Copy Number 10

TECHNICAL DETAILS OF THE BBN PAGER MODEL 701

Theodore R. Strollo
Jerry D. Burchfiel
Raymond S. Tomlinson

22 July 1970

Bolt Beranek and Newman Inc.

50 Moulton Street

Cambridge, Mass. 02138

10. ARITHMETIC PROCESSOR PAGING

10.1 Introduction

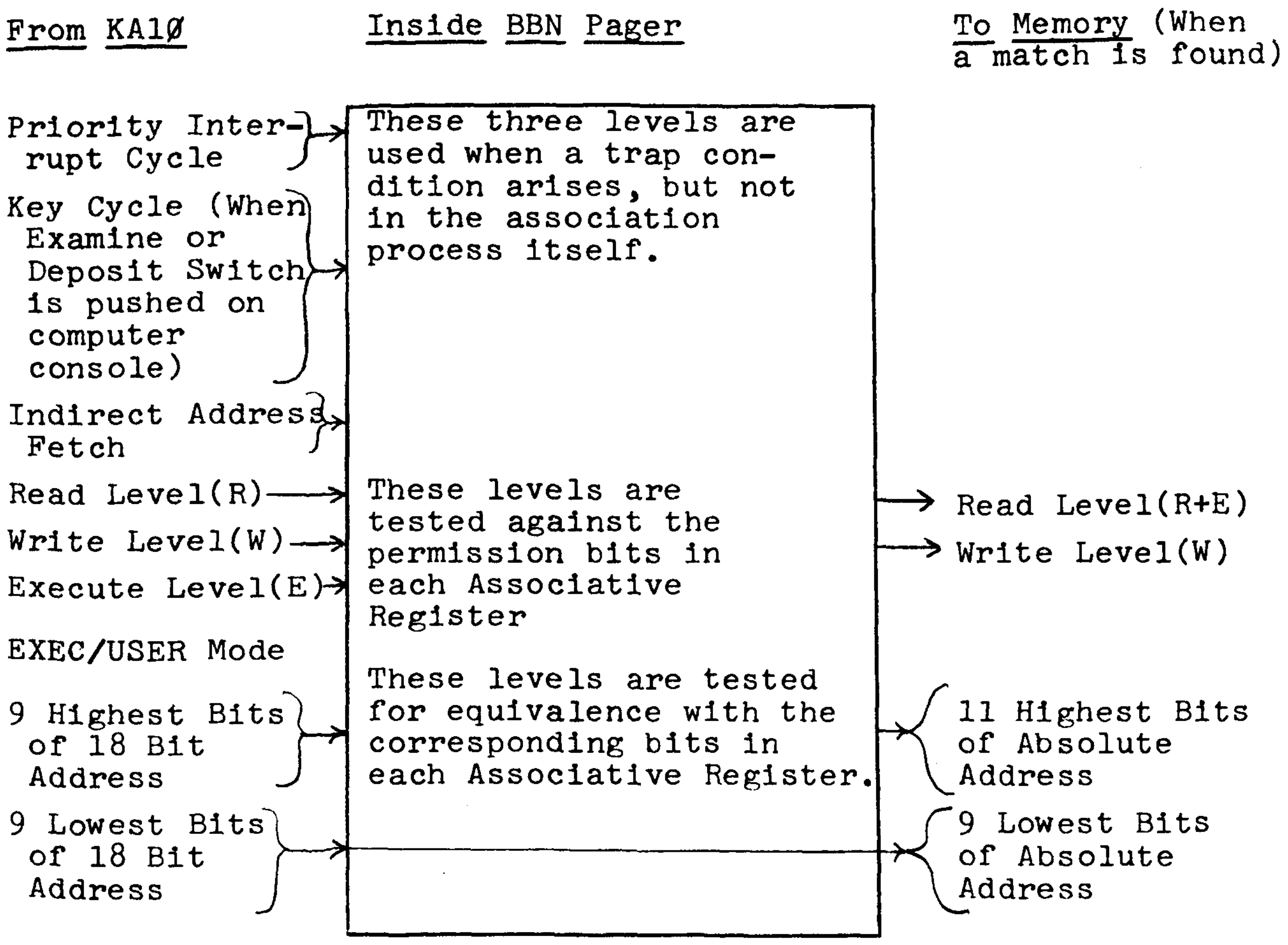
BBN has implemented a device called the BBN Pager which is connected between the KA1Ø (PDP-1Ø arithmetic processor) and the KA1Ø's memory port. In conjunction with a set of hardware modifications to the KA1Ø, the BBN Pager changes the core memory mapping mechanism such that core memory is allocated and 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


From KA10
Inside BBN Pager
To Memory (When a match is found)
Priority Interrupt Cycle
Key Cycle (When Examine or Deposit Switch is pushed on computer console)
Indirect Address Fetch
These three levels are used when a trap condition arises, but not in the association process itself.
Read Level(R)
Write Level(W)
Execute Level(E)
These levels are tested against the permission bits in each Associative Register
Read Level(R+E)
Write Level(W)
EXEC/USER Mode
9 Highest Bits of 18 Bit Address
9 Lowest Bits of 18 Bit Address
These levels are tested for equivalence with the corresponding bits in each Associative Register.
11 Highest Bits of Absolute Address
9 Lowest Bits of Absolute Address

Bits in Associative Registers

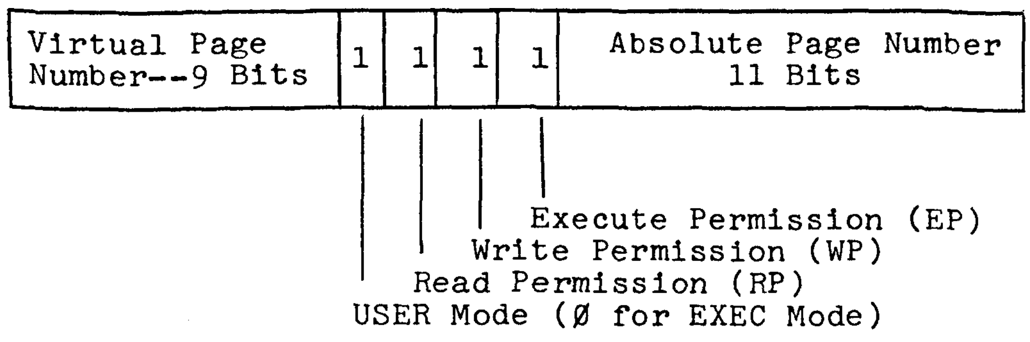

Virtual Page Number--9 Bits
1
1
1
1
Absolute Page Number 11 Bits
Execute Permission (EP)
Write Permission (WP)
Read Permission (RP)
USER Mode (Ø for EXEC Mode)

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.

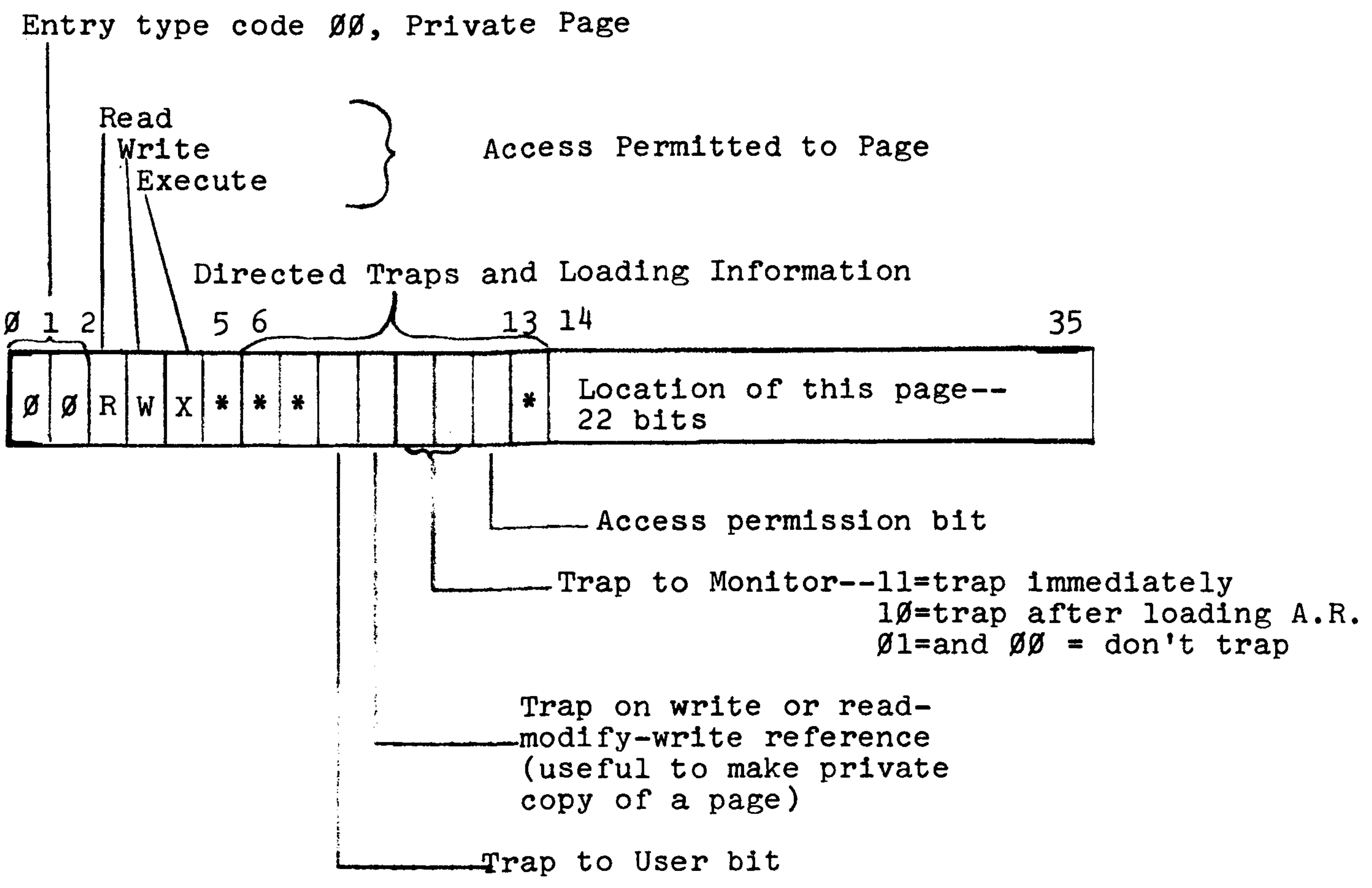
The first stage of the self-loading sequence involves reading some information from a table in core memory called the page table (hereafter abbreviated as P.T.). This table is 512 words long and is itself a page which may be anywhere in core memory. The origin of the P.T. is specified by the contents 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.



Entry type code ØØ, Private Page
Read
Write
Execute
Access Permitted to Page
Directed Traps and Loading Information
Ø 1 2
5 6
13 14
35
Ø Ø R W X * * *
*
Location of this page--
22 bits
Access permission bit
Trap to Monitor--11=trap immediately
1Ø=trap after loading A.R.
Ø1=and ØØ = don't trap
Trap on write or read-modify-write reference (useful to make private copy of a page)
Trap to User bit

*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.

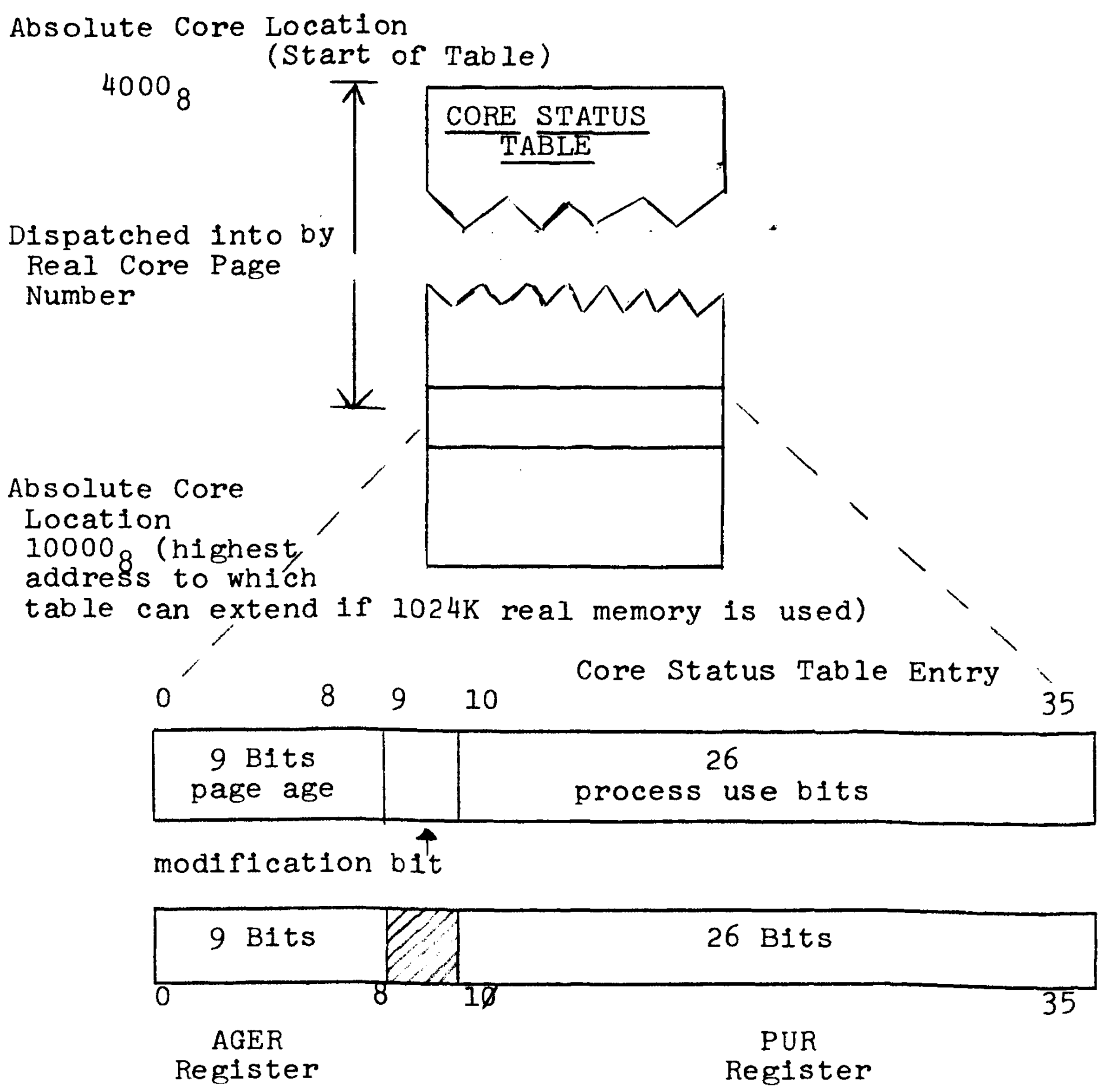
The use of the core status table (C.S.T.) causes one additional read-modify-write cycle of overhead (while referencing the C.S.T.) in the self-loading sequence.

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.

$$\left(\begin{array}{l}\text{A.R. write permit} \leftarrow \text{PT write permit} \wedge (\text{WRRQ} \vee \text{modification bit}) \\ \text{new modification bit} \leftarrow \text{old modification} \vee (\text{WRRQ} \wedge \text{PT write permit})\end{array}\right)$$

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.

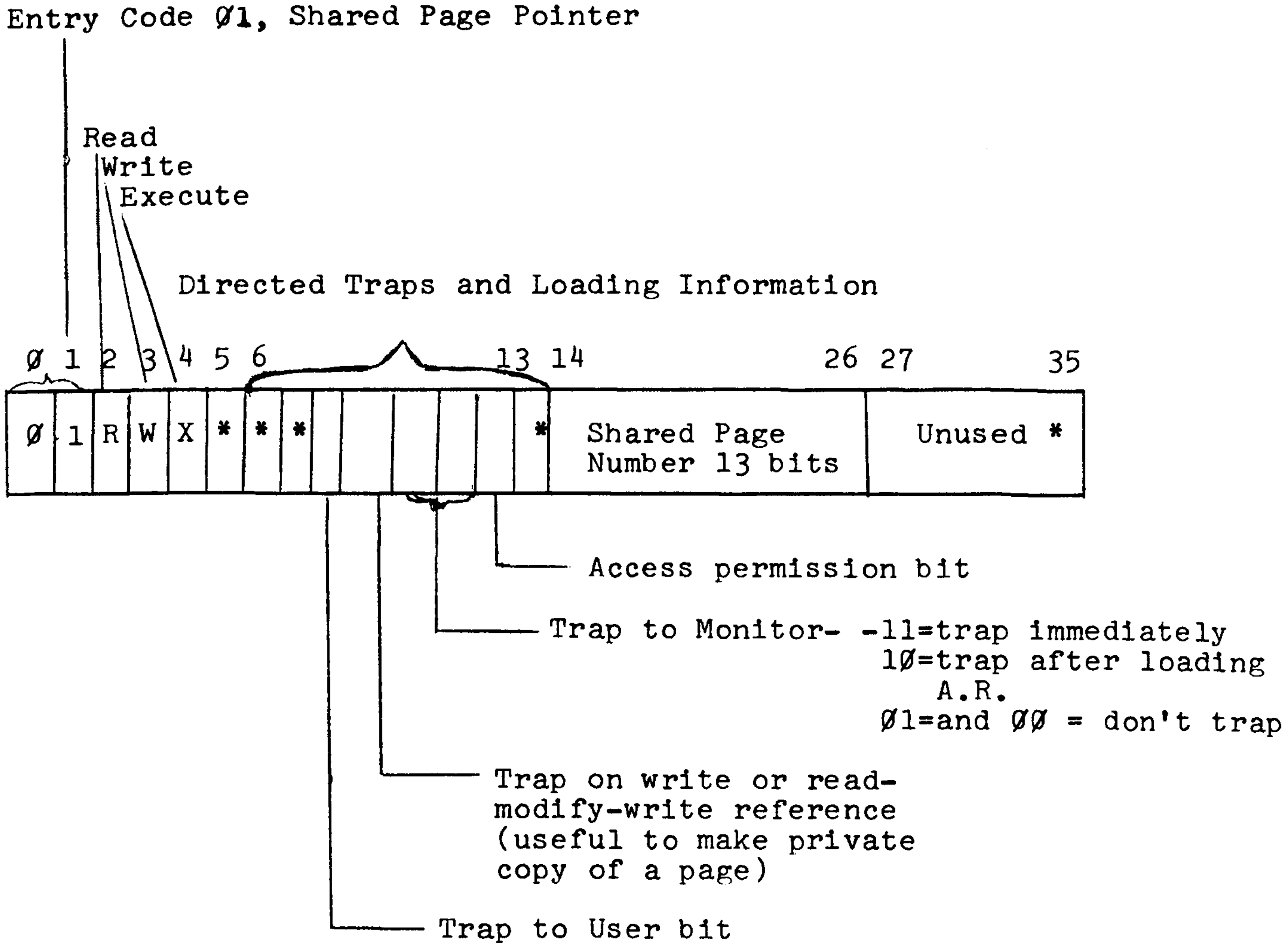

Entry Code Ø1, Shared Page Pointer
Read
Write
Execute
Directed Traps and Loading Information
Ø 1 2 3 4 5 6 13 14 26 27 35
Ø 1 R W X * * * *
Shared Page Number 13 bits
Unused *
Access permission bit
Trap to Monitor- -11=trap immediately 1Ø=trap after loading A.R. Ø1=and ØØ = don't trap
Trap on write or read-modify-write reference (useful to make private copy of a page)
Trap to User bit

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 2Ø,ØØØ$_8$. 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.

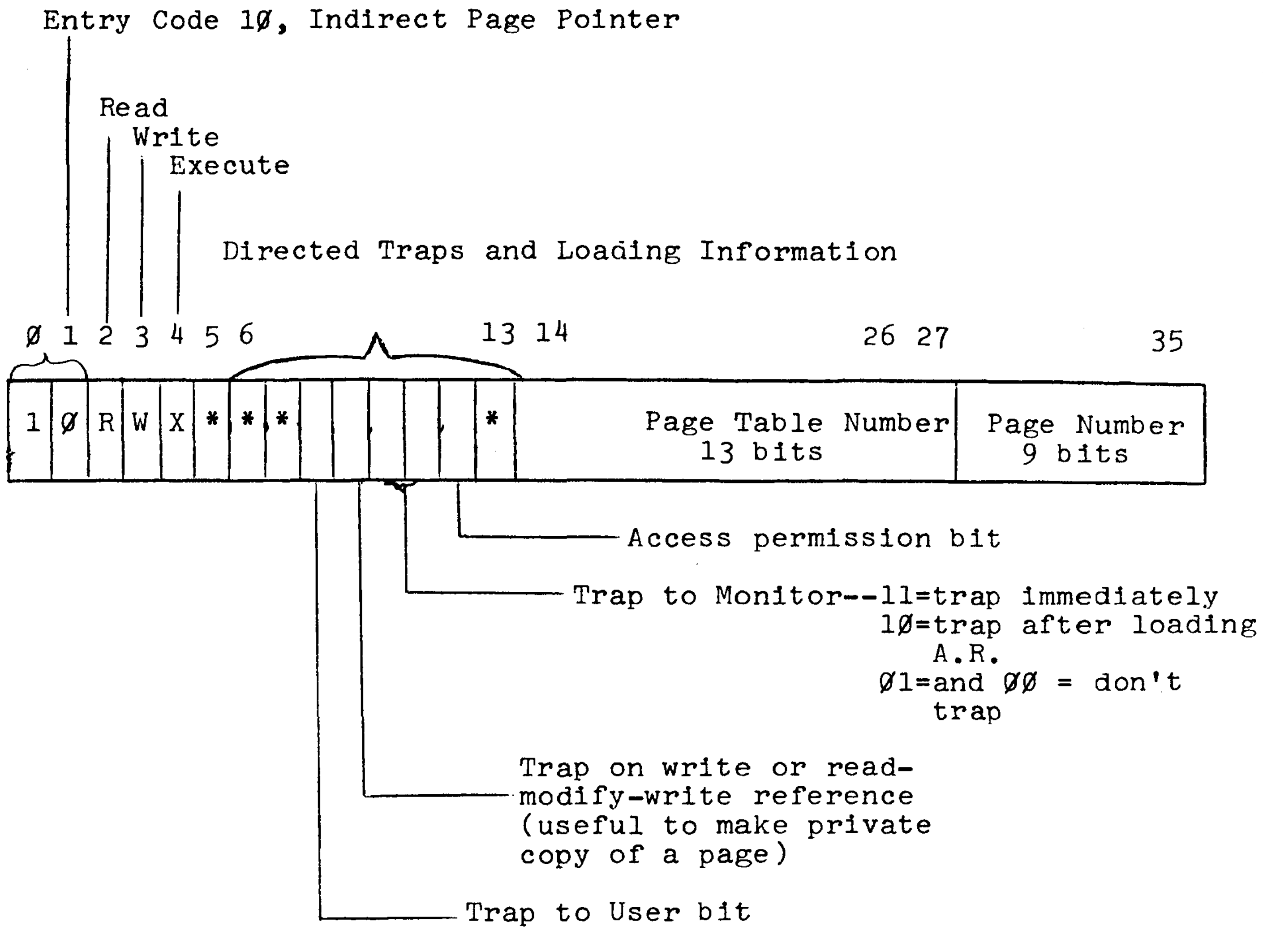

Entry Code 1Ø, Indirect Page Pointer
Read
Write
Execute
Directed Traps and Loading Information
Ø 1 2 3 4 5 6 13 14 26 27 35
1 Ø R W X * * * *
Page Table Number 13 bits
Page Number 9 bits
Access permission bit
Trap to Monitor--11=trap immediately 1Ø=trap after loading A.R. Ø1=and ØØ = don't trap
Trap on write or read-modify-write reference (useful to make private copy of a page)
Trap to User bit

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.

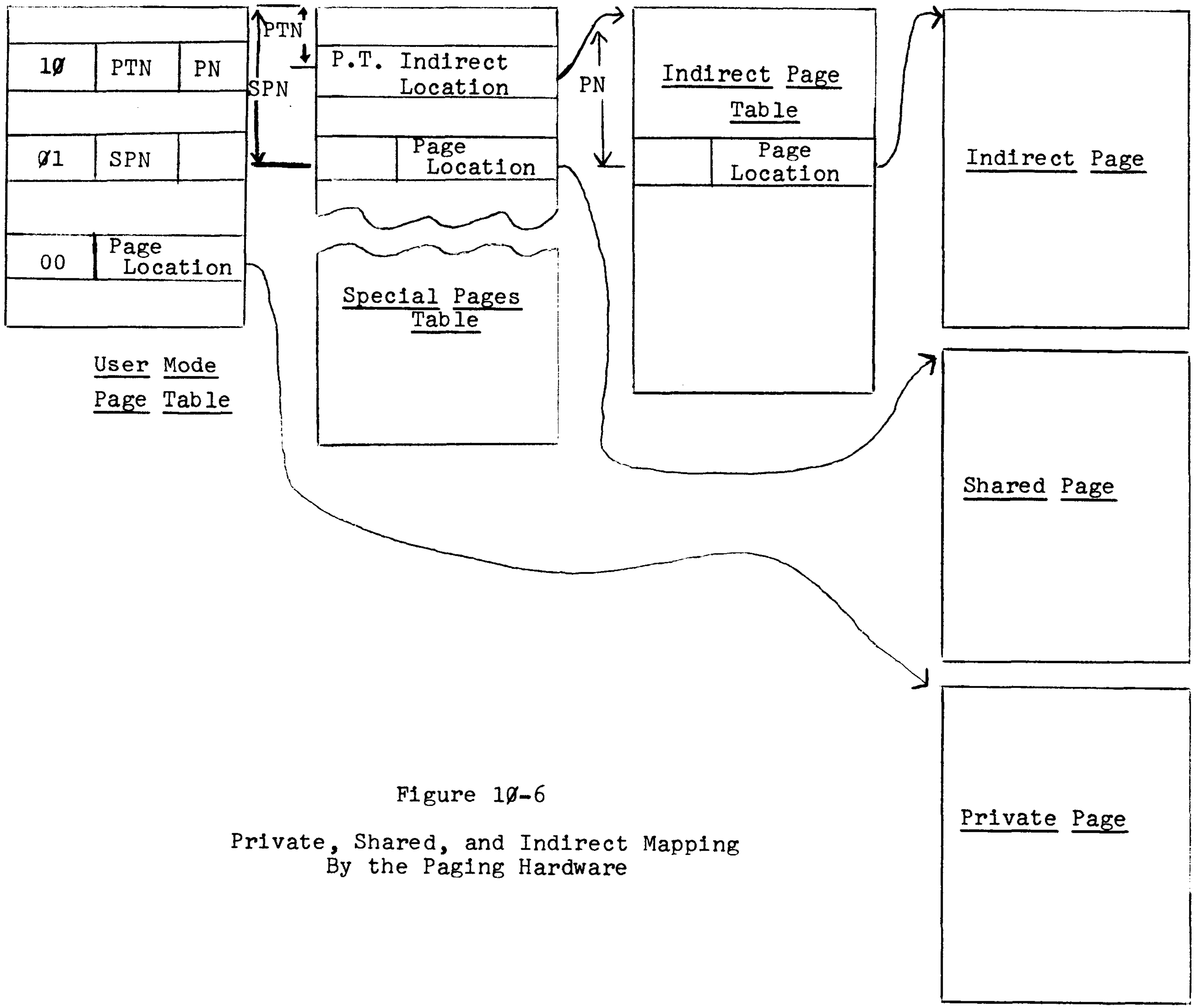


Figure 1Ø-6

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.

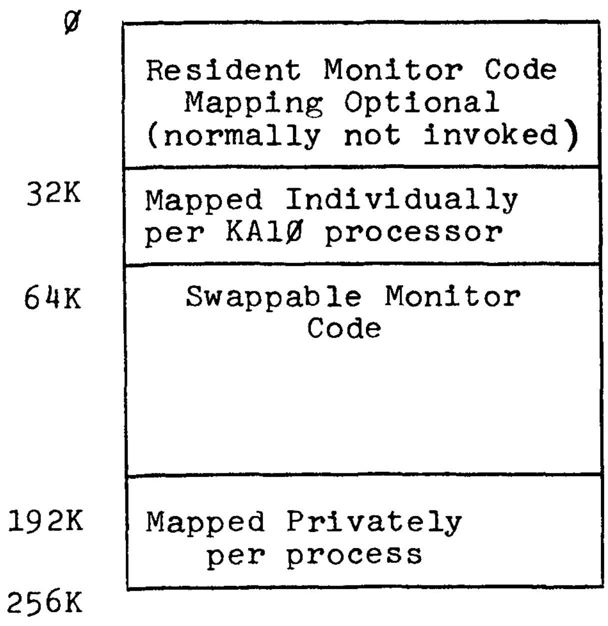

Ø
Resident Monitor Code Mapping Optional (normally not invoked)
32K
Mapped Individually per KA1Ø processor
64K
Swappable Monitor Code
192K
Mapped Privately per process
256K

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

100	M	A	I	X	Y
0 6	7 8	9 12	13	14 17	18 35

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.

AC Field

1Ø	4	2	1
Effective Address Computation, Last Address of BLT	Data Fetch, Byte Pointer Fetch, P~~op~~ Stack, Push Memory, 'From' of BLT	Address Computation From Byte Pointer	Data Store, Byte Fetch/Store, Push Stack, Pop Memory, 'To' of BLT
9	1Ø	11	12

Bit

Figure 1Ø-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 Ø-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 $<2Ø_8$. 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).

1Ø.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 = Ø.

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 1Ø-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 $7Ø_8$ ($17Ø_8$ if second APR). Location $7Ø_8$ 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 $7Ø_8$ 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 $7Ø_8$ ($17Ø_8$).
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:

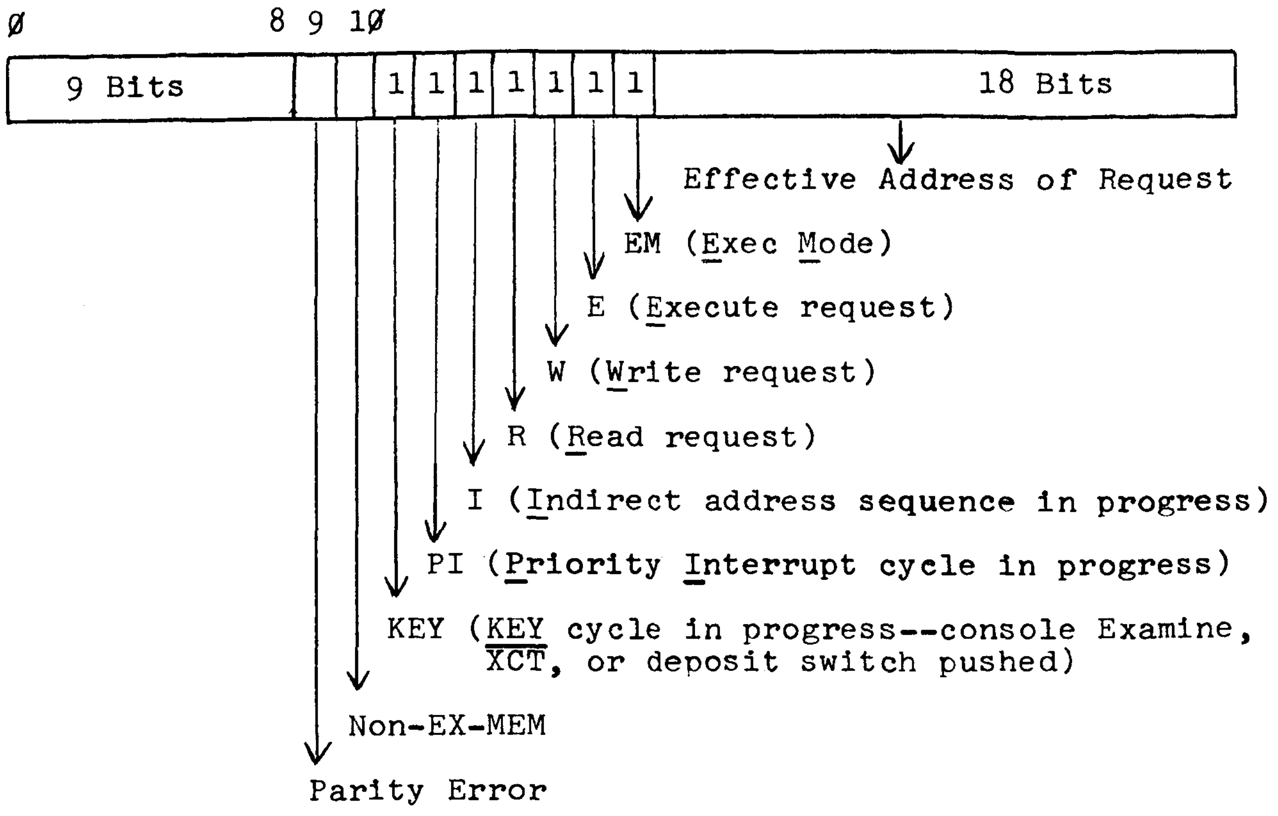
```
        CONO PGR, Ø           ;LOAD PAGER WITH NEW BASE REGISTERS,
                               ETC. (see section 10.7 for details)

        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 7Ø8

                              ;SAVES THE FLAGS AND PC IN 777573).

MONWR:  MOVEM (1)             ;COMPLETE MONITOR MODE WRITE

        JRST BEGIN
```

Figure 1Ø-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 1Ø-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



Ø
8 9 10
9 Bits
1 1 1 1 1 1 1
18 Bits
Effective Address of Request
EM (Exec Mode)
E (Execute request)
W (Write request)
R (Read request)
I (Indirect address sequence in progress)
PI (Priority Interrupt cycle in progress)
KEY (KEY cycle in progress--console Examine, XCT, or deposit switch pushed)
Non-EX-MEM
Parity Error

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

(in 2nd or 3rd page table)

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)

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:

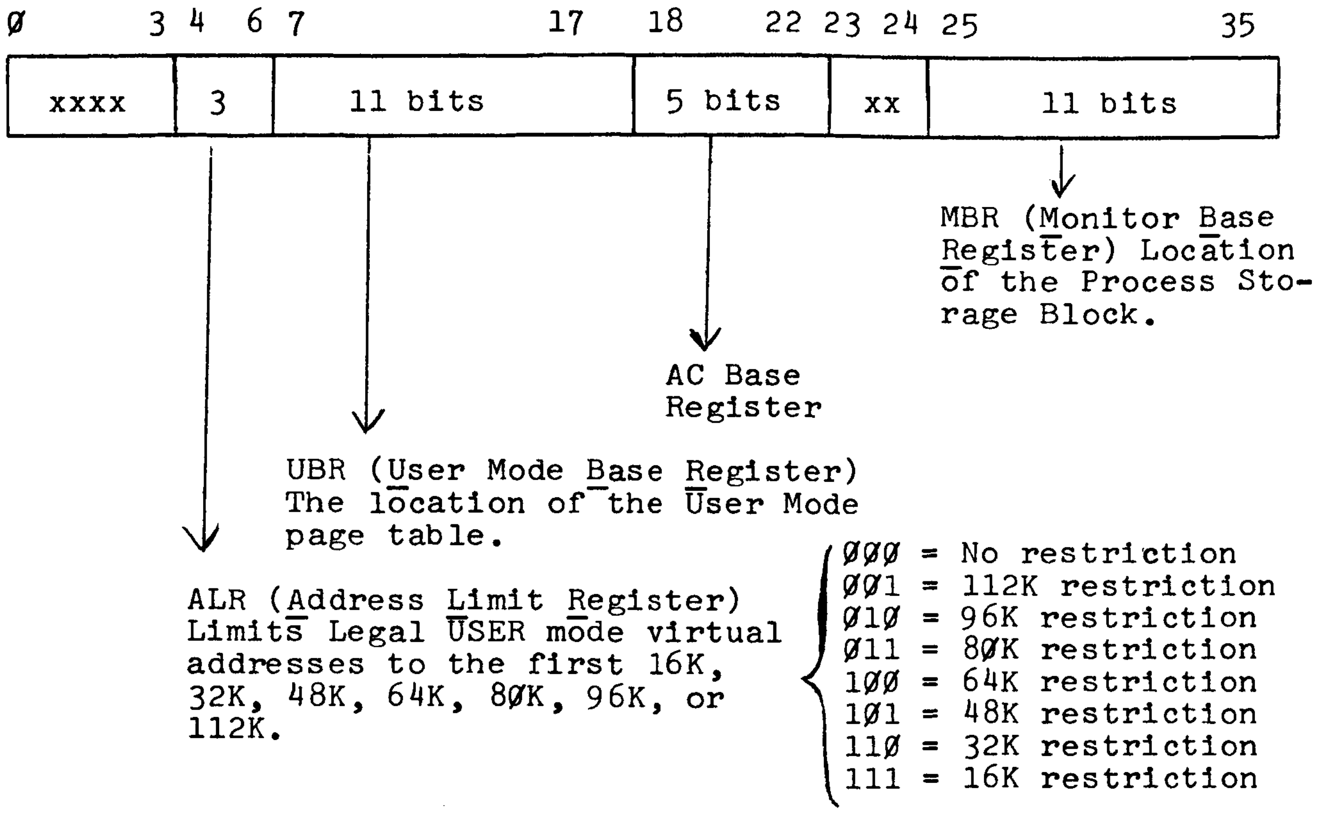
```
CONO PI, 4ØØ        ;TURN OFF PI SYSTEM
CONO PGR, 2         ;CLEAR PAGE OF NEXT WRITE
MOVEM PAGE*1ØØØ     ;CLEAR PAGE
CONO PI, 2ØØ        ;TURN PI SYSTEM BACK ON
```

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 $2Ø-77777_8$) and turns on USER mode mapping and mapping of EXEC space $100000_8 - 777777_8$
CONO PGR, 7	Turns on mapping for all address $2Ø_8 - 777777_8$ 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.



Ø 3 4 6 7 17 18 22 23 24 25 35
xxxx
3
11 bits
5 bits
xx
11 bits
MBR (Monitor Base Register) Location of the Process Storage Block.
AC Base Register
UBR (User Mode Base Register) The location of the User Mode page table.
ALR (Address Limit Register) Limits Legal USER mode virtual addresses to the first 16K, 32K, 48K, 64K, 80K, 96K, or 112K.
ØØØ = No restriction
ØØ1 = 112K restriction
Ø1Ø = 96K restriction
Ø11 = 8ØK restriction
1ØØ = 64K restriction
1Ø1 = 48K restriction
11Ø = 32K restriction
111 = 16K restriction

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

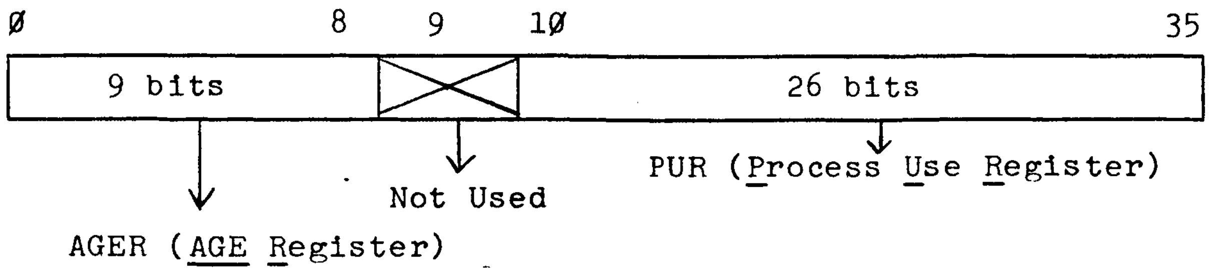
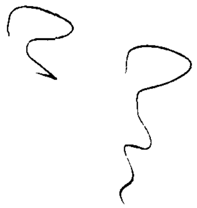


Figure 1Ø-Ø

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



this is a crock.

This is a crock.

Now is the

This is a test.
Now is the time for all
good men to come to the
aid of Uncle Sam.

[illegible]

BBN Pager Model 701 9July70

I. INTRODUCTION

The BBN Pager Model 701 is a device available to research PDP-10 users from Bolt Beranek and Newman Inc. for changing the memory mapping mechanism of the PDP-10. In conjunction with a set of modifications to the DEC PDP-10 arithmetic processor (the KA10), the BBN Pager allows paging of core memory, that is individual relocation of each 512 word page of the machine's address space.

II. ADVANTAGES OF PAGING

There are a number of advantages to the entire computer system when core memory is paged.

Efficient Use of Core Memory

One advantage of paging is that core memory is used more efficiently. Pieces (i.e. pages) of programs may be scattered anywhere in real core and the BBN pager relocates each page to provide a contiguous "virtual memory" for the user. Thus, the system no longer has to worry about collecting "holes" in core memory (as is required by the current PDP-10 dual relocation hardware) in order to fit programs in a contiguous area of real core.

Virtual Memory Increases the Effective Size of Real Core

Another advantage of paging is that a program can run which would physically take more core than the real core available in the system. This concept is called "virtual memory". Only pages that are needed at the moment must be in core. When new pages which are not in core are requested they can be swapped in from the drum and the program can then continue execution. Running partially loaded programs can substantially increase core memory efficiency.

Access Protection on a Page Basis

Another feature of the pager is the ability to provide separate access protection for each 512 word page in the address space. Independent Read, Write, and Execute protections are provided.

III. PARTICULAR ADVANTAGES OF THE BBN PAGER MODEL 701

The Core Status Table

The BBN pager maintains a core status table which keeps records of the activity of the pages in core memory. The pager notes when a page has been used, which processes have used that page, how long that page has been in core, and whether or not the page has been written into.

High Quality Engineering

The pager is constructed by machine wire wrapping and is made up primarily of standard DEC M-series modules, except for the BBN designed associative register card. Good engineering practice has been observed throughout the implementation of the pager. For ease in maintenance and software debugging, indicator lights are used on every control flip-flop and all critical levels in the pager. Also, the pager has a single stepping feature which enables one to step through or loop on each hardware "subroutine" of the pager cycle. In addition, the wire list for the pager has been carefully processed by computer programs to assure that no signal overloads exist, etc. The pager is also isolated from the memory buss during power up and down sequencing.

Software Support

The BBN pager has a complete set of diagnostic support and time sharing support software. The BBN TENEX time sharing system was explicitly designed for use with the BBN pager.

Compatibility with DEC Software

Modifications to the KA10 have been made in such a way that the hardware will continue to run in a so-called transparent mode if desired. In this transparent mode all of the DEC diagnostics will run and the DEC 10/50 system will also run.

IV. BRIEF TECHNICAL DESCRIPTION OF BBN PAGER

The paging mechanism involves the "association" of the 9 high order address bits from the KA10 with the contents of a set of associative registers. If a match is found, the register which contained the match also contains 11 "real" high order core address bits. (The reason for 11 bits is to permit up to 1 million words of real core to be referenced by the KA10). If no match is found, reference is made to a 512 word "page table" in real core memory. The word in this page table which is referenced is determined by a dispatch based on the original 9 high order address bits. In the simple case of a private page which is in core, the 11 high order address bits are found in this word and are automatically loaded into an associative register by the pager. There are 3 other cases:

a. The page is not in core or is non-existent in which case a page fault (trap) will occur.

b. The page is shared--in which case a reference is made to another table in core to find out where the page is really located.

c. The page indirectly points to an entry in another page table.

The pager maps both the user's address space and the monitor's address space separately. [The full technical details of the memory mapping mechanism are described in a forthcoming document.] One of the particular advantages of our mapping implementation is that the real core address of a page (even a shared page) is kept in one (and only one) place in the system!

ore details

V. PRICE, DELIVERY, AND OPTIONS

BBN will supply a fully assembled and checked out pager for $50K*, F.O.B., Cambridge, Mass. The pager will be checked out on a PDP-10 at BBN, Cambridge, Mass. and will undergo an 8 hour reliability test at BBN which the customer may witness. Payment terms are 30 days after acceptance at BBN, Cambridge. Delivery is 6 to 8 months ARO.*

Included with the pager are a complete set of pager prints, pager wire list, and the pager diagnostic (symbolic and binary) on a DECTAPE. Also included are a package of print updates and a set of instructions for the necessary wiring changes and module additions which must be made to the KA10 to accomodate the BBN pager. Contact Ted Strollo for details on assistance from BBN on the KA10 modifications.

Associative Registers

The BBN pager is normally configured with 16 associative registers. The pager will function with 1 to 54 associative

*Price and Delivery quotes are subject to change.

registers but if you ever intend to go beyond 16 registers, let us know with yourorder! (A larger power supply is needed). The number 16 is reasonable for most use of the TENEX system. Installations which will be running many user programs with large "working sets" (e.g. LISP) will operate more efficiently with more associative registers. These registers are available at approximately $250 per register from BBN. Such registers are added in the field by simply plugging them in.

VI. CONDITIONS OF PAGER SALE

The KA10 processor mods for the BBN pager have been engineered to be compatible with KA10's which have up to and including ECO #KA10-00060 installed in the machine. BBN cannot take responsibility for keeping the processor mods compatible with any future DEC modifications of the KA10. It is envisioned that there will be very few modifications to the KA10 in the future since DEC is busily working on new machines and most of the bugs and problems with the KA10 have been solved. (We have confirmed that no KA10 ECO's are in progress or envisioned by DEC personnel). If, of course, a major problem is found with the KA10 which is fixed by a DEC ECO, BBN will make every effort to find a way to make its modifications compatible with the ECO or vice versa. When we find this possible we will distribute these changes to our customers.

The BBN KA10 processor modifications assume the KA10 wiring runs are exactly the same as in BBN's two KA10's. This means that the basic KA10 wiring runs as well as all ECO's must not have been changed by either DEC or customer personnel. If there have been any deviations, our add/delete lists cannot be followed literally to achieve the correct results. In order to help those who may have deviated, we have included the mnemonics of signals being added and deleted with our processor modifications.

It has been our experience at BBN that DEC will continue to maintain a KA10 with the BBN processor mods installed.

The customer is responsible for obtaining the memory buss cable to connect the pager to the memory buss connection of the KA10. Two special additional cables to connect the KA10 to the pager are provided by BBN (length to be specified by customer with order and not to exceed 30 feet).

BBN will distribute documentation on pager ECO's (should any be issued) to our customers.

VII TENEX SOFTWARE DISTRIBUTION

The TENEX software will be available to research PDP-10 users under a special licensing arrangment (detailed in a forthcoming document). Those customers who buy the BBN Pager Model 701 are entitled to a copy of the TENEX software package at the price of reproduction. Those research PDP-10 users who are not interested in buying the BBN Pager Model 701 but would like a copy of the TENEX software may obtain the TENEX software package for a licensing fee of $15K plus the price of reproduction.

Because of variations in customer equipment configurations, this TENEX system software will in most cases, require some software modifications to work on the customer's PDP-10 installation. BBN will be able to provide assistance with these modifications; contact Ted Strollo for details.

Our customers will also be included on all public distributions of TENEX software modifications, improvements, and bug fixes which will be distributed for the price of reproduction.

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 any 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 1ØØ nanoseconds.

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.

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:

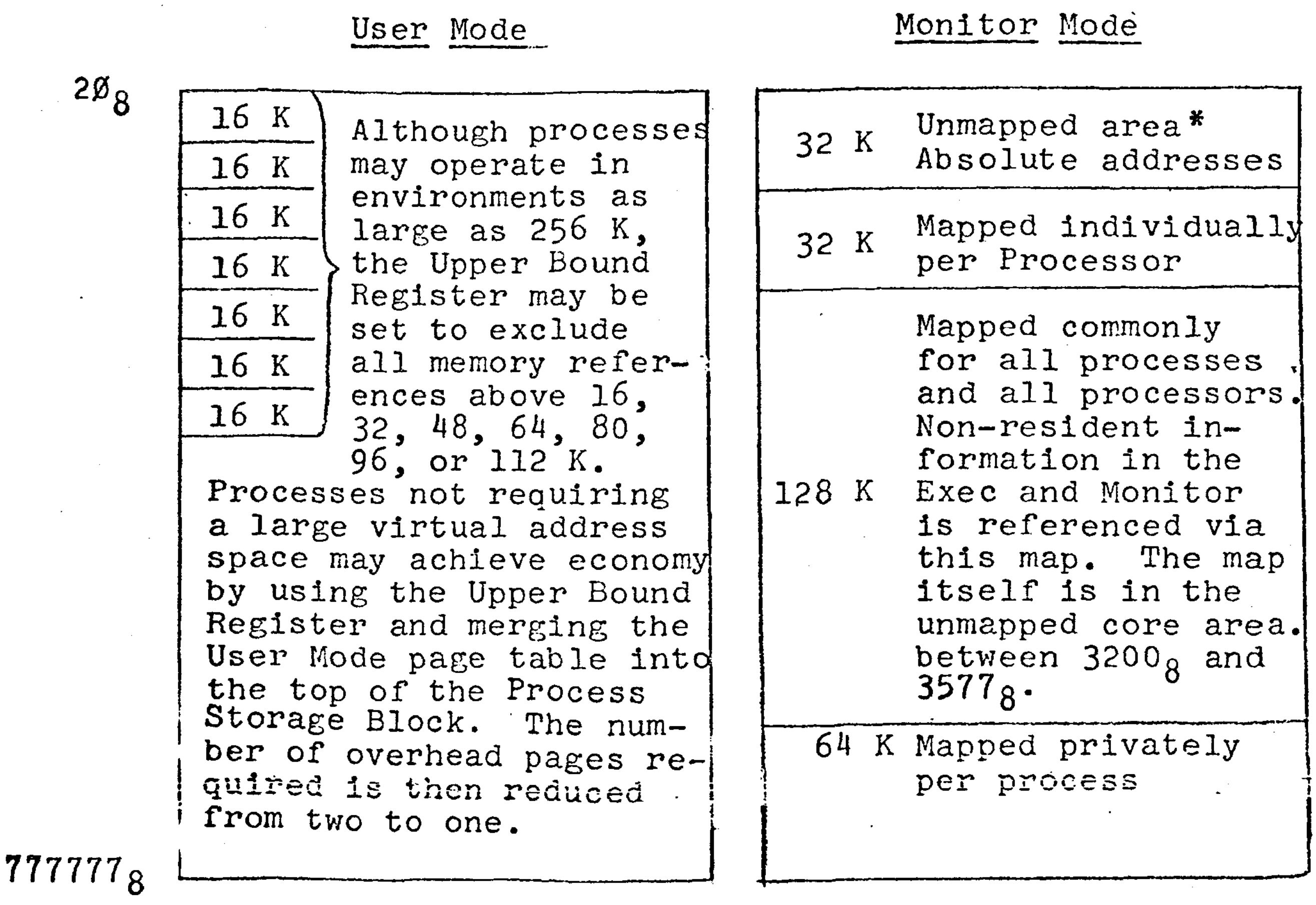
Optional Map (Not used unless Resident Monitor Mapping is turned on)	3000_8 to 3077_8	Absolute locations
First APR's Map	3100_8 to 3177_8	
Second APR's Map	3700_8 to 3777_8	
Common Map	3200_8 to 3577_8	
Private Map	600_8 to 777_8	of the Process Storage Block

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 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 number 24 and interprets the three low bits of CONO 24, X as described below. Other bits of the CONO are ignored.

CONO 24,0	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 registers from location 570 of the (newly mapped) Process Storage Block.
CONO 24,1	Clears all associative registers mapping monitor mode pages.
CONO 24,2	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.



User Mode

20_8

16 K
16 K
16 K
16 K
16 K
16 K
16 K

Although processes may operate in environments as large as 256 K, the Upper Bound Register may be set to exclude all memory references above 16, 32, 48, 64, 80, 96, or 112 K.
Processes not requiring a large virtual address space may achieve economy by using the Upper Bound Register and merging the User Mode page table into the top of the Process Storage Block. The number of overhead pages required is then reduced from two to one.

777777_8

Monitor Mode

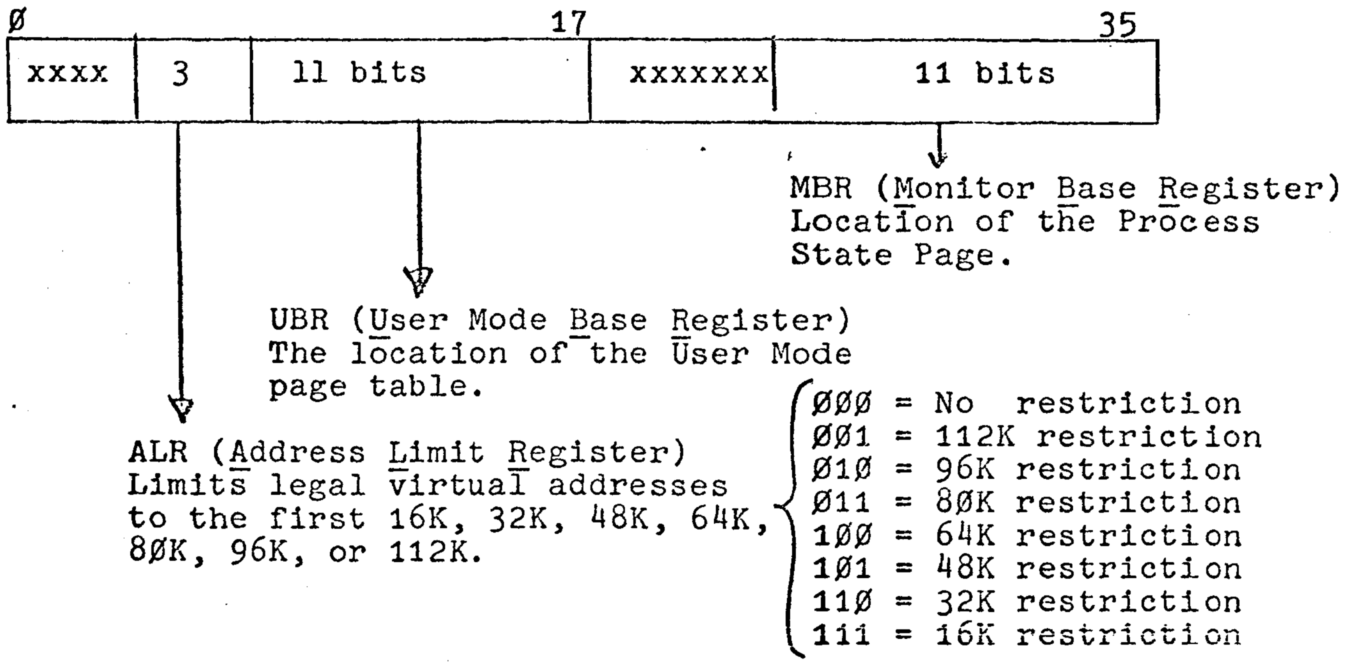
32 K	Unmapped area* Absolute addresses
32 K	Mapped individually per Processor
128 K	Mapped commonly for all processes and all processors. Non-resident information in the Exec and Monitor is referenced via this map. The map itself is in the unmapped core area between 3200_8 and 3577_8.
64 K	Mapped privately per process

Figure 1Ø-2
Mapping of Virtual Addresses

*Optionally mapped by locations 3ØØØ to 3Ø77

CONO 24,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.


Ø
17
35
xxxx
3
11 bits
xxxxxxx
11 bits
MBR (Monitor Base Register) Location of the Process State Page.
UBR (User Mode Base Register) The location of the User Mode page table.
ALR (Address Limit Register) Limits legal virtual addresses to the first 16K, 32K, 48K, 64K, 8ØK, 96K, or 112K.
ØØØ = No restriction
ØØ1 = 112K restriction
Ø1Ø = 96K restriction
Ø11 = 8ØK restriction
1ØØ = 64K restriction
1Ø1 = 48K restriction
11Ø = 32K restriction
111 = 16K restriction

2. Read location (MBR) $57Ø_8$.

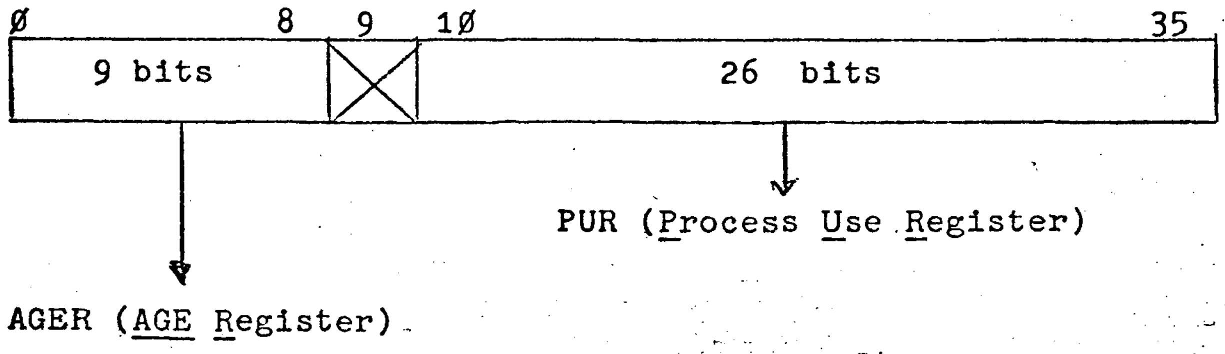
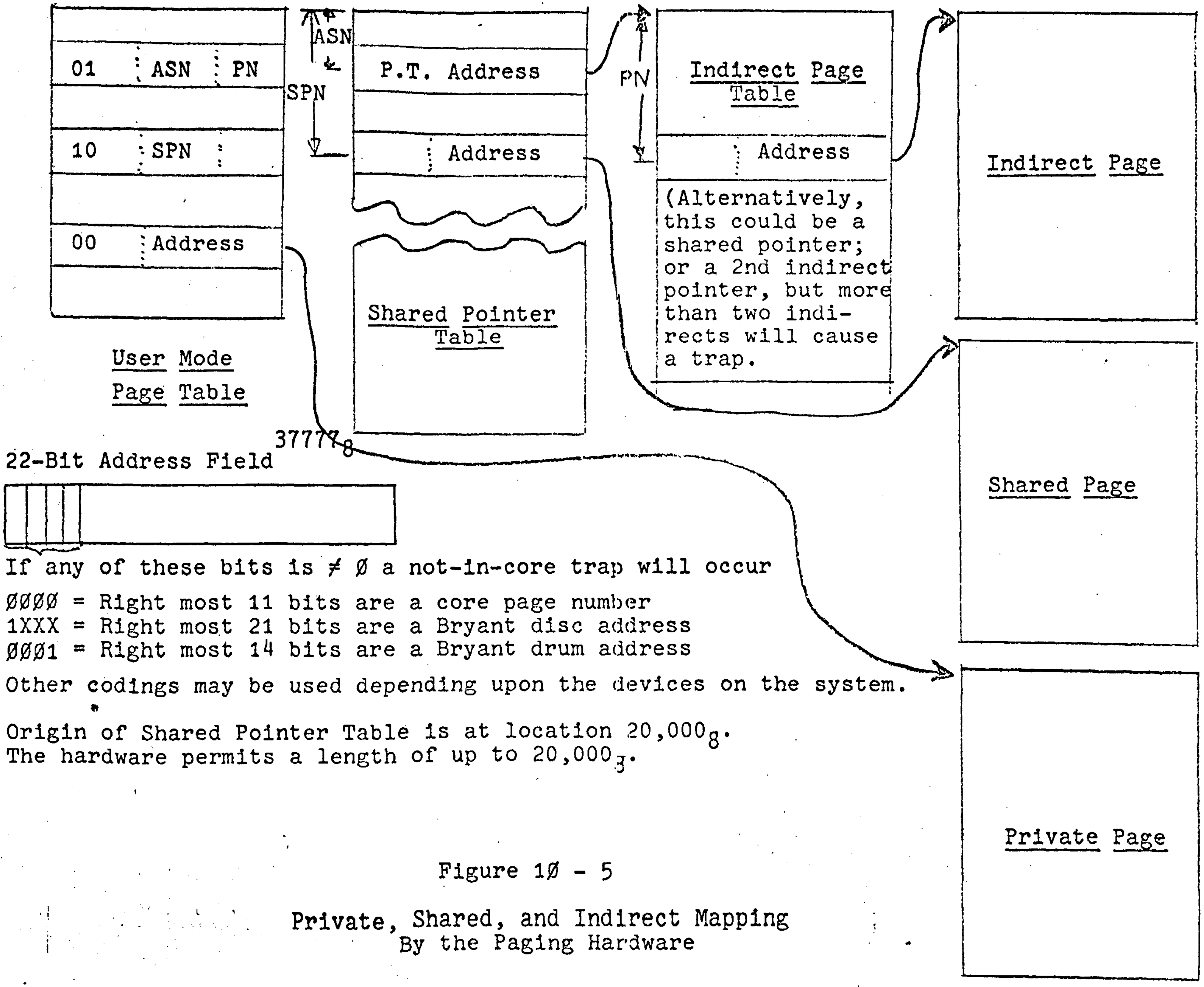


Figure 1Ø-3

Initializing the Pager For a New Process



If any of these bits is ≠ Ø a not-in-core trap will occur

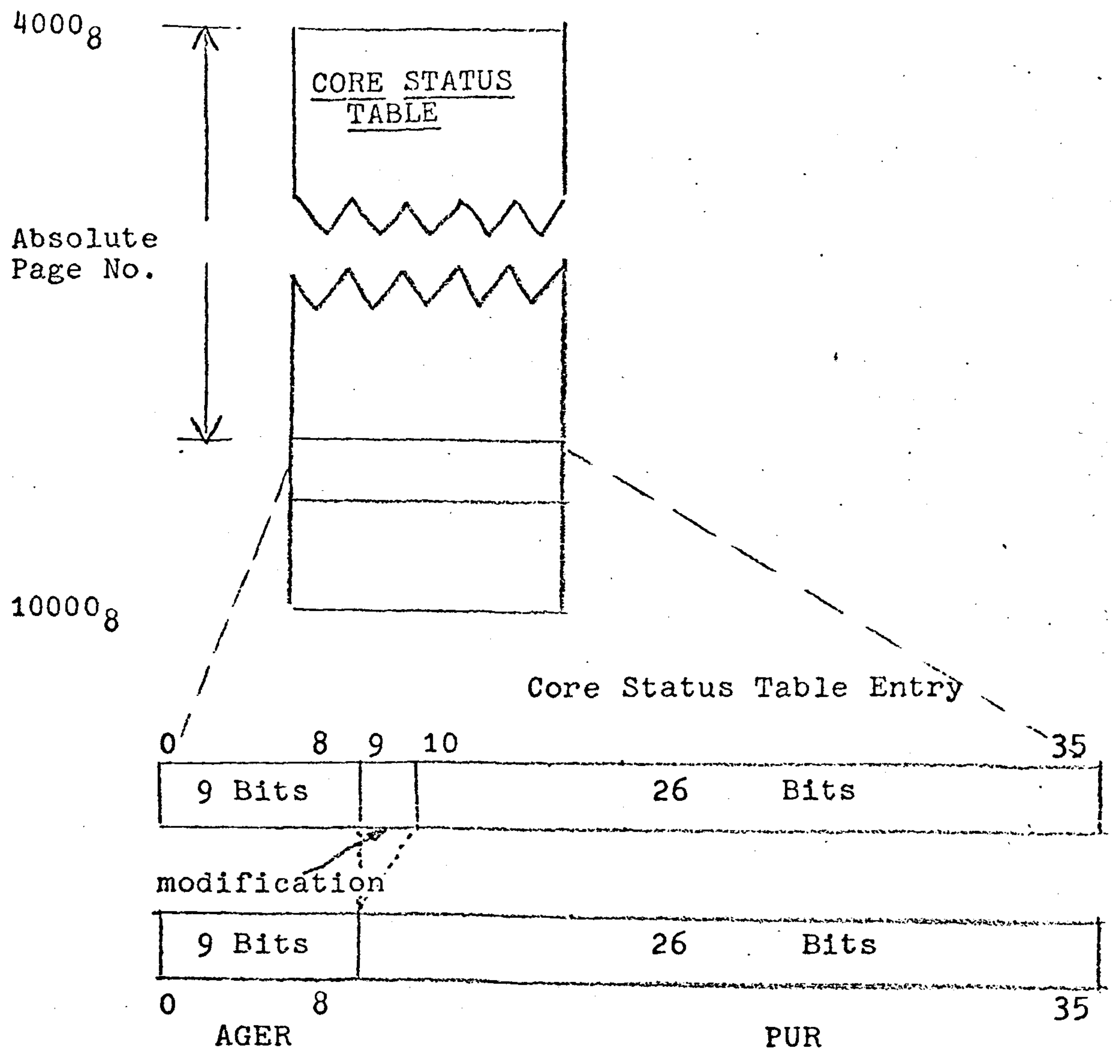
ØØØØ = Right most 11 bits are a core page number
1XXX = Right most 21 bits are a Bryant disc address
ØØØ1 = Right most 14 bits are a Bryant drum address

Other codings may be used depending upon the devices on the system.

Origin of Shared Pointer Table is at location $20,000_8$.
The hardware permits a length of up to $20,000_3$.

Figure 1Ø - 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 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 - 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.

1Ø.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 with the three highest bits = Ø (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 Storage block at location 571_8. The format of this word is shown in Figure 1Ø-7. Then, if the APR operation in progress was a write, it stores the data into location 572_8 of the state page. Finally, it forces the APR to execute absolute location $7Ø_8$ ($17Ø_8$ if second APR). Location $7Ø_8$ 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 $7Ø_8$ is correct.
2. If the read or execute bits in 571_8 of the state page are set, restart is completed by performing a JRSTF @ through the PC word saved by the JSYS at location $7Ø_8$ ($17Ø_8$).
3. If the read bit is not set but the write bit is set in 571_8, the data in 572_8 must be written into the address of 571_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,Ø          ;LOAD PAGER WITH NEW BASE REGISTERS
                           ;UPPER BOUND REGISTER, AND DUMP REGISTER
        MOVE 1,777571      ;GET TRAP STATUS WORD
        SETZM 777571       ;CLEAR TRAP STATUS FOR NEXT TIME
        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 7Ø8
                           ;SAVES THE FLAGS AND PC IN 777573).
MONWR:  MOVEM (1)          ;COMPLETE MONITOR MODE WRITE
        JRST BEGIN
```

In Figure 1Ø-7 the function of the PI cycle, Key Cycle, and Indirect Address Sequence bits may not be clear. 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 1Ø-7 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 miss latency). However, a pager trap during a PI cycle is a disaster and 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 new memory 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. Such statistics may be very useful in evaluating core management strategies and in evaluating the performance of the pager.

1Ø.6 Core Management Philosophy

Core management for processes with large virtual address spaces is a significant problem. To minimize difficulties we have designed into the Pager several features (Figure 1Ø-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.

(2) Identifying Processes using a page

By loading the process use register (PUR with a bit identifying 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 for each process 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) 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 $2Ø_8$. When the software is about to examine some entry in the Core Status Table, it may EXCH a word with the left-most three bits = Ø for the Core Status Table entry. This will prevent the other APR from inadvertently loading this page during the examination.