Homework 07
Due 10/29/08

6.1 [5] <§6.1> If the time for an ALU operation can be shortened by 25% (compared to the description in Figure 6.2 on page 373);

a. Will it affect the speedup obtained from pipelining? If yes, by how much? Otherwise, why?

b. What if the ALU operation now takes 25% more time?

Instruction class	Instruction fetch	Register read	ALU operation	Data access	Register write	Total time
Load word (lw)	200 ps	100 ps	200 ps	200 ps	100 ps	800 ps
Store word (sw)	200 ps	100 ps	200 ps	200 ps		700 ps
R-format (add, sub, and, or, slt)	200 ps	100 ps	200 ps		100 ps	600 ps
Branch (beq)	200 ps	100 ps	200 ps			500 ps

FIGURE 6.2 Total time for each instruction calculated from the time for each component. This calculation assumes that the multiplexors, control unit, PC accesses, and sign extension unit have no delay.

a). Since memory access requires 200 ps, making the ALU operations less than 200 ns will not effect performance since the segment clock must remain 200 ns to accommodate the memory.

b). If the ALU operation is increase by 25 ns the segment clock period must be increased to 200 x 1.25 = 250 ns.

6.4 [10] <§6.1> Identify all of the data dependencies in the following code. Which dependencies are data hazards that will be resolved via forwarding? Which dependencies are data hazards that will cause a stall?

```
add $3, $4, $2
sub $5, $3, $1
lw  $6, 200($3)
add $7, $3, $6
```

ANS: The last three instructions are dependent on the $3 value computed by the first instruction. There is a data dependency through $6 between the lw instruction and the last instruction. The $3 dependencies can all be resolved by the forwarding techniques we studied. The $6 dependency cannot be resolved by forwarding.

6.17 [5] <§§6.4, 6.5> Consider executing the following code on the pipelined datapath of Figure 6.36 on page 416:

```
add    $2,  $3,  $1
sub    $4,  $3,  $5
add    $5,  $3,  $7
add    $7,  $6,  $1
add    $8,  $2,  $6
```

At the end of the fifth cycle of execution, which registers are being read and which register will be written?

TIME														
	add $2, $3, $1													
1	IF	sub $4, $3, $5												
2	ID	IF	add $5,$3,$7											
3	EX	ID	IF	**ADD $7,$6,$1**										
4	DM	EX	ID	IF	add $8,$2,$6									
5	**WB**	DM	EX	**ID**	IF									
6		WB	DM	EX	ID									
7			WB	DM	EX									
8				WB	DM									
9					WB									

A the end of the fifth cycle of execution instruction 4 (add $7,$6,$1) reads registers $6 and $1. Instruction 1 writes to register $2 (actually register $2 is written a half clock earlier than the end of the fifth cycle).

6.39 [10] <§§6.4–6.6> The example on page 378 shows how to *maximize* performance on our pipelined datapath with forwarding and stalls on a use following a load. Rewrite the following code to *minimize* performance on this datapath—that is, reorder the instructions so that this sequence takes the *most* clock cycles to execute while still obtaining the same result.

```
lw    $2, 100($6)
lw    $3, 200($7)
add   $4, $2, $3
add   $6, $3, $5
sub   $8, $4, $6
lw    $7, 300($8)
beq   $7, $8, Loop
```

6.39

Good code		Bad code	
lw	$2, 100($6)	lw	$2, 100($6)
lw	$3, 200($7)	add	$4, $2, $3
add	$4, $2, $3	lw	$3, 200($7)
add	$6, $3, $5	add	$6, $3, $5
sub	$8, $4, $6	sub	$8, $4, $6
lw	$7, 300($8)	lw	$7, 300($8)
beq	$7, $8, Loop	beq	$7, $8, Loop

I think the Bad code is what the book wanted; however, the Bad code does not produce the same answer. Since the add $4,$2,$3 can't be executed before the lw $3, 200($7). The program can not be rearranged for worst performance.