## EME Digital Verification Intensive Camp

# Verilog Lab 3

## 32-bit Single Cycle RISC-V

9th Sep 2023

**Yaseen Salah Mohamed Abdelhameed**

New Capital Group

**Submitted to**

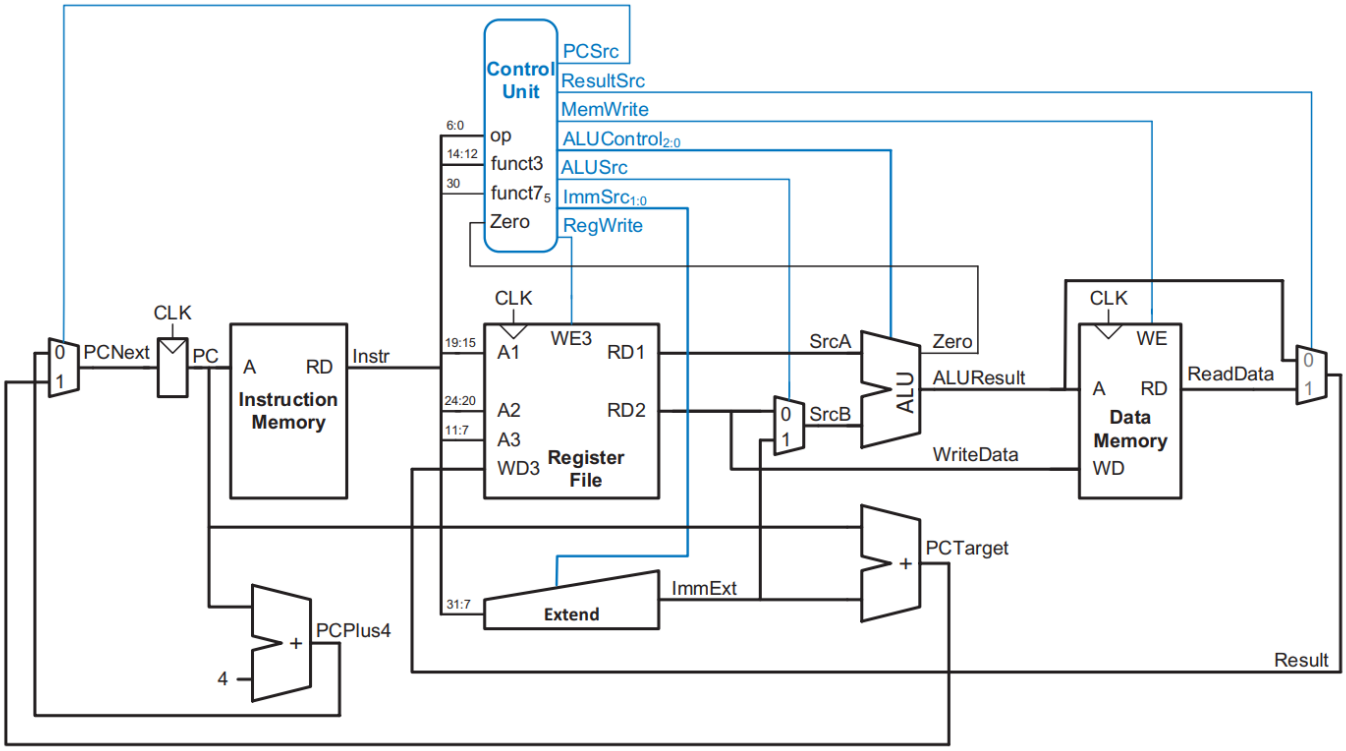
Dr. Dina Tantawy

Eng. Sara Sameh

# Introduction

RISC-V is an open-source instruction set architecture (ISA) known for its simplicity, modularity, and flexibility. It has gained popularity in the digital design community, particularly for the implementation of 32-bit single-cycle RISC-V processors. These processors follow a streamlined execution model, where each instruction is completed within a single clock cycle.

In the digital design of a 32-bit single-cycle RISC-V processor, key components include Control Unit that decodes the instruction received from the instruction memory, and the Datapath including its ALU (Arithmetic Logic Unit), Register File, Adders, and Multiplexers. Both Instruction and Data Memories are separated from the RISC-V core. The ALU performs arithmetic and logical operations, and the control unit manages instruction and data flow within the processor. The register file stores operands and results during execution.




*Complete single-cycle processor*

The Instruction Set Architecture (ISA) of RISC-V processor supports many instructions with different types and cases, but it is not essential to support all these instructions in our implementation as long as our application and needs don't require this.

# Design Requirements

In this project, we wish to design 32-bit single cycle RISC-V processor that supports these instructions:

**R-Type:** add, sub, and, or

**I-Type:** addi, andi, ori, lw, jalr

**B-Type:** beq, bne

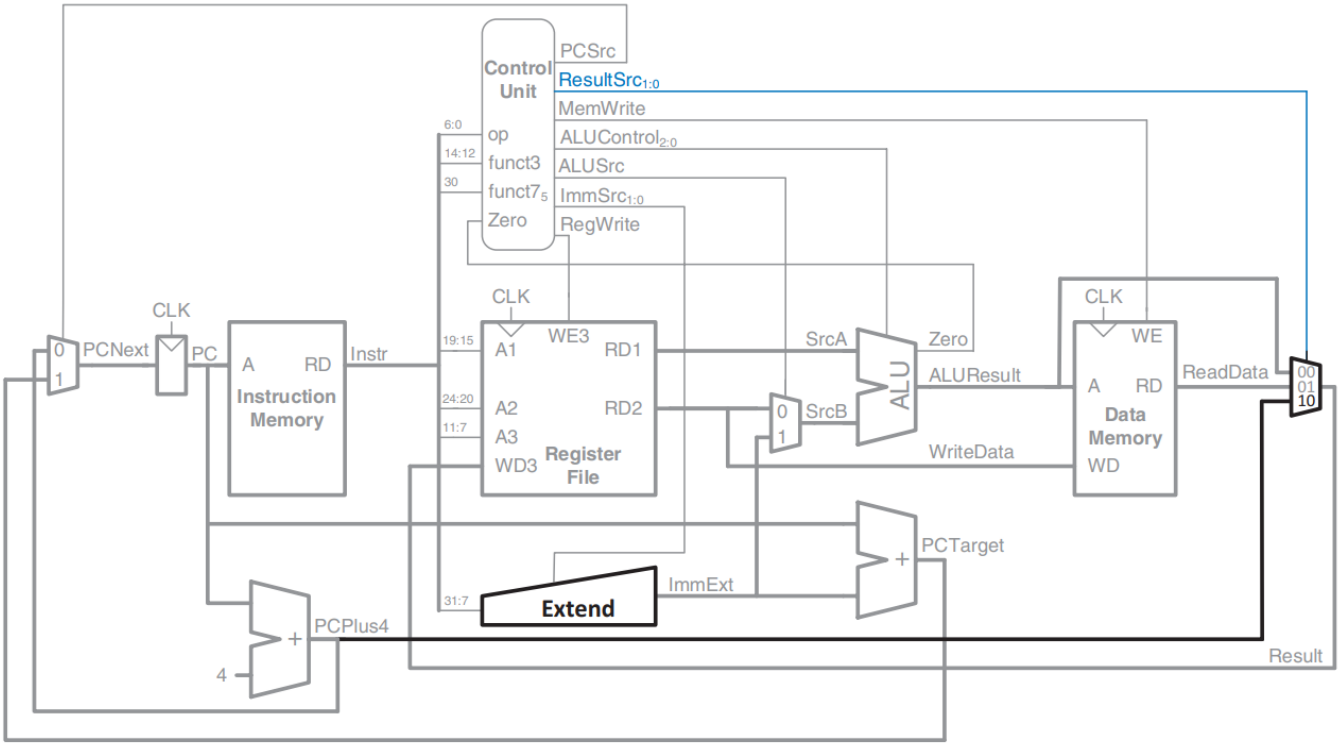
**J-Type:** jal

**S-Type:** sw

But we can notice that the architecture in the previous figure doesn't support jal, jalr, and bne. So, we have to modify the architecture to support these instructions.

# Architecture Modifications

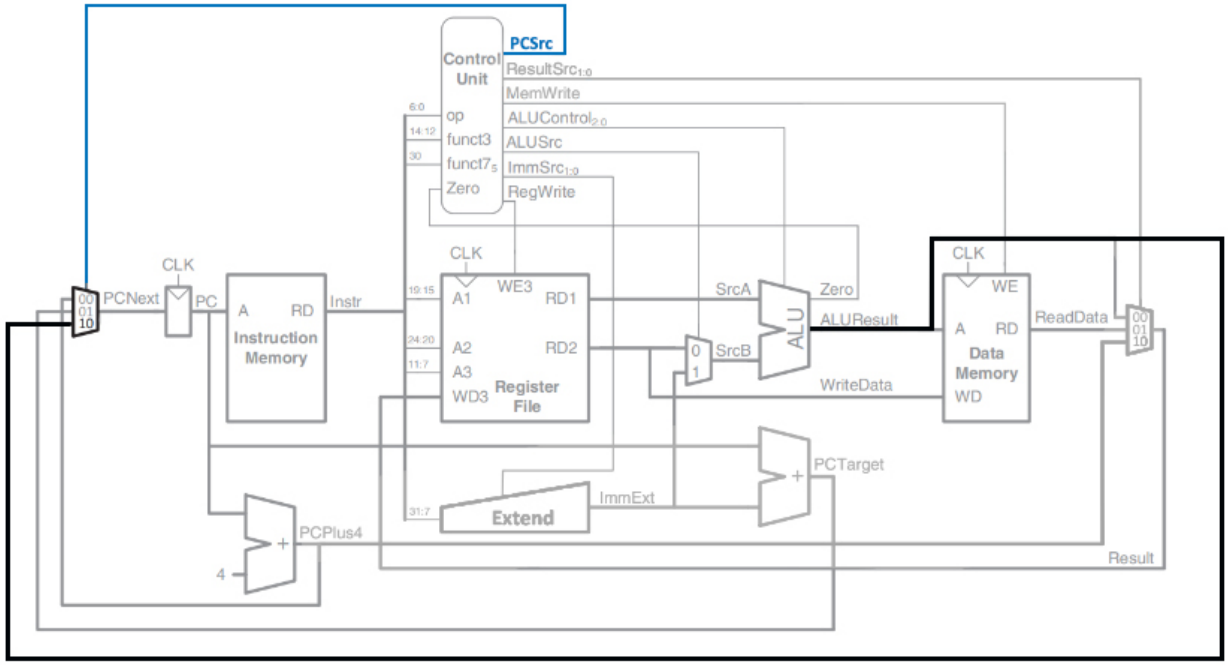
1. **Adding jal**
   Storing the value of the PC+4 in the register through the Result Mux.




*Enhanced datapath for jal*

2. **Adding jalr**

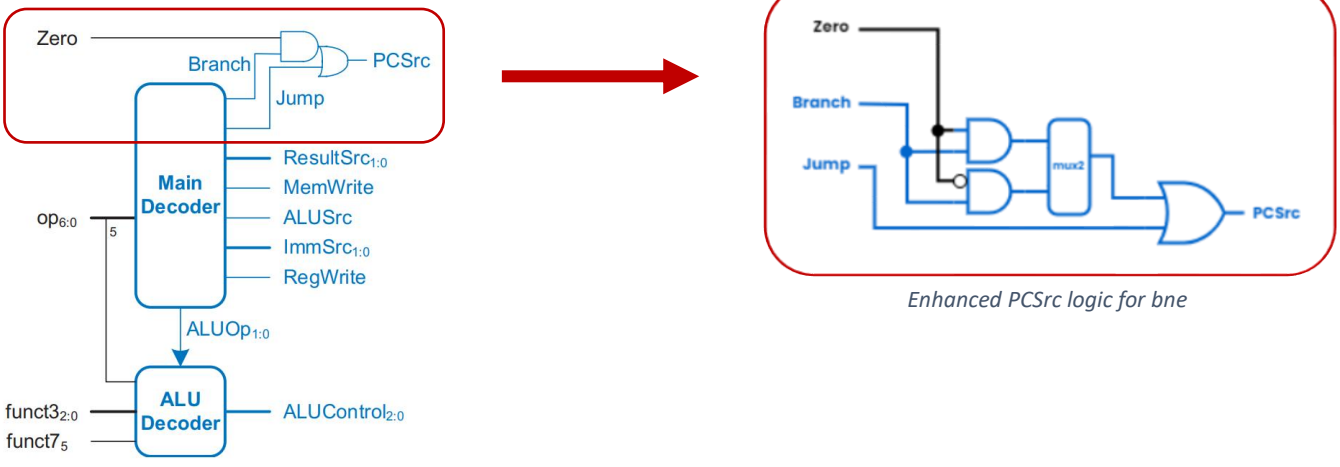
   Getting the PCNext as an ALUResult from adding the destination address stored in the register to an immediate offset through expanding the PC Mux.




*enhanced datapath for jalr*

3. **Adding bne**

   Modifying the Branch/Jump PCSrc logic inside the Control Unit by adding a conditional mux that checks for the lsb of funct3 to differ between beq and bne instructions. In the case of beq, Branch signal will be ANDed with Zero flag as before, while in the case of bne, Branch signal will be ANDed with the Inverse of Zero flag.

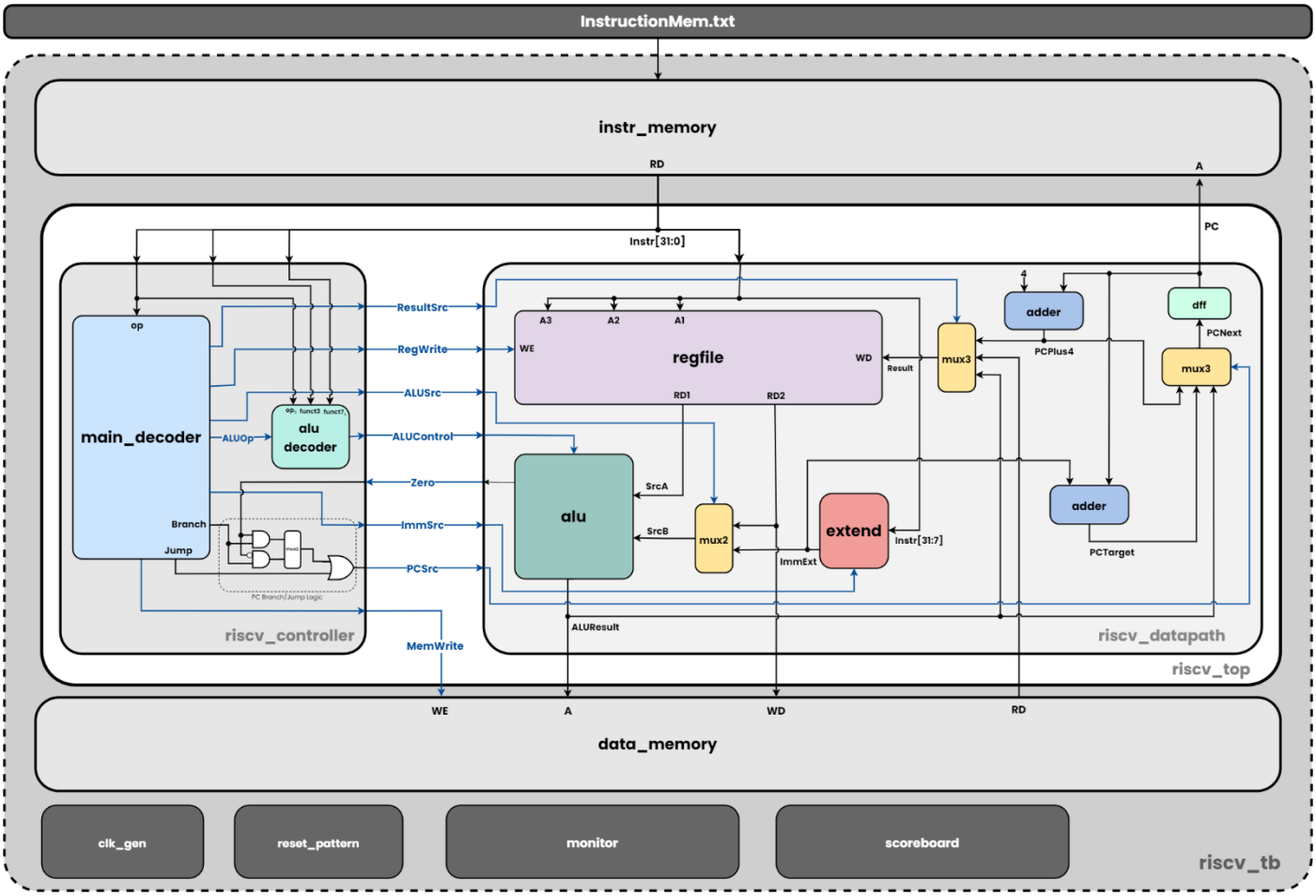



*Enhanced PCSrc logic for bne*

# Design Block Diagram

After the previous architecture enhancements and considering the external memories, we can define our system with the following hierarchy:

- The testbench environment contains the RISC-V core as DUT and connects it with the external instruction and data memories.
- The RISC-V core contains the RISC_V Controller and the RISC-V Datapath.
  - The RISC-V Controller contains the Main Decoder and the ALU Decoder with PCSrc logic.
  - The RISC-V Datapath contains the Register File, ALU, Sign-Extend Unit, Adders, and Muxs.




*RISC-V System Block Diagram*

# Design Verilog Codes

## 0. RISC-V Top

```
module riscv_top (
      input    wire                i_CLK        ,
      input    wire                i_Reset      ,
      input    wire     [31:0]     i_Instr      ,
      input    wire     [31:0]     i_ReadData   ,
      output   wire     [31:0]     o_PC         ,
      output   wire     [31:0]     o_ALUResult  ,
      output   wire                o_MemWrite   ,
      output   wire     [31:0]     o_WriteData
      );

wire         Zero       ;
wire  [1:0]  PCSrc      ;
wire  [1:0]  ResultSrc  ;
wire  [2:0]  ALUControl ;
wire         ALUSrc     ;
wire  [1:0]  ImmSrc     ;
wire         RegWrite   ;

riscv_controller controller (
      .i_op         (i_Instr[6:0])   ,
      .i_funct3     (i_Instr[14:12]) ,
      .i_funct7     (i_Instr[30])    ,
      .i_Zero       (Zero)           ,
      .o_PCSrc      (PCSrc)          ,
      .o_ResultSrc  (ResultSrc)      ,
      .o_MemWrite   (o_MemWrite)     ,
      .o_ALUControl (ALUControl)     ,
      .o_ALUSrc     (ALUSrc)         ,
      .o_ImmSrc     (ImmSrc)         ,
      .o_RegWrite   (RegWrite)
      );

riscv_datapath datapath (
      .i_CLK        (i_CLK)       ,
      .i_Reset      (i_Reset)     ,
      .i_Instr      (i_Instr)     ,
      .i_PCSrc      (PCSrc)       ,
      .i_ResultSrc  (ResultSrc)   ,
      .i_ALUControl (ALUControl)  ,
      .i_ALUSrc     (ALUSrc)      ,
      .i_ImmSrc     (ImmSrc)      ,
      .i_RegWrite   (RegWrite)    ,
      .i_ReadData   (i_ReadData)  ,
      .o_Zero       (Zero)        ,
      .o_PC         (o_PC)        ,
      .o_ALUResult  (o_ALUResult) ,
      .o_WriteData  (o_WriteData)
      );

endmodule
```

# 1. RISC-V Controller

```
module riscv_controller (
      input    wire     [6:0] i_op         ,
      input    wire     [2:0] i_funct3     ,
      input    wire           i_funct7     ,
      input    wire           i_Zero       ,
      output   wire     [1:0] o_PCSrc      ,
      output   wire     [1:0] o_ResultSrc  ,
      output   wire           o_MemWrite   ,
      output   wire     [2:0] o_ALUControl ,
      output   wire           o_ALUSrc     ,
      output   wire     [1:0] o_ImmSrc     ,
      output   wire           o_RegWrite
      );

wire              Branch  ;
wire              BeqBne  ;
wire              Jump    ;
wire              ImmJump ;
wire  [1:0]       ALUOp   ;

main_decoder cu1_main_decoder (
      .i_op        (i_op)        ,
      .o_Branch    (Branch)      ,
      .o_Jump      (Jump)        ,
      .o_ImmJump   (ImmJump)     ,
      .o_ResultSrc (o_ResultSrc) ,
      .o_MemWrite  (o_MemWrite)  ,
      .o_ALUSrc    (o_ALUSrc)    ,
      .o_ImmSrc    (o_ImmSrc)    ,
      .o_RegWrite  (o_RegWrite)  ,
      .o_ALUOp     (ALUOp)
      );

alu_decoder cu2_alu_decoder (
      .i_ALUOp      (ALUOp)        ,
      .i_op         (i_op[5])      ,
      .i_funct3     (i_funct3)     ,
      .i_funct7     (i_funct7)     ,
      .o_ALUControl (o_ALUControl)
      );

//bne instruction enhancement
assign BeqBne = (!i_funct3[0])?
      ( i_Zero & Branch) : //funct3(beq)=000
      (~i_Zero & Branch) ; //funct3(bne)=001

assign o_PCSrc = {ImmJump , (BeqBne | Jump)} ;

endmodule
```

## 1.1. Main Decoder

```
module main_decoder (
      input    wire     [6:0] i_op        ,
      output   reg            o_Branch    ,
      output   reg            o_Jump      ,
      output   reg            o_ImmJump   , //jalr instruction enhancement
      output   reg      [1:0] o_ResultSrc , //Values are defined in Figure 7.17
      output   reg            o_MemWrite  ,
      output   reg            o_ALUSrc    ,
      output   reg      [1:0] o_ImmSrc    , //Values are defined in Table 7.5
      output   reg            o_RegWrite  ,
      output   reg      [1:0] o_ALUOp       //Values are defined in Table 7.3
      );

always @(*)
      begin
            case (i_op)
            //-- R-instructions
            7'b0110011: //add,sub,and,or
                  begin
                        o_Branch    = 1'b0  ;
                        o_Jump      = 1'b0  ;
                        o_ImmJump   = 1'b0  ;
                        o_ResultSrc = 2'b00 ;
                        o_MemWrite  = 1'b0  ;
                        o_ALUSrc    = 1'b0  ;
                        o_ImmSrc    = 2'b00 ;
                        o_RegWrite  = 1'b1  ;
                        o_ALUOp     = 2'b10 ;
                  end


            //-- I-instructions
            7'b0010011: //addi,andi,ori
                  begin
                        o_Branch    = 1'b0  ;
                        o_Jump      = 1'b0  ;
                        o_ImmJump   = 1'b0  ;
                        o_ResultSrc = 2'b00 ;
                        o_MemWrite  = 1'b0  ;
                        o_ALUSrc    = 1'b1  ;
                        o_ImmSrc    = 2'b00 ;
                        o_RegWrite  = 1'b1  ;
                        o_ALUOp     = 2'b10 ;
                  end
            7'b0000011: //lw
                  begin
                        o_Branch    = 1'b0  ;
                        o_Jump      = 1'b0  ;
                        o_ImmJump   = 1'b0  ;
                        o_ResultSrc = 2'b01 ;
                        o_MemWrite  = 1'b0  ;
                        o_ALUSrc    = 1'b1  ;
                        o_ImmSrc    = 2'b00 ;
                        o_RegWrite  = 1'b1  ;
                        o_ALUOp     = 2'b00 ;
                  end
```

```
        7'b1100111: //jalr
              begin
                    o_Branch    = 1'b0  ;
                    o_Jump      = 1'b0  ;
                    o_ImmJump   = 1'b1  ;
                    o_ResultSrc = 2'b10 ;
                    o_MemWrite  = 1'b0  ;
                    o_ALUSrc    = 1'b1  ;
                    o_ImmSrc    = 2'b00 ;
                    o_RegWrite  = 1'b1  ;
                    o_ALUOp     = 2'b10 ;
              end


        //-- B-instructions
        7'b1100011: //beq,bne
              begin
                    o_Branch    = 1'b1  ;
                    o_Jump      = 1'b0  ;
                    o_ImmJump   = 1'b0  ;
                    o_ResultSrc = 2'b00 ;
                    o_MemWrite  = 1'b0  ;
                    o_ALUSrc    = 1'b0  ;
                    o_ImmSrc    = 2'b10 ;
                    o_RegWrite  = 1'b0  ;
                    o_ALUOp     = 2'b01 ;
              end


        //-- J-instructions
        7'b1101111: //jal
              begin
                    o_Branch    = 1'b0  ;
                    o_Jump      = 1'b1  ;
                    o_ImmJump   = 1'b0  ;
                    o_ResultSrc = 2'b10 ;
                    o_MemWrite  = 1'b0  ;
                    o_ALUSrc    = 1'b0  ;
                    o_ImmSrc    = 2'b11 ;
                    o_RegWrite  = 1'b1  ;
                    o_ALUOp     = 2'b00 ;
              end


        //-- S-instructions
        7'b0100011: //sw
              begin
                    o_Branch    = 1'b0  ;
                    o_Jump      = 1'b0  ;
                    o_ImmJump   = 1'b0  ;
                    o_ResultSrc = 2'b00 ;
                    o_MemWrite  = 1'b1  ;
                    o_ALUSrc    = 1'b1  ;
                    o_ImmSrc    = 2'b01 ;
                    o_RegWrite  = 1'b0  ;
                    o_ALUOp     = 2'b00 ;
              end
```

```
            //-- Default Case
            default:
                  begin
                        o_Branch    = 1'b0  ;
                        o_Jump      = 1'b0  ;
                        o_ImmJump   = 1'b0  ;
                        o_ResultSrc = 2'b00 ;
                        o_MemWrite  = 1'b0  ;
                        o_ALUSrc    = 1'b0  ;
                        o_ImmSrc    = 2'b00 ;
                        o_RegWrite  = 1'b0  ;
                        o_ALUOp     = 2'b00 ;
                  end
            endcase
      end


endmodule
```

## 1.2. ALU Decoder

```
module alu_decoder (
      input    wire     [1:0] i_ALUOp      ,
      input    wire           i_op         ,
      input    wire     [2:0] i_funct3     ,
      input    wire           i_funct7     ,
      output   reg      [2:0] o_ALUControl
      );
always @(*)
      begin
            case (i_ALUOp)
            2'b00:
                  begin
                        o_ALUControl = 3'b000 ; //lw,sw
                  end
            2'b01:
                  begin
                        o_ALUControl = 3'b001 ; //beq
                  end
            2'b10:
                  begin
                        case (i_funct3)
                        3'b000:
                              begin
                                    if ({i_op , i_funct7} == 2'b11)
                                          begin
                                                o_ALUControl = 3'b001 ; //sub
                                          end
                                    else
                                          begin
                                                o_ALUControl = 3'b000 ; //add
                                          end
                              end
                        3'b010:
                              begin
                                    o_ALUControl = 3'b101 ; //slt
                              end
                        3'b110:
                              begin
                                    o_ALUControl = 3'b011 ; //or
                              end
                        3'b111:
                              begin
                                    o_ALUControl = 3'b010 ; //and
                              end
                        default:
                              begin
                                    o_ALUControl = 3'b000 ;
                              end
                        endcase
                  end
            default:
                  begin
                        o_ALUControl = 3'b000 ;
                  end
            endcase
      end
endmodule
```

## 2. RISC-V Datapath

```
module riscv_datapath (
      input    wire               i_CLK        ,
      input    wire               i_Reset      ,
      input    wire    [31:0]     i_Instr      ,
      input    wire    [1:0]      i_PCSrc      ,
      input    wire    [1:0]      i_ResultSrc  ,
      input    wire    [2:0]      i_ALUControl ,
      input    wire               i_ALUSrc     ,
      input    wire    [1:0]      i_ImmSrc     ,
      input    wire               i_RegWrite   ,
      input    wire    [31:0]     i_ReadData   ,
      output   wire               o_Zero       ,
      output   wire    [31:0]     o_PC         ,
      output   wire    [31:0]     o_ALUResult  ,
      output   wire    [31:0]     o_WriteData
      );

wire  [31:0] RD1       ;
wire  [31:0] RD2       ;
wire  [31:0] ImmExt    ;
wire  [31:0] SrcB      ;
wire  [31:0] ALUResult ;
wire  [31:0] Result    ;
wire  [31:0] PCPlus4   ;
wire  [31:0] PCTarget  ;
wire  [31:0] PCNext    ;
wire  [31:0] PC        ;



regfile du1_regfile (
      .i_CLK       (i_CLK)          ,
      .i_Reset     (i_Reset)        ,
      .i_A1        (i_Instr[19:15]) ,
      .i_A2        (i_Instr[24:20]) ,
      .i_A3        (i_Instr[11:7])  ,
      .i_WE3       (i_RegWrite)     ,
      .i_WD3       (Result)         , //from Result Mux
      .o_RD1       (RD1)            ,
      .o_RD2       (RD2)
      );

extend du2_extend (
      .i_Imm    (i_Instr[31:7]) ,
      .i_ImmSrc (i_ImmSrc)      ,
      .o_ImmExt (ImmExt)
      );

mux2 du3_mux (
      .i_DataInA (RD2)      ,
      .i_DataInB (ImmExt)   ,
      .i_Select  (i_ALUSrc) ,
      .o_DataOut (SrcB)
      );
```

```
alu du4_alu (
      .i_SrcA       (RD1)          ,
      .i_SrcB       (SrcB)         ,
      .i_ALUControl (i_ALUControl) ,
      .o_Zero       (o_Zero)       ,
      .o_ALUResult  (ALUResult)
      );

mux3 du5_mux (
      .i_DataInA (ALUResult)   ,
      .i_DataInB (i_ReadData)  ,
      .i_DataInC (PCPlus4)     ,
      .i_Select  (i_ResultSrc) ,
      .o_DataOut (Result)
      );

adder du6_adder (
      .i_SrcA        (PC)        ,
      .i_SrcB        (32'd4)     ,
      .o_AdderResult (PCPlus4)
      );

adder du7_adder (
      .i_SrcA        (PC)        ,
      .i_SrcB        (ImmExt)    ,
      .o_AdderResult (PCTarget)
      );

mux3 du8_mux (
      .i_DataInA (PCPlus4)   ,
      .i_DataInB (PCTarget)  ,
      .i_DataInC (ALUResult) ,
      .i_Select  (i_PCSrc)   ,
      .o_DataOut (PCNext)
      );

dff du9_dff (
      .i_D     (PCNext)  ,
      .i_CLK   (i_CLK)   ,
      .i_Reset (i_Reset) ,
      .o_Q     (PC)
      );

assign o_ALUResult = ALUResult ;
assign o_PC        = PC        ;
assign o_WriteData = RD2         ;

endmodule
```

## 2.1. Register File

```
module regfile (
      input    wire                i_CLK   ,
      input    wire                i_Reset ,
      input    wire     [4:0]      i_A1    ,
      input    wire     [4:0]      i_A2    ,
      input    wire     [4:0]      i_A3    ,
      input    wire                i_WE3   ,
      input    wire     [31:0]     i_WD3   ,
      output   wire     [31:0]     o_RD1   ,
      output   wire     [31:0]     o_RD2
      );

reg [31:0] register [31:0] ;
integer i ;

always @(posedge i_CLK or posedge i_Reset)
      begin
            if (i_Reset)
                  begin
                        for (i=0 ; i<32 ; i=i+1)
                              begin
                                    register[i] <= 32'b0 ;
                              end
                  end
            else if (i_WE3)
                  begin
                        register[i_A3] <= i_WD3 ;
                  end
      end

assign  o_RD1 = (i_A1 != 5'b0)? register[i_A1] : 32'b0 ;
assign  o_RD2 = (i_A2 != 5'b0)? register[i_A2] : 32'b0 ;

endmodule
```

## 2.2. Sign-Extend Unit

```
module extend (
      input    wire     [31:7]      i_Imm    ,
      input    wire     [1:0] i_ImmSrc ,
      output   reg      [31:0]      o_ImmExt
      );

always @(*)
      begin
            case (i_ImmSrc)
            2'b00: //-- I-instructions
                  begin
                        o_ImmExt = { {20{i_Imm[31]}} , i_Imm[31:20] } ;
                  end
            2'b01: //-- S-instructions
                  begin
                        o_ImmExt = { {20{i_Imm[31]}} , i_Imm[31:25] , i_Imm[11:7] } ;
                  end
            2'b10: //-- B-instructions
                  begin
                        o_ImmExt = { {20{i_Imm[31]}} , i_Imm[7] , i_Imm[30:25] ,
i_Imm[11:8] , 1'b0 } ;
                  end
            2'b11: //-- J-instructions
                  begin
                        o_ImmExt = { {12{i_Imm[31]}} , i_Imm[19:12] , i_Imm[20] ,
i_Imm[30:21] , 1'b0 } ;
                  end
            default:
                  begin
                        o_ImmExt = 32'b0 ;
                  end
            endcase
      end


endmodule
```

## 2.3. D- Filpflop (32-bit Register)

```
module dff (
      input    wire  [31:0]   i_D     ,
      input    wire            i_CLK   ,
      input    wire            i_Reset ,
      output   reg   [31:0]   o_Q
      );

always @(posedge i_CLK or posedge i_Reset)
      begin
            if (i_Reset)
                  begin
                        o_Q <= 32'b0 ;
                  end
            else
                  begin
                        o_Q <= i_D ;
                  end
      end
endmodule
```

## 2.4. Multiplexer(2X1)

```
module mux2 (
    input    wire  [31:0]  i_DataInA  ,
    input    wire  [31:0]  i_DataInB  ,
    input    wire           i_Select   ,
    output   reg   [31:0]  o_DataOut
    );

always @(*)
    begin
        case (i_Select)
        1'b0:
            begin
                o_DataOut = i_DataInA ;
            end
        1'b1:
            begin
                o_DataOut = i_DataInB ;
            end
        default:
            begin
                o_DataOut = 32'b0 ;
            end
        endcase
    end
endmodule
```

## 2.5. Multiplexer(3X1)

```
module mux3 (
    input    wire  [31:0]  i_DataInA  ,
    input    wire  [31:0]  i_DataInB  ,
    input    wire  [31:0]  i_DataInC  ,
    input    wire  [1:0]   i_Select   ,
    output   reg   [31:0]  o_DataOut
    );

always @(*)
    begin
        case (i_Select)
        2'b00:
            begin
                o_DataOut = i_DataInA ;
            end
        2'b01:
            begin
                o_DataOut = i_DataInB ;
            end
        2'b10:
            begin
                o_DataOut = i_DataInC ;
            end
        default:
            begin
                o_DataOut = 32'b0 ;
            end
        endcase
    end
endmodule
```

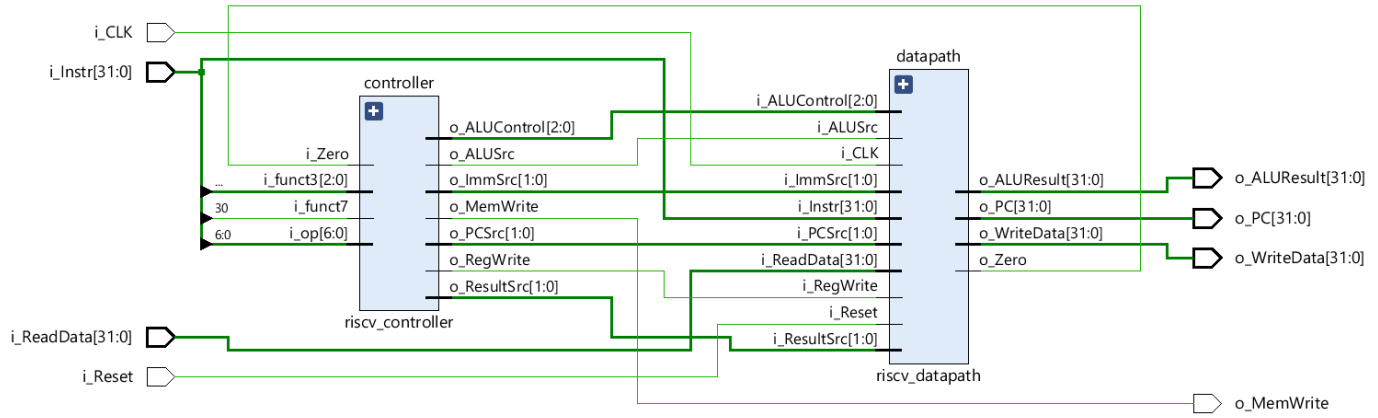
## 2.6. ALU

```
module alu (
      input    wire     signed  [31:0]    i_SrcA       ,
      input    wire     signed  [31:0]    i_SrcB       ,
      input    wire             [2:0]     i_ALUControl ,
      output   wire                       o_Zero       ,
      output   wire     signed  [31:0]    o_ALUResult
      );

reg [31:0] ALUResult ;

always @(*)
      begin
            case (i_ALUControl)
            3'b000: //--ADD
                  begin
                        ALUResult = i_SrcA + i_SrcB ;
                  end
            3'b001: //--SUB
                  begin
                        ALUResult = i_SrcA - i_SrcB ;
                  end
            3'b101: //--SLT
                  begin
                        //for hardware optimization, instead of adding comparator logic
                        //result is 1 if A-B is negative (A less than B) and 0 if A-B is positive
                        //check Figure 5.18
                        ALUResult = i_SrcA - i_SrcB ;
                        ALUResult = { {31{1'b0}} , ALUResult[31] } ;
                  end
            3'b011: //--OR
                  begin
                        ALUResult =  i_SrcA | i_SrcB ;
                  end
            3'b010: //--AND
                  begin
                        ALUResult = i_SrcA & i_SrcB ;
                  end
            default:
                  begin
                        ALUResult = 32'b0 ;
                  end
            endcase
      end

assign o_ALUResult = ALUResult ;

assign o_Zero = (ALUResult == 32'b0)? 1 : 0 ;

endmodule
```

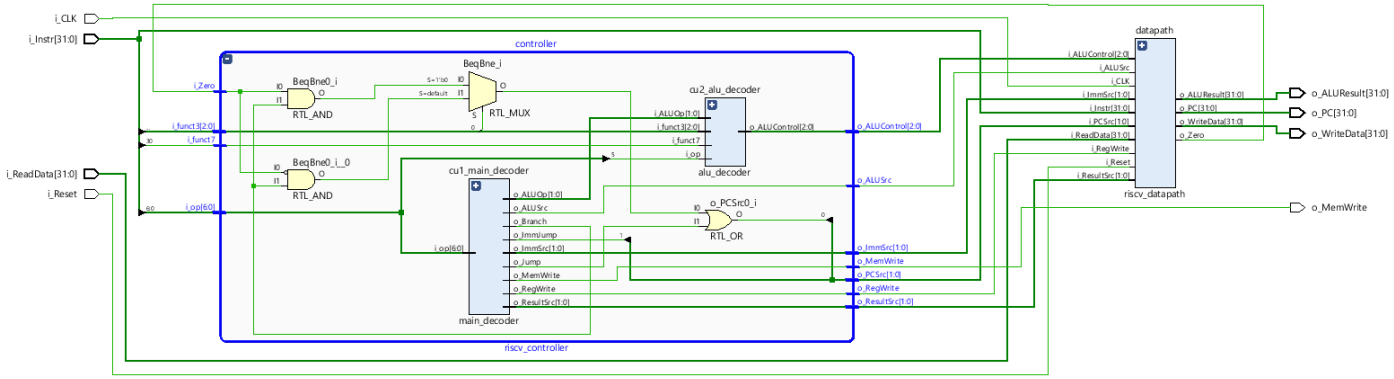
## 2.7. Adder

```
module adder (
      input    wire     signed  [31:0]    i_SrcA       ,
      input    wire     signed  [31:0]    i_SrcB       ,
      output   wire     signed  [31:0]    o_AdderResult );
assign o_AdderResult = i_SrcA + i_SrcB ;
endmodule
```

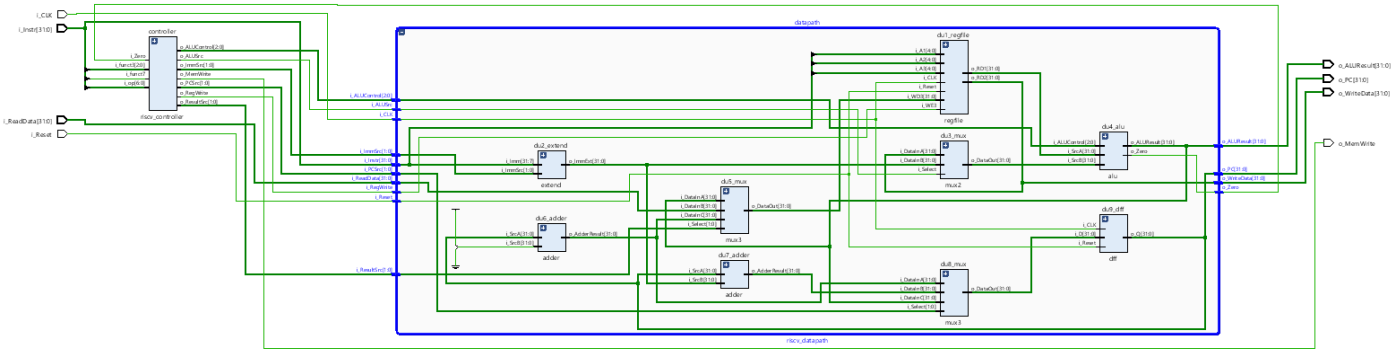
# Design Elaboration and Synthesis




*RISC-V Top view shows the two main parts, Controller and Datapath*



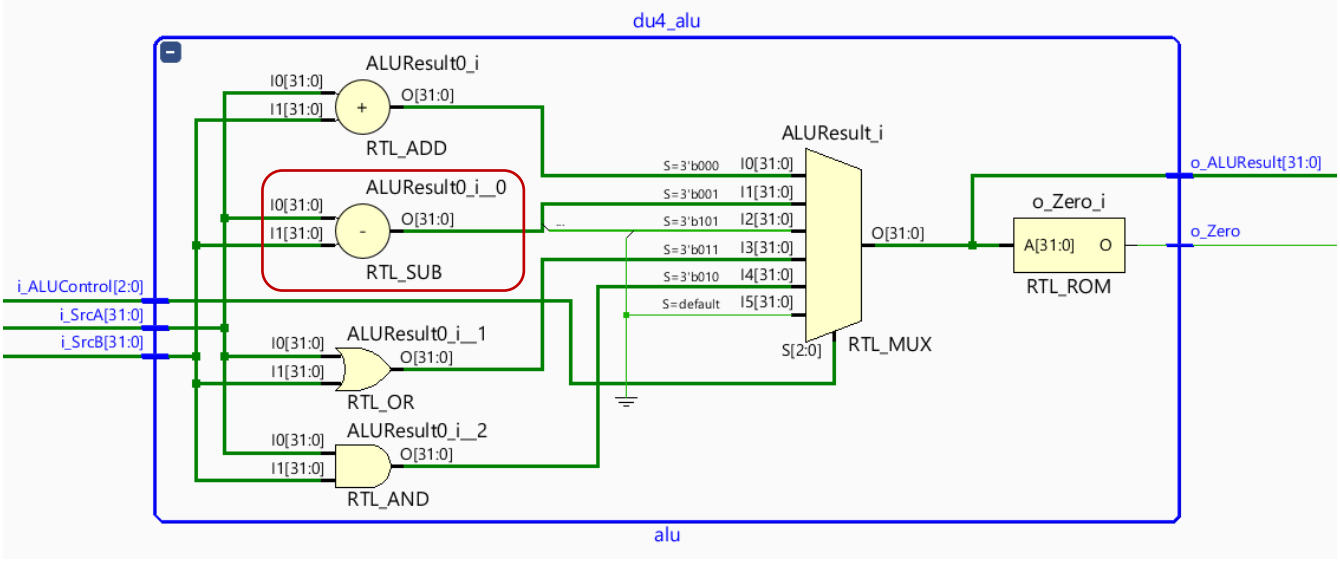

*Controller Top view shows its two modules, Main Decoder and ALU Decoder beside the PCSrc logic.*



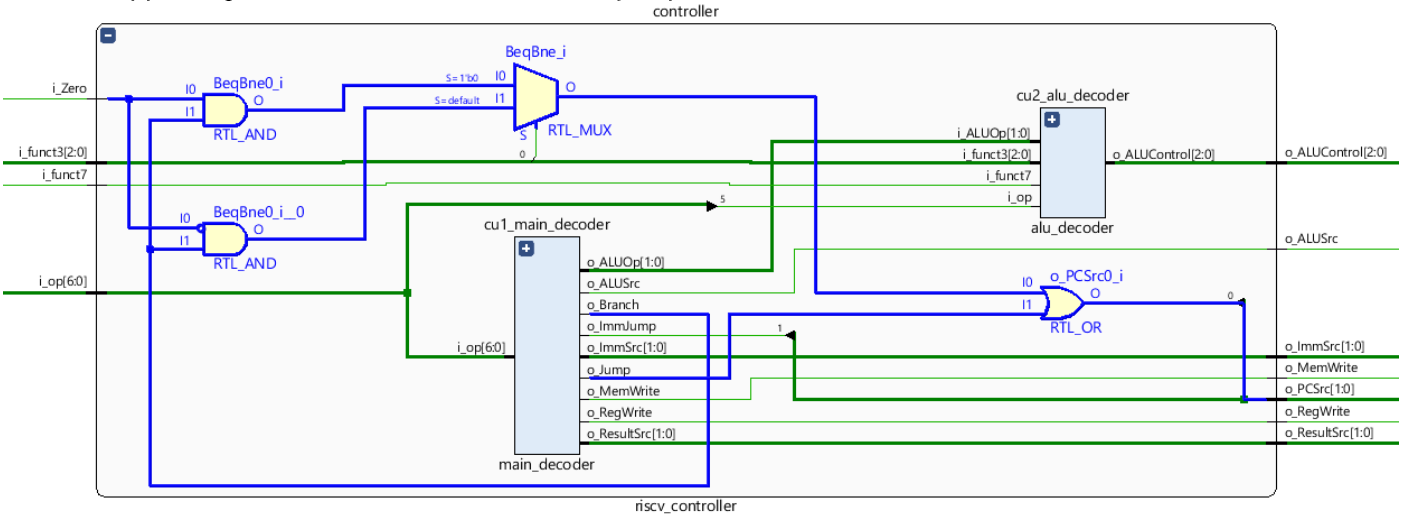
*Datapath Top view shows its two modules, Register File, ALU, Sign-Extend Unit, Adders, and Multiplexers.*

# Implementation Comments

- As we discussed in the ALU Verilog code, reusing the Subtractor unit in Set Less Than (SLT) operation is significantly optimize the resulted hardware.




- We can check that our intended Branch/Jump PCSrc logic in the final **RISC-V Block Diagram** after supporting *bne* instruction is successfully implemented in the Controller Unit.



# Testbench Verilog Codes

```
`timescale 1ns / 1ps
module riscv_tb ();
reg                CLK_tb       ;
reg                Reset_tb     ;
wire [31:0]        Instr_tb     ;
wire [31:0]        ReadData_tb  ;
wire [31:0]        PC_tb        ;
wire [31:0]        ALUResult_tb ;
wire               MemWrite_tb  ;
wire [31:0]        WriteData_tb ;

parameter CLK_PERIOD = 1 ; // 1 GHz Clk

riscv_top RISCV_DUT (
      .i_CLK       (CLK_tb)       ,
      .i_Reset     (Reset_tb)     ,
      .i_Instr     (Instr_tb)     ,
      .i_ReadData  (ReadData_tb)  ,
      .o_PC        (PC_tb)        ,
      .o_ALUResult (ALUResult_tb) ,
      .o_MemWrite  (MemWrite_tb)  ,
      .o_WriteData (WriteData_tb)
      );
instr_memory i_mem (
      .i_A  (PC_tb[9:2]) ,
      .o_RD (Instr_tb)
      );
data_memory d_mem (
      .i_CLK   (CLK_tb)           ,
      .i_Reset (Reset_tb)         ,
      .i_A     (ALUResult_tb[9:2]) ,
      .i_WE    (MemWrite_tb)      ,
      .i_WD    (WriteData_tb)     ,
      .o_RD    (ReadData_tb)
      );

`include "clk_gen.v"
`include "reset_pattern.v"
`include "monitor.v"
`include "scoreboard.v"

initial
      begin
        reset_pattern();
        wait (PC_tb == 32'h48)
            begin
                  #1
                  $display("=================Data Memory Results=================") ;
                  $display("%0d is stored successfully in data memory[96]" ,
                                    RISCV_DUT.datapath.du1_regfile.register[7]) ;
                  $display("%0d is stored successfully in data memory[92]" ,
                                    RISCV_DUT.datapath.du1_regfile.register[2]) ;
                  $stop ;
            end
      end

endmodule
```

# NOTE

We made a modular (layered) testbench that include different procedurals and tasks in different files.

```
`include "clk_gen.v"
`include "reset_pattern.v"
`include "monitor.v"
`include "scoreboard.v"
```

clk_gen.v

```
//-----Clock Generation
initial
    begin
        CLK_tb   = 1'b1 ;
        forever
        #(CLK_PERIOD*0.5) CLK_tb = ~CLK_tb ;
    end
```

reset_pattern.v

```
/*--------------- Reset Pattern ---------------*/
task reset_pattern ();
    begin
        Reset_tb = 1'b0 ;
        #1
        Reset_tb = 1'b1 ;
        #1
        Reset_tb = 1'b0 ;
    end
endtask
```

monitor.v

```
reg [31:0] rd_reg   ; // Destination Register

always @(negedge CLK_tb)
    begin
        rd_reg = Instr_tb[11:7] ;
        case (Instr_tb[6:0])
        //-- R-instructions
        7'b0110011: //add,sub,and,or
            begin
                case (Instr_tb[14:12])
                3'b000:
                    begin
                        if ({Instr_tb[5] , Instr_tb[30]} == 2'b11)
                            begin
                                $display("PC=%h %s Instruction | expected x%0d=%0d, actual is %0d" ,
                                        PC_tb , "SUB " , rd_reg , rd_value , RISCV_DUT.datapath.Result) ;
                            end
                        else
                            begin
                                $display("PC=%h %s Instruction | expected x%0d=%0d, actual is %0d" ,
                                        PC_tb , "ADD " , rd_reg , rd_value , RISCV_DUT.datapath.Result) ;
                            end
                    end
                3'b110:
                    begin
                        $display("PC=%h %s Instruction | expected x%0d=%0d, actual is %0d" ,
                                PC_tb , "OR  " , rd_reg , rd_value , RISCV_DUT.datapath.Result) ;
                    end
                3'b111:
                    begin
                        $display("PC=%h %s Instruction | expected x%0d=%0d, actual is %0d" ,
                                PC_tb , "AND " , rd_reg , rd_value , RISCV_DUT.datapath.Result) ;
                    end
                endcase
            end
        //-- I-instructions
        7'b0010011: //addi,andi,ori
            begin
                case (Instr_tb[14:12])
                3'b000:
                    begin
                        $display("PC=%h %s Instruction | expected x%0d=%0d, actual is %0d" ,
                                PC_tb , "ADDI" , rd_reg , rd_value , RISCV_DUT.datapath.Result) ;
                    end
                3'b110:
                    begin
                        $display("PC=%h %s Instruction | expected x%0d=%0d, actual is %0d" ,
                                PC_tb , "ORI " , rd_reg , rd_value , RISCV_DUT.datapath.Result) ;
                    end
                3'b111:
                    begin
                        $display("PC=%h %s Instruction | expected x%0d=%0d, actual is %0d" ,
                                PC_tb , "ANDI" , rd_reg , rd_value , RISCV_DUT.datapath.Result) ;
                    end
                endcase
```

scoreboard.v

```
reg [31:0] rd_value ; // Destination Register Value

always @(*)
    begin
        case (PC_tb)
        32'h00: // x2 = 5
            begin
                rd_value = 32'd5 ;
            end
        32'h04: //  x3 = 12
            begin
                rd_value = 32'd12 ;
            end
        32'h08: //  x7 = (12 - 9) = 3
            begin
                rd_value = 32'd3 ;
            end
        32'h0C: //  x4 = (3 OR 5) = 7
            begin
                rd_value = 32'd7 ;
            end
        32'h10: //  x5 = (12 AND 7) = 4
            begin
                rd_value = 32'd4 ;
            end
        32'h14: //  x5 = 4 + 7 = 11
            begin
                rd_value = 32'd11 ;
            end
        32'h18: //  beq x5, x7, end (NO)
            begin
                rd_value = rd_reg ;
            end
        32'h1C: //  beq x4, x0, around (NO)
            begin
                rd_value = rd_reg ;
            end
        32'h20: //  addi x5, x0, 0
            begin
                rd_value = 32'd0 ;
            end
        32'h24: //  add x7, x4, x5  <-around
            begin
                rd_value = 32'd7 ;
            end
        32'h28: //  sub x7, x7, x2
            begin
                rd_value = 32'd2 ;
            end
        32'h2C: //  sw x7, 84(x3)
            begin
                rd_value = 32'd2 ;
            end
        32'h30: //  lw x2, 96(x0)
            begin
```

To finalize the system, we'll design a behavioral models of the Instruction Memory as 1KB ROM and the Data Memory as 1KB RAM, both are word-addressable (256 locations)

## Instruction Memory

```
module instr_memory (
      input    wire    [7:0]      i_A  ,
      output   wire    [31:0]     o_RD
      );

reg [31:0] mem [255:0] ; //1KB word-addressable ROM

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

assign  o_RD = mem[i_A] ;

endmodule
```

## Data Memory

```
module data_memory (
      input    wire               i_CLK   ,
      input    wire               i_Reset ,
      input    wire    [7:0]      i_A     ,
      input    wire               i_WE    ,
      input    wire    [31:0]     i_WD    ,
      output   wire    [31:0]     o_RD
      );

reg [31:0] mem [255:0] ; //1KB word-addressable RAM
integer i ;

always @(posedge i_CLK or posedge i_Reset)
      begin
            if (i_Reset)
                  begin
                        for (i=0 ; i<256 ; i=i+1)
                              begin
                                    mem[i] <= 32'b0 ;
                              end
                  end
            else if (i_WE)
                  begin
                        mem[i_A] <= i_WD ;
                  end
      end

assign  o_RD = mem[i_A] ;

endmodule
```

The first 19 locations in instruction memory are filled by the text file that includes our testcases of different instructions converted to the machine code in hexadecimal.

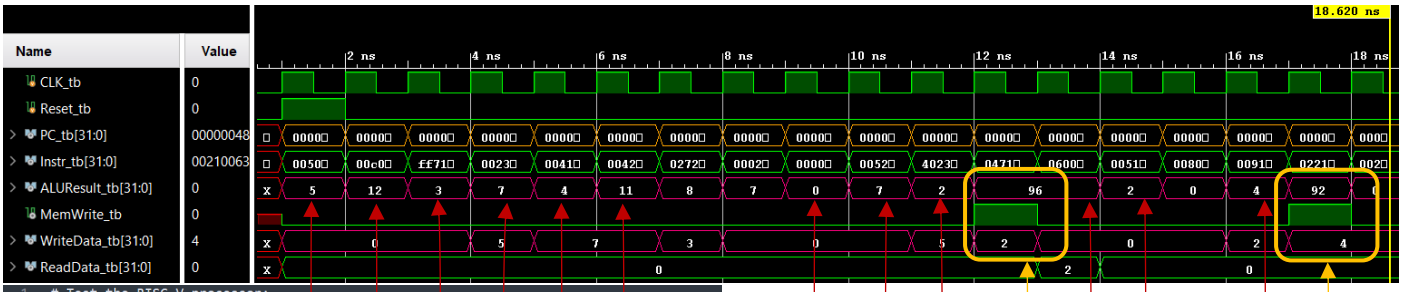
InstructionMem - Notepad

File Edit Format View Help

```
00500113
00C00193
FF718393
0023E233
0041F2B3
004282B3
02728863
00020463
00000293
005203B3
402383B3
0471AA23
06002103
005104B3
008001EF
00100113
00910133
0221A023
00210063
```

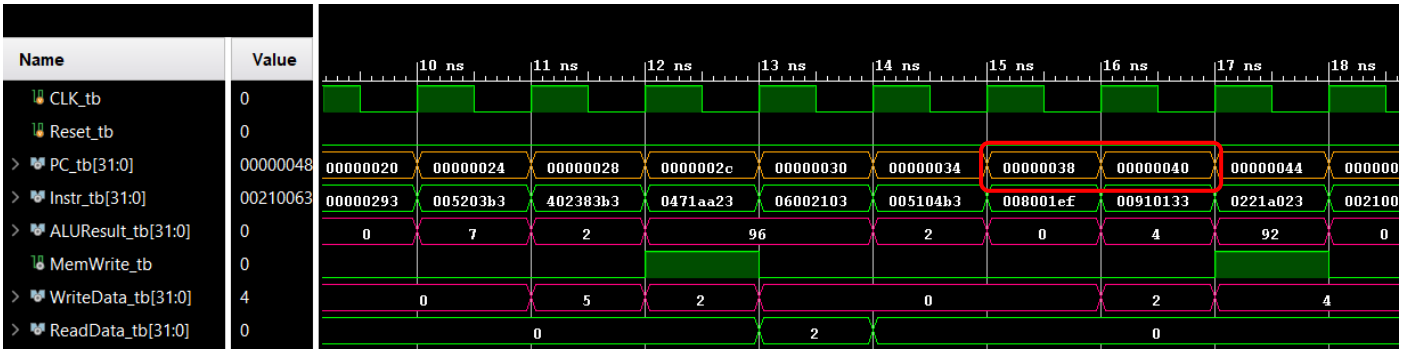
```
1   # Test the RISC-V processor:
2   # add, sub, and, or, addi, lw, sw, beq, jal
3   # If successful, it should write the value 4 to address 92
4
5   # RISC-V Assembly          Description               Address    Machine Code
6   main:   addi x2, x0, 5     # x2 = 5                  0          00500113
7           addi x3, x0, 12    # x3 = 12                 4          00C00193
8           addi x7, x3, -9    # x7 = (12 - 9) = 3       8          FF718393
9           or   x4, x7, x2    # x4 = (3 OR 5) = 7       C          0023E233
10          and  x5, x3, x4    # x5 = (12 AND 7) = 4     10         0041F2B3
11          add  x5, x5, x4    # x5 = 4 + 7 = 11         14         004282B3
12          beq  x5, x7, end   # shouldn't be taken      18         02728863
13          beq  x4, x0, around # shouldn't be taken     1C         00020463
14          addi x5, x0, 0     # x5 = 0 + 0 = 0          20         00000293
15  around: add  x7, x4, x5    # x7 = (7 + 0) = 7        24         005203B3
16          sub  x7, x7, x2    # x7 = (7 - 5) = 2        28         402383B3
17          sw   x7, 84(x3)    # [96] = 2                2C         0471AA23
18          lw   x2, 96(x0)    # x2 = [96] = 2           30         06002103
19          add  x9, x2, x5    # x9 = (2 + 0) = 2        34         005104B3
20          jal  x3, end       # jump to end, x3 = 0x3C  38         008001EF
21          addi x2, x0, 1     # shouldn't execute       3C         00100113
22  end:    add  x2, x2, x9    # x2 = (2 + 2) = 4        40         00910133
23          sw x2, 0x20(x3)    # [92] = 4                44         0221A023
24  done:   beq x2, x2, done   # infinite loop           48         00210063
```

# Testbench Simulation Results




```
# Test the RISC-V processor:
# add, sub, and, or, addi, lw, sw, beq, jal
# If successful, it should write the value 4 to address 92

# RISC-V Assembly          Description              Address    Machine Code
main:   addi x2, x0, 5     # x2 = 5                 0          00500113
        addi x3, x0, 12    # x3 = 12                4          00C00193
        addi x7, x3, -9    # x7 = (12 - 9) = 3      8          FF718393
        or   x4, x7, x2    # x4 = (3 OR 5) = 7      C          0023E233
        and  x5, x3, x4    # x5 = (12 AND 7) = 4    10         0041F2B3
        add  x5, x5, x4    # x5 = 4 + 7 = 11        14         004282B3
        beq  x5, x7, end   # shouldn't be taken     18         02728863
        beq  x4, x0, around # shouldn't be taken    1C         00020463
        addi x5, x0, 0     # x5 = 0 + 0 = 0         20         00000293
around: add  x7, x4, x5    # x7 = (7 + 0) = 7       24         005203B3
        sub  x7, x7, x2    # x7 = (7 - 5) = 2       28         402383B3
        sw   x7, 84(x3)    # [96] = 2               2C         0471AA23
        lw   x2, 96(x0)    # x2 = [96] = 2          30         06002103
        add  x9, x2, x5    # x9 = (2 + 0) = 2       34         005104B3
        jal  x3, end       # jump to end, x3 = 0x3C 38         008001EF
        addi x2, x0, 1     # shouldn't execute      3C         00100113
end:    add  x2, x2, x9    # x2 = (2 + 2) = 4       40         00910133
        sw x2, 0x20(x3)    # [92] = 4               44         0221A023
done:   beq x2, x2, done   # infinite loop          48         00210063
```

**Comment:** ALUResult is matched.




**Comment:** Jal instruction affects the PC and jumps from 0x38 to 0x40, ignoring 0x3C.

## Simulator TCL Console

```
PC=00000000 ADDI Instruction | expected x2=5, actual is 5
PC=00000004 ADDI Instruction | expected x3=12, actual is 12
PC=00000008 ADDI Instruction | expected x7=3, actual is 3
PC=0000000c OR   Instruction | expected x4=7, actual is 7
PC=00000010 AND  Instruction | expected x5=4, actual is 4
PC=00000014 ADD  Instruction | expected x5=11, actual is 11
PC=00000018 BEQ  Instruction
PC=0000001c BEQ  Instruction
PC=00000020 ADDI Instruction | expected x5=0, actual is 0
PC=00000024 ADD  Instruction | expected x7=7, actual is 7
PC=00000028 SUB  Instruction | expected x7=2, actual is 2
PC=0000002c SW   Instruction | expected x7=2, actual is 2
PC=00000030 LW   Instruction | expected x2=2, actual is 2
PC=00000034 ADD  Instruction | expected x9=2, actual is 2
PC=00000038 JAL  Instruction | expected x3=3c, actual is 3c
PC=00000040 ADD  Instruction | expected x2=4, actual is 4
PC=00000044 SW   Instruction | expected x2=4, actual is 4
PC=00000048 BEQ  Instruction
================================Data Memory Results================================
2 is stored successfully in data memory[96]
4 is stored successfully in data memory[92]
```