**Programming Assignment 7 -- CS429 Fall 2013**

Due Date: 5 December 2013 (end of last day of class).

**Purpose**

The purpose of this assignment is to give you lots more experience with C programming.

**Motivation**

The PDP-429 was a 12-bit machine. Given it's success, it would make sense to evolve it to be a 16-bit machine. This involves a lot of different design decisions. For our class, this becomes the [PDP-429](http://www.cs.utexas.edu/users/peterson/pdp429/pdp429.html).

To determine if this is a good design, we need to evaluate it. We could build it in hardware, but it is more cost-effective to model it in software first, so we need an interpreter for the PDP-429. This is basically the same as Program 4, but for a more complex machine.

**Program Specification**

Write a program to simulate a PDP-429 computer system. See [pdp429.html](http://www.cs.utexas.edu/users/peterson/pdp429/pdp429.html)for a description of the machine.

Assume each instruction takes 1 cycle for every memory reference (including the instruction fetch), but no additional time for any computation. So, time is dominated by just the memory fetch time.

Execute instructions until you get the HLT instruction.

The IOT instruction is handled specially. Ignore the function code. If the device is 3, do a getchar() to read a byte from standard input (stdin) and put that byte in the register specified in the IOT instruction. An EOF on stdin should cause the register to be set to 0xFFFF. If the device is 4, take the low-order 8 bits from the specified register and output it as a byte to standard output (putchar()).

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:

pdp429 -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(s) of the instruction executed.
* The names and values of all memory and registers that are referenced by the exection of the instruction.

For example:

Time 1684: PC=0x011B instruction = 0xB515 (JMP): 0x0115 -> PC Time 1685: PC=0x0115 instruction = 0xE081 (MOD): A -> 0x0009, B -> 0x000A, 0x0009 -> C Time 1687: PC=0x0116 instruction = 0x1911 (ADDC): C -> 0x0009, M[0x0111] -> 0x0030, 0x0039 -> C Time 1690: PC=0x0117 instruction = 0x9B0F (I STC): M[0x010F] -> 0x010A, C -> 0x0039, 0x0039 -> M[0x010A] Time 1693: PC=0x0118 instruction = 0xB10F (ISZ): M[0x010F] -> 0x010A, 0x010B -> M[0x010F]

This comes from the following format:

fprintf(stderr, "Time %3lld: PC=0x%04X instruction = 0x%04X (%s)", time, old\_PC, inst, get\_opcode\_names()); char \*regs = ...; if (regs != NULL) fprintf(stderr, ": %s", regs); fprintf(stderr, "\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-429 object file. A PDP-429 object file is a binary file similar to the PDP-8 binary object file format, but with a full 16-bits in each 2-byte pair (see below).

The PDP-429 object file is given by a file name on the command line.

**The PDP-429 Binary Object File Format**

In the binary object file format, the first 4 characters of a PDP-429 object file are "OBJG". Next, we will have two bytes which will store the 16-bit starting address for the object file. The high-order 8 bits in the first byte and the low-order 8 bits in the second byte. This same representation (two bytes, 8-bits in each, high-order first), will be used for all PDP-429 addresses and 16-bit values in the object file format.

After the magic number ("OBJG") and the entry point address (2 bytes) will be a series of blocks. There will be as many blocks as necessary. Each block will be a set of bytes. The first byte (n) is the number of bytes in the entire block, including the first byte. So this first byte cannot be less than 1, and is of no practical value if it's less than 5. After this first byte will be a PDP-429 address (in the two byte format of upper 8 bits/lower 8 bits), followed by a set of values. The values will be PDP-429 16-bit values that are to be stored, starting at the address given, and continuing word after word, until we have exhausted the number of bytes at the beginning of the block.

See [objspec.gif](http://www.cs.utexas.edu/users/peterson/prog5/objspec.gif)but remember that the magic number is "OBJG", not "OBJ8".

**Test Cases**

Two test case are provided to start. Look in [prog7](http://www.cs.utexas.edu/users/peterson/prog7/). There you will see two test cases: jan and prime. I will add more as they are developed.

For each test (jan and prime), we have test.asm (the assembly), and test.lst (the output listing of the assembler), plus an object file (test.obj). When running the test, you will need its input file (test.in) and its output file (test.out) plus the log of verbose print messages (test.log). So each test has 6 files. The object file, input file, output file, and log file are used/produced as:

pdp429 -v test.obj < test.in > test.out 2> test.log

**Extensions**

Extensions to consider:

* Consider a much more precise model of timing, where each instruction takes a different amount of time, depending on it's complexity. So an ADDA instruction would take 2 cycles, but a MULA would take 8 cycles, and a DIVA would take 12 cycles. A memory reference would take 100 cycles. Since an ADDA requires an instruction fetch (100), plus an operand fetch (100) plus the actual addition (2). An ADD (register to register) would only take the instruction fetch (100) plus the addition (2).
* How do things change if we introduce a cache. A cache hit would only take 8 cycles; a miss would still take 100 cycles.
* Could we pipeline this machine? How would that change the interpreter?
* What is missing from this ISA? How would we add what is missing? What if we try to extend it to 32 bits? To 64 bits?

**Submission**

You need to submit files to create an executable called pdp429 for program7. More specifically, for submission you should use

turnin --submit zxy program7 Makefile \*.h \*.c

if you have more than one file. If you have only the one file, you can just:

turnin --submit zxy program7 pdp429.c

Due Date: 5 December 2013