KV10 Processor Reference Manual

David Bridgham
<dab@froghouse.org>

April 12, 2019

Abstract

This document describes the KV10 processor, an implementation of the DEC PDP-10 architecture. Much of the description is how it differs from Dave Conroy's PDP-10/X.

Contents

Chapter 1

Introduction

The KV10 is an implementation of the Digital Equipment Corporation's PDP-10 architecture, written in Verilog, and realized (thus far) on an FPGA. It mostly follows the PDP-10/X architecture, designed by Dave Conroy and described in his *PDP-10/X System Manual*[1], and, like Dave's design, is targeted at supporting the Incompatible Timesharing System (ITS) from MIT.

For now, this document is woefully incomplete. It contains a mix of things I've done, things I plan to do, and flights of fancy about where this project could go one day. It is slowly growing to describe more and more of the processor and may one day even delineate those three categories. For now, determining what's real and what's speculative is left as an exercise to the reader.

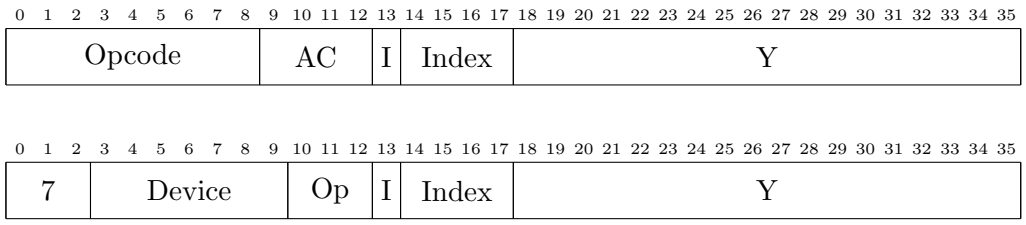


Figure 1.1: Normal and I/O Instruction Formats

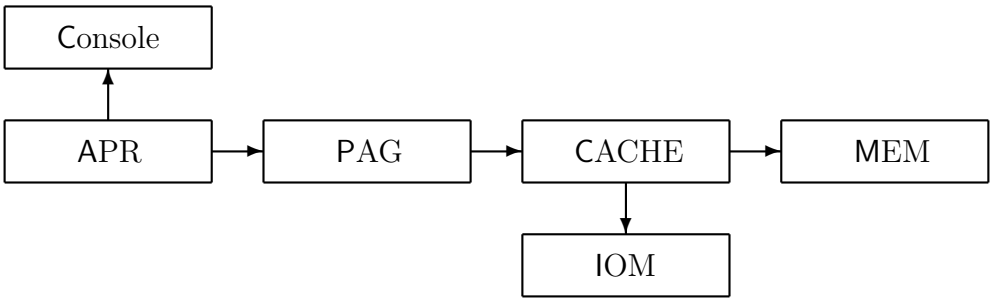


Figure 1.2: Internal modules of the KV10

1.1 Internal Modules

Arithmetic Processor – APR

The APR is the core of the processor, responsible for executing the instructions and maintaining the processor state.

Pager – PAG

The pager is modeled after the paging box that MIT designed and built to support ITS. It maps the 18-bit virtual addresses generated by the APR into 22-bit physical addresses and generates page faults on protection violations or missing memory pages.

Cache

The cache is a two-way, set associative, write-through cache. It also handles converting between the 36-bit words of the PDP-10 and the width of the memory which is likely not 36-bits.

Additionally, the cache is where the I/O bus is split out from the memory bus. Since the pager and cache have their own I/O devices, they grab their own device operations as they go by. Any that make it past the cache are directed to the IOM. When, in the future, we add DMA devices, they can communicate with the cache to either bypass it, to update it, or to invalidate entries as appropriate.

Memory Controller – MEM

The memory controller varies, depending on the memory technology in use such as SRAM, SDRAM, or MRAM. Because of variables like the memory width, mapping of that width into the cache's linelength, and optimizations like burst-mode, the memory controller and the cache are closely tied.

I/O Multiplexer – IOM

The I/O Multiplexer includes connections to all the rest of the I/O devices in the system. Simple devices may be implemented inside the IOM while more complex ones will get their own modules.

Console

The operator's console or front-panel was a staple of computers from the 1950s and 1960s. Using the console, an operator could control the computer, enter simple programs, or boot the machine. Front-panels have become passé, with good reason, but one is available for the KV10.[1]

1.2 On Names

1.2.1 Processor

This implementation of the PDP-10 architecture is called the KV10. As mentioned above, it was designed explicitly to run the ITS operating system, namely with a KA-like interrupt system[2] and the MIT Paging Box (also CIRC and one-proceed (yet to be implemented)). However, being implemented in Verilog, it will be fairly straightforward to have variations such as extra instructions, different paging hardware, and extended memory so it can run TOPS-20.

The exact combination of compile options as well as the version of the code used are identified by option codes following the name.

KV10A-FDICp-10-1.3

[1]The console should be optional but I'll have to figure out how to boot the machine without one.

[2]I'm considering adding some of the interrupt system from the KI10

model

A The base model. Essentially this implements the KA10 instruction set and interrupt architecture, a single set of accumulators, and so on. Really it's taken more from the PDP-10/X.

L The extended instructions of the KL10 processor, its interrupt architecture, and multiple accumulator sets. There are probably other differences but I haven't studied the KL10 very much. Anyway, if someone puts the work in to let the KV10 run TOPS-20, it will get a model designation.

options Options are listed in this order.

F Implemented on an FPGA.

V Implemented on an ASIC.

P Implemented as a printed IC.

D SDRAM controller.

1 DDR SDRAM controller.

2 DDR2 SDRAM controller.

3 DDR3 SDRAM controller.

4 DDR4 SDRAM controller.

I The ITS pager as described for the PDP-10/X.

C Implements the CIRC instruction.

p Implements One-Proceed.

U Unassigned Codes indirect through 60/61 instead of 40/41.

X Implements extended addressing.

cache

The size of the cache is indicated by log_2 of the number of words in the cache. For instance, 10 indicates a 1,024 word cache. 0 indicates no cache rather than one word.

version

The version number of the code as major.minor.

1.2.2 I/O Devices

I/O devices get their own names separately from the processor [to be described later]. Since many of the I/O devices are actually implemented along with the processor, it might make sense to have a list like the option list along with the processor name. I haven't decided yet.

1.2.3 Computer

Computers include a realization of the processor, some memory, plus a set of supported I/O devices and are named separately from all of those. The initial realization, only run a few times, was on a Terasic DE2-115 FPGA board and called Trilobyte. The current intention is to design and build a daughter card to go with the ZTEX 2.13 FPGA module providing a useful set of I/O ports. This will be called Ammonite.

Chapter 2

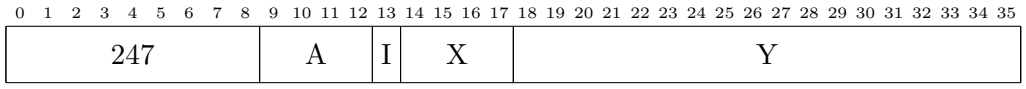
APR

2.1 Instructions

JRST

The JRST instruction works like on the KA10, with the four bits of the AC field enabling special functions. However, the microcode dispatches to 16 locations allowing for more variations. In particular, I'm considering adding a JRST that halts but leaves interrupts enabled like Wait for Interrupt from the PDP-11.

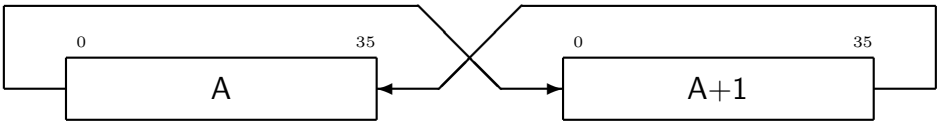
CIRC - Circulate



CIRC was added to the MIT KA10s in place of what was usually an unassigned code. This instruction rotates the two accumulators A and A+1 the number of positions specified by E very much like ROTC except it rotates A+1 in the opposite direction. If E is positive, bit 0 of A is rotated into bit 0 of A+1 and bit 35 of A+1 is rotated into bit 35 of A. If E is negative, bit 35 of A is rotated into bit 35 of A+1 and bit 0 of A+1 is rotated into bit 0 of A.

The DEC documentation specifies that E is taken modulo 256; it never has a magnitude greater than 255. The diagnostics make it clear that -256 is expected to move 256 positions.

CIRC A,36 will swap A and A+1 while reversing their bit order.



Inclusion of the CIRC instruction is controlled by the `CIRC` option in the `constants.vh`.

2.2 Interrupts and UUOs

2.2.1 Interrupt Instructions

On the KV10, any instruction may be used as an interrupt instruction though that doesn't mean any instruction makes sense there and most haven't been tested. There are three broad classes of instruction that have different effects when used as an interrupt instruction.

JSR, JSP, PUSHJ, JSA, or JRST These instructions will clear the CM flag (put the processor in exec mode) and jump to the destination address as per the instruction. JSR is the typical example of this and the most commonly used interrupt instruction. The interrupt service routine is expected to clear the condition causing the interrupt (if needed) and dismiss the interrupt with JEN.[1]

AOSx, SKIPx, SOSx, CONSx, or BLKx If the skip condition is satisfied, then the APR dismisses the interrupt and returns to the interrupted program. If the skip condition is not satisfied, the instruction in the second interrupt location for the channel is executed as an interrupt instruction.

everything else All other instructions take their action then dismiss the interrupt and return to the interrupted program. This differs from the KA10 but is like the KI10 and later models. Differing from the KI10, jump and skip instructions (other than those listed above) do not jump or skip but simply return to the interrupted instruction.

[1] The KI10 allows MUUO instructions as interrupt instructions. This might work but has not been tested.

2.2.2 Unassigned Codes

The unassigned codes are 100-107, 114-117, 123, 247 (if CIRC is not included), and 257.

The KA10 handles unassigned codes much like MUUOs except they go through locations 60 and 61 rather than 40 and 41. The KX10 sends both MUUOs and unassigned codes through locations 40 and 41. The KV10 may do it either way and is controlled through the `UAC` option in `constants.vh`. Defining `UAC` causes the KV10 to use locations 60 and 61 for unassigned code processing. [2]

[2] At some point I expect I will settle on one or the other and remove this compile option.

Chapter 3

Input / Output

3.1 BLKI

On some I/O devices, BLKI works a little differently than on most PDP-10s. Normally BLKI skips only if the count in the left half of location E is not zero so not-skipping indicates the read buffer is full and reading is finished. However, in some cases the I/O device may know better than the software when the end is reached. These devices may notify the APR they've reached the end and this will cause BLKI to not skip even though the count has not yet reached 0.

An example of this might be an Ethernet device. It can start delivering data to the processor before it knows the length of the packet so a device driver that wanted to use BLKI as the interrupt instruction would not know how many words to set up in its block. With this modification, when either the end of the packet or the end of the buffer is reached, BLKI doesn't skip. The code outside of the BLKI loop then needs to check why it was reached, if it was end of packet or buffer overflow.

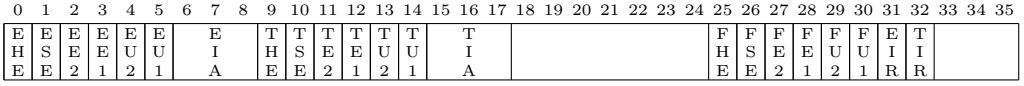
3.2 I/O Devices

Some I/O devices go with the processor itself (APR, PI, and PAG) while others are specific to the particular realization of the computer. It may be eventually that this distinction will be important and the implementation specific devices will be separated out into separate manuals. That's awfully forward looking; my optimism knows no bounds.

The operations of CONSZ, CONSO, BLKO, and BLKI are not described in here as they're just operations on top of the base of CONI, DATAO, and DATAI. Any exceptions will be noted – so far, there are none.

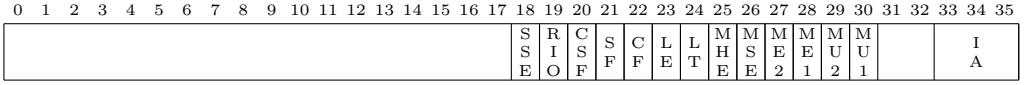
3.2.1 APR – Device 000

CONI APR, E



See PDP-10/X documentation.

CONO APR, E



See PDP-10/X documentation.

DATAI APR, E

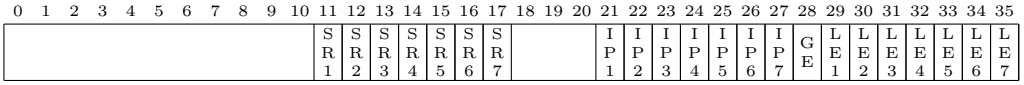
DATAO APR, E

Since the switches in the BL10 console are capacitive touch switches whose current state is held by internal flip-flops, like the KI10 their values can be written as well as read. Unlike the KI10, writing is done to the APR device rather than paper-tape reader (PTR). This switch register is reflected in and controlled by the DATA SWITCHES on the console.

If the console is not included, the switch register is still there and can be read and written under software control. It's just that it can't be observed by looking at the front of the computer and it can't be changed manually.

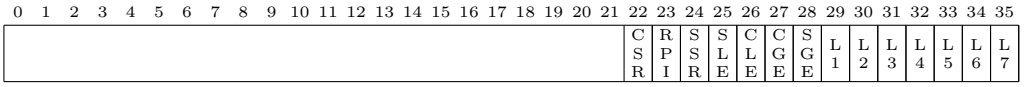
3.2.2 PI – Device 004

CONI PI, E



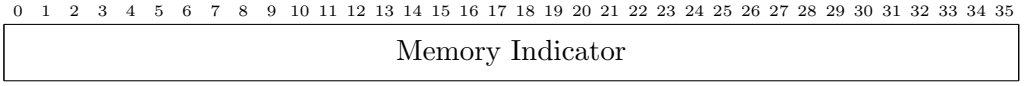
See PDP-10/X documentation.

CONO PI, E



See PDP-10/X documentation.

DATAO PI, E



Display the contents of location E on the console MEMORY/PROGRAM DATA and light the MEM light.

DATAI PI, E

This one is up for grabs. I could just read back in the value written out to the Memory Indicator but since those can't be changed manually, it seems rather pointless. I could read in the address switches but I'd be disappointed not to have a way to set them too. I could make use of DATAO PTR like the KI did for writing the address switches. I could read the READ IN DEVICE switches on the console. This has the potential use of letting boot code know where it booted from and those switches could pick up the functionality of the SENSE switches that the BL10 console doesn't have.

Chapter 4

Console

4.1 Hardware

The BL10 console device is an array of lights and capacitive touch switches designed to display some of the internal state of a PDP-10 processor and let an operator control the computer. It communicates with the processor over an I2C bus so it doesn't come close to keeping up at full speed. Humans can't see lights blink that rapidly anyway.

It has an alternative communications port of a logic-level async serial interface with a connector that matches with the FTDI serial to USB cables that are readily available. This is to enable someone to implement console functions on a simulated processor such as the KLH10 or SIMH.

4.2 Operation

The console gives an operator direct, low-level control over the KV10. It can be used to poke around with the base machine, enter simple programs, or boot the computer.

Starting in the top, right: green lights indicate power is applied and the processor is running. Three red lights indicate the processor is stopped and the reason for stopping, whether manual (from the console), the program (it executed a HALT instruction, aka JRST 4), or the processor encountered a double hard error.

The PAG lights show if the pager is enabled and when it signals page fault and the Processor Flags show those architectural flags that the programmer

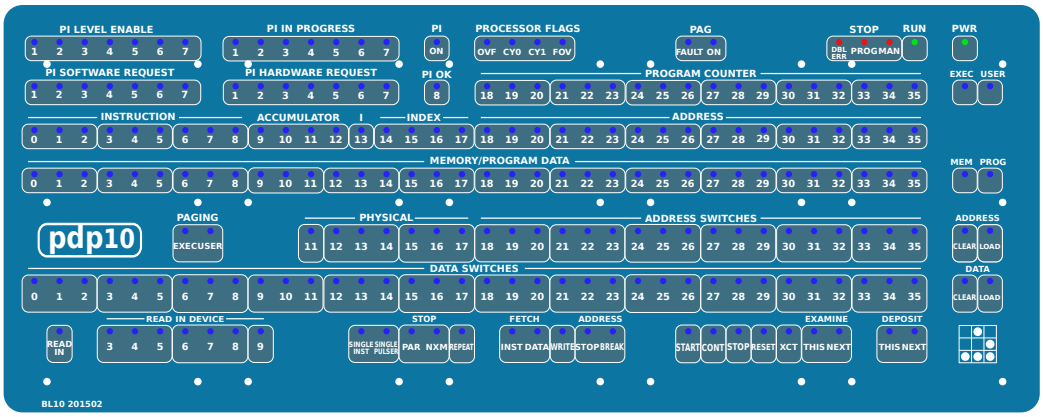


Figure 4.1: BL10 Operator's Console

might use.

To the right of those are a block of lights showing the status of the PI system: whether the system as a whole is enabled, which levels are enabled, which are in-progress, and whether there are PI requests either from hardware devices or internally from the APR.

Moving down we have the Program Counter and below that the instruction that's executing. To the right of the Program Counter are lights indicating if that instruction is being executed in exec or user mode.

The Memory/Program Data lights serve a dual purpose. When the APR is running, a DATAO PI will write to these lights and be indicated by lighting the PROG light. When the APR is halted, these lights will show the results of reading memory from the console and this state will be indicated by the MEM light.

The Address Switches set the address to be read or written from the console. Clear sets the Address Switches to all 0 while Load sets them to the corresponding bits shown in the Memory Data indicators.

To the left of the Address Switches are the Paging controls. Reading and writing to memory can be done to physical memory or to virtual memory of either the executive or user memory spaces, as defined by the pager. If virtual memory, only the right 18-bits of the Address Switches are used. The current pager only defined 22-bits of physical address but there is potential room in the page-table entries for up to three more bits so I've left room for

that.

The Data Switches are used for writing data to memory locations. The clear and load switches to the right work like they work for the Address Switches.

The bottom row of switches are what initiate the various operations of the console. [need to write a lot more here, once I know what operations I'm going to have.]

The Read In Device is for booting the computer. Enter the device to boot from, then push Read In to load the boot code from that device and run it. Read In resets the device and then reads, using repeated DATAI operations, a .SAV file from the device until it reaches the end where it finds a JRST instruction and it starts the processor at the location indicated. The KV10 actually does execute this final instruction of the .SAV file (in the sense of the XCT instruction) so if you arrange to put some other instruction at the end of your .SAV file, you may get unexpected results. Or perhaps that's what you wanted to happen.

Chapter 5

Internals

The KV10 is a horizontal micro-code machine with a 64 bit micro-word.[1] The data-paths are shown in Figure 5.1.

A key aspect to the micro-engine is the conditional branches that are available to the micro-code. These branches are OR'd into the next micro-address, therefore requiring that branch destinations be suitably aligned.

brJUMP The default is just to transfer to the next micro-instruction as specified by the next instruction field of the micro-instruction.

brREAD A two-way branch that allows waiting for the read ack to appear when reading memory. The read data will be read one clock cycle after the ack. The address does not have to be held on mem_addr after mem_read is asserted.

brWRITE A two-way branch that allows waiting for the write ack to appear when writing to memory. The address and data to be written do not have to be held after mem_write is asserted.

brIX A three-way branch on the Index and Indirect fields of an instruction to allow for effective address calculation. This is currently taken from write_data rather than inst because we want to do the branch one cycle early. Indexing takes priority since we want to do that calculation first.

0 Index is non-zero

1 Indirect

[1] Currently not all bits are used and this is subject to change as more is implemented.

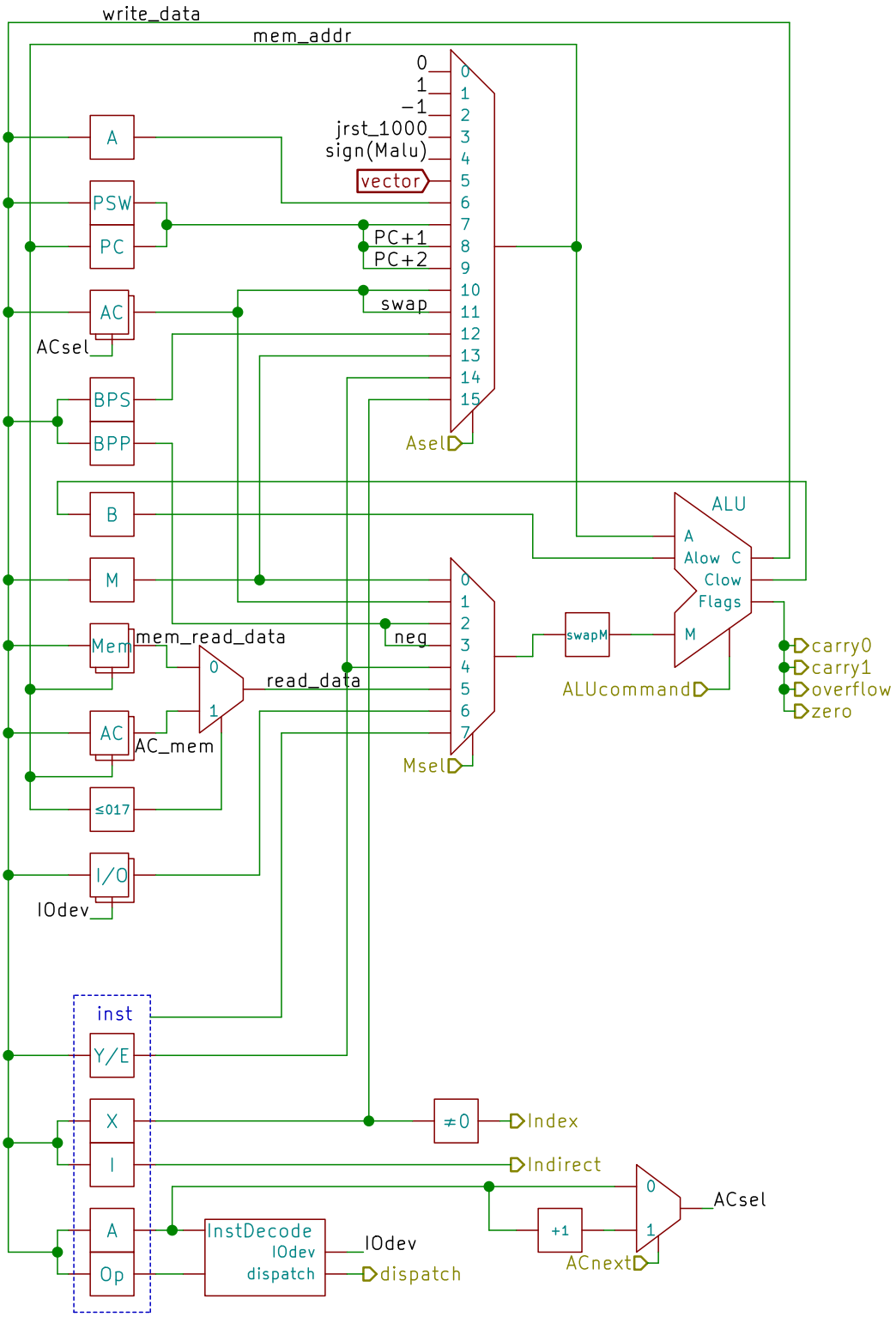


Figure 5.1: KV10 Data Paths

2 Otherwise

brI If we process a Index during the Effective Address calculation, then we need to come back and check for Indirect. This check is for that and operates off the inst register since the instruction will have been written by now.

brDISPATCH This is a complicated branch that handles a variety of different tasks, all put into a single branch to save cycles. It is primarily driven from the instruction decode ROM. The default is that the branch dispatch is the opcode of the instruction.

Instructions that read the contents of the memory at the effective address branch to 740_8 to execute that read. An internal flag is set so when the instruction is dispatched again, it doesn't just read the contents at E again.

Some instructions get special dispatching, namely JRST and the I/O instructions.[2]

brCOMP Branch if the comparison (as specified from the instruction decode) matches.

brCOMP0 Branch if the comparison with 0 (as specified from the instruction decode) matches.

brSELF For instructions that write to self, check if AC is not equal to 0.

brTEST Branch if the bitwise AND of the A and M inputs to the ALU is not 0. This is used for the TEST instructions.

brJFCL For the JFCL instruction, branch if and of the processor flags are about to be cleared.

brMUL An 8-way branch for the different MUL and DIV instructions.

0 IMUL/IDIV

1 IMULI/IDIVI

[2]Eventually I wish to attempt merging more of the effective address calculations into brDISPATCH to save a clock cycle on any instructions that don't use Indexing or Indirection.

2 IMULM/IDIVM

3 IMULB/IDIVB

4 MUL/DIV

5 MULI/DIVI

6 MULM/DIVM

7 MULB/DIVB

brOVR Branch if the ALU is indicating overflow.

brBPDISP A 4-way branch for the different Byte instructions.

0 ILDB

1 LDB

2 LDPB

3 DPB

brFPD Branch if the First Part Done flag is set.

brBLTDONE Branch if RIGHT(AC) equals E – used for terminating BLT.

brIOREAD Branch when we receive the ack for an I/O read.

brIOWRITE Branch when we receive the ack for an I/O write.

During a memory read we may receive an interrupt. In this case the micro-engine immediately jumps to location 777_8 to begin the interrupt processing.

During a memory read or write we may receive a page fault. In this case the micro-engine immediately jumps to location 775_8 which simply aborts the current instruction execution and begins again. If interrupts are set up correctly, the attempt to fetch the instruction will be blocked and the interrupt will be taken. If not, this is a double-error.[3]

[3]Need to implement double errors.

Bibliography

[1] Dave Conroy, *PDP-10/X System Manual*. `http://fpgaretrocomputing.org/pdp10x/`

[2] Digital Equipment Corporation, *PDP-10 System Reference Manual*. `http://bitsavers.com`, 1970.

[3] Digital Equipment Corporation, *DECsystem-10 System Reference Manual*. DEC-10-HGAD-D, 1971.

[4] Digital Equipment Corporation, *DECsystem-10/DECSYSTEM-20 Processor Reference Manual*. AH-H391A-T1, June 1982.