Computer Architecture Qual., Spring 2013

- This is a 2-hour, closed-book, closed-note test. Calculator allowed.
- This test has 4 questions for a total of 100 points.

1. [25 points] **Cache performance**

   The processor has a 2-level cache memory hierarchy. Both 2-way 32KB L1 instruction cache and 2-way 32KB L1 data cache have 2-cycle access time. Unified 8-way 1MB L2 cache has 10-cycle access time. DRAM memory has 200-cycle access time. When a graphics application runs on this processor, instruction access shows 3% miss rate on L1 instruction cache and 5% miss rate on L2 cache. Data access shows 10% miss rate on L1 data cache and 15% miss rate on L2 cache. 20% of instructions are loads and stores. Assume the base CPI using a perfect memory system is 1. Compute the average CPI of this processor.

2. [25 points] **Branch prediction**

   Consider the following code running on the MIPS processor. Two branch instructions, `B1` and `B2`, access same 2-bit bimodal predictor, which is initialized to the strongest *not taken* state. Initially, `R7` and `R8` registers have 16 and 12, respectively, and other registers (`R0, R1, R2`) have 0.

```
          ADDI  R1, R0, #0      ;R1 = R0 + 0
zoo:      NOP
          ADDI  R2, R0, #0      ;R2 = R0 + 0
foo:      NOP
          ADDI  R2, R2, #4      ;R2 = R2 + 4
          BNE   R2, R7, foo     ;branch if R2!=R7  <-------- B1
          ADDI  R1, R1, #4      ;R1 = R1 + 4
          BNE   R1, R8, zoo     ;branch if R1!=R8  <-------- B2
```

   (a) [10 points] Show the prediction results for each execution of `B1` and `B2`.

   (b) [10 points] If `R8` is initialized to 1024, compute the total number of mis-predictions.

   (c) [5 points] Draw a state transition diagram for the 2-bit bimodal branch predictor.

3. [25 points] **Memory consistency**

Consider the following code segment running on three processors (P1, P2, P3). The initial values of `A, B, X, Y, Z` are 0.

```
P1:                    P2:                    P3:
   X = 2;                 while (A == 0);        while (B == 0);
   A = 1;                 Y = X;                 Z = Y;
                          B = 1;
```

(a) [6 points] At the end of the code segment, what are the values you would expect for `Y` and `Z`?

(b) [6 points] A system with a general-purpose interconnection network, a directory-based cache coherence protocol, and support for nonblocking loads generates a result where `Y` is 0. Describe a scenario where this result is possible.

(c) [6 points] To make the system sequentially consistent, what are the key constraints you would need to impose?

(d) [7 points] Assume that a processor supports a relaxed memory consistency model, which requires synchronization to be explicitly identified. A *barrier* instruction ensures that all memory operations preceding the barrier instruction complete before any memory operations following the barrier are allowed to begin. Change the above code segment by using barrier instruction to ensure that you get the intuitive results of sequential consistency.

4. [25 points] **Dynamic scheduling**

Let's consider dynamic scheduling based on the Tomasula algorithm. We assume multiply takes 4 cycles. We assume first load takes 8 clock cycles (cache miss) and second load takes 1 cycle (cache hit). Static branch predictor, predicted-taken, is used. The hardware has 2 reservation stations for FP multipliers, 3 load buffers, and 3 store buffers. The following code is dynamically scheduled by the single-issue Tomasula algorithm with speculation.

```
loop:   L.D    F0, 0(R1)      ;F0 = Mem[R1 + 0]
        MUL.D  F4, F0, F2     ;F4 = F0 + F2
        S.D    F4, 0(R1)      ;Mem[R1 + 0] = F4
        SUBI   R1, R1, #8     ;R1 = R1 - 8
        BNEZ   R1, loop       ;branch if R1 is not zero
```

Show the time diagram of issue (IS), execution (EX), write-back (WB), and commit (CM) stages for each instruction.

Iteration	Instruction	IS	EX	WB	CM
1	`L.D F0, 0(R1)`	1			
1	`MUL.D F4, F0, F2`				
1	`S.D F4, 0(R1)`				
1	`SUBI R1, R1, #8`				
1	`BNEZ R1, loop`				
2	`L.D F0, 0(R1)`				
2	`MUL.D F4, F0, F2`				
2	`S.D F4, 0(R1)`				
2	`SUBI R1, R1, #8`				
2	`BNEZ R1, loop`				