CS211 Computer Architecture II

Out-of-order Execution and Speculative Execution

Assigned 22/10/2022 Homework #2 *Due 23:59:59 06/11/2022*

https://toast-lab.sist.shanghaitech.edu.cn/courses/CS211@ShanghaiTech/Fall-2022/

The homework is intended to help you learn the material, and we encourage you to collaborate with other students and to ask questions in discussion sections and office hours to understand the problems. However, each student must turn in their own solution to the problems.

The homework also provides essential background material for the mid-term and final exams. It will be graded primarily on an effort basis, but if you do not work through it, you are unlikely to succeed on the mid-term and final exams! We will distribute solutions to homework assignment. Note that each homework assignment is due at its respective due time, and all assignments are to be submitted **in English**. However, late submissions will **NOT** be accepted, except for extreme circumstances and/or with prior arrangement.

Name: 俞政宏

ID: 2020533156

Problem 1: Out-of-order Scheduling (3.0 Points)

Your friend Li Hua is adding a floating-point unit to the basic five-stage pipeline. He has patterned the design after the classic IBM 360/91's floating-point unit. His FPU has one adder, one multiplier, and one load/store unit. The adder has a four-cycle latency and is fully pipelined. The multiplier has a ten-cycle latency and is fully pipelined. Assume that loads and stores take 1 cycle (plus one cycle for the write-back stage for loads) and that we have perfect branch prediction.

There are 4 floating-point registers, F0-F3. These are separate from the integer registers Rn (n>=1). There is a single write-back port to each register file. In the case of a write-back conflict, the older instruction writes back first. Floating-point instructions (and loads writing floating point registers) must spend one cycle in the write-back stage before their result can be used. Integer results are available for bypass the next cycle after issue.

Li Hua is now deciding whether to go with (a) in-order issue using a scoreboard and (b) out-of-order issue. He is using the following pseudocode that implements the operation Y = aX - Y for a vector of length 1000. Initially R1 contains the base address for X, R2 contains the base address for Y, and F0 contains a. You are to evaluate the performance of the three scheduling alternatives on this loop.

Instruction sequence number	Instruction	comment
I1	L.D F2, 0(R1)	Load X(i)
I2	L.D F3, 0(R2)	Load Y(i)
I3	MUL.D F1, F2, F0	Multiply a * X(i)
I4	SUB.D F3, F1, F3	Sub a * X(i) - Y(i)
I5	S.D F3, 0(R2)	Store Y(i)
I6	DADDUI R1, R1, 8	Increment X index
I7	DADDUI R2, R2, 8	Increment Y index
I8	DSGTUI R3, R1, 8000	Test if done
I9	BEQZ R3, loop	Loop if not done

Problem 1.1 (1.5 Points)

Fill in the scoreboard in the following table to simulate the execution of one iteration of the loop for in-order issue using a scoreboard. The table only shows the issuance and write-back for I1. Keep in mind that, in this scheme, no instruction is issued that has a WAW hazard with any previous instruction that has not written back (as mentioned in the lecture slides). Recall the WB stage is only relevant for FP instructions (integer instructions can forward results). You may use ellipses in the table to represent the passage of time (to compress repetitive lines). In steady state, how many cycles does each iteration of the loop take? What is the bottleneck?

22 cycles, I3, I4 is bottleneck

Inst. issued	Time (cycle)	Functional Unit Status					Registers reserved for writes
		Int	Load (1)	Add (4)	Mul (10)	WB	
I1	0		F2				F2
I2	1		F3			F2	F2, F3
I3	2				F1	F3	F3, F1
	3				F1		F1
	⋮ 11				8 blocks		F1
	12					F1	F1
I4	13			F3			F3
	⋮ 16			3 blocks			F3
	17					F3	F3
I5	18	F3					F3
I6	19	R1					
I7	20	R2					
I8	21	R3					
I9	22						

Problem 1.2 (1. 5 Points)

Now consider a single-issue out-of-order implementation. In this scheme, the issue stage buffer holds multiple instructions waiting to issue. The decode stage can add up to one instruction per cycle to the issue buffer. The decode stage adds an instruction to the issue buffer if there is space and if the instruction does not have a WAR hazard with any previous instruction that has not issued or a WAW hazard with any previous instruction that has not written back. Assume you have an infinitely large issue buffer. Assume only one instruction can be dispatched from the issue buffer at a time.

The following table represents the execution of one iteration of the loop in steady state. Fill in the cycle numbers for the cycles at which each instruction issues and writes back. The first row has been filled out for you already; please complete the rest of the table. Note that the order of instructions listed is not necessarily the issue order. We define cycle 0 as the time at which instruction I1 is issued. Draw arrows for the RAW, WAR, and WAW dependencies that are involved in the critical path of the loop in table. In steady state, how many cycles does each iteration of the loop take?

Inst. issued	Time			OP	Dest	Src1	Src2
	Decode → Issue Buffer	Issued	WB				
I1	-1	0	1	L.D	F2	R1	
I2	0	1	2	L.D	F3	R2	
I3	1	2	12	MUL.D	F1	F2	F0
I4	3	13	17	SUB.D	F3	F1	F3
I5	4	18	—	S.D		F3	R2
I6	5	6	7	DADDUI	R1	R1	
I7	6	7	8	DADDUI	R2	R2	
I8	7	8	9	DSGTUI	R3	R3	
I9	8	9	—	BEQZ		R3	

Problem 2: Branch Prediction (1.0 points)

Suppose that your friend Li Hua is designing a processor with the complex pipeline illustrated below:

I	PC Generation/Mux
F	Instruction Fetch
B	Branch Address Calc/Begin Decode
I	Complete Decode
J	Steer Instructions to Functional units
R	Register File Read
E	Integer Execute
…….	Remainder of execute pipeline (+another 5 stages)

The processor has the following characteristics:

- Issues at most one instruction per cycle.
- Branch addresses are known at the end of the B stage (Branch Address Calc/Begin Decode).
- Branch conditions (taken/not taken) are known at the end of the R stage (Register File Read).
- Branches always go through the pipeline without any stalls or queuing delays.

Li Hua's target program is shown below:

```
for(int i = 0; i <= 99999999; i++) {
      if(i % 2 == 0) //Branch B1
      {     //Not taken
            (Do something A)
      }
      if(i % 5 == 0) //Branch B2
      {     //Not taken
            (Do something B)
      }
} //Branch B3
```

```
      ANDi R1 0              ; R1 = 0 & X
Loop: MODi R2 R1 2           ; R2 = R1 % 2
      BNE  R2 Here           ; BR1, jump if R2 is not 0
      ……                     ; Do something
Here: MODi R3 R1 5           ; R3 = R1 % 5
      BNE  R3 End            ; BR2, jump if R3 is not 0
      ……                     ; Do something
End:  ADDi R1 R1 1           ; R1 = R1 + 1
      SUBi R4 R1 99999999
      BNE  R4 Loop           ; BR3, jump if R4 is not 0
      ……                     ; Do something
```

Problem 2.1 (0.2 Points)

In steady state, what is the probability for each branch in the code to be taken/not taken on average? Fill in the table below.

Branch	Probability to be taken	Probability to be not taken
BR1	50%	50%
BR2	80%	20%
BR3	100%	0%

Problem 2.2 (0.4 Point)

In steady state, how many cycles per iteration are lost on average if the processor always speculates that every branch is **not** taken (i.e., next PC is PC+4)?

We take 10 iterations as an example

Miss numbers BR1 5
BR2 8
BR3 10

cycles per iteration lost = $\frac{5+8+10}{10} \cdot 5 = 11.5$

each misspredict will cause 5 cycles lost
each successful prediction will cause 0 cycles lost

Problem 2.3 (0.4 Point)

Li Hua designs a static branch predictor to improve performance. This predictor always predicts **not** taken for **forward** jumps and **taken** for **backward** jumps. The prediction is available at the end of the B stage. In steady state, how many cycles per iteration are lost on average?

Miss cycles: BR1 5
BR2 8
BR3 0

cycles per iteration lost = $\frac{5+8+0}{10} \cdot 5 + \frac{5+2+10}{10} \cdot 2$
$= 6.5 + 3.4$
$= 9.9$ cycles

each successful prediction will cause 2 cycles lost
each misspredictor prediction will cause 5 cycles lost