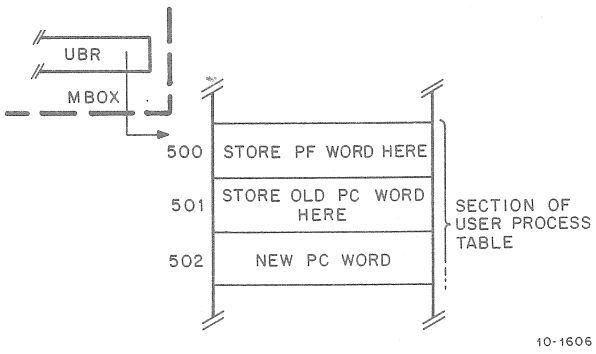


Figure 2-36 Process Table PF Location

2.4.2 Process Table References

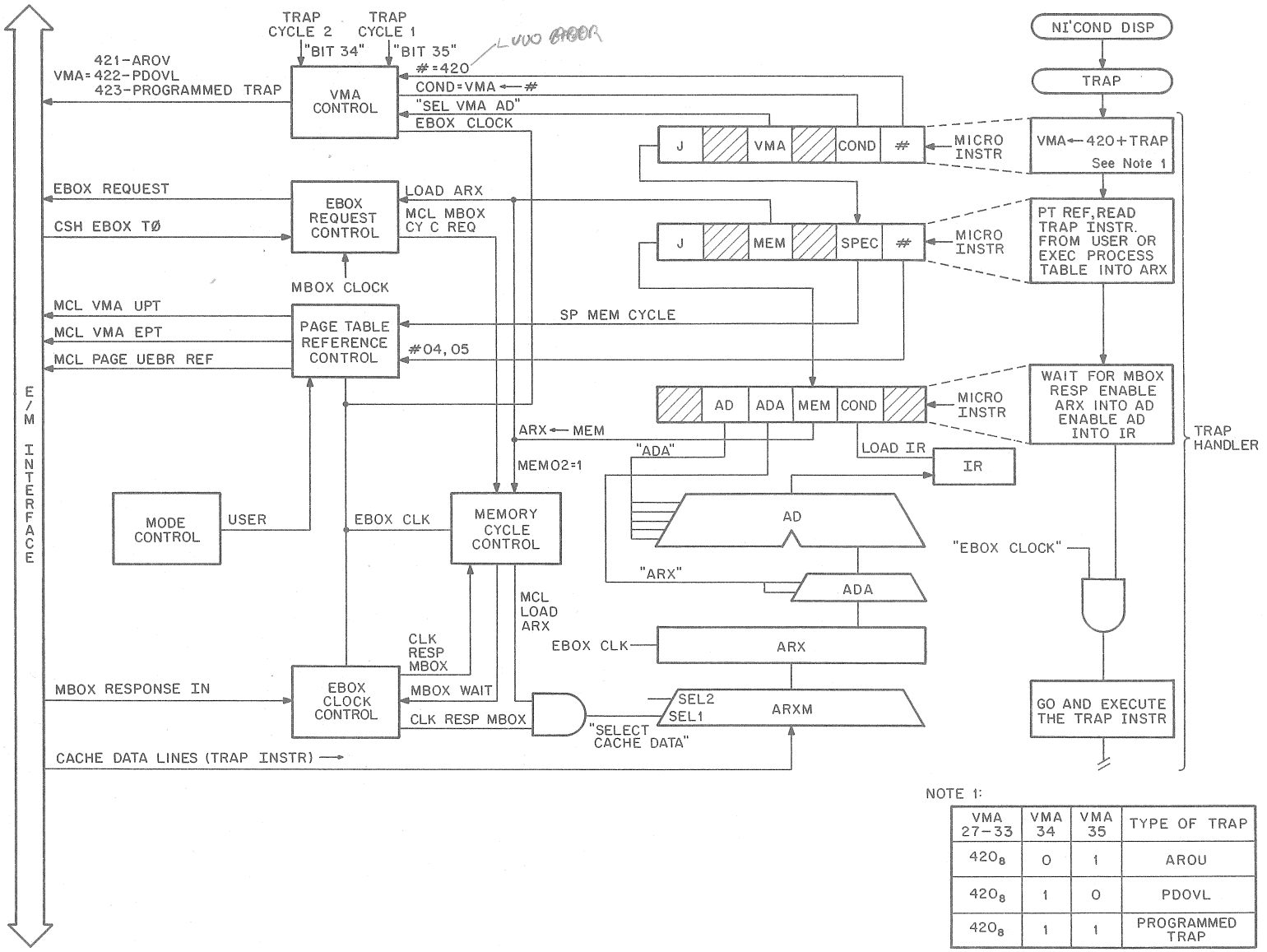
The VMA is loaded with low-order process table location 500 and an EBox request is issued to write the PF word (concurrently in AR) into process table location UBR+500. The next microinstruction is loaded and EBox clock sets MEM CYCLE, causing MBOX WAIT. The AR is enabled from the old PC word; the input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

2.5 TRAP CYCLE – INTRODUCTION

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

2.5.1 Trap Handling

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



NOTE 1:

VMA 27–33	VMA 34	VMA 35	TYPE OF TRAP
420_8	0	1	AROU
420_8	1	0	PDOVL
420_8	1	1	PROGRAMMED TRAP

10-1607

Figure 2-37 Trap Cycle

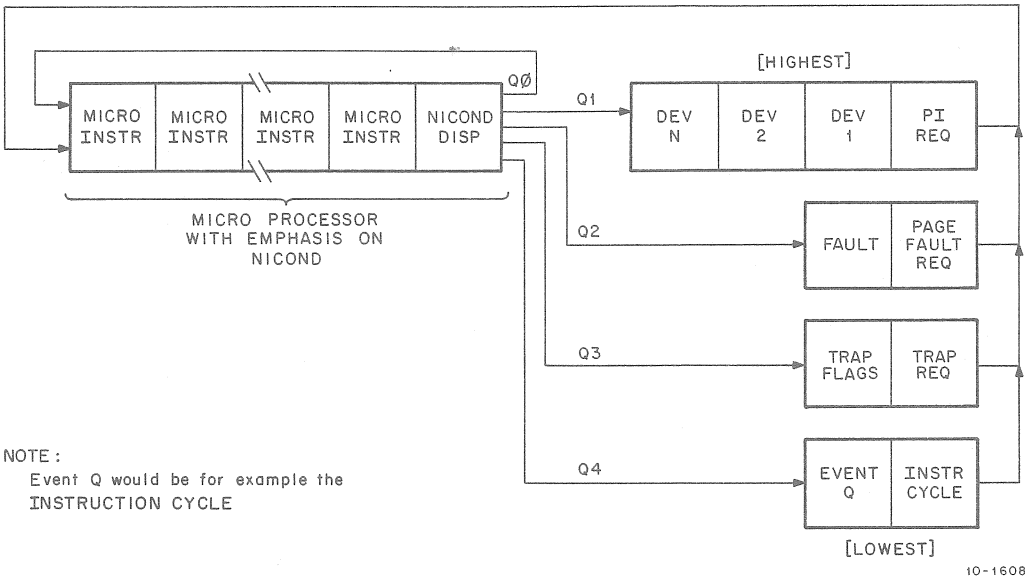


Figure 2-38 Central-Server Model (Round Robin Priorities)

2.5.2 Address Generation

Referring to Figure 2-37, the VMA is enabled to be input from the VMA ADDER. The condition field of the current microinstruction enables the number field to generate the process table low-order address 420; the low-order two bits of VMA AD 34 and 35 assume the state of the trap flags.

2.5.3 PT Reference for Trap Instruction

The next microinstruction must generate the EBox request and enable the appropriate qualifiers to appear on the E/M Interface lines. The page table reference control samples the state of the USER, together with the special function and number bits and then asserts either MCL VMA UPT and MCL PAGE UEBR REF for a USER trap situation or asserts MCL EPT and MCL PAGE UEBR REF for an EXEC trap situation. The MEM field is coded to load ARX and enable the EBox request.

Assuming no page fault occurs, the MBox fetches the instruction, places it on the cache data lines, and asserts MBOX RESPONSE IN. The MEM cycle control samples the MEM field function LOAD ARX to enable one leg of the ARXM and CLK RESP MBOX enables the other leg. Thus, the instruction enters ARX on the next EBox clock. Next, op code and AC field of the instruction in ARX must be enabled into the ADDER and then latched into IR. The condition field of the current microinstruction COND/LOAD IR unlatches the IR for one EBox cycle, allowing the AD to load into IR. On the next EBox clock, it latches again. The final step is to perform the trap instruction. This completes the trap cycle.

2.6 INTERRUPT CYCLE - INTRODUCTION

The system must possess a true priority interrupt system that is flexibly structured and controlled. Its operation in establishing priorities and recording and sequencing interrupt requests is essentially instantaneous and independent of EBox action. Interrupts of high priority must be permitted to interrupt partially completed responses to those of lower priority. To maintain fast response, interrupt requests should require no decoding action on the part of the EBox to determine their source or nature. Capability for dynamically varying the priority structure to meet the demands of a changing environment must be available. In addition, no other system element may be designed such that its proper operation requires inhibition of the priority interrupt system for any period of time.

The basic priority interrupt level has four mutually exclusive states that can be described as Disarmed (-PI ON), Armed (PI ON), Waiting (PI REQ), and Active (PI HOLD). Figure 2-39 shows the basic concept of the interrupt system for two channels. It is arranged in four groups, the interrupt state, the FF configuration for two of the seven possible channels, the level enable, and the source of change signal. In the Disarmed state, the interrupt level rejects all incoming interrupt trigger signals. By performing a CONO PI and specifying the appropriate bits, the priority interrupt system can be armed or disarmed for any or all channels.

In Figure 2-39, the processor (CPU) performs a CONO PI and arms both channels. In the armed state, the interrupt level accepts a trigger signal from an outside source or from an internal source, e.g., the APR, and moves to the waiting state (REQUEST STATE), where it remains until it is acknowledged by the EBox. All waiting and enabled requests are input to a priority network where they are compared with the current state of the priority interrupt system. In this example, both channel 1 and channel 2 are requesting service, and both channels have previously been armed by a CONO PI instruction. In addition, an interrupt is shown holding on channel 2. Thus, until it is dismissed by the processor, the channel 2 request pending is held in abeyance. Furthermore, the channel 1 request causes the device subroutine for channel 2 to be interrupted, diverting the processor to the device subroutine for channel 1. The first instruction that will be executed as a result of an interrupt (subroutine type service) is a JSR instruction. This instruction saves the processor flags, program counter value, and also holds the interrupt.

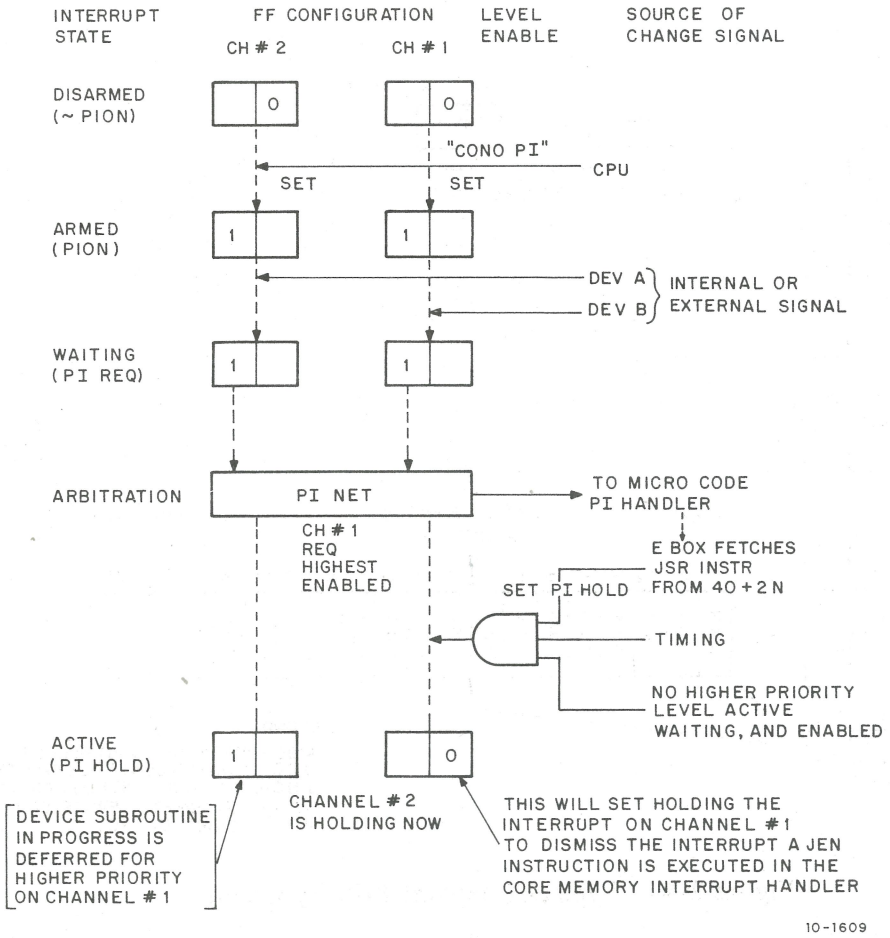


Figure 2-39 Interrupt Level Operations

When service has been completed, the service routine dismisses the interrupt, restores the flags and program counter, and the channel 2 subroutine continues. Interrupt channels are organized into seven basic levels, which are software assignable (armed): the lowest number has the highest priority within the numbered sequence (Figure 2-40). Each channel is subdivided into finer levels or priority by hard-wired physical device numbers. As indicated, the first eight physical numbers (0–7) are assigned to 1–8 Massbus controllers in the system. The next four physical numbers (8–11) are assigned to 1–4 DTE20s (10/11 Interfaces); and numbers 12–14 are reserved for expansion. Finally, physical number 15_{10} is assigned to the I/O bus adapter (one exists per system, if needed).

Each interrupt channel has a dedicated pair of unique locations within the EPT. These locations may be indicated as 40 + 2n, and 41 + 2n, where n represents the channel number. When a device initiates an interrupt in the KL10 system and is selected for service, the device places onto the EBus a special function word hereafter labeled API function. This function contains information that specifies the type of service required. Figure 1-32 indicates the format of this word. Note that the format varies from device to device, but the functions that can be specified in bits 3–5 are common to all system devices. Function codes of 0, 1, and 7 cause instruction fetches from 40 + 2n initially and, depending upon the type of instruction in 40 + 2n, may at some point perform an instruction fetch from 41 + 2n. In general, 40 + 2n contains one of the following types of instructions:

JSR
JSP*
PUSHJ*
MUUO

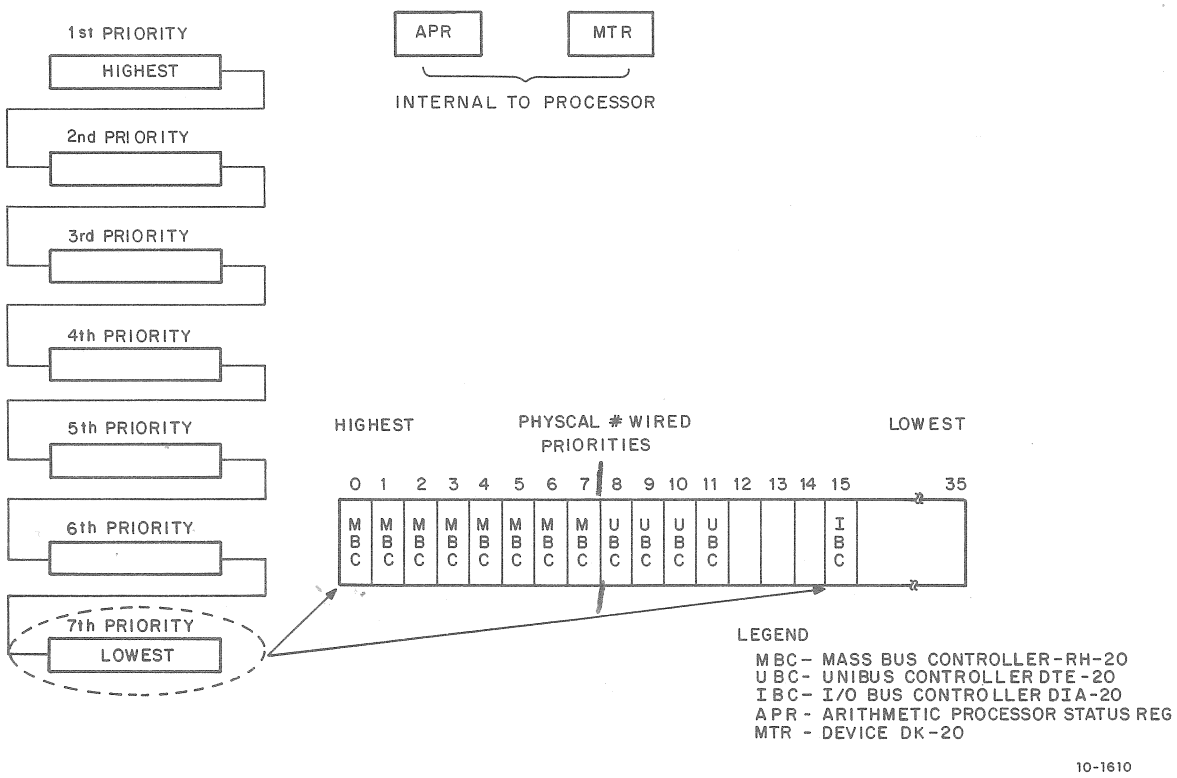


Figure 2-40 Typical Interrupt Priority Chain

*These instructions should not be used because nothing is known about the ACs when the interrupt occurs. JSR or MUUO are better choices.

All of these instructions save the flags and PC, a requirement when entering the device service routine. If the instruction at 40 + 2n is a BLKX instruction, a specified number of transfers are performed, one transfer at a time, each time returning to the interrupted program or to a higher level subroutine. On the last transfer, the return to the interrupted program is "NOT SKIPPED" and an instruction is fetched from 41 + 2n. In a similar fashion, if 40 + 2n contains a SKIP class instruction; when the skip condition is satisfied, a return to the interrupted program takes place. If the skip is not satisfied, the instruction in 41 + 2n is executed instead of the return. The API function generated by the Massbus controller is always a function code of 2 in bits 3-5; this implies a dispatch to the physical address provided in the API function word. The dispatch is into the device handler for the Massbus devices. The type of API function requested varies with the device or controller responding.

It is possible for the processor to generate a program request for an interrupt on any of the seven channels. This permits the processor to carry out the highly time-sensitive portion of the interrupt response, and to then create for itself a low priority interrupt to call for the deferred servicing of the less time-sensitive portion at a less pressing time.

2.6.1 Duration of Uninterruptable Intervals

Such an interrupt system is of little value if the CPU can remain in an uninterruptable state for any significant period of time. Under normal operating conditions, the longest uninterruptable interval must be kept short. In addition, no malfunctioning peripheral hardware or software can be allowed to "hang up" the CPU in a noninterruptable state.

2.6.2 Interruptable Instructions

To ensure that the longest uninterruptable interval that the EBox may experience in normal operation is short, some long instructions have been designed so that they may be interrupted during execution. First, all instructions are interruptable at indirect references during the effective address calculation. Second, instructions that consist of two parts may be interrupted between the two parts, a PC flag being set to record this for later, when only the second part will be done. Third, iterative instructions, such as BLT, may be interrupted at any point, as an AC pointer defining work still to be done is beng updated continually.

2.6.3 General Interrupt Sequencing

The mechanism for handling the various levels of interrupt priority in the hardware, and the relation between this mechanism and the device subroutine call and return sequence as it might occur in practice are shown in Figure 2-41. Three channels are armed by setting their PION flags. Channel 2 has highest priority, followed by channel 3, and finally by channel 4. Note that the execution of a CONO PI instruction caused the PION flags to set. Three separate interrupts occur simultaneously on channels 2, 3, and 4. The priority network is shown arbitrating the three priorities. The lowest channel (highest priority) is serviced, provided it is of higher priority than the current level.

In this example, all three channels are requesting and no channels are currently holding interrupts; thus, the channel with the lowest number is selected. As a result of the arbitration, the selected channel number is combined with the appropriate constant to form the address 44[40+2X (2)]. In Figure 2-41, the device subroutine is entered by fetching and executing the instruction in EPT location 44, which in this instance is a JSR. The request is not cleared until the program issues CONO, DEV. Notice during the entire service routine (in this example), the requests on channels 3 and 4 are waiting for the processor. The last instruction to be executed in the device subroutine is a JEN (JRST 12); this restores the flags saved by the JSR instruction executed in 40 + 2n and dismisses the interrupt on channel 2, which is holding off channels 3 and 4.

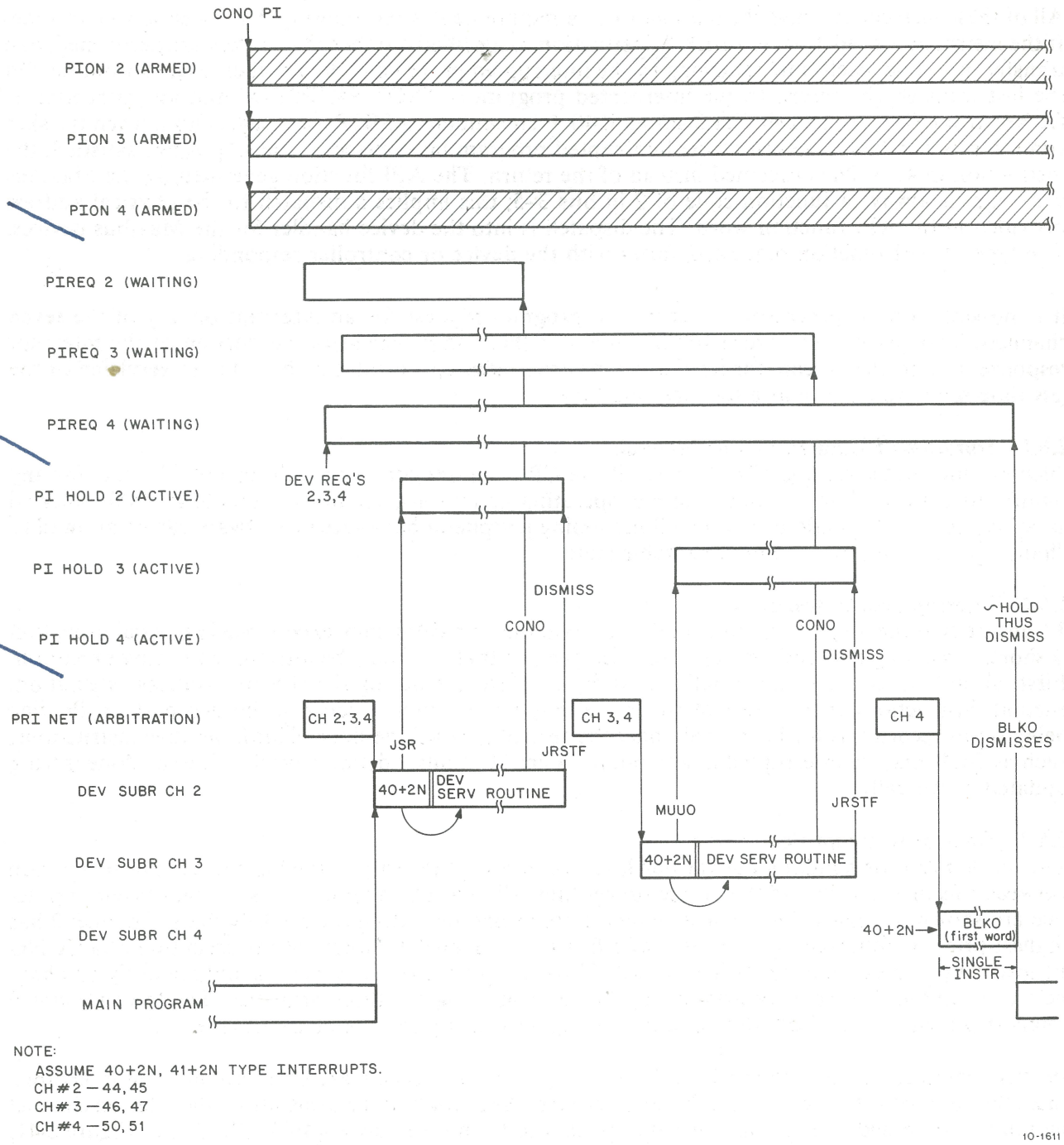


Figure 2-41 Basic Interrupt Sequencing

2.6.4 Interrupt Dialogue

The handling of the EBus dialogue and processor bus requests during I/O instruction execution and priority interrupts is provided by the Priority Interrupt Board, which comprises the necessary interfacing logic, control logic, and registers. Initially (Figure 2-42), assume that the appropriate PION flags have been set on the PI Board and it is now capable of accepting interrupts. For this example, the DTE20 will generate an interrupt for a byte of data. The drawing is divided into three sections: EBox, control activity, and DTE20. The control activity consists of control action taken by either the EBox or the DTE20, as appropriate.

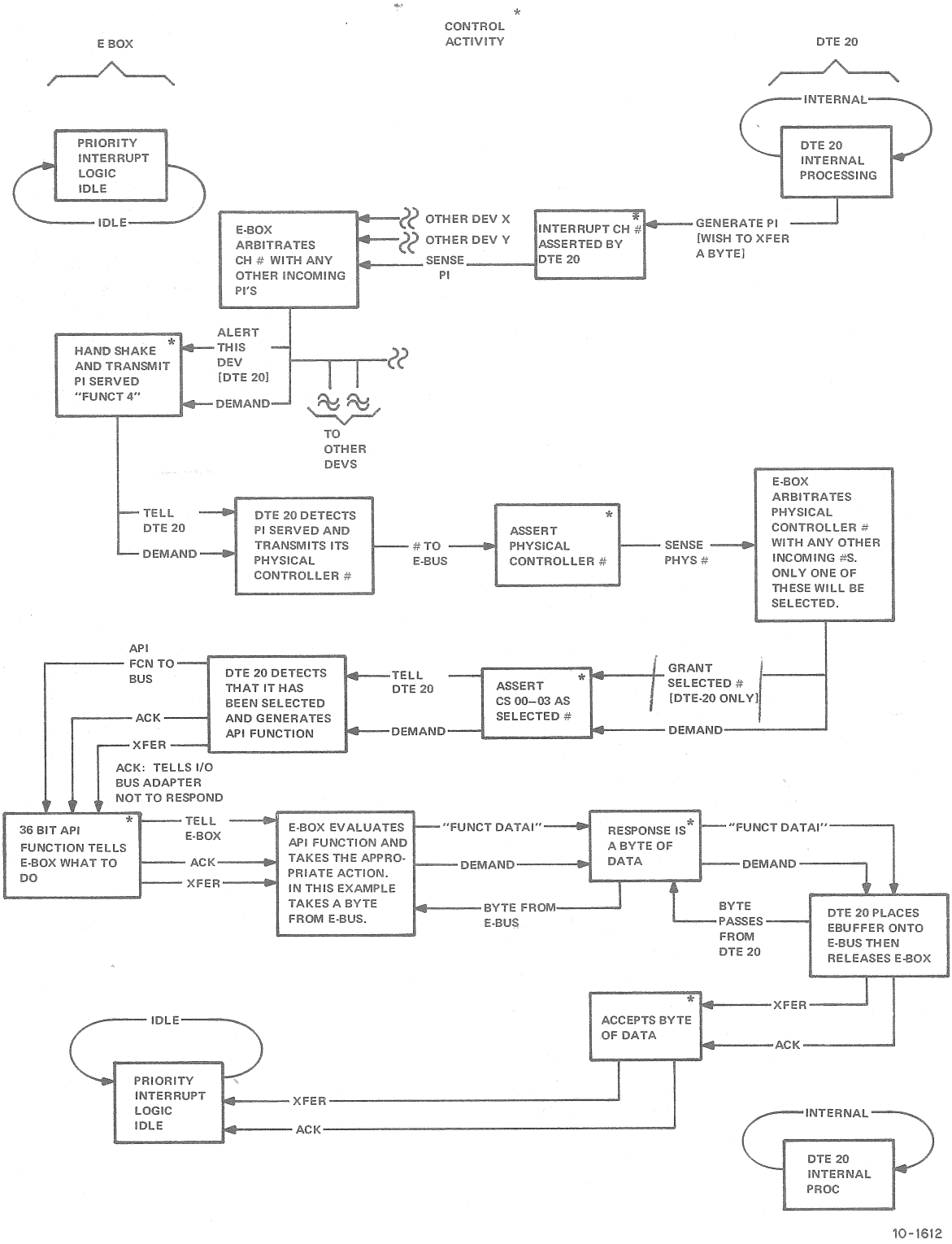


Figure 2-42 Interrupt Dialogue Overview

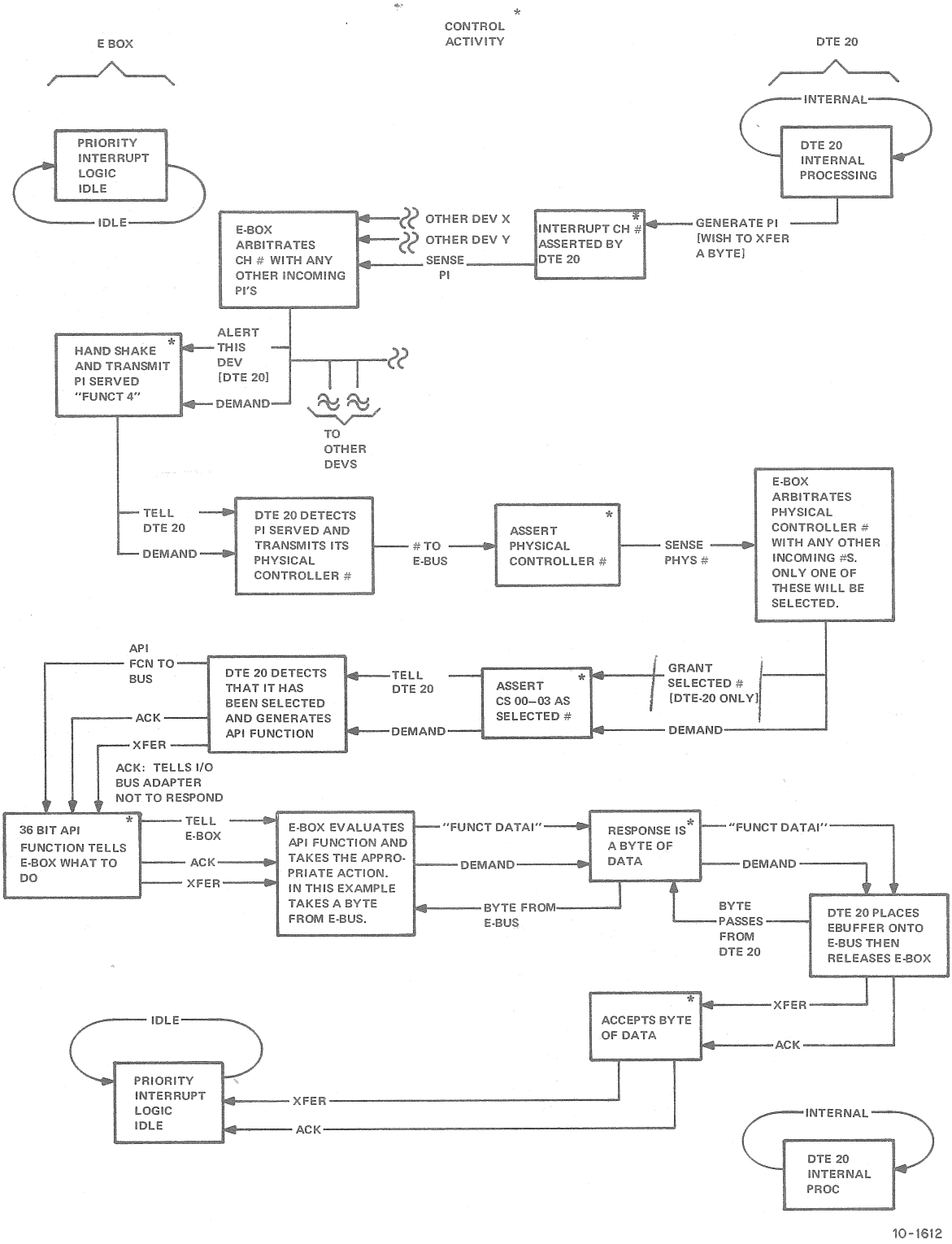


Figure 2-42 Interrupt Dialogue Overview

The DTE20 asserts one of its interrupt lines PI 1–7; this level enters the PI Board where, as indicated, it is arbitrated with any other incoming requests and any holding interrupts. The PI Board now commences a dialogue between all candidates on the selected interrupt channel. The selected channel number is encoded in controller select (CS) lines 04–06. The function "PI SERVED" is encoded in function (F) lines 00–02. These signals are placed on the EBus and 200 ns later the PI Board asserts the signal DEMAND. This signal instructs the device (DTE20) to place its physical controller number on a prespecified bit position of the EBus (bit positions 8–11). Each controller, therefore (including the I/O bus adapter, bit position 15, disks or drums, bit positions 0–7), on the selected channel does the same. Approximately 400 ns later, the EBox drops DEMAND; however, the controller select and function lines do not change for an additional 150 ns after DEMAND is removed. The physical controller numbers received by the EBox over the EBus are arbitrated in much the same way as the channel priorities. An exception is the ARP, which is an internal KL10 device, and does not fall into quite the same type of scheme, i.e., it does not place a physical number on the EBus; obviously this is not necessary because it is already within the EBox. Rather, it provides a physical number directory on the board. This device vies with the device that is selected on the basis of physical number highest priority (Figure 2-40). Basically, the lower the numeric value of the EBus bit position onto which the device is hardwired to place its physical number, the higher the priority of that bit. The highest physical number priority, therefore, is given to bit position 0, and the next to bit 1, and so on. The highest priority physical number (in this example only) is assumed to be that of the DTE20 (one of four such possible Unibus controllers on the EBus).

The PI Board now asserts the encoded physical number of the selected controller (DTE20) in controller select (CS) lines 00–03, the interrupting channel number encoded in CS lines 04–06, and the function "PI ADDRESS IN" is encoded in function lines (F) 00–02. Again, the EBox waits a period of 200 ns and then asserts DEMAND. At this point only, one controller has been selected; it compares its physical number (hardwired on its backplane) to the number received on EBus bits 00–03. Upon determining that it is the selected controller, the DTE20 places the required API interrupt function onto the EBus data lines and asserts ACKNOWLEDGE and TRANSFER to the EBox. The ACKNOWLEDGE signal causes the I/O bus adapter to ignore the function code "PI ADDRESS IN." In the absence of ACKNOWLEDGE, PI ADDRESS IN would enable the I/O bus adapter to send its API function to the EBox, because no decoding and comparison logic exists in the adapter. This logic does exist in the DTE20 and other devices. The TRANSFER signal specifies to the EBox that the appropriate device has responded, and alerts the EBox that an interrupt is set up and pending. If the API function is sent during a DTE20 to 10 byte transfer, this could specify that the EBox perform a DATAI function to the DTE20; in this way, a byte of data is picked up as indicated in Figure 2-40.

The case of DTE20 byte transfer is somewhat unique in that the DTE20 holds onto the EBus until the EBox transmits the appropriate function, in this case DATAI encoded in function select lines 00–02 (at this time CS00–06 = 0). The byte is picked up by the EBox, and the DTE20 generates ACKNOWLEDGE and TRANSFER once again. This completes the operation. Note that ACKNOWLEDGE informs the I/O bus adapter not to respond to the functions being carried out. Because the requests on channels 3 and 4 have been pending during the service routine, when the interrupt that has been holding on channel 2 is dismissed, the priority net arbitrates between channels 3 and 4 and selects 3 for service. This generates the address 46 (40 + 2n), and this time the instruction is an MUUO. As with the JSR during the execution of the MUUO, the request is transferred to the channel 3 hold flag. Note that in the example, the request on channel 4 is still waiting for service. Finally, the JEN instruction at the end of the channel 3 service routine restores the flags and priority interrupt system, dismissing the interrupt on channel 3. In the same fashion as with the other interrupts, the priority net generates the

The DTE20 asserts one of its interrupt lines PI 1–7; this level enters the PI Board where, as indicated, it is arbitrated with any other incoming requests and any holding interrupts. The PI Board now commences a dialogue between all candidates on the selected interrupt channel. The selected channel number is encoded in controller select (CS) lines 04–06. The function "PI SERVED" is encoded in function (F) lines 00–02. These signals are placed on the EBus and 200 ns later the PI Board asserts the signal DEMAND. This signal instructs the device (DTE20) to place its physical controller number on a prespecified bit position of the EBus (bit positions 8–11). Each controller, therefore (including the I/O bus adapter, bit position 15, disks or drums, bit positions 0–7), on the selected channel does the same. Approximately 400 ns later, the EBox drops DEMAND; however, the controller select and function lines do not change for an additional 150 ns after DEMAND is removed. The physical controller numbers received by the EBox over the EBus are arbitrated in much the same way as the channel priorities. An exception is the ARP, which is an internal KL10 device, and does not fall into quite the same type of scheme, i.e., it does not place a physical number on the EBus; obviously this is not necessary because it is already within the EBox. Rather, it provides a physical number directory on the board. This device vies with the device that is selected on the basis of physical number highest priority (Figure 2-40). Basically, the lower the numeric value of the EBus bit position onto which the device is hardwired to place its physical number, the higher the priority of that bit. The highest physical number priority, therefore, is given to bit position 0, and the next to bit 1, and so on. The highest priority physical number (in this example only) is assumed to be that of the DTE20 (one of four such possible Unibus controllers on the EBus).

The PI Board now asserts the encoded physical number of the selected controller (DTE20) in controller select (CS) lines 00–03, the interrupting channel number encoded in CS lines 04–06, and the function "PI ADDRESS IN" is encoded in function lines (F) 00–02. Again, the EBox waits a period of 200 ns and then asserts DEMAND. At this point only, one controller has been selected; it compares its physical number (hardwired on its backplane) to the number received on EBus bits 00–03. Upon determining that it is the selected controller, the DTE20 places the required API interrupt function onto the EBus data lines and asserts ACKNOWLEDGE and TRANSFER to the EBox. The ACKNOWLEDGE signal causes the I/O bus adapter to ignore the function code "PI ADDRESS IN." In the absence of ACKNOWLEDGE, PI ADDRESS IN would enable the I/O bus adapter to send its API function to the EBox, because no decoding and comparison logic exists in the adapter. This logic does exist in the DTE20 and other devices. The TRANSFER signal specifies to the EBox that the appropriate device has responded, and alerts the EBox that an interrupt is set up and pending. If the API function is sent during a DTE20 to 10 byte transfer, this could specify that the EBox perform a DATAI function to the DTE20; in this way, a byte of data is picked up as indicated in Figure 2-40.

The case of DTE20 byte transfer is somewhat unique in that the DTE20 holds onto the EBus until the EBox transmits the appropriate function, in this case DATAI encoded in function select lines 00–02 (at this time CS00–06 = 0). The byte is picked up by the EBox, and the DTE20 generates ACKNOWLEDGE and TRANSFER once again. This completes the operation. Note that ACKNOWLEDGE informs the I/O bus adapter not to respond to the functions being carried out. Because the requests on channels 3 and 4 have been pending during the service routine, when the interrupt that has been holding on channel 2 is dismissed, the priority net arbitrates between channels 3 and 4 and selects 3 for service. This generates the address 46 (40 + 2n), and this time the instruction is an MUUO. As with the JSR during the execution of the MUUO, the request is transferred to the channel 3 hold flag. Note that in the example, the request on channel 4 is still waiting for service. Finally, the JEN instruction at the end of the channel 3 service routine restores the flags and priority interrupt system, dismissing the interrupt on channel 3. In the same fashion as with the other interrupts, the priority net generates the

address 50 (40 + 2n). In this case, however, location 50 contains a BLKO instruction, which cannot save the flags or PC of the interrupted process. This type of instruction behaves in a special manner when used in an interrupt location; the BLKO instruction performs a series of transfers to a specific device; however, after each transfer, return is passed to the current PC value, whatever it is. This continues until the last transfer is completed, when the instruction in EPT location 51 (41 + 2n) is executed. This instruction should be of the type that saves the flags and PC, and will generally enter a subroutine probably to set up a new block pointer, because the current one has been expended. Note that in the beginning some main program, perhaps the monitor, was interrupted, and now control is passed back to it.

2.7 BASIC MACHINE MODES INTRODUCTION

In general, the KL10 permits the operation of a number of different programs, all resident in the machine simultaneously, without interference or undesired interaction among them whether due to an inadvertent program bug or maliciousness. The operation of the machine is divided into two modes, User mode and Exec mode, each with two submodes. User mode consists of Public mode and Concealed mode. Exec mode consists of Supervisor mode and Kernel mode. The machine mode structure and hierarchy are illustrated in Figure 2-43.

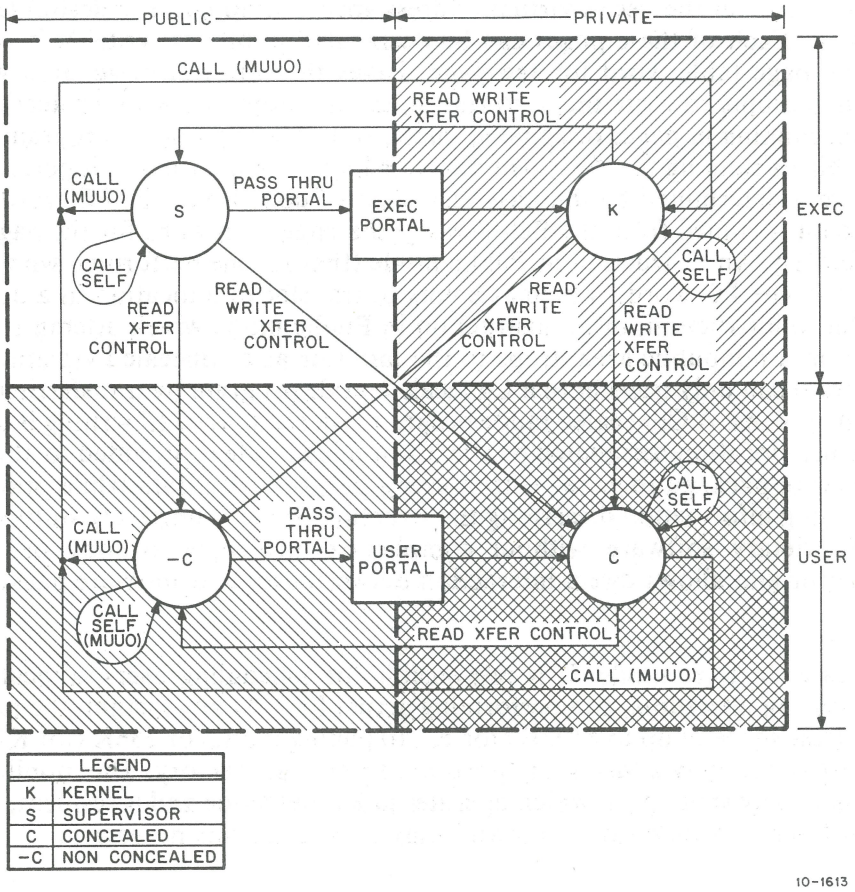


Figure 2-43 Mode Structure and Hierarchy

Basically, the programs of individual users operate in Public User mode, where the program can have access to one of two possible virtual address spaces. If KL10 paging is in effect, the user has access to a virtual address space of 256K words via an 18-bit virtual address, which may not be referred to by any other user (without the cooperation of the monitor). If KI10 paging is turned on, the program has access to a virtual address space of 256K addressed via a 18-bit virtual address, which as previously pointed out cannot be referenced by any other user without the monitor's cooperation. All instructions that do not compromise the integrity of the system are legal; this includes the following:

1. The halt instruction (JRST 4)
2. Any instruction attempting to affect the PI system (JEN)
3. Any I/O instruction directed at devices with device select codes below 740
4. Any reference to the concealed address space except for fetching of a portal instruction
5. All illegal instructions or op codes.

The user's address space (when KL10 paging is in effect) is divided into 32 (decimal) sections; each section contains 512 (decimal) pages and each page consists of 512 (decimal) words. The existence of these pages is nominally invisible to the user program. However, the amount of physical address space available is actually a number of these pages (at least one page), none of which need be contiguous either in physical core or in the user's virtual address space, although it is desirable from a machine standpoint to do so. Each of these pages can be designated public or writable by a 1 in bit 1 or 2, respectively, in the page table word for the page. Pages that are not designated writable cause an instruction, which attempts to write them, to trap to the monitor as a write protection violation page failure. A program running in pages designated public is in Public mode. A program running in pages not designated public is running in Concealed mode. Whether an instruction is performed from Public or Concealed mode is determined by the Last Instruction Public bit of the PC word (bit 7). The Last Instruction Public bit is copied from the Public bit of the page map word for the page from which the instruction was fetched. An instruction in Public mode (that is, one performed with the Last Instruction Public bit a 1 in the PC word), which attempts to transfer to a location in a nonpublic area not containing any Portal instruction, or an instruction in Public mode which attempts to read, write, or execute a location in a nonpublic area, traps to the monitor as a concealed violation page failure. A Public mode program can only transfer to a Concealed mode program by transferring to locations that contain Portal instructions. A Concealed mode program can read, write (if writing is allowed), execute, or transfer to any user location designated public. Concealed mode is provided to allow the loading of a proprietary software package together with a user's program and data while preventing the user's program from copying information discerning the structure of the proprietary software. This provides protection of proprietary software without complicated protective overlay or transfer schemes involving the monitor and allows direct interaction between user and software package with virtually no overhead.

The monitor operates in Exec mode. It is responsible for scheduling users, allocating memory and other facilities, servicing interrupts, and performing actual I/O. At any instant, the monitor has access to an effective address space of up to 8192K (for KL10 paging mode) or 256K (for KI10 paging mode) words and by overt action may address any portion of physical memory. The monitor can be divided into two parts: a normally small part, which operates in Kernel mode and is resident, and a larger part, which operates in User or Supervisor mode and may be swapped as necessary.

The Kernel mode part of the monitor handles the PI system, performs the direct I/O for the system, performs page management, and performs all other functions that affect all users of the system. The Supervisor mode part of the monitor performs the general management of the system (such as MUUO handling and dispatch) functions which affect only one user at a time. The Supervisor mode and Kernel mode of the monitor are analogous to the Public mode and Concealed mode of the user's programs in that the Supervisor runs in that part of the Exec address space designated public and the Kernel runs in that part of the Exec address space which is designated nonpublic; this simplifies illegal reference detection logic. Each address from 20 through 337,777 is broken up into pages, but these addresses can be made to refer to the same addresses in the physical memory space by making the virtual page address equal to the physical address portion in the corresponding page table entry. The entire Exec address space is broken into pages of 512 words which may be designated either accessable or not accessable, public or nonpublic, and writable or nonwritable and can be swapped out. An instruction in Supervisor mode that attempts to write into a page which is not writable will trap as a page failure. An instruction in Kernel mode may write into any location whether or not it is designated public. An instruction in Supervisor mode (that is, one performed with the Last Instruction Public bit a 1 in the PC word) that attempts to transfer to a location in an Exec nonpublic area not containing a Portal instruction traps to the monitor as a page failure. An instruction in Supervisor mode that attempts to read, write, or execute a location in an Exec nonpublic area traps to the monitor. In each instance, the trap is a Kernel violation page failure. A Supervisor mode program can only transfer, i.e., jump to a Kernel mode program, by transferring to locations that contain Portal instructions (JRST 1).

A Supervisor mode program can also reach Kernel mode (or any other mode) by performing an MUUO or other instruction that causes a trap, if the appropriate trap new PC word indicates that the next instruction is in Kernel mode. A Kernel mode program can read, write, execute, or transfer to any location designated public, i.e., in Supervisor mode; all instructions illegal in User mode are also illegal in Supervisor mode.

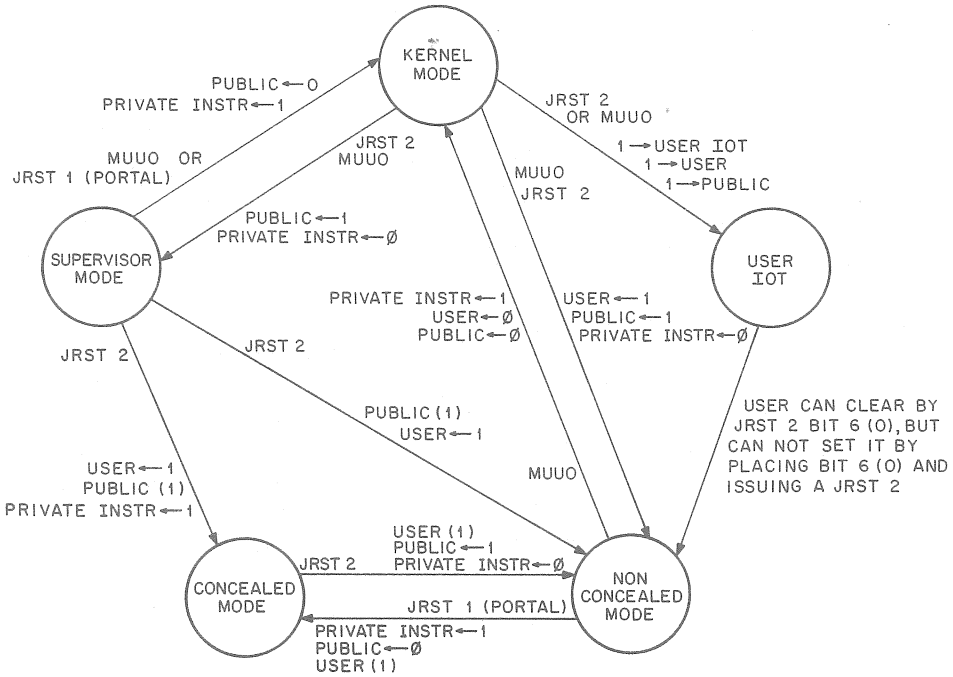
The mode control logic consists of the following:

User Mode
Public Mode
User IOT
Private INSTR
Miscellaneous Combinational Logic

The mode control exerts a powerful influence over the disposition of the processor. It monitors instruction fetches from Public mode to prevent illegal entry to either Concealed mode from User Public mode or Kernel mode from Supervisor. In addition, it detects the fetch of a Portal instruction and adjusts the state of the mode logic accordingly. The relationships between the various modes and their transfer instructions are shown in Figure 2-44. In general, two instructions allow flags that affect processor modes to be manipulated. These instructions are:

MUUO
JRST 2

Of the two, only the MUUO can cause transfers to any mode from any other mode. The JRST 1 (Portal 1) simply allows entry to a Private mode from a Public mode. Each time an instruction fetch is specified and the reference is to a nonpublic page, a test for illegal entry must take place to maintain integrity in the system.



BIT ASSIGNMENTS				
CONTROL OF	BIT05	BIT06	BIT07	NOTES
USER MODE	X			
USER IOT		X		IN USER MODE
PUBLIC MODE			X	
PREVIOUS CONTEXT		X		IN EXEC MODE

10-1614

Figure 2-44 Mode Transfer

Referring to Figure 2-44, assume a User Public program has been started by a monitor routine that performed a JRST 2 (a jump and restore flags). To place the processor in User Public mode, bits 7 and 5 of the flag's PC word must be set; this results in the setting of Public mode and user mode, respectively. The processor is now in User Public mode. Assume that the User executes some miscellaneous instructions and then performs an instruction fetch from a nonpublic area. The following test takes place: instruction fetch is decoded from the microinstruction MEM field or specified as a prefetch in the DRAM A field. The E/M Interface asserts EBOX READ and loads the address into VMA. Note that if a reference to a private address for a read or write of data is attempted, it page fails on the attempted reference because PAGE TEST PRIVATE is asserted. However, in this case, the fetch must be allowed from the private address space. Its identity is checked in the EBox and, if it is not a JRST 1 (portal), a page failure occurs on the very next memory reference. This is implemented by delaying generation of the signal that would cause a page failure to be generated by the MBox (PAGE ILLEGAL ENTRY), until the instruction fetch is completed. When the MBox responds with the level – PAGE TABLE PUBLIC (PT PUBLIC), this signal with the MB response sets PRIVATE INSTRUCTION. This causes the generation of PAGE ILLEGAL ENTRY. If the instruction which is decoded by the hardware is not a Portal, Public mode remains set maintaining PAGE ILLEGAL ENTRY, which enables a page fault on the next MBox reference for whatever reason. If the instruction fetched is a portal (JRST 1), then Public is cleared and the processor enters Concealed mode.

All user references and concealed references are paged. The difference between the types of paged references is that user paged references are public while concealed references are nonpublic when referencing the concealed address space and may be public when referencing the users address space. Executive references are paged, this includes both Kernel and Supervisor references. Supervisor mode programs must be capable of reading both User Public and User Concealed address spaces. To bypass the portal mechanism normally necessary for any public program to reference a nonpublic program area, a bypass exists, which is under control of the Kernel; when operational, the Supervisor is allowed to read and possibly write the concealed area as necessary, remembering, of course, that the supervisor is part of the operating system and it is performing job-related tasks within that context.

Normally a public program is only allowed to fetch an instruction from a nonpublic area and this instruction must be a portal (JRST 1) instruction; however, this is necessary for the supervisor to perform its system tasks. Basically, the process for checking a User Public program's reference to a concealed address is as follows. The mode is User Public and an instruction fetch begins. EBOX REQUEST is issued to the MBox, together with the appropriate paging qualifiers and any other appropriate signals. The MBox performs the necessary check of the page descriptor bits; then the state of the Public bit in the page table is asserted over the E/M Interface where, together with signal MB XFER and a signal indicating an instruction fetch is being performed, it is used to enable the setting of Private instruction. If the Page Table Public bit is off, Private instruction is set on the clock occurring concurrently with MBox response. PAGE ILLEGAL ENTRY is not asserted. The response given by the MBox was given at the same time it placed the desired instruction onto the cache data lines; this instruction is now in ARX. If the instruction is indeed a portal instruction (JRST 1), the Public mode will be cleared, removing the PAGE ILLEGAL ENTRY signal. This procedure then has effected the proper entry to Concealed mode. If the instruction was not a Portal, then the PAGE ILLEGAL ENTRY signal will not be removed nor will Public be cleared, which constitutes an illegal state in the EBox. On the very next MBox request by the EBox (providing VMA AC REF is false), a page fault occurs and an appropriate code is placed in the EBus register in the MBox identifying the disposition of this fault. This will shortly be followed by a trap to the operating system as a concealed violation page failure. This same procedure is applied to a Supervisor reference to the Kernel address space, and in this way the integrity of the system is protected from any unwarranted references. Figure 2-45 shows a typical layout of the virtual address space for the various modes. The space shown is for KI10 paging mode (256K, made up of 512 pages numbered 0–777 octal). Any program can address locations 0–17 as these are in a fast memory block and are completely unrestricted (although the same addresses may be in different blocks for different programs). The Public mode user program operates in the public area, part of which may be write protected. The Public program cannot access any locations in the concealed area, except to fetch instructions from prescribed entry points. The Concealed mode user program has access to both the public and concealed areas, but it cannot alter any write protected location whether public or concealed; fetching an instruction from the public area automatically returns the processor to Public mode. The Supervisor mode program is confined within the paged area of the address space. Part of the public area in this space may be write protected, but the program can read information in the concealed area. It cannot, however, alter any location in a concealed area, whether that area is write protected or not. Pages 340–377 constitute the per process area, which contains information specific to individual users and whose mapping accompanies the user page map. In other words, the physical memory corresponding to these virtual pages can be changed simply by switching from one user to another, rather than the operating system changing its own page map. The Kernel mode program can access all of the unpaged area without restriction and can reference all of the accessible paged area both public and concealed, with the usual restriction that it cannot alter a write protected area. As in the case of Concealed mode, fetching an instruction from a public area returns control to Supervisor mode.

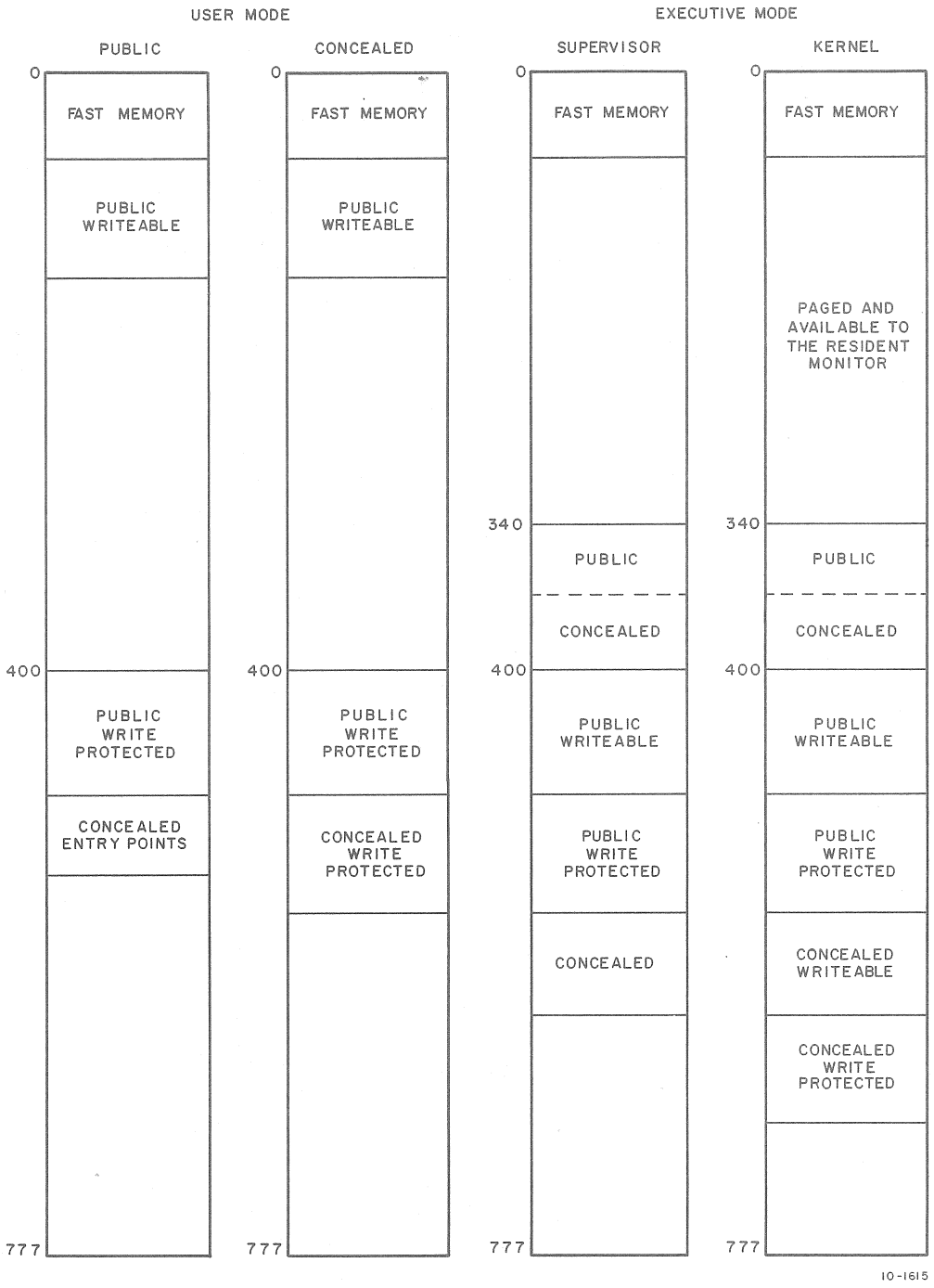


Figure 2-45 Typical Virtual Address Space Configuration

2.7.1 Mode Initialization – Private Instruction

When the KL10 system is powered up, the power control issues the signal CROBAR for approximately 5 seconds. This results in the generation of RESET, which causes LEAVE USER to be asserted. LEAVE USER enables the clearing of USER, USER IOT, and PUBLIC and sets PRIVATE INSTRUCTION. This action places the KL10 in Kernel mode. Referring to Figure 2-46, each time an instruction is fetched from either Fast Memory or Core Memory (via MBox), the private instruction recirculation path is broken (Figure 2-47).

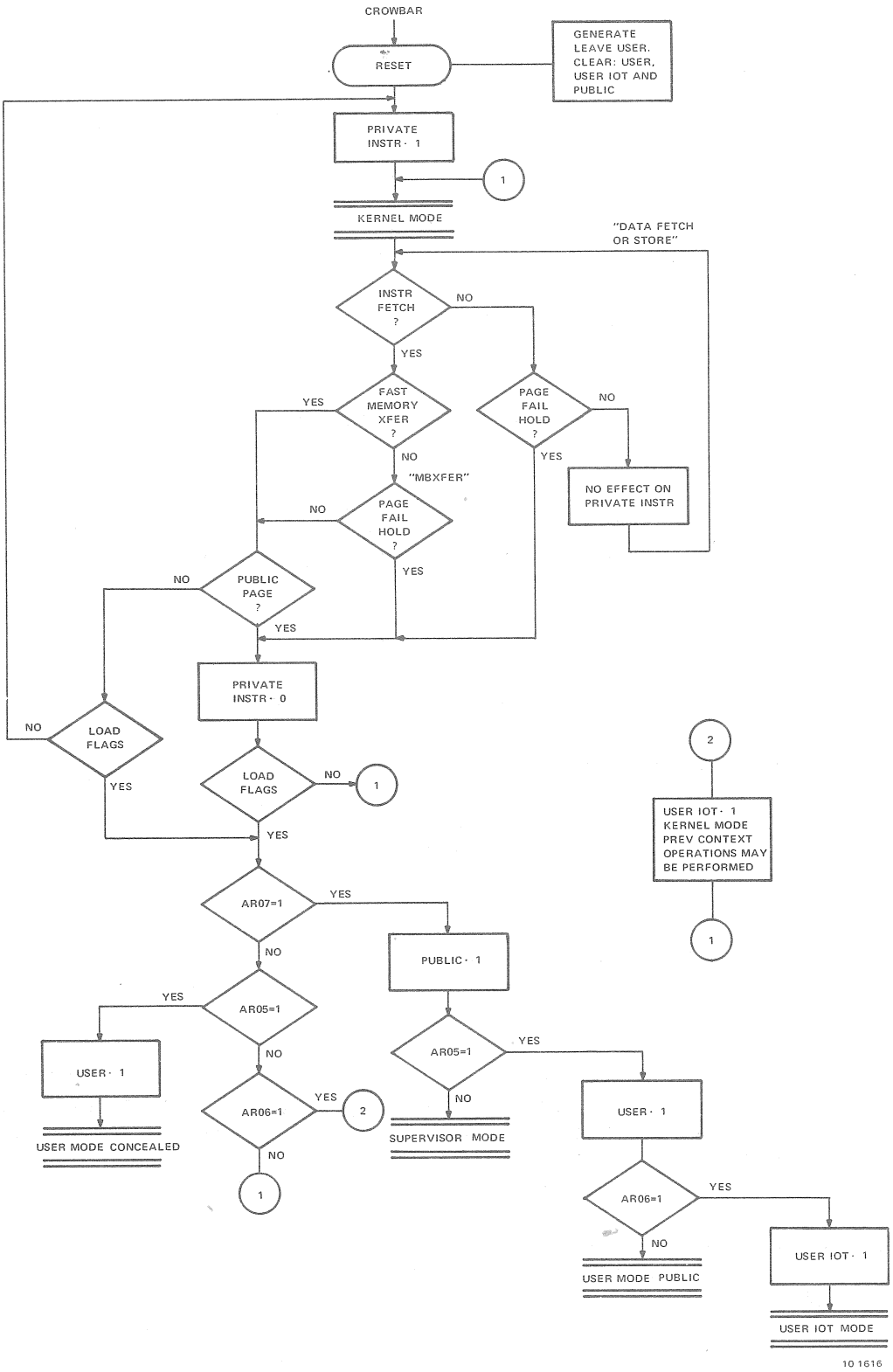


Figure 2-46 Mode Initialization

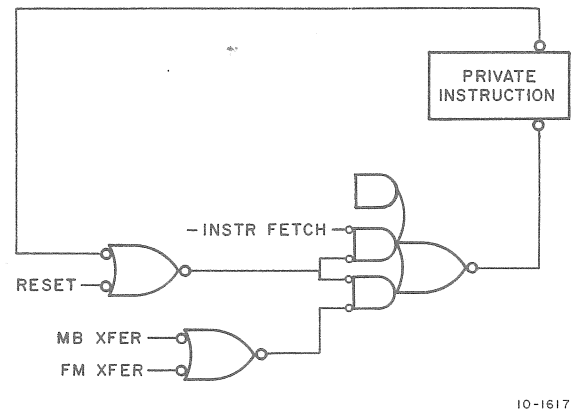


Figure 2-47 Private Instruction Recirculation Path Simplified

If the instruction is fetched from a nonpublic address space (-PUBLIC PAGE), or the mode of the machine is not public (-PUBLIC), then the private instruction is enabled to be set once again (Figure 2-48).

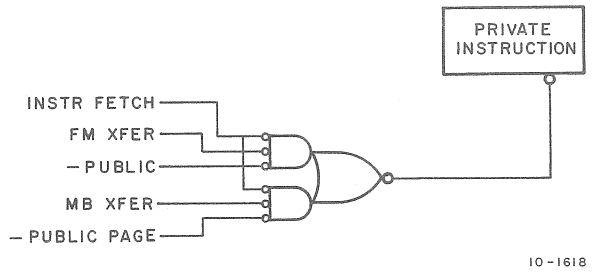


Figure 2-48 Setting Private Instruction

Note that if data is read or written, the upper recirculation leg (Figure 2-48) is not disabled. The Private Instruction flip-flop is used with additional logic that (with the exception of previous context references) detects references to Public mode; together, these elements detect entry to a privileged address space. The Kernel may access any part of the address space regardless of its type. Because the Kernel does not operate in Public mode, illegal entry has no significance.

2.7.2 Loading Flags and Changing Mode

Two instructions can change the mode of the machine. These instructions are MUUO and JRST with AC bit 11 set, i.e., JRSTF.

As indicated in Table 2-6, AR bits 05 and 07 are used in various combinations to enter appropriate submodes.

Table 2-6 Flags Effecting Mode

Instruction being performed is MUUO,JRSTF (See Note)					Major Mode			
Enable PREVCONTXT	User IOT	Flag Bits AR06	Effecting Modes		Exec Submodes		User Submodes	
			AR05	AR07	Kernel	Super	Concealed	Public
0	0	0	0	0	1	0	0	0
1	0	1	0	0	1	0	0	0
		N/A	0	1	0	1	0	0
0	0	0	1	1	0	0	0	1
0	1	1	1	1	0	0	0	1
0	0	0	1	0	0	0	1	0
0	1	1	1	0	0	0	1	0

NOTE
A JRSTF may not clear user by placing bit 05 (0) but an MUUO may.

In addition, for Direct User I/O, bit 06 (USER IOT) is available to allow the running of privileged user programs with paging in effect. This mode provides some protection against partially debugged monitor routines, and permits running infrequently used device service routines as a user job. Direct control by the user program of special devices is particularly important in real-time applications. A special MUUO is available to enter USER IOT mode, but it is privileged because time-sharing is effectively stopped while in this mode.

2.7.3 User Public Mode

Once the processor is in User Public Mode (Figure 2-49), the user program can freely read and write data in the user public address space with the cooperation of the system. When demand paging is in effect, each reference to a previously unreferenced page causes an access page fault. The operating system page manager must assess the fault, obtain the page from mass storage, and build an entry in the user's process table.

Assuming that the current user's process table (PAGE TABLE PART) is initially clear, the first reference causes a NOT IN CORE page fault (Figure 2-50). The EBox, upon detecting the PAGE FAIL HOLD signal from the MBox, enters a microcode page fault handling routine that communicates the failure to the operating system. Next, the page manager or a related routine requests the page from mass storage. When the page is in core, the appropriate process table is constructed and the reference by the user program may be tried once again (Figure 2-51).

The MBox performs the reference to the process table; the use bits now reflect the following:

PAGE IS IN CORE A = 1
PAGE IS WRITABLE W = 1
PAGE IS PUBLIC P = 1
PAGE SHOULD BE CACHED C = 1

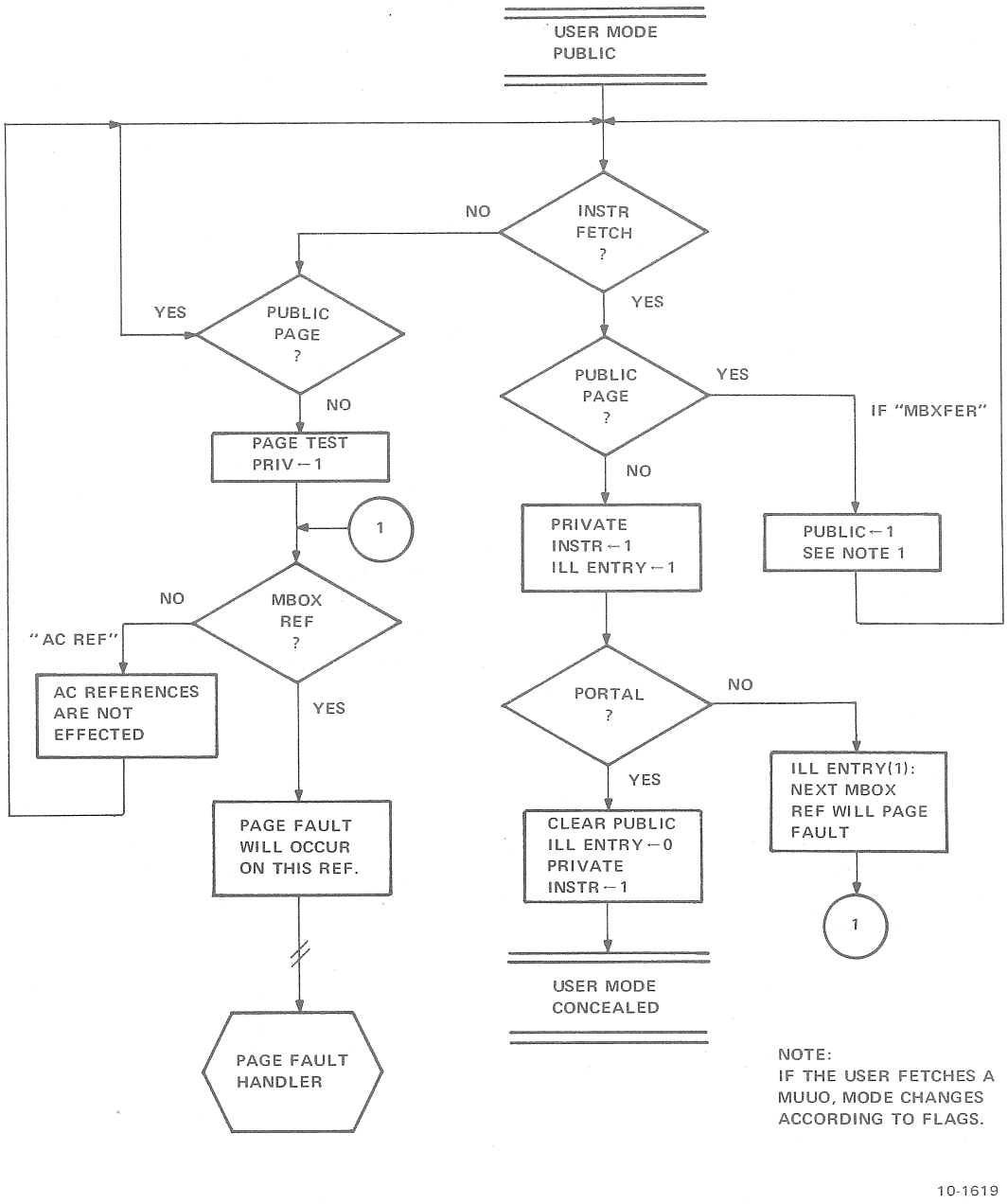


Figure 2-49 User Mode Functional Flow

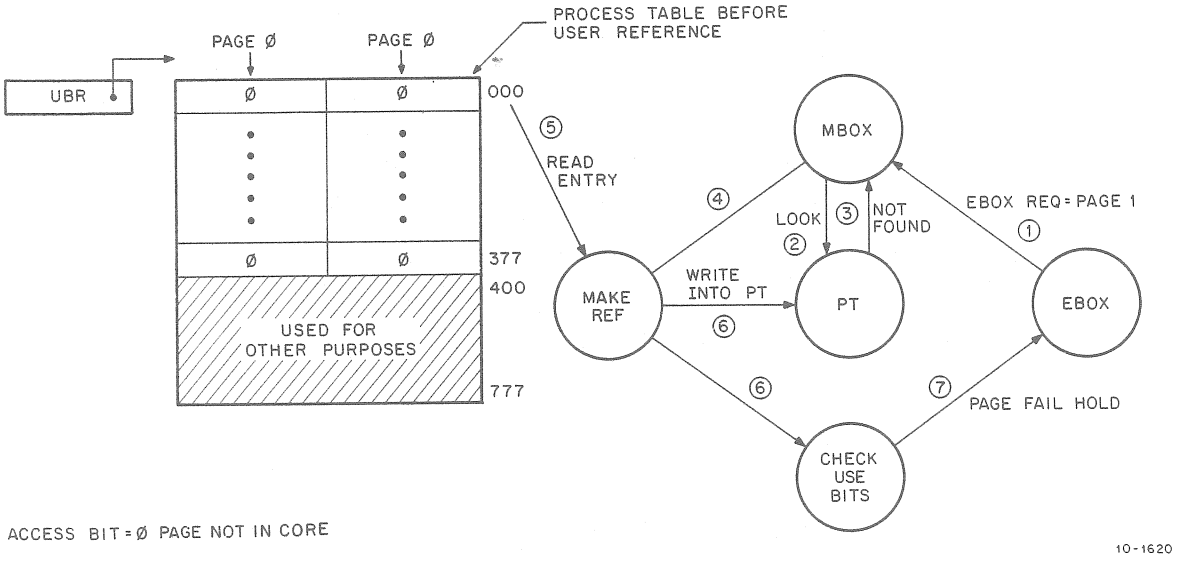


Figure 2-50 User Mode Public Initial Reference

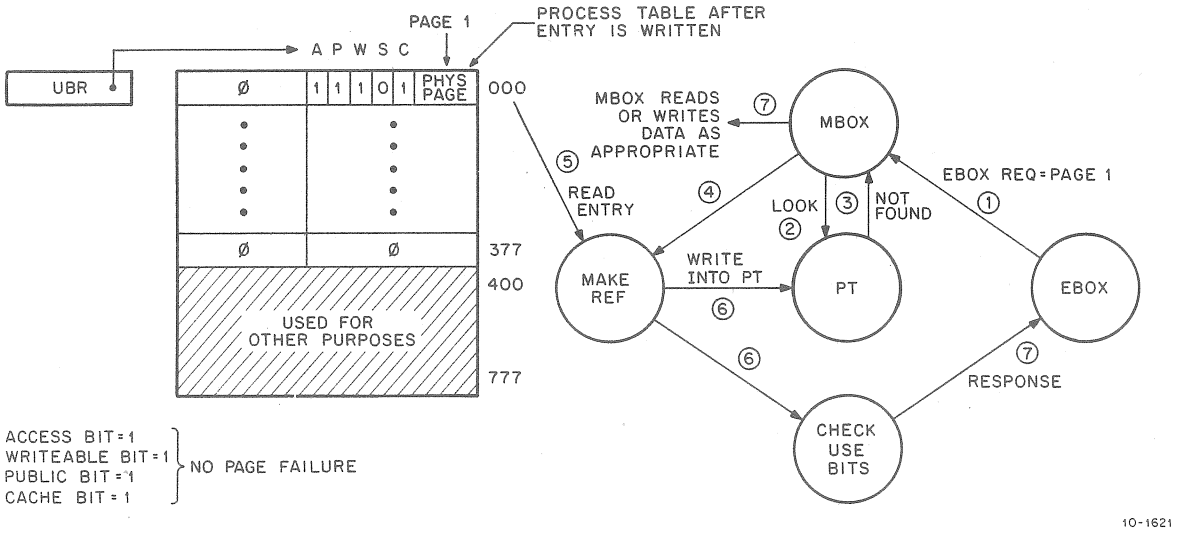


Figure 2-51 User Mode Public Second Reference

The entry (one of eight half-word entries fetched) is written into the page table in the MBox, the MBox then performs the data reference part of the request. This can involve reading or writing and depends upon the type of EBox request. During the reference, PAGE ILLEGAL ENTRY was not asserted because the reference made by the user program was to a public page and it was for an instruction.

2.7.3.1 Entry from User Public Mode to User Concealed – To correctly enter User Concealed mode, the User Public program must execute a Portal instruction (Figure 2-49) from the concealed address space. The EBox generates the EBox request and provides the MBox via VMA with the concealed address. The MBox either finds the page entry and use bits in the MBox Page Table (hardware) or performs a refill cycle to obtain it from core memory. Figure 2-52 shows the typical Concealed Page Table format. Presumably, the entry is nonpublic and write protected, and may or may not be cached.

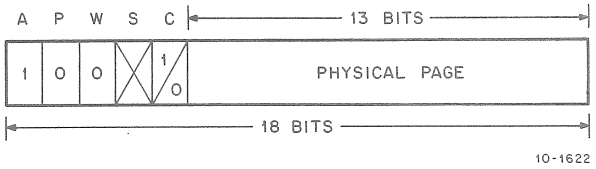


Figure 2-52 Typical Concealed Page Table Format (Half Table Entry)

The MBox asserts PT PUBLIC (0) and MBOX RESPONSE IN to the EBox. Referring to Figure 2-48, MB XFER resulting from MBox response and -PUBLIC PAGE resulting from PT PUBLIC (0) enables the setting of Private instruction. The instruction fetched by the MBox is in ARX at this time. If it is a JRST 1 (Portal), its execution clears Public and the processor enters User Concealed mode. If the instruction is anything else, Public remains set and the next MBox reference occurs with PAGE ILLEGAL ENTRY true, PUBLIC PRIVATE INSTR (1); this causes a page failure.

2.7.3.2 Concealed Violation Data Reference – If a User Public program references the concealed address space for read or write, PAGE TEST PRIVATE is asserted during the EBox request and results in an immediate page fault. Page Test Private is a signal composing Public and -INSTR FETCH.

2.7.4 Restoration of Programs by the Supervisor

The Supervisor portion of the operating system deals with those tasks which affect one job at a time. It must, therefore, have the ability to restore various programs to an operational status, e.g., by executing a JRST 2 instruction that picks up a PC word consisting of the appropriate flags in the left half and a virtual PC in the right half of the word.

2.7.4.1 Restoring a Concealed Program – The Supervisor may restore a concealed program providing it also sets User. Referring to Figure 2-53, while executing a JRST 2 instruction, LOAD FLAGS is derived from the presence in the magic number field of bit 04, and this together with -User (User is off in Supervisor mode), and AD bit 05 (which will set User) generated CLR PUBLIC. Thus, on the next clock pulse, Public clears and User sets, restoring Concealed mode. Figure 2-54 shows the necessary conditions. Note that performing a JRST 2 cannot generate Leave User, unless the processor is in Kernel mode.

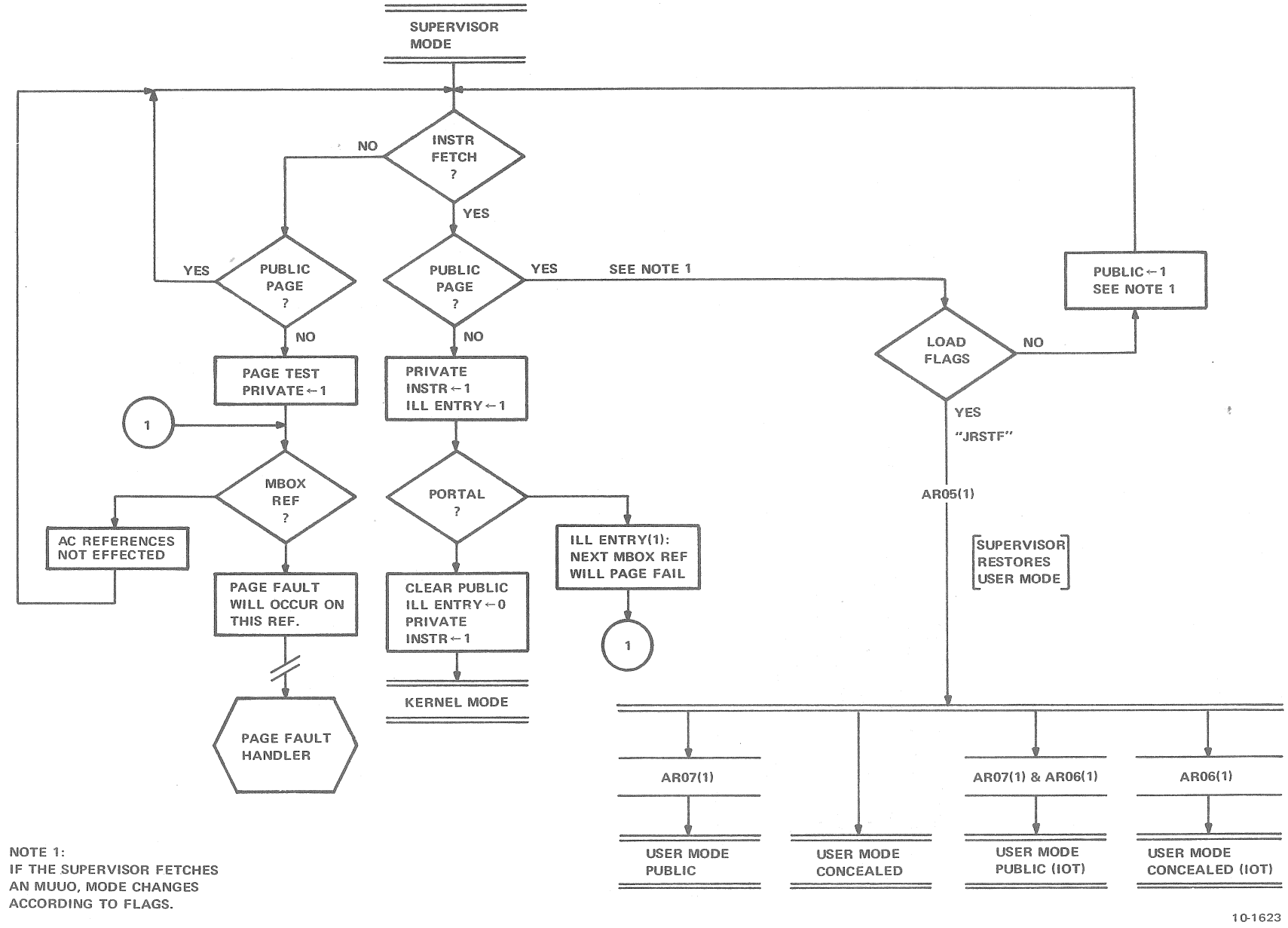


Figure 2-53 Supervisor Mode Functional Flow

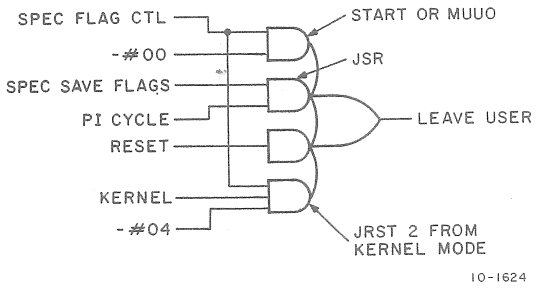


Figure 2-54 Leaving User

2.7.4.2 Restoring a Kernel Program – The restoration of a Kernel mode program from Supervisor mode is somewhat different in its mechanics than the restoration of the Concealed program. Basically, the Supervisor must first perform a JRST 2 instruction; this instruction restores all flags except for Public. The JRST must enable the fetching of a Portal instruction that clears Public, placing the machine in Kernel mode. This is a safeguard in the event that the Supervisor may, in error, try to restore some random set of bits and cause the Kernel to be disturbed. In addition, it forces entry to Kernel mode at a known and unique entry point. Figure 2-55 shows that it is not possible for a JRST 2 instruction to clear Public while not setting User as well. Note that a JRST 2 instruction does not generate Leave User unless it is given in Kernel mode. The conditions which enable Leave User are indicated on Figure 2-54.

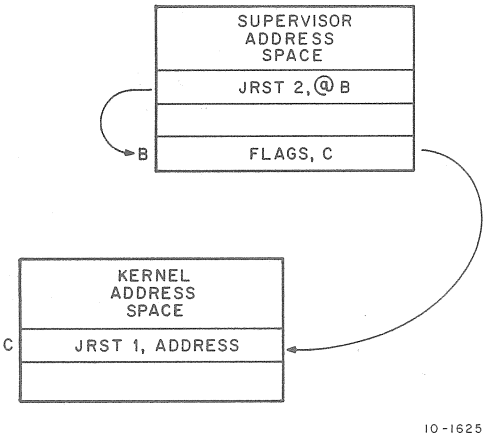


Figure 2-55 Restoring Kernal Program

2.7.4.3 Restoring a User Public Program – To restore a User Public program, the Supervisor gives a JRST 2, which sets User. This is the only requirement because both Supervisor and the User Public program run with Public set. The special field function SPEC FLAG CTL, together with magic number 04(1) enables SPEC/LOAD flags which, with AD bit 05, enables User to set on the next clock.

2.7.4.4 Saving Flags and Leaving User – It is not generally known at just what moment an interrupt will occur with respect to execution of a given instruction. The microprogram governs the handling of interrupts by looking for interrupts only at certain times. In general, an interrupt is sampled for between each instruction and during certain classes of instructions. The following classes of instructions can be interrupted:

Byte Instructions
Block Transfer Instruction
Input/Output Instructions

In addition, for any instruction, an interrupt is sampled during the portion of the microprogram that performs indirect addressing (INDRCT). An interrupt has higher priority than a Page Fault and thus, upon entry to the Page Failure microroutine, an interrupt condition is tested for; if found, a dispatch to the microroutine for interrupt handling is given.

When an interrupt occurs and the PI logic has completed the handshake, it informs the EBox by asserting a signal PI READY. This results in the microprogram generating a skip to a microinstruction that asserts SPEC/SET PI CYCLE. As a result, Kernel cycle (normally false as long as PI CYCLE is clear) sets, and MCL VMA PUBLIC is disabled. This is necessary to disable the MCL PAGE ILLEGAL ENTRY signal when PI CYCLE sets because the interrupt instruction, which will be fetched from a Kernel address, must not generate a page fault.

When the interrupt instruction is being fetched, User and Public may be set, or Public alone may be set. In the last instance, a page fault would result if some action were not taken to prevent it. This is why MCL PAGE ILLEGAL ENTRY is disabled (by setting PI CYCLE). At the time of the interrupt, the state of the current user ACs is unknown. The instruction in 40 + 2n, therefore, must not disturb the ACs in any way while transferring the flags and PC to the Kernel mode subroutine. Therefore, JSR is a likely instruction for use in 40 + 2n. The JSR instruction causes the flags and current PC to be stored in the effective address of the JSR instruction and then enters the subroutine by performing an instruction fetch from E + 1. After calculating the effective address for the JSR instruction, the microprogram performs a write test which, if successful, is followed by a branch via the DRAM J field to the executor. Now the flags and PC are loaded to be copied into the AR for storage and are then disabled. The microinstruction asserts SPEC FLAG CTL; this with PI CYCLE generates LEAVE USER, which detaches the feedback path for User, User IOT, and Public. In addition, if User were set, User IOT would be set at this time and represent "Previous Context User." This is an indicator to the hardware that previous context references must be in User mode. In any event, the processor enters Kernel mode and begins to handle the interrupt.

2.7.4.5 User Concealed – This mode is useful for running certain proprietary programs in User mode without allowing the user to discern the composition of the concealed program. For example, assume a user has developed a program that performs circuit analysis. The user is a time-sharing house and desires that this program be available to users for execution only, that is, the user must not be able to read or write into this program.

In some computer systems, complex overlays in core memory are necessary to assure concealment of the program from its users. In the KL10, this program has been solved by creating two submodes from User mode, each with separate powers and each separate from the other. Both modes, however, run with User on. Figure 2-56 indicates the hierarchial structure present in the KL10 processor. The User Public program can only transfer to a concealed program at a selected entry called a Portal. The instruction fetched must be a Portal instruction (JRST 1). The concealed program can read or write data to the Public area. Figure 2-57 is the Concealed mode functional flow diagram.

INSTRUCTION	EFFECT ON USER, PUBLIC, AND PRIVATE INSTR FLAGS	PREVIOUS MODE	NEW MODE
MUUO	PUBLIC AND USER CLEAR, PRIVATE INSTR SETS	USER	KERNEL
MUUO	PUBLIC CLEARS, PRIVATE INSTR SETS	SUPERVISOR	KERNEL
MUUO	USER CLEARS	CONCEALED	KERNEL
MUUO	NO EFFECT	KERNEL	KERNEL
PORTAL (JRST 1)	PRIVATE INSTR SETS, PUBLIC CLEARS	USER	CONCEALED
PORTAL (JRST 1)	PRIVATE INSTR SETS, PUBLIC CLEARS	SUPERVISOR	KERNEL

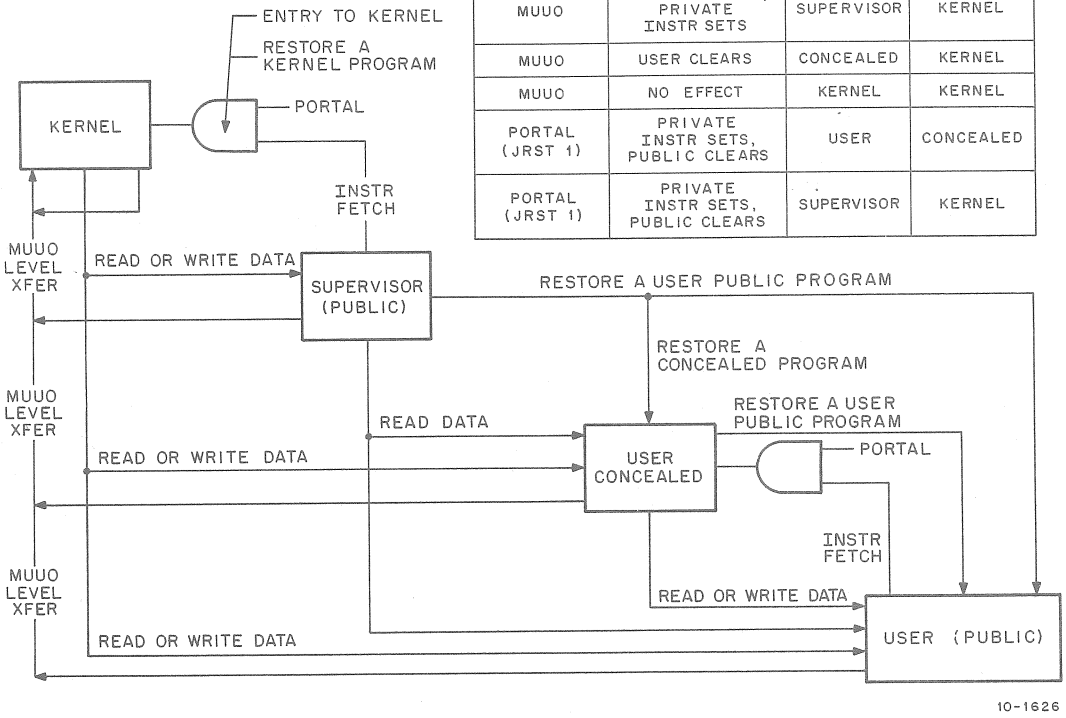


Figure 2-56 Mode Hierarchy

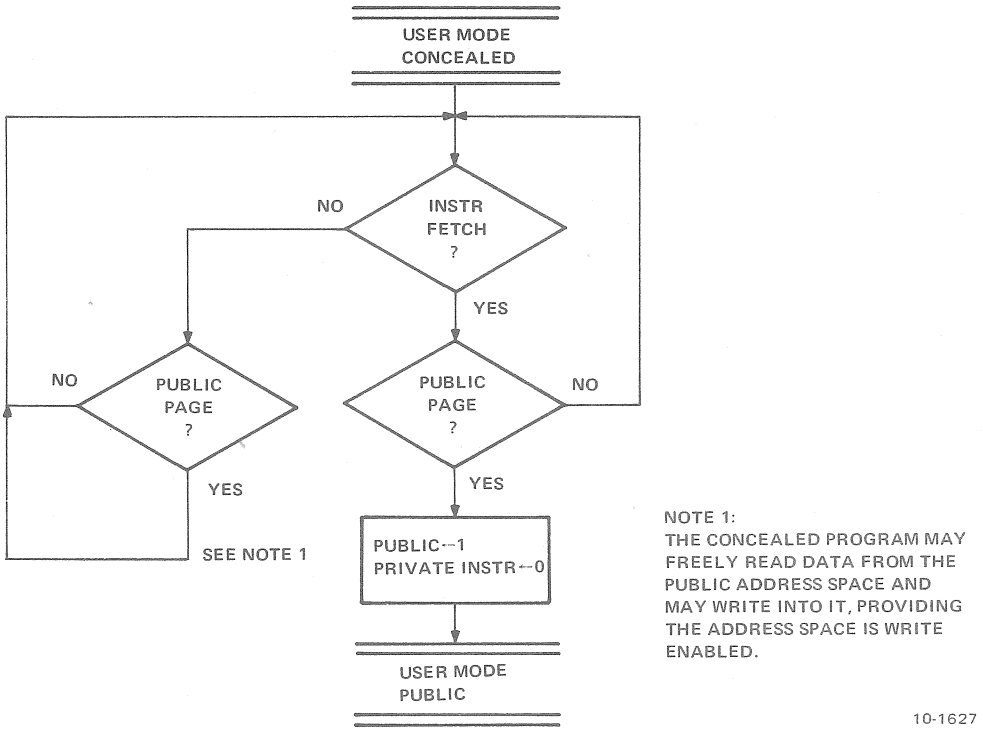


Figure 2-57 Concealed Mode Functional Flow

2.8 ADDRESS PATHS

The address paths contained within the EBox are illustrated in Figure 2-58. These paths are implemented to facilitate the formation of the appropriate MBox virtual address. This address is translated by the MBox for KI paging mode and by the microprogram and the MBox for KL paging mode. The MBox can generate the following two basic forms of physical addresses:

1. Refill Address (Relocated)
2. Physical Page Address (Paged)

The VMA serves as a source of data when loading the following MBox registers:

1. User Base Register (UBR)
2. Executive Base Register (EBR)
3. Cache Clearer (CCA)

In addition, it serves as an address and data source when loading the cache refill RAM. As indicated in Figure 2-59, the VMA has the three following basic sources of input:

1. Previous Context Section register (PCS)
2. Virtual Memory Address Adder (VMA AD)
3. Adder (AD)

The following two major addressable areas are addressed by the VMA:

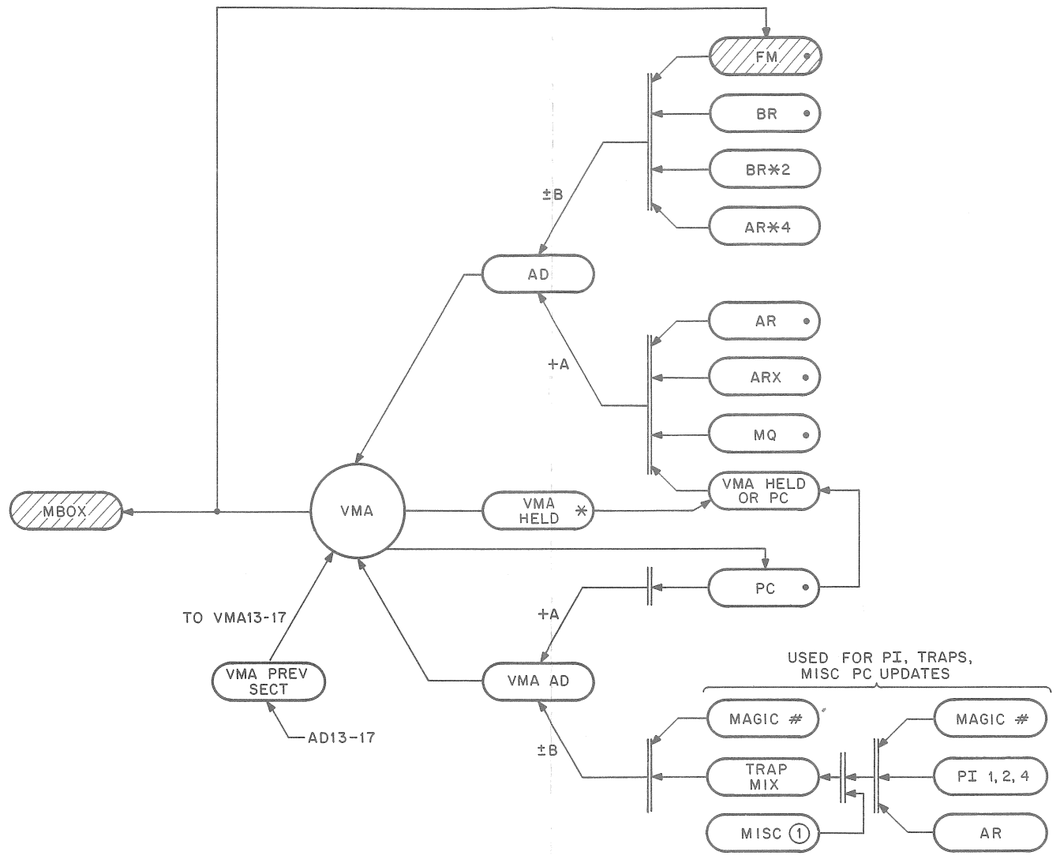
1. MBox
2. Fast Memory (FM)

The MBox may be addressed logically by two types of addresses. Within each type (18-bit and 23-bit addressing) is a class of process table addresses. These addresses are identified to the MBox by the qualifiers asserted during the EBox request (Table 2-7).

Table 2-7 Virtual Address Classification

Type of Address	Class	Addressing Information Supplied
18-Bit	KI Paged	VMA 13–17 = 0 VMA 18–26 = Virtual Page VMA 27–35 = Quad Word
18-Bit	KI Process Table Reference	VMA 13–17 = MBox Ignores VMA 18–26 = MBox Ignores VMA 27–35 = Process Table Word
23	KL Paged	VMA 13–17 = Virtual Section VMA 18–26 = Virtual Page VMA 27–35 = Quad Word
23	KL Process Table Reference	VMA 13–17 = MBox Ignores VMA 18–26 = MBox Ignores VMA 27–35 = Process Table Reference

NOTE
There are several other special VMA combinations. These will be covered elsewhere.



• – THESE REGISTERS MAY ACTIVELY BE INVOLVED IN SOME FORM OF ADDRESS. CALCULATION WHICH WILL ULTIMATELY BE PLACED INTO VMA.

* – USED DURING KL10 STYLE PAGING ONLY

(1) – USED TO FORCE PC+1 OR PC+2

Figure 2-58 EBox Address Paths Simplified Path Diagram

	VMA				
ADDRESS TYPE	VMA 13–17	VMA 18–35	SOURCE OF ADDRESS	BY WAY OF	
18 BIT	PC 13–17	VMA AD 18–35	PC	VMA AD	KI PAGING
18 BIT	RECIRCULATED	AD 18–35	E	AD	
18 BIT	CLEAR	VMA AD 18–26=0, 27–35	TRAP	VMA AD	
18 BIT	CLEAR	VMA AD 18–26=0, 27–35	PI OR SPECIAL	VMA AD	
18 BIT	RECIRCULATED	AD 18–35	@	AD	
18 BIT	RECIRCULATED	AD 18–35	MISC	AD	
23 BIT	VMA PREV SECT 13–17	AD 18–35	PC 13–17 OR E BUS	VMA PREV SECT AD	KL PAGING
23 BIT	AD 13–17	AD 18–35	E (EXTENDED)	AD	
23 BIT	CLEAR	VMA AD 18–26=0, 27–35	TRAP	VMA AD	
23 BIT	CLEAR	VMA AD 18–26=0, 27–35	PI OR SPECIAL	VMA AD	
23 BIT	PC 13–17	VMA AD 18–35	PC	VMA AD	

		USER		~ USER	
		KI PAGING MODE	KL PAGING MODE	KI PAGING MODE	KL PAGING MODE
PUBLIC	VMA AC REF	VMA 13–33=0 VMA 32–35= FM ADDRESS [USER PUBLIC]	VMA 13–33=0 VMA 32–35= FM ADDRESS [USER PUBLIC]	VMA 13–33=0 VMA 32–35= FM ADDRESS [SUPERVISOR]	VMA 13–33=0 VMA 32–35= FM ADDRESS [SUPERVISOR]
	~ VMA AC REF	VMA 13–17=0 VMA 18–26= VIRTUAL PAGE VMA 27–35= QUAD WORD [USER PUBLIC]	VMA 13–17=SECT VMA 18–26= VIRTUAL PAGE VMA 27–35= QUAD WORD [USER PUBLIC]	VMA 13–17=0 VMA 18–26= VIRTUAL PAGE VMA 27–35= QUAD WORD [SUPERVISOR]	VMA 13–17=SECT VMA 18–26= VIRTUAL PAGE VMA 27–35= QUAD WORD [SUPERVISOR]
~ PUBLIC	VMA AC REF	VMA 13–33=0 VMA 32–35= FM ADDRESS [USER CONCEALED]	VMA 13–33=0 VMA 32–35= FM ADDRESS [USER CONCEALED]	VMA 13–33=0 VMA 32–35= FM ADDRESS [KERNEL]	VMA 13–33=0 VMA 32–35= FM ADDRESS [KERNEL]
	~ VMA AC REF	VMA 13–17=0 VMA 18–26= VIRTUAL PAGE VMA 27–35= QUAD WORD [USER CONCEALED]	VMA 13–17=SECT VMA 18–26= VIRTUAL PAGE VMA 27–35= QUAD WORD [USER CONCEALED]	VMA 13–17=0 VMA 18–26= VIRTUAL PAGE VMA 27–35= QUAD WORD [KERNEL]	VMA 13–17=SECT VMA 18–26= VIRTUAL PAGE VMA 27–35= QUAD WORD [KERNEL]

NOTE: THIS IS THE GENERAL FORMAT ONLY.

10-1628

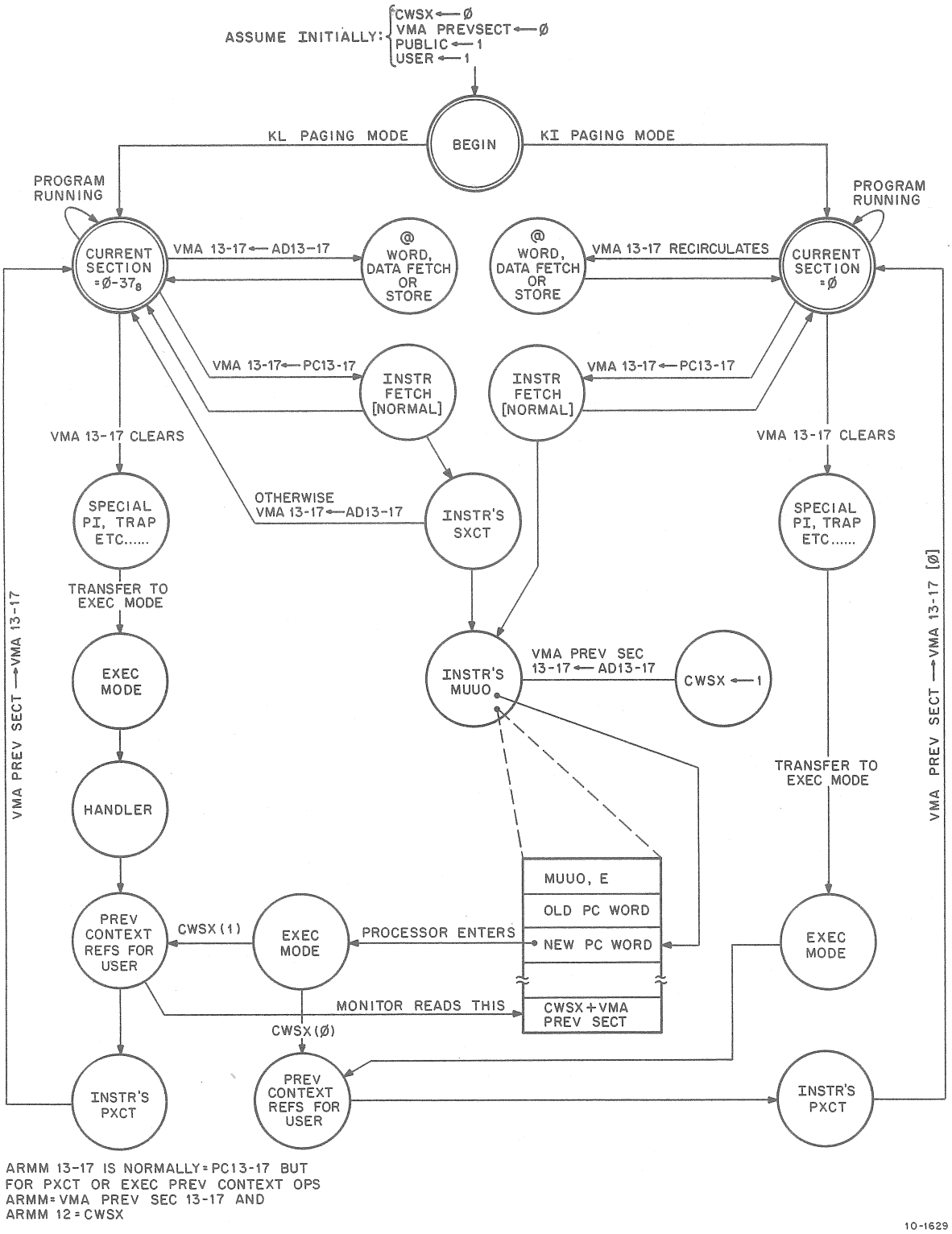


Figure 2-59 Typical VMA 13-17 Manipulations

For these process table references the EBox supplies valid addressing information only on VMA bits 27–35. The MBox replaces VMA 13–26 with the PMA mixer 14–26 to generate a proper physical address.

2.9 DATA PATHS

The specific address and data paths in the EBox are illustrated in Figure 2-60.

The functional elements in the address path between the VMA at the MBox/EBox Interface and the primitive address source involved in forming the virtual addresses are:

Virtual Memory Address Register (VMA)
VMA Held or PC Mixer
VMA Held Register
VMA Previous Section
VMA Mixer
VMA Adder (VMA AD)
SCD TRAP Mixer
ADDER (AD)
Arithmetic Register Extension (ARXML)
Arithmetic Register (AR)
Program Counter (PC)
Microinstruction Number Field
Other Miscellaneous EBox Registers

The appropriate virtual address is formed by the VMA under explicit control of the VMA control and the microprogram.

2.9.1 Virtual Memory Address Register

The VMA is loaded during an EBox request and remains latched until the MBox responds (Figure 2-61). The VMA is a 23-bit register that accepts input from a double mixer arrangement. Thus, the incrementing or decrementing is performed in the register itself. When both VMA SEL 2 and 1 are clear, the lower mixer is enabled into VMA. The level VMA ← AD selects AD as input. The default is VMA AD as input.

In general, the VMA AD contains one of the following:

PC (18–35)
PC+1 (18–35) + (1)
PC+2 (18–35) + (2)
Process Table Address (27–35)
Fast Memory Address (32–35)

The AD contains one of the following:

Effective Address
@ Word Address
Some Special Address

The VMA Held register is loaded during each MBox memory request [MEM 02 (1)]. The left-most 12 bits of VMA Held are loaded with the request qualifiers, type of paging, context of the reference, and various other signals asserted during the request. The right-most 23 bits of VMA are preserved in VMA Held right. The contents of VMA Held are used during KL Paging mode to buffer the request state while the page fault handler sets up an MBox Page Refill cycle. This operation is generally described in Subsection 1.2.4.2, KL Style Paging and is described later in greater detail.