CS 1541
Homework 2 Answer Key | Total 20 points

Question 1 (6 points): Assume that the cycle time in the pipelined architecture does not allow the writing and reading of the register file in the same cycle. Hence, if at a given cycle, the ID stage wants to read from the register file and the WB stage wants to write to the register file, then one of the two stages has to stall and use the register file in a later cycle. In this question, assume that when the ID and WB stages compete for the register file in a given cycle, the WB stage gets to use the register file.

a. Using a table similar to the one shown below, trace the execution of the first 7 instructions of the following code segment (ignore instructions I8 and I9 for now) on a 5-stage pipeline architecture that has hardware support for forwarding and stalling:

I1:	lw	$2, 100($1)
I2:	addi	$1, $1, -4
I3:	sw	$2, 100($1)
I4:	add	$2, $2, $5
I5:	sw	$2, 100($3)
I6:	addi	$3, $3, -4
I7:	bneq	$1, $6, I1
I8:	sub	$1, $1, $7
I9	sub	$3, $3, $7

	IF	ID	EXE	MEM	WB
Cycle 1	lw $2, 100($1)				
Cycle 2	addi $1, $1, -4	lw $2, 100($1)			
Cycle 3	sw $2, 100($1)	addi $1, $1, -4	lw $2, 100($1)		
Cycle 4	add $2, $2, $5	sw $2, 100($1)	addi $1, $1, -4	lw $2, 100($1)	
Cycle 5	sw $2, 100($3)	add $2, $2, $5	sw $2, 100($1)	addi $1, $1, -4	lw $2, 100($1)
Cycle 6	sw $2, 100($3)	add $2, $2, $5	----	sw $2, 100($1)	addi $1, $1, -4
Cycle 7	sw $2, 100($3)	add $2, $2, $5	----	----	sw $2, 100($1)
Cycle 8	addi $3, $3, -4	sw $2, 100($3)	add $2, $2, $5	----	----
Cycle 9	bneq $1, $6, I1	addi $3, $3, -4	sw $2, 100($3)	add $2, $2, $5	----
Cycle 10	sub $1, $1, $7	bneq $1, $6, I1	addi $3, $3, -4	sw $2, 100($3)	add $2, $2, $5
Cycle 11	sub $1, $1, $7	bneq $1, $6, I1	----	addi $3, $3, -4	sw $2, 100($3)
Cycle 12	lw $2, 100($1)	----	bneq $1, $6, I1	----	addi $3, $3, -4
Cycle 13	addi $1, $1, -4	lw $2, 100($1)	----	bneq $1, $6, I1	----
Cycle 14	sw $2, 100($1)	addi $1, $1, -4	lw $2, 100($1)	----	bneq $1, $6, I1
Cycle 15		sw $2, 100($1)	addi $1, $1, -4	lw $2, 100($1)	----
Cycle 16			sw $2, 100($1)	addi $1, $1, -4	lw $2, 100($1)
Cycle 17				sw $2, 100($1)	addi $1, $1, -4
Cycle 18				--	sw $2, 100($1)

a. In your answer to part a, only stalling due to structural hazards is considered (there is no data or control hazards). In this part, you will assume that the branch condition is resolved in the ID stage, and hence, if the branch condition is true, then the instruction that entered the pipeline after the branch will have to be flushed to deal with control hazards. To demonstrate this effect, augment the table in your answer to part (a) by tracing the pipeline configuration for a few additional cycles after the bneq instruction reaches the WB stage assuming that the branch condition evaluates to "true".

   See table in part (a)

b. Assuming that the loop (I1-I7) will execute 10000 iterations (in each iteration, except the last time, the branch condition will be true), compute the CPI during the execution of the loop (ignore the time to fill up the pipeline).

It is possible to extend the answer to part (b) to complete iteration 2 and trace all 10000 iterations. However, it is clear that the execution pattern repeats every 11 cycles, which means that it take 11 cycles to execute the 7 instructions in one iteration (there are 4 no-ops between instances of I1 in consecutive loop iterations). To simplify the calculation, we ignored the fact that the last iteration will take only 10 cycles.

CPI = $\frac{cycles}{instruction} = \frac{11*10000}{7*10000} = 1.57$

Question 2 (6 points):

In this question, we will assume that the following code is executed on a 5- stage pipeline architecture that does not implement forwarding or stalling in hardware.

```
(1)     add $4, $1, $1
(2)     lw $2, 40($4)
(3)     sub $2, $2, $5
(4)     sw $2, 80($4)
(5)     lw $3, 10($4)
(6)     sub $3, $3, $5
(7)     sw $3, 90($4)
(8)     beq $3, $6, L
…..
    L:  ……
```

a. Identify the data dependences that cause hazards (use sentences such as "dependence of instruction (y) on instruction (z) caused by register $x).

a. Instruction 2 depends on instruction 1 (register $4)
b. Instruction 3 depends on instruction 2 (register $2)
c. Instruction 4 depends on instructions 3 and 2 (register $2)
d. Instruction 6 depends on instruction 5 (register $3)
e. Instruction 7 depends on instruction 6 and 7 (register $3)
f. Instruction 8 depends on instruction 6 (register $3)

b. If the hazards identified in (a) are to be prevented by the compiler through no-op insertions but without reordering the instructions, indicate where would the compiler add no-ops (use sentences such as “insert x no-ops between instructions (y) and (z)).
   a. Insert 2 NOP between instructions 1 and 2
   b. Insert 2 NOP between instructions 2 and 3
   c. Insert 2 NOP between instructions 3 and 4
   d. Insert 2 NOP between instructions 5 and 6
   e. Insert 2 NOP between instructions 6 and 7

c. Now assume a more intelligent compilers that can move instructions in the code. Can such a compiler reduce the number of added no-ops? If so, show the new code and indicate the added no ops.

```
(1) add $4, $1, $1
    nop
    nop
(2) lw  $2, 40($4)
(5) lw  $3, 10($4)
    nop
(3) sub $2, $2, $5
(6) sub $3, $3, $5
    nop
(4) sw $2, 80($4)
(7) sw $3, 90($4)
(8) beq $3, $6, L
```

By re-ordering instructions we can reduce the NOPs from 10 to 4.

In fact, by moving the first instruction to after the two “lw” instructions and modifying the immediate constants in the “lw” instructions appropriately, we can reduce the number of no-ops to 1 as follows:

```
lw   $2, 44($4)
lw   $3, 14($4)
add $4, $1, $1
sub $2, $2, $5
sub $3, $3, $5
```

```
nop
sw  $2, 80($4)
sw  $3, 90($4)
beq $3, $6, L
```

Question 3 (5 points): In this question, you will consider the instruction "R-type-rm $r1, $r2, $r3" that is described in Question 3 of Homework 1. This figure shows the modifications to the 5-stage pipelined architecture to accommodate the R-type-rm instruction. Specifically, a second ALU stage, EX2, is added to perform the arithmetic/logic operation for R-type-rm instructions after the second operand is read from memory, thus resulting in a 6-stage pipeline.

(a) Specify the values of the control signals (ALUsrc1, ALUsrc2, MemtoReg, MemWrite and MemRead) to implement the correct data paths for the R-type and the R-type-rm instructions. Also, specify what operation should the ALU in the first ALU stage (EX1) perform when an R-type-rm instruction is in the EX1 stage.
for R-type: ALUsrc1=1 , ALUsrc2=0, MemtoReg = 10, MemWrite= MemRead=0
for R-type-rm: ALUsrc1=1 , ALUsrc2=x, MemtoReg = 00, MemWrite= 0, MemRead=1
The ALU in the EX1 stage should perform "output = the first (upper) input"

(b) In the 5-stage pipeline architecture, we added two forwarding paths from the EX/MEM and the MEM/WB buffers to the ID/EX buffer to avoid data hazards. These paths can be denoted EX/MEM → ID/EX and MEM/WB → ID/EX, respectively. We can demonstrate the use of the MEM/WB → ID/EX path in avoiding hazards using this short code segment:

```
add $1, $2, $3
sw   $4, 100($5)
sub  $6, $7, $1
```

Using a similar notation, identify the forwarding paths that should be added to the 6-stage pipeline to avoid data hazards and for each forwarding path, give a short example of a code segment that uses this path to avoid hazard.

```
EX1/MEM → ID/EX1   example code      add $1, $2, $3
                                     sub  $6, $7, $1
MEM/EX2 → ID/EX1   example code      add $1, $2, $3
                                     sw   $4, 100($5)
                                     sub  $6, $7, $1
EX2/WB → ID/EX1    example code      add-rm $1, $2, $3
                                     sw   $4, 100($5)
                                     sub  $8, $9, $10
                                     sub  $6, $7, $1
```

In the 5-stage pipeline, a forwarding path MEM/WB → EX/MEM may be added to avoid stalling the pipe when lw is immediately followed by sw that depends on it.
In the 6-stage pipeline, forwarding paths MEM/EX2 → EX1/MEM and and

EX2/WB → MEM/EX2 may be added for the same purpose**. You are not required to provide this part of the answer since MEM/WB → EX/MEM was not presented as a standard path for the 5-stage pipeline.**

(c) Even with the forwarding paths, the 5-stage pipeline had to stall for one cycle during the execution of the following code segment

```
lw   $4, 100($5)
sub  $6, $7, $4
```

to avoid data hazards. In other words, a one-cycle stall is needed when a "lw" instruction writes to a register and is immediately followed by an instruction that reads from this register. Use short code segments to describe the cases that will necessitate the stalling of the 6-stage pipeline. Note that there are more than one case and in some cases, the pipeline may have to stall for more than one cycle.

We still have to stall if a lw that writes into a register is followed by an instruction that reads from that register

```
lw   $4, 100($5)
sub  $6, $7, $4
```

we also have to stall if an R-type-rm instruction that writes into a register and an instruction that reads that register is not separated from the R-type-rm instruction by at least two instructions For example, the code segment

```
Rtype-rm $1, $2, $3
sub      $4, $5, $5
add      $6, $7, $1
```

will cause a stall for one cycle, while the code

```
Rtype-rm $1, $2, $3
sub      $4, $5, $1
```

will cause a stall for two cycles

Question 4 (3 points):

Consider two designs for the 5-stage pipelined architecture. In the first design, the branch condition and target are resolved in the MEM stage, while in the second design, the branch condition and target are resolved in the ID stage but the clock cycle time is 10% larger than in the first design. Assuming that

- the instruction mix executing on the pipeline contains 25% branch instructions, 70% of which are taken
- the CPI of both designs is 1.5 if we do not account for stalling due to control hazards

a. Compute the CPI for each of the two designs when the effect of control hazards is accounted for

CS 1541
Homework 2 Answer Key | Total 20 points

We assume that the branch prediction is always “not taken” so a taken branch results in 1 or more flushed instructions.

Design 1 CPI: Resolved in the MEM stage, so 3 instructions are flushed following a taken branch
CPI = 1.5 + (0.25 * 0.7 * 3) = 2.025

Design 2 CPI: Resolved in the ID stage, so 1 instruction is flushed following a taken branch
CPI = 1.5 + (0.25 * 0.7 * 1) = 1.675

b. Specify which of the two designs would be more efficient.

Given a fixed-size program run on both designs, assume the clock cycle of Design 1 is *x* and that the clock cycle of Design 2 is *1.1x.*

Calculating the execution time for the same fixed size program:

Execution time on Design 1 = 2.025 x
Execution time on Design 2 = 1.675 * 1.1 x = 1.842

Design 2 would take less time to execute the same program, Hence it is the more efficient design.