Computer Architecture and Engineering

CS152 Quiz #3

March 30, 2010

Professor Krste Asanovic

Name:_____ANSWER KEY_____

This is a closed book, closed notes exam.

80 Minutes

9 Pages

Notes:

- **Not all questions are of equal difficulty, so look over the entire exam and budget your time carefully.**
- **Please carefully state any assumptions you make.**
- **Please write your name on every page in the quiz.**
- **You must not discuss a quiz's contents with students who have not yet taken the quiz. If you have inadvertently been exposed to the quiz prior to taking it, you must tell the instructor or TA.**
- **You will get no credit for selecting multiple choice answers without giving explanations if the instructions ask you to explain your choice.**

Writing name on each sheet	________	**1 Point**
Question 1	________	**27 Points**
Question 2	________	**26 Points**
Question 3	________	**26 Points**
TOTAL	________	**80 Points**

NAME:________________________

Problem Q4.1: Scheduling 27 points

In this problem, we examine the execution of a code segment on in-order and out-of-order processors. The code we consider scales a vector of floating-point numbers by a constant.

```
I1   loop: LD.D    F0, 0(R1)
I2         MUL.D   F0, F2, F0
I3         ST.D    F0, 0(R1)
I4         ADDI    R1, R1, 8
I5         BNE     R1, R2, loop
```

Problem Q4.1.A 8 points

First, we consider the fully-bypassed, single-issue, in-order pipeline in Figure Q4.1-1. Load results are available at the end of the X3 stage, and floating-point multiply results are available at the end of the X4 stage. Stores don't need their data until the X2 stage. Assume perfect branch target prediction (i.e. there are no bubbles due to branches.)

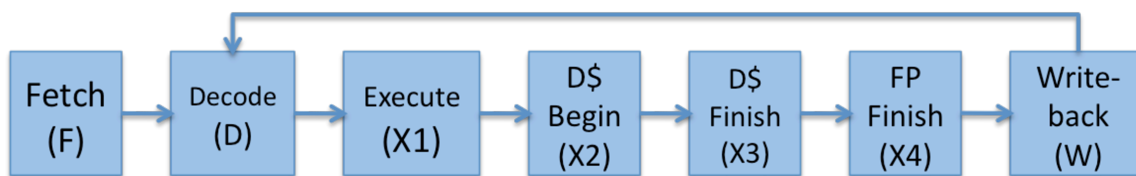

Figure Q4.1-1. In-order pipeline. Full bypassing is not shown.

Fill in Table Q4.1-1 on page 4 to indicate how the given code will execute for *two* loop iterations. In each cell, write the name of the stage that the instruction is currently in. The first two rows have been completed for you.

Problem Q4.1.B 4 points

Averaged over a large number of iterations, what is the throughput of the loop executing on the processor in Problem Q4.1.A, measured in cycles per iteration?

The writeback of I5 for the second iteration is 9 cycles after the writeback of I5 for the first iteration. This pattern will continue, so we'll average 9 cycles per iteration.

________9________ Cycles per Iteration

NAME:__________________________

Problem Q4.1.C **7 points**

Next, we consider an idealized single-issue, out-of-order pipeline *without* register renaming, shown in Figure Q4.1-2. Instructions can issue out-of-order once no RAW, WAW, or WAR hazards exist. The issue stage can hold an unbounded number of waiting instructions. As with Q4.1.A, the pipeline is fully bypassed, load results are available at the end of X3, multiply results are available at the end of X4, store data is consumed in the X2 stage, and branch target prediction is perfect. Writebacks can occur out of order.

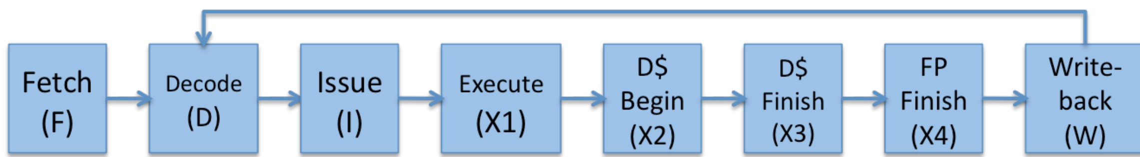

Figure Q4.1-2. Out-of-order pipeline. Full bypassing is not shown.

Fill in Table Q4.1-2 on page 4 to indicate how the given code will execute for *two* loop iterations. In each cell, write the name of the stage that the instruction is currently in. The first three rows have been completed for you.

Problem Q4.1.D **4 points**

Averaged over a large number of iterations, what is the throughput of the loop executing on the processor in Problem Q4.1.D, measured in cycles per iteration?

Without renaming, the bottleneck will be the latency from the issue of the load to the issue of the store (inclusive), which is 7 cycles.

________7________ Cycles per Iteration

Problem Q4.1.E **4 points**

Finally, consider a pipeline that is identical to the one in Q4.1.C, except that it now has register renaming with an unbounded number of physical registers. Averaged over a large number of iterations, what is the throughput of the loop executing on this processor, measured in cycles per iteration? (Hint: it may not be necessary to manually schedule the code to determine the answer to this question.)

Now, only RAW hazards matter, so we will get multiple iterations of the loop in-flight at a time, allowing us to sustain a CPI of 1.

_____5_________ Cycles per Iteration

Inst \ Cycle	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
$I_{1,\text{ iteration 1}}$	F	D	X1	X2	X3	X4	W																							
$I_{2,\text{ iteration 1}}$		F	D	D	D	X1	X2	X3	X4	W																				
$I_{3,\text{ iteration 1}}$			F	F	F	D	D	D	X1	X2	X3	X4	W																	
$I_{4,\text{ iteration 1}}$						F	F	F	D	X1	X2	X3	X4	W																
$I_{5,\text{ iteration 1}}$									F	D	X1	X2	X3	X4	W															
$I_{1,\text{ iteration 2}}$										F	D	X1	X2	X3	X4	W														
$I_{2,\text{ iteration 2}}$											F	D	D	D	X1	X2	X3	X4	W											
$I_{3,\text{ iteration 2}}$												F	F	F	D	D	D	X1	X2	X3	X4	W								
$I_{4,\text{ iteration 2}}$															F	F	F	D	X1	X2	X3	X4	W							
$I_{5,\text{ iteration 2}}$																		F	D	X1	X2	X3	X4	W						

Table Q4.1-1. Code schedule for the in-order processor in Q4.1.A.

Inst \ Cycle	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
$I_{1,\text{ iteration 1}}$	F	D	I	X1	X2	X3	X4	W																						
$I_{2,\text{ iteration 1}}$		F	D	I	I	I	X1	X2	X3	X4	W																			
$I_{3,\text{ iteration 1}}$			F	D	I	I	I	I	I	X1	X2	X3	X4	W																
$I_{4,\text{ iteration 1}}$				F	D	I	I	X1	X2	X3	X4	W																		
$I_{5,\text{ iteration 1}}$					F	D	I	I	X1	X2	X3	X4	W																	
$I_{1,\text{ iteration 2}}$						F	D	I	I	I	X1	X2	X3	X4	W															
$I_{2,\text{ iteration 2}}$							F	D	I	I	I	I	I	X1	X2	X3	X4	W												
$I_{3,\text{ iteration 2}}$								F	D	I	I	I	I	I	I	I	X1	X2	X3	X4	W									
$I_{4,\text{ iteration 2}}$									F	D	I	X1	X2	X3	X4	W														
$I_{5,\text{ iteration 2}}$										F	D	I	X1	X2	X3	X4	W													

Table Q4.1-2. Code schedule for the out-of-order processor in Q4.1.C.

Problem Q4.2: Branch Prediction in the Pipeline 26 points

Ben Bitdiddle is designing a 7-stage in-order pipeline and is concerned about the performance implications of branches. The baseline processor he is considering is similar to the classic 5-stage RISC pipeline, except instruction cache access and data cache access each take two stages, as shown in Figure Q4.2-1. Initially, the pipeline lacks any branch prediction mechanism.

All branches in Ben's ISA are simple enough that they can execute without an ALU. His ISA has no branch delay slots.

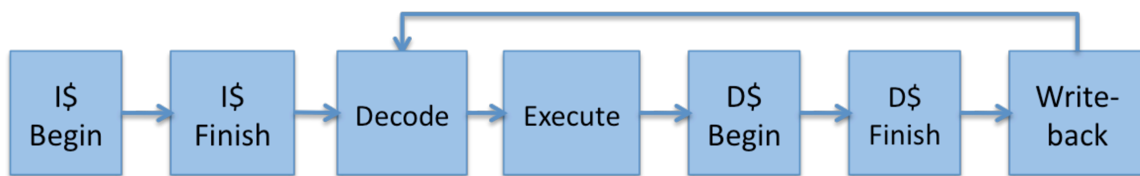

Figure Q4.2-1. Ben Bitdiddle's in-order pipeline.

Problem Q4.2.A 7 points

What is the earliest stage that branches can be resolved in Ben's pipeline? How many instructions are squashed on a taken branch? Assuming 1 in 6 instructions is a branch and 3/5 branches are taken, and assuming a base CPI of 1, what is the CPI of Ben's pipeline?

Simple branches can be resolved in decode, so two fetches are wasted on a taken branch. The CPI is 5/6 (non-branches) + 1/6*2/5 (untaken branches) + 3*1/6*3/5 (taken branches), or 6/5.

____Decode____ Branch Resolution Stage

____2____ # Instructions Squashed

____6/5____ CPI

NAME:________________________

Problem Q4.2.B — 6 points

Ben is considering adding a branch history table (BHT) to predict branch outcomes. In what pipeline stage would the BHT's prediction be acted upon? In what situations will the BHT improve performance?

The BHT is untagged, so its prediction shouldn't be acted upon until the instruction is known to be a branch (i.e. in the decode stage). Thus, the BHT usually doesn't help at all, unless a RAW hazard prevents it from resolving in the decode stage—namely, if a load precedes a dependent branch.

______Decode____ BHT Prediction Stage

Problem Q4.2.C — 7 points

Ben considers adding a single-cycle latency branch target buffer (BTB) as an alternative to the BHT. In what pipeline stage would the BTB produce a next-PC prediction? If the BTB predicts taken branches correctly 80% of the time and mistakenly predicts not-taken branches as taken 20% of the time, what is Ben's new CPI, assuming a base CPI of 1?

The BTB is indexed by the current PC, so its prediction can be used as soon as the PC is known, during the I$ Begin stage.

There is no bubble on taken branches now (assuming a correct prediction); mispredicted branches (either taken or not-taken) will incur a 2-cycle penalty. So the CPI is 5/6 + 1/6*0.8 (correctly-predicted branches) + 3*1/6*0.2 (mispredicted branches), or 16/15.

_____I$ Begin________ BTB Prediction Stage

_______16/15________ CPI

NAME:________________________

Problem Q4.2.D **6** points

Ben's original BTB design consisted of a next-PC prediction, a tag, and a valid bit per entry. The BTB's prediction was only used if the valid bit was set and the tag matched the current PC. Ben wishes to keep the BTB's behavior exactly the same, but save silicon area by omitting the valid bit. How can he accomplish this goal?

The important insight is that when we predict that a branch is not-taken, we're predicting that the next PC is PC+4. So, for predicted-not-taken branches (or non-branches), we can set the next-PC prediction to PC+4 instead of clearing the valid bit.

NAME:________________________

Problem Q4.3: Potpourri 26 points

Problem Q4.3.A 7 points

Describe how precise exceptions are maintained in out-of-order processors.

Exceptions are detected when an instruction executes out-of-order and saved in the ROB. Since instructions commit in-order from the ROB, exceptions can still be taken in program order by not actually taking an exception until the corresponding instruction is at the head of the ROB, about to commit.

Problem Q4.3.B 5 points

Consider an out-of-order processor with register renaming using a unified physical register file. A new physical register is allocated for each instruction's destination register in the decode stage, but since physical registers are a finite resource, they must be deallocated at some point in time. Carefully explain when it is safe to deallocate a physical register.

The physical register can be freed when the *next* writer of the same *architectural* register commits. At that point, it is guaranteed that no instructions remaining in the pipeline need to read the old physical register.

NAME:________________________

Problem Q4.3.C **7 points**

Suppose the following code executes on an ideal out-of-order processor with unlimited renaming registers and reordering resources, perfect memory disambiguation, perfect branch prediction, and infinite superscalar issue width. In the steady state, how many loop iterations will complete per cycle? Describe how the code could be restructured to improve its performance on this processor.

```
I1   loop: LD   R3, 0(R1)     # for(i = 0; i < N; i++)
I2         ST   R3, 0(R2)     #   A[i] = B[i];
I3         ADDI R1, R1, 4     # A and B don't overlap.
I4         ADDI R2, R2, 4     # Initially, R1 = A,
I5         BNE  R1, R4, loop  # R2 = B, and R4 = &A[N].
```

There is a RAW dependence between ADDI instructions in different loop iterations. This will limit the IPC of the ADDI instructions to 1, which limits the processor to completing one iteration per cycle. This limitation can be mitigated by unrolling the loop and folding the index calculation into the loads, reducing the number of RAW hazards per useful work.

For example, the following code will execute twice as fast:

```
ld    r3, 0(r1)
st    r3, 0(r2)
ld    r3, 4(r1)
st    r3, 4(r2)
addi  r1, r1, 8
addi  r2, r2, 8
bne   r1, r4, loop
```

________1________ Iterations per Cycle

NAME:_________________________

Problem Q4.3.D — 7 points

Consider again the processor from Q4.3.C. To simplify its design, we institute the requirement that loads are not reordered with respect to stores, and that stores are not reordered with respect to each other. Now, when this processor executes the code from Q4.3.C, how many loop iterations will complete per cycle in the steady state? Describe how the code could be restructured to improve performance on this processor.

The loads and stores will be serialized in the original code, so two cycles will be needed to complete each iteration. To speed things up, we need to unroll the loop and group all of the loads together (since they can freely execute in parallel). For example:

```
ld      r3, 0(r1)
ld      r5, 4(r1)
st      r3, 0(r2)
st      r5, 4(r2)
addi    r1, r1, 8
addi    r2, r2, 8
bne     r1, r4, loop
```

________1/2_______ Iterations per Cycle

END OF QUIZ