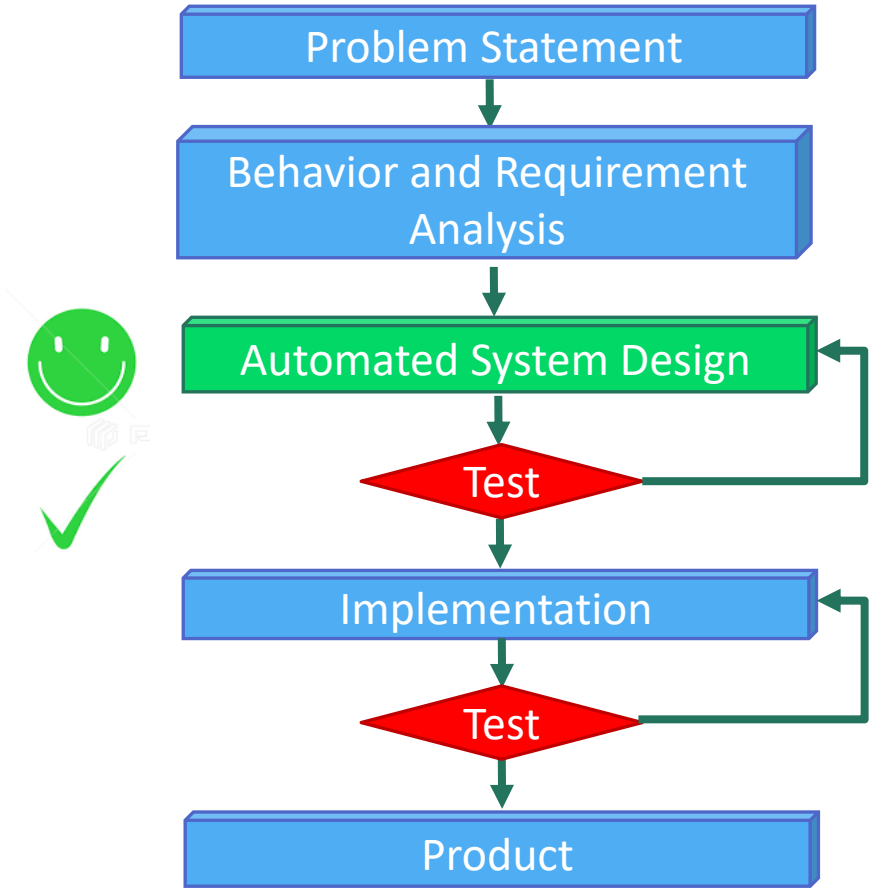
Digital System Design

Hajar Falahati

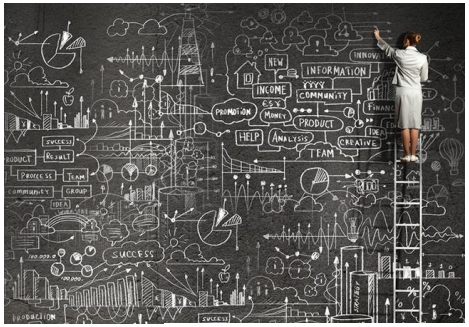
hfalahati@ipm.ir
hfalahati@ce.sharif.edu

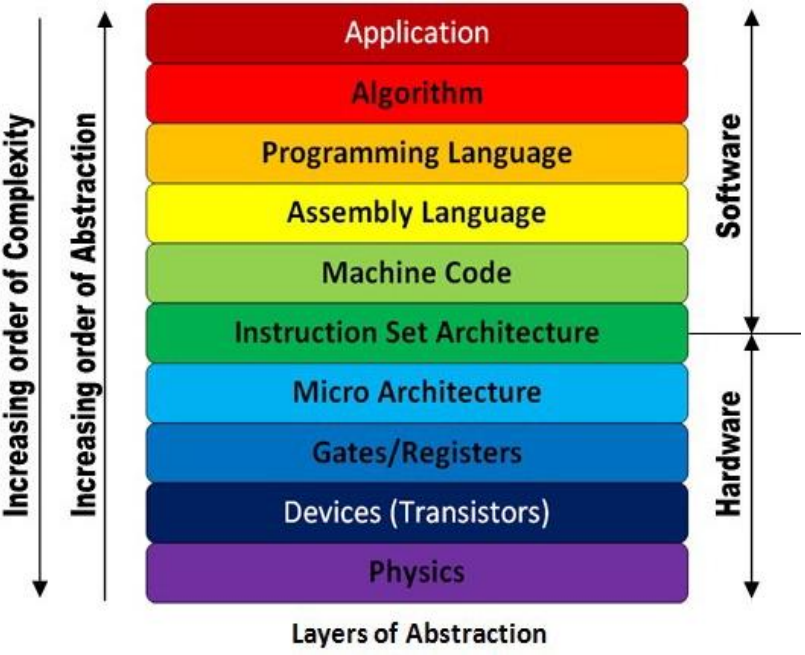
Automated Design Flow

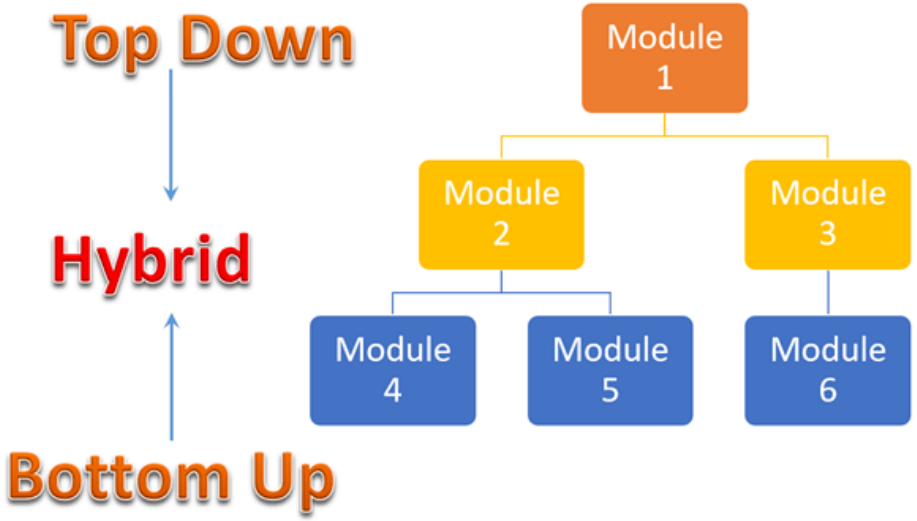


Design Complexity

- How to handle the **complexity in design and modeling**?







Outline

- Modeling
  - **Why?**
  - **FSM**
  - **ASM**

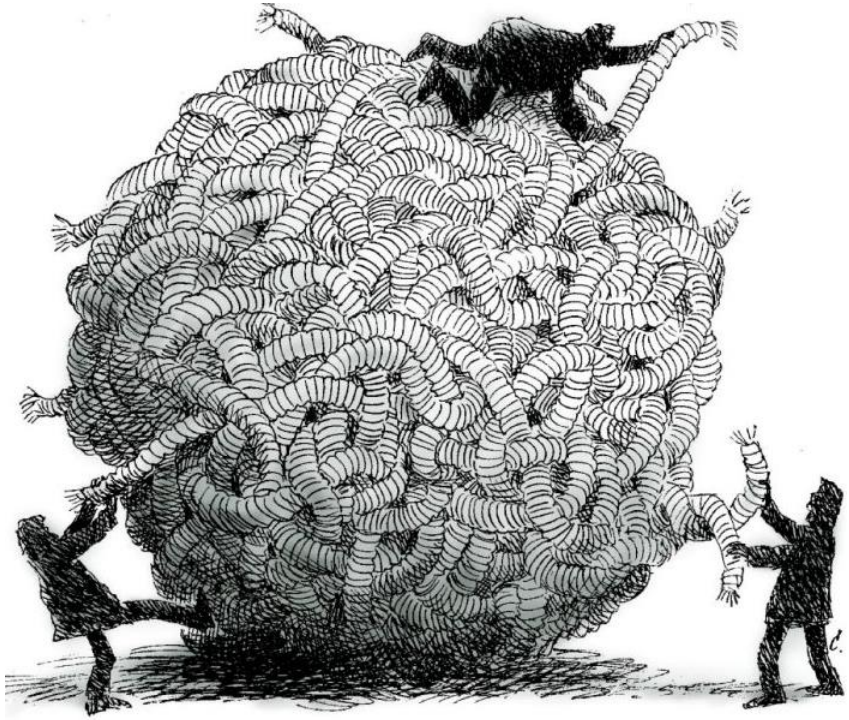
Customers!

Help My Friend!

- My friend is a digital designer and recently receives some proposals from different customers.
  - **Handle the traffic problem in a highway and country road intersection**
  - **Monitor the parking space**
  - **Optimize an elevator to reach as fast as possible**
  - **Control forest fire**
  - **Make my home safer**

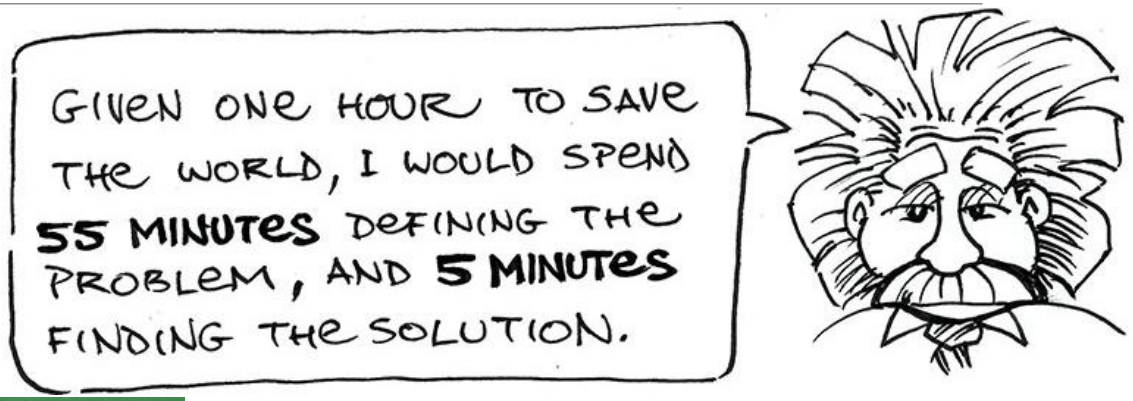
What is My Friend's Problem!

- " The customer never knows exactly what he/she wants!"
- " Their descriptions are too high-level"



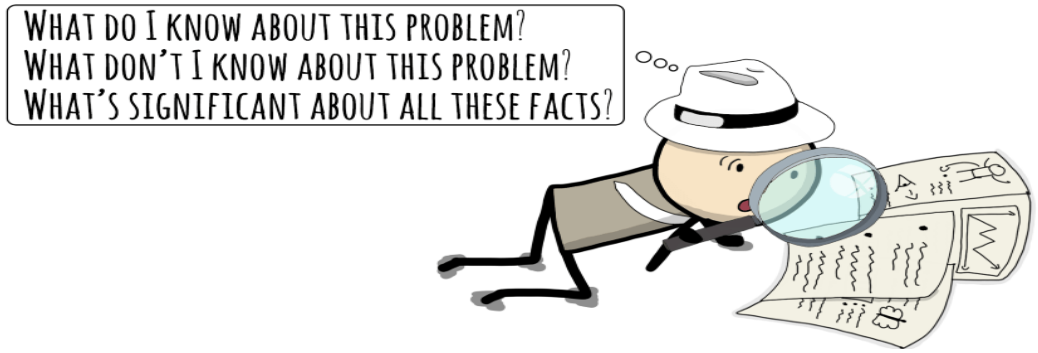
Problem Definition

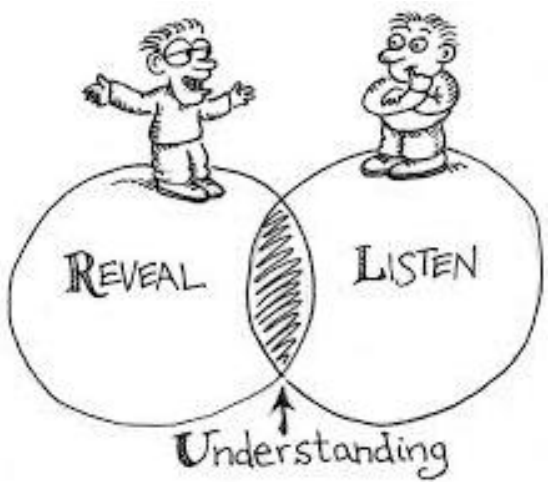
- Identify the problem
- What is the problem?

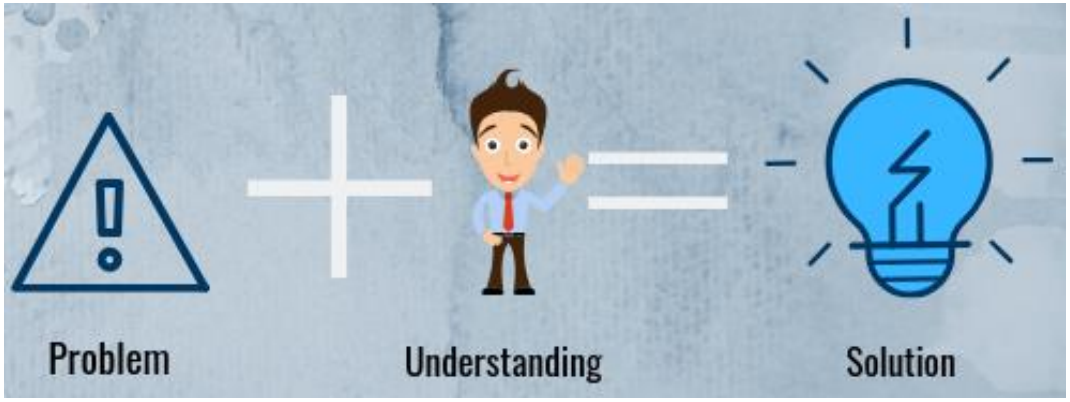


Problem Understanding

- Carefully explore about the problem
  - Listen to customer

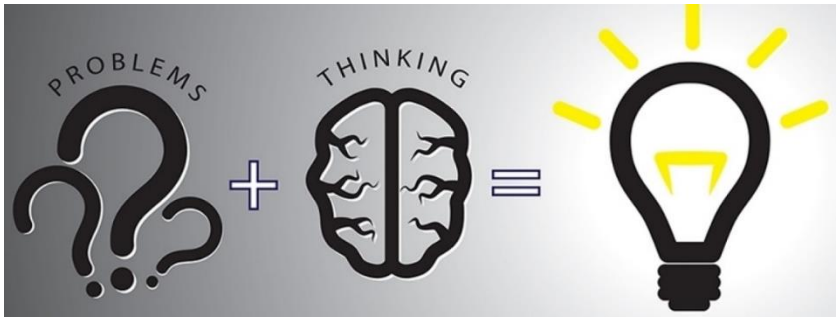


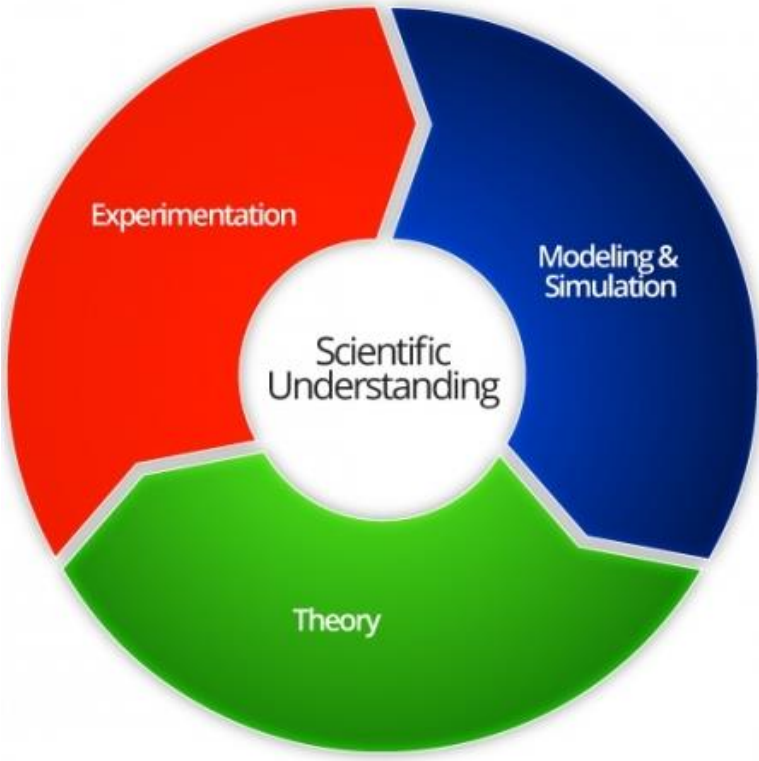




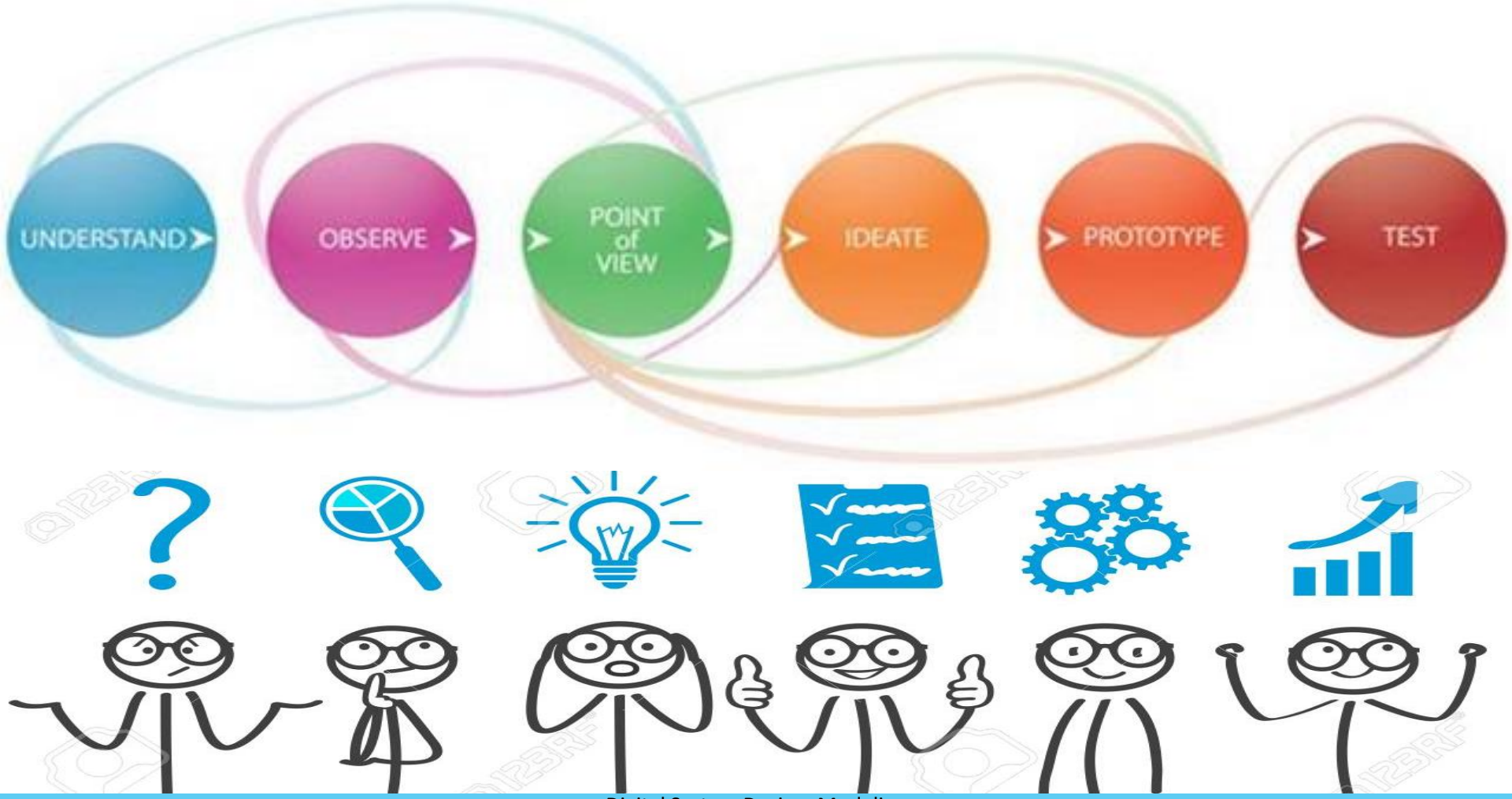
Problem Modeling

- Formulate the problem
  - Involves its **detection**, **identification,** and **definition**
- Interpreting the problem





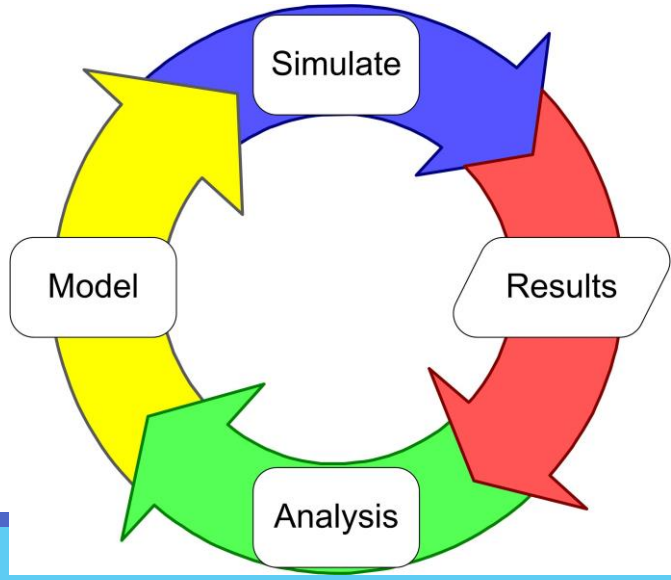
Problem Solving Strategy



Modeling

Why Modeling?

- Specific language to **describe higher level concepts** and **abstractions**
- Graphical representation helps us in
  - Understanding
  - Documenting
  - Explaining a problem
- Writing down **ideas** in a structured way
  - ➔ Tackle complexity

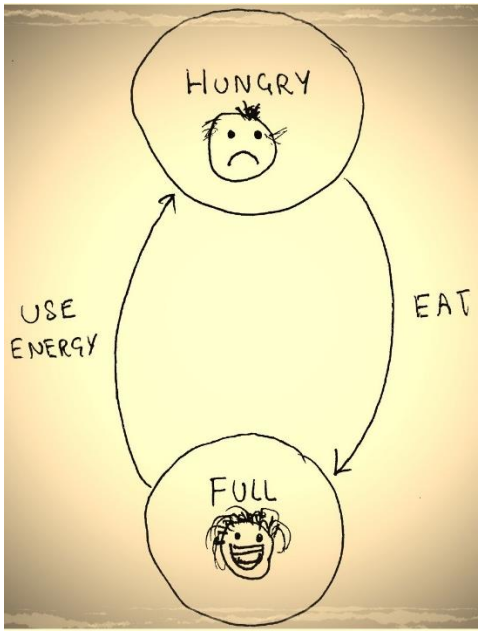


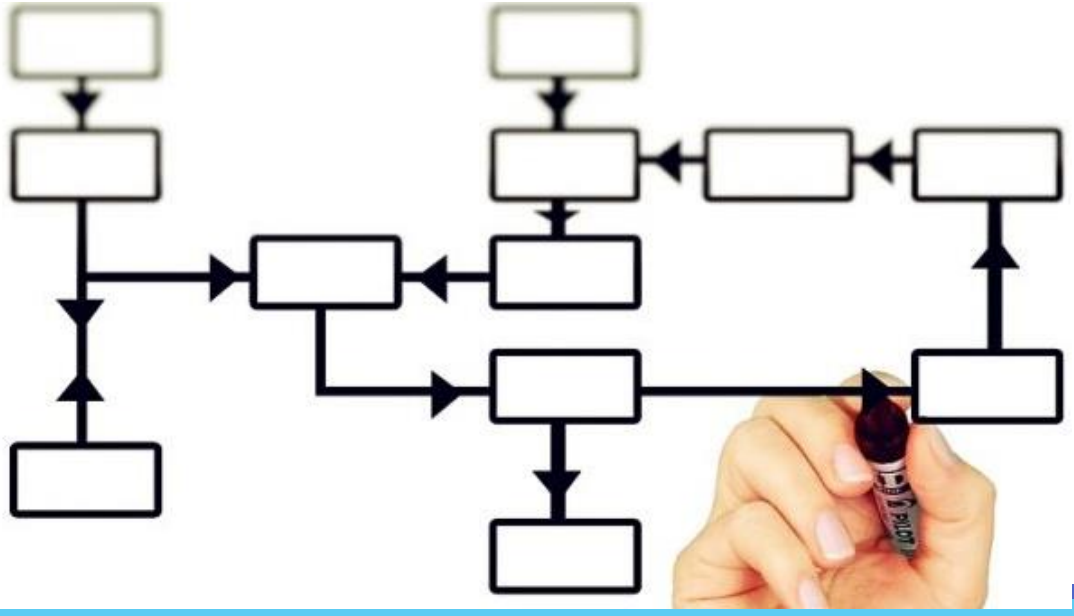
Modeling is Necessary

- Not All Modeling, Of course !

Modeling Types

- Finite State Machine
  - FSM
- Algorithmic State Machine
  - ASM

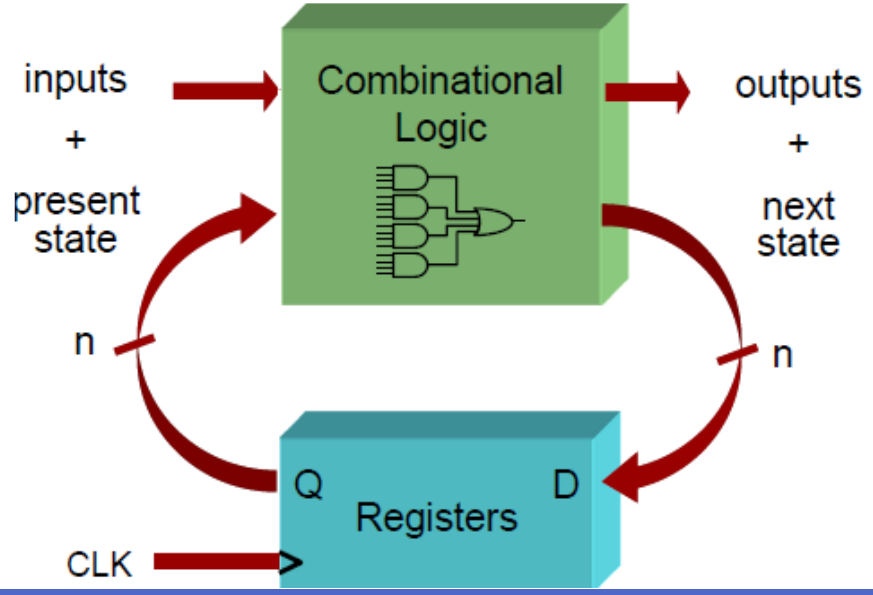




FSM

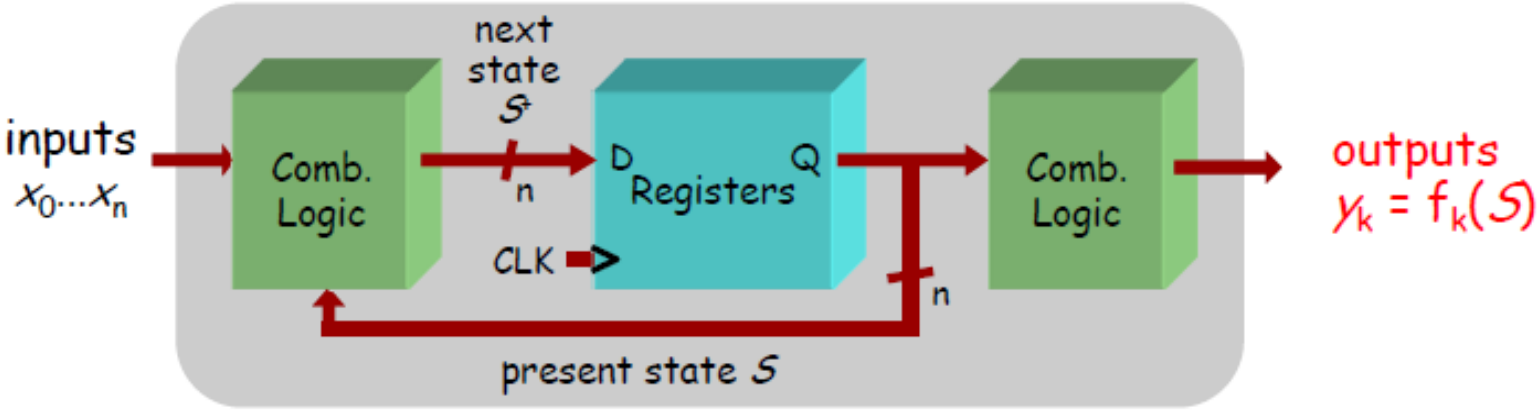
Finite State Machine (FSM)

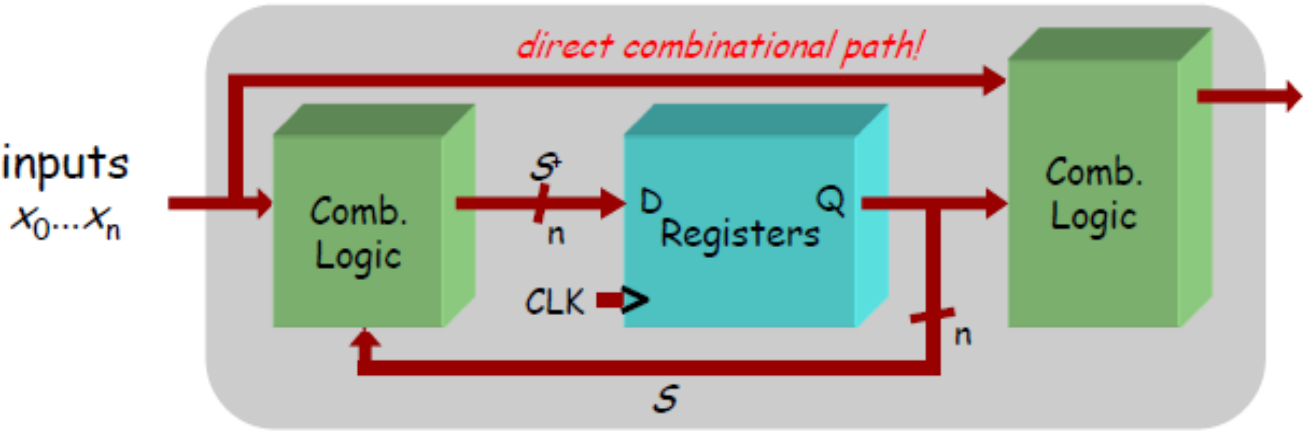
- A system that visits a **finite** number of **logically distinct states**
- A useful **abstraction** for **sequential circuits** with **centralized states of operation**
- At each **clock edge**, combinational logic computes
  - **Outputs** and **next state** as a function of **inputs** and **present state**



FSM Types

Moore FSM

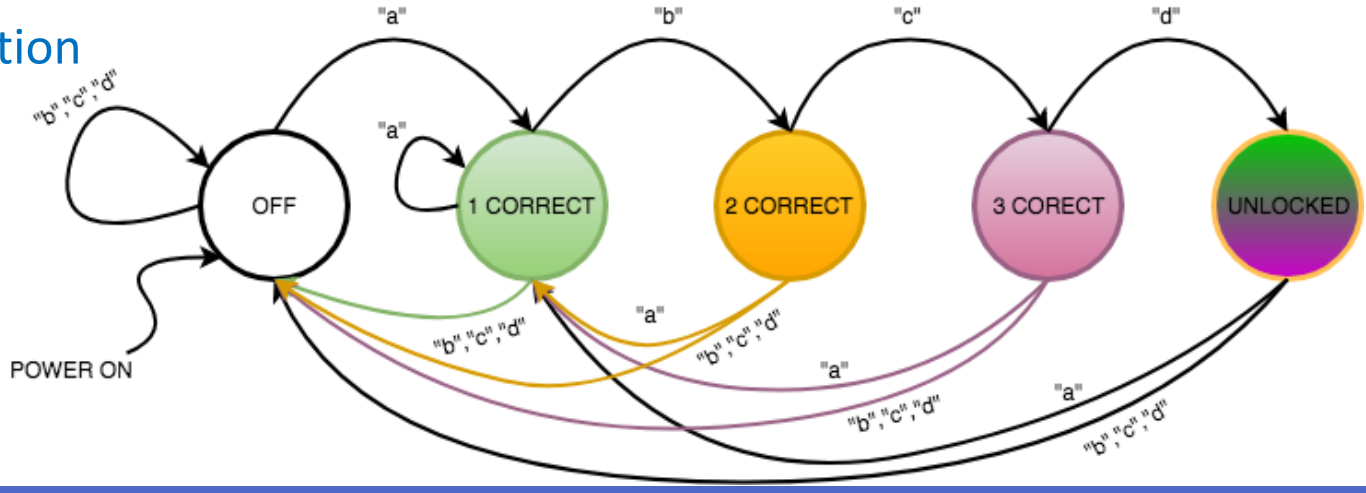




Mealy FSM

State Diagram

- A set of states
  - Nodes in a graph
- A set of inputs and outputs
  - Edges in a graph
- A set of state transition function
  - Edges in a graph
- An output function

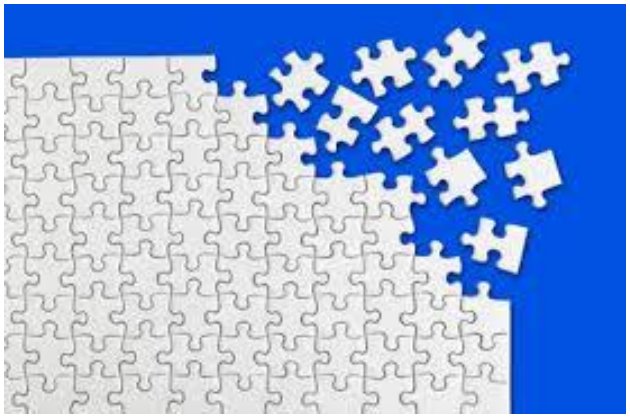


Vending Machine

- The computer department need a new soda machine
- We decided to ask computer students to design a controller for the new vending machine

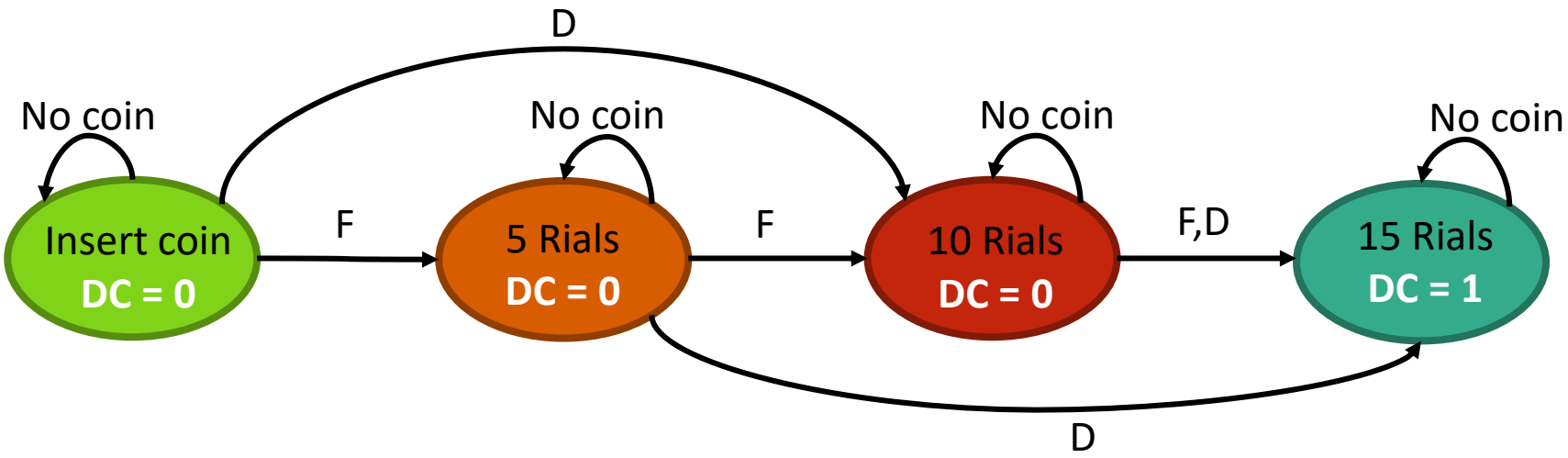
Vending Machine: Characteristics

- Characteristics
  - All selections cost **15** Rial
  - Machine does not return changes!
- Input
  - **D**: dim inserted (10 Rial!)
  - **F**: five inserted (5 Rial!)
- Output
  - **DC**: dispense can



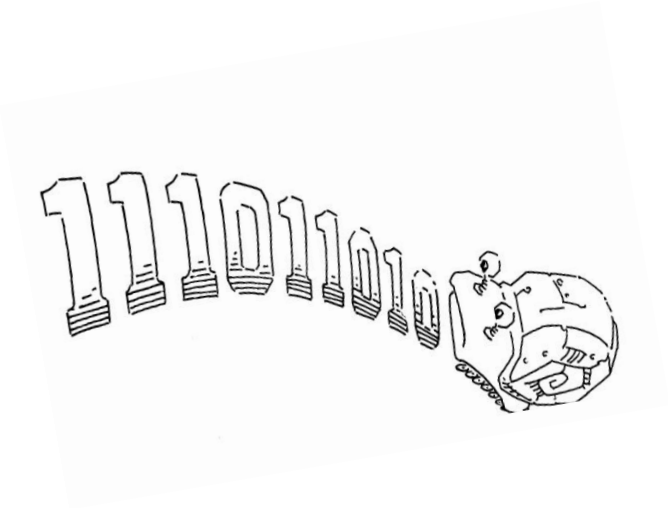
Vending Machine: FSM

- **Insert coin**
  - Default state
  - No money has been inserted
- **5 Rials**
  - A 5 Rial coin has been inserted
- **10 Rials**
  - A 10 Rial coin has been inserted
- **15 Rials**
  - Total money has been reached to 15 Rial
  - Done!



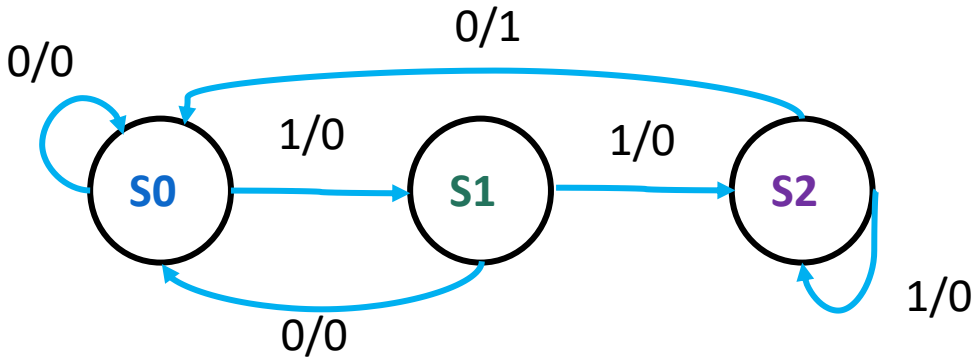
Pattern Recognition

- Recognize a specific bit pattern (*110*) in a bitstream



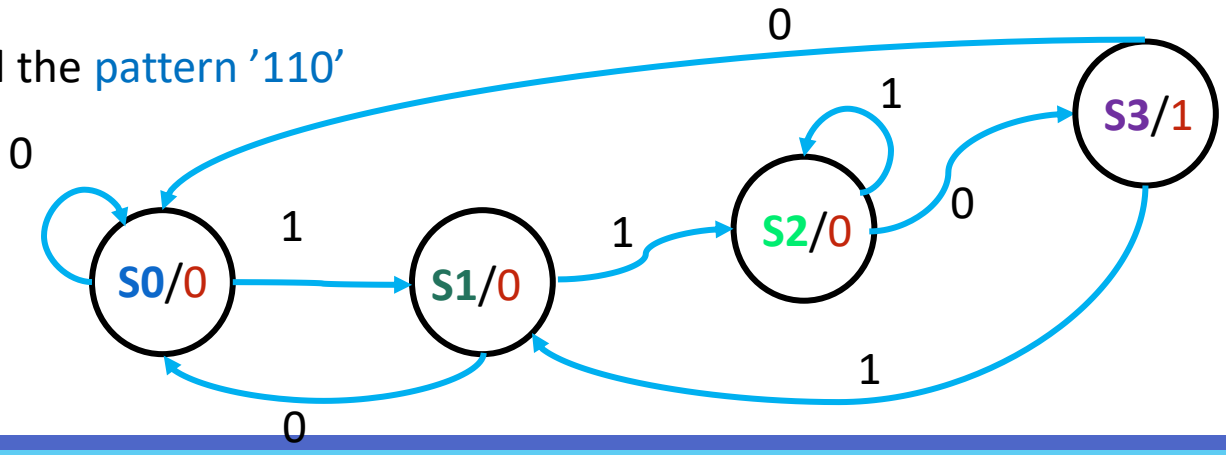
110 Mealy Detector

- State **S0**
  - We have not recognized any useful pattern
- State **S1**
  - We have recognized the pattern ’1’
- State **S2**:
  - We have recognized the pattern ’11’
  - **Output**: recognizing an input bit ’0’ in state S2



110 Moore Detector

- State **S0**
  - We have not recognized any useful pattern
- State **S1**
  - We have recognized the pattern ’1’
- State **S2**:
  - We have recognized the pattern ’11’
- State **S3**:
  - We have recognized the pattern ’110’
  - Output becomes 1

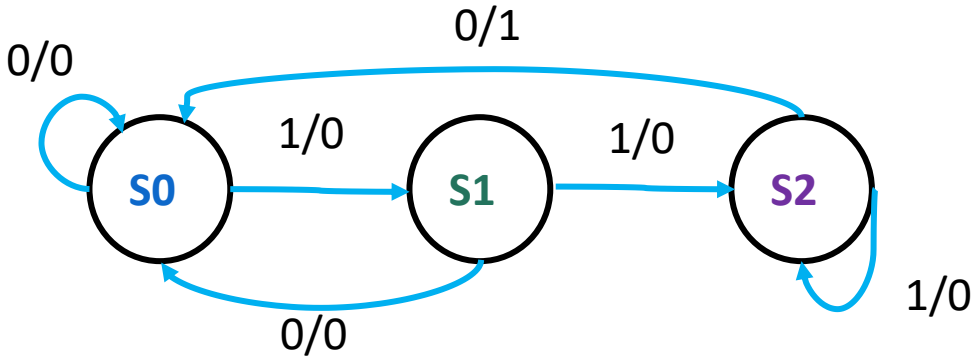


FSM Type Conversion

- Mealy → Moore
  - Make a state transition table for the Mealy FSM
  - Next-state is combined with the relevant output
    - State 1 and input '1' → state S2 with output 0 (S2,0)
  - There is one Moore state for each unique (next-state, output) pattern
    - (S2, 0)

Mealy → Moore

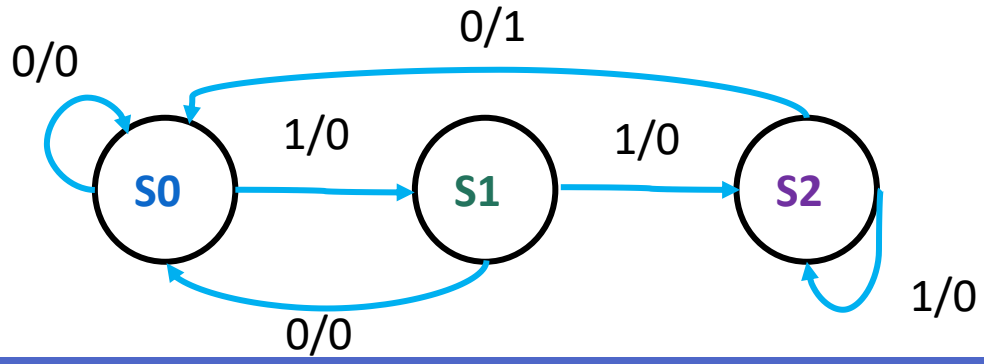
- Convert the Mealy 110 detector to Moore 110 detector



State Transition Table

- States
  - (S0, 0) = SA
  - (S0, 1) = SB
  - (S1, 0) = SC
  - (S2, 0) = SD

Current State	$X = 0$	$X = 1$
S0	S0, 0	S1, 0
S1	S0, 0	S2, 0
S2	S0, 1	S2, 0

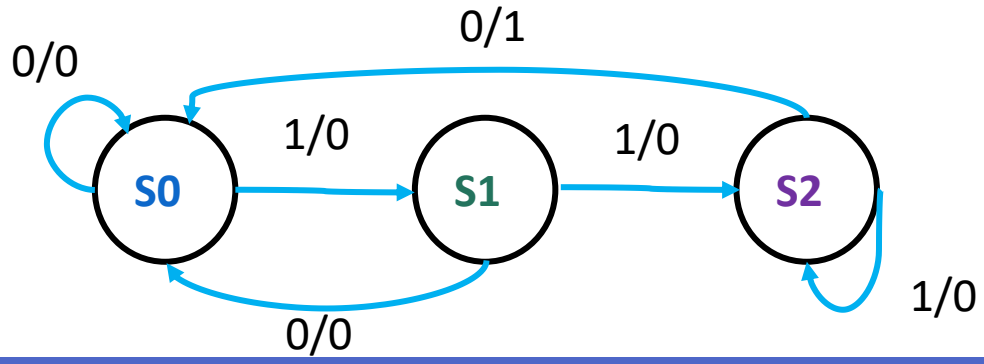


State Transition Table

- States
  - (S0, 0) = SA
  - (S0, 1) = SB
  - (S1, 0) = SC
  - (S2, 0) = SD

Current State	$X = 0$	$X = 1$
S0	S0, 0	S1, 0
S1	S0, 0	S2, 0
S2	S0, 1	S2, 0

- There *may* be **multiple Moore** state**s** for **a single Mealy** state
  - **Mealy** state S0 is split into **Moore** state**s** (S0, 0) and (S0, 1)



Make Pair (NextState, Out)

Current State	$X = 0$	$X = 1$
S0	S0, 0	S1, 0
S1	S0, 0	S2, 0
S2	S0, 1	S2, 0

Current State	$X = 0$	$X = 1$	Output
SA = (S0 , 0)	SA	SC	0
SB = (S0 , 1)	SA	SC	1
SC = (S1 , 0)			
SD = (S2 , 0)			

Make Pair (NextState, Out)

Current State	$X = 0$	$X = 1$
S0	S0, 0	S1, 0
S1	S0, 0	S2, 0
S2	S0, 1	S2, 0

Current State	$X = 0$	$X = 1$	Output
SA = (S0 , 0)	SA	SC	0
SB = (S0 , 1)	SA	SC	1
SC = (S1 , 0)	SA	SD	0
SD = (S2 , 0)			0

Make Pair (NextState, Out)

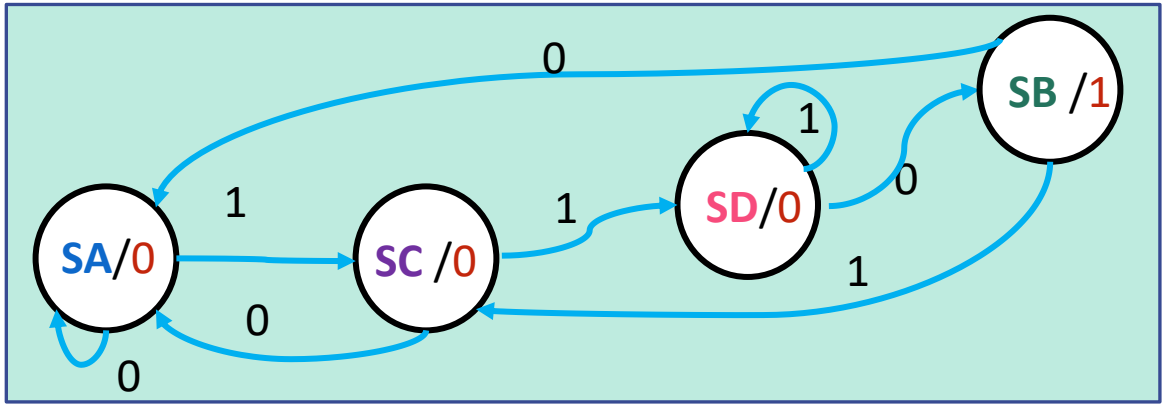
Current State	$X = 0$	$X = 1$
S0	S0, 0	S1, 0
S1	S0, 0	S2, 0
S2	S0, 1	S2, 0

Current State	$X = 0$	$X = 1$	Output
SA = (S0 , 0)	SA	SC	0
SB = (S0 , 1)	SA	SC	1
SC = (S1 , 0)	SA	SD	0
SD = (S2 , 0)	SB	SD	0

Make Pair (NextState, Out)

Current State	$X = 0$	$X = 1$
S0	S0, 0	S1, 0
S1	S0, 0	S2, 0
S2	S0, 1	S2, 0

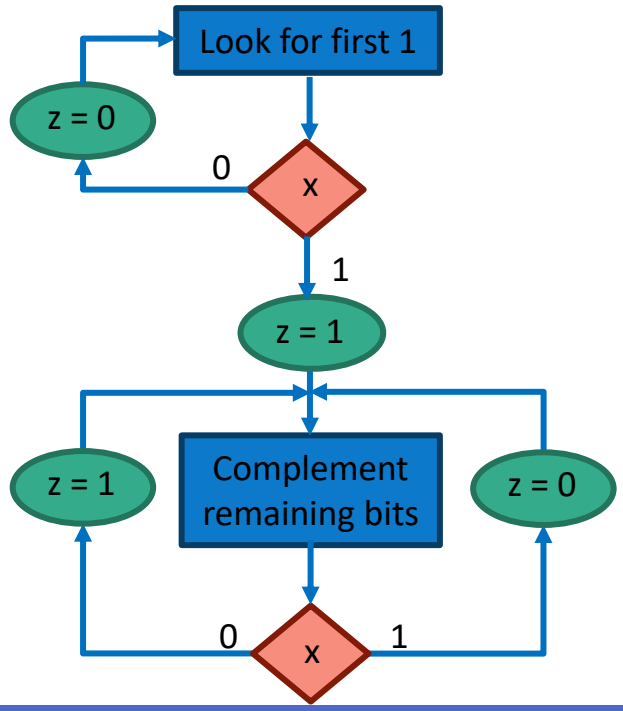
Current State	$X = 0$	$X = 1$	Output
SA = (S0 , 0)	SA	SC	0
SB = (S0 , 1)	SA	SC	1
SC = (S1 , 0)	SA	SD	0
SD = (S2 , 0)	SB	SD	0



ASM

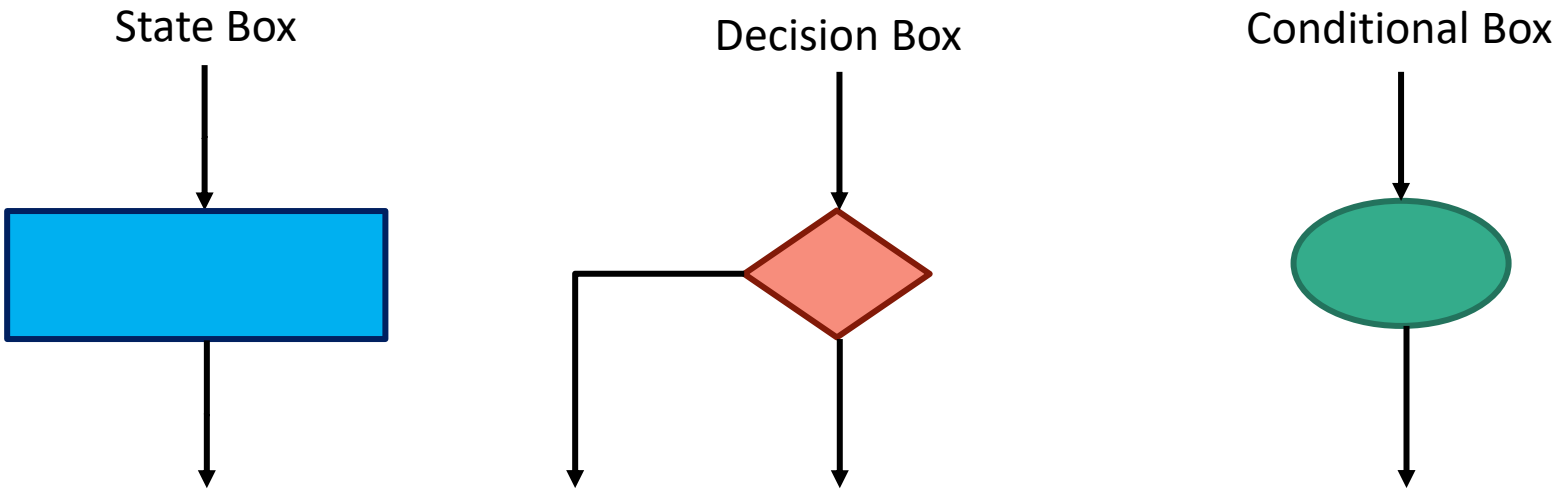
Algorithmic State Machine (ASM)

- A **systematic** way to **design complex** digital systems
  - Complex digital systems
  - Large number of inputs and outputs
- **Describes** the **sequence of events**
- **Describes timing relationship** between the states
- **Behavioral model**
- **Basic Idea**
  - Flowchart



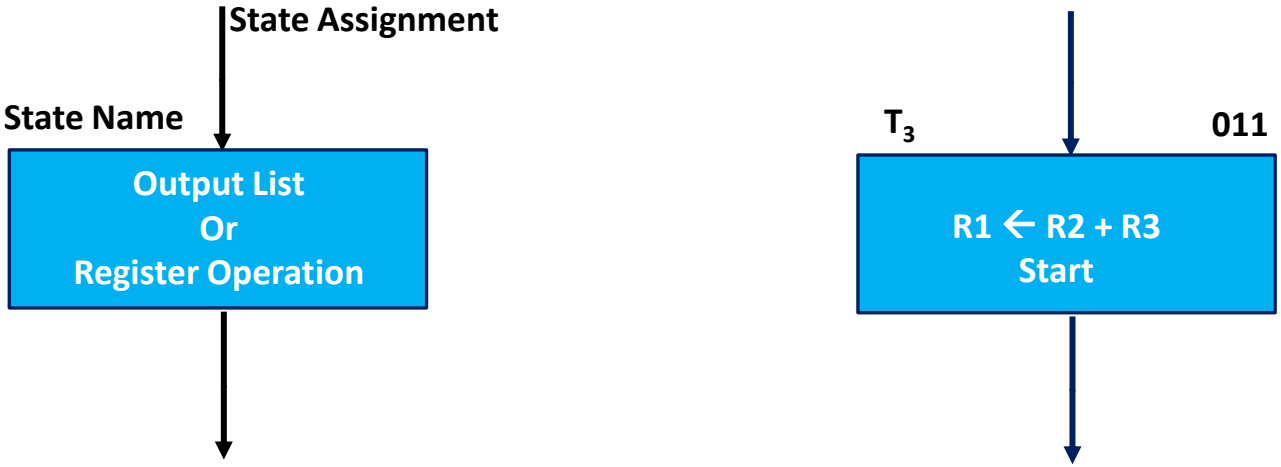
ASM chart Elements

- ASM chart elements
  - State box
  - Decision box
  - Conditional box



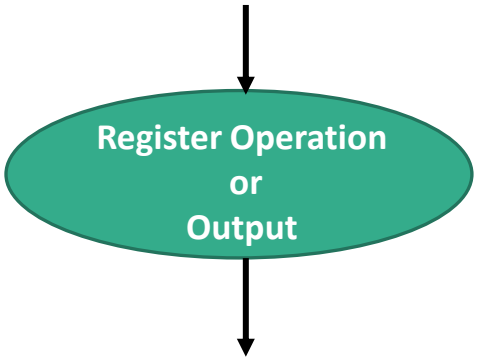
State Box

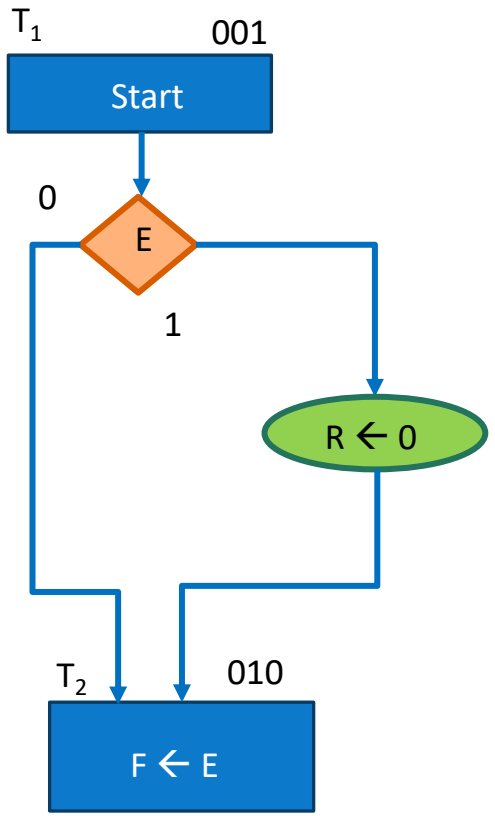
- Represents the state of the system
- Register Transfers
- Takes one cycle to be executed
  - R1 <- R2 + R3



Conditional Box

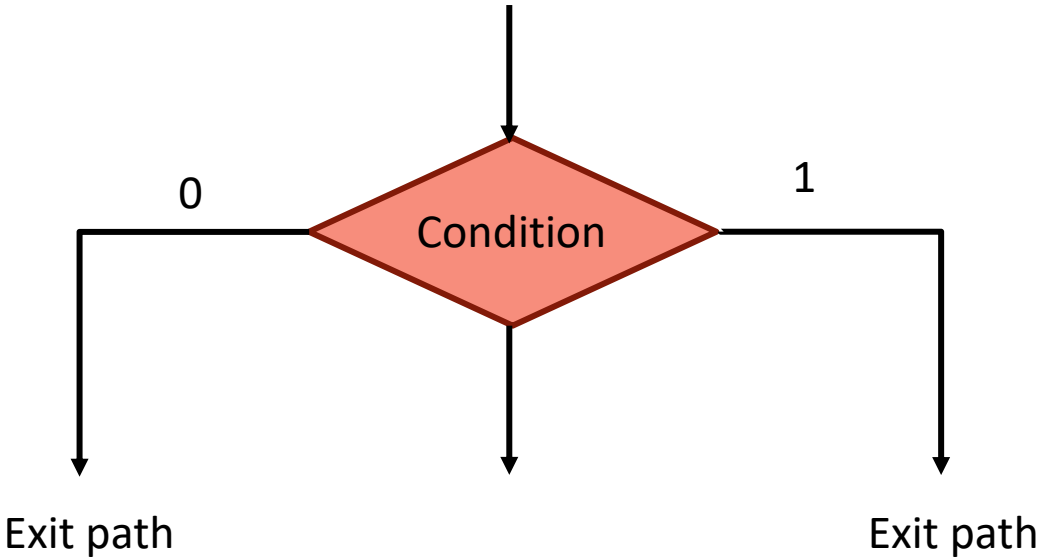
- Register Transfers
- Contain conditional output list
  - Depends on both the state of the system and the inputs
  - A.k.a., mealy output
  - A condition output must follow a decision box
- Takes one cycle to be executed
  - R1 <- R2 + R3





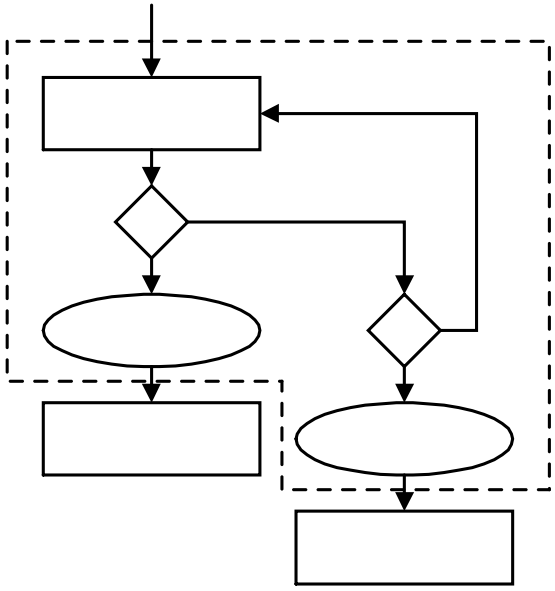
Decision Box

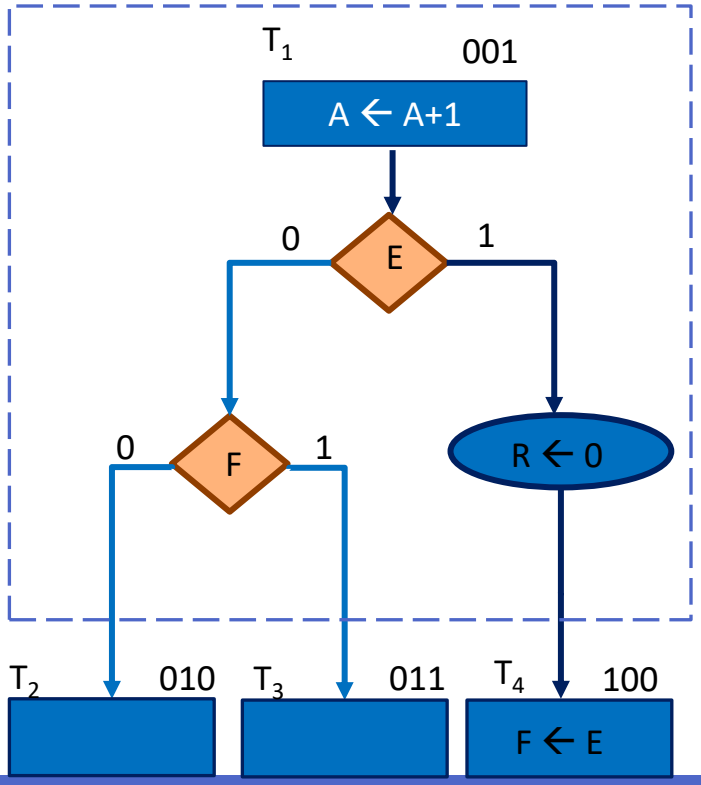
- Binary expressions
  - ~R1[7]



ASM Block

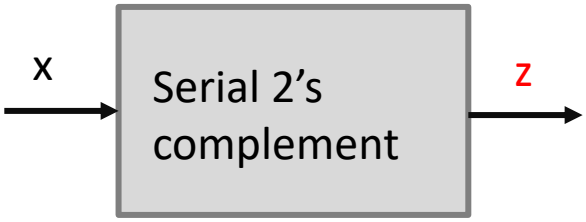
- Represents **what happens** in the system during one clock cycle
- Includes
  - Only **one state box**
  - All other boxes (decision and conditional boxes)
    - Connected to the exit path of the state box





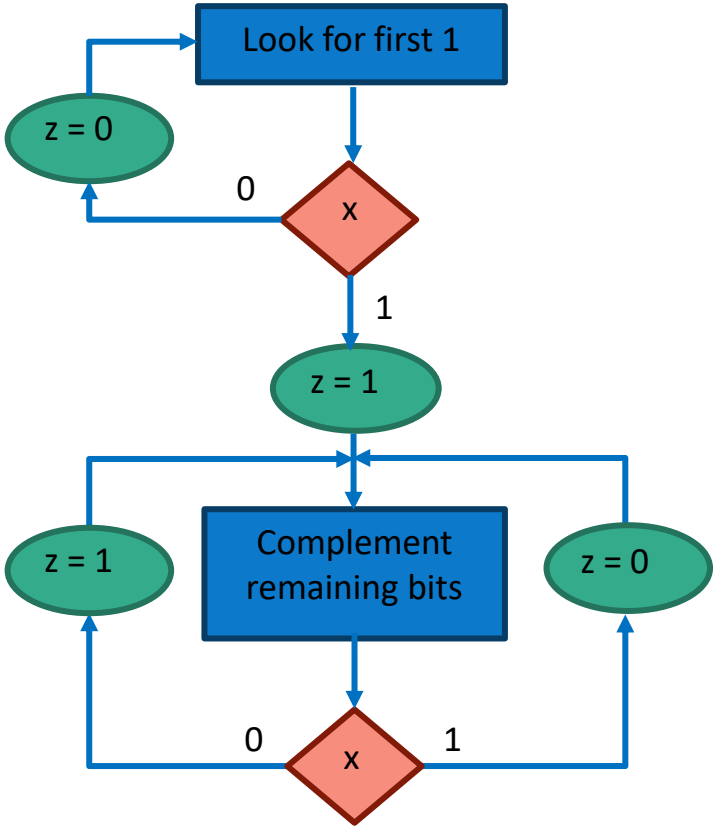
Sample Design 1

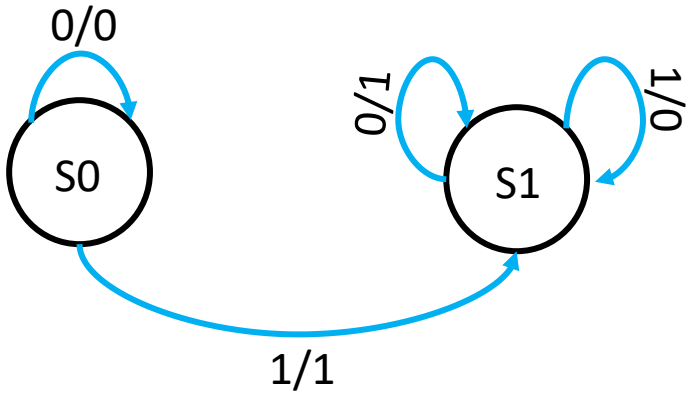
- Design a control unit for a serial 2's complement



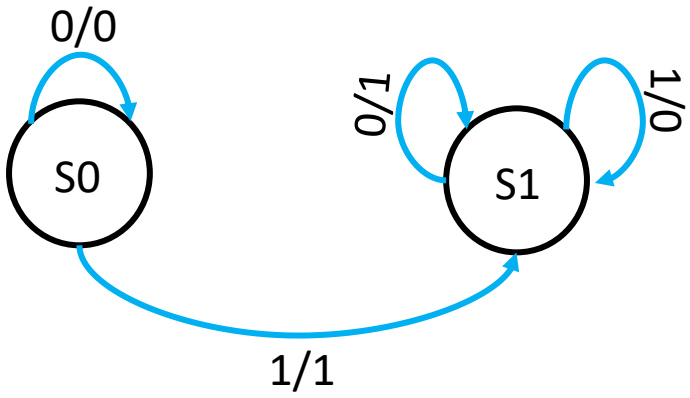
t	x	z
0	0	0
1	0	0
2	1	1
3	1	0

Sample Design 1: Modeling





Sample Design 1: Excitation Table

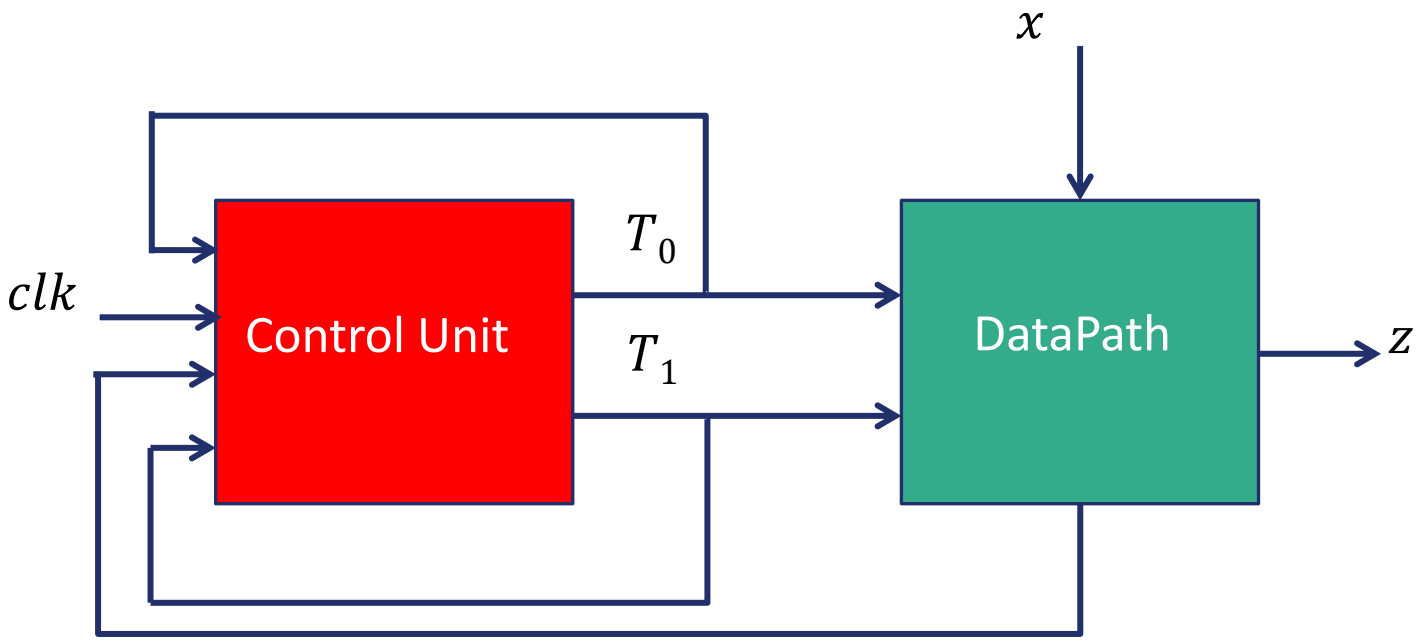


State	Input (x)	Next State	Output (z)	D
A(0)	0	A	0	0
A	1	B	1	1
B(1)	0	B	1	1
B	1	B	0	1

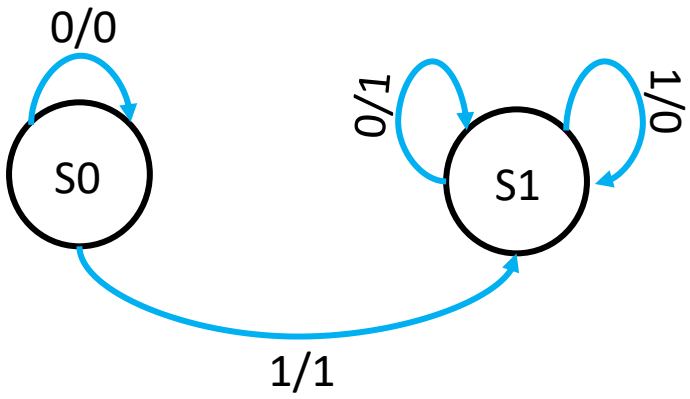
$z = Ax+Bx'$

$D=Ax+B$

Sample Design 1: Block Diagram



Sample Design 1: Design



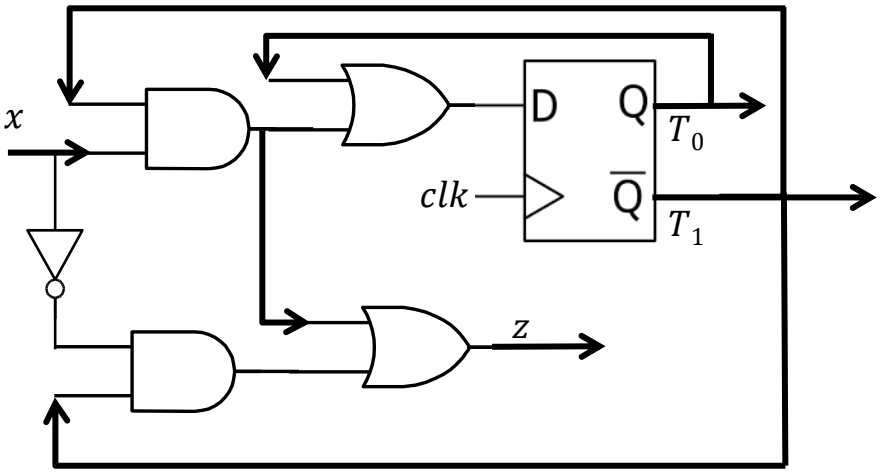
State	Input (x)	Next State	Output (z)	D
A(0)	0	A	0	0
A	1	B	1	1
B(1)	0	B	1	1
B	1	B	0	1

z = Ax+Bx’

z = Q’ x+Qx’

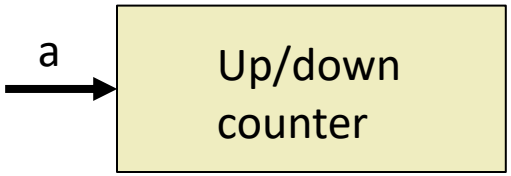
D=Ax+B

D= Q’ x + Q

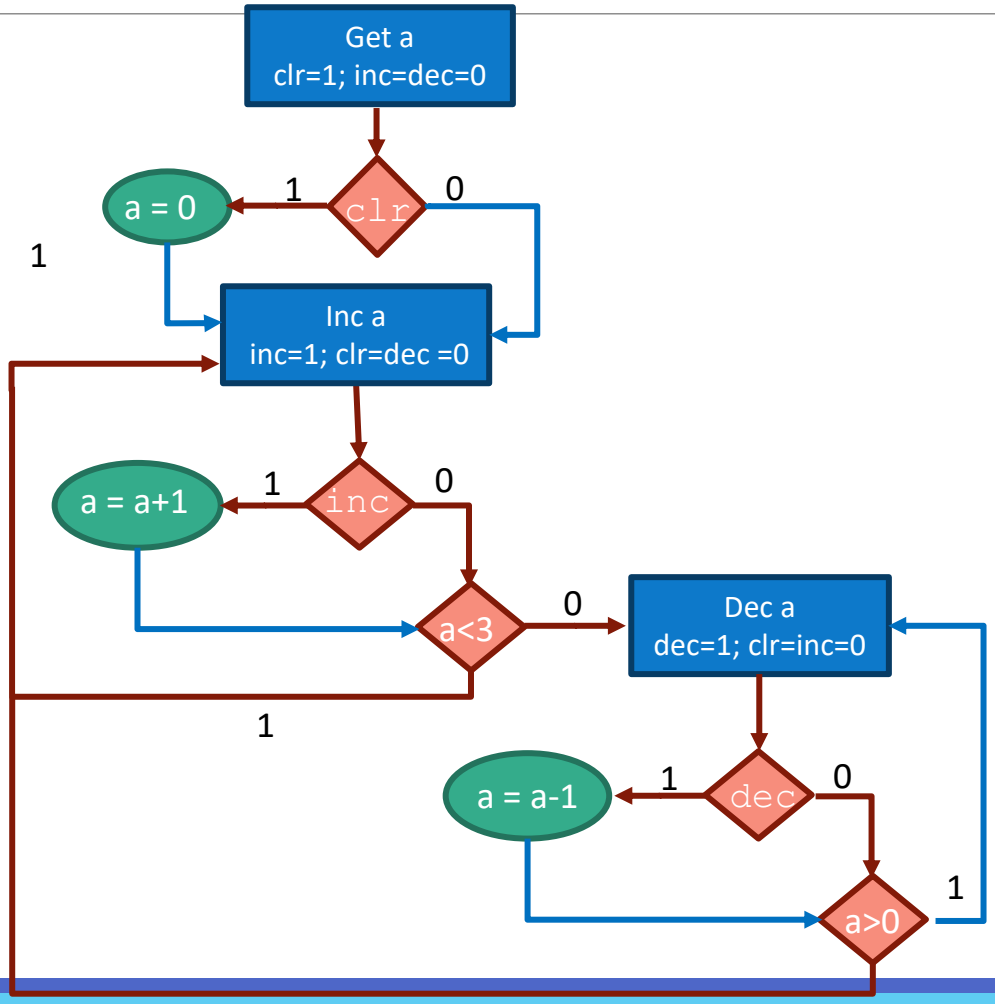


Sample 3

- Design an up/down counter



Up/Down Counter ASM



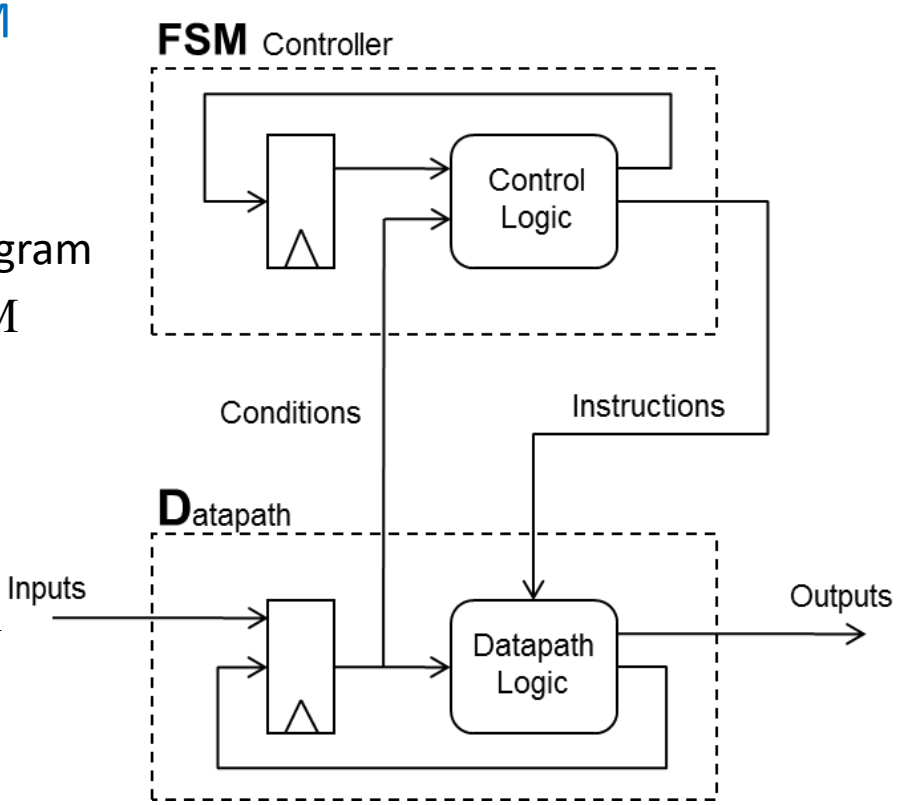
FSMD

FSMD

- Combines a hardware control model (an FSM) with a datapath
- FSMD datapath uses conditionally executed always blocks
  - Instructions will be executed ONLY when a controller invokes them
- FSMDs are useful because they capture control-flow as well as data-flow in hardware
  - C programs are a combination of control-flow and data-flow which implicitly 'connects' hardware FSMD models and C programs

FSMD (cont'd)

- An FSMD contains two stacked FSM
- Controller
  - Top FSM
  - Specified using a state transition diagram
  - Sends instructions to the bottom FSM
- Datapath
  - Bottom FSM
  - Specified using expressions
  - Receives status information in return



Sample 4

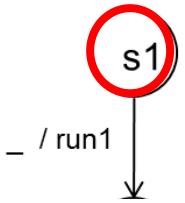
Greatest Common Divisor (GCD) using Euclid's algorithm.

GCD

```
1: int gcd(int a, int b) {
2:   while (a != b) {
3:     if (a > b)
4:       a = a - b;
       else
5:       b = b - a;
     }
6:   return a;
   }
```

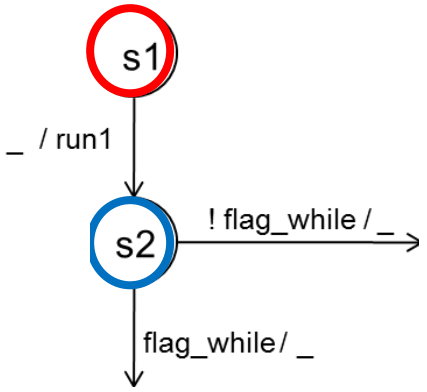
GCD Control Unit

```
1: int gcd(int a, int b) {
2:    while (a != b) {
3:      if (a > b)
4:        a = a - b;
        else
5:        b = b - a;
      }
6:    return a;
   }
```



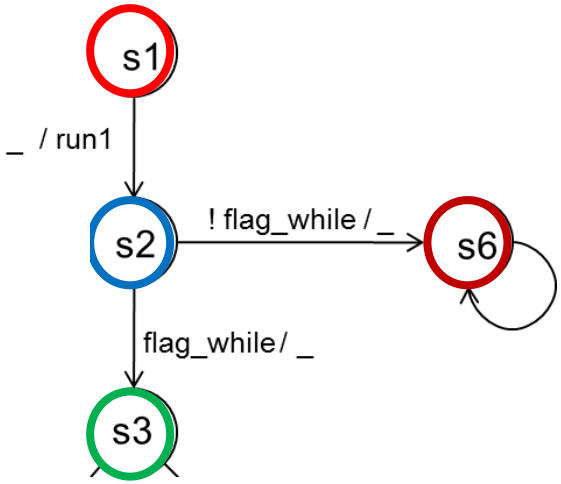
GCD Control Unit

```
1: int gcd(int a, int b) {
2:    while (a != b) {
3:       if (a > b)
4:          a = a - b;
         else
5:          b = b - a;
      }
6:    return a;
   }
```



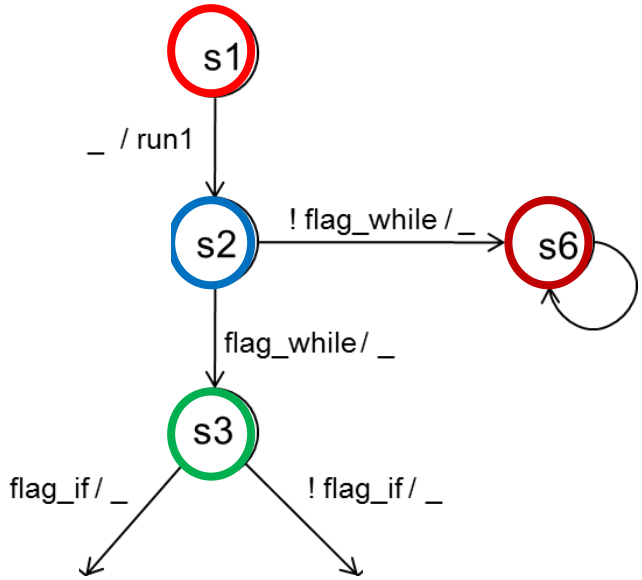
GCD Control Unit

```
1: int gcd(int a, int b) {
2:    while (a != b) {
3:       if (a > b)
4:          a = a - b;
         else
5:          b = b - a;
      }
6:    return a;
   }
```



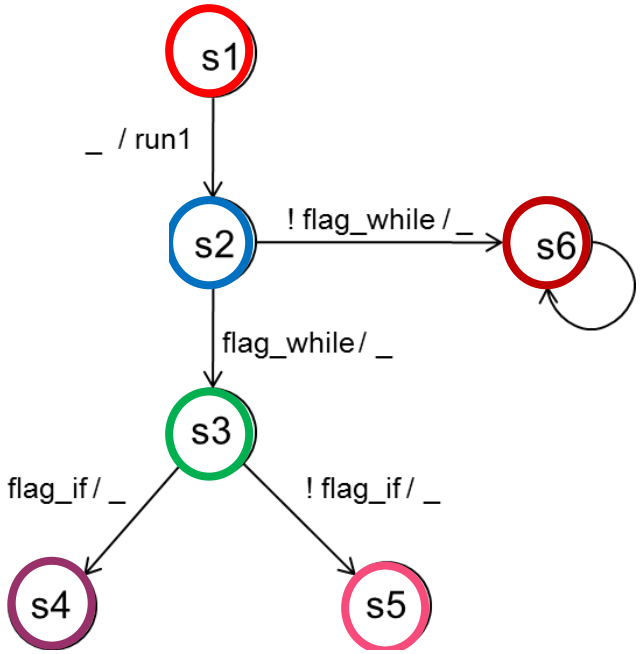
GCD Control Unit

```
1: int gcd(int a, int b) {
2:    while (a != b) {
3:       if (a > b)
4:          a = a - b;
         else
5:          b = b - a;
      }
6:    return a;
   }
```



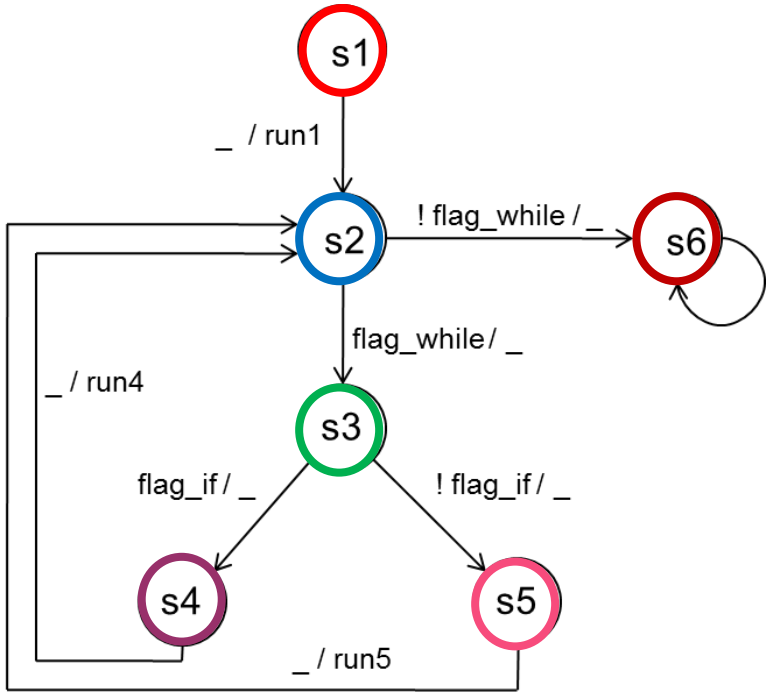
GCD Control Unit

```
1: int gcd(int a, int b) {
2:    while (a != b) {
3:       if (a > b)
4:          a = a - b;
         else
5:          b = b - a;
      }
6:    return a;
   }
```



GCD Control Unit

```
1: int gcd(int a, int b) {
2:    while (a != b) {
3:       if (a > b)
4:          a = a - b;
         else
5:          b = b - a;
       }
6:    return a;
    }
```



GCD Datapath

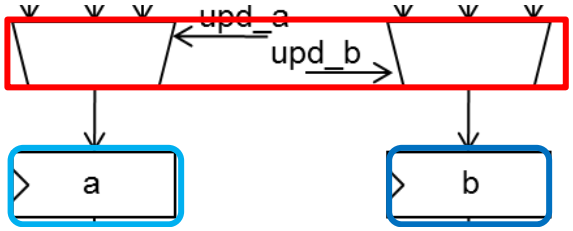
```
1: int gcd(int a, int b) {
2:   while (a != b) {
3:     if (a > b)
4:       a = a - b;
       else
5:       b = b - a;
     }
6:   return a;
   }
```

Variables → Registers

```
1: int gcd(int a, int b) {
2:   while (a != b) {
3:     if (a > b)
4:       a = a - b;
       else
5:       b = b - a;
     }
6:   return a;
   }
```

Who Feeds the Registers?

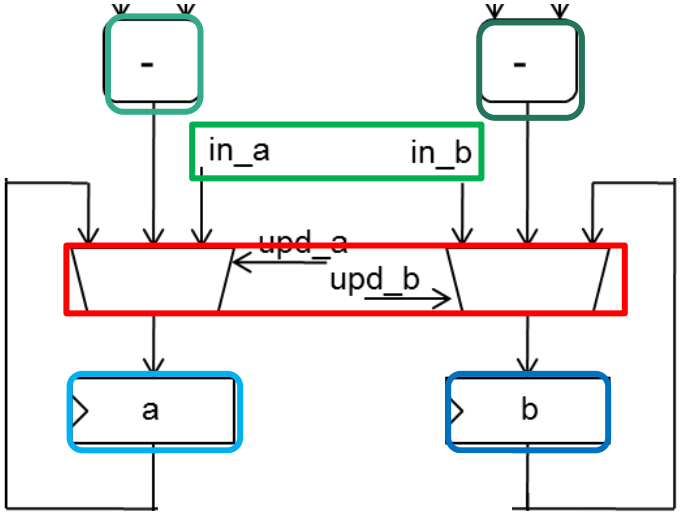
```
1: int gcd(int a, int b) {
2:   while (a != b) {
3:     if (a > b)
4:       a = a - b;
       else
5:       b = b - a;
     }
6:   return a;
   }
```



More than 1!

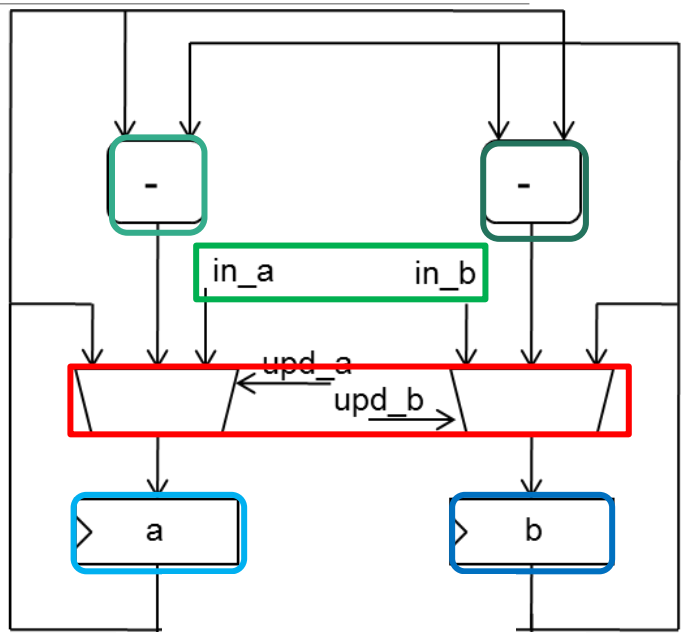
Sources!

```
1: int gcd(int a, int b) {
2:   while (a != b) {
3:     if (a > b)
4:       a = a - b;
       else
5:       b = b - a;
     }
6:   return a;
   }
```



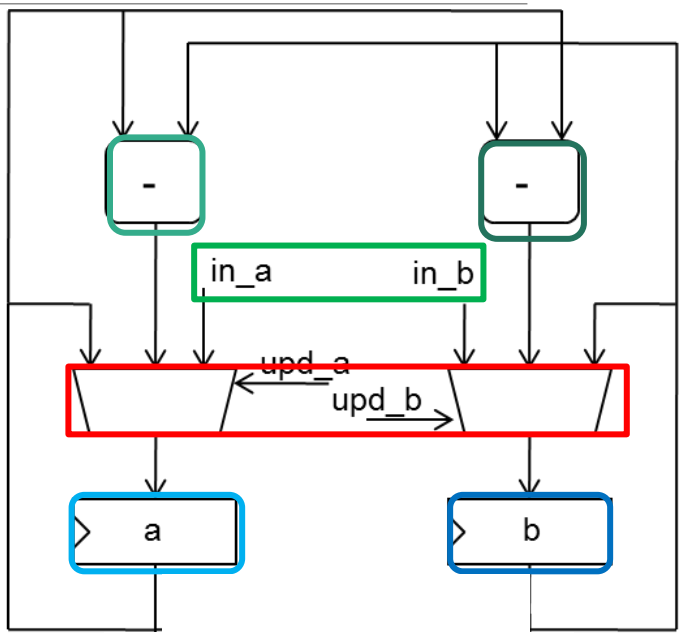
Functional Units

```
1: int gcd(int a, int b) {
2:   while (a != b) {
3:     if (a > b)
4:       a = a - b;
       else
5:       b = b - a;
     }
6:   return a;
   }
```



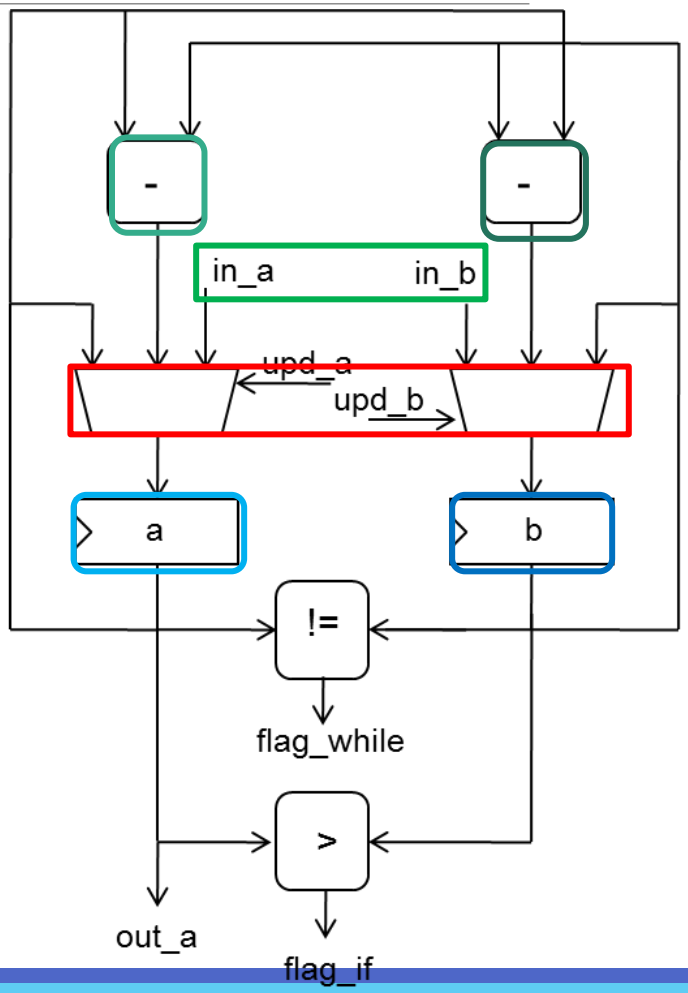
Generate Control Signals

```
1: int gcd(int a, int b) {
2:   while (a != b) {
3:     if (a > b)
4:       a = a - b;
       else
5:       b = b - a;
     }
6:   return a;
   }
```



Generate Control Signals

```
1: int gcd(int a, int b) {
2:   while (a != b) {
3:     if (a > b)
4:       a = a - b;
       else
5:       b = b - a;
     }
6:   return a;
   }
```

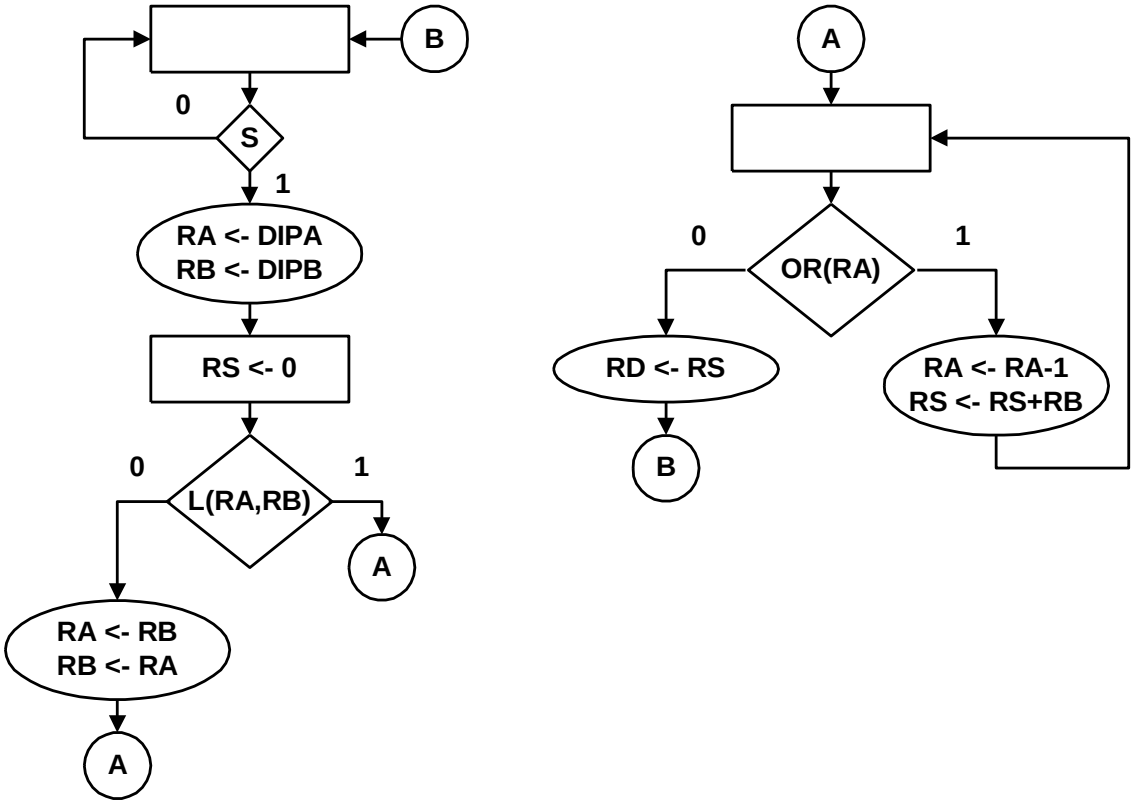


Samples

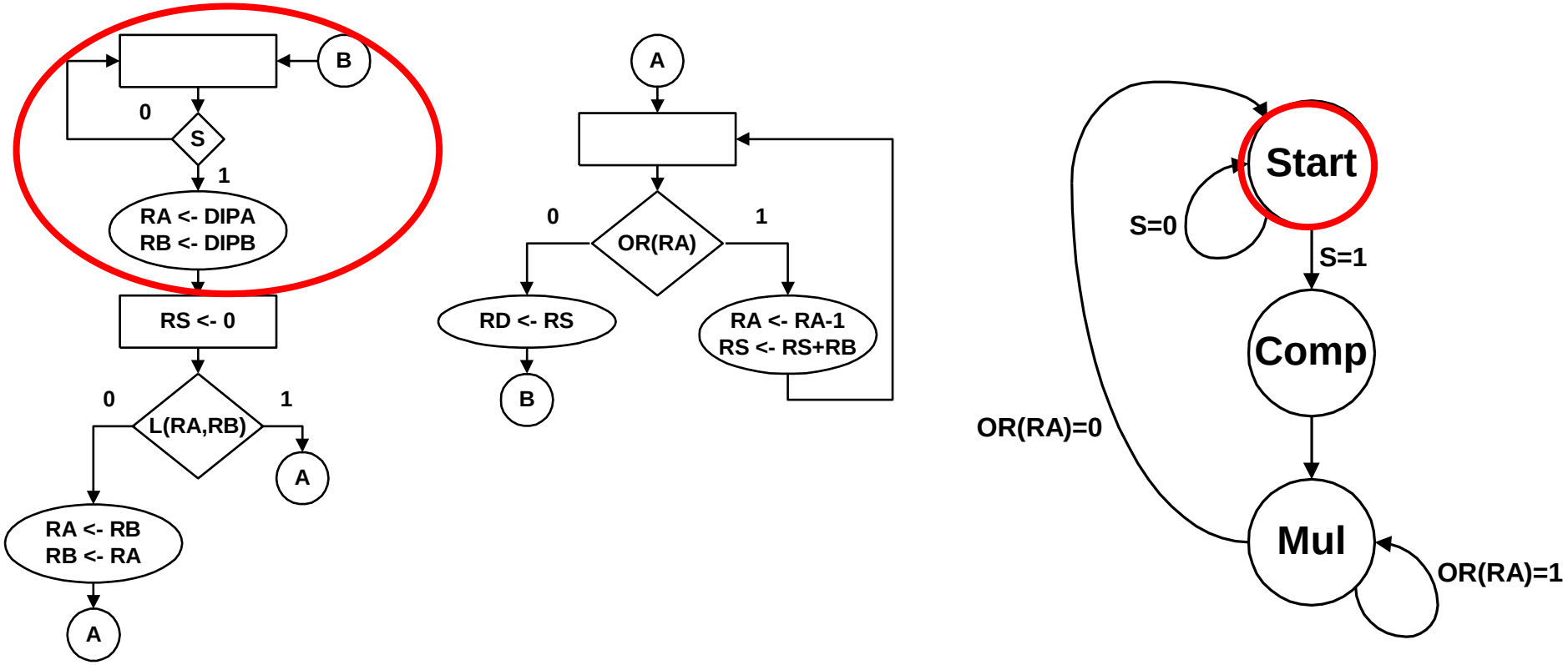
Sample 5

- ASM for multiplier?

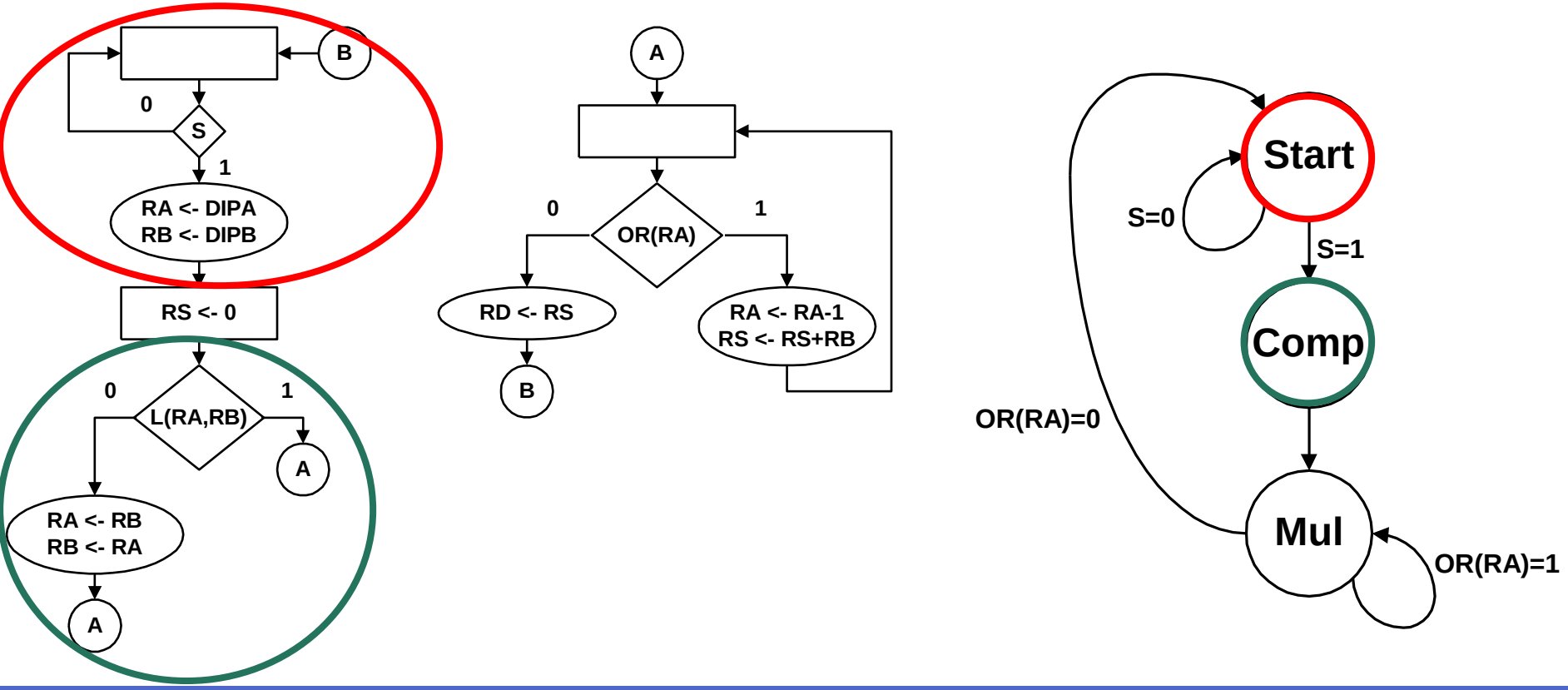
Multiplier ASM



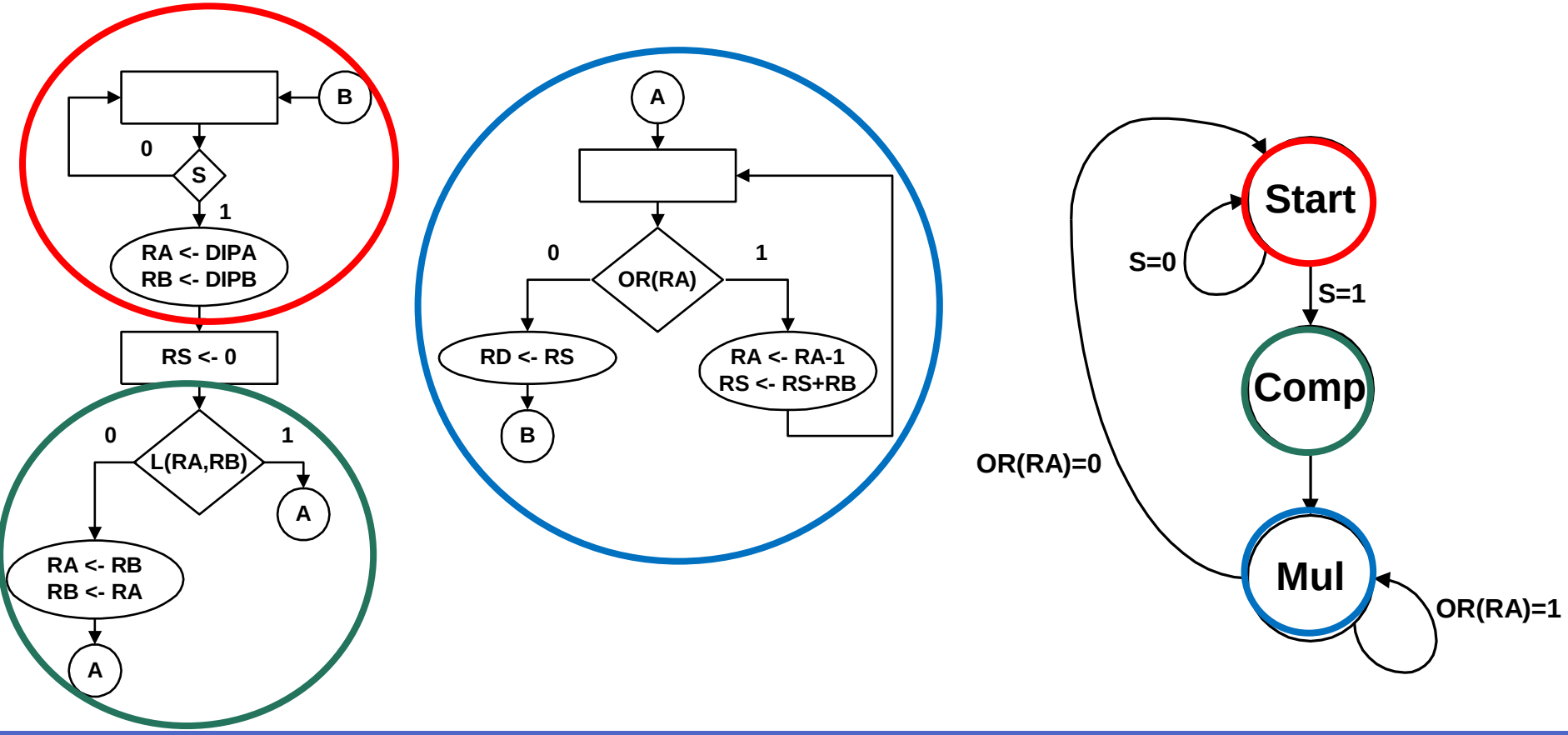
Multiplier: Control Unit



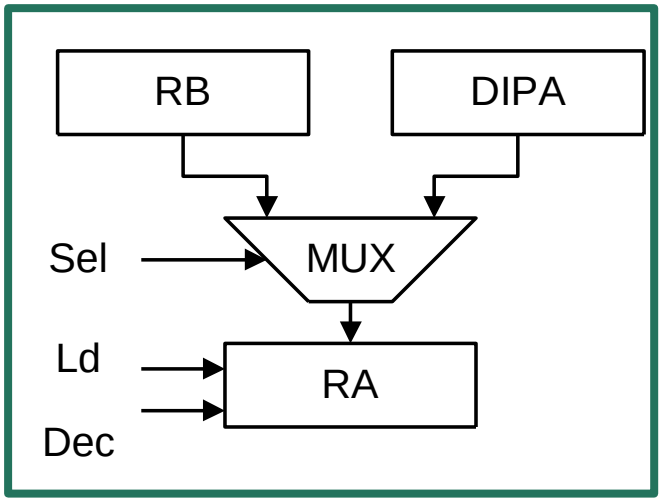
Multiplier: Control Unit



Multiplier: Control Unit



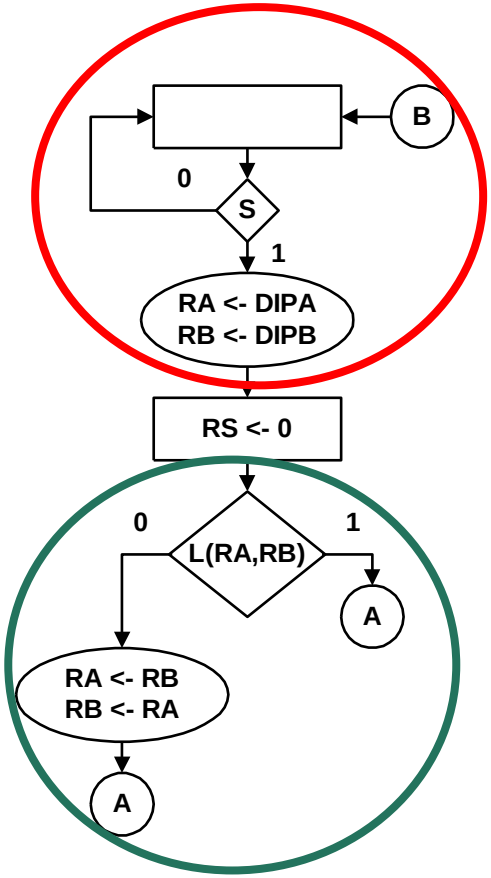
Multiplier Datapath

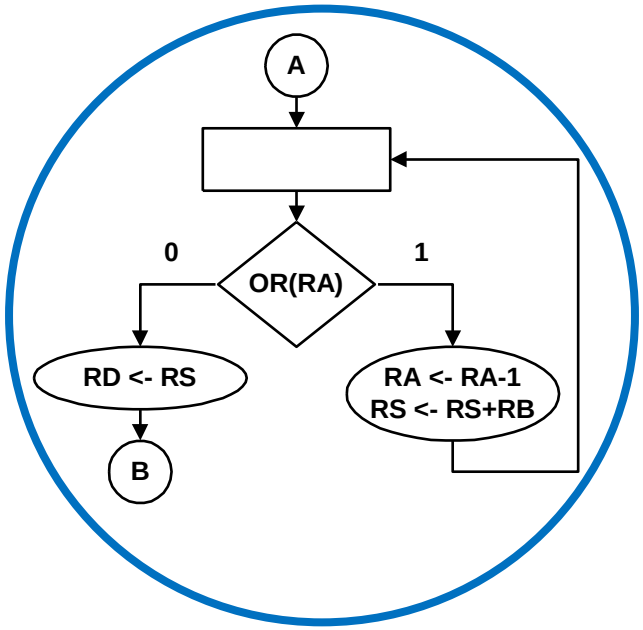


RA<-DIPA

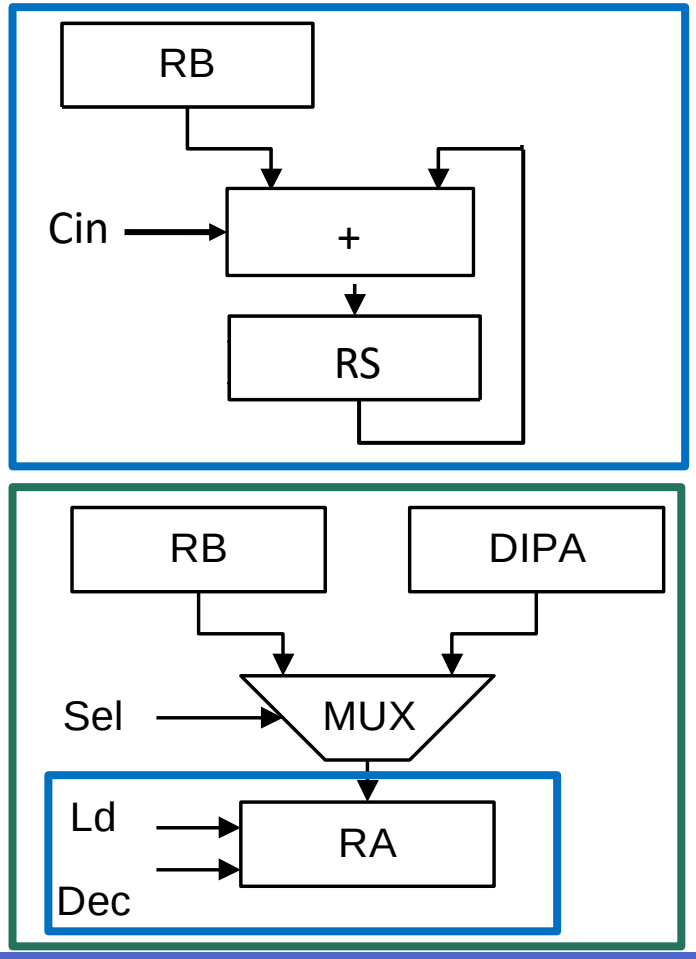
RA<-RB

RA<-RA-1

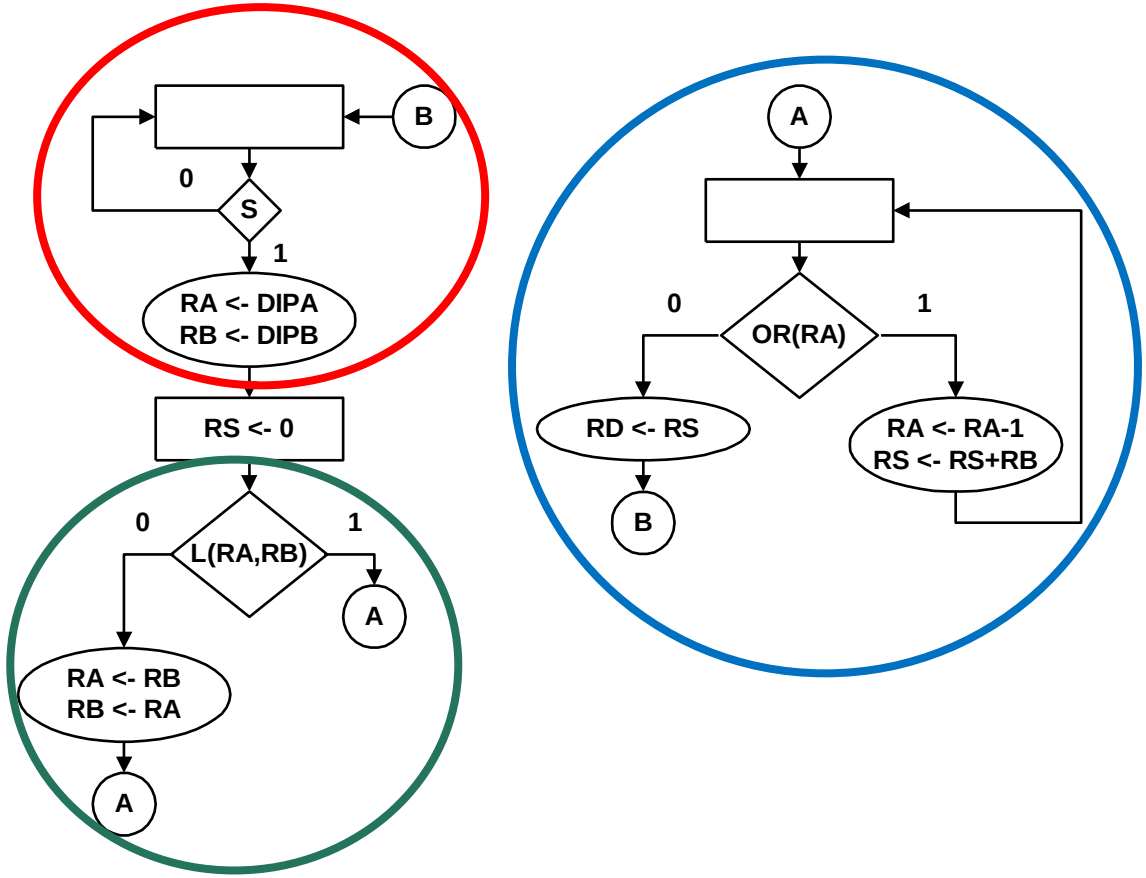




Multiplier Datapath

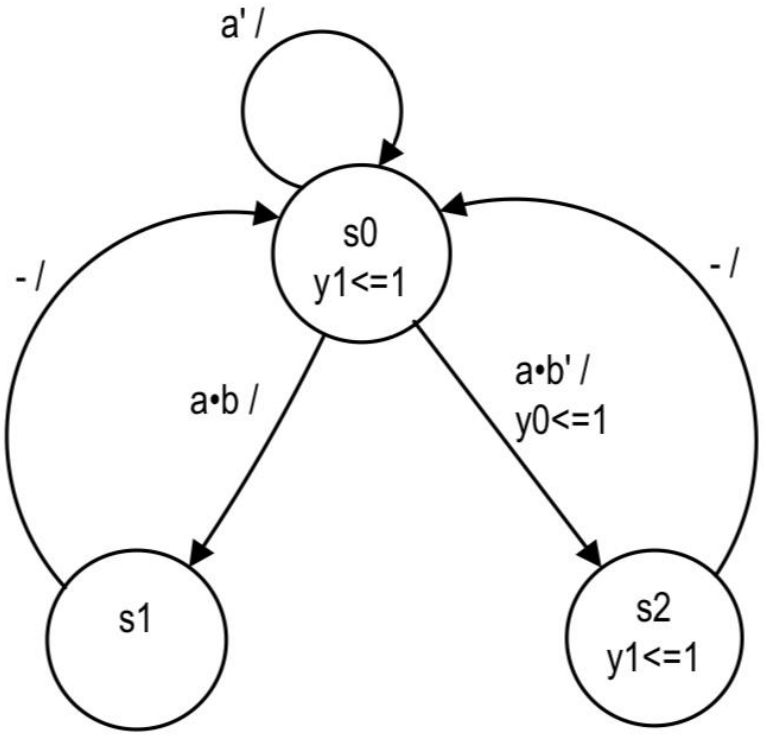


RS<- RS + RB



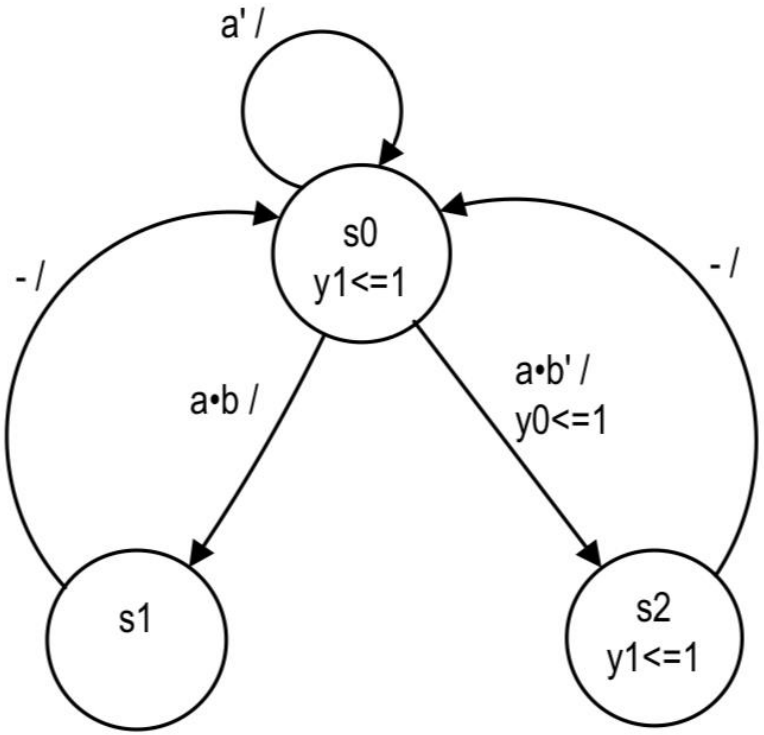
Sample 6

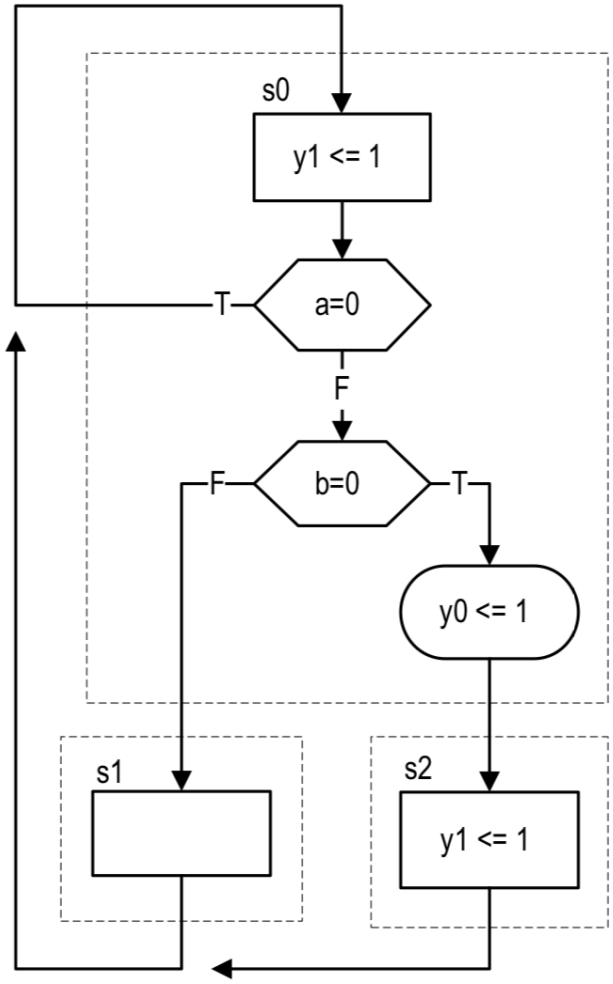
ASM chart for the below FSM?



ASM

ASM chart for the below FSM?





Thank You