Department of Computer Science, The University of Hong Kong
COMP2120
Computer Organization
Assignment 3
Deadline: 4 May 2017, before 11:55pm

Preamble:

In this assignment, you need to design and implement new instructions for a simulated CPU, and to write an assembly language program based on your newly implemented and existing instructions to perform certain task. Through this assignment, you are expected to learn more about the operations of CPU at microcode level.

You shall be using the following CPU Simulator for completing this assignment:

CPU Sim - A Java based CPU Simulator (version 4.0.9)
http://www.cs.colby.edu/djskrien/CPUSim/

Before you start working on it, you should downloadthe assignment package provided in moodle, which includes the aforementioned CPU simulator.

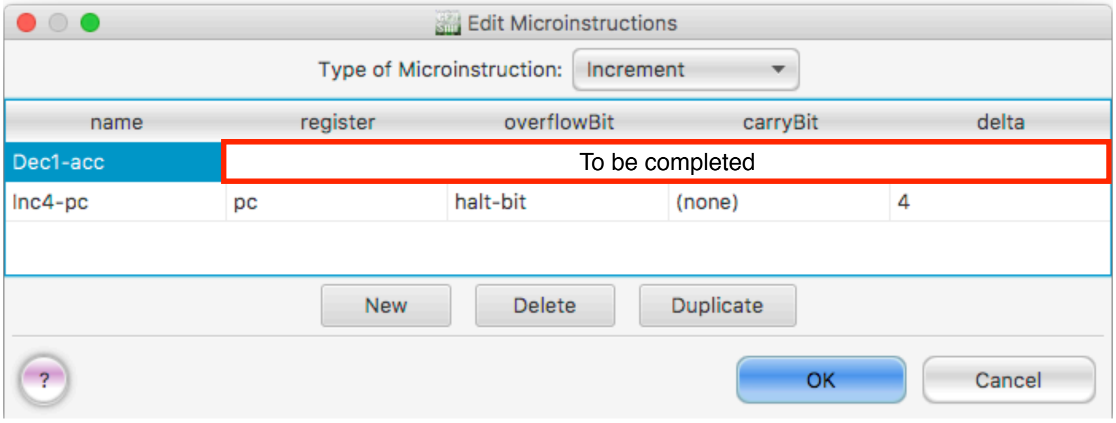
Once the simulator is installed, launch the application and open up a machine file with name "comp2120.cpu" (under the sub-folder 'machine' under the folder 'CPUSim4.0.9'). You shall need to modify this machine by incorporating new microinstructions as well as new machine instructions.

Handin:

Complete the following 9 questions, and save assembly language program (question 9) and the machine with newly designed machine instructions (questions 1-8) in a machine file. The two files have to be handin through moodle.

Questions:

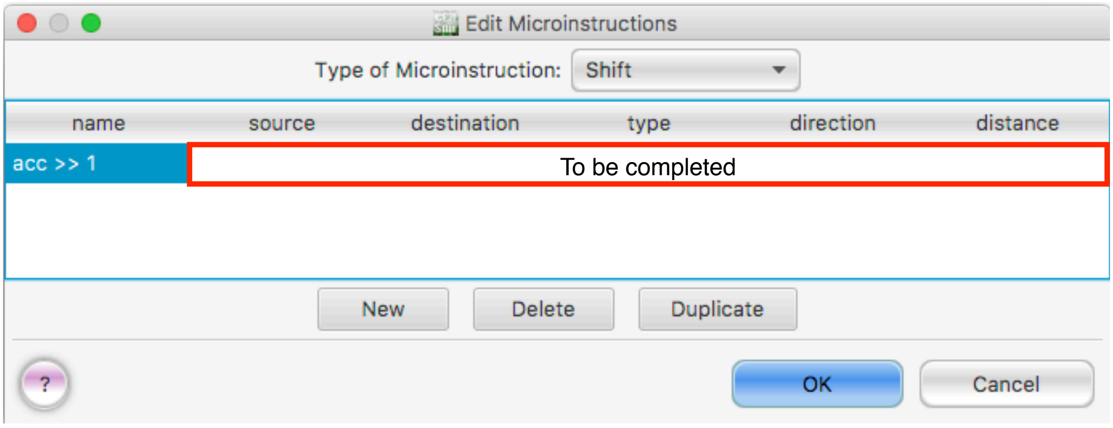
1. Add a new Microinstruction "***Dec1-acc***" under the type *Increment* so that it decrements the value of *acc* register by 1. (3%)



Type of Microinstruction: Increment

name	register	overflowBit	carryBit	delta
Dec1-acc	To be completed			
Inc4-pc	pc	halt-bit	(none)	4

2. Add a new Microinstruction "***acc >> 1***" under the type *Shift* so that it shifts the bit pattern in *acc* register to the right logically by 1 bit. (3%)

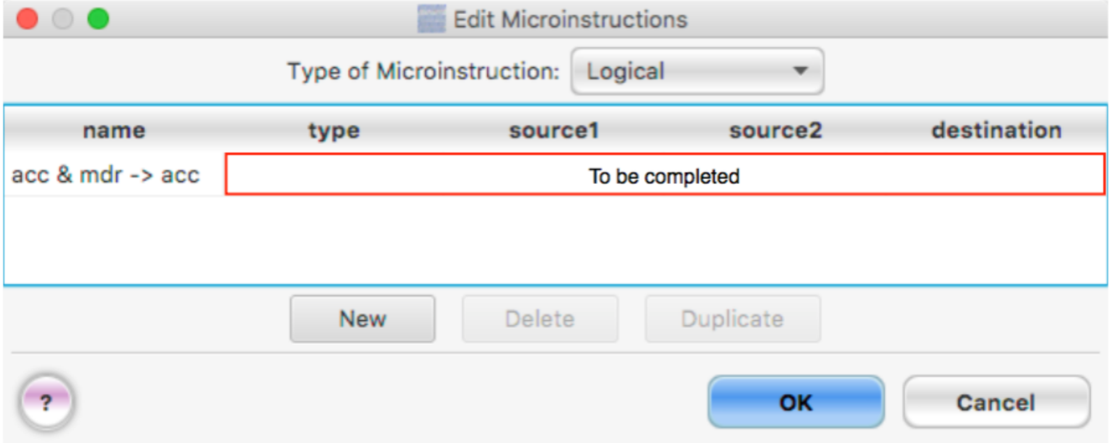


Edit Microinstructions

Type of Microinstruction: Shift

name	source	destination	type	direction	distance
acc >> 1	To be completed				

3. Add a new Microinstruction "***acc & mdr -> acc***" under the type *Logical* so that it performs the bit-wise AND operation between *acc* and *mdr* registers, with the result stored in *acc* register. (3%)

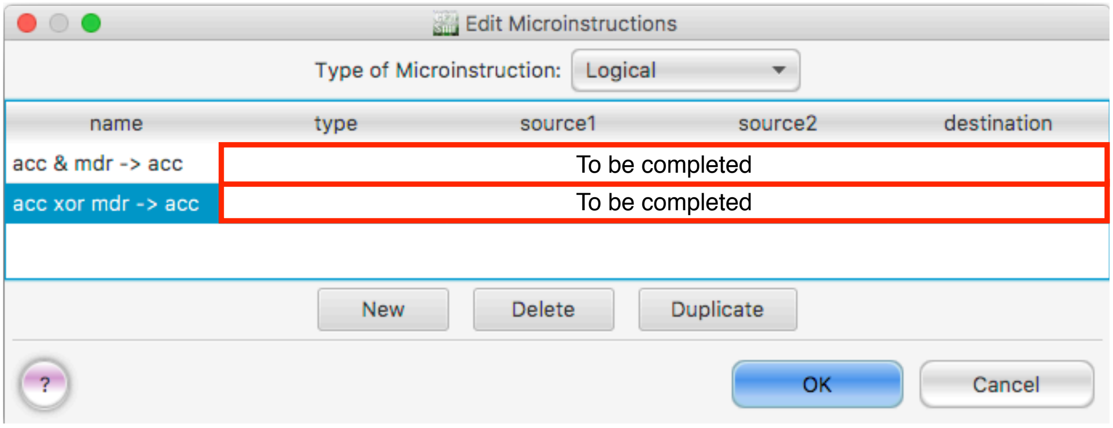


Edit Microinstructions

Type of Microinstruction: Logical

name	type	source1	source2	destination
acc & mdr -> acc	To be completed			

4. Add a new Microinstruction "***acc xor mdr -> acc***" under the type *Logical* so that it performs the bit-wise XOR operation between *acc* and *mdr* registers, with the result stored in *acc* register. (3%)

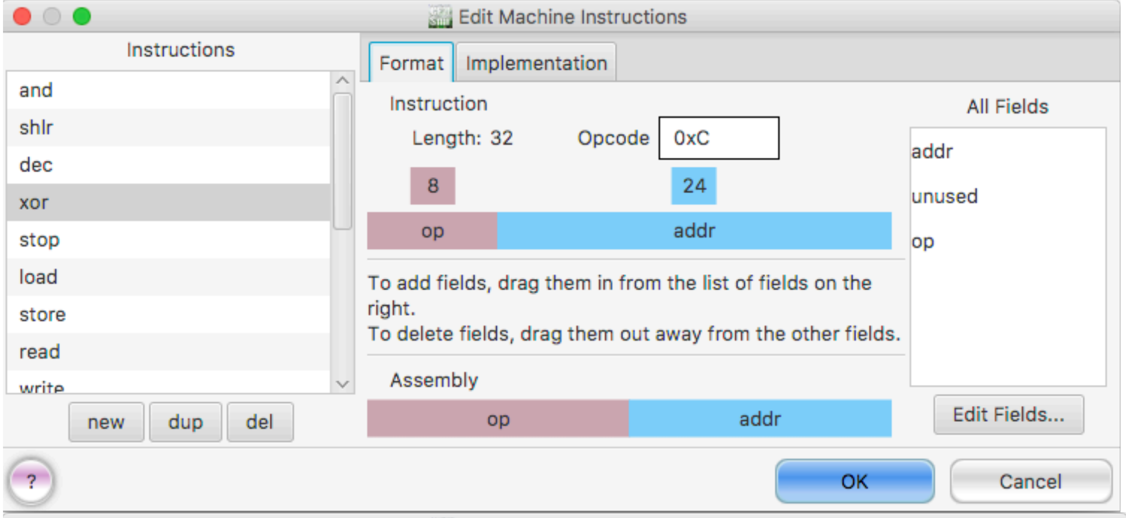


Edit Microinstructions

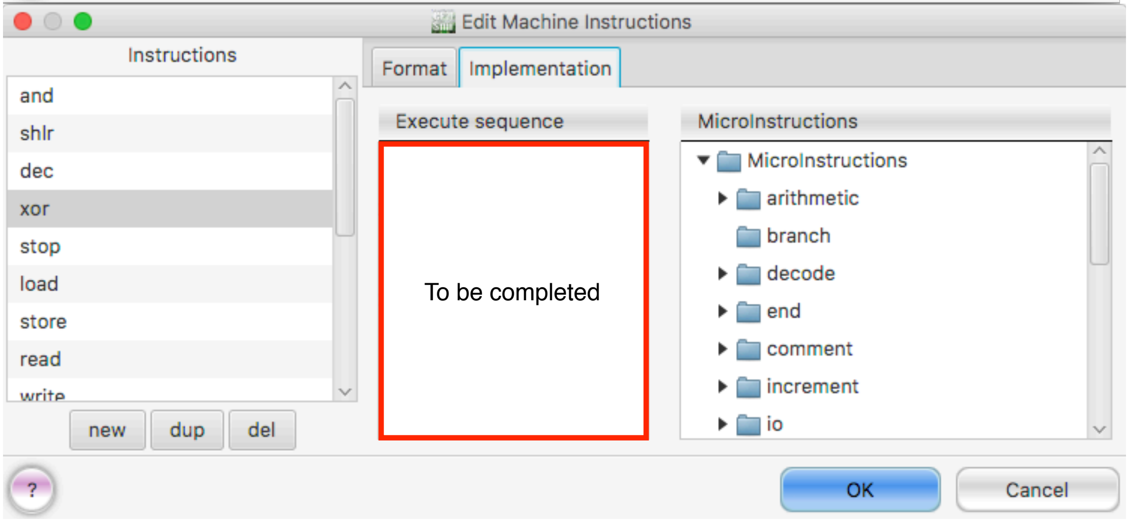
Type of Microinstruction: Logical

name	type	source1	source2	destination
acc & mdr -> acc	To be completed			
acc xor mdr -> acc	To be completed			

5. Add a new machine instruction "**xor**" with 8-bit opcode 0x0C, such that it takes one operand from 24-bit memory address *addr*, and perform the bit-wise XOR operation between the value stored at *addr* and *acc* register. The result has to be stored back to the *acc* register. (5%)



To add fields, drag them in from the list of fields on the right.
To delete fields, drag them out away from the other fields.



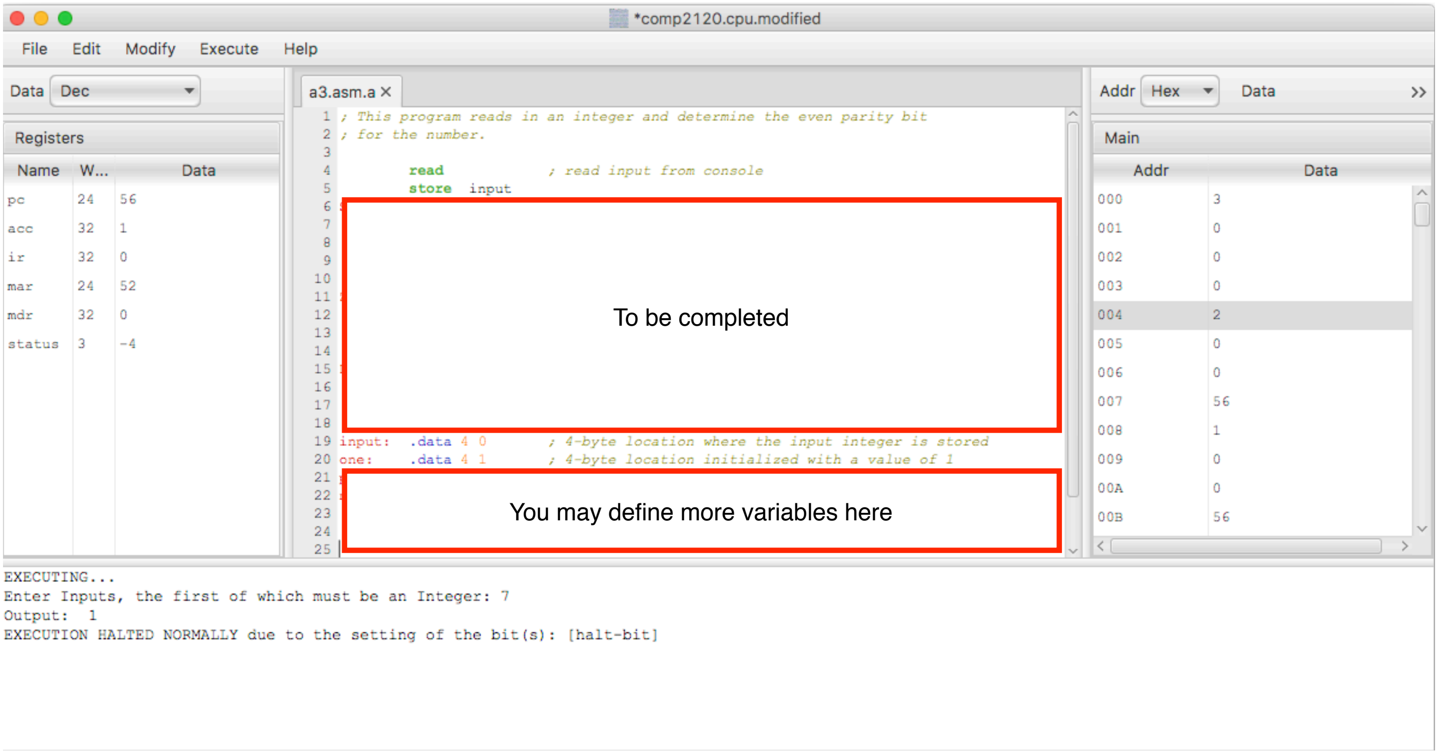
To be completed

6. Add a new machine instruction "***dec***" with 8-bit opcode 0x0D, such that it takes one operand from 24-bit memory address *addr*, and increment the integer value and store the result back to the same memory address *addr*. (6%)

7. Add a new machine instruction "***shlr***" with 8-bit opcode 0x0E, such that it takes one operand from 24-bit memory address *addr*, and shift the bit-pattern logically to right by 1 bit and store the result back to the same memory address *addr*. (6%)

8. Add a new machine instruction "**and**" with 8-bit opcode 0x0F, such that it takes one operand from 24-bit memory address *addr*, and perform the bit-wise AND operation between the value stored at *addr* and *acc* register. The result has to be stored back to the *acc* register. (5%)

9. Write an assembly program, which read in one 32-bit integer from console, and determine the parity bit for this integer so that the total number of ‘1’s is always even. This parity bit will be output to the console. You can assume the input value is always a 32-bit integer and so no range check is necessary. (16%)



Registers

Name	W...	Data
pc	24	56
acc	32	1
ir	32	0
mar	24	52
mdr	32	0
status	3	-4

```
; This program reads in an integer and determine the even parity bit
; for the number.

        read            ; read input from console
        store  input
```

To be completed

```
input:  .data 4 0       ; 4-byte location where the input integer is stored
one:    .data 4 1       ; 4-byte location initialized with a value of 1
```

You may define more variables here

Main

Addr	Data
000	3
001	0
002	0
003	0
004	2
005	0
006	0
007	56
008	1
009	0
00A	0
00B	56

```
EXECUTING...
Enter Inputs, the first of which must be an Integer: 7
Output:  1
EXECUTION HALTED NORMALLY due to the setting of the bit(s): [halt-bit]
```