

PDP-10/X System Manual

David G. Conroy (dgcx@mac.com)

Saturday, July 30, 2011

Introduction

This document describes the PDP-10/X, a new implementation of the venerable PDP-10 architecture from Digital Equipment Corporation.

This document describes the PDP-10/X at a level appropriate for assembly language programmers, and includes considerable information regarding low-level I/O device programming. It is not a complete description of the PDP-10 architecture (the original DEC manuals serve this function), although it does describe implementation details in places where the DEC manuals are unclear and/or there is a tradition of variation from model to model. It is also not a complete description of some of the I/O components (the manufacturer's data sheets and external standards serve this function), although it does describe how the components are interconnected.

The PDP-10/X has three major blocks.

The first block is the processor (APR). The APR block is a new hardwired model of the PDP-10 that most closely resembles a KA10 to which ITS paging has been added (for example, it implements KA10-style floating point arithmetic, it has one block of ACs, the MUUO and PI instructions are read from fixed locations in memory, and APR conditions such as traps and page failures are reported using the PI system) but which is not exactly compatible with the KA10 (for example, some of the quirks of the KA10's implementation that were fixed in all later models are fixed).

The second block is the pager and memory controller (PAG). The PAG block services all memory-space reads and writes that appear on the system bus, translating the protocol used on the system bus into the protocol used by the memories. The current version of the PAG block implements 22-bit physical addressing, but the PDP-10/X only implements 512KW of memory (five 512Kx8 15ns SRAM chips). The system bus protocol allows the APR block to access memory using virtual addresses, which are translated into physical addresses by the PAG block using a method appropriate for running ITS; although it might seem more appropriate to perform this translation in the APR block, putting it in the PAG block allows the in-memory structures used by address translation to be accessed at memory speed instead of system bus speed.

The third block is the basic I/O system (BIO). The BIO block contains the I/O devices needed by a basic, but complete, ITS system; a line frequency clock, an real-time clock with a battery backup (to keep track of the date and time, even when the system is not running), a console terminal interfaces, and a disk interface.

The line frequency clock does not actually run off the line (it runs off the crystal used to generate the bit clock for the console terminal) so it always runs at 60 Hz, even if the system is running off a battery.

The console terminal interface communicates with 2-wire RS-232 devices at 9600 bits per second. The programming model is more like a traditional DEC interface than any industry-standard interface, and is designed to simplify the modifications to the ITS terminal driver (a previous PDP-10 design used an industry-standard 16450/16550 UART, and the modifications to the ITS terminal driver to handle such a UART were harder to do than they first appeared to be; it's hard to make a 16450/16550 generate a transmit-side PI on demand).

The disk interface that supports one or two ATA-2 disks. All disk transfers are performed using programmed I/O (programmed I/O is almost as good as DMA in a system like the PDP-10/X, since an ATA-2 disk contains a sector buffer, and provides data in a long burst that would monopolize the memory and cause the APR to stop dead anyway), and since disk DMA is not used, disk-like devices that are more-or-less compatible with the ATA-2 standard but do not support DMA (like compact flash cards) can be used. The disk interface does not resemble any traditional PDP-10 design.

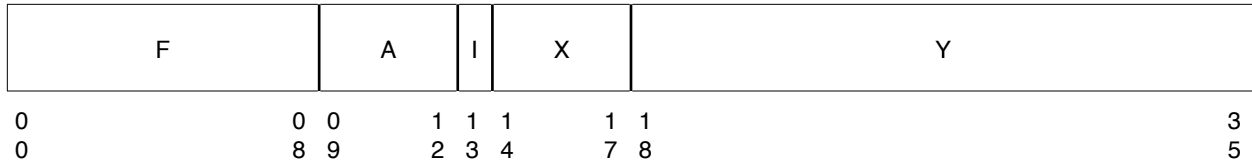
The three major blocks (APR, PAG, BIO) are connected together by a the system bus (a fourth block, the CLK block, generates all the clocks and resets on the system bus, and orchestrates the loading of the FPGAs in the other three blocks). The system bus is a simple synchronous bus that allows one device (the master) to read and write 36-bit words in any other device (the slave). The APR block is the default master, but can relinquish mastership of the system bus to another block on demand (the facility is not currently used, since there are no I/O devices that do DMA). Reads and writes can be in I/O space (in either the I/O condition space or the I/O data space; the PDP-10/X uses old-style PDP-10 I/O instructions) or in memory space (using either physical or APR virtual addresses).

The PDP-10/X runs a custom version of the Incompatible Timesharing System (ITS), created and maintained at MIT between 1966 and 1990. The PDP-10/X runs ITS both for its obvious hack value, and because a complex project like the PDP-10/X is best tackled in stages, and ITS will run on a somewhat simpler system than either TOPS-10 or TOPS-20/TENEX. Some thought has been given to running TOPS-10 and/or TOPS-20/TENEX, and the structure of the system is suitable for building a version of the PDP-10/X capable of running a modified version of TOPS-10 and/or TOPS-20/TENEX (the vast majority of the changes to the hardware are, of course, in the PAG block).

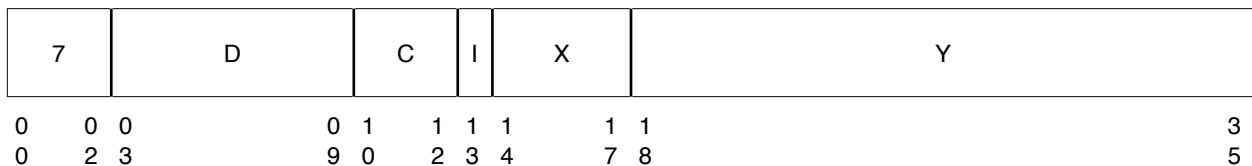
APR

Instruction Set

Non-I/O instructions use the standard PDP-10 format.



I/O instructions, which have bits [00..02] = 111, use the old-style I/O instruction format.



The D field specifies the device and the C field specifies the I/O command in the traditional way. The APR does not implement BLKI (C = 0) or BLKO (C = 2), and the behavior of an instruction with a C field is 0 or 2 is undefined; the current implementation treats BLKI as DATAI and treats BLKO as DATAO.

The APR implements the user-mode instruction set of the KA10 (including the somewhat odd KA10 double-precision floating point), although it does not replicate the quirks of the KA10 that were eliminated from all later models (for example, the fact that adjustments to things like stack and byte pointers were done with fullword adds and subtracts, and that a carry in the right half could ripple into the left half), nor does it replicate the incorrect answers that were generated by some KA10 instructions when presented with corner-case operands (usually 400000000000).

ITS was originally designed for the KA10 (and its predecessor, the 166), and although ITS can (and was) made to work on newer models of the PDP-10, the executive mode architecture of those newer models (process tables, page failures that are not reported via a PI) is somewhat at odds with the design. For this reason the executive mode architecture of the APR resembles that of the KA10, albeit one with paged memory management; there is a single block of ACs, the MUUO and PI vectors are in fixed locations in low memory, and traps and errors are reported using the PI system.

The APR block's APR device is considerably different from that of the KA10; the set of traps and errors is different, any trap or error can be assigned to one of two PI levels, and there are some special features to simplify the context switch.

Flags

A	C	C	F	F	C		F	D	
O	Y	Y	O	P	M		X	C	
V	0	1	V	D			U	K	
0	0	0	0	0	0		1	1	1
0	1	2	3	4	5	6	0	1	2
							3		7

The APR has no public mode, so bit [00] is always the AOV flag, and bit [07] is always 0. The APR has no user I/O mode nor previous mode, so bit [06] is always 0. The APR has no address failure inhibit, so bit [08] is always 0. The APR has a KA10-like APR trap architecture, so bits [09..10] (the TRAP1 and TRAP2 flags) are always 0.

The FPD flag is only altered by ILDB and IDPB; instructions that can exit in the middle and that cannot simply be restarted from the beginning (BLKI and BLKO would be on this list if they were implemented, but they are not). All other instructions leave the FPD flag alone. This is how all PDP-10 models actually worked, but this is not clearly explained in the documentation.

A user mode program (CM = 1) can only set CM to 0 by taking an interrupt or executing an MUUO.

JRST

The APR, like the KA10, treats the AC field of JRST as a set of bits, each of which enables a special function.

- [09] If one or more priority interrupt levels are active dismiss the highest priority level; do nothing if no priority interrupt levels are active
- [10] Halt.
- [11] Restore the flags from the left half of the word last read by the effective address calculation, be it an indirect address word or an accumulator used as an index register. If the effective address calculation does not read any words (if the initial instruction word does not specify indirection or indexing) the flags are restored from the left half of the instruction word. A JRST with bit [11] set to 1 only loads CM if CM is set to 0; a program running in user mode, with CM set to 1, cannot set CM to 0.
- [12] Set CM to 1 (user mode).

In executive mode (CM = 0) all special functions are allowed, although bit [12] is ignored if both bit [11] and bit [12] are set to 1.

In user mode (CM = 1) only the special function enabled by bit [11] is allowed. A JRST with any or all of bits [09], [10], or [12] set to 1 is treated as an MUUO.

The APR implements JRST with bit [11] set to 1 by keeping track of the left half of the last word as a side effect of the evaluation of the effective address. Some implementations (for example, the KS10), re-evaluated the effective address assuming that PC-1 was the address of the JRST, which is incorrect in some cases (for example, when the JRST is the target of an XCT).

MUOU and LUOU

The APR handles LUOU and MUOU instructions in almost the same way. A word containing the LUOU/MUOU opcode in bits [00..08], the LUOU/MUOU accumulator field in bits [09..12], zeros in bits [13..17], and the effective address in bits [18..35] is stored in some location 000040, and then the instruction in some location 000041 is executed.

If the instruction is an MUOU then the instruction word is stored in location 000040 of the executive mode address space, and the instruction from location 000041 of the executive mode address space is read and executed in (temporarily forced) executive mode; if the instruction in location 000041 is a JSR to the MUOU handler it will store FL,,PC+1 in the usual way and the return address will be that of the instruction immediately after the MUOU. If the instruction is a jump the APR will remain in executive mode; the CM flag will be set to 0 (JSR and JSP, as always, also set the FPD flag to 0).

If the instruction is an LUOU then the instruction word is stored in location 000040 of the current mode address space, and the instruction from location 000041 of the current mode address space is read and executed as an ordinary (non-interrupt) instruction in the current mode; if the instruction in location 000041 is a JSR to the LUOU handler it will store FL,,PC+1 in the usual way and the return address will be that of the instruction immediately after the LUOU.

Note that an LUOU and an MUOU end up being identical in executive mode, although their processing details are different.

If an MUOU is executed, and the instruction in location 000041 of the executive mode virtual address space contains an LUOU, the APR will execute the LUOU in (temporarily forced) executive mode, which will effectively make it an MUOU.

XCTR and XCTRI

Executive mode software needs to be able to access system call parameters in the user mode address space. The APR makes this straightforward by allowing executive mode software to modify the behavior of the XCT instruction so that a variety of useful subsets of the memory references made by the executed instruction use the user mode address space, even if the CM bit of the flags specifies executive mode.

XCTR/XCTRI C,E

2 5 6	C	I	X	Y
0	0 0	1 1 1	1 1	3
0	8 9	2 3 4	7 8	5

The XCTR and XCTRI instructions execute the instruction in the memory location specified by the effective address, and use the C field to hold a code that specifies which of the memory references made by the executed instruction should use the user mode address space.

Each memory reference made during the execution of an instruction belongs to one of five memory reference classes.

- IF Instruction fetches.
- E1 Index register references and memory references that are part of the effective address calculation of instructions, including the index register in the original instruction word.
- D1 Memory references that are simple memory operands, including the source operand of PUSH, the destination operand of POP, the destination operand of BLT, and the (first) byte pointer word.
- E2 Index register references and memory references that are part of the effective address calculation of byte pointers, including the index register reference in the (first) byte pointer word (this bit would also control source effective address calculations and operands in EXTEND, but the APR does not implement EXTEND).
- D2 Memory references that are byte data, and the source operand in BLT (this bit would also control destination effective address calculations and operands in EXTEND, but the APR does not implement EXTEND).

These are the same meanings of E1, D1, E2, and D2 that are used by the Tenex/TOPS-20 PXCT instruction, and inside the APR block itself.

Bits [10..12] of the C field encode several interesting selections of the E1, D1, E2, and D2 bits. The encoding is as follows.

ITS Symbol	C[10..12]	IF	E1	D1	E2	D2
-	0	-	-	-	-	-
XR, XW, XRW, XBW	1	-	-	X	-	-
XBR	2	-	-	-	-	X

ITS Symbol	C[10..12]	IF	E1	D1	E2	D2
XBRW	3	-	-	X	-	X
XEA	4	-	X	-	-	-
XBYTE	5	-	-	X	X	X
-	6	-	-	-	-	-
-	7	-	-	-	-	-

Bit [09] of the C field specifies how errors on I/O and relocated memory cycles should be reported. If this bit is 0 (XCTR) then errors are reported in the normal fashion; the current instruction is flushed, the HE or SE flag in the APR device is set, and a PI is generated if enabled. If this bit is 1 (XCTRI) then the current instruction is flushed and the PC is incremented by 2 so that the instruction after the current instruction is skipped. The status register in PAG is updated on soft errors in both cases; PI routines that use XCTRI will need to save and restore the status register in PAG to avoid confusion.

The XCTR instruction encodes the C field in a way that is different from the encoding used by the AC field of PXCT. It exploits the fact that ITS uses only a subset of the legal PXCT AC field values in order to find space in the C field for the bit that is used to distinguish between XCTR and XCTRI.

XCTRI skips on both hard and soft errors. XCTRI is normally used to read/write a location in user space from executive mode, with the skip telling the executive mode program that the read/write failed for some PAG-related reason. One might argue that XCTRI should only skip on soft errors, so that hard errors are not lost. On the other hand, if XCTRI skips on both hard and soft errors it can be used by the ROM program to avoid a double hard error halt should an E or D command be aimed at a non-existent location. The second design was selected, but it's not quite the right thing.

I/O cycles are treated as relocated cycles to allow the console program to use XCTRI to avoid a double hard error halt if an E or D command is aimed at a device register that does not exist. Normal software might find this feature useful for things like testing for the existence of a device.

The XCTR instruction only performs its relocation function when executed in executive mode. The C field is ignored in user mode.

Relocated references to locations 0-17 normally access locations 0-17 of the user mode address space (the shadow locations, normally hidden by the accumulators), but the PAG block has the ability to redirect these references to a special "AC block". ITS uses this feature to relocate references to these locations to one of the two "AC blocks" in a job's variables (UUOACS, AC0S-AC17S).

APR does not optimize the BLT case where the destination address is equal to the source address + 1 (the "filling memory with a constant value" case). It does not, therefore, need to do anything special to deal with the case where the BLT is the target of an XCTR or XCTRI and the code is XBR or XBW.

Traps and Errors

The APR has six condition flags that capture various exceptional events.

The E1 and U1 flags capture trap-1 (arithmetic overflow) events. The E1 flag captures trap-1 events that happen in executive mode, and the U1 flag captures trap-1 events that happen in user mode.

The E2 and U2 flags capture trap-2 (push-down overflow) events. The E2 flag captures trap-2 events that happen in executive mode, and the U2 flag captures trap-2 events that happen in user mode.

The SE flag captures soft error events on any cycles that the APR runs on the system bus. The SE flag becomes set if a device claims the cycle and completes it with a soft error acknowledge and the SE flag is not already set. In theory any device can claim a cycle and complete it with a soft error acknowledge, but the intention that the only cycles that would be completed in this way would be memory cycles claimed by the PAG device whose virtual addresses could not be successfully translated to physical addresses (that is, page failures).

The HE flag captures hard error events on any cycles that the APR runs on the system bus. The HE flag becomes set if no device claims the cycle, or some device claims the cycle and completes it with a hard error acknowledge. The HE flag also becomes set if some device claims the cycle and completes it with a soft error acknowledge and the SE flag is already set (double soft error). The system bus distinguishes between condition reads/writes and data reads/writes with an additional device address bit, so a hard error is generated if software executes a DATAI/DATAO on a device that only understands CONI/CONO/CONSZ/CONSO or executes a CONI/CONO/CONSZ/CONSO on a device that only understands DATAI/DATAO.

Software can arrange (by setting the appropriate bits in the APR device) that any of these six condition flags generate PI requests on one or both of two PI levels; having two levels allows conditions that correspond to fatal events to use a high-priority PI level and conditions that correspond to non-fatal events to use a low-priority PI level.

The APR guarantees that if an instruction sets a flag, the flag is configured to generate a PI, and the PI can be recognized, then the PI will be recognized before the next instruction begins execution. Traps and errors are reported using interrupts, but from a practical point of view, they can be treated as precise. The next instruction is not read and ignored (even though this would have simplified the implementation) so software

does not need to handle the troublesome situation where an instruction gets a trap, the read of the next instruction (which is ignored) gets another error, and both the trap and the (bogus) error get reported.

The trap flags (U1, U2, E1, E2) are set when the instruction that caused the trap completes, so if one of these flags causes a PI, the PC saved by the PI will point to the instruction after the instruction that set the trap flag. The error flags (HE, SE) are set when the instruction that caused the error is abandoned, so if one of these flags causes a PI, the PC saved by the PI will point to the instruction that set the error flag, and when the PI is dismissed the instruction that set the error flag will be re-executed.

The PI level assignments for trap and/or error handlers must be chosen so that the PI routines run at a higher priority than any code that might actually cause the events. At best, a trap will stick around until the JEN, when the trap PI handler will run, and promptly get very confused, since the saved PC has nothing to do with the PC at the time of the trap. At worst, an error will cause the APR re-execute the failing instruction, get another error, and get a double-error halt.

TOPS10 6.03 runs the APR's PI handler at high priority. There are comments in the code that say "happens only if user is enabled when monitor causes condition" and "store offending PC so we can fix code", which suggests that user jobs that enabled trapping were rare enough that occasional the occasional trap in the monitor was not a big deal (this code is just after the label APRER0 in CLOCK1.MAC). TENEX 134 does essentially the same thing. ITS does the same sort of thing as well (look at the code after the labels AROV and/or ARFOV) but it runs the APR PI handler at low priority (CLKCHN = 7) so it seems that traps in PI handlers could confuse things.

Priority Interrupts

A priority interrupt (PI) request on a level can be generated by hardware (the request signal from the APR-internal APR device and/or the request signal on the system bus associated with the level is true) or by software (the software request signal associated with the level is asserted). The APR can recognize a PI request on a level if a PI is not already in progress on the level or on any higher priority level, the PI system is enabled, and the PI level is enabled.

The KL10, and the KS10 with microcode revisions before revision 47, recognized a software interrupt on a level even if the level was not enabled. The PDP-10/X APR requires that the level be enabled, making it compatible with the KC10, and with the KS10 with microcode revision of 47 or later. ITS is not sensitive to this difference, since the trouble it caused was resolved long ago, and requiring the level to be enabled is the more modern interpretation.

The APR checks for PI requests just before it begins reading an initial instruction word, just before it begins reading the target instruction word of an XCT, just before it begins writing the F/A/EA word into location 000040 on a UWO, and just before it begins

reading any indirect addressing word (including any byte pointer word, which can be considered to be a kind of indirect addressing word). The APR also checks for PI requests just before it loops back to move an additional word on a BLT. These checks ensure that it is possible for a PI to break out of a within-an-instruction loop.

If the APR finds one or more PI requests it picks the one having the highest priority, and begins recognizing that PI request. The APR recognizes a PI request on level N by setting the in-progress flag for level N, reading an instruction from executive mode location $000040+2*N$, and “executing” the instruction in executive mode.

The interrupt instruction is not really “executed” by the APR. The only instruction that is legal as an interrupt instruction is JSR (264000,,EA), and it is easier to “execute” the JSR using a dedicated logic sequence (all it needs to do is write FL,,PC into executive mode location EA, update FL in the manner of a JSR, and write EA+1 into PC).

Restricting the set of legal interrupt instructions to simplify their PI handling was commonplace in PDP-10 designs; the KS10, for example, only allowed JSR and XPCW as interrupt instructions.

If the APR begins recognizing a PI request, but cannot complete recognizing the PI request (because it cannot read the instruction from executive mode location $000040+2*N$, or the instruction is not of the form 264000,,EA, or it cannot write FL,,PC to executive mode location EA) the APR halts.

HALTS

The PDP-10 architecture allows a APR to halt; when halted a APR does not execute instructions until it is restarted by some explicit command, either from a switch on a front panel, or by a command executed by a front-end processor, depending on the implementation.

The PDP-10/X APR supports halt, but barely. When the APR halts it stops executing instructions, but there is no way to restart it other than resetting the entire system. This is acceptable because the APR is only halted in the most desperate of situations; in slightly less desperate situations control gets passed to EDDT, which supports the same kind of low-level debugging as a front panel, but with a nicer interface.

A normal halt happens when a HALT instruction is executed.

An abnormal halt happens when the APR gets into a state in which forward progress is impossible. Currently forward progress is considered to be impossible if a hard error is detected and the HE flag is set (double hard error), or if the APR begins but cannot complete a PI entry sequence (either because of a hard or soft error on any of the memory references associated with the PI entry sequence, or because the interrupt instruction is not a JSR instruction).

There is no front-panel switch to force the APR to halt; such a switch would not be terribly useful. There is a front-panel switch, however, to generate an level 1 PI request; no software runs at this high-priority PI level, so the PI request generated by this switch will almost always be recognized, and can pass control to EDDT.

One idea is to have the APR try and read and discard the instruction pointed to by PC before it halts, the idea being that this will allow a front panel, snooping the system bus, to capture the PC and the instruction word.

I/O Instructions

The PDP-10/X APR uses I/O instructions with D[03..05] = 000 act upon I/O devices, located internal to the APR, that deal with traps and PI requests.

CONI, APR, E

7 0 0 2 4				I	X	Y				
0				1	1	1		1	1	3
0				2	3	4		7	8	5

Read the status of the APR into location E. The word read has the following format.

E	E	E	E	E	E	E	T	T	T	T	T	T	T		F	F	F	F	F	F	E	T		
H	S	E	E	U	U	I	H	S	E	E	U	U	I		H	S	E	E	U	U	I	I		
E	E	2	1	2	1	A	E	E	2	1	2	1	A		E	E	2	1	2	1	R	R		
0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	2	2	3
0	1	2	3	4	5	6	8	9	0	1	2	3	4	5	7	8	0	0	0	0	0	4	5	6
																								3
																								5

EHE Error interrupt enable for the hard error flag.

ESE Error interrupt enable for the soft error flag.

EE2 Error interrupt enable for the executive mode trap-2 flag.

EE1 Error interrupt enable for the executive mode trap-1 flag.

EU2 Error interrupt enable for the user mode trap-2 flag.

EU1 Error interrupt enable for the user mode trap-1 flag.

EIA The PI assignment for the error interrupt. If EIA is 0 the error interrupt is not connected to the PI system.

THE Trap interrupt enable for the hard error flag.

TSE Trap interrupt enable for the soft error flag.

- TE2 Trap interrupt enable for the executive mode trap-2 flag.
- TE1 Trap interrupt enable for the executive mode trap-1 flag.
- TU2 Trap interrupt enable for the user mode trap-2 flag.
- TU1 Trap interrupt enable for the user mode trap-1 flag.
- TIA The PI assignment for the trap interrupt. If EIA is 0 the trap interrupt is not connected to the PI system.
- FHE The hard error flag.
- FSE The soft error flag.
- FE2 The executive mode trap-2 flag.
- FE1 The executive mode trap-1 flag.
- FU2 The user mode trap-2 flag.
- FU1 The user mode trap-1 flag.
- EIR Indicates that an error interrupt is pending, even if the error interrupt is not connected to the PI system because EIA is 0.
- TIR Indicates that a trap interrupt is pending, even if the trap interrupt is not connected to the PI system because TIA is 0.

CONO APR,E

7 0 0 2 0	I	X	Y
0	1 1 1	1 1	3
0	2 3 4	7 8	5

Perform the function specified by the effective conditions E (an immediate quantity).

	S	R	C	S	C	L	L	M	M	M	M	M	M	M	M	M	I
	E	O	E	F	F	E	T	H	S	E	E	J	U				A
0	1 1 1	2 2 2	2 2 2	2 2 2	2 2 2	2 2 2	2 2 2	2 2 2	2 2 2	2 2 2	2 2 2	3 3 3	3 3 3	3 3 3	3 3 3	3 3 3	3
0	7 8 9	0 1 2	3 4 5	6 7 8	9 0 1	2 3 4	5 6 7	8 9 0	1 2 3	4 5 6	7 8 9	0 1 2	3 4 5	6 7 8	9 0 1	2 3 4	5

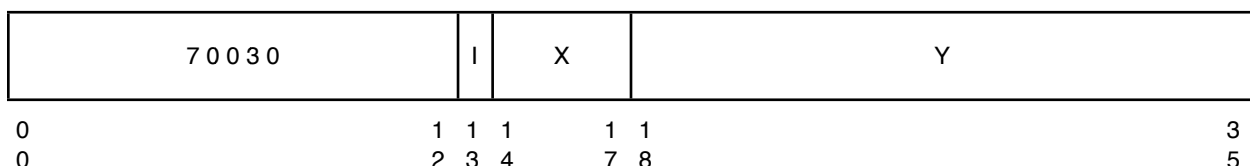
SSE Set the soft error flag.

RIO	Reset the I/O system. This does not effect anything associated with device 0, which means it does not reset anything in the APR or the PAG device.
CSE	Clear the soft error flag.
SF	Set the flags corresponding to the bits that are set in the mask (bits [25..30]).
CF	Clear the flags corresponding to the bits that are set in the mask (bits [25..30]).
LE	Load the error interrupt enables from the mask (bits [25..30]) and load the error interrupt PI assignment from the IA field (bits[33..35]).
LT	Load the trap interrupt enables from the mask (bits [25..30]) and load the trap interrupt PI assignment from the IA field (bits[33..35]).
MHE	Mask bit for hard error enables and/or flags.
MSE	Mask bit for soft error enables and/or flags.
ME2	Mask bit for executive mode trap-2 enables and/or flags.
ME1	Mask bit for executive mode trap-1 enables and/or flags.
MU2	Mask bit for user mode trap-2 enables and/or flags.
MU1	Mask bit for user mode trap-1 enables and/or flags.
IA	The value that can be loaded into the PI assignments for the error and/or trap interrupts.

The effect of a CONO APR that attempts to both set and clear a flag is undefined; the current design gives setting a flag priority over clearing a flag.

The SE bit is treated specially (that is, there is a SSE bit and a CSE bit) to allow a single CONO APR (the one in the context switch) to both load the trap enables for a job and set the SE flag should a high-priority PI routine wish to cause a fake soft error because an XCTRI detected an error.

CONSZ APR,E



Read the status of the APR (a word with the same format as described for CONI), mask it with the effective address E (an immediate quantity), and skip if the result is zero.

CONSO APR,E

7 0 0 3 4	I	X	Y			
0	1	1	1	1	1	3
0	2	3	4	7	8	5

Read the status of the APR (a word with the same format as described for CONI), mask it with the effective address E (an immediate quantity), and skip if the result is non-zero.

CONI PI,E

7 0 0 6 4	I	X	Y			
0	1	1	1	1	1	3
0	2	3	4	7	8	5

Read the status of the PI system into location E. The word read has the following format.

	S	S	S	S	S	S	S	S		I	I	I	I	I	I	I	G	L	L	L	L	L	L	L
	R	R	R	R	R	R	R	R		P	P	P	P	P	P	P	E	E	E	E	E	E	E	E
	1	2	3	4	5	6	7			1	2	3	4	5	6	7		1	2	3	4	5	6	7
0	1	1	1	1	1	1	1	1	2	0	0	0	2	2	2	2	2	2	3	3	3	3	3	3
0	0	1	2	3	4	5	6	7	8	0	0	0	0	4	5	6	7	8	9	0	1	2	3	5

- SR1-7 Software request flags for levels 1-7.
- IP1-7 In-progress flags for levels 1-7. The in-progress flag for level n is set when an interrupt request is accepted on level n, and the highest priority (lowest n) in-progress flag is reset by JRST 10 or JRST 12.
- GE The global enable flag.
- LE1-7 Enable flags for levels 1-7.

At reset all of the software request flags, all of the in-progress flags, the global enable flag, and all of the enable flags are set to 0.

An SRn bit makes a request on a level even if the LEn bit for the level is clear; only GE needs to be set. This is compatible with the last models of the PDP-10 running the newest microcode, but is incompatible with the KA10, which required that the LEn bit be set. ITS always sets the LEn bits.

An SRn bit remains set until it is cleared by software. This is compatible with all models of the PDP-10 except the KA10, which cleared SRn upon delivery of a PI at level n. ITS conditionalizes the explicit clearing of SRn bits on a model-by-model basis.

CONO PI,E

7 0 0 6 0	I	X	Y
0	1 1 1	1 1	3
0	2 3 4	7 8	5

Perform the function specified by the effective conditions E (an immediate quantity).

	C	R	S	S	C	C	S	L	L	L	L	L	L	L	
	S	P	S	L	L	G	G	1	2	3	4	5	6	7	
	R	I	R	E	E	E	E								
0	2	2	2	2	2	2	2	2	3	3	3	3	3	3	
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5

- CSR Clear the software request(s) corresponding to the bit(s) that are set in the mask (bits [29..35]).
- RPI Reset the PI system. All of the software request bits, all of the in-progress bits, the global enable bit, and all of the level enable bits are cleared.
- SSR Set the software request(s) corresponding to the bit(s) that are set in the mask (bits [29..35]).
- SLE Set the level enable bit(s) corresponding to the bit(s) that are set in the mask (bits [29..35]).
- CLE Clear the level enable bit(s) corresponding to the bit(s) that are set in the mask (bits [29..35]).
- CGE Clear the global enable.
- SGE Set the global enable.
- L1-7 Mask bits for levels 1-7.

The effect of a CONO PI that attempts to both set and clear a flag is undefined; the current design gives setting a flag priority over clearing a flag.

CONSZ PI,E

7 0 0 7 0	I	X	Y
-----------	---	---	---

0		1 1 1	1 1		3
0		2 3 4	7 8		5

Read the status of the PI system (a word with the same format as described in CONI PI), mask it with the effective address E (an immediate quantity), and skip if the result is zero.

CONSO PI,E

7 0 0 7 4	I	X	Y
-----------	---	---	---

0		1 1 1	1 1		3
0		2 3 4	7 8		5

Read the status of the PI system (a word with the same format as described in CONI PI), mask it with the effective address E (an immediate quantity), and skip if the result is non-zero.

PAG

The PAG block is the pager and the memory controller. It responds to all memory cycles on the system bus, translating virtual memory addresses into physical memory addresses in a way that is appropriate for ITS if necessary.

The Paging Process

Programs generate 18-bit addresses.

When paging is disabled programs can only reference some (usually the lowest) 256KW of the 4096KW physical address space.

When paging is enabled programs can reference up to 256K words of virtual address space, consisting of up to 256 1K word virtual pages, each mapped to any of the 4096 1K word physical pages in the 4096K word physical address space.

The mapping of virtual pages to physical pages is performed by page tables located in physical memory. Each page table is 64 words long, and contains mappings for 128 virtual pages. There are four page tables; two page tables map the low and high 128K words of the executive mode virtual address space, and two page tables map the low and high 128K words of the user mode virtual address space.

The page tables are pointed to by four page table base registers. Page tables are aligned on 64-word boundaries in the physical address space; this causes no trouble for ITS (which stores page tables in the front of the user variable block, which is easily aligned, since on the KL10/KS10 it also served as the UPT, which had alignment requirements), and eliminates the need for an adder in a critical path.

A page table word has the following format.

EVEN PAGE (A[10]=0)							ODD PAGE (A[10]=1)							
P		A					P		A					
0	0	0	0	0	0	0	1	1	1	2	2	2	2	2
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4
														3
														5

The P field is the protection code. No access is allowed if the protection code is 0. Read access is allowed if the protection code is 1, 2, or 3, Write access is allowed if the protection code is 3. Read-only pages normally have protection code 1; protection code 2, usually called "read/write first", is used to trap the first write to pages that would normally have protection code 3.

The A field is the age flag. It does not participate in translation or protection, but it is set to 0 when the page table entry is used by the pager to map an address (when the pager

sets the A bit to 0 is disturbs no other bits in the page table entry, including bits [02..03] and bit [05]).

The PPN field is the physical page number. The PPN field is 12 bits wide, so there can be 4096 physical pages, which is enough to address up to 4096KW of memory.

PAG contains two 256-entry RAMS which cache the most recently fetched page table entry for each of the virtual pages in each of the address spaces. The entry in the RAM corresponding to the virtual page is loaded any time a page table entry is fetched from a page table. When a virtual to physical translation is needed, PAG first looks in the RAM to see if it contains a page table entry that can satisfy the translation. If it does then the memory access proceeds. If it does not (either because the entry was not valid, or because the protection check failed) the page table lookup process begins.

Software can invalidate individual entries and/or all 256 entries in either or both of the page table entry cache RAMS using the DATAO PAG+1 instruction. In addition, any time a page table base register is updated using a DATAO PAG+4, DATAO PAG+5, DATAO PAG+6, or DATAO PAG+7 instruction all 256 entries in the RAM associated with the virtual address space mapped by the updated page table base are invalidated.

Note that because the page table lookup process is begun anytime the page table entry cache cannot satisfy the request, it is not necessary to invalidate an entry in the page table entry cache if a page table entry is changed from invalid to valid, or from a state which does not allow writing into a state that does allow writing. It is only necessary to invalidate an entry in the page table entry cache when a page table entry is switches to a more restrictive protection code.

Bulk invalidations of the page table entry cache are performed in a way that minimizes the impact on the system. When a page table entry cache RAM needs to be bulk invalidated, it is placed into the "invalidating" state. In this state the cache is invalidating one entry per cycle, so the whole RAM is invalidated in 256 cycles. If software makes a request that requires the use of the RAM while it is invalidating the request is processed via a full page table lookup, and the page table entry is not written into the RAM. Instructions continue to be executed during the invalidation, but memory operates at about half speed. Note also that accesses made to the executive mode address space are not stalled while the user mode page table entry cache RAM is invalidating, so when the operating system switches jobs, some of the invalidation time is overlapped with the executive mode code that is restoring the state of the job.

APR does not check write access for the memory operand of an instruction that both reads and writes memory until the write actually happens; a DIVM whose memory operand is 0 and in a location that cannot be written will get a "no divide" trap rather than an error from PAG on the failed write.

When PAG detects that a cycle generated by the currently executing instruction cannot be completed it captures some information about the cycle in a register within PAG, and

completes the read or write with a soft error acknowledge. The APR abandons the currently executing instruction without updating any state (so the PC is left pointing at the instruction that could not be completed). The soft error acknowledge normally sets the SE flag and generates a PI. If a PI is not generated the instruction is re-executed, which may result in a double soft error (which normally sets the HE flag and generates a PI) or a double hard error (which generates a halt).

I/O Instructions

The PAG block appears to be eight I/O devices starting at device address 0.10; in all of the following descriptions the symbol PAG has value 010.

CONI, PAG+0,E

7 0 4 2 4	I	X	Y
0	1 1 1	1 1	3
0	2 3 4	7 8	5

Read the status of PAG into location E. The word read has the following format.

	H	P	R				
	M	E	E				
0	2	2	2	3	3	3	3
0	7	8	9	0	1	2	3
							5

HM Highest moby. Determines which 256KW region of the physical address space referenced by reads/writes when the PE bit is 0. If the HM bit is 1 then the highest 256KW is referenced. If the HM bit is 0 then the lowest 256KW is referenced.

PE Paging enable. If the PE bit is 1 reads/writes reference (up to) 256KW of virtual address space directed to (up to) 256 1KW pages located anywhere in the 4096KW (512KW) physical address space. If the PE bit is 0 reads/writes reference 256KW of virtual address space redirected to the 256KW of the physical address space specified by the HM bit.

RE ROM enable. If the RE bit is 1 reads from virtual locations 400000-777777 reference 128 copies of the 1KW ROM, and writes to virtual locations 400000-777777 are ignored. If the RE bit is 0 virtual locations 400000-777777 receive no special treatment.

The normal "paging off" configuration has HM = 0, PE = 0, and RE = 0. The normal "paging on" configuration has PE = 1, and RE = 0. The console and bootstrapping ROM has HM = 1, PE = 0, and RE = 1/0.

Power-up reset sets HM to 1, PE to 0, and RE to 1. A software-generated I/O reset (caused by CONO APR, 200000) does not effect HM, PE, or RE.

The layout of the bits in this register anticipates adding a PAG interrupt at some point in the future; bit [31] would be the IE flag, bit [32] would be the IR flag, and bits [33..35] would be the PI level assignment.

CONO, PAG+0,E

7 0 4 2 0	I	X	Y
0	1 1 1	1 1	3
0	2 3 4	7 8	5

Set up the PAG device from the effective conditions E as shown.

	H	P	R				
	M	E	E				
0	2	2	2	3	3	3	3
0	7	8	9	0	1	2	3
							5

HM Highest moby.

PE Paging enable.

RE ROM enable.

The layout of the bits in this register anticipates adding a PAG interrupt at some point in the future; bit [31] would be the IE flag, bit [32] would be unused, and bits [33..35] would be the PI level assignment.

CONSZ PAG+0,E

7 0 4 3 0	I	X	Y
0	1 1 1	1 1	3
0	2 3 4	7 8	5

Read the status of PAG (a word with the same format as described in CONI PAG), mask it with the effective address E (an immediate quantity), and skip if the result is zero.

CONSO PAG+0,E

7 0 4 3 4	I	X	Y
-----------	---	---	---

0		1 1 1	1 1	3
0		2 3 4	7 8	5

Read the status of PAG (a word with the same format as described in CONI PAG), mask it with the effective address E (an immediate quantity), and skip if the result is non-zero.

DATAI PAG+0,E

7 0 4 0 4	I	X	Y
-----------	---	---	---

0		1 1 1	1 1	3
0		2 3 4	7 8	5

DATAO PAG+0,E

7 0 4 1 4	I	X	Y
-----------	---	---	---

0		1 1 1	1 1	3
0		2 3 4	7 8	5

Read/write the register that describes the most recent soft error encountered by PAG. The register has the following format.

W R	U M	P			V A
--------	--------	---	--	--	--------

0	0	0	0	0	0	0		1	1	3
0	1	2	3	4	5	6		7	8	5

WR Set to 1 if the error happened on a write, and set to 0 if the error happened on a read.

UM Set to 1 if the error happened on a user mode read/write, and set to 0 if the error happened on an executive mode read/write. Executive mode programs can generate traps with UM = 1 with XCTR or XCTRI.

P The P field of the page table entry that was accessed and determined that the read/write should complete with a soft error.

VA The virtual address of the read/write that generated the error.

Bits [06..17] are unused, so there are enough bits for a full 30-bit VA in some future extended addressing design, even though it is unlikely that any such design would be done, since ITS does not understand extended addressing.

This register is updated on any error generated by PAG, even errors that are generated by an instruction under an XCTRI that causes the APR to skip instead of generating APR PI (it needs to work this way to allow the code at label INTPFL to be written). This register is not updated on bus errors, even if the bus error was generated by PAG; doing so would be useless because there are APR-detected bus errors.

This register is read/write to allow a high-priority PI routine that uses XCTRI instructions to save and restore this register, since ITS processes page failures in a low-priority interrupt handler (APRCHN = 7). This is actually how ITS works when using a Systems Concepts paging box, but it is not obvious that this is the case (the save and restore is a side effect of SPM and LPMR instructions executed by the PI routine).

Power-up reset sets UM to 0, WR to 0, P to 0, and VA to 0. A software-generated I/O reset (caused by CONO APR, 200000) does not effect UM, WR, P, or VA.

DATAO PAG+1,E

7 0 4 5 4				I	X	Y				
0				1	1	1		1	1	3
0				2	3	4		7	8	5

Invalidate entries in PAG's page table entry cache specified by the contents of location E. The word read from location E has the following format.

E	U	I					I			
I	I	S					A			
E	E									
0	0	0	0	0	0	0		1	1	3
0	1	2	3	4	5	6		7	8	5

EIE Set to 1 to enable invalidation of the executive mode virtual address space.

UIE Set to 1 to enable invalidation of the user mode virtual address space.

IS Set to 1 if this is an invalidate single (only the page table entry cache location that maps the virtual address specified by the IA field) and set to 0 if this is an invalidate all (all page table entry cache locations).

IA The address being invalidated if the IS field is set to 1.

The ITS CLRPGM macro (flush the entire page table entry cache) can be implemented with DATAO PAG+1,[600000,,000000].

DATAI PAG+3,E

7 0 5 4 4			I	X	Y		
0	1	1	1	1	1		3
0	2	3	4	7	8		5

DATAO PAG+3,E

7 0 5 5 4			I	X	Y		
0	1	1	1	1	1		3
0	2	3	4	7	8		5

Read/write the quantum timer, which is an 18-bit timer that increments every 4 microseconds (that is, by the 16 MHz system bus clock divided by 64) any time the APR is running at level 0 (that is, no PI is active, as indicated by the BPIA_L signal on the system bus). Nothing special happens when the quantum timer overflows (the quantum timer in the Systems Concepts ITS paging box for the KA10 generated an interrupt for this condition, but ITS did not use it, and actually went to considerable trouble to ensure that the interrupt did no happen).

The quantum timer in the Systems Concepts paging box was 19 bits wide and incremented at 1 MHz, but this size/rate was never really used; any time the value in the quantum timer was read it was immediately shifted right by 2 bits. The quantum timer in the PAG device increments at the desired rate, and as a side effect of doing so, is actually 1 bit wider than the one in the Systems Concepts paging box.

The word written/read has the following format.

			T M R		
0	1	1			3
0	7	8			5

Power-up reset sets TMR to 0. A software-generated I/O reset (caused by CONO APR, 200000) does not effect TMR.

The simulator does not increment the quantum timer by 1 every 4 microseconds, but by 2500 every 10 milliseconds.

DATAI PAG+4,E

7 0 6 0 4	I	X	Y
0	1 1 1	1 1	3
0	2 3 4	7 8	5

DATAO PAG+4,E

7 0 6 1 4	I	X	Y
0	1 1 1	1 1	3
0	2 3 4	7 8	5

Read/write the executive mode low segment base register (the base register ITS normally calls DBR4). The word written/read has the following format.

	E L B	
0	1 1	2 3 3
0	3 4	9 0 5

Note that bits [30..35] are unused, which means the 64-word page table must be 64-word aligned.

The 64-word alignment restriction saves an adder in the critical path to getting a physical address to the memory, and causes no trouble, since the only value ITS ever puts in this register is a constant.

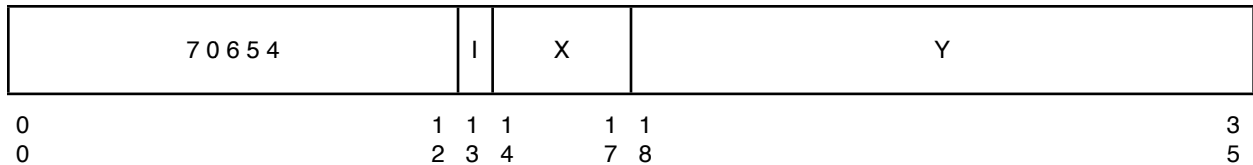
All 256 entries in the executive mode page table entry cache are invalidated any time the executive mode low segment base register is written.

Power-up reset sets ELB to 0. A software-generated I/O reset (caused by CONO APR, 200000) does not effect ELB.

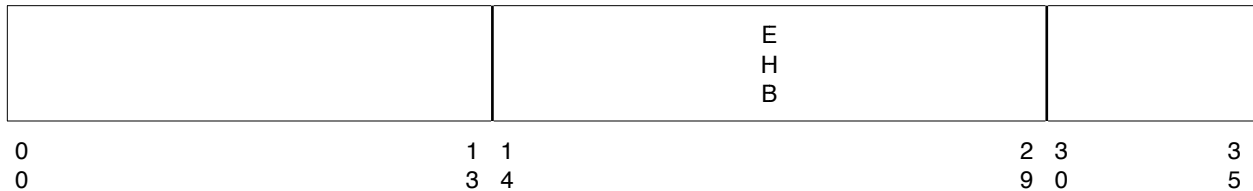
DATAI PAG+5,E

7 0 6 4 4	I	X	Y
0	1 1 1	1 1	3
0	2 3 4	7 8	5

DATAO PAG+5,E



Read/write the executive mode high segment base register (the base register ITS normally calls DBR3). The word written/read has the following format.



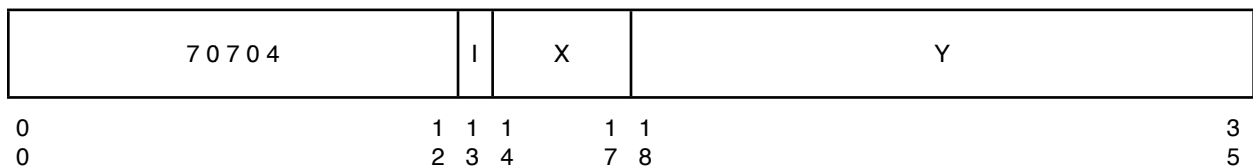
Note that bits [30..35] are unused, which means the 64-word page table must be 64-word aligned.

The 64-word alignment restriction saves an adder in the critical path to getting a physical address to the memory, and causes no trouble, since the only value ITS ever puts in this register is a constant.

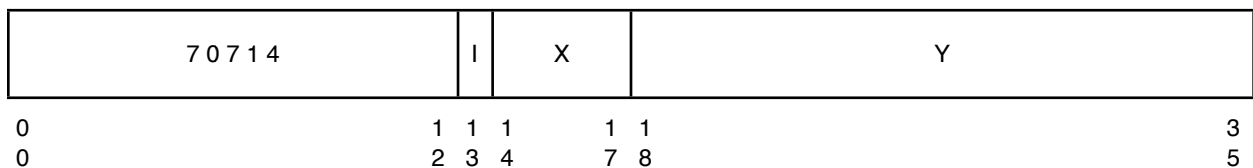
All 256 entries in the executive mode page table entry cache are invalidated any time the user mode high segment base register is written.

Power-up reset sets EHB to 0. A software-generated I/O reset (caused by CONO APR, 200000) does not effect EHB.

DATAI PAG+6,E



DATAO PAG+6,E



Read/write the user mode low segment base register (the base register ITS normally calls DBR1). The word written/read has the following format.

The 64-word alignment restriction saves an adder in the critical path to getting a physical address to the memory, and causes no trouble, since the only values ITS ever puts in this register are fixed offsets from the start of a job's variables (UPGCP and UPGMP+100), and it's easy to align the start of a job's variables (since the job's variables contain a UPT on some machines, and on some of these machines, the UPT needs to be aligned).

All 256 entries in the user mode page table entry cache are invalidated any time the user mode high segment base register is written.

Power-up reset sets UHB to 0. A software-generated I/O reset (caused by CONO APR, 200000) does not effect UHB.

RTC

The RTC device, which is part of the BIO block, is a periodic 60 Hz interrupt, a free-running timer that works in 60 Hz ticks, and a date/time clock with battery backup.

The free-running timer is a 36-bit register that increments at a 60 Hz rate if its value is not 0. By convention if the timer contains 0 it means the timer is not valid; a side effect of this definition is that the timer becomes invalid any time it overflows. By convention the timer holds the date/time, encoded as a number of 60 Hz ticks since 01-Jan-2001, 00:00:00, in the local standard (not daylight) time zone, for use by ITS.

The date/time clock is a Dallas Semiconductor DS1337, accessed over a bit-banged I2C bus. The DS1337 contains a date/time clock, with battery backup, that works in HH:MM:SS DD:MM:YY format. The date/time clock is read by the ROM, which uses it to initialize the free-running timer.

The RTC device generates the 60 Hz clock used by both the 60 Hz interrupt and by the timer by dividing a 1.8432 MHz clock (the same clock that is used by the TTY device to generate serial line bit-rate clocks) by 30720. A design that used the 60 Hz power line as the time base was rejected because it would not allow battery operation.

CONI, RTC,E

7 1 0 2 4				I	X	Y			
0		1	1	1		1	1		3
0		2	3	4		7	8		5

Read the status of RTC into location E. The word read has the following format.

							D	D	C	C	F	I	I	I	
							O	O	O	O	E	R	A		
0							2	2	2	2	2	3	3	3	3
0							5	6	7	8	9	0	1	2	3

- D The value that is on the SDA signal of the I2C bus, without regard for which device (RTC, DS1337, or both) is actually driving.
- DO The value that RTC is driving onto the SDA signal of the I2C bus. The RTC's driver is open-drain, so RTC drives the SDA signal to 0 when DO is 0, and does not drive the SDA signal when DO is 1.
- C The value that is on the SCK signal of the I2C bus, without regard for which device (RTC, DS1337, or both) is actually driving.

- CO The value that RTC is driving onto the SCK signal of the I2C bus. The RTC's driver is open-drain, so RTC drives the SCK signal to 0 when CO is 0, and does not drive the SCK signal when CO is 1.
- F The clock flag, set by the 60 Hz time base, and cleared by CONO RTC with the CF bit set.
- IE The PI enable.
- IR The PI request, set if F is 1 and IE is 1, even if the RTC device is not assigned to any PI level.
- IA The PI assignment.

Reset (which can be a power-up reset, or a software-generated I/O reset caused by a CONO APR, 200000) sets DO to 1, CO to 1, F to 0, IE to 0, and IA to 0. This causes D to be 1 and C to be 1 (nothing is driving the SDA and SCK signals, so they become 1 because of the I2C pull-up resistors) and IR to be 0 (since IE is 0). Because F is set every 1/60th of a second, observing that F is set to 0 by reset requires effort.

CONO RTC,E

7 1 0 2 0	I	X	Y
0	1 1 1	1 1	3
0	2 3 4	7 8	5

Set up the RTC device from the effective conditions E as shown.

	D O	C O	C F	I E	I A
0	2 2 2	2 2 3	3 3 3	3	3
0	5 6 7	8 9 0	1 2 3	5	

- DO The value that RTC should drive onto the SDA signal of the I2C bus. The RTC's driver is open-drain, so RTC drives the SDA signal to 0 when DO is 0, and does not drive the SDA signal when DO is 1.
- CO The value that RTC should drive onto the SCK signal of the I2C bus. The RTC's driver is open-drain, so RTC drives the SCK signal to 0 when CO is 0, and does not drive the SCK signal when CO is 1.
- CF The clear flag command; the flag is cleared by a CONO with CF set to 1.
- IE The PI enable.

IA The PI assignment.

CONSZ RTC,E

7 1 0 3 0	I	X	Y	
0	1	1 1	1 1	3
0	2	3 4	7 8	5

Read the status of the RTC (as described in CONI RTC), mask it with the effective address E (an immediate quantity), and skip if the result is zero.

CONSO RTC,E

7 1 0 3 4	I	X	Y	
0	1	1 1	1 1	3
0	2	3 4	7 8	5

Read the status of the RTC (as described in CONI RTC), mask it with the effective address E (an immediate quantity), and skip if the result is non-zero.

DATAI RTC,E

7 1 0 0 4	I	X	Y	
0	1	1 1	1 1	3
0	2	3 4	7 8	5

Read the free-running timer into memory location E.

Power-up reset sets the free-running timer to 0. Software-generated I/O reset (caused by CONO APR, 200000) does not reset the free-running timer; the date and time set by the ROM persists through any software-generated I/O reset.

DATAO RTC,E

7 1 0 1 4	I	X	Y	
0	1	1 1	1 1	3
0	2	3 4	7 8	5

Write memory location E into the free-running timer.

TTY

The TTY device, which is part of the BIO block, is a simple double-buffered full-duplex 2-wire UART intended for use as the console terminal.

The receiver and transmitter are hardwired to operate at 9600 bits/second (the 16x9600 Hz clock needed by the transmitter and receiver is generated by dividing a 1.8432 MHz crystal clock by 12), with eight data bits, with one stop bit, and with no parity. The configuration of the transmitter and receiver is hardwired, so bootstrap software can use the TTY device in a busy-wait style without any initialization.

CONI TTY,E

7 2 0 2 4	I	X	Y
0	1 1 1	1 1	3
0	2 3 4	7 8	5

Read the status of the TTY device into location E as shown.

	T B E	T I E	T I R	R B F	R I E	R I R	I A
0	2	2	2	2	3	3	3
0	6	7	8	9	0	1	2
					3	3	5

TBE The transmit buffer empty status, set if the transmit buffer register is not holding a character. Writing a character into the transmit side (with a DATAO TTY) clears this bit; the bit re-sets as soon as the character is moved from the transmit buffer register into the transmit shift register.

TIE The transmit PI enable.

TIR The transmit PI request, set if the TBE and TIE bits are set, even if the TTY device is not connected to any PI level (that is, if the IA field is 0).

RBF The receive buffer full status, set if the receive buffer register is holding a character. Reading a character from the receive side (with a DATAI TTY) clears this bit; the bit re-sets as soon as the next character is moved from the receive shift register into the receive buffer register.

RIE The receive PI enable.

RIR The receive PI request, set if the RBF and RIE bits are set, even if the TTY device is not connected to any PI level (that is, if the IA field is 0).

7 2 0 0 4	I	X	Y
0	1 1 1	1 1	3
0	2 3 4	7 8	5

Read the TTY receive buffer register into bits [28..35] of memory location E. Bits [00..27] are unpredictable. Clear the RBF bit in the TTY status.

This instruction does not check for a possible soft error on location E before it reads the TTY receive buffer register, so if there is a soft error data can be lost. This is not a problem if E specifies an accumulator or a location in wired-down memory, which is expected to always be the case.

DATAO TTY,E

7 2 0 1 4	I	X	Y
0	1 1 1	1 1	3
0	2 3 4	7 8	5

Write bits [28..35] of memory location E to the TTY transmit buffer register. Bits [00..27] of memory location E are ignored. Clear the TBE bit in the TTY status. Restart the transmitter if it is not running.

This definition of status bits and interrupt enables has been designed to work well with ITS terminal service. Receive is initialized by a CONO TTY, RIE, which sets RIE. Any time a character becomes available RBF becomes set, which immediately generates a PI, and the PI routine does a DATAI TTY, which reads the character and clears RBF. Transmit is initialized by the same CONO TTY, RIE, which clears TIE. Transmit is started by a CONO TTY, TIE+RIE executed from the TTYST (signal that output is available on a previously idle TTY) table, which immediately generates a PI. The transmit side is continued by a DATAO TTY executed from the TTYDO (output one character) table, and stopped by a CONO TTY, RIE, which clears TIE, executed from the TTYDFF (tell TTY to stop interrupting) table.

DSK

The DSK device, which is part of the BIO block, provides an interface to a single string of ATA-2 disks (one or two disks).

The DSK device always uses programmed I/O; this reduces performance on disks, but allows a number of disk-like devices that implement a subset of the ATA-2 interface that does not include DMA, like Compact Flash cards, to also be used (this means the ATA-2 DMARQ signal is ignored, and the ATA-2 DMACK signal is always driven false).

Nothing prevents a future version of DSK from implementing DMA on disks and disk-like devices that support it, and there is lots of free space in the BIO FPGA. It is expected that the DMA registers for DSK (D[03..05] = 100) would have D[03..05] = 101.

The DSK device always uses mode 0 (8-bit) and mode 2 (16-bit) PIO cycles, and assumes that the disk never needs to delay a cycle, which means the IORDY signal can be ignored. The DSK device does not support ancient ATA-2 disks, and assumes that the disk supports true 16-bit accesses to its data register, which means the IOCS16 signal can be ignored.

The INTRQ signal is pulled down by a 10K resistor; this ensures that INTRQ is false when the interrupt is “disabled” by the IEN bit in the device control register (when the ATA-2 specification talks about enabling and disabling interrupts it means enabling and disabling the tri-state buffer driving the INTRQ signal).

CONI DSK,E

7 4 0 2 4	I	X	Y
0	1 1 1	1 1	3
0	2 3 4	7 8	5

Read the status of the DSK device into location E as shown.

	I R Q	I E	I R	I A
0	2	3	3	3
0	9	0	1	2
	3	3	3	5

IRQ The raw ATA-2 interrupt request, set if the INTRQ signal on the ATA-2 bus is true, even if the DSK device does not have the PI enabled (that is, if the IE bit is 0) or is not connected to any PI level (that is, if the IA field is 0).

IE The PI enable.

IR The PI request, set if the IRQ bit is set and the IE bit is set, even if the DSK device is not connected to any PI level (that is, if the IA field is 0).

IA The PI assignment.

Reset (which can be a power-up reset, or a software-generated I/O reset caused by a CONO APR, 200000) sets IE to 0, and IA to 0. This forces IR to 0 (because IE is 0) and IRQ to 0 (because reset implies an ATA-2 reset, which causes the disk to tri-state the INTRQ signal, which causes the INTRQ signal to get pulled false by the 10K resistor).

CONO DSK,E

7 4 0 2 0	I	X	Y
0	1 1 1	1 1	3
0	2 3 4	7 8	5

Set up the DSK device from the effective conditions E as shown.

	I	E	I	A
0	2 3	3 3 3	3	3
0	9 0	1 2 3	5	

IE The PI enable.

IA The PI assignment.

CONSZ DSK,E

7 4 0 3 0	I	X	Y
0	1 1 1	1 1	3
0	2 3 4	7 8	5

Read the status of the DSK device (as described in CONI DSK), mask it with the effective address E (an immediate quantity), and skip if the result is zero.

CONSO DSK,E

7 4 0 3 4	I	X	Y
0	1 1 1	1 1	3
0	2 3 4	7 8	5

Read the status of the DSK device (as described in CONI DSK), mask it with the effective address E (an immediate quantity), and skip if the result is non-zero.

DATAI DSK+R,E

7 4 0 0 4 + (R << 3)	I	X	Y
0	1 1 1	1 1	3
0	2 3 4	7 8	5

Read the contents of the ATA-2 register specified by the R field into location E. Bits R[07..09] supply A[02..00] to the ATA-2 bus, and bit R[06] specifies if the register is in the command block (R[06] = 0) or the control block (R[06] = 1).

If the R field specifies the data register in the command block (R[06] = 0, R[07..09] = 0) then the DSK device reads the ATA-2 register four times, using a 16-bit PIO mode 2 read cycle, assembles the read data into a 36-bit word, and writes the word into location E. Each read of the ATA-2 register contributes 9 bits to the word (from ATA-2 D[08..00]), with the first read contributing bits [00..08] to the word, and the last read contributing bits [27..35] to the word.

If the R field specifies anything else then the interface reads the disk register once, using an 8-bit PIO mode 0 read cycle, and writes the read data into bits [28..35] of location E; bits [00..27] of location E are unpredictable.

This instruction does not check for a possible soft error on location E before it reads any ATA-2 register; if the ATA-2 register read has a side effect and there is a soft error data can be lost. This is not a problem if E specifies an accumulator or a location in wired-down memory, which is expected to always be the case.

DATAO DSK+R,E

7 4 0 1 4 + (R << 3)	I	X	Y
0	1 1 1	1 1	3
0	2 3 4	7 8	5

Write the contents of location E into the ATA-2 register specified by the R field. Bits R[07..09] supply A[02..00] to the ATA-2 bus, and bit R[06] specifies if the register is in the command block (R[06] = 0) or the control block (R[06] = 1).

If the R field specifies the data register in the command block (R[06] = 0, R[07..09] = 0) then the DSK device reads a 36-bit word from location E, disassembles the word into four 9-bit chunks, and writes each chunk to the ATA-2 bus using four 16-bit PIO mode 2 write cycles. Each 9-bit chunk supplies D[08..00] to the ATA-2 bus; D[15..09] are forced to 0. The first chunk comes from bits [00..08] of the word, and the fourth chunk comes from bits [27..35] of the word.

If the R field specifies anything else the DSK device reads a 36-bit word from location E, and writes bits [28..35] of the word to the ATA-2 bus using an 8-bit PIO mode 0 write cycle. The word supplied D[07..00] to the ATA-2 bus; D[15..08] are forced to 0.

ETH

The ETH device, which is part of the BIO block, provides an interface to a WIZnet WIZ830MJ 10/100-base-T embedded ethernet module.

The WIZ8300MJ module was selected because the WIZnet W5300 controller upon which it is based can be run in a mode where it is little more than an ethernet chip with 128KB of shared buffering (which can be used by operating systems) or in a mode where it contains a complete, albeit fairly simple, TCP/UDP/ICMP/IP stack (which can be used by a future version of the ROM that implements network booting and/or by a stand-alone program that allows network updates of the disk).

The ETH device runs the WIZ830MJ in indirect register mode and with an 8-bit data bus; the BIT16EN pin is connected to ground, the A[09..03] pins are connected to ground, and the D[15..08] pins are floating.

CONI ETH,E

7 6 0 2 4	I	X	Y
0	1 1 1	1 1	3
0	2 3 4	7 8	5

Read the status of the ETH device into location E as shown.

	I N T	I E	I R	I A
0	2	3	3	3
0	9	0	1	2
			3	5

INT The raw interrupt request, set if the interrupt request pin (/INT) pin of the WIZ830MJ is true (low), even if the ETH device does not have the PI enabled (that is, if the IE bit is 0) or is not connected to any PI level (that is, if the IA field is 0).

IE The PI enable.

IR The PI request, set if the INT bit is set and the IE bit is set, even if the ETH device is not connected to any PI level (that is, if the IA field is 0).

IA The PI assignment.

Reset (which can be a power-up reset, or a software-generated I/O reset caused by a CONO APR, 200000) sets IE to 0, sets IA to 0, resets the WIZ830MJ. Setting IE to 0 causes IR to be set to 0, and resetting the WIZ830MJ causes INT to be set to 0.

CONO ETH,E

7 6 0 2 0	I	X	Y
0	1	1 1	1 1 3
0	2	3 4	7 8 5

Set up the ETH device from the effective conditions E as shown.

	I	E	I	A
0	2	3 3	3 3 3	3
0	9	0 1	2 3	5

IE The PI enable.

IA The PI assignment.

CONSZ ETH,E

7 6 0 3 0	I	X	Y
0	1	1 1	1 1 3
0	2	3 4	7 8 5

Read the status of the ETH device (as described in CONI ETH), mask it with the effective address E (an immediate quantity), and skip if the result is zero.

CONSO ETH,E

7 6 0 3 4	I	X	Y
0	1	1 1	1 1 3
0	2	3 4	7 8 5

Read the status of the ETH device (as described in CONI ETH), mask it with the effective address E (an immediate quantity), and skip if the result is non-zero.

DATAI ETH+R,E

7 6 0 0 4 + (R << 3)	I	X	Y
0	1	1 1	1 1 3
0	2	3 4	7 8 5

Read the contents of the WIZ830MJ register specified by the R field into location E.

If R[06]=0 then a 16-bit value is read, as two 8-bit values, from the WIZ830MJ and written into bits [00..15] of location E, and bits [16..35] of location E are set to 0. If R[06]=1 then two 16-bit values are read, as four 8-bit values, from the WIZ830MJ and written into bits [00..15] (the first 16-bit value) and bits [16..31] (the second 16-bit value) of location E, and bits [32..35] of location E are set to 0.

If R[07]=0 then the 16-bit values are read from the WIZ830MJ in big-endian order (the most significant 8-bit value is read first, and the least significant 8-bit read is read second). If R[07]=1 then the 16-bit values are read from the WIZ830MJ in little-endian order (the least significant 8-bit value is read first, and the most significant 8-bit read is read second). Big-endian order is used when reading bytes and when reading 16-bit words in ARPANET order. Little-endian order is used when read 16-bit words in CHAOSNET order.

R[08..09] supplies A[02..01] to the WIZ830MJ. A[00] to the WIZ830MJ is sequenced automatically based on R[06..07].

This instruction does not check for a possible soft error on location E before it reads any WIZ830MJ register; if the WIZ830MJ register read has side effects and there is a soft error data can be lost. This is not a problem if E specifies an accumulator or a location in wired-down memory, which is expected to always be the case.

DATAO ETH+R,E

7 6 0 1 4 + (R << 3)	I	X	Y
0	1 1 1	1 1	3
0	2 3 4	7 8	5

Write the contents of location E into the WIZ830MJ register specified by the R field.

If R[06]=0 then the 16-bit value in bits [00..15] of location E is written, as two 8-bit values, into the WIZ830MJ. If R[06]=1 then two 16-bit values, the first one in bits [00..15] of location E, and the second one in bits [16..31] of location E, is written, as four 8-bit values, into the WIZ830MJ.

If R[07]=0 then the 16-bit values are written into the WIZ830MJ in big-endian order (the most significant 8-bit value is written first, and the least significant 8-bit value is written second). If R[07]=1 then the 16-bit values are written into the WIZ830MJ in little-endian order (the least significant 8-bit value is written first, and the most significant 8-bit value is written second). Big-endian order is used when writing bytes and when writing 16-bit words in ARPANET order. Little-endian order is used when writing 16-bit words in CHAOSNET order.

R[08..09] supplies A[02..01] to the WIZ830MJ. A[00] to the WIZ830MJ is sequenced automatically based on R[06..07].

ROM

The PDP-10/X does not have a tradition switches-and-lights front panel, nor does it have a second small (front end) processor that implements some complicated console interface. Instead, it has a 1KW ROM and some simple logic in PAG that allows the initial instructions executed by APR to come from the ROM, and implements its bootstrap and console functions (mainly a loader for system images) as a PDP-10 program.

Implementing bootstrap and console functions as a PDP-10 program is more difficult than it first appears. When ITS is read from the disk it is loaded into several disjoint regions of the lowest 256KW of memory; ITS at the low end, NSALV in the middle, and DDT at the high end. It is difficult to find a place in the lowest 256KW to hide the bootstrap and console program while it is executing. The PDP-10/X solves this problem with some simple logic in PAG that allows the console program to run out of the highest 256KW of memory, but be able to load ITS into the lowest 256KW of memory; there are no conflicts between the bootstrap and console program and any program that would normally run with paging disabled.

When PAG is reset PE is 0, RE is 1, and HM is 1. Since PE is 0 paging is disabled, but since RE is 1 and HM is 1 addresses in the range 000000 to 377777 reference the lowest 128KW of the highest 256KW of memory, and addresses in the range 400000 to 777777 reference 128 copies of the 1KW ROM. When APR is reset FL is 0 and PC is 400000, so the first instruction executed comes from the first word of the ROM.

The ROM is written so that it runs at addresses 000000 to 001777. The first few instructions of the ROM, which are executed out-of-place, copy the ROM image from the high 128KW of the address space to the low 128KW of the address space, then jump into the low 128KW of the address space. Once control is in the low 128KW of the address space the ROM is no longer needed, so RE is set to 0.

```
LOC    0
HRLZI  A,400020          ; COPY 400020-401777
HRRI   A,20              ; INTO 000020-001777
BLT    A,1777
JRST   L20

LOC    20
L20:   CONO  PAG+0,HM    ; SET RE=0, TURN OFF ROM
```

The ROM command language can be very simple, because its only function is to load standalone programs into the lowest 256KW of memory and jump to them. There are commands to read and write device registers and/or locations in the lowest 256KW of memory, but these are intended to be used only in desperation; it is expected that serious debugging will be done using a standalone version of DDT, loaded into memory by the ROM with the program being debugged.

The ROM commands are as follows.

B [/N:v]/[P:dpn] [bpn] Boot. Load the contents of the boot partition specified by “bpn”, which defaults to the first boot partition in the disk’s partition table, into the low 256KW of memory, and jump to the starting address of the boot partition.

The /N option, if present, specifies the boot flags; if there is no /N option then the boot flags are 0.

The /P option, if present, specifies the name of the associated disk partition; if there is no /P option then the name of the associated disk partition defaults to the name of the boot partition.

D/[C]/[D]/[N:v] a d Deposit.

E/[C]/[D]/[N:v] a Examine.

I Initialize the I/O system.

L bpn Load boot partition.

R Read a binary file from the terminal and load it into the specified locations in the low 256KW of memory. The file is essentially a DECSAV file (in that load data consists of blocks beginning by a “-n,,a-1” word) but it ends with a single JRST word, rather than with two JRST words.

S/[P:dpn]/[N:v] [sa] Start.

Z Zero memory.

The ROM sends jumps to the standalone program by copying a trampoline program into the accumulators (a CONO PAG+0,0 to set HM=0, and a JRST to the start address of the standalone program) and then jumping into the accumulators. When the first instruction of the standalone program some useful information is in accumulators 1-4.

- 1 The boot flags. Specified by the /N option on the B or S command, or 0 if there was no /N option on the B or S command.
- 2 Unused. Always 0.
- 3 The base of the associated disk partition, in cylinders, or 0 if the ROM was unable to find the associated disk partition.

- 4 The size of the associated disk partition, in cylinders, or 0 if the ROM was unable to find the associated disk partition.

The starting address of a boot partition containing ITS (or, to be more precise, a copy of ITS, NSALV, and DDT) points to the label “BEG” in ITS. The following code is located at the label “BEG” in ITS.

```
BEG:  MOVEM  A,BINFO+0          ; SAVE USEFUL INFO
      MOVEM  B,BINFO+1
      MOVEM  C,BINFO+2
      MOVEM  D,BINFO+3

      TRNN  D,1                ; [35]=1 => STOP BOOT AT DDT
      JRST  BEG1
      MOVEI  T,[0]             ; SET EMPTY MACRO COMMAND STRING
      HRLI  T,440700
      MOVEM  T,MACCR
      MOVEI  T,BEG1            ; SET <ALT>P ADDRESS
      MOVEM  T,@DDT-6
      JRST  DDT

BEG1:  MOVE  T,BINFO+3          ; CHECK SIZE OF DISK
      CAIE  T,NCYLS+XCYLS
      BUG   AWFUL,[DISK IS WRONG SIZE]

      SKIPN SALV                ; CHECK IF NSALV IS PESENT
      BUG   AWFUL,[NO NSALV]
      JSR   SALV+1              ; NSALV, NO QUESTIONS

BEG2:  ...
```

By default the system will run NSALV in no-questions mode and start up the system. If the system is booted with bit [35] set to 1 the system will enter DDT immediately after it boots, and if a “\$P” command is given in DDT, the system will proceed to run NSALV in no-questions mode and start up the system. If, for some reason, one wants to bring up the system without running NSALV then the system should be booted with bit [35] set to 1, and then started at label “BEG2” explicitly from DDT.

DISK LAYOUT

PDP-10 software is often aware of the physical geometry of the disk. The DSK uses ATA-2 disks run in LBA (logical block address) mode, so that the actual physical geometry of the disk is hidden, and uses standardized artificial geometry to satisfy any PDP-10 software that actually cares.

The DSK packs 64 PDP-10 words into each 512-byte disk block. This means that the natural “sector” of the DSK is 64 words long, which is half the size of the natural sector on all DEC disks (128 words, or 576 bytes). This causes no problems for ITS, which always works in 1K-word blocks, but might cause trouble for software that works in 128-word blocks and assumes that the size of a block and the size of a sector are the same.

Each “track” of the DSK is assumed to contain 64 sectors. Each track, therefore, contains $64 \times 64 = 4096$ words. This is the same track size as the DEC RMxx disks (the RM02, RM03, RM05, and RM80 disks all used 128-word sectors and had 32 sectors per track).

Each “cylinder” of the DSK is assumed to contain 16 tracks. Each cylinder, therefore, contains $4096 \times 16 = 65536$ words. This is a slightly larger cylinder size than the RM80 (which had 14 tracks per cylinder) and a slightly smaller cylinder size than the RM05 (which had 19 tracks per cylinder).

The fact that both the sectors/track and tracks/cylinder are powers of two means that a cylinder can be divided exactly into 1K-word ITS blocks; each cylinder contains 1024 sectors, and each 1K-word ITS block is 16 sectors, so there are 64 ITS blocks per cylinder. On some DEC disks (this is not true, and space is wasted).

Modern ATA-2 disks are much larger than any PDP-10 disk. To allow a single modern disk to serve as multiple smaller PDP-10 disks each physical disk is partitioned into one or more smaller logical disks, with each logical disk assigned some number of (65536 word) cylinders. The locations and sizes of the partitions is kept in a partition table located in the very first sector of the physical disk. In addition to holding the locations and sizes of the logical disk partitions, the partition table holds the locations and sizes of 256K-word boot images and space on the physical disk that is not allocated.

The partition table has the following format.

4 4 6 3 5 3	0
DISK SIZE (CYLINDERS)	
0	

	TYPE CODE
SIZE (CYLINDERS)	
VARIES BY TYPE CODE	
VARIES BY TYPE CODE	

CHECKSUM	

0	1 1	3
0	7 8	5

The partition table begins with a three word header. Word 0 of the header contains an identifying tag (the letters DSK in SIXBIT) in its left half, and a partition table version number (currently 0) in its right half. Word 1 of the header contains the size of the disk, in (65536 word) cylinders. Word 2 of the header is unused in a version 0 partition table, and is always 0.

Immediately following the three-word header are fifteen four-word entries. The right half of word 0 of each entry is always a type code. Word 1 of each entry is always the size of the piece of the disk described by the entry, in (65536 word) cylinders.

The first entry is always a “partition table” entry (reserving the cylinder whose first sector contains the partition table). Following the partition table are any number of “free space” entries, “boot partition” entries, and “disk partition” entries. Following those entries are enough “unused” entries to fill up the table. The partition table entries must describe the entire disk, including any free space at the end.

The partition table ends with word containing a checksum of all of the previous words in the partition table. The SBLK algorithm is used. The following code fragment computes the checksum of words BUF+0 to BUF+62, and writes the result into word BUF+63.

```

WRC:  MOVE A, [-63.,,BUF]
      MOVEI B,0
WRC1: ROT B,1
      ADD B,(A)
      AOBJN A,WRC1
      MOVEM B,(A)
      POPJ P,

```

Unused entries, with a type of 0, mark entries in the partition table that are not used for something else. They are only allowed to appear at the end of the partition table, after the last entry that reserves any space on the disk.

			0
		0	
0		1 1	3
0		7 8	5

A partition table entry, with a type of 1, describes the cylinder that holds the partition table in its very first sector. The size is always 1.

OPCODES

The APR block implements the user mode instruction set of the KA10, including KA10-style double precision floating point.

000-077: LUUOs and MUUOs

	xx0	xx1	xx2	xx3	xx4	xx5	xx6	xx7
00x	MUO	LUO	LUO	LUO	LUO	LUO	LUO	LUO
01x	LUO	LUO	LUO	LUO	LUO	LUO	LUO	LUO
02x	LUO	LUO	LUO	LUO	LUO	LUO	LUO	LUO
03x	LUO	LUO	LUO	LUO	LUO	LUO	LUO	LUO
04x	MUO	MUO	MUO	MUO	MUO	MUO	MUO	MUO
05x	MUO	MUO	MUO	MUO	MUO	MUO	MUO	MUO
06x	MUO	MUO	MUO	MUO	MUO	MUO	MUO	MUO
07x	MUO	MUO	MUO	MUO	MUO	MUO	MUO	MUO

100-177: Byte Operations and Floating Point Operations

	xx0	xx1	xx2	xx3	xx4	xx5	xx6	xx7
10x	MUO	MUO	MUO (GFAD)	MUO (GFSB)	MUO (JSYS)	MUO (ADJSP)	MUO (GFMP)	MUO (GFDV)
11x	MUO (DFAD)	MUO (DFSB)	MUO (DFMP)	MUO (DFDV)	MUO (DADD)	MUO (DSUB)	MUO (DMUL)	MUO (DDIV)
12x	MUO (DMOVE)	MUO (DMOVN)	MUO (FIX)	MUO (EXTEND)	MUO (DMOVEM)	MUO (DMOVNM)	MUO (FIXR)	MUO (FLTR)
13x	UFA	DFN	FSC	IBP	ILDB	LDB	IDPB	DPB
14x	FAD	FADL	FADM	FADB	FADR	FADRI	FADRM	FADRB
15x	FSB	FSBL	FSBM	FSBB	FSBR	FSBRI	FSBRM	FSBRB
16x	FMP	FMPL	FMPM	FMPB	FMPR	FMPRI	FMPRM	FMPRB
17x	FDV	FDVL	FDVM	FDVB	FDVR	FDVRI	FDVRM	FDVRB

200-277: Moves and Integer Arithmetic Operations

	xx0	xx1	xx2	xx3	xx4	xx5	xx6	xx7
20x	MOVE	MOVEI	MOVEM	MOVES	MOVS	MOVSI	MOVSM	MOVSS
21x	MOVN	MOVNI	MOVNM	MOVNS	MOVMM	MOVMI	MOVMM	MOVMS
22x	IMUL	IMULI	IMULM	IMULB	MUL	MULI	MULM	MULB
23x	IDIV	IDIVI	IDIVM	IDIVB	DIV	DIVI	DIVM	DIVB
24x	ASH	ROT	LSH	JFFO	ASHC	ROTC	LSHC	MUO
25x	EXCH	BLT	AOBJP	AOBJN	JRST	JFCL	XCT	MUO (MAP)
26x	PUSHJ	PUSH	POP	POPJ	JSR	JSP	JSA	JRA
27x	ADD	ADDI	ADDM	ADDB	SUB	SUBI	SUBM	SUBB

300-377: Compares, Skips, and Jumps

	xx0	xx1	xx2	xx3	xx4	xx5	xx6	xx7
30x	CAI	CAIL	CAIE	CAILE	CAIA	CAIGE	CAIN	CAIG
31x	CAM	CAML	CAME	CAMLE	CAMA	CAMGE	CAMN	CAMG
32x	JUMP	JUMPL	JUMPE	JUMPLE	JUMPA	JUMPGE	JUMPN	JUMPG
33x	SKIP	SKIPL	SKIPE	SKIPLE	SKIPA	SKIPGE	SKIPN	SKIPG
34x	AOJ	AOJL	AOJE	AOJLE	AOJA	AOJGE	AOJN	AOJG
35x	AOS	AOSL	AOSE	AOSLE	AOSA	AOSGE	AOSN	AOSG
36x	SOJ	SOJL	SOJE	SOJLE	SOJA	SOJGE	SOJN	SOJG
37x	SOS	SOSL	SOSE	SOSLE	SOSA	SOSGE	SOSN	SOSG

400-477: Boolean Operations

	xx0	xx1	xx2	xx3	xx4	xx5	xx6	xx7
40x	SETZ	SETZI	SETZM	SETZB	AND	ANDI	ANDM	ANDB
41x	ANDCA	ANDCAI	ANDCAM	ANDCAB	SETM	SETMI	SETMM	SETMB
42x	ANDCM	ANDCMI	ANDCMM	ANDCMB	SETA	SETAI	SETAM	SETAB
43x	XOR	XORI	XORM	XORB	IOR	IORI	IORM	IORB
44x	ANDCB	ANDCBI	ANDCBM	ANDCBB	EQV	EQVI	EQVM	EQVB
45x	SETCA	SETCAI	SETCAM	SETCAB	ORCA	ORCAI	ORCAM	ORCAB
46x	SETCM	SETCMI	SETCMM	SETCMB	ORCM	ORCMI	ORCMM	ORCMB
47x	ORCB	ORCBI	ORCBM	ORCBB	SETO	SETOI	SETOM	SETOB

500-577: Halfword Operations

	57x	xx1	xx2	xx3	xx4	xx5	xx6	xx7
50x	HLL	HLLI	HLLM	HLLS	HRL	HRLI	HRLM	HRLS
51x	HLLZ	HLLZI	HLLZM	HLLZS	HRLZ	HRLZI	HRLZM	HRLZS
52x	HLLO	HLLOI	HLLM	HLLOS	HRLO	HRLOI	HLROM	HLROS
53x	HLLE	HLLEI	HLLEM	HLLES	HRLE	HRLEI	HLREM	HLRES
54x	HRR	HRRI	HRRM	HRRS	HLR	HLRI	HLRM	HLRS
55x	HRRZ	HRRZI	HRRZM	HRRZS	HLRZ	HLRZI	HLRZM	HLRZS
56x	HRRO	HRROI	HRROM	HRROS	HLRO	HLROI	HLROM	HLROS
57x	HRRE	HRREI	HRREM	HRRES	HLRE	HLREI	HLREM	HLRES

600-677: Bit Test Operations

	xx0	xx1	xx2	xx3	xx4	xx5	xx6	xx7
60x	TRN	TLN	TRNE	TLNE	TRNA	TLNA	TRNN	TLNN
61x	TDN	TSN	TDNE	TSNE	TDNA	TSNA	TDNN	TSNN
62x	TRZ	TLZ	TRZE	TLZE	TRZA	TLZA	TRZN	TLZN
63x	TDZ	TSZ	TDZE	TSZE	TDZA	TSZA	TDZN	TSZN
64x	TRC	TLC	TRCE	TLCE	TRCA	TLCA	TRCN	TLCN
65x	TDC	TSC	TDCE	TSCE	TDCA	TSCA	TDCN	TSCN
66x	TRO	TLO	TROE	TLOE	TROA	TLOA	TRON	TLON
67x	TDO	TSO	TDOE	TSOE	TDOA	TSOA	TDON	TSON

700-777: I/O Operations

	xx0	xx1	xx2	xx3	xx4	xx5	xx6	xx7
70x	IO-APR	IO-APR	IO-APR	IO-APR	IO-APR	IO-APR	IO-APR	IO-APR
71x	IO-RTC	IO-RTC	IO-RTC	IO-RTC	IO-RTC	IO-RTC	IO-RTC	IO-RTC
72x	IO-TTY	IO-TTY	IO-TTY	IO-TTY	IO-TTY	IO-TTY	IO-TTY	IO-TTY
73x	IO	IO	IO	IO	IO	IO	IO	IO
74x	IO-DSK	IO-DSK	IO-DSK	IO-DSK	IO-DSK	IO-DSK	IO-DSK	IO-DSK
75x	IO	IO	IO	IO	IO	IO	IO	IO
76x	IO	IO	IO	IO	IO	IO	IO	IO
77x	IO	IO	IO	IO	IO	IO	IO	IO

CHANGE HISTORY

- 12-Jan-2008 Initial version. Based on the text from the LaTeX version, but considerably reworked to include all of the thinking about how the APR device should work, how traps and errors should work, and so on. None of the hardware that has been designed so far conforms to this specification, although it's not wrong by much.
- 30-Jan-2008 I finally figured out the right way for the address mapping while the ROM is running to work (and I got to use the word "moby" in one of the canonical ways). Major changes to the section on PAG, and the section that describes how the ROM works.
- 02-Apr-2008 Minor revisions, mainly around the implementation of floating point, including KA-style double precision. Some reformatting of the opcode table to make it more readable.
- 21-May-2008 Changed the way the APR interrupts for bus and paging errors get generated. In the old design there were flags for bus error and paging error. In the new design there are flags for soft error and hard error, and a soft error is sometimes turned into a hard error. This works better because a soft error generated in a PI routine will get transformed into a hard error (and get into DDT) rather than halting.
- 01-Jun-2008 Changed the RTC device to include the 36-bit free-running timer.
- 08-Jun-2008 Changed the RTC device to so that the 36-bit free-running timer is reset by the power-on reset, not the I/O reset. This allows the ROM to read the date/time clock once, rather than having to teach ITS how to read it after every I/O reset.
- 23-Jun-2008 Changed the way APR interrupts work again, this time to make it easy to context switch the trap enables and (possibly) ask for a fake soft error (needed by code that uses XCTRI) with a single CONO.
- 14-Jul-2008 Changed all of the ASCII art for instructions and registers to use tables in a clever way. There are still some ugly page breaks, but overall the idea of using tables with no edge lines in some cases works.
- 17-Jul-2008 Eliminated previous mode. XCTR and XCTRI always redirect into user mode. This is what ITS really wanted. Removed a bunch of JRSTF instructions from ITS whose only purpose was to force PM=1.
- 27-Sep-2009 Switched to the DS1337 real-time clock. The DS1337 does not have the DS1338's 56 bytes of non-volatile memory, but does come in a

wider array of packaging options, including an 8-bit DIP option that is well suited to being used on a prototype.

20-Mar-2011 Added a new section to describe the ETH interface, including details of how the 16-bit and 32-bit packing and unpacking works.