CS 433 Practice Midterm Exam

Time: 2 Hours

Please print your name and NetID and circle the appropriate category in the space provided below.

<table>
<thead>
<tr>
<th>Name</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>NetID</td>
<td></td>
</tr>
<tr>
<td>Category</td>
<td>Undergraduate   Graduate</td>
</tr>
</tbody>
</table>

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>16</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>8 (for graduates only)</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>54 for undergraduates 62 for graduates</td>
<td></td>
</tr>
</tbody>
</table>
Problem 1 [4 points]

Suppose we apply an enhancement, E1, that speeds up 20% of a program by a factor of 4. And we apply another enhancement, E2, that speeds up another 10% of the program by a factor of 10. Assume the two enhancements are independent and affect different parts of the program (there is no overlap between the 20% and 10% of the program). What is the overall speedup for the entire program using both E1 and E2 simultaneously?

Solution:

This problem tests the understanding of Amdahl’s law.

According to Amdahl’s law:

\[
\text{Speedup} = \frac{\text{old latency}}{\text{new latency}}
\]

\[
\text{new latency} = (1 - f_1 - f_2) \times \text{old latency} + \frac{f_1}{S_1} \times \text{old latency} + \frac{f_2}{S_2} \times \text{old latency}
\]

\[
f_1 = 20\%, S_1 = 4, f_2 = 10\%, S_2 = 10
\]

\[
\text{new latency} = 0.76 \times \text{old latency}
\]

\[
\text{Speedup} = \frac{\text{old latency}}{0.76 \times \text{old latency}} = 1.32
\]

Grading: 1 point for listing Amdahl’s law and the standard equation. 1 point for the equation of this problem. 2 points for substituting the correct values in the equation. No deduction for calculation errors.
Problem 2 [16 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.

- Assume only the following functional units:

<table>
<thead>
<tr>
<th>Functional Unit</th>
<th>Cycles in EX</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer Add</td>
<td>1</td>
</tr>
<tr>
<td>Integer Mul</td>
<td>3</td>
</tr>
<tr>
<td>Integer Div</td>
<td>7</td>
</tr>
</tbody>
</table>

- The processor can issue and commit at most one instruction per cycle.
- The common data bus (CDB) supports only one broadcast per cycle.
- Assume you have an unlimited number of reservation stations and reorder buffer entries.
- An instruction waiting for data on the CDB can move to EX in the cycle after the CDB broadcast.
- An instruction waiting to write to the CDB holds its execution unit until it gets the CDB; i.e., it prevents other instructions needing the same functional unit from beginning execution.
- Assume that integer instructions also follow Tomasulo’s algorithm (analogous to the floating point instructions) so they can be issued out of order and the result from the integer functional unit is also broadcast on the CDB and forwarded to dependent instructions through the CDB.

Complete the blank entries in the following table using the above specifications. For each instruction, fill in the cycle numbers in each pipeline stage (CM stands for commit) and indicate where its source operands are read from (use RF for register file, ROB for reorder buffer, and CDB for common data bus). The entries for the first instruction and for the issue stage are filled in for you (Op means operand). Entries with dots should be ignored. Assume that at the start of the code snippet below, all registers have up-to-date values.
<table>
<thead>
<tr>
<th>IS</th>
<th>Op1 source</th>
<th>Op1</th>
<th>Op2 source</th>
<th>Op2</th>
<th>EX</th>
<th>WB</th>
<th>CM</th>
</tr>
</thead>
<tbody>
<tr>
<td>MUL  R2, R6, R12</td>
<td>1</td>
<td>R6</td>
<td>RF</td>
<td>R12</td>
<td>RF</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>ADD  R1, R1, R2</td>
<td>2</td>
<td>R1</td>
<td>RF</td>
<td>R2</td>
<td>CDB</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>DIV  R2, R2, R4</td>
<td>3</td>
<td>R2</td>
<td>CDB</td>
<td>R4</td>
<td>RF</td>
<td>6</td>
<td>13</td>
</tr>
<tr>
<td>ADDI R7, R4, #4</td>
<td>4</td>
<td>R4</td>
<td>RF</td>
<td>...</td>
<td>...</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>ADD  R5, R5, R7</td>
<td>5</td>
<td>R5</td>
<td>RF</td>
<td>R7</td>
<td>CDB</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>MUL  R6, R3, R5</td>
<td>6</td>
<td>R3</td>
<td>RF</td>
<td>R5</td>
<td>CDB</td>
<td>9</td>
<td>12</td>
</tr>
<tr>
<td>ADDI R9, R7, #8</td>
<td>7</td>
<td>R7</td>
<td>ROB</td>
<td>...</td>
<td>...</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>ADD  R5, R5, R6</td>
<td>8</td>
<td>R5</td>
<td>ROB</td>
<td>R6</td>
<td>CDB</td>
<td>13</td>
<td>14</td>
</tr>
</tbody>
</table>

**Grading:** 1/2 point for each entry (32 total entries). No deduction for entries which are correct given earlier entries, but repeat errors will be penalized.
Problem 3 [18 points]

Consider the following code fragment:

```
Loop:  LD.D  F1, 0(R1)
       LD.D  F2, 0(R2)
       MUL.D F3, F1, F10
       ADD.D F4, F3, F2
       SD.D  F4, 0(R1)
       DADDUI R1, R1, #8
       DADDUI R2, R2, #8
       BNE   R1, R3, Loop
```

Consider a pipeline with the following latencies: 3 cycles between an FP multiply and its consumer, 1 cycle between an FP add and its consumer, and 0 cycles between all other pairs. Thus, there should be three stall cycles between the multiply and addition in the above code for correct operation. Assume that all functional units are pipelined.

Unroll the above loop 4 times and write the resulting code to the left of the table on the next page (the above loop is repeated on the next page for your convenience). You have access to temporary registers T0…T63. Assume that the total number of iterations for the original loop is a multiple of 4. Schedule the unrolled loop for a VLIW machine where each VLIW instruction can contain one memory reference, one FP operation, and one integer operation. Write the scheduled instructions in the table on the next page to minimize the number of stalls. You may use L for L.D, M for MUL.D, etc.
Loop: L.D F1, 0(R1)
L.D F2, 0(R2)
MUL.D F3, F1, F10
ADD.D F4, F3, F2
SD.D F4, 0(R1)
DADDUI R1, R1, #8
DADDUI R2, R2, #8
BNE R1, R3, Loop

<table>
<thead>
<tr>
<th>Mem</th>
<th>FP ALU</th>
<th>Integer ALU</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD.D F1, 0(R1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LD.D T1, 8(R1)</td>
<td>MUL.D F3, F1, F10</td>
<td></td>
</tr>
<tr>
<td>LD.D T5, 16(R1)</td>
<td>MUL.D T3, T1, F10</td>
<td></td>
</tr>
<tr>
<td>LD.D T9, 24(R1)</td>
<td>MUL.D T7, T5, F10</td>
<td></td>
</tr>
<tr>
<td>LD.D F2, 0(R2)</td>
<td>MUL.D T11, T9, F10</td>
<td>DADDUI R1, R1, #32</td>
</tr>
<tr>
<td>LD.D T2, 8(R2)</td>
<td>ADD.D F4, F3, F2</td>
<td></td>
</tr>
<tr>
<td>LD.D T6, 16(R2)</td>
<td>ADD.D T4, T3, T2</td>
<td></td>
</tr>
<tr>
<td>LD.D T10, 24(R2)</td>
<td>ADD.D T8, T7, T6</td>
<td></td>
</tr>
<tr>
<td>SD.D F4, -32(R1)</td>
<td>ADD.D T12, T11, T10</td>
<td>DADDUI R2, R2, #32</td>
</tr>
<tr>
<td>SD.D T1, -24(R1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SD.D T5, -16(R1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SD.D T9, -8(R1)</td>
<td>BNE R1, R3, Loop</td>
<td></td>
</tr>
</tbody>
</table>

DADDUI R1, R1, #32
DADDUI R2, R2, #32
BNE R1, R3, Loop
Grading: 9 points for code on left, 9 points for table. Minimum score is 0 for both.

Deductions (only for first error, no cascading errors):

- 2 points for badly ordered loads.
- 2 points for inconsistent load/store addresses.
- 2 points for badly ordered MULs or ADDs.
- 2 points for late stores.
- 2 points for late branches/integer operations.
- 2 points for missing or wrong-path operations.
- 2 points if instruction uses wrong functional unit.
- 2 points if R1/R2 are updated every iteration.
- 2 points for inconsistent DADDUI offsets.
- 2 points for violating a RAW dependency.
- 1 point if temporary registers are out of bounds (not from T0-T63).
- 1 point if branch delay slot used but not stated in assumptions.
- No deduction for not using temporary registers.

If you did not write the unrolled code, we immediately deducted ½ points. Then we took your VLIW scheduled code and graded it according to the above deductions.
Problem 4 [9 points]

Consider the following piece of code:

```
DADDI R1, R0, #100

L1:   DADDI R1, R1, #-1
     BEQZ R1, END -- Branch 1
     DADDI R12, R0, #2

L2:   DADDI R12, R12, #-1
     BNEZ R12, L2 -- Branch 2

     J     L1

END: ...
```

Assume R0 stores 0. Branch 1 is executed 100 times and branch 2 is executed a total of 198 times. For each branch, how many correct predictions will occur if we use the following prediction schemes (as discussed in class)? Assume separate prediction bits for each branch (i.e., no aliasing). Assume at the beginning of execution, the last branch was not taken. Please explain your answers.

Part A [3 points]

1-bit predictor initialized to T (taken).

Solution:

Branch 1: N, N, N, ..., N, T is predicted correctly every time except the first and last. Branch 2 alternates T, N, T, N, ... and thus is predicted incorrectly every time except the first.

Branch 1: 98 correct predictions, Branch 2: 1 correct prediction.

Grading: ½ point for each number and 1 point for each explanation.

Part B [3 points]

2-bit saturating predictor initialized to 10 (taken).

Solution:

Branch 1 is predicted correctly every time except the first and last. Branch 2’s predictor alternates between 10 and 11, and thus predicts correctly every other time.


Grading: ½ point for each number and 1 point for each explanation.
Part C [3 points]

(1, 1) global correlating predictor, initialized to T/T.

Solution:

When Branch 1 is executed, the last branch is always not taken, so performance is the same as with a 1-bit predictor. Branch 2 is always taken when the last branch was not taken and not taken when the last branch was taken. Thus, there is one incorrect prediction - the first time it is not taken.

Branch 1: 98 correction predictions, Branch 2: 197 correct predictions.

Grading: ½ point for each number and 1 point for each explanation.
**Problem 5 [7 points]**

Consider the following code fragment:

*Loop:*  
LD.D F2, 0(R1)  
MUL.D F4, F6, F2  
ADD.D F4, F4, F8  
SD.D F4, 0(R1)  
DADDUI R1, R1, #8  
BNE R1, R3, Loop

Assume the loop consists of a very large number of iterations. Consider a pipeline with the following latencies:

- 2 cycles between a load and a dependent ALU instruction.
- 4 cycles between two dependent FP ALU instructions.
- 3 cycles between an FP ALU and a dependent store instruction.
- 0 cycles between all other pairs.

Provide the steady-state code for a software pipelined version of the above loop. Indicate the number of stalls incurred by your code. For full credit, your code should incur the minimum number of stalls using the minimum number of static instructions. You do not have to show the start-up or finish-up code (i.e., prolog or epilog).

**Solution:**

(1) SD.D F10, -24(R1)  // i-3  
(2) ADD.D F10, F4, F8  // i-2  
(3) MUL.D F4, F6, F2  // i-1  
(4) LD.D F2, 0(R1)  // i  
(5) DADDUI R1, R1, #8  
(6) BNE R1, R3, Loop

**Grading:**

2 points for correct offsets of (1) and (4).

1 point each for scheduling (1), (2), (3), and (4) across different iterations.

½ point for ordering (5) and (6) such that there are no stalls. ½ point for stating that there are 0 stalls.
Alternate solution:

(1) ADD.D F10, F4, F8 // i-2
(2) MUL.D F4, F6, F2 // i-1
(3) LD.D F2, 0(R1) // i
(4) DADDUI R1, R1, #8
(5) SD.D F10, -24(R1) // i-2
(6) BNE R1, R3, Loop

Grading:

2 points for correct offsets of (3) and (5).
1 point each for scheduling (1), (2), (3), and (5) across different iterations (1 and 5 in the same one).
½ point for ordering (4) and (6) such that there are no stalls. ½ point for stating that there are 0 stalls.
Problem 6 [8 points]

In this problem, we try to understand the implications of the Reorder Buffer (ROB) size on performance. Consider a processor implementing the ROB scheme described in class. Recall each instruction goes through issue (IS), writeback (WB), execute (EX), and commit (CM).

Assume IS, WB, and CM each take one cycle (once all the conditions for these stages are met), as discussed in class.

Assume our machine can fetch 4 instructions each cycle and can commit 4 instructions each cycle.

Assume a branch misprediction is handled when the branch instruction reaches the head of the ROB. It involves flushing that ROB entry and all entries following that entry. Issue is started at the correct successor of the branch from the next cycle.

For now, assume there are no memory accesses (this will change in part C).

Part A [2 points]

Suppose we have a perfect branch predictor and there is no data dependency between instructions. We have infinite execution units of each type and infinite reservation stations. All instructions take one cycle in the execute stage. What is the maximum achievable IPC? What is the minimum ROB size required to guarantee that IPC?

Solution:

Each instruction holds its ROB entry for 4 cycles (Issue, Execute, Write-back, and Commit). Therefore, we must have a minimum ROB size of 16 to avoid any stalls. In that case, the throughput would be 4 instructions per cycle.

Grading: 1 point for correct IPC. 1 point for correct ROB size.
Part B [2 points]

Suppose different FUs have different latencies in the EX stage, as given by the following table. Everything else is the same as in the previous part. What is the minimum size of the ROB required now to avoid issue stalls due to a full ROB?

<table>
<thead>
<tr>
<th>Functional Unit</th>
<th>Cycles in EX</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer</td>
<td>1</td>
</tr>
<tr>
<td>FP Adder</td>
<td>5</td>
</tr>
<tr>
<td>FP Multiplier</td>
<td>10</td>
</tr>
</tbody>
</table>

**Solution:**

If we issue an instruction to the FP multiplier, it would occupy its ROB entry for 13 cycles, and would also block the instructions following it from committing. During that period, we could have issued 52 instructions in all. Thus we need a minimum ROB size of 52 to avoid stalls. In that case we would get a throughput of 4 instructions per cycle.

**Grading:** 1 point for realizing that FP multiplier is the bottleneck. 1 point for correct ROB size.

Part C [2 points]

In addition to the latencies above, now every 10th instruction is a load instruction. Assume the address calculation and cache/memory access parts of the load both happen in the EX stage. The hit rate in the data cache is 95% and the misses are uniformly spaced through the instruction stream. A hit takes 1 cycle in the EX stage. However, upon a miss, the data has to be fetched from the memory and this results in 100 cycles in the EX stage. What is the ROB size required now to avoid any issue stalls?

**Solution:**

A load instruction that misses in the cache would occupy its ROB entry for 1+100+1+1 = 103 cycles. During this time, we could have issued 412 instructions. This is the size of the ROB required to continuously issue 4 instructions each cycle.

**Grading:** 1 point for calculating the correct latency of loads. 1 point for correct ROB size.
Part D [2 points]

Now additionally assume we don’t have perfect branch prediction anymore. Instead, we have a predictor with an accuracy of 95%. Assume every 8th instruction is a branch and the mispredictions are uniformly spaced through the instruction stream.

After a misprediction, how many instructions are issued before the next misprediction is encountered? In light of this result, do you think we need a ROB of the size you derived in Part C? Why/why not? If not, what do you think would be a good ROB size to have?

Solution:

We would issue $8 \times 20 = 160$ instructions before a mispredict is encountered. Given this result, it doesn’t make sense to have a ROB of size 412, because once we have a cache miss, soon after we would run into a branch mispredict, and all the work done after that would be wasted anyway. A reasonable ROB size would be 160.

Grading: 1 point for stating that a smaller ROB is sufficient. 1 point for correct ROB size.