- 3>2>1>0

```
module number2( w,y);
  input[3:0]w;
  output reg [1:0]y;
  always @(w)
  casex (w)
    4'b1xxx: y=3;
    4'b01xx: y=2;
    4'b001x: y=1;
    4'b0001: y=0;
  endcase
endmodule
```

1>2>0>3

```
module number2( w,y);
  input[3:0]w;
  output reg [1:0]y;
  always @(w)
  casex (w)
    4'bxx1x: y=1;
    4'bx10x: y=2;
    4'bx001: y=0;
    4'b1000: y=3;
    default :  y=2'bxx;
  endcase
endmodule
```

- Full adder

```
module fulladd(S,Cout,A,B,Cin);
input A, B, Cin;
output S, Cout;
assign S = A ^ B ^ Cin;
assign Cout = (A & B) | (Cin & (A ^ B));
endmodule
```

Another approach(Full adder):

```
module fadd (S, A, B, Cin);
input Cin;
input [3:0] A,B;
output [4:0] S;
assign S = A+B+Cin;
endmodule
```

- Mux using procedural (if else) 2x1

```
module mux2to1(w, S, f);
    input S;
    input [1:0]w;
    output reg f;

    always @(w, S)  // always @(*)
    begin
            if(S == 0)
                f = w[0];
            else
                f = w[1];
    end
endmodule
```

Mux using procedural (case) 2x1

```
module number2(w, s, f);
  input [1:0] w;
  input s;
  output reg f;
  always @(w or s)
    case(s)
      1'b0:f =w[0];
      1'b1:f =w[1];

    endcase
endmodule
```

—----------

- 4 to 1 mux (CASE)

S 2 bit 00 01 10 11

```
module number2(w, s, f);
  input [3:0] w;
  input [1:0]s;
  output reg f;
  always @(w , s)
    case(s)
      0: f=w[0];
      1: f=w[1];
      2:f= w[2];
      3:f= w[3];
      default : f=1'bx;
```

```
    endcase
Endmodule
```

4 to 1 mux (if else)

```
module number2(w, s, f);
    input [3:0] w;
    input [1:0] s;
    output reg f;

    always @(w, s) begin
        if (s == 2'b00)
            f = w[0];
        else if (s == 2'b01)
            f = w[1];
        else if (s == 2'b10)
            f = w[2];
        else if (s == 2'b11)
            f = w[3];
        else
            f = 1'bx;  // Just in case s has an unknown value
    end
endmodule
```

- blocking

```
module number2(d,clock,q1,q2);
  input d,clock;
  output q1,q2;
  reg q1,q2;

  always @(posedge clock)
  begin
    q1=d;
    q2=q1;

  end
endmodule
```

Non blocking

```
module number2(d,clock,q1,q2);
  input d,clock;
  output q1,q2;
  reg q1,q2;

  always @(posedge clock)
  begin
    q1<=d;
    q2<=q1;

  end
endmodule
```

……

4-to-1 multiplexer with an extra control signal pin. If pin = 1, disable the selector and always output w[0]. If pin = 0, work as a normal 4x1 mux using select lines s

```
module number2(w, s, pin, f);
    input [3:0] w;
    input [1:0] s;
    input pin;
    output reg f;

    always @(w, s, pin) begin
        if (pin == 1) begin
            f = w[0];
        end else begin
            case (s)
                2'b00: f = w[0];
                2'b01: f = w[1];
                2'b10: f = w[2];
                2'b11: f = w[3];
                default: f = 1'bx;
            endcase
        end
    end
endmodule
```

……

……

((a or b) (a’ and c)) xor d

```
module number1(a,b,c,d,f1,f2,f3,f4,f5);
```

```
  input a,b,c,d;
  output f1,f2,f3,f4,f5;
  assign f1= a|b;
  assign f2= ~a;
  assign f3= f2 & c;
  assign f4= f1 & f3;
  assign f5= f4 ^d;
  endmodule
```

Same using Structural Representation:

```
module expt2(f,x1,x2,x3,x4);
input x1,x2,x3,x4;
output f;
and (w1,x1,x3);
and (w2,x2,x4);
or (g,w1,w2);
or (w3,x1,~x3);
or (w4,x4,~x2);
and (h,w3,w4);
or (f, g, h);
endmodule
```

Same using vector

```
module number2(a,f);
  input [3:0] a;
  output [4:0] f;
  assign f[0] =a[0] | a[1];
  assign f[1] = ~a[0];
  assign f[2] =f[1] & a[2];
  assign f[3] =f[0] & f[2];
  assign f[4] =f[3] ^ a[3];
  endmodule
```

,,,

```
module lab_assesment3(d,load,clk,q);
  input[3:0]d;
  input load,clk;
  output reg [3:0]q;
always @(posedge clk)
  if (load)
                    q<=d;
  else
          begin
    q[3]<=q[0];
    q[2]<=q[3];
    q[1]<=q[2];
    q[0]<=q[1];
  end
endmodule
```