The University of Alabama in Huntsville
ECE Department
CPE 431 01, CPE 531 01/01R
Fall 2022
Advanced Pipelining

Due October 12, 2022

1. 0(10), 2.0(10), 3.0.1(10), 3.0.2(10), 4.0(30)

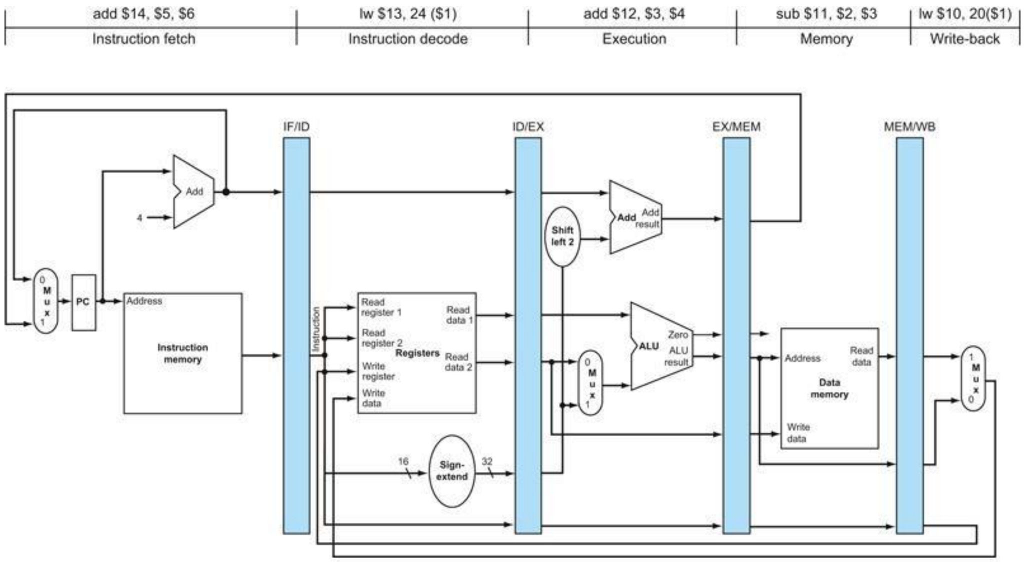
1.0 **<4.7>** This exercise is intended to help you understand the cost/complexity/performance tradeoffs of forwarding in a pipelined processor. Problems in this exercise refer to pipelined datapaths from Figure 4.45. Thse problems assume that, of all the instructions executed in a processor, the following fraction of these instructions have a particular type of RAW data dependence. The type of RAW data dependence is identifies by the stage that produces the result (EX or MEM) and the instruction that consumes the result (1st instruction that follows the one that produces the result, 2nd instruction that follows, or both). We assume that the register write is done in the first half of the clock cycle and that register reads are done in the second half of the cycle so “EX to 3rd” and “MEM to 3rd” dependences are not counted because they cannot result in data hazards. Also, assume that the CPI of the processor is 1 if there are no data hazards.

EX to 1st Only	Memto 1st Only	EX to 2nd Only	MEMto 2nd Only	EX to 1st and MEM to 2nd	Other RAW dependences
10%	25%	5%	10%	15%	10%

EX to 1st only example
```
add   $t0, $t1, $t2
add   $t3, $t0, $t4
```
MEM to 1st only example
```
lw    $t0, 20($t1)
add   $t3, $t0, $t4
```
EX to 2nd only example
```
add   $t0, $t1, $t2
add   $t3, $t5, $t4
add   $t6, $t0, $s0
```
MEM to 2nd only example
```
lw    $t0, 20($t1)
add   $t3, $t5, $t4
add   $t6, $t0, $s0
```
EX to 1st and MEM to 2nd example
```
lw    $t4, 24($t0)
add   $t9, $s0, $s4
add   $s1, $t4, $t9
```
Other RAW Dependence example
```
add   $t0, $t1, $t2  add
$s0, $t3, $s0


addi  $s1, $s1, 4

add   $t3, $t0, $t4
```



If we use forwarding only from the MEM/WB pipeline register, what fraction of cycles are we stalling due to data hazards?

Ex to 1st + memory to first + EX to 1st -Mem 2nd
10 + 25 + 15 = 0.5 (using only MEM/WB)
0.5 + 1 for one cycle stalled
Fraction of stalling using MEM/WB = 0.5/1.5 = 0.33

2.0 **<4.8>** This exercise is intended to help you understand the relationship between delay slots, control hazards, and branch exectution in a pipelined processor. In this exercise, we assume that the following MIPS code is executed on a pipelined processor with a 5-stage pipeline, full forwarding, and a predict-taken branch predictor:

```
         lw    r2, 0(r1)
label1:  beq   r2, r0, label2     # not taken once, then taken
         lw    r3, 0(r2)
         beq   r3, r0, label1     #taken
         add   r1, r3, r1
label2:  sw    r1, 0(r2)
```

Draw the pipeline execution diagram for this code, assuming there are no delay slots and that branches execute in the EX stage.

Cycle	IF	ID	EX	MEM	WB
1	Lw r2				
2	Beq r2	Lw r2			
3	Lw r3	Beq r2	Lw r2		
4	sw r1	Beq r2	bubbles	Lw r2	

5	after	sw r1	Beq r2 (predict wrong)	bubbles	Lw r2
6	Lw r3	------	(clear pipeline)	Beq r2	bubbles
7	Beq r3	Lw r3	------	(clear pipeline)	Beq r2
8	Beq r2	Beq r3	Lw r3	-------	(clear pipeline)
9	Beq r2	Beq r3	Beq r3 (prediction correct)	Lw r3	-------
10	Sw r1	Beq r2	Beq r3	bubbles	Lw r3
11		Sw r1	Beq r2 (prediction correct)	Beq r3	bubbles
12			Sw r1	Beq r2	Beq r3
13				Sw r1	Beq r2
14					Sw r1

3.0 **<4.8>** The importance of having a good branch predictor depends on how often conditional branches are executed. Together with branch predictor accuracy, this will determine how much time is spent stalling due to mispredicted branches. In this exercise, assume that the breakdown of dynamic instructions into various instruction categories is as follows:

R-type	BEQ	JMP	LW	SW
45%	20%	5%	20%	10%

Also, assume the following branch predictor accuracies:

Always-Taken	Always-Not-Taken	2-Bit
40%	60%	80%

3.0.1 Stall cycles due to mispredicted branches increase the CPI. What is the extra CPI due to mispredicted branches with the always-taken predictor? Assume that branch outcomes are determined in the EX stage, that there are no data hazards, and that no delay slots are used.

CPI (midpredict) = (1-0.40) * .2 * 2 (stalls from EX) = 0.24

3.0.1 With the 2-bit predictor, what speedup would be achieved if we could convert half of the branch instructions in a way that replaces a branch instruction with an ALU instruction? Assume that correctly and incorrectly predicted instructions have the same chance of being replaced.

CPI (mispredict) = 1+ (1-.8) * 2 (stalls) * 0.2 = 0.08
CPI(mispredict) = 1 + (1-.8) * 2 (stalls) * 0.1 = 0.04 (Half ALU)

P(alu)/P(original) = CPI(o)/CPI(alu) = 1.08/1.04 = 1.038

4.0 <4.10> In this exercise, we consider the execution of a loop in a statically scheduled superscalar processor that has full forwarding. To simplify the exercise, assume that any combination of instruction types can execute in the same cycle, e.g., in a 3-issue superscalar, the three instructions can be three ALU operations, three branches, three load/store instruction, or any combination of these instructions. Note that this only removes a resource constraint, but data and control dependences must be still be handled correctly. Problems in this exercise refer to the following loop:

```
Loop:  lw   $t3, 0($s1)
lw   $t4, 0($s2)
mul  $t1, $t3, $t4
add  $s0, $t1, $s0
addi $s1, $s1, -8
addi $s2, $s2, -8
bne  $s1, $zero, Loop
```

Unroll this loop so that four iterations of it are done at once and schedule it for a 2-issue static superscalar processor. Assume that the loop always executes a number of iterations that is a multiple of 4. You can use any unused registers when changing the code to eliminate dependences.

	R-type/branch	LW/sw
1	Addi $s1, $s1, -32	Lw $t3, 0($s1)
2	Addi $s2, $s2, -32	Lw t4, 0($s2)
3	nop	Lw $t5, 24($s1)
4	Mul $t1, $t3 $t4	Lw $t6, 24($s2)
5	Add $s0, $t1, $s0	Lw $t7, 16($s1)
6	Mul $t1, $t5, $t6	Lw $t8, 16($s1)
7	Add $s0, $t1, $s0	Lw $t9, 8($s1)
8	Mul $t1, $t7, $t8	Lw $t10, 8($s1)
9	Add $s0, $t1, $s0	
10	Mul $t1, $t9, t10	
11	Add $s0, $t1, $s0	