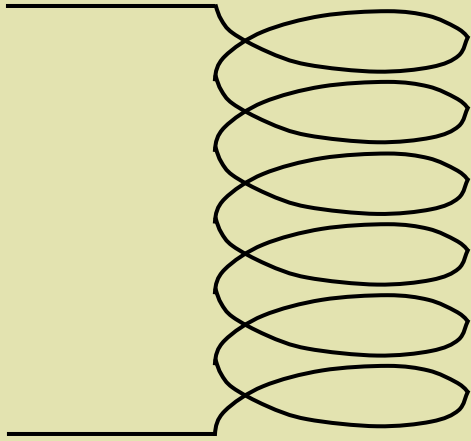
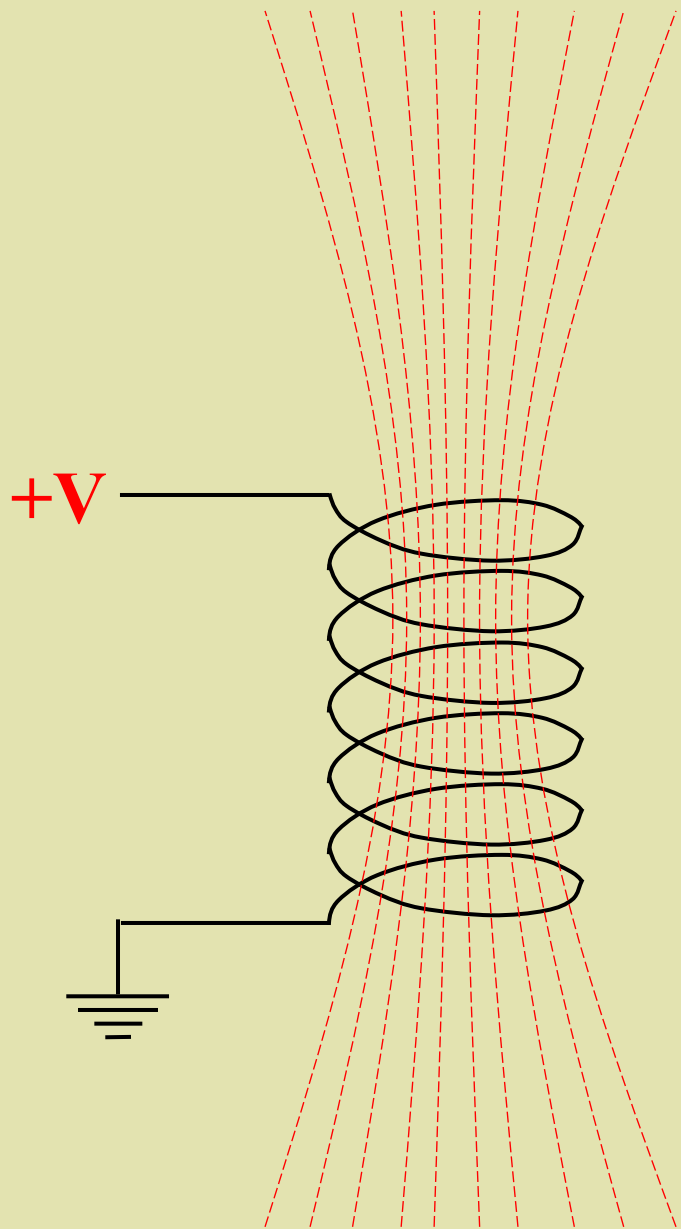


# The Design of a Relay Computer

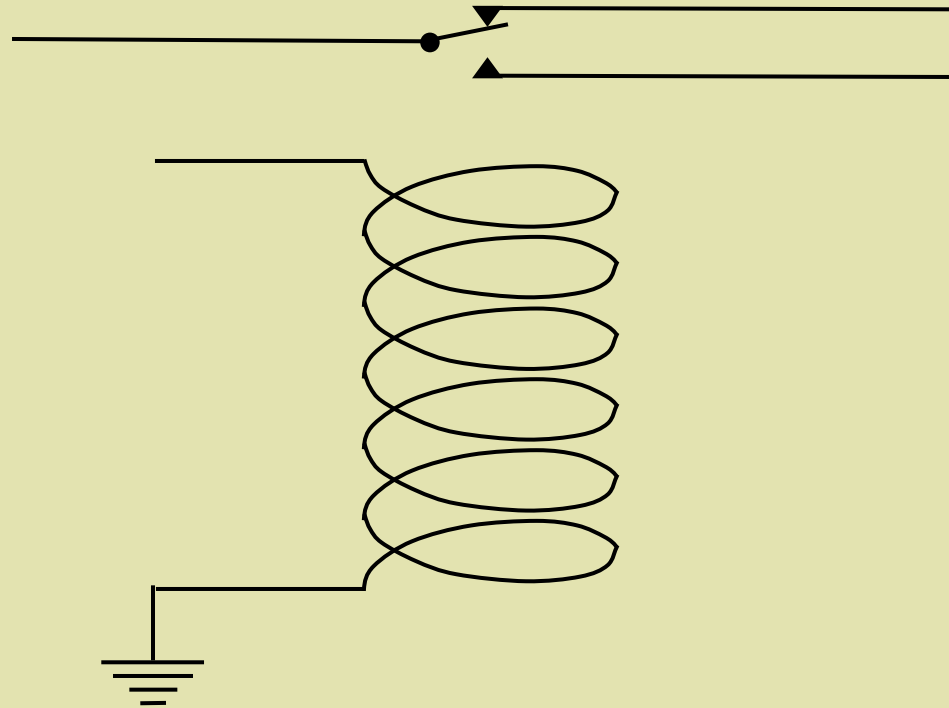
*Harry Porter, Ph.D.*  
Portland State University

April 24, 2006

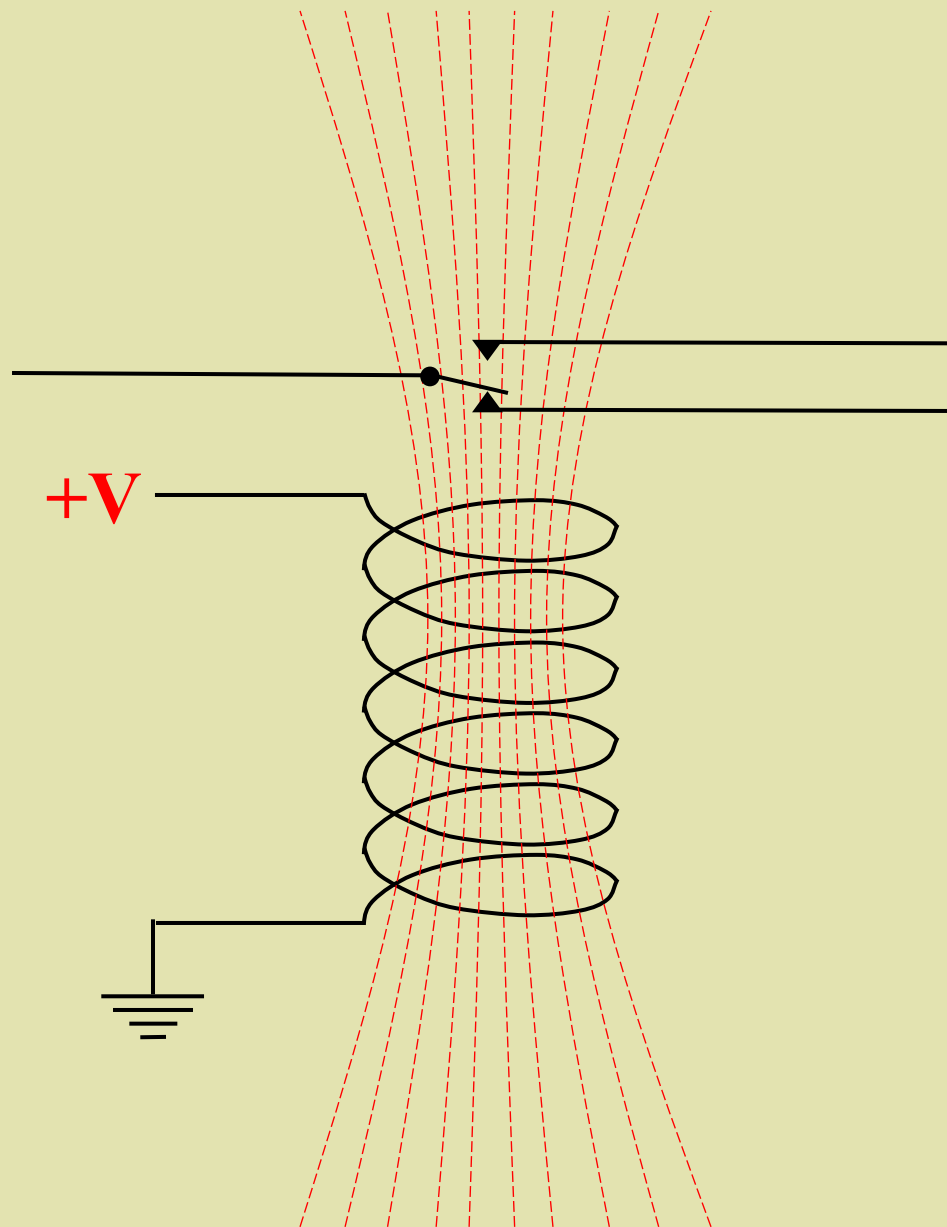




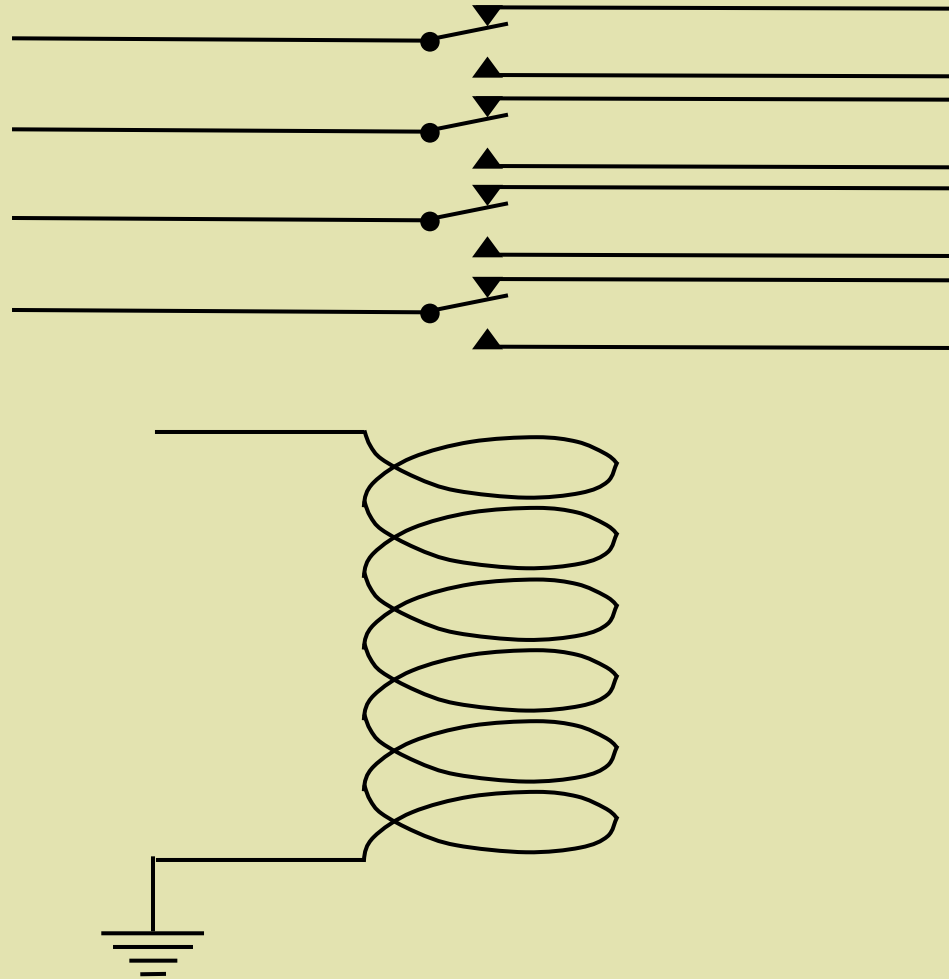
# Double Throw Relay



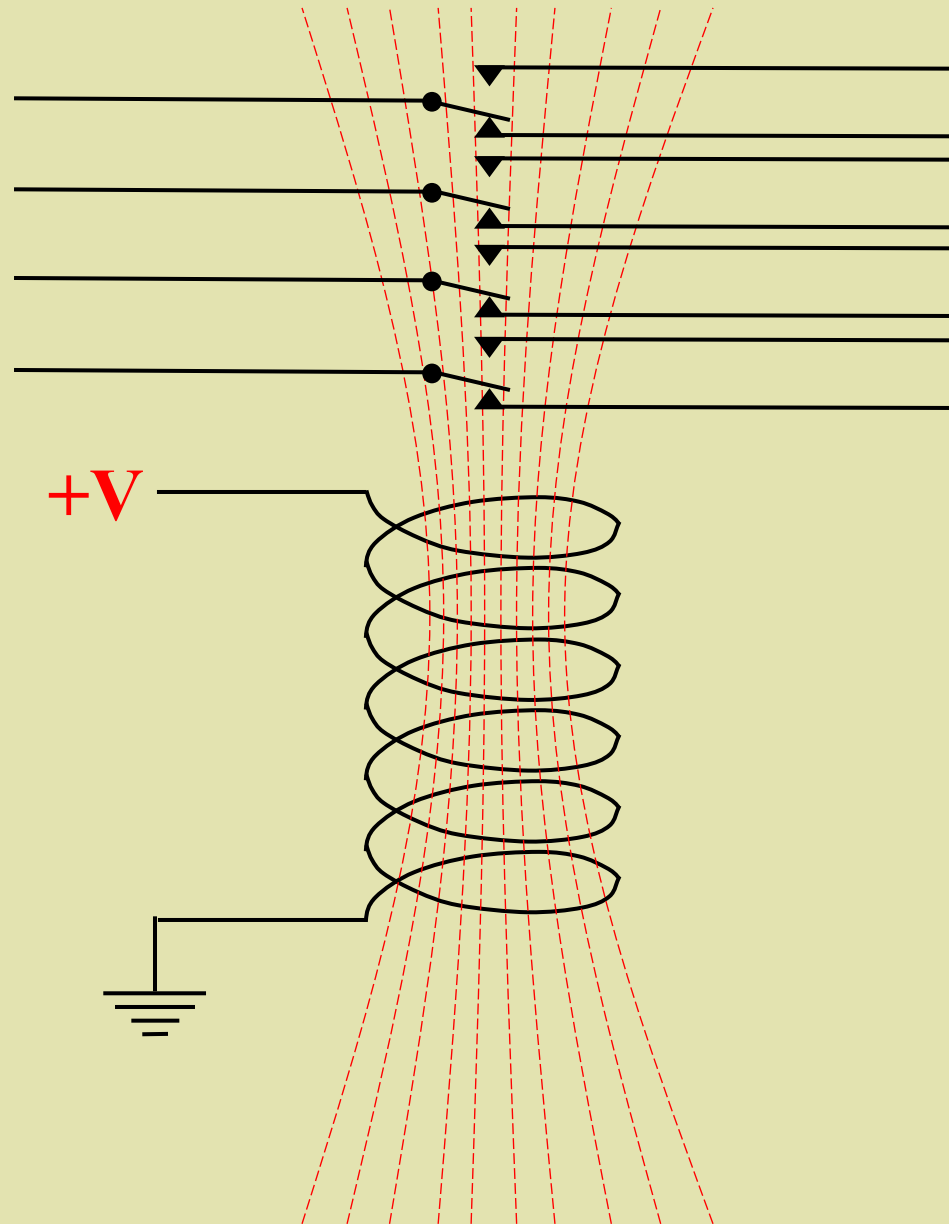
# Double Throw Relay

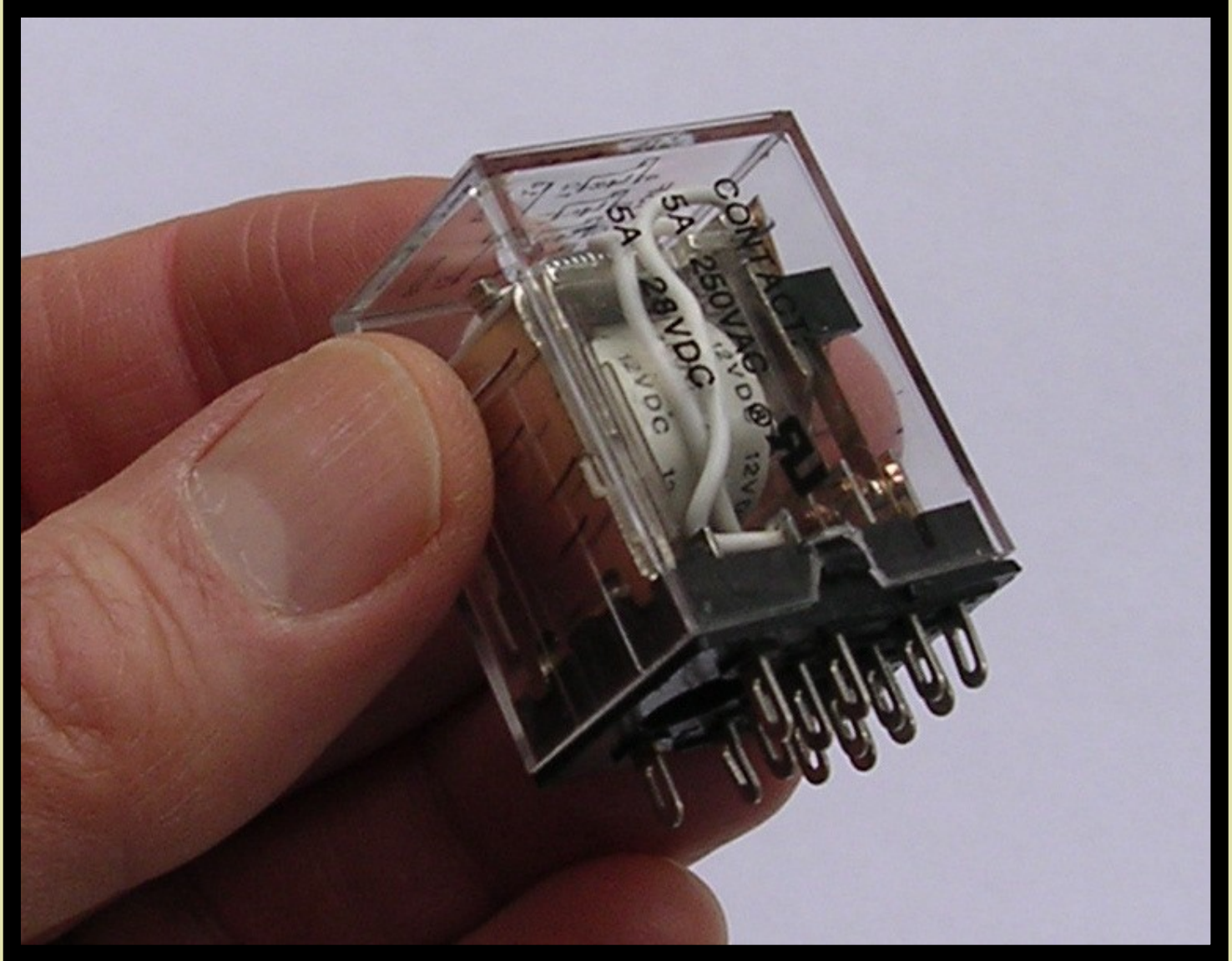


# Four Pole, Double Throw Relay



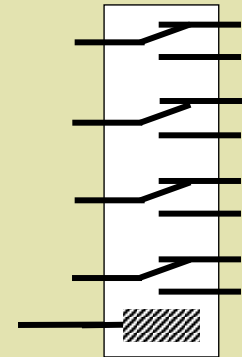
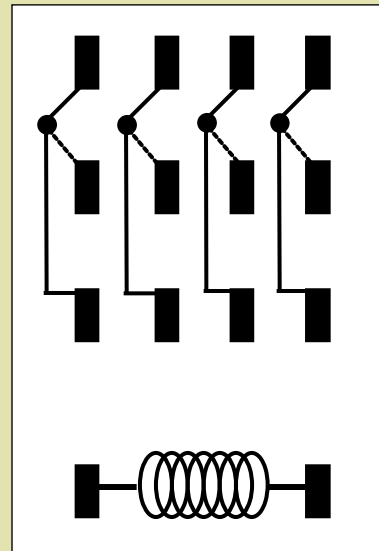
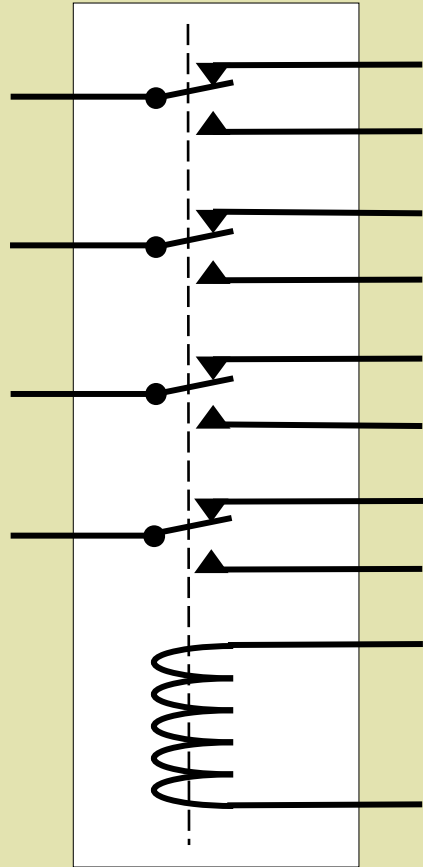
# Four Pole, Double Throw Relay



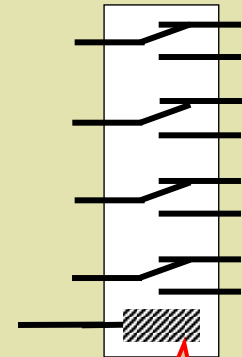
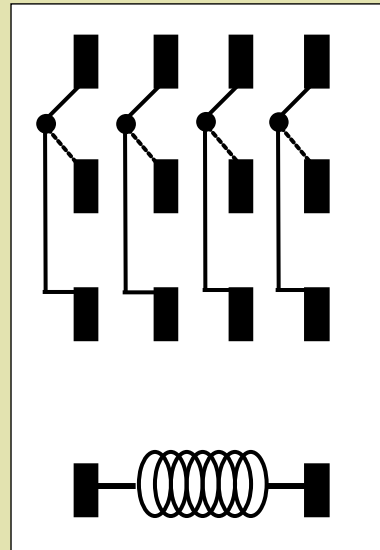
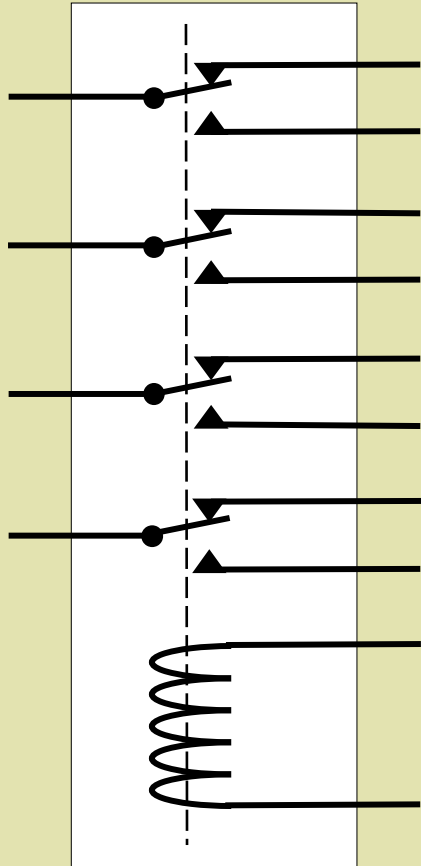




# Schematic Diagrams

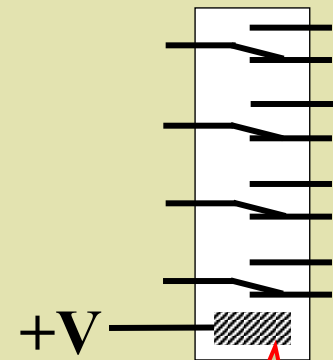
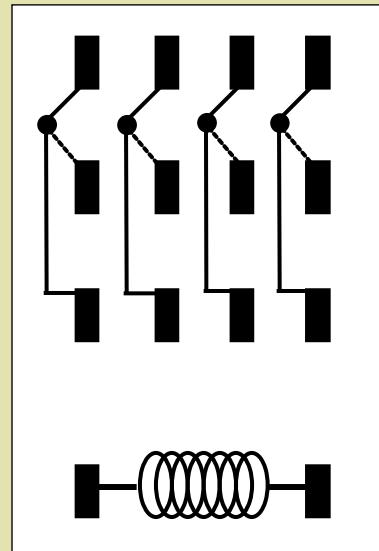
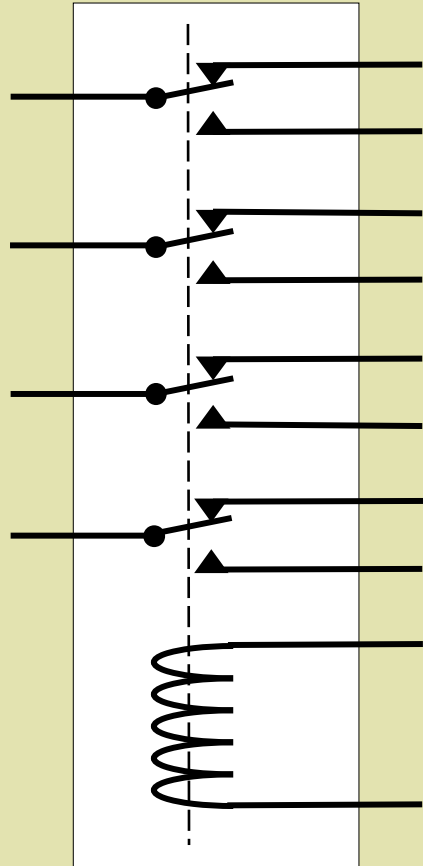


# Schematic Diagrams



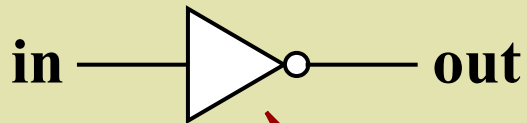
*Assume other terminal  
is connected to ground*

# Schematic Diagrams

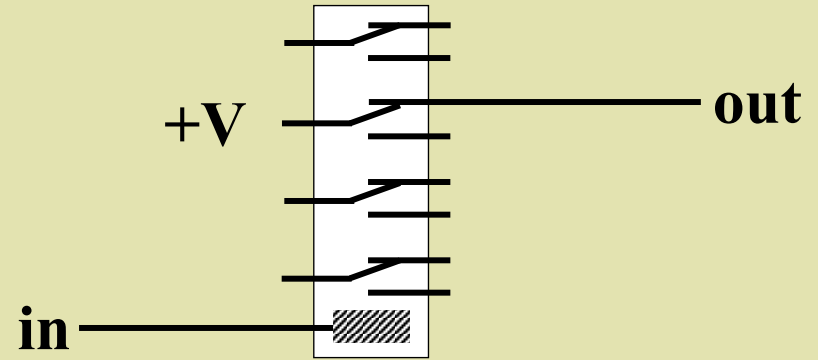


*Assume other terminal  
is connected to ground*

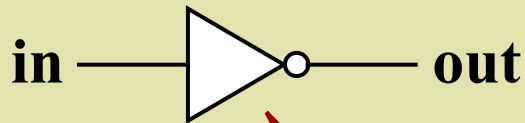
# The “NOT” Circuit



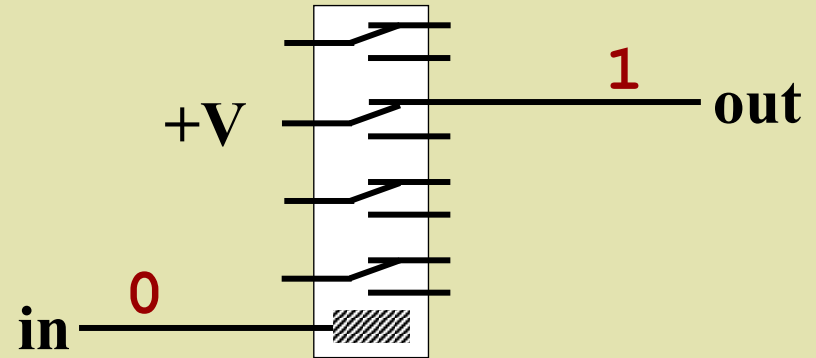
<u>in</u>	<u>out</u>
0	1
1	0



# The “NOT” Circuit



<u>in</u>	<u>out</u>
0	1
1	0

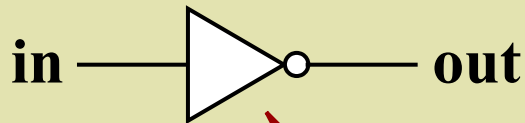


Convention:

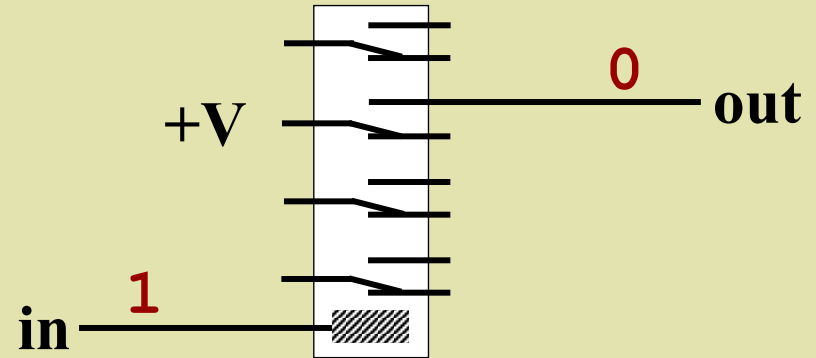
“1” = +12V

“0” = not connected

# The “NOT” Circuit

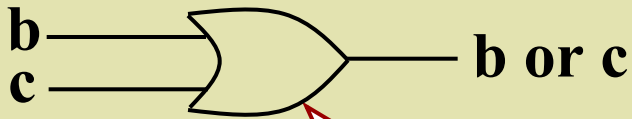


<u>in</u>	<u>out</u>
0	1
1	0

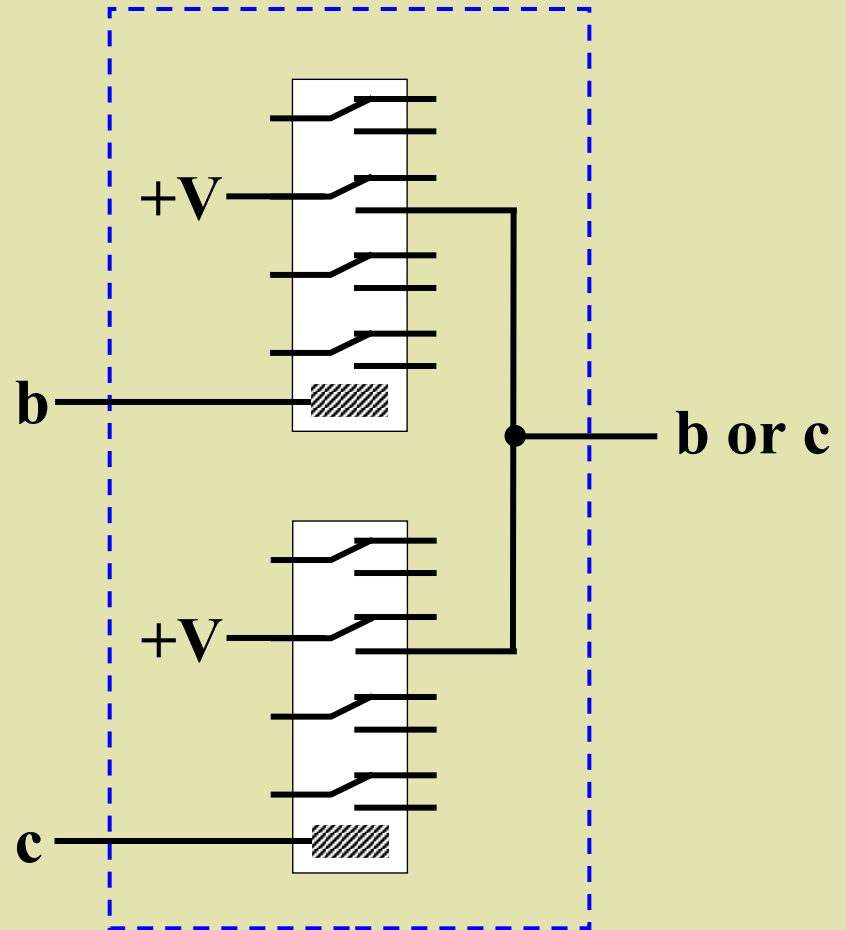


Convention:  
“1” = +12V  
“0” = not connected

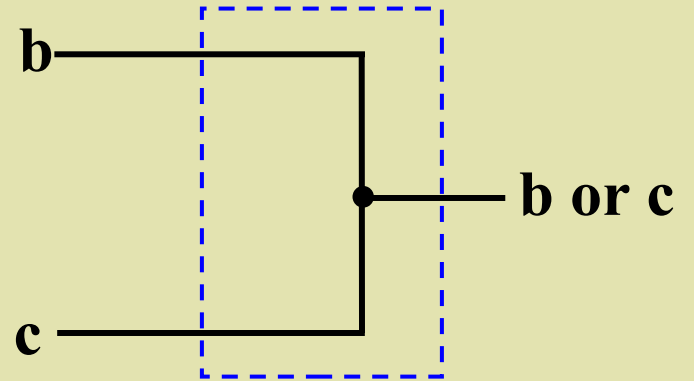
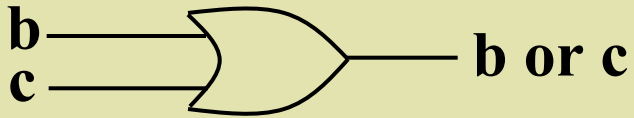
# The "OR" Circuit



<b>b</b>	<b>c</b>	<b>OUT</b>
0	0	0
0	1	1
1	0	1
1	1	1

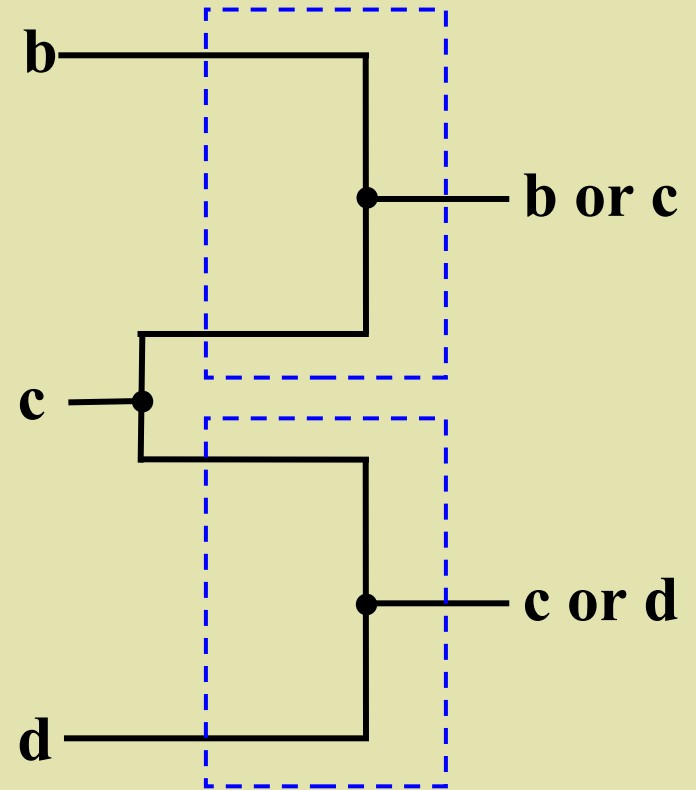
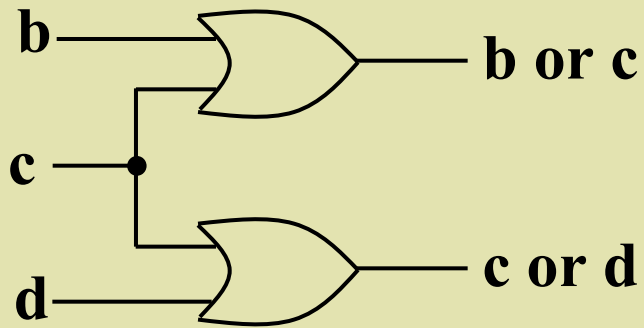


# An Optimization?

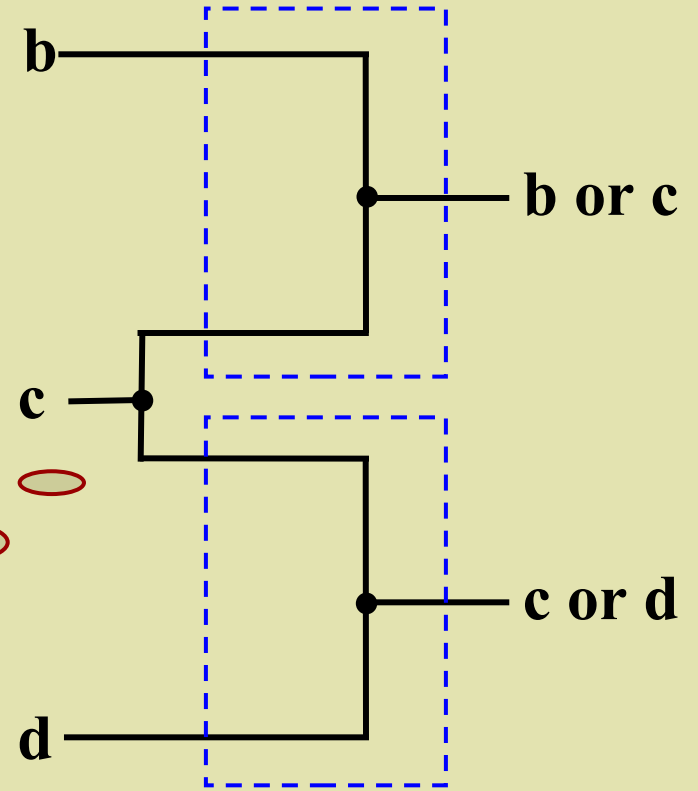
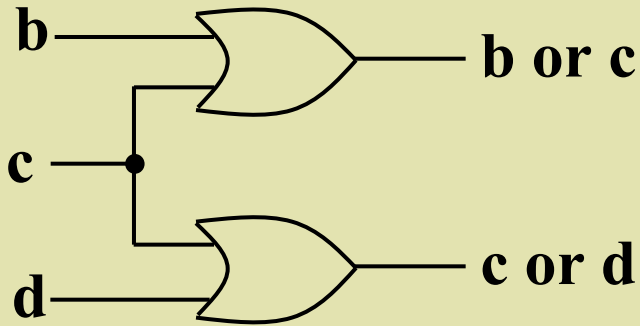




# An Optimization?

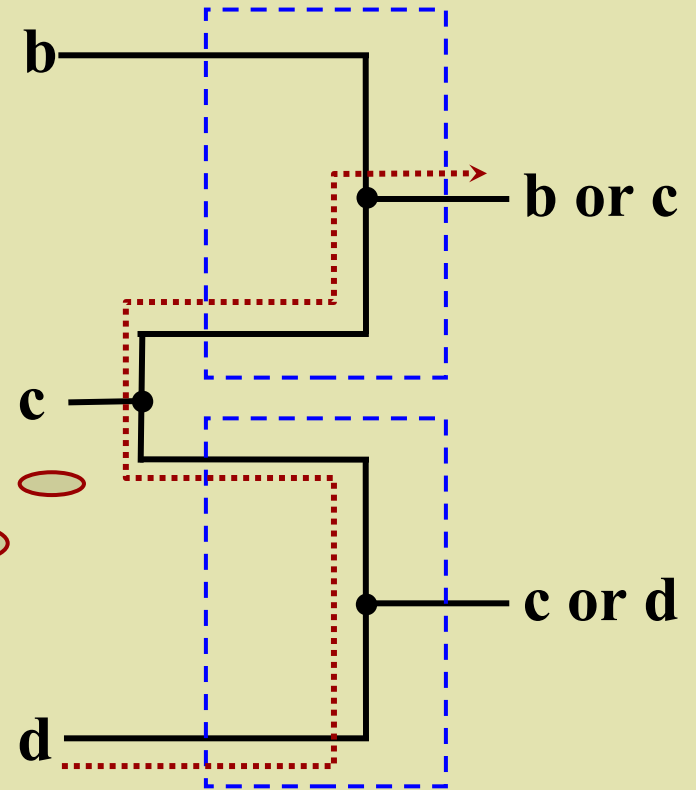
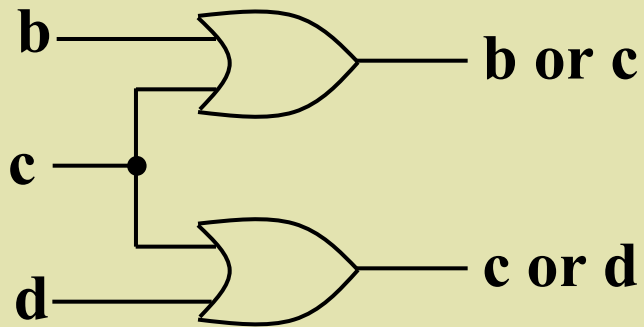


# An Optimization?



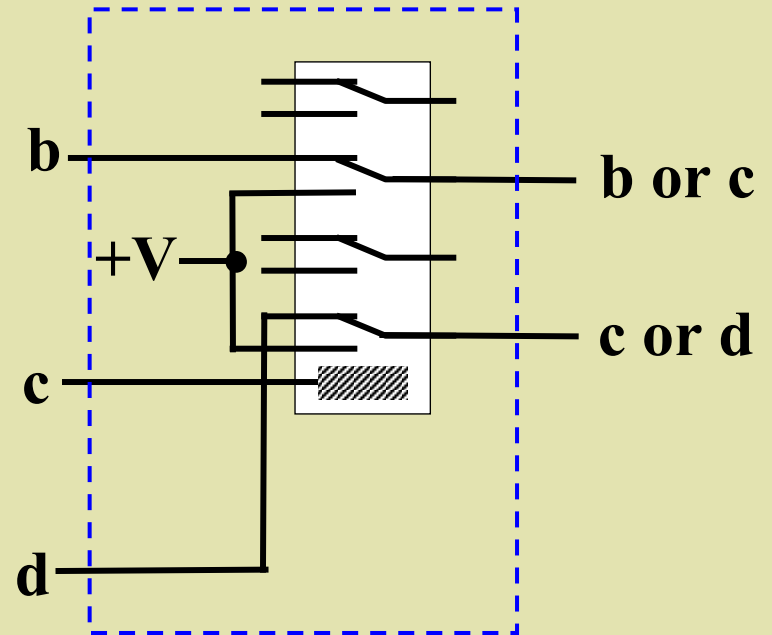
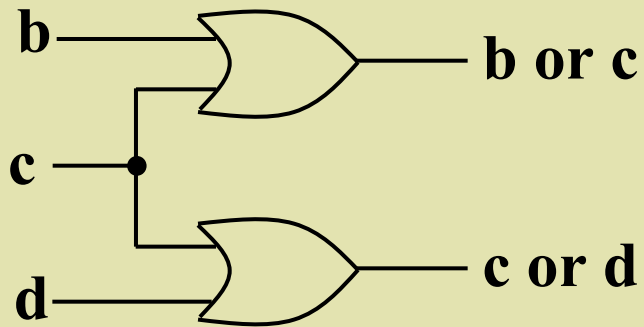
**Whoops!**

# An Optimization?



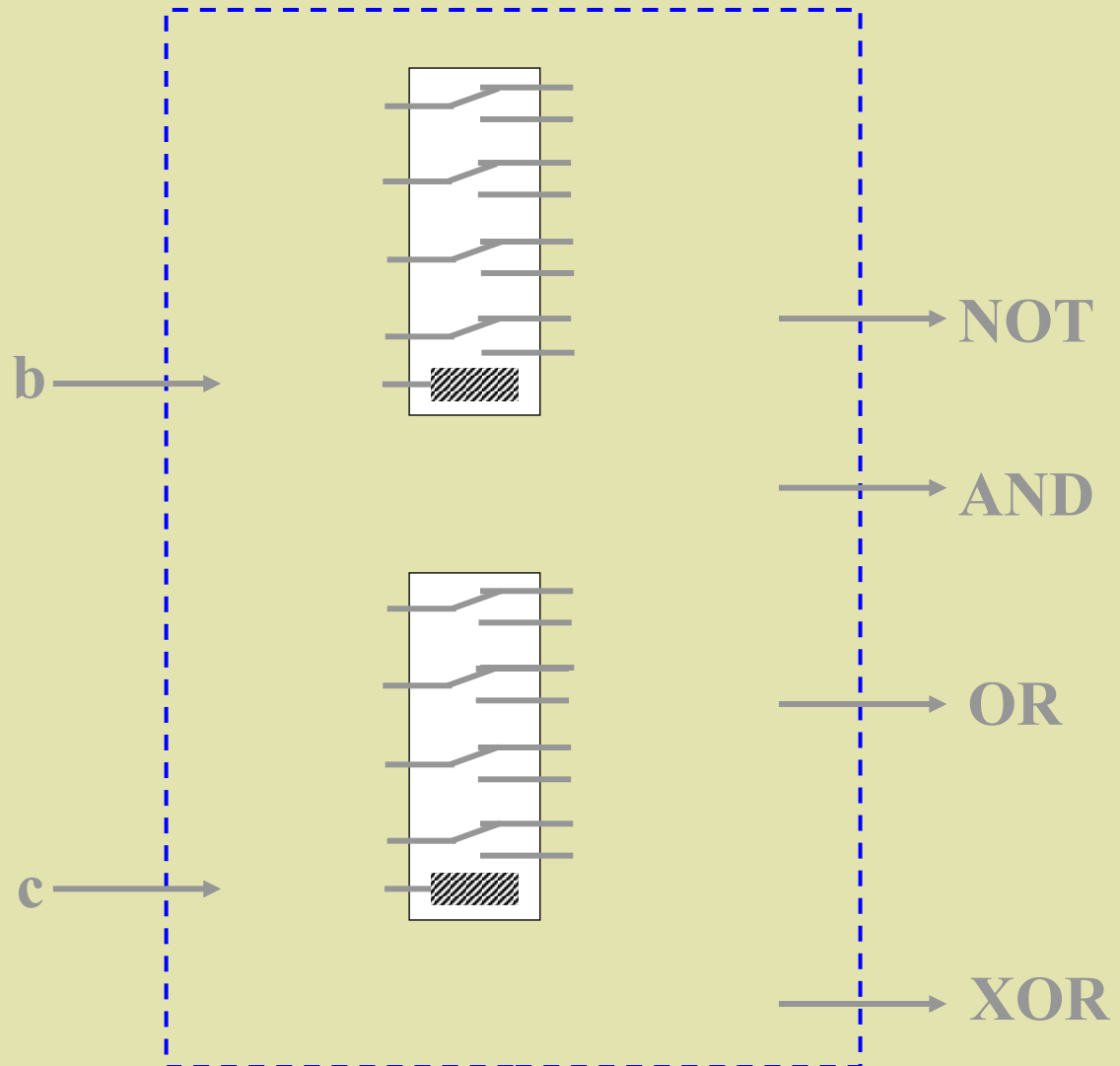
**Whoops!**

# An Optimization?



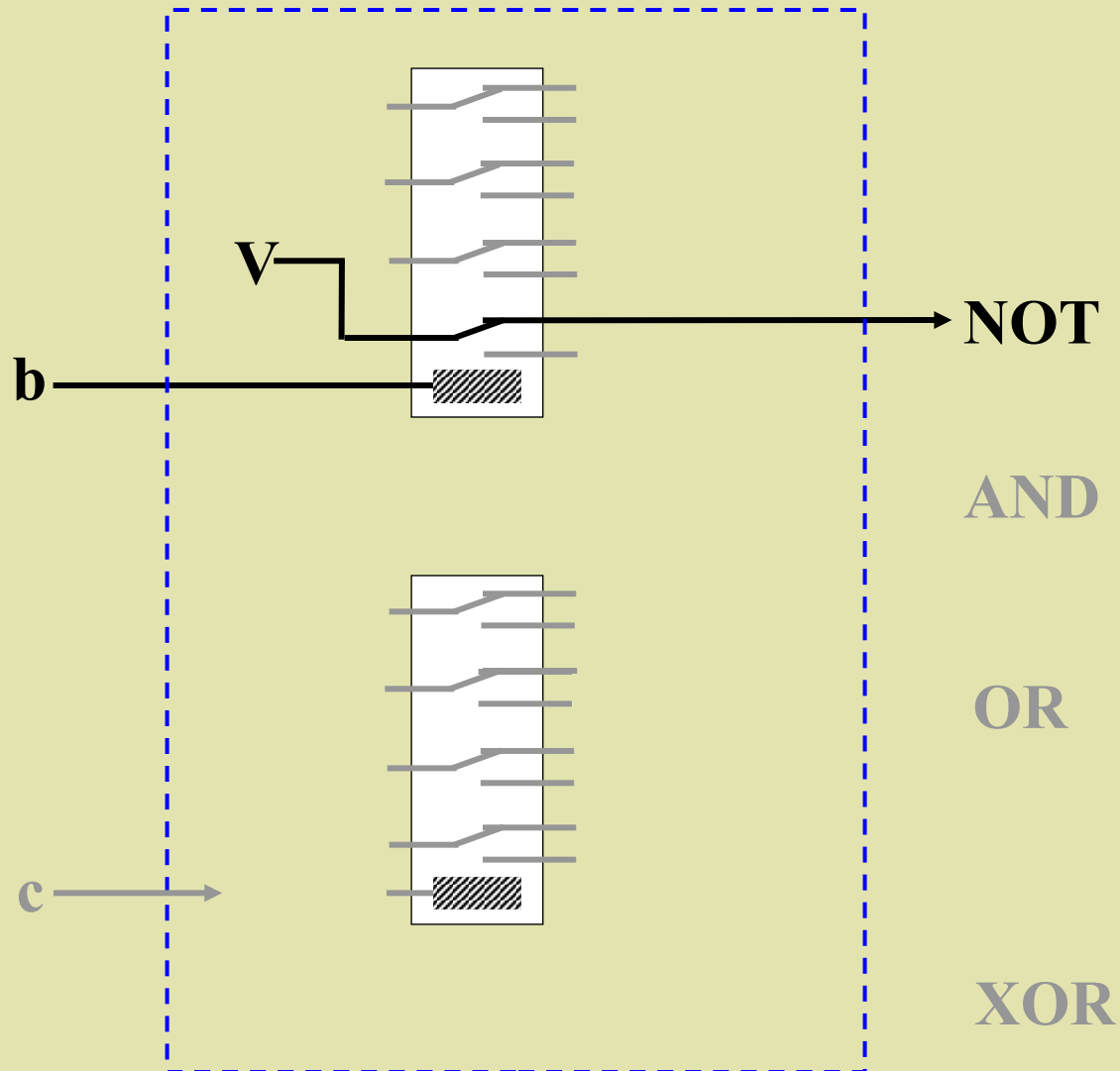
# 1-Bit Logic Circuit

<b>b</b>	<b>c</b>	<b>NOT</b>	<b>AND</b>	<b>OR</b>	<b>XOR</b>
0	0	1	0	0	0
0	1	1	0	1	1
1	0	0	0	1	1
1	1	0	1	1	0



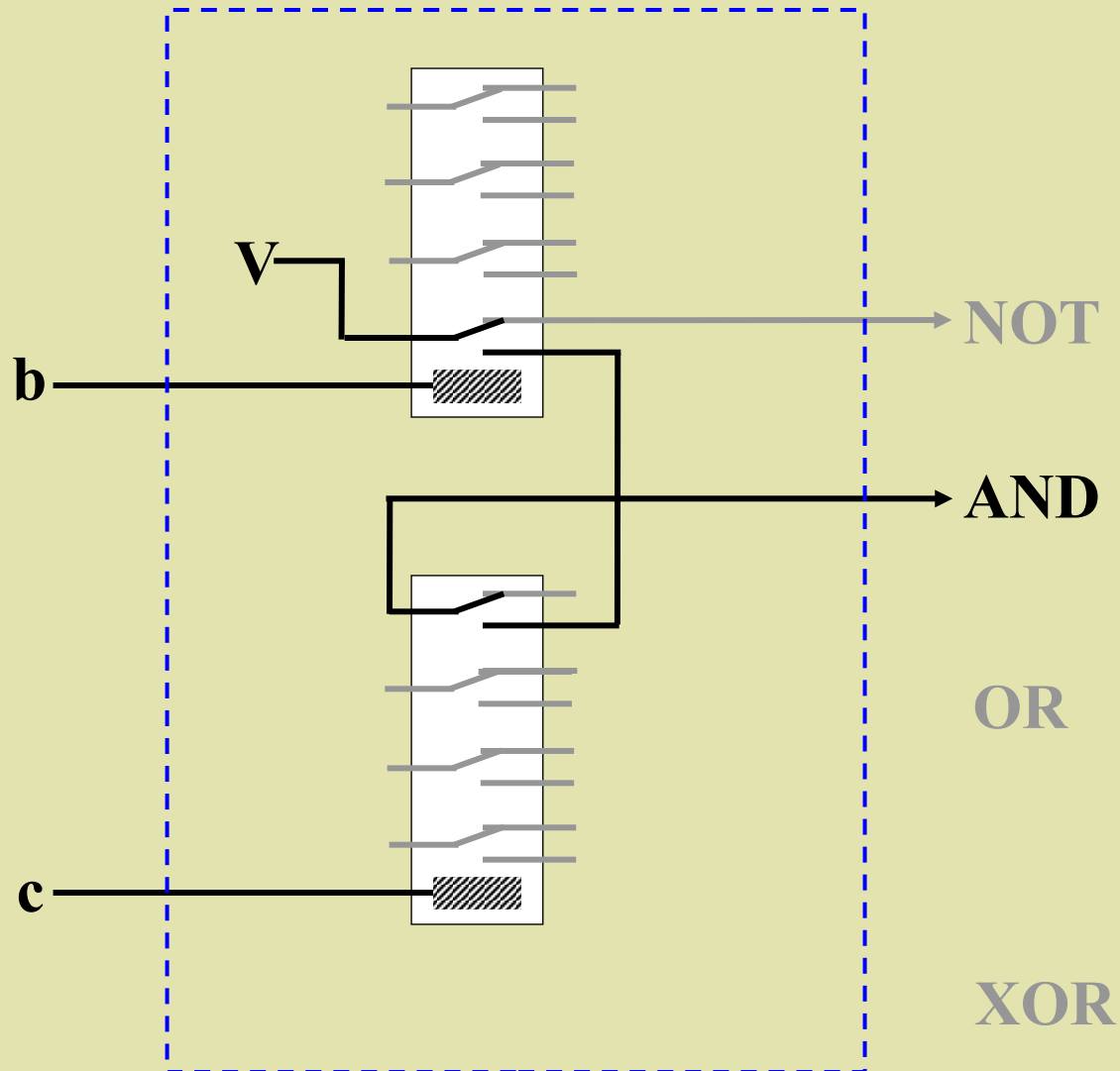
# 1-Bit Logic Circuit

<b>b</b>	<b>c</b>	<b>NOT</b>	<b>AND</b>	<b>OR</b>	<b>XOR</b>
0	0	1	0	0	0
0	1	1	0	1	1
1	0	0	0	1	1
1	1	0	1	1	0



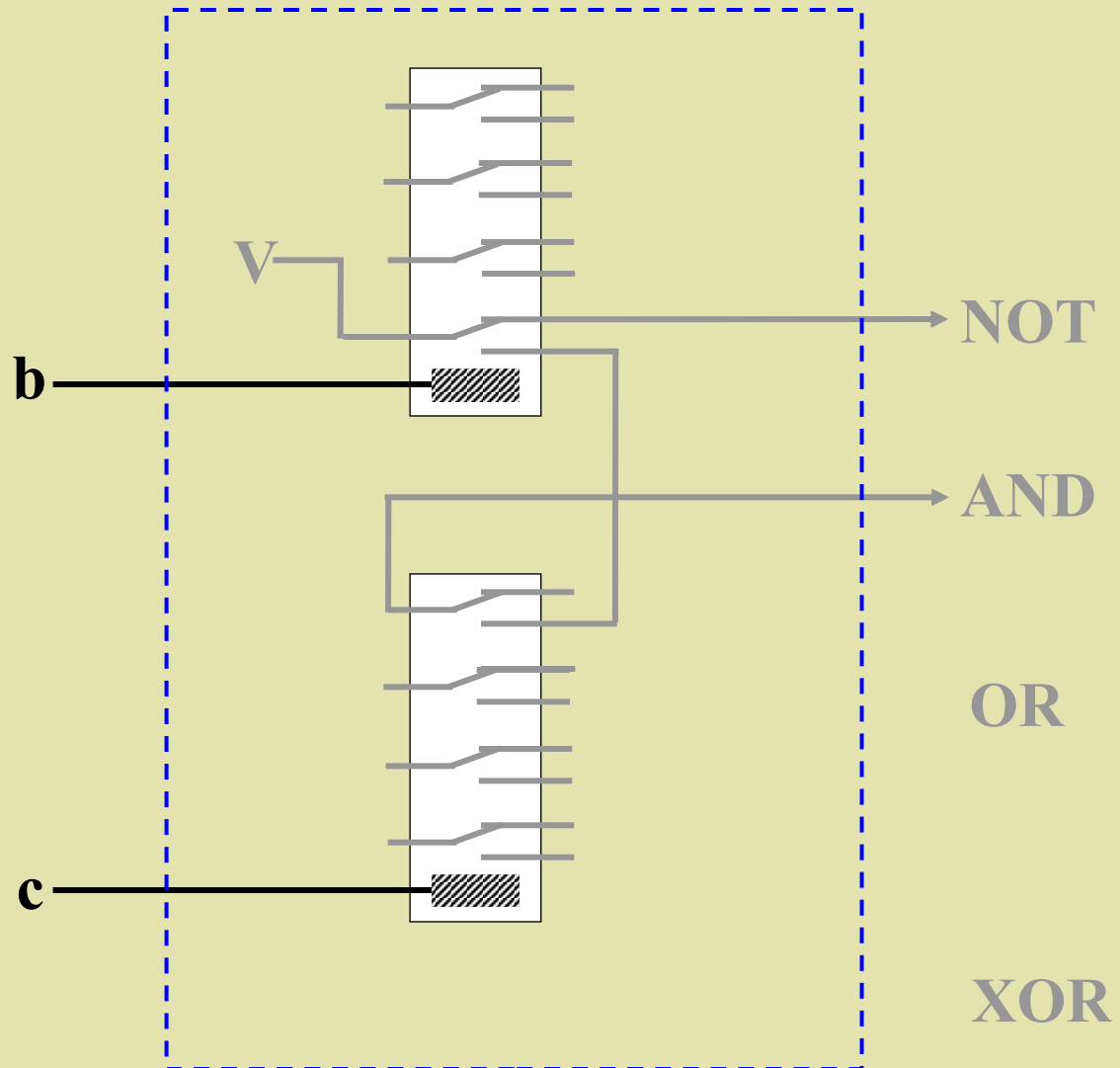
# 1-Bit Logic Circuit

<b>b</b>	<b>c</b>	<b>NOT</b>	<b>AND</b>	<b>OR</b>	<b>XOR</b>
0	0	1	0	0	0
0	1	1	0	1	1
1	0	0	0	1	1
1	1	0	1	1	0



# 1-Bit Logic Circuit

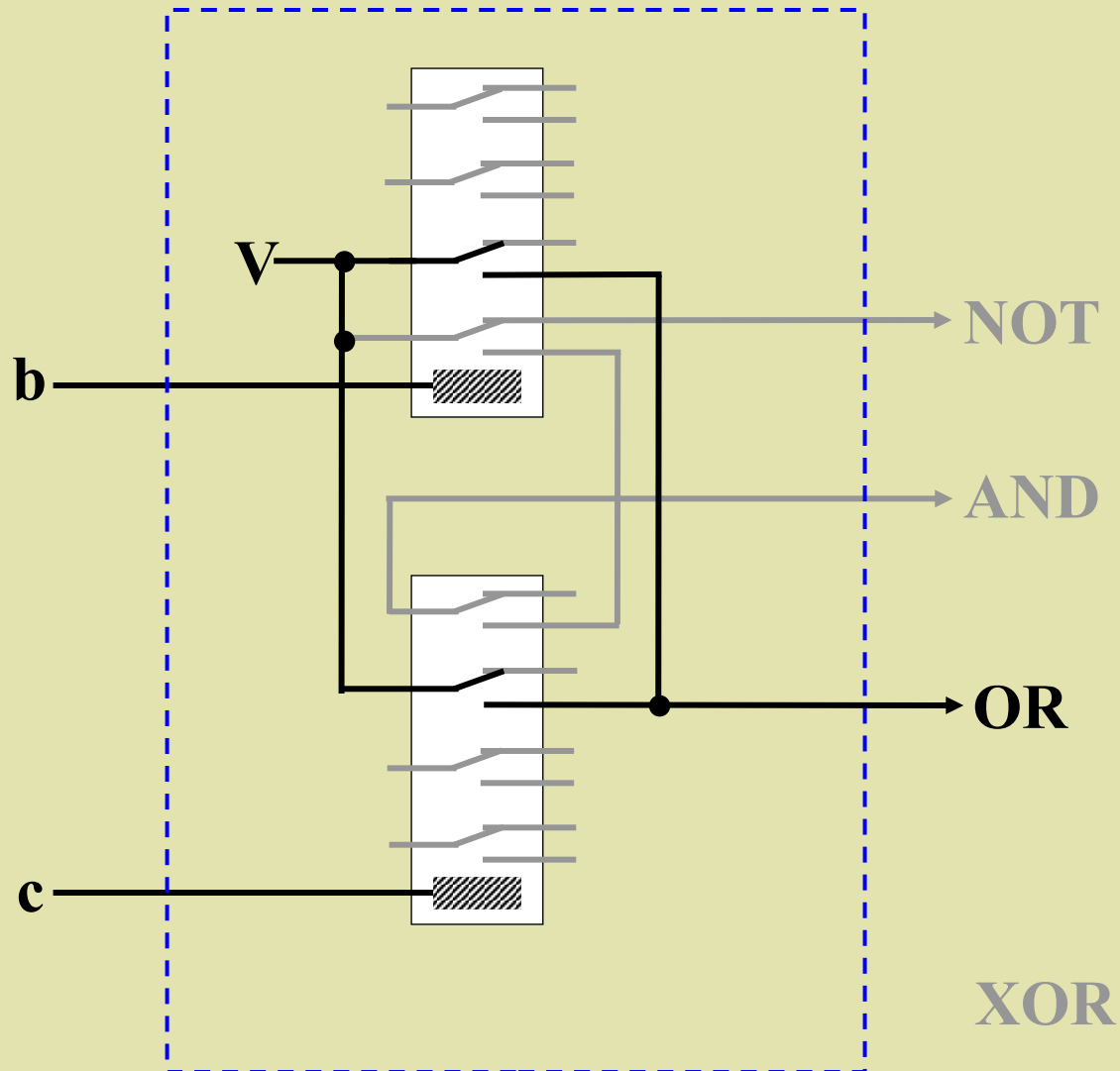
<b>b</b>	<b>c</b>	<b>NOT</b>	<b>AND</b>	<b>OR</b>	<b>XOR</b>
0	0	1	0	0	0
0	1	1	0	1	1
1	0	0	0	1	1
1	1	0	1	1	0





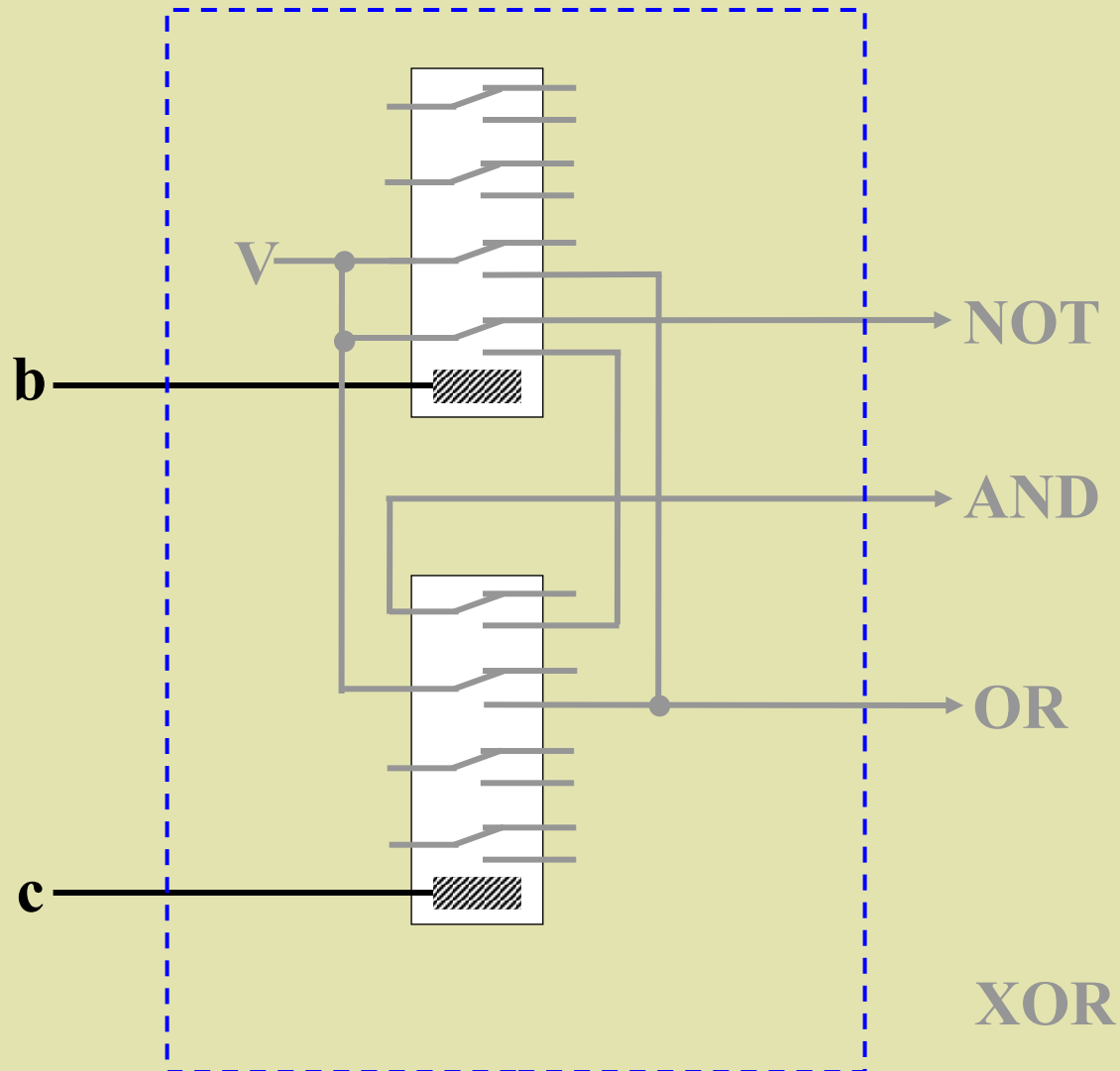
# 1-Bit Logic Circuit

<b>b</b>	<b>c</b>	<b>NOT</b>	<b>AND</b>	<b>OR</b>	<b>XOR</b>
0	0	1	0	0	0
0	1	1	0	1	1
1	0	0	0	1	1
1	1	0	1	1	0



# 1-Bit Logic Circuit

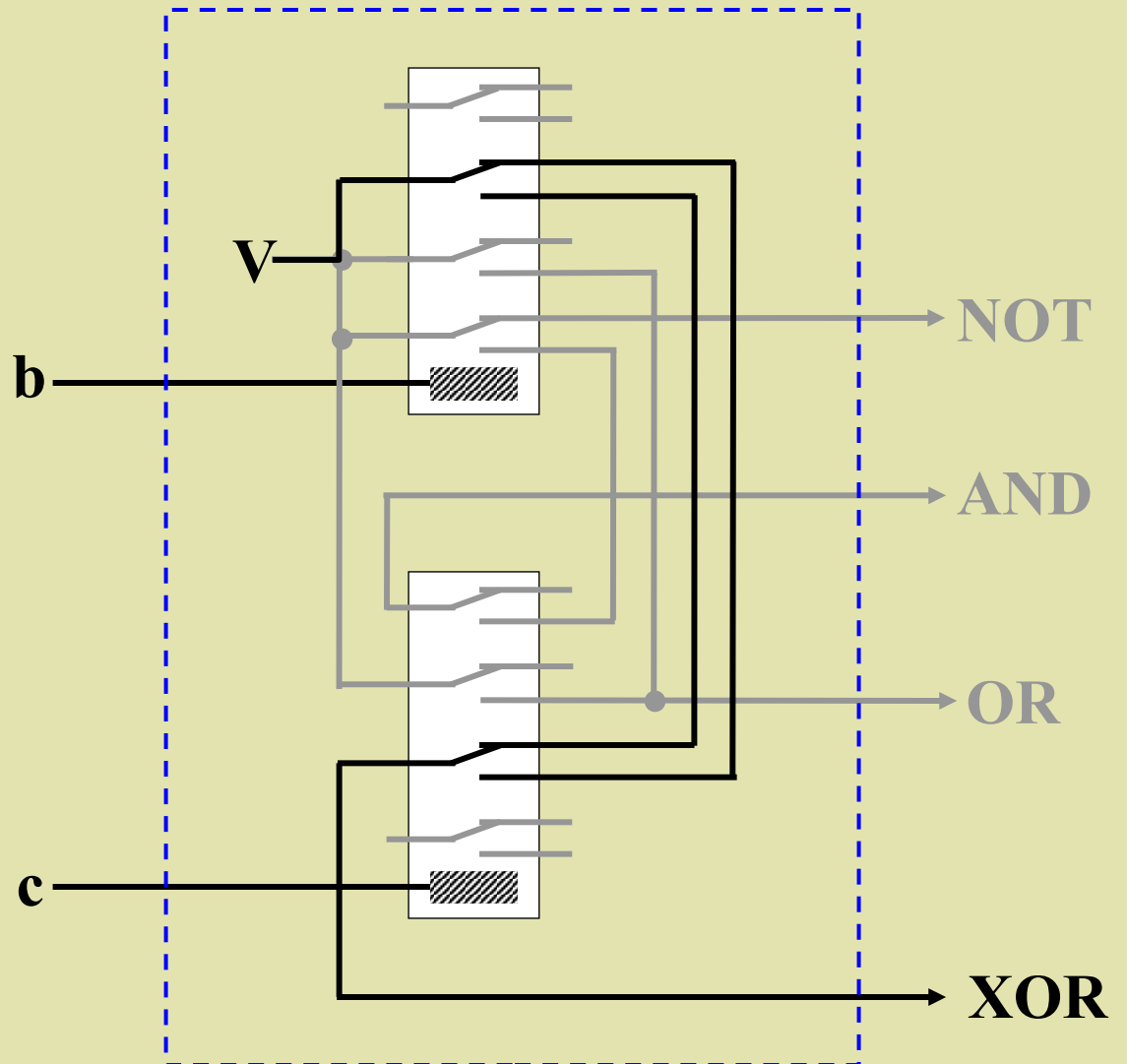
<b>b</b>	<b>c</b>	<b>NOT</b>	<b>AND</b>	<b>OR</b>	<b>XOR</b>
0	0	1	0	0	0
0	1	1	0	1	1
1	0	0	0	1	1
1	1	0	1	1	0



XOR

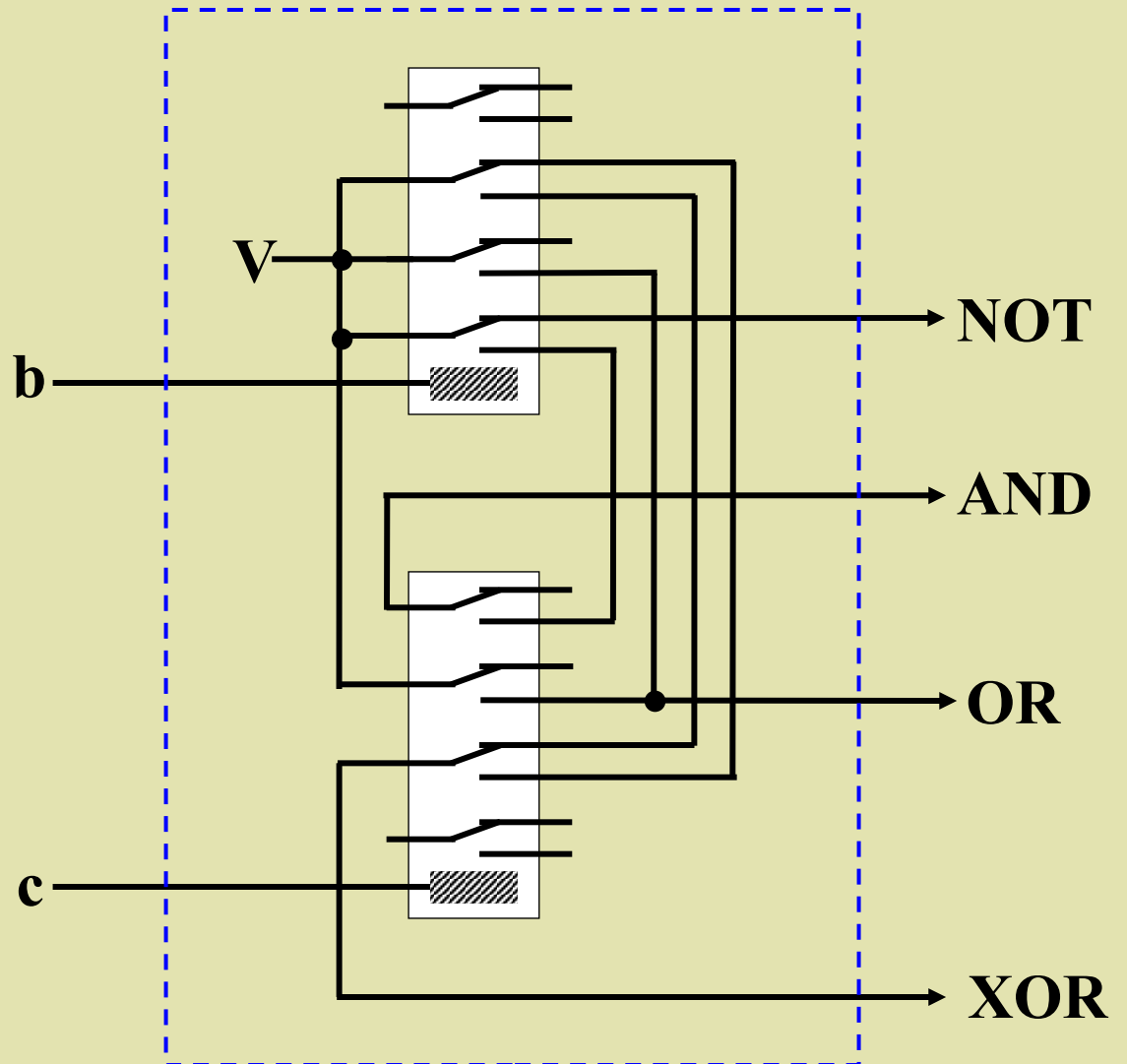
# 1-Bit Logic Circuit

<b>b</b>	<b>c</b>	<b>NOT</b>	<b>AND</b>	<b>OR</b>	<b>XOR</b>
0	0	1	0	0	0
0	1	1	0	1	1
1	0	0	0	1	1
1	1	0	1	1	0



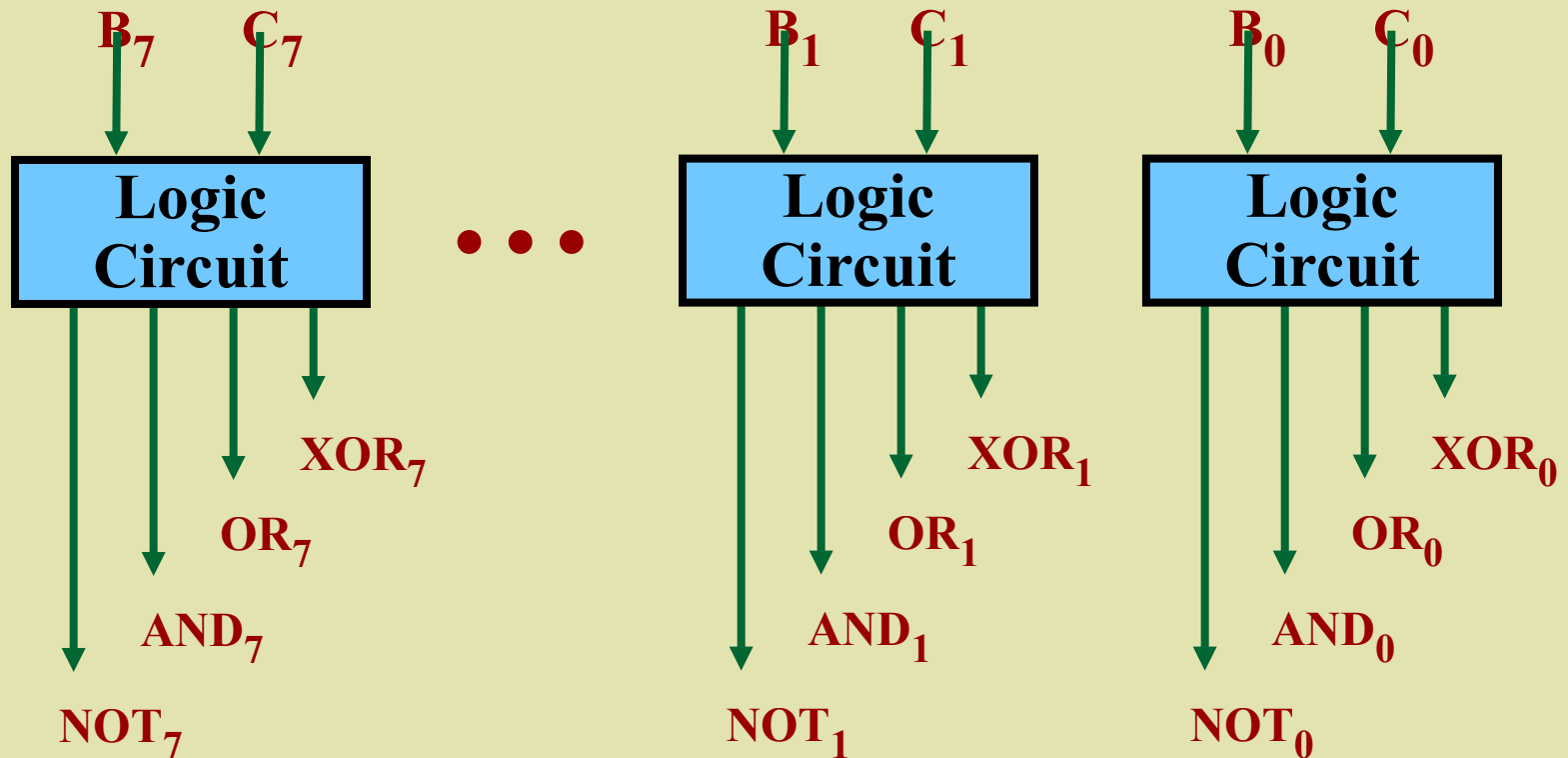
# 1-Bit Logic Circuit

<b>b</b>	<b>c</b>	<b>NOT</b>	<b>AND</b>	<b>OR</b>	<b>XOR</b>
0	0	1	0	0	0
0	1	1	0	1	1
1	0	0	0	1	1
1	1	0	1	1	0



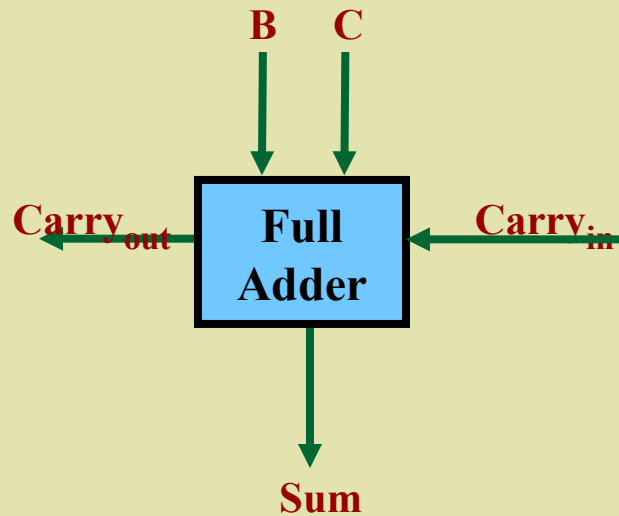
*Eight 1-bit circuits can be combined  
to build an...*

## 8-Bit Logic Circuit



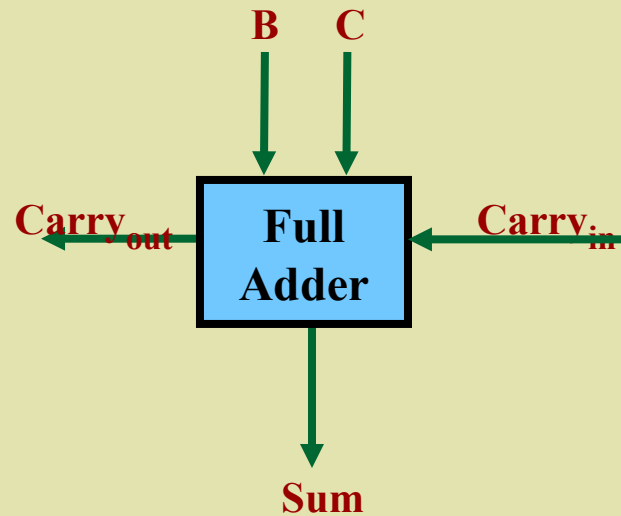
# The “Full Adder” Circuit

$Cy_{in}$	B	C	$Cy_{out}$	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



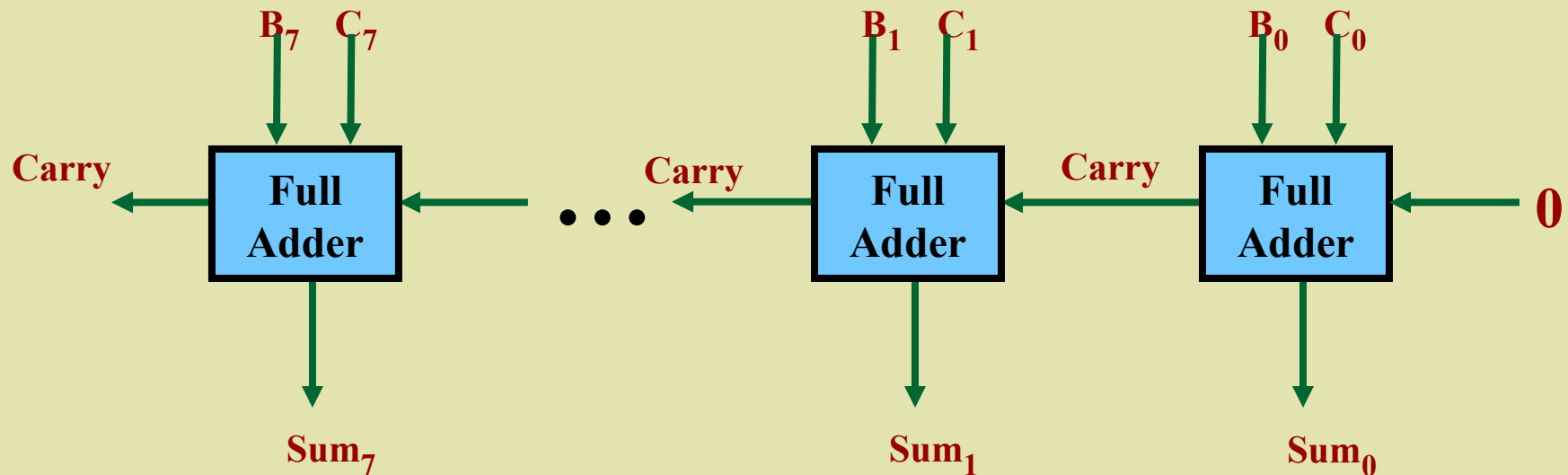
*8 full-adders can be combined to build an...*

## 8-Bit Adder



*8 full-adders can be combined to build an...*

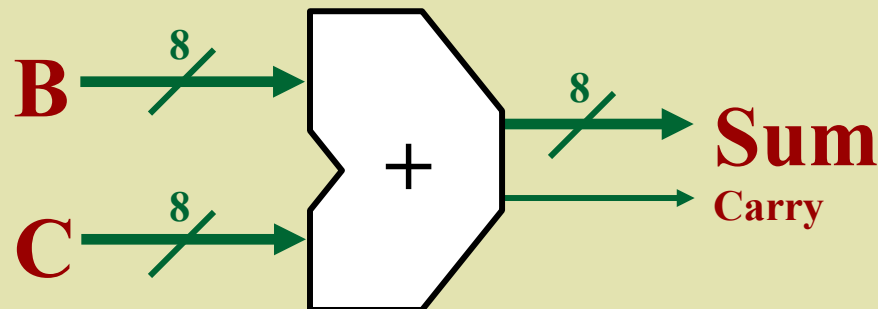
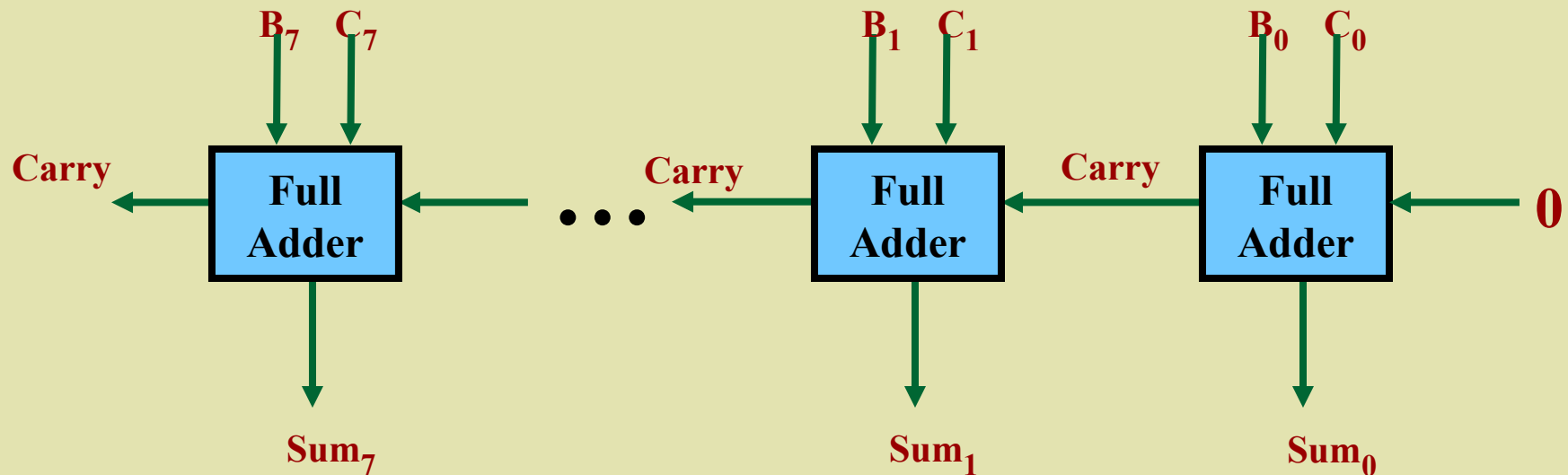
## 8-Bit Adder



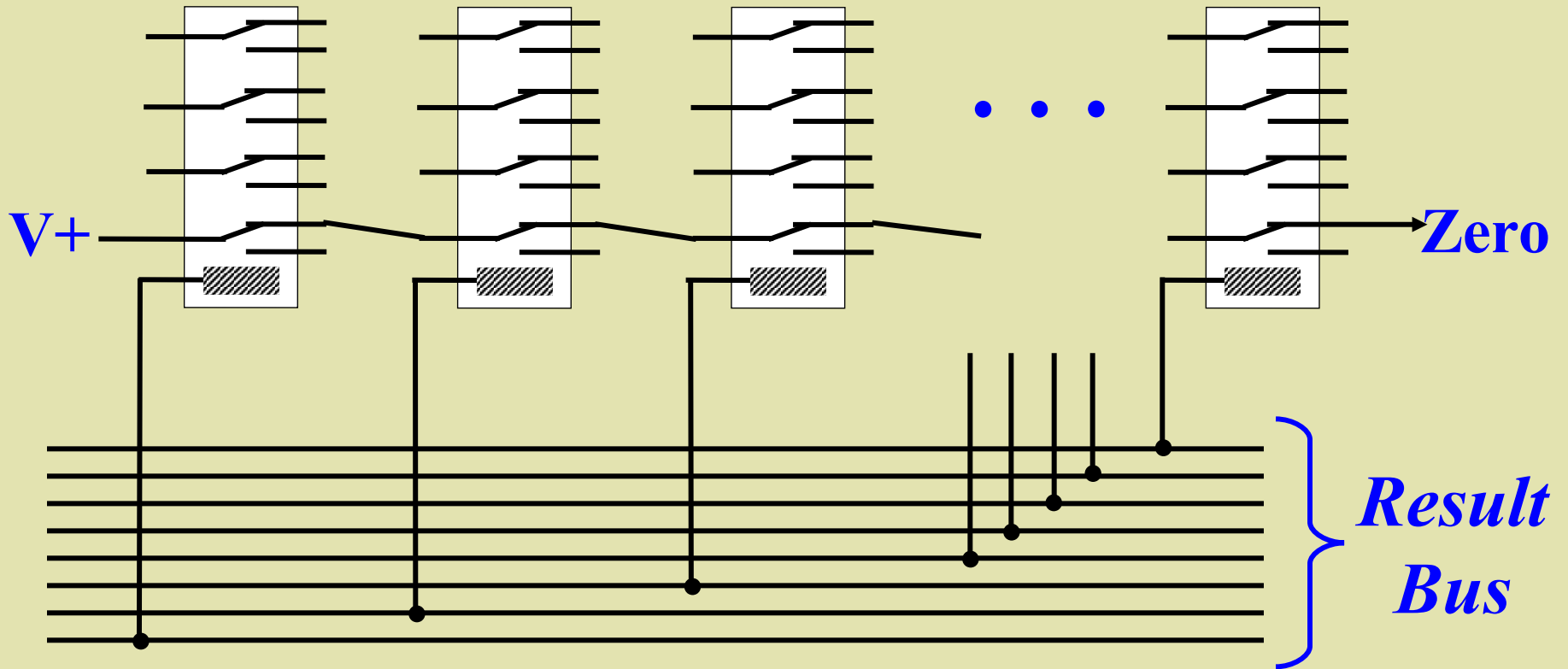


*8 full-adders can be combined to build an...*

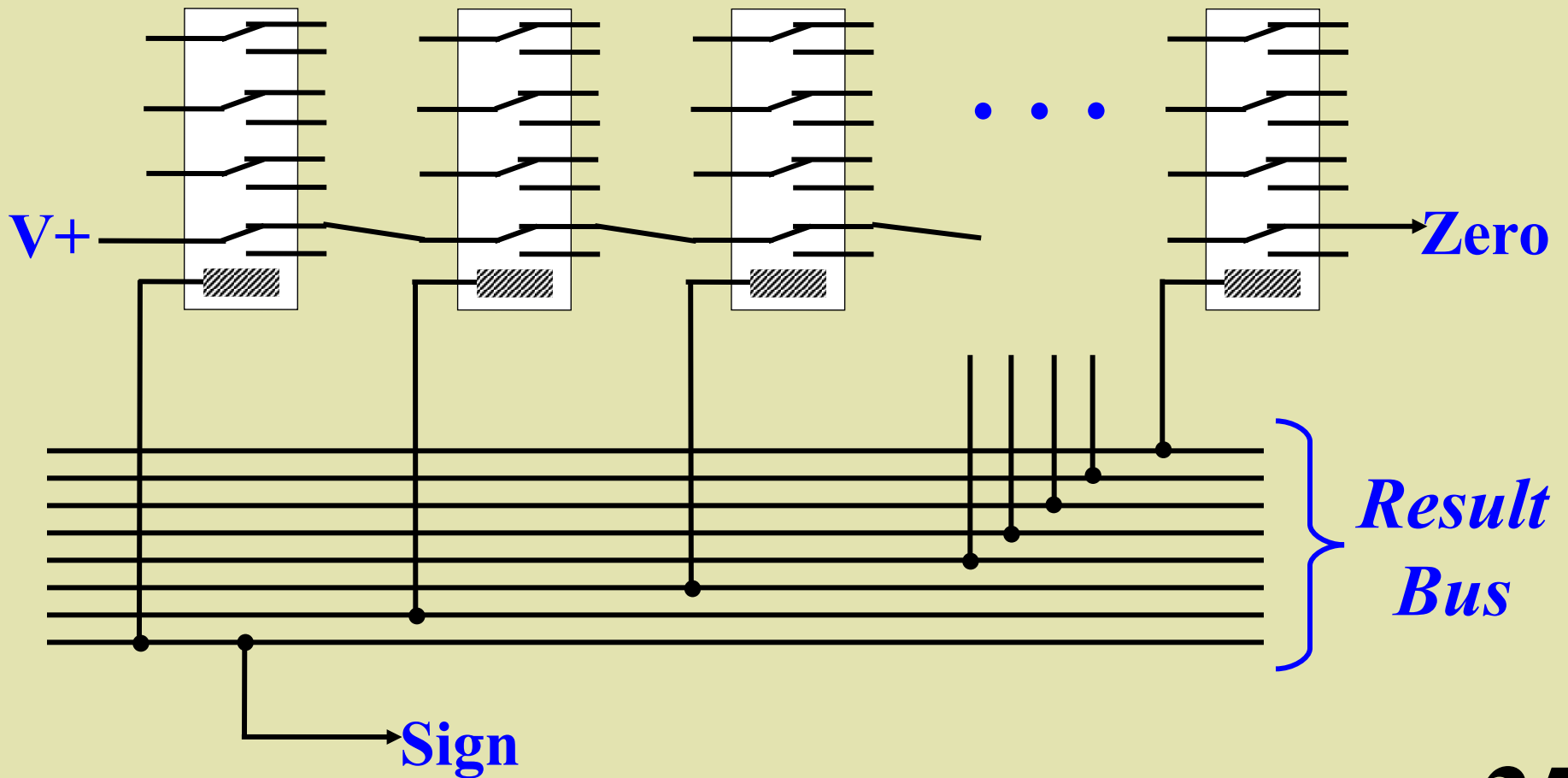
## 8-Bit Adder



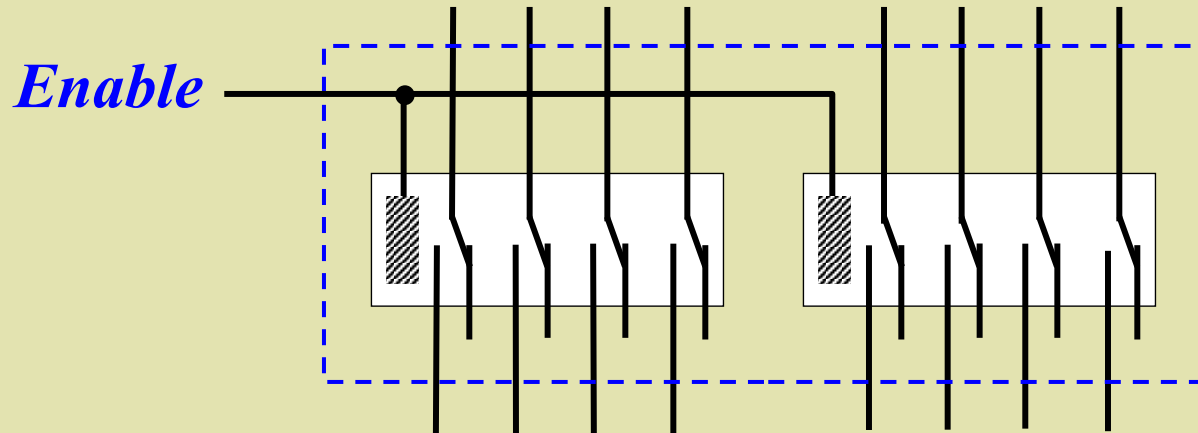
# The Zero-Detect Circuit



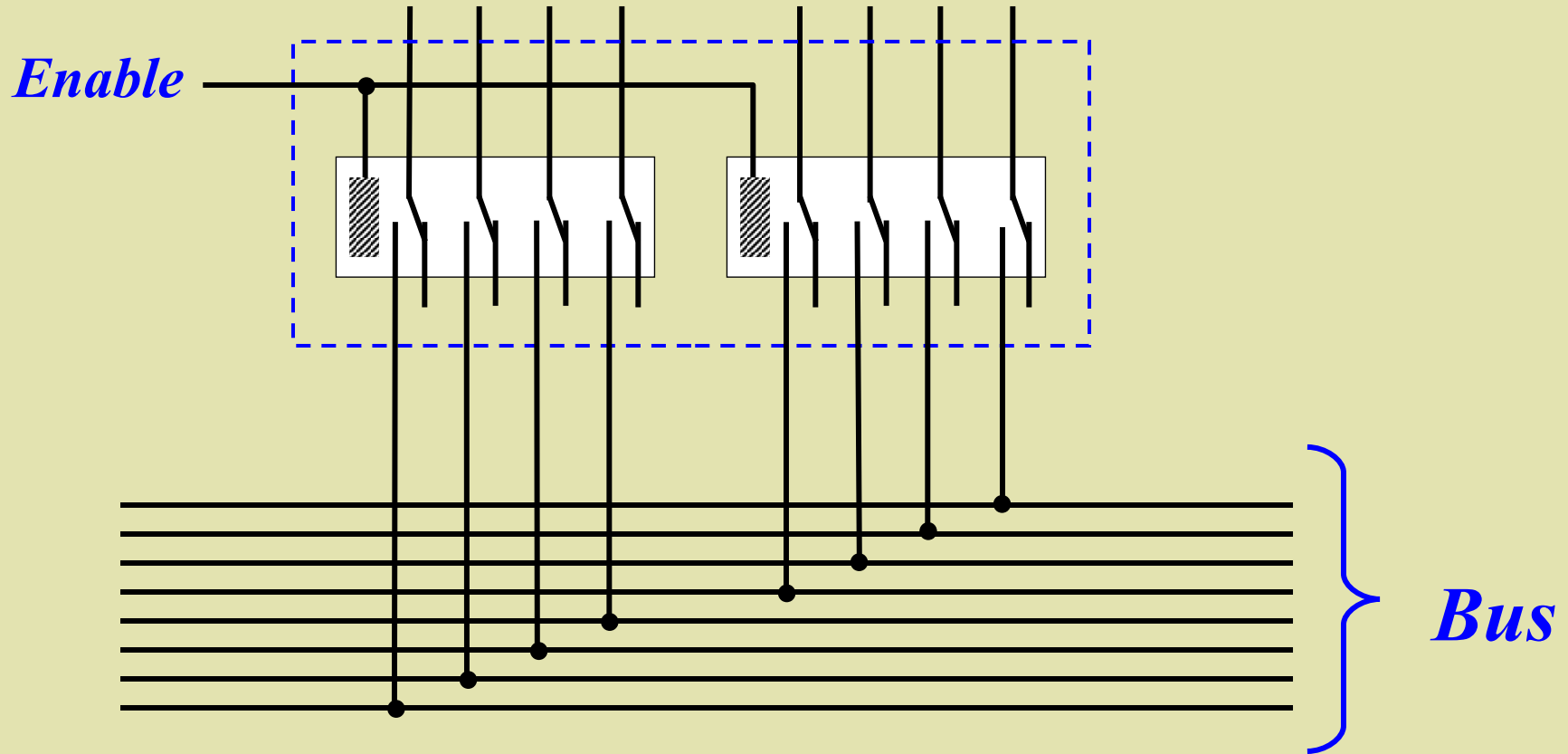
# The Zero-Detect Circuit



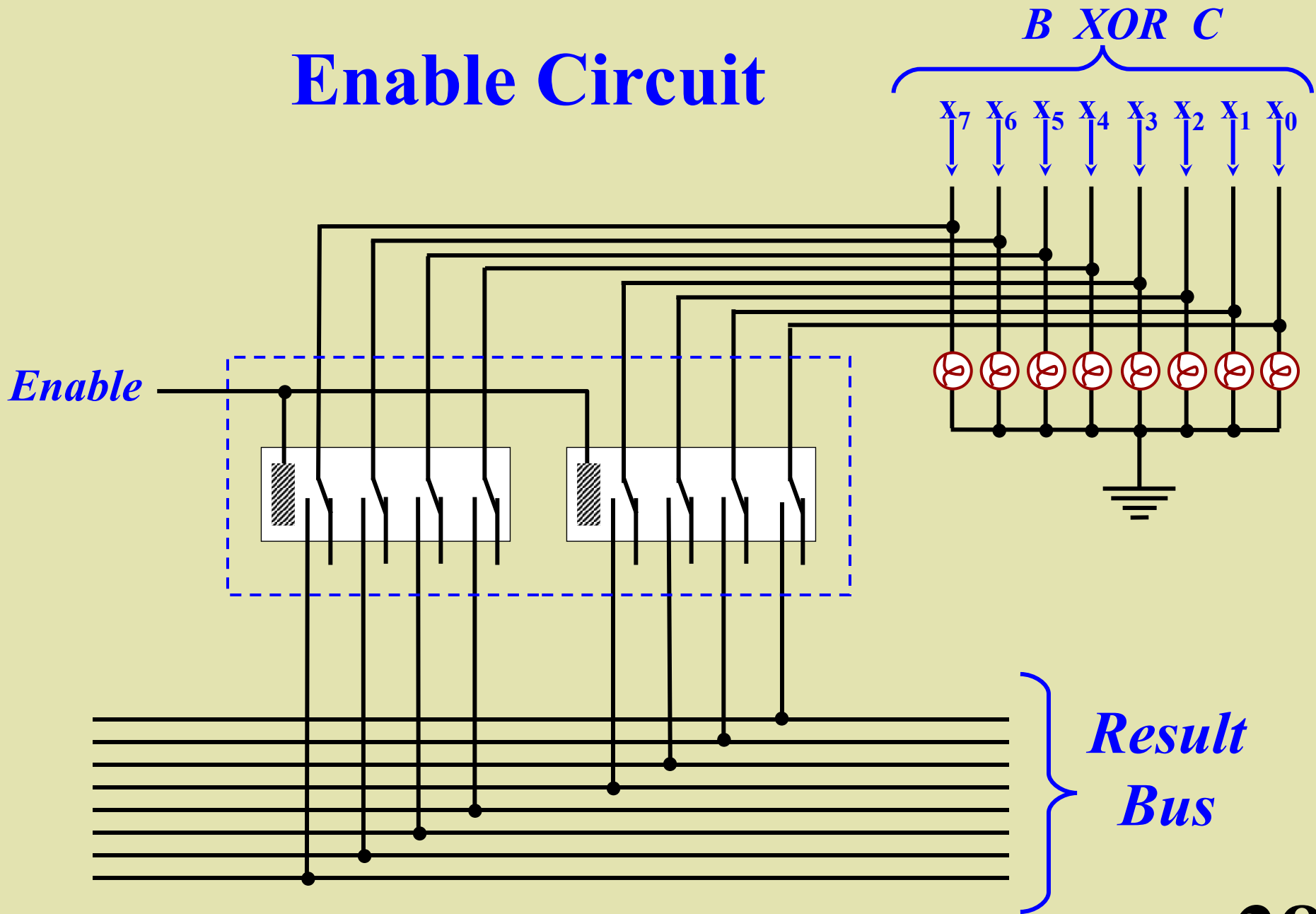
# Enable Circuit



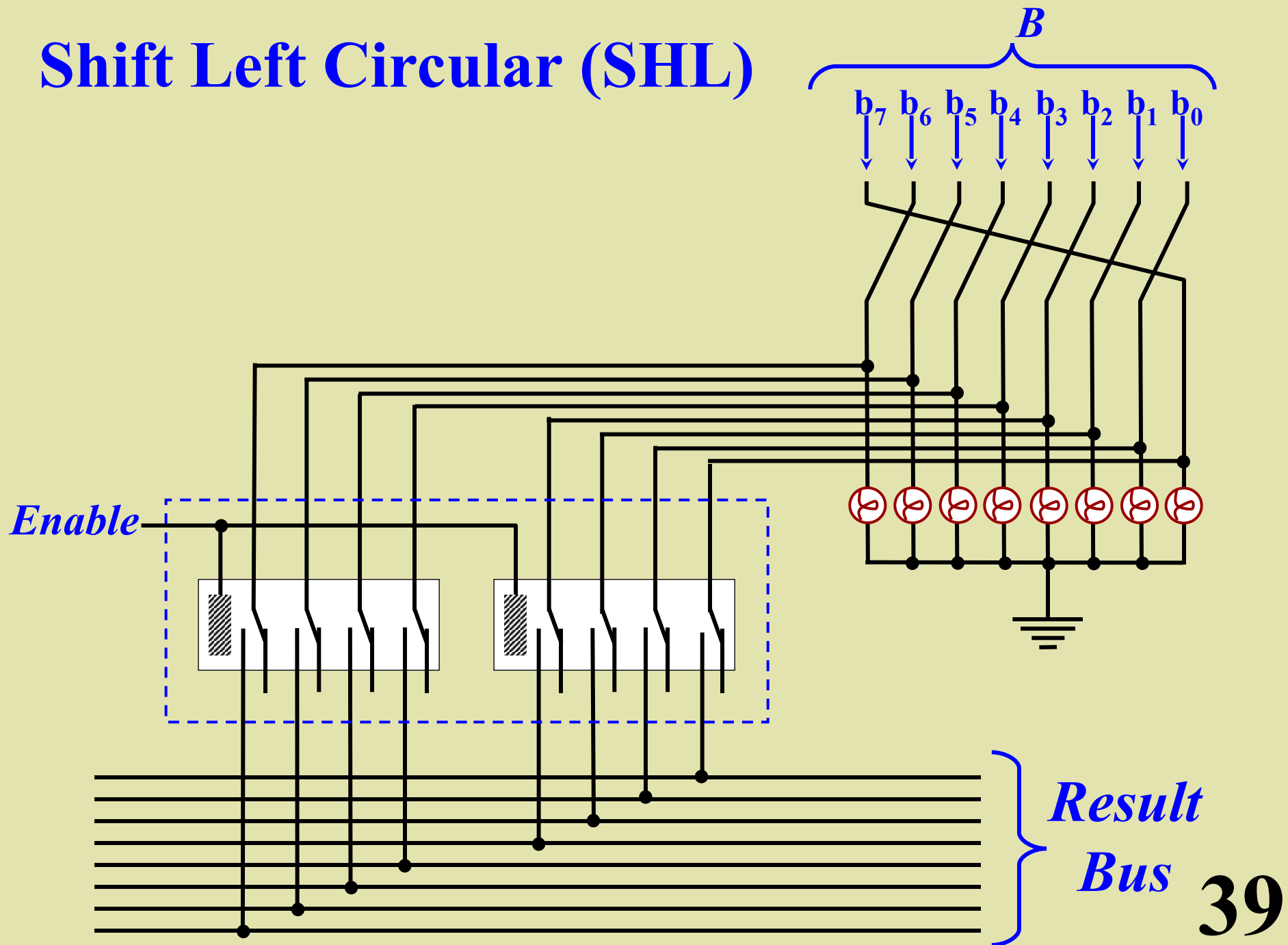
# Enable Circuit



# Enable Circuit



# Shift Left Circular (SHL)



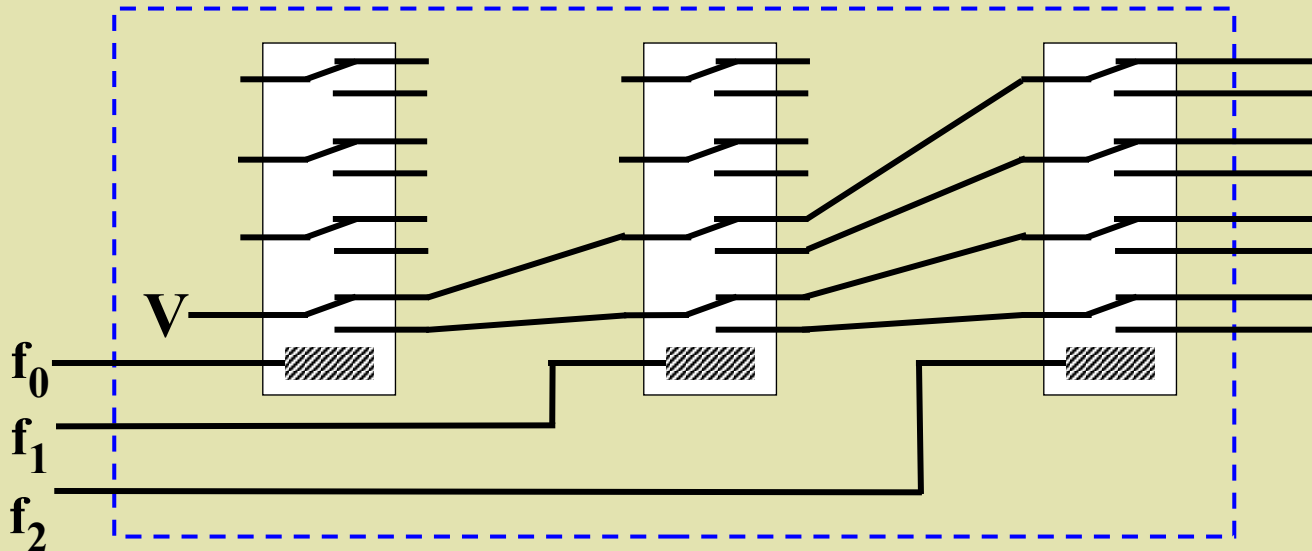
# 3-to-8 Decoder

$f_0$	$f_1$	$f_2$	OUTPUT							
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1



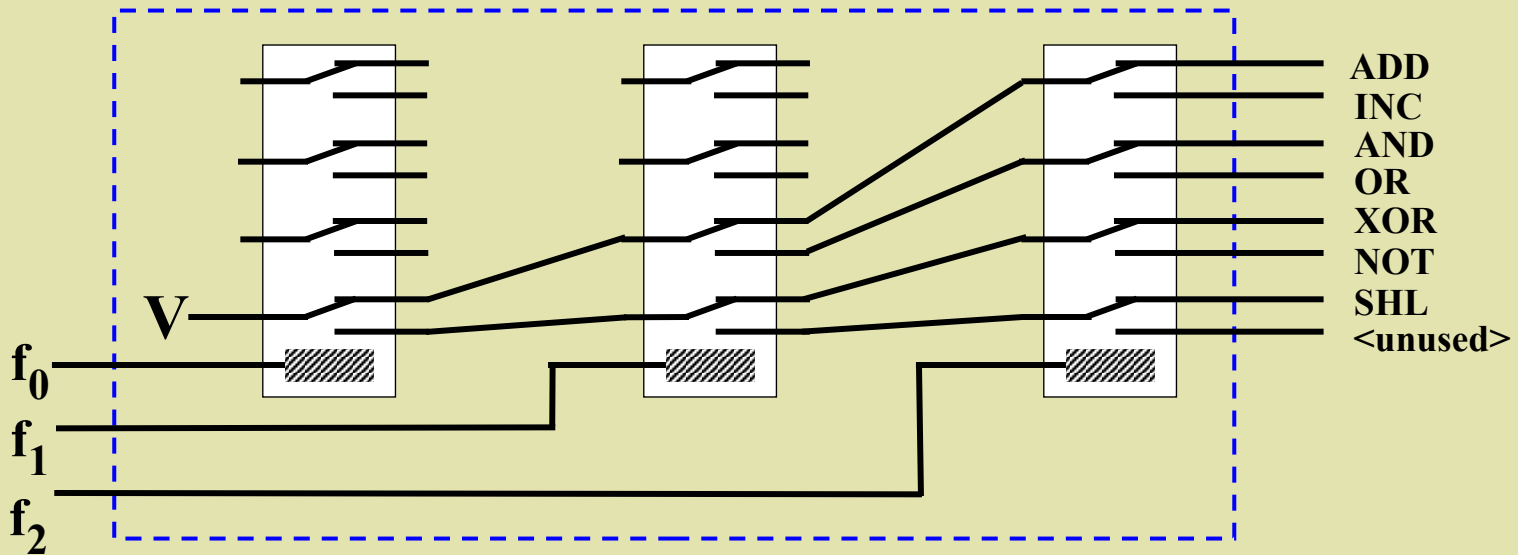
# 3-to-8 Decoder

$f_0$	$f_1$	$f_2$	OUTPUT								
0	0	0	1	0	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	0	0	0	1	0	0	0
1	1	0	0	0	0	0	0	0	1	0	0
1	1	1	0	0	0	0	0	0	0	1	1



# 3-to-8 Decoder

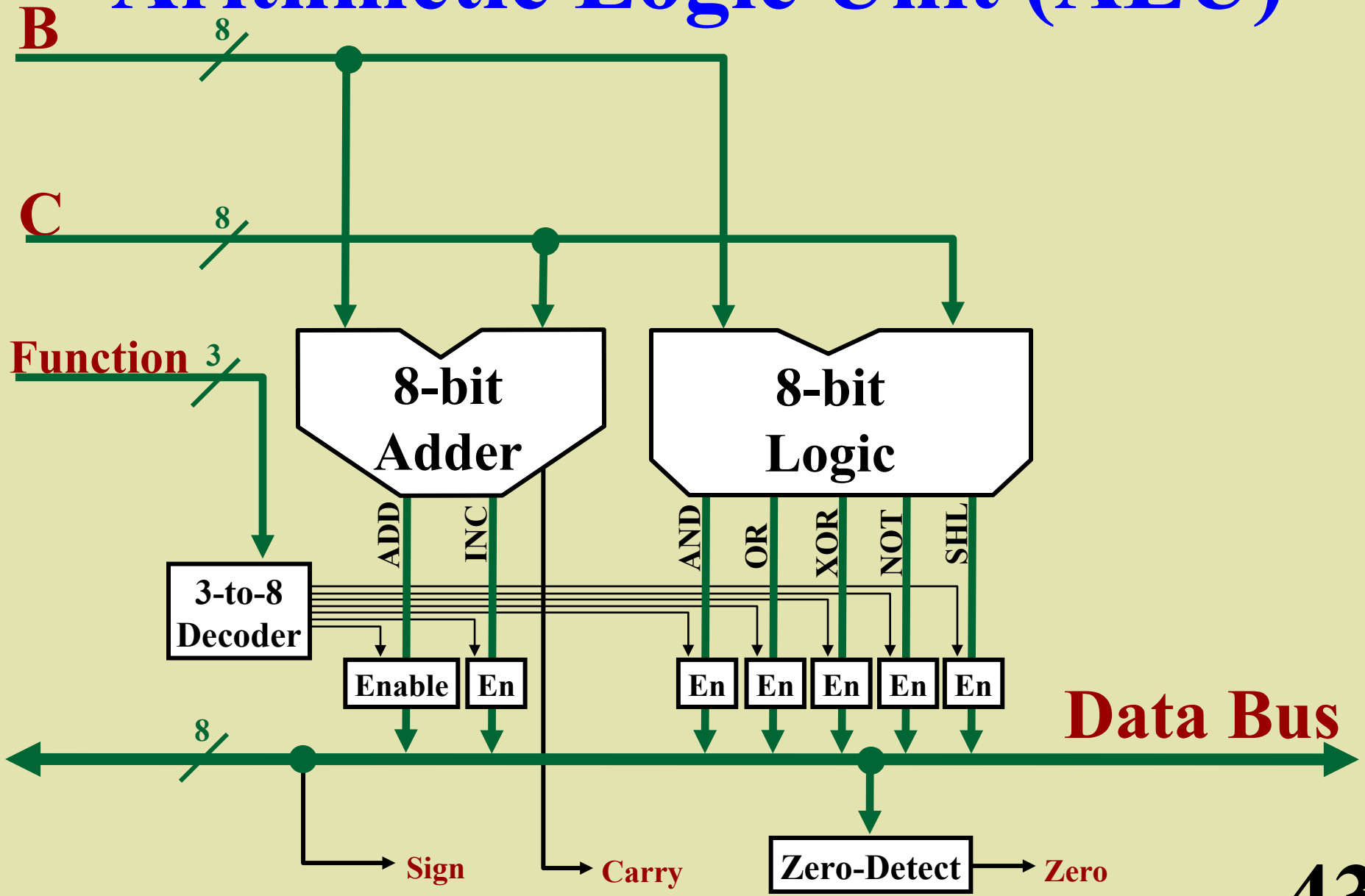
$f_0$	$f_1$	$f_2$	OUTPUT								
0	0	0	1	0	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	0	0	0	1	0	0	0
1	1	0	0	0	0	0	0	0	1	0	0
1	1	1	0	0	0	0	0	0	0	1	1



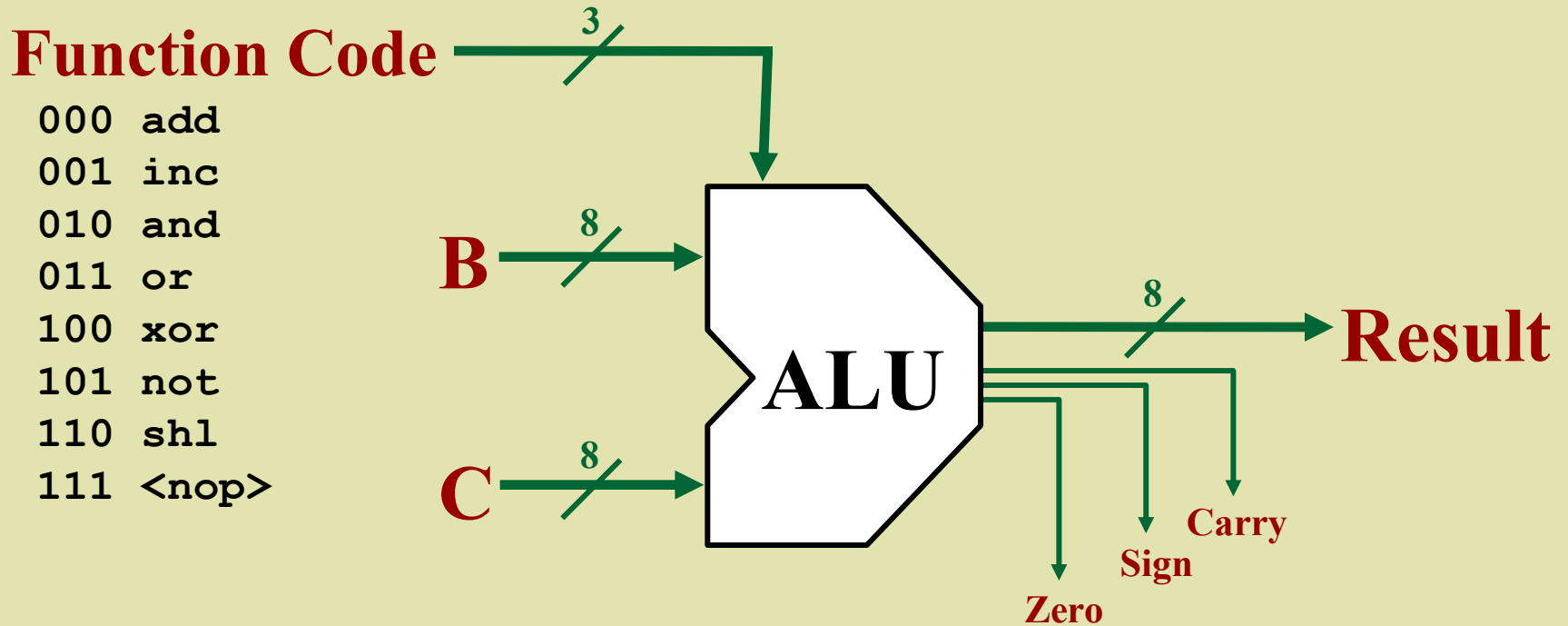
05:12



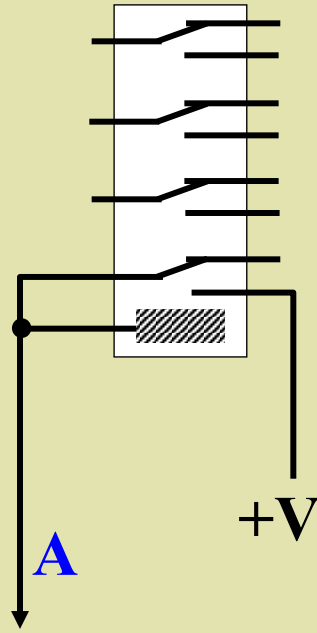
# Arithmetic Logic Unit (ALU)



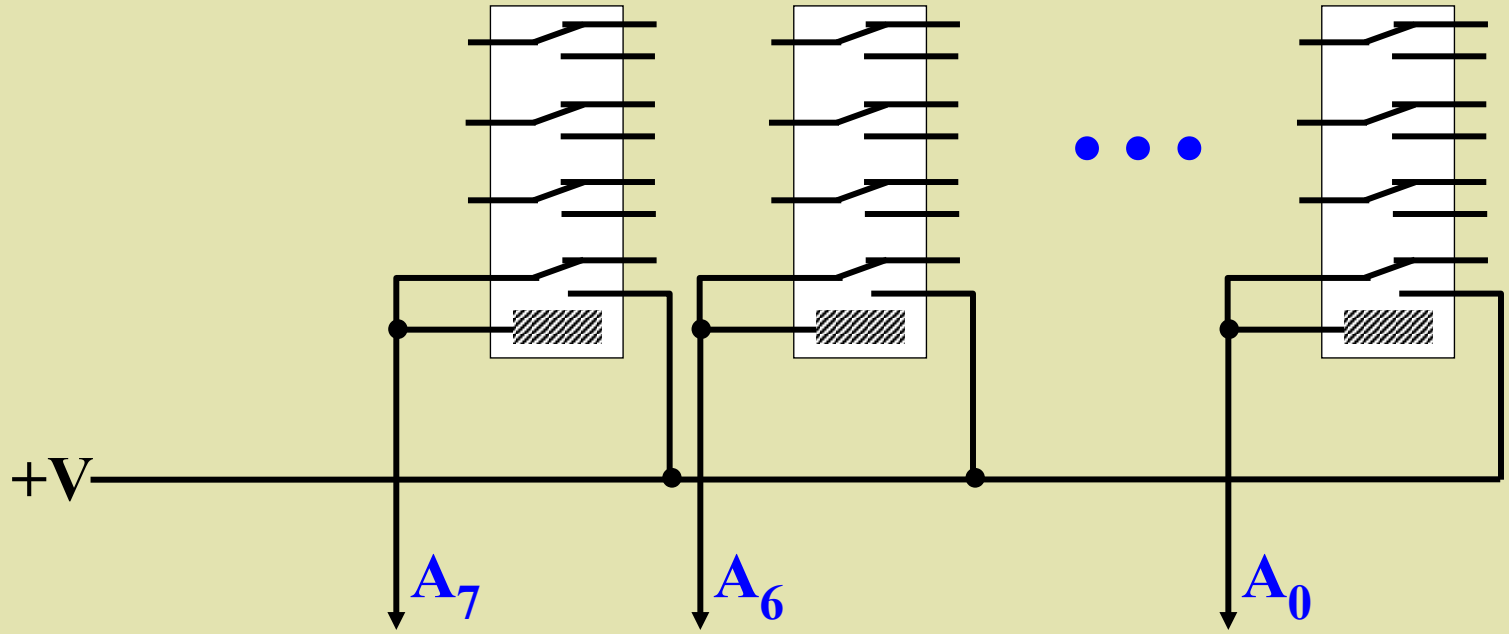
# An 8-Bit ALU



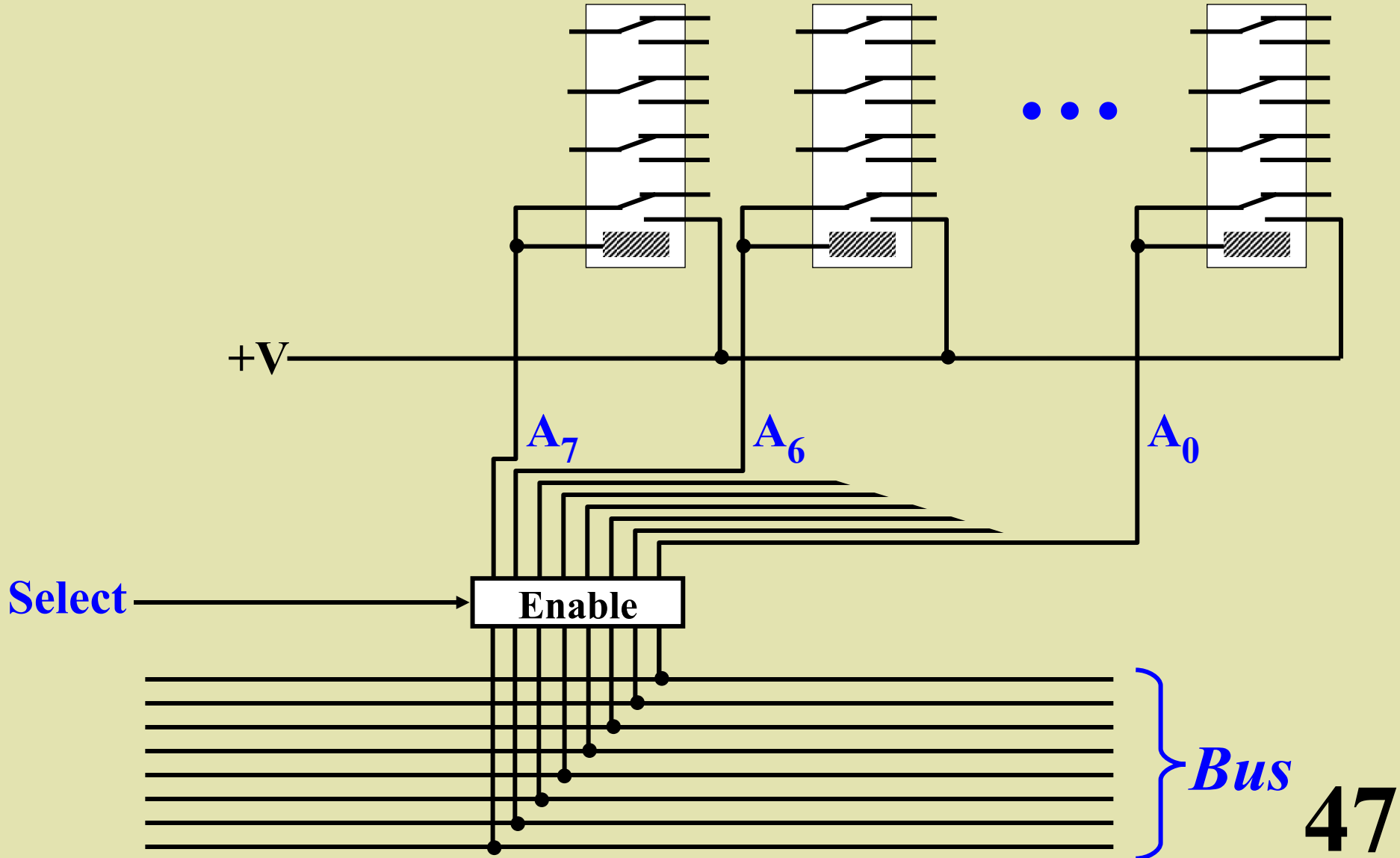
# Register Storage



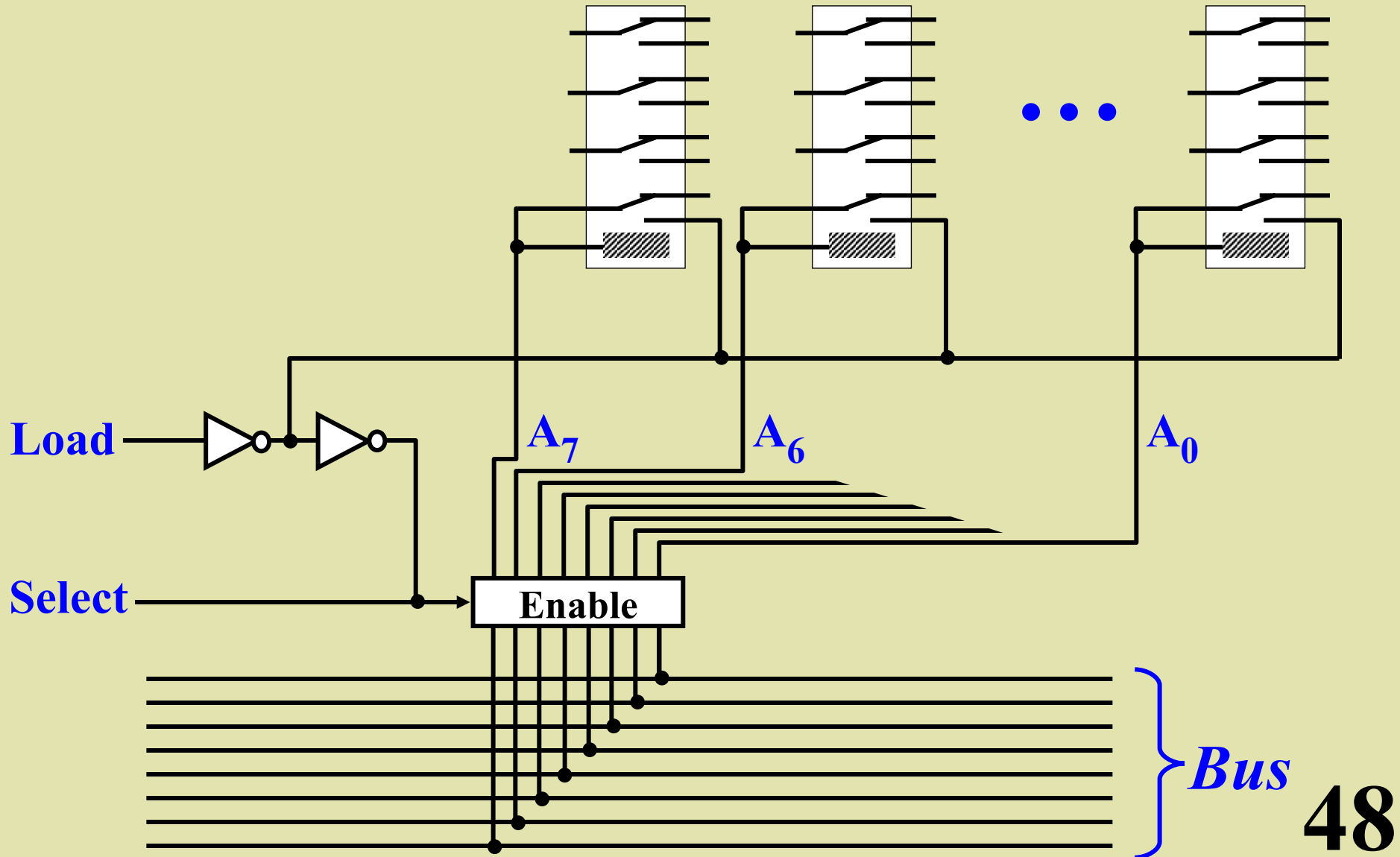
# Register Storage



# Register Storage

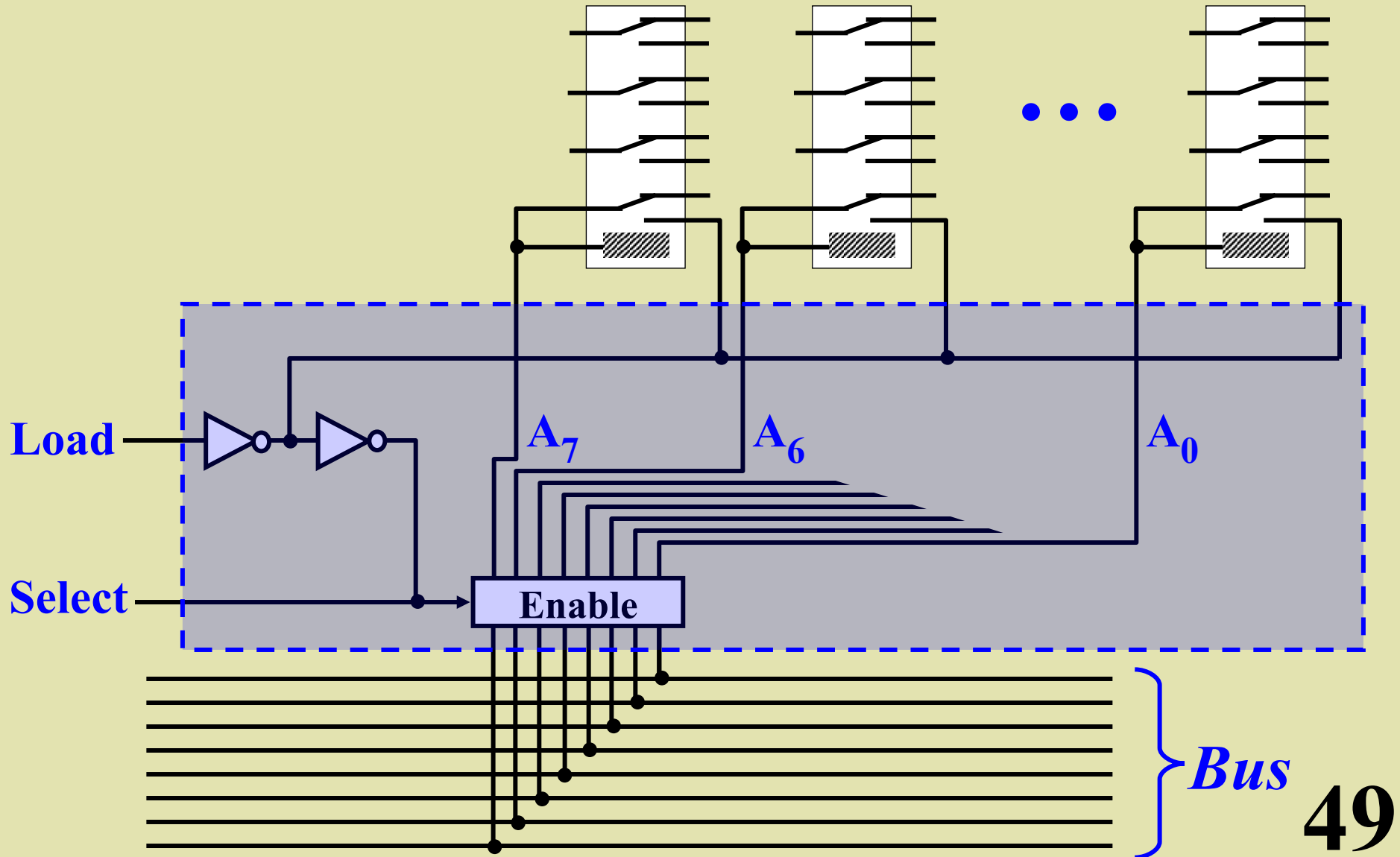


# Register Storage

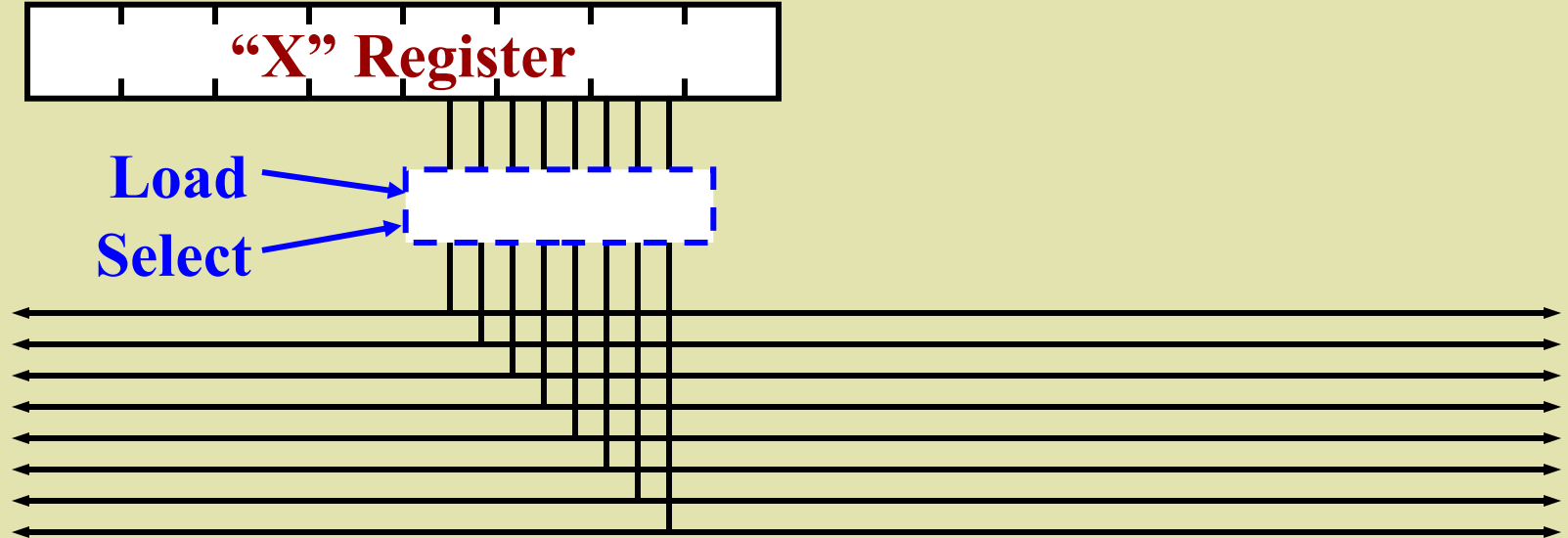




# Register Storage

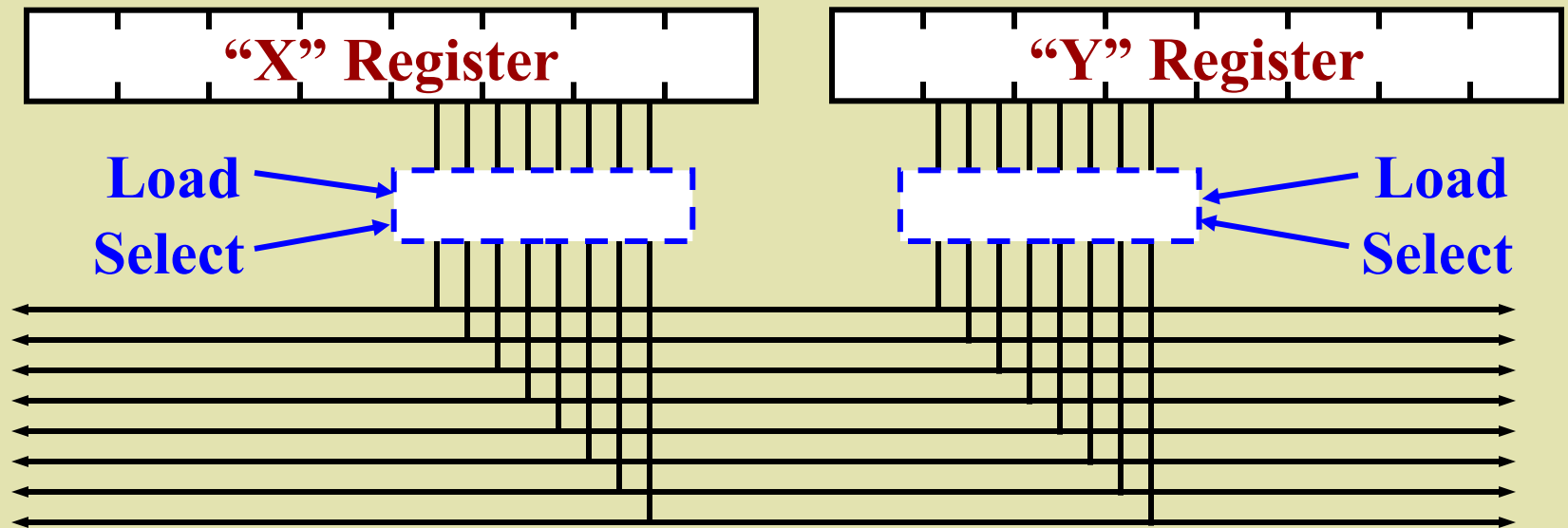


# 8-Bit Registers



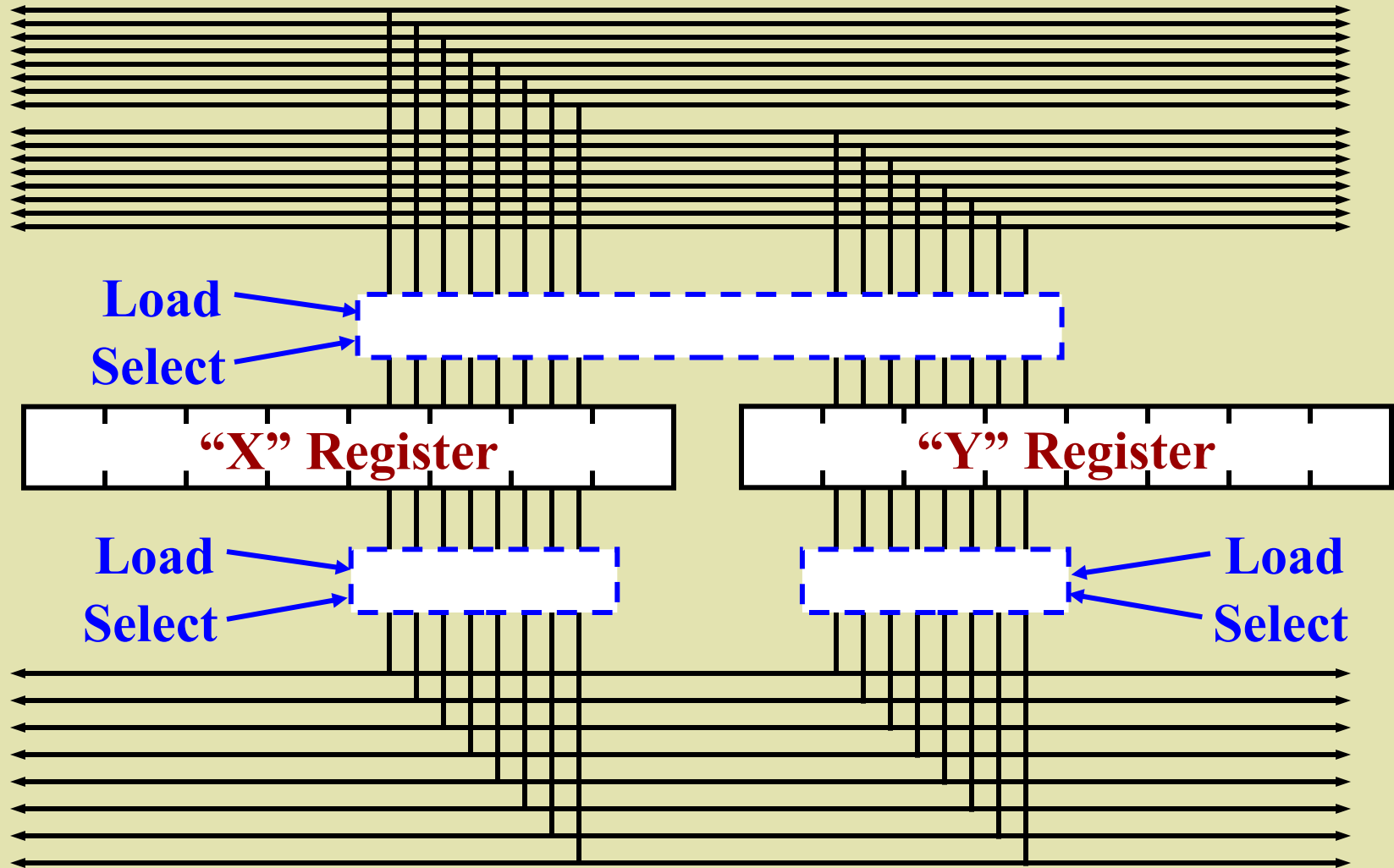
8-Bit "Data" Bus

# 8-Bit Registers



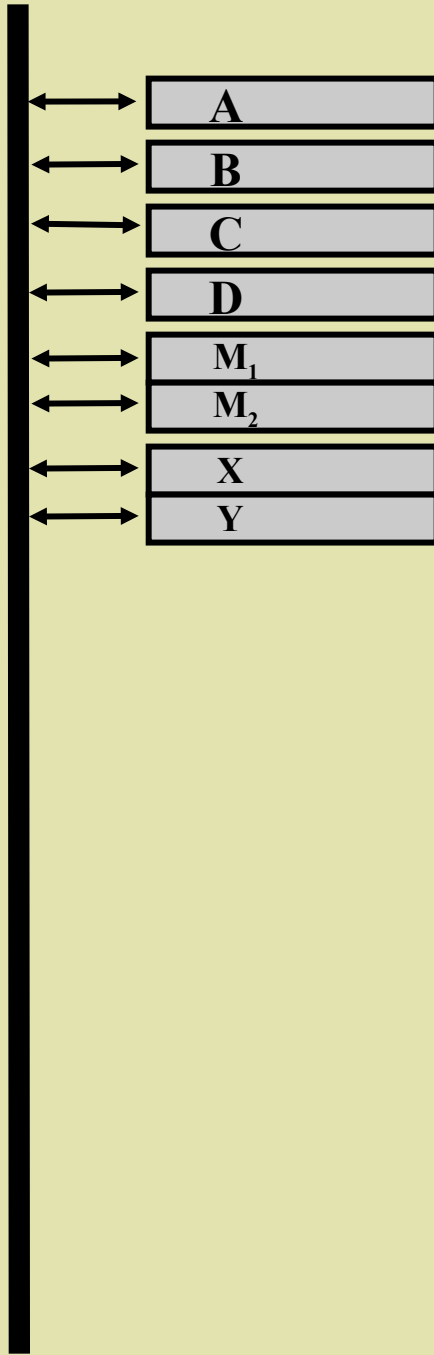
8-Bit "Data" Bus

# 16-Bit “Address” Bus



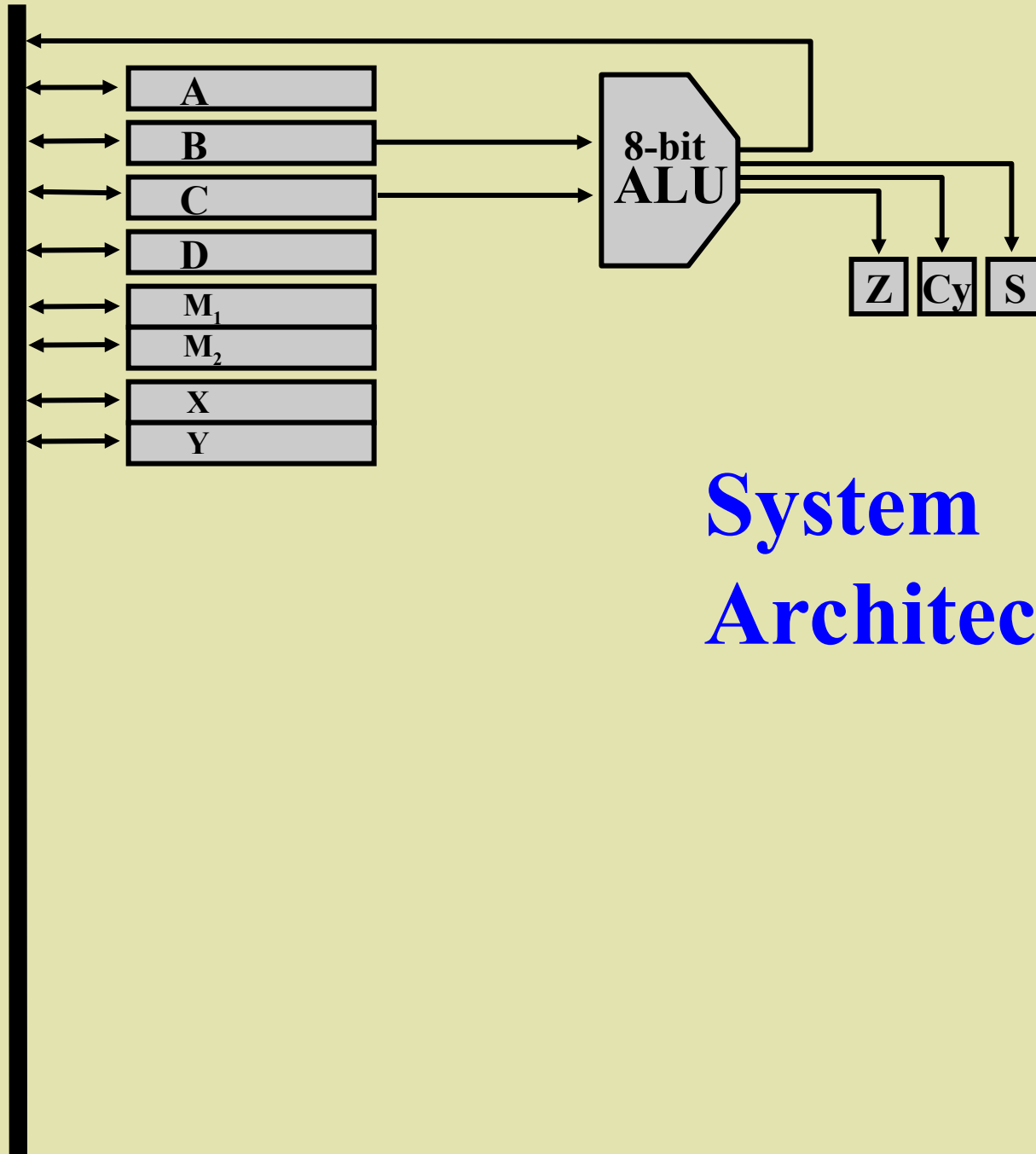
8-Bit “Data” Bus

**8-bit Data Bus**



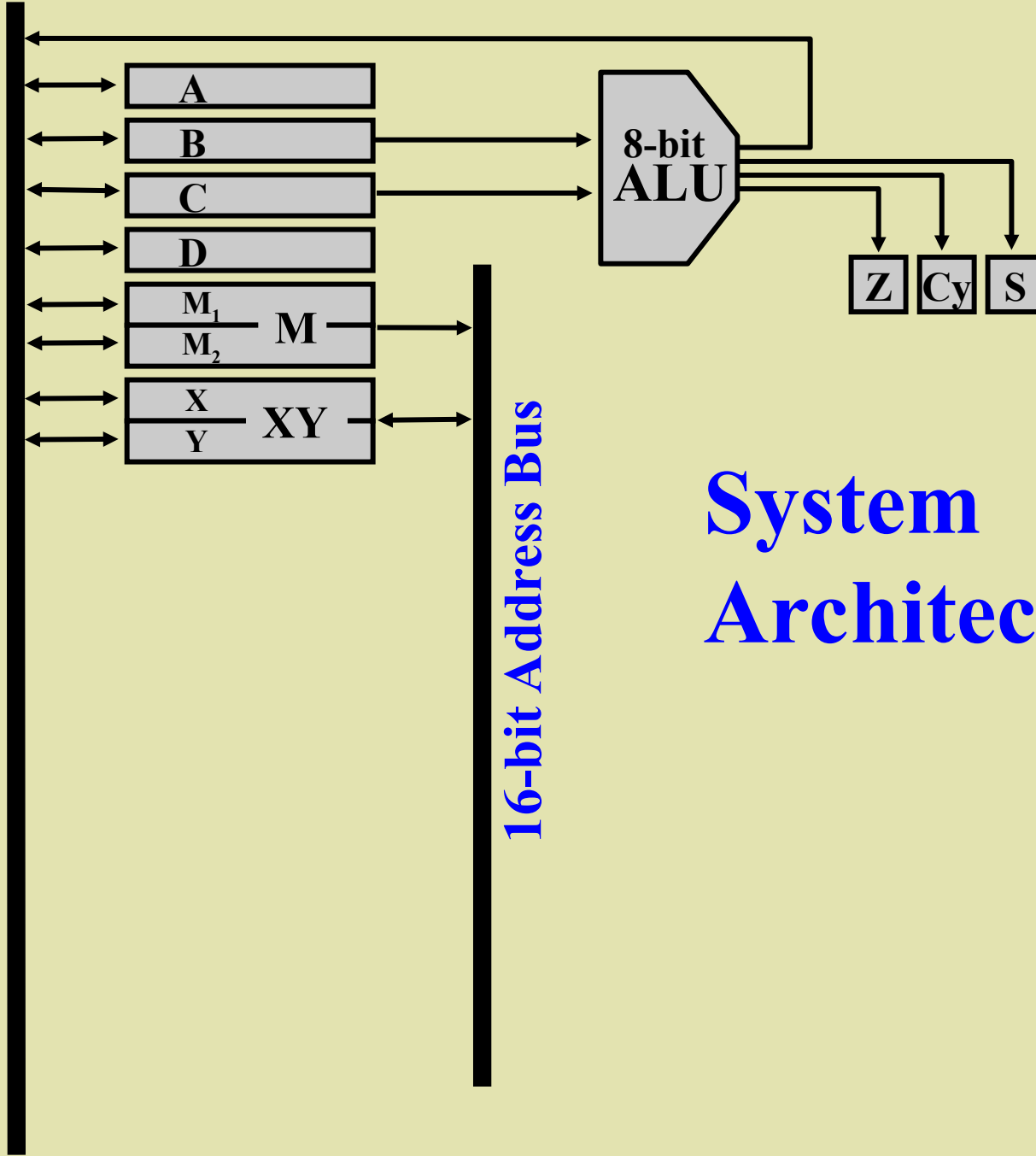
# System Architecture

8-bit Data Bus



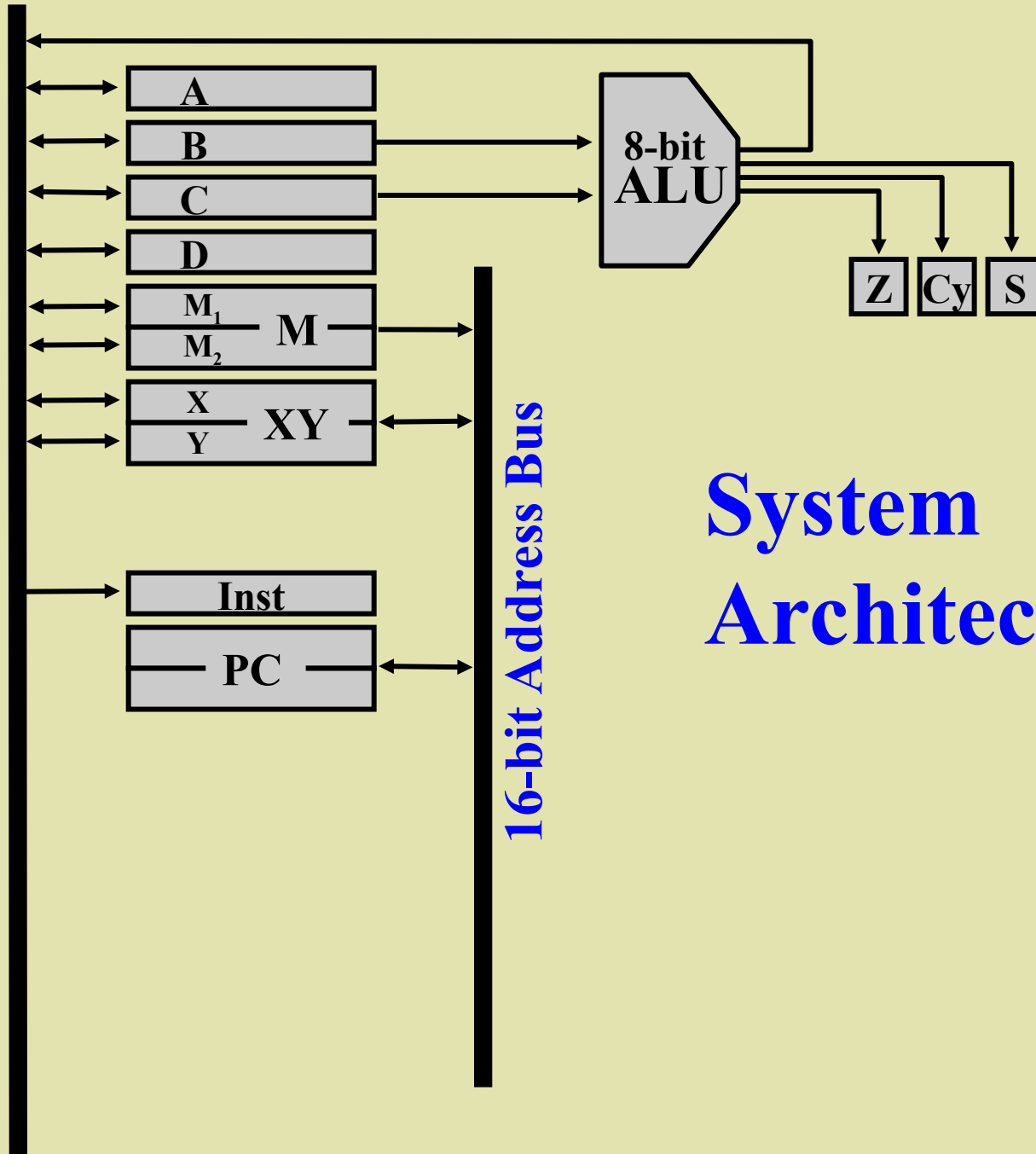
# System Architecture

8-bit Data Bus



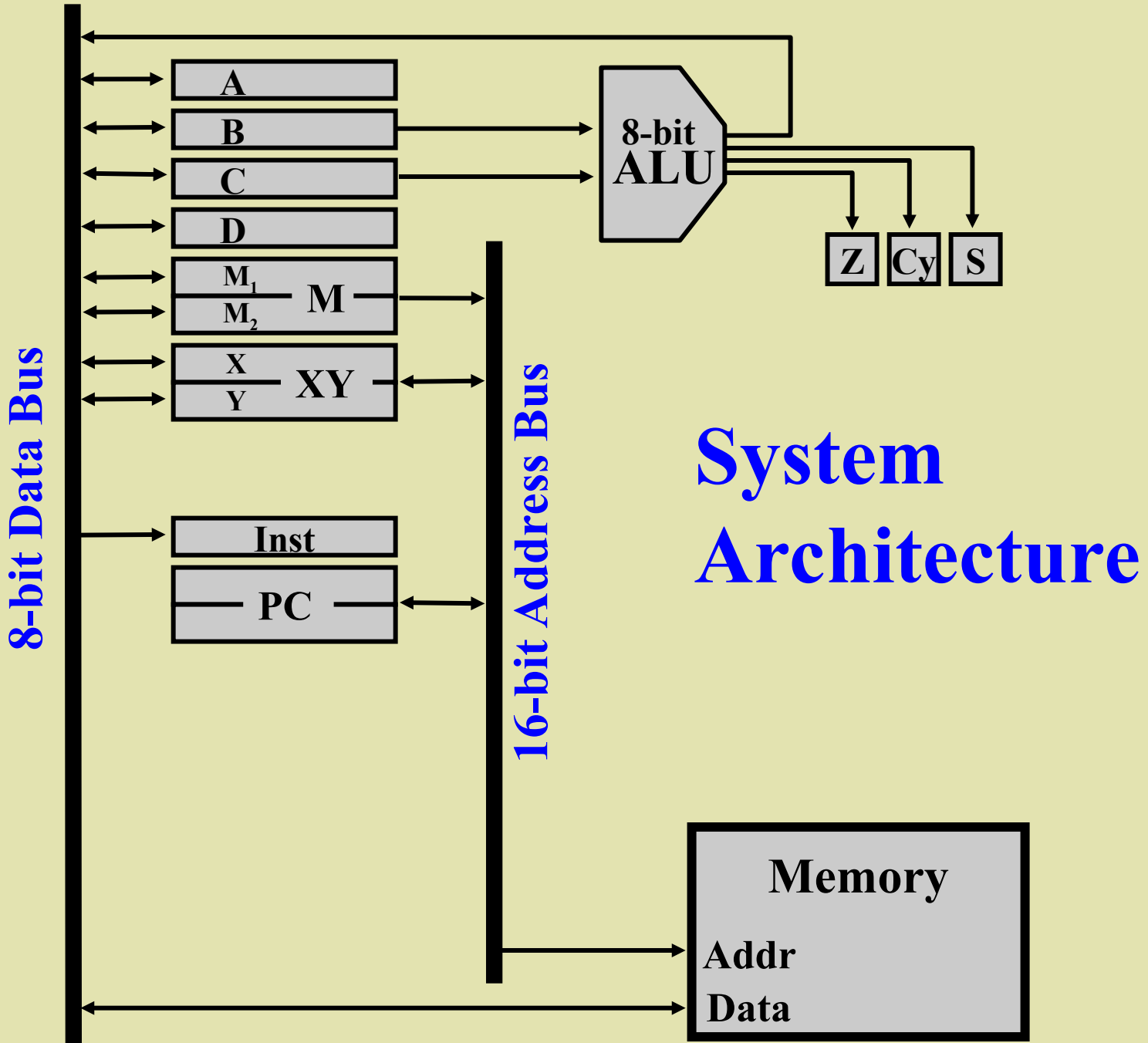
# System Architecture

**8-bit Data Bus**

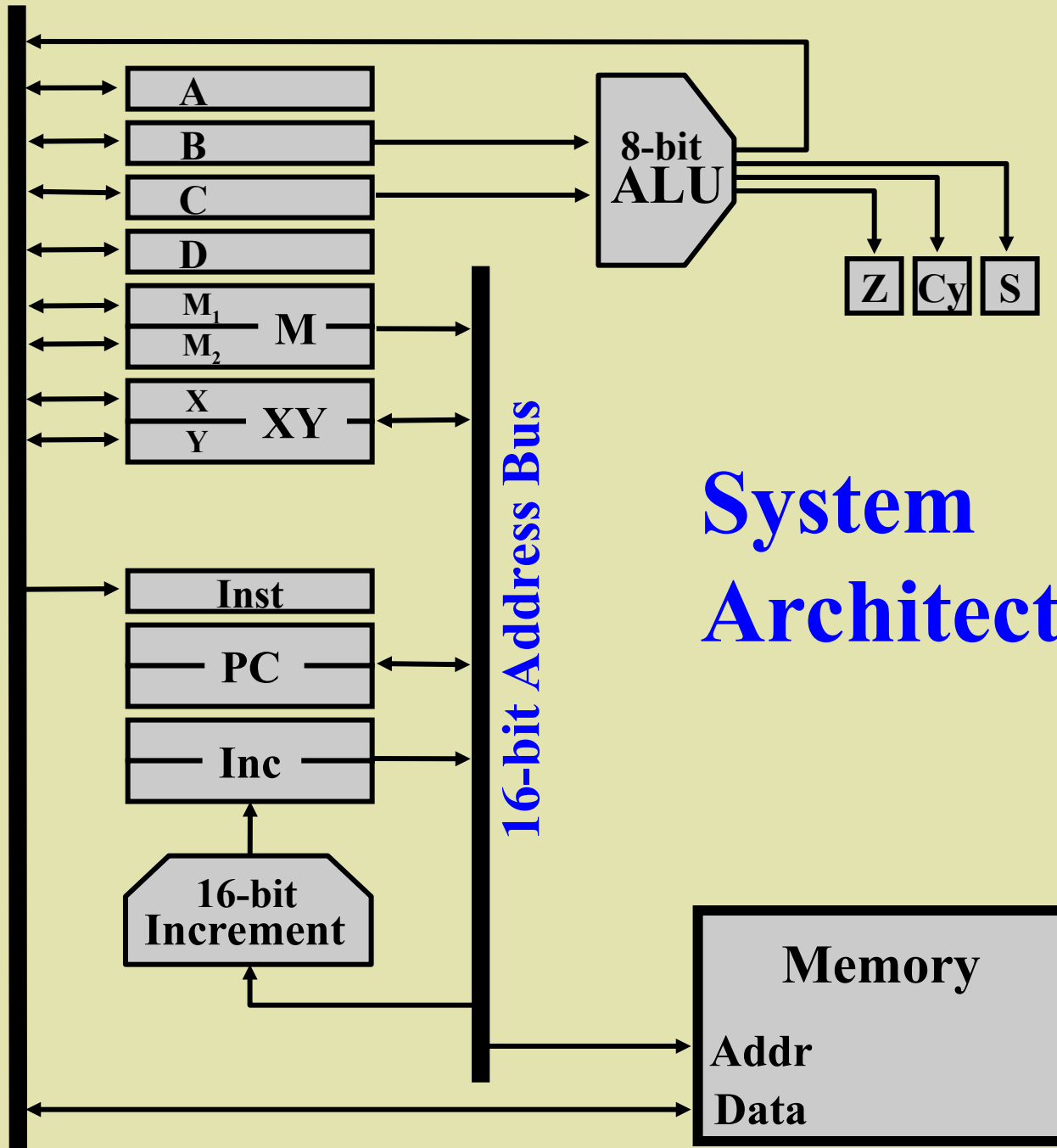


# System Architecture

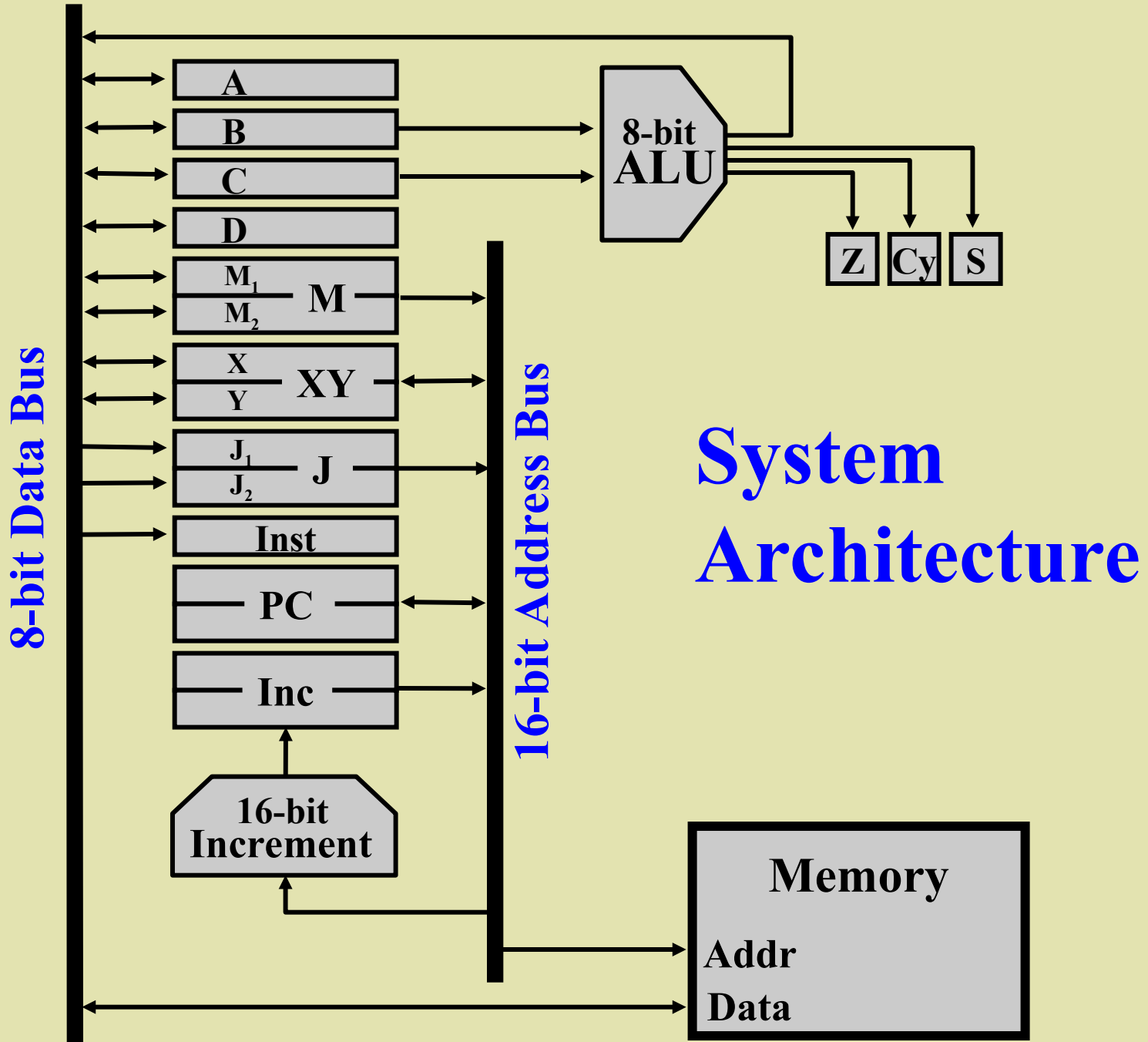




8-bit Data Bus



# System Architecture



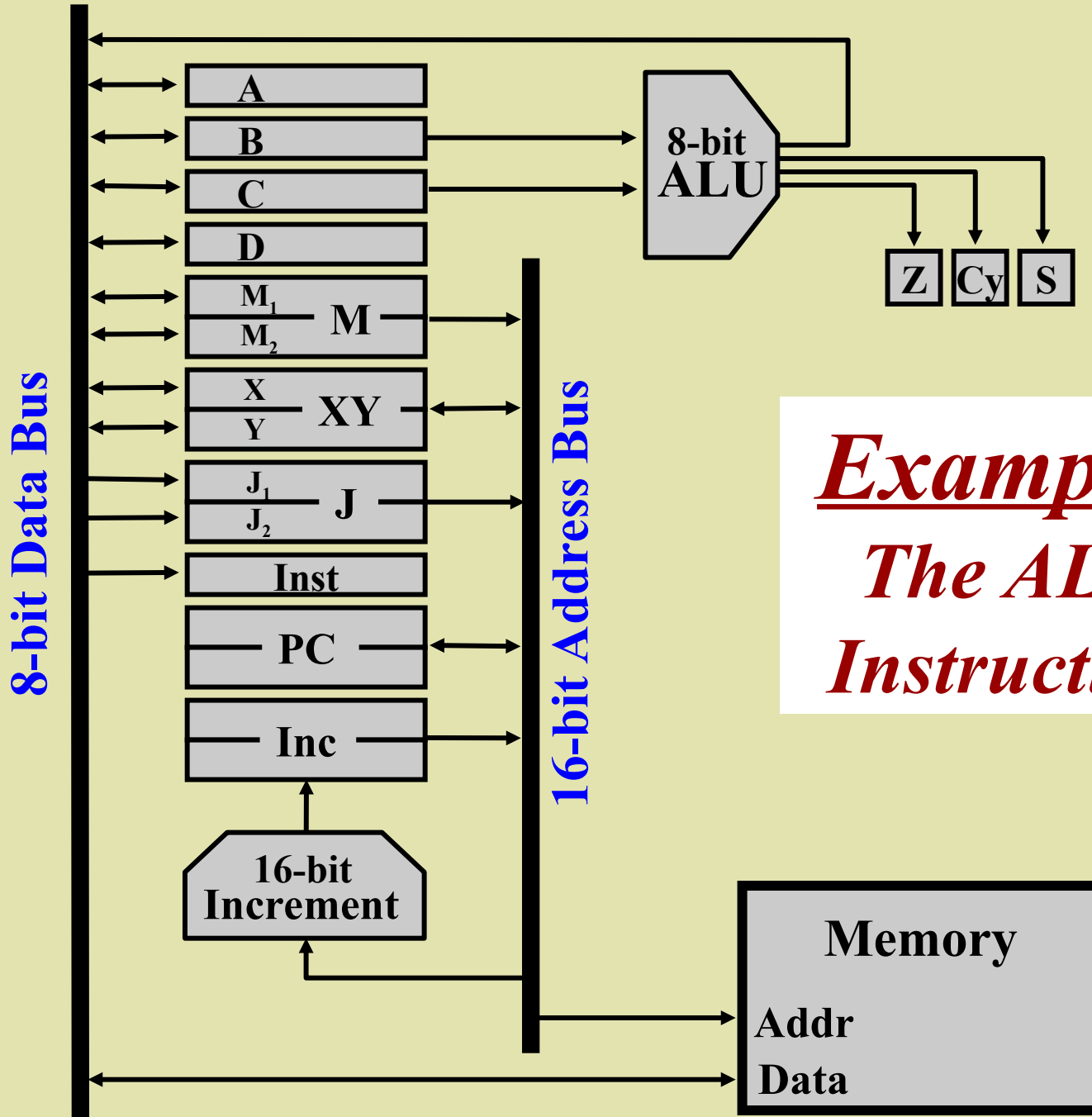
# System Architecture

**32K Byte Static  
RAM Chip**

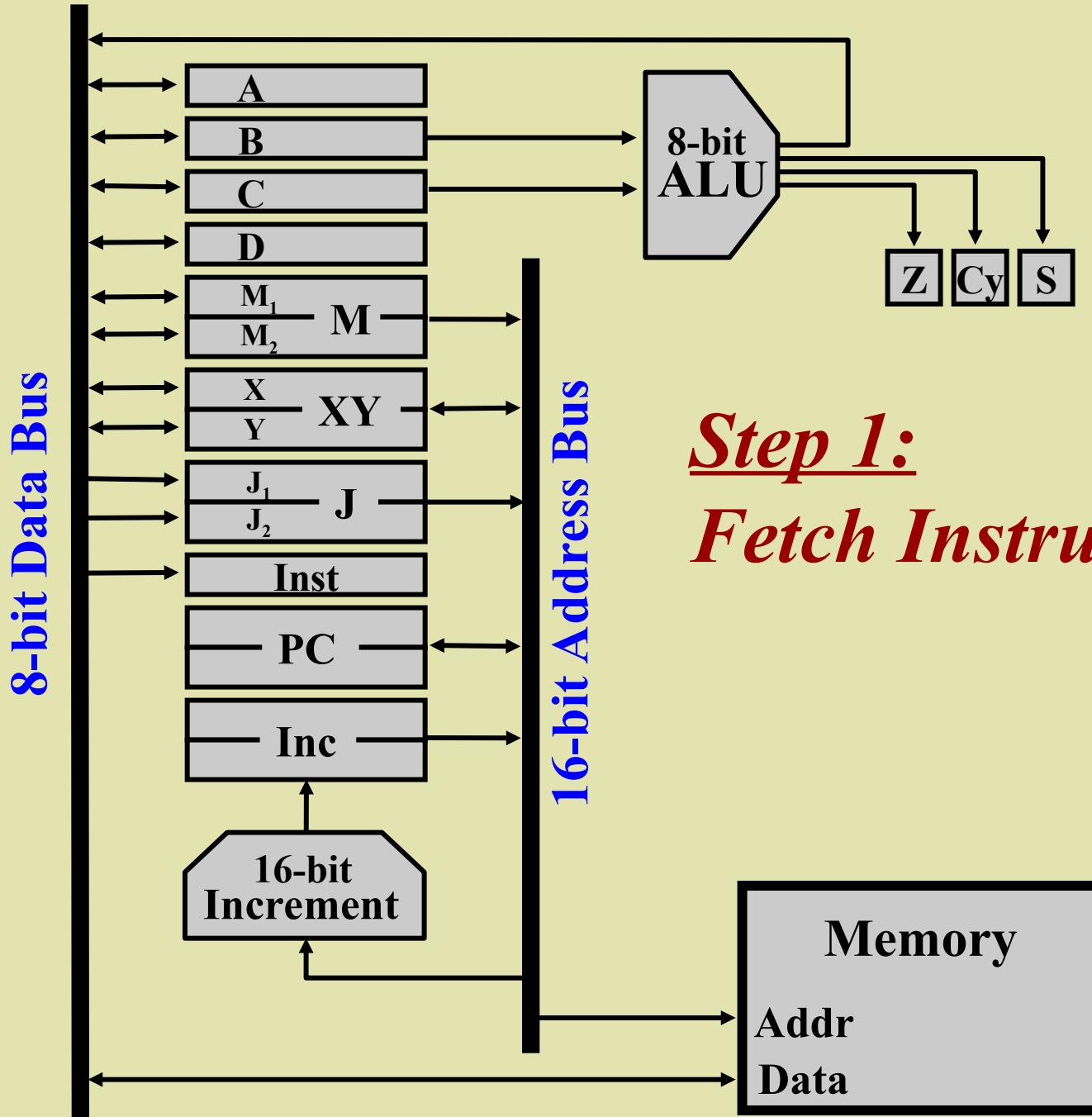
**LEDs**

**8 FET Power Transistors**  
*(to drive relays during  
a memory-read operation)*

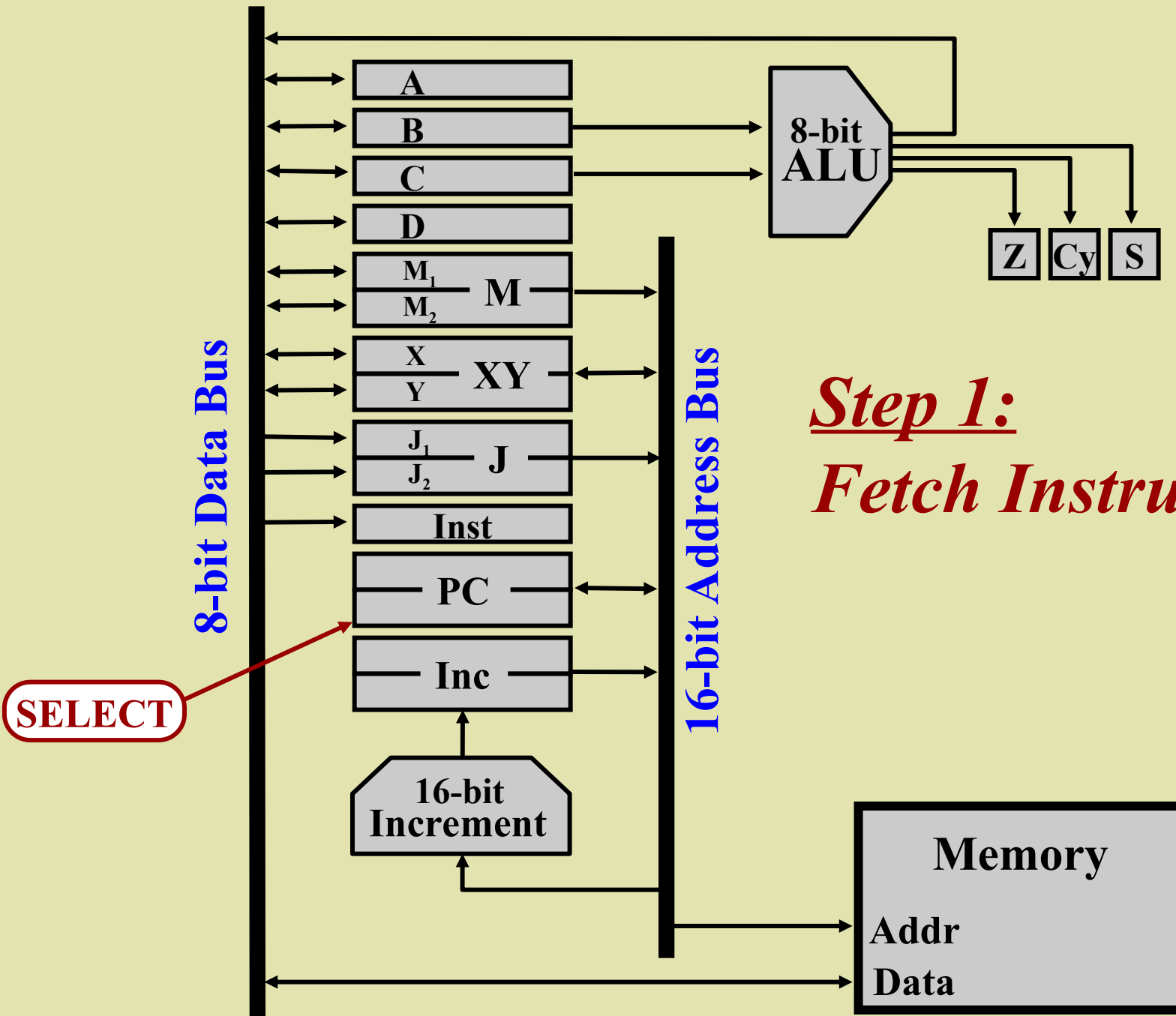
QuickTime™ and a  
TIFF (LZW) decompressor  
are needed to see this picture.



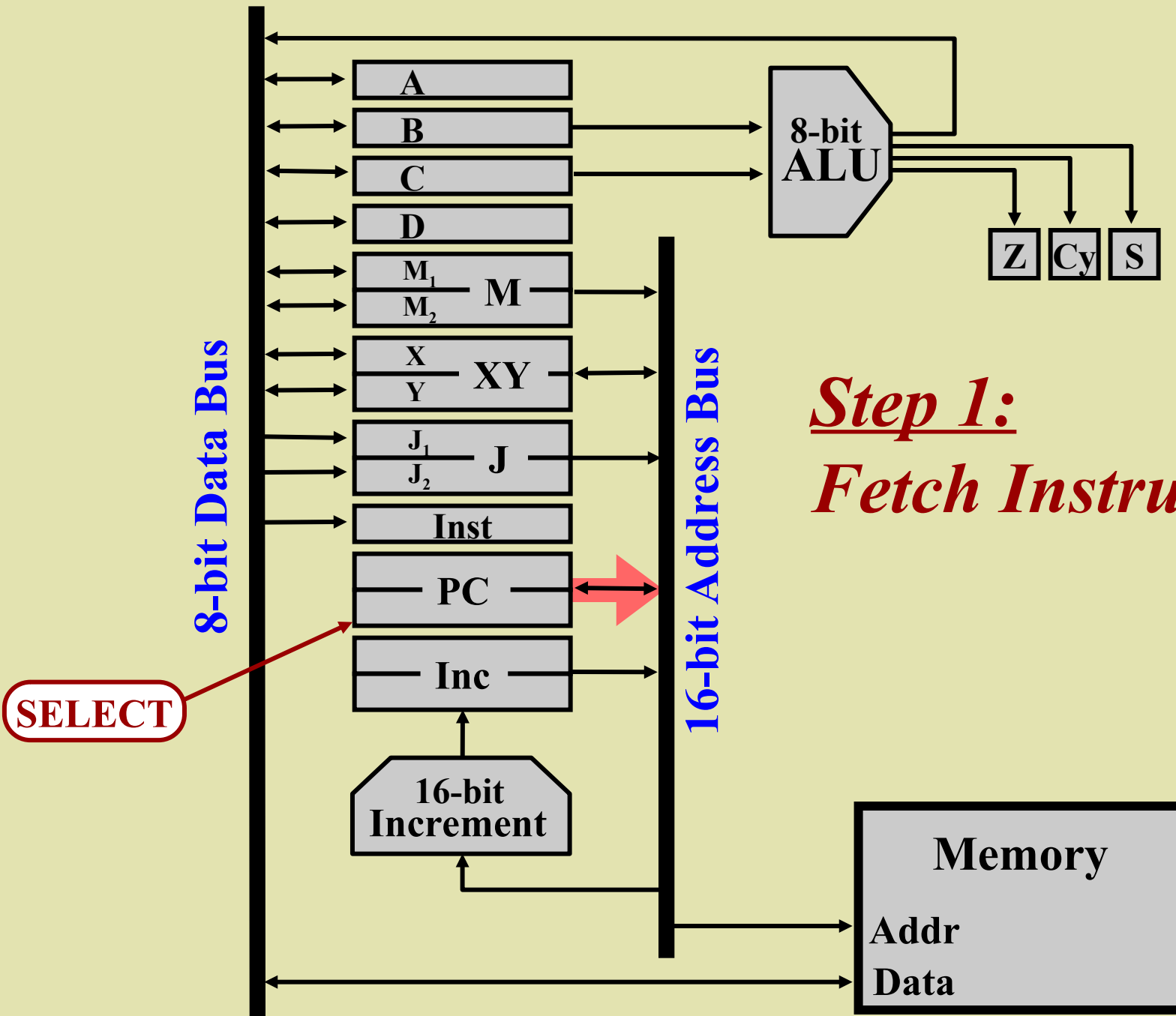
*Example:*  
*The ALU*  
*Instruction*



***Step 1:***  
***Fetch Instruction***

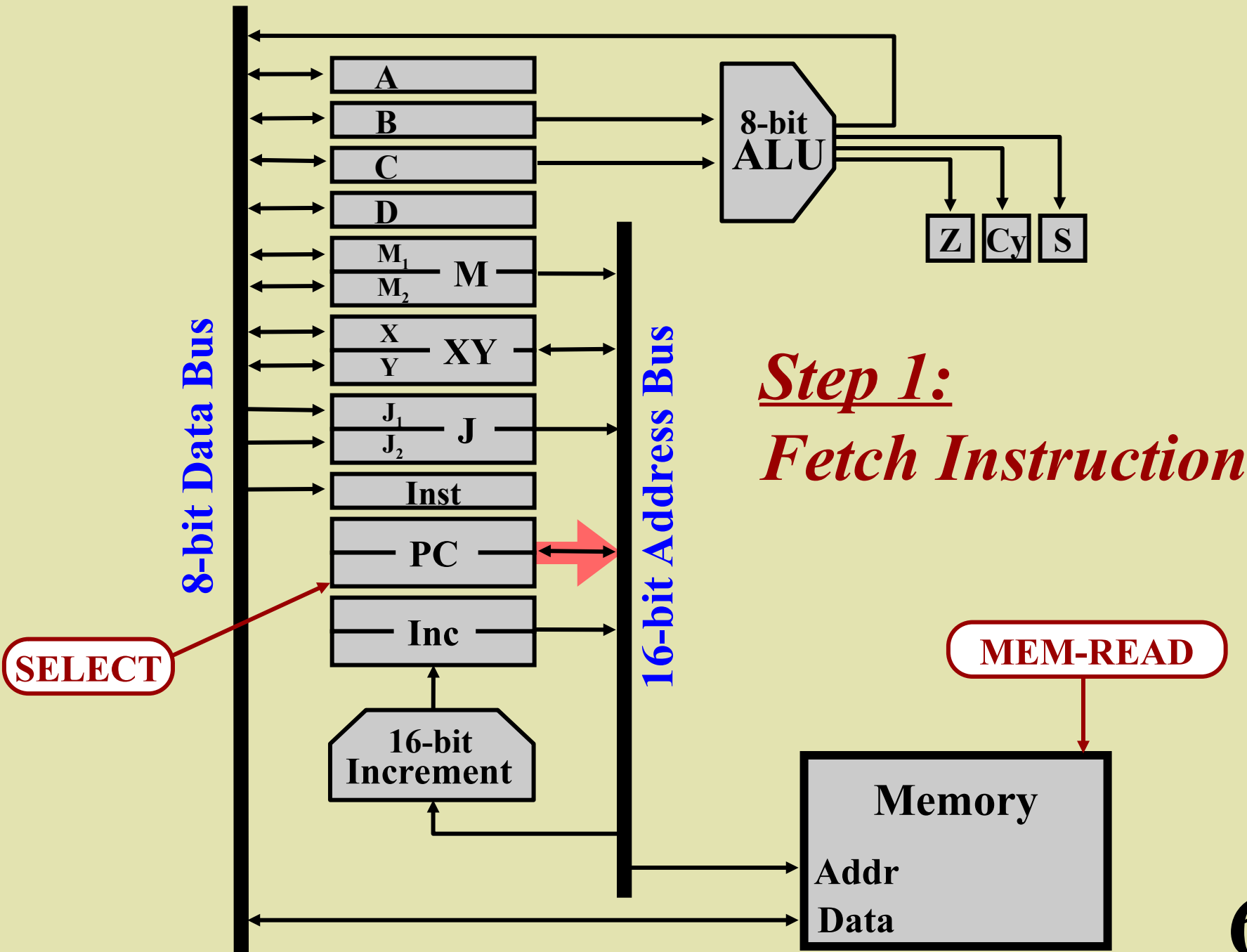


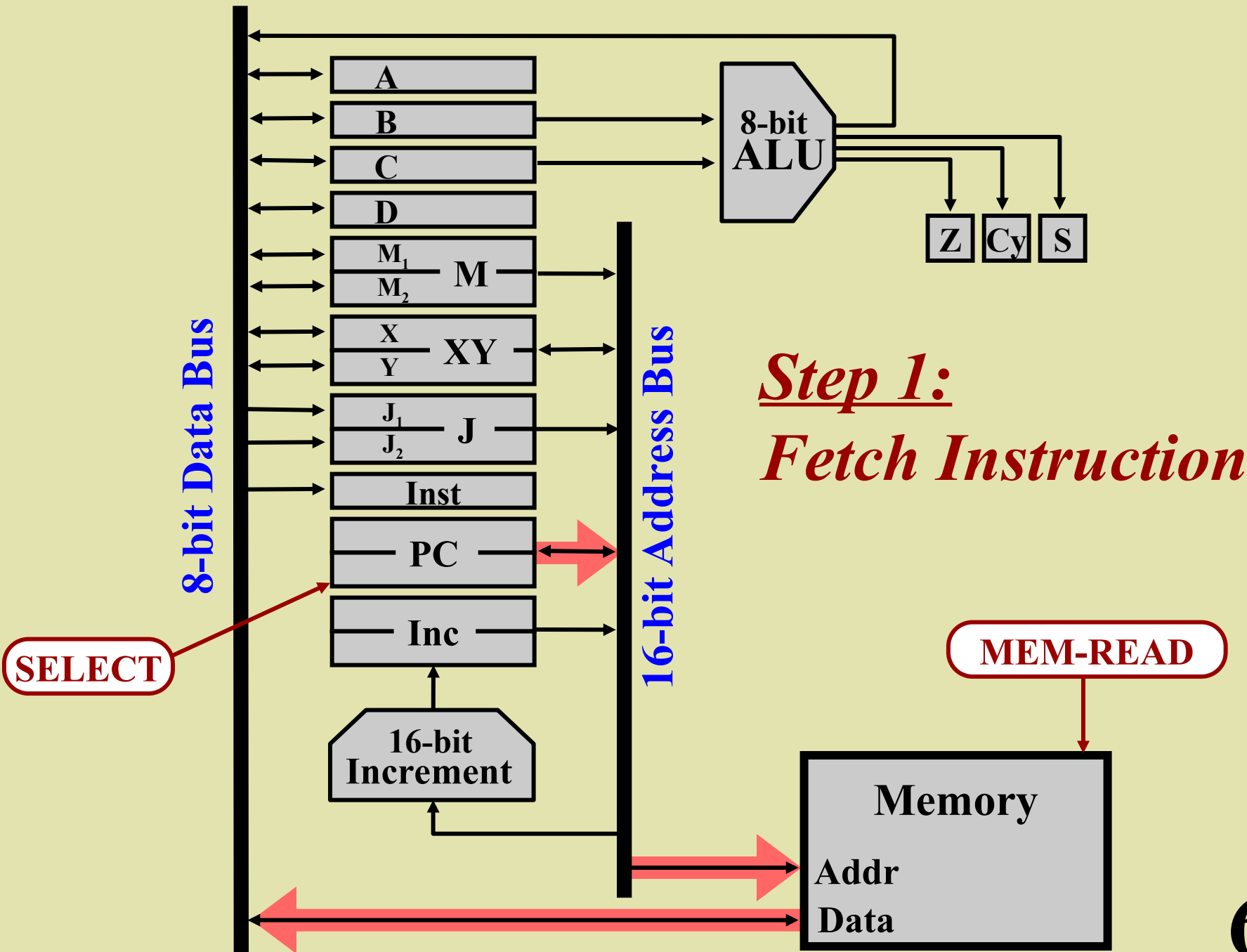
*Step 1:*  
*Fetch Instruction*

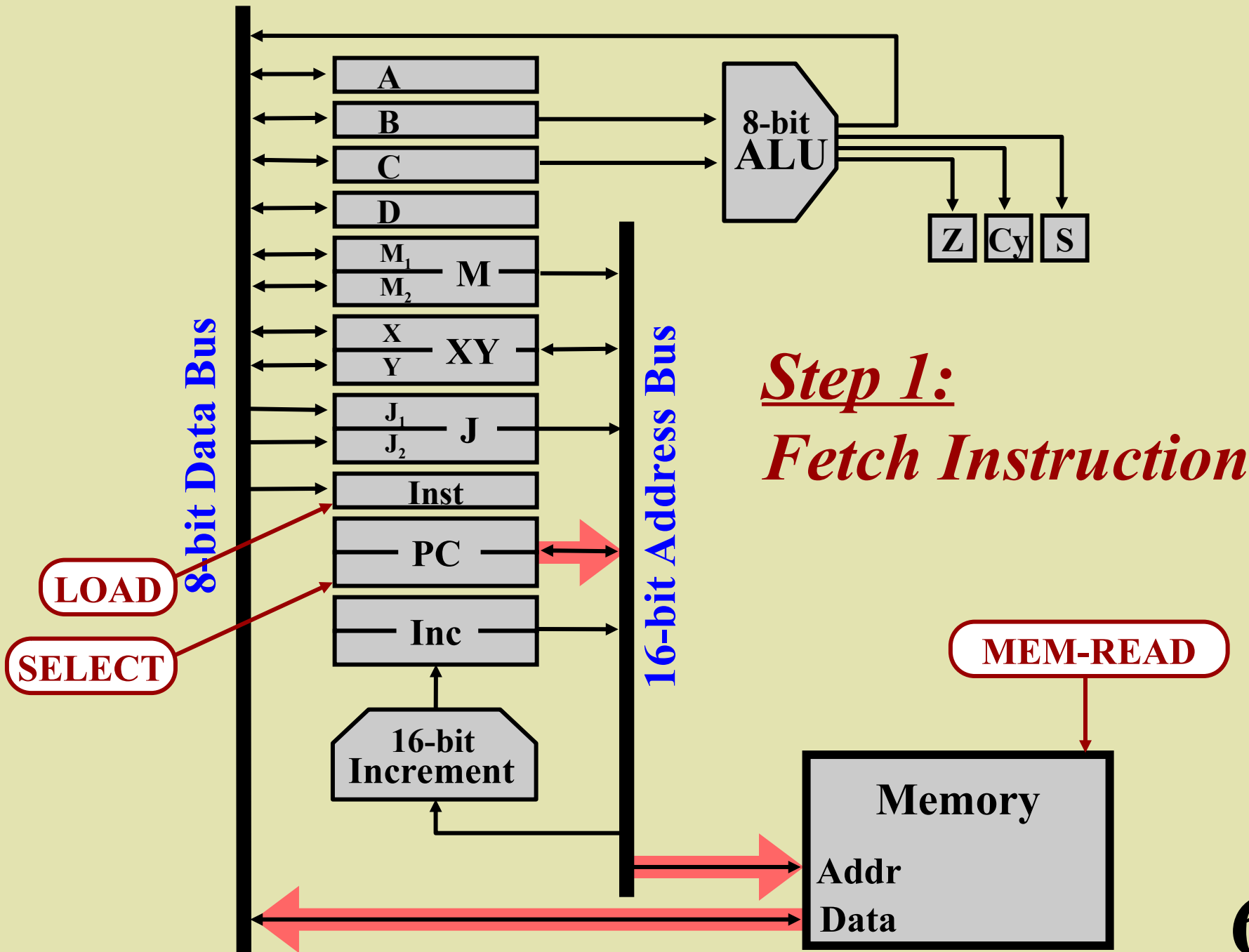


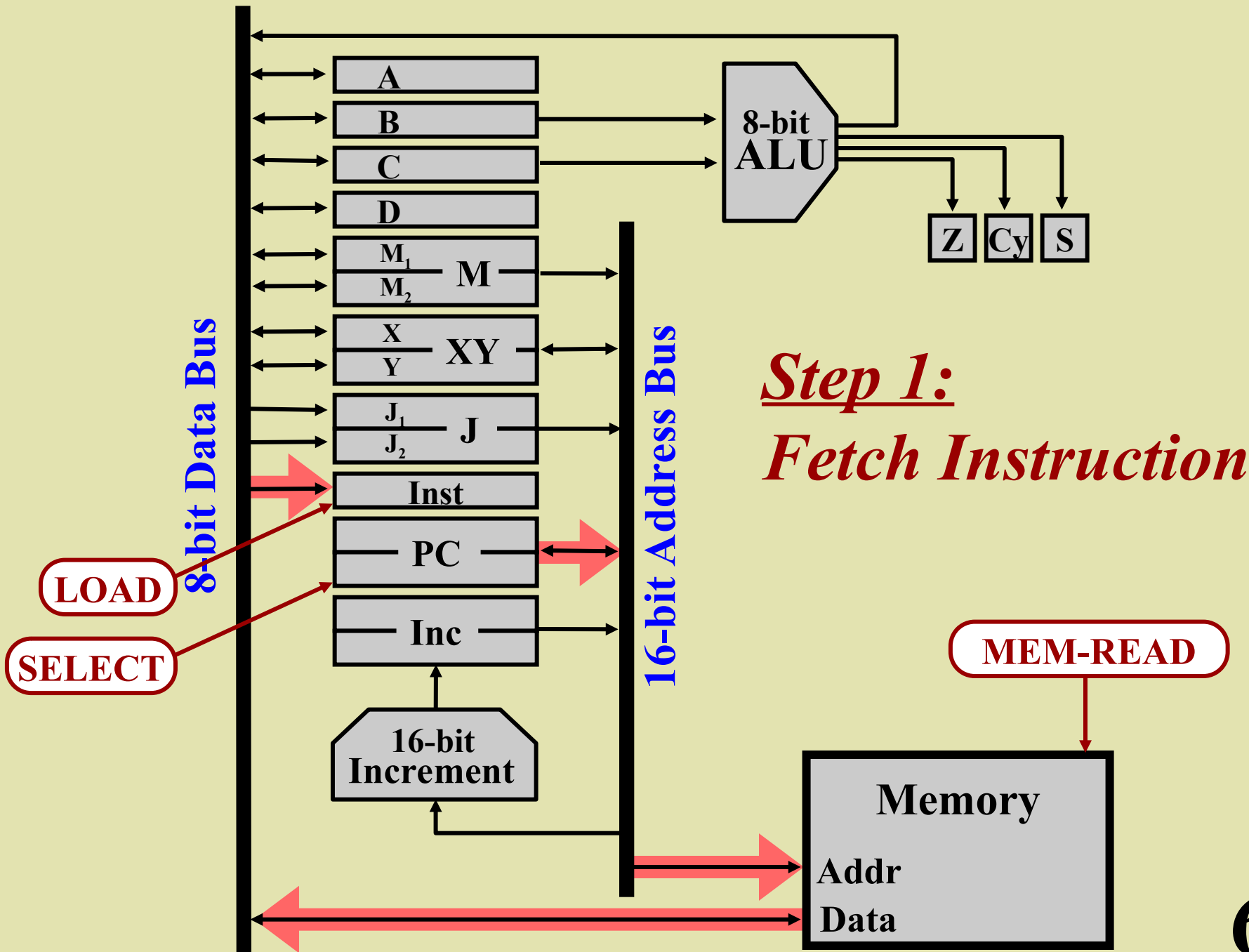
***Step 1:***  
***Fetch Instruction***

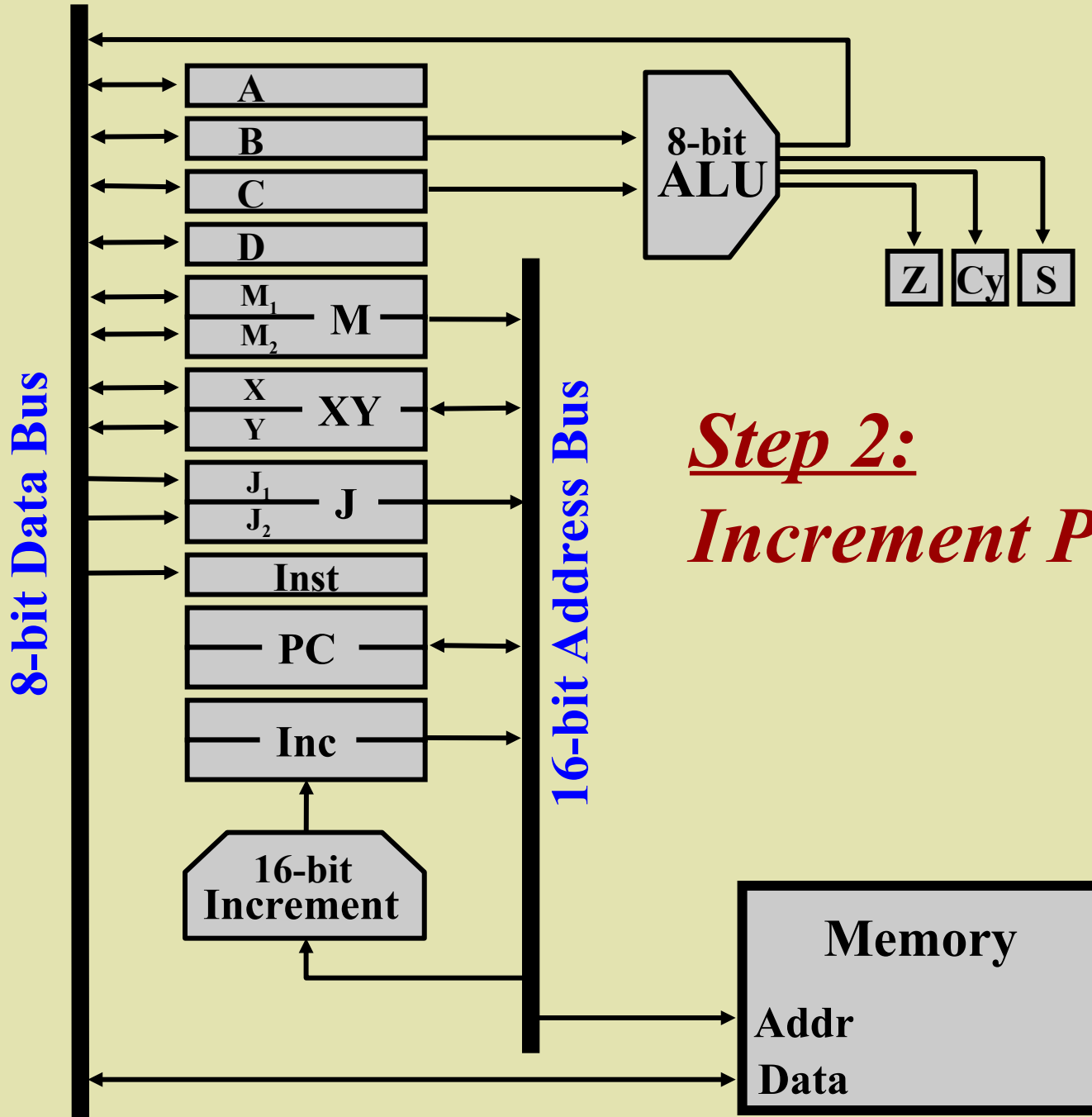




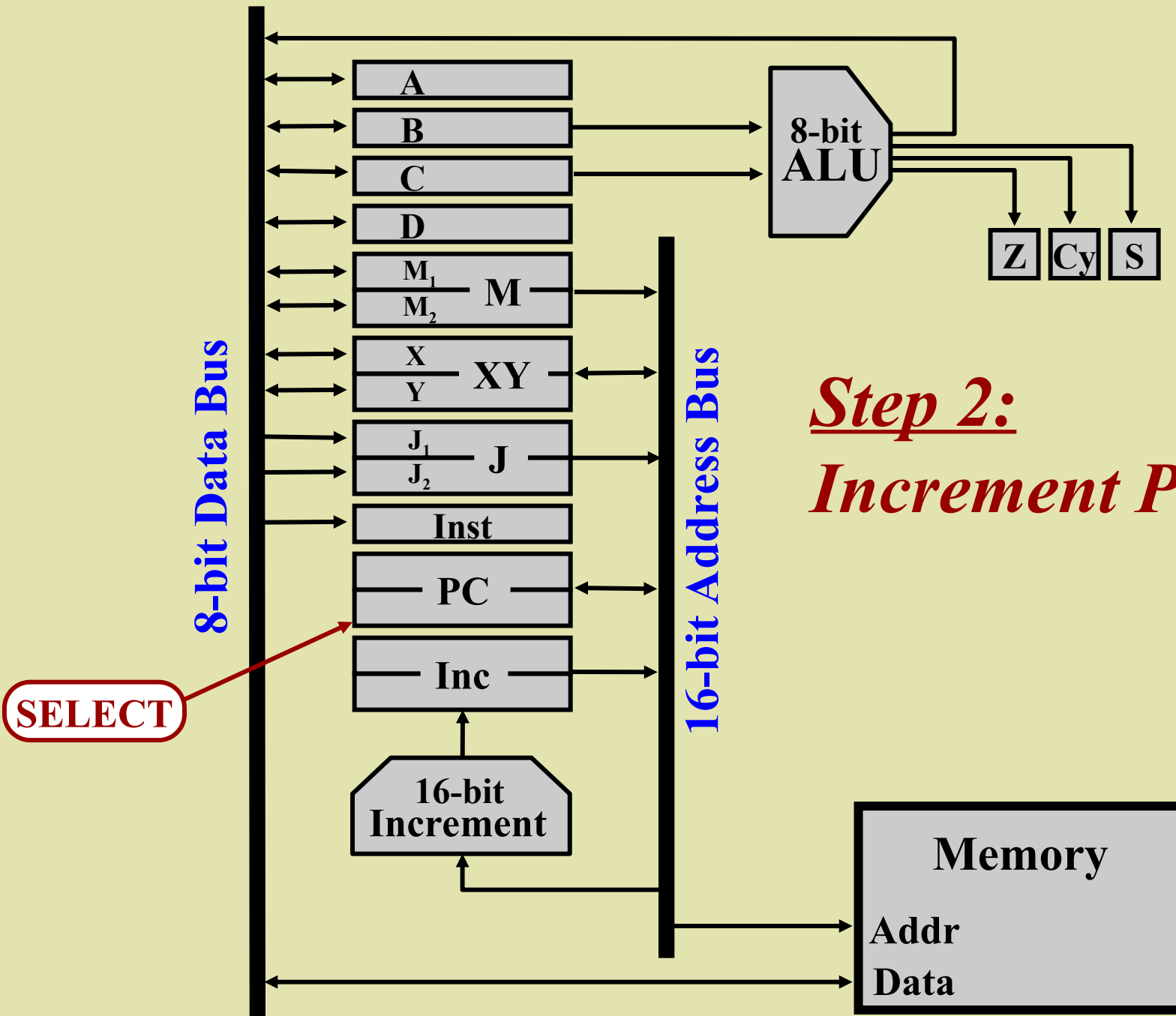




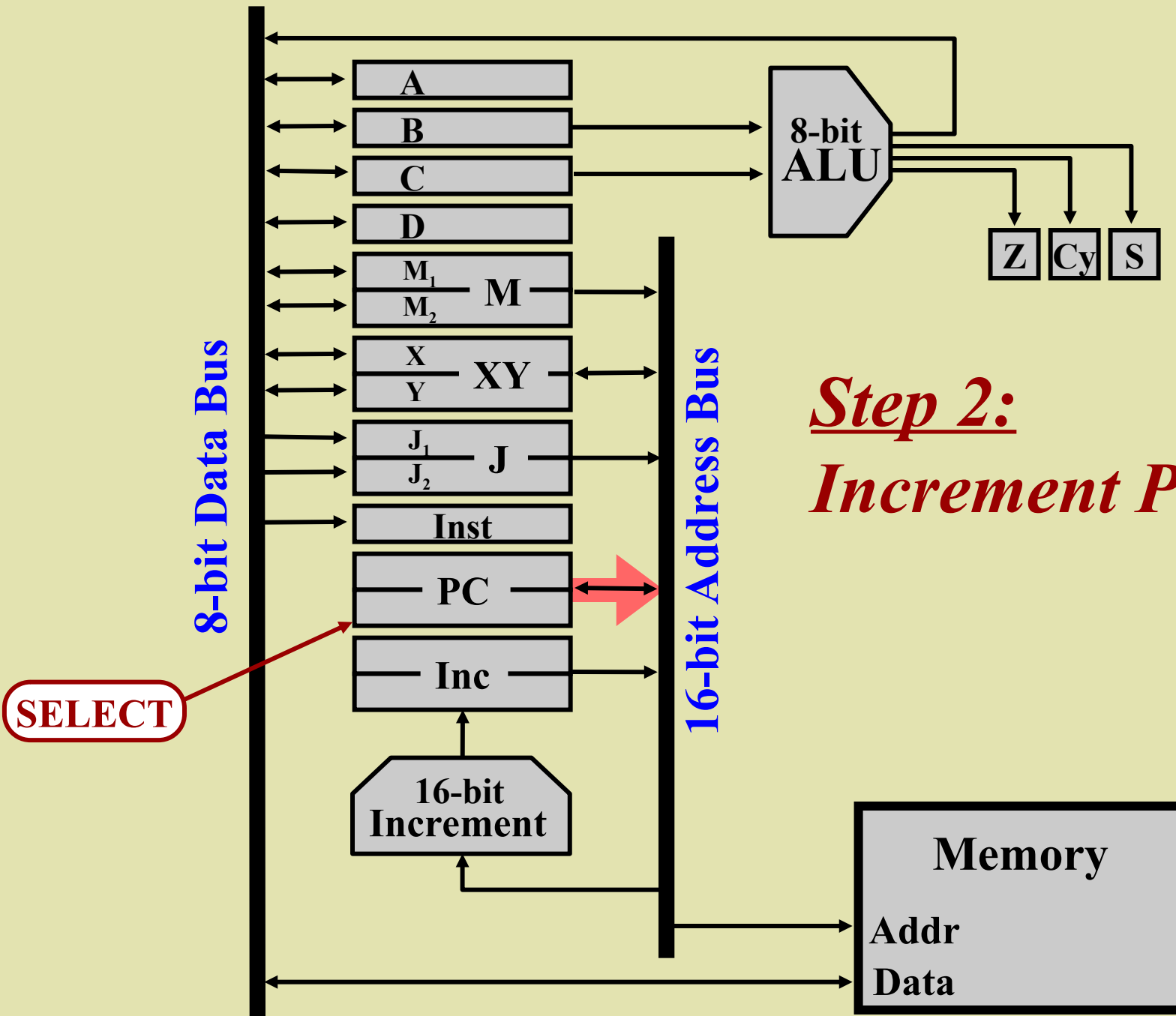




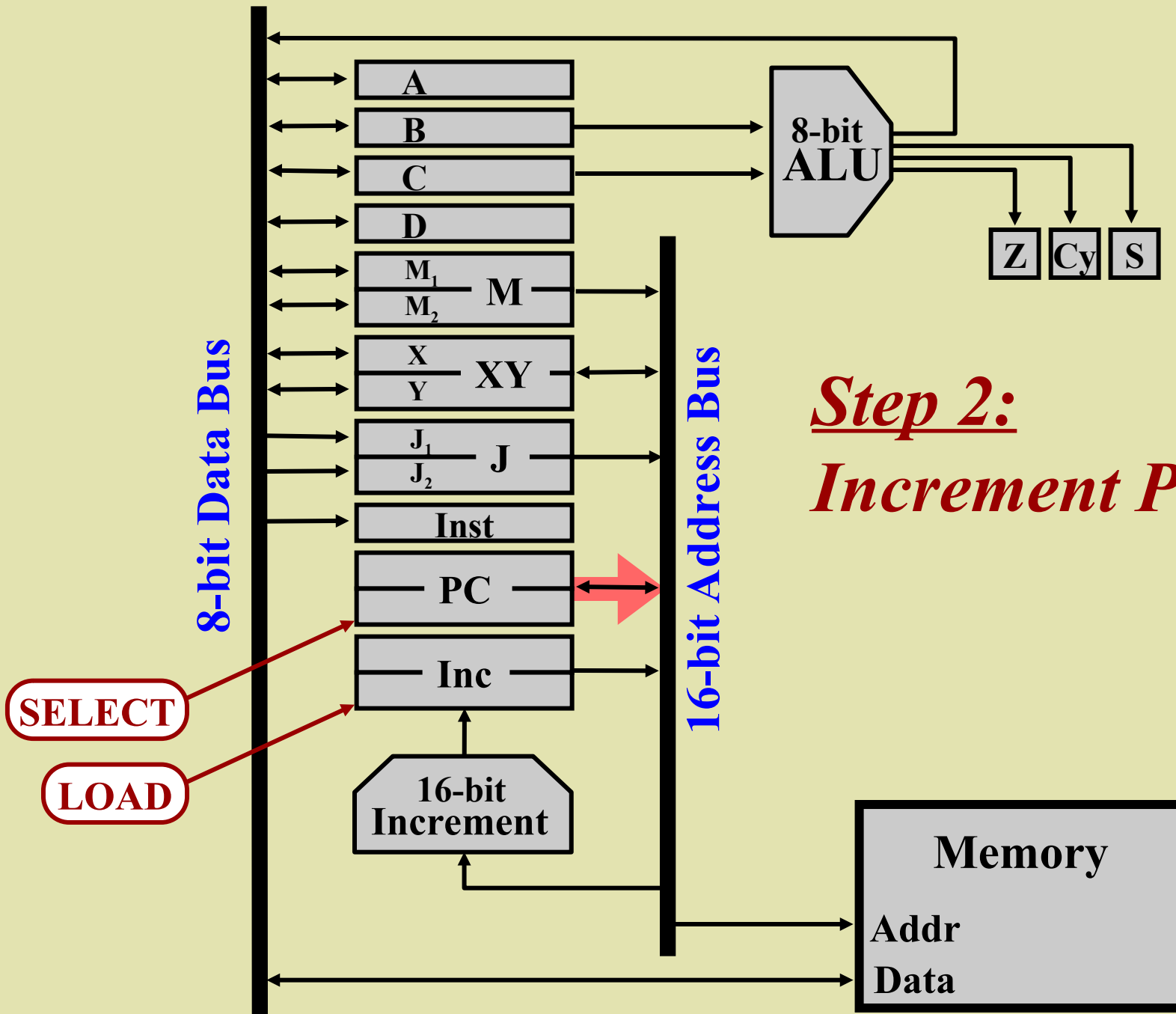
***Step 2:  
Increment PC***



*Step 2:  
Increment PC*

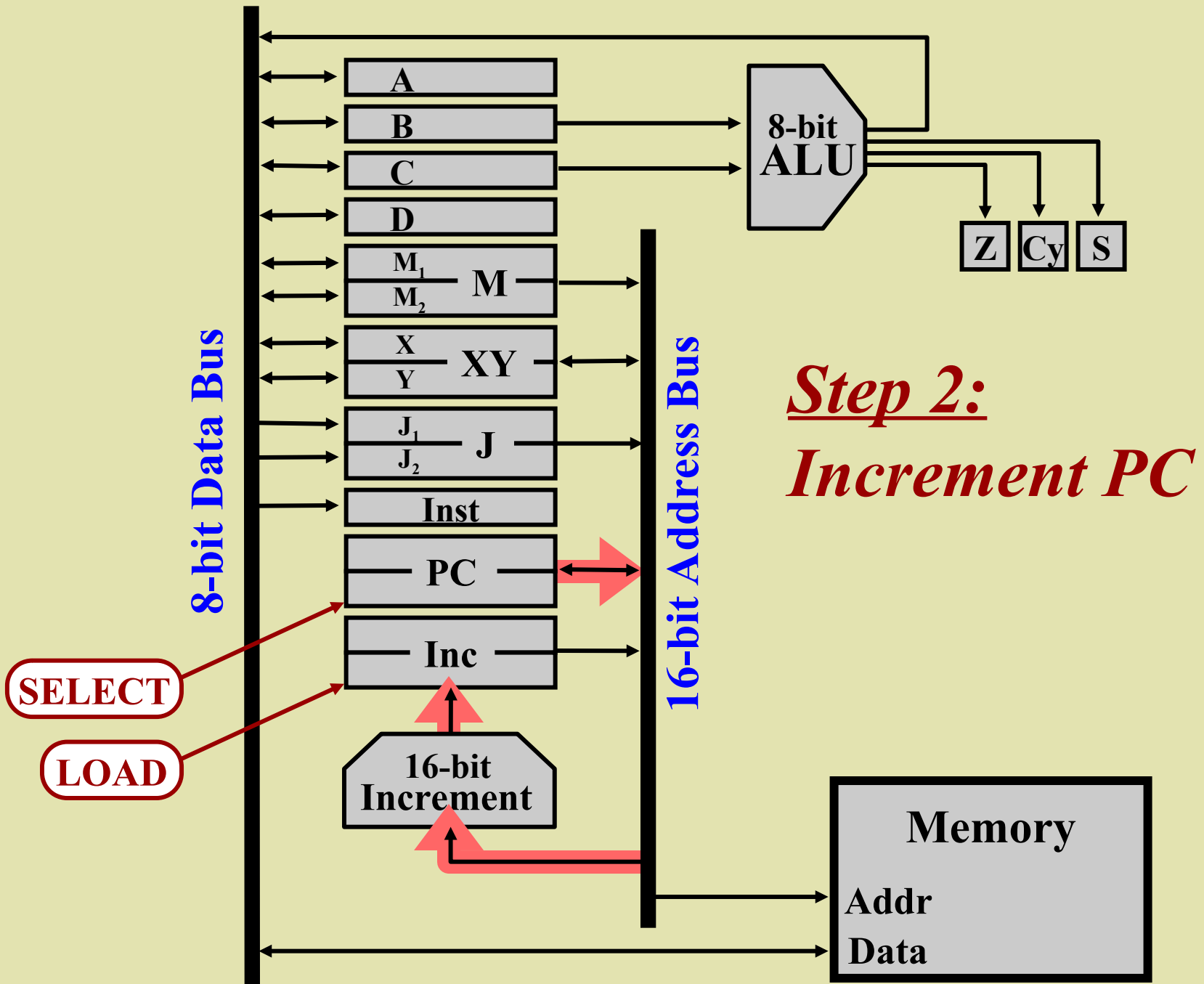


*Step 2:  
Increment PC*

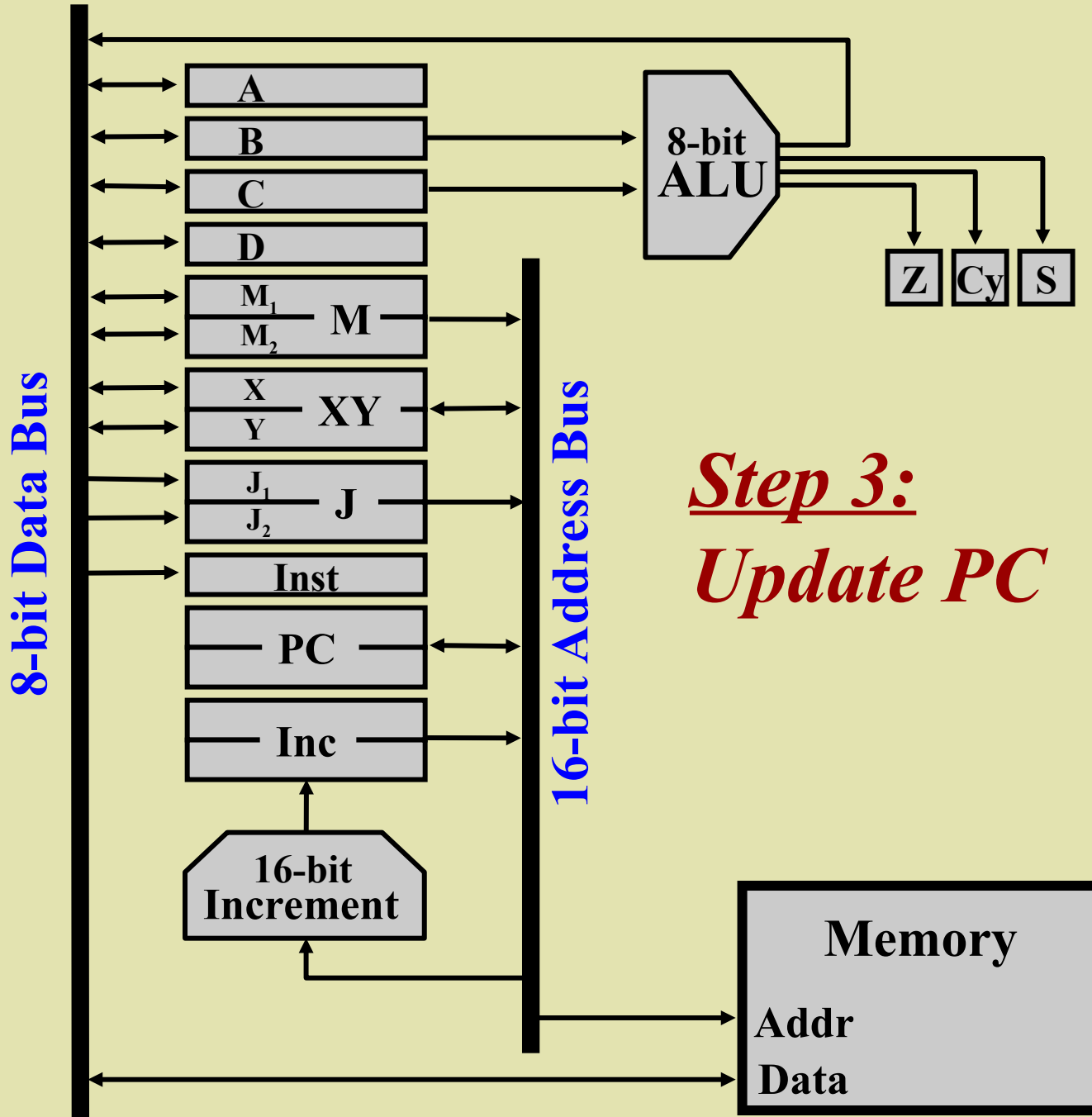


***Step 2:  
Increment PC***

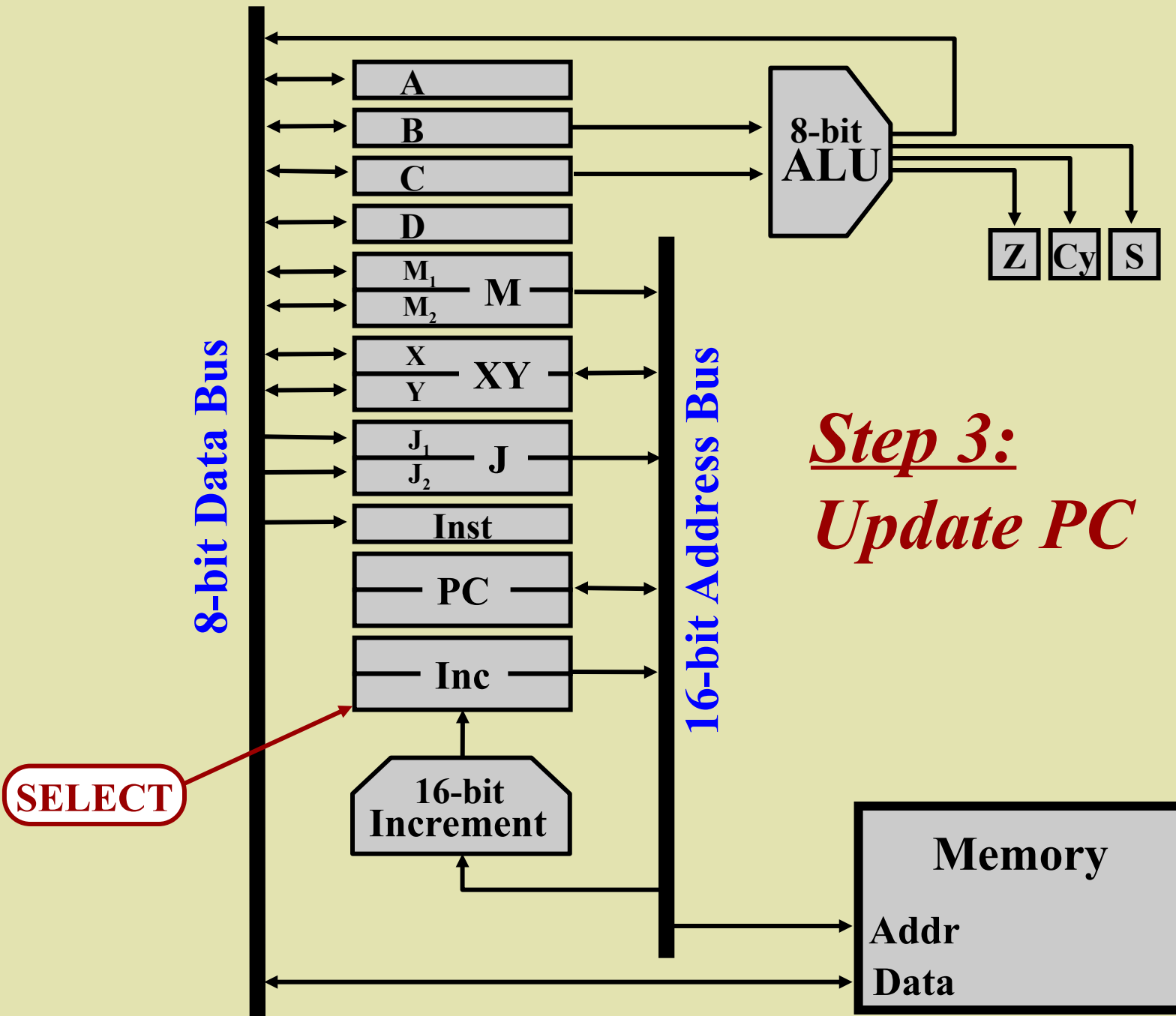


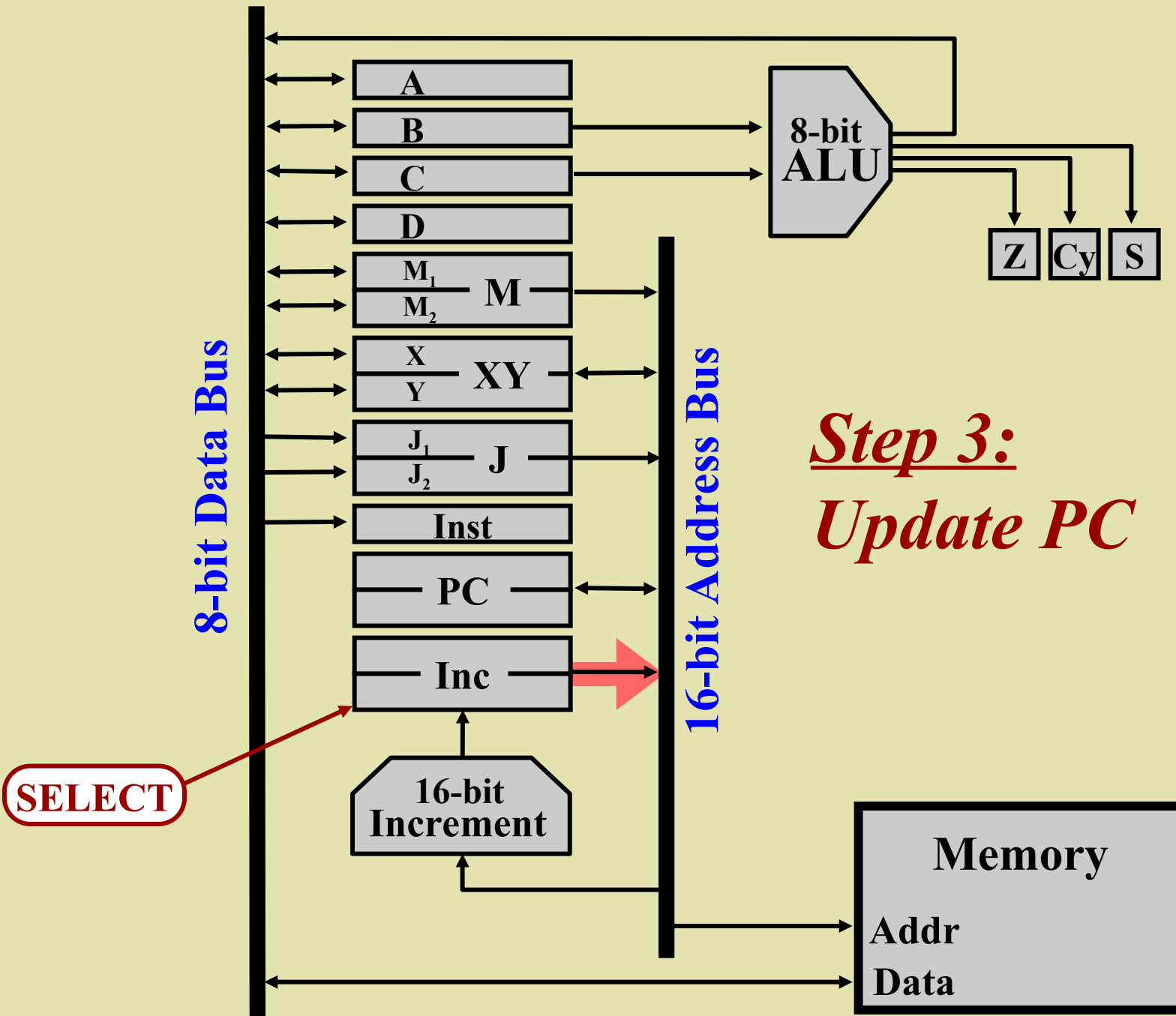


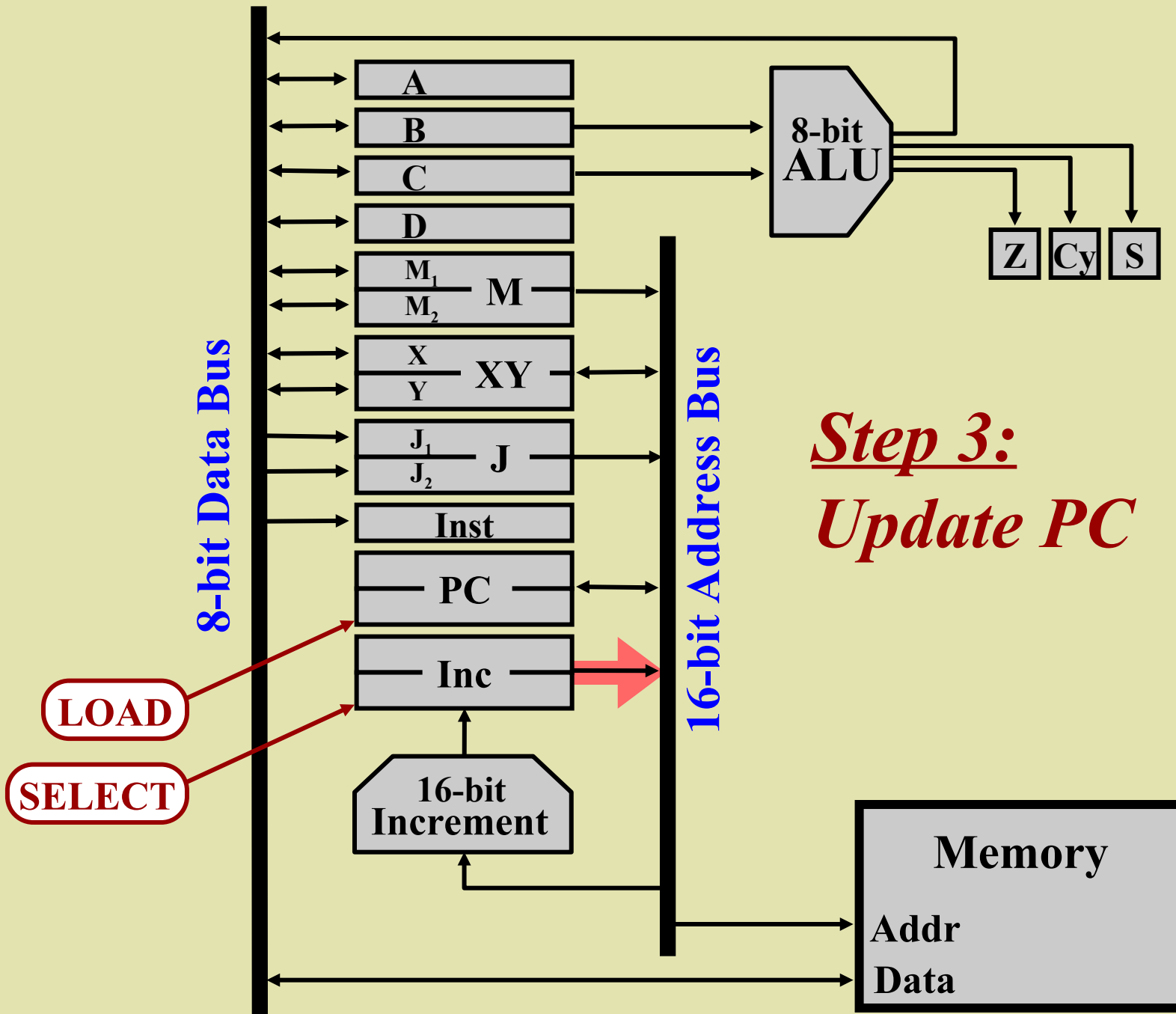
*Step 2:  
Increment PC*



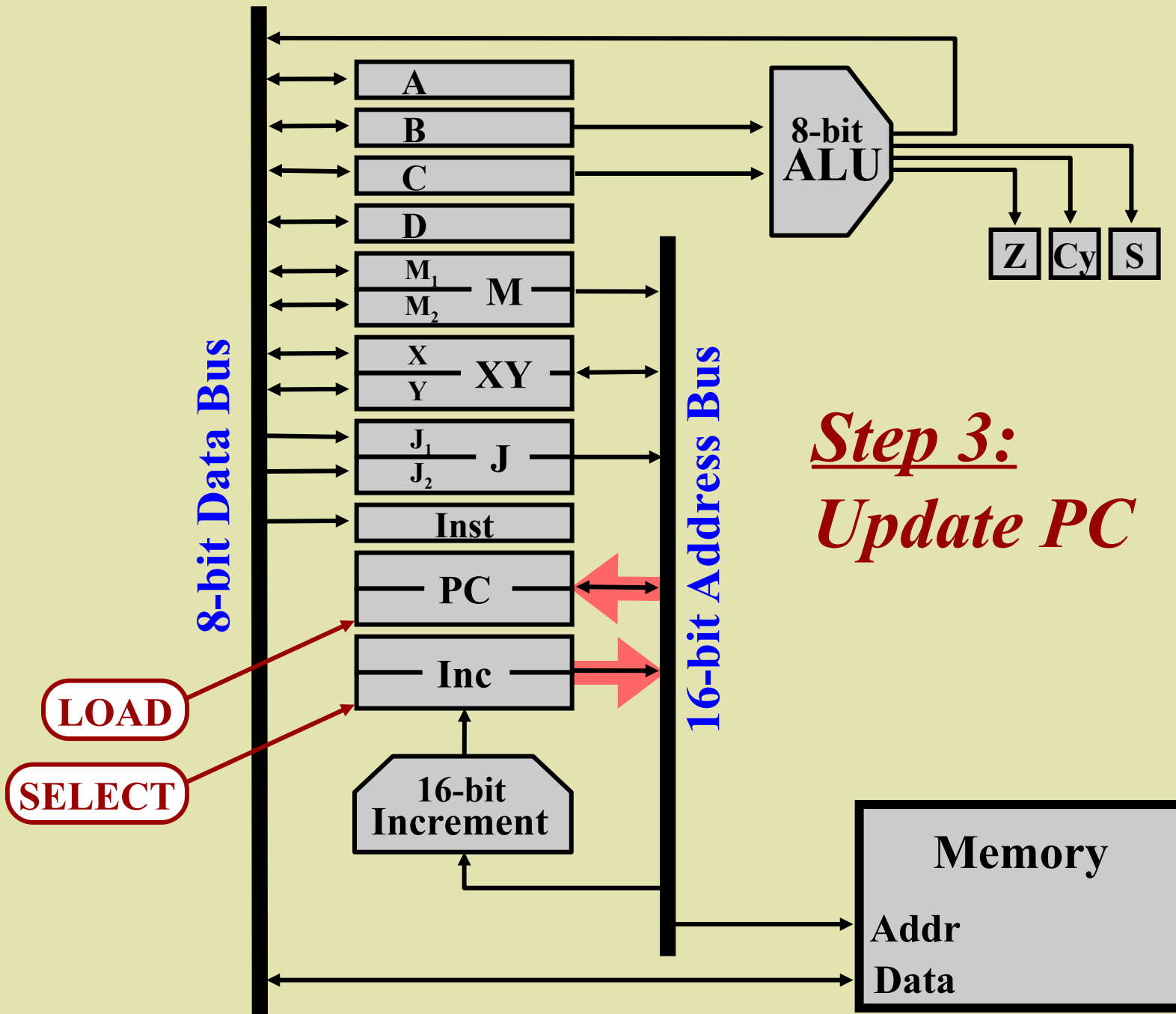
*Step 3:  
Update PC*

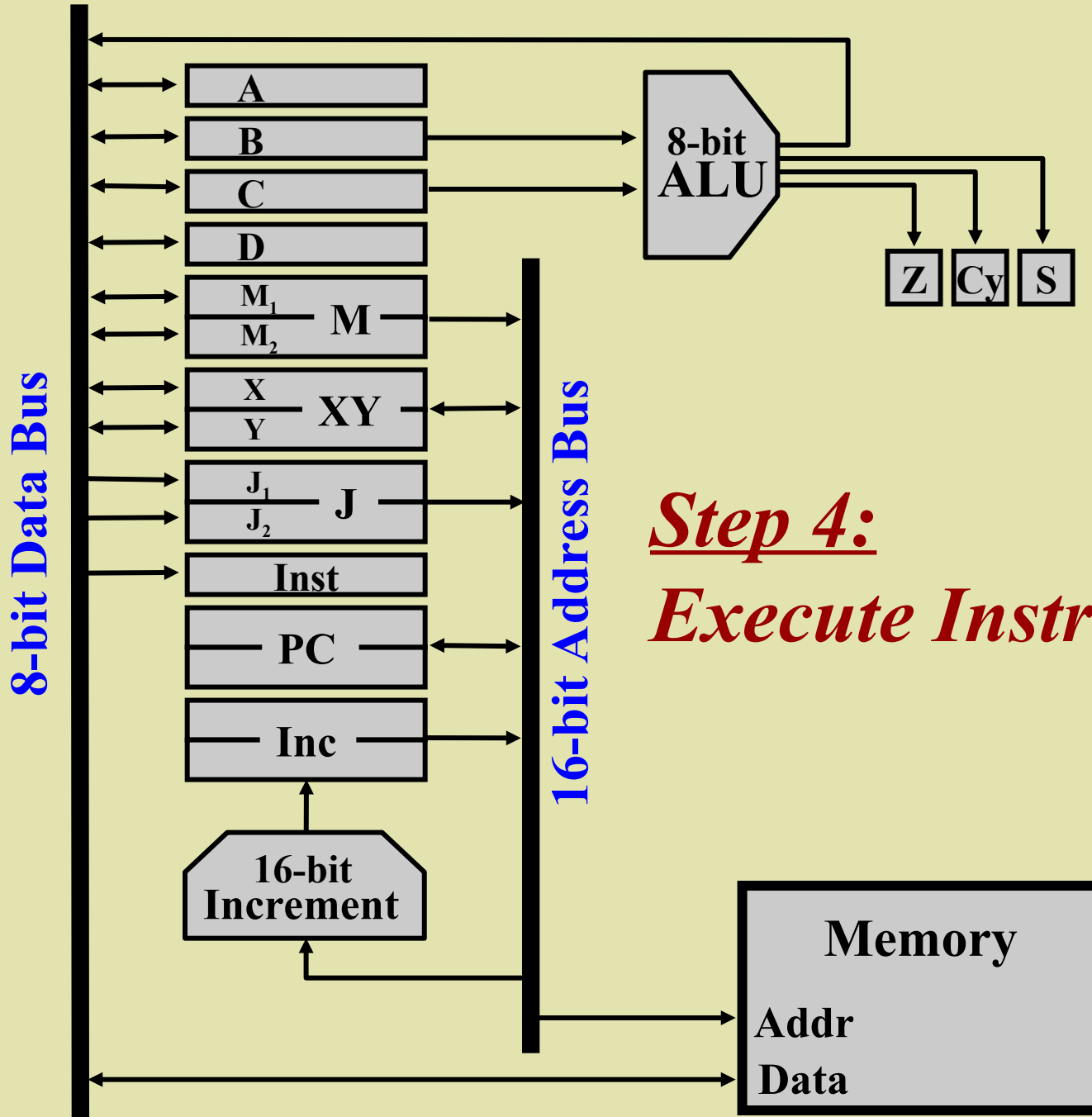




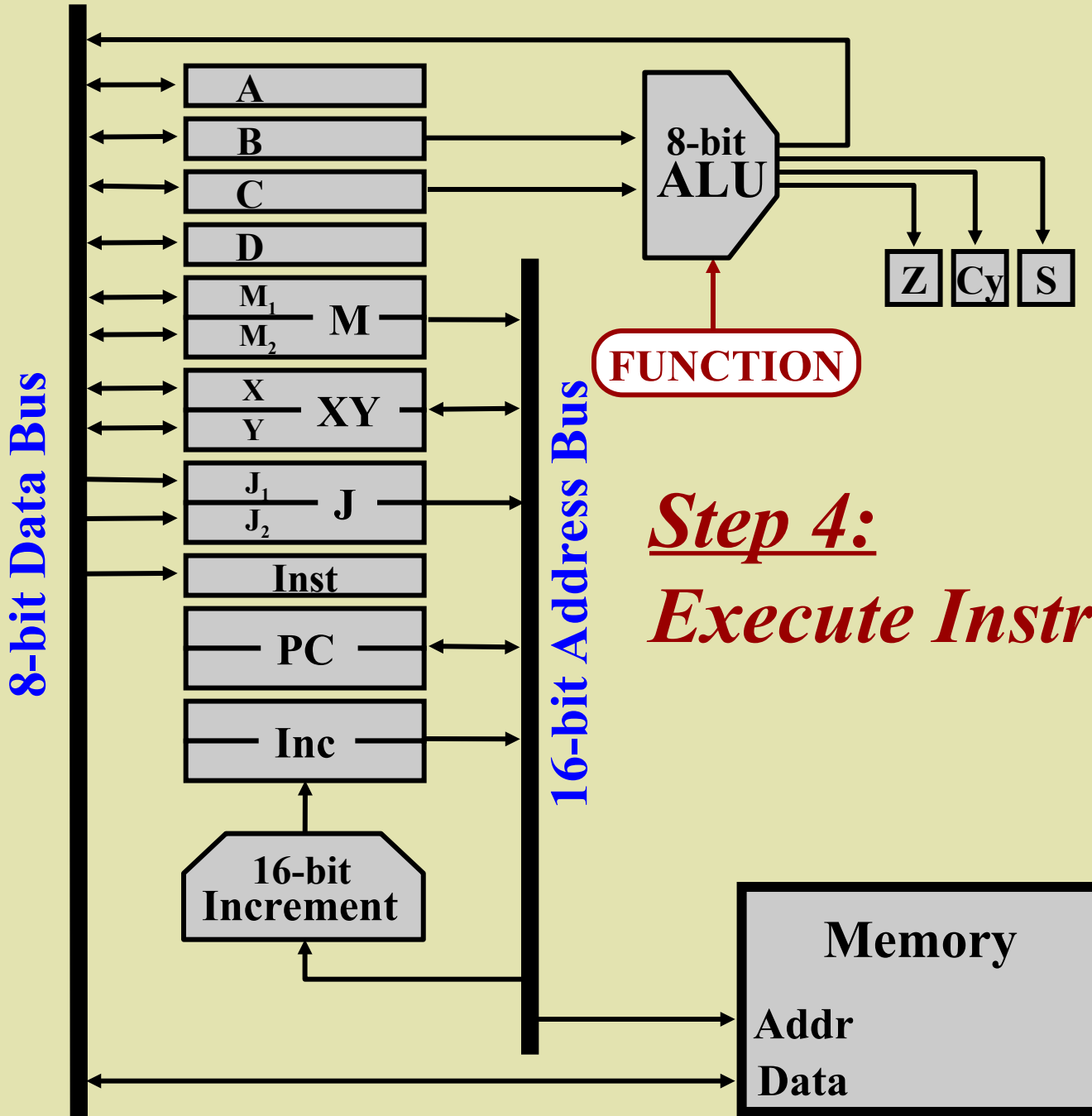


*Step 3:*  
*Update PC*



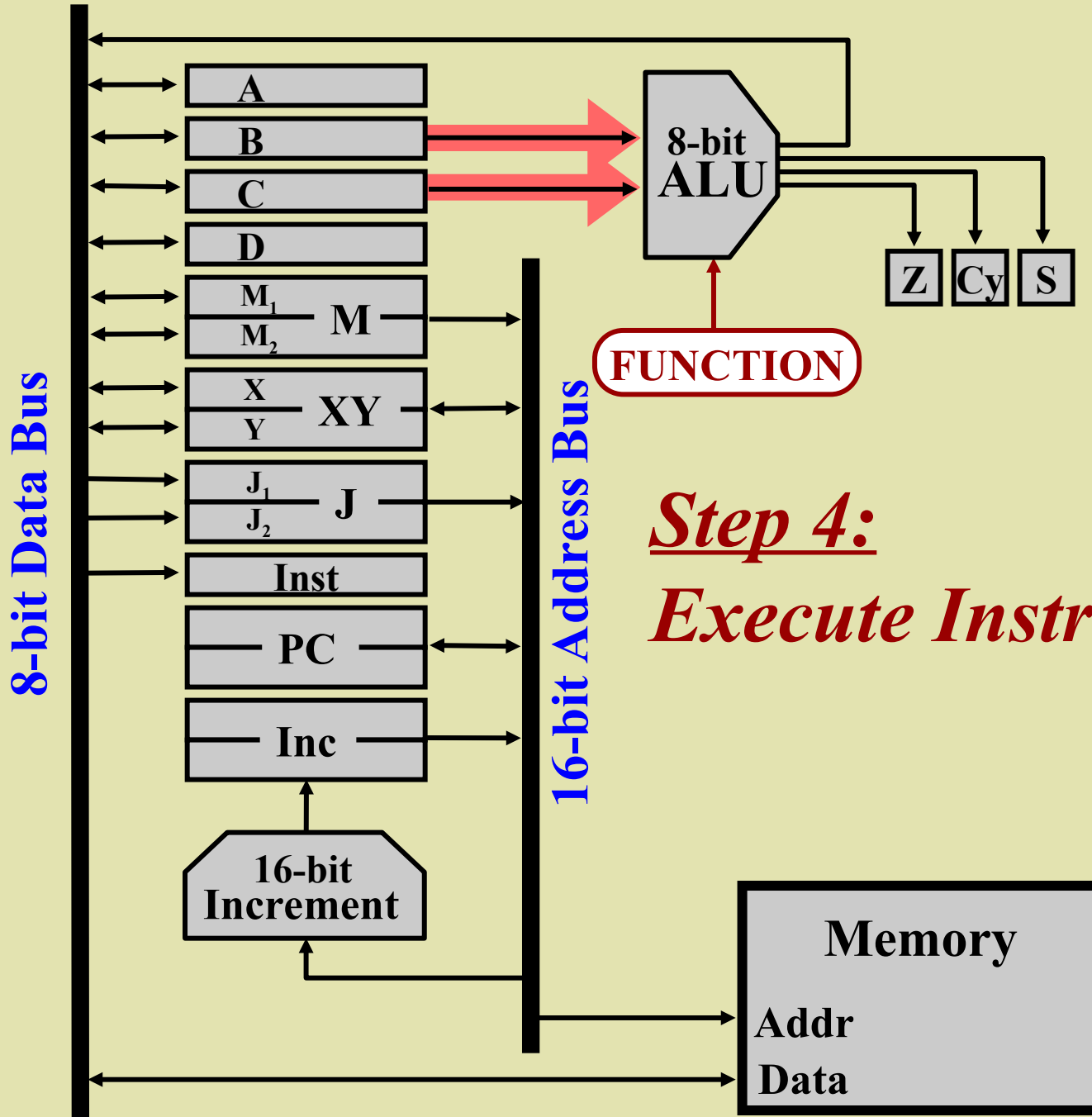


***Step 4:***  
***Execute Instruction***



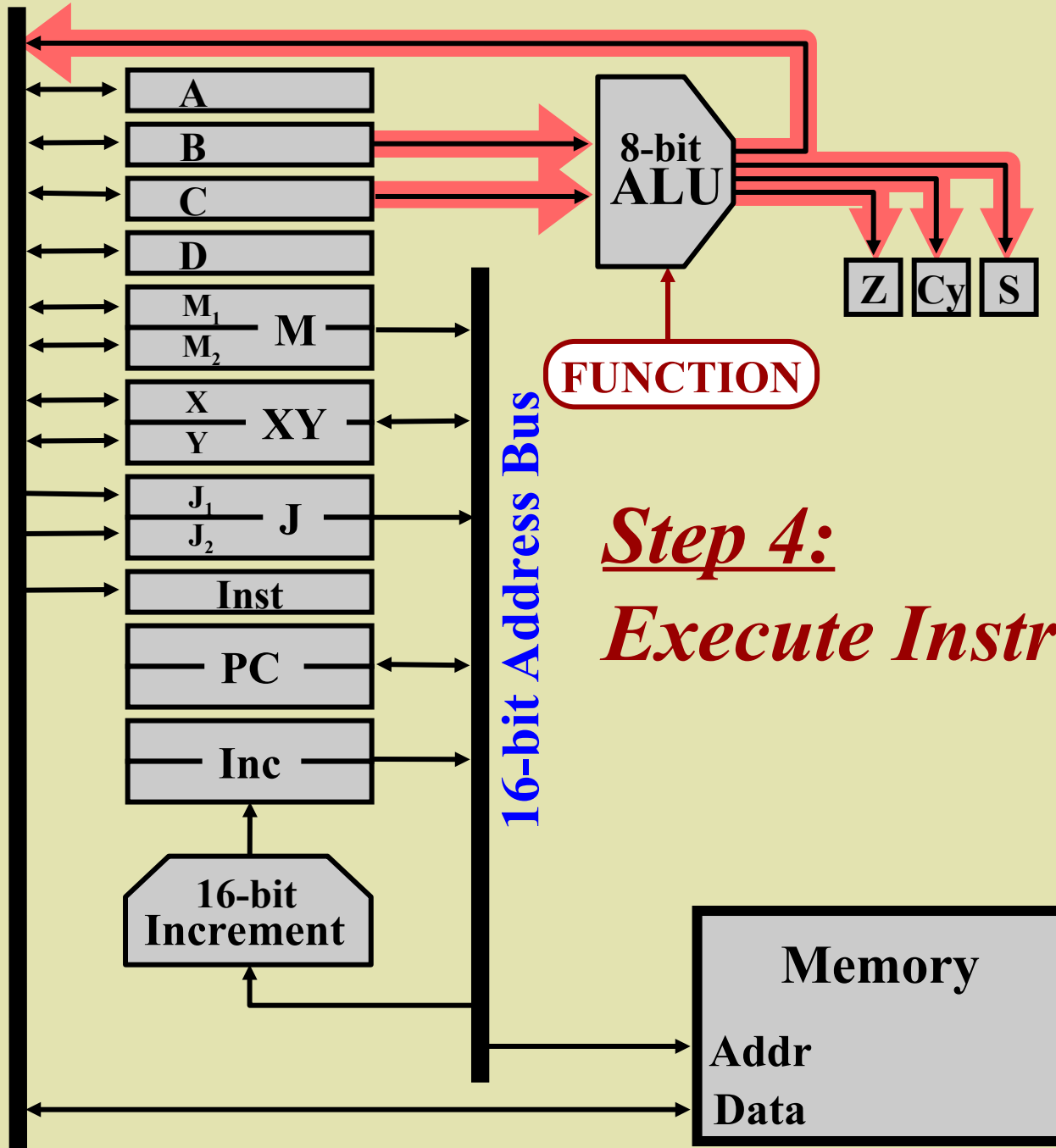
***Step 4:  
Execute Instruction***



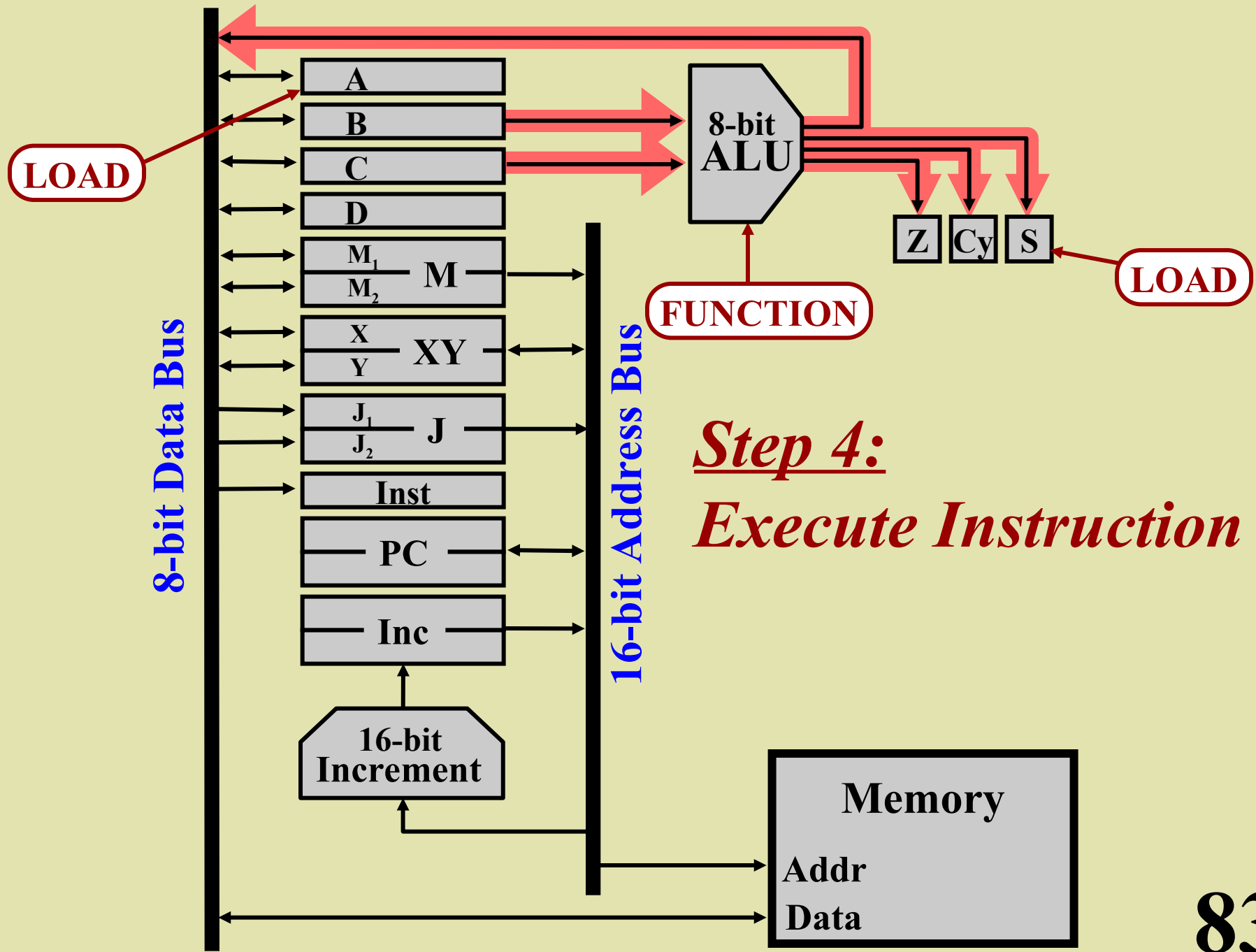


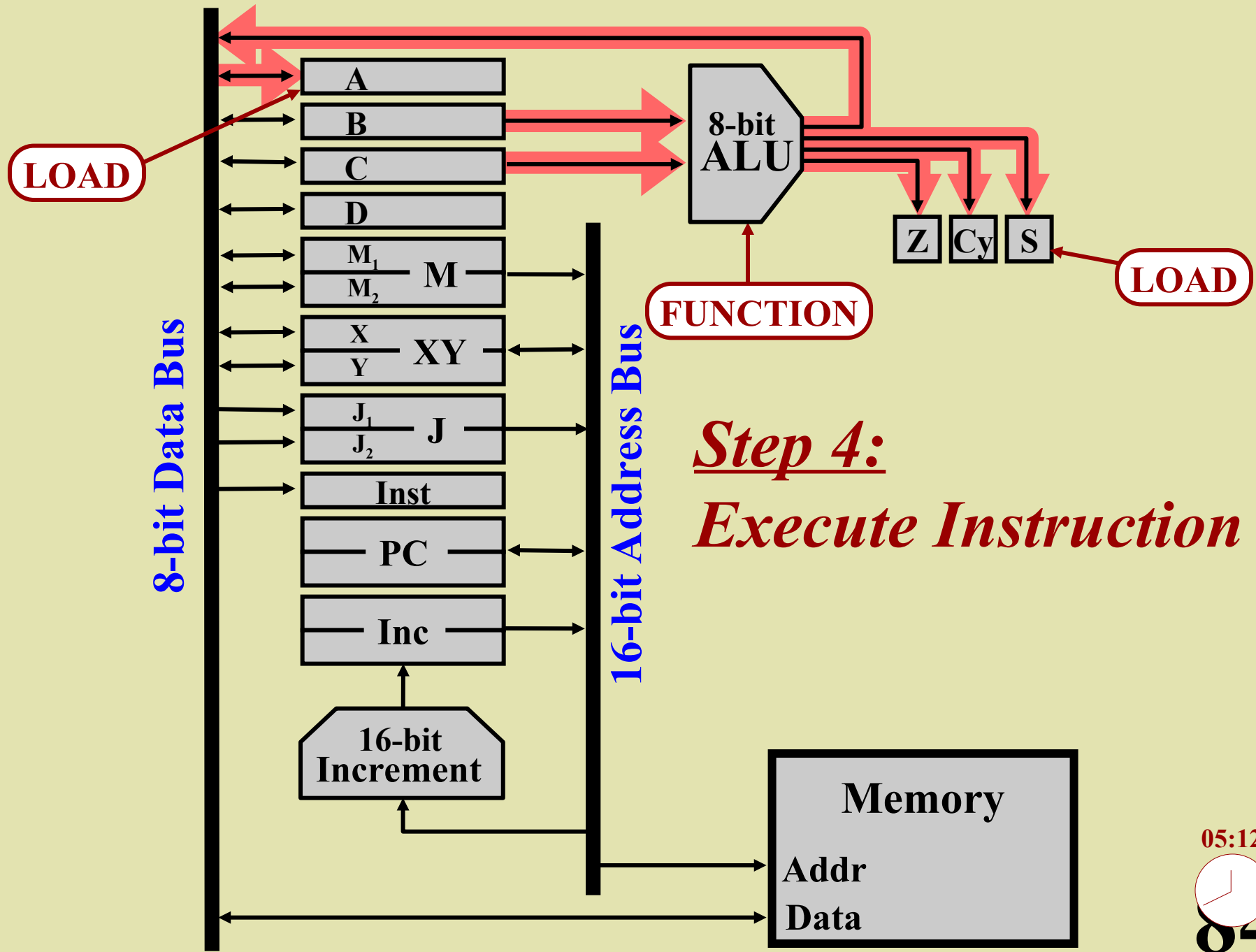
***Step 4:  
Execute Instruction***

8-bit Data Bus



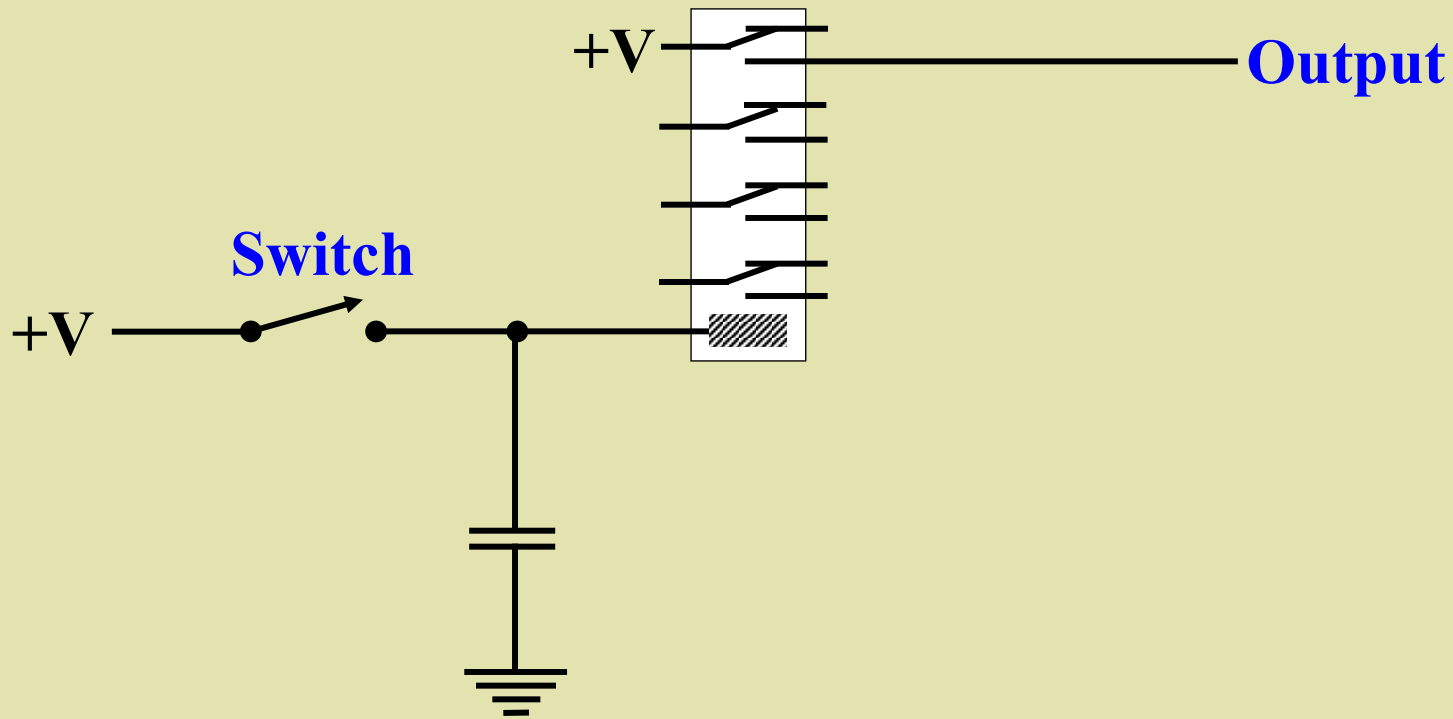
*Step 4:*  
*Execute Instruction*



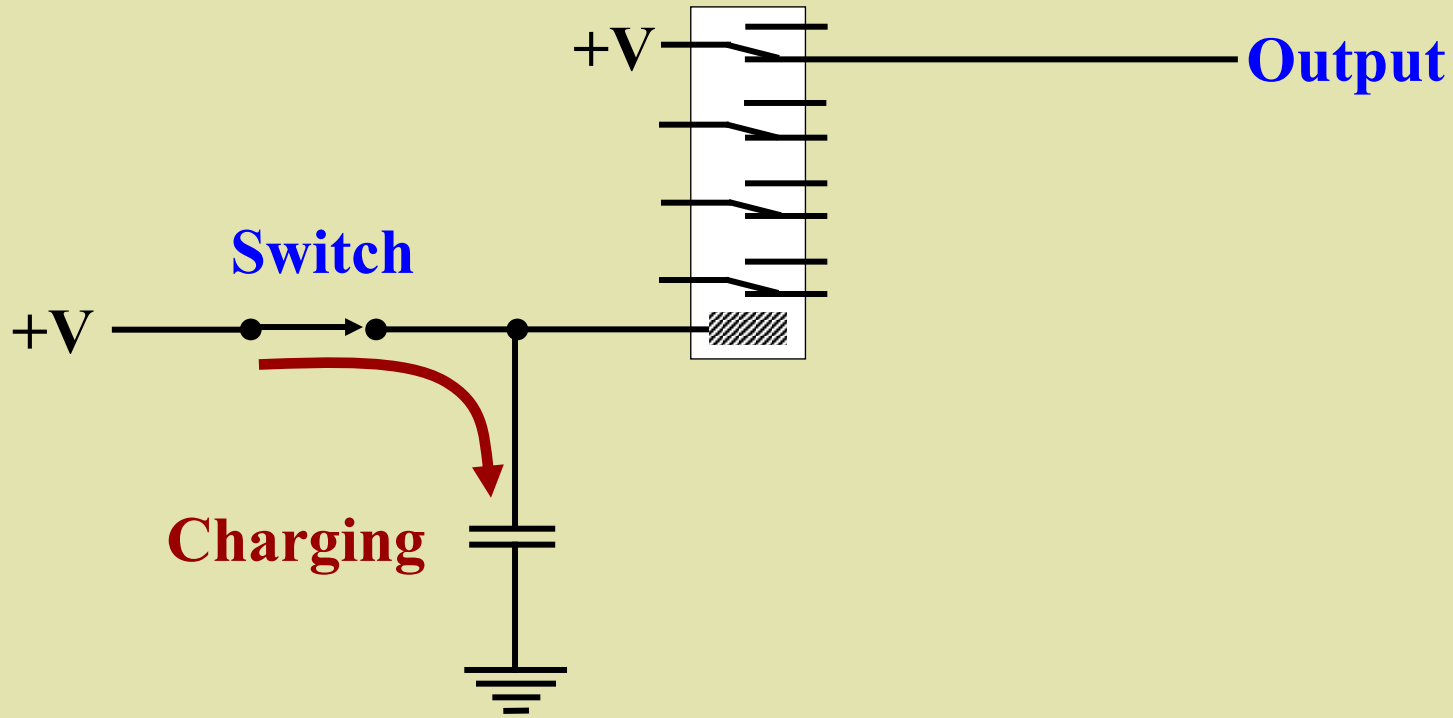


*Step 4:  
Execute Instruction*

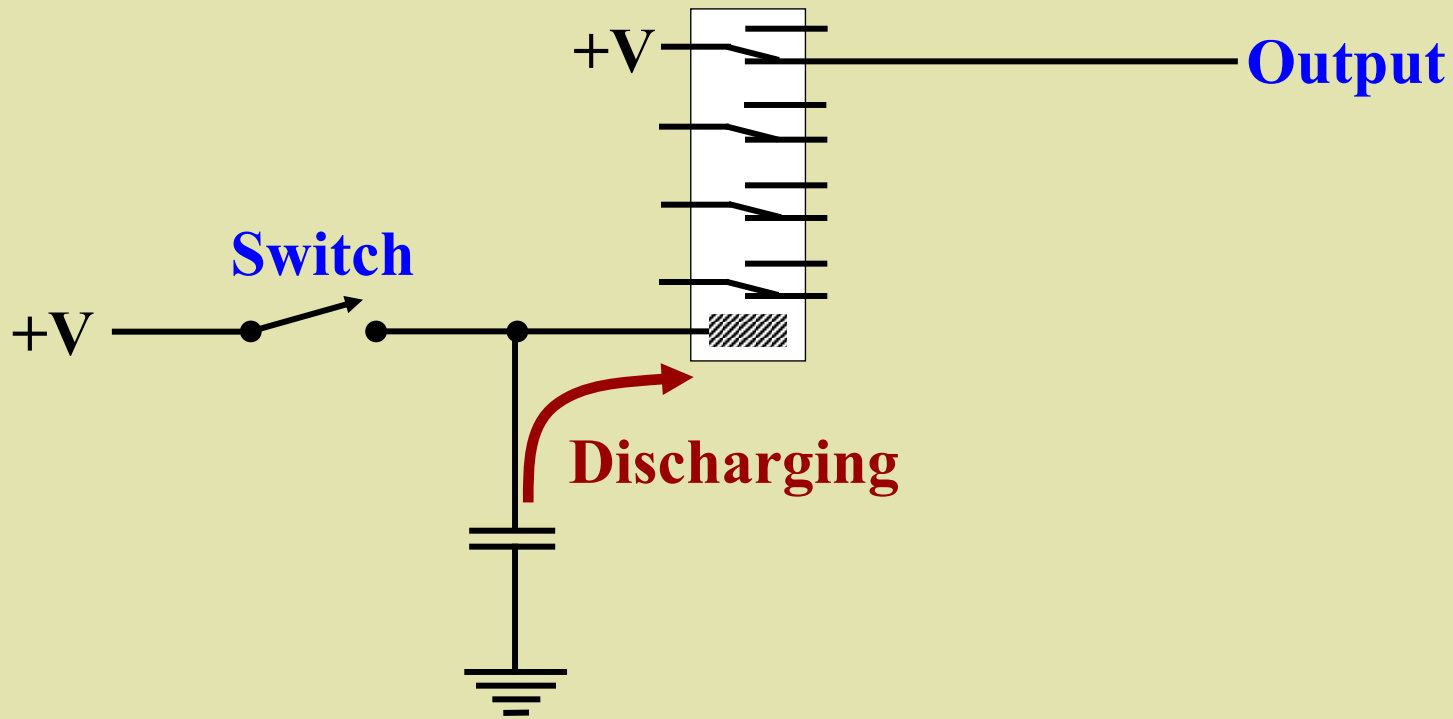
# Clock



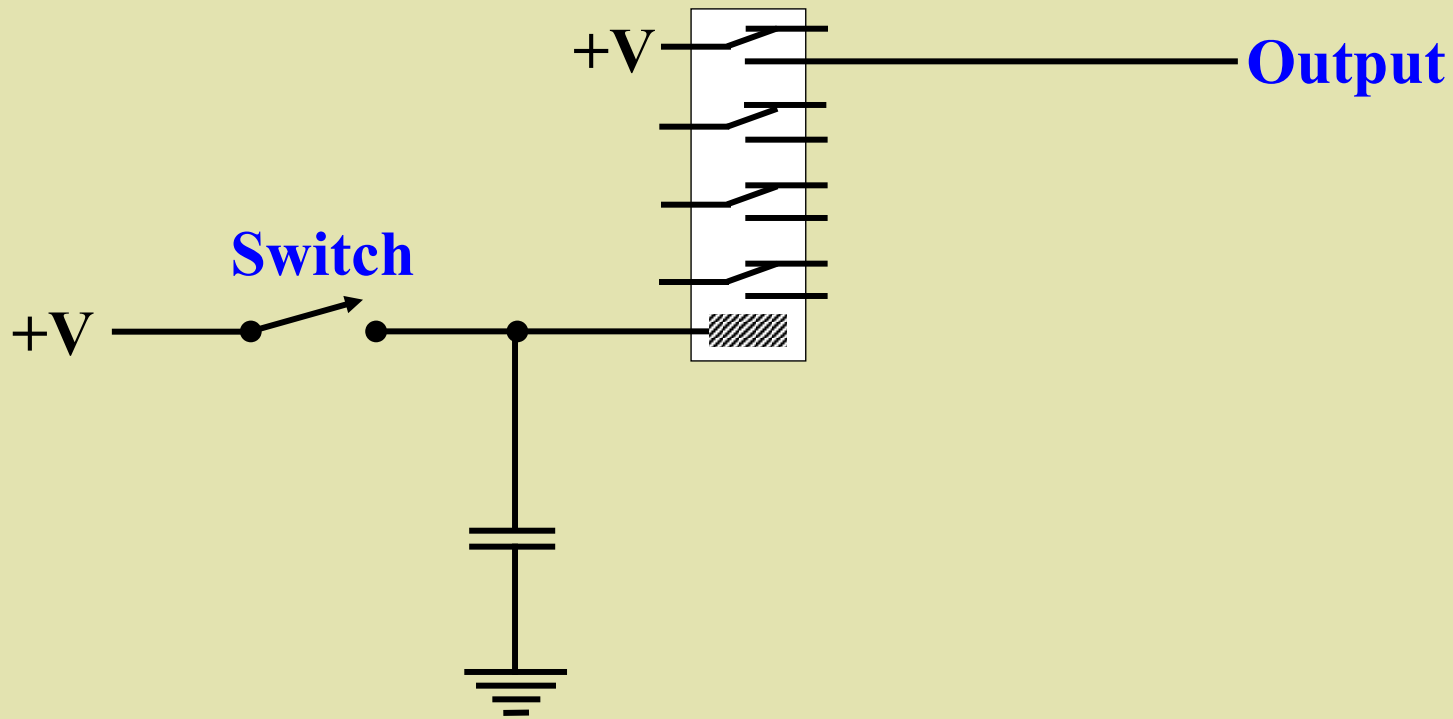
# Clock



# Clock

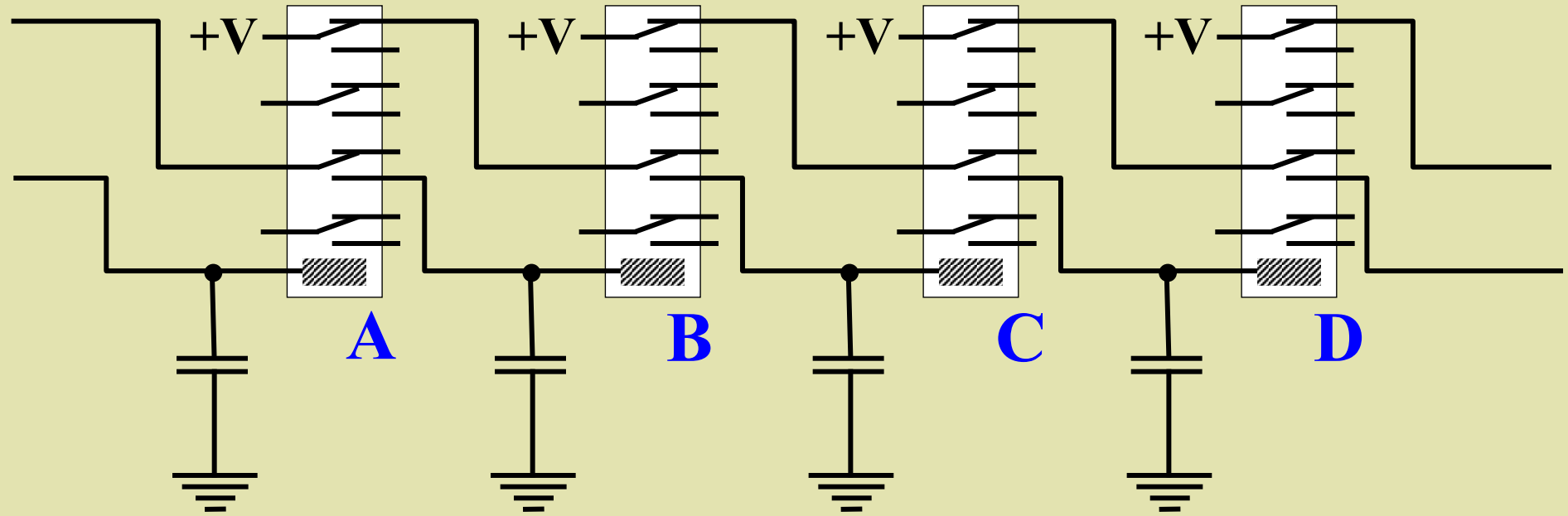


# Clock

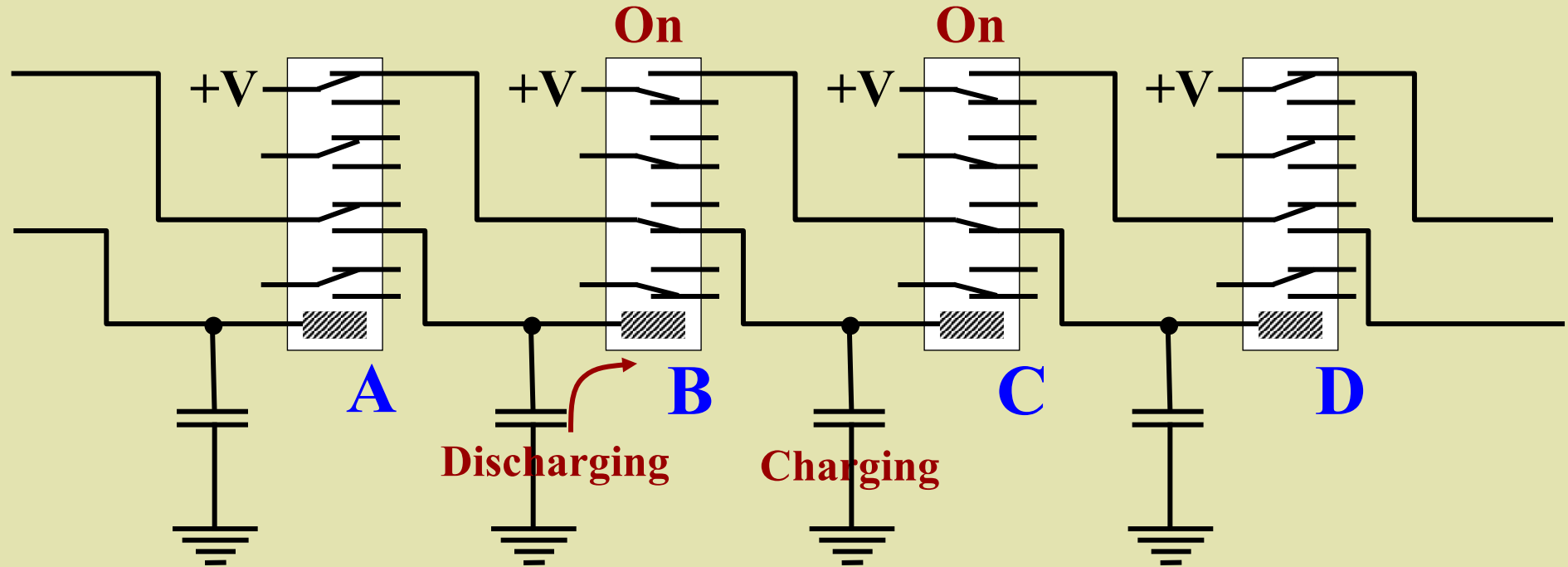




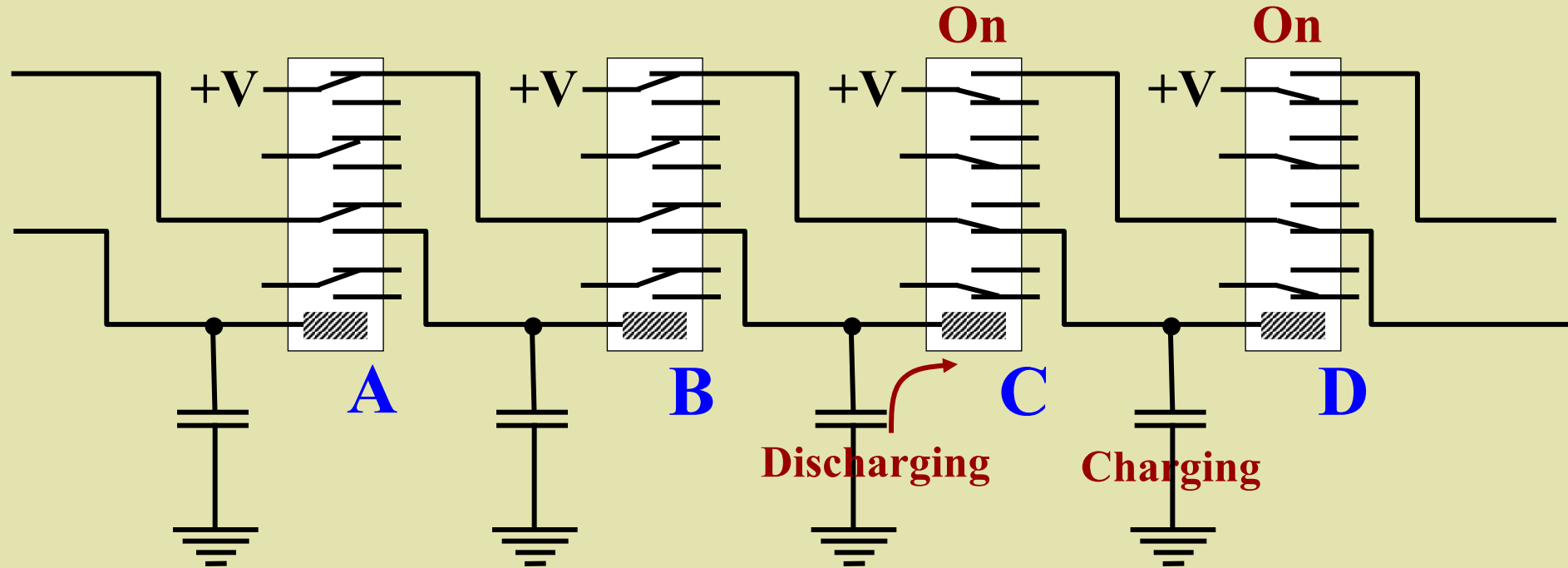
# Clock



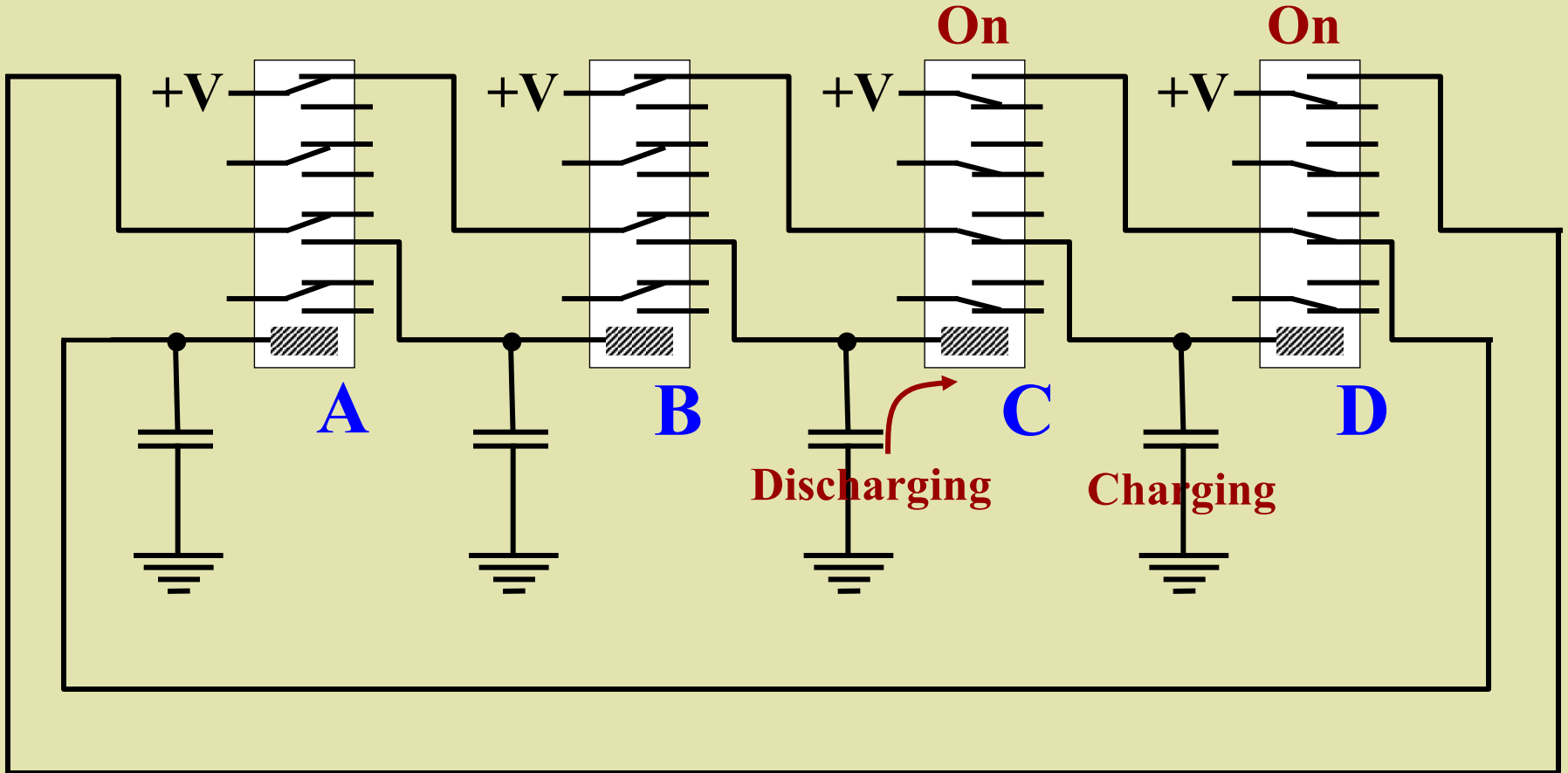
# Clock



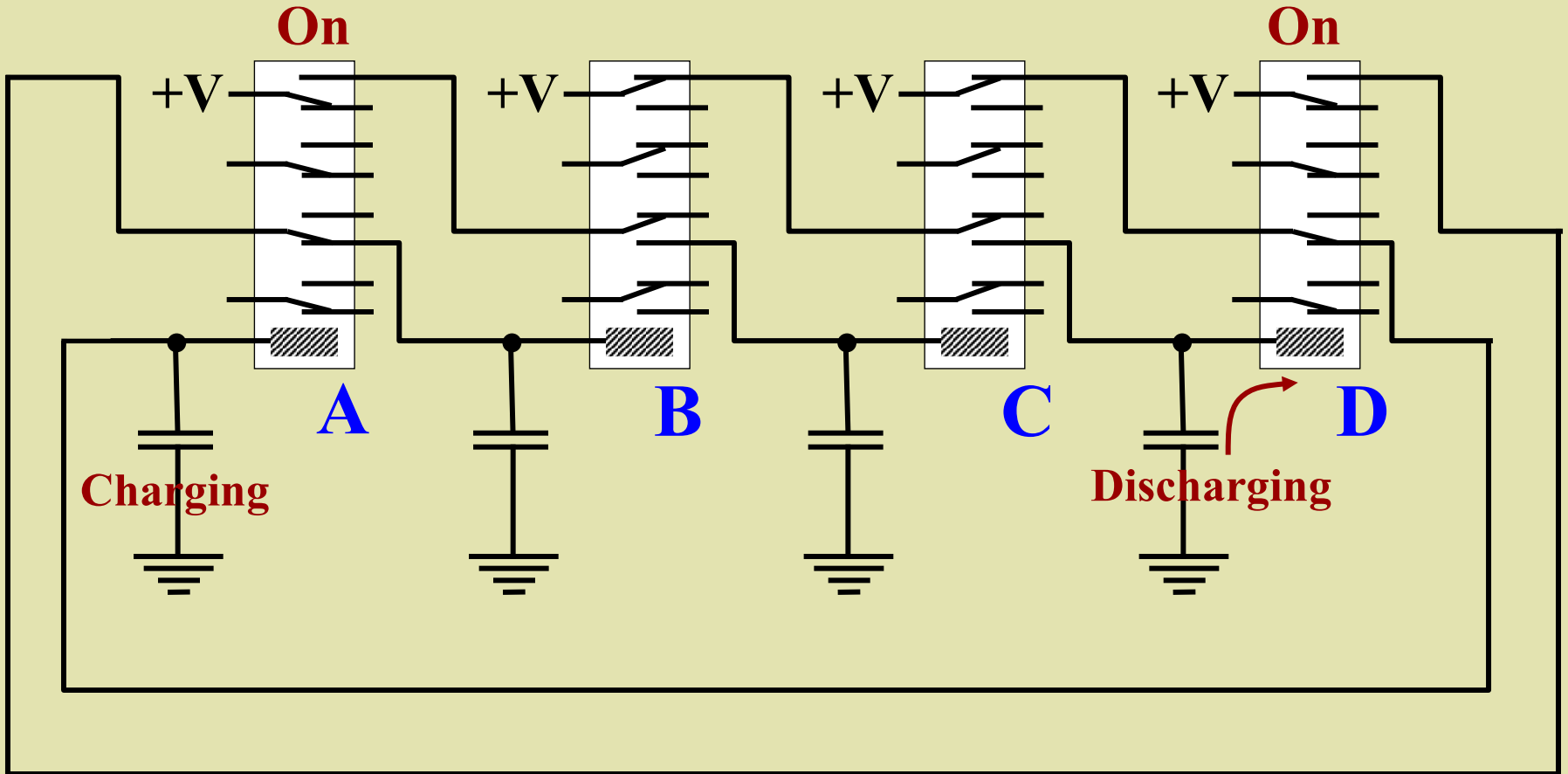
# Clock



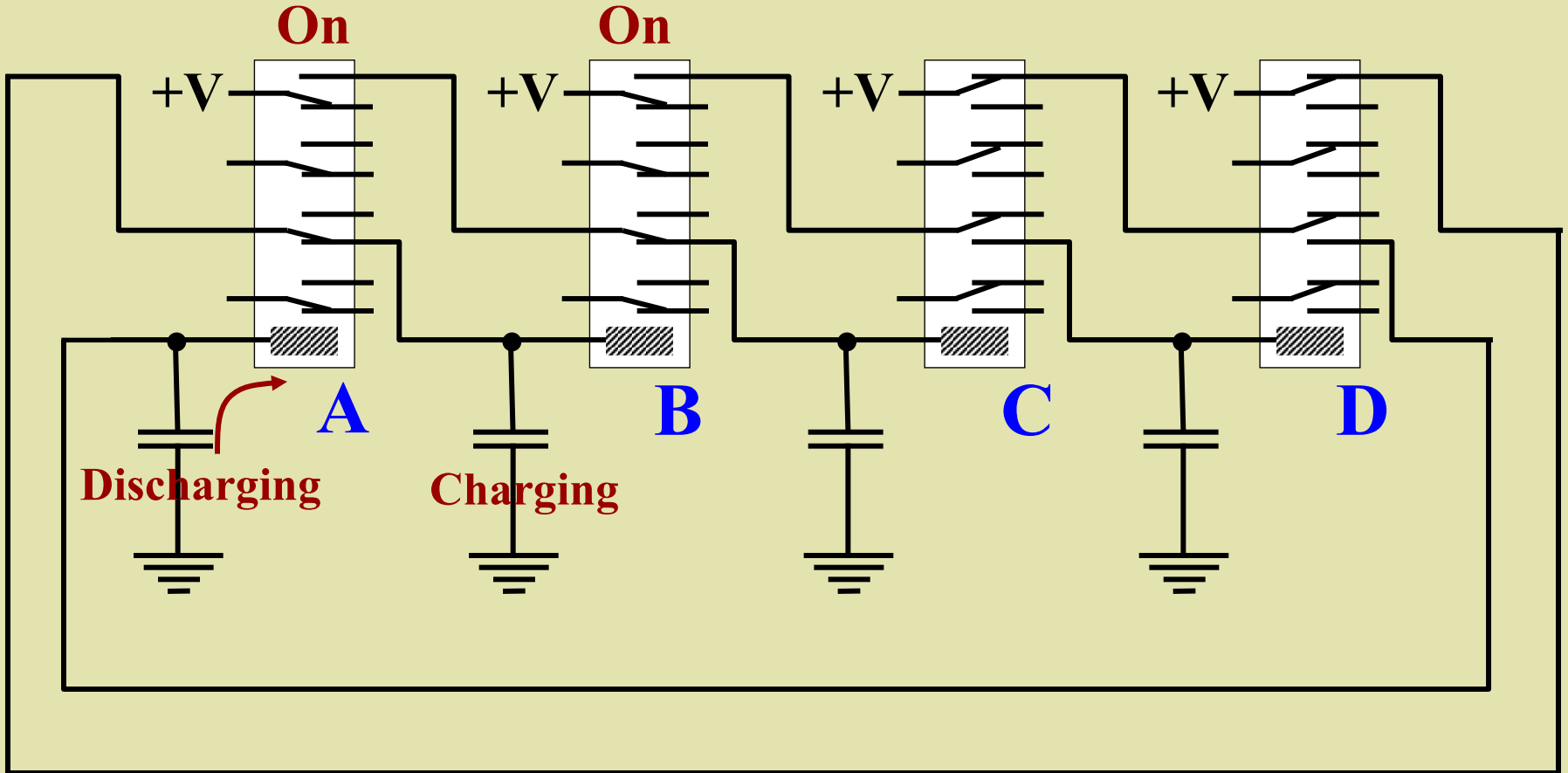
# Clock



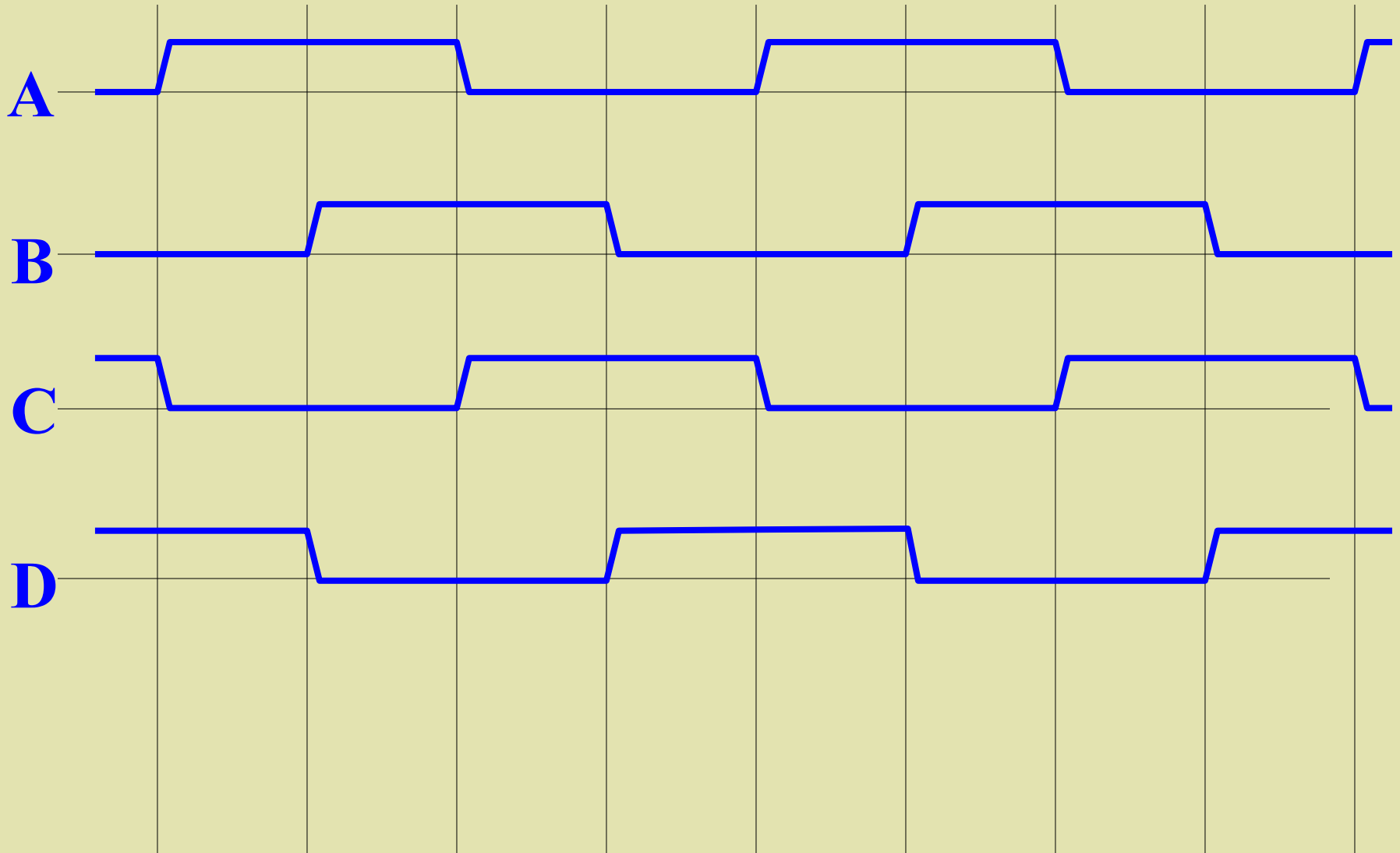
# Clock



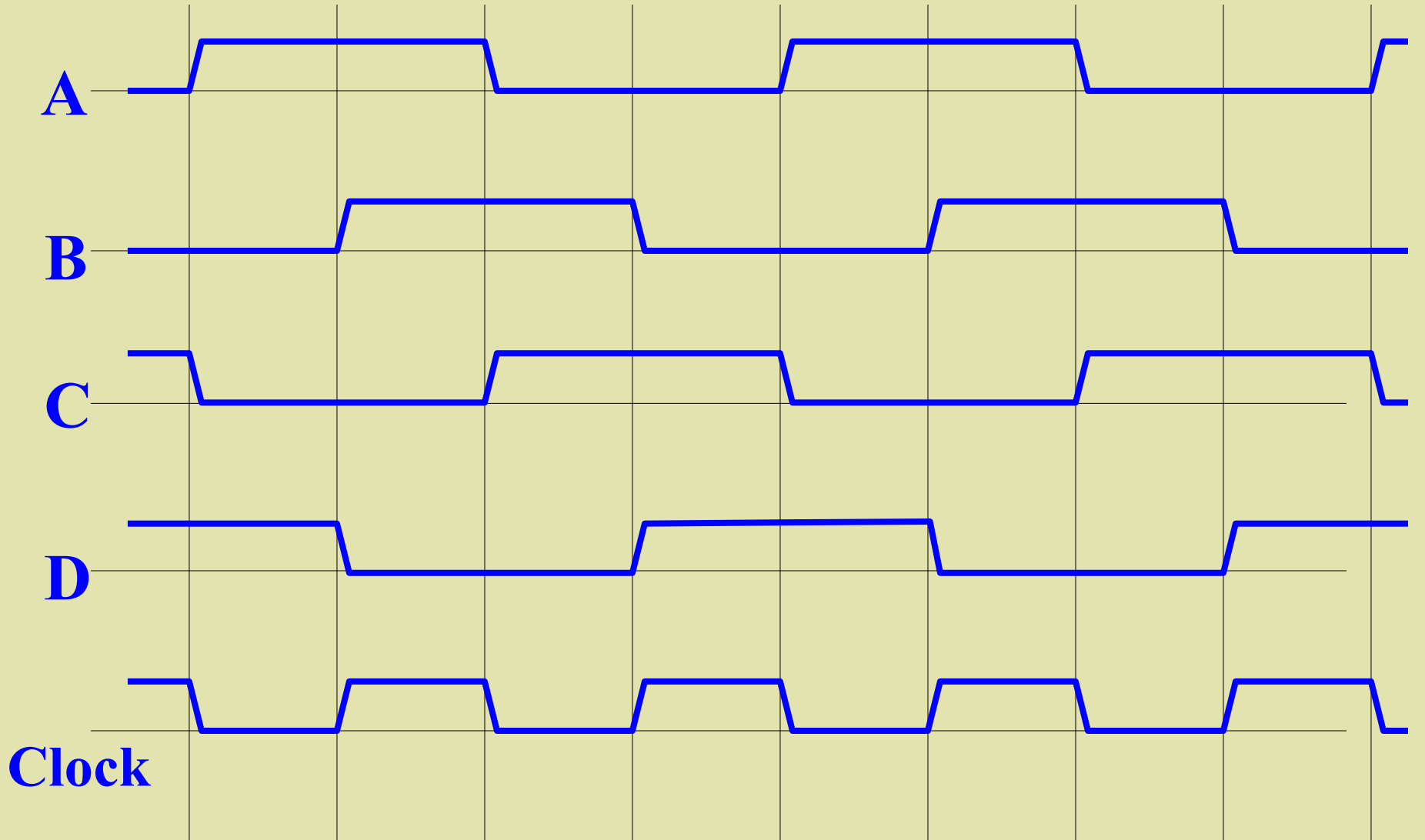
# Clock



# Clock Timing Diagram

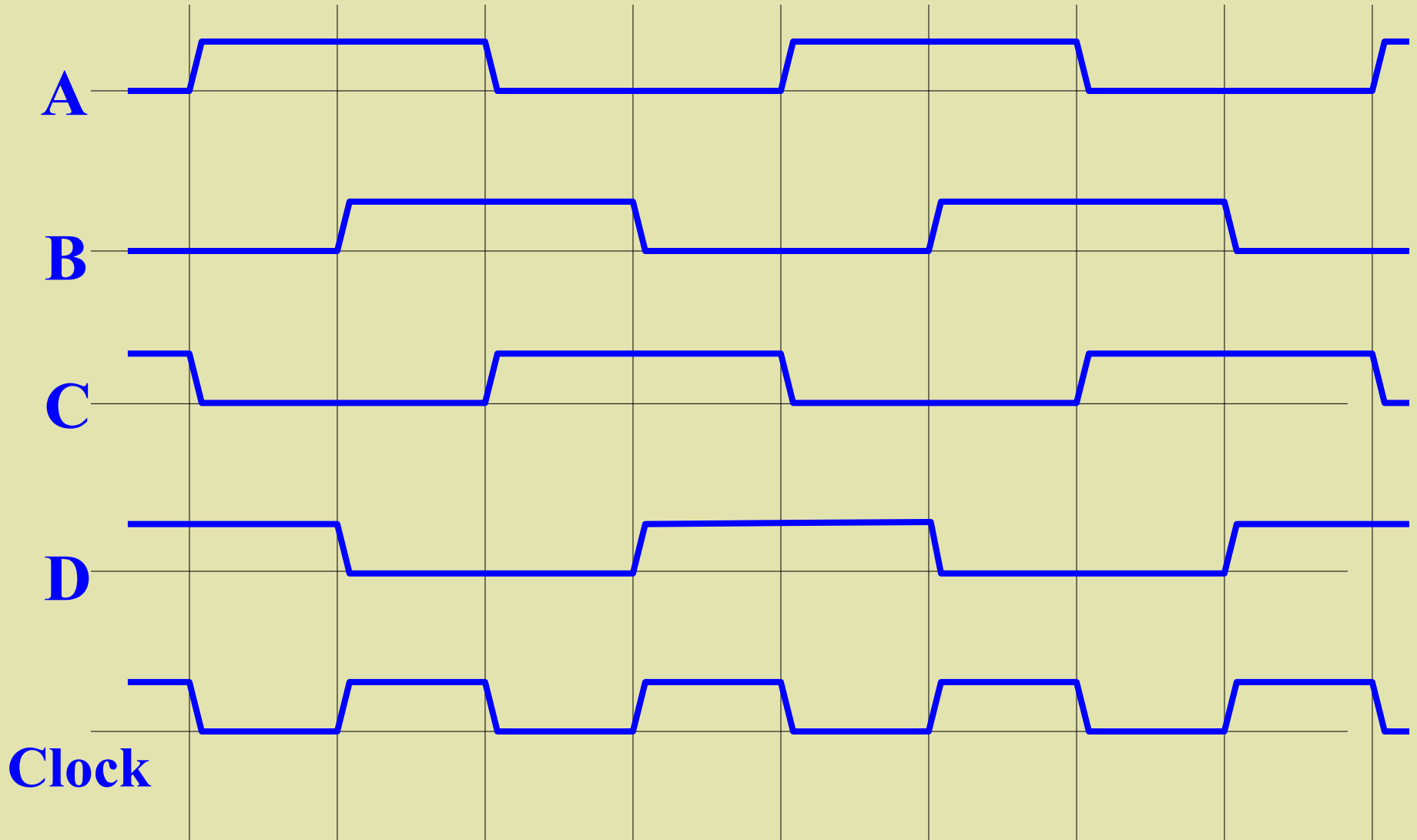


# Clock Timing Diagram



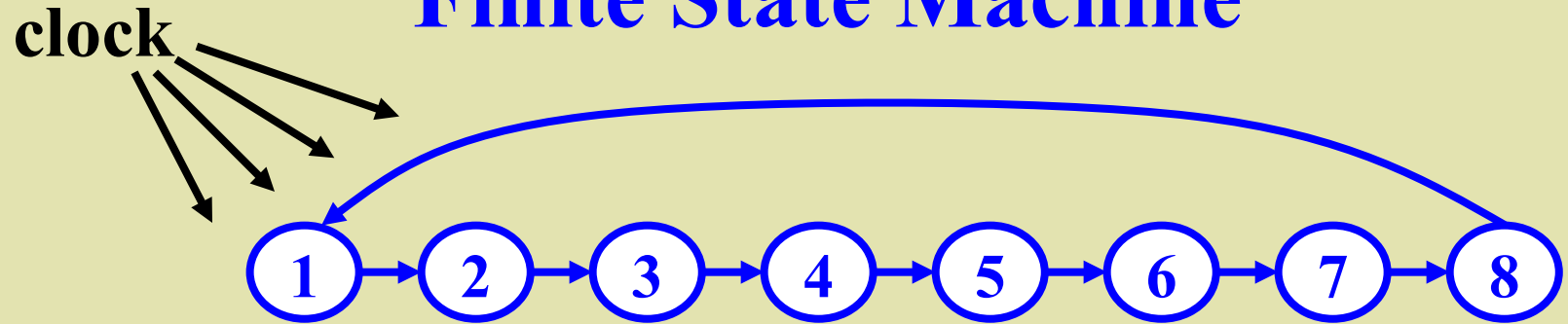


# Clock Timing Diagram

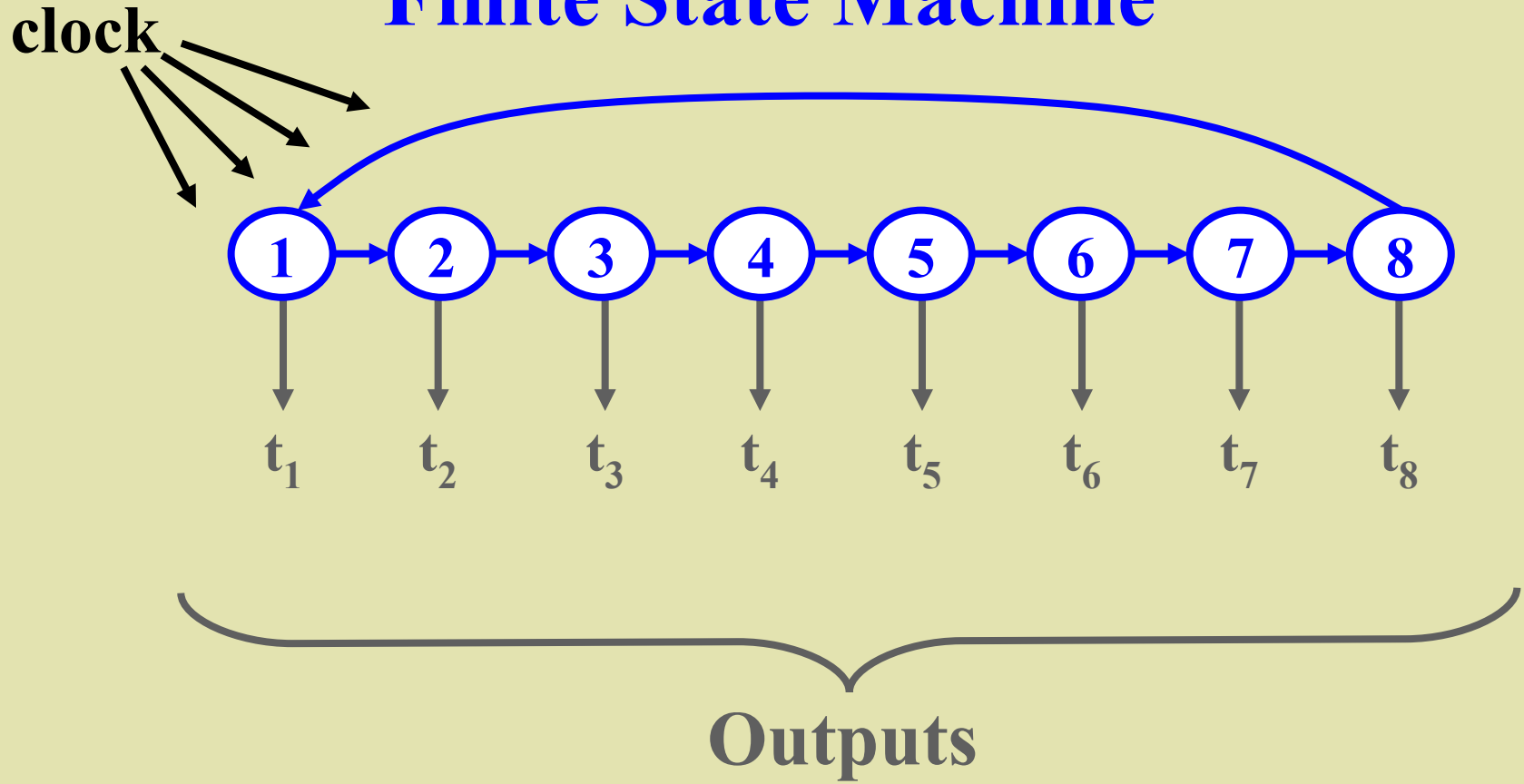


**Clock = (A and B) or (C and D)**

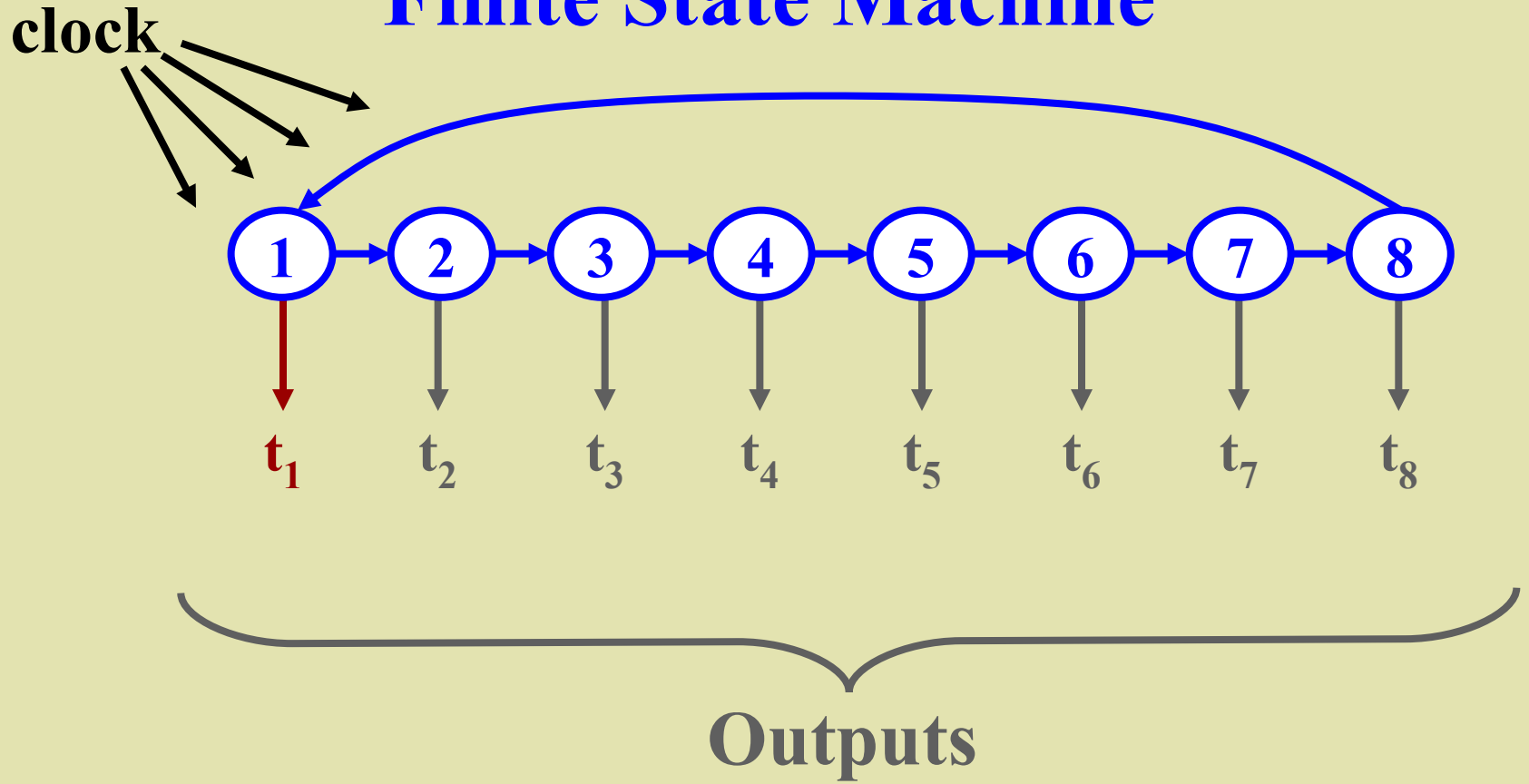
# Finite State Machine



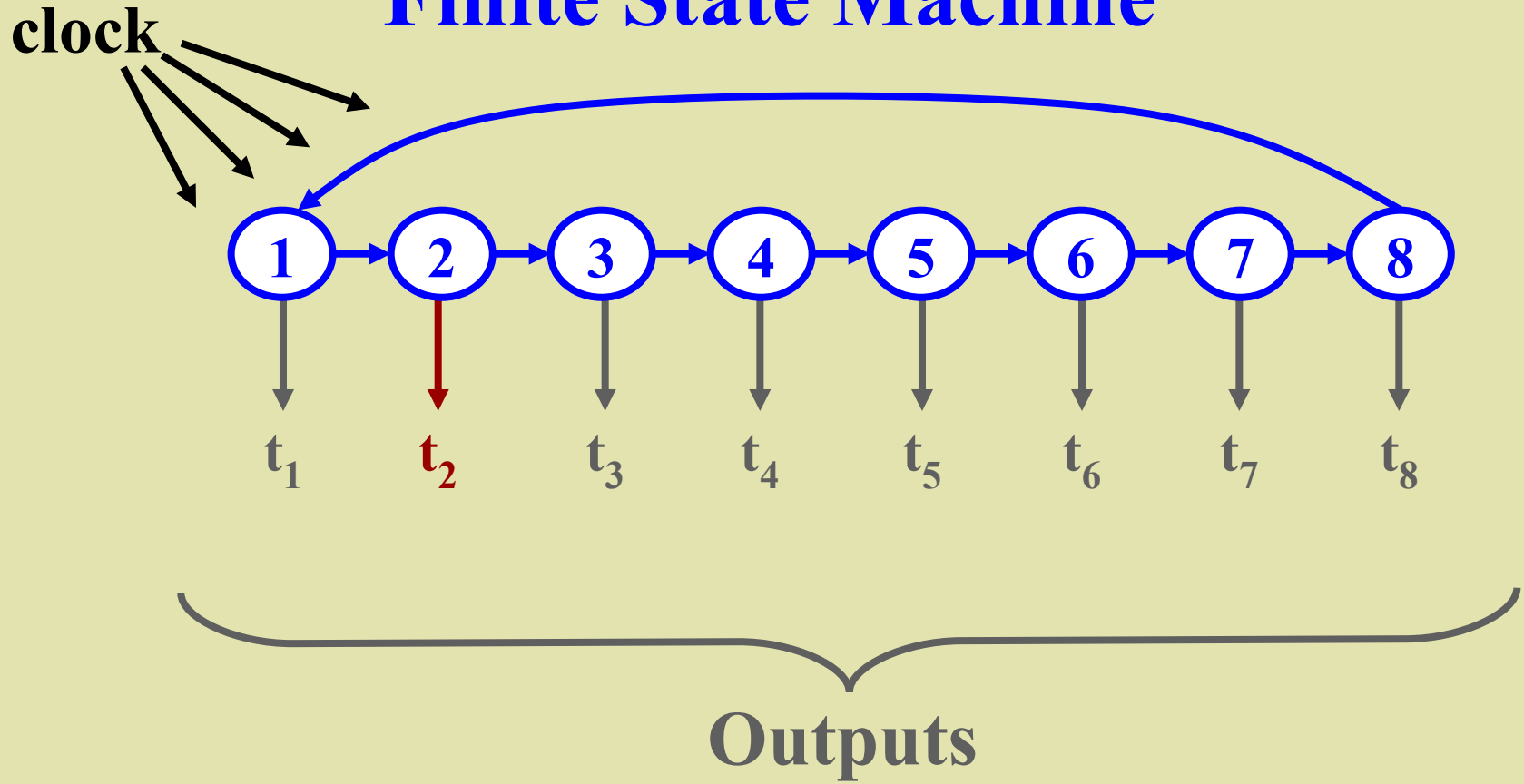
# Finite State Machine



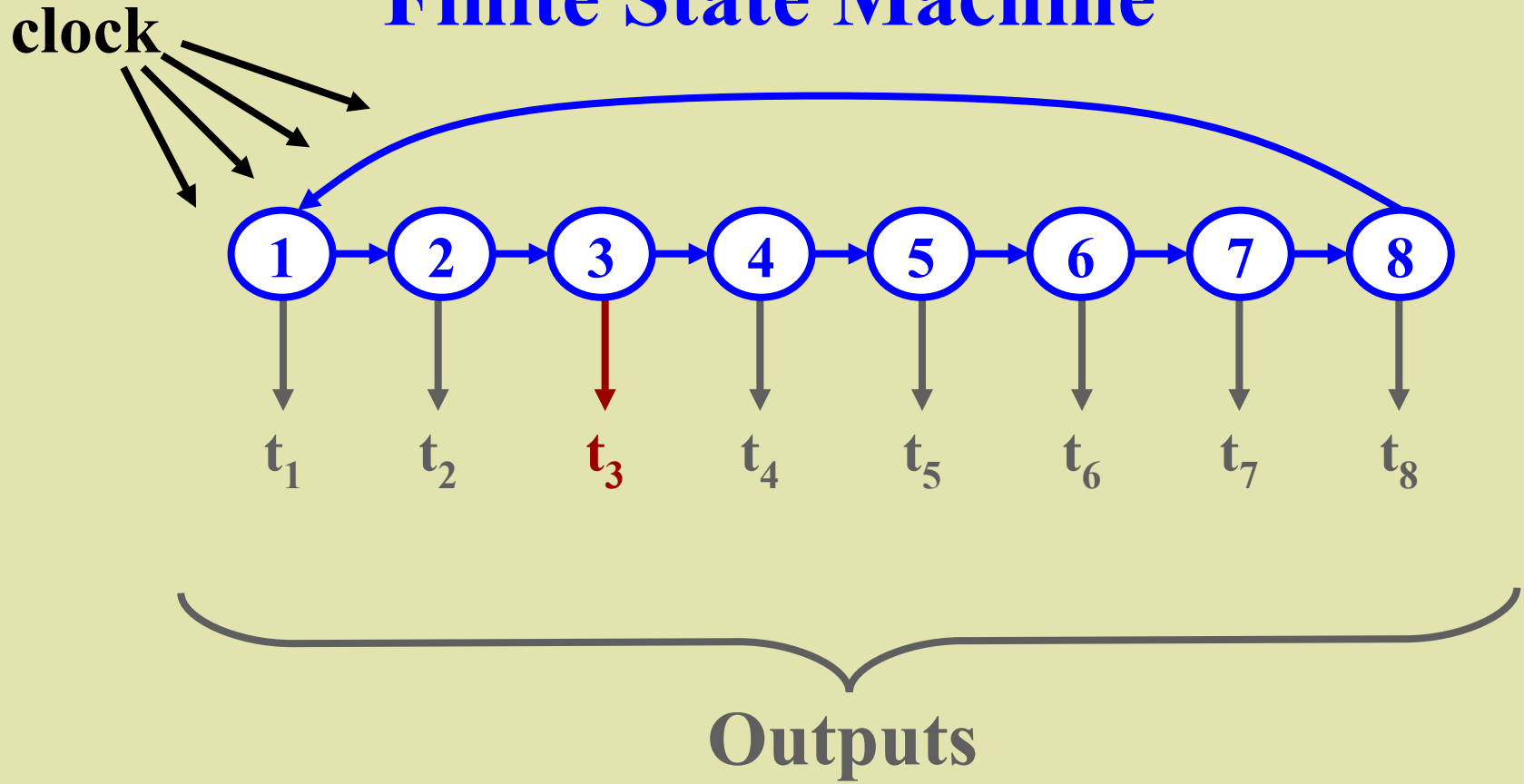
# Finite State Machine



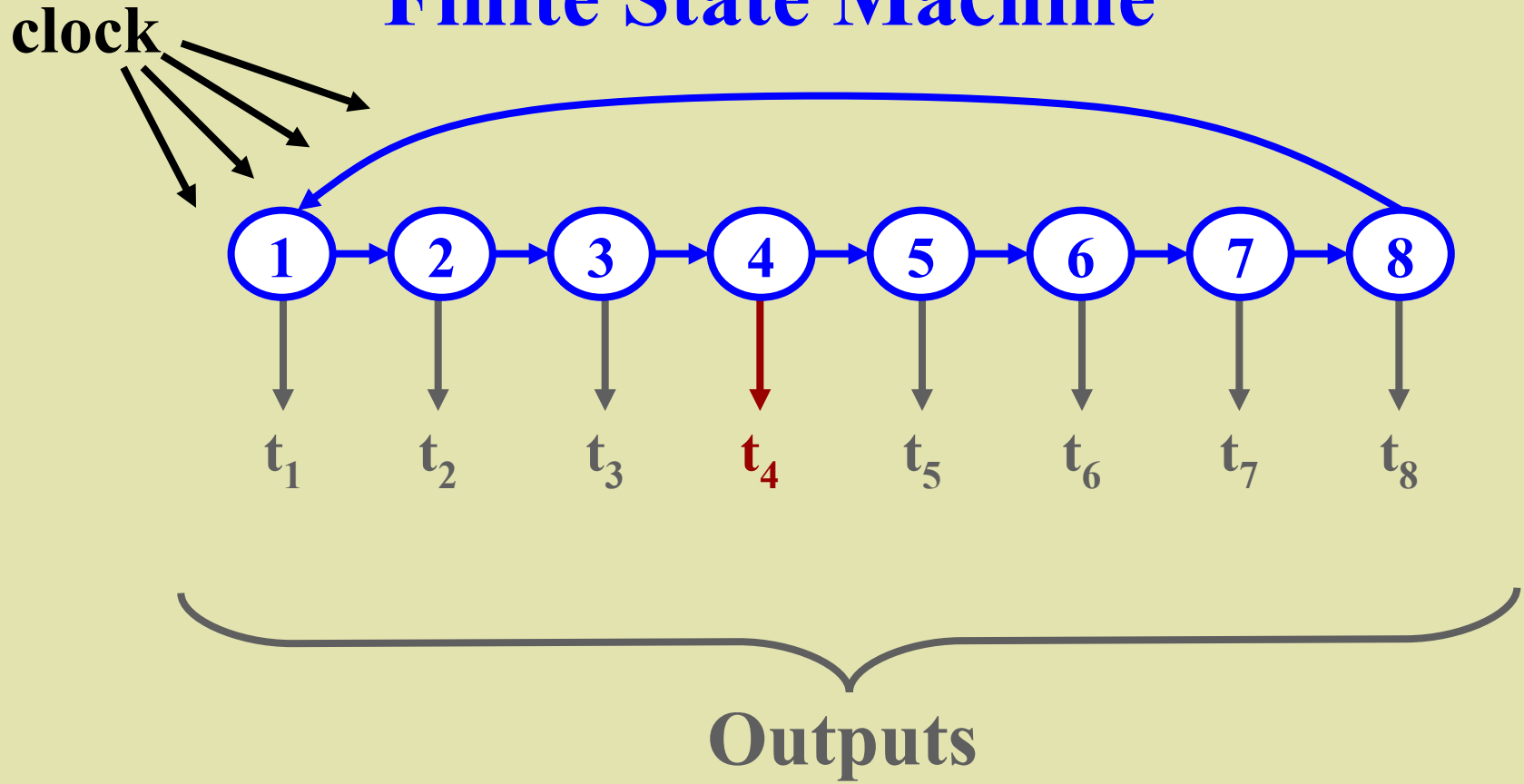
# Finite State Machine



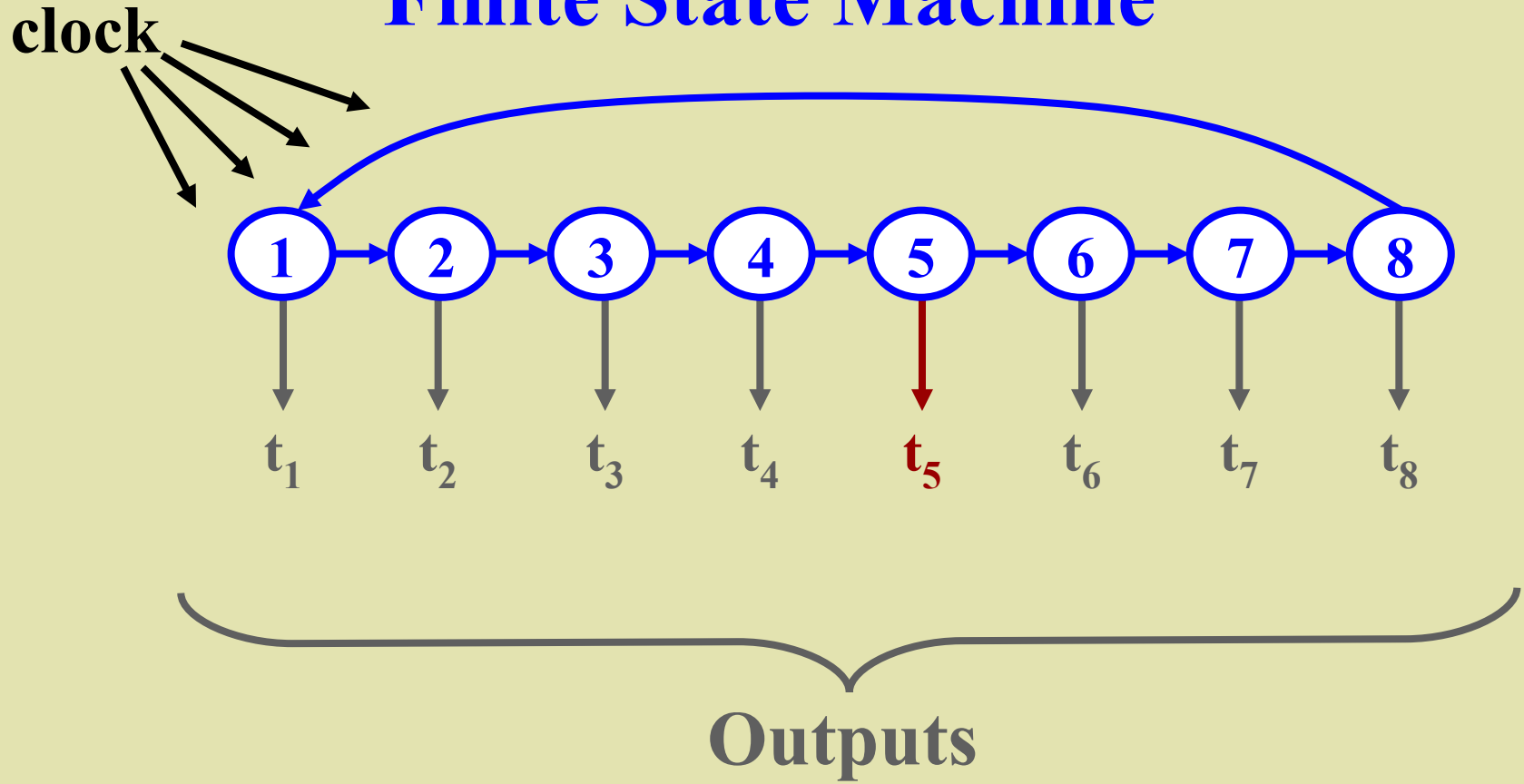
# Finite State Machine



# Finite State Machine

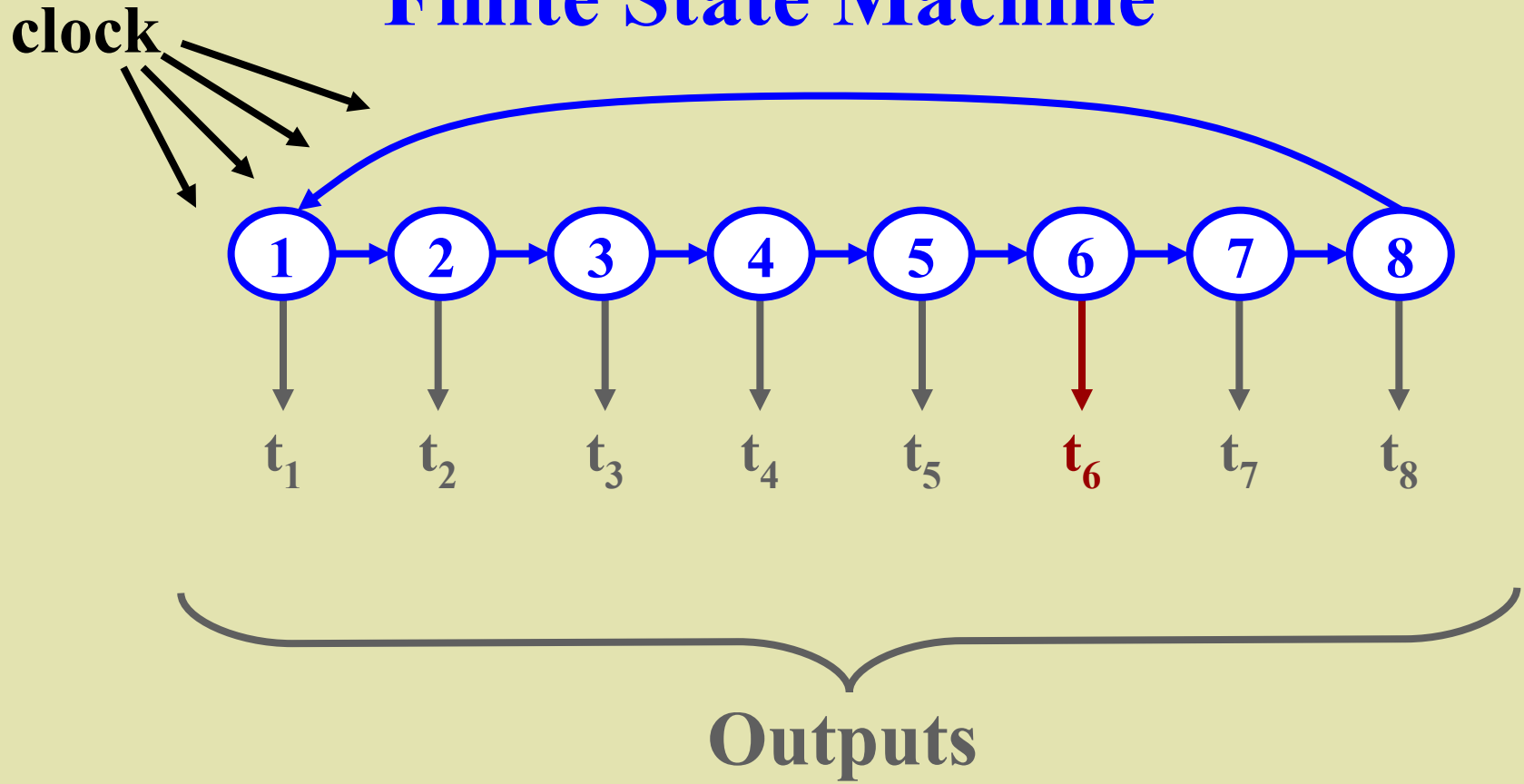


# Finite State Machine

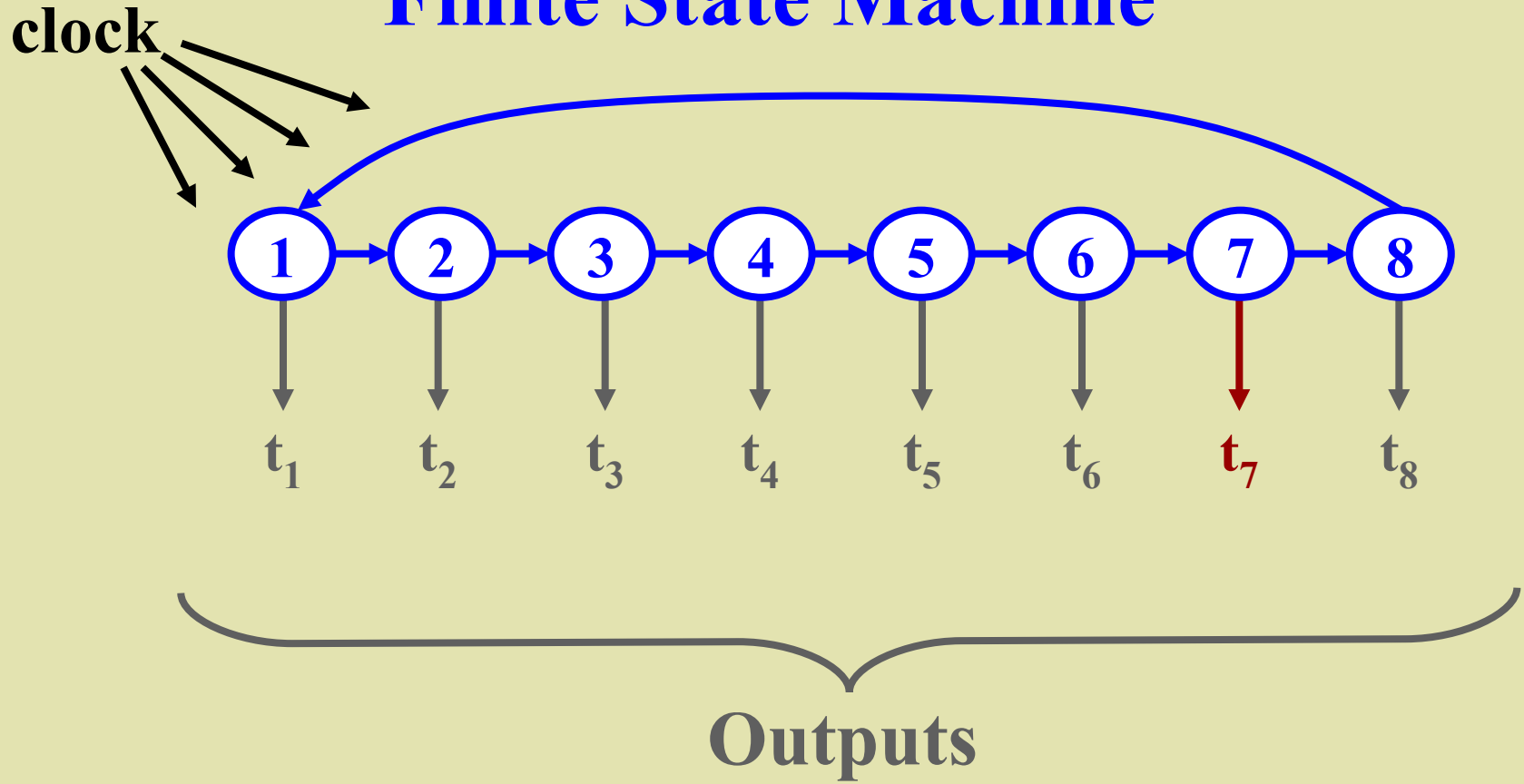




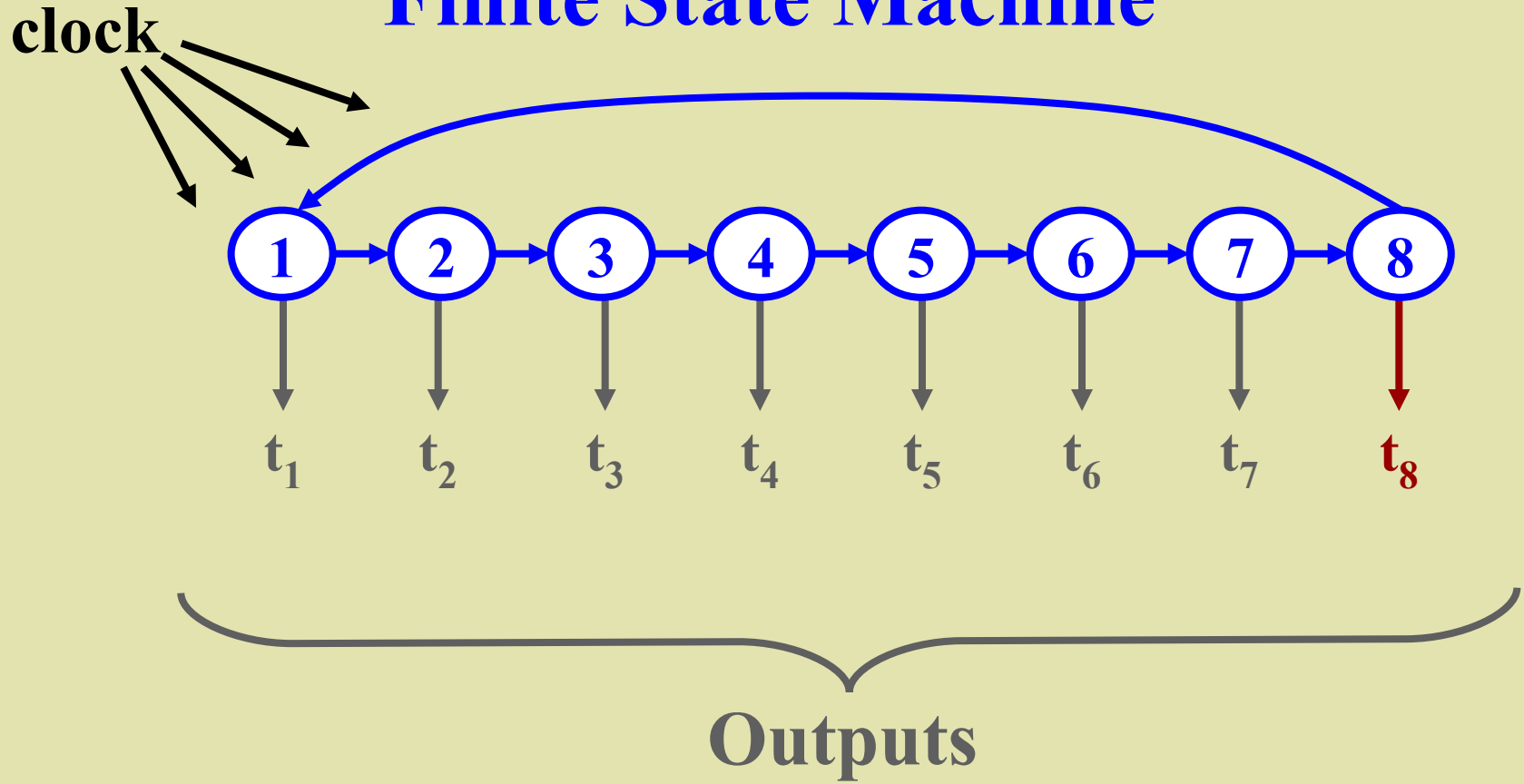
# Finite State Machine



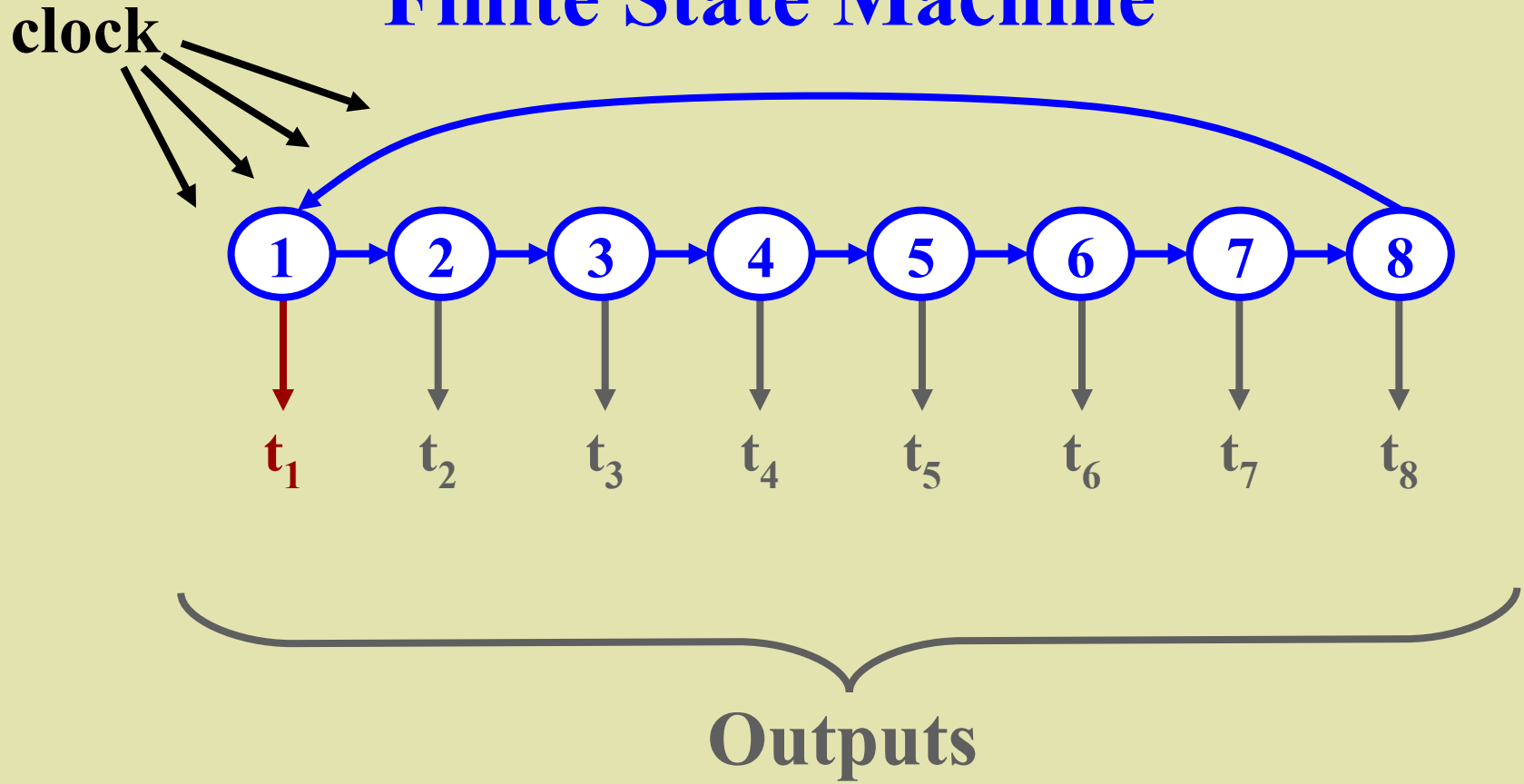
# Finite State Machine



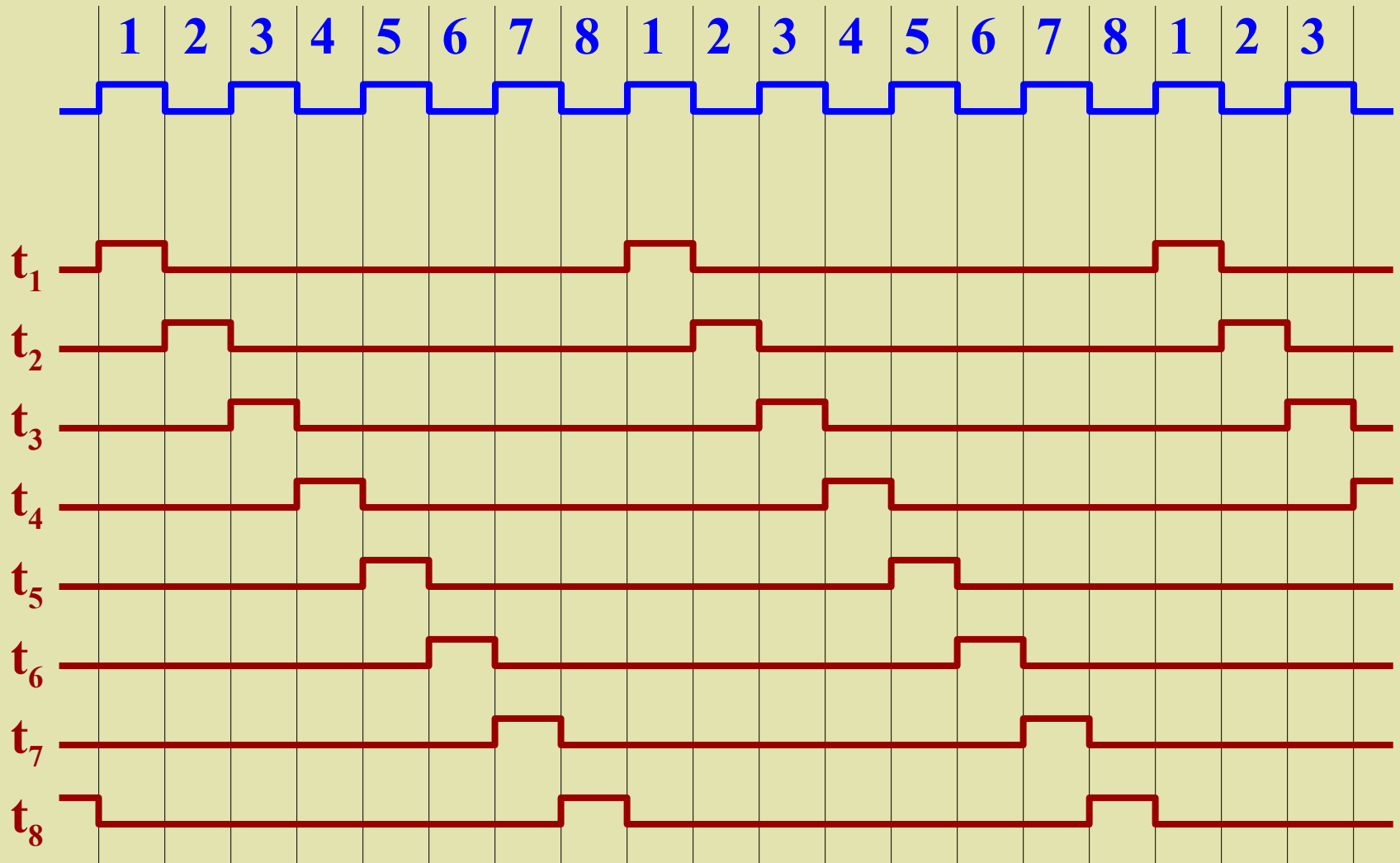
# Finite State Machine



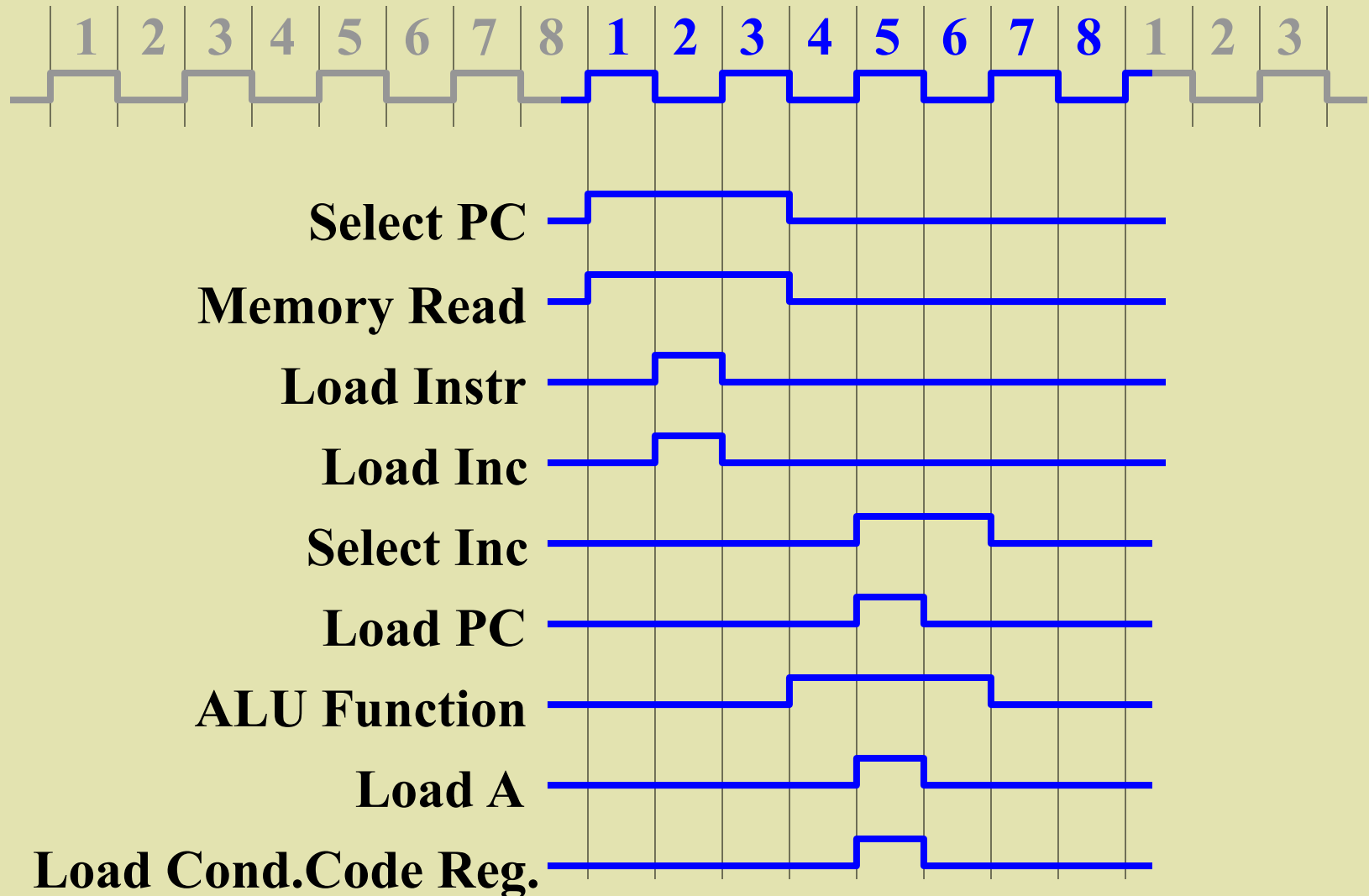
# Finite State Machine



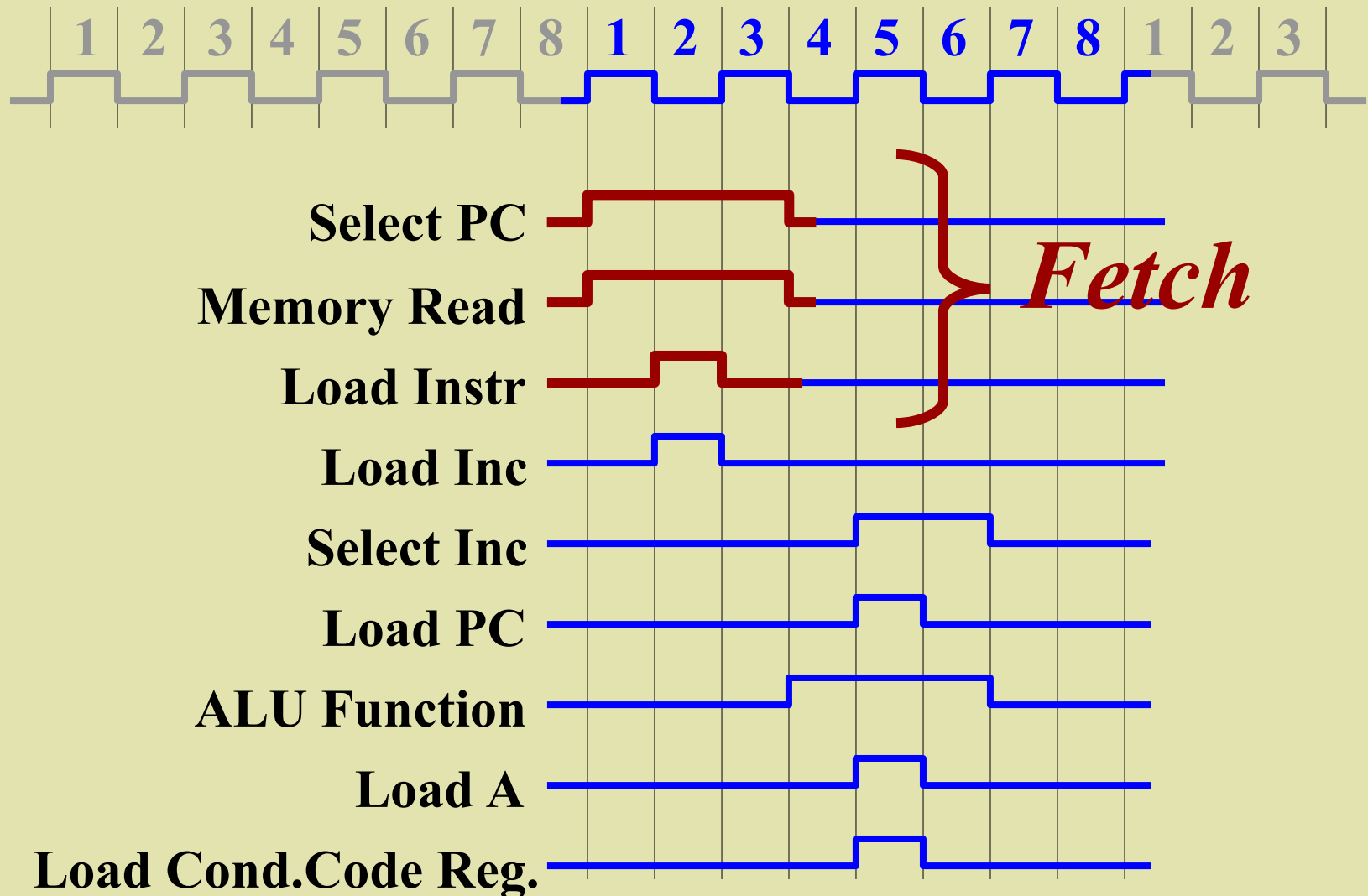
# Output from FSA



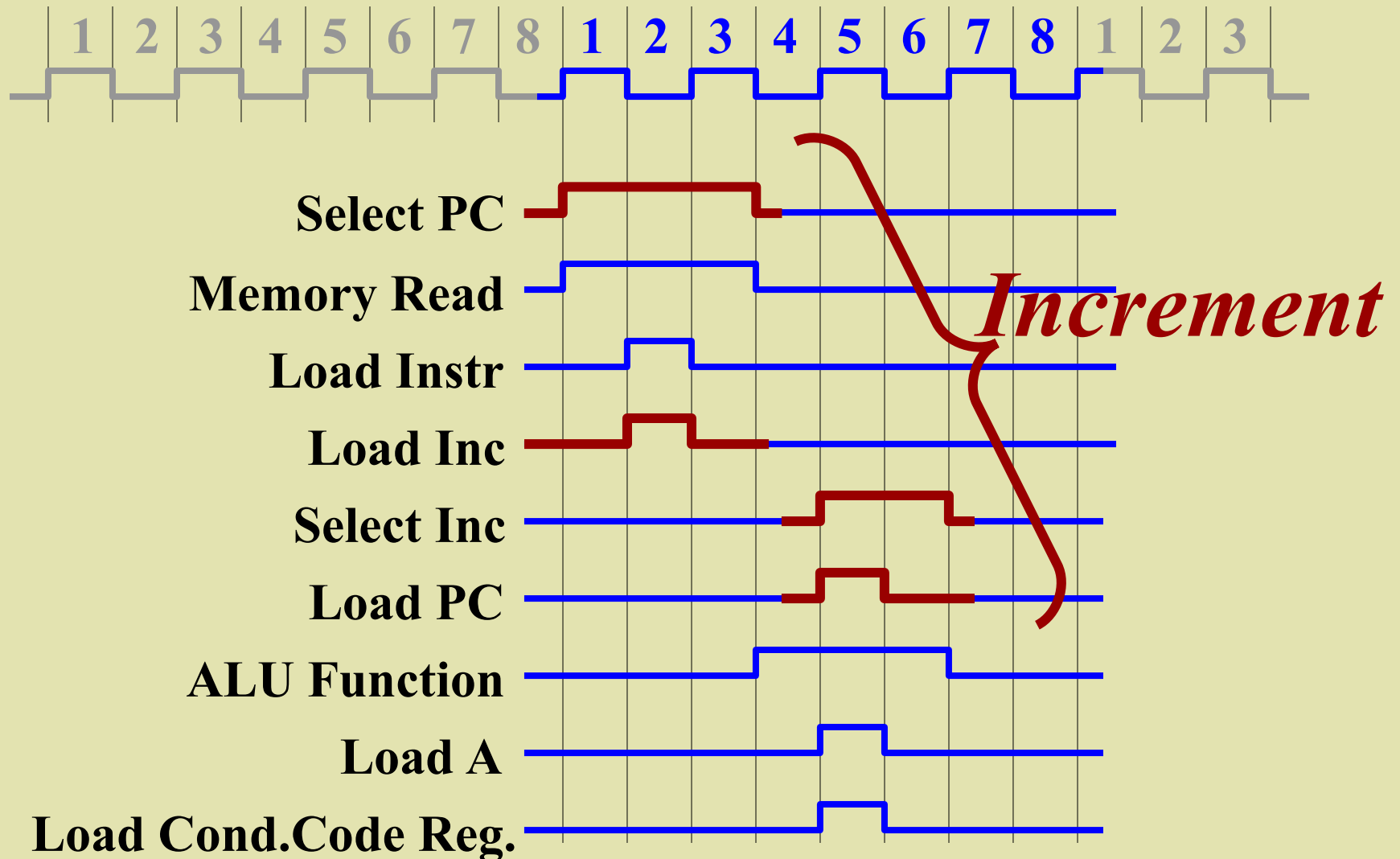
# Instruction Timing - ALU Instruction



# Instruction Timing - ALU Instruction

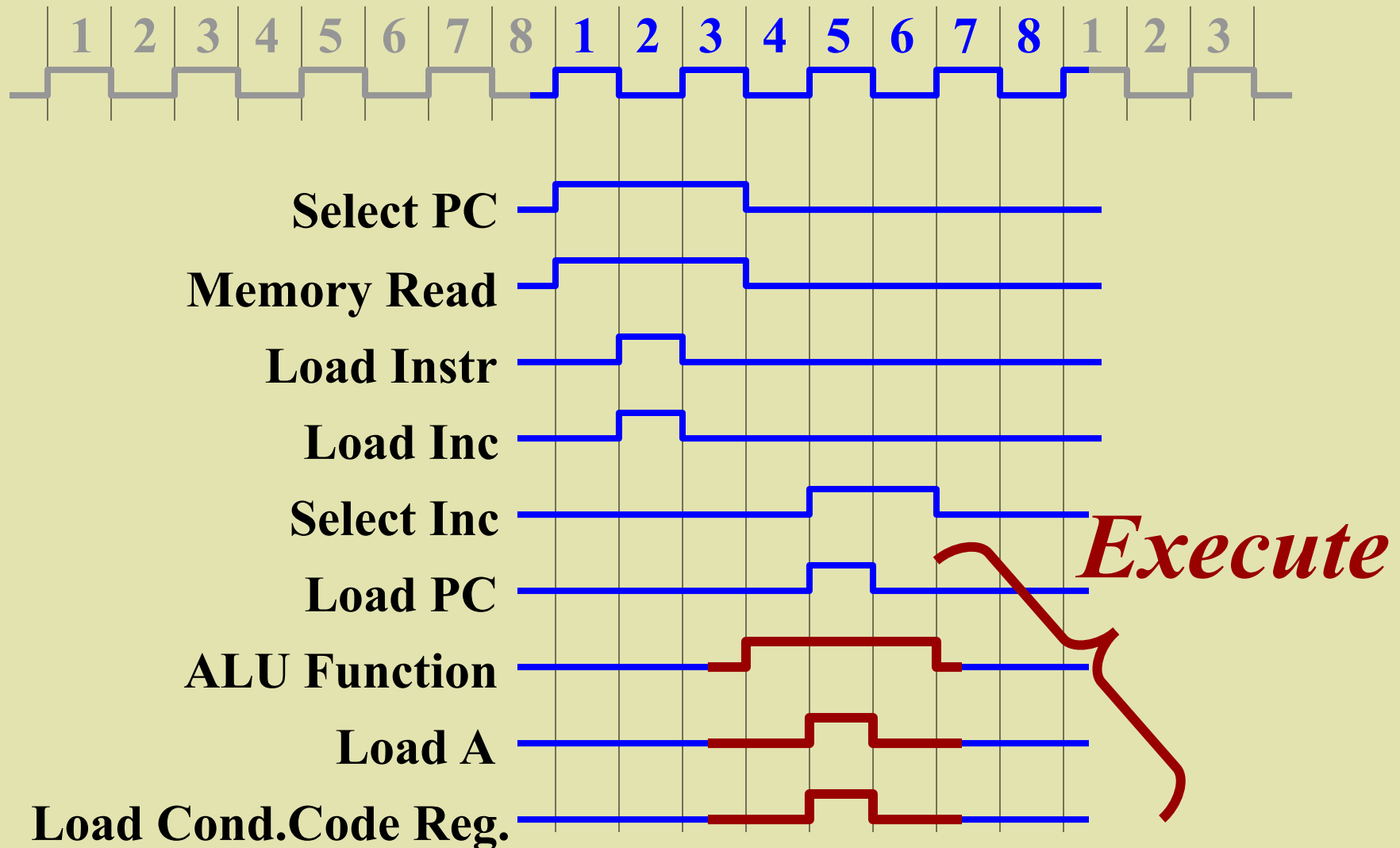


# Instruction Timing - ALU Instruction

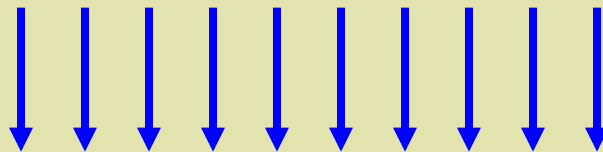
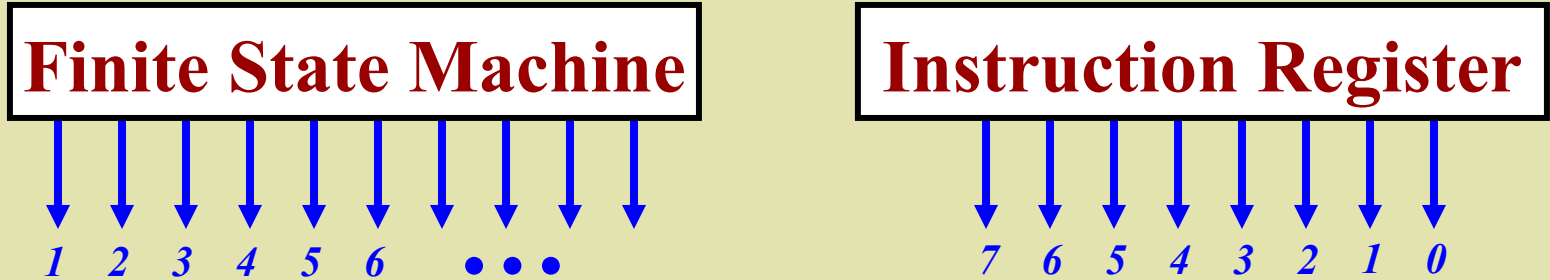




# Instruction Timing - ALU Instruction

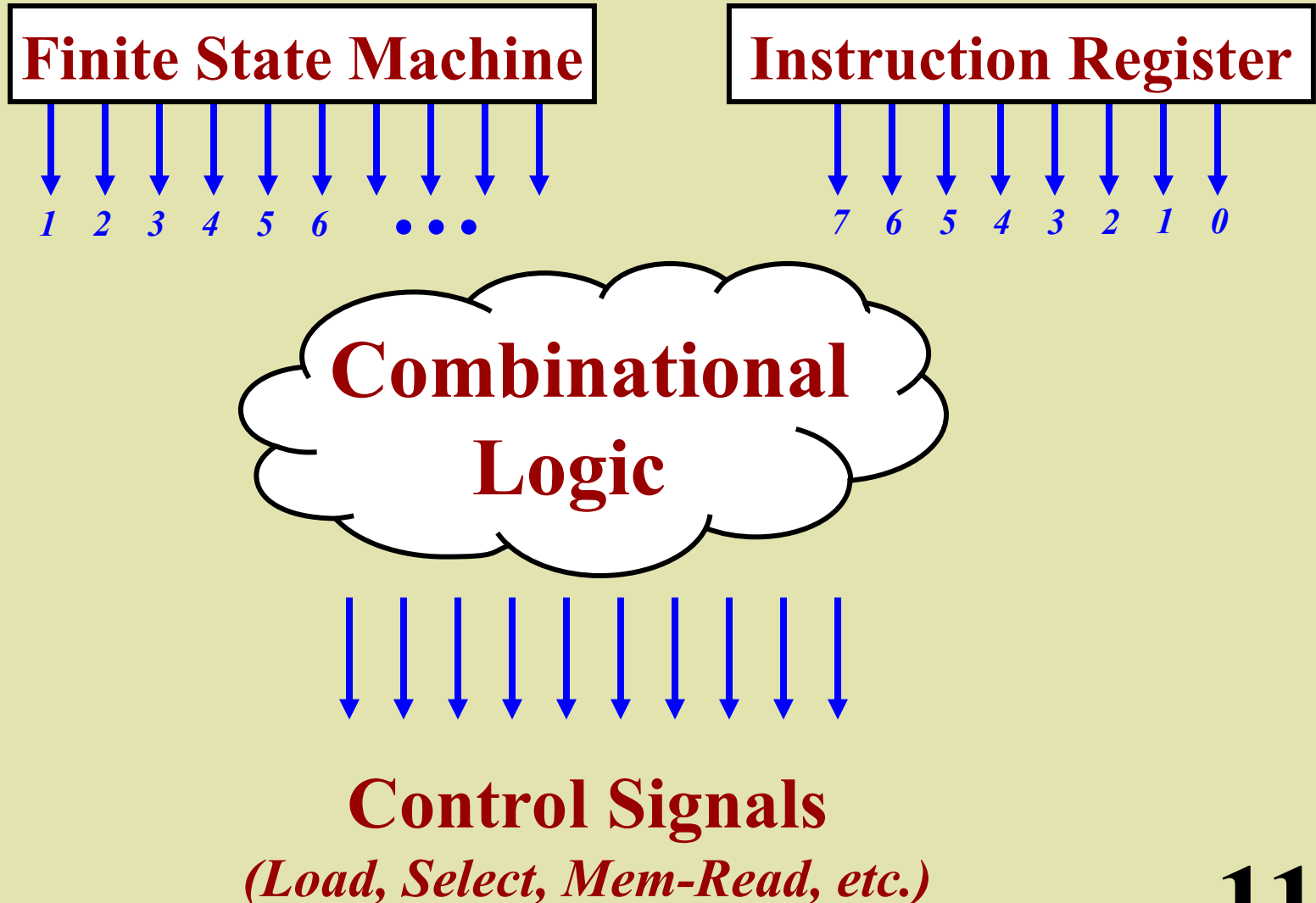


# Instruction Decoding

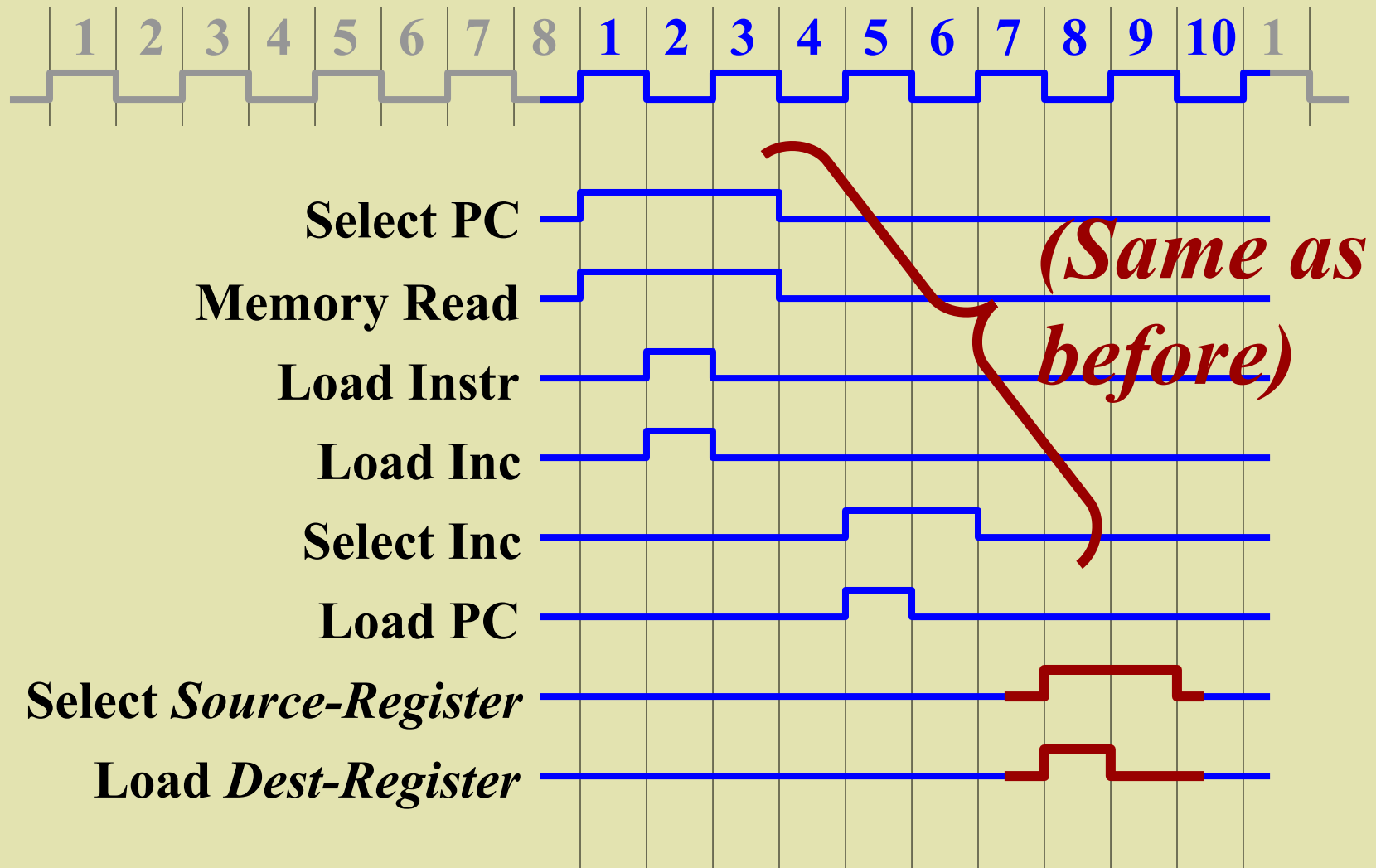


**Control Signals**  
*(Load, Select, Mem-Read, etc.)*

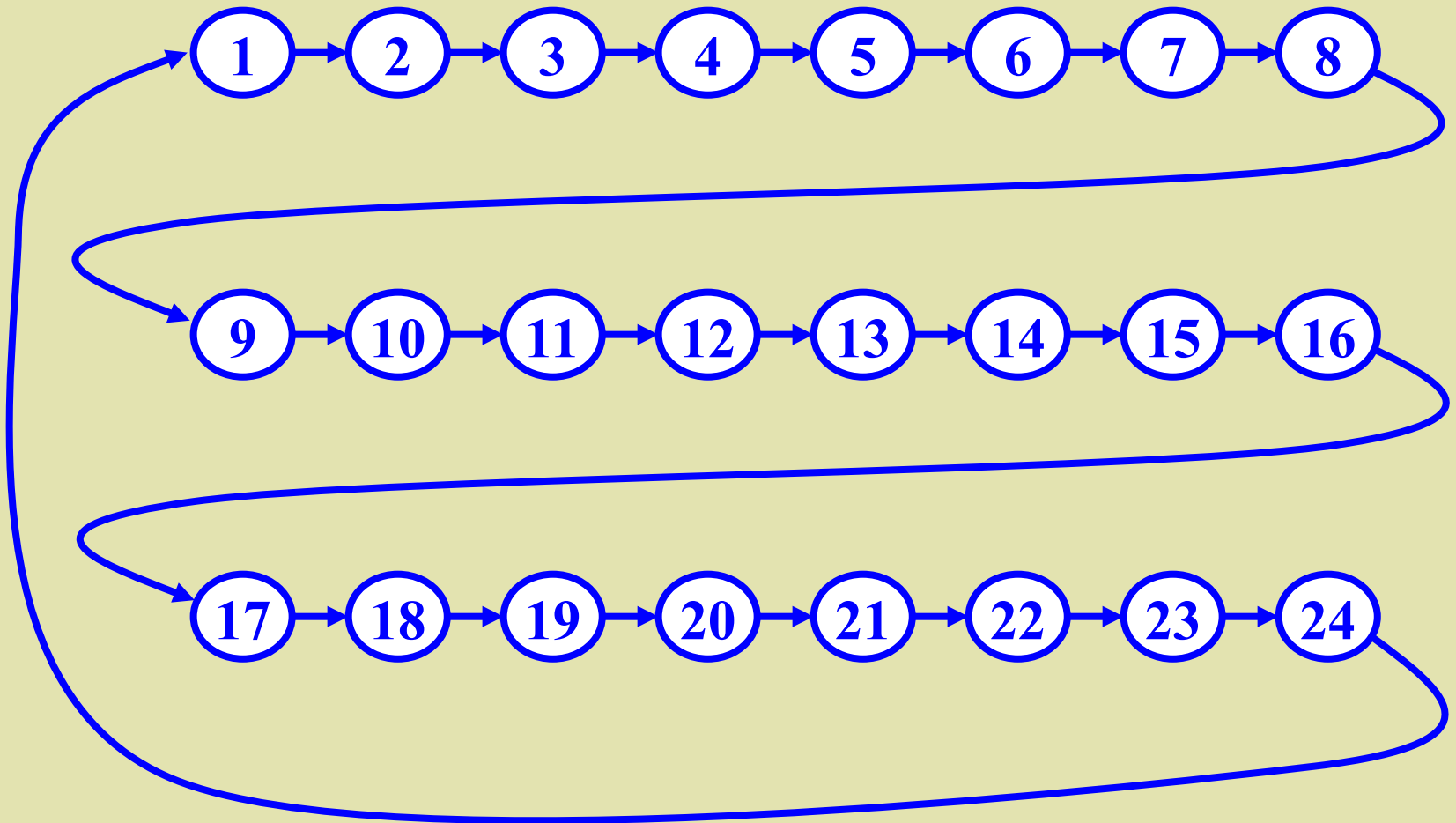
# Instruction Decoding



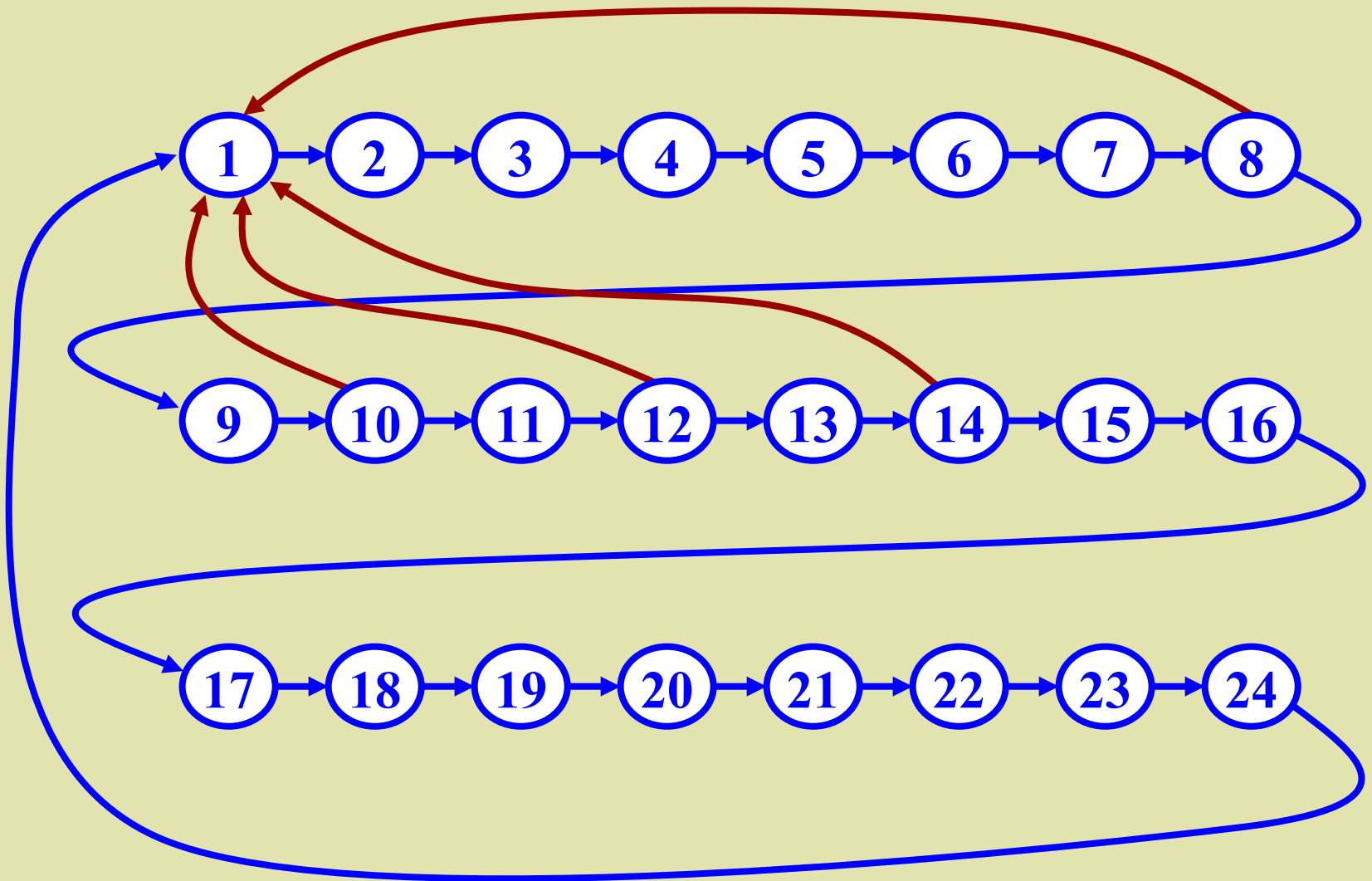
# Instruction Timing - 16-bit Move



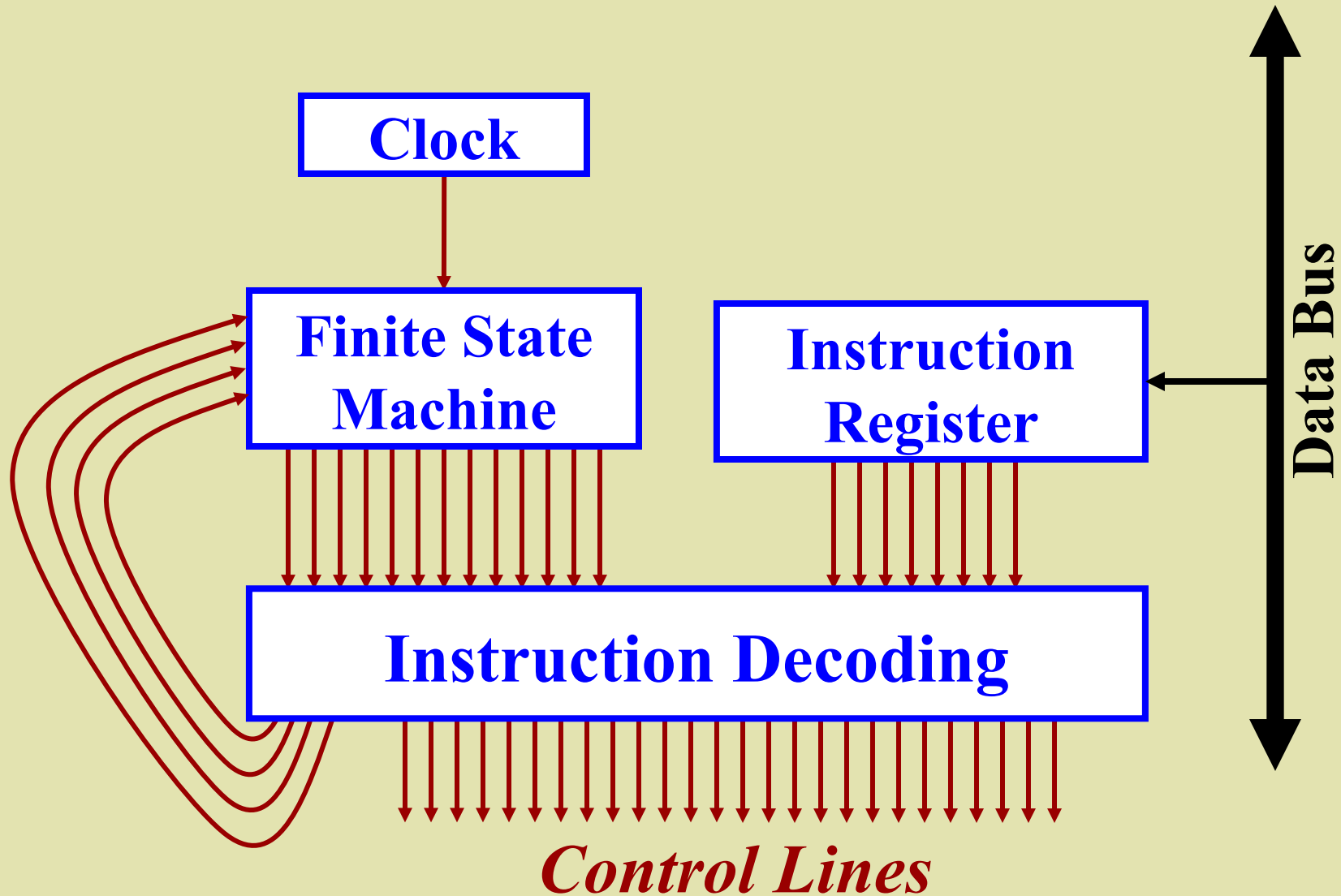
# Finite State Machine



# Finite State Machine

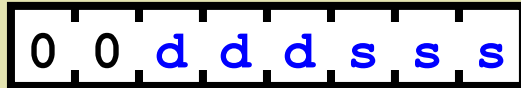


# Instruction Decoding and Control



# The Instruction Set (1)

## Move



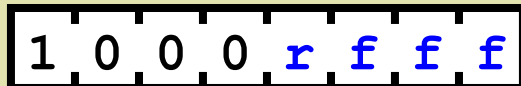
**ddd** = destination register

**sss** = source register

(A, B, C, D, M<sub>1</sub>, M<sub>2</sub>, X or Y)

---

## ALU



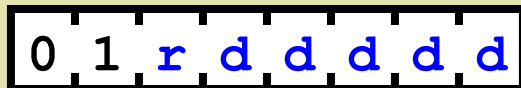
**r** = destination register (A or D)

**fff** = function code

(add, inc, and, or, xor, not, shl)

---

## Load Immediate

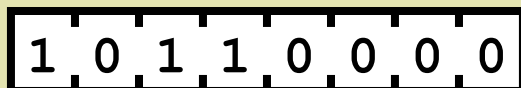


**r** = destination register (A or B)

**dddddd** = value (-16..15)

---

## 16-bit Increment

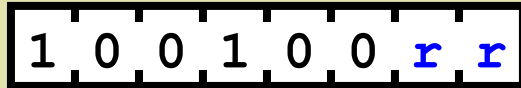


$XY \leftarrow XY + 1$



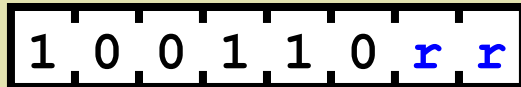
# The Instruction Set (2)

## Load



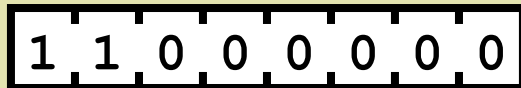
**rr** = destination register (A, B, C, D)  
**reg**  $\leftarrow$  [M]

## Store



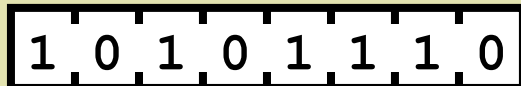
**rr** = source register (A, B, C, D)  
[M]  $\leftarrow$  reg

## Load 16-bit Immediate



Load the immediate value into M (i.e., M<sub>1</sub> and M<sub>2</sub>)

## Halt



# The Instruction Set (3)

## Goto

1 1 1 0 0 1 1 0

a a a a a a a a

a a a a a a a a

Branch to the given address

---

## Call

1 1 1 0 0 1 1 1

a a a a a a a a

a a a a a a a a

Branch to the given address  
Save return location in XY register

---

## Return / Branch Indirect

1 0 1 0 1 0 1 0

PC  $\leftarrow$  XY

---

## 16-Bit Move

1 0 1 0 d s s 0

d = destination register (PC or XY)

ss = source register (M, XY or J)

# The Instruction Set (4)

## Branch If Negative

1 1 1 1 0 0 0 0

a a a a a a a a

a a a a a a a a

Branch to the given address if  $S = 1$

---

## Branch If Carry

1 1 1 0 1 0 0 0

a a a a a a a a

a a a a a a a a

Branch to the given address if  $Cy = 1$

---

## Branch If Zero

1 1 1 0 0 1 0 0

a a a a a a a a

a a a a a a a a

Branch to the given address if  $Z = 1$

---

## Branch If Not Zero

1 1 1 0 0 0 1 0

a a a a a a a a

a a a a a a a a

Branch to the given address if  $Z = 0$

# An Example Program

<u>address</u>	<u>instr</u>	<u>assembly</u>	<u>comment</u>
0000 0000	0011 1001	Y=B	Y ← B
0000 0001	0011 0110	X=0	X ← 0
0000 0010	1000 0101	A=¬B	If sign(Y)==1
0000 0011	1111 0000	BNEG Else	.
0000 0100	0000 0000	.	.
0000 0101	0000 0111	.	.
0000 0110	0011 0010	X=C	X ← C
		Else:	.
0000 0111	0101 1001	A=-7	D ← -7
0000 1000	0001 1000	D=A	.
		Loop:	Loop:
0000 1001	0000 1110	B=X	Shift X left (circular)
0000 1010	1000 0110	A=B<<1	.
0000 1011	0011 0000	X=A	.
0000 1100	0000 1111	B=Y	Shift Y left (circular)
0000 1101	1000 0110	A=B<<1	.
0000 1110	0011 1000	Y=A	.
0000 1111	0000 1111	B=Y	If sign(Y)==1
0001 0000	1000 0101	A=¬B	.
0001 0001	1111 0000	BNEG Else2	.
0001 0010	0000 0000	.	.
0001 0011	0001 0111	.	.
0001 0100	0000 1110	B=X	X ← X + C
0001 0101	1000 0000	A=B+C	.
0001 0110	0011 0000	X=A	.
		Else2:	.
0001 0111	0000 1011	B=D	D ← D + 1
0001 1000	1000 1001	D=B+1	.
0001 1001	1110 0010	BNZ Loop	If D != 0 goto Loop
0001 1010	0000 0000	.	.
0001 1011	0000 1001	.	.
0001 1100	1010 1110	HALT	HALT