CSCI 2321 (Principles of Computer Design), Spring 2015
Homework X

Credit: Up to 50 extra-credit points.

1 Overview

The problems below are a mix of the well-defined and the open-ended. You can receive at most 50 extra-credit points, but credit for the more open-ended problems is similarly open-ended and depends on the level of effort or difficulty. As with previous assignments, anything you send me by e-mail should have something in the Subject line that mentions the course ("csci 2321" or "computer design", e.g.) and the assignment ("extra credit").

NOTE that the usual rules for collaboration do not apply to this assignment. Please do *not* work with others, even to the extent (general discussion) allowed for regular homeworks.

2 Problems

Do as many of the following as you like. You can turn in hardcopy (put it in one of my mailboxes) or send me by e-mail me something I can print (PDF preferred, but anything I can reasonably print from Linux is okay).

1. (Up to 5 points)

   Do the extra-credit problem from Homework 3:

   Do problems 3.12 and 3.18 from the textbook. Problem 3.18 is in my opinion somewhat ambiguously stated, but I think the two inputs (74 and 21) are meant to be interpreted as base-8 numbers.

   (Does it go without saying that you can't get credit for these problems both as part of Homework 3 and as part of this assignment? It should!)

2. (Up to 5 points)

   (In the style of the textbook: This problem explores changes to the single-cycle implementation needed to support the `jr` and `jal` instructions.)

   Section 4.4 of the textbook discusses how information flows through the datapath of Figure 4.17 for three instructions (R-format, `lw`, and `beq` — this is the discussion illustrated by Figures 4.19 through 4.24)). There is also a discussion of what must be added to support the `j` instruction, culminating in Figure 4.24.

   For each of the specified instructions (`jr` and `jal`), first describe (as a numbered list of steps) what execution of the instruction needs to involve, and then discuss what changes (if any) you would need to make to the design shown in Figure 4.24 to make it work. (If you need to make changes to the figure, it might be clearest to print/photocopy the figure and mark it up.)

3. (Up to 5 points)

   (In the style of the textbook: This problem explores changes to the single-cycle implementation needed to support exceptions.)

   Section 4.9 of the textbook discusses changes to the pipelined implementation needed to support exceptions. Describe how you could adapt this approach to the single-cycle implementation — that is, describe what changes you would need to make to the design shown in Figure 4.24. (If you need to make changes to the figure, it might be clearest to print/photocopy the figure and mark it up.)

4. (No maximum, though as a rough guideline a page or so of prose will likely get you about 5 points.)

   In this course we focused on the MIPS architecture and its assembly language because it's simple and regular, and in theory once you have this background you should be well-prepared to learn about other architectures and their assembly languages. Choose some other architecture (x86 comes to mind, but there are others) and write a one-page-or-so executive-level summary of how it compares to the MIPS architecture (e.g., does it also have a notion of general-purpose registers, what if any special-purposes registers does it have, how do (some of) the instructions compare to those used in MIPS, etc.). Include a list of the sources you consulted (parts of the textbook, Web sites, etc.) You can even do this more than once for several different architectures.

5. (No maximum, though as a rough guideline a page or so of prose will likely get you about 5 points.)

   For testing MIPS assembler programs we used a simple emulator (SPIM). Based on a very quick Google search it appears that there are other tools that provide similar or greater functionality (cross-compilers that generate MIPS assembler or object code from C code. full-fledged virtual machines that implement the MIPS architecture.) Find one or more that seem to you likely to be useful for this course and explain why you think it would be useful and what would be involved in installing it.

3 Programming Problems

Do as many of the following as you like; submit your program(s) by e-mail, with each source-code file as an attachment.

1. Write a complete MIPS program to do something you think is (at least a bit) interesting and doable. Your program should consist of a single `.s` file, e.g., `mypgm.s`, and should be runnable using SPIM with the command `spim -f mypgm.s`. How much credit you get depends on the difficulty of the problem. Some ideas:

   - (Up to 5 points.)

     (Inspired by problem 2.37 in the textbook.) Convert a text string meant to represent a hexadecimal value to a (32-bit) integer. For this program, write a MIPS procedure (using the standard conventions for called procedures described in the text) that takes one argument, the address of a text string meant to represent a non-negative hexadecimal value, converts it a 32-bit integer value, and returns the result, or -1 if the text string does not contain something that can be converted as desired. (For example, for the string "10" the procedure should return 16, and for "1E" it should return 30; for "hello" or "-2" it should return -1.) Then add a main program that prompts the user for a line of text, calls your procedure to convert it to an integer, and prints the result.

- (Up to 5 points.)

  Display a 32-bit value in hexadecimal notation. For this program, write a MIPS procedure (using the standard conventions for called procedures described in the text) that takes one argument, a 32-bit value, and displays it in hexadecimal form on the simulated standard output. (For example, 2 should print as 00000002, and -2 should print as FFFF FFFE.) Then add a main program that prompts the user for an integer and calls your procedure to print it in hexadecimal.

2. Some of the homeworks and exams had you do things that (should?) seem very automatable.

   - Converting a number in decimal form to a text form of its IEEE-754 representation, or vice versa.
   - Converting a line or lines of MIPS assembler to a text form of its binary representation, or vice versa. (This might take a while if you want to support all instructions, but you could choose to accept a subset, though obviously(?) the more you do the more credit you can get.)

   Write a program in a high-level language to perform one of these tasks (or some other task you had to do as part of a homework assignment or quiz and that you think is similarly automatable). You can use any high-level language I can easily test from the command line on one of our classroom/lab Linux systems. (For many of you Scala might be a good choice, though C++ might appeal to some, or even straight C.) Number of points here depends on length and difficulty of the program; as rough guideline, something comparable to one of the MIPS programs described above would likely get you about 5 points.