

Homework 2
Caches and Digital Logic
CSCI-UA.0201 Spring 2022

ANSWERS

Caches

1. Define the terms “compulsory miss”, “capacity miss”, and “conflict miss”.

Answer: A compulsory miss occurs the first time a cache line in memory is accessed, so that cache line has not previously been brought into the cache. A capacity miss occurs because the cache isn’t big enough to hold the instructions and data that the program currently needs (e.g. the instructions and data within a loop), so there is always a currently-used instruction or datum that can’t fit in the cache and causes a cache miss when accessed. A conflict miss occurs when two cache lines that are currently in use are mapped to the same entry in the cache (e.g. in a direct-mapped cache), so both cache lines can’t be in the cache simultaneously. Accessing one of the cache lines causes the other to be evicted, resulting in a cache miss when the evicted cache line is subsequently referenced.

2. Which of the above types of cache misses are reduced by having a set-associative cache instead of a direct-mapped cache?

Answer: Conflict misses are reduced by using set-associative caches, since two cache lines that map to the same set can be put into two different entries in the set, allowing them both to be in the cache at the same time.

3. Consider the following statement: “For n-way set-associative caches, as n gets larger (assuming the total number of data bytes stored in the cache remains constant), the number of capacity misses will increase because there are fewer sets in the cache.”

- a. Is the statement true or false?

Answer: The statement is false.

- b. Explain your answer.

Answer: Since the problem assumes that the total capacity of the cache remains constant as n increases, and capacity misses are due only to the size (capacity) of the cache, increasing n has no effect on capacity misses. Increasing n only reduces conflict misses.

- c. If your answer was “true”, why don’t CPU manufacturers just use n=1? If your answer was “false”, why don’t CPU manufacturers set n to the largest possible value (i.e. set n to the number of cache lines that can be stored in the cache).

Answer: Since increasing n has no effect on capacity misses, but does reduce conflict misses, larger values of n will reduce the cache miss rate. However, large values of n (such as setting n to the number of cache lines in the cache, i.e. a fully-associative cache) are too complicated to implement in hardware, requiring too many gates and transistors.

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

Answer: Temporal locality is a property of programs that the same instructions and data will repeatedly be accessed within a small period of time. In the following C code, the instructions and variables (i, j, and k) within the loop will be accessed repeatedly while the loop is executing.

```
int main()
{
    int i, j = 5, k = 10;
    for(i=1; i < 1000; i++) {
        j = j + i;
        k = k + j;
    }
    printf("k = %d\n", k);
}
```

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

Answer: There are two aspects of a cache that exploit temporal locality: (1) Data and instructions are brought into the cache the first time they are accessed, and temporal locality says that they will be accessed again soon, resulting in a cache hit the next time they are accessed, and (2) the NRU cache replacement policy says to keep the recently accessed instructions and data in the cache (evicting those that are not recently used), since temporal locality says that the recently-accessed instructions and data are going to be accessed again soon, resulting in a cache hit.

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

Answer: Spatial locality says that if an instruction or datum is accessed, then neighboring instructions or data will be accessed soon. In the following C program, the array elements and the instructions in the loop are accessed in consecutive order. Thus, the second element of the array is accessed soon after the first element of the array, and the second instruction in the loop is accessed soon after the first instruction in the loop, etc.

```
int main()
{
    int i, a[10];
    a[0] = 1;
    for(i=1; i < 10; i++) {
        a[i] = a[i-1] * 2;
        a[i] = a[i] + 17;
    }
    printf("a[9] = %d\n", a[9]);
}
```

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

Answer: Having a multi-word cache line exploits spatial locality, since accessing an instruction or datum will cause the neighboring instructions and data in the same cache line to be brought into the cache.

8. Suppose a machine has two caches, an L1 cache and an L2 cache. Assume the following characteristics of cache performance:

- The processor is able to execute an instruction every cycle, assuming there is no delay fetching the instruction or fetching the data used by the instruction.
- An L1 cache hit provides the requested instruction or data immediately to the processor (i.e. there is no delay).
- An L2 cache hit incurs a penalty (delay) of 10 cycles.
- Fetching the requested instruction or data from memory – which happens when there is an L2 miss – incurs a penalty of 100 cycles. That is, the total delay is 100 cycles on an L2 miss, don't add the 10-cycle delay above.
- The L1 miss rate for both instructions and data is 3%. That is, 97% of instruction and data fetches result in an L1 cache hit.
- The L2 miss rate for instructions is 1% and for data is 2%. That is, of those instruction fetches that result in an L1 cache miss, 99% of them will result in an L2 cache hit. Similarly, of those data fetches that result in an L1 cache miss, 98% of them will result in an L2 cache hit.
- 30% of instructions require data to be fetched (the other 70% don't).

a. Given a program that executes I instructions, how many cycles would the program take to execute, under the above assumptions? Be sure to show your work.

Answer: To solve this problem, we translate the following into a mathematical formula, as was done in class for the simpler case where there is only a single cache:

- The program executes I instructions such that, assuming no cache miss penalties, the processor can execute an instruction every cycle. Thus, not counting any cache-miss penalties, the program takes I cycles.
- For instructions:
 - 3% of the time there is a L1 cache miss. Of that 3% of instructions that miss the L1 cache, 99% (of the 3%) result in an L2 cache hit, for which the penalty is 10 cycles. Thus, the total penalty for instruction fetches that miss the L1 cache and hit the L2 cache is $(I * 0.03 * .99 * 10)$.
 - Of the 1% of the 3% of instructions that result in both an L1 and L2 cache miss, there is a 100 cycle penalty. Thus, the total penalty for instruction fetches that miss both the L1 and L2 cache is $(I * .03 * .01 * 100)$.
- For data, the computation is similar, but with different hit/miss rates at the L2 cache. Also, only 30% of the I instructions actually need data from memory. So, the total penalty for the 98% of the 3% of data fetches that miss the L1 cache and hit the L2 cache is $(.3 * I * .03 * .98 * 10)$, and the total penalty for the 2% of the 3% of data fetches that miss both the L1 and the L2 caches is $(.3 * I * .03 * .02 * 100)$.

- Thus, the total number of cycles to execute the program is:

$$I + (I * 0.03 * .99 * 10) + (I * .03 * .01 * 100) + (.3 * I * .03 * .98 * 10) + (.3 * I * .03 * .02 * 100)$$

$$= I + (I * 0.297) + (I * 0.03) + (I * .0882) + (I * 0.018)$$

$$= \underline{1.4332 I}$$
- b. Suppose you are manufacturer of the above machine and, in the next release, you have the option of increasing the size of the L1 cache so that the L1 miss rate is reduced by 50% for both data and instructions or increasing the size of the L2 cache so that the L2 miss rate is reduced by 50% for both data and instructions. If your only goal was to make the above program faster, which option would you choose? Show your work.

- **Answer:** If we reduce the L1 miss rate by 50%, then the L1 miss rate goes from 3% to 1.5%. Therefore, the .03 in the above formula gets replaced by .015 (reducing all the penalty terms by half) and the total number of cycles is:

$$I + (I * 0.015 * .99 * 10) + (I * .015 * .01 * 100) + (.3 * I * .015 * .98 * 10) + (.3 * I * .015 * .02 * 100)$$

$$= I + (I * 0.1485) + (I * 0.015) + (I * .0441) + (I * 0.009)$$

$$= \underline{1.2166 I}$$

If we reduce the L2 miss rate by 50%, then the L2 miss rate for instructions goes from 1% to .5% and for data goes from 2% to 1%. Therefore, the .99 in the original formula becomes .995, the .01 becomes .005, the .98 becomes .99, and the .02 becomes .01. The total cycles in this case is:

$$I + (I * 0.03 * .995 * 10) + (I * .03 * .005 * 100) + (.3 * I * .03 * .99 * 10) + (.3 * I * .03 * .01 * 100)$$

$$= I + (I * 0.2985) + (I * 0.015) + (I * 0.0891) + (I * 0.009)$$

$$= \underline{1.4116 I} \text{ (not a big improvement!)}$$

Since increasing the size of L1 reduces the total cycle time more than increasing the size of L2 (under the above assumptions), that's the option to choose. **Note: You will not have to do this complicated arithmetic on the exam. I will choose nice round numbers.**

9. Assuming a word size of 32 bits, so that both addresses and integers are represented as 32 bits, how would the bits of an address be used to support caching on a machine with a 16MB 8-way set-associative cache and a 16-word cache line? That is, specify which bits of the address would be used for the byte offset, the word offset, etc. The 16MB refers to the total amount of data that can be stored in the cache, and does not include the storage required for the tags, valid bit, etc.

Answer: Since the capacity of the cache is $16\text{MB} = 2^{24}$ bytes, and each cache line is 16 words = 64 bytes = 2^6 bytes, it means there are $2^{24}/2^6 = 2^{18}$ cache lines in the cache. Since there are $8 = 2^3$ cache lines per set, there are $2^{18}/2^3 = 2^{15}$ sets in the cache. Therefore,

- Since there are $4 = 2^2$ bytes per word, the byte offset in the address is the rightmost 2 bits, namely bits 0-1.
- Since there are $16 = 2^4$ words per cache line, the word offset is the next 4 bits in the address, namely bits 2-5.

- Since there are 2^{15} sets in the cache (above), the set index is the next 15 bits in the address, namely bits 6-20.
- The rest of the bits in the address comprise the tag, namely the remaining 11 bits, bits 21-31.

Digital Logic (using Logisim Evolution)

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.

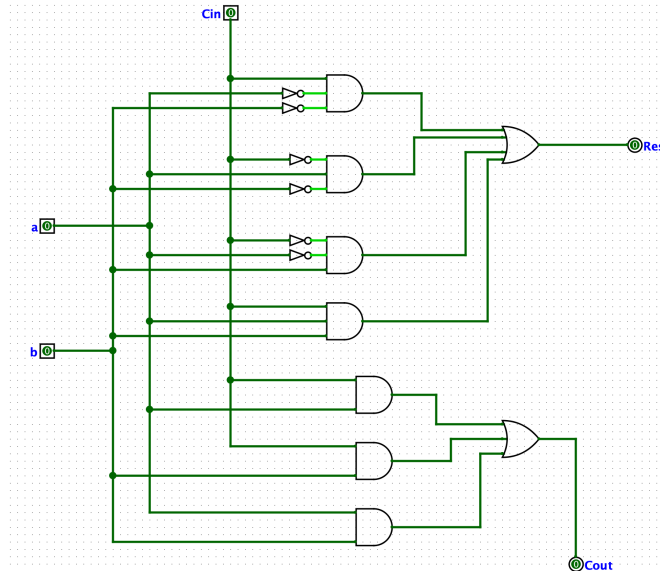
Answer:

a	b	Cin		R	Cout
0	0	0		0	0
0	0	1		1	0
0	1	0		1	0
0	1	1		0	1
1	0	0		1	0
1	0	1		0	1
1	1	0		0	1
1	1	1		1	1

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

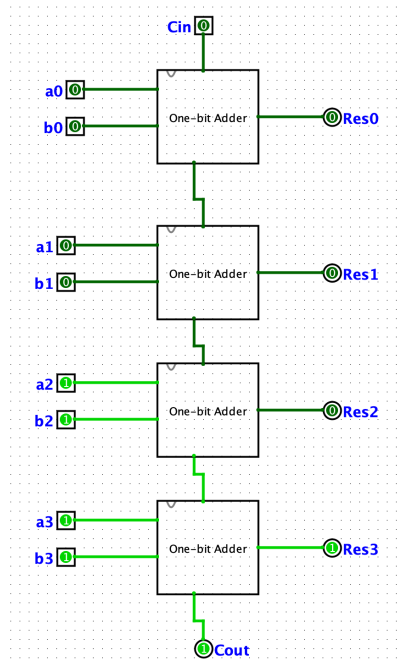
Answer: See the circuit, below. As in class, I made a small improvement in the computation of the carry out, since carry out is 1 if at least two of the inputs are 1. You did not have to make such improvements.

Note: The input and output pins look a little different in the drawings below than they do in your circuits. This is because my circuits were created on an older version of Logisim.



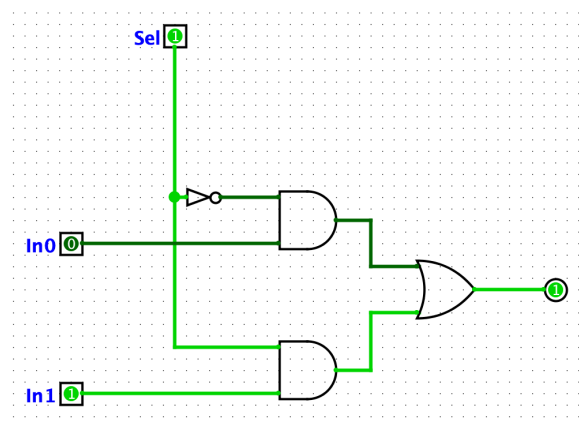
- c. In Logisim, create the four-bit adder by making three more copies of the above 1-bit adder, and then connecting them together. The resulting four-bit adder should have the following inputs: a0, a1, a2, a3, b0, b1, b2, b3 and carry_in. It should have the following outputs: result0, result1, result2, 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).

Answer: For clarity, I show each one-bit adder (above) as a box, rather than showing the gates. You could have done it either way.



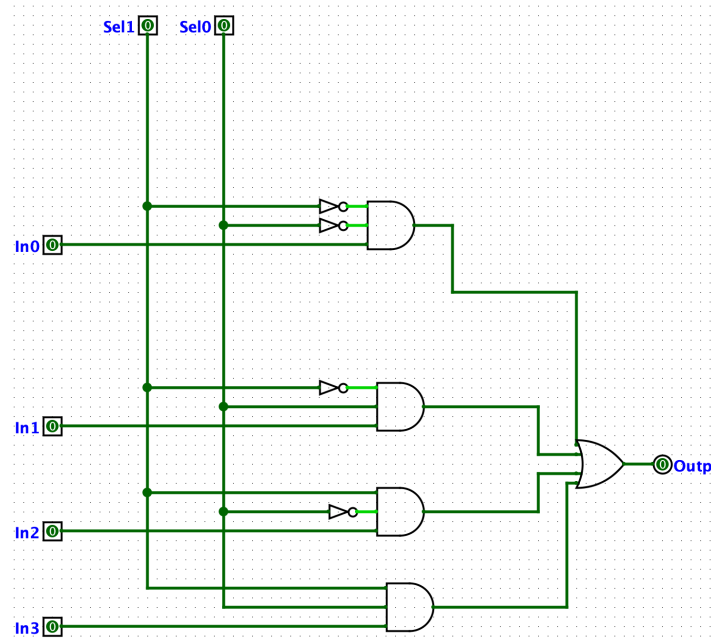
11. In Logisim, create a 2-input, 1-select line multiplexor (mux) from AND, OR, and NOT gates. Give the circuit a name and save the file as “2_input_mux.circ”. (Note: To create a new circuit, click “File” and “New”).

Answer: You could certainly have built a truth table and converted it to a circuit. I recognized that when the select line is 0, the value depends only on In0, so the value of In1 is irrelevant in that case. Conversely, when the select line is 1, only the value of In1 matters.



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”.

Answer: Creating a 6-input truth table isn’t really practical here. The circuit below was arrived at by realizing that when Sel1 and Sel0 are both 0, then the result depends only on In0, when Sel1 is 0 and Sel0 is 1, then the result depends only on In1, and so on.

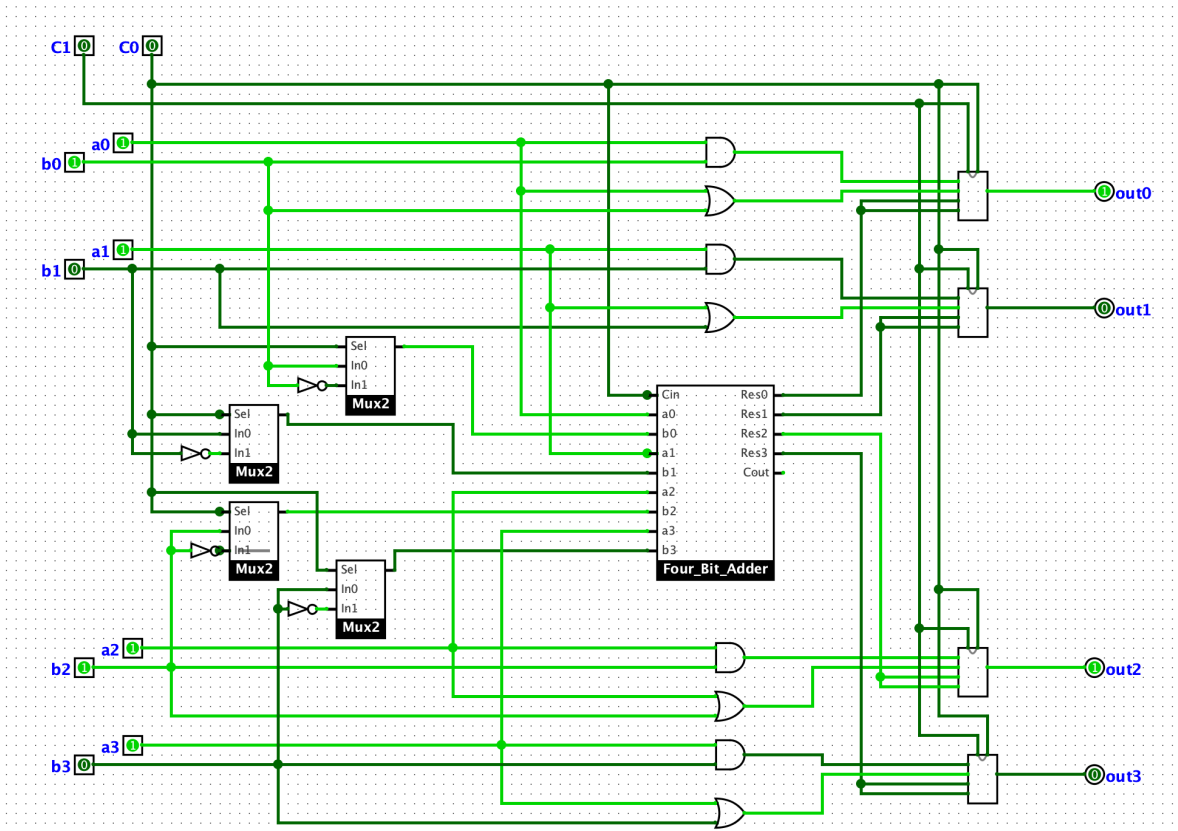


13. In Logisim, using your adder, the mux's you built (as many copies of these as you need), and AND, OR, and NOT gates, create a 4-bit ALU that performs AND, OR, + and -. It should have the following inputs: a0, a1, a2, a3, b0, b1, b2, b3, control0, and control1. It should have following outputs: res0,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

<u>control1</u>	<u>control0</u>	
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.

Answer: See below. I noticed that when control0 is zero, both b_{invert} and c_{in} should be zero (for addition), and when control0 is 1, both b_{invert} and c_{in} should be 1 (for subtraction). Therefore, I just connected control0 to b_{invert} and to c_{in} .

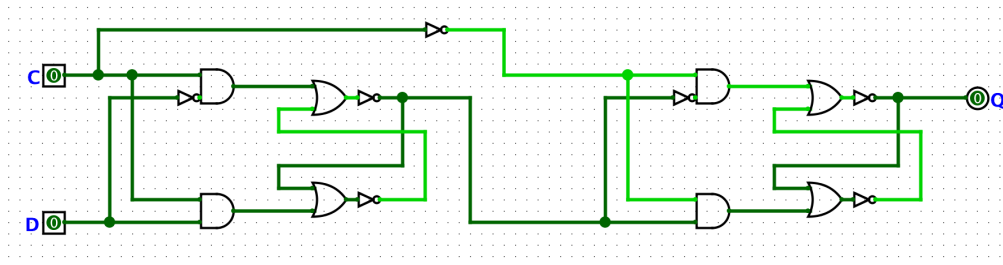


(the unlabeled boxes on the far right are 4-input multiplexers)

Note: You could also have taken the approach discussed in the lecture, where a 1-bit ALU was built using a 1-bit adder, and then four 1-bit ALUs were chained together to form a 4-bit ALU.

14. In Logisim, build a falling-edge triggered D flip-flop out of AND, OR, and NOT gates. To do this, build a simple unlocked-latch, then use the unlocked 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).

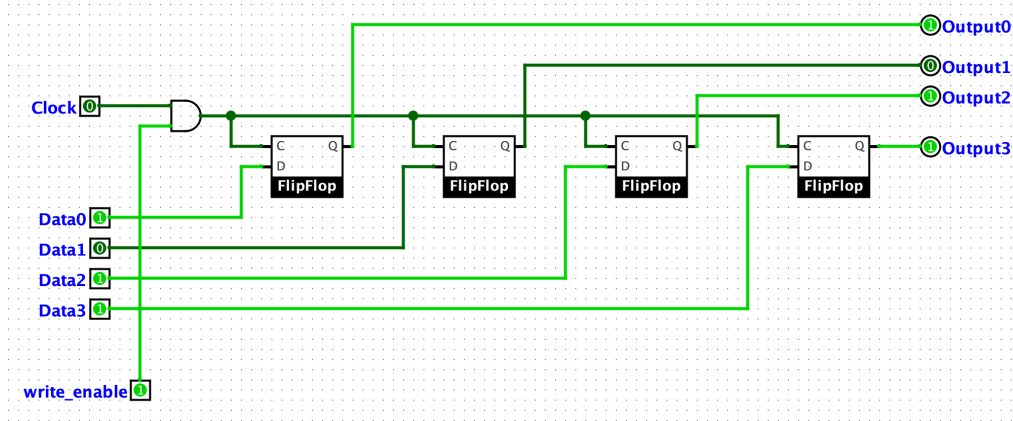
Answer:





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, output1, output2, and output3.

Answer:

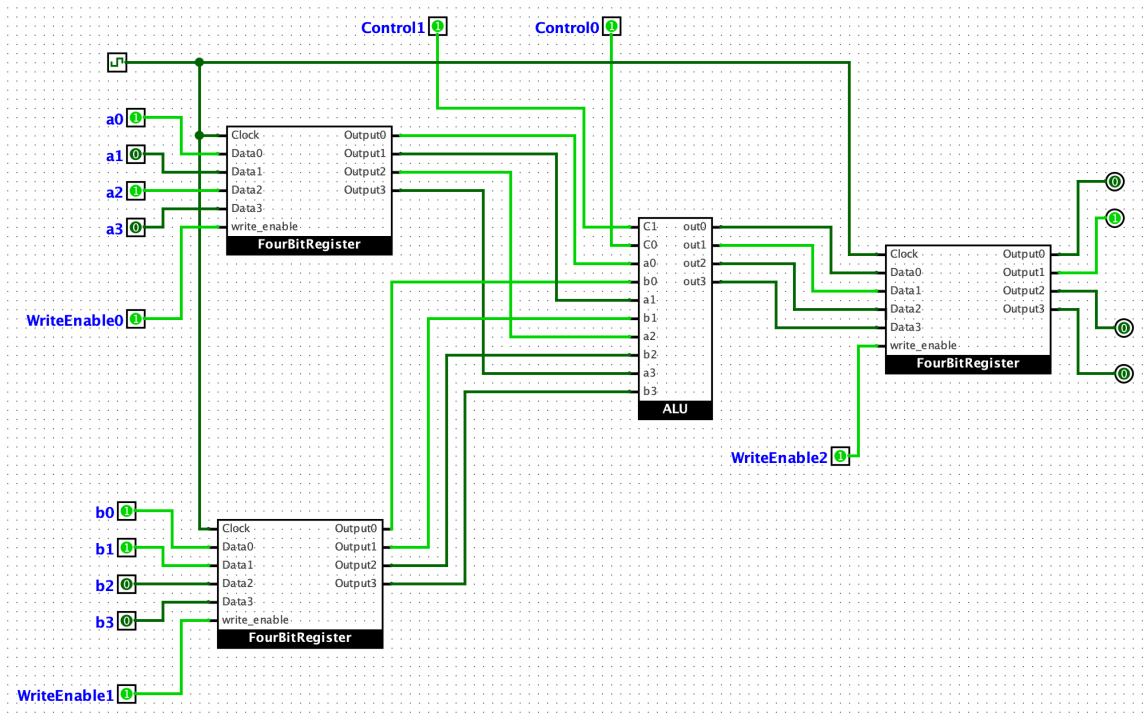


16. Finally, in Logisim, build a circuit containing:

- 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.

Answer:



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.

Answer: The decoder is used to select the register to write to, if the current instruction writes to a register. For example, in the x64 processor, the input to the decoder would be the index of the %rax register when the instruction “addq %rbx, %rax” is executed. In this example, each of the output wires of the decoder carries a zero, disabling the writing to the register that wire is connected to, except for the wire corresponding to the index of the %rax register, enabling %rax to be written to.

- b. What is the function of the two multiplexers?

Answer: Each of the two multiplexers is used to send the output of a single register to a “read data” output of the register file. For example, when the x64 processor executes the instruction “addq %rbx, %rax”, the select line of the first multiplexer carries the index of the %rbx register and the select line of the second multiplexer carries the index of the %rax register. Thus, the “read data0” output of the register file would carry the value of %rbx and the “read data1” output would carry value of %rax.

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

- a. Why is this memory called “Dynamic RAM”?

Answer: It’s called “dynamic” because it has to be refreshed – its value read and written back – on a regular basis. This is in contrast to “static RAM”, made from flip flops, which holds its value without having to be refreshed.

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

Answer: Because the capacitor in each DRAM cell “leaks”, i.e. loses its charge over time, all DRAM cells in memory have to be periodically refreshed. Because reading the value of a cell, as part of fetching an instruction or datum from memory, changes the charge on the capacitor in the cell, the cell needs to be refreshed after it is read.

A refresh of a DRAM cell is performed by reading the value of the cell and then writing that value back to the cell.

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

Answer: No. If the word line is de-asserted, then the N-type CMOS transistor will not let current flow between the capacitor and the bit line. Therefore, if the word line is de-asserted, no matter what voltage is placed on the bit line (high or low for writing, or mid-level voltage for reading), the capacitor will not get charged or discharged due to the voltage level of the bit line.

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 the program counter register (e.g. %rip)?

Answer: The lecture notes assume that each instruction is 8 bytes (although this is not true for the x64). Therefore, consecutive instructions are 8 bytes apart. After fetching an instruction from memory at the address specified in the PC, the PC is incremented by 8 to point to the next instruction in memory.

- b. Why don’t we have to worry about 8 being added to the program counter register while the instruction address is still being sent to the instruction memory?

Because, as with all registers, the PC has a clock input and the new value (i.e. $PC+8$) won’t be stored in the PC until the falling edge of the clock. As long as the clock voltage is high, the value of PC won’t change.

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

- a. Why is the second operand sent to both the second read-select line and the write-select line?

Answer: Because the second operand is both one of the registers to read from, as the input to the computation, and the register to write the result to. For example, for the instruction

addq %rax,%rcx

the addition takes as inputs the values of %rax and %rcx, so the value of %rcx has to be read, and writes the result back to %rcx.

- b. Since the register indicated by the second read-select line and the register indicated by the write-select line are the same, why doesn't the writing of data to that register interfere with the reading of that register while the operation is being performed? Explain.

Answer: The register file has the clock as an input and the value of the register being written to will not change until the falling edge of the clock. Therefore, as long as the clock voltage is high, the register being read will hold its value.

21. Consider the drawing of the portion of the datapath that implements "move" operations involving memory.

- a. Could this datapath be used to move data from one location in memory to another location in memory in a single cycle? Explain your answer.

Answer: No, the datapath in the lecture notes cannot be used to move data from one memory location to another memory location in a single cycle of the clock. That's because the only place that data being read from memory is sent to is the register file, and the register file won't be updated until the clock falls in the second half of the first cycle. So, after one cycle, the data has been moved from memory to the register file. Then, since the only data that can be written to memory comes from the register file and the memory won't be updated until the falling edge of the clock, the data that was written to the register file in the first cycle won't be written back to memory until the falling edge of the second clock cycle. Therefore, it takes two cycles to move data from one location in memory to another.

- b. What would be different, in terms of the values carried by the various lines (wires) shown in the diagram, between "movq %rax,8(%rcx)" and "movq 8(%rcx),%rax".

Answer: For "movq %rax,8(%rcx)", which moves data from a register to memory, no data is written to a register but data is written to memory. Therefore the write enable input to the register file will be 0 and the read/write select input to memory will indicate "write". This will cause the register file to ignore its write-data input.

For "movq 8(%rcx),%rax", which moves data from memory to a register, the write-enable input to the register file will be 1 and the read/write select input to memory will indicate "read". This will cause the memory to ignore its write data input.

22. Consider the drawing of the portion of the datapath that implements conditional branch instructions.

- a. What is the role of the multiplexer in the top right portion of the drawing?

Answer: To select between $PC+8$ and $PC+offset$ to write back to the PC register, depending on whether the condition of the conditional jump is false or true (respectively).

- b. Under what circumstances would that multiplexer send its first input to its output and under what circumstances would it send its second input to its output?

Answer: The first input to the multiplexer, $PC+8$, is selected if the jump condition is false and therefore the jump is not performed. In this case, the PC would be incremented by 8 to point to the next instruction in memory. The second input, $PC+offset$, is selected if the jump condition is true, in which case the jump is performed by writing the address of the target of the jump, namely $PC+offset$, to the PC.

- c. Why does the output of the multiplexer get sent to the program counter register rather than to the register file shown in the drawing?

Answer: The PC is a special-purpose register that is not part of the general-purpose register file. A jump is implemented by setting the PC to the address of the target of the jump. Writing that address into a register in the register file would not cause a jump.