Study Questions for Digital Logic, Processors and Caches

G22.0201 Fall 2009

<u>ANSWERS</u>

Digital Logic

Read: Sections 3.1 – 3.3 in the textbook .
Handwritten digital lecture notes on the course web page.

Study Questions:

1. On pages 226-227 of the textbook, questions: 3, 5, 8, 9, 11.

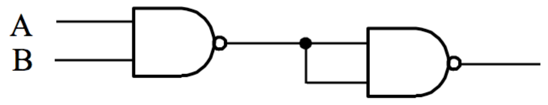
#3. Use a truth table to show that X = (X AND Y) OR (X AND NOT Y)

X	Y	(X AND Y) OR (X AND NOT Y)
0	0	0
0	1	0
1	0	1
1	1	1

Notice that in the truth table, the value of (X AND Y) OR (X AND NOT Y) always has the value of X.

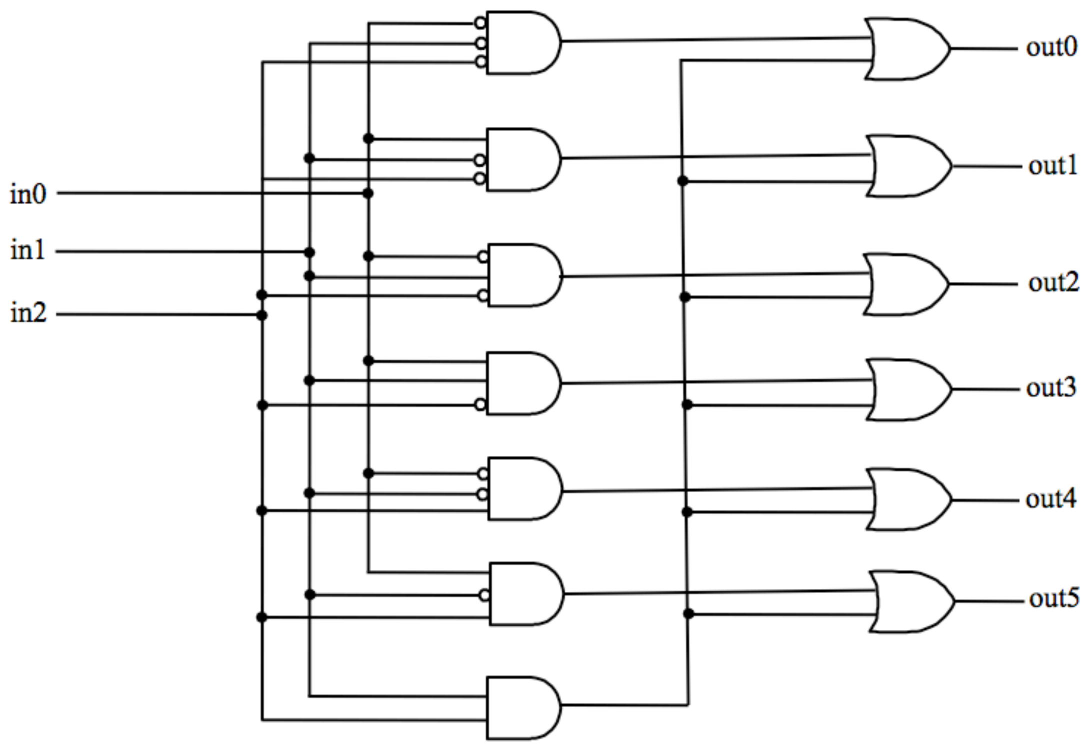
#5. Show and the AND function can be constructed from two NAND gates.

Since (A AND B) is equivalent to (NOT (NOT (A AND B)), you can construct AND by



2. Build a decoder with three input lines but with only six output lines. If the value of the input corresponds to 6 or 7, then all output lines should be asserted to signal an error.

This is just like an ordinary 3-input, 8-output decoder, except that if the input1 and input2 are both 1 (resulting in an input value of 6 or 7), all the outputs should be 1. This is accomplished by ANDing input1 and input2 and then sending that result to be OR'd with the normal value for each output. See below.



#8. Draw the logic diagram of a 2-bit encoder, a circuit with four input lines, exactly one of which is high at any instant, and two output lines whose 2-bit binary value tells which input is high.

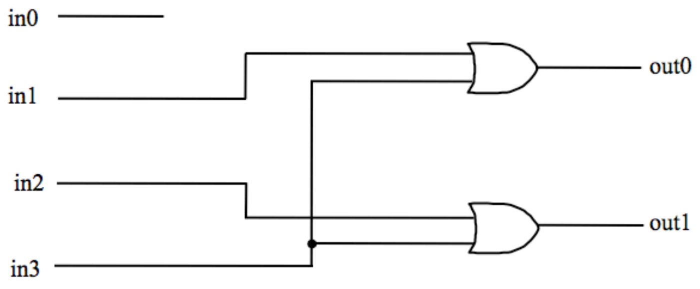
The truth table for an encoder (showing only the valid input values) is:

in3	in2	in1	in0	out1	out0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

From the truth table, we can see that

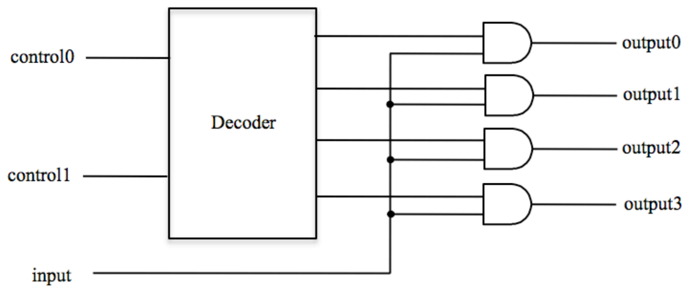
out1 = in2 OR in3
out0 = in1 OR in3

Again, this assumes only valid input values. The logic diagram is below.

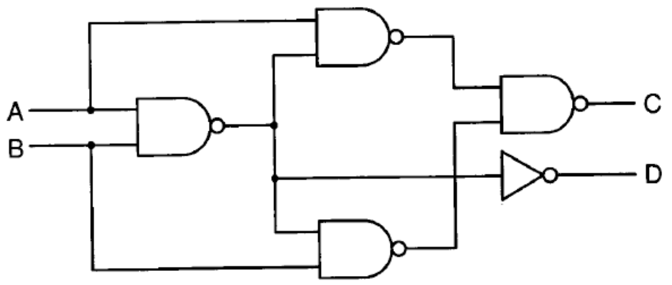


#9. Draw the logic diagram of a 2-bit demultiplexer, a circuit whose single input line is steered to one of the four output lines depending on the state of the two control lines.

While this is easily drawn from gates, another easy way to do it is to use a two-input decoder as follows.



#11. What does this circuit do?



You could do this by building a truth table or writing out the logical formulas and simplifying. The logical formulas would be:

C = NOT (NOT (A AND (NOT (A AND B))) AND (NOT ((NOT (A AND B)) AND B))

Because

A AND (NOT (A AND B)) == A AND (NOT B)

C = NOT (NOT (A AND (NOT B))) AND (NOT ((NOT (A AND B)) AND B))

and because

((NOT (A AND B)) AND B) = (NOT A) AND B,

C = NOT ((NOT A) OR B) AND (NOT ((NOT A) AND B))

and because NOT ((NOT A) AND B) = A OR NOT B,

C = NOT ((NOT A) OR B) AND (A OR (NOT B))

= (NOT ((NOT A) OR B)) OR (NOT (A OR (NOT B)))

= (A AND (NOT B)) OR ((NOT A) AND B))

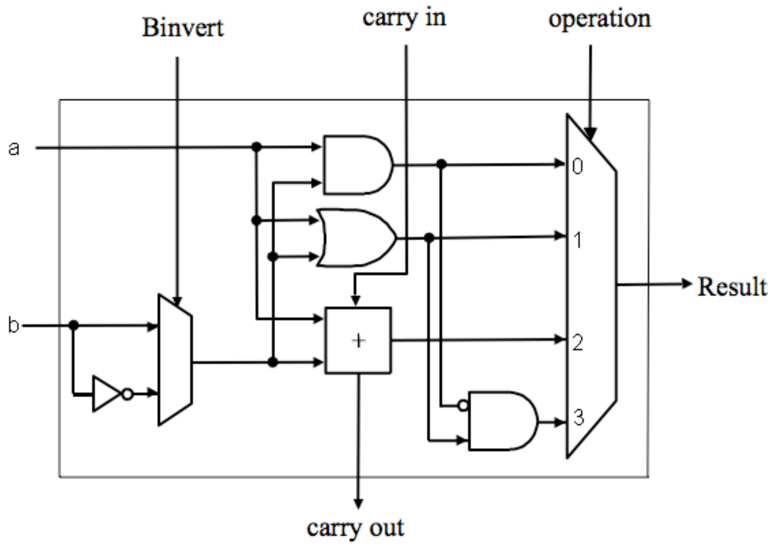
= A XOR B (by the definition of XOR)

D = NOT (NOT (A AND B))
= A AND B

The bottom line is, this would have been easier to do by a truth table!

3. Modify the 1-bit ALU covered in class (supporting AND, OR, +, and -) to implement an XOR (exclusive OR) operation. Draw the entire ALU, adding the fewest possible gates. Use the available op code 11 to select the XOR operation.

Because XOR is "OR and NOT AND" we can just use the output of the existing AND and OR gates in the ALU. See below.



4. Build a register file out of (already built) registers, mutiplexers, decoders, and logic gates that has the following inputs:

    - read select line
    - two write select lines, selecting two registers that can be written to.
    - two write data lines, for writing data to the two registers selected.
    - two write enable lines, for enabling writing to one or both of the selected registers.

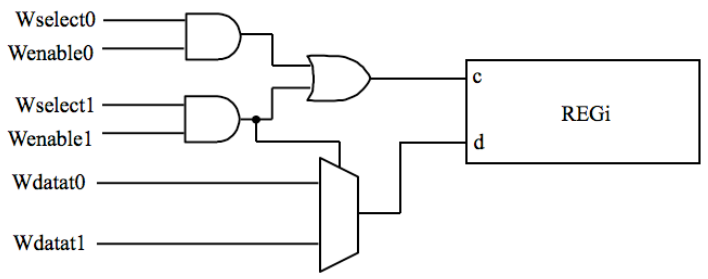
    and an output line for the data that was read from the selected register. Note that this is like the register file covered in class, except that instead of reading from two registers and writing to one register, it supports the writing to two registers and the reading from one register.

The reading side of the register file is easy. Instead of two multiplexers choosing the values of two registers to output, we only need one multiplexer. However, on the writing side of the register file, we now need two decoders, choosing two registers to write to. Therefore, a register is written to if either of the following conditions is true:

- **The first write select (Wsel0) selects the register and the first write enable (Wenable0) is true, or**
- **The second write select (Wsel1) selects the register and the second write enable (Wenable1) is true.**

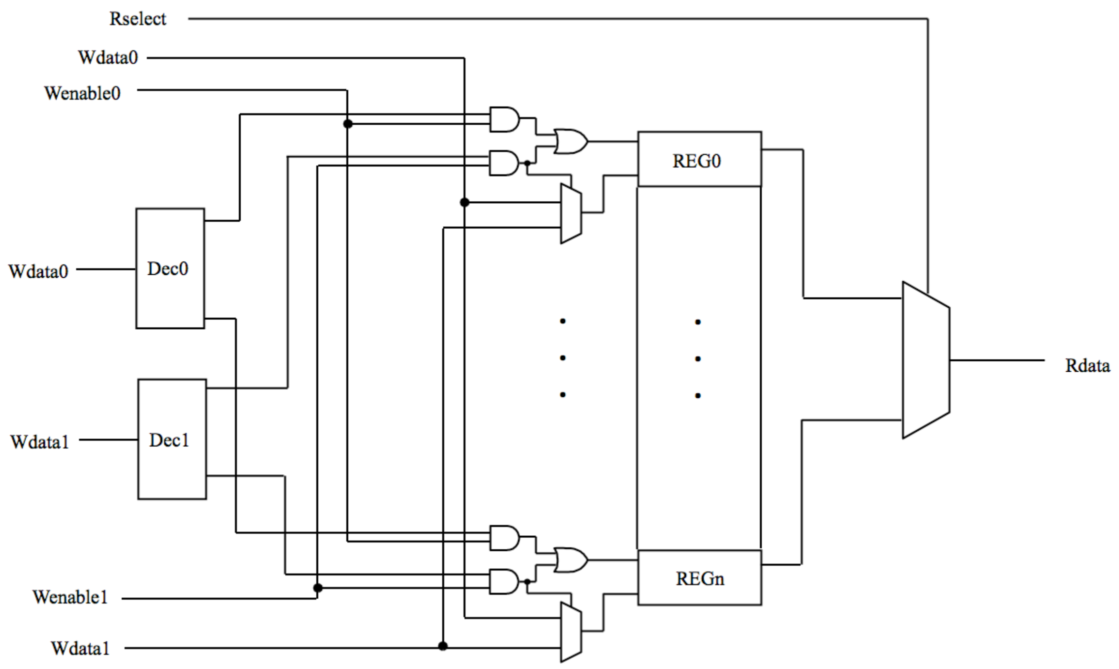
Consider the figure below, showing the configuration for a single register. Note that, as input to the "c" line of the register (the clock line) are gates implementing exactly

the condition above. The Wselect0 or Wselect1 lines are turned on by the decoders if the Wsel0 or Wsel1 input lines select this particular register.



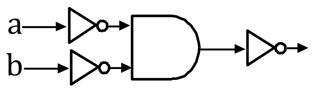
The next question is, what is sent to the data input line of the register? We have to choose between the first write data (Wdata0) or the second write data (Wdata1). To select between them, we use a one-bit multiplexer as shown in the figure above. The select line of the multiplexer is just Wselect1 AND Wenable1. That is, if this register has been selected to be written to by the second select line, then the second data line is sent to the register. Otherwise the first data line is sent.

The entire register file, then, just consists of replicating the above figure N times (for a register file with N registers) and adding the necessary decoders and the multiplexer, as shown below.



5. Can you build a device that, logically, behaves like an **or** gate from only **and** and **not** gates? If so, do so (just for the case where **and** and **or** gates have only two inputs). If not, explain why not. Then, answer the same question, except with regards to constructing an **and** gate from only **or** and **not** gates.

Yes. Since a OR b is the same as NOT ((NOT a) AND (NOT b)), the circuit would be:



6. Consider the register file discussed in class (shown on the first page of the November 11 lecture notes), constructed from D flip-flops.

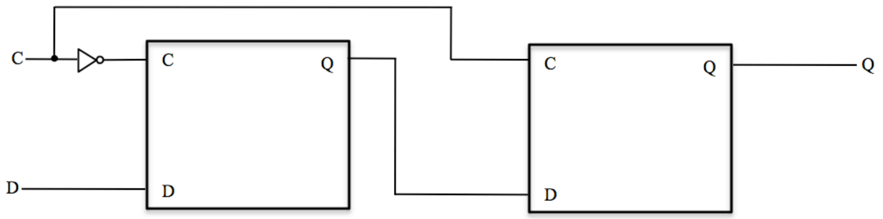
    a. Suppose that, when the clock line is asserted, the values of the "Read register number 1" and "Write register" inputs are both 4 (i.e. 000...100) and "Write" (i.e. write-enable) is asserted. When the "Read data 1" output stabilizes (while the clock is still asserted), would the value of "Read data 1" necessarily be the same as the value of the "Write data" input? Explain.

No. The value on the "Read data 1" line would be the value that was sitting in register 4 at the end of the previous clock cycle (i.e. prior to the most recent rising edge of the clock). This is not necessarily the same value that appears on the "Write Data" during the current clock cycle.

    b. If your answer to part b was "yes", then describe a way of reading the old value of a register at the same time as writing a new value to the same register. If your answer was "no", then indicate (precisely) when the value of "Write data" would appear on the "Read data 1" line, assuming that the values of "Read register number 1" and "Write register" are held constant.

Registers are constructed from D Flip-Flops, which don't change their output until after falling edge of the clock. Thus, the value written to a register using the "Write Data" line would not appear on the registers output (i.e. the "read data" line) until after the falling edge of the clock.

7. The last page of the November 9 lecture notes shows D Flip-Flop with a falling edge trigger. Draw the logic for a D Flip-Flop with a rising edge trigger (following the example in the lecture notes, not in the textbook).



8. Build a two-bit multiplier out of gates (i.e. it multiplies two two-bit numbers resulting in a 4-bit number). As usual, it may be easiest to start out with a truth table.

Building a truth table for to determine the formulas for the outputs and then simplifying gives you (where the two bits for operand a are a0 and a1 and the bits for operand b are b0 and b1):

out0 = a0 AND b0

out1 = ((a1 AND b0) OR (b1 and a0)) AND (NOT ((a1 AND b0) AND (b1 and a0)))

out2 = (a1 AND b1) AND (NOT (a0 AND b0))

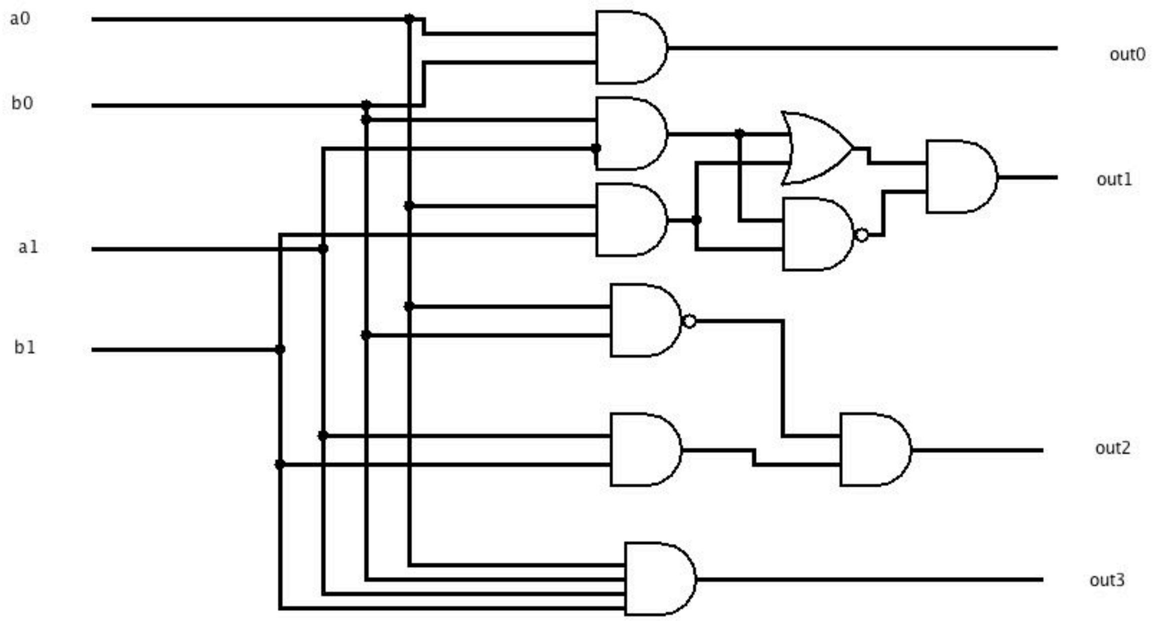
out3 = (a1 AND b1 AND a0 AND b0)

One could also write out the multiplication as you would do it by hand (where 1-bit multiplication is just an AND), by filling in the blanks below (left to the reader).

```
         a1  a0
         b1  b0
        ___ ___
    ___ ___
_______________

___ ___ ___ ___
```

The formulas arrived at would be the same as above. The logic diagram is:



9. True/False Questions

   a. **F** A circuit containing only OR and NOT gates must be a combinational circuit.

   b. **F** For two's complement numbers, the negative of a number can be found by adding one and then inverting the bits.

   c. **F** An disadvantage of two's complement numbers is that, unlike a sign-and-magnitude representation, you cannot tell if a number is negative by looking at only one bit.

d. **T** The number of rows of a truth table depends on the number of inputs, not the number of outputs.

e. **F** The Bnegate line is connected to the carry-in line of each 1-bit ALU within a 32-bit ALU.

f. **T** After a DRAM cell is read, the value read has to be written back to the cell.

g. **T** Asserting a word line and asserting (to "high" voltage) a bit line writes a 1 to a DRAM cell.

h. **F** SRAM is "static" in the sense that if the power is turned off, SRAM will continue to store data (e.g. as in flash memory in MP3 players or USB thumb drives).

i. **T** In the two's complement number representation, a negative number with more leading ones is larger (i.e. less negative) than a negative number with fewer leading ones.

j. **F** If a wire carries a logical value of 0, its voltage level will be 0 Volts.

k. **T** In an S-R Latch, if both S and R inputs are asserted, Q and $\overline{Q}$ will both be 0.

l. **T** A circuit diagram showing a multiplexer with sixteen 32-bit input lines and four select lines is actually referring to an array of 32 multiplexers, each having sixteen one-bit input lines and four select lines.

m. **F** Doubling the number of registers of a register file (but leaving everything else the same) will double the number of input lines to the register file.

n. **T** In a DRAM, all cells have to be periodically refreshed, not just the ones that have been read.

o. **F** When shifting a two's complement number to the left, an overflow can only occur if the number is negative, since positive numbers have a zero in the leftmost bit**. [Here's why: Shifting a large positive number to the left can result in a negative number.]**

p. **T** To perform the operation A – B, where A and B are numbers represented in two's complement, one can build hardware to perform the following steps: flip the bits of A, add B with a carry-in of 1, flip the bits of the result, and then add 1.

<u>Processors (RISC v. CISC, Pipelining)</u>

Read: Section 2.1 in the textbook.
Handwritten pipelining lecture notes on the course web page.

Study Questions:

1. If a single CISC instruction can achieve the same result as multiple RISC instructions, why isn't a program compiled for a CISC machine necessarily faster than the same program compiled (into more instructions) for a RISC machine?

The sum of the execution times for the RISC instructions may be less than the execution time of the CISC instruction. That is, the CISC instruction may take many cycles to execute. Furthermore, the clock rate of the CISC machine may need to be slower.

2. How does the clock rate factor into the RISC v. CISC discussion?

By simplifying the instruction set, removing complex instructions, a RISC processor may be able to achieve a faster clock rate than a comparable CISC processor.

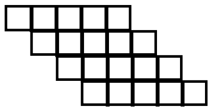
3. Describe in your own words what a "control hazard" is in a pipeline. Then, give an example of a sequence of simple x86 assembly instructions that presents a control hazard in a pipelined processor.

A control hazard refers to the situation in which an instruction currently in the pipeline is a conditional branch instruction and thus it cannot be immediately determined which is the next instruction to feed into the pipeline.

4. Assume a simple 5-stage pipeline (IF, ID, EXE, DF, W) as we did in class, where, in the ideal world, each stage takes a single cycle. Also, assume there are no cache misses. How many cycles would the following code take to execute if there is no special hardware to improve performance in the presence of hazards?

```
        Intel                       AT&T
mov     eax,[ecx+100]       mov     100(%ecx),%eax
mov     ebx,[ecx+104]       mov     104(%ecx),%ebx
add     eax,ebx             add     %ebx,%eax
mov     [ecx+108],eax       mov     %eax,108(%ecx)
mov     edx,[ecx+108]       mov     108(%eax),%edx
add     eax,edx             add     %edx,%eax
```

Draw a picture of the kind we drew in class,



representing the pipelined execution of the above instructions. (Note: Of course, it won't look exactly like the above picture. Also, be sure to identify the various stages: IF, ID, etc.).

The above code takes 14 cycles to execute, as shown below.

	Cycle:	1	2	3	4	5	6	7	8	9	10	11	12	13	14
mov	eax,[ecx+100]	IF	ID	DF	E	W									
mov	ebx,[ecx+104]		IF	ID	DF	E	W								
add	eax,ebx			IF	ID	DF	stall	E	W						
mov	[ecx+108],eax				IF	ID	stall	DF	stall	E	W				
mov	edx,[ecx+108]					IF	ID	stall	DF	stall	stall	E	W		
add	eax,edx						IF	ID	stall	DF	stall	stall	stall	E	W

5. Suppose that a processor feeds a new instruction into the pipeline every cycle, even though the previous instruction might be a branch (i.e. a jump or conditional jump). If the branch is taken, though (as determined by the Execute stage), the two succeeding instructions currently in the pipeline, i.e. those in the instruction fetch and instruction decode stages, should not be executed. It would be complicated and expensive, though, to

try to stall the pipeline and remove those two instructions from the pipeline. Describe, in detail, how you would implement the pipeline to allow the two instructions immediately following the branch to proceed through the pipeline but not violate the meaning of the program being executed.

Perhaps the easiest way to do this is to disable the Write stage of the two instructions following the branch instruction in the pipeline. One could add logic to the pipeline where the Execute stage has an output line into the Write stage that is asserted when a conditional branch is taken.

6. True/False Questions

   a. **T** The number of stall cycles due to data hazards occurring in a pipeline during program execution might be reduced by careful instruction scheduling by a compiler (i.e. by rearranging the order of instructions).

   b. **F** A control hazard occurs in a pipelined CPU when two instructions try to access memory at the same time.

   c. **T** Having every instruction be the same number of bits simplifies pipeline design.

<u>Caches</u>

Read: Section 4.51 in the textbook.
Handwritten cache lecture notes on the web page (ignore the reference to "Chap. 7 in P&H").

Study Questions:

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

Two addresses that map to the same cache entry in a direct-mapped cache will cause a conflict cache miss. Those same two addresses would map to the same set in a set-associative cache, but allowing the corresponding values to co-exist in the cache – thereby avoiding a conflict miss.

2. Write a simple piece of code in C or assembly for which, when executed, you might expect to have a large number of conflict misses in the cache on a machine with a 1MB direct-mapped cache.

Trying to access locations that are 1 MB apart consecutively will result in a cache miss. As an example, write a program that creates two arrays that are 1 MB apart and then accesses corresponding elements of the two arrays.

```
int A[1<<18];  //256K words = 1MB
int B[1<<18];
main()
{  int i;
      for(i=0;i<1<<18;i++)
      A[i] = A[i] + B[i];
}
```

3. 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?

The address would comprise:

- **2 bits for the byte offset within the word**
- **20 bits to determine the entry within the cache (since a 4MB cache contains 1M words, and log 1M = 20).**
- **10 bits for tag, since 32 bits overall minus the 22 bits listed above equals 10.**

4. 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?

The address would comprise:

- **2 bits for the byte offset within the word**
- **18 bits to determine the set within the cache (since each set is 16 bytes , i.e. 4 words, thus a 4MB cache contains 256K sets – and log 256K = 18).**
- **12 bits for tag, since 32 bits overall minus the 20 bits listed above equals 12.**

5. Suppose a program executes N instructions overall. This means that N instructions must be fetched from memory. Suppose also that 30% of the instructions need to fetch data from memory. Suppose that, on your computer, 95% of these memory accesses, for both instructions and data, result in cache hits. Finally, suppose that an instruction that causes no cache misses can be executed in a single cycle, but every cache miss adds an extra 100 cycles to the overall execution time. How many cycles will the program take to execute (note that if there were a 100% cache hit rate, execution would take N cycles).

Total cycles = N + (N * .05 * 100) + (N * .30 * .05 * 100)
= N + 5N + 1.5N
= 7.5N

6. Suppose you were not happy with the performance of the computer being used in question 5 above. After shopping around, you realized you could, for the same amount of money, replace the processor with one of the following:

   - A processor that was twice as fast (i.e. twice the clock speed), but with the same cache size and cache hit rate (95%).
   - A processor with the same clock speed, but with a larger cache so that the hit rate went up to 98%.

   Which would you buy? Explain why (i.e. show quantitatively which would be faster).

The faster processor would take the name number of cycles (7.5N), but each cycle would be twice as fast as on the old processor, thus the new execution time would be equivalent to 3.75N of the old processor cycles.

For the processor with the bigger cache,

Total cycles = N + (N * .02 * 100) + (N * .30 * .02 * 100)
= N + 2N + 0.6N
= 3.6N

Thus, you should choose the processor with the bigger cache (but a slower clock).

7. True/False Questions

   a. **T** A four-way set associative cache means that there are four cache entries in each set.

   b. **F** A 2-way set associative cache requires fewer gates to implement than a direct-mapped cache (think about it). **[A set associative cache requires the gates to compare the tag in an address against all the tags in a set.]**