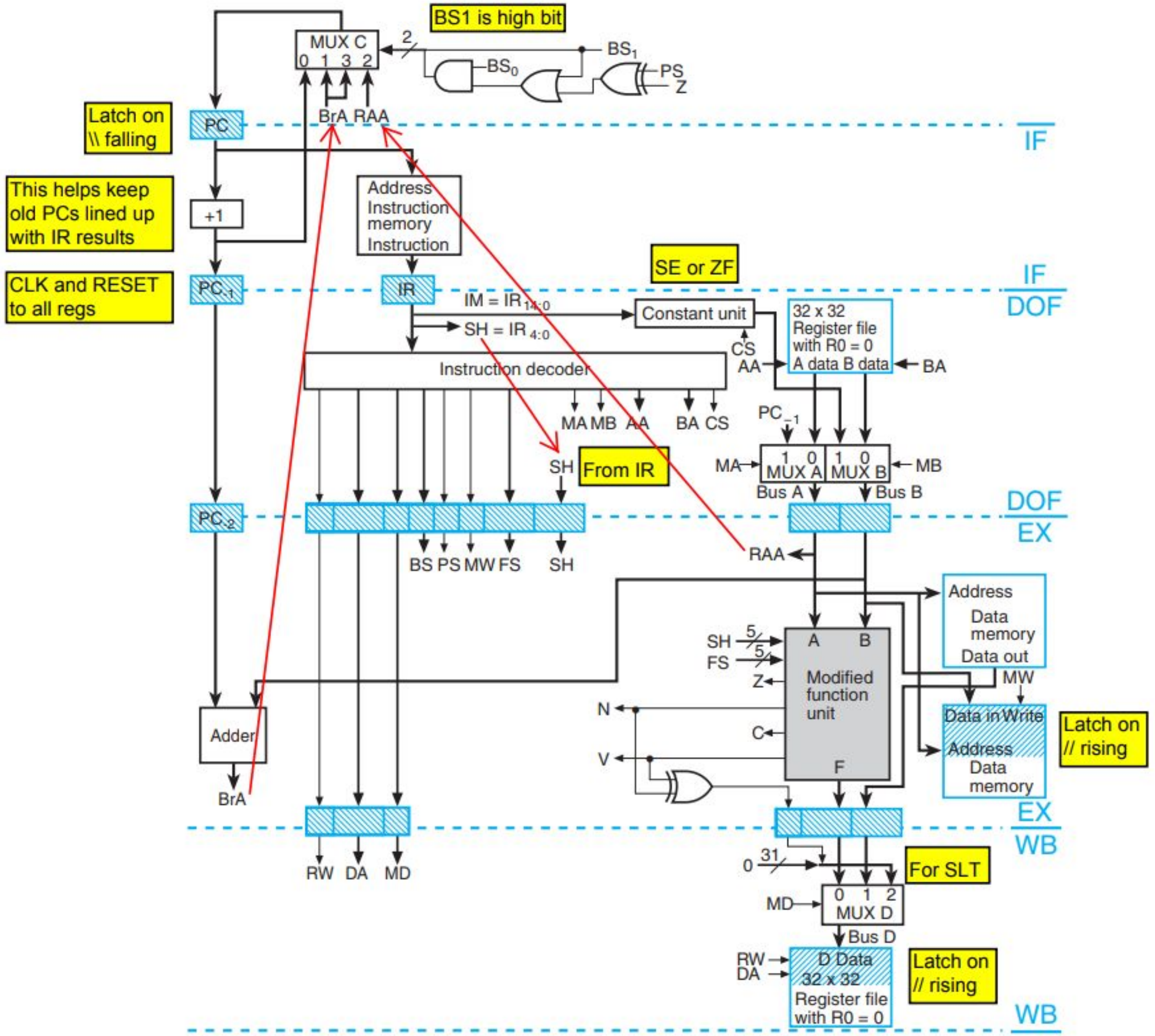
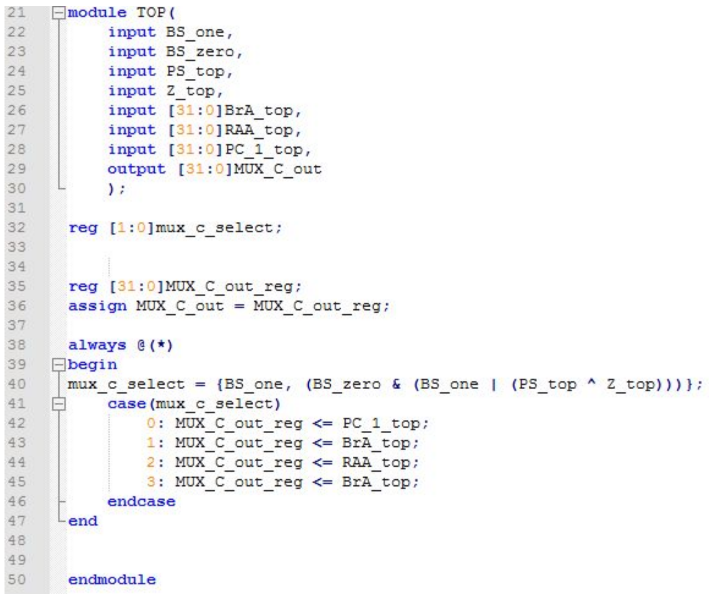
RISC CPU

Austin Goodman

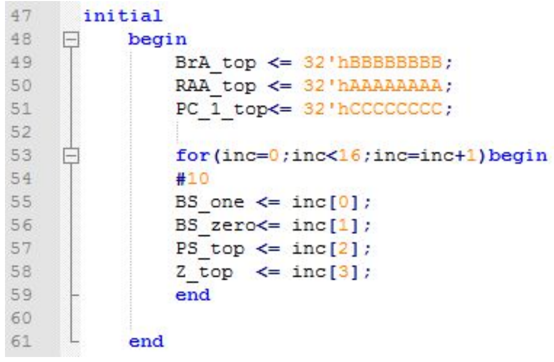

BS1 is high bit
MUX C
0 1 3 2
2
BS1
BS0
PS
Z
Latch on \\ falling
PC
BrA RAA
IF
This helps keep old PCs lined up with IR results
+1
Address
Instruction memory
Instruction
CLK and RESET to all regs
SE or ZF
PC-1
IR
IF
DOF
IM = IR 14:0
SH = IR 4:0
Constant unit
32 x 32 Register file with R0 = 0
A data B data
CS
AA
BA
Instruction decoder
MA MB AA BA CS
PC-1
From IR
SH
MA
MUX A
MUX B
MB
Bus A
Bus B
PC-2
DOF
EX
BS PS MW FS SH
RAA
Address
Data memory
Data out
SH
FS
Z
Modified function unit
A B F
N C V
MW
Data in Write
Address
Data memory
Latch on // rising
Adder
BrA
EX
WB
RW DA MD
0 31
For SLT
MD
MUX D
0 1 2
Bus D
RW
DA
D Data
32 x 32 Register file with R0 = 0
Latch on // rising
WB

Top portion

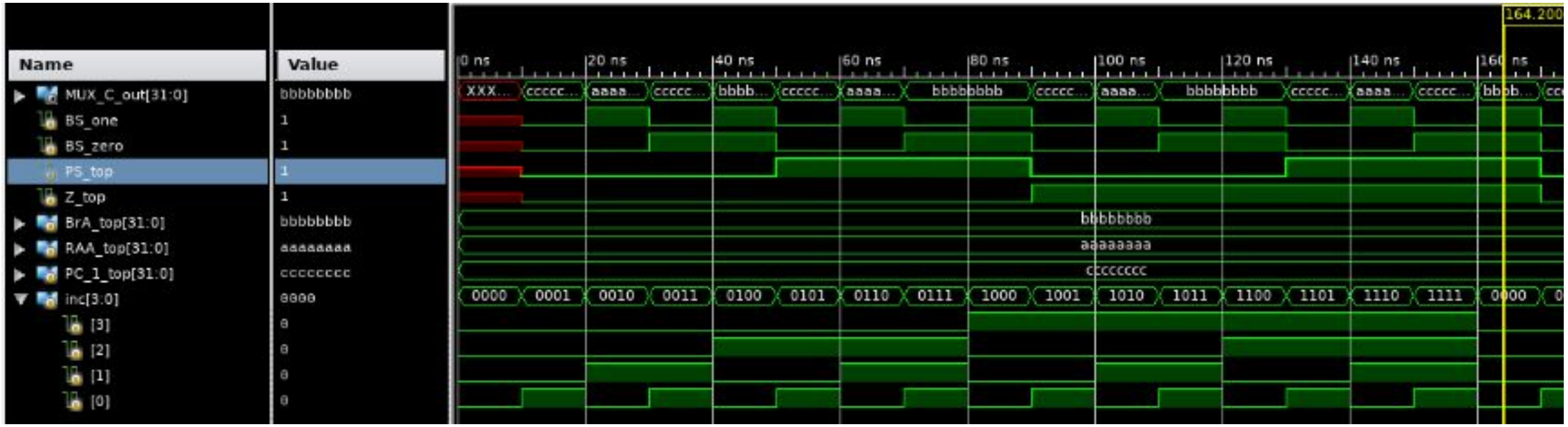


```
module TOP(
    input BS_one,
    input BS_zero,
    input PS_top,
    input Z_top,
    input [31:0]BrA_top,
    input [31:0]RAA_top,
    input [31:0]PC_1_top,
    output [31:0]MUX_C_out
    );

reg [1:0]mux_c_select;


reg [31:0]MUX_C_out_reg;
assign MUX_C_out = MUX_C_out_reg;

always @(*)
begin
mux_c_select = {BS_one, (BS_zero & (BS_one | (PS_top ^ Z_top)))};
    case(mux_c_select)
        0: MUX_C_out_reg <= PC_1_top;
        1: MUX_C_out_reg <= BrA_top;
        2: MUX_C_out_reg <= RAA_top;
        3: MUX_C_out_reg <= BrA_top;
    endcase
end


endmodule
```

TEST BENCH

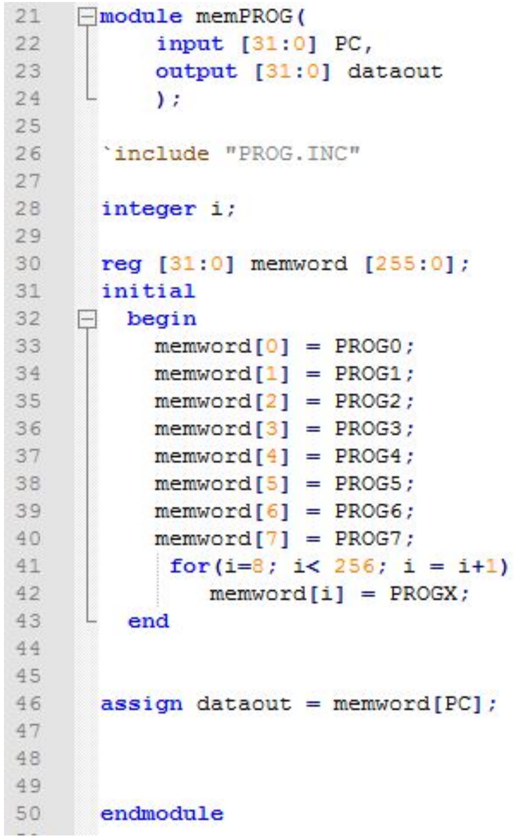


```
initial
    begin
        BrA_top <= 32'hBBBBBBBB;
        RAA_top <= 32'hAAAAAAAA;
        PC_1_top<= 32'hCCCCCCCC;

        for(inc=0;inc<16;inc=inc+1)begin
        #10
        BS_one <= inc[0];
        BS_zero<= inc[1];
        PS_top <= inc[2];
        Z_top  <= inc[3];
        end

    end
```



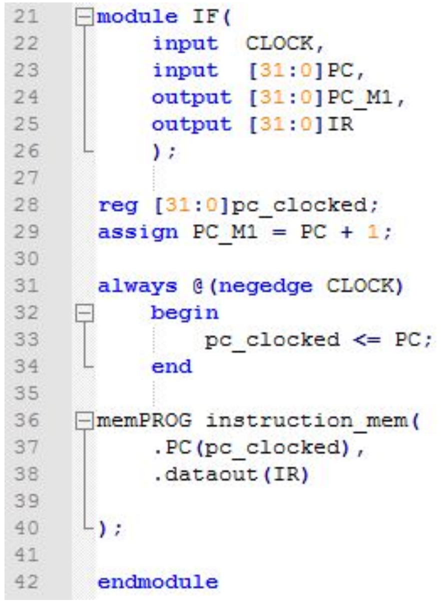
Name	Value
MUX_C_out[31:0]	bbbbbbbb
BS_one	1
BS_zero	1
PS_top	1
Z_top	1
BrA_top[31:0]	bbbbbbbb
RAA_top[31:0]	aaaaaaaa
PC_1_top[31:0]	cccccccc
inc[3:0]	0000
[3]	0
[2]	0
[1]	0
[0]	0

Instruction Memory

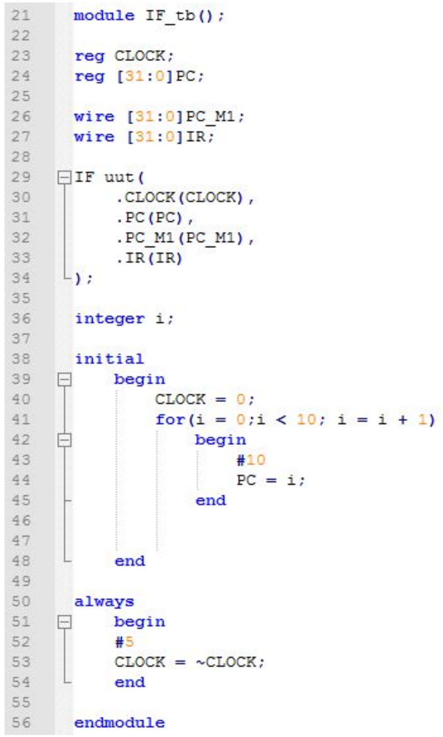


```
module memPROG(
        input [31:0] PC,
        output [31:0] dataout
        );

`include "PROG.INC"

integer i;

reg [31:0] memword [255:0];
initial
  begin
    memword[0] = PROG0;
    memword[1] = PROG1;
    memword[2] = PROG2;
    memword[3] = PROG3;
    memword[4] = PROG4;
    memword[5] = PROG5;
    memword[6] = PROG6;
    memword[7] = PROG7;
     for(i=8; i< 256; i = i+1)
        memword[i] = PROGX;
  end


assign dataout = memword[PC];



endmodule
```

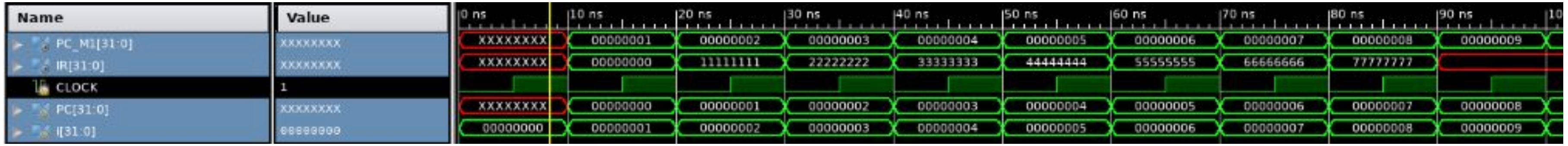
IF



```verilog
module IF(
        input   CLOCK,
        input   [31:0]PC,
        output  [31:0]PC_M1,
        output  [31:0]IR
        );

reg [31:0]pc_clocked;
assign PC_M1 = PC + 1;

always @(negedge CLOCK)
        begin
            pc_clocked <= PC;
        end

memPROG instruction_mem(
        .PC(pc_clocked),
        .dataout(IR)

);

endmodule
```

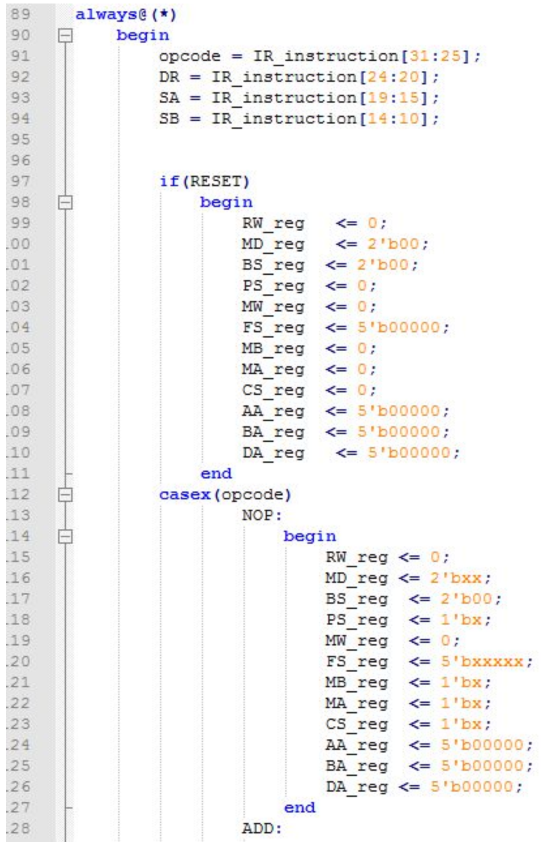


```verilog
module IF_tb();

reg CLOCK;
reg [31:0]PC;

wire [31:0]PC_M1;
wire [31:0]IR;

IF uut(
        .CLOCK(CLOCK),
        .PC(PC),
        .PC_M1(PC_M1),
        .IR(IR)
);

integer i;

initial
    begin
        CLOCK = 0;
        for(i = 0;i < 10; i = i + 1)
            begin
                #10
                PC = i;
            end


    end

always
    begin
    #5
    CLOCK = ~CLOCK;
    end

endmodule
```

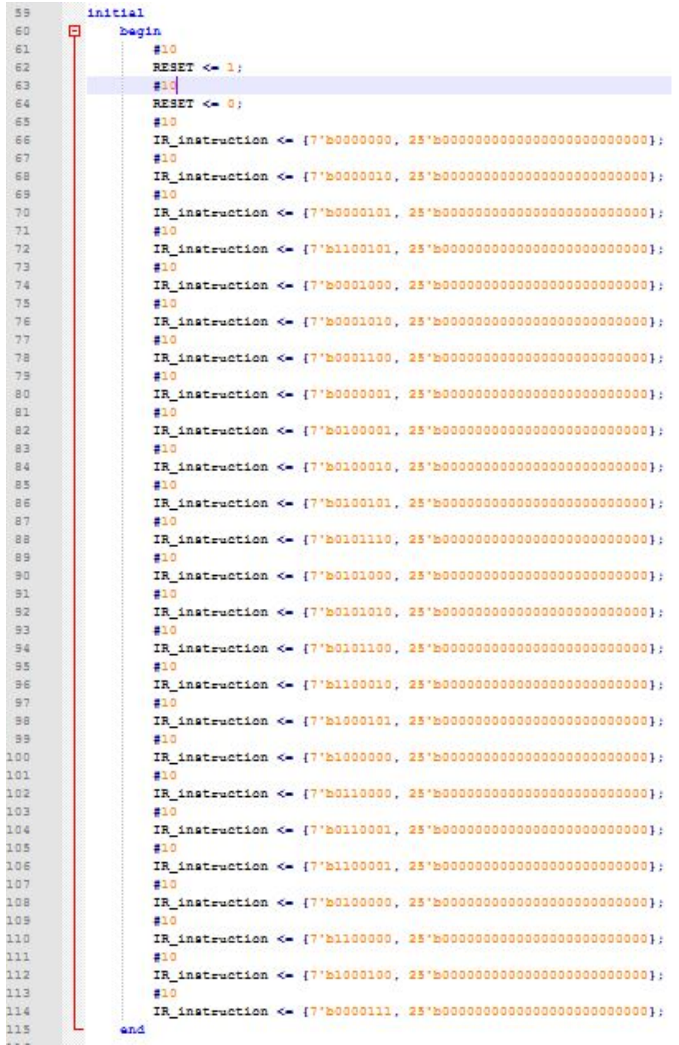


```
Name         Value     |0 ns      |10 ns     |20 ns     |30 ns     |40 ns     |50 ns     |60 ns     |70 ns     |80 ns     |90 ns     |10
PC_M1[31:0]  XXXXXXXX   XXXXXXXX   00000001   00000002   00000003   00000004   00000005   00000006   00000007   00000008   00000009
IR[31:0]     XXXXXXXX   XXXXXXXX   00000000   11111111   22222222   33333333   44444444   55555555   66666666   77777777
CLOCK        1
PC[31:0]     XXXXXXXX   XXXXXXXX   00000000   00000001   00000002   00000003   00000004   00000005   00000006   00000007   00000008
i[31:0]      88888888   00000000   00000001   00000002   00000003   00000004   00000005   00000006   00000007   00000008   00000009
```

Instruction Decoder

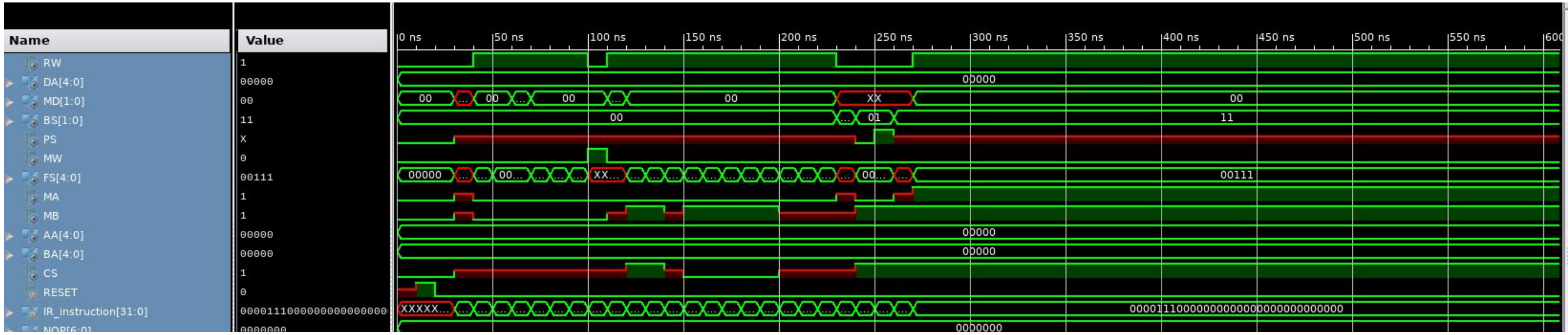


```verilog
always@(*)
    begin
        opcode = IR_instruction[31:25];
        DR = IR_instruction[24:20];
        SA = IR_instruction[19:15];
        SB = IR_instruction[14:10];


        if(RESET)
            begin
                RW_reg   <= 0;
                MD_reg   <= 2'b00;
                BS_reg  <= 2'b00;
                PS_reg  <= 0;
                MW_reg  <= 0;
                FS_reg  <= 5'b00000;
                MB_reg  <= 0;
                MA_reg  <= 0;
                CS_reg  <= 0;
                AA_reg  <= 5'b00000;
                BA_reg  <= 5'b00000;
                DA_reg   <= 5'b00000;
            end
        casex(opcode)
                NOP:
                    begin
                        RW_reg <= 0;
                        MD_reg <= 2'bxx;
                        BS_reg  <= 2'b00;
                        PS_reg  <= 1'bx;
                        MW_reg  <= 0;
                        FS_reg  <= 5'bxxxxx;
                        MB_reg  <= 1'bx;
                        MA_reg  <= 1'bx;
                        CS_reg  <= 1'bx;
                        AA_reg  <= 5'b00000;
                        BA_reg  <= 5'b00000;
                        DA_reg <= 5'b00000;
                    end
                ADD:
```



```verilog
initial
    begin
        #10
        RESET <= 1;
        #10
        RESET <= 0;
        #10
        IR_instruction <= {7'b0000000, 25'b0000000000000000000000000};
        #10
        IR_instruction <= {7'b0000010, 25'b0000000000000000000000000};
        #10
        IR_instruction <= {7'b0000101, 25'b0000000000000000000000000};
        #10
        IR_instruction <= {7'b1100101, 25'b0000000000000000000000000};
        #10
        IR_instruction <= {7'b0001000, 25'b0000000000000000000000000};
        #10
        IR_instruction <= {7'b0001010, 25'b0000000000000000000000000};
        #10
        IR_instruction <= {7'b0001100, 25'b0000000000000000000000000};
        #10
        IR_instruction <= {7'b0000001, 25'b0000000000000000000000000};
        #10
        IR_instruction <= {7'b0100001, 25'b0000000000000000000000000};
        #10
        IR_instruction <= {7'b0100010, 25'b0000000000000000000000000};
        #10
        IR_instruction <= {7'b0100101, 25'b0000000000000000000000000};
        #10
        IR_instruction <= {7'b0101110, 25'b0000000000000000000000000};
        #10
        IR_instruction <= {7'b0101000, 25'b0000000000000000000000000};
        #10
        IR_instruction <= {7'b0101010, 25'b0000000000000000000000000};
        #10
        IR_instruction <= {7'b0101100, 25'b0000000000000000000000000};
        #10
        IR_instruction <= {7'b1100010, 25'b0000000000000000000000000};
        #10
        IR_instruction <= {7'b1000101, 25'b0000000000000000000000000};
        #10
        IR_instruction <= {7'b1000000, 25'b0000000000000000000000000};
        #10
        IR_instruction <= {7'b0110000, 25'b0000000000000000000000000};
        #10
        IR_instruction <= {7'b0110001, 25'b0000000000000000000000000};
        #10
        IR_instruction <= {7'b1100001, 25'b0000000000000000000000000};
        #10
        IR_instruction <= {7'b0100000, 25'b0000000000000000000000000};
        #10
        IR_instruction <= {7'b1100000, 25'b0000000000000000000000000};
        #10
        IR_instruction <= {7'b1000100, 25'b0000000000000000000000000};
        #10
        IR_instruction <= {7'b0000111, 25'b0000000000000000000000000};
    end
```

Instruction Decoder SIM

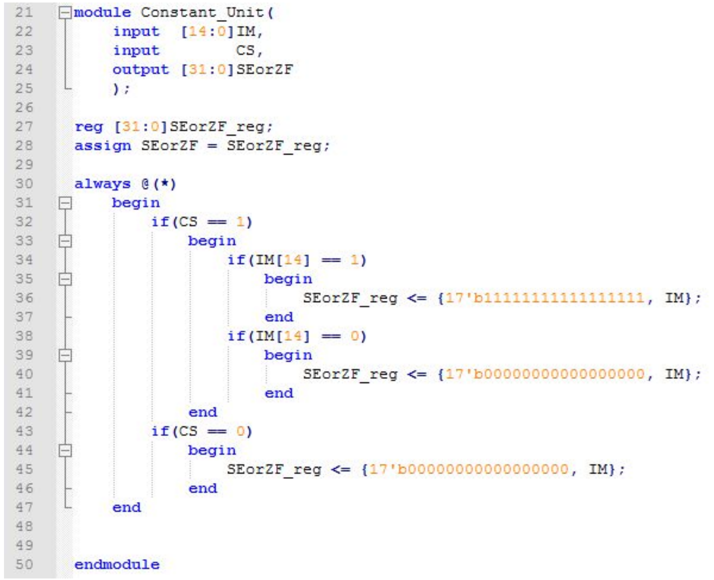


Name	Value
RW	1
DA[4:0]	00000
MD[1:0]	00
BS[1:0]	11
PS	X
MW	0
FS[4:0]	00111
MA	1
MB	1
AA[4:0]	00000
BA[4:0]	00000
CS	1
RESET	0
IR_instruction[31:0]	00001110000000000000000
NOR[6:0]	0000000

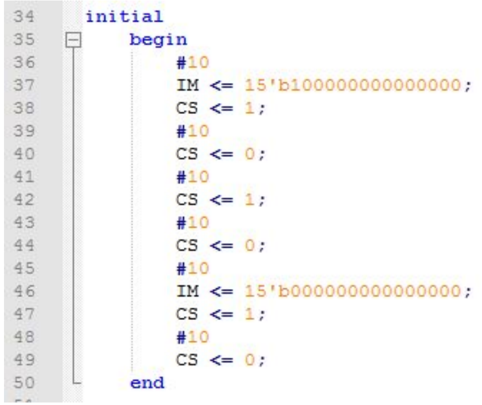
Constant unit

CS = 1 => SE

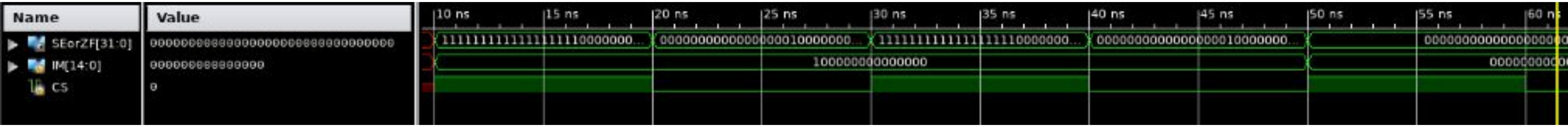
CS = 0 => ZF



```
module Constant_Unit(
    input  [14:0]IM,
    input        CS,
    output [31:0]SEorZF
    );

reg [31:0]SEorZF_reg;
assign SEorZF = SEorZF_reg;

always @(*)
    begin
        if(CS == 1)
            begin
                if(IM[14] == 1)
                    begin
                        SEorZF_reg <= {17'b11111111111111111, IM};
                    end
                if(IM[14] == 0)
                    begin
                        SEorZF_reg <= {17'b00000000000000000, IM};
                    end
            end
        if(CS == 0)
            begin
                SEorZF_reg <= {17'b00000000000000000, IM};
            end
    end


endmodule
```



```
initial
    begin
        #10
        IM <= 15'b100000000000000;
        CS <= 1;
        #10
        CS <= 0;
        #10
        CS <= 1;
        #10
        CS <= 0;
        #10
        IM <= 15'b000000000000000;
        CS <= 1;
        #10
        CS <= 0;
    end
```

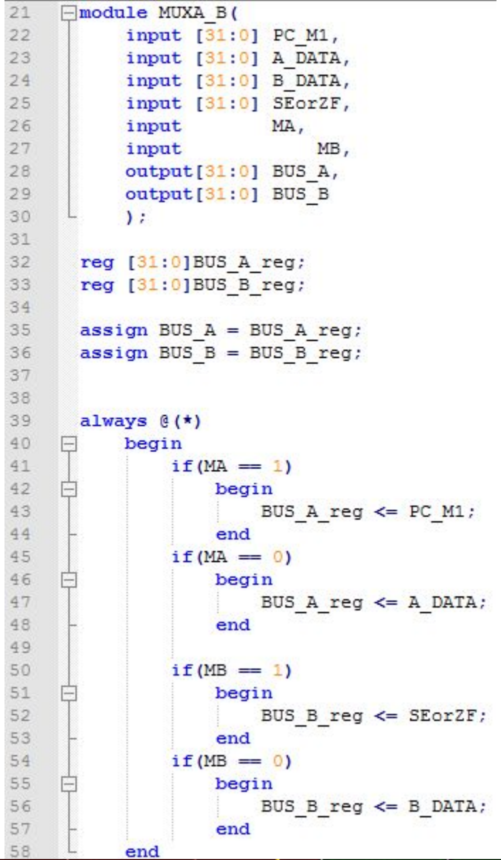


```
Name           Value
SEorZF[31:0]   00000000000000000000000000000000
IM[14:0]       000000000000000
CS             0
```

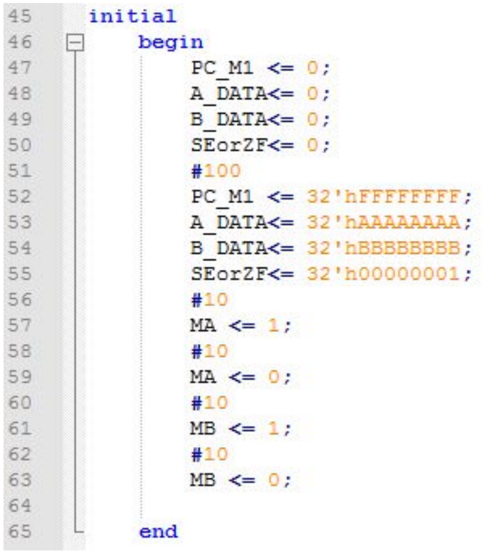
MUX A & B

MA controls BUS_A

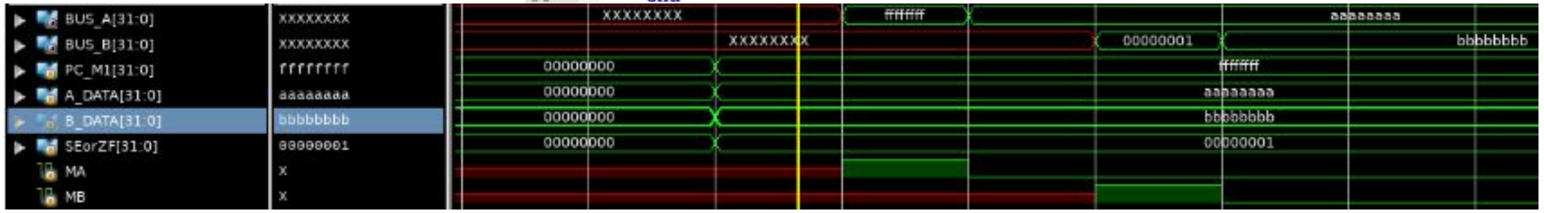
MB controls BUS_B



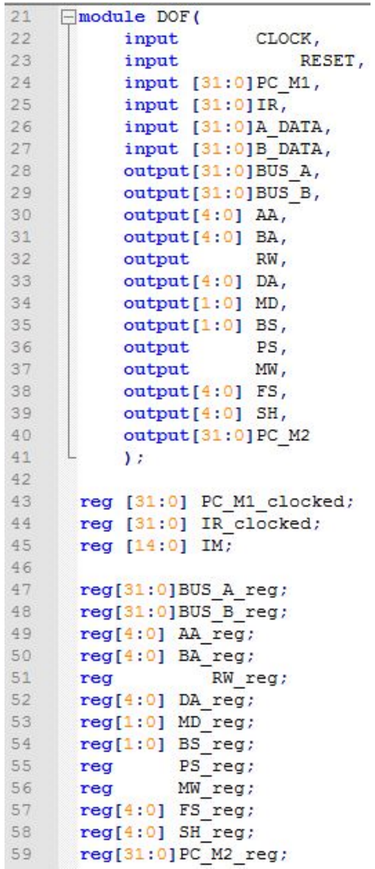
```
module MUXA_B(
    input [31:0] PC_M1,
    input [31:0] A_DATA,
    input [31:0] B_DATA,
    input [31:0] SEorZF,
    input        MA,
    input            MB,
    output[31:0] BUS_A,
    output[31:0] BUS_B
    );

reg [31:0]BUS_A_reg;
reg [31:0]BUS_B_reg;

assign BUS_A = BUS_A_reg;
assign BUS_B = BUS_B_reg;


always @(*)
    begin
        if(MA == 1)
            begin
                BUS_A_reg <= PC_M1;
            end
        if(MA == 0)
            begin
                BUS_A_reg <= A_DATA;
            end

        if(MB == 1)
            begin
                BUS_B_reg <= SEorZF;
            end
        if(MB == 0)
            begin
                BUS_B_reg <= B_DATA;
            end
    end
```



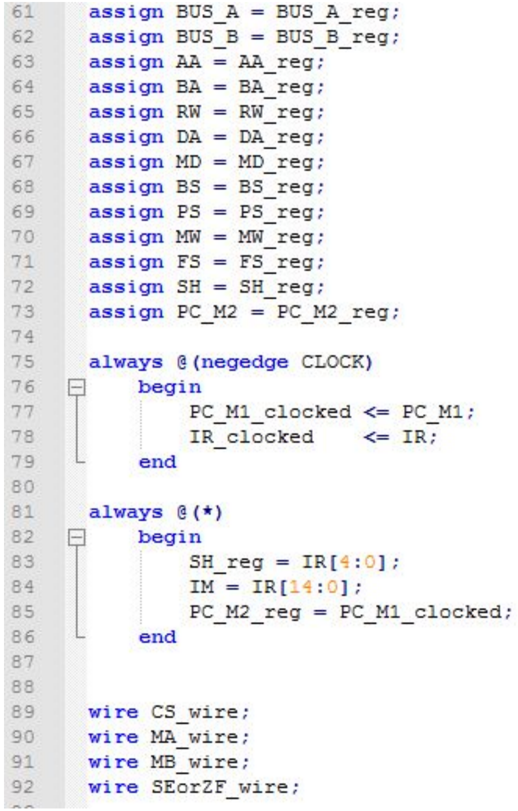
```
initial
    begin
        PC_M1  <= 0;
        A_DATA<= 0;
        B_DATA<= 0;
        SEorZF<= 0;
        #100
        PC_M1  <= 32'hFFFFFFFF;
        A_DATA<= 32'hAAAAAAAA;
        B_DATA<= 32'hBBBBBBBB;
        SEorZF<= 32'h00000001;
        #10
        MA  <= 1;
        #10
        MA  <= 0;
        #10
        MB  <= 1;
        #10
        MB  <= 0;

    end
```



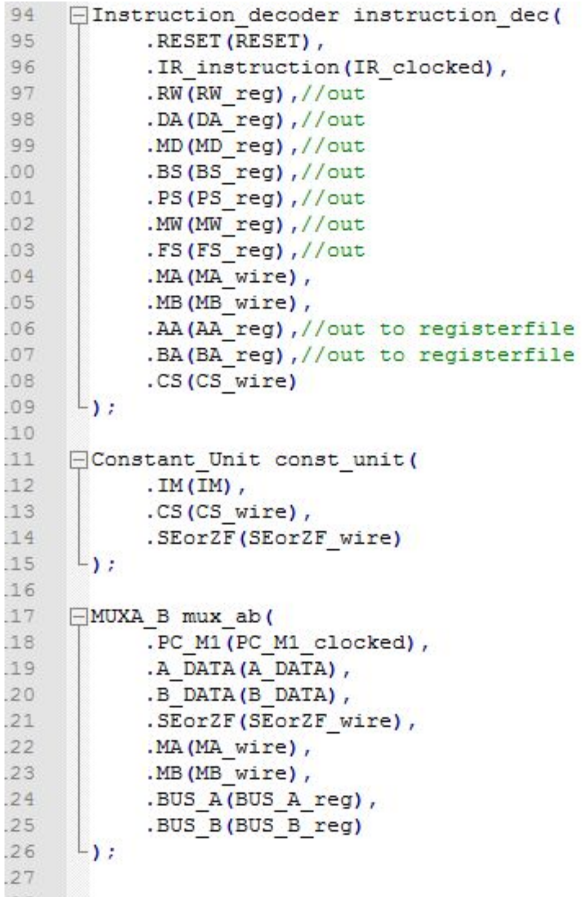
DOF module



```verilog
module DOF(
    input       CLOCK,
    input           RESET,
    input [31:0]PC_M1,
    input [31:0]IR,
    input [31:0]A_DATA,
    input [31:0]B_DATA,
    output[31:0]BUS_A,
    output[31:0]BUS_B,
    output[4:0] AA,
    output[4:0] BA,
    output      RW,
    output[4:0] DA,
    output[1:0] MD,
    output[1:0] BS,
    output      PS,
    output      MW,
    output[4:0] FS,
    output[4:0] SH,
    output[31:0]PC_M2
    );

reg [31:0] PC_M1_clocked;
reg [31:0] IR_clocked;
reg [14:0] IM;

reg[31:0]BUS_A_reg;
reg[31:0]BUS_B_reg;
reg[4:0] AA_reg;
reg[4:0] BA_reg;
reg          RW_reg;
reg[4:0] DA_reg;
reg[1:0] MD_reg;
reg[1:0] BS_reg;
reg      PS_reg;
reg      MW_reg;
reg[4:0] FS_reg;
reg[4:0] SH_reg;
reg[31:0]PC_M2_reg;
```



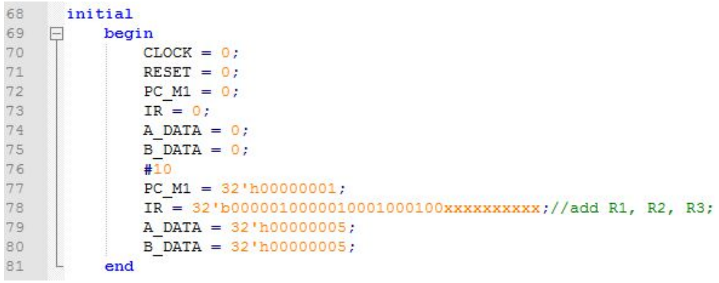
```verilog
assign BUS_A = BUS_A_reg;
assign BUS_B = BUS_B_reg;
assign AA = AA_reg;
assign BA = BA_reg;
assign RW = RW_reg;
assign DA = DA_reg;
assign MD = MD_reg;
assign BS = BS_reg;
assign PS = PS_reg;
assign MW = MW_reg;
assign FS = FS_reg;
assign SH = SH_reg;
assign PC_M2 = PC_M2_reg;

always @(negedge CLOCK)
    begin
        PC_M1_clocked <= PC_M1;
        IR_clocked    <= IR;
    end

always @(*)
    begin
        SH_reg = IR[4:0];
        IM = IR[14:0];
        PC_M2_reg = PC_M1_clocked;
    end


wire CS_wire;
wire MA_wire;
wire MB_wire;
wire SEorZF_wire;
```



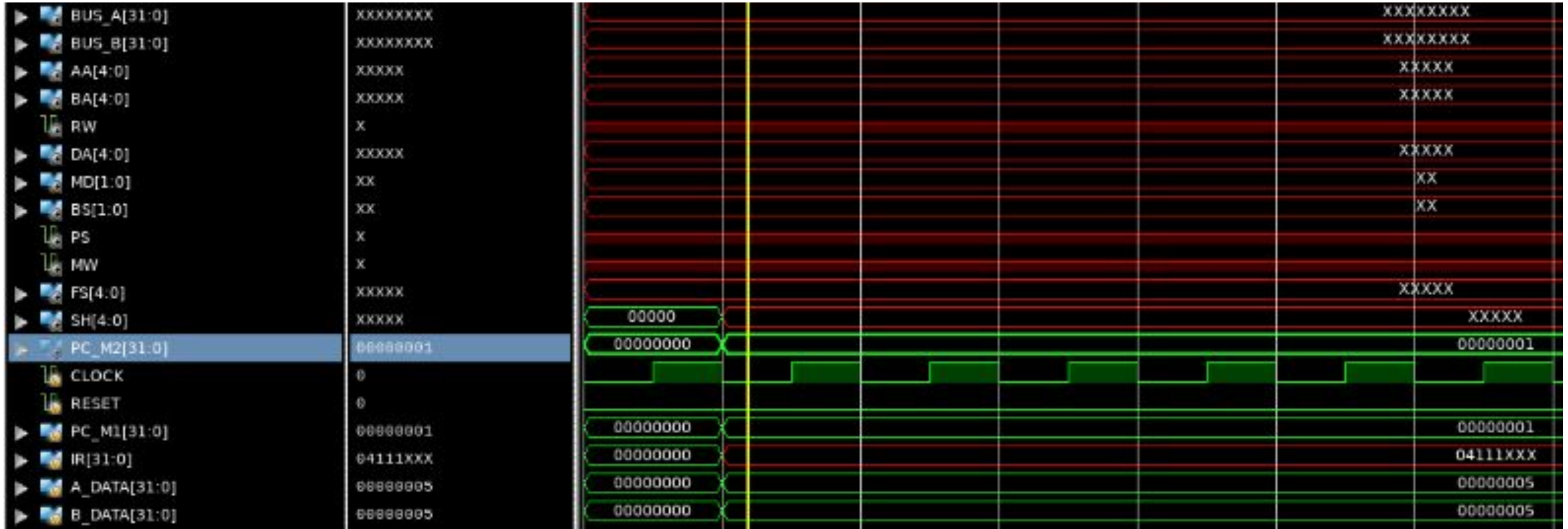
```verilog
Instruction_decoder instruction_dec(
        .RESET(RESET),
        .IR_instruction(IR_clocked),
        .RW(RW_reg),//out
        .DA(DA_reg),//out
        .MD(MD_reg),//out
        .BS(BS_reg),//out
        .PS(PS_reg),//out
        .MW(MW_reg),//out
        .FS(FS_reg),//out
        .MA(MA_wire),
        .MB(MB_wire),
        .AA(AA_reg),//out to registerfile
        .BA(BA_reg),//out to registerfile
        .CS(CS_wire)
);

Constant_Unit const_unit(
        .IM(IM),
        .CS(CS_wire),
        .SEorZF(SEorZF_wire)
);

MUXA_B mux_ab(
        .PC_M1(PC_M1_clocked),
        .A_DATA(A_DATA),
        .B_DATA(B_DATA),
        .SEorZF(SEorZF_wire),
        .MA(MA_wire),
        .MB(MB_wire),
        .BUS_A(BUS_A_reg),
        .BUS_B(BUS_B_reg)
);
```

DOF_tb

No clue what's wrong

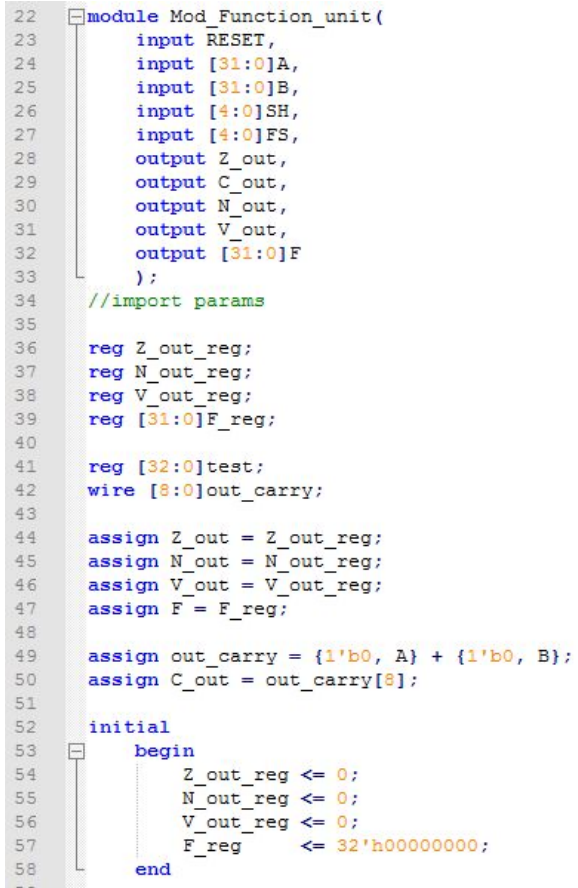


```verilog
initial
    begin
        CLOCK = 0;
        RESET = 0;
        PC_M1 = 0;
        IR = 0;
        A_DATA = 0;
        B_DATA = 0;
        #10
        PC_M1 = 32'h00000001;
        IR = 32'b0000010000010001000100xxxxxxxxxx;//add R1, R2, R3;
        A_DATA = 32'h00000005;
        B_DATA = 32'h00000005;
    end
```

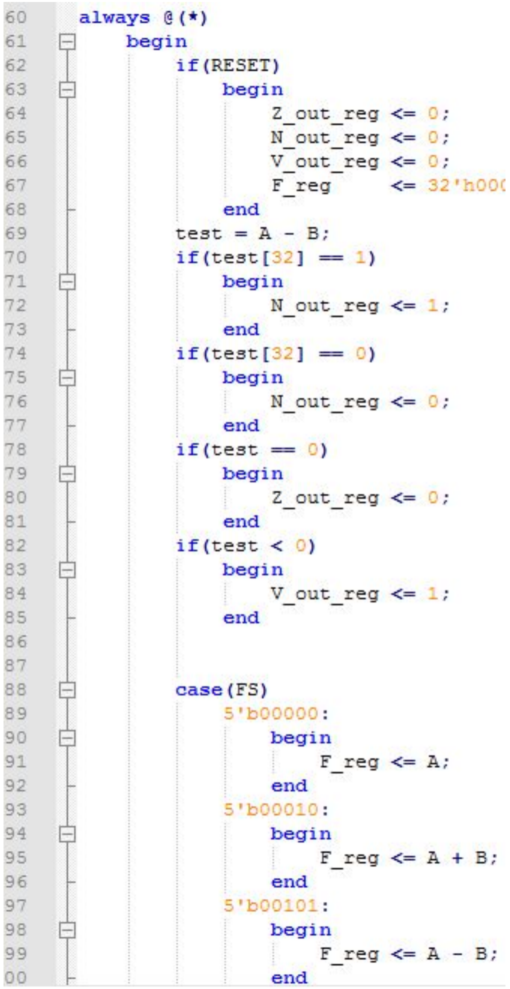


```
BUS_A[31:0]     XXXXXXXX
BUS_B[31:0]     XXXXXXXX
AA[4:0]         XXXXX
BA[4:0]         XXXXX
RW              X
DA[4:0]         XXXXX
MD[1:0]         XX
BS[1:0]         XX
PS              X
MW              X
FS[4:0]         XXXXX
SH[4:0]         XXXXX
PC_M2[31:0]     00000001
CLOCK           0
RESET           0
PC_M1[31:0]     00000001
IR[31:0]        04111XXX
A_DATA[31:0]    00000005
B_DATA[31:0]    00000005
```

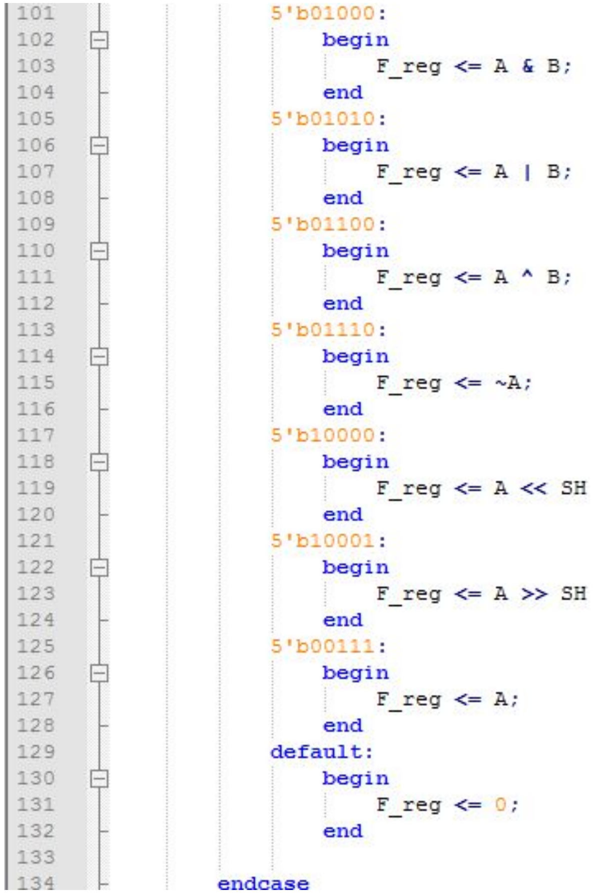
MFU



```
module Mod_Function_unit(
    input RESET,
    input [31:0]A,
    input [31:0]B,
    input [4:0]SH,
    input [4:0]FS,
    output Z_out,
    output C_out,
    output N_out,
    output V_out,
    output [31:0]F
    );
//import params

reg Z_out_reg;
reg N_out_reg;
reg V_out_reg;
reg [31:0]F_reg;

reg [32:0]test;
wire [8:0]out_carry;

assign Z_out = Z_out_reg;
assign N_out = N_out_reg;
assign V_out = V_out_reg;
assign F = F_reg;

assign out_carry = {1'b0, A} + {1'b0, B};
assign C_out = out_carry[8];

initial
    begin
        Z_out_reg <= 0;
        N_out_reg <= 0;
        V_out_reg <= 0;
        F_reg     <= 32'h00000000;
    end
```

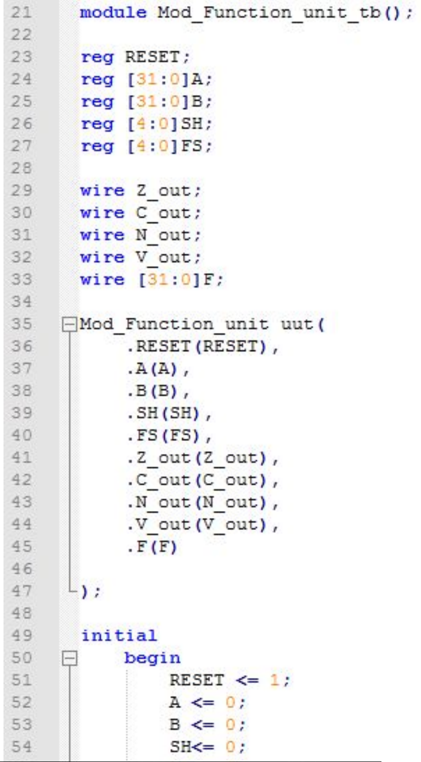


```
always @(*)
    begin
        if(RESET)
            begin
                Z_out_reg <= 0;
                N_out_reg <= 0;
                V_out_reg <= 0;
                F_reg     <= 32'h00
            end
        test = A - B;
        if(test[32] == 1)
            begin
                N_out_reg <= 1;
            end
        if(test[32] == 0)
            begin
                N_out_reg <= 0;
            end
        if(test == 0)
            begin
                Z_out_reg <= 0;
            end
        if(test < 0)
            begin
                V_out_reg <= 1;
            end


        case(FS)
            5'b00000:
                begin
                    F_reg <= A;
                end
            5'b00010:
                begin
                    F_reg <= A + B;
                end
            5'b00101:
                begin
                    F_reg <= A - B;
                end
```

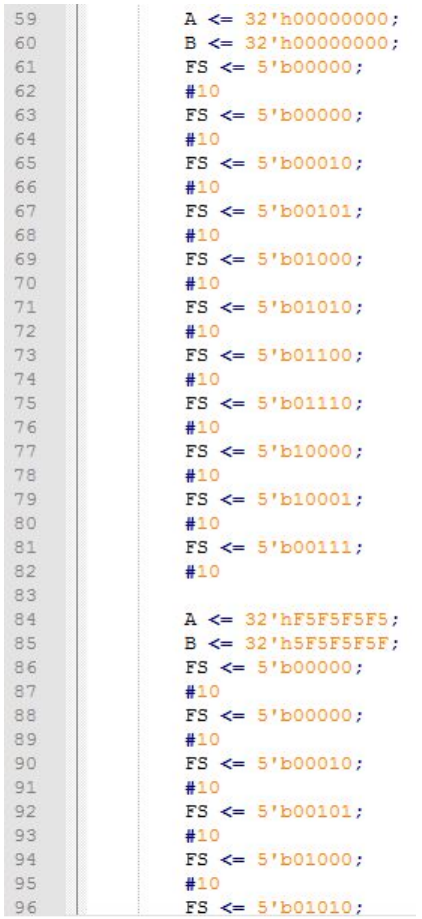


```
            5'b01000:
                begin
                    F_reg <= A & B;
                end
            5'b01010:
                begin
                    F_reg <= A | B;
                end
            5'b01100:
                begin
                    F_reg <= A ^ B;
                end
            5'b01110:
                begin
                    F_reg <= ~A;
                end
            5'b10000:
                begin
                    F_reg <= A << SH
                end
            5'b10001:
                begin
                    F_reg <= A >> SH
                end
            5'b00111:
                begin
                    F_reg <= A;
                end
            default:
                begin
                    F_reg <= 0;
                end

        endcase
```

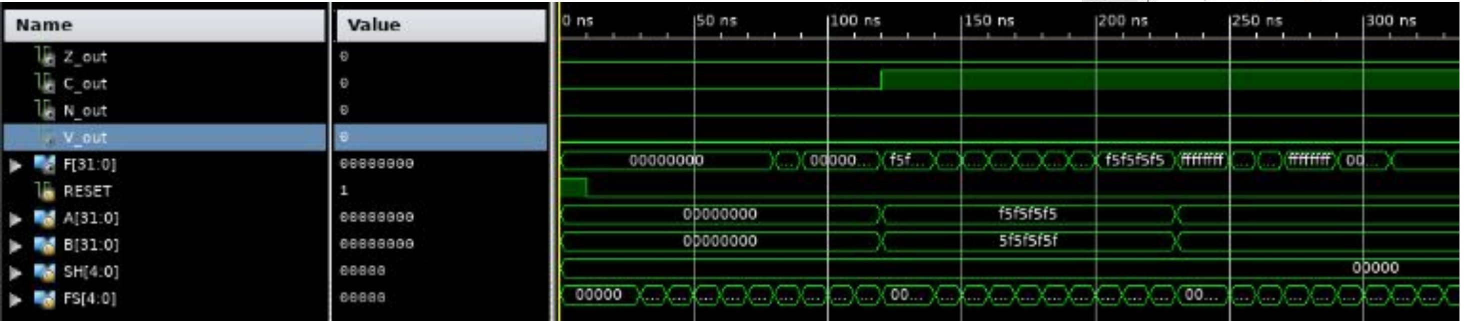
MFU_tb



```
module Mod_Function_unit_tb();

reg RESET;
reg [31:0]A;
reg [31:0]B;
reg [4:0]SH;
reg [4:0]FS;

wire Z_out;
wire C_out;
wire N_out;
wire V_out;
wire [31:0]F;

Mod_Function_unit uut(
        .RESET(RESET),
        .A(A),
        .B(B),
        .SH(SH),
        .FS(FS),
        .Z_out(Z_out),
        .C_out(C_out),
        .N_out(N_out),
        .V_out(V_out),
        .F(F)

);

initial
    begin
        RESET <= 1;
        A <= 0;
        B <= 0;
        SH<= 0;
```



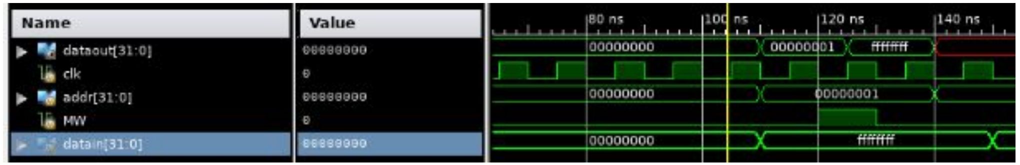
```
            A <= 32'h00000000;
            B <= 32'h00000000;
            FS <= 5'b00000;
            #10
            FS <= 5'b00000;
            #10
            FS <= 5'b00010;
            #10
            FS <= 5'b00101;
            #10
            FS <= 5'b01000;
            #10
            FS <= 5'b01010;
            #10
            FS <= 5'b01100;
            #10
            FS <= 5'b01110;
            #10
            FS <= 5'b10000;
            #10
            FS <= 5'b10001;
            #10
            FS <= 5'b00111;
            #10

            A <= 32'hF5F5F5F5;
            B <= 32'h5F5F5F5F;
            FS <= 5'b00000;
            #10
            FS <= 5'b00000;
            #10
            FS <= 5'b00010;
            #10
            FS <= 5'b00101;
            #10
            FS <= 5'b01000;
            #10
            FS <= 5'b01010;
```



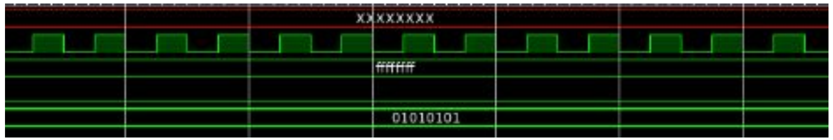
Data Memory

```
module memDATA(
      input clk,
      input [31:0] addr,
      input MW,
      input [31:0] datain,
      output [31:0] dataout
      );

reg [31:0] memword [63:0];
integer i;

initial
  begin
  for(i=0; i<64; i=i+1)
    memword[i] <= i;
  end

always @ (posedge clk)
  begin
    if(MW)
      memword[addr] <= datain;
  end

assign dataout = memword[addr];

endmodule
```

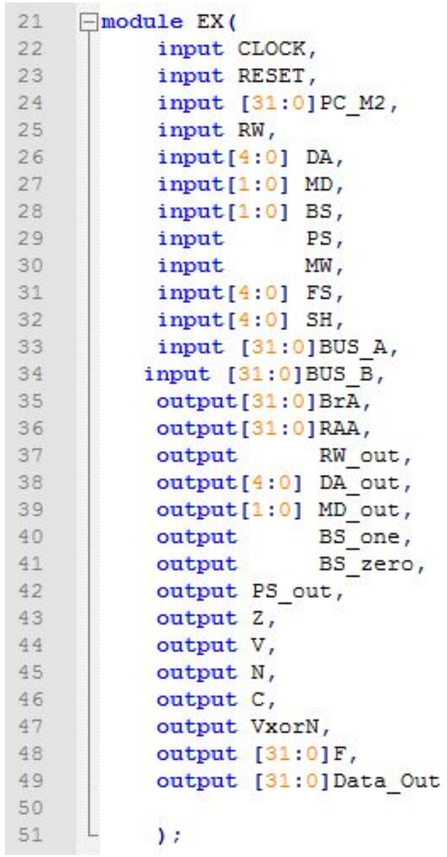
```
module memDATA_tb();

reg clk;
reg [31:0]addr;
reg MW;
reg [31:0]datain;

wire [31:0]dataout;

memDATA uut(
        .clk(clk),
        .addr(addr),
        .MW(MW),
        .datain(datain),
        .dataout(dataout)
);

initial
    begin
        clk = 0;
        addr = 32'h00000000;
        MW = 0;
        datain = 32'h00000000;
        #100
        #10
        addr = 32'h00000001;
        datain = 32'hFFFFFFFF;
        #10
        MW = 1;
        #10
        MW = 0;
        #10
        addr = 32'hFFFFFFFF;
        #10
        datain = 32'h01010101;
    end


always
    begin
        #5
        clk = ~clk;
    end
```



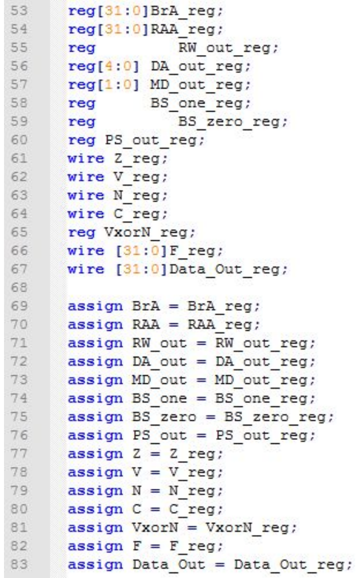
```
Name              Value
dataout[31:0]     00000000
clk               0
addr[31:0]        00000000
MW                0
datain[31:0]      00000000
```



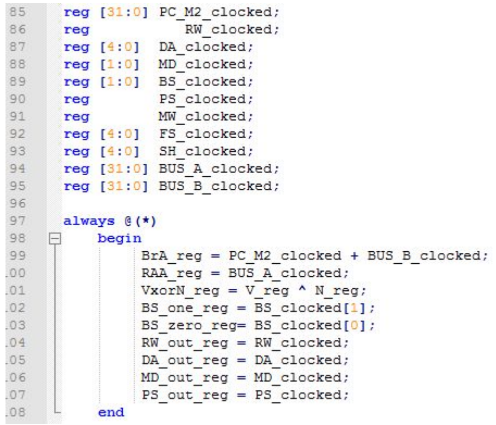
EX



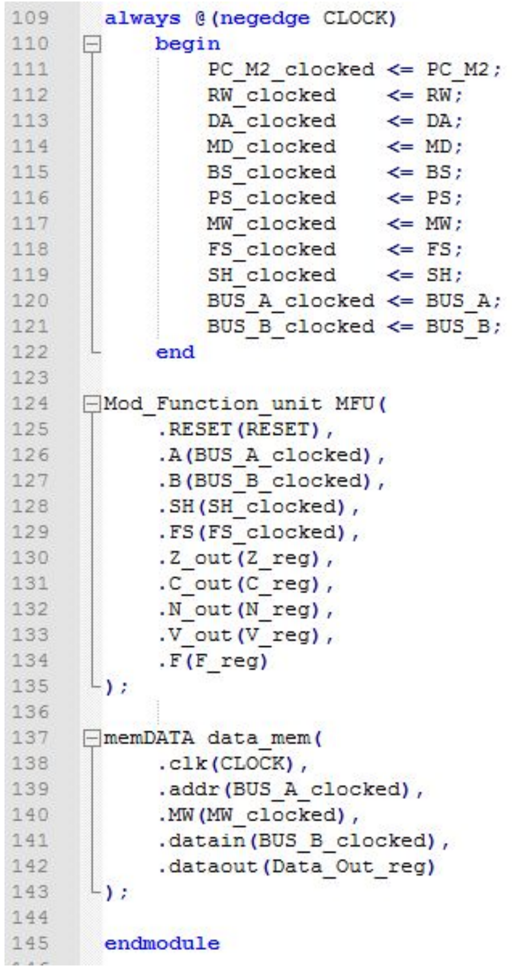
```verilog
module EX(
    input CLOCK,
    input RESET,
    input [31:0]PC_M2,
    input RW,
    input[4:0] DA,
    input[1:0] MD,
    input[1:0] BS,
    input      PS,
    input      MW,
    input[4:0] FS,
    input[4:0] SH,
    input [31:0]BUS_A,
   input [31:0]BUS_B,
    output[31:0]BrA,
    output[31:0]RAA,
    output      RW_out,
    output[4:0] DA_out,
    output[1:0] MD_out,
    output      BS_one,
    output      BS_zero,
    output PS_out,
    output Z,
    output V,
    output N,
    output C,
    output VxorN,
    output [31:0]F,
    output [31:0]Data_Out

    );
```



```verilog
reg[31:0]BrA_reg;
reg[31:0]RAA_reg;
reg         RW_out_reg;
reg[4:0] DA_out_reg;
reg[1:0] MD_out_reg;
reg      BS_one_reg;
reg         BS_zero_reg;
reg PS_out_reg;
wire Z_reg;
wire V_reg;
wire N_reg;
wire C_reg;
reg VxorN_reg;
wire [31:0]F_reg;
wire [31:0]Data_Out_reg;

assign BrA = BrA_reg;
assign RAA = RAA_reg;
assign RW_out = RW_out_reg;
assign DA_out = DA_out_reg;
assign MD_out = MD_out_reg;
assign BS_one = BS_one_reg;
assign BS_zero = BS_zero_reg;
assign PS_out = PS_out_reg;
assign Z = Z_reg;
assign V = V_reg;
assign N = N_reg;
assign C = C_reg;
assign VxorN = VxorN_reg;
assign F = F_reg;
assign Data_Out = Data_Out_reg;
```

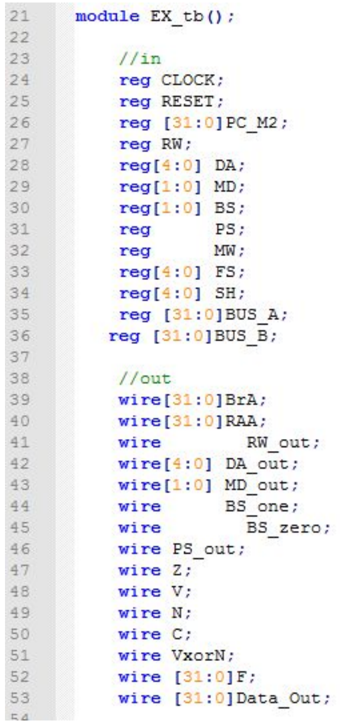


```verilog
reg [31:0] PC_M2_clocked;
reg            RW_clocked;
reg [4:0]  DA_clocked;
reg [1:0]  MD_clocked;
reg [1:0]  BS_clocked;
reg        PS_clocked;
reg        MW_clocked;
reg [4:0]  FS_clocked;
reg [4:0]  SH_clocked;
reg [31:0] BUS_A_clocked;
reg [31:0] BUS_B_clocked;

always @(*)
    begin
        BrA_reg = PC_M2_clocked + BUS_B_clocked;
        RAA_reg = BUS_A_clocked;
        VxorN_reg = V_reg ^ N_reg;
        BS_one_reg = BS_clocked[1];
        BS_zero_reg= BS_clocked[0];
        RW_out_reg = RW_clocked;
        DA_out_reg = DA_clocked;
        MD_out_reg = MD_clocked;
        PS_out_reg = PS_clocked;
    end
```

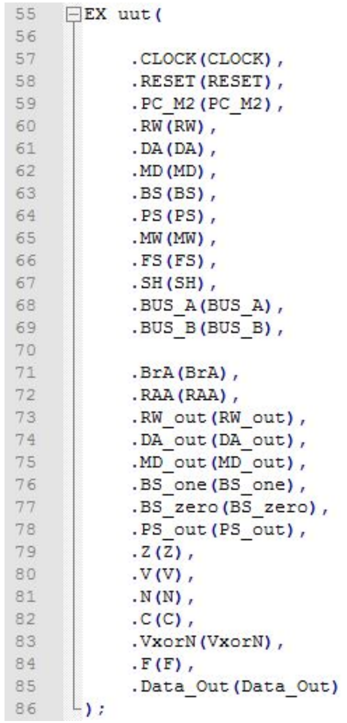


```verilog
always @(negedge CLOCK)
    begin
        PC_M2_clocked <= PC_M2;
        RW_clocked    <= RW;
        DA_clocked    <= DA;
        MD_clocked    <= MD;
        BS_clocked    <= BS;
        PS_clocked    <= PS;
        MW_clocked    <= MW;
        FS_clocked    <= FS;
        SH_clocked    <= SH;
        BUS_A_clocked <= BUS_A;
        BUS_B_clocked <= BUS_B;
    end

Mod_Function_unit MFU(
        .RESET(RESET),
        .A(BUS_A_clocked),
        .B(BUS_B_clocked),
        .SH(SH_clocked),
        .FS(FS_clocked),
        .Z_out(Z_reg),
        .C_out(C_reg),
        .N_out(N_reg),
        .V_out(V_reg),
        .F(F_reg)
);

memDATA data_mem(
        .clk(CLOCK),
        .addr(BUS_A_clocked),
        .MW(MW_clocked),
        .datain(BUS_B_clocked),
        .dataout(Data_Out_reg)
);

endmodule
```

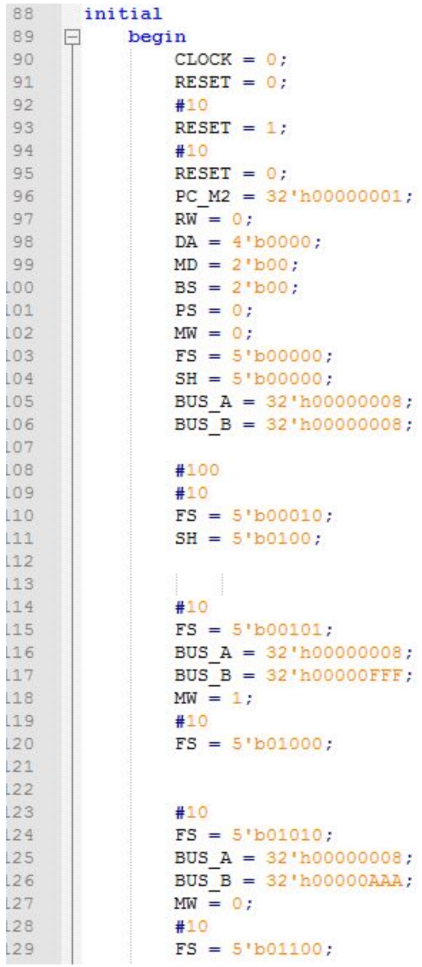
EX_tb



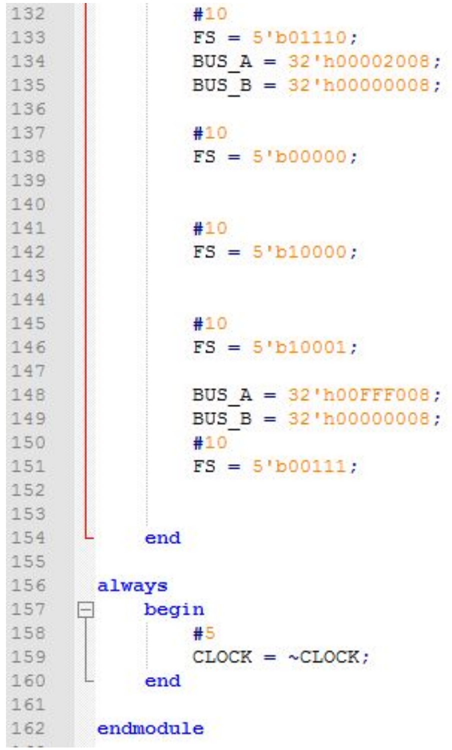
```
module EX_tb();

    //in
    reg CLOCK;
    reg RESET;
    reg [31:0]PC_M2;
    reg RW;
    reg[4:0] DA;
    reg[1:0] MD;
    reg[1:0] BS;
    reg      PS;
    reg      MW;
    reg[4:0] FS;
    reg[4:0] SH;
    reg [31:0]BUS_A;
   reg [31:0]BUS_B;

    //out
    wire[31:0]BrA;
    wire[31:0]RAA;
    wire          RW_out;
    wire[4:0] DA_out;
    wire[1:0] MD_out;
    wire      BS_one;
    wire          BS_zero;
    wire PS_out;
    wire Z;
    wire V;
    wire N;
    wire C;
    wire VxorN;
    wire [31:0]F;
    wire [31:0]Data_Out;
```



```
EX uut(

        .CLOCK(CLOCK),
        .RESET(RESET),
        .PC_M2(PC_M2),
        .RW(RW),
        .DA(DA),
        .MD(MD),
        .BS(BS),
        .PS(PS),
        .MW(MW),
        .FS(FS),
        .SH(SH),
        .BUS_A(BUS_A),
        .BUS_B(BUS_B),

        .BrA(BrA),
        .RAA(RAA),
        .RW_out(RW_out),
        .DA_out(DA_out),
        .MD_out(MD_out),
        .BS_one(BS_one),
        .BS_zero(BS_zero),
        .PS_out(PS_out),
        .Z(Z),
        .V(V),
        .N(N),
        .C(C),
        .VxorN(VxorN),
        .F(F),
        .Data_Out(Data_Out)
);
```

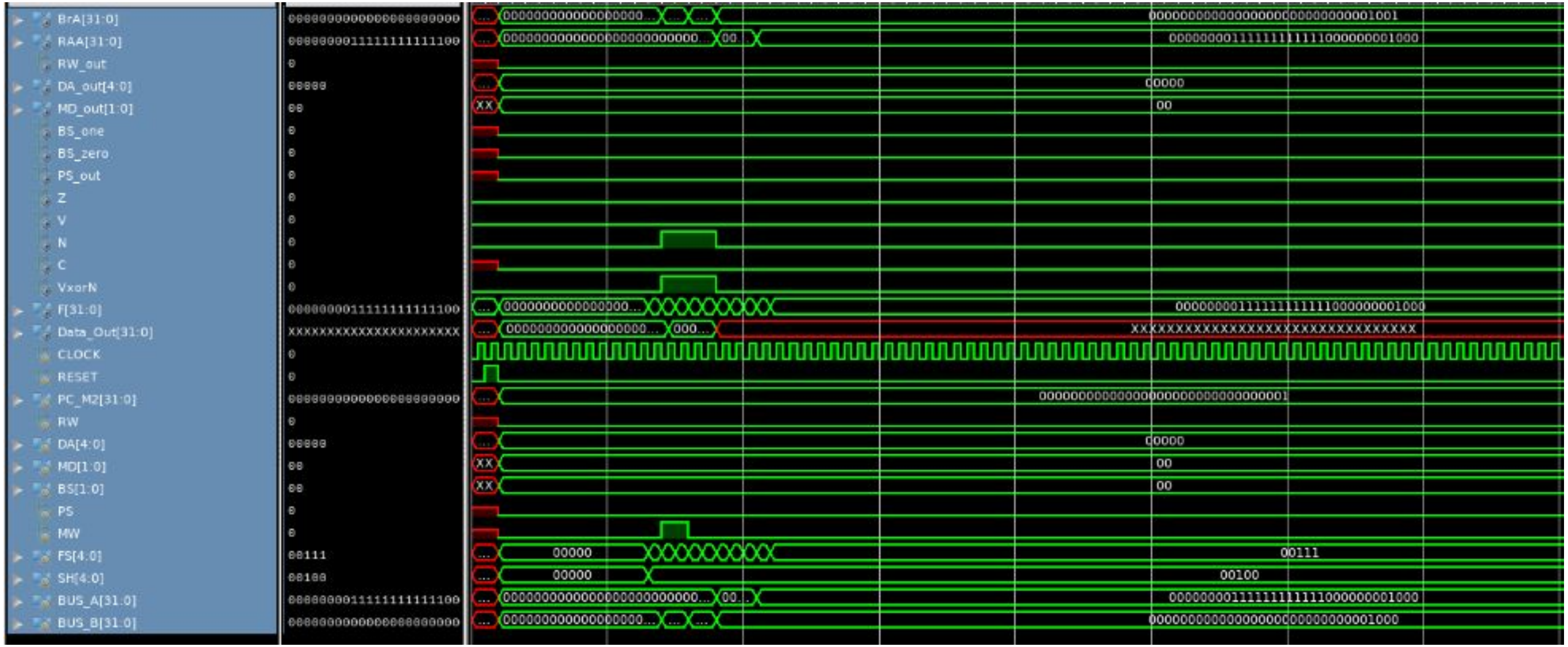


```
initial
    begin
        CLOCK = 0;
        RESET = 0;
        #10
        RESET = 1;
        #10
        RESET = 0;
        PC_M2 = 32'h00000001;
        RW = 0;
        DA = 4'b0000;
        MD = 2'b00;
        BS = 2'b00;
        PS = 0;
        MW = 0;
        FS = 5'b00000;
        SH = 5'b00000;
        BUS_A = 32'h00000008;
        BUS_B = 32'h00000008;

        #100
        #10
        FS = 5'b00010;
        SH = 5'b0100;


        #10
        FS = 5'b00101;
        BUS_A = 32'h00000008;
        BUS_B = 32'h00000FFF;
        MW = 1;
        #10
        FS = 5'b01000;


        #10
        FS = 5'b01010;
        BUS_A = 32'h00000008;
        BUS_B = 32'h00000AAA;
        MW = 0;
        #10
        FS = 5'b01100;
```



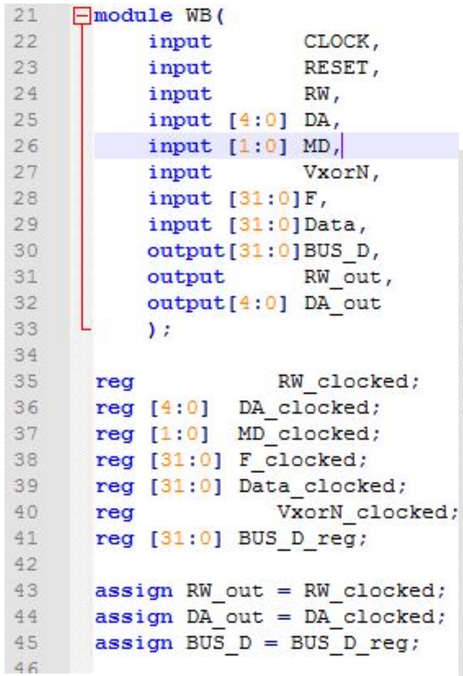
```
        #10
        FS = 5'b01110;
        BUS_A = 32'h00002008;
        BUS_B = 32'h00000008;

        #10
        FS = 5'b00000;


        #10
        FS = 5'b10000;


        #10
        FS = 5'b10001;

        BUS_A = 32'h00FFF008;
        BUS_B = 32'h00000008;
        #10
        FS = 5'b00111;



    end

always
    begin
        #5
        CLOCK = ~CLOCK;
    end

endmodule
```

EX_tb SIM

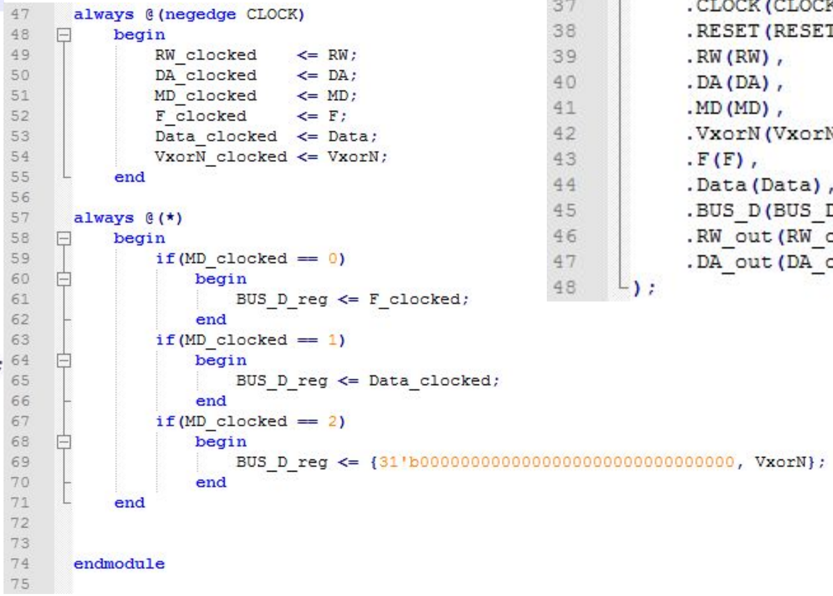


```
BrA[31:0]        0000000000000000000000      00000000000000000000000000001001
RAA[31:0]        0000000011111111111100      00000000111111111111000000001000
RW_out           0
DA_out[4:0]      00000                       00000
MD_out[1:0]      00                          00
BS_one           0
BS_zero          0
PS_out           0
Z                0
V                0
N                0
C                0
VxorN            0
F[31:0]          0000000011111111111100      00000000111111111111000000001000
Data_Out[31:0]   XXXXXXXXXXXXXXXXXXXXXX      XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CLOCK            0
RESET            0
PC_M2[31:0]      0000000000000000000000      00000000000000000000000000000001
RW               0
DA[4:0]          00000                       00000
MD[1:0]          00                          00
BS[1:0]          00                          00
PS               0
MW               0
FS[4:0]          00111                       00111
SH[4:0]          00100                       00100
BUS_A[31:0]      0000000011111111111100      00000000111111111111000000001000
BUS_B[31:0]      0000000000000000000000      00000000000000000000000000001000
```

WB

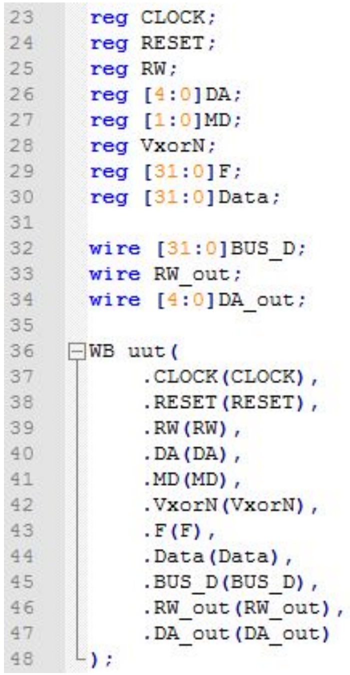


```verilog
module WB(
    input        CLOCK,
    input        RESET,
    input        RW,
    input [4:0]  DA,
    input [1:0]  MD,
    input        VxorN,
    input [31:0]F,
    input [31:0]Data,
    output[31:0]BUS_D,
    output       RW_out,
    output[4:0]  DA_out
    );

reg            RW_clocked;
reg [4:0]   DA_clocked;
reg [1:0]   MD_clocked;
reg [31:0]  F_clocked;
reg [31:0]  Data_clocked;
reg            VxorN_clocked;
reg [31:0]  BUS_D_reg;

assign RW_out = RW_clocked;
assign DA_out = DA_clocked;
assign BUS_D = BUS_D_reg;
```

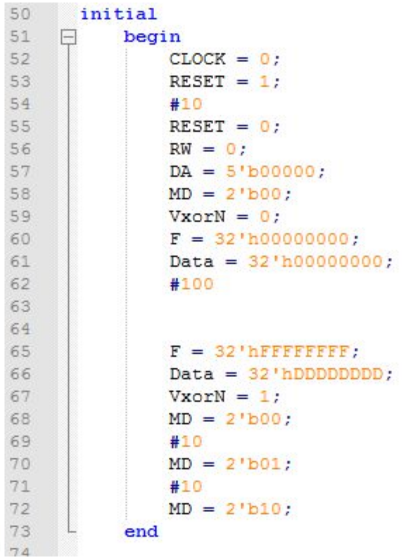


```verilog
always @(negedge CLOCK)
    begin
        RW_clocked    <= RW;
        DA_clocked    <= DA;
        MD_clocked    <= MD;
        F_clocked     <= F;
        Data_clocked  <= Data;
        VxorN_clocked <= VxorN;
    end

always @(*)
    begin
        if(MD_clocked == 0)
            begin
                BUS_D_reg <= F_clocked;
            end
        if(MD_clocked == 1)
            begin
                BUS_D_reg <= Data_clocked;
            end
        if(MD_clocked == 2)
            begin
                BUS_D_reg <= {31'b0000000000000000000000000000000, VxorN};
            end
    end


endmodule
```

TestBench

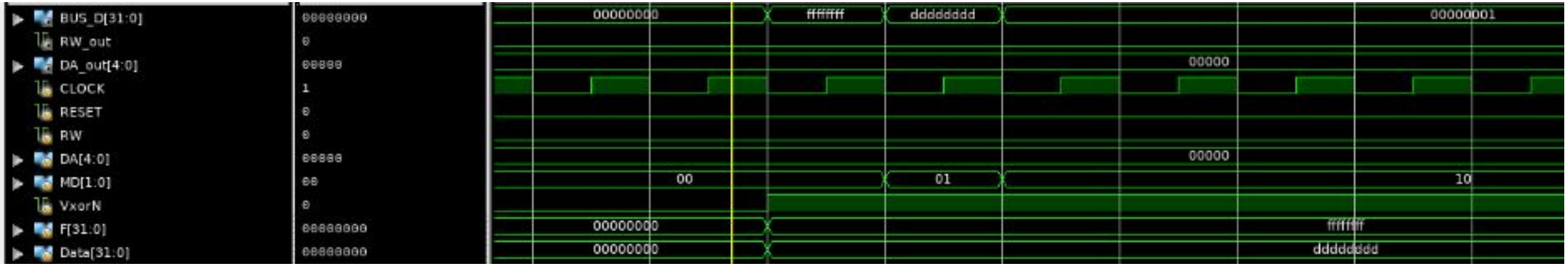


```verilog
reg CLOCK;
reg RESET;
reg RW;
reg [4:0]DA;
reg [1:0]MD;
reg VxorN;
reg [31:0]F;
reg [31:0]Data;

wire [31:0]BUS_D;
wire RW_out;
wire [4:0]DA_out;

WB uut(
        .CLOCK(CLOCK),
        .RESET(RESET),
        .RW(RW),
        .DA(DA),
        .MD(MD),
        .VxorN(VxorN),
        .F(F),
        .Data(Data),
        .BUS_D(BUS_D),
        .RW_out(RW_out),
        .DA_out(DA_out)
);
```



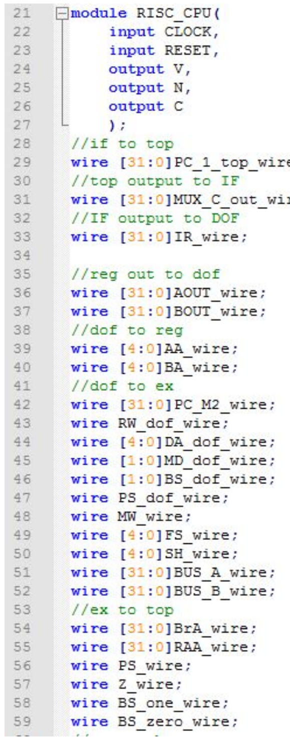
```verilog
initial
    begin
        CLOCK = 0;
        RESET = 1;
        #10
        RESET = 0;
        RW = 0;
        DA = 5'b00000;
        MD = 2'b00;
        VxorN = 0;
        F = 32'h00000000;
        Data = 32'h00000000;
        #100


        F = 32'hFFFFFFFF;
        Data = 32'hDDDDDDDD;
        VxorN = 1;
        MD = 2'b00;
        #10
        MD = 2'b01;
        #10
        MD = 2'b10;
    end
```

WB SIM

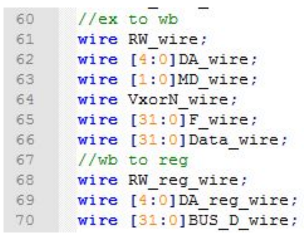


```
BUS_D[31:0]   00000000
RW_out        0
DA_out[4:0]   00000
CLOCK         1
RESET         0
RW            0
DA[4:0]       00000
MD[1:0]       00
VxorN         0
F[31:0]       00000000
Data[31:0]    00000000
```

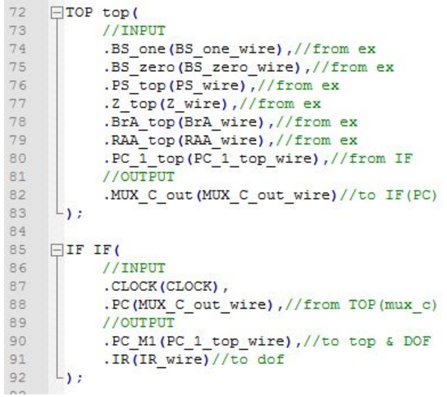
RISC_CPU



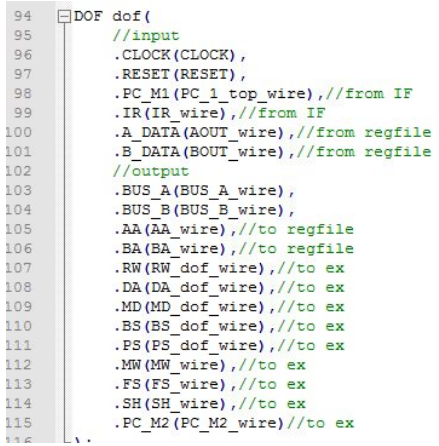
```
module RISC_CPU(
        input CLOCK,
        input RESET,
        output V,
        output N,
        output C
        );
    //if to top
    wire [31:0]PC_1_top_wire;
    //top output to IF
    wire [31:0]MUX_C_out_wire;
    //IF output to DOF
    wire [31:0]IR_wire;

    //reg out to dof
    wire [31:0]AOUT_wire;
    wire [31:0]BOUT_wire;
    //dof to reg
    wire [4:0]AA_wire;
    wire [4:0]BA_wire;
    //dof to ex
    wire [31:0]PC_M2_wire;
    wire RW_dof_wire;
    wire [4:0]DA_dof_wire;
    wire [1:0]MD_dof_wire;
    wire [1:0]BS_dof_wire;
    wire PS_dof_wire;
    wire MW_wire;
    wire [4:0]FS_wire;
    wire [4:0]SH_wire;
    wire [31:0]BUS_A_wire;
    wire [31:0]BUS_B_wire;
    //ex to top
    wire [31:0]BrA_wire;
    wire [31:0]RAA_wire;
    wire PS_wire;
    wire Z_wire;
    wire BS_one_wire;
    wire BS_zero_wire;
```



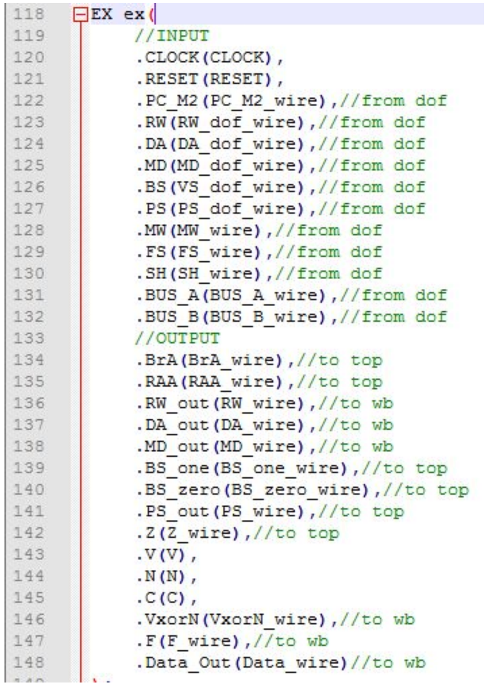
```
    //ex to wb
    wire RW_wire;
    wire [4:0]DA_wire;
    wire [1:0]MD_wire;
    wire VxorN_wire;
    wire [31:0]F_wire;
    wire [31:0]Data_wire;
    //wb to reg
    wire RW_reg_wire;
    wire [4:0]DA_reg_wire;
    wire [31:0]BUS_D_wire;
```



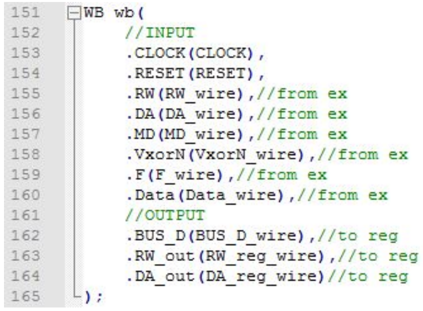
```
TOP top(
    //INPUT
    .BS_one(BS_one_wire),//from ex
    .BS_zero(BS_zero_wire),//from ex
    .PS_top(PS_wire),//from ex
    .Z_top(Z_wire),//from ex
    .BrA_top(BrA_wire),//from ex
    .RAA_top(RAA_wire),//from ex
    .PC_1_top(PC_1_top_wire),//from IF
    //OUTPUT
    .MUX_C_out(MUX_C_out_wire)//to IF(PC)
);

IF IF(
    //INPUT
    .CLOCK(CLOCK),
    .PC(MUX_C_out_wire),//from TOP(mux_c)
    //OUTPUT
    .PC_M1(PC_1_top_wire),//to top & DOF
    .IR(IR_wire)//to dof
);
```



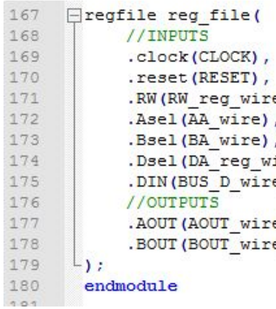
```
DOF dof(
    //input
    .CLOCK(CLOCK),
    .RESET(RESET),
    .PC_M1(PC_1_top_wire),//from IF
    .IR(IR_wire),//from IF
    .A_DATA(AOUT_wire),//from regfile
    .B_DATA(BOUT_wire),//from regfile
    //output
    .BUS_A(BUS_A_wire),
    .BUS_B(BUS_B_wire),
    .AA(AA_wire),//to regfile
    .BA(BA_wire),//to regfile
    .RW(RW_dof_wire),//to ex
    .DA(DA_dof_wire),//to ex
    .MD(MD_dof_wire),//to ex
    .BS(BS_dof_wire),//to ex
    .PS(PS_dof_wire),//to ex
    .MW(MW_wire),//to ex
    .FS(FS_wire),//to ex
    .SH(SH_wire),//to ex
    .PC_M2(PC_M2_wire)//to ex
```



```
EX ex(
    //INPUT
    .CLOCK(CLOCK),
    .RESET(RESET),
    .PC_M2(PC_M2_wire),//from dof
    .RW(RW_dof_wire),//from dof
    .DA(DA_dof_wire),//from dof
    .MD(MD_dof_wire),//from dof
    .BS(VS_dof_wire),//from dof
    .PS(PS_dof_wire),//from dof
    .MW(MW_wire),//from dof
    .FS(FS_wire),//from dof
    .SH(SH_wire),//from dof
    .BUS_A(BUS_A_wire),//from dof
    .BUS_B(BUS_B_wire),//from dof
    //OUTPUT
    .BrA(BrA_wire),//to top
    .RAA(RAA_wire),//to top
    .RW_out(RW_wire),//to wb
    .DA_out(DA_wire),//to wb
    .MD_out(MD_wire),//to wb
    .BS_one(BS_one_wire),//to top
    .BS_zero(BS_zero_wire),//to top
    .PS_out(PS_wire),//to top
    .Z(Z_wire),//to top
    .V(V),
    .N(N),
    .C(C),
    .VxorN(VxorN_wire),//to wb
    .F(F_wire),//to wb
    .Data_Out(Data_wire)//to wb
```



```
WB wb(
    //INPUT
    .CLOCK(CLOCK),
    .RESET(RESET),
    .RW(RW_wire),//from ex
    .DA(DA_wire),//from ex
    .MD(MD_wire),//from ex
    .VxorN(VxorN_wire),//from ex
    .F(F_wire),//from ex
    .Data(Data_wire),//from ex
    //OUTPUT
    .BUS_D(BUS_D_wire),//to reg
    .RW_out(RW_reg_wire),//to reg
    .DA_out(DA_reg_wire)//to reg
);
```



```
regfile reg_file(
    //INPUTS
    .clock(CLOCK),
    .reset(RESET),
    .RW(RW_reg_wire),//from wb
    .Asel(AA_wire),//from dof
    .Bsel(BA_wire),//from dof
    .Dsel(DA_reg_wire),//from wb
    .DIN(BUS_D_wire),//from wb
    //OUTPUTS
    .AOUT(AOUT_wire),//to dof
    .BOUT(BOUT_wire)//to dof
);
endmodule
```