DIGITAL DESIGN

LAB-5

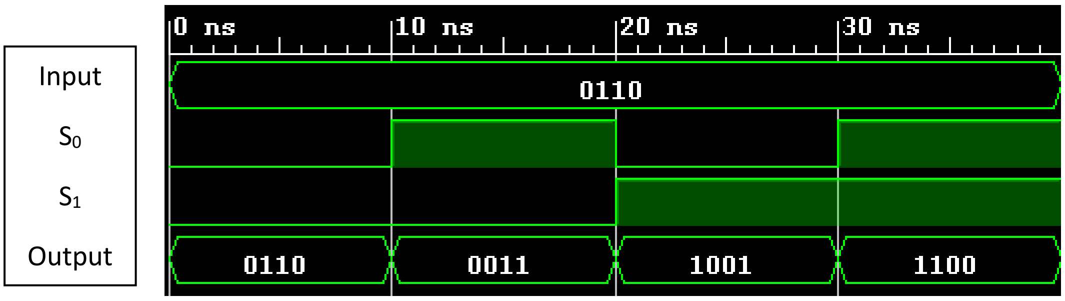
1. Barrel Shifter:

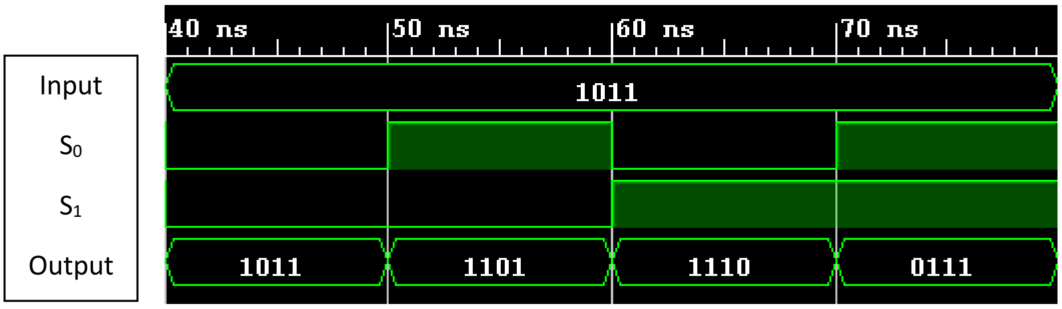
Logic Code:

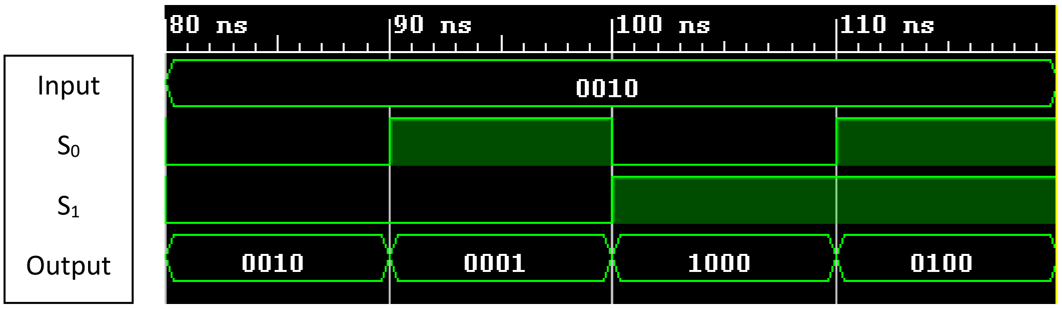
```
module mux(S0,S1,W0,W1,W2,W3,F);
input S0,S1,W0,W1,W2,W3;
output F;

assign F=S1?(S0?W3:W2):(S0?W1:W0);
endmodule


module barrel_shifter(S0,S1,W,Y);
input [3:0] W;
input S0,S1;
output [3:0] Y;

mux M1(S0,S1,W[3],W[0],W[1],W[2],Y[3]);
mux M2(S0,S1,W[2],W[3],W[0],W[1],Y[2]);
mux M3(S0,S1,W[1],W[2],W[3],W[0],Y[1]);
mux M4(S0,S1,W[0],W[1],W[2],W[3],Y[0]);
endmodule
```

Test Bench:

```
module test_barrel_shifter();
reg [3:0]W;
reg S0,S1;
wire [3:0]Y;
barrel_shifter BS(S0,S1,W,Y);

initial
begin
    W=4'd6;
    S0=1'b0;S1=1'b0;
    #10 S0=1'b1;S1=1'b0;
    #10 S0=1'b0;S1=1'b1;
    #10 S0=1'b1;
    #10 W=4'd11;
    S0=1'b0;S1=1'b0;
    #10 S0=1'b1;S1=1'b0;
    #10 S0=1'b0;S1=1'b1;
    #10 S0=1'b1;
    #10 W=4'd2;
    S0=1'b0;S1=1'b0;
    #10 S0=1'b1;S1=1'b0;
    #10 S0=1'b0;S1=1'b1;
    #10 S0=1'b1;
end
initial #120 $finish;
endmodule
```







2. ALU (32 bit):

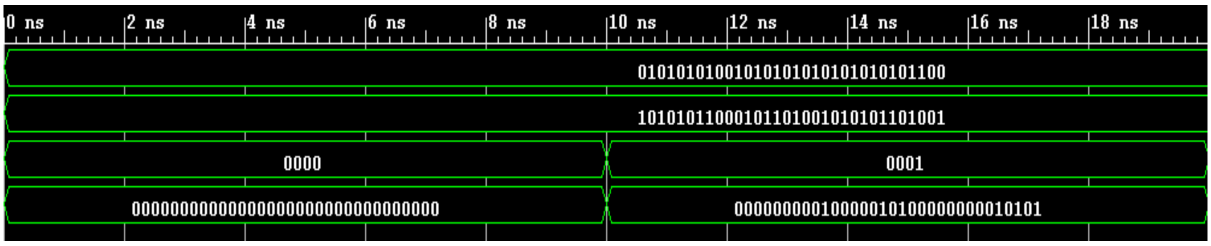
Logic Code:

```
module alu(A,B,S,O);
input [31:0] A,B;
input [3:0] S;
output reg [31:0]O;
always @(S)
    begin
        case(S)
            0: assign O=32'd0;
            1: assign O=A+B;
            2: assign O=A-B;
            3: assign O=A<<1;
            4: assign O=A>>1;
            5: assign O=A&B;
            6: assign O=A|B;
            7: assign O=A^B;
        endcase
    end
endmodule
```

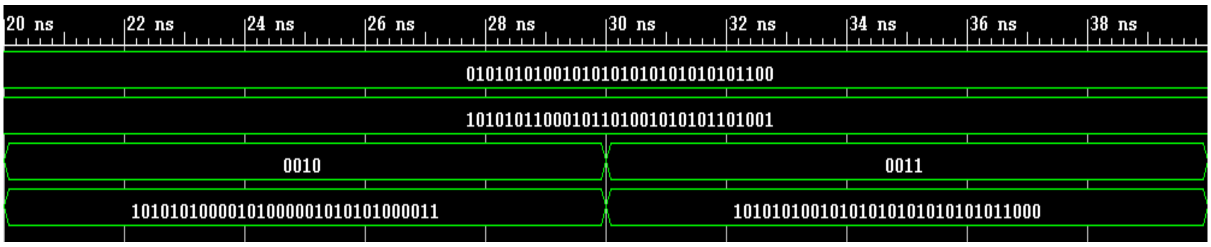
Test Bench:

```
module test_alu();
reg [31:0] A,B;
reg [3:0] S;
wire [31:0]O;
alu ALU(A,B,S,O);

initial
begin
    A=32'b01010101001010101010101010101100;
    B=32'b10101011000101101001010101101001;
    S=3'd0;
    #10 S=3'd1;
    #10 S=3'd2;
    #10 S=3'd3;
    #10 S=3'd4;
    #10 S=3'd5;
    #10 S=3'd6;
    #10 S=3'd7;
end
initial #80 $finish;
endmodule
```

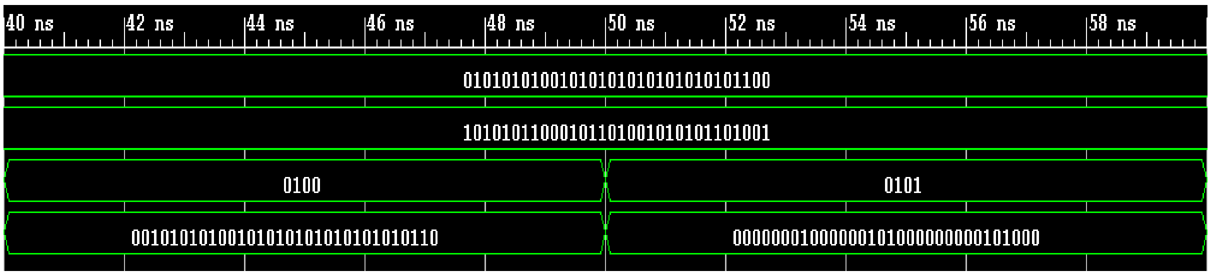
Operation	Input	32 bit Output
Clear	000	0
Addition	001	A + B
Subtraction	010	A - B
A * 2	011	Left shift
A / 2	100	Right Shift
A AND B	101	A & B
A OR B	110	A | B
A XOR B	111	A^B



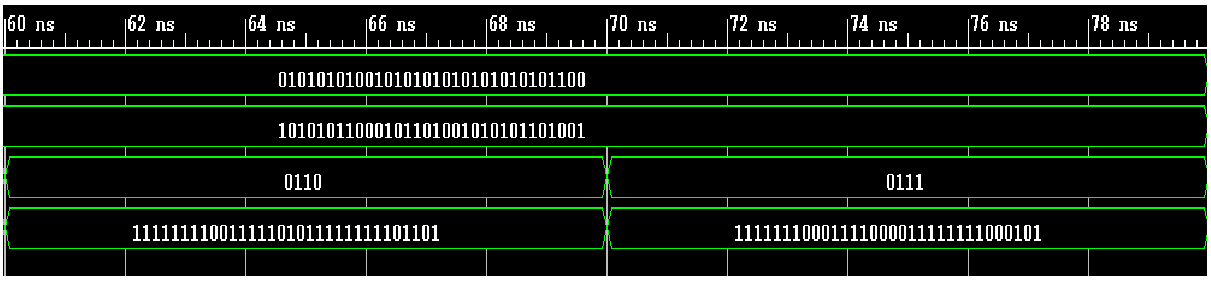
```
01010101001010101010101010101100
10101011000101101001010101101001
0000                              0001
00000000000000000000000000000000  00000000010000010100000000010101
```



```
01010101001010101010101010101100
10101011000101101001010101101001
0010                              0011
10101010000101000001010101000011  10101010010101010101010101011000
```



```
01010101001010101010101010101100
10101011000101101001010101101001
0100                              0101
00101010100101010101010101010110  00000001000000101000000000101000
```



```
01010101001010101010101010101100
10101011000101101001010101101001
0110                              0111
11111111001111101011111111101101  11111110001111000011111111000101
```

3. Priority Encoder:

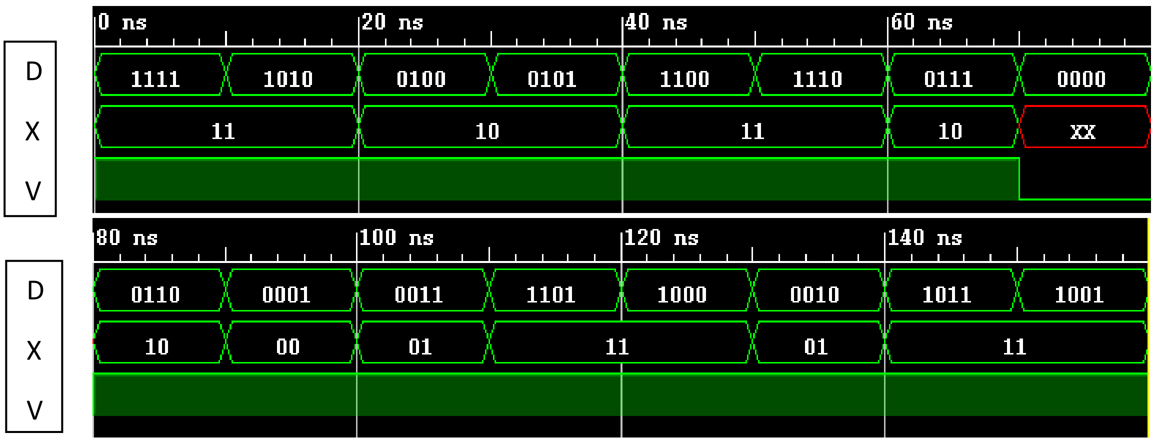
a. Casex Statement:

Logic Code:

```
module priority_encoder_a(D,X,V);
input [3:0] D;
output reg [1:0] X;
output reg V;

always @(D)
    begin
    V=1'b1;
        casex(D)
            4'b1xxx : X=2'b11;
            4'b01xx : X=2'b10;
            4'b001x : X=2'b01;
            4'b0001 : X=2'b00;
            default : begin
                        V=1'b0;
                        X=2'bXX;
                      end
        endcase
    end
endmodule
```

Test Bench:

```
module test_priority_encoder_a();
reg [3:0] D;
wire [1:0] X;
wire V;
priority_encoder_a PEA(D,X,V);

initial
begin
    D=4'd15;
    #10 D=4'd10;
    #10 D=4'd4;
    #10 D=4'd5;
    #10 D=4'd12;
    #10 D=4'd14;
    #10 D=4'd7;
    #10 D=4'd0;
    #10 D=4'd6;
    #10 D=4'd1;
    #10 D=4'd3;
    #10 D=4'd13;
    #10 D=4'd8;
    #10 D=4'd2;
    #10 D=4'd11;
    #10 D=4'd9;
end
initial #160 $finish;
endmodule
```



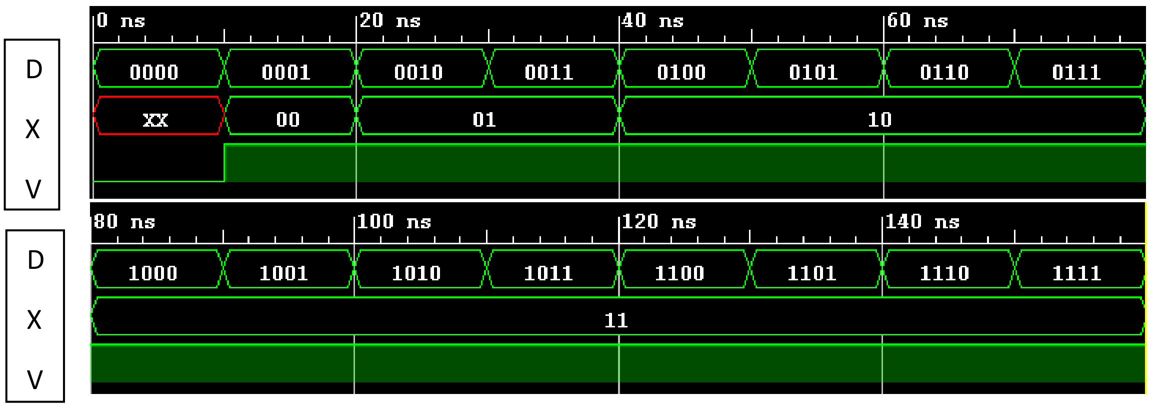
b. For loop:

Logic Code:

```
module priority_encoder_b(D,X,V);
input [3:0] D;
output reg [1:0] X;
output reg V;

integer k;
always @(D)
begin
    X=2'bxx;
    V=0;
    for (k=0;k<4;k=k+1)
    if(D[k])
    begin
        X=k;
        V=1;
    end
end
endmodule
```

Test Bench:

```
module test_priority_encoder_b();
reg [3:0] D;
wire [1:0] X;
wire V;
priority_encoder_b PEB(D,X,V);

initial
begin
    D=4'd0;
    #10 D=4'd1;
    #10 D=4'd2;
    #10 D=4'd3;
    #10 D=4'd4;
    #10 D=4'd5;
    #10 D=4'd6;
    #10 D=4'd7;
    #10 D=4'd8;
    #10 D=4'd9;
    #10 D=4'd10;
    #10 D=4'd11;
    #10 D=4'd12;
    #10 D=4'd13;
    #10 D=4'd14;
    #10 D=4'd15;
end
initial #160 $finish;
endmodule
```



4.

a. BCD Adder/Subtractor:

Logic Code:

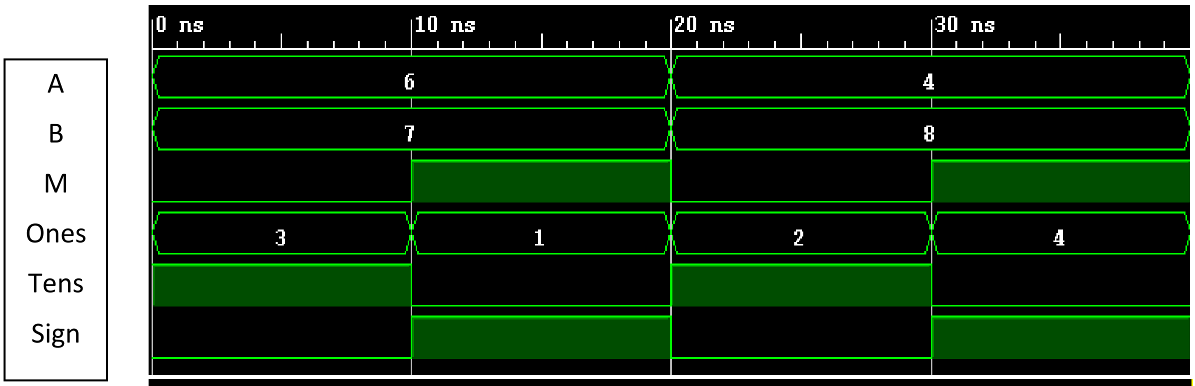
```
module bcd_adder_subtractor(A,B,M,X,Y,S);
input [3:0] A,B;
input M;
reg [4:0] Z;
output reg [3:0] X;
output reg Y,S;

always @(M or A or B)
begin
    Y=0;
    S=0;
    if(M)
    begin
        Z=A-B;
        X=Z[3:0];
        if(Z[4])
        begin
            X=16-X;
            S=1;
        end
    end
    else
    begin
        Z=A+B;
        X=Z[3:0];
        Y=Z[4];
        if (X>9 || Y)
        begin
            X=X+6;
            Y=1;
        end
    end
end
endmodule
```

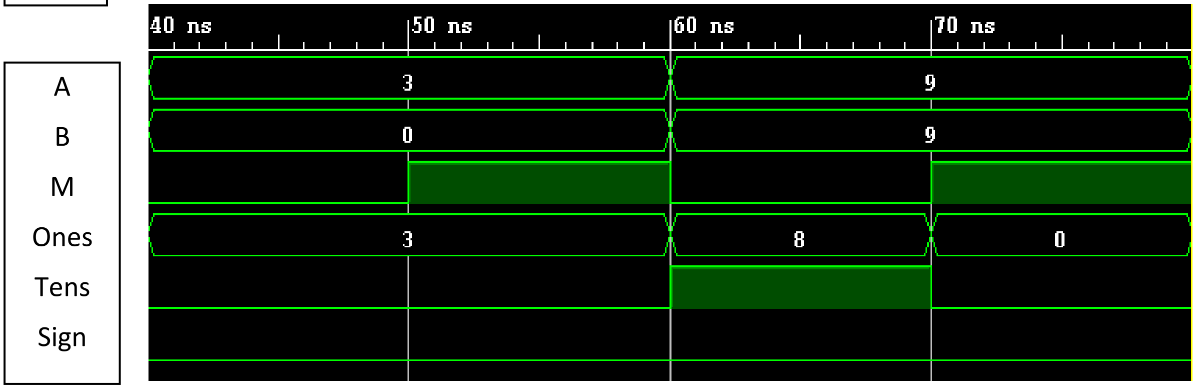
Test Bench:

```
module test_bcd_adder_subtractor();
reg [3:0] A,B;
reg M;
wire [3:0] X;
wire Y,S;
bcd_adder_subtractor BAS(A,B,M,X,Y,S);

initial
begin
     A=4'd6;B=4'd7;M=1'b0;
     #10 M=1'b1;
     #10 A=4'd4;B=4'd8;M=1'b0;
     #10 M=1'b1;
     #10 A=4'd3;B=4'd0;M=1'b0;
     #10 M=1'b1;
     #10 A=4'd9;B=4'd9;M=1'b0;
     #10 M=1'b1;
end
initial #80 $finish;
endmodule
```

M: 0-Adder, 1-Subtractor

Sign: 0-Positive, 1-Negative





b. Multiply by 5:

Logic Code:

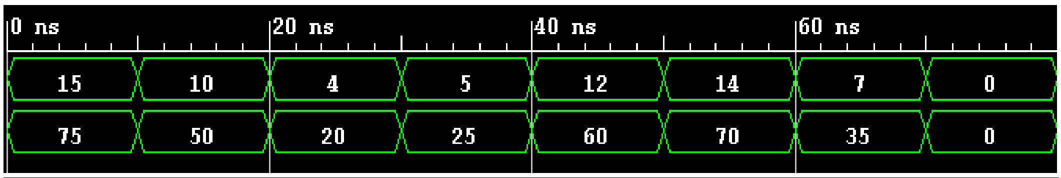
```
module multiply5(A,X);
input [3:0] A;
output [6:0] X;

assign X=(A<<2) + A;
endmodule
```

Test Bench:

```
module test_multiply5();
reg [3:0] A;
wire [6:0] X;
multiply5 M(A,X);

initial
begin
     A=4'd15;
     #10 A=4'd10;
     #10 A=4'd4;
     #10 A=4'd5;
     #10 A=4'd12;
     #10 A=4'd14;
     #10 A=4'd7;
     #10 A=4'd0;
     #10 A=4'd6;
     #10 A=4'd1;
     #10 A=4'd3;
     #10 A=4'd13;
     #10 A=4'd8;
     #10 A=4'd2;
     #10 A=4'd11;
     #10 A=4'd9;
end
initial #160 $finish;
endmodule
```

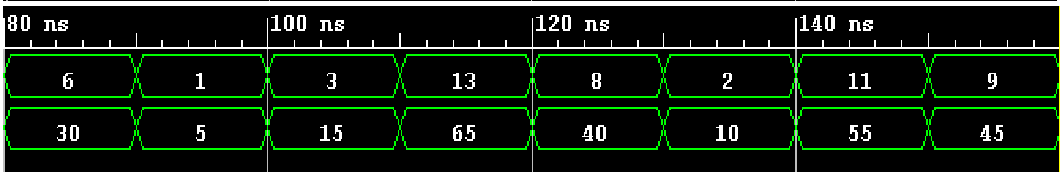
Number

Product



Number

Product


80 ns
100 ns
120 ns
140 ns
6 1 3 13 8 2 11 9
30 5 15 65 40 10 55 45