Homework 2
Caches and Digital Logic

CSCI-UA.0201-005 Fall 2023

Due Monday, December 18 by 11:55pm

<u>No late assignments will be accepted</u>, since the solutions will be posted soon after the deadline.

The questions below involve writing answers (using Word or whatever) in an answer file as well as generating circuit files using Logisim Evolution (below). When you are finished, upload the answer file (in PDF form) as well as the Logisim circuit files that you've created.

Caches

1. Explain why set-associative caches might generally result in better performance than direct-mapped caches.

2. By your reasoning for question 1, it would appear that fully associative caches might have better performance than set-associative caches. So, why don't commercial processors generally use fully associative caches?

3. In your own words, describe what "temporal locality" means and give an example in a tiny C program.

4. Describe what aspect of a cache exploits temporal locality.

5. In your own words, describe what "spatial locality" means and give an example in a tiny C program.

6. Describe what aspect of a cache exploits spatial locality.

7. On a 64-bit processor (where addresses and integers are 64 bits, but only the lower 48 bits of an address are used) with a 4MB direct-mapped cache, where each cache entry is one 64-bit word (i.e. only one word is brought into cache at a time), how many tag bits must be associated with each word in the cache? Note that a "4 MB cache" means that it is able to store 4 MB of actual data, not counting tag bits, etc.

8. On a 64-bit processor with a 4MB 4-way set associative cache, where each cache entry is one word, how many tag bits must be associated with each word in the cache?

9. On a 64-bit processor with a 4MB 8-way set associative cache, where each cache entry (i.e. cache line) contains 4 words, show how a 64-bit address (where the upper 16 bits are ignored) is divided into fields (tag, byte offset, etc.). Be sure to specify how many bits are in each field.

Digital Logic (using Logisim Evolution)

To do this homework, you will need to install Logisim Evolution, the digital logic simulator that I used in class. It should only take you a couple of minutes to install, it's easy to use, and frankly, it's fun.

Instructions for installing and running Logisim Evolution are found under the Resources tab on Brightspace.

Once you've started Logisim, it's very easy to create a circuit. To learn how, just click on "Help" and then "Tutorial" to run the tutorial. It takes only a few minutes to get through the tutorial. As practice, be sure to build the XOR device described in the tutorial.

10. Build a 4-bit adder from AND, OR, and NOT gates as follows.

    a. Write out the truth table for a 1-bit adder, which has three 1-bit inputs: a, b, and carry_in, and two 1-bit outputs: result and carry_out.

    b. Implement, in Logisim, the circuit for the 1-bit adder corresponding to the above truth table.

    c. In Logisim, create the four-bit adder by making three more copies of the above 1-bit adder, and then connecting them together. The resulting four-bit adder should have the following inputs: a0, a1, a2, a3, b0, b1, b2, b3 and carry_in. It should have the following outputs: result0, result1, result2, result3, and carry_out. (Note: To copy and paste the 1-bit adder, select the entire 1-bit adder by dragging the mouse over the adder, copy by typing either control-C or command-C, and then paste by typing control-V or command-V. You'll need to drag each new copy into the right position).

    d. Be sure to test the adder to make sure it is getting the right outputs. Once you have it working, you should:

        i. Give the circuit a name by clicking on "main" under the "untitled" folder in the upper left-hand window and then, in the bottom left where it says "main" next to circuit name, typing in the name you want to give the circuit. In this case "one_bit_adder" is a good name, but it's up to you.

        ii. Save the circuit (by clicking "File" and "Save") to a file named "adder.circ".

11. In Logisim, create a 2-input, 1-select line multiplexer (mux) from AND, OR, and NOT gates. Give the circuit a name and save the file as "2_input_mux.circ". (Note: To create a new circuit, click "File" and "New").

12. In Logisim, create a 4-input, 2-select line (mux) from AND, OR, and NOT gates. Give the circuit a name and save the file as "4_input_mux.circ".

13. In Logisim, using your adder, the mux's you built (as many copies of these as you need), and AND, OR, and NOT gates, create a 4-bit ALU that performs AND, OR, + and -. It should have the following inputs: a0, a1, a2, a3, b0, b1, b2, b3, control0, and control1. It should have following outputs: res0, res1, res2, res3, and carry_out. The two control lines should determine the operation to be performed as follows:

control1	control0	
0	0	AND
0	1	OR

1	0	+
1	1	-

Note that this ALU will be slightly different from the ALU in the class notes, since there is no separate B_{invert} line. Note: To incorporate previously-built circuits in a circuit, click “Project > Load Library > Logisim-evolution Library…” and then choose the file you had saved. On the left hand side, a new folder will appear with the name of the file. Double-click on the folder and you should see a circuit with the name you gave it. It will be represented as a rectangle with inputs (small dots) on the left and outputs (also small dots) on the right. Select and use the circuit just as you would any other device (e.g. AND gate). Be sure to add text to label the devices.

14. In Logisim, build a falling-edge triggered D flip-flop out of AND, OR, and NOT gates. To do this, build a simple unclocked-latch, then use the unclocked latch to build a clocked D latch, then use the D latch to build flip-flop. Don’t use any of the devices that Logisim may provide other than AND, OR, and NOT gates (that is, it provides you with a whole bunch of different devices, including flip-flops, but don’t use them).

15. In Logisim, build a 4-bit register from your flip-flops of the previous step. The inputs to your register should be: data0, data1, data2, data3, clock, and write_enable. The values on data0 through data3 should be stored in the register (upon the falling edge of the clock, of course), but only if write_enable is asserted. The outputs should be output0, ouput1, output2, and output3.

16. Finally, in Logisim, build a circuit containing:

    - Three of your 4-bit registers.

    - Your ALU. The outputs of two of the above registers should be wired to the inputs to the ALU and the outputs of the ALU should be wired to the inputs of the third register.

    - A clock. If you look in the “Wiring” folder in the left hand side of the Logisim screen, you’ll see a clock device to click on. Your circuit should have a single clock input that is wired to all three registers. You can change the value of the clock (from 0 to 1 and 1 to 0) by using the poke tool () and clicking on the clock. You can also make the clock cycle in a regular pattern by clicking on the “Simulate” menu, choosing a “Auto-Tick Frequency”, and clicking “Auto-Tick Enabled”.

    - The inputs to the first two registers should be from pins, using the usual input tool ( ). The control inputs to the CPU should also be from pins.

    This way, you are able to write specific 4-bit values to the first two registers and compute the value of the third register by choosing an operation (AND, OR, +, -) on the ALU.

Digital Logic (not using Logisim - write the answers in your answer file)

17. Consider the drawing of the register file in my lecture notes.

    a. Describe precisely what the function of the decoder is in the operation of the register file. What is the function of the two multiplexers?

    b. Give an example of an Intel x86-64 instruction that would require the write-enable line to be asserted (set to 1). The CPU programming project description lists the MIPS instructions, of course.

    c. Give an example of an Intel x86-64 instruction that would require the write-enable line to be de-asserted (set to 0).

18. Consider the drawing of the "DRAM cell" in my lecture notes.

    a. Why is this memory called "Dynamic RAM"?

    b. When does a DRAM cell have to be "refreshed" and why? How is a DRAM cell refreshed?

    c. If the "word line" is de-asserted, can the DRAM cell be read or written? Explain.

19. Consider the drawing of the portion of the datapath that fetches instructions in the simple processor in my lecture notes.

    a. Why is 8 being added to PC?

    b. Why don't we have to worry about PC+8 being written to the PC register while the instruction address is still being sent to the instruction memory?

20. Consider the drawing of the portion of the datapath that performs simple operations on registers (only) in the lecture notes. Suppose the instruction

    ```
    addq %rax,%rbx
    ```

    is executed. Assuming %rax is register 0 and contains the number 10, and %rbx is register 1 and contains the number 15, for every wire in the drawing (instruction, Read1, Read2, write reg, etc.), indicate what value that wire carries (e.g. "Read1 carries the number 12").