|  |
| --- |
| **DIGITAL SYSTEM DESIGN LABORATORY** |
| **LAB 6** |

**MULTI-CYCLE MICROPROCESSOR DESIGN**

### I. LAB OBJECTIVES

### This Lab experiments are intended to design and test a Multi-Cycle Microprocessor

### II. DESCRIPTION

### Multi Cycle Microprocessor datapath to be implemented is in figure 2.1.

### 

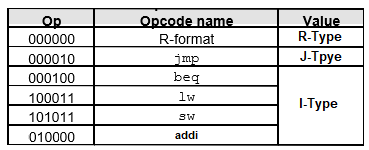
### Figure 2.1: Multi-cyclye Cycle Microprocessor DataPath

### III. LAB PROCEDURE

### III.1 EXPERIMENT NO. 1

##### III.1.1 AIM: To understand and write the assembly codes using MIPS Instruction

##### Instruction Operation codes:

****

Instruction Formats:

**Timeline

Description automatically generated**

**Table

Description automatically generated**

Register names and orders:



Assume the Assembly code code start from address PC=0x00000000, one instruction is store in one memory location.

**Testing Assembly Program 1:**

Instruction Meaning

Begin: addi $s2, $zero, 0x55 // load immediate value 0x55 to register $s2

addi $s3, $zero, 0x22 // load immediate value 0x22 to register $s3

addi $s5, $zero, 0x77 // load immediate value 0x77 to register $s5

add $s4, $s2, $s3 // $s4 = $s2 + $s3 => R20=0x77   
sub $s1, $s2, $s3 // $s1 = $s2 – $s3 => R17=0x22   
sw $s1, 0x02($s2) // Memory[$s2+0x02] = $s1

lw $s6, 0x02($s2) // $s6 = Memory[$s2+0x02]

bne $s5, $s4, End // Next instr. is at End if $s5 != $s4

addi $s8, $zero, 0x10 // load immediate value 10 to register $s8  
beq $s5,$s4, End // Next instr. is at End if $s7 == $s4

addi $s8, $zero, 0x20 // load immediate value 20 to register $s8

End: j End // jump End

**Testing Assembly Program 2:**

Instruction Meaning

Begin: addi $s2, $zero, 0x55 // load immediate value 0x55 to register $s2

addi $s3, $zero, 0x22 // load immediate value 0x22 to register $s3

addi $s5, $zero, 0x77 // load immediate value 0x77 to register $s5

add $s4, $s2, $s3 // $s4 = $s2 + $s3 => R20=0x77   
sub $s1, $s2, $s3 // $s1 = $s2 – $s3 => R17=0x22   
sw $s1, 0x02($s2) // Memory[$s2+0x02] = $s1

lw $s6, 0x02($s2) // $s6 = Memory[$s2+0x02]

beq $s5,$s4, End // Next instr. is at End if $s7 == $s4

addi $s8, $zero, 0x10 // load immediate value 10 to register $s8

bne $s5, $s4, End // Next instr. is at End if $s5 != $s4

addi $s8, $zero, 0x20 // load immediate value 20 to register $s8

End: j End // jump End

**III.1.2 LAB ASSIGNMENT**1) Compile the Assembly **Testing Assembly Program 1** into machine code (decimal code and binary code)

2) What is the value of Register $s8 after running **Testing Assembly Program 1**  program

3) Compile the Assembly **Testing Assembly Program 2** into machine code (decimal code and binary code)

4) What is the value of Register $s8 after running **Testing Assembly Program 2**  program

### III.2 EXPERIMENT NO. 2

##### III.2.1 AIM: To implement Verilog code to test ALL the Components of Multi-Cycle processor

##### 

**III.2.2 CODE**

//Your code

**III.2.3 LAB ASSIGNMENT**1) Write the Verilog modules for all required components in Datapath of the Multi-Cycle Processor Design

2) Write testbenches to verify all the components of the multi-cycle processor, simulate and check the simulation output data.

### III.3 EXPERIMENT NO. 3

##### III.3.1 AIM: To Write Verilog code to implement the complete Datapath of Multi-Cycle processor

##### 

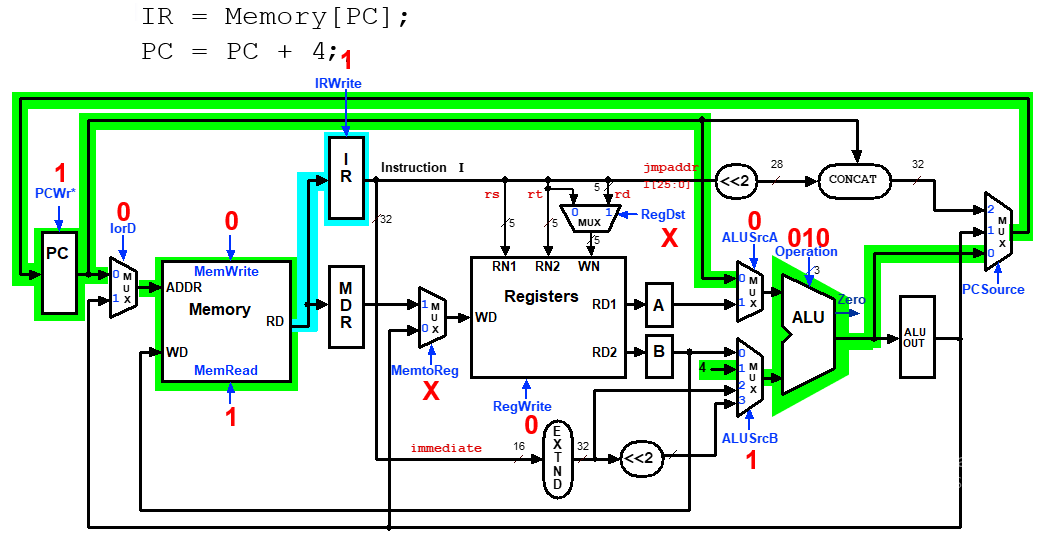
**III.3.2 CODE**

module Datapath\_Multi\_cycle\_Processor()

endmodule

**III.3.3 LAB ASSIGNMENT**1) Write Verilog code to implement complete Datapath Multi-Cycle Processor module

2) Write testbenches to verify the Datapath operation of the Multi-Cycle Processor in Multicycle Control Step 1 (Instruction Fetch : **IF**) (S0)

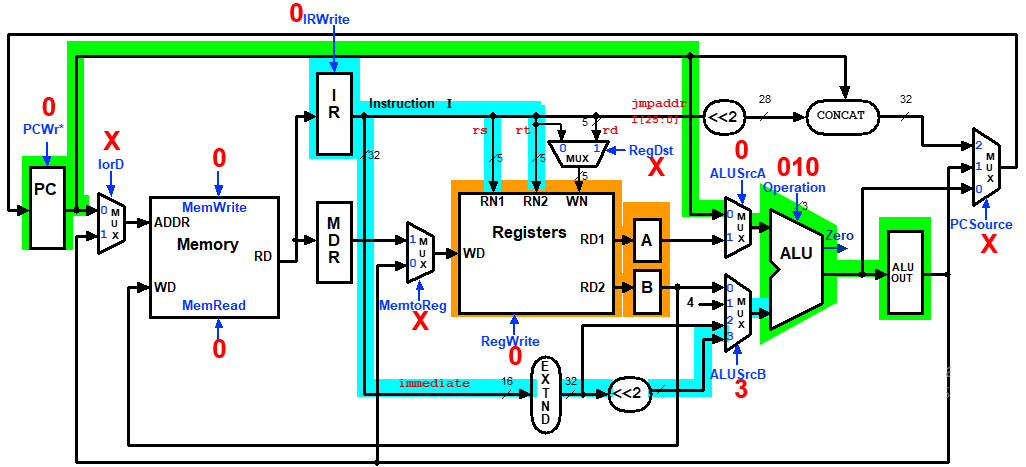


3) Modify the testbenches to verify the Datapath operation of the Multi-Cycle Processor in Multicycle Control Step-2 (Instruction Decode & Register Fetch : **ID**)

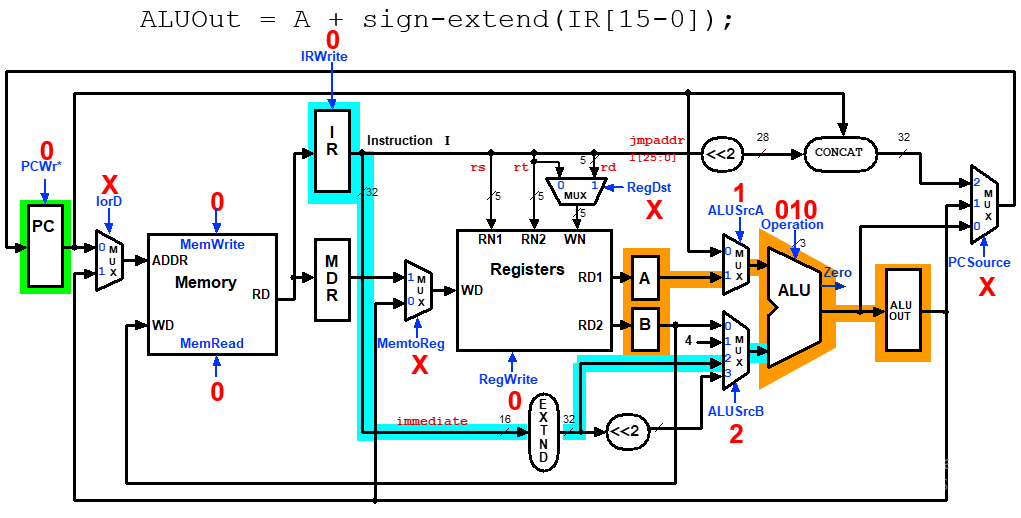
A = Reg[IR[25-21]]; (A = Reg[rs])

B = Reg[IR[20-15]]; (B = Reg[rt])

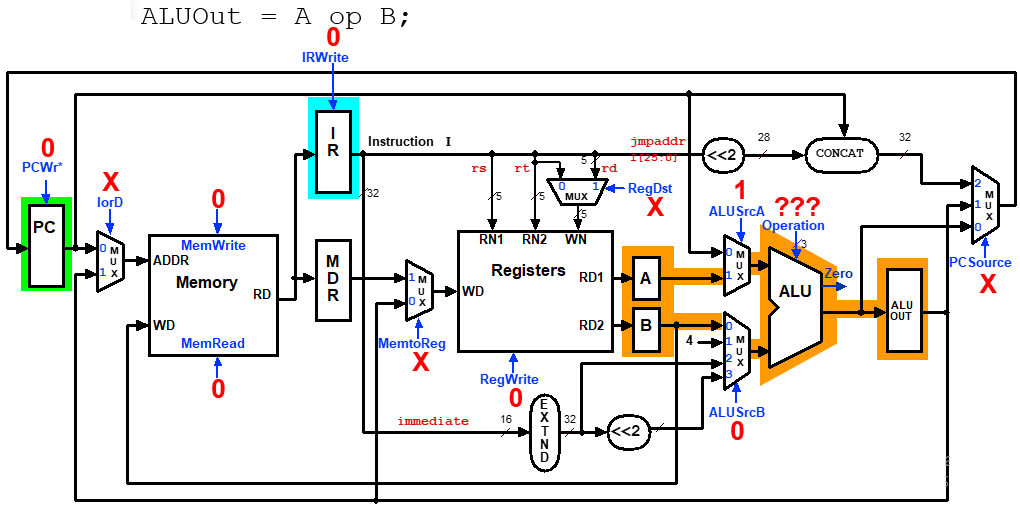
ALUOut = (PC + sign-extend(IR[15-0]) << 2);



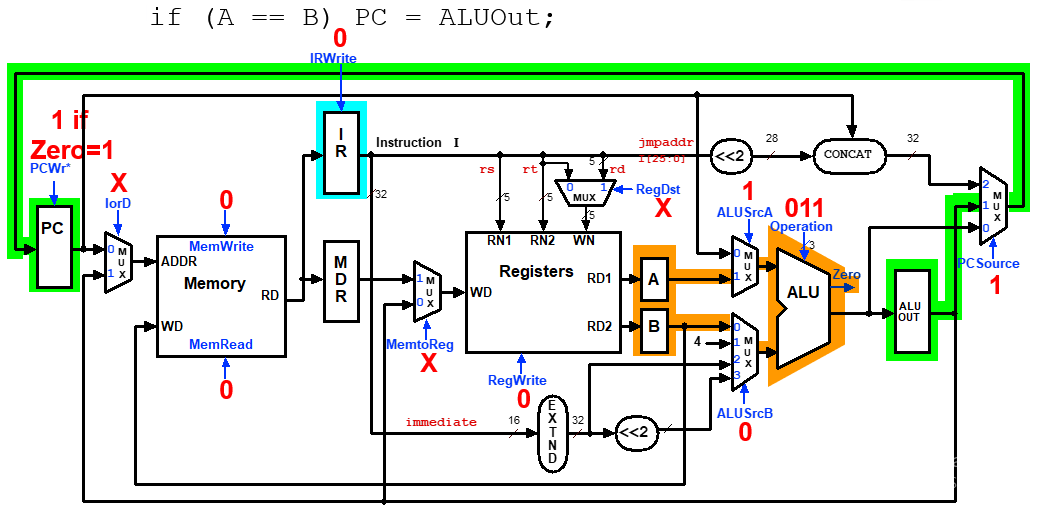
4) Modify the testbenches to verify the Datapath operation of the Multi-Cycle Processor in Multicycle Control Step-3 (Memory Reference Instructions : **MEM**)



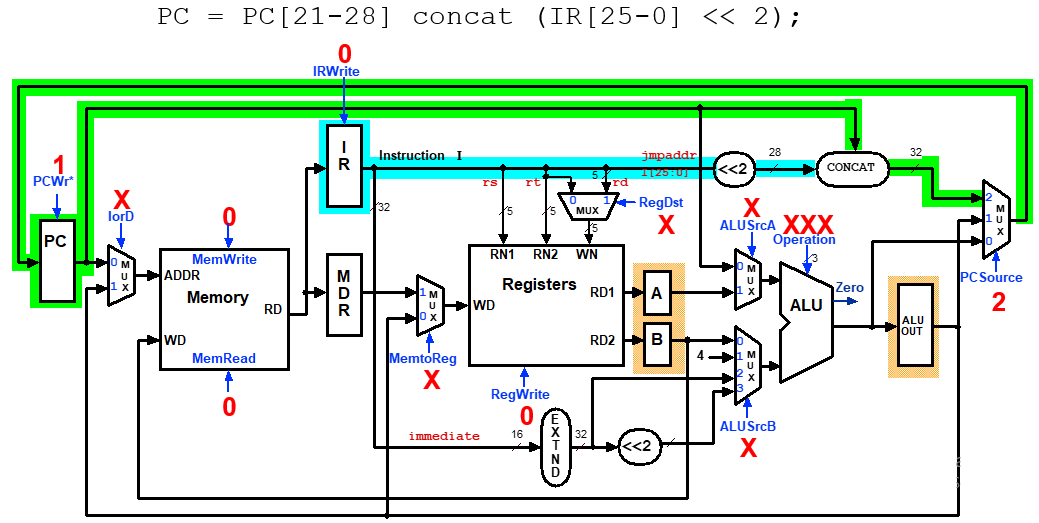
5) Modify the testbenches to verify the Datapath operation of the Multi-Cycle Processor in Multicycle Control Step-3 (ALU Instruction (R-Type))



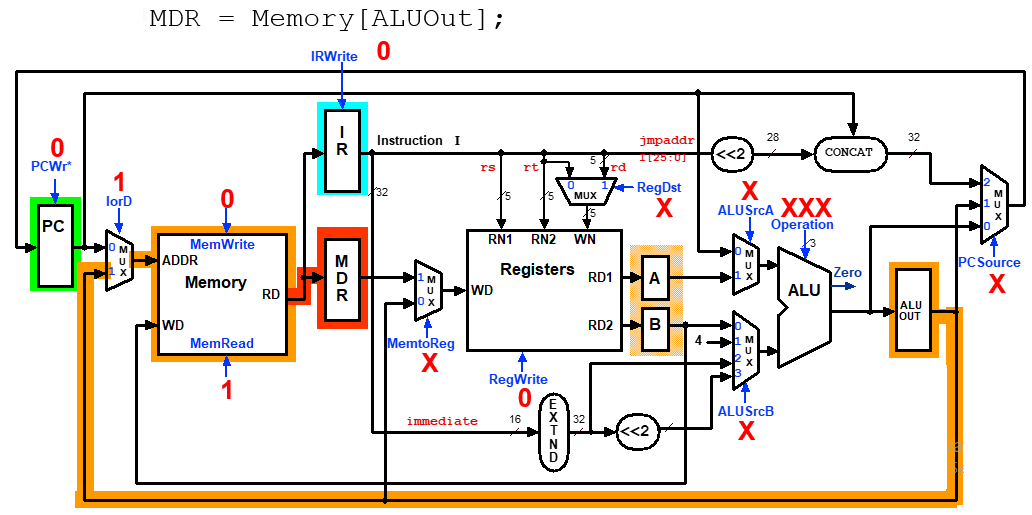
6) Modify the testbenches to verify the Datapath operation of the Multi-Cycle Processor in Multicycle Control Step (3)- Branch Instructions



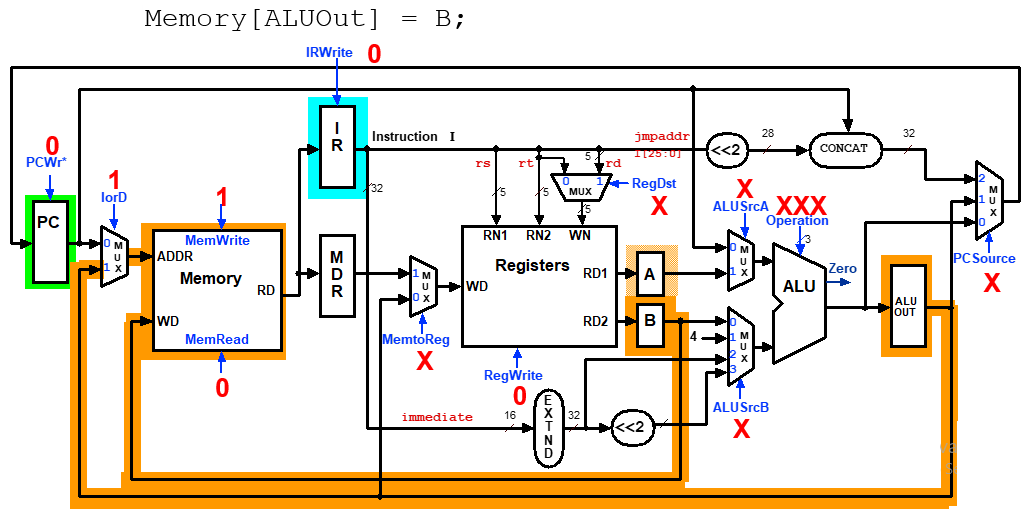
7) Modify the testbenches to verify the Datapath operation of the Multi-Cycle Processor in Multicycle Execution Step (3) - Jump Instruction



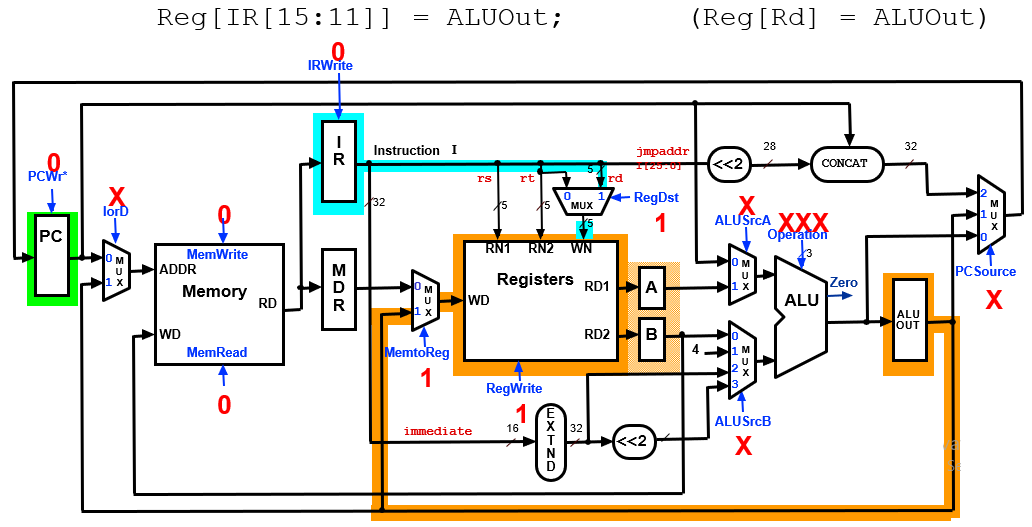
8) Modify the testbenches to verify the Datapath operation of the Multi-Cycle Processor in Multicycle Control Step (4)- Memory Access - Read (lw instruction)



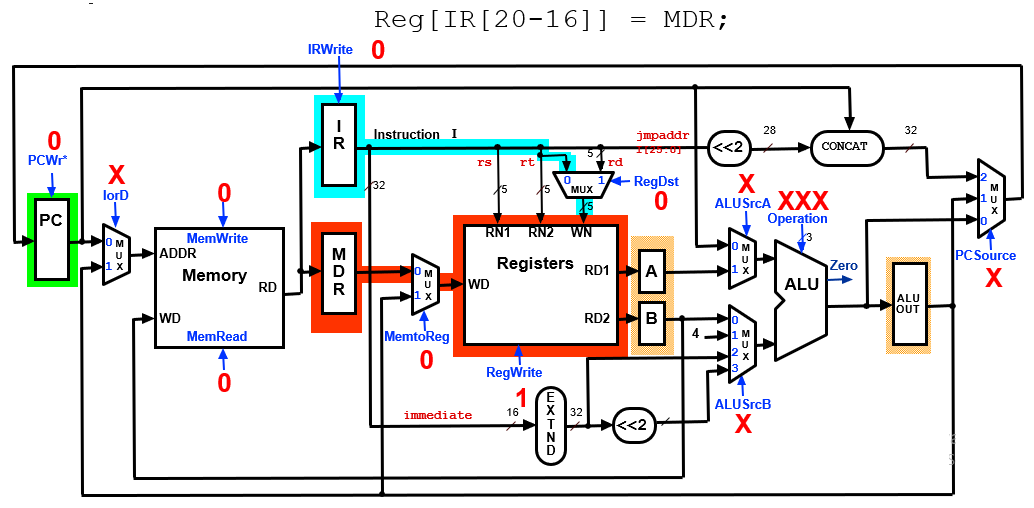
9) Modify the testbenches to verify the Datapath operation of the Multi-Cycle Processor in Multicycle Execution Steps (4) Memory Access - Write (sw)



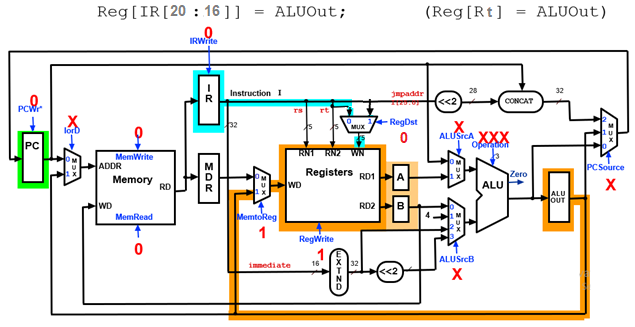
10) Modify the testbenches to verify the Datapath operation of the Multi-Cycle Processor in Multicycle Control Step (4): ALU Instruction (R-Type) and Addi



11) Modify the testbenches to verify the Datapath operation of the Multi-Cycle Processor inMulticycle Execution Steps (5) Memory Read Completion (lw)



12) Modify the testbenches to verify the Datapath operation of the Multi-Cycle Processor in Multicycle Control Step (4): ALU Instruction (I-Type) and ADDI



### III.4 EXPERIMENT NO. 4

##### III.4.1 AIM: To Write Verilog code to implement the complete Multi-Cycle processor.

##### Table Description automatically generated

**III.4.2 CODE**

module Control ( );

endmodule

module Multicycle\_Processor ( );

endmodule

**III.4.3 LAB ASSIGNMENT**

1) Write the Verilog code to implement the Microprocessor Control module using FSM with the following State graph:

Diagram

Description automatically generated

2) Write the ALU control module with the following input and output truth table:

**Table

Description automatically generated**

3) Write Verilog code to implement complete Multi-Cycle Processor module

Diagram, schematic

Description automatically generated

4) Write following Assembly code and compile into binary machine code to verify R-Type Instruction execution and Datapath operation of the complete Multi-Cycle Processor, write the testbench simulate and check the simulation output data.

add $s4, $s2, $s3 // $s4 = $s2 + $s3   
sub $s1, $s2, $s3 // $s1 = $s2 – $s3

and $s4, $s2, $s3 // $s4 = $s2 + $s3   
or $s1, $s2, $s3 // $s1 = $s2 – $s3

5) Write following Assembly code and compile into binary machine code to verify SW and LW Instruction execution and Datapath operation of the complete Multi-Cycle Processor, write the testbench simulate and check the simulation output data.

sw $s1, 0x02($s2) // Memory[$s2+0x02] = $s1

lw $s6, 0x02($s2) // $s6 = Memory[$s2+0x02]

6) Write following Assembly code and compile into binary machine code to verify beq and bne Instruction execution and Datapath operation of the complete Multi-Cycle Processor, write the testbench simulate and check the simulation output data.

bne $s5, $s4, End // Next instr. is at End if $s5 != $s4

add $s4, $s2, $s3//   
beq $s5,$s4, End // Next instr. is at End if $s7 == $s4

add $s4, $s2, $s3

End:

7) Compile the following code into binary machine code and store in Instruction memory to test the complete Multi-Cycle Complete Processor.

**Testing Assembly Program 1:**

Instruction Meaning

Begin: addi $s2, $zero, 0x55 // load immediate value 0x55 to register $s2 R18

addi $s3, $zero, 0x22 // load immediate value 0x22 to register $s3 R19

addi $s5, $zero, 0x77 // load immediate value 0x77 to register $s5 R21

add $s4, $s2, $s3 // $s4 = $s2 + $s3 => R20=R18+R19 = 0x77   
sub $s1, $s2, $s3 // $s1 = $s2 – $s3 => R17=R18-R19=0x33   
sw $s1, 0x02($s2) // Memory[0x55+0x02] = $s1 0x33 => RAM[0x57]

lw $s6, 0x02($s2) // $s6 = Memory[$s2+0x02]

bne $s5, $s4, End // Next instr. is at End if $s5 != $s4

addi $s8, $zero, 0x10 // load immediate value 10 to register $s8  
beq $s5,$s4, End // Next instr. is at End if $s7 == $s4

addi $s8, $zero, 0x20 // load immediate value 20 to register $s8

End: j End // jump End

8) Compile the following code into binary machine code and store in Instruction memory to test the Complete Multi-Cycle Processor.

**Testing Assembly Program 2:**

Instruction Meaning

Begin: addi $s2, $zero, 0x55 // load immediate value 0x55 to register $s2

addi $s3, $zero, 0x22 // load immediate value 0x22 to register $s3

addi $s5, $zero, 0x77 // load immediate value 0x77 to register $s5

add $s4, $s2, $s3 // $s4 = $s2 + $s3 => R20=0x77   
sub $s1, $s2, $s3 // $s1 = $s2 – $s3 => R17=0x22   
sw $s1, 0x02($s2) // Memory[$s2+0x02] = $s1

lw $s6, 0x02($s2) // $s6 = Memory[$s2+0x02]

beq $s5,$s4, End // Next instr. is at End if $s7 == $s4

addi $s8, $zero, 0x10 // load immediate value 10 to register $s8

bne $s5, $s4, End // Next instr. is at End if $s5 != $s4

addi $s8, $zero, 0x20 // load immediate value 20 to register $s8

End: j End // jump End

9) Write the assembly code to carry following calculation formula

Sum = 1+2+3+ …..9;

Compile the following code into binary machine code and store in Instruction memory to test the Complete Multi-cycle Processor.

**III. LAB REPORT GUIDELINES**

Students write up a report on the Verilog HDL implementation experiment projects created in this lab. The lab report should include Assembly Testing code, Verilog code for the module under test, Verilog test bench code and a truth table results, and example data input and output to validate the experiment. Simulation Result in form of Simulation Capture Screen. Analyzing the Calculation.