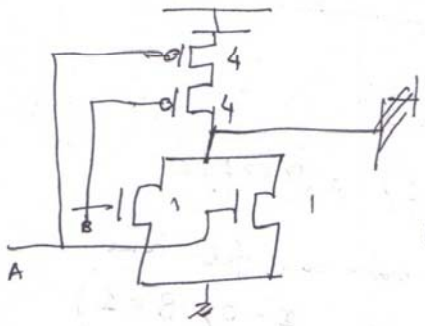
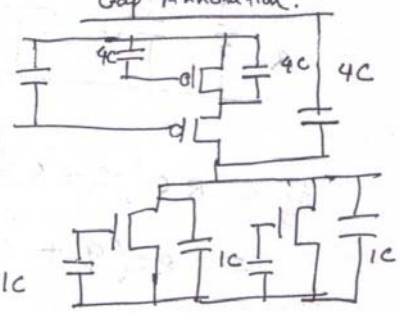
1. The MOS switch operating modes are cutoff and Saturation.

2. OR gate.



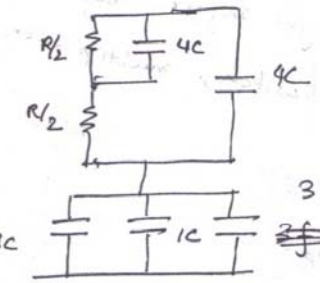
Gap Annotation.

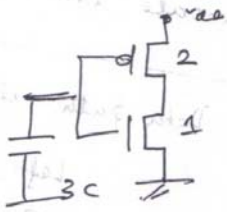


Sizing : Pull up resistance $\propto .2\left(\frac{2}{4}\right) = 1R$.

Pull down resistance $\propto 1R$.

Worst case
Rise time / Fall time



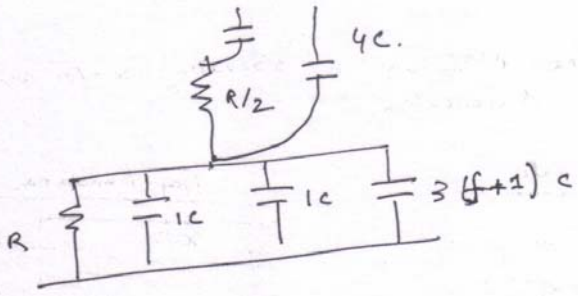


$$t_f = \frac{R}{2}\left(\frac{2}{4}C\right) + R(4C + 2C + 3f + 3)C$$

$$= R(2C + 4C + 2C + 3fC + 3C)$$

$$= R(11C + 3fC) = (11 + 3f)\tau.$$

Fall



$t_f = Rc(2 + 4 + 3f + 3)$

$= Rc(9 + 3f) = (9+3f)\tau.$

(while considering the worst case fall time, we have to consider that A = 0, B = 1)

Bifurcate the net in steps of 3/4.

3 a) Data path elements:

Left shifter, Right Shifter, Comparator, Subtractor, switch, multiplexer.

b) gcd(in: u, v, out: z)

register XR, YR;

XR = u;

YR = v;

```
while (XR != YR) do begin.
 if (XR[0] = 0 ∧ YR[0] = 0)
    {  XR = leftshift (XR);
       YR = leftshift (YR);
       count = count + 1;
    }
else if (XR[0] = 1 ∧ YR[0] = 0)
    {
       YR = leftshift (YR);
else if (YR[0] = 1 ∧ XR[0] = 0)
       XR = leftshift (XR);
 else if (YR[0] = 1 ∧ XR[0] = 1)
    { if (XR ≥ YR)
        {  XR = XR - YR;
           XR = leftshift(XR);
        }
      else
        { YR = YR - XR;
          YR = leftshift (YR);
        }
    }.
end
```

S_1 S_2 S_3 S_4.

$Z = 2^{count} (XR)$.

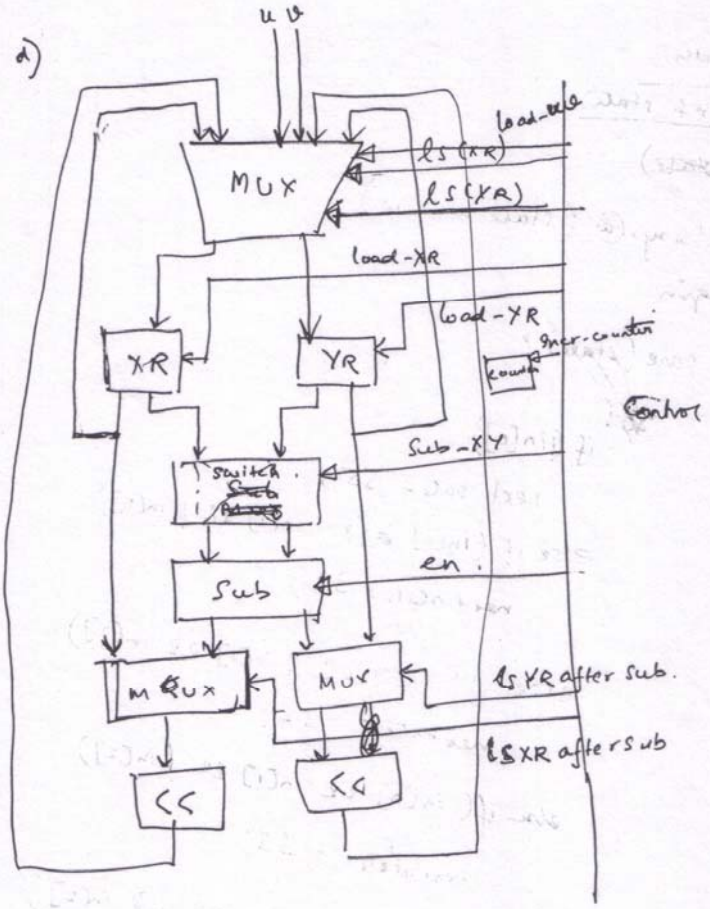
ⓒ

Inputs: Input 1: (XR != YR) | Input 2: XR[0] | Input 3: YR[0] | Input 4: XR ≥ YR

	0_	100_	101_	110_	111_	ls(XR)	ls(YR)	load_XR	load_YR	Sub (join counter)	XY	load_XR after sub	load_YR after sub
S_0	S_5	S_1	S_2	S_3	S_4	0	0	1	1	1	0	0	0
S_1						1	1	0	1	1	1	0	0
S_2						0	1	0	0	1	0	0	0
S_3						1	0	0	1	0	0	0	0
S_4						XR>YR 1	0	0	1	0	0	1	0
						XR<YR 0	1	0	0	1	0	0	1
S_5	S_5	S_5	S_5	S_5	S_5	0	0	0	0	0	0	0	0

State m/c is a mealy machine,

d)



e) left shifter:

```
assign a = a<<1;
```

right shifter

```
assign a = a>>1;
```

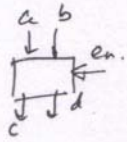
Comparator

```
if (a>b)
    out = 0;
else
    out = 1;
```

Subtractor:

```
assign out = a-b;
```

Switch:




```
assign c = (en) ? a : b;
assign d = (en) ? b : a;
```

mux:

```
assign out = (en) ? a : b;
```

Control path

// next state

~~case (state)~~

always @ (state or in)

begin

case (state)

~~S0~~:

if (!in[0])

next state = S5;

else if (in[0] && ! in[1] && ! in[2])

next state = S1;

else if(in[0] && ! in[1] && in[2])

next state = S2;

else if(in[0] && in[1] && ! in[2])

next state = S3;

else if (in[0] && in[1] && in[2])

next state = S4;

~~S5~~

if (state == S5);

next state = S5;

end

```
// output logic
always @ (~~next~~ state or in)
begin
    ~~if~~(
    case (state)
        S0: begin LSXR = 0;
            LSYR = 0;
            loadKLV = 0;
            loadXR = 1;
            loadYR = 1;
            incr counter = 0;
            subXY = 0;
            load ~~XR~~ XR after sub = 0;
            load YR after sub = 0;
        end
        S1: begin LSYR = 1;
            LSYR = 1;
            load uu = 0;
            load XR = 1;
            load YR = 1;
            incr counter = 1;
            sub XY = 0;
            load XR after sub = 0;
            load YR after sub = 0;
        end
        S2: begin LSXR = 0;
            ~~load~~ LSYR = 1;
            load uu = 0;
            load XR = 0;
            load YR = 1
            incr counter = 0;
            sub XY = 0;
            load XR after sub = 0;
            load YR after sub = 0;
        end
```

```
S3: begin
    lsXR = 1;
    lsYR = 0;
    loadUV = 0;
    loadXR = 1;
    loadYR = 0;
    incrcounter = 0;
    subXR/Y = 0;
    loadXRafter sub = 0;
    loadYR after sub = 0;
end

S4: begin
    if (in[4])
        lsXR = 1;
        lsYR = 0;
        loadUV = 0;
        loadXR = 1;
        loadYR = 0;
        incrcounter = 0;
        subXR = 0;
        loadXR after sub = 1;
        loadYR after sub = 0;
    else
        lsXR = 0;
        lsYR = 1;
        loadUV = 0;
        loadXR = 0;
        loadYR = 1;
        incrcounter = 0;
        subXR = 0;
        loadXR after sub = 0;
        loadYR after sub = 1;
    end

S5: begin
    lsXR = 0;
    lsYR = 0;
    loadUV = 0;
    loadXR = 0;
    loadYR = 0;
    incrcounter = 0;
    subXY = 0;
    loadXR after sub = 0;
    loadYR after sub = 0;
end
endcase
end
// next state logic.

always@(posedge a posedge clk)
begin
    if rst
        state <= 3'b0;
    else
        state <= nextstate;
end
```

(4)

$$S[7] = a[7] \oplus b[7] \oplus c_{in}[7]$$

$$\Rightarrow c_{in}[7] = S[7] \oplus a[7] \oplus b[7].$$

$$\therefore c_{out}[7] = maj(a[7], b[7], c_{in}[7])$$

$$\boxed{c_{out}[7] = a[7]\,b[7] + (a[7] + b[7])(S[7] \oplus a[7] \oplus b[7])}$$

---

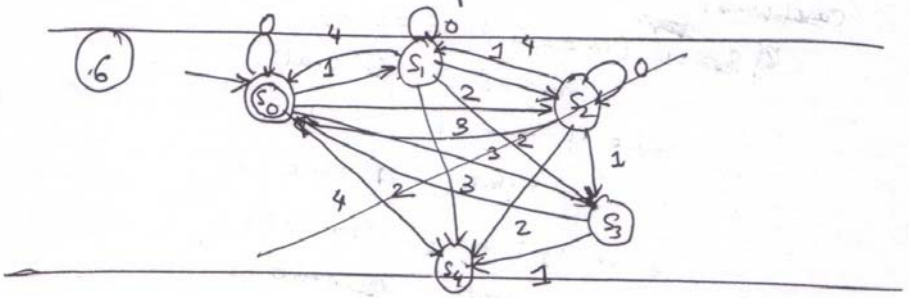
(5) (a) Yes. // priority encoders //

(b) None

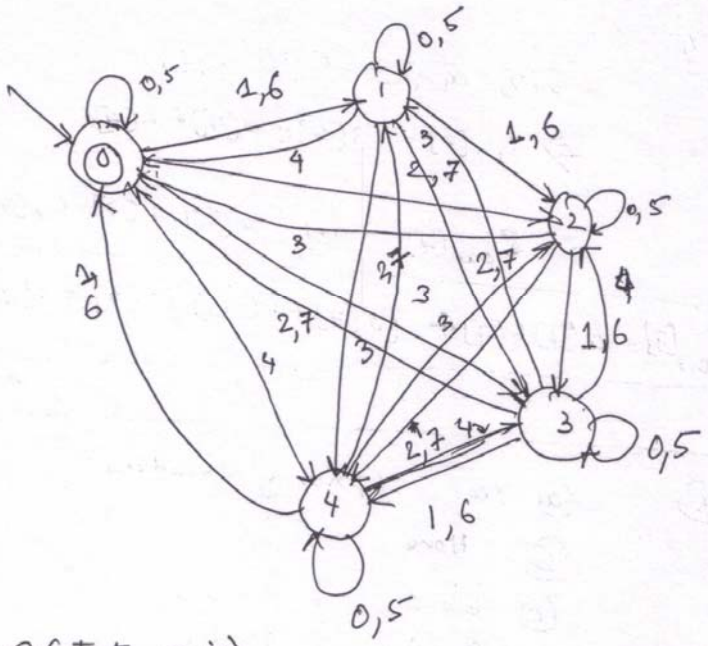
(c)

b = 1;	b = 1;
a = 0;	a = 0;
c = x;	c = x;

(6)



(6)




```
always@(state or in)
begin
case(state)
    if (in == 0 || in == 5)
        nstate = state;
    else if (in == 1 || in == 6)
        nstate = state + 1;
    else if (in == 2 || in == 7)
        nstate = state + 2;
    else if (in == 3)
        nstate = state + 3;
    else if (in == 4)
        nstate = state + 4;
    if (state == 5 || state == 6 || state == 7)
        nstate = 3'b0;
end
```

(7) (a) (ii) does not simulate when en changes

(i) responds to en having 0/1 Logic value, but (ii) responds to en = X also.

(b) (i) b is evaluated at time t=0 and ~~assigned at~~ assigned at t=1 to a.

a is evaluated at time t=1 and assigned at t=3 to c.

(ii) b is evaluated and assigned to a at time t=1 and

a is evaluated and assigned to c at time t=3.