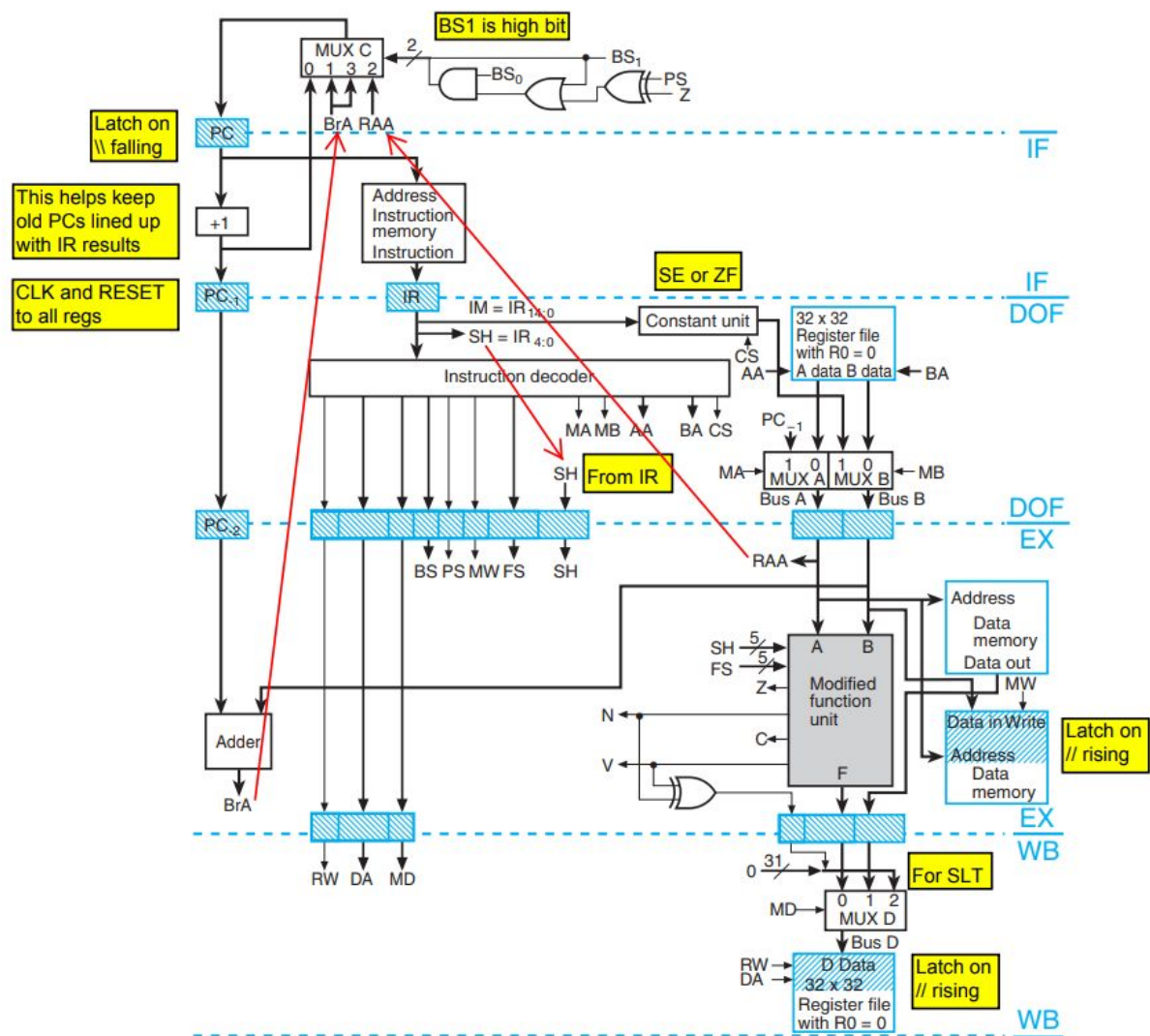


RISC CPU

Austin Goodman

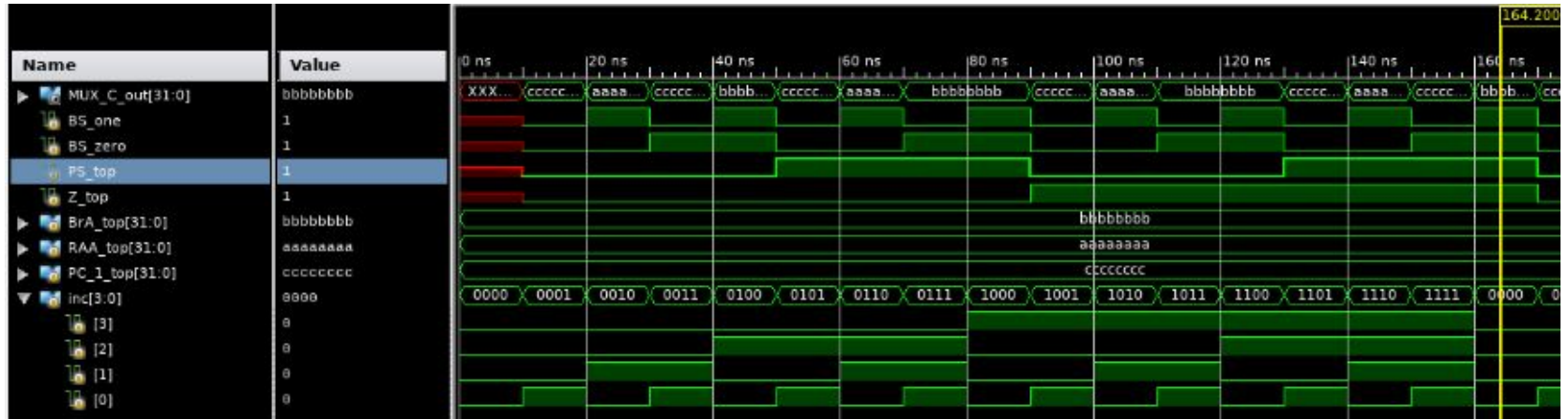


Top portion

```
21 module TOP(  
22     input BS_one,  
23     input BS_zero,  
24     input PS_top,  
25     input Z_top,  
26     input [31:0]BrA_top,  
27     input [31:0]RAA_top,  
28     input [31:0]PC_1_top,  
29     output [31:0]MUX_C_out  
30 );  
31  
32 reg [1:0]mux_c_select;  
33  
34  
35 reg [31:0]MUX_C_out_reg;  
36 assign MUX_C_out = MUX_C_out_reg;  
37  
38 always @(*)  
39 begin  
40     mux_c_select = {BS_one, (BS_zero & (BS_one | (PS_top ^ Z_top)))};  
41     case(mux_c_select)  
42     0: MUX_C_out_reg <= PC_1_top;  
43     1: MUX_C_out_reg <= BrA_top;  
44     2: MUX_C_out_reg <= RAA_top;  
45     3: MUX_C_out_reg <= BrA_top;  
46     endcase  
47 end  
48  
49  
50 endmodule
```

TEST BENCH

```
47 initial  
48 begin  
49     BrA_top <= 32'hBBBBBBBB;  
50     RAA_top <= 32'hAAAAAAAA;  
51     PC_1_top <= 32'hCCCCCCCC;  
52  
53     for(inc=0;inc<16;inc=inc+1)begin  
54         #10  
55         BS_one <= inc[0];  
56         BS_zero <= inc[1];  
57         PS_top <= inc[2];  
58         Z_top <= inc[3];  
59     end  
60  
61 end
```



Instruction Memory

```
21 module memPROG(  
22     input [31:0] PC,  
23     output [31:0] dataout  
24 );  
25  
26 `include "PROG.INC"  
27  
28 integer i;  
29  
30 reg [31:0] memword [255:0];  
31 initial  
32 begin  
33     memword[0] = PROG0;  
34     memword[1] = PROG1;  
35     memword[2] = PROG2;  
36     memword[3] = PROG3;  
37     memword[4] = PROG4;  
38     memword[5] = PROG5;  
39     memword[6] = PROG6;  
40     memword[7] = PROG7;  
41     for(i=8; i< 256; i = i+1)  
42         memword[i] = PROGX;  
43 end  
44  
45  
46 assign dataout = memword[PC];  
47  
48  
49  
50 endmodule
```

IF

```

21 module IF(
22     input  CLOCK,
23     input  [31:0]PC,
24     output [31:0]PC_M1,
25     output [31:0]IR
26 );
27
28     reg [31:0]pc_clocked;
29     assign PC_M1 = PC + 1;
30
31     always @(negedge CLOCK)
32     begin
33         pc_clocked <= PC;
34     end
35
36     memPROG instruction_mem(
37         .PC(pc_clocked),
38         .dataout(IR)
39     );
40
41
42 endmodule

```

```

21 module IF_tb();
22
23     reg CLOCK;
24     reg [31:0]PC;
25
26     wire [31:0]PC_M1;
27     wire [31:0]IR;
28
29     IF uut(
30         .CLOCK(CLOCK),
31         .PC(PC),
32         .PC_M1(PC_M1),
33         .IR(IR)
34     );
35
36     integer i;
37
38     initial
39     begin
40         CLOCK = 0;
41         for(i = 0; i < 10; i = i + 1)
42             begin
43                 #10
44                 PC = i;
45             end
46
47         end
48
49
50     always
51     begin
52         #5
53         CLOCK = ~CLOCK;
54     end
55
56 endmodule

```

Name	Value	0 ns	10 ns	20 ns	30 ns	40 ns	50 ns	60 ns	70 ns	80 ns	90 ns	100 ns
PC_M1[31:0]	XXXXXXXX	XXXXXXXX	00000001	00000002	00000003	00000004	00000005	00000006	00000007	00000008	00000009	00000000
IR[31:0]	XXXXXXXX	XXXXXXXX	00000000	11111111	22222222	33333333	44444444	55555555	66666666	77777777	88888888	99999999
CLOCK	1	0	1	0	1	0	1	0	1	0	1	0
PC[31:0]	XXXXXXXX	XXXXXXXX	00000000	00000001	00000002	00000003	00000004	00000005	00000006	00000007	00000008	00000009
I[31:0]	88888888	00000000	00000001	00000002	00000003	00000004	00000005	00000006	00000007	00000008	00000009	00000000

```

89 always@(*)
90 begin
91     opcode = IR_instruction[31:25];
92     DR = IR_instruction[24:20];
93     SA = IR_instruction[19:15];
94     SB = IR_instruction[14:10];
95
96
97     if(RESET)
98     begin
99         RW_reg    <= 0;
100        MD_reg    <= 2'b00;
101        BS_reg    <= 2'b00;
102        PS_reg    <= 0;
103        MW_reg    <= 0;
104        FS_reg    <= 5'b000000;
105        MB_reg    <= 0;
106        MA_reg    <= 0;
107        CS_reg    <= 0;
108        AA_reg    <= 5'b000000;
109        BA_reg    <= 5'b000000;
110        DA_reg    <= 5'b000000;
111    end
112    casex(opcode)
113        NOP:
114            begin
115                RW_reg <= 0;
116                MD_reg <= 2'bxx;
117                BS_reg <= 2'b00;
118                PS_reg <= 1'bx;
119                MW_reg <= 0;
120                FS_reg <= 5'bxxxxxx;
121                MB_reg <= 1'bx;
122                MA_reg <= 1'bx;
123                CS_reg <= 1'bx;
124                AA_reg <= 5'b000000;
125                BA_reg <= 5'b000000;
126                DA_reg <= 5'b000000;
127            end
128        ADD:

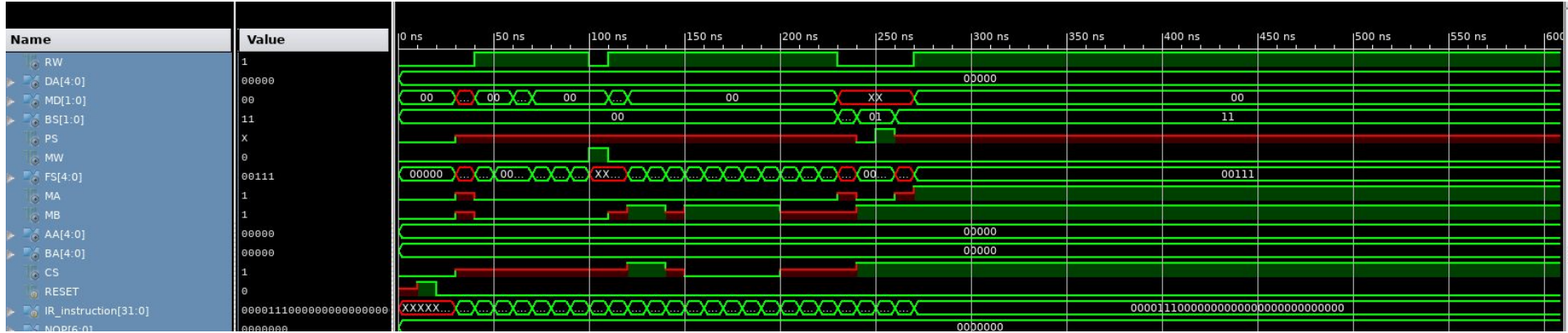
```

```

initial
begin
    #10
    RESET <= 1;
    #10
    RESET <= 0;
    #10
    IR_instruction <= {7'b0000000, 25'b0000000000000000000000000000};
    #10
    IR_instruction <= {7'b0000010, 25'b0000000000000000000000000000};
    #10
    IR_instruction <= {7'b0000101, 25'b0000000000000000000000000000};
    #10
    IR_instruction <= {7'b1100101, 25'b0000000000000000000000000000};
    #10
    IR_instruction <= {7'b0001000, 25'b0000000000000000000000000000};
    #10
    IR_instruction <= {7'b0001010, 25'b0000000000000000000000000000};
    #10
    IR_instruction <= {7'b0001100, 25'b0000000000000000000000000000};
    #10
    IR_instruction <= {7'b0000001, 25'b0000000000000000000000000000};
    #10
    IR_instruction <= {7'b0100001, 25'b0000000000000000000000000000};
    #10
    IR_instruction <= {7'b0100010, 25'b0000000000000000000000000000};
    #10
    IR_instruction <= {7'b0100101, 25'b0000000000000000000000000000};
    #10
    IR_instruction <= {7'b0101110, 25'b0000000000000000000000000000};
    #10
    IR_instruction <= {7'b0101000, 25'b0000000000000000000000000000};
    #10
    IR_instruction <= {7'b0101010, 25'b0000000000000000000000000000};
    #10
    IR_instruction <= {7'b0101100, 25'b0000000000000000000000000000};
    #10
    IR_instruction <= {7'b1100010, 25'b0000000000000000000000000000};
    #10
    IR_instruction <= {7'b1000101, 25'b0000000000000000000000000000};
    #10
    IR_instruction <= {7'b1000000, 25'b0000000000000000000000000000};
    #10
    IR_instruction <= {7'b0110000, 25'b0000000000000000000000000000};
    #10
    IR_instruction <= {7'b0110001, 25'b0000000000000000000000000000};
    #10
    IR_instruction <= {7'b1100001, 25'b0000000000000000000000000000};
    #10
    IR_instruction <= {7'b0100000, 25'b0000000000000000000000000000};
    #10
    IR_instruction <= {7'b1100000, 25'b0000000000000000000000000000};
    #10
    IR_instruction <= {7'b1000100, 25'b0000000000000000000000000000};
    #10
    IR_instruction <= {7'b0000111, 25'b0000000000000000000000000000};
end

```


Instruction Decoder SIM



Constant unit

CS = 1 => SE

CS = 0 => ZF

```
21 module Constant_Unit(  
22     input  [14:0]IM,  
23     input      CS,  
24     output [31:0]SEorZF  
25 );  
26  
27 reg [31:0]SEorZF_reg;  
28 assign SEorZF = SEorZF_reg;  
29  
30 always @(*)  
31     begin  
32         if(CS == 1)  
33             begin  
34                 if(IM[14] == 1)  
35                     begin  
36                         SEorZF_reg <= {17'b1111111111111111, IM};  
37                     end  
38                 if(IM[14] == 0)  
39                     begin  
40                         SEorZF_reg <= {17'b0000000000000000, IM};  
41                     end  
42                 end  
43             if(CS == 0)  
44                 begin  
45                     SEorZF_reg <= {17'b0000000000000000, IM};  
46                 end  
47             end  
48         end  
49  
50 endmodule
```

```
34     initial  
35     begin  
36         #10  
37         IM <= 15'b1000000000000000;  
38         CS <= 1;  
39         #10  
40         CS <= 0;  
41         #10  
42         CS <= 1;  
43         #10  
44         CS <= 0;  
45         #10  
46         IM <= 15'b0000000000000000;  
47         CS <= 1;  
48         #10  
49         CS <= 0;  
50     end
```

Name	Value
SEorZF[31:0]	00000000000000000000000000000000
IM[14:0]	00000000000000
CS	0



MUX A & B

MA controls BUS_A

MB controls BUS_B

```
21 module MUXA_B(  
22     input [31:0] PC_M1,  
23     input [31:0] A_DATA,  
24     input [31:0] B_DATA,  
25     input [31:0] SEorZF,  
26     input      MA,  
27     input      MB,  
28     output[31:0] BUS_A,  
29     output[31:0] BUS_B  
30 );  
31  
32 reg [31:0]BUS_A_reg;  
33 reg [31:0]BUS_B_reg;  
34  
35 assign BUS_A = BUS_A_reg;  
36 assign BUS_B = BUS_B_reg;  
37  
38  
39 always @(*)  
40     begin  
41         if(MA == 1)  
42             begin  
43                 BUS_A_reg <= PC_M1;  
44             end  
45         if(MA == 0)  
46             begin  
47                 BUS_A_reg <= A_DATA;  
48             end  
49  
50         if(MB == 1)  
51             begin  
52                 BUS_B_reg <= SEorZF;  
53             end  
54         if(MB == 0)  
55             begin  
56                 BUS_B_reg <= B_DATA;  
57             end  
58     end  
end
```

```
45 initial  
46     begin  
47         PC_M1 <= 0;  
48         A_DATA<= 0;  
49         B_DATA<= 0;  
50         SEorZF<= 0;  
51         #100  
52         PC_M1 <= 32'hFFFFFFFF;  
53         A_DATA<= 32'hAAAAAAAA;  
54         B_DATA<= 32'hBBBBBBBB;  
55         SEorZF<= 32'h00000001;  
56         #10  
57         MA <= 1;  
58         #10  
59         MA <= 0;  
60         #10  
61         MB <= 1;  
62         #10  
63         MB <= 0;  
64  
65     end
```



DOF module

```

21 module DOF(
22     input      CLOCK,
23     input      RESET,
24     input [31:0] PC_M1,
25     input [31:0] IR,
26     input [31:0] A_DATA,
27     input [31:0] B_DATA,
28     output [31:0] BUS_A,
29     output [31:0] BUS_B,
30     output [4:0] AA,
31     output [4:0] BA,
32     output      RW,
33     output [4:0] DA,
34     output [1:0] MD,
35     output [1:0] BS,
36     output      PS,
37     output      MW,
38     output [4:0] FS,
39     output [4:0] SH,
40     output [31:0] PC_M2
41 );
42
43 reg [31:0] PC_M1_clocked;
44 reg [31:0] IR_clocked;
45 reg [14:0] IM;
46
47 reg [31:0] BUS_A_reg;
48 reg [31:0] BUS_B_reg;
49 reg [4:0] AA_reg;
50 reg [4:0] BA_reg;
51 reg      RW_reg;
52 reg [4:0] DA_reg;
53 reg [1:0] MD_reg;
54 reg [1:0] BS_reg;
55 reg      PS_reg;
56 reg      MW_reg;
57 reg [4:0] FS_reg;
58 reg [4:0] SH_reg;
59 reg [31:0] PC_M2_reg;

```

```

61 assign BUS_A = BUS_A_reg;
62 assign BUS_B = BUS_B_reg;
63 assign AA = AA_reg;
64 assign BA = BA_reg;
65 assign RW = RW_reg;
66 assign DA = DA_reg;
67 assign MD = MD_reg;
68 assign BS = BS_reg;
69 assign PS = PS_reg;
70 assign MW = MW_reg;
71 assign FS = FS_reg;
72 assign SH = SH_reg;
73 assign PC_M2 = PC_M2_reg;
74
75 always @(negedge CLOCK)
76 begin
77     PC_M1_clocked <= PC_M1;
78     IR_clocked    <= IR;
79 end
80
81 always @(*)
82 begin
83     SH_reg = IR[4:0];
84     IM = IR[14:0];
85     PC_M2_reg = PC_M1_clocked;
86 end
87
88 wire CS_wire;
89 wire MA_wire;
90 wire MB_wire;
91 wire SEorZF_wire;

```

```

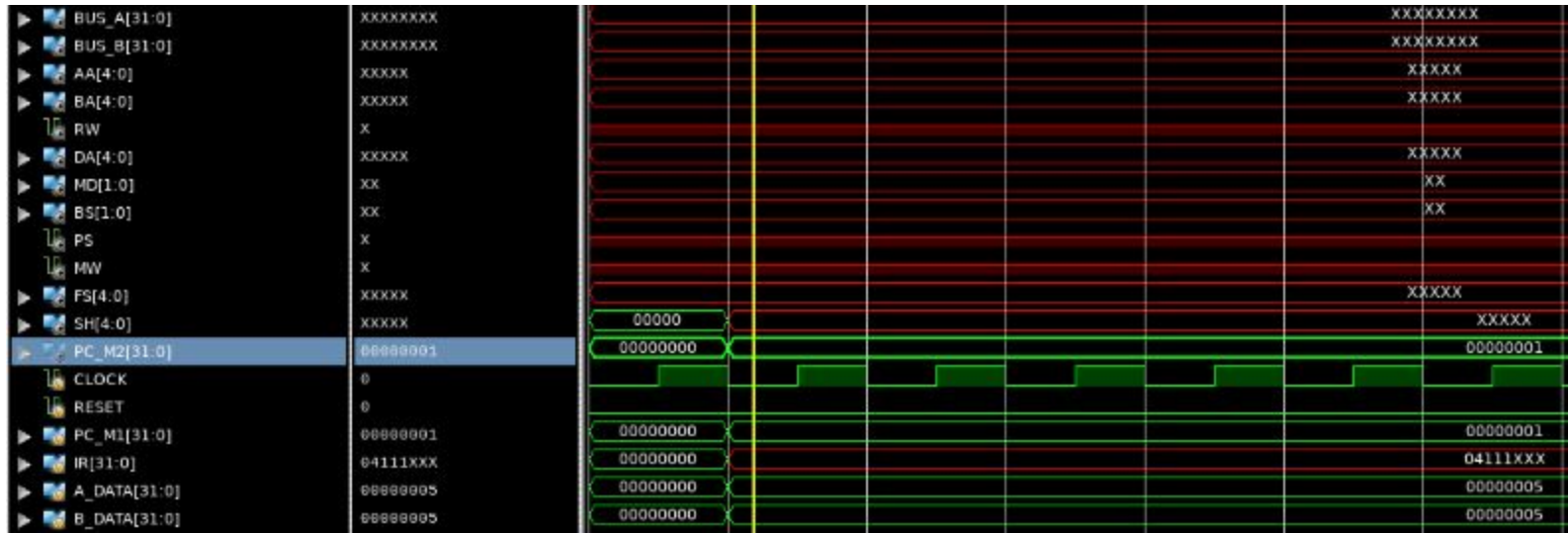
94 Instruction_decoder instruction_dec(
95     .RESET(RESET),
96     .IR_instruction(IR_clocked),
97     .RW(RW_reg), //out
98     .DA(DA_reg), //out
99     .MD(MD_reg), //out
100    .BS(BS_reg), //out
101    .PS(PS_reg), //out
102    .MW(MW_reg), //out
103    .FS(FS_reg), //out
104    .MA(MA_wire),
105    .MB(MB_wire),
106    .AA(AA_reg), //out to registerfile
107    .BA(BA_reg), //out to registerfile
108    .CS(CS_wire)
109 );
110
111 Constant_Unit const_unit(
112     .IM(IM),
113     .CS(CS_wire),
114     .SEorZF(SEorZF_wire)
115 );
116
117 MUXA_B mux_ab(
118     .PC_M1(PC_M1_clocked),
119     .A_DATA(A_DATA),
120     .B_DATA(B_DATA),
121     .SEorZF(SEorZF_wire),
122     .MA(MA_wire),
123     .MB(MB_wire),
124     .BUS_A(BUS_A_reg),
125     .BUS_B(BUS_B_reg)
126 );
127

```

DOF_tb

No clue what's wrong

```
68   initial
69   begin
70       CLOCK = 0;
71       RESET = 0;
72       PC_M1 = 0;
73       IR = 0;
74       A_DATA = 0;
75       B_DATA = 0;
76       #10
77       PC_M1 = 32'h00000001;
78       IR = 32'b000000100000100010001000xxxxxxx; //add R1, R2, R3;
79       A_DATA = 32'h00000005;
80       B_DATA = 32'h00000005;
81   end
```



MFU

```

22 module Mod_Function_unit(
23     input RESET,
24     input [31:0]A,
25     input [31:0]B,
26     input [4:0]SH,
27     input [4:0]FS,
28     output Z_out,
29     output C_out,
30     output N_out,
31     output V_out,
32     output [31:0]F
33 );
34 //import params
35
36 reg Z_out_reg;
37 reg N_out_reg;
38 reg V_out_reg;
39 reg [31:0]F_reg;
40
41 reg [32:0]test;
42 wire [8:0]out_carry;
43
44 assign Z_out = Z_out_reg;
45 assign N_out = N_out_reg;
46 assign V_out = V_out_reg;
47 assign F = F_reg;
48
49 assign out_carry = {1'b0, A} + {1'b0, B};
50 assign C_out = out_carry[8];
51
52 initial
53 begin
54     Z_out_reg <= 0;
55     N_out_reg <= 0;
56     V_out_reg <= 0;
57     F_reg <= 32'h00000000;
58 end

```

```

60 always @(*)
61 begin
62     if(RESET)
63     begin
64         Z_out_reg <= 0;
65         N_out_reg <= 0;
66         V_out_reg <= 0;
67         F_reg <= 32'h00000000;
68     end
69     test = A - B;
70     if(test[32] == 1)
71     begin
72         N_out_reg <= 1;
73     end
74     if(test[32] == 0)
75     begin
76         N_out_reg <= 0;
77     end
78     if(test == 0)
79     begin
80         Z_out_reg <= 0;
81     end
82     if(test < 0)
83     begin
84         V_out_reg <= 1;
85     end
86
87     case(FS)
88     5'b000000:
89     begin
90         F_reg <= A;
91     end
92     5'b000010:
93     begin
94         F_reg <= A + B;
95     end
96     5'b000101:
97     begin
98         F_reg <= A - B;
99     end
100 end

```

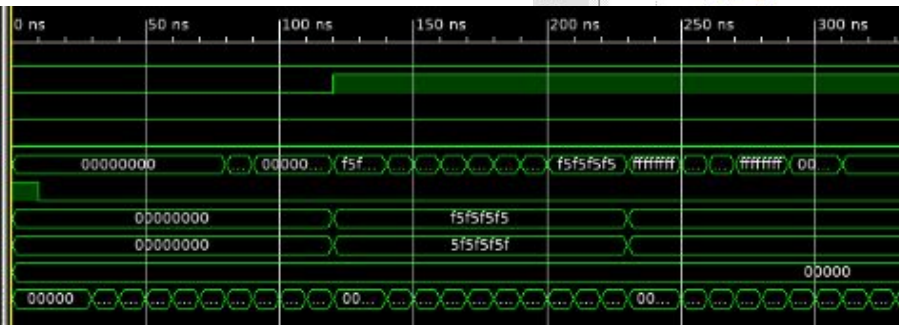
```

101
102     begin
103         F_reg <= A & B;
104     end
105     5'b010010:
106     begin
107         F_reg <= A | B;
108     end
109     5'b01100:
110     begin
111         F_reg <= A ^ B;
112     end
113     5'b01110:
114     begin
115         F_reg <= ~A;
116     end
117     5'b10000:
118     begin
119         F_reg <= A << SH
120     end
121     5'b10001:
122     begin
123         F_reg <= A >> SH
124     end
125     5'b00111:
126     begin
127         F_reg <= A;
128     end
129     default:
130     begin
131         F_reg <= 0;
132     end
133
134 endcase

```


MFU_tb

Name	Value
Z_out	0
C_out	0
N_out	0
V_out	0
F[31:0]	00000000
RESET	1
A[31:0]	00000000
B[31:0]	00000000
SH[4:0]	00000
FS[4:0]	00000



```

21 module Mod_Function_unit_tb();
22
23     reg RESET;
24     reg [31:0]A;
25     reg [31:0]B;
26     reg [4:0]SH;
27     reg [4:0]FS;
28
29     wire Z_out;
30     wire C_out;
31     wire N_out;
32     wire V_out;
33     wire [31:0]F;
34
35     Mod_Function_unit uut(
36         .RESET(RESET),
37         .A(A),
38         .B(B),
39         .SH(SH),
40         .FS(FS),
41         .Z_out(Z_out),
42         .C_out(C_out),
43         .N_out(N_out),
44         .V_out(V_out),
45         .F(F)
46     );
47
48     initial
49     begin
50         RESET <= 1;
51         A <= 0;
52         B <= 0;
53         SH <= 0;
54
55         A <= 32'h00000000;
56         B <= 32'h00000000;
57         FS <= 5'b00000;
58         #10
59         FS <= 5'b00000;
60         #10
61         FS <= 5'b000010;
62         #10
63         FS <= 5'b01000;
64         #10
65         FS <= 5'b01010;
66         #10
67         FS <= 5'b01100;
68         #10
69         FS <= 5'b01110;
70         #10
71         FS <= 5'b10000;
72         #10
73         FS <= 5'b10001;
74         #10
75         FS <= 5'b00111;
76         #10
77         A <= 32'hF5F5F5F5;
78         B <= 32'h5F5F5F5F;
79         FS <= 5'b00000;
80         #10
81         FS <= 5'b00000;
82         #10
83         FS <= 5'b000010;
84         #10
85         FS <= 5'b00101;
86         #10
87         FS <= 5'b01000;
88         #10
89         FS <= 5'b01010;
90
91         #10
92         FS <= 5'b00101;
93         #10
94         FS <= 5'b01000;
95         #10
96         FS <= 5'b01010;
97
98         #10
99         FS <= 5'b00101;
100        #10
101        FS <= 5'b01000;
102        #10
103        FS <= 5'b01010;
104        #10
105        FS <= 5'b00101;
106        #10
107        FS <= 5'b01000;
108        #10
109        FS <= 5'b01010;
110        #10
111        FS <= 5'b00101;
112        #10
113        FS <= 5'b01000;
114        #10
115        FS <= 5'b01010;
116        #10
117        FS <= 5'b00101;
118        #10
119        FS <= 5'b01000;
120        #10
121        FS <= 5'b01010;
122        #10
123        FS <= 5'b00101;
124        #10
125        FS <= 5'b01000;
126        #10
127        FS <= 5'b01010;
128        #10
129        FS <= 5'b00101;
130        #10
131        FS <= 5'b01000;
132        #10
133        FS <= 5'b01010;
134        #10
135        FS <= 5'b00101;
136        #10
137        FS <= 5'b01000;
138        #10
139        FS <= 5'b01010;
140        #10
141        FS <= 5'b00101;
142        #10
143        FS <= 5'b01000;
144        #10
145        FS <= 5'b01010;
146        #10
147        FS <= 5'b00101;
148        #10
149        FS <= 5'b01000;
150        #10
151        FS <= 5'b01010;
152        #10
153        FS <= 5'b00101;
154        #10
155        FS <= 5'b01000;
156        #10
157        FS <= 5'b01010;
158        #10
159        FS <= 5'b00101;
160        #10
161        FS <= 5'b01000;
162        #10
163        FS <= 5'b01010;
164        #10
165        FS <= 5'b00101;
166        #10
167        FS <= 5'b01000;
168        #10
169        FS <= 5'b01010;
170        #10
171        FS <= 5'b00101;
172        #10
173        FS <= 5'b01000;
174        #10
175        FS <= 5'b01010;
176        #10
177        FS <= 5'b00101;
178        #10
179        FS <= 5'b01000;
180        #10
181        FS <= 5'b01010;
182        #10
183        FS <= 5'b00101;
184        #10
185        FS <= 5'b01000;
186        #10
187        FS <= 5'b01010;
188        #10
189        FS <= 5'b00101;
190        #10
191        FS <= 5'b01000;
192        #10
193        FS <= 5'b01010;
194        #10
195        FS <= 5'b00101;
196        #10
197        FS <= 5'b01000;
198        #10
199        FS <= 5'b01010;
200        #10
201        FS <= 5'b00101;
202        #10
203        FS <= 5'b01000;
204        #10
205        FS <= 5'b01010;
206        #10
207        FS <= 5'b00101;
208        #10
209        FS <= 5'b01000;
210        #10
211        FS <= 5'b01010;
212        #10
213        FS <= 5'b00101;
214        #10
215        FS <= 5'b01000;
216        #10
217        FS <= 5'b01010;
218        #10
219        FS <= 5'b00101;
220        #10
221        FS <= 5'b01000;
222        #10
223        FS <= 5'b01010;
224        #10
225        FS <= 5'b00101;
226        #10
227        FS <= 5'b01000;
228        #10
229        FS <= 5'b01010;
230        #10
231        FS <= 5'b00101;
232        #10
233        FS <= 5'b01000;
234        #10
235        FS <= 5'b01010;
236        #10
237        FS <= 5'b00101;
238        #10
239        FS <= 5'b01000;
240        #10
241        FS <= 5'b01010;
242        #10
243        FS <= 5'b00101;
244        #10
245        FS <= 5'b01000;
246        #10
247        FS <= 5'b01010;
248        #10
249        FS <= 5'b00101;
250        #10
251        FS <= 5'b01000;
252        #10
253        FS <= 5'b01010;
254        #10
255        FS <= 5'b00101;
256        #10
257        FS <= 5'b01000;
258        #10
259        FS <= 5'b01010;
260        #10
261        FS <= 5'b00101;
262        #10
263        FS <= 5'b01000;
264        #10
265        FS <= 5'b01010;
266        #10
267        FS <= 5'b00101;
268        #10
269        FS <= 5'b01000;
270        #10
271        FS <= 5'b01010;
272        #10
273        FS <= 5'b00101;
274        #10
275        FS <= 5'b01000;
276        #10
277        FS <= 5'b01010;
278        #10
279        FS <= 5'b00101;
280        #10
281        FS <= 5'b01000;
282        #10
283        FS <= 5'b01010;
284        #10
285        FS <= 5'b00101;
286        #10
287        FS <= 5'b01000;
288        #10
289        FS <= 5'b01010;
290        #10
291        FS <= 5'b00101;
292        #10
293        FS <= 5'b01000;
294        #10
295        FS <= 5'b01010;
296        #10
297        FS <= 5'b00101;
298        #10
299        FS <= 5'b01000;
300        #10
301        FS <= 5'b01010;
302        #10
303        FS <= 5'b00101;
304        #10
305        FS <= 5'b01000;
306        #10
307        FS <= 5'b01010;
308        #10
309        FS <= 5'b00101;
310        #10
311        FS <= 5'b01000;
312        #10
313        FS <= 5'b01010;
314        #10
315        FS <= 5'b00101;
316        #10
317        FS <= 5'b01000;
318        #10
319        FS <= 5'b01010;
320        #10
321        FS <= 5'b00101;
322        #10
323        FS <= 5'b01000;
324        #10
325        FS <= 5'b01010;
326        #10
327        FS <= 5'b00101;
328        #10
329        FS <= 5'b01000;
330        #10
331        FS <= 5'b01010;
332        #10
333        FS <= 5'b00101;
334        #10
335        FS <= 5'b01000;
33
```

Data Memory

```

21 module memDATA(
22     input clk,
23     input [31:0] addr,
24     input MW,
25     input [31:0] datain,
26     output [31:0] dataout
27 );
28
29 reg [31:0] memword [63:0];
30 integer i;
31
32 initial
33 begin
34     for(i=0; i<64; i=i+1)
35         memword[i] <= i;
36     end
37
38 always @(posedge clk)
39 begin
40     if(MW)
41         memword[addr] <= datain;
42     end
43
44 assign dataout = memword[addr];
45
46 endmodule

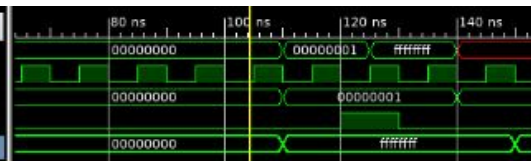
```

```

21 module memDATA_tb();
22
23 reg clk;
24 reg [31:0] addr;
25 reg MW;
26 reg [31:0] datain;
27
28 wire [31:0] dataout;
29
30 memDATA uut (
31     .clk(clk),
32     .addr(addr),
33     .MW(MW),
34     .datain(datain),
35     .dataout(dataout)
36 );
37
38 initial
39 begin
40     clk = 0;
41     addr = 32'h00000000;
42     MW = 0;
43     datain = 32'h00000000;
44     #100
45     #10
46     addr = 32'h00000001;
47     datain = 32'hFFFFFFF;
48     #10
49     MW = 1;
50     #10
51     MW = 0;
52     #10
53     addr = 32'hFFFFFFF;
54     #10
55     datain = 32'h01010101;
56     end
57
58
59 always
60 begin
61     #5
62     clk = ~clk;
63     end
64

```

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



EX

```

21 module EX(
22     input CLOCK,
23     input RESET,
24     input [31:0] PC_M2,
25     input RW,
26     input[4:0] DA,
27     input[1:0] MD,
28     input[1:0] BS,
29     input PS,
30     input MW,
31     input[4:0] FS,
32     input[4:0] SH,
33     input [31:0] BUS_A,
34     input [31:0] BUS_B,
35     output[31:0] BrA,
36     output[31:0] RAA,
37     output RW_out,
38     output[4:0] DA_out,
39     output[1:0] MD_out,
40     output BS_one,
41     output BS_zero,
42     output PS_out,
43     output Z,
44     output V,
45     output N,
46     output C,
47     output VxorN,
48     output [31:0] F,
49     output [31:0] Data_Out
50 );
51 
```

```

54 reg[31:0] BrA_reg;
55 reg[31:0] RAA_reg;
56 reg RW_out_reg;
57 reg[4:0] DA_out_reg;
58 reg[1:0] MD_out_reg;
59 reg BS_one_reg;
60 reg BS_zero_reg;
61 reg PS_out_reg;
62 wire Z_reg;
63 wire V_reg;
64 wire N_reg;
65 wire C_reg;
66 reg VxorN_reg;
67 wire [31:0] F_reg;
68 wire [31:0] Data_Out_reg;
69 assign BrA = BrA_reg;
70 assign RAA = RAA_reg;
71 assign RW_out = RW_out_reg;
72 assign DA_out = DA_out_reg;
73 assign MD_out = MD_out_reg;
74 assign BS_one = BS_one_reg;
75 assign BS_zero = BS_zero_reg;
76 assign PS_out = PS_out_reg;
77 assign Z = Z_reg;
78 assign V = V_reg;
79 assign N = N_reg;
80 assign C = C_reg;
81 assign VxorN = VxorN_reg;
82 assign F = F_reg;
83 assign Data_Out = Data_Out_reg;

```

```

85 reg [31:0] PC_M2_clocked;
86 reg RW_clocked;
87 reg [4:0] DA_clocked;
88 reg [1:0] MD_clocked;
89 reg [1:0] BS_clocked;
90 reg PS_clocked;
91 reg MW_clocked;
92 reg [4:0] FS_clocked;
93 reg [4:0] SH_clocked;
94 reg [31:0] BUS_A_clocked;
95 reg [31:0] BUS_B_clocked;
96
97 always @(*)
98 begin
99     BrA_reg = PC_M2_clocked + BUS_B_clocked;
100     RAA_reg = BUS_A_clocked;
101     VxorN_reg = V_reg ^ N_reg;
102     BS_one_reg = BS_clocked[1];
103     BS_zero_reg = BS_clocked[0];
104     RW_out_reg = RW_clocked;
105     DA_out_reg = DA_clocked;
106     MD_out_reg = MD_clocked;
107     PS_out_reg = PS_clocked;
108 end

```

```

109 always @(negedge CLOCK)
110 begin
111     PC_M2_clocked <= PC_M2;
112     RW_clocked <= RW;
113     DA_clocked <= DA;
114     MD_clocked <= MD;
115     BS_clocked <= BS;
116     PS_clocked <= PS;
117     MW_clocked <= MW;
118     FS_clocked <= FS;
119     SH_clocked <= SH;
120     BUS_A_clocked <= BUS_A;
121     BUS_B_clocked <= BUS_B;
122 end
123
124 Mod_Function_unit MFU(
125     .RESET(RESET),
126     .A(BUS_A_clocked),
127     .B(BUS_B_clocked),
128     .SH(SH_clocked),
129     .FS(FS_clocked),
130     .Z_out(Z_reg),
131     .C_out(C_reg),
132     .N_out(N_reg),
133     .V_out(V_reg),
134     .F(F_reg)
135 );
136
137 memDATA data_mem(
138     .clk(CLOCK),
139     .addr(BUS_A_clocked),
140     .MW(MW_clocked),
141     .datain(BUS_B_clocked),
142     .dataout(Data_Out_reg)
143 );
144
145 endmodule

```

EX_tb

```

21 module EX_tb();
22
23     //in
24     reg CLOCK;
25     reg RESET;
26     reg [31:0] PC_M2;
27     reg RW;
28     reg[4:0] DA;
29     reg[1:0] MD;
30     reg[1:0] BS;
31     reg PS;
32     reg MW;
33     reg[4:0] FS;
34     reg[4:0] SH;
35     reg [31:0] BUS_A;
36     reg [31:0] BUS_B;
37
38     //out
39     wire[31:0] BrA;
40     wire[31:0] RAA;
41     wire RW_out;
42     wire[4:0] DA_out;
43     wire[1:0] MD_out;
44     wire BS_one;
45     wire BS_zero;
46     wire PS_out;
47     wire Z;
48     wire V;
49     wire N;
50     wire C;
51     wire VxorN;
52     wire [31:0] F;
53     wire [31:0] Data_Out;
54

```

```

55 EX uut(
56
57     .CLOCK(CLOCK),
58     .RESET(RESET),
59     .PC_M2(PC_M2),
60     .RW(RW),
61     .DA(DA),
62     .MD(MD),
63     .BS(BS),
64     .PS(PS),
65     .MW(MW),
66     .FS(FS),
67     .SH(SH),
68     .BUS_A(BUS_A),
69     .BUS_B(BUS_B),
70
71     .BrA(BrA),
72     .RAA(RAA),
73     .RW_out(RW_out),
74     .DA_out(DA_out),
75     .MD_out(MD_out),
76     .BS_one(BS_one),
77     .BS_zero(BS_zero),
78     .PS_out(PS_out),
79     .Z(Z),
80     .V(V),
81     .N(N),
82     .C(C),
83     .VxorN(VxorN),
84     .F(F),
85     .Data_Out(Data_Out)
86 );

```

```

88 initial
89     begin
90         CLOCK = 0;
91         RESET = 0;
92         #10
93         RESET = 1;
94         #10
95         RESET = 0;
96         PC_M2 = 32'h00000001;
97         RW = 0;
98         DA = 4'b0000;
99         MD = 2'b00;
100        BS = 2'b00;
101        PS = 0;
102        MW = 0;
103        FS = 5'b00000;
104        SH = 5'b00000;
105        BUS_A = 32'h00000008;
106        BUS_B = 32'h00000008;
107
108        #100
109        #10
110        FS = 5'b00010;
111        SH = 5'b0100;
112
113
114        #10
115        FS = 5'b000101;
116        BUS_A = 32'h00000008;
117        BUS_B = 32'h000000FFF;
118        MW = 1;
119        #10
120        FS = 5'b01000;
121
122
123        #10
124        FS = 5'b01010;
125        BUS_A = 32'h00000008;
126        BUS_B = 32'h000000AAA;
127        MW = 0;
128        #10
129        FS = 5'b01100;

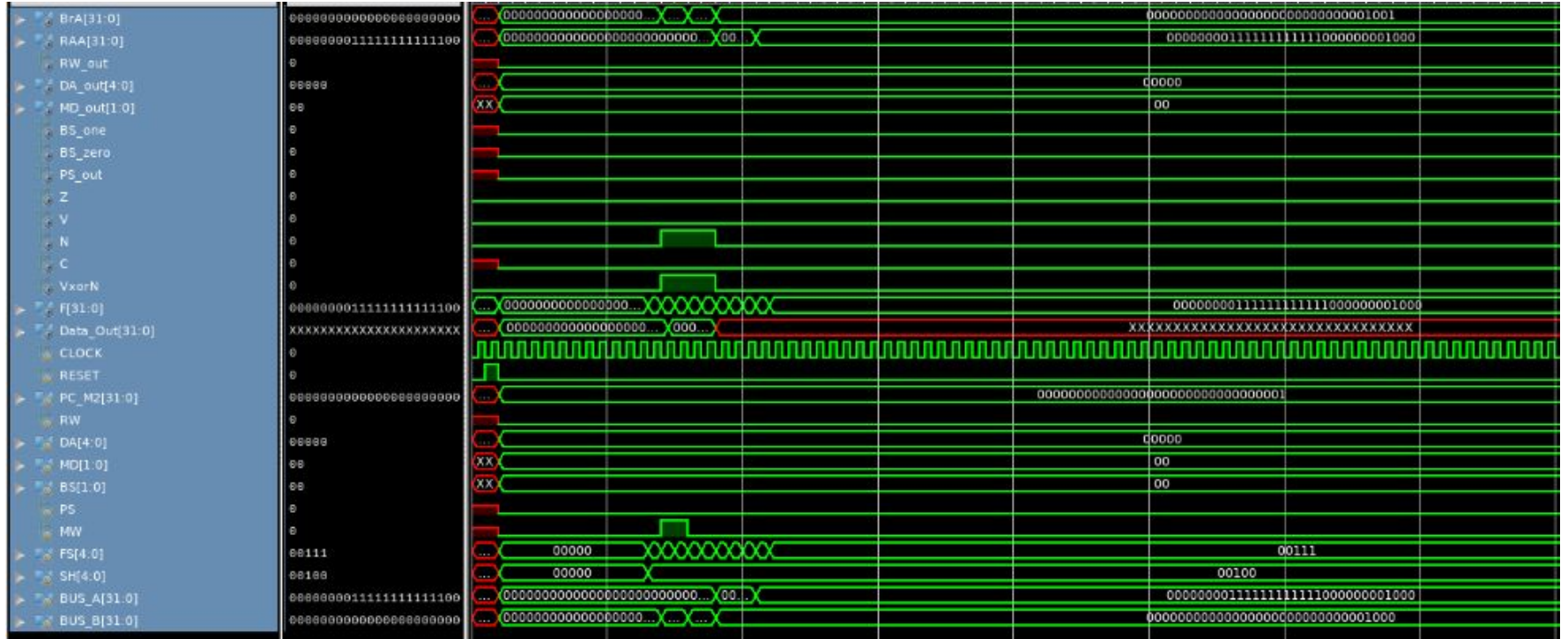
```

```

132        #10
133        FS = 5'b01110;
134        BUS_A = 32'h00002008;
135        BUS_B = 32'h00000008;
136
137        #10
138        FS = 5'b00000;
139
140        #10
141        FS = 5'b10000;
142
143
144        #10
145        FS = 5'b10001;
146
147
148        BUS_A = 32'h00FFF008;
149        BUS_B = 32'h00000008;
150        #10
151        FS = 5'b00111;
152
153    end
154
155
156    always
157    begin
158        #5
159        CLOCK = ~CLOCK;
160    end
161
162 endmodule

```

EX_tb SIM



WB

```

21 module WB(
22     input    CLOCK,
23     input    RESET,
24     input    RW,
25     input [4:0] DA,
26     input [1:0] MD,
27     input    VxorN,
28     input [31:0] F,
29     input [31:0] Data,
30     output [31:0] BUS_D,
31     output    RW_out,
32     output [4:0] DA_out
33 );
34
35 reg        RW_clocked;
36 reg [4:0]  DA_clocked;
37 reg [1:0]  MD_clocked;
38 reg [31:0] F_clocked;
39 reg [31:0] Data_clocked;
40 reg        VxorN_clocked;
41 reg [31:0] BUS_D_reg;
42
43 assign RW_out = RW_clocked;
44 assign DA_out = DA_clocked;
45 assign BUS_D = BUS_D_reg;
46
47 always @(negedge CLOCK)
48 begin
49     RW_clocked <= RW;
50     DA_clocked <= DA;
51     MD_clocked <= MD;
52     F_clocked  <= F;
53     Data_clocked <= Data;
54     VxorN_clocked <= VxorN;
55 end
56
57 always @(*)
58 begin
59     if(MD_clocked == 0)
60     begin
61         BUS_D_reg <= F_clocked;
62     end
63     if(MD_clocked == 1)
64     begin
65         BUS_D_reg <= Data_clocked;
66     end
67     if(MD_clocked == 2)
68     begin
69         BUS_D_reg <= {31'b00000000000000000000000000000000, VxorN};
70     end
71 end
72
73 endmodule
74
75

```

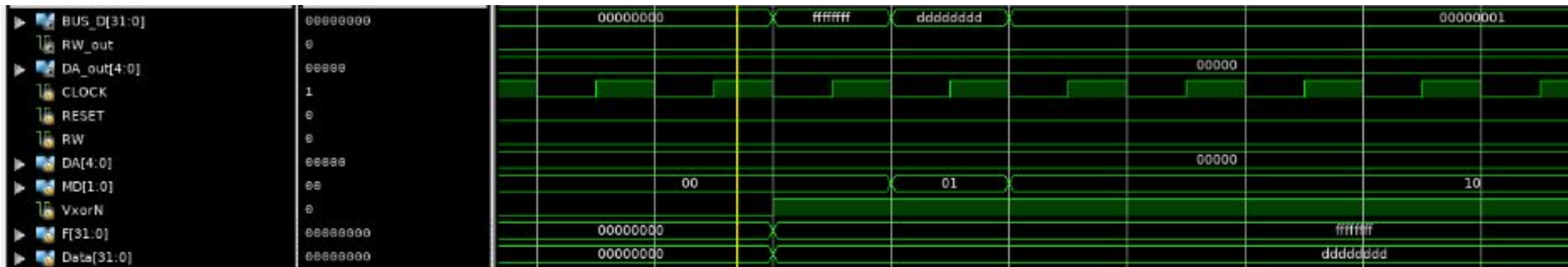
TestBench

```

23 reg CLOCK;
24 reg RESET;
25 reg RW;
26 reg [4:0] DA;
27 reg [1:0] MD;
28 reg VxorN;
29 reg [31:0] F;
30 reg [31:0] Data;
31
32 wire [31:0] BUS_D;
33 wire RW_out;
34 wire [4:0] DA_out;
35
36 WB uut (
37     .CLOCK(CLOCK),
38     .RESET(RESET),
39     .RW(RW),
40     .DA(DA),
41     .MD(MD),
42     .VxorN(VxorN),
43     .F(F),
44     .Data(Data),
45     .BUS_D(BUS_D),
46     .RW_out(RW_out),
47     .DA_out(DA_out)
48 );
49
50 initial
51 begin
52     CLOCK = 0;
53     RESET = 1;
54     #10
55     RESET = 0;
56     RW = 0;
57     DA = 5'b00000;
58     MD = 2'b00;
59     VxorN = 0;
60     F = 32'h00000000;
61     Data = 32'h00000000;
62     #100
63
64     F = 32'hFFFFFFF;
65     Data = 32'hDDDDDDDD;
66     VxorN = 1;
67     MD = 2'b00;
68     #10
69     MD = 2'b01;
70     #10
71     MD = 2'b10;
72 end
73
74

```

WB SIM



RISC_CPU

```

21 module RISC_CPU(
22     input CLOCK,
23     input RESET,
24     output V,
25     output N,
26     output C
27 );
28 //if to top
29 wire [31:0]PC_1_top_wire;
30 //top output to IF
31 wire [31:0]MUX_C_out_wire;
32 //IF output to DOF
33 wire [31:0]IR_wire;
34
35 //reg out to dof
36 wire [31:0]AOUT_wire;
37 wire [31:0]BOUT_wire;
38 //dof to reg
39 wire [4:0]AA_wire;
40 wire [4:0]BA_wire;
41 //dof to ex
42 wire [31:0]PC_M2_wire;
43 wire RW_dof_wire;
44 wire [4:0]DA_dof_wire;
45 wire [1:0]MD_dof_wire;
46 wire [1:0]BS_dof_wire;
47 wire PS_dof_wire;
48 wire MW_wire;
49 wire [4:0]FS_wire;
50 wire [4:0]SH_wire;
51 wire [31:0]BUS_A_wire;
52 wire [31:0]BUS_B_wire;
53 //ex to top
54 wire [31:0]BrA_wire;
55 wire [31:0]RAA_wire;
56 wire PS_wire;
57 wire Z_wire;
58 wire BS_one_wire;
59 wire BS_zero_wire;

```

```

72 TOP top(
73     //INPUT
74     .BS_one(BS_one_wire),//from ex
75     .BS_zero(BS_zero_wire),//from ex
76     .PS_top(PS_wire),//from ex
77     .Z_top(Z_wire),//from ex
78     .BrA_top(BrA_wire),//from ex
79     .RAA_top(RAA_wire),//from ex
80     .PC_1_top(PC_1_top_wire),//from IF
81     //OUTPUT
82     .MUX_C_out(MUX_C_out_wire)//to IF(PC)
83 );
84
85 IF IF(
86     //INPUT
87     .CLOCK(CLOCK),
88     .PC(MUX_C_out_wire),//from TOP(mux_c)
89     //OUTPUT
90     .PC_M1(PC_1_top_wire),//to top & DOF
91     .IR(IR_wire)//to dof
92 );

```

```

60 //ex to wb
61 wire RW_wire;
62 wire [4:0]DA_wire;
63 wire [1:0]MD_wire;
64 wire VxorN_wire;
65 wire [31:0]F_wire;
66 wire [31:0]Data_wire;
67 //wb to reg
68 wire RW_reg_wire;
69 wire [4:0]DA_reg_wire;
70 wire [31:0]BUS_D_wire;

```

```

94 DOF dof(
95     //input
96     .CLOCK(CLOCK),
97     .RESET(RESET),
98     .PC_M1(PC_1_top_wire),//from IF
99     .IR(IR_wire),//from IF
100     .A_DATA(AOUT_wire),//from regfile
101     .B_DATA(BOUT_wire),//from regfile
102     //output
103     .BUS_A(BUS_A_wire),
104     .BUS_B(BUS_B_wire),
105     .AA(AA_wire),//to regfile
106     .BA(BA_wire),//to regfile
107     .RW(RW_dof_wire),//to ex
108     .DA(DA_dof_wire),//to ex
109     .MD(MD_dof_wire),//to ex
110     .BS(BS_dof_wire),//to ex
111     .PS(PS_dof_wire),//to ex
112     .MW(MW_wire),//to ex
113     .FS(FS_wire),//to ex
114     .SH(SH_wire),//to ex
115     .PC_M2(PC_M2_wire)//to ex
116 );

```

```

151 WB wb(
152     //INPUT
153     .CLOCK(CLOCK),
154     .RESET(RESET),
155     .RW(RW_wire),//from ex
156     .DA(DA_wire),//from ex
157     .MD(MD_wire),//from ex
158     .VxorN(VxorN_wire),//from ex
159     .F(F_wire),//from ex
160     .Data(Data_wire),//from ex
161     //OUTPUT
162     .BUS_D(BUS_D_wire),//to reg
163     .RW_out(RW_reg_wire),//to reg
164     .DA_out(DA_reg_wire)//to reg
165 );

```

```

118 EX ex(
119     //INPUT
120     .CLOCK(CLOCK),
121     .RESET(RESET),
122     .PC_M2(PC_M2_wire),//from dof
123     .RW(RW_dof_wire),//from dof
124     .DA(DA_dof_wire),//from dof
125     .MD(MD_dof_wire),//from dof
126     .BS(VS_dof_wire),//from dof
127     .PS(PS_dof_wire),//from dof
128     .MW(MW_wire),//from dof
129     .FS(FS_wire),//from dof
130     .SH(SH_wire),//from dof
131     .BUS_A(BUS_A_wire),//from dof
132     .BUS_B(BUS_B_wire),//from dof
133     //OUTPUT
134     .BrA(BrA_wire),//to top
135     .RAA(RAA_wire),//to top
136     .RW_out(RW_wire),//to wb
137     .DA_out(DA_wire),//to wb
138     .MD_out(MD_wire),//to wb
139     .BS_one(BS_one_wire),//to top
140     .BS_zero(BS_zero_wire),//to top
141     .PS_out(PS_wire),//to top
142     .Z(Z_wire),//to top
143     .V(V),
144     .N(N),
145     .C(C),
146     .VxorN(VxorN_wire),//to wb
147     .F(F_wire),//to wb
148     .Data_Out(Data_wire)//to wb
149 );

```

```

167 regfile reg_file(
168     //INPUTS
169     .clock(CLOCK),
170     .reset(RESET),
171     .RW(RW_reg_wire),//from wb
172     .Asel(AA_wire),//from dof
173     .Bsel(BA_wire),//from dof
174     .Dsel(DA_reg_wire),//from wb
175     .DIN(BUS_D_wire),//from wb
176     //OUTPUTS
177     .AOUT(AOUT_wire),//to dof
178     .BOUT(BOUT_wire)//to dof
179 );
180 endmodule

```