CSE 30321 – Computer Architecture I – Fall 2010
Homework 06 – Pipelined Processors – 85 points
Assigned: November 2, 2010 – **Due:** November 9, 2010
PLEASE DO THE ASSIGNMENT ON THIS HANDOUT!!!

Problem 1: (25 points)

Background:
In this exercise, we examine how pipelining affects the clock cycle time of the processor. Problems in this exercise assume that individual stages of the datapath take the following amounts of time to complete:

IF	ID	EX	MEM	WB
400 ps	225 ps	350 ps	450 ps	300 ps

Questions:

Part A: (5 points)
- How long will it take to completely process a **single** sw instruction in a pipelined processor?
- What about in a non-pipelined, multi-cycle processor?
- Comment on the difference in execution times.

Part B: (5 points)
- How long will it take to completely process 100 add instructions in a pipelined processor? (The add instructions may be dependent on one another, but you can assume that any dependencies can be resolved by forwarding so no stalls are needed.)
- What about in a non-pipelined, multi-cycle processor?
- Comment on the difference in execution times.

Part C: (5 points)
To try to improve performance further, you have decided to break up 2 of the above stages into 2 shorter stages. (In other words, you are going to increase the depth of your pipeline.)

- What current stages would you recommend breaking up?
- How would this affect your answer to part B?

(Note: Your decision about the second stage to break up should be based on the pipeline after you break up the first stage.)

Note:
The remaining parts of this question assume that instructions executed are broken down as follows:

ALU	beq	lw	sw
45%	15%	25%	15%

Part D: (5 points)
Assuming there are no stalls or hazards (and ignoring time to fill or drain the pipeline) what percentage of CCs for a single instruction require a reference to memory in a pipelined datapath?

Part E: (5 points)
Assuming there are no stalls or hazards (and ignoring time to fill or drain the pipeline) what percentage of time do instructions need to access memory in the same CC? (Assume that the instruction memory and data memory are the same "memory".)

<u>Problem</u> <u>2</u>: (25 points)

<u>Background</u>:

In this question, we examine how data dependencies affect execution in the basic 5-stage pipeline discussed in class. Problems in this exercise refer to the following sequence of instructions:

A:	sub	\$1, \$2, \$3
B:	add	\$4, \$1, \$1
C:	lw	\$5, 0(\$1)
D:	add	\$6, \$5, \$2
E:	add	\$7, \$5, \$3
F:	sub	\$8, \$6, \$7
G:	sw	\$7, 0(\$8)

<u>Questions</u>:

<u>Part</u> <u>A</u>: (10 points)

Assume there is full forwarding. Show how the above sequence of instructions would flow through the pipeline. Identify data dependencies and note when/if they become data hazards.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
A: sub \$1, \$2, \$3																	
B: add \$4, \$1, \$1																	
C: lw \$5, 0(\$1)																	
D: add \$6, \$5, \$2																	
E: add \$7, \$5, \$3																	
F: sub \$8, \$6, \$7																	
G: sw \$7, 0(\$8)																	

List Dependencies/Hazards Here:

Note:

- Refer to the following table to answer the remaining 2 questions.

Without forwarding	With full forwarding
350 ps	400 ps

Part B: (10 points)
What is the execution time of this instruction sequence without forwarding and with full forwarding? (A register *may* be read and written in the same CC.) What is the speedup achieved by adding full forwarding to a pipeline that had no forwarding? Complete the trace to help you answer this question.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
A: sub \$1, \$2, \$3																				
B: add \$4, \$1, \$1																				
C: lw \$5, 0(\$1)																				
D: add \$6, \$5, \$2																				
E: add \$7, \$5, \$3																				
F: sub \$8, \$6, \$7																				
G: sw \$7, 0(\$8)																				

Part C: (5 points)
Is there anything that you or the compiler could do to further improve performance of the design *with* full forwarding?

Problem 3: (20 points)

Background:

This problem will help you understand how certain code sequences can degrade the performance of pipelined code, and how hardware can help avoid performance degradation.

Part A: (10 points)

Identify all of the data dependencies in the following code. Which dependencies are data hazards that can be resolved via forwarding? Which dependencies are data hazards that will cause a stall? (Hint: always consider what clock cycle a piece of data is produced in, and in what clock cycle that data is ultimately needed by another instruction.)

```
lw      $10, 32($11)
add     $8, $10, $7
sub     $6, $8, $10
add     $7, $6, $8
sw      $9, 0($7)
```

	1	2	3	4	5	6	7	8	9	10	11
lw $10, 32($11)											
add $8, $10, $7											
sub $6, $8, $10											
add $7, $6, $8											
sw $9, 0($7)											

List Dependencies/Hazards Here:

Part B: (10 points)
Identify all of the data dependencies in the following code. Which dependencies are data hazards that can be resolved via forwarding? Which dependencies are data hazards that will cause a stall? (Hint: always consider what clock cycle a piece of data is produced in and in what clock cycle that data is ultimately needed by another instruction.)

```
lw      $10, 32($11)
sub     $7, $8, $9
add     $8, $10, $7
sub     $10, $8, $6
sw      $10, 0($8)
```

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
lw $10, 32($11)											
sub $7, $8, $9											
add $8, $10, $7											
sub $10, $8, $6											
sw $10, 0($8)											

List Dependencies/Hazards Here:

Problem 4: (15 points)

Background:

The two parts of this question will help you see how branch instructions affect how a sequence of instructions moves through a pipelined datapath.

Part A: (10 points)

- Assume that forwarding **has** been implemented.
- Assume that you **can** read and write a register in the same clock cycle
- Assume that a given register "contains its number"
  - E.g. $1 = 1, $2 = 2, $3 = 3, $4 = 4, $5 = 5, $6 = 6, etc.
- You may assume that each branch is "predicted" correctly.
- Show the instruction trace.

Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	**Comment (if needed)**
Y: add $5,$1,$1															
beq $5,$2,X															
add $5,$5,$5															
add $6,$2,$2															
X: beq $4,$6,Y															
sll $7,$1,4															

Part B: (5 points)

What if branch prediction was **not** perfect and you instead just predicted that a branch was not taken? What would happen given the instruction sequence above? (You don't have to show a pipe trace, just briefly explain.)