FINAL LAB

DIGITAL SYSTEMS ES204

Guntas Singh Saran
guntassingh.saran@iitgn.ac.in
22110089

Nihar Shah
nihar.shah@iitgn.ac.in
22110237

PROCESSOR

GIT REPO

STRUCTURE OF THE MODULE

1. **PROGRAM** MEMORY is a **16x8** array to represent at max **16 instructions of 8 bits each.**

2. The Program Counter **PC** is a **4-bit** register to store the address of the current instruction being executed

3. The Instruction Register **IR** is an **8-bit** register to store the current instruction being executed

4. The **REGISTER** File is a **16x8** array to store the values of **16 registers of 8 bits each**

5. At each clock cycle, the instruction pointed to by the **PC** is fetched from the **PROGRAM** MEMORY stored in **IR**, and executed, and

the PC is incremented by 1

6. All ALU operations are performed on the **ACC** register

7. **RSTN** is the active-low reset signal.

8. **PAUSE** is a control signal to pause the processor, i.e, the **PC** will not get incremented on clock posedge.

9. Overall, there is only one output register **8-bit** named **OUTPUT** since we can select to display any registers in the entire code. Hence, we have a **MODE** as a **5-bit** select line.

10. The user may **PAUSE** the program execution, and at that time, while the clock is running, no register values change; hence, we may select any **MODE** line to display any register value.

11. a separate module has been created to implement a synthesizable **DIVISION**.

12. The instructions are of the format:

1. 0000 0000 : **NOP**

2. 0001 xxxx : **ADD Ri**
3. 0010 xxxx : **SUB Ri**
4. 0011 xxxx : **MUL Ri**
5. 0100 xxxx : **DIV Ri**

6. 0000 0001 : **LSL ACC** (Logical Shift Left the contents of ACC. Does not update CB)
7. 0000 0010 : **LSR ACC** (Logical Shift Right the contents of ACC. Does not update CB)
8. 0000 0011 : **CIR ACC** (Circular Shift Right the contents of ACC. Does not update CB)
9. 0000 0100 : **CIL ACC** (Circular Shift Left the contents of ACC. Does not update CB)
10. 0000 0101 : **ASR ACC** (Arithmetic Shift Right the contents of ACC. Does not update CB)

11. 0101 xxxx : **AND Ri**
12. 0110 xxxx : **XOR Ri**
13. 0111 xxxx : **CMP Ri** (Compare ACC with Ri. If ACC $\geqslant$ Ri, CB = 0, else CB = 1)

14. 0000 0110 : **INC ACC** (Increment ACC by 1. Updates CB if overflow)
15. 0000 0111 : **DEC ACC** (Decrement ACC by 1. Updates CB if underflow)

16. 1000 xxxx : **BR <4-bit address>** (PC is updated, and the program branches to 4-bit address if CB = 1)

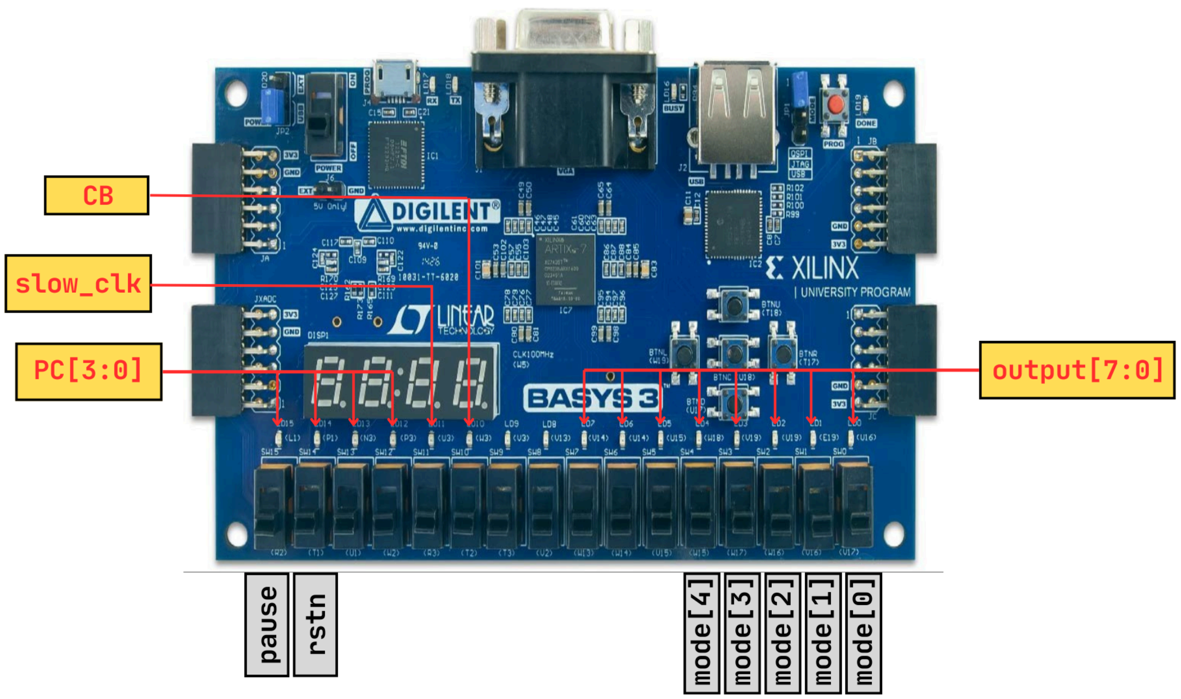
17. 1001 xxxx : **MOV Ri** (Move the contents of Ri to ACC)

18. 1010 xxxx : **MOV ACC Ri** (Move the contents of ACC to Ri)

19. 1011 xxxx : **RET <4-bit address>** (PC is updated, and program returns to the called program)

20. 1111 1111 : **HLT** (Halt the program)

FPGA IMPLEMENTATION



FPGA IMPLEMENTABLE CODE

```
`timescale 1ns / 1ps

module ClockDivide(input main_clk,
output slow_clk);
reg [31:0] counter;
always@(posedge main_clk)
begin
counter = counter + 1;
end
assign slow_clk = counter[27];
endmodule


module Processor(
   clk, rstn, pause,mode,
   CB,
   Output
   ,slow_clk,PC
);
output wire slow_clk;
input [4:0] mode;
input clk, rstn, pause;
reg [7:0] PROGRAM [15:0];
reg [7:0] RegFile [15:0];
reg [7:0] ACC;
reg [7:0] EXT;
output reg [7:0] Output;
output reg CB;
reg [8:0] SUMDIFF;
reg [15:0] MULTDIV;
wire [7:0] Div;
reg [7:0] IR;
output reg [3:0] PC;
wire [7:0] Rem;
integer i;
ClockDivide obj(.main_clk(clk), .slow_clk(slow_clk));
division obj1(.A(ACC), .B(RegFile[IR[3:0]]), .Res(Div), .Rem(Rem));


always @(posedge slow_clk or negedge rstn) begin
   if(!rstn)begin

       RegFile[0] <= 8'd0;
       RegFile[1] <= 8'd1;
       RegFile[2] <= 8'd2;
```

```
        RegFile[3] <= 8'd3;
        RegFile[4] <= 8'd4;
        RegFile[5] <= 8'd5;
        RegFile[6] <= 8'd6;
        RegFile[7] <= 8'd7;
        RegFile[8] <= 8'd8;
        RegFile[9] <= 8'd9;
        RegFile[10] <= 8'd10;
        RegFile[11] <= 8'd11;
        RegFile[12] <= 8'd12;
        RegFile[13] <= 8'd13;
        RegFile[14] <= 8'd14;
        RegFile[15] <= 8'd15;

        PROGRAM[0] <= 8'b10010011;    // 1.  MOV ACC R3   ACC <- 3
        PROGRAM[1] <= 8'b01100011;    // 2.  XOR R3       ACC <- 0
        PROGRAM[2] <= 8'b00010101;    // 3.  ADD R5       ACC <- 5
        PROGRAM[3] <= 8'b00010110;    // 4.  ADD R6       ACC <- 11
        PROGRAM[4] <= 8'b10100111;    // 5.  MOV R7 ACC   R7 <- 11
        PROGRAM[5] <= 8'b00000110;    // 6.  INC ACC ACC = 00001100
        PROGRAM[6] <= 8'b00000111;    // 7.  DEC ACC ACC = 00001011
        PROGRAM[7] <= 8'b00101010;    // 8.  SUB R10 ACC = 00000001
        PROGRAM[8] <= 8'b00110100;    // 9.  MUL R4 ACC = 00000100
        PROGRAM[9] <= 8'b01100101;    // 10. XOR R5 ACC = 00000001
        PROGRAM[10] <= 8'b00110100;   // 11. MUL R4 ACC = 00000100
        PROGRAM[11] <= 8'b01000100;   // 12. DIV R4 ACC = 00000001
        PROGRAM[12] <= 8'b01110101;   // 13. CMP R5 CB = 1
        PROGRAM[13] <= 8'b10000101;   // 14. CB =1 branch to PC = 5
        PROGRAM[14] <= 8'b11111111;   // 15. HLT

        PC <= 4'b0;
        IR <= 8'b0;
        ACC <= 8'b11111111;
        EXT <= 8'b0;
        CB <= 1'b0;
        SUMDIFF <= 9'b0;
        MULTDIV <= 16'b0;
        Output <= 8'b0;
    end

    else begin
        if(mode == 5'b00001)begin
            Output <= RegFile[0];
        end
        else if(mode == 5'b11111)begin
            Output <= ACC;
        end
```

```verilog
else if(mode == 5'b00010)begin
    Output <= RegFile[1];
end
else if(mode == 5'b00011)begin
    Output <= RegFile[2];
end
else if(mode == 5'b00100)begin
    Output <= RegFile[3];
end
else if(mode == 5'b00101)begin
    Output <= RegFile[4];
end
else if(mode == 5'b00110)begin
    Output <= RegFile[5];
end
else if(mode == 5'b00111)begin
    Output <= RegFile[6];
end
else if(mode == 5'b01000)begin
    Output <= RegFile[7];
end
else if(mode == 5'b01001)begin
    Output <= RegFile[8];
end
else if(mode == 5'b01010)begin
    Output <= RegFile[9];
end
else if(mode == 5'b01011)begin
    Output <= RegFile[10];
end
else if(mode == 5'b01100)begin
    Output <= RegFile[11];
end
else if(mode == 5'b01101)begin
    Output <= RegFile[12];
end
else if(mode == 5'b01110)begin
    Output <= RegFile[13];
end
else if(mode == 5'b01111)begin
    Output <= RegFile[14];
end
else if(mode == 5'b10000)begin
    Output <= RegFile[15];
end
else if(mode == 5'b10001)begin
    Output <= IR;
```

```verilog
end
else if(mode == 5'b10010)begin
    Output <= {3'b0,PC};
end
else if(mode == 5'b10011)begin
    Output <= EXT;
end
if (!pause) begin
    IR <= PROGRAM[PC];

    case(IR[7:4] )
        4'b0000:begin
            if (IR[3:0] == 4'b0000)begin
                 //NOP
            end
            else if (IR[3:0] == 4'b0001) begin
             // LSL ACC
                ACC <= ACC << 1;
            end
            else if (IR[3:0] == 4'b0010) begin
                // LSR ACC
                ACC <= ACC >> 1;
            end
            else if (IR[3:0] == 4'b0011) begin
                // CIR ACC
                ACC <= {ACC[0], ACC[7:1]};
            end
            else if (IR[3:0] == 4'b0100) begin
                // CIL ACC
                ACC <= {ACC[6:0], ACC[7]};
            end
            else if (IR[3:0] == 4'b0101) begin
                // ASR ACC
                ACC <= {ACC[7], ACC[7:1]};
            end
            else if (IR[3:0] == 4'b0110) begin
                // INC ACC
                SUMDIFF = ACC + 1;
                CB = SUMDIFF[8];
                ACC = SUMDIFF[7:0];
            end
            else if (IR[3:0] == 4'b0111) begin
                // DEC ACC
                SUMDIFF = ACC - 1;
                CB = SUMDIFF[8];
                ACC = SUMDIFF[7:0];
            end
```

```
  end

4'b0001:begin
    //ADD Ri
    SUMDIFF = ACC + RegFile[IR[3:0]];
    ACC = SUMDIFF[7:0];
    CB = SUMDIFF[8];

 end
4'b0010:begin
    //SUB Ri
    SUMDIFF = ACC - RegFile[IR[3:0]];
    CB = SUMDIFF[8];
    ACC = SUMDIFF[7:0];
end
4'b0011:begin
    //MUL Ri
    MULTDIV = ACC * RegFile[IR[3:0]];
    ACC = MULTDIV[7:0];
    EXT = MULTDIV[15:8];
end
4'b0100:begin
    //DIV Ri
    ACC = Div;
    EXT = Rem;
end
4'b0101:begin
    //AND Ri
    ACC <= ACC & RegFile[IR[3:0]];
end
4'b0110:begin
    //XOR Ri
    ACC <= ACC ^ RegFile[IR[3:0]];
end
4'b0111:begin
    //CMP Ri
    SUMDIFF = ACC - RegFile[IR[3:0]];
    CB = SUMDIFF[8];
end
4'b1000:begin
    //BR <4-bit address>
    if(CB == 1)begin
        PC <= IR[3:0];
    end
end
4'b1001:begin
    //MOV Ri
```

```
                    ACC <= RegFile[IR[3:0]];
                end
                4'b1010:begin
                    //MOV ACC Ri
                    RegFile[IR[3:0]] <= ACC;
                end
                4'b1011:begin
                    //RET <4-bit address>
                    PC <= IR[3:0];
                end
                4'b1111:begin
                    //HLT
                    $finish;
                end
    endcase
        PC = PC + 1;
        end

        else begin
            IR <= IR;
            PC <= PC;
        end
    end

end

endmodule
```

CONSTRAINT FILE

```
set_property IOSTANDARD LVCMOS33 [get_ports {mode[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {mode[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {mode[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {mode[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {mode[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Output[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Output[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Output[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Output[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Output[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Output[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Output[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Output[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {PC[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {PC[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {PC[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {PC[0]}]
set_property PACKAGE_PIN V17 [get_ports {mode[0]}]
set_property PACKAGE_PIN V16 [get_ports {mode[1]}]
set_property PACKAGE_PIN W16 [get_ports {mode[2]}]
set_property PACKAGE_PIN W17 [get_ports {mode[3]}]
set_property PACKAGE_PIN W15 [get_ports {mode[4]}]
set_property PACKAGE_PIN U16 [get_ports {Output[0]}]
set_property PACKAGE_PIN E19 [get_ports {Output[1]}]
set_property PACKAGE_PIN U19 [get_ports {Output[2]}]
set_property PACKAGE_PIN V19 [get_ports {Output[3]}]
set_property PACKAGE_PIN W18 [get_ports {Output[4]}]
set_property PACKAGE_PIN U15 [get_ports {Output[5]}]
set_property PACKAGE_PIN U14 [get_ports {Output[6]}]
set_property PACKAGE_PIN V14 [get_ports {Output[7]}]
set_property PACKAGE_PIN L1 [get_ports {PC[3]}]
set_property PACKAGE_PIN P1 [get_ports {PC[2]}]
set_property PACKAGE_PIN N3 [get_ports {PC[1]}]
set_property PACKAGE_PIN P3 [get_ports {PC[0]}]
set_property PACKAGE_PIN U3 [get_ports CB]
set_property PACKAGE_PIN W5 [get_ports clk]
set_property PACKAGE_PIN R2 [get_ports pause]
set_property PACKAGE_PIN T1 [get_ports rstn]
set_property PACKAGE_PIN W3 [get_ports slow_clk]
set_property IOSTANDARD LVCMOS33 [get_ports CB]
set_property IOSTANDARD LVCMOS33 [get_ports clk]
set_property IOSTANDARD LVCMOS33 [get_ports pause]
set_property IOSTANDARD LVCMOS33 [get_ports rstn]
set_property IOSTANDARD LVCMOS33 [get_ports slow_clk]
```

SIMULATION CODE

```
`timescale 1ns / 1ps

module Processor(
   clk, rstn, pause,mode,
   CB,
   Output
   ,PC
);

input [4:0] mode;
input clk, rstn, pause;
reg [7:0] PROGRAM [15:0];
reg [7:0] RegFile [15:0];
reg [7:0] ACC;
reg [7:0] EXT;
output reg [7:0] Output;
output reg CB;
reg [8:0] SUMDIFF;
reg [15:0] MULTDIV;
wire [7:0] Div;
reg [7:0] IR;
output reg [3:0] PC;
wire [7:0] Rem;
integer i;
division obj1(.A(ACC), .B(RegFile[IR[3:0]]), .Res(Div), .Rem(Rem));

always @(posedge clk or negedge rstn) begin
   if(!rstn)begin

       RegFile[0] <= 8'd0;
       RegFile[1] <= 8'd1;
       RegFile[2] <= 8'd2;
       RegFile[3] <= 8'd3;
       RegFile[4] <= 8'd4;
       RegFile[5] <= 8'd5;
       RegFile[6] <= 8'd6;
       RegFile[7] <= 8'd7;
       RegFile[8] <= 8'd8;
       RegFile[9] <= 8'd9;
       RegFile[10] <= 8'd10;
       RegFile[11] <= 8'd11;
       RegFile[12] <= 8'd12;
       RegFile[13] <= 8'd13;
       RegFile[14] <= 8'd14;
       RegFile[15] <= 8'd15;
```

```
    /*

        THE PROGRAM
        1. MOV ACC R1   IR <- 1001 0001  (Load R1 in ACC)
        2. XOR R1       IR <- 0110 0001  (Clear ACC)
        3. ADD R5       IR <- 0001 0101  (ACC + R5)
        4. ADD R6       IR <- 0001 0110  (ACC + R6 (which is R5 + R6))
        5. MOV R7 ACC   IR <- 1010 0111  (Store ACC in R7)
        6. HLT          IR <- 1111 1111

    */

    /*
        SIMULATION OF THE PROGRAM
        1. MOV ACC R1   ACC <- 1
        2. XOR R1       ACC <- 0
        3. ADD R5       ACC <- 5
        4. ADD R6       ACC <- 11
        5. MOV R7 ACC   R7 <- 11
        6. HLT
    */

    PROGRAM[0] <= 8'b10010001;    // 1. MOV ACC R1   ACC <- 1
    PROGRAM[1] <= 8'b01100001;    // 2. XOR R1       ACC <- 0
    PROGRAM[2] <= 8'b00010101;    // 3. ADD R5       ACC <- 5
    PROGRAM[3] <= 8'b00010110;    // 4. ADD R6       ACC <- 11
    PROGRAM[4] <= 8'b10100111;    // 5. MOV R7 ACC   R7 <- 11
    PROGRAM[5] <= 8'b11111111;    // 6. HLT

    PC <= 4'b0;
    IR <= 8'b0;
    ACC <= 8'b11111111;
    EXT <= 8'b0;
    CB <= 1'b0;
    SUMDIFF <= 9'b0;
    MULTDIV <= 16'b0;
    Output <= 8'b0;
end

else begin
    if(mode == 5'b00001)begin
        Output <= RegFile[0];
    end
```

```
else if(mode == 5'b11111)begin
    Output <= ACC;
end
else if(mode == 5'b00010)begin
    Output <= RegFile[1];
end
else if(mode == 5'b00011)begin
    Output <= RegFile[2];
end
else if(mode == 5'b00100)begin
    Output <= RegFile[3];
end
else if(mode == 5'b00101)begin
    Output <= RegFile[4];
end
else if(mode == 5'b00110)begin
    Output <= RegFile[5];
end
else if(mode == 5'b00111)begin
    Output <= RegFile[6];
end
else if(mode == 5'b01000)begin
    Output <= RegFile[7];
end
else if(mode == 5'b01001)begin
    Output <= RegFile[8];
end
else if(mode == 5'b01010)begin
    Output <= RegFile[9];
end
else if(mode == 5'b01011)begin
    Output <= RegFile[10];
end
else if(mode == 5'b01100)begin
    Output <= RegFile[11];
end
else if(mode == 5'b01101)begin
    Output <= RegFile[12];
end
else if(mode == 5'b01110)begin
    Output <= RegFile[13];
end
else if(mode == 5'b01111)begin
    Output <= RegFile[14];
end
else if(mode == 5'b10000)begin
    Output <= RegFile[15];
```

```
end
else if(mode == 5'b10001)begin
    Output <= IR;
end
else if(mode == 5'b10010)begin
    Output <= {3'b0,PC};
end
else if(mode == 5'b10011)begin
    Output <= EXT;
end
if (!pause) begin
    IR <= PROGRAM[PC];

    case(IR[7:4] )
        4'b0000:begin
            if (IR[3:0] == 4'b0000)begin
                 //NOP
            end
            else if (IR[3:0] == 4'b0001) begin
             // LSL ACC
                ACC <= ACC << 1;
            end
            else if (IR[3:0] == 4'b0010) begin
                // LSR ACC
                ACC <= ACC >> 1;
            end
            else if (IR[3:0] == 4'b0011) begin
                // CIR ACC
                ACC <= {ACC[0], ACC[7:1]};
            end
            else if (IR[3:0] == 4'b0100) begin
                // CIL ACC
                ACC <= {ACC[6:0], ACC[7]};
            end
            else if (IR[3:0] == 4'b0101) begin
                // ASR ACC
                ACC <= {ACC[7], ACC[7:1]};
            end
            else if (IR[3:0] == 4'b0110) begin
                // INC ACC
                SUMDIFF = ACC + 1;
                CB = SUMDIFF[8];
                ACC = SUMDIFF[7:0];
            end
            else if (IR[3:0] == 4'b0111) begin
                // DEC ACC
                SUMDIFF = ACC - 1;
        CB = SUMDIFF[8];
        ACC = SUMDIFF[7:0];
    end
  end

4'b0001:begin
    //ADD Ri
    SUMDIFF = ACC + RegFile[IR[3:0]];
    ACC = SUMDIFF[7:0];
    CB = SUMDIFF[8];

 end
4'b0010:begin
    //SUB Ri
    SUMDIFF = ACC - RegFile[IR[3:0]];
    CB = SUMDIFF[8];
    ACC = SUMDIFF[7:0];
end
4'b0011:begin
    //MUL Ri
    MULTDIV = ACC * RegFile[IR[3:0]];
    ACC = MULTDIV[7:0];
    EXT = MULTDIV[15:8];
end
4'b0100:begin
    //DIV Ri
    // Need a synthesizable way to implement division
    ACC = Div;
    EXT = Rem;
end
4'b0101:begin
    //AND Ri
    ACC <= ACC & RegFile[IR[3:0]];
end
4'b0110:begin
    //XOR Ri
    ACC <= ACC ^ RegFile[IR[3:0]];
end
4'b0111:begin
    //CMP Ri
    SUMDIFF = ACC - RegFile[IR[3:0]];
    CB = SUMDIFF[8];
end
4'b1000:begin
    //BR <4-bit address>
    if(CB == 1)begin
        PC <= IR[3:0];
```

```
                        end
                    end
                    4'b1001:begin
                        //MOV Ri
                        ACC <= RegFile[IR[3:0]];
                    end
                    4'b1010:begin
                        //MOV ACC Ri
                        RegFile[IR[3:0]] <= ACC;
                    end
                    4'b1011:begin
                        //RET <4-bit address>
                        PC <= IR[3:0];
                    end
                    4'b1111:begin
                        //HLT
                         $finish;
                    end
            endcase
                PC = PC + 1;
                end

                else begin
                    IR = IR;
                    PC = PC;
                end
            end

        end

        endmodule
```

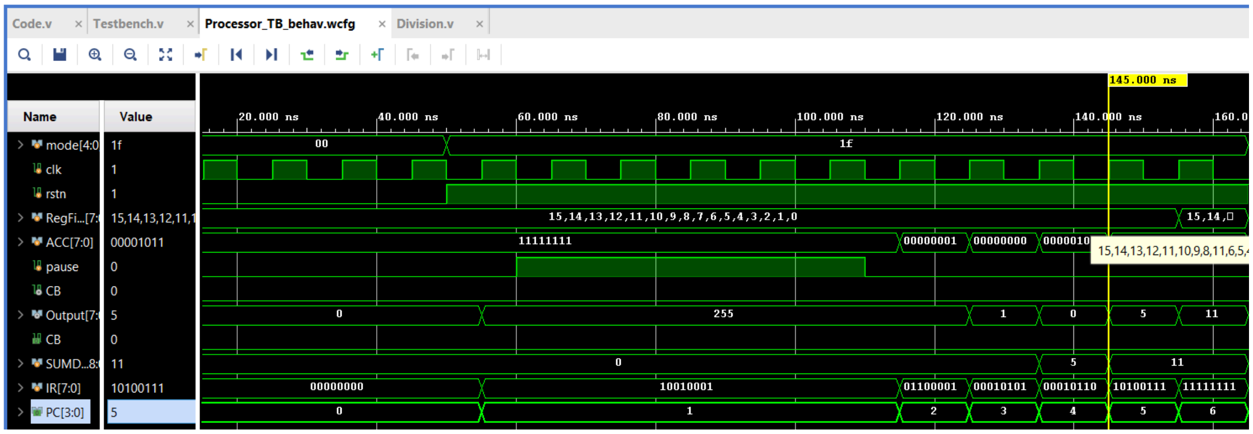
SYNTHESIZABLE DIVISION CODE

```
module division(A, B, Res, Rem);

 parameter WIDTH = 8;

 input [WIDTH-1:0] A;
 input [WIDTH-1:0] B;
 output [WIDTH-1:0] Res;
 output reg [WIDTH-1:0] Rem;

 reg [WIDTH-1:0] Res;
 reg [WIDTH-1:0] a1, b1;
 reg [WIDTH:0] p1;
 integer i;

 always @(A or B)
   begin
     a1 = A;
     b1 = B;
     p1 = 0;

     for (i = 0; i < WIDTH; i = i + 1) begin
       // Shift in the next bit of the dividend and subtract the divisor.
       p1 = {p1[WIDTH-2:0], a1[WIDTH-1]};
       a1[WIDTH-1:1] = a1[WIDTH-2:0];
       p1 = p1 - b1;

       // Check if the result is negative (borrow occurred).
       // If so, add the divisor back and set the corresponding bit of a1 to 1.
       if (p1[WIDTH-1] == 1'b1) begin
         a1[0] = 0;
         p1 = p1 + b1;
       end else begin
         a1[0] = 1;
       end
     end

     // After the loop, the quotient is in a1 and the remainder is in p1.
     Res = a1;
     Rem = p1;
   end

endmodule
```

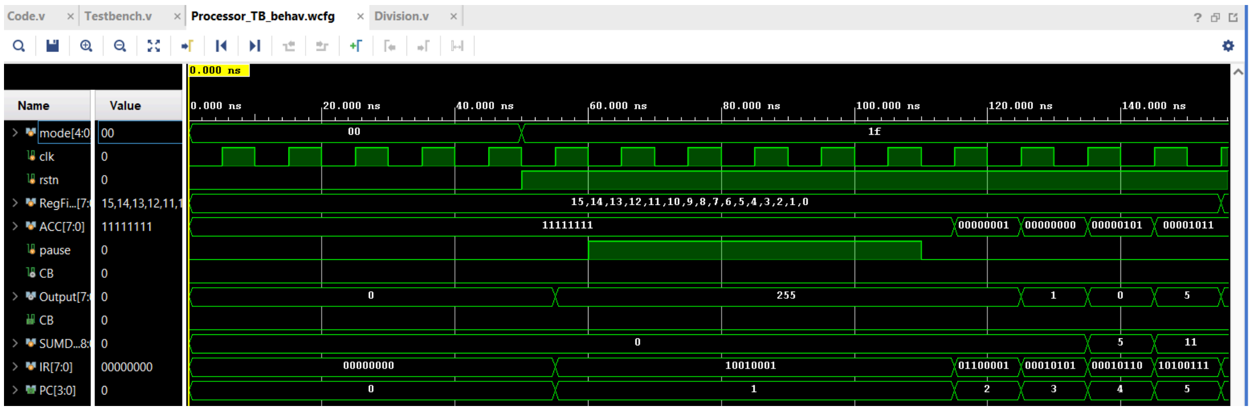
TESTBENCH

```
`timescale 1ns / 1ps

module Processor_TB;
parameter CLK_PERIOD = 10;
reg [4:0] mode;
reg clk,rstn, pause;
wire CB;
wire [7:0] Output;
Processor dut (
   .clk(clk),
   .rstn(rstn),
   .pause(pause),
   .mode(mode),
   .CB(CB),
   .Output(Output)
);

always #((CLK_PERIOD/2)) clk = ~clk;

initial begin
   clk = 0;
   rstn = 0;
   pause = 0;
   mode = 5'b0;
   #50 rstn = 1;mode = 5'b11111;
   #10 pause = 1;mode = 5'b11111;
   #50 pause = 0;mode = 5'b11111;
   #250 $finish;
end

endmodule
```

SIMULATION RESULTS



Name	Value
mode[4:0]	1f
clk	1
rstn	1
RegFi...[7:	15,14,13,12,11,1
ACC[7:0]	00001011
pause	0
CB	0
Output[7:	5
CB	0
SUMD...8:	11
IR[7:0]	10100111
PC[3:0]	5



Name	Value
mode[4:0]	00
clk	0
rstn	0
RegFi...[7:	15,14,13,12,11,1
ACC[7:0]	11111111
pause	0
CB	0
Output[7:	0
CB	0
SUMD...8:	0
IR[7:0]	00000000
PC[3:0]	0