Course Name: Yu Liu

Course Number and Section: **14:332:333:01**

Experiment: [Experiment # 6 –CPU Structure, Pipeline Programming and Hazards]

Lab Instructor: Ali Essam

Date Performed: 12/3/2018

Date Submitted: 12/10/2018

Submitted by: Yu Liu 173001088

Course Name: ________________

Course Number and Section: **14:332:333:01**

! Important: Please include this page in your report if the submission is a paper submission. For electronic submission (email or Sakai) please omit this page.

-------------------------For Lab Instructor Use ONLY-------------------------

GRADE: _______________

COMMENTS:

Assignment 1:

addi t0, t0, 10
lb t1, 32(t0)
ori t2, t0, 4
bne t2, t0, exit

Instruction	ALUsrc	MemtoReg	RegWrite	MemRead	MemWrite	Branch	ALUOp[1:0]
addi t0, t0, 10	1	0	1	0	0	0	00
lb t1, 32(t0)	1	0	1	1	0	0	00
ori t2, t0, 4	0	0	1	0	0	0	01
bne t2, t0, exit	0	1	0	0	0	1	01

1. Addi t0,t0,10
    a. RegWrite will be 1 since the register t0 holds the value of ten. The branch will be 0. MemRead will also be 0. MemReg is 0 because the input of the value was fetched from RegWrite and that came from ALU. No MemWrite for add instructions so it will be 0. ALUsrc is 1 because the sing extended in the 2nd ALU operand and ALUop is 00.
2. Lb t1, 32(t0)
    a. The load byte memory will have a Value of 1 in MemRead and ALUOp is 00 for a load. RegWrite and ALUsrc will be 1 and the rest will be 0.
3. Ori t2, t0, 4
    a. The ori instruction signifies t2 = t0 or 4. So in the RegWrite, the value will be 1 since the register t0 has to be written into it. No branch for the given so it will be 0 and ALUop is 01 for ori. The rest remain 0.
4. Bne t2, t0 , exit
    a. The bne instruction is a branch instruction so Branch will have a value of 1. ALUop will be 01 for bne and beq instructions. Memtoreg's value is also 1 and the rest will be 0.

Assignment 2:

lw t0, 0(t3)
add t0, t0, t1
sub t1, t3, t1
addi t4, t3, 4
sub t5, t5, t4
1.

Instr/Cycle	1	2	3	4	5	6	7	8	9
lw t0, 0(t3)	IF	ID	EX	MEM	WB				
add t0, t0, t1		IF	ID	EX	MEM	WB			
sub t1, t3, t1			IF	ID	EX	MEM	WB		
addi t4, t3, 4				IF	ID	EX	MEM	WB	
sub t5, t5, t4					IF	ID	EX	MEM	WB

2.

t0 = 2
t1 = 5
t2 = 8
t3 = 2
t4 = 4
t5 = 1
lw t0, 0(t3) t0 = t3 = **2**
add t0, t0, t1 t0 = t0 + t1 = 2 + 5 = **7**
sub t1, t3, t1 t1 = t3 – t1 = 2 – 5 = **-3**
addi t4, t3, 4 t4 = t3 + 4 = 2 + 4 = **6**
sub t5, t5, t4 t5 = t5 + t4 = 1 + 6 = **7**

t0 = 7; t1= -3; t4 = -3; t4 = 6; t5 = 7;

3. Hazards:

4.

```
lw t0, 0(t3)
add t0, t0, t1
nop
sub t1, t3, t1
nop
addi t4, t3, 4
nop
sub t5, t5, t4
```

5.
Reordering can affect the code by keeping the pipeline as full as possible and it will also shorten the total amount cycles needed to execute the program.

6.
Forwarding will send the result of an instruction forward to the next instruction so that stalling and NOP will not be needed. Instruction forwarding affects the program by making it faster to execute each line of code

Assignment 3:

```
      addi t1, t0, 3
Loop: addi t2, t1, 1
      sub t0, t0, t1
      bne t0, zero, loop
```

1. It will take 8 cycles to execute without hazard prevention.

2. Execution table:

Instr/Cycle	1	2	3	4	5	6	7	8
Addi t1, t0, 3	IF	ID	EX	MEM	WB			
addi t2, 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

3.
```
      addi t1, t0, 3
      nop
Loop: addi t2, t1, 1
      sub t0, t0, t1
      nop
      bne t0, zero, loop
```

4. Instruction forwarding will shorten the total cycles for this code to run than using nop instructions.

Assignment 4:

```
# quad_sol.s
# This assembly program calculates the integer solutions of a quadratic
polynomial.
# Inputs : The coefficients a,b,c of the equation a*x^2 + b*x + c = 0
# Output : The two integer solutions.
#
# All numbers are 32 bit integers

.globl main

main:                                # Read all inputs and put them in
floating point registers.

       li t1, 1           # a=1
    li t2, -3          # b=-3
    li t3, 2           # c=2


        # In the following lines all the necessary steps are taken to
        # calculate the discriminant of the quadratic equation
        # D = b^2 - 4*a*c

        mul t4,t2,t2           # t4 = t2*t2, where t2 holds b
        mul t5,t1,t3           # t5 = t1*t3, where t1 holds a and t3 holds c

        li a3, 4

        mul t5,t5,a3           # Multiply value of s0 with 4, creating 4*a*c
        sub t6,t4,t5           # Calculate D = b^2-4*a*c

        # calculating the integer square root by the equation x*x = D
        li s0, 1                   # Square Root Partial Result, sqrt(D).
        mv s1,t6                       # Move value in register t6 to register
s1 for safety purposes.

        sqrtloop:                      # calculating the integer square root
of D

                mul s2, s0,s0
                bge s2, s1, endsqrt
                addi s0,s0, 1


        j sqrtloop


        endsqrt:
                neg s2,t2              # calculate -b and save it to s2
                li t0, 2               # Load constant number to integer
register
```

```
            add s3,s2,s0   # Calculate -b+sqrt(D) and save it to s3
            sub s4,s2,s0   # Calculate -b-sqrt(D) and save it to s4

            mul s5,t1,t0   # Calculate 2*a and save it to s5
            div s6,s3,s5   # Calculate first integer solution
            div s7,s4,s5   # Calculate second integer solution

        #Print the calculated solutions.

        li a0, 4                       # Load print_string syscall code to
register v0 for the 1st result string
        la a1, str1                    # Load actual string to register a0
        ecall

        li a0 1                # Load print_int syscall code to register v0
for the 1st result string
        mv a1, s6                      # Load actual integer to register a0
        ecall

        li a0, 4                       # Load print_string syscall code to
register v0 for the 1st result string
        la a1, str2                    # Load actual string to register a0
        ecall

        li a0, 1
        mv a1, s7
        ecall

        li a0, 10
        ecall

.data

        str1: .asciiz "The first integer solution is: "

        str2: .asciiz "\nThe second integer solution is: "
```

1. The total cycle was reduced by 4 cycles.
2. Forwarding would also decrease the total amount of cycles for this code.