With the special field function SCM ALT and SC field equal to zero, FE is selected. Similarly, with SCM ALT and SC field equal to one, AR SHIFT is selected. AR SHIFT consists of bits 18 and 28–35 of AR, which are derived from the effective address for shift instructions. If bit 18 is set, the shift specified is a right shift; otherwise, it is a left shift.

2.9.5.13 SH Field – The SHIFTER field consists of two bits and is used to select four possible inputs to the shifter. The selection is as follows: the combined AR, ARX(0), AR(1), ARX(2), and AR SWAPPED(3). When shifting AR, ARX left (which is the only way SH shifts physically), SC can specify up to 35_{10} shifts. Any number less than 0 or greater than 35_{10} selects ARX as output.

2.9.5.14 The AR Mixer Mixer (ARMM) – The AR Mixer Mixer (ARMM) field consists of two bits and is used with other control signals and the absence of ARM SEL 4, 2, and 1 to select various sources as input to AR mixer.

The ARMM comprises three parts: bits 00–08, bit 12, and bits 13–17. The same field that controls SH controls ARMM00–08. The following may be selected as input to ARMM00–08: #(0), AR SIGN SMEAR(1), SCAD EXP(2), and SCAD POS(3). AR SIGN SMEAR is AR0–8 from AR0. SCAD EXP is AR0–8 via SCAD, and SCAD POS is AR0–5 via SCAD.

ARMM bit 12 is controlled by CRAM SH-ARMM SEL 1 when transferring the previous section to AR for certain operations. ARMM bits 13–17 are also under control of CRAM SH-ARMM SEL 1 but the signal is actually MCL PREV SECT to ARMM. The default value for ARMM 13–17 is PC 13–17 and the selected value is VMA previous section 13–17.

2.9.5.15 VMA Field – The VMA field consists of two bits and is used to select various sources as input to VMA. The following are specified by the CRAM field VMA(0), PC(1), PC+1(2), and AD(3). Address control is presented in Subsection 2.4 and a path diagram is provided to show various combinations in Figure 2-58.

2.9.5.16 MQ Field – The MQ field consists of one bit and is used in combination with the following:

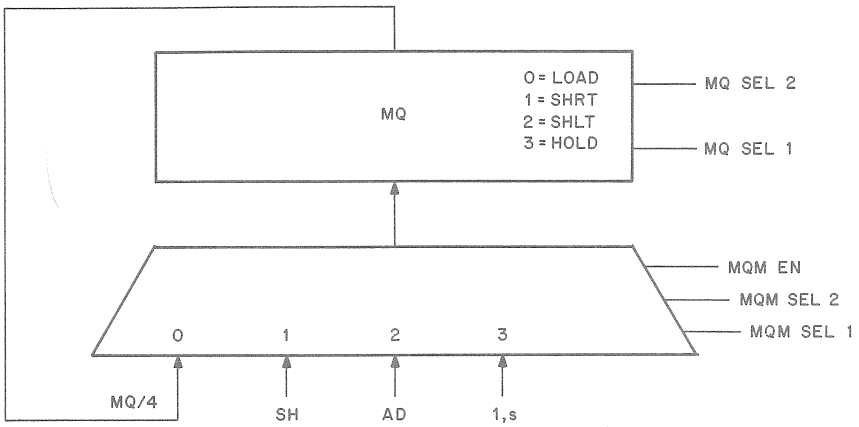
DISP/MUL
DISP/DIV
SPEC/MQ SHIFT
SPEC/REG CONTROL
MAGIC NUMBER FIELD

Refer to Figure 2-73 for various combinations.

2.10 EBOX INSTRUCTION SET FUNCTIONAL OVERVIEW

Figure 2-74 breaks down the KL10 instruction set into several functional areas. These areas are related to the minor machine cycles and to the microcode dispatch RAM decoding. The figure shows seven basic areas as follows:

1.	Group	Class of instruction
2.	Address Calculation	xr, @, B, Y
3.	Data Fetch	IMM, Read, Read-Write, Write, Read, Pse Write
4.	Execution	36-Bit Data Path (DP), 18-Bit Address Path (AP), 23-Bit AP, 10-Bit AP
5.	Special Conditions	Can cause PI, Trap
6.	Store Data	Write
7.	Interruptable	



MQM Out	MQM EN	MQM Sel 2	MQM Sel 1
MQ/4	1	0	0
SH	1	0	1
AD	1	1	0
1's	1	1	1

MQ←	MQ Sel 2	MQ Sel 1
MQM	0	0
MQM/2	0	1
MQM*2	1	0
Hold	1	1

CRAM MQ Field	SELECTED CONTROL SIGNALS					CONTROLLING FIELDS			COND/REG CTL
CRAM MQ	MQM EN	MQM Sel 2	MQM Sel 1	MQ Sel 2	MQ Sel 1	SPEC/MQ SHFT	DISP/ DIV	DISP/ MUL	#07–08
0	0	0	0	1*	1*	0	0	0	00*
0	0	0	0	1*	0*	0	1*	0	0X*
0	0	0	0	1*	0*	1*	0	0	0X*
1	1	1*	0*	0	0				11*
1	1	0*	1*	0	0	0*	0*	0*	00*
1	1	0	0*	0	0	0	0	1*	00
0	0	0	0	1	0	0	0	0	01
1	1	1*	1*	0	0	0	0	0	10*
1	1	0*	0	0	0	0	0	0	11*
Reset	1	0	0	0	0	0	0	0	0

10-1642

Figure 2-73 MQ Selection

[1] INTERRUPTABLE *SPECIAL ADDRESS MANIPULATIONS

	GROUP	ADDRESS CALCULATION				DATA FETCH					EXECUTION				SPECIAL CONDS		STORE DATA	
OP CODES	CLASS	XR	@[1]	B	Y	IMM	READ	READ–WRITE	WRITE	READ PSE WRITE	36 BIT DP	18 BIT AP	23 BIT AP	10 BIT DP	CAN CAUSE PI	TRAP	WRITE	INTERRUPTABLE
200–217	MOVE GROUP	YES	YES	NO	YES	IMM	BASIC		MEM	SELF	ALL				NO	NO	IMM TO FM BASIC TO FM MEM TO E SELF TO E AND FM	
500–577	HALF WORD GROUP	YES	YES	NO	YES	IMM	BASIC		MEM	SELF	ALL				NO	NO	SAME AS FULL WORD GROUP	
120–125	DOUBLE WORD FULL WORD GROUP	YES	YES	NO	YES		BASIC		MEM		ALL				NO	NO	BASIC TO FM, FM+1 MEM TO E, E+1	
400–477	BOOLEAN GROUP	YES	YES	NO	YES	IMM	BASIC	SET MB	MEM	BOTH	ALL				NO	NO	SAME AS FULL WORD GROUP	
260–263	STACK GROUP	YES	YES	NO	YES	IMM	READ				ALL	ALL			NO	YES [PDOVL]	IMM TO FM ALSO CAUSES A FETCH FROM FM	
104–105	JSYS AND ADJSP	YES	YES	NO	YES	IMM					ALL	JSYS			NO	YES	IMM TO FM	
600–677	TEST GROUP	YES	YES	NO	YES	IMM	BASIC				ALL				NO	NO	IMM TO FM BASIC TO FM	
330–337 350–357 370–377	ARITHMETIC SKIPS	YES	YES	NO	YES		SKIPXX			SO SXX AO SXX	ALL				NO	NO	SKIPXX: IFA ≠ STORE (E); INAC	
300–317	COMPARES	YES	YES	NO	YES	CAIXX	CAMXX				ALL	CONDITIONAL ALL			NO	NO	CAIXX STORES NOTHING CAMXX STORES NOTHING	
320–367 252–253	CONDITIONAL JUMPS	YES	YES	NO	YES	IMM					ALL	CONDITIONAL ALL			NO	AOJX SOJX [AROV]	JUMPX STORES NOTHING AOJX TO FM SOJX TO FM	
252–253	ARITHMETIC TESTING	YES	YES	NO	YES	IMM					ALL	CONDITIONAL ALL			NO	NO	ALL TO FM	
264–267	SUBROUTINE CALL	YES	YES	NO	YES	IMM					ALL	UNCONDITIONAL ALL			NO	NO	JSR TO E JSP TO FM JSA TO E AND FM JRA TO AC } ALL CAUSE FETCH	
254–255	AC DECODED JUMPS	YES	YES	NO	YES	IMM					ALL	JRSTS ARE UNCONDITIONAL			NO	NO	HIGHER LEVEL FUNCTIONS PERFORMED	
256	XCT*	YES	YES	NO	YES	IMM					ALL	UNCONDITIONAL			NO	NO	FETCH IN KERNAL MODE PXCT	
257	MAP	YES	YES	NO	YES	IMM					ALL	UNCONDITIONAL			NO	NO	PAGING INFO TO FM	
270–277	FIXED POINT ARITH	YES	YES	NO	YES	ADDI SUBI	ADD SUB		ADDM SUBM	ADDB SUBB	ALL		YES		NO	ALL [AROV]	SAME AS FULL WORD	
220–227 230–237	FIXED POINT ARITH	YES	YES	NO	YES	IMUL IDIV		XMULM, XMULB XDIVM, XDIVB			ALL			ALL	NO	ALL [AROV]	IMULI, IDIVI TO FM IMUL, IDIV TO FM IMULM, IDIVM TO MEM IMULB, IDIVB TO FM, MEM MULI, DIVM TO FM, FM+1 MUL, DIV TO FM, FM+1 MULM, DIVM TO E, FM+1 MULB, DIVB TO E, FM+1	
114–117	DOUBLE INTEGER	YES	YES	NO	YES		BASIC				ALL			ALL	NO	ALL [AROV]	D ADD TO FM, FM+1 D SUB TO FM, FM+1 D MUL TO AC, AC+1, +2, +3 D DIV TO E, E+1	
140–147 150–157 160–167 170–177	SINGLE PREC FLOATING POINT	YES	YES	NO	YES	IMM	BASIC	MEM BOTH			ALL			ALL	NO	ALL [AROV]		
130–132 122 126–127	UFA, DFN, FSC FIX, FIXR FLTR	YES	YES	NO	YES	FSC	UFA, FIX FIXR FLT, FLTR			DFN						ALL [AROV]		
000–103	UUO'S	YES	YES	NO	YES	IMM					ALL			ALL	NO	NO	HIGHER LEVEL FUNCTIONS	
134–137	BYTE GROUP*	YES	YES	NO	YES		AC≠0 ADJBP AC=0 IBP	ILDB IDPB [FPD]			ALL			ALL	NO	NO	IBP, UPDATE POINTER (E) ILDB, UPDATE POINTER (E) BYTE → FM IDPB, UPDATE POINTER (E) BYTE ← AC	YES IN @ LOOP
240–247 (NOT 243)	SHIFTS AND ROTATES	YES	YES	NO	YES	IMM					ALL			ALL	NO	NO	LSH, ASH: AC → FM LSHC, ASHC: AC+1 → FM+1 ROT: AC → FM ROTC: AC+1 → FM+1	
251	BLT*	YES	YES	NO	YES	IMM					YES	YES	NO	YES	NO	NO	MULTIPLE WORDS MOVED SOURCE+N TO DEST+N	YES
700–777	INPUT OUTPUT	YES	YES	NO	YES	IMM	BASIC	BLKX	MEM		ALL			ALL	YES [CONO PI]	NO	E/E INTERFACE OPERATIONS	
250	EXCH	YES	YES	NO	YES					EXCH	ALL			NO	NO	NO	E ⇄ FM	
110–113	DOUBLE PREC FLOATING POINT	YES	YES	NO	YES		ALL				ALL			ALL	NO	ALL [AROV]	DFAD, DFSB: RESULT TO AC, AC+1 DFMP, DFDV: RESULT TO AC, AC+1	

10-1643

Figure 2-74 Instruction Set Divisions

Once the instruction has been loaded into IR and ARX, the major machine cycle begins; this is shown in Figure 2-75.

Three functional flows and two tables are included to supplement the functional descriptions of the address, fetch, and store cycles that follow.

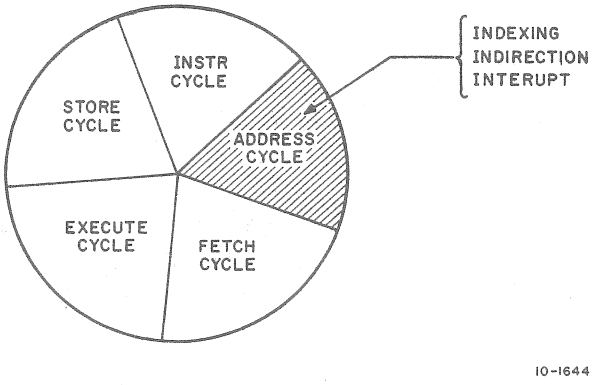


Figure 2-75 Major Machine Cycle

2.10.1 Effective Address Calculation

Figures 2-76 and 2-77 illustrate the instruction word formats. Bits 13–35 have the same format in every instruction whether the instruction addresses a memory location or not. Bit 13 is the indirect bit, bits 14–17 are the Index register address and, if the instruction must reference memory, bits 18–35 are the memory address Y. The effective address E of the instruction depends of the values of I, X, and Y.

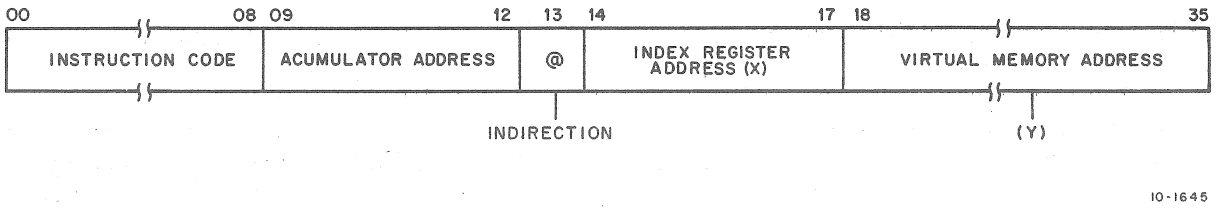


Figure 2-76 Basic Instruction Format

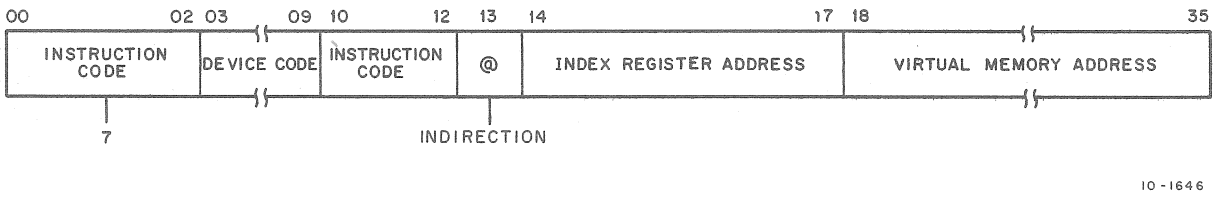


Figure 2-77 In-Out Instruction Format

2.10.1.1 Indexing – If the Index register address is nonzero, the contents of the specified Index register are added to the Y address to produce a modified virtual address.

Referring to Figure 2-78, the EBox tests ARX 14–17; if it is nonzero, the contents of the specified Index register are added to ARX 00–35. The result in AD 18– 35 is loaded into AR 18–35 with AR 00–17 cleared, and also loaded into VMA 18–35 while VMA 13–17 is recirculated.

2.10.1.2 Indirection – Whether indexing is performed or not, if ARX 13 is equal to 1, indirection will be performed. Two cases are to be considered. The first is where no indexing was performed. Here (indicated on Figure 2–78 as Ⓐ) VMA 18–35 is loaded via AD with ARX 18–35. In the second case, indexing is performed and the VMA is loaded via AD with AR. Here AR holds the sum of ARX 18–35 and FM 18–35 effectively, with AD bits 00–17 clear.

In either case, VMA 13–17 is recirculated while VMA 18–35 will be loaded via AD. The microinstruction MEM field function for the indirect request is MEM/AIND. This function has MEM 02 = 0, so MBOX WAIT is conditionally a function of the next microinstruction.

Testing for Interrupts

The microinstruction causing the EBox request also tests for a pending priority interrupt. If an interrupt is pending, the CRAM address is modified to allow entry to the PI Handler (Figure 2-79).

The request, which is made both to fast memory and core memory via the MBox, is ignored as long as it does not page fault. MBOX WAIT is false, so the EBox clock does not stop at this time. The EBox ignores an indirect reference when an interrupt is pending, but the EBox hardware remembers a page fault (if one occurs) until the page fault handler has been called. After the PF Handler is called, Force 1777 will be cleared.

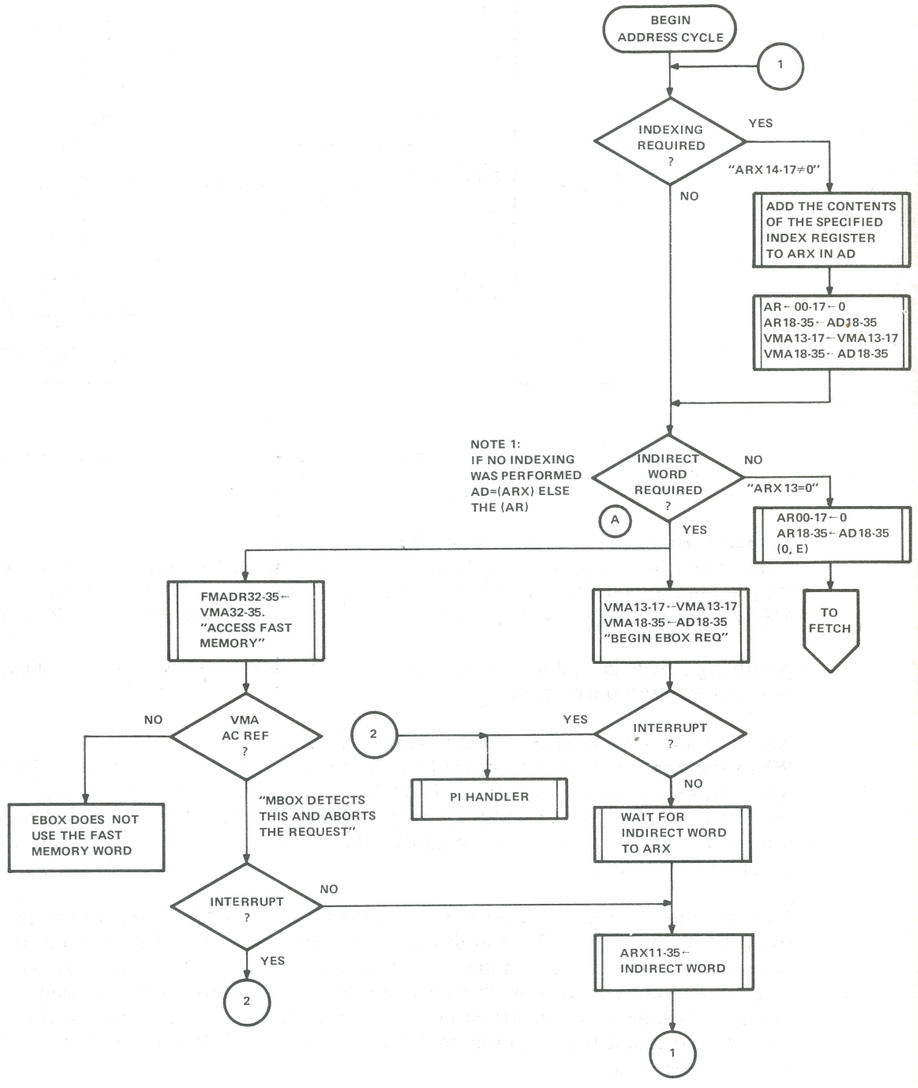
Referring to Figure 2-80, assume the indirect request has been started. Because the indirect reference is always a "READ," the only types of page faults that can occur in KI paging mode are no access (page not in core) or proprietary violation.

The requesting microinstruction detects the interrupt and the microprogram branches (via CRAM Address) to the PI Handler.

If the page fault occurs (for example) because of no access, the MBox must first read from the in core process table to obtain the paging information (use bits A, P, W, S, C and physical page). Reading this can take between 600 and 1000 ns. During this period, the PI Handler is setting up the requested PI service.

Eventually, a read, write or instruction fetch occurs, caused by the handler. When MBOX WAIT becomes true, the clock board (which remembered the Page Fail Hold level) forces the microprogram to the page fault handler.

Now the page fault handler detects the pending interrupt and the microprogram branches back to the PI Handler or to the instruction cycle. Thus, the entry to the page fault handler satisfied the clock board "page fail hold condition" and this condition now clears. Should the EBox make a second MBox reference before the page fault occurs, the EBox waits.



			NUMBER OF EBOX CYCLES REQUIRED		(A)		SEE NOTE 2	
REF CAN PAGE FAIL?	MBOX COMPLETE CYCLE?	VMA AC REF?	MBOX CYCLE	FAST MEM CYCLE	INDIRECT REF?	INTERRUPT ?	REQUESTING MICRO INSTR	NEXT MICRO INSTR
NO	NO MBOX TERMINATES	YES	BEGIN CYCLE, BUT EBOX IGNORES	BEGIN CYCLE, @ WORD TO ARX	YES	NO	MEM/AIND	MEM 02=1 MBOX WAIT
NO	NO MBOX TERMINATES	YES	BEGIN CYCLE, BUT EBOX IGNORES	BEGIN CYCLE BUT EBOX IGNORES	YES	YES-DIVERT TO PI HANDLER	MEM/AIND	MEM02=0
YES, IF SO EBOX DIVERTS TO PF HANDLER	YES	NO	BEGIN CYCLE, @ WORD TO ARX	BEGIN CYCLE, BUT EBOX IGNORES	YES	NO	MEM/AIND	MEM02=1 MBOX WAIT
YES, BUT NOT ACTED UPON UNTIL THE NEXT MBOX WAIT*	YES	NO	BEGIN CYCLE, BUT EBOX IGNORES	BEGIN CYCLE, BUT EBOX IGNORES	YES	YES-DIVERT TO PI HANDLER	MEM/AIND	MEM02=0

*ONCE IN THE PAGE FAULT HANDLER THE INTERRUPT PENDING WILL CAUSE A RETURN TO THE PI HANDLER AND THE PF HOLD EN LEVEL WILL BE CLEARED, REMOVING TEMPORARILY ALL TRACES OF THE FAULT.

NOTE 2:
MEM CYCLE ∧ MEM 02(1) = MBOX WAIT
MBOX RESP OR FM RESP CAUSES MEM CYCLE TO CLEAR MBOX WAIT ∧~VMA AC REF. EBOX CLOCK STOPS IF:

a. MBOX IS SERVICING THE EBOX REQ
b. WORD IS IN THE CACHE AND TIMEFIELD IS <3 OR . . .
a. MBOX IS SERVICING THE EBOX REQ
b. WORD IS NOT IN THE CACHE OR . . . .
a. MBOX IS SERVICING THE EBOX REQ
b. A PAGE FAULT OCCURS OR . . . .
a. A CONTROL RAM PARITY ERROR IS DETECTED OR . . . .

10-1647

Figure 2-78 Effective Address Calculation

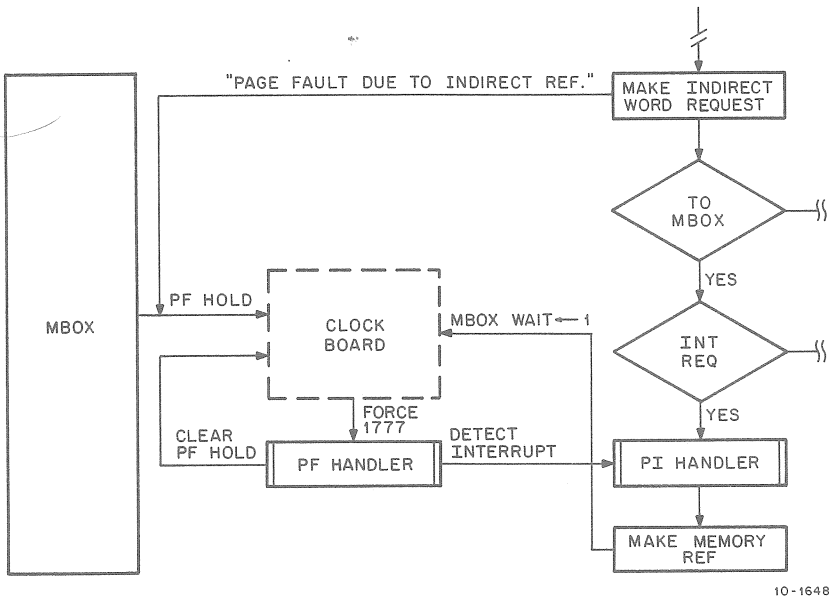


Figure 2-79 Page Fault During Diverted Indirect Reference

Normal Case – No Interrupts, MBox Request
When the EBox request is made specifically to the MBox, no interrupts are pending, the microinstruction following that which made the request (MEM/AIND) has its MEM field coded as ARX ← MEM. This function, together with MEM Cycle (1), will generate MBOX WAIT.

Assuming a page fault does not occur, the word loads into ARX. Now as indicated on Figure 2-79, the loop is reentered once again.

Normal Case – No Interrupts, Fast Memory Request
When the hardware determines that the VMA contains a fast memory address, it asserts VMA AC REF. This signal is used to inform the MBox that the EBox request is not to be handled by the MBox. Note that the fast memory address control uses VMA 32–35 to access fast memory even though the virtual address may be a core memory address. The hardware directs the use of the information accessed in this manner.

The effective address manager (Figure 2-15) branches within itself using the information provided from ARX 13 and 14–17. In addition, each time it samples this information it should branch to a microinstruction that enables the correct registers to be loaded; it may, however, invoke certain "don't care" operations, providing the next microinstruction executed performs the proper action. For example, assume a microinstruction is to always perform the indexing function in AD, but dispatch to a microinstruction that uses this information only if ARX 14–17 $\neq$ 0. This approach simplifies the design of the logic.

The table at the bottom of Figure 2-78 lists the four possible conditions resulting from indirect references to either MBox or fast memory.

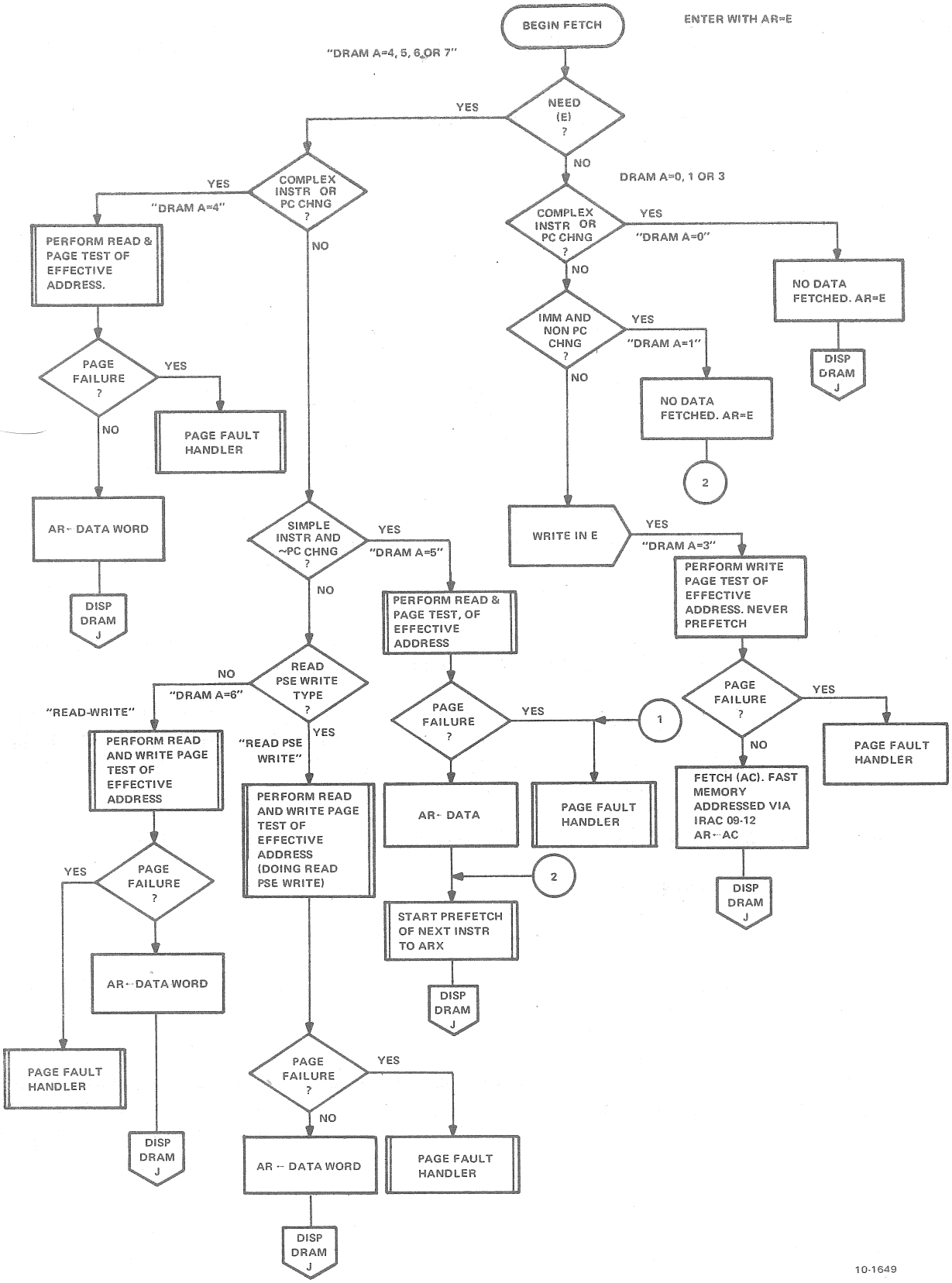


Figure 2-80 EBox Data Fetch

2.10.1.3 No Indirection or Indexing – For this case, ARX 18–35 contains the effective address. Here, it remains only to load AR 18–35 via AD with E and clear AR 00–17. The Fetch cycle is now entered.

2.10.2 Fetch Cycle
Once the effective address has been calculated, the second minor machine cycle is entered. This is the Fetch cycle and is illustrated in Figure 2-81.

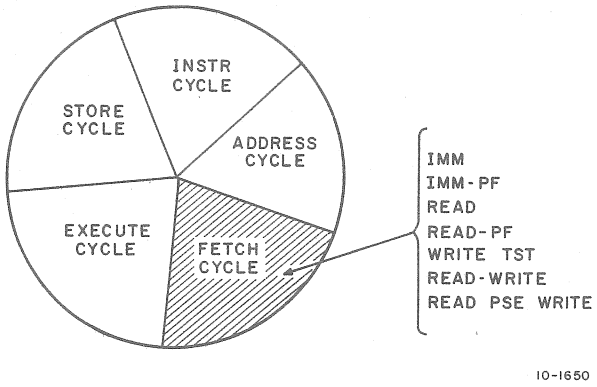


Figure 2-81 Fetch Minor Cycle

After the effective address has been calculated, the microprogram effective address manager gives "A READ DISPATCH" and control is passed to the Data Fetch Manager.

In general, two major classes of instructions exist in terms of the Data Fetch cycle. These two classes are those instructions that require the contents of the effective address and those that do not. Within each of these two categories are a number of divisions. The flow of the Fetch cycle is illustrated in Figure 2-80.

2.10.2.1 Instructions That Do Not Require (E) – Three general groups form this category.

1. Complex or PC change instructions
2. Immediate non-PC change instructions
3. Instructions that write in E

For these three groups, the DRAM A field is coded 0, 1, and 3, respectively. The AREAD Dispatch functions are listed in Table 2-16.

Complex or PC Change Instructions
The DRAM A field is coded as 0, and no data is requested. In addition, the next instruction is not prefetched. The AREAD/Dispatch dispatches directly to the execute code. This consists of a table lookup, where one discrete entry exists for each instruction. Thus, for example, the move instruction indexes into location "200" in the DRAM. The organization of the DRAM is illustrated in Figure 1-4.

Immediate and Non-PC Change Instructions
The DRAM A field is coded as 1, and no data is requested. The next instruction is prefetched and loads into ARX when the instruction becomes available. The AREAD/Dispatch dispatches directly to the execute code.

Table 2-16 AREAD Dispatch

DRAM A	DISP/AREAD	MEM/AREAD	Require (E)
0	Executor	No Prefetch	No
1	Executor	Start Prefetch	No
2	Not used		N/A
3	Symbolic Address 43*	Perform write test.	No
4	Symbolic Address 44*	"LOAD AR."	Yes
5	Symbolic Address 45*	A read operation is in progress: "LOAD AR, PREFETCH."	Yes
6	Symbolic Address 46*	LOAD AR. READ-PAUSE-WRITE	Yes
7	Symbolic Address 47*	LOAD AR, WRITE TEST	Yes

*The Data Fetch manager is a combination of hardware mostly on MCL and the microprogram consisting of 43–47.

Instructions That Write in E

The DRAM A field is coded as 3 and a write page test is initiated. If the address is not writable, a page failure occurs. This action causes a transfer to the page fault handler as indicated in Figure 2-80.

The appropriate Fetch EBox Qualifiers may be determined by referring to Figure 2-82. For DRAMA = 3 the following qualifiers are specifically asserted:

EBOX REQUEST
EBOX PSE
EBOX WRITE

In addition, the state of the qualifiers is more complex and may depend on the previous history of the EBox. The state is indicated by an asterisk (*). Once again referring to Figure 2-80, if the write page test is successful, the EBox fetches the contents of the addressed fast memory location (via IRAC 09–12) and then dispatches via the DRAM J field to the executor.

				EBOX REQUEST QUALIFIERS														
CYCLE	MEM FUNC	DRAM A	DRAM B	EBOX REQ	EBOX READ	EBOX PSE	EBOX WRITE	EBOX USER	MAY BE PAGED	KI PAGING MODE	VMA AC REF	PAGE ILLEGAL ENTRY	PAGE TEST PRIVATE	PAGE ADR COND	CACHE LOAD	CACHE LOOK	REMARKS
ADDRESS	A IND FOLLOWED BY LOAD ARX			X	X			*	*	*	*				*	*	INDIRECT WORD READ, MAY BE TO MBOX OR TO FAST MEMORY. VMA AC REF INDICATES WHICH VMA HOLDS ADR.
FETCH	FETCH	1 OR 5		X	X			*	*	*		•		*	*	*	INSTR FETCH. MAY OCCUR FOLLOWING A READ WITH DRAM A=1 OR 5 TOGETHER WITH MEM/FETCH.
FETCH	A READ	0		X	X			*	*	*	*	•		*	*	*	INSTR FETCH FOR JRST 0 (IR=JRST0)
EXECUTE STORE	FETCH			X	X			*	*	*	*	•		*	*	*	PI CYCLE IS CLEAR. USED WHERE NO PREFETCH WAS ISSUED TO CAUSE AN INSTR FETCH.
FETCH	A READ	4–5		X	X			*	*	*	*		①	*	*	*	DATA READ ISSUED BY INSTRUCTIONS REQUIRING THE (E) AS FOLLOWS: COMPLEX OR PC CHANGE INSTRUCTIONS OR SIMPLE NON PC CHANGE INSTRUCTIONS. ① ASSERTED IF ATTEMPTING TO READ DATA FROM A PRIVATE ADDRESS SPACE WITHOUT PROPER PROTOCOL. MBOX READ PAGE TESTS.
FETCH	A READ	6		X	X		X	*	*	*	*		②	*	*	*	DATA READ–WRITE ISSUED BY INSTRUCTIONS REQUIRING THE (E) WHICH CONDITIONALLY WRITE INTO E. THESE INSTRUCTIONS ARE AS FOLLOWS: NON READ PSE WRITE TYPE ② ASSERTED IF ATTEMPTING TO READ DATA FROM A PRIVATE ADDRESS SPACE WITHOUT PROPER PROTOCOL. MBOX READ AND WRITE PAGE TESTS. AR LOADS.
FETCH	A READ	7		X	X	X	X	*	*	*	*		③	*	*	*	DATA READ PSE WRITE ISSUED BY INSTRUCTIONS REQUIRING THE (E) WHICH WILL UNCONDITIONALLY WRITE INTO E. ③ ASSERTED IF ATTEMPTING TO READ DATA FROM A PRIVATE ADDRESS SPACE WITHOUT THE PROPER PROTOCOL. MBOX READ AND WRITE PAGE TESTS. IF CACHE IS DISABLED FOR THE CYCLE MBOX WAITS FOR WRITE PORTION OF CYCLE. I.E., PT CACHE (0) OR CACHE LOAD (0)∧ NOT FOUND. AR LOADS.
FETCH	A READ	3		X		X	X	*	*	*	*		④	*	*	*	DATA WRITE PAGE TEST ONLY. ISSUED BY INSTRUCTIONS NOT REQUIRING (E) BUT WHICH WILL WRITE INTO E. ④ ASSERTED IF ATTEMPTING TO WRITE DATA INTO A PRIVATE ADDRESS SPACE WITHOUT THE PROPER PROTOCOL. MBOX WRITE PAGE TESTS.
STORE	B WRITE	2–3		X			X	*	*	*	*		⑤	*	*	*	DATA WRITE (WRITE PAGE TEST AND WRITE DATA) USED BY THE GENERAL 4 MODE TYPE INSTRUCTIONS. I.E., IMM, BASIC, MEM, SEL FOR BOTH. SELF MODE STORES CONDITIONALLY IN E WHILE BOTH MODES ALWAYS STORE IN E. IN ADDITION BOTH MODES STORE UNCONDITIONALLY IN AC WHILE SELF E MODE STORES CONDITIONALLY IN AC. STORE VIA AR. ⑤ SAME AS ④.
EXECUTE	BYTE IND			X	X			*	*	*	*		⑥	*	*	*	BYTE POINTER INDIRECT WORD READ. USED AFTER BYTE POINTER HAS BEEN FETCHED WHEN BIT 13 OF THE POINTER IS 1. USED ONLY BY BYTE TYPE INSTRUCTIONS. ACTS LIKE EBOX READ TO MBOX. MBOX READ PAGE TESTS. BOTH AR AND ARX ARE LOADED. ⑥ SAME AS ③.
EXECUTE	BYTE RD			X	X			*	*	*	*		⑦	*	*	*	BYTE DATA READ. USED AFTER BYTE INDIRECT HAS COMPLETED. USED BY BYTE TYPE INSTRUCTIONS. ACTS LIKE EBOX READ TO MBOX. MBOX READ PAGE TESTS. BOTH AR AND ARX ARE LOADED. ⑦ SAME AS ⑥.
EXECUTE STORE MISC	WRITE			X			X	*	*	*	*		⑧	*	*	*	GENERAL PURPOSE WRITE. USED MANY PLACES. SOME EXAMPLES WOULD BE INSTRUCTIONS WHICH STORE MORE THAN ONE OPERAND, SUCH AS DOUBLE TYPE INSTRUCTIONS. INSTRUCTIONS WHICH SKIP, OR MODIFY AND SKIP BUT DID NOT FETCH (E) AND ARE GOING TO WRITE INTO E. MBOX TREATS AS WRITE. WRITE PAGE TESTS.
EXECUTE	LOAD AR			X	X		*	*	*	*	*			*	*	*	

• IF AN INSTRUCTION IS FETCHED BY A PUBLIC PROGRAM FROM A PRIVATE ADDRESS SPACE, AND THE INSTRUCTION IS NOT A PORTAL, ILL ENTRY WILL CAUSE THE MBOX TO PAGE FAIL ON THE NEXT MBOX REF.

* THESE QUALIFIERS ARE TRUE OR FALSE DEPENDING ON THE SPECIFIC TYPE OF REQUEST BEING MADE.

10-1651

Figure 2-82 Address-Fetch-Execute-Store General Memory References

2.10.2.2 Instructions That Require (E) – Under this category are four general groups. These groups are as follows:

1. Complex or PC change instructions
2. Simple non-PC change instructions
3. Non-(read-PSE-write) type instructions
4. Read PSE write type instructions

For these four groups, the DRAM A field is coded 4, 5, 6, or 7, respectively.

Complex or PC Change Instructions
The DRAM A field is coded as 4, causing a dispatch to location 44. A read page test is performed by the MBox. If the address is not accessible (not in core), the MBox performs a refill cycle and then checks the use bits.

If the access bit is clear, a page fault occurs and the EBox transfers to the page fault handler (microcode page fault handler). Otherwise, the requested word is loaded into AR. For the appropriate EBox qualifiers, refer to Figure 2-83. Finally, a DRAM J dispatch is performed to the executor.

Simple Non-PC Change Instructions
The DRAM A field is coded as 5, causing a dispatch to location 45. The basic read is the same as for DRAM A = 4. If no page fault occurs, the MBox issues MBox RESPONSE with the data word. Now the VMA loads with the prefetch address and this cycle begins. This MBox cycle will run in parallel with the Execution cycle, which may not use ARX. Finally, a DRAM J dispatch is performed at location 45; the VMA is loaded with PC + 1 and the prefetch begins.

Non-Read PSE Write-Type Instructions
A number of instructions are in this category; a few examples follow.

The first example is SETMB. This instruction (Boolean Group), reads a word from memory and unconditionally stores it in memory and AC. Because writing the word back into the same address is redundant, only a write page test is required to assure that the word (if in core memory) is writable. If this page fails, then the operation is aborted anyway. Otherwise, the word read is stored only in fast memory as addressed by IRAC 09–12. The read-write (separate cycles) may be thought of as consisting of a read and conditional write. If the write cycle is really desired, the MEM field function MEM/Write may be used to write (Figure 2–82).

The second example concerns instructions such as IDIVM, IDIVB, DIVM, and DIVB. These instructions reference memory for both read and can generate no divide. This aborts the division operation. If the class of instruction is read PSE write and the cache is disabled for the reference, then the MBox waits for the write portion of the cycle; the EBox performs an unnecessary write operation.

A third cause is for BLKI and BLKO I/O instructions. Here a pointer word is fetched from the effective address. This pointer is normally updated and stored back in the effective address.

One problem is that the legality of performing the I/O instruction is tested after the pointer has been fetched. This is necessary because the pointer is fetched during the Fetch cycle, while legality (IO LEGAL) is tested during execution. Should the BLKX instruction be illegal in the current EBox mode, an unnecessary pointer back off and write would be necessary.

Other cases are concerned with very long instructions, which could hold up the MBox.

CYCLE	COND FUNC	SPEC FUNC	MEM FUNC	# FIELD 0 – 8	EBOX REQUEST QUALIFIERS: EBOX REQ	VMA EPT	VMA UPT	PAGE UEBR REF	EBOX MAP	EBOX LOAD REG	EBOX READ REG	EBOX CCA	EBOX UBR	EBOX EBR	EBOX ERA	EN REFILL RAM WR	EBOX SBUS DIAG	WRPT SEL 0	WRPT SEL 1	PT WR	PT DIR WR	MBOX CTL 03	MBOX CTL 06	REMARKS
EXECUTE MUUO PART 1,2. METER REQUEST, PAGE FAIL PART 1,2.		SP MEM CYCLE	WRITE SEE FIG 2-54	=220	X		X	X																WRITE INTO USER PROCESS TABLE. MBOX USES VMA 27–35 ONLY. MBOX APPENDS UBR 14–26 TO FORM PHYSICAL REFERENCE UNPAGED. CANNOT PAGE FAIL.
EXECUTE MUUO PART 3, PAGE FAIL PART 3		SP MEM CYCLE	LOAD AR FOLLOWED BY MB WAIT SEE FIG 2-54	=220	X		X	X																READ NEW PC WORD FROM USER PROCESS TABLE. MBOX USES VMA 27–35 ONLY. MBOX APPENDS UBR 14–26 TO FORM PHYSICAL REFERENCE UNPAGED. CANNOT PAGE FAIL.
EXECUTE DEPOSIT		SP MEM CYCLE #08(1): PHYS REF	WRITE SEE FIG 2-54	=110	X	X		X																FOR DEPOSIT WRITE ACCORDING TO RELOCATED VIRTUAL ADDRESS. THERE WAS NO PROTECTION VIOLATION.
EXECUTE EXAMINE, DEPOSIT, METER REQ		SP MEM CYCLE #08(1): PHYS REF	LOAD AR FOLLOWED BY MB WAIT SEE FIG 2-54	=110	X	X		X																FOR DEPOSIT OR EXAMINE, READ PROTECTION AND RELOCATION INFORMATION FROM EXEC PROCESS TABLE. FOR MTR REQUEST READ DOUBLE PRECISION WORD AS APPROPRIATE.
EXECUTE STD 40+2N INT 2ND PART STD INT		SP MEM CYCLE #00(1): FETCH EN IN	LOAD ARX	=510	X	X		X																INSTR FETCH FROM EPT FOR STD INTERRUPT OR 2ND PART OF STD INTERRUPT. MBOX USES VMA 27–35 ONLY. MBOX APPENDS EBR 14–26 TO FORM PHYSICAL REFERENCE UNPAGED. CANNOT PAGE FAIL.
EXECUTE NICOND PI VECT, EXAMINE SEE FIG 1-25		SP MEM CYCLE #08(1): PHYS REF	LOAD AR FOLLOWED BY MB WAIT SEE FIG 2-54	=110 IF EPT SPECIFIED SEE FIG 1-25	X	*	SPEC EXEC KILLS USER EN	*																FETCH WORD FROM ADDRESS SPACE SPECIFIED VIA API WORD BITS 0–2 MAY BE: EPT, EXEC VIRTUAL, UPT, USER VIRTUAL, PHYSICAL. THE STATE OF VMA EPT, VMA UPT, MAY BE PAGED CONTROLS CONTEXT OF REFERENCE.
NICOND TRAP FETCH		SP MEM CYCLE	LOAD ARX FOLLOWED BY LOAD ARX	=30	X	*	*	*																FETCH INSTR FROM USER PROCESS TABLE IF TRAP OCCURRED IN USER MODE AND EXEC PROCESS TABLE IF TRAP OCCURRED IN EXEC MODE. THE STATE OF VMA UPT AND VMA EPT IS A FUNCTION OF MCL USER EN.
EXECUTE BLKO PI	I/O LEGAL		REG FUNC	=7	X												X							TRANSMITS 36 BITS OF CONTROL INFORMATION FETCHED FROM E TO THE INTERNAL OR EXTERNAL MEMORIES. THIS IS PERFORMED VIA THE MBOX. THE MEMORY CONTROLLER SELECTED RETURNS A WORD WHICH IS STORED IN E+1.
EXECUTE DATAI PAG	I/O LEGAL		REG FUNC	=142	X						X		X											READS INFORMATION FROM EBOX AND MBOX. FIRST READS AC BLOCKS, CWSX AND VMA PREV SECT THEN REQUESTS MBOX TO PUT UBR INTO EBUS REG. EBOX THEN READS EBUS REG.
EXECUTE CONI PAG	I/O LEGAL		REG FUNC	=143	X						X			X										READS INFORMATION FROM EBOX AND MBOX. FIRST READS LOOK, LOAD, SEC TRAPEN, THEN REQUESTS MBOX TO PUT EBR INTO EBUS REG. EBOX THEN READS EBUS REG AND STORES RESULT IN E.
EXECUTE BLKI PI	I/O LEGAL		REG FUNC	=144	X						X				X									READS THE MBOX ERROR ADDRESS REGISTER. THE WORD IS STORED IN THE SPECIFIED AC.
EXECUTE BLKO PAG	MBOX CTL			STEP 1=1 STEP 2=#21 STEP 3=#0																X		X		INVALIDATE ENTRIES IN THE PAGE TABLE WITHIN THE MBOX. EBOX USER REFLECTS USER MODE, AND THE PAGE TO INVALIDATE IS IN VMA.
EXECUTE DATAO PAG	I/O LEGAL		REG FUNC	=242	X					X			X											LOADS CONDITIONALLY AC BLKS, PREV CONTEXT, AND UBR (IN MBOX) AND CLEARS THE PAGE TABLE IN THE MBOX. VMA 25–35 CONTAIN THE DATA TO BE LOADED INTO UBR.
EXECUTE CONO PAG	I/O LEGAL		REG FUNC	=243	X					X				X										LOADS THE CACHE STRATEGY BITS (LOOK, LOAD) SECTION AND TRAP EN FLAGS IN THE EBOX. THE EBR (IN THE MBOX) IS LOADED VIA VMA 24–35.
EXECUTE MAP	I/O LEGAL		REG FUNC	=540	X				X		X													READS PAGE FAILWORD FROM MBOX EBUS REGISTER. THE WORD IS STORED IN THE SPECIFIED AC.
EXECUTE BLKO APR	I/O LEGAL		REG FUNC	=145	X					X						X								WRITE THE CACHE REFILL ALGORITHM. VMA 18–20 CONTAINS THE ALGORITHM BITS AND VMA 27–33 CONTAIN THE REFILL ALGORITHM ADDRESS.

10 1652

Figure 2-83 Execute-Register-MBox Control and Miscellaneous General Memory References

The DRAM A field is coded as 6, causing a dispatch to location 46; the MBox performs both a read and write page test. The address must be both accessible and writable, even though this portion of the operation only reads a word. If a page failure occurs, the EBox transfers control to the page fault handler. Otherwise, the word enters AR and then a DRAM J dispatch is issued.

Read PSE Write Type Instruction
The DRAM A field is coded as 7 causing a dispatch to location 47; the request qualifiers are shown on Figure 2-82. The MBox performs both a read and write test, and if no page fault occurs, reads a word from the specified (Xlated) address.

If the cache is disabled for the reference and the word requested was not in the cache (a Refill cycle was necessary first), then the MBox is held waiting until the EBox issues the write portion of the cycle. The word requested loads into AR and a DRAM J dispatch is issued to enter the Executor.

2.10.3 Execution Cycle

The Executor is entered from the Fetch cycle. While in the Fetch cycle, the (E) or (AC) is fetched in accordance with the DRAM A field. In addition, read and/or write page testing is performed while in the Fetch cycle. The EBox Execution cycle overview is in Figure 2-84.

Early in the Instruction cycle, the DRAM is accessed using one of three basic types of addresses.

Referring to Figure 2-84, if the instruction is JRST 0–17, then the IR address is used to address the DRAM initially as indicated. Thus, the JRSTs handler is entered at location 254 for JRST and 255 for JFCL.

From the initial dispatch into the handler, the IRAC is used to redispatch within the handler for the proper type of JRST. For JFCL, a JUMP is made to a separate handler from the initial dispatch

If the instruction is an I/O type, then the DRAM address is formed by the hardware such that the dispatch is in the range of 700–777. Once the I/O handler has been entered, a determination must be made as to whether the instruction is legal in the current processor mode. If it is determined that the instruction is not legal, the MUUO executor is used to store the illegal instruction and PC word in the user process table. Following this, a new PC word is fetched. This new PC word causes the processor to enter an executive routine in core memory. If the I/O instruction is legal, use of the EBus is obtained and the appropriate EBus dialogue is carried out. The specific actions evoked depend upon the device and the type of I/O instruction being performed.

The remaining instructions index into the DRAM utilizing the op code in IR bits 00–08. Two general categories exist as follows:

1. Simple Type – stores in AC, E, or both

2. Complex Type – may store in AC, AC+1, E via normal store cycle or else store via a special handler, or may do some of each

The complex instructions may nest microcode subroutines up to four levels deep.

Referring to Figure 2-85, the mechanism consists of CRA LOC, a register that is loaded with the "current microinstruction address." This register is loaded at the same time that the CRAM register is loaded with a new microinstruction. In addition, a 4-word stack is provided. The contents of CRA LOC are pushed onto the top of the stack when the call has been asserted by the microinstruction. To return from a subroutine, the returning microinstruction asserts DISP/Return. This pops the top entry off of the stack and onto the CRAM address mixer lines, where it is logically ORed with the J field of the microinstruction, asserting DISP/Return.

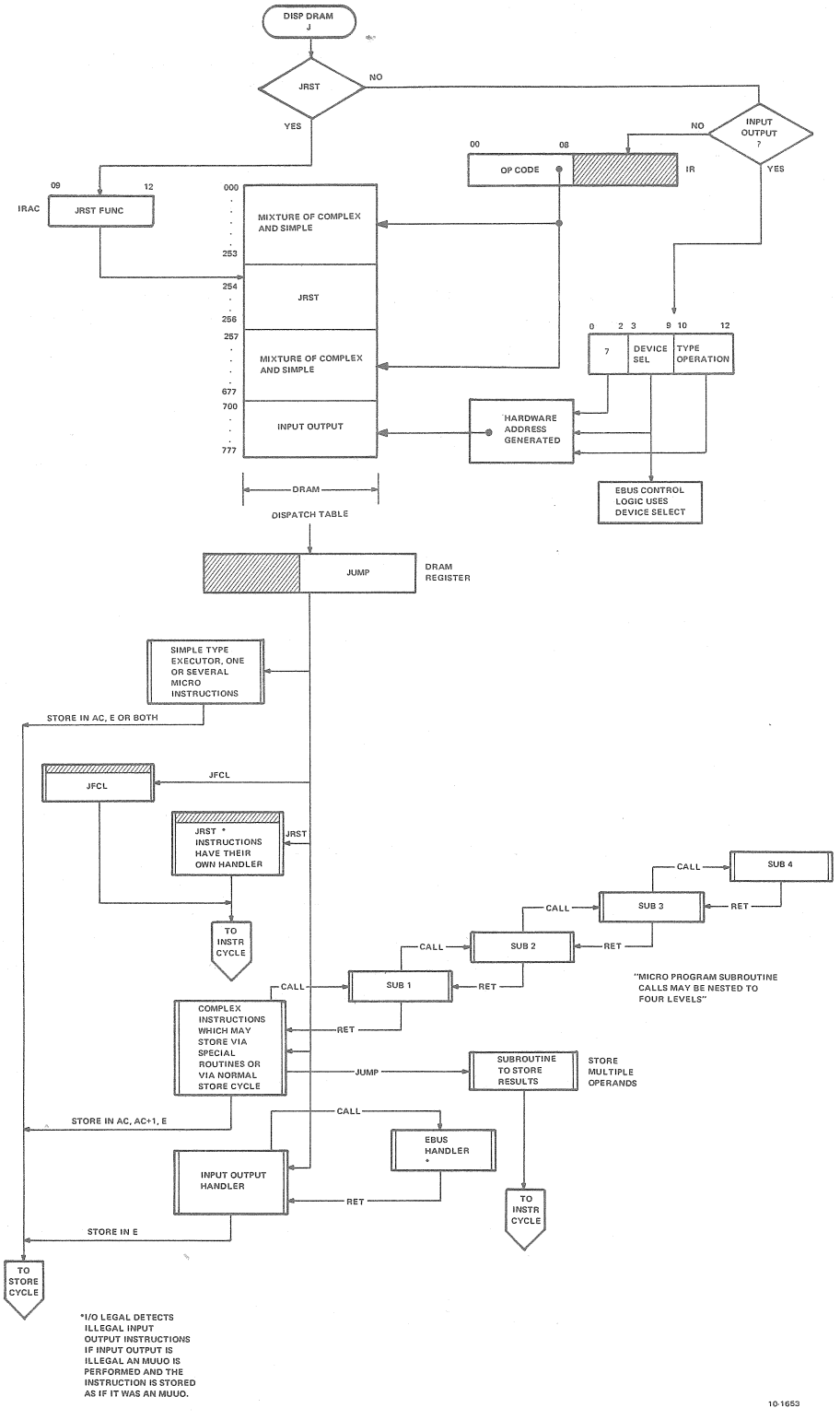


Figure 2-84 EBox Execution Cycle Overview

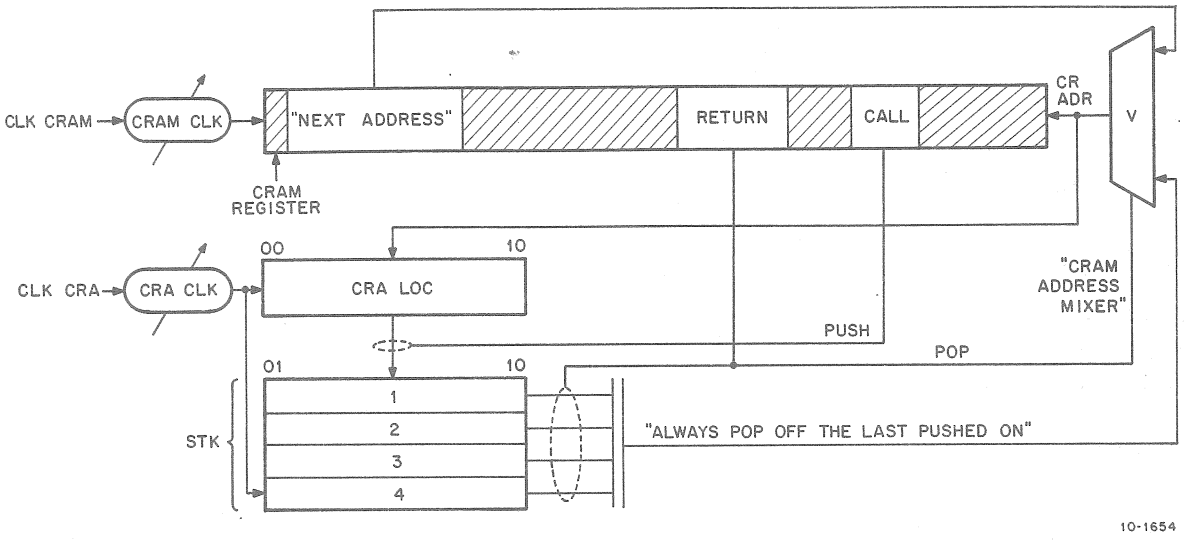


Figure 2-85 Microstack Operation

Some of the complex instructions, such as DMUL, which stores in AC, AC+1, AC+2, and AC+3, use a separate handler for storing multiple operands. This type of instruction does not pass through the normal store cycle. Other complex instructions, such as MULB, which stores in AC, AC+1 and E, store multiple operands via the normal store cycle.

2.10.4 EBox Data Store Cycle

The flow for the EBox Store cycle, illustrated in Figure 2-86, is used by most of the instructions executed by the microprogram Executor. Exceptions to this are certain instructions such as DMUL, which stores more than two ACs. For these instructions, a special handler exists that is entered from the executor. This handler stores all the operands and then issues an instruction fetch followed by a NICOND Dispatch. In this text, the more general categories (which do use the normal store cycle) are covered.

2.10.4.1 Basic Four Mode Type Instructions – This type of instruction may have one of four basic modes as follows:

1. Immediate or Basic – store in AC only
2. Memory – store in E
3. Both – store both in AC and E
4. SELF – store in E and conditionally store in AC. Note that if writing back in E is redundant, the write cycle is skipped.

Writing for these four mode instructions is controlled by MEM/DRAM B and the DRAM B field code. The store cycle is dispatched with DISP/DRAM B. Thus, the dispatch RAM B field (three bits) is used to form the low-order three bits of the Store cycle address.

Immediate or Basic Mode

Referring to Figure 2-87, the DRAM B field is coded as 5. The contents of AR are written into fast memory, which is addressed via IRAC 09–12. Because a large number of these instructions prefetch the next instruction, it is necessary to assert MB WAIT in the event MEM cycle is set waiting for a response from the MBox. This has no affect if MEM cycle is clear. NICOND Dispatch enables entry to the instruction cycle if no priority interrupts, page faults, or traps are pending.

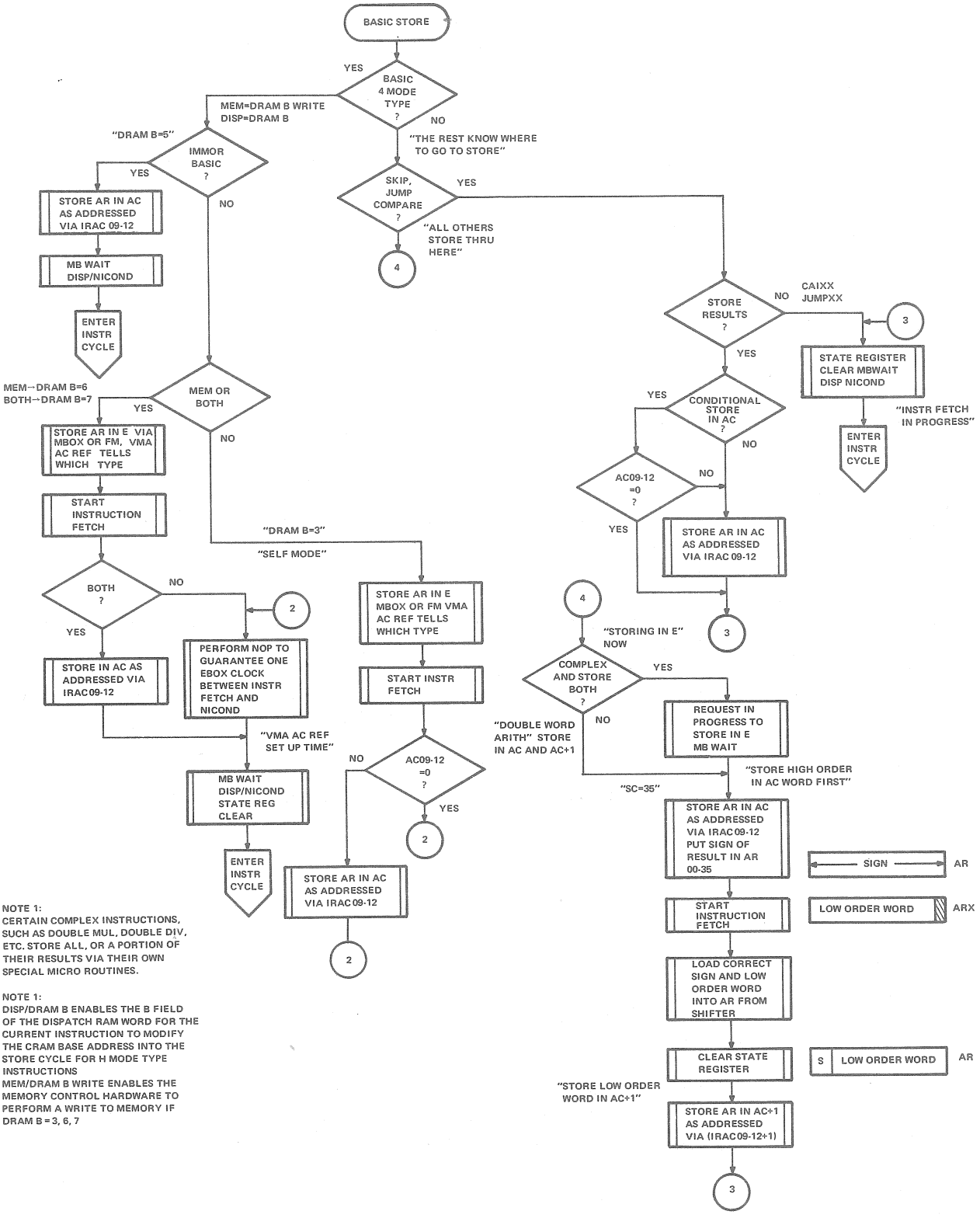


Figure 2-86 EBox Data Store

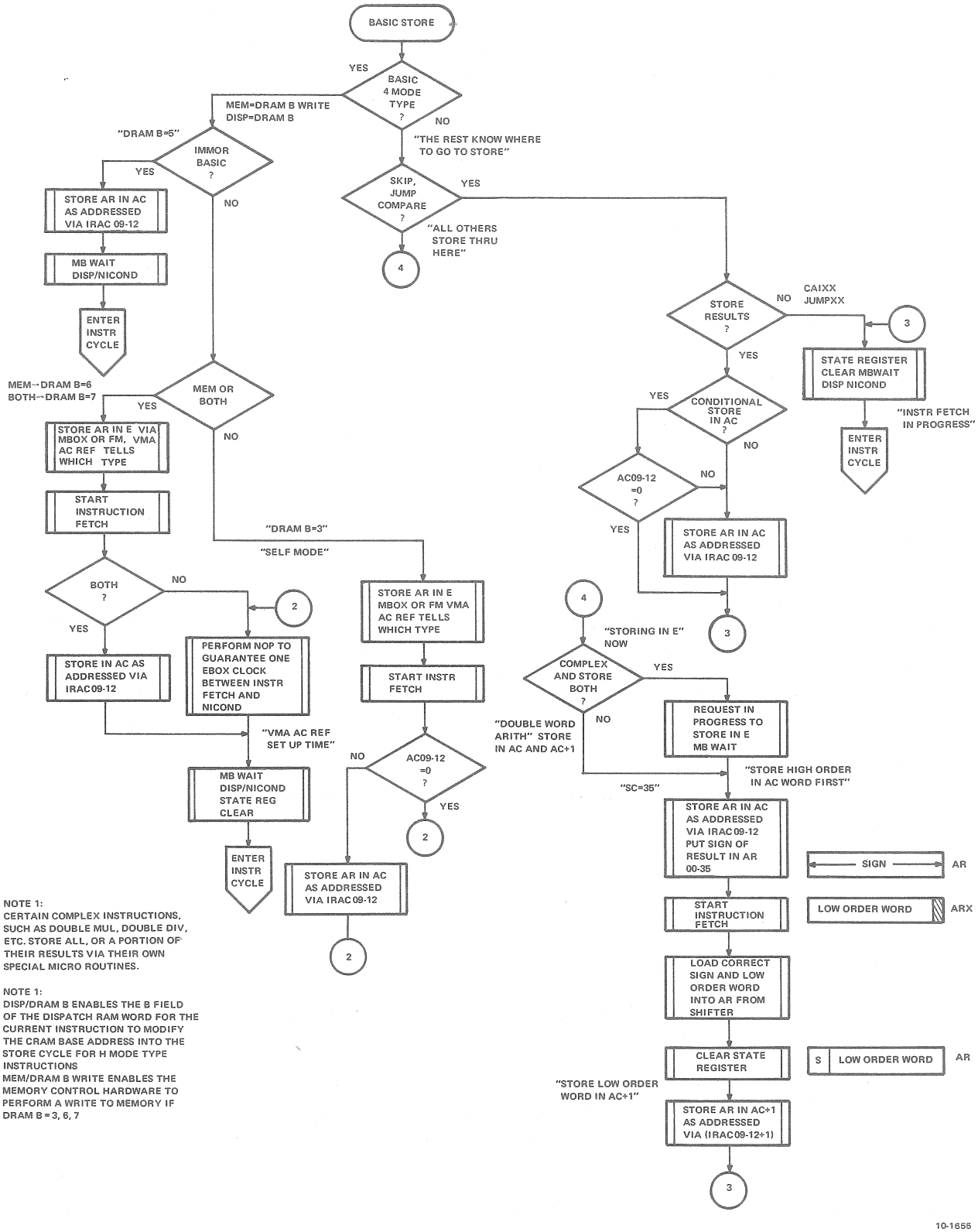


Figure 2-86 EBox Data Store

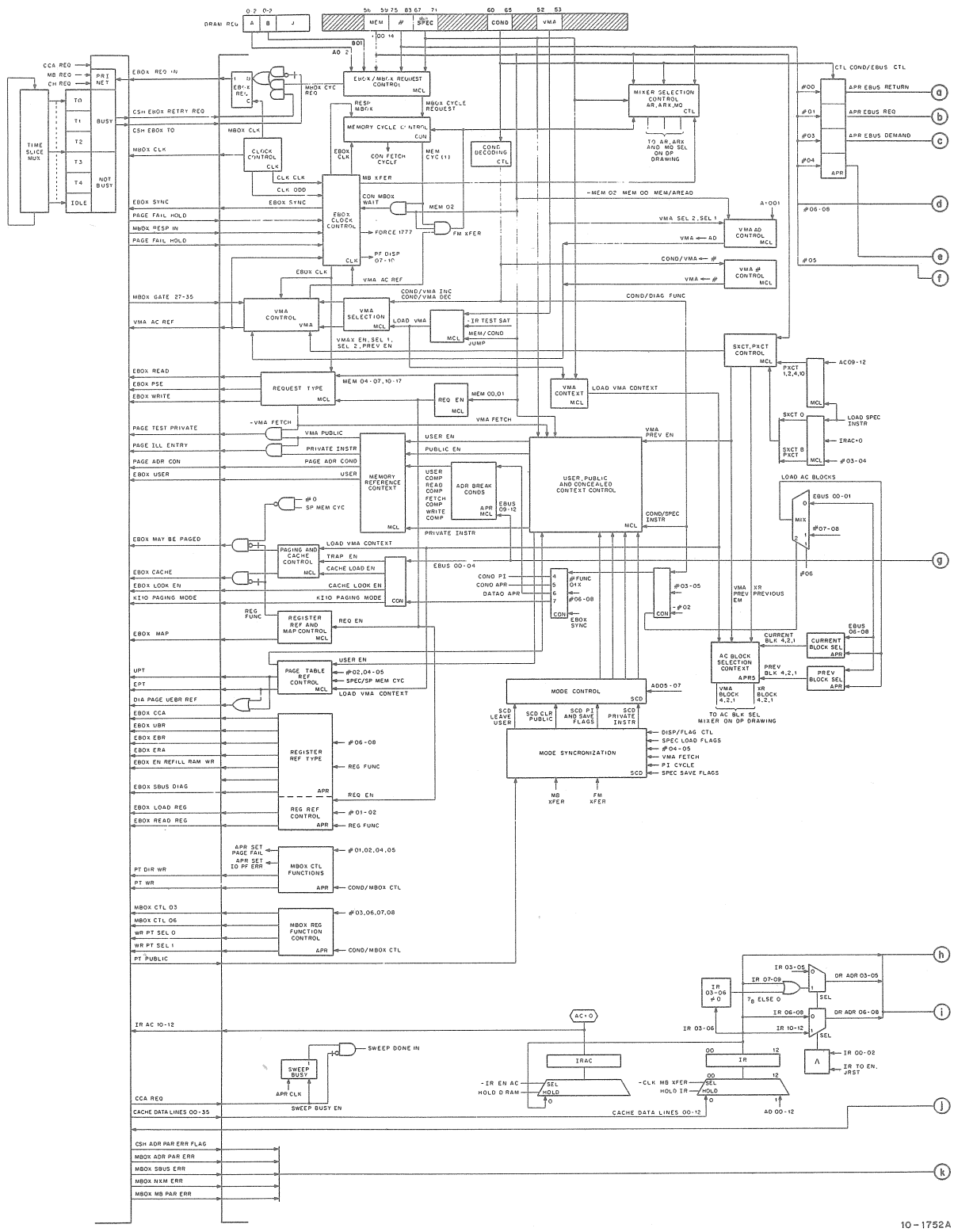


Figure 2-87 MBox-EBox-EBus Control (Sheet 1 of 2)

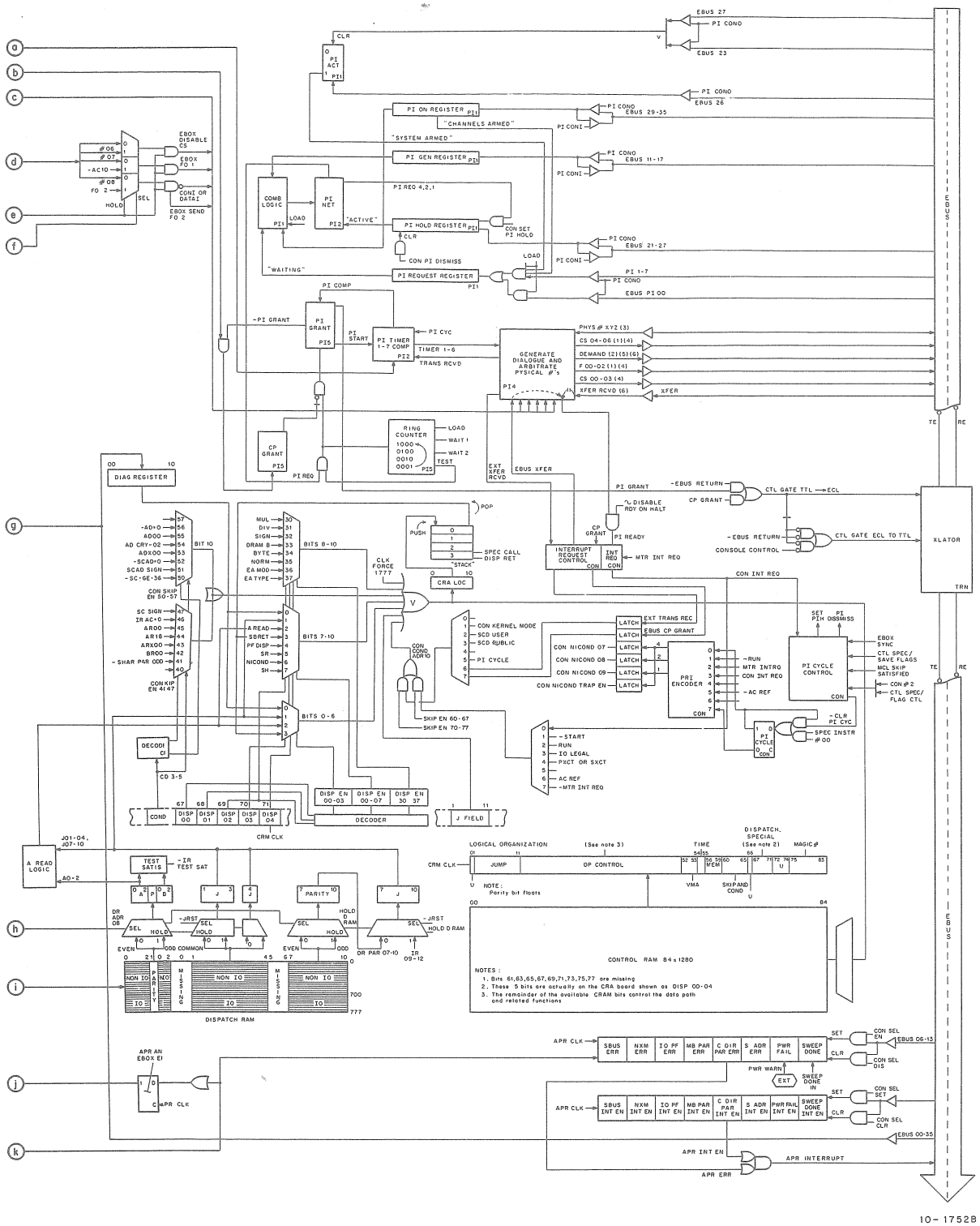


Figure 2-87 MBox-EBox-EBus Control (Sheet 2 of 2)

Memory or Both Mode
The DRAM B field is coded as 6 for memory mode instructions. If VMA 13–33 is clear, storing is to fast memory. Otherwise, an MBox request is made to store AR in cache memory. VMA AC REF notifies the MBox to abort the cycle when it is to fast memory. An unconditional instruction fetch is enabled at this time. The VMA input is via VMA AD (PC+1) and, as soon as MBox RESPONSE is received, this is latched into VMA.

To allow VMA addressing to stabilize in case the instruction is being fetched from fast memory, a NOP microinstruction is performed. This is followed by MB WAIT, state register clear (in case the instruction fetch page fails), and finally NICOND Dispatch is issued.

For Both Mode, DRAM B is coded as 7. Here, the departure is made after storing the AR in E. The AR is also stored in fast memory as addressed by IRAC 09–12. Now MB WAIT is asserted while clearing the state register and NICOND Dispatch is issued.

SELF Mode
Once again referring to Figure 2-87, the DRAM B field is coded as 3. SELF mode instructions are generally read/write type; this means that the virtual address was read and write page tested during the fetch cycle.

Writing is allowed only if not redundant, or as specified by IRAC being nonzero. AR is stored in E, the instruction fetch is started, and the AC field of the instruction is tested (in IRAC). If IRAC is nonzero, the AR is stored in the addressed fast memory location (as addressed via IRAC). If IRAC is zero, no storing in fast memory is performed. In either case, a microinstruction NOP is performed. This guarantees one EBox clock between the instruction fetch and the NICOND Dispatch to follow, allowing adequate setup time for the NICOND logic to detect a fast memory reference (VMA AC REF) for those cases where the instruction fetch is to fast memory.

2.10.4.2 SKIP, JUMP Compare Instructions – The following instructions listed in Table 2-17 fall into this category.

Table 2-17 Skip, Jump, Compare Instructions

Main Group	Instr	Unconditional Store	Conditional Store	Stores Nothing	Op Code
Arithmetic Skips	SKIPXX	No	Yes if IRAC ≠ 0	No	330–337
	AOSXX	Yes	Yes if IRAC ≠ 0	No	350–357
	SOSXX	Yes	No	No	370–377
Conditional Jumps	JUMPXX	No	No	Yes	320–327
	AOJXX	Yes	No	No	340–347
	SOJXX	Yes	No	No	360–367
Arithmetic Testing	AOBJP	Yes	No	No	252
	AOBJN	Yes	No	No	253
Compares	CAIXX	No	No	Yes	300–307
	CAMXX	No	No	Yes	310–317

No Results Stored – CAIXX, JUMXX
Referring to Figure 2-87, because CAIXX and JUMPXX store no results, preparations are made for entry to the instruction cycle. The state register is cleared, MB WAIT is asserted, and a NICOND Dispatch is issued. Depending upon the outcome of Test Satisfied, the next instructionfetch is from PC+1, PC+2, or E.

Conditional Storage in AC – SKIPXX AOSXX, SOSXX
IRAC is sampled and if nonzero, AR is stored in fast memory as addressed via IRAC 09–12. Depending upon the outcome at Test Satisfied, the next instruction fetch is from PC+1 or PC+2 and this is in progress. The state register is cleared, MB WAIT is asserted, and a NICOND Dispatch is issued.

Unconditional Storage – SOJXX, AOJXX, AOBJX
These instructions all store unconditionally, in fast memory from AR, as addressed via IRAC, then prepare to enter the Instruction cycle. The state register is cleared, MB WAIT is asserted, and NICOND Dispatch is issued. Both SOSXX and AOSXX unconditionally store in E and conditionally store in AC.

2.10.4.3 Store Cycle for Other Instructions – Generally, the remaining instructions that use the Store cycle fall into two groups. These are instructions that store results in AC, AC+1 and E, and those instructions that store results in AC and AC+1 only. All these are complex instructions.

Complex and Store Both
For these instructions, the store flow is entered with a write request already in progress to store the high-order result of some operation and MB WAIT is asserted (MEM/MBWAIT). Also, the shift counter (SC) contains 35, enabling alignment of the low-order word with the sign of the high-order word later in this flow. The AR is now stored in fast memory as addressed via IRAC and the sign is smeared in AR 00–35. At this time, AR contains all sign bits and ARX contains the low-order word left-justified. The instruction fetch begins. The AR and ARX are shifted left 35 places and the result (correctly signed) is loaded into AR via SH. Now the state register is cleared and the low-order word (in AR) is stored in IRAC + 1. The EBox hardware facilitates the incrementation of IRAC by +1. Finally, the appropriate entry to the instruction cycle is made.

Complex and Store in AC, AC+1
The basic difference here is that these instructions bypass the storage into E. Otherwise, the operation is identical to that for Complex and Store Both.

2.11 INTERFACE CONTROL

2.11.1 Introduction
Figure 2-88 illustrates the major functional control elements of the EBox. The purpose of this drawing is to support the functional descriptions contained in this section. In addition, it is provided to support the E/M interface control and E/E interface control functional descriptions to follow.

The EBox is associated with two interfaces, the EBox/MBox Interface and the EBox/EBus Interface. The E/M interface is treated as a pseudo-bus because in many ways it behaves as a bus. In the first portion of the functional description, the basic organization and function of the firmware microprogram was described. In addition, the major machine cycle was defined and described in terms of its functional elements.

Thus, the individual microprogram modules (Figure 2-13), taken collectively, comprise the main microprogram. The blending of this program with certain pieces of EBox hardware constitutes the basic machine cycle (Figure 2-88).

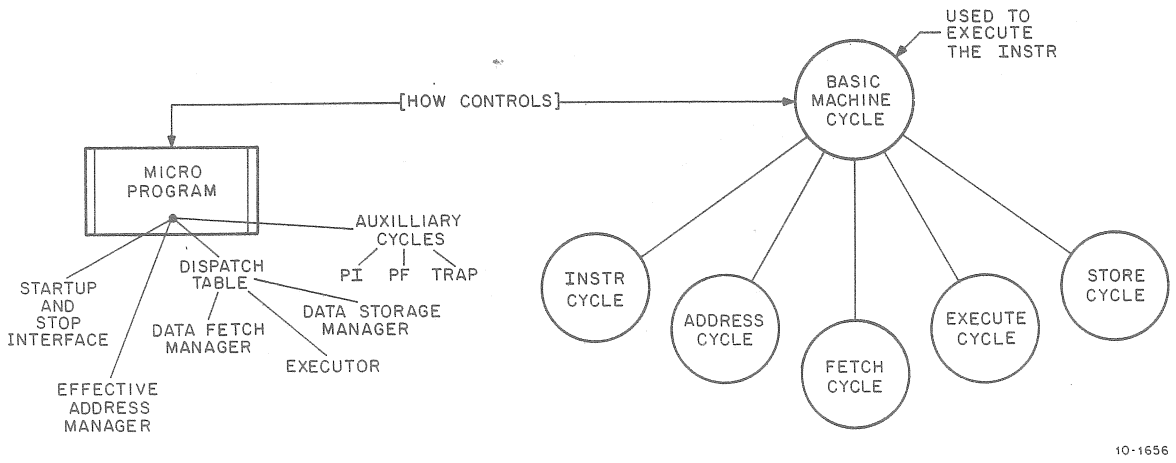


Figure 2-88 Basic Machine Cycle Summary

Figure 2-89 is the subcycle summary and Figure 2-90 is the hardware cycle summary.

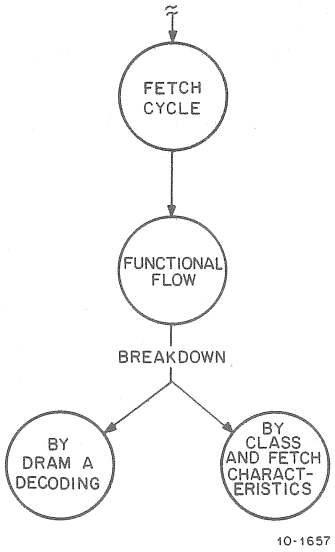


Figure 2-89 Subcycle Summary

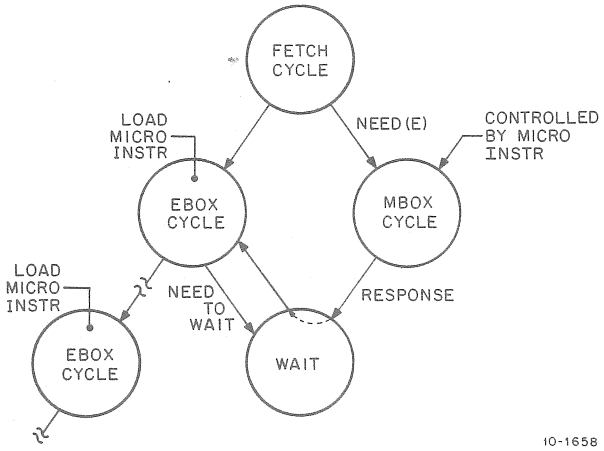


Figure 2-90 Hardware Cycle Summary

Next, the basic subcycle was presented in terms of a functional flow with additional graphics to support the description; in the interface section, the relationship of the hardware to the internal EBox cycles was described. These basic cycles were introduced in Subsection 2.1 as EBox, MBox, and EBus cycles. For example, the fetch cycle can be viewed as composed of a number of EBox and MBox cycles.

2.11.2 MBox Control

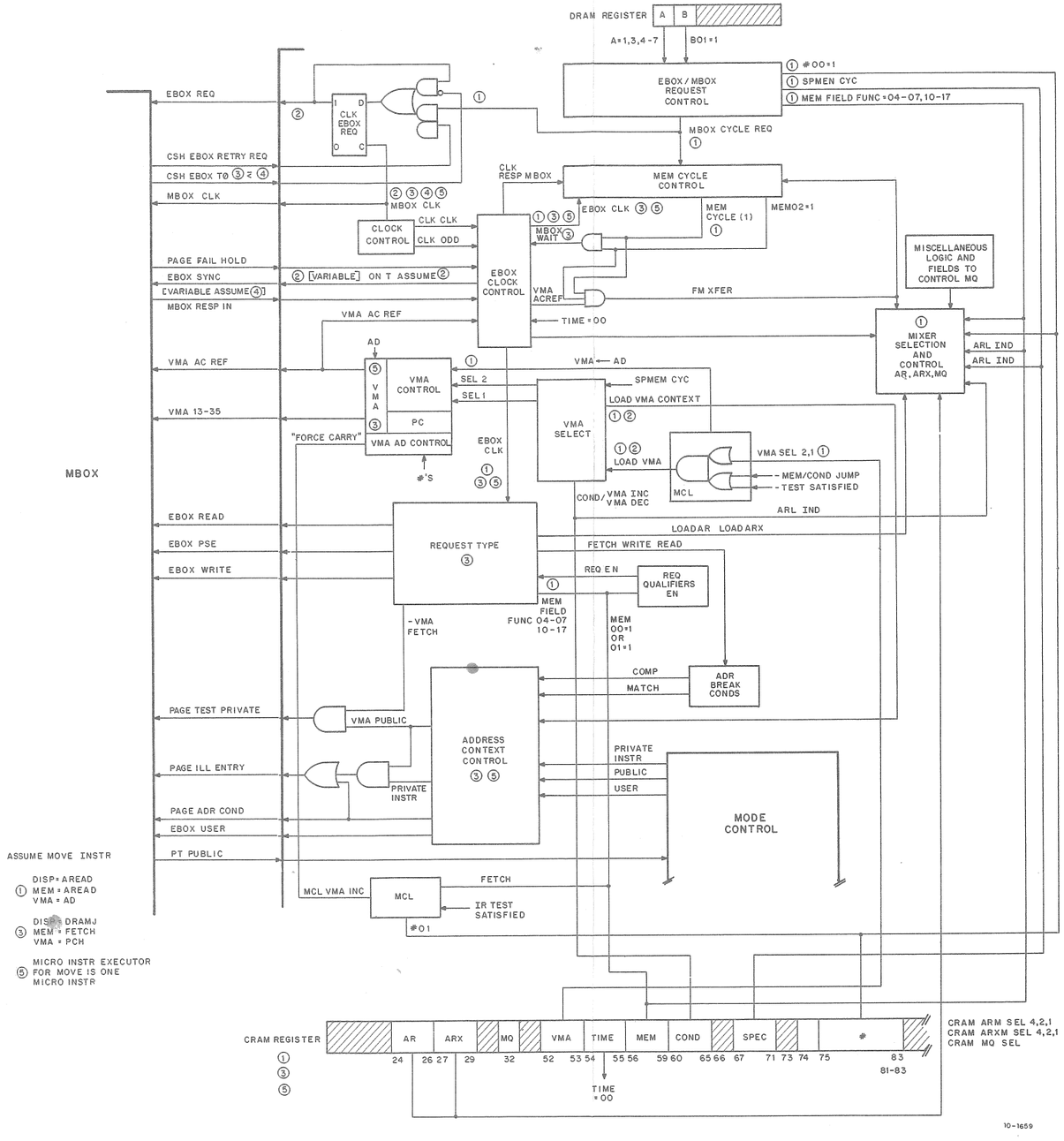
Referring to Figure 2-91, a number of functional elements work together to implement the basic MBox cycle. The grouping of the interface signals shown is as listed in Table 2-18.

To exercise the functional areas illustrated on Figure 2-91, a basic data fetch is covered in four steps. These steps are related to EBox timing in terms of occurrence.

Table 2-18 Request Summary

Grouping	Signals
Basic EBox Request Handshake	EBOX REQUEST CSH EBOX TO CSH EBOX RETRY REQ PF HOLD MBOX RESPONSE IN
Address and Address Control	VMA 13–35 VMA AC REF
Timing	EBOX SYNC MBOX CLOCK
Type Request	EBOX USER EBOX READ EBOX PSE EBOX WRITE
Address Violation Logic	PAGE TEST PRIVATE PT PUBLIC PAGE ILLEGAL ENTRY PAGE ADDRESS COND

MBOX



ASSUME MOVE INSTR

① DISP=AREAD
MEM=AREAD
VMA=AD

③ DISP=DRAMJ
MEM=FETCH
VMA=PCH

⑤ MICRO INSTR EXECUTOR FOR MOVE IS ONE MICRO INSTR

Figure 2-91 General Memory Request Control Simplified

2.11.2.1 DATA FETCH REQUEST EN – Begin EBox Cycle (Figure 2-92) – The flow is entered at an EBox clock and the CRAM register loads. The microinstruction begins to be decoded. Note that the MEM field is the major input to the MBox control logic. Assume that the effective address has been calculated, the MEM field is coded as AREAD, and the dispatch RAM A field is 5. In Figure 2-91 at ① , the MEM field function AREAD is a code of 4. This enables MBOX CYCLE REQ. In addition, if MEM 01 = 1, then REQ EN is asserted to enable the request qualifiers to be latched on the next EBox clock. MBOX CYCLE REQ enables the EBox request to be asserted on the next MBox clock. As indicated on the flow, this is a fast cycle. Two basic classes exist: fast and slow. The timing is illustrated in Figure 2-93.

Signal CLK SYNC EN must wait to occur, so that (for a fast cycle) EBOX SYNC sets at the same time as EBox request.

Referring to Figure 2-91, the VMA field, with other signals, enables LOAD VMA. In addition, the effective address must be input to VMA via AD so the VMA code (3) generates VMA ←AD.

The basic period between the leading edge of one EBox clock and the leading edge of the next is controlled by the T field of each microinstruction, along with certain other hardware signals. The basic pulse width of the positive EBox clock is fixed at 32 ns but the time between clocks is variable. EBOX SYNC occurs one MBox clock prior to the MBox clock that causes EBox clock to occur. The basic relationships are indicated in Figure 2-94.

2.11.2.2 Begin MBox Cycle – End Current EBox Cycle and Start Next (Figure 2-95) – As soon as SYNC EN is true, EBOX SYNC sets and MBOX CYCLE REQ (FAST CYCLE) enables EBox request to set (refer to ② on Figure 2-91). At this point, MBOX WAIT is tested and found clear. (This function is described in basic terms is Subsection 2.2.4.)

To summarize, the EBox request is then issued, and the VMA input mixer is set up and enabled to load with E via AD. The request type logic is enabled to assert the appropriate combination of EBox Read, PSE, and/or Write (which occur on the EBox clock to come at ③). In addition, the Address Context Control is enabling the proper combination of its qualifiers also to be asserted at ③.

Now another MBox clock occurs ③ ; simultaneously, an EBox clock occurs. The following actions result:

EBOX CLOCK ← 1
EBOX REQ ← 1 (REDUNDANT)
MEM CYCLE ← 1; MBOX WAIT ← 1
VMA LOADS AND LATCHES
CRAM ← NEXT MICRO INSTR
EBOX QUALIFIERS LATCHED

Thus, we have passed through one EBox cycle and now reenter the flow to begin a second EBox cycle.

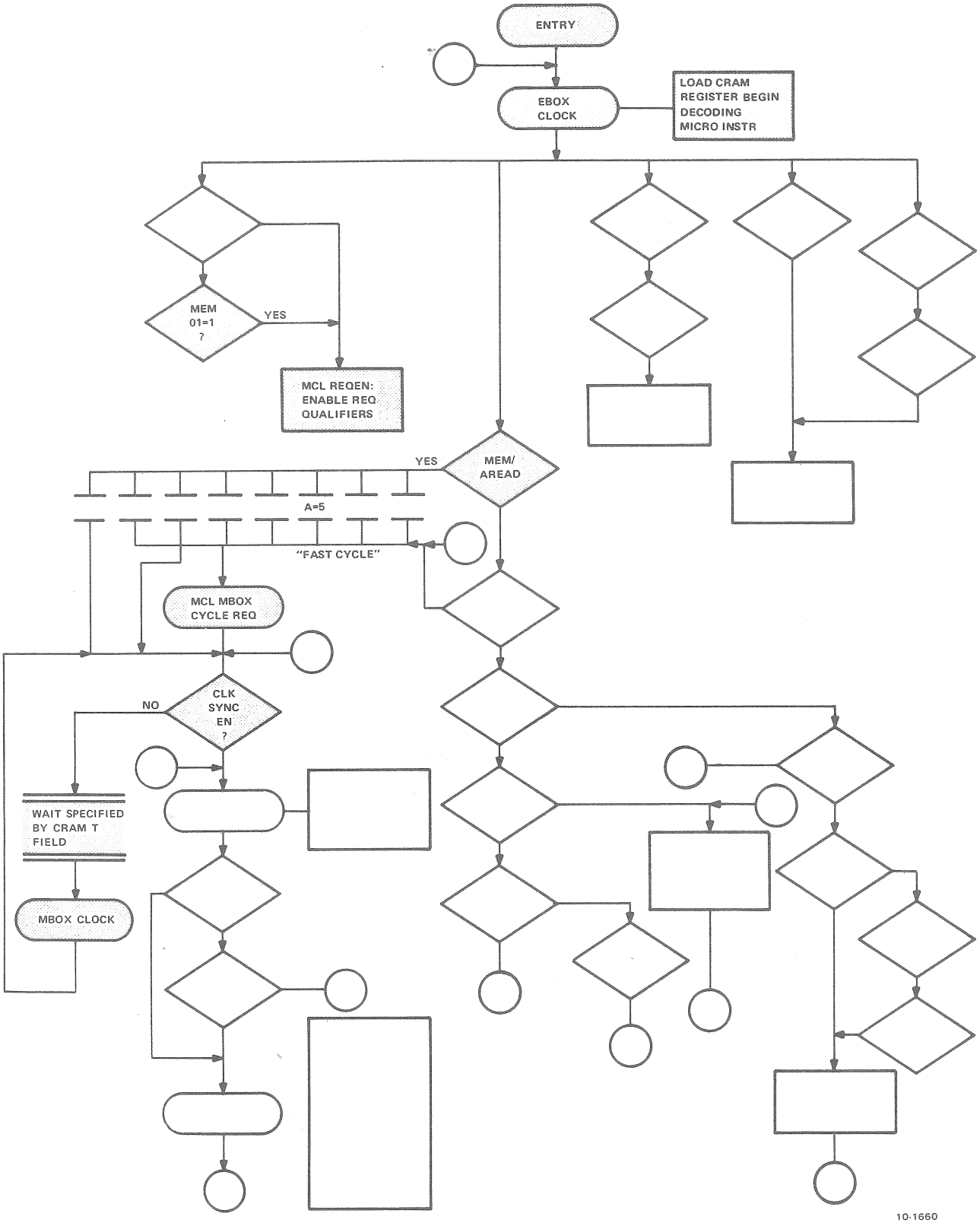


Figure 2-92 Begin EBox Cycle Data Fetch Request

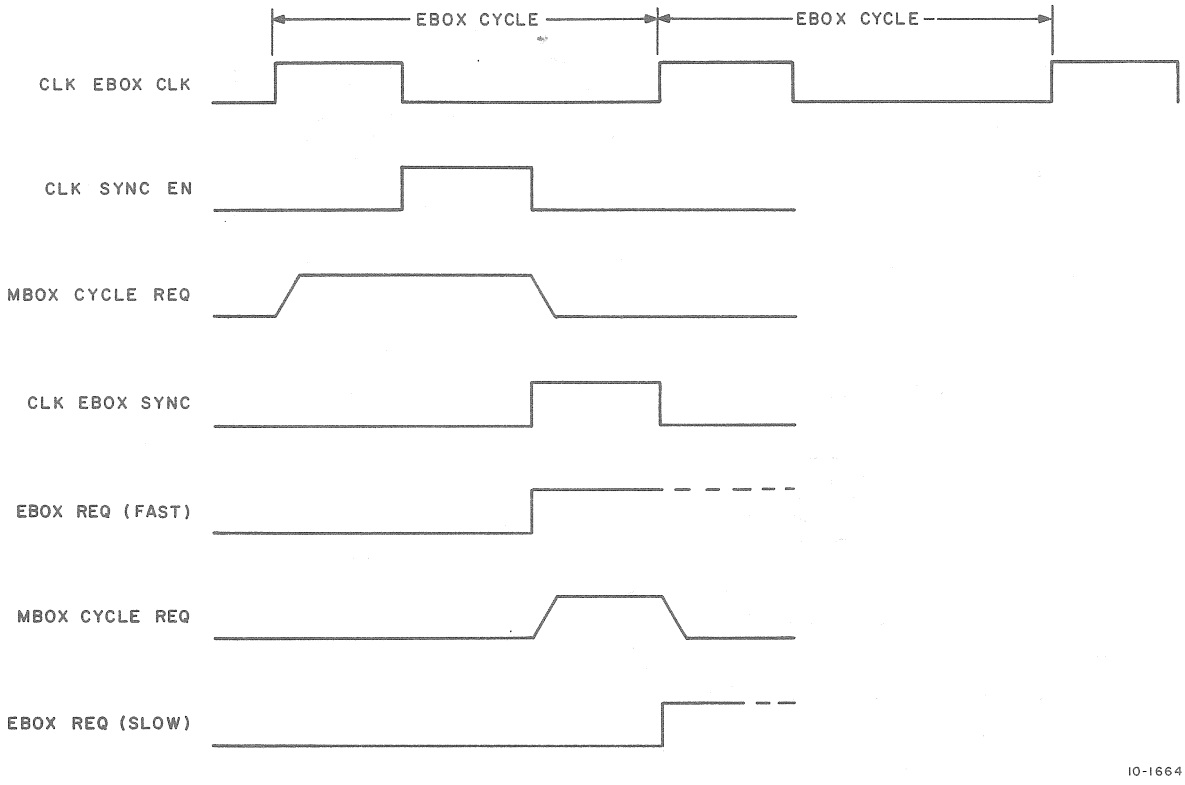


Figure 2-93 EBox Request Fast or Slow

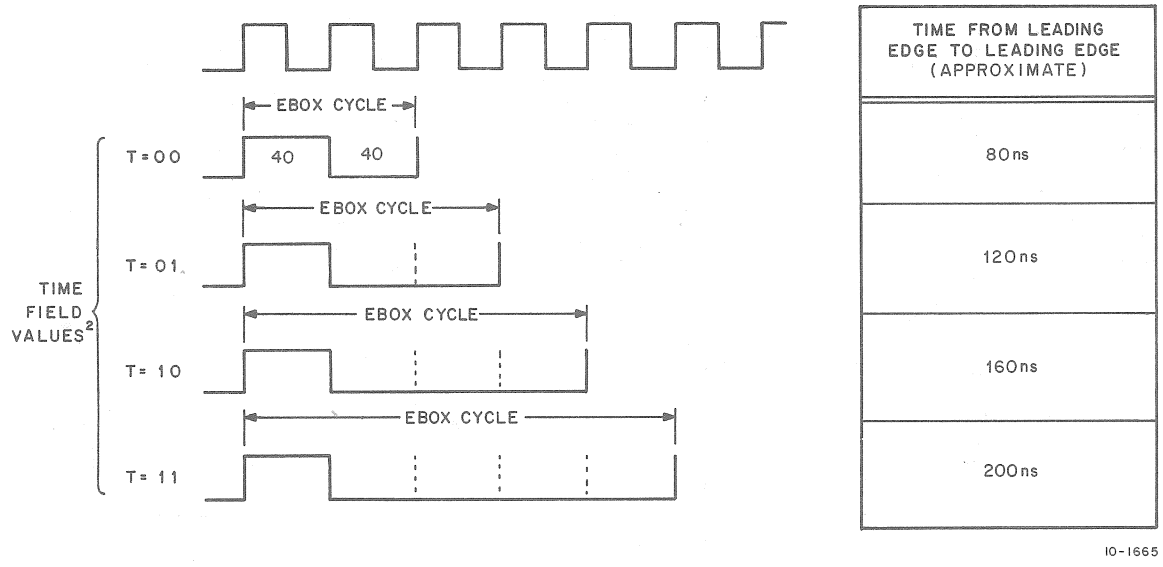


Figure 2-94 Basic EBox Clock Period

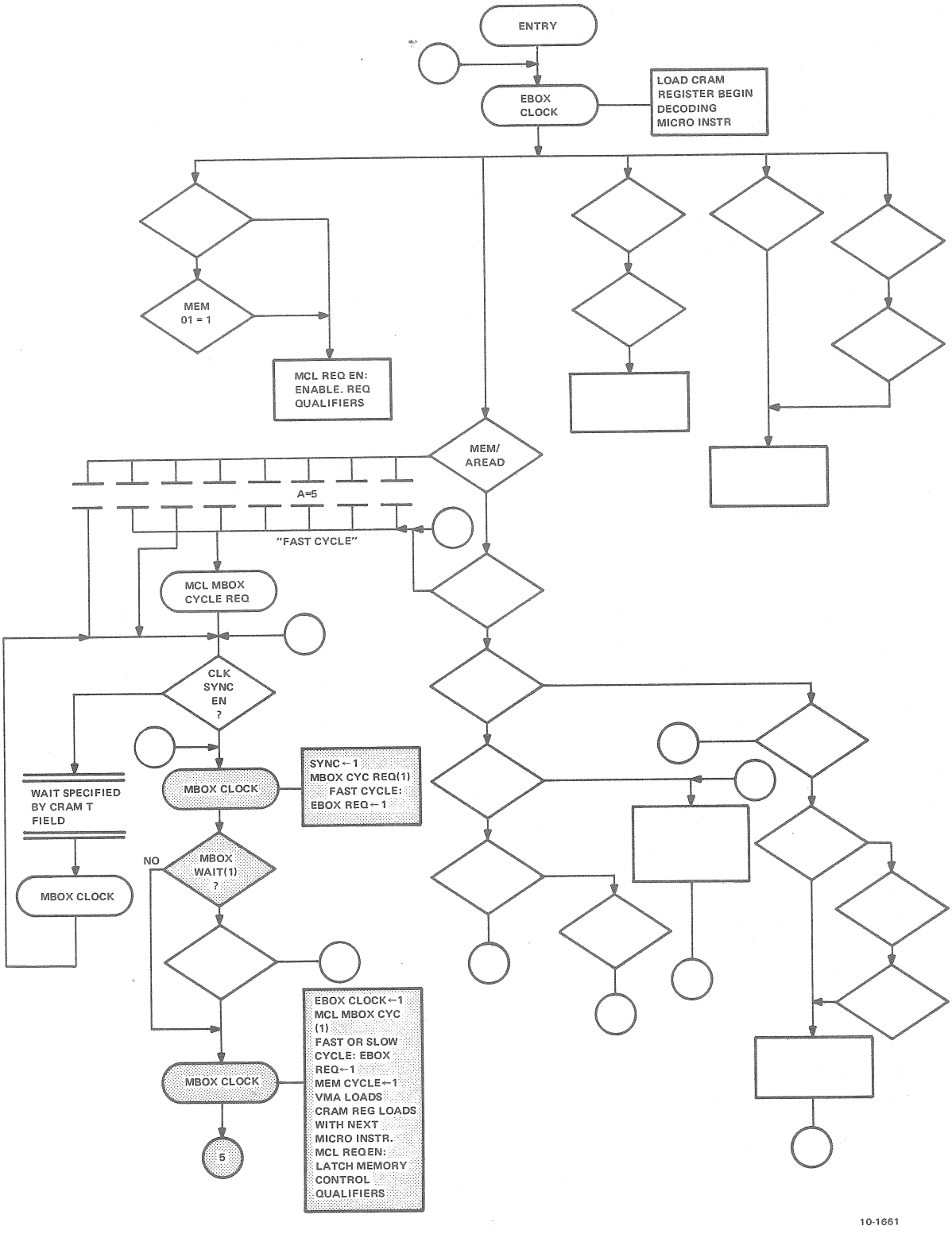


Figure 2-95 Begin MBox Cycle, End Current EBox Cycle, Begin Next EBox Cycle

2.11.2.3 SETUP PREFETCH – Wait for MBox Response – Referring to Figure 2-96, the flow is reentered at ⑤ where the EBox clock generated loads the second microinstruction (Figure 2-91 ③). Now the MEM field function is FETCH and MEM 02 = 1. If the MBox has not responded with the word requested (E), MEM cycle is still set. The combination of MEM 02 (1) and MEM Cycle (1) generates MBOX WAIT. Providing that the request is not to fast memory, the EBox stops until the MBox response occurs.

This is true whether a page fault occurs or not, although PF hold is asserted 5 MBox clocks before MBOX RESPONSE is asserted when a page fault has occurred. In this example, assume that the MBox is working on the request, but has not yet responded.

Referring to the flow (Figure 2-96), the current microinstruction MEM field function fetch is a code of 6. Note, however, that because a priority interrupt takes precedence over any other activity, PI CYCLE is checked before enabling the MCL MBOX CYCLE REQ. Here PI CYCLE is clear, so ② points to a "Fast Request." Again, a wait for SYNC EN, as defined by the T field, takes place. The state of the SYNC EN during MBOX WAIT is always true; this keeps EBOX SYNC true until the response is received.

The MBox continues to run during the waiting period. Thus, MBOX CLOCK sets EBOX REQUEST even though the VMA is still latched up with E. During the waiting period, the VMA input receives PC+1 via VMA AD.

The EBox now loops, waiting for MBOX RESPONSE to restart the EBox clock.

2.11.2.4 MBOX RESPONSE RECEIVED – Referring to Figure 2-97, MBOX RESPONSE enables the EBox clock. Thus, EBOX CLOCK becomes true and, simultaneously, EBOX SYNC becomes false. The third microinstruction is now loaded into the CRAM register (Figure 2-91 ⑤) and is decoded. In addition, the VMA is loaded and latched with PC+1, the request qualifiers are latched and now, with the requested data word in AR, a DRAM J dispatch is issued.

2.11.2.5 General Memory Cycle Control – Figure 2-98 contains all combinations of the MEM field that can generate MCL MBOX CYCLE, and hence EBOX REQ. In general, the following functions are of the "Slow Cycle" type:

B WRITE
PI FETCHES
SKIP SATISFIED FETCHES
REG FUNCTIONS
SP MEM CYCLES

A Slow cycle is required during MEM/REG FUNC because the MBox requires additional time to decode the type of request. In all the "slow" cycle types, the EBOX does not necessarily have time to determine whether to make the request (or not) before EBOX SYNC. Thus, the decision, and therefore the request, is delayed purely for hardware timing reasons.

2.12 EBUS INTERFACE CONTROL

The I/O system for the KL10 processor includes the EBus, the peripheral equipment with its interfaces to the EBus, and various control logic. The EBus interface may be controlled either by the EBox during input or output instruction execution, or by the PI system during priority interrupt handling. Subsection 2.8.1 gives a basic summary of the EBus signals. This is followed by a functional description of the interface, which is covered at two levels. The first level describes the basic functional organization and operation of the PI board and other related logic. The second description deals with the microprogram to PI board interfacing. This description attempts to give insight into the manner in which the hardware and the microprogram interact to carry out various interface related functions.

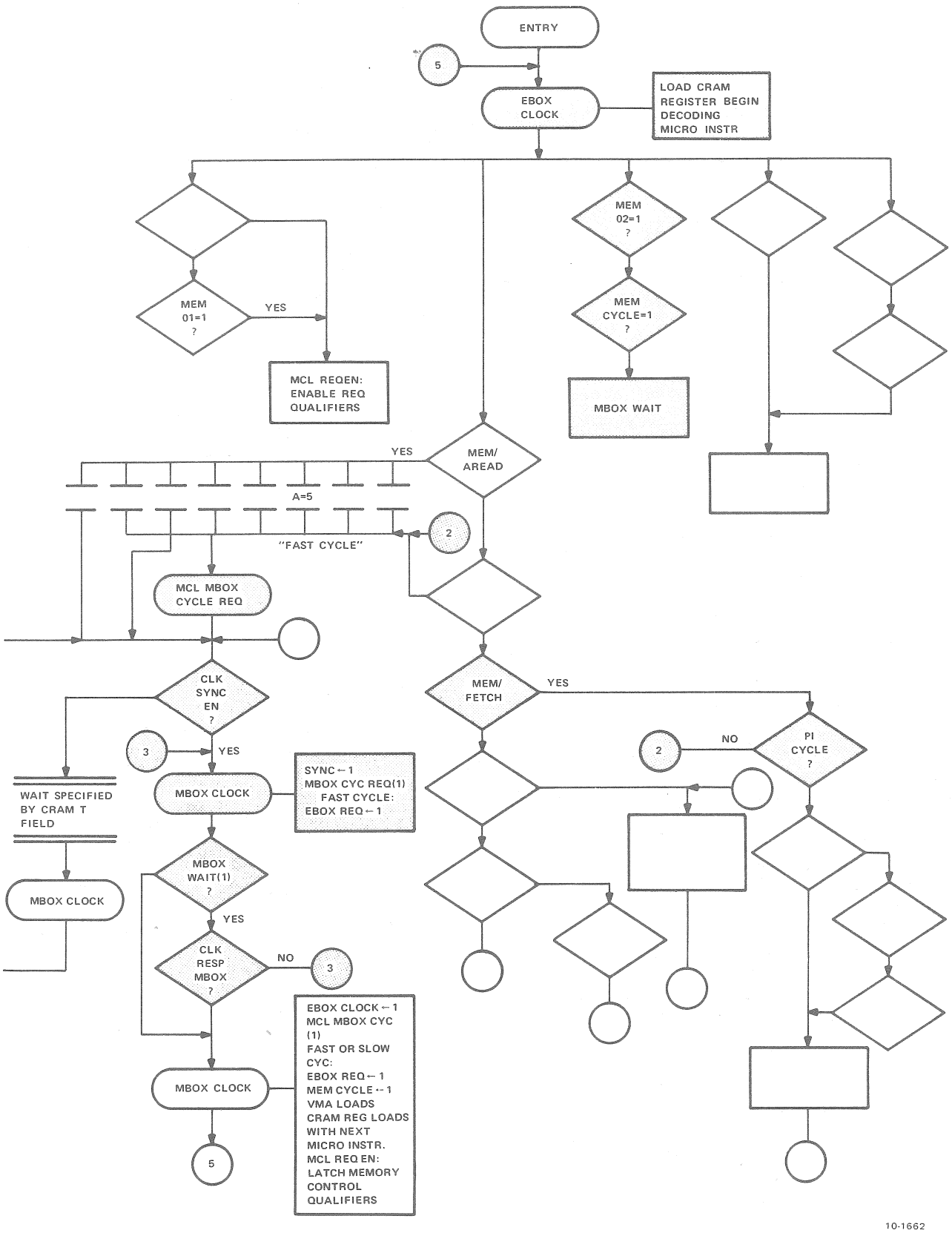


Figure 2-96 Setup Prefetch Waiting for MBox Response

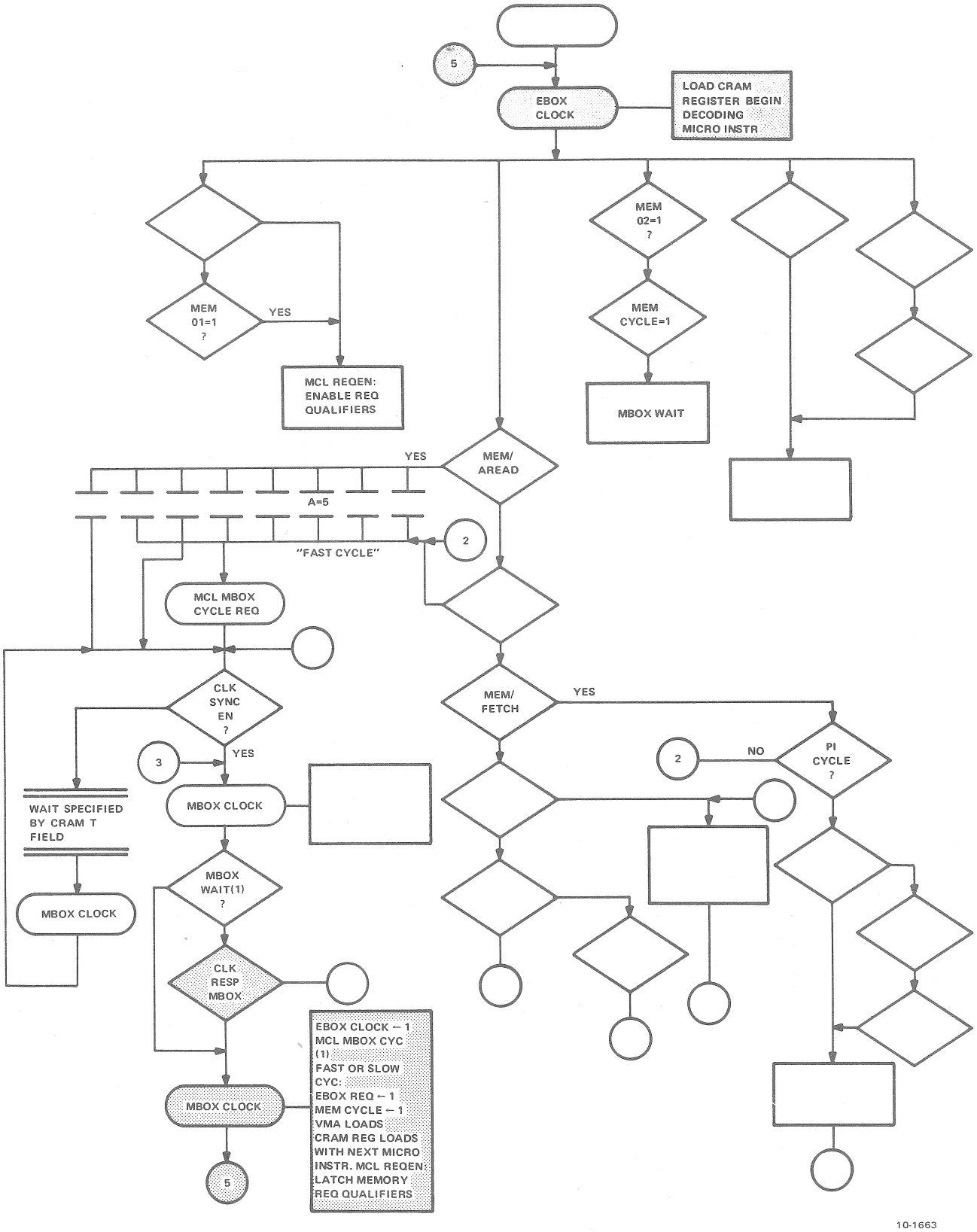


Figure 2-97 Receive MBox Response, End Current MBox Cycle, End Current EBox Cycle, Begin Next EBox Cycle, Begin MBox Cycle

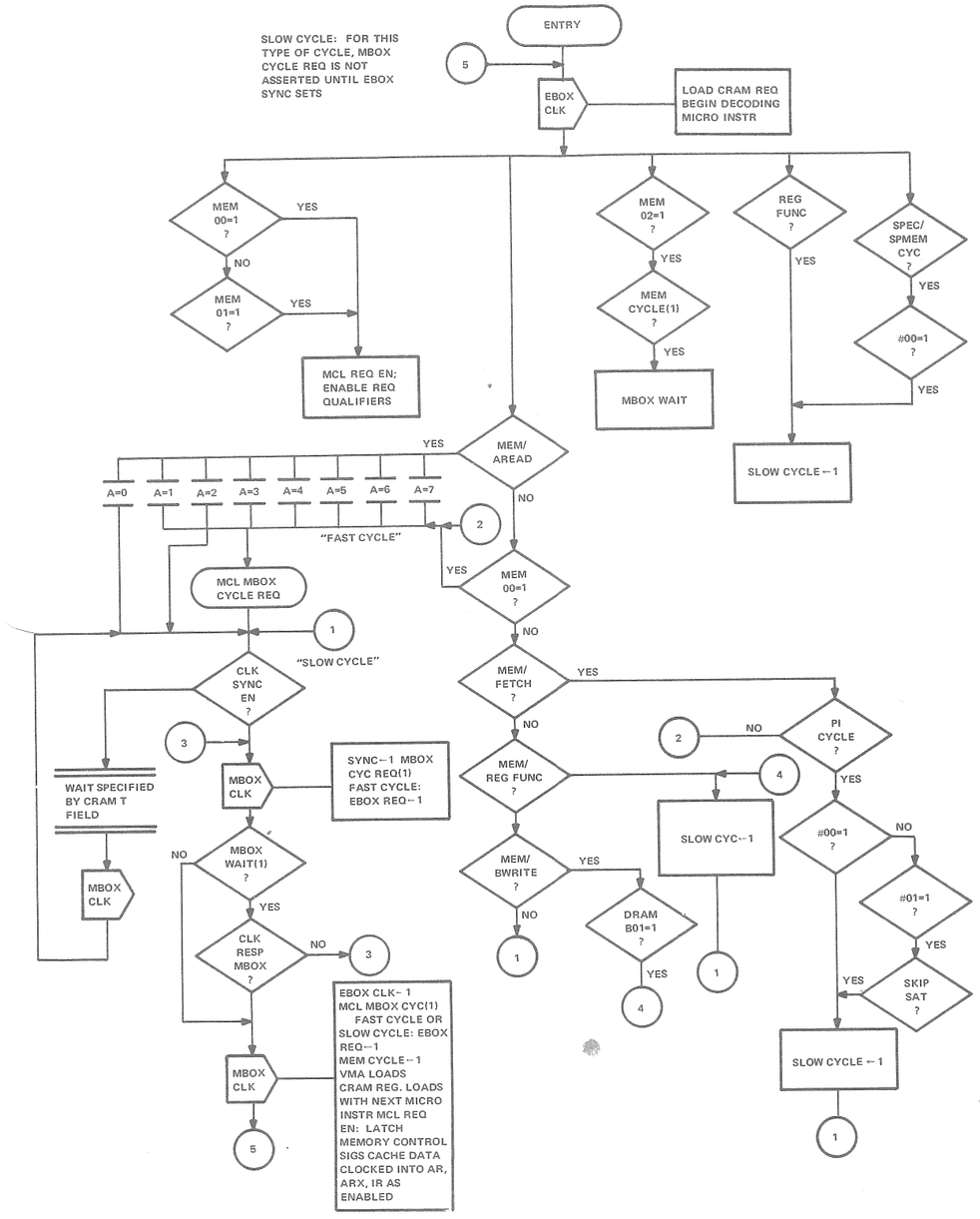


Figure 2-98 General Memory Cycle Control Flow

2.12.1 EBus Signal Lines

The EBus consists of 60 signals. All devices, including the KL10, are connected to these lines in parallel. The bidirectional nature of 36 of the signals permits some information to flow in both directions. These lines are the data lines. The remaining 24 signals are used for control functions. Table 2-19 lists the data transfer signals.

Table 2-19 Data Transfer Signals

Name	Mnemonic	Number of Lines
Data	D(00:35)	36
Controller Select	CS(00:06)	7
Function	F(00:02)	3
Demand	DEM	1
Acknowledge	ACK	1
Transfer	XFER	1

DATA LINES D(00:35) – The 36-data lines transfer information between the EBox and its devices. The most significant bit is bit 00; the least significant bit is bit 35.

CONTROLLER SELECT LINES CS(00:06) – These seven lines select the desired controller for a data transfer. Each controller has a unique select code hardwired on the backplane of the device.

FUNCTION LINES F(00:02) – The function lines specify the type of data transfer (or non data transfer) to take place. Table 2-20 lists the functions implemented.

Table 2-20 Table Data Transfer Commands

F00	F01	F02	Operation
0	0	0	CONO
0	0	1	CONI
0	1	0	DATAO
0	1	1	DATAI

DEMAND (DEM) – This line causes the addressed controller to inspect the CS and F lines and decode their meaning. Upon implementing the specified function, Transfer and Acknowledge are asserted in response and data is placed onto or taken from the EBus as specified by the decoded function.

ACKNOWLEDGE (ACK) – This signal line notifies the I/O bus adapter not to respond to the current operation. If it does not detect ACKNOWLEDGE within some period following assertion of DEMAND, it attempts to perform the transfer. It does not decode the CS lines as the standard KL10 devices do.

TRANSFER – This line is asserted by the selected controller when it is ready to execute the specified function as decoded in F(00:02).

PRIORITY TRANSFER LINES – To perform priority interrupts between the KL10 and its devices, the same basic set of signals is used in a slightly modified form. Table 2-21 lists the necessary signals as they are used.

Table 2-21 Priority Transfer Signals

Name	Mnemonic	Number of Lines
Controller Select	CS(04:06)	3
Controller Select	CS(00:03)	4
Function	F(00:02)	3
Demand	DEM	1
Acknowledge	ACK	1
Transfer	XFER	1

CONTROLLER SEL CS (04:06) – During interrupt arbitration, these three lines represent the octal encode of the interrupting channel.

CONTROLLER SEL CS(00:03) – These four lines specify the controller or device that the EBox is to honor during this interrupt sequence. This is, of course, only a single device or controller, even though several may be interrupting on the same channel. This code also corresponds to the hardwired physical device number of the appropriate controller or device. In CONTROLLER SEL CS(00:03), the range is 0 through 17.

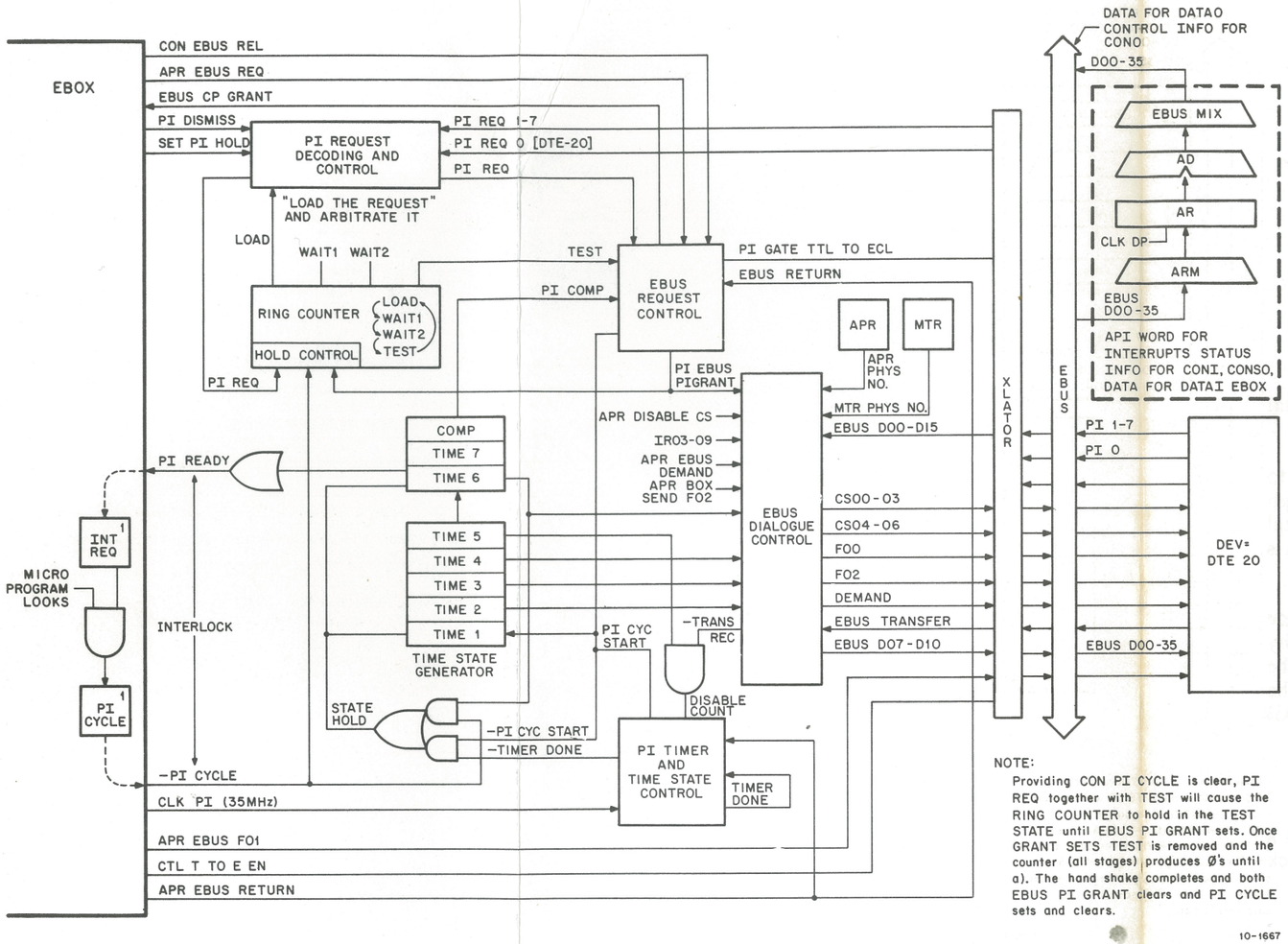
FUNCTION F(00:02) – Two functions are generated during the interrupt dialogue. The first is a code of 4 in F(00:02) and specifies to the interrupting controllers that those controllers being addressed by Channel number in CS(04:06) should send their Physical Controller number by placing them onto the EBus upon sensng DEMAND. The second function is a code of 5 in F(00:02) and specifies to the interrupting controllers or devices that one has been selected. The selected controller will see CS(00:03) as the same number as its physical controller number.

ACKNOWLEDGE (ACK) – Same as for data transfers.

TRANSFER (XFER) – In the case of interrupts, the device selected for service by the EBox places a special function on the EBus data lines D(00:35). Figure 2-99 is the EBus interface functional block diagram. Table 2-22 lists the priority transfer commands.

Table 2-22 Priority Transfer Commands

F00	F01	F02	Operation
1	0	0	PI SERVED
1	0	1	PI ADDRESS IN



NOTE:
Providing CON PI CYCLE is clear, PI REQ together with TEST will cause the RING COUNTER to hold in the TEST STATE until EBUS PI GRANT sets. Once GRANT SETS TEST is removed and the counter (all stages) produces Ø's until a). The hand shake completes and both EBUS PI GRANT clears and PI CYCLE sets and clears.

10-1667

Figure 2-99 EBus Interface Functional Block Diagram

2.12.2 EBus Interface Organization

Referring to Figure 2-99, the interface consists basically of six functional elements. These elements are as follows:

1. PI Request Decoding and Control
2. PI Request Counter and Control
3. EBus Request and Control
4. EBus Dialogue Control
5. PI Timer and Time State Control
6. Time State Generator

The EBus request control and EBus dialogue control are used both by the EBox to carry out I/O transfers and by the PI system in response to an interrupt. During priority interrupt handling, the EBus dialogue is carried out in asynchronous fashion. This operation is controlled by the PI timer and time state control, together with the time state generator.

To obtain the use of the EBus dialogue control, the PI request decoding and control logic must compete with the EBox. No priority exists, and control is obtained on a first-come, first-served basis. Once the EBus has been granted to the EBox, the priority interrupt must wait until the EBox releases the bus.

If the PI system obtains the EBus, the EBox may "demand" the EBus if a page fault occurs (EBus Return).

2.12.3 Interrupt Handling – Loading the Request

Referring to Figure 2-99, there are two cases. The first is an interrupt request from some device on PI 1–7. This may be from any KL10 device, including the APR. The second case is an interrupt from the DTE20 on channel 0. Only the DTE20 may generate channel 0 interrupt requests.

In either case, the PI request enters the PI request decoding and control logic. Here there is a variation in priority. The PI system must be turned on in order for a request on channel 1–7 to be inspected, while interrupts on channel 0 will always be inspected whether the PI system is on or off. The ring counter controls the sampling of PI requests and also determines when a particular request (the highest) is ready to be serviced. In general, "PI LOAD" enables all active requests 0–7 into a request register, providing corresponding PI ON enables are on for channels 1–7.

A programmer may disable interrupts on selected channels by clearing PI ON for each channel he desires to inhibit (note PION0 is in the DTE20). This is done by performing a CONO PI instruction. While the ring counter advances through "WAIT 1" and "WAIT 2," the priority network arbitrates all incoming priority interrupt levels and selects the one with the highest priority (numerically lowest number).

2.12.3.1 Testing the Request – Next, PI TEST is asserted with PI REQ to request the EBus. PI TEST remains true until EBUS PI GRANT sets, giving the EBus to the PI system. Once PI GRANT sets, the PI TEST condition is cleared and the ring counter is disabled until the entire EBus dialogue is carried out and PI CYCLE is "set and cleared" by the microprogram.

2.12.3.2 Requesting the EBus – Setting EBUS PI GRANT begins the EBus dialogue by enabling the assertion of CS 04–06 as the selected channel and F00(4) as function PI SERVED, and also causes the PI timer to begin its sequence by setting PI CYC START.

In general, all external devices that connect to the EBus are presumed to be composed of TTL logic. The PI and EBox logic consist of ECL logic. To temporarily connect these two different types of logic requires use of a logic level shifter. This device is called a translator. The translator must be notified of the conversion direction, TTL to ECL or ECL to TTL. Actually, only the data portion of the EBus is switched from one level to the other. The control signals are connected to fixed level shifting logic. For example, EBUS DEMAND is a unidirectional signal and it is connected to a noncontrollable level shifting gate on the translator module (ECL to TTL).

2.12.3.3 Beginning the Dialogue – The setting of PI EBUS PI GRANT asserts the level PI GATE TTL TO ECL, which causes translation of incoming data from TTL logic levels to ECL logic levels. The PI timer and time state control manipulates the time state generator such that each time state is held for the appropriate length of time. The following relationships exist between the dialogue signals and the time state logic:

CSH 04–06: EBUS PI GRANT
F00: EBUS PI GRANT
DEMAND: sent at T2, T5, and T6
LATCH INCOMING PHYS numbers: T3
CS00–03: T3
F02: T4
EBUS TRANSFER: WAIT AT T5 FOR TRANSFER
PI CYCLE: WAIT AT T6 FOR PI CYCLE TO SET

2.12.3.4 Interlocks and Dialogue Completion – Upon entering T5, the timer is inhibited from incrementing the count until EBUS TRANSFER is received or forced. While waiting, the timer holds the loaded count. As soon as TRANSFER is received and recognized by the PI logic, the timer is once again allowed to count down T5.

Thus, while T5 is counted down, the API word is stabilizing on the input to AR. Next, T6 is entered and here the absence of PI cycle causes STATE HOLD to be asserted. This time the timer may count down and even generate TIMER DONE. If this point is reached and PI CYCLE is still false, the timer loads the count specified by T6 and continues to count while waiting for PI CYCLE to set. The PI board must not begin to service a second interrupt before the microprogram has a chance to look at the first one. Hence, the timer is prevented from entering T7 COMP, until the microprogram has set PI CYCLE. This also enables the ring counter to perform load.

Assuming PI CYCLE sets, the time state generator proceeds through T7 and into complete (COMP). Note that the EBus dialogue control removes DEMAND some time before removing the CS and F lines. This avoids the possibility of misselection of a device. The generation of COMP enables PI EBUS PI GRANT to clear, removing F00 and CS04–06.

2.12.4 Basic Input Output Control

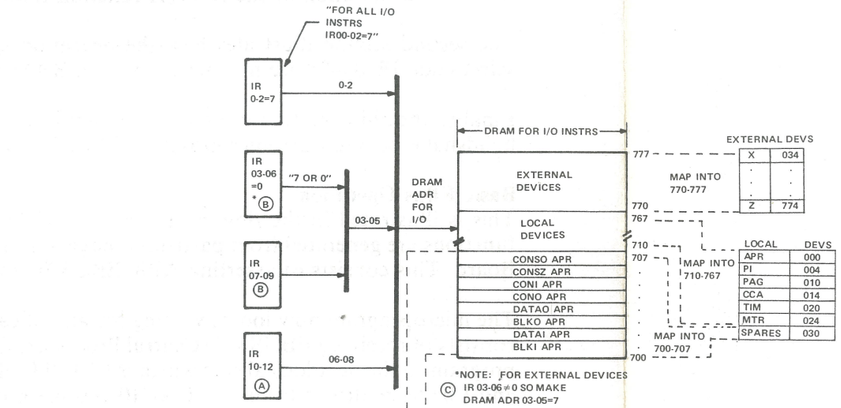
Referring to Figure 2-99, the implementation of I/O operations is similar to interrupt processing, if taken at the point where the EBus is requested. The difference is that instead of a hardware arbitration process taking place, followed by a single request subsequently asking for the EBus, the microprogram I/O handler (part of the executor) requests the EBus. This is accomplished utilizing the condition field function COND/EBUS CTL, together with a particular pattern in the magic number field all in the same microinstruction. Only the resulting signal is indicated on the figure (APR EBUS REQ) but the various other signals are simply formed by combinations of COND/EBUS CTL and an appropriate magic number.

2.12.4.1 Requesting the EBus – The EBus request control treats both an EBox-EBus request (APR EBUS REQ) and a PI EBus request equally. Whichever request is seen by the EBus request control first receives the EBus.

The microprogram is waiting for an indication that it has been granted the EBus. The indication of this condition is EBUS CP GRANT. The microprogram loops, waiting for this signal to become true. Once this occurs, the next step in the operation may be performed.

2.12.4.2 Dialogue Overview – Basically, the EBox decodes bits 10–12 of the instruction to determine which type of I/O operation is to be performed. Eight possible combinations exist; these are indicated in Figure 2-100 at the bottom left. The logical mapping of I/O op code into appropriate DRAM addresses is also illustrated in Figure 2-100.

	OPERATION	COND/ EBUS CTL	MAGIC # FIELD 0 1 2 3 4 5 6 7 8	FUNCTION	# 0 APR EBUS RETURN	# 1 APR EBUS REQ	# 2 CON EBUS REL	# 3 APR EBUS DEMAND	# 4 HOLD STATE	# 5 (0) # 5 (1) SELECTION CONTROL	# 6 # 6 APR EBOX DISABLE CS	– APR AC10 (1) # 7 APR EBUS F01	APR F02 EN (1) # 8 APR EBOX SEND F02	CS 00–06
	BASIC EBUS OPERATION USED BY ALL I/O INSTR	YES	0 1 0 0 0 0 0 0 0	REQUEST EBUS		X								
		YES	0 0 0 0 1 1 0 0 0	I/O INIT					X	X		X	X	IR 03–09
		YES	0 0 0 1 1 0 0 0 0	SET EBUS DEMAND				X	X			Holding	Holding	Holding
		YES	0 0 0 0 1 0 0 0 0	CLEAR EBUS DEMAND					X			Holding	Holding	Holding
		YES	0 0 1 0 0 0 0 0 0	RELEASE EBUS			X							
MTR INT, INTERNAL DEVICE CONTROL, PAGE FAIL HANDLER, READ EBUS REG	USED TO OBTAIN ONLY THE ECL SIDE OF EBUS. THEN GIVE BACK LATER	YES	1 0 0 0 0 0 0 0 0	GRAB ECL EBUS	X									
		YES	0 0 0 0 0 0 0 0 0	RELEASE ECL EBUS										
	DEPOSIT, BYTE XFER OR PI DATAO, FOLLOWED BY BASIC EBUS OPERATION	YES	0 0 0 0 1 0 1 1 0	SET DATAO					X		X	X		0'S
	EXAMINE, BYTE XFER OR PI DATAI, FOLLOWED BY BASIC EBUS OPERATION	YES	0 0 0 0 1 0 1 0 0	SET DATAI					X		X			0'S



OPERATION	AC10	AC11	AC12	F00	F01	F02
CONO	1	0	0	0	0	0
CONI	1	0	1	0	0	1
DATAO	0*	1	1	0	1*	0
DATAI	0*	0	1	0	1*	1
BLKO	0*	1	0	0	1*	0
BLKI	0*	0	0	0	1*	1
CONSO	1	1	1	0	0	1
CONSZ	1	1	0	0	0	1

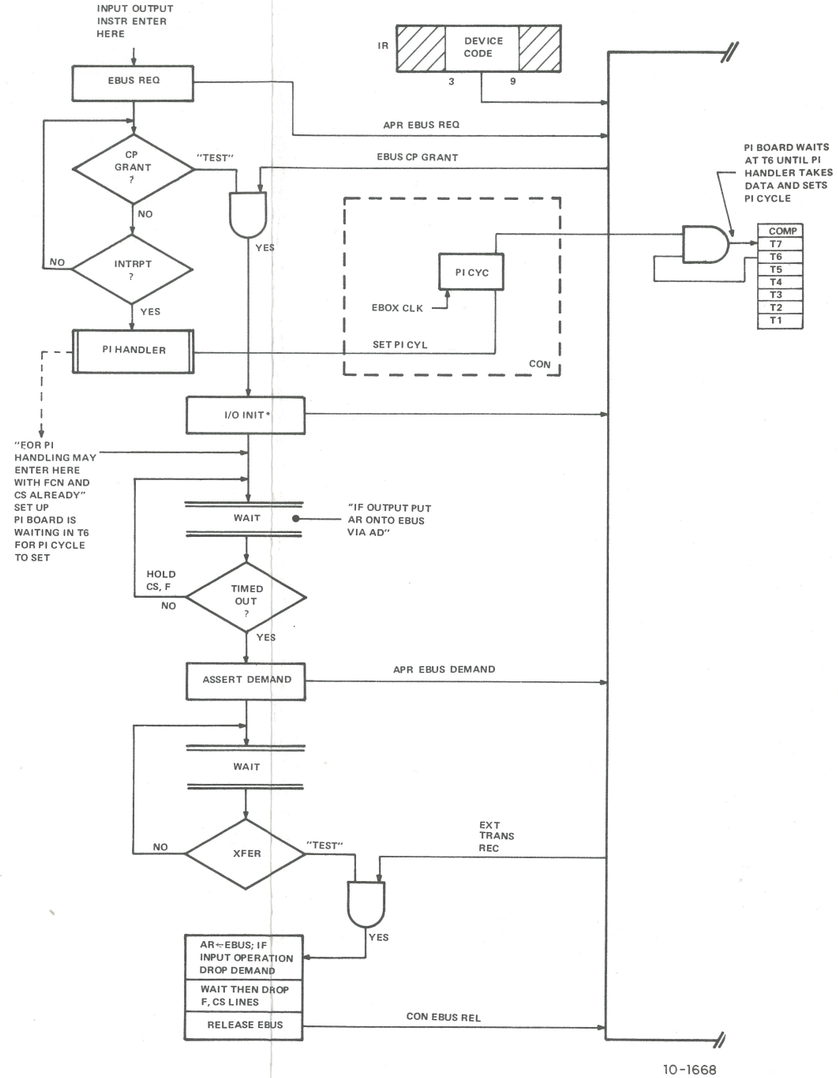


Figure 2-100 EBus Control Functions

The dispatch to the proper operation is obtained by mapping bits 10–12 into DRAM ADR 06–08, while the device address 3–6 is mapped into DRAM ADR bits 03-05. Thus, for example, a DATAI APR with op code 701 is mapped into DRAM address 701. Similarly, BLKO PAG, with op code 722 is mapped into DRAM address 722. This is device 010_8; therefore, the type of operation performed is determined in advance and the DRAM jump address is coded to cause a jump to the appropriate group of microinstructions. The device select code is in bits 3–9 of IR and must be used to address the device. This addressing is accomplished by converting 3–9 to CS00–06 in the proper form. The function is controlled by the combination of two EBox control signals, APR EBOX SEND F02 and APR EBUS F01. With these two signals, all combinations of input and output operations may be performed as indicated on Figure 2-100. Notice that EBus F00 is not used for any of the operations. This signal is generated during priority interrupt dialogue for the function PI SERVED (Function 4) and for PI ADDRESS IN (Function 5).

2.12.4.3 Functional Breakdown – Figure 2-100 is essentially composed of three sections. The first is a breakdown of the EBus microcode operations into four basic suboperations as follows:

1. Basic EBus operation as used by all I/O instructions.
2. ECL EBus acquisition and subsequent release
3. Generation of the DATAO function followed by the basic EBUS operation
4. Generation of the DATAI function followed by the basic EBus operation

The second section illustrates how the operation specified in IR 10–12 and a portion of the device select code IR 03–05 are mapped into the DRAM words that pertain to I/O operations.

Finally, the third section consists of a simplified flow of the basic EBus operation, including the handshake between the microprogram EBus driver and the PI Board.

Basic EBus Operation
This is illustrated in the flow on the bottom right of Figure 2-100. Five basic COND/EBUS CTL functions are generated from particular magic number bits. The first is to request the EBus from the PI Board. This consists of asserting APR EBUS REQ.

The microprogram now loops, waiting for an indication that it has obtained the EBus. The indication consists of receiving EBUS CP (Central Processor) GRANT from the PI Board. This moves the microprogram to the next logical step which is IO INIT. Here magic number 5 enables the function lines F01 and F02 to be driven from -APR AC10 and APR F02 EN, respectively. The table of I/O operations given at the bottom left on Figure 2-100 shows that F01 is true whenever AC10 is false. This is true for DATAO, DATAI, BLKO, and BLKI. Conversely, F02 is true whenever AC10 is true, or both AC10 and AC11 are false.

Magic number 4 is used to latch the particular function (HOLD IT). Note that during the IO INIT period, IR 03–09 is passed to the PI Board to become CS00–06. A fixed delay is generated by the microcode at this time to allow the controller select lines to set up at the device.

Next, SET EBUS DEMAND is issued, while holding the previous function lines F01 and F02 as previously set up. Once again, the microprogram waits a predetermined period. The waiting is controlled by the time field and the number of successive microinstructions issued. Thus, two successive microinstructions with T = 5 is approximately 300 ns.

Now the microprogram loops, waiting for TRANSFER from the device. This signal indicates that the device has completed the specified transaction and has either taken or transmitted status, data, or control over the EBus. At this time, if the operation was CONSO, CONSZ, CONI, BLKI or DATAI, the EBus is loaded into AR. If the operation was CONO, BLKO or DATAO, during IO INIT the AD is enabled to the EBus. The AD contains the contents of AR.

Finally, DEMAND is removed by issuing the function CLR EBUS DEMAND. Notice that number 4 holds the function lines up. It is necessary to remove DEMAND first while still maintaining the function and CS lines in order to prevent a spurious misselection. Now the function and CS lines are dropped and the EBus is relinquished by issuing RELEASE EBUS. This action causes EBUS CP GRANT to clear.

PI Handler and EBus Operation
Once again referring to the flow on Figure 2-100, note that after issuing EBUS REQUEST and while testing for CP GRANT, an interrupt is tested for. If an interrupt is pending, the PI Handler is entered. This means that EBUS PI GRANT was set when EBUS REQUEST was issued and EBUS CP GRANT could not set anyway.

The PI Board has negotiated with the device for the API function word that is now on the input to AR. The PI Board is holding in T6, waiting for PI cycle to be set.

Examine, Deposit, or Byte transfers requested by the 10-11 interface require separate control of the controller select and function lines. For these cases, SET DATAO or SET DATAI is issued independently. Then the EBus routine is entered at the point where the CS and F lines are setting up. If the operation is DATAO of T011 transfer, the AR is placed onto the EBus via AD. The remainder of the EBus operation is identical to that for basic EBus operation.

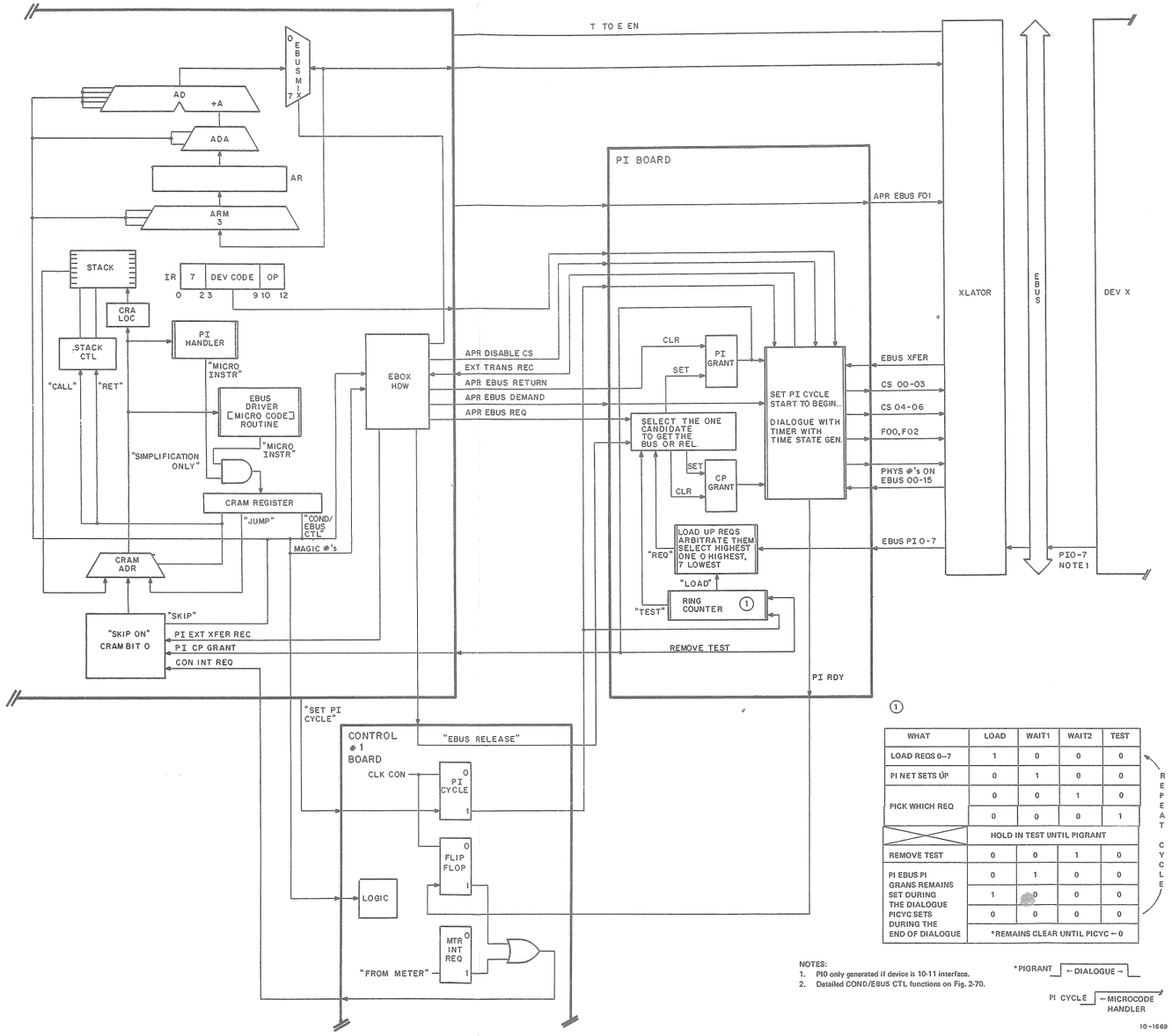
ECL EBus Acquisition – At various times, the ECL portion of the EBus is required for some form of transfer. Some examples of this requirement would be processing interrupts for internal devices such as APR, PI SYSTEM, or TIM. Also, performing I/O instructions involving these devices would require the use of the ECL EBus. A second example is the case of page fault handling in the microcode. At some time, the MBox-EBus register must be read over the EBus into AR. Thus, the ECL EBus is necessary for this operation. The function necessary to acquire the ECL EBus is COND/EBUS CTL with magic number bit 0 set. This actually takes the EBus away from the PI system. It does not abort the PI operation (if any) but merely causes it to be dealyed. The signal APR EBUS RETURN causes the PI timer and time state generator to HOLD and it clears EBUS PI GRANT. The ECL EBus is relinquished by issuing RELEASE ECL EBUS, which takes away APR EBUS RETURN. Now the PI may continue from the point at which it was held.

2.12.5 PI and EBus to Microcode Interface

Figures 2-101, and 2-102 are concerned with the interaction of the PI Board and certain other EBox related hardware with the PI Handler and EBus Driver. Both of these handlers are microprograms. Figure 2-101, illustrates the basic signal interfacing between fucntional elements of the PI Board, Control Number 1 Board, and various EBox hardware used during EBus transactions with the Microcode PI Handler and EBus Driver. Figure 2-102 generally relates the PI Handler and EBus Driver functions to the PI Board hardware for given operations. Figure 2-103 is supplied to support functional descriptions to follow.

2.12.5.1 Sensing the Interrupt – Initially, assume that the PI Board is enabled and idle. Two devices (DSK) assert interrupts on the same priority interrupt channel; DSKA on channel 5 and DSKB on channel 5. Thus, based on the fixed physical number scheme, the range of physical numbers is 0–7. Further, assume that DSKA is wired to be physical number 1 and that DSKB is wired to be physical number 7, and that DSKA is the device selected.

Referring to Figure 2-101, PI Level 5 is received from both devices and is loaded into PI Request register 5 for arbitration. Because both DSKs are interrupting on the same channel, the PI Network need only check those channels holding interrupts. If none is holding on 5 through 1 (0 is DTE20 and never holds), then channel 5 is selected. The next phase begins by asserting REQ to obtain use of the EBus.



①

WHAT	LOAD	WAIT1	WAIT2	TEST
LOAD REQS 0–7	1	0	0	0
PI NET SETS UP	0	1	0	0
PICK WHICH REQ	0	0	1	0
	0	0	0	1
	HOLD IN TEST UNTIL PIGRANT			
REMOVE TEST	0	0	1	0
PI EBUS PI GRANS REMAINS SET DURING THE DIALOGUE PICYC SETS DURING THE END OF DIALOGUE	0	1	0	0
	1	0	0	0
	0	0	0	0
	*REMAINS CLEAR UNTIL PICYC ← 0			

REPEAT CYCLE

NOTES:
1. PI0 only generated if device is 10-11 interface.
2. Detailed COND/EBUS CTL functions on Fig. 2-70.

*PIGRANT ← DIALOGUE →
PI CYCLE ← MICROCODE HANDLER

10-1669

Figure 2-101 EBox PI Board to Microcode Interface

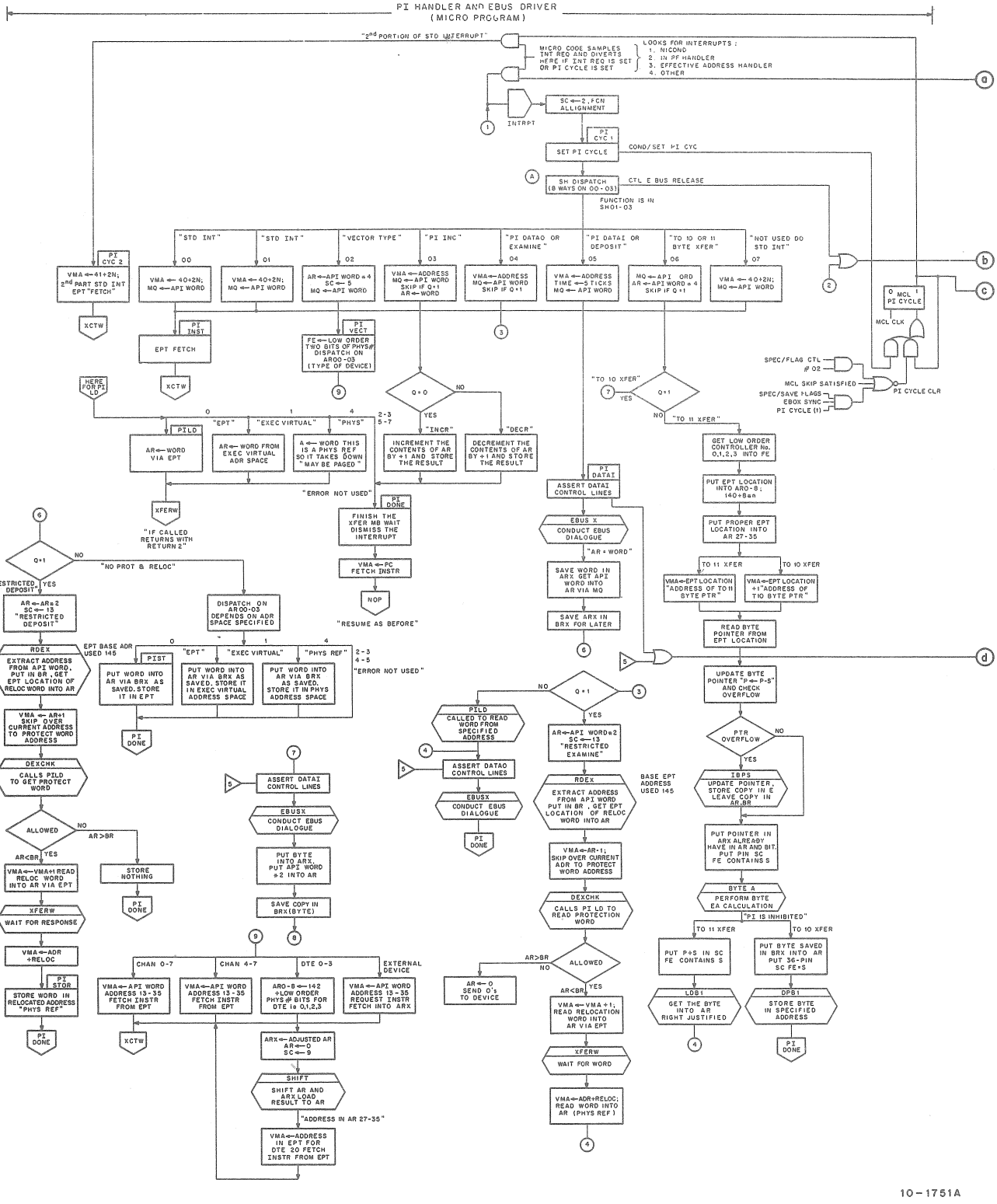


Figure 2-102 EBus Control Hybrid Flow (Sheet 1 of 2)

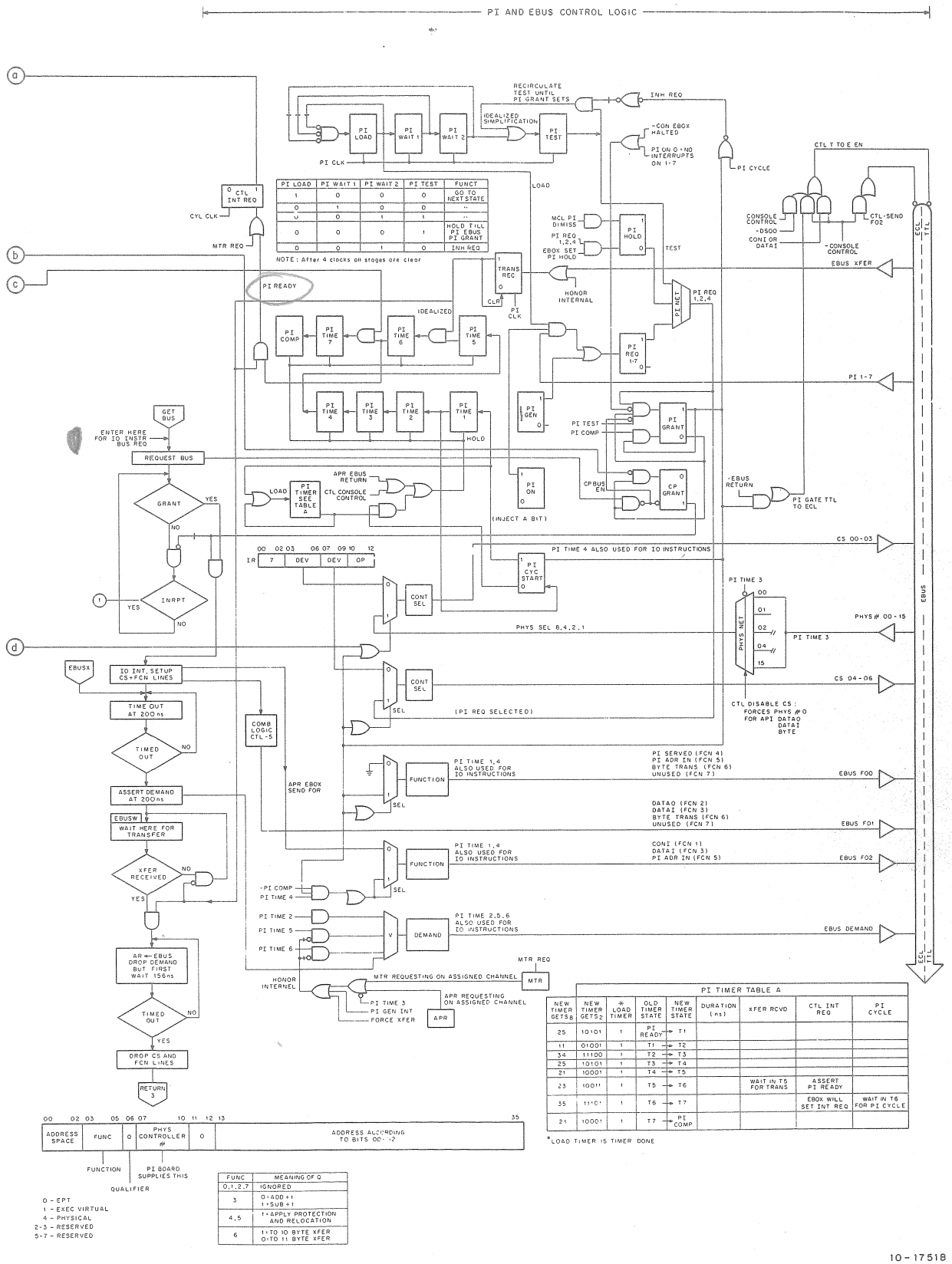


Figure 2-102 EBus Control Hybrid Flow (Sheet 2 of 2)

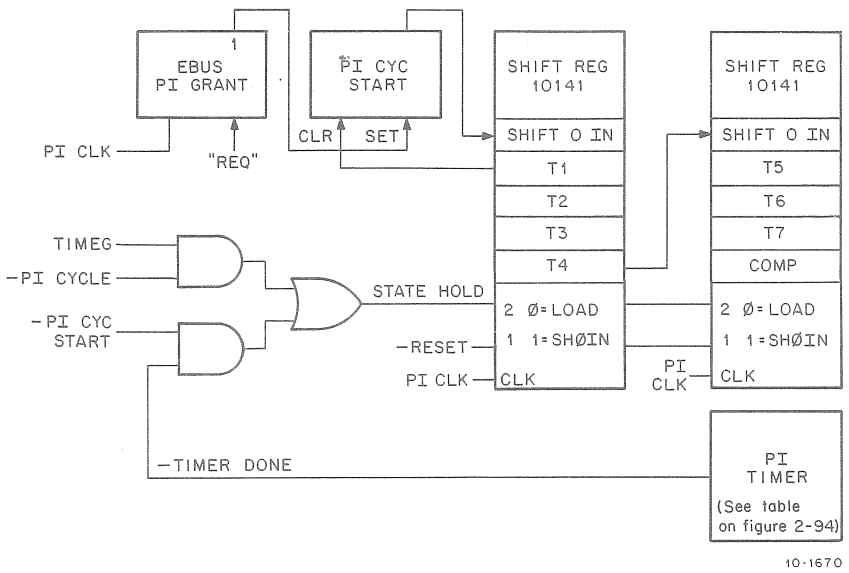


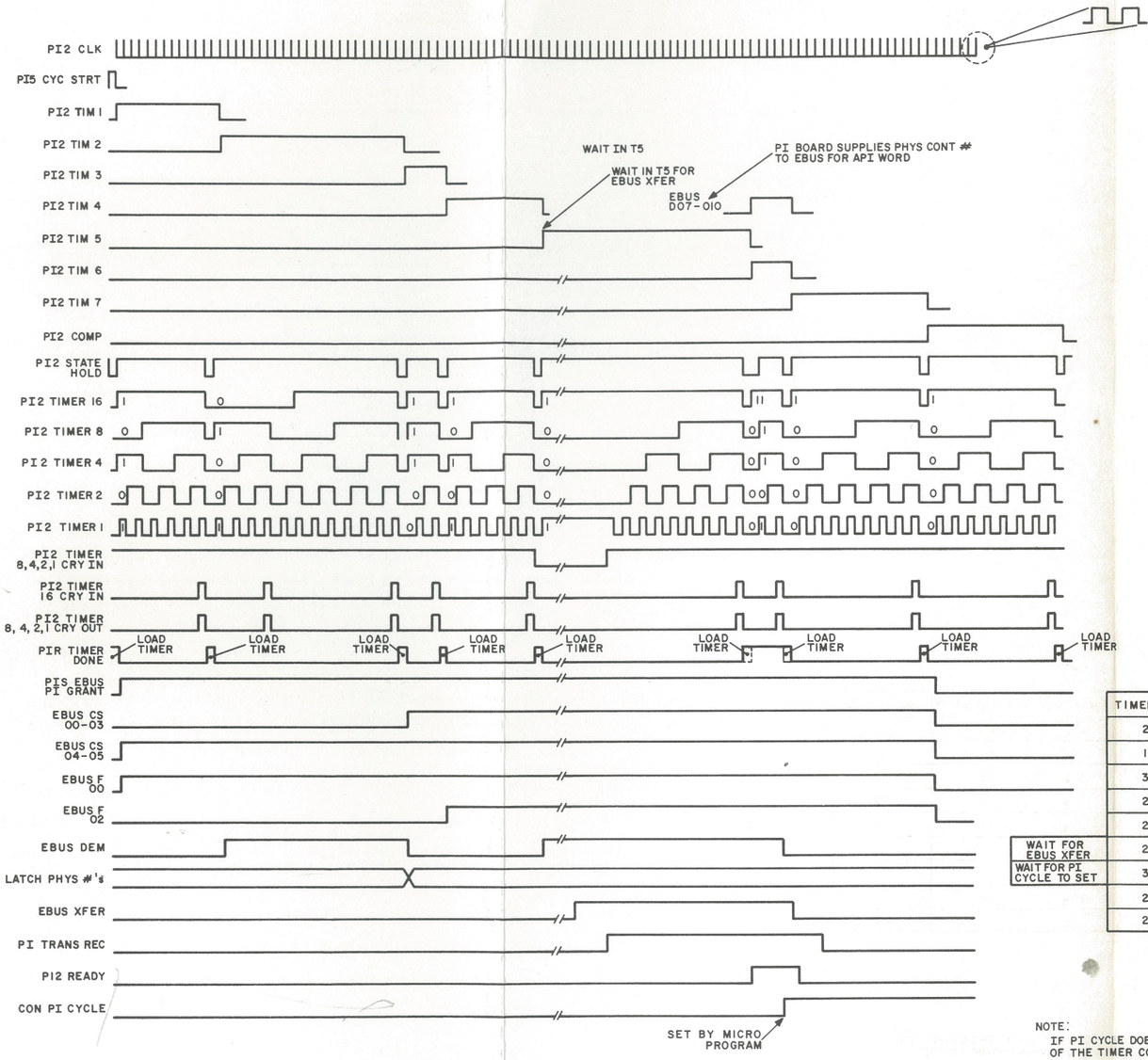
Figure 2-103 Time State Generator Control

2.12.5.2 Requesting the EBus – To obtain use of the EBus, the PI logic must set EBUS PI GRANT. This is illustrated on Figure 2-102. Note that the following requirements must be fulfilled to set EBUS PI GRANT:

1. PI test must come up.
2. REQ must be true (PI 4, 2, 1 = some selection).
3. The Ebox may not be halted or there are no interrupts selected on 1–7.
4. EBUS PI GRANT is currently clear.
5. The PI Board is not trying to set CP Grant.

If all five conditions are satisfied, EBUS PI GRANT sets. If the conditions are not currently satisfied, the interrupt waits.

2.12.5.3 Beginning the Dialogue – At this time, several events take place. The setting of EBUS PI GRANT enables setting of cycle state, which begins the dialogue. In addition, the PI Timer (see the table on Figure 2-104) is loaded with 25_8, which defines the duration of the time state entered, in this case time 1. The time states are used to direct the EBus dialogue from beginning to completion. EBUS PI GRANT forces F00 to a 1. This function (4) is PI served and is issued together with CS 04–06, which are encoded to be the selected channel (5). The interrupting devices (in this example two DSKs) decode the function lines F00–02, together with the controller select lines CS 04–06. The PI timer counts from 25_8 to 37_8 then generates TIMER DONE. The devices have now had sufficient time to decode the CS and F lines so the next phase of the dialogue begins. The timer is now loaded with 11_8, Time T1 is removed and T2 is entered.



TIMER GETS	CURRENT STATE	NEW STATE
25_8	PI CYC START	T1
11_8	T1	T2
34_8	T2	T3
25_8	T3	T4
21_8	T4	T5
20_8	T5	T6
35_8	T6	T6
20_8	T6	T7
20_8	T7	COMP

WAIT FOR EBUS XFER

WAIT FOR PI CYCLE TO SET

NOTE:
IF PI CYCLE DOES NOT SET BY THE COMPLETION OF THE TIMER COUNTDOWN DURING T6 THE PI BOARD HOLDS AT T6 UNTIL PI CYCLE SETS

10-1754

Figure 2-104 PI Timing

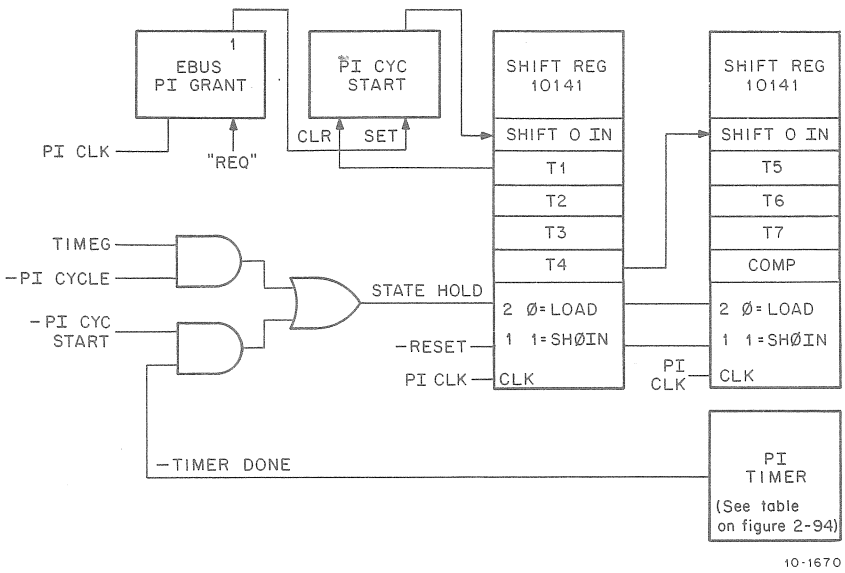


Figure 2-103 Time State Generator Control

2.12.5.2 Requesting the EBus – To obtain use of the EBus, the PI logic must set EBUS PI GRANT. This is illustrated on Figure 2-102. Note that the following requirements must be fulfilled to set EBUS PI GRANT:

1. PI test must come up.
2. REQ must be true (PI 4, 2, 1 = some selection).
3. The Ebox may not be halted or there are no interrupts selected on 1–7.
4. EBUS PI GRANT is currently clear.
5. The PI Board is not trying to set CP Grant.

If all five conditions are satisfied, EBUS PI GRANT sets. If the conditions are not currently satisfied, the interrupt waits.

2.12.5.3 Beginning the Dialogue – At this time, several events take place. The setting of EBUS PI GRANT enables setting of cycle state, which begins the dialogue. In addition, the PI Timer (see the table on Figure 2-104) is loaded with 25_8, which defines the duration of the time state entered, in this case time 1. The time states are used to direct the EBus dialogue from beginning to completion. EBUS PI GRANT forces F00 to a 1. This function (4) is PI served and is issued together with CS 04–06, which are encoded to be the selected channel (5). The interrupting devices (in this example two DSKs) decode the function lines F00–02, together with the controller select lines CS 04–06. The PI timer counts from 25_8 to 37_8 then generates TIMER DONE. The devices have now had sufficient time to decode the CS and F lines so the next phase of the dialogue begins. The timer is now loaded with 11_8, Time T1 is removed and T2 is entered.

Time 2 enables EBUS DEMAND. Note that the function PI served and controller select lines are maintained. The DSKs are commanded to place their "hardwired" physical numbers onto the EBus, bit 1 for physical number 1 and bit 7 for number 7. Referring to Figure 2-103, DEMAND is held up through Time 2 and then removed while the F and CS lines are maintained. It is good procedure to remove the DEMAND signal before attemping to change the function lines; this avoids any spurious misselection. The timer is next loaded with 25_8 and T3 (a brief time state) is entered. Here, two functions are performed:

1. The physical numbers, by now on the inputs to a register on the PI Board, are clocked into that register for arbitration.

2. The PI Board is timing out a period of time until it is safe to change the function lines.

The next part of the dialogue is begun when Time 4 is entered.

Here, F00 and F02(5) are asserted; CS00–03 reflect the encoded physical number that has highest priority (#01) and CS04–06 still reflect the PI channel being served. When Time 4 is removed and T5 sets, DEMAND is asserted once again. This time DSKA is selected as the DSK to be serviced. DEMAND commands DSKA to place its API word on the EBus and to assert EBUS TRANSFER to the EBox. The PI Board waits in Time 5 until TRANSFER is received, or forced. If, for example, the interrupting device (DSKA) can respond to most of the dialogue but cannot send EBUS TRANSFER, the PI Board waits. If TRANSFER is not forthcoming, TRANSFER is forced and the EBus (which contains zeros) is treated as an API function of 0. This ultimately causes a 40 + 2n interrupt on the interrupting channel. The DSKs service routine must then decide what went wrong. Assume that the DSKs succeed in placing the appropriate API function word on the EBus and generate TRANSFER. The timer is loaded with 35_8 and Time 6 is entered where PI READY is asserted. At this point, the PI Board is notifying the EBox microprogram that the API word is currently on the AR mixer inputs.

2.12.5.4 Terminating the Dialogue – With the assertion of PI READY, the PI Board waits in Time 6 until the PI Handler (microcode handler) looks at the interrupt. PI READY enables INT REQ to set in the EBox and when the PI Handler detects this, it sets PI CYCLE. Now the timer continues by entering Time 7, drops DEMAND and finally enters COMP, where the CS and FUNC lines, together with EBUS PI GRANT, are removed. This completes the PI Boards dialogue.

2.12.5.5 Entry to the PI Handler – Referring to Figure 2-102, the handler is entered at symbolic location INTRPT, with the API word loading into AR, and PI CYCLE not yet set. Thus, the PI Board is at this time in Time 6, waiting for PI CYCLE to be set. The shift counter is loaded with 2, in order to enable the API word in AR to be shifted left two positions, bringing the function code in bits 03–05 into bits 01–03. PI CYCLE is set and then a shift dispatch is given; depending upon the function 0–7, the dispatch is to one of eight routines within the main handler.

Function 00 – STD INTERRUPT NO TRANSFER
The word is buffered in MQ. The VMA is loaded with the appropriate 40 + 2n address. This address is implemented via the SCD TRAP mixer (refer to Figure 2-60) and derived from number with PI 4, 2, 1. PI 4, 2, 1 is simply the octal equivalent of the channel on which the interrupt was taken. Thus, the instruction is fetched from 40 + (2 × 5) in the example cited in Subsection 2.8.5.3. This yields an address in VMA of 0000050.

The program branches to Execute Wait (XCTW) where the microprogram waits for the instruction fetched to load into AR. This instruction should be a "JSR," which saves the flags and PC and then enters a subroutine in main memory to deal with the situation. The performing of a JSR causes SPEC/SAVE flags, which clear PI cycle and set PI HOLD, to hold the interrupt.

Function 01 – STD INTERRUPT KI10, KA10 Device via I/O Bus Adapter or KL10 Device via EBus
The implementation of this function is identical to that for Function 00. The difference between the function codes is that Function 01 is a premeditated request for a "STD INTERRUPT," where Function 00 is a bus failure condition.

Function 02 – VECTOR INTERRUPT
The word is buffered in MQ. The API word contains an address in bits 13–35 and an address space qualifier in bits 0–2. The address is loaded into VMA. Now a dispatch is given on AR00–03. The API word format is presented on Figure 2-102. Note that only three address spaces may currently be specified:

0 – EXEC PROCESS TABLE (EPT)
1 – EXEC VIRTUAL ADDRESS SPACE
4 – PHYSICAL ADDRESS

A routine is called for the storage operation PILD (illustrated in Figure 2-102).

Fetching from EPT – T
VMA bits 27–35 receive the AR bits 27–35 via AD. The EBox makes an EPT reference. Referring to Figure 2-83, the qualifiers asserted to the MBox are as follows:

EBOX REQUEST
VMA EPT
PAGE UEBR REF

The hardware normally looks at a combination of SPEC/SP MEM cycle with magic number and user enable to select either VMA EPT or UPT, depending on the state of user. In this case, however, user must be disabled to enable a direct reference to EPT. The AR is loaded with the instruction fetched from CPT. This instruction is either the first of a series of instructions in a service routine or an instruction directing entry to a service routine. As with 40 + 2n interrupt instructions, the instruction should be a JSR to save the flags and PC. By performing a JSR, SPEC/SAVE flags clear PI CYCLE and set PI HOLD on the PI Board. This holds the interrupt.

Fetching from EXEC Virtual Address Space
The API word is buffered in the MQ. For this case, the address in bits 13–35 of the API word is a complete virtual address. In fetching from EPT, only bits 27–35 of the address in bits 13–35 contain address information. The MBox appended a base address (EBR) to this 9-bit address. Here the request qualifiers are as follows:

EBOX REQUEST
EBOX READ

The MBox translates the address and supplies the instruction that loads into AR. Once again, transfer is to XCTW, to wait until the instruction actually loads into AR. Then the instruction is performed as with the previous EPT reference.

Fetching from Physical Memory
Here, the address contained in the API word bits 13–35, contains a physical address in bits 22–35 while bits 13–17 are clear. To cause a physical reference to occur, the magic number field is coded with number 08 set and this, together with SPEC/SP MEM cycle, inhibits the qualifier MAY BE PAGED. If this signal is not present during EBus request, the MBox does not page the address. The instruction loads into AR as before and then performs. Once again, SPEC/SAVE flags clears PI CYCLE and sets PI HOLD.

Function 03 – PI INCREMENT

This function causes a word in the specified address (API word bits 13–35) to be incremented or decremented as a function of the Q BIT in the API word. If Q = 1, the function is decremented; otherwise, it specifies increment. Referring to Figure 2-102, the API word is buffered in MQ and Q is tested. If Q = 0, the contents of the address specified in the API word 13–35 are fetched and incremented. The incremented word is then stored back in the same address and an instruction fetch is performed from PC. This contains the interrupted program. Note that the microcode must set PI HOLD in order to hold an interrupt on the PI Board. This is done when the 40 + 2n or vector function fetches and performs a JSR or similar instruction. Here, after completion of the storage operation, the interrupt is dismissed and PI CYCLE is cleared. PI CYCLE is cleared with SPEC/FLG CTL and number 02.

Function 04 – PI DATAO or EXAMINE

The 10-11 interface may perform an Examine function to either core memory or fast memory. In addition, the address supplied in the API word may be a relocated address or not depending on the Q BIT in the API word. Associated with the Examine operation are two words of information for each 10-11 interface in the system. These word pairs are in predefined areas in the EPT. One word of the pair is a protection constant, which limits the address of the virtual address sent in the API word. The number of pages specified in bits 13–26 may be less than or equal to the value of the protection constant, but not greater than that value. The microprogram utilizes the low-order 2 bits of the physical number supplied to the API word (bits 7–10) and forms an address 140 + 8n, where n is the low-order 2 bits of the physical number for the interrupting 10-11 interface. The physical numbers are hardwired as 10_8–13_8. This gives low-order 0, 1, 2, or 3. The FPT location thus obtained is accessed for the protection constant and the comparison is made. If a violation occurs (protection violation), a word of zeros is transmitted to the 10-11 interface via the EBus. If no violation occurs, the relocation word is fetched from EPT and added to the address supplied in 13–26 of the API word. This address is now treated as a physical reference and it is not paged. The word is obtained and transmitted via DATAO function to the 10-11 interface. Upon completion of the EBus dialogue, the PI CYCLE is cleared. Note that for the 10-11 interface Examine function, the interrupt occurs on channel 0.

This channel is implemented solely to enable the 10-11 interface to utilize the PI facility at any time, whether it is on or off for DMA type transfers. No HOLD flip-flop exists for PI0, so clearing PI CYCLE effectively releases the PI0 interrupt. Devices other than the 10-11 interface may utilize this operation under the classification PI DATAO. Two differences in its implementation from that of Examine exist. First, no protection or relocation is applied and hence no violation can occur. A page fault, however, can occur. If this occurs, the PF Handler sets IOPF and transfers control to the operating system. The second difference is that other devices interrupt on channels in the range of 1–7. Once again, holding the interrupt for this one time transfer is unnecessary and only clearing PI CYCLE is necessary to release the PI Board. Other than these differences, the operation is identical to Examine.

Function 05 – PI DATAO or DEPOSIT

In terms of the 10-11 interface, this operation is the reverse of Examine, except that after the 10-11 interface sends the API function (which contains the address), the EBox must perform a DATAI function to obtain the 36-bit word to deposit in the specified address. A second difference is that if a violation occurs, after performing the protection check a violation occurs, no word is stored in the specified address. With these exceptions, the operation is basically the same from the point where the 36-bit word is obtained from the 10-11 interface to the completion of the operation.

Function 06 – PI BYTE TRANSFER
This function can only be carried out between a 10-11 interface and the EBox. This function is initiated on PI channel 0 as are Examine and Deposit. The transfer is part of either a T011 or T010 byte transfer occurring in the 10-11 interface. The information being transferred is either a byte right-justified in EBus bits 28–35, or a word right-justified in EBus bits 20–35. The API word specifies whether the transfer is T010 or T011 by the state of the Q BIT. If Q = 1, the transfer is T010; otherwise, it is a T011 transfer. In addition, the PI Board is supplying the physical number in bits 07–10 of the EBus while the API word is present. The other portions of the word 0–2, 11–35 are ignored.

T010 Byte Pointer Fetch, Byte Read, and XFER
The low-order two bits of the physical controller number 0, 1, 2, or 3 are obtained and combined with EPT base location 14X to form the EPT location of the T011 byte pointer. Next, the byte pointer is obtained from the EPT and updated. The pointer is a standard KL10 byte pointer. The microcode for load byte instructions is used for the pointer update. Note that the byte pointer may specify indirection and/or indexing. Once the effective address has been calculated, the updated byte pointer is stored back in its slot in EPT and the byte is obtained by performing an EBox request. Finally, the byte now in AR is transferred via the EBus (DATAO) to the 10-11 interface and PI CYCLE is cleared.

T010 Byte Pointer Fetch, Byte Transfer and Storage
The byte is initially requested by issuing a DATAI to the 10-11 interface. The byte is then picked up via EBus 28–35 and loaded into ARX and into BRX. Next, the low-order two bits of the physical controller number 0, 1, 2, or 3 are obtained and combined with EPT base location 14X to form the EPT location of the T010 byte pointer. The byte pointer is obtained from the EPT and updated. The pointer is a standard KL10 byte pointer. For the T011 XFER, the microcode for deposit byte is used for the pointer update and, as with the byte pointer for T011 XFER, may specify indirection and/or indexing. Once the effective address has been calculated, the updated byte pointer is stored back in its slot in the EPT and the byte is stored in the pointer's effective address. Finally, PI CYCLE is cleared and this terminates the operation.

Function 07 – UNASSIGNED
This function is unassigned and currently behaves the same as function 00.