# CSCI 2321 (Computer Design), Spring 2021

# Homework 6

**Credit:** 40 points.

## 1 Reading

Be sure you have read, or at least skimmed, Chapter 4 up through section 4.4.

## 2 Problems

Answer the following questions. You may write out your answers by hand and scan them, or you may use a word processor or other program, but please submit a PDF or plain text via e-mail to my TMail address. (No links to shared files on Google Drive please.) Please use a subject line that mentions the course and the assignment (e.g., "csci 2321 hw 6" or "computer design hw 6").

- In this assignment your mission is to trace through what happens during execution of different instructions using the single-cycle implementation of the MIPS architecture as represented in Figure 4.17 in the textbook. You are to assume that at the beginning of a clock cycle the following is true for each of the instructions. (As usual, a value of the form `0xN` denotes a base-16 value of `N`, e.g., `0x10` denotes 16 in base 10.)

  - The program counter (PC) has a value of `0x4`.
  - Location `0x4` in the instruction memory contains the binary representation of the MIPS assembler instruction in question.
  - Register and data-memory contents are as described for individual instructions.

  At the point at which values are written into state elements, what values will the following have?

  - Input and output of the block labeled `PC`.
  - Input and output of instruction memory.
  - Inputs and output of the two adders at the top of the diagram (they don't have names, so let's call them the top-left and top-right adders).
  - All control signals output from the logic block labeled `Control` (`RegDst`, `ALUSrc`, etc.); get these from Figure 4.18.
  - Inputs and outputs of register file (`Read register 1`, `Read data 1`, etc.).
  - Inputs and output of the main ALU. Inputs include control signal `ALU control` (output of the block labeled `ALU control`, from Figure 4.13). Outputs include control signal `Zero`. (Meanings of `ALU control` are given in Figure 4.12.)
  - Inputs and output of data memory.
  - Any values changed in state elements:
    * For the PC, give the new value.
    * If any registers are changed, give their number and new contents.

  - If anything in data memory is changed, give its address and new contents, the address as a hexadecimal number and the new contents as base-10 or hexadecimal.

You should show these in whatever form makes the most sense:

- For addresses and instructions, hexadecimal.
- For register numbers or data values (e.g.., contents of a register), base 10 or hexadecimal (e.g. for 10 you can either write 10 or `0xa`).
- For other parts of the instruction (e.g., opcode), a bit string.

Or if a value doesn't make any difference to what is saved into the state elements, just write "not used" (e.g., if `RegWrite` is zero, the values of `Write register` and `Write data` are not used). *RESOURCES* below includes a template file that lists everything you need to fill in, plus the examples from the April 14 lecture. *Note* that in order to determine output of the ALU you have to know what the various values of `ALU control` mean. This information is shown in Figure 4.12, and also in Appendix B.

*RESOURCES:*

- Template for answers `hw07-template.txt`.
- `add` example `hw07-add.txt`.
- `lw` example `hw07-lw.txt`.
- `beq` example `hw07-beq.txt`.

*HINTS:*

- My suggestion for how to proceed is to use the approach I use in the example from the April 14 lecture: First write down what you know (current PC). Then start filling in fields: Output of the PC element is its current value; this feeds into the instruction memory and lets you determine output from the instruction memory (the questions give you its machine-language form). You can now write down inputs and outputs of the top-left adder, some inputs to the register file (the two read registers), and the main control-logic block. From those you can get more values, continuing until everything is filled in.
- I think you will learn more if you try to go strictly by what the circuit (in the figure) does rather than trying to apply what you know about what should happen, though when you're done the results should be consistent with what you think should happen! (E.g., if the instruction is supposed to store $n$ in register $r$ `RegWrite` should be 1, the write register number should be $r$, and the data input to the register file should be $n$.

(Yes, this is tedious, but I think it's the only way to really understand how this all works.)

Instructions to trace:

1. (7.5 points)
   `sub $t2, $t0, $t1`, if `$t0` contains the value `0x6` and `$t1` contains the value `0x2`. (In machine language this is `0x01095022`.)

2. (7.5 points)
   `lw $t0, 4($s0)`, if `$s0` contains the value `0x10000000`, and the data memory starting at `0x10000000` contains the values `0x1`, `0x2`, `0x3`, and `0x4` (with each value occupying 4 bytes). (In machine language this is `0x8e080004`.)
3. (7.5 points)
   `sw $t0, 4($s0)`, if `$s0` contains the value `0x10000000`, `$t0` contains the value `0x10`, and the data memory starting at `0x10000000` contains the values `0x1`, `0x2`, `0x3`, and `0x4` (with each value occupying 4 bytes). (In machine language this is `0xae080004`.)
4. (7.5 points)
   `beq $t0, $t1, LBL`, if `$t0` contains the value `0x2`, `$t1` contains the value `0x1`, and `LBL` corresponds to the instruction at location `0x18` in instruction memory (offset of `0x10` from updated PC).
5. (2.5 points)
   What would change during execution of the same `beq` as the preceding question, if `$t1` contained the value `0x2` instead of `0x1`.
6. (7.5 points) (For this problem, use Figure 4.24, and add a control signal `Jump`.)
   `j target`, if `target` is a label for the instruction at location `0x100`. (In machine language this is `0x08000040`.)

# 3 Pledge

For programming assignments, this section should go in the body of the e-mail or in a plain-text file `pledge.txt` (no word-processor files please). For written assignments, please put it in the text or PDF file with your answers.

Include the Honor Code pledge or just the word "pledged", *plus* at least one of the following about collaboration and help (as many as apply). Text *in italics* is explanatory or something for you to fill in.

- I did not get outside help *aside from course materials, including starter code, readings, sample programs, the instructor.*
- I worked with *names of other students* on this assignment.
- I got help with this assignment from *source of help — ACM tutoring, another student in the course, etc. (Here, "help" means significant help, beyond a little assistance with tools or compiler errors.)*
- I got help from *outside source — a book other than the textbook (give title and author), a Web site (give its URL), etc.. (Here too, you only need to mention significant help — you don't need to tell me that you looked up an error message on the Web, but if you found an algorithm or a code sketch, tell me about that.)*
- I provided help to *names of students* on this assignment. *(And here too, you only need to tell me about significant help.)*

# 4 Essay

For programming assignments, this section should go in the body of the e-mail or in a plain-text file `pledge.txt` (no word-processor files please). For written assignments, please put it in the text or PDF file with your answers.

Include a brief essay (a sentence or two is fine, though you can write as much as you like) telling me what if anything you think you learned from the assignment, and what if anything you found interesting, difficult, or otherwise noteworthy.