Assignment 1 - EECS 211

Basics of Context Switching

Winter 2022

Problem 1

Currently, the LC3 VM implements the "Trap" and "Interrupt" handling differently than how the hardware actually works. However, since the OS context switching depends mainly on the interrupts and traps, it is important to re-implement this part in the LC3 VM to match what happens in an actual hardware.

A trap occurs whenever the assembly instruction "TRAP (op code = 0xF)" is used. The assembly instruction also contains a number called the trap vector (trapvect8). When the CPU executes the trap, it performs the following two steps:

- The value in the program counter (PC) register will be copied to R7
- The PC will point to the address saved in Memory[trapvect8].

Let's consider the example below:

Listing 1: Assembly Code # 1

```
        .ORIG   0x0025
        .FILL TRAP_HALT

        .ORIG 0x0100
TRAP_HALT
            ....                ; put your Trap Service Routine code here



            .ORIG 0x3000
            ....                ; the software code starts from here
            TRAP 0x25           ; this is exactly equivalent to .HALT
```

Recall that the HALT trap has a trapvect8 equal to 0x25. In the example above, the value of 0x0100 is saved inside the memory location 0x25. Therefore, executing the HALT trap assembly instruction will force the PC will point to the address 0x0100.

In this problem, make the following changes to the LC3 VM:

1. Change the LC3 VM such to implement the right sequence of actions for the traps. Recall this is the trap vector for the LC3 VM:
   - TRAP_GETC = 0x20, /* get character from keyboard, not echoed onto the terminal */
   - TRAP_OUT = 0x21, /* output a character */
   - TRAP_PUTS = 0x22, /* output a word string */
   - TRAP_IN = 0x23, /* get character from keyboard, echoed onto the terminal */
   - TRAP_PUTSP = 0x24, /* output a byte string */
   - TRAP_HALT = 0x25 /* halt the program */
2. If the software uses the TRAP instruction with any other value for trapvect8 (other than the ones above), your VM should dump the memory/registers to a file (using the same template used in the previous assignment).
3. Now you can write code for each trap service routines (which is considered the first part of your OS). The code for the trap service routines will be located at 0x0100. For now, will have very simple trap service routines that simply uses TRAP 0xFF to dump the memory followed by an infinite loop.

To test your code, use the simple code below:

Listing 2: Assembly Code # 2

```
            .ORIG 0x3000
            TRAP 0x25           ; this is exactly equivalent to .HALT
```

Deliverable

You need to upload one file named "memory_dump_p1_1" that contains the memory/registers for the assembly code #2 above.

Problem 2

In this problem, we are going to edit the HALT trap to implement a simple OS scheduler. This OS scheduler will go through the available processes and start executing them in order.

To simplify the implementation, we will make the following assumptions. First, the maximum number of processes that can be loaded in the memory is equal to 5. The OS will keep track of the state of each process using the following data structure:

- Status of the first process (unsigned integer 8)

- Status of the second process (unsigned integer 8)
- Status of the third process (unsigned integer 8)
- Status of the fourth process (unsigned integer 8)
- Status of the fifth process (unsigned integer 8)

A value of 1 means that the corresponding process is ready to run, a value of 2 means that this process is the one that is currently running, and a value of 0 means that the corresponding process has terminated. This data structure should be saved in the location 0x0200 which marks the beginning of the OS stack section.

The scheduler should go through the state variables above (in order), and once it finds a process that is ready to execute, it will jump to the code of the corresponding process. When the process halts (by using the HALT trap), the OS will change the status of this process from 1 to 0 and then search for another process to run. To simplify the implementation, we will assume the code of the first process will be located in 0x4000, the second process will be located in 0x5000, the third process will be located in 0x6000, and so on. If no process needs to run, then the scheduler will enter an infinite loop.

To debug the scheduler and make sure it is implemented correctly, the scheduler should force a dump of the memory at the very beginning of the HALT trap service routine (by using TRAP 0xFF instruction) and at the very end (just before it switches to the next process).

The main body of the OS is located at 0x3000 (where the CPU starts executing). For this simple OS, it just needs to initialize the values in the state variables followed by forcing the scheduler to operate by using a trap instruction.

Deliverable

Write the code of 5 simple (dummy) processes that will be located in 0x4000, 0x5000, 0x6000, 0x7000, and 0x8000, respectively. These processes should perform the following instructions:

1. Load a certain value in the register R1. For example, the first process should load the value of “1“ into the register R1, the second process should load the value of “2” into the register R1, and so on.
2. Halt.

Implement the scheduler described above to switch between these five processes. You will upload one file called “os.asm”, one zip file that contains your VM, a read-me file explaining how to use your VM, and 12 memory dump files called “memory_dump_p2_1”, “memory_dump_p2_2”, “memory_dump_p2_3”, ..., etc, that corresponds to the memory dumps issued during the switches between these processes.

Problem 3

In this problem, we are going to increase the set of allowed Traps. In particular, we would like to add a trap for "YIELD". Such trap can be used by the software to inform the OS that it would like to yield the processor for now so the OS can run another process/thread. The main difference between YIELD and HALT is that HALT means that the software finished all its instructions and does not need the processor anymore, while YIELD means that the process still needs the processor in the future.

- Edit the LC3 VM to add another trap called "TRAP_YIELD". This new trap should be assigned a trap vector of 0x26.
- Write a trap service routine for the TRAP_YIELD. This trap service routine should change the status of the process back to 1 (which means ready) followed by calling the scheduler function to pick a new process to execute.

Deliverable

Write the code of 5 simple (dummy) processes that will be located in 0x4000, 0x5000, 0x6000, 0x7000, and 0x8000, respectively. These processes should perform the following instructions:

1. Load a certain value in the register R1. For example, the first process should load the value of "1" into the register R1, the second process should load the value of "2" into the register R1, and so on.
2. Yield.
3. Increment the value of R1 by 2.
4. Halt.

Augment the scheduler to implement the Yield functionality. You will upload one file called "os.asm", one zip file that contains your VM, a read-me file explaining how to use your VM, and the first 12 memory dump files called "memory_dump_p3_1", "memory_dump_p3_2", "memory_dump_p3_3", ..., etc, that corresponds to the memory dumps issued during the switches between these processes.