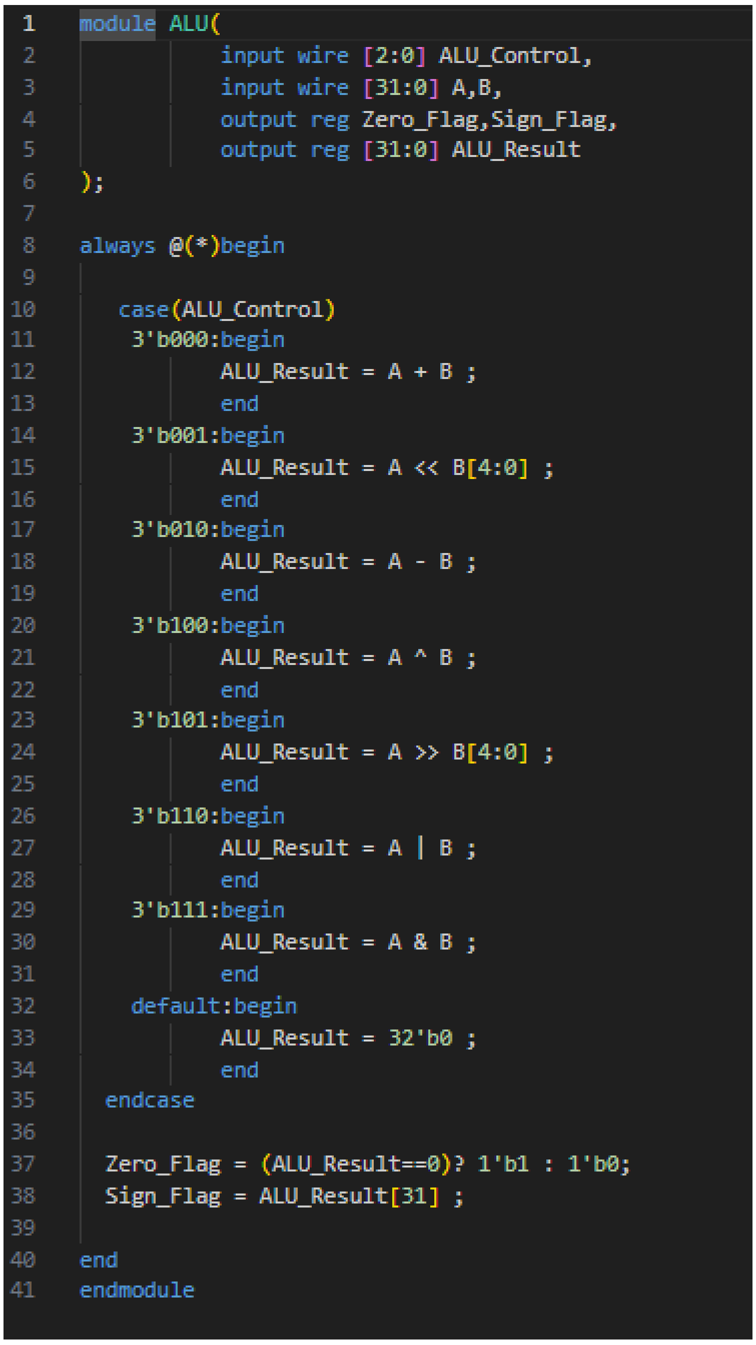
FINAL PROJECT

RISC_V

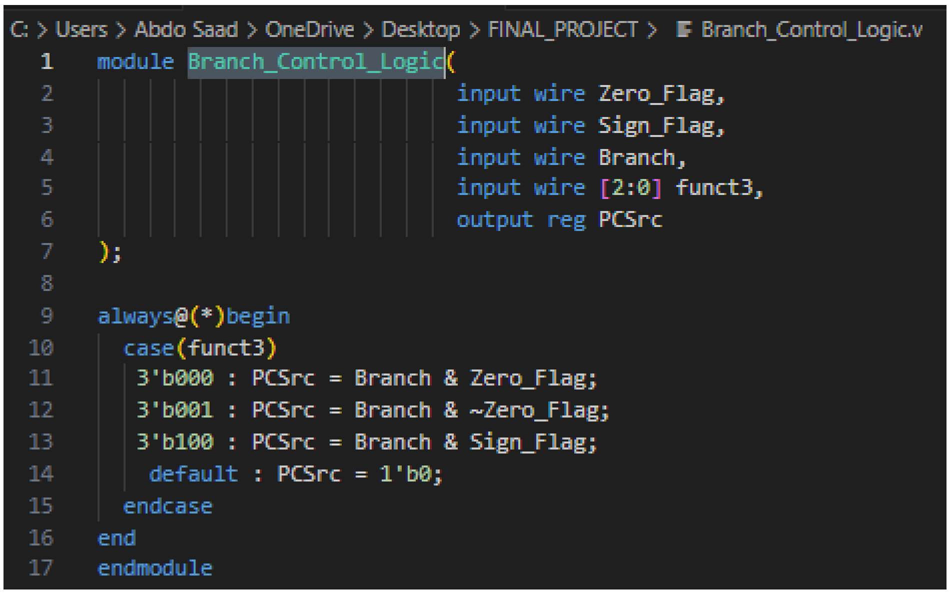
الإسم
عبدالرحمن سعد سمهودي

ALU



```verilog
module ALU(
                input wire [2:0] ALU_Control,
                input wire [31:0] A,B,
                output reg Zero_Flag,Sign_Flag,
                output reg [31:0] ALU_Result
);

always @(*)begin

    case(ALU_Control)
     3'b000:begin
                ALU_Result = A + B ;
                end
     3'b001:begin
                ALU_Result = A << B[4:0] ;
                end
     3'b010:begin
                ALU_Result = A - B ;
                end
     3'b100:begin
                ALU_Result = A ^ B ;
                end
     3'b101:begin
                ALU_Result = A >> B[4:0] ;
                end
     3'b110:begin
                ALU_Result = A | B ;
                end
     3'b111:begin
                ALU_Result = A & B ;
                end
     default:begin
                ALU_Result = 32'b0 ;
                end
   endcase

   Zero_Flag = (ALU_Result==0)? 1'b1 : 1'b0;
   Sign_Flag = ALU_Result[31] ;

end
endmodule
```

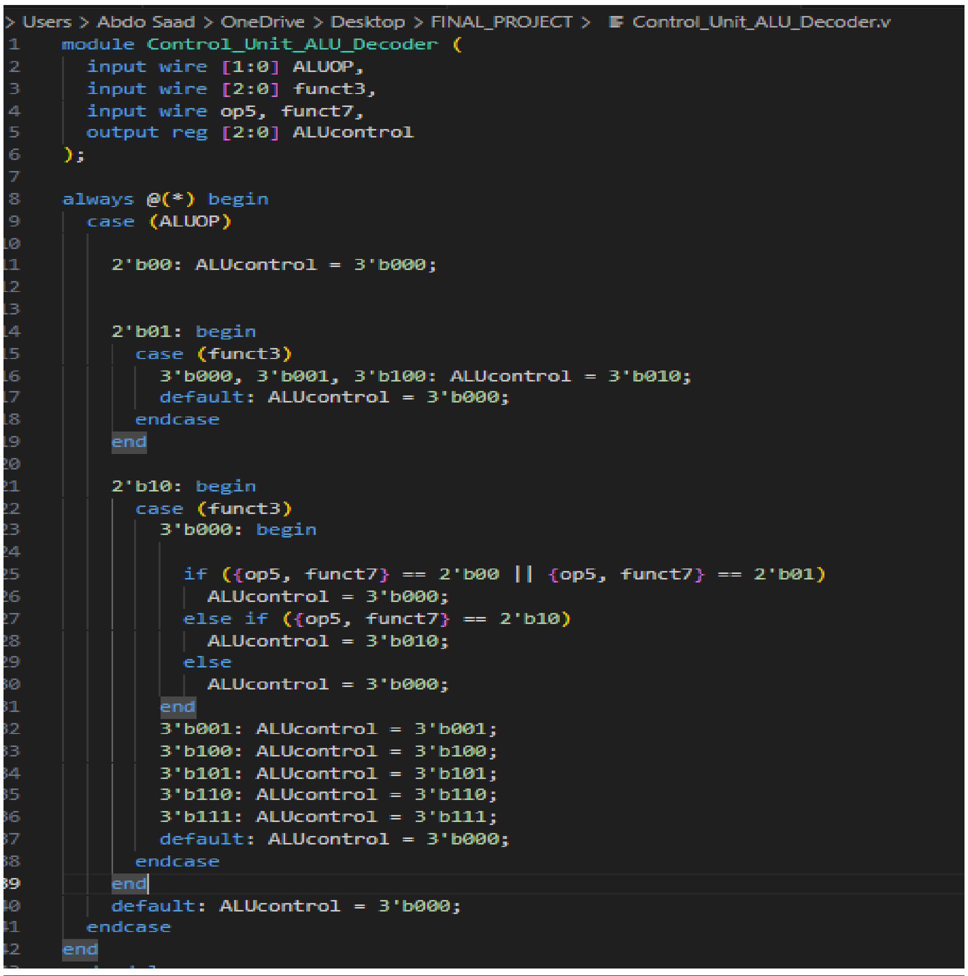
Branch_Control_Logic



C: > Users > Abdo Saad > OneDrive > Desktop > FINAL_PROJECT > Branch_Control_Logic.v

```verilog
module Branch_Control_Logic(
                              input wire Zero_Flag,
                              input wire Sign_Flag,
                              input wire Branch,
                              input wire [2:0] funct3,
                              output reg PCSrc
);

always@(*)begin
  case(funct3)
   3'b000 : PCSrc = Branch & Zero_Flag;
   3'b001 : PCSrc = Branch & ~Zero_Flag;
   3'b100 : PCSrc = Branch & Sign_Flag;
    default : PCSrc = 1'b0;
  endcase
end
endmodule
```

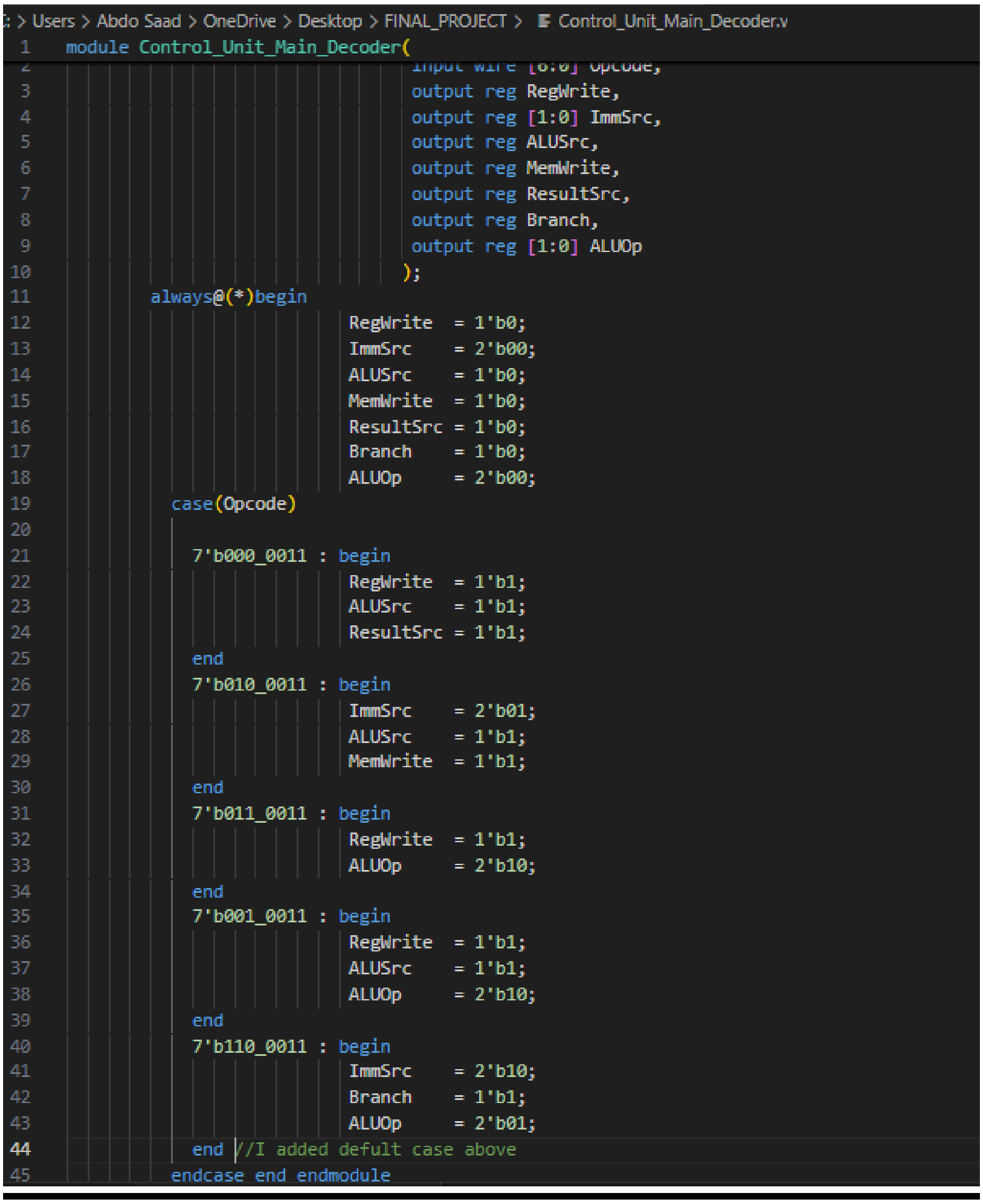
Control_Unit_ALU_Decoder



> Users > Abdo Saad > OneDrive > Desktop > FINAL_PROJECT > Control_Unit_ALU_Decoder.v

```verilog
module Control_Unit_ALU_Decoder (
  input wire [1:0] ALUOP,
  input wire [2:0] funct3,
  input wire op5, funct7,
  output reg [2:0] ALUcontrol
);

always @(*) begin
  case (ALUOP)

    2'b00: ALUcontrol = 3'b000;


    2'b01: begin
      case (funct3)
        3'b000, 3'b001, 3'b100: ALUcontrol = 3'b010;
        default: ALUcontrol = 3'b000;
      endcase
    end

    2'b10: begin
      case (funct3)
        3'b000: begin

          if ({op5, funct7} == 2'b00 || {op5, funct7} == 2'b01)
            ALUcontrol = 3'b000;
          else if ({op5, funct7} == 2'b10)
            ALUcontrol = 3'b010;
          else
            ALUcontrol = 3'b000;
        end
        3'b001: ALUcontrol = 3'b001;
        3'b100: ALUcontrol = 3'b100;
        3'b101: ALUcontrol = 3'b101;
        3'b110: ALUcontrol = 3'b110;
        3'b111: ALUcontrol = 3'b111;
        default: ALUcontrol = 3'b000;
      endcase
    end
    default: ALUcontrol = 3'b000;
  endcase
end
```

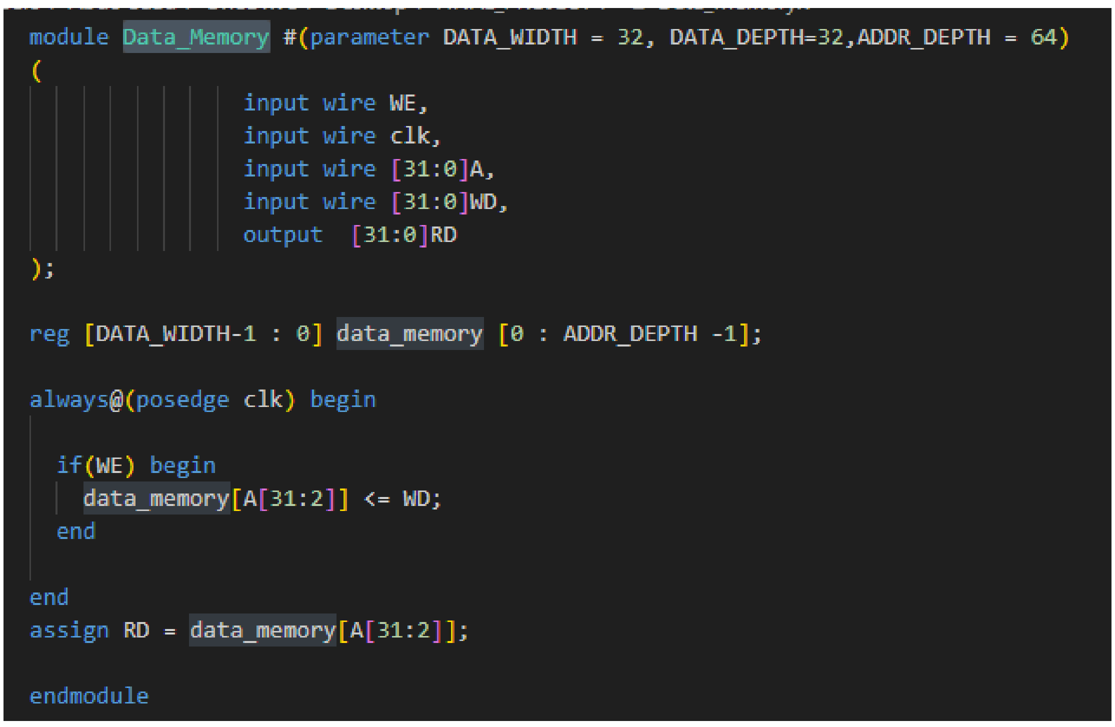
Control_Unit_Main_Decoder



: > Users > Abdo Saad > OneDrive > Desktop > FINAL_PROJECT > Control_Unit_Main_Decoder.v

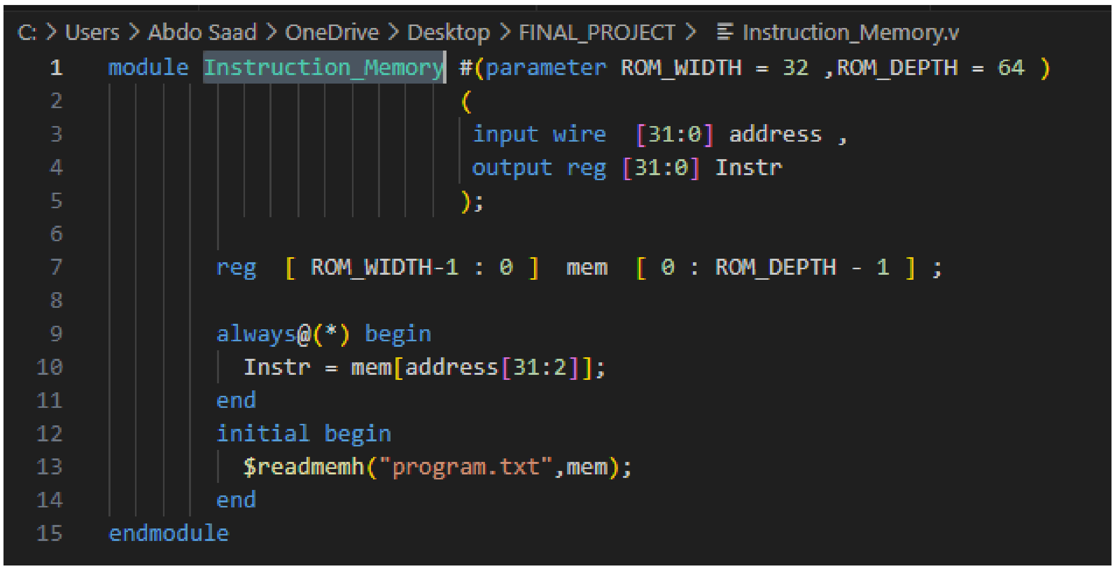
```verilog
module Control_Unit_Main_Decoder(
                        input wire [6:0] Opcode,
                        output reg RegWrite,
                        output reg [1:0] ImmSrc,
                        output reg ALUSrc,
                        output reg MemWrite,
                        output reg ResultSrc,
                        output reg Branch,
                        output reg [1:0] ALUOp
                        );
        always@(*)begin
                    RegWrite  = 1'b0;
                    ImmSrc    = 2'b00;
                    ALUSrc    = 1'b0;
                    MemWrite  = 1'b0;
                    ResultSrc = 1'b0;
                    Branch    = 1'b0;
                    ALUOp     = 2'b00;
          case(Opcode)

            7'b000_0011 : begin
                    RegWrite  = 1'b1;
                    ALUSrc    = 1'b1;
                    ResultSrc = 1'b1;
            end
            7'b010_0011 : begin
                    ImmSrc    = 2'b01;
                    ALUSrc    = 1'b1;
                    MemWrite  = 1'b1;
            end
            7'b011_0011 : begin
                    RegWrite  = 1'b1;
                    ALUOp     = 2'b10;
            end
            7'b001_0011 : begin
                    RegWrite  = 1'b1;
                    ALUSrc    = 1'b1;
                    ALUOp     = 2'b10;
            end
            7'b110_0011 : begin
                    ImmSrc    = 2'b10;
                    Branch    = 1'b1;
                    ALUOp     = 2'b01;
            end //I added defult case above
          endcase end endmodule
```

Data_Memory



```verilog
module Data_Memory #(parameter DATA_WIDTH = 32, DATA_DEPTH=32,ADDR_DEPTH = 64)
(
                input wire WE,
                input wire clk,
                input wire [31:0]A,
                input wire [31:0]WD,
                output  [31:0]RD
);

reg [DATA_WIDTH-1 : 0] data_memory [0 : ADDR_DEPTH -1];

always@(posedge clk) begin

  if(WE) begin
    data_memory[A[31:2]] <= WD;
  end

end
assign RD = data_memory[A[31:2]];

endmodule
```

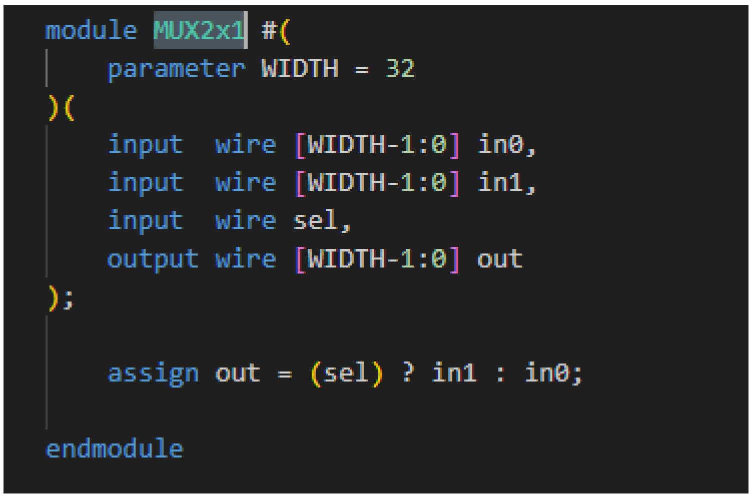
Instruction_Memory



C: > Users > Abdo Saad > OneDrive > Desktop > FINAL_PROJECT > Instruction_Memory.v

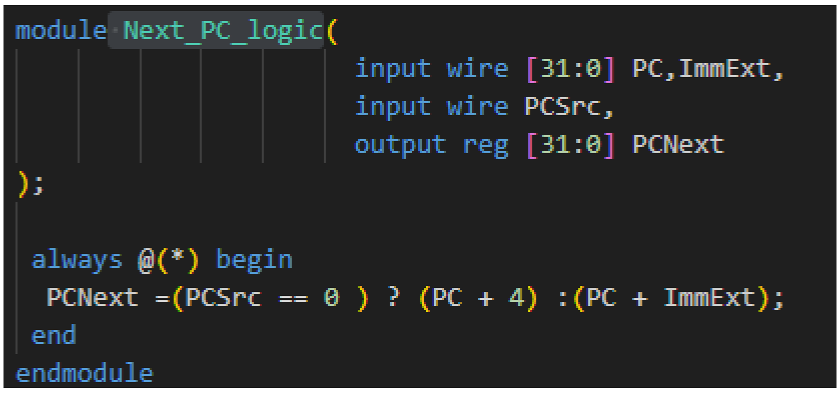
```verilog
module Instruction_Memory #(parameter ROM_WIDTH = 32 ,ROM_DEPTH = 64 )
                         (
                          input wire  [31:0] address ,
                          output reg [31:0] Instr
                         );

        reg  [ ROM_WIDTH-1 : 0 ]  mem  [ 0 : ROM_DEPTH - 1 ] ;

        always@(*) begin
          Instr = mem[address[31:2]];
        end
        initial begin
          $readmemh("program.txt",mem);
        end
endmodule
```

MUX2x1



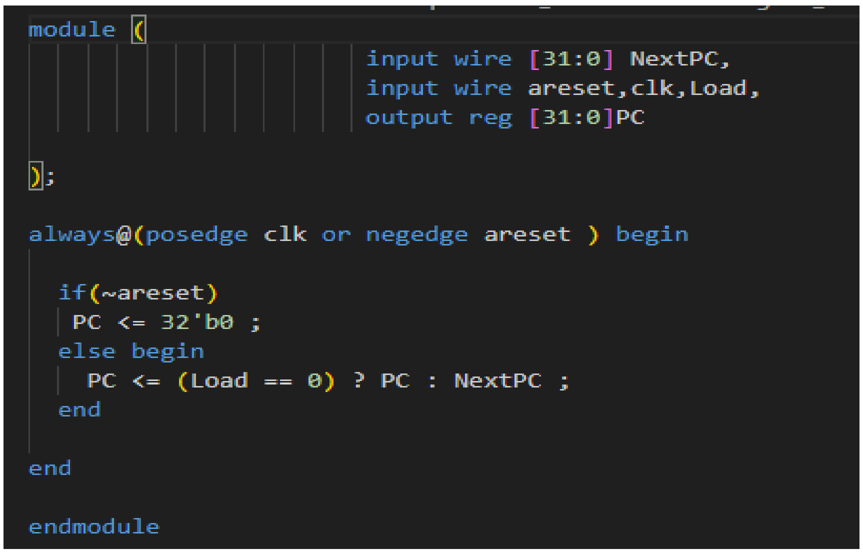
```
module MUX2x1 #(
    parameter WIDTH = 32
)(
    input  wire [WIDTH-1:0] in0,
    input  wire [WIDTH-1:0] in1,
    input  wire sel,
    output wire [WIDTH-1:0] out
);

    assign out = (sel) ? in1 : in0;

endmodule
```

Next_PC_logic



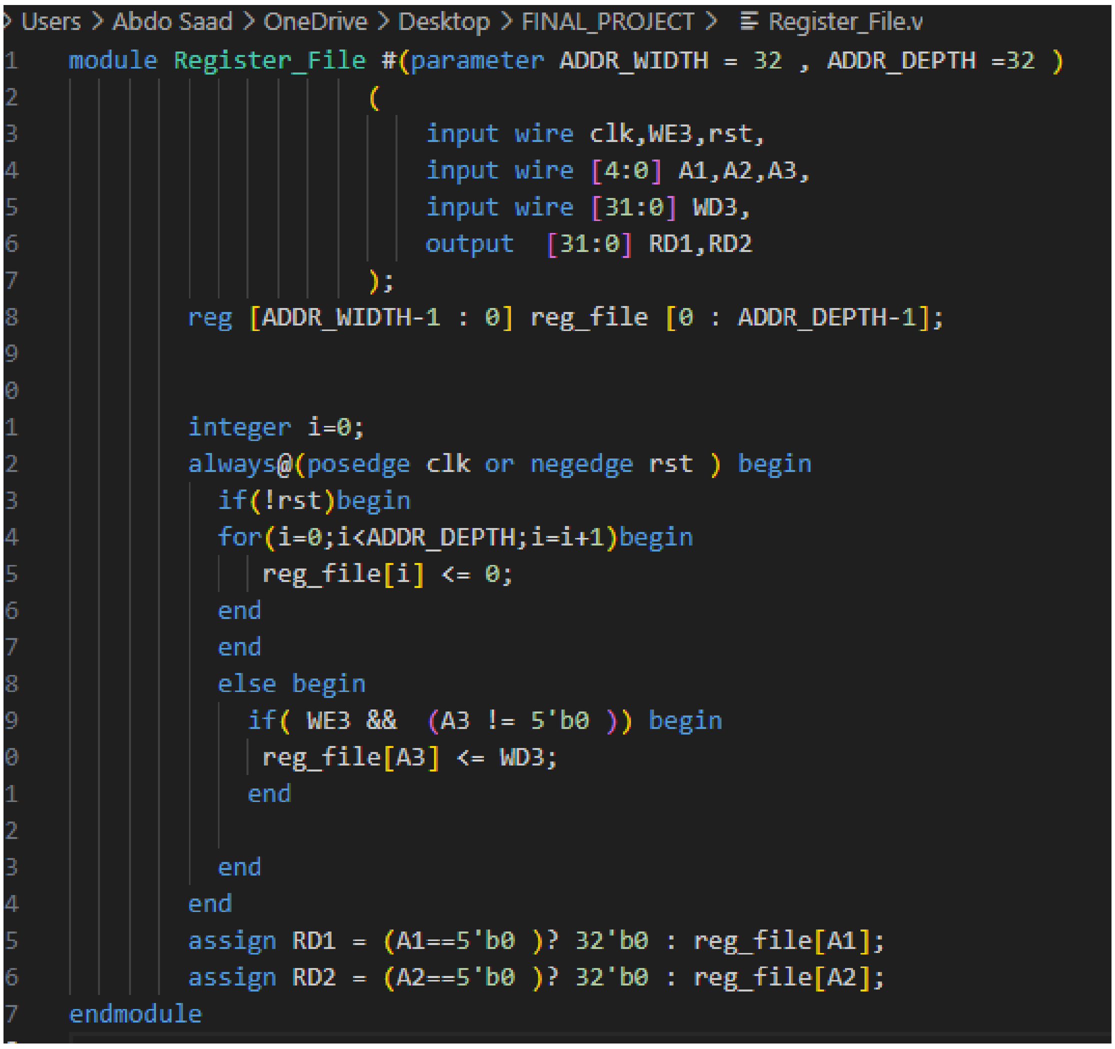
```
module Next_PC_logic(
                      input wire [31:0] PC,ImmExt,
                      input wire PCSrc,
                      output reg [31:0] PCNext
);

 always @(*) begin
  PCNext =(PCSrc == 0 ) ? (PC + 4) :(PC + ImmExt);
 end
endmodule
```

Program Counter



```
module (
                          input wire [31:0] NextPC,
                          input wire areset,clk,Load,
                          output reg [31:0]PC

);

always@(posedge clk or negedge areset ) begin

  if(~areset)
   PC <= 32'b0 ;
  else begin
    PC <= (Load == 0) ? PC : NextPC ;
  end

end

endmodule
```

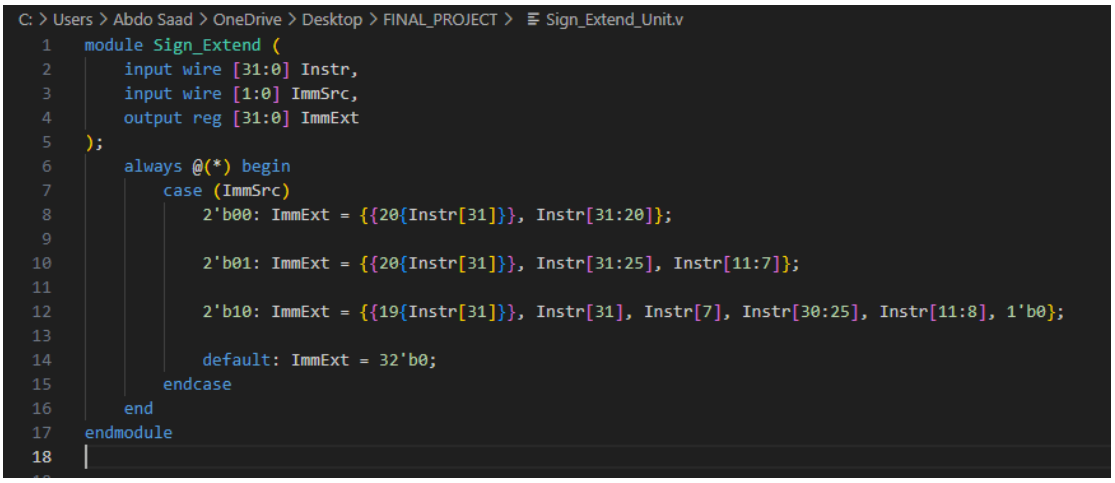
Register File



```
> Users > Abdo Saad > OneDrive > Desktop > FINAL_PROJECT > Register_File.v
```

```verilog
module Register_File #(parameter ADDR_WIDTH = 32 , ADDR_DEPTH =32 )
                    (
                        input wire clk,WE3,rst,
                        input wire [4:0] A1,A2,A3,
                        input wire [31:0] WD3,
                        output  [31:0] RD1,RD2
                    );
        reg [ADDR_WIDTH-1 : 0] reg_file [0 : ADDR_DEPTH-1];


        integer i=0;
        always@(posedge clk or negedge rst ) begin
          if(!rst)begin
          for(i=0;i<ADDR_DEPTH;i=i+1)begin
             reg_file[i] <= 0;
          end
          end
          else begin
            if( WE3 &&  (A3 != 5'b0 )) begin
             reg_file[A3] <= WD3;
            end

          end
        end
        assign RD1 = (A1==5'b0 )? 32'b0 : reg_file[A1];
        assign RD2 = (A2==5'b0 )? 32'b0 : reg_file[A2];
endmodule
```

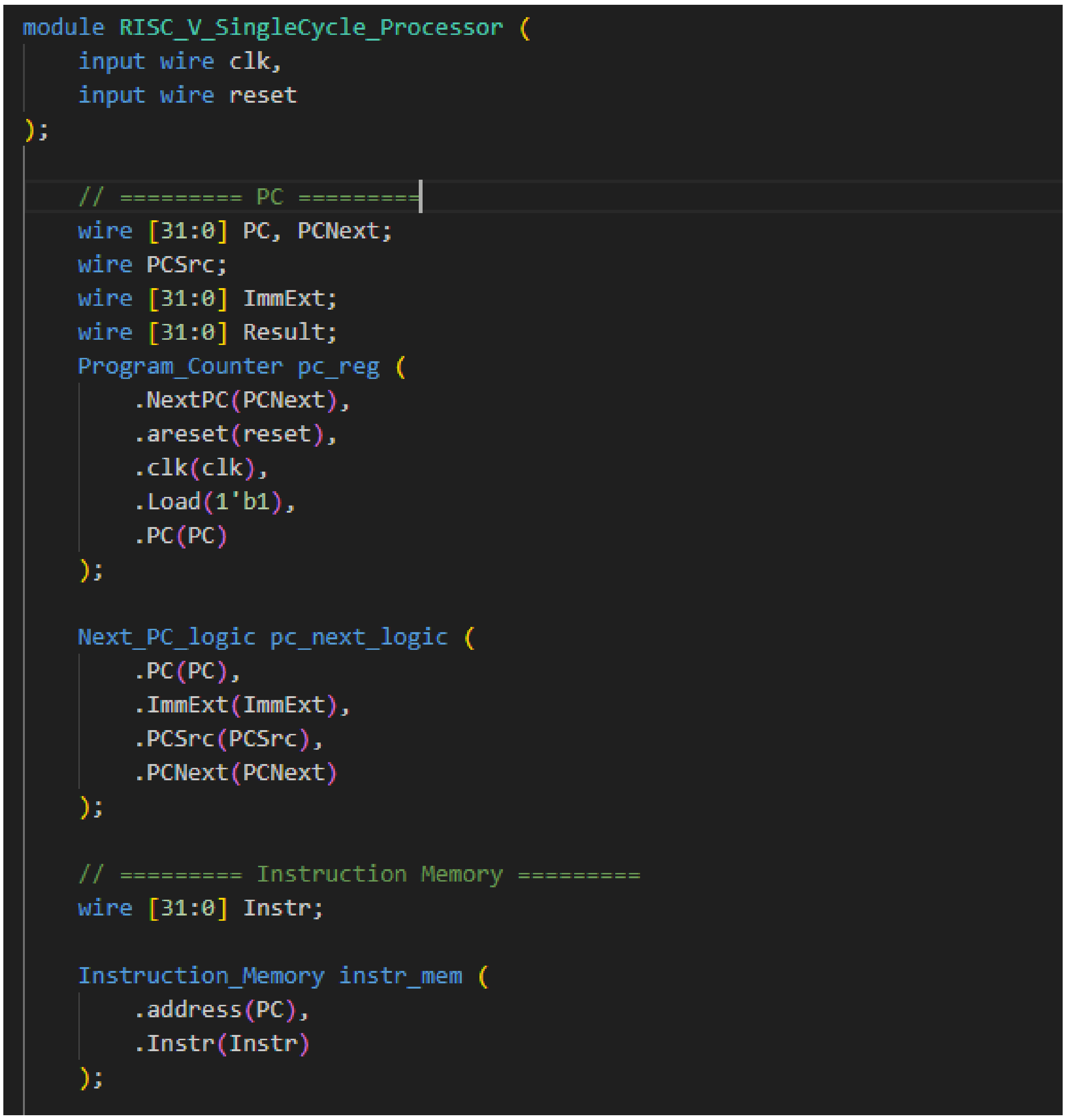
Sign Extend



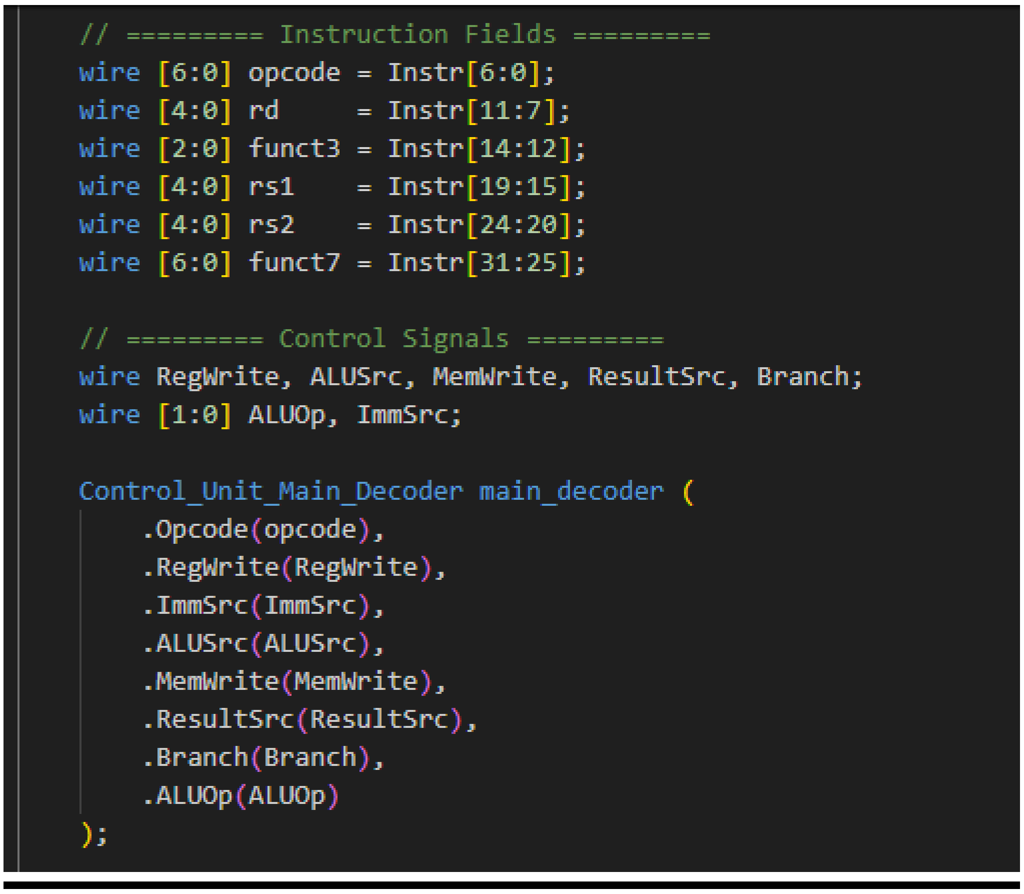
```
C: > Users > Abdo Saad > OneDrive > Desktop > FINAL_PROJECT > Sign_Extend_Unit.v
```

```verilog
module Sign_Extend (
    input wire [31:0] Instr,
    input wire [1:0] ImmSrc,
    output reg [31:0] ImmExt
);
    always @(*) begin
        case (ImmSrc)
            2'b00: ImmExt = {{20{Instr[31]}}, Instr[31:20]};

            2'b01: ImmExt = {{20{Instr[31]}}, Instr[31:25], Instr[11:7]};

            2'b10: ImmExt = {{19{Instr[31]}}, Instr[31], Instr[7], Instr[30:25], Instr[11:8], 1'b0};

            default: ImmExt = 32'b0;
        endcase
    end
endmodule

```

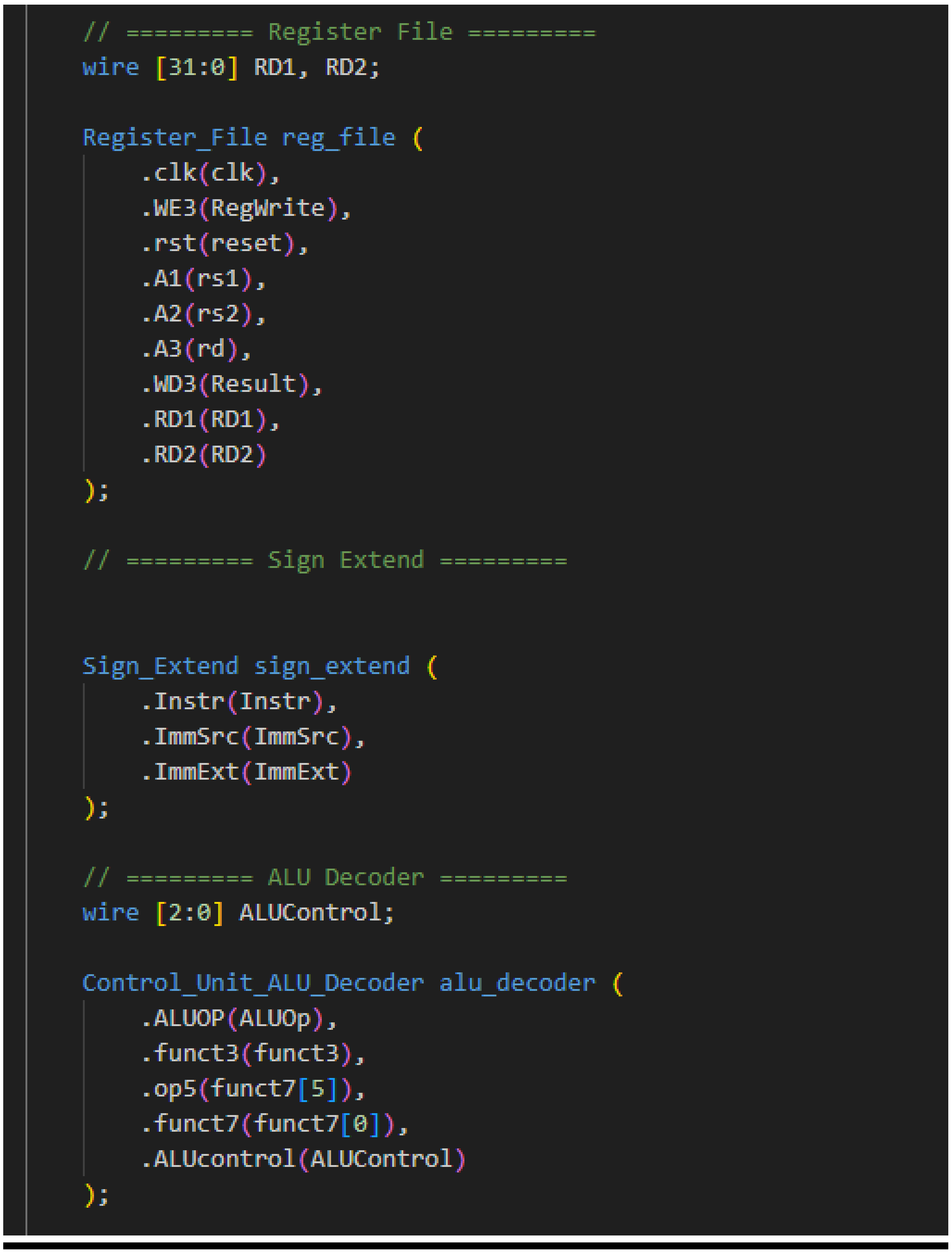
TOP_MODULE:



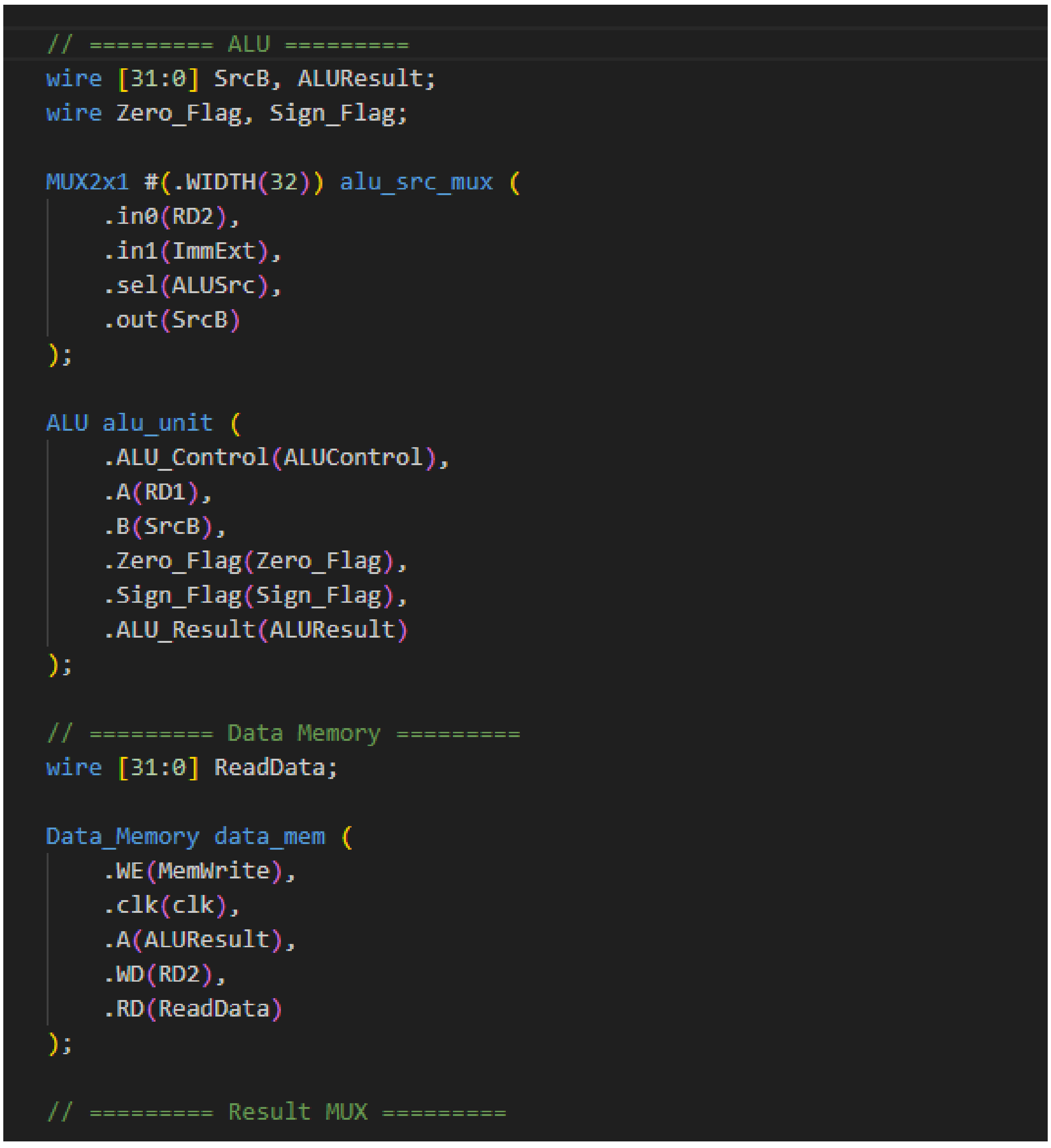
```
module RISC_V_SingleCycle_Processor (
    input wire clk,
    input wire reset
);

    // ========= PC =========
    wire [31:0] PC, PCNext;
    wire PCSrc;
    wire [31:0] ImmExt;
    wire [31:0] Result;
    Program_Counter pc_reg (
        .NextPC(PCNext),
        .areset(reset),
        .clk(clk),
        .Load(1'b1),
        .PC(PC)
    );

    Next_PC_logic pc_next_logic (
        .PC(PC),
        .ImmExt(ImmExt),
        .PCSrc(PCSrc),
        .PCNext(PCNext)
    );

    // ========= Instruction Memory =========
    wire [31:0] Instr;

    Instruction_Memory instr_mem (
        .address(PC),
        .Instr(Instr)
    );
```



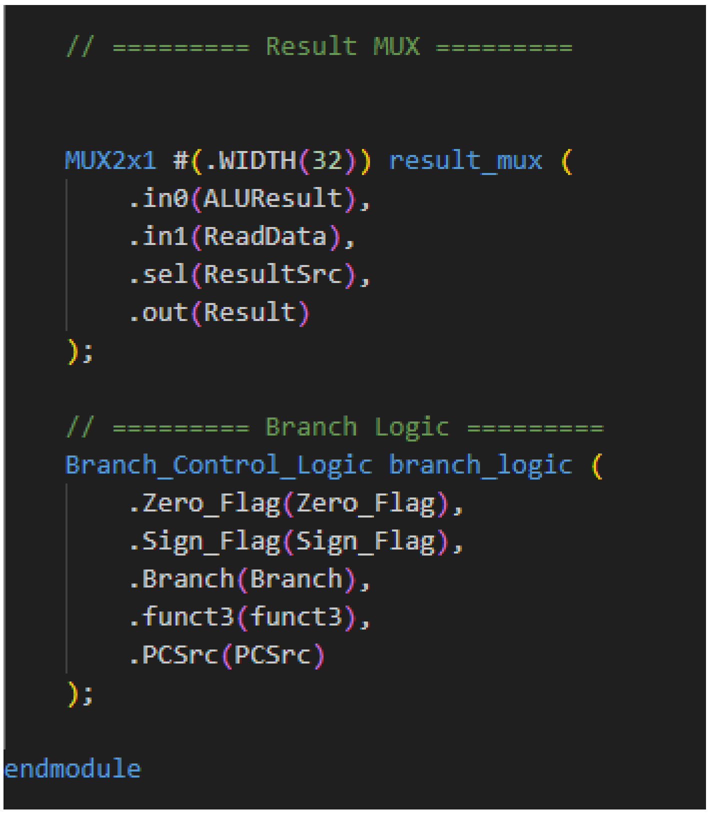
```verilog
// ========= Instruction Fields =========
wire [6:0] opcode = Instr[6:0];
wire [4:0] rd     = Instr[11:7];
wire [2:0] funct3 = Instr[14:12];
wire [4:0] rs1    = Instr[19:15];
wire [4:0] rs2    = Instr[24:20];
wire [6:0] funct7 = Instr[31:25];

// ========= Control Signals =========
wire RegWrite, ALUSrc, MemWrite, ResultSrc, Branch;
wire [1:0] ALUOp, ImmSrc;

Control_Unit_Main_Decoder main_decoder (
    .Opcode(opcode),
    .RegWrite(RegWrite),
    .ImmSrc(ImmSrc),
    .ALUSrc(ALUSrc),
    .MemWrite(MemWrite),
    .ResultSrc(ResultSrc),
    .Branch(Branch),
    .ALUOp(ALUOp)
);
```



```verilog
// ========= Register File =========
wire [31:0] RD1, RD2;

Register_File reg_file (
    .clk(clk),
    .WE3(RegWrite),
    .rst(reset),
    .A1(rs1),
    .A2(rs2),
    .A3(rd),
    .WD3(Result),
    .RD1(RD1),
    .RD2(RD2)
);

// ========= Sign Extend =========


Sign_Extend sign_extend (
    .Instr(Instr),
    .ImmSrc(ImmSrc),
    .ImmExt(ImmExt)
);

// ========= ALU Decoder =========
wire [2:0] ALUControl;

Control_Unit_ALU_Decoder alu_decoder (
    .ALUOP(ALUOp),
    .funct3(funct3),
    .op5(funct7[5]),
    .funct7(funct7[0]),
    .ALUcontrol(ALUControl)
);
```



```
// ========= ALU =========
wire [31:0] SrcB, ALUResult;
wire Zero_Flag, Sign_Flag;

MUX2x1 #(.WIDTH(32)) alu_src_mux (
    .in0(RD2),
    .in1(ImmExt),
    .sel(ALUSrc),
    .out(SrcB)
);

ALU alu_unit (
    .ALU_Control(ALUControl),
    .A(RD1),
    .B(SrcB),
    .Zero_Flag(Zero_Flag),
    .Sign_Flag(Sign_Flag),
    .ALU_Result(ALUResult)
);

// ========= Data Memory =========
wire [31:0] ReadData;

Data_Memory data_mem (
    .WE(MemWrite),
    .clk(clk),
    .A(ALUResult),
    .WD(RD2),
    .RD(ReadData)
);

// ========= Result MUX =========
```



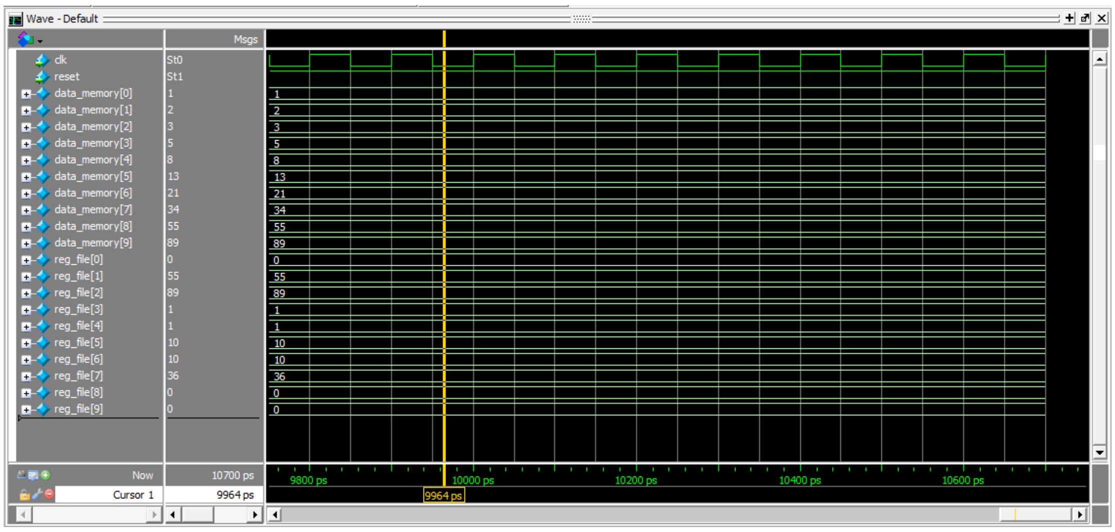
```verilog
    // ========= Result MUX =========


    MUX2x1 #(.WIDTH(32)) result_mux (
        .in0(ALUResult),
        .in1(ReadData),
        .sel(ResultSrc),
        .out(Result)
    );

    // ========= Branch Logic =========
    Branch_Control_Logic branch_logic (
        .Zero_Flag(Zero_Flag),
        .Sign_Flag(Sign_Flag),
        .Branch(Branch),
        .funct3(funct3),
        .PCSrc(PCSrc)
    );

endmodule
```

AFTER RUN THIS MACHINE CODE IN THE INSTRUCTION MEMORY

```
00004033
00000093
00100113
00100193
00100213
00000293
00a00313
00000393
00418c63
00110133
404181b3
00229393
0023a023
00420a63
002080b3
004181b3
00229393
0013a023
00128293
fc62cae3
00000000
```

This code is implemented in the Fibonacci sequence

The waveform



Wave - Default

	Msgs
clk	St0
reset	St1
data_memory[0]	1
data_memory[1]	2
data_memory[2]	3
data_memory[3]	5
data_memory[4]	8
data_memory[5]	13
data_memory[6]	21
data_memory[7]	34
data_memory[8]	55
data_memory[9]	89
reg_file[0]	0
reg_file[1]	55
reg_file[2]	89
reg_file[3]	1
reg_file[4]	1
reg_file[5]	10
reg_file[6]	10
reg_file[7]	36
reg_file[8]	0
reg_file[9]	0
Now	10700 ps
Cursor 1	9964 ps