Computer Sciences Department
University of Wisconsin-Madison
CS/ECE 552 – Introduction to Computer Architecture
In-Class Exercise (02/25) SOLUTION

Answers to all questions should be uploaded on Canvas.

1. [2 points] From the textbook: (Twist on Check Yourself 4.8) Consider three branch prediction schemes: predict not taken, predict taken, and dynamic prediction. Assume that they all have zero penalty when they predict correctly and two cycles when they are wrong. Assume the average prediction accuracy of the dynamic predictor is 90%. Which predictor is the best choice for the following branches?

(1) A branch that is taken with 5% frequency
(2) A branch that is taken with 95% frequency
(3) A branch that is taken with 70% frequency

Given that predict not taken is the best for branch 1, predict taken is the best for branch 2, and the dynamic predictor is best for branch 3, what are the prediction rates for each of the three predictors, for each of the 3 branches?

Solution:

For branch 1, predict not taken is the best, because it will be right 95% of the time (whereas the dynamic predictor will only be right 90% of the time and the predict taken predictor will only be right 5% of the time)

For branch 2, predict taken is the best, because it will be right 95% of the time. Predict not taken will only be right 5% of the time, and the dynamic predictor will again be right 90% of the time.

For branch 3, the dynamic predictor is the best, because it will be right 90% of the time. Predict not taken will only be right 30% of the time, and predict taken will be right 70% of the time.

2. [4 points] Consider the following assembly program to be executed in the five-stage pipeline with full forwarding and bypassing:

```
...
L1.     or      $s3, $s1, $s0
L2.     lw      $s2, 64($s3)
L3.     lw      $s3, 0($s2)
L4.     ori     $s4, $s3, 4
```

(a) [2 points] Complete the pipeline timing diagram below, where F=fetch (IF), D=decode (ID), X=execute (EX), M=memory (MEM) and W=writeback (WB). Use "*" to indicate a pipeline stall at the clock cycle of the corresponding instruction. For example, L1 stalls in the D stage in cycle 1 (due to some instruction before it, not shown) and then completes the D stage in cycle 2, so in cycle 1 we label it "D*".

Solution:

Instr / Cycle	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**

or $s3, $s1, $s0	F	D*	D	X	M	W									
lw $s2, 64($s3)		F*	F	D	X	M	W								
lw $s3, 0($s2)				F	D*	D	X	M	W						
ori $s4, $s3, 4					F*	F	D*	D	X	M	W				

(b) [2 points] Give the conditions (control signals and register comparisons) that enabled each EX-stage or MEM-stage forwarding.

Solution:

Cycle 4:
- EX/MEM.RegWrite == 1 and ID/EX.Rs == EX/MEM.Rd, hence EX → EX forwarding (EX-stage forwarding) is activated from L1 to L2

Cycle 6:
- MEM/WB.RegWrite == 1 and ID/EX.Rs == MEM/WB.Rt, hence MEM-stage forwarding is activated from L2 to L3 (NOTE: you could potentially include "MEM/WB.MemRead == 1" in this calculation, to ensure that we only use this forwarding path with Rt for accesses that load from memory)

Cycle 8:
- MEM/WB.RegWrite == 1 and ID/EX.Rs == MEM/WB.Rt, hence MEM-stage forwarding is activated from L3 to L4 (NOTE: you could potentially include "MEM/WB.MemRead == 1" in this calculation, to ensure that we only use this forwarding path with Rt for accesses that load from memory)

3. [4 points] Assume the five-stage pipeline with full forwarding and bypassing. Assume that branch decisions are made at the ID stage of the pipeline and that **data forwarding** from the EX stage is available to the branch decision circuit at the ID stage. Assume no branch delay slot. Use a predict-not-taken policy: until the branch decision is resolved, instructions on the branch-not-taken path continue to be fetched and are then **flushed** if the branch is found to be taken. Consider the following two code segments:

Code Segment A	Code Segment B

`L1: lw $t1, 0($t2)`
`L2: lw $t3, 0($s2)`
`L3: beq $t1, $t3, L1`
`L4: sub $t4, $t2, $s1`
	`L1: add $t1, $s0, $s1`
	`L2: add $t1, $s0, $t1`
	`L3: beq $t1, $0, L1`

(a) [2 points] For code segment A, complete the pipeline timing diagram until L4 is executed. Use "*" to indicate an instruction stalling (e.g., "D*" implies stalling in decode) and "=" to indicate an instruction getting flushed. Assume the branch is taken the first time and not taken the next time. **Note: the extra rows are for you to fill in L1-L4 as needed to show how the branch affects the pipeline diagram. You may not need all of the rows.**

Solution:

Instr / Cycle	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**

L1	F	D	X	M	W												
L2		F	D	X	M	W											
L3			F	D*	D*	D	X	M	W								
L4				F*	F*	F	=										
L1							F	D	X	M	W						
L2								F	D	X	M	W					
L3									F	D*	D*	D	X	M	W		
L4										F*	F*	F	D	X	M	W	

(b) [2 points] For code segment B, identify all the data dependences. For each one, specify if it leads to a data hazard, if forwarding is needed and if any stall cycles are needed.

Solution:

WAW: L1-L2 ($t1, no hazard, no stall)
WAW: L2-L1 ($t1, no hazard, no stall)
RAW: L1-L2 ($t1, hazard, EX forwarding, no stall)
RAW: L2-L3 ($t1, hazard, EX forwarding, one stall cycle)
WAR: L3-L1 ($t1, no hazard, no stall)