Remove Watermark Wondershare PDFelement

Homework 1: P-Machine

COP 3402: Systems Software

See Webcourses and the syllabus for due dates.

Purpose

In this homework you will form a team [Collaborate] and implement a virtual machine called the P-machine [UseConcepts] [Build].

Directions

There are two parts to this homework:

1. (10 points) Email our GTA, Jie Lin (jie.lin@Knights.ucf.edu) with subject "Our team for COP 3402" and a list of the 1 or 2 members of your team. These team members must all be registered for the same lecture section (0002 for Dr. Montagne) and it is recommended that they all be in the same lab section as well. Your team will jointly do all project implementations in the course.

   One student must be identified as the contact person for the group, who will submit assignments and respond to emails from the staff, and another student must be identified as responsible for making sure that all team members understand everything about the solution. We will randomly ask questions of students in teams to ensure that all understand their solution.

2. (100 points) Implement and submit the P-machine as described in the rest of this document.

For the implementation, your code must be written in ANSI standard C and must compile with gcc and run correctly on Eustis. (See http://newton.i2lab.ucf.edu/wiki/Help:Eustis for information on how to access Eustis.) We recommend using the flag -Wall and fixing all warnings.

What to Read

Our recommended book is Systems Software: Essential Concepts (by Montagne) in which we recommend reading chapters 1-3.

In this assignment, you will implement a virtual machine (VM) known as the P-machine (PM/0).

Remove Watermark Wondershare PDFelement

P-Machine Architecture

The P-machine is a stack machine that conceptually has one memory area called the process address space (PAS). The process address space is divide into two contiguous segments: the “text”, which contains the instructions for the VM to execute and the “stack,” which is organized as a data-stack to be used by the PM/0 CPU.

Registers

The PM/0 has a few built-in registers used for its execution: The registers are named:

- base pointer (BP), which points to the base of the current activation record
- stack pointer (SP), which points to the current top of the stack. The stack grows downwards.,
- program counter (PC), which points to the next instruction to be executed.
- Instruction Register (IR), which store the instruction to be executed

The use of these registers will be explained in detail below. The stack grows downwards.

Instruction Format

The Instruction Set Architecture (ISA) of the PM/0 has instructions that each have three components, which are integers (i.e., they have the C type int) named as follows.

OP is the operation code.
L indicates the lexicographical level (We will give more details on L below)
M depending of the operators it indicates:
- A number (when OP is LIT or INC).
- A program address (when OP is JMP, JPC, or CAL).
- A data address (when OP is LOD, STO)
- The identity of the arithmetic/relational operation associated to the OPR op-code. (e.g. OPR 0 2 (ADD) or OPR 0 4 (MUL))

The list of instructions for the ISA can be found in Appendix A and B.

P-Machine Cycles

The PM/0 instruction cycle conceptually does the following for each instruction:

The PM/0 instruction cycle is carried out in two steps. The first step is the fetch cycle, where the instruction pointed to by the program counter (PC) is fetched from the “text” segment, placed in the instruction register (IR) and the PC is incremented to point to the next instruction in the code list. In the second step the instruction in the IR is executed using the “stack” segment. **(This does not mean that the instruction is stored in the “stack segment.”)**

Fetch Cycle:

1.- IR.OP ← pas[pc]
IR.L ← pas[pc + 1]
IR.M ← pas[pc + 2]
(note that each instruction need 3 entries in array “TEXT”.
2.- (PC← PC + 3).

Remove Watermark Wondershare PDFelement

Execute Cycle:
The op-code (OP) component in the IR register (IR.OP) indicates the operation to be executed. For example, if *IR* encodes the instruction "2 0 2", then the machine adds the top two elements of the stack, popping them off the stack in the process, and stores the result in the top of the stack (so in the end sp is one less than it was at the start). Note that arithmetic overflows and underflows happen as in C int arithmetic.

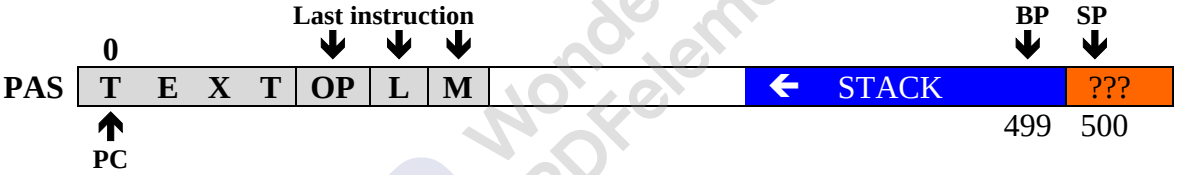
PM/0 initial/Default Values

When the PM/0 starts execution.

BP == 499, SP == 500, and PC == 0;

This means that execution starts with the "text segment" element 0. Similarly, the initial "stack" segment values are all zero (BP=499 and SP = BP + 1).

The figure bellow illustrate the process address space:



Size Limits

Initial values for PM/0 CPU registers are:
BP = 499
SP = BP + 1;
PC = 0;
Initial process address space values are all zero:
pas[0] =0, pas[1] =0, pas[3] =0…..[n-1] = 0.
Constant Values:
ARRAY_SIZE is 500

Note: Be aware that in PM/0 the stack is growing downwards

Remove Watermark Wondershare PDFelement

Assignment Instructions and Guidelines:

1. The VM must be written in C and must run on Eustis3. If it runs in your PC but not on Eustis, for us it does not run.
2. The input file name should be read as a command line argument at runtime, for example: $ ./a.out input.txt (A deduction of 5 points will be applied to submissions that do not implement this).
3. Program output should be printed to the screen, and should follow the formatting of the example in Appendix C. A deduction of 5 points will be applied to submissions that do not implement this.
4. Submit to Webcourses:
   a) A readme text file indicating how to compile and run the VM.
   b) The source code of your PM/0 VM which should be named "vm.c"
   c) Student names should be written in the header comment of each source code file, in the readme, and in the comments of the submission
   d) Do not change the ISA. Do not add instructions or combine instructions. Do not change the format of the input. If you do so, your grade will be zero.
   e) Include comments in your program. If you do not comments, your grade will be zero.
   f) Do not implement each VM instruction with a function. If you do, a penalty of -100 will be applied to your grade. You should only have two functions: main and base (Appendix D).
   g) The team member(s) must be the same for all projects. In case of problems within the team. The team will be split and each member must continue working as a one-member team for all other projects.
   h) On late submissions:
      - **One day late 10% off.**
      - **Two days late 20% off.**
      - **Submissions will not be accepted after two days.**
      - **Resubmissions are not accepted after two days.**
      - **Your latest submission is the one that will be graded.**

We will be using a bash script to test your programs. This means your program should follow the output guidelines listed (see Appendix C for an example). You don't need to be concerned about whitespace beyond newline characters. We use diff -w.

Remove Watermark Wondershare PDFelement

Rubric:

If you submit a program from another semester or we detect plagiarism your grade is F for this course.
Using functions to implement instructions even if only one is implemented that way, means that your grade will be "zero".
Pointers and handling of dynamic data structures is not allowed. If you do your grade is "zero"
-100 – Does not compile
10 – Compiles
25 – Produces lines of meaningful execution before segfaulting or looping infinitely
5 – Follows IO specifications (takes command line argument for input file name and prints output to console)
10 – README.txt containing author names
5 – Fetch cycle is implemented correctly
10 – Well commented source code
5 – Arithmetic instructions are implemented correctly
5 – Read and write instructions are implemented correctly
10 – Load and store instructions are implemented correctly
10 – Call and return instructions are implemented correctly
5 – Follows formatting guidelines correctly, source code is named vm.c

Wondershare PDFelement

Remove Watermark Wondershare PDFelement

Appendix A

Instruction Set Architecture (ISA) – (eventually we will use "stack" to refer to the stack segment in PAS)

In the following tables, italicized names (such as *p*) are meta-variables that refer to integers. If an instruction's field is notated as "-", then its value does not matter (we use 0 as a placeholder for such values in examples).

ISA:

01 –	**LIT 0, M**	Pushes a constant value (literal) **M** onto the stack
02 –	**OPR 0, M**	Operation to be performed on the data at the top of the stack. (or return from function)
03 –	**LOD L, M**	Load value to top of stack from the stack location at offset **M** from **L** lexicographical levels down
04 –	**STO L, M**	Store value at top of stack in the stack location at offset **M** from **L** lexicographical levels down
05 –	**CAL L, M**	Call procedure at code index **M** (generates new Activation Record and PC ← **M**)
06 –	**INC 0, M**	Allocate **M** memory words (increment SP by M). First four are reserved to **Static Link (SL)**, **Dynamic Link (DL)**, **and Return Address (RA)**
07 –	**JMP 0, M**	Jump to instruction **M (PC ← M)**
08 –	**JPC 0, M**	Jump to instruction **M** if top stack element is 0
09 –	**SOU 0, 1**	Write the top stack element to the screen
	SIN 0, 2	Read in input from the user and store it on top of the stack
	EOP 0, 3	End of program (**Set "eop" flag to zero**)

Remove Watermark Wondershare PDFelement

OP Code Number	**OP Mnemonic**	**L**	**M**	**Comment (Explanation)**
01	LIT	*0*	*n*	Literal push: sp ← sp- 1; pas[sp] ← *n*
02	RTN	0	0	Returns from a subroutine is encoded 0 0 0 and restores the caller's AR: sp ← bp + 1; bp ← pas[sp + 2]; pc ← pas[sp + 3];
03	LOD	*n*	*a*	Load value to top of stack from the stack location at offset *o* from *n* lexicographical levels down sp ← sp - 1; pas[sp] ← pas[base(bp, *n*) - *o*];
04	STO	*n*	*o*	Store value at top of stack in the stack location at offset *o* from *n* lexicographical levels down pas[base(bp, n) - o] ← pas[sp]; sp = sp +1;
05	CAL	*n*	*p*	Call the procedure at code index p, generating a new activation record and setting PC to *p:* pas[sp - 1] ← base(bp, **n**); /* static link (SL) pas[sp - 2] ← bp; /* dynamic link (DL) pas[sp - 3] ← pc; /*return address (RA) bp ← sp - 1; pc ← *p*;
06	INC	*0*	*m*	Allocate *m* locals on the stack sp ← sp - m;
07	JMP	*0*	*a*	Jump to the address in stack[SP-1] and pop: PC ← stack[SP-1]; SP ← SP-1
08	JPC	*0*	*a*	Jump conditionally: if the value in stack[sp] is 0, then jump to *a* and pop the stack: **if** (stack[SP] == 0) **then** { pc ← *a*; } sp ← sp+1
09	SYS	*0*	*1*	Output of the value in stack[SP] to standard output as a character and pop: **putc**(stack[sp]); sp ← sp+1
	SYS	0	2	Read an integer, as character value, from standard input (stdin) and store it on the top of the stack. sp ← sp-1; stack[sp] ← **getc**();
	SYS	0	3	Halt the program (Set "eop" flag to zero)

Remove Watermark Wondershare PDFelement

Appendix B (Arithmetic/L0gical Instructions)

ISA Pseudo Code

02 – **OPR 0, #** (1 <= # <= 10)

1	ADD	pas[sp + 1] ← pas[sp + 1] + pas[sp] sp ← sp + 1;
2	SUB	pas[sp + 1] ← pas[sp + 1] - pas[sp] sp ← sp + 1;
3	MUL	pas[sp + 1] ← pas[sp + 1] * pas[sp] sp ← sp + 1;
4	DIV	pas[sp + 1] ← pas[sp + 1] / pas[sp] sp ← sp + 1;
5	EQL	pas[sp + 1] ← pas[sp + 1] == pas[sp] sp ← sp + 1;
6	NEQ	pas[sp + 1] ← pas[sp + 1] != pas[sp] sp ← sp + 1;
7	LSS	pas[sp + 1] ← pas[sp + 1] < pas[sp] sp ← sp + 1;
8	LEQ	pas[sp + 1] ← pas[sp + 1] <= pas[sp] sp ← sp + 1;
9	GTR	pas[sp + 1] ← pas[sp + 1] > pas[sp] sp ← sp + 1;
10	GEQ	pas[sp + 1] ← pas[sp + 1] >= pas[sp] sp ← sp + 1;

Remove Watermark Wondershare PDFelement

Appendix C

Example of Execution

This example shows how to print the stack after the execution of each instruction.

INPUT FILE

For every line, there must be 3 values representing **OP**, **L** and **M**.

```
7 0 45
7 0 6
6 0 4
1 0 4
1 0 3
2 0 3
4 1 4
1 0 14
3 1 4
2 0 7
8 0 39
1 0 7
7 0 42
1 0 5
2 0 0
6 0 5
9 0 2
5 0 6
9 0 1
9 0 3
```

Wondershare PDFelement

When the input file (program) is read in to be stored in the text segment starting at location 0 in the process address space, each instruction will need three memory locations to be stored. Therefore, the PC must be incremented by 3 in the fetch cycle.

0			**3**			**6**			**9**			**12**			**15**			…
7	**0**	**45**	7	0	6	**6**	**0**	**4**	1	0	4	**1**	**0**	**3**	2	0	4	etc

The initial CPU register values for the example in this appendix are:

SP = 500;
BP = SP - 1;
PC = 0;
IR = 0 0 0; (a struct or a linear array can be used to implement IR)

Remove Watermark Wondershare PDFelement

Hint: Each instruction uses 3 array elements and each data value just uses 1 array element.

OUTPUT FILE

Print out the execution of the program in the virtual machine, showing the stack and pc, bp, and sp.

NOTE: It is necessary to separate each Activation Record with a bar "|".

```
                  PC   BP   SP   stack
Initial values:   0    499  500

     JMP  0    45   45   499  500
     INC  0    5    48   499  495  0 0 0 0 0
Please Enter an Integer: 3
     SIN  0    2    51   499  494  0 0 0 0 0 3
     CAL  0    6    6    493  494  0 0 0 0 0 3
     INC  0    4    9    493  490  0 0 0 0 0 3 | 499 499 54 0
     LIT  0    4    12   493  489  0 0 0 0 0 3 | 499 499 54 0 4
     LIT  0    3    15   493  488  0 0 0 0 0 3 | 499 499 54 0 4 3
     MUL  0    3    18   493  489  0 0 0 0 0 3 | 499 499 54 0 12
     STO  1    4    21   493  490  0 0 0 0 12 3 | 499 499 54 0
     LIT  0    14   24   493  489  0 0 0 0 12 3 | 499 499 54 0 14
     LOD  1    4    27   493  488  0 0 0 0 12 3 | 499 499 54 0 14 12
     LSS  0    7    30   493  489  0 0 0 0 12 3 | 499 499 54 0 0
     JPC  0    39   39   493  488  0 0 0 0 12 3 | 499 499 54 0
     LIT  0    5    42   493  489  0 0 0 0 12 3 | 499 499 54 0 5
     RTN  0    0    54   499  494  0 0 0 0 12 3
Output result is: 3
     SOU  0    1    57   499  495  0 0 0 0 12
     EOP  0    3    60   499  495  0 0 0 0 12
```

Remove Watermark Wondershare PDFelement

Appendix D

Helpful Tips

This function will be helpful to find a variable in a different Activation Record some **L** levels down:

```
/*********************************************/
/*            Find base L levels down              */
/*                                                 */
/*********************************************/

int base( int BP, int L)
{
      int arb = BP;      // arb = activation record base
      while ( L > 0)     //find base L levels down
      {
            arb = pas[arb];
            L--;
      }
      return arb;
}
```

For example in the instruction:

STO L, M - You can do stack [base (IR.**L**) + IR.**M**]= pas[SP] to store the content of the top of the stack into an AR in the stack, located **L** levels down from the current AR.

Note1: we are working at the CPU level therefore the instruction format must have only 3 fields. Any program whose number of fields in the instruction format is grater than 3 will get a zero.

Note2: If your program does not follow the specifications, your grade will get a zero.

Note3: if any of the instructions is implemented by calling a function, your grade will be zero.