DIGITAL SYSTEM DESIGN LABORATORY

LAB 7 REPORT

MULTI-CYCLE MICROPROCESSOR DESIGN

NAME: Bùi Gia Bảo
ID: ITITIU22019

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.

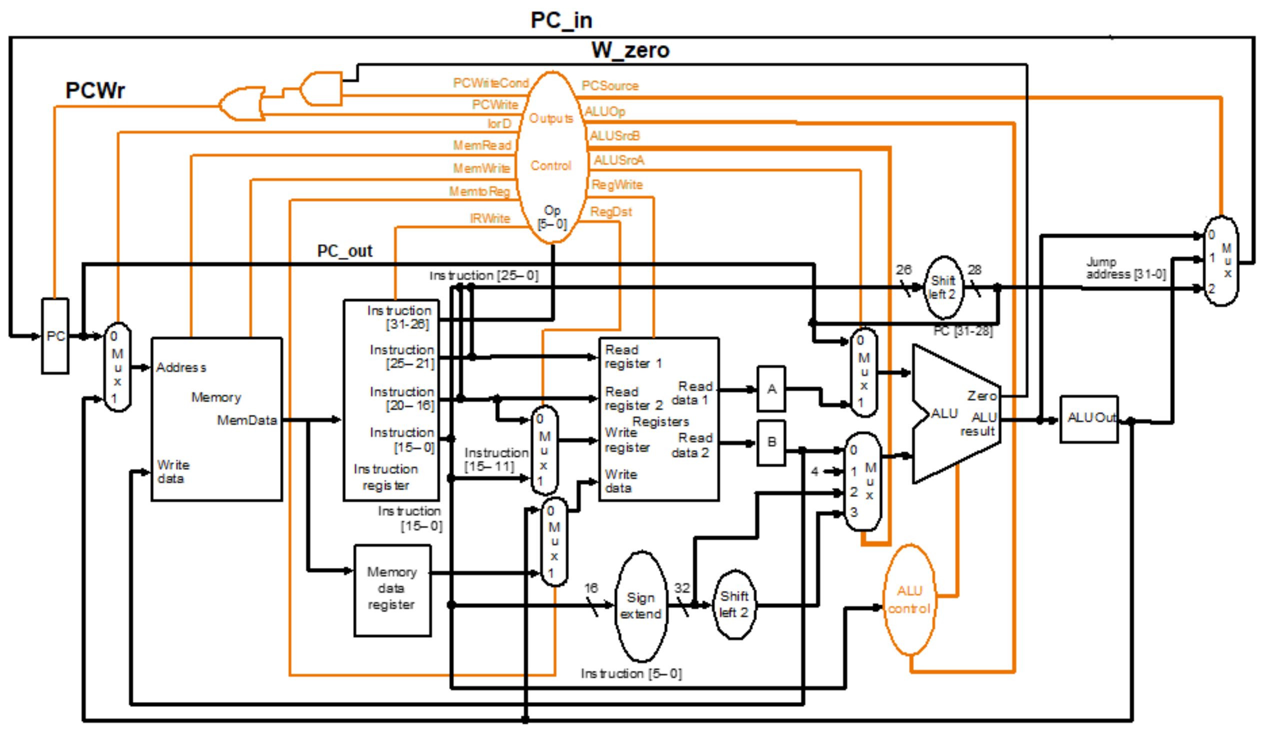


Figure 2.1: Multi-cyclye Cycle Microprocessor DataPath

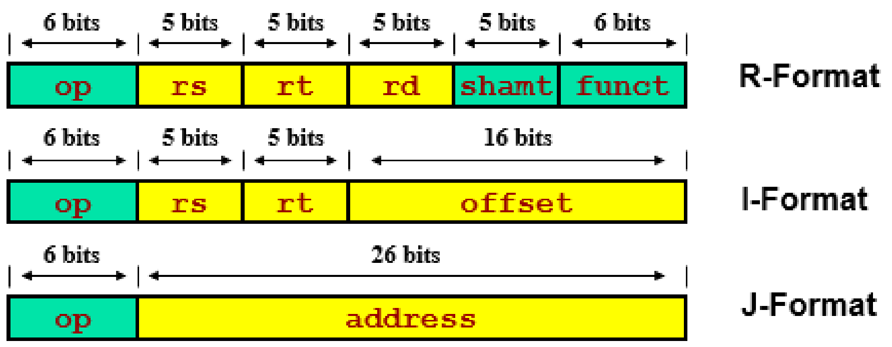
III. LAB PROCEDURE

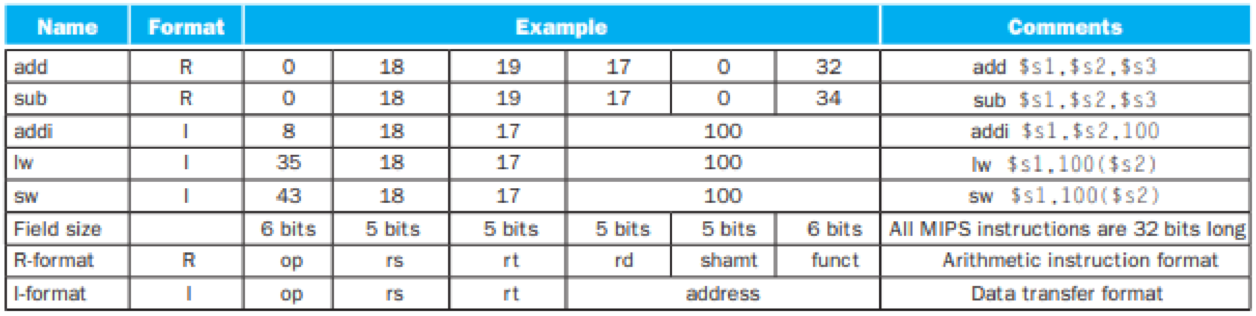
III.1 EXPERIMENT NO. 1

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

Op	Opcode name	Value
000000	R-format	R-Type
000010	jmp	J-Tpye
000100	beq	I-Type
100011	lw	
101011	sw	
010000	addi	

Instruction Formats:





Name	Format	Example						Comments
add	R	0	18	19	17	0	32	add $s1,$s2,$s3
sub	R	0	18	19	17	0	34	sub $s1,$s2,$s3
addi	I	8	18	17	100			addi $s1,$s2,100
lw	I	35	18	17	100			lw $s1,100($s2)
sw	I	43	18	17	100			sw $s1,100($s2)
Field size		6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	All MIPS instructions are 32 bits long
R-format	R	op	rs	rt	rd	shamt	funct	Arithmetic instruction format
I-format	I	op	rs	rt	address			Data transfer format

Register names and orders:

Name	Register number	Usage
`$zero`	0	the constant value 0
`$v0-$v1`	2-3	values for results and expression evaluation
`$a0-$a3`	4-7	arguments
`$t0-$t7`	8-15	temporaries
`$s0-$s7`	16-23	saved (by callee)
`$t8-$t9`	24-25	more temporaries
`$gp`	28	global pointer
`$sp`	29	stack pointer
`$fp`	30	frame pointer
`$ra`	31	return address

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)

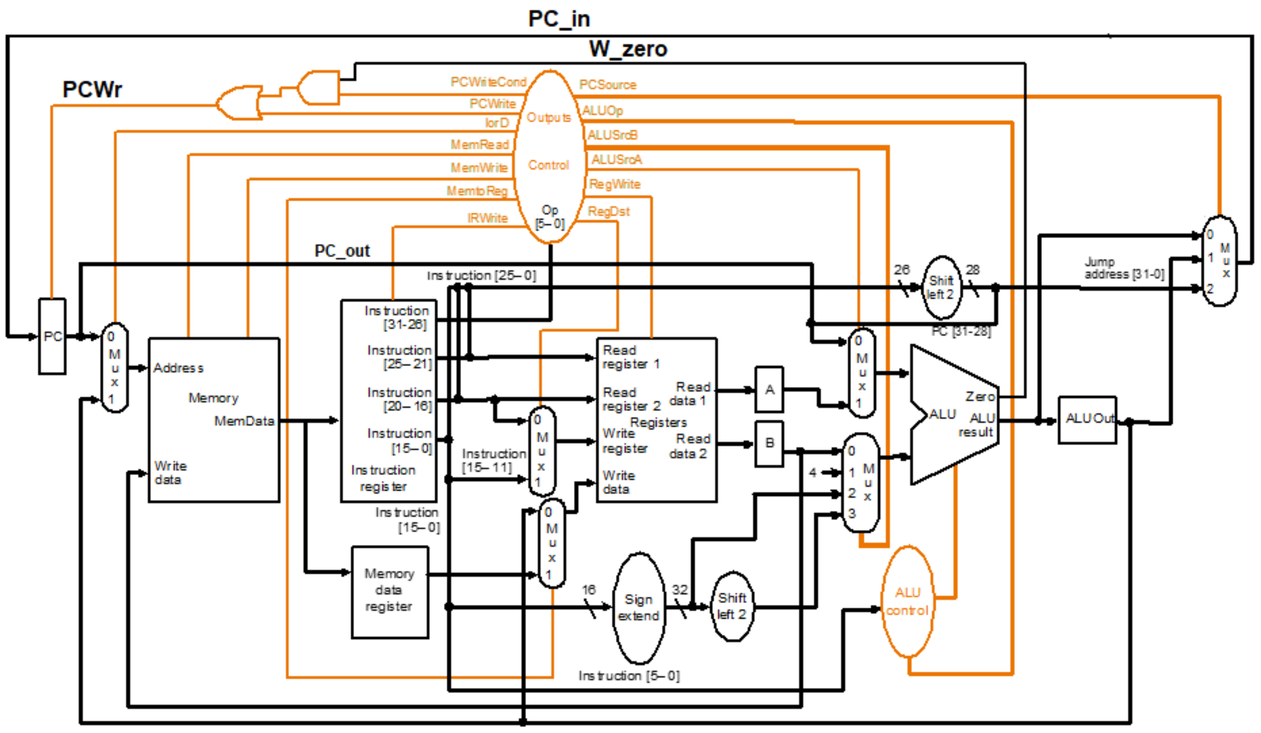
2) What is the value of Register $s8 after running **Testing Assembly Program 1** program

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.4 EXPERIMENT NO. 4

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

Op	Opcode name	Value
000000	R-format	R-Type
000010	jmp	J-Tpye
000100	beq	I-Type
100011	lw	
101011	sw	
010000	addi	

III.4.2 CODE

```
module lab7(SW,LEDG,LEDR,
HEX0,HEX1,HEX2,HEX3,HEX4,HEX5,HEX6,HEX7);
        input  [17:0] SW;
        output[17:0] LEDR;
        output[7:0] LEDG;
```

```
	output [0:6] HEX0,HEX1,HEX2,HEX3,HEX4,HEX5,HEX6,HEX7;
	wire [31:0] w_hex;




	Datapath_Multi_cycle_Processor dut(.clk(SW[17]),
.in_reset(SW[16]),.instruction(w_hex), .state(LEDR[3:0]));


	HEX_7SEG_DECODE H0(.BIN(w_hex[3:0]), .SSD(HEX0));
	HEX_7SEG_DECODE H1(.BIN(w_hex[7:4]), .SSD(HEX1));
	HEX_7SEG_DECODE H2(.BIN(w_hex[11:8]), .SSD(HEX2));
	HEX_7SEG_DECODE H3(.BIN(w_hex[15:12]), .SSD(HEX3));
	HEX_7SEG_DECODE H4(.BIN(w_hex[19:16]), .SSD(HEX4));
	HEX_7SEG_DECODE H5(.BIN(w_hex[23:20]), .SSD(HEX5));
	HEX_7SEG_DECODE H6(.BIN(w_hex[27:24]), .SSD(HEX6));
	HEX_7SEG_DECODE H7(.BIN(w_hex[31:28]), .SSD(HEX7));

endmodule

module HEX_7SEG_DECODE(BIN, SSD);
  input [3:0] BIN;
  output reg [0:6] SSD;
  always begin
    case(BIN)
      0:SSD=7'b0000001;
      1:SSD=7'b1001111;
      2:SSD=7'b0010010;
      3:SSD=7'b0000110;
      4:SSD=7'b1001100;
      5:SSD=7'b0100100;
      6:SSD=7'b0100000;
      7:SSD=7'b0001111;
      8:SSD=7'b0000000;
      9:SSD=7'b0001100;
      10:SSD=7'b0001000;
      11:SSD=7'b1100000;
      12:SSD=7'b0110001;
      13:SSD=7'b1000010;
      14:SSD=7'b0110000;
      15:SSD=7'b0111000;
    endcase
  end
endmodule


module  Datapath_Multi_cycle_Processor(clk, in_reset, instruction, state);
          input clk;
          input in_reset;
          output[3:0] state;
          wire reset;


          wire[31:0] pc_in;
          wire[31:0] pc_out;

          wire[31:0] alu_out;

          wire[31:0] addr_in;
          wire[31:0] Mem_Read_data;

          output[31:0] instruction;

          wire[31:0] MDR_out;

          wire[31:0] ALU_out_hold;
          wire[31:0] mux_2_out;

          wire[31:0] W_RD1, W_RD2;

          wire[31:0] Extend_out;
          wire[31:0] Branch_addr;
          wire[31:0] B_data,A_data;
          wire[31:0] ALU_in_A;
          wire[31:0] ALU_in_B;
          wire[31:0] ALU_out;

          wire [27:0] jump_28_bit;

          wire pc_write;
```

```
        wire IorD;
        wire MemRead;
        wire MemWrite;
        wire IRwrite;
        wire MemtoReg;
        wire RegDst;
        wire RegWrite;
        wire ALUSrcA;
        wire ALUSrcB;
        wire zero;
        wire PCWrite;
        wire PCWrcond;

        wire[4:0] mux_3_out;
        wire and_out;


        Program_Counter c1(.clk(clk), .reset(reset), .PC_write(pc_write)
,.PC_in(pc_in) , .PC_out(pc_out));

        Mux_32_bit c2(.in0(pc_in), .in1(alu_out), .mux_out(addr_in),
.select(IorD));

        Data_Memory c3(.clk(clk),.addr(addr_in), .write_data(B_data),
.read_data(Mem_Read_data), .MemRead(MemRead), .MemWrite(MemWrite));

        holding_reg c4(.output_data(instruction), .input_data(Mem_Read_data),
.write(IRwrite), .clk(clk), .reset(reset));

        holding_reg c5(.output_data(MDR_out), .input_data(Mem_Read_data),
.write(1'b1), .clk(clk), .reset(reset));


        Mux_32_bit c6(.in0(ALU_out_hold), .in1(MDR_out),
.mux_out(mux_2_out), .select(MemtoReg));

   Mux_5_bit c7(.in0(instruction[20:16]), .in1(instruction[15:11]),
.mux_out(mux_3_out), .select(RegDst));

   Register_File c8(        .clk(clk),
                                          .read_addr_1(instruction[25:21]),
                                          .read_addr_2(instruction[20:16]),
                                          .write_addr(mux_3_out),
                                          .read_data_1(W_RD1),
                                          .read_data_2(W_RD2),
                                          .write_data(mux_2_out),
                                          .RegWrite(RegWrite));

        Sign_Extension c9(.sign_in(instruction[15:0]), .sign_out(Extend_out));

        shift_left_2 c10(.sign_in(Extend_out), .sign_out(Branch_addr));

        holding_reg c11(.output_data(A_data), .input_data(W_RD1), .write(1'b1),
.clk(clk), .reset(reset));

        holding_reg c12(.output_data(B_data), .input_data(W_RD2), .write(1'b1),
.clk(clk), .reset(reset));

        Mux_32_bit c13(.in0(pc_out), .in1(A_data), .mux_out(ALU_in_A),
.select(ALUSrcA));

        Mux4_32_bit c14(.in0(B_data), .in1(32'd4) ,.in2(Extend_out),
.in3(Branch_addr), .mux_out(ALU_in_B), .select(ALUSrcB));

        alu c15(.alufn(Operation_ALU),
              .ra(ALU_in_A),
              .rb_or_imm(ALU_in_B),
              .aluout(ALU_out),
              .zero(zero));

        ALU_Control c16(.Op_intstruct(instruction[5:0])
,.ints_function(Operation_ALU), .ALUOp(ALUop));

        shift_left_2_28bit c17(.sign_in(instruction[25:0]), .sign_out(jump_28_bit));

        holding_reg c18(.output_data(ALU_out_hold), .input_data(ALU_out),
.write(1'b1), .clk(clk), .reset(reset));

        concate
c19(.PC_in(pc_out[31:28]),.IR_in(jump_28_bit),.PC_out(Jump_addr));
```

```
        Mux4_32_bit c20(.in0(ALU_out), .in1(ALU_out_hold) ,.in2(Jump_addr),
.in3(32'b0), .mux_out(pc_in), .select(PCSource));

        controller c21(.in_reset(in_reset)
        ,.opcode(instruction[31:26])
        ,.reset(reset)
        ,.clk(clk)
        ,.PCWrite(PCWrite)
        ,.Iord(Iord)
        ,.MemRead(MemRead)
        ,.MemWrite(MemWrite)
        ,.IRwrite(IRwrite)
        ,.MemtoReg(MemtoReg)
        ,.RegWrite(RegWrite)
        ,.RegDst(RegDst)
        ,.ALUSrcA(ALUSrcA)
        ,.ALUSrcB(ALUSrcB)
        ,.PCSource(PCSource)
        ,.ALUop(ALUop)
        ,.PCWrcond(PCWrcond)
        ,.state(state));

        and  c22(and_out,zero,PCWrcond);
    or  c23(pc_write,and_out,PCWrite);

endmodule

module controller(in_reset,opcode,
reset,clk,PCWrite,Iord,MemRead,MemWrite,IRwrite,MemtoReg,RegWrite,RegD
st,ALUSrcA,ALUSrcB,PCSource,ALUop,PCWrcond,state);

  // ~~~~~~~~~~~~~~~~~~~ PORTS ~~~~~~~~~~~~~~~~~~~ //

  // opcode, clock, and reset inputs
  input [5:0] opcode;        // from instruction register
  inputclk,in_reset;

  // control signal outputs
  output reg
PCWrite,Iord,MemRead,MemWrite,IRwrite,MemtoReg,RegWrite,RegDst,ALUSr
cA;
```

```
output reg [1:0] ALUSrcB,PCSource;
output reg [2:0] ALUop;
output reg PCWrcond;
output reg reset;
// ~~~~~~~~~~~~~~~~~~~~ REGISTER ~~~~~~~~~~~~~~~~~~~~ //

// 4-bit state register
output reg [3:0]    state;

// ~~~~~~~~~~~~~~~~~~~~ PARAMETERS ~~~~~~~~~~~~~~~~~~~~ //

// state parameters
parameter s0  = 4'd0;
parameter s1  = 4'd1;
parameter s2  = 4'd2;
parameter s3  = 4'd3;
parameter s4  = 4'd4;
parameter s5  = 4'd5;
parameter s6  = 4'd6;
parameter s7  = 4'd7;
parameter s8  = 4'd8;
parameter s9  = 4'd9;
parameter s10  = 4'd10;
parameter s_Reset  = 4'd11;     // reset



// opcode[5:4] parameters
parameter J       = 6'b000010;   // Jump or NOP
parameter R       = 6'b000000;  // R-type
parameter BEQ     = 6'b000100;        // Branch
parameter BNE     = 6'b000101;    // Branch
parameter SW      = 6'b101011;          // I-type
parameter LW      = 6'b100011;     // I-type
parameter ADDI    = 6'b001000;    // I-type


// OP code control for ALU

parameter OP_R_TYPE  = 3'b000;
parameter OP_I_TYPE  = 3'b001;
```

```
  parameter OP_J_TYPE  = 3'b010;
  parameter OP_BR_TYPE = 3'b011;
  parameter OP_IF_TYPE = 3'b100;
  parameter OP_ID_TYPE = 3'b101;
  parameter OP_RS_TYPE = 3'b110;


  // ~~~~~~~~~~~~~~~~~~~~ STATE MACHINE ~~~~~~~~~~~~~~~~~~~~ //

  // control state machine
  always @(posedge clk or posedge in_reset)
  begin

    // check for reset signal. If set, write zero to PC and switch to Reset State on
next CC.
    if (in_reset) begin
      PCWrite=0;
      Iord=0;
      MemRead=1;
      MemWrite=0;
      IRwrite=1;
      MemtoReg=0;
      RegWrite=0;
      RegDst=0;
      ALUSrcA=0;
      ALUSrcB=2'b01;
      PCSource=2'b00;
      ALUop=OP_RS_TYPE;
      PCWrcond=0;
      reset =1;
      state <= s_Reset;
    end
    else
    begin       // if reset signal is not set, check state at pos edge
      case (state)
       s_Reset:
           begin
             PCWrite=0;
             Iord=0;
             MemRead=1;
             MemWrite=0;
```

```
            IRwrite=1;
            MemtoReg=0;
            RegWrite=0;
            RegDst=0;
            ALUSrcA=0;
            ALUSrcB=2'b01;
            PCSource=2'b00;
            ALUop=OP_RS_TYPE;
            PCWrcond=0;
            reset =0;
            state <= s0;
            $display("state Reset");
          end
    s0:
       begin
          Iord=0;
          MemRead=1;
          MemWrite=0;
          IRwrite=1;
          MemtoReg=0;
          RegWrite=0;
          RegDst=0;
          ALUSrcA=0;
          ALUSrcB=2'b01;
          PCSource=2'b00;
          ALUop=OP_IF_TYPE;
          PCWrcond=0;
          state <= s1;
          PCWrite=1;
          $display("state 0");
       end
    s1:
       begin
          PCWrite=0;
          Iord=0;
          MemRead=0;
          MemWrite=0;
          IRwrite=0;
          MemtoReg=0;
          RegWrite=0;
          RegDst=0;
```

```
        ALUSrcA=0;
        ALUSrcB=2'b11;
        PCSource=2'b00;
        ALUop=OP_ID_TYPE;
        PCWrcond=0;
        $display("state 1");
        case(opcode[5:0])
            J:  state <= s9;
            R:  state <= s6;
            SW:  state <= s2;
                  LW:  state <= s2;
                  ADDI: state <= s2;
            BEQ: state <= s8;
        endcase
      end

    s2:
      begin
        PCWrite=0;
        Iord=1;
        MemRead=1;
        MemWrite=0;
        IRwrite=0;
        MemtoReg=0;
        RegWrite=0;
        RegDst=0;
        ALUSrcA=1;
        ALUSrcB=2'b10;
        PCSource=2'b00;
        ALUop=OP_I_TYPE;
        PCWrcond=0;
        $display("state 2");
        if(opcode[5:0]== ADDI)
          begin
            state <= s10;
            $display("ADDI state");
          end
        else if(opcode[5:0]== SW)
             begin
               state <= s5;
               $display("SW state");
```

```
                end
            else
                begin
                  state <= s3;
                  $display("SW state");
                end
        $display("state 2");
      end
    s3:
      begin
        PCWrite=0;
        Iord=1;
        MemRead=1;
        MemWrite=0;
        IRwrite=0;
        MemtoReg=0;
        RegWrite=0;
        RegDst=0;
        ALUSrcA=1;
        ALUSrcB=2'b10;
        PCSource=2'b00;
        ALUop=OP_I_TYPE;
        PCWrcond=0;
        state <= s4;
        $display("state 3");
      end
    s4:
      begin
        PCWrite=0;
        Iord=1;
        MemRead=0;
        MemWrite=0;
        IRwrite=0;
        MemtoReg=0;
        RegWrite=1;
        RegDst=0;
        ALUSrcA=0;
        ALUSrcB=2'b10;
        PCSource=2'b00;
        ALUop=OP_I_TYPE;
        PCWrcond=0;
        state <= s0;
        $display("state 4");
      end
    s5:
      begin
        PCWrite=0;
        Iord=1;
        MemRead=0;
        MemWrite=1;
        IRwrite=0;
        MemtoReg=0;
        RegWrite=0;
        RegDst=0;
        ALUSrcA=0;
        ALUSrcB=2'b10;
        PCSource=2'b00;
        ALUop=OP_I_TYPE;
        PCWrcond=0;
        state <= s0;
        $display("state 5");
      end

    s6:
      begin
        PCWrite=0;
        Iord=0;
        MemRead=0;
        MemWrite=0;
        IRwrite=0;
        MemtoReg=0;
        RegWrite=0;
        RegDst=0;
        ALUSrcA=1;
        ALUSrcB=2'b00;
        PCSource=2'b00;
        ALUop=OP_R_TYPE;
        PCWrcond=0;
        state <= s7;
        $display("state 6");
      end
```

```
s7:
  begin
    PCWrite=0;
    Iord=0;
    MemRead=0;
    MemWrite=0;
    IRwrite=0;
    MemtoReg=1;
    RegWrite=1;
    RegDst=1;
    ALUSrcA=1;
    ALUSrcB=2'b00;
    PCSource=2'b00;
    ALUop=OP_R_TYPE;
    PCWrcond=0;
    state <= s0;
    $display("state 7");
  end

s8:
  begin
    PCWrite=0;
    Iord=0;
    MemRead=0;
    MemWrite=0;
    IRwrite=0;
    MemtoReg=0;
    RegWrite=0;
    RegDst=0;
    ALUSrcA=1;
    ALUSrcB=2'b00;
    PCSource=2'b01;
    ALUop=OP_BR_TYPE;
    PCWrcond=1;
    state <= s0;
    $display("state 8");
  end

s9:
  begin
    PCWrite=1;
```

```
        Iord=0;
        MemRead=0;
        MemWrite=0;
        IRwrite=0;
        MemtoReg=0;
        RegWrite=0;
        RegDst=0;
        ALUSrcA=1;
        ALUSrcB=2'b00;
        PCSource=2'b10;
        ALUop=OP_J_TYPE;
        PCWrcond=0;
        state <= s0;
        $display("state 9");
      end
    s10:
      begin
        PCWrite=0;
        Iord=0;
        MemRead=0;
        MemWrite=0;
        IRwrite=0;
        MemtoReg=1;
        RegWrite=1;
        RegDst=0;
        ALUSrcA=1;
        ALUSrcB=2'b00;
        PCSource=2'b00;
        ALUop=OP_R_TYPE;
        PCWrcond=0;
        state <= s0;
        $display("state 7");
      end
     default: begin
        PCWrite=0;
        Iord=0;
        MemRead=0;
        MemWrite=0;
        IRwrite=0;
        MemtoReg=0;
        RegWrite=0;
```

```
            RegDst=0;
            ALUSrcA=0;
            ALUSrcB=2'b01;
            PCSource=2'b00;
            ALUop=OP_RS_TYPE;
            PCWrcond=0;
            $display("state default control");
          state <= s_Reset;
        end
      endcase
    end
  end
endmodule

module ALU_Control(Op_intstruct,ints_function,ALUOp);
    input [5:0] ints_function;
    input [2:0] Op_intstruct;
    output reg [2:0] ALUOp;
      // OP code control for ALU

  parameter OP_R_TYPE  = 3'b000;
  parameter OP_I_TYPE  = 3'b001;
  parameter OP_J_TYPE  = 3'b010;
  parameter OP_BR_TYPE = 3'b011;
  parameter OP_IF_TYPE = 3'b100;
  parameter OP_ID_TYPE = 3'b101;
  parameter OP_RS_TYPE = 3'b110;

    always @(*)
    begin
        case(Op_intstruct)
            OP_R_TYPE:   // R -Type Instruction look at fuction
                begin
                    ALUOp =3'b000;
                    if(ints_function==6'b100000) // add
                        begin
                            ALUOp =3'b000;
                            $display("fuction Add");
                        end

                    if(ints_function==6'b100010) // sub
          begin
             ALUOp =3'b001;
             $display("fuction sub");
          end

       if(ints_function==6'b100100) // and
          begin
             ALUOp =3'b010;
             $display("fuction and");
          end

       if(ints_function==6'b100101) // or
          begin
             ALUOp =3'b010;
             $display("fuction or ");
          end
    end
 OP_I_TYPE:
    begin
       ALUOp =3'b000;
       $display("LW or SW");
    end
 OP_J_TYPE:
    begin
       ALUOp =3'b000;
       $display("Jump");
    end
 OP_BR_TYPE: // beq instruction
    begin
       ALUOp =3'b111;
       $display("BEQ or BNE");
    end
 OP_IF_TYPE: // Store Instruction
    begin
       ALUOp =3'b000;
       $display("Add IF");
    end
 OP_ID_TYPE: // addi Instruction
    begin
       ALUOp =3'b000;
       $display("add ID");
```

```
            end
          OP_RS_TYPE:  //bne
            begin
              ALUOp =3'b111;
              $display("Reset OP");
            end
          default :
            begin
              ALUOp =3'b000;
                    $display("ALU default");
            end
        endcase
    end
endmodule
module Mux_5_bit (in0, in1, mux_out, select);
     parameter N = 5;
     input [N-1:0] in0, in1;
     output [N-1:0] mux_out;
     input select;
     assign mux_out = select? in1: in0 ;
endmodule

module Sign_Extension (sign_in, sign_out);
     input [15:0] sign_in;
     output [31:0] sign_out;
     assign sign_out[15:0]=sign_in[15:0];
     assign sign_out[31:16]=sign_in[15]?16'b1111_1111_1111_1111:16'b0;
endmodule

module Program_Counter (clk, reset,PC_write ,PC_in, PC_out);
     input clk, reset,PC_write;
     input [31:0] PC_in;
     output reg [31:0] PC_out;
     always @ (posedge clk or posedge reset)
     begin
          if(reset==1'b1)
               PC_out<=0;
          else if (PC_write==1'b1)
               PC_out<=PC_in;
     end
endmodule
```

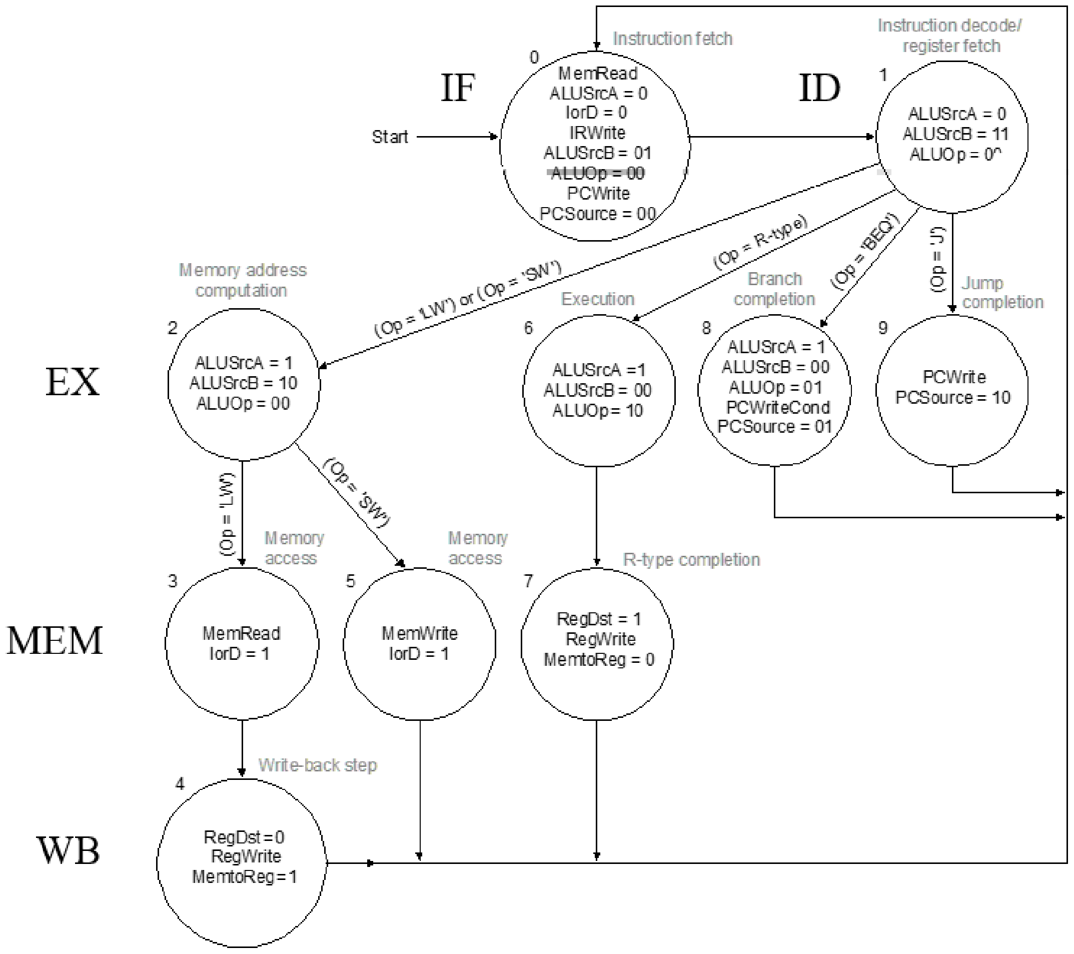
```
module alu(
        input [2:0] alufn,
        input [31:0] ra,
        input [31:0] rb_or_imm,
        output reg [31:0] aluout,
        output reg zero);
        parameter    ALU_OP_ADD      = 3'b000,
                        ALU_OP_SUB      = 3'b001,
                        ALU_OP_AND      = 3'b010,
                        ALU_OP_OR       = 3'b011,
                        ALU_OP_XOR      = 3'b100,
                        ALU_OP_LW       = 3'b101,
                        ALU_OP_SW       = 3'b110,
                        ALU_OP_BEQ      = 3'b111;

   always @(*)
      begin
                case(alufn)
                        ALU_OP_ADD      : aluout = ra + rb_or_imm;
                        ALU_OP_SUB      : aluout = ra - rb_or_imm;
                        ALU_OP_AND      : aluout = ra & rb_or_imm;
                        ALU_OP_OR       : aluout = ra | rb_or_imm;
                        ALU_OP_XOR      : aluout = ra ^ rb_or_imm;
                        ALU_OP_LW       : aluout = ra + rb_or_imm;
                        ALU_OP_SW       : aluout = ra + rb_or_imm;
                        ALU_OP_BEQ      : begin
                                        zero = (ra==rb_or_imm)?1'b1:1'b0;
                                        aluout = ra - rb_or_imm;
                                       end
                endcase
      end
endmodule

module Register_File (clk,read_addr_1, read_addr_2, write_addr, read_data_1,
read_data_2, write_data, RegWrite);
        input [4:0] read_addr_1, read_addr_2, write_addr;
        input [31:0] write_data;
        input  clk,RegWrite;
        reg checkRegWrite;
        output reg [31:0] read_data_1, read_data_2;
      reg [31:0] Regfile [31:0];
      integer k;
      initial
        begin
          for (k=0; k<32; k=k+1)
                  begin
                       Regfile[k] = 32'd10;
                  end
                Regfile[8]=32'd1;
                Regfile[9]=32'd2;
                Regfile[10]=32'd3; //$t2
                Regfile[11]=32'd4; //$t3


                Regfile[17]=32'd99;
                Regfile[18]=32'd60;
                Regfile[19]=32'd30;
        end

      //assign read_data_1 = Regfile[read_addr_1];
     always @(read_data_1 or Regfile[read_addr_1])
          begin
           if (read_addr_1 == 0) read_data_1 = 0;
           else
           begin
           read_data_1 = Regfile[read_addr_1];

//$display("read_addr_1=%d,read_data_1=%h",read_addr_1,read_data_1);
           end
          end
      //assign read_data_2 = Regfile[read_addr_2];
     always @(read_data_2 or Regfile[read_addr_2])
          begin
           if (read_addr_2 == 0) read_data_2 = 0;
           else
           begin
           read_data_2 = Regfile[read_addr_2];

//$display("read_addr_2=%d,read_data_2=%h",read_addr_2,read_data_2);
           end
          end
        always @(posedge clk)
            begin
                if (RegWrite == 1'b1)
                  begin
                    Regfile[write_addr] = write_data;
                    $display("Rigister File write_addr=%d
write_data=%d",write_addr,write_data);
                  end
            end
endmodule

module holding_reg (
    output reg [31:0] output_data,
    input      [31:0] input_data,
    input            write,
    input            clk,
    input            reset
);

always @(posedge clk or posedge reset) begin
    if (reset)
        output_data <= 32'b0;
    else if (write)
        output_data <= input_data;
end

endmodule

module Mux_32_bit (in0, in1, mux_out, select);
        parameter N = 32;
        input [N-1:0] in0, in1;
        output [N-1:0] mux_out;
        input select;
        assign mux_out = select? in1: in0 ;
endmodule

module shift_left_2 (sign_in, sign_out);
        input [31:0] sign_in;
        output [31:0] sign_out;
        assign sign_out[31:2]=sign_in[29:0];
        assign sign_out[1:0]=2'b00;
```

```
endmodule

module concate(PC_in,IR_in,PC_out);
   input [3:0] PC_in;
   input [27:0] IR_in;
   output[31:0] PC_out;
   assign PC_out={PC_in, IR_in};
endmodule

module Mux4_32_bit (in0, in1,in2, in3, mux_out, select);
        parameter N = 32;
        input [N-1:0] in0, in1,in2,in3;
        output [N-1:0] mux_out;
        input [1:0]select;
        assign mux_out = select[1]? (select[0]?in3: in2):(select[0]?in1:in0);
endmodule

module shift_left_2_28bit (sign_in, sign_out);
        input [25:0] sign_in;
        output [27:0] sign_out;
        assign sign_out={2'b00,sign_in};
endmodule

module Data_Memory (clk,addr, write_data, read_data, MemRead, MemWrite);
   input [31:0] addr;
   input [31:0] write_data;
   output [31:0] read_data;
   input MemRead, MemWrite,clk;
   reg [31:0] DMemory [63:0];
   integer k;
   initial begin
      for (k=0; k<64; k=k+1)
         begin
            DMemory[k] = 32'b0;
         end
      //sw  $s1, 0x02($s2)     //    Memory[$s2+0x02] = $s1
      DMemory[0] = 32'b10101110010100010000000000000010;

      //add $s4,  $s2, $s3       //    $s4 = $s2 + $s3  => R20=0x90
      DMemory[4] = 32'b00000010010100111010000000100000;
```

```
    //add $s5 $t0 $t1
    DMemory[8] = 32'b00000001000010011010100000100000;


    //sub $s1, $s2, $s3        //    $s1 = $s2 – $s3  => R17=0x22
    DMemory[12] = 32'b00000010010100111000100000100010;

    //sw  $s1, 0x02($s2)    //    Memory[$s2+0x02] = $s1
    DMemory[16] = 32'b10101110010100010000000000000010;


    //lw $s1, 0x02($s2)         //$s1 = Memory[$s2+0x02]
    DMemory[20] = 32'b10001110010100010000000000000010;


    //beq $t2,$t3, End      //beq $t2,$t3, 0x03
    DMemory[24] = 32'b00010001010010110000000000000011;

    //addi $s7, $zero, 0x10
    DMemory[28] = 32'b00100000000101110000000000010000;
         //j 0x00
    DMemory[32] = 32'b00001000000000000000000000000000;
    //addi $s2, $zero, 0x55 //  load immediate value 0x55 to register $s2
    DMemory[36] = 32'b00100000000100100000000001010101;
    //addi $s3, $zero, 0x22 //  load immediate value 0x22 to register $s3
    DMemory[40] = 32'b00100000000100110000000000100010;
    //addi $s5, $zero, 0x77 //  load immediate value 0x77 to register $s5
    DMemory[44] = 32'b00100000000101010000000001110111;
    end

  assign read_data = (MemRead) ? DMemory[addr] : 32'bx;

  always @(posedge clk)
    begin
      if (MemWrite)
      begin
        DMemory[addr] = write_data;
        $display("Data memory write_addr=%d
write_data=%d",addr,write_data);
      end
    end
endmodule
```

III.4.3 LAB ASSIGNMENT

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

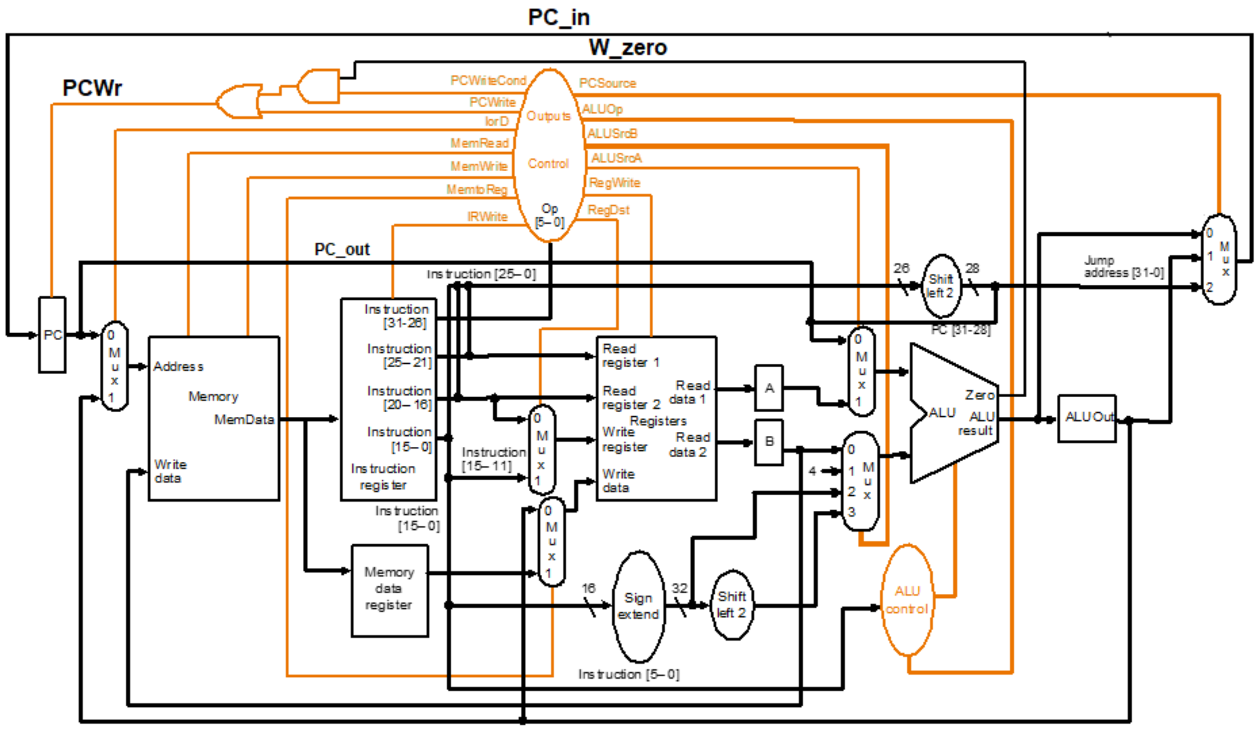


2) Write the ALU control module with the following input and output truth table:

$ALUOp_{1:0}$	Meaning
00	Add
01	Subtract
10	Look at Funct
11	Not Used

$ALUOp_{1:0}$	Funct	$ALUControl_{2:0}$
00	X	010 (Add)
X1	X	110 (Subtract)
1X	100000 (add)	010 (Add)
1X	100010 (sub)	110 (Subtract)
1X	100100 (and)	000 (And)
1X	100101 (or)	001 (Or)
1X	101010 (slt)	111 (SLT)

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



4) Write following Assembly code and compile into binary machine code to verify 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.