Virginia Tech
Bradley Department of Electrical and Computer Engineering
ECE 3574: Applied Software Engineering ◆ Spring 2013

Homework 4

Submission Details

Your solution for this homework must be submitted as an electronic submission. Your submission will be a tarred and gzipped file containing the source files for the programming problem. These files must be in a directory entitled HW4_*LAST_FIRST* where LAST is your last or family name and FIRST is your first or given name. Your program will be run to evaluate its correctness, and the source code will be reviewed for adherence to Qt programming style. Your program must run on the latest version of Ubuntu and be compiled/built using the GNU C/C++ compiler and qmake. The following information must be included in your source files: your full name, your student ID number, your email address, class (ECE 3574), title of the assignment (Homework 4), and date of submission. You must submit the solutions using the "Submission" button on HW 4 assignment page on the Scholar class web site. The file must be given a name of the following form: HW4_*LAST_FIRST*.tar.gz where LAST and FIRST are as specified above. You can make multiple submissions. Paper and email submissions will not be accepted. All work must be submitted by the announced due date/time.

Honor Code

As stated in the syllabus, in working on homework and projects, discussion and cooperative learning are allowed. However, copying or otherwise using another person's detailed solutions to assigned problems is an honor code violation. See syllabus for details.

Homework 4:

For this homework you will build a simulator for a simple computer architecture and GUI for this simulator. The computer architecture and the GUI are described in the next two sections.

For this homework, the classes that you develop should all be QObjects. Interactions between classes should use signals and slots.

Please note that Homework 5 will build upon this homework. Thus, you should not use this homework as the one homework that you plan to drop for your course grade.

The Computer Architecture

The simulator models a particularly simple computer architecture. Our architecture has no data registers, just an accumulator. The computer has two memories, an instruction memory and a memory for storing data.

We won't worry about how the instruction memory encodes instructions; we will just assume that each instruction can fit in one word of instruction memory. We will represent (and display) each instruction as a character string as specified below. The computer includes a program counter register called "PC". The PC holds the address of the next instruction to be executed in the instruction memory. The instruction memory can contain 128 instructions, numbered from 0 to 127. The first instruction executed by any program is 0. The computer increments the PC by one after executing an instruction (except for the case of the branch and halt instructions).

The computer also has a data memory which has 512 locations, numbered from 0 to 511. Each location can store 10 bits of data. For our computer programs, you can assume that the data stored in these memory locations is always a 10-bit signed integer (stored as twos complement). For the purpose of displaying the data, you can always show this data as a signed decimal integer. For displaying the memory locations, you

should always show the memory locations in hexadecimal (e.g., data address "0x00A" is memory address 10 in decimal).

The computer has one data register called the accumulator, or AC. The AC can contain one 10-bit word. Again, for our purposes one can assume that it contains a 10-bit signed integer.

The Instruction Set Architecture (ISA)

The architecture has the following instructions:

LOAD <hex-data-address>

STORE <hex-data-address>

ADDV <signed-decimal-number>

ADD <hex-data-address>

BGTZ <decimal-instruction-address>

NOP

CLEAR

HALT

OUT <string-with-possible-accumulator-value>

One can write a program using these instructions as a text file, then load this program into the instruction memory, and then run the program in the simulator. The definitions of each instruction is the following:

- The LOAD instruction loads the value from the memory address <hex-data-address> into the accumulator. For example, "LOAD 1FF" loads the value stored in location 0x1FF of the data memory into the accumulator.
- The STORE instruction stores the value in the accumulator to the memory address <hex-data-address> in memory. The addresses in <hex-data-address> are given in hexadecimal in the program.
- The ADDV instruction adds the signed decimal number <signed-decimal-number> to the value in the accumulator. For example, if the accumulator had the value 10, then executing the instruction "ADDV -2" would result in the value 8 being in the accumulator.
- The ADD instruction adds the value stored at the address <hex-data-address> to the value in the accumulator. For example, if the accumulator had the value 10, and the value 20 was stored in memory location 1FF, then executing the instruction "ADD 1FF" would result in the value 30 being in the accumulator.
- The BGTZ instruction sets the program counter to the value given by the <decimal-instruction-address> if the value of the accumulator is greater than 0. If the value in the accumulator is less than or equal to zero, the program counter is incremented by 1. For example, executing the instruction "BGTZ 10" would set the program counter to 10 if the value in the accumulator was greater than zero. Otherwise, the program counter would be incremented by one.
- The NOP is the "no op" instruction that does nothing but increment the program counter by one.
- The CLEAR instruction sets the value in the accumulator to 0.
- The HALT instruction halts the computer. The HALT instruction should always be the last instruction of a program. After the HALT instruction, the program counter stops incrementing.
- The OUT function outputs a string to standard output. There is a special symbol "%A" to allow one to print out the value in the accumulator. For example, if the value in the accumulator was 100, the instruction:

  OUT "The value in the accumulator is %A."

Would print the string

The value in the accumulator is 100.

To standard output.

The Simulator GUI

You will need to design a GUI that displays the computer's state, including both memories (instruction and data), and the state of the program counter and accumulator. The simulator is started from a terminal window with the command:

./computerSim

The program can be read in from a GUI "File" menu entry as before. Whenever a new program file is read in, the instruction and data memory should be cleared and the program counter and accumulator should be set to 0.

This GUI should have the following features.

1. The GUI should have a "Step" button or menu item that allows the user to step through the program. Once a program is read into memory, the GUI should be ready to execute the first instruction in the program. The step button executes the instruction and updates the computer based on the instruction executed.
2. The GUI should have a "Run" button or menu item that allows the user to run the entire program, stopping at the HALT instruction.
3. In the case of an error in the program (e.g., an instruction that doesn't make sense), the program should halt at that instruction and print an error to standard out.
4. The GUI should display the current contents of the data memory (initially all memory values should be zero).
5. The GUI should display the contents of the instruction memory. The instruction that corresponds to the value of the program counter should be indicated in the GUI. . When the program gets to the HALT instruction the program counter stops incrementing, so the GUI should continue to show the next instruction as the HALT.
6. The GUI should have a window for displaying the output to standard out (i.e., for program error messages and for output from the OUT instruction).

The overall design of this GUI is completely up to you. You may either implement your QUI completely in Qt code, or you can use Qt Designer to design the layout.

Grading Policy

- QT style programming and indentation – 20%.
- The code complies and runs – 0%.
- The code satisfies the specifications – 60%.
- The design your GUI interface – 20%.