10. ARITHMETIC PROCESSOR PAGING

10.1 Introduction

BBN is designing an interface between the KA-10 arithmetic processor and core memory which we call the Pager. This device receives from the APR over a standard memory buss execute, read, write, and read-modify-write requests. It then maps the incoming virtual address into an appropriate real core address (provided the request is legal) and passes the request to the memory modules over another standard memory buss. The mapping requires about 150 nsec.

The paging hardware performs the following functions:

(1) Independently maps each 512-word block (or page) of the 262,144-word virtual address space into an absolute core location, or, if the block is not currently in core, traps to a core managing program.

(2) Provides independent protection for each 512-word page in the read, write, and execute modes.

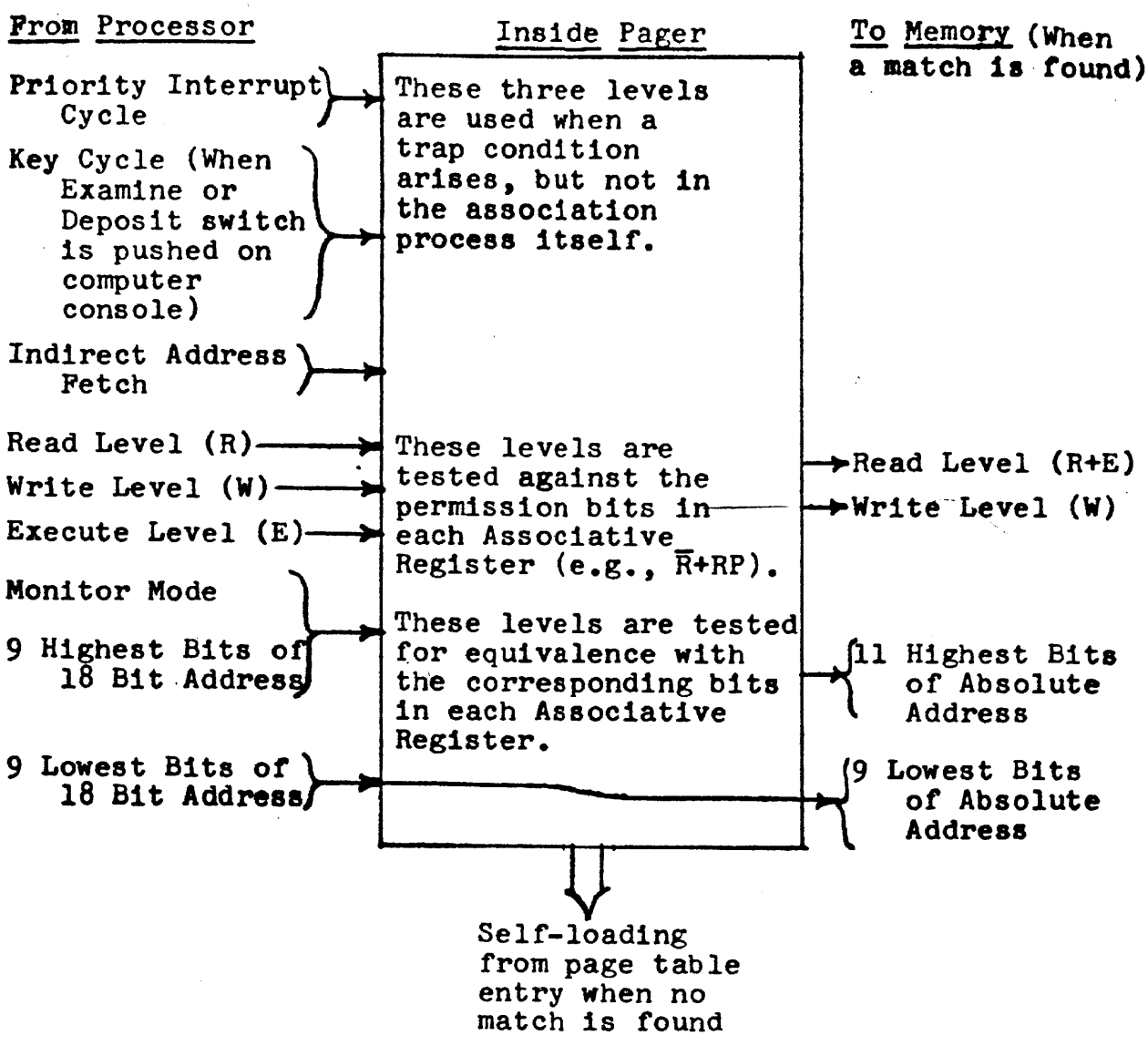
(3) Records statistics in a Core Status Table which are useful to the core management program.

The principal advantage of paging over the current dual-protect-and-relocate scheme incorporated in the KA-10 is that each process is provided with a large, constant 262,144-word virtual machine, yet requires only those pages which are referenced during an interaction to be in core. This can significantly reduce the core memory requirements of running programs.

10.2 Mapping

It is presently impractical to keep mapping information for all 512 pages of a virtual address space in hardware because of the quantity of hardware required. For this reason, a limited number (16 originally, expandable to 32) of associative hardware registers are employed and the mapping information is kept in 512-word Page Tables in core memory. The manner in which virtual addresses are mapped into real addresses is shown in Figure 10-1.

Whenever a page not mapped by the associative registers is referenced, the pager initiates a loading sequence (requiring about three memory cycles) during which the appropriate page table entry is referenced and an associative register loaded with the required mapping information. Associative registers are reloaded in a round-robin fashion. We hold the theory that a program's memory references will be sufficiently "collected" that 16 mapping registers are enough to prevent too frequent reloading.



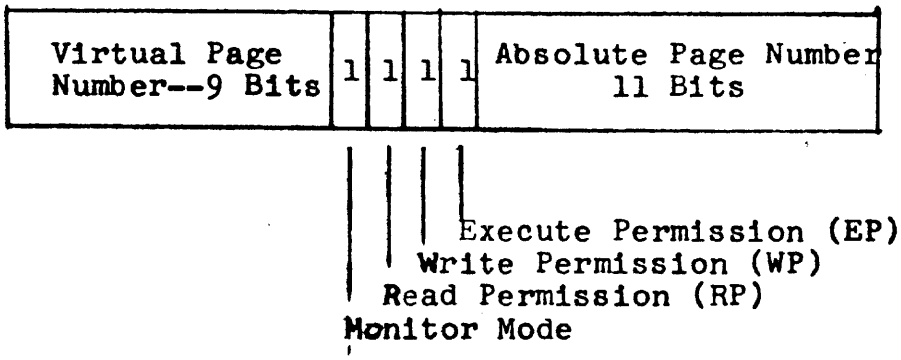


Figure 10-1
Mapping From Virtual to Absolute Addresses

As shown in Figure 10-2, which page table is referenced during the loading sequence depends upon the memory request. There is one page table for user mode requests and three possible partial page tables for monitor mode requests. The locations of the monitor mode page tables are as follows:

First APR's Map	3100_8 to 3177_8	Absolute locations
Second APR's Map	3000_8 to 3077_8	
Common Map	3200_8 to 3677_8	
Private Map	700_8 to 777_8	of the Process State Page

The locations of the user mode page table and of the process state page are loaded into the pager when processes are switched, as described later.

10.3 Switching Processes

The IOB reset pulse generated by CONO APR,2ØØØØØ causes the pager to clear itself. Thereafter, the pager may be switched from one process to another by first storing an appropriate word in location 71_8 and then executing CONO 24, xxx. The pager does not interpret any of the effective address bits of the CONO and will not respond to any other I/O instructions (except IOB reset). When CONO 24, xxx is received, the pager will complete the page number dump for the old process (described later), clear itself, and reload its base registers from 71_8. This is shown in Figure 10-3.

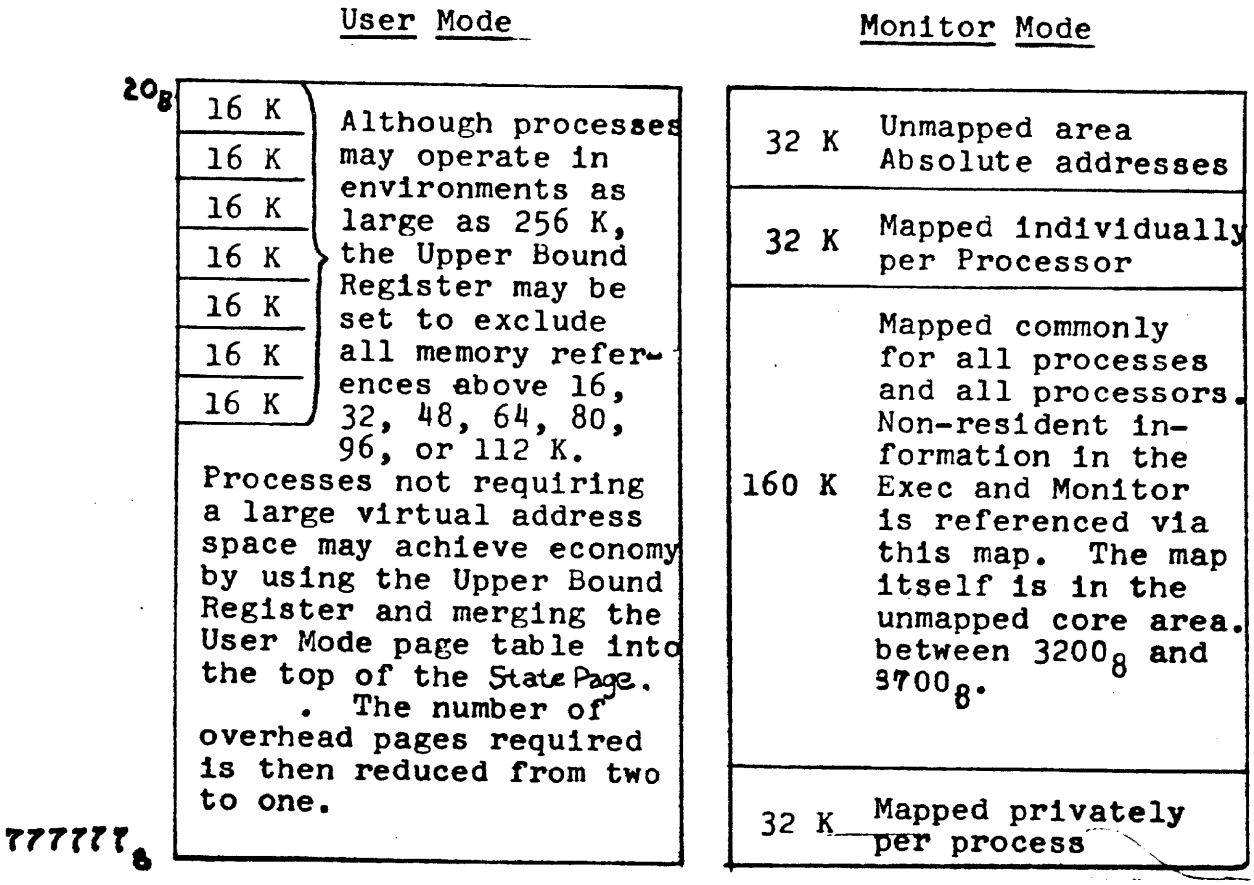
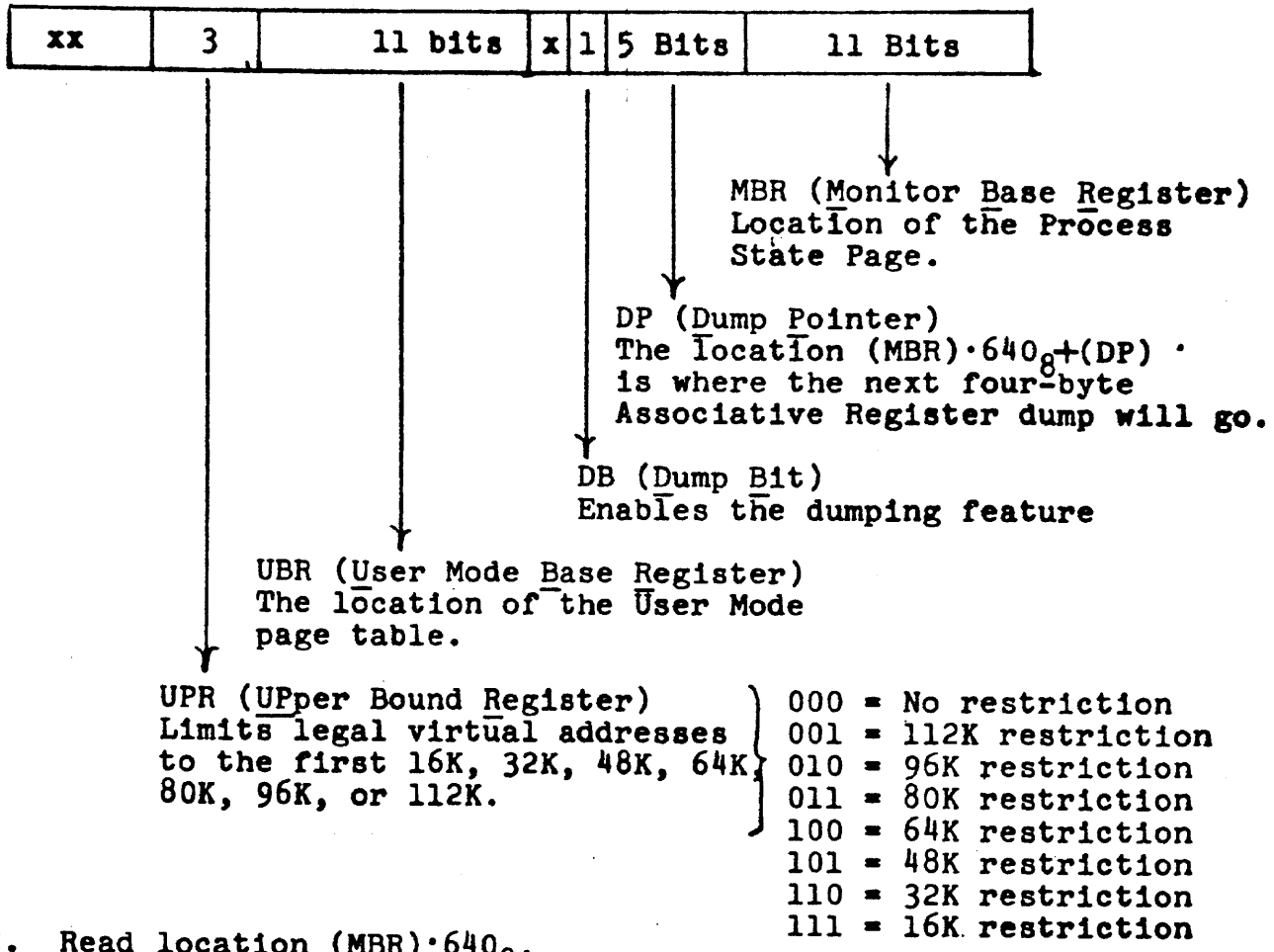


Figure 10-2
Mapping of Virtual Addresses

CONO 24, xxx causes the Pager to reload its main registers as follows:

1. Read absolute location 71_8, interpreted as below.



2. Read location (MBR)·640_8.

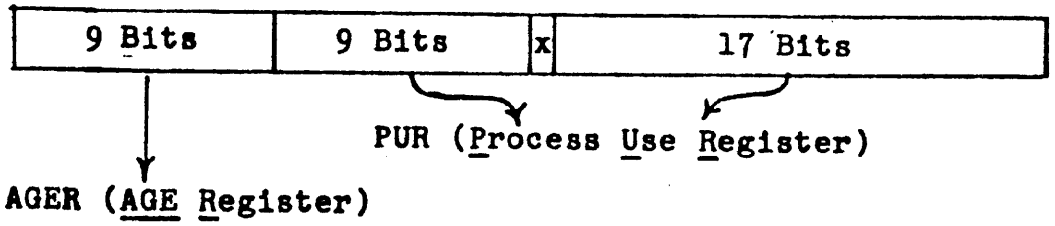


Figure 10-3
Initializing the Pager For a New Process

10.4 The Self-Loading Sequence

When a memory request is not mapped by an associative register, the pager enters its self-loading sequence. This consists of the following steps.

(1) Reading an appropriate page table entry.

(2) Checking read, write, and execute protection bits for access violation.

(3) Checking the directed trap bits for trap conditions.

} Figure 10-4

(4) Tracking down shared and indirect pointers to determine the absolute address of the page and trapping if the page is not in core.

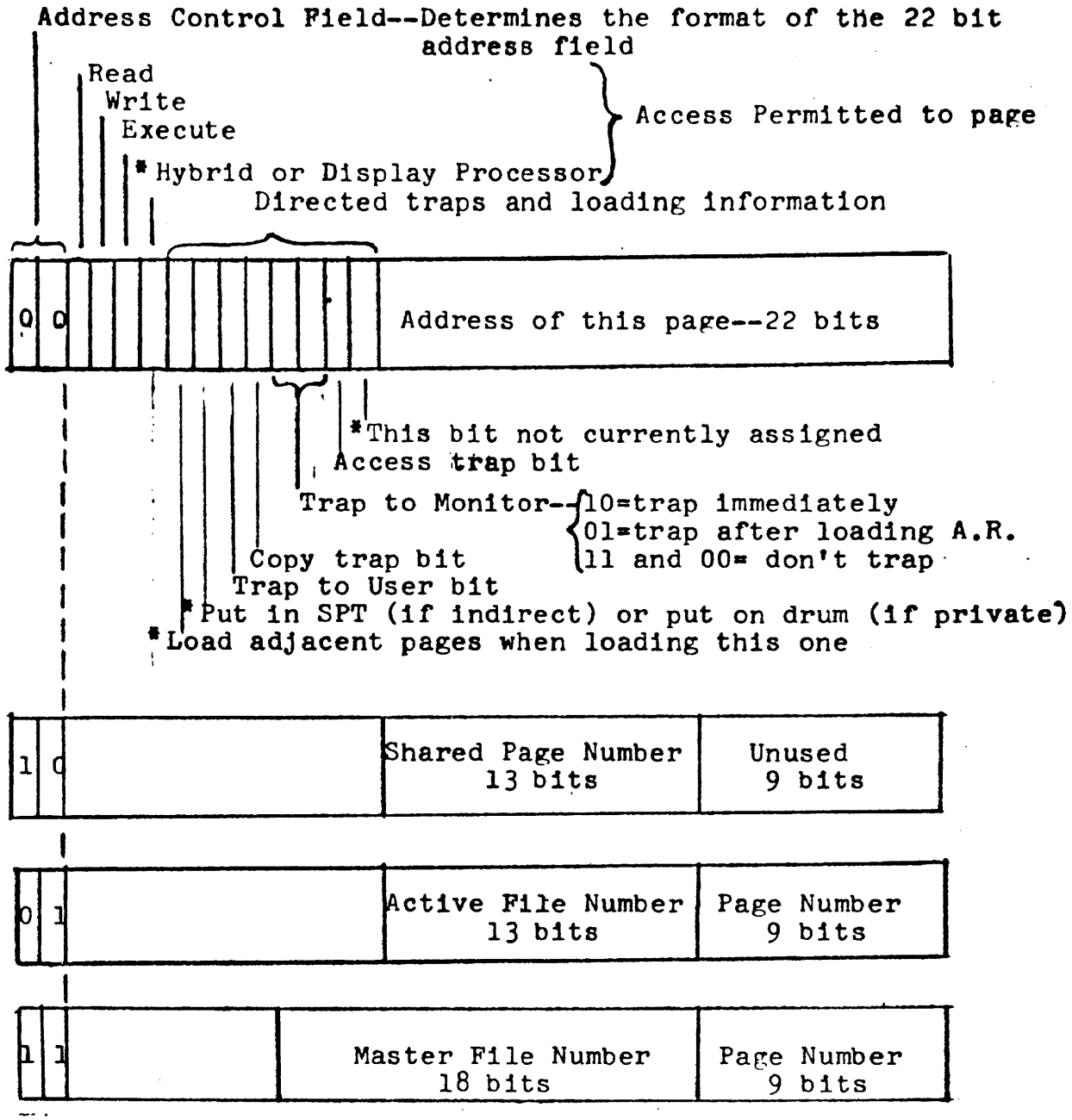
} Figure 10-5

(5) Modifying the Core Status Table entry corresponding to the core address of the page and trapping if the page field is less than 20_8.

} Figure 10-6

(6) Dumping the last three and the current nine-bit virtual page numbers every fourth time a user-mode page is loaded into an associative register (only when the dumping feature is enabled). These four nine-bit bytes are stored in the Process State Page at location 640_8 + contents of the dump counter register. The dump counter is a 28-position counter that assumes values between 4 and 37_8, so that the dump is actually into the region 644_8 to 677_8.

Dumping a record of the last 112 user-mode pages loaded into the associative registers (not necessarily all different pages, however, because the pager may reload the same page several times). By means of this record the working set of a process may be swapped in quickly following rescheduling.



This last pointer format should not be encountered by the paging hardware; it is used only on inactive files. The Monitor converts entries of this type to shared or indirect pointers when the file is activated.

Figure 10-4
Format of Page Table Entries
I. Private pointers
II. Shared pointers
III. Indirect pointers
IV. Indirect file pointers

* These bits are ignored by the pager.

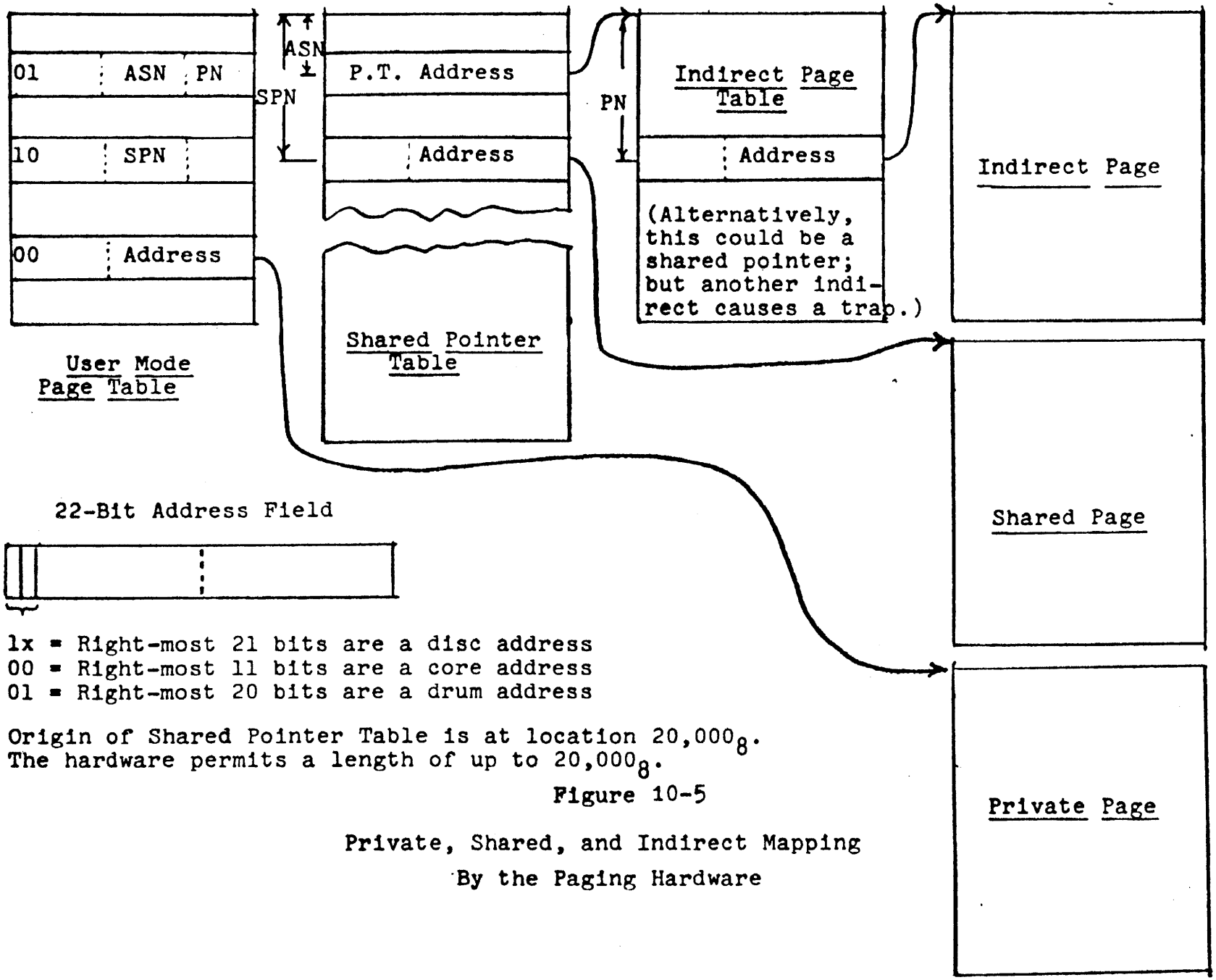
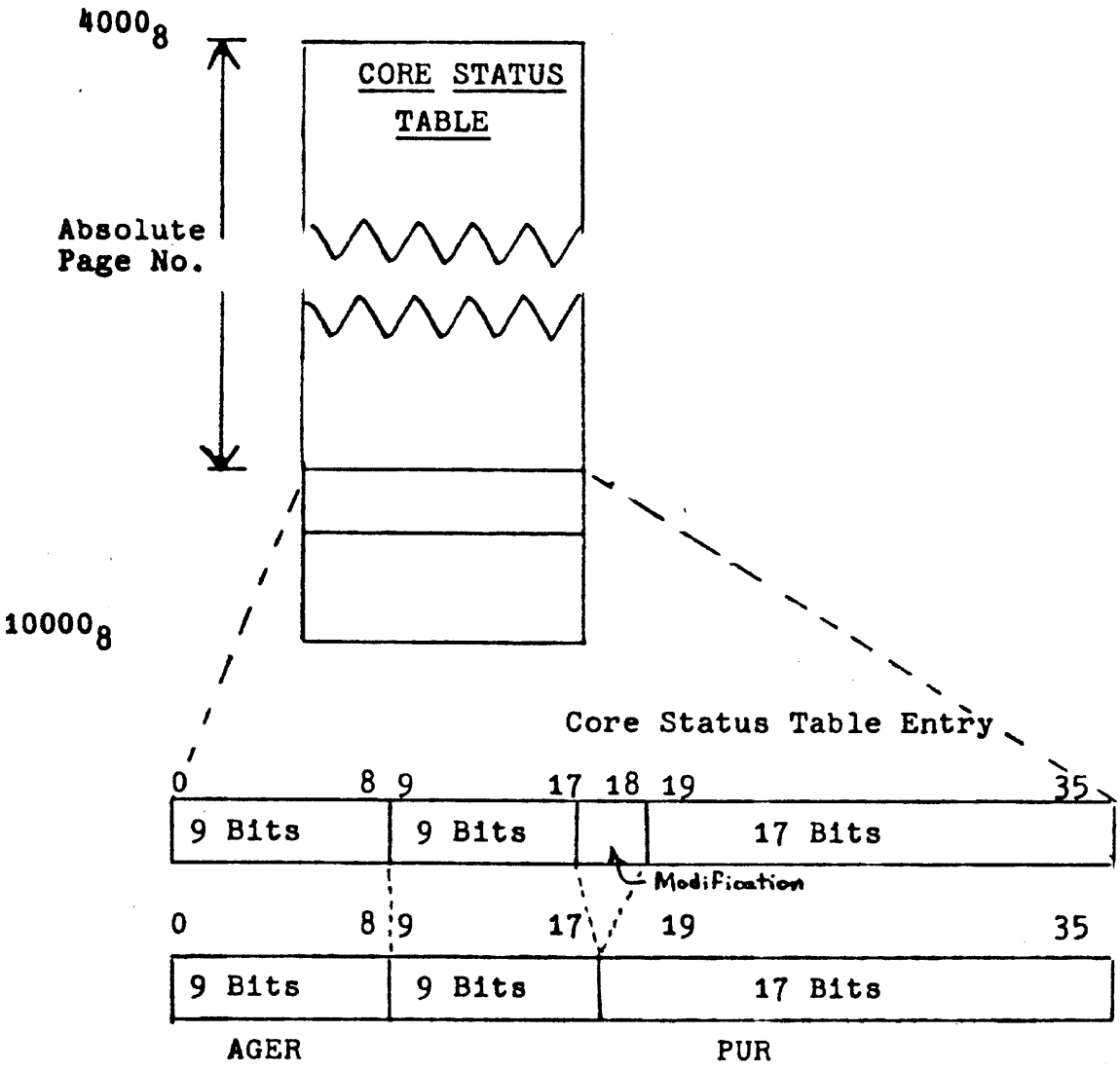


Figure 10-5

Private, Shared, and Indirect Mapping
By the Paging Hardware



1. Bits Ø-8 of AGER replace Bits Ø-8 of CST entry.
2. Bits 9-17 and 19-35 are "ORed" into Bits 9-17 and 19-35 of CST entry.
3. Write request level is "ORed" into Bit 18 of the CST entry (if the write is permitted)

Figure 10-6

Pager References to the Core Status Table

If no trap condition occurs during self-loading, an associative register is loaded with the required mapping information and the pager proceeds. Note, however, that the write permission bit in an associative register is set only when both the modification bit in the Core Status Table and the write permit bit in the page table entry are set.

10.5 Pager Traps

A paging trap will occur whenever one of the following events happens:

1. The trap bits in the process page table force a trap.
2. The addressed page (or indirecting page table) is not in core.
3. An illegal condition is detected.
4. The Core Status Table entry for an addressed page contains an age less than 20_8 (meaning that the core manager is controlling the page).

When one of the above conditions happens, the pager first stores the cause and location of the trap into the Process State Page at location 642_8. The format of this word is shown in Figure 10-7. Then, if the APR operation in progress was a write, it stores the data into location 643_8 of the state page. Finally, it forces the APR to execute absolute location 70_8. Locations 70_8 should contain a JSYS instruction to a trap routine.

The arrangement of the Trap Cause field was chosen so that decoding of the cause could be easily accomplished by the JFFO instruction.

To restart a process which was terminated by a pager trap, the following information is of value:

1. The program counter (PC) saved by the JSYS in location 70_8 is correct.
2. If the read or execute bits in 642_8 of the state page are set, restart is completed by performing a JRSTF to the first location of the trap routine.
3. If the read bit is not set but the write bit is set in 642_8, the data in 643_8 must be written into the effective address of 642_8 before returning control to the process via JRSTF.

A sample program to restart a process which was terminated by a paging trap is given below:

```
            CONO 24,0           ;LOAD PAGER WITH NEW BASE REGISTERS
                                ;UPPER BOUND REGISTER, AND DUMP REGISTER
            MOVE 1,777642       ;GET TRAP STATUS WORD
            TLNE 1,12           ;SKIP IF NEITHER READ NOR EXECUTE BITS SE
            JRST BEGIN
            MOVE 777643         ;GET DATA WORD
            TLNE 1,1            ;SKIP IF USER MODE
            JRST MONWR
            UMOVEM (1)          ;COMPLETE USER MODE WRITE
BEGIN:      HRLZI 17,777620     ;RESTORE AC'S FOR THE PROCESS
            BLT 17,17
            JRSTF@777641        ;RESUME PROCESS (JSYS IN LOCATION 70
                                ;SAVES THE FLAGS AND PC IN 777641).
MONWR:      MOVEM (1)           ;COMPLETE MONITOR MODE WRITE
            JRST BEGIN
```

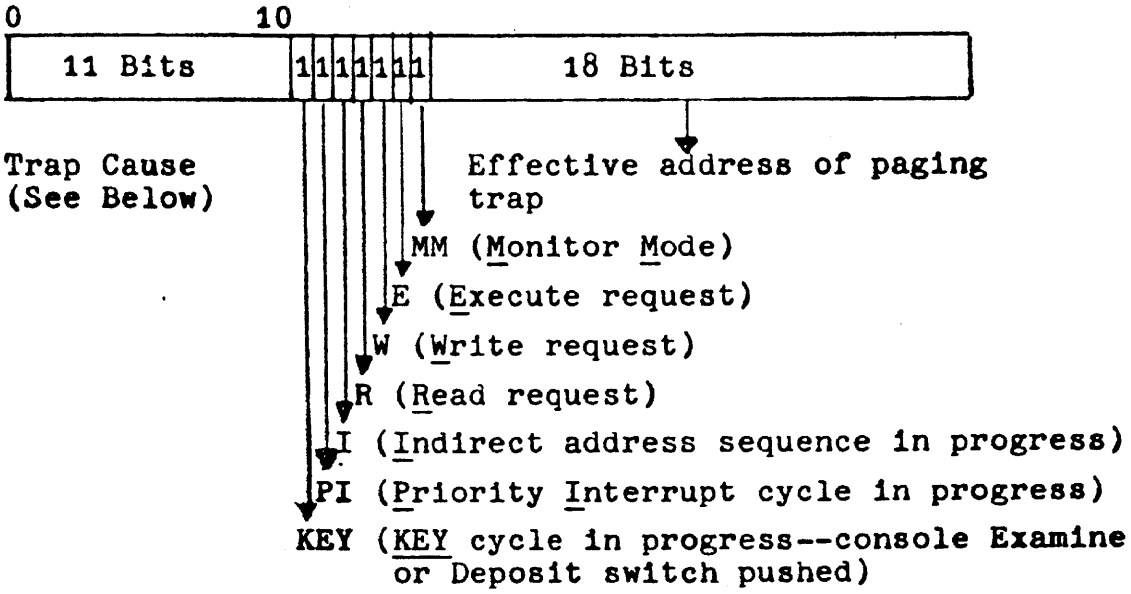
In Figure 10-7 the function of the PI cycle, Key Cycle, and Indirect Address Sequence bits may not be clear. A paging trap should not occur during a PI cycle, if the time-sharing software is coded properly since it is impossible to recover from such a trap. The function of this bit is to notify the software that a disaster has occurred so that crash recovery may be attempted. Key Cycle traps will not occur very often under time-sharing, but we would like to be able to handle them since someone may wish to examine or deposit into the non-resident portion of the Monitor's address space. The indirect address sequence bit is used to distinguish data reads of non-existent memory from indirect addressing reads of non-existent memory. In the former case, the software may wish to create new memory and proceed, but in the latter case, a programming error has been made.

10.6 Core Management Philosophy

Core management for processes with large virtual address spaces can prove to be a significant problem. To minimize difficulties we have designed into the Pager several features (Figure 10-6).

(1) Recording modification
Because a write request will set the modified bit in the Core Status Table, core-to-drum swaps need be performed only for pages with this bit set.

TRAP STATUS WORD



Interpretation of the Trap Cause Field:

Ø & 1 Copy Trap (Bit 9 set in Page Table entry
2 User Trap (Bit 8 set in Page Table entry)
3 Immediate Monitor Trap (Bits 9-10=10 in Page Table entry)
4 Illegal Read (Bit 2=0 in page table and read request)
5 Illegal Write (Bit 3=0 in page table and write request)
6 Illegal Execute (Bit 4=0 in page table and execute request
7 Access Bit not set (Bit 12=0 in page table entry)
8 Upper Bound Register Exceeded
9 Memory request not serviced within 70 μsec

1 Indirect & 2 Private } Not in core
& 3 Shared }
& 4 Page Table }
& 5 Excessive (Double indirecting is not permitted)
& 10 Illegal

2 Private } Not in core
3 Shared }
4 On Replacement Queue (Bits Ø-6=Ø in Core Status entry)
5 Drum Write Requested (Bits Ø-6=1 in Core Status entry)
6 Not yet in core (but swap-in requested)
(Bits Ø-6=2 in Core Status entry)
7 Blocked by core manager (Bits Ø-6=3 in Core Status entry)
8 Monitor after-loading trap (Bits 9-1Ø=Ø1 in page table entry)
9 Parity error
1Ø Illegal pointer format (Bits Ø-1=11 in page table entry)

Figure 10-7

Trap Status Word Format

(2) Identifying Processes using a page
By loading the process use register (PUR) with a bit identified with the current process the Core Status Table entry for each page will contain a record of the processes which have used each page, the core management program can then discriminate pages in use by active processes from those which were used by processes now inactive.

(3) Marking time-of-last-reference
When a large process is compute bound, its "working set" will frequently change with time. By periodically incrementing the age register, the core manager can look at the Core Status Table and distinguish recently referenced pages from ones not referenced for a long time. The latter are likely to be outside current working sets and are good candidates for replacement by new pages.

(4) Recording the working set of a process
By examining the nine-bit page numbers dumped by the pager into the region 644_8 to 677_8, the core manager can make a good guess about the current working set of the process. If the process was rescheduled and swapped out of core, it may be reasonable to preload these pages into core before next scheduling the process.

(5) Provide for dual-processor operation
Because one APR may be computing for a process, while the other APR is tampering with the Core Status Table, we have included a special check in the paging hardware for ages less than 20_8. When the software is about to examine some entry in the Core Status Table, it may EXCH a word with the left-most five bits = Ø for the Core Status Table entry. This will prevent the other APR from inadvertently loading this page during the examination.

10. ARITHMETIC PROCESSOR PAGING

10.1 Introduction

BBN has implemented a device called the BBN Pager which is connected between the KA1Ø (PDP-1Ø arithmetic processor) and the KA1Ø's memory port. In conjunction with a set of hardware modifications to the KA1Ø, the BBN Pager changes the core memory mapping mechanism such that core memory is allocated and protected in 512 word pages. The address space of the machine is mapped for EXEC mode as well as USER mode with the BBN Pager. The paging mechanism can be bypassed by executing a specific CONO to the pager or by executing (or depressing) IOB reset to invoke a so called "transparent mode" for the running of the standard DEC monitors or diagnostic software.

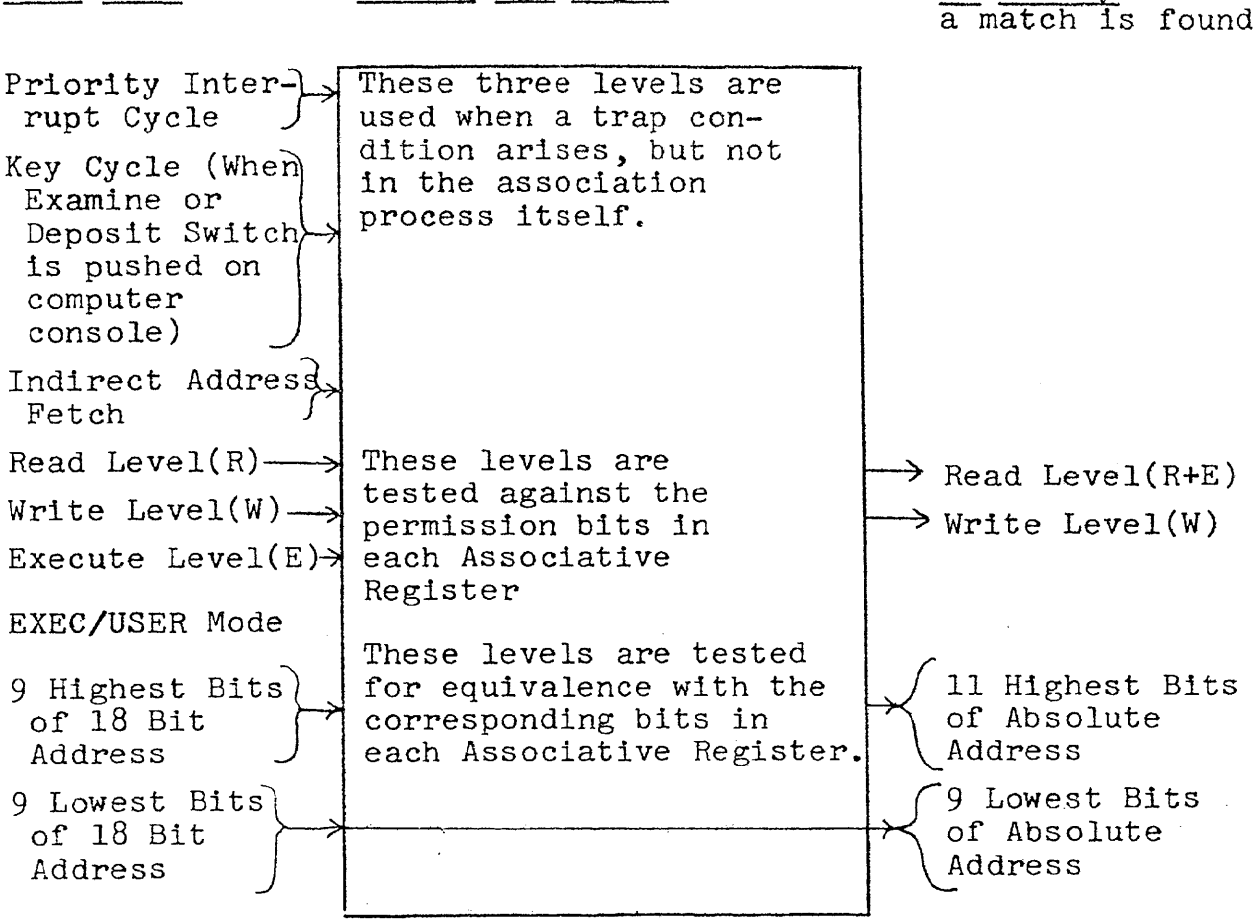
10.2 The Associative Mapping Process

When mapping is enabled in the pager, the 9 high order virtual address bits, state of the EXEC/USER mode flip flop, and type of request (read, write, execute) from the KA1Ø are compared and tested with the contents of 1 to 54 (depends on pager configuration) associative registers. This comparison is performed on all associative registers simultaneously. If a match is found, the particular associative register containing the match also contains 11 bits which become the high order 11 bits of the real core address (hereafter abbreviated as R.C.A.).** The use of 11 R.C.A. high order bits permits the KA1Ø to reference up to 1024K* words of memory. In this simple case, the overall delay directly attributed to the pager is about 100 nanoseconds plus cable delay. The simple case is represented by Figure 10-1.

---

*K=1024 words

** A glossary of terms is presented at the end of this section



Bits in Associative Registers

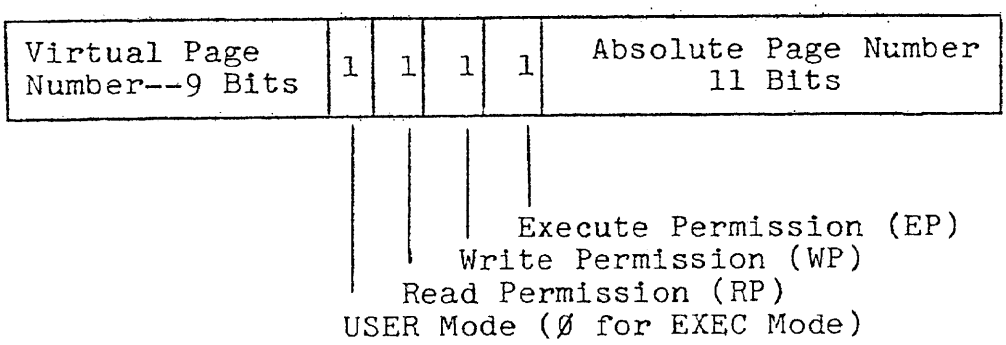


Figure 10-1
Mapping From Virtual to Absolute Addresses
(Simplest Case, Match Found in Associative Register)

10.3 USER Mode Mapping when the Association Fails

When a match is not found in an associative register (hereafter abbreviated as A.R.), the pager begins a self-loading sequence which basically involves the loading of an A.R. from information found in one or more tables in core memory. The particular A.R. to get self-loaded is determined in a very simple cyclic fashion. That is, if the last A.R. loaded was A.R. 5, the next to be loaded will be A.R. 6(if it exists; if not, the next existing A.R. in the cyclic sequence is used) ... The average pager is configured with 16 A.R.'s which means that if the program confines most of its references to an 8K or less working set, self-loading will be invoked infrequently.

The first stage of the self-loading sequence involves reading some information from a table in core memory called the page table (hereafter abbreviated as P.T.). This table is 512 words long and is itself a page which may be anywhere in core memory. The origin of the P.T. is specified by the contents of a register in the pager called the User Mode Base Register (hereafter abbreviated as U.B.R.). The 11 bits of the U.B.R. are used as the 11 high order address bits and are concatenated with the original 9 high order address bits (on which the association failed) to reference the appropriate word in the P.T.

The word which is read from the page table is of one of four types as determined by bits Ø and 1 of the word. These types are:

ØØ private page

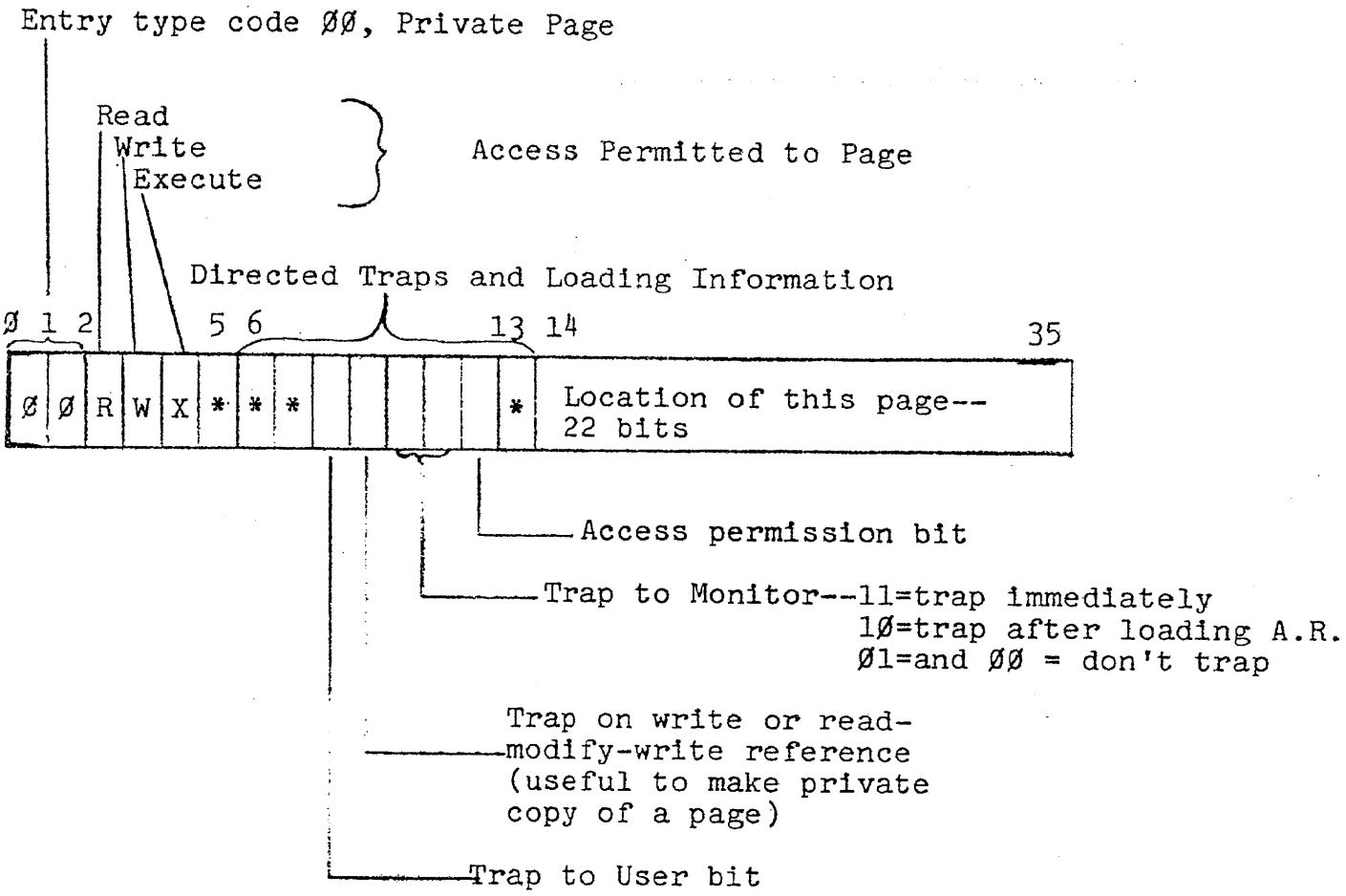
Ø1 shared page pointer

1Ø indirect page pointer

11 illegal format

10.3.1 Private Page

In the simplest case of a private page type, the rightmost 11 bits contain the high order R.C.A. bits and bits 2-4 contain the read, write, execute access information which are used to self-load an associative register. The private page entry has many options which are detailed in Figure 10-2.



*not used by Pager hardware

Figure 10-2, Private Page Entry in P.T.

Most of the option bits cause traps to occur which cause the KA1Ø to take some special action when a page is referenced in a particular way.

10.3.1.1 The Location Field

The location field is 22 bits wide. This permits the specification of where a page really is in primary, secondary, or even tertiary storage. If bits 14-17 are all Ø's, the right most 11 bits contain the high order R.C.A. bits. If any of bits 14-17 are set, a page not-in-core trap will be invoked which will cause the KA1Ø to take special action.

10.3.1.2 Limiting the Size of the User Address Space

The pager contains a register called the Address Limit Register (A.L.R.) which is capable of restricting the legal USER mode virtual addresses to the first 16K, 32K, 48K, 64K, 80K, 96K, 112K or the entire 256K.

10.3.2 The Core Status Table

Whenever an associative register is successfully loaded, a word in the core memory status table is updated. This table contains an entry for every page of real core in the system. An entry contains information about that page related to: the relative amount of time the page has been in core (contained in a 9 bit pager register called A.G.E.R.--AGE Register), whether the page has been written into (the modification bit), and which processes (of a subset of all processes existing in the system) have referenced the page. The particular processes referencing a page are identified by bits in the process use field of the entry. These bits are updated by the contents of a 26 bit pager register called P.U.R.--Process Use Register.

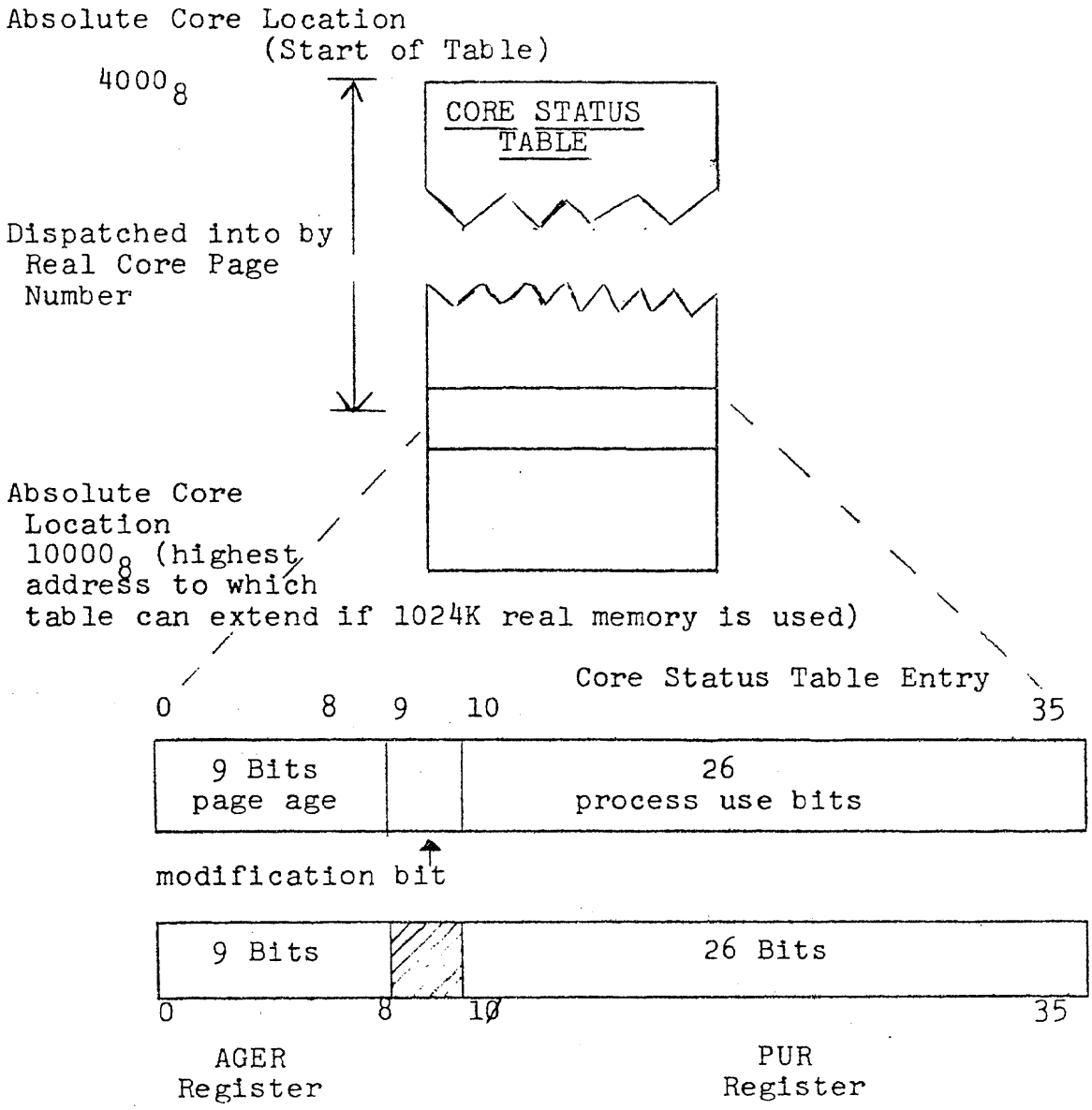
The use of the core status table (C.S.T.) causes one additional read-modify-write cycle of overhead (while referencing the C.S.T.) in the self-loading sequence.

The modification bit and write permission bits at the A.R. are handled in a slightly complicated way. The modification bit is set only if the memory request which initiated the loading of an A.R. was a write or read-modify-write cycle. The modification bit is never cleared by the pager. If an A.R. is loaded due to a non-write request, the write permission bit of the A.R. is not set regardless of whether writes are permitted by the page table entry unless the page has already been modified (indicated by the modification bit already set from some previous operation) and write permission is specified by the page table entry.

(A.R. write permit ← PT write permit ∧ (WRRQ ∨ modification bit)
new modification bit ← old modification ∨ (WRRQ ∧ PT write permit))

As special aid for the control of shared pages, a pager trap is generated if the three high order bits of the page age field in the C.S.T. entry are all Ø's. This provides a way for the core manager to defer use of a particular real core page while the page's state is being tested or changed.

The core status table starts at absolute real core location $4ØØØ_8$. A diagram of the table and its use by the pager is shown in Figure 10-3.



1. Bits Ø - 8 of AGER replace Bits Ø - 8 of CST entry.

2. Bits 1Ø - 35 are "ORed" into Bits 1Ø - 35 of CST entry.

3. Write request level is "ORed" into Bit 9 of the CST entry (if the write is permitted).

Figure 1Ø-3

Pager References to the Core Status Table

10.3.3 Shared Page Pointer

The Shared Page Pointer entry in the P.T. is used for the most commonly shared pages in the system. This type of entry is detailed in Figure 10-4.

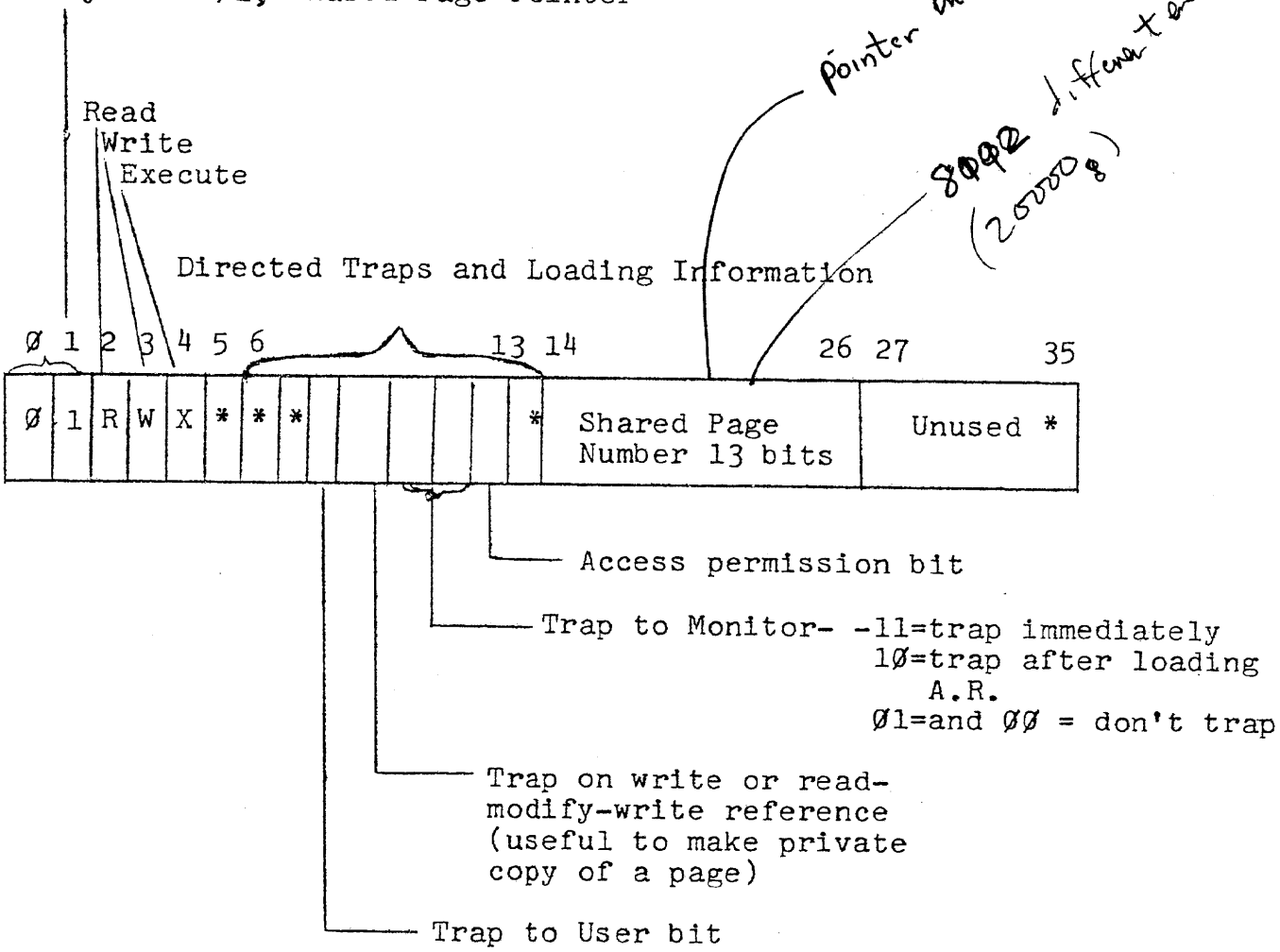


Figure 10-4 Shared Page Pointer

The Shared Page Number/field is used as a dispatch into the Special Pages Table (S.P.T.) which starts at absolute core location $20,000_8$. The contents of the specified S.P.T. entry contains the page location information in bits 14-35 in the same format as a private P.T. entry bits 14-35. (see section 10.3.1.1).

10.3.4 Indirect Page Pointer

The Indirect Page Pointer entry in the P.T. is used for uncommonly shared files or processes or for indirectly referencing the dynamic address space of a file or process which is expected to change. This type of entry is detailed in Figure 10-5.

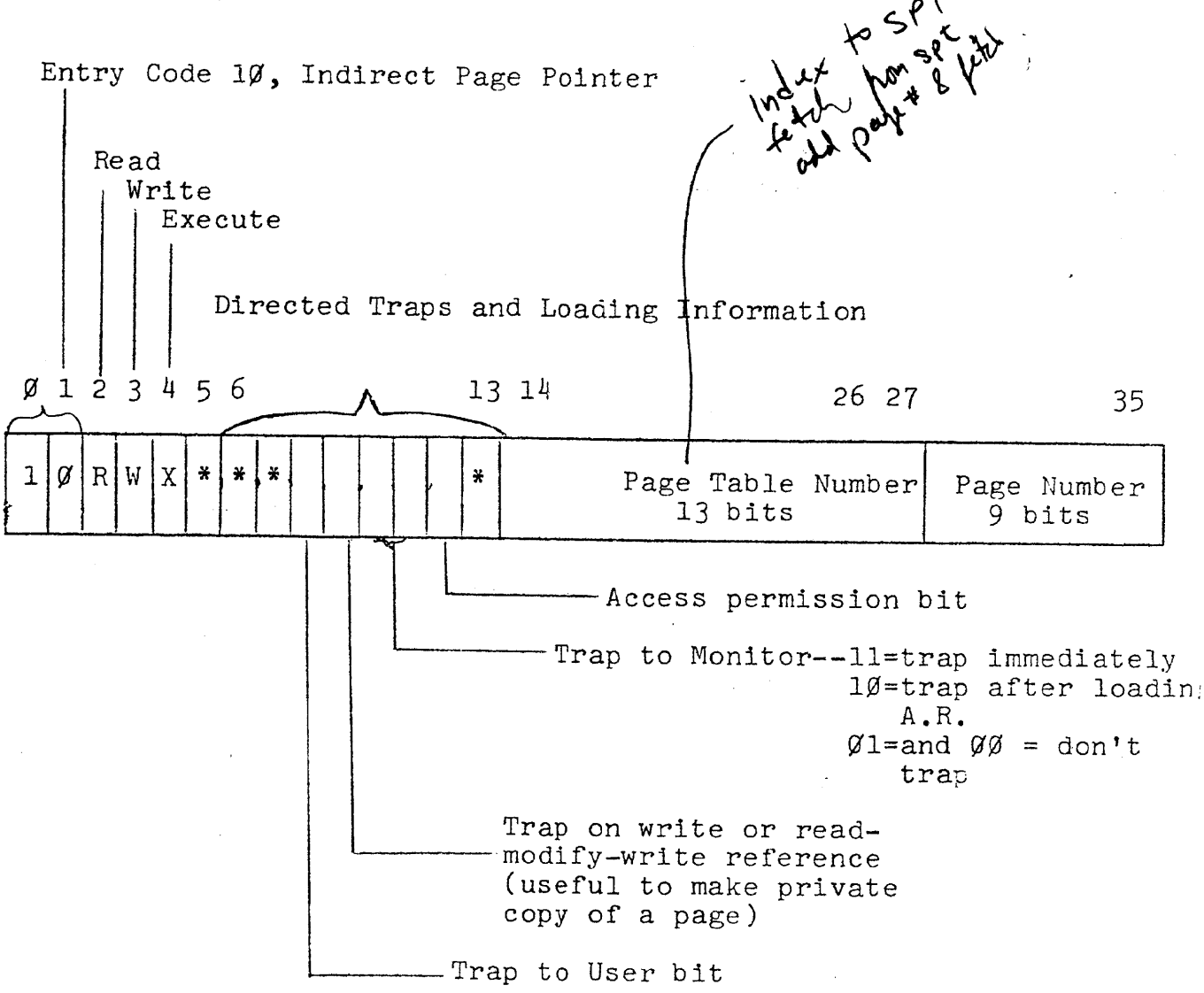


Figure 10-5 Indirect Page Pointer

The Page Table Number (P.T.N.) is used as a dispatch into S.P.T. to fetch an entry which contains the location in bits 14-35 (see section 10.3.1.1) of the indirect page table. The Page Number Field of the Indirect Page Pointer is used as a dispatch into the Indirect Page Table. The specified entry of this table can be any of the three page table entry types just described. An attempt to use indirect page pointers to a depth of more than 2 will result in a pager trap.

The access permission finally granted via Indirect Page Pointer mapping is the "AND" of the R,W,X bits and other access permissions starting with the first Indirect Page Pointer down through all P.T. entries until the destination page is found. This generally results in a reduction of the final access granted.

10.3.5 Summary of P.T. Entry Types

All three page table entry types have the virtue that the actual location of a page is kept in only one place instead of being replicated in many page tables (for example). Detailed examples of the three P.T. entries are presented in Figure 10-6.

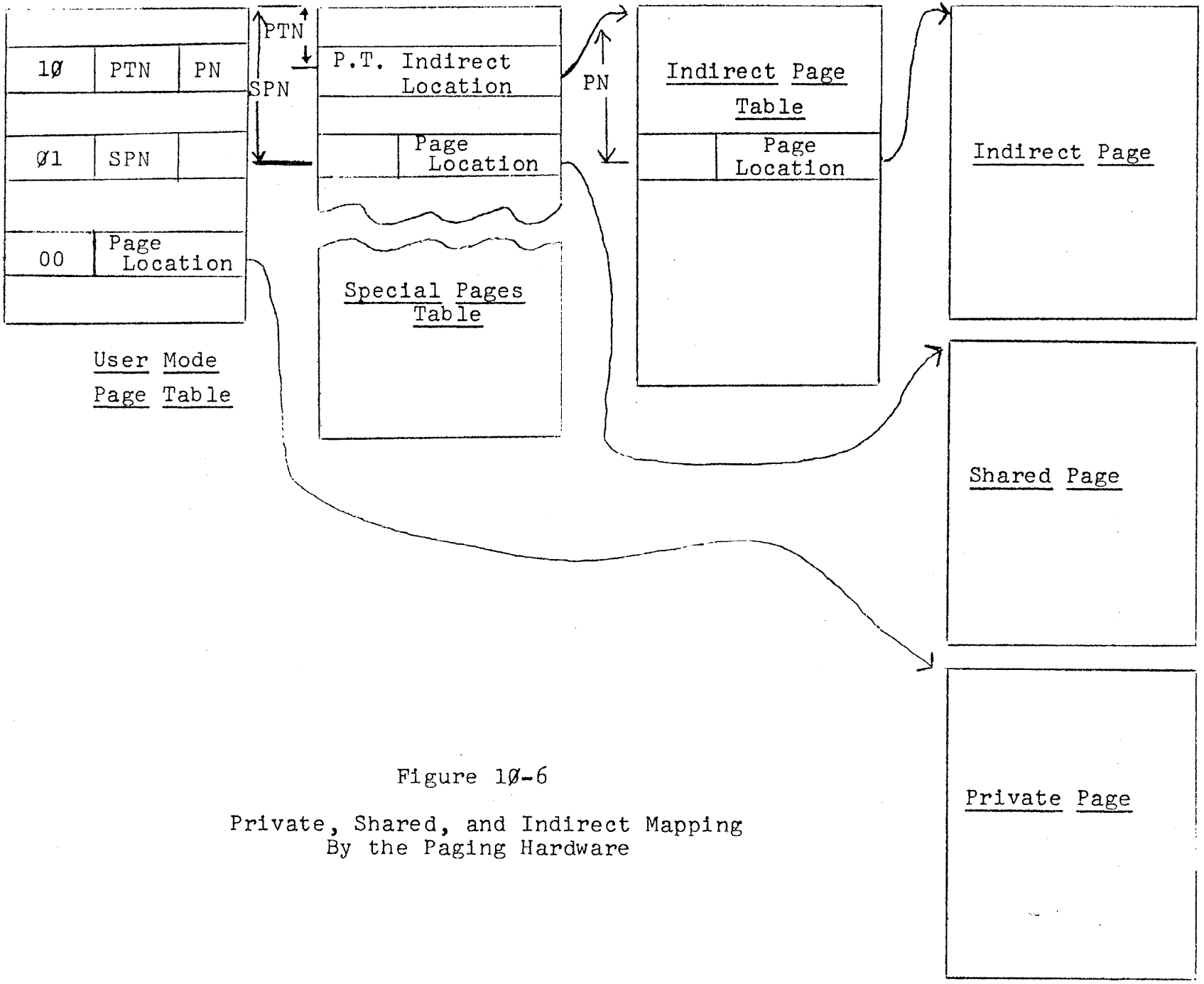


Figure 1Ø-6

Private, Shared, and Indirect Mapping
By the Paging Hardware

10.4 EXEC Mode Mapping

EXEC Mode mapping is really quite similar to USER Mode mapping. The major difference being that four distinct areas of the EXEC Mode address space are separately mapped and mapping of the "Resident Monitor" is separately enabled. These four areas are shown in Figure 10-7.

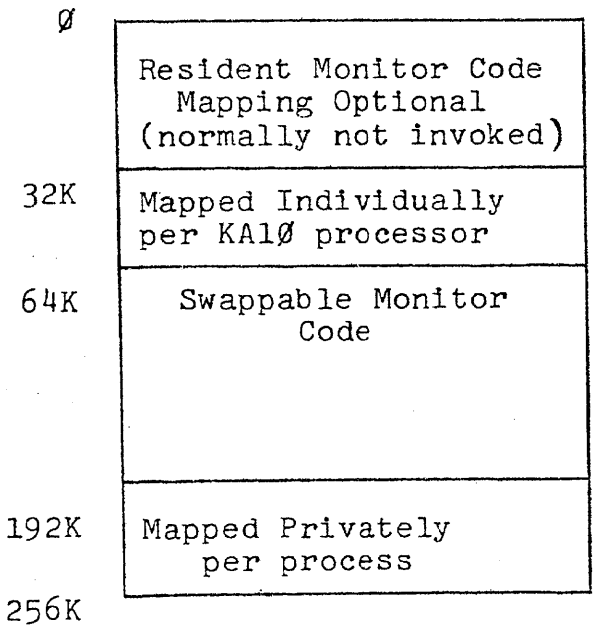


Figure 10-7 Exec Mode Address Space

The associative mapping process is exactly the same for EXEC mode as USER mode. However, most of the page table (for the self-loading sequence) for the EXEC mode address space is fixed in absolute addresses. Namely:

Optional Resident Monitor Map (Not used unless Resident Monitor Mapping is turned on)	$3ØØØ_8$ to $3Ø77_8$	absolute Real Core locations
First KA1Ø's Map	$31ØØ_8$ to 3177_8	
Second KA1Ø's Map	$37ØØ_8$ to 3777_8	
Common Swappable Monitor Map	$32ØØ_8$ to 3577_8	

The area that is mapped privately per process is actually mapped by an area of the special overhead page associated with each process called the Process Storage Block (P.S.B.). The address of the P.S.B. is specified by the contents of an 11 bit pager register called the Monitor Base Register (M.B.R.). Only the highest 128 words of the P.S.B. are used for mapping purposes. The remainder is used for process specific temporary storage, stacks, and 2 more words are used by the pager. Thus locations 600_8 - 777_8 of the P.S.B. map the highest 64K of the EXEC mode address space.

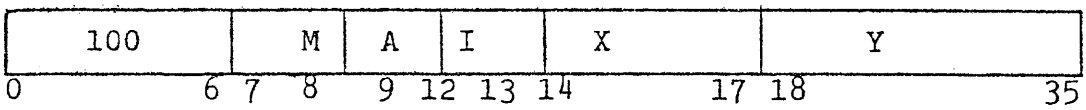
10.5 Invoking the USER Address Space With the KA1Ø in EXEC Mode

There are two classes of instruction modifications which were made to the KA1Ø to enable the system to make references to parameters in the user's address space when the machine is in EXEC mode.

10.5.1 UMOVEx

The first class of instruction modifications is the UMOVEx set which forces MOVEs to and from USER space.

UMOVE User Map Move



Move one word from the source to the destination specified by M, using the user address map. The source is unaffected, the original contents of the destination are lost.

UMOVE	User Move	100
UMOVEI	User Move Immediate	101
UMOVEM	User Move to Memory	102
UMOVES	User Move to Self	103

These instructions provide a convenient way for the monitor to invoke the USER address mapping to fetch or store information into the USER address space (UMOVE or UMOVEM). UMOVEI provides a way for the monitor to do address computation using indirect addressing through the USER address space. Of course, indexing and AC references are not affected by the choice of USER map/EXEC map. However, addresses which indirect through the USER AC's are handled specially (see section 10.5.3).

10.5.2 XCT AC, E

The next instruction change is a modification to the XCT instruction to use the AC Field (formerly ignored) to affect which map is used.

The AC field is interpreted as shown in Figure 10-8. If the specified bit is on, use of USER address space is forced for any of the conditions indicated.

XR = 4
XW = 1
XRW = 5
XLB = XDB = 2

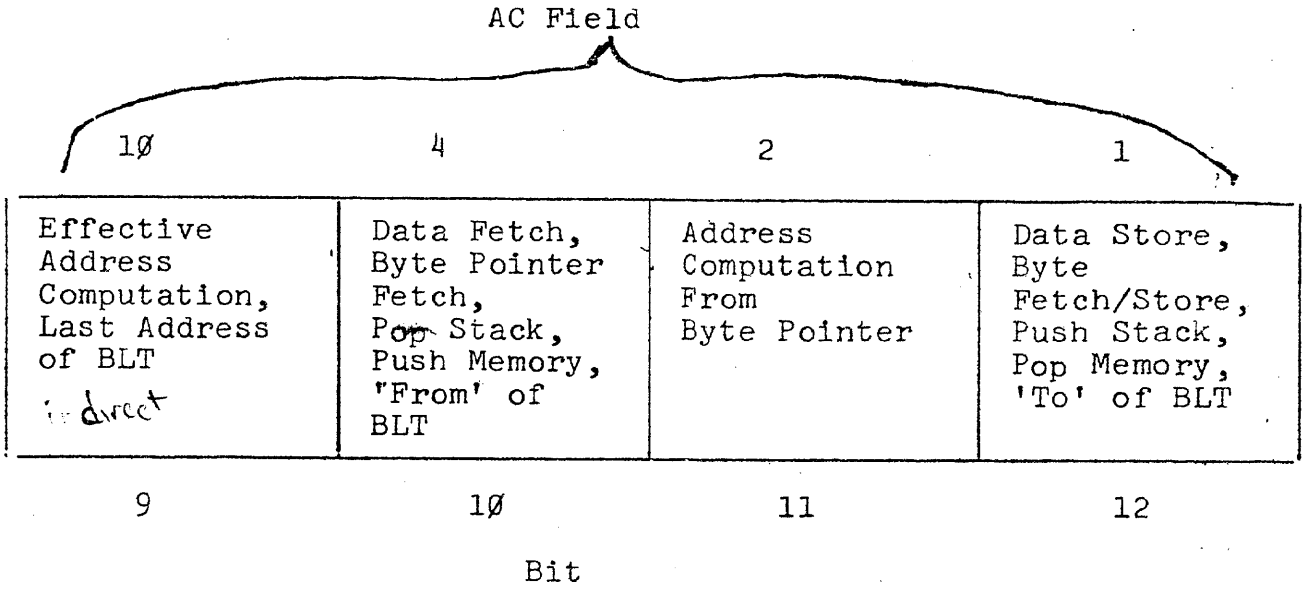


Figure 10-8 XCT AC bits

The instruction to be executed is always fetched from monitor space. To BLT a data block from a user location specified in AC left,

```
        HRRI AC, FIRST
        XCT 4, [BLT AC, LAST]
             ...
FIRST:  BLOCK N
        LAST = .-1
```

To BLT a data block into a user region specified by the first and last user locations in AC left and right respectively,

```
        HRRM AC, INSTR
        HLR AC, AC
        HRLI AC, FIRST
        XCT 11, INSTR
             ...
```

INSTR: BLT AC, Ø

FIRST: BLOCK N

XCT 15, INSTR can BLT data from one place to another in the user's address space. (Useful for zeroing out a region)

To transfer a series of bytes specified by a user byte pointer in AC,

LP: XCT 3, [ILDB AC2, AC]
(or [IDPB AC2, AC])

UMOVEx AC, E is equivalent to XCT 15, [MOVEx AC, E] .

10.5.3 Call From Monitor Flag (PC Flag Bit 7)

Bit 7 of the PC flags word is used to store the state of a flip flop named CALL FROM MONITOR. This bit is saved and restored in the same fashion as the other PC flag bits. It is cleared by MR START, and set whenever an EX JSYS (effective address<1000) is executed in EXEC mode. This bit indicates to the called JSYS routine that effective addresses, byte pointers, and BLT pointers passed as arguments should refer to the EXEC mode address space, not the current USER address space. When this bit is on, special XCT and UMOVEX references are automatically forced into the EXEC mode address space instead of the USER's space.

This feature simplifies the coding of EX JSYS routines which accept pointers as arguments and which may be called either from USER mode or EXEC mode. The routine merely makes use of any pointers with UMOVEx, or special XCT instructions, and the CALL FM MON flag automatically forces references into the correct address space.

10.5.4 Forced References to USER Space Locations $\emptyset$-17_8

A 5 bit AC BASE REGISTER exists in the pager to provide an independent mapping mechanism for saved accumulators. This special mapping process to reference saved AC's is invoked by forced references to USER space (UMOVEX or special XCT) with addresses $<20_8$.

During the mapping process, the low 4 bits are taken from the virtual address, the next 5 bits (27-31) are supplied from the AC BASE REGISTER, the top 9 bits (18-26) are forced to 775, and EXEC mode addressing is forced. This intermediate virtual address is then passed to the pager for mapping in the standard fashion. This means that the saved accumulators are mapped into one of 32 blocks, (selected by the AC BASE REGISTER) each 16 words long, located in page 775 of the EXEC mode virtual address space. (Recall this page is mapped privately per process).

This space is ordinarily used as a stack of saved AC's. Upon entry to an EX JSYS which is pseudo-interruptable, the AC's are BLT'ed into this save region. The EX JSYS then references its own AC's in the normal fashion, and the AC's saved from the calling program via UMOVEX and XCT instructions with forced user space effective addresses $<2\emptyset_8$. Pointers passed from the calling program which originally pointed into the AC's are evaluated with UMOVE or special XCT instructions, and automatically reference these saved AC's. This feature operates in the same-fashion whether the calling program was USER mode or EXEC mode, i.e. the CALL FM MON flag forces special references $\geq 20_8$ into the EXEC mode space, but special references $<20_8$ go into the saved AC stack independent of this flag.

A side effect of this feature is that forced references $<20_8$ in USER mode reference the user's shadow core. (The first 20_8 locations of the user's page zero).

10.6 Pager Traps

A paging trap will occur whenever one of the following events happens:

1. The trap bits in the process page table force a trap.
2. The addressed page (or indirecting page table) is not in core.
3. An illegal condition is detected.
4. The Core Status Table entry for an addressed page contains an age with the three highest bits = 0.

When one of the above conditions happens, the pager first stores the cause and location of the trap into the P.S.B. at location 571_8. The format of this word is shown in Figure 10-9. Then, if the APR operation in progress was a write, it stores the data into location 572_8 of the P.S.B. Finally, it forces the APR to execute absolute location 70_8 (170_8 if second APR). Location 70_8 should contain a JSYS instruction to a trap routine.

The arrangement of the Trap Cause field was chosen so that decoding of the cause could be easily accomplished by the JFFO instruction.

To restart a process which was terminated by a pager trap, the following information is of value:

1. The program counter (PC) saved by the JSYS at location 70_8 is correct.
2. If the read or execute bits in 571_8 of the P.S.B. are set, restart is completed by performing a JRSTF @ through the PC word saved by the JSYS at location 70_8 (170_8).
3. If the read bit is not set but the write bit is set in 571_8 of the P.S.B., the data in 572_8 of the P.S.B. must be written into the address in 571_8 of the P.S.B. before returning control to the process via JRSTF.

A sample program to restart a process which was terminated by a paging trap is given below:

```
        CONO PGR, Ø             ;LOAD PAGER WITH NEW BASE REGISTERS,
                                 ETC. (see section 10.7 for details)

        MOVE 1, 777571          ;GET TRAP STATUS WORD

        TLNE 1, 12              ;SKIP IF NEITHER READ NOR EXECUTE
                                ;BITS SET

        JRST BEGIN

        MOVE 777572             ;GET DATA WORD

        TLNE 1, 1               ;SKIP IF USER MODE

        JRST MONWR

        UMOVEM (1)              ;COMPLETE USER MODE WRITE

BEGIN:  HRLZI 17, 777520        ;RESTORE AC's FOR THE PROCESS

        BLT 17, 17

        JRSTF @777573           ;RESUME PROCESS (JSYS IN LOCATION 70₈

                                ;SAVES THE FLAGS AND PC IN 777573).

MONWR:  MOVEM (1)               ;COMPLETE MONITOR MODE WRITE

        JRST BEGIN
```

Figure 10-1 shows that the PI cycle, Key Cycle, and Indirect Address Fetch levels are provided to the pager. The reason for the PI cycle and KEY cycle bits is to distinguish traps of the running program from PI and KEY cycle traps (KEY cycles occur when the console EXAMINE, DEPOSIT, or XCT switches are pushed). The Trap Status Word in Figure 10-9 contains sufficient information to simulate a KEY cycle operation and recover from the trap provided timing is not critical (e.g. BLKO or BLKI to magtape or dectape might get data late indications). However, a pager trap during a PI cycle should never occur and is a disaster so recovery is impossible. The running program can be continued after such a trap by JRSTF @777573 as in the sample restart program. The indirect address sequence bit is used to distinguish data reads of non-existent memory from indirect addressing reads of non-existent memory.

In the former case, the software may wish to create a "new memory page" and proceed, but in the latter case, a programming error has been made.

Another interesting feature of the pager is the monitor after-loading trap. All other directed traps take place at the beginning of the self-loading sequence, before any associative register has been loaded with mapping information for the new page, but the after-loading trap takes place at completion of the self-loading sequence. This enables the Monitor to perform statistics-taking operations for specified pages each time one is loaded into an associative register.

TRAP STATUS WORD

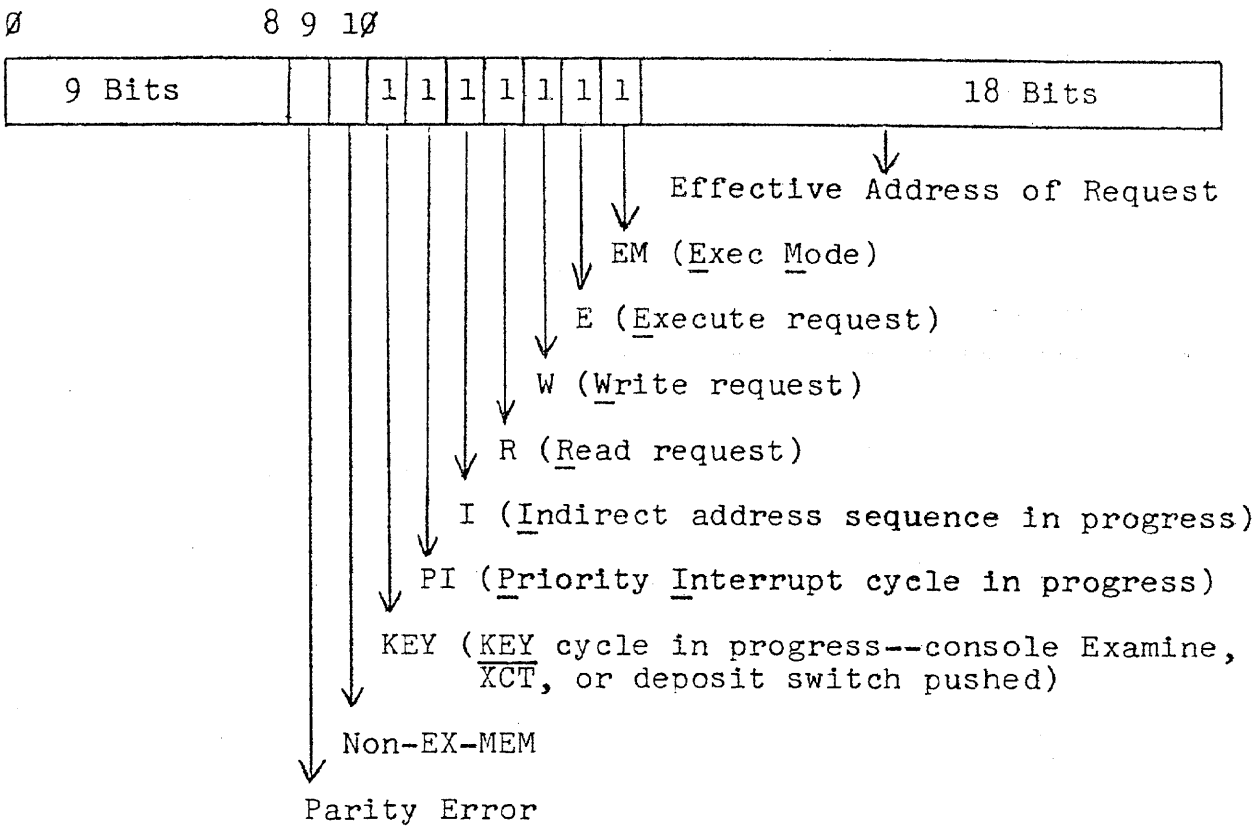


Figure 1Ø-9

Bits Ø-8, trap cause are decoded as follows: Bits Ø and 1 define one of four groups each defined below:

Group Ø: TSR Ø, 1 = ØØ

Bit	Meaning if ON	
2	AGE = ØØX	as read from C.S.T.
3	AGE = Ø2X	
4	AGE = Ø4X	
5	AGE = Ø6X	
6	Monitor After-Loading A.R. trap	

Group 1: TSR Ø, 1 = Ø1

Bit	Meaning if on
3	Shared not in core
4	page table not in core (p.t.2)
5	2nd indirect, private not in core (p.t.3)
6	Indirect shared not in core (p.t.2 or p.t. 3)
7	Indirect page table not in core (p.t.3)
8	Excessive Indirect pointers (>2)

Group 2: TSR Ø, 1 = 1Ø

Bit	Meaning if on
2	Private Not in core
3	Write copy trap (bit 9 in P.T.)
4	User trap (bit 8 in P.T.)
5	Access trap (P.T. bit 12 = Ø or bits 1Ø-11=3)
6	Illegal Read or Execute
7	Illegal Write
8	Address Limit Register Violation or P.T. bits Ø,1=3 (illegal format)

Group 3: TSR Ø, 1 = 11

Bit	Meaning if on
2	Private Not in core
3	Write copy trap (bit 9 in P.T.)
4	User trap (bit 8 in P.T.)
5	Access trap (P.T. bit 12 = Ø or bits 1Ø-11=3)
6	Illegal Read or Execute
7	Illegal Write
8	Address Limit Register Violation or P.T. bits Ø,1=3 (illegal format)

(in 2nd or 3rd page table)

10.7 Controlling the Pager via I/O Buss CONO's

The IOB reset pulse generated by the APR causes the pager to completely clear itself. In the cleared state no mapping is performed by the pager and all memory requests are passed unchanged to the memory buss. The pager is assigned device mnemonic PGR (device number 24) and interprets the three low bits of CONO PGR, X as described below. Other bits of the CONO are ignored.

CONO PGR, Ø — Clears all associative registers and reloads the Monitor and User mode base registers and Address Limit Register from location 71 and the Core Status age and process use registers from location 72. (see Figure 10-10)

CONO PGR, 1 — Clears all associative registers mapping EXEC mode pages.

CONO PGR, 2 — Clears the associative register mapping the page addressed by the next write (or read-modify-write) memory reference. The Pager operates in the normal manner both before and after this write reference but does not complete the write operation.

Note that because a priority interrupt may occur between the execution of this CONO and the following write instruction, it is normally required to do the following:

```
CONO PI, 4ØØ       ;TURN OFF PI SYSTEM
CONO PGR, 2        ;CLEAR PAGE OF NEXT WRITE
MOVEM PAGE*1ØØØ    ;CLEAR PAGE
CONO PI, 2ØØ       ;TURN PI SYSTEM BACK ON
```

CONO PGR, 3 — Clears all associative registers mapping USER mode pages

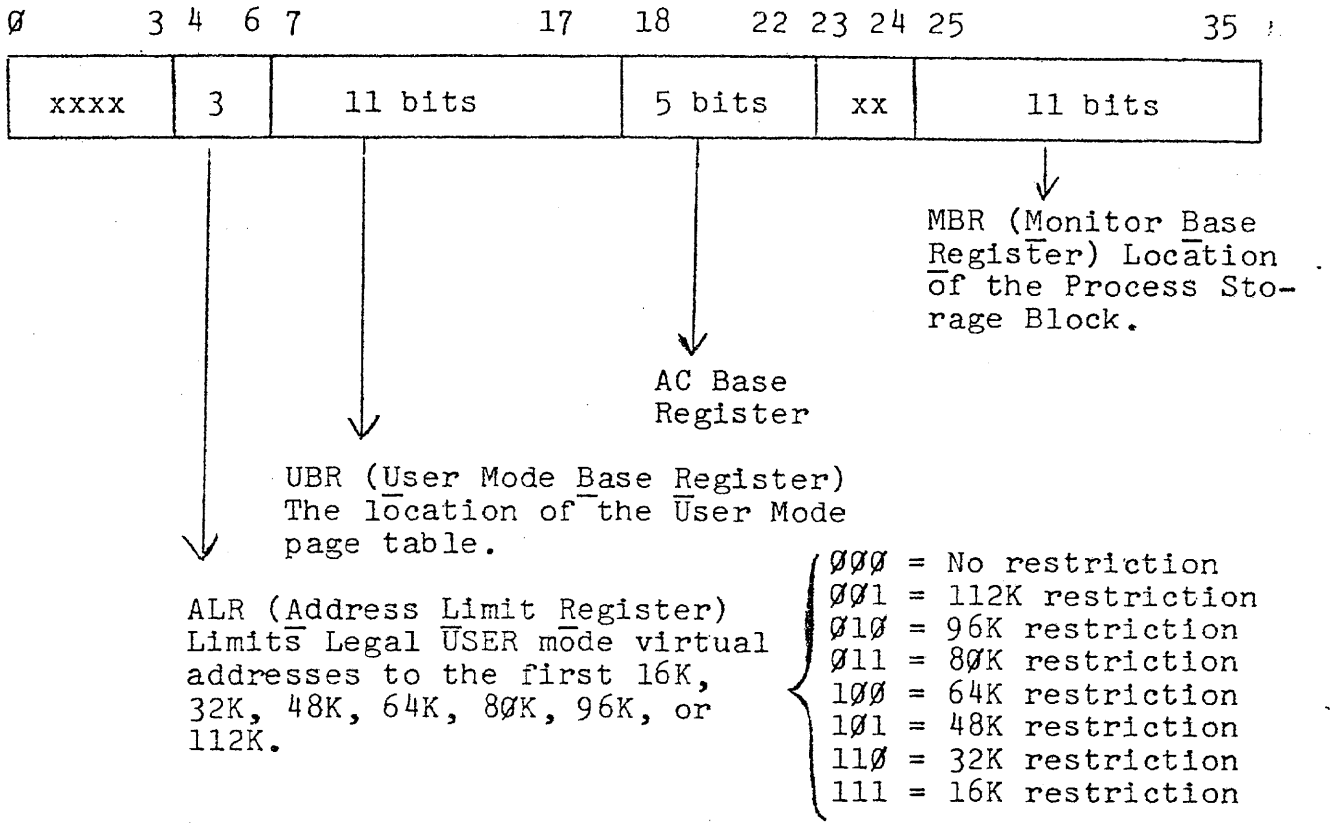
CONO PGR, 4 — Turns off all mapping, leaving base registers and associative registers unchanged

CONO PGR, 5 — is equivalent to CONO PGR, 4

CONO PGR, 6	Turns off mapping for resident monitor (virtual addresses $20\text{-}77777_8$) and turns on USER mode mapping and mapping of EXEC space $100000_8 - 777777_8$
CONO PGR, 7	Turns on mapping for all address $20_8 - 777777_8$ for both EXEC mode and USER mode references

CONO PGR, Ø causes the Pager to reload its main registers as follows:

1. Read absolute location 71_8. (171_8 if second APR), interpreted as below.



2. Read location 72_8 (172_8 if second APR)

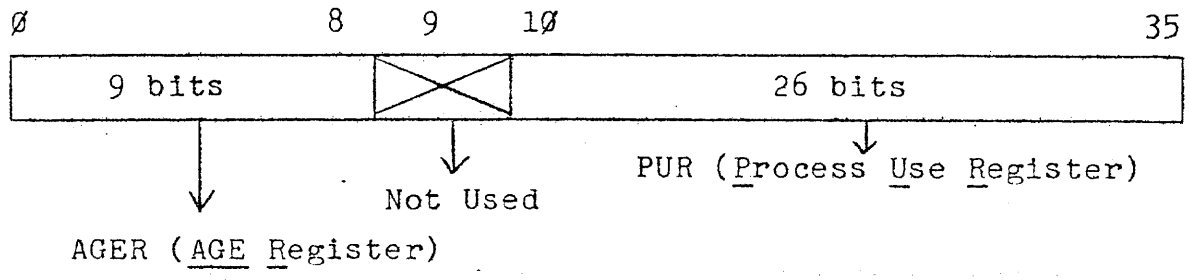


Figure 1Ø-Ø

Initializing the Pager For Running a New Process

Glossary

A.G.E.R.	AGE Register
A.L.R.	Address Limit Register
A.P.R.	Arithmetic PRocessor (KA1Ø)
A.R.	Associative Register
C.S.T.	Core Status Table
M.B.R.	Monitor Base Register
PGR	PaGeR device mnemonic
P.S.B.	Process Storage Block
P.T.	Page Table
P.T.N.	Page Table Number
P.U.R.	Process Use Register
R.C.A.	Real Core Address
S.P.N.	Shared Page Number
S.P.T.	Special Pages Table
U.B.R.	User Mode Base Register