2. Central Processor

The Central Processor (CP) emulates the Mesa Processor as defined by the Mesa Processor Principles of Operation, and provides ALU service for the integral I/O controllers. The central processor is modeled almost exactly after the architecture of the Dandelion, but executes a slightly different version of the Dandelion microinstruction set. In addition, a small part of the Dandelion Mesa Emulator has been modified.

Figure 1.1 in Section 1 illustrates the relationship of the Mesa Processor Board (MPB) to the rest of the system. Figure 2.1 illustrates the MPB functional blocks that are described in this section.

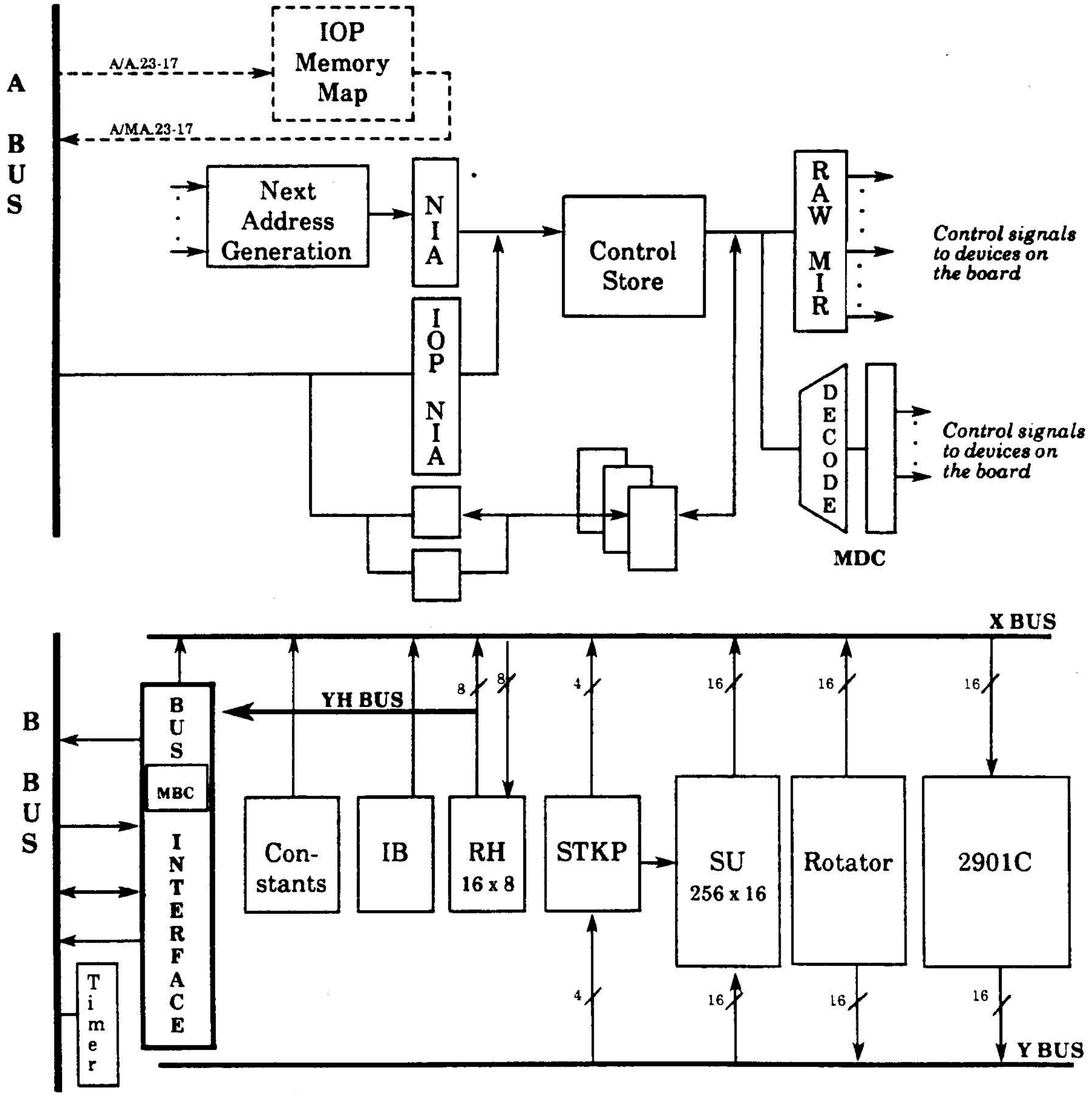


Figure 2.1. Mesa processor board logical blocks

2.1 General Board Hardware

The central processor is a microprogrammed, 16-bit general purpose computer consisting of approximately 170 ICs of various sizes and complexity. It resides on a 10.9-inch by 16-inch printed wiring board assembly (PWBA), referred to as the Mesa Processor Board (MPB), located in slot 3 of the backplane.

The MPB contains:

- 4K by 48-bit writable control store and associated registers for loading and decoding microinstructions. Control store is expandable to 8K by 48 bits. An 84-pin gate array provides look-ahead decoding of certain microinstructions; the raw microinstruction register (RAW MIR) stores microinstructions.

- Four 2901C LSI chips that make up the core of the central processor. The 2901C is a 4-bit processor; the four chips are cascaded to provide a 16-bit processor. Supporting the 2901C are four register sets (U, RH, IB, and Link), a four-bit rotator, and four emulator registers (stackP, ibPtr, pc16, and MInt).

- The 68-pin gate array that serves as the Mesa bus controller, and the logic that interfaces the controller to internal buses (X and Y) and to the Mesa bus and backplane.

- Support devices, such as the process timer and trap machine.

- A 16 MHz clock generator which distributes a clock signal across the backplane.

- IOP address mapping, which is part of the I/O subsystem, and is not discussed in this manual.

Note: Not all devices are shown in Figure 2.1.

Most devices are described in detail in the appropriate subsections of Section 2. This subsection describes the overall MPB; that is, board layout, interfaces, power requirements, and internal clock generation.

2.1.1 Mesa Processor Board (MPB)

Figure 2.2 illustrates the layout of the Mesa Processor board.

Figure 2.2. Mesa Processor Board layout

2.1.2 Backplane Interface

Table 2.1 lists the Mesa processor board interface to the backplane. On the backplane, pins are grouped in six rows of three columns each. The table reflects the grouping.

The board interfaces to the Mesa processor or B bus and the 80186 or A bus. Two interrupt lines connects the Mesa processor with the IOP.

Tables 2.2 and 2.3 list the pins and signals for the interfaces to the Mesa bus and 80186 bus, respectively.

Table 2.1. MPB Backplane Pin Assignment (Front View)

Outmost Inmost

1) 86 Bus

Spare-1	J3.001	GND	J3.002	Spare-2	J3.003
A/AD.07 (bi)	J3.004	A/DT/R' (i)	J3.005	A/AD.15 (bi)	J3.006
A/AD.06 (bi)	J3.007	A/DEN' (i)	J3.008	A/AD.14 (bi)	J3.009
A/AD.05 (bi)	J3.010	GND	J3.011	A/AD.13 (bi)	J3.012
A/AD.04 (bi)	J3.013	A/MemRdy (i)	J3.014	A/AD.12 (bi)	J3.015
A/AD.03 (bi)	J3.016	A/ALE' (i)	J3.017	A/AD.11 (bi)	J3.018
A/AD.02 (bi)	J3.019	A/IOPMemWr' (i)	J3.020	A/AD.10 (bi)	J3.021
A/AD.01 (bi)	J3.022	Spare-3	J3.023	A/AD.09 (bi)	J3.024
A/AD.00 (bi)	J3.025	GND	J3.026	A/AD.08 (bi)	J3.027
GND	J3.028	A/CLK (i)	J3.029	VCC	J3.030

J3.031		GND	J3.032		GND	J3.033		GND
J3.034	(i)	A/AA.19	J3.035	(i)	A/S2'	J3.036	(i)	A/AA.23
J3.037	(i)	A/AA.18	J3.038	(i)	A/S1'	J3.039	(i)	A/AA.22
J3.040	(i)	A/AA.17	J3.041	(i)	A/S0'	J3.042	(i)	A/AA.21
J3.043	(i)	A/AA.16	J3.044	(i)	A/BHE'	J3.045	(i)	A/AA.20
J3.046		A/UCS'	J3.047		GND	J3.048	(i)	A/IOR'
J3.049	(o)	Reserved-0	J3.050	(i)	A/LocRamCS'*	J3.051		Spare-4
J3.052	(i)	A/IOPLock'	*J3.053		GND	J3.054	(i)	A/IOW'
J3.055		A/PCHoldAToArb*	J3.056	(o)	A/IOPMemRd'	J3.057		Spare-5
J3.058		-5V	J3.059		-5V	J3.060		-5V

GND	J3.061
A/IORdy*	J3.064
Spare7	J3.067
SpareID'	J3.070
DBRK/Daisy'*	J3.073
A/PEINT'*	*J3.076
A/MEBIntr'	J3.079
A/VRETINT'*	J3.082
GND	J3.085
A/RawCLK (o)	J3.088

2) MPB–DCM

GND	J3.062	GND	J3.063
A/MA.23 (o)	J3.065	A/MA.22 (o)	J3.066
A/MA.21 (0)	J3.068	A/MA.20 (o)	J3.069
A/MA.19 (o)	J3.071	A/MA.18 (o)	J3.072
A/MA.17 (0)	J3.074	A/AA.16B (o)	J3.075
GND	J3.077	MPB-DCM-spare1	J3.078
MPB-DCM-spare1	J3.080		

GND	J3.083
MPB-DCM-spare3	J3.086
GND	J3.089

3) Mesa Bus

B/MWT' (o)	J3.081
Dawn-Mp-spare1	J3.084
B/MRD' (o)	J3.087
VCC	J3.090

* not used

- more-

Table 2-1. MPB Backplane Pin Assignment (continued)

Outmost | Inmost

IOP-Mesa				**3) MesaBus (continued)**	
J3.091	GND	J3.092	GND	J3.093	GND
J3.094 (i)	A/IOPIntMP'	J3.095	CSWREN*	J3.096	(reserved)
J3.097 (o)	A/MPIntIOP'	J3.098	CSLOAD/SHIFT'*	J3.099 (o)	B/Lock'
J3.100	A/IOPRdNIA	J3.101	CSBUFFEREN*	J3.102 (o)	B/IOR'
J3.103 (o)	A/Halt'	J3.104	CSDATAIN*	J3.105 (o)	B/MemRef'
J3.106	IOP-S-spare1	J3.107	CSSHIFTCLK*	J3.108 (o)	B/IOW'
J3.109 (o)	A/ResetMPB'	J3.110	CSDATAOUT*	J3.111 (i)	B/Rdy
J3.112	VCC	J3.113	VCC	J3.114	VCC
J3.115	VCC	J3.116	VCC	J3.117 (i)	VCC
J3.118	VCC	J3.119	VCC	J3.120	VCC

GND	J3.121	GND	J3.122	GND	J3.123
GND	J3.124	GND	J3.125	GND	J3.126
GND	J3.127	GND	J3.128	GND	J3.129
INTDIS'	J3.130	B/A.18 (o)	J3.131	B/ALE' (o)	J3.132
B/A.23 (o)	J3.133	B/A.17 (o)	J3.134	Dawn-Mp-Spare3	J3.135
B/A.22 (o)	J3.136	GND	J3.137	B/D.11 (bi)	J3.138
B/A.21 (o)	J3.139	B/D.15 (bi)	J3.140	B/D.10 (bi)	J3.141
B/A.20 (o)	J3.142	B/D.14 (bi)	J3.143	B/D.09 (bi)	J3.144
B/A.19	J3.145	B/D.13 (bi)	J3.146	B/D.08 (bi)	J3.147
-12V	J3.148	-12V	J3.149	-12V	J3.150

J3.151	GND	J3.152	GND	J3.153	GND
J3.154 (bi)	B/D.07	J3.155 (bi)	B/D.12	J3.156 (bi)	B/D.06
J3.157 (bi)	B/D.05	J3.158	GND	J3.159 (bi)	B/D.04
J3.160 (bi)	B/D.03	J3.161 (i)	A/Reset'	J3.162 (bi)	B/D.02
J3.163 (bi)	B/D.01	J3.164	GND	J3.165 (bi)	B/D.00

* Not used

Table 2.2. Mesa (B) Bus Interface

Pin (Connector J1)	Signal	Signal Description	Pin (Connector J1)	Signal	Signal Description
J1.140	B/D.15 }		J1.133	B/A.23 }	
J1.143	B/D.14 }		J1.136	B/A.22 }	
J1.146	B/D.13 }		J1.139	B/A.21 }	
J1.155	B/D.12 }		J1.142	B/A.20 }	**B-Bus Address line**
J1.138	B/D.11 }		J1.145	B/A.19 }	
J1.141	B/D.10 }		J1.131	B/A.18 }	
J1.144	B/D.09 }	**B-Bus multiplexed**	J1.134	B/A.17 }	
J1.147	B/D.08 }	**Address/Data line,**			
J1.154	B/D.07 }	**bidirectional data**	J1.105	B/MemRef'	B-bus Mesa memory reference
J1.156	B/D.06 }		J1.087	B/MRD'	B-bus Memory Read
J1.157	B/D.05 }		J1.081	B/MWT'	B-bus Memory Write
J1.159	B/D.04 }		J1.102	B/IOR'	B-bus I/O read
J1.160	B/D.03 }		J1.108	B/IOW'	B-bus I/O write
J1.162	B/D.02 }		J1.099	B/Lock'	B-bus lock request to memory
J1.163	B/D.01 }		J1.132	B/ALE'	Address Latch Enable
J1.165	B/D.00 }		J1.111	B/Rdy	Ready

Table 2.3. 80186 (A) Bus Interface

Pin (Connector J1)	Signal	Signal Description	Pin (Connector J1)	Signal	Signal Description
J1.065	A/MA23 }		J1.088	A/RawCLK	16 MHz clock
J1.066	A/MA.22 }		J1.029	A/CLK	8 MHz clock (not used)
J1.068	A/MA.21 }	**A-bus**	J1.161	A/Reset'	System reset
J1.069	A/MA.20 }	**Mapped Address**	J1.109	A/Reset MPB'	
J1.071	A/MA.19 }		J1.103	A/Halt'	
J1.072	A/MA.18 }		J1.041	A/S0'	A-bus status line
J1.074	A/MA.17 }		J1.038	A/S1'	A-bus status line
J1.075	A/A.16B }		J1.035	A/S2'	A-bus status line
			J1.017	A/ALE'	Address Latch Enable
			J1.005	A/DT/R'	A Bus Data Transmit/Receive
J1.036	A/A.23 }				
J1.039	A/A.22 }		J1.008	A/DEN'	Data Enable
J1.042	A/A.21 }	**A-bus**	J1.020	A/IOPMemWr'	Memory Write
J1.045	A/A.20 }	**Address line**	J1.054	A/IOW'	IO Write
J1.034	A/A.19 }		J1.056	A/IOPMemRd'	Memory Read
J1.037	A/A.18 }		J1.048	A/IOR'	IO Read
J1.040	A/A.17 }		J1.044	A/BHE'	Byte High Enable
J1.043	A/A.16 }		J1.097	A/MPIntIOP'	Mesa processor interrupts IOP, A-bus←Mesa
J1.006	A/AD.15 }		J1.094	A/IOPIntMP'	IOP interrupts Mesa, Mesa←A-bus
J1.009	A/AD.14 }	**Multiplexed**			
J.1012	A/AD.13 }	**Address/Data**	J1.046	A/UCS'	Upper Chip Select
J1.015	A/AD.12 }	**line, A-bus**			
J1.018	A/AD.11 }	**Address line,**			
J1.021	A/AD.10 }	**bidirectional data**			
J1.024	A/AD.09 }				
J1.027	A/AD.08 }				
J1.004	A/AD.07 }				
J1.007	A/AD.06 }				
J1.010	A/AD.05 }				
J1.013	A/AD.04 }				
J1.016	A/AD.03 }				
J1.019	A/AD.02 }				
J1.022	A/AD.01 }				
J1.025	A/AD.00 }				

2.1.3 Power

Power consumption estimates for the Mesa Processor Board are:

Typical 8.187 A (40.94 W)
max. 12.318 A (61.59 W)

$$\text{Average} = \frac{\text{Typical + Max}}{2}$$

$$= 10.253\text{ A}\quad(51.26\text{ W})$$

Table 2.4 lists power interface connections.

Note: For detailed dc power distribution, please see Section 1.3.

Table 2.4. Power Interface

RAW 5v	J1.030	GND	J1.002	GND	J1.092
	J1.090		J1.011		J1.093
	J1.112		J1.026		J1.121
	J1.113		J1.028		J1.122
	J1.114		J1.031		J1.123
	J1.115		J1.032		J1.124
	J1.116		J1.033		J1.125
	J1.117		J1.047		J1.126
	J1.118		J1.061		J1.127
	J1.119		J1.062		J1.128
	J1.120		J1.063		J1.129
			J1.077		J1.137
			J1.083		J1.151
			J1.085		J1.152
			J1.089		J1.153
			J1.091		J1.158
					J1.164

2.1.4 Clock Generation

Figure 2.3 illustrates the relationship of the generated clocks to the system clock.

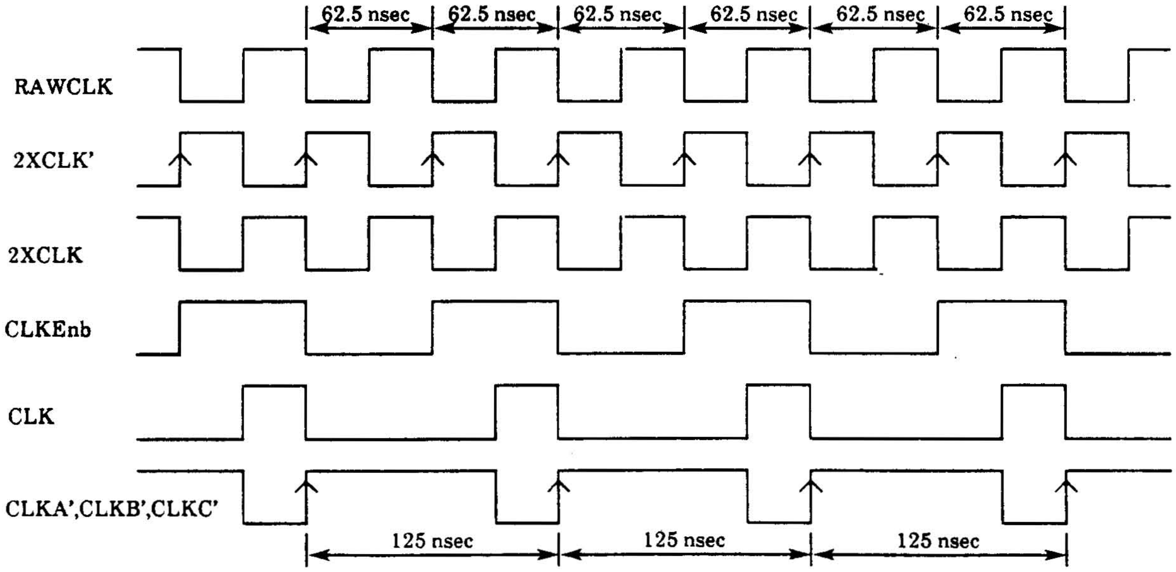


Figure 2.3. Internally generated clocks

2.2 Microinstructions

Dove microcode implements Mesa bytecodes, as defined by the Mesa Processor Principles of Operation. The microcode does not control I/O devices, which are controlled by the IOP, an Intel 80186 microprocessor.

The microcode resides in RAM control store. The Mesa processor interprets control store through one of two devices: a microinstruction decoder gate array chip (MDC), and a microinstruction register (MIR). Control store is written by the IOP; the IOP also reads control store, but only the next instruction and only 8 bits at a time. During booting or debugging, the IOP can load microcode into control store, initialize the microcode program counter, and start and stop Mesa processor execution.

Microcode source files consist of lists of microinstructions, assembler macros, and comments. Microinstructions consist of a list of one or more phrases. (Refer to the examples at the end of this subsection.)

In this and subsequent sections discussing microinstructions, the following symbols are used:

~ logical complement
← assignment
,, (double comma) concatenation.

2.2.1 Hardware

Microinstruction hardware is described in section 2.3 titled "Control Architecture."

2.2.2 Theory of Operations

Up to 8K microinstructions can be written into (or read from) the control store RAM by the IOP.

Each microinstruction is decoded and executed in 125 nanoseconds, or one cycle. Microinstructions are not pipelined over several cycles, except that while one microinstruction is being executed, its successor is being read from control store.

Cycles are enumerated in c1, c2, and c3 order, and then c1 again. The sequence is never interrupted or altered. Consequently, both targets of a two-way branch must be specified with the same cycle number. (Strictly speaking, this is necessary only if the target microinstructions contain cycle-dependent operations.)

Three successful cycles, c1, c2, and c3, are grouped into one click. Five consecutive clicks (numbered 0..4) are grouped into a round. Each click of a round is permanently allocated to one or more of the I/O controllers. If an I/O controller does not request the service of its corresponding task microcode, the emulator microcode task runs during that click instead.

Microinstruction alignment, so that microinstructions execute in successful cycles, is, therefore, a necessary outcome of the fixed-task click structure. Moreover, when one desires code which is speed optimized, this structure requires the elimination of three microinstructions instead of one.

Look-ahead decoding of microinstructions is done in the MDC for the function fields fS, fX, fY, and fZ. The raw MIR stores the entire microinstruction except the Immediate Next Instruction Address (pINIA) field and the fS field.

The pINIA field, together with the branching logic, generates the Next Instruction Address inputs (pNIA) to the Next Instruction Address register.

2.2.3 Programmer Interface

Microinstructions are executed from a 4K by 48-bit, writable control store. Each 48-bit microinstruction contains the 12-bit address of the next instruction. Throughout this section, the subsections titled "Programmer Interface" are described in terms of microinstructions. Microinstruction examples in subsection 2.2.3.2 illustrate how certain elementary functions are accomplished.

Refer to Daybreak Microcode Reference Manual for detailed microcode instructions.

2.2.3.1 Microinstruction Format

Frequently applied operations are encoded in the smallest number of bits, and most of the important Mesa Emulator operations execute in one click.

The three major parts of a 48-bit microinstruction are: 1) the 2901 control bits (bits 0 through 15); miscellaneous function bits (bits 16-35); and the 'goto' address field (bits 36-47).

2901 control bits occupy the first word. They control the R register ports A and B, and specify ALU source address, function, and destination address.

Miscellaneous function fields control carry input, enable the stack and U registers, specify a memory operation, and specify functions (fX, fY, and fZ).

The fS field controls the decoding of the fY and fZ function fields:

Depending on fS0-1, **fY field** can

- specify a miscellaneous function (fYNorm)
- name a branch or multi-way dispatch (DispBr)
- name an I/O register to be loaded (IOOut)
- equal the high nibble of an 8-bit constant (Byte)

Depending on fS3-4, **fZ field** can

- specify a miscellaneous function (fZNorm)
- equal the low half of a U register address (Uaddr)
- name an I/O register to be read (IOXIn)
- equal a 4-bit constant or the low half of an 8-bit constant (Nibble)

The 'Goto' address, INIA, occupies 12 bits and specifies a control store address unless the previous microinstruction specifies condition bits. Condition bits are ORed into INIA, resulting in a branch or dispatch. Thus, every microinstruction is a potential jump instruction.

Figure 2.4 illustrates the microinstruction format and describes the fields and subfields.

00	04	08	11	14	16	20	24	28	32	36	47
rA	rB	aS	aF	aD	~~eP~~,Cin, enU, mem	fS	fX	fY	fZ	INIA	

Field	Description
rA	2901 A reg addr, U addr [0-3]
rB	2901 B reg addr, RH addr
aS	2901 ALU source operand pair
aF	2901 ALU function
aD	2901 ALU desination/shift control
~~eP~~	~~Even parity~~
Cin	2901 Carry in, shift ends, write U(if enU = 1)
mem	MAR←(if c1), MDR←(if c2), ←MD (if c3)
fS	Function field selector
fX	X function
fY	Y function
fZ	Z function
INIA	Next instruction address

2901 Control Bits

aS	R,S
0	A,Q
1	A,B
2	0,Q
3	0,B
4	0,A
5	D,A
6	D,Q
7	D,0

aF	F
0	R+S+Cin
1	S-R-Cin'
2	0,Q
3	0,B
4	0,A
5	D,A
6	D,Q
7	D,0

sh,,aD	r[rB]	Q←	Y bus←
0	no write	F	F
1	no write	no write	F
2	F	no write	A
3	F	no write	F
4	F/2	Q/2	F
5	F/2	no write	F
6	2F	2Q	F
7	2F	no write	F

Function Field Selector and Function Fields

fS[0-1]	fY =
0	DispBr
1	fYNorm
2	IOOut
3	Byte*

* X[0-11]←0.
X[8-15]←fY,fZ

fS[2-3]	fZ =	SU addr[0-7]
0	fZNorm	0,,stackP
1	Nibble*	0,,stackP
2	Uaddr[4-7]	rA,,fZ I rA,,Y[12-15]** if fZ = AltUaddr**
3	IOXIn	rA,,fZ I rA,,Y[12-15]** if fZ = AltUaddr**

* X[0-11]←0,
X[12-15]←fZ

**As executed by a previous µinstruction

fX	fXNorm
0	pCall/Ret0*
1	pcall/Ret1*
2	pCall/Ret2*
3	pCall/Ret3*
4	pCall/Ret4*
5	pCall/Ret5*
6	pCall/Ret6*
7	pCall/Ret7*
8	Noop
9	RH←
A	shift (shiftX)
B	cycle (cycleX)
C	Cin←pc16
D	Map←, or (MapRefX)
E	pop (popX)
F	push (pushX)

* PCall when NIA7 = 0;
PRet when NIA7 = 1

fY	fYNorm	DispBr	IOOut
0	ClrMPIntIOP	NegBr	DebA←
1	SetMPIntIOP	ZeroBr	ExCtrl←
2	ClrIntErr (ClrIntTrap)	NZeroBr	
3	IBDisp	MesaIntBr	
4	MesaIntRq	PgCarryBr	
5	stackP←	CarryBr	
6	IB←	XRefBr	
7	cycle (cycle Y)	NibCarryBr	
8	Noop	XDisp	
9	Map←. or (MapRefY)	YDisp	
A	MAPA←	XC2npcDisp	
B	push (pushY)	XWtOKDisp	
C	IO←	XwdDisp	IO←
D	Bank←	XHDisp	
E	ClrIE	XLDisp	
F	SetIE	PgDrOvDisp	

fZ	fZNorm	IOXIn
0		
1	IBPtr←1	
2	IBPtr←0	
3	Cin←pc16	
4		
5	pop (popZ)	
6	push (pushZ)	←ExtStat
7	AltUaddr	IO←
8	Noop	←DebB
9		←IntStat
A	ClrLOCK	←ErrnIBnStkp (Misc)
B	SetLOCK	←RH
C	LRot0	←ibNA
D	LRot12	←ib
E	LRot8	←ibLow
F	LRot4	←ibHIgh

Figure 2.4. Microinstruction format and subfield formats

2.2.3.2 Microinstruction Examples

The Central Processor hardware should be viewed in light of its corresponding microcode. The following four examples of microcode illustrate how and in what time frame certain elementary functions are accomplished. See the Daybreak Microcode Reference Manual for a description of the microcode format.

(1) The Mesa Emulator Load Local 1 (LL1) macroinstruction indexes the local frame pointer and then pushes the addressed word from memory onto the Stack. If the indexing operation does not cross a page boundary, then the microinstruction executes in one click. If a page cross occurs, then the microinstruction executes in three clicks. If the Map flags must be updated (RMapFix), then another two clicks are required.

```
@LL1:    MAR ← Q ← [rhL, L + 1], L1←L1.PopDec, push,  c1, opcode[1'b];
LLn:STK ← TOS, PC ← PC + PC16, IBDisp, L2←L2.LL, BRANCH[LLa,LLb,1],     c2;
LLa: TOS ← MD, push, fZpop, DISPNI[OpTable],c3;
LLb: Rx ← UvL,     c3;

LSMap:  Noop,     c1;
     Q ← Q - Rx, L2Disp,      c2;
     Q ← Q and 0FF, RET[LSRtn],      c3;

LLMap:  Map ← Q ← [rhMDS, Rx + Q],      c1, at[3,10,LSRtn];
     Noop,     c2;
     Rx ← rhRx ← MD, XRefBr,   c3;

     MAR ← [rhRx, Q + 0], L0←L0.R, BRANCH[RMUD,$],     c1;
     IBDisp, GOTO[LLa],     c2;
RMUD:  CALL[RMapFix],  c2;
```

(2) The Mesa Emulator Read 1 (R1) macroinstruction indexes the virtual address on the top of stack and then pushes the addressed word from memory onto the stack. The microinstruction executes in two clicks. If the page has been read for the first time, then four clicks are required; that is, the Map flags must be updated.

```
@R1:     Map ← Q ← [rhMDS, TOS + 1], L1←L1.Dec, pop,     c1, opcode[101'b];
     push, PC ← PC + PC16,c2;
     Rx ← rhRx ← MD, XRefBr,   c3;

     MAR ← [rhRx, Q + 0], L0←L0.R, BRANCH[RMUD,$],     c1;
     IBDisp, GOTO[LLa],     c2;
```

(3) The Mesa Emulator Jump 2 (J2) macroinstruction increments the program counter by 2 bytecodes and then refills the instruction buffer. The microinstruction executes in two clicks. If the jump crosses a page boundary, then five clicks are required .

```
@J2:     MAR ←PC ←[rhPC, PC+1], push,     c1,opcode[201'b];
     STK ←TOS, L2 ←L2.Pop0IncrX, Xbus←0, XC2npcDisp, DISP2[jnPNoCross], c2;


jnPNoCross:  IB ←MD, pop, DISP4[JPtr1Pop0, 2],   c3, at[0,4,jnPNoCross];
jnP1Cross:    Q ←0FF + 1, L0 ←L0.JRemap, CANCELBR[UpdatePC, 0F],  c3,
at[2,4,jnPNoCross];


JPtr1Pop0:    MAR ←[rhPC, PC + 1], IBPtr←1, push, GOTO[Jgo], c1,
at[2,10,JPtr1Pop0];
JPtr0Pop0:    MAR ←[rhPC, PC + 1], IBPtr←0, push, GOTO[Jgo], c1,
at[3,10,JPtr1Pop0];
Jgo: TOS ←STK, AlwaysIBDisp, L0 ←L0.NERefill.Set, DISP2[NoRCross],   c2;
```

(4) The Mesa Emulator instruction buffer refill code executes in one click if the buffer was not empty. If the buffer was empty, then two clicks are required. If the refill occurs across a page boundary, then four to six clicks are required

```
{Buffer Empty Refill. Control goes from NoRCross to RefillNE since RefillE + 1 does not
contain an IBDisp.}
RefillE:  MAR ←[rhPC, PC], PC ←PC-1, L0 ←L0.ERefill,      c1, at[400];
     PC ←PC + 1, DISP2[NoRCross],  c2;


{Buffer Not Empty Refill.}
OpTable:      {"Noop" location of Instruction Dispatch table}
RefillNE: MAR ←[rhPC, PC + 1],c1, at[500];
     AlwaysIBDisp, L0 ←L0.NERefill.Set, DISP2[NoRCross], c2;


NoRCross:     IB ←MD, uPCCross ←0, DISPNI[OpTable],      c3, at[0,4,NoRCross];
RCross:  Q ←0FF + 1, GOTO[UpdatePC], c3, at[2,4,NoRCross];
```

2.3 Microinstruction Control Architecture

Microinstructions are loaded into control store from the IOP via data transfer on the 80186 bus. During the execution of a program, microinstructions are read from control store RAM and stored in the Raw Microinstruction register, except for the pINIA field and the pfS field.

The pINIA field, together with the branching logic, generates the pNIA inputs to the Next Instruction Address register.

The encoded pfS, pfX, pfY, and pfZ fields are fed to the MDC, decoded into instruction commands, and stored in corresponding command registers in the MDC.

Because the Mesa processor is split into lower and upper bytes with no propagated carry, the paS and paF fields are modified to provide separate aShL and aFL for the low byte and aShH and aFH for the high byte.

Figure 2.5 illustrates microinstruction control architecture.

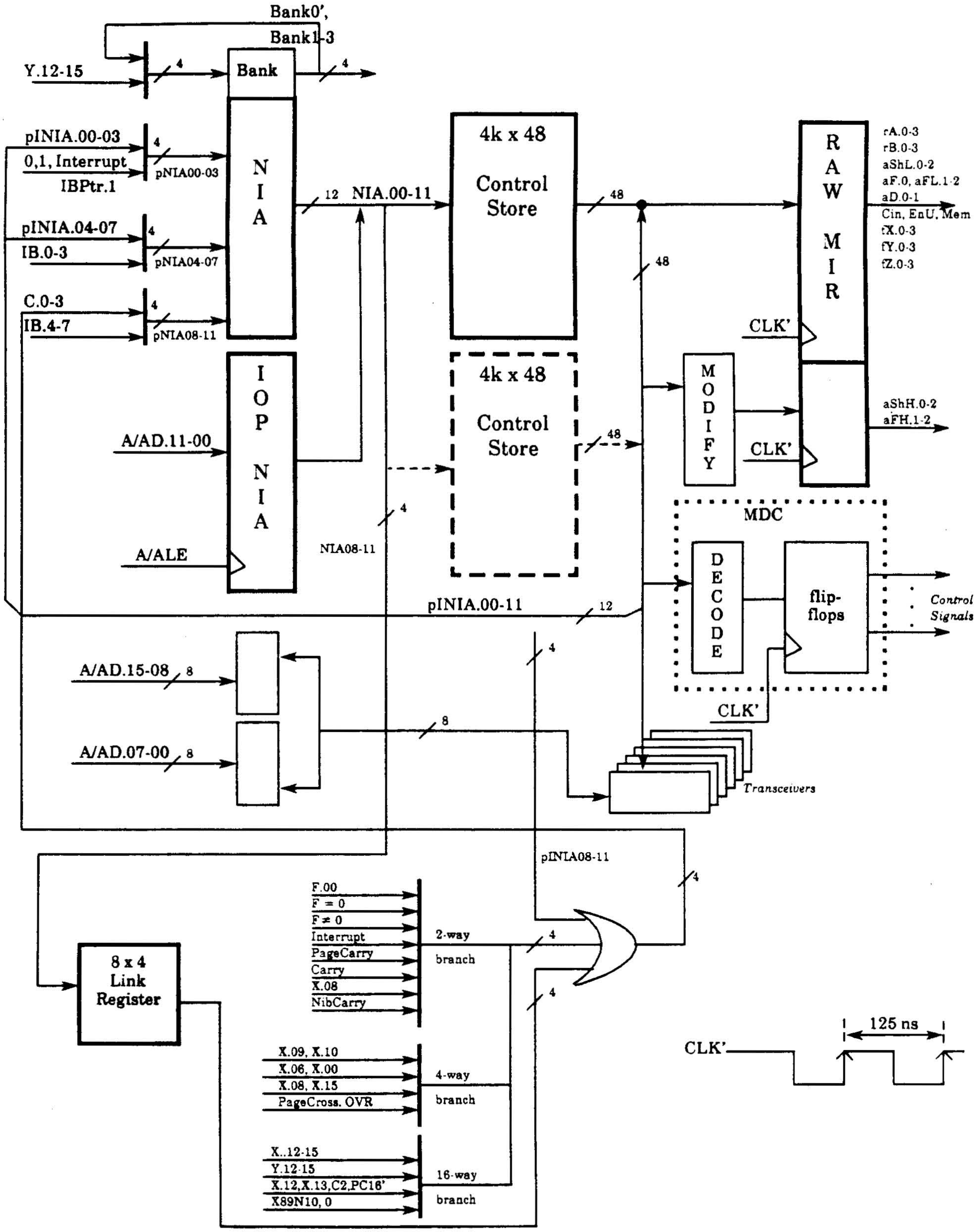


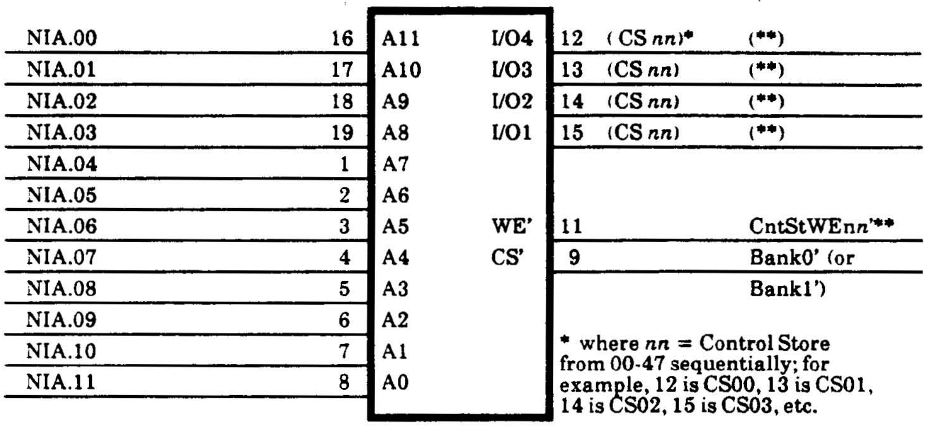
Figure 2.5. Microinstruction control architecture

2.3.1 Hardware

Control hardware described below consists of the control store, the control store interface, the Microinstruction Decoder Chip (MDC), and the MicroInstruction Register (MIR). Other control hardware, consisting of the interface to the IOP, miscellaneous support logic, the trap machine, and MInt and Link registers, is described elsewhere.

2.3.1.1 Control Store

Two banks, each consisting of twelve 4K x 4 static RAM chips with a 55 ns access time, make up the 4K x 48 writable control store, expandable to 8K x 48. Figure 2.6 illustrates the control store pins and signals. For signal functions, refer to Figure 2.4, illustrating microinstruction format.



* where *nn* = Control Store from 00-47 sequentially; for example, 12 is CS00, 13 is CS01, 14 is CS02, 15 is CS03, etc.

**Control Store Assignments

As above (NIA.00:11) for all

IF CS is	THEN I/O4 is	 I/O3 is	 I/O2 is	 I/O1 is	 WE' is
0-3	prA.0	prA.1	prA.2	prA.3	0'
4-7	prB.0	prB.1	prB.2	prB.3	0'
8-11	paS.0	paS.1	psS.2	paF.0	1'
12-15	paF.1	paF.2	paD.0	paD.1	1'
16-19	pEP	pCin	pEnU	pMem	2'
20-23	pfS.0	pfS.1	pfS.2	pfS.3	2'
24-27	pfX.0	pfX.1	pfX.2	pfX.3	3'
28-31	pfY.0	pfY.1	pfY.2	pfY.3	3'
32-35	pfZ.0	pfZ.1	pfZ.2	pfZ.3	4'
36-39	p.INIA.00	.01	.02	.03	4'
40-43	p.INIA.04	.05	.06	.07	5'
44-47	p.INIA.08'	.09'	.10'	.11'	5'

Figure 2.6. Writable control store pins and signals

2.3.1.2 Control Store Interface

The control store interface consists of a bank register, address registers, and supporting logic. Table 2.5 summarizes control store interface signals.

Table 2.5. Control Store Interface Signals

Interface to:	Signal	Function
Control Store	1. CntStWE0'/1'	1. Control store Write Enable 0 (byte 0) and 1 (byte 1).
	2. Bank0'/1/2/3	2. Control store RAM bank select 0, 1, 2, 3.
	3. NIA.00-11	3. Next Instruction Address 00-11 - Control store RAM address.
Control store 00-03	4. prA.0-3	4. Pipelined microinstruction rA field.
04-07	5. prB.0-3	5. Pipelined microinstruction rB field.
14-15	6. paD.0-1	6. Pipelined microinstruction aD field.
11-13	7. paF.0-2	7. Pipelined microinstruction aF field.
08-10	8. paS.0-2	8. Pipelined microinstruction aS field.
	9. CntStWE2'/3'	9. Control store Write Enable2 (byte 2) and 3 (byte 3).
17	10. pCin	10. Pipelined microinstruction CarryIn bit.
18	11. pEnU	11. Pipelined microinstruction Enable U register bit.
16	12. pEP	12. Pipelined microinstruction Enable Parity bit (not used).
19	13. pMem	13. Pipelined microinstruction Memory reference bit
20-23	14. pfS.0-3	14. Pipelined microinstruction fS field.
24-27	15. pfX0-3	15. Pipelined microinstruction fX field.
28-31	16. pfY.0-3	16. Pipelined microinstruction fY field.
	17. CntStWE4'/5'	17. Control store Write Enable 4 (byte 4)and 5 (byte 5).
36-43	18. pINIA.00-07	18. Pipelined Immediate Next Instruction address 00-07.
44-47	19. pINIA.08'-11'	19. Pipelined Immediate Next Instruction address 08-11.
32-35	20. pfZ.0-3	20. Pipelined microinstruction fZ field.
Mesa processor control line	RunModeB'	Mesa processor Run mode
Next Address Register	NIA.00:11	Next Instruction Address 00:11.
80186 bus interface	1. A/AD.00:15	1. Address/Data bus 00:15.
	2. A/BHE'	2. Byte High Enable.
	3. A/IOR'	3. I/O Read.
	4. A/DEN'	4. Data Enable.
	5. A/DT/R'	5. Data Transmit/Receive (direction).
	6. A/IOW'	6. I/O write.
	7. A/S0'-2'	7. A bus Status bits 0-2
	8. A/ALE	8. Address Latch Enable.
Internal logic signalS	1. A/A.11B:15B	1. Buffered A bus address 11-15.
	2. A/DENB'	2. Buffered A bus Data Enable
	3. A/DirB	3. Buffered A bus Direction control.
	4. A/IOWB'	4. Buffered A bus I/O write.
	5. CntStDen0'-5'	5. Control store Data Enable 0' - 5'.
	6. CSMP'	6. Control store or Mesa processor select.
	7. DB.0-7	7. Transceiver-buffered A bus data (byte only).
	8. IOPDisable (IOPDisable')	8. Disable signal from debugger.
	9. LdBankReg'	9. Load Bank Register (A bus).
	10. NIAEn'	10. Next Instruction Address Enable (A bus side).
	11. SelectMP'	11. Select Mesa Processor (E000-E777).
	12. EnNIA	12. Next Instruction Addresss enable (Mesa processsor side).
	13. WriteBank	13. Write Bank register (Mesa processor).

2.3.1.3 Microinstruction Decoder Chip (MDC)

Microinstructions are decoded via an 84-pin gate array chip (MDC). Figure 2.7 illustrates the signals for the microinstruction decoder gate array chip. Table 2.6 describes the signals.

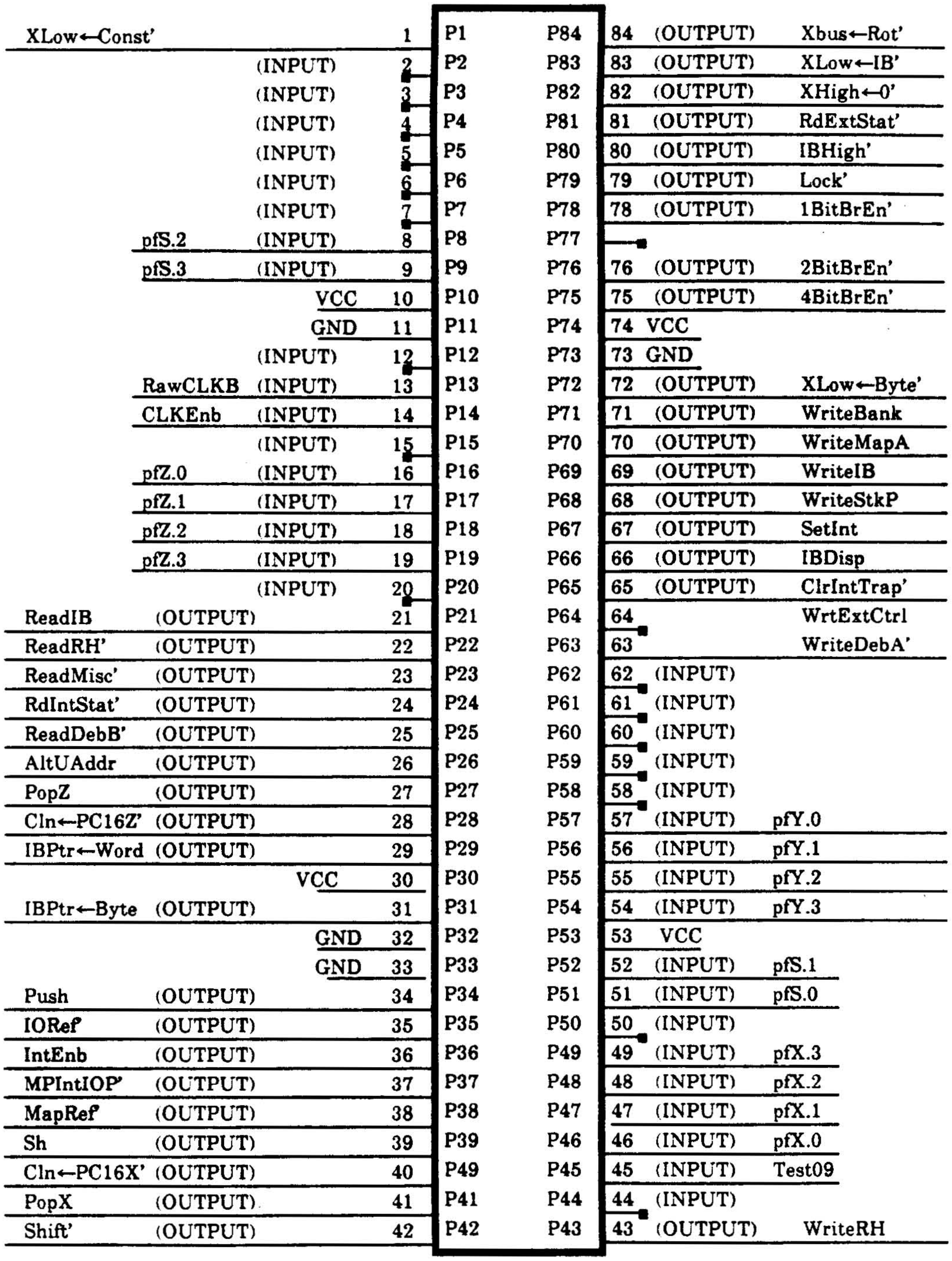


Figure 2.7. MDC pins and signals

Table 2.6. MDC Signal Description

Signal	Function
1/2/4BitBrEn'	1 bit, 2 bit, 4 bit branch enable.
AltUAddr	Select U address source from lower Y bus nibble.
CLKEnb	Processor clock enable (8 MHz).

Cln←PC16Z'
Cln←PC16X'
	PC16 becomes the carry input of the 2901 (ALU).

ClrIntTrap'	Clear interrupt trap.
IBDisp	Instruction buffer dispatch.
IBHigh'	Puts the high 4 bits of the IBFront onto X (12-15). High order X bus bits are zeroed.
IBPtr←Byte	Instruction buffer pointer gets byte.
IBPtr←Word	Instruction buffer pointer gets word.
IntEnb	Enable interrupt.
IORef'	Input/Output reference.
Lock'	Memory lock by Mesa.
MapRef'	Memory map reference.
MPIntIOP'	Mesa processor interrupts IOP.
pfS.0-3	Pipelined microinstruction fS field.
pfX.0-3	Pipelined microinstruction fX field.
pfY.0-3	Pipelined microinstruction fY field.
pfZ.0 - 3	Pipelined microinstruction fZ field.
PopZ/X	Pop from the stack.
Push	Push to the stack.
RawCLKB	16 MHz system clock.
RdExtStat'	Read External Status.
RdIntStat'	Read Interrupt Status.
ReadIB	Read Instruction Buffer.
ReadMisc'	Read miscellaneous status information through the X bus.
ReadRH'	Read RH registers through X bus.
SetInt	Set interrupt.
Sh	ALU destination control with shift up and down enable.
Shift'	Single, double, left, right shift enable.
Test09	Testability input, sets all outputs of MDC high.

- more -

Table 2.6. MDC Signal Description (continued)

Signal	Function
WriteBank	Write Bank selection data from Y-bus (Y12-15) to the bank register.
WriteDebA'	Write to the debugger mailbox from the X bus.
WriteIB	Write to instruction buffer.
WriteRH	Write RH register enable.
WriteStkP	Write to the stack pointer.
Xbus←Rot'	X bus gets Rotation.
XHigh←0'	X bus high byte gets zero.
XLow←Byte'	X bus low byte gets constant data such that X.08-11 = fY0-3.
XLow←Const'	X bus low byte gets constant, byte or nibble constant.
XLow←IB'	X bus low byte gets ibFront. Either the full byte or nibble can be read into the X bus, such that all other X bits are set to zero.

2.3.1.4 MicroInstruction Register (MIR)

The MicroInstruction Register (MIR) consists of five ALS374 chips. The MIR stores microinstructions, except for fS and pINIA fields. Table 2.7 summarizes MIR interfaces.

Table 2.7. Raw MIR Interfaces

Interface to:	Signal(s)	Function
ALU	aD.0/1; aF.0; aFH.1/2; aFL.1/2; aShH.0-2; aShL.0-2; rA.0-3; rB.0-3	See Figure 2.4 for microinstruction field definitions.
ALU Carry & Shift	aD.0/1; aFL.1; Cln	See Figure 2.4 for microinstruction field definitions.
Branch and Link registers	fX.0-3; fY.1-3; fZ.0-1	See Figure 2.4 for microinstruction field definitions.
Constants register	fY.0:3; fZ.0:3	See Figure 2.4 for microinstruction field definitions.
Control store	Bank0 [0-15]: paD.0/1; paF.0:2; paS.0:2; pMem; prA.0-3;.prB.0-3 Bank0 [16-31]: pCln;.pEnU; pEP; pfX.0-3; pfY.0-3; pfZ.0-3; pMem	See Figure 2.4 for microinstruction field definitions.
RH register	rB.0-3	See Figure 2.4 for microinstruction field definitions
Rotator	fZ.2-3	See Figure 2.4 for microinstruction field definitions
U register	EnU; Cin	See Figure 2.4 for microinstruction field definitions
IB state control	Mem	Memory bit. If set and the instruction is executing in c1, then MAR is loaded from YH,,Y. If set in c2, then memory write data register is loaded from the Y bus and the memory location is written. If set in c3, then returning memory data is placed onto the X bus.
MBC	Cycle3	Mesa processor cycle 3.
	CLKB'	8 MHz processor clock.

2.3.2 Theory of Operations: Mode Control

The IOP interfaces with the CP both as a standard I/O controller and as a boot loader/debugger. This subsection discusses the loading of control store (Boot mode), the initial trapping of microprograms (Run mode), and the reading of Next Instruction (Stop mode).

2.3.2.1 Boot Mode

Either of the two reset signals, A/Reset' or A/ResetMPB', initializes the CP to a quiescent condition. The first A/Halt' signal puts the CP into Boot mode.

In Boot mode, the IOP loads control store by writing to its I/O space 8000H to DFFFH, 24 Kbytes (or 4K x 48 bits; that is, one bank). The least six decoded numbers (000 - 101) of A bus address bits 14, 13, and 12, together with the A bus status bits A/S0-2 and Data Enable (A/DEN'), generate six enable signals (En0' - 5') to enable data to the control store RAM. The corresponding six decoded write enables (CntStWE0' - 5') are applied to the appropriate control store RAM to write in the data.

Data is enabled a byte at a time, with the selection of high or low byte controlled by the signal A/BHE' (A bus Byte High Enable) and A/A.00B (A bus Address bit 00 buffered), respectively. Direction of data flow is controlled by A/DT/R' (A bus Data Transmit/Receive NOT), a high A/DT/R' implying data transmitted from the IOP to the control store and vice versa. Since control store is 48 bits wide and data is enabled only 8 bits at a time, a software algorithm must be exercised to load the correct byte into the correct location.

2.3.2.2 Run Mode

When the IOP finishes loading control store, it deactivates the A/Halt' signal to put the CP into Run mode. As CP enters Run mode, the first order of business is to generate an InitTrap (Initialization Trap) signal, which in turn causes a trap to location 0.

In Run mode, the IOP is isolated from the Mesa processsor except for the two reset signals (A/Reset' and A/ResetMPB'), two mutual interrupt lines (A/IOPIntMP' and A/MPIntIOP'), and the A/Halt' signal.

2.3.2.3 Stop Mode

An active A/Halt' during the Run mode puts the processor in the Stop mode.

In the Stop mode, the 8 MHz processor clock (CLK, CLKA, CLKB, and CLKC) is stopped, putting the Mesa processor in a hold state. At this time, the IOP can activate the A/IOPRdNIA (IOP Read Next Instruction) signal, which enables the Next Instruction Address to the control store RAM.

IOP then issues an A/IOR' (A bus I/O Read) to read the "Next Instruction." This Next Instruction being read is addressed by the Next Instruction Address register (not alterable by the IOP). However, the byte to be read is selectable by the IOP and is controlled by the A bus address bits 14, 13, and 12. The signal A/IOPRdNIA remains active for the entire duration of the IOP Read Next Instruction process.

It must be pointed out that the IOP can only halt the Mesa processor randomly and that only the Next Instruction, **not** the Next Instruction Address, can be read back a byte at a time. The actual Next Instruction Address has to be arrived at through certain deductions from the information read back.

The IOP can restore the Mesa processor to its Run mode simply by dropping the A/Halt signal.

2.3.2.4 Mode Control Timing

Figure 2.8 illustrates mode control timing.

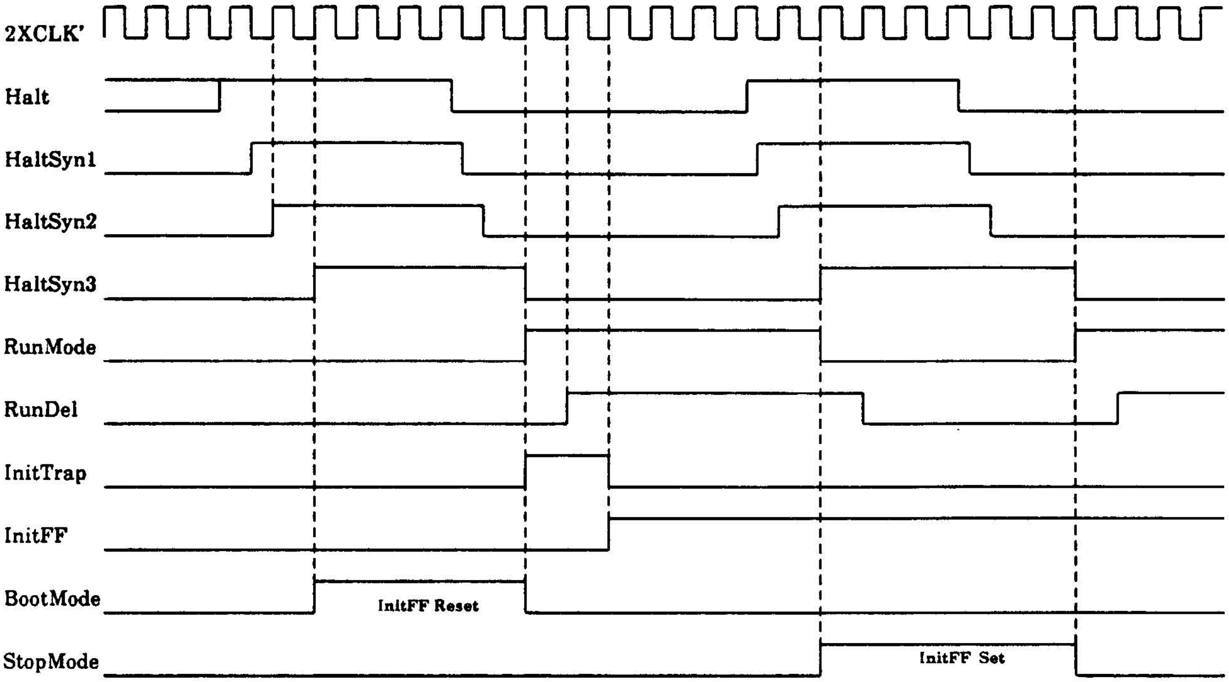


Figure 2.8. Mode (boot/run/stop) control timing

2.3.3 Programmer Interface

This section briefly discusses the algorithms that control the execution of microinstructions. For each subsection, refer also to the Daybreak Microcode Reference Manual. Refer also to Figure 2.4.

Figure 2.9 illustrates the location of the writable control store in the I/O address space map.

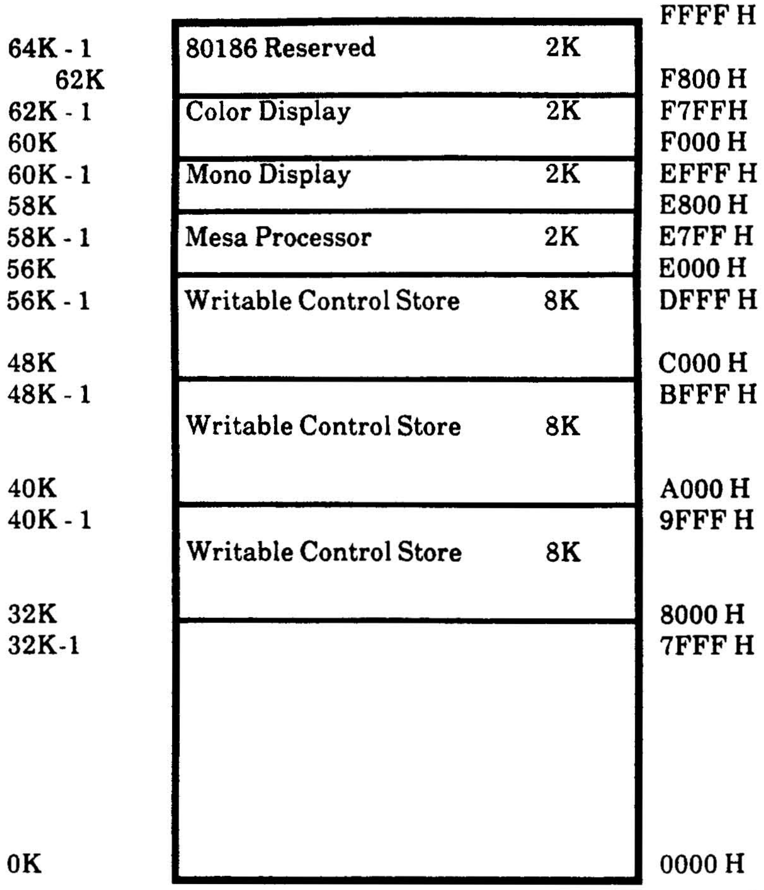


Figure 2.9. I/O address space map

2.3.3.1 Conditional Branching and Dispatching

Every microinstruction can potentially branch. During each cycle, condition bits specified by the executing microinstruction are ORed into the next instruction's go-to-address field (INIA) being read from control store. At the end of the cycle, the resulting addresss (NIA) reads the next microinstruction. If the executing microinstruction does not specify a branch function, then 0 is ORed into INIA, and a branch does not occur. When a microinstruction specifies a dispatch function, up to 4 bits are ORed into the INIA field, selecting one of up to 16 target microinstructions.

Thus, all branches and dispatches take two cycles to complete: one cycle to specify the branch and one cycle to read out the target microinstruction. The microinstruction bits required to specify a branch are fS[0-1] = DispBr and the fY field that names the branch or dispatch.

Table 2.8 lists branches and dispatches.

Table 2.8. Conditional Branching and Dispatching

Mnemonic	Source	INIA	Remarks
NegBR	F[0]	11	sign of ALU result (not necessarily Y[0])
ZeroBr	F=0	11	ALU output equal to zero
NZeroBr	F≠0	11	ALU output not equal to zero
CarryBr	Cout[0]	11	ALU carry out
NibCarryBr	Cout[12]	11	ALU carry out from low nibble
PgCarryBr	Cout[8]	11	ALU carry out from low byte
XRefBr	X[11]	11	present & referenced Map bit
MesaIntBr	Interrupt	11	Emulator interrupt
XwdDisp	X[9],,X[10]	[10-11]	write protect & dirty Map bits
XHDisp	X[4],,X[0]	[10-11]	X (high) bus
XLDisp	X[8],,X[15]	[10-11]	X (low) bus
PgCrOvDisp	PgCross,,OVR	[10-11]	pageCross & ALU overflow
XDisp	X[12-15]	[8-11]	low nibble of X bus
YDisp	Y[12-15]	[8-11]	low nibble of Y bus
XC2npcDisp	X[12-13],,c2,,~pc16	[8-11]	X bus, cycle 2, inverse of pc16
XW + OKDisp	1,1,(X.08 and X.09 and X.10'),0	[8-11]	I/O branches (bp = backplane pin) X.08 = ref; X.09 = dirty; X.10 = wp'
IBDisp	ibFront	[4-11]	Instruction Buffer
LnDisp	Linkn	[8-11]	Link register (n = 0..3)

Equivalent names: Ether Disp = YIODisp; XDirtyDisp = XLDisp

Both targets of a two-way branch must be specified with the same cycle number.

The following notation is used to specify branching behavior:

- A microinstruction is located in control store at its Instruction Address, IA.
- The Next Instruction Address, NIA, is the control store address register.
- The Intermediate Next Instruction Address, INIA, is the 12-bit goto address present in each microinstruction.

At every cycle, the condition bits specified by fY (DispBr) and the Link register specified by fX are ORed into INIA, thereby producing the NIA value used for the next cycle; that is,

NIA[0-11] ← INIA[0-11] OR DispBr[0-3] OR Link[0-3]

For dispatches, target instructions for each possible outcome need not be provided. A particular condition bit is ignored when its corresponding position in INIA equals 1. This method can also cancel unwanted, pending branches.

Note that, in some cases, there is more than one way to branch on a particular bit; note also that a bit on the low half of the X bus can be branched on. The NZeroBr allows code to be more readily shared.

2.3.3.2 Instruction Buffer Dispatch

The instruction buffer dispatch (IBDisp) is a special dispatch, since more than four bits are ORed into INIA. Consequently, IBDisp can occur only in c1 or c2, and is restricted by convention to c2. Refer to section 2.4.3.7 for a discussion of the instruction buffer.

Assuming that the instruction buffer is full, IBDisp can cause a 256-way dispatch based on the value of ibFront. NIA[4-7] is set to the high nibble of ibFront, and the low nibble of ibFront is ORed with INIA[8-11]. Except for the four IB-Refill trap values, INIA[0-3] is unaffected by the IBDisp. Therefore, up to twelve 256-way dispatch tables can be used concurrently.

If the buffer is not full (ibPtr ≠ full) when an IBDisp is executed, or if an interrupt is pending, then an IB-Refill trap occurs. Refer to section 2.3.3.5.

A special version of IBDisp (AlwaysIBDisp) never traps to IB-Refill, but dispatches on ibFront even if an interrupt is pending (MInt = 1) or if the buffer is not full. AlwaysIBDisp is used in the emulator refill and jump microcode to dispatch on ibFront while the buffer is still being filled. AlwaysIBDISP is encoded:

fY = IBDisp and fZ = IBPtr←1.

If the microinstruction executed before an IBDisp or AlwaysIBDisp causes an IB-Empty Error trap, or if the microinstruction contains a MAR← and the 2901 computation results in pageCross = 1, then the IB dispatch (or possible IB-Refill trap) does not occur and ibPtr remains unaffected. Since INIA is not modified in this case, control transfers to the first entry of the macroinstruction dispatch table. Accordingly, emulator opcode 0 should not be assigned to a macroinstruction.

2.3.3.3 Interrupt Register

The 1-bit Interrupt Register interrupts the contiguous execution of emulator macroinstructions. When it is set in an antecedent cycle, IBDisp traps instead of dispatches. Interrrupt can be set from the following sources:

- From microcode with fY = MesaIntRq
- From the IOP (A/IOPIntMP') or from the AI Interface (IntExternal')
- From Interval Timer1 (TimerInt)

The interrupt register is reset by microcode with fY = ClrIntTrap. Interrupt can also be enabled or disabled by microcode with fY = SetIE or fY = ClrIE, respectively.

2.3.3.4 Link Registers

The central processor has eight 4-bit link registers which can be loaded from the low four bits of the control store address. Generally, link registers hold four bits of state information derived directly from the flow of control. Thus, previously determined state information can be easily recalled by dispatching on a link register. Moreover, macroinstructions can share common code at various stages of their execution, and link registers can be used for subroutine call and return structures.

The link register addressed by fX is written with the low nibble of NIAX (which equals NIA. A link register is written when fX is in [0..7] and NIA[7] = 0; that is,

Link[fX] ← NIAX8-11

A link register is ORed into the low nibble of INIA when fX is in [0..7] and NIA[7]= 1, causing a potential 16-way dispatch. Since the link register is designated by an fX function, the fY field is free to specify other condition bits that can be ORed into INIA8-11.

If a preceding microinstruction does not specify a branch or dispatch condition, then the link register is loaded with a constant. However, if the prior instruction contains a branch or dispatch, then the value loaded depends on the outcome of the branch or dispatch. The low four bits of the IB dispatch value can be recorded in this way.

2.3.3.5 Microcode Traps

The two general classes of microcode traps are:

- IB-Refill - occurs as a result of IBDisp; hence, between execution of macroinstructions.
- Error - occurs in any cycle and always traps to location 0 in c1. Error traps have priority over IB-Refill traps and cannot be disabled.

IB-Refill Traps

If an IBDisp is executed and ibPtr ≠ full or MInt = 1, then the ibFront dispatch does not occur; instead, an IB-Refill trap is caused. Specifically, ibPtr is unaffected, INIA4-11 is not modified, and NIA0-3 is set to the 4-bit quantity 0,,1,,MInt,,ibPtr1.

Table 2.9 summarizes the interpretation of IB-Refill trap locations.

Table 2.9. IB-Refill Traps

NIA[0-3]	MInt	IbPtr
4	0	empty
5	0	not empty (i.e., byte or word)
6	1	Empty or full
7	1	Byte or word

Note: If an IB-Refill trap occurs and MInt = 0, then ibPtr cannot equal full.

AlwaysIBDisp does not trap to IB-Refill; a pageCross branch caused by MAR← or an IB-Empty Error trap cancels a potential IB-Refill trap.

Error Traps

Error traps result when one or more predefined error conditions are detected in the central processor. All error traps cause the instruction at microstore location 0 to be executed in c1 by the emulator or Kernel, depending on the error type. Error traps cannot be disabled. Error traps are reset by the ClrIntTrap command, which also resets any pending interrupts.

Table 2.10 lists, in the order of their priority, the error types encoded by Trap0-1 in the Trap Machine.

Note: The error traps, Trap0-1, are read onto X[8-9] respectively with the ReadMisc' or RdIntStat command.

Table 2.10. Error Types

Trap0-1 (X.08-09)	Error Type
0	not used
1	Init trap
2	stackPointer overflow or underflow
3	IB-Empty error

Stack Pointer Overflow or Underflow

If a pop or push is executed with the values of the stackPointer given in Table,2.11 then a trap to location 0 in c1 occurs. However, stackP is still modified.

To improve detection of stack overflow or underflow, multiple pops and pushes can be specified per microinstruction. For example, fXpop (the pop in the fX field), fZpop, and push executed together leave the stackPointer unmodified, yet simulate two pops with respect to stack underflow detection. fXpop with push checks for stack overflow while not moving the stackPointer, and, likewise, push and fZpop check for underflow. Table 2.11 lists the cases.

Table 2.11. Stack Pointer Overflow or Underflow

functions	stackP	Trap is	if stackP is
pop	-1	underflow	0
push	+1	overflow	15
fXpop, push	0	underflow	0
push, fZpop	0	overflow	15
fXpop, fZpop	-1	underflow	0 or 1
fXpop, fZpop, push	0	underflow	0 or 1

If the emulator top-of-stack (TOS) element is kept in an R register and the rest of the stack is in the U registers, and if it is assumed that TOS can always be stored away into the stack, then the values given in the table imply a maximum stack size of 14 words.

IB-Empty Error

If an ←ib, ←ibNA, ←ibLow, or ←ibHigh is executed when ibPtr = empty, then an IB-Empty error trap occurs to location 0 in c1. If the IB-Empty Error occurs in c1, then an MDR← in the next cycle is canceled.

In normal operation, the instruction buffer is guaranteed to have enough (two) operand bytes before a macroinstruction begins executing. However, when the macroprogram counter points to the last word of a page, the buffer is intentionally not refilled by the Emulator refill microcode and the IB-Empty trap can occur, indicating that control has actually proceeded across a page boundary.

If the IB-Empty error occurs in c1, then control transfers to location 0 in the next emulator c1. If the error occurs in c2 or c3, then the hardware requires the execution of one additional emulator click before the trap at location 0. Consequently, an emulator click can intervene between the occurrence of the IB-Empty error in c2 or c3 and the trap code. In particular, if such a click executed an MDR←

with an address that was a function of an IB value read in the previous c2 or c3, then a random memory location can be written.

The instruction buffer is not read during c2 or c3 of a macroinstruction's last click. A memory write with an MAR← or Map← address that is a function of the IB value read in c2 or c3 must not immediately follow an ←ib, ←ibNA, ←ibLow, or ←ibHigh function executed in c2 or c3.

InitTrap

Although InitTrap (Initialization Trap) is grouped with IBEmpty Trap and Stack Pointer Trap and labeled "Error Traps," it is not an error. InitTrap is a signal generated by the MBC mode control logic when it exits the Boot mode and enters the Run mode. The signal is fed to the Trap Machine to cause a trap to location 0.

Note: The trap machine is a 512-word x 8-bit PROM with the following signals:

ClrIntTrap'	Clears error trap which has just been serviced.
Cycle1	Signal from MCB indicaty cycle 1 of MBC state machine.
IBEmptyTrap'	Signifies an IB empty error.
InitTrap'	Initial trap after booting.
StackTrap	Signifies a stak pointer error.
Trap	Traps next address to Bank 0 Location 0.
Trap.0', 1'	Error trap bits enabled onto X bus 08 and 09, respectivcly, by either a ReadMisc or a RdIntStat command. (see Section 2.3.2.5)

2.4 Registers and Data Paths

The subsection titled "Hardware" briefly describes the central processor registers and their interfaces. The subsection titled "Theory of Operation" describes external and internal data paths. The subsection titled "Programmer Interface" provides a detailed register description at the microcode level.

2.4.1 Hardware

Hardware consists of the Arithmetic Logic Unit (ALU), registers, instruction buffer state control, and X and Y bus interfaces. Bus interfaces are described in section 2.5 titled "Mesa Bus Control."

2.4.1.1 Arithmetic Logic Unit

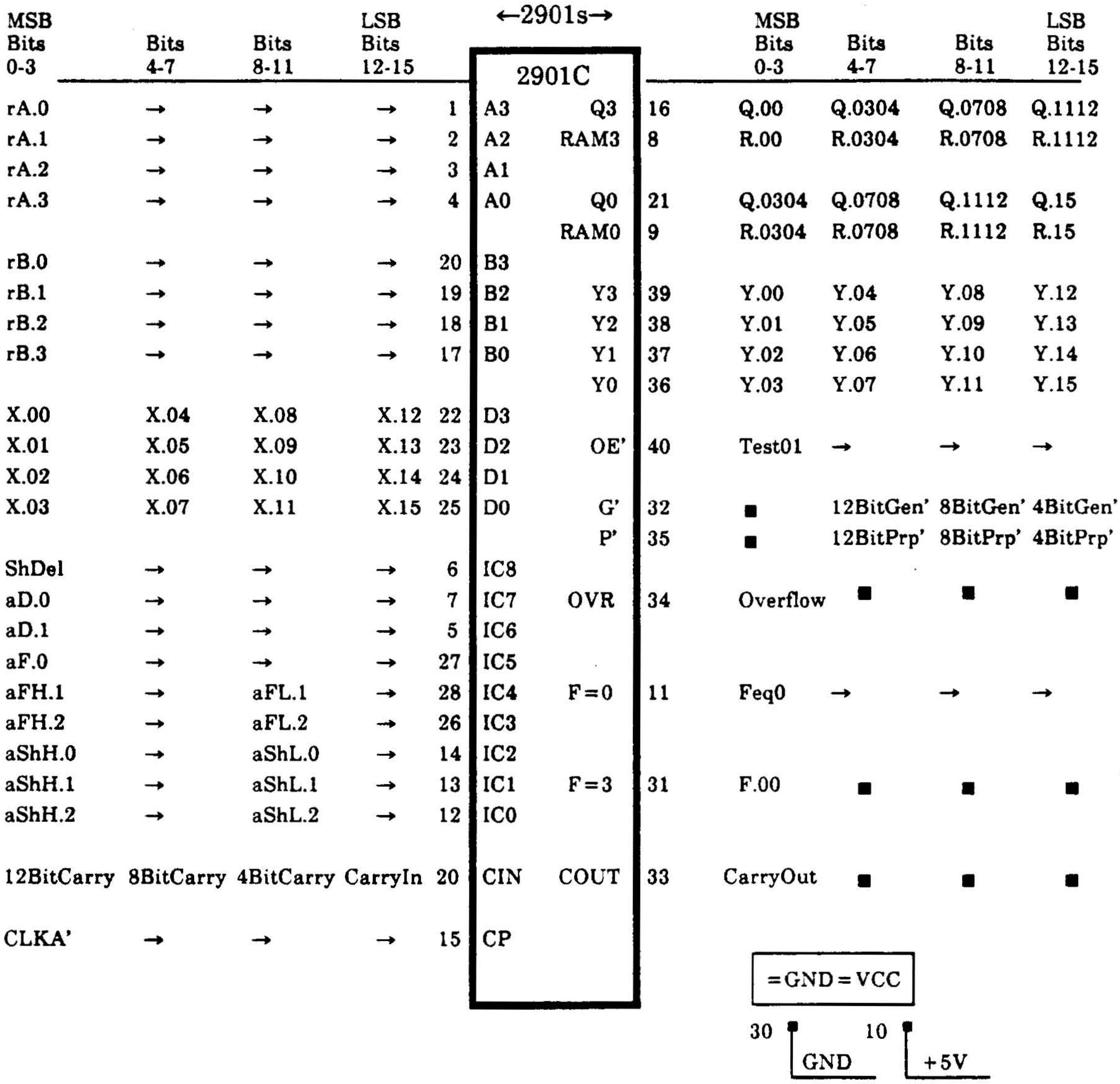
The ALU is implemented with four 2901C bit slice microprocessor chips. For a detailed description of the the 2901C, refer to the Bipolar Microprocessor Logic and Interface 1983 Data Book, Advanced Micro Devices.

Registers on the ALU are the R registers and Q register. The register functions are discussed in more detail in section 2.4.3, titled "Programmer Interface."

R registers make up a 16-word, two-port register file. Output ports are labeled A and B. R registers are the "fast" registers of the central processor, and hold temporaries, memory data and addresses, and arithmetic operands.

The Q register is a 16-bit register which can be written with the ALU output or with its old value single-bit shifted left or right.

Figure 2.10 illustrates the pins and signals for the four 2901C bit slice processors. Table 2.12 describes ALU signals.



←2901s→

MSB Bits 0-3	Bits 4-7	Bits 8-11	LSB Bits 12-15		2901C			MSB Bits 0-3	Bits 4-7	Bits 8-11	LSB Bits 12-15
rA.0	→	→	→	1	A3	Q3	16	Q.00	Q.0304	Q.0708	Q.1112
rA.1	→	→	→	2	A2	RAM3	8	R.00	R.0304	R.0708	R.1112
rA.2	→	→	→	3	A1						
rA.3	→	→	→	4	A0	Q0	21	Q.0304	Q.0708	Q.1112	Q.15
						RAM0	9	R.0304	R.0708	R.1112	R.15
rB.0	→	→	→	20	B3						
rB.1	→	→	→	19	B2	Y3	39	Y.00	Y.04	Y.08	Y.12
rB.2	→	→	→	18	B1	Y2	38	Y.01	Y.05	Y.09	Y.13
rB.3	→	→	→	17	B0	Y1	37	Y.02	Y.06	Y.10	Y.14
						Y0	36	Y.03	Y.07	Y.11	Y.15
X.00	X.04	X.08	X.12	22	D3						
X.01	X.05	X.09	X.13	23	D2	OE'	40	Test01	→	→	→
X.02	X.06	X.10	X.14	24	D1						
X.03	X.07	X.11	X.15	25	D0	G'	32	■	12BitGen'	8BitGen'	4BitGen'
						P'	35	■	12BitPrp'	8BitPrp'	4BitPrp'
ShDel	→	→	→	6	IC8						
aD.0	→	→	→	7	IC7	OVR	34	Overflow	■	■	■
aD.1	→	→	→	5	IC6						
aF.0	→	→	→	27	IC5						
aFH.1	→	aFL.1	→	28	IC4	F=0	11	Feq0	→	→	→
aFH.2	→	aFL.2	→	26	IC3						
aShH.0	→	aShL.0	→	14	IC2						
aShH.1	→	aShL.1	→	13	IC1	F=3	31	F.00	■	■	■
aShH.2	→	aShL.2	→	12	IC0						
12BitCarry	8BitCarry	4BitCarry	CarryIn	20	CIN	COUT	33	CarryOut	■	■	■
CLKA'	→	→	→	15	CP						

■ = GND = VCC

30 GND

10 +5V

Figure 2.10. 2901C pins and signals

Table 2.12. ALU External Signal Description

Signal (AMD signal)	Type	Function
4/8/12BitCarry (CIN)	Input	Carry-in to the ALU, as named.
4/8/12BitGen' (G')	Output	Carry-generate signal of the internal ALU for carry look-ahead.
4/8/12BitPrp' (P')	Output	Carry-propagate signal of the internal ALU for carry look-ahead.
aD0/D1 (IC7-6)	Input	Indicates that aD0 or aD1 is to be deposited in the Q register or in the register stack.
aF0, aFH.1/2 aFL.1/2 (IC5-3)	Input	Designates the function to be performed (aF0, aFH.1/2 or aFL.1/2).
aShH.0-2 aShL.0-2 (IC0-2)	Input	Instruction control lines identifying data source applied to the ALU.
CarryOut (COUT)	Output	Carry-out from internal ALU.
CLKA' (CP)	Input	Clock A input. The Q register and register stack outputs change on the clock low-to-high transition. The clock "low" time is the internal write-enable to the 16 x 4 RAM that constitutes the master latches of the register stack. While the clock is low, the slave latches on the RAM outputs are closed, storing the data previously on the RAM outputs. This scheme allows synchronous master-slave operation of the register stack.
F.00 (F3)	Output	The most significant ALU output bit.
Feq0 (F=0)	Output	Open collector that goes high when all data on the outputs F0-3 are low. In positive logic, F=0 indicates that the result of an ALU operation is 0.
Overflow (OVR)	Output	XOR of the carry-in and carry-out of the MSB of the ALU. At the most significant end of the word, indicates that the result of an arithmetic two's complement operation has overflowed into the sign bit.
Q.0-15 (Q3) R.0-15 (RAM3)	I/O	Shift lines at the MSB of the Q register (Q3) and the register stack (RAM3). Electrically these lines are three-state output connected to TTL inputs internal to the device. If IC6-8 indicates an up shift, then the three-state outputs are enabled; the MSB of the Q register is available on the Q3 pin, and the MSB of the ALU output is available on the RAM3 pin. If IC6-8 indicates a down shift, then the pins are used as data inputs to the MSB of the Q register and RAM.
rA0-3 (A0-3)	Input	Address inputs to the register stack for selection of the register which will have its contents displayed through the A-port.
rB0-3 (B0-3)	Input	Address inputs to the register stack for selection of the register that will have its contents displayed through the B-port. New data can be written into the selected register when the clock goes low.
ShDel (IC8)	Input	Indicates that Sh is to be deposited in the Q register or register stack.
Test01 (OE')		For test. (If OE is high, then Y outputs are off. If OE is low, then Y outputs are active.)
X.00-15 (D0-3)	Input	A 4-bit data field that can be selected as one of the ALU data sources for entering data into the 2901C. D0 is the LSB.
Y.00-15 (Y0-3)	Output	Three-state output lines. If enabled, they display either the four outputs of the ALU or the data on the A-port of the register stack. IC6 = 8 determines the display.

2.4.1.2 Registers

In addition to the ALU registers, the central processor contains the registers briefly described below. Section 2.4.3 discusses the registers in more detail.

U registers make up a 256-word register file which can be written from the Y bus and read onto the X bus. These 16-bit, general purpose, "slow" registers hold a 16-word stack, virtual page addresses, temporaries, counters, and constants.

U registers are situated between main memory and the R registers. They cannot be both read and written in the same cycle, nor can they be used as an operand or destination register in 16-bit ALU arithmetic.

The stackP register is a 4-bit stack pointer that addresses one location from U register bank. The register can be incremented or decremented independently of the 2901. Unlike the U and RH registers, stackP can be read and written in the same cycle.

RH registers, an extension of the R registers of the 2901C, make up the 16 x 8-bit register file located on the X bus. This small memory holds the highest-order memory address bits, and can also be used as general purpose storage for flags, counters, temporaries, and subroutine return pointers.

The pc16 register is a low-order, 1-bit extension of the R register that holds the Mesa emulator's macroprogram counter (PC). pc16 can be used as the byte index of a PC memory address.

The Instruction Buffer registers consist of three 8-bit registers:
IB[0] – holds the even code segment byte
IB[1] – holds the odd code segment bytes
ibFront– shuffles bytes in even/odd, sequential order

Four states enumerate the location of data bytes among the holding registers. The states are indicated by the 2-bit register ibPtr.

Constants that are 4- or 8-bit constants can be placed onto the X bus for use in branching, can be loaded into X bus destination registers, or can be an ALU operand. Constants greater than 8-bit can be preloaded into U registers and, except for timing, are used like normal constants.

Interrupt is a 1-bit control register used to interrupt the contiguous execution of emulator macroinstructions (see 2.3.3.3).

A Link register is one of eight 4-bit registers that holds four bits of state information derived directly from the flow of control.

2.4.1.3 Instruction Buffer State Machine

Instruction Buffer (IB) state control is implemented on 512 by 8 fast ROM. Figure 2.11 illustrates the pins and signals. Table 2.13 summarizes the IB state control interfaces.

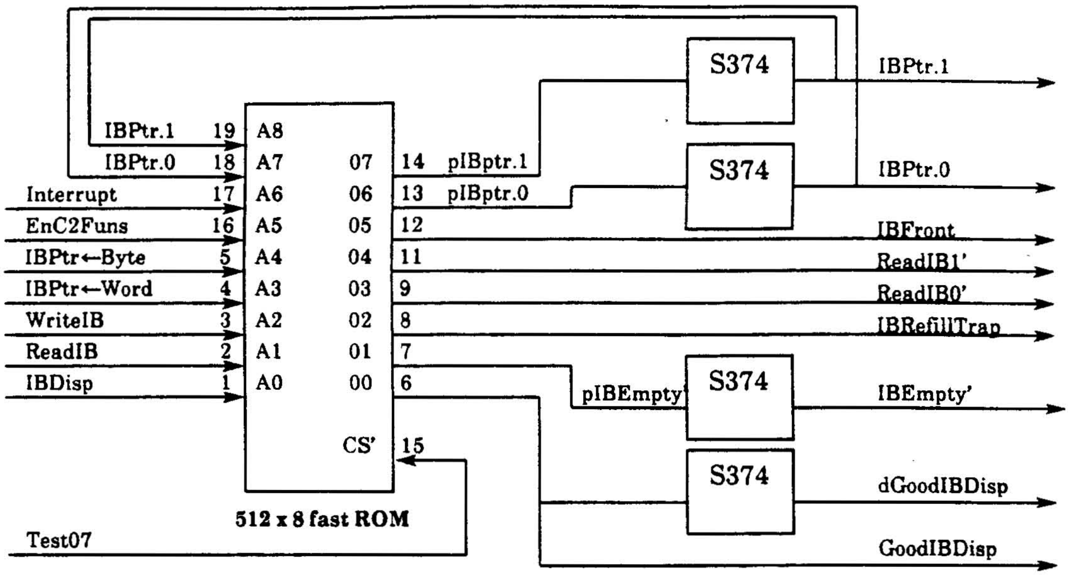


Figure 2.11. IB state control pins and signals

Table 2.13. IB State Interfaces

Interface to:	Signal	Function
ALU Carry & Shift	PageCross	Equals the XOR of pageCarry and aF.2, where pageCarry is the carry out of the low 8 ALU bits.
Branch and Link registers	MapPageCross'	Equals (mem . Cycle1 . pageCross)'
IB registers	1. ReadIB0'/1' 2. Write IBFront	1. Read instruction buffer register 0 or 1. 2. Write to the IBFront registers.
MBC	1. Interrupt 2. MemRef' 3. Cycle1	1. Sent to Mesa processor. Mesa, IOP, and Timer interrupts are grouped into one signal. 2. Memory reference. 3. Cycle 1 of MBC state machine.
MDC	1. IBPtr←Byte 2. IBPtr←Word 3. Read/WriteIB 4. IBDisp 5. XLow←IB'	1. IB pointer gets byte. 2. IB pointer gets word. 3. Read from or write to instruction buffer operations. 4. Instruction buffer dispatch. 5. X bus lower bytes get ibFront, either the full byte or nibbles can be read into the X bus, such that all other X bits are set to zero.
MIR (raw)	Mem	Mem bit. If c1, then MAR←; if c2, then MDR←; if c3, then ←MD.
Next Address register	1. IBPtr.1 2. IBRefillTrap 3. GoodIBDisp	1. Instruction buffer pointer bit 1. 2. Instruction buffer refill trap. 3. Good instruction buffer dispatch.
Stack Pointer register	IBPtr.1/0	Instruction buffer point bits 0 and 1. IBPtr[0:1] is encoded to indicate state of the instruction buffer.
Trap Machine	IBEmptyTrap'	Signals instruction-buffer-empty error.
Enable Cycle2 Function Logic	1. enC2Funs 2. IBEmpty'	1. Enable Cycle2 functions. 2. Instruction buffer empty.
	CLKC'	8 MHz processor clock.

2.4.2 ALU Theory of Operations

Figure 2.12 illustrates the register and ALU data paths and is a reference figure for the subsections that follow.

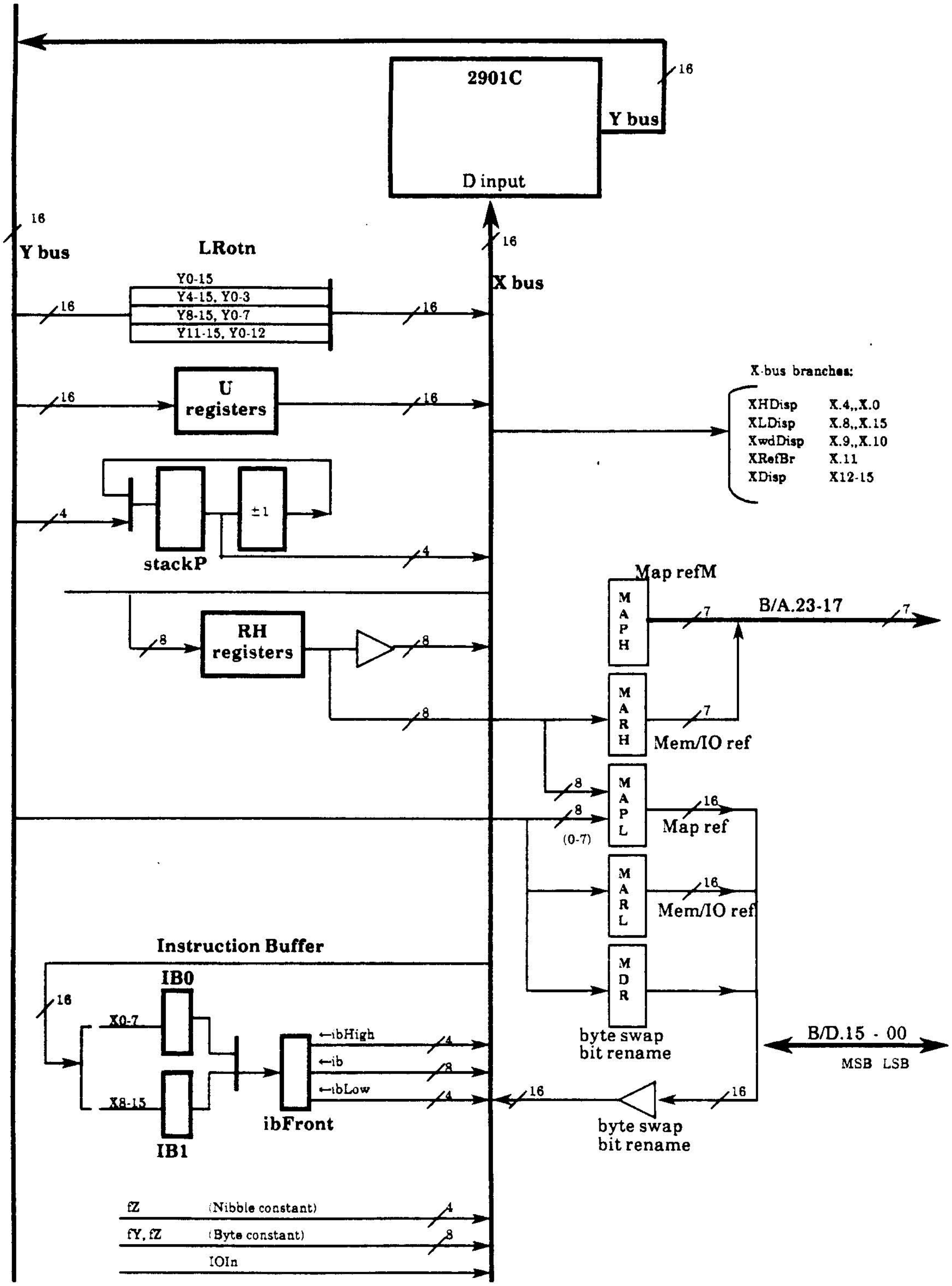


Figure 2.12. Daybreak central processor data paths

2.4.2.1 External Data Paths

The X bus and the Y bus are the two major 16-bit data buses external to the 2901. The YH bus, an 8-bit extension of the Y bus, addresses memory.

The X bus is the major system bus and is connected to multiple drivers and multiple receivers. The X bus sinks are : D inputs of the 2901, RH registers, Instruction Buffer(IB), and branching logic. The X bus sources include: U Registers, RH Registers, Instruction Buffer, Rotator, constants, Stack Pointer, Trap Status, IB Pointer, and Memory Data. The IB and the RH Registers receive data via the X bus ; they can be loaded from memory in one click.

Figure 2.13 illustrates specific external paths to the ALU.

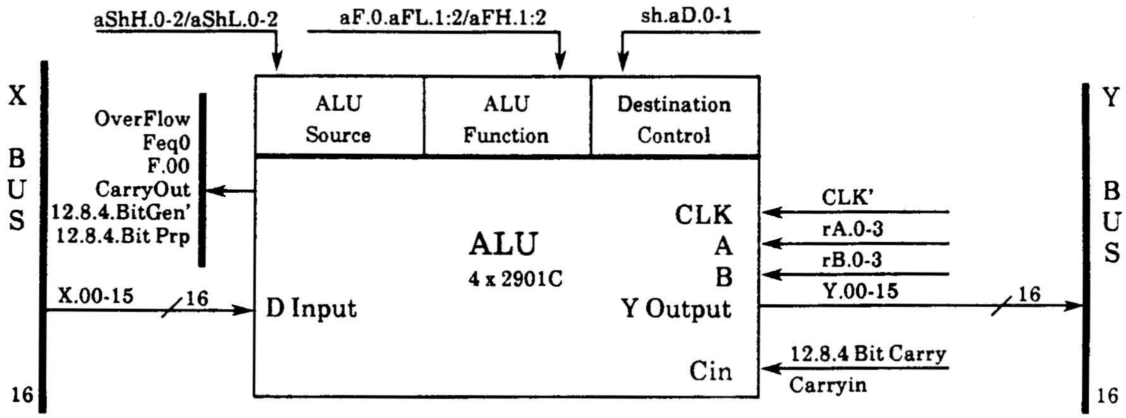


Figure 2.13. ALU external data paths

Data is passed from the Y bus to the X bus via a 4-bit rotator (LRotn). Data can be rotated zero, four, eight, or twelve positions to the left, as specified by the fZ field. A zero rotation allows Y bus data to be placed unaffected onto the X bus.

Eight- or four-bit constants are placed onto the X bus directly from the fY and/or fZ fields. The upper 8 or 12 bits of the X bus are set to zero.

Table 2.14 lists the registers addressable by the central processor and the buses to which they interface.

Table 2.14. Registers Addressed by the Central Processor

Register	Inputs From	Register	Outputs To	
MAR←	YH,,Y	←MD	X	Memory
Map←	YH,,Y			
IB←	X	←ib, ←ibNA	X	Instruction Buffer
		←ibLow, ←ibHigh	X[12-15]	
		~ibPtr	X[10-11]	
RH←	X[8-15]	←RH	X[8-15]	
U←	Y	←U	X	
stackP←	Y[12-15]	~stackP	X	
MDR←	Y	ErrTrap	X[8–9]	
MCtl←	Y	←MStatus	X	Memory

2.4.2.2 Internal Data Paths

Figure 2.14 illustrates internal data paths of the 2901C.

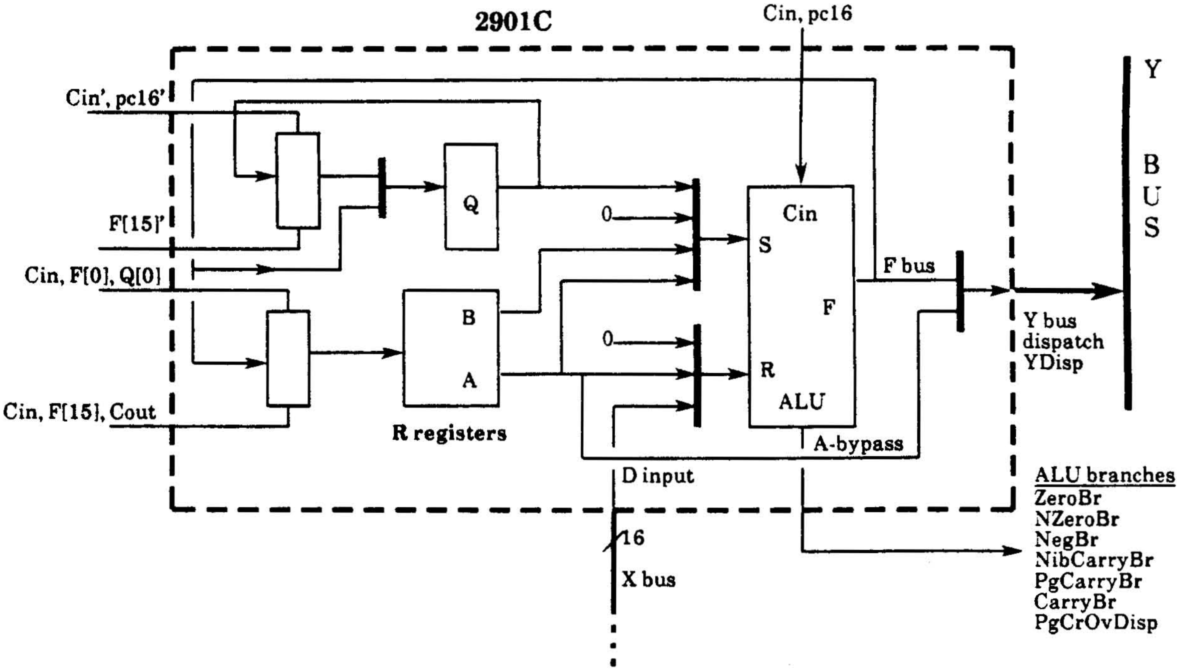


Figure 2.14. ALU internal block diagram

Internally, the 2901 Arithmetic Logic Unit (ALU) has three inputs: R, S, and Carryin (Cin). The R input can be set to the output of the A port, to the value of the X bus, or to zero. The S input can be driven by the output of the A or B ports, by the value of the Q register, or by zero. Cin can be the value of the single-bit Emulator register (pc16) or can be 0 or 1.

The F output of the ALU can be written into an R register, loaded into the Q register, or placed onto the Y bus.

Computations

The 2901 performs three arithmetic and five logical operations which are specified by the ALU-Function (aF) field. Arithmetic follows two's complement conventions. Three of the logical operations are symmetrical with respect to R and S: logical OR, AND, and XOR. The remaining two logical operations complement R: ~R XOR S and ~R AND S.

Figure 2.15 illustrates ALU computations as a function of possible aS and aF values.

	aS									rA=rB=R
aF	Cin	(A,Q)	(A,B)	(0,Q)	(0,B)	(0,A)	(D,A)	(D,Q)	(D,0)	(A,B)
R+S	0	A+Q	A+B	Q	B	A	X+A	X+Q	X	2R
	1	A+Q+1	A+B+1	Q+1	B+1	A+1	X+A+1	X+Q+1	X+1	2R+1
S-R	0	Q-A-1	B-A-1	Q-1	B-1	A-1	A-X-1	Q-X-1	-X-1	-1
	1	Q-A	B-A	Q	B	A	A-X	Q-X	-X	0
R-S	0	A-Q-1	A-B-1	-Q-1	-B-1	-A-1	X-A-1	X-Q-1	X-1	-1
	1	A-Q	B-A	-Q	-B	-A	X-A	X-Q	X	0
R OR S		A OR Q	A OR B	Q	B	A	X OR A	X OR Q	X	R
R AND S		A AND Q	A AND B	0	0	0	X AND A	X AND Q	0	R
~R AND S		~A AND Q	~A AND B	Q	B	A	~X AND A	~X AND Q	0	0
R XOR S		A XOR Q	A XOR B	Q	B	A	X XOR A	X XOR Q	X	0
~R XOR S		~A XOR Q A XOR ~Q	~A XOR B A XOR ~B	~Q	~B	~A	~X XOR A X XOR ~A	~x XOR Q X XOR ~Q	~X	-1

Figure 2.15. ALU operations as a function of aS, aF, and Cin

A-Bypass Mode

A-bypass mode routes the output-port A of the R register file onto the Y bus, which normally receives the F output.

The 2-bit ALU-Destination (aD) field, in combination with a 1-bit value (sh), specifies whether R and/or Q is written and whether F or A-bypass is placed on the Y bus. The sh field is defined by certain functions of the microinstruction word (refer to Figure 2.4). In general, when sh =1, the F output is shifted one bit position before being written back into R or Q. The shift is accomplished inside the 2901 by 3-input multiplexers at the inputs to R and Q. The type of shift is determined by what is shifted into the ends of R or Q.

Table 2.15 lists the type of loading that occurs when sh is concatentated with aD (sh,,aD).

Table 2.15. sh,,aD Actions

sh,,aD	Register Loading	Y bus Loading
000	Q - ALU output	ALU output
001	No register loaded	ALU output
010	R - ALU output	A-bypass value
011	R - ALU output	ALU output

Notes: When A-bypass is used, an R register must be written.
F cannot be written simultaneously to R and Q.

Shift Operations

Figure 2.16 illustrates the two major types of shift operations: double-word shift of F,,Q; and single-word shift of F alone. The two types of shifting, combined with two directions, are named by the four values of aD when sh = 1.

function	aD	fX or fY
Rshift1	1	shift
Lshift1	3	shift
RRot1	1	cycle
LRot1	3	cycle
DARShift1	0	shift
DALShift1	2	shift
?DLShift1	0	cycle
?DRShift1	2	cycle

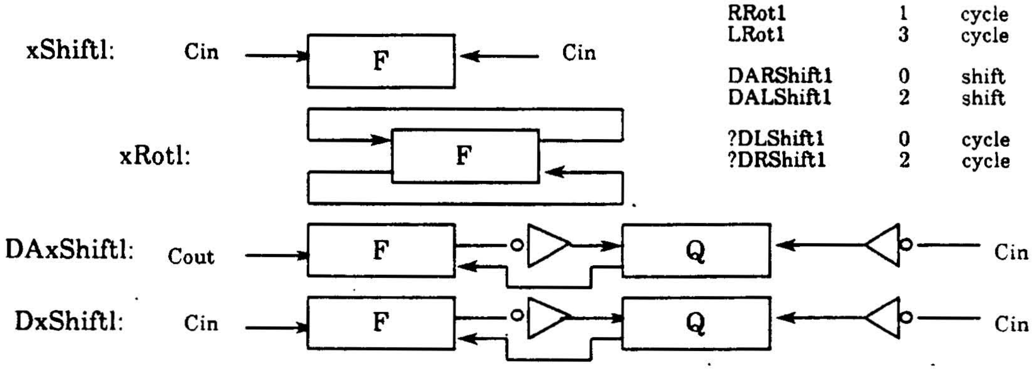


Figure 2.16. CP single-bit shifting

When sh = 1, a single-bit shifting operation is performed on the ALU output and/or Q. For single-word shifts the R register receives the ALU output shifted one bit to the left or right; the Q register is unaffected. The end of F, which is vacated by the shift operation, is replaced by Cin or by the bit shifted out of the opposite side of F (a single-bit rotate).

For double-word shifts, the ALU output and the Q register are shifted together. The low-order bit of the ALU output is "connected" to the Q register high-order bit to form a 32-bit quantity. The high-order bit of F, which is vacated by a right double shift, can be written with Cin or with the Carryout (Cout) of the current ALU computation.

Similarly, the low end of Q is written with the complement of Cin (~Cin if the shift direction is left). Note that the high bit of Q is written with the complement of the low bit of F. A general rule, then, is that shift inputs into Q are *complemented.*

Note: A-bypass cannot be used with single-bit shifts or when loading Q.

Single-bit rotates (LRot1 and RRot1) are applied to the output of the ALU; the results can go only to an R register or to the Q register on double length shifts. Single bit rotates use the fX or fY field.

2.4.2.3 Timing Limitations

The architecture of the central processor allows execution of microinstructions which will not alway properly complete because of slow X bus operands or slow destination registers; that is, certain sources cannot be loaded into certain destinations because the source value is not stable in time. The delay time of the source plus the setup time of the destination must be less than the cycle time of 125 ns. The microcode assembler flags such instructions with a timing violation error.

If the ALU operation uses an X bus operand (aS = D,A, D,Q, D,0), then, depending on the register, the operation may not complete in time. In general, all X bus sources can at least be loaded into an R register, which is a logical operation (aS = D,0, aF = R OR S).

All ALU internal register-to-register operations complete on time. All Y bus destinations can be loaded as a result of any ALU operation that does not use the X bus as an operand (except for the high 12 bits of a U register).

Branching and dispatching have timing different from the basic ALU operations, and a potential statement must meet both conditions. In general, zero, negative, or overflow branching is not possible with an X bus operand.

Figure Description

Figure 2.17 illustrates allowable X bus operations; use the figure to determine whether a microinstruction is legal with respect to X bus timing. The figure lists all possible X bus sources and destinations, X-bus-source-to-X-bus-destination, X bus ALU operands, and X bus branching and dispatching. In the figure,

- Intersections marked with a W (word), b (lower byte), or n (lowest nibble) indicate legal source/destination combinations or branching phrases.
- "X + R" represents the three arithmetic operations: aF = R+S, S–R, R–S.
- "X or R" represents the five logical operations: aF = R OR S; R ANDS; ~RANDS; RXORS; ~RXORS.
- B← implies the loading of an R register; Q← has the same timing.
- pgCross refers to the automatic page cross branch with MAR←.
- pageCross and OVR refer to PgCrOvDisp.

The ALU performs arithmetic at three different speeds, depending on which bits of the result are looked at. Thus, Figure 2.17 has three numbers for arithmetic operations. ALU0-7 are the slowest bits, since they depend on a carry from the lookahead unit. ALU8-11 are faster, since they depend on a ripple carry from the low nibble. ALU12-15 are fastest, since Cin arrives early relative to X bus sources. Thus, the low nibble always has the timing of a corresponding ALU logic operation.

Note: Some +1 or –1 operations do not necessarily imply use of the X bus, but use Cin instead. Thus, R←R+1, NegBr is legal, where R←R+2, NegBr is not.

All arithmetic operations with the ALU internal zero as an operand (aS = 0,Q, 0,B, 0,A, or D,0) complete on time.

	Clock period = 125 ns		X setup	X Source U A	 MD B	 RH C	 Constants ←ib, StkpA D	 LRotn E	 (A OR B) LRotn F	 (A AND B) LRotn G
	X Source Time			79.1 (61.8)	107.72 (103.48)	72.8 (67.28)	57.75 (47.27)	75.3	80.3	106.3 106.3 80.3
X Operations	B←X or R	1	21	W	W	W	W	W	——	——
	B←X or R, Zero Br	2	64 (55)	W		W	W		——	——
	B←X or R, NZeroBr	3	69 (60)							
	B←X or R, NegBr	4	56 (47)	W		W	W		——	——
	B←X or R, YDisp	5	75 (53)				W		——	——
	B←LShift1 (X or R)	6	38	W		W	W	——	——	——
	B←LRot1 (X or R)	7	52 (46)	W		W	W	——	——	——
	MAR←X or R	8	40		——		W	——	——	——
	Map←X or R	9	40		——		W	——	——	——
	MDR←X or R	10	40	W	——	W	W	——	——	——
	U←X or R	11	66	——				——	——	——
	B←X+R	12	53 53 21	n	n	b	W	n	——	——
	B←X+R, ZeroBr	13	88 (79)						——	——
	B←X+R, NegBr	14	85 (76)						——	——
	B←X+R, OVR	15	85 (76)						——	——
	B←X+R, CarryBr	16	83 (74)				W		——	——
	B←X+R, NibCarryBr	17	63 (74)	n		n	n		——	——
	B←X+R, PgCarryBr	18	63 (54)	b		b	b		——	——
	B←X+R, pageCross	19	85 (74)				b		——	——
	MAR←X+R, pgCross	20	71.5 (62.5)	b		b*	b		——	——
	B←X+R, YDisp	21	75 (53)				b		——	——
	B←RShift1 (X+R)	22	53 53 38	n		n	n	——	——	——
	B←RRot1 (X+R)	23	67 (61) 67 (61) 52 (46)	n		n	n	——	——	——
	MAR←X+R	24	69		——		W	——	——	——
	Map←X+R	25	69		——			——	——	——
	MDR←X+R	26	69 69 40	n		——	n	——	——	——
	U←X+R	27	94 94 65	——				——	——	——
	Xbus←X, XDisp	28	45 (23)	W	W*	W	W	W	W	n
	RH←X	29	22	W	W*	W	W	W	W	
	IB←X	30	32 (7)	W	W	W	W	W	W	

W = word b = low byte n = low nibble —— = not applicable
Number in parentheses is attained using higher-speed parts.

Figure 2.17. Allowable X bus operations

2.4.3 Programmer Interface

This section describes the 2901C registers, the rotator, and other registers of the central processor, as illustrated in Figure 2.12.

2.4.3.1 2901C Registers

R Registers

At each cycle, the contents of the R registers, selected by the rA and rB fields of the microinstruction, are available at the respective A and B ports. If rA = rB, then the same data appears at both ports.

If the aD field (ALU destination) of the microinstruction specifies a write back into an R register, then the rB field specifies which R register. At the end of the cycle, register B is written with the ALU output (F) or is written with F shifted one bit.

Q Register

The Q register is implicitly referenced by the aS field of the microinstruction and can be used for double-word shifting (refer to section 2.4.2.2).

2.4.3.2 Rotator

Figure 2.18 illustrates the data paths of the Rotator.

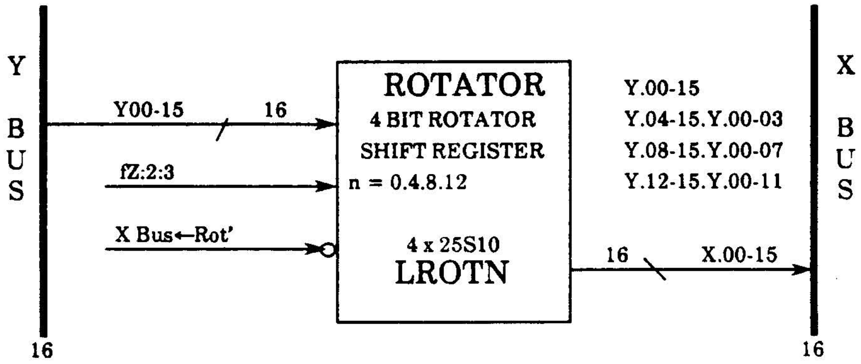


Figure 2.18. Rotator data paths

Data can be rotated zero, four, eight, or twelve positions to the left, as specified by the fZ field.

Zero rotation allows Y bus data to be placed unaffected onto the X bus.

Four-bit rotations (LRot0, LRot4, LRot8, and LRot12) are done on data being moved from the Y bus to the X bus. Four-bit rotations use the fZ field. If the result of the rotation is destined for an R register, then the data must have been placed onto the Y bus via A-bypass. Four-bit rotations are abbreviated LRotn.

2.4.3.3 RH Register

Figure 2.19illustrates the data paths of the RH register.

The RH registers are addressed by the rB field, and since this field names the R register to be written, an RH register can only be written into its corresponding R register (or into the Q register).

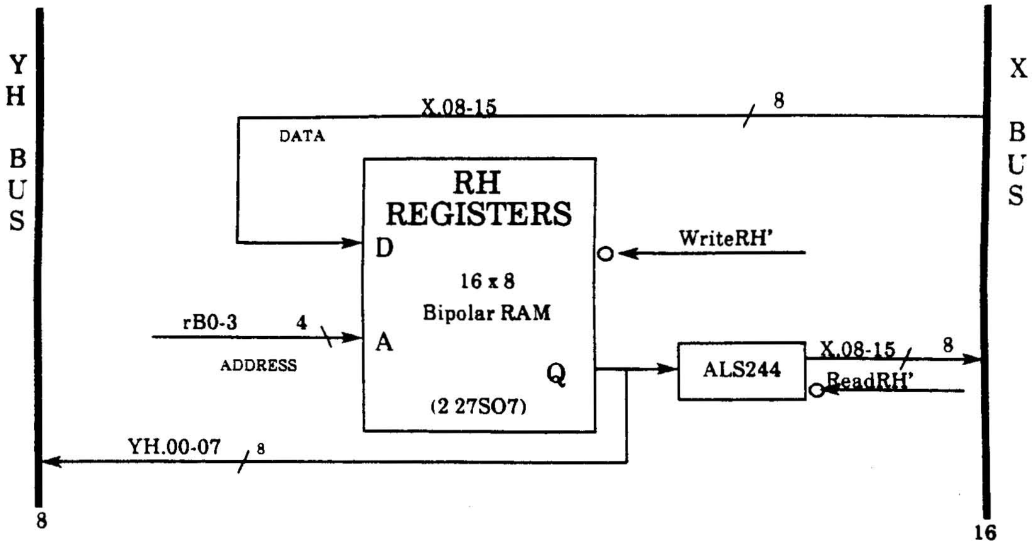


Figure 2.19. RH register data paths

RH registers cannot be both read and written in the same cycle. An RH register is written from the low byte of the X bus when fX = RH←, and is read onto X8-15 when fZ = ←RH. When the RH register is read onto the X bus, the high half of the bus is set to zero.

At every cycle, the 8-bit YH bus is driven with the value of the addressed RH register, thereby supplying the high order memory address bits to memory. However, these bits are only used by the memory if a MAR← or Map← is specified. An RH register cannot be loaded if the microinstruction also executes a MAR← or Map←.

2.4.3.4 pc16 Register

If fX or fZ is Cin←pc16, then the pc16 bit becomes the carry input of the 2901, and pc16 is inverted at the end of the cycle. Thus, Cin←pc16, in combination with ALU addition and subtraction, adjusts the byte program counter (PC,,pc16) to 17 bits.

Since Cin is also the shift ends, Cin←pc16 can shift pc16 into the low-order bit of an R register in one cycle, thereby reconstructing a byte program counter in an R register.

Because of the hardware implementation of the carry input, the fX version of Cin←pc16 must be used when the Cin field of the microinstruction is 0. If Cin = 1, then either the fX or fZ version of Cin←pc16 can be specified.

2.4.3.5 U Register

Figure 2.20 illustrates the data paths of the U register.

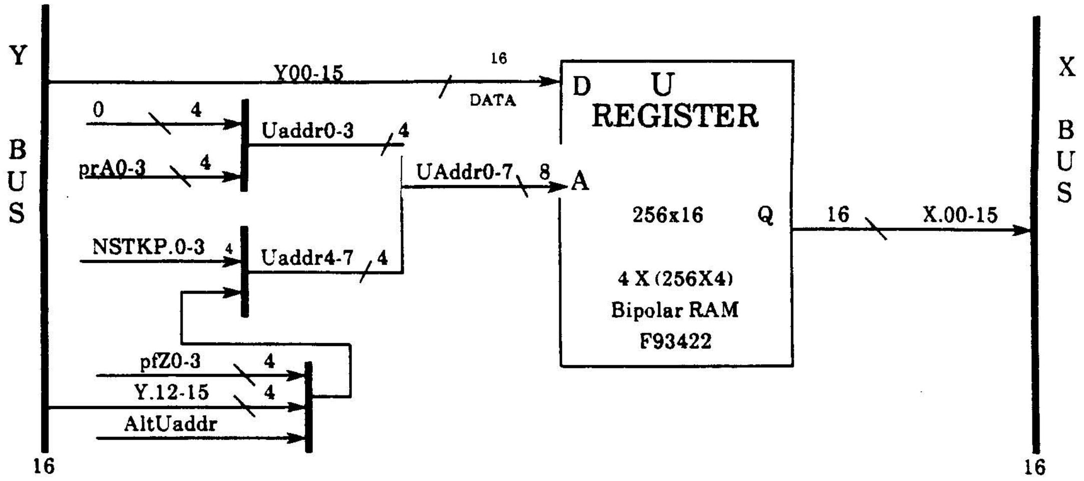


Figure 2.20. U register data paths

U registers are situated between main memory and the R registers. They cannot be both read and written in the same cycle, nor can they be used as an operand or destination register in 16-bit ALU arithmetic.

In addition to the microinstruction fields described below under "addressing modes," U registers are controlled by two other microinstruction fields: enSU and Cin. The enSU bit is 1 for any cycle that either reads or writes a U register. For writing, Cin must be 1; for reading, Cin must be 0. Thus, if a U register is written and the ALU function is addition or subtraction, then the computation executes with Cin = 1. Note that normal two's complement subtraction implies Cin = 1.

Figure 2.21 illustrates three ways to form an 8-bit U register address: normal, stack pointer, and alternate. The addressing modes are described in the following paragraphs.

0 3	4 7	
rA	fZ	Normal
0	stackP	stackPointer
rA	Y.12-15	Alternate

Figure 2.21. U register addressing modes

Normal Addressing Mode

In the normal mode, true when fS[2] = 1, the U register address is defined by the concatenation of the rA and fZ microinstruction fields. In general, a U register can be loaded into any R register, since the rB field defines the write address. However, an arbitrary U register and an arbitrary R register cannot both be ALU operands unless the upper four bits of the U register address equal the R register address. This addressing mechanism partitions the U registers into sixteen, 16-word banks such that, in one cycle, a U register of a bank can be combined only with that bank's corresponding R register.

For reading or writing U registers, the fZ field can specify both a U register address and another function; for example, fZ can take on IOXIn values when fS[2,3] = 3. (Applies also to alternate addressing mode.)

Stack Pointer Addressing Mode

In the stack pointer addressing mode, true when fS[2] = 0, the U register is selected by the 4-bit stackPointer register (stackP) from the low bank; that is, the address is 0,,stackP. The stackP is not explicitly modified with this addressing mode. If an instruction uses this mode and also executes a pop or push function, then the stackP before modification is used to access the U register.

Note: When the stack pointer addressing mode is used, the fZ field can be interpreted either as fZNorm or as a nibble.

Alternate Addressing Mode

The alternate addressing mode provides indirect addressing, and is used when fS[2] = 1 and fZ = AltUaddr for the previously executed microinstruction. In alternate mode, the low nibble of the U address equals the least significant Y bus nibble for the previously executed miroinstruction; that is, the same microinstruction for the AltUaddr. Thus, the U address is rA,,Y[12-15] instead of rA,,fZ.

2.4.3.6 stackP Register

Figure 2.22 illustrates the data paths of the stackP register.

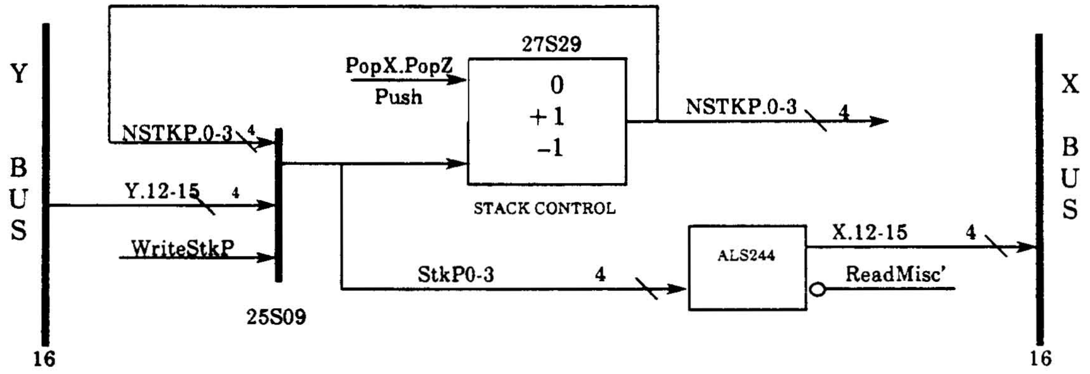


Figure 2.22. stackP data paths

stackP addresses one location from U register bank and can be incremented or decremented independently of the 2901. The pop function decrements (modulo 16) and the push function increments (modulo 16) the stackP at the end of a cycle. Unlike the U and RH registers, the stackP can be read and written in the same cycle.

The stackP is loaded from Y[12-15] with an fY function. One cycle must intervene between a stackP← and a microinstruction that uses the stack pointer addressing mode and that expects a new value. A pop or push can be used in the intervening instruction, and appropriately modifies the value loaded.

Pop and push functions occur throughout the microinstruction function fields to improve checking of stack overflow or underflow. The push function occurs in all three function fields; the pop function occurs in fX and fZ. Because of this arrangement, the stackP does not change when push is specified in the same microinstruction as pop. Multiple pops or pushes can be specified; as long as both are specified, the stackP is unaffected. Multiple pops or pushes in the same instruction do not decrement or increment the stackP by more than one. Multiple pop and push functions check for stack overflow or underflow.

2.4.3.7 Instruction Buffer Registers

The instruction buffer (IB) consists of three 8-bit registers: IB[0], IB[1], and ibFront. IB[0] holds the even code segment byte; IB[1] holds the odd code segment byte. The bytes are shuffled through ibFront in even/odd, sequential order.

Figure 2.23 illustrates the data paths of the instruction buffer.

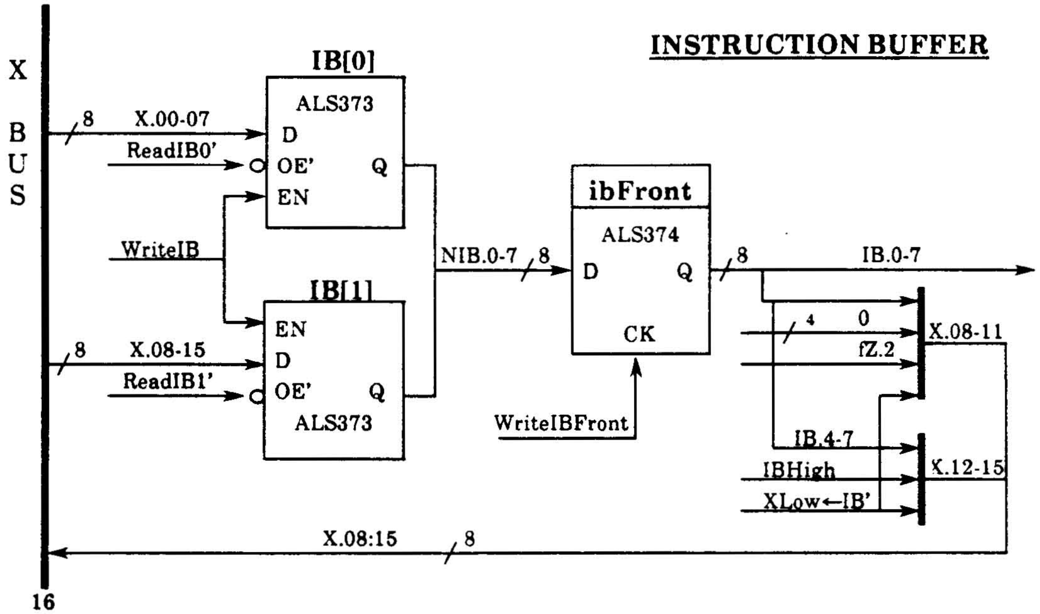
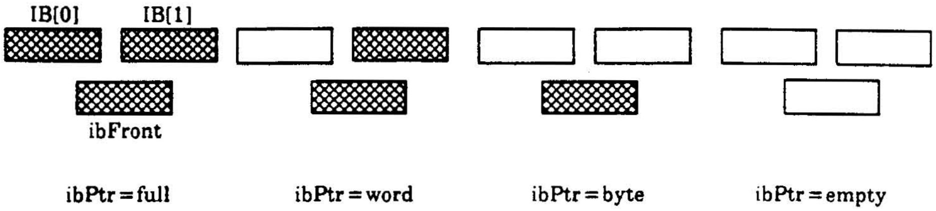


Figure 2.23. Instruction buffer data paths

Four states enumerate the location of data bytes among the holding registers. The states are indicated by the 2-bit register ibPtr.

Figure 2.24 illustrates the instruction buffer states. Cross-hatching indicates the position of the data bytes.

The instruction buffer holds up to three Emulator macroinstructions or data bytes, in a first-in, first-out queue. Data loaded into the IB from the X bus can be read back onto the X bus, or can be used to



State Name	Bytes in IB	ibPtr
full	3	2
word	2	3
byte	1	1
empty	0	0

Figure 2.24. Instruction buffer states

define a 256-way dispatch in control store. The IB is loaded by special emulator "refill" microcode; the actual control of the registers is accomplished by a hardware state machine.

Operand Bytes for Macroinstructions

The Mesa Emulator maintains the instruction buffer in such a way that macroinstructions always find the necessary code segment operands there.

In the instruction buffer the 256-way dispatch is made on the next macroinstruction to be executed. The dispatch (IBDisp) occurs in c2 so that the next macroinstruction begins in c1, thereby adjoining the previous one.

When IBDisp is executed and the buffer is not full, a microcode trap occurs, and the refill microcode loads the buffer with more bytes from memory. If an IBDisp is executed and an interrupt (MInt = 1) is pending, then special interrupt trap (IB-Refill) microcode runs instead of the refill microcode.

The minimum number of bytes in the buffer required to prevent an IB-Refill trap is three (the maximum size of a Mesa macroinstruction). A trap occurs only between the execution of macroinstructions. If the buffer requires two bytes, then the refill code completes in one click. If four bytes are needed, then the refill code completes in two clicks. Because the buffer is small, the only bytecodes that do not result in an IB-Refill trap are single-byte opcodes executed from even memory locations. Since the IB-Refill trap runs at memory speed, operand bytes are efficiently supplied to the macroinstruction.

Microinstruction Functions for the Instruction Buffer

Eight microinstruction functions affect the IB. In general, the functions maintain the original even/odd byte ordering while updating ibPtr and ibFront. Table 2.16 lists the functions and their effect on ibPtr, ibFront, and the X bus.

Table 2.16. Effects of IB–Related Microinstruction Functions

Function	New ibPtr	New ibFront	X bus ←
←ib	ibPtr-1	IF ibPtr[1]=0 THEN IB[0] ELSE IB[1]	0,,ibFront
←ibNA	unchanged	unchanged	0,,ibFront
←ibHigh	unchanged	unchanged	0,,ibFront[0-3]
←ibLow	unchanged	unchanged	0,,ibFront[4-7]
IBDisp	ibPtr-1	IB[ibPtr[1]]	unaffected
AlwaysIBDisp	ibPtr-1	IB[ibPtr[1]]	unaffected
IB←	IF empty THEN word ELSE full	IF ibPtr=empty THEN X[0-7] ELSE unchanged	unaffected
IB←, IbPtr←1	IF empty THEN byte ELSE full	IF ibPtr=empty THEN X[8-15] ELSE unchanged	unaffected
IbPtr←0	word	IB[0]	unaffected
IbPtr←1	byte	IB[1]	unaffected
←ErrnIBnStkp	unchanged	unchanged	X[10-11]←~ibPtr

Operating modes for the microinstruction functions listed in the table are described below.

Load. The IB is loaded from the X bus. The high-order, even byte is written into IB[0]; the low-order, odd byte is written into IB[1]. If the buffer is empty, then the X bus byte passes through IB[0] or IB[1] and is loaded directly into ibFront in one cycle. Thus, data can be used immediately in the cycle following the IB load.

Write. The IB write operation defaults to writing ibFront with X0-7. However, if IbPtr←1 is coincident with IB←, then ibFront is written with X8-15, thereby discarding the even data byte. If one or two bytes are in the buffer, then IB[0] and IB[1] are loaded, and no feed-through into ibFront occurs.

Read. ibFront can be read onto the X bus. When ←ib or ←ibNA is specified, ibFront is placed onto X8-15, and the high byte of the X bus is set to zero.

The basic read can be varied. With the ←ibHigh function, ibFront[0-3] is placed onto X12-15. Analogously, ←ibLow places ibFront[4-7] onto X12-15. In both cases the upper 12 bits of the X bus are set to zero.

Execution. When ←ib is executed, a funneling process occurs. ibFront is loaded with the next byte from either IB[0] or IB[1], and ibPtr is "decremented" by one; that is, ibPtr is gray-code decremented: 2, 3, 1, and then 0. Thus, the low order bit of ibPtr divides the values of ibPtr into two classes with respect to refill: empty and not empty.

Note: The execution scheme equates the empty and full states; however, the buffer is not full when the IB-Refill trap occurs.

Microcode. Several of the microcode functions have no effect on the state of the buffer. The ←ibNA function (reads the IB without advancing ibPtr), ←ibHigh, and ←ibLow do not change ibPtr.

As with the RH and U registers, simultaneous read and write of IB is not possible. Therefore, the combination of IB← and ←ib in the same cycle does nothing.

The functions IBPtr←0 and IBPtr←1, when used alone, merely load ibFront from IB[0] or IB[1], respectively. The functions typically occur in the cycle after the IB has been loaded with a jump-target bytecode, thereby selecting the even or odd destination opcode.

The complement of ibPtr can be read onto X[12-13] with the ←ErrnIBnStkp function.

2.4.3.8 Constants

Figure 2.25 illustrates the data paths for constants.

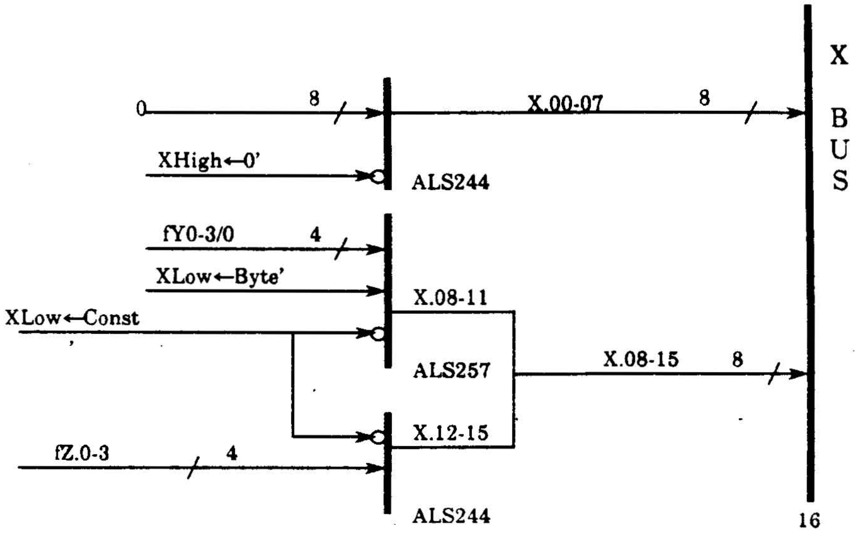


Figure 2.25. Constants data paths

Four-bit constants (Nibble) use the fZ microinstruction field; 8-bit constants (byte) use the fY and fZ field. The upper 12 or 8 bits, respectively, are zeroed.

Larger constants can be preloaded into U registers and used like normal constants. Zero is available inside the ALU and does not use the X bus. ALU "+1" and "-1" operations are also possible without the X bus, since they are an artifact of Cin.

Sixteen-bit constants with identical halves can be constructed in two cycles instead of the three normally required in the general case.

2.5 System Memory Addressing

The central processor sends memory addresses to the system memory controller by way of the Mesa bus. A custom gate array performs bus controller functions for the Mesa processor board. This section describes memory addressing and Mesa bus control.

Figure 2.26 illustrates data paths from the MPB through the Mesa bus to main memory.

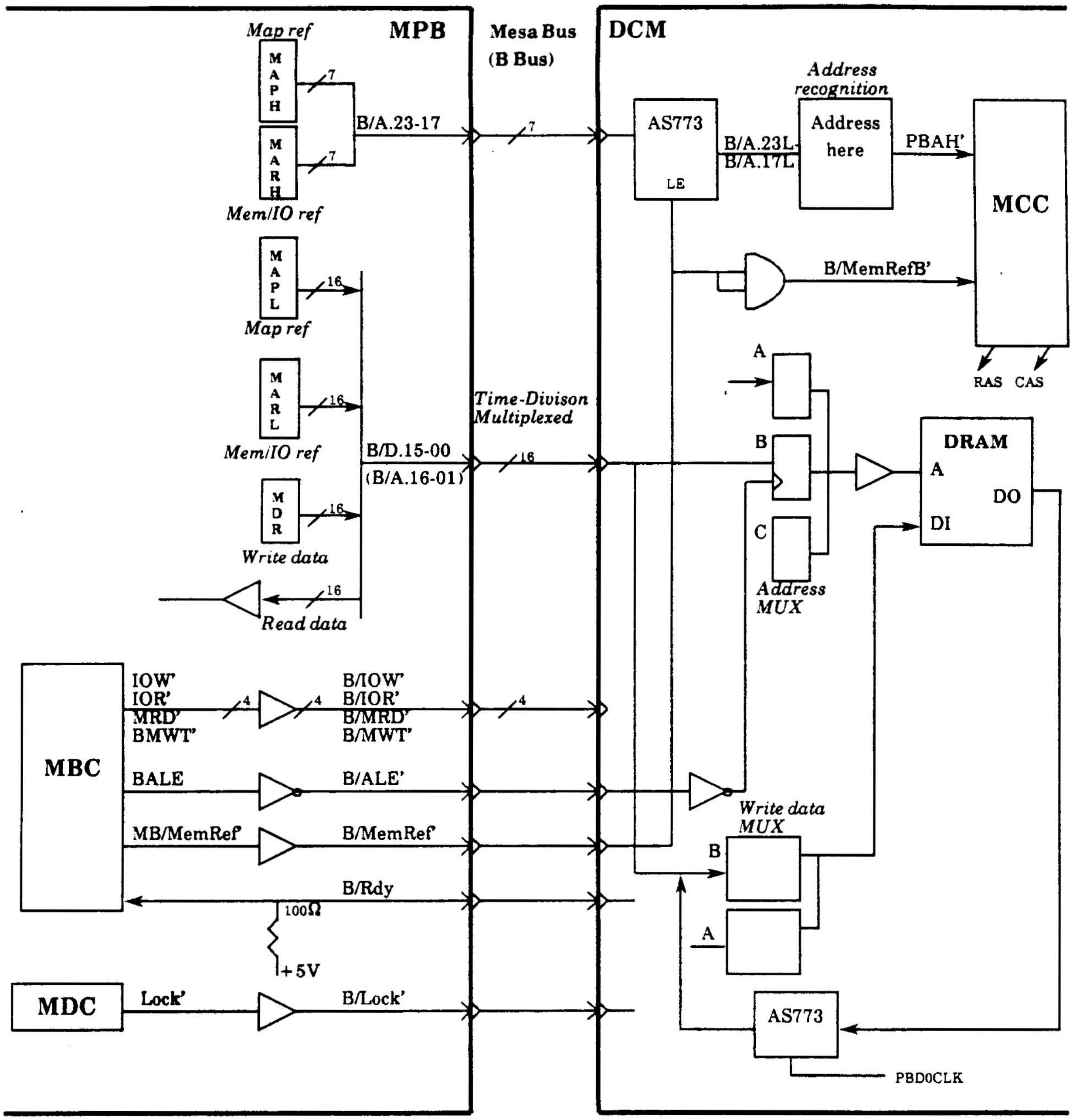


Figure 2.26. Main memory addressing

2.5.1 Hardware

This subsection describes the Mesa bus hardware, principally the Mesa bus controller. Associated hardware is the timer and miscellaneous logic. The 80186 or A bus and A bus control logic is described in "Dove IOP Board Technical Reference Manual."

Bus interfaces to the backplane are listed in the Section 2.1.2 titled "General Board Hardware."

2.5.1.1 Mesa Bus Controller Chip

The Mesa Bus Controller (MBC) is a 68-pin gate array chip. Figure 2.27 illustrates the pins and signals for the chip. Table 2.17 describes the MBC signals.

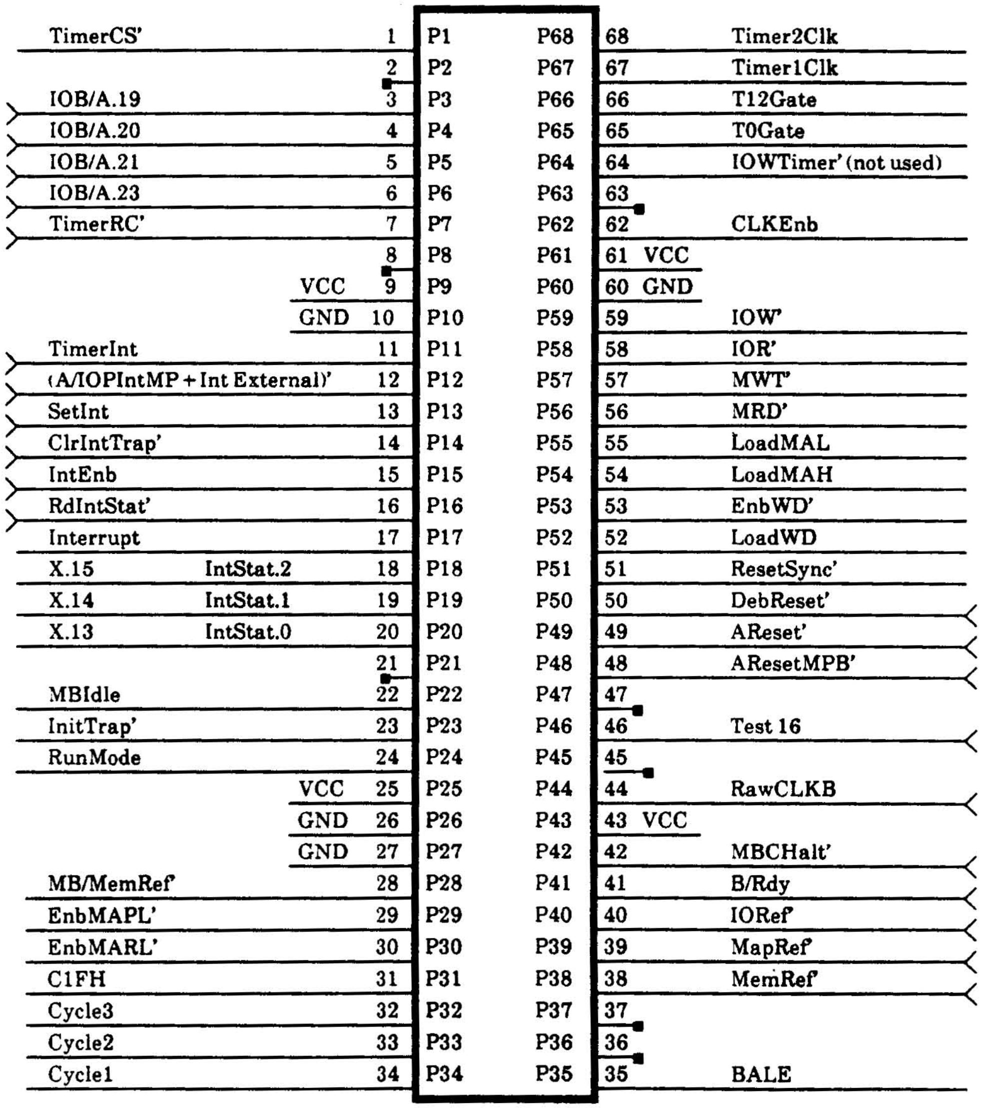


Figure 2.27. MBC pins and signals

Table 2.17. MBC Signal Description

Signal	**Full Name.** Function
(A/IOPIntMP +IntExternal)'	**IOP Interrupt Mesa Processor** or External Interrupt from Artifical Intelligence interface.
AReset'	**Reset.** (includes power up).
AResetMPB'	**Reset MPB.** Resets MPB after request through software.
IOB/A.19,20, 21,23	**Mesa Bus Address.** 19, 20, 21, and 23 latched up and decoded, as follows: B/A.23 B/A.21 B/A.20 B/A.19 Command 1 0 0 X Timer Chip select 1 0 1 0 Clear T0 Gate 1 0 1 1 Set T0 Gate 1 1 0 0 Clear T1-T2 Gate 1 1 0 1 Set T1-T2 Gate
BALE	**B Bus Address Latch Enable.**
B/Rdy	**B bus Ready.** From the DCM; indicates that the B bus is ready for data transfer.
C1FH	**Cycle One First Half.** Goes into effect during the first half of cycle 1.
CLKEnb	**Clock Enable.** Enables 8 MHz system clock.
ClrIntTrap'	**Clear Interrupt Trap.** Clears interrupt.
Cycle1/2/3	Corresponds to cycles 1, 2, and 3 of MBC state machine.
DebReset'	**Debugger Reset.** Used with Burdock to reset the Mesa processor.
EnbMAPL'	**Enable Map Address Low 0-15.** Enables low map address.
EnbMARL'	**Enable Memory Address Low 0-15.** Enables low memory address.
EnbWD'	**Enable Write Data.** Enables the write data on B bus.
IntEnb	**Interrupt Enable.** Signal from the decoder chip (MDC) to enable Interrupt.
Interrupt	**Interrupt.** Output of Interrupt Register.
InitTrap'	**Initial Trapping.** Generated after the initial booting procedure. Traps the first microcode instruction.
IOR'	**I/O Read.** Generated by MBC; reads the timer on the MPB board.
IORef'	**I/O Reference.** From the decoder chip (MDC) to indicate an I/O reference instruction.
IOW'	**I/O Write.** Writes to the Mesa bus I/O, including Timer.
LoadMAH	**Load Memory Address High.** Loads the high order memory address 17-23.
LoadMAL	**Load Memory Address Low.** Loads the low order memory address 00-15
LoadWD	**Load Write Data.** Loads write data from Y bus into data register.
MapRef'	**Map Reference.** From MDC to inform the MBC that a map reference is occurring.
MBCHalt'	**Halt.** From the IOP to stop the MBC; occurs at the end of cycle 2 or at the beginning of cycle 3 when bus is idle.

- more -

Table 2.17. MBC Signal Description (continued)

Signal	Full Name. Function
MBIdle	**Mesa Bus Idle.** Active when B bus is not performing a memory reference, an I/O reference, or a Map reference transaction. MBC can only be halted when MBIdle is active.
MB/MemRef'	**Mesa Bus Memory Reference.** Sent to the DCM to indicate that Mesa bus is requesting a memory or map reference.
MemRef'	**Memory Reference.** From MDC to signal a memory reference.
MRD'	**Memory Read.**
MWT'	**Memory Write.**
RawCLKB	**Raw Clock Buffered.**
RdIntStat'	**Read Interrupt Status.** Enables interrupt status bits 0-2 onto X bus 13-15, respectively. Interrupt status bit 0 = Mesa interrupt; bit 1 = IOP interupt or External Interrupt from AI Interface (if used); bit 2 = timer interrupt.
ResetSync'	**Reset Synchronized.** Generated from three resets that are synchronized with the system clock.
RunMode	**Run Mode.** Indicates that the Mesa Processor is operating in the Run mode.
SetInt	**Set Interrupt.** Sets an interrupt from the decoder chip.
T0Gate	**Timer 0 Gate.** Enables timer 0 to begin counting. Note: First clock after gate goes active loads the count.
T12Gate	**Timer 1 and 2 Gate.** Enables timer 1 and 2 to begin counting. Note: as above.
Test 16	For testing.
Timer1Clk	Clock input to Timer counter 0 and 1.
Timer2Clk	Clock input to Timer counter 2.
TimerCS'	**Timer Chip Select.**
TimerInt	**Timer Interrupt.** An interrupt from timer 0 after one cycle.
TimerRC'	**Timer 1 Ripple Carry.** Derivation of clock input to timer 2.
X.13:15	When RdIntStat is true, interrupt status bits 0-2 are enabled onto X bus 13-15, as follows: X.13 - IntStat.0, Mesa Interrupt X.14 - IntStat.1, IOP or AI Interrupt X.15 - IntStat.2, Timer Interrupt

2.5.1.2 Control Logic

Figure 2.28 illustrates the Mesa bus interface and control logic. Table 2.18 describes the signals shown in the figure.

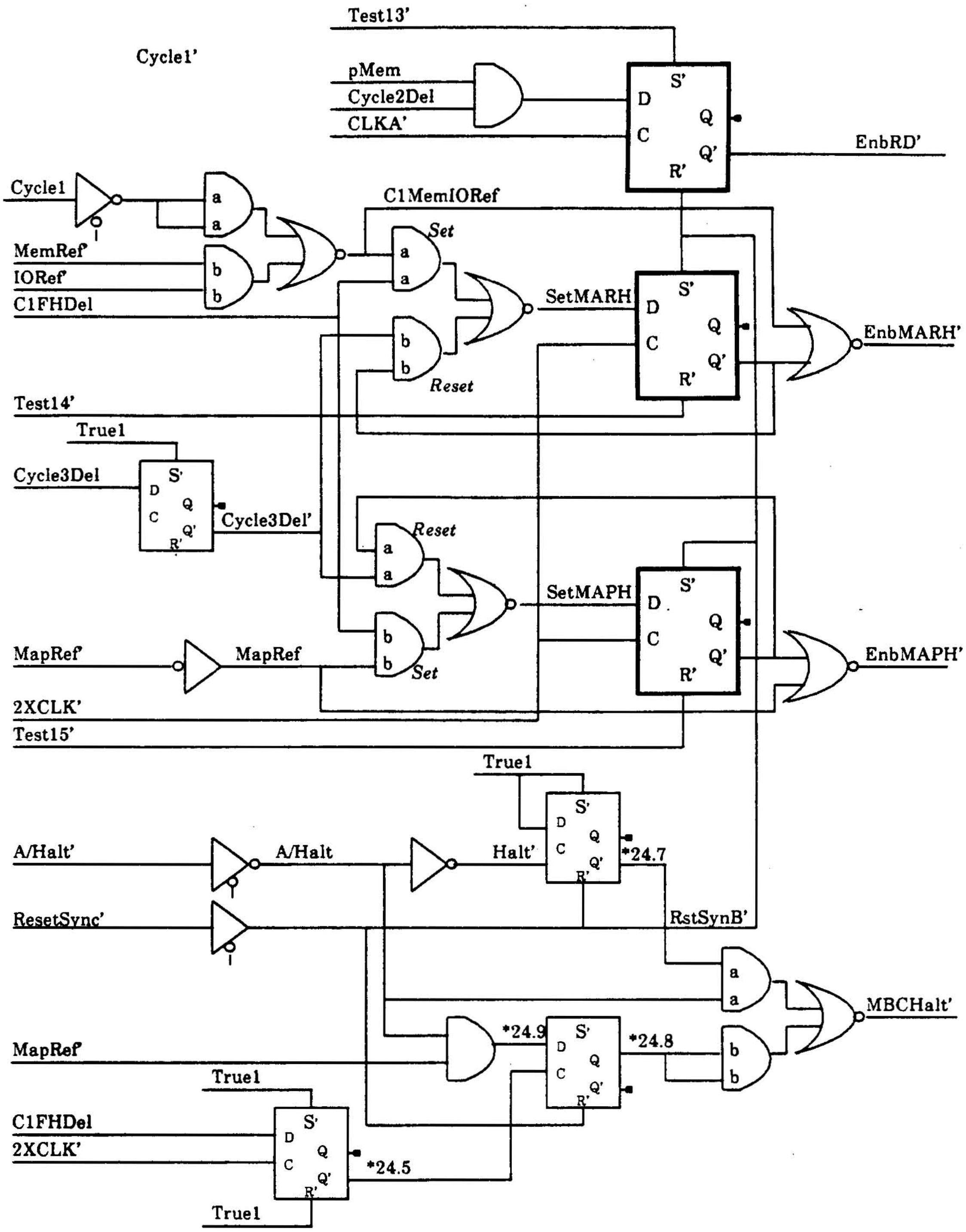


Figure 2.28. Mesa bus logic interface and control

Table 2.18. B Bus Interface and Control Signals

Signal	Function
C1FH	From MBC. Goes active to indicate first half of MBC state machine cycle 1.
Cycle2	From MBC. Goes active to indicate cycle 2 of MBC state machine.
CLK, CLKC', 2XCLK'	From MPB clock generation.
EnbMAPH'	To address path logic. Enables high order (17-23) of MAP Address register.
EnbMARH'	To address path logic. Enables high order (17-23) of Memory Address register.
EnbRD'	To data path logic. Enables ReadData from B bus onto X bus. This signal is activated at the end of cycle 2 if the cycle 3 "mem" bit is predicted to be true (pmem = 1) through the instruction pipeline, signifying a memory read cycle.
IORef'	From MDC to MBC to indicate an I/O reference.
LoadMAL	From MBC. Loads low order (00-15) of both Memory and Map Address registers.
LoadWD	From MBC. Enables the CLK to load WriteData from Y bus into Write Data register.
MapRef'	From MDC to MBC to indicate a map reference.
MemRef'	To MBC to indicate a memory reference.
ResetSync'	From MBC. Synchronized reset signal.
Test13'/14'/15'	For testing only.

2.5.1.3 Memory Address Interfaces

Figure 2.29 illustrates in detail the memory address registers shown in Figure 2.26.

The figure also illustrates the interfaces between the Mesa bus and the central processor X and Y buses.

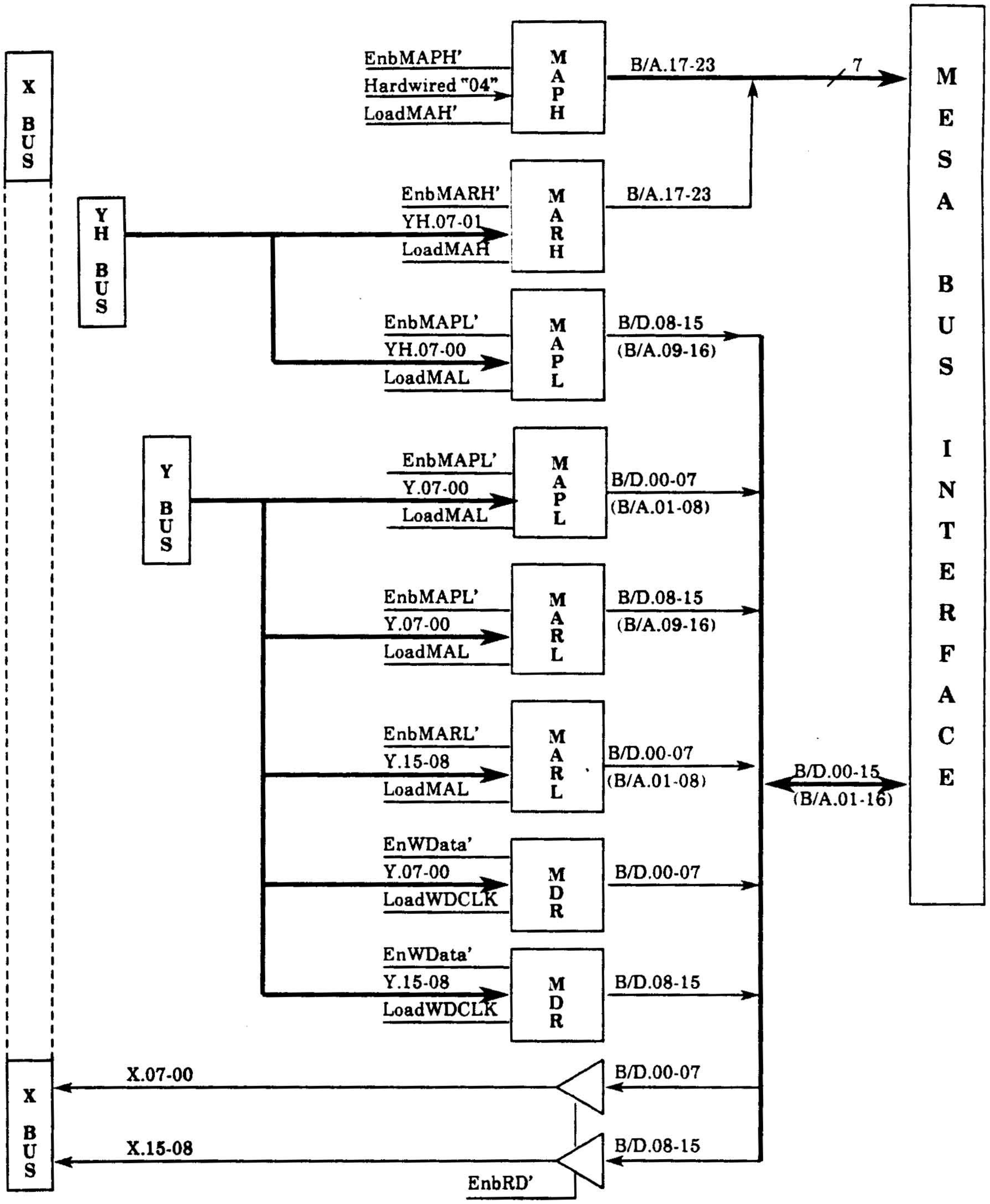


Figure 2.29. Mesa bus interface address/data paths

2.5.1.4 Timer

The process timeout and interval timer is an Intel 8254-2 interval timer/counter. Figure 2.30 illustrates the timer. Table 2.19 describes the signals.

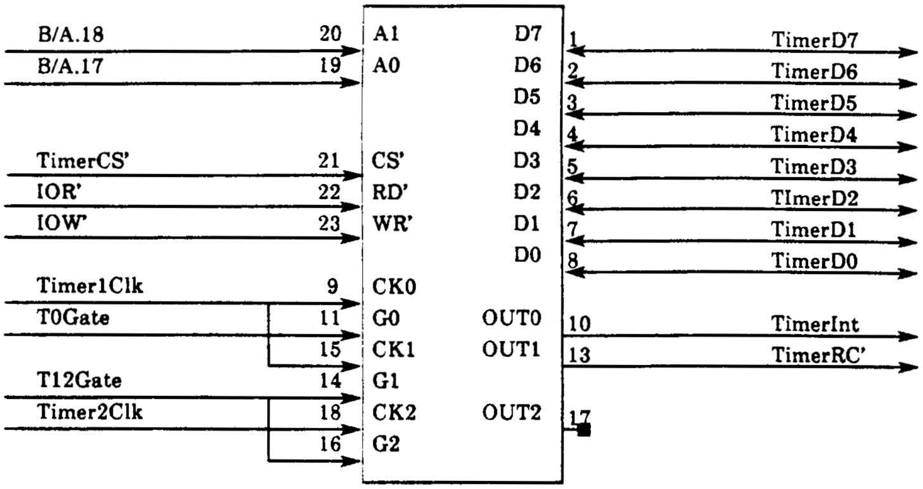


Figure 2.30. Process timeout and interval timer pins and signals

Table 2.19. Timer Signal Description

Signal (Intel Signal Name)	Type	Function
B/A.18,17 (A1, A0)	Input	Selects counter or control register, as follows: A1 A0 Selects 0 0 Counter 0 0 1 Counter 1 1 0 Counter 2 1 1 Control Word Register
Timer1Clk	Input	Clock (period = 16 ms) input of counters 0 and 1. Note: First clock after gate goes active loads count into counting element.
Timer2Clk	Input	Clock input of counter 2. Derived from ripple carry (RC) of counter 1. Note: as above.
TimerCS' (CS')	Input	Chip Select: enables RD' and WR'; otherwise ignored.
TimerD0-7 (D0-7)	I/O	Tri-state data bus lines, connected to system data bus.
T0Gate, T12Gate (G0-2)	Input	Enable corresponding counters
TimerInt (OUT0)	Output	Output of counter 0
TimerRC'	Output	Output of counter 1
IOR' (RD')	Input	Low during CPU read I/O
IOW' (WR')	Input	Low during CPU write I/O

2.5.2 Theory of Operations: Mesa Bus Controller

This section describes the functions of the Mesa bus controller; that is, state machine control of memory addressing, interrupt control, mode control, and timer control.

Figure 2.31 illustrates the functional blocks of the MBC chip.

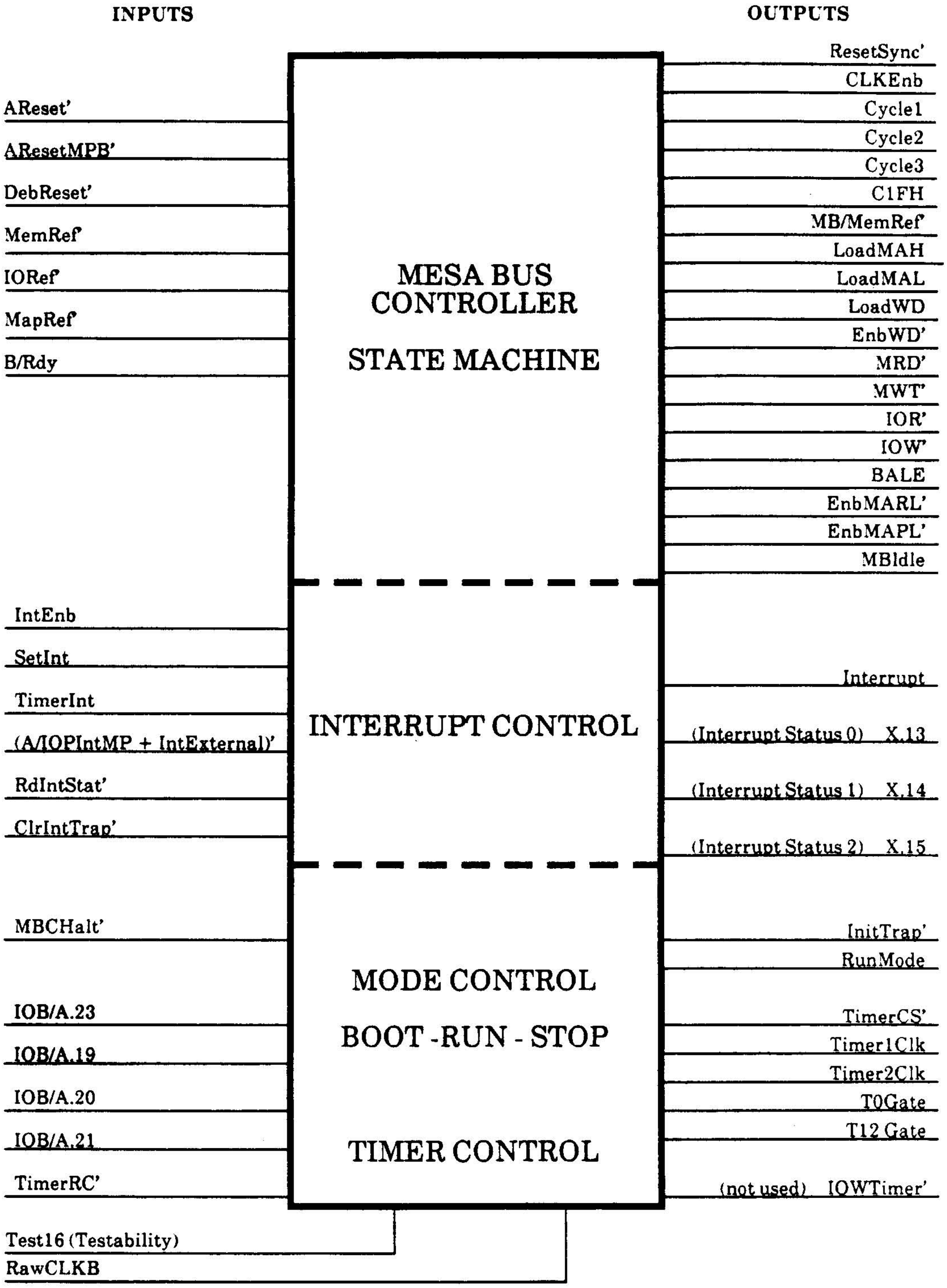


Figure 2.31. MBC functional block diagram

2.5.2.1 States of the State Machine

The MBC state machine is best described with reference to the Memory Reference Timing Diagram, Figure 2.32.

All timing is referenced to the rising edge of 2xCLK' unless otherwise specified.

When a signal is stated to be set or reset, it is set or reset at the end of the clock cycle. On the other hand, when a signal is said to be generated, it is generated during the clock cycle.

Note also that some memory bus control signals are generated outside the MBC chip, because of timing limitations.

Figure 2.33 illustrates state transitions of the Mesa bus controller. The number at the top of each block is a state number for convenient reference in the text and in the flow diagrams that further illustrate the states of the state machine.

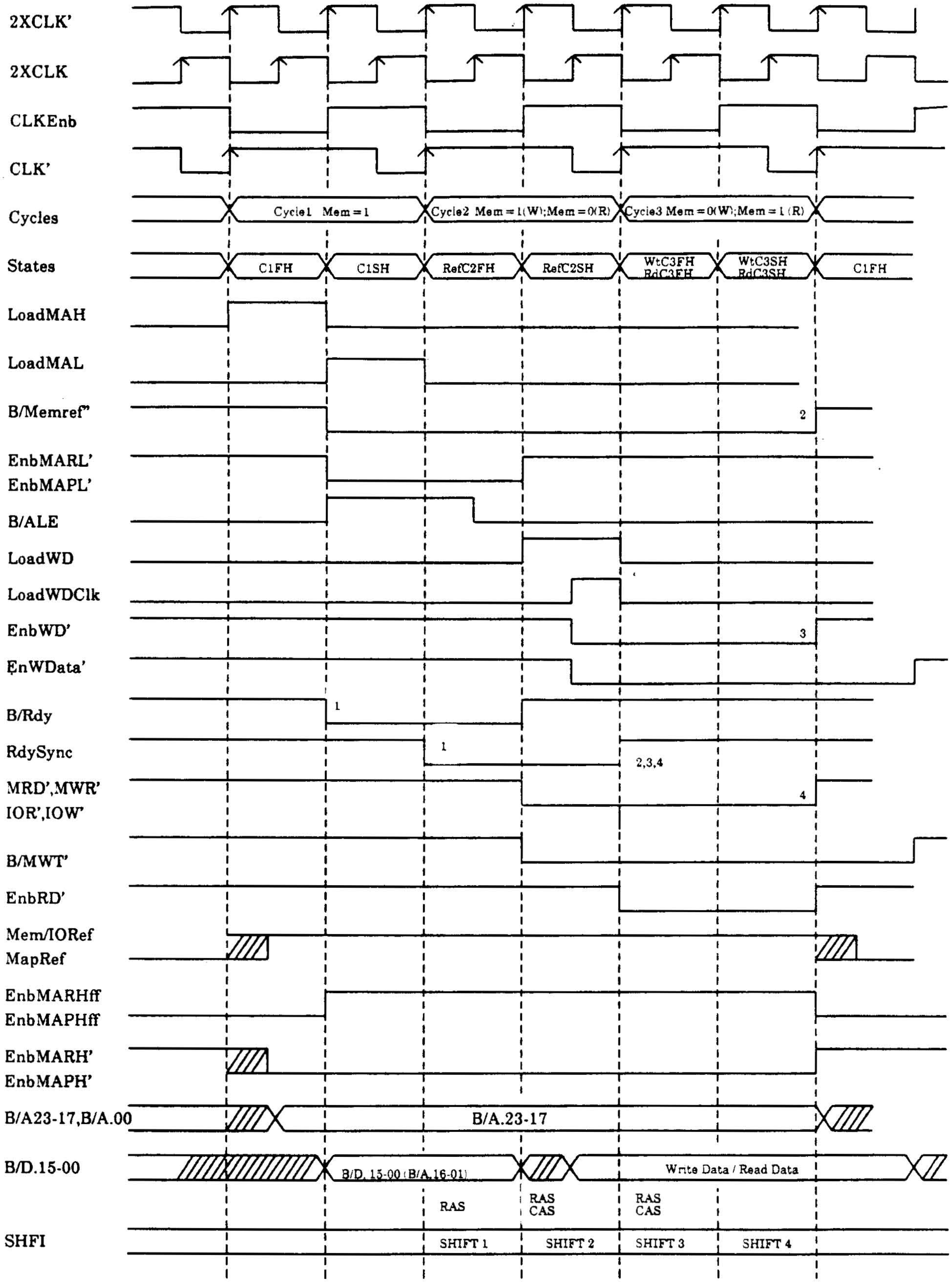


Figure 2.32. State machine memory reference timing

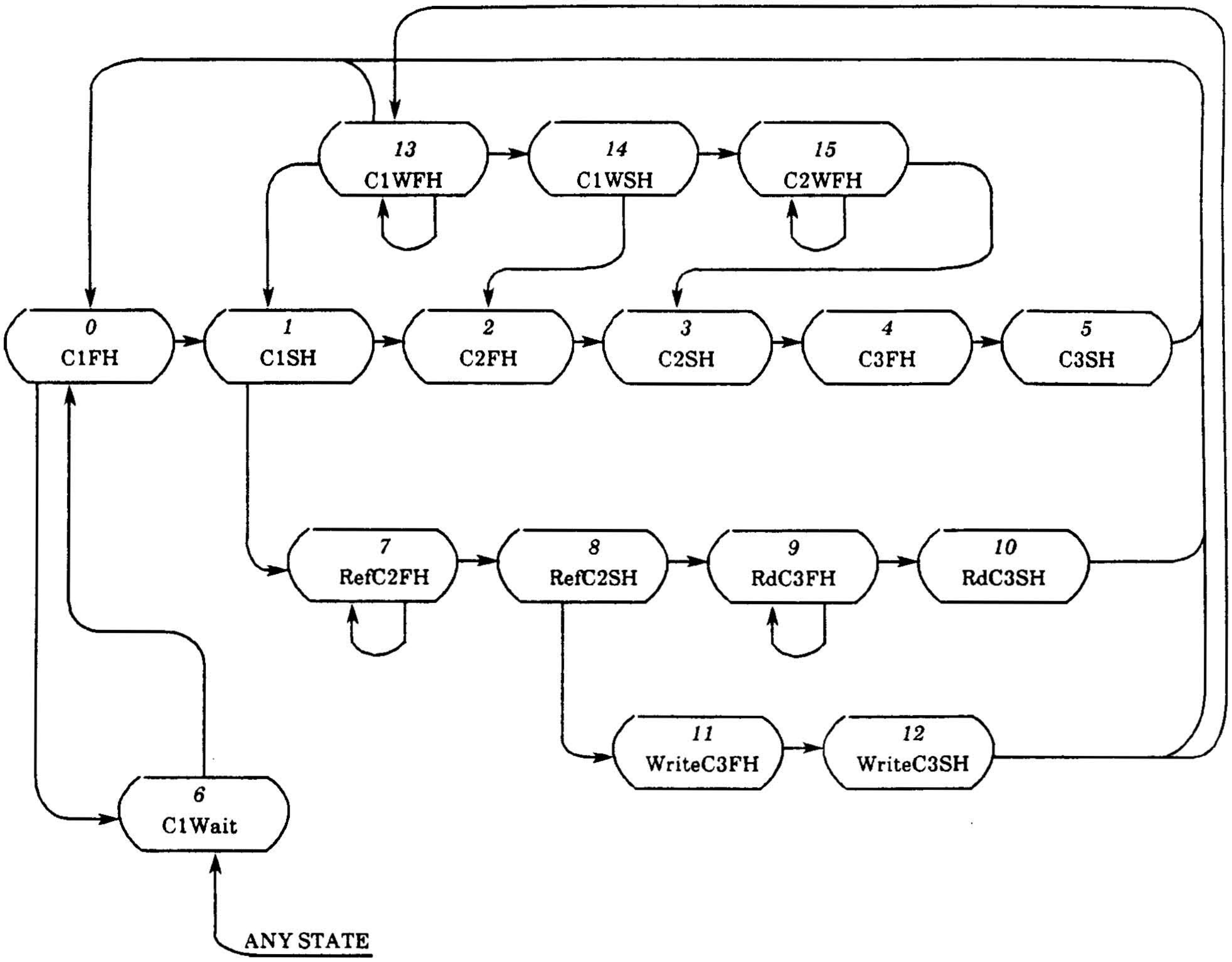


Figure 2.33. MBC state transitions

Boot Mode

State 0, State 6

The MBC state machine is initialized to a no-state condition when one of the following reset inputs is activated:

- AReset - hard reset from the IOP. It includes power-up reset and system reset/boot button.
- AResetMPB - software reset from the IOP.
- DebReset - from the debugger.

The first Halt (A/Halt'/MBCHalt') signal puts the Mesa processor into Boot mode. On entering Boot mode, the MBC state machine enters the C1Wait (Cycle 1 Wait - *state 6*) state. In this state, the state machine sets Cycle 1 and generates a Clear Command signal to clear all pending commands. The state machine then enters C1FH (Cycle 1 First Half - *state 0*) state.

Run Mode

States 1-5:

During booting, the state machine alternates between the C1Wait and C1FH states. When Halt goes inactive, the processor exits Boot mode and enters Run Mode.

In Run mode, when executing a non-reference (NOT memory-, Map-, or I/O-reference) instruction, the state machine sequences through Cycle 1 (C1), Cycle 2 (C2), and Cycle 3 (C3), where:

> C1 encompasses Cycle 1 First Half (C1FH) and Cycle 1 Second Half (C1SH - *state 1*);
>
> C2 encompasses Cycle 2 First Half (C2FH - *state 2*) and Cycle 2 Second Half (C2SH - *state 3*);
>
> and C3 encompasses Cycle 3 First Half (C3FH - *state 4*) and Cycle 3 Second Half (C3SH - *state 5*).

A CLKEnb signal is generated in the second half of every cycle to enable the system clock.

For reference instructions, the state machine still sequences through C1, C2, and C3, and C1 still encompasses C1FH and C1SH. However, in C1FH, the signal LoadMAH is generated and B/ALE is set. For Map references, the EnbMAPH flip-flop, EnbMAPL, and B/MemRef are also set. For Memory references, the EnbMARH flip-flop, EnbMARL, and B/MemRef are also set. For I/O references, the EnbMarH flip-flop is also set. The state machine then enters C1SH.

In a reference C1SH, if the reference is an I/O reference, then the state machine generates the signal LoadMAL and sets the I/O Command. The state machine then enters RefC2FH (Reference Cycle 2 First Half - *state 7*) instead of C2FH.

State 7

In RefC2FH, the state machine resets B/ALE at 2xCLK (center of half cycle). It also resets EnbMARL or EnbMAPL. In addition, if Mem bit is active (MemRef = 1), then WtCmd (Write Command) is set; if Mem bit is inactive (MemRef = 0), then RdCmd (Read Command) is set. The state machine then enters RefC2SH (Reference C2 Second Half - *state 8*).

State 8

In RefC2SH, if pMem = 1, then the state machine sets EnbRD (EnableRead Data). Note: pMem = 1 in Cycle 2 signifies Mem = 1 in Cycle 3, indicating a Read instruction.

If Mem = 1 (MemRef = 1), then the state machine sets EnbWD (Enable Write Data) and generates LoadWD (Load Write Data) before entering WtC3FH (Write Cycle 3 First Half - State 11). If Mem = 0, then the state machine branches into RdC3FH (Read Cycle 3 First Half - *state 9*).

State 9

ReadySyn is the Ready signal B/Rdy from memory which is internally synchronized with 2xCLK'. It informs the MBC that a memory- or map- reference instruction has been completed.

State 10

In RDC3FH, the state machine monitors the RdySyn and loops in RdC3FH if RdySyn = 0. When RdySyn is active, the state machine exits RdC3FH and enters RdC3SH (Read Cycle 3 Second Half - *state 10*).

In RdC3SH, the state machine resets EnbMARH or EnbMAPH flip-flop (whichever is set) and, if RdySyn is true, then generates a Clear Command signal. The state machine then returns to C1FH (*state 0*).

Write Operation

State 11

For a write operation, the state machine goes from WtC3FH into WtC3SH (Write Cycle 3 Second Half - *state 12*).

State 12

In WtC3SH, the state machine resets EnbMARH or EnbMAPH flip-flop (whichever is set). It then monitors the signal Ref+Rdysyn, where REF = any reference, memory, map, or I/O. If Ref+Rdysyn is true, then the machine generates a Clear Command signal and returns to C1FH. If Ref+Rdysysn is false, then the machine enters C1WFH (Cycle 1 Wait First Half - *state 13*).

State 13

In C1WFH, if Ref = 1 and Rdysyn = 0, then the state machine loops in C1WFH.

If both Ref and Rdysyn are true, then the state machine generates Clear Command and returns to C1FH.

If Ref = 0 and Rdysyn = 1, then the state machine generates Clear Command and returns to C1SH.

If both Ref and Rdysyn are false, then the state machine enters C1WSH (Cycle 1 Wait Second Half - *state 14*).

State 14

In C1WSH, if Rdysyn = 1, then the state machine generates Clear Command and returns to C2FH. If Rdysyn = 0, then the state machine enters C2WFH (Cycle 2 Wait First Half - *state 15*).

State 15

The state machine loops in C2WFH until Rdysyn becomes active. It then generates Clear Command and returns to C2SH (*state 3*).

Stop Mode

A Halt signal in Run mode puts the state machine in Stop mode. However, in order not to affect the operation in progress, certain conditions must be met before the state machine enters Stop mode:

1. The MBC bus must be idle (MBIdle = 1; that is, no memory reference, map reference, or I/O reference pending.
2. The state machine must enter Stop mode at the end of Cycle 2 and beginning of Cycle 3.

During Stop mode, CLKEnb is not generated; that is, all system clocks (CLK, CLKA', CLKB', and CLKC') are inhibited.

When Halt become inactive, the state machine re-enters Run mode and resumes normal operation at the beginning of Cycle 3, exactly where it had stopped.

Cycles of the State Machine

The figures that follow illustrate the sequence of operations at each memory cycle, including non-reference cycles. The circled numbers refer to the numbered states illustrated in Figure 2.33 and described in the text. The figures are:

2.34	Cycle 1
2.35	Cycle 2
2.36	Cycle 3 (Read)
2.37	Cycle 3 (Write)

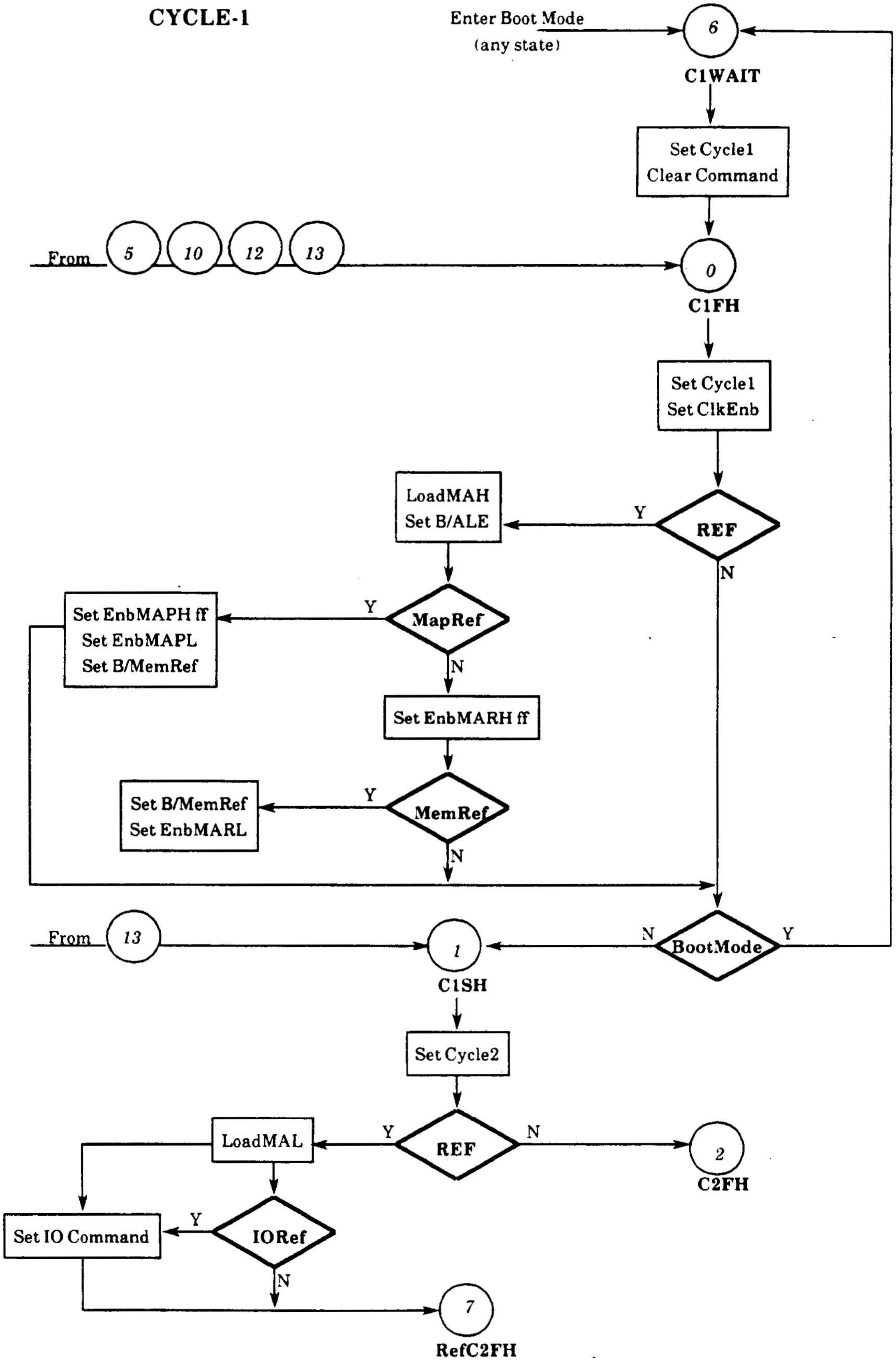


Figure 2.34. Cycle 1: state machine flow

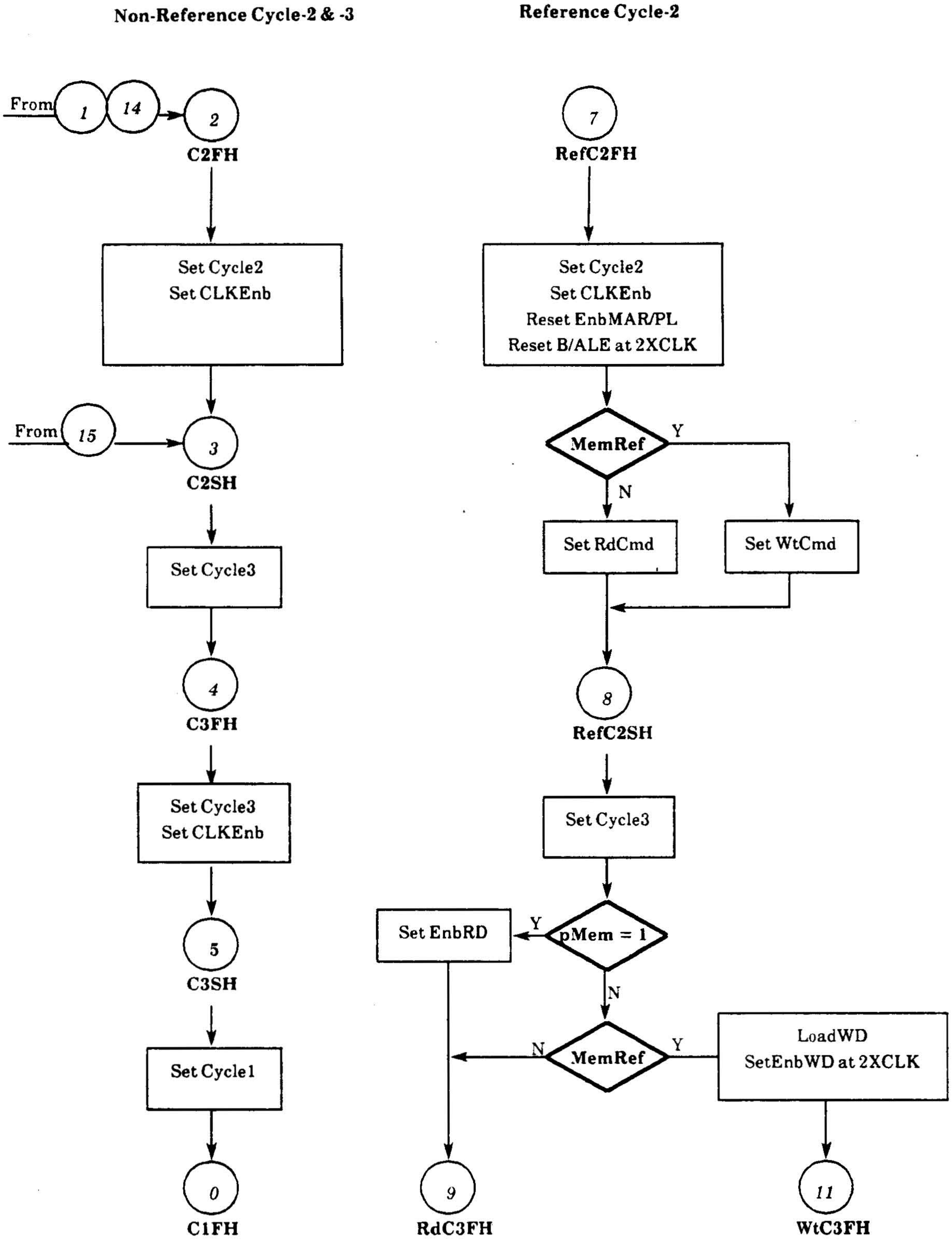


Figure 2.35. Cycle 2: state machine flow

Read Cycle-3

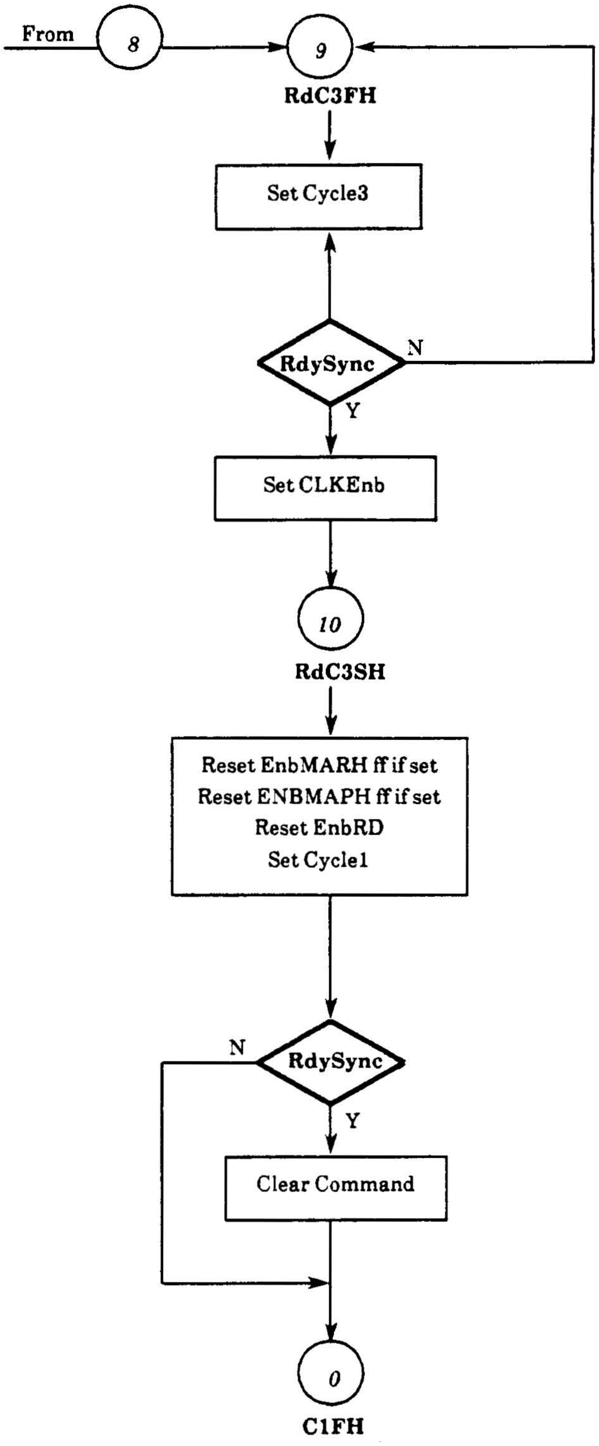


Figure 2.36. Read Cycle 3: state machine flow

Write Cycle-3

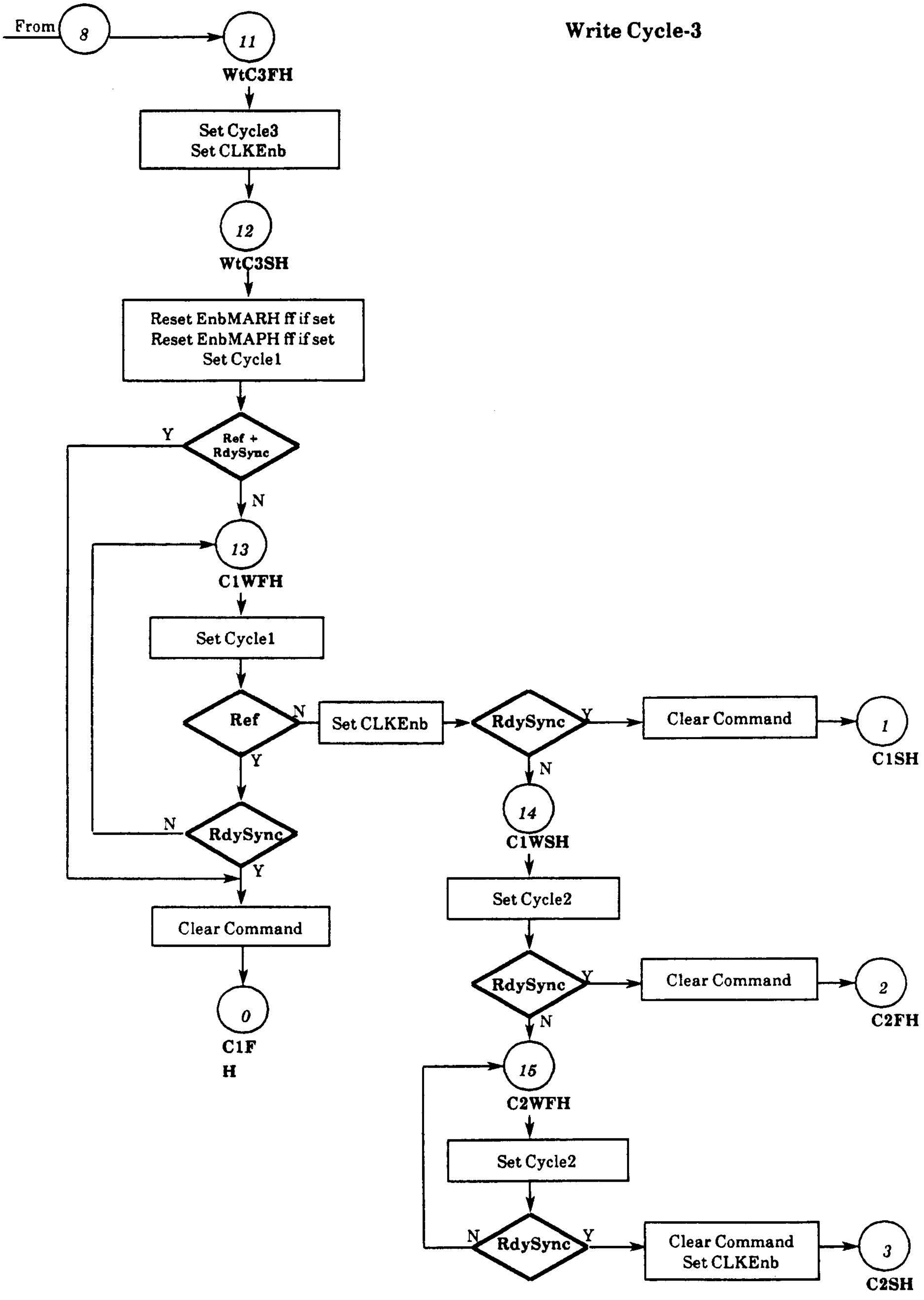


Figure 2.37. Write Cycle 3: state machine flow

2.5.2.2 Interrupt Control

The contiguous execution of emulator macroinstructions can be interrupted if immediate action is required by the interrupting source. Interrupt sources include :

- Microcode with fY = MesaIntRq
- IOP or AI Interface
- Interval TImer

An interrupt request sets the 1-bit Interrupt Register. At the same time, it also sets one bit in the Interrupt Status Register. Interrupt Status bits are assigned as follows:

Interrupt Source	Interrupt Status
Mesa microcode	bit 0
IOP/AI Interface	bit 1
Interval Timer	bit 2

When microcode issues a RdIntStat (Read Interrupt Status) command, the status bits are enabled onto the X bus as follows:

Interrupt Status	X Bus
bit 0	X.13
bit 1	X.14
bit 2	X.15

After the interrupt (or interrupts) has been serviced, a ClrIntTrap (Clear Interrupt Trap) is issued by microcode to clear both the 1-bit Interrupt Register and the Interrupt Status Register.

Note: All three status bits are cleared by the ClrIntTrap whether or not their corresponding interrupts have been serviced.

Interrupts can also be enabled or disabled by microcode with fY=SetIE (Set Interrupt Enable) or ClrIE (Clear Interrupt Enable) respectively.

2.5.2.4 Timer Control

Two Interval timers are implemented using the 8254 timer chips. As part of the Mesa Processor I/O, both interval timers are clocked by a 16-microsecond input clock.

In normal applicaton, only Interval Timer1 (Timer0 of 8254) is programmable and generates an interrupt (timer Interrupt) when the programmed count is reached.

Interval Timer2 is implemented by cascading Timer1 and Timer2 of the 8254 Timer Chip into one modulo 2^{32} counter. Timer2 input clock is derived from the RC (Ripple Carry) output of Timer1. No interrupt is generated by Interval Timer2.

Figure 2.38 illustrates Timer Gate and Timer Clock. Figure 2.39 illustrates Timer1 Ripple Carry Clock and Fig.2.40 illustrates Timer2 Clock Timing.

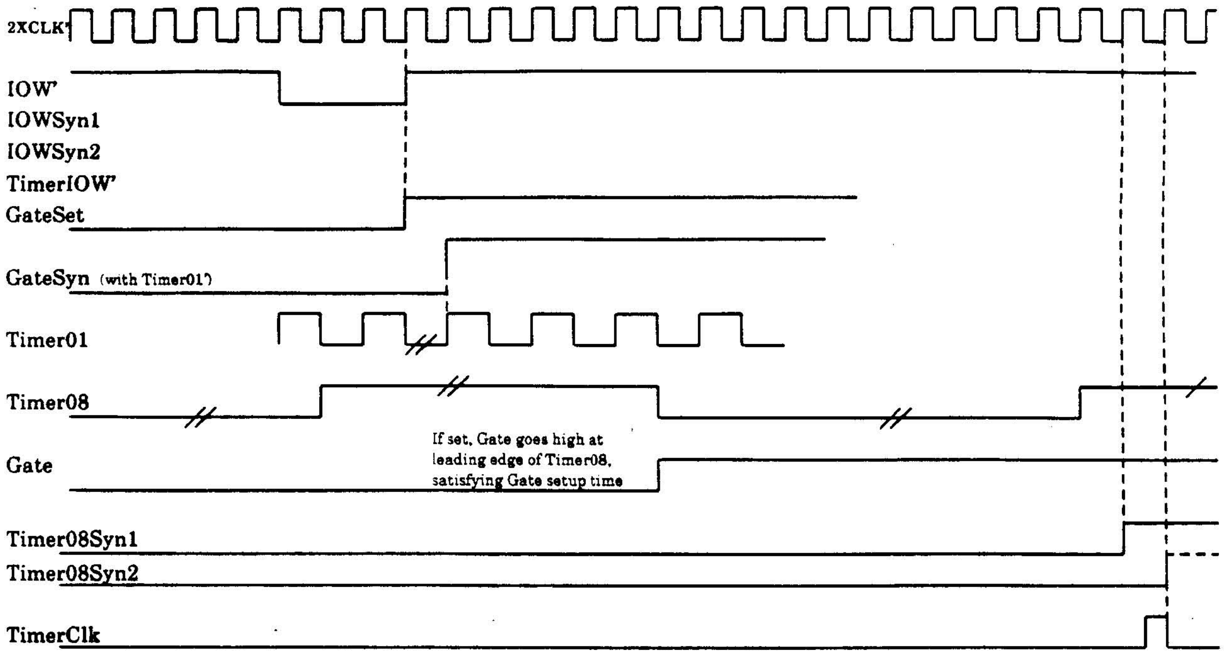


Figure 2.38. Timer gate and clock timing

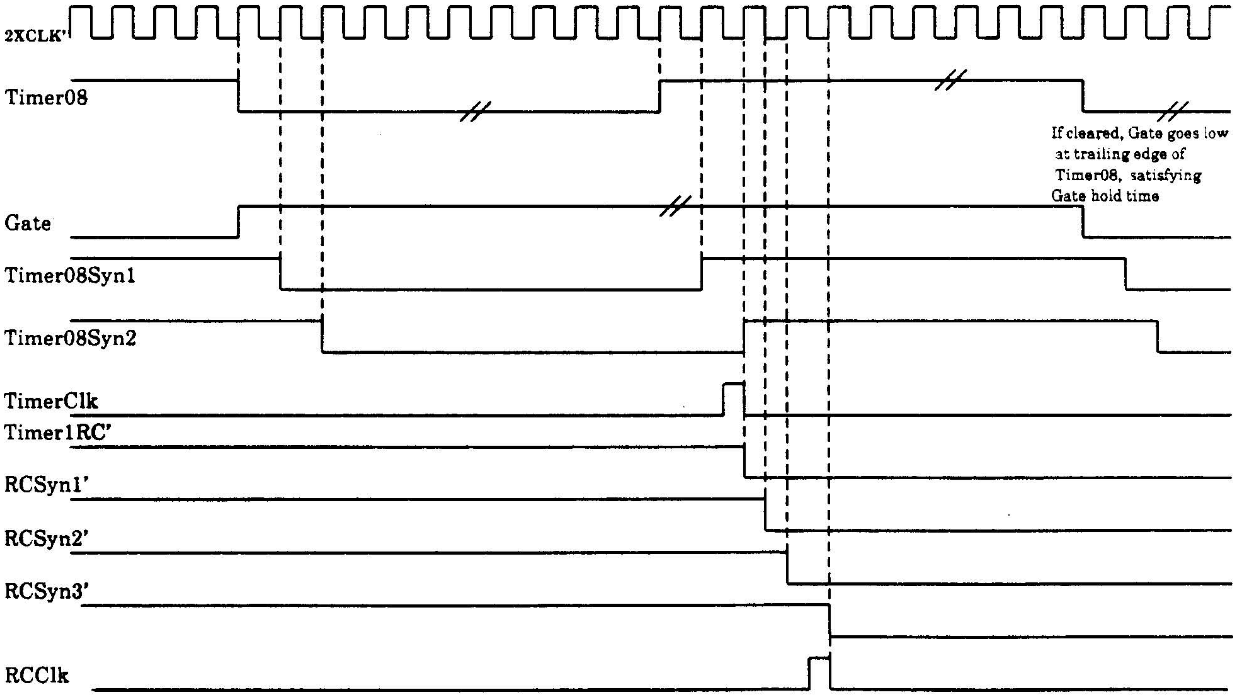
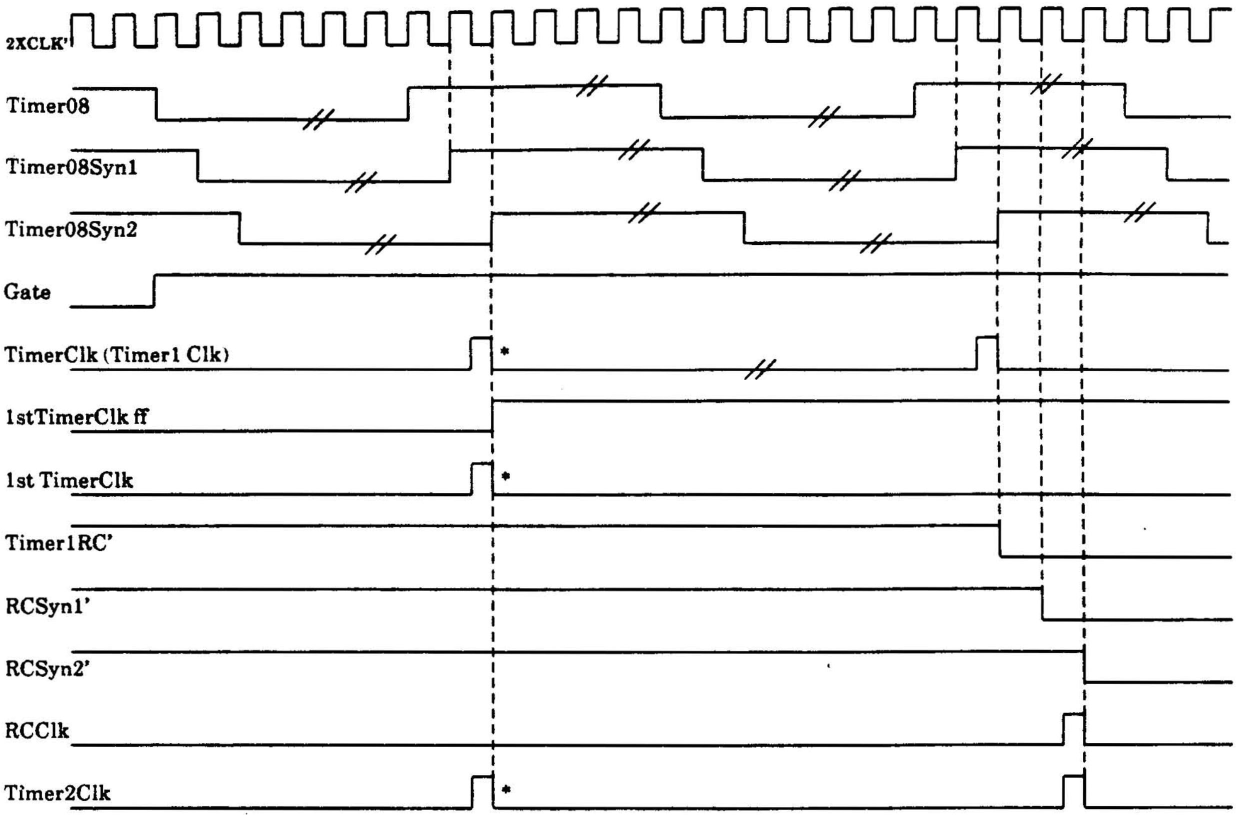


Figure 2.39. Timer1 ripple carry clock



* First clock pulse after Gate goes high loads count into counting element

Figure 2.40. Timer2 clock timing

2.5.3 Programmer Interface: Main Memory Addressing

Refer also to Daybreak Microcode Reference Manual for detailed information.

The memory system accepts two types of addresses: real and virtual. Real references result in a read or write to the addressed location itself. Virtual references cause the memory system to ignore the low byte of the address. Using the remaining 16 bits, the memory system then reads or writes the Map, located at real address 10000 hex.

For both reference types, a write occurs (MDR←) when the memory operation (mem) field is set in c2; a read occurs (←MD) when the mem field is set in c3. Read and write should not both be specified in the same click. Furthermore, if a click specifies an MDR← or ←MD without a corresponding MAR←, then memory is not written, and a potential memory error trap does not occur.

Microcode instructions for memory addressing are described in this subsection. The memory system varies, depending on display memory size. In this section, maximum size is assumed; that is, 20-bit real addresses and 24-bit virtual addresses.

2.5.3.1 Real Address References

The mem bit true in cycle 1 causes a real reference. A real reference is specified by using the MAR← macro in c1. The memory address is sent to the DCM from the Y and YH buses via the B bus interface. The Y bus can be driven either from the 2901 F bus or by A-bypass; addresses can be either pre- or post-modified. The YH bus, which supplies the high-order address bits, is always driven by the RH register addressed by rB. YH[0-3] are ignored by memory.

With MAR← the following effects, described below, occur:

1. The 2901 is divided such that the high half executes a fixed function;
2. a special "address-overflow" branch (pageCross) is enabled;
3. an MDR← or IBDisp in the next cycle is canceled if the branch is taken.

Split 2901

If mem = 1 in c1, then the 2901 is divided such that the high half executes with its aS and aF inputs equal to (0,B) and (aF OR 3), while the low half executes with the aS and aF values given by the microinstruction. This division causes the high byte of the ALU output to equal the high byte of the R register addressed by rB (or its complement if aF is in [4-7]).

As an outcome of the bipartition, a carry out from the low half does not propagate into the high half; that is, the high byte of rB remains unchanged after a MAR← (unless aF is in [4-7]), even when A-bypass is used.

Figure 2.41 illustrates real address modes.

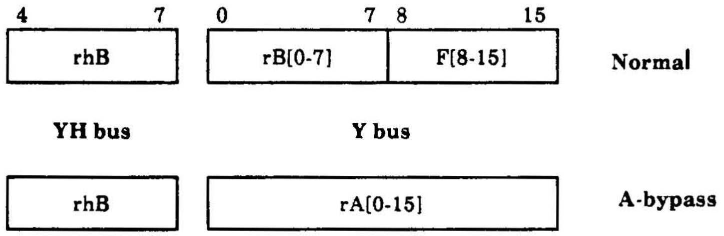


Figure 2.41. MAR Address Types

If A-bypass is not specified, then the upper 12 bits of the memory address (the page address) come from the Rh/R pair named by the rB field. The lower 8 bits (the page displacement) are defined by the desired ALU operation. Assuming the Y bus is driven from the F bus, the 20-bit real address is rhB[4-7],,rB[0-7],,F[8-15].

Note: This feature can be used to combine the real page number, as read from the Map in the previous cycle, with a displacement into the page.

If A-bypass is specified, then the lowest 16 address bits come from the R register addressed by rA. The 20-bit real address is rhB[4-7],,rA[0-15].

pageCross Branch

If the ALU operation results in a carry out from the low half, then MAR← automatically specifies a pageCross branch; 1 is ORed into INIA[10]. Thus, although the carry out from the low byte does not propagate into the high byte, it can be detected as a transfer of control. A true pageCross branch can imply that the real address is invalid, and that a remapping of the virtual address originally generating the real address is necessary. Since pageCross is not ORed into INIA[11], other simple branches can be specified concurrently.

pageCross is defined as (pageCarry XOR aF[2]), where pageCarry is the carry out from the low 2901 byte. For addition, pageCross equals pageCarry. For subtraction, the XOR has the effect of toggling pageCarry.

Notes: 1. The aF = (R-S) form of subtraction does not cause pageCarry to be inverted, since aF[2] = 0. However, the aF = (R-S) form covers the most common subtraction requirements.
2. A complication of pageCross branch is that pageCross can equal 1 if the 2901 executes a logical function instead of an arithmetic function.

Cancellation of c2 Functions

If pageCross = 1 during a MAR←, then a following MDR←, IBDisp, or AlwaysIBDisp in c2 is ignored. This effect increases the need to avoid logic functions during a MAR←.

Note: The cancellation effect can be used to prevent writing into the wrong page or to prevent dispatching on the next Emulator instruction when the corresponding virtual address should be remapped.

2.5.3.2 Virtual Address References

Translation of virtual to real address is done explicitly in microcode. Figure 2.42 illustrates virtual-to-real address mapping.

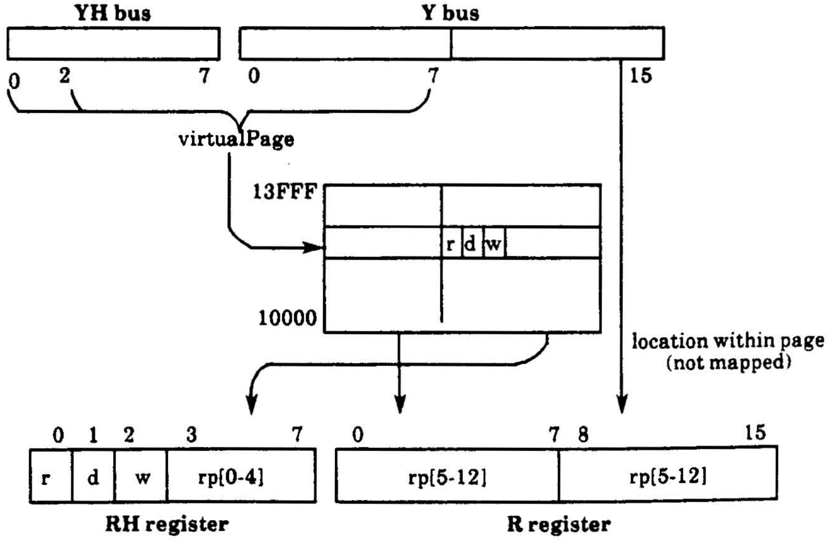


Figure 2.42. Virtual-to-real address mapping

When either the fX or fY field equals MAP← in cycle 1, a memory reference to the virtual-to-real, page-translation map is caused. Map is a 16 KWord table whose first entry is at location 10000 hex , just after the display bank. During a Map reference, the memory system uses the upper 16 bits of the virtual address (14 bits for a 22-bit virtual address) to index into the table. Each entry of the table contains a 12-bit real-page number and three flags pertaining to the virtual page.

Figure 2.43 illustrates map address types.

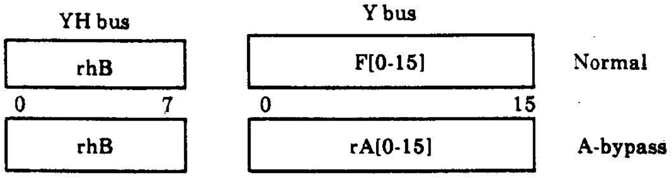
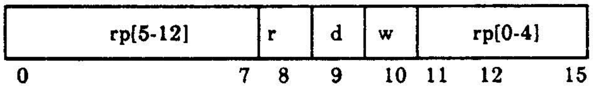


Figure 2.43. Map address types

The virtual address is carried on the Y and YH buses. The low byte of the Y bus is ignored, without affecting the ALU. Since the Y bus can be driven from the 2901's bus or from A-bypass, addresses can be either pre- or post-modified.

For 24-bit virtual reference, all of the YH bus is used.

Figure 2.44 illustrates the format of a Map entry. Refer to the Daybreak Microcode Reference Manual for a description of how the Map flag bits are maintained.



rp[0-12]	Real page Number
r	Referenced and Present flag
w	Write Protect flag
d	Dirty flag

Figure 2.44. Map Entry Format

The mem field should not be set in c1 along with a Map← unless the side effects of MAR← are explicitly desired.

2.5.3.3 Memory Address Register

Figure 2.45 illustrates the memory address register (MAR). The contents of YH[4-7],,Y[0-15] are used as the memory address.

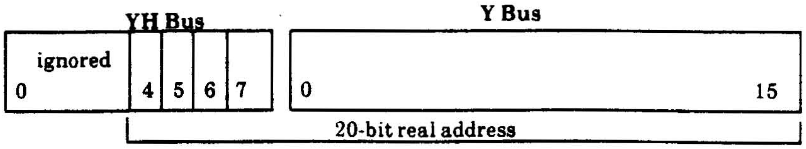


Figure 2.45. Memory Address Register address generation

MAR← [rhReg, <arithPhrase>] designates a real address reference to memory. rhReg specifies the RH register that holds the two high order address bits. The rB field is set to the value of rhReg; <arithPhrase> can be any notation that occurs on the right side of an arithmetic clause.

2.5.3.4 Map Reference

A map reference occurs when mem is set in c1. The action is the same as for MAR← except that the physical address is derived differently. An access is started in the 65K - 80K bank of memory; the location accessed is specified by the page number.

Figure 2.46 illustrates the derivation of the physical address.

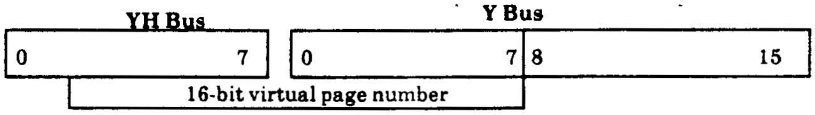


Figure 2.46. Map← address generation

2.5.3.5 Memory Data Register

The memory write data register (MDR) is loaded with the contents of the Y bus when mem is set in c2. The contents are written into the memory location specified by the contents of MAR loaded during the first cycle of the click. If the low 64K bank is selected and is being used by the display, then no write occurs.

2.5.3.6 Memory Data

Memory Read data (MD) is placed on the X bus when mem is set in c3.

Before the next memory read (←MD) is done, the status of a given read operation can be found in MStatus .