

# *ASM Chart*

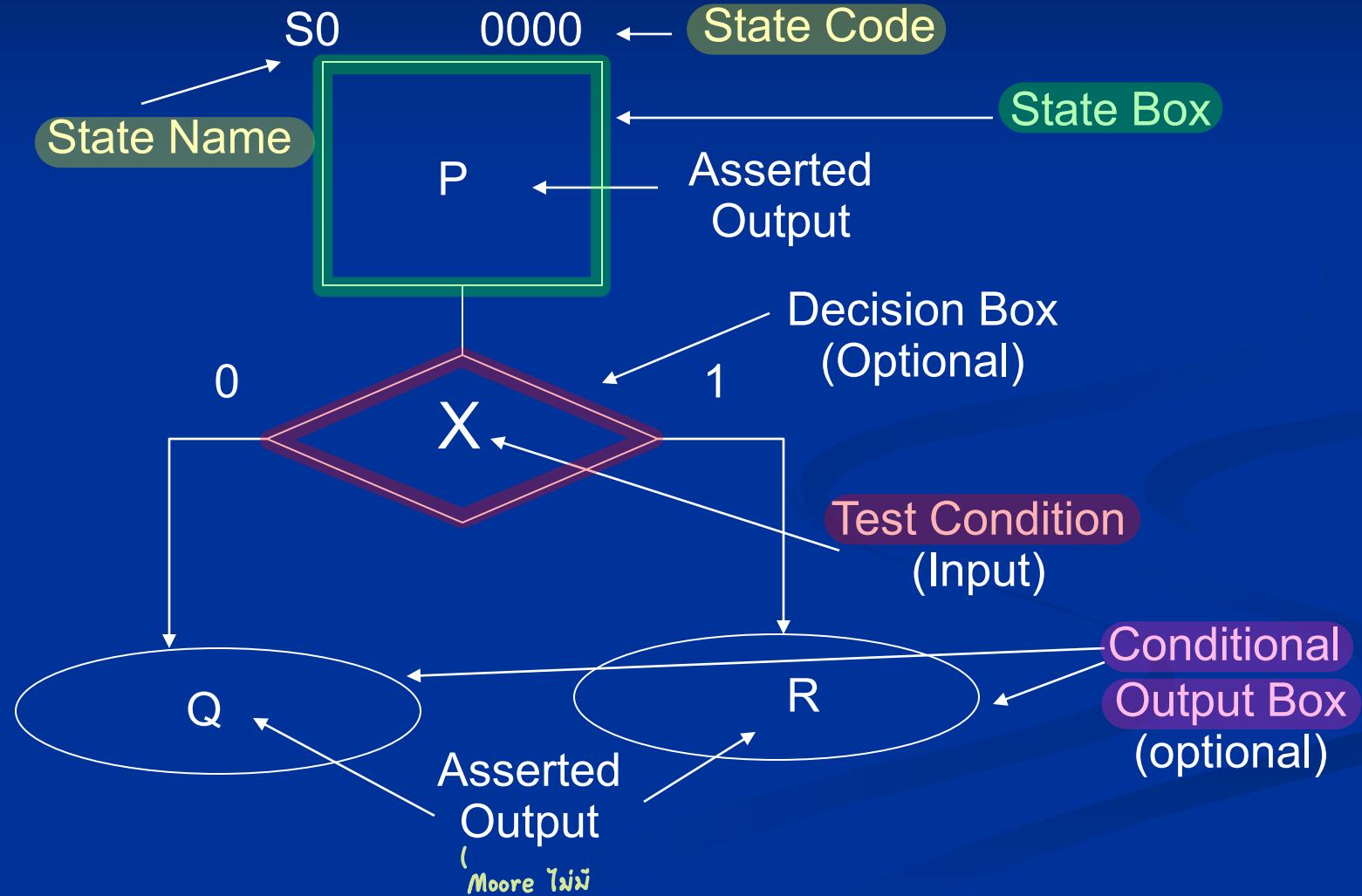
## *(Algorithmic State Machine)*

Road Map of hardware design

## *ASM Chart*

- T.E. Osborne (ที่บริษัท Hewlett Packard) เป็นผู้คิดค้นขึ้นในต้นทศวรรษ 1960
- จุดประสงค์เพื่อหา **algorithm** สำหรับแก้ปัญหาและสามารถเปลี่ยนเป็น **hardware** ได้
- ใช้สัญลักษณ์และรูปแบบใกล้เคียงกับ **flow chart** ที่ใช้ในการออกแบบ **software**
- ความแตกต่างระหว่าง flow chart และ ASM chart คือ
  - ASM Chart จะทำงานไปเรื่อยๆ ไม่มีการ **stop**
  - แต่ละ state ของ ASM chart ทำงานใน 1 clock เท่านั้น

# *State (1 clock)* ประกอบด้วย

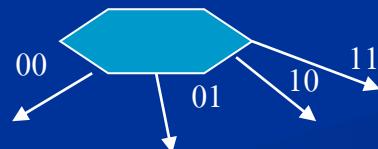


## *State Box*

- ต้องมีในทุก state
- แสดง state output หรือ Moore type output ใน box (ถ้ามี)
  - ถ้าเป็น 1 bit แสดงชื่อ output นั้นใน state ที่ output นั้นถูก assert ใช้ .h หรือ .l ตามหลังชื่อเพื่อแสดงว่า asserted high หรือ asserted low เช่น P.h แสดงว่า P asserted high ถ้าไม่มี output ใน state ใด ถือว่า output ไม่ถูก assert ใน state นั้น
  - ถ้ามีหลาย bits ใช้เครื่องหมาย = ต้องแสดงค่าในทุก state เช่น A=001
  - ใช้เครื่องหมาย  $\leftarrow$  (register transfer) แสดงถึง output ที่ส่งไปควบคุมการทำงานของ sequential device อื่น (data path) ซึ่งจะทำใน clock ถัดไป count  $\leftarrow$  count + 1     $\text{count} = \text{count} + 1$  ใน state ถัดไป

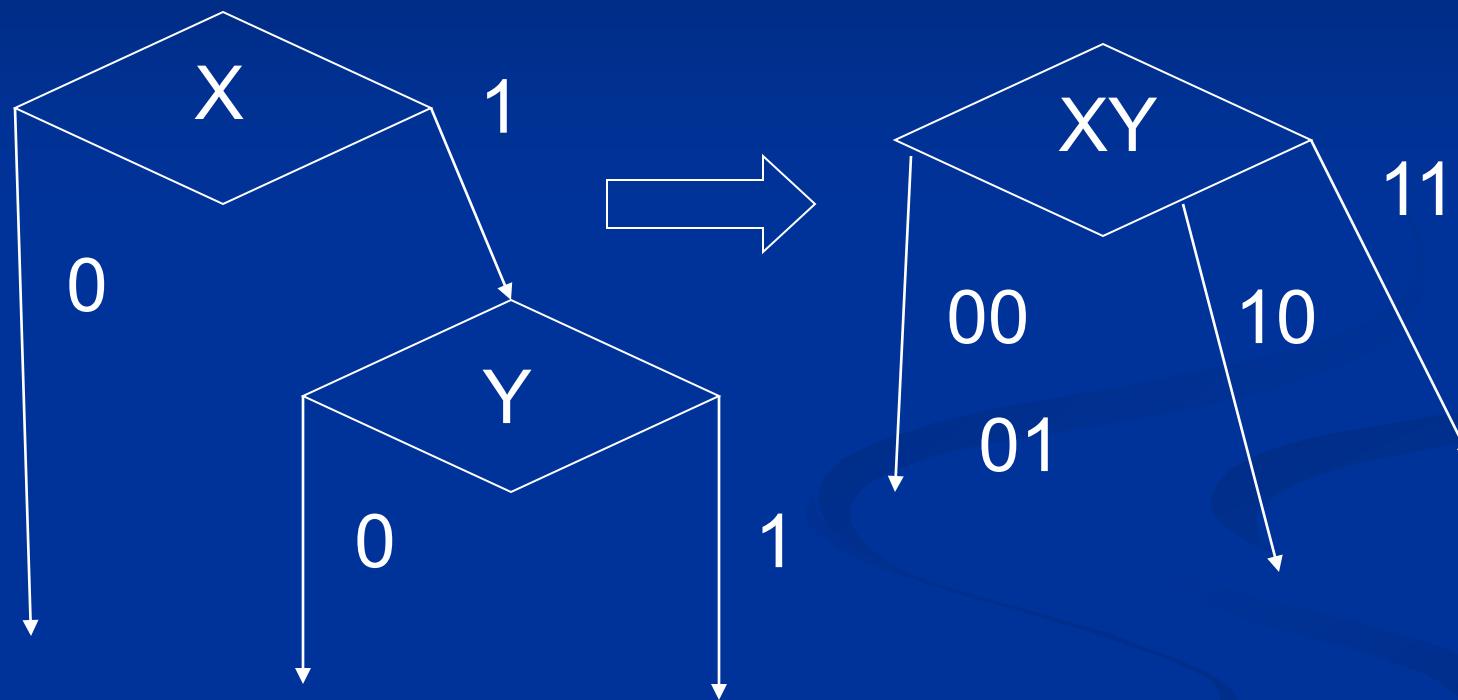
## *Decision Box*

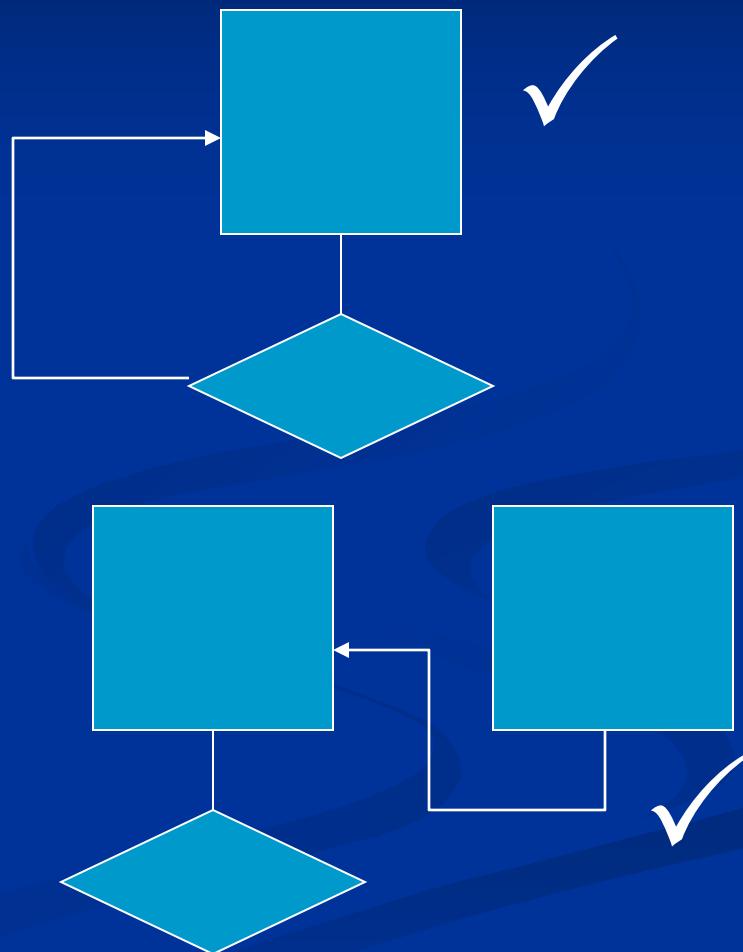
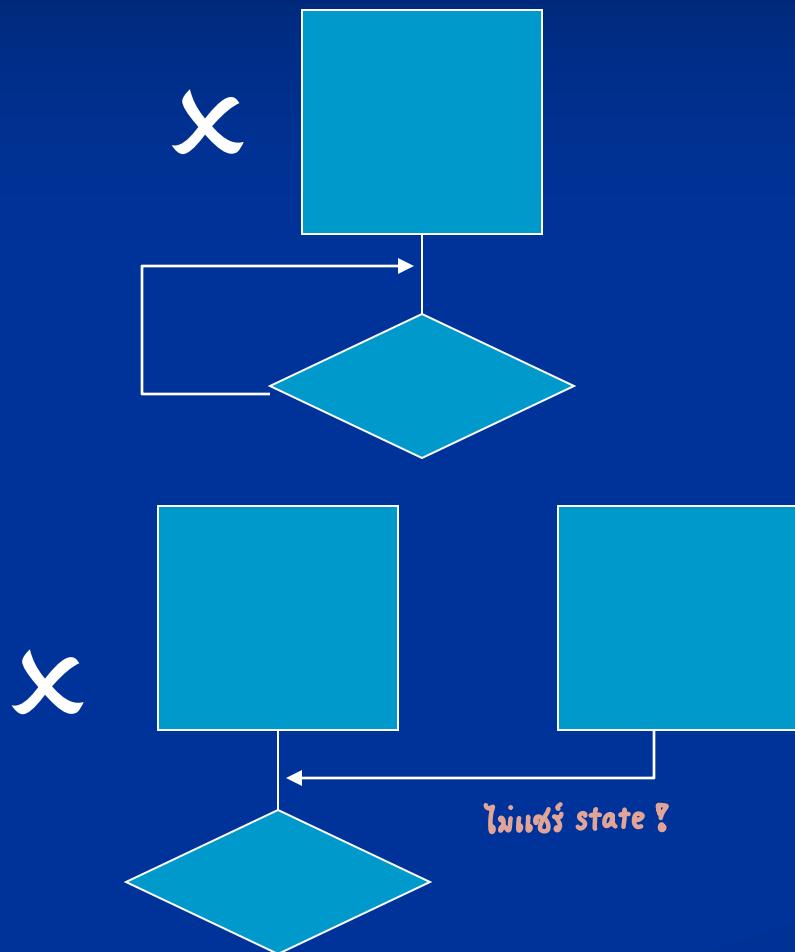
- อาจจะไม่มีหรือมีมากกว่า 1 อัน ได้
- ใช้รูป  หรือ 
- ภายในแสดงค่าที่ใช้ในการตัดสินใจ เพื่อกำหนด next state โดยมากเป็นค่า input
- ในกรณีที่ค่าที่ใช้ตัดสินใจมีหลาย bits อาจแสดงทางเลือกได้มากกว่า 2 ทาง เช่น
- ในกรณีที่มีมากกว่า 1 อันทุกอันทำใน clock เดียวกัน ดังนั้นไม่มี order ของการทำงาน



## *Conditional Output Box*

- อาจจะมีหรือไม่ก็ได้
- ใช้แสดง output ที่ขึ้นกับ condition หรือ Mealy type output
- รูปแบบของการแสดง output ใช้ระบบเดียวกับ State box





# ការធំងនពុម្ពយោ

**A ← B**

$A = B$

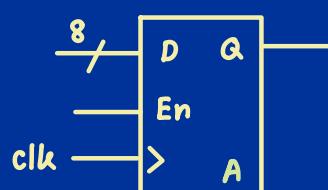
**Load Register**

**A ← A + 1**

**Counter count up  
(down)**

**A ← A + C**

**Accumulator**

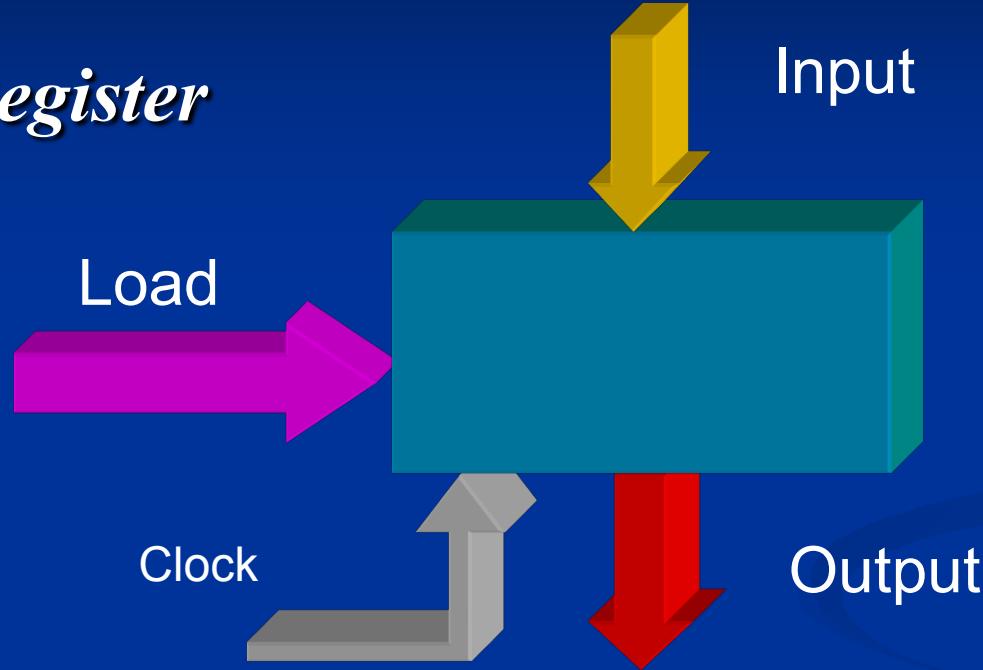


En	D	$Q^+$
0	D	Q
1	D	$Q^+$

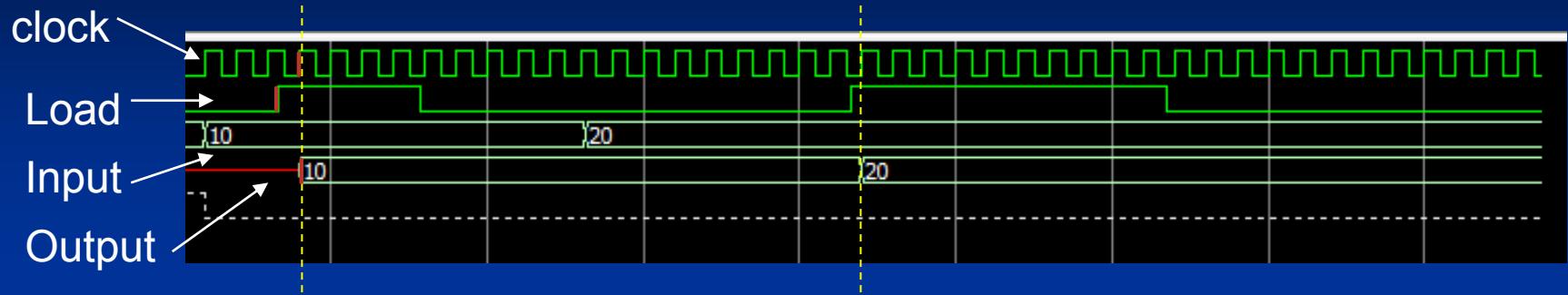
# ความหมายของ



*Register*



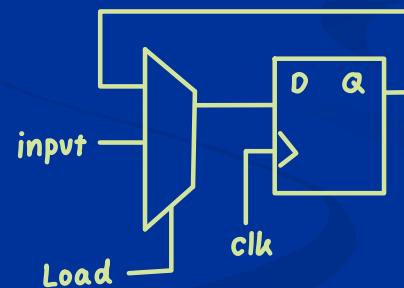
เป็นอุปกรณ์ Sequential (มี clock) เมื่อมีสัญญาณ Load (Synchronous) แล้วมี clock จะทำให้ Input ถูกเก็บใน register และ Output จะเป็นค่าที่เก็บใน register

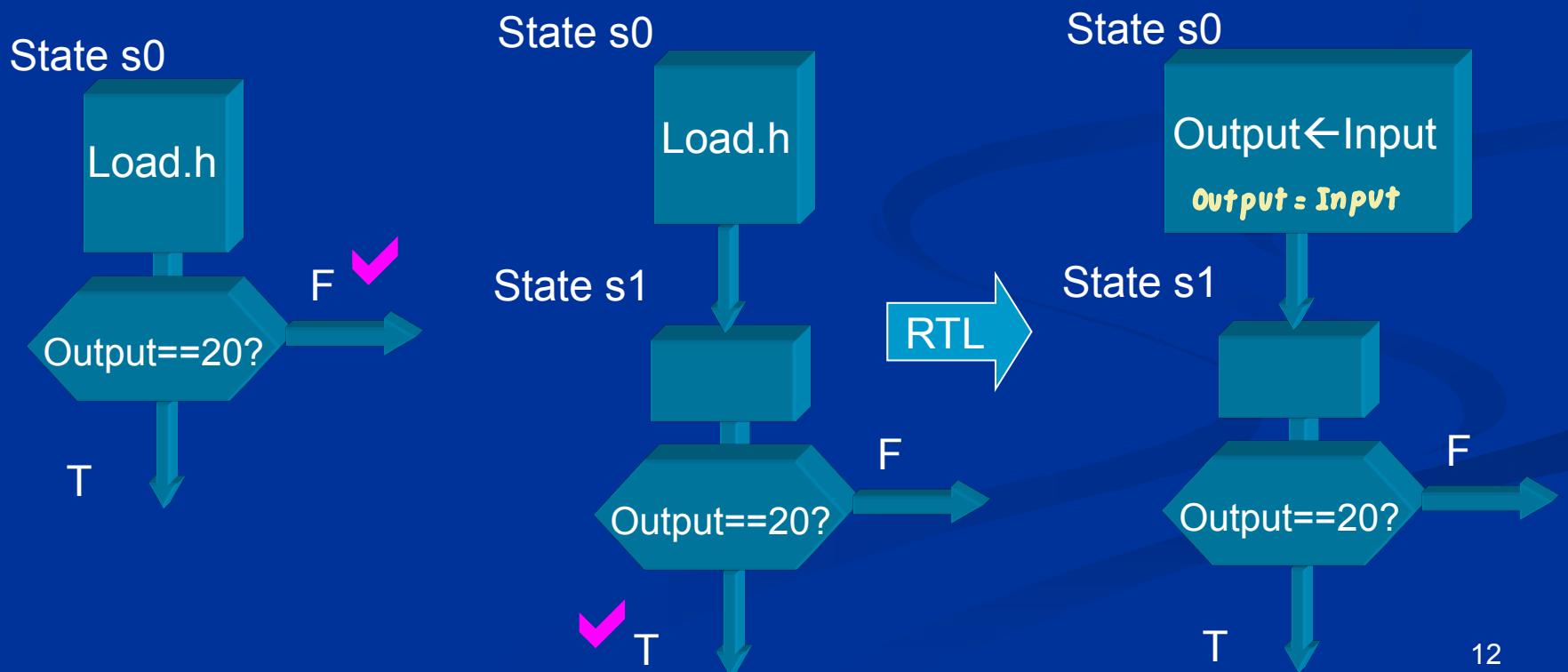


เนื่องจาก output จะเปลี่ยน หลังจาก clock คือ เปลี่ยน หลังจาก state เปลี่ยน ใน asm chart จะใช้ สัญลักษณ์

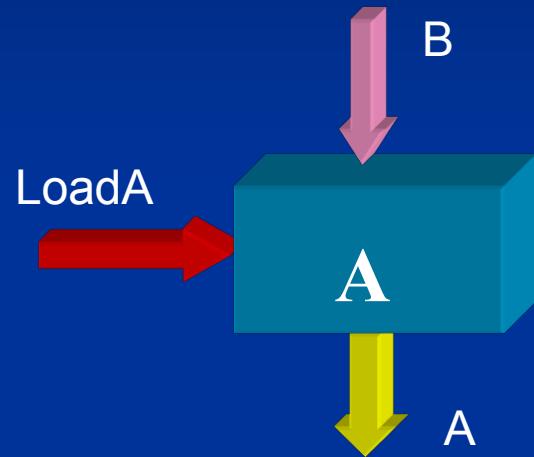
เช่น Output ← Input

← แทน =



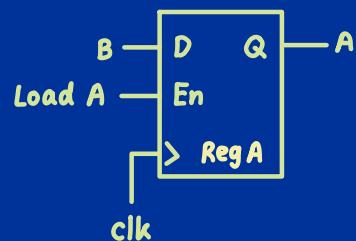


# រូបរាងកិច្ចការណ៍ (Convention) នៃ Register



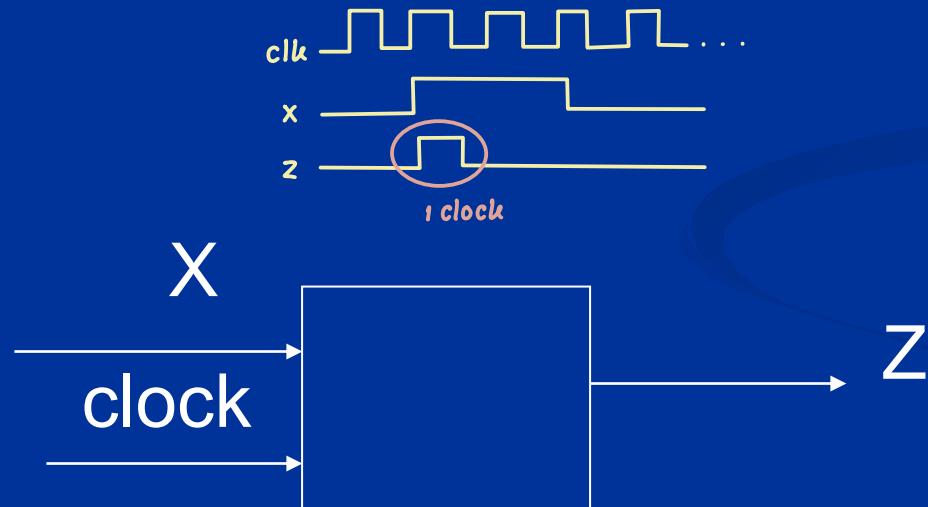
RTL

$A \leftarrow B$

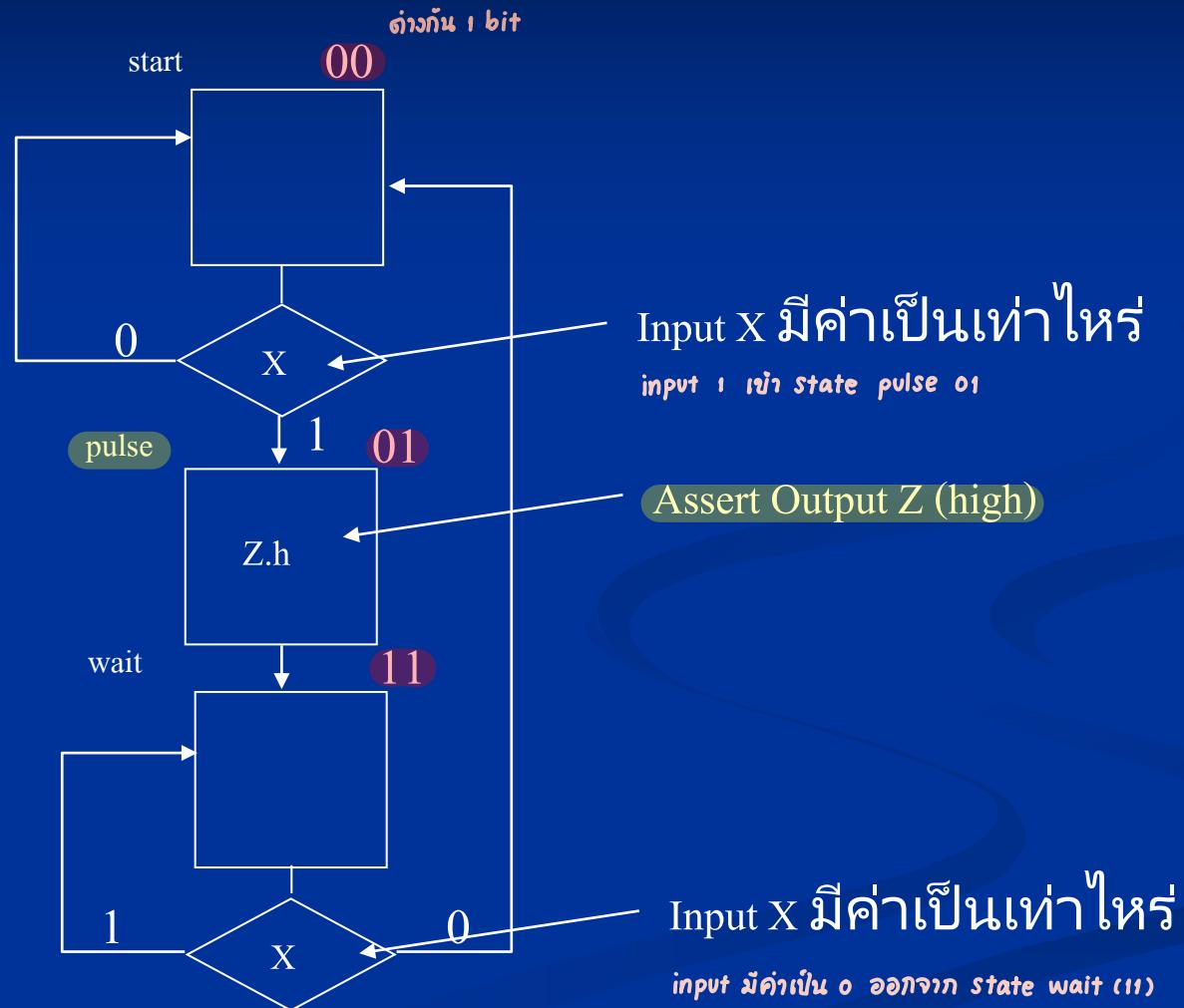


## ตัวอย่าง *Single Pulser*

- Output จะเป็น 1 เมื่อ input เป็น 1 และเป็น 1 อยู่ 1 clock แล้วเป็น 0 จนกว่า input จะเป็น 0 แล้วกลับเป็น 1 ใหม่



# ASM Chart แบบ Moore



## *Implement FSM โดยใช้ MUX*

- ถ้าใช้ D Flip-flop สร้าง FSM

input ของ d flip-flop จะเป็น Next State ( $Q+$ ) ไม่ต้องเทียบกับ Present State ( $Q$ )

- ใช้ Mux เพื่อหา D โดยใช้ Present State เป็นตัว select ของ Mux
- ใช้ “เงื่อนไข” ที่ทำให้ D เป็น 1 เป็น input ของ Mux

Transition Table

# *Implement FSM โดยใช้ MUX. (2)*

Present State QaQb	Next State Qa+Qb+	Condition
0 0	0 0	X=0
	0 1	X=1
	1 1	None
1 1	1 1	X=1
	0 0	X=0

# *Implement FSM โดยใช้ MUX. (3)*

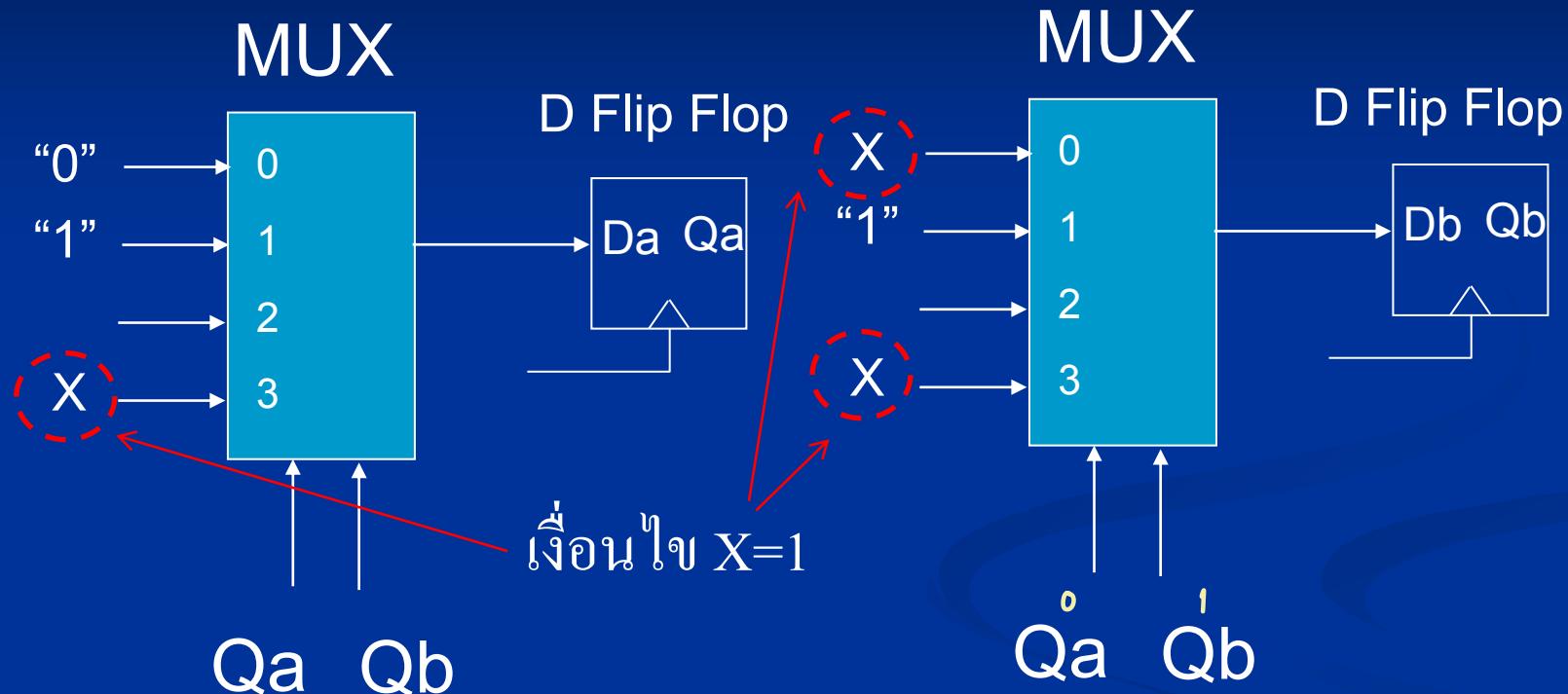
Present State	Next State	Condition
QaQb	QaQb	
0 0	0 0	X=0
	0 1	X=1
0 1	1 1	none
1 1	1 1	X=1
	0 0	X=0

ใช้กับ D Flip Flop

Qa+ เป็น 1 เมื่อ  
 $QaQb=01$  ไม่มีเงื่อนไข  
 $QaQb=11$  เงื่อนไข  $X=1$

Qb+ เป็น 1 เมื่อ  
 $QaQb=00$  เงื่อนไข  $X=1$   
 $QaQb=01$  ไม่มีเงื่อนไข  
 $QaQb=11$  เงื่อนไข  $X=1$

# *Implement FSM โดยใช้ MUX. (4)*



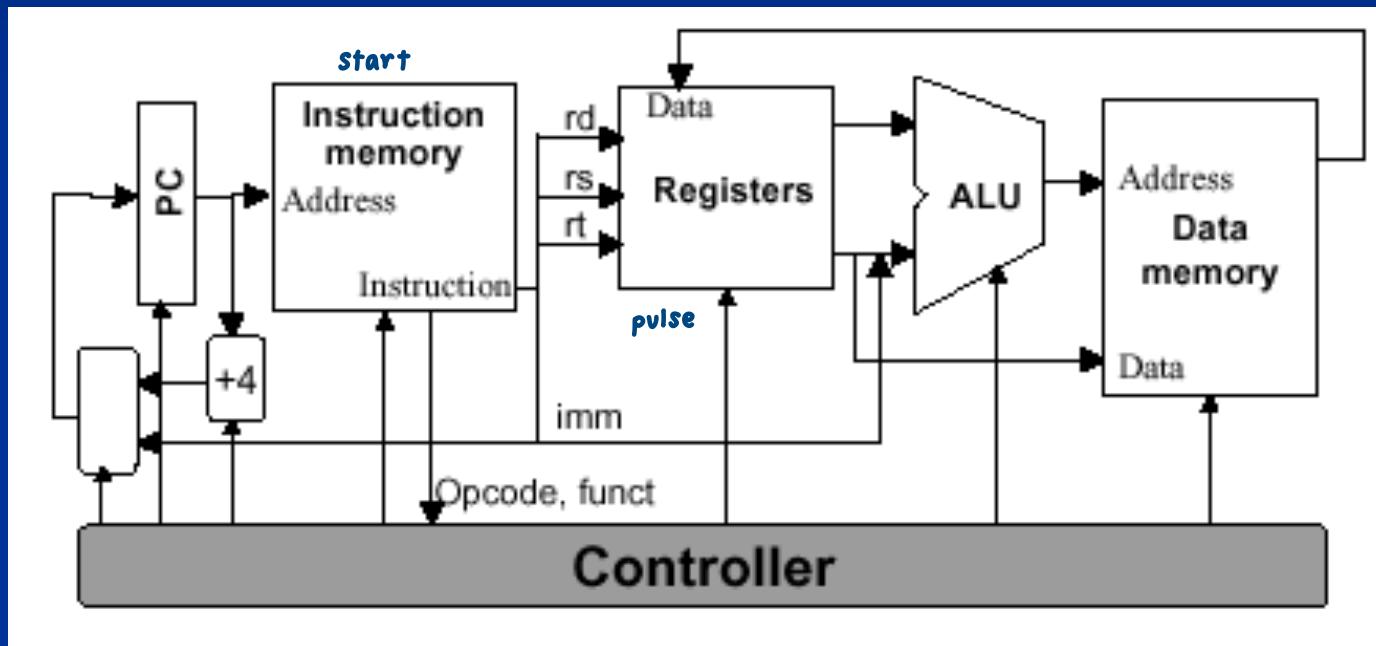
ไม่มีเงื่อนไข  $\rightarrow "1"$

$$Z = \underline{Q_a \bullet Q_b} \quad (Z.h \text{ ใน State 01})$$

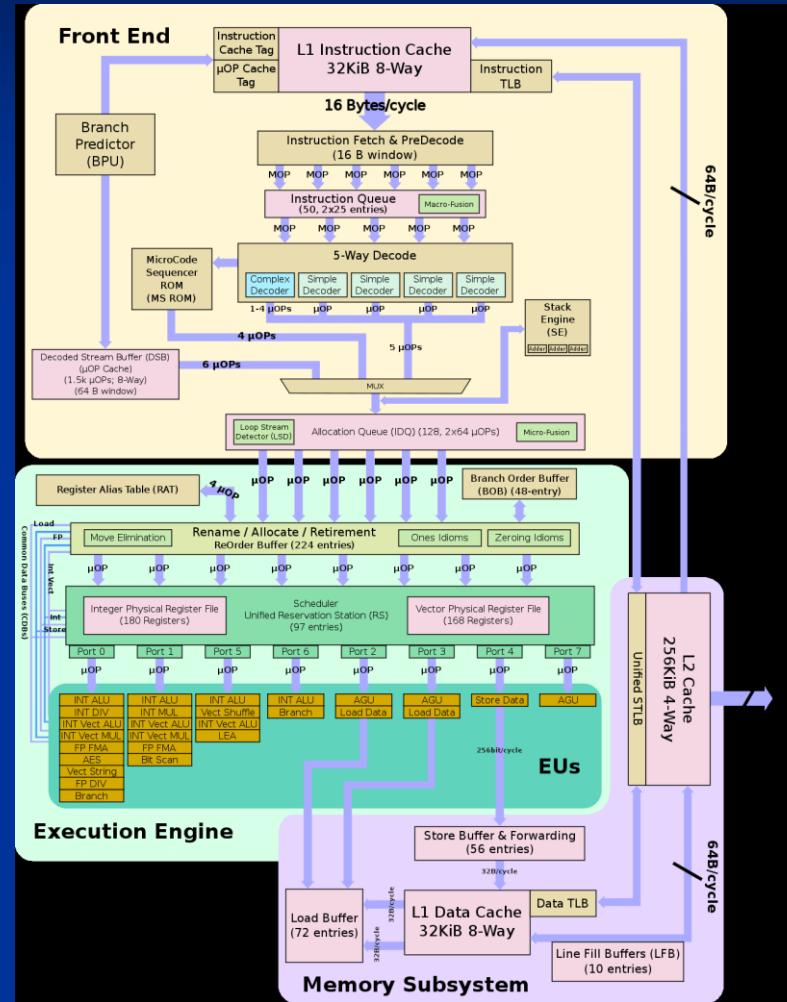
## *Datapath และ Control Unit*

- Data path เป็นส่วน “ทำงาน” ของวงจรประกอบด้วยวงจรพื้นฐานทั่วไป เช่น Adder, Comparator, Mux, Register, Counter
- Control Unit เป็นส่วนของวงจรที่สร้างสัญญาณเพื่อควบคุมการทำงานของ Datapath

## *Example: Single Cycle CPU datapath*



# Example: Intel Skylake Datapath (Subset)



# สรุป

- ต้องมี Algorithm ก่อน
- ASM Chart “บรรยาย” Algorithm เป็นแบบ Hardware (Synchronous Sequential) : ทำงานตาม clock
- FSM ที่ได้จาก ASM Chart คือ Control Unit
- การทำงานใน State Box ทำโดย data path โดยมี สัญญาณควบคุมจาก Control Unit
- ส่วน Decision Box เป็นการตรวจสอบ Input

# ຕຳອຍ່າງ: $X/Y$ (*childish Algorithm*)

ແບ່ງເຫຼື້ອ 11 ອັນ ເປັນ 3 ກອງ

(11/3)



# ຕົວອຍ່າງ: $X/Y$ (*childish Algorithm*)

$Q=0$

while ( $X \geq Y$ ) {

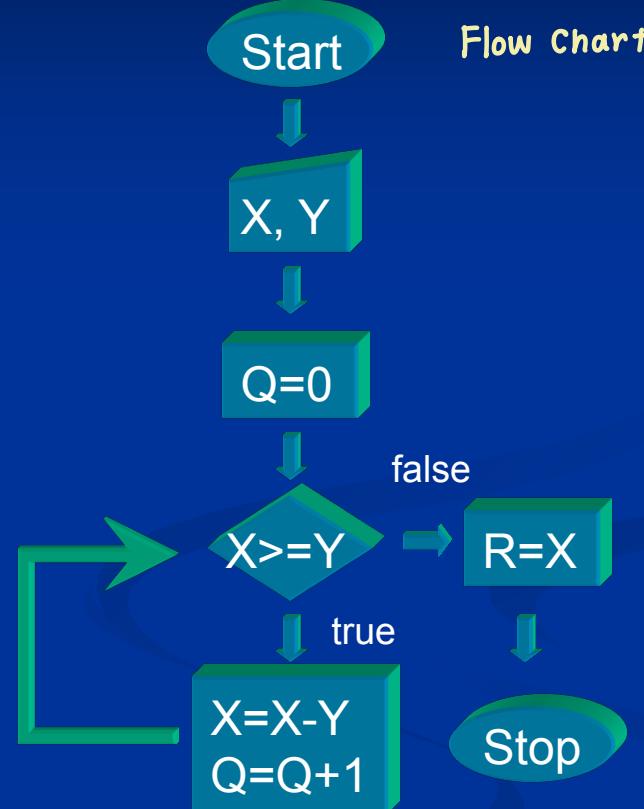
$X=X-Y$

$Q=Q+1$

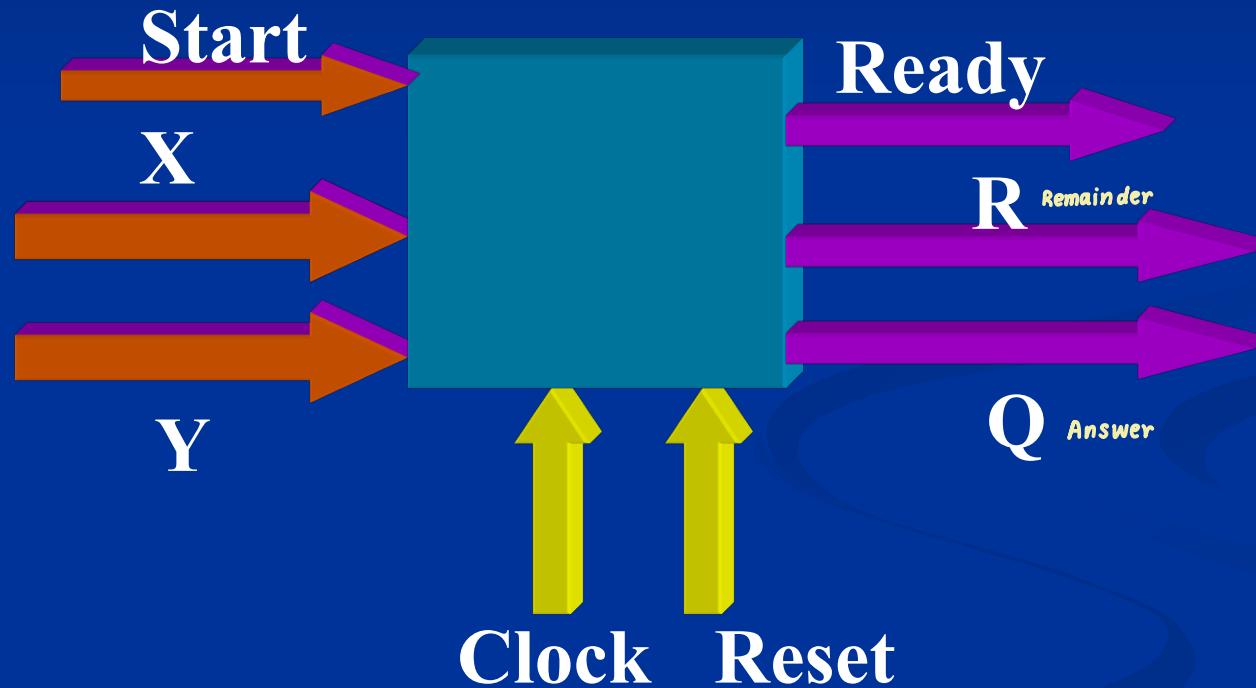
}

$R=X$  //Remainder

//  $Q$  is the Answer



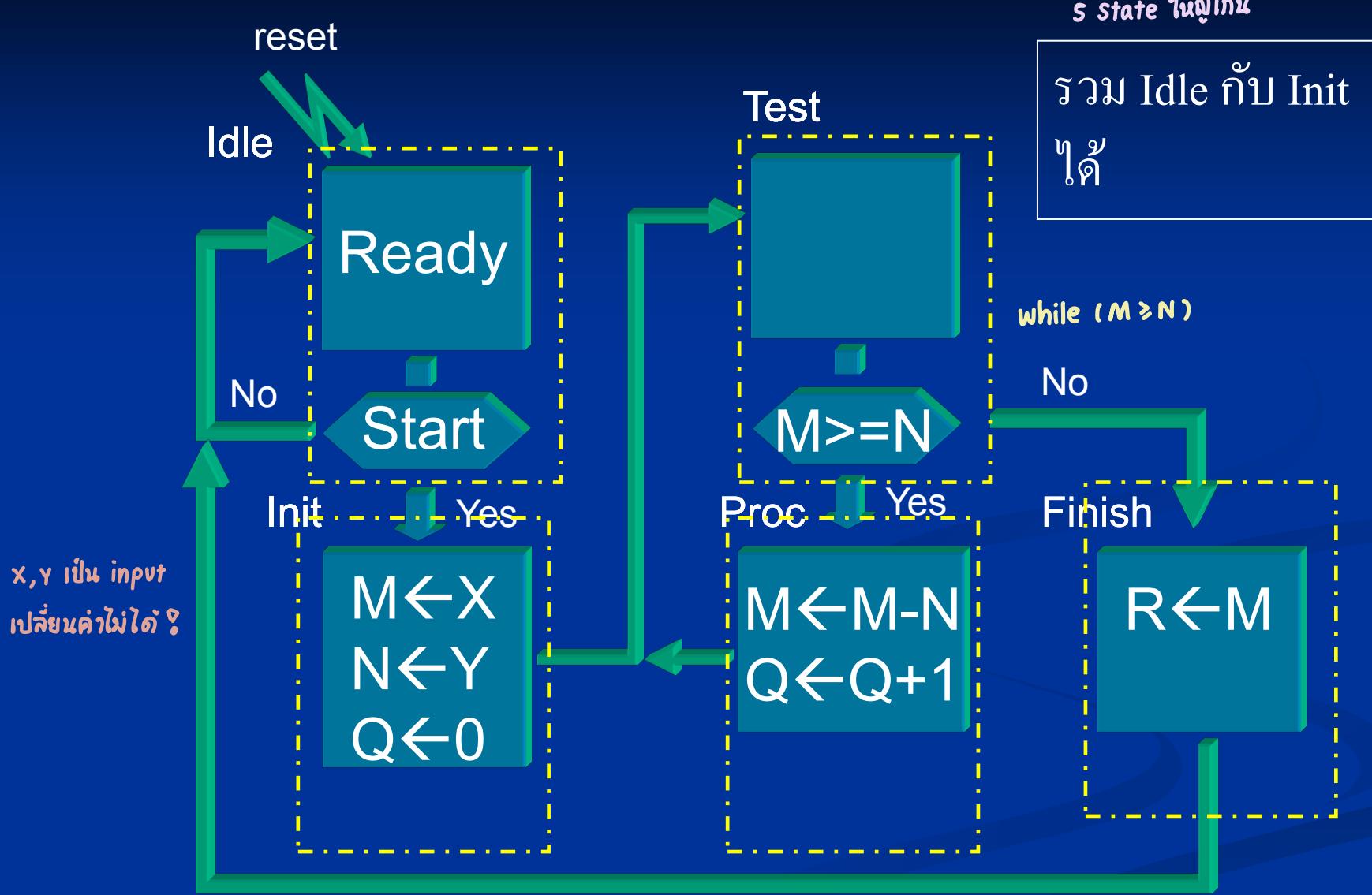
# *Block Diagram of the Circuit*

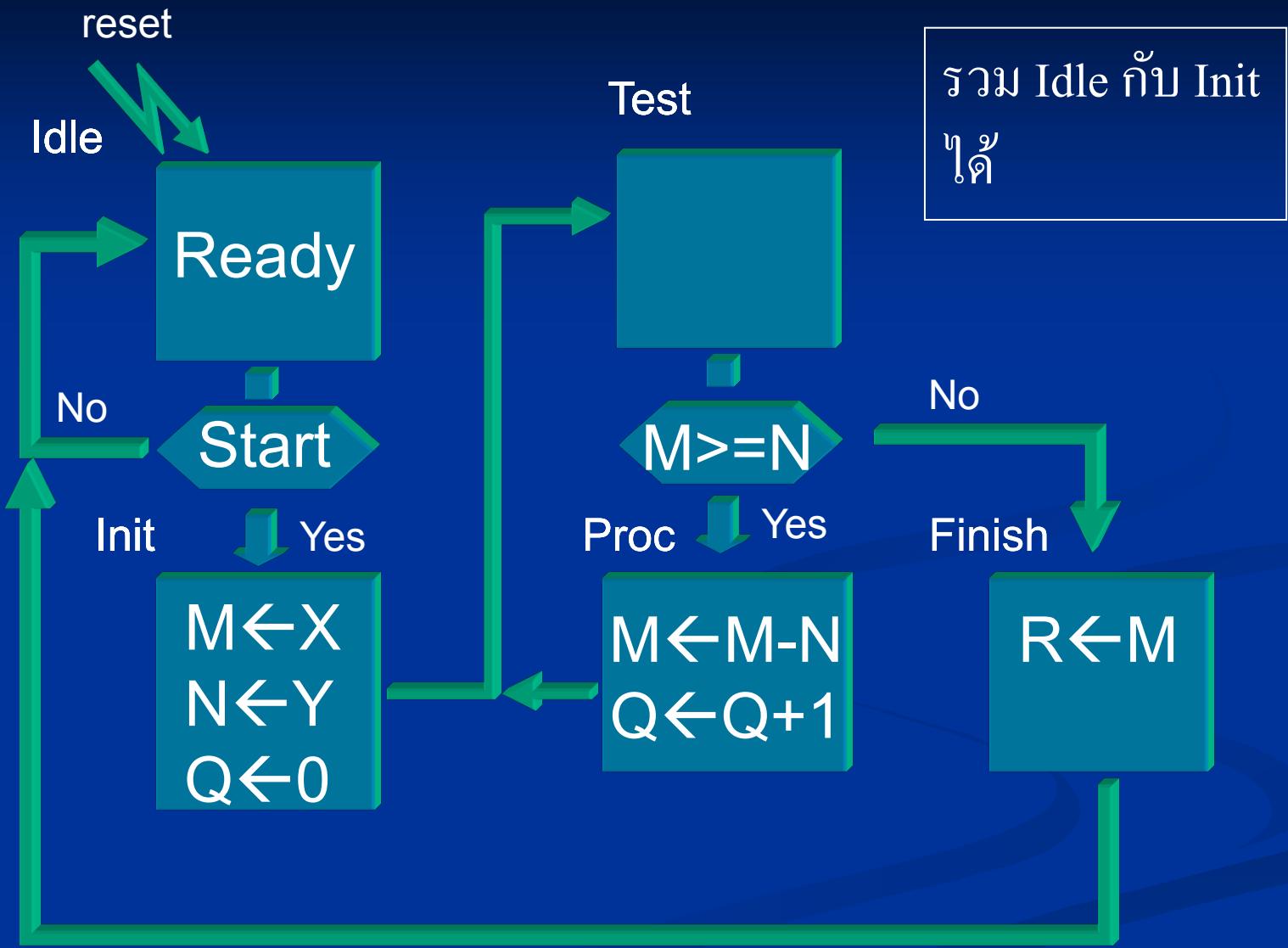


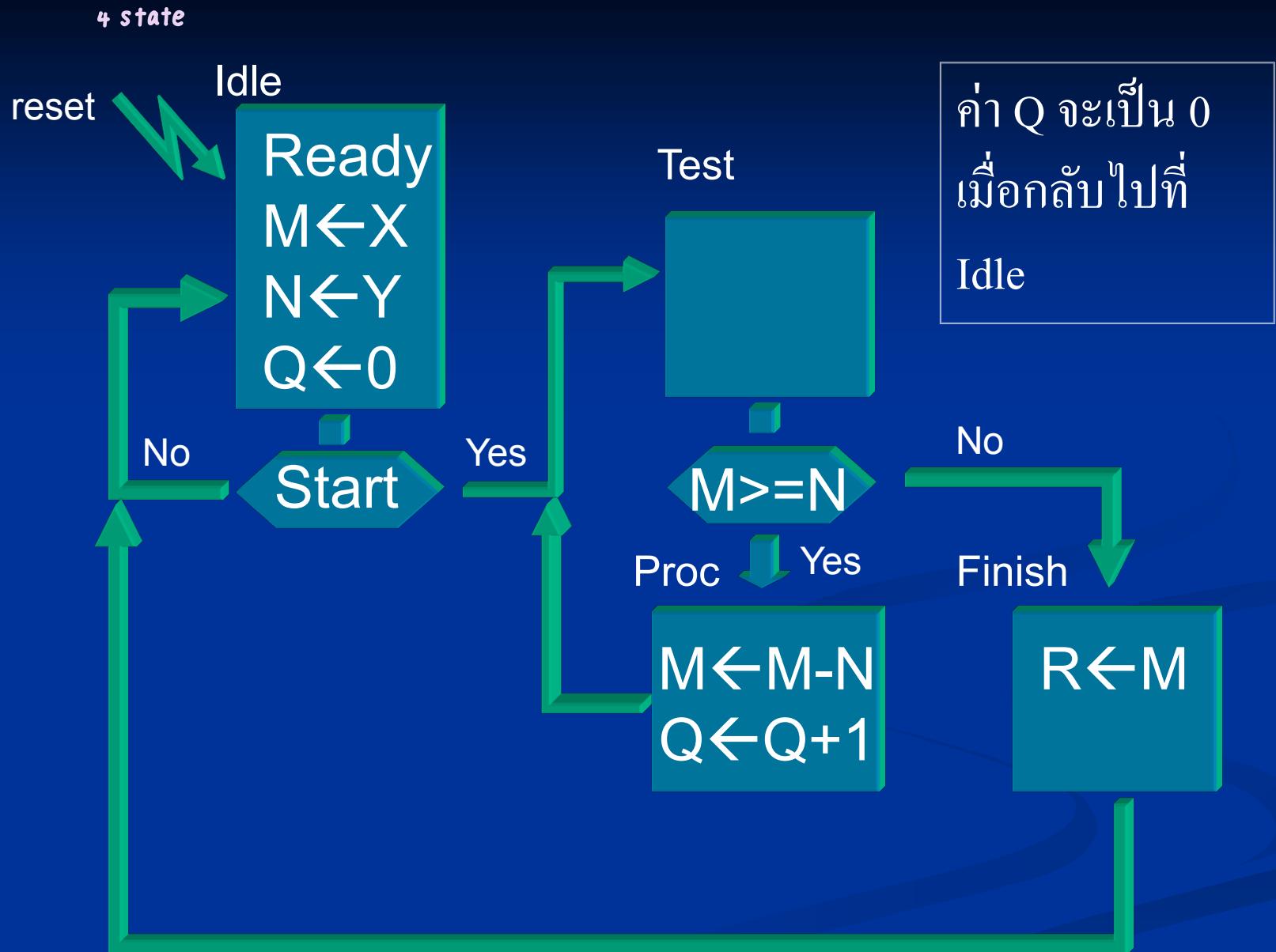
## แนวคิดทาง *Hardware*

จาก  $X=X-Y$  แสดงว่าต้องเปลี่ยนค่าของ  $X$  ซึ่งเป็น Input  
แต่เมื่อเป็น Hardware Input คือ “สัญญาณ” จากนอก  
วงจร ซึ่งวงจรทำไม่ได้

วงจรต้อง “เก็บ” ไว้ภายใน “ส่วนความจำ” ของวงจร  
ส่วนความจำอาจเป็น RAM หรือ Register

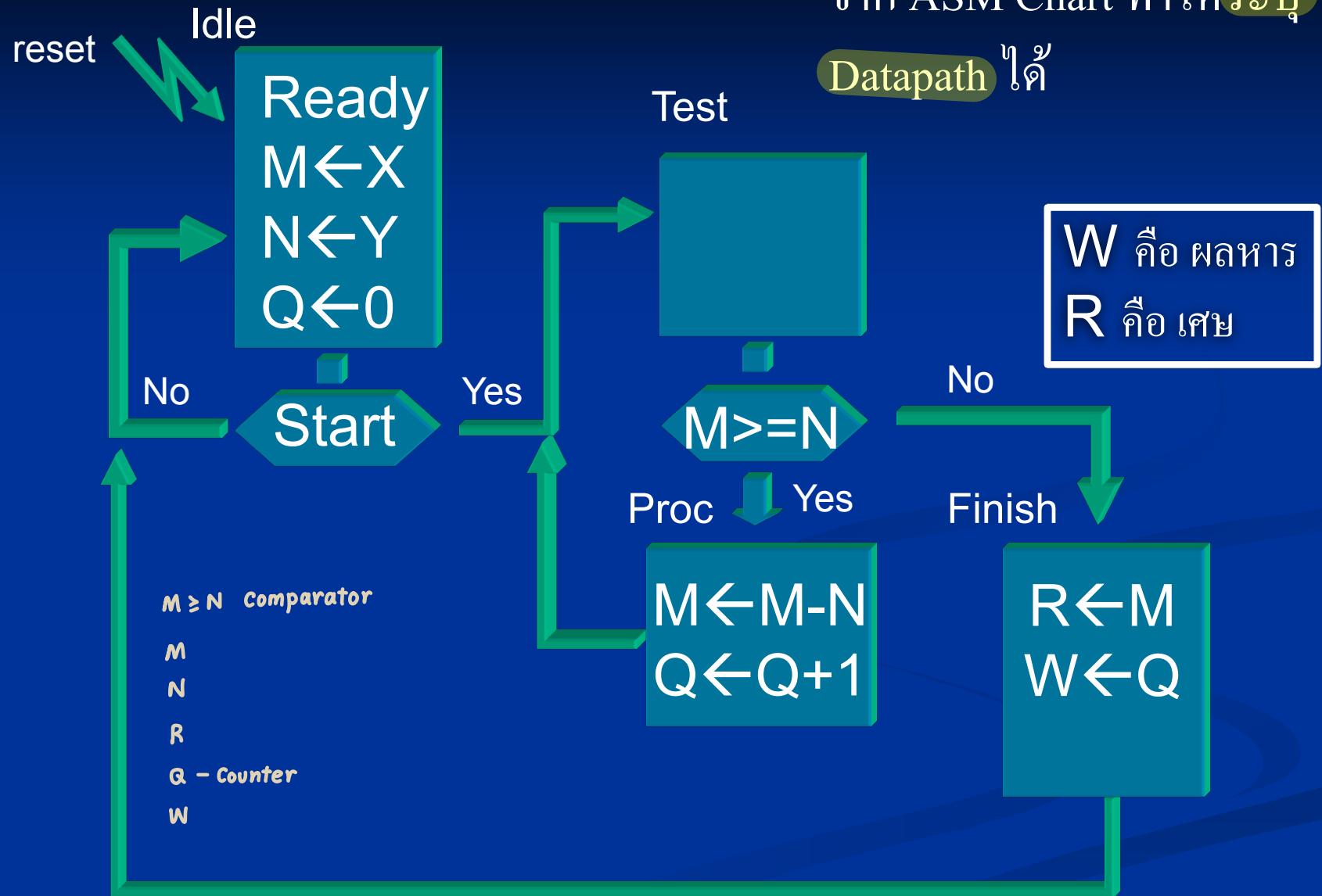






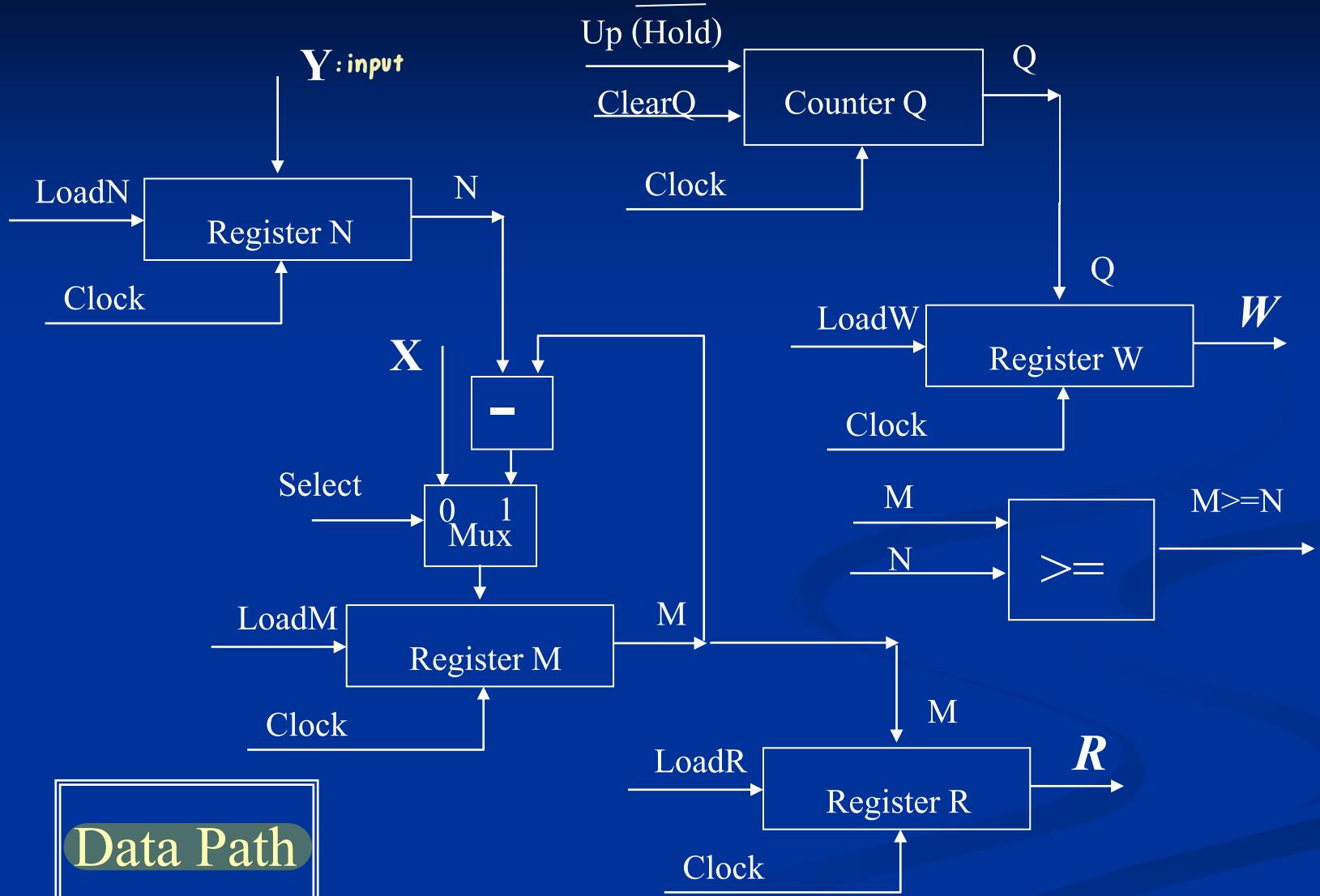
จาก ASM Chart ทำให้รับบู

Datapath ได้

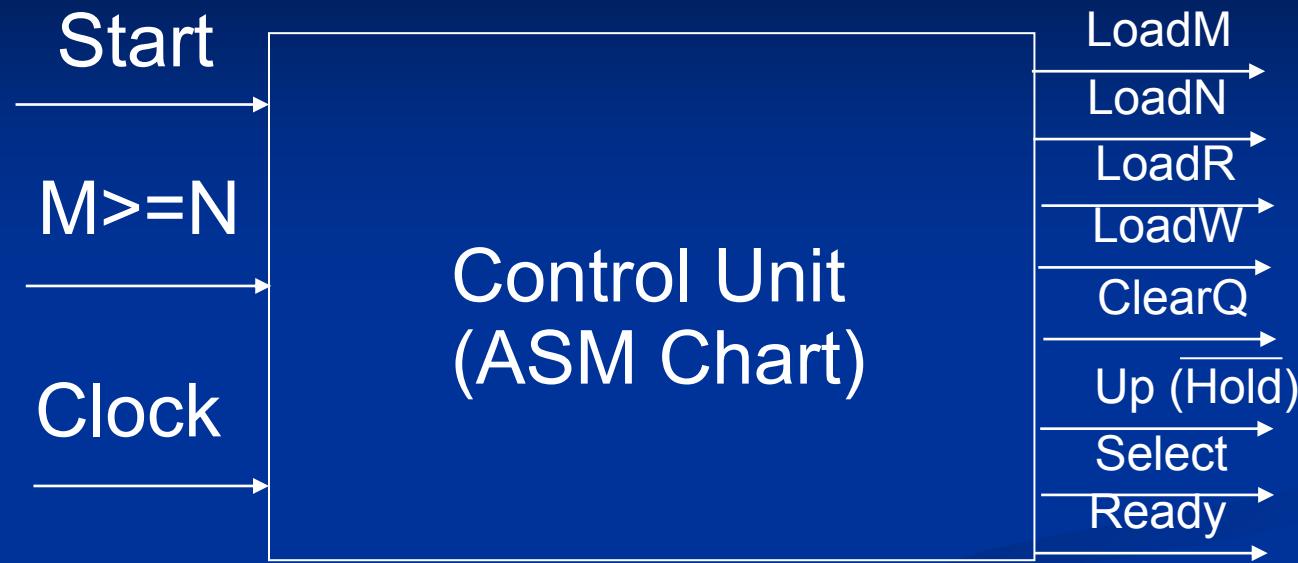


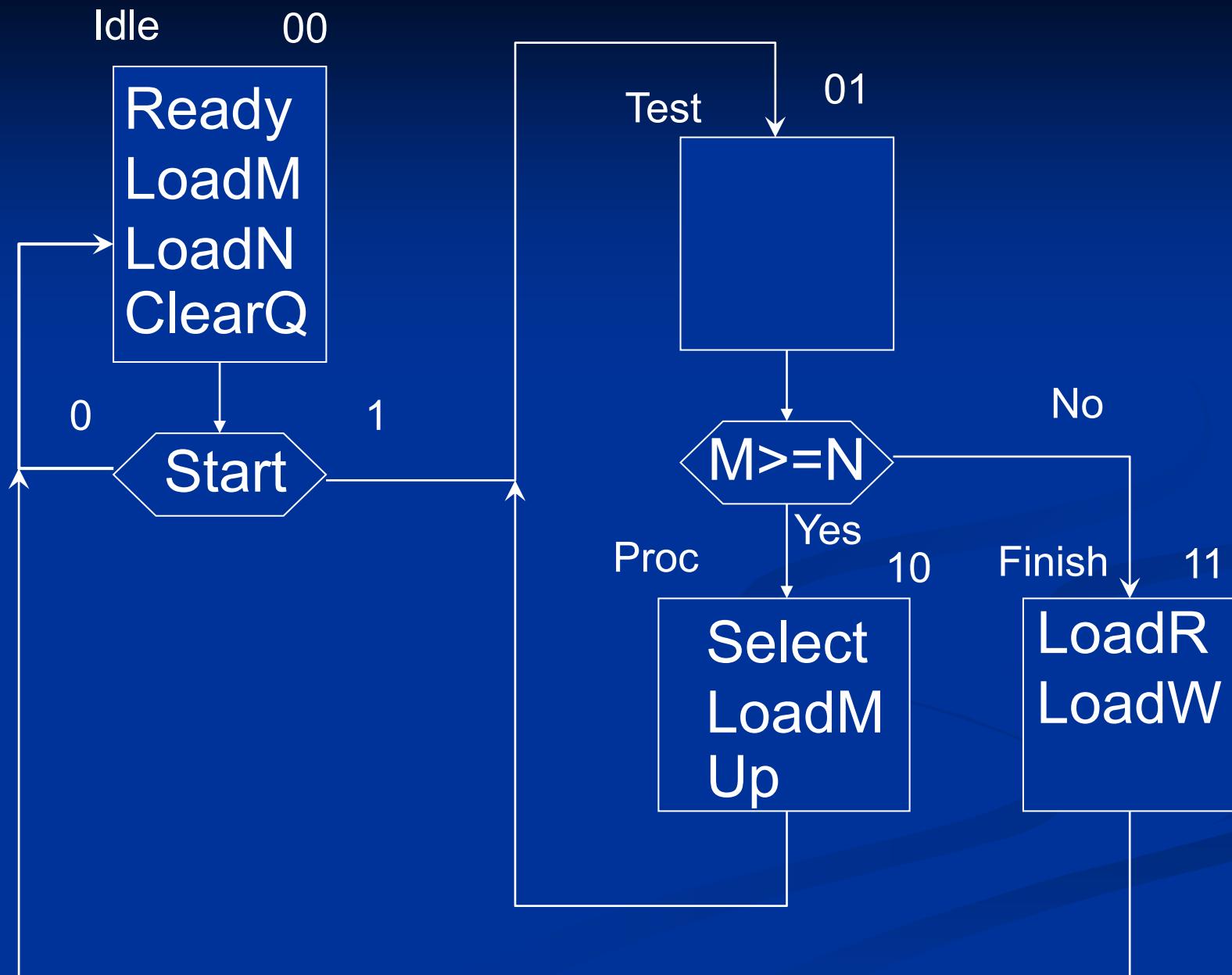
## *Datapath*

- Register M
- Register N
- Counter Q (Up+Hold+Clear Counter)
- Comparator ( $M \geq N$ ) և Combination Circuit
- Subtractor ( $M - N$ ) և Combination Circuit
- Register W
- Register R



Register Transfer System



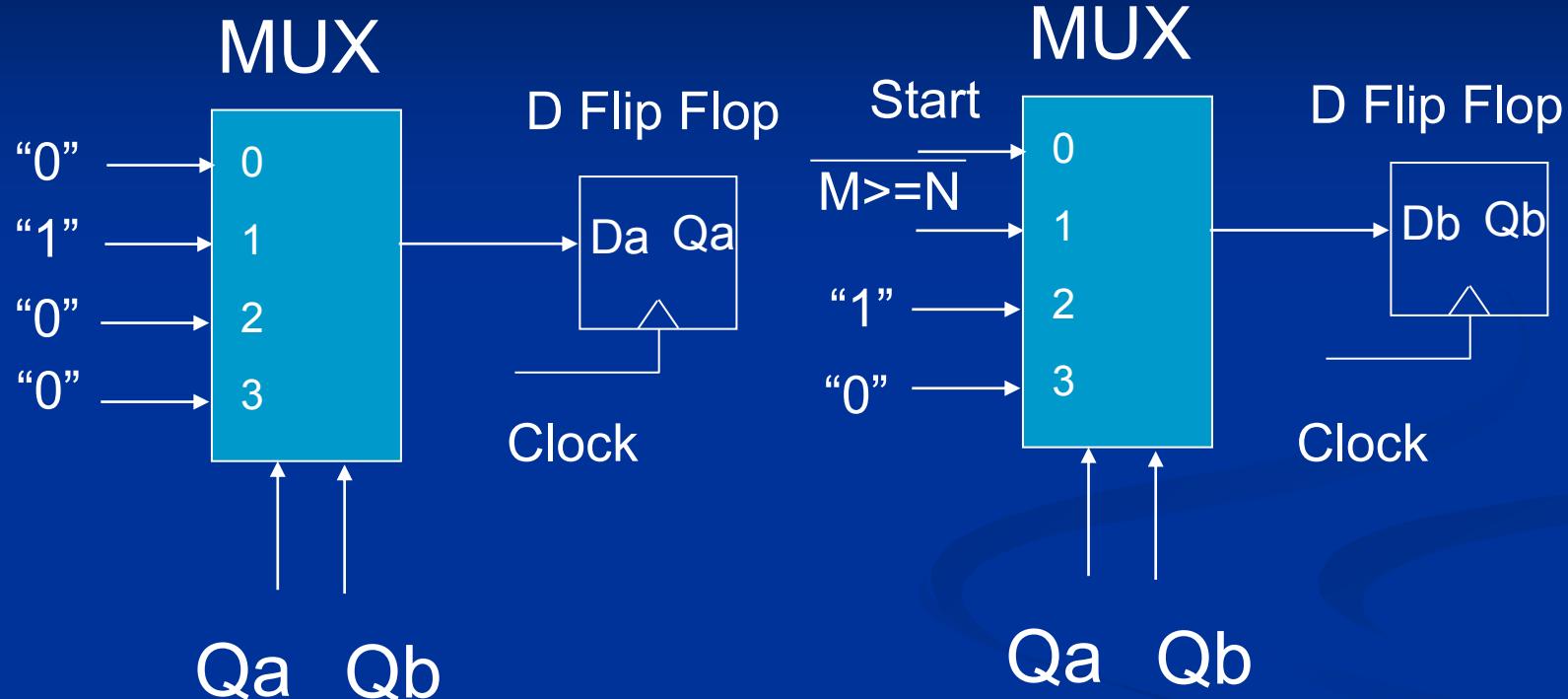




# Implement Control Unit

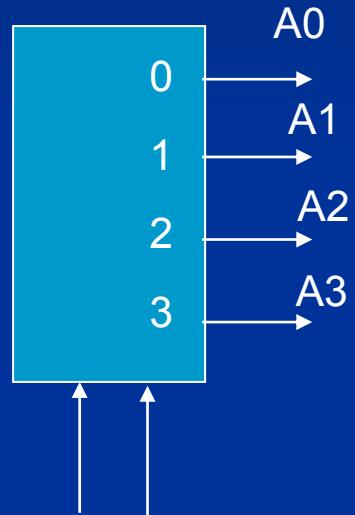
Present State	Next State	Condition
$Q_a Q_b$	$Q_a + Q_b +$	
0 0	0 0	Start=0
	0 1	Start=1
0 1	1 0	$(M \geq N) == 1$
	1 1	$(M \geq N) == 0$
1 0	0 1	none
1 1	0 0	none

# *Implement ໂຄຍິ່ງ Mux.*



# *Implement ໂຄຍລື່ມ Mux. (2)*

## Decoder



Ready=A0  
LoadM=A0+A2  
LoadN=ClearQ=A0  
Select=Up=A2  
LoadR=LoadW=A3



ตัวอย่างการออกแบบวงจร

วงจร สร้าง

Fibonacci Sequence

# បីលូហា

- វងចរមិ input គីអ N និង Start
- Output គីអ F និង Ready
- វងចររំនៅការងារមើនេះ Start active និងមើនេះការងារត្រួតត្រូវ Ready ទៅត្រួតត្រូវ 1 និង F ទៅត្រួតត្រូវលេខ Fibonacci លាត់លុយទី N
- លេខ Fibonacci មិត្តសញ្ញាណមិត្តសញ្ញាណនេះ
  - $F(K)=F(K-1)+F(K-2)$
  - $F(0)=0$
  - $F(1)=1$
- ការងារទី N មិត្តសញ្ញាណត្រួតត្រូវពី 1 ទៅត្រួតត្រូវ N ។

# *Algorithm หา Fibonacci ลำดับที่ N*

F=1

Y=0

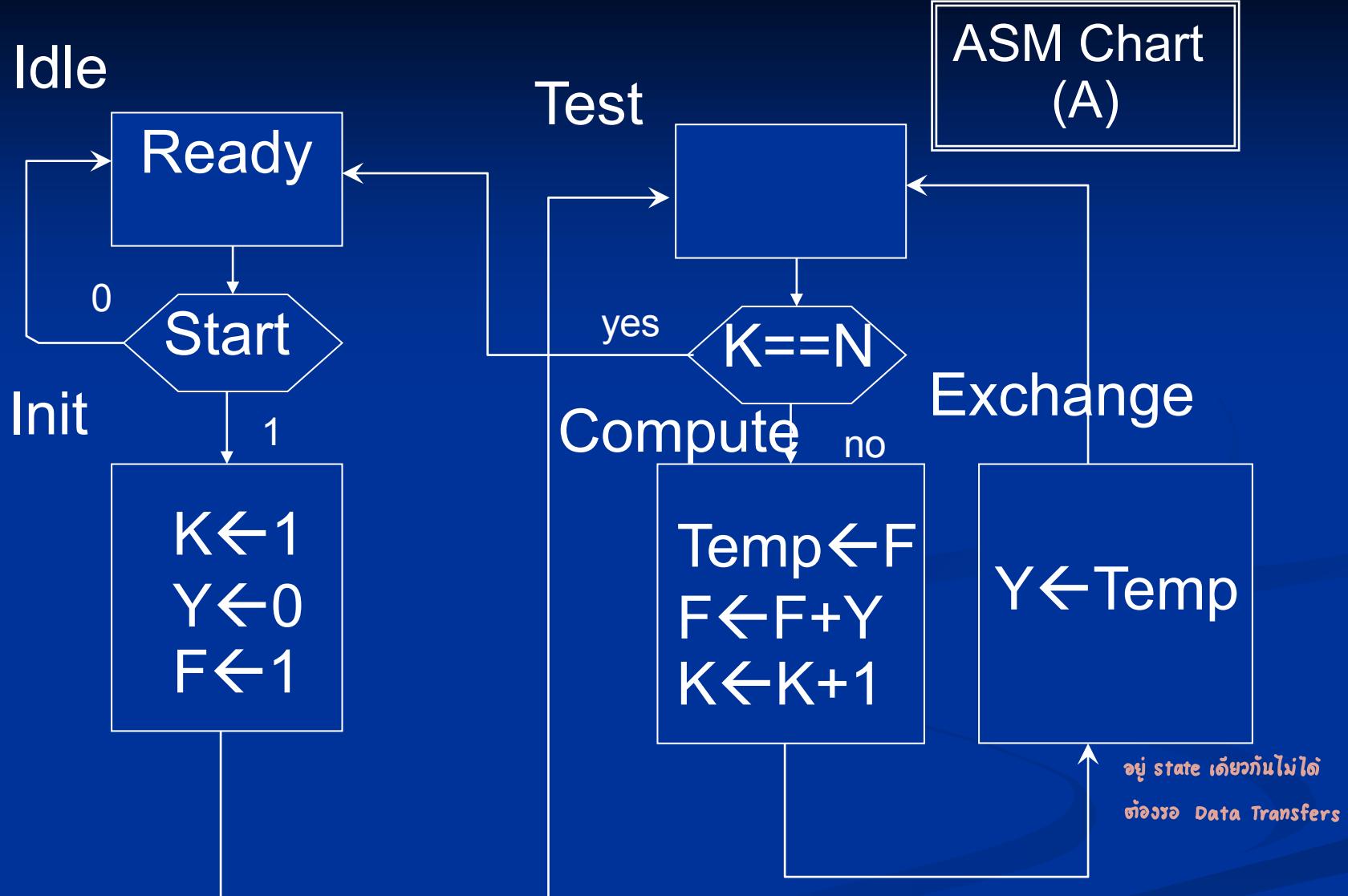
FOR K=1 TO N {

Temp=F

F=F+Y

Y=Temp

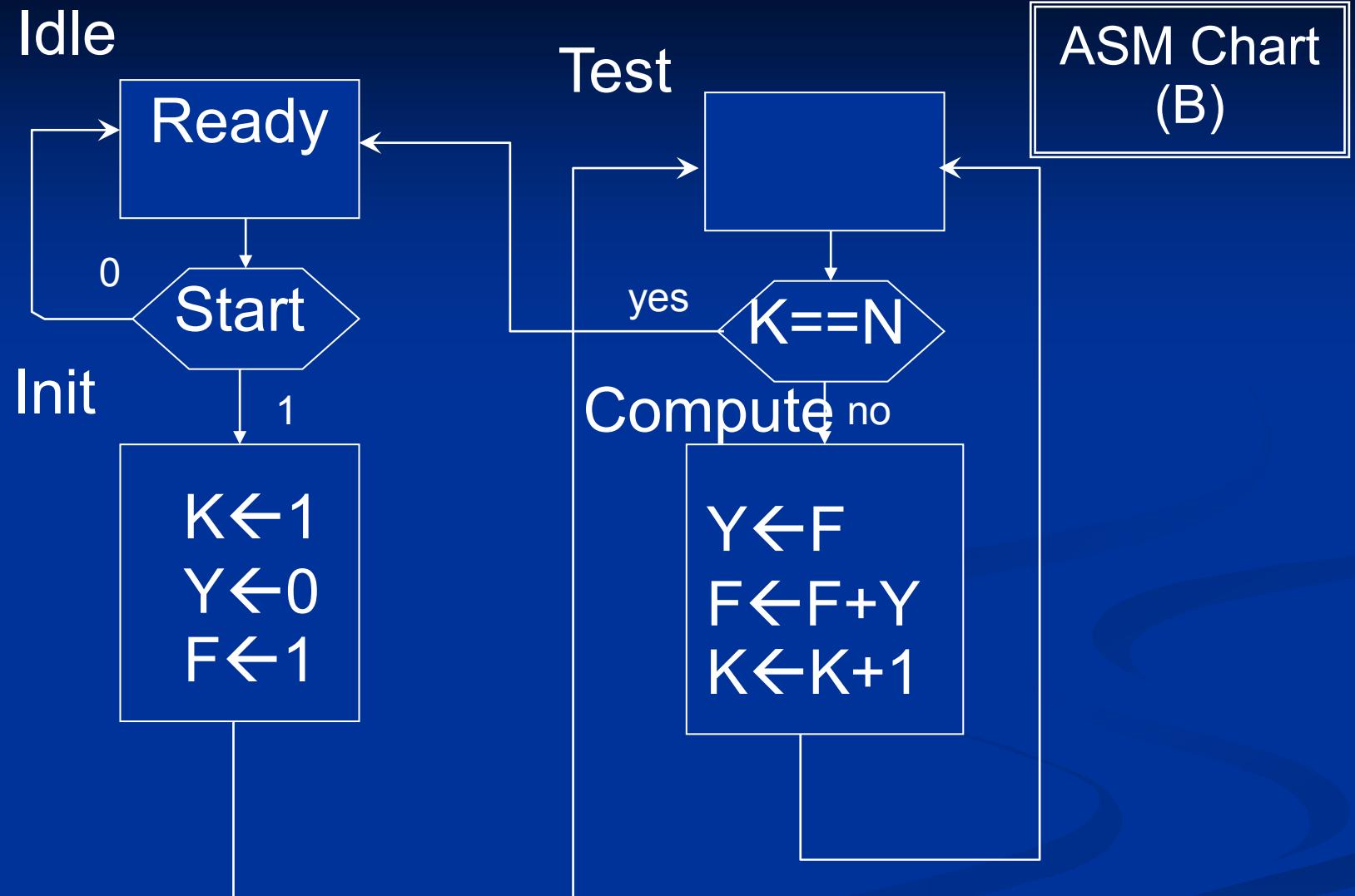
}



อยู่ state เดียวกันไม่ได้  
ต้องรอ Data Transfers

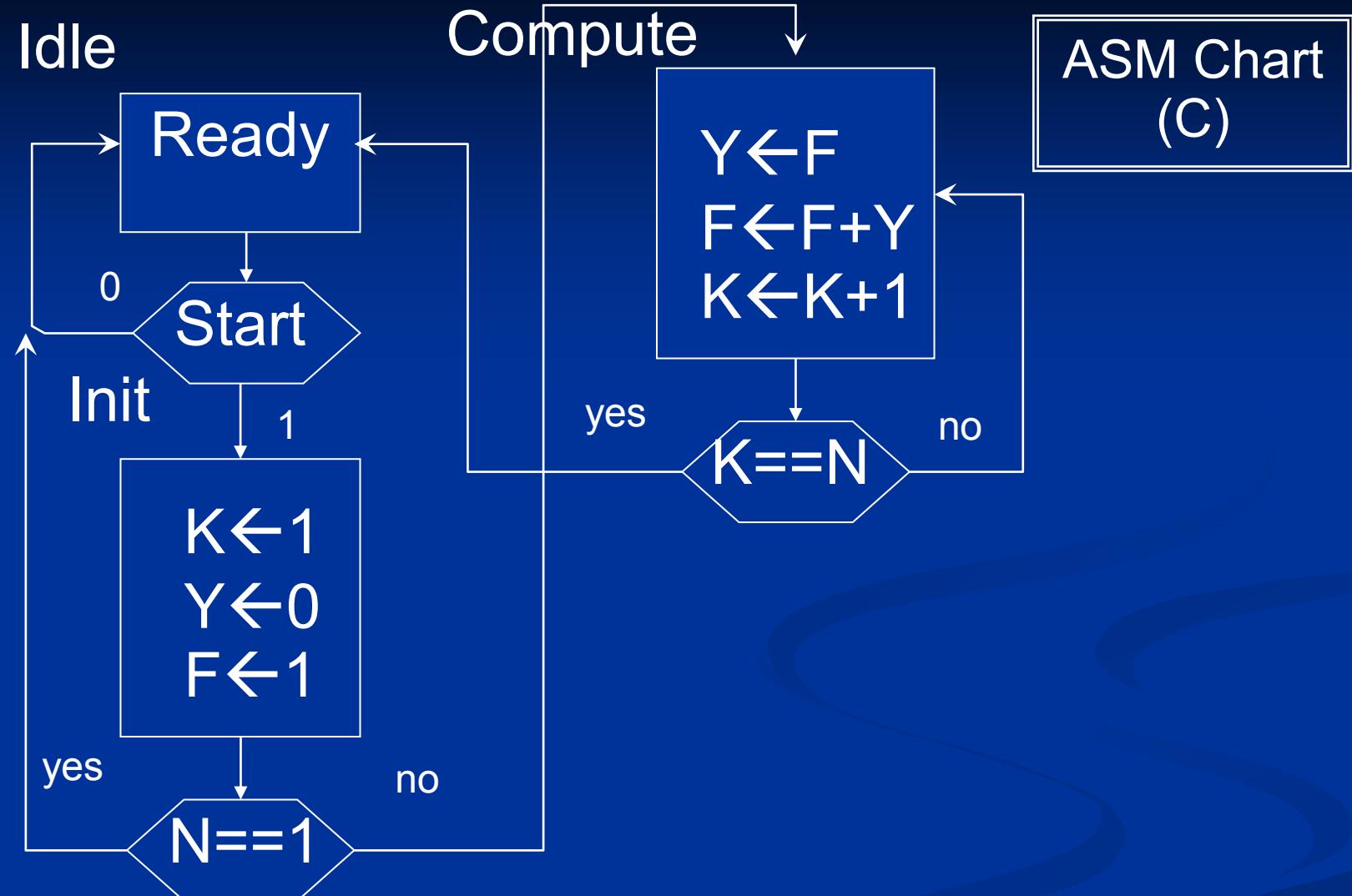
## วิเคราะห์ *ASM Chart (A)*

- มี 5 states และใน loop มี 3 states เวลาในการทำงาน  $> 3*N$  clock
- ใน state Compute; Temp จะได้ค่าเดิม(ก่อนบวก)ของ F เพราะการทำงานทั้งหมดทำใน clock เดียวกัน
- ใน state Exchange; Y จะได้ค่าเดิมของ F เพราะทำงานหลังจาก Temp รับค่าแล้ว 1 clock
- เมื่อพิจารณาดังนี้ Temp ไม่จำเป็น เพราะถ้าต้องการค่า “เดิม” ของ F สามารถทำใน Compute ได้ และตัด Exchange ทิ้ง ได้



## วิเคราะห์ *ASM Chart (B)*

- วงจรทำงานเร็วขึ้น เพราะใน loop ลดเหลือ 2 states
- ใน state Test ไม่มีการทำงาน น่าจะบูรณา  
กับ state Compute ได้ และใช่การทดสอบ  
 $N=1$  ไว้ที่ state Init



## วิเคราะห์ *ASM Chart (C)*

- วงจรทำงานเร็วขึ้นใน loop เหลือเพียง 1 state
- แต่เมื่อวิเคราะห์การทำงาน ผลที่ได้ไม่ถูกต้อง
- กรณี  $N=2 \Rightarrow F(2)=1$

Idle                     $F=? Y=? K=? N=2$  Start=1 NS=Init

Init                     $F=? Y=? K=? N=2$  Start=0 NS=Compute

Compute                 $F=1 Y=0 K=1 N=2$  Start=0 NS=Compute

Compute                 $F=1 Y=1 K=2 N=2$  Start=0 NS=Idle

Idle                     $F=2 Y=1 K=3 N=2$  Start=0 NS=Idle

# វិគរាងអ៊ីស៊ីម៉ាតុល្យ (ASM Chart) ទៅ

■ រាល់ N=3 => F(3)=2

Idle                    F=? Y=? K=? N=3 Start=1 NS=Init

Init                    F=? Y=? K=? N=3 Start=0 NS=Compute

Compute                F=1 Y=0 K=1 N=3 Start=0 NS=Compute

Compute                F=1 Y=1 K=2 N=3 Start=0 NS=Compute

Compute                F=2 Y=1 K=3 N=3 Start=0 NS=Idle

Idle                    F=3 Y=2 K=4 N=3 Start=0 NS=Idle

# វិគរាងអ៊ីស៊ីម៉ាតុល្យ (ASM Chart) ទៅ

■ ក្រណី  $N=4 \Rightarrow F(4)=3$

Idle                     $F=? Y=? K=? N=4$  Start=1 NS=Init

Init                     $F=? Y=? K=? N=4$  Start=0 NS=Compute

Compute                 $F=1 Y=0 K=1 N=4$  Start=0 NS=Compute

Compute                 $F=1 Y=1 K=2 N=4$  Start=0 NS=Compute

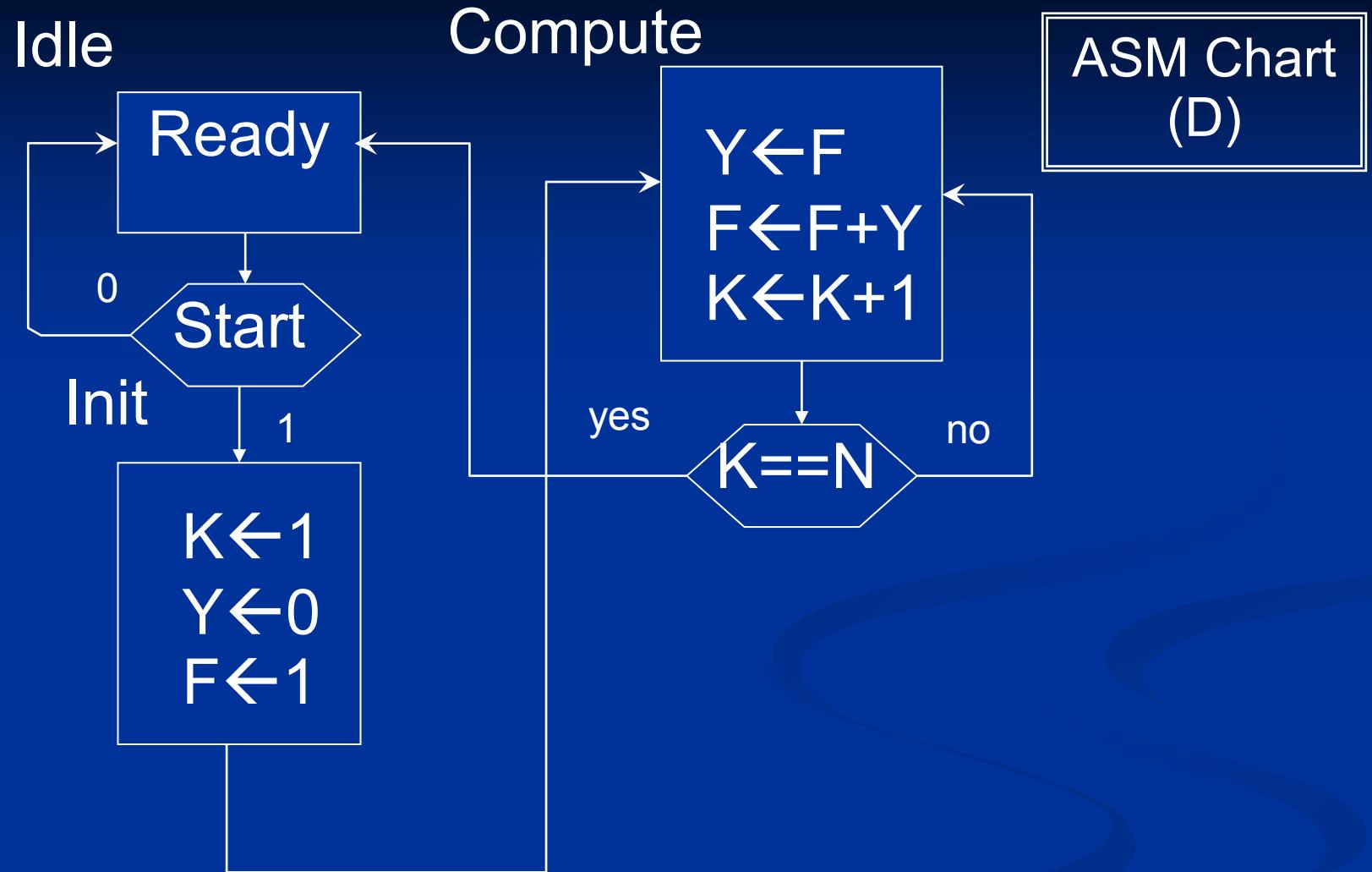
Compute                 $F=2 Y=1 K=3 N=4$  Start=0 NS=Compute

Compute                 $F=3 Y=2 K=4 N=4$  Start=0 NS=Idle

Idle                     $F=5 Y=3 K=5 N=4$  Start=0 NS=Idle

## วิเคราะห์ *ASM Chart (C)* ต่อ

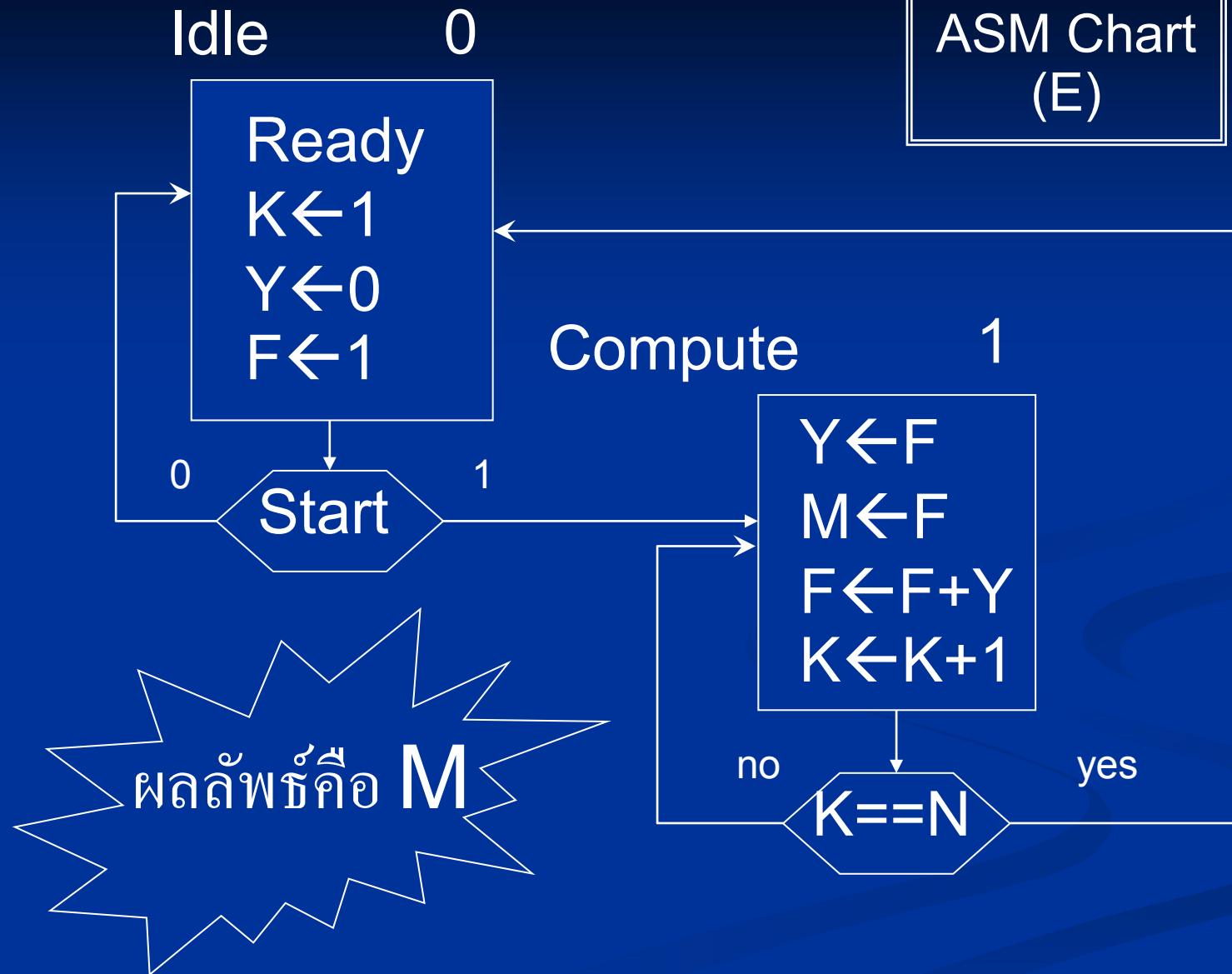
- แทนที่จะทำ loop N ครั้ง กลับทำ  $N+1$  ครั้ง เพราะ K increment ใน clock ถัดไป
- Y เป็นค่า  $F(K-1)$  ดังนั้นเมื่อทำงานใน loop  $N+1$  ครั้ง Y จะเป็น  $F(N)$  หรือคำตอบที่ต้องการ
- แต่ในกรณี  $N=1$  คำตอบไม่ใช่ Y เพราะการทดสอบใน state Init ถ้าทำการทดสอบใน state Init ออกค่าใน Y จะเป็นคำตอบที่ถูกต้อง



## วิเคราะห์ *ASM Chart (D)*

- State Init ใช้เพื่อกำหนดค่าเริ่มต้น ซึ่งค่าเหล่านี้จะเป็นช่วงเดิมทุกครั้งที่วงจรเริ่มทำงาน
- ปุบ state Init รวมกับ state Idle แต่ค่า Y จะเป็น 0 ทันทีใน state Idle
- ทางแก้ใช้ Register อีกอันหนึ่งเก็บค่า F (Y ยังต้องใช้เพื่อการคำนวณ)

ASM Chart  
(E)



$$F(0) = 0$$

$$F(1) = 1$$

$$F(2) = 1$$

$$F(3) = 2$$

$$F(4) = 3$$

## វិគ្រាជអ៊ីស៊ម Chart (E)

■ រាល់ N=4 => F(4)=3

Idle                    F=? Y=? K=? N=4 M=? Start=0 NS=Idle

Idle                    F=1 Y=0 K=1 N=4 M=? Start=0 NS=Idle

Idle                    F=1 Y=0 K=1 N=4 M=? Start=1 NS=Compute

Compute                F=1 Y=0 K=1 N=4 M=? Start=0 NS=Compute

Compute                F=1 Y=1 K=2 N=4 M=1 Start=0 NS=Compute

Compute                F=2 Y=1 K=3 N=4 M=1 Start=0 NS=Compute

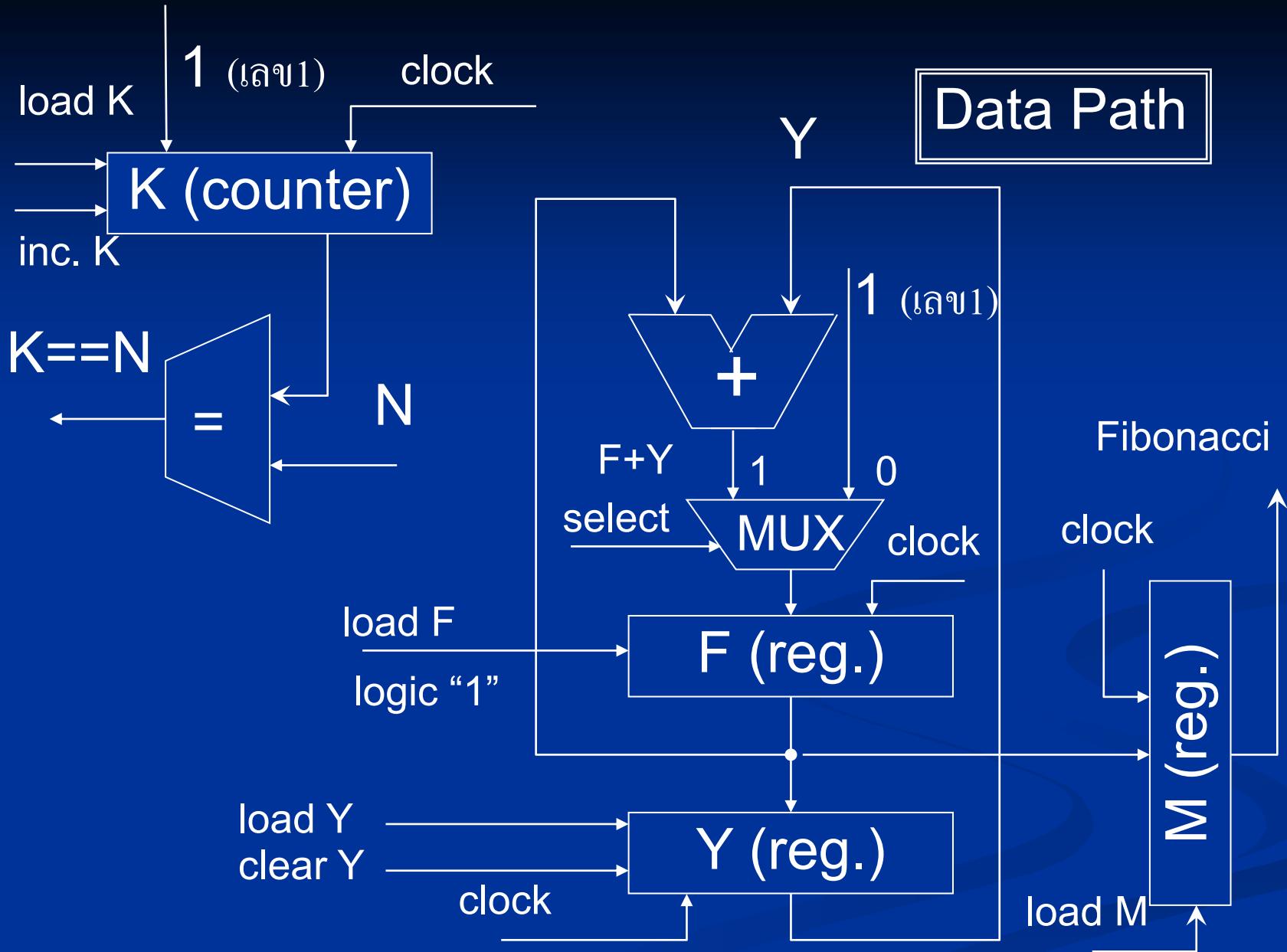
Compute                F=3 Y=2 K=4 N=4 M=2 Start=0 NS=Idle

Idle                    F=5 Y=3 K=5 N=4 M=3 Start=0 NS=Idle

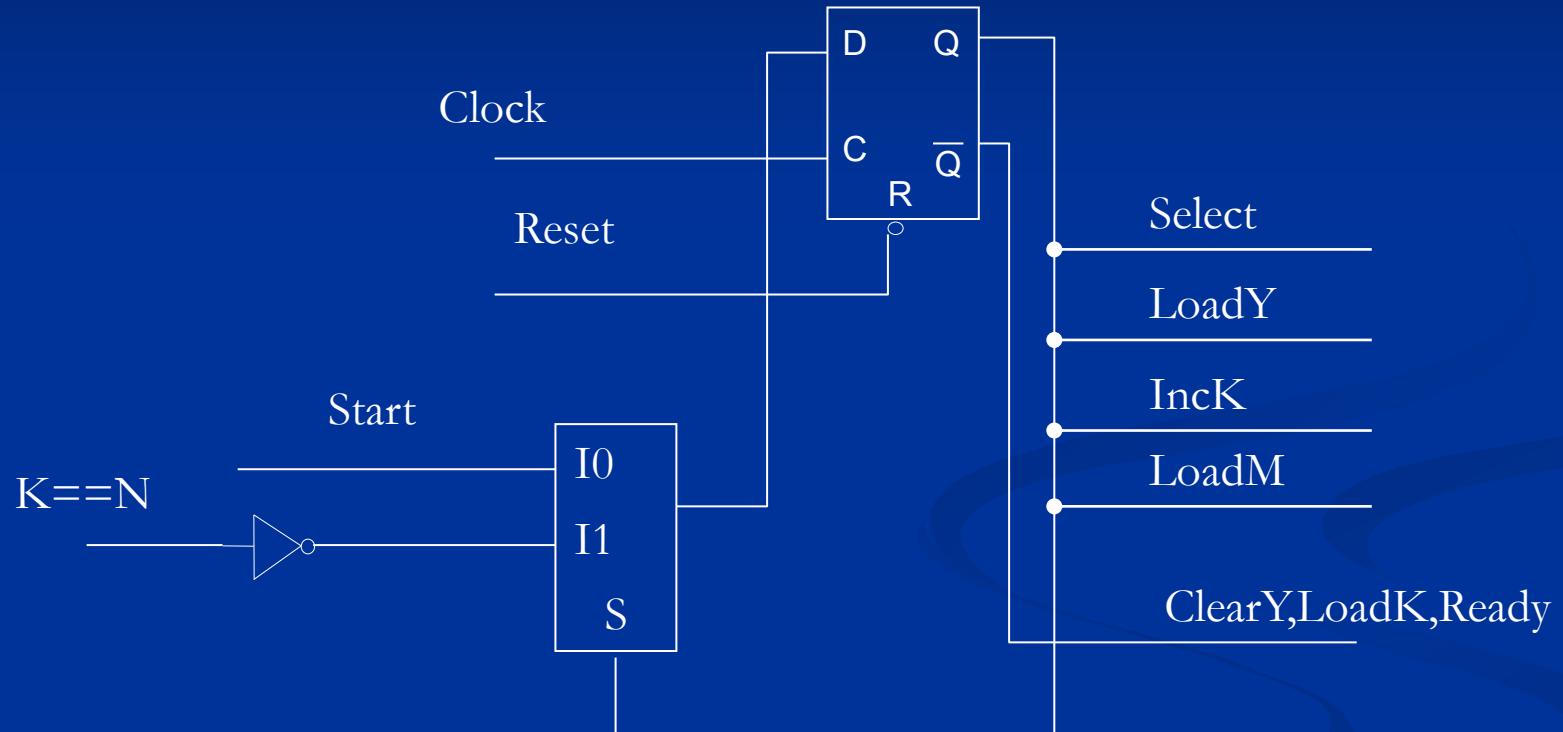
Idle                    F=1 Y=0 K=1 N=? M=3 Start=0 NS=Idle

## *Data Path*

- หน่วยความจำที่ต้องใช้ F, Y, M และ K
- K ต้องสามารถ load ค่า 1 ได้และ increment ได้เลือกใช้ counter
- F,M และ Y เป็น register
- F อาจถูก load ด้วย 1 หรือ  $F+Y$  ใช้ MUX เลือกค่าที่จะ load (F ถูก load ในทุก clock)
- Y clear หรือ load ด้วย F; M ถูก load ด้วย F
- ต้องมีวงจร comparator และ adder



# *Control Unit*

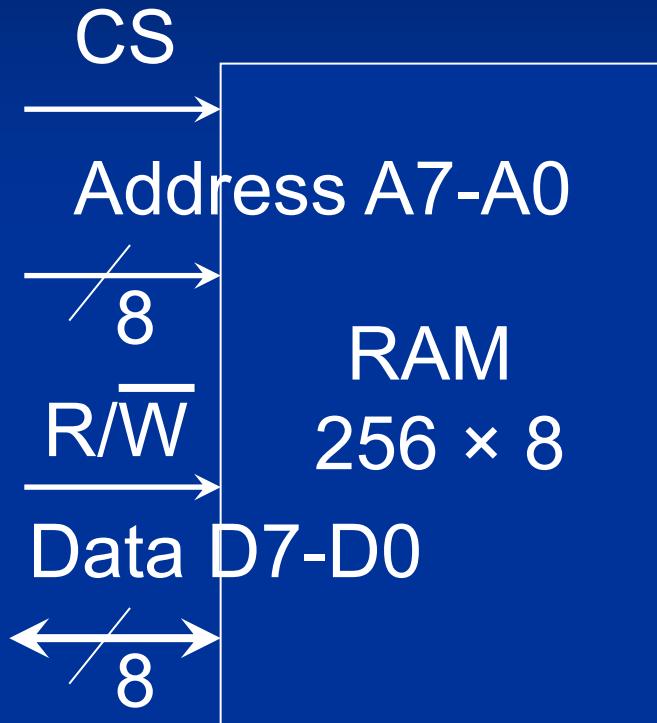


# ปัญหา : sslับค่าใน RAM

# ប្រព័ន្ធអាជីវកម្ម

- ត្រូវការសម្រេចលទ្ធផលរបស់ RAM ទម្ងន់ 256 × 8 ធ្វើឡើងជា Input
  - X ទម្ងន់ 8 បិត
  - Y ទម្ងន់ 8 បិត និង
  - Start ទម្ងន់ 1 បិត
- เมื่ីយោង Start ត្រូវបានក្លាយជាប្រព័ន្ធអាជីវកម្ម RAM នៃការដកចាត់នៅ address X ក្នុងការដកចាត់នៅ address Y
- ការសម្រេចនៃ output Ready នឹងត្រូវត្រួតពិនិត្យថា វាអាចបានក្លាយជាប្រព័ន្ធអាជីវកម្ម RAM បាន
- ការដកចាត់នៃ RAM នឹងរួមចំណាំថាបានក្លាយជាប្រព័ន្ធអាជីវកម្ម RAM
- RAM មិនមែនបានគ្រប់គ្រងចំណាំទៅបាន

## Asynchronous RAM



## RAM

- CS (Chip Select) ถ้า active RAM ทำงาน ถ้าไม่ active RAM ไม่ทำงาน Data เป็น High Impedance
- Address กำหนดตำแหน่งใน RAM
- Data เป็นทางเข้าออก (bidirectional) ของข้อมูล
- R/W เป็น 1 แสดงว่าอ่านข้อมูล จาก RAM ถ้าเป็น 0 แสดงว่า เก็บข้อมูลลงเก็บใน RAM

## *Algorithm*

- โดยทั่วไปการสลับค่า เช่นใน array Q ถ้าต้องการสลับค่าระหว่าง  $Q(K)$  กับ  $Q(L)$  algorithm จะเป็น

$\text{temp} = Q(K)$

$Q(K) = Q(L)$

$Q(L) = \text{temp}$

แต่...

## *Algorithm (ต่อ)*

- เนื่องจาก RAM ไม่สามารถกำหนด address 2 ค่าได้พร้อมกัน ดังนั้น  $Q(K)=Q(L)$  ทำไม่ได้ ต้องแก้ algorithm เป็น

$A=RAM[X]$

$B=RAM[Y]$

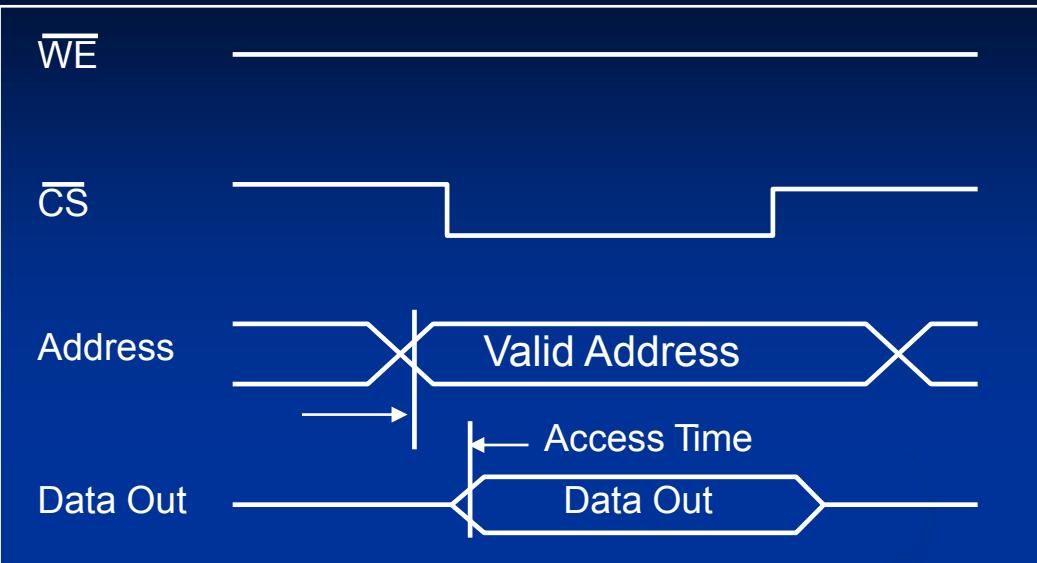
$RAM[Y]=A$

$RAM[X]=B$

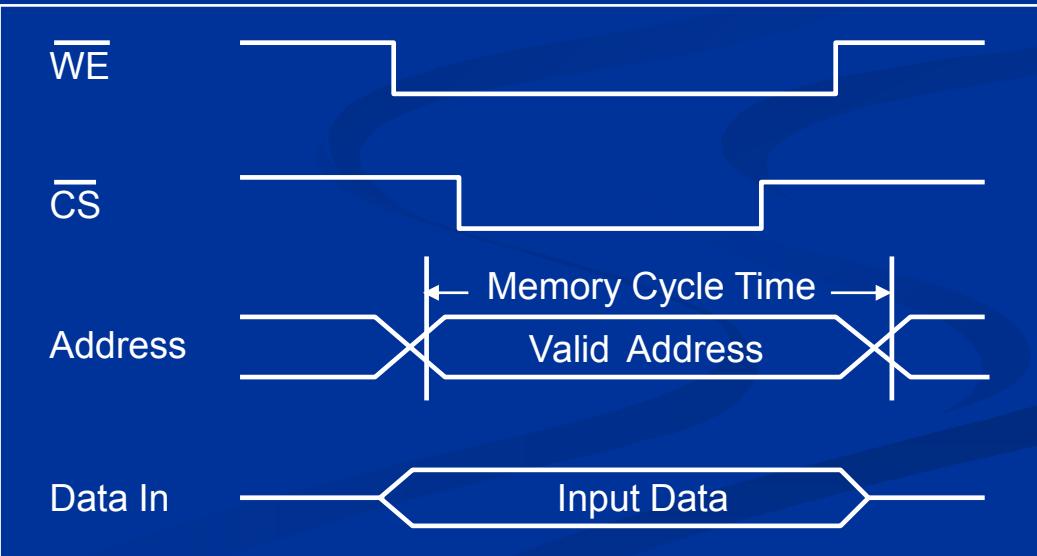
# *RAM Timing*

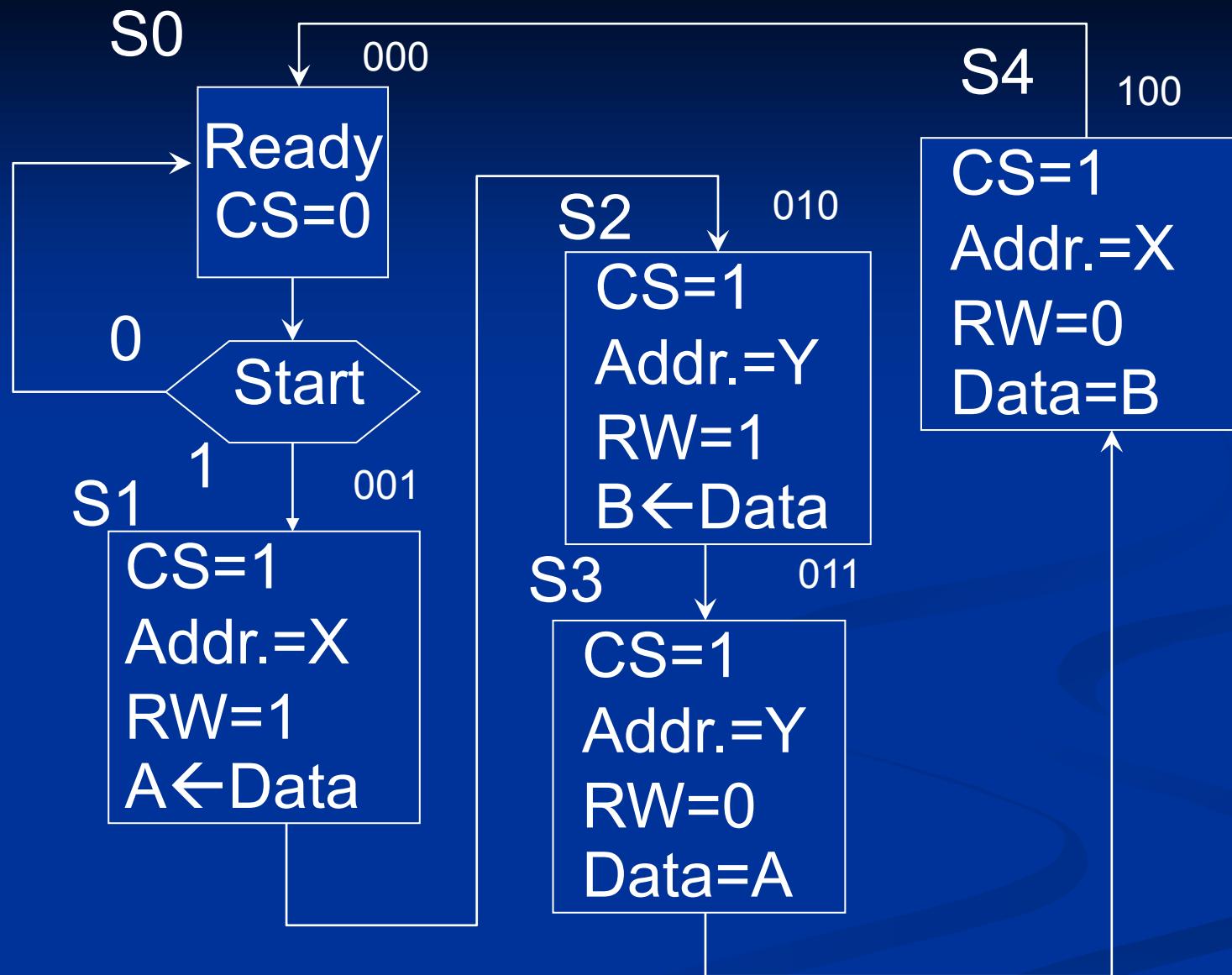
## **Simplified Read Timing**

# *Random Access Memories*



## **Simplified Write Timing**





# วิเคราะห์การทำงาน

## ■ RAM ทำงานแบบ Asynchronous ไม่มี clock

ทดสอบ timing ของ state S1

- clock มี period 100 time unit
- RAM access time 30 time unit
- Propagation delay ของ control unit 10 time unit
- Register delay 5 time unit หลังจาก clock edge

# *Timing*

Time

clk ↑ 100 cs=? rw=? address=? data=? ld A=? A=?



control unit (10) 110 cs=1 rw=1 address=X data=? ld A=1 A=?

RAM (30) 140 cs=1 rw=1 address=X data=RAM[X] ld A=1 A=?

clk ↑ 200 cs=1 rw=1 address=X data=RAM[X] ld A=1 A=?



register (s) delay 205 cs=1 rw=1 address=X data=RAM[X] ld A=1 A=RAM[X]

210 cs=1 rw=1 address=Y data=RAM[X] ld A=0 A=RAM[X]

240 cs=1 rw=1 address=Y data=RAM[Y] ld A=0 A=RAM[X]

hold time ของ register ต้องน้อยกว่า propagation delay ของ control unit

S1

CS=1  
Addr.=X  
RW=1  
 $A \leftarrow Data$

การเขียนในลักษณะนี้เรียกว่า **RTL**

(Register Transfer Level)

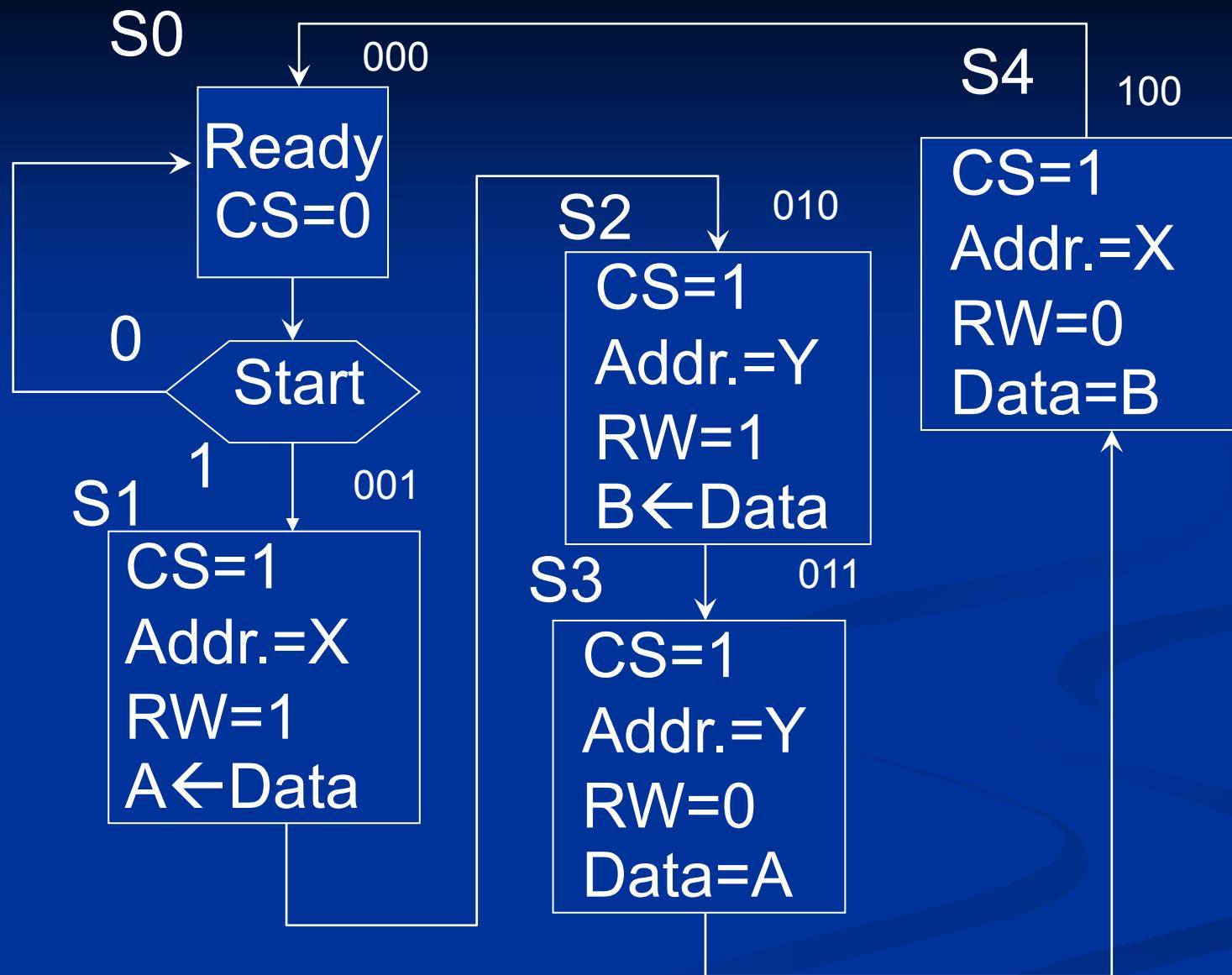
คือระบุการทำงาน ไม่ได้ระบุสัญญาณ

การทำงานใน state S1 มีสองแบบ คือ  
แบบที่ 1 CS=1, Addr.=X ,RW=1

แบบที่ 2  $A \leftarrow Data$

แบบที่ 1 เป็น **output** ที่ให้ค่า “ทันที” ส่งค่าออกไป

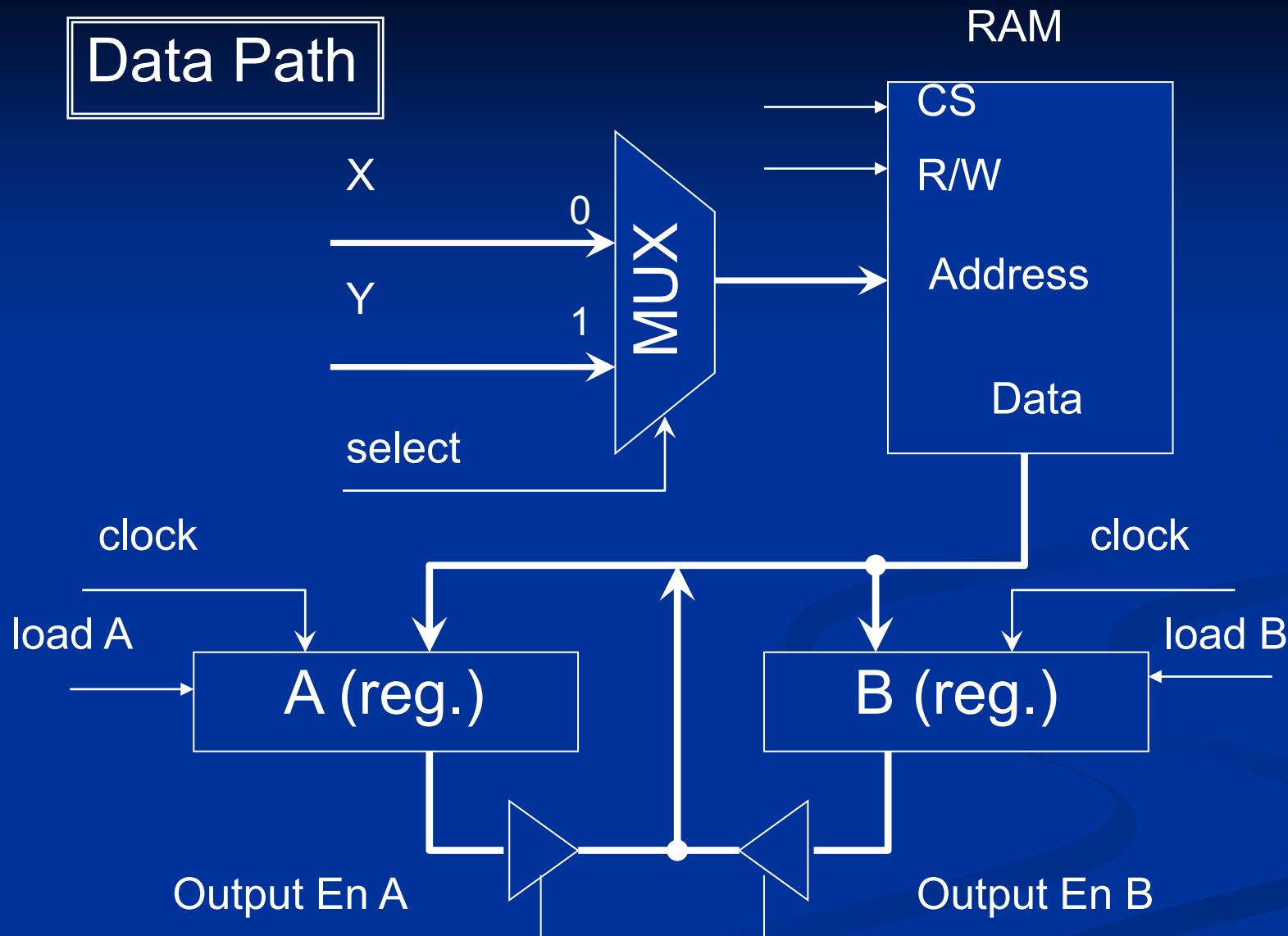
แบบที่ 2 เป็นการทำงานที่ต้องการ ซึ่งจะเกิดใน **clock** ถัดไป และอาจต้องใช้  
สัญญาณควบคุมหลายอัน ในบางครั้งเพื่อความชัดเจนจะใช้  $A \leftarrow Data$



## *Data Path*

- A และ B เป็นหน่วยความจำ ใช้ register
- “ค่า” ของ A และ B ต้องต่อกับ data bus ของ RAM ดังนี้นต้องต่อกับ tri-state buffer
- ถือว่า input X และ Y ไม่เปลี่ยนแปลงตลอดการทำงาน





**control unit**

