SECTION 3
LOGIC DESCRIPTIONS

In this section, a selection of the twelve board types comprising the EBox are described in detail. Wherever possible, a functional perspective is given to highlight the particular functions a board or portion of a board implements, and multiple boards are shown interconnected to aid in tracing various control signals from one functional area to another.

PHYSICAL CONFIGURATION

The EBox consists of a total of 23 modules, configured as indicated in Figure 3-1. A brief description of each module is contained in the following paragraphs.

Module M8532, Priority Interrupt Control (PIC) – One board, illustrated on customer prints PIC 1–6, contains PI ON register 1–7, PI GEN register 1–7, PI REQUEST Register 0–7, PI HOLD register 1–7, and the PI ACTIVE flip-flop. In addition, it contains the priority interrupt networks for arbitration of priority interrupt requests, EBus dialogue logic, control and internal timing, and the assignment registers for the ABR: PIA APR 1,2,4 and Meter PIA 1,2,4.

Module 8526, Clock (CLK) – One board, illustrated on customer prints CLK 1–6, contains the crystal-controlled master clock oscillator and crystal-controlled margin clock oscillator, as well as Source and Rate Selection registers and their associated logic. It contains logic and counters to produce the EBus clock, SBus clock, MBox clocks, and EBox clocks. In addition, it contains single step, burst, normal, and diagnostic mode logic and registers. It also contains MR reset, EBus reset, crobar logic, error detection logic, page fail, and MBox request logic.

Module 8539, Arithmetic Processor Status (APR) – One board, illustrated on customer prints APR–7, contains an 8-bit APR Status register, 8-bit Interrupt Enable register, and associated interrupt request detection logic. It contains the EBus dialogue control logic used while performing I/O instructions. In addition, it contains the address break compare enable bits, fetch comp, read comp, write comp, and user comp. It contains a 5-bit section register, fast memory bit 36, RAM storage, and parity network. It also contains the fast memory block and word addressing logic, mixers, adder network and current, previous XR, and VMA Block Selection registers. It also contains MBox control and MBox register function decoding logic.

Module 8525, EBox Control No. 2 (CON) – One board, illustrated on customer prints CON 1–6, contains CRAM condition field decoding; COND and SKIP enables; and VMA select lines CON VMA SEL 1 and 2. It contains meter, interrupt request and interrupt request detection logic, run and continue logic, IR strobe, DRAM strobe, start logic, various flip-flops, and associated sychronizer logic. It also contains the NICOND decoding and COND ADR bit 10 logic. It contains a 4-bit State register, diagnostic function decoding logic, Parity Enable register, Cache Strategy register, paging enable, trap-enable bits, and I/O control signals for CONO APR, CONO PI, CONO PAG, and DATAO APR. It contains the Load AC blocks and Load Previous Context signals, 4-bit Microcode State register, AR and ARX bit 36 with associated logic, fast memory, write logic, various PI control signals, and associated logic.

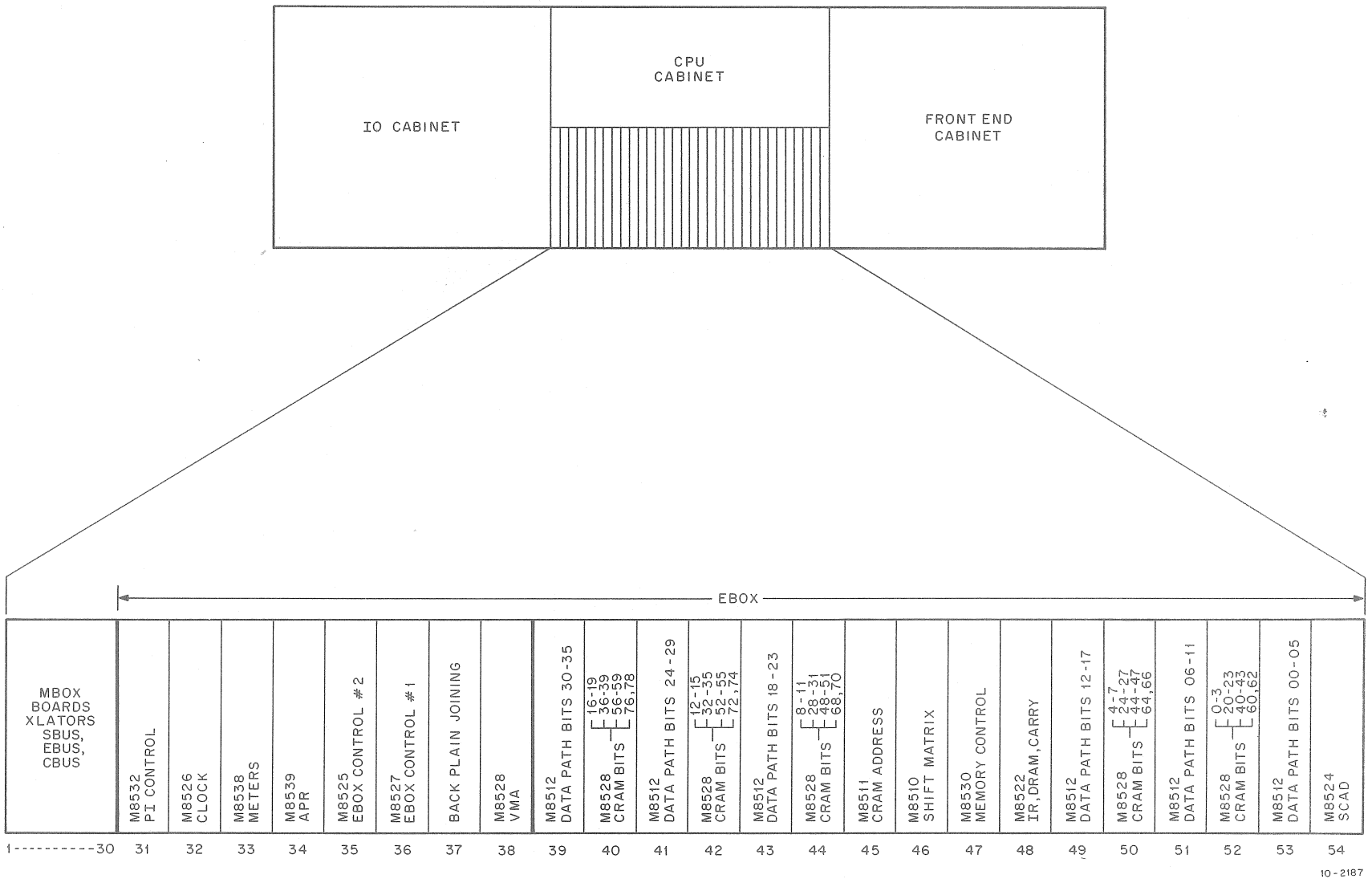


Figure 3-1 EBox Module Utilization

Module 8527, EBox Control No.1 (CTL) – One board, illustrated on customer prints CTL 1–4, contains CRAM dispatch, field decoding, some adder carry control logic, and register mixer selection control logic for AR, ARX, MQ, and PC. It also contains the majority of the diagnostic decoding logic and the translator enables T to E enable and E to T enable.

Module 8523, Virtual Memory Address (VMA) – One board, illustrated on customer prints VMA 1–6, contains an 18-bit VMA adder, VMA AC reference detection logic, a 23-bit VMA register, and associated input mixing logic. It also contains a 23-bit Address Break register, associated match logic, 23-bit Program Counter register, 23-bit VMA Held register, and AR Mixer Mixer (ARMM) logic bits 13–17.

Module 8528, Data Path (DP) – Six boards, illustrated on customer prints DP 1–5, each contain six bits of a full 36-bit data path. Each board contains the following mixers: AR Mixer (ARM), ARX Mixer (ARXM), MQ Mixer (MQM), ADA Input Mixer, ADB Input Mixer, ADXA Input Mixer, and ADXB Input Mixer. In addition, each board contains the following registers: Arithmetic Register (AR), Arithmetic Register extension (ARX), Buffer Register (BR), Buffer Register extension (BRX), and Multiplier Quotient register (MQ). It also contans fast memory, the adder (AD), and adder extension (ADX). In addition, it contains the fast memory, write pulse generation logic, and fast memory, write pulse generation logic, and fast memory parity network.

Module 8512, Control RAM (CR) – Five boards, illustrated on customer prints CR 1–7, each contain 14 bits of the control word (microinstruction) stored in RAMs containing 1280 words. In addition, each board contains CRAM address gating and 14 bits of the CRAM output register (CRAM register).

Module 8511, Control Ram Address (CRA) – One board, which is illustrated on customer prints CRA 1–6. This board contains the circuitry to generate the address of the next CRAM word. This includes the microcode push-down stack, plus the Dispatch and Skip logic.

Module 8510, Shift Matrix (SH) – One board, illustrated on customer prints SHM 1–5, contains shift counter decoding logic, shift matrix, and AR and ARX parity networks.

Module 8530, Memory Control (MCL) – One board, illustrated on customer prints MCL 1–7, contains CRAM MEM field decoding; memory request enable logic; request type decoding, e.g., MCL VMA Read, MCL VMA Pause, MCL VMA Write. It also contains User and Public Enable logic, as well as all the request-type qualifiers. It contains bits 1–12 of the VMA Held or PC Mixers, together with various VMA Control and Selection logic.

Module 8522, IR, DRAM, and Carry (IRD) – One board, illustrated on customer prints IRD 1–5, contains the 13-bit Instruction register (IR), 4-bit IRAC register, DRAM address mixers, DRAM, and 15-bit DRAM Output register. In addition, it contains the IR Test Satisfied logic and normalization CRAM address bits (IR NORM 08–10). It also contains the AD and ADX carry anticipation networks (CARRY SKIPPER).

Module 8524, Shift Counter Adder (SCAD) – One board, illustrated on customer prints SCD 1–6, contains the 10-bit Shift Counter register and associated input mixer, 10-bit Floating Exponent register, and associated input mixer, AR Mixer Mixer (ARMM) bits 0–8, and SCD TRAP Mixer (32–35). It also contains the 10-bit Shift Counter Adder (SCAD) as well as the Program Counter Flags register and mode control logic.

3.1 INSTRUCTION REGISTER LOADING AND CONTROL

Refer to Figures 3-2 and 3-3. The IR is composed of 13 mixer latches as illustrated. The default selection is AD selected by -CLK MB XFER. The alternate selection is the cache data lines selected by CLK MB XFER. Because the IR consists of latches (DC devices), the clock is used indirectly to synchronize unlatching and latching of IR. This is done by ORing the EBox clock with the control signal on the IR Board. Unlatching the IR may be accomplished in one of three ways.

Module 8527, EBox Control No.1 (CTL) – One board, illustrated on customer prints CTL 1–4, contains CRAM dispatch, field decoding, some adder carry control logic, and register mixer selection control logic for AR, ARX, MQ, and PC. It also contains the majority of the diagnostic decoding logic and the translator enables T to E enable and E to T enable.

Module 8523, Virtual Memory Address (VMA) – One board, illustrated on customer prints VMA 1–6, contains an 18-bit VMA adder, VMA AC reference detection logic, a 23-bit VMA register, and associated input mixing logic. It also contains a 23-bit Address Break register, associated match logic, 23-bit Program Counter register, 23-bit VMA Held register, and AR Mixer Mixer (ARMM) logic bits 13–17.

Module 8528, Data Path (DP) – Six boards, illustrated on customer prints DP 1–5, each contain six bits of a full 36-bit data path. Each board contains the following mixers: AR Mixer (ARM), ARX Mixer (ARXM), MQ Mixer (MQM), ADA Input Mixer, ADB Input Mixer, ADXA Input Mixer, and ADXB Input Mixer. In addition, each board contains the following registers: Arithmetic Register (AR), Arithmetic Register extension (ARX), Buffer Register (BR), Buffer Register extension (BRX), and Multiplier Quotient register (MQ). It also contans fast memory, the adder (AD), and adder extension (ADX). In addition, it contains the fast memory, write pulse generation logic, and fast memory, write pulse generation logic, and fast memory parity network.

Module 8512, Control RAM (CR) – Five boards, illustrated on customer prints CR 1–7, each contain 14 bits of the control word (microinstruction) stored in RAMs containing 1280 words. In addition, each board contains CRAM address gating and 14 bits of the CRAM output register (CRAM register).

Module 8511, Control Ram Address (CRA) – One board, which is illustrated on customer prints CRA 1–6. This board contains the circuitry to generate the address of the next CRAM word. This includes the microcode push-down stack, plus the Dispatch and Skip logic.

Module 8510, Shift Matrix (SH) – One board, illustrated on customer prints SHM 1–5, contains shift counter decoding logic, shift matrix, and AR and ARX parity networks.

Module 8530, Memory Control (MCL) – One board, illustrated on customer prints MCL 1–7, contains CRAM MEM field decoding; memory request enable logic; request type decoding, e.g., MCL VMA Read, MCL VMA Pause, MCL VMA Write. It also contains User and Public Enable logic, as well as all the request-type qualifiers. It contains bits 1–12 of the VMA Held or PC Mixers, together with various VMA Control and Selection logic.

Module 8522, IR, DRAM, and Carry (IRD) – One board, illustrated on customer prints IRD 1–5, contains the 13-bit Instruction register (IR), 4-bit IRAC register, DRAM address mixers, DRAM, and 15-bit DRAM Output register. In addition, it contains the IR Test Satisfied logic and normalization CRAM address bits (IR NORM 08–10). It also contains the AD and ADX carry anticipation networks (CARRY SKIPPER).

Module 8524, Shift Counter Adder (SCAD) – One board, illustrated on customer prints SCD 1–6, contains the 10-bit Shift Counter register and associated input mixer, 10-bit Floating Exponent register, and associated input mixer, AR Mixer Mixer (ARMM) bits 0–8, and SCD TRAP Mixer (32–35). It also contains the 10-bit Shift Counter Adder (SCAD) as well as the Program Counter Flags register and mode control logic.

3.1 INSTRUCTION REGISTER LOADING AND CONTROL

Refer to Figures 3-2 and 3-3. The IR is composed of 13 mixer latches as illustrated. The default selection is AD selected by -CLK MB XFER. The alternate selection is the cache data lines selected by CLK MB XFER. Because the IR consists of latches (DC devices), the clock is used indirectly to synchronize unlatching and latching of IR. This is done by ORing the EBox clock with the control signal on the IR Board. Unlatching the IR may be accomplished in one of three ways.

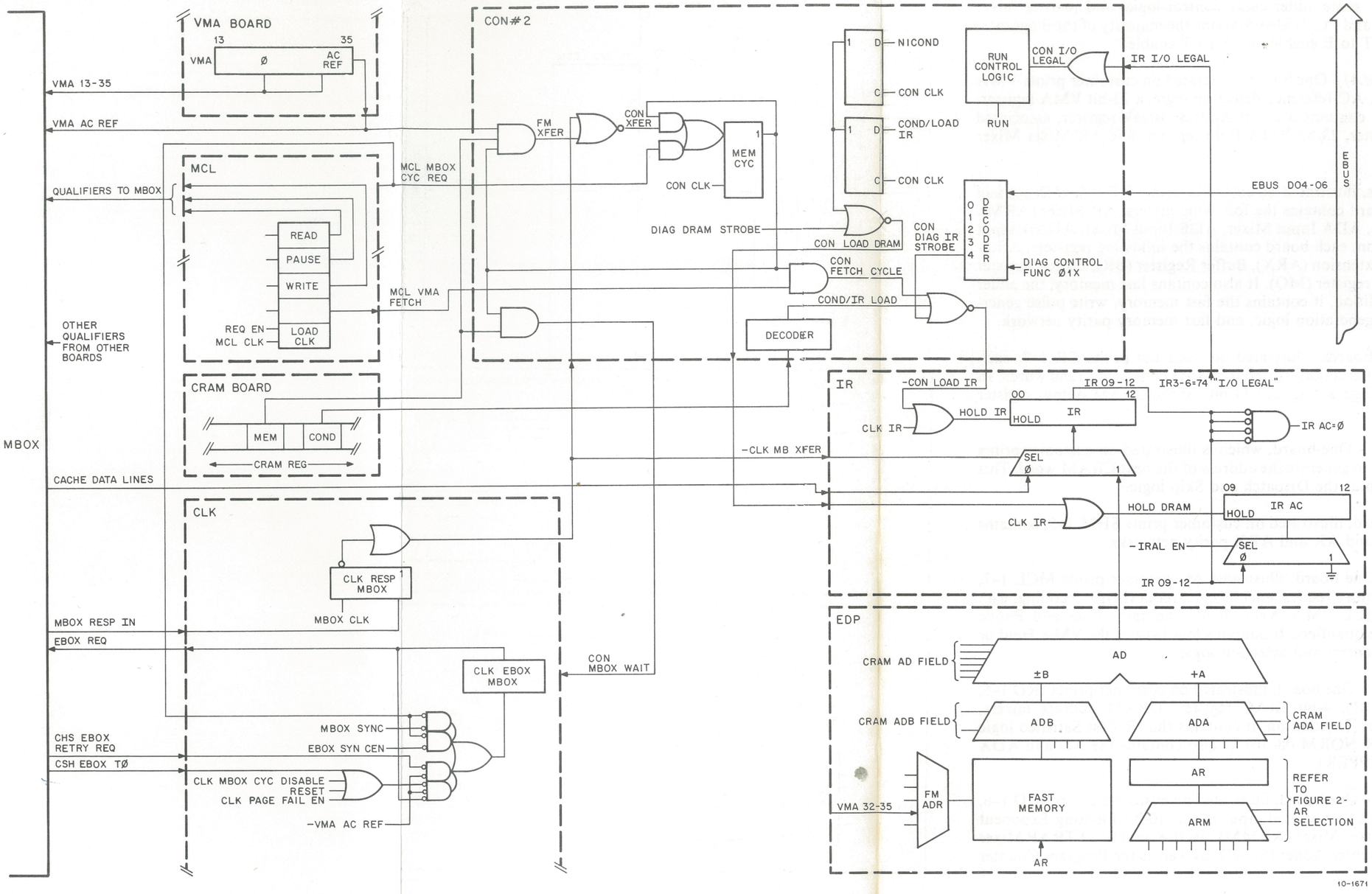


Figure 3-2 IR DRAM Control (Part 1)

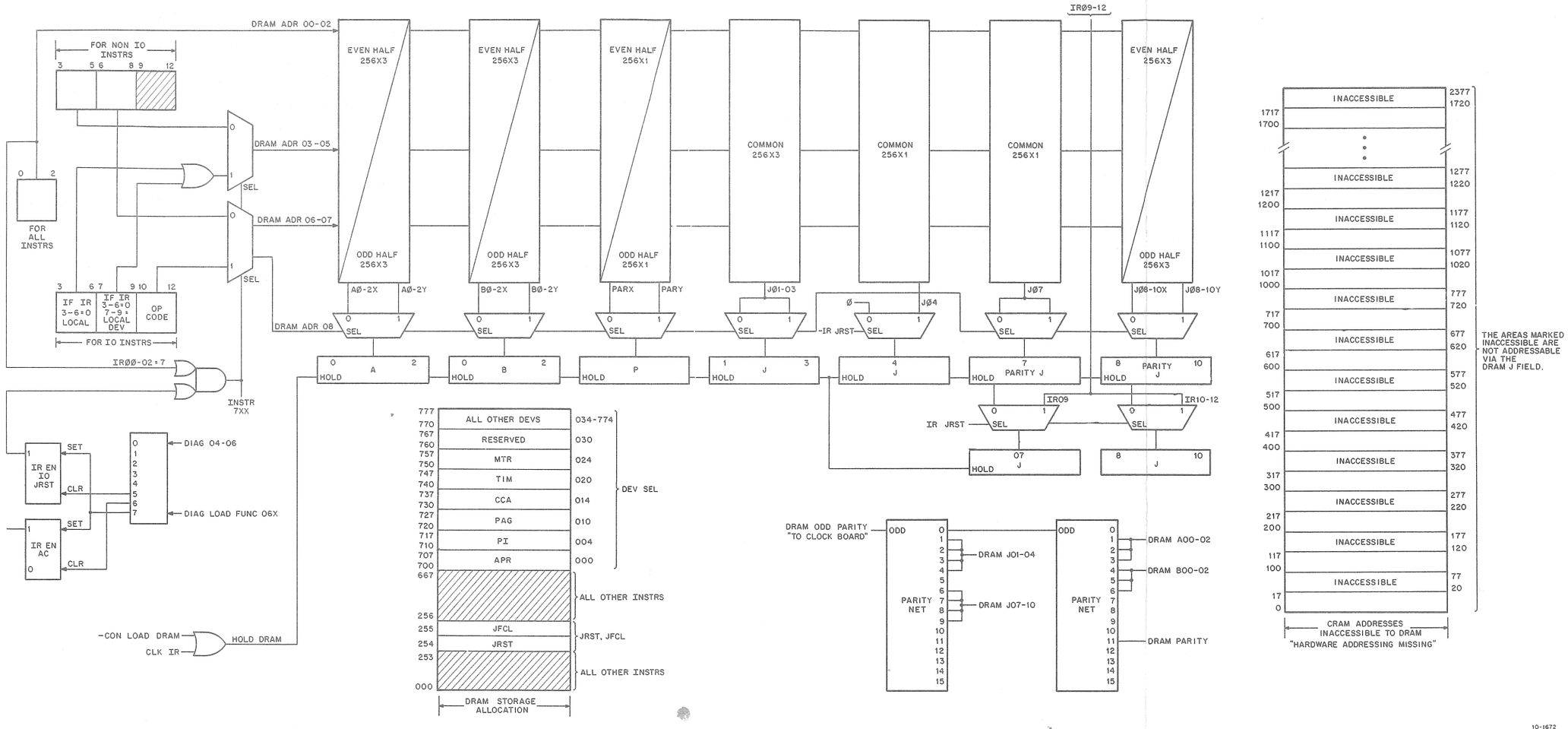


Figure 3-3 IR DRAM Control (Part 2)

During an instruction fetch, a logic level MCL FETCH is developed together with EBox Read. These qualifiers are latched at the same time that the VMA is latched during the EBox request. They are latched until the next EBox request. Each time a memory cycle is begun for any reason, MEM CYCLE sets. It remains set until one of two events occurs. Either MBXFER occurs in response to an MBox cycle, or FM XFER occurs in response to an internal fast memory cycle. Either of these decouples the feedback path for the MEM CYCLE flip-flop. Note that while MEM CYCLE and MCL VMA FETCH are true, the IR is unlatched because -CON LOAD IR becomes false removing HOLD IR.

A second method for unlatching the IR is via the microinstruction COND field function COND/LOAD IR. This may be used in cases where an instruction is loaded into AR to be executed. The microinstruction selects the AD function as "A" while selecting the AR on the ADA input. Because the default selection for IR is the AD, the instruction in AR would appear on the IR input mixer.

The operation of unlatching and loading in this manner takes one microinstruction as indicated in Figure 3-4. Note that CLK IR is logically ORed with -CON LOAD IR on the IR Board.

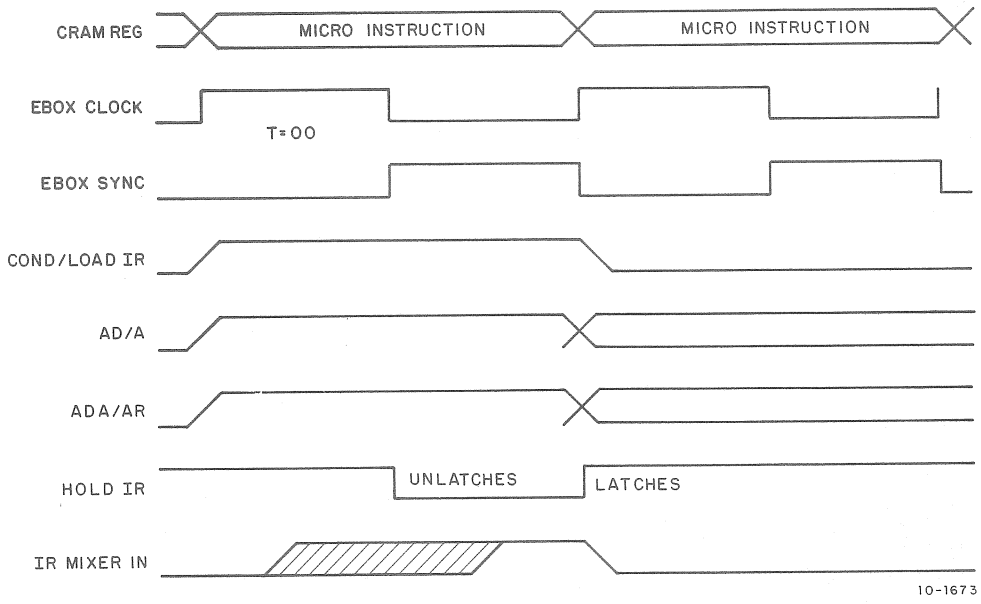


Figure 3-4 IR Loading Via AR (COND/LOAD IR)

By using diagnostic console function 014 (STROBE IR), information previously loaded into AR or ARX may be loaded into IR. This provides a powerful diagnostic tool. In addition, this function is used to address the DRAM while loading it.

When fetching instructions from fast memory via AD, it is sometimes necessary to use the COND/IR LOAD function to enable AD to IR. Referring to Figures 3-2 and 3-5, VMA bits 32–35 address fast memory as specified by the microinstruction FM ADR field. At the same time (for example), the ARX field selects AD while the AD field selects "B". The ADB field function is FM and once again the COND field is LOAD IR.

Once again, note that the unlatching and latching of IR is in step with the EBox clock (CLK IR).

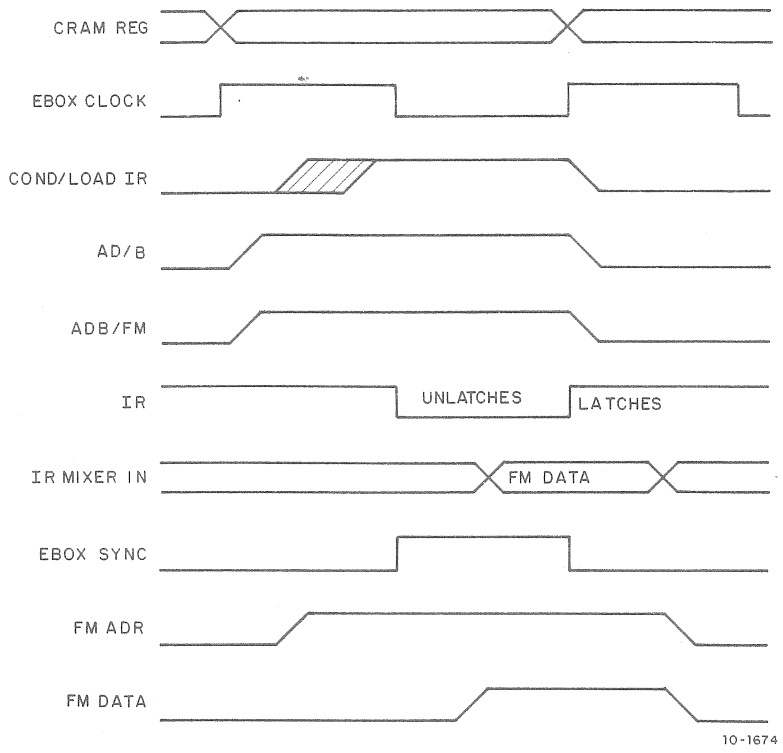


Figure 3-5 Loading IR Via FM (COND/LOAD IR)

3.1.1 DRAM and IRAC Control

The DRAM register is controlled in a manner similar to that of IR. The DRAM register consists of 19 mixer latches. Refer to Figure 3-3; unlatching the DRAM register may be accomplished in one of three ways. As with IR, note unlatching and latching of the DRAM register is synchronized by ORing the EBox clock with the control signal on the IR Board.

Each time that the COND/LOAD IR function is used to unlatch the IR, it also enables the generation of CON LOAD DRAM on the next EBox clock. Thus, the IR unlatches beginning with the trailing edge of one EBox clock and latches on the leading edge of the next. Similarly, the DRAM register unlatches beginning with the trailing edge of the EBox clock that latched IR, and latches once again on the leading edge of the following EBox clock. The timing is illustrated in Figure 3-6.

A similar operation takes place following NICOND Dispatch. Referring to Figures 3-2 and 3-7, NICOND is latched into a flip-flop on the control board at the same time that the microinstruction selected by the NICOND Dispatch loads into the CRAM register.

Here we assume the case where some instruction has completed its store cycle. An earlier micro-instruction generated MEM/FETCH which started the EBox Request.

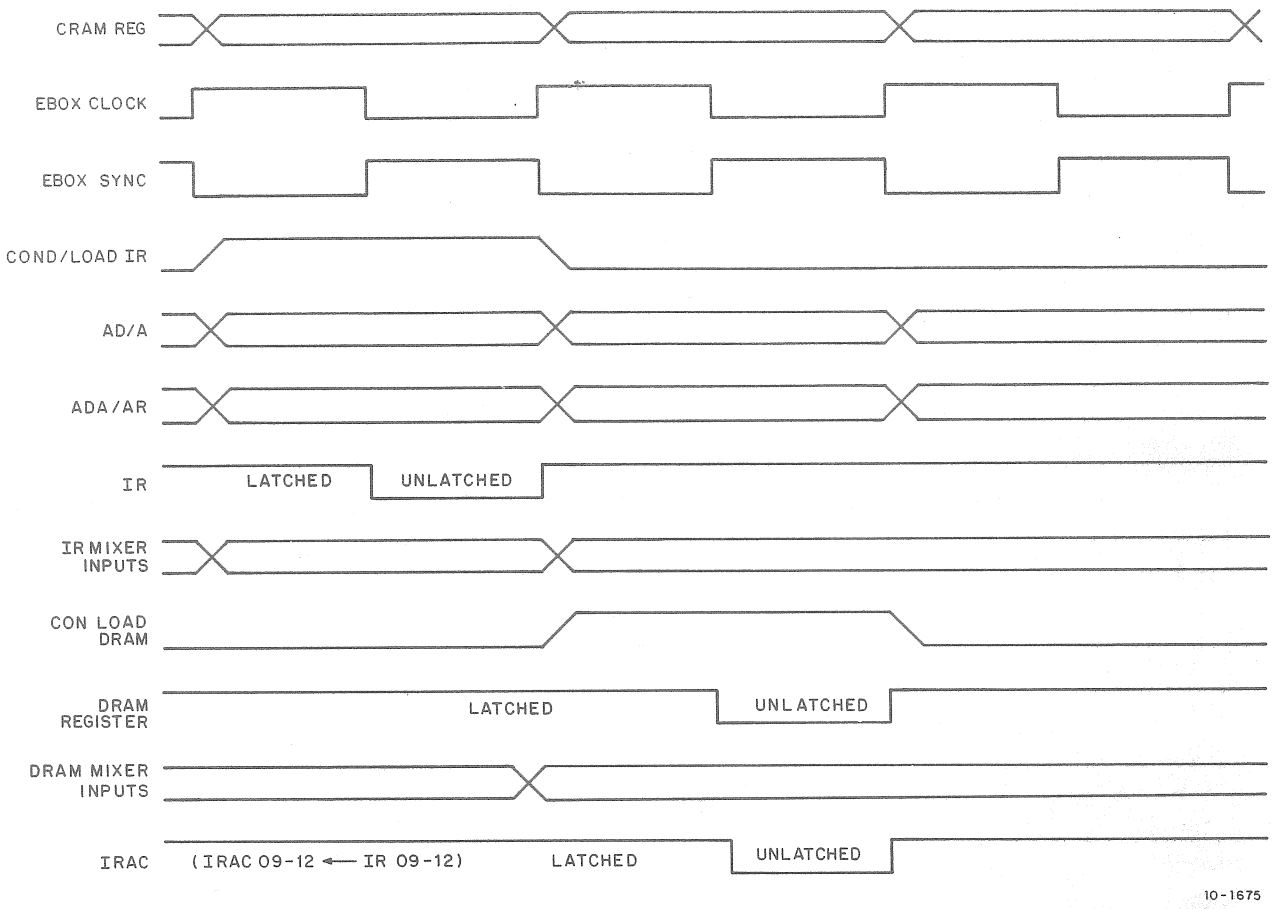


Figure 3-6 DRAM Loading Following COND/LOAD IR

3.1.2 DRAM Addressing and Selection

Assume IR EN IO, JRST, and IR EN AC are set. The DRAM addressing logic maps the incoming instruction code into the DRAM register as indicated in Figure 3-3. Note that I/O instructions address the DRAM in a slightly different fashion than non-I/O instructions. I/O instructions have bits 0–2 of IR equal to 7; this is detected on the IR Board as IR INSTR 7XX and enables the DRAM ADR to be formed as follows:

DRAM ADR 00–02 ←IR 00–02
DRAM ADR 03–05 [IR 7–9 v111]
DRAM ADR 06–08 ←IR 10–12

As indicated on the figure, for I/O instructions, IR 3–9 is the device select code. If bits 3–6 are equal to zero, the device is local to the processor, i.e., in the EBox. Currently, there are six local devices:

APR: DEV 000
PI: DEV 004
PAG: DEV 010
CCA: DEV 014
TIM: DEV 020
MTR: DEV 024
(UNUSED: DEV 030)

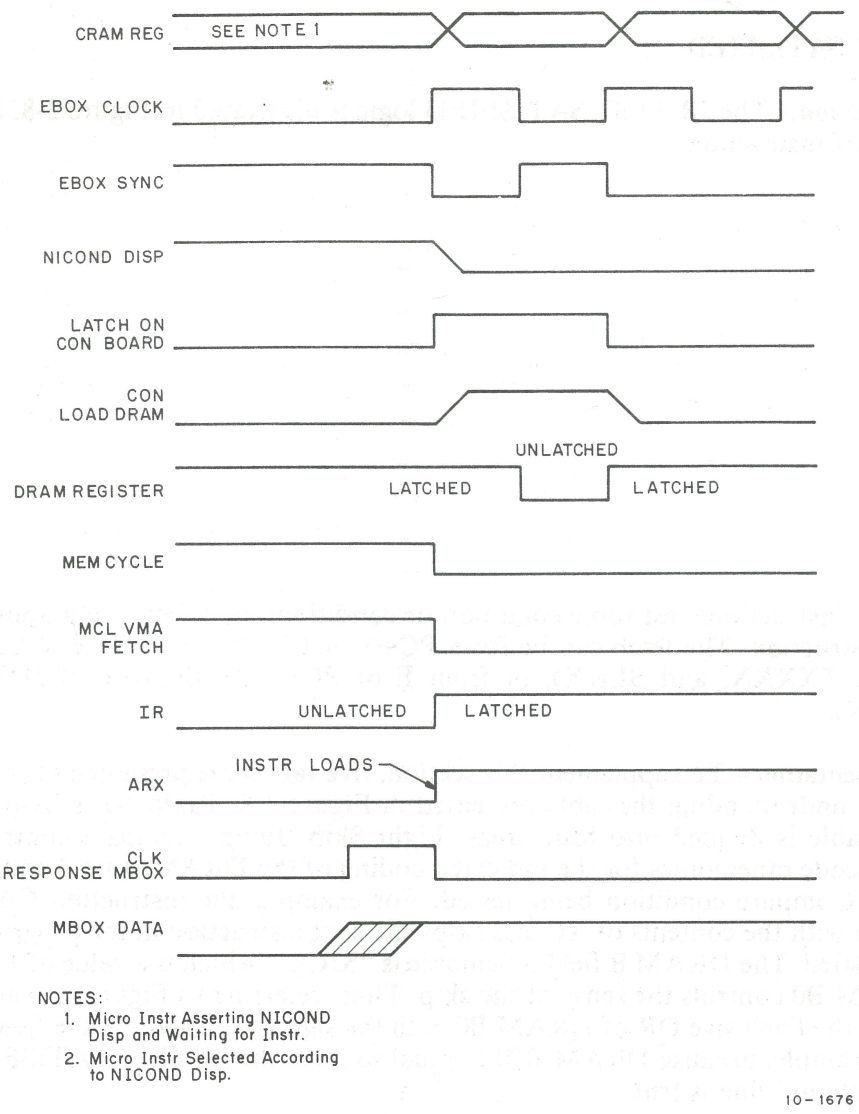


Figure 3-7 NICOND Dispatch and Waiting

If IR bits 3–6 are nonzero, the device is external to the processor. This includes device select codes 034 to 774.

All other op codes in the range of 000–677 address locations in the DRAM that correspond to locations 000–677. This is illustrated in Figure 3-2. DRAM address 00–02 is formed from IR 00–02, while DRAM address 03–08 is formed from IR 03–08.

AC decoded jumps JRST and JFCL reference locations in the DRAM that correspond to their numerical op codes (254 and 255, respectively). The DRAM register is loaded specially for JRST. Note that IR JRST (Figure 3-3) forces DRAM register J4 to zero while enabling DRAM J07–10 to be input from IR 09–12. This enables the microcode for JRST to be entered at the appropriate location relative to the type of code in IR 09–12.

DRAM register bits 00, 05, and 06 are missing in the hardware (Figure 3-3). This prevents DRAM J Dispatch from accessing certain CRAM locations.

3.1.3 IR TEST SATISFIED

3.1.3.1 Introduction – The IR TEST SATISFIED logic is illustrated in Figure 3-8. It is used with the following types of instructions:

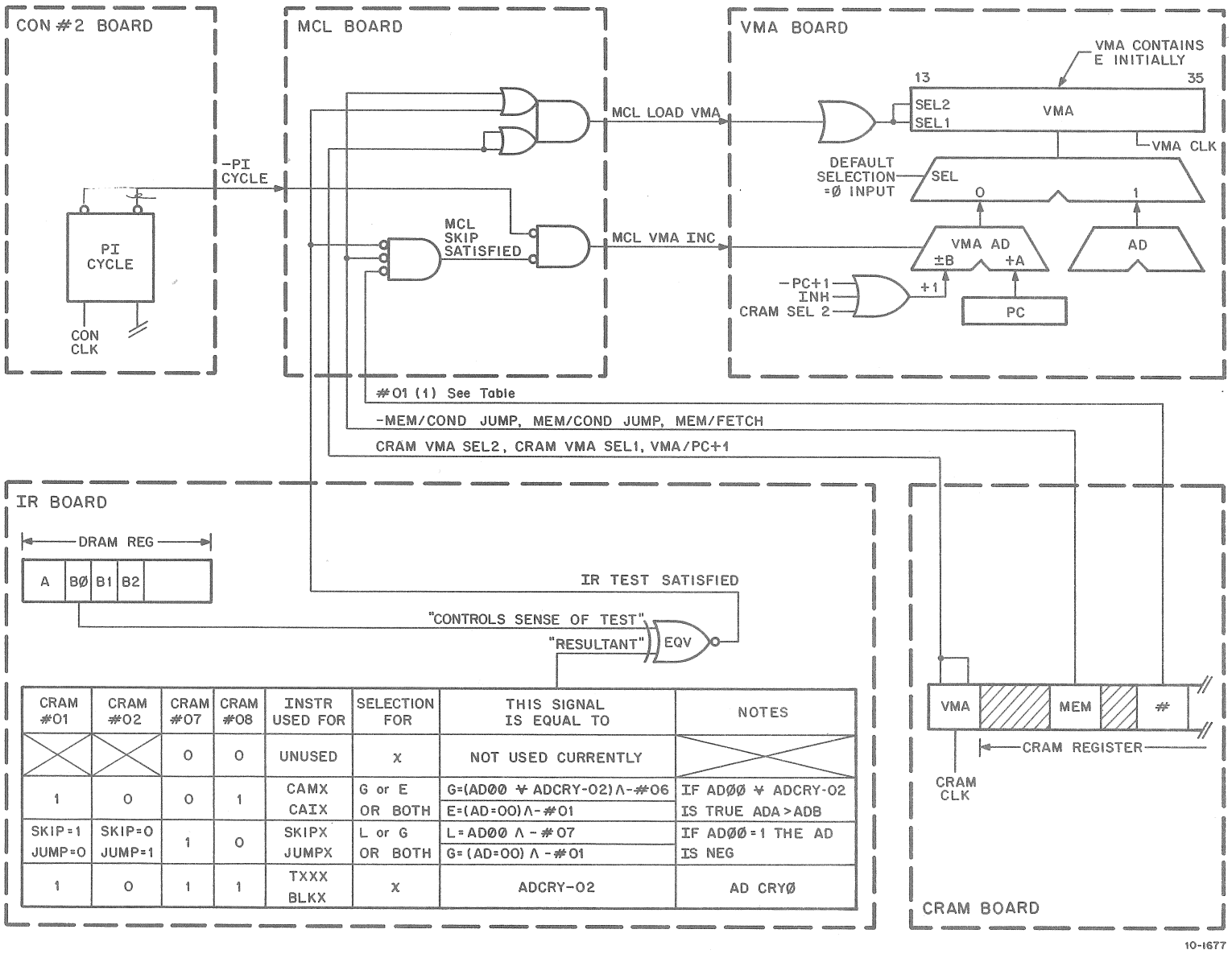
CAMXX
CAIXX
SKIPXX
JUMPXX
TXXXX
BLKX
AOSXX
SOSXX
AOJXX
SOJXX
AOBJX
JFCL

In general, these instructions test some condition or conditions and, depending upon the result of the test, fetch an instruction. The fetch can be from PC+1 or PC+2, (in the case of CAIXX, CAMXX, SKIPXX, AOS, TXXXX, and BLKX), or from E or PC+1 (in the case of JUMPXX, AOJXX, SOJXX, AOBJX).

3.1.3.2 Implementation – To supplement this section, five tables are presented (Tables 3-1 through 3-5), which aid in understanding the table presented in Figure 3-8. Table 3-1 is Skip, Jump, Compare controls. This table is divided into four areas. Eight Skip, Jump, Compare controls are indicated. These are microcode mnemonics for the indicated coding of the DRAM B field and imply the type of Skip, Jump, or Compare condition being tested. For example, the instruction CAIE compares the effective address with the contents of AC and skips the next instruction in the program sequence if the condition is satisfied. The DRAM B field mnemonic is "SJCE," which is a value of 1 in DRAM B. The coding of DRAM B0 controls the sense of the skip. Thus, referring to Figures 3-9 and 3-10, IR TEST SATISIFIED is the Exclusive OR of DRAM B0 with the signal indicated on the figure as "resultant." In the current example, because DRAM B00 is equal to zero, the IR TEST SATISFIED signal is true only if the "resultant" line is true.

As indicated in Figure 3-9, the combination of AD = 0 with DRAM B 01 (0) and CRAM #07(1) enables "resultant" to be true. This yields IR TEST SATISFIED. Referring to Figure 3-8, the VMA contains E, which it received at AREAD time. The VMA field function is PC+1 [CRAM VMA SEL 1 (0) ∧CRAM VMA SEL 2 (1)]. Because PC+1 INHIBIT is false at this time, the "B" input to VMA AD is equivalent to +1, while the VMA AD function is "A+B." The MEM field function is "FETCH," and the magic number field function is "COMP FETCH," which is coded as #201. Thus, #01 (1) with "FETCH" and IR TEST SATISFIED gives MCL SKIP SATISFIED. Providing PI CYCLE is clear, MCL VMA INC increments the VMA AD SUM, which is now PC+1, to a value of PC+2.

Note that either bit of the CRAM VMA field enables one side of the MCL VMA load gate and that IR TEST SATISFIED or -MEM/COND JUMP enables the other side. This is necessary to allow IR TEST SATISFIED to inhibit loading the VMA during Jump-type instructions. VMA contained the jump address prior to the test. Note that the magic number field function and MEM field function for Jump-type instructions is different than that for Skips and Compares. It is necessary to prevent PC+2 from occurring and this is accomplished by blocking the term MCL SKIP SATISFIED. Because the magic number field function for jumps, which is "JUMP FETCH," has #01 (0), the gate is inhibited. If the test is not satisfied, VMA loads with PC+1 and program operation continues.



CRAM #01	CRAM #02	CRAM #07	CRAM #08	INSTR USED FOR	SELECTION FOR	THIS SIGNAL IS EQUAL TO	NOTES
		0	0	UNUSED	X	NOT USED CURRENTLY	
1	0	0	1	CAMX CAIX	G or E OR BOTH	G=(ADØØ ⊻ ADCRY-02)∧-#06 E=(AD=00)∧-#01	IF ADØØ ⊻ ADCRY-02 IS TRUE ADA>ADB
SKIP=1 JUMP=0	SKIP=0 JUMP=1	1	0	SKIPX JUMPX	L or G OR BOTH	L=ADØØ ∧ -#07 G=(AD=00) ∧ -#01	IF ADØØ=1 THE AD IS NEG
1	0	1	1	TXXX BLKX	X	ADCRY-02	AD CRYØ

Figure 3-8 IR Test Satisfied

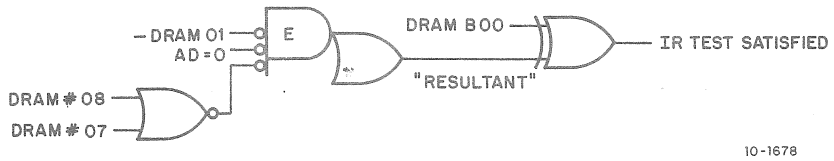


Figure 3-9 IR Test Equal

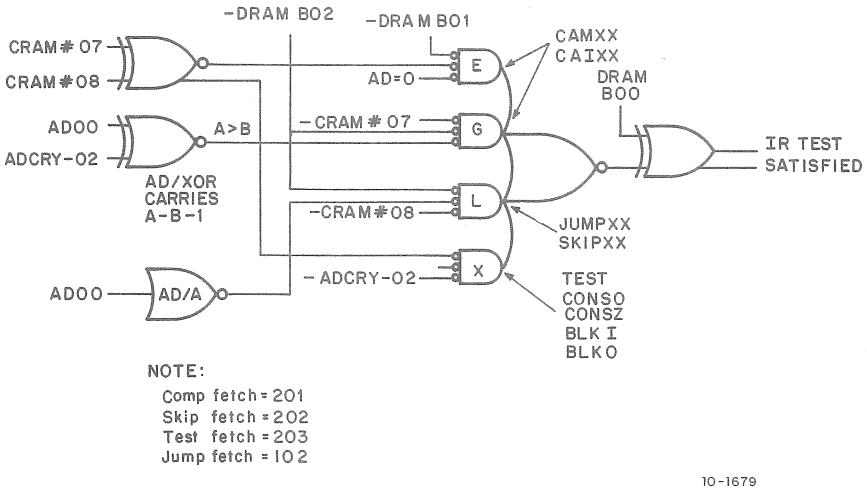


Figure 3-10 IR Test Satisfied Logic

Table 3-1 Skip, Jump, Compare Controls

DRAM B Field	Skip, Jump, Compare Controls	Controls Sense of Skips, Jumps, and Compares DRAM B00
3	SJC-	0
2	SJCL	0
1	SJCE	0
0	SJCLE	0
7	SJCA	1
6	SJCGE	1
5	SJCN	1
4	SJCG	1

NOTE
See Table 3-4; uses Skip or Jump fetch with various AD functions.

Table 3-2 Test Controls

DRAM B Field	Test Controls	Controls Sense of Test DRAM B00
4	TN-	1
0	TNE	0
0	TNA	0
4	TNN	1
5	TZ-	1
1	TZE	0
1	TZA	0
5	TZN	1
6	TC-	1
2	TCE	0
2	TCA	0
6	TCN	1
7	TO-	1
3	TOE	0
3	TOA	0
7	TON	1

NOTE
See Table 3-4; uses TEST fetch with various AD functions.

Table 3-3 CONSX and BLKX Controls

DRAM B Field	CONSX, BLKX Controls	Controls Sense of CONSX, BLKX, Skip DRAM B00	COND Causing Skip
2	BLKI	0	TEST FETCH TEST BRL
0	BLKO	0	TEST FETCH TEST BRL
5	CONSO	1	TEST FETCH TEST AR BR
1	CONSZ	0	TEST FETCH TEST AR BR

Table 3-4 Fetch Control Modifiers

Actual Instruction Using	Microinstruction Function	MEM Field	Magic No. Field	01	02	07	08
CAMXX, CAIXX	COMP FETCH	FETCH	201	1	0	0	1
SKIPXX	SKIP FETCH	FETCH	202	1	0	1	0
BLKO, BLKJ, CONSO, CONSZ, TXXXX	TEST FETCH	FETCH	203	1	0	1	1
JUMPXX	JUMP FETCH	FETCH	102	0	1	1	0

Table 3-5 CRY0 Generation (MACRO)

Instruction That Uses	CRY0 Generators Used	AD Field Function	Additional Signal
BLKI, BLKO	TEST BRL	ORCB+1	GEN CRY 18
CONSO, CONSZ	TEST AR·BR	CRY A·B#0	
TEST	TEST AR·ACO	CRY A·B#0	
TEST	NO CRY	SETCA	

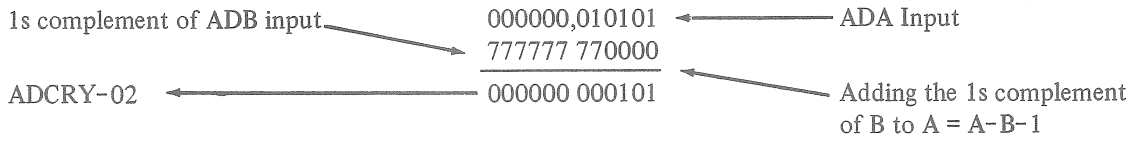
Figure 3-10 illustrates the actual logic that develops IR TEST SATISFIED. The use of the E, G, L and X portions is indicated. The result of the test in the AD determines one of the conditions on each gate. For Equal (E), the term is straightforward AD = 0. In the case of Greater (G), the Exclusive OR of the sign of AD (AD00) with a carry out of the AD sign (AD CRY –02) produces the A >B output when AD is performing the Exclusive OR function. For example, assume CAIG AC, 010101.

```
AR = 000000, 010101   ;O,E
AC = 000000, 007777   ;(AC)
```

The function performed in AD is:

```
ADB←FM; (AC)
ADA←AR; O, E
AD = XOR
```

Note that while the AD performs the logical function XOR, the carry function is A-B-1 (Table 2-8, ALU Functions). Therefore, the ADB input is 000000,007777 and the ADA input is 000000,010101. The operation is as follows:



Note that the following relation is true:

-B = $\bar{B}$+1
-B-1 = $\bar{B}$+1 -1
-B-1 = $\bar{B}$, which is the 1s complement of B.

XORing AD CRY -02 with AD00, which is 0, should indicate A >B.

For less than (L), the term is AD00, and this indicates the AD result as a negative value. Skips utilize the Boolean AD function A. Here, the carries function is really A-1. Thus, if the instruction is SKIP L 0, E, the contents of E are compared with zero and a SKIP occurs if (E) is any negative value. The implementation follows:

```
X: SKIPL 0, E
(E) = 777777, 777774 ; -4
AR = (E)
```

The function performed in AD is ADA ←AR, AD = A and effectively the (AR) is compared to zero because any negative value in AR satisfies the SKIP until a value of zero is placed in AR. This turns off AD00.

The remaining term (X) is used during TEST, BLKI, BLKO, CONSO, and CONSZ instructions. The AD carries function is AB-1. For example, assume the instruction is CONSO DEV, 1. At the time of the test, BR contains 000000,000001, the effective address, and AR contains the bits (if any) from the device. The implementation follows:

```
BR = 000000,000001      ;O,E
AR = 000000, 000001     ;assume the bit was set in the device
```

	000000,000001
"AND"	000000 000001
	000000,000001
For the carries function add -1	777777,777777
AD CRY -02 ←	000000,000000

Here ADCRY–02 inhibits the (X) function but DRAM B0 is coded to enable the IR TEST SATISFIED condition. The PC is updated by +2 and loaded into VMA (Figure 3-9). If the instruction were CONSZ DEV, 1 and the device flag was not set, the AD function [000000,000000-1] yields -1 and -AD CRY–02. This satisfies the (X) function and DRAM B0 is clear. Once again, the IR TEST SATISFIED condition is satisfied and the SKIP occurs.

3.2 PROCESSOR TIMING

The KL10 is a synchronous machine. Figure 3-10 illustrates the basic clock layout and distribution.

3.2.1 Clock Overview

The clock resides in the EBox and contains a selectable source (Figure 3-12). This source can be a crystal controlled 50 MHz oscillator, for normal processor operations, but may be an external source for special applications or a 56 MHz crystal-controlled oscillator for speed margining.

Basically, the clock consists of three other rather distinct sections: the clock control, the EBox clock control, and the clock diagnostic control labeled ① , ② , ③ , respectively, in Figure 3-13.

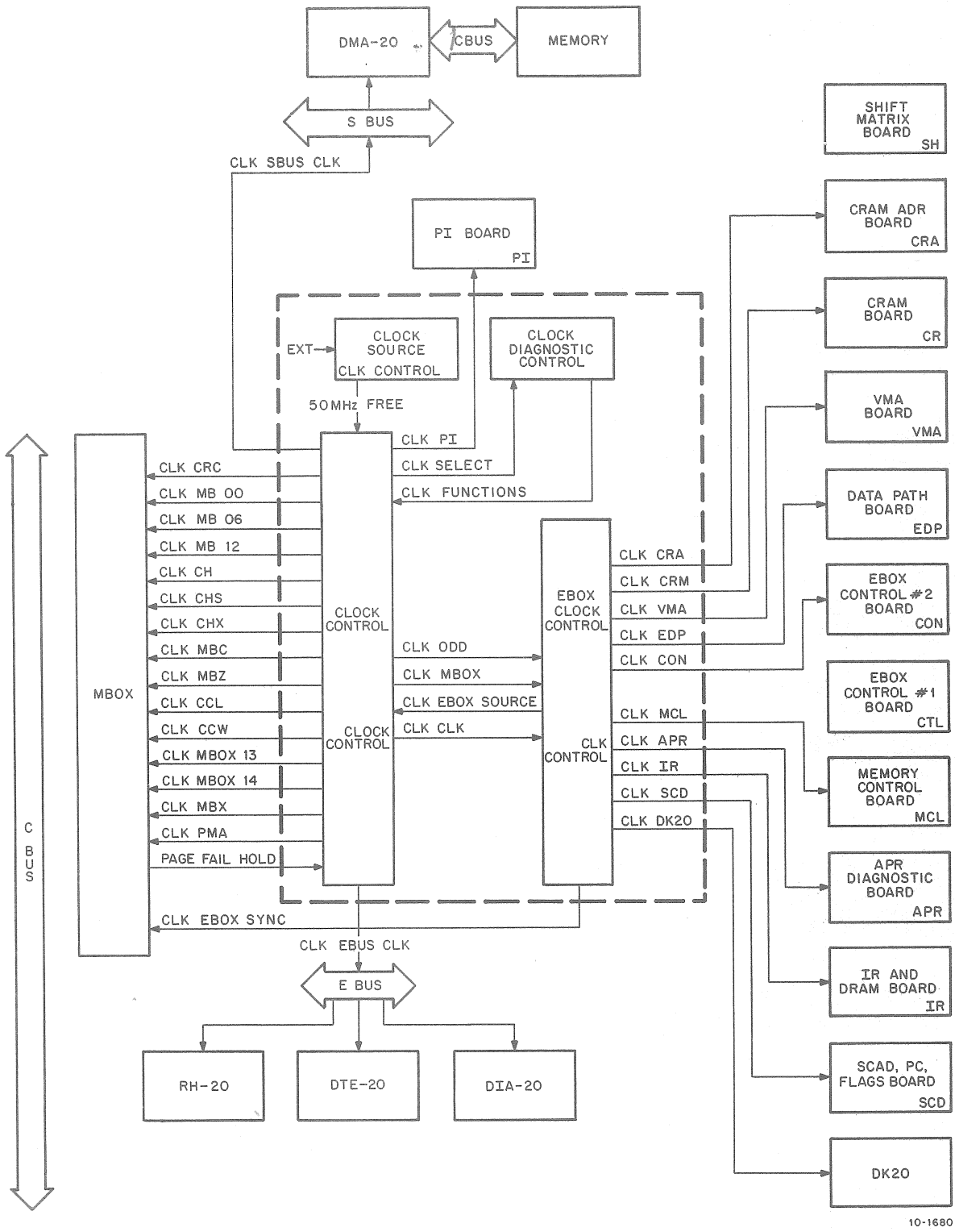


Figure 3-11 Clock Basic Block Diagram

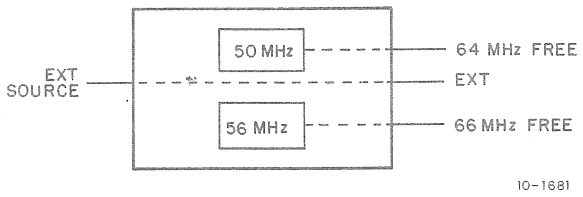


Figure 3-12 Clock Source Simplified

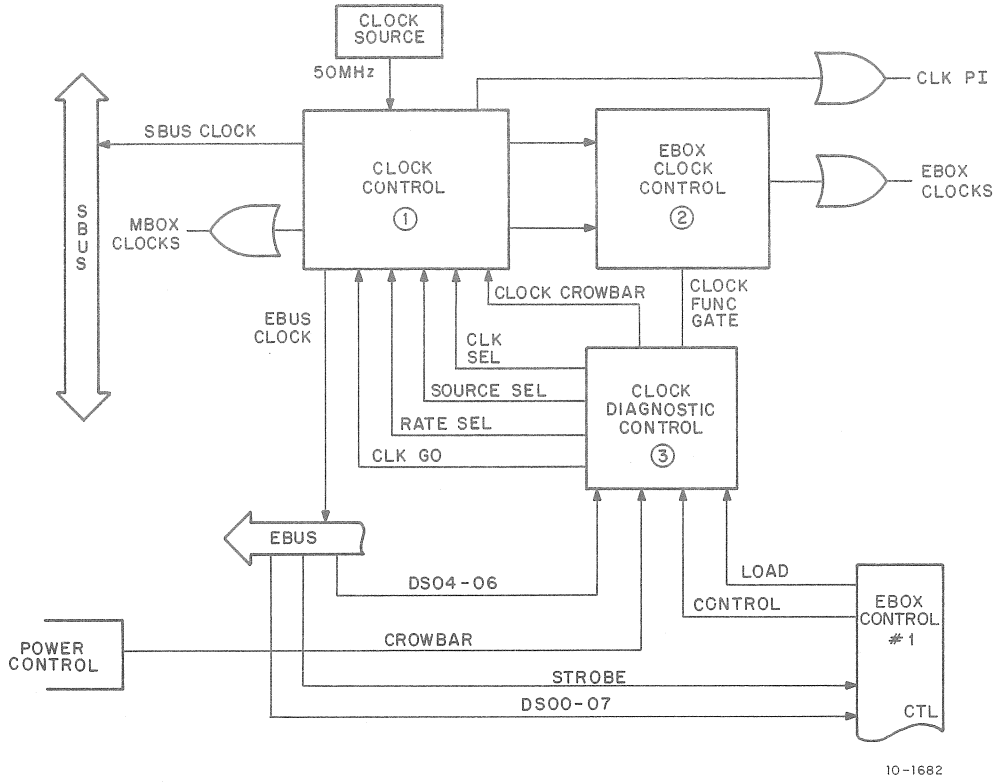


Figure 3-13 Basic Clock Block Diagram

3.2.2 Crobar and Clock Initialization

When the KL10 system is powered up, the EBox clock board must be initialized to a known state. In addition, the device controllers on the EBus must be initialized and a series of MBox, EBox, SBus, and EBus clocks must be generated for various initializtion purposes. First, the power controller asserts CROBAR for approximately 5 seconds. This signal is passed to the clock diagnostic control logic, where it enables the initialization process. The clock diagnostic logic contains a 2-bit source selection register, a 2-bit rate selection register, and various other registers and logic. During power up, the state of these registers is undefined. To avoid an improper source selection, the clock CROBAR signal is used directly to select the 50-MHz oscillator as the clock source to be used during the power up initialization phase (Figure 3-14).

The selected 50-MHz source is now divided down as indicated in Figure 3-15 to provide 25-MHz, 12.5-MHz, and 6.25-MHz free-running clocks.

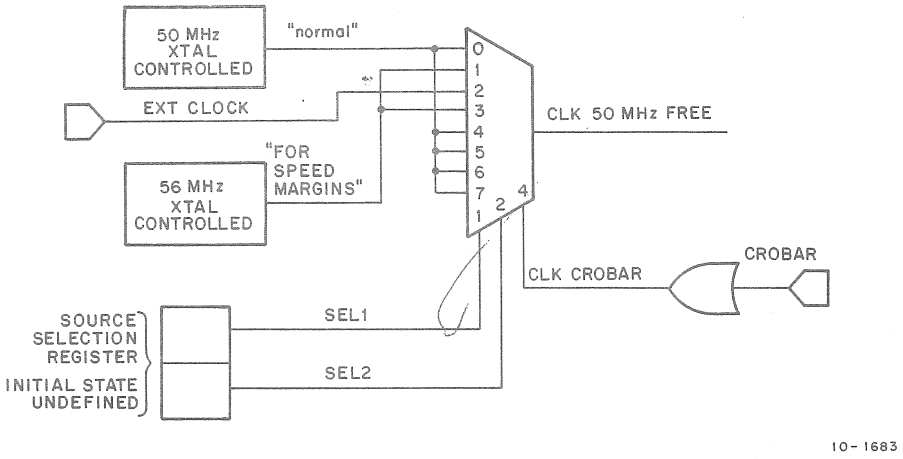


Figure 3-14 Basic Source Selection

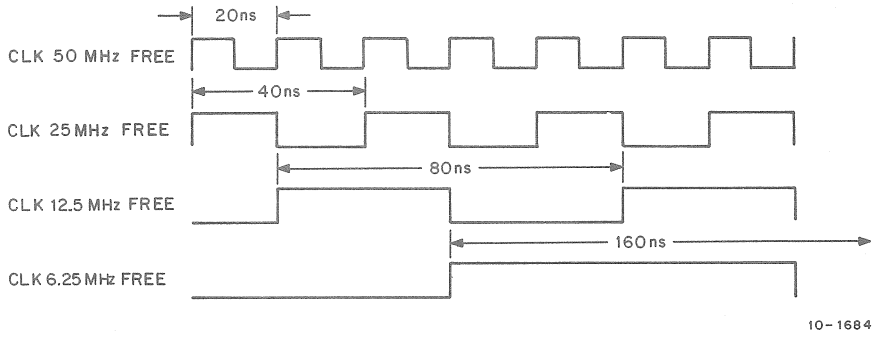
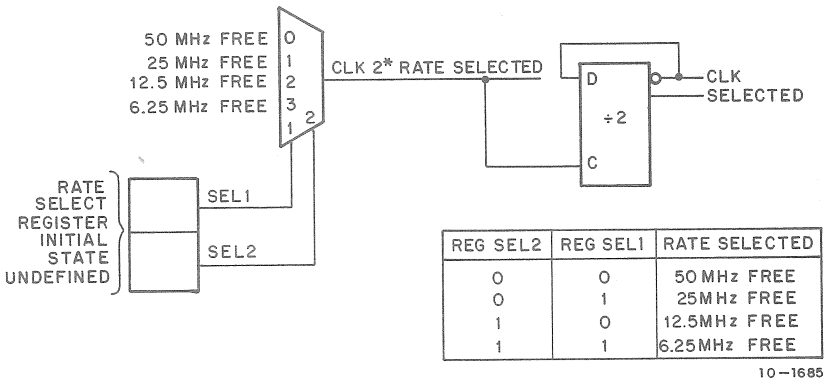


Figure 3-15 Free-Running Clocks

The 50 MHz FREE clock source is next passed to a rate-selectable mixer. However, because the Rate register may initially be in an undefined state, the selected rate is apt not to be the 50 MHz source. This presents no problem because the inputs to the mixer (50 MHz FREE, 25 MHz FREE, 12.5 MHz FREE, or 6.25 MHz FREE) are all even multiples; the rate is not critical during the power up phase of operation. The mixer is shown in Figure 3-16. Its output is labeled 2*Rate Selected, and this output is twice the clock selected frequency.



REG SEL2	REG SEL1	RATE SELECTED
0	0	50 MHz FREE
0	1	25MHz FREE
1	0	12.5MHz FREE
1	1	6.25MHz FREE

Figure 3-16 Basic Rate Selection

3.2.3 EBus Reset

Referring to Figure 3-18, the CLK CROBAR signal enables the counter to subtract one on each 12.5 MHz clock pulse. Once again, the initial state of the counter is undefined. During the crobar period (approximately 5 seconds), the counter is decremented toward zero. When zero is reached, a carry is generated and if CROBAR is false at this time, the -1 function is disabled and the counter is loaded with zeros. This removes ~~EBUS RESET~~. In practice, the counter passes through zero many times until finally CROBAR is removed by the Power Controller logic. Signal EBUS RESET is a 1280 ns square wave.

3.2.3.1 Initialization Clock Pulse Generation – As shown in Figure 3-18, CROBAR is shifted four places into the shift register, activating the CLK SS stage. This, with the Clock Selected flip-flop, enables the gated clock. It is this signal (GATED CLK) that becomes the source of the clocks generated via the clock control and EBox Clock Control. When CROBAR is removed, 4 CLK selected pulses later, CLK SS is also removed. The approximate sequence is indicated in Figure 3-17. Figure 3-19 shows the power up timing. Note that this shift register also serves to synchronize CROBAR.

3.2.4 EBox Clock Control

The EBox Clock Control provides a source of clocks for the EBox boards together with an MBOX Sync Point (EBOX SYNC), which is always asserted one MBOX Clock prior to the generation of the EBox clock (Figure 3-20).

Depending upon the nature of the EBox cycle (a period extending from the rising edge of one EBox clock to the rising edge of the next), the period between EBOX CLOCK pulses may be extended by some multiple of 40 ns, i.e., 80, 120, 160, 200, etc.

Refer to Figure 3-22; this drawing illustrates the functional structure of the EBOX CLOCK Control. It consists of an MBOX CLOCK counter/marker generator, a clock phase sync detector, an EBox sync source, and an EBox clock source. The CRAM time field (T00, T01) specifies the duration of the EBox cycle (Figure 3-21).

The marker generator consists of a shift register that may be loaded with zeros when EBOX CLK EN is true or have ones shifted in (beginning with the 40-ns stage) for each MBOX CLK generated, as long as EBOX CLK is false. Table 3-6 describes the marker generator.

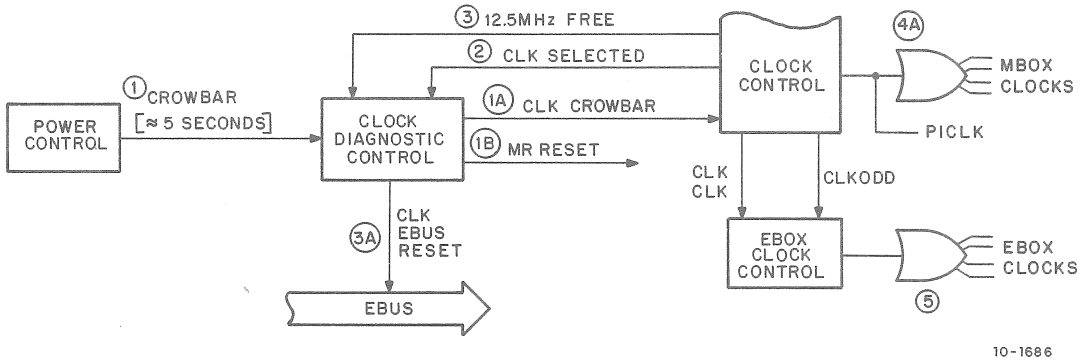


Figure 3-17 Clock Initialization

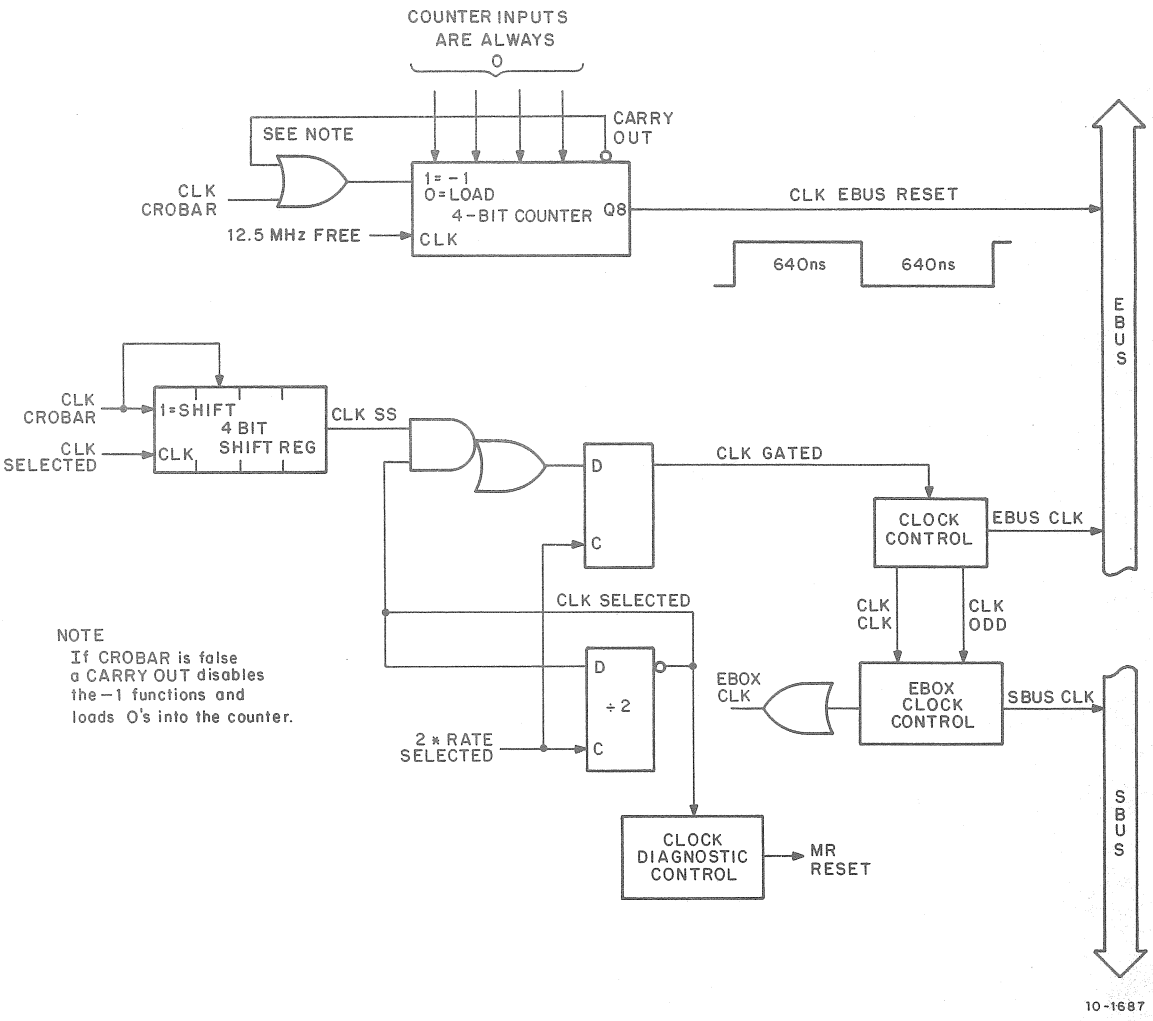


Figure 3-18 EBus Reset and Clock Initialization

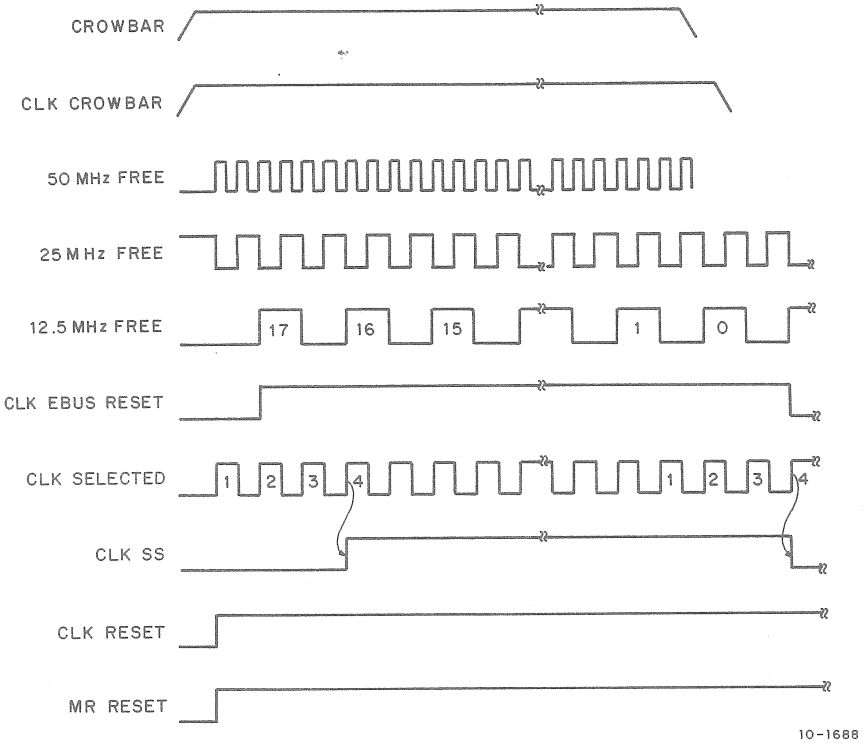


Figure 3-19 Power Up Timing

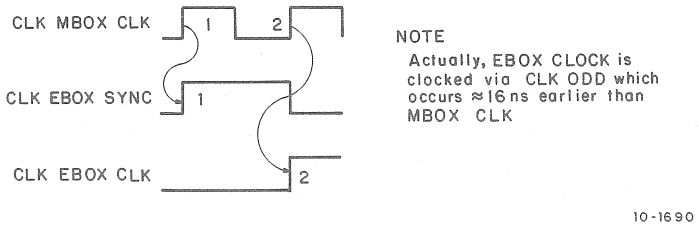


Figure 3-20 Simplified Diagram, MBox Clock, Sync, EBox Clock

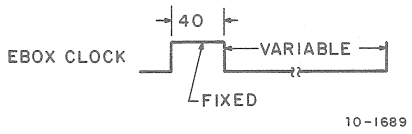


Figure 3-21 EBox Cycle

Table 3-6 Marker Generator Function

T00	T01	Duration	MBOX CLK	Marker Generator			EBOX CLK EN	EBOX CLK	EBOX SYNC
				40 ns	80 ns	120 ns			
0	0		1	0	0	0	0	1	0
0	0	80	2	1	0	0	1	0	1
0	1		1	0	0	0	0	1	0
0	1		2	1	0	0	0	0	0
0	1	120	3	1	1	0	1	0	1
1	0		1	0	0	0	0	1	0
1	0		2	1	0	0	0	0	0
1	0		3	1	1	0	0	0	0
1	0	160	4	1	1	1	1	0	1
1	1		1	0	0	0	0	1	0
1	1		2	1	0	0	0	0	0
1	1		3	1	1	0	0	0	0
1	1		4	1	1	1	0	0	0
1	1	200	5	1	1	1	1	0	1
x	x		1	0	0	0	0	1	0

The clock phase sync detector compares the marker generator content with the CRAM time field (loaded at EBOX CLOCK TIME) whenever EBOX CLOCK EN is false. If the marker count compares with the bit combination in the time field, SYNC EN is asserted and the next MBox clock sets EBOX SYNC. EBOX SYNC then enables EBOX CLOCK EN and similarly disables the detector. This completes one cycle.

Note that with MBOX WAIT true, -EBOX CLK EN is also true and EBOX CLK EN is false (Figure 3-22). This enables the MBox clock counter/marker generator to keep shifting 1s from the 40-ns stage toward the 120-ns stage. Similarly, the detector is enabled and when the marker compares with the bit combination in the time field of the CRAM word, SYNC EN will be asserted and remain so until the MBox responds or aborts the cycle. Thus, one MBOX CLK after SYNC EN is asserted, EBOX SYNC will set. In other words, EBOX SYNC is asserted one MBOX CLOCK prior to where EBOX CLOCK would have been asserted.

With SYNC EN true when MBox response is received (Figure 3-22) EBOX CLOCK EN becomes true allowing the marker to reset to 000, and SYNC EN is removed allowing EBOX SYNC to clear on the next MBOX CLOCK. At the same time, EBOX CLK EN becomes true and EBOX SOURCE EN is also true; thus, when EBOX SYNC is cleared, EBOX CLOCK sets (Figure 3-23).

3.2.5 Error Detection

Figure 3-24 illustrates the logic that stops all clocks in the event of any of the following:

1. A DRAM parity error occurs.
2. A CRAM parity error occurs.
3. A fast memory parity error occurs.

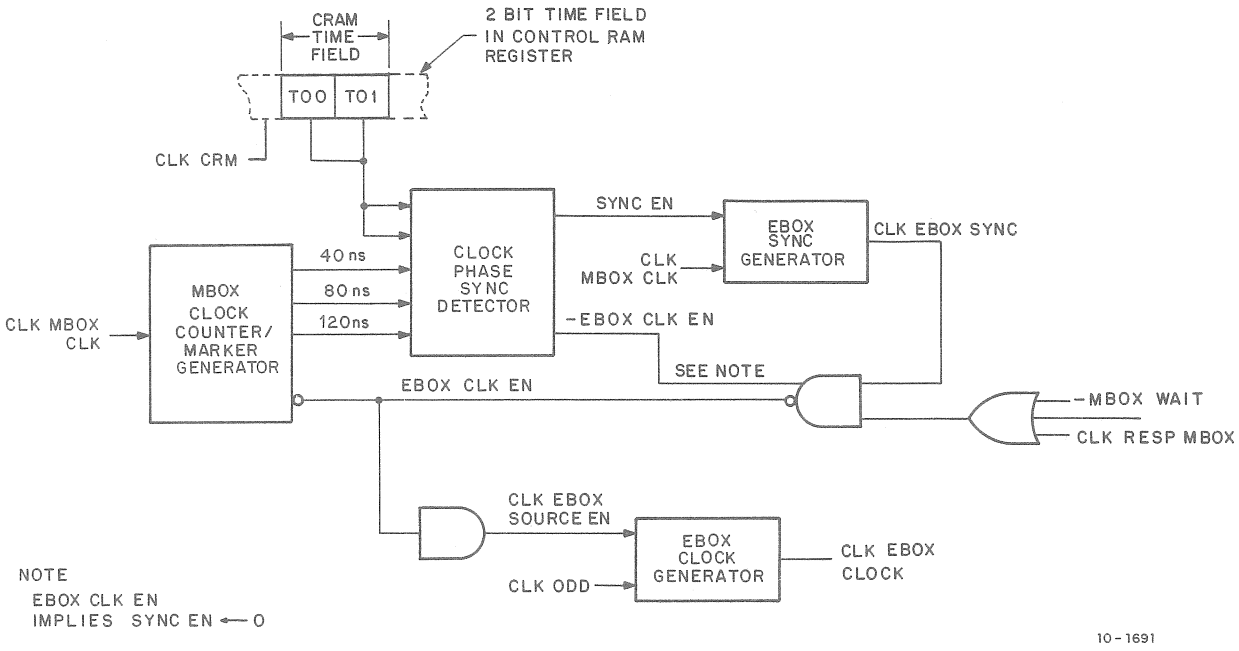


Figure 3-22 EBox Clock Control Block Diagram

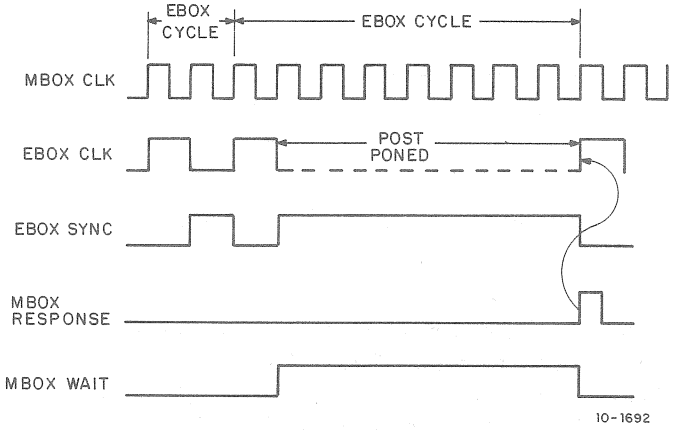


Figure 3-23 Basic MBox Cycle Timing

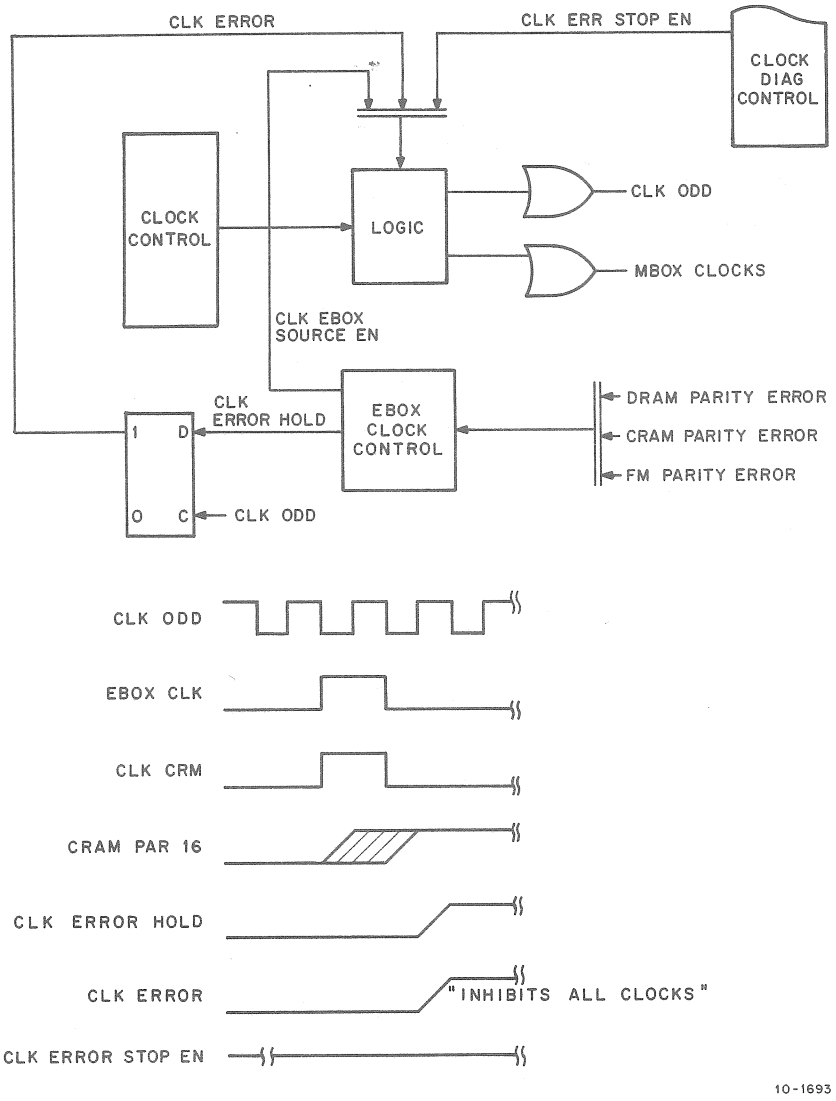


Figure 3-24 Clock Error Stop

The timing shown is for a CRAM parity error. The CRAM register is clocked by CLK CRM; sometime later, the parity network settles and asserts -CRAM PAR 16. This indicates that the CRAM word has dropped or picked up bits and is not correct. The signal -CRAM PAR 16, together with an enable previously set by a diagnostic cycle (CLK CRAM PAR CHECK), enables the generation of CLK ERROR HOLD.

If it is desired to stop on parity errors, CLK ERROR STOP EN must have been set by the console. In this case, on the next occurrence of CLK EBOX SOURCE EN, the CLK ODD gate will be latched false, inhibiting all clocks and freezing the system.

3.2.6 Clock Control Logical and Skew Delays

Figure 3-25, illustrates the delays necessary to assure that the proper timing relationship exists between the actual MBOX CLOCKS, EBOX CLOCKS, and the sampling of the CRAM time field. The lumped delay of ≈128 ns consists of fixed logic delays, gate and wire delays. The output is CLOCK ODD and is used to clock a 10141 Shift register, which has a propagation delay of ≈2.65 ns.

NOTE

The times given here are approximate times only.

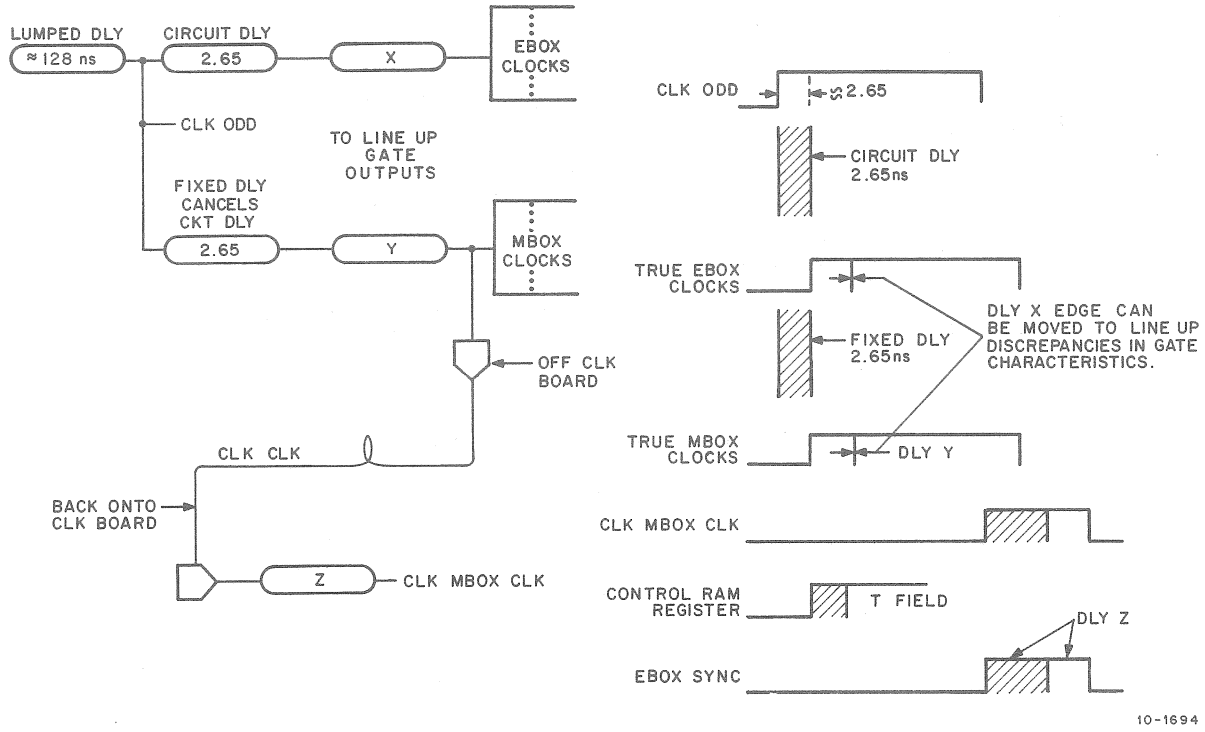


Figure 3-25 Logical Delays and Skew

The output of the Shift register feeds various gates and the various EBox boards receive their clocks from these gates. Delay X allows for lining up the outputs of the gates, "deskewing" the EBox clocks.

The delays are actually etch paths near the fingers on the board and once the delay has been ascertained, a permanent connection is made at the proper point. Figure 3-26 shows the EBox clock fanout; Figure 3-27 shows the MBox clock fanout.

To cancel the effect of the 10141 circuit propagation delay, a fixed 2.65 ns have been inserted in the path between the lumped DLY and the MBOX CLOCKS. Connected in this path also is DLY Y, which performs the same function as DLY X does for the EBOX CLOCKS.

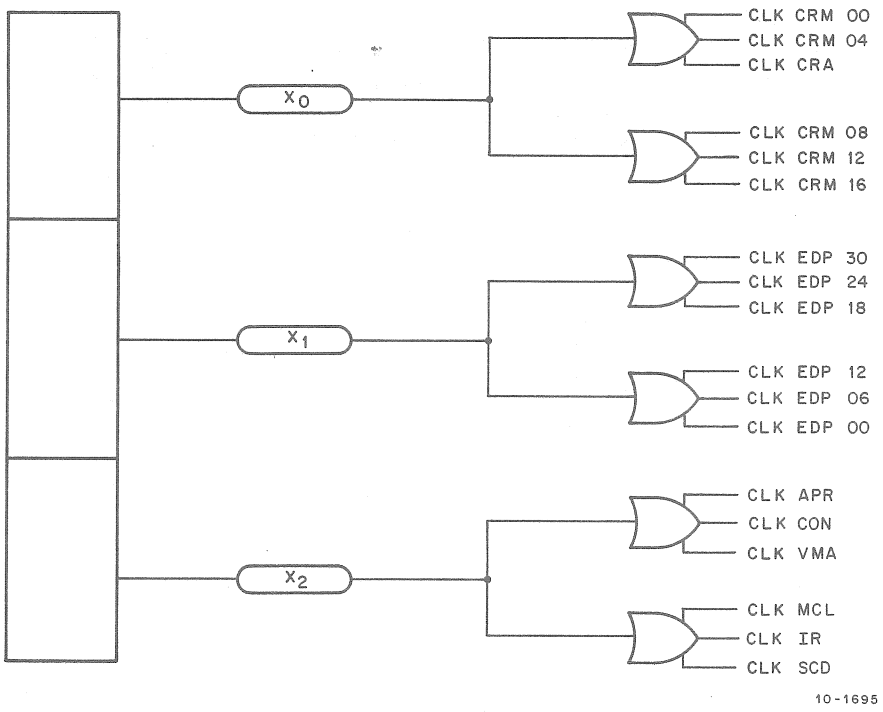


Figure 3-26 EBox Clock Fanout

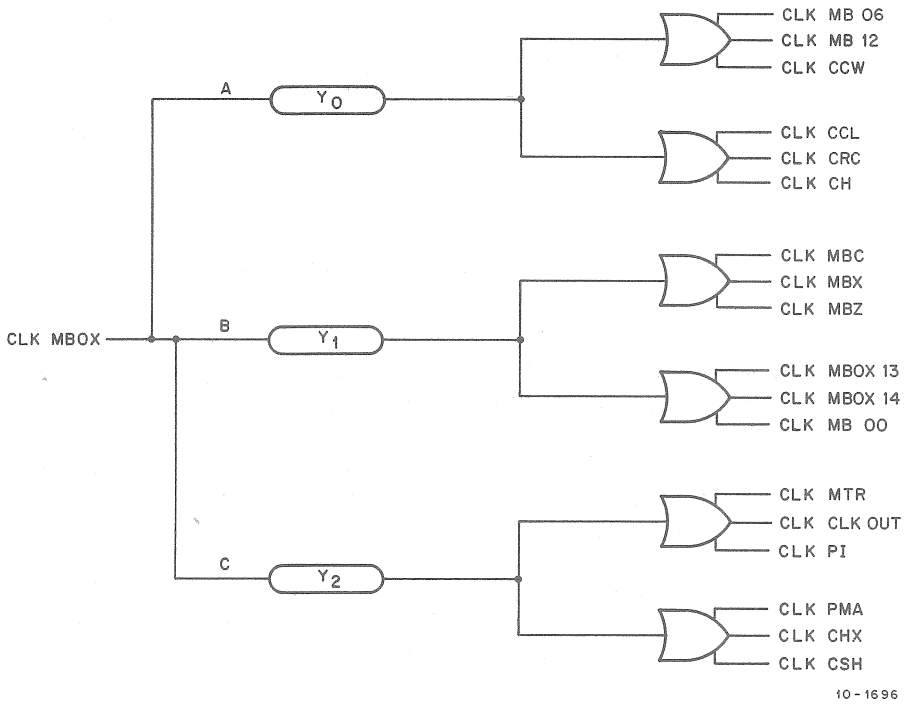


Figure 3-27 MBox Clock Fanout

All the EBOX CLOCKS and the MBOX CLOCKS are lined up leaving the clock board. In order to synchronize the CLK BOARD with the other boards, CLK CLK is passed out through the etch connection on the board. It then reenters the board at DLY 2 where it is deskewed via a coaxial cable, as are all the other CLK signals.

Figure 3-28 illustrates the basic timing for the clock board. Six basic cycles are presented: clock start-up, EBox cycle T = 01_2, EBox cycle T = 10_2, EBox cycle including a memory cycle T = 00_2, EBox cycle T = 00_2 and finally EBox cycle including a memory cycle and a page fault.

3.3 ARITHMETIC PROCESSOR FACILITY

3.3.1 Introduction

This facility controls and contains logic relating to the following hardware in the EBox.

Address Break Facility
Arithmetic Processor Status
Processor Identification
Cache Refill RAM Facility
MBox Error Address Register
Fast Memory Addressing and Control

These areas are set up via four KL10 instructions as follows:

DATAO APR – Sets up address break facility.

CONO APR – Sets selected flags in the APR STATUS REG, and/or enables interrupts to occur on selected APR priority interrupt channel.

APRID – Reads the following information from the EBox:

Microcode options
Microcode version number
Hardware options
Processor serial number

RDERA – Reads the ERA register located in the MBox

3.3.2 Address Break

One possible use of this hardware in the EBox is associated with the SET BREAK command, which may be issued to the monitor by a user (e.g., during the debugging process). This is primarily useful when the program that is being debugged:

1. Will not fail when DDT has been loaded

2. Destroys DDT when DDT is loaded

3. Destroys the contents of a memory location at an unpredictable point during program execution.

It is possible to break when the specified location is read from, written into, and/or fetched. It is also possible to break on monitor references to items in the user's address space.

Figure 3-29 contains the address break logic. A break may occur at three places in an instruction:

On Instruction FETCH
On DATA FETCH
On DATA WRITE

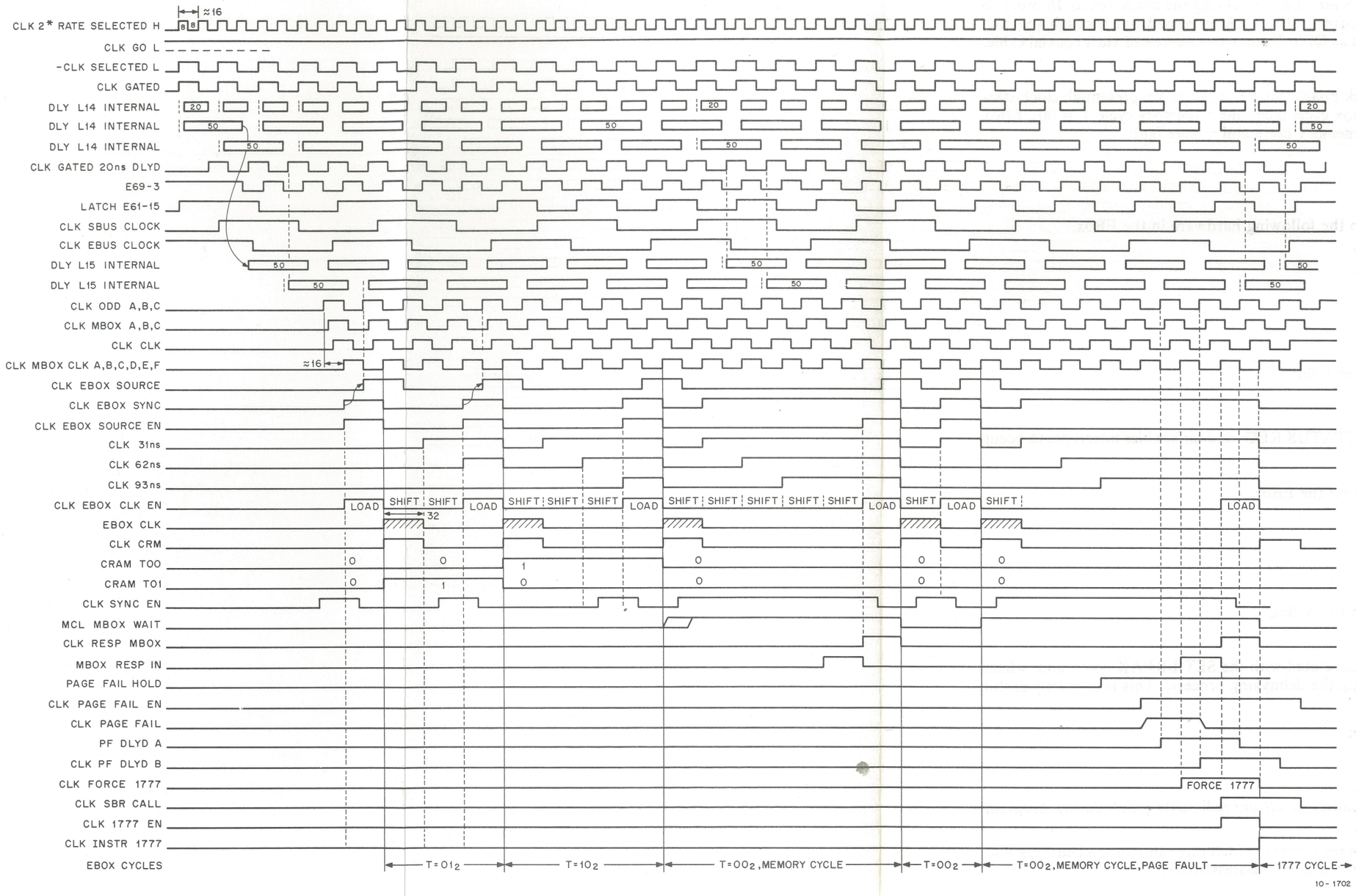


Figure 3-28 Clock Control,
EBox Clock Control Timing

All the EBOX CLOCKS and the MBOX CLOCKS are lined up leaving the clock board. In order to synchronize the CLK BOARD with the other boards, CLK CLK is passed out through the etch connection on the board. It then reenters the board at DLY 2 where it is deskewed via a coaxial cable, as are all the other CLK signals.

Figure 3-28 illustrates the basic timing for the clock board. Six basic cycles are presented: clock start-up, EBox cycle T = 01_2, EBox cycle T = 10_2, EBox cycle including a memory cycle T = 00_2, EBox cycle T = 00_2 and finally EBox cycle including a memory cycle and a page fault.

3.3 ARITHMETIC PROCESSOR FACILITY

3.3.1 Introduction

This facility controls and contains logic relating to the following hardware in the EBox.

Address Break Facility
Arithmetic Processor Status
Processor Identification
Cache Refill RAM Facility
MBox Error Address Register
Fast Memory Addressing and Control

These areas are set up via four KL10 instructions as follows:

DATAO APR – Sets up address break facility.

CONO APR – Sets selected flags in the APR STATUS REG, and/or enables interrupts to occur on selected APR priority interrupt channel.

APRID – Reads the following information from the EBox:

Microcode options
Microcode version number
Hardware options
Processor serial number

RDERA – Reads the ERA register located in the MBox

3.3.2 Address Break

One possible use of this hardware in the EBox is associated with the SET BREAK command, which may be issued to the monitor by a user (e.g., during the debugging process). This is primarily useful when the program that is being debugged:

1. Will not fail when DDT has been loaded

2. Destroys DDT when DDT is loaded

3. Destroys the contents of a memory location at an unpredictable point during program execution.

It is possible to break when the specified location is read from, written into, and/or fetched. It is also possible to break on monitor references to items in the user's address space.

Figure 3-29 contains the address break logic. A break may occur at three places in an instruction:

On Instruction FETCH
On DATA FETCH
On DATA WRITE

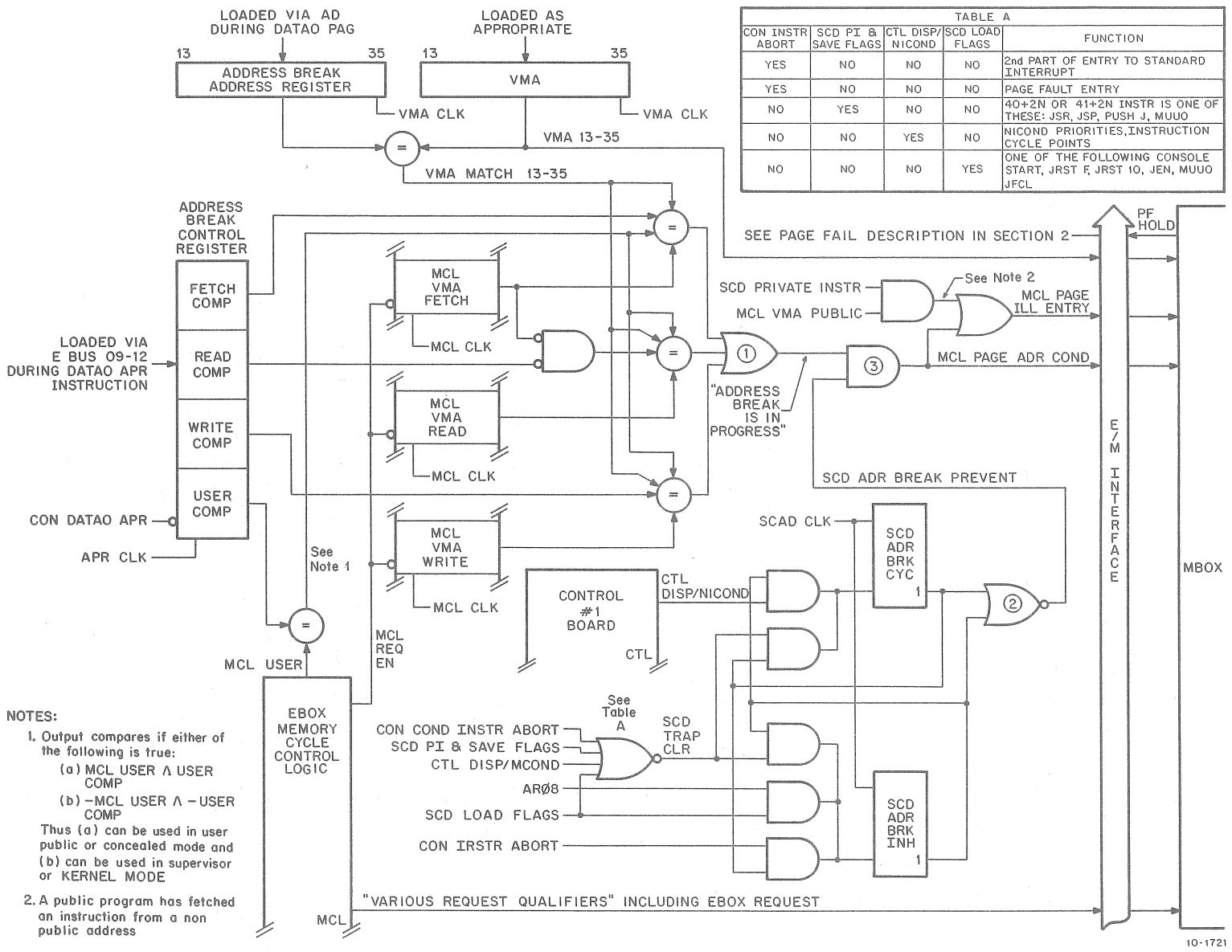


TABLE A

CON INSTR ABORT	SCD PI & SAVE FLAGS	CTL DISP/ NICOND	SCD LOAD FLAGS	FUNCTION
YES	NO	NO	NO	2nd PART OF ENTRY TO STANDARD INTERRUPT
YES	NO	NO	NO	PAGE FAULT ENTRY
NO	YES	NO	NO	40+2N OR 41+2N INSTR IS ONE OF THESE: JSR, JSP, PUSH J, MUUO
NO	NO	YES	NO	NICOND PRIORITIES, INSTRUCTION CYCLE POINTS
NO	NO	NO	YES	ONE OF THE FOLLOWING CONSOLE START, JRST F, JRST 10, JEN, MUUO JFCL

NOTES:

1. Output compares if either of the following is true:
   (a) MCL USER Λ USER COMP
   (b) −MCL USER Λ −USER COMP
   Thus (a) can be used in user public or concealed mode and (b) can be used in supervisor or KERNEL MODE
2. A public program has fetched an instruction from a non public address

Figure 3-29 Address Break Facility

In addition, the reference may be further qualified to a user or executive reference. The address break conditions are loaded into the EBox hardware by performing a DATAO APR instruction. The left half of (E) specifies the following:

Bit 09: Address Break on FETCH
Bit 10: Address Break on DATA READ
Bit 11: Address Break on DATA WRITE
Bit 12: Address Break on USER REF

The right half of (E) specifies the break address in bits 13–35, where 13–17 represents the virtual section number and 18–35 the virtual page number, line number.

The Address Break Inhibit logic, illustrated in Figure 3-29, may be set up to inhibit an address break by performing any of the following instructions:

JRSTF – JRST2
JEN – JRST 12
JRST 10
MUUO

The PC word provided by these instructions must have bit 8 = 1 to set SCD ADR BRK INH. If a JRSTF is given setting SCD ADR BRK INH, the NICOND Dispatch occurring during the JRSTF transfers the set state of SCD ADR BRK INH into SCD ADR BRK CYC, while clearing ACD ADR BRK INH. Therefore, for the duration of the next instruction, address breaks cannot occur. This is useful, for example, when continuing from an address which subsequently caused an address break. Consider the following example:

```
677/    SETO 3,          ;PUT -1 IN AC3
700/    ADDM 3,300       ;ADD TO TABLE
701/    AOS 700          ;ADD 1 TO TABLE ADR
702/    HRRZ 4,700       ;PUT CURRENT TABLE
703/    CAIE 4,1000      ;ADR IN AC4
704/    JRST 700         ;WHEN IT IS 1000 ALL DONE
```

NOTE

This sample program illustrates the use of ADR BRK INH and is not meant to be a well-structured program.

The sample program adds -1 to a table beginning at location 300_8 and ending at location 1000_8. A bug exists, however, in this program. Note that the AOS instruction in location 701 is incrementing the table address in the right half of location 700. The problem occurs when the right half of the instruction in 700 becomes 700. At this time, the instruction becomes ADDM 3,700 and this wipes out the instruction in location 700. Several references to location 700 are in the program. First the monitor is requested from a terminal to set ADR break on data write for address 700 to assure that the AOS instruction is working correctly, i.e., attempting a write into 700. The monitor performs a DATAO APR, which sets USER COMP, WRITE COMP, and loads the address break register with 700. At this time, ADR BRK INH is clear and when the EBox performs the write request, the comparator will satisfy the OR gate labeled ① because the following conditions are true:

1. VMA 13–35 = ADR BRK register 13–35
2. MCL VMA WRITE = WRITE COMP
3. MCL VMA USER = USER COMP

At this time, both SCD ADR BRK INH and SCD ADR BRK CYC are clear; therefore, the signals MCL PAGE ADR COND and MCL PAGE ILL ENTRY are asserted together with all other necessary request qualifiers. The MBox detects this condition and places a page fail word in its EBus register (indicating an address break page failure) and asserts PF HOLD to the EBox. The EBox senses this, and enters the microcode page fault handler. Now the EBox flags must be gathered for storage in user process table location 501. Because SCD ADR BRK INH is one of the processor flags, it must be made available; however, at this time it is clear. Regardless of this, the process of obtaining this flag will be discussed. Upon entry to the microcode, CON INSTR ABORT is generated to cause proper termination of the faulting instruction. Referring to Figure 3-29, CON INSTR ABORT enables SCD TRAP CIR, which breaks the recirculation paths for both SCD ADR BRK INH and SCD ADR BRK CYCLE; it also transfers the state of SCD ADR BRK CYC into SCD ADR BRK INH. This makes the flag available for storage in 501. The page fault handler reads the MBox EBus register and stores a page fail word in user process table location 500, stores the flags PC word (PC is now 701) in 501 and then fetches a new PC word from user process table location 502. The processor now enters Execute mode and handles the page failure appropriately.

Eventually, after evaluating the page fault word in 500 and other data, the monitor informs the user at his terminal that a write was attempted to location 700. If after giving the problem some thought, the user requests a break on the same address for write but now suspects that somehow the instruction in 700 is being overwritten by itself, the break can be inhibited. Now the monitor wishes to continue the program by performing the entire AOS instruction to ascertain that it works but also must avoid thewrite page fault associated with this instruction.

The monitor can perform a JRSTF instruction that sets ADR BRK INH and restores the old PC of 701 for the AOS instruction via user process table location 501. Referring to Figure 3-29, during the execution portion of JRSTF, SCD LOAD flag sets SCD ADR BRK INH. During the JRSTF instruction NICOND Dispatch occurs and transfers the set state of SCD ADR BRK INH into the BRK CYCLE flip-flop while clearing SCD ADR BRK INH. The AOS instruction is successfully fetched from 701 and the "AOS write reference" to 700 is prevented from causing MCL PAGE ADR COND because this is blocked by SCD ADR BREAK COND (L). The next NICOND Dispatch clears SCD ADR BRK CYCLE, enabling the ADR BREAK to occur if a write is performed to 700. Eventually, through many tries, the overwrite of the instruction in 700 will be detected by this method. Note this is only a simple example and is not necessarily a practical one.

3.3.2.1 Address Break INH and Saving Flags – The signal CON COND INSTR ABORT is generated by the microcode whenever external conditions require the microcode to abort a partially completed instruction. If this occurs during an address break cycle, this signal copies the state of SCD ADR BRK CYC back into SCD ADR BRK INH, thus making it available to save as a bit in the flag's PC word.

3.3.2.2 Address Break INH and Loading Flags – SCD LOAD FLAGS can be generated in a number of ways: JRSTF, JRST10, JEN, JRST, and MUUO can set SCD ADR BRK INH. The 10-11 interface can place the flags PC word in AR and perform a console start. This causes the microcode to generate SCD LOAD FLAGS. During a JFCL instruction, the flags are read and the specified flags cleared. Then the microcode reloads the flags using the signal SCD LOAD FLAGS.

3.3.3 Arithmetic Processor Status Register

This facility enables special internal conditions to signal the monitor on a priority interrupt channel assigned to the processor. Condition I/O instructions are used to control the appropriate flags and to inspect the conditions of interest.

The arithmetic processor status register consists of two 8-bit registers and associated control logic. One register receives the error or status signals and the other register enables or inhibits the generation of an interrupt when one or more of these error or status flags sets.

Figure 3-30 provides the basic format for the CONO APR word, the basic organization of the error or status flag and the interrupt enable or inhibit for the two registers. In addition, the bit assignments are provided in two tables, as well as the source of the error or status signals available to set the appropriate flags in the APR register.

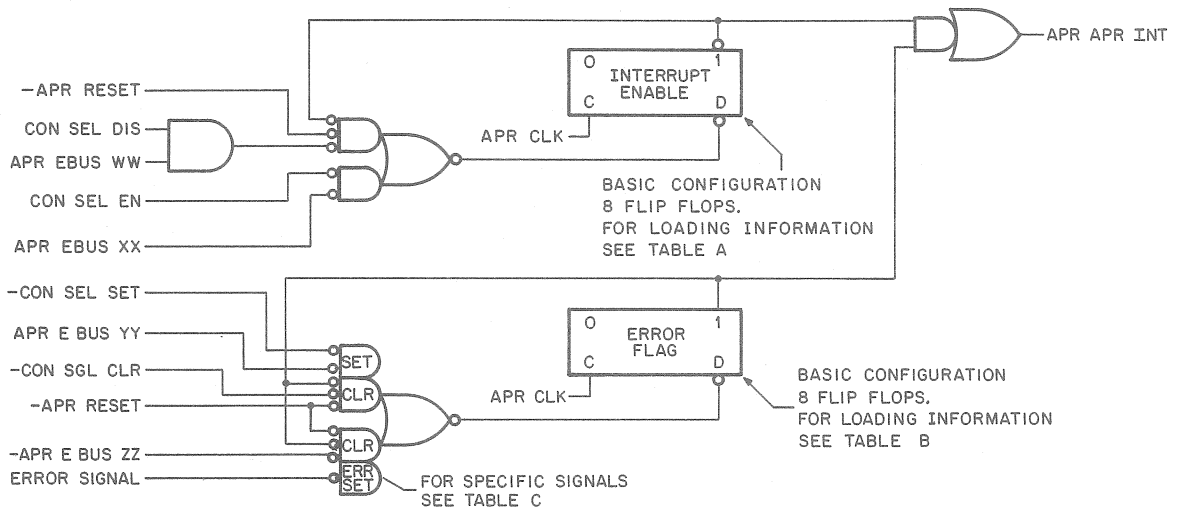


TABLE A

E BUS BIT WW	CON SEL DI'S	CON SEL EN	E BUS BIT XX	INTERRUPT EN SETS	INTERRUPT EN SETS
02		YES	06	S BUS ERR	
03	YES		06		S BUS ERR
02		YES	07	NXM ERR	
03	YES		07		NXM ERR
02		YES	08	I/O PF ERR	
03	YES		08		I/O PF ERR
02		YES	09	MB PAR ERR	
03	YES		09		MB PAR ERR
02		YES	10	C DIR P ERR	
03	YES		10		C DIR P ERR
02		YES	11	S ADR P ERR	
03	YES		11		S ADR P ERR
02		YES	12	PWR FAIL	
03	YES		12		PWR FAIL
02		YES	13	SWEEP DONE	
03	YES		13		SWEEP DONE

TABLE B

E BUS BIT YY	CON SEL SET	CON SEL CLR	E BUS BIT ZZ	ERROR FLAG CLRS	ERROR FLAG SETS
04		YES	06	S BUS ERR	
05	YES		06		S BUS ERR
04		YES	07	NXM ERR	
05	YES		07		NXM ERR
04		YES	08	I/O PF ERR	
05	YES		08		I/O PF ERR
04		YES	09	MB PAR ERR	
05	YES		09		MB PAR ERR
04		YES	10	C DIR P ERR	
05	YES		10		C DIR P ERR
04		YES	11	S ADR P ERR	
05	YES		11		S ADR P ERR
04		YES	12	PWR FAIL	
05	YES		12		PWR FAIL
04		YES	13	SWEEP DONE	
05	YES		13		SWEEP DONE

TABLE C

ERROR FLAG	ERROR SIGNAL
S BUS ERR	MBOX S BUS ERR
NXM ERR	MBOX NXM ERR
I/O PF ERR	APR SET I/O PF ERR
MB PAR ERR	MBOX MB PAR ERR
C DIR P ERR	CSH ADR PAR ERR
S BUS ADR P ERR	MBOX ADR PAR ERR
PWR FAIL	PWR WARN
SWEEP DONE	APR SWEEP BUSY ∧ -APR SWEEP BUSY EN

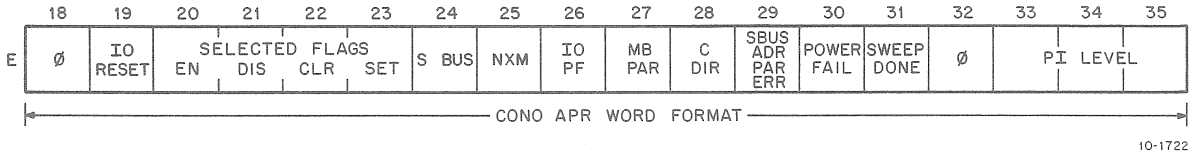


Figure 3-30 APR Register and Interrupt Enables

The basic organization of the APR is illustrated in Figure 3-31. The register is broken down into four sections based on the origin of the error. The first five flags set as a result of an error condition involving some memory activity. Three of the flags: [SBus Error, Nonexistent Memory (NXM) Error, and S ADR Parity Error] originate in the memory adapter (DMA). The remaining two originate in the MBox. The flag IN-OUT PAGE FAIL (IOPF) sets because of an external stimulus, but the actual setting takes place by the microprogram, in response to a page failure that occurred during a priority interrupt. The power failure flag sets when the power controller detects a low voltage condition. The sweep done flag signals the completion of a cache sweep operation. This operation is the result of performing a sweep instruction.

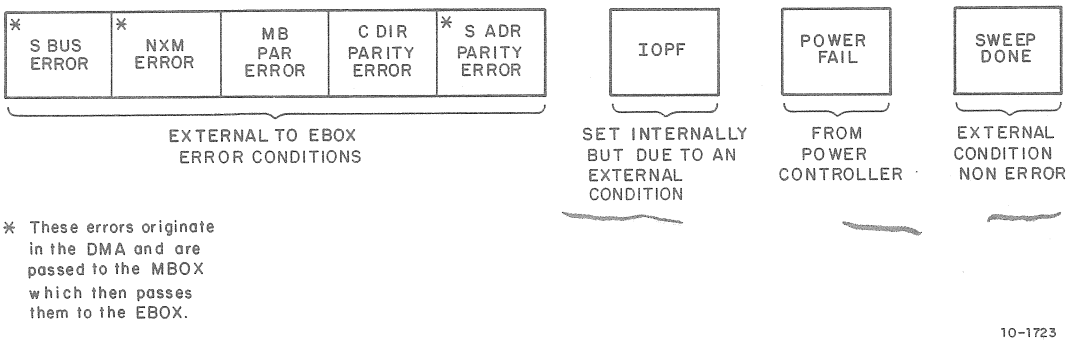


Figure 3-31 APR Register Breakdown

Once again referring to Figure 3-30, to enable interrupts for any or all of the eight conditions, a CONO APR is performed with bit 20 equal to 1 and ones in bits 24 through 31 for the desired flags. Similarly, to disable interrupts for any of the eight flags, which have previously been enabled, place bit 21 equal to 1 and ones in bits 24 through 31 for the flags to be disabled. This means that once the processor has been powered up, and providing a power failure condition has not occurred, that once an interrupt enable has been set, it must be specifically cleared as indicated above.

Any of the eight flags can be selectively set or cleared by placing bit 23 or 22 on, respectively, together with those bits in 24–31 to be changed.

3.3.3.1 SBus Errors – Two error lines are available from the DMA to the MBox. These are SBUS ADR PAR ERR and SBUS ERR. If the DMA starts a memory cycle and also detects bad address parity, it sends SBus Acknowledge (SBUS ACKN) to the MBox, acknowledging receipt of the address and within 125 ns transmits SBUS ADDRESS PAR ERR. The MBox now latches the error address register (ERA), which contains the address in question and additional bits which specify information associated with "data parity error conditions." These two bits specify which of the four memory buffers (MBs) the parity error is associated with. The address used to address memory specifies which word is to be transmitted (for a write) or received (for a read) first. This information is contained in bits 34 and 35 of the address. If, for example, the address in the ERA is 101 [bit 34(0) and bit 35(1)] and the address in the PMA used to address memory is 100, the indication is that the word requested by the EBox, for example, was not the word actually causing the data parity error. Thus, in this example, the EBox requested the contents of location 100, received it, and how, while fetching a word from 101 (of a quadword group), an error occurred associated with that word.

In addition, a 3-bit code identifies the origin of the data in the memory buffer register and indicates the type of reference, i.e., read, write, etc. As the MBox latches the ERA, it transmits MBOX RESPONSE IN and MBOX S ADR PARITY ERROR to the EBox. MBOX S ADR PARITY ERROR occurs concurrently, with an MBox clock and, therefore, on the next MBox clock (that will be also an EBox clock) APR S ADR PARITY ERROR sets. Providing the SBUS ADR PARITY ERROR INTERRUPT enable is set, an interrupt will be requested on the APR channel. In addition, to prevent the MBox error condition from being changed, the APR error flag which sets is sent over the E/M interface to recirculate the MBOX SBUS ADR PARITY ERR COND; also, APR ANY EBOX ERR sets and is passed to the MBox to hold the ERA. As a result of the interrupt, the monitor determines that the APR was the source of the interrupt via a condition I/O instruction (CONSO, CONSZ, CONI, APR), make a determination, and finally clear the error flag, releasing the MBox ERA and associated error logic.

3.3.3.2 Nonexistent Memory – Each time the EBox makes a memory reference, the MBox interprets the request qualifiers and performs all the steps necessary to satisfy the request. A core memory reference must be issued by the MBox in order for NXM to occur. When the MBox issues a memory request to read or write a word to core memory via the memory adapter (DMA), it starts a timeout (32 μs) and waits for SBUS ACKN from the DMA indicating acceptance of the request and address. If 32 μs elapse and SBUS ACKN is not forthcoming, the MBox sets MEM ERR (Figure 3-32).. An additional 32 μs elapses and if SBUS ACKN has not been received by the MBox, MBox NXM error is asserted together with MBOX RESP IN.

Referring to Figure 3-33, MBOX NXM ERROR is loaded into the APR register with APR CLK. If the NXM ERR interrupt enable is set, APR INTERRUPT is asserted to the PI Board. To preserve the ERA and NXM ERROR in the MBox, the APR NXM flag is recirculated back to the MBox. In addition, PAR ANY EBOX ERR sets, holding the ERA information in the ERA register.

3.3.3.3 Other External Errors – Referring to Figure 3-34, all five external error conditions set the appropriate APR ERROR flag and request interrupts (if enabled) on the error channel assigned. Also, all the indicated error flags recirculate to the MBox and all cause APR ANY EBOX ERROR to set, preserving the contents of ERA. Of the five errors, one, MB PAR ERROR, is handled as if it were a page fault. That is, it causes control to be passed to the microcode page fault handler, where it is evaluated. The status word is obtained from the ERA in the MBox. The format for this word is initially as indicated in Figure 3-35.

The page fault microcode places a code in bits 0–5 of 26_8 and places the virtual address for the reference in bits 13–35 where bits 13–17 are 0 for KI paging mode; this word is stored in user process table location 500. The remainder of the operation is identical with that for a page failure and is covered in Section 2.

3.3.3.4 Input/Output Page Failure Error – During a priority interrupt [PI CYCLE (1)], page failures are not expected to occur for interrupt instruction fetches or PI dispatches. This is regarded as a fatal error, and it causes an interrupt on the assigned APR error channel. The page fault handler sets IOPF in the APR register and then dismisses the interrupt. The PC is placed in VMA and an instruction fetch begins while waiting for the PI system to honor the interrupt for the APR.

3.3.3.5 Power Fail – The power controller asserts the signal POWER WARN whenever the power supplies reach a marginal value. This results in the setting of the APR POWER FAIL flag and requests an interrupt on the APR error channel.

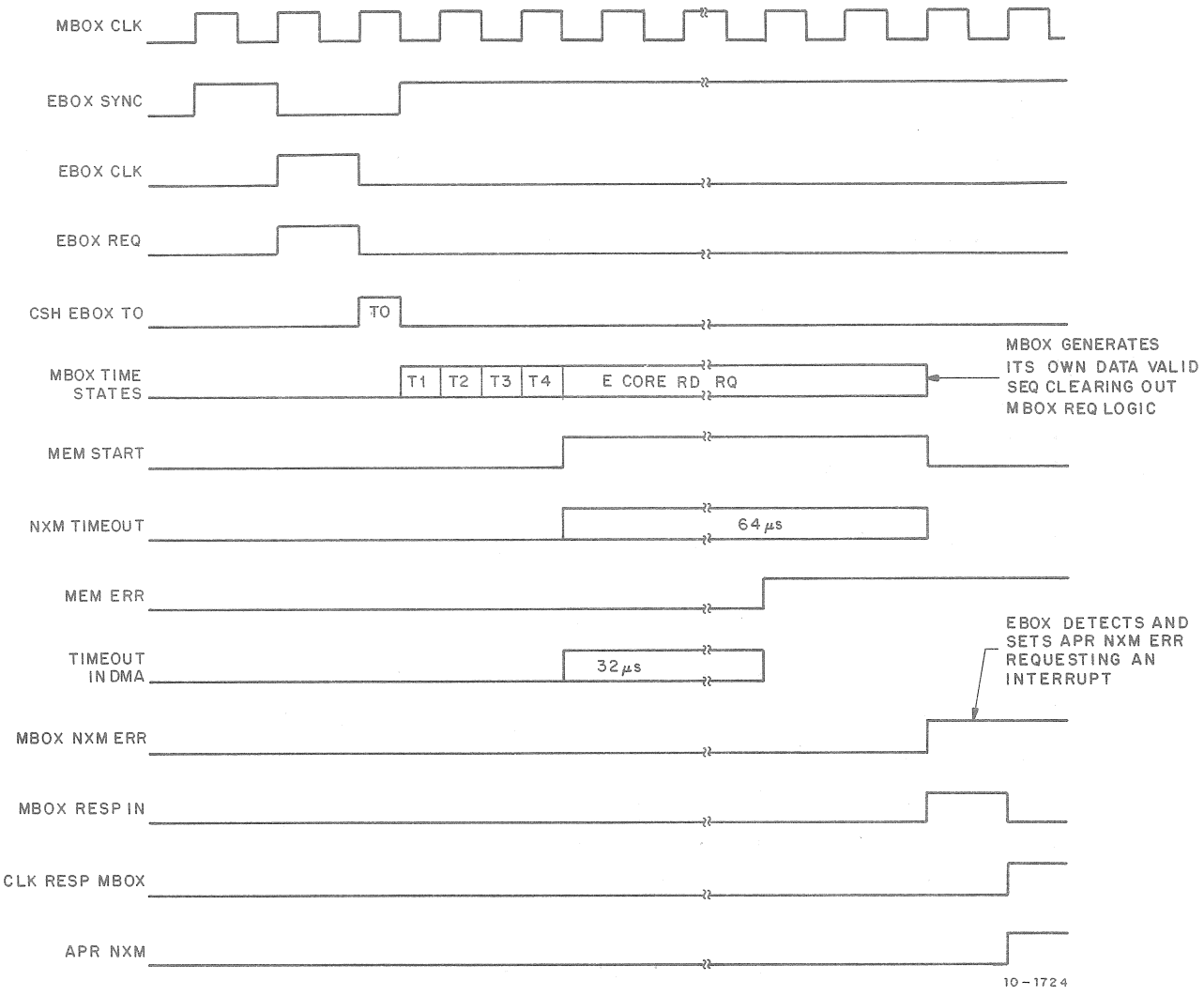


Figure 3-32 NXM Timing Overview

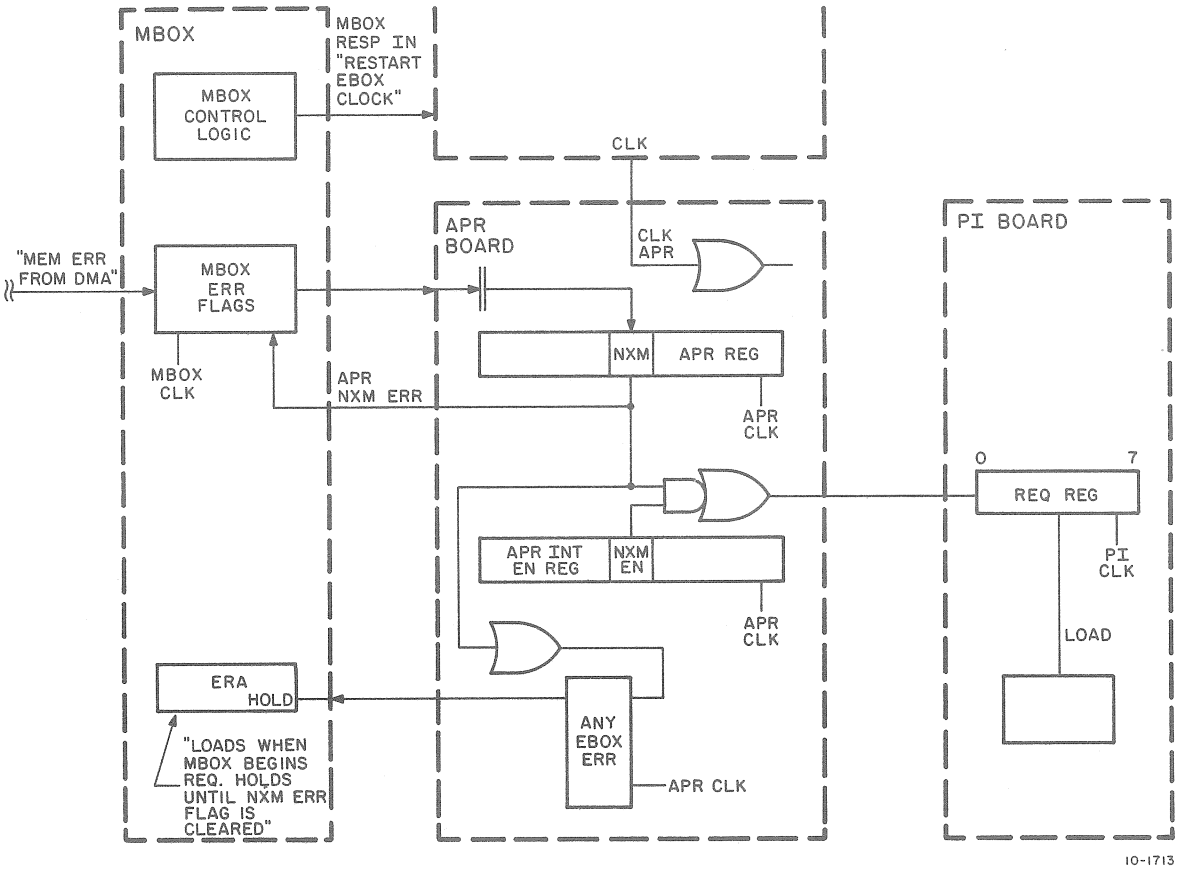


Figure 3-33 NXM Error Overview

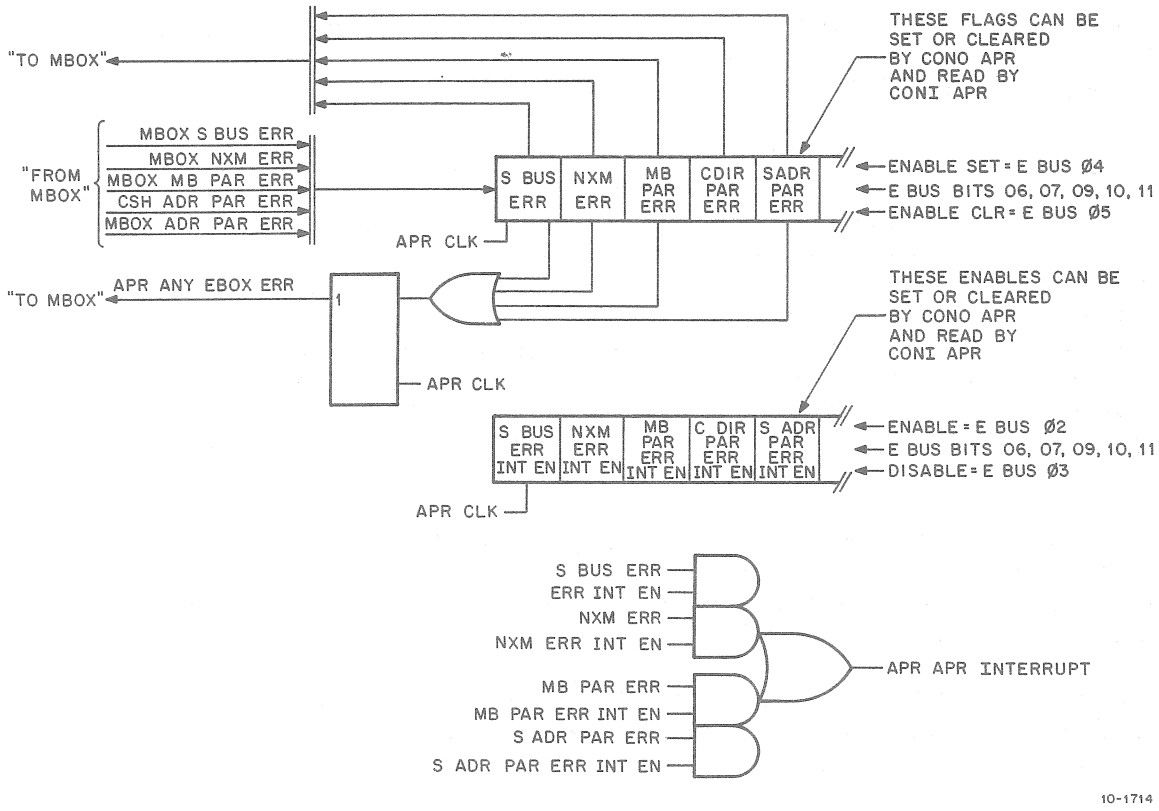


Figure 3-34 External Error Conditions (MBox, SBus)

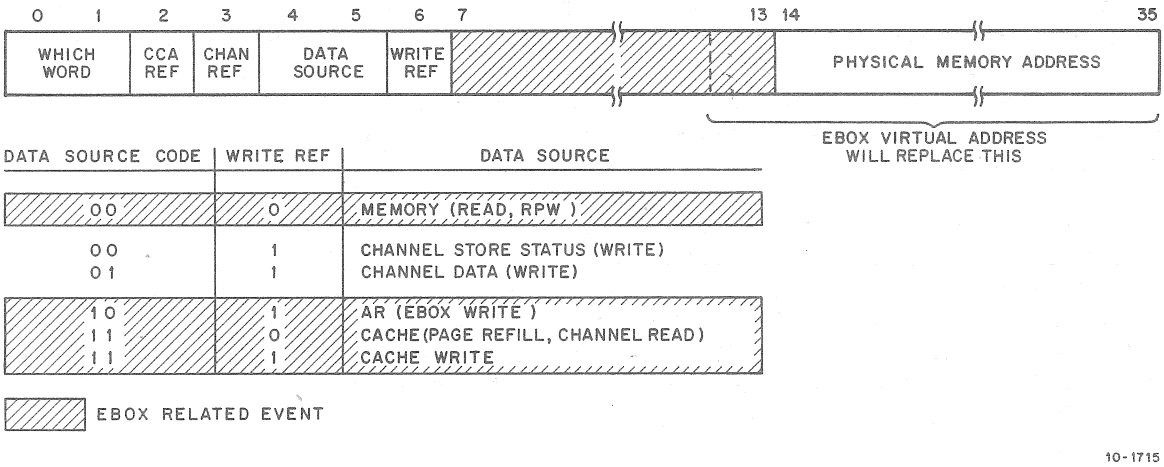


Figure 3-35 ERA Word

3.3.3.6 SWEEP and SWEEP DONE – The MBox contains a section of logic called the Cache Clearer (CCA). This is addressed as if it were a device (014), using I/O instructions. Six operations may be initiated. These are listed in Table 3-7.

Table 3-7 CCA Summary

New Mnemonic	Old Mnemonic	Function
SWPIA	DATAI CCA	Invalidate all cache data; do not update core.
SWPVA	BLKO CCA	Sweep cache, validate core, leave cache valid.
SWPUA	DATAO CCA	Unload all pages updating core; validate the cache.
SWPIO	CONI CCA	Invalidate one page of the cache; do not validate core.
SWPVO	CONSZ CCA	Sweep cache, validate one page of core, leave cache valid.
SWPUO	CONSO CCA	Unload one page, update core, invalidate the cache.

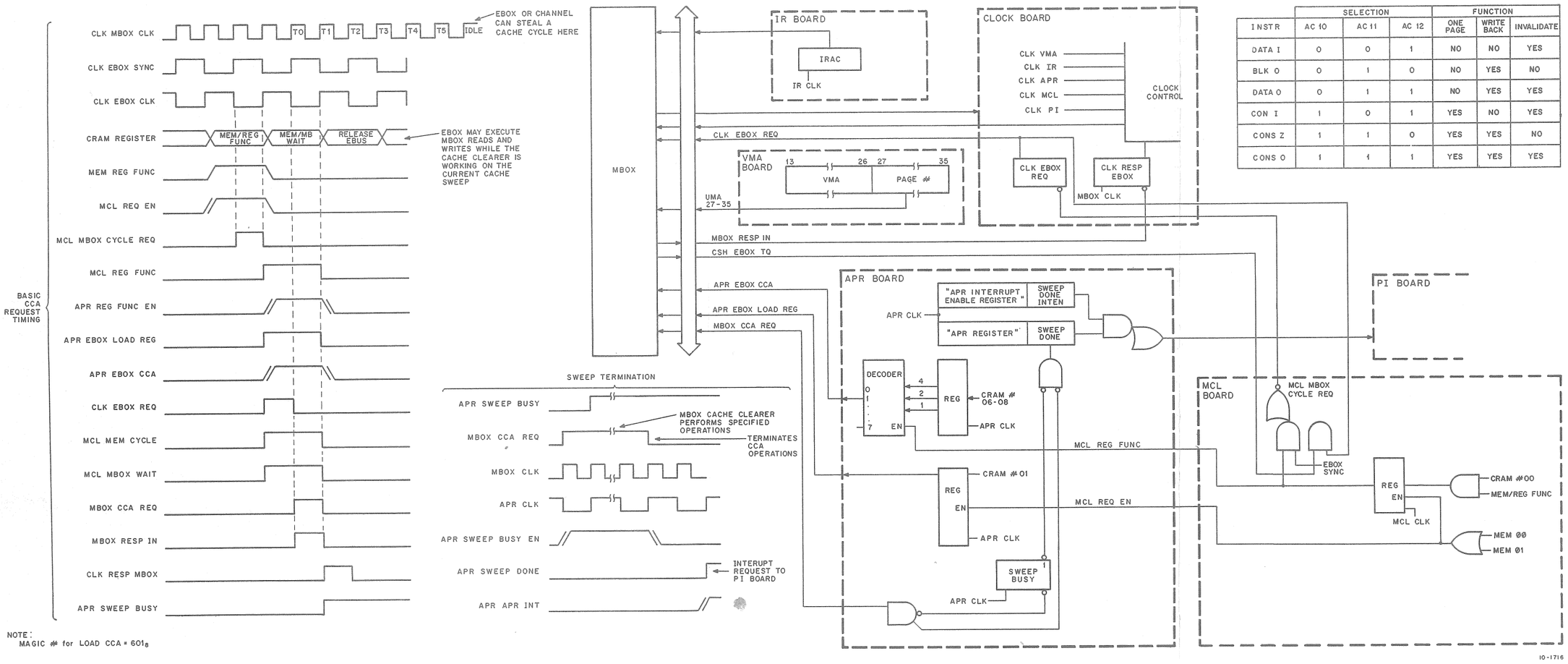
To request CCA cycles from the MBox as a function of one of the six instructions in Table 3-7, the EBox places the virtual page number into VMA 27–35, verifies that the performance of the Sweep instruction (which is privileged) is legal in the current mode of the processor and then either begins the operation or, if illegal, performs an MUUO.

Figure 3-36 illustrates the various logic associated with the sweep operation. Three basic operations can be specified in various combinations by the six types of Sweep instructions. These are illustrated in Figure 3-36 in the table at the upper left.

In the cache, associated with each word of a four word block (quadword), are two bits labeled valid and written. If the valid bit is off for any of the four words, these words are considered to contain incorrect data and, if referenced (for example by the EBox), the words must be fetched from main memory. Similarly, if the written bit is on for any of the valid words, these words contain different data than the copy in main memory and the cache copy is correct. At some point, the written words must be flushed from the cache into core memory. On power up, the cache must be invalidated, clearing all the entries. For this case, the DATAI instruction is performed to device CCA. Because AC bit 10 is 0, the MBox, upon receiving the EBox request and appropriate qualifiers (APR EBOX CCA and APR EBOX LOAD register), will invalidate the entire cache. Similarly, because AC bit 11 is 0, the MBox disregards the written words and no writebacks are performed to core memory. Finally, AC bit 12 is 1, which specifies invalidation.

Referring to Figure 3-36, IRAC contains the AC field 9–12 of the instruction. The microcode executor sets up the request utilizing the MEM field function MEM/REG FUNC together with the magic number field coded as LOAD CCA (601_8). To follow the memory request, it is best to refer to Figure 2-98 which can be found in Subsection 2.7.2.5. Note that on Figure 3-36 MEM/REG FUNC (07) has bit 01 equal to 1 and this generates MCL REQ EN. This signal is used to enable the various registers involved in the EBox request to load with the appropriate information prior to latching the VMA. The following conditions set up for the CCA request.

Controlling Signal(s)	Signal Generated
MEM/REG FUNC	MCL REQ EN
MCL REQ EN∧ MEM/REG FUNC ∧ CRAM#00	MCL REG FUNC
MCL REQ EN∧ MCL REG FUNC ∧CRAM#01	APR EBOX LOAD REG
APR REG FUNC EN∧ CRAM#06–08 = 1	APR EBOX CCA
MCL REG FUNC∧CLK EBOX SYNC	MCL MBOX CYCLE REQ



INSTR	SELECTION AC 10	SELECTION AC 11	SELECTION AC 12	FUNCTION ONE PAGE	FUNCTION WRITE BACK	FUNCTION INVALIDATE
DATA I	0	0	1	NO	NO	YES
BLK O	0	1	0	NO	YES	NO
DATA O	0	1	1	NO	YES	YES
CON I	1	0	1	YES	NO	YES
CONS Z	1	1	0	YES	YES	NO
CONS O	1	1	1	YES	YES	YES

Figure 3-36 Sweep Logic

The basic timing for the CCA request as well as CCA termination is illustrated in Figure 3-36. The VMA must contain the virtual page number in VMA 27–35 for CONI, CONSZ, or CONSO CCA operations. In the current example (DATAI CCA), the MBox cache clearer does not use this information because the entire cache is to be invalidated. However, the cache clearer has an associated register that is loaded by the MBox with VMA 27–35. IRAC bits 10–12 are similarly loaded into the MBox control logic that directs the type of operation carried out. Each time a CCA cycle is completed in the MBox, an idle period occurs where the channels or EBox can obtain an MBox cycle. The EBox can continue to execute instructions but must guard against defeating the purpose of the Sweep operation, i.e., write new data into already swept words in the cache. Summarizing, three of the six instructions operate on one page of the cache (512 words). For these three instructions a different set of sweep functions is available; these are: invalidate, writeback all written words in the specified page, or perform both. Similarly, three instructions operate on the entire cache (2048_{10} words) but the operations are the same as with the other three. In all cases, the EBox performs an EBox Request providing the appropriate qualifiers and the VMA contains (in bits 27–35) the page number. The MBox loads its CCA register and then asserts MBox CCA Request together with MBOX RESPONSE IN. Now the EBox is free to perform operations while waiting for SWEEP DONE to generate an APR interrupt. If a second sweep instruction is started by the EBox before the first is completed, the MBox begins the second sweep just as it would another instruction; however, it reloads the CCA register with the new information supplied by the second sweep instruction and does not complete the first.

3.3.4 Processor Identification

The processor identification consists of four parts:

Microcode options
Microcode version number
Hardware options
Processor serial number

This information is obtained by performing what was traditionally a BLKI APR, now called APRID. The format is illustrated in Figure 3-37.

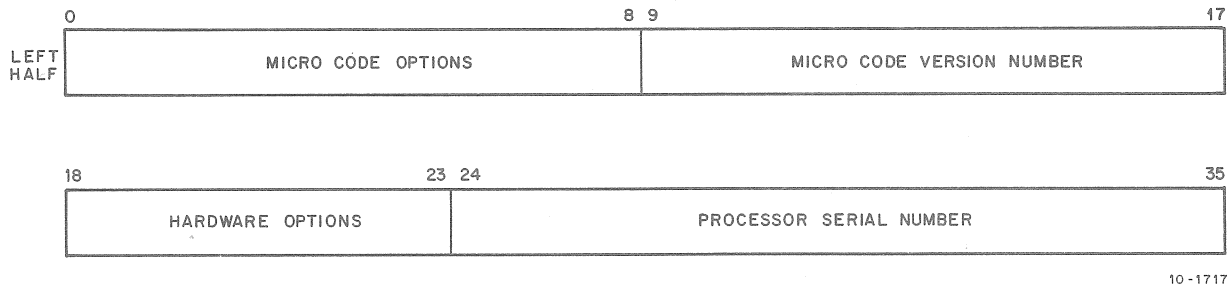


Figure 3-37 APRID Format

This is not strictly a visible hardware function, but rather a combination of microcode and hardware. The microcode for a given version is coded in such a fashion that the version number is obtained utilizing the magic number field and the function AR00–08← number. The microcode obtains the processor serial number that is hardwired to the 0 input of the ADXB mixer and places it in AR. Next, the microcode version number is obtained and adjusted as follows. The serial number in AR is copied to BR and the version number is loaded into AR00–08; next, the ARX. At this time the BR, AR, and ARX are as indicated in Figure 3-38.

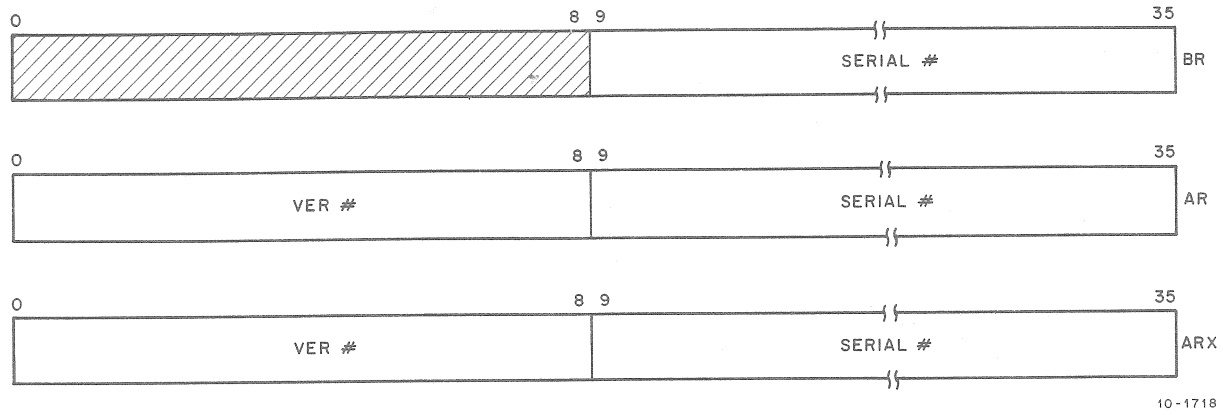


Figure 3-38 Alignment Step 1

The shift counter is loaded with 9_{10} and now the combined AR and ARX are shifted left 9 places with the result placed in AR as indicated in Figure 3-39.

The version number is placed in AR 9–17, the serial number in AR 24–35, and the resulting word is stored in location E.

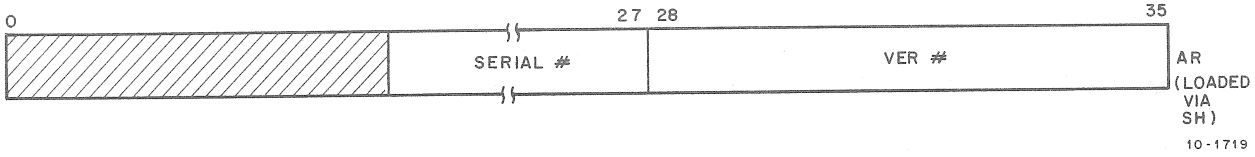


Figure 3-39 Alignment Step 2

3.3.5 Cache Refill RAM Facility

The cache refill RAM in the MBox must be loaded with a set of bit patterns called the refill algorithm. This RAM is used by the MBox with a use table and other associated logic to manage the cache refill operation. Generally speaking, when the cache fills up with words, it becomes necessary to displace old words for new ones. It is desirable to displace the words used most infrequently. To do this, an algorithm was developed that specifies which word is to be displaced each time a refill cycle must write into the cache. Figure 3-40 illustrates the basic structure of the MBox Refill RAM and also indicates the format of the effective address provided by the BLKO APR instruction (new mnemonic WRFIL). The microcode executor is entered with the effective address (E) in AR. Because the instruction is privileged, legality is checked first. If the instruction is legal for the current mode of the processor (Kernel or User with IOT set), the instruction is performed; otherwise, an MUUO is effectively performed with the illegal instruction stored in the user process table location 424 in the place where the MUUO is normally stored.

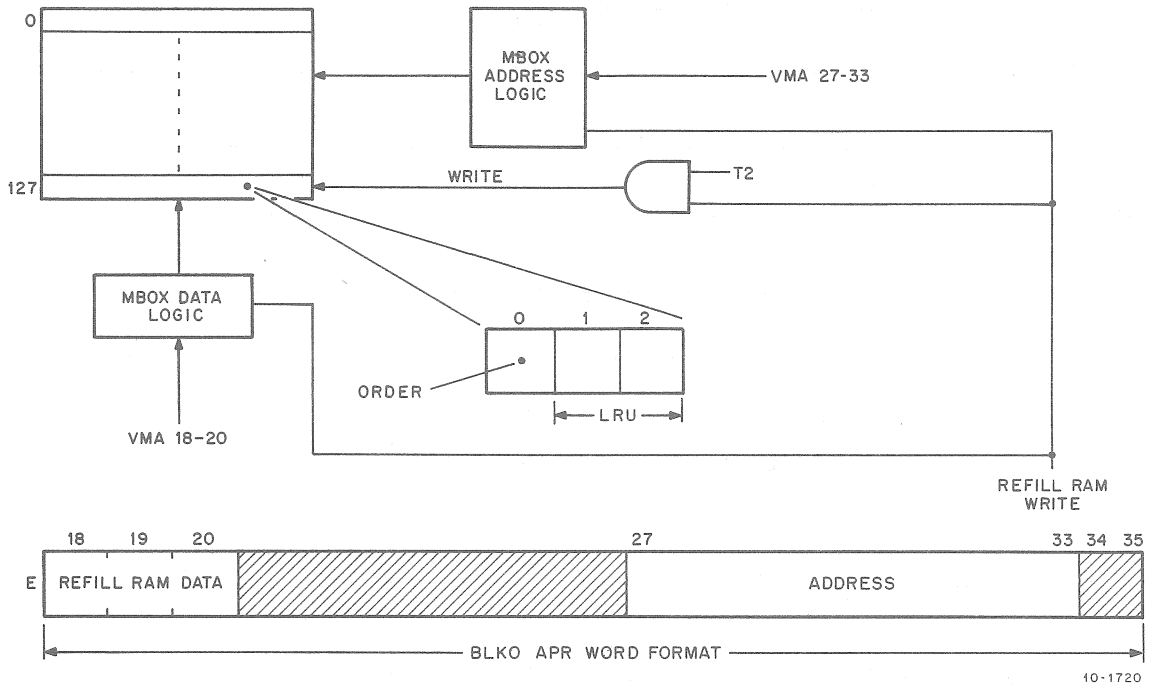


Figure 3-40 Refill RAM Overview

When the instruction is legal, the microcode performs a MEM/REG FUNC with the magic number field coded as WR REFILL RAM. The APR logic decodes the REG FUN during the EBox Request:

```
APR EBOX READ REG
APR EN REFILL RAM WR
```

The MBox writes the three high-order bits (18–20 of VMA) into the refill RAM at the location addressed by bits 27–33 of VMA. Writing the entire algorithm requires a loop using the basic instruction BLKO APR as a focal point. The following is an example:

```
        SETZB Z,AC              ;CLEAR REGISTERS
RAM1:
        MOVE AC,TABLE(Z)        ;PICK UP A WORD
        BLKO APR,0(AC)          ;WRITE THE FILL RAM
        CAIN Z,127              ;DONE ALL 12810 WORDS?
        JSR DONE                ;YES
        AOS Z                   ;NO, UPDATE Z FOR NEXT
        JRST RAM1               ;PICK UP NEXT WORD FROM
        THE TABLE
```

In the sample program, table through table+127 contain the appropriate entries to be written into the MBox Refill RAM. These words are in the format indicated on Figure 3-40. The refill algorithm may be adjusted by changing the sequence of the bit patterns. By doing this, portions of the cache may be bypassed as appropriate. Normally, all four cache quarters would be used equally. Table 3-8 is reproduced as extracted from the MBox theory section simply as an example.

Table 3-8 Sample Algorithm

Refill RAM Locations	Refill RAM Contents							
0–7	0	1	2	3	4	5	6	7
8–15	3	1	2	3	2	1	2	3
16–23	7	1	2	7	1	1	2	7
24–31	6	5	6	7	5	5	6	7
32–39	0	3	2	3	0	2	2	3
40–47	0	1	2	3	4	5	6	7
48–55	0	7	7	7	0	0	0	7
56–63	4	6	6	6	4	4	6	4
64–71	3	1	3	3	1	1	1	3
72–79	0	7	7	7	0	0	0	7
80–87	0	1	2	3	4	5	6	7
88–95	4	5	5	7	4	5	4	7
96–103	0	1	2	2	0	1	2	1
104–111	0	5	6	6	0	5	6	0
112–119	4	5	6	5	4	5	6	4
120–127	0	1	2	3	4	5	6	7

3.3.6 MBox Error Address Register

The MBox contains a number of registers that can be loaded and read by the EBox. These registers are address registers for storing the address in the event of an error and for modifying the physical memory address in response to certain request qualifiers. The registers are:

a. User Base Register – UBR
b. Executive Base Register – EBR
c. Cache Clearer Address – CCA
d. Error Address – ERA

The ERA register can only be read by the EBox. In addition, the EBox can also read the contents of the page table to transform (map) the virtual address to the physical address and load the cache refill RAM with the cache refill algorithm.

A status word is formed and stored by the MBox in the event that an error is discovered. The error address is basically a status word that is formed and stored by the MBox when an error is sensed. In the case of a parity, time-out, or an NXM error, the corresponding error flags are set and the error address and associated status bits are loaded into the ERA register. The format of this word was shown in Figure 3-35. This register is read by the EBox when an RDERA (BLKI, PI) instruction is executed.

3.4 CONTROL RAM ADDRESSING

Figure 3-41 contains an overview of the CR addressing logic, while Figure 3-47 contains a more detailed version. The CR addressing logic consists of the following general parts:

Pushdown Stack, 4 words × 11 bits
Current Location register (CRA LOC)
CRAM dispatch field for holding the dispatch bits
Miscellaneous CR address gates
Diagnostic register
Dispatch decoding register 0–3 EN, 0–7 EN, 30–37 EN
CRAM loading logic
CRAM address output gates.

The type of function being performed on the CRA board determines the portions of the above-mentioned logic that are used. These functions are broadly classified as:

1. Loading into the CRAM dispatch

   Diagnostic register
   Control RAM dispatch field
   Write logic

2. Decoding the Jump, Dispatch, and Cond (Skip) fields of a microinstruction

   Mixers
   Optionally the Stack
   Optionally the A READ Logic
   Dispatch decoding register

3. Forcing a special CR address during a page fault

   CR address output gates.

In addition to these three classes, diagnostic logic is present on the CRA board for reading various registers, mixers, and signals onto the EBus. This logic is described in a separate section on EBox diagnostic logic.

3.4.1 Pushdown Stack

The pushdown stack, consists of eleven clocked shift registers configured as an 11-bit SILO. Two control signals, CTL SPEC/CALL and CTL DISP/RET, control the stack. Figure 3-42 illustrates the basic operation for a sequence of two subroutine calls followed by two subroutine returns. The example presented on the figure is not a practical example of subroutine calling and return, but an example of how the stack behaves in response to the call and return control signals. In practice, each subroutine consists of a number of microinstructions. For convenience, these additional instructions have been omitted. In the example the first microinstruction (J = A) asserts the first call. Note that during the first microinstruction, the CR address is "A", which is the address of the next microinstruction. When CRA CLK occurs, three significant events occur.

1. The CR address "A" is clocked into the current address buffer (CRC LOC).

2. The second microinstruction at location "A" is clocked into the CRAM register.

3. The decoding of this microinstruction begins and, in particular, enables the stack to push CRA LOC on the next CRA CLOCK.

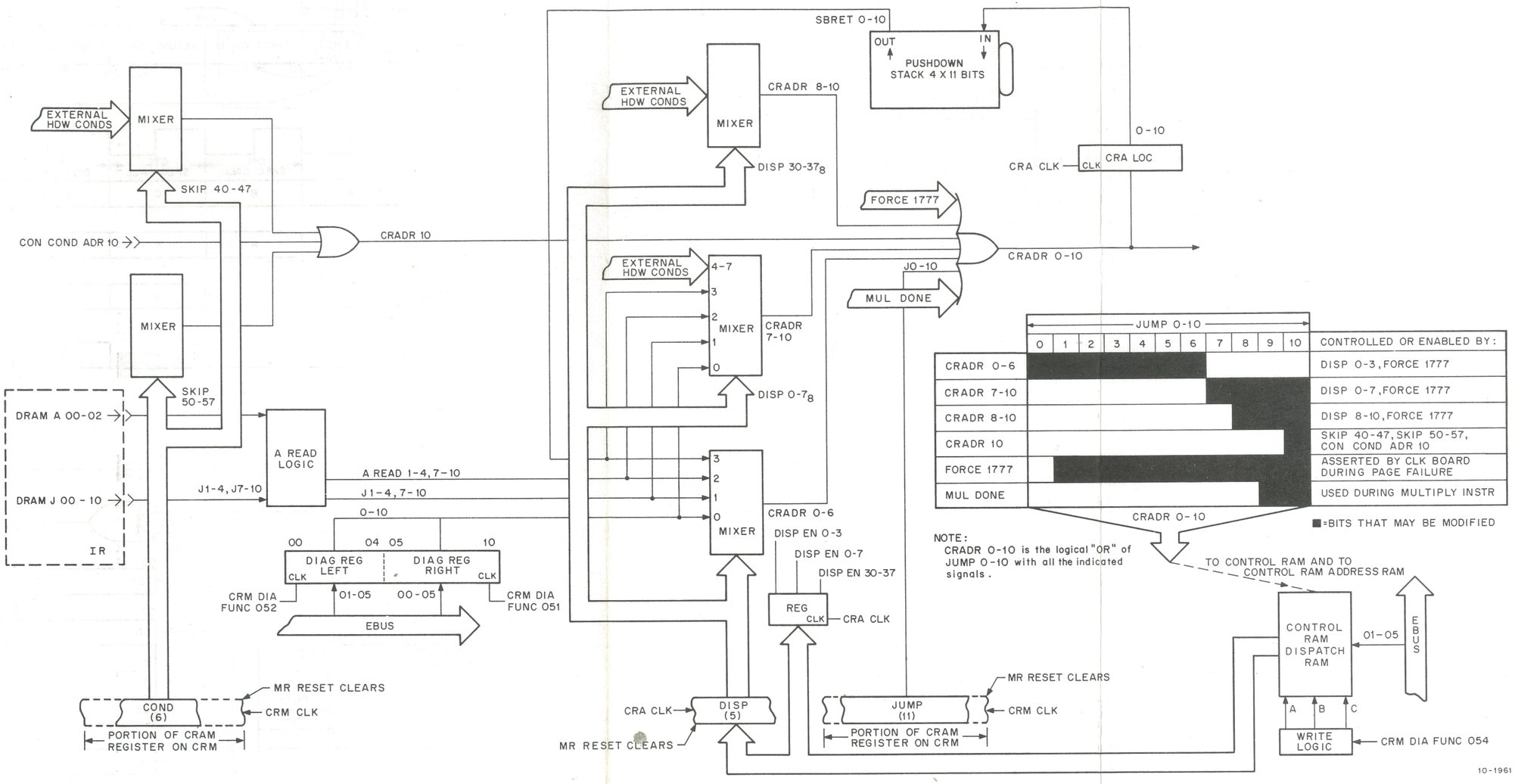


Figure 3-41 CR Addressing Overview

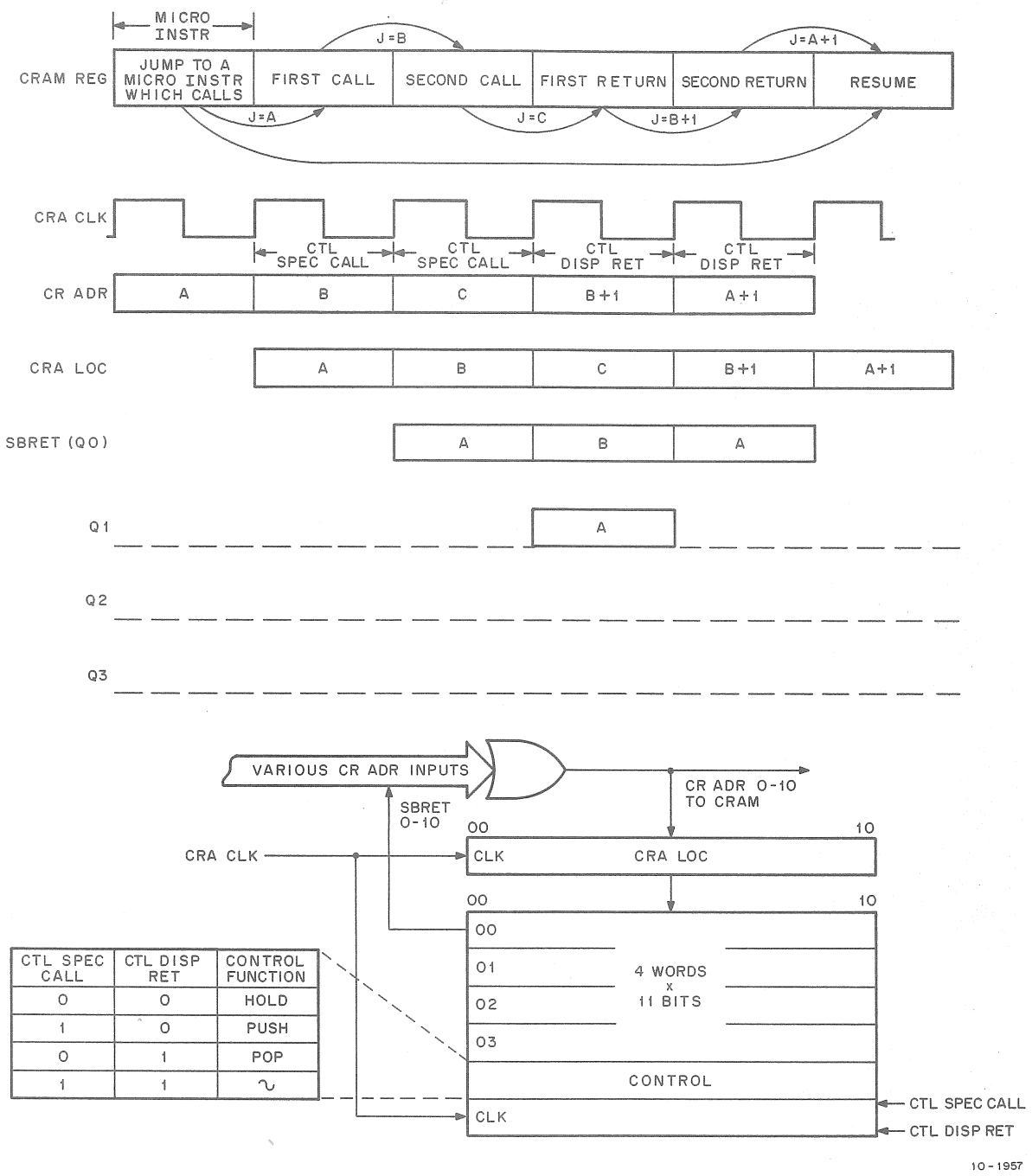


Figure 3-42 Stack Operation Example

Now the CR address becomes "B" as specified by the second microinstruction. Normally, this is the address of the first microinstruction in the subroutine. In the example, it contains a second call (J = B). The next CRA CLOCK again enables the three events indicated above, with the difference being that the CR address is now "B." CRA LOC contains "A." At the next CRA CLOCK, a second push occurs; CRA LOC "B" is pushed onto the stack (Q0) while the previous contents of Q0, which is "A," are pushed one level deeper into Q1 as indicated on the figure. Also, on this clock, the address "C" is clocked into CRA LOC. This time the microinstruction specifies the return function and the Jump address is coded so as to modify the address that will be popped off the stack on the next CRA CLOCK. For example, if the return is to be to the microinstruction following the one that made the call and the top address on the stack is "B" then the least significant bit of the "modifier," which is simply the Jump field of a returning microinstruction, is 1. Thus, the CR address is the logical OR of the address popped off the stack, "B," with the modifier 1, producing the return address B+1. Continuing the example, CRA CLK pops "B" from the stack, clocks the previous CR address (modifier 1) into CRA LOC, and returns to the microinstruction at B+1, which is a second return. Once again, the return is decoded and will enable the address "A" now at the top of the stack to be popped off and logically ORed with the modifier (once again +1) producing a CR address of A+1. This completes the example.

NOTE

In this example, A and B are assumed to be even numbers.

3.4.2 Current Location Register (CRA LOC)

This register consists of 11 clocked D-type flip-flops. Its two main functions are:

1. To provide the current address for the pushdown stack
2. To provide the current address for diagnostic purposes.

3.4.3 Control RAM Dispatch Field

The majority of the control storage for the microprogram is on the CRM board. However, the dispatch field, 1280 words of 5 bits, is contained on the CRA board. The Diagnostic register on the CRA board is used to address the entire CRAM, and this includes the portion on the CRAM board as well. Diagnostic functions are used to enable loading data placed on the EBus into the appropriate portion of the CRAM. Refer to Figure 3-41. The Diagnostic register is selected as input to the CRADR 0–6 and 7–10 mixers following power-up. This is true because the entire CRAM register is reset to zero during MR RESET, and this provides a dispatch field of zero. Using diagnostic functions 052 and 051, the Diagnostic register may be loaded from the EBus. This address now selects a word in the CRAM for loading or reading.

3.4.4 Miscellaneous CR Address Gates

Refer to Figure 3-43. Functionally, there are four sections of gating:

CR Address 00–06
CR Address 07–10
CR Address 08–10
CR Address 10

This grouping corresponds to the way in which portions of the CR address lines may be controlled. The CRAM, of course, sees only an address 0–10.

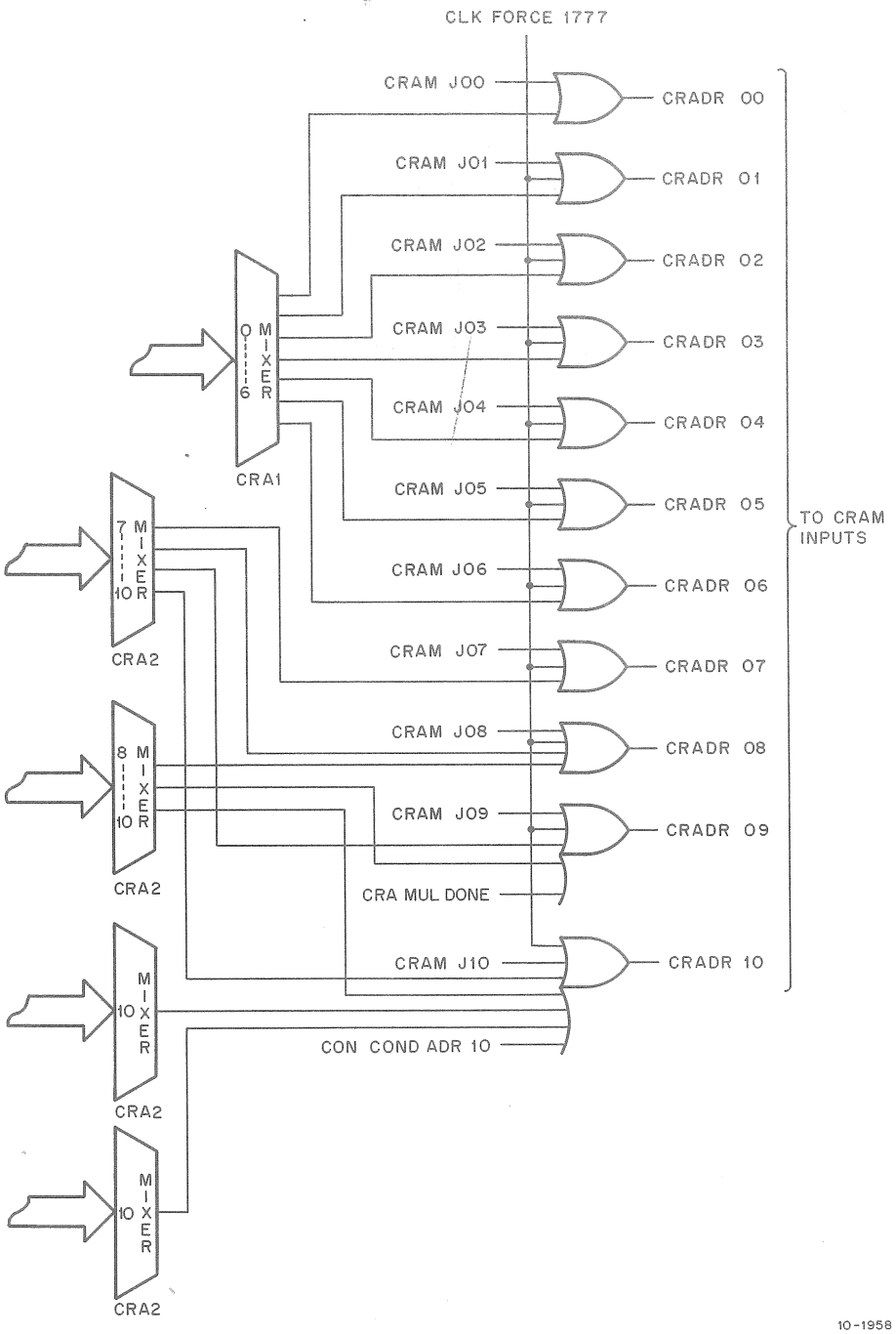
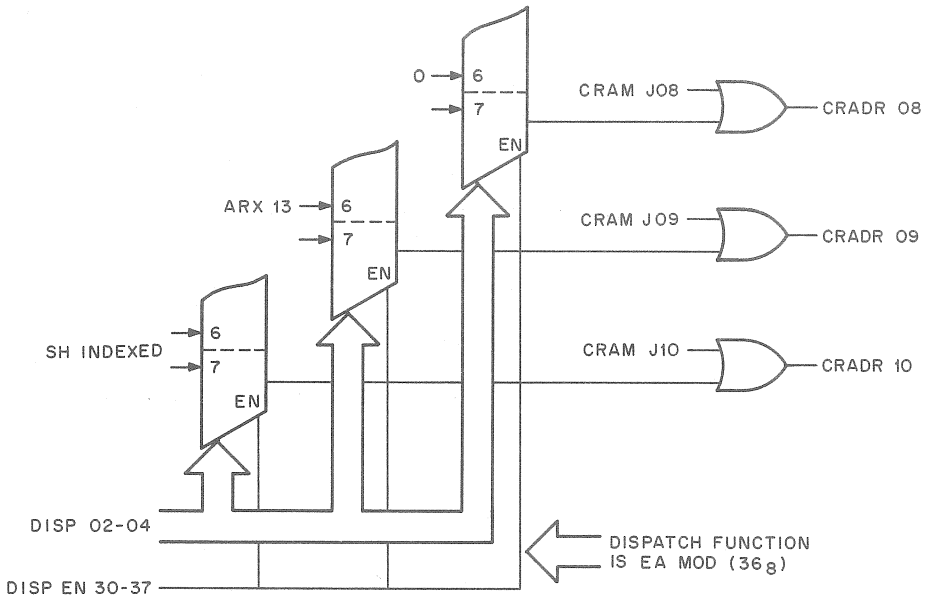


Figure 3-43 CRADR Gates

The fact that the CR address gates are OR gates should be kept in mind when trying to determine an CR output address from a particular input condition or set of conditions. To enable a particular CR address line only requires one of its input lines to be true. For example, consider the example presented in Figure 3-44, which shows the mixers that are used to select conditions to modify CR address bits 08–10. In the example, the dispatch function is effective address modification (EA MOD), which is encoded in the dispatch field as 36_8. Note that in the example the J field (CRAM J 08–10) is 4 in bits 08–10. The four possible combinations of ARX 13 and SH indexed allow any of the following:

1. No modification to CR ADR 09 and 10
2. Modification to only CRADR 10
3. Modification to only CRADR 09
4. Modification to both CRADR 09 and 10.

Because CRAM J 08 is a 1, the respective output gate, CRADR 08, will be a 1 even though the open pin on that mixer (input 6) is effectively a 0.



CONTROL		INPUTS			OUTPUT
DISPEN 30–37	DISP 02–04	ARX 13	SH INDEXED	CRAM J 08–10	CRADR 08–10
YES	6	0	0	4	4
YES	6	0	1	4	5
YES	6	1	0	4	6
YES	6	1	1	4	7

10-1959

Figure 3-44 Example CRADR 08–10

3.4.5 Special CR Address Modification Considerations

Three special CR address modification considerations are:

1. CLK FORCE 1777
2. CRA MUL DONE
3. CON COND ADR 10.

3.4.5.1 CLK FORCE 1777 – This signal originates on the clock board and is used to force the output gates CR address 01–10 to the address 1777_8. This event occurs during a page fault. The page failure microcode handler begins at CRAM location 1777. Thus, the EBox, as controlled by the clock, enters a prearranged page fail sequence. Loading the first microinstruction from the page fault handler, CLK FORCE 1777 forces the CRAM address lines, as indicated, and then issues a single CRM CLK, which loads the microinstruction into the CRAM register. At this point, EBox's normal operation continues. Note that CLK FORCE 1777 does not affect CR ADR 00, and thus may force the microcode to either 1777 or 3777. The first step of the page fault handler is duplicated in these two locations.

Note, also, that at the same time as the CLK board is forcing CLK FORCE 1777, the CTL board is forcing CTL SPEC CALL in order to place the return address on the pushdown stack.

3.4.5.2 CON COND ADR 10 – This external signal is formed on the CON board and routed to CRA 2 as CON COND ADR 10. Refer to Figure 3-45, which shows the boards involved in decoding the Cond and Dispatch fields. Note that each board contains tables indicating those functions that are decoded on that board. The signal CON COND ADR 10 is formed when Skip 60–67 or Skip 70–77 are decoded. The various hardware conditions involved are indicated on the tables.

3.4.5.3 MUL DONE – During the Dispatch function, MUL, the state of the sign of FE, as well as MQ34 and MQ35, are used to modify the CRAM address in the multiply loop. When the sign of FE becomes false an exit is made from the multiply loop. This is done via CR ADR 08. Simultaneously, MUL DONE (Figure 3-46) is generated to force address bits 09 and 10. This is done merely to save microcode words. Without this logic, MUL DISP would be an 8-way branch; with this logic, it is a 5-way branch.

3.4.6 AREAD Logic

Refer to Figure 3-47. The AREAD logic is shown on the lower right-hand side. It consists of a mixer and various gating elements. Basically, this logic is controlled by bits of the DRAM A field. Specifically, when the DRAM A field bits 00 and 01 are 0s; then the AREAD logic AREAD 01–04 and AREAD 07–10 become equivalent (bit for bit) to DRAM J01–04 and DRAM J07–10. When DRAM A00 or 01 is a 1, then AREAD 01–04 and 07–10 generate 40_8 +A, dispatching to location 42 through 47 in the microcode.

The outputs of the AREAD logic (to be able to modify the CR address lines) must be selected in the appropriate mixers. Once again referring to Figure 3-47, the mixers involved are those controlling CRADR bits 00–06 and 07–10. These mixers will select the AREAD function when the dispatch field is coded as "2."

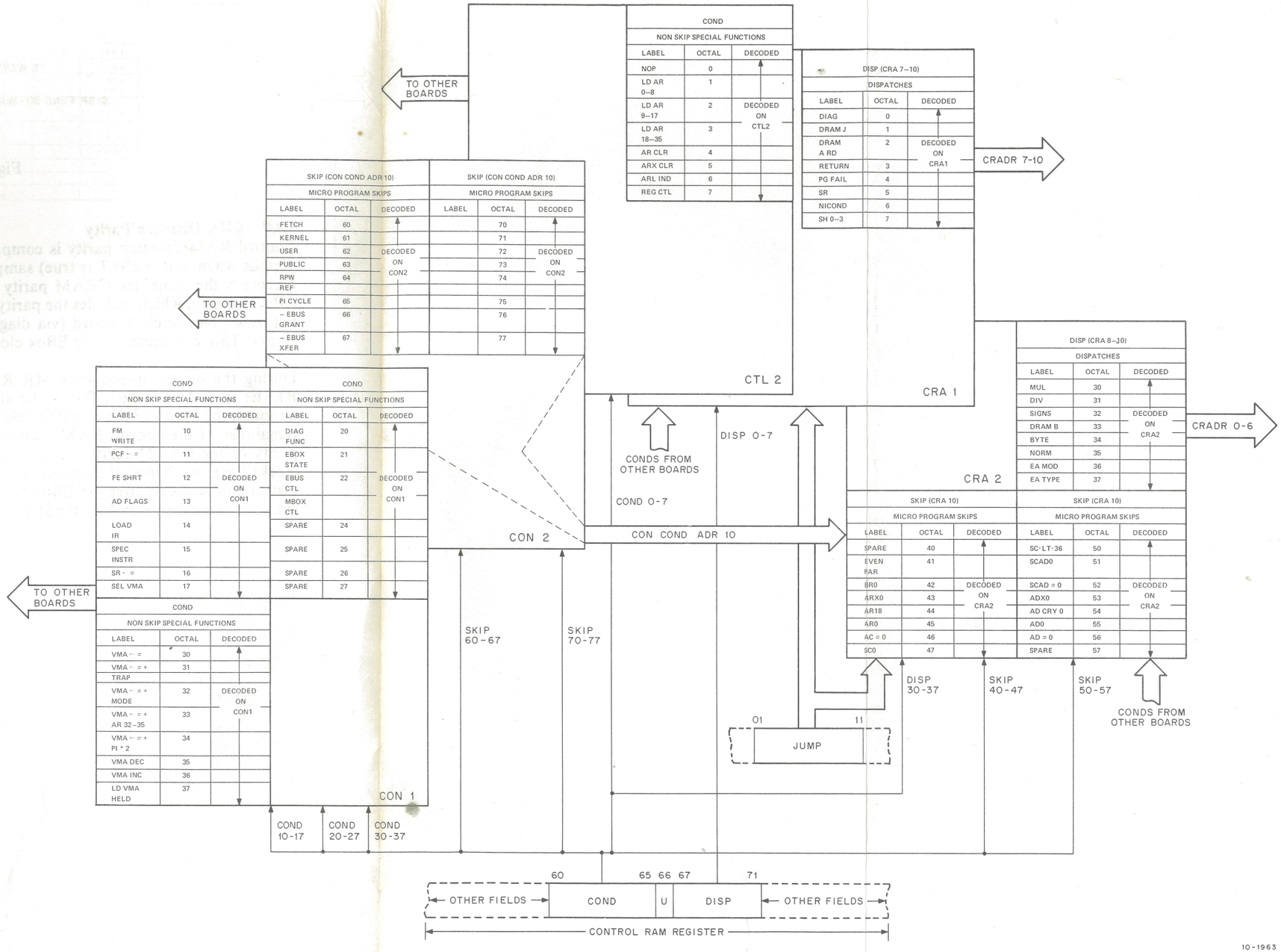


Figure 3-45 COND and Dispatch Layout and Control

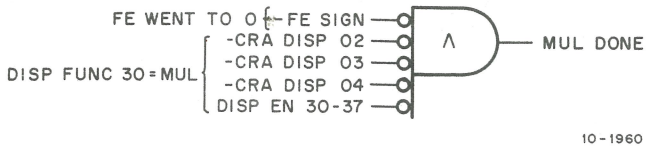
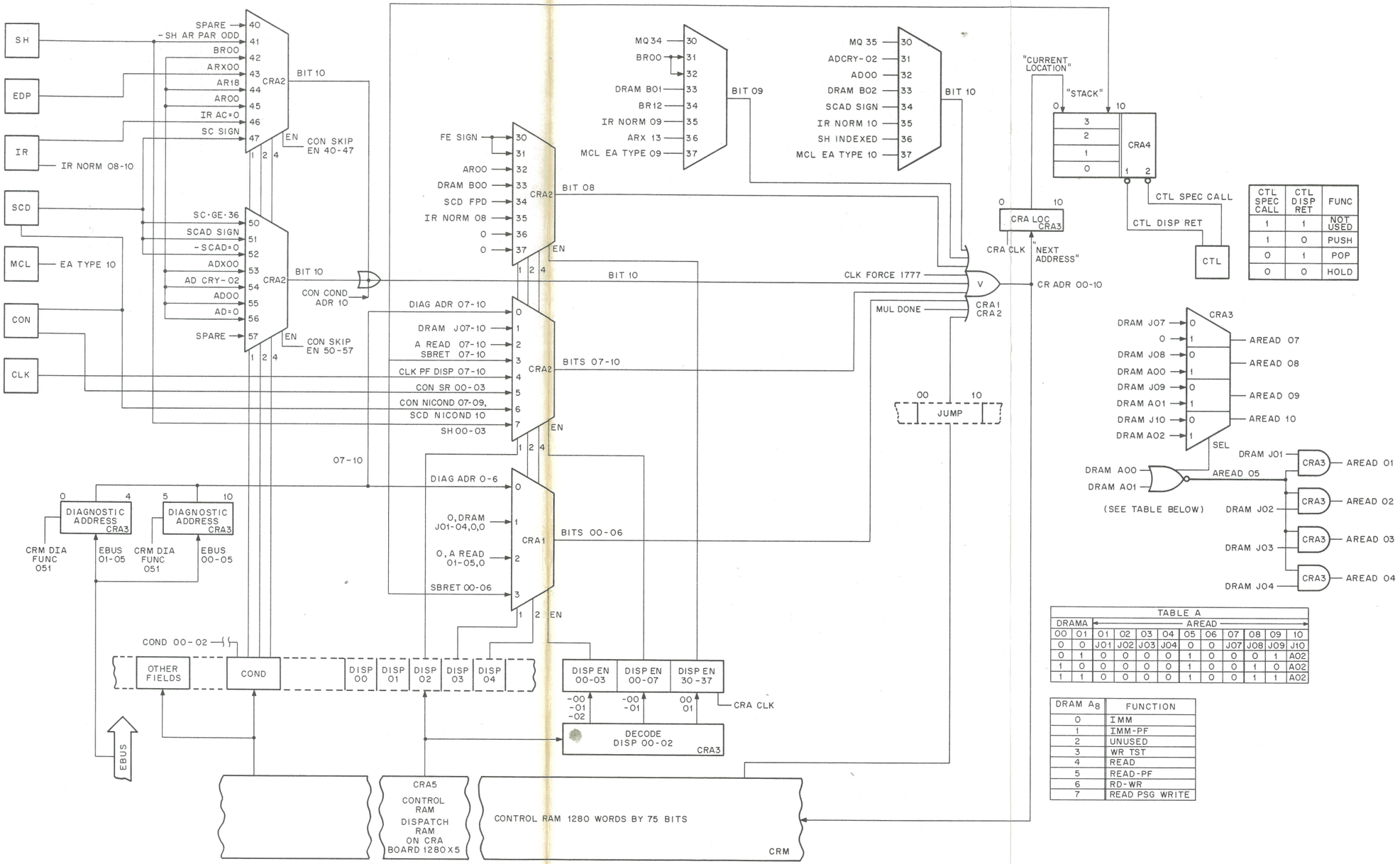


Figure 3-46 MUL Done

3.4.7 CRA Dispatch Parity

Control RAM dispatch parity is computed using a 10160 parity circuit. This circuit (except during periods when MR RESET is true) samples CRA DISP bits 00–04 and computes CRA DISP parity. Normally the combined CRAM parity is odd, when correct. The clock board monitors the state of CRAM parity, which includes the parity for the dispatch field. If the CLK CRAM PARITY CHECK flag is set on the clock board (via diagnostic function 044), then any CRAM parity error stops all clocks. This will occur on the EBox clock following the CRAM parity error.

During the power up sequence MR RESET sets and remains set. This generates the signal DISP RESET PARITY, which forces the state of the dispatch parity network to indicate odd parity, although the parity of the dispatch field (which now contains all zeros) is even. This, together with the remainder of the control RAM register which is clear, yields odd parity. The effect is to make the parity of the CRAM register appear to be odd following MR RESET. This logic assures that the clocks have no chance of stopping in the event that CLK CRAM PAR check is true when a CONO instruction is issued after the EBox has been powered up and this instruction causes MR RESET or similarly if a diagnostic MR RESET is issued.



CTL SPEC CALL	CTL DISP RET	FUNC
1	1	NOT USED
1	0	PUSH
0	1	POP
0	0	HOLD

TABLE A

DRAMA		AREAD									
00	01	01	02	03	04	05	06	07	08	09	10
0	0	J01	J02	J03	J04	0	0	J07	J08	J09	J10
0	1	0	0	0	0	1	0	0	0	1	A02
1	0	0	0	0	0	1	0	0	1	0	A02
1	1	0	0	0	0	1	0	0	1	1	A02

DRAM A8	FUNCTION
0	IMM
1	IMM-PF
2	UNUSED
3	WR TST
4	READ
5	READ-PF
6	RD-WR
7	READ PSG WRITE

10-1962

Figure 3-47 Control RAM Addressing

APPENDIX A
UNDERSTANDING THE MICROCODE

The control portion of the EBox comprises the DRAM and the CRAM. The DRAM has storage for 512 decimal words, one for each KL instruction. During instruction execution, the DRAM word provides information about the type of memory references by the executing instruction. It also provides an index into the main control programs contained in the CRAM.

The CRAM provides storage for 1280 microinstruction words that are structured into a complicated control program called the "microcode." This section defines and explains the microcode. Although many figures of sample listings from the microcode listing are used throughout the discussion, an assumption is made that the reader has an up-to-date copy of the microcode listing (either hard copy or microfiche). The examples shown here refer to specific sections of the listing; the reader may wish to follow the examples through the actual listing while reading this section.

The discussion begins by introducing the microcode and describing field, value, label, and microinstruction definitions. This leads into defining macros, pseduo-operators, and location control. Then, two instructions (MOVE and ADD) are illustrated, leading the reader through the microcode listing. Figures A-17 through A-23 (located at the end of this appendix) complement the discussion and define all the CRAM and DRAM fields. Refer to these figures whenever necessary.

The microcode is presented in groups, with each group (designated a through g) representing four octal digits as they appear in the listing. Each group represents one or more fields. For example, the listing for microcode address 0724 is shown in Figure A-1.

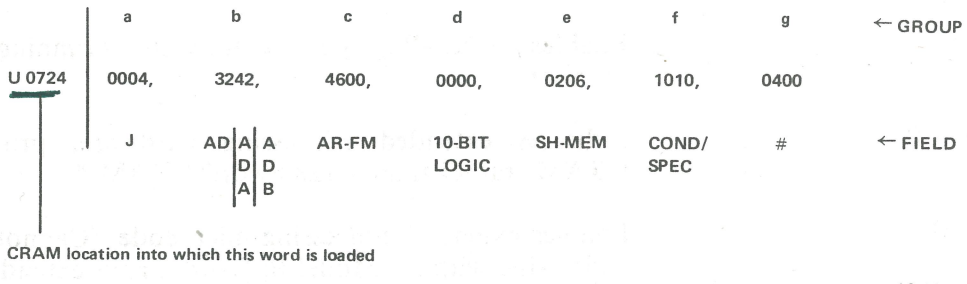


Figure A-1 Sample Microcode Listing

Each of the group's coding is defined in the respective figures listed below:

Group	Figure
a	A-17
b	A-18
c	A-19
d	A-20
e	A-21
f	A-22
g	A-23–25

The DRAM contains storage for each instruction. During instruction execution, the DRAM word (Figure A-4) provides information about the type of memory references required by the executing instruction and also provides an index into the main control program located in the CRAM.

Conditional Assembly Variable Definitions

The Conditional Assembly variables observed in the microcode listing are listed and defined below. (The definitions are presented for the variable set to 1. The values shown are the normal settings.)

Variable	Definition
TRACKS = 0	Enables storing the PC after every instruction and creates DATAI/O PI, to read/setup the PC Buffer address.*
OP.CNT = 0	Enables code to build a histogram in core to count the usage of each op code in both USER and EXEC mode.*
OP.TIME = 0	Enables code to accumulate time spent by each op code.*
FPLONG = 1	Enables KA style double precision floating-point instructions (e.g., FADL, FSBL). This feature is not supported in systems running the TOPS-20 monitor.
MULTI = 0	If operating a multiprocessor system, this suppresses cache on unpaged references; paged references are left up to EXEC.*
KLPAGE = 0	Enables the KL-Paging mode, for systems running the TOPS-20 monitor.
MODEL.B = 0	Indicates extended addressing hardware, primarily a 2K CRAM, rather than a 1280 word CRAM.*
XADDR = 0	Enables extended addressing microcode. (Cannot do extended addressing without Model B; Cannot have extended addressing without KL page).*
IMULI.OPT = 0	Enables optimization of IMULI to take only nine multiply steps.

*This feature is not supported.

Variable	Definition
SXCT = 1	Enables special XCT instruction, which allows diagnostics to generate large addresses. (Do not need SXCT with extended addressing. Cannot do it in Model B hardware.)
EXTEND = 1	Enables the extended instruction set.
DBL.INT = 1	Enables double integer instructions.
ADJBP = 1	Enables adjust byte pointer.
RPW = 1	Enables Read-Pause-Write cycles for non-cached references by some instructions.
WRTST = 0	Enables Write-Test cycles at AREAD time for instructions such as MOVEM and SETZM.*
BACK.BLT = 0	Enables BLT to decrement addresses on each step if E < RH (AC); breaks many programs.*
.SET/INSTR .STAT = 0	Enable instruction statistics code.*

Field Definitions

The actual (physical) CRAM bits are derived from the CRAM Board logic. However, no logical relationship exists between the physical bits and the respective microword bit names. Figures A-2 and A-3 are located at the end of the introductory discussion, just before the two examples. Figure A-2 shows how the physical CRAM bits are derived. Figure A-3 shows the physical bits and the corresponding microword bit position (and name). The microcode listing is ordered with respect to the microword bit positions, not the actual CRAM order.

Microcode field definitions have the form SYMBOL/ = J, K, L, M. The J parameter is only meaningful when "D" is specified as the default mechanism. The K parameter defines the field size in the number of bits (in decimal). The L parameter defines the field position (in decimal) as the bit number of the right-most bit of the field; bits are numbered from 0 on the left. Note that the position of bits in the microcode word (Figure A-3) bears no relation to the ordering of bits in the hardware microword, where fields are often broken up and scattered. The M parameter is optional; it selects a default mechanism for the field. The legal values of this parameter are the characters D, T, P, or +, where:

D	Indicates that J is the default value of the field if no explicit value is specified.
T	Is used on the time field to specify that the value of the field depends on the time parameters selected for other fields. Within the microcode, T1 parameters are used to specify functions that depend on the adder setup time; T2 parameters are used for functions that require additional time for correct selection of the next microinstruction address.
P	Is used on the parity field to specify that the value of the field should default, such that parity of the entire word is odd. If this option is selected on a field where the size (K) is zero, the microassembler attempts to find a bit somewhere in the word for which no value is either specified or defaulted.

*This feature is not supported.

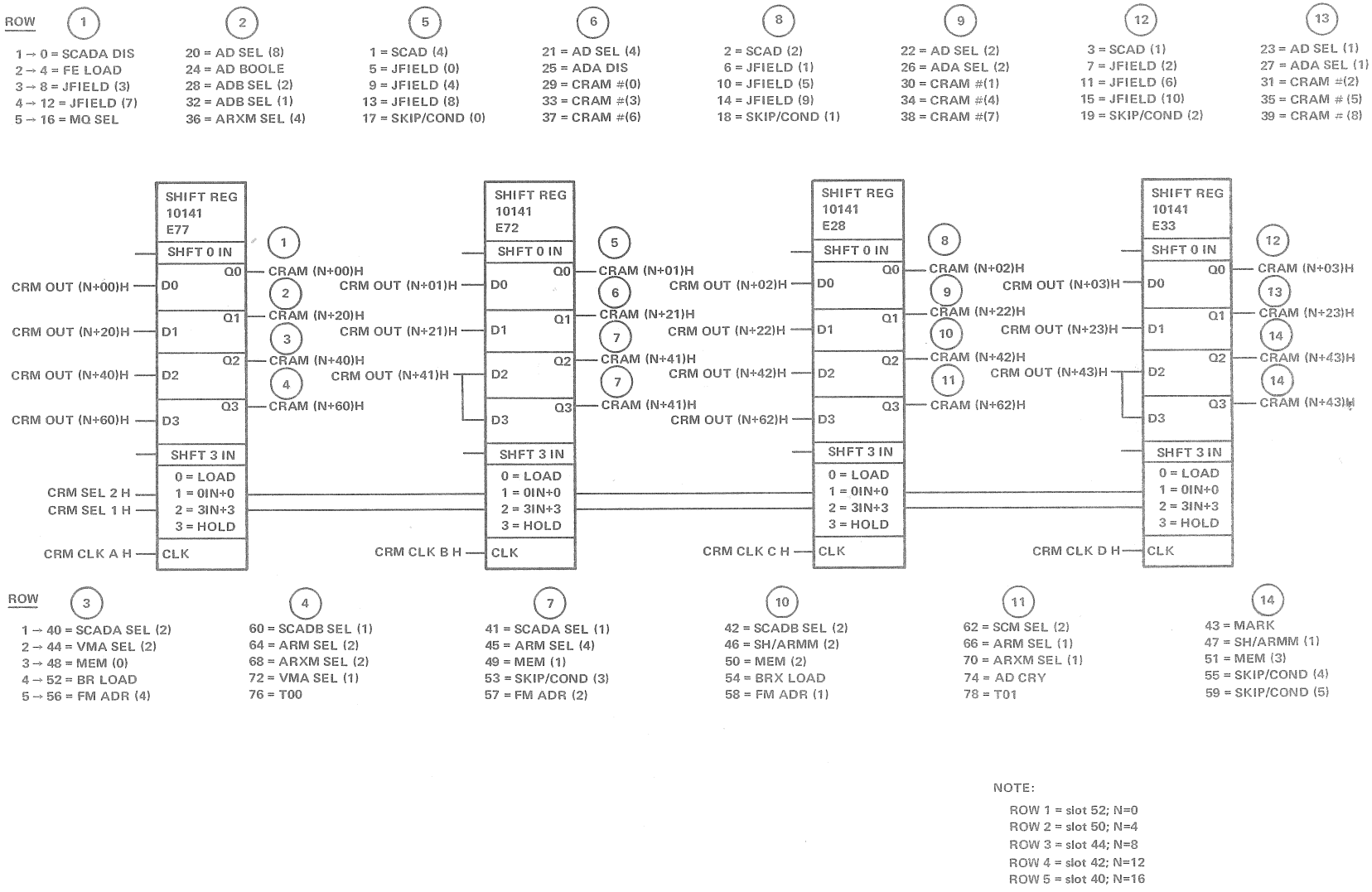


Figure A-2 CRAM Board Logic Physical Bit Position Derivation

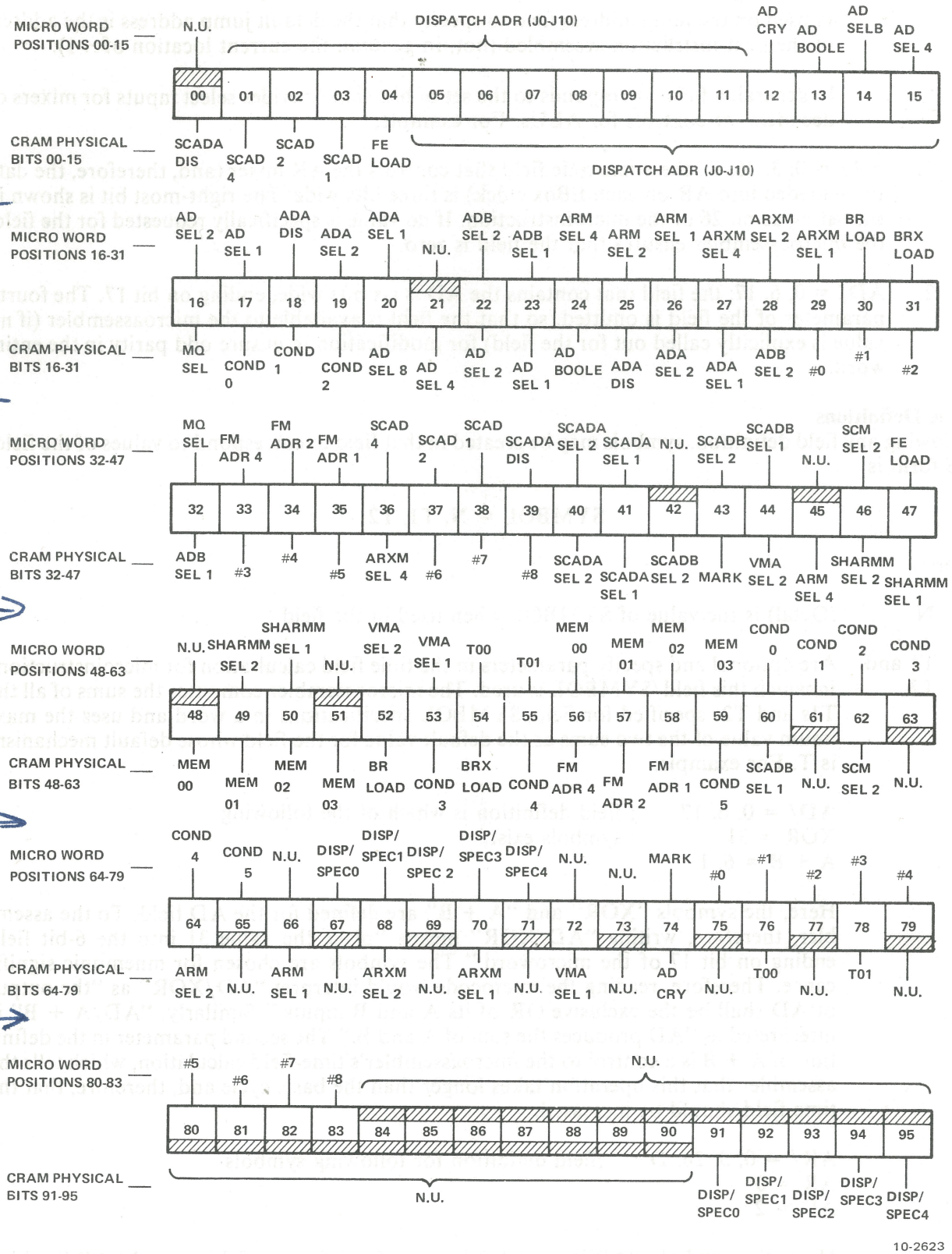


Figure A-3 Actual CRAM Physical Bit Position to Microword Bit Position Correlation

+ Is used on the jump address field to specify that the default jump address is the address of the next instruction assembled (not, in general, the current location of +1).

In general, a field corresponds to the set of bits that provides select inputs for mixers or decoders, or controls for ALUs. For example:

1. AR/ = 0, 3, 26, D; the microcode field that controls the AR mixer (and, therefore, the data to be loaded into AR on each EBox clock) is three bits wide. The right-most bit is shown in the listing as bit 26 of the microinstruction. If no value is specifically requested for the field, the microassembler ensures that the field is zero.

2. AD/ = 0, 6, 17; the field that contains the AD is six bits wide, ending on bit 17. The fourth parameter of the field is omitted, so that the field is available to the microassembler (if no value is explicitly called out for the field) for modification to ensure odd parity in the entire word.

Value Definitions

Following any field definition, symbols may be created in that field to correspond to values of the field. The form is

SYMBOL = N, T1, T2

where:

N (Octal) is the value of SYMBOL when used in the field;

T1 and T2 Are optional and specify parameters in the time field calcultation for microinstructions in which this field/SYMBOL is used. The microassembler computes the sums of all the T1s and T2s specified for field/SYMBOL specifications in a word and uses the maximum value of the two sums as the default value for the field whose default mechanism is T. For example:

```
AD/ = 0, 6, 17      ; field definition is which of the following
XOR = 31            ; symbols exist.
A + B = 6, 1
```

Here, the symbols "XOR" and "A + B" are defined for the AD field. To the assembler, therefore, writing "AD/XOR" means "place the value 31 into the 6-bit field ending on bit 17 of the microword." The symbols are chosen for mnemonic significance. Therefore, reading the microcode would interpret "AD/XOR" as "the output of AD shall be the exclusive OR of its A and B inputs." Similarly, "AD/A + B" is interpreted as "AD produces the sum of A and B." The second parameter in the definition of A + B is a control to the microassembler's time-field calculation, which tells the assembler that this operation takes longer than the basic cycle and, therefore, that the time field should be increased.

```
AR/ = 0, 3, 26, D     ;field definition for following symbols
AR = 0
AD = 2
```

Here, the symbols "AR" and "AD" are defined for the field named "AR," which controls the AR mixer. Because only the default case is used, the AR does not change unless a specific request to do so is made. Therefore, the field definition specifies zero as the default value of the field. If the AR is loaded from the AD output, AR/AD is written to set the mixer selects to pass the AD output into the AR.

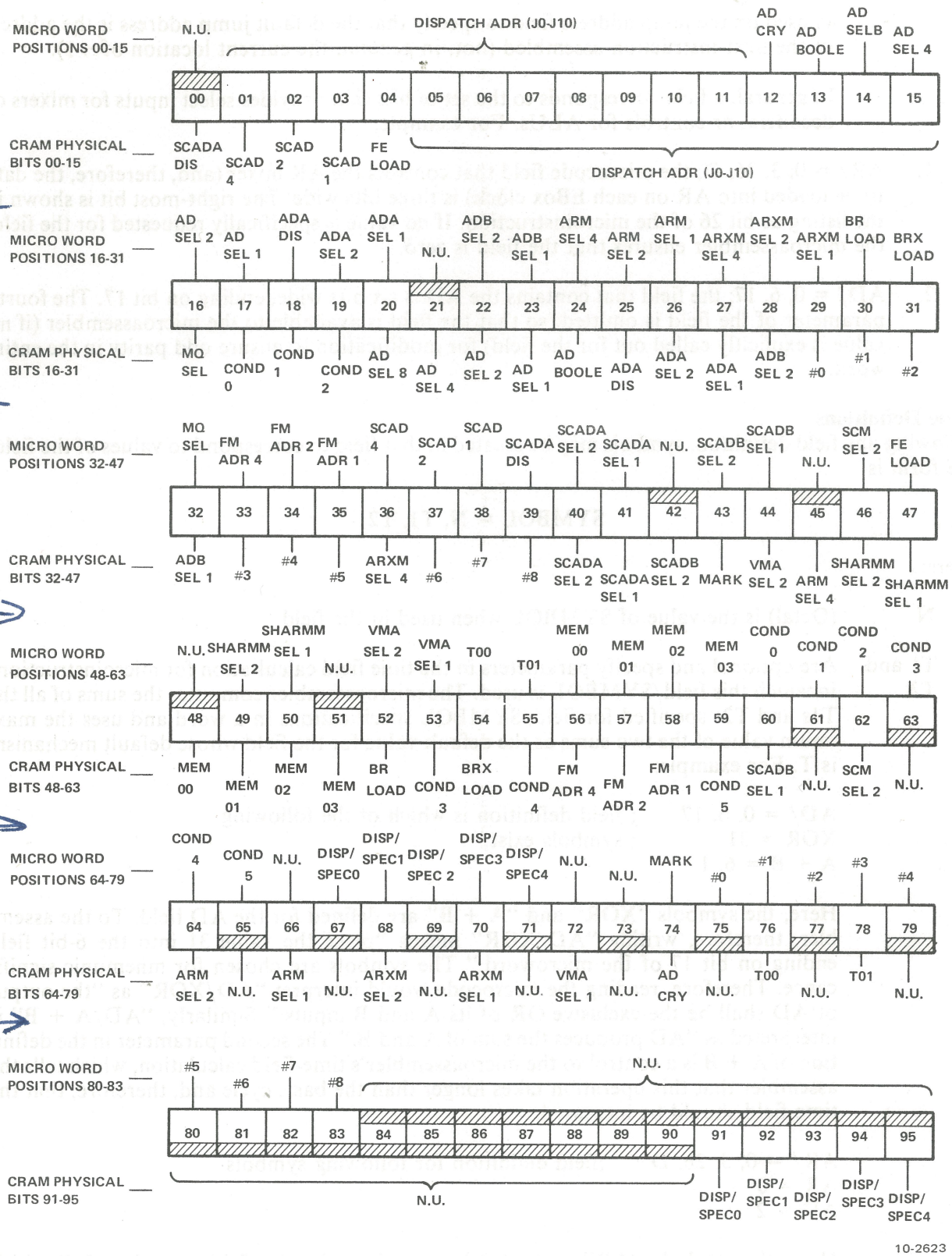


Figure A-3 Actual CRAM Physical Bit Position to Microword Bit Position Correlation

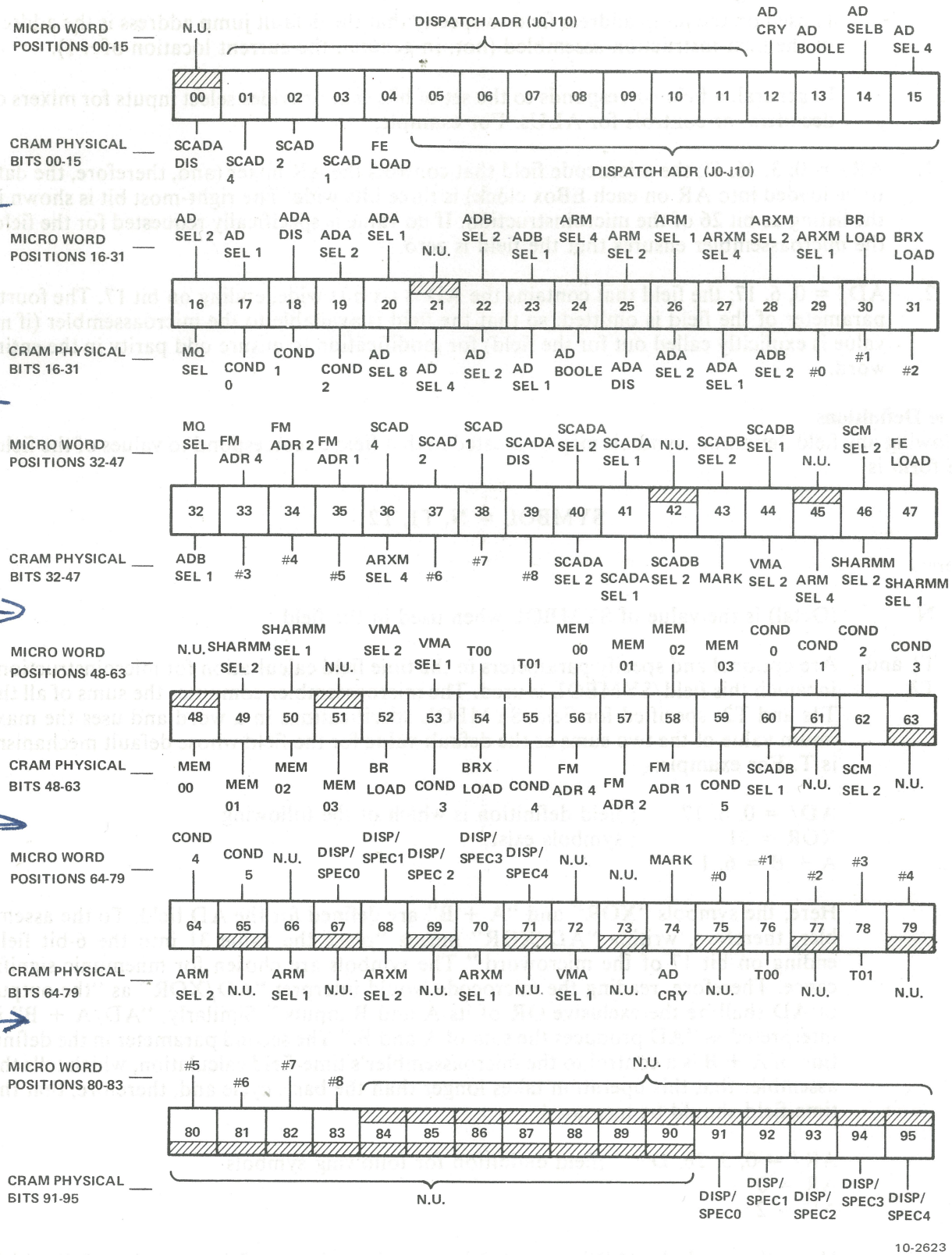


Figure A-3 Actual CRAM Physical Bit Position to Microword Bit Position Correlation

Label Definitions
A microinstruction may be labeled by a symbol followed by a colon preceding the microinstruction definition. The address of the microinstruction becomes the value of the symbol in the field titled "J." For example:

```
TOP:  J/TOP
```

This is a microinstruction whose J field (Jump Address) contains the value "TOP." It also defines the symbol "TOP" to be the address of itself. Therefore, if executed by the microprocessor, the microinstruction would loop on itself.

Comments
A semicolon anywhere on a line causes the remainder of the line to be ignored by the assembler; it is purely information to the reader. For example:

```
AD/0, 6, 17          ;field definition in which following symbols
                     ;exist.
```

Only AD/0, 6, 17 is relevant to the assembler; that data following the semicolon is useful information to the reader.

Microinstruction Definition
A word of microcode is defined by specifying a field name, followed by a slash (/), followed by a value. The value may be a symbol defined for that field, an octal digit string, or a decimal digit string (distinguished by the fact that it contains "8" and/or "9" and/or is terminated by a period). Several fields may be specified in one microinstruction, by separating field/value specifications with commas. For example:

```
ADB/BR, ADA/AR, AD/A + B, AR/AD
```

In this example, the field named "ADB" is given the value named "BR" (to cause the mixer on the B side of AD to select BR); field "ADA" has the value "AR;" field has the value "A + B," and field "AR" has the value "AD."

Continuation
The definition of a microinstruction may be continued onto two or more lines by breaking the definition after any comma. That is, if the last nonblank, noncomment character on a line is a comma, the instruction specification is continued on the following line. For example:

```
ADB/BR, ADA/AR,          ;select AR and BR as AD inputs
    AD/A + B, AR/AD      ;take the sum into AR
```

By convention, continuation lines are indented on extra tab.

Macros
A macro is a symbol, the value of which is one or more field/value specifications and/or macros. A macro definition is a line containing the macro name followed by a quoted string that is the value of the macro. For example:

```
AR   AR + BR        "ADB/BR, ADA/AR, AD/A + B, AR/AF"
```

The appearance of a macro in a microinstruction definition is equivalent to the appearance of its value.

Pseudo-Operators
The microassembler contains ten pseudo-operators:

1-2.	.DCODE and .UCODE	Select the RAM into which subsequent microcode is loaded and, therefore, the field definitions and macros that are meaningful in subsequent microcode.
3.	.TITLE	Defines a string of text to appear in the page header.
4.	.TOC	Defines an entry for the Table of Contents at the beginning.
5.	.SET	Defines the value of a conditional assembly parameter.
6.	.CHANGE	Redefines a conditional assembly parameter.
7.	.DEFAULT	Assigns a value to an undefined value.
8.	.IF	Enables assembly if the value of the parameter is not zero.
9.	.IFNOT	Enables assembly if the parameter value is zero.
10.	.ENDIF	Re-enables assembly if suppressed by the parameter named.

Location Control
A microinstruction labeled with a number is assigned to that address. The character "=" at the beginning of a line, followed by a string of 0s, 1s, and/or *s, specifies a constraint on the address of the following microinstructions. The number of characters in the constraint string (excluding the "=") is the number of low-order bits contained in the address. The microassembler attempts to find an unused location whose address has zero bits in the positions corresponding to 0s in the constraint string and one bits where the constraint has 1s. Asterisks denote "don't care" bit positions.

If any zeros are in the constraint string, the constraint implies a block of (2 * * N) microwords, where N is the number of 0s in the string. All locations in the block have 1s in the address bits corresponding to 1s in the string. Bit positions denoted by *s are the same in all block locations.

In such a constraint block, the default address progression is counting in the "0" positions of the constraint string, but a new constraint string occurring within a block may force skipping over some locations of the block. Within a block, a new constraint string does not change the pattern of default address progression, it merely advances the location counter over those locations. The microassembler fills them in later.

A NULL constraint string ("=" followed by anything except 0, 1, or *) serves to terminate a constraint block. For example:

a. = 0

This specifies that the low-order address bit must be zero. The microassembler finds an even-odd pair of locations and places the next two microinstructions into them.

b. = 11

This specifies that the two low-order bits of the address must both be ones. Because there are no zeros in this constraint, the assembler finds only one location meeting this constraint.

c. = 0*****

This specifies an address in which the 40_8 bit is zero. Due to the implementation of this feature in the assembler, the default address progression applies only to the low-order five bits. Therefore, this constraint finds one word in which the 40_8 bit is 0 and does not attempt to find one where that bit is a 1.

Microcode Examples

The following paragraphs lead the reader through the microcode, while defining two instructions: MOVE and ADD. The requirements that the microcode is loaded and running (i.e., in the HALT loop) are assumed. A dispatch (test for an interrupt) occurs during a HALT loop. Once an interrupt is present, the microcode leaves the HALT loop and goes to the first microinstruction.

MOVE Instruction

Refer to Figure A-4. The initial dispatch is a NICOND Dispatch. It is a decision starting at microcode address 140 that is used to decide which condition (e.g., TRAP, NICOND) is satisfied. Looking up Next Instruction Dispatch in the microcode listing Table of Content refers the reader to line 2549 in the listing. The decision begins at line 2549. Notice that the actual decisions and respective implementations begin at microcode address 140 (NEXT), and assuming a NICOND Dispatch is present, the listing refers the reader to NEXT + 12 (microcode address 152).

The NICOND Dispatch is the normal case; the instruction is in the ARX and begins execution. Location NEXT + 12 leads (jump to the correct decision) the reader to microcode address 152 (XCTGO), line 2606. Notice in the listing (and Figure A-5):

At XCTGO, on line 2606, the comments state "save the instruction, sign extend Y and calculate the effective address (EA)." The macros define all the things that happen here. Initially, one should consider where to go next. That information is contained in:

1. The J-field, which typically contains the "suggested" next address. In this example, it is 160. Whether that is used or not depends on item 2.

2. The Dispatch (or SPEC) field.

The SPEC field follows the "f" column in the microcode listing (Figure A-22). Specifically, the field observes the last two digits of the "f" column. In this example, those digits are "36." Going to Figure A-22, notice that a decoded 36 in the SPEC field is an EA MODE Dispatch.

An EA calculation is called for, which indicates that under certain conditions the J-field (160) may not be the actual next address. These conditions are Indexing (bits 14–17 of the instruction), Indirection (bit 13), both conditions, or neither condition. In this case, EA MODE dispatch looks at those bits of information in the instruction and then ORs them with 160 (the J-field). Because this simple MOVE instruction uses neither indexing nor indirection, go directly to 160. This appears on line 2647 (if you cannot easily locate this, go to the index at the rear of the listing, look up address 160 and find that line 2647 is where microcode address 160 appears). Refer to Figure A-6.

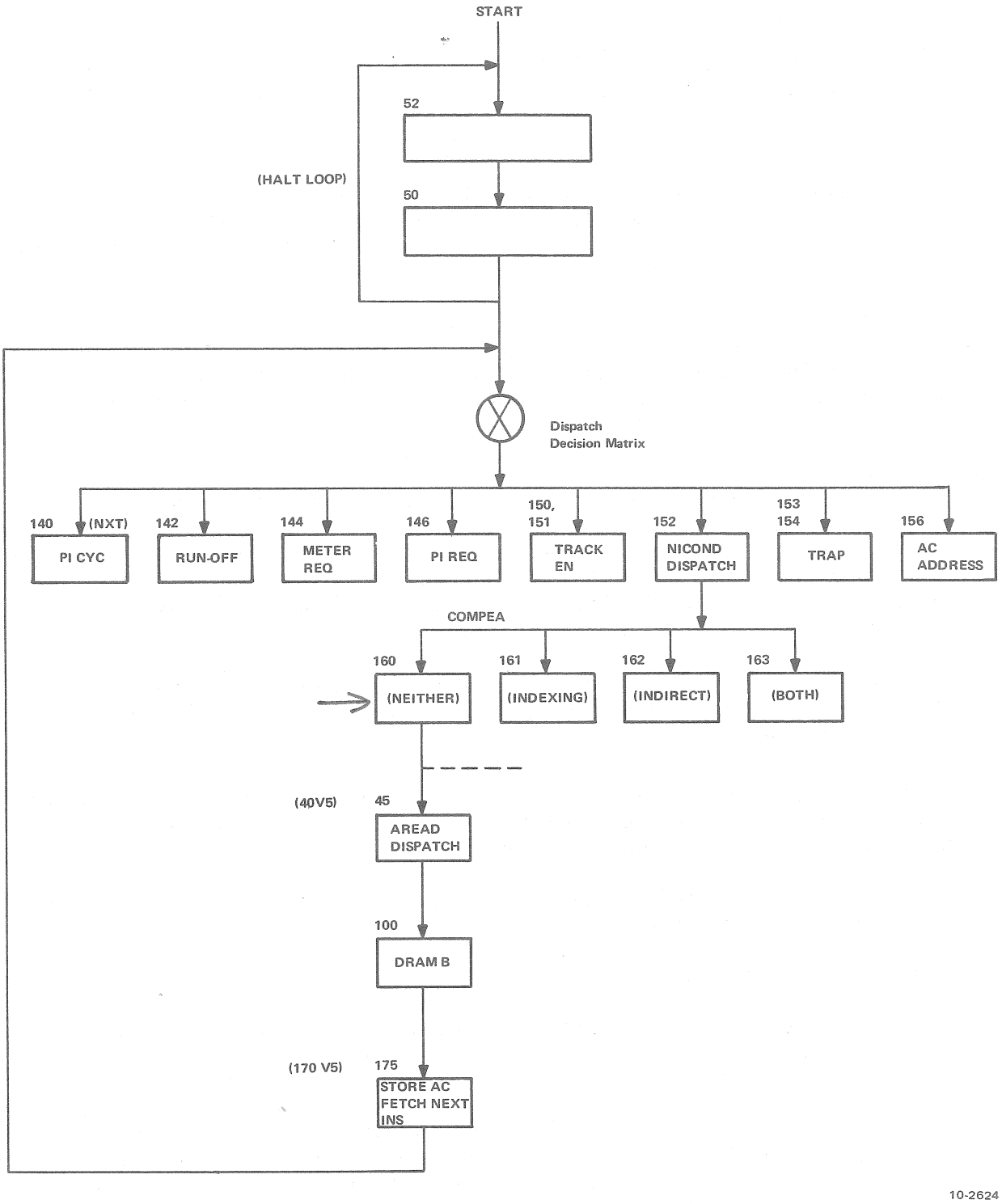


Figure A-4 MOVE Instruction Flow Diagram

```
                                                ;2606 XCTGO: BRX/ARX,AR  ARX,SET ACCOUNT EN,   ;SAVE INSTR, SIGN EXTEND Y,
U0152,0160,0001,4022,2000,2136,0105 ;2607                    XR,EA MOD DISP, J/COMPEA   ;GO CALCULATE EA
```

Figure A-5 Microcode Address 152

```
U0160,0000,3701,0000,0000,0204,0002,0000  ;2647  COMPEA: GEN AR, A READ          ;LOCAL
```

Figure A-6 Microcode Address 160

Again looking at the "f" column, observe the SPEC field is "02." Checking Figure A-22, SPEC code 02 indicates doing an A READ Dispatch by stating DRAM A RD. Go to the microcode listing index for DRAM words (it appears just before the microcode address index). The MOVE instruction is op code 200. Find 200 and notice it refers you to line 2782. Refer to Figure A-7.

```
D0200,5500,0100      ;2782  200:  R-PF,  AC,  J/MOVE     ;BASIC MOVE
```

Figure A-7 DRAM Word 200

This is the DRAM word for the basic MOVE instruction. The A-field is a "5" (Figure A-26). This "5" is ORed with 40 (a constant used whenever an A RD DISPATCH is performed) and the J-field (0000) of microinstruction 160. This results in a "45." Turning again to the index, look up microcode address 45. The index indicates line 2711; see Figure A-8.

```
                                                ;2711 BR/AR,FIN XFER, I FETCH,   ;GET OPERAND, PREFETCH,
U0045,0000,3240,0043,0000,0226,0001,0400  ;2712                TIME 3T, IR DISP, J/O   ;& START EXECUTE
```

Figure A-8 Microcode Address 45

This part of the microcode states: get the operand (from the MBox), begin a prefetch of the next instruction, and begin instruction execution. Notice also in the macros, that an IR Dispatch is called. Looking now at the SPEC field, it is "01;" looking this up in Figure A-22 states DRAM J DISPATCH. A DRAM J DISPATCH dictates calculating where to go by taking *only* the J-field of the DRAM word as the address. In the case of the simple MOVE instruction (look back at Figure A-7), notice the A-field is "5," the B-field is "5," and the J-field is "100."

Looking up microcode address 100 in the index leads the reader to line 2819 (Figure A-9).

```
U0100,0170,0001,0000,0000,0005,0033,0000  ;2819  MOVE:  EXIT     ;STORE AS IS FROM AR
```

Figure A-9 Microcode Address 100

The SPEC field is "33" and, again referring to Figure A-22, now a DRAM B is called. DRAM B is the actual "store the operand." The MOVE began by fetching the operand and placing it in the AR. Finally, it is placed in a particular AC. The DRAM B Dispatch takes the B-field of the DRAM word (5) and ORs it with the J-field (170) of the current microinstruction (address 100). This results in: 170 V 5 = 175. The index takes the reader to line 2749; see Figure A-10.

```
U0175,0140,4001,0000,0403,0002,1006,0000  ;2749  STAC: AC0_AR,NXT INSTR  ;NORMAL AND IMMEDIATE MODES
```

Figure A-10 Microcode Address 175

Observing the SPEC field indicates "06;" this is a NICOND Dispatch. Also, the J-field is 140, taking the reader to the original decision matrix. Again, all the possibilities are considered when the next instruction arrives and the process continues.

Not all fields were discussed here, only the major fields. All fields are illustrated and defined in Figures A-17 through A-26. It is left to the reader to check the unmentioned code fields with the respective defining figures.

ADD Instruction

Many of the assumptions used in the MOVE example are used again here (refer to Figure A-11). Assume that the last instruction was a NICOND Dispatch; go through the decision matrix to microcode address 152. Assume Indexing this time, this leads the reader to address 161. Locate address 161 on line 2648 of the listing (see also Figure A-12).

Indexing is handled at this time. The AR is added to the contents of the XR (Index register) to generate the EA. Also, an A READ Dispatch is called out. The A READ leads to the next microcode instruction, which is where the operand is located. Assume AC3 is being used (for example) and its content is "50;" assume the Y-field contains "100." This results in EA = 150.

Again, because of a COMP EA (EA calculation), a "40" is forced into the J-field by the hardware during the A READ Dispatch. Figure A-13 shows the DRAM word for the ADD (270) instruction. Use the DRAM index to locate line 4091.

The A-field of the DRAM word is "5." This, ORed with the forced "40," results in "45." This is microcode address 45, just as in the MOVE example. Locate address 45 on line 2712; this is where the operands are fetched (see Figure A-14).

A "01" is in the "f" column of the SPEC field, a DRAM J Dispatch. Looking back at Figure A-13, notice that the J-field of the DRAM word is "504." Go to the microcode address index and locate address 504 at line 4098 (see Figure A-15).

This is where the ADD takes place. The macros state "A plus B (the two operands) into the AD." The SPEC field (Figure A-13) is a "5." The J-field of the current microinstruction is 170. These two are ORed, resulting in 175. Using the index again, locate address 175 on line 2749 (see Figure A-16).

The operand is stored in AC0 and the J-field leads the reader back to location 140 again, the NICOND Dispatch. The microcode is now ready for the next instruction.

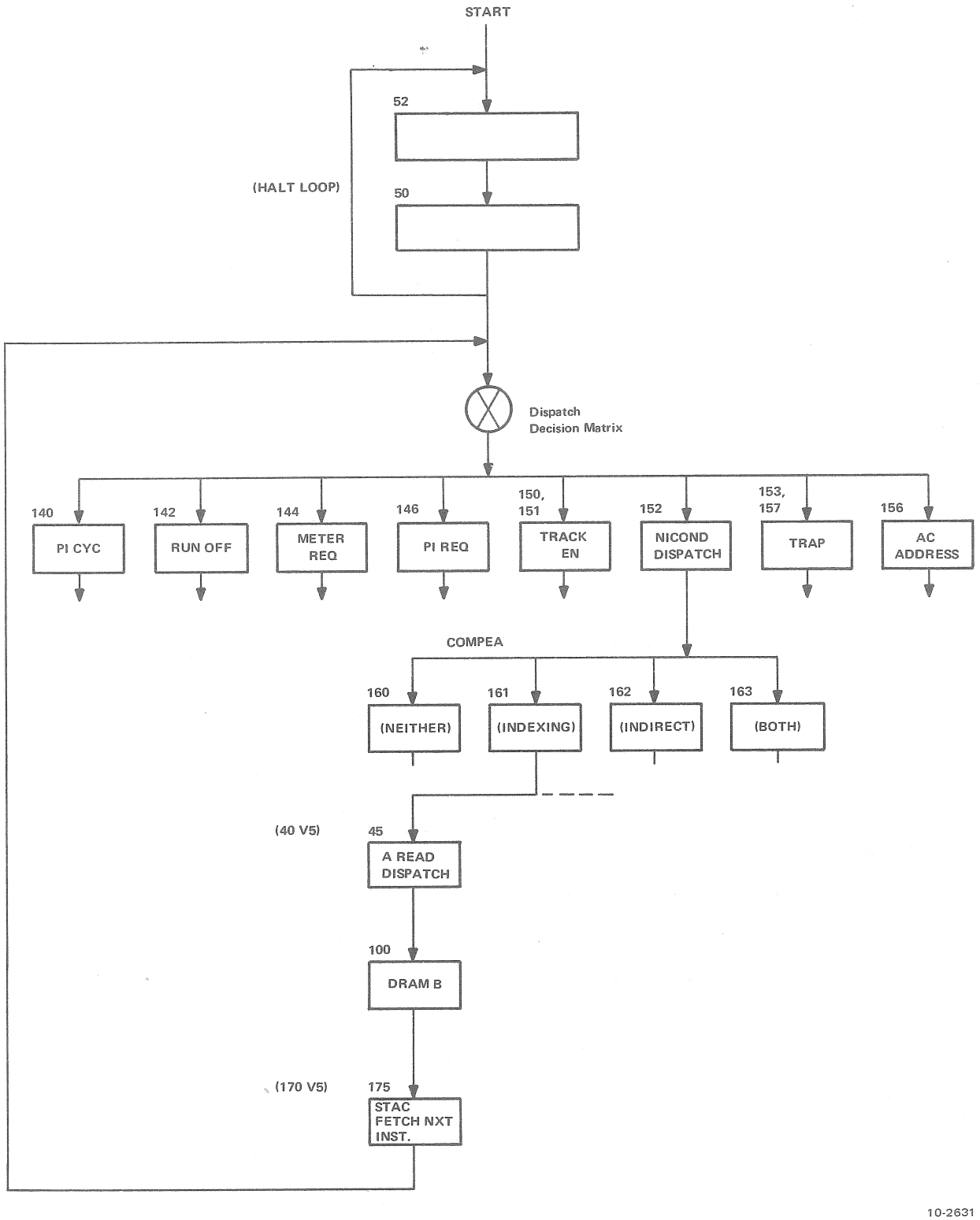


Figure A-11 ADD Instruction Flow Diagram

```
U0160,0000,3701,0000,0000,0204,0002,0000 ;2647  COMPEA: GEN AR, A READ              ;LOCAL
U0161,0000,0600,0002,4000,2224,0002,0000 ;2648 *     GEN AR + XR, INDEXED, A READ  ;LOCAL UNLESS XR> 0
```

Figure A-12 Microcode Address 160, 161

```
D 0270,5500,0504              ;4091  270:  R-PF,  AC,    J/ADD
```

Figure A-13 DRAM Word 270

```
                                        ;2711  BR/AR, FIN XFER, I FETCH  ;GET OPERAND, PREFETCH,
U0045,0000,3240,0043,0000,0226,0001,0400 ;2712  TIME 3T, I/R DISP, J/O     ;& START EXECUTE
```

Figure A-14 Microcode Address 45

```
U0504,0170,0600,2000,0000,0025,1333,0000  ;4098  ADD:  AR  AR*AC0, AD/A+B, AD FLAGS, EXIT
```

Figure A-15 Microcode Address 504

```
U0175,0140,4001,0000,0403,0002,1006,0000  ;2749  STAC:  AC0_AR,NXT INSTR  ;NORMAL AND IMMEDIATE MODES
```

Figure A-16 Microcode Address 175

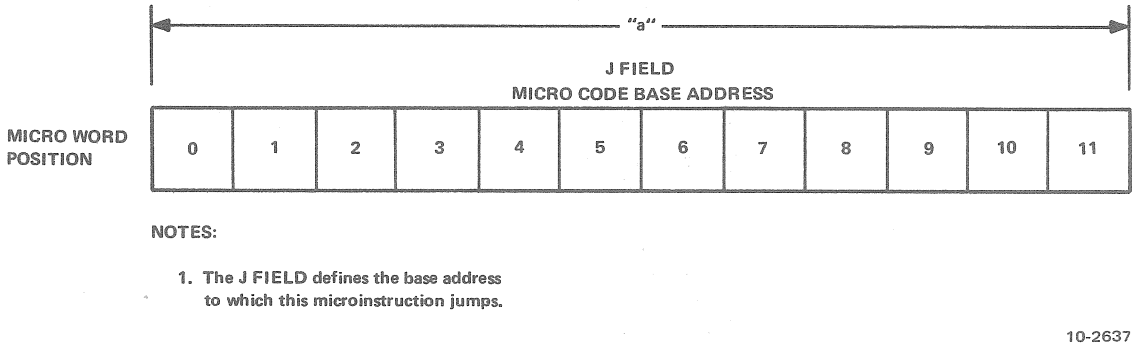


Figure A-17 Microword "a" Field

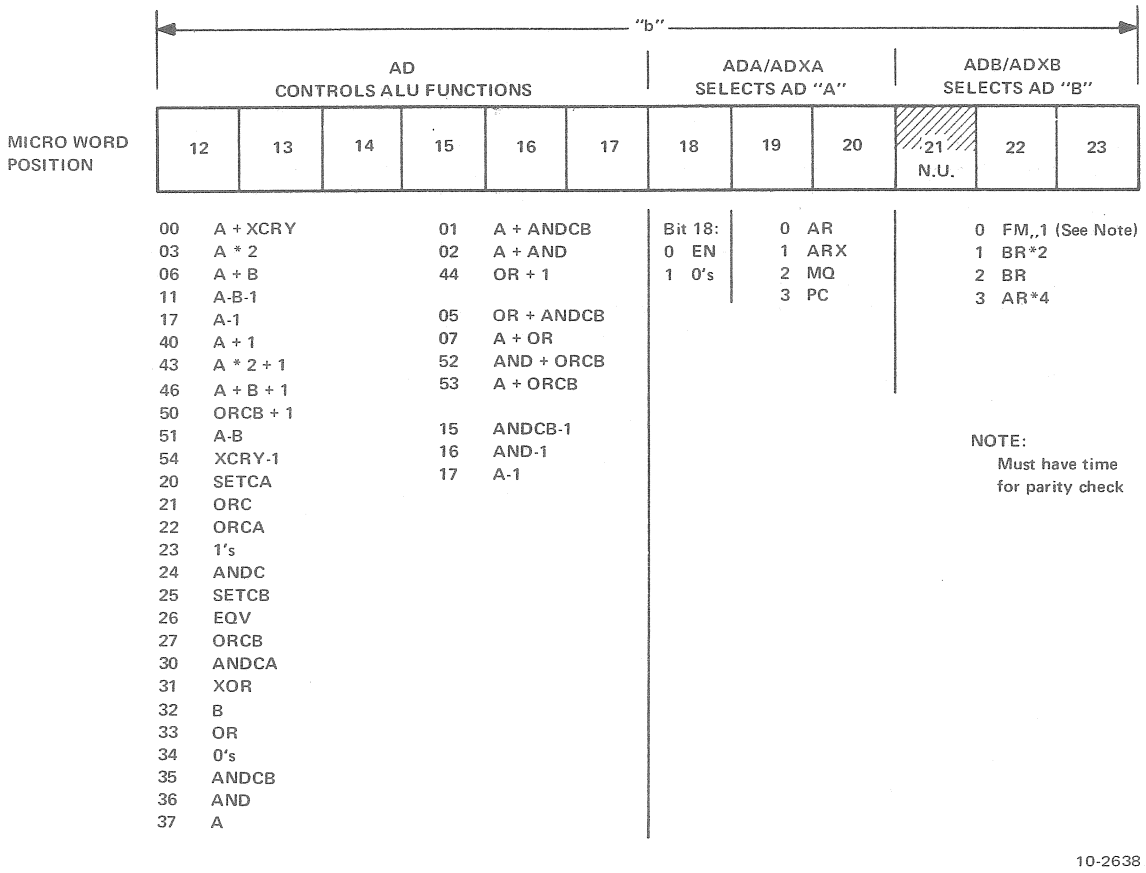


Figure A-18 Microword "b" Field

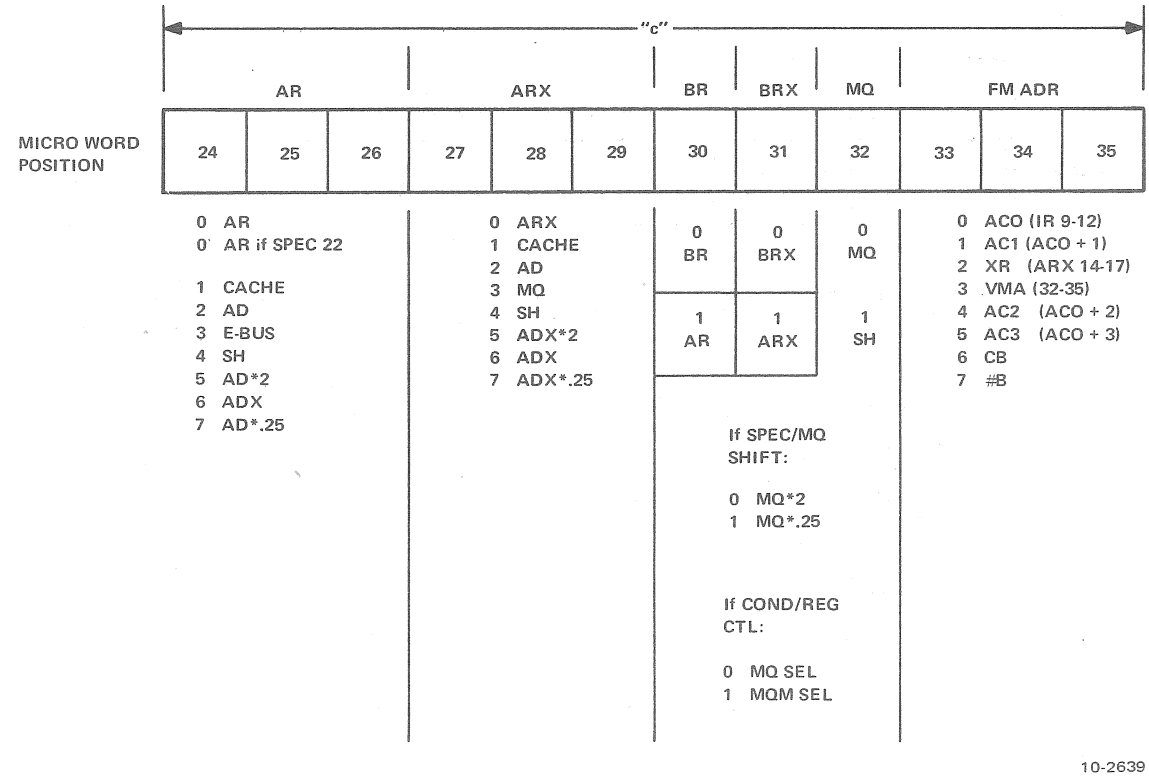
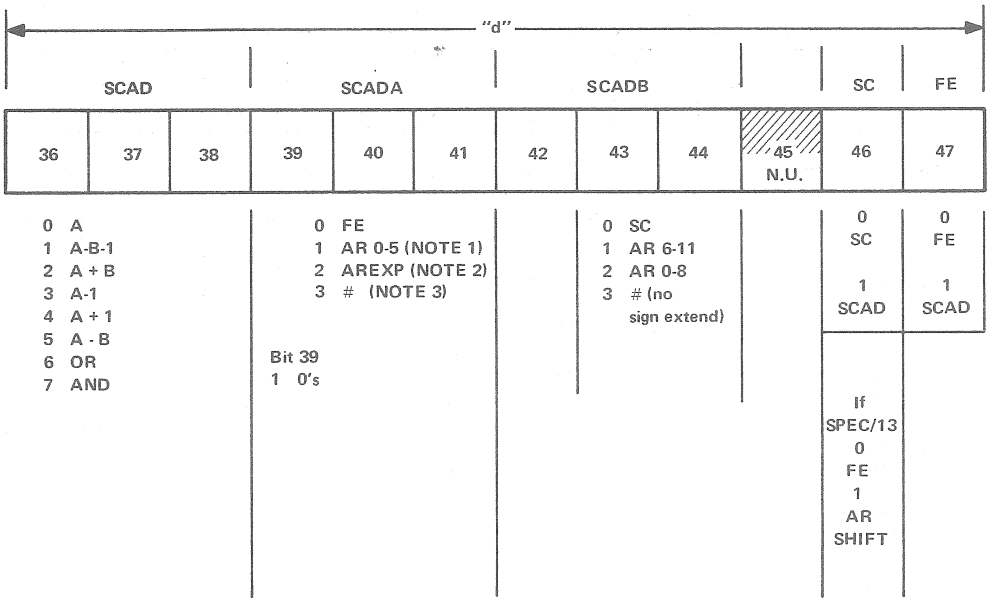


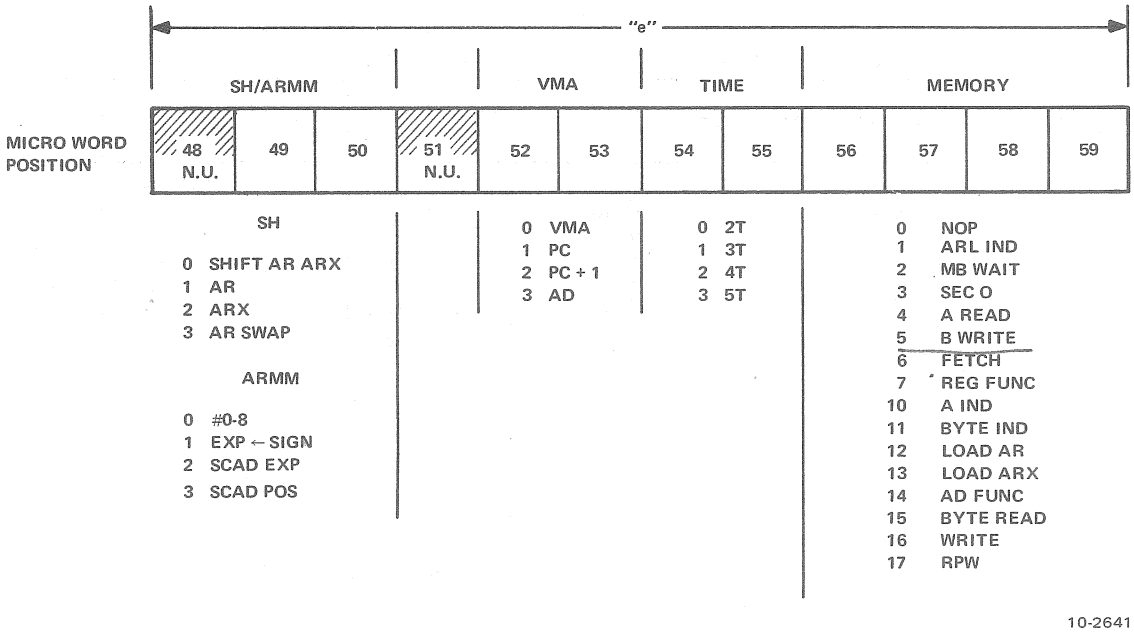
Figure A-19 Microword "c" Field



NOTES:
1. Byte Pointer Position Field
2. |AR (01-08)| XOR |AR00|
3. Sign extended with #00

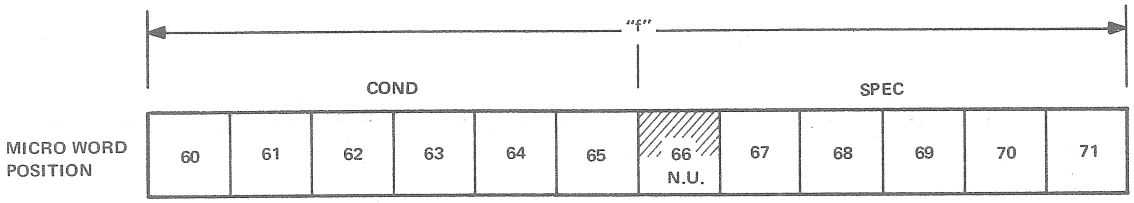
10-2640

Figure A-20 Microword "d" Field



10-2641

Figure A-21 Microword "e" Field



NON-SKIP FUNCTIONS

0	NOP
1	LD AR 0-8
2	LD AR 9-17
3	LD AR 18-35
4	AR CLR
5	ARX CLR
6	ARL IND
7	REG CTL
10	FM WRITE
11	PCF ← #
12	FE SHRT
13	AD FLAGS
14	LOAD IR
15	SPEC INST
16	SR ← #
17	SEL VMA
20	DIAG FUNC
21	EBOX STATE
22	EBUS CTL
23	MBOX CTL
24	SPARE
25	SPARE
26	SPARE
27	SPARE
30	VMA ← #
31	VMA ← # + TRAP
32	VMA ← # + MODE
33	VMA ← # +AR 32-35
34	VMA ← # +PI*2
35	VMA DEC
36	VMA INC
37	LD VMA HELD

SKIP FUNCTIONS

40	SPARE
41	EVEN PAR
42	BR0
43	ARX 0
44	AR 18
45	AR 0
46	AC #0
47	SC 0
50	SC. LT. 36
51	SCAD 0
52	SCAD ≠ 0
53	ADX 0
54	AD CRY 0
55	AD 0
56	AD ≠ 0
57	SPARE
60	FETCH
61	KERNEL
62	USER
63	PUBLIC
64	RPW REF
65	PI CYC
66	EBUS GRANT
67	EBUS TRANS
70	INTRPT
71	-START
72	RUN
73	IO LEGAL
74	
75	EBOX PF
76	AC REF
77	-MTR REQ

DISPATCHES

0	DIAG
1	DRAM J
2	DRAM A RD
3	RETURN
4	PG FAIL
5	SR
6	NICOND
7	SH 0-3
30	MUL
31	DIV
32	SIGNS
33	DRAM B
34	BYTE
35	NORM
36	EA MODE
37	EA TYPE

NON-DISPATCHES

10	NOP
11	INH CRY 18
12	MQ SHIFT
13	SCM ALT
14	CLR FPD
15	LD PC
16	XCRY AR0
17	GEN CRY 18
20	SEC HOLD
21	CALL
22	ARL IND
23	MTR CTL
24	FLAG CTL
25	SAVE FLAGS
26	SP MEM CYCLE
27	AD LONG

10-2642

Figure A-22 Microword "f" Field

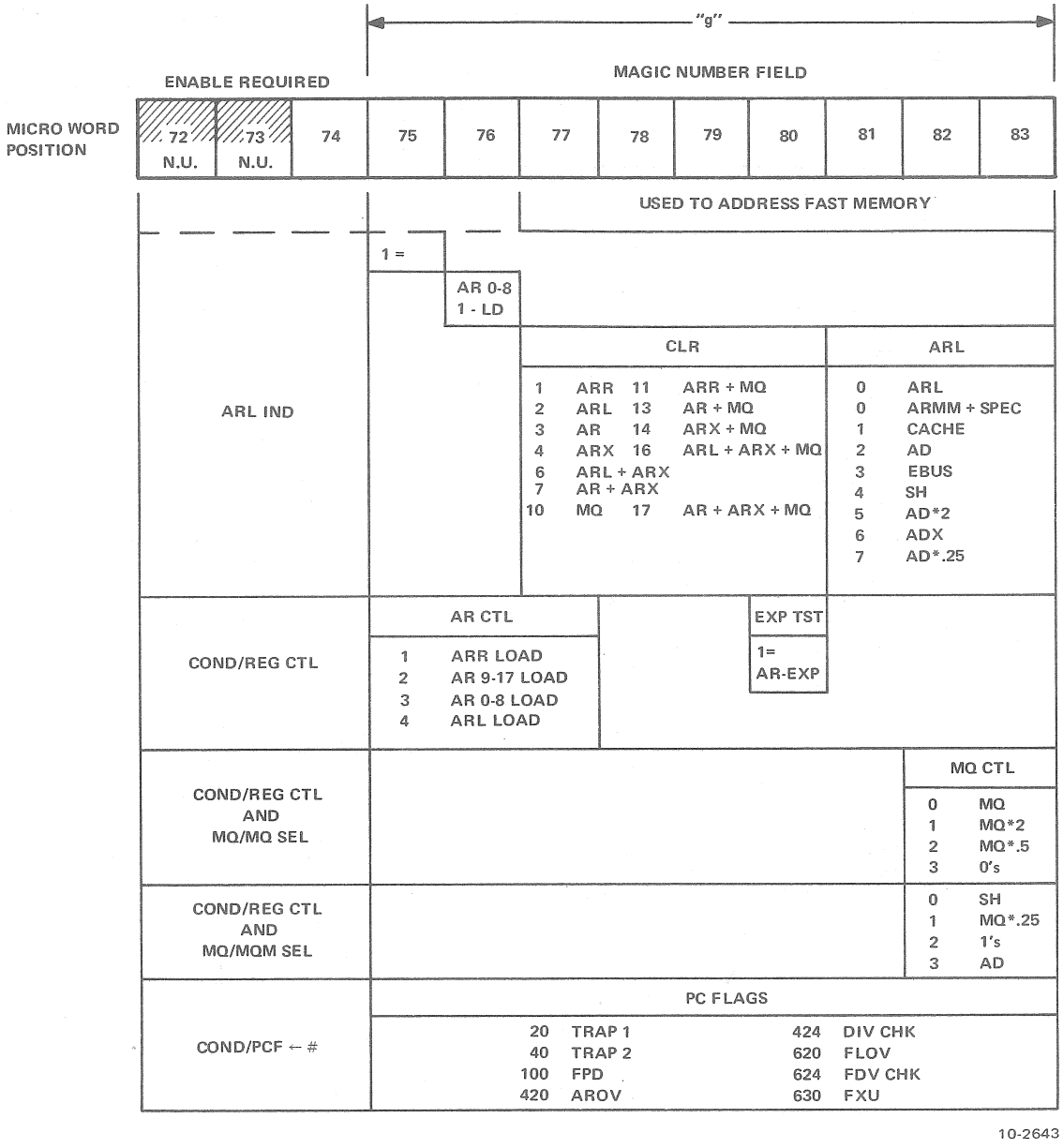
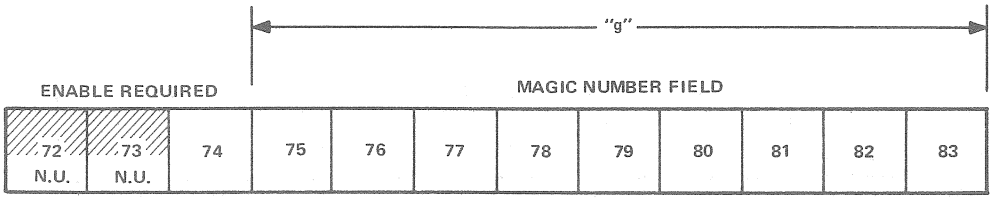


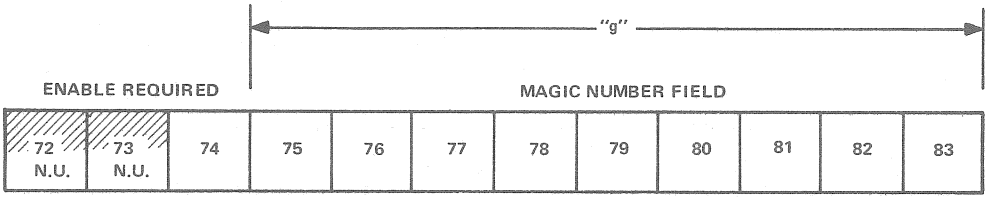
Figure A-23 Microword "g" Field (Magic Numbers)(Sheet 1 of 3)



SPEC/FLG CTL	FLAG CTL			
	20	SET FLAGS	502	DISMISS
	412	PORTAL	602	JFCL
	420	RSTR FLAGS	622	JFCL + LD
	442	HALT		
COND/SPEC INST	SPEC INSTR			
	4	INSTR ABORT	100	IN H PC + 1
	10	INTRPT IN H	200	KERNEL CYCLE
	10	CONT	302	HALTED
	20	PXCT	310	CONS XCT
	40	SXCT	704	SET PI CYCLE
MEM/FETCH	FETCH			
	201	COMP	400	UNCOND
	202	SKIP	502	JUMP
	203	TEST	503	JFCL
SPEC/SP MEM CYCLE				
	2	CACHE IN H	111	EPT
	10	EPT EN	200	USER
	20	UPT EN	221	UPT
	31	PT	400	FETCH
	40	SEC 0	431	PT FETCH
	100	EXEC	511	EPT FETCH
	101	UNPAGED	621	UPT FETCH
MEM/REG FUNC	MREG FUNC			
	7	SBUS DIAG	505	WR REFILL RAM
	140	MPA	601	LOAD CCA
	502	READ UBR	602	LOAD UBR
	503	READ EBR	603	LOAD EBR
	504	READ ERA		
COND/MBOX CTL	MBOX CTL			
	00	NORMAL	21	CLR PT LINE
	01	PT DIR CLR	100	SET I/O PF ERR
	10	PT WR	200	SET PAGE FAIL
	20	PT DIR WR		

10-2644

Figure A-23 Microword "g" Field (Magic Numbers)(Sheet 2 of 3)



COND/EBUS CTL	EBUS CTL	
	0 REL EBUS	26 DATAO
	1 INPUT	27 DATAI
	2 DATA I/O	30 I/O INIT
	4 DISABLE CS	60 EBUS DEMAND
	10 CTL - IR	100 REL EBUS
	20 EBUS NO DEMAND	200 REQ EBUS
		400 GRAB E EBUS
COND/DIAG FUNC	DIAG FUNC	
	400 .5 μSEC	511 DATAI PAG (L)
	404 LD PA LEFT	511 RD PERF CNT
	405 LD PA RIGHT	512 CONI APR (L)
	406 CONO MTR	512 RD EBOX CNT
	407 CONO TIM	513 DATAI APR
	414 CONO APR	513 RD CACHE CNT
	415 CONO PI	514 RD INTRVL
	416 DATAO APR	516 CONI MTR
	425 LD AC BLKS	517 RD MTR REQ
	426 LD PCS + CWSX	530 CONI PI (PAR)
	500 CONI PI (R)	531 CONI PAG
	501 CONI PI (L)	567 RD EBUS REG
	510 CONI APR (R)	620 DATAO PAG
	510 RD TIME	
SPEC/MTR CTL	0 CLR TIM	1 CLR PERF
	2 CLR E COUNT	3 CLR M COUNT
	4 LD PA LH	5 LD PA RH
	6 CONO MTR	7 CONO TIM

10-2645

Figure A-23 Microword "g" Field (Magic Numbers)(Sheet 3 of 3)

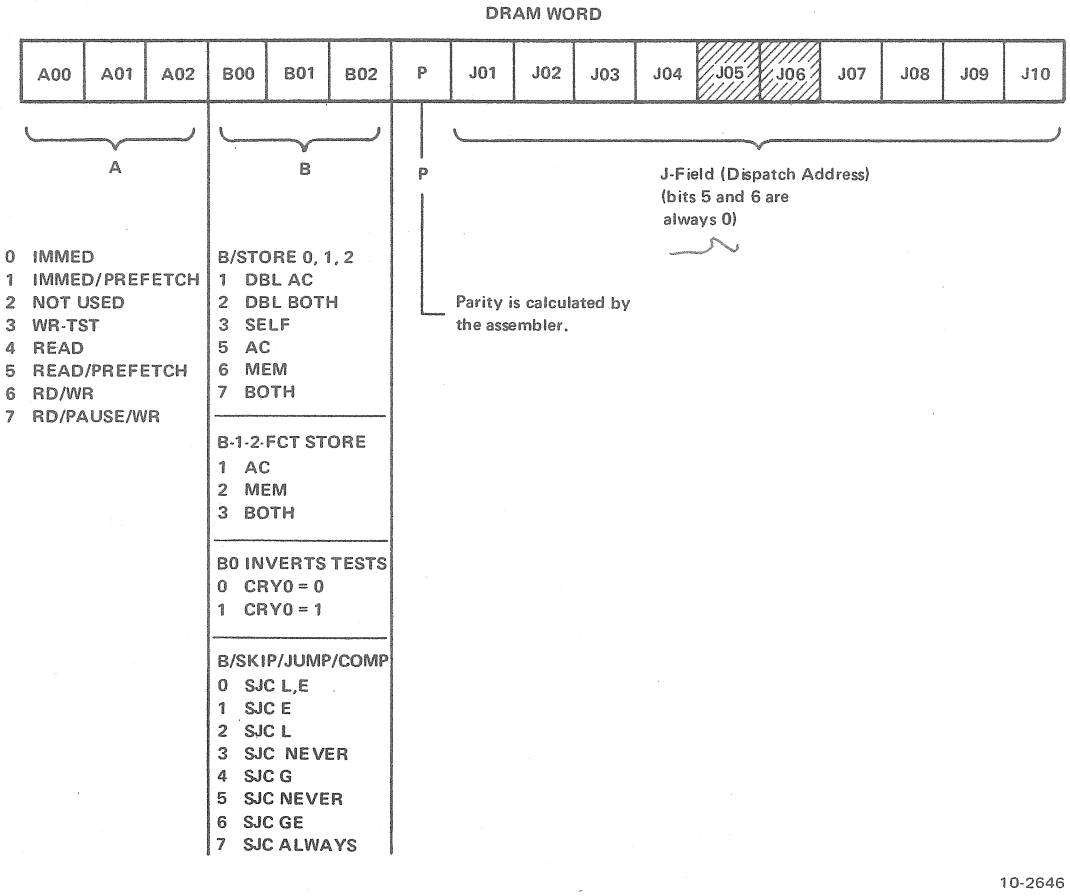


Figure A-24 DRAM Word Format

APPENDIX B
ABBREVIATIONS AND MNEMONICS

	A
AC	Accumulator
ACKN	Acknowledge
ACT	Action
AD	Adder
ADA	Adder A
ADB	Adder B
ADR	Address
ADX	Adder Extension
AF	Action Flag
ALT	Alternate
ALU	Arithmetic Logic Unit
APR	Arithmetic Processor Register
AR	Arithmetic Register
ARL	Arithmetic Register Left
ARM	Arithmetic Register Mixer
ARMM	Arithmetic Register Mixer Mixer
ARR	Arithmetic Register Right
ARX	Arithmetic Register Extension
ARXL	Arithmetic Register Extension Left
ARXM	Arithmetic Register Extension Mixer
ARXR	Arithmetic Register Extension Right

	B
BOOLE	Boolean
BR	Buffer Register
BRK	Break
BRX	Buffer Register Extension
BUF	Buffer

	C
CRAM	Control RAM
CRY	Carry
CS	Controller Select
CSH	Cache
CTL	Control
CTOM	Controller-to-Memory or Cache-to-Memory
CTR	Counter
CWSX	Called With Special Execute
CYC	Cycle

	D
D	Data
DIAG	Diagnostic
DIR	Directory
DIS	Disable
DISP	Dispatch
DIV	Divide
DRAM	Dispatch RAM

	E
E	Effective Address
E to T	ECL to TTL
EBR	Executive Base Register
EBUS	Execution Bus
ECL	Emitter-Coupled Logic
EDP	EBox Data Path
EN	Enable
ENA	Enable
ERR	Error
ERA	Error Address
EPT	Executive Process Table
EX	Extension
EXP	Exponent
EXT	External
EXT TRA	External
REC	Transfer Receiver

	F
F	Function
FE	Floating Exponent
FE	Front End
FLG	Flag
FM	Fast Memory
FOV	Floating Overflow
FPD	First Part Done
FPD	Floating Point Divide
FUNC	Function
FXU	Floating Exponent Underflow
	G, H
G	Gated
GE	Greater or Equal
GEN	Generate
H	High
	I
INC	Increment
INH	Inhibit
INSTR	Instruction
INT	Internal
INTR	Interrupt
INVAL	Invalid
IOT	Input/Output Transfer
IR	Instruction Register
	J, K, L
J	Jump
L	Low
LRU	Least Recently Used
	M
MB	Memory Buffer
MBC	MBox Control
MBX	MBox Control
MBZ	MBox Control
MCL	Memory Control
MEM	Memory
MHz	Mega Hertz
MIX	Mixer
MQ	Multiplier Quotient
MQM	Multiplier Quotient Mixer
MR	Master
MTR	Meter
	N, O
NICOND	Next Instruction Condition
NXM	Non-Existent Memory
NXT	Next
OP	Operation (code)
OVN	Overrun
	P, Q
PA	Physical Address
PAG	Pager
PAR	Parity
PC	Program Counter
PCF_#	Previous Context Flags from Number
PCP	Previous Context Public
PC	Program Counter
PERF	Performance
PF	Page Fault
PGRF	Page Refill
PI	Priority Interrupt
PIA	Priority Interrupt Assignment
PIH	Priority Interrupt Hold
PMA	Physical Memory Address
PREV	Previous
PT	Page Table
PWR	Power
	R
RAM	Random Access Memory
RD	Read
RE	Receive ECL
REC	Receive
REF	Reference
REG	Register
REL	Release
REQ	Request
RESP	Response
RET	Return
RIP	Request in Progress
RQ	Request

S	
S ADR P	Storage Address Parity
SBR	Subroutine
SBUS	Storage Bus
SC	Shift Count
SCAD	Shift Count Adder
SCADA	Shift Count Adder A
SCADB	Shift Count Adder B
SCD	Shift Count Adder
SCM	Shift Count Mixer
SEL	Select
SH	Shifter
SHRT	Shift Right
SIM	Simulate
SP	Special
SPEC	Special
SR	State Register
SYNC	Synchronize

T	
T to E	TTL to ECL
TE	Transmit ECL
T	Time
TRA	Transfer
TTL	Transistor-Transistor Logic

U, V	
UBR	User Base Register
UCODE	Micro Code
VAL	Valid
VMA	Virtual Memory Address
XFER	Transfer
XR	Index Register

W, X, Y, Z	
WARN	Warning
WC	Word Count
WD	Word
WR	Write

Index

I, J, K

L

M

Reader's Comments

EBOX INSTRUCTION EXECUTION UNIT
UNIT DESCRIPTION
EK-EBOX-UD-003

Your comments and suggestions will help us in our continuous effort to improve the quality and usefulness of our publications.

What is your general reaction to this manual? In your judgment is it complete, accurate, well organized, well written, etc.? Is it easy to use? _____

_____

_____

_____

What features are most useful? _____

_____

_____

_____

What faults do you find with the manual? _____

_____

_____

_____

Does this manual satisfy the need you think it was intended to satisfy? _____

Does it satisfy *your* needs? _____ Why? _____

_____

_____

_____

Would you please indicate any factual errors you have found. _____

_____

_____

_____

Please describe your position. _____

Name _____ Organization _____

Street _____ Department _____

City _____ State _____ Zip or Country _____

CUT OUT ON DOTTED LINE

Fold Here

Do Not Tear - Fold Here and Staple

FIRST CLASS
PERMIT NO. 33
MAYNARD, MASS.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

Postage will be paid by:

Digital Equipment Corporation
Technical Documentation Department
146 Main Street
Maynard, Massachusetts 01754