**Programming Assignment 4 -- CS429 Fall 2013**

Due Date: 27 October 2013

**Purpose**

The purpose of this assignment is to increase the complexity of your C programming, and give you experience with the architecture of a computer system.

**Program Specification**

Write a program to simulate a PDP-8 computer system. See [pdp8.html](http://www.cs.utexas.edu/users/peterson/pdp8/pdp8.html)for a description of the machine. You need to support the memory reference instructions, the operate instruction (both group 1 and group 2), and the IOT instruction.

Execute instructions until you get the HLT instruction.

Treat the OSR instruction as a NOP.

The IOT instruction is handled specially. Ignore the function code. If the device is 3, do a getchar() to read a character from standard input and put that character in the A register. If the device is 4, take the low-order 8 bits from the A register and output it as an ASCII character to standard output (putchar()). Assume these IOT instructions take 1 cycle.

Your program should either work silently -- with only the output from an IOT to device 4 on standard output, or in verbose mode. Verbose mode is selected by a "-v" option on the command line:

pdp8 -v test.obj.

In verbose mode, print one line for each instruction executed. That line should have

* The time (in cycles) after each instruction is executed.
* The PC of the instruction executed.
* The hex of the entire instruction executed.
* The symbolic opcode of the instruction executed.
* The values of the A register and the Link register after the instruction is executed.

For example:

Time 1: PC=0x003 instruction = 0xE81 (CLA IAC), rA = 0x001, rL = 0 Time 3: PC=0x004 instruction = 0x600 (DCA), rA = 0x000, rL = 0 Time 4: PC=0x005 instruction = 0xE80 (CLA), rA = 0x000, rL = 0 Time 6: PC=0x006 instruction = 0x200 (TAD), rA = 0x001, rL = 0 Time 8: PC=0x007 instruction = 0x849 (JMS), rA = 0x001, rL = 0 Time 9: PC=0x04A instruction = 0xE08 (RAR), rA = 0x000, rL = 1 ...

This comes from the following format:

"Time %lld: PC=0x%03X instruction = 0x%03X (%s), rA = 0x%03X, rL = %d\n"

These trace lines should be printed to stderr, since stdout is used for the IOT (device 4) output.

Your input will be a PDP-8 object file. A PDP-8 object file is a number of lines. Each line has two fields, separated by a colon (and a blank). The first field is normally a hex representation of an memory address, and the second field is the hex representation of the contents of that location. There should be one line with a first field of "EP" to define the starting PC value for the program (the entry point). The second field for this line is the hex representation of the initial PC value.

The PDP-8 object file is given by a file name on the command line.

**Example**

For example, assume the PDP-8 object file was test.obj:

EP: 067 064: 000 065: 000 066: F02 067: 664 068: 275 069: E21 ...

Then running this in normal mode would be just:

pdp8 test.obj

and there would be no output (there are no IOT instructions to device 4, so no output).

In verbose mode,

pdp8 -v test.obj

Output would be

Time 2: PC=0x067 instruction = 0x664 (DCA), rA = 0x000, rL = 0 Time 4: PC=0x068 instruction = 0x275 (TAD), rA = 0x011, rL = 0 Time 5: PC=0x069 instruction = 0xE21 (CMA IAC), rA = 0xFEF, rL = 1 ...

See how the entry point in the object file (EP:067) is the first instruction to be executed in the verbose output, and the instructions in the verbose output match the values defined for those locations in the object file.

**Error Cases**

If you have an error in the object file, you can just issue an appropriate error message and stop.

If you have an IOT to a device other than 3 or 4, halt.

If you get an illegal instruction, treat it as a Halt instruction or a NOP. The only illegal instructions I can think of is if the low-order bit of a Group 2 operate instruction was "1", or both RAR and RAL for Group 1 were set.

**Test Cases**

Provided test case is [test.obj](http://cs.utexas.edu/users/peterson/prog4/test.obj)which produces verbose output of [test.out](http://cs.utexas.edu/users/peterson/prog4/test.out).

**Submission**

You need to submit files to create an executable called pdp8 for program4. More specifically, for submission you should use

turnin --submit zxy program4 \*.h \*.c [Mm]akefile

Turn in enough that we can say "make pdp8" to build an executable for testing. This can be a single C file (pdp8.c) or it can be multiple C files and header files, with a Makefile that knows how to "make pdp8".

**Extensions**

Extensions to consider:

* There are multiple different PDP-8 models: PDP-8/I, PDP-8/E, PDP-8/S, PDP-8/L, PDP-8/A. How would you modify your program to allow the user to pick a specific model and interpret programs for that model only?
* Could you extend your program to put up a Window that looks like a PDP-8 console, and simulate both the console and the machine behind it? Now you can implement the OSR micro-instruction.
* The original PDP-8 ran at 1.5 microseconds per cycle. Could you modify your program so that, running on a stand-alone x86 computer, it ran at that same speed?

Due Date: 27 October 2013