Homework 1 - Supplemental test cases, resources, and rubric

February 17, 2018

1 Instructions & OP codes

The *Register-based OP codes* are shown below. In several cases, primarily with the mathematical operations, the Instruction Set Architecture (ISA) **pseudo-code** is included in the rightmost column. The instructions requiring more detailed psuedo-code are shown in the next table.

Table 1: Instructions

OP-code	Instruction	Explanation
01	LIT R, 0, M	Loads a constant value (literal) M into Register R
02	RTN 0, 0, 0	Returns from a subroutine and restore the caller environment
03	LOD R, L, M	Load value into a selected register from the stack location at offset M from L lexicographical levels down
04	STO R, L, M	Store value from a selected register in the stack location at offset M from L lexicographical levels down
05	CAL 0, L, M	Call procedure at code index M (generates new Activation Record and pc M)
06	INC 0, 0, M	Allocate M locals (increment sp by M). First four are Functional Value, Static Link (SL), Dynamic Link (DL), and Return Address (RA)
07	JMP 0, 0, M	Jump to instruction M
08	JPC R, 0, M	Jump to instruction M if R = 0
09	SIO R, 0, 1	Write a register to the screen
09	SIO R, 0, 2	Read in input from the user and store it in a register
09	SIO 0, 0, 3	End of program (program stops running)
10	NEG	$(R[i] \leftarrow -R[j])$
11	ADD	$(R[i] \leftarrow R[j] + R[k])$
12	SUB	$(R[i] \leftarrow R[j] - R[k])$
13	MUL	$(R[i] \leftarrow R[j] * R[k])$
14	DIV	$(R[i] \leftarrow R[j] / R[k])$
15	ODD	$(R[i] \leftarrow R[i] \bmod 2)$ or ord(odd(R[i]))
16	MOD	$(R[i] \leftarrow R[j] \bmod R[k])$
17	EQL	$(R[i] \leftarrow R[j] == R[k])$
18	NEQ	$(R[i] \leftarrow R[j] != R[k])$
19	LSS	$(R[i] \leftarrow R[j] < R[k])$
20	LEQ	$(R[i] \leftarrow R[j] <= R[k])$
21	GTR	$(R[i] \leftarrow R[j] > R[k])$
22	GEQ	$(R[i] \leftarrow R[j] >= R[k])$

The following table contains the *pseudo-code* for the instructions that manipulate the stack pointer, base pointer, return link, dynamic link, memory and a bit more.

Table 2: Psuedo-code

Op-code	Mnemonic	Pseudo-code
01	LIT R, 0, M	R[i] ← M

02	RTN 0, 0, 0	sp ← bp - 1;
		bp ← stack[sp + 3];
		pc ← stack[sp + 4];

03	LOD R, L, M	R[i] ← stack[base(L, bp) + M]
04	STO R, L, M	stack[base(L, bp) + M] ← R[i]
05	CAL 0, L, M	stack[sp + 1] ← 0; /* space to return value stack[sp + 2] ← base(L, bp); /* static link (SL) stack[sp + 3] ← bp; /* dynamic link (DL) stack[sp + 4] ← pc; /* return address (RA) bp ← sp + 1; pc ← M;
06	INC 0, 0, M	sp ← sp + M
07	JMP 0, 0, M	pc ← M
08	JPC R, 0, M	if R[i] == 0 then pc ← M;

1.1 Output format specifications

The code used to generate the test cases described in the following section use the following ***printf*** formats:

```
//The first line header for all the INSTRUCTION and PC, BP & SP
   printf("\n OP   Rg Lx Vl[ PC BP SP]\n");
// The SIO input and output specification
   scanf("%d",&registers);  //NOTE the registers variable is a PLACEHOLDER
   printf("%d\n",registers);//You can use whatever variable name you need
// The op code, register, lexical level, value, PC, BP, & SP
   printf("%-4s%3d%3d%3d[%3d%3d%3d] ", op, reg, lex, value, pc, bp, sp);
// The STACK section
   printf("|"); //for the BEGINNING of a new lexical level
   printf("%3d ",stack[i]); //for each member of the current level
   printf("\n"); // at the end of full stack display
// The REGISTERS
   printf("\tRegisters:[%3d%3d%3d%3d%3d%3d%3d%3d]\n",
          r0, r1, r2, r3, r4, r5, r6, r7);
```

Please note that there is **NO REQUIREMENT** to use the same variable names as shown above.

It is a bit of a dicey proposition to CUT and PASTE from a PDF. You might save a bit of time by directly typing the format specifications as defined above.

2 Test Cases

There are four test cases supplied for testing. They are described below. The input files and all the expected output files are in the ZIP file for this assignment. See the notes on usage.

1. The test case named *cube10Test.txt* is shown below. And the expected output file is in ZIP file. The expected output is named **cube10TestOutput.txt**.

```
01 07 00 10
05 00 00 04
09 06 00 01
09 00 00 03
13 06 07 07
13 06 07 06
02 00 00 00
```

2. The text case named *factorialTest.txt* was provided as part of the assignment. it consists of the following instructions to be used as input to the Pcode *Virtual Machine*. The output is in the file named **factorialTestOutput.txt**.

```
06 00 00 06
01 00 00 03
04 00 00 04
01 00 00 01
04 00 00 05
05 00 00 07
07 00 00 19
06 00 00 04
03 00 01 04
03 01 01 05
13 01 00 01
04 01 01 05
01 01 00 01
12 00 00 01
04 00 01 04
18 00 00 01
08 00 00 18
05 00 01 07
02 00 00 00
03 00 00 05
09 00 00 01
09 00 00 03
```

3. The test case named *lodStoTest.txt* is shown below. And the expected output file is in ZIP file. The expected output is named **lodStoTestOutput.txt**.

```
01 00 00 05
01 01 00 03
06 00 00 06
04 00 00 04
04 01 00 05
09 00 00 01
09 01 00 01
09 00 00 03
```

4. The test case named *lodStoCalTest.txt* is shown below. And the expected output file is in ZIP file. The expected output is named **lodStoCalTestOutput.txt**.

   ```
   01 00 00 05
   01 01 00 03
   06 00 00 06
   04 00 00 04
   04 01 00 05
   05 00 01 07
   09 00 00 03
   03 06 01 04
   03 05 01 05
   13 07 01 00
   09 07 00 01
   02 00 00 00
   ```

5. The input file named *square.txt* expects input as the very first activity. In the event you want to test your submission for input use the following command sequence:

   ```
   mcalpin@eustis$./pm0vm squareTest.txt

    OP   Rg Lx Vl[ PC BP SP]
   50
   SIO   1  1  2[  1  1  0]
   Registers:[  0 50  0  0  0  0  0  0]
   CAL   0  1  4[  4  1  4] |  0   0   0   1
   Registers:[  0 50  0  0  0  0  0  0]
   MUL   1  1  1[  5  1  4] |  0   0   0   1
   Registers:[  02500  0  0  0  0  0  0]
   RTN   0  0  0[  2  1  0]
   Registers:[  02500  0  0  0  0  0  0]
   2500
   SIO   1  0  1[  3  1  0]
   Registers:[  02500  0  0  0  0  0  0]
   SIO   0  0  3[  4  1  0]
   Registers:[  02500  0  0  0  0  0  0]
   ```

   Note that the 50 under the OP heading is the *unsolicited and unprompted* input.

3 Usage Notes

The files supplied in the ZIP file are:

- Input test files consisting of raw instructions as described in the assignment.
- The expected *output* files, all with the ****Output.txt*** in the filename.
- The **shell script** to run the program with four input files described above.
- Note there is a fifth input file named **squareTest.txt** for your testing.

Download the ZIP file to your machine, then upload it to your homework 1 working directory. The **shell script** named **testingShell** expects the source code to be in a file named ***pm0vm.c*** located in the **same** diectory with the all of the **unzipped** contents of the ZIP file.

4 Grading

Scoring will be based on the following rubric:

Table 3: Grading Rubric

Deduction	Description
-100	Code does not compile on *Eustis*
-100	Code does not accept the input filename from the command line
- 15	Code does not show an error message and/or does not exit safely when there is a file I/O problem
- 20	crashed on cube10Test.txt, or output does not match
- 20	crashed on factorialTest.txt, or output does not match
- 20	crashed on lodStoTest.txt, or output does not match
- 20	crashed on lodStoCalTest.txt, or output does not match