Course Name: Computer Architecture Lab

Experiment: 4 – CPU Structure, Parallel Programming & Hazards,

Exceptions & Interrupts

Lab Instructor: Christos Mitropoulos

Date Performed: March 22th, 2017

Date Submitted: April 5th, 2017

Submitted by: Leonardo Roman

Assignment 1

Suppose you have the CPU of Figure 1 and that there is not any pipeline scheme. For the following program indicate the values of the control signals for each instruction.

addi $t0, $t0, 10

sw $t0, 32($s0)

bne $t2, $t0, QUIT

xor $s0, $t1, $t2

j Print

By analyzing figure 1 we can determine the values of the control signal of each instruction. The values are determined by the type of format (R,I or J).

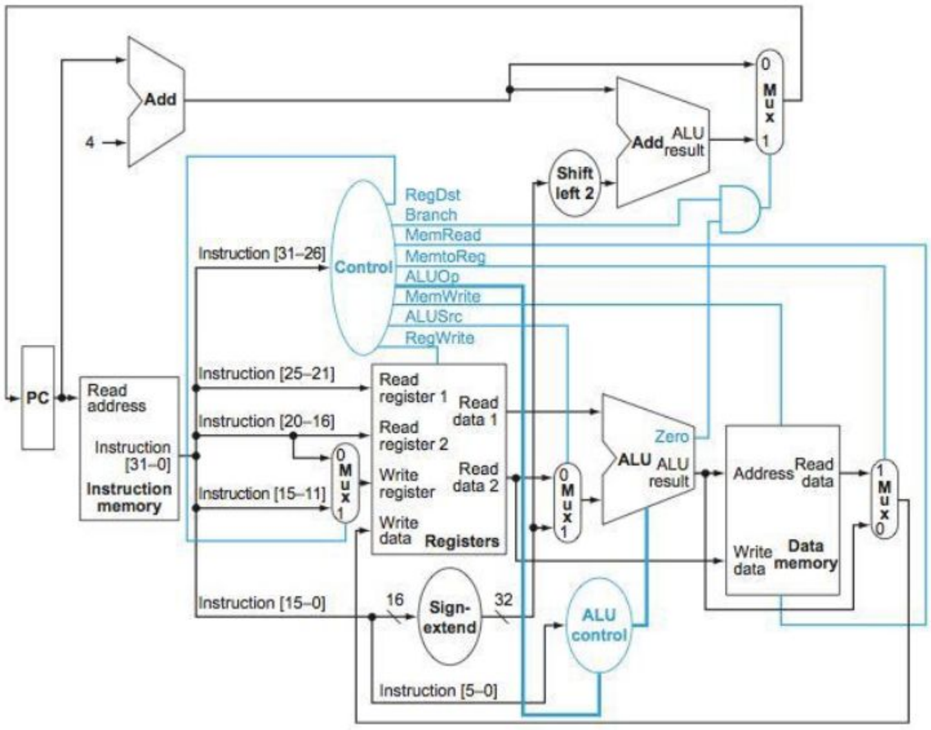


Figure 1 The simple datapath with the control unit

In this figure we can see that for all R-type rd would be 1 and I-type 0. Following the circuit in more detail we get the following logic signals table.

Instruction	Reg Dst	ALUSrc	Mento Reg	Reg Write	Mem Read	Men Write	Branch	ALUop	Jump
Addi $t0,$t0,10	0	1	0	1	0	0	0	00	0
Sw $t0,32($s0)	x	1	x	0	0	1	0	00	0
Bne $t2,$t0,QUIT	x	0	x	0	0	0	1	01	0
Xor $s0,$t1,$t2	1	0	0	1	0	0	0	10	0
J Print	x	x	x	0	0	0	x	xx	1

The x mark represents a "don't care" since in some cases it is obvious that some signals will not be used for some instructions such as sw,lw and jumps.

Assignment 2

Suppose you have a machine that supports the five pipeline stages mentioned above. Using the following programs to answer the questions:

lw $t0,0($t3)
add $t1, $t0, $t2
sub $t3, $t3, $t1
addi $t4, $t4, 4
add $t5, $t5, $t4

1. Suppose that registers $t0,t1,$t2,$t3,$t4 and $t5 have the respective values 2, 5, 8, 2, 4, and 1. Give the values of the registers after running the program, without the processor handling any hazard.

REGISTER VALUES

Reg	Initial	Final
$t0	2	x
$t1	5	10
$t2	8	8
$t3	2	-3
$4	4	8
$t5	1	-3

***2.* Insert the necessary NOP instructions so that the values of the registers would be the same as if the program was run by a processor that executes one instruction per cycle.**

Instructions	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
lw $t0,0($t3)	IF	ID	EX	MEM	WB										
NOP		S	S	S	S	S									
NOP			S	S	S	S	S								
Add $t1, $t0, $t2				IF	ID	EX	MEM	WB							
NOP					S	S	S	S	S						
NOP						S	S	S	S	S					
sub $t3, $t3, $t1							IF	ID	EX	MEM	WB				
addi $t4, $t4, 4								IF	ID	EX	MEM	WB			
NOP									S	S	S	S	S		
NOP										S	S	S	S	S	
add $t5, $t5, $t4											IF	ID	EX	MEM	WB

Where 'S' is stalling

3. Discuss how instruction reordering can avoid the hazards in the code section above.

We can take advantage of the independency between instructions by reordering them in any way that keeps the pipeline as full as possible. By looking at the instructions above. If we decided to do reordering such that $t0 gets assigned the value from 0($t3) before the next instruction needs to use $t0 then we are minimizing hasard. I.e. We can start the instruction in the following way:

lw $t0,0($t3)
addi $t4, $t4, 4
Add $t1, $t0, $t2
And we will not need $t0 until in the fifth cycle, which is when $t0 gets the value(WB) and at the same time the addition operation will need this value. In conclusion reordering delays timing in which might need to use a register. This time delay or stalling helps to insure that a particular register would be ready by the time a next instruction needs it.
4. Discuss how forwarding can affect the execution of the code section above.

With forwarding, extra hardware is added to obtain the data from internal resources, that will be needed in next instructions. Moreover, forwarding can not prevent all pipeline stalls. Stalling may still be needed even when using forwarding, however, forwarding will significantly reduce the number of stalls within a program. In the case of the previus code, forwarding can help to reduce time delay and cut down some stalls. Forwarding is valid only if the destination stage will be later in time than the source stage. For example, by looking at instruction *lw* \$*t*0,0(\$*t*3) we noticed that there is not possible valid forwarding from the output of the memory access at stage (MEM). This is because the input data for next instruction is needed at clock cycle 5. Hence a stall is needed to wait for \$t0 to be ready.

5. Compare it with NOP and instruction reordering.

a. Forwarding with NOP.

Instructions	1	2	3	4	5	6	7	8	9	10
lw \$t0,0(\$t3)	IF	ID	EX	MEM	WB					
NOP		S	S	S	S	S				
Add \$t1, \$t0, \$t2			IF	ID	EX	MEM	WB			
sub \$t3, \$t3, \$t1				IF	ID	EX	MEM	WB		
addi \$t4, \$t4, 4					IF	ID	EX	MEM	WB	
add \$t5, \$t5, \$t4						IF	ID	EX	MEM	WB

b. Forwarding with NOP and reordering.

Instructions	1	2	3	4	5	6	7	8	9
lw \$t0,0(\$t3)	IF	ID	EX	MEM	WB				
Add \$t1, \$t0, \$t2		IF	ID	EX	MEM	WB			
sub \$t3, \$t3, \$t1			IF	ID	EX	MEM	WB		
addi \$t4, \$t4, 4				IF	ID	EX	MEM	WB	
add \$t5, \$t5, \$t4					IF	ID	EX	MEM	WB

As we can see Forwarding with NOP takes most of the stalling away while the combination of forwarding with NOP and reordering smooths even greatly the clock cycle per instructions.

Assignment 3

Consider the following code:

addiu $t0,$t0,3
loop: addiu $t1,$t1, 1
sub $t0, $t0, $t1
bne $t0, $zero, loop

We can see that for the first two instructions

1. Determine how many cycles would it take for the program to execute without hazards handling.

Because of the presence of a loop, in the code above, we can see that there are 7 different instruction. Before the loop label the first instruction will take 5 cycles. Once we enter the loop there will be 3*(3-1) instructions left incrementing one cycle per instruction. Hence the total of cycles before hazard handling that this program would take is 11 cycles assuming that $t0 and $t1 are initialized to zero.

2. Draw the execution table and determine the type of hazards.

Instruction	1	2	3	4	5	6	7	8	9	10	11
addiu $t0,$t0,3	IF	ID	EX	MEM	WB						
addiu $t1,$t1, 1		IF	ID	EX	MEM	WB					
sub $t0, $t0, $t1			IF	ID	EX	MEM	WB				
bne $t0, $zero, loop				IF	ID	EX	MEM	WB			
addiu $t1,$t1, 1					IF	ID	EX	MEM	WB		
sub $t0, $t0, $t1						IF	ID	EX	MEM	WB	
bne $t0, $zero, loop							IF	ID	EX	MEM	WB

There are data and control hazards in the given set of instructions. Defined by colors, respectively, the first data hazard occurs during instruction 3 clock cycle 4. As we can see, in instruction three, registers $t0 and $t1 are need it in clock cycle 4 but they won't be ready until cycle five and six respectively. The second data hazard would occur during instruction four clock cycle 5 when register $t0 is needed. After branch instruction there will be a control hazard because the cpu has to wait whether to execute the next instruction (PC+4) or go back to the next instruction in the loop. Similarly, the next hazard occurs during instruction seven in the table which is data hazard and again after branch instruction there would be a control hazard.

3. Write program using NOP instructions to avoid the hazards.

```
addiu $t0,$t0,3
loop: addiu $t1,$t1,1
NOP
NOP
sub $t0,$t0,$t1
NOP
NOP
bne $t0,$zero,loop
NOP
```

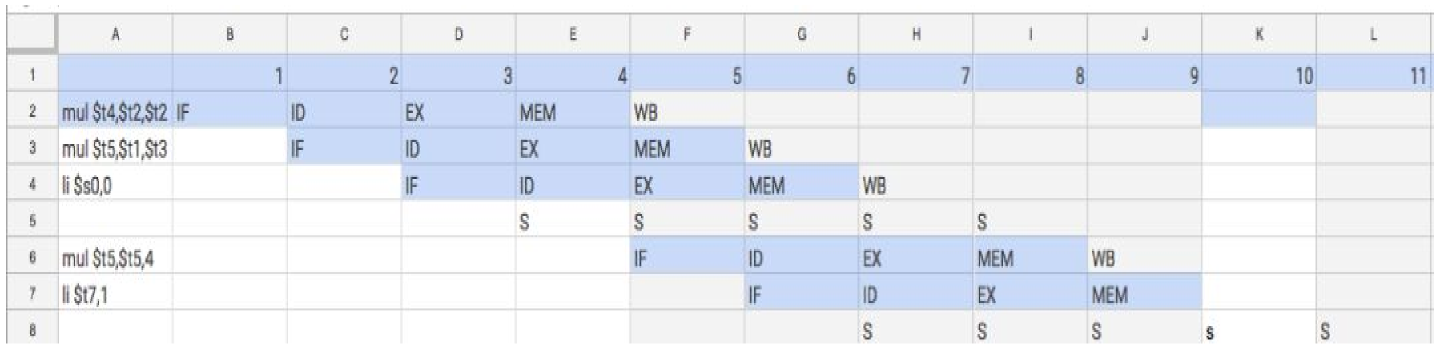
Last NOP after the the branch instruction represents that registers are being clear during looping cycles. The process of clearing registers after a branch instruction pretty much makes the fetched instruction into a NOP. This makes sense because looping instructions use same registers over and over until loop ends and the program needs to take care of the data hazards before repeating cycles.

4. Describes the effects of instruction forwarding to this code.

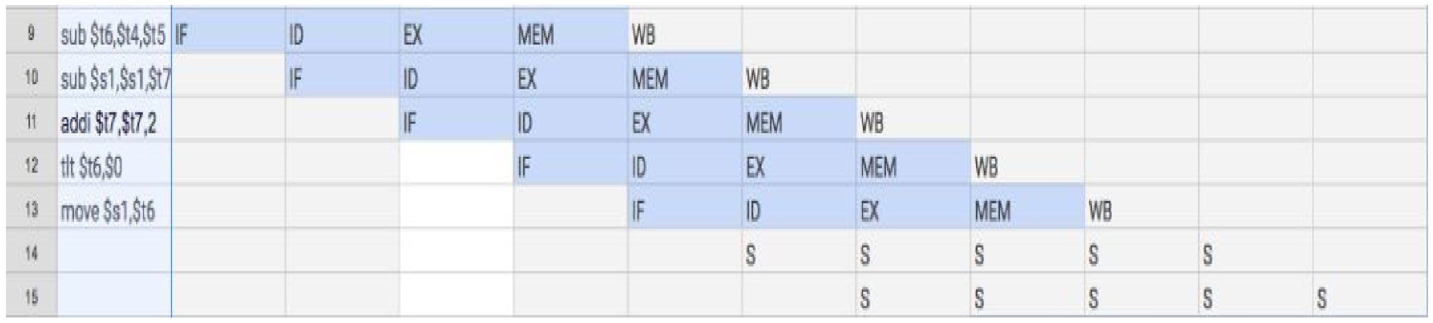
By using NOP to avoid data hazard, delay will increase, and at the same time delay would increase by a factor of number of loop iterations. If the number of iterations in the loop is increased there will be 4 to 5 additional cycles per loop. As result the program will take longer to execute. On the other hand, by forwarding, as explained in the table, the program takes only one extra cycle per iteration giving a total of 11 cycles. Since the data from earlier instruction will be only available after EX stage and won't be needed until EX stage of the next instruction, NOP is not necessary in this code.

Assignment 4

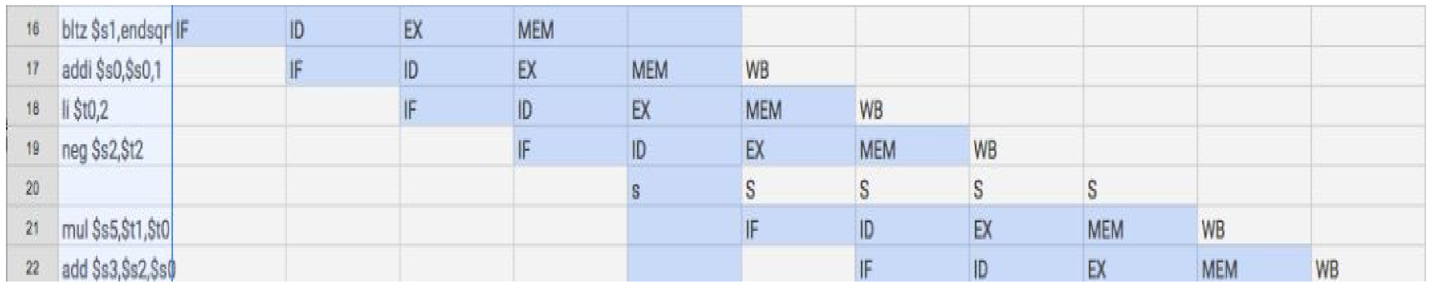
1) Rewrite the program by reordering its instructions so that you reduce the cycles needed for the execution. Also, indicate the reduction percentage and explain it.



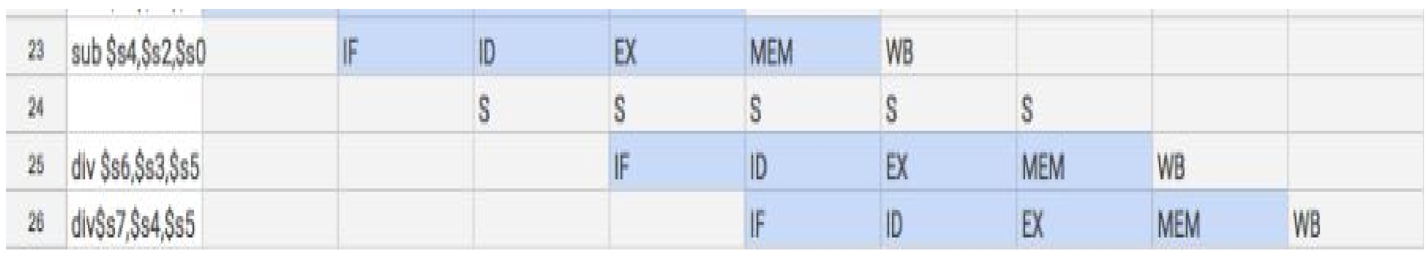
	A	B	C	D	E	F	G	H	I	J	K	L
1		1	2	3	4	5	6	7	8	9	10	11
2	mul $t4,$t2,$t2	IF	ID	EX	MEM	WB						
3	mul $t5,$t1,$t3		IF	ID	EX	MEM	WB					
4	li $s0,0			IF	ID	EX	MEM	WB				
5					S	S	S	S	S			
6	mul $t5,$t5,4					IF	ID	EX	MEM	WB		
7	li $t7,1						IF	ID	EX	MEM		
8								S	S	S	s	S



9	sub $t6,$t4,$t5	IF	ID	EX	MEM	WB						
10	sub $s1,$s1,$t7		IF	ID	EX	MEM	WB					
11	addi $t7,$t7,2			IF	ID	EX	MEM	WB				
12	tlt $t6,$0				IF	ID	EX	MEM	WB			
13	move $s1,$t6					IF	ID	EX	MEM	WB		
14							S	S	S	S	S	
15								S	S	S	S	S



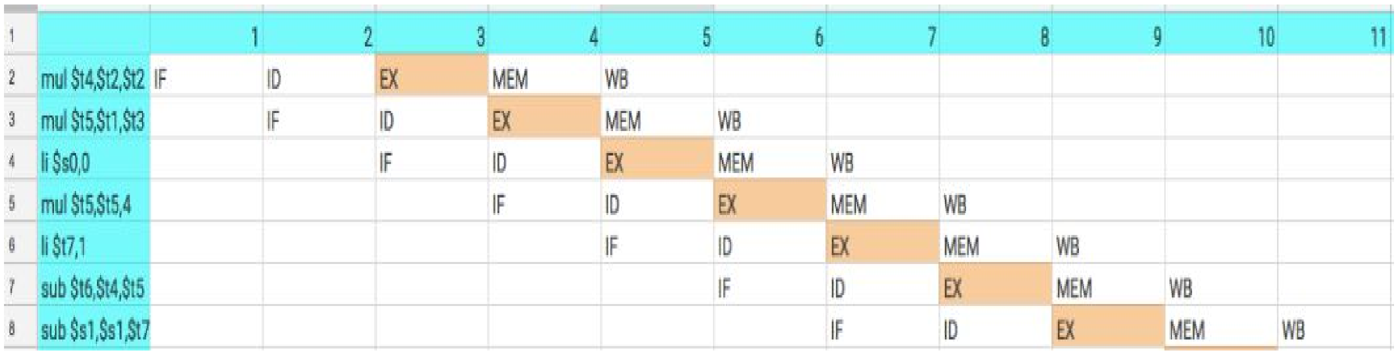
16	bltz $s1,endsqr	IF	ID	EX	MEM							
17	addi $s0,$s0,1		IF	ID	EX	MEM	WB					
18	li $t0,2			IF	ID	EX	MEM	WB				
19	neg $s2,$t2				IF	ID	EX	MEM	WB			
20						s	S	S	S	S		
21	mul $s5,$t1,$t0						IF	ID	EX	MEM	WB	
22	add $s3,$s2,$s0							IF	ID	EX	MEM	WB



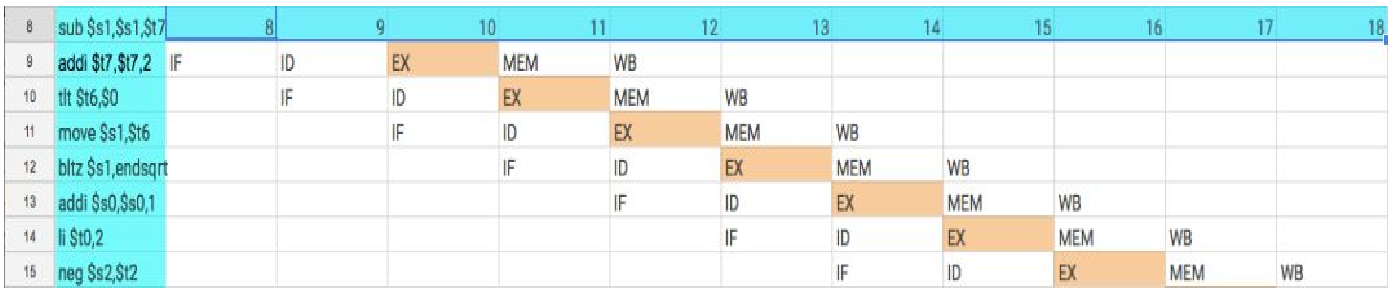
23	sub $s4,$s2,$s0		IF	ID	EX	MEM	WB			
24				S	S	S	S	S		
25	div $s6,$s3,$s5				IF	ID	EX	MEM	WB	
26	div$s7,$s4,$s5					IF	ID	EX	MEM	WB

There are 32 instructions with 36 syscalls giving a total of 68 cycles. With reordering there are 29 cycles as shown in the table above plus the 36 original syscalls, giving a total of 65 cycles. Hence there is a drop of 3 cycles.

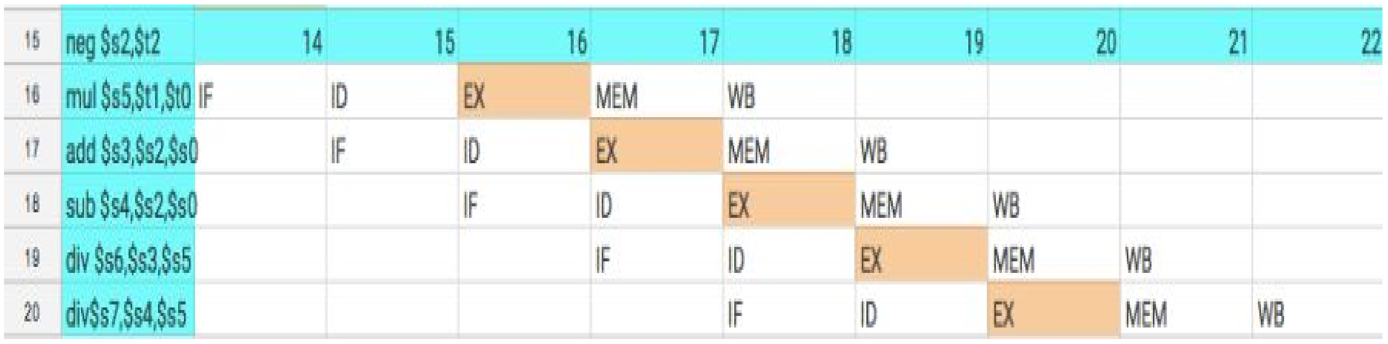
2) Explain how the execution of the program would change if instruction forwarding was supported.



1		1	2	3	4	5	6	7	8	9	10	11
2	mul $t4,$t2,$t2	IF	ID	EX	MEM	WB						
3	mul $t5,$t1,$t3		IF	ID	EX	MEM	WB					
4	li $s0,0			IF	ID	EX	MEM	WB				
5	mul $t5,$t5,4				IF	ID	EX	MEM	WB			
6	li $t7,1					IF	ID	EX	MEM	WB		
7	sub $t6,$t4,$t5						IF	ID	EX	MEM	WB	
8	sub $s1,$s1,$t7							IF	ID	EX	MEM	WB



8	sub $s1,$s1,$t7	8	9	10	11	12	13	14	15	16	17	18
9	addi $t7,$t7,2	IF	ID	EX	MEM	WB						
10	tlt $t6,$0		IF	ID	EX	MEM	WB					
11	move $s1,$t6			IF	ID	EX	MEM	WB				
12	bltz $s1,endsqrt				IF	ID	EX	MEM	WB			
13	addi $s0,$s0,1					IF	ID	EX	MEM	WB		
14	li $t0,2						IF	ID	EX	MEM	WB	
15	neg $s2,$t2							IF	ID	EX	MEM	WB



15	neg $s2,$t2	14	15	16	17	18	19	20	21	22
16	mul $s5,$t1,$t0	IF	ID	EX	MEM	WB				
17	add $s3,$s2,$s0		IF	ID	EX	MEM	WB			
18	sub $s4,$s2,$s0			IF	ID	EX	MEM	WB		
19	div $s6,$s3,$s5				IF	ID	EX	MEM	WB	
20	div$s7,$s4,$s5					IF	ID	EX	MEM	WB

As we can see in the table above. If instruction forwarding was support then the program would reduce cycle instructions greatly. The table shows a total of 22 cycles in comparison with the reordering table there is a difference of 7 cycles. In conclusion we can see how forwarding is very essential in a system. It reduces time delay in execution for all instructions.

Conclusion.

In this lab we have learned how the a system handle all type of hazards. Different techniques,such as reordering instruction and stalling, were introduced in order to comprehend how instruction work behind scene and how these techniques would handle type of instruction, data and control hazards. Moreover, in this lab we learned the importance of pipelining and how forwarding can benefit cycle time. All calculation and estimations were conducted as the lab manual described.