ELEC0028 Advanced Digital Design Coursework

Part 1 SystemVerilog Description and Simulation of the RISC-V Microarchitecture

Task 1.1 Specifying control signals

Table 1 Table of RISC-V control signals

Inst	**RegWrite**	**ImmSrc**	**ALUSrc**	**ALUControl**	**MemWrite**	**ResultSrc**	**PCSrc**
add	**1**	**XXX**	**0**	**X0X10**	**0**	**01**	**00**
sub	**1**	**XXX**	**0**	**X1X10**	**0**	**01**	**00**
or	**1**	**XXX**	**0**	**XX111**	**0**	**01**	**00**
and	**1**	**XXX**	**0**	**XX011**	**0**	**01**	**00**
sll	**1**	**XXX**	**0**	**0XX00**	**0**	**01**	**00**
srl	**1**	**XXX**	**0**	**1XX00**	**0**	**01**	**00**
slt	**1**	**XXX**	**0**	**X1X01**	**0**	**01**	**00**
addi	**1**	**000**	**1**	**X0X10**	**0**	**01**	**00**
ori	**1**	**000**	**1**	**XX111**	**0**	**01**	**00**
andi	**1**	**000**	**1**	**XX011**	**0**	**01**	**00**
slli	**1**	**000**	**1**	**0XX00**	**0**	**01**	**00**
srli	**1**	**000**	**1**	**1XX00**	**0**	**01**	**00**
slti	**1**	**000**	**1**	**X1X01**	**0**	**01**	**00**
lw	**1**	**000**	**1**	**X0X10**	**0**	**10**	**00**
sw	**0**	**001**	**1**	**X0X10**	**1**	**XX**	**00**
beq	**0**	**010**	**0**	**X1X10**	**0**	**XX**	**{0, Zero}**
bne	**0**	**010**	**0**	**X1X10**	**0**	**XX**	**{0, ~Zero}**
blt	**0**	**010**	**0**	**X1X10**	**0**	**XX**	**{0, Negative}**
bge	**0**	**010**	**0**	**X1X10**	**0**	**XX**	**{0, ~Negative}**
jal	**1**	**100**	**X**	**XXXXX**	**0**	**11**	**01**
jalr	**1**	**000**	**1**	**X0X10**	**0**	**11**	**10**
lui	**1**	**011**	**X**	**XXXXX**	**0**	**00**	**00**

Task 1.2 Writing SystemVerilog descriptions of the sub-blocks

Program_counter.sv code

```
module program_counter (output logic [31:0] PC, PCPlus4,
                        input logic [31:0] PCTarget, ALUResult,
                        input logic [1:0] PCSrc,
                        input logic Reset, CLK);

// Enter your code here

    assign PCPlus4 = PC + 4;

    // Synchronous PC update with reset
    always_ff @(posedge CLK or posedge Reset) begin
        if (Reset) begin
            PC <= 32'b0;  // Reset PC to 0 (start executing from
0x00000000)
        end else begin
            case (PCSrc)
                2'b00: PC <= PCPlus4;   // Sequential execution (PC +
4)
                2'b01: PC <= PCTarget;  // Branch target address
                2'b10: PC <= ALUResult; // JALR computed target
                default: PC <= PCPlus4; // Default case: fall back to
sequential
            endcase
        end
    end

endmodule
```

Control_unit.sv

```
module control_unit (output logic [1:0] PCSrc,
                     output logic [1:0] ResultSrc,
                     output logic MemWrite, ALUSrc, RegWrite,
                     output logic [4:0] ALUControl,
                     output logic [2:0] ImmSrc,
                     input logic [31:0] Instr,
                     input logic Zero, Negative);

// Enter your code here

    logic [6:0] opcode;
    logic [2:0] funct3;
    logic [6:0] funct7;
    assign opcode = Instr[6:0];   // Opcode:
    assign funct3 = Instr[14:12]; // Funct3:
    assign funct7 = Instr[31:25]; // Funct7: R

    // **1. Immsrc**
    always_comb begin
        case (opcode)
            7'b0000011: ImmSrc = 3'b000; // I (LW)
            7'b0010011: ImmSrc = 3'b000; // I (Arithmetic)
            7'b1100111: ImmSrc = 3'b000; // I (JALR)
            7'b0100011: ImmSrc = 3'b001; // S (SW)
            7'b1100011: ImmSrc = 3'b010; // B (Branch)
            7'b0110111: ImmSrc = 3'b011; // U (LUI)
            7'b1101111: ImmSrc = 3'b100; // J (JAL)
            default: ImmSrc = 3'b000;
        endcase
    end

    // **2. PCSrc**
    always_comb begin
        case (opcode)
            7'b1100011: // (BEQ, BNE, BLT, BGE)
                PCSrc = (funct3 == 3'b000 && Zero)  ? 2'b01 :  // BEQ
                        (funct3 == 3'b001 && !Zero) ? 2'b01 :  // BNE
                        (funct3 == 3'b100 && Negative) ? 2'b01 : // BLT
                        (funct3 == 3'b101 && !Negative) ? 2'b01 : //
BGE
                        2'b00;
            7'b1101111: PCSrc = 2'b01; // JAL
            7'b1100111: PCSrc = 2'b10; // JALR
            default: PCSrc = 2'b00;
        endcase
    end

    // **3. ResultSrc**
    always_comb begin
        case (opcode)
            7'b0000011: ResultSrc = 2'b10; // LW
            7'b1101111: ResultSrc = 2'b11; // JAL
            7'b1100111: ResultSrc = 2'b11; // JALR (PC+4)
            7'b0110111: ResultSrc = 2'b00; // LUI
            7'b0010011,
            7'b0110011: ResultSrc = 2'b01; // I & R
            default: ResultSrc = 2'b00;
        endcase
    end

    // **4. ALUSrc**
    assign ALUSrc = (opcode == 7'b0010011 ||  // I (Arithmetic)
                     opcode == 7'b0000011 ||  // I (LW)
                     opcode == 7'b0100011 ||  // S (SW)
                     opcode == 7'b1100111);   // I (JALR)

    // **5. MemWrite**
    assign MemWrite = (opcode == 7'b0100011); // SW

    // **6. RegWrite**
    assign RegWrite = (opcode == 7'b0110011 ||  // R
                       opcode == 7'b0010011 ||  // I
                       opcode == 7'b0000011 ||  // LW
                       opcode == 7'b1101111 ||  // JAL
                       opcode == 7'b1100111 ||  // JALR
                       opcode == 7'b0110111);   // LUI

    // **7. ALUControl**
    always_comb begin
        case (opcode)
            7'b0110011: begin // **R
                case ({funct7, funct3})
                    10'b0000000000: ALUControl = 5'b00110; // ADD
                    10'b0100000000: ALUControl = 5'b01010; // SUB
                    10'b0000000110: ALUControl = 5'b00111; // OR
                    10'b0000000111: ALUControl = 5'b00011; // AND
                    10'b0000000001: ALUControl = 5'b00000; // SLL
                    10'b0000000101: ALUControl = 5'b10000; // SRL
                    10'b0000000010: ALUControl = 5'b01001; // SLT
                    default: ALUControl = 5'b00000; //
                endcase
            end

            7'b0010011: begin // **I
                case (funct3)
                    3'b000: ALUControl = 5'b00110; // ADDI
                    3'b110: ALUControl = 5'b00111; // ORI
                    3'b111: ALUControl = 5'b00011; // ANDI
                    3'b001: ALUControl = 5'b00000; // SLLI
                    3'b101: ALUControl = 5'b10000; // SRLI
                    3'b010: ALUControl = 5'b01001; // SLTI
                    default: ALUControl = 5'b00000;
                endcase
            end

            7'b0000011, // **LW**
            7'b0100011, // **SW**
            7'b1100111: // **JALR**
                ALUControl = 5'b00110;
            7'b1100011: begin // **Branch**
                case (funct3)
                    3'b000, // BEQ
                    3'b001, // BNE
                    3'b100, // BLT
                    3'b101: // BGE
                        ALUControl = 5'b01010;
                    default: ALUControl = 5'b00000;
                endcase
            end

            7'b1101111: ALUControl = 5'b00000; // JAL
            7'b0110111: ALUControl = 5'b00000; // LUI

            default: ALUControl = 5'b00000;
        endcase
    end



endmodule
```

Extend.sv code:

```
module extend (output logic [31:0] ImmExt,
                input logic [31:0] Instr,
                input logic [2:0] ImmSrc);

// Enter your code here
    always_comb begin
        case (ImmSrc)
            3'b000: // I-type (e.g., LW, JALR, arithmetic immediates)
                ImmExt = {{20{Instr[31]}}, Instr[31:20]};
            3'b001: // S-type (e.g., SW)
                ImmExt = {{20{Instr[31]}}, Instr[31:25],
Instr[11:7]};
            3'b010: // B-type (e.g., BEQ, BNE, BLT, BGE)
                ImmExt = {{19{Instr[31]}}, Instr[31], Instr[7],
Instr[30:25], Instr[11:8], 1'b0};
            3'b011: // U-type (e.g., LUI, AUIPC)
                ImmExt = {Instr[31:12], 12'b0};
            3'b100: // J-type (e.g., JAL)
                ImmExt = {{11{Instr[31]}}, Instr[31], Instr[19:12],
Instr[20], Instr[30:21], 1'b0};
            default:
                ImmExt = 32'b0;
        endcase
    end
endmodule
```

alu.sv code

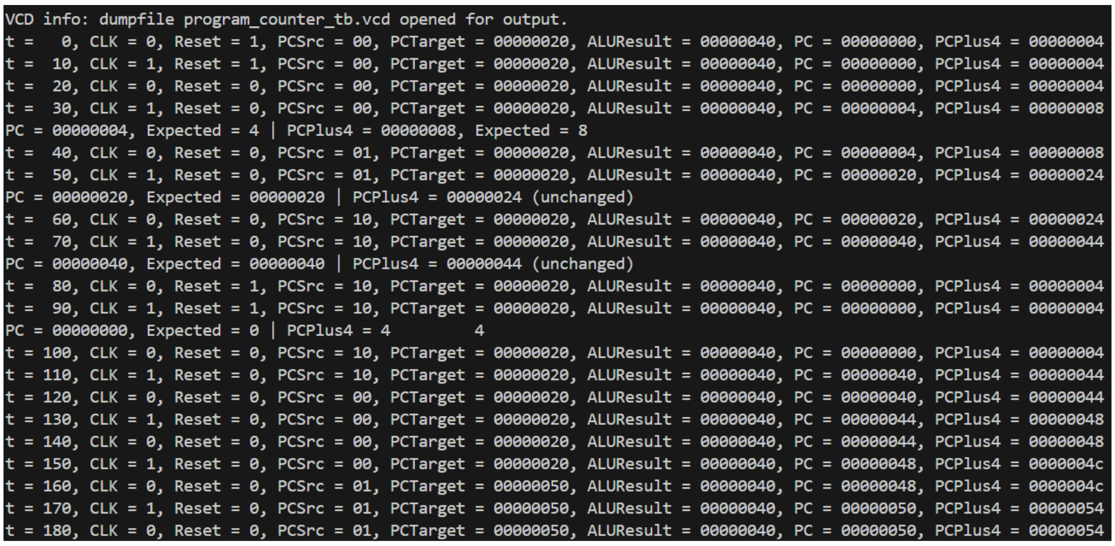
```
module alu (output logic signed [31:0] ALUResult,
            output logic Zero, Negative,
            input logic signed [31:0] SrcA, SrcB,
            input logic [4:0] ALUControl);

// Enter your code here

    always_comb begin

        case (ALUControl)
            5'b10110, 5'b10010, 5'b00010, 5'b00110: ALUResult = SrcA +
SrcB;
            5'b11110, 5'b11010, 5'b01010, 5'b01110: ALUResult = SrcA -
SrcB;
            5'b00111, 5'b01111, 5'b10111, 5'b11111: ALUResult = SrcA |
SrcB;
            5'b00011, 5'b01011, 5'b10011, 5'b11011: ALUResult = SrcA &
SrcB;

            5'b00000, 5'b00100, 5'b01000, 5'b01100: ALUResult = SrcA <<
SrcB[4:0];
            5'b10000, 5'b10100, 5'b11000, 5'b11100: ALUResult = SrcA >>
SrcB[4:0];

            5'b11001, 5'b11101, 5'b01101, 5'b01001: ALUResult = (SrcA <
SrcB) ? 32'b1 : 32'b0;

        endcase

        Zero = (ALUResult == 32'b0);
        Negative = ALUResult[31];
    end


endmodule
```

always @(*) begin is used first for alu module, after reinstalling for the latest version of Verilog, always_comb begin works as well.

Their corresponding testbench code is added in the appendix.

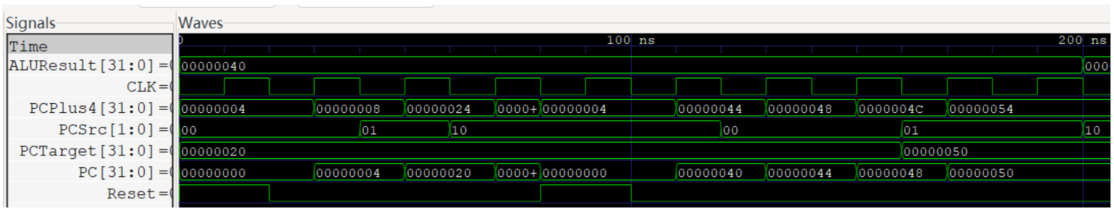
Task 1.3 Simulations of sub-blocks using testbenches

Program_counter_tb text results:



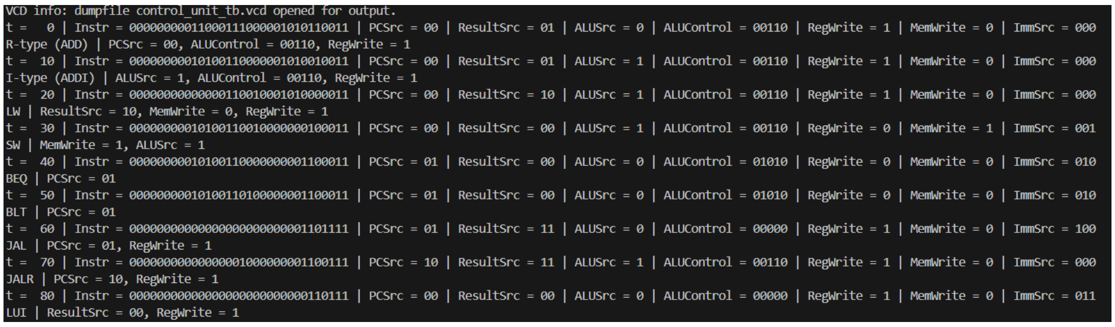
```
VCD info: dumpfile program_counter_tb.vcd opened for output.
t =   0, CLK = 0, Reset = 1, PCSrc = 00, PCTarget = 00000020, ALUResult = 00000040, PC = 00000000, PCPlus4 = 00000004
t =  10, CLK = 1, Reset = 1, PCSrc = 00, PCTarget = 00000020, ALUResult = 00000040, PC = 00000000, PCPlus4 = 00000004
t =  20, CLK = 0, Reset = 0, PCSrc = 00, PCTarget = 00000020, ALUResult = 00000040, PC = 00000000, PCPlus4 = 00000004
t =  30, CLK = 1, Reset = 0, PCSrc = 00, PCTarget = 00000020, ALUResult = 00000040, PC = 00000004, PCPlus4 = 00000008
PC = 00000004, Expected = 4 | PCPlus4 = 00000008, Expected = 8
t =  40, CLK = 0, Reset = 0, PCSrc = 01, PCTarget = 00000020, ALUResult = 00000040, PC = 00000004, PCPlus4 = 00000008
t =  50, CLK = 1, Reset = 0, PCSrc = 01, PCTarget = 00000020, ALUResult = 00000040, PC = 00000020, PCPlus4 = 00000024
PC = 00000020, Expected = 00000020 | PCPlus4 = 00000024 (unchanged)
t =  60, CLK = 0, Reset = 0, PCSrc = 10, PCTarget = 00000020, ALUResult = 00000040, PC = 00000020, PCPlus4 = 00000024
t =  70, CLK = 1, Reset = 0, PCSrc = 10, PCTarget = 00000020, ALUResult = 00000040, PC = 00000040, PCPlus4 = 00000044
PC = 00000040, Expected = 00000040 | PCPlus4 = 00000044 (unchanged)
t =  80, CLK = 0, Reset = 1, PCSrc = 10, PCTarget = 00000020, ALUResult = 00000040, PC = 00000000, PCPlus4 = 00000004
t =  90, CLK = 1, Reset = 1, PCSrc = 10, PCTarget = 00000020, ALUResult = 00000040, PC = 00000000, PCPlus4 = 00000004
PC = 00000000, Expected = 0 | PCPlus4 = 4         4
t = 100, CLK = 0, Reset = 0, PCSrc = 10, PCTarget = 00000020, ALUResult = 00000040, PC = 00000000, PCPlus4 = 00000004
t = 110, CLK = 1, Reset = 0, PCSrc = 10, PCTarget = 00000020, ALUResult = 00000040, PC = 00000040, PCPlus4 = 00000044
t = 120, CLK = 0, Reset = 0, PCSrc = 00, PCTarget = 00000020, ALUResult = 00000040, PC = 00000040, PCPlus4 = 00000044
t = 130, CLK = 1, Reset = 0, PCSrc = 00, PCTarget = 00000020, ALUResult = 00000040, PC = 00000044, PCPlus4 = 00000048
t = 140, CLK = 0, Reset = 0, PCSrc = 00, PCTarget = 00000020, ALUResult = 00000040, PC = 00000044, PCPlus4 = 00000048
t = 150, CLK = 1, Reset = 0, PCSrc = 00, PCTarget = 00000020, ALUResult = 00000040, PC = 00000048, PCPlus4 = 0000004c
t = 160, CLK = 0, Reset = 0, PCSrc = 01, PCTarget = 00000050, ALUResult = 00000040, PC = 00000048, PCPlus4 = 0000004c
t = 170, CLK = 1, Reset = 0, PCSrc = 01, PCTarget = 00000050, ALUResult = 00000040, PC = 00000050, PCPlus4 = 00000054
t = 180, CLK = 0, Reset = 0, PCSrc = 01, PCTarget = 00000050, ALUResult = 00000040, PC = 00000050, PCPlus4 = 00000054
```

Program_counter_tb graphical results:



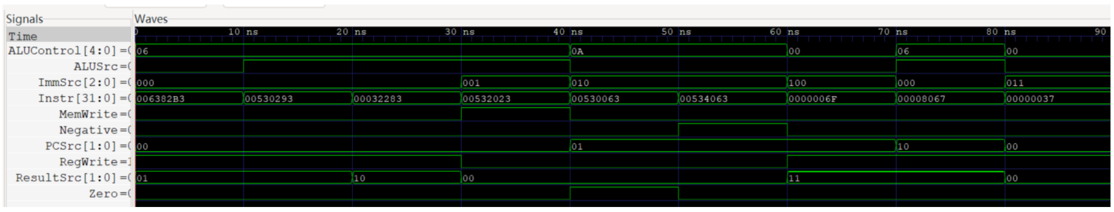
The PC will be written to 0x00000000 each time when the Reset is signal is pulled high. When the CLK signal is positive edged and the Reset is 0, the PC will be written depending on the value of PCSrc, which is followed by the microarchitecture provided: 00 for PCPlus4, 01 for PCTarget and 10 for ALUResult. This works well.

Control_unit_tb text results:



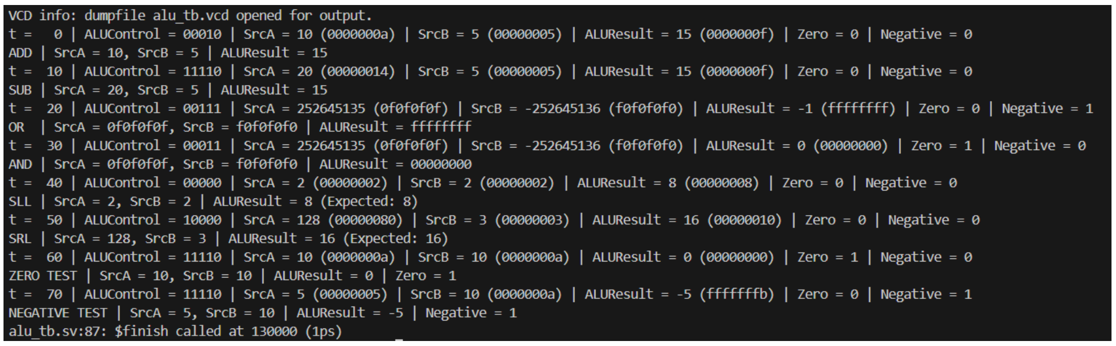
```
VCD info: dumpfile control_unit_tb.vcd opened for output.
t =   0 | Instr = 00000000011000111000001010110011 | PCSrc = 00 | ResultSrc = 01 | ALUSrc = 0 | ALUControl = 00110 | RegWrite = 1 | MemWrite = 0 | ImmSrc = 000
R-type (ADD) | PCSrc = 00, ALUControl = 00110, RegWrite = 1
t =  10 | Instr = 00000000010100110000001010010011 | PCSrc = 00 | ResultSrc = 01 | ALUSrc = 1 | ALUControl = 00110 | RegWrite = 1 | MemWrite = 0 | ImmSrc = 000
I-type (ADDI) | ALUSrc = 1, ALUControl = 00110, RegWrite = 1
t =  20 | Instr = 00000000000000110010001010000011 | PCSrc = 00 | ResultSrc = 10 | ALUSrc = 1 | ALUControl = 00110 | RegWrite = 1 | MemWrite = 0 | ImmSrc = 000
LW | ResultSrc = 10, MemWrite = 0, RegWrite = 1
t =  30 | Instr = 00000000010100110010000000100011 | PCSrc = 00 | ResultSrc = 00 | ALUSrc = 1 | ALUControl = 00110 | RegWrite = 0 | MemWrite = 1 | ImmSrc = 001
SW | MemWrite = 1, ALUSrc = 1
t =  40 | Instr = 00000000010100110000000001100011 | PCSrc = 01 | ResultSrc = 00 | ALUSrc = 0 | ALUControl = 01010 | RegWrite = 0 | MemWrite = 0 | ImmSrc = 010
BEQ | PCSrc = 01
t =  50 | Instr = 00000000010100110100000001100011 | PCSrc = 01 | ResultSrc = 00 | ALUSrc = 0 | ALUControl = 01010 | RegWrite = 0 | MemWrite = 0 | ImmSrc = 010
BLT | PCSrc = 01
t =  60 | Instr = 00000000000000000000000001101111 | PCSrc = 01 | ResultSrc = 11 | ALUSrc = 0 | ALUControl = 00000 | RegWrite = 1 | MemWrite = 0 | ImmSrc = 100
JAL | PCSrc = 01, RegWrite = 1
t =  70 | Instr = 00000000000000001000000001100111 | PCSrc = 10 | ResultSrc = 11 | ALUSrc = 1 | ALUControl = 00110 | RegWrite = 1 | MemWrite = 0 | ImmSrc = 000
JALR | PCSrc = 10, RegWrite = 1
t =  80 | Instr = 00000000000000000000000000110111 | PCSrc = 00 | ResultSrc = 00 | ALUSrc = 0 | ALUControl = 00000 | RegWrite = 1 | MemWrite = 0 | ImmSrc = 011
LUI | ResultSrc = 00, RegWrite = 1
```

Control_unit_tb graphical results:



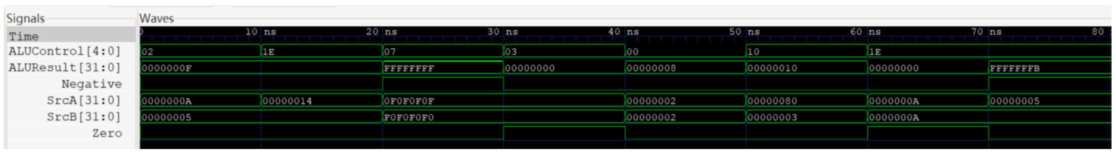
This control unit will be input a Instr from the instruction_memory file, which converts the machine codes in the txt file into a series of instructions. It will divide the Instr into several parts followed by the Instruction format and subset provided. Combined with the control signal I have written in the first task, this unit is basically linking all the control signals with the Instr, such as the opcode will be used to determine the instruction type and generate the ImmSrc. The control_unit_tb result is the same as the control signal table. This works well.

Alu_tb text results:



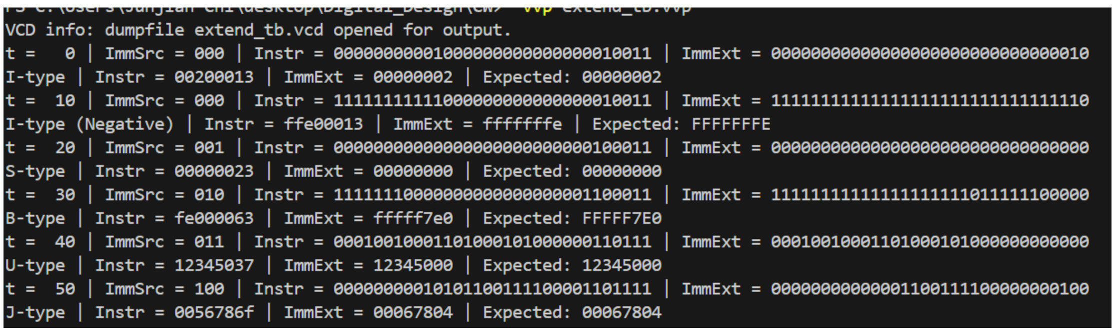
```
VCD info: dumpfile alu_tb.vcd opened for output.
t =   0 | ALUControl = 00010 | SrcA = 10 (0000000a) | SrcB = 5 (00000005) | ALUResult = 15 (0000000f) | Zero = 0 | Negative = 0
ADD | SrcA = 10, SrcB = 5 | ALUResult = 15
t =  10 | ALUControl = 11110 | SrcA = 20 (00000014) | SrcB = 5 (00000005) | ALUResult = 15 (0000000f) | Zero = 0 | Negative = 0
SUB | SrcA = 20, SrcB = 5 | ALUResult = 15
t =  20 | ALUControl = 00111 | SrcA = 252645135 (0f0f0f0f) | SrcB = -252645136 (f0f0f0f0) | ALUResult = -1 (ffffffff) | Zero = 0 | Negative = 1
OR  | SrcA = 0f0f0f0f, SrcB = f0f0f0f0 | ALUResult = ffffffff
t =  30 | ALUControl = 00011 | SrcA = 252645135 (0f0f0f0f) | SrcB = -252645136 (f0f0f0f0) | ALUResult = 0 (00000000) | Zero = 1 | Negative = 0
AND | SrcA = 0f0f0f0f, SrcB = f0f0f0f0 | ALUResult = 00000000
t =  40 | ALUControl = 00000 | SrcA = 2 (00000002) | SrcB = 2 (00000002) | ALUResult = 8 (00000008) | Zero = 0 | Negative = 0
SLL | SrcA = 2, SrcB = 2 | ALUResult = 8 (Expected: 8)
t =  50 | ALUControl = 10000 | SrcA = 128 (00000080) | SrcB = 3 (00000003) | ALUResult = 16 (00000010) | Zero = 0 | Negative = 0
SRL | SrcA = 128, SrcB = 3 | ALUResult = 16 (Expected: 16)
t =  60 | ALUControl = 11110 | SrcA = 10 (0000000a) | SrcB = 10 (0000000a) | ALUResult = 0 (00000000) | Zero = 1 | Negative = 0
ZERO TEST | SrcA = 10, SrcB = 10 | ALUResult = 0 | Zero = 1
t =  70 | ALUControl = 11110 | SrcA = 5 (00000005) | SrcB = 10 (0000000a) | ALUResult = -5 (fffffffb) | Zero = 0 | Negative = 1
NEGATIVE TEST | SrcA = 5, SrcB = 10 | ALUResult = -5 | Negative = 1
alu_tb.sv:87: $finish called at 130000 (1ps)
```

Alu_tb graphical results:



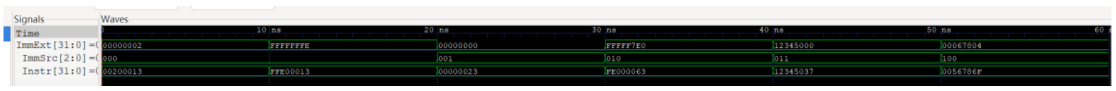
This alu module is using the ALUControl signal generated from the control unit to achieve the actual function, such as add is adding the values of two registers and put this into the ALUResult signal. The arithmetic operation is two’s complement as required. Though the negative is pulled high even for the OR operation, but it will not cause problem as we set the condition for the branch for both Negative and Func3. Therefore, this module works well.

Expand_tb text results:



```
VCD info: dumpfile extend_tb.vcd opened for output.
t =   0 | ImmSrc = 000 | Instr = 00000000001000000000000000010011 | ImmExt = 00000000000000000000000000000010
I-type | Instr = 00200013 | ImmExt = 00000002 | Expected: 00000002
t =  10 | ImmSrc = 000 | Instr = 11111111111000000000000000010011 | ImmExt = 11111111111111111111111111111110
I-type (Negative) | Instr = ffe00013 | ImmExt = fffffffe | Expected: FFFFFFFE
t =  20 | ImmSrc = 001 | Instr = 00000000000000000000000000100011 | ImmExt = 00000000000000000000000000000000
S-type | Instr = 00000023 | ImmExt = 00000000 | Expected: 00000000
t =  30 | ImmSrc = 010 | Instr = 11111110000000000000000001100011 | ImmExt = 11111111111111111111011111100000
B-type | Instr = fe000063 | ImmExt = fffff7e0 | Expected: FFFFF7E0
t =  40 | ImmSrc = 011 | Instr = 00010010001101000101000000110111 | ImmExt = 00010010001101000101000000000000
U-type | Instr = 12345037 | ImmExt = 12345000 | Expected: 12345000
t =  50 | ImmSrc = 100 | Instr = 00000000010101100111100001101111 | ImmExt = 00000000000001100111100000000100
J-type | Instr = 0056786f | ImmExt = 00067804 | Expected: 00067804
```

Expand_tb graphical results:



This module simply extracts the ImmExt based on the Instr and Immsrc as the same as the Table 3 provided. It works well.

Task 1.4 Completing top-level RISC-V SystemVerilog description

Risc_v. sv code:

```
`include "instruction_memory.sv"
`include "reg_file.sv"
`include "extend.sv"
`include "alu.sv"
`include "data_memory_and_io.sv"
`include "program_counter.sv"
`include "control_unit.sv"

module risc_v(output logic [31:0] CPUOut,
                input logic [31:0] CPUIn,
                input logic Reset, CLK);

logic [31:0] Instr, WD3, RD1, RD2, SrcA, SrcB, ALUResult, WD, RD,
Result, ImmExt, PCTarget, PCNext, PC, PCPlus4;
logic [4:0]  A1, A2, A3;
logic        MemWrite, ALUSrc, RegWrite;
logic        Zero, Negative;
logic [4:0]  ALUControl;
logic [2:0]  ImmSrc;
logic [1:0]  ResultSrc;
logic [1:0]  PCSrc;

// Enter your code here
assign PCTarget = PC + ImmExt;

program_counter program_counter_inst (
    .PC(PC),
    .PCPlus4(PCPlus4),
    .PCTarget(PCTarget),
    .ALUResult(ALUResult),
    .PCSrc(PCSrc),
    .Reset(Reset),
    .CLK(CLK)
);


instruction_memory instruction_memory_inst (
    .PC(PC),
    .Instr(Instr)
);


control_unit control_unit_inst (
    .PCSrc(PCSrc),
    .ResultSrc(ResultSrc),
    .MemWrite(MemWrite),
```

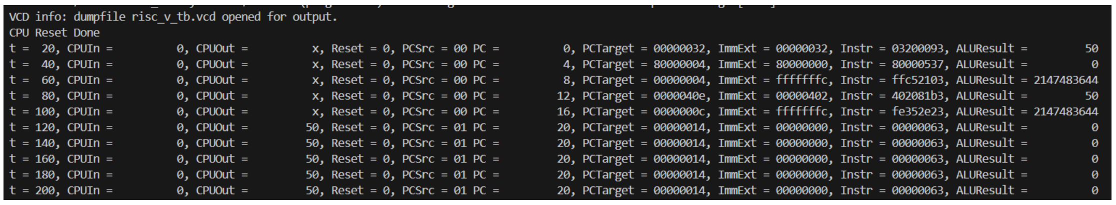
```verilog
    .ALUSrc(ALUSrc),
    .RegWrite(RegWrite),
    .ALUControl(ALUControl),
    .ImmSrc(ImmSrc),
    .Instr(Instr),
    .Zero(Zero),
    .Negative(Negative)
);


assign A1 = Instr[19:15]; // rs1
assign A2 = Instr[24:20]; // rs2
assign A3 = Instr[11:7];  // rd

reg_file reg_file_inst (
    .RD1(RD1),
    .RD2(RD2),
    .WD3(WD3),
    .A1(A1),
    .A2(A2),
    .A3(A3),
    .WE3(RegWrite),
    .CLK(CLK)
);


extend extend_inst (
    .Instr(Instr),
    .ImmSrc(ImmSrc),
    .ImmExt(ImmExt)
);


assign SrcA = RD1;
assign SrcB = (ALUSrc) ? ImmExt : RD2; // Select SrcB based on ALUSrc
control

alu alu_unit (
    .ALUResult(ALUResult),
    .Zero(Zero),
    .Negative(Negative),
    .SrcA(SrcA),
    .SrcB(SrcB),
    .ALUControl(ALUControl)
);

data_memory_and_io data_mem (
    .RD(RD),
    .CPUOut(CPUOut),
    .A(ALUResult),
    .WD(RD2),
    .CPUIn(CPUIn),
    .WE(MemWrite),
    .CLK(CLK)
);

always_comb begin
    case (ResultSrc)
        2'b00: Result = ImmExt;
        2'b01: Result = ALUResult;
        2'b10: Result = RD;
        2'b11: Result = PCPlus4;
        default: Result = 32'b0;
    endcase
end

assign WD3 = Result;

endmodule
```

Risc_v_tb.sv code:

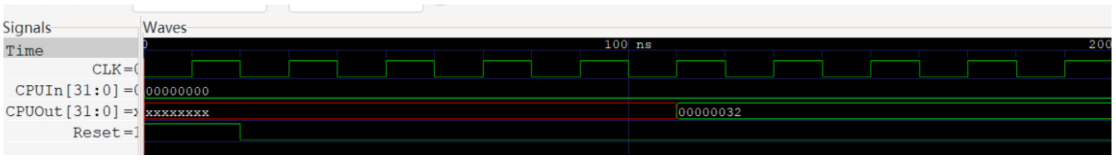
```
`timescale 1ns/1ps
`include "risc_v.sv"

module risc_v_tb;
logic [31:0] CPUOut, CPUIn;
logic Reset, CLK;

risc_v dut(CPUOut, CPUIn, Reset, CLK);

initial begin // Generate clock signal with 20 ns period
CLK = 0;
forever #10 CLK = ~CLK;
end

initial begin // Apply stimulus

// Enter your code here
$dumpfile("risc_v_tb.vcd");
$dumpvars(0, risc_v_tb);

Reset = 1;
CPUIn = 32'b0;
// Step 2: Release reset
#20 Reset = 0;
$display("CPU Reset Done");
```

```
// Step 3: Run CPU for several clock cycles
repeat (50) begin
    #20;
    CPUIn = 0;
end

// Step 4: End simulation
$display("Simulation complete.");

$finish; // This system tasks ends the simulation
end

always @ (negedge CLK)
$display ("t = %3d, CPUIn = %d, CPUOut = %d, Reset = %b, PCSrc = %b PC
= %d, PCTarget = %h, ImmExt = %h, Instr = %h, ALUResult = %d", $time,
CPUIn, CPUOut, Reset, dut.PCSrc, dut.PC, dut.PCTarget, dut.ImmExt,
dut.Instr, dut.ALUResult);

endmodule
```

Risc_v_tb results:



```
VCD info: dumpfile risc_v_tb.vcd opened for output.
CPU Reset Done
t =  20, CPUIn =          0, CPUOut =          x, Reset = 0, PCSrc = 00 PC =          0, PCTarget = 00000032, ImmExt = 00000032, Instr = 03200093, ALUResult =         50
t =  40, CPUIn =          0, CPUOut =          x, Reset = 0, PCSrc = 00 PC =          4, PCTarget = 80000004, ImmExt = 80000000, Instr = 80000537, ALUResult =          0
t =  60, CPUIn =          0, CPUOut =          x, Reset = 0, PCSrc = 00 PC =          8, PCTarget = 00000004, ImmExt = fffffffc, Instr = ffc52103, ALUResult = 2147483644
t =  80, CPUIn =          0, CPUOut =          x, Reset = 0, PCSrc = 00 PC =         12, PCTarget = 0000040e, ImmExt = 00000402, Instr = 402081b3, ALUResult =         50
t = 100, CPUIn =          0, CPUOut =          x, Reset = 0, PCSrc = 00 PC =         16, PCTarget = 0000000c, ImmExt = fffffffc, Instr = fe352e23, ALUResult = 2147483644
t = 120, CPUIn =          0, CPUOut =         50, Reset = 0, PCSrc = 01 PC =         20, PCTarget = 00000014, ImmExt = 00000000, Instr = 00000063, ALUResult =          0
t = 140, CPUIn =          0, CPUOut =         50, Reset = 0, PCSrc = 01 PC =         20, PCTarget = 00000014, ImmExt = 00000000, Instr = 00000063, ALUResult =          0
t = 160, CPUIn =          0, CPUOut =         50, Reset = 0, PCSrc = 01 PC =         20, PCTarget = 00000014, ImmExt = 00000000, Instr = 00000063, ALUResult =          0
t = 180, CPUIn =          0, CPUOut =         50, Reset = 0, PCSrc = 01 PC =         20, PCTarget = 00000014, ImmExt = 00000000, Instr = 00000063, ALUResult =          0
t = 200, CPUIn =          0, CPUOut =         50, Reset = 0, PCSrc = 01 PC =         20, PCTarget = 00000014, ImmExt = 00000000, Instr = 00000063, ALUResult =          0
```

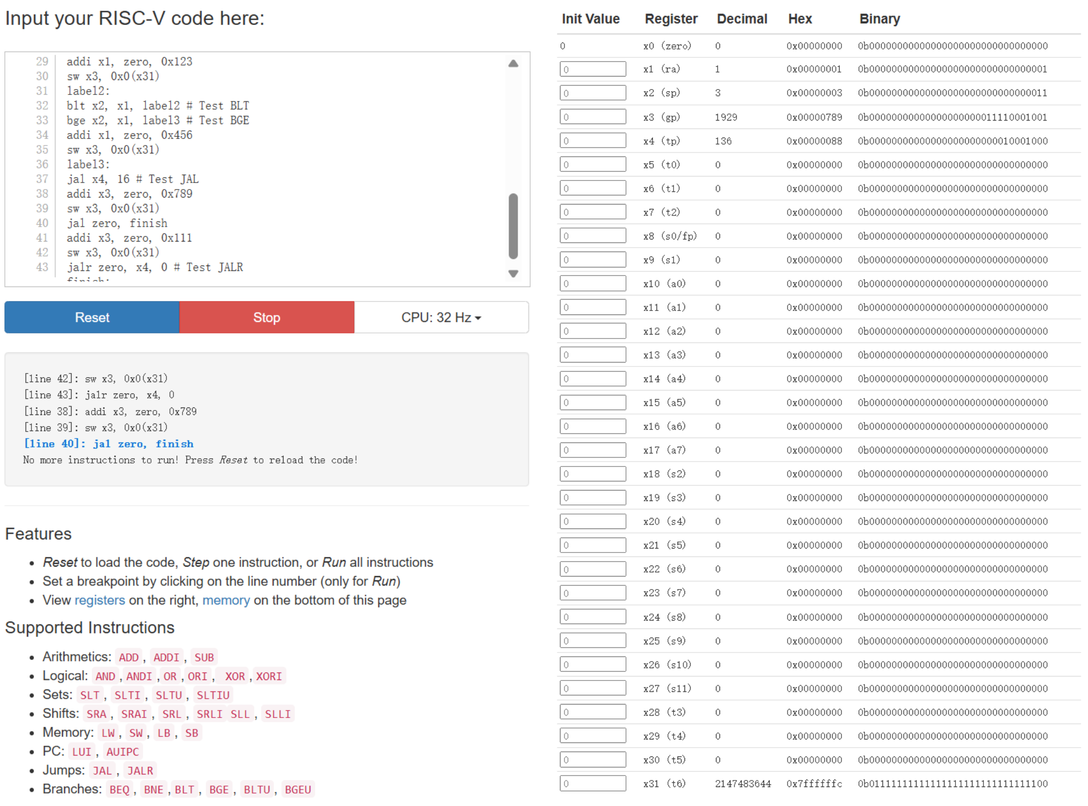


The machine code is input from the program.txt file, after disassembling the machine code into assembly code, we know that:

addi x1, x0, 50	Load the value 50 into register x1
lui x10, 0x80000	Load upper immediate to x10 (sets x10 to 0x80000000)
lw x2, -4(x10)	Load a word from memory at address (x10 - 4) = 0x7FFFFFFC into x2
sub x3, x1, x2	Subtract x2 from x1 and store the result in x3
sw x3, -4(x10)	Store x3 back into memory at address (x10 - 4) = 0x7FFFFFFC
beq x0, x0, 0	Infinite loop (branch to itself)

In the risc_v_tb file, the CPUIn is set to 0, therefore x2 is always 0. From the data_memory_and_io file, it is clear that 0x7FFFFFFC is a specific address instead of normal memory address, it is related to CPUOut. Hence, sw x3, -4(x10), will store the value of x3 register to CPUOut in the following positive clock edge. The result from the testbench code is aligned with the assembly code.

Task 1.5 Running a test program



Input your RISC-V code here:

```
29  addi x1, zero, 0x123
30  sw x3, 0x0(x31)
31  label2:
32  blt x2, x1, label2 # Test BLT
33  bge x2, x1, label3 # Test BGE
34  addi x1, zero, 0x456
35  sw x3, 0x0(x31)
36  label3:
37  jal x4, 16 # Test JAL
38  addi x3, zero, 0x789
39  sw x3, 0x0(x31)
40  jal zero, finish
41  addi x3, zero, 0x111
42  sw x3, 0x0(x31)
43  jalr zero, x4, 0 # Test JALR
```

```
[line 42]: sw x3, 0x0(x31)
[line 43]: jalr zero, x4, 0
[line 38]: addi x3, zero, 0x789
[line 39]: sw x3, 0x0(x31)
[line 40]: jal zero, finish
No more instructions to run! Press Reset to reload the code!
```

Features

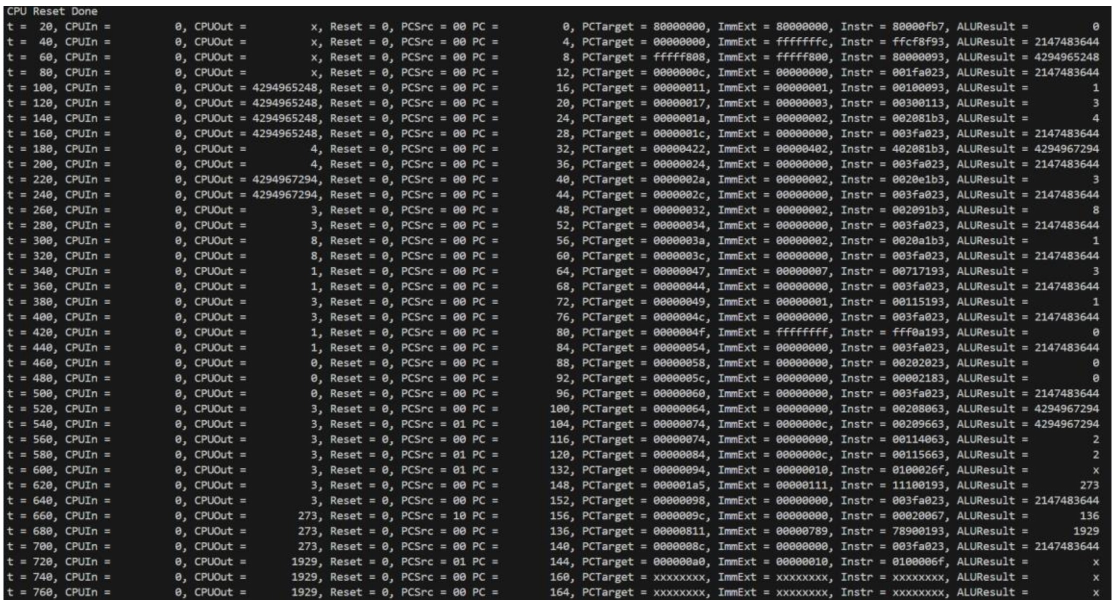
- *Reset* to load the code, *Step* one instruction, or *Run* all instructions
- Set a breakpoint by clicking on the line number (only for *Run*)
- View registers on the right, memory on the bottom of this page

Supported Instructions

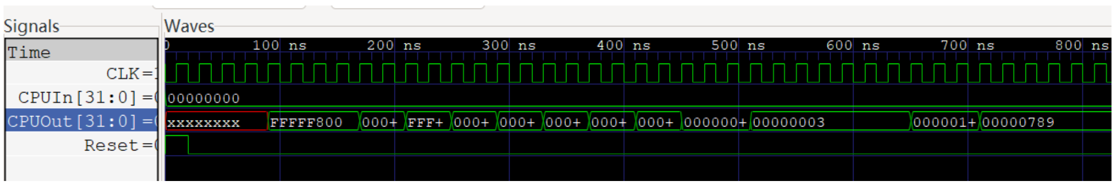
- Arithmetics: ADD, ADDI, SUB
- Logical: AND, ANDI, OR, ORI, XOR, XORI
- Sets: SLT, SLTI, SLTU, SLTIU
- Shifts: SRA, SRAI, SRL, SRLI SLL, SLLI
- Memory: LW, SW, LB, SB
- PC: LUI, AUIPC
- Jumps: JAL, JALR
- Branches: BEQ, BNE, BLT, BGE, BLTU, BGEU

Init Value	Register	Decimal	Hex	Binary
0	x0 (zero)	0	0x00000000	0b00000000000000000000000000000000
0	x1 (ra)	1	0x00000001	0b00000000000000000000000000000001
0	x2 (sp)	3	0x00000003	0b00000000000000000000000000000011
0	x3 (gp)	1929	0x00000789	0b00000000000000000000011110001001
0	x4 (tp)	136	0x00000088	0b00000000000000000000000010001000
0	x5 (t0)	0	0x00000000	0b00000000000000000000000000000000
0	x6 (t1)	0	0x00000000	0b00000000000000000000000000000000
0	x7 (t2)	0	0x00000000	0b00000000000000000000000000000000
0	x8 (s0/fp)	0	0x00000000	0b00000000000000000000000000000000
0	x9 (s1)	0	0x00000000	0b00000000000000000000000000000000
0	x10 (a0)	0	0x00000000	0b00000000000000000000000000000000
0	x11 (a1)	0	0x00000000	0b00000000000000000000000000000000
0	x12 (a2)	0	0x00000000	0b00000000000000000000000000000000
0	x13 (a3)	0	0x00000000	0b00000000000000000000000000000000
0	x14 (a4)	0	0x00000000	0b00000000000000000000000000000000
0	x15 (a5)	0	0x00000000	0b00000000000000000000000000000000
0	x16 (a6)	0	0x00000000	0b00000000000000000000000000000000
0	x17 (a7)	0	0x00000000	0b00000000000000000000000000000000
0	x18 (s2)	0	0x00000000	0b00000000000000000000000000000000
0	x19 (s3)	0	0x00000000	0b00000000000000000000000000000000
0	x20 (s4)	0	0x00000000	0b00000000000000000000000000000000
0	x21 (s5)	0	0x00000000	0b00000000000000000000000000000000
0	x22 (s6)	0	0x00000000	0b00000000000000000000000000000000
0	x23 (s7)	0	0x00000000	0b00000000000000000000000000000000
0	x24 (s8)	0	0x00000000	0b00000000000000000000000000000000
0	x25 (s9)	0	0x00000000	0b00000000000000000000000000000000
0	x26 (s10)	0	0x00000000	0b00000000000000000000000000000000
0	x27 (s11)	0	0x00000000	0b00000000000000000000000000000000
0	x28 (t3)	0	0x00000000	0b00000000000000000000000000000000
0	x29 (t4)	0	0x00000000	0b00000000000000000000000000000000
0	x30 (t5)	0	0x00000000	0b00000000000000000000000000000000
0	x31 (t6)	2147483644	0x7ffffffc	0b01111111111111111111111111111100

Risc_v_tb simulation result:



```
CPU Reset Done
t =  20, CPUIn =          0, CPUOut =          x, Reset = 0, PCSrc = 00 PC =          0, PCTarget = 80000000, ImmExt = 80000000, Instr = 80000fb7, ALUResult =          0
t =  40, CPUIn =          0, CPUOut =          x, Reset = 0, PCSrc = 00 PC =          4, PCTarget = 00000000, ImmExt = fffffffc, Instr = ffcf8f93, ALUResult = 2147483644
t =  60, CPUIn =          0, CPUOut =          x, Reset = 0, PCSrc = 00 PC =          8, PCTarget = fffff808, ImmExt = fffff800, Instr = 80000093, ALUResult = 4294965248
t =  80, CPUIn =          0, CPUOut =          x, Reset = 0, PCSrc = 00 PC =         12, PCTarget = 0000000c, ImmExt = 00000000, Instr = 001fa023, ALUResult = 2147483644
t = 100, CPUIn =          0, CPUOut = 4294965248, Reset = 0, PCSrc = 00 PC =         16, PCTarget = 00000011, ImmExt = 00000001, Instr = 00100093, ALUResult =          1
t = 120, CPUIn =          0, CPUOut = 4294965248, Reset = 0, PCSrc = 00 PC =         20, PCTarget = 00000017, ImmExt = 00000003, Instr = 00300113, ALUResult =          3
t = 140, CPUIn =          0, CPUOut = 4294965248, Reset = 0, PCSrc = 00 PC =         24, PCTarget = 0000001a, ImmExt = 00000002, Instr = 002081b3, ALUResult =          4
t = 160, CPUIn =          0, CPUOut = 4294965248, Reset = 0, PCSrc = 00 PC =         28, PCTarget = 0000001c, ImmExt = 00000000, Instr = 003fa023, ALUResult = 2147483644
t = 180, CPUIn =          0, CPUOut =          4, Reset = 0, PCSrc = 00 PC =         32, PCTarget = 00000422, ImmExt = 00000402, Instr = 402081b3, ALUResult = 4294967294
t = 200, CPUIn =          0, CPUOut =          4, Reset = 0, PCSrc = 00 PC =         36, PCTarget = 00000024, ImmExt = 00000000, Instr = 003fa023, ALUResult = 2147483644
t = 220, CPUIn =          0, CPUOut = 4294967294, Reset = 0, PCSrc = 00 PC =         40, PCTarget = 0000002a, ImmExt = 00000002, Instr = 0020e1b3, ALUResult =          3
t = 240, CPUIn =          0, CPUOut = 4294967294, Reset = 0, PCSrc = 00 PC =         44, PCTarget = 0000002c, ImmExt = 00000000, Instr = 003fa023, ALUResult = 2147483644
t = 260, CPUIn =          0, CPUOut =          3, Reset = 0, PCSrc = 00 PC =         48, PCTarget = 00000032, ImmExt = 00000002, Instr = 002091b3, ALUResult =          8
t = 280, CPUIn =          0, CPUOut =          3, Reset = 0, PCSrc = 00 PC =         52, PCTarget = 00000034, ImmExt = 00000000, Instr = 003fa023, ALUResult = 2147483644
t = 300, CPUIn =          0, CPUOut =          8, Reset = 0, PCSrc = 00 PC =         56, PCTarget = 0000003a, ImmExt = 00000002, Instr = 0020a1b3, ALUResult =          1
t = 320, CPUIn =          0, CPUOut =          8, Reset = 0, PCSrc = 00 PC =         60, PCTarget = 0000003c, ImmExt = 00000000, Instr = 003fa023, ALUResult = 2147483644
t = 340, CPUIn =          0, CPUOut =          1, Reset = 0, PCSrc = 00 PC =         64, PCTarget = 00000047, ImmExt = 00000007, Instr = 00717193, ALUResult =          3
t = 360, CPUIn =          0, CPUOut =          1, Reset = 0, PCSrc = 00 PC =         68, PCTarget = 00000044, ImmExt = 00000000, Instr = 003fa023, ALUResult = 2147483644
t = 380, CPUIn =          0, CPUOut =          3, Reset = 0, PCSrc = 00 PC =         72, PCTarget = 00000049, ImmExt = 00000001, Instr = 00115193, ALUResult =          1
t = 400, CPUIn =          0, CPUOut =          3, Reset = 0, PCSrc = 00 PC =         76, PCTarget = 0000004c, ImmExt = 00000000, Instr = 003fa023, ALUResult = 2147483644
t = 420, CPUIn =          0, CPUOut =          1, Reset = 0, PCSrc = 00 PC =         80, PCTarget = 0000004f, ImmExt = ffffffff, Instr = fff0a193, ALUResult =          0
t = 440, CPUIn =          0, CPUOut =          1, Reset = 0, PCSrc = 00 PC =         84, PCTarget = 00000054, ImmExt = 00000000, Instr = 003fa023, ALUResult = 2147483644
t = 460, CPUIn =          0, CPUOut =          0, Reset = 0, PCSrc = 00 PC =         88, PCTarget = 00000058, ImmExt = 00000000, Instr = 00202023, ALUResult =          0
t = 480, CPUIn =          0, CPUOut =          0, Reset = 0, PCSrc = 00 PC =         92, PCTarget = 0000005c, ImmExt = 00000000, Instr = 00002183, ALUResult =          0
t = 500, CPUIn =          0, CPUOut =          0, Reset = 0, PCSrc = 00 PC =         96, PCTarget = 00000060, ImmExt = 00000000, Instr = 003fa023, ALUResult = 2147483644
t = 520, CPUIn =          0, CPUOut =          3, Reset = 0, PCSrc = 00 PC =        100, PCTarget = 00000064, ImmExt = 00000000, Instr = 00208063, ALUResult = 4294967294
t = 540, CPUIn =          0, CPUOut =          3, Reset = 0, PCSrc = 01 PC =        104, PCTarget = 00000074, ImmExt = 0000000c, Instr = 00209663, ALUResult = 4294967294
t = 560, CPUIn =          0, CPUOut =          3, Reset = 0, PCSrc = 00 PC =        116, PCTarget = 00000074, ImmExt = 00000000, Instr = 00114063, ALUResult =          2
t = 580, CPUIn =          0, CPUOut =          3, Reset = 0, PCSrc = 01 PC =        120, PCTarget = 00000084, ImmExt = 0000000c, Instr = 00115663, ALUResult =          2
t = 600, CPUIn =          0, CPUOut =          3, Reset = 0, PCSrc = 01 PC =        132, PCTarget = 00000094, ImmExt = 00000010, Instr = 0100026f, ALUResult =          x
t = 620, CPUIn =          0, CPUOut =          3, Reset = 0, PCSrc = 00 PC =        148, PCTarget = 000001a5, ImmExt = 00000111, Instr = 11100193, ALUResult =        273
t = 640, CPUIn =          0, CPUOut =          3, Reset = 0, PCSrc = 00 PC =        152, PCTarget = 00000098, ImmExt = 00000000, Instr = 003fa023, ALUResult = 2147483644
t = 660, CPUIn =          0, CPUOut =        273, Reset = 0, PCSrc = 10 PC =        156, PCTarget = 0000009c, ImmExt = 00000000, Instr = 00020067, ALUResult =        136
t = 680, CPUIn =          0, CPUOut =        273, Reset = 0, PCSrc = 00 PC =        136, PCTarget = 00000811, ImmExt = 00000789, Instr = 78900193, ALUResult =       1929
t = 700, CPUIn =          0, CPUOut =        273, Reset = 0, PCSrc = 00 PC =        140, PCTarget = 0000008c, ImmExt = 00000000, Instr = 003fa023, ALUResult = 2147483644
t = 720, CPUIn =          0, CPUOut =       1929, Reset = 0, PCSrc = 01 PC =        144, PCTarget = 000000a0, ImmExt = 00000010, Instr = 0100006f, ALUResult =          x
t = 740, CPUIn =          0, CPUOut =       1929, Reset = 0, PCSrc = 00 PC =        160, PCTarget = xxxxxxxx, ImmExt = xxxxxxxx, Instr = xxxxxxxx, ALUResult =          x
t = 760, CPUIn =          0, CPUOut =       1929, Reset = 0, PCSrc = 00 PC =        164, PCTarget = xxxxxxxx, ImmExt = xxxxxxxx, Instr = xxxxxxxx, ALUResult =          x
```



This program test is basically test each control signals along with the ALUResult respectively. All of the simulated output are performed as expected with the comparison of RISCV emulator.

Though, there is a few points might cause confusion: firstly, immediates like 0x800 in our simulation is different with the emulator as the instruction extend table provided for operating addi function is different. The ImmExt extracted becomes 0xFFFFF800 instead of 0x00000800. Besides, though the ALUResult in the control module is defined as signed, it is defined as unsigned in the risc_v module, therefore the output of the ALUResult is unsigned.

Apart from the above two harmless difference, other functions performs very well. The PC increments in the same way as the emulator, increasing by 4 after each instruction. Arithmetic operations are correct, such as add x3, x1, x2 → ALUResult = 4. Jump and branch instructions correctly modify PCTarget in the simulation.

At final state, the result from RISC-V emulator is that the code would jump to finish at line 40 (machine code: 0100006f), which is exactly aligned with out simulation result no ALU. Similar to the last task, it is clear that at line 38, the assembly code is addi x3, zero, 0x789, which stores 0x789 to x3. Then the line 39 sw x3, 0x0(x31), it again stores the value of x3 to CPUOut, we can see that in our simulation result, the CPUOut is written 1929 in decimal in the next positive clock edge.

Our risc_v code perfectly performs the arithmetic operation and stores the value into the exact register position as required. CPUOur perfectly respond to the sw function. The emulator shows the same function as our designed risc_v program.

Part 2 RISC-V Assembly Programming

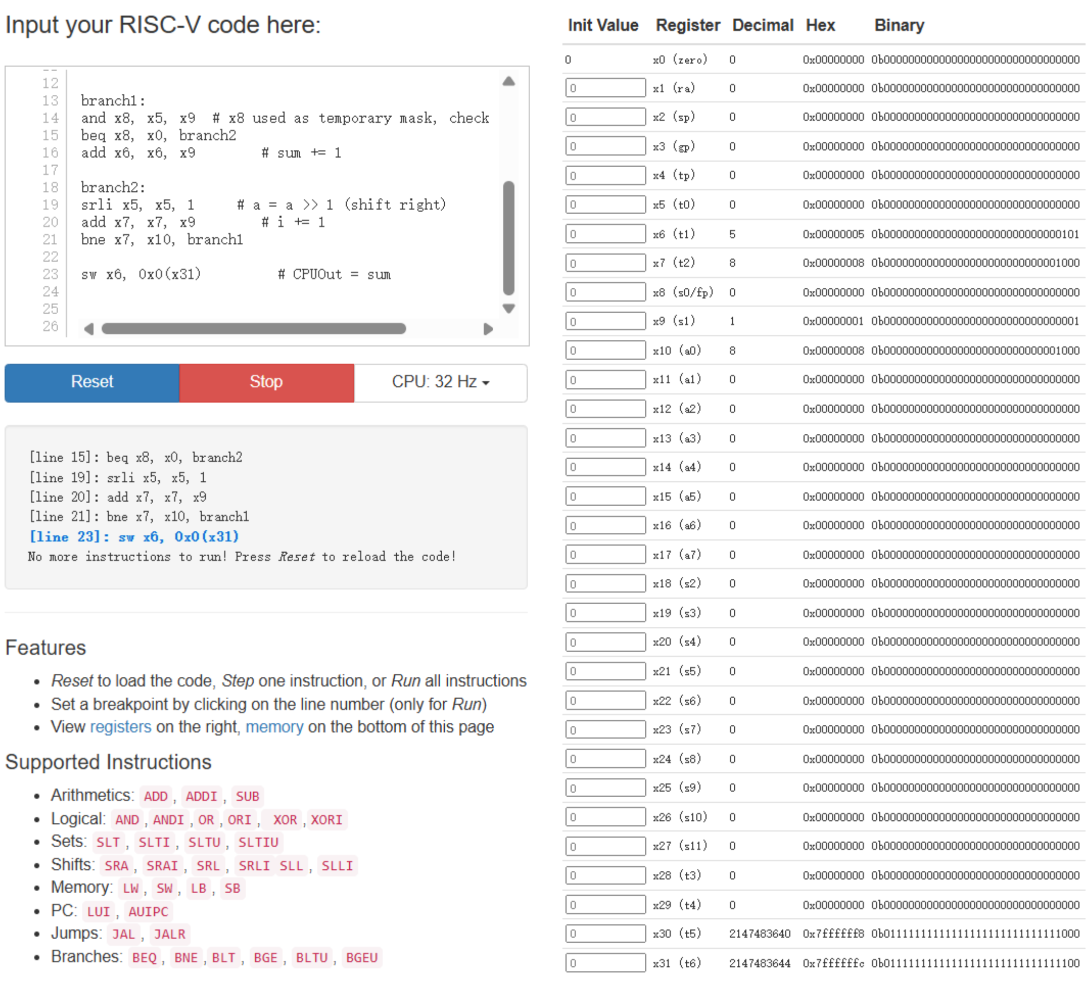
Task 2.1 A program to count the number of 1's in an input byte

Assembly code:

```
# Initialize CPUIn and CPUOut

Lui x31, 0x80000

addi x31, x31, -4           # CPUOut and CPUIn


# Initialize parameters: a, sum, i

lw x5, 0(x31)          # x5 = a = CPUIn

addi x6, x0, 0         # x6 = sum = 0

addi x7, x0, 0         # x7 = i = 0

addi x9, x0, 1       # Load constant 1

addi x10, x0, 8       # Load constant 8 (loop limit)


#  Start

branch1:

and x8, x5, x9   # x8 used as temporary mask, check if LSB is 1

beq x8, x0, branch2

add x6, x6, x9       # sum += 1


branch2:

srli x5, x5, 1     # a = a >> 1 (shift right)

add x7, x7, x9       # i += 1

bne x7, x10, branch1

#  Stop

sw x6, 0x0(x31)           # CPUOut = sum
```

After inputting the assembly code into the emulator by setting the CPUIn to 0x000001F, the CPUOut is 5 in decimal as expected. As the register 6 stores the value of CPUOut, the assembly code works well.

RiscV emulator result:



```
12
13  branch1:
14  and x8, x5, x9  # x8 used as temporary mask, check
15  beq x8, x0, branch2
16  add x6, x6, x9        # sum += 1
17
18  branch2:
19  srli x5, x5, 1     # a = a >> 1 (shift right)
20  add x7, x7, x9        # i += 1
21  bne x7, x10, branch1
22
23  sw x6, 0x0(x31)         # CPUOut = sum
24
25
26
```

```
[line 15]: beq x8, x0, branch2
[line 19]: srli x5, x5, 1
[line 20]: add x7, x7, x9
[line 21]: bne x7, x10, branch1
[line 23]: sw x6, 0x0(x31)
No more instructions to run! Press Reset to reload the code!
```

Features

- *Reset* to load the code, *Step* one instruction, or *Run* all instructions
- Set a breakpoint by clicking on the line number (only for *Run*)
- View registers on the right, memory on the bottom of this page

Supported Instructions

- Arithmetics: ADD, ADDI, SUB
- Logical: AND, ANDI, OR, ORI, XOR, XORI
- Sets: SLT, SLTI, SLTU, SLTIU
- Shifts: SRA, SRAI, SRL, SRLI SLL, SLLI
- Memory: LW, SW, LB, SB
- PC: LUI, AUIPC
- Jumps: JAL, JALR
- Branches: BEQ, BNE, BLT, BGE, BLTU, BGEU

Init Value	Register	Decimal	Hex	Binary
0	x0 (zero)	0	0x00000000	0b00000000000000000000000000000000
0	x1 (ra)	0	0x00000000	0b00000000000000000000000000000000
0	x2 (sp)	0	0x00000000	0b00000000000000000000000000000000
0	x3 (gp)	0	0x00000000	0b00000000000000000000000000000000
0	x4 (tp)	0	0x00000000	0b00000000000000000000000000000000
0	x5 (t0)	0	0x00000000	0b00000000000000000000000000000000
0	x6 (t1)	5	0x00000005	0b00000000000000000000000000000101
0	x7 (t2)	8	0x00000008	0b00000000000000000000000000001000
0	x8 (s0/fp)	0	0x00000000	0b00000000000000000000000000000000
0	x9 (s1)	1	0x00000001	0b00000000000000000000000000000001
0	x10 (a0)	8	0x00000008	0b00000000000000000000000000001000
0	x11 (a1)	0	0x00000000	0b00000000000000000000000000000000
0	x12 (a2)	0	0x00000000	0b00000000000000000000000000000000
0	x13 (a3)	0	0x00000000	0b00000000000000000000000000000000
0	x14 (a4)	0	0x00000000	0b00000000000000000000000000000000
0	x15 (a5)	0	0x00000000	0b00000000000000000000000000000000
0	x16 (a6)	0	0x00000000	0b00000000000000000000000000000000
0	x17 (a7)	0	0x00000000	0b00000000000000000000000000000000
0	x18 (s2)	0	0x00000000	0b00000000000000000000000000000000
0	x19 (s3)	0	0x00000000	0b00000000000000000000000000000000
0	x20 (s4)	0	0x00000000	0b00000000000000000000000000000000
0	x21 (s5)	0	0x00000000	0b00000000000000000000000000000000
0	x22 (s6)	0	0x00000000	0b00000000000000000000000000000000
0	x23 (s7)	0	0x00000000	0b00000000000000000000000000000000
0	x24 (s8)	0	0x00000000	0b00000000000000000000000000000000
0	x25 (s9)	0	0x00000000	0b00000000000000000000000000000000
0	x26 (s10)	0	0x00000000	0b00000000000000000000000000000000
0	x27 (s11)	0	0x00000000	0b00000000000000000000000000000000
0	x28 (t3)	0	0x00000000	0b00000000000000000000000000000000
0	x29 (t4)	0	0x00000000	0b00000000000000000000000000000000
0	x30 (t5)	2147483640	0x7ffffff8	0b01111111111111111111111111111000
0	x31 (t6)	2147483644	0x7ffffffc	0b01111111111111111111111111111100

Then, we convert the assembly code into machine code for further simulation. It should be noticed that instead of using branch1 in the assembly code, it should be counted as an immediate number (steps * 4).

Machine code:

80000fb7

ffcf8f93

000fa283

00000313

00000393

00100493

00800513

0092F433

00040463

00930333

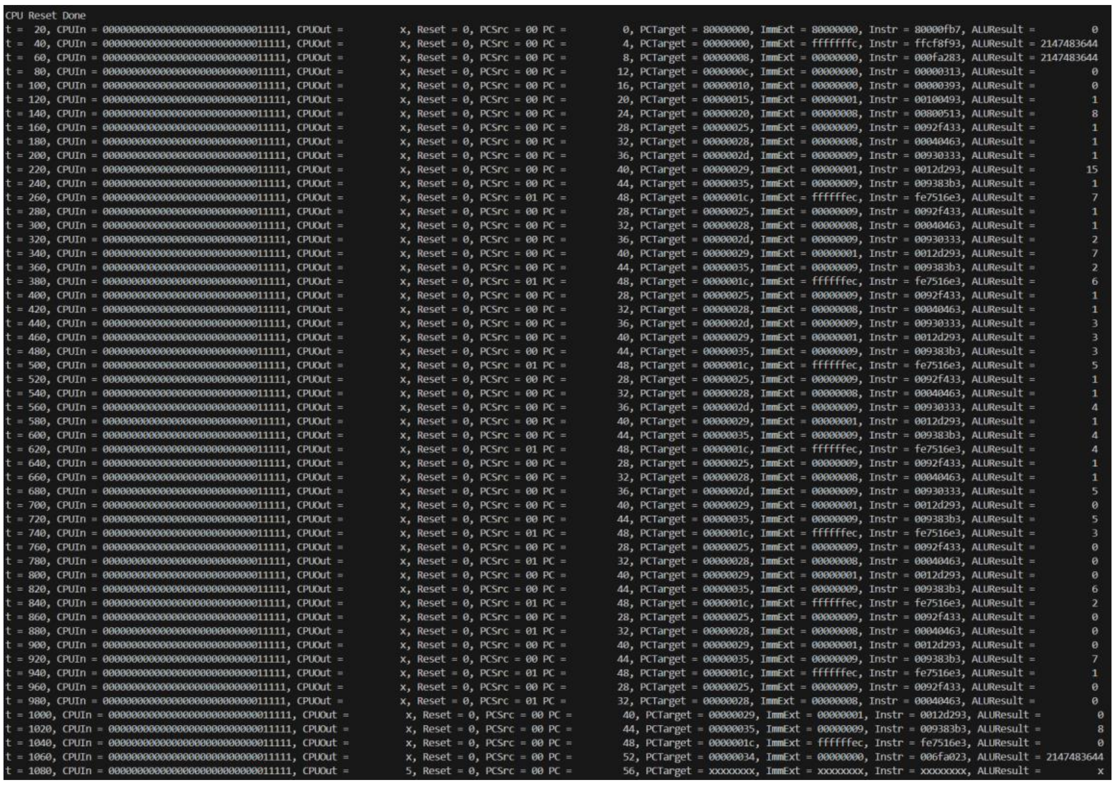
0012d293

009383b3

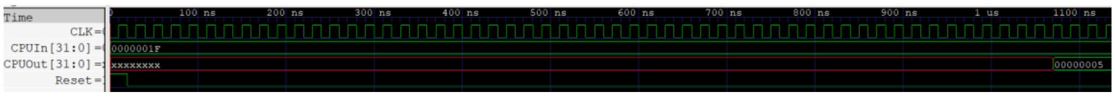
fe7516e3

006fa023

Here we save the machine code into a new text file and simulate the result.



```
CPU Reset Done
t =  20, CPUIn = 00000000000000000000000000011111, CPUOut =          x, Reset = 0, PCSrc = 00 PC =          0, PCTarget = 80000000, ImmExt = 80000000, Instr = 80000fb7, ALUResult =          0
t =  40, CPUIn = 00000000000000000000000000011111, CPUOut =          x, Reset = 0, PCSrc = 00 PC =          4, PCTarget = 00000000, ImmExt = fffffffc, Instr = ffcf8f93, ALUResult = 2147483644
t =  60, CPUIn = 00000000000000000000000000011111, CPUOut =          x, Reset = 0, PCSrc = 00 PC =          8, PCTarget = 00000008, ImmExt = 00000000, Instr = 000fa283, ALUResult = 2147483644
t =  80, CPUIn = 00000000000000000000000000011111, CPUOut =          x, Reset = 0, PCSrc = 00 PC =         12, PCTarget = 0000000c, ImmExt = 00000000, Instr = 00000313, ALUResult =          0
t = 100, CPUIn = 00000000000000000000000000011111, CPUOut =          x, Reset = 0, PCSrc = 00 PC =         16, PCTarget = 00000010, ImmExt = 00000000, Instr = 00000393, ALUResult =          0
t = 120, CPUIn = 00000000000000000000000000011111, CPUOut =          x, Reset = 0, PCSrc = 00 PC =         20, PCTarget = 00000015, ImmExt = 00000001, Instr = 00100493, ALUResult =          1
t = 140, CPUIn = 00000000000000000000000000011111, CPUOut =          x, Reset = 0, PCSrc = 00 PC =         24, PCTarget = 00000020, ImmExt = 00000008, Instr = 00800513, ALUResult =          8
t = 160, CPUIn = 00000000000000000000000000011111, CPUOut =          x, Reset = 0, PCSrc = 00 PC =         28, PCTarget = 00000025, ImmExt = 00000009, Instr = 0092f433, ALUResult =          1
t = 180, CPUIn = 00000000000000000000000000011111, CPUOut =          x, Reset = 0, PCSrc = 00 PC =         32, PCTarget = 00000028, ImmExt = 00000008, Instr = 00040463, ALUResult =          1
t = 200, CPUIn = 00000000000000000000000000011111, CPUOut =          x, Reset = 0, PCSrc = 00 PC =         36, PCTarget = 0000002d, ImmExt = 00000009, Instr = 00930333, ALUResult =          1
t = 220, CPUIn = 00000000000000000000000000011111, CPUOut =          x, Reset = 0, PCSrc = 00 PC =         40, PCTarget = 00000029, ImmExt = 00000001, Instr = 0012d293, ALUResult =         15
t = 240, CPUIn = 00000000000000000000000000011111, CPUOut =          x, Reset = 0, PCSrc = 00 PC =         44, PCTarget = 00000035, ImmExt = 00000009, Instr = 009383b3, ALUResult =          1
t = 260, CPUIn = 00000000000000000000000000011111, CPUOut =          x, Reset = 0, PCSrc = 01 PC =         48, PCTarget = 0000001c, ImmExt = ffffffec, Instr = fe7516e3, ALUResult =          7
t = 280, CPUIn = 00000000000000000000000000011111, CPUOut =          x, Reset = 0, PCSrc = 00 PC =         28, PCTarget = 00000025, ImmExt = 00000009, Instr = 0092f433, ALUResult =          1
t = 300, CPUIn = 00000000000000000000000000011111, CPUOut =          x, Reset = 0, PCSrc = 00 PC =         32, PCTarget = 00000028, ImmExt = 00000008, Instr = 00040463, ALUResult =          1
t = 320, CPUIn = 00000000000000000000000000011111, CPUOut =          x, Reset = 0, PCSrc = 00 PC =         36, PCTarget = 0000002d, ImmExt = 00000009, Instr = 00930333, ALUResult =          2
t = 340, CPUIn = 00000000000000000000000000011111, CPUOut =          x, Reset = 0, PCSrc = 00 PC =         40, PCTarget = 00000029, ImmExt = 00000001, Instr = 0012d293, ALUResult =          7
t = 360, CPUIn = 00000000000000000000000000011111, CPUOut =          x, Reset = 0, PCSrc = 00 PC =         44, PCTarget = 00000035, ImmExt = 00000009, Instr = 009383b3, ALUResult =          2
t = 380, CPUIn = 00000000000000000000000000011111, CPUOut =          x, Reset = 0, PCSrc = 01 PC =         48, PCTarget = 0000001c, ImmExt = ffffffec, Instr = fe7516e3, ALUResult =          6
t = 400, CPUIn = 00000000000000000000000000011111, CPUOut =          x, Reset = 0, PCSrc = 00 PC =         28, PCTarget = 00000025, ImmExt = 00000009, Instr = 0092f433, ALUResult =          1
t = 420, CPUIn = 00000000000000000000000000011111, CPUOut =          x, Reset = 0, PCSrc = 00 PC =         32, PCTarget = 00000028, ImmExt = 00000008, Instr = 00040463, ALUResult =          1
t = 440, CPUIn = 00000000000000000000000000011111, CPUOut =          x, Reset = 0, PCSrc = 00 PC =         36, PCTarget = 0000002d, ImmExt = 00000009, Instr = 00930333, ALUResult =          3
t = 460, CPUIn = 00000000000000000000000000011111, CPUOut =          x, Reset = 0, PCSrc = 00 PC =         40, PCTarget = 00000029, ImmExt = 00000001, Instr = 0012d293, ALUResult =          3
t = 480, CPUIn = 00000000000000000000000000011111, CPUOut =          x, Reset = 0, PCSrc = 00 PC =         44, PCTarget = 00000035, ImmExt = 00000009, Instr = 009383b3, ALUResult =          3
t = 500, CPUIn = 00000000000000000000000000011111, CPUOut =          x, Reset = 0, PCSrc = 01 PC =         48, PCTarget = 0000001c, ImmExt = ffffffec, Instr = fe7516e3, ALUResult =          5
t = 520, CPUIn = 00000000000000000000000000011111, CPUOut =          x, Reset = 0, PCSrc = 00 PC =         28, PCTarget = 00000025, ImmExt = 00000009, Instr = 0092f433, ALUResult =          1
t = 540, CPUIn = 00000000000000000000000000011111, CPUOut =          x, Reset = 0, PCSrc = 00 PC =         32, PCTarget = 00000028, ImmExt = 00000008, Instr = 00040463, ALUResult =          1
t = 560, CPUIn = 00000000000000000000000000011111, CPUOut =          x, Reset = 0, PCSrc = 00 PC =         36, PCTarget = 0000002d, ImmExt = 00000009, Instr = 00930333, ALUResult =          4
t = 580, CPUIn = 00000000000000000000000000011111, CPUOut =          x, Reset = 0, PCSrc = 00 PC =         40, PCTarget = 00000029, ImmExt = 00000001, Instr = 0012d293, ALUResult =          1
t = 600, CPUIn = 00000000000000000000000000011111, CPUOut =          x, Reset = 0, PCSrc = 00 PC =         44, PCTarget = 00000035, ImmExt = 00000009, Instr = 009383b3, ALUResult =          4
t = 620, CPUIn = 00000000000000000000000000011111, CPUOut =          x, Reset = 0, PCSrc = 01 PC =         48, PCTarget = 0000001c, ImmExt = ffffffec, Instr = fe7516e3, ALUResult =          4
t = 640, CPUIn = 00000000000000000000000000011111, CPUOut =          x, Reset = 0, PCSrc = 00 PC =         28, PCTarget = 00000025, ImmExt = 00000009, Instr = 0092f433, ALUResult =          1
t = 660, CPUIn = 00000000000000000000000000011111, CPUOut =          x, Reset = 0, PCSrc = 00 PC =         32, PCTarget = 00000028, ImmExt = 00000008, Instr = 00040463, ALUResult =          1
t = 680, CPUIn = 00000000000000000000000000011111, CPUOut =          x, Reset = 0, PCSrc = 00 PC =         36, PCTarget = 0000002d, ImmExt = 00000009, Instr = 00930333, ALUResult =          5
t = 700, CPUIn = 00000000000000000000000000011111, CPUOut =          x, Reset = 0, PCSrc = 00 PC =         40, PCTarget = 00000029, ImmExt = 00000001, Instr = 0012d293, ALUResult =          0
t = 720, CPUIn = 00000000000000000000000000011111, CPUOut =          x, Reset = 0, PCSrc = 00 PC =         44, PCTarget = 00000035, ImmExt = 00000009, Instr = 009383b3, ALUResult =          5
t = 740, CPUIn = 00000000000000000000000000011111, CPUOut =          x, Reset = 0, PCSrc = 01 PC =         48, PCTarget = 0000001c, ImmExt = ffffffec, Instr = fe7516e3, ALUResult =          3
t = 760, CPUIn = 00000000000000000000000000011111, CPUOut =          x, Reset = 0, PCSrc = 00 PC =         28, PCTarget = 00000025, ImmExt = 00000009, Instr = 0092f433, ALUResult =          0
t = 780, CPUIn = 00000000000000000000000000011111, CPUOut =          x, Reset = 0, PCSrc = 01 PC =         32, PCTarget = 00000028, ImmExt = 00000008, Instr = 00040463, ALUResult =          0
t = 800, CPUIn = 00000000000000000000000000011111, CPUOut =          x, Reset = 0, PCSrc = 00 PC =         40, PCTarget = 00000029, ImmExt = 00000001, Instr = 0012d293, ALUResult =          0
t = 820, CPUIn = 00000000000000000000000000011111, CPUOut =          x, Reset = 0, PCSrc = 00 PC =         44, PCTarget = 00000035, ImmExt = 00000009, Instr = 009383b3, ALUResult =          6
t = 840, CPUIn = 00000000000000000000000000011111, CPUOut =          x, Reset = 0, PCSrc = 01 PC =         48, PCTarget = 0000001c, ImmExt = ffffffec, Instr = fe7516e3, ALUResult =          2
t = 860, CPUIn = 00000000000000000000000000011111, CPUOut =          x, Reset = 0, PCSrc = 00 PC =         28, PCTarget = 00000025, ImmExt = 00000009, Instr = 0092f433, ALUResult =          0
t = 880, CPUIn = 00000000000000000000000000011111, CPUOut =          x, Reset = 0, PCSrc = 01 PC =         32, PCTarget = 00000028, ImmExt = 00000008, Instr = 00040463, ALUResult =          0
t = 900, CPUIn = 00000000000000000000000000011111, CPUOut =          x, Reset = 0, PCSrc = 00 PC =         40, PCTarget = 00000029, ImmExt = 00000001, Instr = 0012d293, ALUResult =          0
t = 920, CPUIn = 00000000000000000000000000011111, CPUOut =          x, Reset = 0, PCSrc = 00 PC =         44, PCTarget = 00000035, ImmExt = 00000009, Instr = 009383b3, ALUResult =          7
t = 940, CPUIn = 00000000000000000000000000011111, CPUOut =          x, Reset = 0, PCSrc = 01 PC =         48, PCTarget = 0000001c, ImmExt = ffffffec, Instr = fe7516e3, ALUResult =          1
t = 960, CPUIn = 00000000000000000000000000011111, CPUOut =          x, Reset = 0, PCSrc = 00 PC =         28, PCTarget = 00000025, ImmExt = 00000009, Instr = 0092f433, ALUResult =          0
t = 980, CPUIn = 00000000000000000000000000011111, CPUOut =          x, Reset = 0, PCSrc = 01 PC =         32, PCTarget = 00000028, ImmExt = 00000008, Instr = 00040463, ALUResult =          0
t = 1000, CPUIn = 00000000000000000000000000011111, CPUOut =          x, Reset = 0, PCSrc = 00 PC =         40, PCTarget = 00000029, ImmExt = 00000001, Instr = 0012d293, ALUResult =          0
t = 1020, CPUIn = 00000000000000000000000000011111, CPUOut =          x, Reset = 0, PCSrc = 00 PC =         44, PCTarget = 00000035, ImmExt = 00000009, Instr = 009383b3, ALUResult =          8
t = 1040, CPUIn = 00000000000000000000000000011111, CPUOut =          x, Reset = 0, PCSrc = 00 PC =         48, PCTarget = 0000001c, ImmExt = ffffffec, Instr = fe7516e3, ALUResult =          0
t = 1060, CPUIn = 00000000000000000000000000011111, CPUOut =          x, Reset = 0, PCSrc = 00 PC =         52, PCTarget = 00000034, ImmExt = 00000000, Instr = 006fa023, ALUResult = 2147483644
t = 1080, CPUIn = 00000000000000000000000000011111, CPUOut =          5, Reset = 0, PCSrc = 00 PC =         56, PCTarget = xxxxxxxx, ImmExt = xxxxxxxx, Instr = xxxxxxxx, ALUResult =          x
```



It is clear that the simulation aligned with the expected output. The value of CPUIn is written from the testbench code into the register file, counts the number of 1's in the least significant byte (CPUIn[7:0]), and outputs this value via CPUOut after fixed 8 repeated cycles.

Task 2.1 Testbench code:

```
`timescale 1ns/1ps
`include "risc_v.sv"

module risc_v_tb;
logic [31:0] CPUOut, CPUIn;
logic Reset, CLK;

risc_v dut(CPUOut, CPUIn, Reset, CLK);

initial begin // Generate clock signal with 20 ns period
CLK = 0;
forever #10 CLK = ~CLK;
```

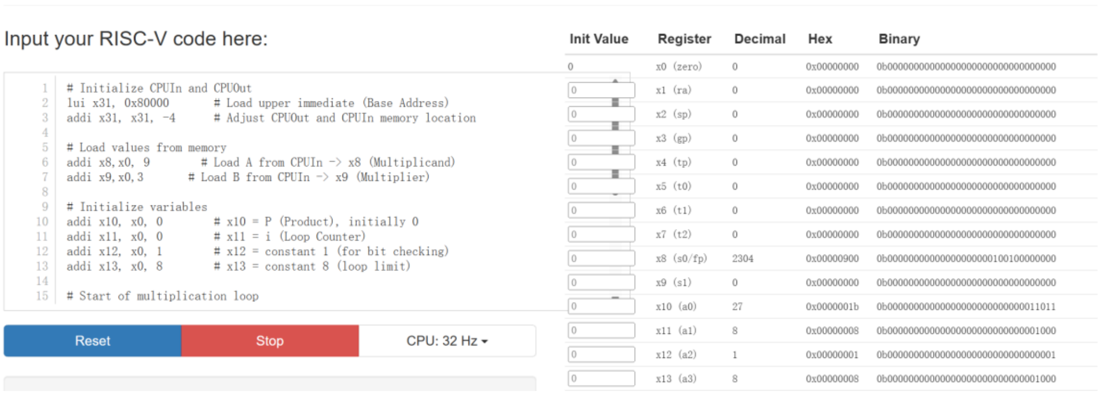
```
end

initial begin // Apply stimulus

// Enter your code here
$dumpfile("risc_v_tb.vcd");
$dumpvars(0, risc_v_tb);

Reset = 1;
CPUIn = 32'h0000001F;


// Step 2: Release reset
#20 Reset = 0;
$display("CPU Reset Done");

// Step 3: Run CPU for several clock cycles
repeat (70) begin
    #20;
end

// Step 4: End simulation
$display("Simulation complete.");

$finish; // This system tasks ends the simulation
end

always @ (negedge CLK)
$display ("t = %3d, CPUIn = %b, CPUOut = %d, Reset = %b, PCSrc = %b PC
= %d, PCTarget = %h, ImmExt = %h, Instr = %h, ALUResult = %d", $time,
CPUIn, CPUOut, Reset, dut.PCSrc, dut.PC, dut.PCTarget, dut.ImmExt,
dut.Instr, dut.ALUResult);

endmodule
```

Task 2.2 A program to multiply two bytes using the shift and add method

Assmbly code:

```
# Initialize CPUIn and CPUOut
lui x31, 0x80000       # Load upper immediate (Base Address)
addi x31, x31, -4      # Adjust CPUOut and CPUIn memory location

# Load values from memory
lw x8, 0x0(x31)        # Load A from CPUIn -> x8 (Multiplicand)
lw x9, 0x0(x31)        # Load B from CPUIn -> x9 (Multiplier)

# Initialize variables
addi x10, x0, 0        # x10 = P (Product), initially 0
addi x11, x0, 0        # x11 = i (Loop Counter)
addi x12, x0, 1        # x12 = constant 1 (for bit checking)
addi x13, x0, 8        # x13 = constant 8 (loop limit)

# Start of multiplication loop
mul_loop:
and x14, x9, x12       # Check if LSB of B (x9) is 1
beq x14, x0, skip_add  # If B[0] == 0, skip addition
add x10, x10, x8       # P = P + A

skip_add:
slli x8, x8, 1         # A = A << 1 (Shift Left)
srli x9, x9, 1         # B = B >> 1 (Shift Right)
addi x11, x11, 1       # i += 1
bne x11, x13, mul_loop # If i < 8, repeat loop

# Store result
sw x10, 0x0(x31)       # Store result in CPUOut
```

Emultor result (3*9=27)



Input your RISC-V code here:

```
# Initialize CPUIn and CPUOut
lui x31, 0x80000        # Load upper immediate (Base Address)
addi x31, x31, -4       # Adjust CPUOut and CPUIn memory location

# Load values from memory
addi x8,x0, 9         # Load A from CPUIn -> x8 (Multiplicand)
addi x9,x0,3        # Load B from CPUIn -> x9 (Multiplier)

# Initialize variables
addi x10, x0, 0         # x10 = P (Product), initially 0
addi x11, x0, 0         # x11 = i (Loop Counter)
addi x12, x0, 1         # x12 = constant 1 (for bit checking)
addi x13, x0, 8         # x13 = constant 8 (loop limit)

# Start of multiplication loop
```

Init Value	Register	Decimal	Hex	Binary
0	x0 (zero)	0	0x00000000	0b00000000000000000000000000000000
0	x1 (ra)	0	0x00000000	0b00000000000000000000000000000000
0	x2 (sp)	0	0x00000000	0b00000000000000000000000000000000
0	x3 (gp)	0	0x00000000	0b00000000000000000000000000000000
0	x4 (tp)	0	0x00000000	0b00000000000000000000000000000000
0	x5 (t0)	0	0x00000000	0b00000000000000000000000000000000
0	x6 (t1)	0	0x00000000	0b00000000000000000000000000000000
0	x7 (t2)	0	0x00000000	0b00000000000000000000000000000000
0	x8 (s0/fp)	2304	0x00000900	0b00000000000000000000100100000000
0	x9 (s1)	0	0x00000000	0b00000000000000000000000000000000
0	x10 (a0)	27	0x0000001b	0b00000000000000000000000000011011
0	x11 (a1)	8	0x00000008	0b00000000000000000000000000001000
0	x12 (a2)	1	0x00000001	0b00000000000000000000000000000001
0	x13 (a3)	8	0x00000008	0b00000000000000000000000000001000

machine code:

80000fb7

ffcf8f93

000fa403

000fa483

00000513

00000593

00100613

00800693

00c4f733

00070463

00850533

00141413

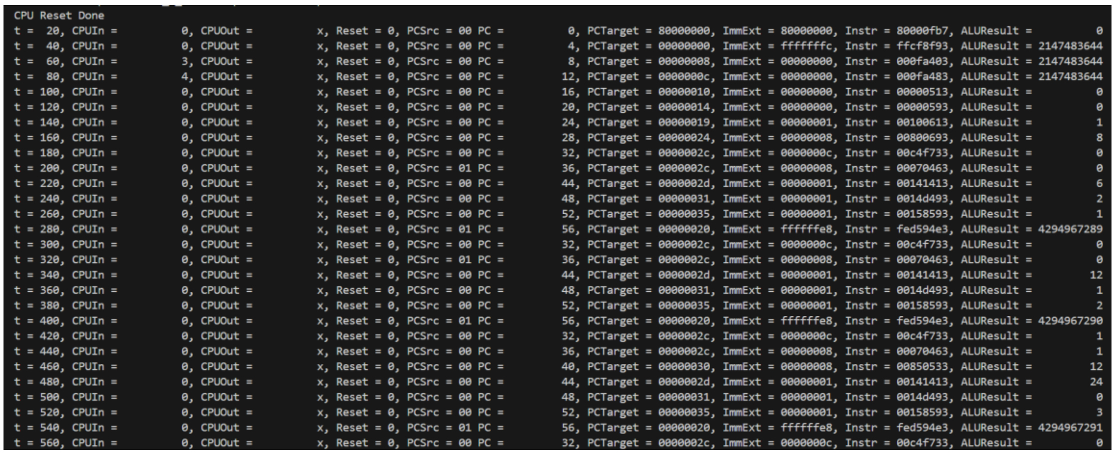
0014d493

00158593

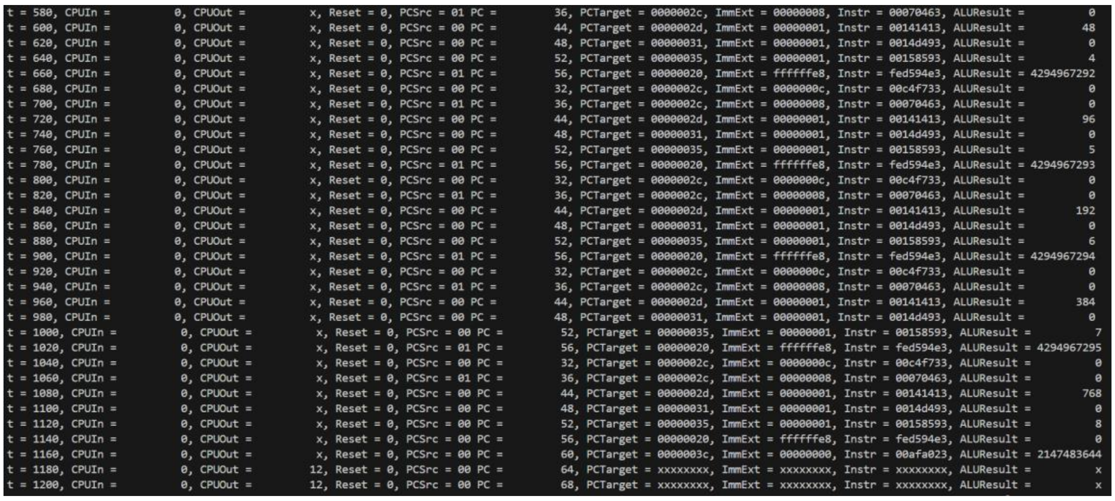
fed594e3

00afa023

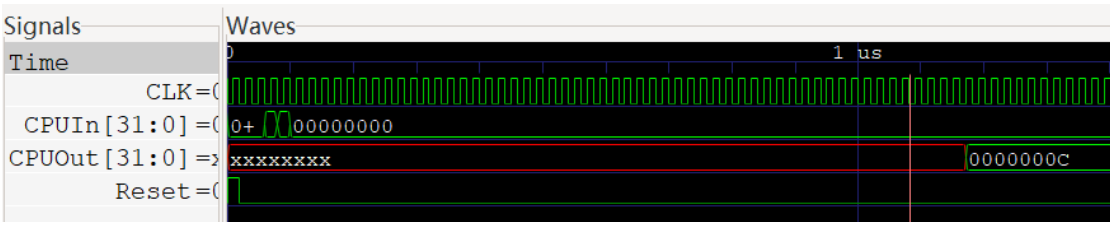
Riscv_tb simulation result:



```
CPU Reset Done
t =  20, CPUIn =          0, CPUOut =          x, Reset = 0, PCSrc = 00 PC =          0, PCTarget = 80000000, ImmExt = 80000000, Instr = 80000fb7, ALUResult =          0
t =  40, CPUIn =          0, CPUOut =          x, Reset = 0, PCSrc = 00 PC =          4, PCTarget = 00000000, ImmExt = fffffffc, Instr = ffcf8f93, ALUResult = 2147483644
t =  60, CPUIn =          3, CPUOut =          x, Reset = 0, PCSrc = 00 PC =          8, PCTarget = 00000008, ImmExt = 00000000, Instr = 000fa403, ALUResult = 2147483644
t =  80, CPUIn =          4, CPUOut =          x, Reset = 0, PCSrc = 00 PC =         12, PCTarget = 0000000c, ImmExt = 00000000, Instr = 000fa483, ALUResult = 2147483644
t = 100, CPUIn =          0, CPUOut =          x, Reset = 0, PCSrc = 00 PC =         16, PCTarget = 00000010, ImmExt = 00000000, Instr = 00000513, ALUResult =          0
t = 120, CPUIn =          0, CPUOut =          x, Reset = 0, PCSrc = 00 PC =         20, PCTarget = 00000014, ImmExt = 00000000, Instr = 00000593, ALUResult =          0
t = 140, CPUIn =          0, CPUOut =          x, Reset = 0, PCSrc = 00 PC =         24, PCTarget = 00000019, ImmExt = 00000001, Instr = 00100613, ALUResult =          1
t = 160, CPUIn =          0, CPUOut =          x, Reset = 0, PCSrc = 00 PC =         28, PCTarget = 00000024, ImmExt = 00000008, Instr = 00800693, ALUResult =          8
t = 180, CPUIn =          0, CPUOut =          x, Reset = 0, PCSrc = 00 PC =         32, PCTarget = 0000002c, ImmExt = 0000000c, Instr = 00c4f733, ALUResult =          0
t = 200, CPUIn =          0, CPUOut =          x, Reset = 0, PCSrc = 01 PC =         36, PCTarget = 0000002c, ImmExt = 00000008, Instr = 00070463, ALUResult =          0
t = 220, CPUIn =          0, CPUOut =          x, Reset = 0, PCSrc = 00 PC =         44, PCTarget = 0000002d, ImmExt = 00000001, Instr = 00141413, ALUResult =          6
t = 240, CPUIn =          0, CPUOut =          x, Reset = 0, PCSrc = 00 PC =         48, PCTarget = 00000031, ImmExt = 00000001, Instr = 0014d493, ALUResult =          2
t = 260, CPUIn =          0, CPUOut =          x, Reset = 0, PCSrc = 00 PC =         52, PCTarget = 00000035, ImmExt = 00000001, Instr = 00158593, ALUResult =          1
t = 280, CPUIn =          0, CPUOut =          x, Reset = 0, PCSrc = 01 PC =         56, PCTarget = 00000020, ImmExt = ffffffe8, Instr = fed594e3, ALUResult = 4294967289
t = 300, CPUIn =          0, CPUOut =          x, Reset = 0, PCSrc = 00 PC =         32, PCTarget = 0000002c, ImmExt = 0000000c, Instr = 00c4f733, ALUResult =          0
t = 320, CPUIn =          0, CPUOut =          x, Reset = 0, PCSrc = 01 PC =         36, PCTarget = 0000002c, ImmExt = 00000008, Instr = 00070463, ALUResult =          0
t = 340, CPUIn =          0, CPUOut =          x, Reset = 0, PCSrc = 00 PC =         44, PCTarget = 0000002d, ImmExt = 00000001, Instr = 00141413, ALUResult =         12
t = 360, CPUIn =          0, CPUOut =          x, Reset = 0, PCSrc = 00 PC =         48, PCTarget = 00000031, ImmExt = 00000001, Instr = 0014d493, ALUResult =          1
t = 380, CPUIn =          0, CPUOut =          x, Reset = 0, PCSrc = 00 PC =         52, PCTarget = 00000035, ImmExt = 00000001, Instr = 00158593, ALUResult =          2
t = 400, CPUIn =          0, CPUOut =          x, Reset = 0, PCSrc = 01 PC =         56, PCTarget = 00000020, ImmExt = ffffffe8, Instr = fed594e3, ALUResult = 4294967290
t = 420, CPUIn =          0, CPUOut =          x, Reset = 0, PCSrc = 00 PC =         32, PCTarget = 0000002c, ImmExt = 0000000c, Instr = 00c4f733, ALUResult =          1
t = 440, CPUIn =          0, CPUOut =          x, Reset = 0, PCSrc = 00 PC =         36, PCTarget = 0000002c, ImmExt = 00000008, Instr = 00070463, ALUResult =          1
t = 460, CPUIn =          0, CPUOut =          x, Reset = 0, PCSrc = 00 PC =         40, PCTarget = 00000030, ImmExt = 00000008, Instr = 00850533, ALUResult =         12
t = 480, CPUIn =          0, CPUOut =          x, Reset = 0, PCSrc = 00 PC =         44, PCTarget = 0000002d, ImmExt = 00000001, Instr = 00141413, ALUResult =         24
t = 500, CPUIn =          0, CPUOut =          x, Reset = 0, PCSrc = 00 PC =         48, PCTarget = 00000031, ImmExt = 00000001, Instr = 0014d493, ALUResult =          0
t = 520, CPUIn =          0, CPUOut =          x, Reset = 0, PCSrc = 00 PC =         52, PCTarget = 00000035, ImmExt = 00000001, Instr = 00158593, ALUResult =          3
t = 540, CPUIn =          0, CPUOut =          x, Reset = 0, PCSrc = 01 PC =         56, PCTarget = 00000020, ImmExt = ffffffe8, Instr = fed594e3, ALUResult = 4294967291
t = 560, CPUIn =          0, CPUOut =          x, Reset = 0, PCSrc = 00 PC =         32, PCTarget = 0000002c, ImmExt = 0000000c, Instr = 00c4f733, ALUResult =          0
```



```
t = 580, CPUIn =          0, CPUOut =          x, Reset = 0, PCSrc = 01 PC =         36, PCTarget = 0000002c, ImmExt = 00000008, Instr = 00070463, ALUResult =          0
t = 600, CPUIn =          0, CPUOut =          x, Reset = 0, PCSrc = 00 PC =         44, PCTarget = 0000002d, ImmExt = 00000001, Instr = 00141413, ALUResult =         48
t = 620, CPUIn =          0, CPUOut =          x, Reset = 0, PCSrc = 00 PC =         48, PCTarget = 00000031, ImmExt = 00000001, Instr = 0014d493, ALUResult =          0
t = 640, CPUIn =          0, CPUOut =          x, Reset = 0, PCSrc = 00 PC =         52, PCTarget = 00000035, ImmExt = 00000001, Instr = 00158593, ALUResult =          4
t = 660, CPUIn =          0, CPUOut =          x, Reset = 0, PCSrc = 01 PC =         56, PCTarget = 00000020, ImmExt = ffffffe8, Instr = fed594e3, ALUResult = 4294967292
t = 680, CPUIn =          0, CPUOut =          x, Reset = 0, PCSrc = 00 PC =         32, PCTarget = 0000002c, ImmExt = 0000000c, Instr = 00c4f733, ALUResult =          0
t = 700, CPUIn =          0, CPUOut =          x, Reset = 0, PCSrc = 01 PC =         36, PCTarget = 0000002c, ImmExt = 00000008, Instr = 00070463, ALUResult =          0
t = 720, CPUIn =          0, CPUOut =          x, Reset = 0, PCSrc = 00 PC =         44, PCTarget = 0000002d, ImmExt = 00000001, Instr = 00141413, ALUResult =         96
t = 740, CPUIn =          0, CPUOut =          x, Reset = 0, PCSrc = 00 PC =         48, PCTarget = 00000031, ImmExt = 00000001, Instr = 0014d493, ALUResult =          0
t = 760, CPUIn =          0, CPUOut =          x, Reset = 0, PCSrc = 00 PC =         52, PCTarget = 00000035, ImmExt = 00000001, Instr = 00158593, ALUResult =          5
t = 780, CPUIn =          0, CPUOut =          x, Reset = 0, PCSrc = 01 PC =         56, PCTarget = 00000020, ImmExt = ffffffe8, Instr = fed594e3, ALUResult = 4294967293
t = 800, CPUIn =          0, CPUOut =          x, Reset = 0, PCSrc = 00 PC =         32, PCTarget = 0000002c, ImmExt = 0000000c, Instr = 00c4f733, ALUResult =          0
t = 820, CPUIn =          0, CPUOut =          x, Reset = 0, PCSrc = 01 PC =         36, PCTarget = 0000002c, ImmExt = 00000008, Instr = 00070463, ALUResult =          0
t = 840, CPUIn =          0, CPUOut =          x, Reset = 0, PCSrc = 00 PC =         44, PCTarget = 0000002d, ImmExt = 00000001, Instr = 00141413, ALUResult =        192
t = 860, CPUIn =          0, CPUOut =          x, Reset = 0, PCSrc = 00 PC =         48, PCTarget = 00000031, ImmExt = 00000001, Instr = 0014d493, ALUResult =          0
t = 880, CPUIn =          0, CPUOut =          x, Reset = 0, PCSrc = 00 PC =         52, PCTarget = 00000035, ImmExt = 00000001, Instr = 00158593, ALUResult =          6
t = 900, CPUIn =          0, CPUOut =          x, Reset = 0, PCSrc = 01 PC =         56, PCTarget = 00000020, ImmExt = ffffffe8, Instr = fed594e3, ALUResult = 4294967294
t = 920, CPUIn =          0, CPUOut =          x, Reset = 0, PCSrc = 00 PC =         32, PCTarget = 0000002c, ImmExt = 0000000c, Instr = 00c4f733, ALUResult =          0
t = 940, CPUIn =          0, CPUOut =          x, Reset = 0, PCSrc = 01 PC =         36, PCTarget = 0000002c, ImmExt = 00000008, Instr = 00070463, ALUResult =          0
t = 960, CPUIn =          0, CPUOut =          x, Reset = 0, PCSrc = 00 PC =         44, PCTarget = 0000002d, ImmExt = 00000001, Instr = 00141413, ALUResult =        384
t = 980, CPUIn =          0, CPUOut =          x, Reset = 0, PCSrc = 00 PC =         48, PCTarget = 00000031, ImmExt = 00000001, Instr = 0014d493, ALUResult =          0
t = 1000, CPUIn =          0, CPUOut =          x, Reset = 0, PCSrc = 00 PC =         52, PCTarget = 00000035, ImmExt = 00000001, Instr = 00158593, ALUResult =          7
t = 1020, CPUIn =          0, CPUOut =          x, Reset = 0, PCSrc = 01 PC =         56, PCTarget = 00000020, ImmExt = ffffffe8, Instr = fed594e3, ALUResult = 4294967295
t = 1040, CPUIn =          0, CPUOut =          x, Reset = 0, PCSrc = 00 PC =         32, PCTarget = 0000002c, ImmExt = 0000000c, Instr = 00c4f733, ALUResult =          0
t = 1060, CPUIn =          0, CPUOut =          x, Reset = 0, PCSrc = 01 PC =         36, PCTarget = 0000002c, ImmExt = 00000008, Instr = 00070463, ALUResult =          0
t = 1080, CPUIn =          0, CPUOut =          x, Reset = 0, PCSrc = 00 PC =         44, PCTarget = 0000002d, ImmExt = 00000001, Instr = 00141413, ALUResult =        768
t = 1100, CPUIn =          0, CPUOut =          x, Reset = 0, PCSrc = 00 PC =         48, PCTarget = 00000031, ImmExt = 00000001, Instr = 0014d493, ALUResult =          0
t = 1120, CPUIn =          0, CPUOut =          x, Reset = 0, PCSrc = 00 PC =         52, PCTarget = 00000035, ImmExt = 00000001, Instr = 00158593, ALUResult =          8
t = 1140, CPUIn =          0, CPUOut =          x, Reset = 0, PCSrc = 00 PC =         56, PCTarget = 00000020, ImmExt = ffffffe8, Instr = fed594e3, ALUResult =          0
t = 1160, CPUIn =          0, CPUOut =          x, Reset = 0, PCSrc = 00 PC =         60, PCTarget = 0000003c, ImmExt = 00000000, Instr = 00afa023, ALUResult = 2147483644
t = 1180, CPUIn =          0, CPUOut =         12, Reset = 0, PCSrc = 00 PC =         64, PCTarget = xxxxxxxx, ImmExt = xxxxxxxx, Instr = xxxxxxxx, ALUResult =          x
t = 1200, CPUIn =          0, CPUOut =         12, Reset = 0, PCSrc = 00 PC =         68, PCTarget = xxxxxxxx, ImmExt = xxxxxxxx, Instr = xxxxxxxx, ALUResult =          x
```



It is clear that the multiplication result is correct for 3*4=12. The s0 and s1 is input via the CPUIn, which is written in the testbench code.

Task 2.2 Testbench code:

```
`timescale 1ns/1ps
`include "risc_v.sv"

module risc_v_tb;
logic [31:0] CPUOut, CPUIn;
logic Reset, CLK;

risc_v dut(CPUOut, CPUIn, Reset, CLK);
```

```verilog
initial begin // Generate clock signal with 20 ns period
CLK = 0;
forever #10 CLK = ~CLK;
end

initial begin // Apply stimulus

// Enter your code here
$dumpfile("risc_v_tb.vcd");
$dumpvars(0, risc_v_tb);

Reset = 1;
CPUIn = 32'b0;
// Step 2: Release reset
#20 Reset = 0;
$display("CPU Reset Done");
#20 CPUIn = 32'b0;
#20 CPUIn = 32'h00000003;
#20 CPUIn = 32'h00000004;
// Step 3: Run CPU for several clock cycles
repeat (200) begin
    #20;
    CPUIn = 0;
end

// Step 4: End simulation
$display("Simulation complete.");

$finish; // This system tasks ends the simulation
end

always @ (negedge CLK)
$display ("t = %3d, CPUIn = %d, CPUOut = %d, Reset = %b, PCSrc = %b PC
= %d, PCTarget = %h, ImmExt = %h, Instr = %h, ALUResult = %d", $time,
CPUIn, CPUOut, Reset, dut.PCSrc, dut.PC, dut.PCTarget, dut.ImmExt,
dut.Instr, dut.ALUResult);

endmodule
```

Appendix for testbench code

Alu_tb

```
`timescale 1ns/1ps
`include "alu.sv"

module alu_tb;

    // **Inputs**
    logic signed [31:0] SrcA, SrcB;
    logic [4:0] ALUControl;

    // **Outputs**
    logic signed [31:0] ALUResult;
    logic Zero, Negative;

    // **Instantiate ALU module**
    alu dut (
        .ALUResult(ALUResult),
        .Zero(Zero),
        .Negative(Negative),
        .SrcA(SrcA),
        .SrcB(SrcB),
        .ALUControl(ALUControl)
    );

    // **Test sequence**
    initial begin
        $dumpfile("alu_tb.vcd");
        $dumpvars(0, alu_tb);

        // **Test ADD**
        SrcA = 32'd10;
        SrcB = 32'd5;
        ALUControl = 5'b00010;
        #10;
        $display("ADD | SrcA = %0d, SrcB = %0d | ALUResult = %0d",
SrcA, SrcB, ALUResult);

        // **Test SUB**
        SrcA = 32'd20;
        SrcB = 32'd5;
        ALUControl = 5'b11110;
        #10;
        $display("SUB | SrcA = %0d, SrcB = %0d | ALUResult = %0d",
SrcA, SrcB, ALUResult);

        // **Test OR**
        SrcA = 32'h0F0F0F0F;
        SrcB = 32'hF0F0F0F0;
        ALUControl = 5'b00111;
        #10;
        $display("OR  | SrcA = %h, SrcB = %h | ALUResult = %h", SrcA,
SrcB, ALUResult);

        // **Test AND**
        SrcA = 32'h0F0F0F0F;
        SrcB = 32'hF0F0F0F0;
        ALUControl = 5'b00011;
        #10;
        $display("AND | SrcA = %h, SrcB = %h | ALUResult = %h", SrcA,
SrcB, ALUResult);

       // **Test Logical Left Shift (SLL)**
        SrcA = 32'h00000002;
        SrcB = 32'h00000002; // shift by 2
        ALUControl = 5'b00000;
        #10;
        $display("SLL | SrcA = %0d, SrcB = %0d | ALUResult = %0d
(Expected: 8)", SrcA, SrcB, ALUResult);

        // **Test Logical Right Shift (SRL)**
        SrcA = 32'h00000080;
        SrcB = 32'h00000003; // shift by 3
        ALUControl = 5'b10000;
        #10;
        $display("SRL | SrcA = %0d, SrcB = %0d | ALUResult = %0d
(Expected: 16)", SrcA, SrcB, ALUResult);

        // **Test Zero flag**
        SrcA = 32'd10;
        SrcB = 32'd10;
        ALUControl = 5'b11110; // 10 - 10 = 0
        #10;
        $display("ZERO TEST | SrcA = %0d, SrcB = %0d | ALUResult = %0d
| Zero = %b", SrcA, SrcB, ALUResult, Zero);

        // **Test Negative flag**
        SrcA = 32'd5;
        SrcB = 32'd10;
        ALUControl = 5'b11110; // 5 - 10 = -5
        #10;
        $display("NEGATIVE TEST | SrcA = %0d, SrcB = %0d | ALUResult
= %0d | Negative = %b", SrcA, SrcB, ALUResult, Negative);

        // **End simulation**
        #50;
        $finish;
    end

    // **Monitor ALU computations**
    initial begin
        $monitor("t = %3d | ALUControl = %b | SrcA = %0d (%h) | SrcB
= %0d (%h) | ALUResult = %0d (%h) | Zero = %b | Negative = %b",
                  $time, ALUControl, SrcA, SrcA, SrcB, SrcB, ALUResult,
ALUResult, Zero, Negative);
    end

endmodule
```

Control_unit_tb

```
`timescale 1ns/1ps
`include "control_unit.sv"

module control_unit_tb;

  // Inputs
  logic [31:0] Instr;
  logic Zero, Negative;

  // Outputs
  logic [1:0] PCSrc, ResultSrc;
  logic MemWrite, ALUSrc, RegWrite;
  logic [4:0] ALUControl;
  logic [2:0] ImmSrc;

  // Instantiate the DUT (Device Under Test)
  control_unit dut (
    .PCSrc(PCSrc),
    .ResultSrc(ResultSrc),
    .MemWrite(MemWrite),
    .ALUSrc(ALUSrc),
    .RegWrite(RegWrite),
    .ALUControl(ALUControl),
    .ImmSrc(ImmSrc),
    .Instr(Instr),
    .Zero(Zero),
    .Negative(Negative)
  );

  // Test sequence
  initial begin
```

```
    $dumpfile("control_unit_tb.vcd");
    $dumpvars(0, control_unit_tb);

    // Test Case 1: R-type (ADD x5, x6, x7) - opcode: 0110011, funct3:
000, funct7: 0000000
    Instr = 32'b0000000_00110_00111_000_00101_0110011;
    Zero = 0; Negative = 0; #10;
    $display("R-type (ADD) | PCSrc = %b, ALUControl = %b, RegWrite
= %b", PCSrc, ALUControl, RegWrite);

    // Test Case 2: I-type (ADDI x5, x6, 10) - opcode: 0010011, funct3:
000
    Instr = 32'b000000000101_00110_000_00101_0010011;
    Zero = 0; Negative = 0; #10;
    $display("I-type (ADDI) | ALUSrc = %b, ALUControl = %b, RegWrite
= %b", ALUSrc, ALUControl, RegWrite);

    // Test Case 3: Load Word (LW x5, 0(x6)) - opcode: 0000011, funct3:
010
    Instr = 32'b000000000000_00110_010_00101_0000011;
    Zero = 0; Negative = 0; #10;
    $display("LW | ResultSrc = %b, MemWrite = %b, RegWrite = %b",
ResultSrc, MemWrite, RegWrite);

    // Test Case 4: Store Word (SW x5, 0(x6)) - opcode: 0100011,
funct3: 010
    Instr = 32'b0000000_00101_00110_010_00000_0100011;
    Zero = 0; Negative = 0; #10;
    $display("SW | MemWrite = %b, ALUSrc = %b", MemWrite, ALUSrc);

    // Test Case 5: Branch if Equal (BEQ x5, x6, offset) - opcode:
1100011, funct3: 000
    Instr = 32'b0000000_00101_00110_000_00000_1100011;
    Zero = 1; Negative = 0; #10;
    $display("BEQ | PCSrc = %b", PCSrc);

    // Test Case 6: Branch if Less Than (BLT x5, x6, offset) - opcode:
1100011, funct3: 100
    Instr = 32'b0000000_00101_00110_100_00000_1100011;
    Zero = 0; Negative = 1; #10;
    $display("BLT | PCSrc = %b", PCSrc);

    // Test Case 7: JAL (Jump and Link) - opcode: 1101111
    Instr = 32'b00000000000000000000_00000_1101111;
    Zero = 0; Negative = 0; #10;
    $display("JAL | PCSrc = %b, RegWrite = %b", PCSrc, RegWrite);

    // Test Case 8: JALR (Jump and Link Register) - opcode: 1100111
    Instr = 32'b000000000000_00001_000_00000_1100111;
    Zero = 0; Negative = 0; #10;
    $display("JALR | PCSrc = %b, RegWrite = %b", PCSrc, RegWrite);

    // Test Case 9: LUI (Load Upper Immediate) - opcode: 0110111
    Instr = 32'b00000000000000000000_00000_0110111;
    Zero = 0; Negative = 0; #10;
    $display("LUI | ResultSrc = %b, RegWrite = %b", ResultSrc,
RegWrite);

    // End test
    #50;
    $finish;
  end

  // Monitor values
  initial begin
    $monitor("t = %3d | Instr = %b | PCSrc = %b | ResultSrc = %b |
ALUSrc = %b | ALUControl = %b | RegWrite = %b | MemWrite = %b | ImmSrc
= %b",
              $time, Instr, PCSrc, ResultSrc, ALUSrc, ALUControl,
RegWrite, MemWrite, ImmSrc);
  end

endmodule
```

Extend_tb

```
`timescale 1ns/1ps
`include "extend.sv"

module extend_tb;

    // **Inputs**
    logic [31:0] Instr;
    logic [2:0] ImmSrc;

    // **Outputs**
    logic [31:0] ImmExt;

    // **Instantiate Extend module**
    extend dut (
        .ImmExt(ImmExt),
        .Instr(Instr),
        .ImmSrc(ImmSrc)
    );
    // **Test sequence**
    initial begin
        $dumpfile("extend_tb.vcd");
        $dumpvars(0, extend_tb);

        // **Test I-type**
        Instr = 32'h00200013;
        ImmSrc = 3'b000;
        #10;
        $display("I-type | Instr = %h | ImmExt = %h | Expected:
00000002", Instr, ImmExt);

        // **Test I-type Negative**
        Instr = 32'hFFE00013;
        ImmSrc = 3'b000;
        #10;
        $display("I-type (Negative) | Instr = %h | ImmExt = %h |
Expected: FFFFFFFE", Instr, ImmExt);

        // **Test S-type**
        Instr = 32'h00000023;
        ImmSrc = 3'b001;
        #10;
        $display("S-type | Instr = %h | ImmExt = %h | Expected:
00000000", Instr, ImmExt);

        // **Test B-type**
        Instr = 32'hFE000063;
        ImmSrc = 3'b010;
        #10;
        $display("B-type | Instr = %h | ImmExt = %h | Expected:
FFFFF7E0", Instr, ImmExt);

        // **Test U-type**
        Instr = 32'h12345037;
        ImmSrc = 3'b011;
        #10;
        $display("U-type | Instr = %h | ImmExt = %h | Expected:
12345000", Instr, ImmExt);

        // **Test J-type**
        Instr = 32'h0056786F;
        ImmSrc = 3'b100;
        #10;
        $display("J-type | Instr = %h | ImmExt = %h | Expected:
00067804", Instr, ImmExt);

        // **End simulation**
        $finish;
    end

    // **Monitor all test cases**
    initial begin
        $monitor("t = %3d | ImmSrc = %b | Instr = %b | ImmExt = %b",
                  $time, ImmSrc, Instr, ImmExt);
    end

endmodule
```

Program_counter_tb

```
`timescale 1ns/1ps
`include "program_counter.sv"

module program_counter_tb;

  // Inputs
  logic CLK;
  logic Reset;
  logic [1:0] PCSrc;
  logic [31:0] PCTarget, ALUResult;

  // Outputs
  logic [31:0] PC, PCPlus4;

  // Instantiate the `program_counter` module
  program_counter dut (
    .PC(PC),
    .PCPlus4(PCPlus4),
    .PCTarget(PCTarget),
    .ALUResult(ALUResult),
    .PCSrc(PCSrc),
    .Reset(Reset),
    .CLK(CLK)
  );

  // Clock generation (Toggle every 10ns, 20ns period)
  initial begin
    CLK = 0;
    forever #10 CLK = ~CLK;
  end

  // Stimulus
  initial begin
    $dumpfile("program_counter_tb.vcd");
    $dumpvars(0, program_counter_tb);

    // Initialize inputs
    Reset = 1; PCSrc = 2'b00;
    PCTarget = 32'h00000020;  // Example branch target address
    ALUResult = 32'h00000040; // Example JALR computed address

    // Apply Reset
    #20 Reset = 0;

    // Test vector 1: Sequential Execution (PC + 4)
    PCSrc = 2'b00; #20;
    $display("PC = %h, Expected = 4 | PCPlus4 = %h, Expected = 8", PC,
PCPlus4);

    // Test vector 2: Branch Target (PCTarget)
    PCSrc = 2'b01; #20;
    $display("PC = %h, Expected = %h | PCPlus4 = %h (unchanged)", PC,
PCTarget, PCPlus4);

    // Test vector 3: JALR (ALUResult)
    PCSrc = 2'b10; #20;
    $display("PC = %h, Expected = %h | PCPlus4 = %h (unchanged)", PC,
ALUResult, PCPlus4);

    // Test vector 4: Reset
    Reset = 1; #20;
    Reset = 0;
    $display("PC = %h, Expected = 0 | PCPlus4 = 4", PC, PCPlus4);

    // Additional test cases
    #20 PCSrc = 2'b00; #20;
    #20 PCSrc = 2'b01; PCTarget = 32'h00000050; #20;
    #20 PCSrc = 2'b10; ALUResult = 32'h00000080; #20;
    #20 PCSrc = 2'b00; #20;

    // Allow some additional time for simulation
    #100;
    $finish;
  end

  // Monitor PC and other signals
  initial begin
    $monitor("t = %3d, CLK = %b, Reset = %b, PCSrc = %b, PCTarget = %h,
ALUResult = %h, PC = %h, PCPlus4 = %h",
              $time, CLK, Reset, PCSrc, PCTarget, ALUResult, PC,
PCPlus4);
  end
```

```
endmodule
```