS INCLUDES REGISTER ADDRESSES.

ASRC: THIS CODE IS USED FOR INSTRUCTIONS THAT REQUIRE THE CALCULATED EFFECTIVE ADDRESS TO BE USED AS THE OPERAND.THIS EXCLUDES REGISTER ADDRESSES.

SPECIFIER DECODE TABLE

MODE	MNEMONIC	R/=PC	R=PC	QUAD	ABORT
0	S^#	00	00	02	01/03
1	S^#	00	00	02	01/03
2	S^#	00	00	02	01/03
3	S^#	00	00	02	01/03
4	E	0C	1C	--	1D
5	R	04	14	6/16	7/17,5/15
6	(R)	08	18	--	-----
7	-(R)	0A	1A	--	-----
8	(R)+	09	19	1F	-----
9	@(R)+	0B	1B	--	-----
A	D8	0D	0D	--	-----
B	@D8	0F	0F	--	-----
C	D16	0D	0D	--	-----
D	@D16	0F	0F	--	-----
E	D32	0D	0D	--	-----
F	@D32	0F	0F	--	-----

CONTEXT TABLE

LENGTH	TYPE	USE
BYTE	INTEGER	BYTE DATA TYPE
WORD	INTEGER	WORD DATA TYPE
LONG	INTEGER	LONGWORD DATA TYPE
QUAD	INTEGER	QUADWORD DATA TYPE
BYTE	FLOAT	UNDEFINED
WORD	FLOAT	UNDEFINED
LONG	FLOAT	FLOATING DATA TYPE
QUAD	FLOAT	DOUBLE-FLOATING DATA TYPE

A.2 ABORT CONDITION

ABORT CONDITIONS: THE FOLLOWING IS A LIST OF ABORT ADDRESSES AND THE CONDITIONS WHICH CAUSE THEM TO BE GENERATED.

ADDRESS	CONDITIONS
01	1. WRITING INTO A SHORT LITERAL. 2. E MODE FOLLOWED BY A SHORT LITERAL. 3. USING A SHORT LITERAL AS AN VSRC OR ASRC.
03	QUAD CONTEXT AND 1. WRITING INTO A SHORT LITERAL. 2. E MODE FOLLOWED BY A SHORT LITERAL. 3. USING A SHORT LITERAL AS AN VSRC OR ASRC.
05	1. USING REGISTER MODE FOR AN ASRC. 2. E MODE FOLLOWED BY REGISTER MODE.
07	CONTEXT TYPE IS QUAD AND 1. USING REGISTER MODE AS AN ASRC. 2. E MODE IS FOLLOWED BY REGISTER MODE.
14	1. REGISTER MODE AND RN EQUALS PC
15	RN EQUALS PC AND 1. USING REGISTER MODE AS AN ASRC. 2. E MODE IS FOLLOWED BY REGISTER MODE.
16	RN EQUALS PC WITH QUAD CONTEXT.
17	CONTEXT IS QUAD AND "RN" IS EQUAL TO PC 1. USING REGISTER MODE AS AN ASRC. 2. E MODE IS FOLLOWED BY REGISTER MODE.
18	THE ADDRESSING MODE IS REGISTER DEFFERED AND "RN" IS EQUAL TO PC.
1A	THE ADDRESSING MODE IS AUTO DECREMENT AND THE "RN" IS EQUAL TO THE PC.
1C	1. E MODE WITH THE "RN" EQUAL TO PC.
1D	1. E MODE FOLLOWED BY E MODE.

00 HALT

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	READ	EXECUTE	0
EP1	LONG	INT	READ	EXECUTE	8
EP2	LONG	INT	READ	EXECUTE	8
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

01 NOP

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	READ	EXECUTE	1
EP1	LONG	INT	READ	EXECUTE	8
EP2	LONG	INT	READ	EXECUTE	8
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

02 REI

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	READ	EXECUTE	2
EP1	LONG	INT	READ	EXECUTE	8
EP2	LONG	INT	READ	EXECUTE	8
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

03 BPT

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	READ	EXECUTE	3
EP1	LONG	INT	READ	EXECUTE	8
EP2	LONG	INT	READ	EXECUTE	8
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

04 RET

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	READ	EXECUTE	4
EP1	LONG	INT	READ	EXECUTE	8
EP2	LONG	INT	READ	EXECUTE	8
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

05 RSB

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	READ	EXECUTE	5
EP1	LONG	INT	READ	EXECUTE	8
EP2	LONG	INT	READ	EXECUTE	8
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

06 LDPCTX

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	READ	EXECUTE	6
EP1	LONG	INT	READ	EXECUTE	8
EP2	LONG	INT	READ	EXECUTE	8
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

07 SVPCTX

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	READ	EXECUTE	7
EP1	LONG	INT	READ	EXECUTE	8
EP2	LONG	INT	READ	EXECUTE	8
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

08 CVTPS

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	WORD	INT	READ	SELSPC	0
EP1	BYTE	ASRC	READ	SELSPC	0
EP2	LONG	INT	BRANCH	EXECUTE	F
EP3	LONG	INT	BRANCH	EXECUTE	A
EP4	WORD	INT	READ	SELSPC	0
EP5	BYTE	ASRC	READ	SELSPC	0
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	WRITE	EXECUTE	3

09 CVTSP

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	WORD	INT	READ	SELSPC	0
EP1	BYTE	ASRC	READ	SELSPC	0
EP2	LONG	INT	BRANCH	EXECUTE	F
EP3	LONG	INT	BRANCH	EXECUTE	E
EP4	WORD	INT	READ	SELSPC	0
EP5	BYTE	ASRC	READ	SELSPC	0
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	WRITE	EXECUTE	4

0A INDEX

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	READ	SELSPC	0
EP1	LONG	INT	READ	SELSPC	0
EP2	LONG	INT	BRANCH	EXECUTE	6
EP3	LONG	INT	READ	SELSPC	0
EP4	LONG	INT	READ	SELSPC	0
EP5	LONG	INT	READ	SELSPC	0
EP6	LONG	INT	WRITE	SELSPC	0
EP7	LONG	INT	READ	EXECUTE	8

0B CRC

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	BYTE	ASRC	READ	SELSPC	0
EP1	LONG	INT	READ	SELSPC	0
EP2	LONG	INT	BRANCH	EXECUTE	F
EP3	LONG	INT	READ	EXECUTE	4
EP4	WORD	INT	READ	SELSPC	0
EP5	BYTE	ASRC	READ	SELSPC	0
EP6	LONG	VSRC	READ	SELSPC	0
EP7	LONG	INT	WRITE	EXECUTE	E

0C PROBER

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	BYTE	INT	READ	SELSPC	0
EP1	WORD	INT	READ	SELSPC	0
EP2	LONG	INT	BRANCH	EXECUTE	1
EP3	BYTE	ASRC	READ	SELSPC	0
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

0D PROBEW

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	BYTE	INT	READ	SELSPC	0
EP1	WORD	INT	READ	SELSPC	0
EP2	LONG	INT	BRANCH	EXECUTE	1
EP3	BYTE	ASRC	READ	SELSPC	0
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

0E INSQUE

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	BYTE	ASRC	READ	SELSPC	0
EP1	BYTE	ASRC	READ	SELSPC	0
EP2	BYTE	ASRC	BRANCH	EXECUTE	0
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

0F REMQUE

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	BYTE	ASRC	READ	SELSPC	0
EP1	LONG	INT	READ	EXECUTE	B
EP2	LONG	INT	WRITE	SELSPC	0
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

10 BSBB

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	READ	EXECUTE	D
EP1	LONG	INT	READ	EXECUTE	8
EP2	LONG	INT	READ	EXECUTE	8
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

11 BRB

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	BRANCH	EXECUTE	0
EP1	LONG	INT	READ	EXECUTE	8
EP2	LONG	INT	READ	EXECUTE	8
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

12 BNEQ/BNEQU

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	BRANCH	EXECUTE	0
EP1	LONG	INT	READ	EXECUTE	8
EP2	LONG	INT	READ	EXECUTE	8
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

13 BEQL/BEQLU

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	BRANCH	EXECUTE	0
EP1	LONG	INT	READ	EXECUTE	8
EP2	LONG	INT	READ	EXECUTE	8
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

14 BGTR

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	BRANCH	EXECUTE	0
EP1	LONG	INT	READ	EXECUTE	8
EP2	LONG	INT	READ	EXECUTE	8
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

15 BLEQ

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	BRANCH	EXECUTE	0
EP1	LONG	INT	READ	EXECUTE	8
EP2	LONG	INT	READ	EXECUTE	8
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

16 JSB

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	BYTE	ASRC	READ	SELSPC	0
EP1	LONG	INT	READ	EXECUTE	0
EP2	LONG	INT	READ	EXECUTE	8
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

17 JMP

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	BYTE	ASRC	READ	SELSPC	0
EP1	LONG	INT	READ	EXECUTE	1
EP2	LONG	INT	READ	EXECUTE	8
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

18 BGEQ

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	BRANCH	EXECUTE	0
EP1	LONG	INT	READ	EXECUTE	8
EP2	LONG	INT	READ	EXECUTE	8
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

19 BLSS

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	BRANCH	EXECUTE	0
EP1	LONG	INT	READ	EXECUTE	8
EP2	LONG	INT	READ	EXECUTE	8
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

1A BGTRU

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	BRANCH	EXECUTE	0
EP1	LONG	INT	READ	EXECUTE	8
EP2	LONG	INT	READ	EXECUTE	8
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

1B BLEQU

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	BRANCH	EXECUTE	0
EP1	LONG	INT	READ	EXECUTE	8
EP2	LONG	INT	READ	EXECUTE	8
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

1C BVC

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	BRANCH	EXECUTE	0
EP1	LONG	INT	READ	EXECUTE	8
EP2	LONG	INT	READ	EXECUTE	8
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

1D BVS

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	BRANCH	EXECUTE	0
EP1	LONG	INT	READ	EXECUTE	8
EP2	LONG	INT	READ	EXECUTE	8
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

1E BGEQU/BCC

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	BRANCH	EXECUTE	0
EP1	LONG	INT	READ	EXECUTE	8
EP2	LONG	INT	READ	EXECUTE	8
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

1F BLSSU/BCS

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	BRANCH	EXECUTE	0
EP1	LONG	INT	READ	EXECUTE	8
EP2	LONG	INT	READ	EXECUTE	8
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

20 ADDP4

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	WORD	INT	READ	SELSPC	0
EP1	BYTE	ASRC	READ	SELSPC	0
EP2	LONG	INT	BRANCH	EXECUTE	5
EP3	WORD	INT	READ	SELSPC	0
EP4	BYTE	ASRC	READ	SELSPC	0
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	WRITE	EXECUTE	C

21 ADDP6

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	WORD	INT	READ	SELSPC	0
EP1	BYTE	ASRC	READ	SELSPC	0
EP2	LONG	INT	BRANCH	EXECUTE	5
EP3	WORD	INT	READ	SELSPC	0
EP4	BYTE	ASRC	READ	SELSPC	0
EP5	WORD	INT	READ	SELSPC	0
EP6	BYTE	ASRC	READ	SELSPC	0
EP7	LONG	INT	WRITE	EXECUTE	C

22 SUBP4

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	WORD	INT	READ	SELSPC	0
EP1	BYTE	ASRC	READ	SELSPC	0
EP2	LONG	INT	BRANCH	EXECUTE	5
EP3	WORD	INT	READ	SELSPC	0
EP4	BYTE	ASRC	READ	SELSPC	0
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	WRITE	EXECUTE	C

23 SUBP6

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	WORD	INT	READ	SELSPC	0
EP1	BYTE	ASRC	READ	SELSPC	0
EP2	LONG	INT	BRANCH	EXECUTE	5
EP3	WORD	INT	READ	SELSPC	0
EP4	BYTE	ASRC	READ	SELSPC	0
EP5	WORD	INT	READ	SELSPC	0
EP6	BYTE	ASRC	READ	SELSPC	0
EP7	LONG	INT	WRITE	EXECUTE	C

24 CVTPT

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	WORD	INT	READ	SELSPC	0
EP1	BYTE	ASRC	READ	SELSPC	0
EP2	LONG	INT	BRANCH	EXECUTE	F
EP3	LONG	INT	WRITE	EXECUTE	D
EP4	BYTE	ASRC	READ	SELSPC	0
EP5	WORD	INT	READ	SELSPC	0
EP6	BYTE	ASRC	READ	SELSPC	0
EP7	LONG	INT	WRITE	EXECUTE	3

25 MULP

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	WORD	INT	READ	SELSPC	0
EP1	BYTE	ASRC	READ	SELSPC	0
EP2	LONG	INT	BRANCH	EXECUTE	4
EP3	WORD	INT	READ	SELSPC	0
EP4	BYTE	ASRC	READ	SELSPC	0
EP5	WORD	INT	READ	SELSPC	0
EP6	BYTE	ASRC	READ	SELSPC	0
EP7	LONG	INT	WRITE	EXECUTE	8

26 CVTTP

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	WORD	INT	READ	SELSPC	0
EP1	BYTE	ASRC	READ	SELSPC	0
EP2	LONG	INT	BRANCH	EXECUTE	F
EP3	LONG	INT	WRITE	EXECUTE	C
EP4	BYTE	ASRC	READ	SELSPC	0
EP5	WORD	INT	READ	SELSPC	0
EP6	BYTE	ASRC	READ	SELSPC	0
EP7	LONG	INT	WRITE	EXECUTE	4

27 DIVP

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	WORD	INT	READ	SELSPC	0
EP1	BYTE	ASRC	READ	SELSPC	0
EP2	LONG	INT	BRANCH	EXECUTE	2
EP3	WORD	INT	READ	SELSPC	0
EP4	BYTE	ASRC	READ	SELSPC	0
EP5	WORD	INT	READ	SELSPC	0
EP6	BYTE	ASRC	READ	SELSPC	0
EP7	LONG	INT	WRITE	EXECUTE	8

28 MOVC3

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	WORD	INT	READ	SELSPC	0
EP1	BYTE	ASRC	READ	SELSPC	0
EP2	LONG	INT	BRANCH	EXECUTE	F
EP3	LONG	INT	WRITE	EXECUTE	5
EP4	BYTE	ASRC	READ	SELSPC	0
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	WRITE	EXECUTE	7

29 CMPC3

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	WORD	INT	READ	SELSPC	0
EP1	BYTE	ASRC	READ	SELSPC	0
EP2	LONG	INT	BRANCH	EXECUTE	F
EP3	LONG	INT	BRANCH	EXECUTE	D
EP4	BYTE	ASRC	READ	SELSPC	0
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	WRITE	EXECUTE	6

2A SCANC

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	WORD	INT	READ	SELSPC	0
EP1	BYTE	ASRC	READ	SELSPC	0
EP2	LONG	INT	BRANCH	EXECUTE	F
EP3	LONG	INT	READ	EXECUTE	9
EP4	BYTE	ASRC	READ	SELSPC	0
EP5	BYTE	INT	READ	SELSPC	0
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	WRITE	EXECUTE	5

2B SPANC

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	WORD	INT	READ	SELSPC	0
EP1	BYTE	ASRC	READ	SELSPC	0
EP2	LONG	INT	BRANCH	EXECUTE	F
EP3	LONG	INT	READ	EXECUTE	9
EP4	BYTE	ASRC	READ	SELSPC	0
EP5	BYTE	INT	READ	SELSPC	0
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	WRITE	EXECUTE	5

2C MOVC5

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	WORD	INT	READ	SELSPC	0
EP1	BYTE	ASRC	READ	SELSPC	0
EP2	LONG	INT	BRANCH	EXECUTE	F
EP3	LONG	INT	WRITE	EXECUTE	1
EP4	BYTE	INT	READ	SELSPC	0
EP5	WORD	INT	READ	SELSPC	0
EP6	BYTE	ASRC	READ	SELSPC	0
EP7	LONG	INT	WRITE	EXECUTE	7

2D CMPC5

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	WORD	INT	READ	SELSPC	0
EP1	BYTE	ASRC	READ	SELSPC	0
EP2	LONG	INT	BRANCH	EXECUTE	F
EP3	LONG	INT	READ	EXECUTE	C
EP4	BYTE	INT	READ	SELSPC	0
EP5	WORD	INT	READ	SELSPC	0
EP6	BYTE	ASRC	READ	SELSPC	0
EP7	LONG	INT	WRITE	EXECUTE	6

2E MOVTC

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	WORD	INT	READ	SELSPC	0
EP1	BYTE	ASRC	READ	SELSPC	0
EP2	LONG	INT	BRANCH	EXECUTE	C
EP3	BYTE	INT	READ	SELSPC	0
EP4	BYTE	ASRC	READ	SELSPC	0
EP5	WORD	INT	READ	SELSPC	0
EP6	BYTE	ASRC	READ	SELSPC	0
EP7	LONG	INT	WRITE	EXECUTE	7

2F MOVTUC

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	WORD	INT	READ	SELSPC	0
EP1	BYTE	ASRC	READ	SELSPC	0
EP2	LONG	INT	BRANCH	EXECUTE	C
EP3	BYTE	INT	READ	SELSPC	0
EP4	BYTE	ASRC	READ	SELSPC	0
EP5	WORD	INT	READ	SELSPC	0
EP6	BYTE	ASRC	READ	SELSPC	0
EP7	LONG	INT	WRITE	EXECUTE	7

30 BSBW

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	READ	EXECUTE	D
EP1	LONG	INT	READ	EXECUTE	8
EP2	LONG	INT	READ	EXECUTE	8
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

31 BRW

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	BRANCH	EXECUTE	0
EP1	LONG	INT	READ	EXECUTE	8
EP2	LONG	INT	READ	EXECUTE	8
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

32 CVTWL

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	WORD	INT	READ	SELSPC	0
EP1	WORD	INT	WRITE	EXECUTE	3
EP2	LONG	INT	WRITE	SELSPC	0
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

33 CVTWB

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	WORD	INT	READ	SELSPC	0
EP1	BYTE	INT	WRITE	EXECUTE	4
EP2	WORD	INT	READ	EXECUTE	9
EP3	BYTE	INT	WRITE	SELSPC	0
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

34 MOVP

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	WORD	INT	READ	SELSPC	0
EP1	BYTE	ASRC	READ	SELSPC	0
EP2	LONG	INT	BRANCH	EXECUTE	F
EP3	LONG	INT	READ	EXECUTE	B
EP4	BYTE	ASRC	READ	SELSPC	0
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	WRITE	EXECUTE	4

35 CMPP3

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	WORD	INT	READ	SELSPC	0
EP1	BYTE	ASRC	READ	SELSPC	0
EP2	LONG	INT	BRANCH	EXECUTE	F
EP3	LONG	INT	READ	EXECUTE	6
EP4	BYTE	ASRC	READ	SELSPC	0
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	WRITE	EXECUTE	A

36 CVTPL

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	WORD	INT	READ	SELSPC	0
EP1	BYTE	ASRC	READ	SELSPC	0
EP2	LONG	INT	WRITE	EXECUTE	F
EP3	LONG	INT	WRITE	SELSPC	0
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	WRITE	EXECUTE	3

37 CMPP4

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	WORD	INT	READ	SELSPC	0
EP1	BYTE	ASRC	READ	SELSPC	0
EP2	LONG	INT	BRANCH	EXECUTE	F
EP3	LONG	INT	READ	EXECUTE	A
EP4	WORD	INT	READ	SELSPC	0
EP5	BYTE	ASRC	READ	SELSPC	0
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	WRITE	EXECUTE	A

38 EDITPC

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	WORD	INT	READ	SELSPC	0
EP1	BYTE	ASRC	READ	SELSPC	0
EP2	LONG	INT	BRANCH	EXECUTE	9
EP3	BYTE	ASRC	READ	SELSPC	0
EP4	BYTE	ASRC	READ	SELSPC	0
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	WRITE	EXECUTE	E

39 MATCHC

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	WORD	INT	READ	SELSPC	0
EP1	BYTE	ASRC	READ	SELSPC	0
EP2	LONG	INT	BRANCH	EXECUTE	F
EP3	LONG	INT	READ	EXECUTE	5
EP4	WORD	INT	READ	SELSPC	0
EP5	BYTE	ASRC	READ	SELSPC	0
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	WRITE	EXECUTE	E

3A LOCC

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	BYTE	INT	READ	SELSPC	0
EP1	WORD	INT	READ	SELSPC	0
EP2	LONG	INT	BRANCH	EXECUTE	F
EP3	LONG	INT	READ	EXECUTE	7
EP4	BYTE	ASRC	READ	SELSPC	0
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	BRANCH	EXECUTE	E

3B SKPC

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	BYTE	INT	READ	SELSPC	0
EP1	WORD	INT	READ	SELSPC	0
EP2	LONG	INT	BRANCH	EXECUTE	F
EP3	LONG	INT	READ	EXECUTE	7
EP4	BYTE	ASRC	READ	SELSPC	0
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	BRANCH	EXECUTE	E

3C MOVZWL

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	WORD	INT	READ	SELSPC	0
EP1	WORD	INT	WRITE	EXECUTE	5
EP2	LONG	INT	WRITE	SELSPC	0
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

3D ACBW

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	WORD	INT	READ	SELSPC	0
EP1	WORD	INT	READ	SELSPC	0
EP2	WORD	INT	BRANCH	EXECUTE	B
EP3	WORD	INT	MODIFY	SELSPC	0
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

3E MOVAW

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	WORD	ASRC	READ	SELSPC	0
EP1	LONG	INT	WRITE	SELSPC	0
EP2	LONG	INT	READ	EXECUTE	8
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

3F PUSHAW

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	WORD	ASRC	READ	SELSPC	0
EP1	LONG	INT	WRITE	EXECUTE	7
EP2	LONG	INT	READ	EXECUTE	8
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

40 ADDF2

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	FLOAT	READ	OPT	0
EP1	LONG	FLOAT	MODIFY	EXEC/R	0
EP2	LONG	FLOAT	READ	EXECUTE	1
EP3	LONG	FLOAT	WRITE	EXECUTE	E
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

41 ADDF3

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	FLOAT	READ	SELSPC	0
EP1	LONG	FLOAT	READ	OPT	0
EP2	LONG	FLOAT	READ	EXECUTE	1
EP3	LONG	FLOAT	WRITE	SELSPC	0
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

42 SUBF2

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	FLOAT	READ	OPT	0
EP1	LONG	FLOAT	MODIFY	EXEC/R	0
EP2	LONG	FLOAT	READ	EXECUTE	1
EP3	LONG	FLOAT	WRITE	EXECUTE	E
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

43 SUBF3

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	FLOAT	READ	SELSPC	0
EP1	LONG	FLOAT	READ	OPT	0
EP2	LONG	FLOAT	READ	EXECUTE	1
EP3	LONG	FLOAT	WRITE	SELSPC	0
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

44 MULF2

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	FLOAT	READ	SELSPC	0
EP1	LONG	FLOAT	MODIFY	EXEC/R	F
EP2	LONG	FLOAT	READ	EXECUTE	0
EP3	LONG	FLOAT	WRITE	EXECUTE	E
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

45 MULF3

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	FLOAT	READ	SELSPC	0
EP1	LONG	FLOAT	READ	SELSPC	0
EP2	LONG	FLOAT	READ	EXECUTE	0
EP3	LONG	FLOAT	WRITE	SELSPC	0
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

46 DIVF2

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	FLOAT	READ	SELSPC	0
EP1	LONG	FLOAT	MODIFY	EXEC/R	E
EP2	LONG	FLOAT	READ	EXECUTE	2
EP3	LONG	FLOAT	WRITE	EXECUTE	E
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

47 DIVF3

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	FLOAT	READ	SELSPC	0
EP1	LONG	FLOAT	READ	SELSPC	0
EP2	LONG	FLOAT	READ	EXECUTE	2
EP3	LONG	FLOAT	WRITE	SELSPC	0
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

48 CVTFB

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	FLOAT	READ	SELSPC	0
EP1	BYTE	INT	WRITE	EXECUTE	A
EP2	BYTE	INT	WRITE	SELSPC	0
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

49 CVTFW

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	FLOAT	READ	SELSPC	0
EP1	WORD	INT	WRITE	EXECUTE	A
EP2	WORD	INT	WRITE	SELSPC	0
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

4A CVTFL

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	FLOAT	READ	SELSPC	0
EP1	LONG	INT	WRITE	EXECUTE	A
EP2	LONG	INT	WRITE	SELSPC	0
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

4B CVTRFL

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	FLOAT	READ	SELSPC	0
EP1	LONG	INT	BRANCH	EXECUTE	E
EP2	LONG	INT	WRITE	SELSPC	0
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

4C CVTBF

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	BYTE	INT	READ	SELSPC	0
EP1	BYTE	INT	WRITE	EXECUTE	0
EP2	LONG	FLOAT	WRITE	SELSPC	0
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

4D CVTWF

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	WORD	INT	READ	SELSPC	0
EP1	WORD	INT	WRITE	EXECUTE	0
EP2	LONG	FLOAT	WRITE	SELSPC	0
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

4E CVTLF

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	READ	SELSPC	0
EP1	LONG	INT	WRITE	EXECUTE	0
EP2	LONG	FLOAT	WRITE	SELSPC	0
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

4F ACBF

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	FLOAT	READ	SELSPC	0
EP1	LONG	FLOAT	READ	SELSPC	0
EP2	LONG	FLOAT	BRANCH	EXECUTE	A
EP3	LONG	FLOAT	MODIFY	SELSPC	0
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

50 MOVF

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	FLOAT	READ	SELSPC	0
EP1	LONG	FLOAT	READ	EXECUTE	A
EP2	LONG	FLOAT	WRITE	SELSPC	0
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

51 CMPF

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	FLOAT	READ	SELSPC	0
EP1	LONG	FLOAT	READ	SELSPC	0
EP2	LONG	FLOAT	BRANCH	EXECUTE	E
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

52 MNEGF

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	FLOAT	READ	SELSPC	0
EP1	LONG	FLOAT	READ	EXECUTE	A
EP2	LONG	FLOAT	WRITE	SELSPC	0
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

53 TSTF

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	FLOAT	READ	EXEC/R	6
EP1	LONG	FLOAT	READ	EXECUTE	2
EP2	LONG	INT	READ	EXECUTE	8
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

54 EMODF

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	FLOAT	READ	SELSPC	0
EP1	BYTE	INT	READ	SELSPC	0
EP2	LONG	INT	BRANCH	EXECUTE	7
EP3	LONG	FLOAT	READ	SELSPC	0
EP4	LONG	INT	WRITE	SELSPC	0
EP5	LONG	FLOAT	WRITE	SELSPC	0
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

55 POLYF

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	FLOAT	READ	SELSPC	0
EP1	WORD	INT	READ	SELSPC	0
EP2	LONG	INT	BRANCH	EXECUTE	D
EP3	BYTE	ASRC	READ	SELSPC	0
EP4	LONG	FLOAT	WRITE	EXECUTE	7
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	FLOAT	BRANCH	EXECUTE	D

56 CVTFD

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	FLOAT	READ	SELSPC	0
EP1	QUAD	FLOAT	WRITE	EXECUTE	8
EP2	QUAD	FLOAT	WRITE	SELSPC	0
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

57 RESERVED

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	READ	EXECUTE	8
EP1	LONG	INT	READ	EXECUTE	8
EP2	LONG	INT	READ	EXECUTE	8
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

58 ADAWI

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	WORD	INT	READ	SELSPC	0
EP1	LONG	INT	MODIFY	EXECUTE	1
EP2	WORD	VSRC	READ	SELSPC	0
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

59 RESERVED

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	READ	EXECUTE	8
EP1	LONG	INT	READ	EXECUTE	8
EP2	LONG	INT	READ	EXECUTE	8
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

5A RESERVED

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	READ	EXECUTE	8
EP1	LONG	INT	READ	EXECUTE	8
EP2	LONG	INT	READ	EXECUTE	8
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

5B RESERVED

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	READ	EXECUTE	8
EP1	LONG	INT	READ	EXECUTE	8
EP2	LONG	INT	READ	EXECUTE	8
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

5C RESERVED

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	READ	EXECUTE	8
EP1	LONG	INT	READ	EXECUTE	8
EP2	LONG	INT	READ	EXECUTE	8
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

5D RESERVED

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	READ	EXECUTE	8
EP1	LONG	INT	READ	EXECUTE	8
EP2	LONG	INT	READ	EXECUTE	8
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

5E RESERVED

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	READ	EXECUTE	8
EP1	LONG	INT	READ	EXECUTE	8
EP2	LONG	INT	READ	EXECUTE	8
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

5F RESERVED

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	READ	EXECUTE	8
EP1	LONG	INT	READ	EXECUTE	8
EP2	LONG	INT	READ	EXECUTE	8
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

60 ADDD2

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	QUAD	FLOAT	READ	OPT	0
EP1	QUAD	FLOAT	MODIFY	EXEC/R	1
EP2	QUAD	FLOAT	READ	EXECUTE	5
EP3	QUAD	FLOAT	WRITE	EXECUTE	B
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

61 ADDD3

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	QUAD	FLOAT	READ	SELSPC	0
EP1	QUAD	FLOAT	READ	OPT	0
EP2	QUAD	FLOAT	READ	EXECUTE	5
EP3	QUAD	FLOAT	WRITE	SELSPC	0
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

62 SUBD2

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	QUAD	FLOAT	READ	OPT	0
EP1	QUAD	FLOAT	MODIFY	EXEC/R	1
EP2	QUAD	FLOAT	READ	EXECUTE	5
EP3	QUAD	FLOAT	WRITE	EXECUTE	B
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

63 SUBD3

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	QUAD	FLOAT	READ	SELSPC	0
EP1	QUAD	FLOAT	READ	OPT	0
EP2	QUAD	FLOAT	READ	EXECUTE	5
EP3	QUAD	FLOAT	WRITE	SELSPC	0
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

64 MULD2

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	QUAD	FLOAT	READ	SELSPC	0
EP1	QUAD	FLOAT	MODIFY	EXEC/R	7
EP2	QUAD	FLOAT	READ	EXECUTE	3
EP3	QUAD	FLOAT	WRITE	EXECUTE	B
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

65 MULD3

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	QUAD	FLOAT	READ	SELSPC	0
EP1	QUAD	FLOAT	READ	SELSPC	0
EP2	QUAD	FLOAT	READ	EXECUTE	3
EP3	QUAD	FLOAT	WRITE	SELSPC	0
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

66 DIVD2

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	QUAD	FLOAT	READ	SELSPC	0
EP1	QUAD	FLOAT	MODIFY	EXEC/R	6
EP2	QUAD	FLOAT	READ	EXECUTE	4
EP3	QUAD	FLOAT	WRITE	EXECUTE	B
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

67 DIVD3

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	QUAD	FLOAT	READ	SELSPC	0
EP1	QUAD	FLOAT	READ	SELSPC	0
EP2	QUAD	FLOAT	READ	EXECUTE	4
EP3	QUAD	FLOAT	WRITE	SELSPC	0
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

68 CVTDB

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	QUAD	FLOAT	READ	SELSPC	0
EP1	BYTE	INT	WRITE	EXECUTE	C
EP2	BYTE	INT	WRITE	SELSPC	0
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

69 CVTDW

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	QUAD	FLOAT	READ	SELSPC	0
EP1	WORD	INT	WRITE	EXECUTE	C
EP2	WORD	INT	WRITE	SELSPC	0
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

6A CVTDL

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	QUAD	FLOAT	READ	SELSPC	0
EP1	LONG	INT	WRITE	EXECUTE	C
EP2	LONG	INT	WRITE	SELSPC	0
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

6B CVTRDT

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	QUAD	FLOAT	READ	SELSPC	0
EP1	LONG	INT	BRANCH	EXECUTE	A
EP2	LONG	INT	WRITE	SELSPC	0
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

6C CVTBD

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	BYTE	INT	READ	SELSPC	0
EP1	BYTE	INT	BRANCH	EXECUTE	3
EP2	QUAD	FLOAT	WRITE	SELSPC	0
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

6D CVTWD

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	WORD	INT	READ	SELSPC	0
EP1	WORD	INT	BRANCH	EXECUTE	3
EP2	QUAD	FLOAT	WRITE	SELSPC	0
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

6E CVTLD

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	READ	SELSPC	0
EP1	LONG	INT	BRANCH	EXECUTE	3
EP2	QUAD	FLOAT	WRITE	SELSPC	0
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

6F ACBD

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	QUAD	FLOAT	READ	SELSPC	0
EP1	QUAD	FLOAT	READ	SELSPC	0
EP2	QUAD	FLOAT	READ	EXECUTE	B
EP3	QUAD	FLOAT	MODIFY	SELSPC	0
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

70 MOVD

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	QUAD	FLOAT	READ	SELSPC	0
EP1	QUAD	FLOAT	WRITE	EXECUTE	E
EP2	QUAD	FLOAT	WRITE	SELSPC	0
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

71 CMPD

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	QUAD	FLOAT	READ	SELSPC	0
EP1	QUAD	FLOAT	READ	SELSPC	0
EP2	QUAD	FLOAT	READ	EXECUTE	D
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

72 MNEGD

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	QUAD	FLOAT	READ	SELSPC	0
EP1	QUAD	FLOAT	WRITE	EXECUTE	E
EP2	QUAD	FLOAT	WRITE	SELSPC	0
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

73 TSTD

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	QUAD	FLOAT	READ	EXEC/R	6
EP1	QUAD	FLOAT	READ	EXECUTE	3
EP2	LONG	INT	READ	EXECUTE	8
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

74 EMODD

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	QUAD	FLOAT	READ	SELSPC	0
EP1	BYTE	INT	READ	SELSPC	0
EP2	BYTE	INT	READ	EXECUTE	9
EP3	QUAD	FLOAT	READ	SELSPC	0
EP4	LONG	INT	WRITE	SELSPC	0
EP5	QUAD	FLOAT	WRITE	SELSPC	0
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

75 POLYD

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	QUAD	FLOAT	READ	SELSPC	0
EP1	WORD	INT	READ	SELSPC	0
EP2	LONG	INT	READ	EXECUTE	A
EP3	BYTE	ASRC	READ	SELSPC	0
EP4	QUAD	FLOAT	WRITE	EXECUTE	7
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	QUAD	FLOAT	BRANCH	EXECUTE	C

76 CVTDF

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	QUAD	FLOAT	READ	SELSPC	0
EP1	LONG	FLOAT	BRANCH	EXECUTE	2
EP2	LONG	FLOAT	WRITE	SELSPC	0
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

77 RESERVED

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	READ	EXECUTE	8
EP1	LONG	INT	READ	EXECUTE	8
EP2	LONG	INT	READ	EXECUTE	8
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

78 ASHL

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	BYTE	INT	READ	SELSPC	0
EP1	LONG	INT	READ	SELSPC	0
EP2	LONG	INT	WRITE	EXECUTE	3
EP3	LONG	INT	WRITE	SELSPC	0
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

79 ASHQ

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	BYTE	INT	READ	SELSPC	0
EP1	QUAD	INT	READ	SELSPC	0
EP2	QUAD	INT	WRITE	EXECUTE	C
EP3	QUAD	INT	WRITE	SELSPC	0
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

7A EMUL

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	READ	SELSPC	0
EP1	LONG	INT	READ	SELSPC	0
EP2	LONG	INT	READ	EXECUTE	6
EP3	LONG	INT	READ	SELSPC	0
EP4	QUAD	INT	WRITE	SELSPC	0
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

7B EDIV

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	READ	SELSPC	0
EP1	QUAD	INT	READ	SELSPC	0
EP2	QUAD	INT	READ	EXECUTE	7
EP3	LONG	VSRC	READ	SELSPC	0
EP4	LONG	VSRC	READ	SELSPC	0
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

7C CLRQ

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	QUAD	INT	WRITE	EXECUTE	2
EP1	QUAD	INT	WRITE	SELSPC	0
EP2	LONG	INT	READ	EXECUTE	8
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

7D MOVQ

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	QUAD	INT	READ	SELSPC	0
EP1	QUAD	INT	WRITE	SELSPC	0
EP2	LONG	INT	READ	EXECUTE	8
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

7E MOVAQ/MOVAD

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	QUAD	ASRC	READ	SELSPC	0
EP1	LONG	INT	WRITE	SELSPC	0
EP2	LONG	INT	READ	EXECUTE	8
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

7F PUSHAQ/PUSHAD

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	QUAD	ASRC	READ	SELSPC	0
EP1	LONG	INT	WRITE	EXECUTE	7
EP2	LONG	INT	READ	EXECUTE	8
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

80 ADDB2

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	BYTE	INT	READ	OPT	0
EP1	BYTE	INT	MODIFY	EXEC/R	8
EP2	BYTE	INT	WRITE	EXECUTE	4
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

81 ADDB3

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	BYTE	INT	READ	SELSPC	0
EP1	BYTE	INT	READ	OPT	0
EP2	BYTE	INT	WRITE	EXECUTE	5
EP3	BYTE	INT	WRITE	SELSPC	0
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

82 SUBB2

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	BYTE	INT	READ	OPT	0
EP1	BYTE	INT	MODIFY	EXEC/R	8
EP2	BYTE	INT	WRITE	EXECUTE	4
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

83 SUBB3

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	BYTE	INT	READ	SELSPC	0
EP1	BYTE	INT	READ	OPT	0
EP2	BYTE	INT	WRITE	EXECUTE	5
EP3	BYTE	INT	WRITE	SELSPC	0
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

84 MULB2

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	BYTE	INT	READ	SELSPC	0
EP1	BYTE	INT	MODIFY	EXEC/R	2
EP2	BYTE	INT	READ	EXECUTE	E
EP3	BYTE	INT	WRITE	EXECUTE	E
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

85 MULB3

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	BYTE	INT	READ	SELSPC	0
EP1	BYTE	INT	READ	SELSPC	0
EP2	BYTE	INT	READ	EXECUTE	E
EP3	BYTE	INT	WRITE	SELSPC	0
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

86 DIVB2

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	BYTE	INT	READ	SELSPC	0
EP1	BYTE	INT	MODIFY	EXEC/R	3
EP2	BYTE	INT	READ	EXECUTE	F
EP3	BYTE	INT	WRITE	EXECUTE	E
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

87 DIVB3

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	BYTE	INT	READ	SELSPC	0
EP1	BYTE	INT	READ	SELSPC	0
EP2	BYTE	INT	READ	EXECUTE	F
EP3	BYTE	INT	WRITE	SELSPC	0
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

88 BISB2

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	BYTE	INT	READ	OPT	0
EP1	BYTE	INT	MODIFY	EXEC/R	8
EP2	BYTE	INT	WRITE	EXECUTE	4
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

89 BISB3

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	BYTE	INT	READ	SELSPC	0
EP1	BYTE	INT	READ	OPT	0
EP2	BYTE	INT	WRITE	EXECUTE	5
EP3	BYTE	INT	WRITE	SELSPC	0
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

8A BICB2

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	BYTE	INT	READ	OPT	0
EP1	BYTE	INT	MODIFY	EXEC/R	8
EP2	BYTE	INT	WRITE	EXECUTE	4
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

8B BICB3

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	BYTE	INT	READ	SELSPC	0
EP1	BYTE	INT	READ	OPT	0
EP2	BYTE	INT	WRITE	EXECUTE	5
EP3	BYTE	INT	WRITE	SELSPC	0
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

8C XORB2

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	BYTE	INT	READ	OPT	0
EP1	BYTE	INT	MODIFY	EXEC/R	8
EP2	BYTE	INT	WRITE	EXECUTE	4
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

8D XORB3

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	BYTE	INT	READ	SELSPC	0
EP1	BYTE	INT	READ	OPT	0
EP2	BYTE	INT	WRITE	EXECUTE	5
EP3	BYTE	INT	WRITE	SELSPC	0
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

8E MNEGB

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	BYTE	INT	READ	OPT	0
EP1	BYTE	INT	WRITE	EXECUTE	6
EP2	BYTE	INT	WRITE	SELSPC	0
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

8F CASEB

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	BYTE	INT	READ	SELSPC	0
EP1	BYTE	INT	READ	SELSPC	0
EP2	BYTE	INT	READ	EXECUTE	C
EP3	BYTE	INT	READ	SELSPC	0
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

90 MOVB

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	BYTE	INT	READ	OPT	0
EP1	BYTE	INT	WRITE	SELSPC	0
EP2	LONG	INT	READ	EXECUTE	8
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

91 CMPB

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	BYTE	INT	READ	OPT	0
EP1	BYTE	INT	READ	EXEC/R	0
EP2	BYTE	INT	WRITE	EXECUTE	6
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

92 MCOMB

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	BYTE	INT	READ	OPT	0
EP1	BYTE	INT	WRITE	EXECUTE	6
EP2	BYTE	INT	WRITE	SELSPC	0
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

93 BITB

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	BYTE	INT	READ	OPT	0
EP1	BYTE	INT	READ	EXEC/R	0
EP2	BYTE	INT	WRITE	EXECUTE	6
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

94 CLRB

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	BYTE	INT	WRITE	EXECUTE	0
EP1	BYTE	INT	WRITE	SELSPC	0
EP2	LONG	INT	READ	EXECUTE	8
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

95 TSTB

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	BYTE	INT	READ	EXEC/R	0
EP1	BYTE	INT	WRITE	EXECUTE	2
EP2	LONG	INT	READ	EXECUTE	8
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

96 INCB

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	BYTE	INT	MODIFY	EXEC/R	0
EP1	BYTE	INT	WRITE	EXECUTE	1
EP2	LONG	INT	READ	EXECUTE	8
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

97 DECB

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	BYTE	INT	MODIFY	EXEC/R	0
EP1	BYTE	INT	WRITE	EXECUTE	1
EP2	LONG	INT	READ	EXECUTE	8
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

98 CVTBL

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	BYTE	INT	READ	SELSPC	0
EP1	BYTE	INT	WRITE	EXECUTE	3
EP2	LONG	INT	WRITE	SELSPC	0
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

99 CVTBW

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	BYTE	INT	READ	SELSPC	0
EP1	BYTE	INT	WRITE	EXECUTE	3
EP2	WORD	INT	WRITE	SELSPC	0
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

9A MOVZBL

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	BYTE	INT	READ	SELSPC	0
EP1	BYTE	INT	WRITE	EXECUTE	5
EP2	LONG	INT	WRITE	SELSPC	0
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

9B MOVZBW

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	BYTE	INT	READ	SELSPC	0
EP1	BYTE	INT	WRITE	EXECUTE	5
EP2	WORD	INT	WRITE	SELSPC	0
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

9C ROTL

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	BYTE	INT	READ	SELSPC	0
EP1	LONG	INT	READ	SELSPC	0
EP2	LONG	INT	WRITE	EXECUTE	2
EP3	LONG	INT	WRITE	SELSPC	0
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

9D ACBB

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	BYTE	INT	READ	SELSPC	0
EP1	BYTE	INT	READ	SELSPC	0
EP2	BYTE	INT	BRANCH	EXECUTE	B
EP3	BYTE	INT	MODIFY	SELSPC	0
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

9E MOVAB

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	BYTE	ASRC	READ	SELSPC	0
EP1	LONG	INT	WRITE	SELSPC	0
EP2	LONG	INT	READ	EXECUTE	8
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

9F PUSHAB

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	BYTE	ASRC	READ	SELSPC	0
EP1	LONG	INT	WRITE	EXECUTE	7
EP2	LONG	INT	READ	EXECUTE	8
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

A0 ADDW2

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	WORD	INT	READ	OPT	0
EP1	WORD	INT	MODIFY	EXEC/R	8
EP2	WORD	INT	WRITE	EXECUTE	4
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

A1 ADDW3

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	WORD	INT	READ	SELSPC	0
EP1	WORD	INT	READ	OPT	0
EP2	WORD	INT	WRITE	EXECUTE	5
EP3	WORD	INT	WRITE	SELSPC	0
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

A2 SUBW2

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	WORD	INT	READ	OPT	0
EP1	WORD	INT	MODIFY	EXEC/R	8
EP2	WORD	INT	WRITE	EXECUTE	4
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

A3 SUBW3

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	WORD	INT	READ	SELSPC	0
EP1	WORD	INT	READ	OPT	0
EP2	WORD	INT	WRITE	EXECUTE	5
EP3	WORD	INT	WRITE	SELSPC	0
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

A4 MULW2

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	WORD	INT	READ	SELSPC	0
EP1	WORD	INT	MODIFY	EXEC/R	2
EP2	WORD	INT	READ	EXECUTE	E
EP3	WORD	INT	WRITE	EXECUTE	E
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

A5 MULW3

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	WORD	INT	READ	SELSPC	0
EP1	WORD	INT	READ	SELSPC	0
EP2	WORD	INT	READ	EXECUTE	E
EP3	WORD	INT	WRITE	SELSPC	0
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

A6 DIVW2

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	WORD	INT	READ	SELSPC	0
EP1	WORD	INT	MODIFY	EXEC/R	3
EP2	WORD	INT	READ	EXECUTE	F
EP3	WORD	INT	WRITE	EXECUTE	E
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

A7 DIVW3

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	WORD	INT	READ	SELSPC	0
EP1	WORD	INT	READ	SELSPC	0
EP2	WORD	INT	READ	EXECUTE	F
EP3	WORD	INT	WRITE	SELSPC	0
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

A8 BISW2

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	WORD	INT	READ	OPT	0
EP1	WORD	INT	MODIFY	EXEC/R	8
EP2	WORD	INT	WRITE	EXECUTE	4
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

A9 BISW3

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	WORD	INT	READ	SELSPC	0
EP1	WORD	INT	READ	OPT	0
EP2	WORD	INT	WRITE	EXECUTE	5
EP3	WORD	INT	WRITE	SELSPC	0
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

AA BICW2

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	WORD	INT	READ	OPT	0
EP1	WORD	INT	MODIFY	EXEC/R	8
EP2	WORD	INT	WRITE	EXECUTE	4
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

AB BICW3

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	WORD	INT	READ	SELSPC	0
EP1	WORD	INT	READ	OPT	0
EP2	WORD	INT	WRITE	EXECUTE	5
EP3	WORD	INT	WRITE	SELSPC	0
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

AC XORW2

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	WORD	INT	READ	OPT	0
EP1	WORD	INT	MODIFY	EXEC/R	8
EP2	WORD	INT	WRITE	EXECUTE	4
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

AD XORW3

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	WORD	INT	READ	SELSPC	0
EP1	WORD	INT	READ	OPT	0
EP2	WORD	INT	WRITE	EXECUTE	5
EP3	WORD	INT	WRITE	SELSPC	0
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

AE MNEGW

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	WORD	INT	READ	OPT	0
EP1	WORD	INT	WRITE	EXECUTE	6
EP2	WORD	INT	WRITE	SELSPC	0
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

AF CASEW

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	WORD	INT	READ	SELSPC	0
EP1	WORD	INT	READ	SELSPC	0
EP2	WORD	INT	READ	EXECUTE	C
EP3	WORD	INT	READ	SELSPC	0
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

B0 MOVW

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	WORD	INT	READ	OPT	0
EP1	WORD	INT	WRITE	SELSPC	0
EP2	LONG	INT	READ	EXECUTE	8
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

B1 CMPW

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	WORD	INT	READ	OPT	0
EP1	WORD	INT	READ	EXEC/R	0
EP2	WORD	INT	WRITE	EXECUTE	6
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

B2 MCOMW

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	WORD	INT	READ	OPT	0
EP1	WORD	INT	WRITE	EXECUTE	6
EP2	WORD	INT	WRITE	SELSPC	0
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

B3 BITW

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	WORD	INT	READ	OPT	0
EP1	WORD	INT	READ	EXEC/R	0
EP2	WORD	INT	WRITE	EXECUTE	6
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

B4 CLRW

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	WORD	INT	WRITE	EXECUTE	0
EP1	WORD	INT	WRITE	SELSPC	0
EP2	LONG	INT	READ	EXECUTE	8
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

B5 TSTW

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	WORD	INT	READ	EXEC/R	0
EP1	WORD	INT	WRITE	EXECUTE	2
EP2	LONG	INT	READ	EXECUTE	8
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

B6 INCW

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	WORD	INT	MODIFY	EXEC/R	0
EP1	WORD	INT	WRITE	EXECUTE	1
EP2	LONG	INT	READ	EXECUTE	8
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

B7 DECW

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	WORD	INT	MODIFY	EXEC/R	0
EP1	WORD	INT	WRITE	EXECUTE	1
EP2	LONG	INT	READ	EXECUTE	8
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

B8 BISPSW

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	WORD	INT	READ	SELSPC	0
EP1	WORD	INT	READ	EXECUTE	4
EP2	LONG	INT	READ	EXECUTE	8
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

B9 BICPSW

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	WORD	INT	READ	SELSPC	0
EP1	WORD	INT	READ	EXECUTE	4
EP2	LONG	INT	READ	EXECUTE	8
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

BA POPR

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	WORD	INT	READ	SELSPC	0
EP1	WORD	INT	READ	EXECUTE	5
EP2	LONG	INT	READ	EXECUTE	8
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

BB PUSHR

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	WORD	INT	READ	SELSPC	0
EP1	WORD	INT	READ	EXECUTE	6
EP2	LONG	INT	READ	EXECUTE	8
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

BC CHMK

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	WORD	INT	READ	SELSPC	0
EP1	WORD	INT	READ	EXECUTE	7
EP2	LONG	INT	READ	EXECUTE	8
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

BD CHME

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	WORD	INT	READ	SELSPC	0
EP1	WORD	INT	READ	EXECUTE	7
EP2	LONG	INT	READ	EXECUTE	8
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

BE CHMS

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	WORD	INT	READ	SELSPC	0
EP1	WORD	INT	READ	EXECUTE	7
EP2	LONG	INT	READ	EXECUTE	8
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

BF CHMU

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	WORD	INT	READ	SELSPC	0
EP1	WORD	INT	READ	EXECUTE	7
EP2	LONG	INT	READ	EXECUTE	8
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

C0 ADDL2

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	READ	OPT	0
EP1	LONG	INT	MODIFY	EXEC/R	8
EP2	LONG	INT	WRITE	EXECUTE	4
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

C1 ADDL3

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	READ	SELSPC	0
EP1	LONG	INT	READ	OPT	0
EP2	LONG	INT	WRITE	EXECUTE	5
EP3	LONG	INT	WRITE	SELSPC	0
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

C2 SUBL2

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	READ	OPT	0
EP1	LONG	INT	MODIFY	EXEC/R	8
EP2	LONG	INT	WRITE	EXECUTE	4
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

C3 SUBL3

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	READ	SELSPC	0
EP1	LONG	INT	READ	OPT	0
EP2	LONG	INT	WRITE	EXECUTE	5
EP3	LONG	INT	WRITE	SELSPC	0
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

C4 MULL2

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	READ	SELSPC	0
EP1	LONG	INT	MODIFY	EXEC/R	2
EP2	LONG	INT	READ	EXECUTE	E
EP3	LONG	INT	WRITE	EXECUTE	E
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

C5 MULL3

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	READ	SELSPC	0
EP1	LONG	INT	READ	SELSPC	0
EP2	LONG	INT	READ	EXECUTE	E
EP3	LONG	INT	WRITE	SELSPC	0
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

C6 DIVL2

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	READ	SELSPC	0
EP1	LONG	INT	MODIFY	EXEC/R	3
EP2	LONG	INT	READ	EXECUTE	F
EP3	LONG	INT	WRITE	EXECUTE	E
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

C7 DIVL3

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	READ	SELSPC	0
EP1	LONG	INT	READ	SELSPC	0
EP2	LONG	INT	READ	EXECUTE	F
EP3	LONG	INT	WRITE	SELSPC	0
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

C8 BISL2

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	READ	OPT	0
EP1	LONG	INT	MODIFY	EXEC/R	8
EP2	LONG	INT	WRITE	EXECUTE	4
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

C9 BISL3

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	READ	SELSPC	0
EP1	LONG	INT	READ	OPT	0
EP2	LONG	INT	WRITE	EXECUTE	5
EP3	LONG	INT	WRITE	SELSPC	0
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

CA BICL2

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	READ	OPT	0
EP1	LONG	INT	MODIFY	EXEC/R	8
EP2	LONG	INT	WRITE	EXECUTE	4
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

CB BICL3

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	READ	SELSPC	0
EP1	LONG	INT	READ	OPT	0
EP2	LONG	INT	WRITE	EXECUTE	5
EP3	LONG	INT	WRITE	SELSPC	0
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

CC XORL2

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	READ	OPT	0
EP1	LONG	INT	MODIFY	EXEC/R	8
EP2	LONG	INT	WRITE	EXECUTE	4
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

CD XORL3

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	READ	SELSPC	0
EP1	LONG	INT	READ	OPT	0
EP2	LONG	INT	WRITE	EXECUTE	5
EP3	LONG	INT	WRITE	SELSPC	0
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

CE MNEGL

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	READ	OPT	0
EP1	LONG	INT	WRITE	EXECUTE	6
EP2	LONG	INT	WRITE	SELSPC	0
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

CF CASEL

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	READ	SELSPC	0
EP1	LONG	INT	READ	SELSPC	0
EP2	LONG	INT	MODIFY	EXECUTE	1
EP3	LONG	INT	READ	SELSPC	0
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

D0 MOVL

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	READ	OPT	0
EP1	LONG	INT	WRITE	SELSPC	0
EP2	LONG	INT	READ	EXECUTE	8
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

D1 CMPL

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	READ	OPT	0
EP1	LONG	INT	READ	EXEC/R	0
EP2	LONG	INT	WRITE	EXECUTE	6
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

D2 MCOML

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	READ	OPT	0
EP1	LONG	INT	WRITE	EXECUTE	6
EP2	LONG	INT	WRITE	SELSPC	0
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

D3 BITL

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	READ	OPT	0
EP1	LONG	INT	READ	EXEC/R	0
EP2	LONG	INT	WRITE	EXECUTE	6
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

D4 CLRL

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	WRITE	EXECUTE	0
EP1	LONG	INT	WRITE	SELSPC	0
EP2	LONG	INT	READ	EXECUTE	8
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

D5 TSTL

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	READ	EXEC/R	0
EP1	LONG	INT	WRITE	EXECUTE	2
EP2	LONG	INT	READ	EXECUTE	8
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

D6 INCL

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	MODIFY	EXEC/R	0
EP1	LONG	INT	WRITE	EXECUTE	1
EP2	LONG	INT	READ	EXECUTE	8
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

D7 DECL

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	MODIFY	EXEC/R	0
EP1	LONG	INT	WRITE	EXECUTE	1
EP2	LONG	INT	READ	EXECUTE	8
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

D8 ADWC

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	READ	OPT	0
EP1	LONG	INT	MODIFY	EXEC/R	8
EP2	LONG	INT	WRITE	EXECUTE	4
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

D9 SBWC

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	READ	OPT	0
EP1	LONG	INT	MODIFY	EXEC/R	8
EP2	LONG	INT	WRITE	EXECUTE	4
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

DA MTPR

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	READ	SELSPC	0
EP1	LONG	INT	READ	SELSPC	0
EP2	LONG	INT	WRITE	EXECUTE	8
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

DB MFPR

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	READ	SELSPC	0
EP1	LONG	INT	BRANCH	EXECUTE	8
EP2	LONG	INT	WRITE	SELSPC	0
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

DC MOVPSL

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	WRITE	EXECUTE	1
EP1	LONG	INT	WRITE	SELSPC	0
EP2	LONG	INT	READ	EXECUTE	8
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

DD PUSHL

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	READ	SELSPC	0
EP1	LONG	INT	WRITE	EXECUTE	7
EP2	LONG	INT	READ	EXECUTE	8
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

DE MOVAL/MOVAF

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	ASRC	READ	SELSPC	0
EP1	LONG	INT	WRITE	SELSPC	0
EP2	LONG	INT	READ	EXECUTE	8
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

DF PUSHAL/PUSHAF

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	ASRC	READ	SELSPC	0
EP1	LONG	INT	WRITE	EXECUTE	7
EP2	LONG	INT	READ	EXECUTE	8
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

E0 BBS

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	READ	SELSPC	0
EP1	BYTE	VSRC	READ	EXEC/R	7
EP2	BYTE	VSRC	WRITE	EXECUTE	A
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

E1 BBC

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	READ	SELSPC	0
EP1	BYTE	VSRC	READ	EXEC/R	7
EP2	BYTE	VSRC	WRITE	EXECUTE	A
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

E2 BBSS

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	READ	SELSPC	0
EP1	BYTE	VSRC	READ	EXEC/R	7
EP2	BYTE	VSRC	WRITE	EXECUTE	A
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

E3 BBCS

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	READ	SELSPC	0
EP1	BYTE	VSRC	READ	EXEC/R	7
EP2	BYTE	VSRC	WRITE	EXECUTE	A
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

E4 BBSC

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	READ	SELSPC	0
EP1	BYTE	VSRC	READ	EXEC/R	7
EP2	BYTE	VSRC	WRITE	EXECUTE	A
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

E5 BBCC

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	READ	SELSPC	0
EP1	BYTE	VSRC	READ	EXEC/R	7
EP2	BYTE	VSRC	WRITE	EXECUTE	A
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

E6 BBSSI

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	READ	SELSPC	0
EP1	BYTE	VSRC	READ	EXEC/R	7
EP2	BYTE	VSRC	WRITE	EXECUTE	A
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

E7 BBCCI

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	READ	SELSPC	0
EP1	BYTE	VSRC	READ	EXEC/R	7
EP2	BYTE	VSRC	WRITE	EXECUTE	A
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

E8 BLBS

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	READ	EXEC/R	1
EP1	BYTE	INT	BRANCH	EXECUTE	0
EP2	LONG	INT	READ	EXECUTE	8
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

E9 BLBC

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	READ	EXEC/R	1
EP1	BYTE	INT	BRANCH	EXECUTE	0
EP2	LONG	INT	READ	EXECUTE	8
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

EA FFS

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	READ	SELSPC	0
EP1	BYTE	INT	READ	SELSPC	0
EP2	LONG	INT	WRITE	EXECUTE	1
EP3	BYTE	VSRC	READ	SELSPC	0
EP4	LONG	INT	READ	EXECUTE	F
EP5	LONG	INT	WRITE	SELSPC	0
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

EB FFC

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	READ	SELSPC	0
EP1	BYTE	INT	READ	SELSPC	0
EP2	LONG	INT	WRITE	EXECUTE	1
EP3	BYTE	VSRC	READ	SELSPC	0
EP4	LONG	INT	READ	EXECUTE	E
EP5	LONG	INT	WRITE	SELSPC	0
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

EC CMPV

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	READ	SELSPC	0
EP1	BYTE	INT	READ	SELSPC	0
EP2	LONG	INT	WRITE	EXECUTE	1
EP3	BYTE	VSRC	READ	SELSPC	0
EP4	LONG	INT	READ	EXECUTE	D
EP5	LONG	INT	READ	SELSPC	0
EP6	LONG	INT	WRITE	EXECUTE	6
EP7	LONG	INT	READ	EXECUTE	8

ED CMPZV

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	READ	SELSPC	0
EP1	BYTE	INT	READ	SELSPC	0
EP2	LONG	INT	WRITE	EXECUTE	1
EP3	BYTE	VSRC	READ	SELSPC	0
EP4	LONG	INT	READ	EXECUTE	1
EP5	LONG	INT	READ	SELSPC	0
EP6	LONG	INT	WRITE	EXECUTE	6
EP7	LONG	INT	READ	EXECUTE	8

EE EXTV

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	READ	SELSPC	0
EP1	BYTE	INT	READ	SELSPC	0
EP2	LONG	INT	WRITE	EXECUTE	1
EP3	BYTE	VSRC	READ	SELSPC	0
EP4	LONG	INT	READ	EXECUTE	D
EP5	LONG	INT	WRITE	SELSPC	0
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

EF EXTZV

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	READ	SELSPC	0
EP1	BYTE	INT	READ	SELSPC	0
EP2	LONG	INT	WRITE	EXECUTE	1
EP3	BYTE	VSRC	READ	SELSPC	0
EP4	LONG	INT	READ	EXECUTE	1
EP5	LONG	INT	WRITE	SELSPC	0
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

F0 INSV

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	READ	SELSPC	0
EP1	LONG	INT	READ	SELSPC	0
EP2	LONG	INT	WRITE	EXECUTE	7
EP3	BYTE	INT	READ	SELSPC	0
EP4	BYTE	VSRC	READ	SELSPC	0
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

F1 ACBL

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	READ	SELSPC	0
EP1	LONG	INT	READ	SELSPC	0
EP2	LONG	INT	BRANCH	EXECUTE	B
EP3	LONG	INT	MODIFY	SELSPC	0
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

F2 AOBLSS

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	READ	SELSPC	0
EP1	LONG	INT	MODIFY	EXEC/R	C
EP2	LONG	INT	WRITE	EXECUTE	9
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

F3 AOBLEQ

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	READ	SELSPC	0
EP1	LONG	INT	MODIFY	EXEC/R	C
EP2	LONG	INT	WRITE	EXECUTE	9
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

F4 SOBGEQ

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	MODIFY	EXEC/R	1
EP1	BYTE	INT	BRANCH	EXECUTE	1
EP2	LONG	INT	READ	EXECUTE	8
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

F5 SOBGTR

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	MODIFY	EXEC/R	1
EP1	BYTE	INT	BRANCH	EXECUTE	1
EP2	LONG	INT	READ	EXECUTE	8
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

F6 CVTLB

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	READ	SELSPC	0
EP1	BYTE	INT	WRITE	EXECUTE	4
EP2	LONG	INT	READ	EXECUTE	9
EP3	BYTE	INT	WRITE	SELSPC	0
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

F7 CVTLW

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	READ	SELSPC	0
EP1	WORD	INT	WRITE	EXECUTE	4
EP2	LONG	INT	READ	EXECUTE	9
EP3	WORD	INT	WRITE	SELSPC	0
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

F8 ASHP

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	BYTE	INT	READ	SELSPC	0
EP1	WORD	INT	READ	SELSPC	0
EP2	LONG	INT	BRANCH	EXECUTE	3
EP3	BYTE	ASRC	READ	SELSPC	0
EP4	BYTE	INT	READ	SELSPC	0
EP5	WORD	INT	READ	SELSPC	0
EP6	BYTE	ASRC	READ	SELSPC	0
EP7	LONG	INT	BRANCH	EXECUTE	A

F9 CVTLP

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	READ	SELSPC	0
EP1	WORD	INT	READ	SELSPC	0
EP2	LONG	INT	WRITE	EXECUTE	D
EP3	BYTE	ASRC	READ	SELSPC	0
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	WRITE	EXECUTE	3

FA CALLG

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	BYTE	ASRC	READ	SELSPC	0
EP1	BYTE	ASRC	READ	SELSPC	0
EP2	BYTE	ASRC	WRITE	EXECUTE	0
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

FB CALLS

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	READ	SELSPC	0
EP1	BYTE	ASRC	READ	SELSPC	0
EP2	BYTE	ASRC	WRITE	EXECUTE	0
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

FC XFC

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	READ	EXECUTE	9
EP1	LONG	INT	READ	EXECUTE	8
EP2	LONG	INT	READ	EXECUTE	8
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

FD ESCD

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	READ	EXECUTE	A
EP1	LONG	INT	READ	EXECUTE	8
EP2	LONG	INT	READ	EXECUTE	8
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

FE ESCE

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	READ	EXECUTE	E
EP1	LONG	INT	READ	EXECUTE	8
EP2	LONG	INT	READ	EXECUTE	8
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

FF ESCF

EXC PT	LENGTH	TYPE	ACCESS	MODE	ADDRESS
EP0	LONG	INT	READ	EXECUTE	C
EP1	LONG	INT	READ	EXECUTE	8
EP2	LONG	INT	READ	EXECUTE	8
EP3	LONG	INT	READ	EXECUTE	8
EP4	LONG	INT	READ	EXECUTE	8
EP5	LONG	INT	READ	EXECUTE	8
EP6	LONG	INT	READ	EXECUTE	8
EP7	LONG	INT	READ	EXECUTE	8

APPENDIX B

WRITABLE CONTROL STORE

B.1 WRITEABLE CONTROL STORE MEMORY

The writable control store memory consists of 1024 words. Each word contains 96 data bits, of which 3 parity bits (1 per each 32 bits of data). The Control Store (CS) bus outputs are tri-state.

TTL HIGH = "1"
TTL LOW = "0" The normal parity generated is even.

B.2 WRITE DATA TO WCS

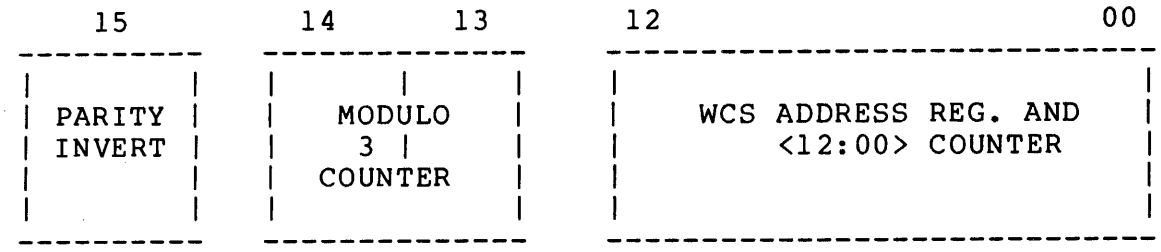
The write sequence will require two machine cycles. The first being the writing of the WCS address register from the ID bus. The second being the writing of the WCS data to WCS memory from the ID bus. The write WCS address cycle and the WCS write data cycle may either be consecutive cycles or the address may be loaded and sometime later the data may be loaded.

The same 1K module of WCS MUST NOT be the control store in use by the CPU and have a WCS write data operation being performed on it during the same machine cycle. The data written into WCS is in true form. Writing WCS under console control is done by writing the WCS address register and then doing a WCS write data operation for each 32 bits of data to be written.

The WCS address counter bits <14:13> (on the micro sequencer) are loaded with a WCS write address command. These stages are a modulo-3 counter that selects the 32 bit data group to be written. At the end of each WCS write data command the counter is incremented, when the modulo-3 counter overflows; the WCS address register bits <12:00> are incremented by one.

B.3 WCS ADDRESS REGISTER

Layout



BIT 15 = 1 = INVERT PARITY
= 0 = NORMAL PARITY

BIT<14:13> = 00 = GROUP A = DATA BITS<31:00>
= 01 = GROUP B = DATA BITS<63:32>
= 10 GROUP C = DATA BITS<95:64>
= 00 GROUP A

B.4 EXTERNAL JUMPER SELECTIONS AND RAM TYPE SELECTION

There are five back panel pins to be used with jumpers or switches for selecting the one of eight 1K segments to be assigned the WCS module. The voltage level will be at '0' volts with jumper installed and at '+3' volts with the jumper removed. The jumper (J5) when out selects the lower 4K area, and when in selects the upper 4K area. The jumpers J1, J2, J3, J4 are to select a WCS module for a particular 1K area from a possible of four 1K areas.

J1	J2	J3	J4	MEM BANK SELECTED
OUT	IN	IN	IN	1ST 1K SEGMENT
X	OUT	IN	IN	2ND 1K SEGMENT
X	X	OUT	IN	3RD 1K SEGMENT
X	X	X	OUT	4TH 1K SEGMENT

X = DON'T CARE CONDITION

There is a provision for reading the summary of available writeable control store segments by doing an ID bus read of the WCS memory data register. When this command is received by the micro sequencer control, a signal is generated which tells all WCS modules to transmit bits <07:00> onto the ID bus. ID bus bit 00 represents the 1st 1K of available WCS and bit 07 represents the 8th 1K of available WCS memory segments.

APPENDIX C

MICRO-CODE DEBUGGER INTERFACE

C.1 OBJECTIVES

Develop a tool needed to debug micro-programs, both in ROM & WCS. Features required and specified here-in are:

- Entry & Exit, Micro-code Debugger
- Micro-Machine State control
- Examine & Deposit to Internal Registers
- Examine Control Storage Location
- Deposit to WCS Location

Also covered are the expected usage restrictions.

> "The prime objective of the micro-code debugger is to allow the user to micro-step his micro-program and to examine the micro-machines internal registers without destroying the micro-machine's state".

C.2 MICRO-CODE DEBUGGER ENTRY & EXIT

The Micro-Code Debugger may be entered from the console program by WCS. The console program is re-entered upon exit from the Micro-code debugger.

The transfer between the micro-code debugger and the console program, on both entry and exit, can occur without loss of micro-machine state if the micro-machine's clock is running or stopped.

On both entry and exit, the program (LSI-11) entered is in its initialized state.

Specific features provided are:

1. Load Micro-Code Debugger

   If the micro-machine is running or stopped, allow the Micro-code debugger program to be loaded such that the micro-machine can be continued correctly upon the debugger assuming control of the micro-machine.

2. Load Console Program

   If the micro-machine is running or stopped, allow the console program to be loaded such that the micro-machine can be continued correctly upon the console program assuming control of the micro-machine.

C.3 MICRO-CODE DEBUGGER- MICRO-MACHINE- STATE CONTROL

On entry to the Debugger from the Console program, the micro-machine can be either running or stopped in any TIME STATE (T0-T3). (time state readable over V-Bus by console)

If it is stopped the micro-code debugger will advance the clock using single state step IMO "T3" and will put the micro-machine into the micro-step mode.

On stopping the micro-machine (or on initial entry if stopped), the micro-code debugger will examine the Machine State and the next micro-instruction to be executed to determine if it is dependent upon the current micro-machine state; if so, special action is required by the micro-code debugger (later referred to as a dependent micro-instruction):

"STALL" Asserted

If "stall" is asserted, the micro-debugger will "single time state" the micro-machine until "stall" is de-asserted during T3 by the micro-machine. (if not de-asserted after 600 micro-machine cycles, Error Message is printed).

"Dependent Micro-Instruction"

The micro-code debugger will examine the next micro-instruction to be executed to determine if it is a dependent one. This is done by comparing each of the micro-orders in the micro-instruction against a list of dependent micro-orders. If any exist, a warning message is printed on the console, not to use the micro-debugger's examine & deposit features.

Specific features are:

1. Enter Micro-Step Mode

   The debugger will be entered into the Micro-Step Mode if the Micro-Machine is stopped.

2. Micro-Step

   If the micro-machine is stopped, the micro-machine will execute one micro-instruction and then stop during T3 prior to starting execution of the next micro-instruction.

   Also single state stepping of the micro-machines clock is required through the micro-instruction for several reasons, i.e., setup time required for "slow constants".

3. Leave Micro-Step Mode

   The debugger will leave Micro-Step Mode & the micro-machine will start running at normal clock speed at the address in the UPCSV on the next "proceed" command.

4. Set Micro-Break (SOMM)

   If the micro-machine is stopped, the "SOMM" bit will be set, and On, "PROCEED". The micro-machine will be stopped after executing the micro-instruction whose address is specified in ID[21].

   This feature uses the micro-machine's stop-on-micro-break match feature to stop itself during T0 of the addressed micro-instruction. Then the micro-debugger will advance execution of the current instruction into "T3" state.

   The micro-machine will remain stopped until it gets a "proceed" command.

5. STOP

   If the clock is running, this feature will stop it and advance it into "T3" of the next micro-instruction.

In addition the micro-code debugger will implement a number of switches, etc. to allow micro-breaking on a list of micro-addresses using the micro-step feature, i.e., Proceed/B.

C.4 MICRO-CODE DEBUGGER INTERNAL REGISTER & MEMORY EXAMINE & DEPOSIT

The following internal registers and memory may be examined and/or modified using this feature. Since the micro-state of the micro-machine may be modified by the supporting micro-routines in the micro-machine, the micro-code debugger prints a warning message when this will occur. (specified earlier in the spec.)

The user can generally get around this problem by entering the machine into micro-step mode, and micro-stepping the micro-machine until a warning message is not printed.

The micro-machine facilities that can be examined and or modified are:

IBA, Q, VA, MICRO-PC, D, SC, LA, LB, STATE LC, FE, PC, RLOG & PCSV (R.O.), RA, RC, ID, & MEMORY.

The micro-code debugger completes execution of the current micro-instruction by advancing the clock using single state step into T0 and applying "ROM NOP" to stop starting execution of the next micro-instruciton.

Then micro-code debugger saves and restores the various registers that can be indirectly changed by the supporting micro-routines so that micro-machine can be successfully restarted (if a warning message was not printed) on the next micro-instruction to be executed.

On debugger initiated memory references, the user must clear ID-Registers, TBER1, SBI.ERR, & PARITY if a memory exception occurs and he wants to continue his micro-program.

Deposits & Examines to memory during memory management firmware will destroy the micro-machine state (no warning printed).

Examine Control Store Location

Since the control store cannot be read by the debugger, the micro-code debugger examines a load image file that has been opened on a floppy disk and prints the contents of the field(s) requested. (warning messages can be ignored)

Deposit To WCS Location

Deposits to the field(s) specified in WCS and on the load image floppy file.

Previous warning messages must be observed.

C.5 LIST OF DEPENDENT MICRO-ORDERS

	PRINT WARNING MESSAGE*	
	BEFORE	AFTER
ACF/SYNC	*	*
BEN/ALU 1-0.NEQ.ZERO	*	
BEN/TB.TEST	*	
FS/=0&MCT/(ALL CODES EXCEPT "ALLOW.IB.READ)		*
FS/=0&MCT/LOCKREAD.V.NOCHK	*	
LOCKREAD.V.WCHK	*	
SBI.HOLD	*	
SBI.HOLD+UNJAM	*	
LOCKREAD.P	*	
SUB/SPEC		*

Sequences that must be avoided using micro-step:

1. Interlocked read/writes.

2. SBI UNJAM sequence.

3. I/O programs.

4. CS parity error.

5. Memory Examine/Deposit during memory management F/W.

On sequences 1), 2), & 5) above, Micro-break match can be set to a micro-instruction following the sequence, then clearing "step" and then doing the "Proceed" command will execute the sequence at machine speed, allowing the micro-programmer to skip over these problems.

---

* Before or after execution of next micro-instruction.

Future Possible Enhancements

1. Reducing restrictions on MCT/FIELD micro-orders.

   By reading over the V-Bus, determine if micro-trap is pending or not. If not, do not print warning message (associated one).

2. Examining ID Bus Registers without using Micro-Machine clock.

   When the machine is stopped during T3, allow examining of ID-REGS.

   If not in T3, use micro-code routine to do the examine after asking the user if he wants to destroy the machine state.

3. Examine/Deposit after Warning Message.

   Ask user if he really wants to destroy state. If response is "YES", do requested Examine or Deposit.

APPENDIX D

WCS DEBUGGER HELP FILE

The WCS Debugger help file may be accessed by typing at the console:

```
>>>@WCSMON.HLP
```

To call the WCS debugger, type at the console:

```
>>>WCS
```

```
MICRO-DEBUGGER HELP FILE          REV-0   MAY 1977
TO STOP PRINTING, TYPE C

DEBUGGER COMMANDS(ALL TERMINATED BY CARRIAGE RETURN)

 'E/P <ADDRESS>'                  -EXAMINE PHYSICAL MEMORY
 'E/ID <ADDRESS>'                 -EXAMINE ID BUS REGISTER

 'E <ADDRESS>'                    -EXAMINE WCS LOCATION, DISPLAY ALL FIELDS
 'E <ADDRESS> <FIELDNAME-1>,<FIELDNAME-2>,,,,<FIELDNAME-N>
                        EXAMINE WCS LOCATION, DISPLAY ONLY FIELDS
                        THE FIELDS SPECIFIED
        NOTE: <FIELDNAMES> =    ACF,ACM,ADS,ALU,BEN,BMX,CCK,CID,DK,DT,EAL
                                EBM,FEK,FS,IBC,IEK,UJM,KMX,MCT,MSC,PCK,QK
                                RMX,SCK,SGN,SHF,SI,SMX,SPO,USU,VAK

 'E RA <ADDRESS>'                 -EXAMINE AN RA REGISTER
 'E RC <ADDRESS>'                 -EXAMINE AN RC REGISTER
```

'E <SYMBOLIC-NAME>' -EXAMINE ONE OF THE SYMBOLICALLY NAMED REGISTERS

NOTE: <SYMBOLIC-NAMES> = DR,FER,IBA,LA,LB,LC,Q,RL,SC,SR,UPC

'D/P <ADDRESS> <DATA>' -DEPOSIT <DATA> TO PHYSICAL MEMORY

'D/ID <ADDRESS> <DATA>' -DEPOSIT <DATA> TO ID BUS REGISTER

'D <ADDRESS> <FIELDNAME-1> <DATA-1>,<FIELDNAME-2> <DATA-2>,.....

-DEPOSIT TO WCS LOCATION, PUTTING <DATA-1> INTO <FIELDNAME-1>, ETC. UNSPECIFIED FIELDS ARE UNCHANGED.

NOTE: THE '/Z' QUALIFIER MAY BE USED TO CAUSE ALL UNSPECIFIED FIELDS TO BE CLARED.

'D RA <ADDRESS> <DATA>' -DEPOSIT <DATA> TO AN RA REGISTER

'D RC <ADDRESS> <DATA>' -DEPOSIT <DATA> TO AN RC REGISTER

'D <SYMBOLIC-NAME> DATA>' -DEPOSIT <DATA> TO ONE OF THE SYMBOLICALLY NAMED REGISTERS (SEE LIST ABOVE).

NOTE: DEPOSITS TO THE RLOG STACK(RL) ARE NOT SUPPORTED.

'CONTINUE' -RESUME MICRO-INSTRUCTION EXECUTION AS SPECIFIED BY CONTENTS OF MICRO-PC(UPC)

'START <ADDRESS>' -START MICRO-SEQUENCER AT <ADDRESS>.

'HALT' -HALT THE MICRO-SEQUENCER

'SET SOMM' -SET THE 'STOP ON MICRO-MATCH' ENABLE

'CLEAR SOMM' -CLEAR THE 'STOP ON MICRO-MATCH' ENABLE

'SET STEP' -ENABLE SINGLE MICRO-INSTRUCTION STEP MODE. START OR CONTINUE WILL ALLOW ONE MICRO-INSTRUCTION TO EXECUTE, THEN HALT THE MICRO-SEQUENCER.

'CLEAR STEP' -DISABLE SINGLE MICRO-INSTRUCTION STEP MODE.

'RETURN' -RETURN TO THE CONSOLE PROGRAM

'OPEN <FILENAME>' -OPEN SPECIFIED FILE ON FLOPPY DRIVE 0

'OPEN DX1:<FILENAME>' -OPEN SPECIFIED FILE ON FLOPPY DRIVE 1

NOTE: 'OPEN' IS USED TO SPECIFY A FILE CONTAINING THE MICRO-CODE CURRENTLY LOADED IN THE WCS PORTION OF THE CONTROL STORE. (ADDRESSES 1000(16) & UP IN THE CONTROL STORE) THIS FILE WILL BE USED FOR ALL EXAMINES OF THE WCS, SINCE THE WCS IS NOT DIRECTLY READABLE.

APPENDIX E

PROM CONTROL STORE SPECIFICATION

The control store (CS) bus outputs are tri-state

TTL HIGH = "1"
TTL LOW = "0"

CS bus output loading by other subsystems

A maximum of 2 loads/bit, external to the control store and the micro sequencer modules will be allowed. User receivers are to be within 6 inches of the module fingers.

E.1 PROM ADDRESS PATH

The PROM address is selected by Micro Program Counter (UPC) bits <09:00>. UPC Bit <12> selects either the lower 4K bank or the higher 4K bank of PROM. UPC Bits <11:10> select one of the four 1K segments to be accessed.

E.2 PARITY ERROR DETECTION

The parity tree is 99 bits wide and is done in two levels. The parity checking is for 96 data bits and 3 parity bits. Each parity bit makes up even parity for 32 consecutive data bits. That is, there will be an even number of 1's in the 33 bit field (32 data and 1 parity). For example:

BIT	31	30	29	----------	3	2	1	0	PARITY	
	1	0	1	<--------0-------->	1	0	0	0	1	EVEN
	1	0	0	<--------0-------->	1	0	0	0	0	EVEN

If the micro program tried to access non-existant control store memory, then NO CS bus drivers would be enabled. This would cause an all 1's condition including parity on the CS bus due to the terminator pull-up resistor. The parity error detection logic will see this as odd parity and flag a micro word parity error, then resulting in a micro trap.

E.3 EXTERNAL JUMPER SELECTIONS AND CS TYPE SELECTION

There are five back panel pins to be used with jumpers or switches for board and segment selection. The jumper (J5) when in selects the lower 4K bank, and when out selects the upper 4K bank. The jumpers J1, J2, J3, J4 each select a 1K segment when removed. When a jumper is in, that particular 1K segment is disabled.

J1 selects or disables the 1st 1K segment
J2 selects or disables the 2nd 1K segment
J3 selects or disables the 3rd 1K segment
J4 selects or disables the 4th 1K segment

INDEX

VAX 11/780 DATA PATH
DESCRIPTION
AA-H307A-TE

READER'S COMMENTS

NOTE: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. If you require a written reply and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

Did you find errors in this manual? If so, specify the error and the page number.

Please indicate the type of reader that you most nearly represent.

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Other (please specify)_____

Name_____ Date_____

Organization_____

Street_____

City_____ State_____ Zip Code_____
or

Please cut along this line.

---------- Fold Here ----------

---------- Do Not Tear - Fold Here and Staple ----------

FIRST CLASS
PERMIT NO. 33
MAYNARD, MASS.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

Postage will be paid by:

digital

MICROWARE GROUP ML3-5/E32
146 MAIN STREET