The *Halt Handler* routine is entered from the Startup and Stop Interface when the RUN flip-flop is clear at NICOND Dispatch time. The RUN flip-flop can be cleared by various mechanisms. For example, when a HALT instruction is executed, RUN is disabled. On power up, RUN must be set by a diagnostic function initiated from the DTE20.

The *I/O Handler* (Figure 2-21) is dispatched via IR Dispatch from the Dispatch Table on DATAO, CONO after the data or status has already been fetched, or directly on DATAI, CONI, CONSO, or CONSZ. The handler calls the EBus driver, which generates the necessary EBus dialogue with the device. On BLKI or BLKO, the pointer has been fetched but must be updated, stored back at E, and the first word fetched. This is performed in the I/O Handler first. When the data has been fetched, the EBus driver is called. On DATAI or CONI, the EBus driver is called to negotiate the transfer from the selected device over the EBus to the EBox. The I/O Handler then passes control to the Data Store Manager where the data is stored.

2.3 BASIC MACHINE CYCLE

The basic machine cycle for a typical instruction is illustrated in Figures 2-22 and 2-23. The cycle begins at the EBox clock following NICOND Dispatch and terminates at the trailing edge of the next NICOND Dispatch. In this example, assume that the instruction MOVE 3 @ 200 (1) has been fetched from core memory symbolic location PC. The following information relates to the example:

PC/	MOVE 3 @ 200 (1)	Current Instruction
PC+1/	NEXT INSTRUCTION	
300/	000000, 000 100	Indirect Address = 300
100/	171717, 111111	Effective Address = 100
1/	000000, 000100	Index Register = 1

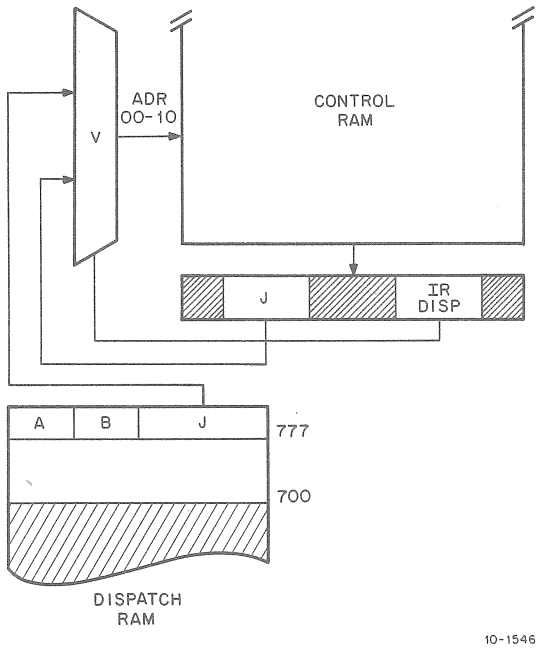
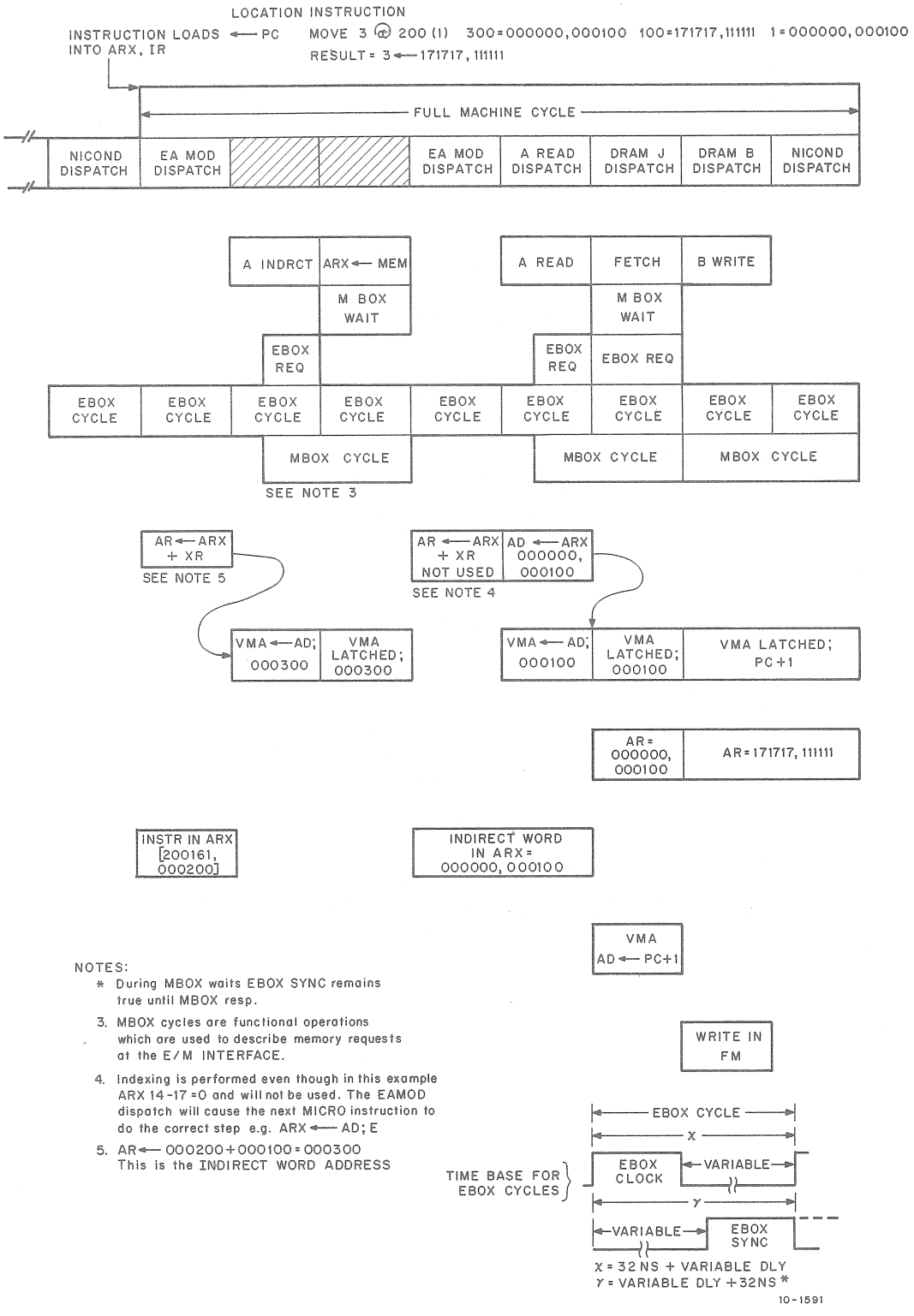


Figure 2-21 Input/Output Handler



NOTES:

* During MBOX waits EBOX SYNC remains true until MBOX resp.

3. MBOX cycles are functional operations which are used to describe memory requests at the E/M INTERFACE.
4. Indexing is performed even though in this example ARX 14-17 = 0 and will not be used. The EAMOD dispatch will cause the next MICRO instruction to do the correct step e.g. ARX ← AD; E
5. AR ← 000200 + 000100 = 000300
   This is the INDIRECT WORD ADDRESS

Figure 2-22 Basic Machine Cycle Overview (Sheet 1 of 2)

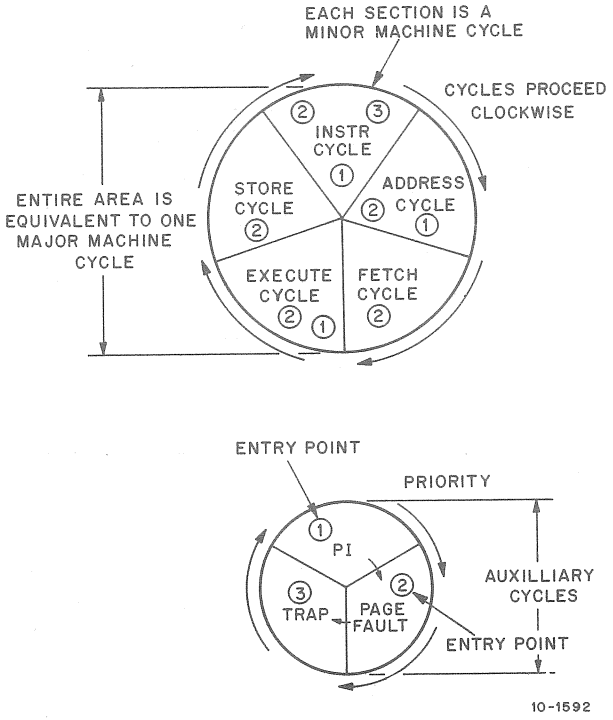


Figure 2-22 Basic Machine Cycle Overview (Sheet 2 of 2)

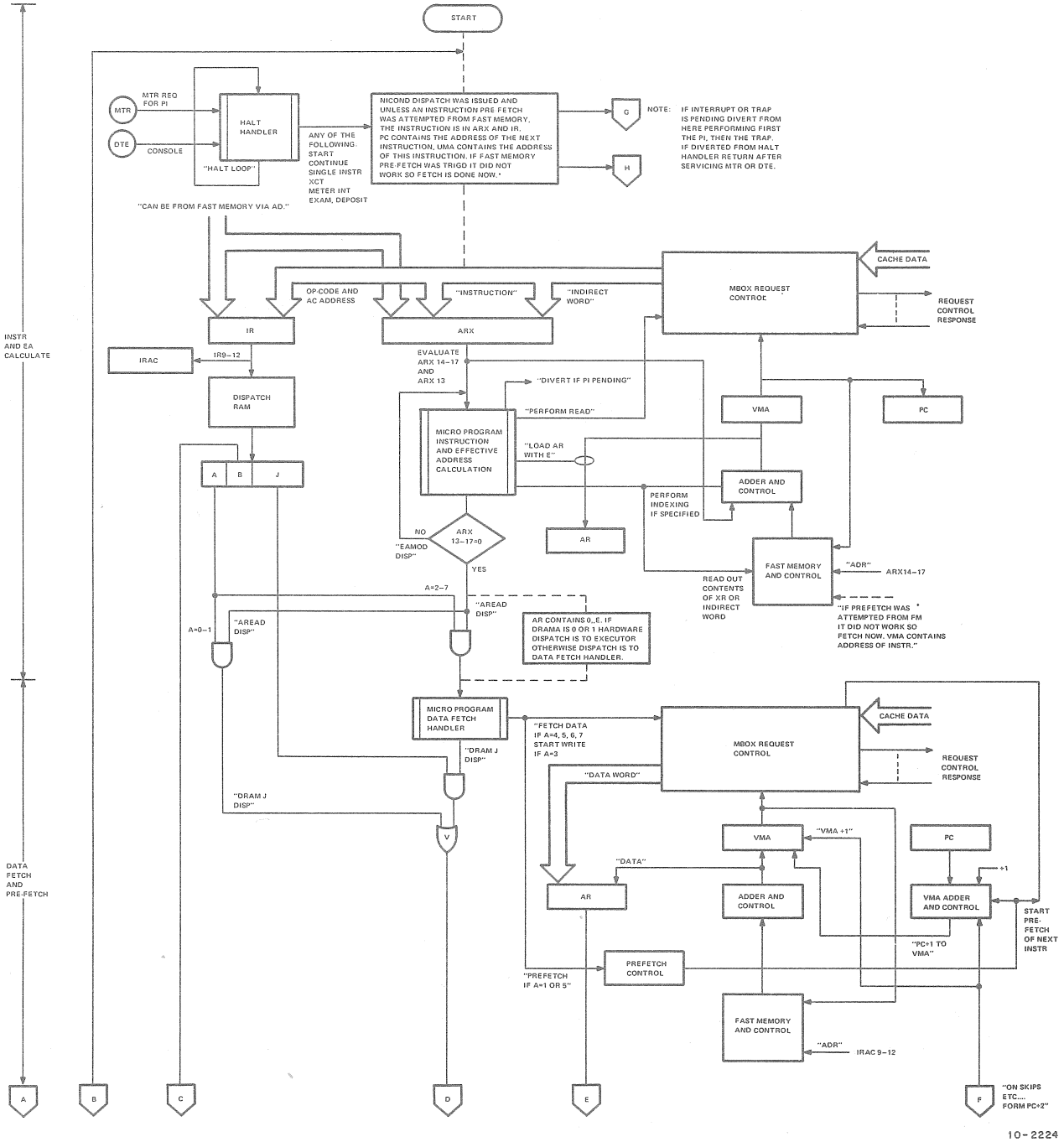


Figure 2-23 KL10 Processor Sequence of Operation (Sheet 1 of 2)

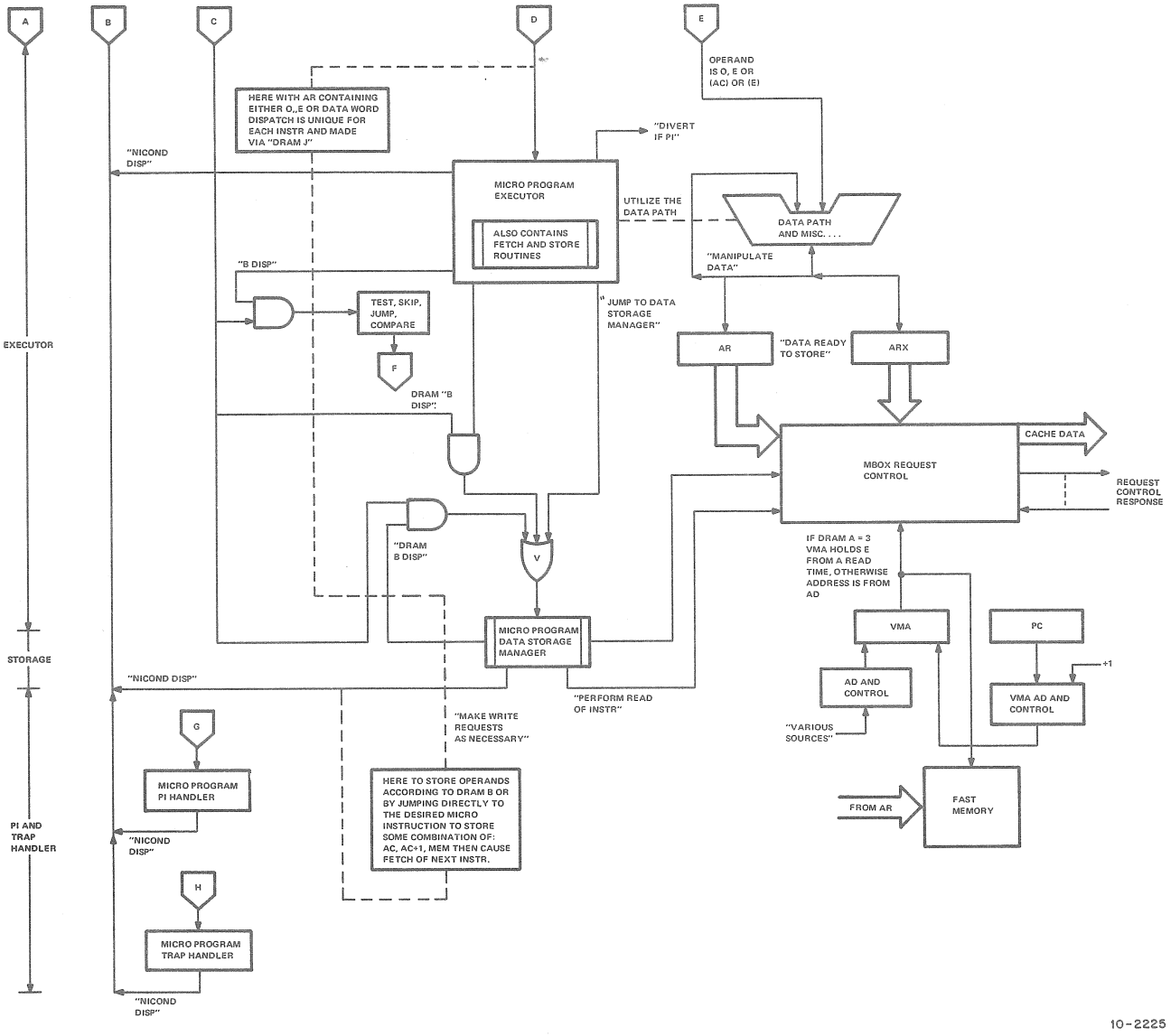
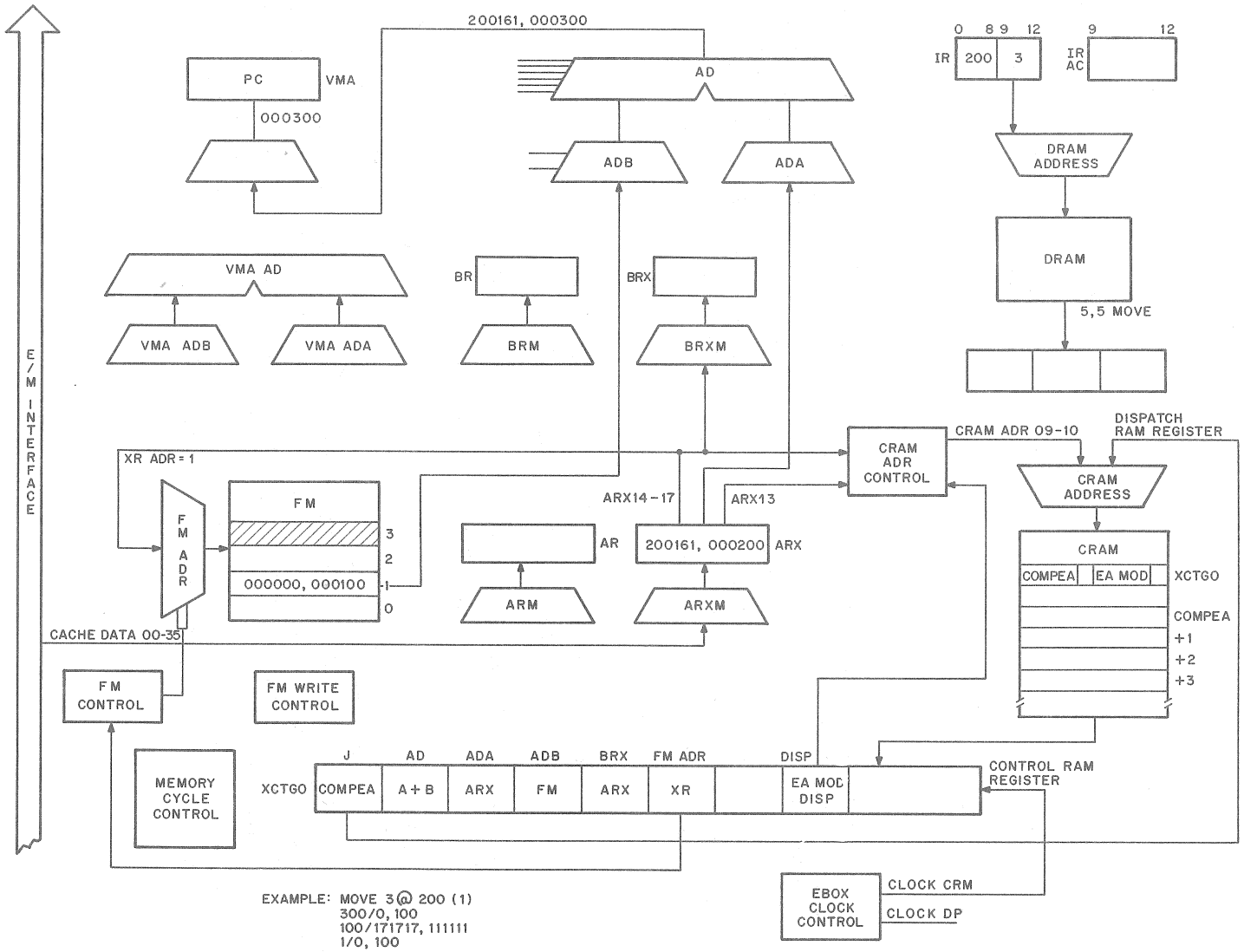


Figure 2-23 KL10 Processor Sequence of Operation (Sheet 2 of 2)

Figures 2-24 through 2-33 illustrate the microprogram steps and basic EBox hardware used to perform the example instruction. Figure 2-22 can be used to follow the various operations at each micro-instruction step.

2.3.1 Instruction Cycle – NICOND Dispatch to XCTGO

The instruction enters the ARX through the ARX mixer (ARXM) via the cache data lines. Although not shown, the MBox response enables the mixer selection and the EBox clock (CLK DP) loads the ARX on the Data Path Board with the instruction. The NICOND Dispatch for this example is to symbolic location XCTGO; Figure 2-24 indicates the major microinstruction fields. The Jump address contains the base address of a 4-word block used to calculate the effective address. Each micro-instruction in this block is used for a different form of address calculation, and is selected based upon the state of ARX14–17 and ARX13 when EA MOD DISPATCH is given. The EBox hardware utilizes ARX14–17 and ARX13 to modify bits 09–10 of the CRAM address. This yields the possibilities listed in Table 2-4.



NOTE:
This operation reflects the micro code version 71.

Figure 2-24 Instruction Cycle: NICOND Dispatch → XCTGO

Table 2-4 Address Calculation

CRAM Address	ARX14–17	ARX13	Function
COMPEA	0	0	ARX = E
COMPEA+1	Nonzero	0	Perform indexing as specified by ARX14–17.
COMPEA+2	0	1	Perform indirection VMA ← ARX18–35
COMPEA+3	Nonzero	1	Perform indexing as specified by ARX14–17, then perform indirection VMA ← ARX18–35 + (XR)

While at XCTGO, to speed things up, the indexing operation is started. The fast memory address field in the microinstruction causes the FM control to address fast memory utilizing ARX14–17, which in the example is 1. The ADA input is enabled to select the ARX as input to the ADDER A input. This is controlled by the microinstruction ADA field. Similarly, the ADB field enables the ADB input to select addressed FM location 1. The microinstruction AD field specifies the ADDER function as A+B. Thus, the ADDER begins to add the contents of location 1 in fast memory to the instruction in ARX. At this time, the Buffer register extension is enabled from ARX by the microinstruction BRX field.

NOTE

The IR contains the op code of the instruction MOVE, which is 200, and the AC field, which is 3.

The op code value (200) is used to address the DRAM to obtain the appropriate word for this instruction. This word is indicated on the input to the DRAM register (5,5,MOVE).

2.3.2 Indirect Word Request

For an Indirect Word request, the CRAM register contains the microinstruction fetched from COMPEA+3 as indicated in Figure 2-25. The Jump address now specifies a direct jump to symbolic location INDRCT. The AD, ADA, ADB, and FMADR fields are maintaining the indexing calculation and the calculated address 000300 is forming at the input to the VMA. The MEM microinstruction field is coded as A IND. This enables the memory cycle control to set up and generate an MBox cycle (Figure 2-26). This begins with the assertion of EBOX REQUEST IN, together with the qualifier EBOX READ. Table 2-5 lists the MEM field function that generates requests. An IND is a function that may be followed by a microinstruction having the MEM field coded as MB WAIT.

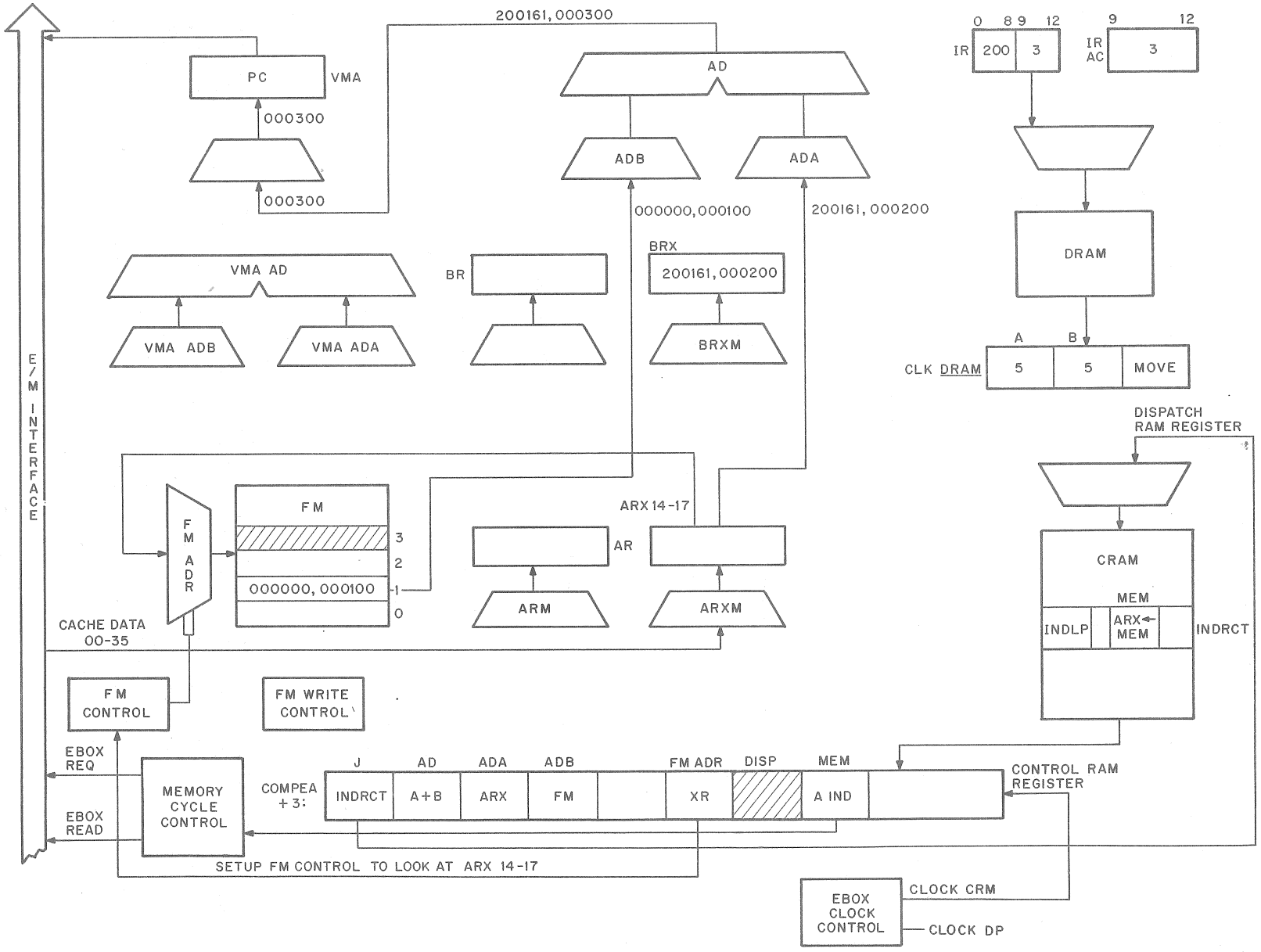


Figure 2-25 Set Up and Make Indirect Work Request

Table 2-5 MBox Cycle Requests

MEM 02	MEM Field	MEM 00	Function	Causes	MBox Wait
0	04	0	A READ	Fetch Cycle	No
0	05	0	B WRITE	Store Cycle	No
1	06	0	FETCH	Instruction Fetch	Yes
1	07	0	REG FUNC	MBox register reference	Yes
0	10	1	A IND	Indirect reference during effective address calculation	No
0	11	1	BYTE IND	Indirect reference for byte instruction special	No
1	12	1	LOAD AR	Data read during execution, loads into AR	Yes
1	13	1	LOAD ARX	Data read during execution, loads into ARX	Yes
0	14	1	AD FUNC	Not used	No
0	15	1	BYTE RD	Data read during byte execution loads into AR and ARX	No
1	16	1	WRITE	Store data during execution, writes from AR	Yes
1	17	1	RPW	Initiates a read PSE write cycle, data loads into AR	Yes

The time field for the microinstruction at location COMPEA+3 specifies a period between the EBox clock that loaded the microinstruction from COMPEA+3 and the next EBox clock. It allows sufficient time for the access of fast memory to be completed. Note that EBox request and EBox sync are concurrent (Figure 2-26). The earliest time that the MBox can clear the request is on the MBox clock following EBox sync. In Figure 2-26, EBox sync occurs one MBox clock prior to where the time field indicates EBox clock can occur, but because MBox wait is true and the MBox has not yet responded, the EBox clock is postponed as indicated.

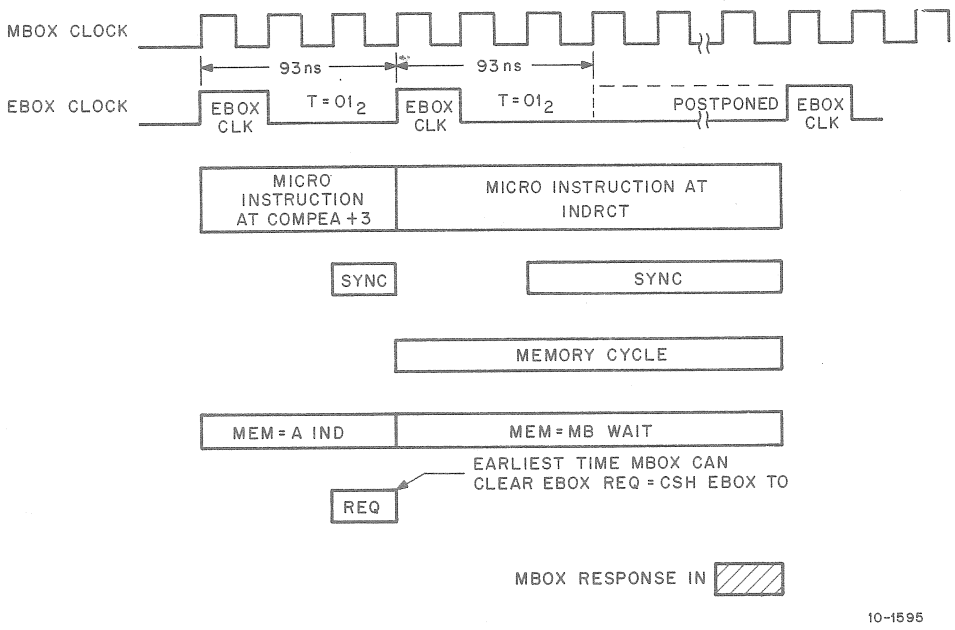


Figure 2-26 MBox Cycle

2.3.3 MBox Response to Indirect Word Request

Figure 2-27 illustrates the microinstruction fetched from symbolic location INDRCT. Again, a direct Jump is specified (in this instance, to INDLP). A response from the MBox is anticipated. ARX ← MEM is a MACRO statement. It specifies MEM to be MB WAIT and also selects FM as addressed by VMA 32–35. The ARXM is actually input from both AD on the 2 input and the cache data on the 1 input. The MBox response causes the EBox hardware to generate MB XFER, which selects the correct input. In this example, the cache data lines containing the indirect word 000000,000100 are loaded into ARX.

2.3.4 Address Calculation Continues

Referring to Figure 2-28, the CRAM register contains the microinstruction fetched from symbolic location INDLP. This setup is once again to perform indexing as though it were really specified. At this time, ARX contains indirect word 000000,000100; ARX14–17 and ARX13 are zero. Thus, even though the microinstruction specifies the calculation of indexing, the hardware calculates the proper CRAM address based upon ARX14–17 = 0 and ARX13 = 0.

The basic jump address is COMPEA and this is the next CRAM address. The dispatch is EAMOD and, on the next EBox clock, the microinstruction from COMPEA is fetched. Note, too, that the DRAM register is latched and contains the A, B, and Executor Jump address.

2.3.5 A READ Dispatch – Set Up Data Fetch and Prefetch

Refer to Figure 2-29. Once the effective address has been calculated, what has been traditionally called the Fetch cycle follows. The CRAM register contains the microinstruction fetched from COMPEA. The J field is zero in this case. The EBox hardware, upon detecting a Read Dispatch, inspects the dispatch A field and forces the CRAM address to 40 + A. Thus, in this example, the address becomes 45. Address 40 + A is defined by hardware. The effective address in ARX18–35 is enabled into the ADDER A input by the AD field coded as A, with ADA selecting ARX. To begin the data fetch, the MEM field is coded as A READ and this, with the A field, generates EBOX REQUEST and EBOX READ. On the next EBox clock, the effective address is loaded into AR.

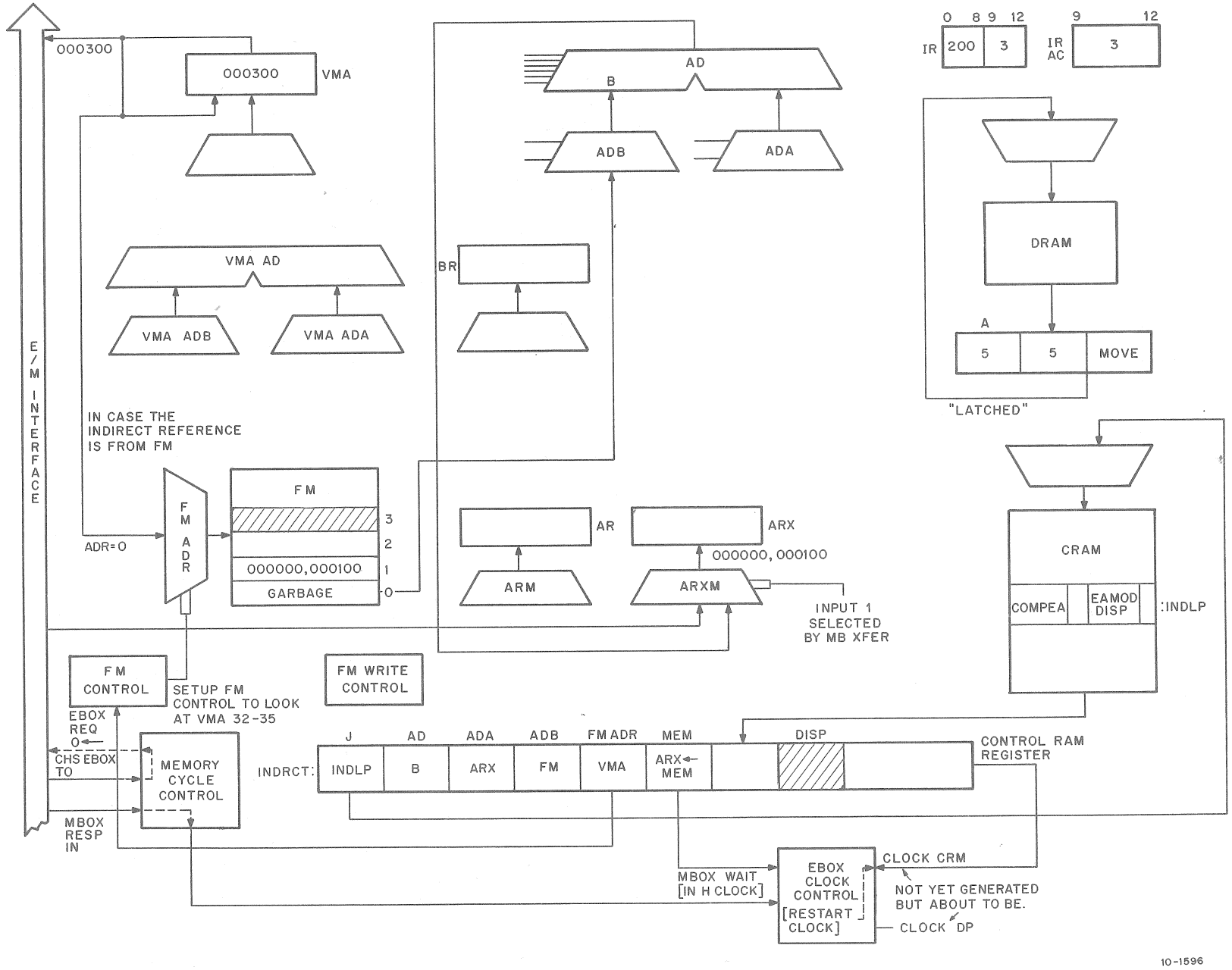


Figure 2-27 MBox Response to Indirect Request

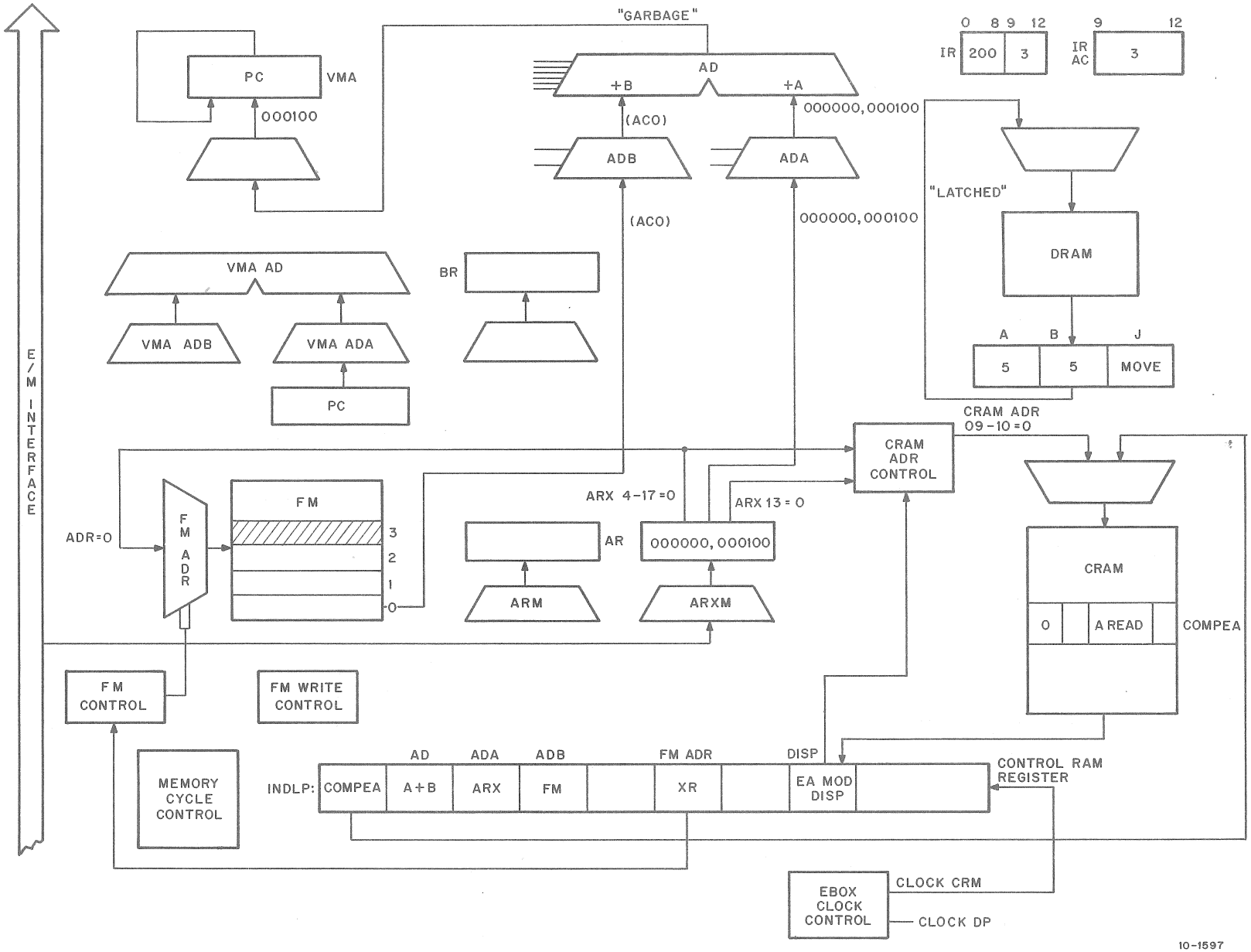


Figure 2-28 Address Calculation Continues

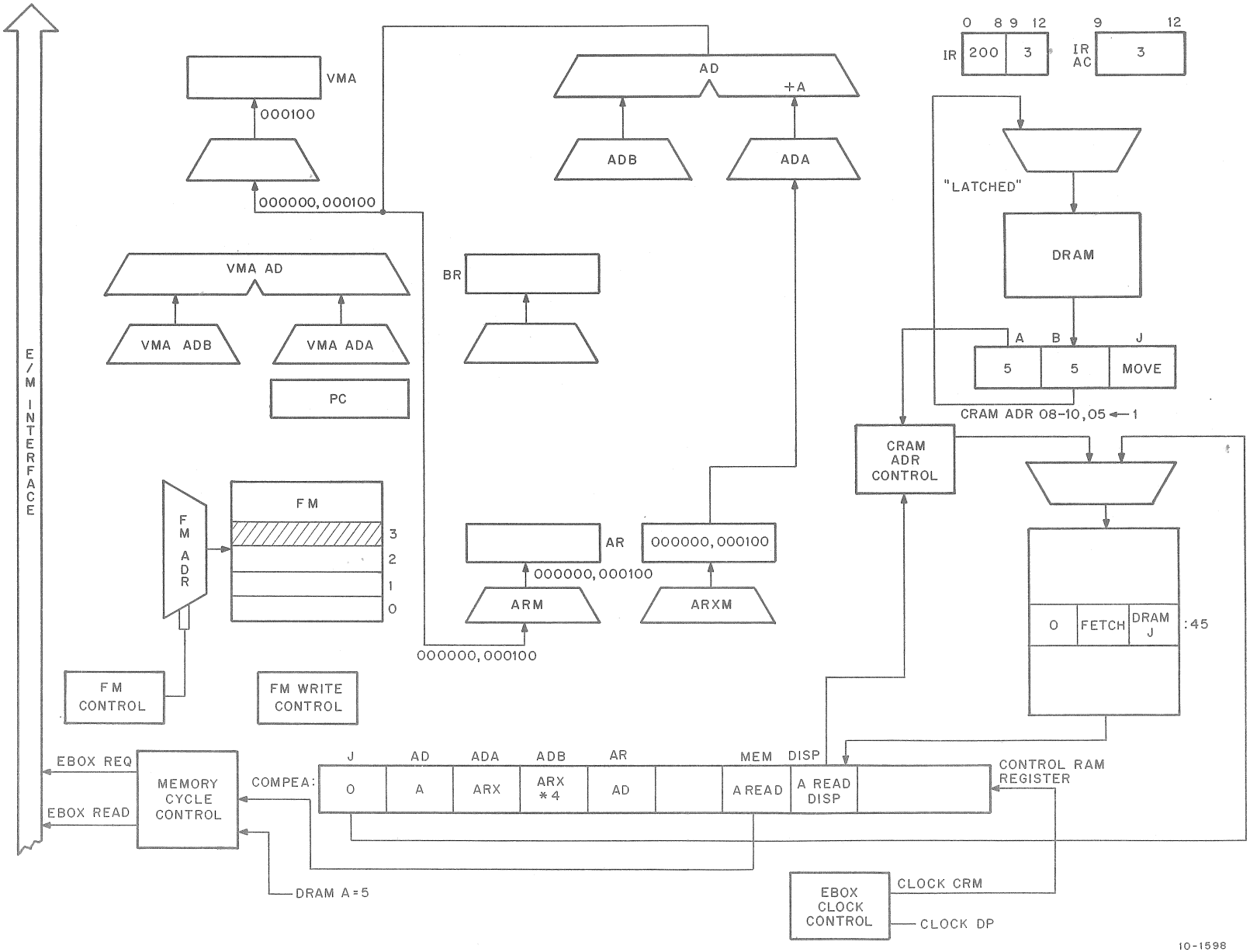


Figure 2-29 AREAD Dispatch Setup Data Fetch

2.3.6 MBox Response to Data Read – Prefetch Begins

Figure 2-30 illustrates the CRAM register containing the microinstruction from location 45. The jump address once again is zero, because the actual jump address is provided by the DRAM register jump field. In the case of MOVE, the symbolic address is "MOVE." This location contains the first microinstruction in the executor for the MOVE instruction. Only one microinstruction is required for the execution of the basic MOVE. This dispatch field contains DRAM J, enabling the CRAM address control to utilize the jump address in the dispatch register. Thus, for the basic MOVE, symbolic location "MOVE" contains the desired microinstruction. The MEM field is coded as fetch to enable the memory cycle control to begin the prefetch by asserting EBox request with EBOX READ.

Until the MBox response to the data read is received, the VMA is latched and only the VMA input contains the updated PC value. When the MBox response is received, the VMA is loaded with the updated PC value (PC+1). At the same EBox clock, the data on the cache data lines is clocked into AR (000100). Referring to Figures 2-30 and 2-31, the FMADR field enables FM to be addressed via VMA 32–35, even though in this example VMA address 000100 is not an FM address. FM location 0 is actually accessed and enabled via ADDER B into the AR mixer.

The Memory Cycle Control asserts LOAD AR. The address in VMA is checked in the VMA Control and, because it is not a fast memory address, –VMA AC REF is asserted. This is passed to EBox Control No. 1 logic and inhibits the generation of FM XFER.

MBox RESPONSE IN is passed to the EBox clock control where it becomes (on the next MBox clock) RESPONSE MBox. This, with LOAD AR, enables the selection of ARM SEL 1, which enables the cache data into AR. The EBox clock then strobes the AR register. This clock also clocks the next microinstruction from symbolic location MOVE into the CRAM register.

2.3.7 Executor – Set Up for Store Cycle

For the basic MOVE instruction, the data word in AR must be stored in the FM location specified in the AC field of the currently executing instruction. The microinstruction J field contains the base address for the data storage microprogram. This is symbolic location ST0. The Dispatch field is coded as DISP B, which enables the B field of the DRAM register to modify the low-order three CRAM address bits (CRAM 08–10). The B field is 5 for MOVE and this yields symbolic location STAC. If, for example, ST0 was physically 60, the resulting address would be generated by logically ORing 60 with 5 for a result of 65, symbolically STAC.

Referring to Figure 2-32, IRAC contains AC address 3, and is enabled to address FM because the microinstruction FM ADR field is coded as AC0. This is the AC specified by AC 09–12. The MEM field specifies B WRITE, but no request is issued. This is because the memory cycle control samples the DRAM B field and inhibits an EBox request when DRAM B01 is a zero.

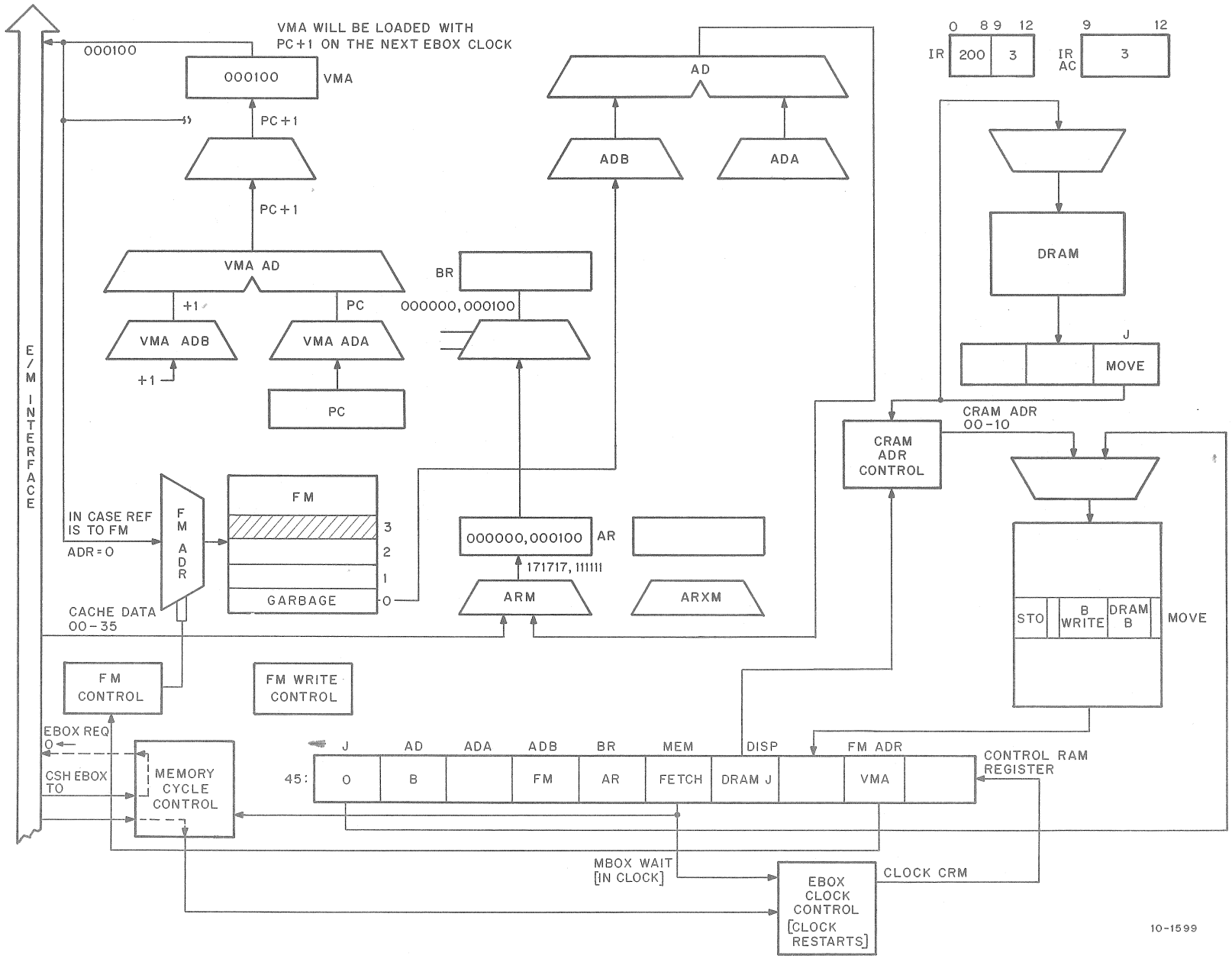


Figure 2-30 MBox Response with Data Word Requested

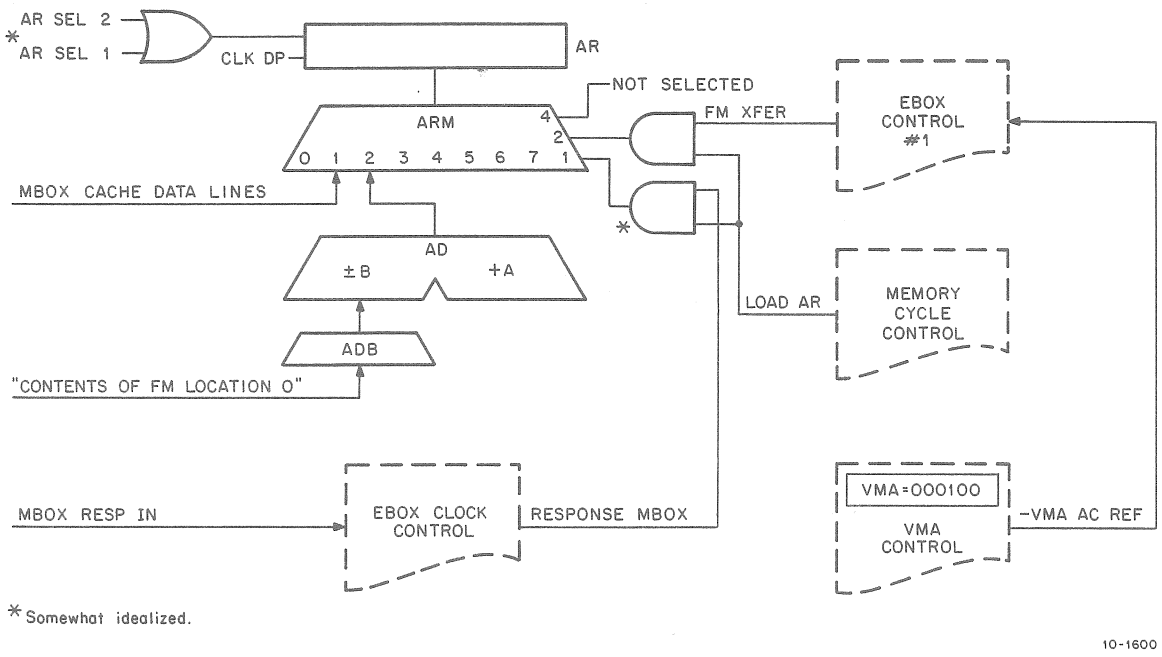


Figure 2-31 Hardware Selection of ARM Data

2.3.8 Finish Store Cycle – Perform NICOND Dispatch

The CRAM register now contains the microinstruction from symbolic location STACK (Figure 2-33). The J field specifies the base address NEXT and the Dispatch field contains NICOND Dispatch. This completes the basic machine cycle by reentering the instruction cycle once again.

The FM ADR field maintains the FM address via IRAC and the COND field is coded as FM WRITE to write the contents of AR into FM location 3. The MEM field is coded as MB WAIT for the cases where the next instruction has been prefetched from memory. This forces the EBox to wait until the instruction enters the ARXM and MBOX RESPONSE is received. If the instruction is being fetched from fast memory, MB WAIT has no effect and the microprogram selects the appropriate micro-instruction to load ARX from fast memory as addressed by VMA 32–35.

2.4 PAGE FAIL CYCLE INTRODUCTION

Normally, primary memory is the MBox cache memory, secondary memory is core memory, and the auxiliary memory is a disk or drum. Information is moved into the core only on demand (Demand Paging), i.e., no attempt is made to move a page into core memory, and consequently words into the cache, until some program references it. Information is returned to core memory in accordance with a hardware algorithm in the MBox hardware. Information is returned from core memory to auxiliary storage at the discretion of the operating system's paging algorithm. Information movement across the gap bridging the level between auxiliary storage and core memory-cache memory is called page traffic.

The MBox, in a sense, is an interface between the EBox (processor) and the SBus. It provides individual mapping (relocation) of each page (512 words) of both user and monitor address spaces, using separate maps for each. The MBox uses hardware storage to access and load the mapping information.

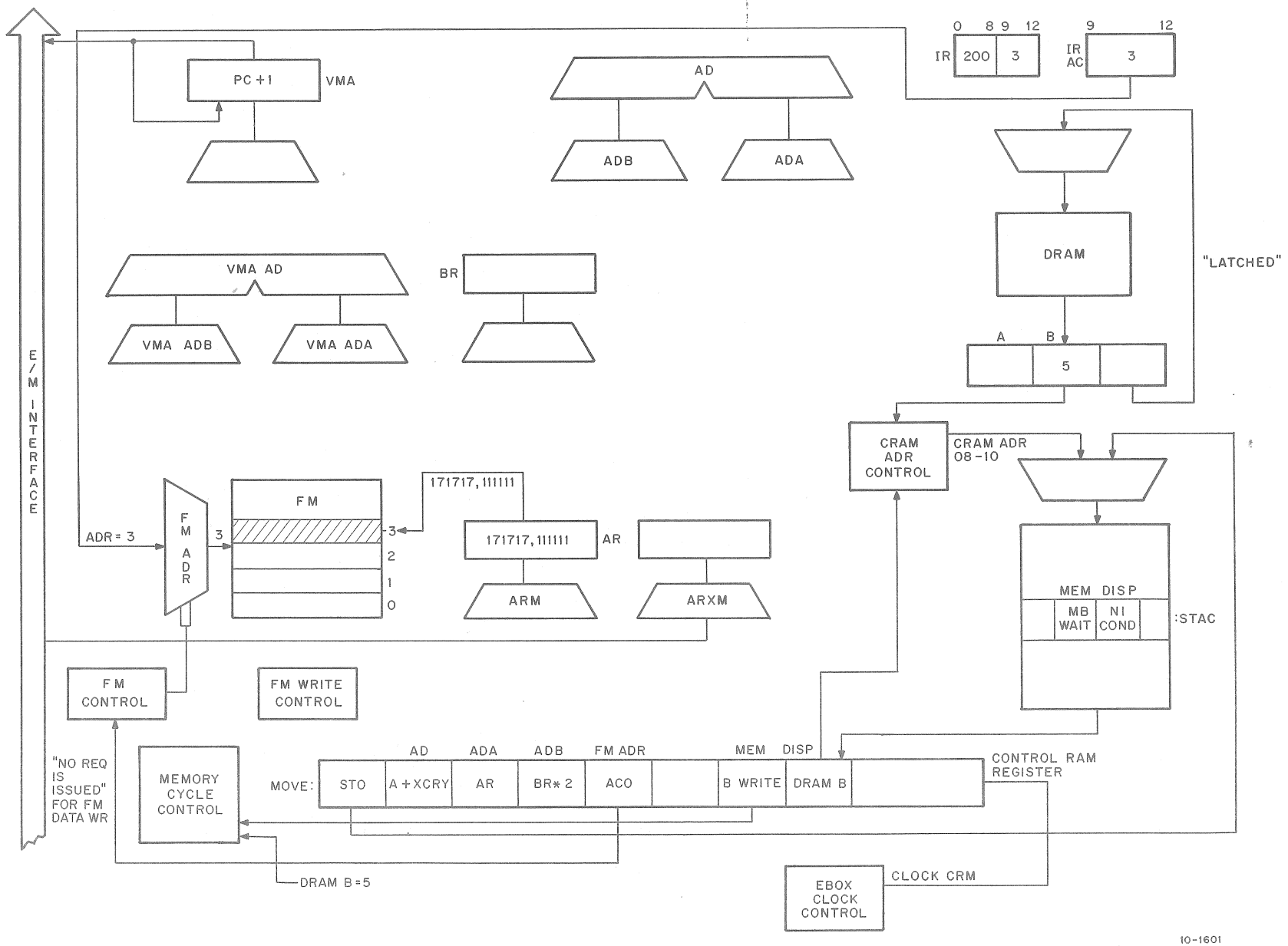


Figure 2-32 Executor Setup for Store Cycle

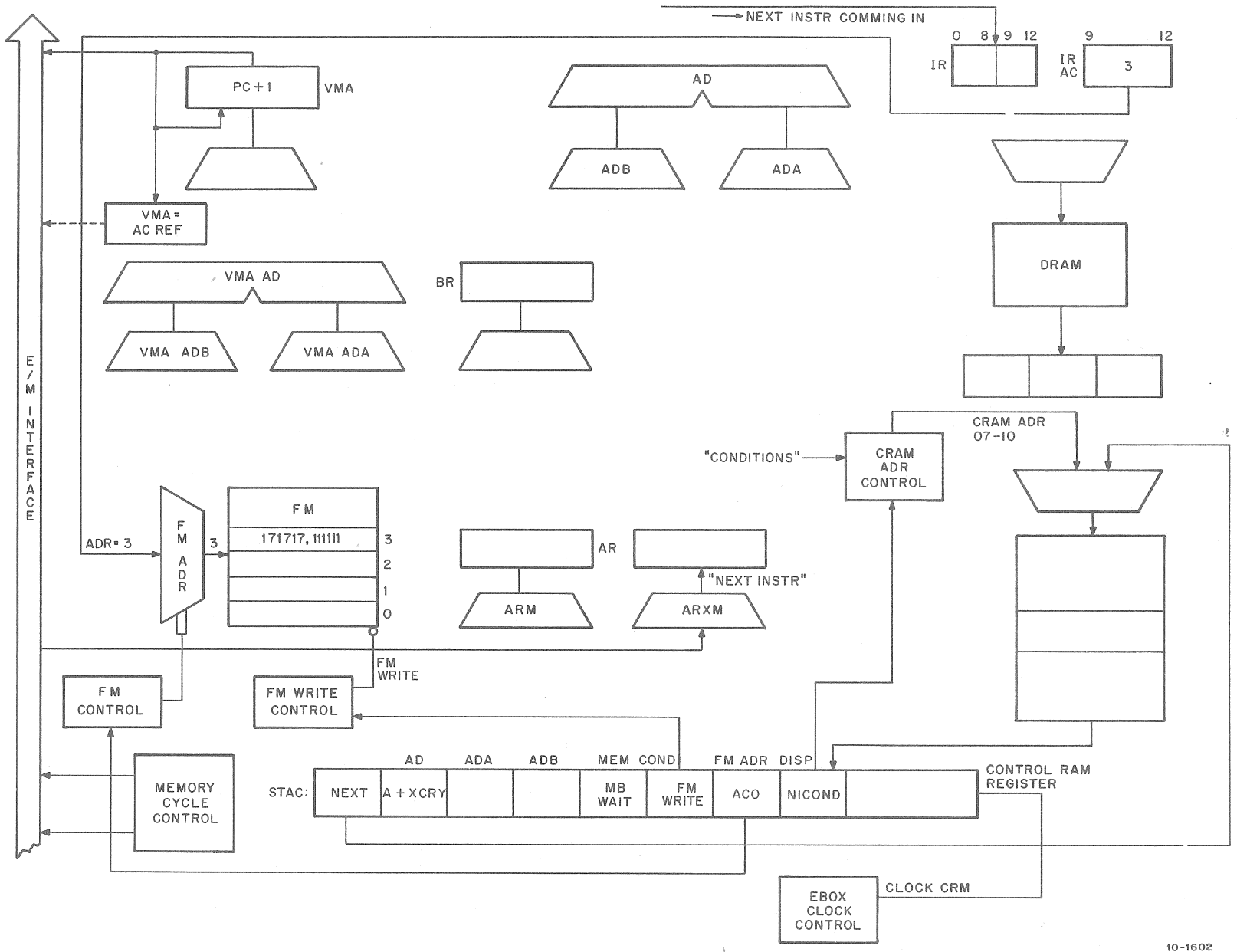


Figure 2-33 Finish Store Cycle, Perform NICOND Dispatch

It also contains a 2048 word cache for holding the data for the mapped references. On each memory request from the EBox, the nine high-order bits of the virtual address and the type of request (read, write) are compared with the contents of the hardware tables in the MBox. If a match is found, the location containing the match also contains 13 high-order address bits to reference the physical page in the cache. If no match is found, a 512-word "Page Table" in physical core memory is referenced. The word selected in this page table is determined by a dispatch based on the original nine high-order address bits. The 13 high-order address bits and use bits found in this word are written into the MBox hardware table; the use bits are checked against the type of EBox reference. Four possible cases exist concerning the disposition of the use bits:

1. The page is not in core.
2. The page is protected from the type of request.
3. The page is nonexistent.
4. The page is in core and is compatible with the type of request.

For the first three cases, a page fault (trap) occurs; for the fourth case, the requested word is fetched from core memory (actually words are fetched four at a time, differing only in the two least significant address bits) and written into the cache. Concern here is with the page fault situations. The MBox constructs a page fault word in one of its internal hardware registers, the EBus register. The word contains information relating to the type of fault that occurred. The EBox is waiting for an MBox response to its request; the MBox, therefore, asserts PF HOLD, and some time later asserts MBOX RESPONSE IN. When the EBox recognizes the PF HOLD signal, it forces the CRAM address to 1777. This is the first microinstruction in the micropage fault handler. The EBox does not issue an EBox clock until the CRAM address has had time to set up. Once the address is stable, a single EBox clock is issued to the CRAM board to access the microinstruction.

2.4.1 Page Fail Handling – Functional Flow

Figure 2-34 is a functional flow of the microprogram page fault handler. The EBox contains a 4-bit state register. This register, during certain instructions, holds a number that may be used to modify the state of the CRAM address. For instructions that do not use the State register, it contains zero. Generally, the STRING, EDIT, and BLT instructions require cleanup following a page fault so that they may be properly terminated. For these cases, the State register contains a value in the range of 1–7. The more general case is discussed here; this is where the State register contains zero. For both cases, INSTR ABORT (coded in the condition field of the microinstruction fetched from CRAM address 1777) performs the following functions:

TRAP REQ 1 ← TRAP CYCLE 1
TRAP REQ 2 ₂ TRAP CYCLE 2
ADR BRK INH ← ADR BRK CYCLE

These actions are necessary to assure that the PC flags reflect the state of the EBox when a page fault occurs during the fetch of the trap instruction, during its execution, or during an address break page fault. A State register dispatch is given, but because the State register is clear, the base address is used to obtain the next microinstruction. A priority interrupt has a higher priority than a page fault (Figure 2-35); therefore, a pending interrupt is checked for first. If INT REQUEST is true, the PI Handler is entered to service the interrupt. If no interrupts are pending, the page fault is handled. The third level of priority is given to traps and finally to all other events being processed by the microprogram.

A page fault occuring in response to an API interrupt function is a fatal error. Thus, when the page fault handler finds PI CYCLE set, it sets the I/O Page Failure flag, dismisses the failing interrupt, and then, if possible, restores the EBox to the state it was in prior to the interrupt. The setting of IOPF eventually causes an interrupt on the APPR error channel. The PF Handler now attempts an instruction fetch.

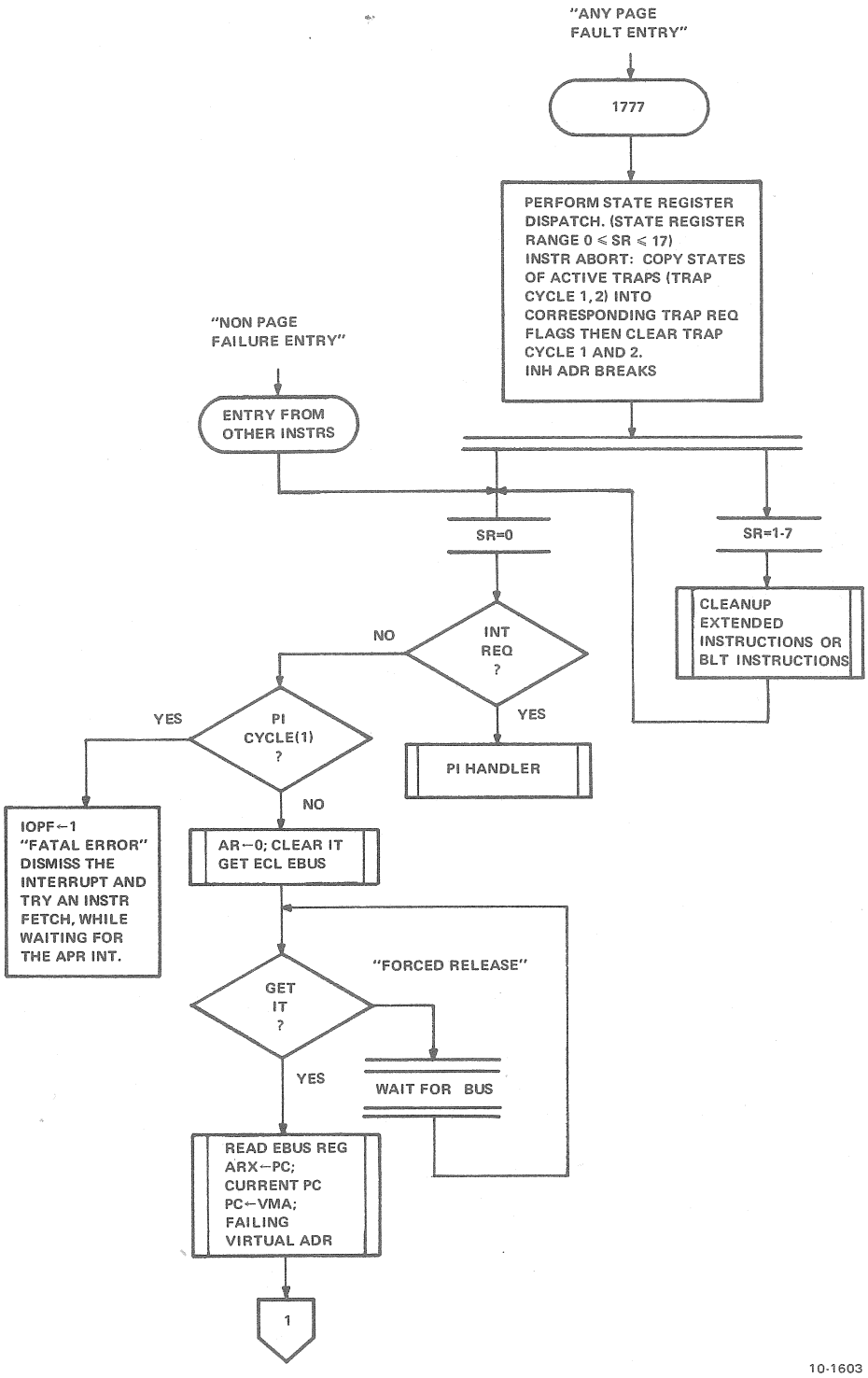


Figure 2-34 Page Fail Handling (Sheet 1 of 2)

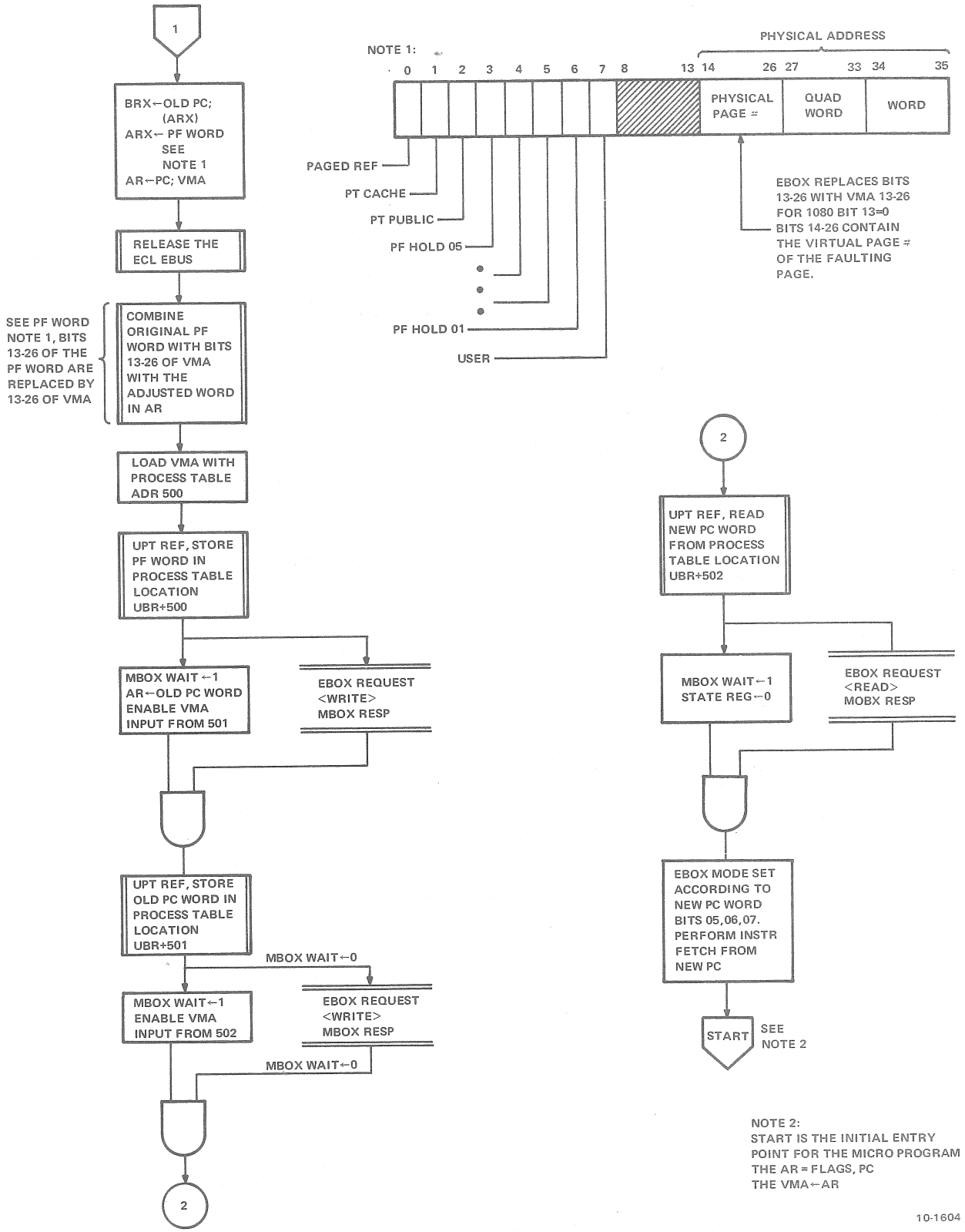


Figure 2-34 Page Fail Handling (Sheet 2 of 2)

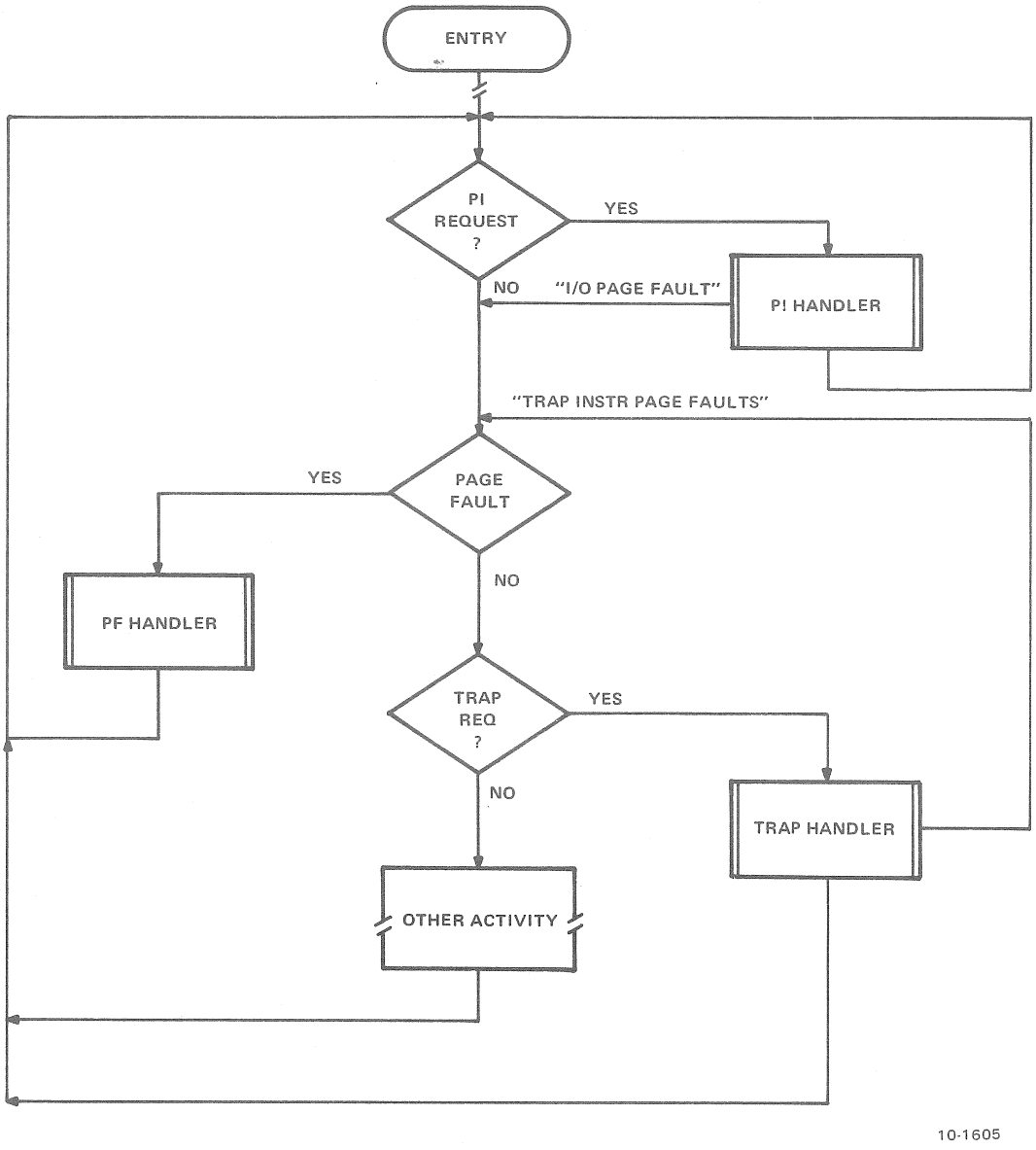


Figure 2-35 EBox Priorities

Obtaining and Adjusting the PF Word – Assuming PI CYCLE is clear, the AR is cleared and the ECL EBus is requested. This is to transfer the PF word from the MBox EBus register to the AR register in the EBox via the EBus. Because the PI system and external or internal devices can also use the EBus, the microprogram must force its release. When the ECL side is obtained, the EBox reads the PF word into AR. The PF word, as it is constructed by the MBox, contains the physical page number in bits 14–26. The EBox must replace this with the virtual address and also clear bit 13. The current virtual PC is temporarily placed into ARX; the failing VMA is placed into AR while the old PC is saved in BRX. The ECL EBus is then released. The ARX and AR are shifted to adjust bits 13–26 to be the VMA 13–26.

Figure 2-36 shows the three locations in the user process table dedicated to page fault handling.

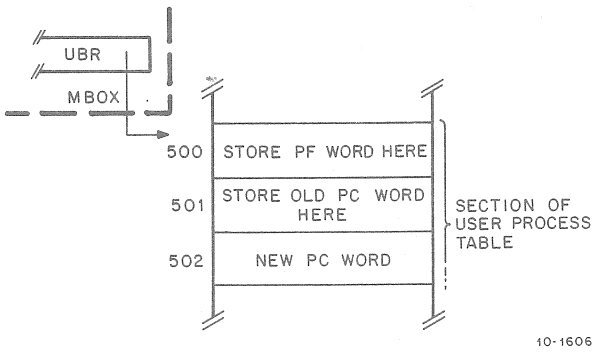


Figure 2-36 Process Table PF Location

2.4.2 Process Table References

The VMA is loaded with low-order process table location 500 and an EBox request is issued to write the PF word (concurrently in AR) into process table location UBR+500. The next microinstruction is loaded and EBox clock sets MEM CYCLE, causing MBOX WAIT. The AR is enabled from the old PC word; the input to VMA is now 501. As soon as the MBox responds, MBOX WAIT is removed and the cycle is repeated. This time the EBox request is to write the old PC word (now in AR) into process table location UBR+501. Once again, the next microinstruction is loaded and EBox clock sets MEM CYCLE, causing MBOX WAIT. The VMA input is now 502. As soon as the MBox responds, MBOX WAIT is removed and the cycle repeats, in this instance for reading a new PC word from process table location UBR+502. The new PC word places the EBox in a specified mode and the first instruction is fetched from the appropriate handler. This completes the page fault cycle.

2.5 TRAP CYCLE – INTRODUCTION

A Trap is produced by setting either of two trap request flags in the EBox (TRAP REQ1 or TRAP REQ2). The programmer knows these flags as TRAP2 and TRAP1. The conditions that set TRAP REQ1 are equivalent to the arithmetic overflow conditions that set SCD OV. TRAP REQ2 is set by the various pushdown overflow conditions: the left half of the pointer is counted down to -1 (no carry out of bit 0) in a POPX, or is counted up to zero in a PUSHX. (The condition for this is the presence of a carry out of bit 0, but the condition is detected by the microprogram and the trap request flag is set.)

2.5.1 Trap Handling

The Trap Handler (Figure 2-37) is entered at NICOND Dispatch time providing its priority is highest of the major priority events. The microprocessor NICOND Dispatch, together with four queues arranged in a round robin priority structure, is shown in Figure 2-38. The TRAP request is served only when no priority interrupt requests are pending and no page fault is pending. It does, however, pre-empt the normal instruction cycle. Both the user and exec process tables contain dedicated locations for processing traps. These locations are XXX 421 for arithmetic overflow (TRAP1), XXX 422 for pushdown overflow (TRAP2), and XXX 423 for the programmed trap (TRAP3). XXX is replaced by the appropriate base register (UBR or EBR), which resides in the MBox. The base register used by the MBox is determined by the state of the qualifiers sent during the EBox request. The MBox fetches the appropriate trap instruction and places it on the cache data lines while issuing MBOX RESPONSE IN. The EBox then executes the trap instruction. It is possible for the EBox request for the trap instruction to cause a page fault. If this occurs, the page fault handler is entered at CRAM address 1777 and the trap cycle flags are pushed into the trap request flags so that the trap flags may be saved; the trap cycle properly reenters at a later time.