CS433 Homework 2
(Chapter 3)

Assigned on 9/19/2017
Due in class on 10/5/2017

Instructions:

1. Please write your name and NetID clearly on the first page.
2. Refer to the course fact sheet for policies on collaboration.
3. Due **IN CLASS** on 10/5/2017.

Problem 1 [40 points]

This problem concerns Tomasulo's algorithm. Consider the following architecture specification:

Functional Unit Type	Cycles in EX	Number of Functional Units
Integer	1	1
FP Adder	5	1
FP Multiplier	8	1
FP Divider	15	1

1) Assume that you have unlimited reservation stations.
2) IS and WB take one cycle each.
3) One Instruction is issued per cycle.
4) Memory accesses use the integer functional unit to perform effective address calculation during the EX stage.
5) For stores, memory is accessed during the EX stage (Tomasulo's algorithm without speculation) or commit stage (Tomasulo's algorithm with speculation).
6) All loads access memory during the EX stage.
7) Loads and Stores stay in EX for one cycle.
8) Functional units are not pipelined.
9) 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.
10) An instruction waiting for data on the CDB can move to its EX stage in the cycle after the CDB broadcast.
11) Only one instruction can write to the CDB in one clock cycle.
12) Branches and stores do not need the CDB.
13) Whenever 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.

14) 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).
15) Assume that the BNEQZ occupies the integer functional unit for its computation and spends one cycle in EX.

Part A [10 points]

Fill in the table below with the cycle number where each instruction occupies the given stage. If a stall occurs, describe the reason for the stall. The reason should include the type of hazard; registers, functional units, etc. that caused the dependence; and the instruction on which the given instruction is dependent (use the IS stage cycle number to identify the instruction). The first two entries are given.

Instruction	IS	EX	WB	Reason for stalls
L.D F0, 0(R1)	1	2	3	
ADD.D F2, F0, F4	2	4-8	9	RAW on F0 (from 1)
MUL.D F4, F2, F6	3	17-24	25	RAW on F2 (from 2); Structural hazard for FP Multiplier (from 7)
ADD.D F6, F8, F10	4	9-13	14	Structural hazard for FP Adder (from 2)
DADDI R1, R1, #8	5	6	7	
L.D F1, 0(R2)	6	7	8	
MUL.D F1, F1, F8	7	9-16	17	RAW on F1 (from 6)
ADD.D F6, F3, F5	8	14-18	19	Structural hazard for FP Adder (from 4)
DADDI R2, R2, #8	9	10	11	

Part B [2 points]

Would pipelining any of the functional units reduce the total execution time of the above code segment? If so, find an example from the code and explain. Otherwise, explain why.

[Answer] Yes. Stalls due to structural hazards can be avoided or shortened. For example, by pipelining FP Multiplier, the first MUL.D instruction can start execution at the cycle 10.

Part C [2 points]

Would adding another multiplier functional unit be more advantageous than pipelining the multiplier for the above code segment? Explain your answer.

[Answer] No. As two MUL.D instructions get ready for the EX stage (i.e., resolve data dependences) at different cycles, an additional multiplier functional unit does not provide extra benefit over pipelining the existing one.

Part D [18 points]

For this part, assume ***hardware speculation*** and ***dual-issue*** added to the Tomasulo pipeline you used for the part A. That is, assume that an instruction can issue even before the branch has completed (or started) its execution (as with perfect branch and target prediction). However, assume that an instruction after a branch cannot issue in the same cycle as the branch; the earliest it can issue is in the cycle immediately after the branch (to give time to access the branch history table and/or buffer). Any other pair of instructions can issue in the same cycle. Assume that a store calculates its target address in EX and performs its memory access during the Commit stage. Recall that stores do not write back.

Additionally, assume that your reorder buffer has ***12 entries*** (at the beginning of execution, the ROB is empty). Furthermore, ***two instructions can commit each cycle***.

Fill in the cycle numbers in each pipeline stage for each instruction in the first two iterations of the loop represented below, assuming the branch is always taken. The entries for the first two instructions of the first iteration are filled in for you. CM stands for the commit stage.

Instruction	IS	EX	WB	CM	Reason for stalls
Iteration 1					
LP: L.D F0, 0(R1)	1	2	3	4	
ADD.D F0, F0, F6	1	4-8	9	10	RAW on F0 (from 1)
DIV.D F2, F2, F0	2	20-34	35	36	RAW on F0 (from 1); Structural hazard for FP Divider (from 3)
L.D F0, 8(R1)	2	3	4	36	
DIV.D F4, F0, F8	3	5-19	20	37	RAW on F0 (from 2)
S.D F4, 16(R1)	3	4		37	
DADDI R1, R1, #-24	4	5	6	38	
BNEZ R1, LP	4	7		38	RAW on R1 (from 4)
Iteration 2					
LP: L.D F0, 0(R1)	5	8	10	39	RAW on R1 (from 4); Structural hazard for Integer (from 4); Structural hazard for CDB (from 1)
ADD.D F0, F0, F6	5	11-15	16	39	RAW on F0 (from 5)
DIV.D F2, F2, F0	6	50-64	65	66	RAW on F0 (from 5); Structural hazard for FP Divider (from 7)
L.D F0, 8(R1)	6	9	11	66	Structural hazard for Integer (from 5); Structural hazard for CDB (from 5)
DIV.D F4, F0, F8	7	35-49	50	67	Structural hazard for FP Divider (from 2);
S.D F4, 16(R1)	11	12		67	ROB full (from 1)
DADDI R1, R1, #-24	37	38	39	68	ROB full (from 2)
BNEZ R1, LP	37	40		68	RAW on R1 (from 37)

Part E [6 Points]

For the code in the part D, which of the following optimizations will cause a performance improvement of at least one cycle per loop iteration: (1) triple issue, (2) three instruction commits per cycle, or (3) reorder buffer of size 14? Explain why.

[Answer]

(1) Triple issue causes the fifth instruction to be fetched at the cycle 2, but EX still has to start in the cycle 5 due to the data dependence. Since DIV unit's latency causes the same bottleneck as before, there will not be any overall improvement.

(2) Triple commit causes the first loop to end in 37 cycles and the second loop to finish by the end of the 67th cycle, hence giving an improvement of 1 cycle per iteration.

(3) Increased reorder buffer again causes more instructions to be fetched, but they cannot commit until DIV instructions complete and commit. Hence there will not be any improvement.

Part F [2 Points]

For the code in the part D, assume a floating point division by 0 incurs an exception. In which clock cycle will the system invoke a jump to the interrupt service routine if F8 used in the fifth instruction has the value 0? Assume that the exception is identified as soon as EX begins and the instruction with the exception gives up the execution unit in the same cycle (i.e., it is available for another instruction in the next cycle). Explain your answer.

[Answer]

The interrupt will be identified at the cycle 5 and the FP Divider will be ready for other instructions at the cycle 6. Thus, the third instruction runs from the cycle 6 until the cycle 20. Then, the third and fourth instruction will commit at the cycle 22. Finally, the fifth instruction will commit at the cycle 23, and the system invokes a jump to the ISR at the cycle 23.

Problem 2 [10 points]

Consider the following MIPS code. The register R0 is always 0.

```
        ADD.D R1, R0, R0

L1:     ADD.D R2, R0, R0

L2:     DADDI R2, R2, #1
        DSUBI R3, R2, #3
        BNEQZ R3, L2          <-- Branch 1

        DADDI R1, R1, #1
        DSUBI R4, R1, #4
        BNEQZ R4, L1          <-- Branch 2
```

Each table below refers to each branch. For instance, the branch 1 will be executed 12 times, and those 12 times should be recorded in the table for the branch 1. Similarly, the branch 2 is executed 4 times.

Part A [4 points]

Assume that 1-bit branch predictors are used. When the processor starts to execute the above code, both predictors contain value N (Not taken). What is the number of correct predictions? Fill the following tables to record the prediction and action of each branch.

Step	Branch 1 Prediction	Actual Branch 1 Outcome
1	N	T
2	T	T
3	T	N
4	N	T
5	T	T
6	T	N
7	N	T
8	T	T
9	T	N
10	N	T
11	T	T
12	T	N

Step	Branch 2 Prediction	Actual Branch 2 Outcome
1	N	T
2	T	T
3	T	T
4	T	N

Part B [6 Points]

Now assume that 2-bit saturation counters are used. When the processor starts to execute the above code, both counters contain value 0. What is the number of correct predictions? Fill the following tables to record the prediction and action of each branch.

Step	Counter Values	Branch 1 Prediction	Actual Branch 1 Outcome
1	00	N	T
2	01	N	T
3	11	T	N
4	10	T	T
5	11	T	T
6	11	T	N
7	10	T	T
8	11	T	T
9	11	T	N
10	10	T	T
11	11	T	T
12	11	T	N

Step	Counter Values	Branch 2 PredictionActual Branch 2 Outcome	
1	00	N	T
2	01	N	T
3	11	T	T
4	11	T	N

Problem 3 [10 Points]

Suppose we have a deeply pipelined processor, for which we implement a branch target buffer (BTB) for conditional branches only. Assume the followings:

(1) 15% of the total instructions are conditional branches.
(2) the BTB hit rate is 90%.
(3) the BTB miss penalty is always 3 cycles.
(4) even if hit on the BTB, if the prediction was incorrect, there is a 4-cycle penalty.
(5) the branch prediction accuracy is 90%.
(6) the base CPI without branch stall is 1.

Part (A) [5 points]

How much faster is this processor compared to a processor that does not have a branch predictor and has a fixed two-cycle branch penalty?

Without a branch prediction and a fixed two-cycle branch penalty,
CPI_NO_PREDICT = 1 + 0.15 * 2 = 1.3
With a branch prediction and a BTB,
CPI_BTB = 1 + 0.15 * 0.1 * 3 + 0.15 * 0.9 * 0.1 * 4 = 1 + 0.045 + 0.054 = 1.099
Speedup = CPI_NO_PREDICT / CPI_BTB = 1.3 / 1.099 = 1.18

[Answer] 1.18

Part (B) [5 points]

For this part, let's assume that the BTB is extended to store one target instruction and used to convert an unconditional branch into the target instruction at the IF stage (i.e., branch folding). Also, assume that 10% of the total instructions are unconditional branches and an unconditional branch originally takes 1 cycle to execute without the BTB. Calculate the CPI with the extended BTB.

With branch folding, if an unconditional branch hits in the BTB, it will be converted into the target instruction and it will not be counted toward CPI.

CPI_BTB_EXTENDED = CPI_BTB – 0.1 * 0.9 = 1.18 – 0.09 = 1.09

[Answer] 1.09

Problem 4 (*for graduate students only*) [10 points]

This problem concerns the implications of the reorder buffer size on performance. Consider a processor implementing Tomasulo's algorithm with reservation stations and the reorder buffer scheme as described in the lecture notes. Assume infinite processor resources unless stated otherwise (e.g., infinite execution units and infinite reservation stations). Assume a perfect branch predictor and assume there are no data dependence in the instruction stream we are considering. Assume the maximum instruction fetch rate is 12 instructions per cycle. The other stages in the pipeline have no constraints (e.g., the processor can decode an unbounded number of instructions per cycle).

Part (A) [2 points]

Suppose all instructions take one cycle to execute and the processor has an infinite reorder buffer. What is the average instructions-per-cycle rate (IPC) of this processor? Explain why.

[Answer]

The average IPC would be 12 since it is limited only by the fetch rate. There is no reason for any stall since there are no data dependence or branch misprediction and we have infinite resources.

Part (B) [3 points]

Consider the system in the part A except that now every 48th instruction is a load that misses in the cache and the miss latency is 500 cycles. What is the average IPC of this processor? Explain why.

[Answer]

The average IPC would again be 12. The ROB would mask out the latencies associated with missing load instructions, and would allow us to keep fetching and issuing 12 instructions each cycle. The misses would introduce a lag of up to 500 cycles between the fetch and commit, but the average throughput would still be 12 instructions each cycle.

Part (C) [5 points]

Consider the system in the part B except that now the reorder buffer size is 48 entries. What is the average IPC for this processor? If the IPC is less than 12, then what is the smallest reorder buffer size for which the IPC will be 12 again (assume the reorder buffer size can only be a multiple of 12).

[Answer]

We can no longer get an IPC of 12 since the limited ROB size will cause stalls. Suppose that at cycle 1, we issue a load that misses. We would keep fetching and issuing instructions until the ROB becomes full, so we would issue 47 more instructions and then stall, until cycle 500, when the instruction at the head of the ROB completes and is ready to commit (along with the other 47 instructions). At that point, we would issue the next missing load, and this cycle would repeat. Thus, every 500 cycles, we can commit 48 instructions, and the IPC will be 48/500. To obtain an average IPC of 12, we need to be able to overlap the execution of 12 instructions per cycle on average during the 500 cycles for which the load is stalled. These instructions cannot commit until the load commits. This requires an ROB size of 12*500=6000 instructions.