PAGE NO. :
DATE : __/__/__

Week #3:

1. Write short notes on controller & datapath of single purpose processor with block diagram.

Ans Datapath:

It stores and manipulates a system's data of different form. Data is a binary number representing external conditions like temperature, speed or digitized photographic image. Datapath contains register units, functional units and connection units. It can be configured to read data from particular register feed that data through functional units, configured to carry out particular operations like add or shift & store the operation results back into particular register.

Controller:

It carries out configuration of data path. It sets the datapath control inputs like register load and MUX selects signals of the register units, functional units and connection units to obtain the desired configuration at a particular time. Controller monitors external control inputs as well as datapath control outputs ~~also~~ known as status signals, coming from functional units and sets external control outputs as well.

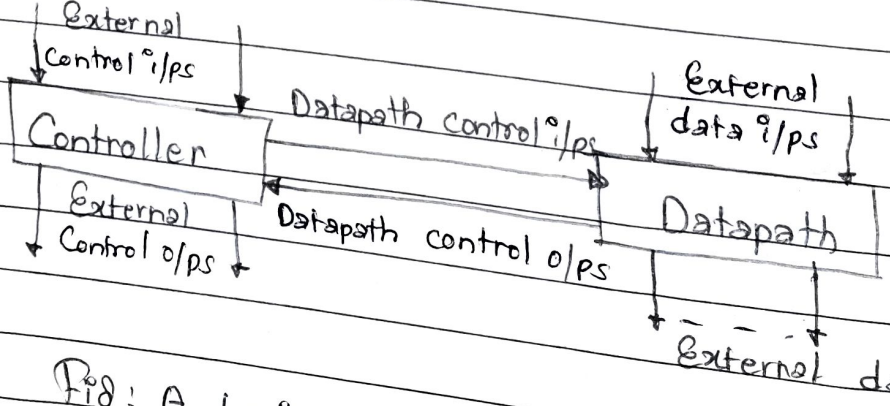


Fig: A basic processor: Controller & Datapath.

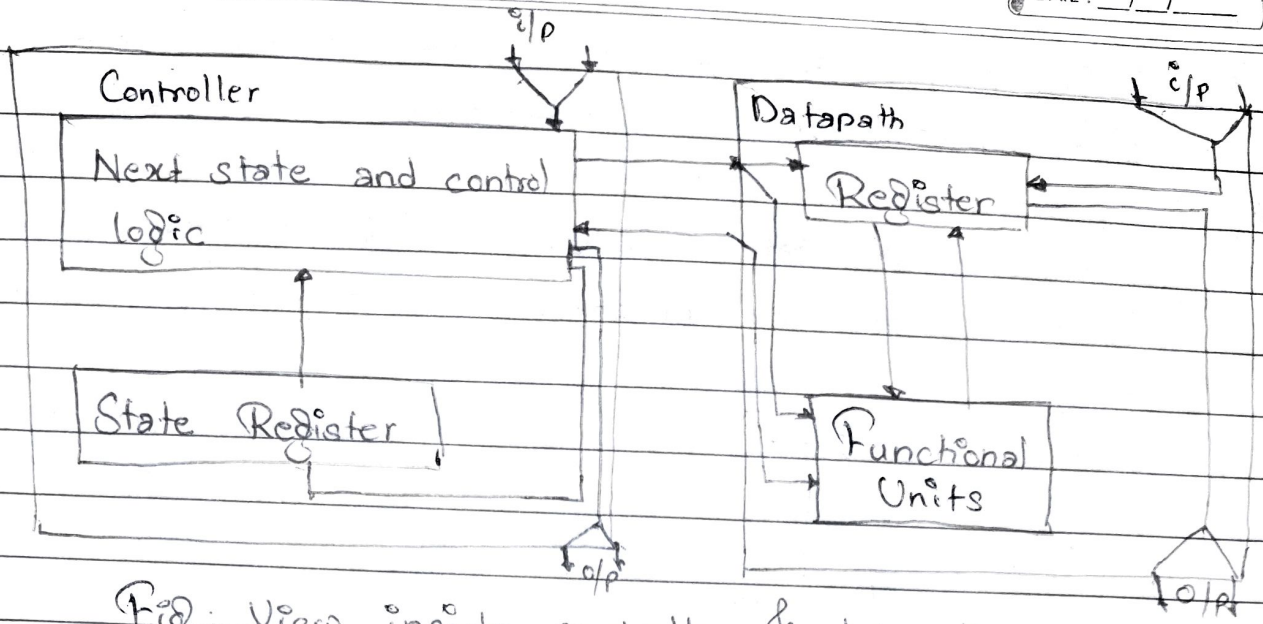


Fig: View inside controller & datapath.

(2) State complete design steps for single-purpose processor.

Ans: Steps :

i) Draw Black Box / Block Diagram
- Simple box with external interfaces of the system
- Includes input, output & control signal.

ii) Write a functionality or program
- is code to provide solution to the problem.
- Input temporary variables, and output signals are assigned to variables.

iii) Design Finite State Machine with Data (FSMD)
- Code in step (ii) are converted to equivalent complex state diagram

iv) Build corresponding datapath
- Register functional units, connections
- Control input/output signals.

v) Develop Finite State Machine (FSM).

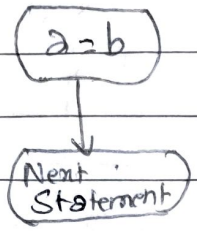
(3) Write short notes on assignment statement, branch statement and loop statement with templates.

Ans: i) Assignment statement:

Assignment statement is used to assign some values to a variable or perform some sequential operation.

Example:

a=b



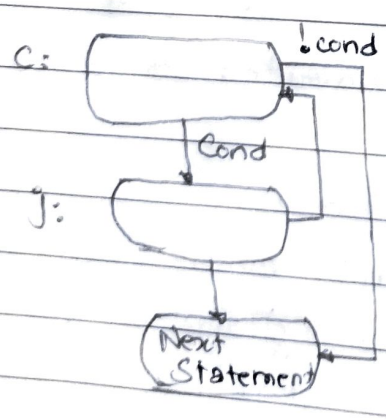
Template for Assignment statement

ii) Loop statements:

Loop statements are statement that are carried out for a number of times.

Example:

```
while (cond)
{
    loop-body-statement ();
}
next statement
```



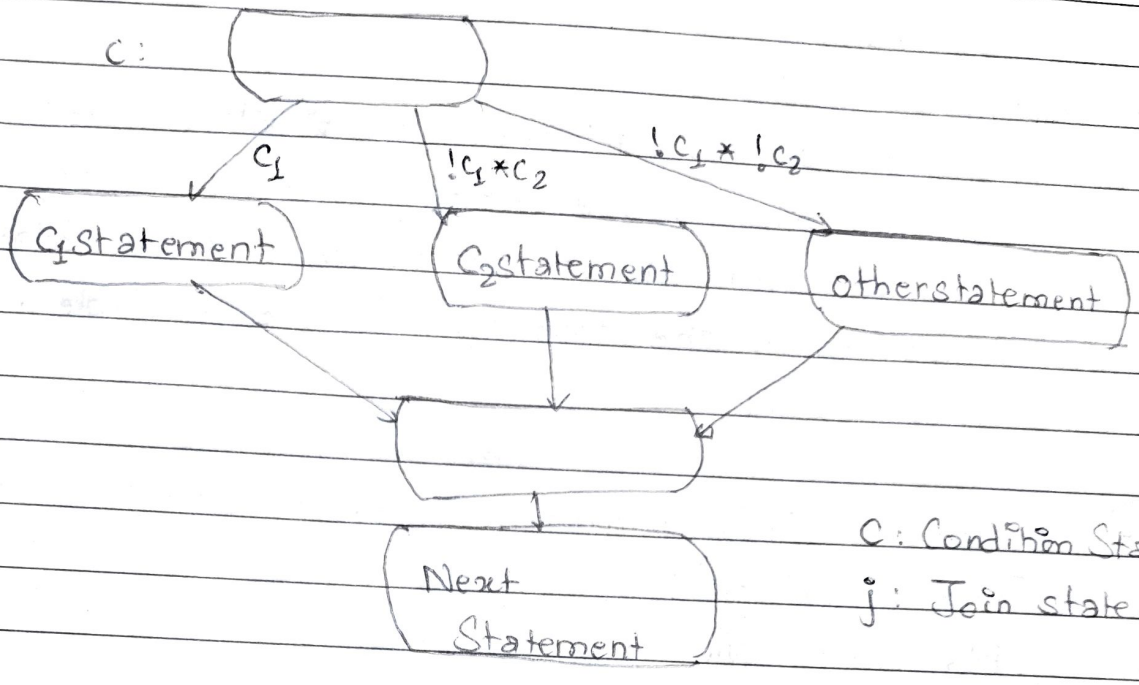
iii) Branch statement:

Example: if (c_1)

c_1 statement

else if c_2

C_2 statement
else
other statement
next statement



C: Condition State
j: Join state.

5) Write brief notes on optimization of single purpose processor in different design steps such as functionality coding, FSMD, Datapath and FSM.

Ans Optimization is the technique of improving the design metrics so as to get the best possible values of various design metrics. The optimization opportunities in Single purpose processor are:

1. Optimization of original program
   The algorithms are analyzed in terms of time complexity and space complexity, and hence we try to develop more efficient alternative algorithms. It involves decreasing of number of computation and size of variables if possible.
2. Optimizing FSMD
   The states that can be merged to reduce the number of states. The design must be aware if whether

o/p timining may or may not be modified.

3. Optimizing Datapath

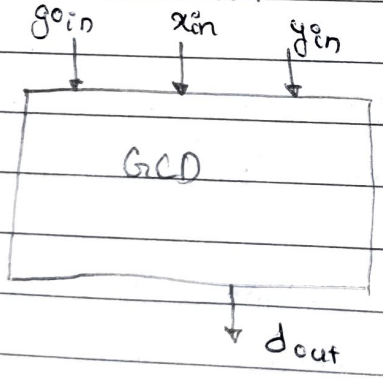
Many functional operations can share a single functional unit. if those operations to occur in different stages.

4. Optimizing FSM:

FSM can be optimized using state encoding and state minimization state. Encoding is the task of assigning a unique $\log_2 n$ bits to encode n states. State minimization is the taste of merging equivalent states into a single state.

(4) Design a single purpose processor to calculate GCD (Greatest Common Divisor):

Ans: i) Black box view:



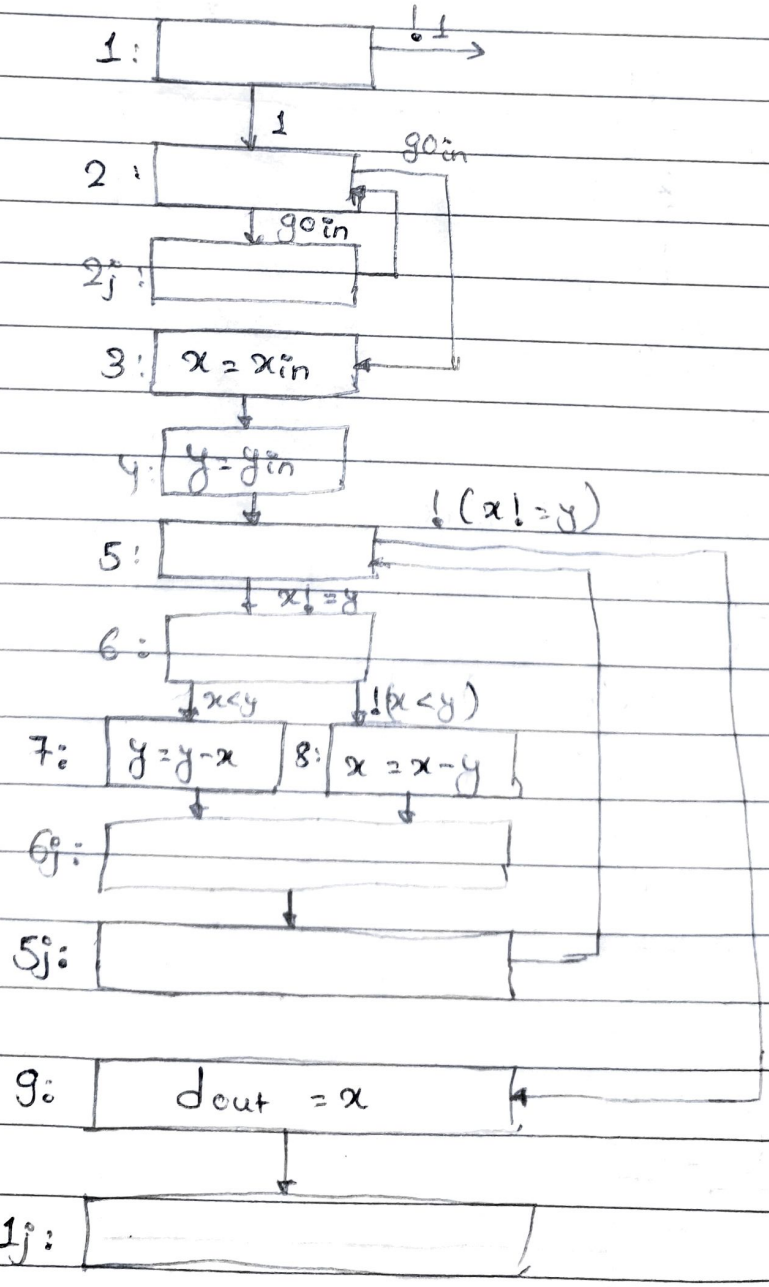
Data input: x_{in}, y_{in}
Data output: d_{out}
Control signals: go_{in}

ii) Functionality code:

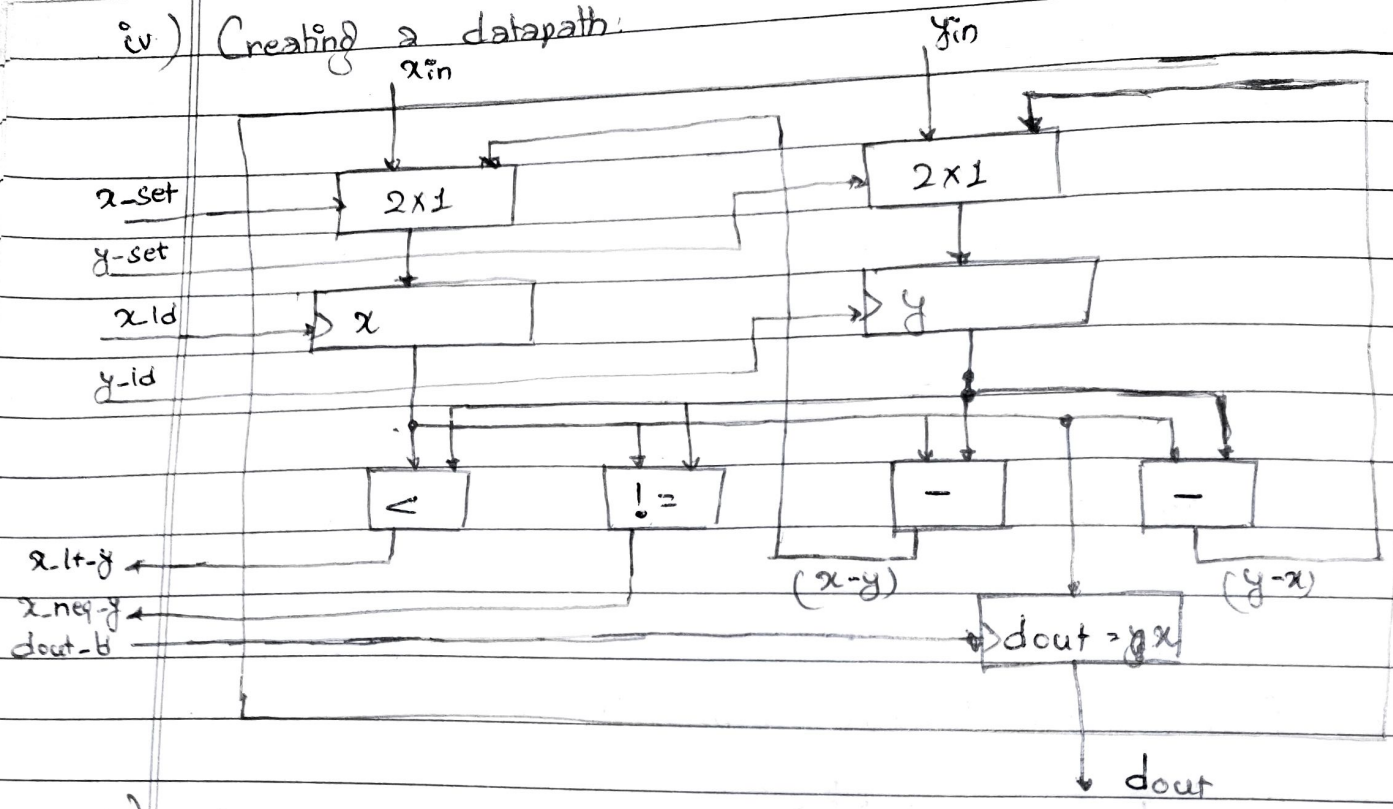
```
int x, y;
while (1)
{
    while (!go_in);
    x = x_in;
```

```
y = yin;
while (x != y)
{
    if (x<y)
        y = y-x;
    else
        x = x-y;
}
dout = x;
}
```

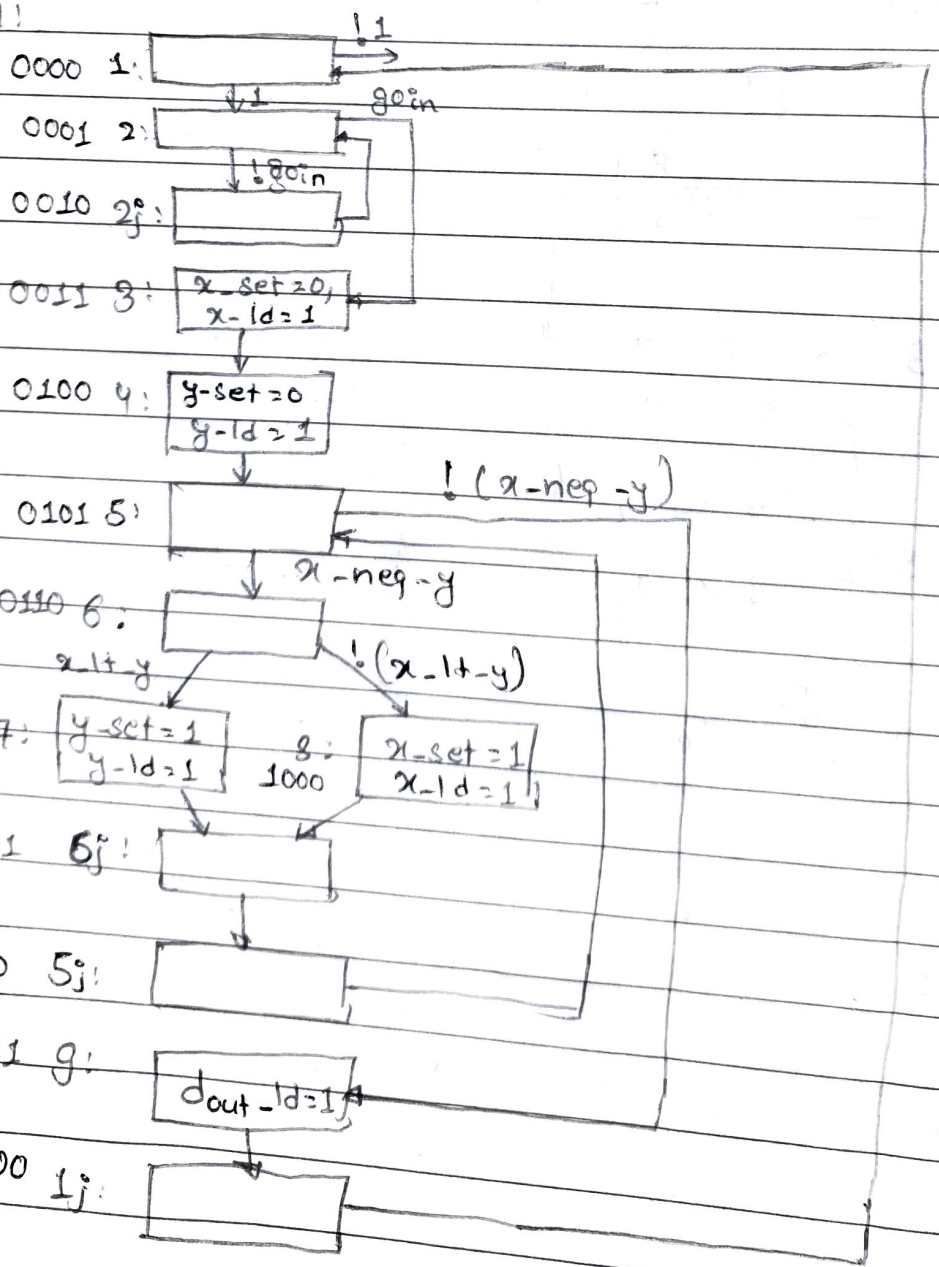
iii) Finite State Machine with Data



iv) Creating a datapath:



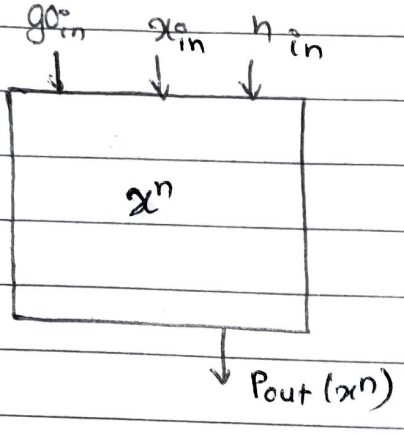
v) FSM:



GOOD MORNING
PAGE NO. :
DATE : __/__/____

(8) Design a single processor that calculates x to the power n : x^n. The design includes black box view, functionality code, FSMD, FSM and implementation block diagram.

1) Black box

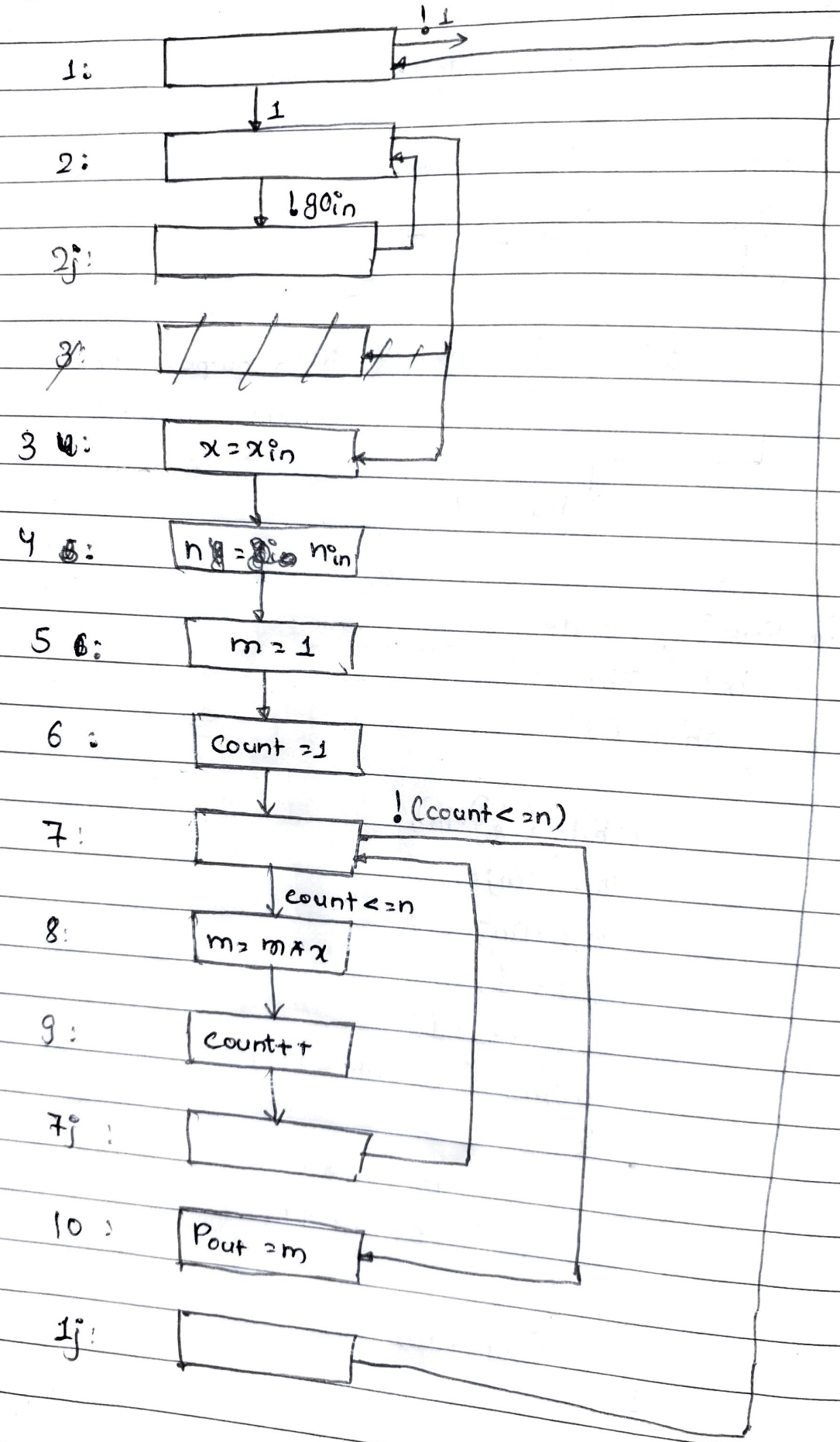


Data input: x_{in}, n_{in}
Data output: Pout
Control signals: go_{in}

2) Functionality code:

```
int xin, nin;
while (1)
{
    while (!goin);
    x = xin;
    n = nin;
    m = 1;
    count = 1;
    while (count <= n)
    {
        m = m * x;
        count ++;
    }
    Pout = m;
}
```

3) FSM.


1:
!1
1
2:
!go in
2j:
3:
3j:
x = x in
4:
n = n in
5:
m = 1
6:
count = 1
7:
!(count <= n)
count <= n
8:
m = m * x
9:
count++
7j:
10:
P out = m
1j:

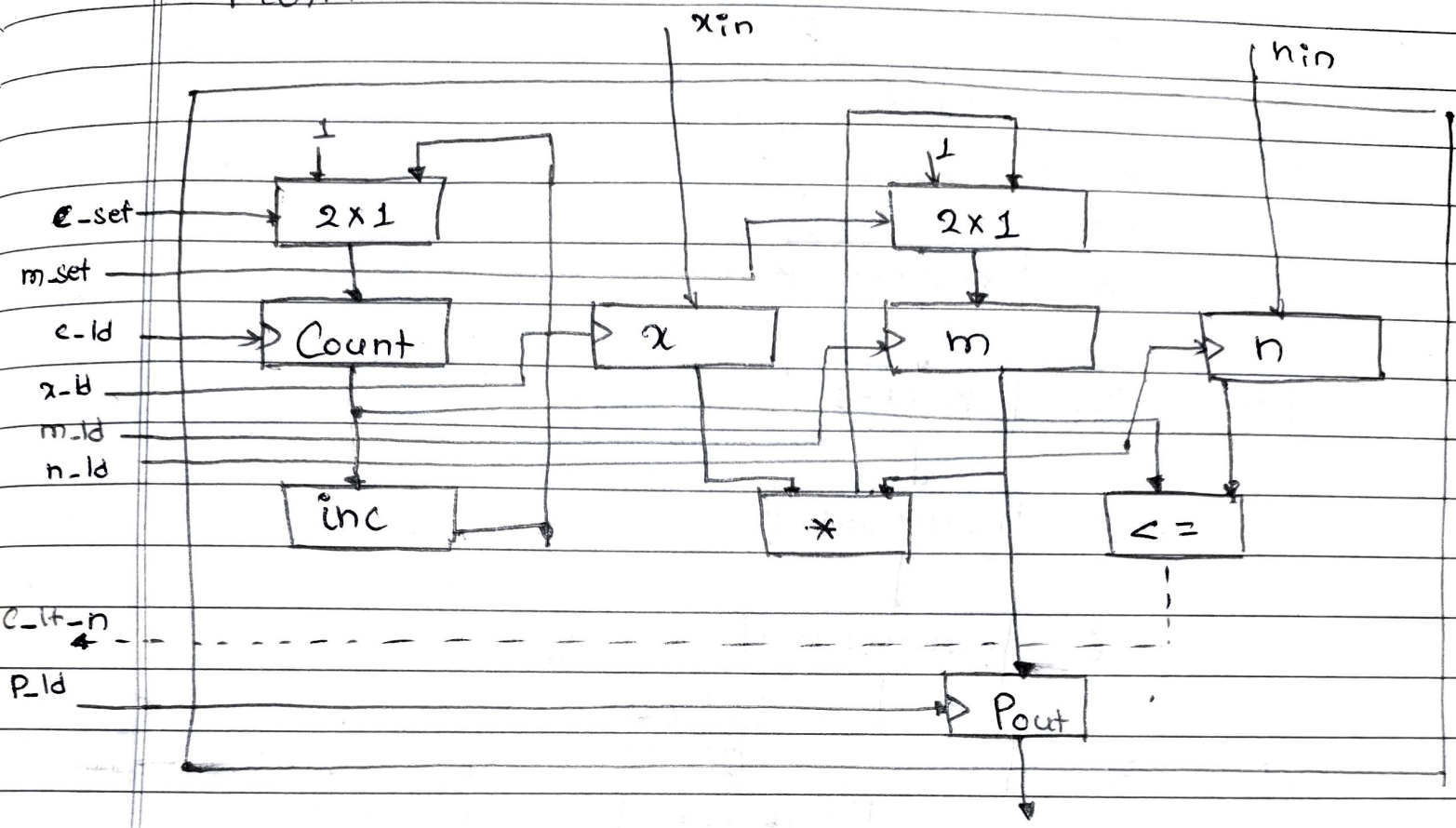
DATE : __/__/____

4) Datapath: ~~mode~~

Registers : x, n, m, count

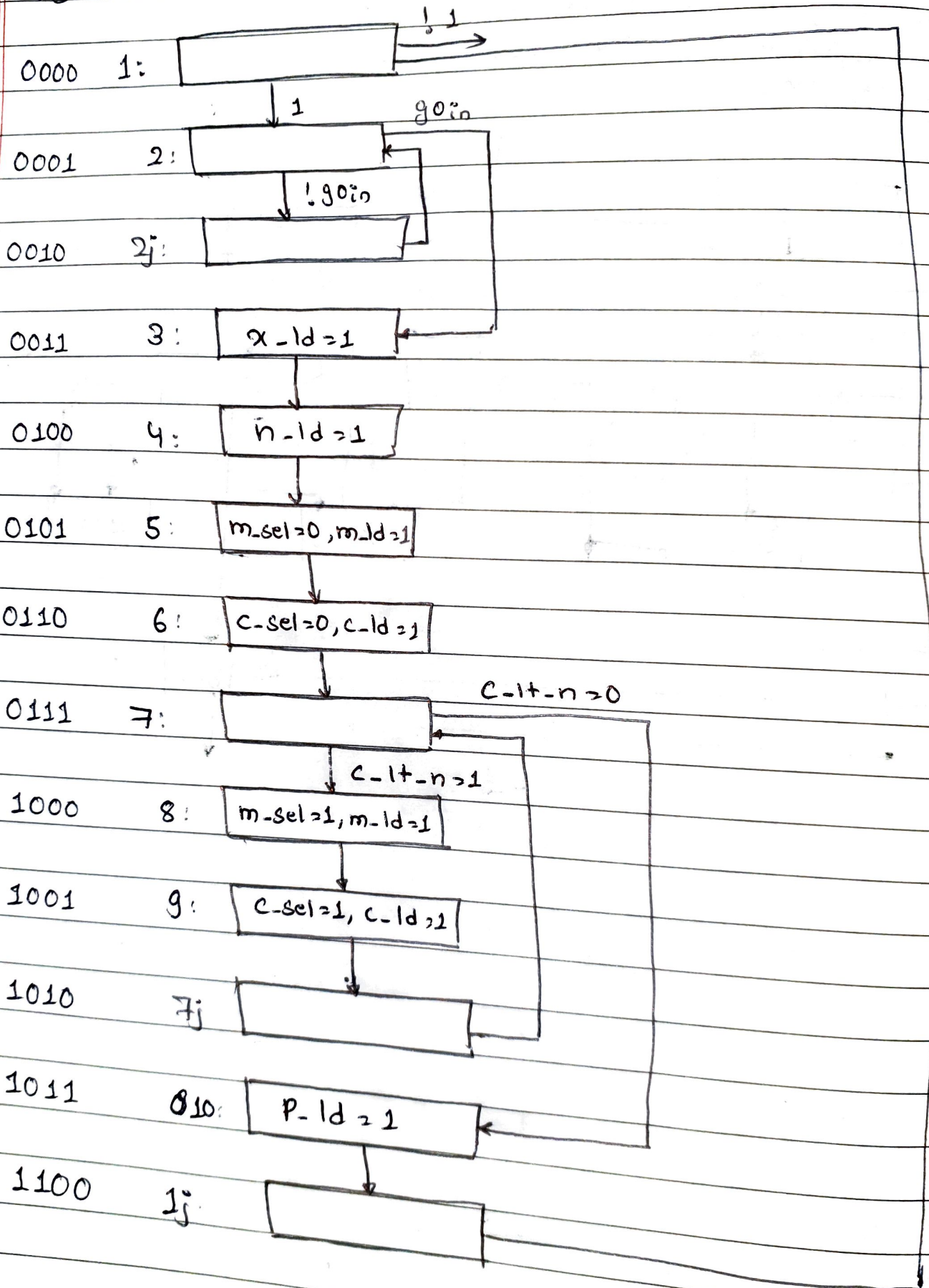
Operators : *, inc, <=

MUX :

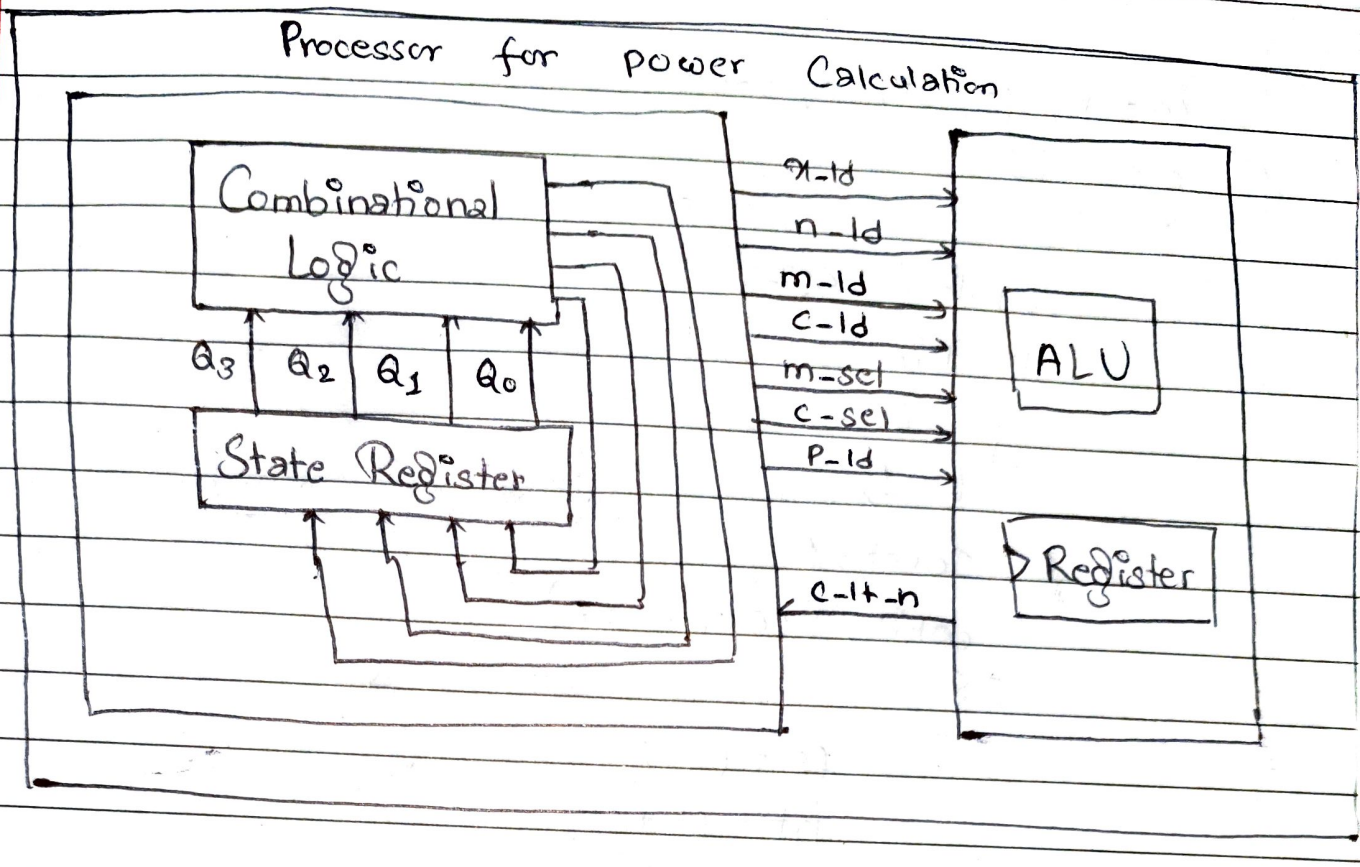


DATE : __/__/__

5) FSMD:

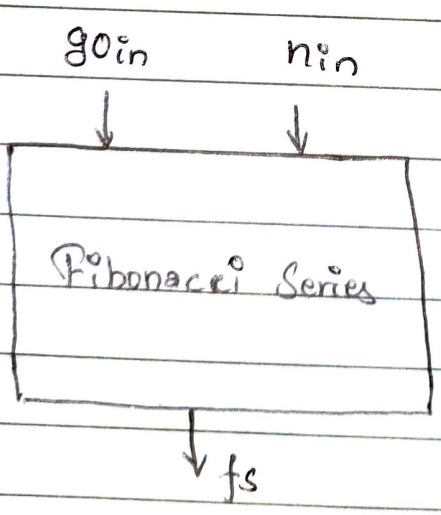

0000 1:
!go_in
1
go_in
0001 2:
!go_in
0010 2j:
0011 3: x_ld=1
0100 4: n_ld=1
0101 5: m_sel=0, m_ld=1
0110 6: c_sel=0, c_ld=1
0111 7:
c_lt_n=0
c_lt_n=1
1000 8: m_sel=1, m_ld=1
1001 9: c_sel=1, c_ld=1
1010 7j
1011 810: P_ld=1
1100 1j

6) Implementation Block Diagram:



9) Design a single processor that generates Fibonacci series:

i) Black Box View



Black Box View

Data input : nin

Data output: fs

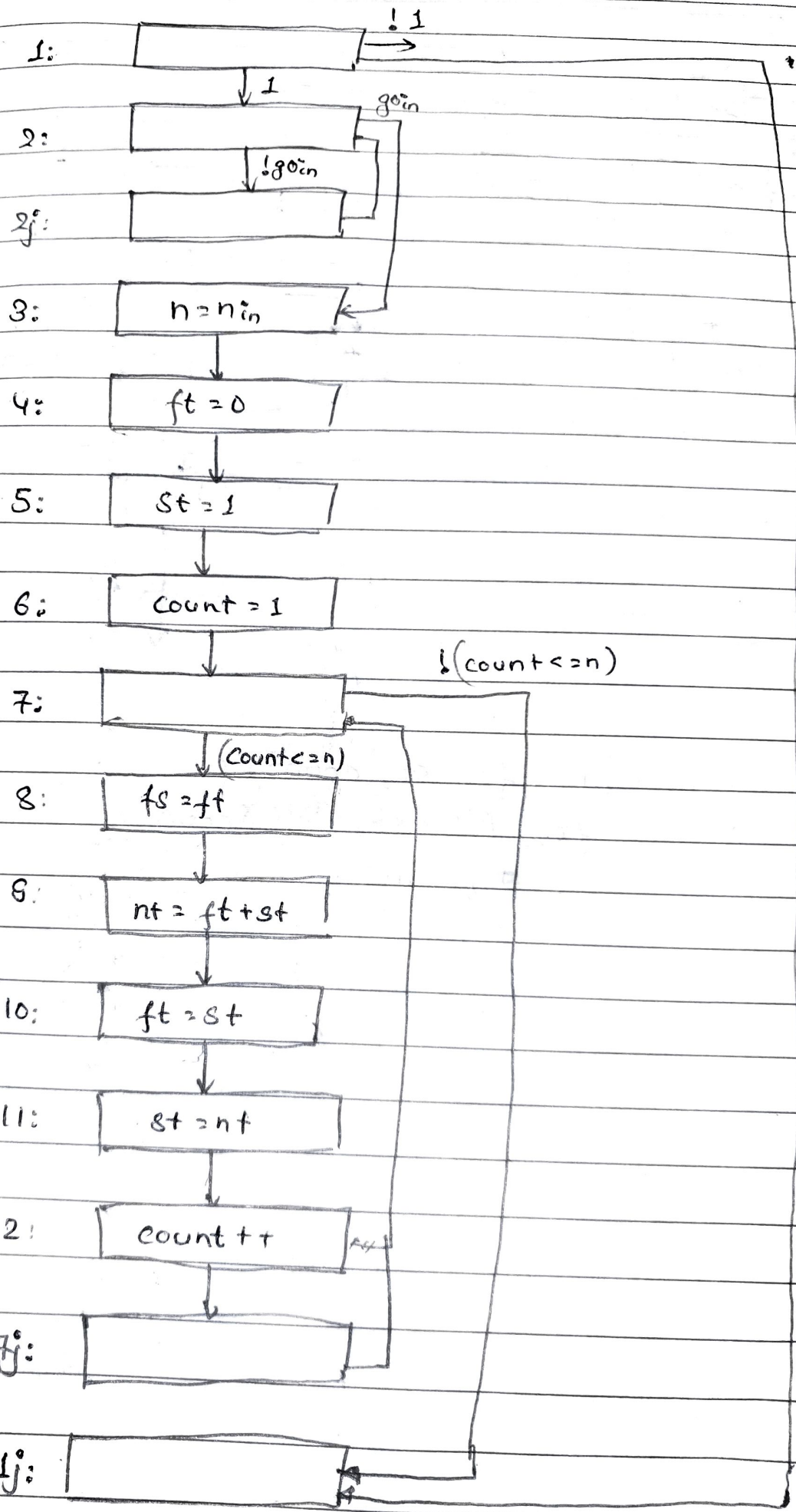
Signal : goin.

GOOD MORNING
PAGE NO. :
DATE : __/__/____

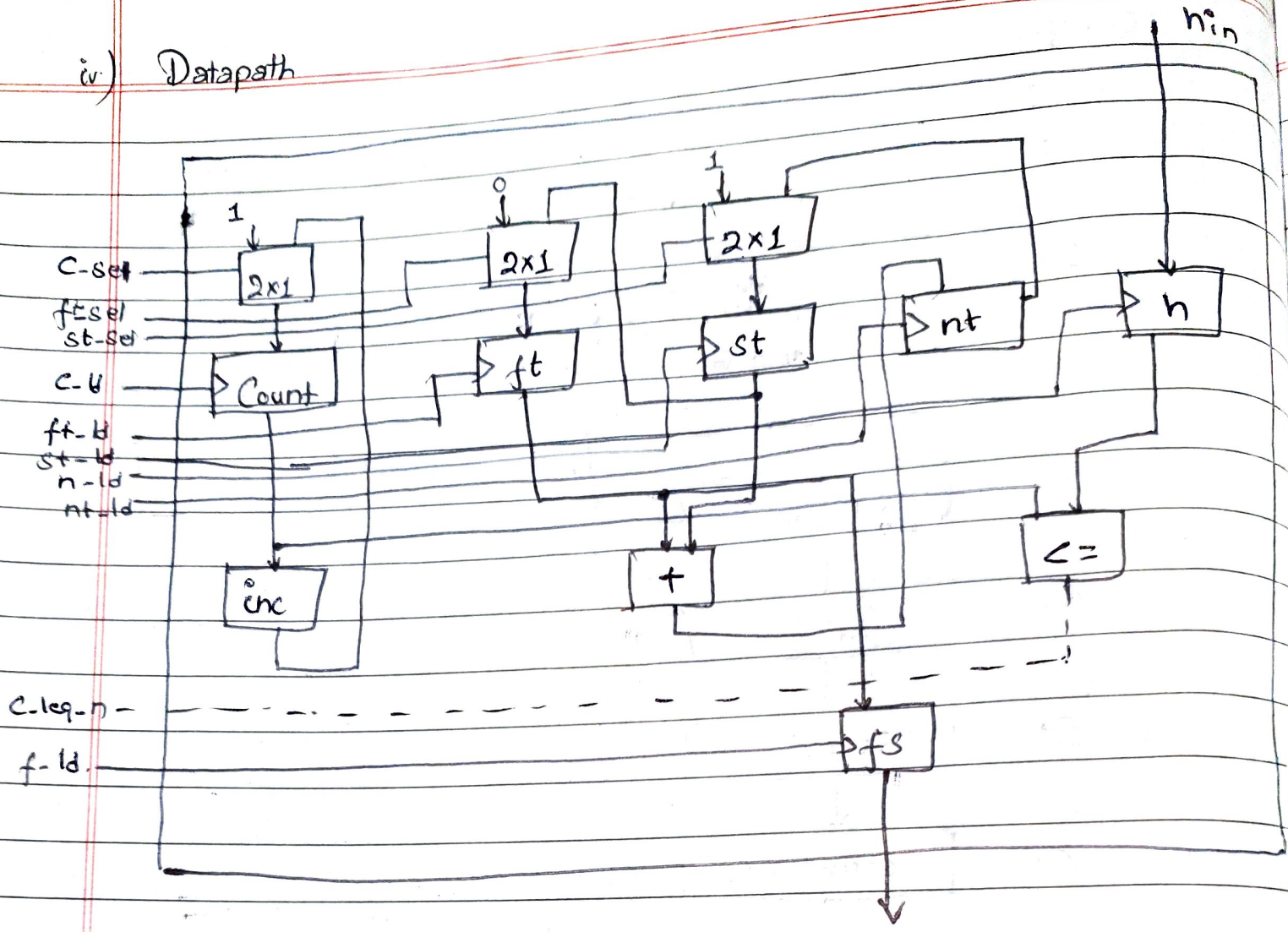
ii) Functionality:

```
int n, ft, st, nt, count
while (1)
{
    while (!goin);
    n = nin
    ft = 0;
    st = 1;
    count = 1;
    while (count <= n)
    {
        fs = ft;
        nt = ft + st;
        ft = st;
        st = nt;
        count ++;
    }
}
```

iii) FSMD:


1:
!1
1
goin
2:
!goin
2j:
3:
n=nin
4:
ft=0
5:
st=1
6:
count=1
7:
!(count<=n)
(count<=n)
8:
fs=ft
9:
nt=ft+st
10:
ft=st
11:
st=nt
12:
count++
7j:
1j:

iv) Datapath

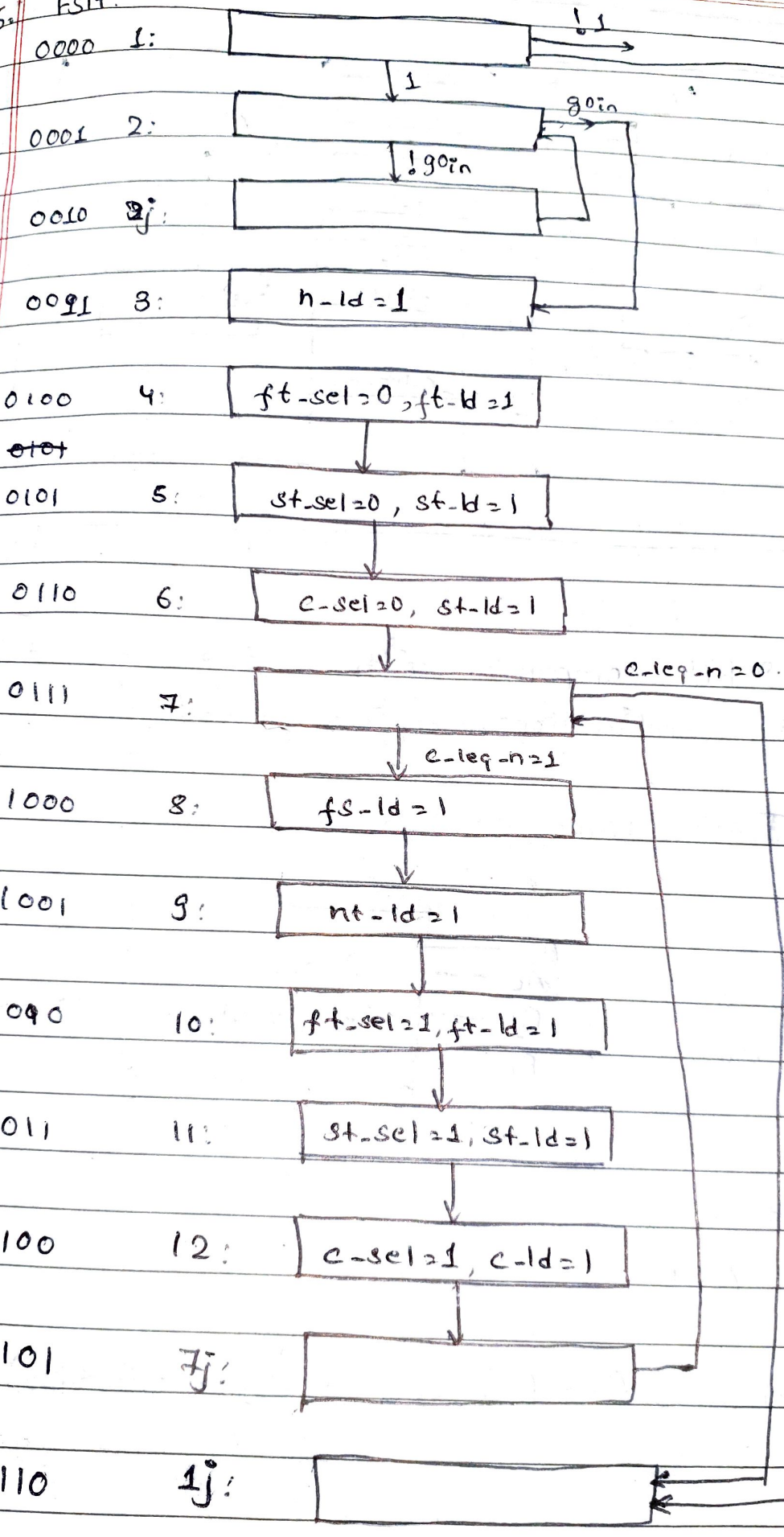


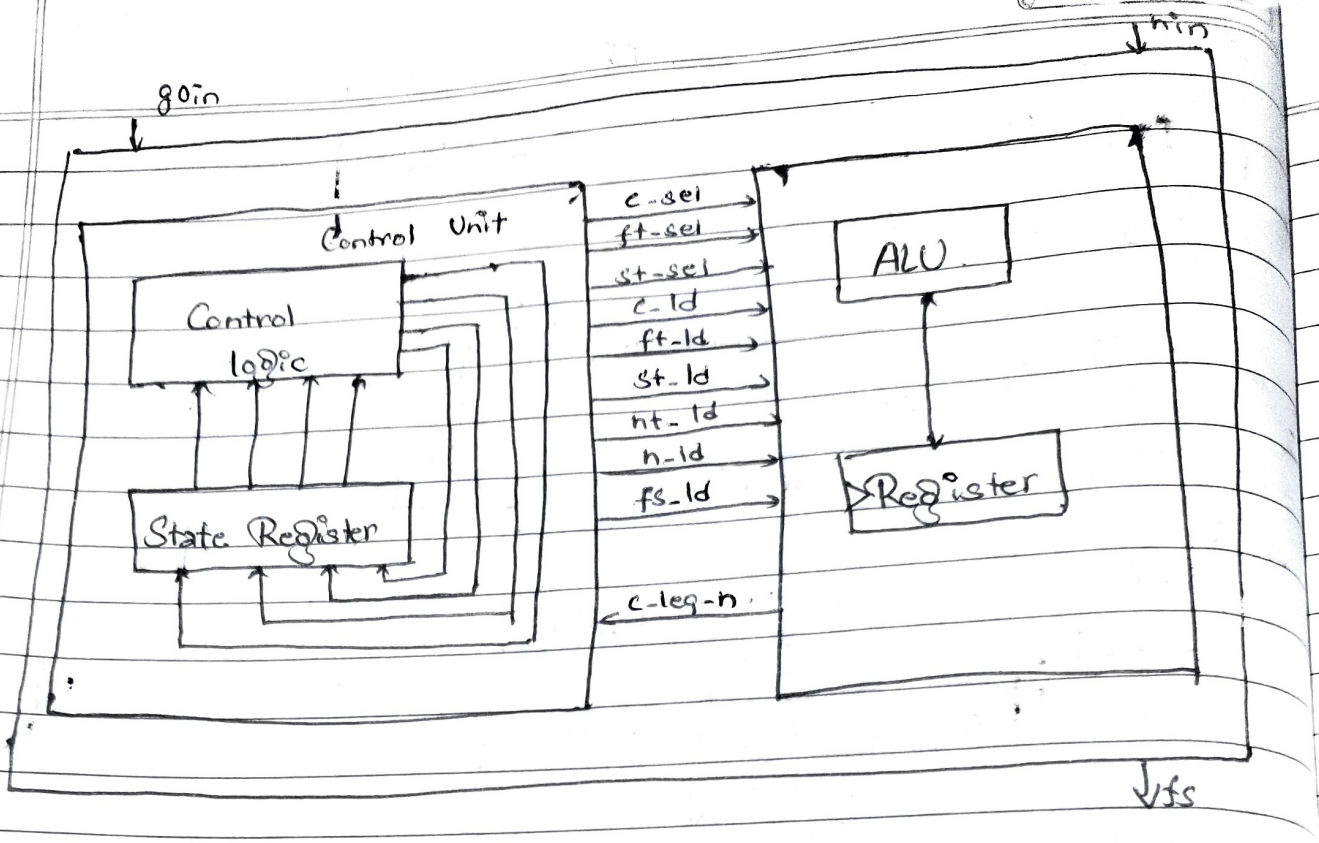
Registers: n, nt, ft, st, count, fs

MUX : MUX_{ft}, MUX_{st}, MUX_{count}

ALU : <= , +, inc.

5. FSM:


0000 1:
!1
1
0001 2:
go_in
!go_in
0010 2j:
0011 3:
h_ld = 1
0100 4:
ft_sel = 0, ft_ld = 1
0101 5:
st_sel = 0, st_ld = 1
0110 6:
c_sel = 0, st_ld = 1
0111 7:
c_leq_n = 0
c_leq_n = 1
1000 8:
fs_ld = 1
1001 9:
nt_ld = 1
1010 10:
ft_sel = 1, ft_ld = 1
1011 11:
st_sel = 1, st_ld = 1
1100 12:
c_sel = 1, c_ld = 1
1101 7j:
1110 1j:



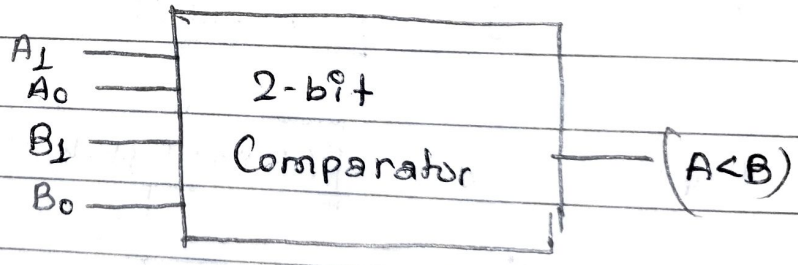
6 @. Design 2-bit comparator with a single output "less than" using the combinational design procedure.

Step 1: Truth table.

Inputs:

A: $A_1 A_0$

B: $B_1 B_0$



Output:

less than

i.e. $A<B$.

$A<B$ when $(A_1 < B_1)$ or $(A_1 = B_1) \cdot (A_0 < B_0)$.

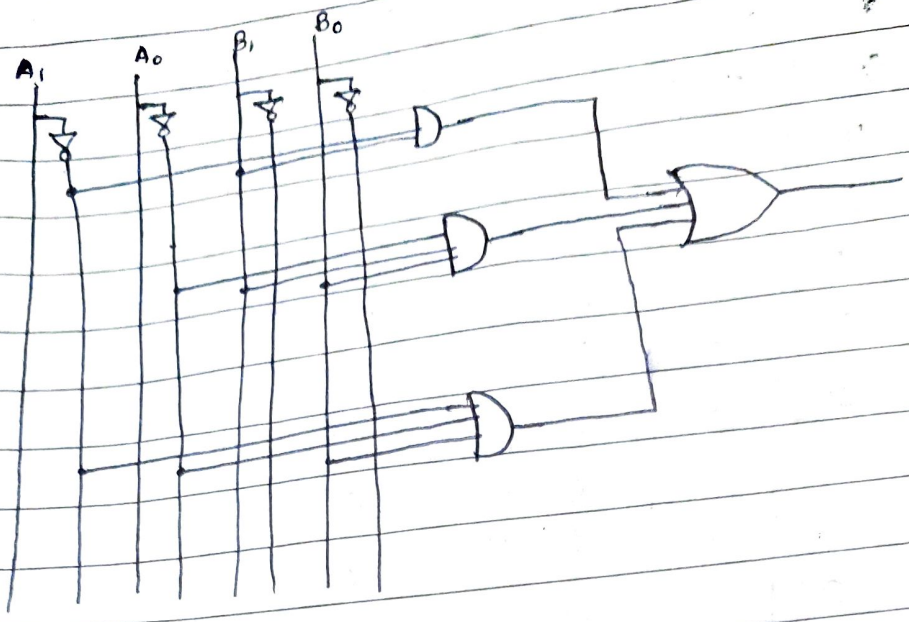
Input				Output
A_1	A_0	B_1	B_0	$A<B$
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

Step 2: K-map:

A_1A_0 \ B_1B_0	00	01	11	10
00	0	1	1	1
01	0	0	1	1
11	0	0	0	0
10	0	0	1	0

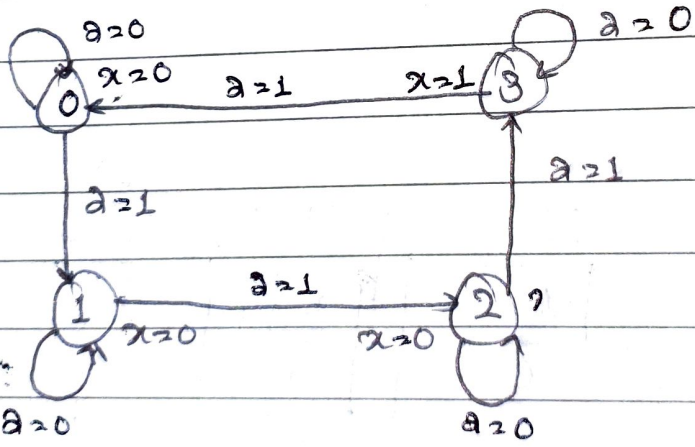
$A<B = A_1'A_0'B_0 + A_1'B_1 + A_0'B_1B_0.$

Step 3: Gate Implementation.

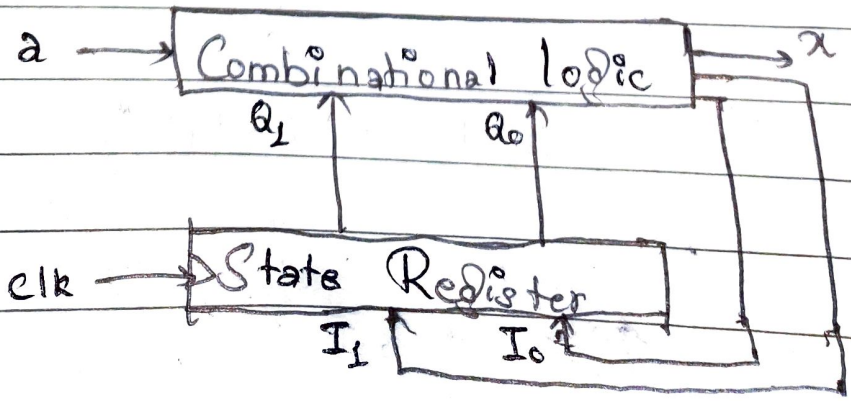


7 Q. Design a pulse divider. Slow down your pre-existing pulse so that you output at 1 every four pulse detected.

State diagram:



Implementation model



State table:

Inputs			Outputs		
Q_1	Q_0	a	I_1	I_0	x
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	1	0
0	1	1	1	0	0
1	0	0	1	0	0
1	0	1	1	1	0
1	1	0	1	1	1
1	1	1	0	0	1

K-maps:

For I_1

Q_1 \ Q_0a	00	01	11	10
0	0	0	1	1
1	0	1	0	1

$I_1 = Q_1 Q_0 a + Q_1 a' + Q_1 Q_0'$

For I_0:

Q_1 \ Q_0a	00	01	11	10
0	0	1	1	0
1	1	0	0	1

$I_0 = Q_0 a' + Q_0' a$

For x:

Q_1 \ Q_0a	00	01	11	10
0	0	0	1	0
1	0	0	1	0

$x = Q_1 Q_0$.

Gate Implementation: