Department of Computer Science and Engineering
The Chinese University of Hong Kong

ENGG 5101/CENG 5410: Advanced Computer Architecture

Assignment One (Deadline: 23:55, February 23, 2019)

Note: This is an individual assignment and please submit it via the Blackboard before the deadline.

In Tutorials 1 and 2, we have discussed one simple Von-Neumann simulator and demonstrated how to schedule two processes based on it. In this assignment, you have two tasks: 1) Implement the simulator with the interrupt; 2) Revised the given simple OS so it can schedule four processes.

- **Task One - Simulator:** As discussed, the simulator can execute a program based on the instruction set listed below. Specifically, inside the CPU, there is a counter and it will be incremented by one in each instruction cycle. In every 5000 cycles, a timer interrupt will be triggered. When a timer interrupt occurs, the corresponding bit of PSR will be set if the interrupt is enabled. At the end of each CPU cycle, we need to check if there is a pending interrupt (if the interrupt bit in PSR is set) and jump to the interrupt handler if the interrupt is enabled.
- **Task Two - OS**: Revised the given OS so it can manage four processes. Specifically, the OS will perform process scheduling in the interrupt handler in a round-robin manner so four processes can print 'A', 'B', 'C', and 'D', respectively, once scheduled (as shown in the given OS with two processes).

More details can be found from the slides of Tutorial 1-2. To run the programs provided in the tutorials, you may use Linux workstations in the CSE department, or use VirtualBox to set up a 32-bit Xubuntu Linux environment based on the instruction and VM image below:

https://lumian2015.github.io/linuxBasic/
https://www.cse.cuhk.edu.hk/~shao/zili_files/csci3150/3150_XUbuntu.ova

In *simulator-interrupt.c*, a C program skeleton is provided and what you need to do is to implement all functions (i.e. fetch(), decode(), execute(), timer_tick(), and check_interrupt()) accordingly. For *lab-os.asm*, you need to revise it so it can schedule four processes.

What to submit: A zip file containing: (1) **simulator-interrupt.c** (the simulator); (2) **4p-os.asm** (the OS); (3) Readme.txt (for how to compile and run your program). Your programs should be able to be tested as below:

```
gcc -o  icpu simulator-interrupt.c -Wall
./assembler 4p-os.asm ios
./icpu ios xx  (xx is the entry address you provide to run the revised OS).
```

Note:

1) For the simulator, you can use other programming languages as long as all functions can be provided to run the revised OS.
2) *icpu* and *assembler* provided are tested based on the 32-bit XUbuntu.

Appendix – Instruction Set

halt (Stop the CPU and exit):
Machine code: 0x00000000

NOP (No operation):
Machine code: 0x01000000

addi (Add the immediate value and the source register, and store the result to the target register):
Machine code: 0x02xxyyzz
Effect: R[yy] <- R[xx]+zz; PC <- PC+1
Description: R[yy] is R[xx]+zz; PC is incremented by 1

move_reg (Move from the source register to the target register):
Machine code: 0x03xxyy00
Effect: R[yy] <- R[xx]; PC <- PC+1
Description: R[yy] is equal to R[xx]; PC is incremented by 1

movei (Move the immediate value to the target register):
Machine code: 0x0400yyzz
Effect: R[yy] <- zz; PC <- PC+1
Description: R[yy] is equal to zz; PC is incremented by 1

lw (Load a word to the target register from the memory address obtained by the summation of the source register and the immediate value):
Machine code: 0x05xxyyzz
Effect: R[yy] <- M{ R[xx] + zz}; PC <- PC+1
Description: R[yy] is equal to the content of the memory at the address obtained by R[xx]+zz; PC is incremented by 1

sw (Store the content of the target register to the memory address obtained by the summation of the source register and the immediate value):
Machine code: 0x06xxyyzz
Effect: M{R[xx]+zz} <- R[yy]; PC <- PC+1
Description: The content of the memory at the address obtained by R[xx]+zz is equal to R[yy]; PC is incremented by 1

blez (Branch if the source register is less than or equal to zero):
Machine code: 0x07xx00zz
Effect:
If R[xx] <= 0
    PC <- PC + 1 + zz
else
    PC <- PC +1
Description: If R[xx] is less than or equal to zero, PC is equal to PC+1+zz; otherwise, PC is equal to PC+1.

la: Load address
Assembly: la Ryy, zz
Format: 0x0800yyzz;
Effect: R[yy] <- PC + 1 + zz; PC <- PC+1

add: Add Rxx to Ryy
Assembly: add Rxx, Ryy
Format: 0x0bxxyy00
Effect: R[yy] <- R[xx] + R[yy]; PC <- PC+1

jmp: Unconditional jump
Assembly: jmp zz
Format: 0x0c0000zz
Effect: PC <- PC + 1 + zz

push: Push a register onto the stack
Assembly: push Rxx
Format: 0x09xx0000
Effect: SP <- SP-1;
M{SP} <- R[xx];
PC <- PC+1

pop: Pop a word from the stack to a register
Assembly: pop Ryy
Format: 0x0a00yy00
Effect: R[yy] <- M{SP};
SP <- SP+1; PC <- PC+1

iret: Interrupt return
Assembly: iret
Format: 0x10000000
Effect: PC <- Pop(); PSR <- Pop()

put: Print on screen
Assembly: put Rxx
Format: 0x11xx0000
Effect: print R[xx]; PC<- PC+1