Problems 1, 3, and 5 will be graded. There are 60 points on these problems. Note: You must do all the problems, even the non-graded ones. If you do not do some of them, half as many points as they are worth will be subtracted from your score on the graded problems.

**Problem 1.** (15 points) In a deeply pipelined processor, a branch-target buffer is used for conditional branches only. Assume—

- 20% of the instructions are branches.
- The misprediction penalty is 5 cycles.
- The BTB miss penalty is 4 cycles.
- The BTB has a 90% hit rate and 90% branch-prediction accuracy,
- Instructions without branch stalls take 1 cycle.

What is the speedup of this processor with the branch–target buffer versus a processor that has a fixed 2-cycle branch penalty?

**Problem 2.** (20 points) [H&P 3.11] Figure 3.15 on page 206 and Figure 3.18 on page 208 show that the prediction accuracy of a local 2-bit predictor improves very slowly with increasing branch-prediction buffer size, once size exceeds some threshold. Because prediction buffers must be of finite size, two or more branches may be mapped to the same buffer entry. While it is possible for sharing to improve branch prediction by accidentally allowing sharing of information between related branches, typically the sharing results in destructive interference. For the following, assume that prediction accuracy is always worse when branches share a predictor.

(a) For a branch-prediction buffer implementing a given type of predictor, what characteristic of any program guarantees that prediction accuracy as a function of increasing buffer size must eventually become constant (i.e., independent of buffer size)? *Hint:* Examine the contents of and compare the prediction accuracy of branch-prediction buffers of different sizes on a simple code fragment such as the following:

```
Loop: DSUBI R1, R1, #1
        BNEZ R1, Loop
        LD R10, 0(R3)
```

(b) Figure 3.15 shows that, to within the precision of the measurements, the SPEC89 benchmarks *nasa7*, *tomcatv*, and *gcc* were the only programs to have less branch misprediction when buffer size increased from 4096 entries to infinite. Based on the answer to part (a), what quantitative measure can you infer about the machine instruction count of the executable codes for these four benchmarks?

(c) Can you infer anything similar to the result in part (b) about the instruction counts of the other seven benchmarks?

(d) How might an optimizing compiler improve prediction accuracy for the other seven benchmarks in part (c), and when would this be and not be possible?
Problem 3. (20 points) [H&P 3.3] Consider the following MIPS assembly code:

```
LD     R1, 45(R2)
DADD   R7, R1, R5
DSUB   R8, R1, R6
OR     R9, R5, R1
BNEZ   R7, target
DADD   R10, R8, R5
XOR    R2, R3, R4
```

(a) Identify each dependence by type; list the two instructions involved; identify which instruction is dependent; and, if there is one, name the storage location involved.

(b) Use information about the MIPS five-stage pipeline from Appendix A of H&P, and assume a register file that writes in the first half of the clock cycle and reads in the second half-cycle forwarding. Which of the dependences that you found in part (a) become hazards and which do not? Why?

Problem 4. (20 points) Register renaming and delayed execution are the two approaches taken by Tomasulo's Algorithm to remove WAR and WAW hazards and RAW hazards respectively. Consider this code sequence:

1. LD R1, (R2)
2. ADD R3, R4, R1
3. SUB R5, R3, R7
4. MUL R5, R4, R8

Assume the following –

- The functional units present in the system are Load1, Load2, Add1, Add2, Mult1 and Mult2.
- The functional unit latencies of the operation types, in execution cycles, are ADD: 5, SUB: 5, LD: 5 and MUL: 7.

Reservation stations

<table>
<thead>
<tr>
<th>Name</th>
<th>Busy</th>
<th>Op</th>
<th>V_j</th>
<th>V_k</th>
<th>Q_j</th>
<th>Q_k</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Load2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mult1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mult2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(a) Show the values in the reservation stations after 4 cycles, and tell which hazards were avoided by register renaming.
Problem 5. (25 points) [H&P 3.6(a)] In this exercise, we will look at running Tomasulo’s algorithm on a common vector loop. The loop is the so-called DAXPY loop (double-precision $aX$ plus $y$) and is the central operation in Gaussian elimination. The following code implements the operation $Y = aX + Y$ for a vector of length 100. Initially, $R1 = 0$ and $F0$ contains $a$.

```assembly
Foo:
    L.D  F2,0(R1) ; load $X(i)$
    MUL.D F4,F2,F0 ; multiply $a*Y(i)$
    L.D  F6,0(R2) ; load $Y(i)$
    ADD.D F6,F4,F6 ; add $a*Y(i) + Y(i)$
    S.D  F6,0(R2) ; store $Y(i)$
    DADDUI R1,R1,#8 ; increment $X$ index
    DADDUI R2,R2,#8 ; increment $Y$ indexs
    DSGTUI R3,R1,#800 ; test if done
    BEQZ R3,foo ; loop if not done
```

The pipeline functional units are described as follows:

<table>
<thead>
<tr>
<th>FU type</th>
<th>Cycles in EX</th>
<th>Number of FUs</th>
<th># of reservation stns.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer</td>
<td>1</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>FP adder</td>
<td>4</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>FP multiplier</td>
<td>15</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

Assume the following:
- Functional units are not pipelined.
- There is no forwarding between functional units; results are communicated by the CDB.
- The execution stage (EX) does both the effective address calculation and the memory access for loads and stores. Thus the pipeline is IF/ID/IS/EX/WB.
- Loads take 1 clock cycle.
- The issue (IS) and write-result (WB) stages each take one clock cycle.
- There are 5 load buffer slots and 5 store buffer slots.
- Assume that the BNEQZ instruction takes 0 clock cycles.

Use the single-issue Tomasulo MIPS pipeline of Figure 3.2 with the pipeline latencies below. Show the number of stall cycles for each instruction and what clock cycle each instruction begins execution (i.e., enters its first EX cycle) for three iterations of the loop. How many clock cycles does each loop iteration take? Report your answer in the form of a table like that in Figure 3.25.

<table>
<thead>
<tr>
<th>Instruction producing result</th>
<th>Instruction using result</th>
<th>Latency in clock cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>FP multiply</td>
<td>FP ALU op</td>
<td>6</td>
</tr>
<tr>
<td>FP add</td>
<td>FP ALU op</td>
<td>4</td>
</tr>
<tr>
<td>FP multiply</td>
<td>FP store</td>
<td>5</td>
</tr>
<tr>
<td>FP add</td>
<td>FP store</td>
<td>3</td>
</tr>
<tr>
<td>Integer operation (incl. load)</td>
<td>Any</td>
<td>0</td>
</tr>
</tbody>
</table>