Instructions

1. No books, papers, notes, or any other typed or written materials are allowed. No calculators or other electronic materials are allowed.

2. Please do not turn in loose scrap paper. Limit your answers to the space provided if possible. If this is not possible, please write on the back of the same sheet. You may use the back of each sheet for scratch work.

3. In all cases, show your work. No credit will be given if there is no indication of how the answer was derived. Partial credit will be given even if your final solution is incorrect, provided you show the intermediate steps in reaching the final solution.

4. If you believe a problem is incorrectly or incompletely specified, make a reasonable assumption and solve the problem. The assumption should not result in a trivial solution. In all cases, clearly state any assumptions that you make in your answers.

5. This exam has 6 problems and 12 pages (including this one). All students should solve problems 1 through 5. Only graduate students should solve problem 6. Please budget your time appropriately. Good luck!

<table>
<thead>
<tr>
<th>Problem</th>
<th>Maximum Points</th>
<th>Received Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>10 Graduate Problem</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>Undergraduate (48 points), Graduates (58 points)</td>
<td></td>
</tr>
</tbody>
</table>
Problem 1 [4 points]:

Consider the following instruction mix for a hypothetical benchmark.

<table>
<thead>
<tr>
<th>Instruction type</th>
<th>Percentage of all instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load from a memory address</td>
<td>15.5%</td>
</tr>
<tr>
<td>Store to a memory address</td>
<td>7.3%</td>
</tr>
<tr>
<td>Multiply</td>
<td>12.2%</td>
</tr>
<tr>
<td>Other</td>
<td>65%</td>
</tr>
</tbody>
</table>

You are considering making one of two possible enhancements to the machine. Enhancement E1 can make multiply operations run six times faster than before while enhancement E2 can make memory accesses run two times faster than before. Determine the speedup from each enhancement. You can express your answer in terms of an equation with all the variables explicitly substituted. You are not required to perform numerical calculations.

**Solution:**

The enhancements mentioned only affect multiply and memory access instructions.

Recall Amdahl’s law,

\[
\text{Speedup}_{\text{overall}} = \frac{1}{(1 - f) + f/S}
\]

First for our hypothetical benchmark:

\[
f_{\text{mul}}(E1) = 12.2% \\
f_{\text{mem}}(E2) = 15.5% + 7.3% = 22.8%
\]

\[
\text{Speedup}_{\text{mul}}(E1) = \frac{1}{(1 - 0.122) + 0.122/6} \\
\text{Speedup}_{\text{mul}}(E1) = \frac{1}{(1 - 0.122) + 0.203/6} \\
\text{Speedup}_{\text{mul}}(E1) = 1.113
\]

\[
\text{Speedup}_{\text{mem}}(E2) = \frac{1}{(1 - 0.228) + 0.228/2} \\
\text{Speedup}_{\text{mem}}(E2) = \frac{1}{(1 - 0.228) + 0.114} \\
\text{Speedup}_{\text{mem}}(E2) = 1.129
\]

So E2 will give the best performance improvement for this benchmark.

**Grading:**

2 points for the Amdahl's law equation. 1 point for plugging in the correct values for enhancement 1 and 1 point for plugging in the correct values for enhancement 2.
Problem 2 [6 points]:

Suppose we have a deeply pipelined processor for which we implement a branch-target buffer for conditional branches only. Assume that the misprediction penalty is always four cycles and the buffer miss penalty is always three cycles. Assume a 90% hit rate in the buffer, 90% prediction accuracy (for the hits), and 15% branch frequency. How much faster is the processor with the branch-target buffer versus a processor that has a fixed two-cycle branch penalty? Assume a base clock cycle per instruction (CPI) without branch stalls is one cycle.

Solution: For this problem we are given the base CPI without branch stalls. From this we can compute the number of stalls given by no BTB and with the BTB.

\[
\text{Speedup} = \frac{\text{CPI}_1}{\text{CPI}_2} = \frac{1 + \text{Stalls}_1}{1 + \text{Stalls}_2} \quad \text{where subscript 1 represents the system without the BTB and subscript 2 represents the system with the BTB}
\]

\[
\text{Stalls}_1 = 15\% \times 2 = 0.30
\]

\[
\text{Stalls}_2 = (1.5\% \times 3) + (12.1\% \times 0) + (1.3\% \times 4) = 0.097
\]

\[
\text{Speedup} = \frac{1.0 + 0.30}{1.0 + 0.097} = 1.2
\]

Grading:
1 point for the stalls using no branch prediction. 1 point for the correct penalty for each of the three BTB cases: BTB Miss, BTB Hit with correct prediction, and BTB hit with incorrect prediction. 2 points for the correct use of the speedup formula.
Problem 3[15 points]:

This problem concerns Tomasulo’s algorithm (with reservation stations) with the reorder buffer as discussed in detail in the lecture notes, with the following changes/additions/clarifications.

<table>
<thead>
<tr>
<th>Functional Unit Type</th>
<th>Cycles in EX (NON-PIPELINED)</th>
<th>Number of Functional Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>FP Adder</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>FP Divider</td>
<td>6</td>
<td>1</td>
</tr>
</tbody>
</table>

1) Assume dual issue and dual commit.
2) Assume unlimited reservation stations.
3) Loads use the integer functional unit to perform effective address calculation during the EX stage. They also access memory during the EX stage. Loads stay in EX for 1 cycle.
4) If an instruction moves to its WB stage in cycle x, then an instruction that is waiting on the same functional unit (due to a structural hazard) can start executing in cycle x.
5) An instruction waiting for data on the CDB can move to EX in the cycle after the CDB broadcast.
6) Only one instruction can write to the CDB in one clock cycle. Branches/stores do not need the CDB.
7) When there is a conflict for a functional unit or the CDB, assume that the oldest (by program order) of the conflicting instructions gets access, while others are stalled.
8) Assume that the result from the integer functional unit is also broadcast on the CDB and forwarded to dependent instructions through the CDB (just like any floating point instruction).

Complete the blank entries of the following table, indicating the cycle numbers that an instruction spends in a specific stage (CM means commit). For the last column, enter all the reasons that the instruction experiences a stall (leave blank if there is no stall). The first row is filled for you.

<table>
<thead>
<tr>
<th>No</th>
<th>Instruction</th>
<th>IS</th>
<th>EX</th>
<th>WB</th>
<th>CM</th>
<th>Reasons for stall</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>L.D F0, 0(R1)</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>--</td>
</tr>
<tr>
<td>2</td>
<td>ADD.D F0, F0, F3</td>
<td>1</td>
<td>4-6</td>
<td>7</td>
<td>8</td>
<td>RAW due to F0 from (1)</td>
</tr>
<tr>
<td>3</td>
<td>DIV.D F8, F0, F6</td>
<td>2</td>
<td>11-16</td>
<td>17</td>
<td>18</td>
<td>RAW due to F0 from (2), Structural hazard due to (6)</td>
</tr>
<tr>
<td>4</td>
<td>L.D F6, 8(R1)</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>18</td>
<td>--</td>
</tr>
<tr>
<td>5</td>
<td>ADD.D F4, F6, F2</td>
<td>3</td>
<td>7-9</td>
<td>10</td>
<td>19</td>
<td>RAW due to F6 from (4), Structural hazard due to (2)</td>
</tr>
<tr>
<td>6</td>
<td>DIV.D F6, F6, F2</td>
<td>3</td>
<td>5-10</td>
<td>11</td>
<td>19</td>
<td>RAW due to F6 from (4)</td>
</tr>
<tr>
<td>7</td>
<td>L.D F6, 16(R1)</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>20</td>
<td>--</td>
</tr>
</tbody>
</table>
Problem 4 [9 points]

Consider a loop that is entered a trillion times in a program. Each time it is entered, the loop performs 10 iterations. Each iteration executes five branches with the following outcomes (branch 1 occurs before branch 2 which occurs before branch 3 which occurs before branch 4 which occurs before branch 5 in each iteration):

<table>
<thead>
<tr>
<th>Iteration</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>branch1</td>
<td>T</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>branch2</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>branch3</td>
<td>T</td>
<td>N</td>
<td>T</td>
<td>N</td>
<td>T</td>
<td>N</td>
<td>T</td>
<td>N</td>
<td>T</td>
<td>N</td>
</tr>
<tr>
<td>branch4</td>
<td>T</td>
<td>T</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>N</td>
</tr>
<tr>
<td>branch5</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>N</td>
</tr>
</tbody>
</table>

When Branch 5 is not taken at iteration 10, the program counter leaves the loop. Assume there are no other conditional branches in the program besides the ones above. (There may be unconditional jumps, but assume those are ignored by branch predictors.) Also assume that each branch has its own predictor entries; i.e., there is no aliasing among the multiple branches on the predictor entries.

Of all the dynamic branch predictors studied in class, state the predictor that will give the best misprediction rate for each of the following branches. State the misprediction rate of that predictor. In a case where multiple predictors will give the best misprediction rate, give the cheapest such predictor. Explain why. (No credit without explanation.) Focus on the steady state; i.e., ignore the first few times the loop is executed when evaluating your misprediction rate.

(A) Branch 1:
Solution:
Branch 1 is always opposite the previous branch, which is branch 5. A global (1,1) predictor will always predict correctly, after the first execution of the loop. Half credit for saying 2-bit predictor.

(B) Branch 2:
Solution:
Branch 2’s outcome follows the pattern established by the previous 3 branches (branches 4 and 5 from the previous iteration and branch 1 from the same iteration). Therefore, a (3,1) correlating predictor is best.

(C) Branch 5 – for this part and this part alone, do not consider correlating predictors:
Solution:
Branch 5 is almost always taken. We need to use a 2-bit predictor. It will have one misprediction each time the code exits the loop. In contrast, a 1 bit predictor will have two mispredictions— at the entry and exit of each loop invocation.
Grading:
3 points for each part. Half credit for a predictor which is more expensive, but attains the minimal error rate. Half credit if the explanation gives an incorrect prediction rate. No credit without explanation.

Problem 5 [14 points]:

Consider the standard five stage pipeline discussed in class with the following extensions/clarifications:

- The MEM stage takes 4 cycles but is pipelined.
- Branches are resolved in the decode stage, and have one branch delay slot.
- All instructions take one cycle in the EX stage. Address calculation for memory instructions occurs in the EX stage as in the basic pipeline.
- Assume all forwarding paths as needed.

Consider the loop below. It calculates the sum of a list of numbers stored in an indirect representation. Instead of storing the number, the locations 0(R1), 4(R1), 8(R1), etc. contain the addresses of the values. This is similar to the memory access pattern of sparse matrix codes. The sum is accumulated in register F1.

loop:
1) LD R3, 0(R1)
2) LD.D F2, 0(R3)
3) ADD.D F1, F1, F2
4) SUBIU R2, R2, #1
5) BNEZ R2, loop
6) ADDIU R1, R1, #4

Part A [3 points]:

List all the stalls incurred by the above loop and the reason for the stalls. If a dependence results in multiple stall cycles, indicate the number of cycles.

Solution:
1->2: 4 stall cycles due to RAW on R3.
2->3: 4 stall cycles due to RAW on F2.
4->5: 1 stall cycle due to RAW on R2
Grading: ½ point for identifying each stall. ½ point for the correct number of cycles. 3 points total.

Part B [4 points]:

Software pipeline the loop to minimize the stalls. Assume infinite registers are available. Only the most efficient solution will fetch a perfect score. Only provide the steady state code. Do not worry about start-up and finish-up code. Do not unroll the loop for this part.

Solution:

loop:
   LD.D F2, 0(R3)
   LD R3, 0(R1)
   SUBIU R2, R2, #1
   ADDIU R1, R1, #4
   BNEZ R2, loop
   ADD.D F1, F1, F2

Grading: 2 points if the solution appears to “spread out” the two loads and the add in different iterations. 3 points for a mostly correct solution. 4 points for a perfect solution.

Part C [5 points]:

Now consider loop unrolling without any software pipelining. What is the minimum number of original iterations that you would need to include in the unrolled loop to minimize the stalls in the above code? Show the unrolled code. Ensure you minimize the loop overhead in your unrolled code even if it means that you are executing a different number of instructions in parts B and C. Assume you have infinite registers.

Solution: Four iterations of the original loop are needed in each iteration of the new loop.

loop:
   LD R3, 0(R1)
   LD R4, 4(R1)
   LD R5, 8(R1)
   LD R6, 12(R1)
   SUBIU R2, R2, #4
LD.D F2, 0(R3)
LD.D F4, 0(R4)
LD.D F6, 0(R5)
LD.D F8, 0(R6)
ADDIU R1, R1, #16
ADD.D F1, F1, F2
ADD.D F1, F1, F4
ADD.D F1, F1, F6
BNEZ R2, loop
ADD.D F1, F1, F8

Grading: 2 points for identifying the correct number of iterations. 2 points for correct unrolling of the code for the identified number of iterations – partial credit of 1 point if at least half of the unrolled address arithmetic and registers are correct. 1 point for scheduling for minimal stalls for the identified number of iterations.

Part D [1 point]
What is the advantage of using software pipelining over loop unrolling for the above code?

Solution: There are fewer static instructions in a software pipelined loop and so there is less pressure on the instruction cache.

Grading: 1 point for identifying fewer static instructions as the reason.

Part E [1 point]
What is the advantage of using loop unrolling over software pipelining for the above code?

Solution: There are fewer dynamic instructions because the loop overhead is reduced.

Grading: 1 point for correct answer.
Problem 6 [10 points]:

Consider the following loop:

```plaintext
LOOP: ADD RA, R*, R*
      SRA RB, RA
      ADD RC, RB, R*
      SLA RD, RC
      SUB RE, RD, R*
      BNE RF, RE LOOP
```

You need not concern yourself with the actual action of the loop or whether it is doing anything useful. Focus only on the dependences in the loop. Here all registers marked as R* indicate that there are no dependences for them and you need not care about them. Note that there are no RAW dependences across loop iterations. Assume the loop executes 10 times.

Part A [3 points]:

Consider a machine that uses Tomasulo’s algorithm for dynamic scheduling with renaming. Suppose the machine does not support speculation and so stalls on all branches until they are resolved. Suppose the machine can fetch, decode, and issue (to the reservations stations) an unbounded number of instructions per cycle. Assume the machine has unlimited functional units. How large an instruction window must the machine support to best exploit ILP in this code? Assume that the instruction window size can only be a multiple of six. Recall that the instruction window refers to the total number of instructions in flight at a time. Justify your answer and clearly state any assumptions you make.

Solution:

Due to the long dependence, instructions of a given iteration are serialized and none of them can execute out of order. Using an instruction window larger than the loop iteration will not be of benefit since the machine does not use speculation and the direction of the branch is dependent on the rest of the loop computation. Therefore, an instruction window size of 6 would be appropriate.

Grading: 3 points for the correct size and justification.
Part B [3 points]:

Suppose now that the machine in part (A) supports speculation and has a magical branch predictor with 100% accuracy. How would your answer to part (A) change?

Solution:

Since each iteration of the loop is independent of the others, all iterations could be executed in parallel. So to achieve maximum performance, an issue window of 60 should be used.

Grading: 3 points for the correct size and justification.

Part C [4 points]:

You are now told of some new research that does not require stalling on RAW dependences (e.g., value prediction allows speculation on values, and triggers a rollback if the speculation was wrong). Now how would your answer to part (B) change? Do you expect to see a change in IPC from part (B) to part (C)? Again, be sure to justify your answers.

Solution:

The answer to part (B) would remain the same. Instruction window size would still be 60 for the same reason as above. Since dependences in each iteration are broken via value prediction, IPC would be expected to increase provided that the value predictor has a reasonable prediction rate.

Grading: 2 points for the statement about instruction window size, 2 points for the statement about IPC. The comment about value prediction rate is not required. 4 points total.