Lab 3:
LC-3 Simulator Implementation

Assigned: Dec. 06, 2021
Due: Jan. 06, 2022

Instructor: Jingwen Leng
Teaching Assistant: Yue Guan, Zhihui Zhang

1 Introduction

This is the last but the largest lab about the LC-3 this semester. In this lab, you will write an instruction-level simulator for the LC-3. The simulator will take one input file entitled `isaprogram`, which is an assembled LC-3 program. The simulator should be able to execute the input LC-3 program, one instruction at a time and modify the architectural state of the LC-3 after each instruction.

The file `isaprogram` should consist of 4 hex characters per line. Each line of 4 hex characters should be prefixed with '0x'. For example, the instruction NOT R1, R6 is assembled to 1001001110111111. So this instruction would be represented in the program file as 0x93BF. To assemble a program into the above-mentioned format, you can follow the next two steps. First, use the assembler `lc3as` of the LC-3 simulator that you used in the previous labs.

```
lc3as example.asm
```

This will generate the machine code binary file `example.obj` (and the symbol table `example.sym`). Then use the following command to convert the binary file to the format for your simulator shell.

```
hexdump -v -e ' "0x" 2/1 "%02X" "\n" ' example.obj | tee isaprogram
```

As a result, this will generate the input file `isaprogram` with the correct format.

2 Instruction

The simulator is partitioned into two main sections: the shell and the simulation routines. We are providing you with the shell. Your job is to write the simulation routines.

2.1 The Shell

The purpose of the shell is to provide the user with commands to control the execution of the simulator. The shell accepts one or more ISA programs as arguments and loads them into the memory image. The address of the first ISA program listed at the command line is loaded into the PC before the simulation begins. If you compiled your simulator to an executable called `simulate`, then you can run the simulator by typing the following into your terminal:

```
./simulate <main_program_file> [extra_file] [extra_file] ...
```

In order to extract information from the simulator, a file named `dumpsim` will be created to hold the information requested from the simulator. The shell supports the following commands:

1. go: simulate the program until a HALT instruction is executed.
2. run <n>: simulate the execution of the machine for n instructions.
3. mdump <low> <high>: dump the contents of memory, from location low to location high to the screen and file.
4. rdump: dump the current instruction count, the contents of R0–R7, PC, and condition codes to the screen and file.
5. ?: print out a list of all shell commands.
6. quit: quit the shell

2.2 Simulation Routines

The simulation routines carry out the instruction-level simulation of the input LC-3 program. During the execution of an instruction, the simulator should take the current architectural state and modify it according to the ISA description in our appendix. The architectural state includes the PC, the general-purpose registers, the condition codes and the memory image. The state is modeled by the following C code:

```
#define WORDS_IN_MEM    0x08000
#define LC_3_REGS 8

typedef struct System_Latches_Struct{
  int PC;                /* program counter */
  int N;                 /* n condition bit */
  int Z;                 /* z condition bit */
  int P;                 /* p condition bit */
  int REGS[LC_3_REGS];   /* register file.  */
} System_Latches;

System_Latches CURRENT_LATCHES, NEXT_LATCHES;
int MEMORY[WORDS_IN_MEM][2];
```

The shell code we provide includes the skeleton of a function named process_instruction, which is called by the shell to simulate the next instruction. You have to write the code for process_instruction to simulate the execution of instructions. You can also write additional functions to make the simulation modular. You should read the current system state from the global variable CURRENT_LATCHES. The results of executing the current instruction should be written to the global variable NEXT_LATCHES.

2.3 What To Do

The shell code mentioned in Section 2.1 has been provided in our course website Architecture_Lab_3.zip/lc3sim.c, which is able to read in the input program and initialize the LC-3 machine state.

It is your responsibility to complete the simulation routines in the `lc3sim.c` that simulate the instruction execution of the LC-3. Add your code to the end of the shell code. Do not modify the shell code. The accuracy of your simulator is your main priority. Specifically, make sure the architectural state is correctly updated after the execution of each instruction.

It is also your responsibility to verify that your simulator is working correctly. You should write one or more programs using all of the LC-3 instructions and execute them one instruction at a time. You can use the rdump command to verify that the state of the machine is updated correctly after the execution of each instruction.

You should be sure that your code compiles on the linux machines using gcc with the `-std=c99` flag. This means that you need to write your code in C such that it conforms to the C99 standard.

```
gcc -std=c99 -o simulate lc3sim.c
```

If you decide to use any of the math functions in math.h, you also have to link the math library by using the command:

```
gcc -lm -std=c99 -o simulate lc3sim.c
```

You can choose any Linux distribution and install it any where you like for this lab. But we recommend Ubuntu Desktop installed on a virtual machine (e.g., VMware, VirtualBox) if you do not have a preference, which is the most secure and convenient way to run and manage a Linux together with your own operating system.

2.4 Notes and Clarifications

1. You need to implement the TRAP instruction as the LC-3 ISA defines in Appendix A. However, you do not need to implement the TRAP routines. The trap vector table will be initialized to all zeroes by the shell code provided. Thus, whenever a TRAP instruction is processed, PC will be set to 0. The shell code provided will halt the simulator whenever PC becomes 0.

2. You do not have to implement the RTI instruction for this lab. You can assume that the input file to your simulator will not contain any RTI instructions.
3. Please remove any print statements you add to the program before submission. Do not remove print statements listed in the shell code.
4. For this assignment, you can assume that the programmer will always give aligned addresses/valid opcodes, and your simulator does not need to worry about unaligned cases.
5. You do not need to implement memory-mapped I/O for this lab.
6. You may assume that the code running on your simulator has been assembled correctly and that the instructions your simulator sees comply with the ISA specification.
7. In your code that you write for lab 2, do not assign the current latches to the next latches. This is already done in the shell code.
8. The Low16bits macro provided in the shell code zeroes out the top 16 bits of its argument, like so:

   ```
   #define Low16bits(x) ((x) & 0xFFFF)
   ```

   You may use this macro to avoid getting values like 0xFFFFFFFF in architectural registers. The variables in your C program are 32-bit values, so the number -1 is 0xFFFFFFFF. However, the LC-3 is a 16-bit machine, in which -1 is represented as 0xFFFF. Thus, you have to make sure that when you store a value in a variable representing an LC-3 register, you mask it properly (for example, using the macro given above).
9. LEA should NOT set the machine's condition codes, despite what the reference textbooks may say.

3 Submission

Please submit the single gcc-compilable file of `lc3sim.c` to the course website without other project files if any. You are also supposed to write and submit a FORMAL report as a pdf file that describes (1) the control logic, (2) important functions you think in the implementation, and (3) the verification/testing of your LC-3 simulator.

NOTE: the score in this lab depends mainly on your report. But we will still check the source code for functional correctness and plagiarism. So please be careful and do it by yourself.