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

8. Consider the drawing of the register file on page 14 of my handwritten lecture notes on digital logic (that’s the first page of Part 3 of the notes).

a. Notice that I left out an explicit clock input to the register file. Of course, a clock input is needed. Describe precisely how the clock input would be added to the drawing (i.e. what would it be connected to, if any additional gates would be required, etc).

Simply AND the clock with the Write Enable Line and send the result to where, in the picture, the Write Enable Line is currently going. That way, when the clock is 0, no register can be written to. Alternatively, you could send the clock line into the AND gate that is to the left of each register.

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

The decoder is used to select the register that will be written to (if the write enable line is asserted). That is, the input to the decoder – namely the write select – is the index of the register to write to (e.g. the RD value in an R-instruction), so the only output of the decoder that is asserted is the one for the specified index. The function of the two multiplexors is to select the values coming from the two registers specified by the two read select lines.

c. Give an example of a MIPS instruction that would require the write-enable line to be asserted.

add $7, $6, $5

which writes the sum of register 6 and register 6 into register 7. In this case, since the result of the addition needs to be written to a register, the write enable line would have to be asserted.

d. Give an example of a MIPS instruction that would require the write-enable line to be de-asserted.

beq $7, $6, L1
which branches to the label L1 (which is an address) if register 7 is equal to register 6. Since no register is being written to, the write enable line would be de-asserted.

9. Consider the drawing of the “DRAM cell” on page 18 of my lecture notes (2nd page of Part 4).
   
   a. Why is this memory called “Dynamic RAM”?

   Because each DRAM cell has been periodically written to, or else it will lose its value.

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

   It needs to be refreshed (re-written with the value that it holds) whenever it has been read as well as periodically even if it has not been read. This is because reading the DRAM cell will change the voltage level in the capacitor, potentially wiping out the value that it holds. Furthermore, the capacitor slowly loses its charge over time, so the value it holds has to be refreshed before it is lost. The DRAM cell is refreshed simply by reading its value (0 or 1) and writing the same value (0 or 1, but at the right voltage levels) back to the cell.

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

   If the word line is de-asserted, the DRAM cell cannot be read or written. That’s because the N-type transistor in the cell will not allow current to flow between the bit line and the capacitor when the word line is de-asserted.

10. Consider the upper drawing on page 47 of the notes (7th page of Part 4).

    a. Why is 4 being added to PC? In what type of CPU would 8 need to be added to the PC?

    Because, on the MIPS processor being described by these circuits, each instruction is exactly 4 bytes. Thus, fetching the next instruction requires incrementing the PC by 4 each time (unless there’s a jump). On a CPU where each instruction was 8 bytes, the PC would need to be incremented by 8 each time.

    b. Why don’t we have to worry about PC+4 being written to the PC register while the instruction address is still being sent to the instruction memory?

    Because the PC – being a register constructed of flip-flops – can only change upon the falling edge of the clock. By the time the clock falls (becomes 0), the CPU had better have finished whatever it needed to do with the old value of the PC.

11. Consider the lower drawing on page 47 of the notes (7th page of Part 4). Suppose the instruction

    add $10, $12, $7
is executed. 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”).

Instruction carries the 32-bit machine code instruction
Read1 carries the number 12 (which is the value of RS)
Read2 carries the number 7 (which is the value of RT)
WriteReg carries the number 10 (which the value of RD)
DataRead1 carries the value of register 12.
DataRead2 carries the value of register 7
Operation carries the control value for selecting the add operation in the ALU
Zero indicates whether or not the result of the addition is zero (not discussed in class)
WriteData carries the result of the addition
WriteEnable is 1, to allow the writing of the result to register 10.

12. Consider the drawing on page 48 of the notes (8th page of Part 4).

a. Why is the offset (i.e. the immediate value) only 16 bits? Instead of sign-extending the offset to 32 bits, why not make the offset 32 bits to begin with?

Since the entire instruction – including the opcode, rs, rt, and offset fields – is 32 bits, the offset couldn’t be 32 bits all by itself.

b. If the “mem write” line going in to the memory is asserted, then the memory will store the value coming in to memory on the “write data” line. Which of the two instructions shown above the drawing would cause the “mem write” line to be asserted?

The store (SW) instruction would cause “mem write” to be asserted, since it needs to store the value of a register into memory. The load instruction (LW) only reads a value from memory, so the “mem write” line would be de-asserted.

c. Why is the output of the ALU being sent to memory?

The ALU is computing the memory address to be used by adding the value of the register indicated by the RS field to the offset. This address needs to be sent to the memory in order to read from or write to the memory (for the LW or SW instructions, respectively) at that address.

13. Consider the drawing on page 49 of the notes (the 9th page of Part 4).

a. What are the values of the two inputs (not the select line) to the multiplexor in the top right of the drawing? Under what circumstances would the upper input be selected and under what circumstances would be lower input be selected?

The upper input carries the value of PC+4 and the lower input carries the value of PC+4+(sign_extend(imm)<<2). The upper input would be selected in the case that a branch is not taken – in which case the next instruction in memory would be executed. The lower input would be selected in the case that the branch is taken (in the particular instruction shown in the notes, when “$reg1” equals “$reg2”).
b. As a hardware designer, suppose you wanted the CPU to support a “bge” instruction, which is just like “beq” except that the branch is taken if the first operand is greater than or equal to the second operand. Describe how the “control logic” would determine which input to the multiplexor is selected (just give a description, you don’t need to draw the logic).

If the first operand is greater than or equal to the second operand, the result of the subtraction will be non-negative (i.e. positive or zero). In that case, the most significant bit of the result would be 0. Thus, the control logic simply needs to use the most significant bit (i.e. the most significant wire of the result output from the ALU) to determine which input to the multiplexor to select.

c. Why are there no wires shown coming into the “write reg” or “write data” inputs to the register file?

Since the branch operation doesn’t write to a register, I didn’t bother to show the write reg or write data inputs (in the actual hardware, they’re there, of course).

Floating Point

14. Read the two pages my handwritten notes on Floating Point that I put on the course web page and take a look at the C code I wrote for printing a floating point number. Describe precisely how the hardware would add two floating point numbers together. That is, describe: which bits constitute which fields of each number; what shifting may need to occur; what fields are added together; etc.

The exponents, bits 23-30, are extracted from each operand and 127 (the bias) is subtracted from each exponent. The significands, bits 0-22, are extracted from each operand and a 1 bit is appended to the left of each significand (at bit position 23). Then, the significand from the operand with the smaller exponent is shifted right by the difference between the exponents. Therefore, for the moment, the exponent of the result will be the same as the larger of the two operand exponents.

The two significands are then added together (taking account of the sign bits, bit 31, of both operands – assume sign+magnitude arithmetic). If the result of the addition has a 1 in bit position 24 (meaning there was a carry), shift the result to the right by 1 and add 1 to the exponent. Finally, write the sign of the result, the exponent of the result, and the significand of the result (only bits 0-22, dropping the leading 1) into the resulting 32-bit floating point number.

Caches

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

If a program is using, say, two variables whose addresses map to the same location in a direct-mapped cache, then they both can’t be in the direct-mapped cache at the same time. If they’re both being used within the same loop, then
each variable will repeatedly kick the other one out of the cache and you will repeatedly get cache misses and poor performance. A set-associative cache, in this case, will be able to store both variables in the same set so that you will have far fewer cache misses.

16. On a 32-bit processor (i.e. addresses are 32 bits) with a 4MB direct-mapped cache, where each cache entry is one 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?

Since the cache is 4M bytes and there are 4 bytes per entry in the cache, there are \(1M = 2^{20}\) entries. Therefore 20 bits of an address is used to select an entry in the cache. Additionally, since a cache entry holds 4 bytes of data, the rightmost 2 bits of the address are used to select the byte in the data. Since addresses are 32 bits, the remaining \(32 - (20+2) = 10\) bits are used as the tag bits.

17. On a 32-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?

(THIS WILL NOT BE ASKED ON OUR FINAL EXAM)
Since each set holds 4 words of data, which is 16 bytes of data, the number of sets in the cache is \(4M/16 = 2^{22}/2^4 = 2^{18}\). Thus, 18 bits of the address are needed to access the set and, once a word is retrieved from the set, 2 bits of the address are needed to select the byte from the word. Therefore, the remaining bits, namely \(32 - (18+2) = 12\) bits are used as the tag bits.