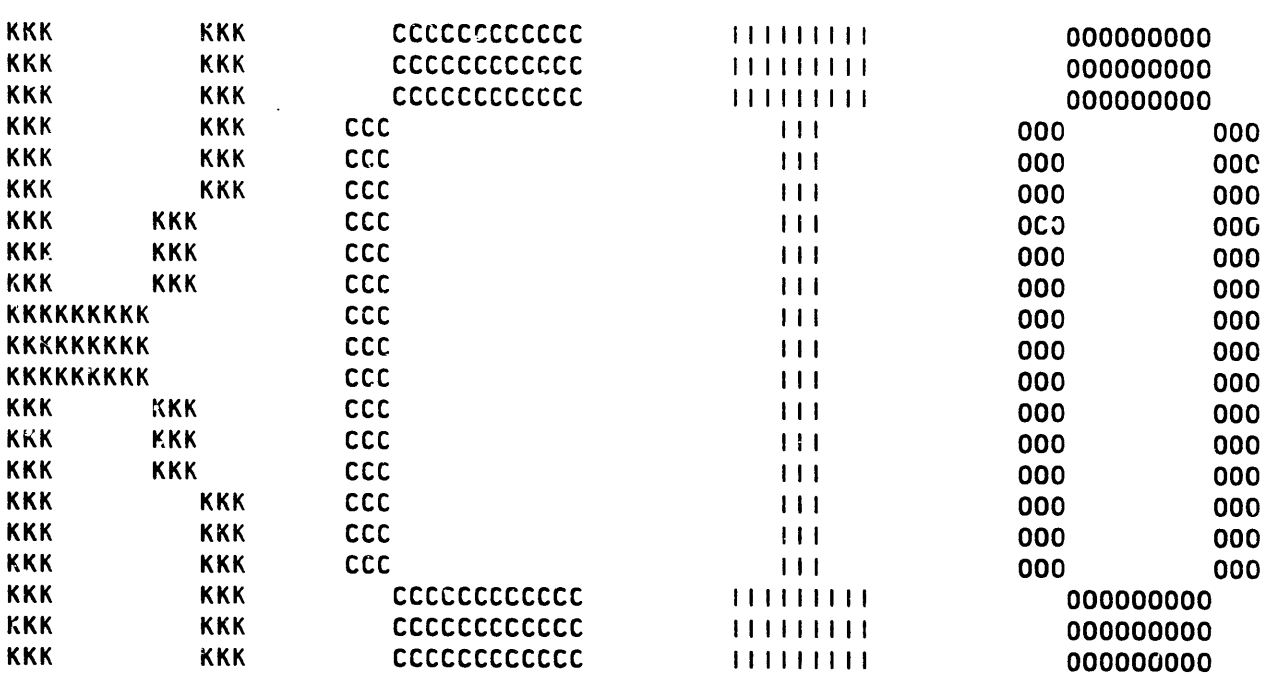
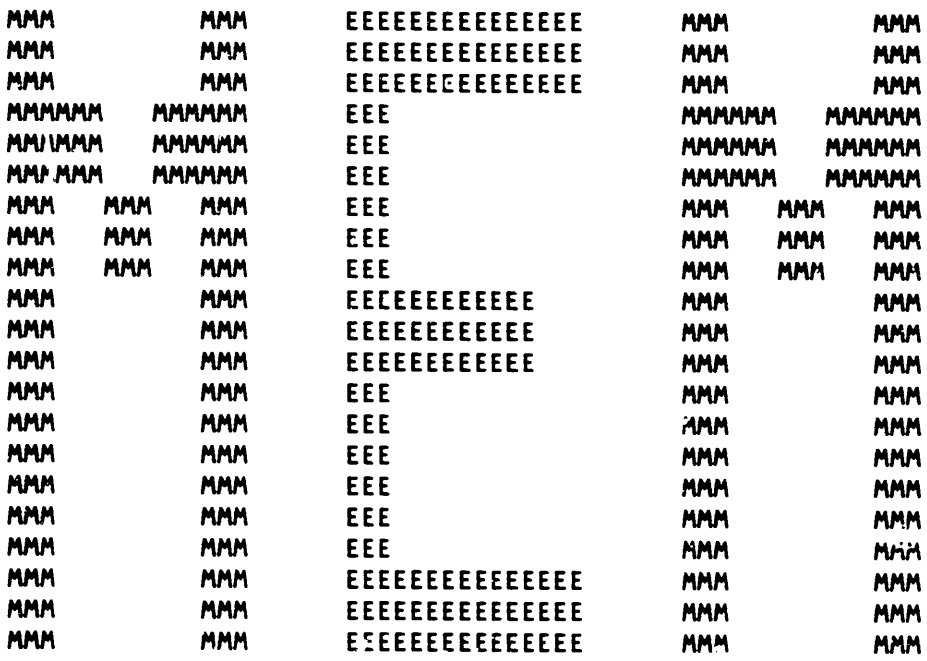




START Job ETHER Req #83 for TARL [10,5535] Date 26-Jul-83 0:52:11 Monitor: RL152A DEC10 Dev
File DSKA:KC10.MEM[5,100], created: 7-Mar-82 21:17:00, printed: 26-Jul-83 0:52:25
Job parameters: Request created:26-Jul-83 0:52:07 Page limit:3033 Forms:XEROX Account:l'd r
File parameters: Copy: 1 of 1 Spacing:SINGLE File format:ASCII Print mode:ASCII

+---------------+
!d i g i t a l! I N T E R O F F I C E M E M O R A N D U M
+---------------+

TO: List

DATE: 07 Mar 82

FROM: R. McLean
G. M. Uhler

DEPT: L.S.E.G

LOC: MRi-2/E85

EXT: 231-6113

DISTRIBUTED: 07 Mar 82
REVISION: 6

FILE: KC10.RNO

SUBJ: KC10 Exec Mode

Changes made to revision 5 of KC10 to create revision 6 on 08-Mar-82.

1. Declare XBLT to be legal in section 0.

2. Document the differences between the KL and KC implementation of JRST n,. Allow SFM in any section. Add the XJRST (JRST 15,) instruction.

3. Document the occurrance of an IBOX flush on WRCTX, WREBR, CLRPT, and SWPIA.

4. Redesign the WRTMB and RDTMB instructions. Add the WRACT, RDACT instructions.

5. Remove the 18 bit EA restriction from PMOVE and PMOVEM in section 0.

6. Document the changes to the legal PXCT bits.

7. Change SETCU from 700100,,0 to 701000,,0.

8. Change RDUBR from 700040,,0 to 701400,,0.

9. Change the format of E+1 of the WRCTX, RDCTX, and RDUBR blocks to match that of the first word of the flags/PC double word.

10. Make UMOVE/UMOVEM generate an illegal instruction trap if executed in user mode.

11. Add functional descriptions for WRCTX, WREBR, WRTMB, and WRACT.

12. Change the polarity of bit 10 returned in the MAP instruction Ac.

13. Remove function code 3 from the trap function word.

14. Remove the "do not load current AC block values" bit from the interrupt vector word.

15. Move the control bits around in word E of the WRCTX (and RDCTX and RDUBR) block to make it easier to implement.

16. Remove the "do not load AC block values" bit from the WRCTX block.

17. Move the mapping for exec pages 0-337 with TOPS-10 paging from words 600-757 in the EPT to 0-157 in the EPT.

18. Remove the "FPA present" from the hardware options field in APRID and put it in the microcode options field.

19. Remove the JEN instruction.

20. Redefine the algorithm used to compute a physical address for PMOVE, PMOVEM, and the queue instructions.

21. Redefine words 453-454 in the UPT (page fail block) to be "additional data" returned on a page fail. These words will be different for each type of page fail code and will be documented as such.

22. Modify MUUO, page fail, I/O page fail, and traps (MUUO function) to "load" PAB instead of "setting" it.

23. Make super section pointer types 1 (immediate) and 3 (indirect) legal.

24. Start the page fail block in the UPT at offset 451 instead of 450.

25. Change the page number field in an immediate pointer from bits 20-35 to 18-35.

26. Document the format of the "page address word" encountered in pointer traces.

27. Add a description of the debugging instruction implemented by the microcode.

28. Describe the paging information cache.

29. Change the format of the page fail word to include the "level" at which the page fail was detected.

30. Describe the CST update operation.

31. List the processor flags in the description of the Flags/PC double word.

32. Define the page fail codes for TOPS-20 paging.

33. Describe bits 0-10 of the page fail PMA as "reserved" instead of zeros. They come back undefined from the MBOX and it didn't seem particularly necessary to mask them since the monitor probably won't use the word anyway.

34. Note that the trap 1 and trap 2 flags are never stored in the LUUO/MUUO block when a trap that specifies function code 1 (MUUO) or 2 (LUUO) is processed.

35. Note that an instruction that references multiple words (e.g., BLT) will not cause an address break for every word in the address break range if the monitor restarts the instruction with the "inhibit address break" flag set.

36. Add the trap enable bit to WREBR and RDEBR and explain how it works.

37. Add descriptions for the debugging instructions RDTRAX and WRTRAX.

38. Add a new chapter for miscellany and start it by describing the halt status codes.

39. Change the format of the I/O page fail block to make it more consistent with the normal page fail block.

40. Document the effect of a write reference for the various combination of written and modified bits in the translation buffer.

Changes made to revision 4 of KC10 to create revision 5 on 18-Aug-81.

1. Add the definition of the bits returned by the MAP instruction.

2. Define bit 0 of the microcode options field in APRID as "diagnostic microcode loaded".

3. Move the remaining APR flag bits in WRAPR and RDAPR to bits 30 and 31 to right justify them in the field and make them contiguous

4. Add the "VM mode" state bit as bit 10 in word E for WRCTX, RDCTX, and RDUBR. Also use UPT location 431 as the VM mode new PC MUUO dispatch.

5. Add a statement that pager clears caused by CLRPT and WREBR ignore the keep me bit.

6. Declare the results of executing a UMOVE or UMOVEM in user mode as undefined.

7. Define bit 19 in the RNGB? instructions to perform a port init for the port specified by bits 33-35.

8. Put all the accounting meter stuff under an "available with the accounting meter option only" disclaimer.

9. Remove the "Port interrupt logout word" from the I/O page and add 8 "Port interrupt PI status words".

10. Remove the SWPVA instruction from the instruction set.

11. Change the spec to indicate that no flags are loaded on a LUUO.

12. Move the address break condition bits from 0-7 to 10-17 in the WRCTX/RDCTX argument.

13. Require that paging be turned on to load the address break conditions with WRCTX or to read them with RDCTX.

14. Rearrange the WRCTX control bits.

15. Rearrange the bits in RNGB and RNGBW one more time.

16. Remove the cache look and load bits from WREBR and RDEBR and the TB cacheable bit from the page table pointer and add cache on/off controls bits to WREBR and RDEBR.

17. Do not cause an IBOX flush as the result of a cache sweep instruction.

18. Remove time base 1 completely and remove the time base 2 words in the EPT (they're now kept strictly in EBOX scratchpad locations).

19. Redesign the time base and interval timer instructions.

20. Add the RDURTM instruction to read the user runtime meter.

21. Add the page fail formats.

22. Remove the discussion of the UBA from the interrupt vector definitions.

Changes made to revision 3 of KC10 to create revision 4 on 30-Apr-81.

1. Reserve bits 0-10 of the link words of physical queues.

2. Change the console reload bit in RNGB and RNBBW from bit 19 to bit 18.

3. Describe the relationship between the effective address calculation and the reference address for those instructions that use the EA as a physical address.

4. Reserve bits 0-10 of AC for the queue instructions and define the action on bit 0 in the AC on an empty/non-empty queue.

5. Document the MUUO block and the new PC dispatch vectors. Change the spec to reflect the new dispatch algorithm approved by the architecture committee. Move the page fail locations from location starting at 440 to locations starting at 450 to make room for the new MUUO dispatch.

6. Document the LUUO block format and the action of the processor to an LUUO.

7. Clean up the trap function word definitions and change functions 1 and 2 to agree with the decisions of the architecture committee.

8. Clean up the interrupt vector description and document the format of the I/O page fail block in the I/O page.

9. Clean up the queue instruction explanations. Thank you Judy Hall.

10. Redo the entire chapter on virtual addressing (and rename it "paging") to add much more information.

11. Describe the changes to the MAP instruction.

Changes made to revision 2 of KC10 to create revision 3 on 19-Mar-81:

1. Add the SETCU instruction to set the CST update needed bit in each page table entry.

2. Note that PMOVE, PMOVEM and the physical queue instructions do not cause the CST to be updated.

3. Remove the INSQUE and REMQUE instructions and the references to virtual queues.

4. Add pictures for the EPT, UPT and I/O page. Also provide separate picture for TOPS-10 and TOPS-20 for the first two.

5. Change the format of the trap function word that simulates a LUUO to specify the opcode of the LUUO to be used in the function word.

6. Change the definition of the queue manipulation instructions to skip return if the entry has been successfully added to the queue with bit 0 in the AC set if the entry was added to an empty queue. The instruction will not skip if the secondary interlock was timed out.

Changes made to revision 1 of KC10 to create revision 2 on 9-Mar-81:

1. Add definition of "reserved" fields of instruction operands and data.

2. Define all bits of instruction operands and data.

3. Add "KL/KS compatibility" section to each instruction description.

4. Add an enable bit to load the CPU PIA in WRAPR.

5. Change the page number field for WRCTX, WREBR and WRIOP from bits 18-35 to bits 20-35

6. Remove the commitment to make a PXCT of a CLRPT work.

7. Define what "TOPS-10" paging really means.

8. Change the "Interrupt 2080 console" bit in RNGB and RNBGW from bit 18 to bit 32.

9. Change the description of the operation of PMOVE and PMOVEM to do a normal effective address calculation and use the result as a physical address.

10. Change the definition of WRIOP from an immediate mode instruction to one which takes its data from the word addressed by E.

CHAPTER 1

INTRODUCTION

This document describes the operation of the Exec Mode and I/O instructions on the KC10. It also describes the differences between the instruction set implementation on the KL and KS processors and that implemented by the KC10. All I/O instructions are ordinary instructions with the same format as normal instructions (opcode, AC and effective address). The opcodes are in the range 700 thru 777.

Any operation code in the range 700 thru 777, AC number, field or bit not described in this document should be considered reserved to DEC.

Instructions with opcodes between 700 and 737, inclusive, are legal in exec mode or in user I/O mode. Instructions with opcodes between 740 and 777, inclusive, are legal in both exec and user modes.

OPCODE Assignment Map

	0	1	2	3	4	5	6	7
700	APR0	APR1	APR2	-	UMOVE	UMOVEM	PMOVE	PMOVEM
710	RNGB	RNGBW	SNBSY	-	-	-	-	-
720	INSQHI	INSQTI	REMQHI	REMQTI	-	-	-	-
730	-	-	RDTRAX*	WRTRAX*	READTB*	WRITTB*	DUMPTB*	-
740	-	-	-	-	-	-	-	-
750	-	-	-	-	-	-	-	-
760	-	-	-	-	-	-	-	-
770	-	-	-	-	-	-	-	-

* - Debug instructions. Not in production machine.

AC Field Assignments

AC	700	701	702
00	APRID	SETCU	RDSPB
01	-	RDCTX	RDCSB
02	-	CLRPT	RDPUR
03	-	WRCTX	RDCSTM
04	WRAPR	WREBR	RDTMB
05	RDAPR	RDEBR	RDINT
06	SZAPR	WRIOP	RDTIME
07	SNAPR	RDIOP	RDURTM
10	-	RDUBR	WRSPB
11	-	SWPIA	WRCSB
12	-	-	WRPUR
13	-	SWPUA	WRCSTM
14	WRPI	-	WRTMB
15	RDPI	-	WRINT
16	SZPI	-	WRACT
17	SNPI	-	RDACT

CHAPTER 2

INTERNAL I/O INSTRUCTIONS

These instructions control the CPU. They transfer no data between the outside world and the CPU or memory.

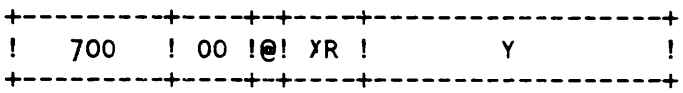
******* WARNING *******

In some instances, fields of the operands of an instruction or fields of the values returned by an instruction are described as "reserved". This means simply that no guarantee is made of the correct operation of an instruction whose "reserved" fields are set non-zero by the program or of the state of the bits in the "reserved" fields returned by an instruction. If you wish to experiment and find a result to your liking, you are hereby warned that your program may well not be compatible with any other processor, with any other model of your processor, with the same model of your processor at some other installation, or even with your own processor running at some other time with a different version of the microcode or Monitor.

Conventions

When the definition of a bit is given in this document, that definition applies when the bit is set to a 1 (unless explicitly stated otherwise). If the bit is set to a zero, the logical complement of the definition applies. For example, bit 1 in word E in APRID below is described as "FPA present". This means that the FPA is present in the machine if the bit is a 1 and not present if the bit is a 0.

APRID



This instruction stores the microcode version number, CPU serial number, and processor options in the words addressed by E and E+1. The format of the first word (E) is:

0-8	Reserved for microcode options.
0	Diagnostic microcode loaded
1	FPA present
9-17	Hardware options
18-35	Processor serial number

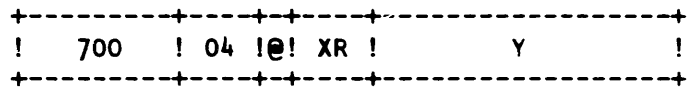
The format of the second word (E+1) is:

0-35	Microcode version number

KL/KS compatibility

The KL and KS returned microcode options, microcode version number, hardware options and processor serial number in one word. The KC returns two words and defines different hardware and microcode option bits.

WRAPR



This immediate mode instruction decodes its effective address to control the processor. The effective address bits are used as follows:

18 Load the PIA for the CPU from bits 33-35.

19 I/O reset. When this bit is set, "reset" is asserted on the KC10 I/O bus. This will reset all the port micromachines (but affects no internal devices, such as the pager and processor flags).

20 Enable conditions selected by bits 24 thru 31 to cause interrupts.

21 Disable interrupts for conditions selected by bits 24 thru 31.

22 Clear flags indicated by bits 24 thru 31.

23 Set flags indicated by bits 24 thru 31.

WARNING

The action of the processor is not defined when both bits 20 and 21 or 22 and 23 are set in the same instruction.

24-31 Selected flags

- 24-29 Reserved
- 30 Console attention
- 31 Power failure

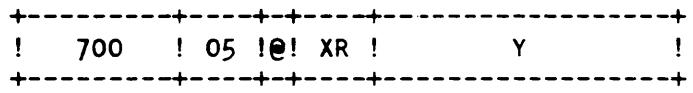
32 Reserved

33-35 PIA for CPU

KL/KS compatibility

The KL and KS unconditionally set the CPU PIA. The KC only sets the PIA if bit 18 is on. The KC also defines different flags in bits 24-31.

RDAPR



This instruction stores the APR status in the word addressed by E. The status is as follows:

0-5	Reserved
6-13	Interrupts enabled
6-11	Reserved
12	Console attention enabled
13	Power failure enabled
14-23	Reserved
24-31	Interrupts pending
24-29	Reserved
30	Console attention
31	Power failure
32	Interrupt requested (IOR of bits 24-31).
33-35	PIA for CPU

KL/KS compatibility

The KC defines different flags in bits 24-31.

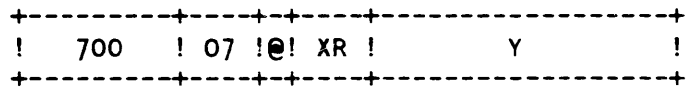
SZAPR

This instruction tests the input conditions from the APR right 18 bits. If all condition bits selected by 1s in E are 0s, skip the next instruction in sequence.

KL/KS compatibility

Functionally identical to CONSZ APR,E.

SNAPR

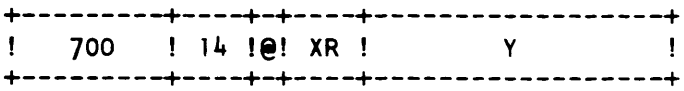


This instruction tests the input conditions from the APR right 18 bit. If any condition bit selected by 1 in E is 1, skip the next instruction in sequence.

KL/KS compatibility

Functionally identical to CONSO APR,E.

WRPI



This immediate mode instruction decodes its effective address to control the priority interrupt system. The effective address bits are used as follows:

18-21 Reserved

22 Turn off program requests on the levels selected by 1s in bits 29-35.

23 Clear PI system.

24 Initiate interrupts on the levels selected by 1s in bits 29-35. Such interrupts vector through the software interrupt vector words in the I/O page. An interrupt is not initiated on a level unless the PI system and the requested level are on.

25 Turn on the levels selected by 1s in bits 29-35.

26 Turn off the levels selected by 1s in bits 29-35.

WARNING

The action of the processor is not defined when both 25 and 26 or 22 and 24 are set in the same instruction.

27 Turn off PI system

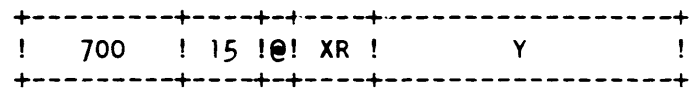
28 Turn on PI system

29-35 Select levels for bits 22, 24, 25, and 26.

KL/KS compatibility

The KL used bits 18-20 to force parity errors. The KL initiated an interrupt on a level as the result of a 1 in bit 24 even if the specified level was off. Although the KS is documented to act in the same manner, it did not initiate the interrupt unless the level was on.

RDPI



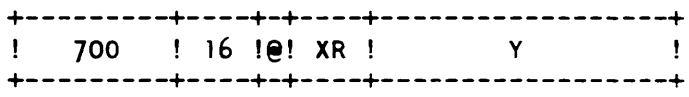
This instruction stores the PI status in the word addressed by E. The status is a follows:

0-10	Reserved
11-17	Program requests on levels.
18-20	Returned as zeroes
21-27	Interrupt in progress on levels.
28	PI system on.
29-35	Levels on.

KL/KS compatibility

The KL returned the state of the forced parity error bits in bits 18-20. Otherwise, it is functionally equivalent to CONI PI,E on the KL and RDPI E on the KS.

SZPI

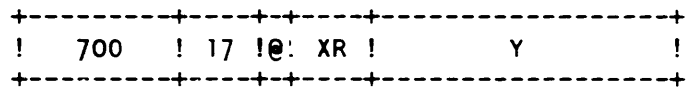


This instruction tests the input conditions from the PI right 18 bits. If all condition bits selected by 1s in E are 0s, skip the next instruction in sequence.

KL/KS compatibility

Functionally equivalent to CONSZ PI,E.

SNPI

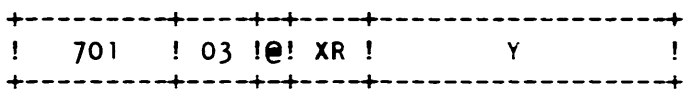


This instruction tests the input conditions from the PI right 18 bit. If any condition bit selected by 1 in E is 1, skip the next instruction in sequence.

KL/KS compatibility

Functionally equivalent to CONSO PI,E.

WRCTX



This instruction loads user process context from a variable number of words addressed by E through E+4 depending on the flag bits in the first word. The process context includes previous and current AC blocks, previous context section, user base register, virtual machine simulation mode enable, and address break enable and conditions.

The format of the first word (E) is:

0 Load current and previous AC block numbers (CAB, PAB) from bits 18-23 of word E+1.

1 Load previous context section (PCS) from bits 24-35 of word E+1.

2 Load the user base register (UBR) from bits 20-35 of this word.

3 Clear all pages from the hardware page table (include "kept" pages) when the pager clear is done.

4 Do not change the state of the user runtime meter (nether update it into the old PT or load it from the new UPT).

5 Load VM-moue enable from bit 6 of this word.

6 Enable virtual machine (VM-mode) simulation mode for this user context.

7 Inhibit all address break conditions for the next instruction executed. The effect of setting this bit is to set the inhibit address break PC flag for the next instruction.

8 Load address break conditions from the words at E+2 through E+4.

9 Load address break enable from bit 10 of this word.

10 Enable address break using the existing conditions (the conditions may be also be changed with the same instruction).

11-19 Reserved

20-35 Physical page number of UPT.

The format of the second word (E+1) is:

0-17 Reserved

18-20 Current AC block

21-23 Previous AC block

24-35 Previous Context Section

The format of the third word (E+2) is:

0-9 Reserved

10 Enable address break for a normal fetch of an instruction in the program under control of PC.

11 Enable address break for any reference that reads except the normal fetch of an instruction.

12 Enable address break for any reference that writes.

13 Enable address break for a reference made in user virtual address space. (0 selects executive space).

14 Enable address break for a reference made by the CPU to memory.

15 Enable address break for a reference made by a port to memory.

16 Enable address break for a physical memory reference. (0 selects virtual memory references).

17 Compare only bits 18 through 35 of the reference address with the address range when doing address compares.

18-35 Reserved

The format of the fourth word (E+3) is:

0-5 Reserved

6-35 The lower bound break address.

The format of the fifth word (E+4) is:

0-5 Reserved

6-35 The upper bound break address.

In word E, bits 0-2, 5, and 7-9 control the action of this instruction; when a bit is 0, the corresponding action is ignored. The actions are as follows:

1. If bit 0 is on, load CAB and PAB from bits 18-20 and 21-23, respectively, from word E+1. If bit 0 is off, do not change CAB and PAB.

2. If bit 1 is on, load PCS from bits 24-35 of word E+1. If bit 1 is off, do not change PCS.

3. If bit 2 is on, perform the following functions:

    1. If bit 3 in E is 0, clear all pages except those marked "kept" in the page table. If bit 3 is 1, clear all entries.

    2. Load bits 20-35 of E into the User Base Register

    3. If bit 4 in E is 0, perform the following functions:

        1. Update the user runtime meter into the previous UPT by simulating a RDURTM instruction and storing the resulting doubleword in the previous UPT in locations 504-505.

        2. Load the user runtime meter kept in the EBOX internal registers from locations 504-505 of the new UPT.

        If bit 4 is a 1, do not update the user runtime meter into the previous UPT and do not reload it from the new UPT.

4. Clear the internal cache of paging information kept by the EBOX and re-initialize it with the first super section pointer from the EPT and the UPT (offset 520).

5. If bit 5 is on, perform the following functions:

   1. If bit 6 is on, enable virtual machine simulation mode (VM-mode) for this user context. If bit 6 is off, disable VM-mode for this user context.

   If bit 5 is off, do not change the state of VM-mode.

6. If bit 7 is on, inhibit all address break traps for the next instruction executed after the WRCTX. If bit 7 is off, do not inhibit address break traps.

7. If bit 8 is on, load the address break conditions from the words at E+2 through E+4. If bit 8 is off, the address break conditions remain unchanged.

NOTE

Paging must be enabled (with WREBR bit 4) to load the address break conditions. If paging is not enabled, the result of loading the address break conditions is undefined.

8. If bit 9 is on, perform the following functions:

   1. If bit 10 is on, turn on address break using the existing break conditions (which may be set in the same instruction by setting bit 8 to a one). If bit 10 is off, turn off address break but leave the break conditions unchanged.

   If bit 9 is off, do not change the state of address break enable.

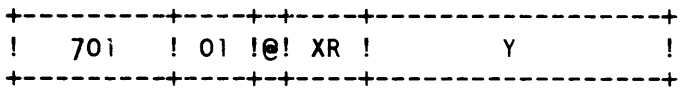
9. Flush and restart the IBOX.

See the chapter on Address Break for a functional description.

KL/KS compatibility

The KL set current and previous AC blocks, previous context section, and the physical page number of the UPT with a DATAO PAG,E. Address break conditions and address were set with a DATAO APR,E. The KS set current and previous AC blocks, and the physical page number of the UPT with a WRUBR E. The KC WRCTX instruction combines the functions of these instructions.

RDCTX



This instruction stores the user process context in the five words addressed by E through E+4 in exactly the same format as used by WRCTX. In order to allow these words to be used directly in a WRCTX instruction, bits 0-2, 5, 8, and 9 are set to 1 and bits 3, 4, and 7 are set to 0 in E.

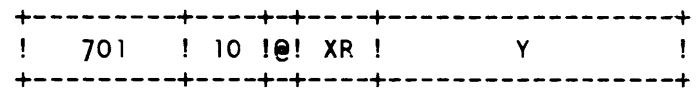
NOTE

Paging must be enabled (with WREBR bit 4) to read the address break conditions. If paging is not enabled, the values returned in words E+2 through E+4 (address break conditions, lower and upper address bounds) are undefined.

KL/KS compatibility

The KL returned current and previous AC blocks, previous context section, and the physical page number of the UPT with a DATAI PAG,E. Address break conditions were returned with a DATAI APR,E. The KS returned current and previous AC blocks, and the physical page number of the UPT with a RDUBR E. The KC RDCTX instruction combines the functions of these instructions.

RDUBR

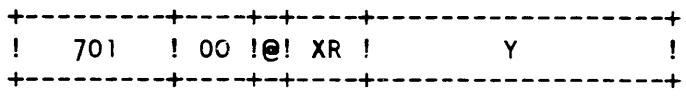


This instruction stores the user process context in the two words addressed by E and E+1 in exactly the same format as the first two words used by WRCTX. In order to allow these words to be used directly in a WRCTX instruction, bits 0-2, 5, and 9 are set to 1 and bits 3,4, 7, and 8 are set to 0 in E.

KL/KS compatibility

The KL returned current and previous AC blocks, previous context section, and the physical page number of the UPT with a DATAI PAG,E. The KS returned current and previous AC blocks, and the physical page number of the UPT with a RDUBR E.

SETCU



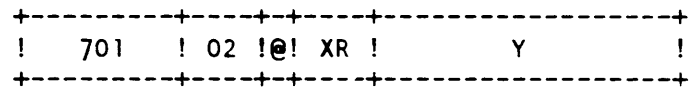
701	00	@	XR	Y

This instruction causes the "CST update needed" bit to be set in each entry in the hardware page table so that the next virtual reference to each page will cause the EBOX to update the CST entry for the page. This instruction blocks further CPU activity until all bits are set in the page table. Upon completion of the operation, the IBOX is flushed and restarted. The indirect, index register, and Y fields of this instruction are not used and are reserved.

KL/KS compatibility

No functional equivalent on the KL or KS.

CLRPT



This immediate mode instruction clears the hardware page table entry for the virtual page addressed by E so that the next virtual reference to a word in that page will cause an EBOX page fail trap to occur. The page table "keep me" bit is ignored by this instruction and an uncoditional clear is done. The actions performed by this instruction are as follows:

1. Clear the translation buffer entry for the virtual page addressed by E.

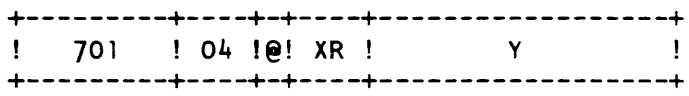
2. Clear the internal cache of paging information kept by the EBOX (but do not clear the first exec and user super section pointer).

3. Flush and restart the IBOX.

KL/KS compatibility

Functionally equivalent to CLRPT E on the KL.

WREBR



This instruction loads the exec mode context from the word addressed by E. The exec mode context includes cache enable, type of paging, pager enable, trap enable, and the exec base register. The format of the word is:

0	Load the cache enable bit from bit 1. This bit should never be set by the monitor. The cache will only be turned off as the result of a serious error and should remain off until the problem is fixed. The ability to enable and disable the use of cache is provided strictly for diagnostics.
1	Enable use of the cache for all references. Enabling the use of the cache with this bit does not enable the use of all four cache quadrants if one has been turned off because of an error. It simply causes the cache to be used for any quadrants that are on.
2	Reserved
3	TOPS-20 paging (See the chapter on Paging)
4	Pager enable.
5-7	Reserved
8	Load trap enable from bit 9
9	Enable full processing of traps, LUUOs, MUUOs, and page fails
10-19	Reserved
20-35	Physical page number of EPT.

The actions performed by this instruction are as follows:

1. If bit 8 is a 1, perform the following functions:

    1. If bit 9 is a 1, enable full processing of traps, LUUOs, MUUOs, and page fails by the monitor as described in the appropriate sections below.

2. If bit 9 is a 0, change the prccessing of certain processor conditions as follows:

   1. Traps. Treat trap 1, 2, and 3 conditions as if the trap function word had specified "ignore trap".

   2. LUUOs. Process section 0 LUUOs in the normal manner. Halt the machine on LUUOs executed in non-zero sections.

   3. MUUOs. Halt the machine.

   4. Page fails. Process page fails that can be resolved by the EBOX microcode alone in the normal manner. Halt the machine on page fails that must be processed by the monito-.

If bit 8 is a 0, do not change the state of trap enable

2. If bit 0 is a 1, perform the following functions:

   1. If bit 1 is a 1, enable the use of the cache as described above. If bit 1 is a 0, disable the use of the cache.

If bit 0 is a 0, do not change the state of the cache enable.

3. If bit 4 is a 1, perform the following functions:

   1. Enable the use of the paging hardware for virtual-to-physical translations for memory references and select the type of paging to be used.

   2. If bit 3 is a 1, select TOPS-20 style paging. If bit 3 is a 0, select TOPS-10 style paging.

NOTE

The type of paging referred to as "TOPS-10 paging" is really that implemented by the KI10 and used by the TOPS-10 monitor prior to the 7.02 release. It is used here for historical reasons only.

If bit 4 is a 0, disable paging so that all memory references are to physical locations unpaged. Note that disabling the pager does not mean there can be no page failures, as these can be caused by conditions that have nothing to do with paging.

CAUTION

Paging can be disabled only for kernel mode. A user mode process will not run correctly unless the pager is turned on.

4. Load bits 20-35 into the Exec Base Register.

5. Clear the internal cache of paging information kept by the EBOX and re-initialize it with the first super section pointer from the EPT and UPT (offset 520).

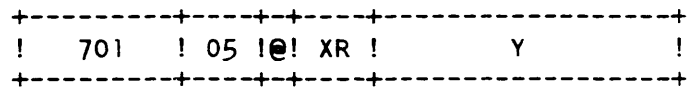
6. Invalidate all entries in the MBOX translation buffer, ignoring the state of the "keep-me" bits.

7. Flush the IBOX and restart it.

KL/KS compatibility

The KL set exec mode context with the IMMEDIATE MODE instruction CONO PAG,E. The KS set exec mode context with the IMMEDIATE MODE instruction WREBR E.

RDEBR



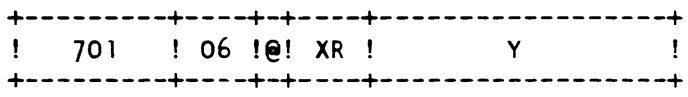
This instruction stores the exec mode context in the word addressed by E. The format of the word is:

0	Returned as zero
1	The use of cache is enabled for all references. This bit being set does not imply that all four cache quadrants are in use. It simply means that the cache will be used for any quadrants that are on.
2	Reserved
3	TOPS-20 paging (See the chapter on Paging)
4	Pager enable.
5-7	Reserved
8	Returned as 0
9	Full processing of traps, LUUOs, MUUOs, and page fails is enabled
10-19	Reserved
20-35	Physical page number of EPT.

KL/KS compatibility

The KL and KS returned the exec mode context with CONI PAG,E and RDEBR E, respectively. The KC returns the same fields as the KL CONI PAG,E but the bit positions have changed because of the increased size of the EPT page number.

WRIOP



This instruction loads the I/O page base register from the word addressed by E. The format of the word is:

0-19 Reserved

20-35 Physical page number of the I/O page

The I/O page base register is set to physical page 1 by the console under the following conditions:

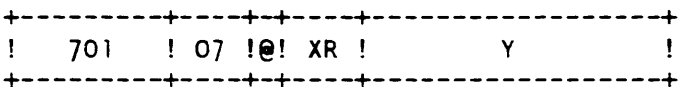
1. TBS

See the chapter on Special System Pagers for a description of the laycut of this page.

KL/KS compatibility

No functional equivalent on the KL or KS.

RDIOP



This instruction returns the value of the I/O page base register and stores it in the word addressed by E. The format of the word is:

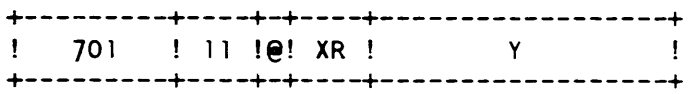
0-19 Reserved

20-35 Physical page number of the I/O page

KL/KS compatibility

No functional equivalent on the KL or KS.

SWPIA



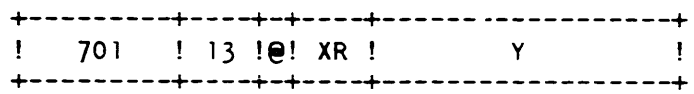
Sweep Cache, Invalidate All Pages

Clear the valid and written state in all cache entries. This instruction blocks further CPU activity until the cache sweep is complete. Upon completion of the operation, the IBOX is flushed and restarted. The indirect, index register, and Y fields of this instruction are not used and are reserved.

KL/KS compatibility

The KS had no cache sweep instructions. The KL allowed other requests to happen in parallel with the sweep, setting sweep busy and sweep done in the APR status to indicate the sweep-in-progress interval.

SWPUA



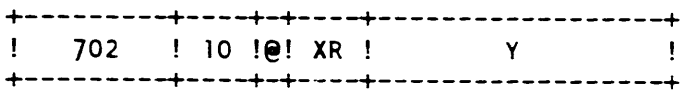
Sweep Cache, Unload All Pages

Write into storage all cache entries that are in the written state. Invalidate all entries (i.e. clear valid and written state). This instruction blocks further CPU activity until the cache sweep is complete. The indirect, index register, and Y fields of this instruction are not used and are reserved.

KL/KS compatibility

The KS had no cache sweep instructions. The KL allowed other requests to happen in parallel with the sweep, setting sweep busy and sweep done in the APR status to indicate the sweep-in-progress interval.

WRSPB



Write SPT Base Register

This instruction loads the SPT base register from the word addressed by E. The word format is:

0-10 Reserved

11-35 Physical address of the start of the SPT.

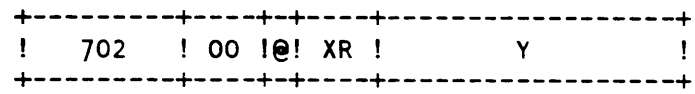
BASE REGISTER FORMAT

The SPT base register is loaded with a physical word address. The address need not be on a page boundary and may be any location in physical memory. There is no range check on SPT offsets. The monitor is assumed to always put correct data into the SPT base register.

KL/KS compatibility

Functionally identical to the KS WRSPB E instruction. The KL kept the SPT address in AC block 6.

RDSPB



Read SPT Base Register

This instruction stores the SPT base register in the word addressed by E. The format of the word is:

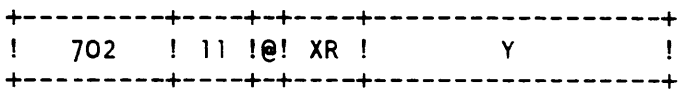
0-10 Reserved

11-35 Physical addresss of the SPT

KL/KS compatibility

Functionally identical to the KS RDSPB E instruction. The KL kept the address of the SPT in AC block 6.

WRCSB



Write Core Status Table Base Register

This instruction loads the CST base register from the word addressed by E. The word format is:

0-10 Reserved

11-35 Physical address of the start of the CST. If this address is zero, the microcode will make no CST references.

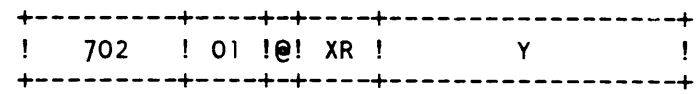
BASE REGISTER FORMAT

The CST base register is loaded with a physical word address. The address need not be on a page boundary and may be anyplace in physical memory. There is no range check on CST offsets. The monitor is assumed to always put correct data into the CST base register.

KL/KS compatibility

Functionally identical to the KS WRCSB E instruction. The KL kept the CST address in AC block 6.

RDCSB



Read Core Status Table Base Register

This instruction stores the CST base register in the word addressed by E. The format of the word is:

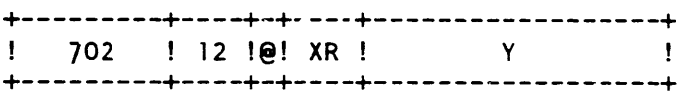
0-10 Reserved

11-35 Physical addresss of the CST

KL/KS compatibility

Functionally identical to the KS RDCSB E instruction. The KL kept the address of the CST in AC block 6.

WRPUR



Write Process Use Register

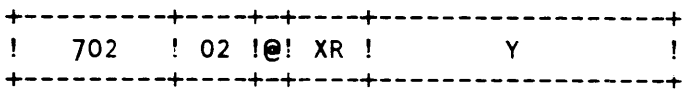
This instruction loads the process use register from the word addressed by E. The microcode updates a CST entry by ANDing the CST mask word (see below) with the entry and ORing the process use register into the entry.

See the chapter on paging for the format of the process use register.

KL/KS compatibility

Functionally identical to the KS WRPUR E instruction. The KL kept the process use register in AC block 6.

RDPUR



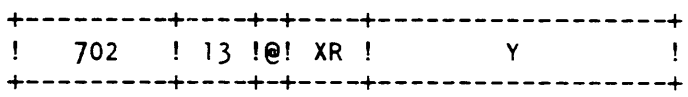
Read Process Use Register

This instruction stores the process use register in the word addressed by E.

KL/KS compatibility

Functionally identical to the KS RDPUR E instruction. The KL kept the process use register in AC block 6.

WRCSTM



Write CST Mask Register

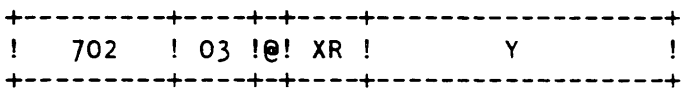
This instruction loads the CST mask register from the word addressed by E. The microcode updates a CST entry by ANDing the CST mask word with the entry and ORing the process use register into the entry.

See the chapter on paging for the format of the CST mask register.

KL/KS compatibility

Functionally identical to the KS WRCSTM E instruction. The KL kept the CST mask register in AC block 6.

RDCSTM



Read CST Mask Register

This instruction stores the CST mask register in the word addressed by E.

KL/KS compatibility

Functionally identical to the KS RDCSTM E instruction. The KL kept the CST mask register in AC block 6.

WRTMB

Write Time Base Controls

This immediate instruction decodes its effective address to control the time base and the inverval timer. The effective address bits are used as follows:

18	Load PI assignment for interval timer from bits 33-35.
19	Load time base controls from bits 20 and 23.
20	Clear time base.
21-22	Reserved
23	Turn on time base.
24-32	Reserved
33-35	PIA for interval timer.

The actions of this instruction are as follows:

1. If bit 18 is a 1, load the interval timer PI assignment from bits 33-35. If bit 18 is a 0, do not change the interval timer PIA.

2. If bit 19 is a 1, perform the following operations:

    1. If bit 23 is a 1, turn on the time base. If bit 23 is a 0, turn off the time base.

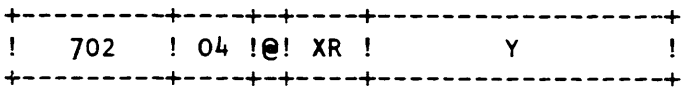
    2. If bit 20 is a 1, clear the time base. If bit 20 is a 0, it is ignored.

    If bit 19 is a 0, do not change the state of the time base.

KL/KS compatibility

The KL CONO MTR, instruction controls the time base and the interval timer PIA in a manner very analogous to this instruction. The KS had no equivalent instruction.

RDTMB



Read Time Base Enables

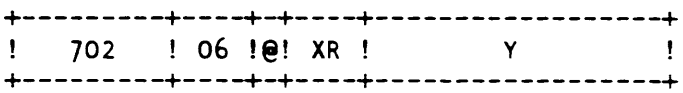
This instruction reads the status of the time base and the interrupt level assigned to the interval timer into the word addressed by E. The status is as follows:

0-22	Reserved
23	Time base on.
24-32	Reserved
33-35	PIA for interval timer.

KL/KS compatibility

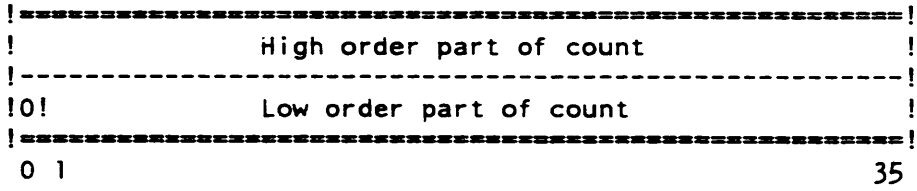
The KL CONI MTR, instruction returned the time base enable and interval timer PIA in a manner very analogous to this instruction. The KS had no direct equivalent.

RDTIME



Read Time Base Value

This instruction updates the time base double word from the hardware counter and returns the updated double word in the words addressed by E and E+1. The double word is double precision integer in 1 microsecond units with the following format:



KL/KS compatibility

The KL RDTIME instruction also updated the time base double word kept in EPT locations 510 and 511. The KS RDTIM instruction returned the double word in the same manner as this instruction.

RDACT

702	17	@	XR	Y	

Read accounting information

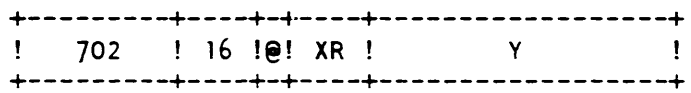
This instruction reads the status of the user runtime meter into the word addressed by E. The status is as follows:

0-18	Reserved
19	The user runtime meter has been enabled to count during exec PI time.
20	The user runtime meter has been enabled to count during exec non-PI time.
21	User runtime meter on.
22-35	Reserved

KL/KS compatibility

The KL CONI MTR, instruction returned the accounting meter controls in a manner very analogous to this instruction. The KS had no direct equivalent.

WRACT



Write accounting information

This immediate instruction decodes its effective address to control the user runtime meter. The effective address bits are used as follows:

18	Load user runtime meter controls from bits 19-21.
19	Enable user runtime meter count during exec PI time.
20	Enable user runtime meter count during exec non-PI time.
21	Turn on user runtime meter
22-35	Reserved

The actions of this instruction are as follows:

1. If bit 18 is a 1, perform the following operations:

    1. If bit 19 is a 1, enable the user runtime meter to count during exec PI processing. If bit 19 is a 0, disable the user runtime meter from counting during exec PI processing.

    2. If bit 20 is a 1, enable the user runtime meter to count during exec non-PI processing. If bit 20 is a 0, disable the user runtime meter from counting during non-PI processing.

    3. If bit 21 is a 1, turn on the user runtime meter. If bit 21 is a 0, turn off the user runtime meter.

    If bit 18 is a 0, do not change the state of the user runtime meter.

KL/KS compatibility

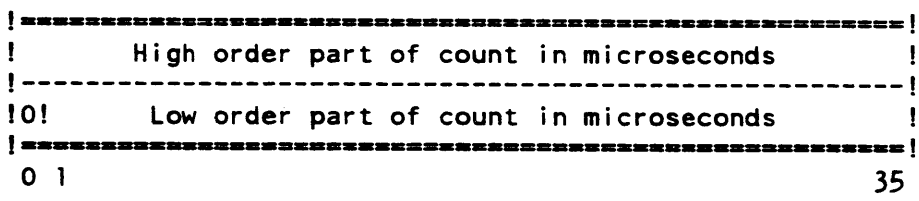
The KL CONO MTR, instruction controls the accounting meters in a manner very analogous to this instruction. The KS had no equivalent instruction.

RDURTM

702	07	@	XR	Y

Read User Runtime Meter

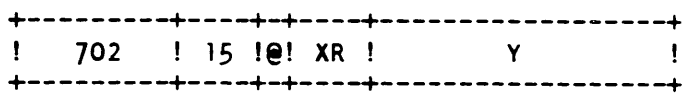
This instruction updates the user runtime meter double word from the hardware counter and returns the updated double word in the words addressed by E and E+1. The double word is double precision integer in 1 microsecond units with the following format:



KL/KS compatibility

The KL and KS had no comparable instructions.

WRINT



Write Interval Timer

This immediate mode instruction decodes its effective address to setup the interval timer. The effective address bits are used as follows:

18	Clear interval timer.
19-20	Reserved
21	Turn interval timer on.
22	Clear interval flags.
23	Reserved
24-35	Interval period.

A 1 in bit 18 clears the counter and can be given simultaneously with a 1 or 0 in bit 21 to turn the counter on or off. A 1 in bit 22 clears both Interval Done and Interval Overflow. If the counter is on, Interval Done will set when the count reaches the value specified by bits 24-35.

KL/KS compatibility

This instruction is functionally equivalent to the KL CONO TIM, instruction.

RDINT

Read The Interval Register

Read the status of the interval timer into the word addressed by E. The status is as follows:

0-5	Reserved
6-17	Interval count (current contents of the counter).
18-20	Reserved
21	Interval timer on.
22	Interval timer done (causes interrupt).
23	Overflow (implies bit 22).
24-35	Interval period.

Bits 22 and 23 are the counter flags; note that Interval timer done can be set alone, but a 1 in bit 23 implies a 1 in bit 22 as well. Bits 24-35 are the period supplied by WRINT, and bits 6-17 are the current contents of the counter.

KL/KS compatibility

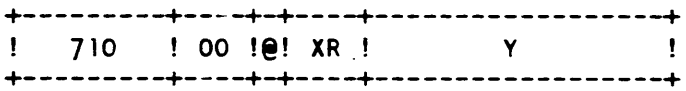
This instruction is functionally equivalent to the KL CONI TIM, instruction

CHAPTER 3

EXTERNAL I/O INSTRUCTIONS

The external I/O instructions on the KC10 allow a program to communicate with the I/O ports and the console. In particular they will manipulate the I/O Command/Response Queues and Port Doorbell mechanism. See the "I/O Bus Spec." in the 2080 EFS for a complete description of the Queue and Doorbell features. The interface to the KC10 ports is primarily data areas called "mailboxes" and a doorbell. It is the doorbell mechanism that the following instructions manipulate. The Command/Response Queues will be covered by the queue instructions in the next section. In general the BUSY and RING signals work as follows: The CPU can assert RING on the I/O Bus if BUSY is clear. Upon setting RING and a port number, the CPU must observe BUSY setting and then clearing before it can assume that the I/O Port has seen its command. The following 2 instructions (RNGB and RGNBW) will be skipping instructions if no bus timeouts occur. The Console does not use this protocol and therefore RING and BUSY signals are ignored if any console functions are requested by RNGB or RNGBW.

RNGB



Ring Doorbell

This immediate mode instruction will assert a port or console number on the KC10 I/O bus and set RING ("doorbell"). If no bus timeout errors occur the next instruction is taken from PC+2 (ie. the instruction skips). The EA of this instruction is the port number and is interpreted as follows:

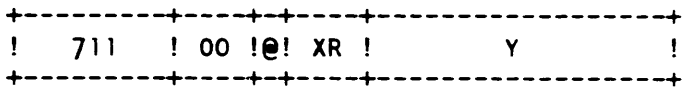
18	Cause console to reload (electronic boot finger)
19	Initialize the port specified by bits 33-35 to the power-up state.
20	Interrupt KC10 Console
21-32	Reserved
33-35	Port number (Must be zero if bit 18 or bit 20 is set)

Setting both bits 18 and 20 in a single instruction will produce unspecified results.

NOTE

This instruction waits for BUSY to be clear before asserting RING on the KC10 I/O bus. If for any reason BUSY does not set or clear in ??ms, the CPU takes the next instruction from PC+1 rather than PC+2.

RNGBW



711	00	@	XR	Y	

Ring Doorbell and Wait (for BUSY to clear).

This immediate mode instruction will assert a port or console number on the KC10 I/O bus and set RING ("doorbell"). The instruction then waits for BUSY to clear. If no bus timeout errors occur the next instruction is taken from PC+2 (ie. the instruction skips). The EA of this instruction is the port number and is interpreted as follows:

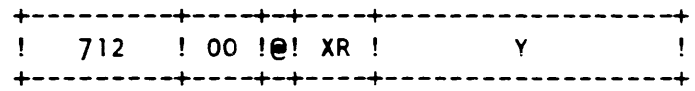
18	Cause console to reload (electronic boot finger)
19	Initialize the port specified by bits 33-35 to the power-up state.
20	Interrupt KC10 Console (does not wait for BUSY- same function as RNGB)
21-32	Reserved
33-35	Port number (Must be zero if bit 18 or bit 20 is set)

Setting both bits 18 and 20 in a single instruction will produce unspecified results.

NOTE

This instruction waits for BUSY to be clear before asserting RING on the KC10 I/O bus. If for any reason BUSY does not set or clear in ??ms, the CPU takes the next instruction from PC+1 rather than PC+2.

SNBSY



Skip if BUSY not set

This instruction skips to PC+2 if the BUSY line of the KC10 I/O bus is not set. When used in combination with the RNGB instruction, one can achieve the identical effect of RNGBW as follows:

```
RNGB    pn              ; Assert RING to port "pn"
SNBSY                   ; Busy set?
 JRST   .-1             ; Yes, wait.
...                     ; No - proceed
```

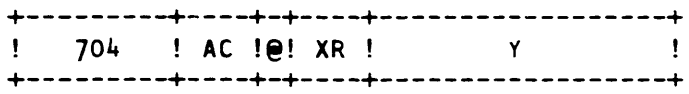
CHAPTER 4

OTHER I/O INSTRUCTIONS

This chapter describes other instructions with opcodes in the range 700-737 that cannot be considered as either internal I/O or external I/O instructions.

Like all instructions whose opcode is in the range 700-737, inclusive, these instructions may only be executed in user mode if user I/O is set. If these instructions are executed in user mode without user I/O, they execute as an MUUO, trapping through the user undefined I/O opcode dispatch in location 435 of the UPT.

UMOVE



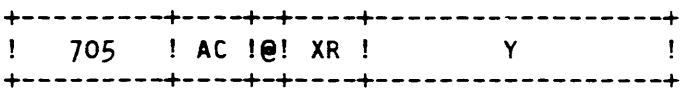
User Move from Memory

Load previous context memory location addressed by E into AC.

NOTE

This is just a replacement instruction for XCT 4,[MOVE AC,MEMORY].

UMOVEM



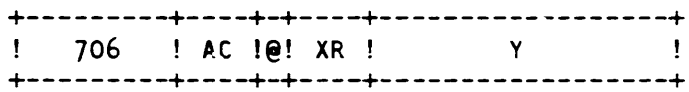
User Move to Memory

Store AC into previous context memory location addressed by E.

NOTE

This is just a replacement instruction for XCT 4,[MOVEM AC,MEMORY].

PMOVE



Physical Move from Memory

Perform a physical EA-calc using the word addressed by E, then load the physical memory location addressed by the result of the EA-calc into the AC.

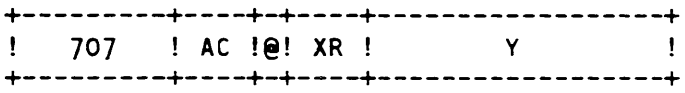
See the chapter on queues and queue manipulation instructions for a discussion of the physical EA-calc algorithm.

NOTE

Effective addresses 0-17 will reference physical memory 0-17, not the ACs.

No CST update will be performed as the result of this instruction

PMOVEM



Physical Move to Memory

Perform a physical EA-calc using the word addressed by E, then store AC into the physical memory location addressed by the result of the EA-calc.

See the chapter on queues and queue manipulation instructions for a discussion of the physical EA-calc algorithm.

NOTE

Effective addresses 0-17 will reference physical memory 0-17, not the ACs.

No CST update will be performed as the result of this instruction.

CHAPTER 5

OTHER FUNCTIONAL CHANGES

This chapter describes non-I/O instructions and other operations whose functionality has changed from previous machines.

MAP

```
+---------+----+-+----+-----------------+
!   257   ! AC !@! XR !        Y        !
+---------+----+-+----+-----------------+
```

Map an address

If the pager is on and the processor is in executive or user I/O mode, this instruction reads the hardware page table location corresponding to the effective address. If the page table contains a valid mapping for that page, the mapping is returned in the format described below. If the page table does not contain a valid mapping for the page, the EBOX microcode does a page refill pointer chase to compute the mapping and returns that in the format described below. The result of the mapping is returned in the AC.

This instruction does not change the hardware page table mapping for the page specified by the effective address calculation.

This instruction cannot be performed in a user program unless user I/O is set. Instead of mapping the address, it executes as an MUUO, dispatching through the user undefined I/O opcode dispatch in location 435 of the UPT. If the pager is off, the results of the instruction are undefined.

The format returned by the MAP instruction in the AC is as follows:

```
!=============================================================!
!H!K!?!V!C!W!M!?!K!U!T!                Physical             !
!D!1!?!L!U!T!D!?!2!S!B!                Address              !
!=============================================================!
 0 1 2 3 4 5 6 7 8 9 1 1                                     3
                     0 1                                     5
```

The fields are as follows:

0 This bit is a one if the instruction failed to generate a valid mapping because of a hardware error. In this case, bits 1-4 contain a failure code instead of the bits described below. These codes are the same as those returned for a page fail with bit 0 set in the page fail word.

1 See the description of bit 8 below.

2 The state of this bit is undefined.

3 If this bit is a one, the rest of the information returned, including the physical address is valid. If this bit is a zero, there is no valid mapping for the virtual address and bits 18-35 contain the reason the microcode couldn't find a valid mapping. This information has the same format as bits 18-35 of the page fail word that would be returned if the specified page was referenced.

4 If this bit is a one, the next virtual reference to the page being mapped will cause the EBOX microcode to perform a CST update operation for the page.

5 If this bit is a one, the page being mapped is writable. If this bit is a zero, the page being mapped is write protected.

6 If this bit is a one, the page being mapped has been modified since being brought into memory, i.e., the page is newer than any backup copy. If this bit is a zero, the page has not been modified since being brought into memory.

7 The state of this bit is undefined.

8 If bit 1 or this bit is a one, the hardware virtual-to-physical mapping for the page being mapped will not be invalidated on a conditional pager clear. If both bit 1 and this bit are zeros, the mapping will be invalidated on all pager clears.

9 If this bit is a one, the mapping for this page is in user space. If this bit is a zero, the mapping for this page is in executive space.

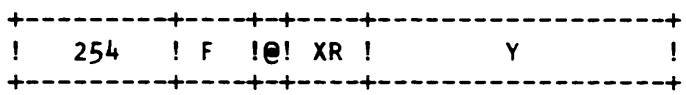
10 If this bit is a zero, the microcode found valid information in the hardware page table for this mapping. If this bit is a one, the microcode performed a pointer trace to compute the mapping.

11-35 The physical address corresponding to the virtual address of the effective address calculation.

KL/KS compatibility

The KL and KS returned different bits describing the mapping.

JRST



Jump and Restore

The KC10 implementation of JRST is very similar to the KS10 and extended KL10 implementation with several exceptions. The exceptions are as follows:

F Mnemonic Function

05 XJRSTF Restore the program flags (as appropriate for the mode of the processor) and PC from the flag-PC double word in locations E and E+1 and continue performing instructions in normal sequence beginning at the location then addressed by PC. If the instruction is executed in exec mode, also restore CAB, PAB, and PCS from the first word of the flag-PC double word.

06 XJEN Restore the level on which the highest priority interrupt is currently being held and then perform an XJRSTF.

07 XPCW Save the program flags, CAB, PAB, PCS, and PC in a flag PC double word in locations E and E+1. Then restore the program flags, CAB, and PC from the flag-PC double word in locations E+2 and E+3 and continue performing instructions in normal sequence beginning at the location then addressed by PC. Do not restore PAB or PCS from E+2.

10 Always execute as an MUUO through the I/O undefined opcode new PC words in the UPT.

12 JEN Always execute as an MUUO through the I/O undefined opcode new PC words in the UPT. Since the KC10 always stores flag-PC double words in XJEN format, there is no need for JEN.

14 SFM Save the program flags in bits 0-12 of the word addressed by E and clear bits 13-17. If the instruction is executed in exec mode, store CAB, PAB, and PCS in bits 18-20, 21-23, and 24-35, respectively, of the same word. If the instruction is executed in user mode, clear bits 18-35. This instruction is legal in any section.

15 XJRST Restore the PC from bits 6-35 of the word addressed by E and continue performing instructions in normal sequence beginning at the location then addressed by PC. Do not

change the program flags, CAB, PAB, or PCS.

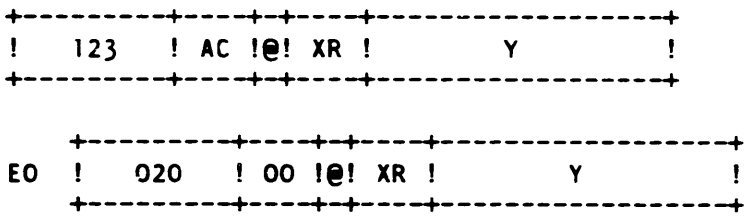
For each of the 16 possible JRST functions, the table given below indicates where each form of the instruction is legal. The meanings of the symbols used to define the legal domains of the functions are as follows:

Yes	Legal everywhere
Z	Legal only in section zero
K	Legal only in kernel (executive) mode
No	Legal nowhere
-H	Legal where indicated by first symbol but causes a halt

If the JRST function is illegal in the mode or context in which it is executed, the instruction traps as an MUUO through the I/O undefined opcode new PC words in the UPT.

Function	Mnemonic	Legal domain
JRST 0,	JRST	Yes
JRST 1,	PORTAL	Yes
JRST 2,	JRSTF	Z
JRST 3,		No
JRST 4,	HALT	K-H
JRST 5,	XJRSTF	Yes
JRST 6,	XJEN	K
JRST 7,	XPCW	K
JRST 10,		No
JRST 11,		No
JRST 12,		No
JRST 13,		No
JRST 14,	SFM	Yes
JRST 15,	XJRST	Yes
JRST 16,		No
JRST 17,		No

XBLT



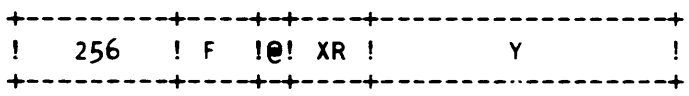
Extended Block Transfer

Move a block of words from one area of memory to another. The block size and the locations of the source and destination areas are defined by the contents of a block of three accumulators as described in the Hardware Reference Manual. This instruction may be exectued in both section zero and non-zero sections and may be used to transfer data between any arbitrary sections.

KL/KS compatibility

The KL and KS exectued this instruction as an MUUO if it was exectuted in section zero.

PXCT



Previous Context Execute

This instruction executes another instruction with certain specified references in the previous context. The operations performed in the previous context are determined by the bits in the AC field. The KC10 changes the legal bits that may be set for PXCTed stack and MOVSLJ instructions. The following table gives the only legal combinations for each type:

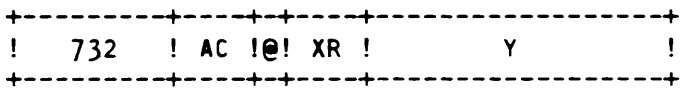
Instructions	9	10	11	12	References
Stack	0	1	0	0	Memory data
	1	1	0	0	E, memory data
MOVSLJ	0	0	0	1	Destination
	0	0	1	0	Source
	0	0	1	1	Source, destination

CHAPTER 6

SPECIAL DEBUGGING INSTRUCTIONS

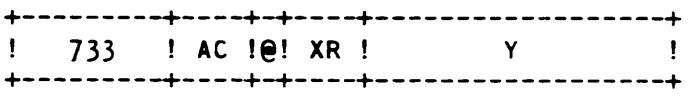
This chapter describes several instructions that have been added to the instruction set to aid in debugging the hardware and microcode. They will not appear in the final production microcode and are documented here only for completeness.

RDTRAX



Read tracks buffer information

WRTRAX



Write tracks buffer address/enable

This instruction sets the tracks buffer address, length, and enables or disables the microcode tracks processing. The word addressed by E controls the operation of the instruction and has the following format:

0 Enable the microcode tracks processing. When this feature is enabled, the microcode stores the PC of each instruction executed in a circular buffer in physical memory. If this bit is off, the tracks processing is disabled.

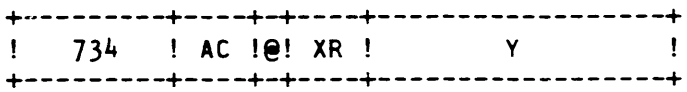
1-17 Two's complement length of the tracks buffer in words. Note that if tracks processing is being enabled, this makes the entire left half of this word be the two's complement length of the buffer.

18-35 Physical page number of the start of the buffer. The microcode will begin storing PCs starting at this physical page and continuing for the length of the buffer. When the buffer limit is reached, the microcode will reset its pointers and start at the beginning of the buffer again.

NOTE

Enabling tracks processing will significantly degrade the speed of the machine. Besides the overhead of one memory write for each instruction executed, the implementation of this feature also causes the IBOX to be flushed at the end of every instruction, thereby completely defeating the pipeline mechanism.

READTB



Read translation buffer entry

This instruction allows the monitor to directly read an MBOX translation buffer entry. The word addressed by E contains the index into the translation buffer and may be thought of as a virtual address. In addition to the normal 30 bit VMA in bits 6-35 of the word, bit 5 is used to specify whether the reference is for exec (bit=0) or user (bit=1) translation. Bits 16-26 are the index into the translation buffer, but bit 16 is complemented if bit 5 is on. To simply read a specified translation buffer location, bit 5 should be zero and bits 16-26 should give the desired index into the translation buffer. Bits 5-15 only need be specified if the read is also to do a valid translation check as indicated by bit 7 returned in the AC (see below). This instruction does not modify the contents of the translation buffer entry (except as the possible result of a translation buffer refill that occurs as the result of the instruction or data fetch). The translation buffer entry is returned in the AC and has the following format:

0 Hardware error. If this bit is on, the translation buffer access caused a hardware error.

1 TB KEEP. If this bit is on, the translation buffer entry has the "keep" bit set.

2 Undefined.

3 TB VALID. If this bit is on, the translation buffer entry contains a valid translation.

4 TB CST UPDATE. If this bit is on, the translation buffer "CST update needed" bit is set.

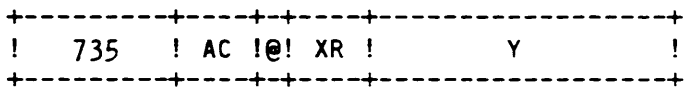
5 TB WRITABLE. If this bit is on, the translation buffer "writable" bit is on.

6 TB MODIFIED. If this bit is on, the translation buffer "modified" bit is on.

7 -VALID TRANSL. If this bit is on, there was no valid translation for the requested address. This bit should normally be ignored since the READTB instruction specifies an index into the translation buffer and not a full address.

8	Returned as zero.
9	TB USER. If this bit is on, the mapping in the entry is for a user page. If this bit is off, the mapping is for an exec page.
10-19	TB DIR<6:15> This field contains the translation buffer directory entry for the mapping. This is bits 6-15 of the VMA for the mapping (bits 16-26 are implicitly specified by the offset in the translation buffer).
20-35	TB PPN<11:26> This field contains bits 11-26 of the physical address for the mapping.

WRITTB



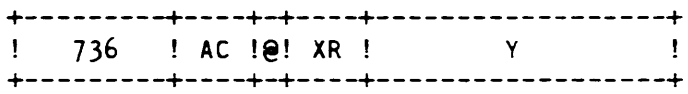
Write translation buffer entry

This instruction allows the monitor to directly write an MBOX translation buffer entry. The word addressed by E contains the index into the translation buffer and may be thought of as a virtual address. In addition to the normal 30 bit VMA in bits 6-35 of the word, bit 5 is used to specify whether the reference is for exec (bit=0) or user (bit=1) translation. Bits 16-26 are the index into the translation buffer, but bit 16 is complemented if bit 5 is on. The data to be written into the translation buffer entry is taken from the AC specified by the instruction and is written into the translation buffer entry specified by bits 5 and 16-26 of the VMA. The format of the data is as follows:

0-2	Ignored.
3	TB VALID. If this bit is on, the translation buffer will contain a valid translation.
4	TB CST UPDATE. If this bit is on, the translation buffer "CST update needed" bit will be set.
5	TB WRITABLE. If this bit is on, the translation buffer "writable" bit will be set.
6	TB MODIFIED. If this bit is on, the translation buffer "modified" bit will be set.
7	Ignored.
8	TB KEEP. If this bit is on, the translation buffer keep bit will be set.
9	TB USER. If this bit is on, the mapping in the entry is for a user page. If this bit is off, the mapping is for an exec page.
10-19	TB DIR<6:15> This field contains the translation buffer directory entry for the mapping. This is bits 6-15 of the VMA for the mapping (bits 16-26 are implicitly specified by the offset in the translation buffer).

20-35 TB PPN<11:26> This field contains bits 11-26 of the physical address for the mapping.

DUMPTB



Dump translation buffer

This instruction allows the monitor to dump the entire MBOX translation buffer into 2048 contiguous physical memory locations. AC contains the physical memory address of the first translation buffer entry to be stored. The address need not be on a page boundary but the location must be contiguous in physical memory. Each entry dumped has the format described for the READTB instruction described above. The indirect, index and Y fields of the instruction are not used and are ignored.

CHAPTER 7

QUEUES AND QUEUE MANIPULATION INSTRUCTIONS

7.1 INTRODUCTION

The KC10 provides instructions to manipulate queues. These instructions are available in EXEC mode only, and are intended to allow sharing of queues among any combination of the following:

1. One or more processes running in the CPU.
2. One or more ports.

7.2 DATA STRUCTURES

7.2.1 The Queues

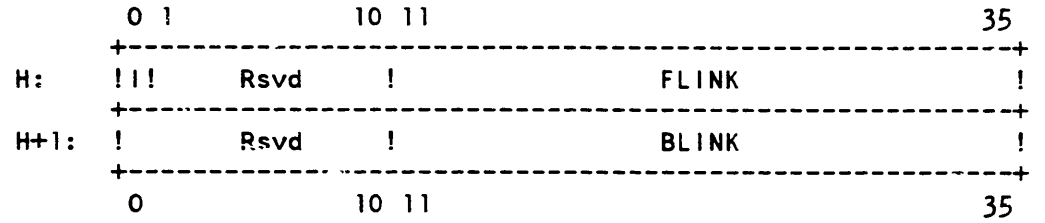
Queues that are manipulated by these instructions must

1. Be doubly-linked.
2. Contain a forward pointer in offset 0 of each entry.
3. Contain a backward pointer in offset 1 of each entry.
4. Be pointed to by a pair of header words.
5. Be referenced by physical addresses.

7.2.2 Formats

A queue header consists of a pair of words. Offset 0 points to the first entry in the queue; offset 1 points to the last entry. If a queue is empty, both header words point to offset 0.

The format of the header words is as follows:



The format of word H is as follows:

0 The secondary queue interlock bit

1-10 Reserved

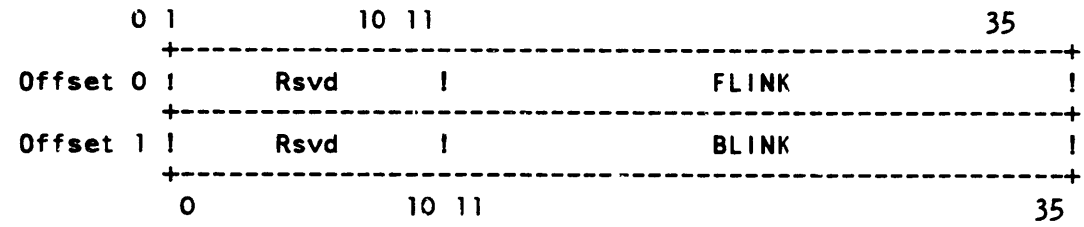
11-35 Physical address of first entry in queue (FLINK)

The format of word H+1 is as follows:

0-10 Reserved

11-35 Physical address of last entry in queue (BLINK)

Each entry contains forward and backward pointers in the following format:



The format of the first link word in a queue entry is as follows:

0-10 Reserved

11-35 Physical address of next entry in queue (FLINK)

The format of the second link word in a queue entry is as follows:

0-10 Reserved

11-35 Physical address of previous entry in queue (BLINK)

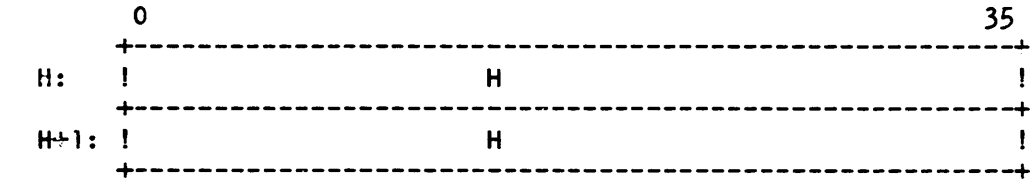
7.3 OPERATIONS

The instructions provide four functions:

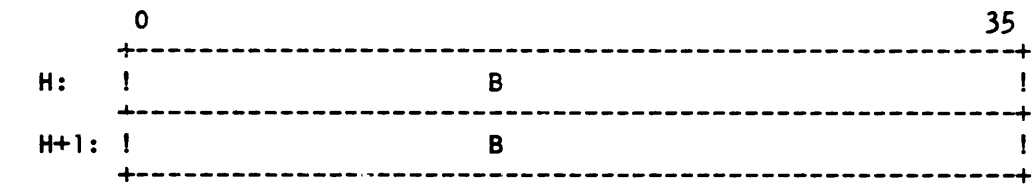
1. Insertion of an entry at the head of a queue.
2. Insertion of an entry at the tail of a queue.
3. Removal of an entry from the head of a queue.
4. Removal of an entry from the tail of a queue.

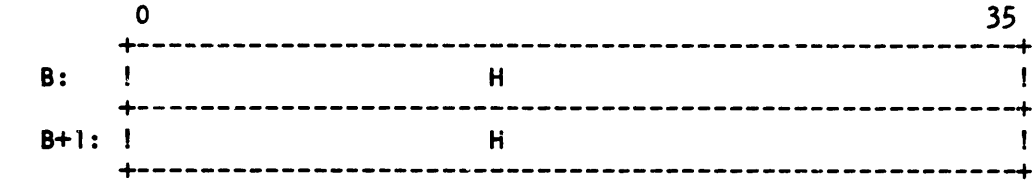
7.3.1 Insertion

An empty queue is specified by its header at address H:

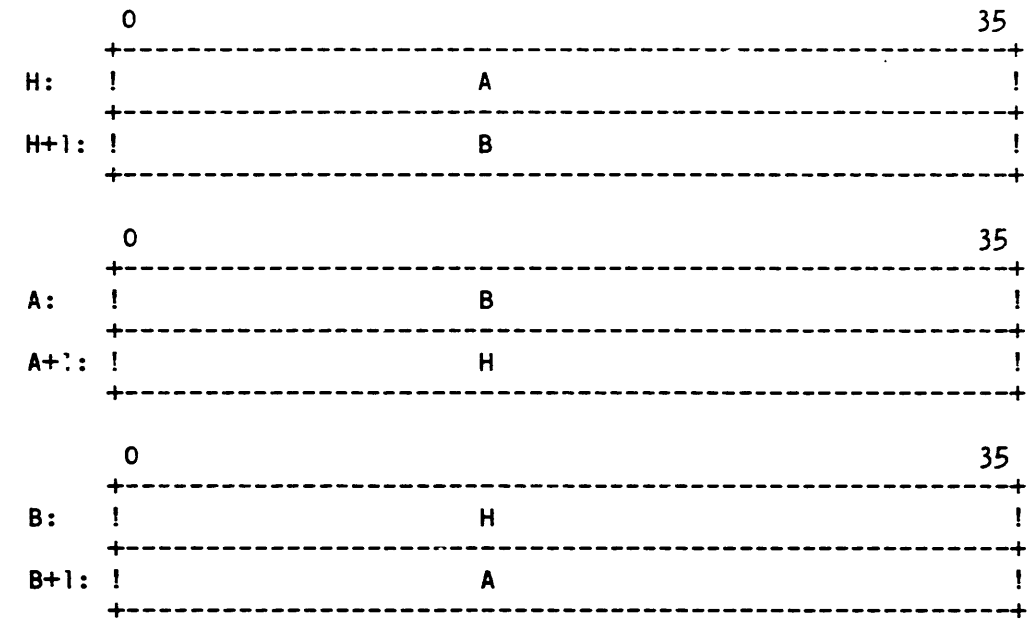


If an entry at address B is inserted into an empty queue (at either the head or tail), the queue is as shown below:

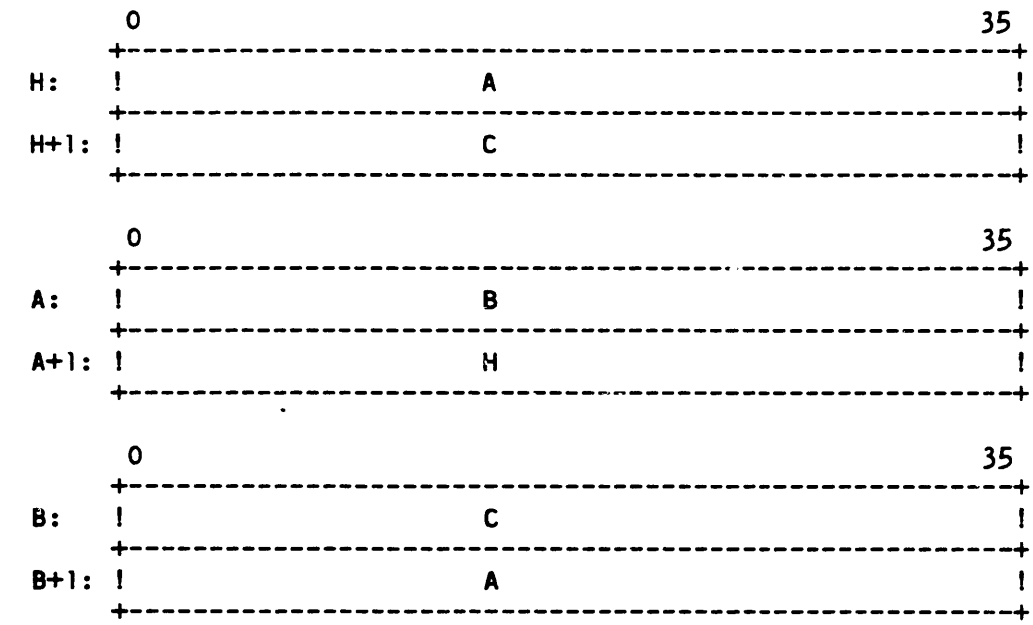


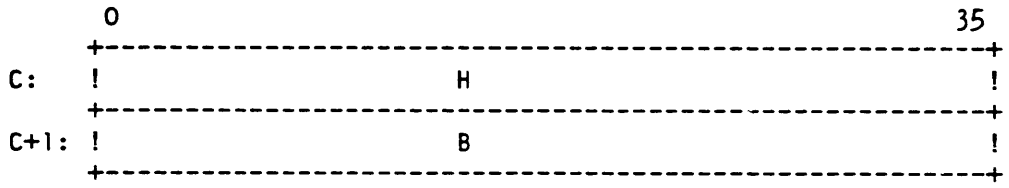


If an entry at address A is inserted at the head of the queue, the queue is show below:



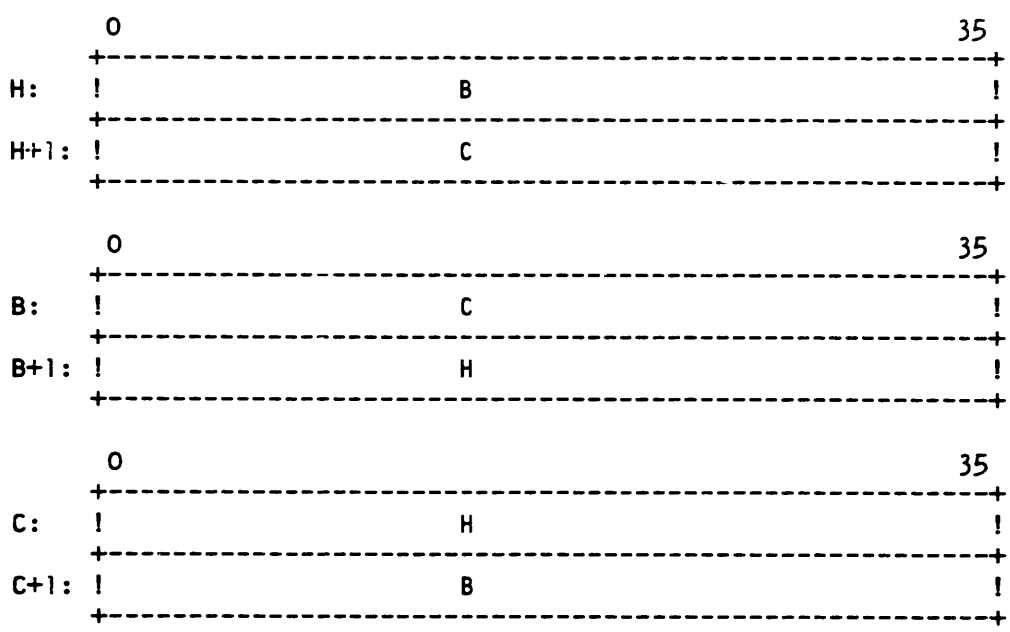
Finally, if an entry at address C is inserted at the tail, the queue appears as follows:





7.3.2 Removal

In the example above with the queue containing entries A, B, and C, the entry at address A can be removed giving:



7.4 INTERLOCKS

Cooperating users of a queue can ensure that no conflicts occur by using only the queue instructions when adding or deleting entries.

When executing a queue instruction, the CPU uses two interlocks. First it uses the MBOX "read-interlock" function to read the queue header. This function sets a hardware interlock that delays any subsequent read-interlock request.

Bit 0 of the queue header provides a secondary interlock. If the bit is off, the queue is available, and the CPU uses the MBOX "write-release" function to set the bit in the header word and release the interlock. At this point, any pending read-interlock finds the bit set in the header.

Having obtained the secondary interlock, the CPU performs the queue manipulation specified by the instruction. It then performs another read-interlock on the header, clears the secondary interlock, and performs a write-release.

Alternatively, if the CPU finds the secondary interlock set, it performs the write-release without changing the header and retries <TBS> times in an attempt to get the secondary interlock. If all retries are unsuccessful, control returns to the user.

The I/O ports manipulate the queues in a similar way. This allows the ports and CPU to cooperate in the use of I/O queues.

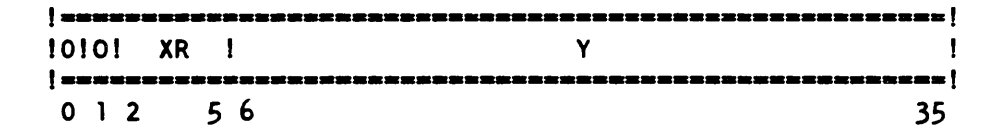
If the queue instruction returns an interlock failure, it may be necessary for the CPU to free the interlock. This action would probably consist of reinitializing the port that has the interlock, cleaning up the queue, and then clearing the secondary interlock bit in the queue header so that the queue is accessible again. It is assumed that there is a direct association between the queue that is interlocked and a particular port.

7.5 THE INSTRUCTIONS

The queue instructions have common characteristics, as follows:

1. For insertions, the AC contains the physical address of word zero of an entry to be inserted. Bits 0-10 must be 0. For removals, the physical address of word zero of the entry that was removed is returned in the AC. If the AC contains a value in the range 0-17, it is interpreted as a physical address, not an AC.

2. E addresses a physical EA-calc word that is evaluated to produce a 25 bit physical address of the queue header. A physical EA-calc word is very similar to a virtual EFIW word and looks as follows:



Bits 2-5 of the physical EA-calc word are the index register address and bits 6-35 are the physical memory address Y. The physical effective address is Y alone if XR is zero. If XR is non-zero, the contents of the index register are added to Y to produce a 25 bit physical effective address. A physical effective address in the range 0-17, inclusive, addresses physical memory locations 0-17, not the ACs. Bits 0 and 1 of the EA-calc word must be zero and the execution of the instruction will generate a page fail if they are not.

3. If the secondary interlock is locked, the instruction returns +1. Otherwise, it returns +2.

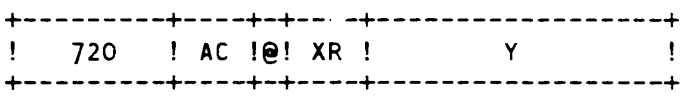
4. If the instruction skips, it may provide further information. For insertion instructions, if the queue is empty before the insertion, the instruction sets bit 0 of the AC provided. For removal, if the queue is empty, the instruction sets bit 0 of the AC provided. Note that the instructions never clear bit 0; software must clear it in order to test for an empty queue.

5. No CST update is performed.

7.6 ERRORS

To be supplied

INSQHI



Insert Entry into Queue at Head, Interlocked

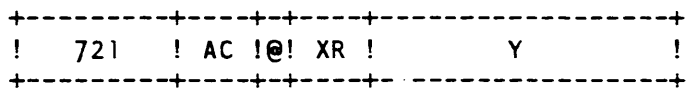
Perform a physical EA-calc using the word addressed by E, then insert the entry specified by the physical address contained in AC into a queue following the header specified by the result of the physical EA-calc. If the entry inserted was the first one in the queue (i.e. C(E) = C(E+1) after insertion), the instruction skips to PC +2 and sets bit 0 in the AC.

If the secondary interlock was unavailable (ie bit 0 of the queue header = 1) the instruction returns to PC+1. The correct way to insert an item at the head of a queue is as follows:

```
INSQHI AC,E
 CALL INTERR          ;interlock error
JUMPL AC,EMPTYQ       ;entry into empty queue
<entry into non-empty queue>
```

See the section above for a discussion of the physical EA-calc algorithm.

INSQTI



Insert Entry in Queue at tail, Interlocked

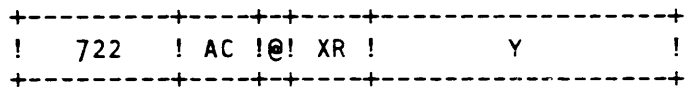
Perform a physical EA-calc using the word addressed by E, then insert the entry specified by the physical address contained in AC into a queue preceding the header specified by the result of the physical EA-calc. If the entry inserted was the first one in the queue (i.e. C(E) = C(E+1) after insertion), the instruction skips to PC +2 and sets Bit 0 in the AC.

If the secondary interlock was unavailable (ie bit 0 of the queue header = 1) the instruction returns to PC+1. The correct way to insert an item at the tail of a queue is as follows:

```
INSQTI AC,E
 CALL INTERR          ;interlock error
JUMPL AC,EMPYQ        ;entry into empty queue
<entry into non-empty queue>
```

See the section above for a discussion of the physical EA-calc algorithm.

REMQHI



Remove Entry from Queue at Head, Interlocked

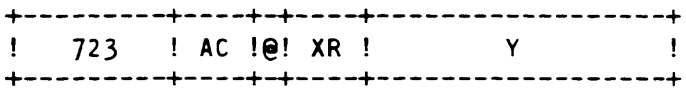
Perform a physical EA-calc using the word addressed by E, then remove the queue entry following the header specified by the result of the physical EA-calc. The 25-bit physical address of the entry removed is place in AC. If there was no entry in the queue (i.e. C(E) = C(E+1) before removal), the instruction skips to PC +2 and sets Bit 0 in the AC. otherwise it skips to PC+2.

If the secondary interlock was unavailable (ie bit 0 of the queue header = 1) the instruction returns to PC+1. The correct way to remove an item from the head of a queue is as follows:

```
REMQHI AC,E
 CALL INTERR          ;interlock error
JUMPL AC,NOENTR       ;no entry found
<entry returned in AC>
```

See the section above for a discussion of the physical EA-calc algorithm.

REMQTI



Remove Entry from Queue at Tail, Interlocked

Perform a physical EA-calc using the word addressed by E, then remove the queue entry preceding the header specified by the result of the physical EA-calc. The 25-bit physica, address of the entry removed is place in AC. If there was no entry in the queue (i.e. C(E) = C(E+1) before removal), the instruction skips to PC +2 and sets Bit 0 in the AC. otherwise it skips to PC+2.

If the secondary interlock was unavailable (ie bit 0 of the queue header = 1) the instruction returns to PC+1. The correct way to remove an item from the tail of a queue is as follows:

```
REMQTI AC,E
 CALL INTERR          ;interlock error
JUMPL AC,NOENTR       ;no entry found
<entry returned in AC>
```

See the section above for a discussion of the physical EA-calc algorithm.

CHAPTER 8

PAGING

8.1 INTRODUCTION

The KL10 implemented both TOPS-10 paging, which supported only one section of virtual address space, and TOPS-20 paging, which supported a maximum of 32 sections of virtual address space. The paging data structures used on the KL10 imposed these limitations.

The KC10 will implement TOPS-10 paging, a TOPS-20 paging KL10 compatible sub-mode that will support a maximum of 32 sections, and a new mode that supports 4096 sections of virtual address space with TOPS-20 paging. This will be done in such a manner as to allow different processes using TOPS-20 paging to be in different paging modes without having to implement a new "mode" bit in the process context variables.

8.2 PAGING HARDWARE AND MICROCODE

The KC10 translation buffer (or page table as it was known on the KL10) is 2K words long, 1 way associative, and has a 1 word block size. In the KL10, translation buffer refills for only TOPS-20 paging were done by the EBOX microcode. In the KC10, all translation buffer refills are done by the EBOX microcode.

The KC10 translation buffer is indexed by virtual address bits 16-26 with the state of bit 16 inverted in user space to separate user and exec entries for the same page. Each translation buffer slot contains virtual address bits 6-15 for the current entry, state bits (described below), and the corresponding physical address bits 11-26 for the current entry. The translation buffer state bits are as follows:

User — A 1 in this bit indicates that this entry describes a user page. A 0 indicates that this entry describes an exec page.

Valid — A 1 in this bit indicates that this entry contains a valid mapping. A 0 in this bit indicates that no valid mapping exists in this translation buffer entry and that an EBOX translation buffer refill is required when a virtual

reference is made.

Modified — A 1 in this bit indicates that the mapping describes a page that has been modified since being brought into memory, i.e., that this page is newer than any backup copy. A 0 in this bit indicates that the mapping describes a page that has not been modified since being brought into memory. A write reference to a page whose translation buffer "modified" bit is 0 will cause an EBOX page fail trap. The EBOX will update the CST entry for that page to set the M bit (bit 35), set this bit in the translation buffer entry, and restart the reference.

Writable — A 1 in this bit indicates that the mapping describes a page that is writable. A 0 in this bit indicates that the mapping describes a page that is write-protected. A write reference to a page whose translation buffer "writable" bit is 0 will cause an EBOX page fail trap and a corresponding page fail trap to the monitor.

Keep — A 1 in this bit indicates that this mapping is not to be invalidated if the translation buffer is cleared with a WRCTX instruction that does not specify all pages (bit 3 in E of WRCTX). It has no effect on translation buffer clears caused by WREBR or CLRPT instructions. A 0 in this bit indicates that there are no restrictions in clearing this entry on a translation buffer clear.

CST update — A 1 in this bit causes an EBOX page fail trap on the next virtual reference to the page described by this entry. The EBOX performs a CST update operation and clears this bit in the entry without clearing the rest of the mapping. A SETCU instruction sets this bit in all entries in the translation buffer and it is cleared by the EBOX for individual entries when the CST update has been performed.

8.3 CACHING OF PAGING INFORMATION

In an attempt to make the virtual-to-physical translation performed by the pager as fast as possible, the KC10 keeps a cache of several levels of information about recent translations. The most obvious example of this caching is the MBOX translation buffer which stores, in hardware, up to 2K translations. In addition to this, the EBOX microcode caches some information about the last few translation buffer refills that it performed in working storage inside the EBOX. The EBOX cache is intended to make translation buffer refills as fast as possible in the case where there is no valid translation in the MBOX.

One aspect of this caching is that the monitor must tell the microcode and hardware when it changes a mapping. In previous machines, this simply meant that the monitor did a CLRPT instruction to clear a

translation for a single virtual page or did a CONO PAG, or DATAO PAG, to clear the entire translation buffer. The same concept holds for the KC10, although the invalidation also effects the EBOX caching.

The CLRPT, WRCTX, and WREBR instructions still clear the MBOX translation buffer entry or entries as appropriate but they also clear all or part of the EBOX information. This process should be transparent to the monitor programmer; if the invalidation would work on a KL10, it will work on a KC10 since the same algorithms apply.

There is one case, however, where the KC10 is different. In order to optimize the processing of KL compatible paging vs. KC paging, the EBOX microcode caches the first super section pointers from EPT and UPT locations 520. These two locations are read and cached anytime the monitor does a WREBR irstruction or a WRCTX that changes the UBR, and the information is NOT cleared on a CLRPT. The ONLY way to flush this information is with another WREBR or WRCTX.

8.4 TOPS-20 PAGING

8.4.1 Pager Data Structure

The KL10's implementation of extended sections was to allow a maximum of 32 section pointers to be placed in EPT/UPT locations 540-577. A single page full of section pointers can only reference 512 sections. 8 pages of section pointers will be required to address 4096 sections. Since we are going to create some new data items and structure, let us define some terms:

1. A page containing section pointers will be called a "Section Table" or ST. The pointer types found herein are identical to those already found in EPT/UPT locs 540-577 on a KL10.

2. A page containing map pointers will be called a page map.

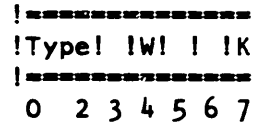
3. VMA<6:8> will be called the "Super Section Number" and will be used to determine which of the 8 Section Tables to look in.

4. EPT/UPT locations 520-527 will be a "Super Section Table" or SST, and will be indexed by VMA<6:8>.

5. The Super Section Table will contain new pointer types called "Super Section Pointers" defined below.

8.4.2 Pointers

The microcode evaluates three kinds of pointers: super section pointers, section pointers, and map pointers. These are used in super section tables, section tables, and page maps, respectively. There are 5 types of pointers distinguished by a type code in bits 0-2 of the pointer; of these, three are access pointers that allow access to the given super section, section, or page and are identical in the format of the left 8 bits. This format is as follows:



Bits 3, 5, and 6 are ignored by the microcode and may be used by the software.

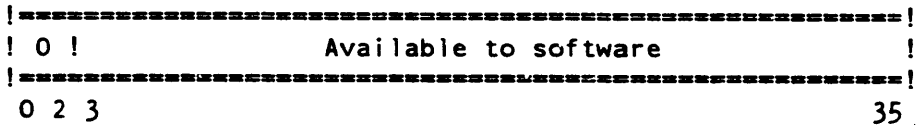
Every access pointer of this type must have "use" bits for the super section, section, or page it represents. These bits, W and K, indicate whether the super section, section, or page is writable, or kept. Throughout the evaluation procedure the microcode effectively ands these bits from one pointer to the next, so the final result requires that the given characteristics be specified at every step.

In other words, if W is 1 in the final pointer for the mapping, the page is writable provided the super section and the section were also specified as writable by the original super section and section pointers, and "writable" has been specified by every other pointer encountered along the way.

Note that the W bit is also ANDed with the W bit in the CST entry for the final data paga to determine the state of the translation buffer W state bit. Thir final operation is not done if the CST base address is zero.

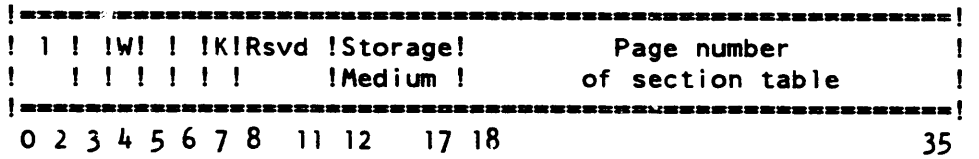
8.4.2.1 Super Section Pointers - Entries in the Super Section Table in EPT/UPT locations 520-527 are of the following five types. All other types are reserved and will cause a page fail if the microcode encounters them on a refill.

No access



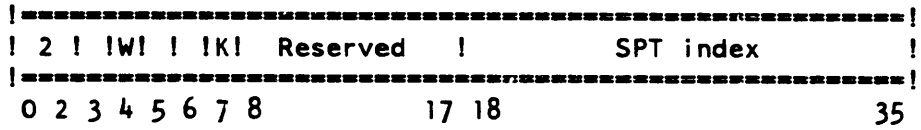
The super section is inaccessible.

Immediate



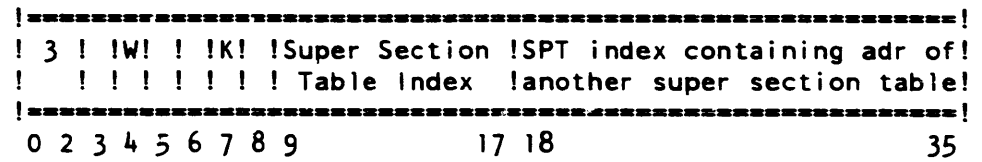
If bits 12-17 are zero, the section table is in the page specified by bits 18-35. Otherwise, the page is not in memory.

Shared



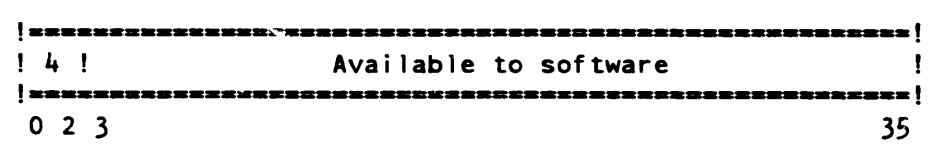
The page address of the section table is in the SPT at the offset specified by bits 18-35

Indirect



In the SPT offset specified by bits 18-35 is the page address of a secondary super section table. The next super section pointer to be evaluated is in that table at the offset specified by bits 9-17.

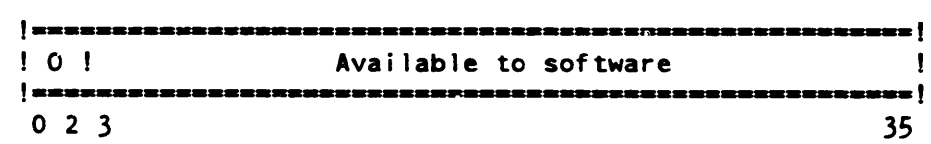
KL compatible



This type of pointer may ONLY appear in EPT/UPT offset 520 and indicates that KL compatible paging is to be used. If VMA<6:12> is zero, use VMA<13:17> as an index into the KL compatible section table starting at EPT/UPT offset 540 and perform the pointer evaluation exactly as a KL10 would. If VMA<6:12> is non-zero or if this type of pointer appears in a super section table entry other than that at EPT/UPT offset 520, a page fail trap will occur. See the section on page fail conditions for the page fail codes.

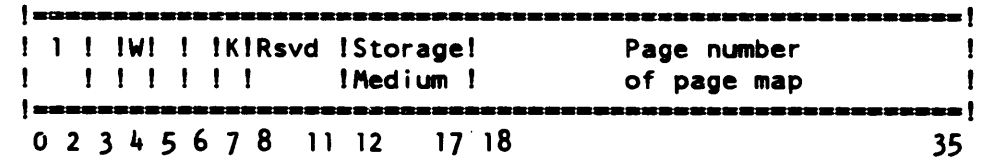
8.4.2.2 Section Pointers - Entries in a section table are of the following four types. All other types are reserved and will cause a page fail if the microcode encounters them on a refill.

No access



The section is inaccessible.

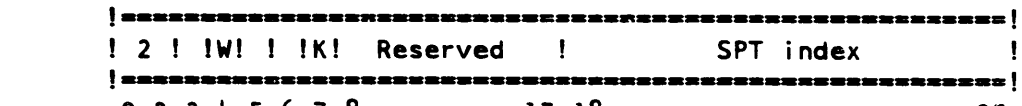
Immediate



If bits 12-17 are zero, the page map is in the page specified by bits

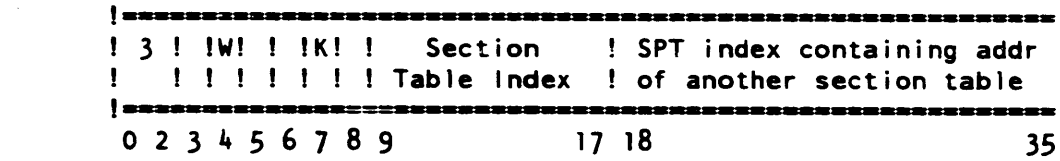
18-35. Otherwise, the page is not in memory.

Shared



The page address of the page map is in the SPT at the offset specified by bits 18-35

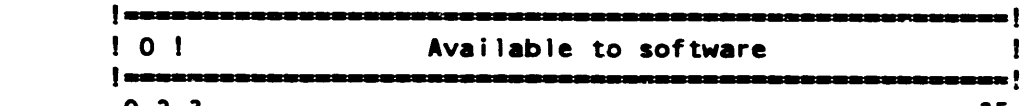
Indirect



In the SPT offset specified by bits 18-35 is the page address of a secondary section table. The next section pointer to be evaluated is in that table at the offset specified by bits 9-17.

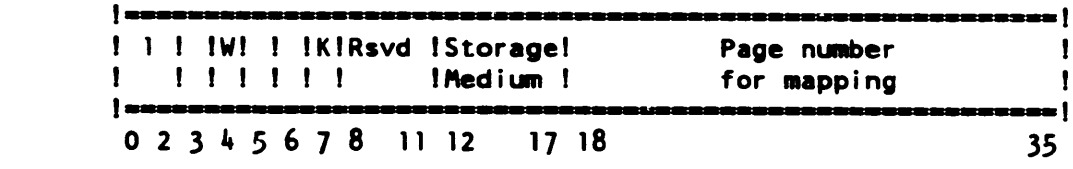
8.4.2.3 Map Pointers - Entries in a page map are of these four types. All other types are reserved and will cause a page fail if the microcode encounters them on a refill.

No access



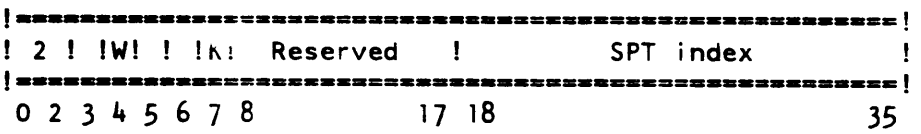
The page is inaccessible.

Immediate



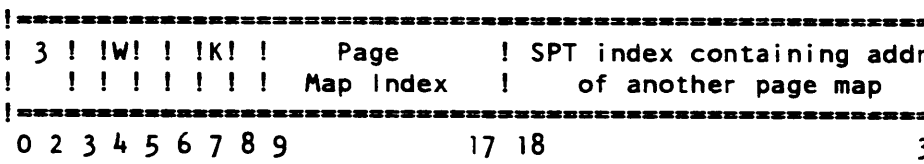
If bits 12-17 are zero, the physical page specified by bits 18-35 corresponds to the referenced virtual page. Otherwise, the page is not in memory.

Shared



The page address for the mapping for the referenced virtual page is in the SPT at the offset specified by bits 18-35.

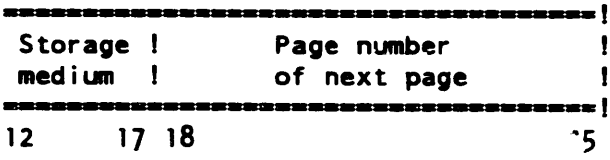
Indirect



In the SPT offset specified by bits 18-35 is the page address of a secondary page map. The next map pointer to be evaluated is in that map at the offset specified by bits 9-17.

8.4.3 Page Address Words

The translation buffer refill process causes the microcode to follow pointers in memory to finally determine the physical page number of the data page that should be mapped by the virtual page that caused the page fault. In order to do this, the microcode must evaluate 3 different kinds of pointer levels, super section, section, and page pointers. At each level, the microcode must encounter a "page address word" that gives the page number of the page for the next level. For the page pointer evaluation, the page address word actually gives the page number of the final data page. This page address word has the following format:

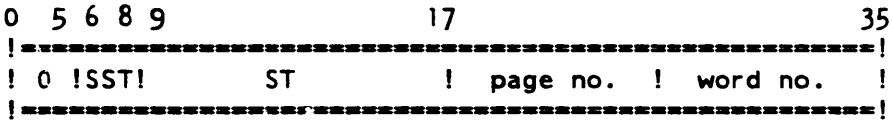


If bits 12-17 are zero, the storage medium is memory, i.e., bits 18-35 supply the number of a page that is in memory. If bits 12-17 are nonzero, the page exists but is stored on some other medium and the microcode traps to the monitor to bring the page into memory. The page address word may be extracted from bits 12-35 of an immediate pointer, or from bits 12-35 of the SPT for share or indirect pointers. For indirect pointers, the microcode will actually encounter more than one page address word.

8.4.4 Conversion Of Virtual To Physical Addresses

An address is converted to a physical page number as follows:

VMA<6:8> is used to index into the Super Section Table. One of the 5 pointer types (Super Section Pointers) can occur here: No Access, immediate, shared, indirect, or KL compatible. Immediate, shared, or indirect pointers yield the physical page number of a Section Table page. VMA<9:17> is used to index into the Section Table to obtain a Section Pointer. Address translation then proceeds as on the KL10 after the section pointer fetch. (See DECsystem10/20 Processor Reference Manual, AA-H391A-TK for a complete description). The VMA can be thought of as follows:



8.4.5 Page Refill

8.4.5.1 CST Updates - The EBOX microcode performs an operation called a "CST update" at several points during the processing of a page fault detected by the MBOX translation buffer. The operations performed by a CST update are as follows:

1. If the CST base address is zero, skip the rest of the steps.
2. Read the CST entry for the physical page in question from the word addressed by the sum of the CST base register and the physical page number.
3. If the age in the entry (bits 0-5) is zero, start an age-too-small page fail trap to the monitor and skip the rest of the steps.
4. AND the entry with the contents of the CST mask register (set by the WRCSTM instruction).
5. OR the masked entry with the contents of the process use register (set by the WRPUR instruction).
6. Set the modified (M) bit in the entry, if necessary.
7. Write the entry back into the CST in memory, if necessary.

The cases under which a CST update is performed are as follows:

1. A page fault caused by a write reference to a page that is writable but not yet modified. This case sets the modified bit in the entry and writes it back into the CST.

2. A page fault caused by the CST-update-needed bit set in the translation buffer entry for the referenced page. This case writes the entry back into the CST.

3. A pointer trace evaluates the address of a new physical page. This case performs only steps 1-3 as described above for the intermediate pages in the pointer trace. For the final data page that is evaluated by the pointer trace, the full update is performed and the updated entry is written back into the CST.

8.4.5.2 CST Entry Format - The CST is a table indexed by physical page number and checked whenever a new memory page is referenced by the microcode. In addition, it is updated for the final data page obtained in a page fail pointer trace and for writable-but-not-yet-modified and CST-update-needed EBOX page fails. The CST format is as follows:

The monitor keeps a state code in bits 0-8 of the entry; within the code, bits 0-5 represent the page age, which must be non-zero for the page to be usable. A zero page age results in an age-too-small page fail trap to the monitor. The "W" bit is the master write-enable bit for the physical page and is ANDed with the "W" bits in the page pointers when a data page address is written into the translation buffer. The "M" bit indicates that the page has been modified since being brought into memory and is set by the microcode on a writable-but-not-yet-modified EBOX page fail trap.

8.4.5.3 CST Mask Register Format - The CST mask register is ANDed with the CST entry during the CST update process. It should contain a one in every bit position that must be preserved during the update procedure and a zero in every bit position that must be cleared during the update. Therefore, the CST mask register should always contain ones in bits 34 and 35 (the W and M bits) and zeros in bits 0-5 (the page age).

8.4.5.4 Process Use Register Format - The Process Use Register is ORed with the masked CST entry during the CST update process. It should contain a zero in every bit position that must be preserved during the update procedure and a one in every bit position that should be set. Therefore, the Process Use Register should always contain zeros in bits 34 and 35 (the W and M bits) and the new page age in bits 0-5.

8.4.5.5 Translation Buffer State Bits - A refill sets the translation buffer state bits as a function of the logical and of all the pointer use bits that it evaluated in the pointer chase. The relationship is as follows:

State bit	Set if the following condition is met
User	1 if this mapping is for user space.
Valid	Always set to a 1.
Modified	1 if the physical page corresponding to this mapping has already been modified according to the CST entry for that page.
Writable	1 if the logical and of the W pointer use bits of all pointers evaluated was a 1.
Keep	1 if the logical and of the K pointer use bits of all pointers evaluated was a 1.
CST update	Always set to a 0

8.4.5.6 Write References - When a virtual write reference is made to the MBOX, the result is a function of the translation buffer entry corresponding to the virtual address specified by the EBOX. Write references are particularly interesting because they can succeed or fail based on the state of the writable and modified bits in the translation buffer. The relationship between write references and the four possible combinations of the writable and modified bits is as follows:

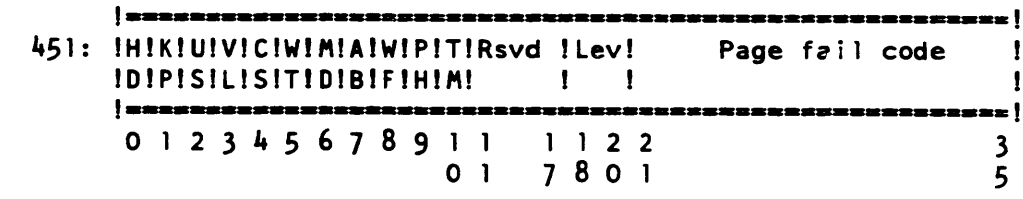
Writable	Modified	Effect
0	0	The page is not writable. A write failure page fail trap will be given to the monitor.

0	1	The page is not writable. A write failure page fail trap will be given to the monitor.
1	0	The page is writable but not yet modified. The EBOX microcode will get a page fail trap, update the CST entry for the page to set the M bit, set the modified bit in the translation buffer, and retry the reference. Note that the EBOX microcode can give an illegal age page fail trap to the monitor if the CST age for the referenced page is illegal.
1	1	The write will succeed.

8.4.6 Page Fail Conditions And Formats

A page failure occurs when the pager is unable to make a desired memory reference, the EBOX detects an illegal condition while executing an instruction (e.g., incorrectly formated indirect word, illegal one-word-global byte pointer, etc.), or the MBOX detects a hardware failure while processing a memory request. When such a condition occurs, the EBOX microcode stores information about the page fail in UPT locations 451-455, stores the current flag-PC double word in UPT locations 456-457 and loads the new flags, CAB, and PC from the new flag-PC double word in UPT locations 460-461. The format of each of these words is described below.

UPT location 451 contains the page fail word that describes the condition that caused the page fail. The format is as follows:



The definition of each field is as follows:

0	This page fail was caused by a "hard" error. This does not necessarily mean that a hardware failure occurred. If this bit is set, bits 1-4 contain a code that describes the failure. The EBOX microcode copies the code to bits 27-35 and the valid codes are described below.
1	This bit gives the state of the translation buffer "keep" state bit for a page fail that resulted from a virtual translation failure.

2 This bit is returned as a 1 if the reference was to user space. If the reference was to exec space, this bit is returned as a 0.

3 This bit gives the state of the translation buffer "valid" state bit for a page fail that resulted from a virtual translation failure.

4 This bit gives the state of the translation buffer "CST update needed" state bit for a page fail that resulted from a virtual translation failure.

5 This bit gives the state of the translation buffer "writable" state bit for a page fail that resulted from a virtual translation failure.

6 This bit gives the state of the translation buffer "modified" state bit for a page fail that resulted from a virtual translation failure.

7 If this bit is a 1, the memory reference caused an address break match.

8 If this bit is a 1, the page fail was caused by a reference that write-failed because of the state of the translation buffer writable and modified state bits. Such a reference may either be a write or a write test. This bit is valid only for a page fail that resulted from a virtual reference.

9 If this bit is a 1, the memory request was a physical reference. If the bit is a 0, the memory request was a virtual reference.

10 If this bit is a 1, there was no valid translation buffer mapping for the virtual address in the request.

11-17 Reserved

18-20 This field gives the level at which this page fail was detected. The level is primarily used to tell the monitor where a translation buffer refill pointer trace stopped and is used in conjunction with the additional data words described below. This field can contain one of four values as follows:

0 This page fault was not the result of a pointer trace, or the page fail condition was detected before the first pointer was fetched.

1 This page fault was detected while processing a super section pointer.

2 This page fault was detected while processing a section pointer.

3 This page fault was detected while processing a page pointer.

21-35 This field gives a code that describes the cause of the page fail. The monitor should never have to look at anything other than bits 0 (hard), 2 (user), 9 (physical reference), 18-20 (level), and this code to determine the exact cause of the page fail. The rest of the bits in this word are returned only as additional information to be used to debug problems. There are two types of codes that are returned in this field, depending on the state of bit 0. If bit 0 is a zero, the page fail and code are the result of one of the following conditions:

1. There was no valid translation for the reference address.

2. A write reference failed because the page wasn't writable.

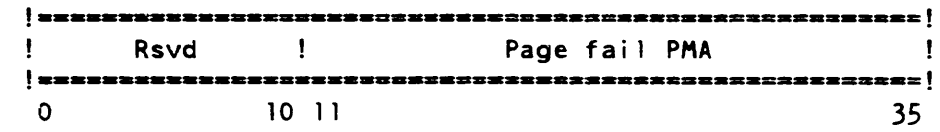
3. An address break occurred.

4. The EBOX detected an illegal condition while executing an instruction.

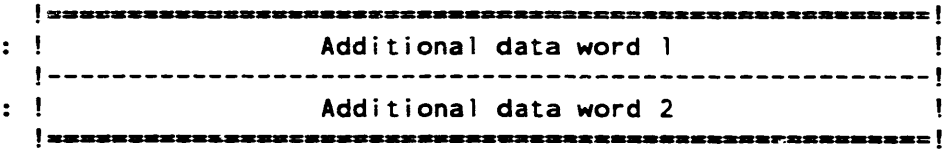
If bit 0 is a one, the page fail and code are the result of a "hard" error. Each case is described separately in the section on page fail codes.

UPT location 452 contains the reference address (if any) for the request that page failed. This address is the virtual memory address for virtual requests and the physical memory address for physical requests. It is only valid for those page fail conditions that resulted from a virtual reference. The table at the end of this section describes under which page fail conditions it is valid.

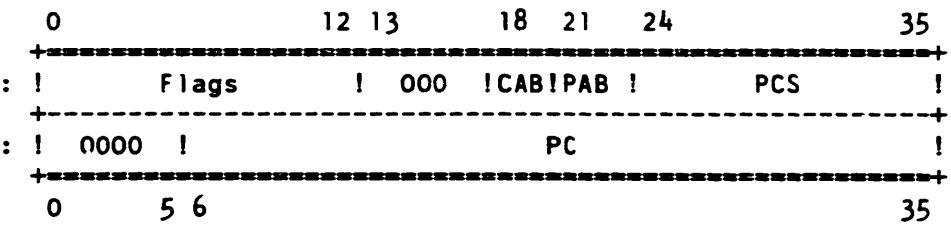
UPT location 453 contains the physical memory address (if any) for the request that page failed. It is only vaild for those page fail conditions that have a valid PMA. The table at the end of this section describes under which page fail conditions it is vaild.



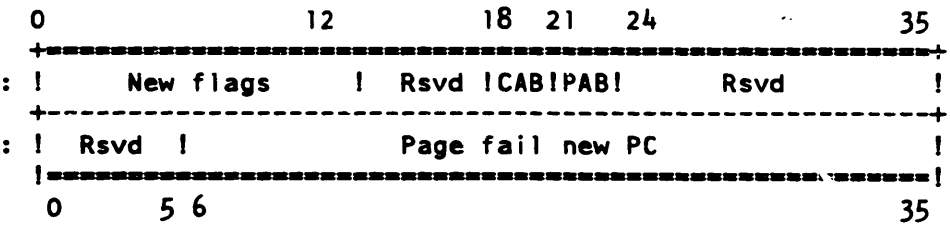
UPT locations 454 and 455 contain additional data that is different for each type of page fail. The contents of these words are given for each page fail at the end of this section. The format of these words is as follows:



UPT locations 456-457 contain the flags, CAB, PAB, PCS, and PC at the time of the page fail in the following format:



UPT locations 460-461 are setup by the monitor and contain the flags, CAB, PAB, and new PC to be loaded when a page fail occurs. The words are in the following format:



8.4.6.1 Tops-20 Page Fail Codes And Additional Data - This section defines the page fail codes that may appear in bits 21-35 of the page fail word and the additional data words returned for each code. For each code below, "RAD", "PMA", "AD1", and "AD2" represent the data returned in words 452-455 of the UPT.

Caution

The page fail codes described below are generated by the EBOX microcode and can be easily changed. These page fail codes are a first-pass attempt at assigning values. They may very well change as we add or delete codes. It is strongly suggested that you not make assumptions about the numeric value of any particular code.

If bit 0 is off in the page fail word (indicating that this page fail is not the result of a "hard" error), the codes that may appear in bits 21-35 of the page fail word are as follows:

1 Write failure - A write reference was attempted to a write-protected page (W bit off in the translation buffer).

RAD Reference address that caused the page fail.

PMA Physical address corresponding to the reference address.

AD1 Undefined.

AD2 Undefined

2 Illegal age - An illegal CST age was detected for a page during the processing of one of the following page fails:

1. CST update needed.

2. Write reference to a writable but not yet modified page.

RAD Reference address that caused the page fail.

PMA Physical address corresponding to the reference address

AD1 Undefined.

AD2 Undefined.

3 Address break - An address break occurred.

RAD Reference address that caused the page fail.

PMA Undefined.

AD1 Undefined.

AD2 Undefined.

4 Illegal super section pointer 0 - A pointer with type 5, 6, or 7 was found in super section table offset 0.

RAD Reference address that caused the page fail.

PMA Undefined.

AD1 Undefined.

AD2 The illegal super section pointer.

5 Section greater than 37 - In KL compatible mode, a virtual reference was made to a section greater than 37.

RAD Reference address that caused the page fail.

PMA Undefined.

AD1 Undefined.

AD2 Super section pointer.

6 Illegal pointer - A pointer with type 4, 5, 6, or 7 was found in the super section table, section table, or page table.

RAD Reference address that caused the page fail.

PMA Undefined.

AD1 Source of last word processed (see below).

AD2 The illegal pointer.

7 No access pointer - A no-access pointer was discovered during a pointer trace.

RAD Reference address that caused the page fail.

PMA Undefined.

AD1 Source of last word processed (see below).

AD2 The no-access pointer

10 Page not in core - A page-address word was discovered whose storage medium field (bits 12-17) was non-zero.

RAD Reference address that caused the page fail.

PMA Undefined.

AD1 Source of last word processed (see below).

AD2 Last pointer processed.

11 Illegal age - An illegal CST age was detected for a page during a pointer trace.

RAD Reference address that caused the page fail.

PMA Undefined.

AD1 Source of last word processed (see below).

AD2 Last pointer processed

12 Must-be-zero bits non-zero - The microcode discovered bits that were declared "must be zero" to be non-zero.

RAD Address of word containing the MBZ bits.

PMA Undefined.

AD1 Undefined.

AD2 Undefined.

13 Illegal indirect - An extended effective address calculation has encountered an indirect word with 11 (binary) in bits 0 and 1.

RAD Address of word containing the illegal indirect.

PMA Undefined.

AD1 The illegal indirect word.

AD2 Undefined.

14 Illegal PXCT - A PXCTed instruction that stores into the ACs was executed with CAB = PAB.

RAD Undefined.

PMA Undefined.

AD1 Undefined.

AD2 Undefined.

15 Illegal physical effective address word - A physical effective address word was discovered with a 1 in bit 0 or 1.

RAD Address of illegal physical effective address word.

PMA Undefined.

AD1 The illegal physical effective address word.

AD2 Undefined.

16 Illegal one-word-global byte pointer - A one-word-global byte pointer was discovered with a code of 77 (octal)

RAD Address of the illegal one-word-global byte pointer.

PMA Undefined.

AD1 Undefined.

AD2 The illegal one-word-global byte pointer.

17 Illegal interrupt vector - An illegal interrupt vector (all zeros) was discovered (I/O page fail only).

RAD Address of the illegal interrupt vector.

PMA Undefined.

AD1 Undefined.

AD2 Undefined.

If bit 0 is on in the page fail word (indicating that this page fail is the result of a "hard" error), the codes that may appear in bits 21-35 of the page fail word are as follows:

To be supplied

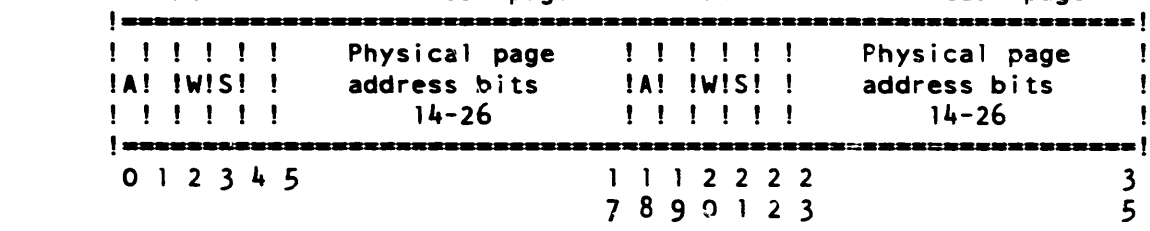
8.4.6.1.1 Additional Data Words For A Pointer Trace - When the EBOX microcode detects a page fail condition during a pointer trace, it stores the source of the last word processed in additional data word 1 (454) and the last pointer fetched in additional data word 2 (455). Additional data word 2 is simply the last pointer processed by the microcode and may be a super section, section, or page pointer. Additional data word 1 specifies the source of the last word processed and may have one of the following forms:

0,,0	If the page fail code is "illegal super section 0 pointer", this word indicates that the pointer trace failed immediately after initialization. If the page fail code is anything else, it is really the following case.
0,,offset	The last word examined was fetched from SPT+offset.
-1,,offset	The last word examined was fetched from UPT+offset or EPT+offset. The user reference bit in the page fail word determines which.
page,,offset	The last word examined was fetched from physical page "page", offset "offset".

8.5 TOPS-10 PAGING

8.5.1 Process Table Mapping Formats

The KC10 process table mapping format is similar to that used on the KL10 and looks as follows:



Each word contains page use bits and the physical page numbers for the even and odd numbered virtual pages corresponding to the map location that holds the word. The page use bits are as follows:

Bit	Meaning of a 1 in the bit
A	Access allowed
W	Writable (not write-protected)
S	Software (not interpreted by the hardware)

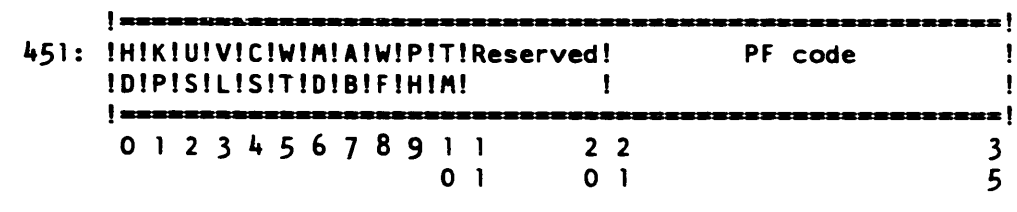
A refill sets the translation buffer state bits as a function of the page use bits as follows:

State bit	Set if the following condition is met
User	The mapping is for user space
Valid	Always set to a 1
Modified	Always set to a 1
Writable	The W page state bit is a 1
Keep	Always set to a 0
CST update	Always set to a 0

8.5.2 Page Fail Conditions And Formats

A page failure occurs when the pager is unable to make a desired memory reference, the EBOX microcode detects an illegal condition while executing an instruction (e.g., illegal one-word-global byte pointer), or the MBOX detects a hardware failure while processing a memory request. When such a condition occurs, the EBOX microcode stores information about the page fail in UPT locations 451-455, stores the current flag-PC double word in UPT locations 456-457 and loads the new flags, CAB, and PC from the new flag-PC double word in UPT locations 460-461. The format of each of these words is described below.

UPT location 451 contains the page fail word that describes the condition that caused the page fail. The format is as follows:



The definition of each field is as follows:

0 This page fail was caused by a "hard" error. This does not necessarily mean that a hardware failure occurred. If this bit is set, bits 1-4 contain a code that describes the failure. The EBOX microcode copies the code to bits 27-35 and the valid codes are described below.

1 This bit gives the state of the translation buffer "keep" state bit for a page fail that resulted from a virtual translation failure.

2 This bit is returned as a 1 if the reference was to user space. If the reference was to exec space, this bit is returned as a 0.

3 This bit gives the state of the translation buffer "valid" state bit for a page fail that resulted from a virtual translation failure.

4 This bit gives the state of the translation buffer "CST update needed" state bit for a page fail that resulted from a virtual translation failure.

5 This bit gives the state of the translation buffer "writable" state bit for a page fail that resulted from a virtual translation failure.

6 This bit gives the state of the translation buffer "modified" state bit for a page fail that resulted from a virtual translation failure.

7 If this bit is a 1, the memory reference caused an address break match.

8 If this bit is a 1, the page fail was caused by a reference that write-failed because of the state of the translation buffer writable and modified state bits. Such a reference may either be a write or a write test. This bit is valid only for a page fail that resulted from a virtual reference.

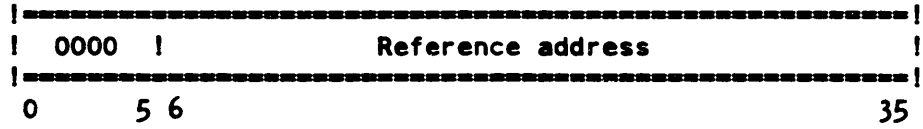
9 If this bit is a 1, the memory request was a physical reference. If the bit is a 0, the memory request was a virtual reference.

10 If this bit is a 1, there was no valid translation buffer mapping for the virtual address in the request.

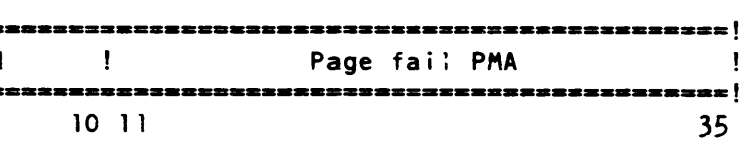
11-20 Reserved

21-35 This field gives a code that describes the cause of the page fail. The monitor should never have to look at anything other than bits 0 (hard), 2 (user), 9 (physical reference) and this code to determine the exact cause of the page fail. The rest of the bits in this word are returned only as additional information to be used to debug problems. There are two types of codes that are returned in this field, depending on the state of bit 0. If bit 0 is a zero, the page fail and code are the result of a virtual reference that failed to generate a valid physical mapping (this also includes write references to write-protected pages, etc.), or a physical reference that caused an address break. If bit 0 is a one, the page fail and code are the result of a "hard" error. Each case is described separately in the section on page fail codes.

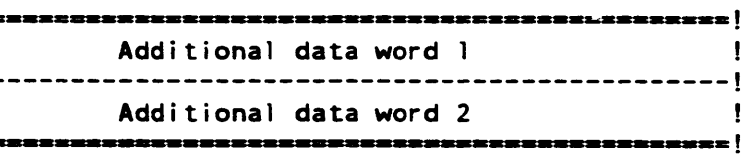
UPT location 452 contains the reference address (if any) for the request that page failed. This address is the virtual memory address for virtual requests and the physical memory address for physical requests. It is only valid for those page fail conditions that resulted from a virtual reference. The table at the end of this section describes under which page fail conditions it is valid.



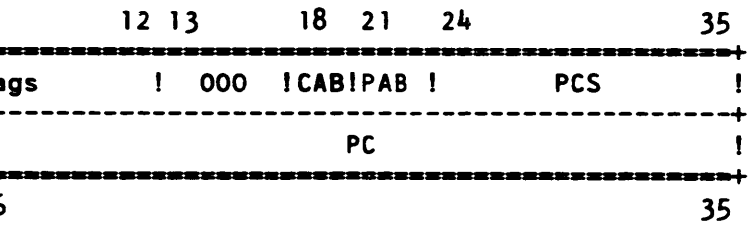
UPT location 453 contains the physical memory address (if any) for the request that page failed. It is only vaild for those page fail conditions that have a valid PMA. The table at the end of this section describes under which page fail conditions it is vaild.



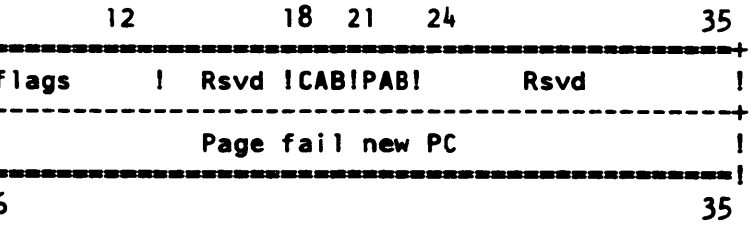
UPT locations 454 and 455 contain additional data that is different for each type of page fail. The contents of these words are given for each page fail at the end of this section. The format of these words is as follows:



UPT locations 456-457 contain the flags, CAB, PAB, PCS, and PC at the time of the page fail in the following format:



UPT locations 460-461 are setup by the monitor and contain the flags, CAB, PAB, and new PC to be loaded when a page fail occurs. The words are in the following format:



8.5.2.1 Tops-10 Page Fail Codes And Additional Data - This section defines the page fail codes that may appear in bits 21-35 of the page fail word and the additional data words returned for each code. For each code below, "RAD", "PMA", "AD1", and "AD2" represent the data returned in words 452-455 of the UPT.

Caution

The page fail codes described below are generated by the EBOX microcode and can be easily changed. These page fail codes are a first-pass attempt at assigning values. They may very well change as we add or delete codes. It is strongly suggested that you do not make assumptions about the numeric value of any particular code.

If bit 0 is off in the page fail word (indicating that this page fail is not the result of a "hard" error), the codes that may appear in bits 21-35 of the page fail word are as follows:

To be supplied

If bit 0 is on in the page fail word (indicating that this page fail is the result of a "hard" error), the codes that may appear in bits 21-35 of the page fail word are as follows:

To be supplied

CHAPTER 9

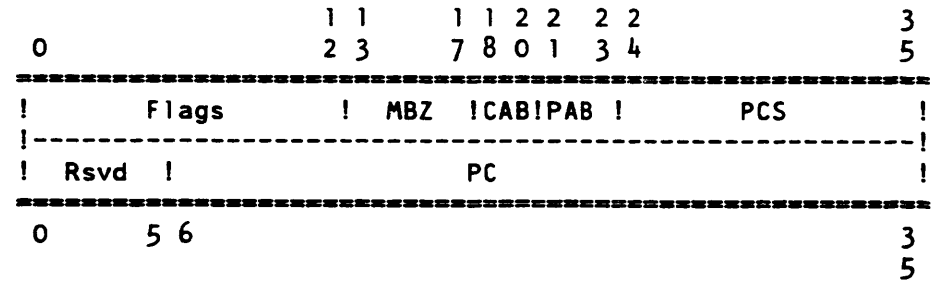
PROCESS CONTEXT VARIABLES

9.1 INTRODUCTION

In order to take advantage of the full 4096 section virtual address space implemented by the KC10 processor, the flag-PC double word format has been changed to allow for a larger section number. In addition, the PAB and CAB fields have been added.

9.1.1 New Flag-PC Double Word

The format of the double word is as follows:



Where:

Flags — PC flags. The action of these flags is the same as for the KL10, unless stated otherwise. The flags are as follows:

0 Overflow.

1 Carry 0.

2 Carry 1.

3 Floating Overflow.

4 First Part Done. This PC flag is used by the microcode as necessary to restart a multi-part instruction. It does not necessarily act the same as any previous machine and the use may change at any time without notice. The monitor should save and restore this flag when changing contexts. The user should never touch it.

5 User.

6 User In-out/Previous Context User.

7 Unused by the KC10 hardware and microcode. On previous machines, this was the Public flag. The KC10 always stores it as zero, and ignores any attempt to set it.

8 Address Failure Inhibit.

9 Trap 2.

10 Trap 1.

11 Floating Underflow.

12 No Divide.

MBZ Must be zero

CAB Current AC Block Number (0-7)

PAB Previous Context AC Block Number (0-7)

PCS Previous Context Section Number

PC PC of the program

1. In kernel mode (XPCW/SFM), or when stored on a page fail or MUUO, all of the above fields will be stored as defined. In kernel mode, XJRSTF and XJEN will restore all fields.

2. In user mode, PCS, PAB, and CAB will always be stored as 0. An XJRSTF in user mode will treat these fields as it does the user mode and user I/O flag now (i.e. ignore them).

9.1.2 Context Changing

Returning to a previous context may be done with an XJRSTF or XJEN instruction which restores the context variables stored in the previously saved PC double word.

Entering a new context will be done as follows: All of the "previous" context variables in the old PC flag word will be set to their corresponding values in the "current" context. If the "current" context is not user-mode, then set the "previous" context from the new PC flag word. The following operations are defined as entering a new context:

1. Monitor call (MUUO).

2. Page fail trap.

3. Priority interrupt initiation.

4. I/O page fail trap.

Each of these operations will store a PC double-word containing the "current" context variables and then load a new PC double-word to set new values for those variables not set automatically. See the chapter on Special System Pagers for a description of the changes to the EPT and UPT.

The following chart summarizes what variables are saved, and what new values are set. It includes for comparison what is currently implemented on the KL10 processor.

Key:

Store — Save in appropriate block (old)

Load — Set from appropriate block (new)

Set — Set "previous" to old "current"

* — In process context word

** — Ucode sets PCS; XPCW stores flags, PC, PCS, and loads flags and PC

		Flags	PC	CAB	PAB	PCS/PCU
XPCW	KL	Store	Store	No	No	Store
		Load	Load	No	No	No
	KC	Store	Store	Store	Store	Store
		Load	Load	Load	No	No
Inter-rupt	KL **	Store	Store	No	No	Store
		Load	Load	No	No	Set (PCS)
	KC	Store	Store	Store	Store	Store
		Load	Load	Load	No	No
MUUO	KL	Store	Store	* Store	* Store	Store
		Clear	Load	No	No	Set (PCS)
	KC	Store	Store	Store	Store	Store
		Load	Load	Load	Load	Set
Page Fail	KL	Store	Store	No	No	Store
		Clear	Load	No	No	Set (PCS)
	KC	Store	Store	Store	Store	Store
		Load	Load	Load	Load	Set
LUUO	KL	Store	Store	No	No	No
		No	Load	No	No	No
	KC	Store	Store	No	No	No
		No	Load	No	No	No
XJRSTF XJEN	KL	No	No	No	No	No
		Load	Load	No	No	Load
	KC	No	No	No	No	No
		Load	Load	Load	Load	Load

CHAPTER 10

SYSTEM TIMERS

10.1 SUMMARY

The KC10 processor implements several kinds of system timers using a combination of hardware and microcode assistance. There are three kinds of timers implemented in the basic CPU, as follows:

1. Time base clock

2. Interval timer

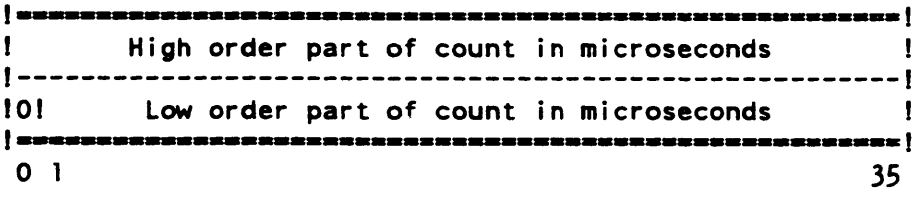
3. User runtime meter

In addition, the console contains a battery backup-up time-of-year clock that can be used to maintain the correct time through a power failure.

Unlike the KL10, the clocks on this machine will never update locations in memory unless requested to by the appropriate instruction in the monitor.

10.2 TIME CLOCKS

The time base and the user runtime meter are returned as a double precision integer with units of 1 microsecond. Both have the following format:



10.2.1 Time Base

The time base is kept in internal EBOX registers and implemented using a 16 bit hardware counter that counts in 1 usec units. The time base is controlled by the WRTMB and RDTMB instructions and it's current value may be read with the RDTIME instruction. The counter will overflow every 7.47 E7 years.

10.2.2 User Runtime Meter

The user runtime meter is similar to the time base described above. It is also a 1 usec counter that is driven from the same source as the time base, however it can be controlled by the monitor to count user runtime. Like the time base, the user runtime meter is also kept in internal EBOX registers and only written into UPT locations 504 and 505 as the result of a WRCTX that changes the UBR.

10.3 INTERVAL TIMER

The interval timer is used to supply a source of interrupts with programmable periods. It is a 12 bit counter that counts in 10us increments (100 kHz). It can therefore count and cause interrupts of any interval from 10us to 40.95ms. Its operation is identical to the KL10 interval timer.

CHAPTER 11

TRAP, UUO AND INTERRUPT HANDLING

11.1 INTRODUCTION

The current implementation of trap and interrupt handling, while quite general, has some problems relating to extended addressing. A new method of transferring control to "sections" other than the current PC section is given in the following section. This limits the number of possible actions that can be taken on a trap or interrupt compared to what the KI10 and KL10 now offer.

In addition, UUO handling has been changed to allow more flexibility and power in handling LUUOs and MUUOs.

11.2 TRAP FUNCTION WORD

EPT/UPT locations 421-423 contain a trap function word that determines the action of the processor when it detects an arithmetic overflow, stack overflow, or trap 3 condition.

The format of each word is as follows:

The format of this word is as follows:

0-1 Function code. This field is interpreted as follows:

- 00 Do nothing on trap condition (ignore)
- 01 Execute MUUO (take new PC from function specific argument)
- 10 Transfer control to exec/user depending on the mode in which the trap occurred. This function uses a LUUO-like block as described in the function specific argument below.
- 11 Do nothing on trap condition (ignore)

2-5 Available to software

6-35 Function specific argument. This field is used in a manner specific to the function performed as follows:

- 0 Ignored for this function.
- 1 New PC for the MUUO.

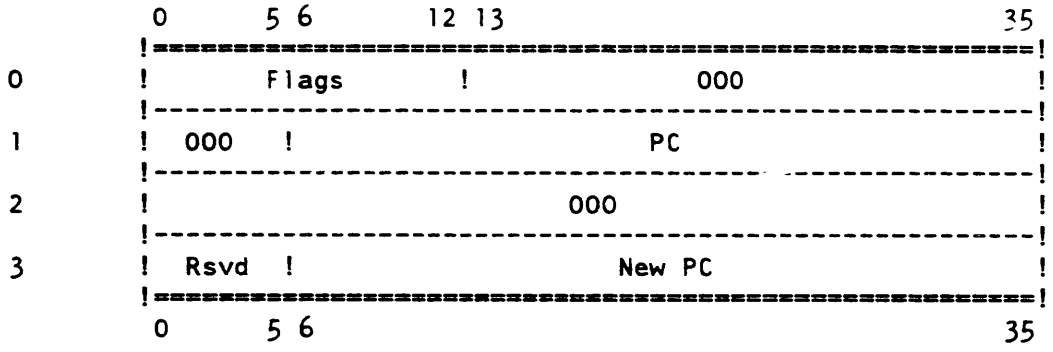
  This function stores only the program flags, CAB, PAB, PCS and the PC in UPT locations 424-425. The opcode, AC, and effective address of the instruction are NOT stored in UPT locations 426-427. The new program flags, CAB, and PAB are loaded from UPT location 430 as in a normal MUUO.
- 2 Virtual address in the current context (exec/user) of a 4 word LUUO-like block.

  This function stores only the program flags and the PC in words 0-1 of the block. The opcode, AC, and effective address of the instruction are NOT stored in words 0 and 2 of the block. The new PC is then taken from the

fourth word of the block.

3 Ignored for this function.

The format of the LUUO-like block used in function 2 is as follows:



Notes

1. The trap 1 and trap 2 flags are never stored in the MUUO (function code 1) or LUUO-like (function code 2) blocks when a trap is processed. It is the responsibility of the program to determine which trap condition occurred by supplying different new PCs for each possible condition.

2. On previous machines, traps were ignored if paging was disabled. On the KC10, traps are processed even if paging is off. It is the responsibility of the monitor to insure that there is an EPT setup with the appropriate trap function words even if paging is disabled.

3. An instruction that causes a trap and also jumps (e.g., AOJA) stores the PC of the destination of the jump, not PC+1 of the jump instruction.

11.3 VIRTUAL MACHINE SIMULATION MODE

The virtual machine simulation mode (VM mode) implemented by the KC10 allows an operating system to run a program in user mode in such a way that the program cannot distinguish its environment from a stand-alone exec mode machine. The primary use for this mode is to allow a monitor to be tested and/or debugged on a timesharing machine in user mode by concealing the fact that it is indeed running in user mode.

This mode is enabled for a user process with bit 9 of word E of WRCTX. In order to do this, the EBOX microcode must generate an MUUO trap for any instruction that differs between exec and user mode. It is then up to the (real) monitor to simulate the instruction properly to conceal the fact that the program is really running in user mode.

If VM mode is enabled for a user process, the EBOX microcode will generate an MUUO trap through the VM mode new PC word (location 431) in the UPT. There are four classes of instructions that trap through the VM mode new PC word as follows:

1. Any instruction that would normally trap as an MUUO through one of the other MUUO new PC pairs. This includes all unassigned opcodes, all legal MUUOs, all undefined EXTEND opcodes, JSYS, I/O instructions, MAP, JRST 3, HALT, XJEN, XPCW, JRST 10, JRST 11, JEN, JRST 13, JRST 16, and JRST 17,. This class of instructions is included because the new PC word for MUUOs is taken from different UPT locations based on whether the MUUO was executed in user or exec mode.

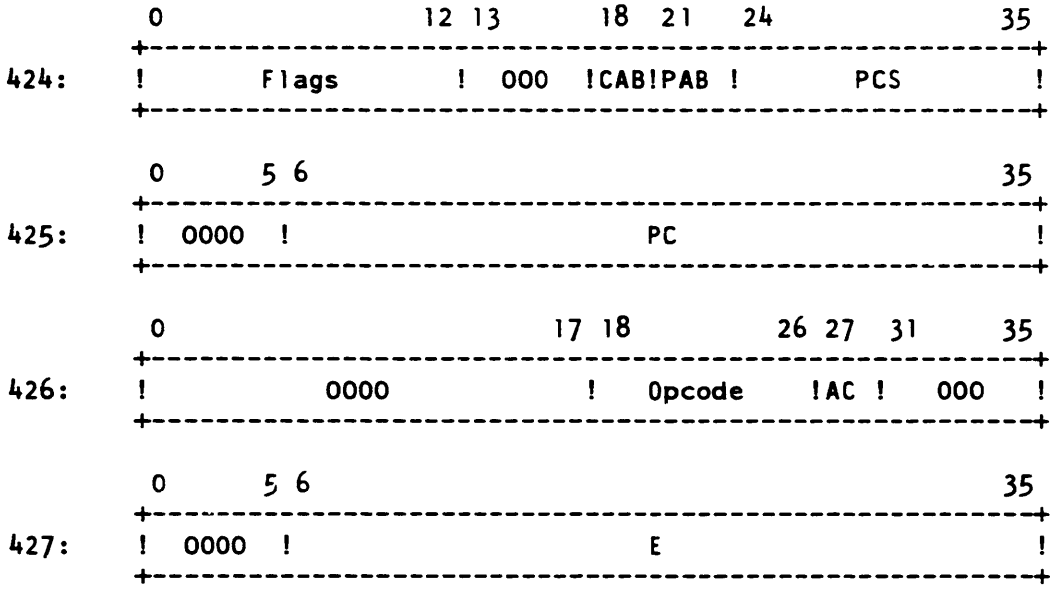
2. XCT with non-zero AC. This class is included because XCT with a non-zero AC in exec mode specifies a PXCT.

3. All LUUOs. This class is included because LUUOs use blocks in either the EPT or UPT based on whether the LUUO was executed in user or exec mode.

4. PUSHJ, JSR, and JSP in section 0. This class is included because the specified instructions store the flags (with the user-mode bit) if they are executed in section 0.

11.4 MUUO HANDLING

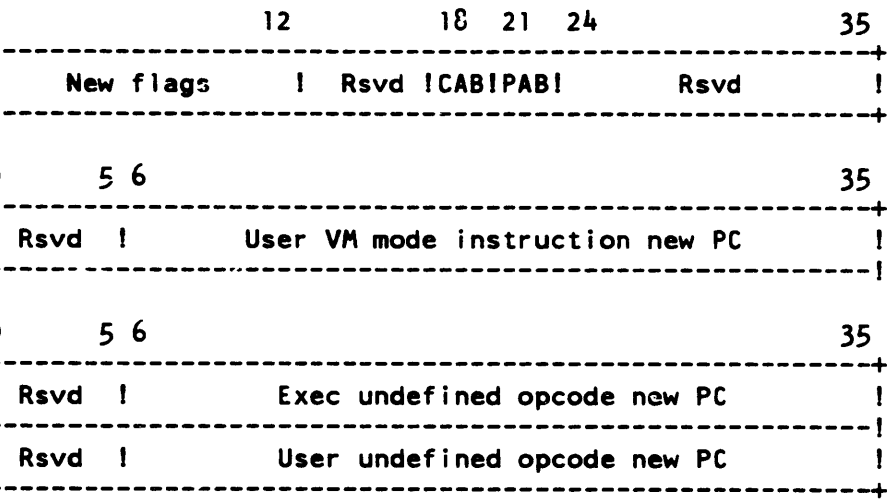
MUUO handling on the KC10 is significantly different from that of any previous processor. Instead of the previous format of UPT locations 424-427, the following format is used to store the program flags, CAB, PAB, PCS, PC, Opcode, AC, and effective address of the MUUO:

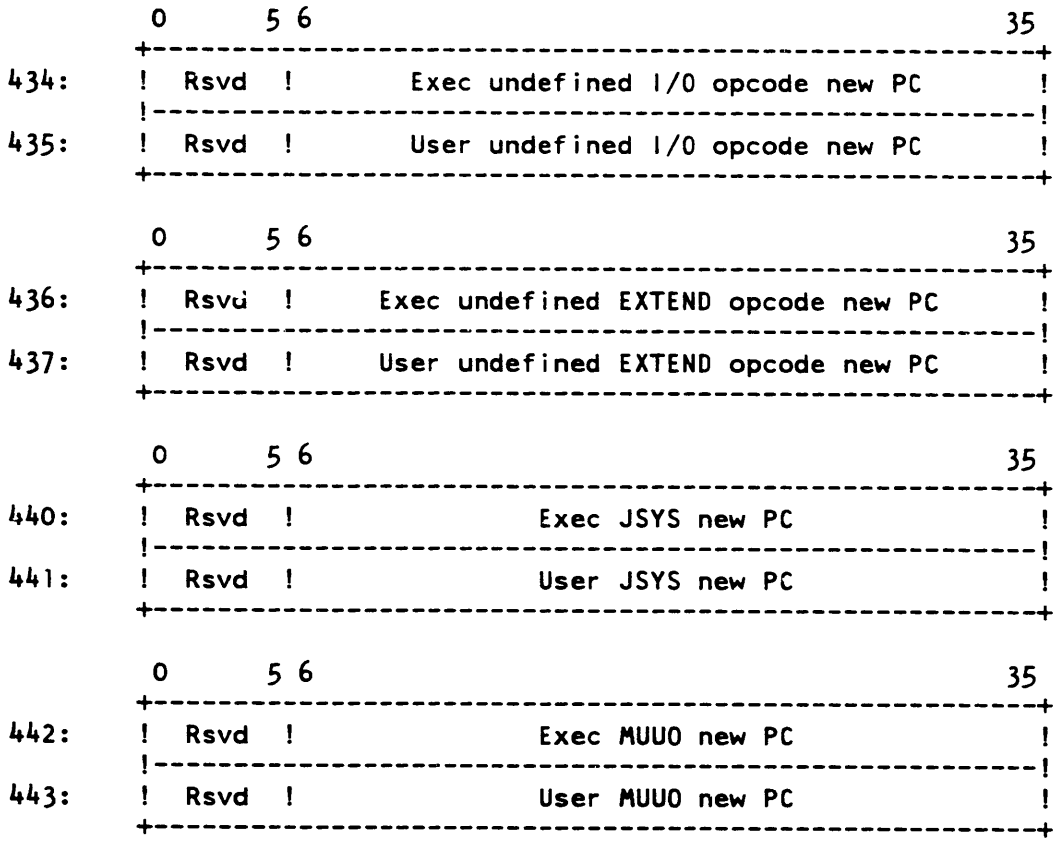


The new program flags, current, and previous AC blocks are loaded from the word at UPT location 430. The new PC is taken from one of the words of the dispatch vector beginning at UPT location 431, based on the MUUO opcode and whether the MUUO was executed in user or executive mode. The dispatch vector consists of pairs of words, one for user and one for exec, (location 431 is the exception to this rule) and contains 5 separate MUUO dispatches plus words reserved for future expansion. The dispatches are as follows:

Offset	Use
431	Instructions trapped in user mode as the result of virtual machine simulation mode enabled. See the discussion above.
432-433	Opcode 0 and all unassigned opcodes less than 700.
434-435	Unassigned opcodes in the range 700-777 plus any instruction that is executed in user mode without user I/O enabled that requires user I/O. This includes all internal and external I/O instructions, MAP, JRSTF executed in a non-zero section, JRST 3, HALT, XJEN, XPCW, JRST 10, JRST 11, JEN executed in a non-zero section or in user mode, JRST 13, JRST 16, and JRST 17,.
436-437	Undefined EXTEND opcodes
440-441	JSYS (opcode 104)
442-443	All other MUUO opcodes

The format of these words is as follows:



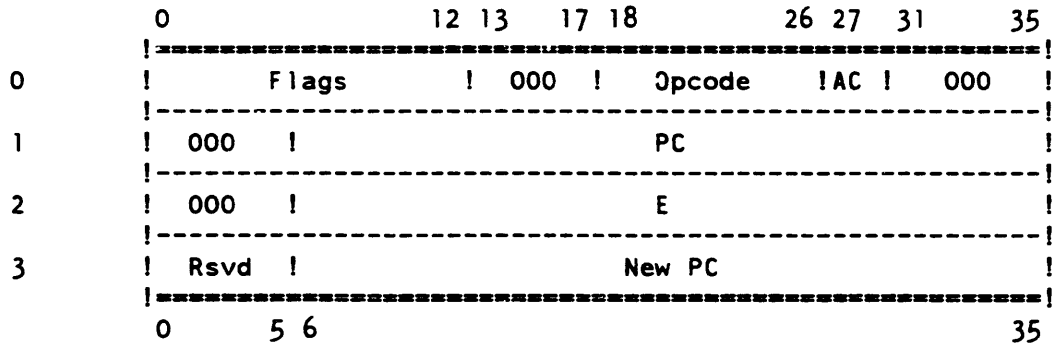

0 5 6 35
434: Rsvd Exec undefined I/O opcode new PC
435: Rsvd User undefined I/O opcode new PC
0 5 6 35
436: Rsvd Exec undefined EXTEND opcode new PC
437: Rsvd User undefined EXTEND opcode new PC
0 5 6 35
440: Rsvd Exec JSYS new PC
441: Rsvd User JSYS new PC
0 5 6 35
442: Rsvd Exec MUUO new PC
443: Rsvd User MUUO new PC

11.5 LUUO HANDLING

If the program is running in section 0, store the opcode, AC, and the effective address in bits 0-8, 9-12, and 18-35 respectively of location 40; clear bits 13-17. Then execute the instruction contained in location 41. An LUUO executed in user mode uses virtual locations 40 and 41 in the user program. An LUUO executed in executive mode uses locations 40 and 41 in executive virtual address space. This action is identical to the KL10 implementation.

If the program is running in a nonzero section, use bits 6-35 of UPT location 420 if the program is running in user mode, or EPT location 420 if the program is running in exec mode, as the address of a block of four words. In the first three locations of the block, store the program flags, opcode, AC, effective address, and PC of the LUUO. Then take the next instruction from the location specified by bits 6-35 of the fourth word of the block. In user mode, this action is identical to the KL10 implementation. In executive mode, this action is different from what is currently documented, but identical to what the KL10 actually implements.

The format of the block is as follows:



11.6 TRAP ENABLE

WREBR bits 8 and 9 affect how the processor handles traps, LUUOs, MUUOs, and page fails. If the monitor enables full processing of these conditions (by setting WREBR argument bits 8 and 9), the microcode will process these conditions as described above. If the monitor disables full processing of these conditions (the default power-up state of the machine), the microcode will process them differently as described below:

1. Traps. The microcode will treat trap 1, 2, and 3 conditions as if the trap function word had specified "ignore trap".

2. LUUOs. LUUOs executed in section zero (or in the low 256K with paging off) will be treated exactly as they are now, i.e., they will store the LUUO in location 40 and execute the instruction in location 41. Note that LINK stores a HALT instruction in location 41 when it loads programs.

   LUUOs executed in non-zero sections will halt the machine.

3. MUUOs. MUUOs will halt the machine.

4. Page fails. Page fails that must be processed by the monitor will halt the machine. Page fails that can be resolved entirely by the EBOX microcode will continue to be processed normally.

This special handling will cause the machine to halt when a condition for which the program is unprepared occurs instead of doing something unexpected. As a result, conditions for which the monitor is unprepared to handle will be detected early as the result of the condition instead of as a by-product of the condition.

11.7 INTERRUPT VECTORS

All interrupts happen through interrupt vectors located in the I/O page. A vector is a 30-bit Exec Virtual Address pointing to a 4-word block that is similiar to a XPCW control block. Return from an interrupt should be made by an XJEN instruction that addresses the same block. The saving and restoring of the "previous" context is described in a preceding section. The new context will be set up from the XPCW control block. The action of an interrupt cycle will be as if an actual XPCW was executed with its EA taken from the appropriate location in the I/O page.

An interrupt vector has the following format:

Where:

0-5 Reserved

6-35 Vector address of control block.

11.8 I/O PAGE FAILURE

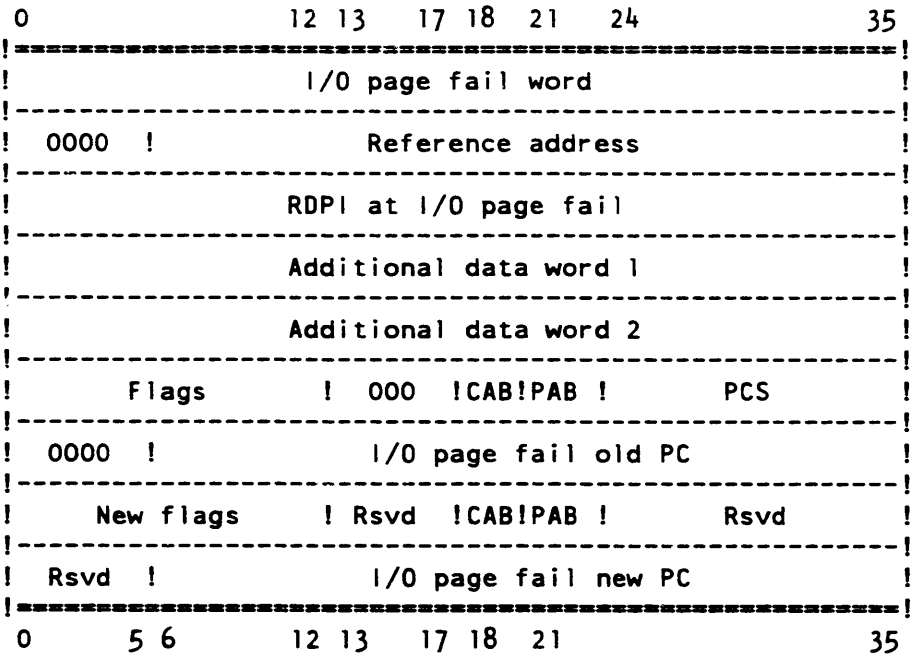
An I/O page fail can occur if the EBOX microcode is unable to fetch a word neccessary to process an interrupt request. This condition can occur if a hardware error or address break page fault occurs while trying to read a port interrupt vector word (I/O page locations 210-217), a port interrupt PI status word (I/O page locations 220-227), or a software interrupt vector word (I/O page locations 231-237). It can also occur if a request to access one of the four words pointed to by an interrupt vector page fails. In this case, the EBOX microcode generates an I/O page failure.

This page fail will be similar to a normal page fail trap, but the page fail information is contained in I/O page locations 240-250 instead of in the UPT. The EBOX stores a page fail word, reference address, PI status at the time of the failure, and the additional data words (identical in format to those stored by a normal page fail) in locations 240-244. The old PC double word is stored in locations 245-246 and the new program flags, CAB, PAB, and PC will then be taken from I/O page locations 247-250 and the processor will resume execution at the PI level on which the failure occurred.

The format and contents of words 240-241 and 243-244 are identical in format to the words stored in UPT locations 450-451 and 453-454 for a normal page fail. The format of these words is described in the chapter on paging.

The PI status stored in word 242 is identical in format to that returned by a RDPI instruction.

The format of the I/O page fail locations in the I/O page is as follows:



CHAPTER 12

MISCELLANY

This chapter contains miscellaneous information about the KC10 that doesn't fit anywhere else.

12.1 HALT STATUS CODES

When the EBOX microcode halts the EBOX for some reason, it stores a halt status code that describes the reason for the halt. This code can be retrieved by the console and printed on the CTY when a halt occurs. Note that such a code is not stored on an EBOX halt that wasn't caused by the EBOX microcode. The halt status codes are as follows:

0 The processor executed a HALT (JRST 4,) instruction.

1 A non-zero section LUUO was executed and trap enable was off in the WREBR argument word.

2 An MUUO was executed and trap enable was off in the WREBR argument word.

3 A page fail that must be resolved by the monitor occurred and trap enable was off in the WREBR argument word.

4 An illegal destination address was generated from EBOX dispatch 67. Early decode bits 4-6 are probably incorrect for the instruction being executed.

5 A MOVSxx memory read that previously caused a page fault did not do so when the reference was retried.

6 A MOVSxx memory write page failed.

7 A page fault generated as the result of a physical memory reference didn't result in a monitor page fail trap. All page fails that can result from a physical reference should require monitor intervention, so the page fault or the page fail word was illegal.

10 An interrupt was requested on PI level 0.

11 The EBOX microcode was trapped to an unimplemented (777x) microtrap vector.

12 The determination of the reason for an IBOX trap to EBOX with EBOX dispatch 25 resulted in an illegal trap reason (no reason bits were indicated in the dispatch).

13 The EBOX microcode page fail handler attempted to decode the reason for the page fail from the page fail word supplied to it and couldn't find a reason for the fault. The page fail word was probably illegal.

14 An instruction that is currently unimplemented in the EBOX microcode was executed.

15 The monitor attempted to turn on the (unimplemented) TOPS-10 paging mode (bit 3 off, bit 4 on in the WREBR argument).

CHAPTER 13

SPECIAL SYSTEM PAGES (EPT / UPT / IOP)

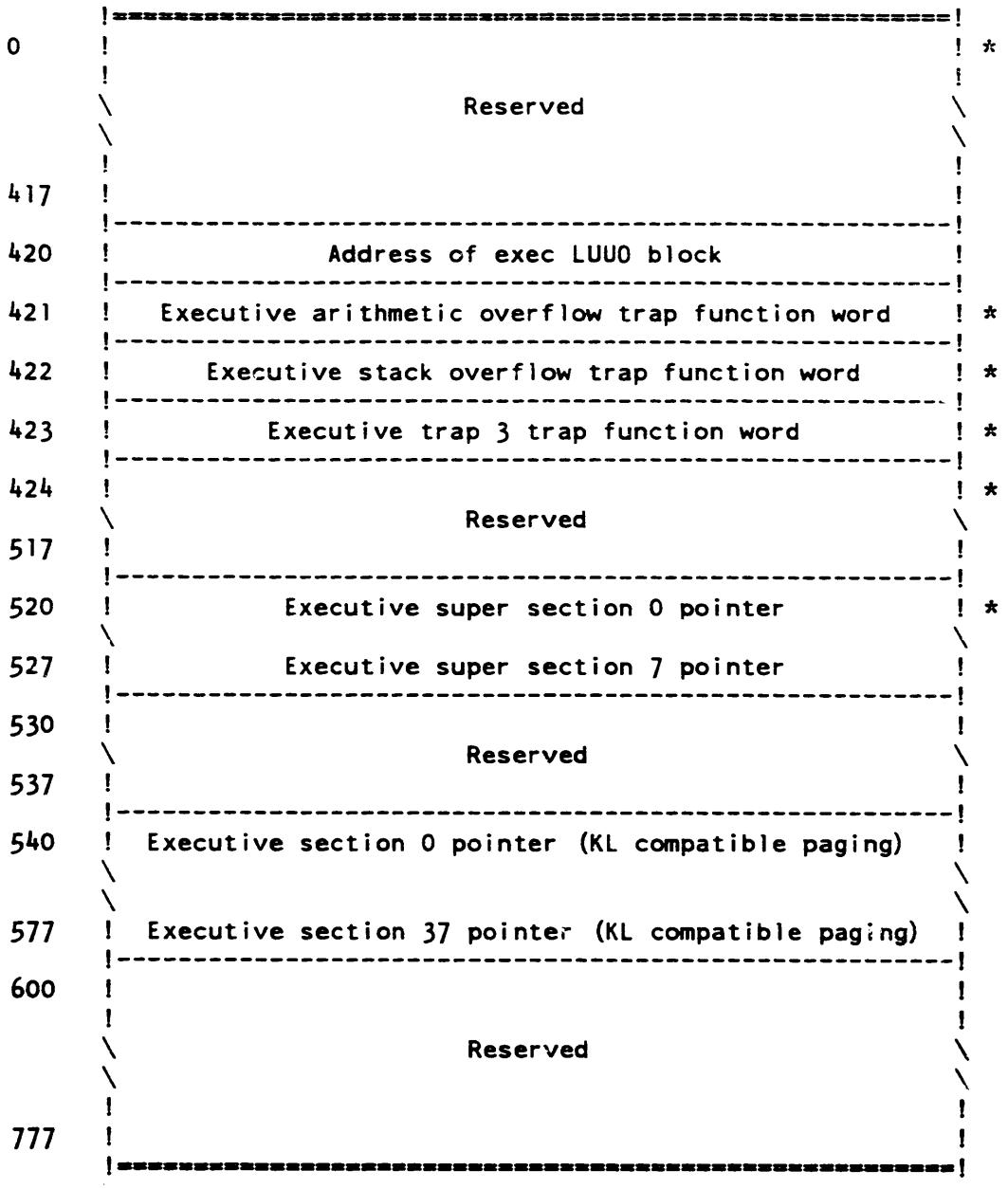
The following EPT/UPT layouts are proposed for the KC10. In addition, there is a new page called the I/O page (IOP) that is used by the KC10 ports and the console for communication with the CPU.

Upon processor reset, the base address of the EPT and UPT will be reset to page 0 and the I/O page will be reset to page 1.

NOTE

All areas that differ from the KL10 are marked with an asterisk (*).

TOPS-20 paging executive process table configuration

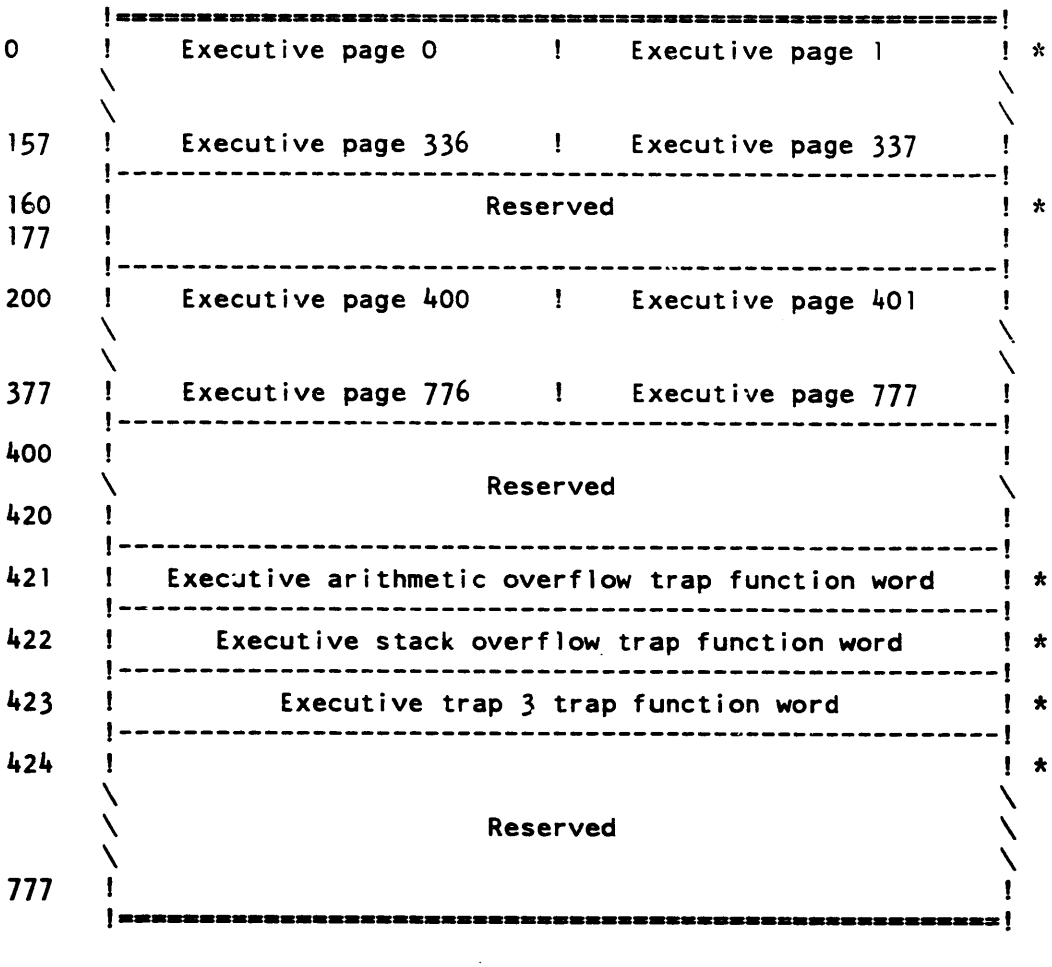


NOTE

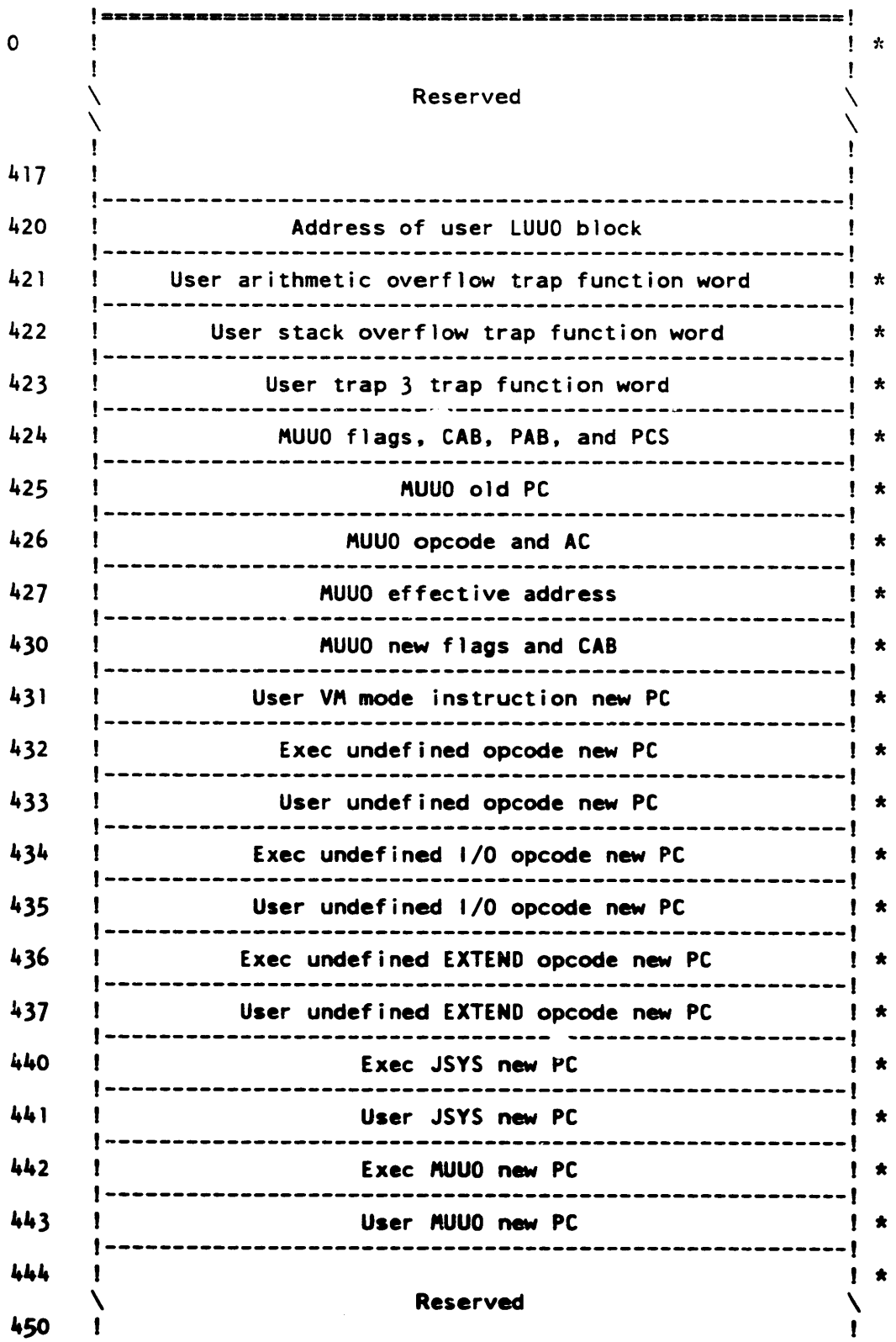
Location 420 of the EPT contains an address of an LUUO block identical to the one in the UPT. LUUO's in Kernel mode from code running in non-zero sections work identically to those in user mode. This is different from what is currently documented, but not

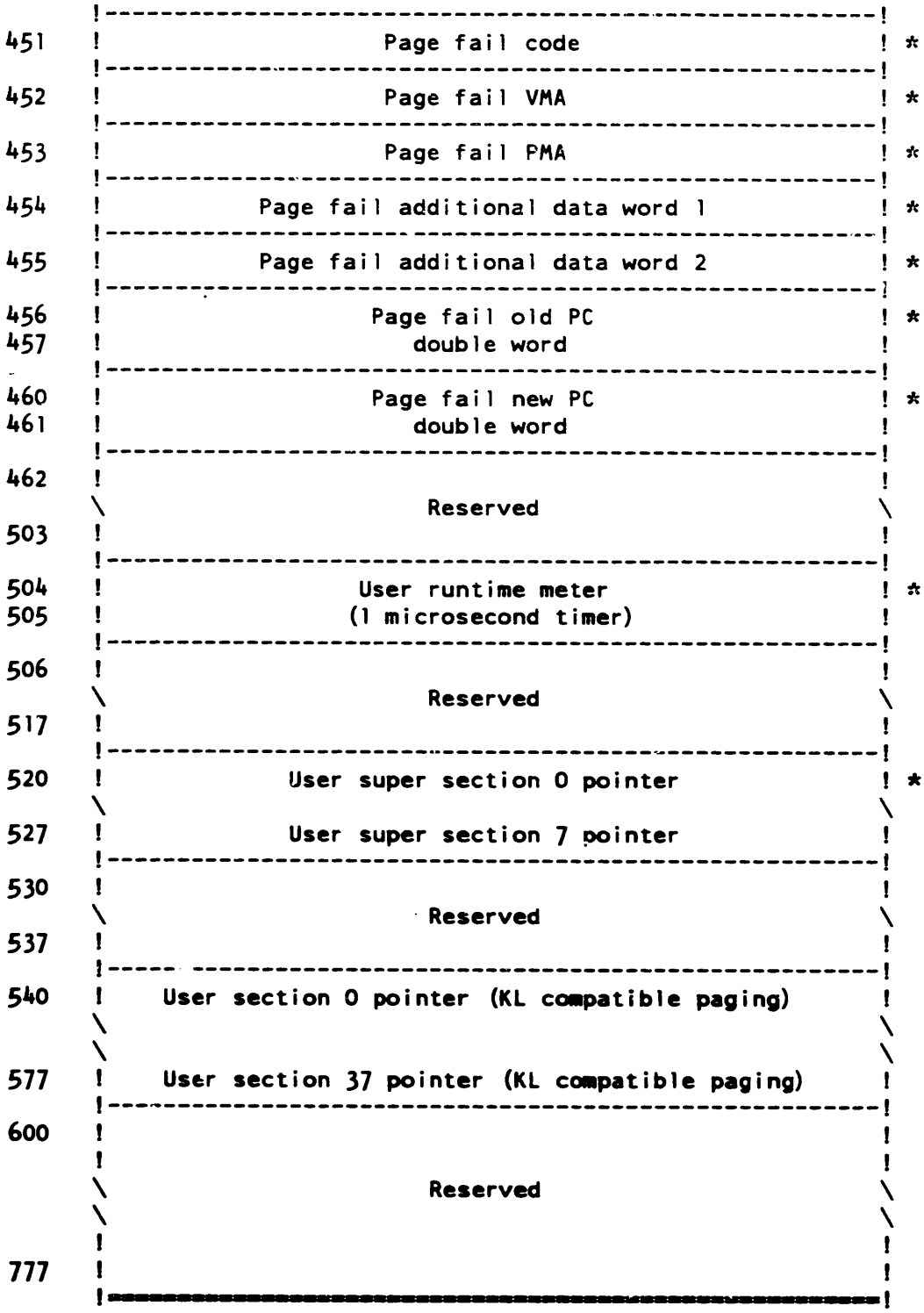
different from what is implemented on the KL10.

TOPS-10 paging executive process table configuration

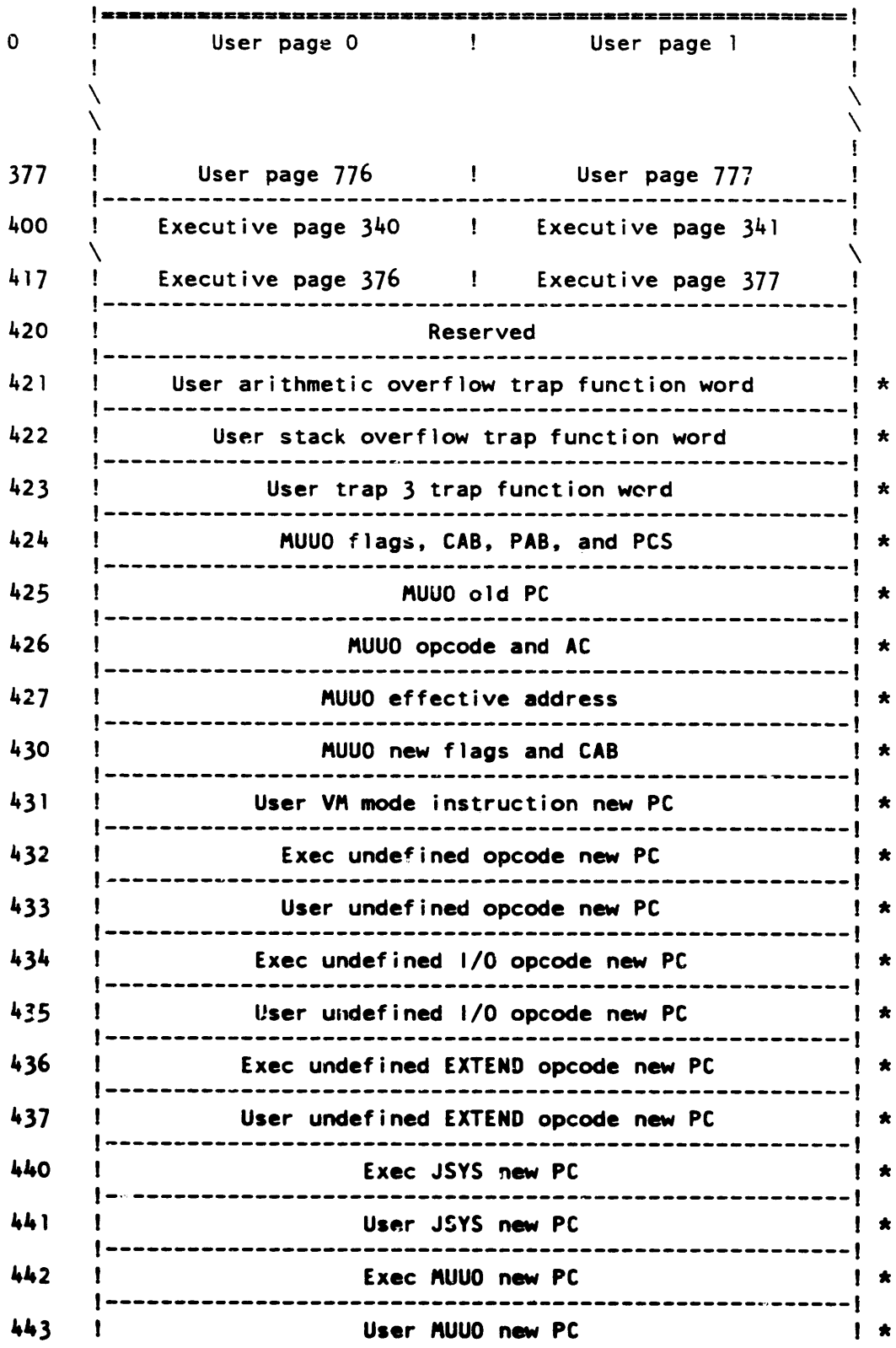


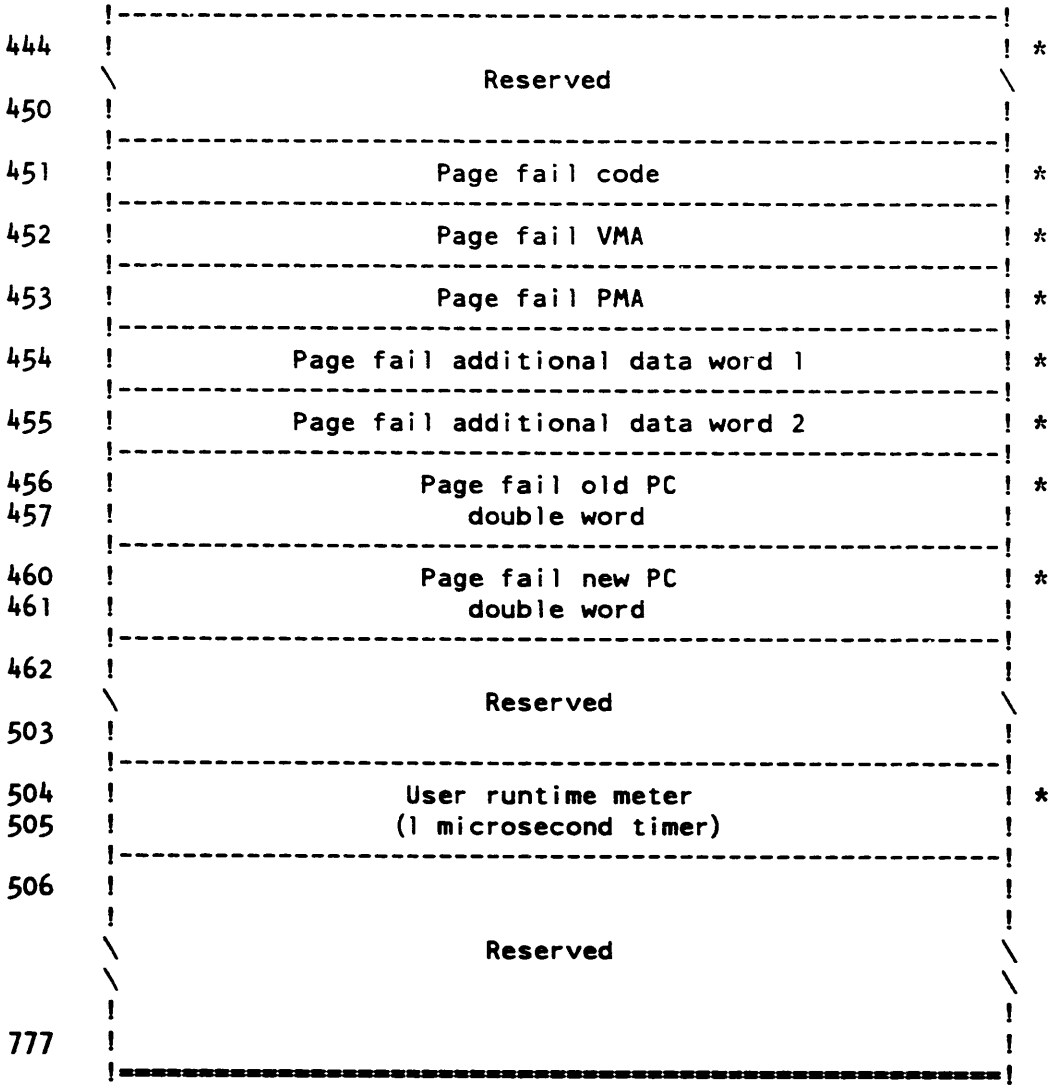
TOPS-20 paging user process table configuration



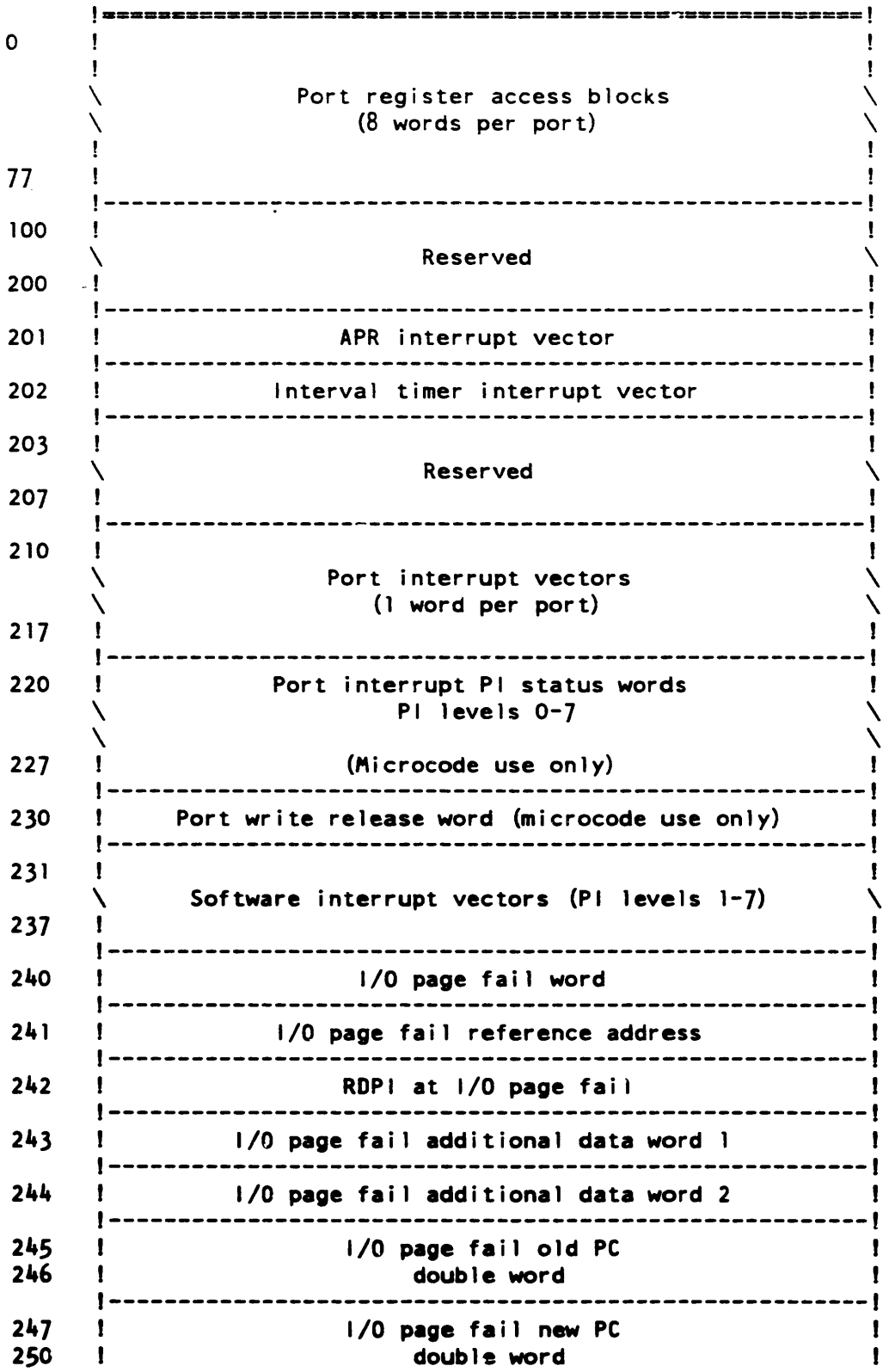

451 Page fail code *
452 Page fail VMA *
453 Page fail PMA *
454 Page fail additional data word 1 *
455 Page fail additional data word 2 *
456 Page fail old PC *
457 double word
460 Page fail new PC *
461 double word
462 Reserved
503
504 User runtime meter *
505 (1 microsecond timer)
506 Reserved
517
520 User super section 0 pointer *
527 User super section 7 pointer
530 Reserved
537
540 User section 0 pointer (KL compatible paging)
577 User section 37 pointer (KL compatible paging)
600 Reserved
777

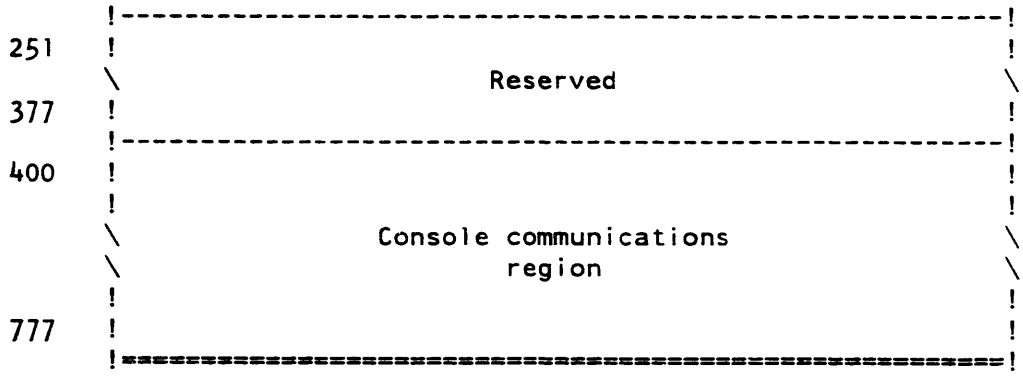
TOPS-10 paging user process table configuration




444
Reserved
450
451
Page fail code
452
Page fail VMA
453
Page fail PMA
454
Page fail additional data word 1
455
Page fail additional data word 2
456
Page fail old PC
457
double word
460
Page fail new PC
461
double word
462
Reserved
503
504
User runtime meter
505
(1 microsecond timer)
506
Reserved
777

I/O Page


0
Port register access blocks
(8 words per port)
77
100
Reserved
200
201
APR interrupt vector
202
Interval timer interrupt vector
203
Reserved
207
210
Port interrupt vectors
(1 word per port)
217
220
Port interrupt PI status words
PI levels 0-7
227
(Microcode use only)
230
Port write release word (microcode use only)
231
Software interrupt vectors (PI levels 1-7)
237
240
I/O page fail word
241
I/O page fail reference address
242
RDPI at I/O page fail
243
I/O page fail additional data word 1
244
I/O page fail additional data word 2
245
246
I/O page fail old PC
double word
247
250
I/O page fail new PC
double word


251
Reserved
377
400
Console communications
region
777

CHAPTER 14

ADDRESS BREAK

The address break feature of the hardware implements a superset of the KL10 address break capability. It may be used to determine whether a program is reading, writing, or fetching instructions from a range of locations in either user or executive address space and in either virtual or physical memory. The address break feature may also be used to determine if a port is reading or writing a range of locations in physical memory.

The address break enable and break conditions may be set by the WRCTX instruction and read by the RDCTX instruction. A description of the address break related fields in the WRCTX instruction follows:

The first word of the effective address (E) of the WRCTX instruction controls the action of the instruction. The bits in this word that affect address break are:

7 Inhibit all address break conditions for the next instruction executed. The effect of setting this bit is to set the inhibit address break PC flag for the next instruction. The intended use of this bit is to allow the instruction sequence:

```
WRCTX   ADR1            ;Turn on address break
XJRSTF  ADR2            ;Dismiss page fault
```

to be placed at the end of the monitor page fail routine. If the address break conditions are such that the hardware is breaking on all monitor instruction fetches, this bit allows the monitor to execute the XJRSTF to dismiss the address break page fault. It is assumed that the PC flags that are the argument to the XJRSTF will also contain the inhibit address break bit to allow the monitor to execute the instruction that caused the original address break page fault.

8 Load address break conditions from the words at E+2 through E+4. If this bit is on, the address break qualifiers are loaded from word E+2, lower bound break address is loaded from E+3 and the upper bound break

address is loaded from E+4. If this bit is off, the address break conditions remain unchanged.

NOTE

Paging must be enabled (with WREBR bit 4) to load the address break conditions. If paging is not enabled, the result of loading the address break conditions is undefined.

9 Load address break enable from bit 10. If this bit is on, address break is turned on or off based on the state of bit 10. If this bit is off, the state of address break enable remains unchanged.

10 Enable/disable address break. If both bit 9 and this bit are on, turn on address break. If bit 9 is on and this bit is off, turn off address break. If bit 9 is off, the state of this bit is ignored.

The third word of the effective address (E+2) of the WRCTX instruction defines the conditions that determine when an address break will occur. The condition bits are as follows:

10 If this bit is on, enable address break for a normal fetch of an instruction in the program under control of PC.

11 If this bit is on, enable address break for any reference that reads except the normal fetch of an instruction. This includes retrieval of operands, address words in an effective address calculation, or an instruction to be executed by an XCT.

12 If this bit is on, enable address break for any reference that writes to memory.

13 If this bit is on, enable address break for a reference made in user virtual address space. If this bit is off, enable address break for a reference made in executive space (either virtual or physical depending on the state of bit 16).

14 If this bit is on, enable address break for any reference made from the CPU, i.e., from the IBOX or EBOX.

15 If this bit is on, enable address break for any reference made by a port, i.e., from the IOBOX. No address break page fail is generated for an address break that occurs as the result of a port reference. Instead, the MBOX completes the request normally (i.e., the read or write

succeeds) and notifies the port that the request caused an address break. The port sets bit 12 in the port status register to indicate to the monitor that an address break occurred as the result of the transfer.

NOTE

Due to the implementation of this feature, the monitor cannot be assured that the transfer completed without errors if an address break occurs. Therefore, the monitor must retry the transfer with port address break disabled.

16 If this bit is on, enable address break for a physical memory reference. If this bit is off, enable address break for a virtual memory reference. Note that the break addresses must be physical if this bit is on and virtual if this bit is off.

17 If this bit is on, compare only the low order 18 bits of the reference address with the address range when doing address compares. This allows the program to cause an address break on an address in any section.

There are certain combinations of the above bits that produce unspecified results. These combinations are as follows:

If bit 10 is on, then bit 15 must be off because ports never fetch instructions.

If bit 10 is on, then bit 16 must be off because instruction fetches are always done from virtual memory.

If bit 13 is on, then bit 16 must be off because user references are always done through virtual space.

If bit 15 is on, then bit 16 must be on because ports always make physical references.

The fourth and fifth words of the effective address (E+3 and E+4) of the WRCTX instruction specify the lower and upper bound break addresses. When doing address break compares, the MBOX compares the reference address with the upper and lower bound break addresses. Normally, the full reference address is used in the compares (i.e., 30 bits of virtual address for virtual compares and 25 bits of physical address for physical compares). However, if bit 17 is on in the third word of the WRCTX argument block, only bits 18 through 35 of the reference address are used in the compares. If the reference address is greater than or equal to the lower bound break address and less

than or equal to the upper bound break address, the address compare succeeds.

The conditions under which an address break will occur in the MBOX may be described as follows:

Let:

A := Condition bit 10 on and an instruction fetch reference.
B := Condition bit 11 on and a read reference.
C := Condition bit 12 on and a write reference.
D := Condition bit 13 on and a user reference, or condition bit 13 off and a executive reference.
E := Condition bit 14 on and a reference made by the CPU.
F := Condition bit 15 on and a reference made by a port.
G := Condition bit 16 on and a physical reference, or condition bit 16 off and a virtual reference.
H := Condition bit 17 is off and the reference address is within the range described by the lower and upper break addresses or condition bit 17 is on and bits 18 through 35 or the reference address is within the range described by the lower and upper break addresses.

Then a port address break will occur if the following expression is true:

(A OR B OR C) AND (F) AND (H)

and a CPU address break page fail will occur if the following expression is true:

(A OR B OR C) AND (E) AND (D) AND (G) AND (H)

The expressions are given separately for CPU and port references because that is the way they are implemented in the MBOX hardware.

If an address break page fault does occur, the microcode will turn off address break before dispatching to the monitor page fault handler. It is the monitor's responsibility to turn address break back on before dismissing the page fault if the page fault was the result of an address break. The microcode WILL NOT turn off address break for any page fault except an address break page fault. This allows the monitor to trace executive instruction fetches by setting the address break conditions to cause a page fail for each instruction fetched from executive virtual space.

NOTE

If address break is enabled for a range of memory addresses, an instruction that references multiple words in this range will only cause an address break condition for the first word referenced if the monitor restarts the instruction with the "inhibit address break" PC flag set. This is because the "inhibit address break" PC flag remains set for the completion of execution of the instruction and blocks further address breaks.

INDEX