CS 433 Midterm Exam – Oct 15, 2018
Professor Sarita Adve

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>NetID</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Undergraduate</td>
</tr>
<tr>
<td></td>
<td></td>
<td>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 11 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>11</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>7 (for graduates only)</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>42 for undergraduates 49 for graduates</td>
</tr>
</tbody>
</table>
Problem 1 [4 points]
Assume 90% of a program can be executed in parallel.

Part A [2 points]
What speedup is required on the parallel section to achieve an overall speedup of 4X for the full program?

Part B [2 points]
What is the maximum possible speedup achievable on the above program through parallelization?
Problem 2 [11 points]

This problem concerns Tomasulo’s algorithm with dual issue and hardware speculation.

Consider the following architecture.

<table>
<thead>
<tr>
<th>Functional Unit Type</th>
<th>Cycles in EX</th>
<th>Number of Functional Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>FP Add/Subtract</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>FP Multiply</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>FP Divide</td>
<td>16</td>
<td>1</td>
</tr>
</tbody>
</table>

- Functional units are not pipelined.
- Memory accesses use the integer functional unit to perform effective address calculation during a single cycle EX stage. Stores access memory during the commit stage while loads access memory during EX. Stores do not need the CDB or the WB stage.
- Assume that integer instructions also follow Tomasulo’s algorithm (analogous to the floating point instructions). So the result from the integer functional unit is also broadcast on the CDB and forwarded to dependent instructions through the CDB. An exception is that addresses generated for memory accesses are not broadcast on the CDB.
- 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.
- An instruction waiting for data on the CDB can move to EX in the cycle after the CDB broadcast.
- Only one instruction can write to the CDB in one clock cycle.
- If there is a conflict for a functional unit or the CDB, the oldest (by program order) instruction gets access.
- Branches use the integer unit with one cycle in EX. They do not need the CDB or the WB stage.
- Assume a perfect branch predictor. That is, an instruction can issue before a prior branch has completed execution, but it cannot issue in the same cycle as the branch issues (the earliest it can issue is the cycle immediately after the branch issues). Any other instruction pair can issue in the same cycle.
- There are unlimited reorder buffer entries and reservation stations.
- Two instructions can commit per cycle.

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). Then indicate where its source operands are read from (use RF for register file, ROB for reorder buffer, and CDB
for common data bus) – the source of the first operand should appear first. In the last two columns, indicate if the instruction is stalled for structural or data hazards. If it is stalled, indicate which instructions cause those stalls – give all such instructions even if the dependence due to one of them is resolved before the other. Some entries are filled for you. You don’t have to fill entries with “---”

<table>
<thead>
<tr>
<th>Instruction</th>
<th>IS</th>
<th>EX</th>
<th>WB</th>
<th>CM</th>
<th>Source of operand1, source of operand2</th>
<th>Stalls due to structural hazards</th>
<th>Stalls due to data hazards</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 L.D F0, 0(R1)</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>2 DIV.D F2, F0, F6</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>CDB, RF</td>
<td>No</td>
<td>Yes: 1</td>
</tr>
<tr>
<td>3 L. D F6, 8(R1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>4 ADD.D F4, F2, F6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>5 MUL.D F8, F6, F4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>6 S.D F4, 16(R1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>7 DADDUI R1, R1, -32</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>8 BNEZ R1, target</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>9 ADD.D F10, F2, F6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
</tbody>
</table>

--- --- --- --- --- --- ---
Problem 3 [9 points]

Consider a loop that is entered multiple 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 (don’t worry about gshare or tagged hybrid predictors). 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 [3 points]
(B) Branch 2 [3 points]

(C) Branch 5 – for this part and this part alone, do not consider correlating predictors [3 points]
Problem 4: [14 points]

Assume an in-order single-issue pipeline with multiple functional units. IF, ID, and WB take 1 cycle as usual, branches are resolved in ID, and all needed forwarding paths are provided. There is a fully pipelined, 3 cycle FP add unit (i.e., 3 cycles elapse from the time an FP add enters its functional unit to the time its result is available for a later instruction). There is a fully pipelined 5 cycle FP multiplier, and a 1 cycle integer ALU for all non-branch integer instructions. Loads and stores execute for 1 cycle in the integer ALU and spend 12 cycles in MEM. Consider the following code on the above machine:

```
loop:
    LD.D     F4, 0(R1)
    LD       R2, 4096(R1)
    LD.D     F10, 0(R2)
    MUL.D    F6, F4, F10
    ADD.D    F8, F8, F6
    DADDI    R1, R1, #8
    BNEQ     R1, R3, loop  // branch if R1 does not equal R3
```

Part (A) [4 points]
Give one pair of instructions with a RAW dependence that requires the most separation between them to avoid stalling? What is the minimum number of instructions that would need to be between this pair in program order to avoid the stall? Explain your answer.

Part (B) [2 points]
Consider the two dependent instructions from Part (A). Just for this part, ignore all other dependences in the loop body; i.e., assume that dependences for all other instructions magically do not require any stalls. Now consider unrolling the loop to remove any stalls between the instructions chosen in Part (A). What is the minimum number of original iterations that the new unrolled loop would need to have to eliminate these stalls? Remember to explain your answer.
Part (C) [8 points]
Now consider the original loop with all the original dependences and stalls as in Part (A). Software pipeline the loop to run in as few cycles as possible, using the start-up code given below. You only have to produce the code for the steady state. Do not write or consider the performance of the clean-up code. Do not modify the start-up code. Do not unroll the loop.

Startup code:
LD R2, 4096(R1) // iteration 1
LD.D F4, 0(R1) // iteration 1
LD.D F10, 0(R2) // iteration 1
LD R2, (8+4096)(R1) // iteration 2
DADDI R1, R1, #16

Your steady-state code here:
Problem 5 [4 Points]

Consider the following format for predicated MIPS instructions:

\[(pT) \text{ ADD } R1, R2, R3\]

where the ADD instruction is predicated on the predicate register pT. Assume a set of 1-bit predicate registers.

Assume compare instructions that set a pair of predicate registers to complementary values:

\[\text{CMP.NE } pT, pF = R8, R0\]

The above compare sets the 1-bit predicate registers, pT and pF, based on the "not equal" (NE) comparison relation as follows:

\[pT = (R8 \neq R0)\]
\[pF = !(R8 \neq R0)\]

So pT is true if R8 is not equal to R0, and pF is the complement of pT.

For the following problem, you can assume the availability of any comparison relation with two operands; e.g., .LE for less than or equal to and .GT for greater than.

Using the predicated instructions described above, write the three basic blocks of the following code fragment as a single basic block; i.e., eliminate all branches using predicated instructions.

\[
\begin{align*}
\text{SUB} & \quad R1, R13, R14 \\
\text{BLT} & \quad R1, R4, L1 \quad //\text{branch if } R1 < R4 \\
\text{ADDI} & \quad R2, R2, #1 \\
\text{SW} & \quad R2, 0(R7) \\
\text{J} & \quad L2 \\
\text{L1:} & \quad \text{DIV.D } F0, F0, F2 \\
& \quad \text{ADD.D } F0, F4, F2 \\
& \quad \text{S.D } F0, 0(R8) \\
\text{L2:} & \quad ...
\end{align*}
\]
Problem 6 [7 points]

Consider the following loop:

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 [2 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 and has infinite 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.
Part B [2 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?

Part C [3 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.