SINGLE CYCLE RV-32I

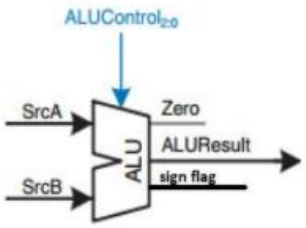
Produced by: Raghad Waleed Mohamed

Main Modules:

1.ALU

ALU.v

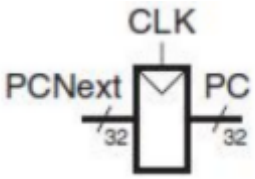
```v
module ALU(
input [31:0] SrcA,SrcB,
output reg SF,ZF,
output reg [31:0]ALUResult,
input [2:0]ALUControl
);

always@(*)
begin
    case (ALUControl)
    3'b000:  ALUResult=SrcA+SrcB;
    3'b001:  ALUResult=SrcA<<SrcB; //shift left
    3'b010:  ALUResult=SrcA-SrcB;
    3'b100:  ALUResult=SrcA^SrcB;
    3'b101:  ALUResult=SrcA>>SrcB; //shift right
    3'b110:  ALUResult=SrcA|SrcB;
    3'b111:  ALUResult=SrcA&SrcB;
    default: begin ALUResult=0; SF=0; ZF=1; end
    endcase
    SF = ALUResult[31];
    if (ALUResult==0) ZF=1;
    else ZF=0;

end

endmodule
```



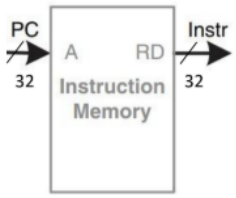
2. Program Counter

PC.v

```v
module PC(
    input [31:0] ImmExt,
    input PCSrc,
    input clk,rst,load,
    output reg [31:0] PC
);

wire [31:0] PCNext;
assign PCNext = PCSrc ? (PC + ImmExt) : (PC + 32'd4);

always @(posedge clk or negedge rst) begin
    if(!rst)
        PC<=32'b0;
    else if(load)
        PC<=PCNext;
    else
        PC<=PC;

end
endmodule
```



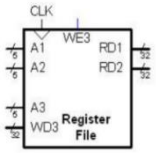
3. Instruction Memory



Imem.v

```
module Imem#(parameter width =32 , depth=64)
(
    input [width-1:0] A,
    output reg [width-1:0] RD
);
reg [width-1:0] mem [0:depth-1]; //64 registers,each 32 bit wide

initial begin
    $readmemh("program.txt", mem);
end

always @(*) begin
    RD = mem[A[7:2]];      //RD = mem[A[31:2]];
end
endmodule

```

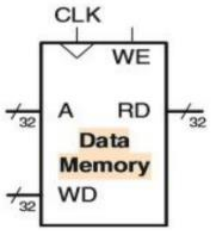
4. Register File



Register_File.v

```
module Register_File( //32 registers,Each register is 32 bits wide
    input clk,rst,
    input [4:0] A1,A2,A3,
    input [31:0]WD3,
    input WE3,
    output reg [31:0]RD1,RD2
);
reg [31:0] registers[0:31];

always @(*) begin // Asynchronous read ports
    RD1 =registers[A1];
    RD2 =registers[A2];
end

integer i;
always @(posedge clk or negedge rst) begin // Synchronous write with asynchronous reset
    if (!rst) begin
        for (i = 0; i < 32; i = i + 1)
            registers[i] <= 32'b0;
    end
    else if (WE3) begin  // Write on rising clock edge
        registers[A3] <= WD3;
    end
end
endmodule
```

5. Data Memory



Data_mem.v

```
module Data_mem(
    input clk,
    input WE,
    input [31:0]A,WD,
    output reg [31:0]RD
);
reg [31:0] Dmem [0:63];    // Depth=64, Width=32

always @(*) begin
    RD=Dmem[A[7:2]];  //read
end

always @(posedge clk) begin
    if (WE)
        Dmem[A[7:2]] <= WD;
end
endmodule
```

6. Control Unit

CU.v

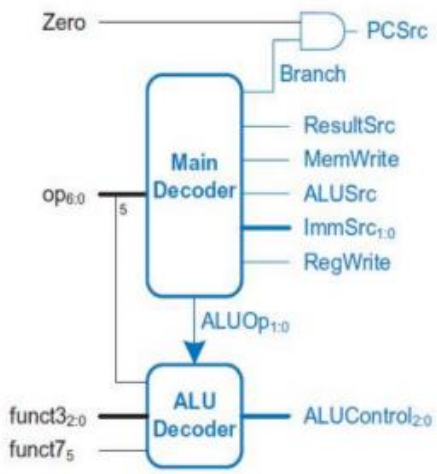
```
module CU(
    input [6:0]op,
    input [2:0]funct3,
    input funct7,ZF,SF,
    output reg RegWrite,ALUSrc,MemWrite,ResultSrc,PCsrc,
    output reg [1:0]ImmSrc,
    output reg [2:0]ALUControl
);

reg Branch;
reg [1:0] ALUOp;

```

6.1 Main Decoder

```
13  always @(*) begin
14      case (op)
15
16      7'b0000011: begin //loadWord
17      RegWrite = 1'b1;
18      ImmSrc = 2'b00;
19      ALUSrc = 1'b1;
20      MemWrite = 1'b0;
21      ResultSrc = 1'b1;
22      Branch = 1'b0;
23      ALUOp = 2'b00;
24      end
25
26      7'b0100011: begin //storeWord
27      RegWrite = 1'b0;
28      ImmSrc = 2'b01;
29      ALUSrc = 1'b1;
30      MemWrite = 1'b1;
31      Branch = 1'b0;
32      ALUOp = 2'b00;
33      end
34
35      7'b0110011: begin //R-Type
36      RegWrite = 1'b1;
37      ALUSrc = 1'b0;
38      MemWrite = 1'b0;
39      ResultSrc = 1'b0;
40      Branch = 1'b0;
41      ALUOp = 2'b10;
42      end
43
44      7'b0010011: begin //I-Type
45      RegWrite = 1'b1;
46      ImmSrc = 2'b00;
47      ALUSrc = 1'b1;
48      MemWrite = 1'b0;
49      ResultSrc = 1'b0;
50      Branch = 1'b0;
51      ALUOp = 2'b10;
52      end
53
54      7'b1100011: begin //branch-instructions
55      RegWrite = 1'b0;
56      ImmSrc = 2'b10;
57      ALUSrc = 1'b0;
58      MemWrite = 1'b0;
59      Branch = 1'b1;
60      ALUOp = 2'b01;
61      end
62
63      default: begin //default values
64      RegWrite = 1'b0; ImmSrc = 2'b00; ALUSrc = 1'b0; MemWrite = 1'b0;
65      ResultSrc = 1'b0; Branch = 1'b0; ALUOp = 2'b00;
66      end
67
68      endcase
69  end
70
```

6.2 ALU Decoder

```
71  //ALU decoder
72  always @(*) begin
73      case(ALUOp)
74
75      2'b00: ALUControl=3'b000; //add --> lw,sw
76
77      2'b01: ALUControl=3'b010; // sub --> branches
78
79      2'b10:begin
80          case (funct3)
81              3'b000: ALUControl = (funct7 == 1 && op[5]==1) ? 3'b010:3'b000 ; // add or sub
82              3'b001: ALUControl = 3'b001; // shift left
83              3'b100: ALUControl = 3'b100; // xor
84              3'b101: ALUControl = 3'b101; // shift right
85              3'b110: ALUControl = 3'b110; // or
86              3'b111: ALUControl = 3'b111; // and
87
88              default: ALUControl = 3'b000;
89          endcase
90      end
91
92      endcase
93  end
94
95  always @(*) begin
96      case (funct3)
97          3'b000: PCsrc = Branch & ZF;     // BEQ
98          3'b001: PCsrc = Branch & ~ZF;    // BNE
99          3'b100: PCsrc = Branch & SF;     // BLT
100         default: PCsrc = 0;
101     endcase
102 end
103
104 endmodule
```



Small Modules:

1. Sign extend

Sign_Extend.v

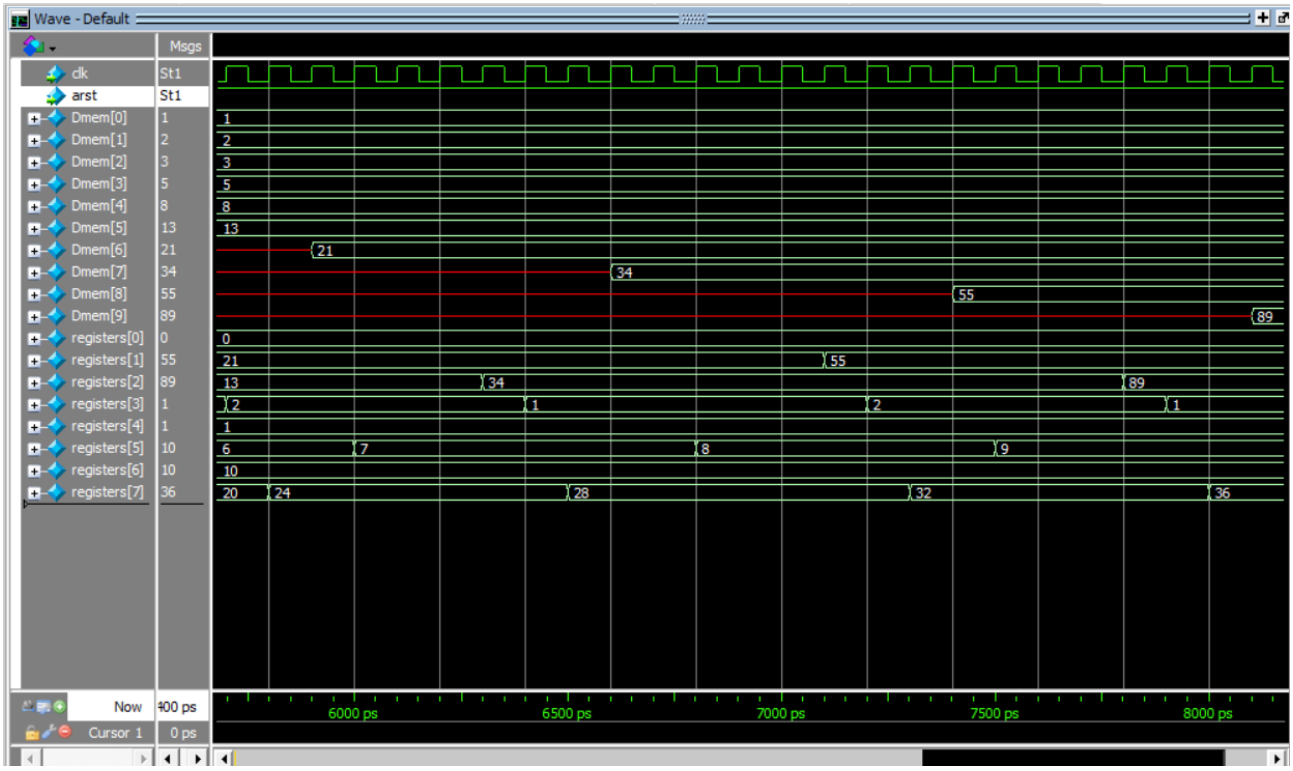
```
module Sign_Extend(
    input [1:0]ImmSrc,
    input [24:0] Instr,
    output reg [31:0] ImmExt
);
always @(*) begin
    case (ImmSrc)
        2'b00: ImmExt = {{20{Instr[24]}}, Instr[24:13]};  // I-type
        2'b01: ImmExt = {{20{Instr[24]}}, Instr[24:18], Instr[4:0]}; // S-type
        2'b10: ImmExt = {{20{Instr[24]}},Instr[0],Instr[24:18], Instr[4:1], 1'b0}; // B-type

        default:  ImmExt = 32'b0;
    endcase
end
endmodule
```

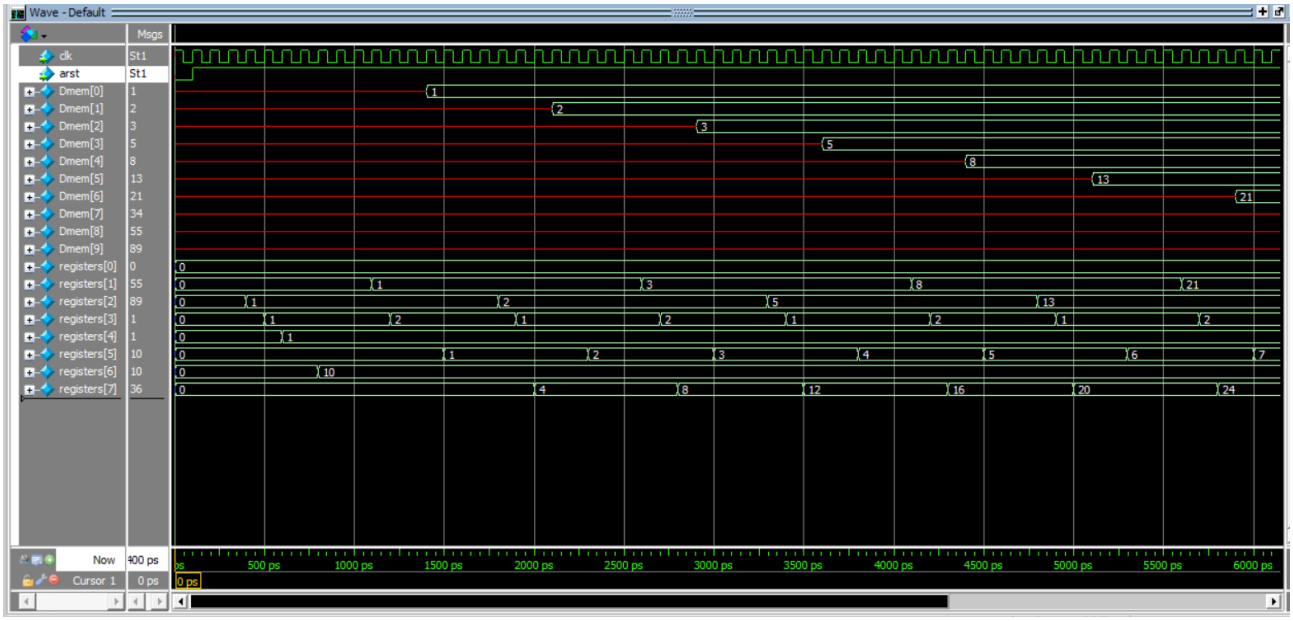
Top Module:

RV.v

```
module RV(input clk,arst);
wire [31:0] SrcA,SrcB,ALUResult,ImmExt,PC,Instr,Result,WriteData,ReadData;
wire ZF,SF,PCSrc,RegWrite,MemWrite,ALUSrc,ResultSrc,load;
wire[2:0] ALUControl;
wire[1:0]ImmSrc;


ALU top_ALU(.SrcA(SrcA),.SrcB(SrcB),.SF(SF),.ZF(ZF),
.ALUResult(ALUResult),.ALUControl(ALUControl));

PC top_PC(.ImmExt(ImmExt),.PCSrc(PCSrc),
.clk(clk),.rst(arst),.load(1'b1),.PC(PC));

Imem top_Imem(.A(PC),.RD(Instr));

Register_File top_Register_File(
    .clk(clk),
    .rst(arst),
    .A1(Instr[19:15]),
    .A2(Instr[24:20]),
    .A3(Instr[11:7]),
    .WD3(Result),
    .WE3(RegWrite),
    .RD1(SrcA),
    .RD2(WriteData));

Data_mem top_Data_mem(.clk(clk),.WE(MemWrite),.A(ALUResult),
.WD(WriteData),.RD(ReadData));

CU top_CU(
    .op(Instr[6:0]),
    .funct3(Instr[14:12]),
    .funct7(Instr[30]),
    .ZF(ZF),.SF(SF),
    .RegWrite(RegWrite),
    .ALUSrc(ALUSrc),
    .MemWrite(MemWrite),
    .ResultSrc(ResultSrc),
    .PCsrc(PCSrc),
    .ImmSrc(ImmSrc),
    .ALUControl(ALUControl));

Sign_Extend top_Sign_Extend(.ImmSrc(ImmSrc),
.Instr(Instr[31:7]),.ImmExt(ImmExt));

//mux
assign Result= ResultSrc? ReadData : ALUResult;
assign SrcB= ALUSrc? ImmExt : WriteData;

endmodule
```

Simulation:






```
# Compile of ALU.v was successful.
# Compile of ALU_tb.v was successful.
# Compile of CU.v was successful.
# Compile of Data_mem.v was successful.
# Compile of Imem.v was successful.
# Compile of PC.v was successful.
# Compile of Register_File.v was successful.
# Compile of RV.v was successful.
# Compile of Sign_Extend.v was successful.
# 9 compiles, 0 failed with no errors.
```