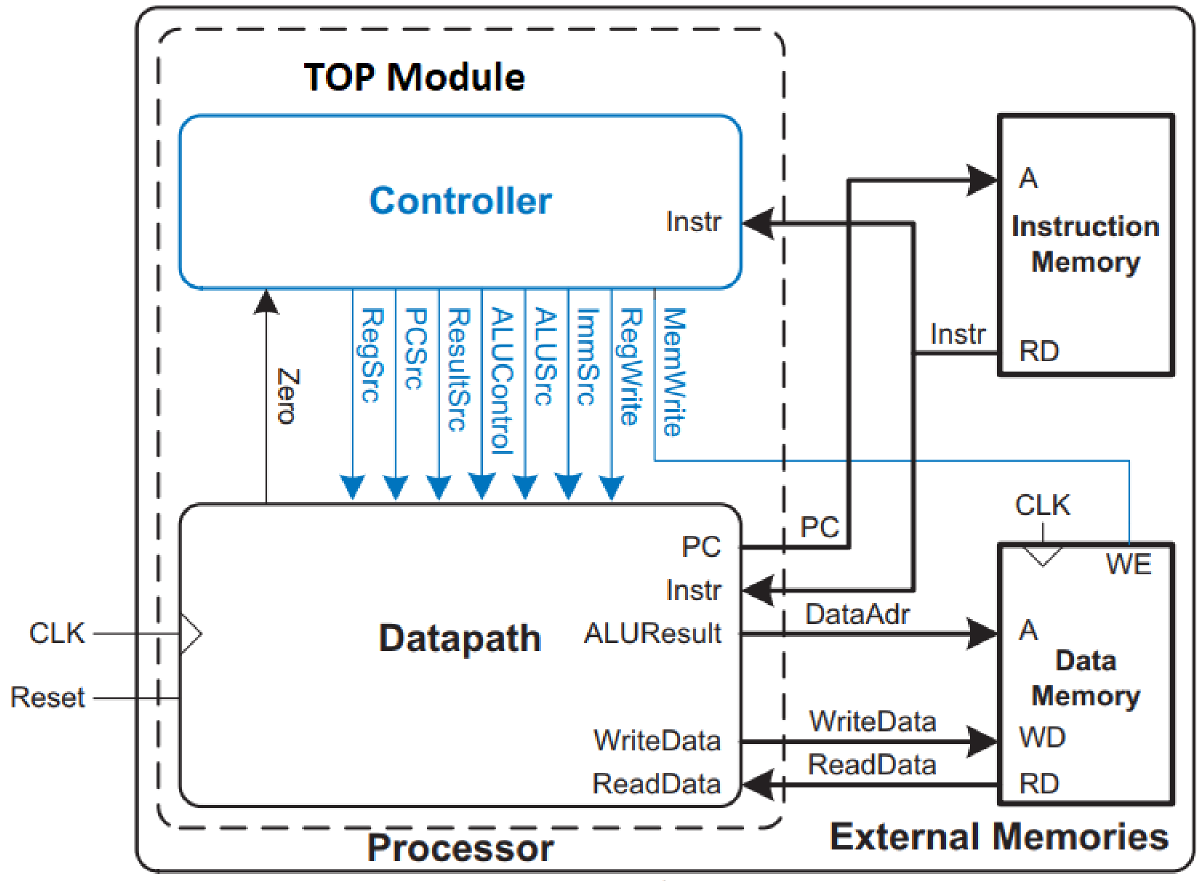
Abdallah Tarek Abdelnaby

Digital_design

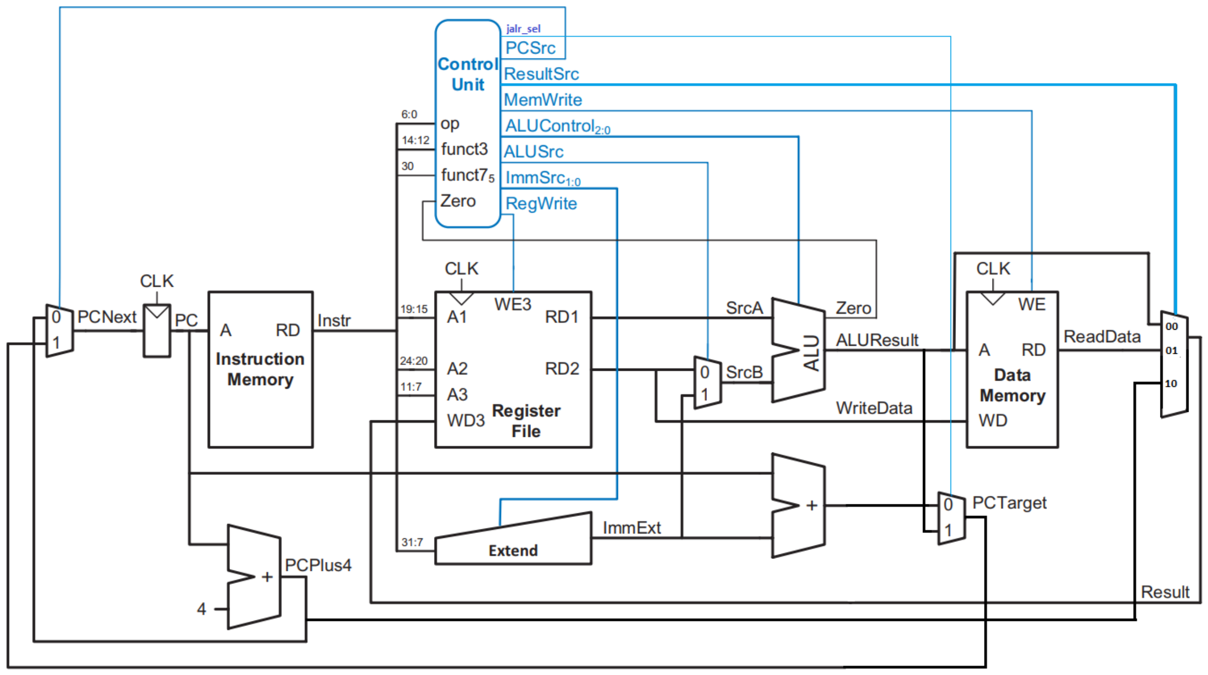
# Project 1: RISC-V Single cycle Processor

Reviewer: eng.sara

## The main block diagrams:



**Test_Beanch**



## The building blocks of the design:

- **Memories**
  - Data_memory: 32bit word addressable RAM

```
// 32bit word data memory
module ram (address,data_in,data_out,clk,we);

parameter length = 256;
parameter width = 32;

input clk,we;

input [width-1:0]address;
input [width-1:0]data_in;
output [width-1:0]data_out;
reg [width-1:0] mem [length-1:0];

wire [$clog2(length)-1:0]trunc_add;

function [$clog2(length)-1:0] address_trunc;
input [width-1:0]add;

address_trunc = add;
endfunction

assign trunc_add = address_trunc (address);
assign data_out = mem [trunc_add];

always@(posedge clk)
begin
    if (we)
        mem [trunc_add] = data_in;
    else
        mem [trunc_add] = mem [trunc_add];
end
endmodule
```

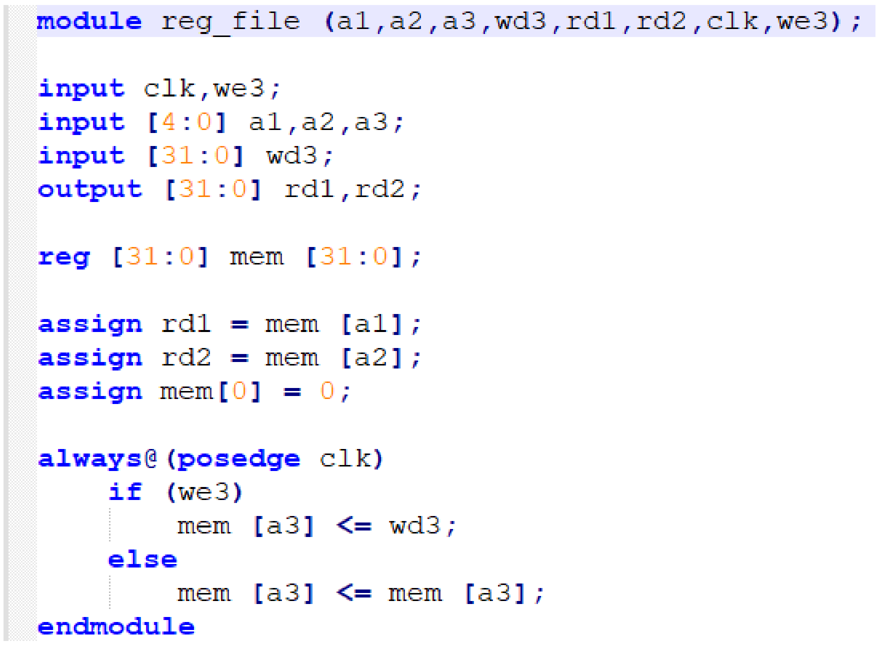
  - Instruction_memory: 32bit Byte addressable ROM

```
module instruction_memory (address,data_out,clk);

parameter length = 256;
parameter width = 32;

input clk;
input [width-1:0] address;
output [width-1:0] data_out;
reg [width-1:0] mem [length-1:0];

assign data_out = mem [address>>2];

//$readmemh("inst.mem", mem);
endmodule
```

- **Control unit**

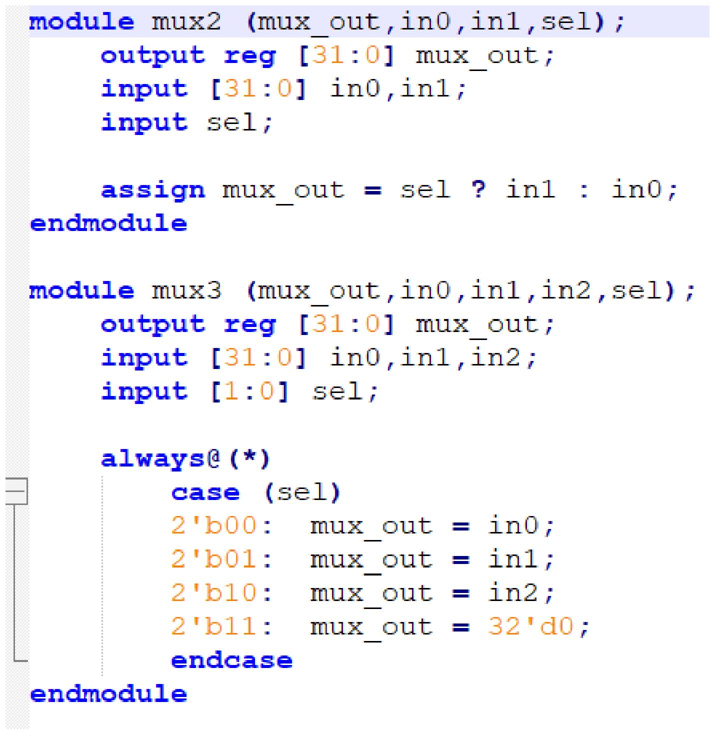
```verilog
//supported instructions lw,sw,and,or,add,sup,addi,andi,ori,slt,jal,jalr,bne,beq
module control_unit (instr,zero,pc_sel,result_sel,mem_write,alu_sel,imm_sel,reg_write,alu_control,jalr_sel);

localparam lw     = 7'b0000011;
localparam sw     = 7'b0100011;
localparam R_type = 7'b0110011;
localparam I_type = 7'b0010011;
localparam jal    = 7'b1101111;
localparam beq    = 7'b1100011;
localparam jalr   = 7'b1100111;

input zero;
input [31:0] instr;

output reg_write,pc_sel,alu_sel,mem_write,jalr_sel;
output reg [2:0] alu_control;
output [1:0] result_sel,imm_sel;

wire branch,jump;
wire [6:0] op_code = instr [6:0];
wire [2:0] func3 = instr [14:12];
wire func7_5 = instr [30];
wire [1:0] alu_operation;
reg [11:0] op;
reg bne_beq_sel;

assign reg_write = op [11];
assign imm_sel = op [10:9];
assign alu_sel = op [8];
assign mem_write = op [7];
assign result_sel = op [6:5];
assign branch = op [4];
assign jump = op [3];
assign alu_operation = op [2:1];
assign jalr_sel = op [0];
assign pc_sel = (bne_beq_sel & branch) | jump;
always@(op_code)    //main decoder
begin
    case (op_code)
//              reg_write   imm_sel   alu_sel   mem_write   result_sel   branch   jump   alu_operation   jalr_sel
    lw: op =     {1'b1,     2'b00,     1'b1,     1'b0,      2'b01,       1'b0,    1'b0,     2'b00,        1'b0};
    sw: op =     {1'b0,     2'b01,     1'b1,     1'b1,      2'b00,       1'b0,    1'b0,     2'b00,        1'b0};
    beq: op =    {1'b0,     2'b10,     1'b0,     1'b0,      2'b00,       1'b1,    1'b0,     2'b01,        1'b0};
    jal: op =    {1'b1,     2'b11,     1'b0,     1'b0,      2'b10,       1'b0,    1'b1,     2'b00,        1'b0};
    I_type: op = {1'b1,     2'b00,     1'b1,     1'b0,      2'b00,       1'b0,    1'b0,     2'b10,        1'b0};
    R_type: op = {1'b1,     2'b00,     1'b0,     1'b0,      2'b00,       1'b0,    1'b0,     2'b10,        1'b0};
    jalr: op =   {1'b1,     2'b00,     1'b1,     1'b0,      2'b10,       1'b0,    1'b1,     2'b00,        1'b1};
    default : op = op;
    endcase
end
always@(*)    //alu decoder
begin
    if (func3 == 3'b001)
        bne_beq_sel = ~zero;
    else
        bne_beq_sel = zero;
    alu_control = alu_control;
    if (alu_operation == 2'b0)
        alu_control = 3'b000;   //add //lw,sw
    else if (alu_operation == 2'b01)
        alu_control = 3'b001;   //sub  //beq
    else        //alu_operation = 2'b10     //R_type
    begin
        case (func3)
        3'b000:
            begin
                if ({op_code[5],func7_5} == 2'b11)
                    alu_control = 3'b001;     ///sub
                else
                    alu_control = 3'b000;     ///add
            end
        3'b010: alu_control = 3'b101;       //set less than ///slt
        3'b110: alu_control = 3'b011;       ///or
        3'b111: alu_control = 3'b010;       ///and
        default :   alu_control = alu_control;
        endcase
    end
end
endmodule
```

- **Datapath components:**
  - Regester_file:



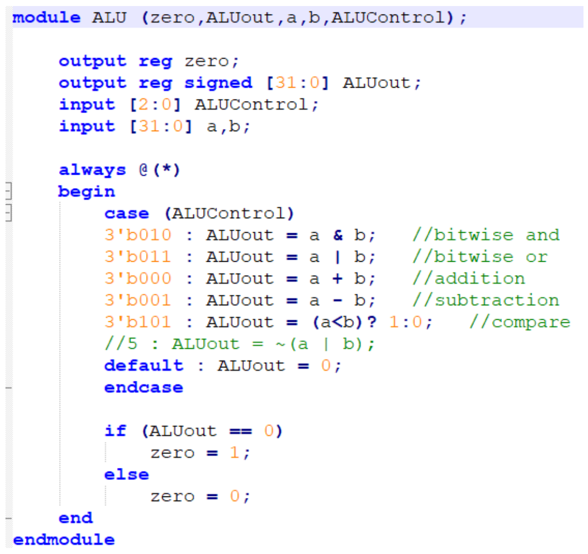
```
module reg_file (a1,a2,a3,wd3,rd1,rd2,clk,we3);

input clk,we3;
input [4:0] a1,a2,a3;
input [31:0] wd3;
output [31:0] rd1,rd2;

reg [31:0] mem [31:0];

assign rd1 = mem [a1];
assign rd2 = mem [a2];
assign mem[0] = 0;

always@(posedge clk)
    if (we3)
        mem [a3] <= wd3;
    else
        mem [a3] <= mem [a3];
endmodule
```

  - 2*1 MUX & 3*1 MUX:



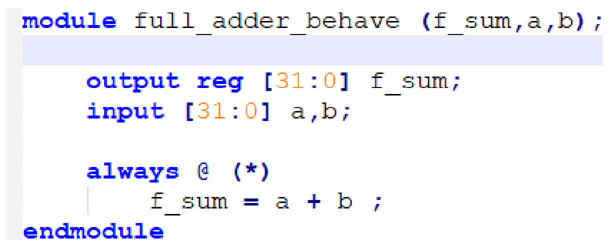
```
module mux2 (mux_out,in0,in1,sel);
    output reg [31:0] mux_out;
    input [31:0] in0,in1;
    input sel;

    assign mux_out = sel ? in1 : in0;
endmodule

module mux3 (mux_out,in0,in1,in2,sel);
    output reg [31:0] mux_out;
    input [31:0] in0,in1,in2;
    input [1:0] sel;

    always@(*)
        case (sel)
        2'b00:  mux_out = in0;
        2'b01:  mux_out = in1;
        2'b10:  mux_out = in2;
        2'b11:  mux_out = 32'd0;
        endcase
endmodule
```

- ALU:



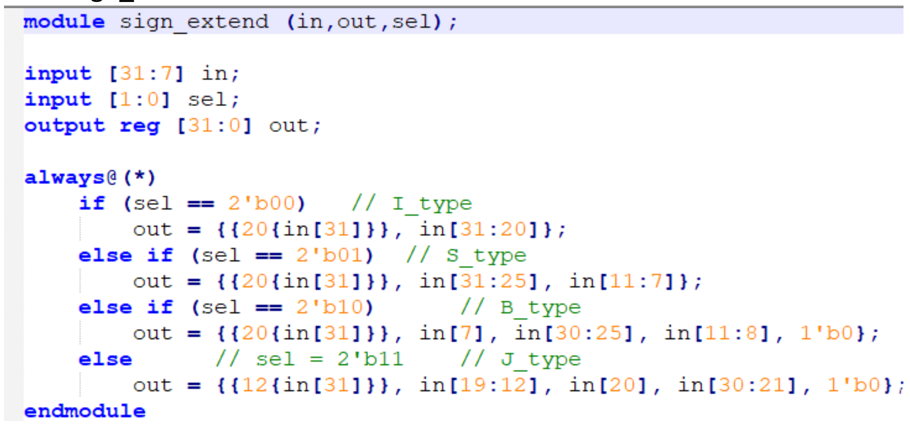
```
module ALU (zero,ALUout,a,b,ALUControl);

    output reg zero;
    output reg signed [31:0] ALUout;
    input [2:0] ALUControl;
    input [31:0] a,b;

    always @(*)
    begin
        case (ALUControl)
        3'b010 : ALUout = a & b;    //bitwise and
        3'b011 : ALUout = a | b;    //bitwise or
        3'b000 : ALUout = a + b;    //addition
        3'b001 : ALUout = a - b;    //subtraction
        3'b101 : ALUout = (a<b)? 1:0;    //compare
        //5 : ALUout = ~(a | b);
        default : ALUout = 0;
        endcase

        if (ALUout == 0)
            zero = 1;
        else
            zero = 0;
    end
endmodule
```

- Adder:



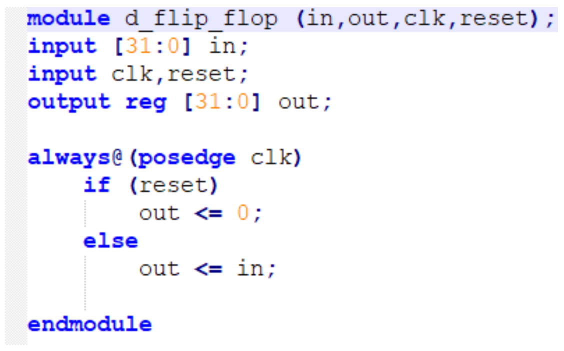
```
module full_adder_behave (f_sum,a,b);

    output reg [31:0] f_sum;
    input [31:0] a,b;

    always @ (*)
        f_sum = a + b ;
endmodule
```

- Sign_extend:



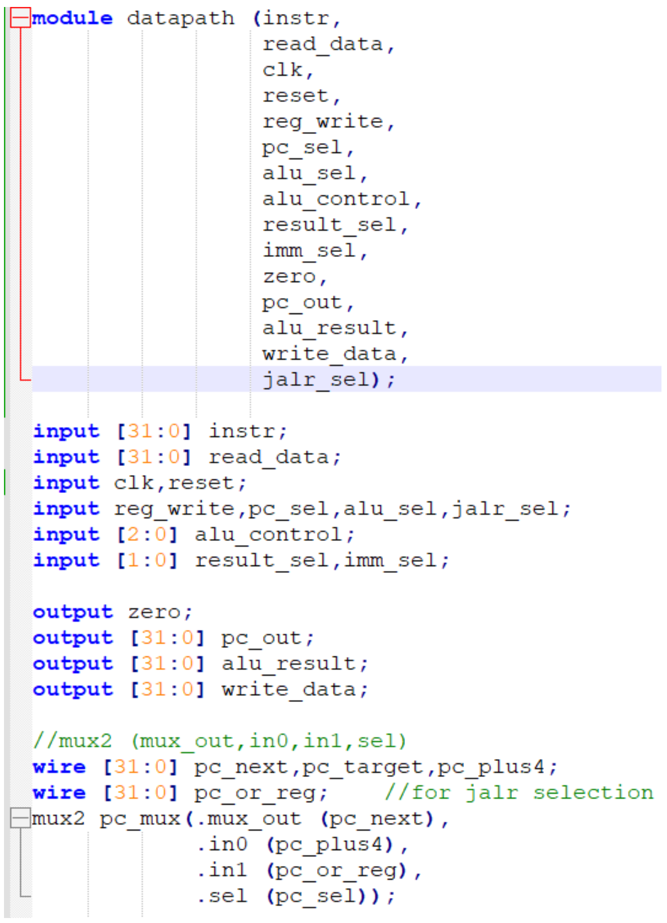
```
module sign_extend (in,out,sel);

input [31:7] in;
input [1:0] sel;
output reg [31:0] out;

always@(*)
    if (sel == 2'b00)    // I_type
        out = {{20{in[31]}}, in[31:20]};
    else if (sel == 2'b01)  // S_type
        out = {{20{in[31]}}, in[31:25], in[11:7]};
    else if (sel == 2'b10)      // B_type
        out = {{20{in[31]}}, in[7], in[30:25], in[11:8], 1'b0};
    else        // sel = 2'b11     // J_type
        out = {{12{in[31]}}, in[19:12], in[20], in[30:21], 1'b0};
endmodule
```

- PC_flip_flop:

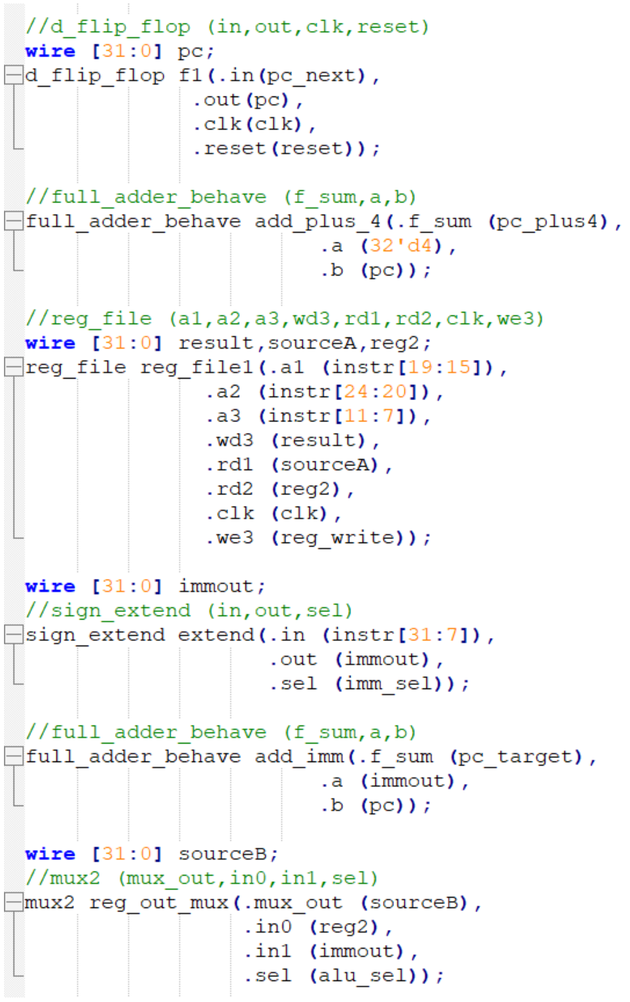


```
module d_flip_flop (in,out,clk,reset);
input [31:0] in;
input clk,reset;
output reg [31:0] out;

always@(posedge clk)
    if (reset)
        out <= 0;
    else
        out <= in;

endmodule
```

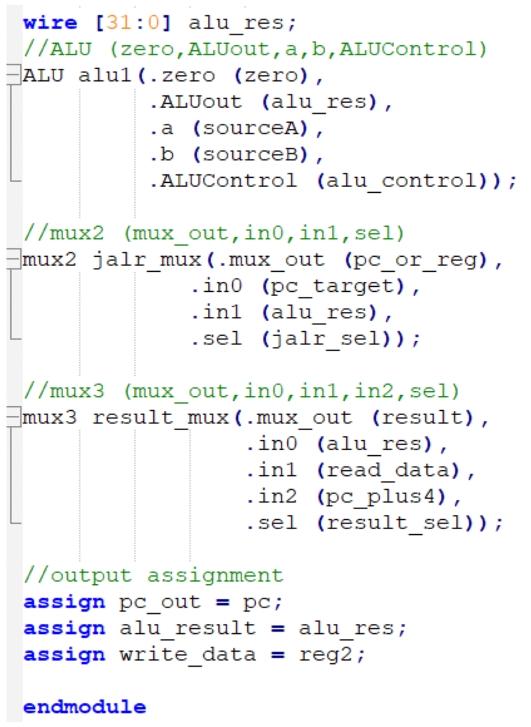
- **Datapath:**



```verilog
module datapath (instr,
                 read_data,
                 clk,
                 reset,
                 reg_write,
                 pc_sel,
                 alu_sel,
                 alu_control,
                 result_sel,
                 imm_sel,
                 zero,
                 pc_out,
                 alu_result,
                 write_data,
                 jalr_sel);

input [31:0] instr;
input [31:0] read_data;
input clk,reset;
input reg_write,pc_sel,alu_sel,jalr_sel;
input [2:0] alu_control;
input [1:0] result_sel,imm_sel;

output zero;
output [31:0] pc_out;
output [31:0] alu_result;
output [31:0] write_data;

//mux2 (mux_out,in0,in1,sel)
wire [31:0] pc_next,pc_target,pc_plus4;
wire [31:0] pc_or_reg;    //for jalr selection
mux2 pc_mux(.mux_out (pc_next),
            .in0 (pc_plus4),
            .in1 (pc_or_reg),
            .sel (pc_sel));
```

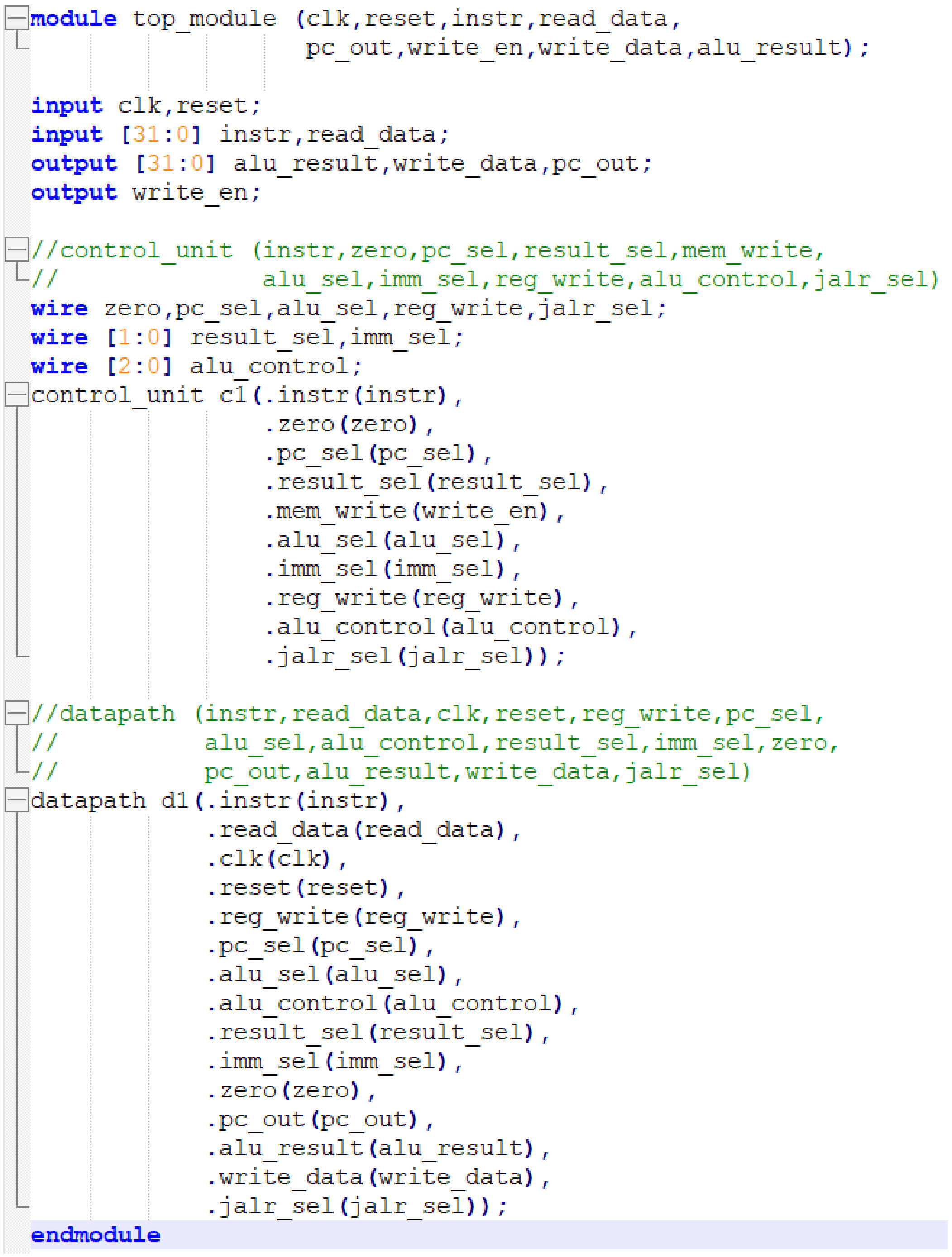


```verilog
//d_flip_flop (in,out,clk,reset)
wire [31:0] pc;
d_flip_flop f1(.in(pc_next),
               .out(pc),
               .clk(clk),
               .reset(reset));

//full_adder_behave (f_sum,a,b)
full_adder_behave add_plus_4(.f_sum (pc_plus4),
                             .a (32'd4),
                             .b (pc));

//reg_file (a1,a2,a3,wd3,rd1,rd2,clk,we3)
wire [31:0] result,sourceA,reg2;
reg_file reg_file1(.a1 (instr[19:15]),
                   .a2 (instr[24:20]),
                   .a3 (instr[11:7]),
                   .wd3 (result),
                   .rd1 (sourceA),
                   .rd2 (reg2),
                   .clk (clk),
                   .we3 (reg_write));

wire [31:0] immout;
//sign_extend (in,out,sel)
sign_extend extend(.in (instr[31:7]),
                   .out (immout),
                   .sel (imm_sel));

//full_adder_behave (f_sum,a,b)
full_adder_behave add_imm(.f_sum (pc_target),
                             .a (immout),
                             .b (pc));

wire [31:0] sourceB;
//mux2 (mux_out,in0,in1,sel)
mux2 reg_out_mux(.mux_out (sourceB),
                 .in0 (reg2),
                 .in1 (immout),
                 .sel (alu_sel));
```



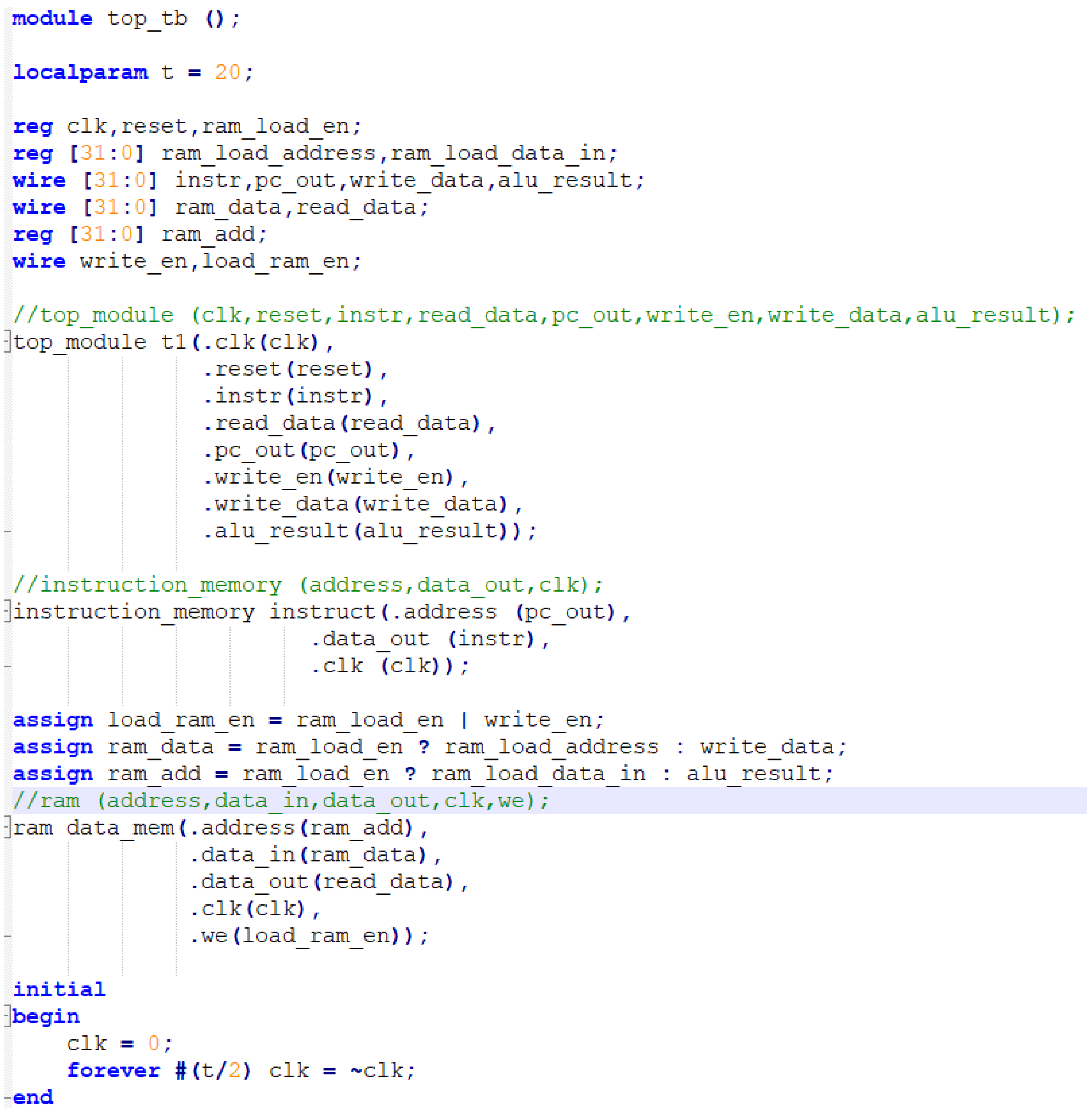
```verilog
wire [31:0] alu_res;
//ALU (zero,ALUout,a,b,ALUControl)
ALU alu1(.zero (zero),
         .ALUout (alu_res),
         .a (sourceA),
         .b (sourceB),
         .ALUControl (alu_control));

//mux2 (mux_out,in0,in1,sel)
mux2 jalr_mux(.mux_out (pc_or_reg),
              .in0 (pc_target),
              .in1 (alu_res),
              .sel (jalr_sel));

//mux3 (mux_out,in0,in1,in2,sel)
mux3 result_mux(.mux_out (result),
                .in0 (alu_res),
                .in1 (read_data),
                .in2 (pc_plus4),
                .sel (result_sel));

//output assignment
assign pc_out = pc;
assign alu_result = alu_res;
assign write_data = reg2;

endmodule
```

- **Top_module:**

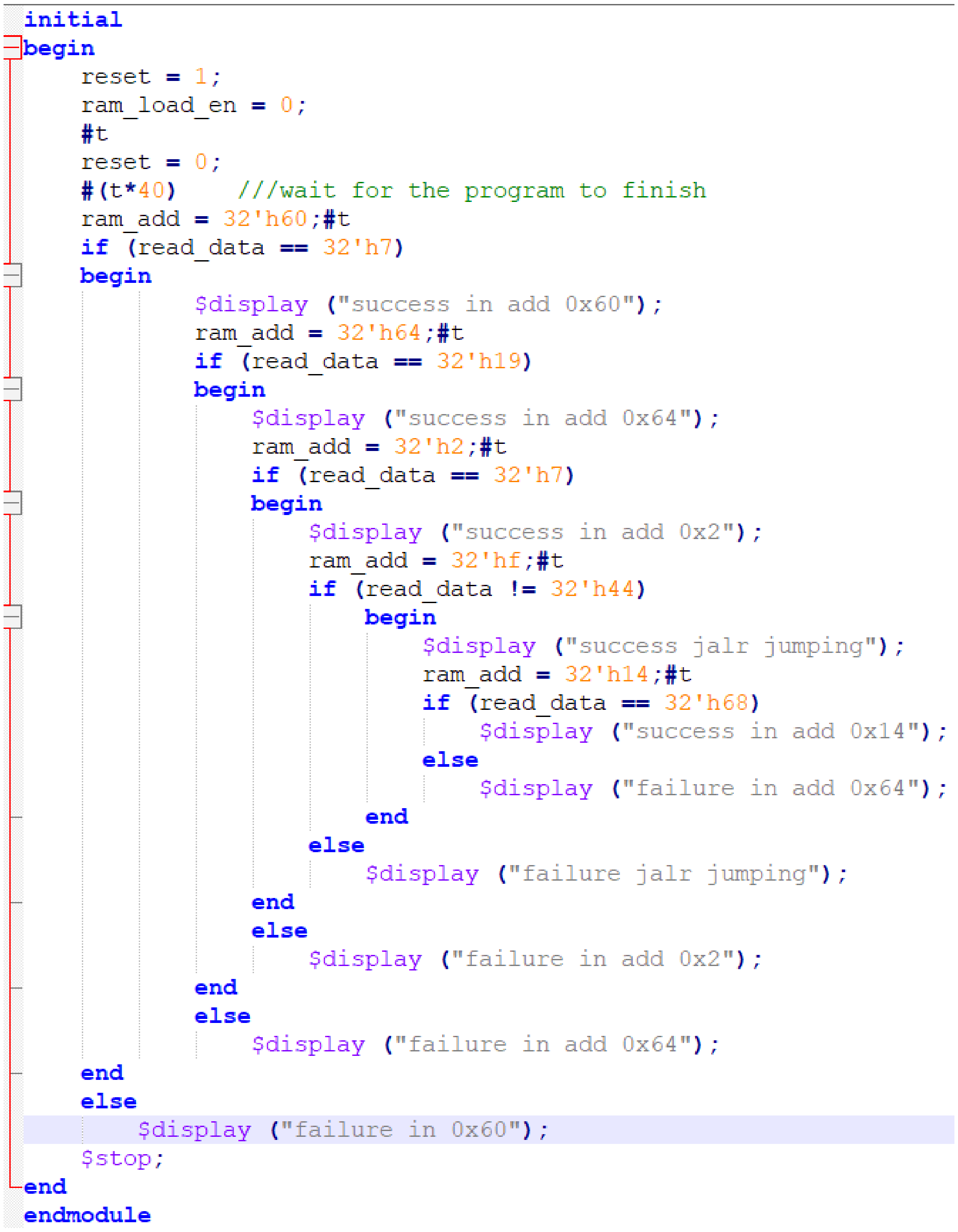


```
module top_module (clk,reset,instr,read_data,
                   pc_out,write_en,write_data,alu_result);

input clk,reset;
input [31:0] instr,read_data;
output [31:0] alu_result,write_data,pc_out;
output write_en;

//control_unit (instr,zero,pc_sel,result_sel,mem_write,
//                alu_sel,imm_sel,reg_write,alu_control,jalr_sel)
wire zero,pc_sel,alu_sel,reg_write,jalr_sel;
wire [1:0] result_sel,imm_sel;
wire [2:0] alu_control;
control_unit c1(.instr(instr),
                .zero(zero),
                .pc_sel(pc_sel),
                .result_sel(result_sel),
                .mem_write(write_en),
                .alu_sel(alu_sel),
                .imm_sel(imm_sel),
                .reg_write(reg_write),
                .alu_control(alu_control),
                .jalr_sel(jalr_sel));

//datapath (instr,read_data,clk,reset,reg_write,pc_sel,
//            alu_sel,alu_control,result_sel,imm_sel,zero,
//            pc_out,alu_result,write_data,jalr_sel)
datapath d1(.instr(instr),
            .read_data(read_data),
            .clk(clk),
            .reset(reset),
            .reg_write(reg_write),
            .pc_sel(pc_sel),
            .alu_sel(alu_sel),
            .alu_control(alu_control),
            .result_sel(result_sel),
            .imm_sel(imm_sel),
            .zero(zero),
            .pc_out(pc_out),
            .alu_result(alu_result),
            .write_data(write_data),
            .jalr_sel(jalr_sel));
endmodule
```

- **Test_bench:**



```verilog
module top_tb ();

localparam t = 20;

reg clk,reset,ram_load_en;
reg [31:0] ram_load_address,ram_load_data_in;
wire [31:0] instr,pc_out,write_data,alu_result;
wire [31:0] ram_data,read_data;
reg [31:0] ram_add;
wire write_en,load_ram_en;

//top_module (clk,reset,instr,read_data,pc_out,write_en,write_data,alu_result);
top_module t1(.clk(clk),
              .reset(reset),
              .instr(instr),
              .read_data(read_data),
              .pc_out(pc_out),
              .write_en(write_en),
              .write_data(write_data),
              .alu_result(alu_result));

//instruction_memory (address,data_out,clk);
instruction_memory instruct(.address (pc_out),
                            .data_out (instr),
                            .clk (clk));

assign load_ram_en = ram_load_en | write_en;
assign ram_data = ram_load_en ? ram_load_address : write_data;
assign ram_add = ram_load_en ? ram_load_data_in : alu_result;
//ram (address,data_in,data_out,clk,we);
ram data_mem(.address(ram_add),
             .data_in(ram_data),
             .data_out(read_data),
             .clk(clk),
             .we(load_ram_en));

initial
begin
    clk = 0;
    forever #(t/2) clk = ~clk;
end
```



```
initial
begin
    reset = 1;
    ram_load_en = 0;
    #t
    reset = 0;
    #(t*40)    ///wait for the program to finish
    ram_add = 32'h60;#t
    if (read_data == 32'h7)
    begin
            $display ("success in add 0x60");
            ram_add = 32'h64;#t
            if (read_data == 32'h19)
            begin
                $display ("success in add 0x64");
                ram_add = 32'h2;#t
                if (read_data == 32'h7)
                begin
                    $display ("success in add 0x2");
                    ram_add = 32'hf;#t
                    if (read_data != 32'h44)
                        begin
                            $display ("success jalr jumping");
                            ram_add = 32'h14;#t
                            if (read_data == 32'h68)
                                $display ("success in add 0x14");
                            else
                                $display ("failure in add 0x64");
                        end
                    else
                        $display ("failure jalr jumping");
                end
                else
                    $display ("failure in add 0x2");
            end
            else
                $display ("failure in add 0x64");
    end
    else
        $display ("failure in 0x60");
    $stop;
end
endmodule
```

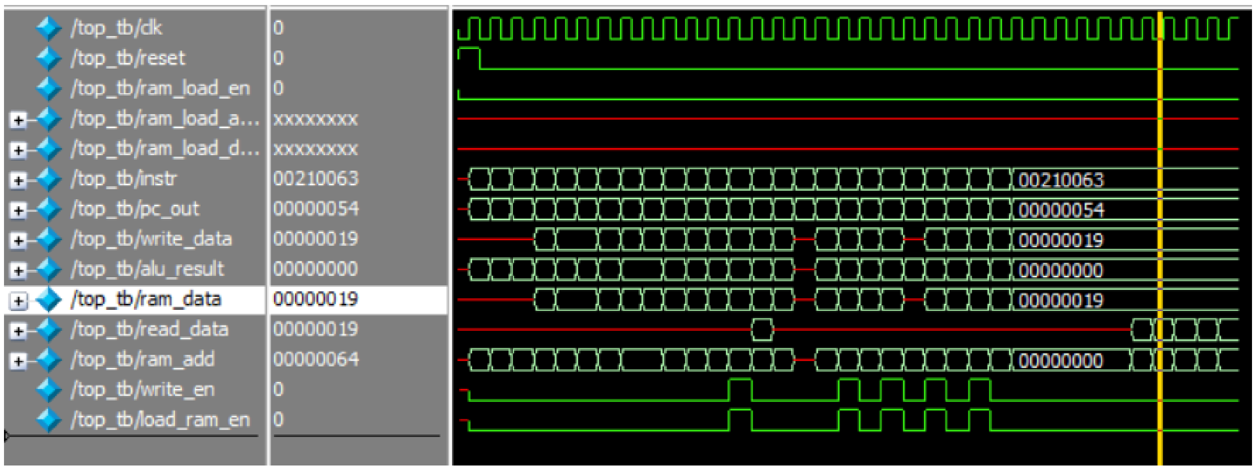
## The program loaded in the instruction memory:

```
main: addi x2, x0, 5 # x2 = 5 (0) 0x00500113

addi x3, x0, 12 # x3 = 12 (4)  0x00C00193

addi x7, x3, -9 # x7 = (12 - 9) = 3 (8) 0xFF718393

or x4, x7, x2 # x4 = (3 OR 5) = 7 (C)   0x0023E233

and x5, x3, x4 # x5 = (12 AND 7) = 4 (10) 0x0041F2B3

add x5, x5, x4 # x5 = 4 + 7 = 11 (14) 0x004282B3

beq x5, x7, end # shouldn't be taken (18) 0x02728863

slt x4, x3, x4 # x4 = (12 < 7) = 0 (1C) 0x0041A233

beq x4, x0, around # should be taken (20) 0x00020463

addi x5, x0, 0 # shouldn't execute (24) 0x00000293

around: slt x4, x7, x2 # x4 = (3 < 5) = 1 (28) 0x0023A233

add x7, x4, x5 # x7 = (1 + 11) = 12 (2C) 0x005203B3

sub x7, x7, x2 # x7 = (12 - 5) = 7 (30) 0x402383B3

sw x7, 84(x3) # [96] = 7 (34) 0x0471AA23

lw x2, 96(x0) # x2 = [96] = 7 (38) 0x06002103

add x9, x2, x5 # x9 = (7 + 11) = 18 (3C) 0x005104B3

jal x3, end # jump to end, x3 = 0x44 (40) 0x008001EF

addi x2, x0, 1 # shouldn't execute (44) 0x00100113

end: add x2, x2, x9 # x2 = (7 + 18) = 25 (48) 0x00910133

sw x2, 0x20(x3) # [100] = 25 (4c) 0x0221A023

bne x7,x5,test_bne # should be taken (50) 0x00539463

done: beq x2, x2, done # infinite loop (54) 0x00210063

test_bne: sw x7, 2(x0) # [2] = 7 (58) 0x00702123

addi x11, x0, 8 # x11 = 8 (5c) 0x00800593

sw x11, 15(x0) # [15] = 8 (60) 0x00B027A3

my_place: jalr x7, x11, my_place  #(64) 0x060583E7

sw x3, 15(x0) # [15] != 0x44 (68) 0x003027A3

test_jlr: sw x7, 20(x0) # [20] = 0x68 (6c) 0x00702A23

bne x7,x5,done # should be taken (70) 0xFE5394E3
```

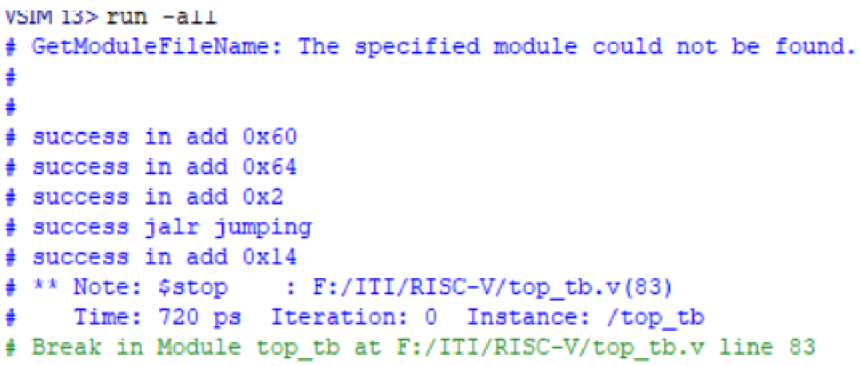
## The machine code loaded in the instruction memory:

```
0x00500113

0x00C00193

0xFF718393

0x0023E233

0x0041F2B3

0x004282B3

0x02728863

0x0041A233

0x00020463

0x00000293

0x0023A233

0x005203B3

0x402383B3

0x0471AA23

0x06002103

0x005104B3

0x008001EF

0x00100113

0x00910133

0x0221A023

0x00539463

0x00210063

0x00702123

0x00800593

0x00B027A3

0x064583E7

0x003027A3

0x00702A23

0xFE5392E3
```

## The result from test bench:



```
/top_tb/clk            0
/top_tb/reset          0
/top_tb/ram_load_en    0
/top_tb/ram_load_a...  xxxxxxxx
/top_tb/ram_load_d...  xxxxxxxx
/top_tb/instr          00210063
/top_tb/pc_out         00000054
/top_tb/write_data     00000019
/top_tb/alu_result     00000000
/top_tb/ram_data       00000019
/top_tb/read_data      00000019
/top_tb/ram_add        00000064
/top_tb/write_en       0
/top_tb/load_ram_en    0
```



```
VSIM 13> run -all
# GetModuleFileName: The specified module could not be found.
#
#
# success in add 0x60
# success in add 0x64
# success in add 0x2
# success jalr jumping
# success in add 0x14
# ** Note: $stop    : F:/ITI/RISC-V/top_tb.v(83)
#    Time: 720 ps  Iteration: 0  Instance: /top_tb
# Break in Module top_tb at F:/ITI/RISC-V/top_tb.v line 83
```

## Comments:

- It stored in the address 0x60 (0d96) 0x7 which is the value of 0x7 register in the register file which verify the following instructions (as 0x7 can't have 0x7 value] unless the success of the instructions) :

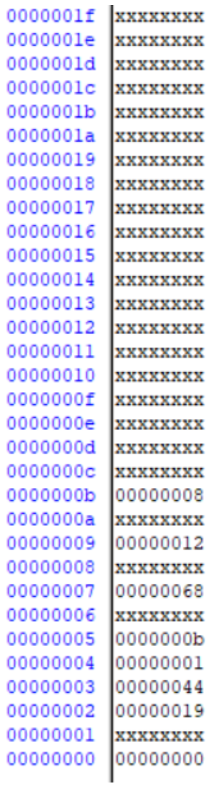
  Sw, add, sub, beq, slt, addi, or, and.
- It stored in the address 0x64 (0d100) 0x19 (0d25) which is the value of 0x2 register in the register file which verify the following instructions (as 0x2 can't have 0x19 (0d25) value unless the success of the instructions) :

  lw, jal.
- It stored in the address 0x2 (0d2) 0x7 (0d7) which is the value of 0x7 register in the register file which verify the following instructions (as this instruction won't be executed if it wasn't for bne (bnq jumped is infinite loop)):

  bne.
- It didn't store in the address 0x3 (0d3) 0x44 (0d68) which is the value of 0x3 register in the register file which verify the jumping part of the following instructions (as this instruction jumps the sw instruction successfully):

  jalr.
- It didn't store in the address 0x14 (0d20) 0x68 (0d104) which is the value of 0x7 register in the register file which verify the storing of the return (pc+4) part of the following instructions (as this instruction stores the next instruction address (pc+4) in the 0x7 (destination register) successfully): jalr.

reg_file



| | |
|---|---|
| 0000001f | xxxxxxxx |
| 0000001e | xxxxxxxx |
| 0000001d | xxxxxxxx |
| 0000001c | xxxxxxxx |
| 0000001b | xxxxxxxx |
| 0000001a | xxxxxxxx |
| 00000019 | xxxxxxxx |
| 00000018 | xxxxxxxx |
| 00000017 | xxxxxxxx |
| 00000016 | xxxxxxxx |
| 00000015 | xxxxxxxx |
| 00000014 | xxxxxxxx |
| 00000013 | xxxxxxxx |
| 00000012 | xxxxxxxx |
| 00000011 | xxxxxxxx |
| 00000010 | xxxxxxxx |
| 0000000f | xxxxxxxx |
| 0000000e | xxxxxxxx |
| 0000000d | xxxxxxxx |
| 0000000c | xxxxxxxx |
| 0000000b | 00000008 |
| 0000000a | xxxxxxxx |
| 00000009 | 00000012 |
| 00000008 | xxxxxxxx |
| 00000007 | 00000068 |
| 00000006 | xxxxxxxx |
| 00000005 | 0000000b |
| 00000004 | 00000001 |
| 00000003 | 00000044 |
| 00000002 | 00000019 |
| 00000001 | xxxxxxxx |
| 00000000 | 00000000 |