EECS 470 Homework 3

This assignment is worth a bit less than 2% of your grade in the class and is graded out of 30 points. Remember you may drop one homework assignment or quiz score.

The following 32-bit MIPS code segment is used in both problem 1 and 2.

```
loop:   LD      R2, 0(R3)
        SLTI    R4, R2, 0
        BNEZ    R4, skip
        ADDI    R2, R0, 0
skip:   ADD     R6, R2, R6
        ADDI    R3, R3, 4
        BNE     R3, R7, loop
```

Assume R0=0 and R7=R3+4000 and that loop=0x408.

1. Consider the *gshare* branch predictor running on the above assembly code and starting with the table as shown. The global branch history is 101. Show what the table looks like after the 2nd, 4th, and 6th branch have been resolved. How many branches are predicted correctly? Assume that bits 27 to 29 of the PC are used, and that the first value loaded is positive while the rest are negative. Assume we are shifting the global branch history to the left and that we are using the branch predictor found on page 198. Show your work. [5]

Index	State
0	01
1	11
2	10
3	11
4	10
5	11
6	10
7	00

2. For this problem, assume that a random 40% of all values loaded by the LD instruction are negative. (You can assume each load is treated as a Bernoulli trial.) [8]
    a. How many instructions will be executed? [2]
    b. How many times would you expect a standard 1-bit PC-based branch predictor will be wrong? A 2-bit branch predictor (saturating up-down counter)? hint: the 2-bit question is harder than it looks. You probably need either a Markov model or to write a quick simulator to model it. Show your work. [6]

3. Consider an ISA where the instructions are 32-bits each and there are 64 general-purpose registers. Say in this ISA there is class of instructions which take 2 register arguments and one 16-bit immediate. If instructions of this class used half of all the possible instruction encodings, how many instructions of this class are there? Show your work. [3]

4. Branch alignment is a compiler optimization that converts strongly biased taken branches to strongly biased not-taken branches, by rearranging instruction layout in memory. How can this optimization be used to improve BTB performance? (keep your answer to a few sentences.) [3]

5. A major focus in modern computer architectures is instruction level parallelism. The following questions ask you to think about ILP and speculation. Consider the following psuedo-assembly code. Notice that each instruction has a name (A, B, etc.) given as a comment. [11]

```
loop: r1 ← r6+r4          // inst A
           r2 ← r3+r5     // inst B
           r6 ← r6+r1     // inst C
           r2 ← r6+r4     // inst D
           r3 ← r3+r2     // inst E
           r7 ← r8+r1     // inst F
           if(r7!=0)goto loop  // inst G
exit:     halt
```

a) Draw a *data dependency* (not control or name dependency) diagram for the above code assuming that the loop was taken the first time and not taken the second. Use the instruction name (A, B, etc.) and the loop iteration number (1 or 2) to describe each executed assembly instruction (so A1, B1, B2, etc.). Draw arrows from an instruction to the instruction it is *directly* dependent on. [3]

b) Assume we have a processor that can start and finish any number of instructions in one cycle. *Assuming we don't speculate anything (values, branches being taken or not taken, etc.)* but we are willing to execute out-of-order and that we use register renaming to deal with name dependencies, how many cycles would the code described in part a) take to execute? Clearly explain your answer. You might find it useful to show us which instructions are executing at any given time. [4]

c) Consider the same code as in parts a and b (loop taken the first time not take the second time). Say we can correctly guess (speculate) on the taken/not taken value of the branch (both times). Further, that we can correctly speculate on the *result* of any one instruction execution (so B1 but no other for example.

What would be the best instruction execution to guess in terms of being able to reduce the number of cycles required to execute the whole piece of code? What would be the time to execute the code if we could successfully speculate on that result? Show and explain your work. [4]