CS-UY 2214 — Homework 4

Jeff Epstein

Introduction

Unless otherwise specified, put your answers in a plain text file named `hw4.txt`. Number each answer. Submit your work on Gradescope.

You may consult the E15 cheat sheet, which is available on Brightspace. You may also consult the Verilog reference materials in the same place. You may consult the E20 manual.

Problems

E15 questions

1. Consider the following fragment of an E15 assembly language program:

   ```
   myROM[14] = { cmp, Rg0, Rg0, 4'b0000 };
   myROM[15] = { jnz, RXX, RXX, 4'b1110 };
   ```

   What is the address of the instruction that will be executed immediately after the instruction at address 15? If the address of the next instruction cannot be determined from the information given, say so.

2. In E15 assembly language, there is no single instruction to perform "greater than (or equal to)" and "less than (or equal to)" operations. We have only the `cmp` and `cmpi` instructions to test for equality of two values.

   Write a program in E15 assembly language in a file named `leq.v` that tests whether the value in `Rg0` is less than or equal to the value in `Rg1`. Assume that the values in both registers are unsigned numbers. If the test is true (i.e. if `Rg0` is less than or equal to `Rg1`), then set `Rg2` to 1, otherwise to 0.

   Let's initially assume that we want to compare the decimal values 7 and 2. Use ROM location 0 to store 7 into Rg0 with the `movi` instruction; then use ROM location 1 to store the value 2 into Rg1.

   The remainder of the program, beginning at ROM location 2, should perform the comparison of the value of Rg0 by the value of Rg1, and store the result into Rg2 before halting. Note that your program should work with arbitrary inputs, but for testing and grading purposes, make sure that your submitted code is for the specific values 7 and 2.

   When your program has finished, use the usual `{jmp, RXX,RXX, 4'b0000}` instruction to halt execution.

   Test your solution with the E15 implementation. You'll need to modify the provided `E15Process.v` file in order to make it load your assembly program. Look for the line `` `include "program1.v" `` and change the filename to match that of your program (`leq.v`). Do not change the file `E15Process.v` in any other way. Test your code by compiling and running the `E15Process_tb.v` test bench and examining its output: when you run the test bench, it will print out the final state of the registers.

   Briefly describe how your solution works in a few sentences.

   Submit your E15 assembly language file `leq.v`. Do not submit your modified `E15Process.v` or the test bench.

3. As a developer of software for the E15 processor, you are often frustrated by some of its design limitations. It has technical limits that make it infeasible for executing certain kinds of software. For example, try to imagine writing a web browser for the E15.

   In particular, consider this hardware limitation: the instruction store for E15 programs (i.e. the `myROM` array) is limited to a maximum of sixteen instructions. This means that it is impossible to write a program containing more than sixteen instructions. Obviously, this limitation means that many programs simply cannot be written for E15 hardware.

   Your task is to design a new processor (the "E15 Turbo"), based on the E15, that has a capacity for up to 1024 instructions. Starting with the E15 design provided to you, you will propose changes to accommodate the larger instruction store. This modification requires extensive modification to various parts of the processor. For each of the following areas of the processor, discuss what changes will be necessary. If no changes are needed to that part of the processor, say so, and explain why.

   There may be multiple possible answers for each part of the question, depending on what approach you take. You may consult the E15 processor's source code to help you. You may also cite the source code or use Verilog code to describe particular changes in your answer. For example, your answer may be in the form of a proposed "before and after" comparison of code. In all answers, credit will be given for a thorough answer that demonstrates an understanding of the consequences of a given change, both its pros and its cons. Your answer must include a justification of your proposal. Base your proposal on the provided E15 implementation. For all questions, be specific.

   This is a *design* question, not a *coding* question. Although you may use Verilog code to illustrate key facets of your design proposal, you are not required to do so, and you should *not* submit a complete source code implementation of your proposal. The only work required is English-language answers to the following questions.

   (a) What changes will need to be made to the program counter (`pc`)? Why?

   (b) Currently, the instruction format is 12 bits: four bits for the opcode, two bits for the source register, two bits for the destination register, and four bits for the immediate. What changes will be needed to the instruction format? Why? Consider all instructions in your answer.

   (c) What changes will need to be made to the general-purpose registers (Rg0..Rg3)? Why?

   (d) What changes will be needed to the instruction store (`myROM`)? Why?

   (e) Currently, both ALUs use 4-bit inputs and a 4-bit output. What changes will be needed to the ALUs? Why?

E20 questions

4. Consider the following E20 program:

```
    movi $4, 9
    addi $4, $4, 5
    jeq $4, $0, xyz
    movi $1, 0
    halt
xyz:
    movi $1, 1
    halt
abc:
```

   (a) What is the final value of the program counter?

   (b) What is the final value of the `$1` register?

(c) What is the value of the label `xyz`?

(d) What is the value of the label `abc`?

(e) What is the address of the `jeq` instruction?

(f) What will be the value of the immediate field of the `jeq` machine code instruction? Consult the E20 manual. Don't forget that in E20 machine code, the `jeq` instruction's immediate is a *relative* address. Your answer will depend on the address of the `jeq` instruction and also on the address of `xyz`. Give your answer as a 7-bit binary number.

5. Write E20 assembly instructions as described. When directed to write "a pair" of instructions, you should write two instructions that, when executed in sequence, will have the described effect.

   Give each solution as both E20 assembly language program and the equivalent E20 machine language representation. Give each machine language instruction as a 16-bit binary number. Write each instruction on a separate line. Write all 16 digits of each number.

   (a) Write an E20 instruction that will jump to the address identified by label `target`.

   When translating to machine language, assume that `target` has address 3, and that the current program counter is 9.

   (b) Write an E20 instruction that will jump to the address identified by label `target`, if register `$4` is zero, and otherwise will proceed to the next instruction.

   When translating to machine language, assume that `target` has address 3, and that the current program counter is 9.

   (c) Write a pair of E20 instructions that will set the value of the memory cell identified by label `isless` to 1 when the value of `$4` is less than 10 (decimal), and to 0 otherwise.

   When translating to machine language, assume that `isless` has address 3.

   (d) Write a pair of E20 instructions that will move the value stored at memory cell 29 into memory cell 30.

6. Write an E20 assembly language program that will add 1 to register `$1`, then multiply register `$2` by 2. Don't forget to include a `halt` at the end.

   Translate your E20 assembly language program into the equivalent machine language representation. Give each instruction as a 16-bit binary number. Write each instruction on a separate line. Write all 16 digits of each number.

   Give your answer both in E20 assembly language (using correct E20 syntax) and E20 machine language (as a sequence of 16-bit binary numbers).

7. Write an E20 assembly language program that will read the values of memory cells 20 and 30, and store their sum into memory cell 40. Don't forget to include a `halt` at the end.

   Give your answer only in E20 assembly language (using correct E20 syntax).

8. Write an E20 assembly language program that will add 1 to each of the memory cells between 10 and 20, inclusive. Use a loop. Don't forget to include a `halt` at the end.

   Translate your E20 assembly language program into the equivalent machine language representation. Give each instruction as a 16-bit binary number. Write each instruction on a separate line. Write all 16 digits of each number.

   Give your answer both in E20 assembly language (using correct E20 syntax) and E20 machine language (as a sequence of 16-bit binary numbers).