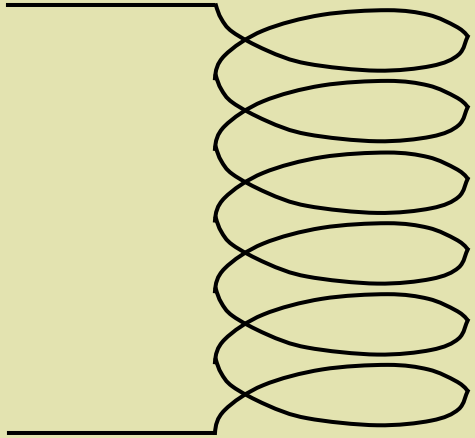
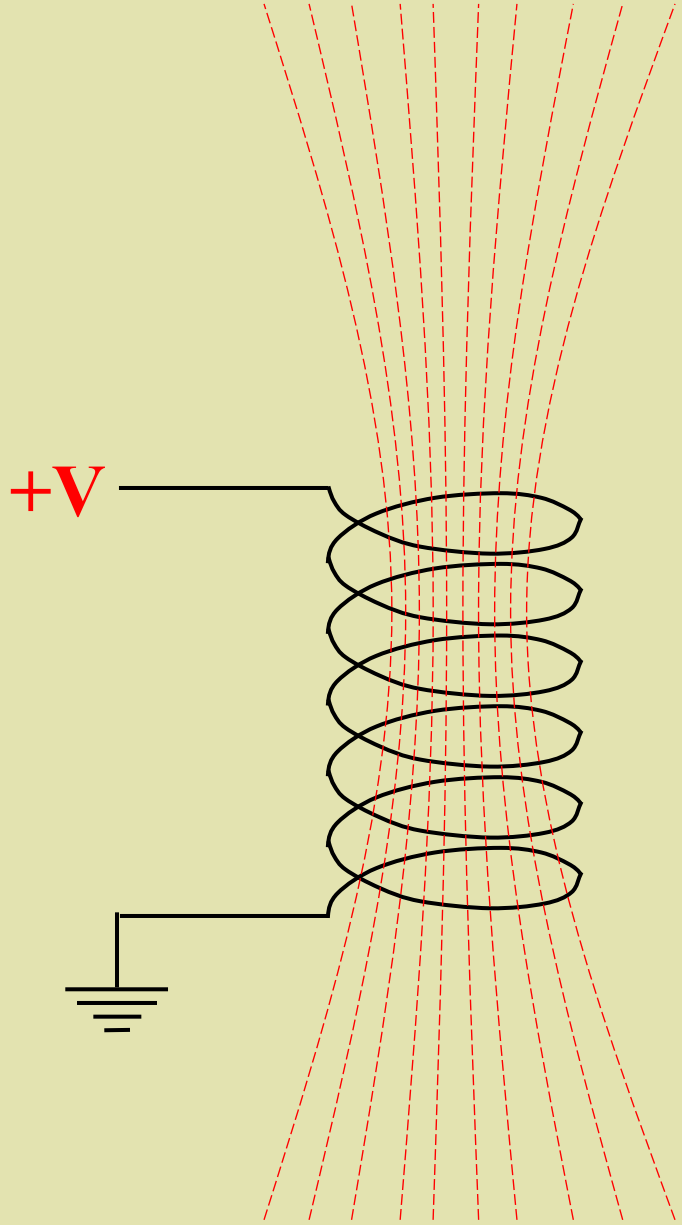
The Design of a Relay Computer

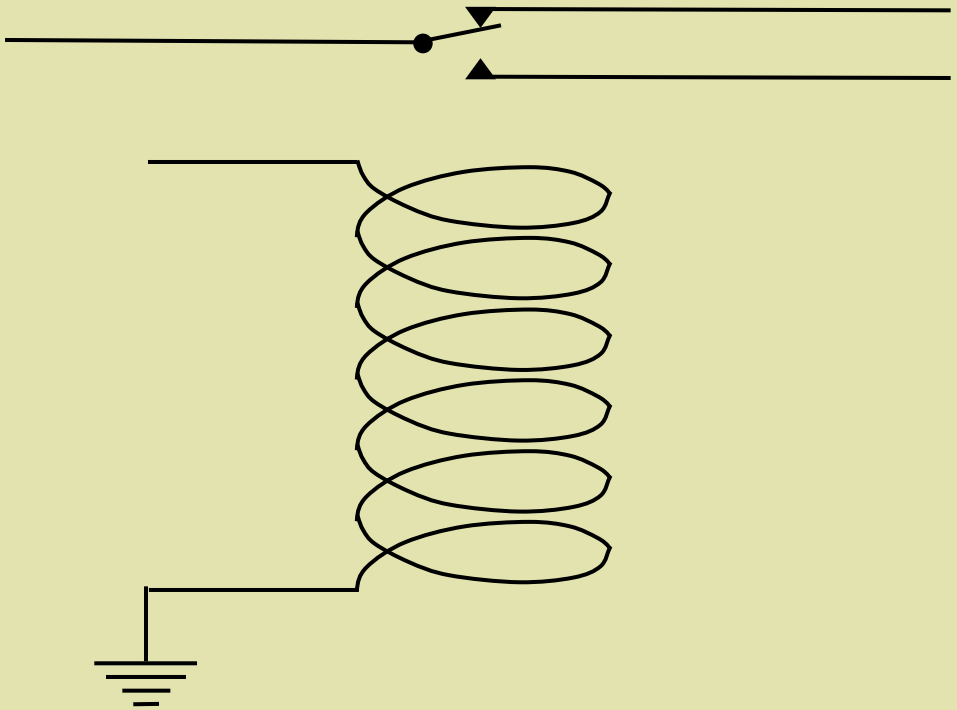
Harry Porter, Ph.D.
Portland State University

April 24, 2006

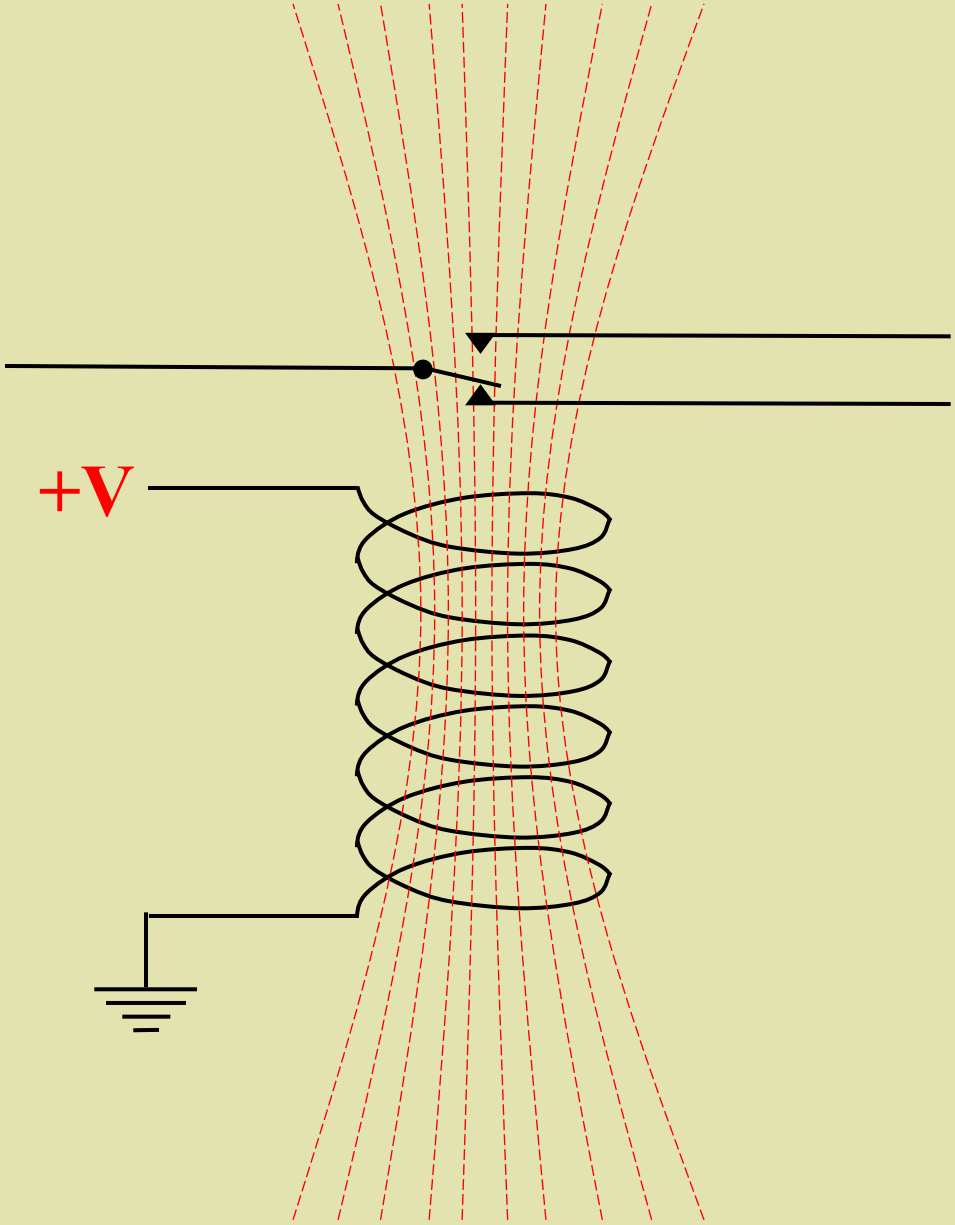



+V

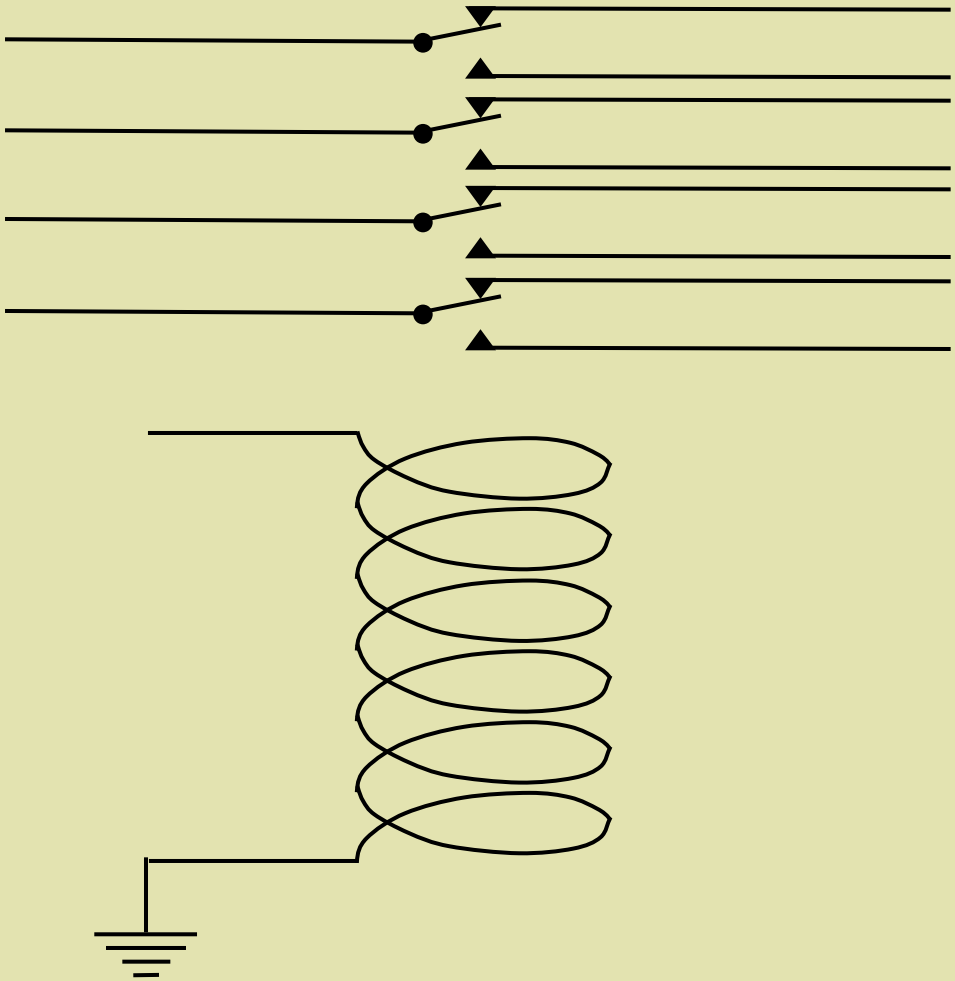
Double Throw Relay



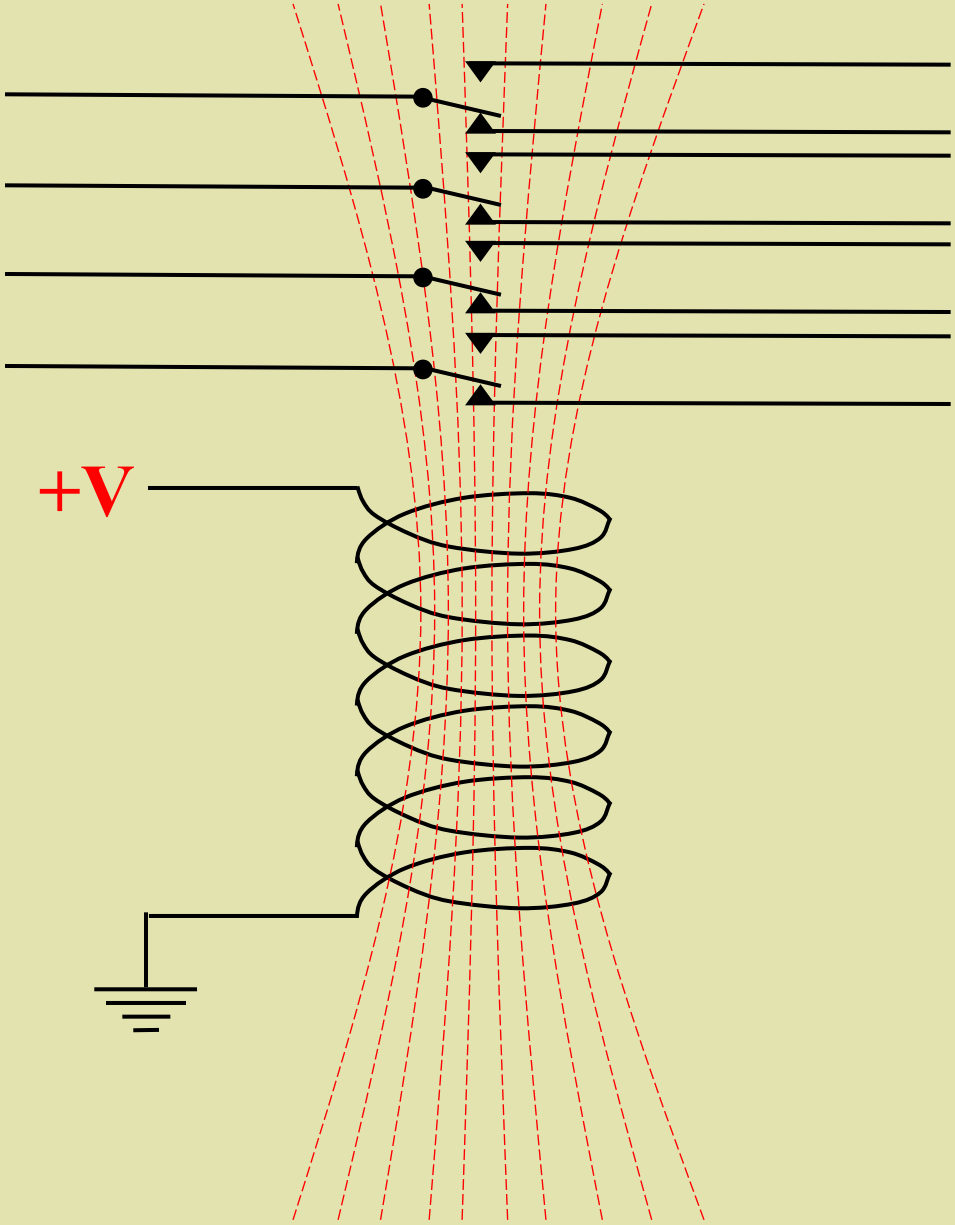
Double Throw Relay

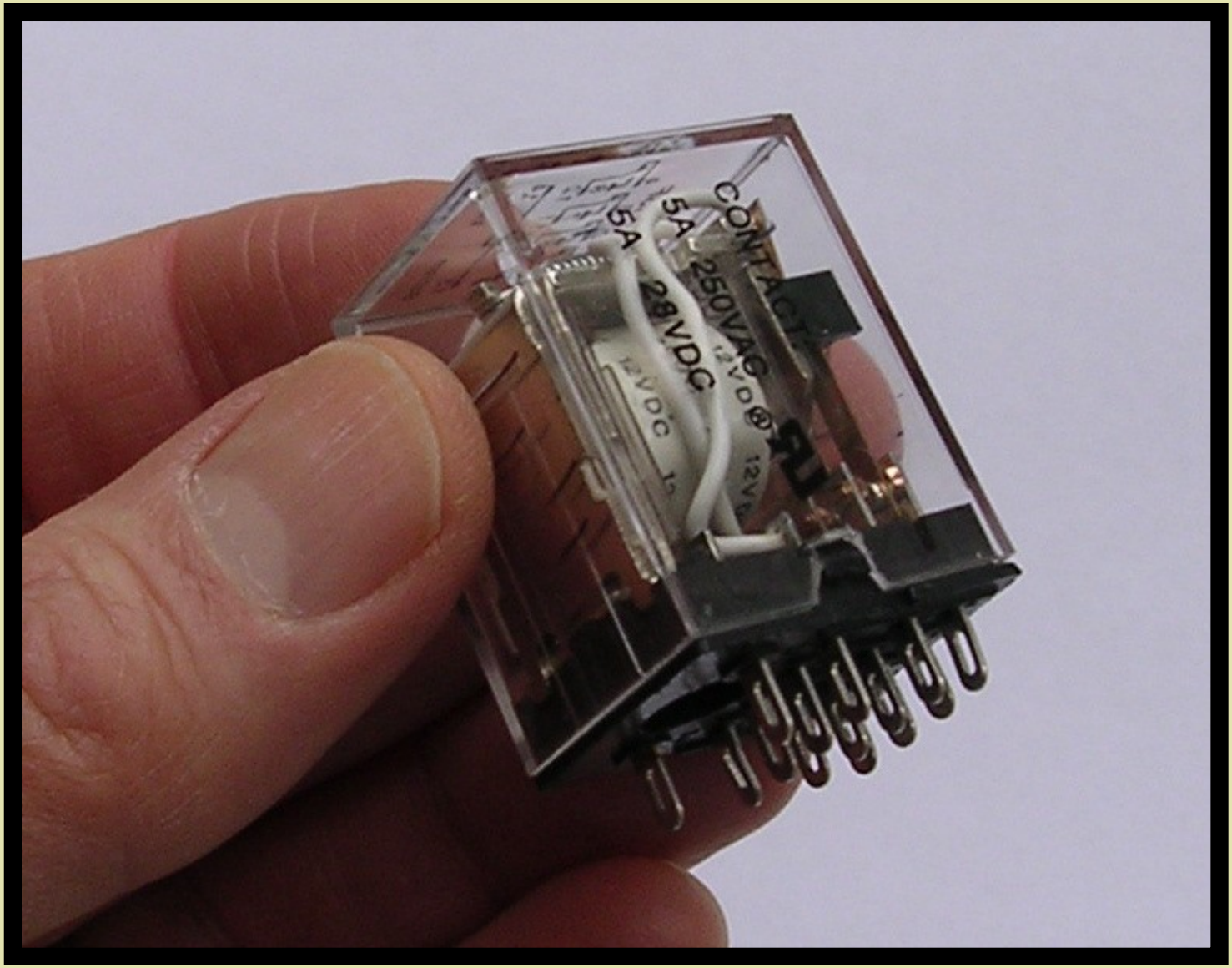


Four Pole, Double Throw Relay

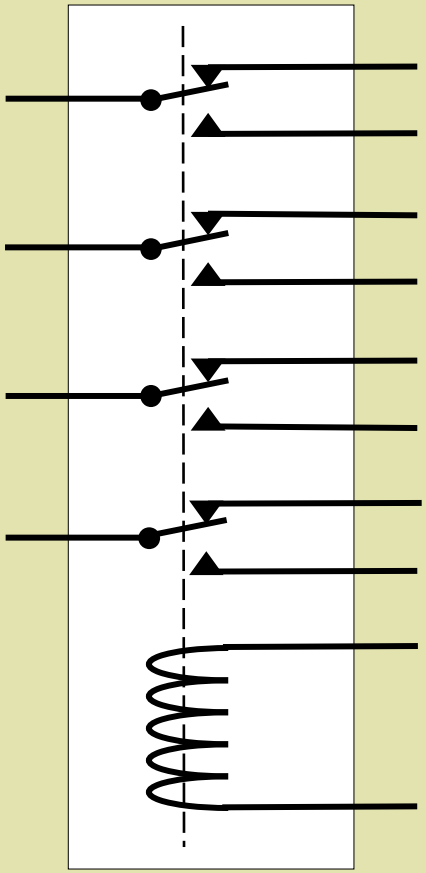


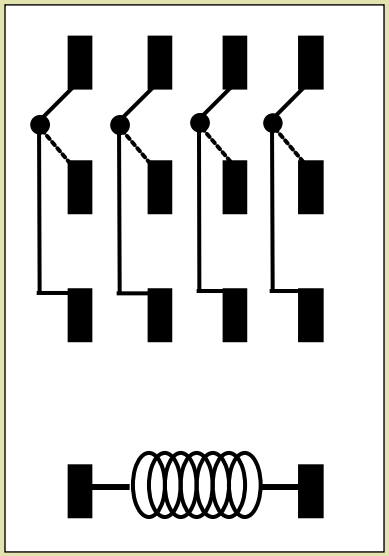
Four Pole, Double Throw Relay

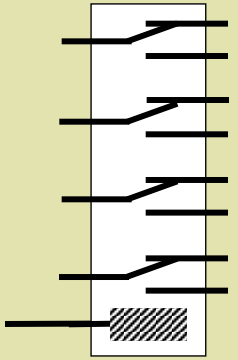



CONTACT
5A 250VAC
5A 28VDC
12VDC

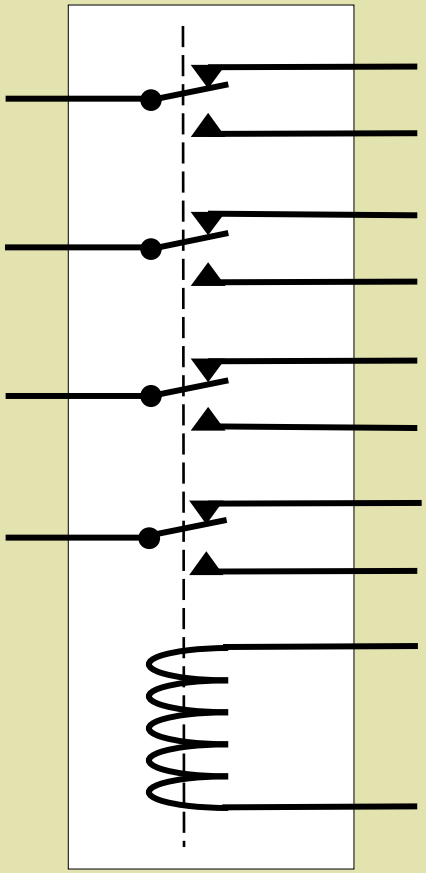
Schematic Diagrams

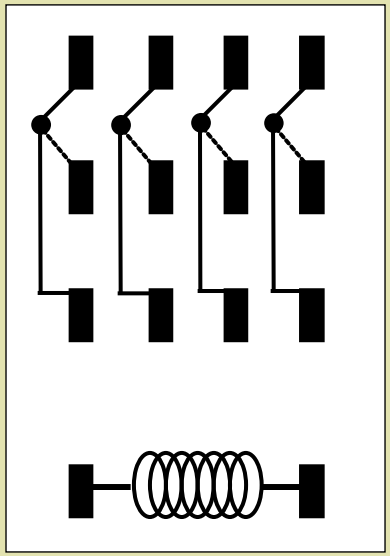


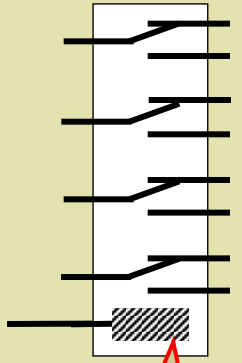




Schematic Diagrams

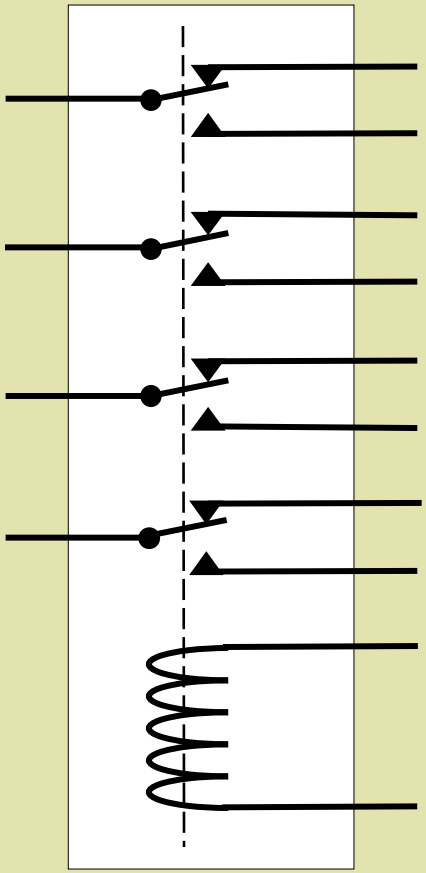


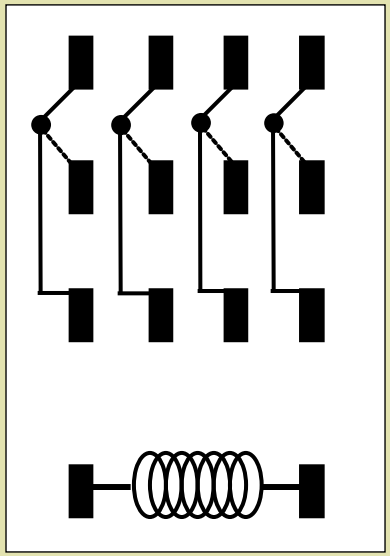


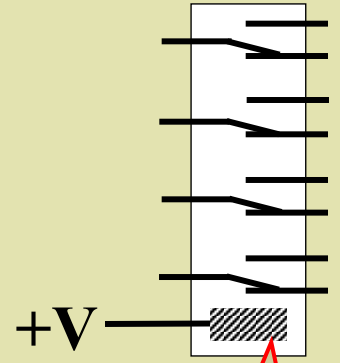


Assume other terminal is connected to ground

Schematic Diagrams

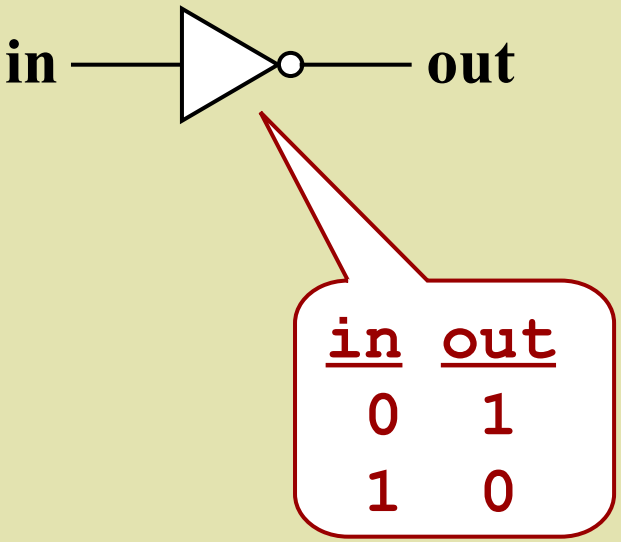


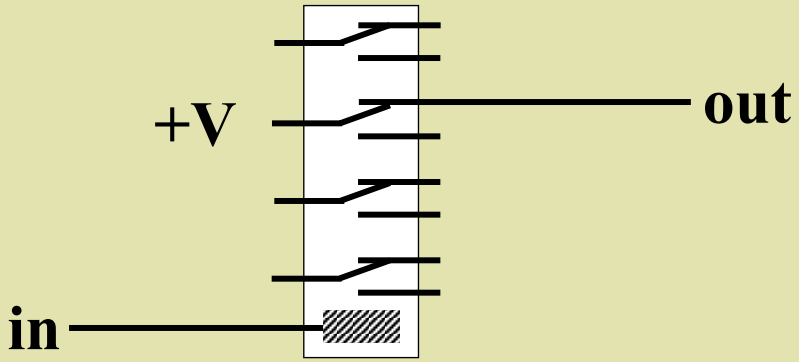




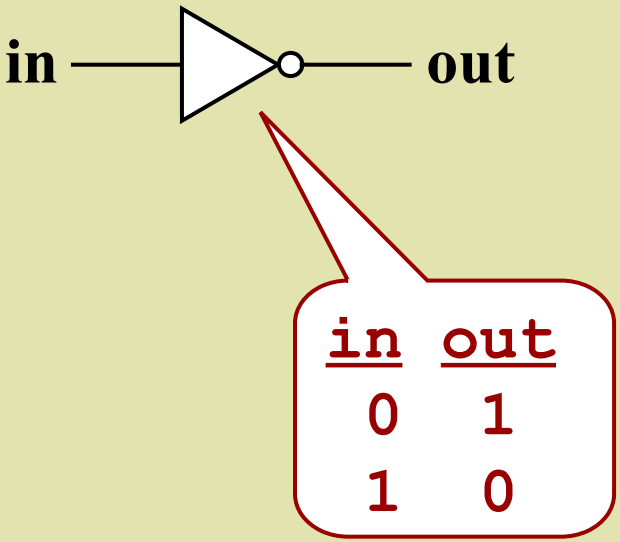
Assume other terminal is connected to ground

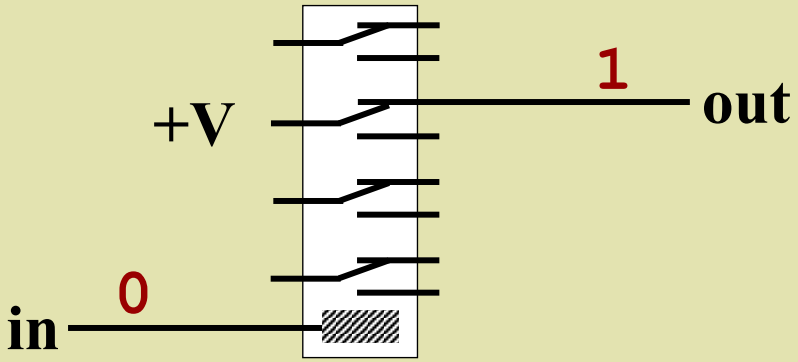
The "NOT" Circuit





The "NOT" Circuit



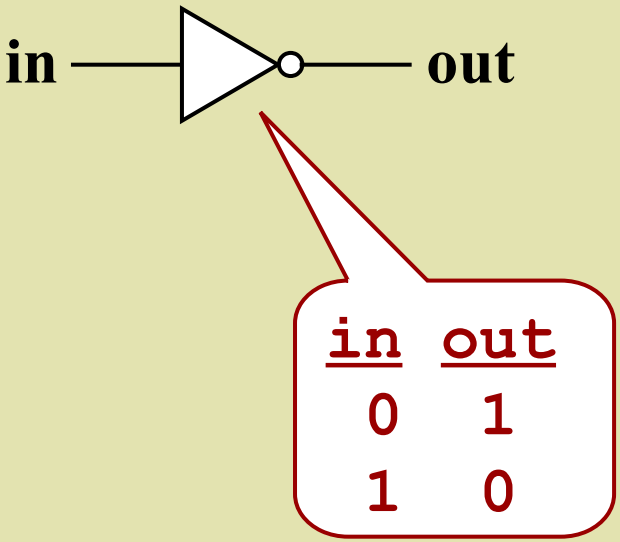


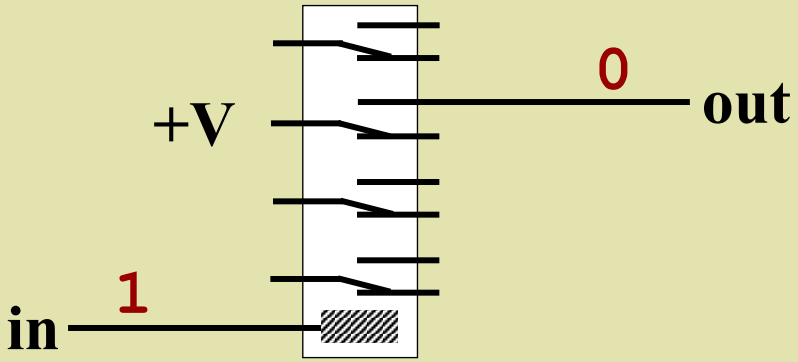
Convention:

"1" = +12V

"0" = not connected

The "NOT" Circuit



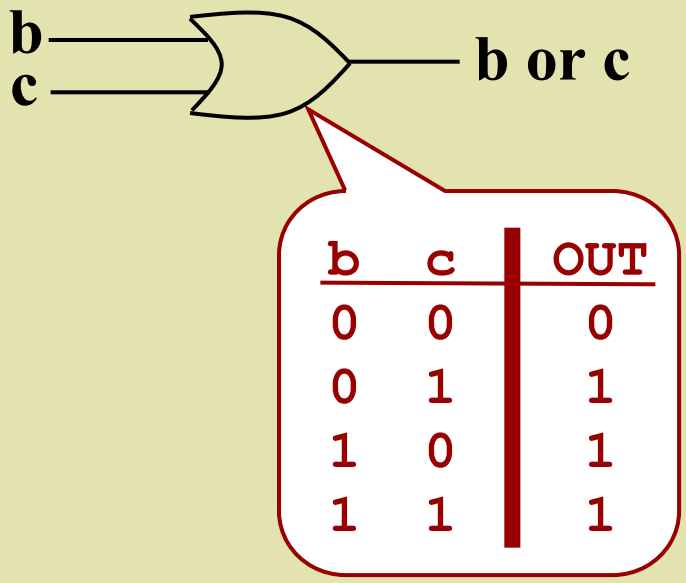


Convention:

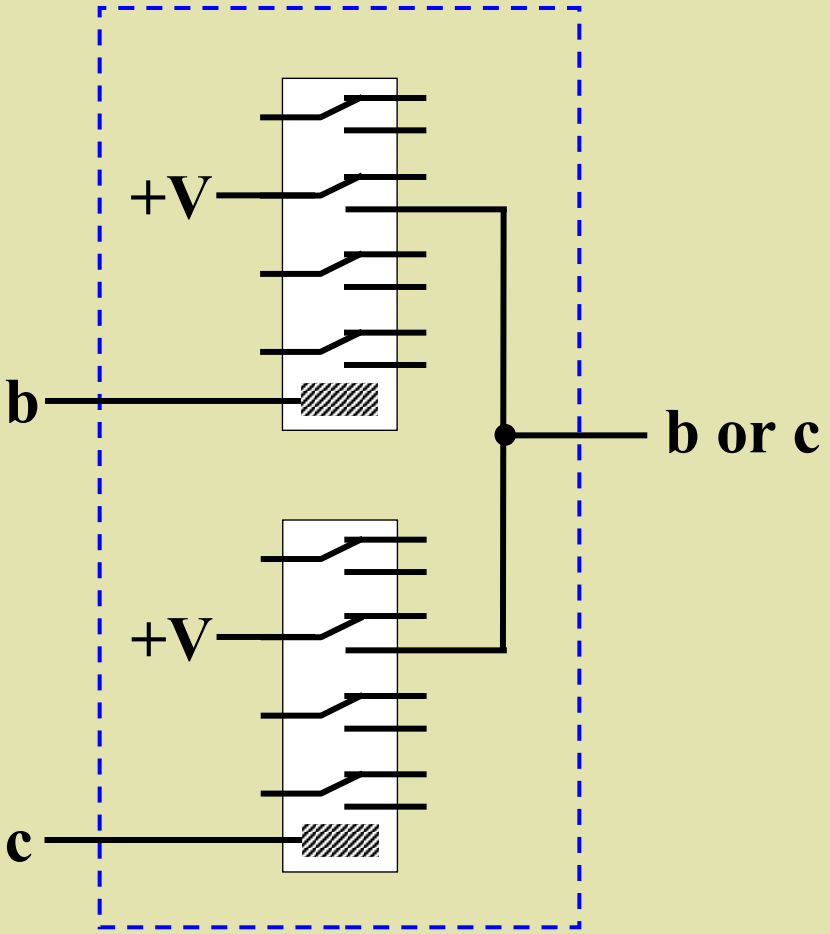
"1" = +12V

"0" = not connected

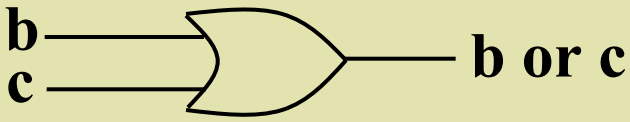
The "OR" Circuit

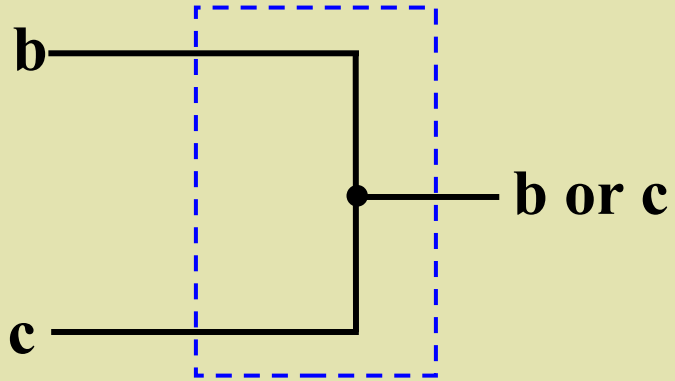


b	c	OUT
0	0	0
0	1	1
1	0	1
1	1	1

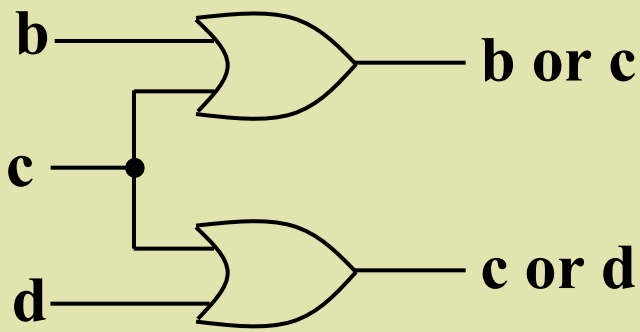


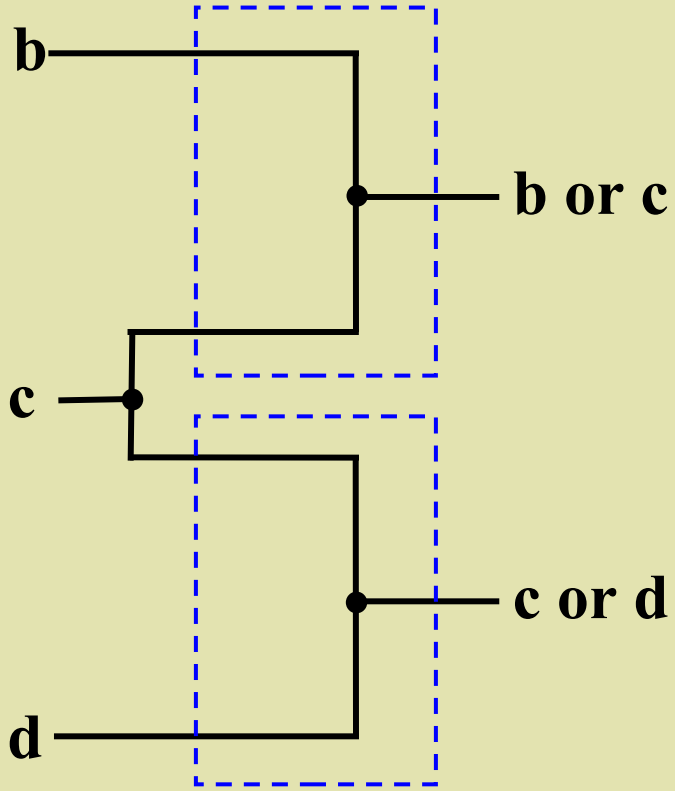
An Optimization?



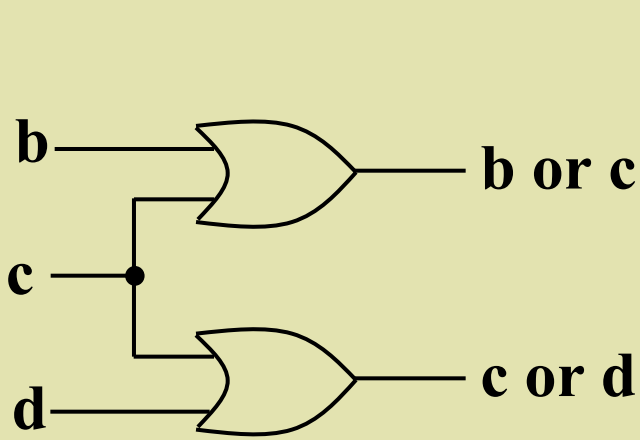


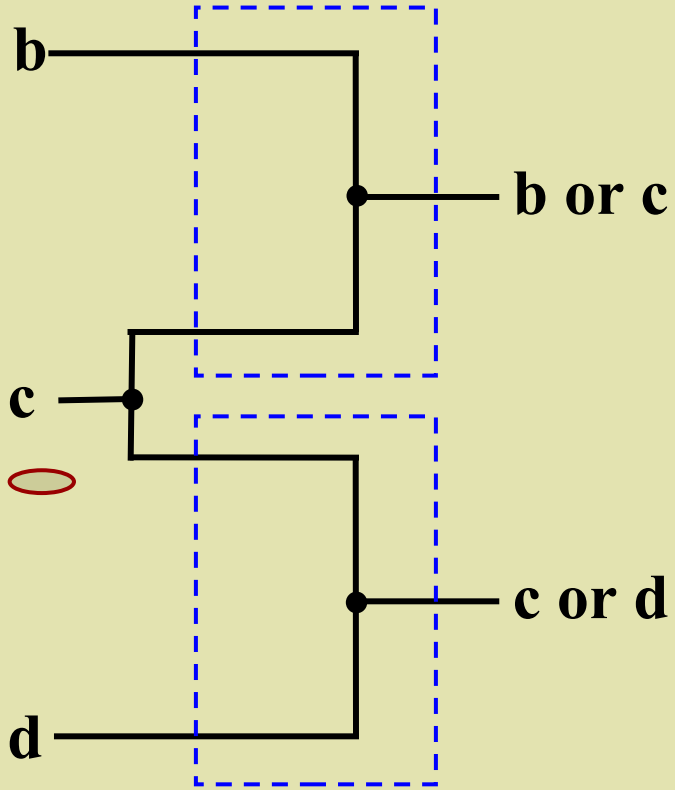
An Optimization?





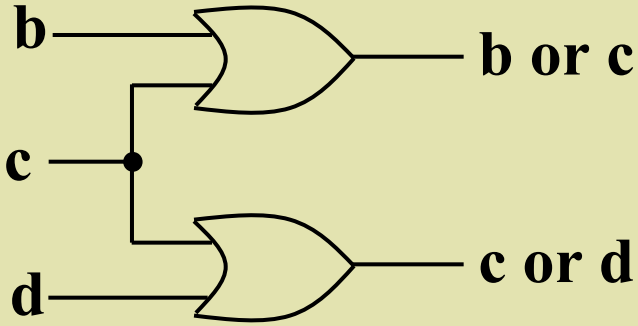
An Optimization?

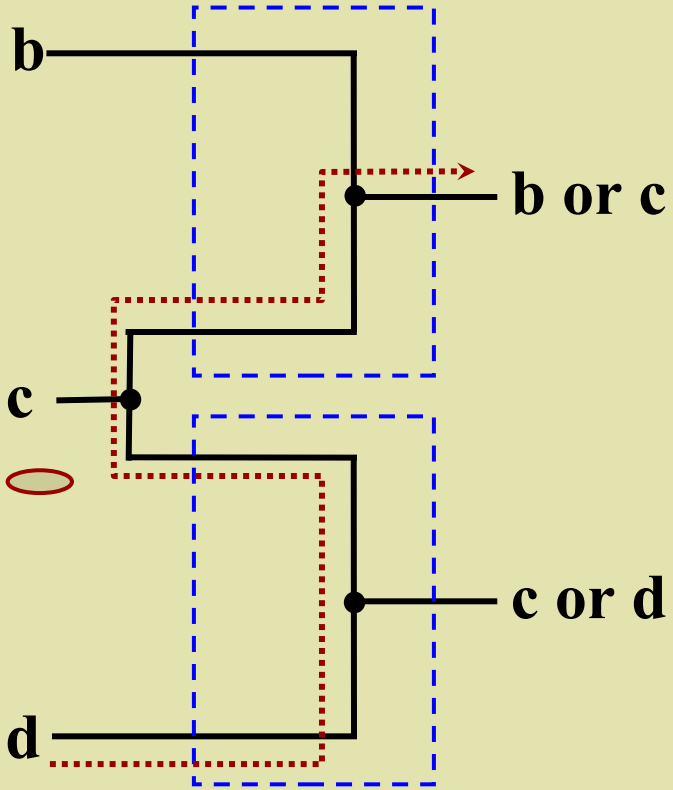




Whoops!

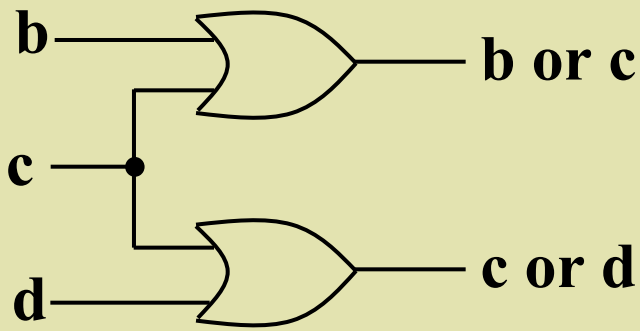
An Optimization?

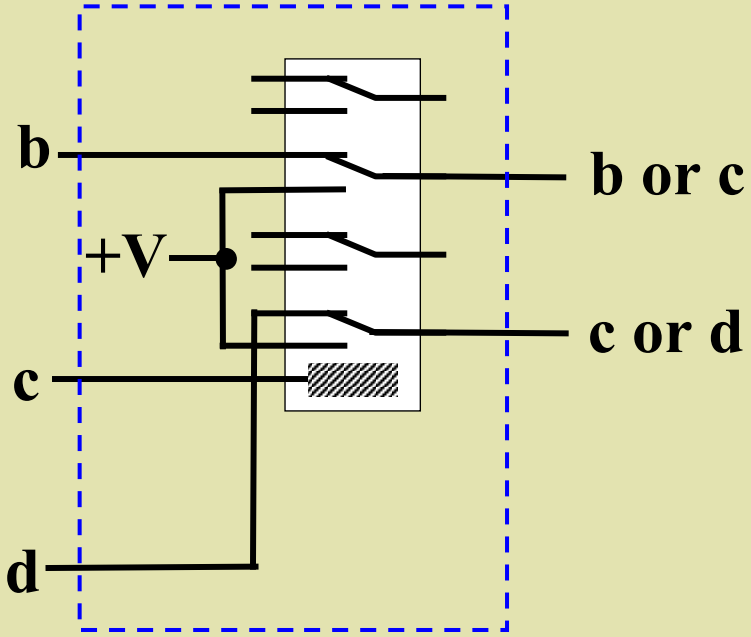




Whoops!

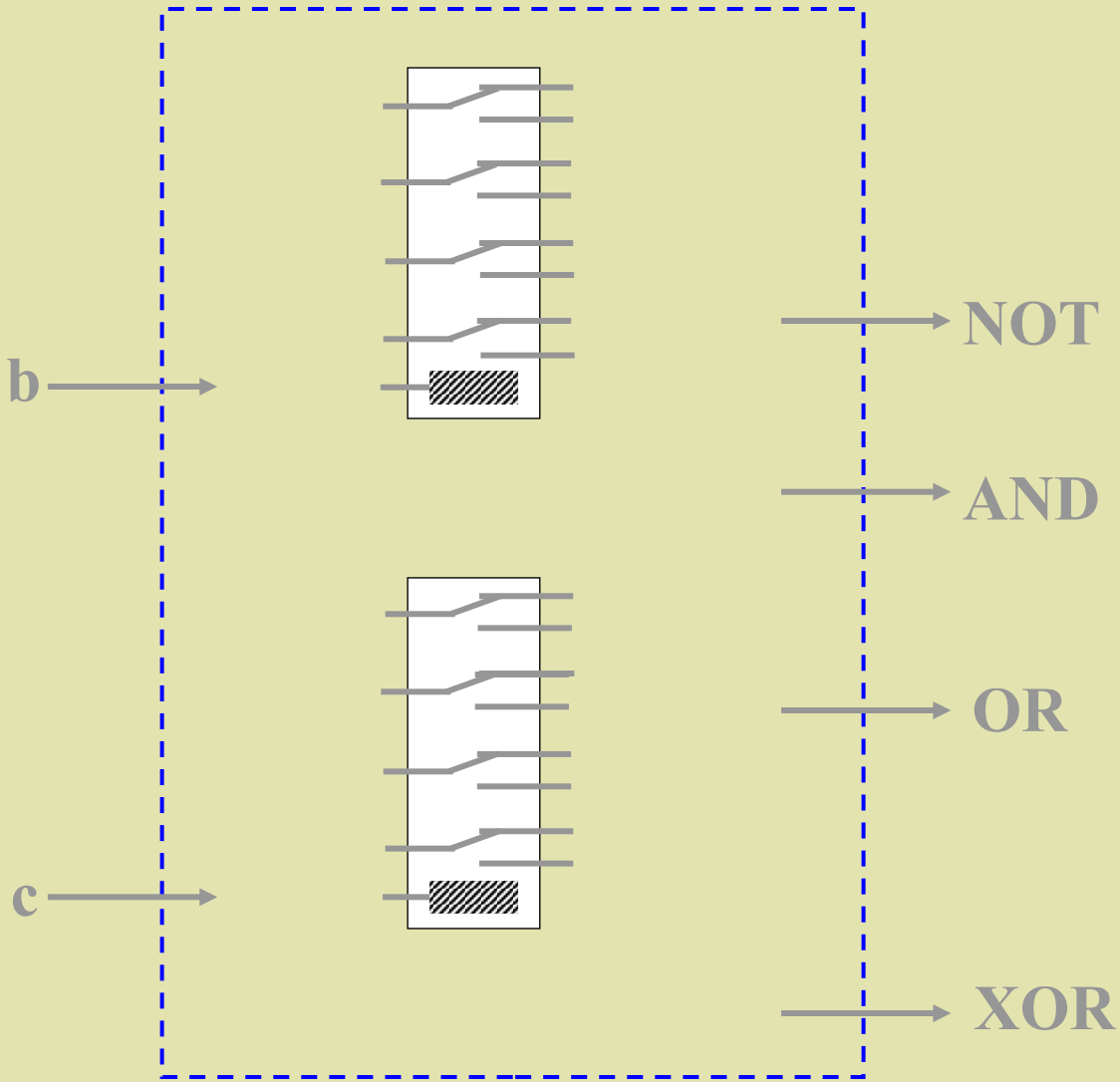
An Optimization?





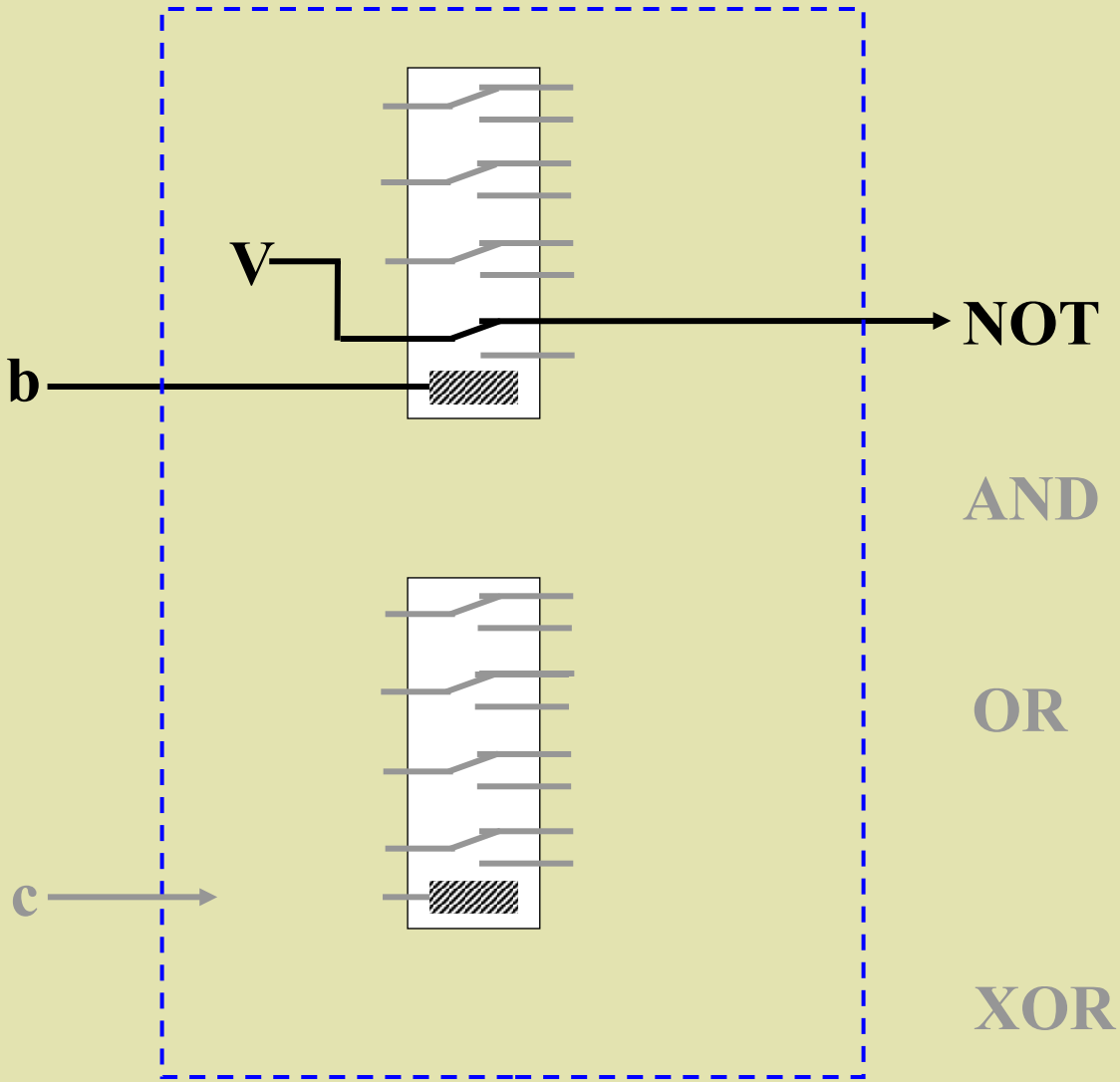
1-Bit Logic Circuit

b	c	NOT	AND	OR	XOR
0	0	1	0	0	0
0	1	1	0	1	1
1	0	0	0	1	1
1	1	0	1	1	0



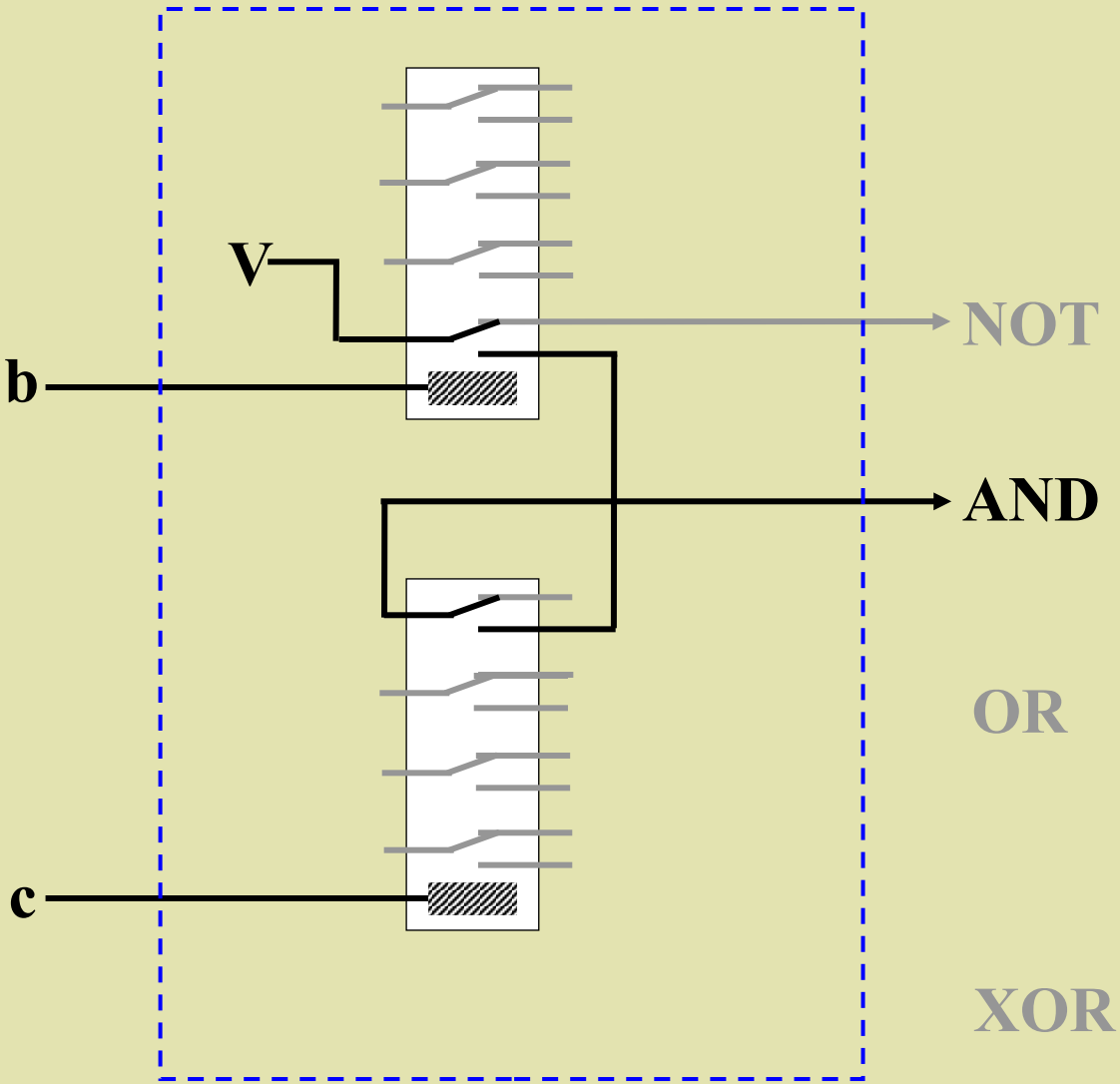
1-Bit Logic Circuit

b	c	NOT	AND	OR	XOR
0	0	1	0	0	0
0	1	1	0	1	1
1	0	0	0	1	1
1	1	0	1	1	0



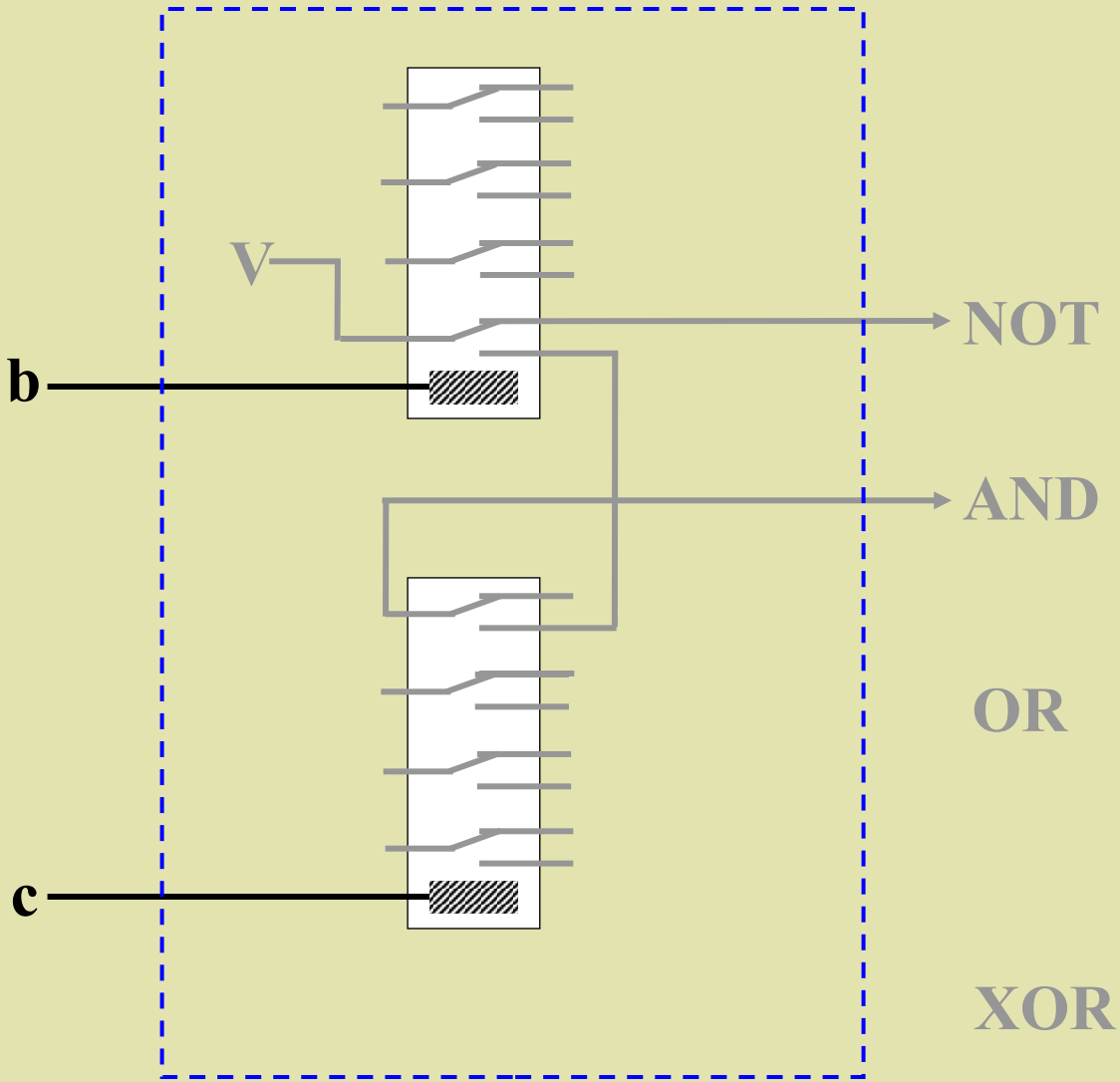
1-Bit Logic Circuit

b	c	NOT	AND	OR	XOR
0	0	1	0	0	0
0	1	1	0	1	1
1	0	0	0	1	1
1	1	0	1	1	0



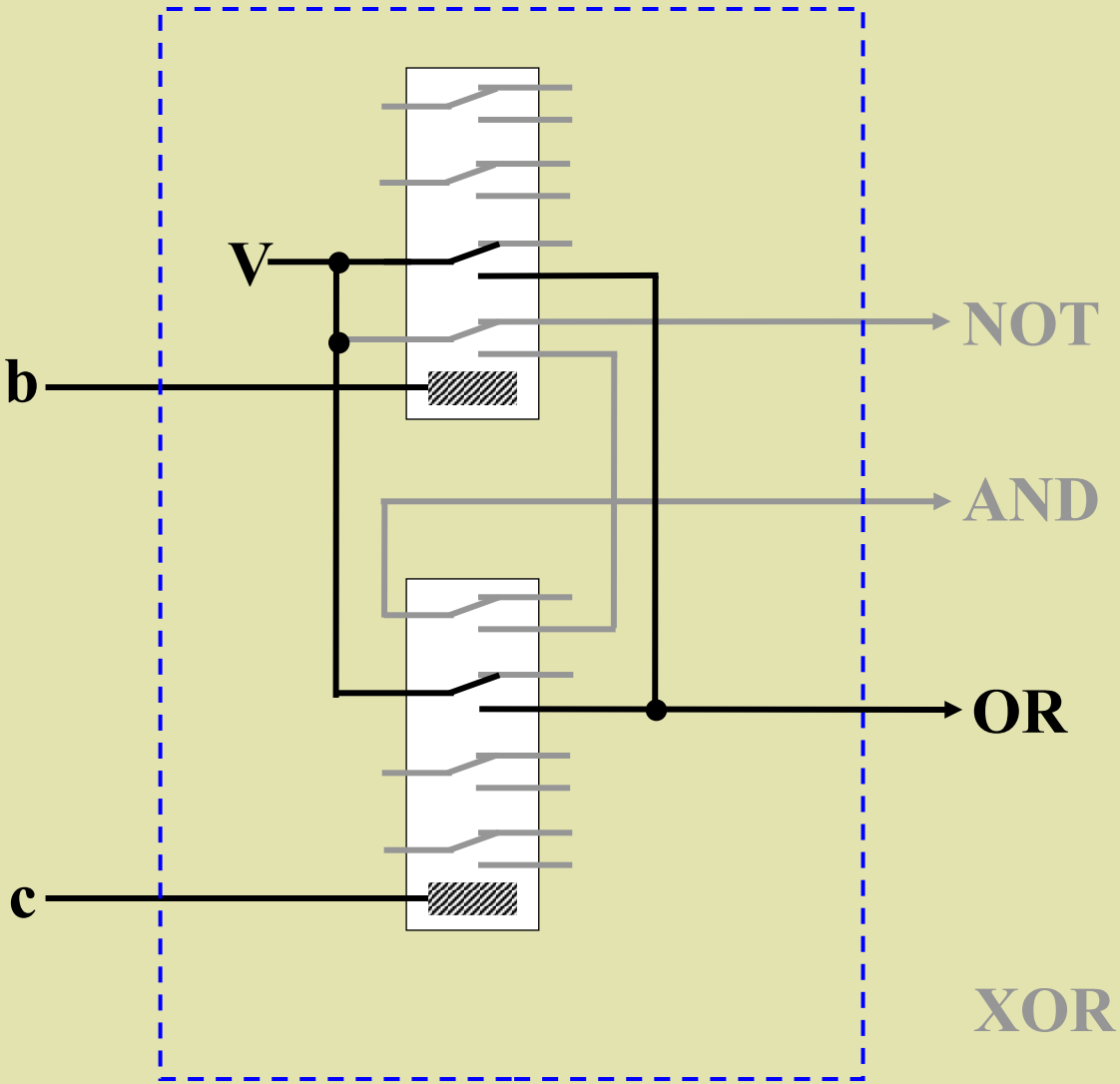
1-Bit Logic Circuit

b	c	NOT	AND	OR	XOR
0	0	1	0	0	0
0	1	1	0	1	1
1	0	0	0	1	1
1	1	0	1	1	0



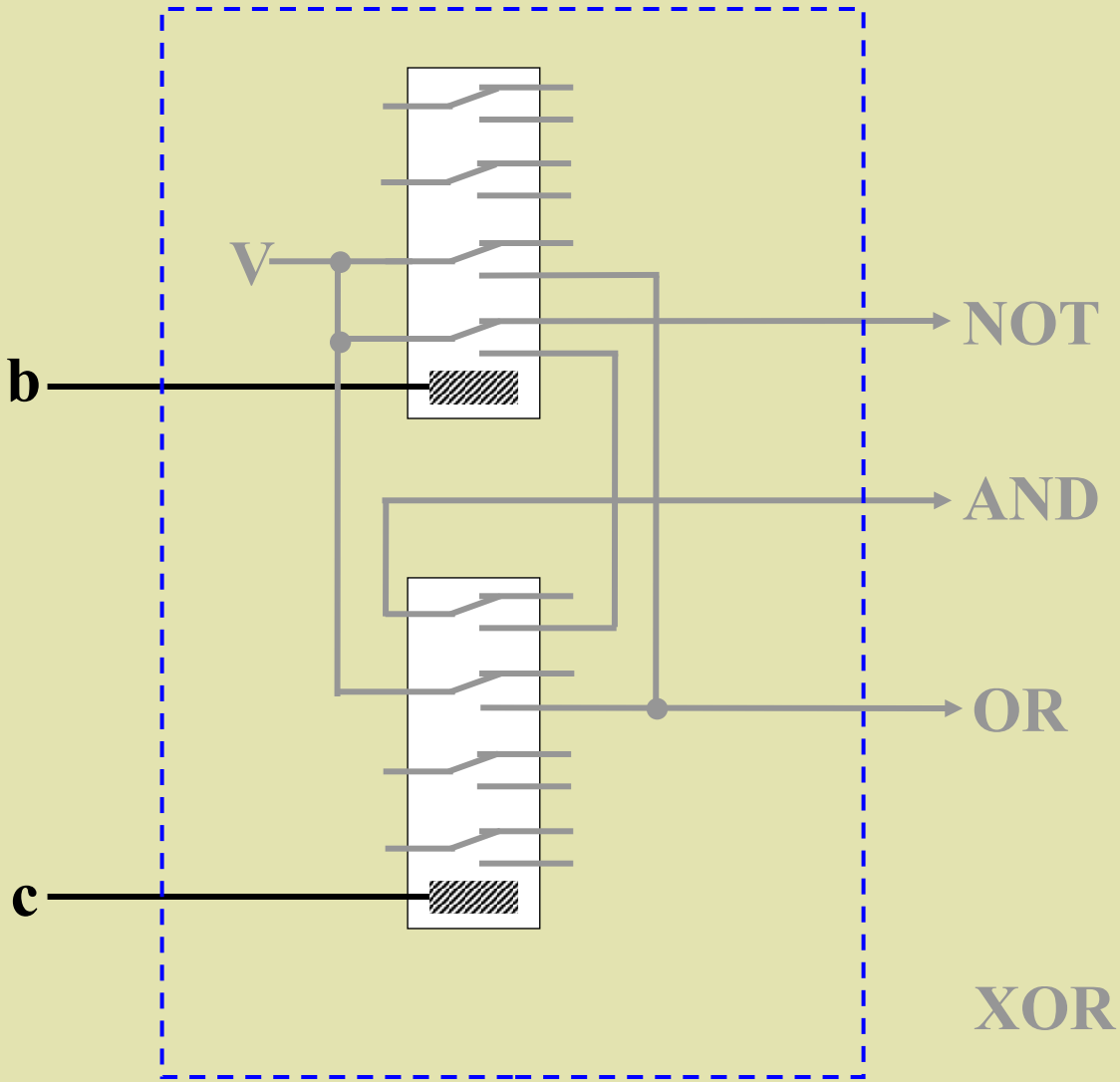
1-Bit Logic Circuit

b	c	NOT	AND	OR	XOR
0	0	1	0	0	0
0	1	1	0	1	1
1	0	0	0	1	1
1	1	0	1	1	0



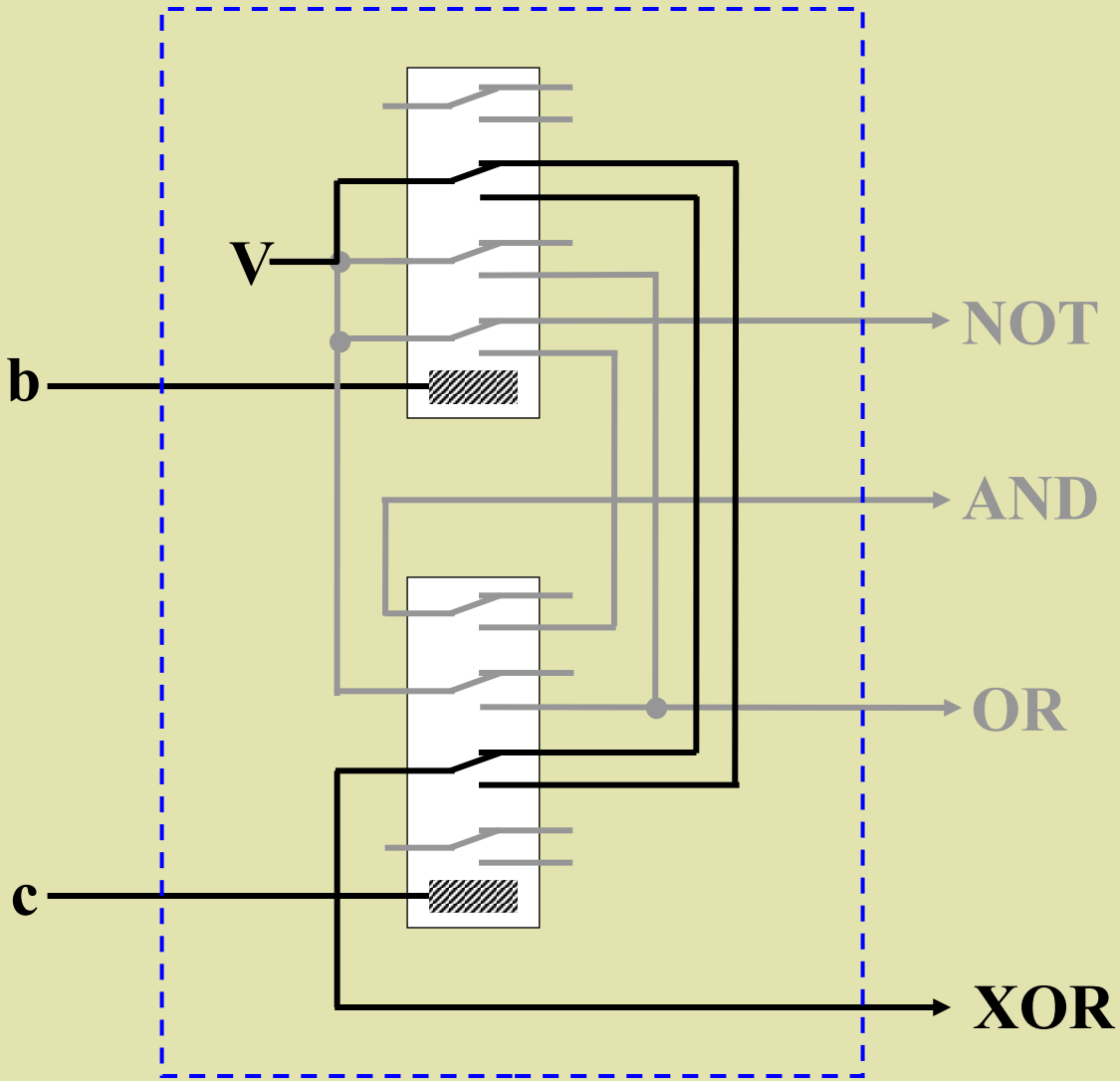
1-Bit Logic Circuit

b	c	NOT	AND	OR	XOR
0	0	1	0	0	0
0	1	1	0	1	1
1	0	0	0	1	1
1	1	0	1	1	0



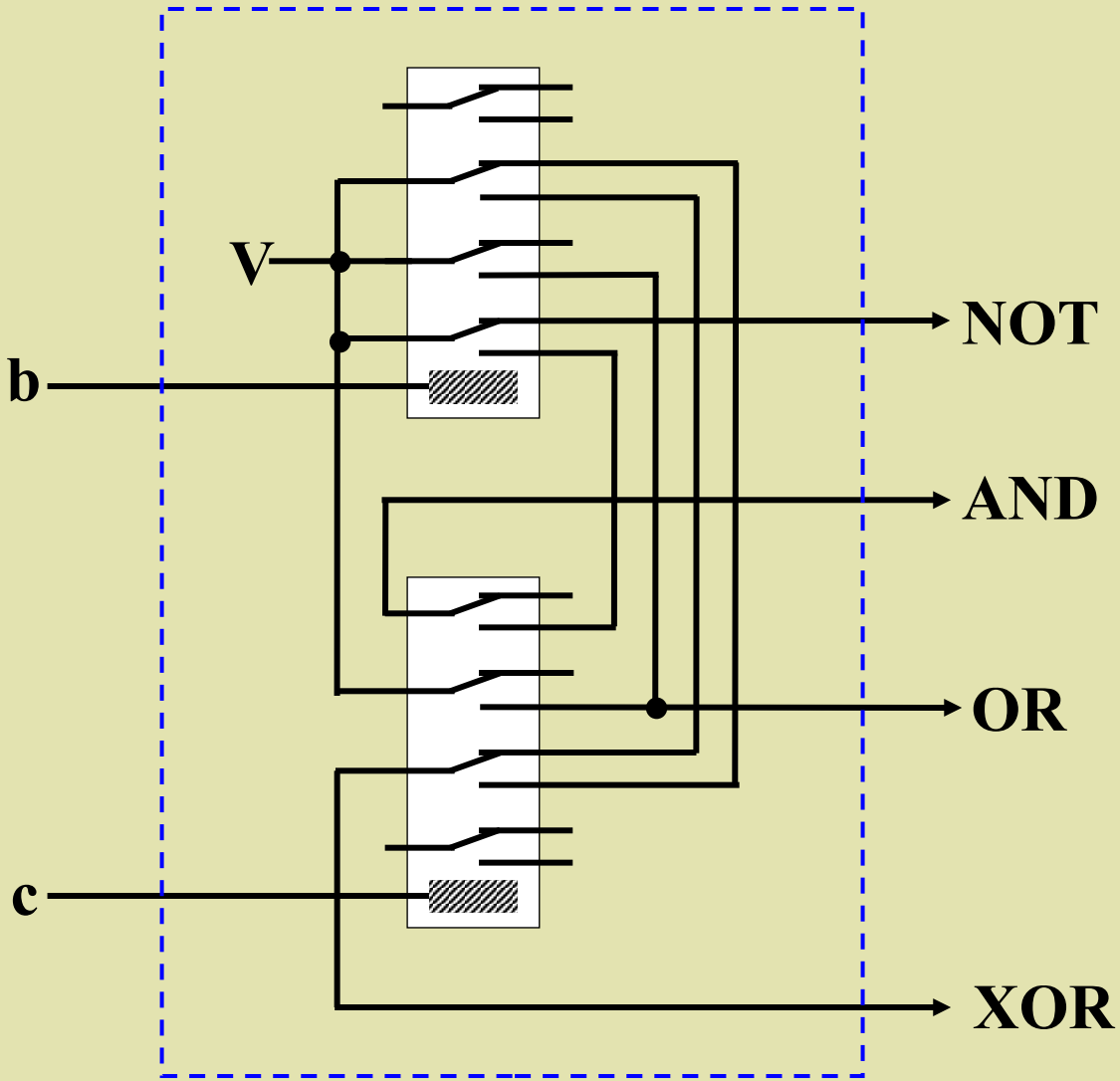
1-Bit Logic Circuit

b	c	NOT	AND	OR	XOR
0	0	1	0	0	0
0	1	1	0	1	1
1	0	0	0	1	1
1	1	0	1	1	0



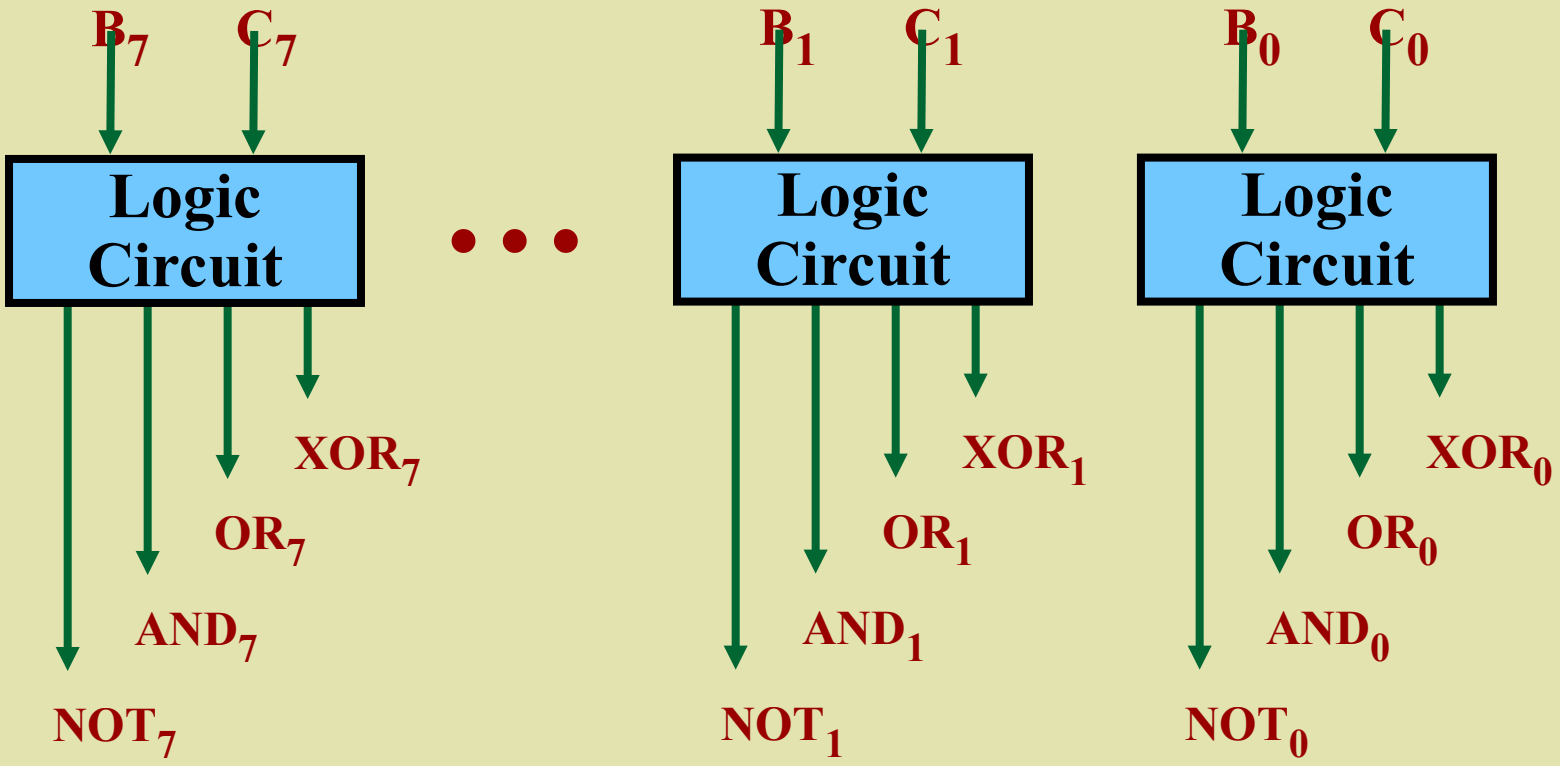
1-Bit Logic Circuit

b	c	NOT	AND	OR	XOR
0	0	1	0	0	0
0	1	1	0	1	1
1	0	0	0	1	1
1	1	0	1	1	0



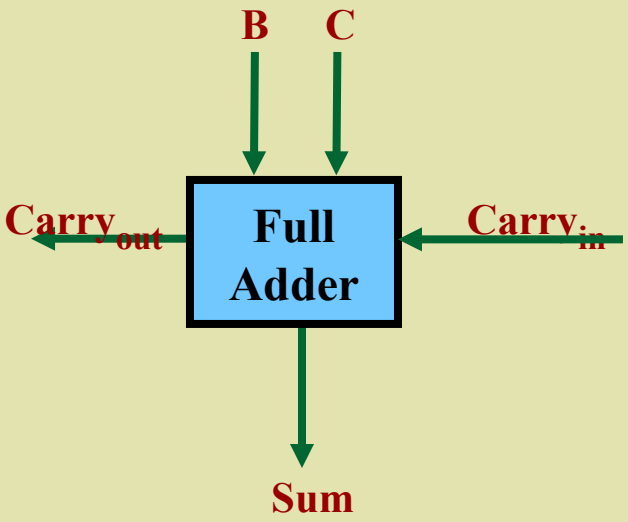
Eight 1-bit circuits can be combined to build an...

8-Bit Logic Circuit



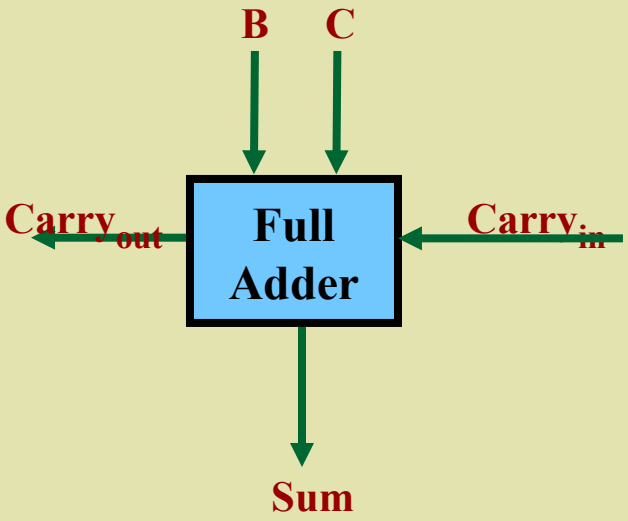
The "Full Adder" Circuit

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



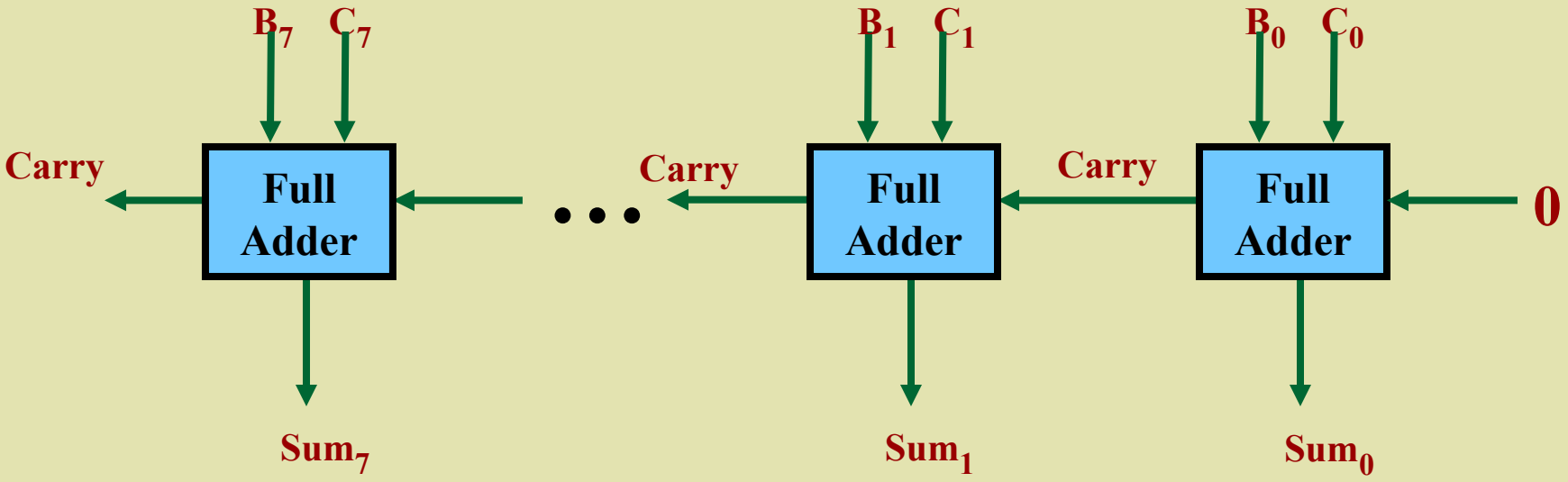
8 full-adders can be combined to build an...

8-Bit Adder



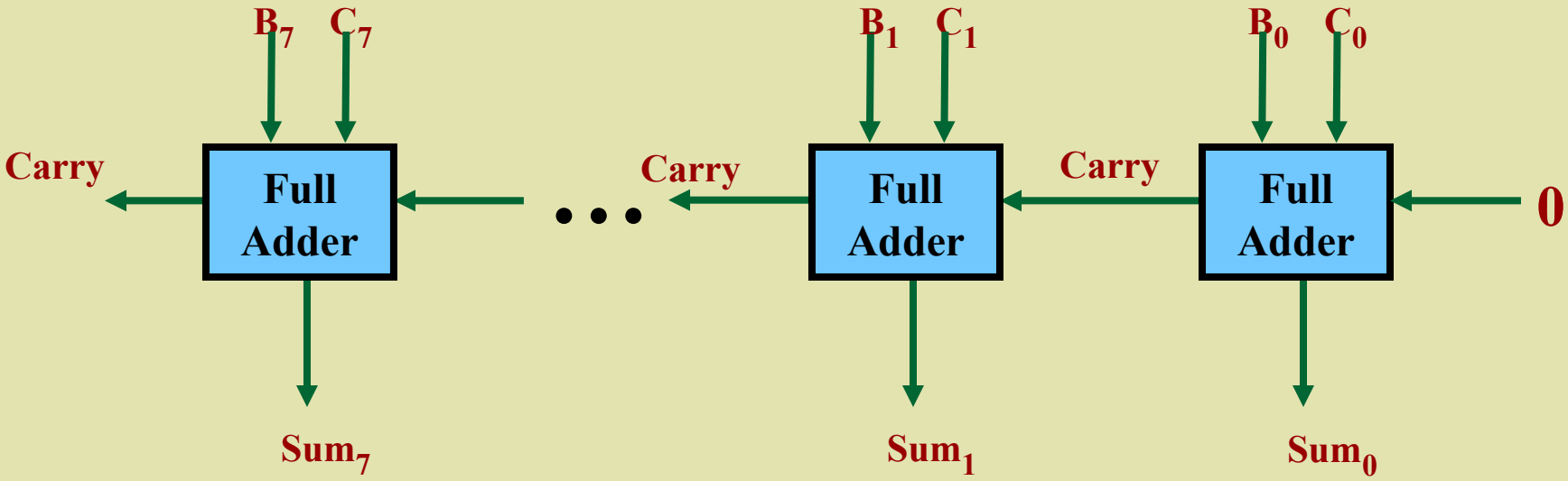
8 full-adders can be combined to build an...

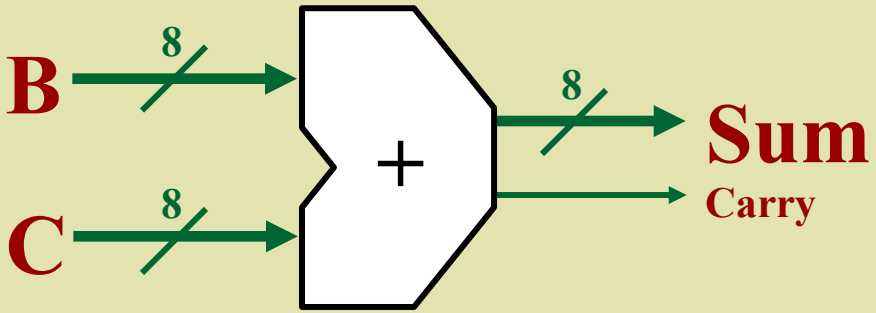
8-Bit Adder



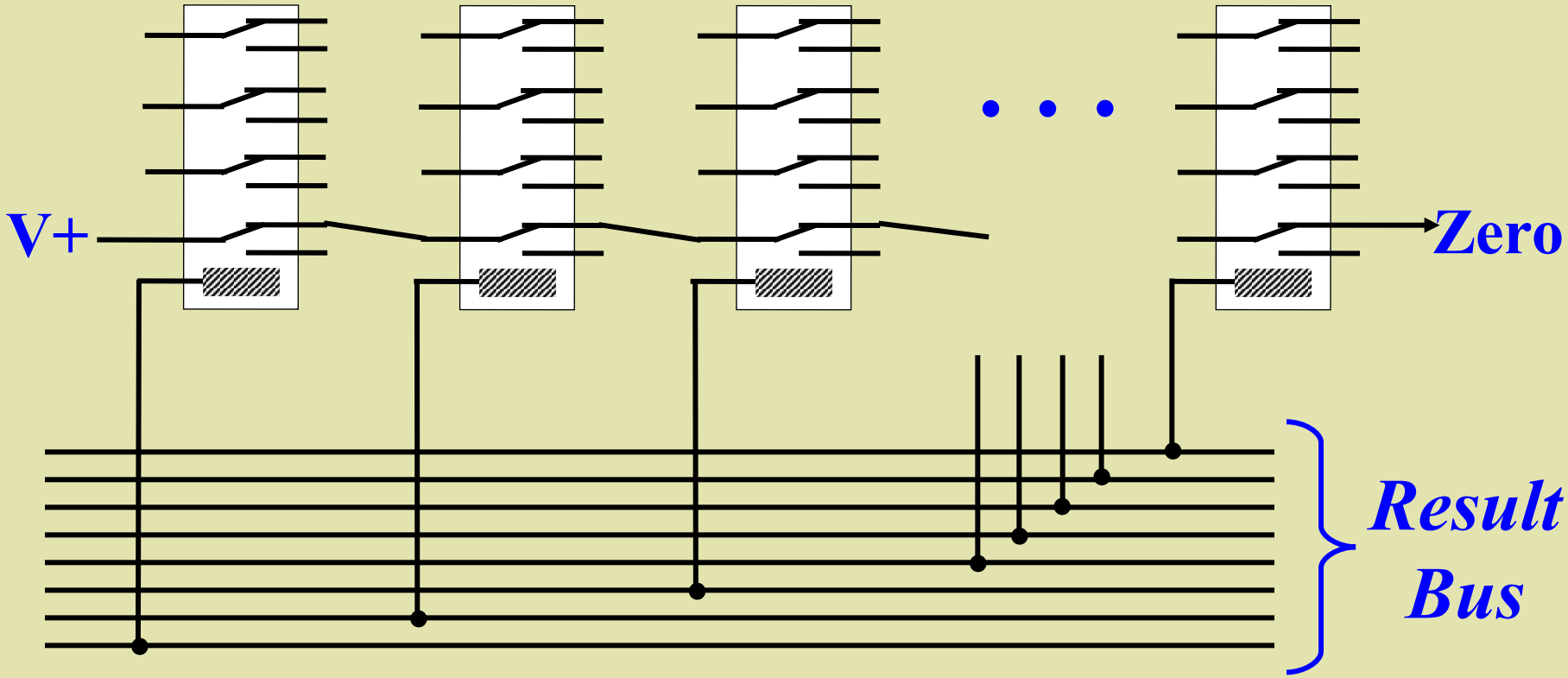
8 full-adders can be combined to build an...

8-Bit Adder

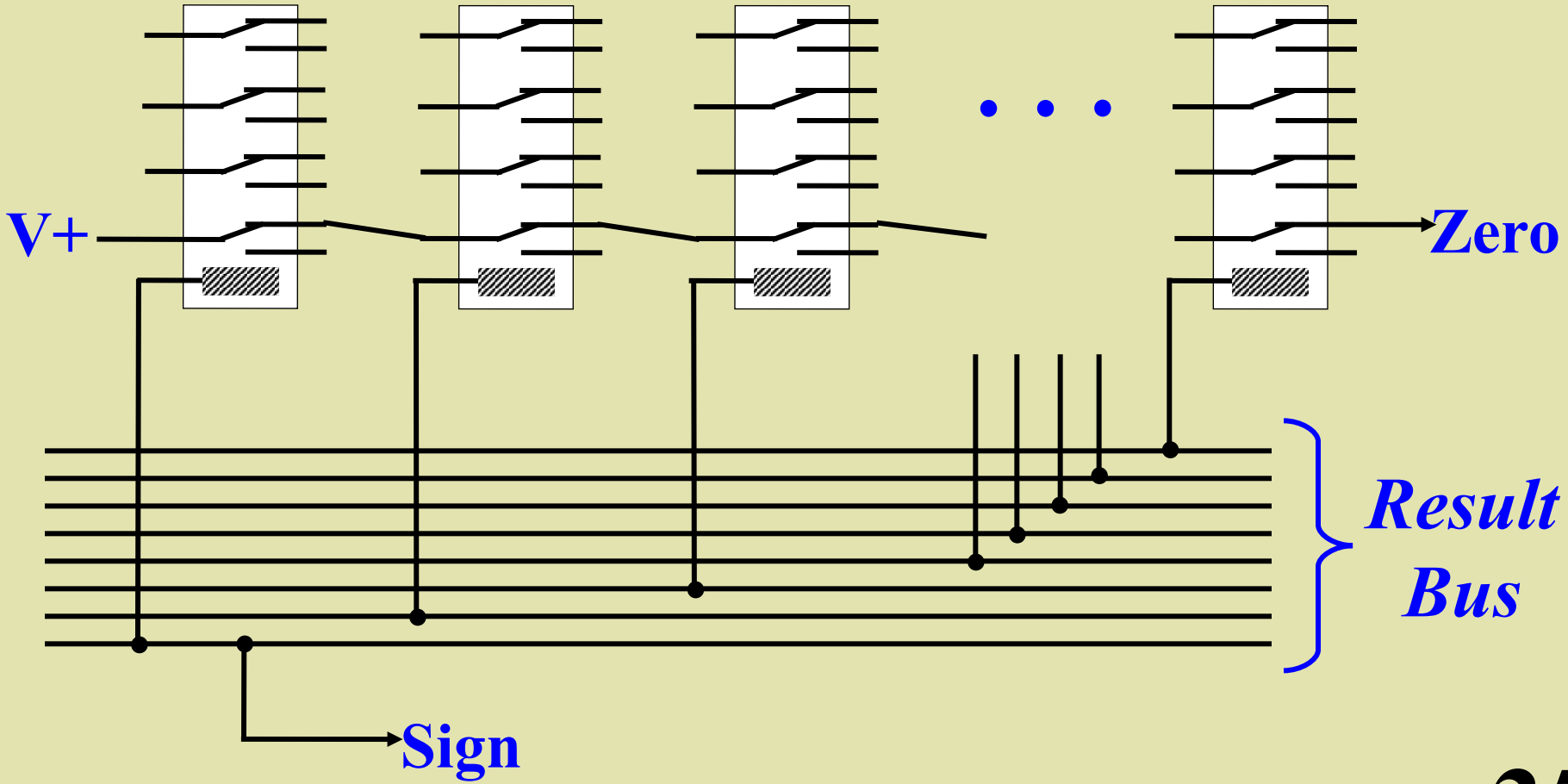




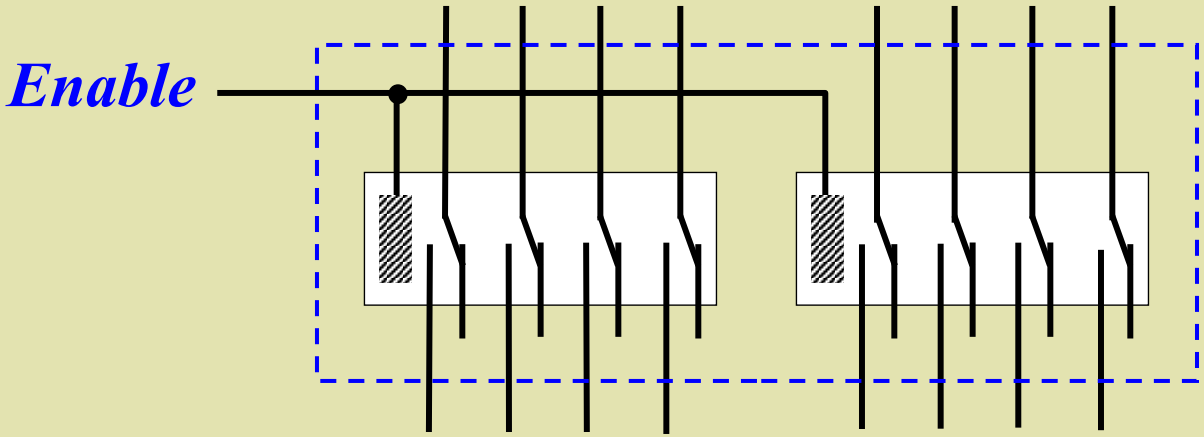
The Zero-Detect Circuit



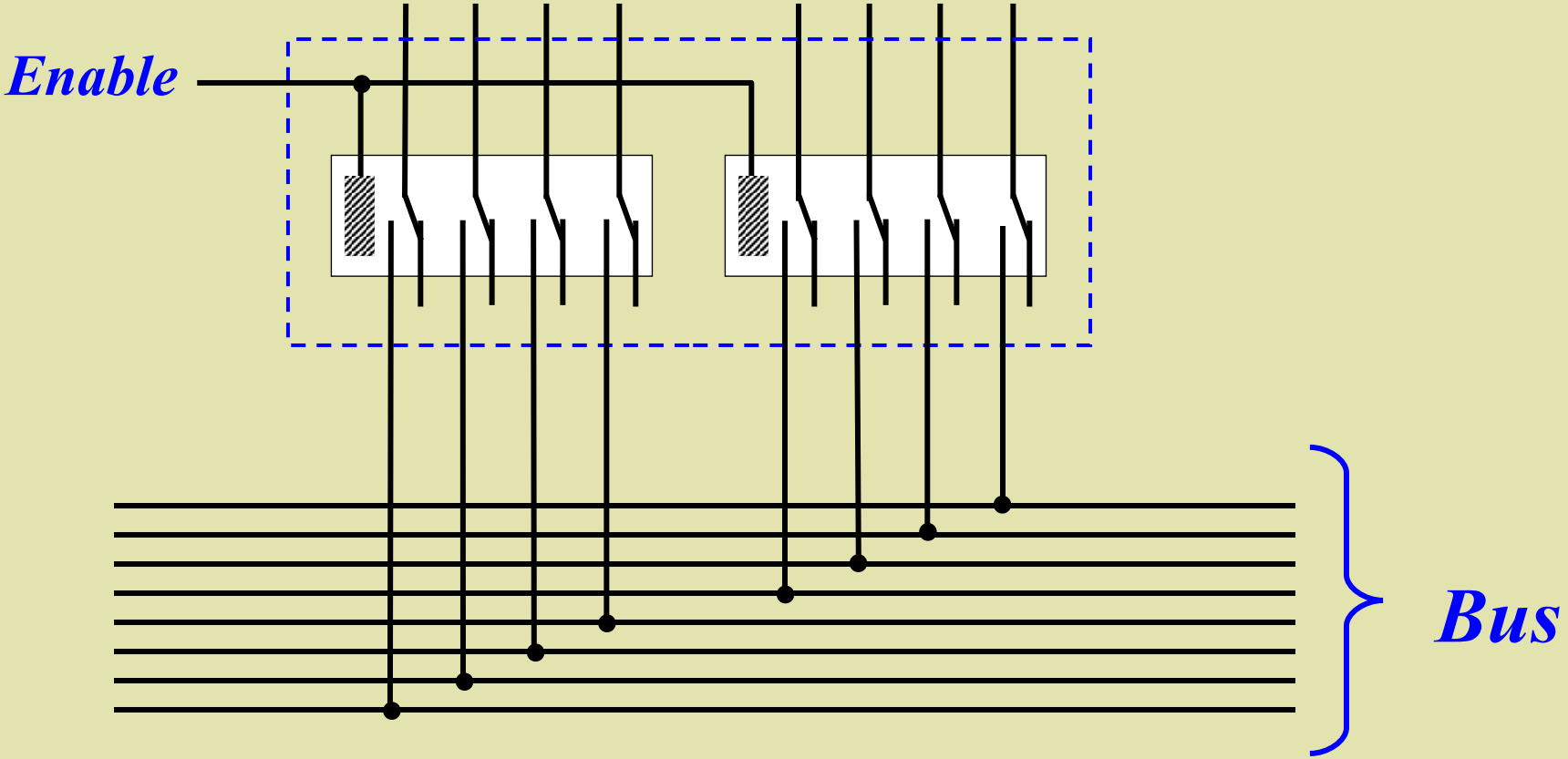
The Zero-Detect Circuit



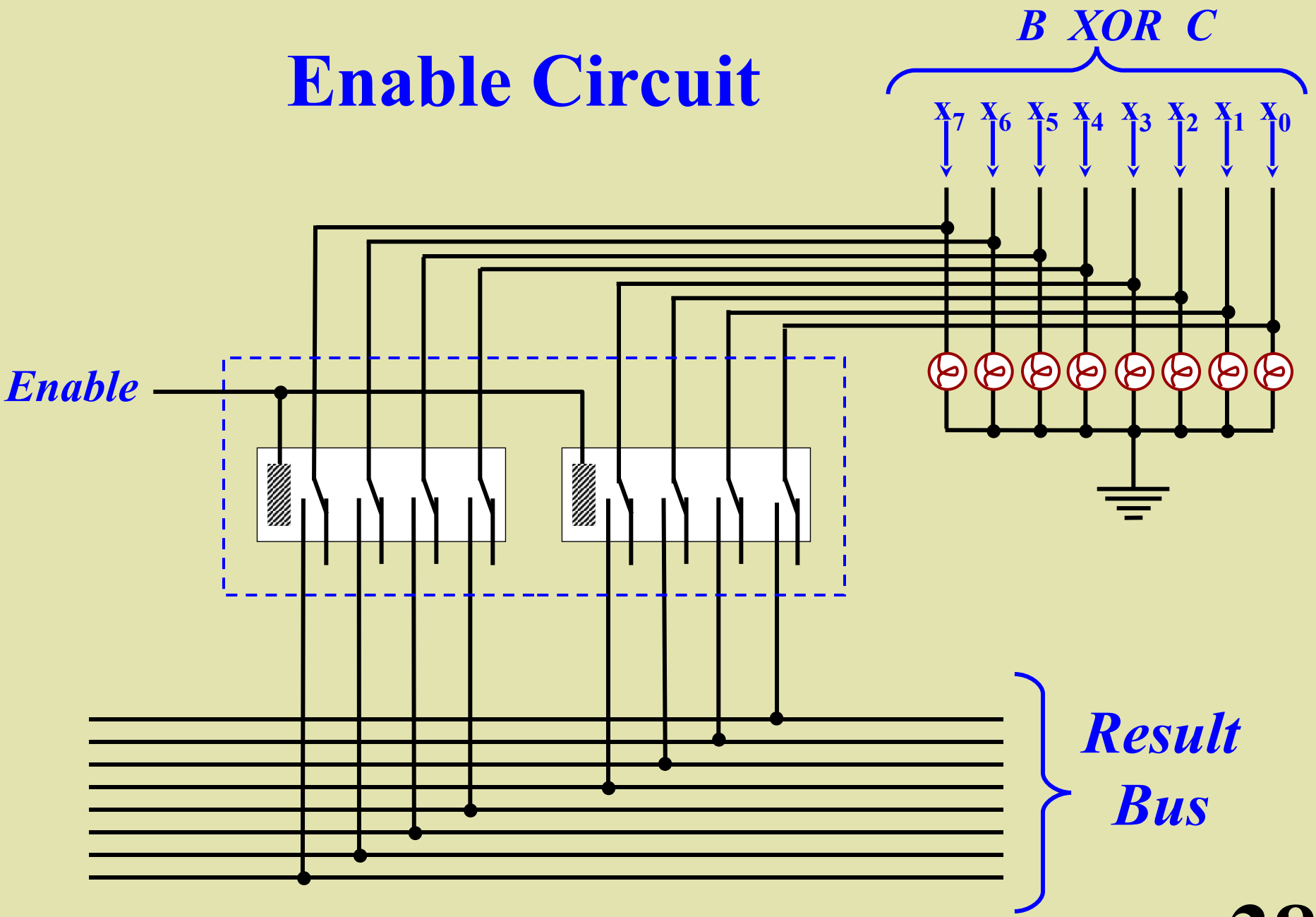
Enable Circuit



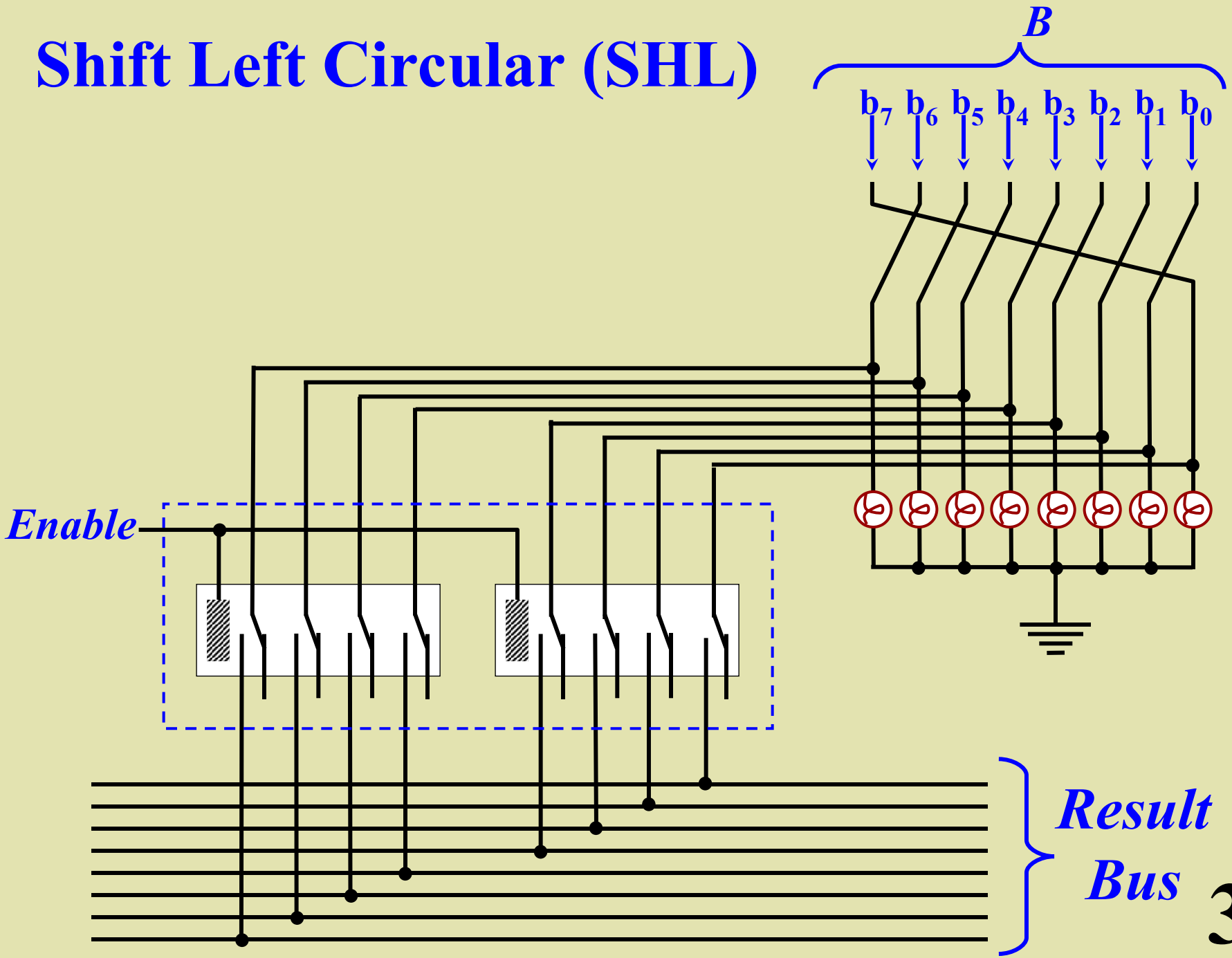
Enable Circuit



Enable Circuit



Shift Left Circular (SHL)

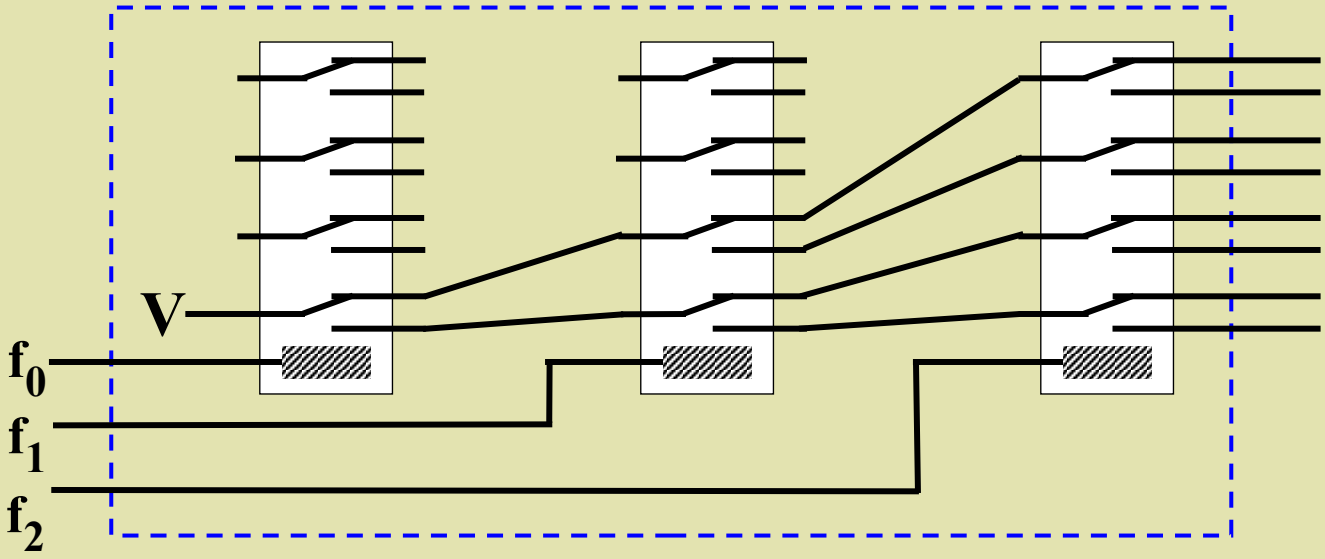


3-to-8 Decoder

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

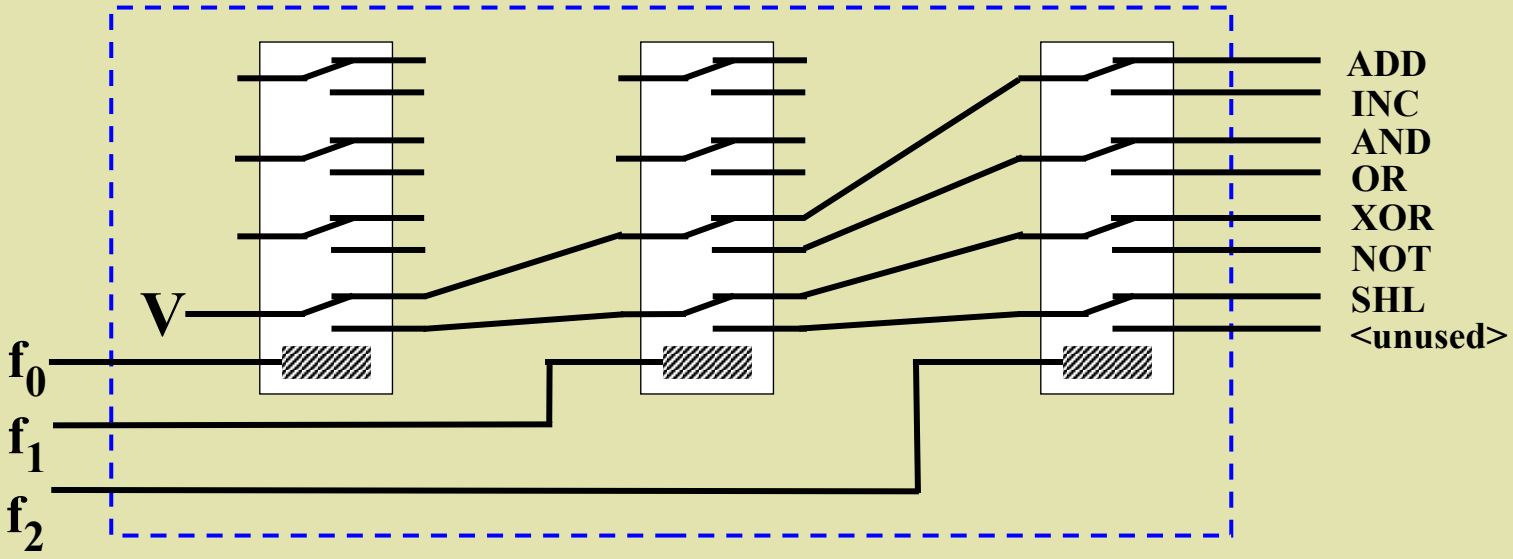
3-to-8 Decoder

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

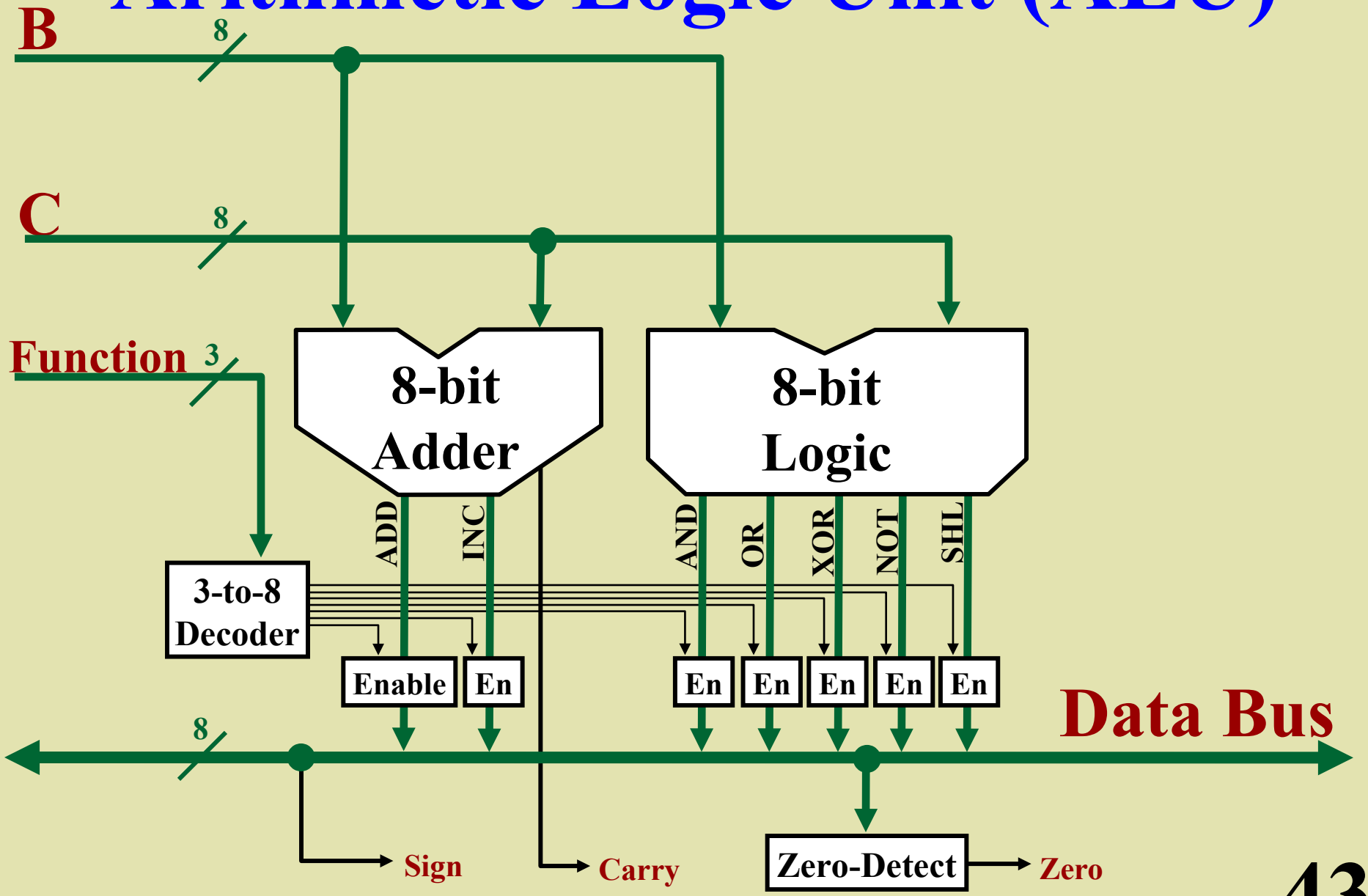


3-to-8 Decoder

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



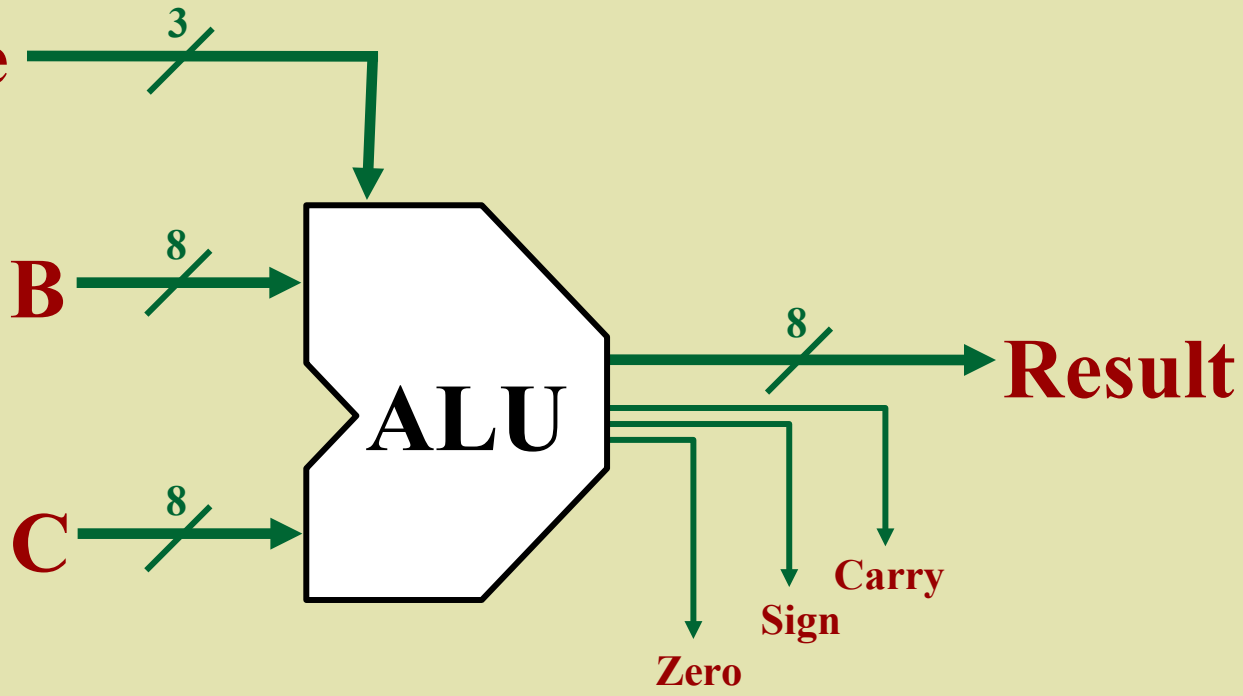
Arithmetic Logic Unit (ALU)



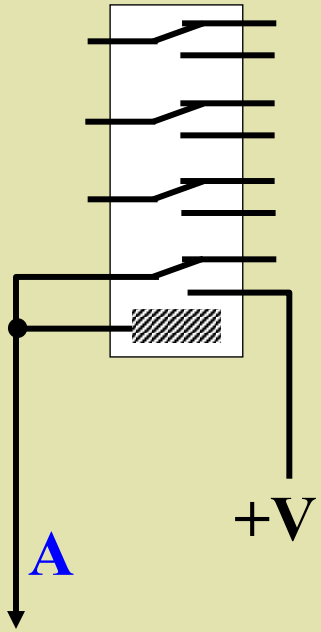
An 8-Bit ALU

Function Code

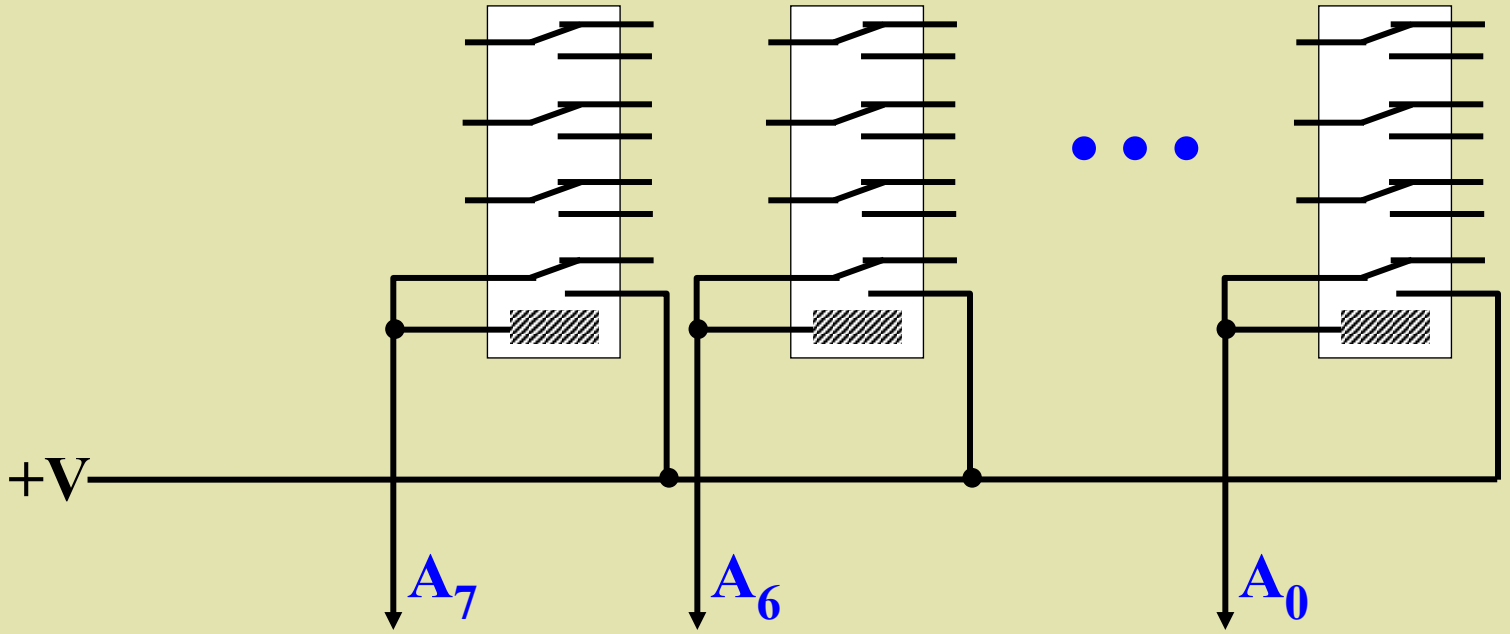
```
000 add
001 inc
010 and
011 or
100 xor
101 not
110 shl
111 <nop>
```



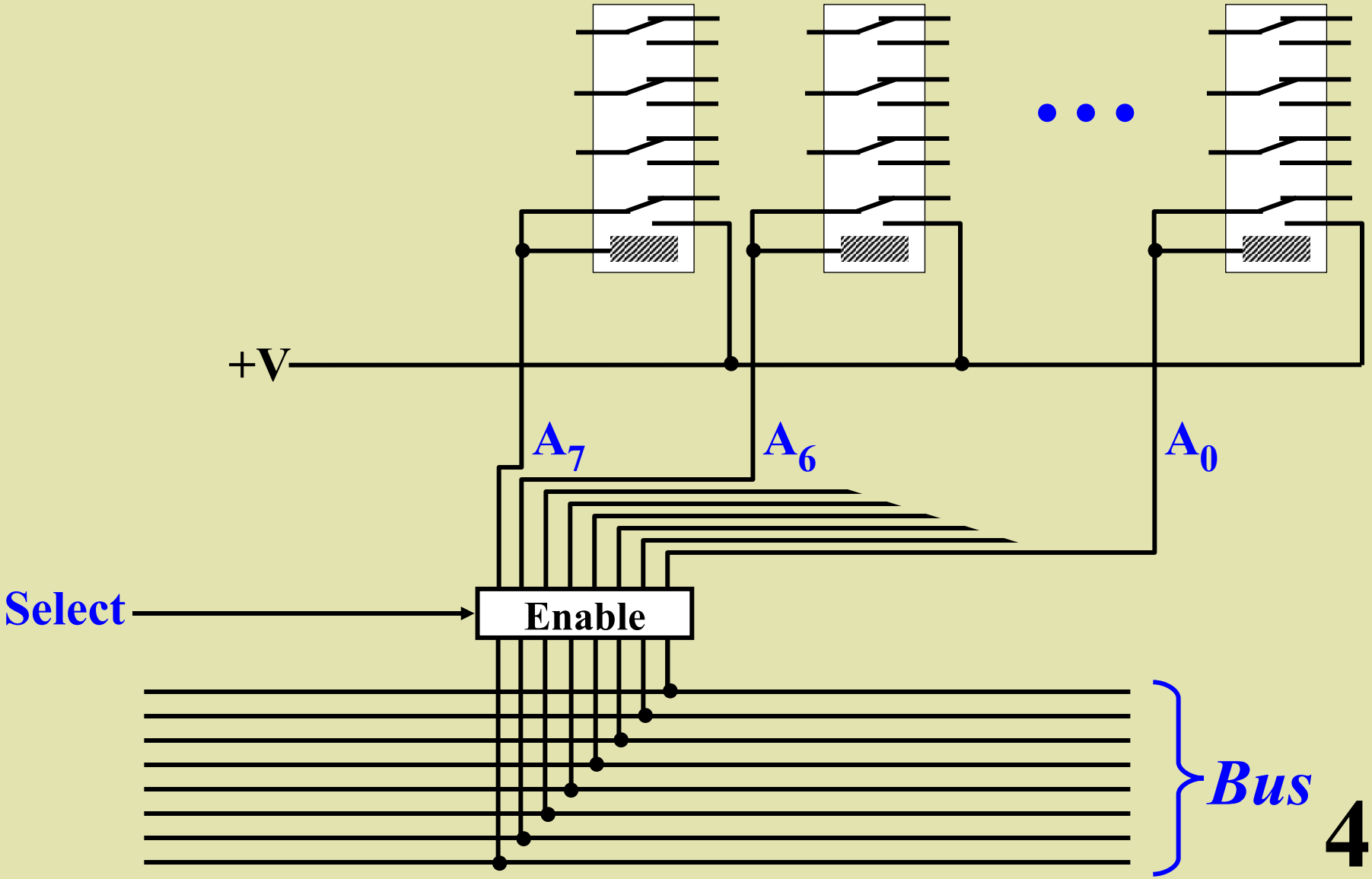
Register Storage



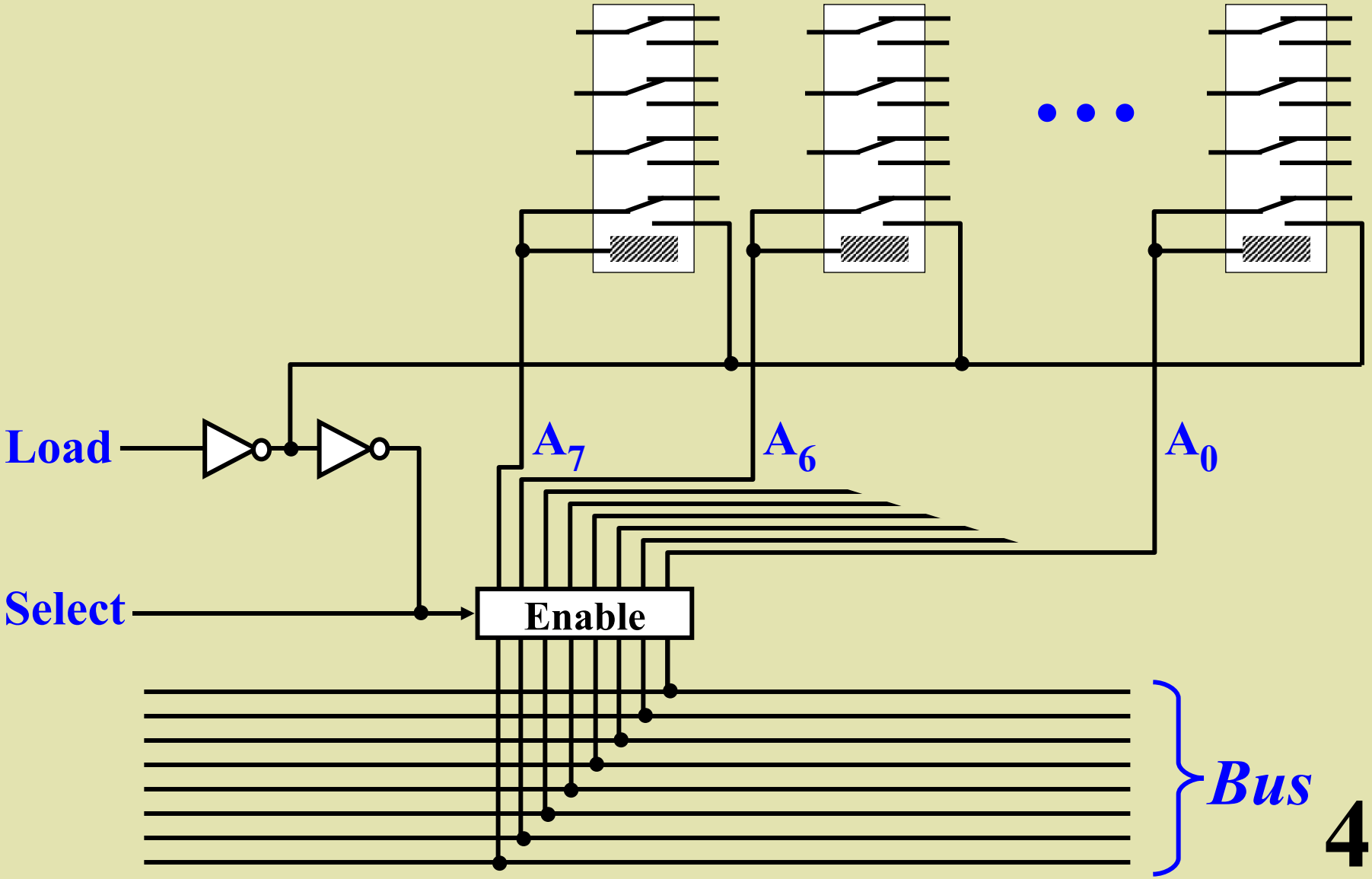
Register Storage



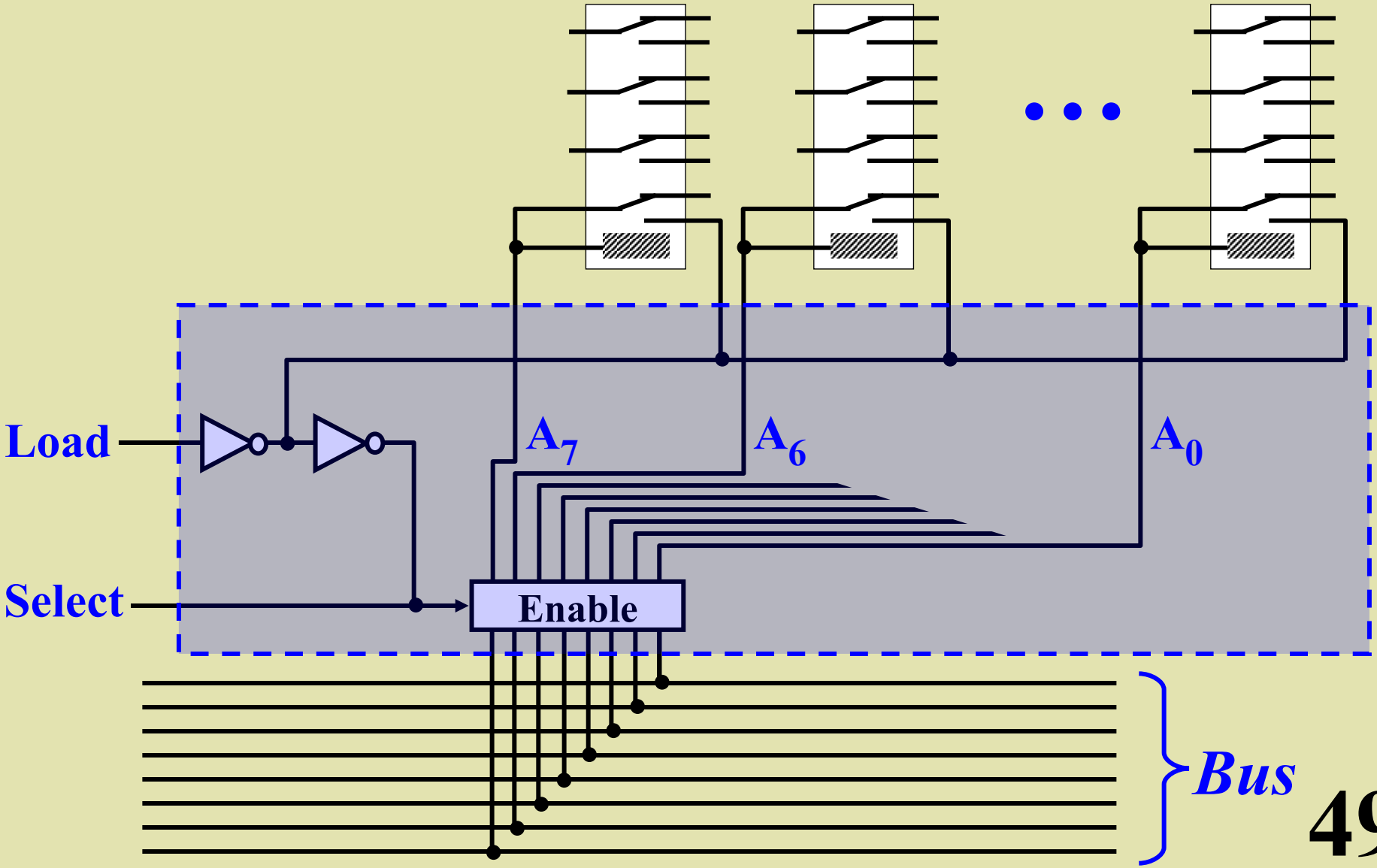
Register Storage



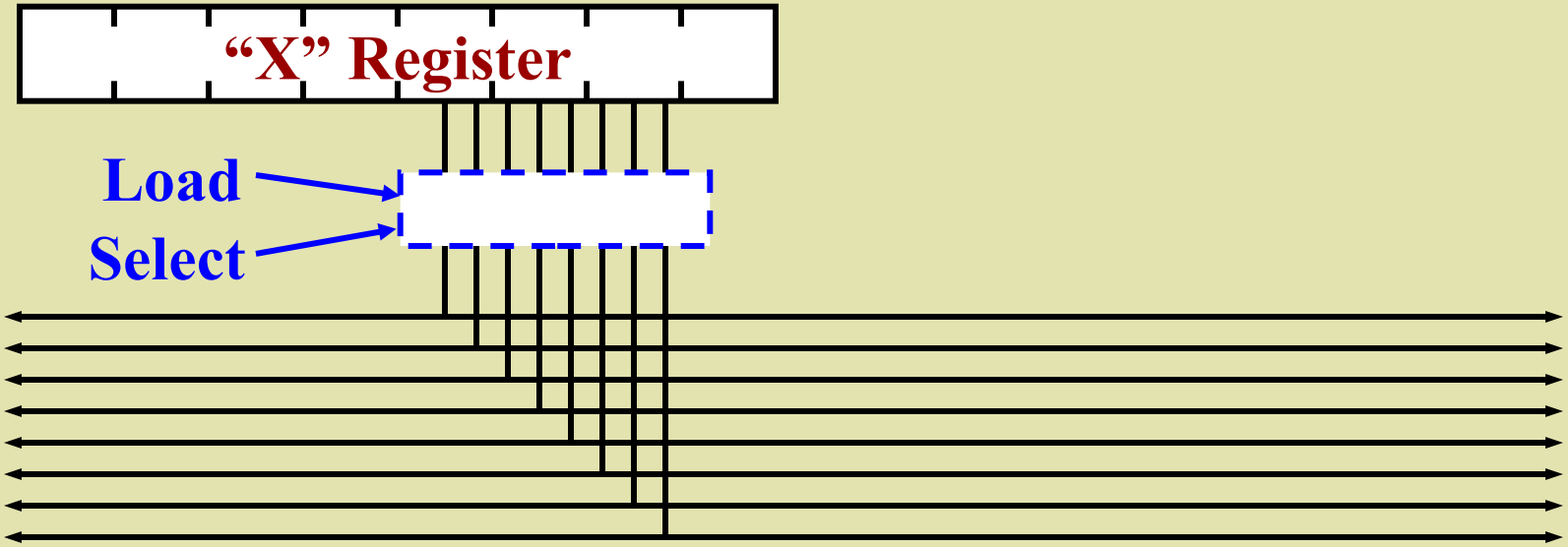
Register Storage



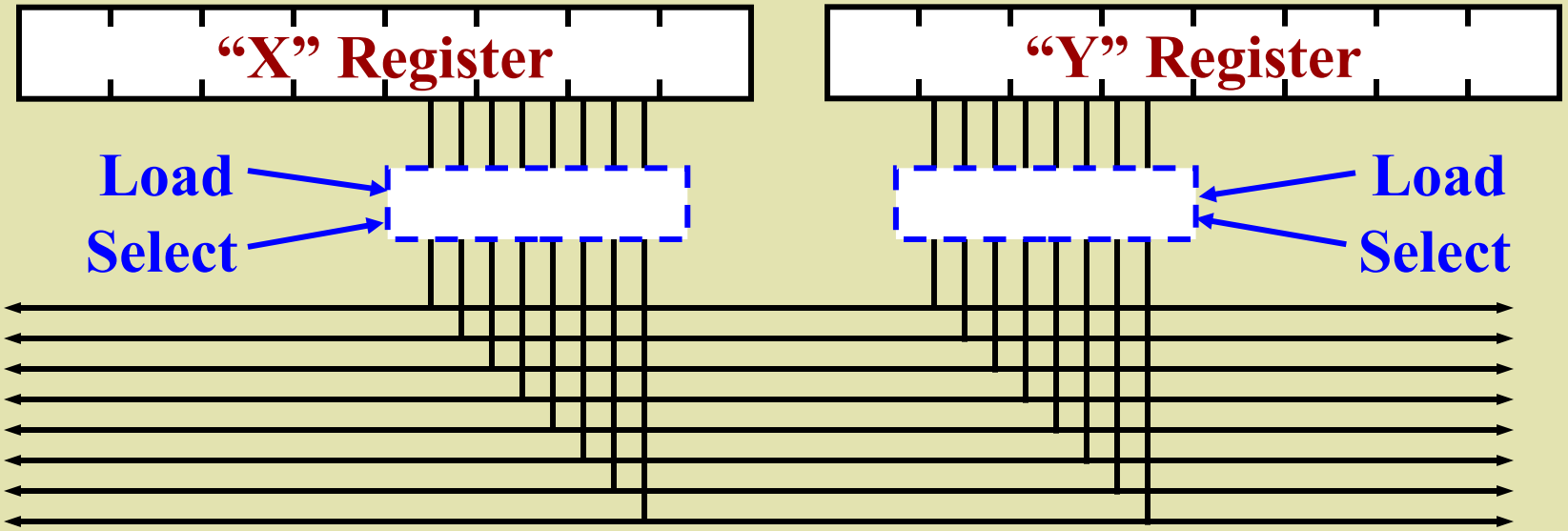
Register Storage



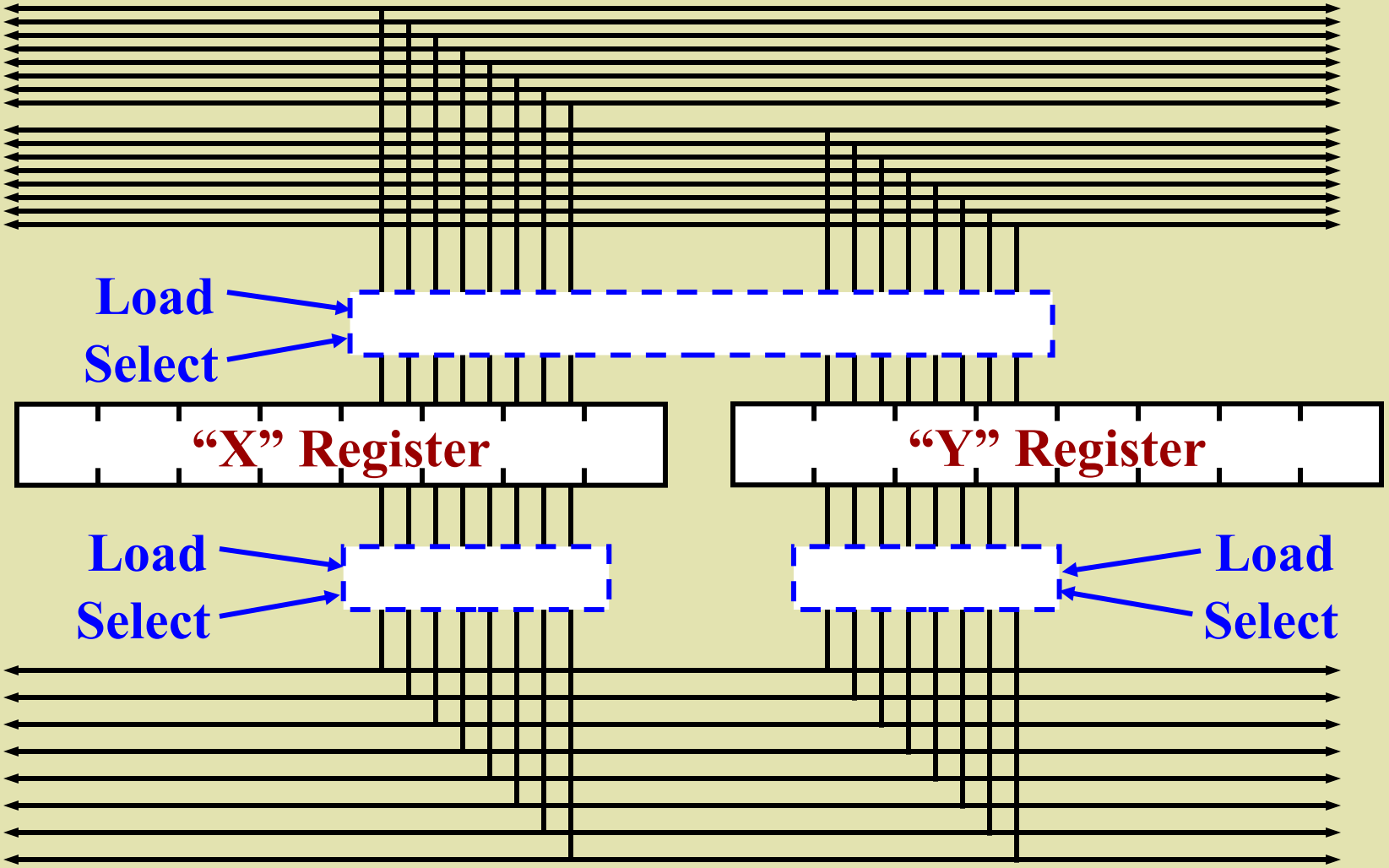
8-Bit Registers



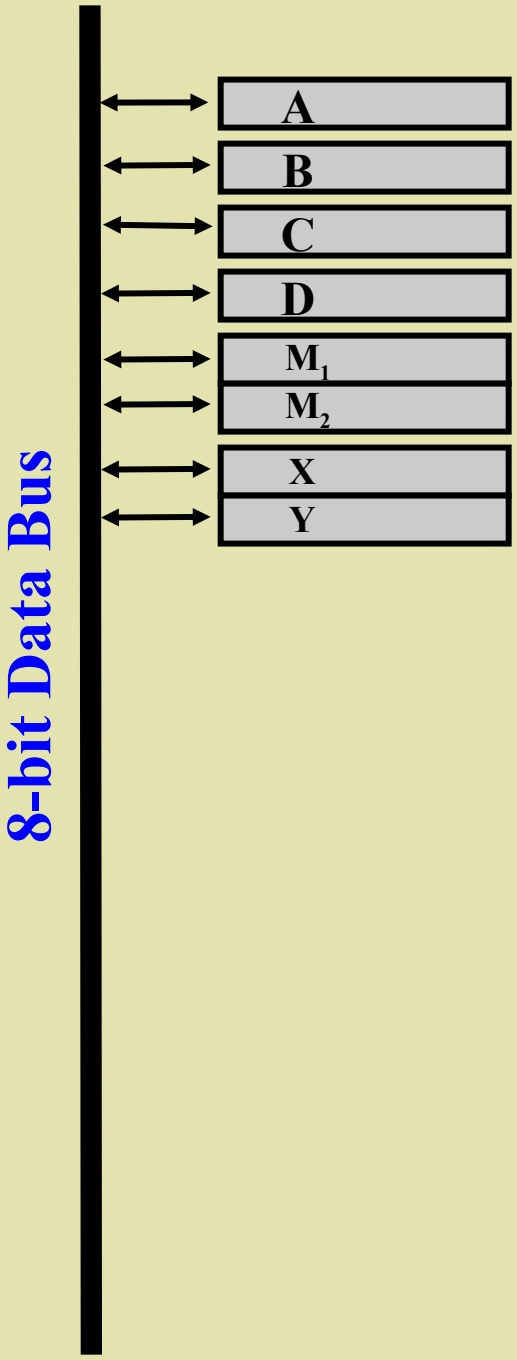
8-Bit Registers



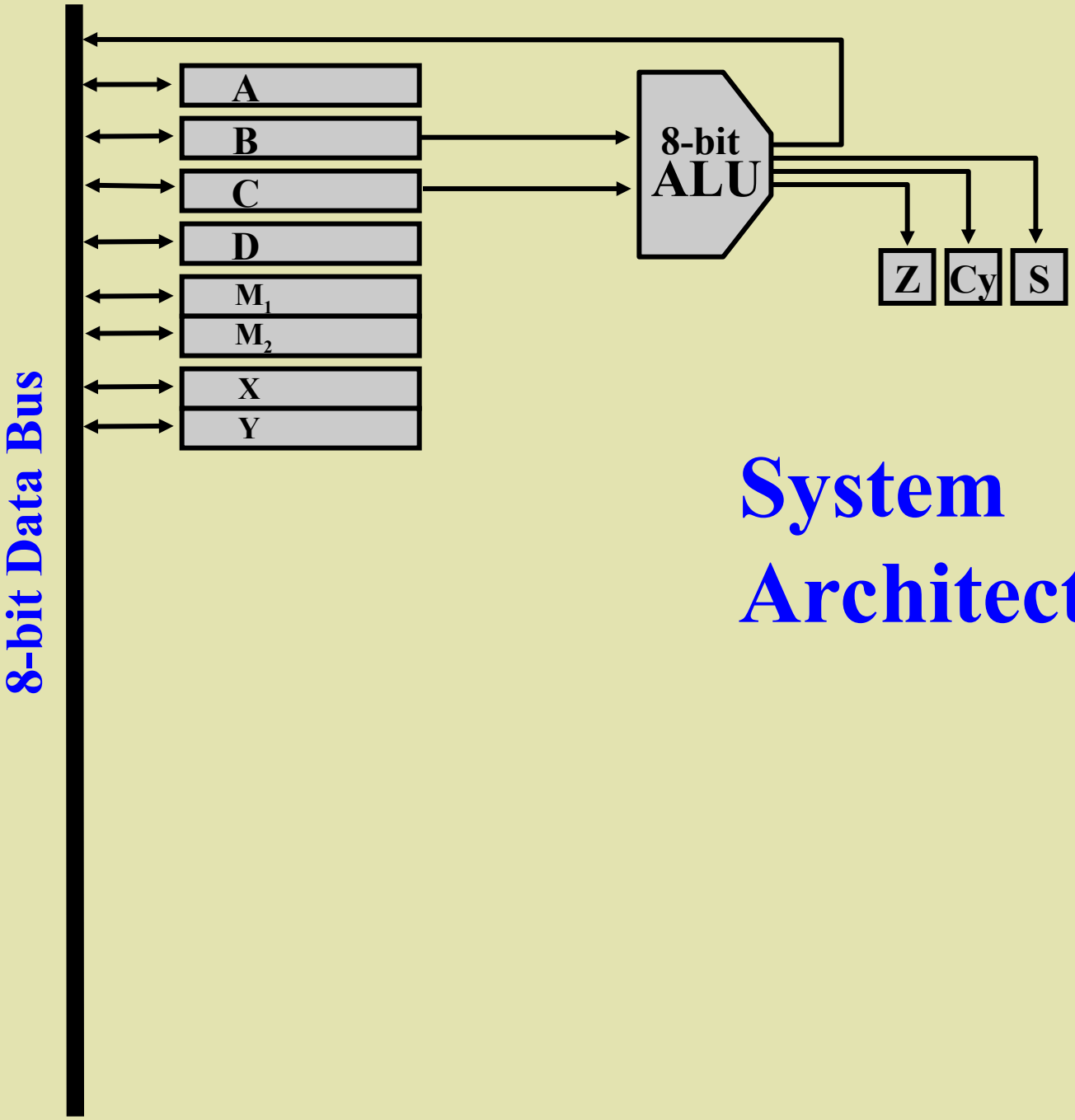
16-Bit "Address" Bus



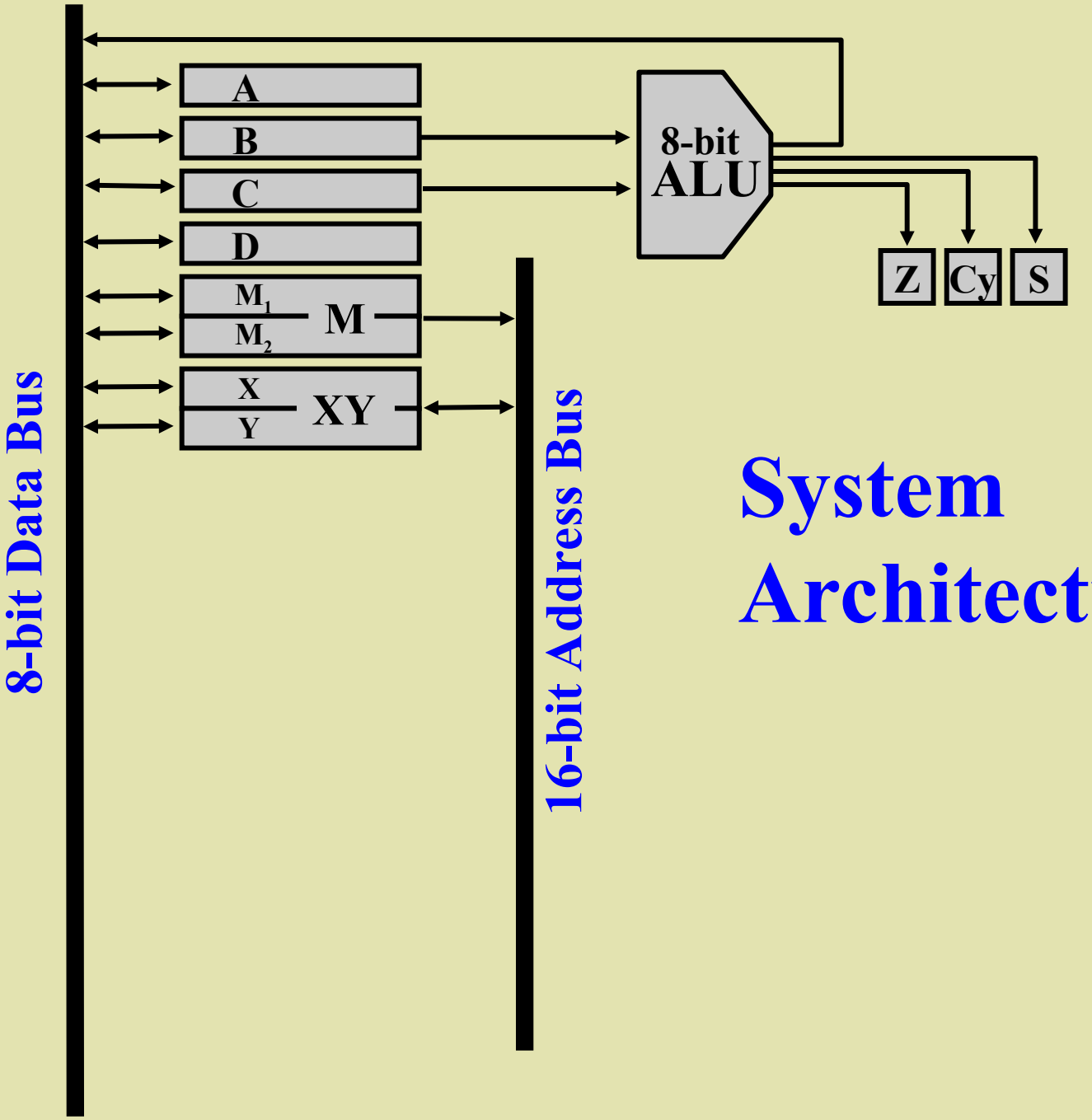
8-Bit "Data" Bus



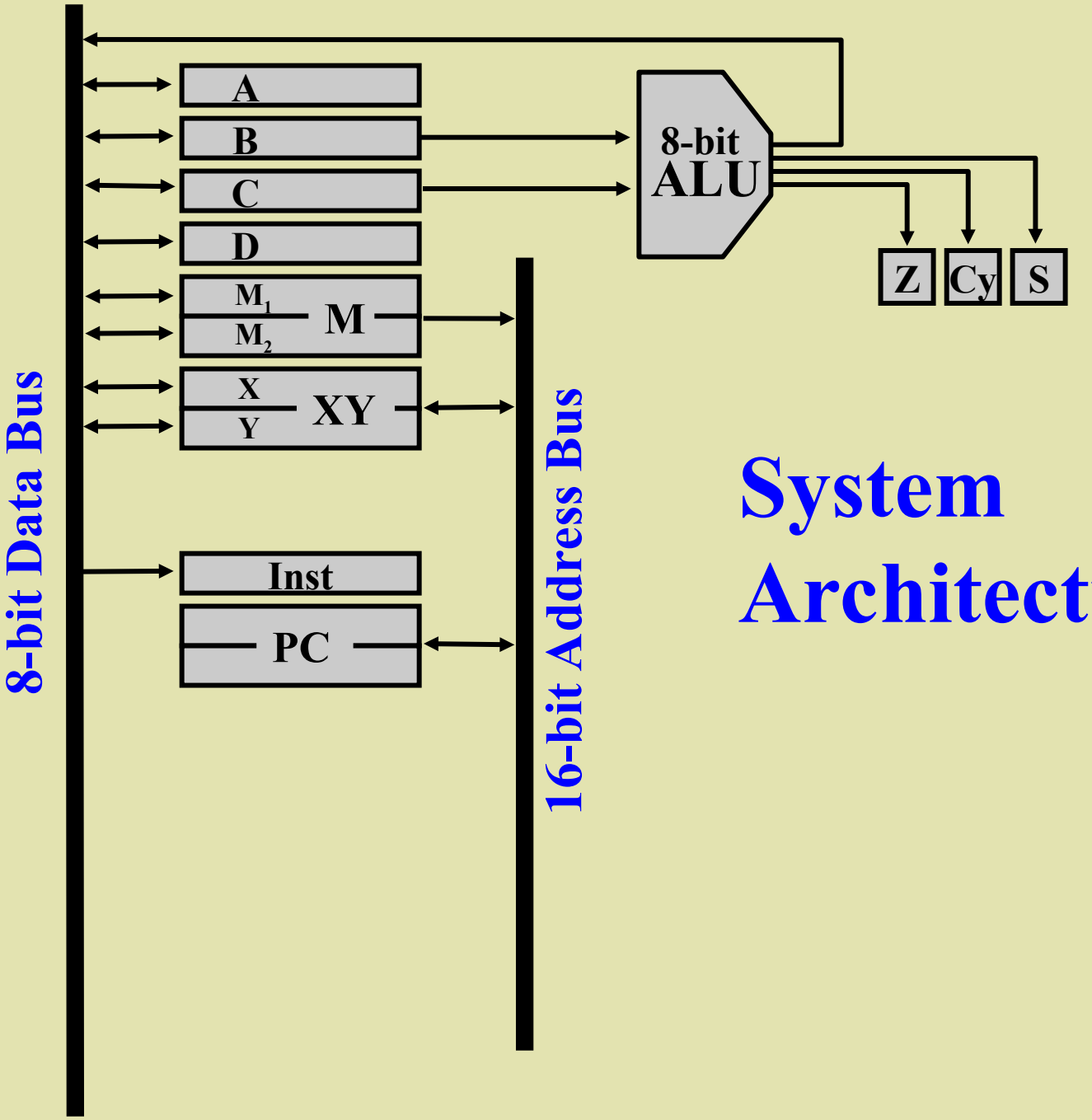
System Architecture



System Architecture

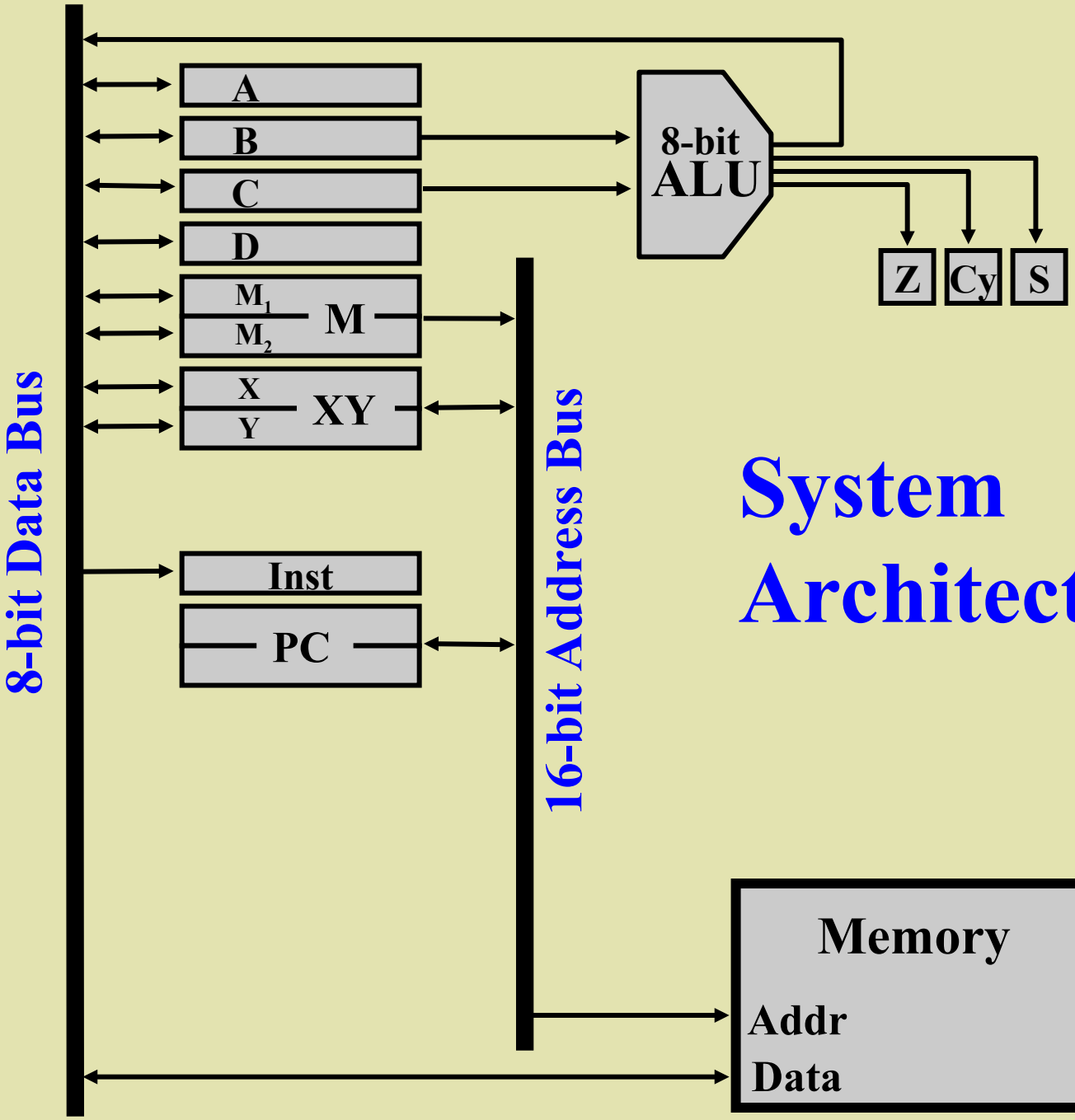


System Architecture

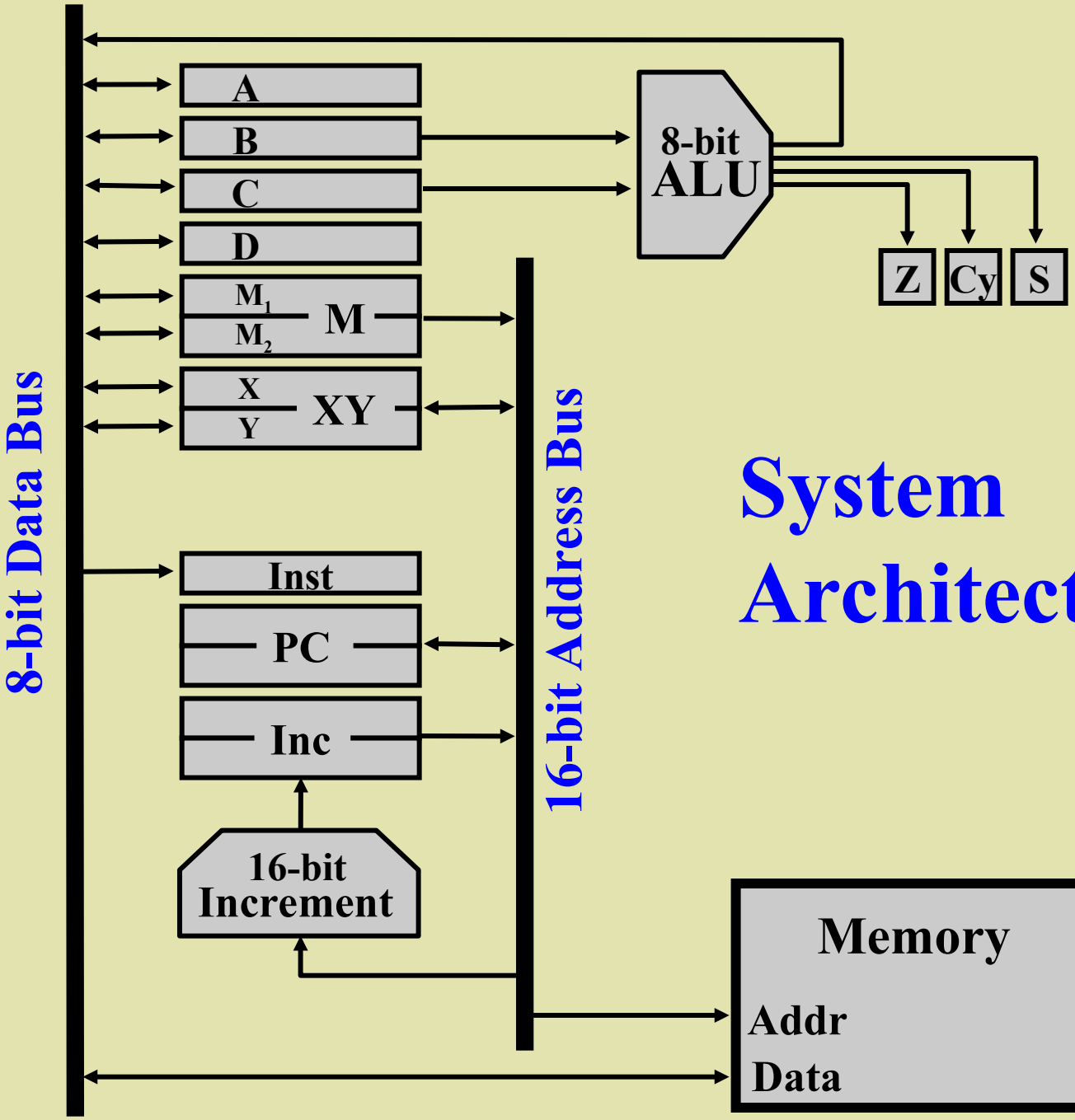


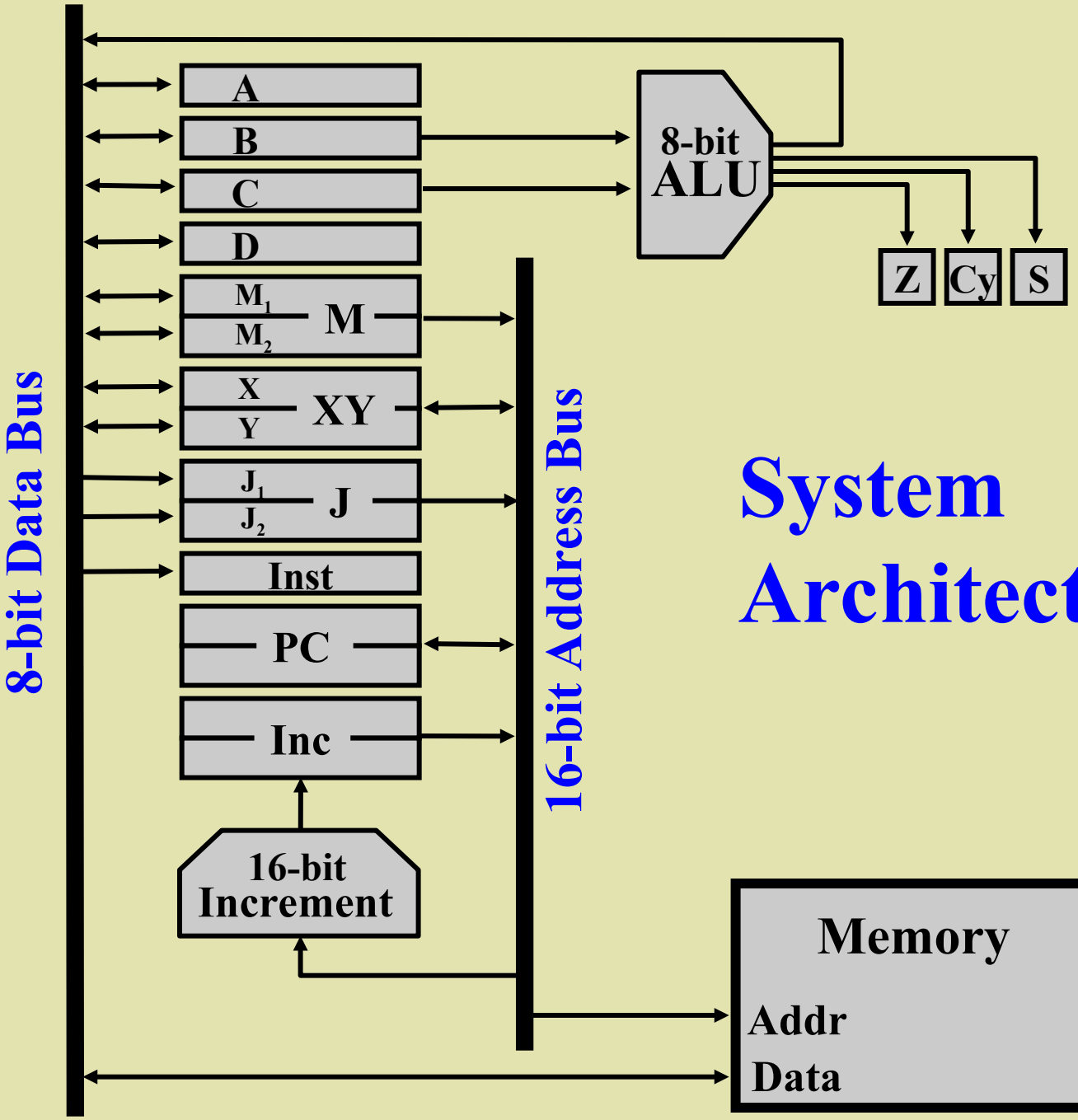
System Architecture

System Architecture

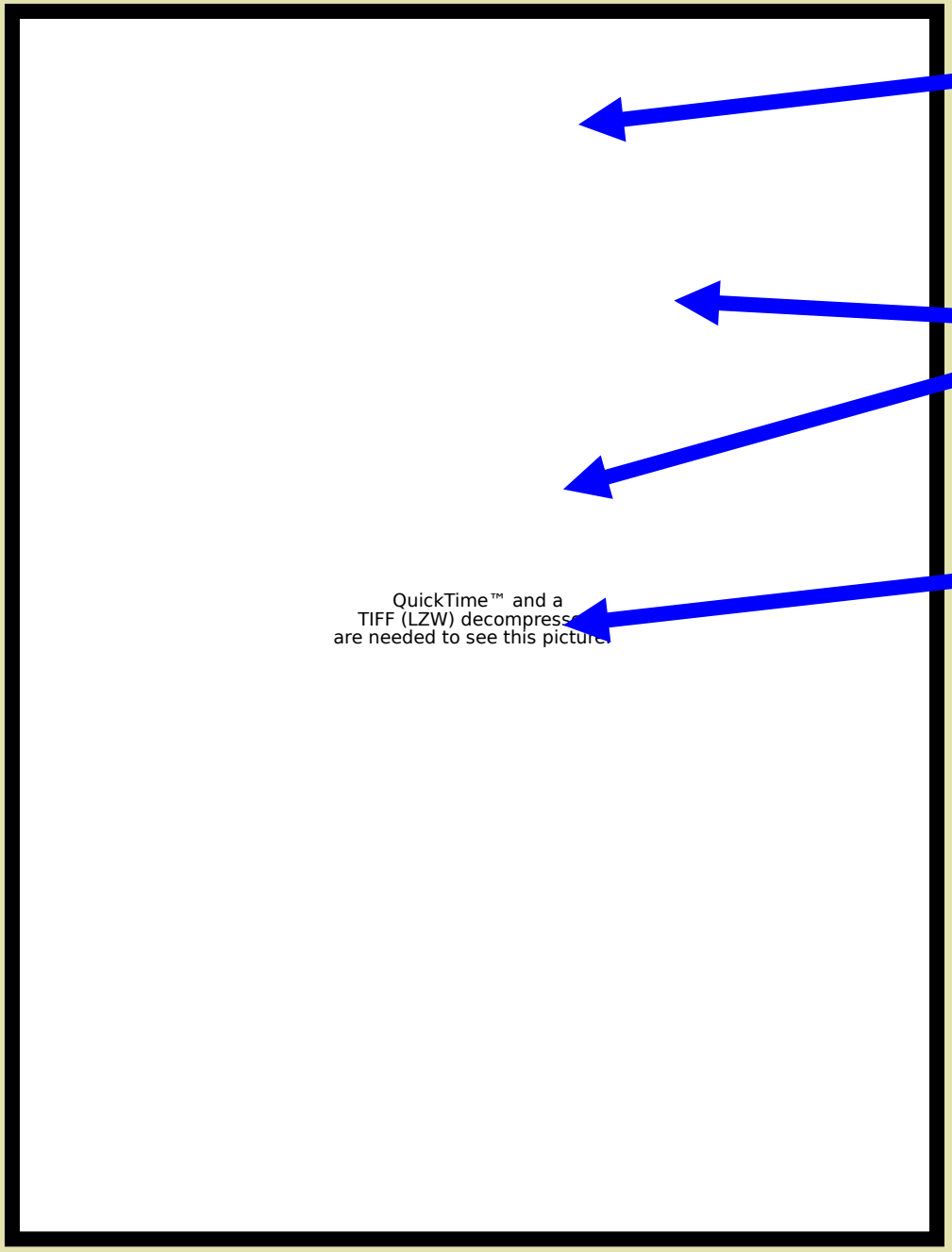


System Architecture





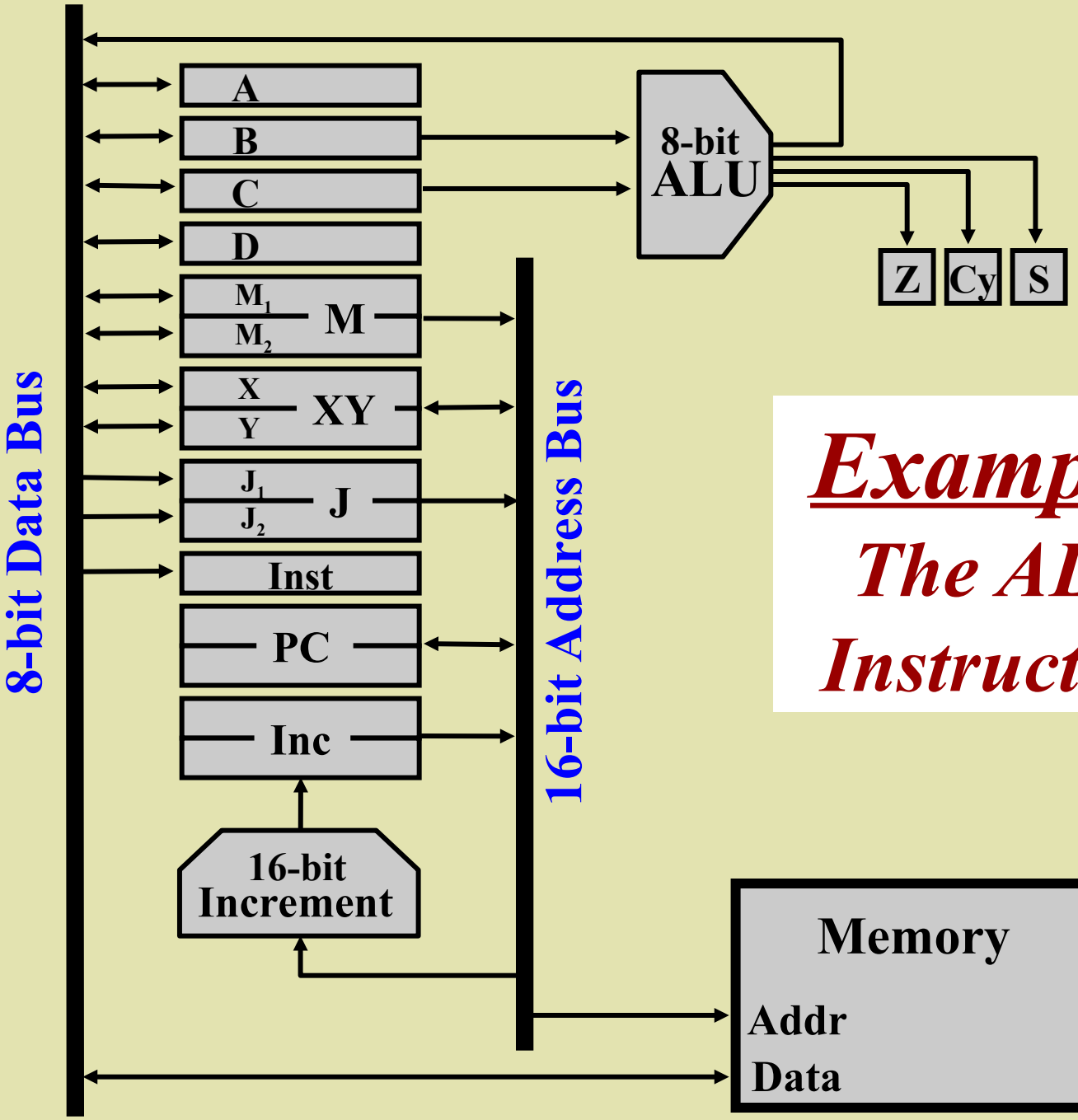
System Architecture



32K Byte Static RAM Chip

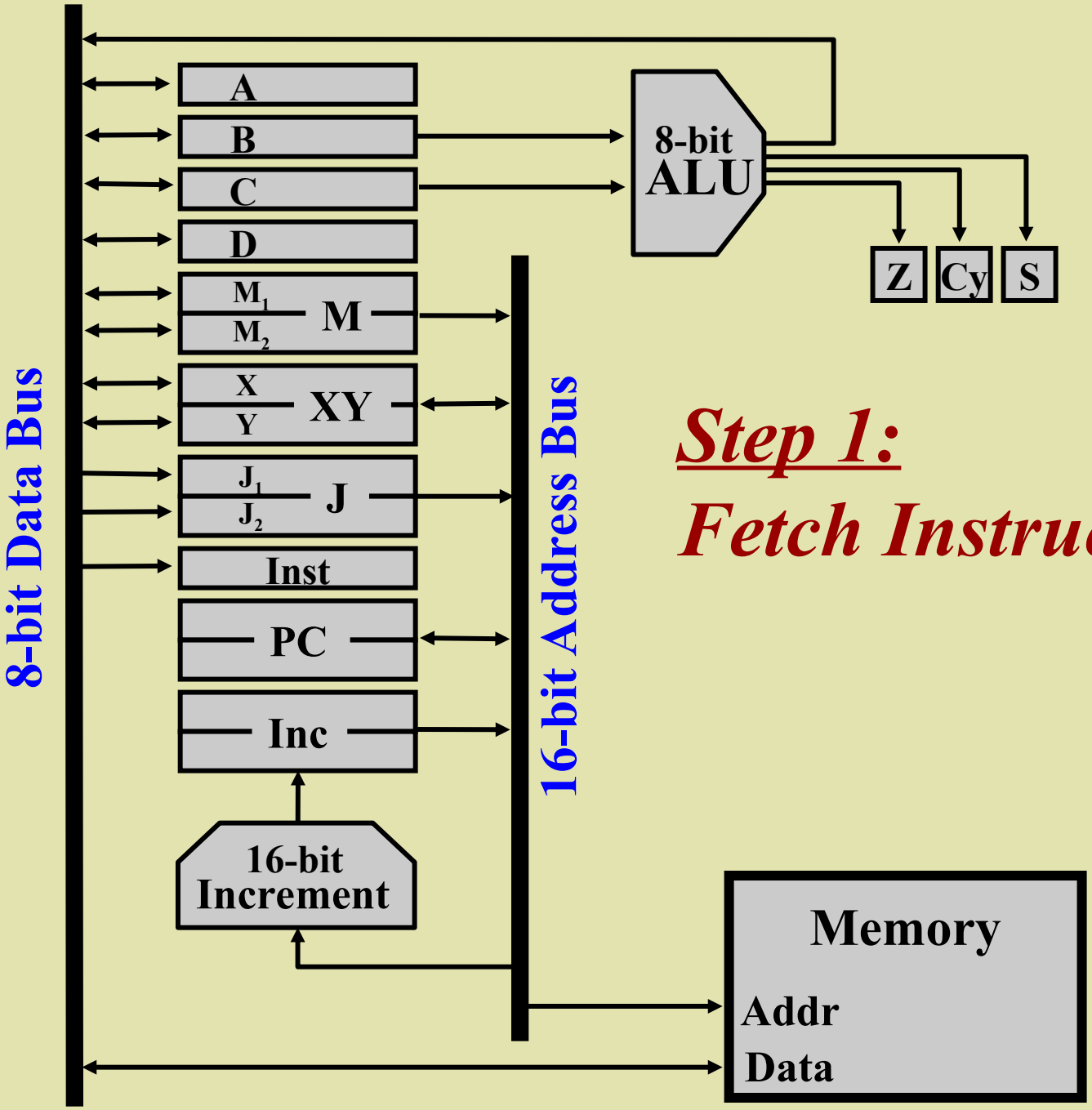
LEDs

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

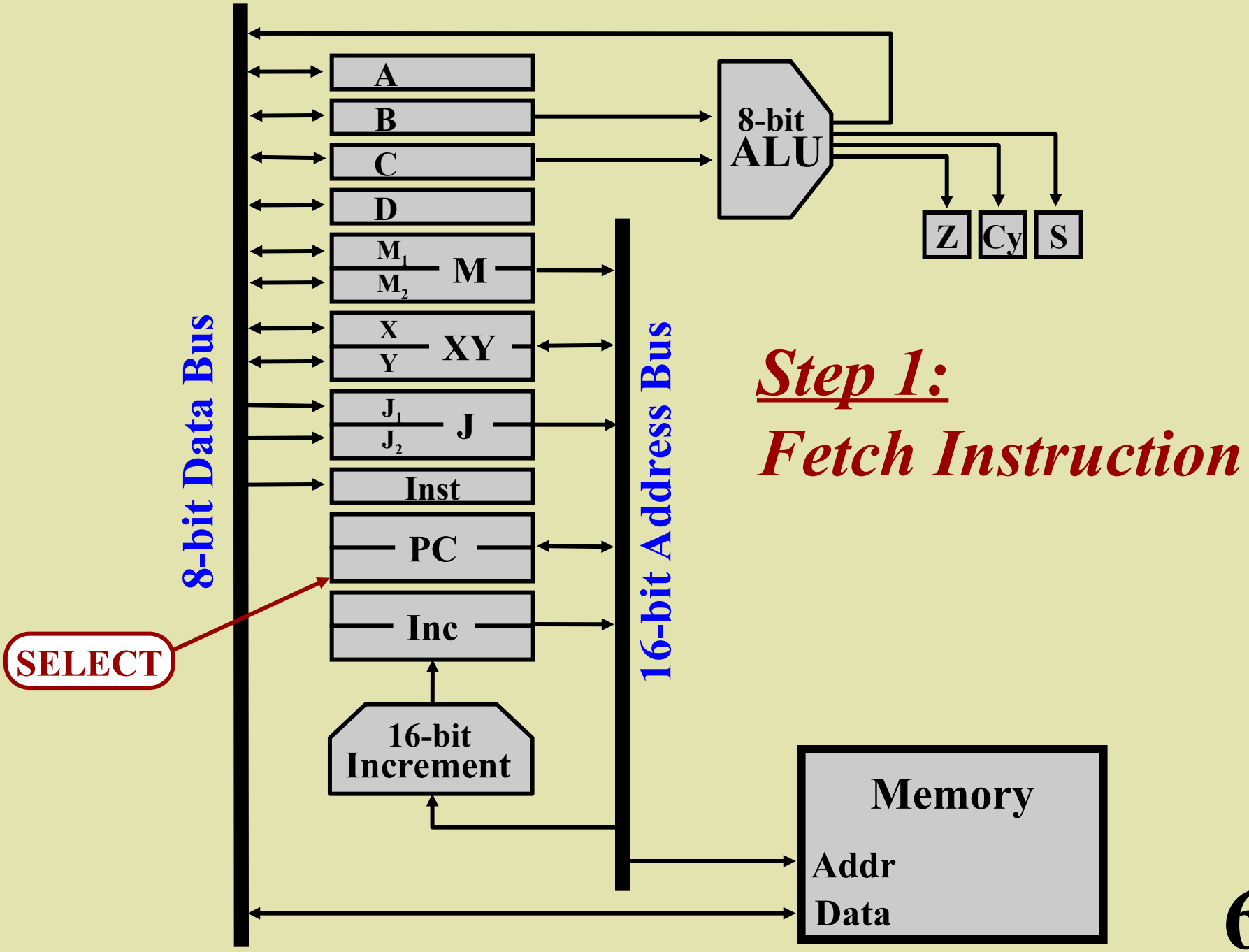


Example:

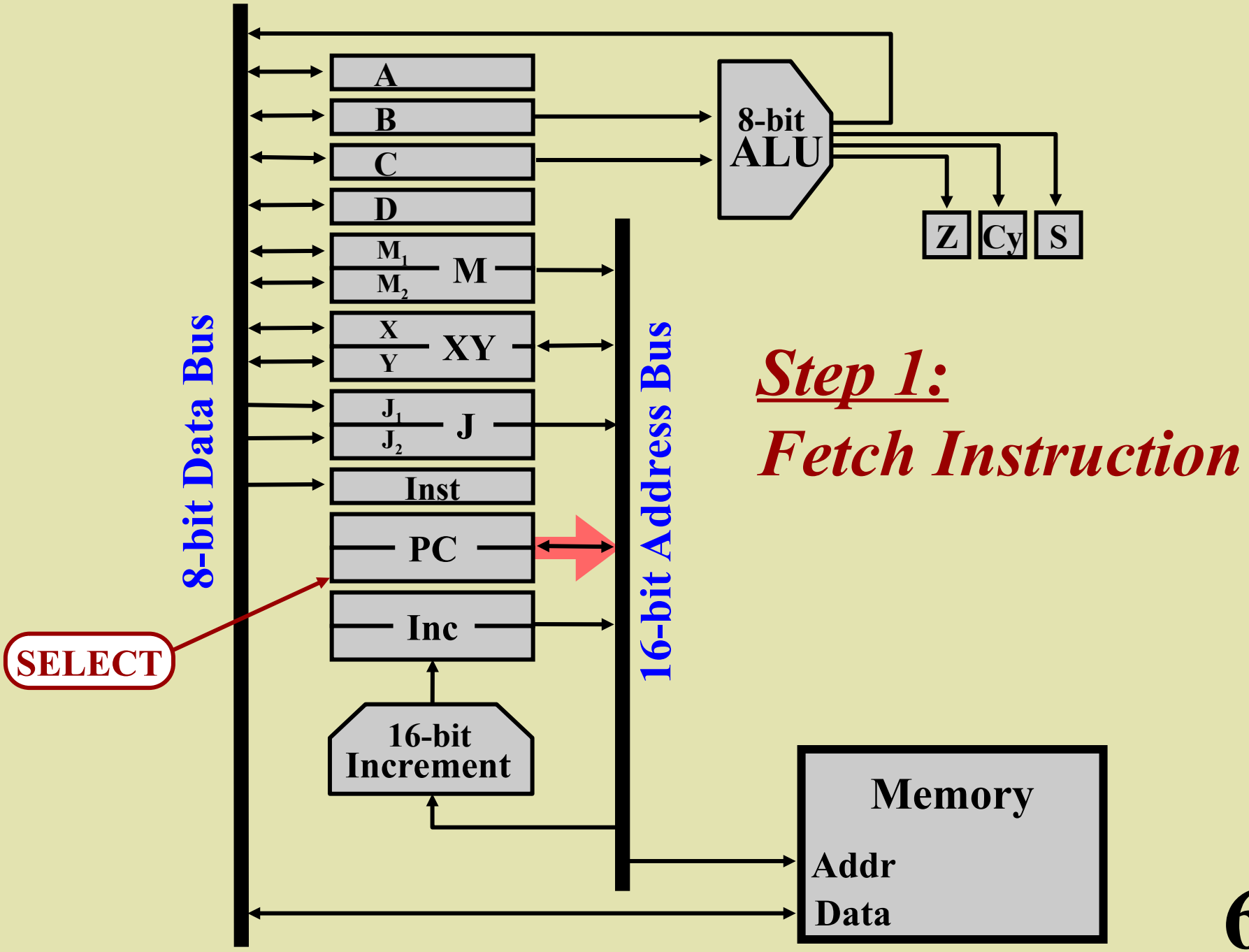
The ALU Instruction



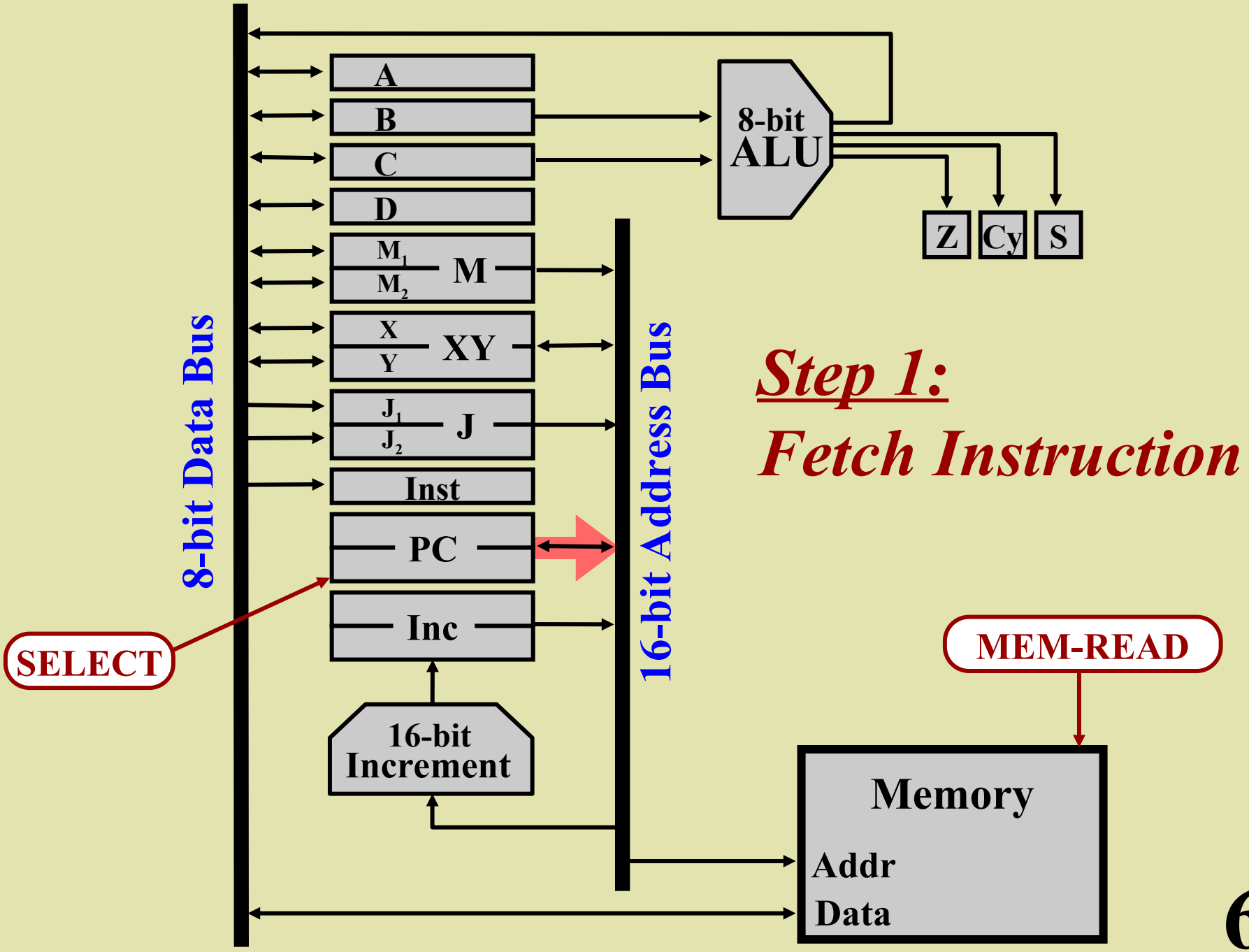
Step 1:
Fetch Instruction

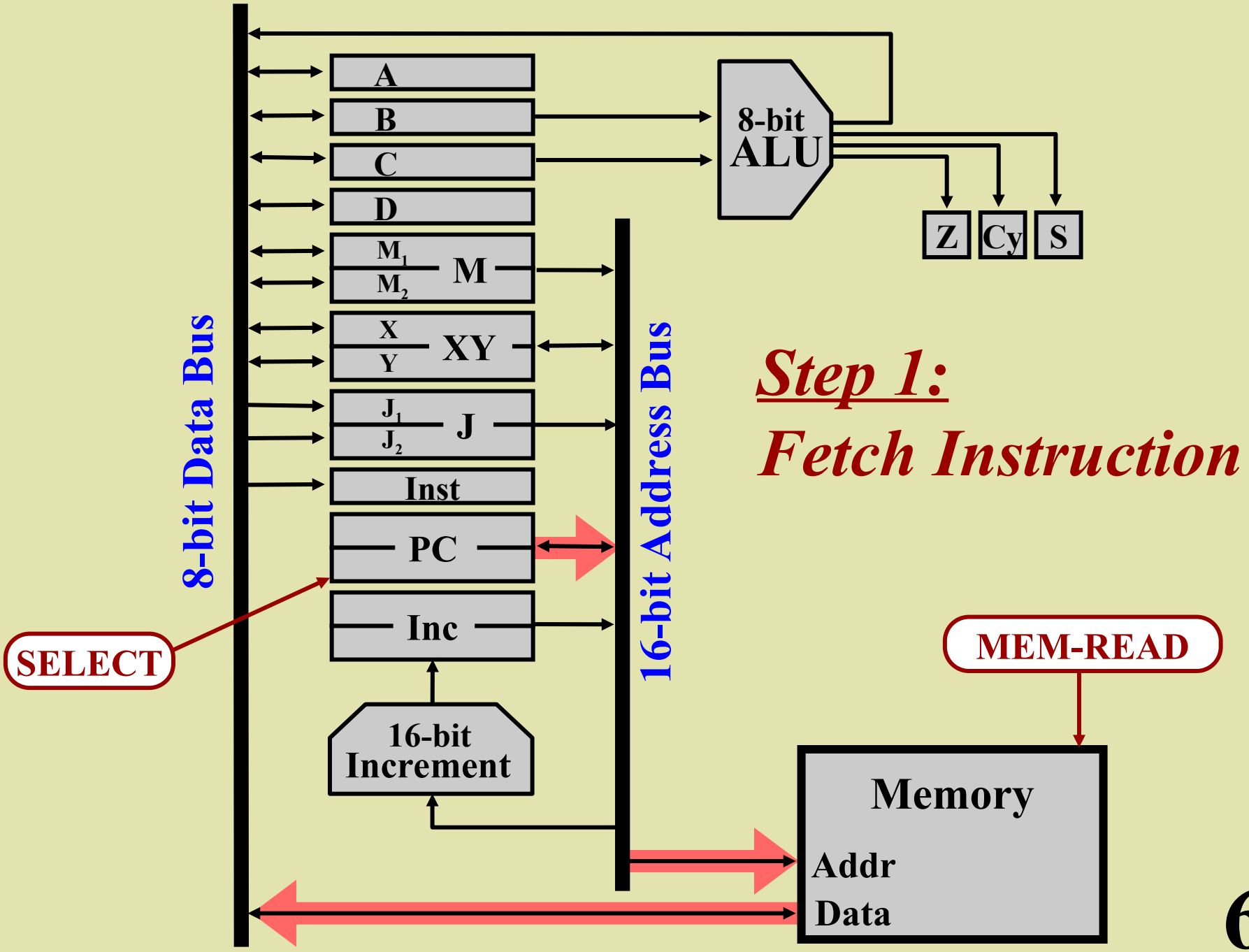


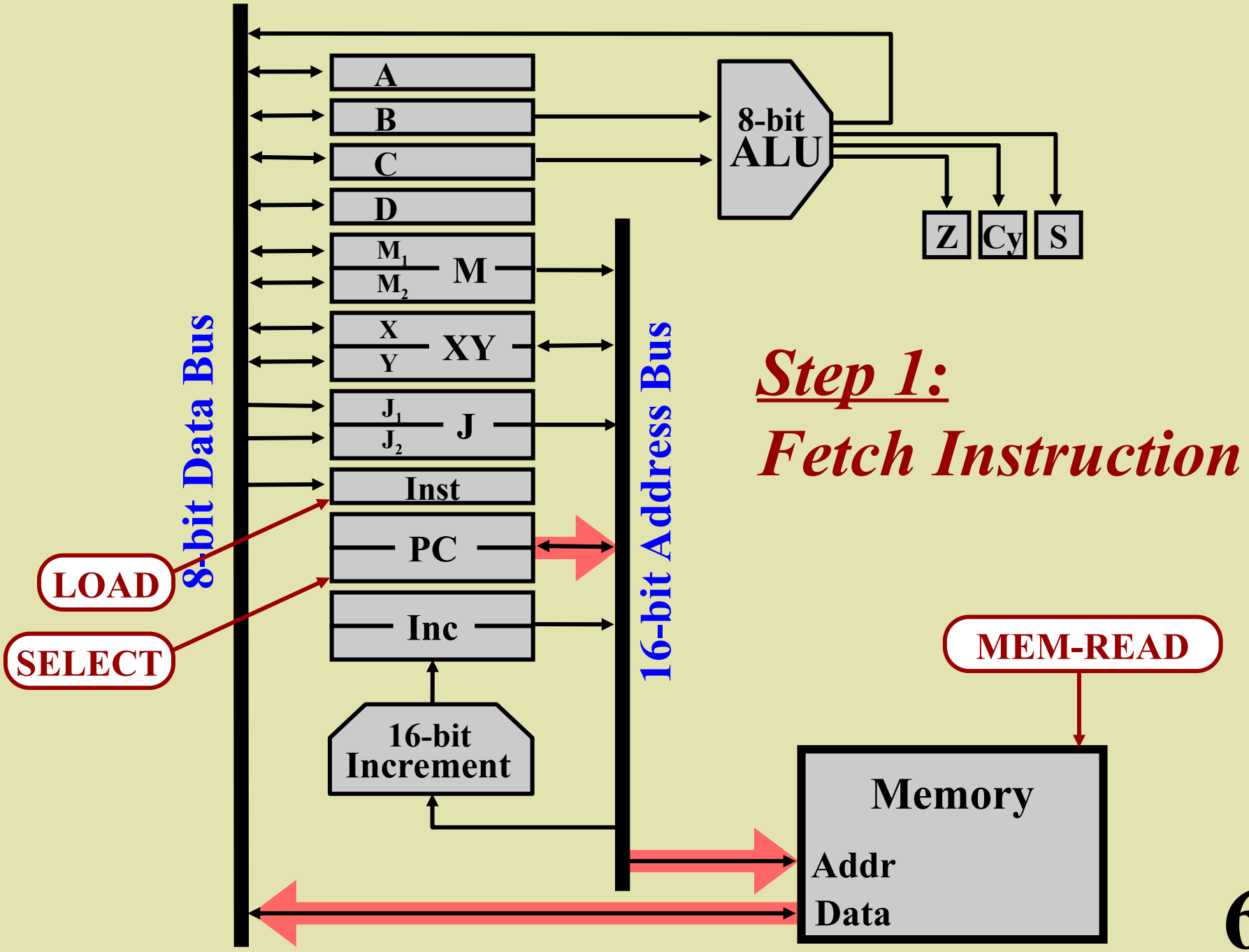
Step 1:
Fetch Instruction

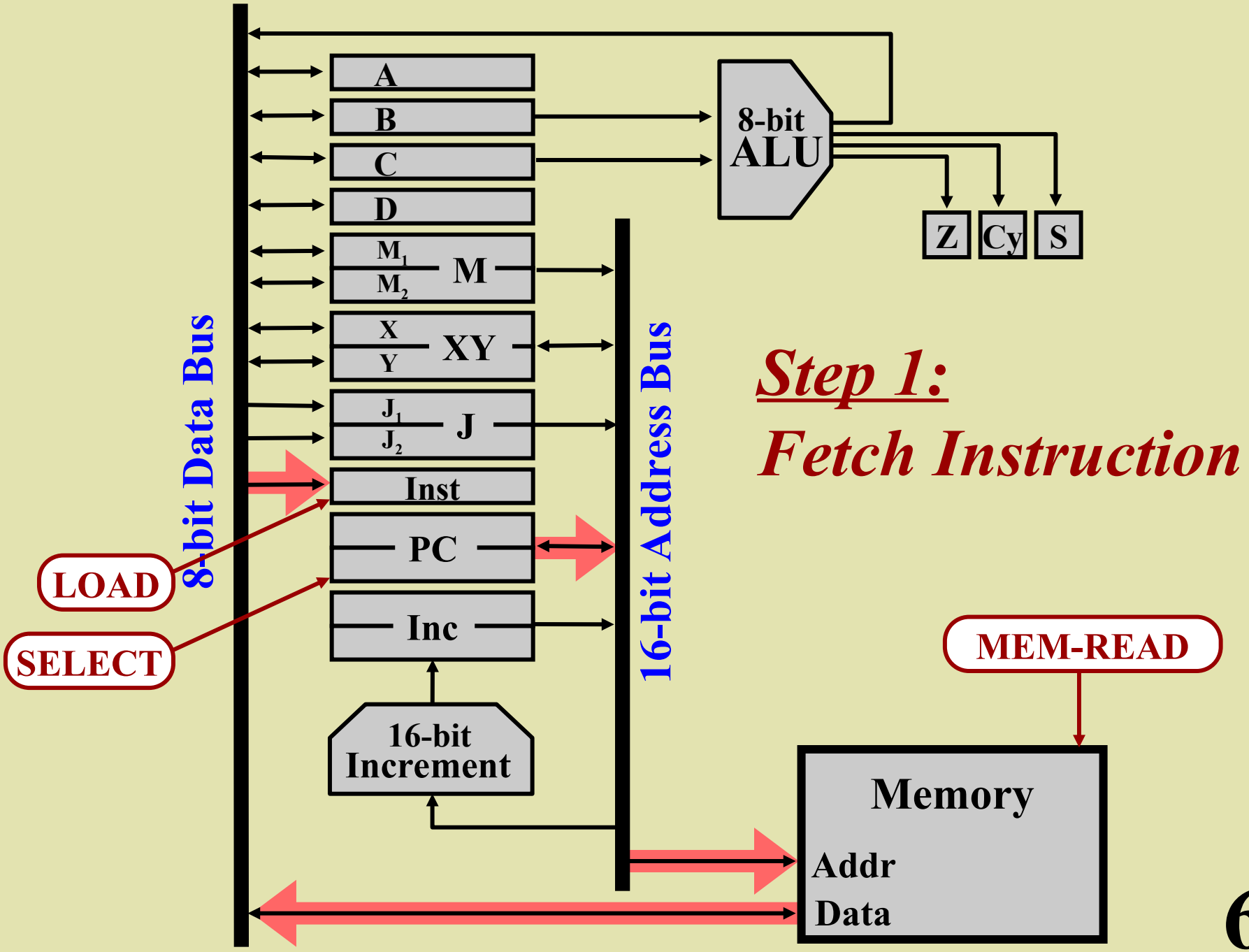


Step 1:
Fetch Instruction


A
B
C
D
M1
M2
M
X
Y
XY
J1
J2
J
Inst
PC
Inc
8-bit Data Bus
16-bit Address Bus
8-bit ALU
Z
Cy
S
SELECT
16-bit Increment
MEM-READ
Memory
Addr
Data
Step 1:
Fetch Instruction

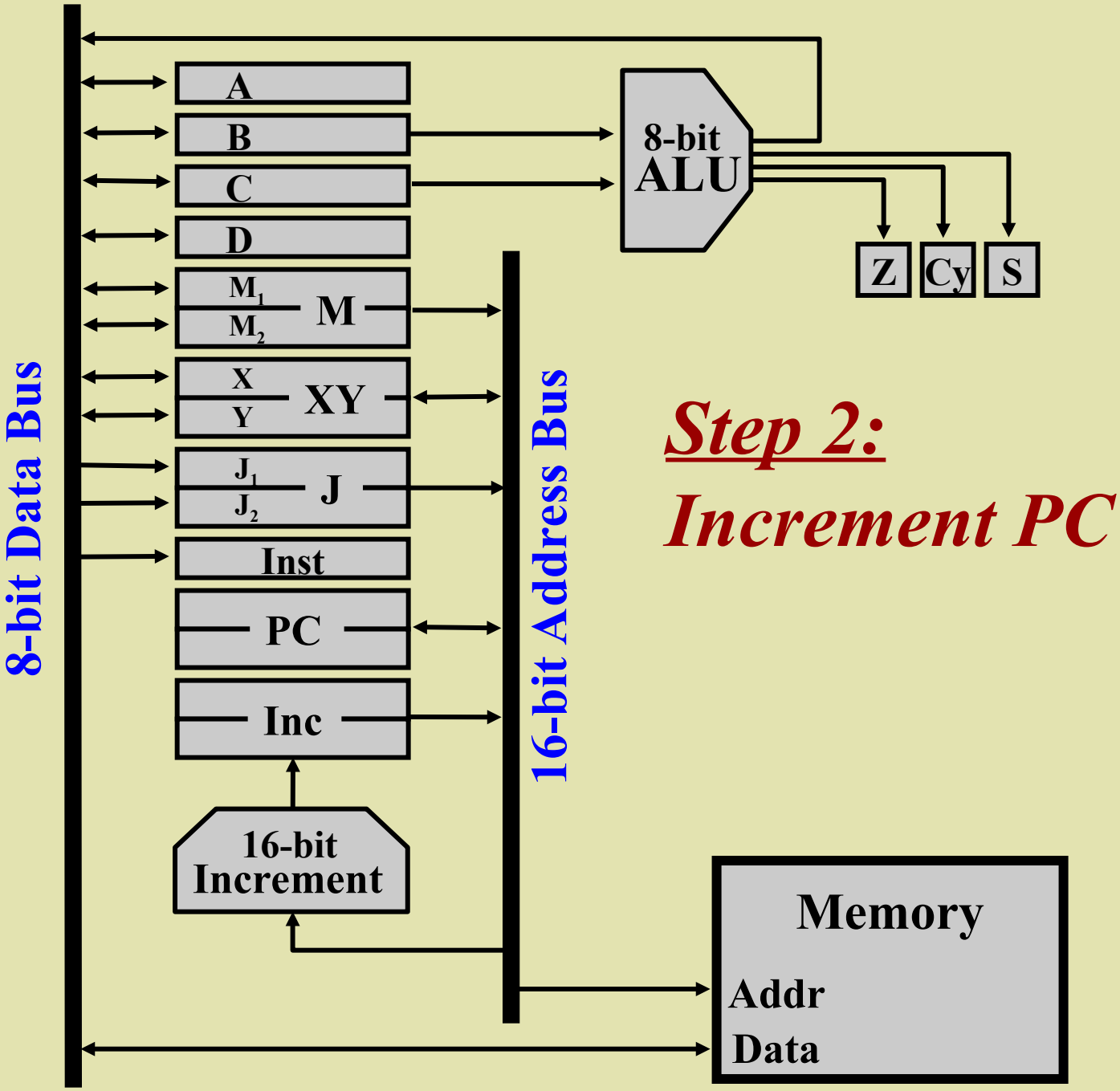

A
B
C
D
M1
M2
M
X
Y
XY
J1
J2
J
Inst
PC
Inc
8-bit Data Bus
16-bit Address Bus
8-bit ALU
Z
Cy
S
SELECT
16-bit Increment
MEM-READ
Memory
Addr
Data
Step 1:
Fetch Instruction


A
B
C
D
M1
M2
M
X
Y
XY
J1
J2
J
Inst
PC
Inc
8-bit ALU
Z
Cy
S
8-bit Data Bus
16-bit Address Bus
LOAD
SELECT
16-bit Increment
MEM-READ
Memory
Addr
Data
Step 1:
Fetch Instruction

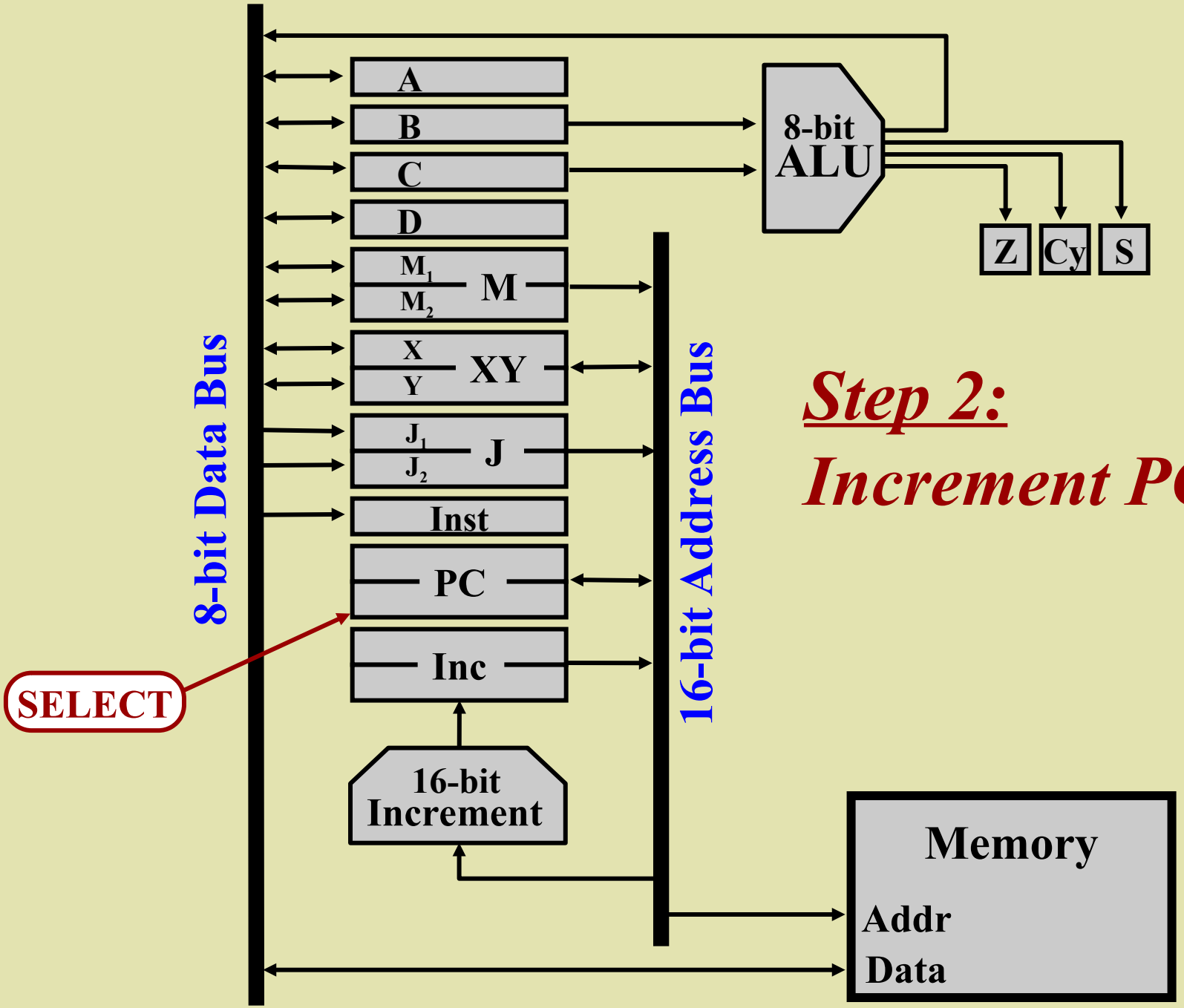


Step 1:

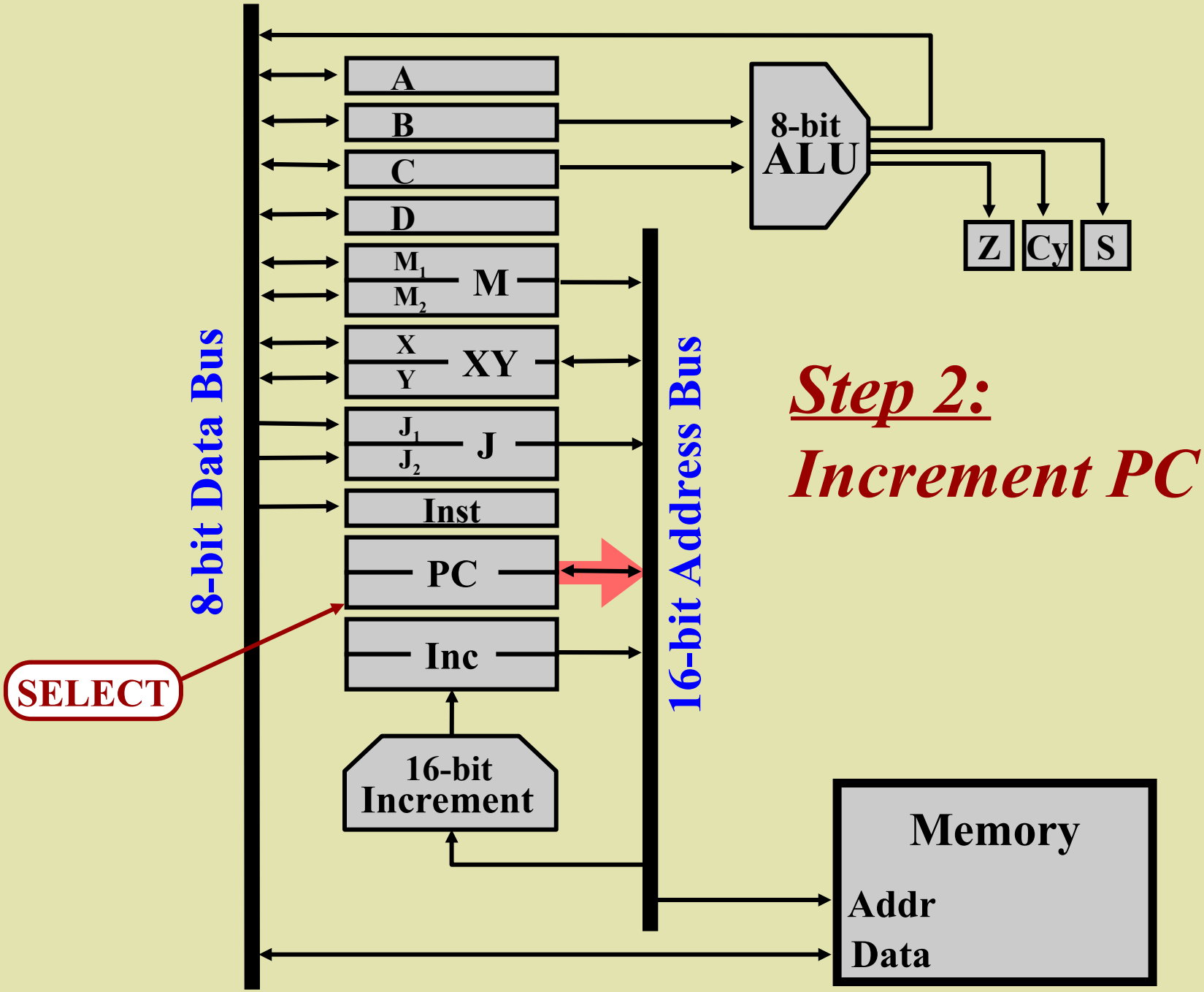
Fetch Instruction



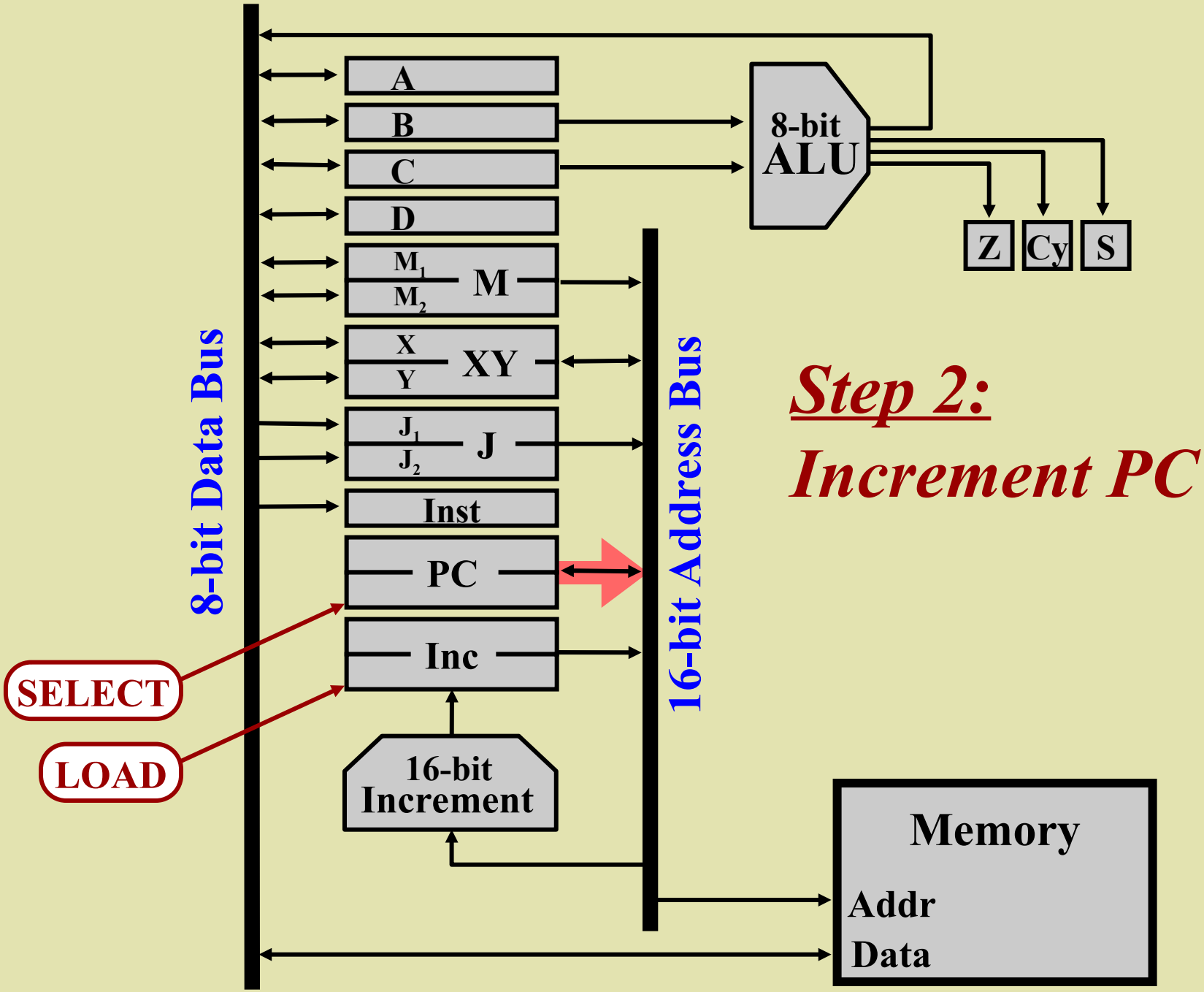
Step 2:
Increment PC



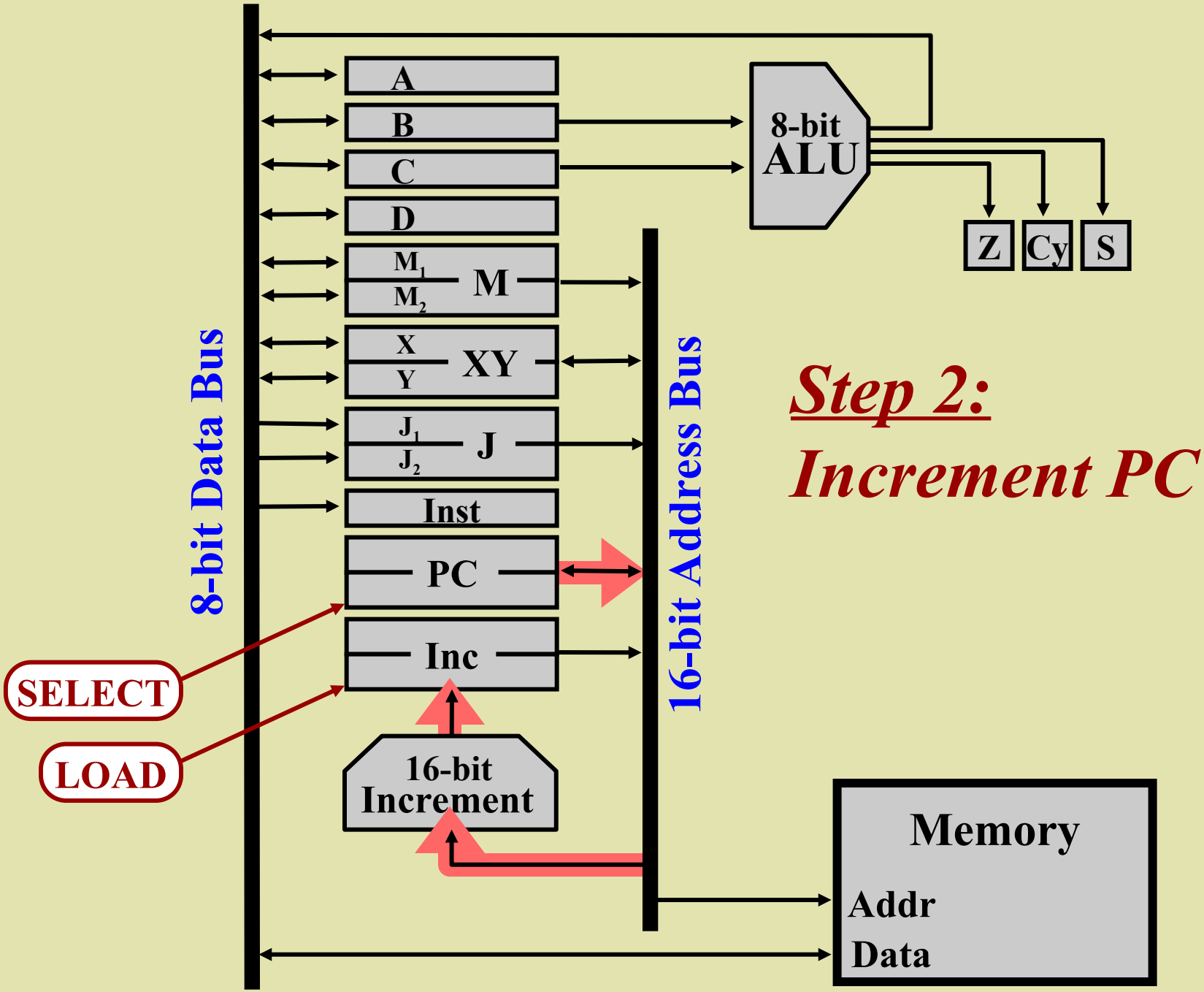
Step 2:
Increment PC



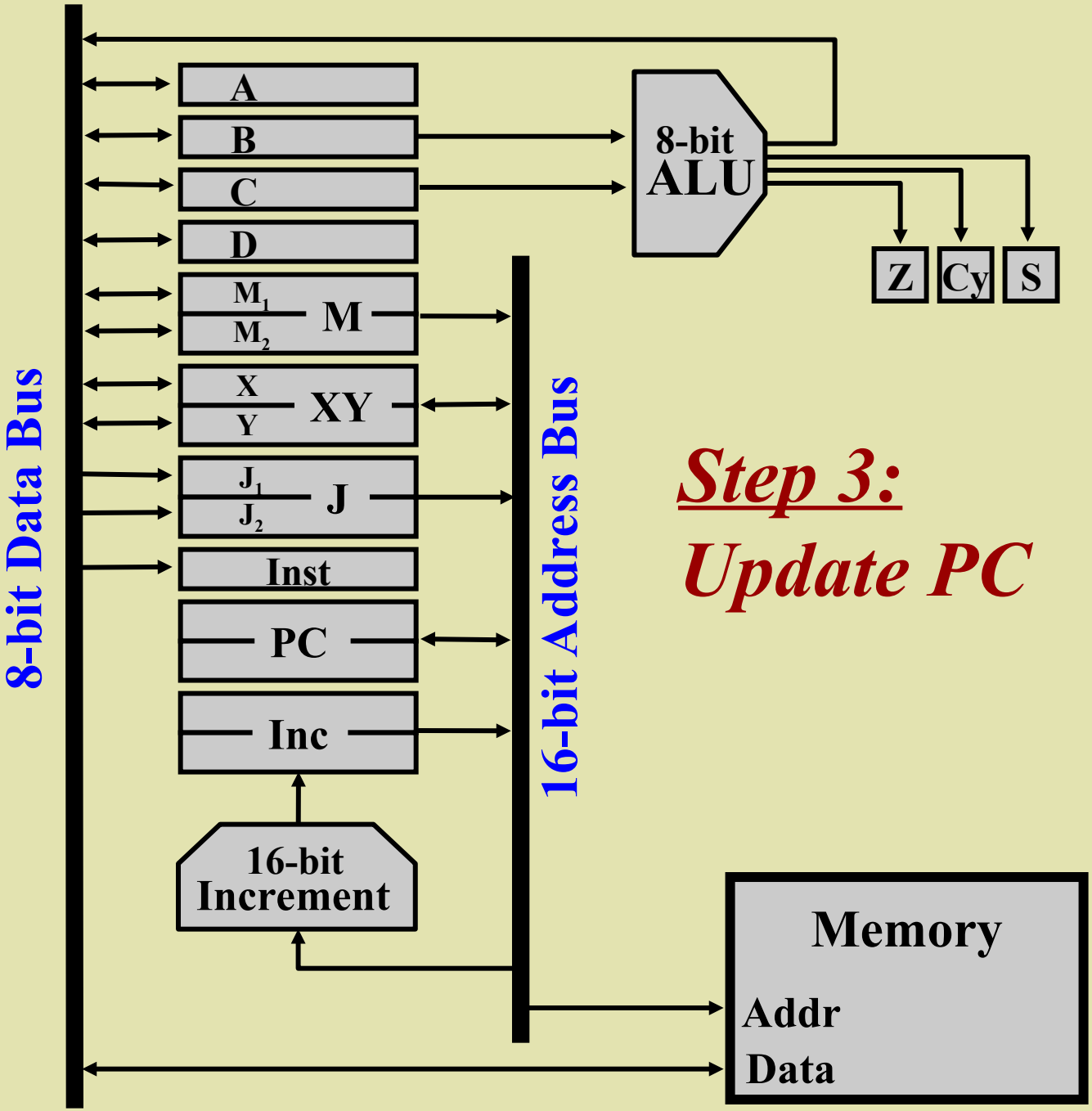
Step 2:
Increment PC



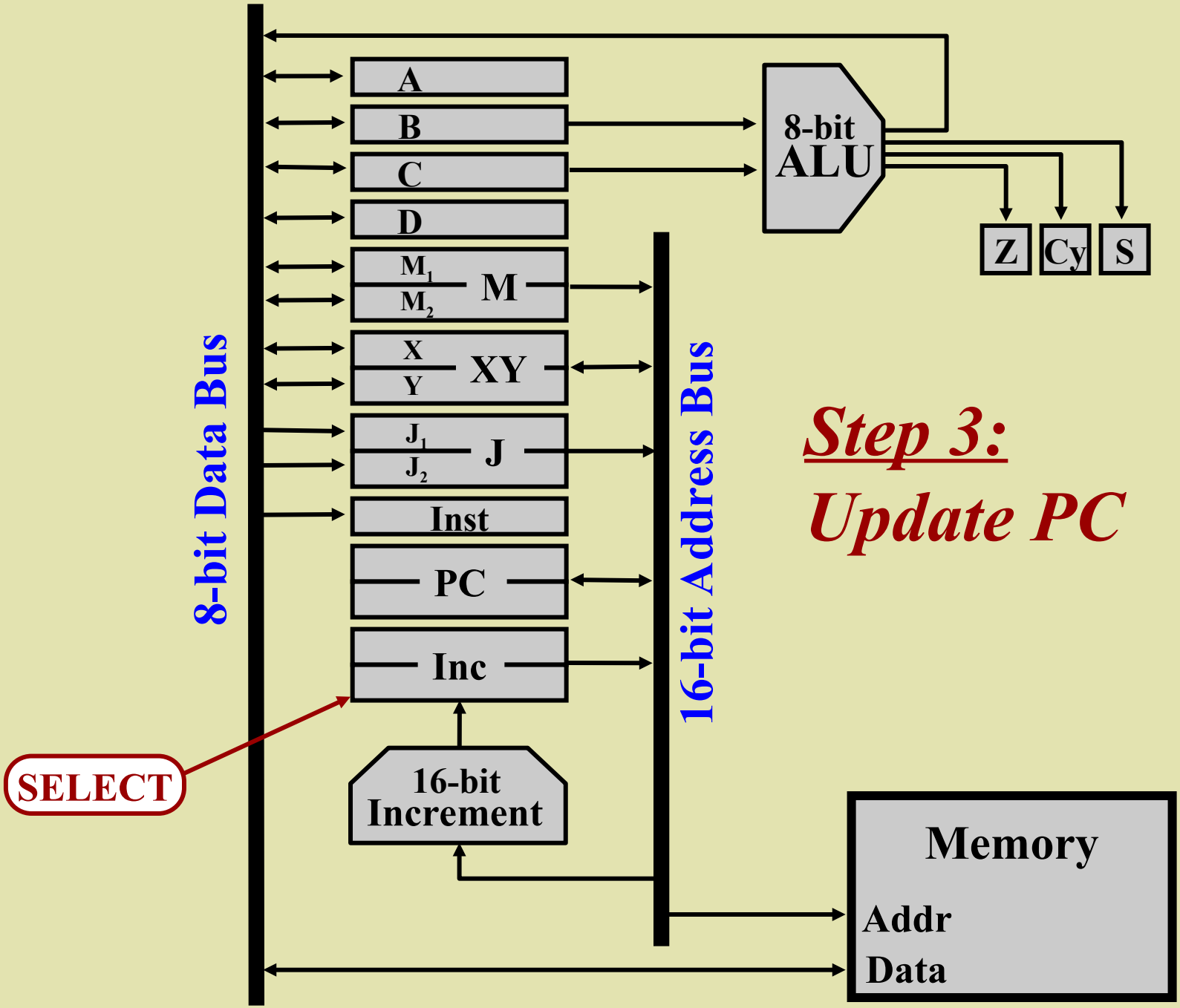
Step 2:
Increment PC



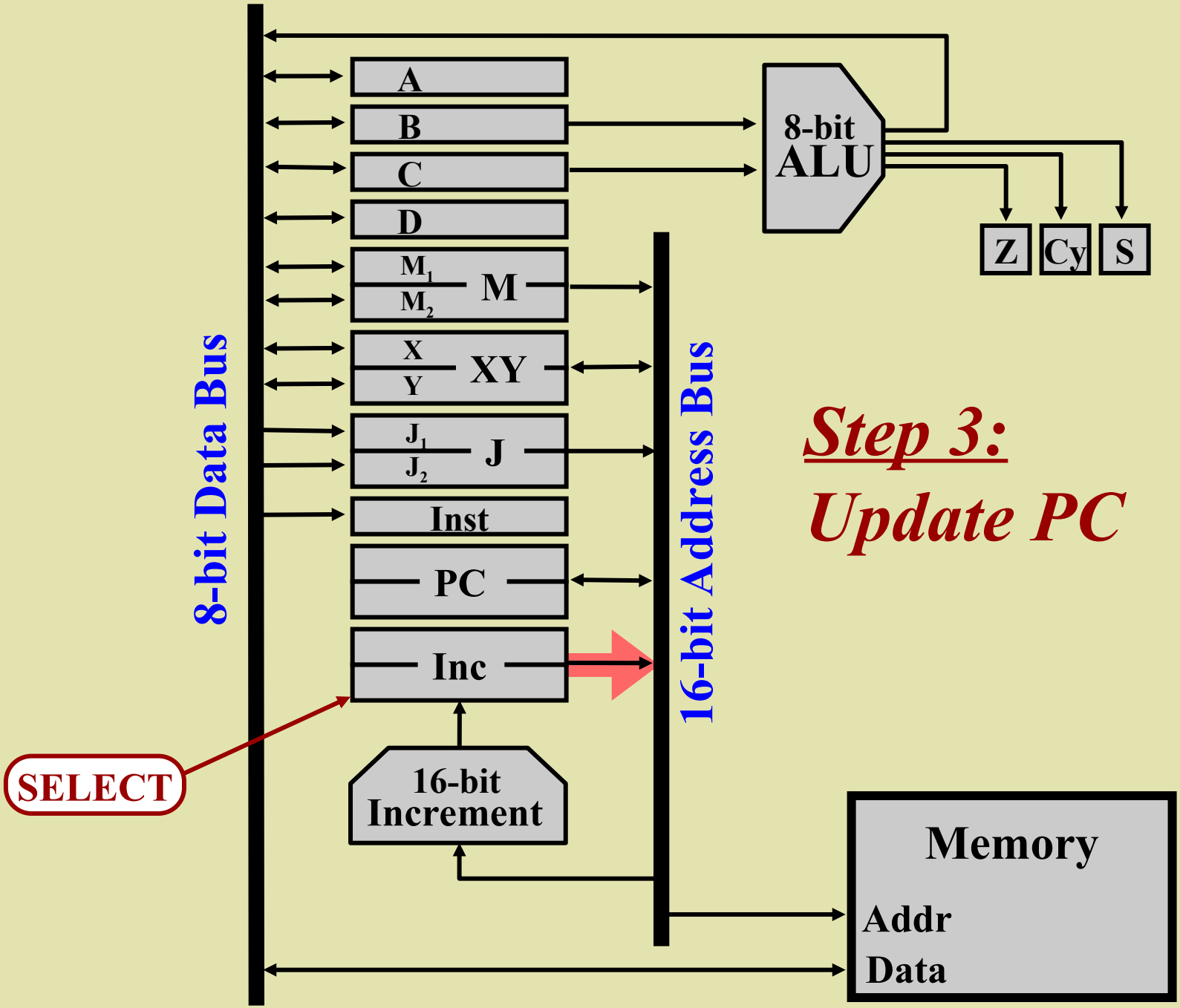
Step 2:
Increment PC



Step 3:
Update PC

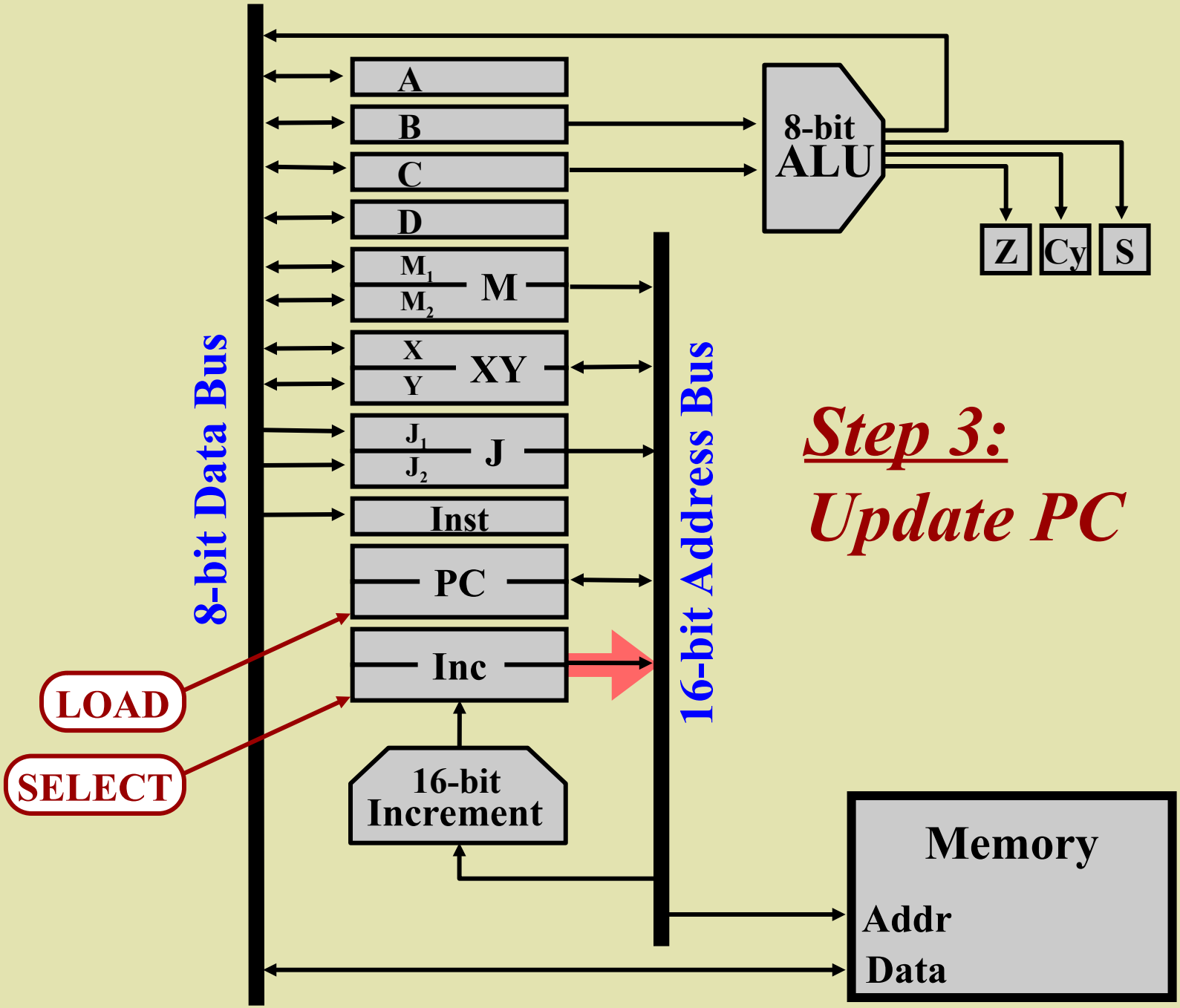


Step 3: Update PC



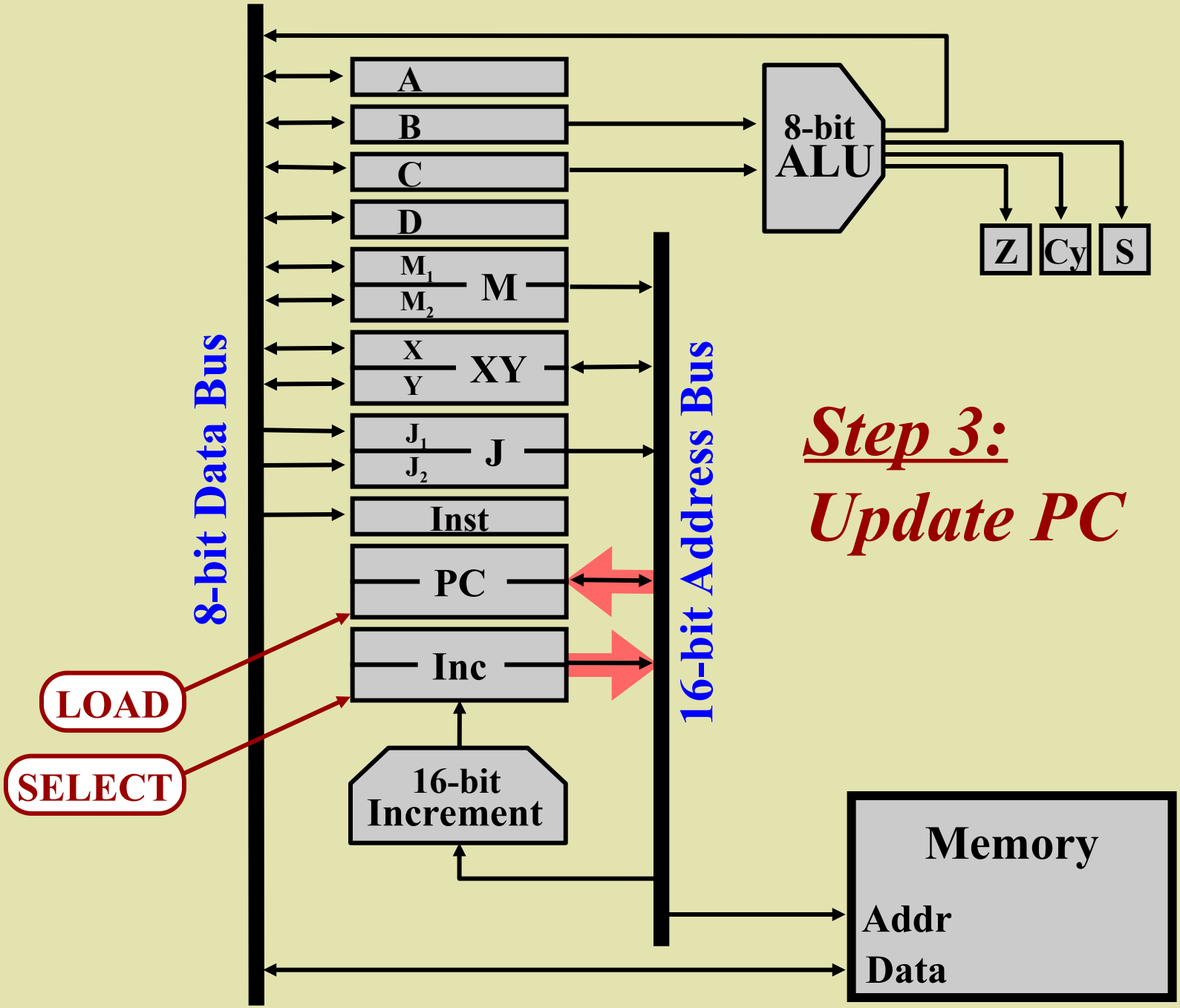
Step 3:

Update PC



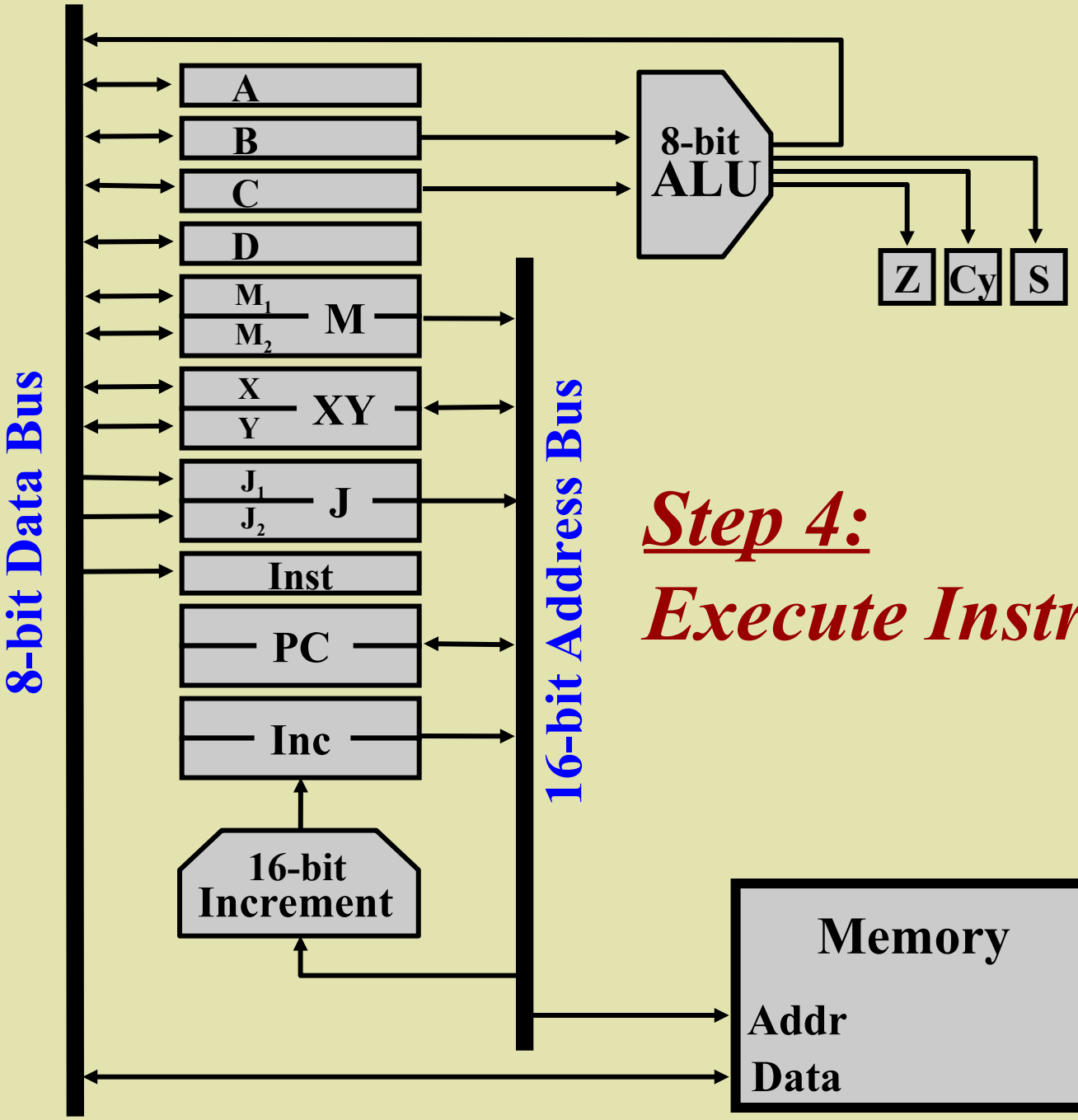
Step 3:

Update PC

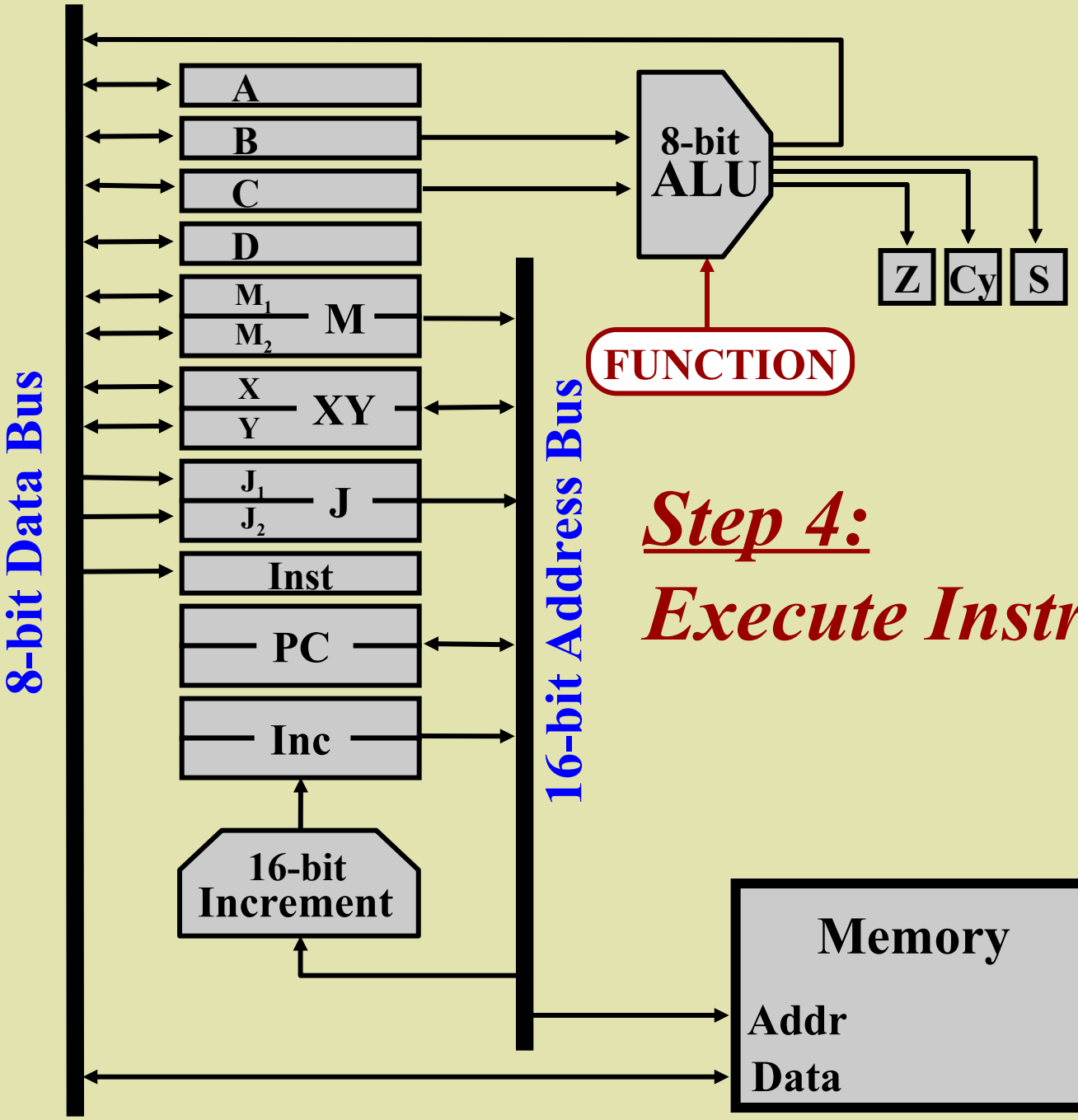


Step 3:

Update PC

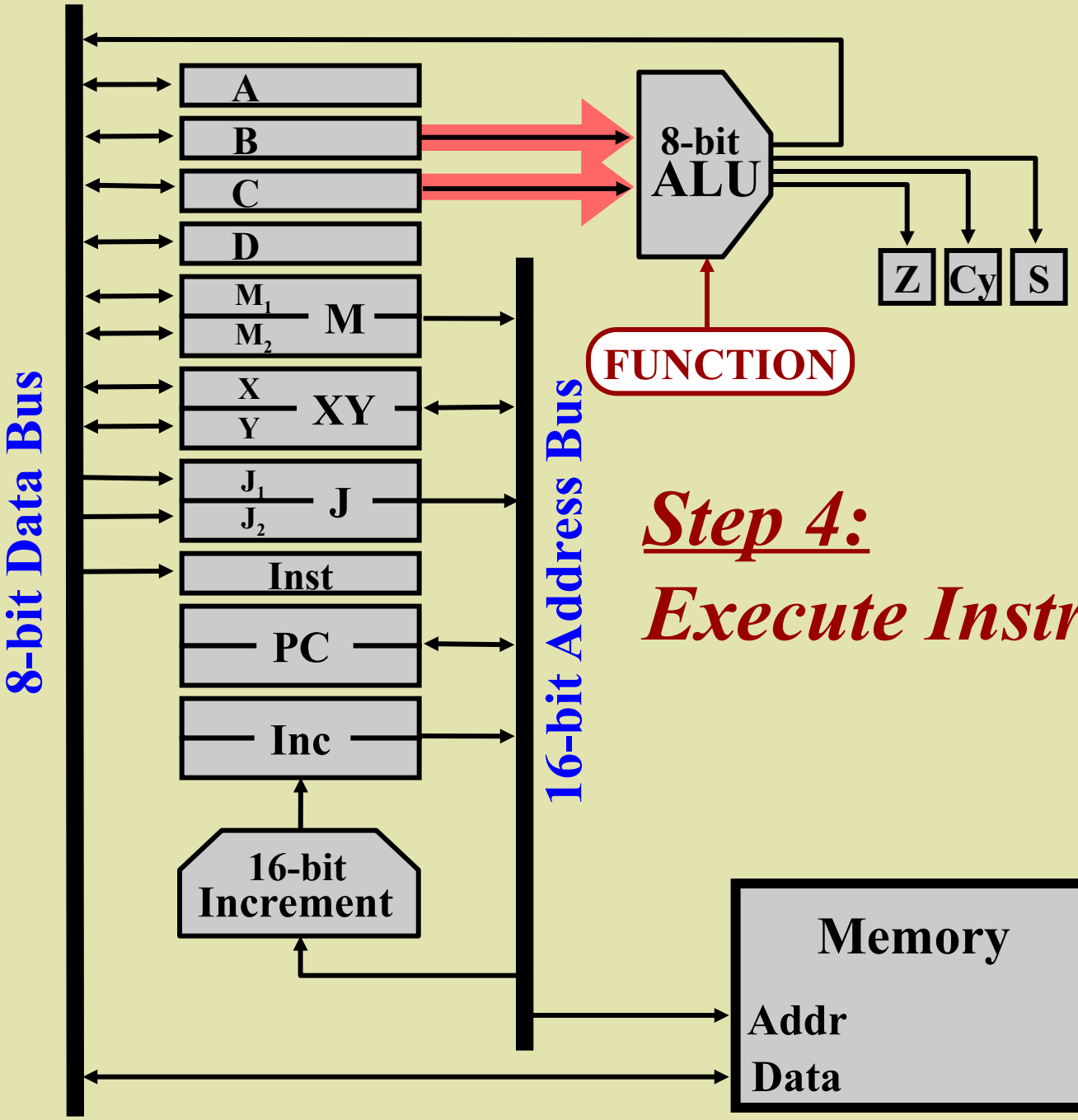


Step 4:
Execute Instruction



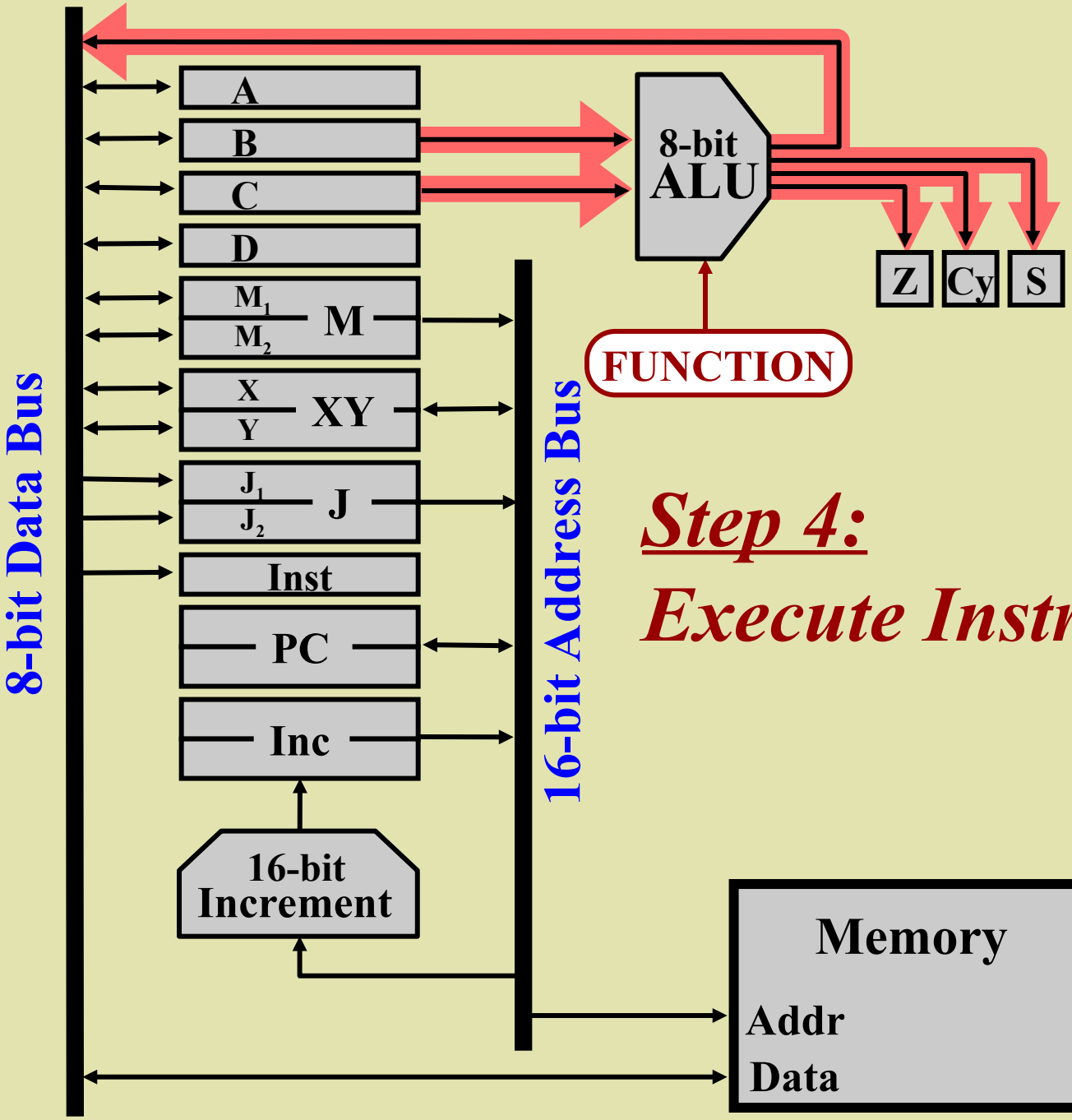
Step 4:

Execute Instruction



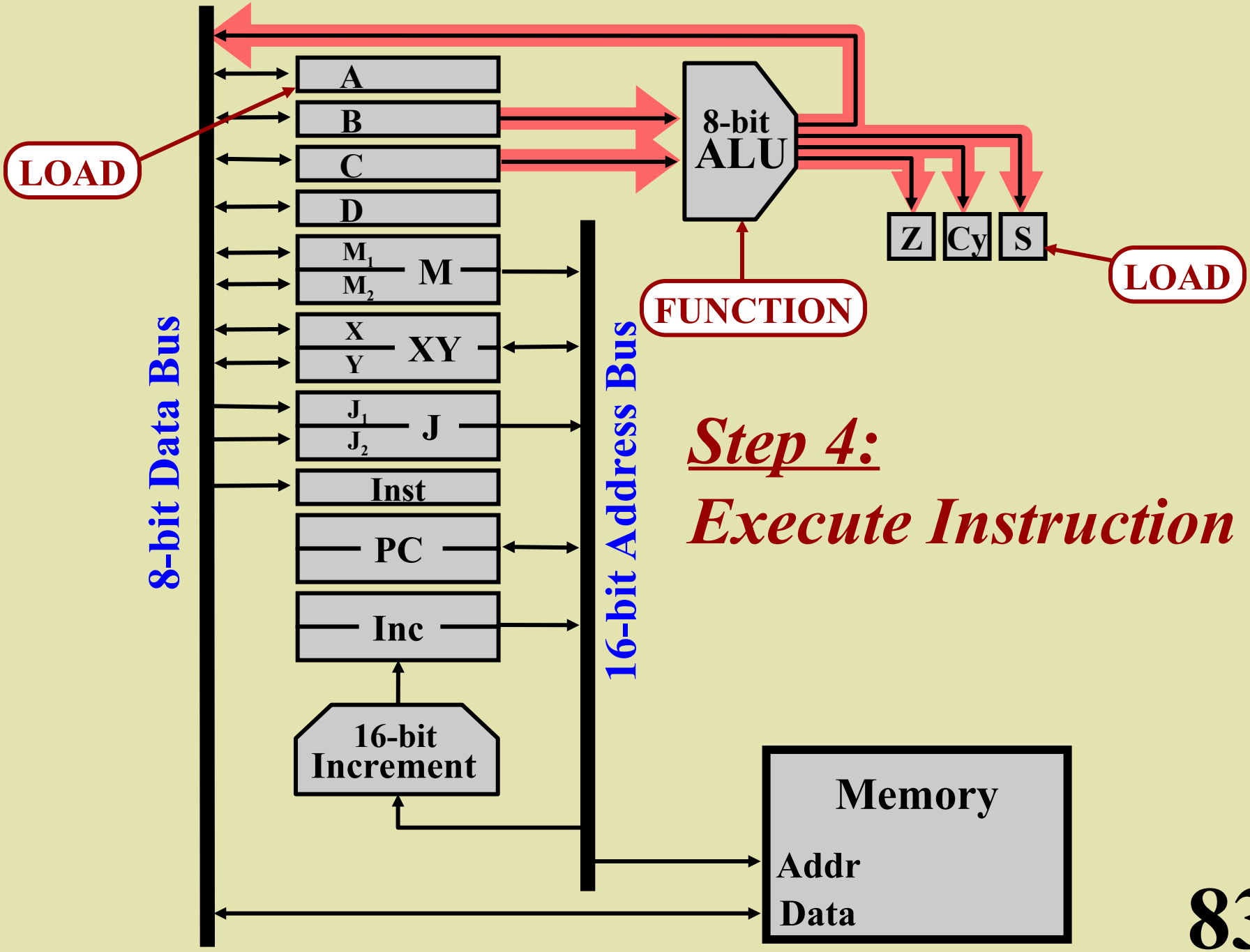
Step 4:

Execute Instruction



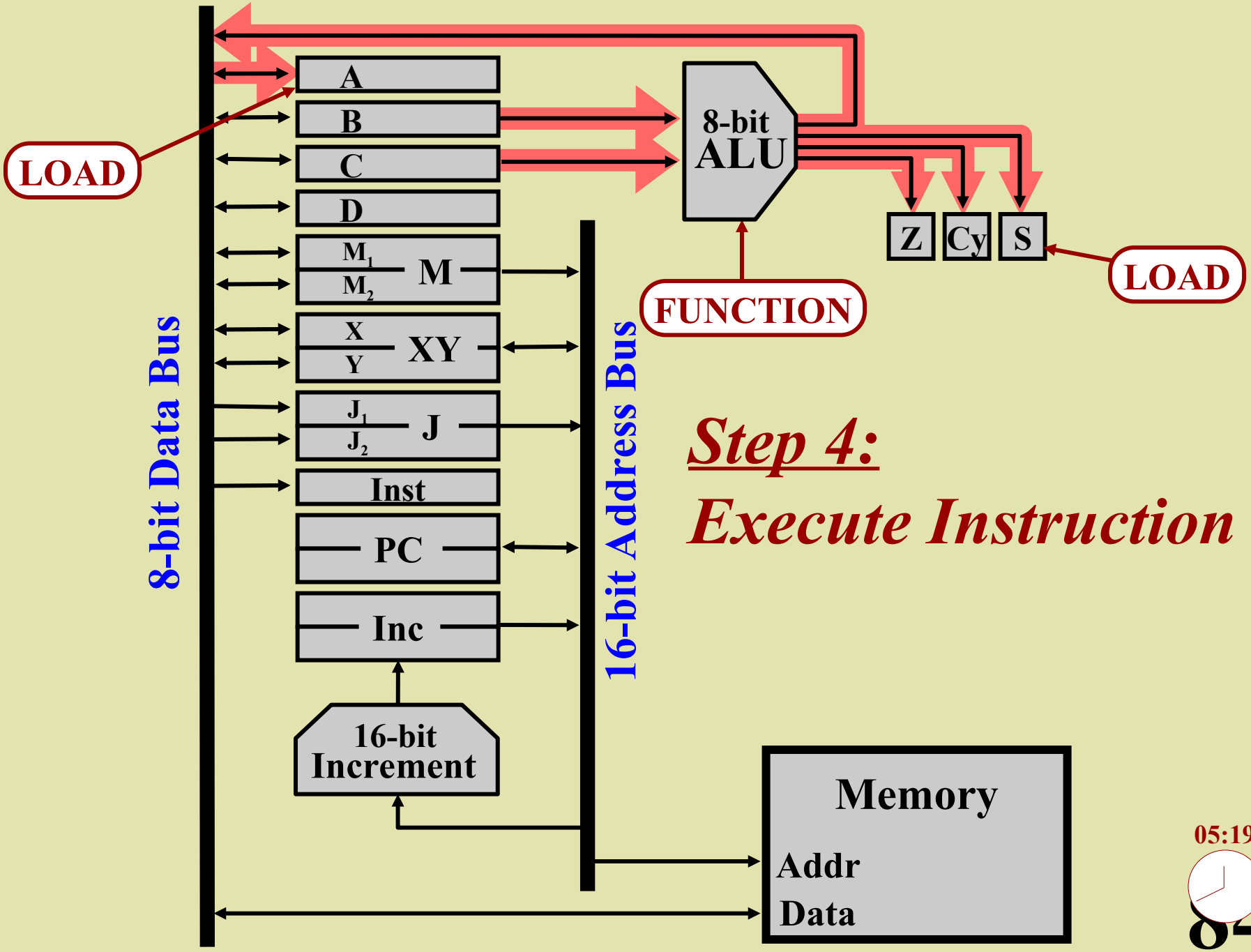
Step 4:

Execute Instruction



Step 4:

Execute Instruction

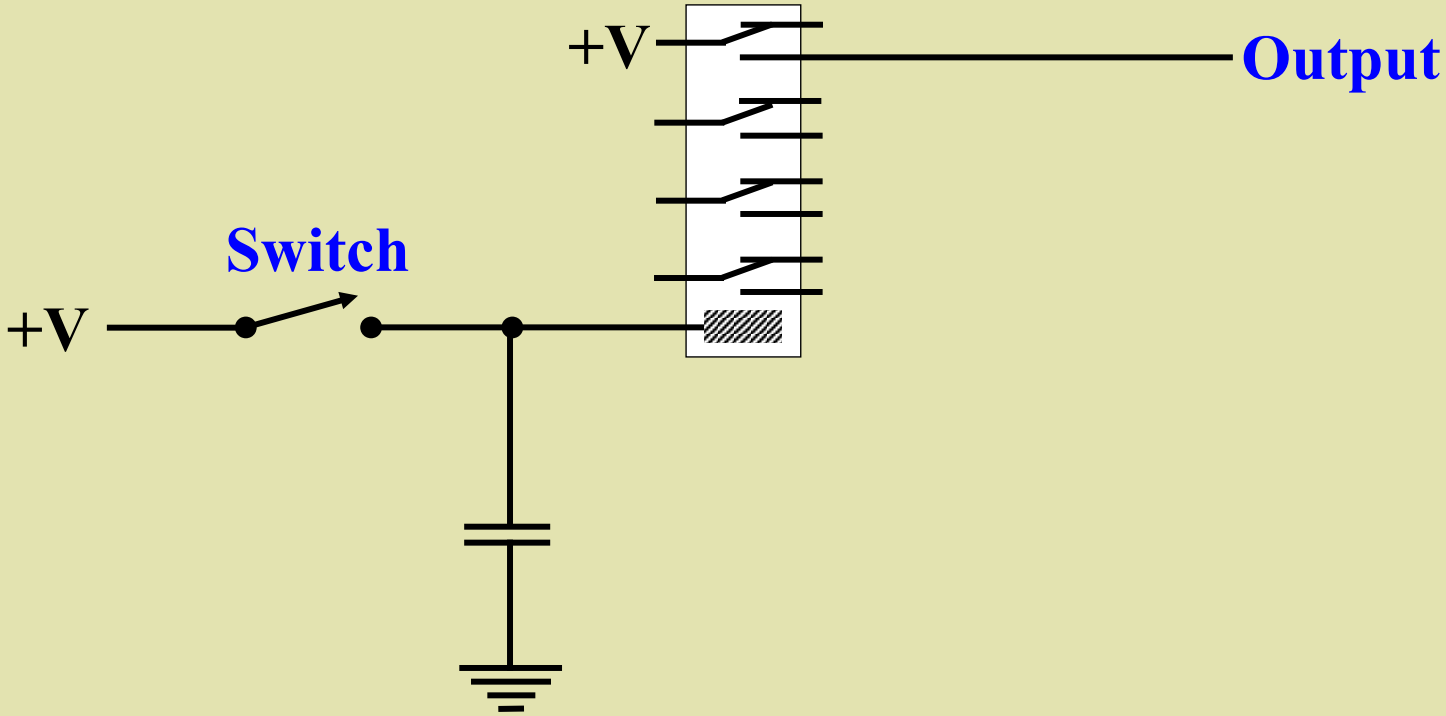


Step 4:

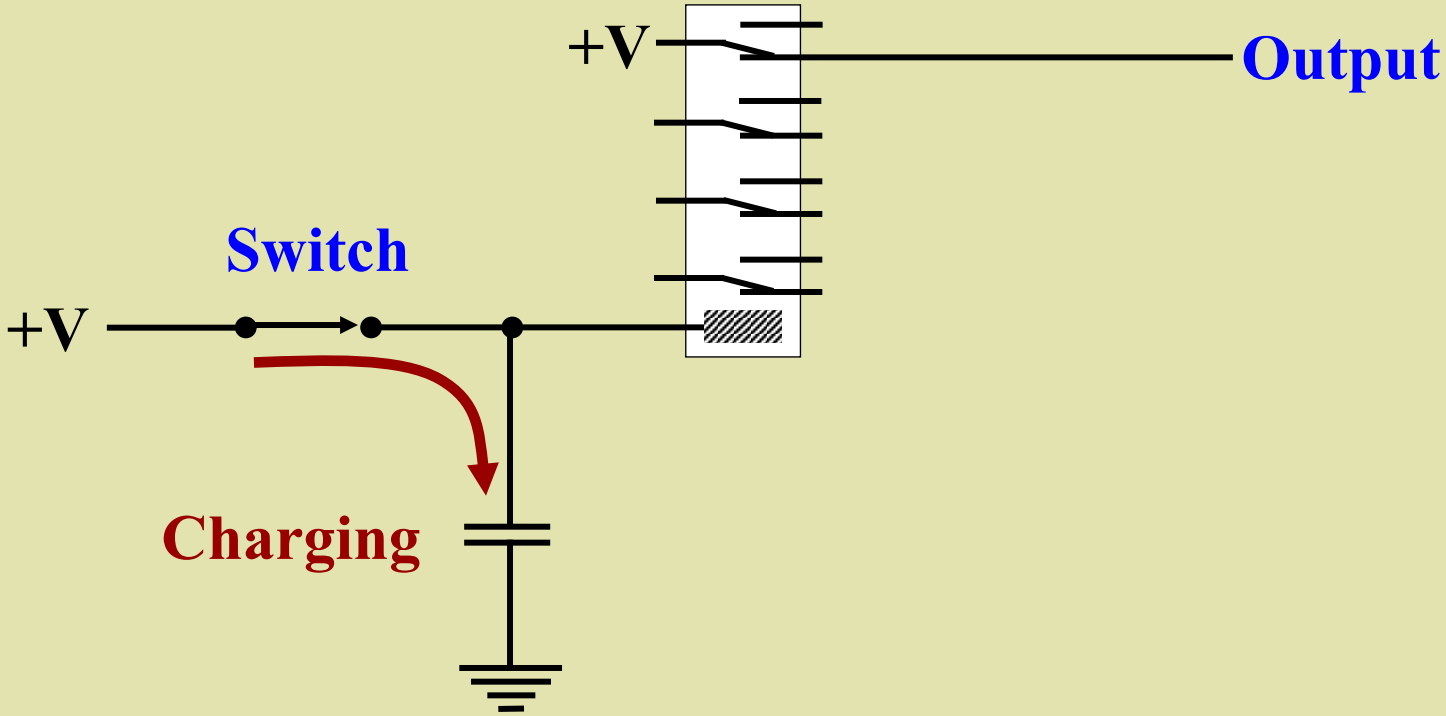
Execute Instruction

05:19

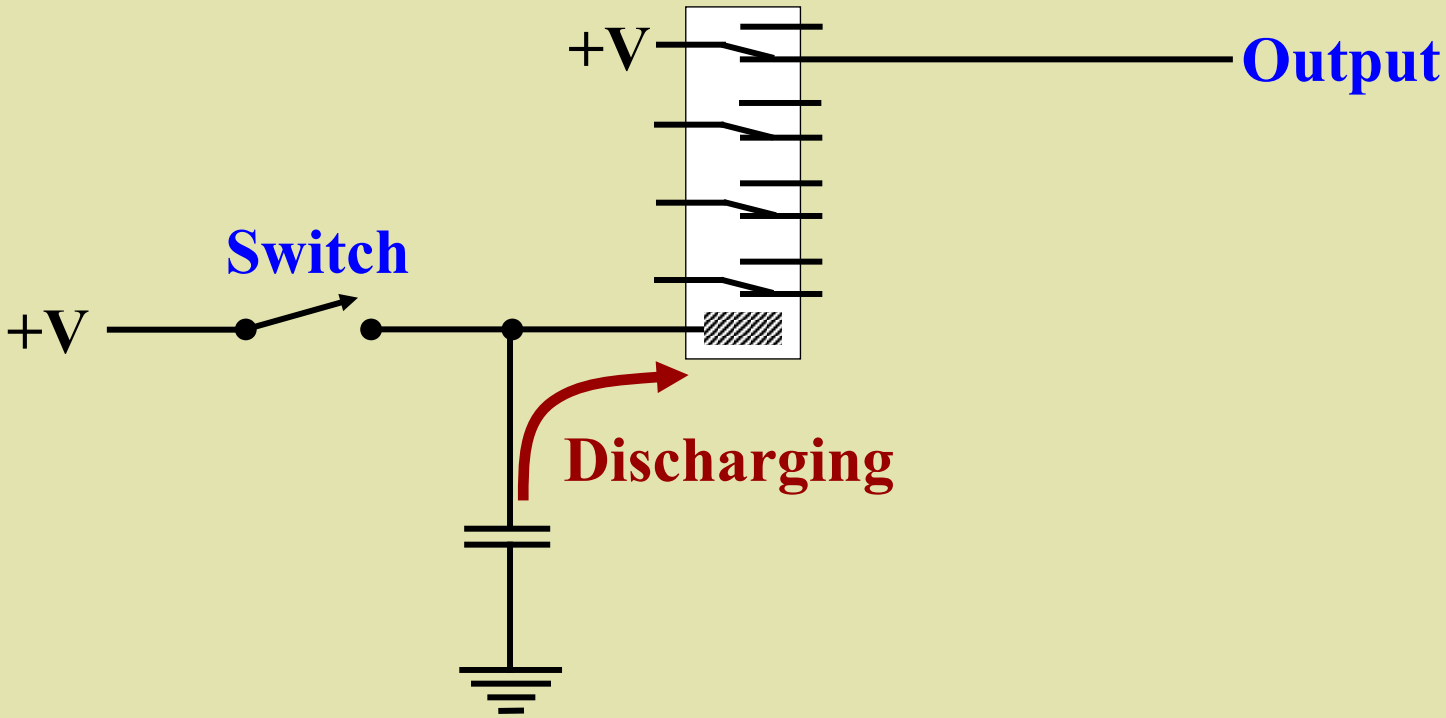
Clock



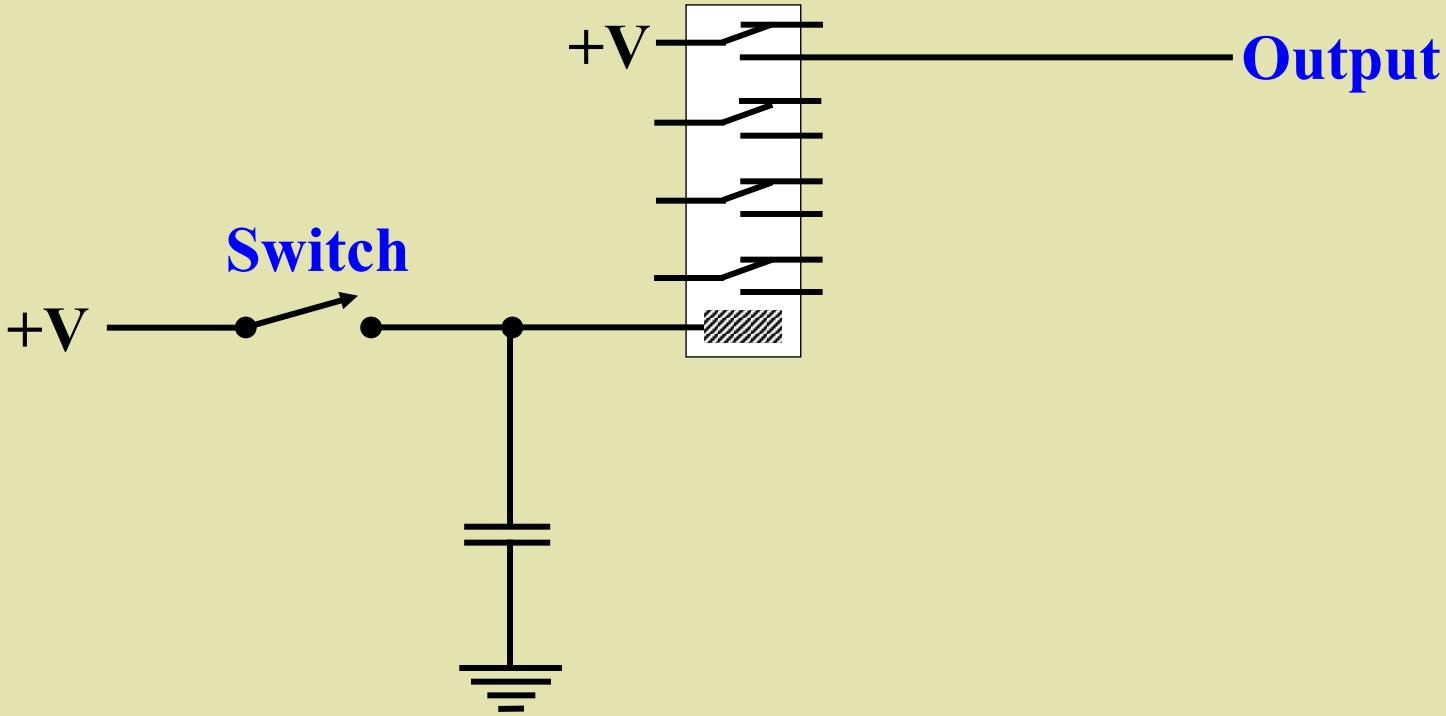
Clock



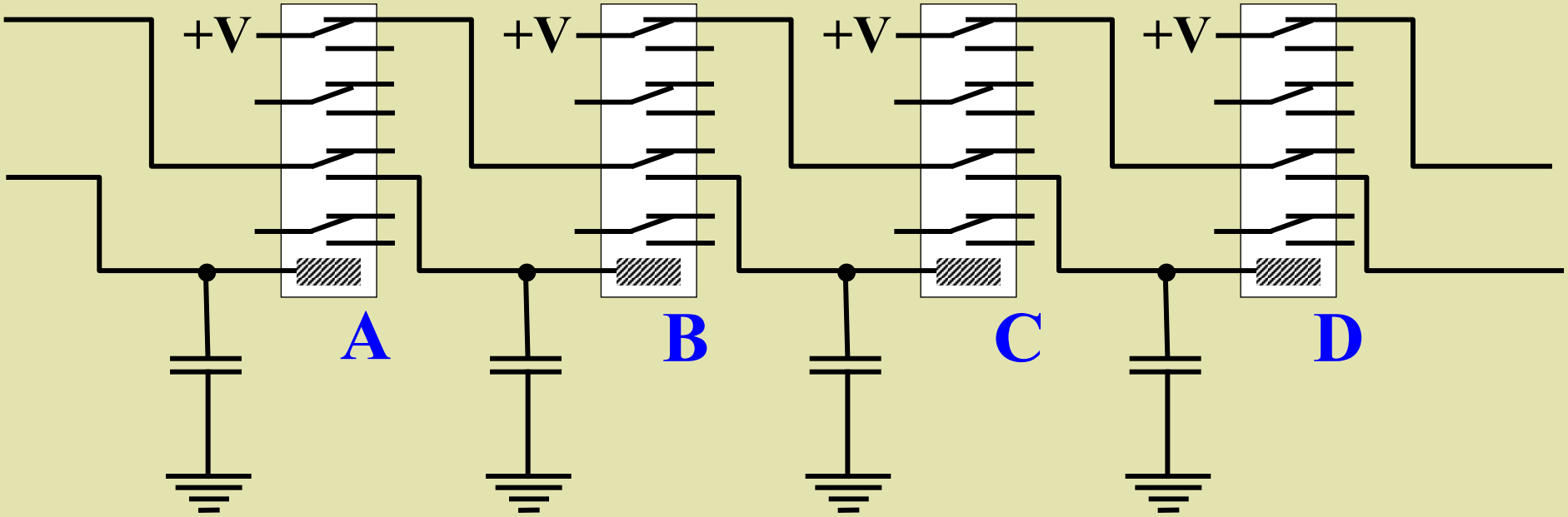
Clock



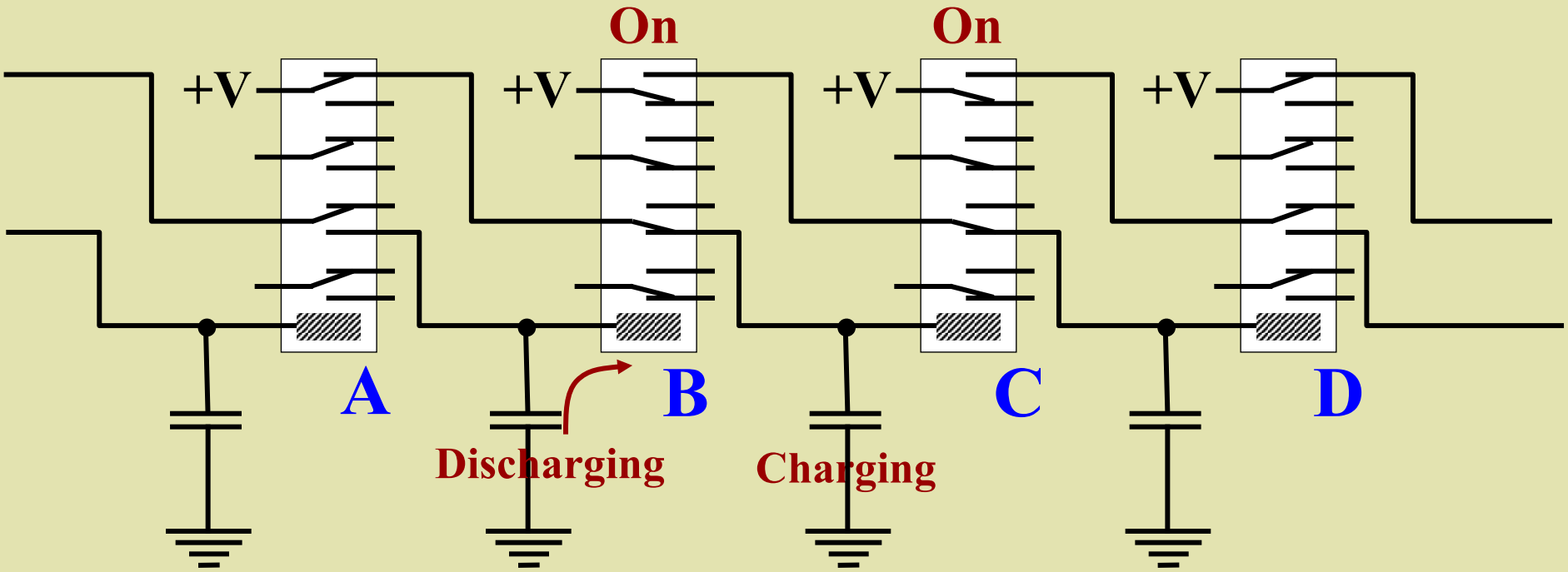
Clock



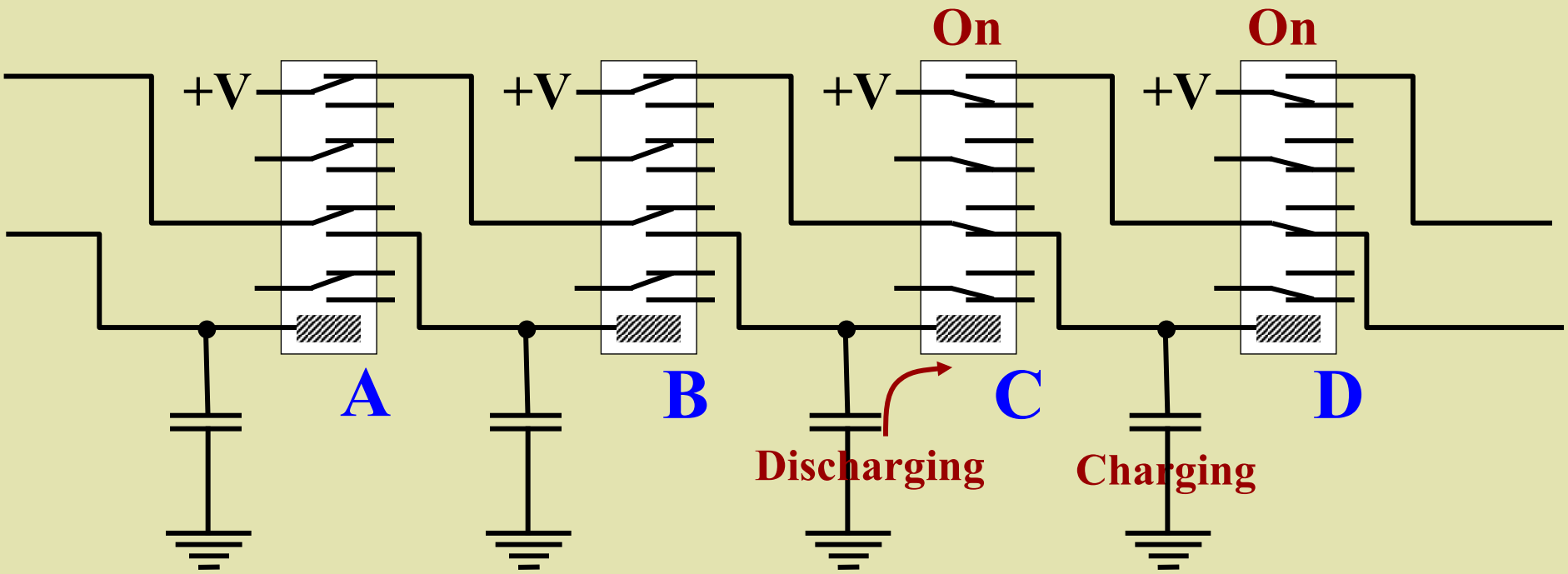
Clock



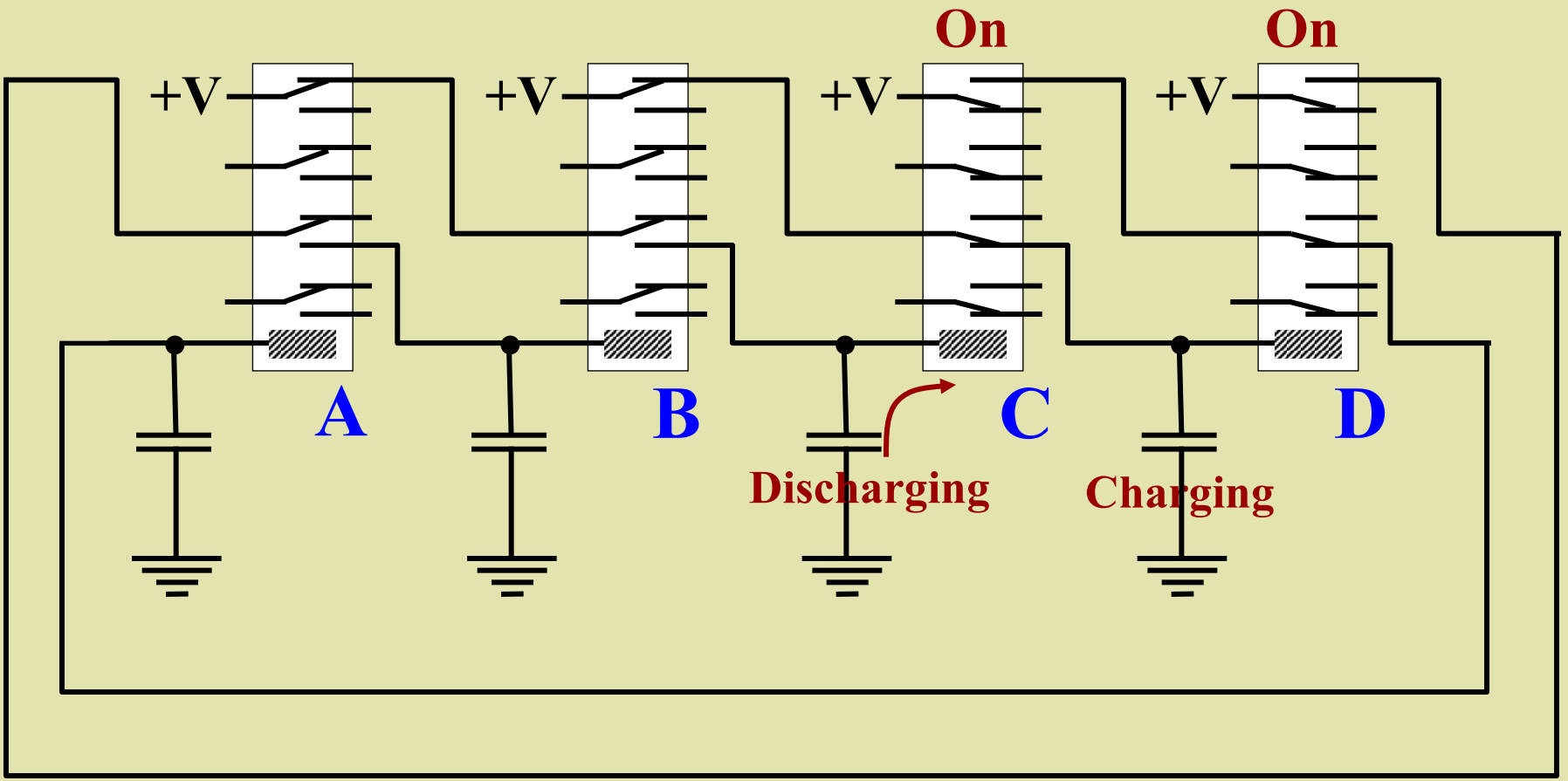
Clock



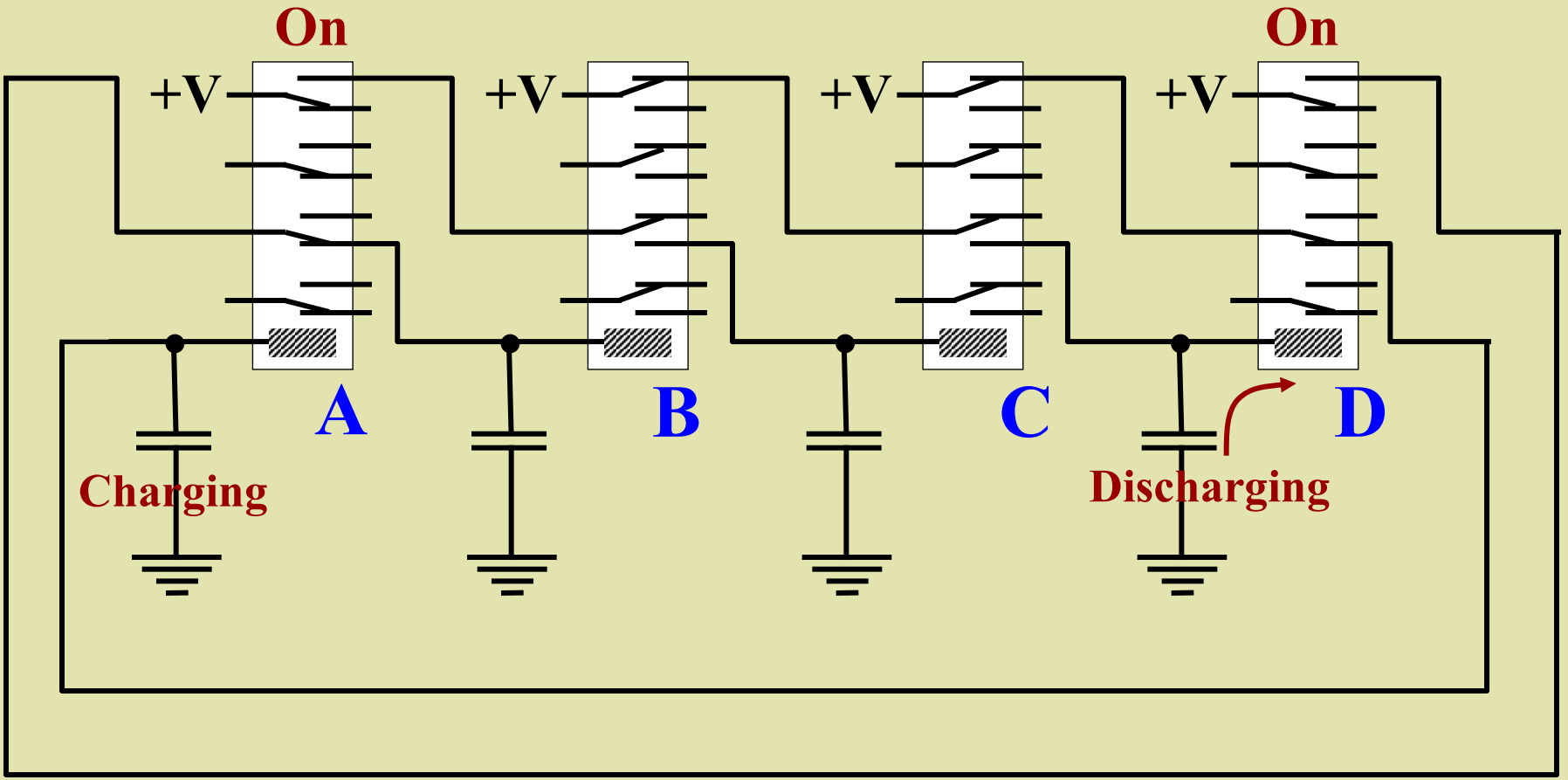
Clock



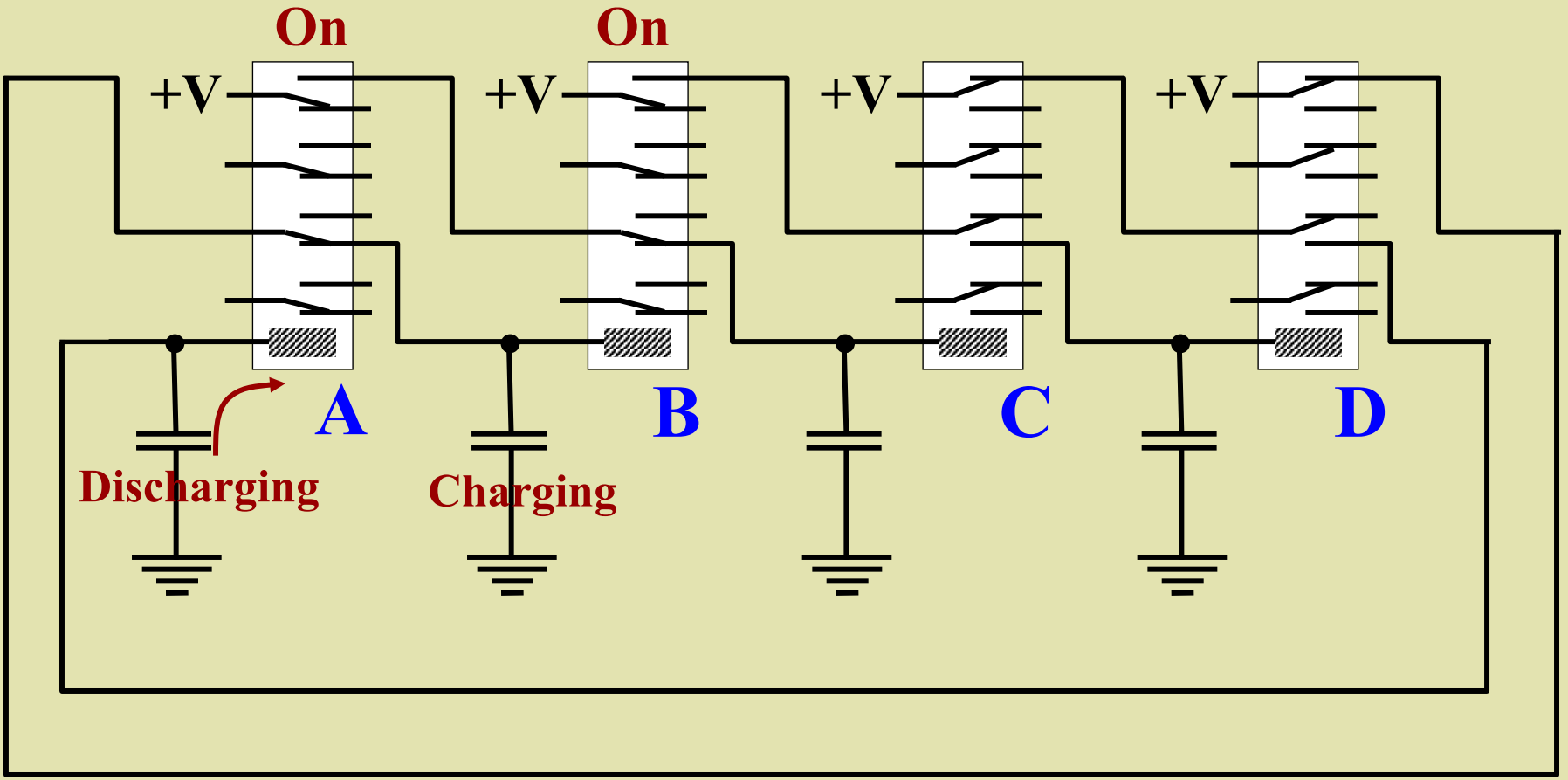
Clock



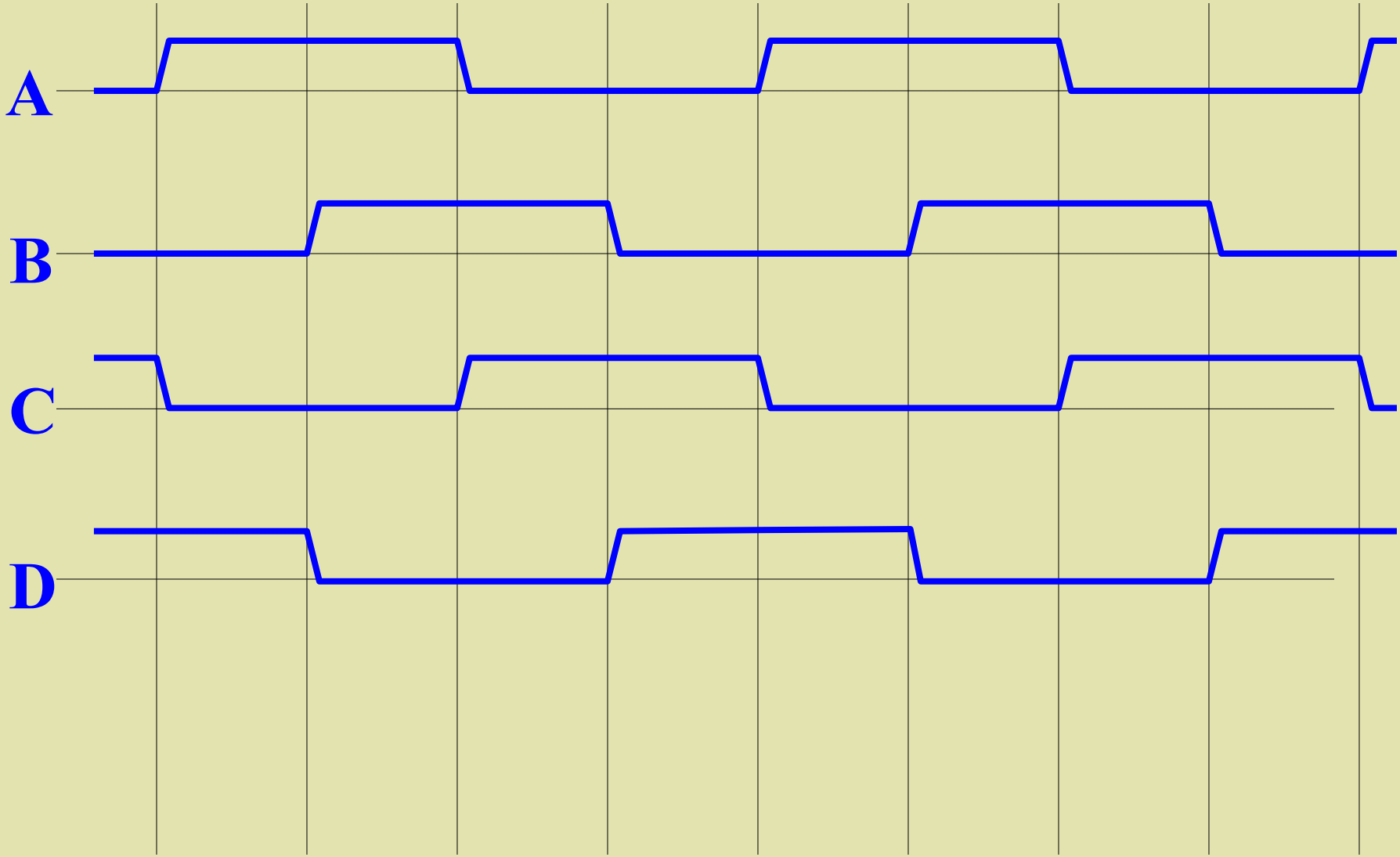
Clock



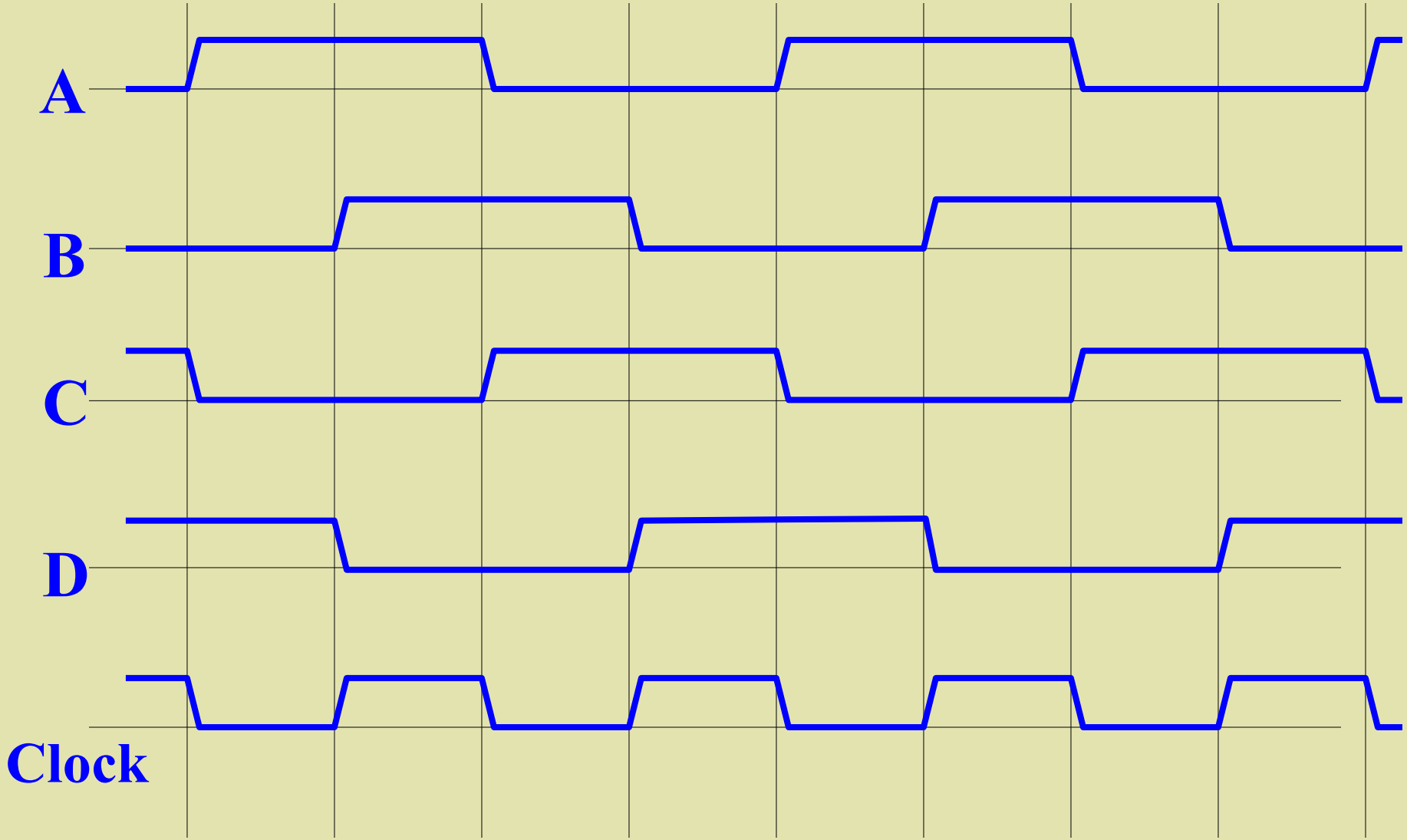
Clock



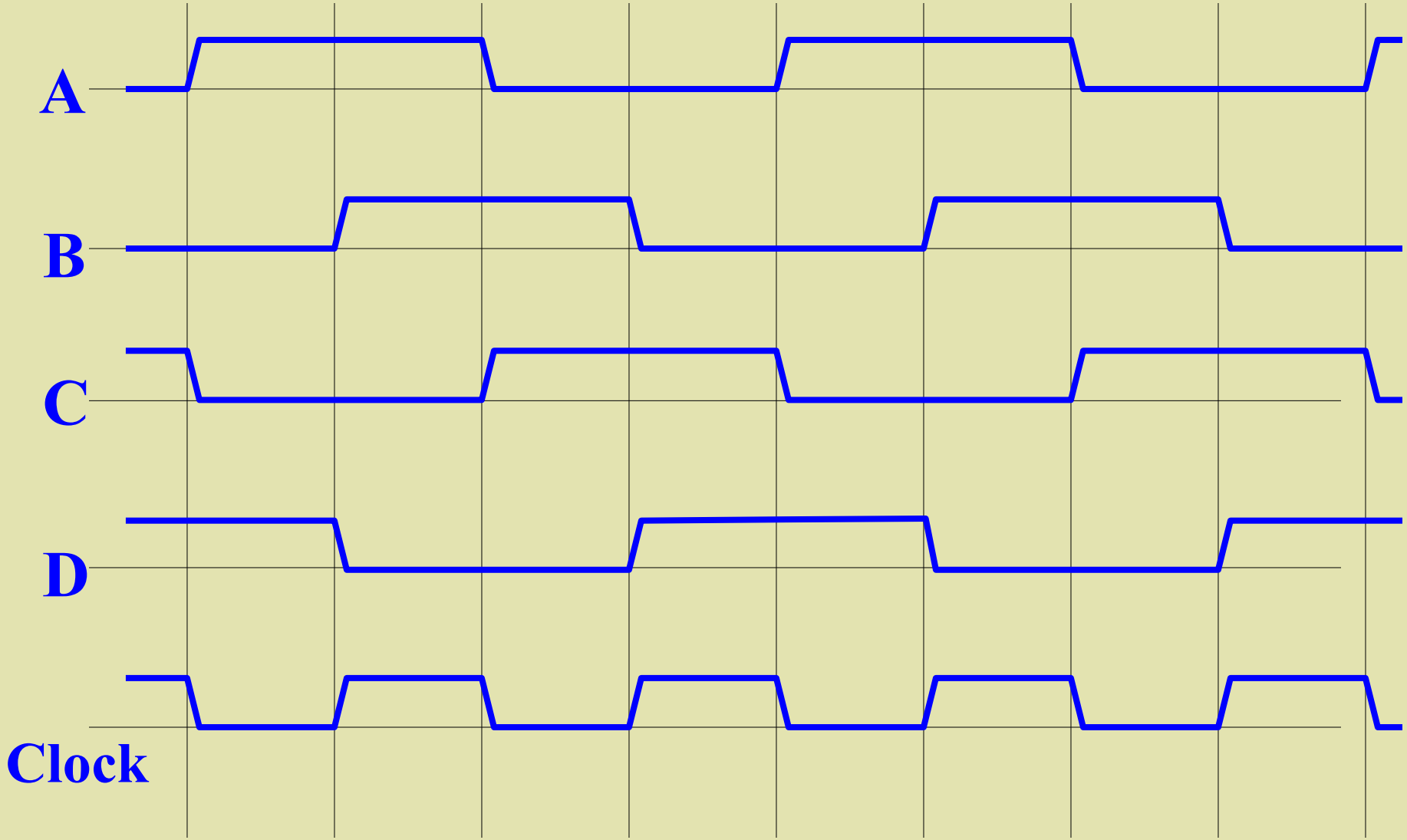
Clock Timing Diagram



Clock Timing Diagram

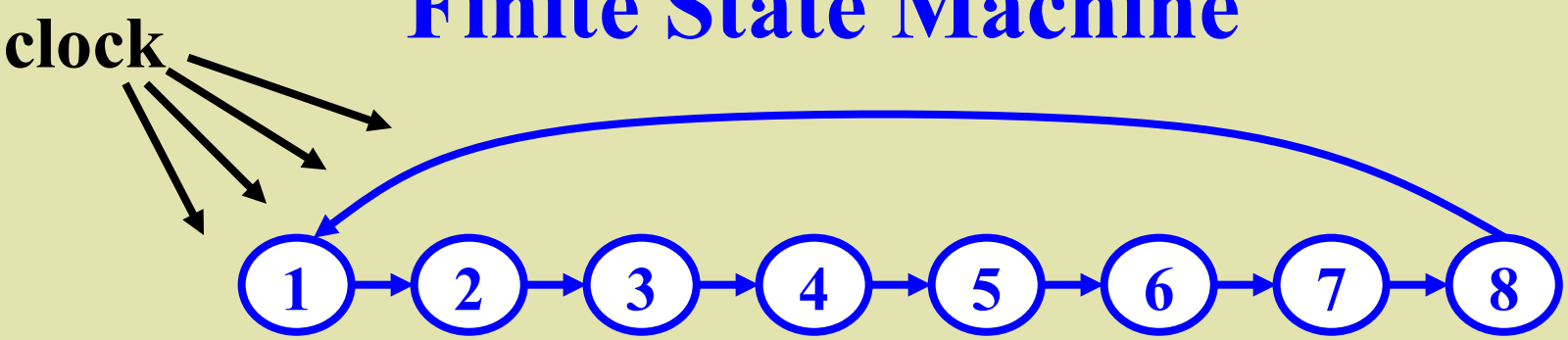


Clock Timing Diagram

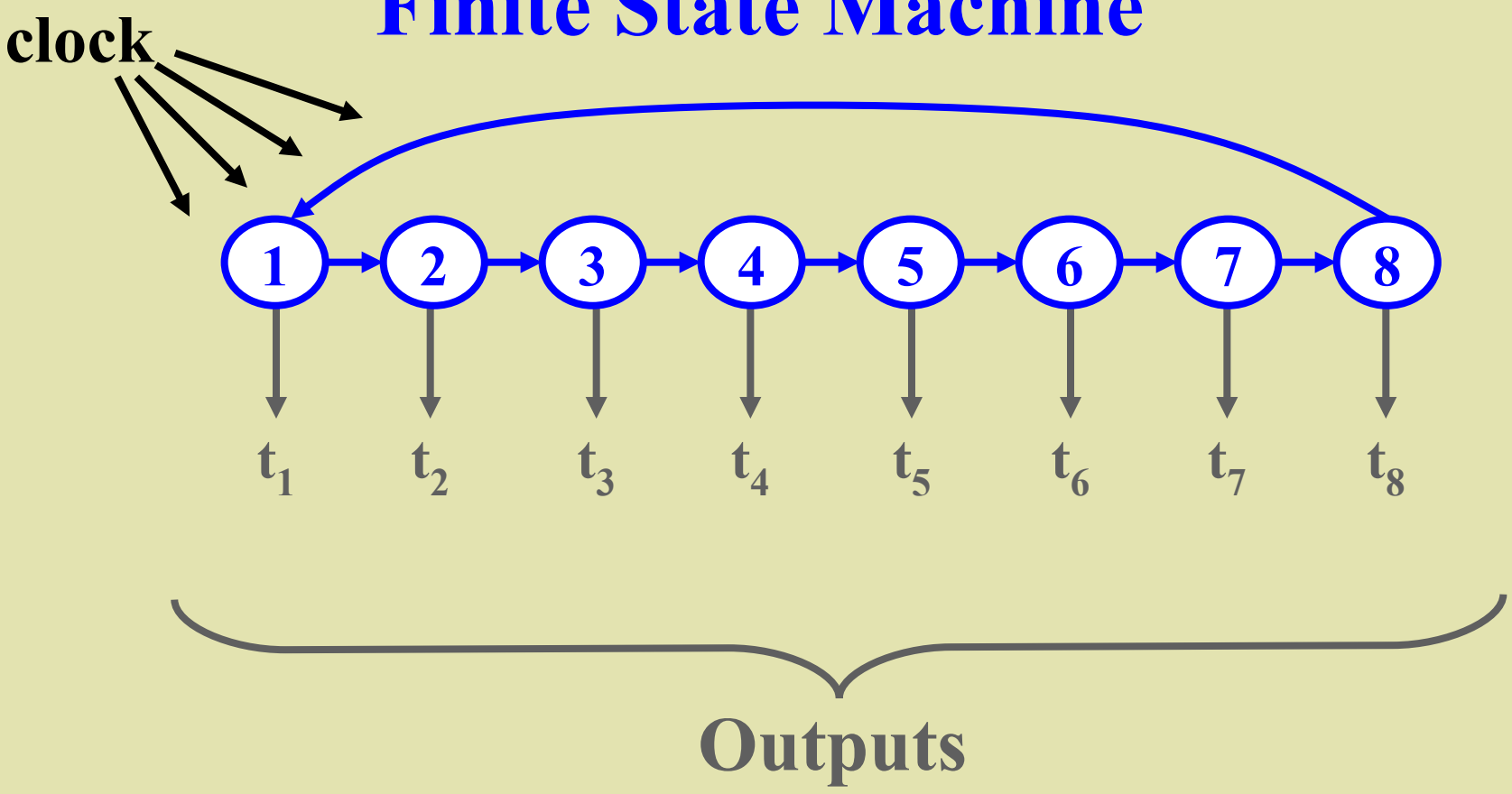


Clock = (A and B) or (C and D)

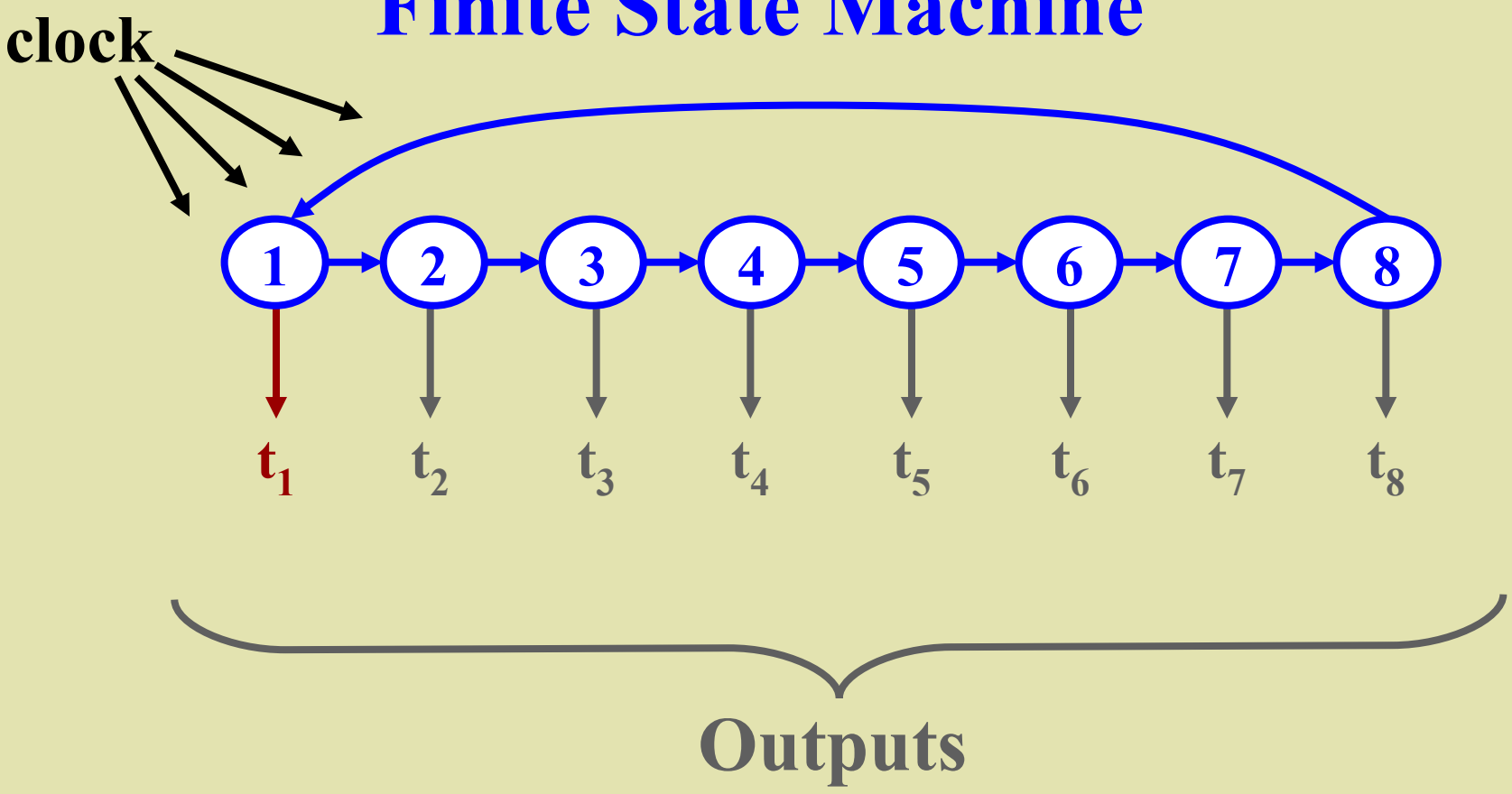
Finite State Machine



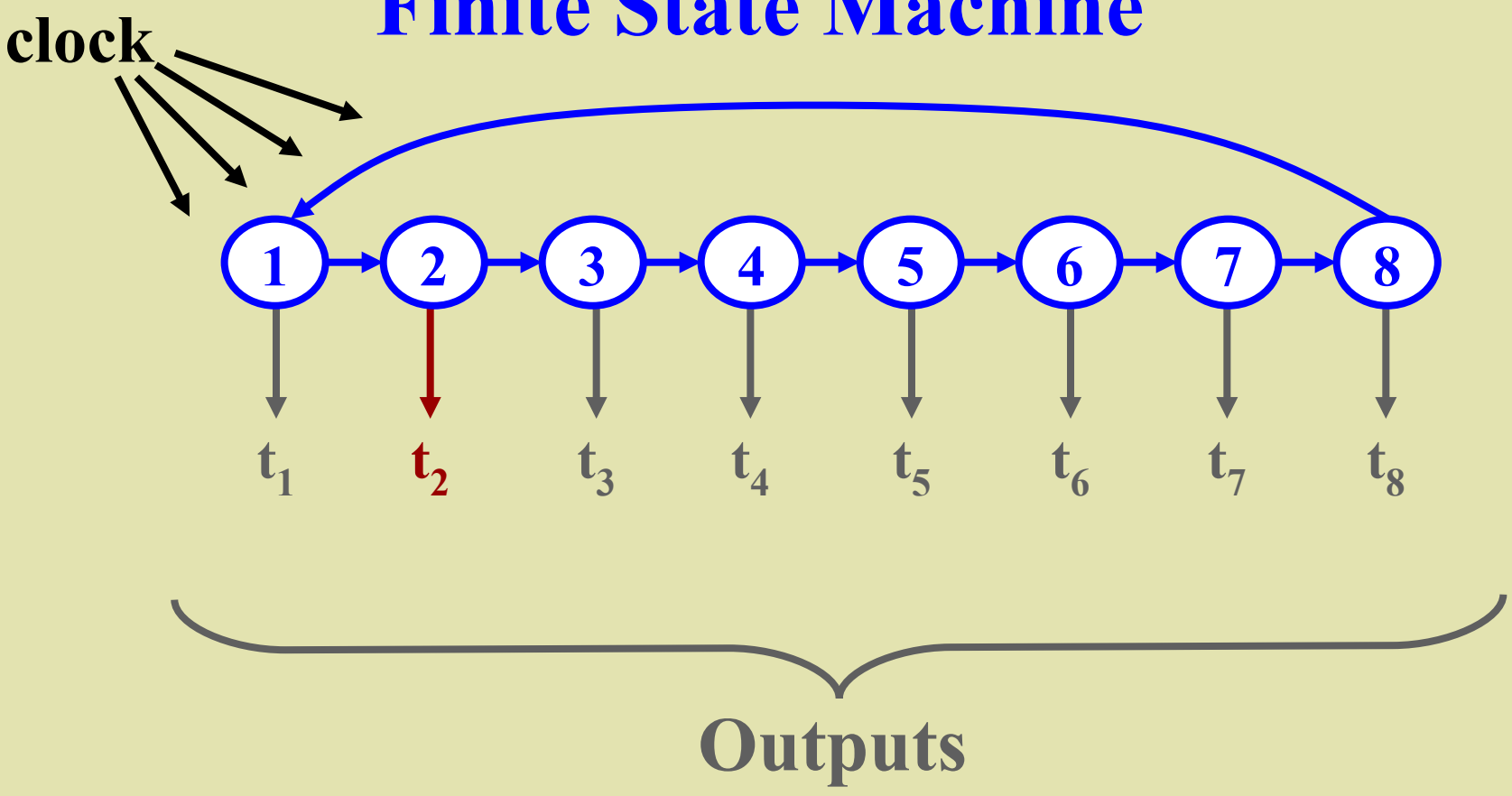
Finite State Machine



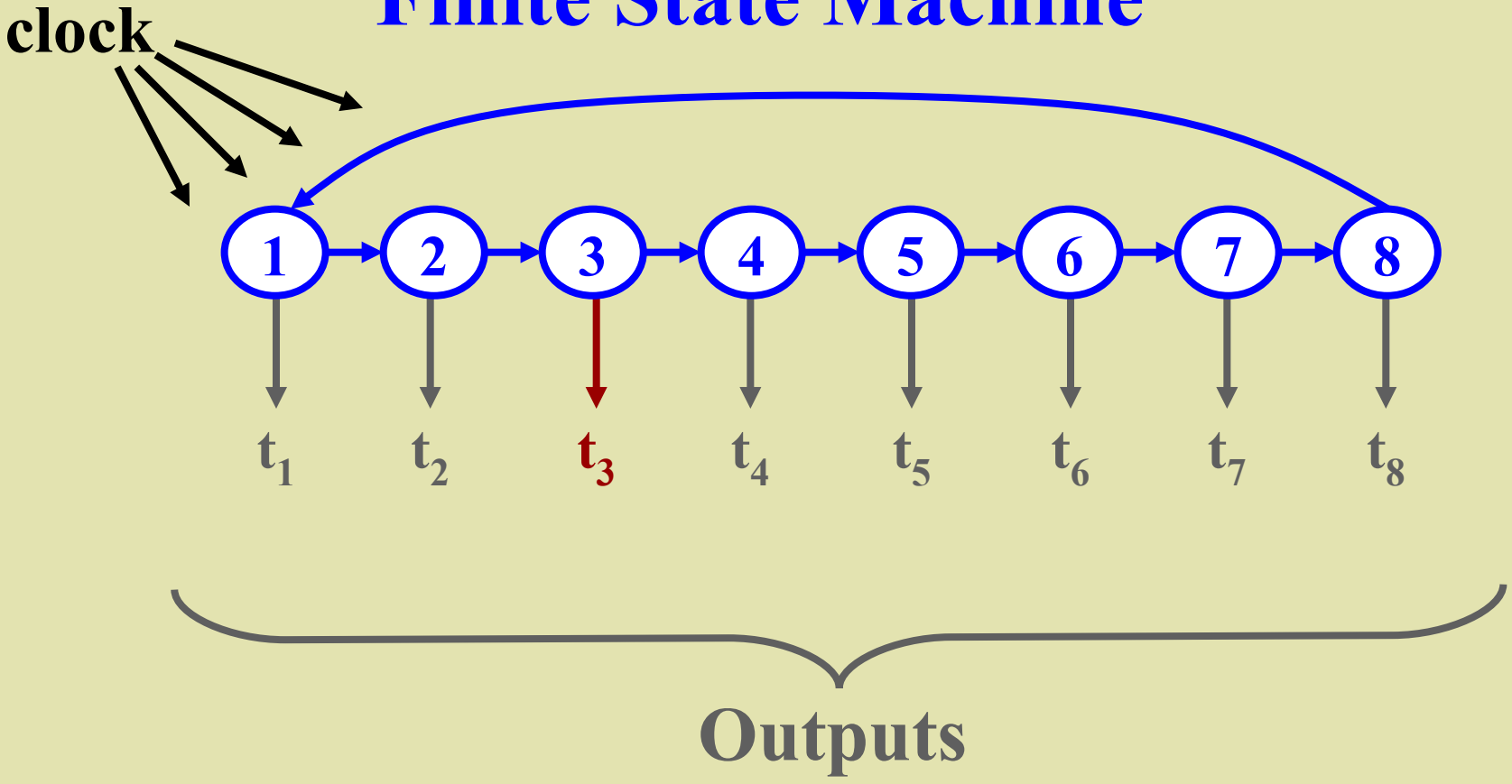
Finite State Machine



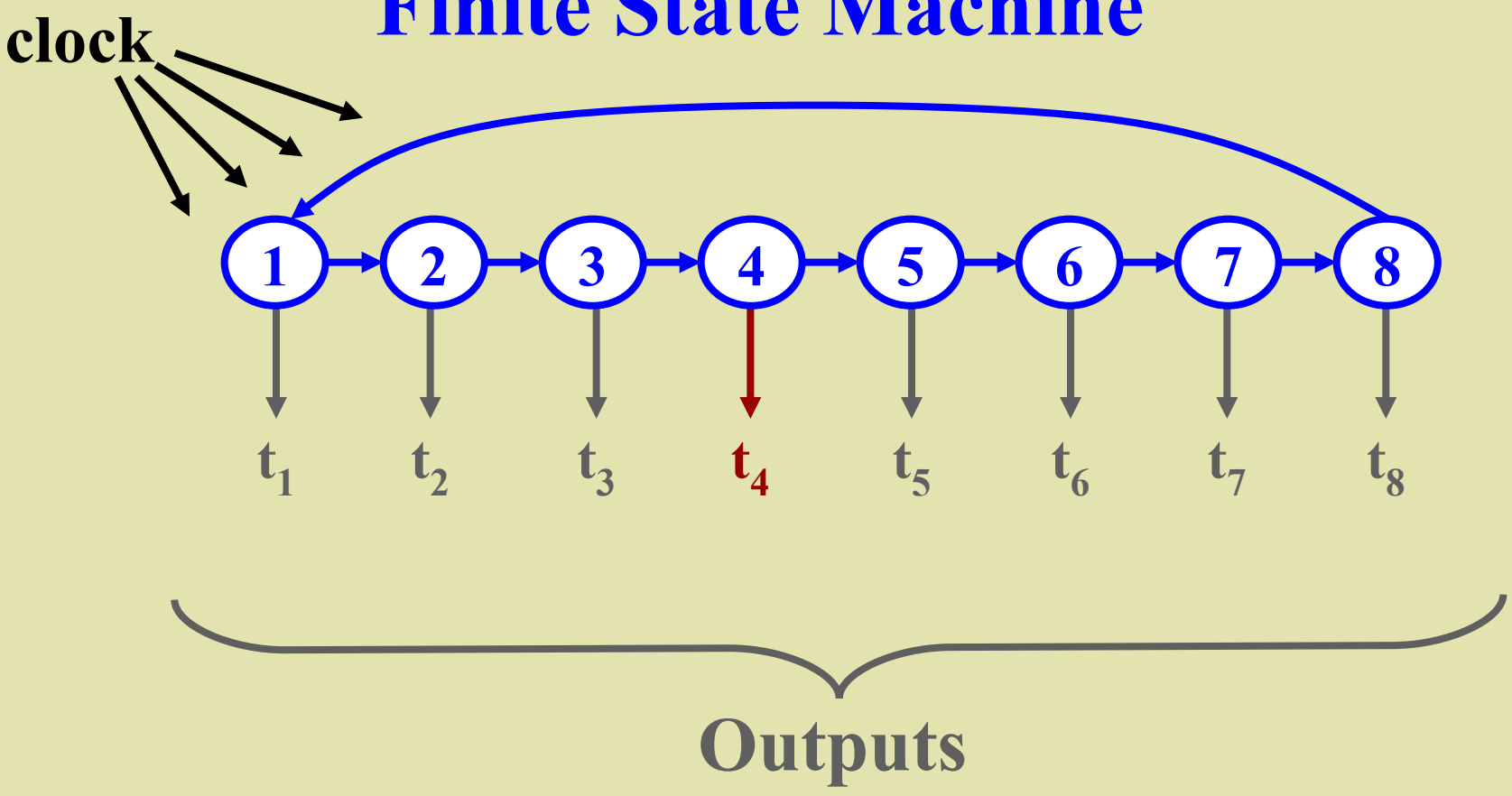
Finite State Machine



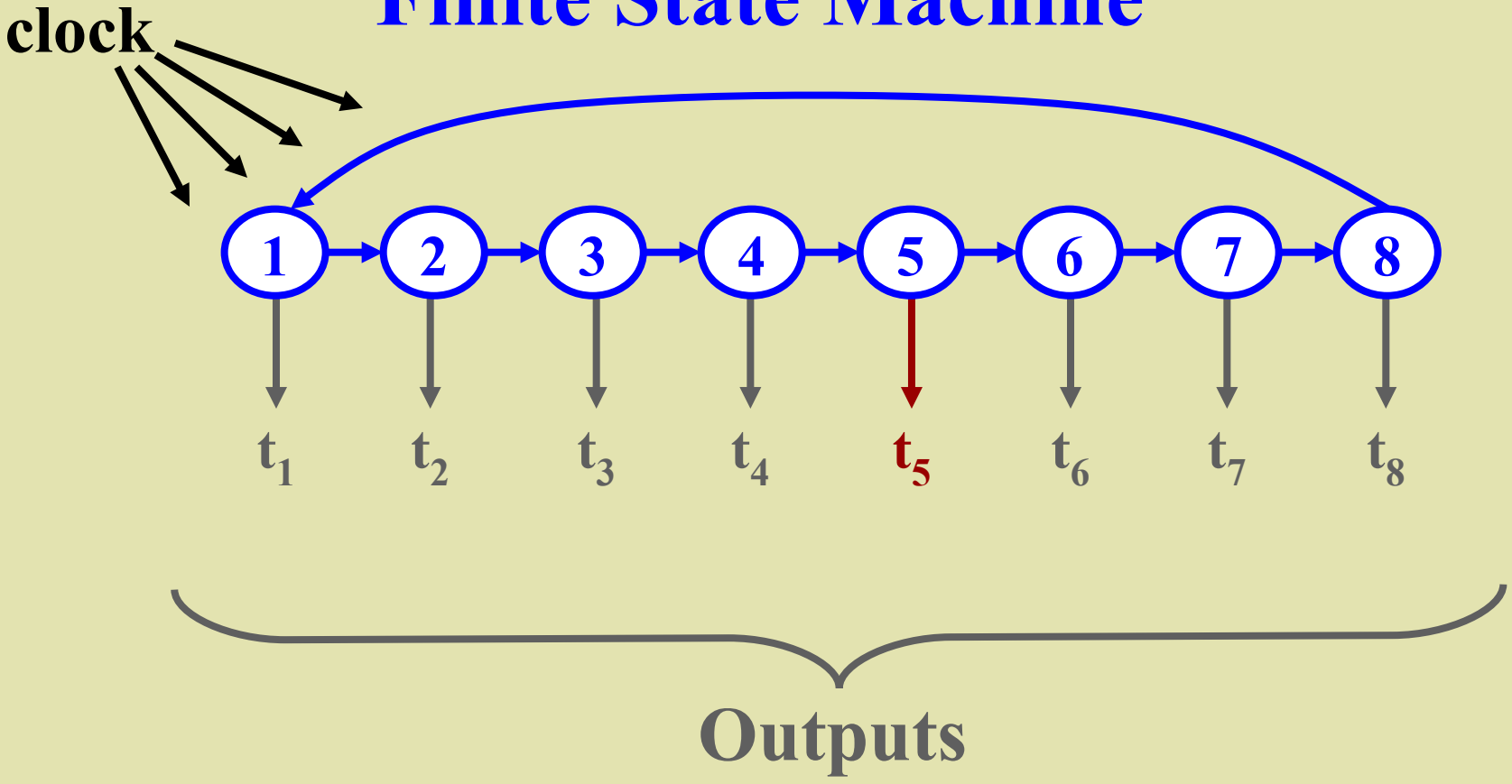
Finite State Machine



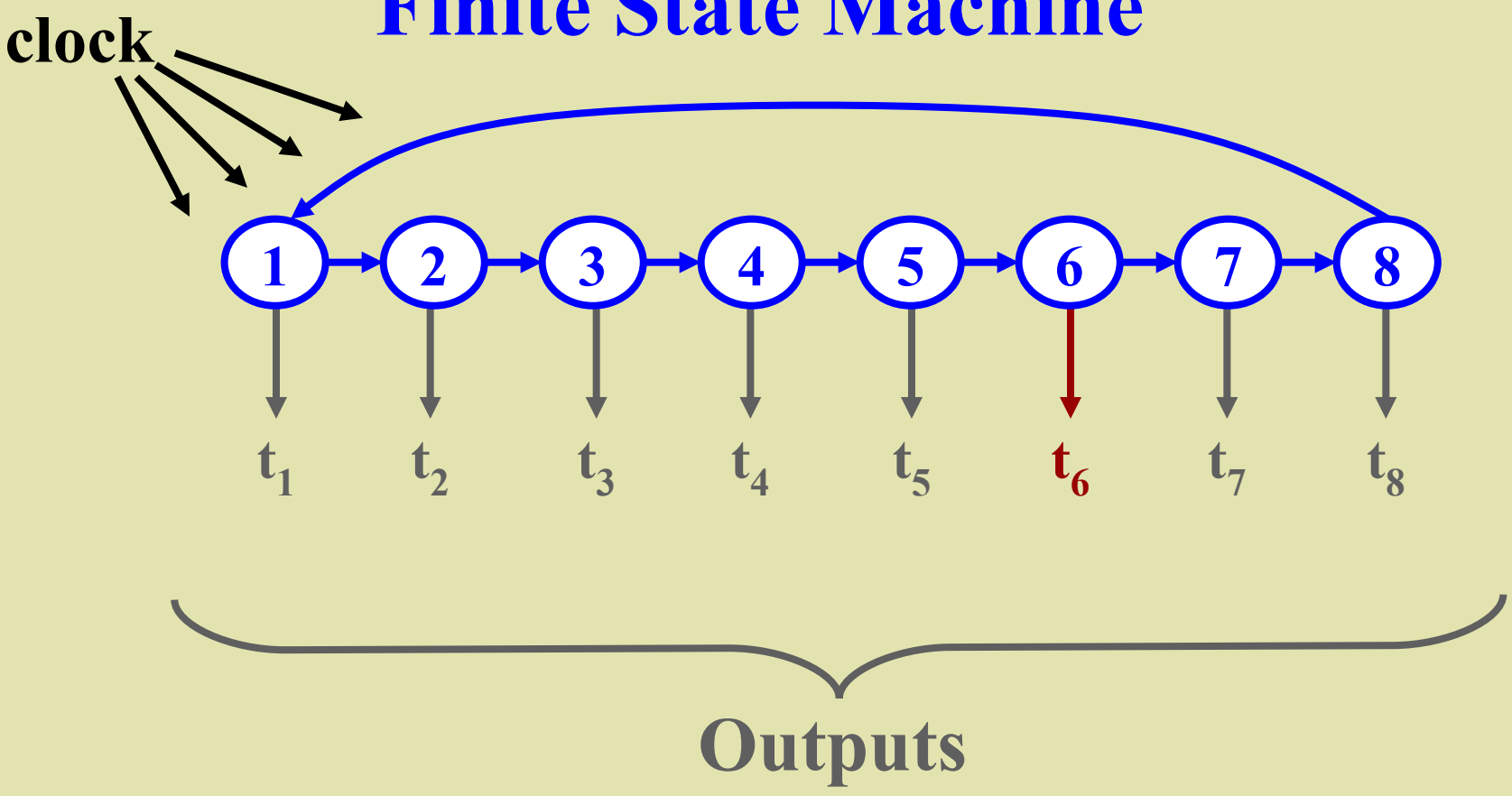
Finite State Machine



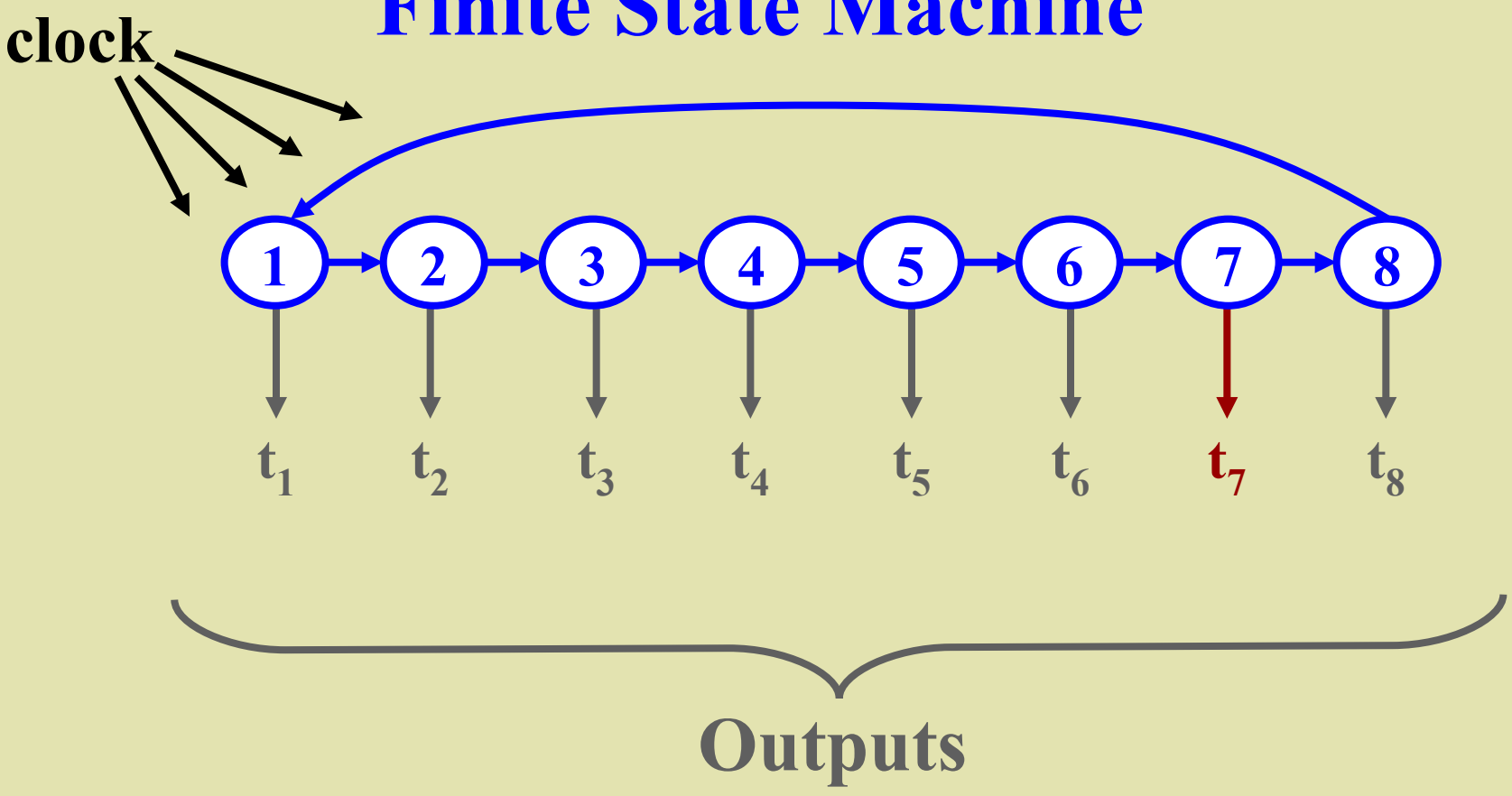
Finite State Machine



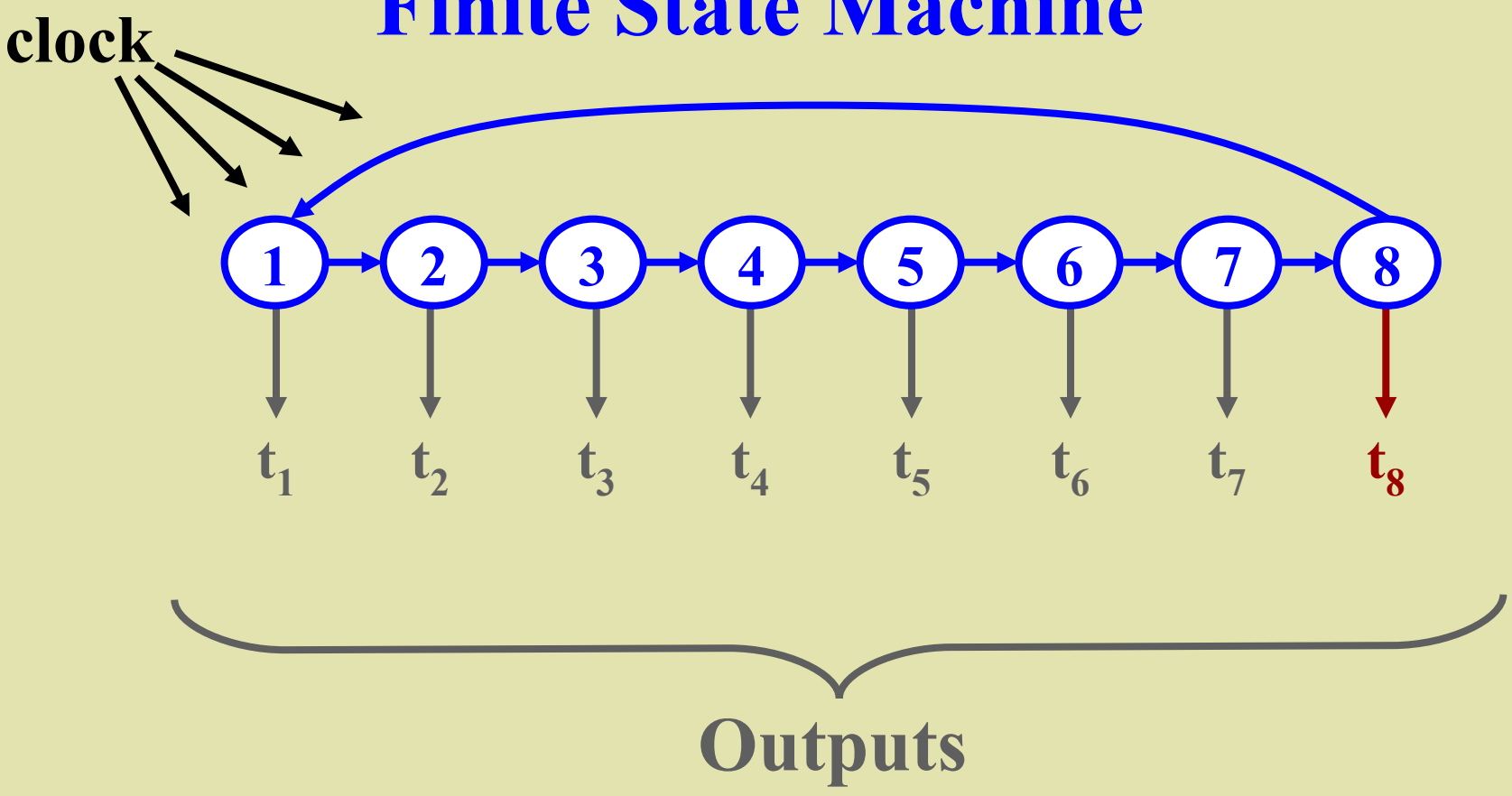
Finite State Machine



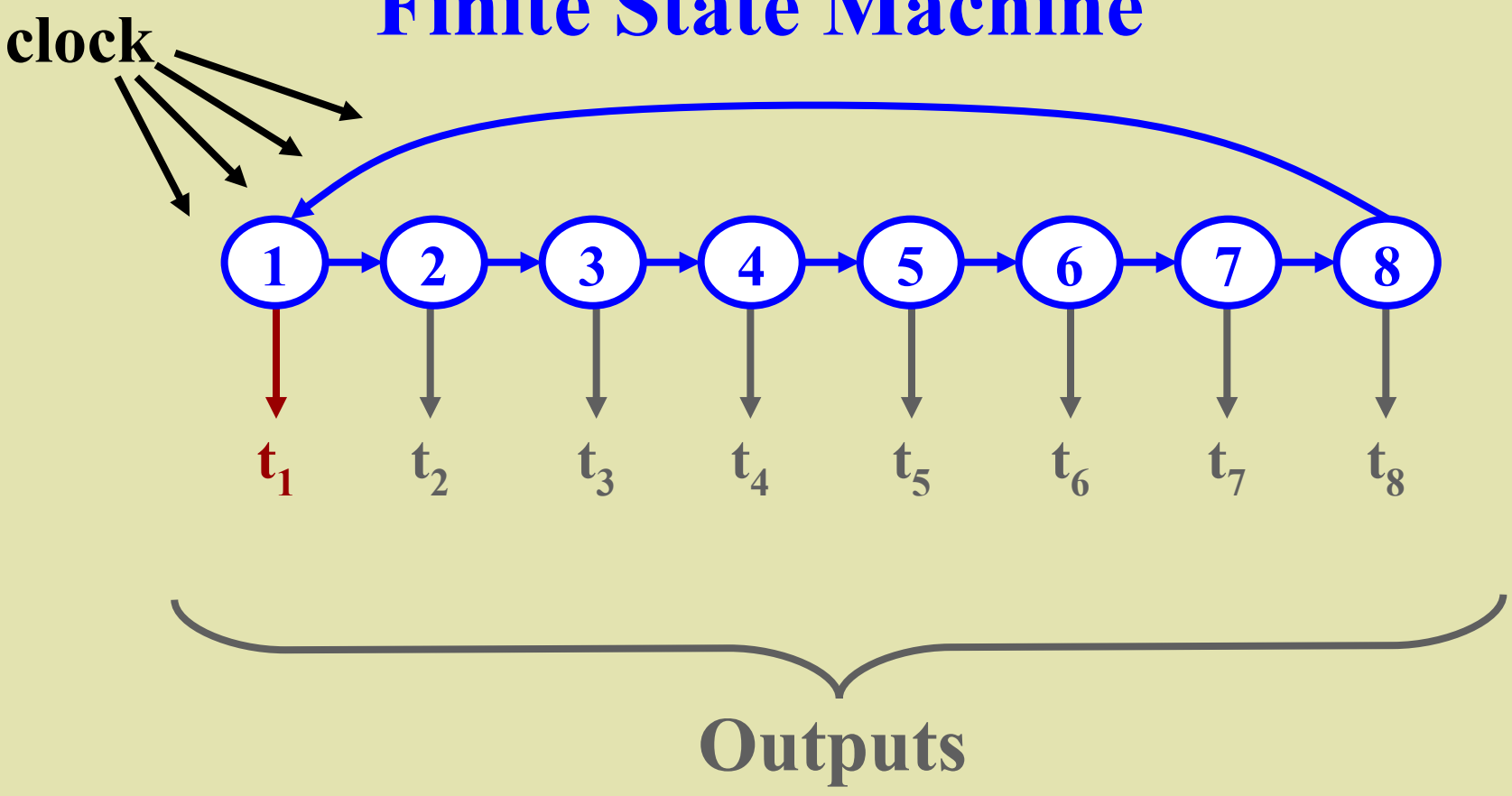
Finite State Machine



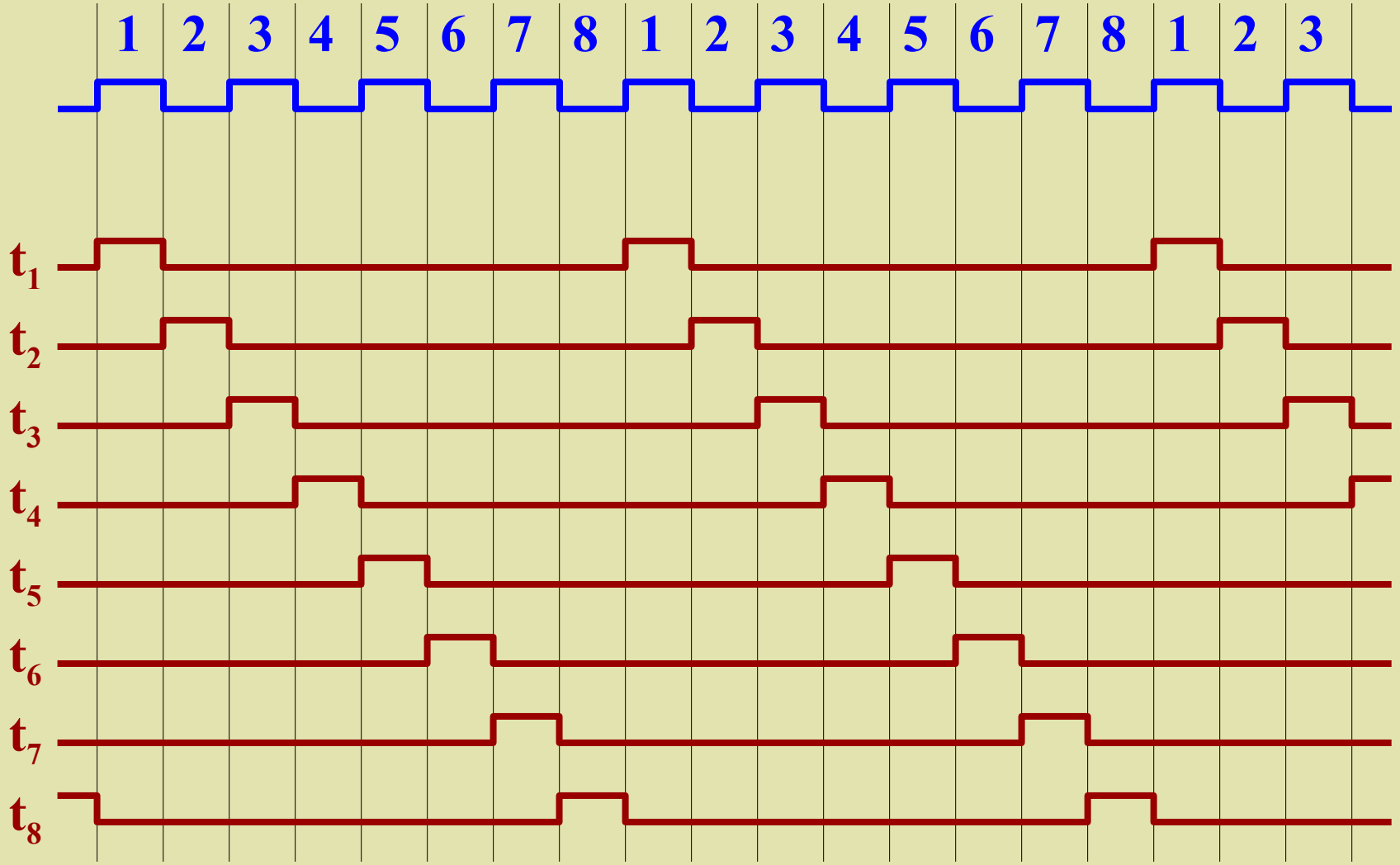
Finite State Machine



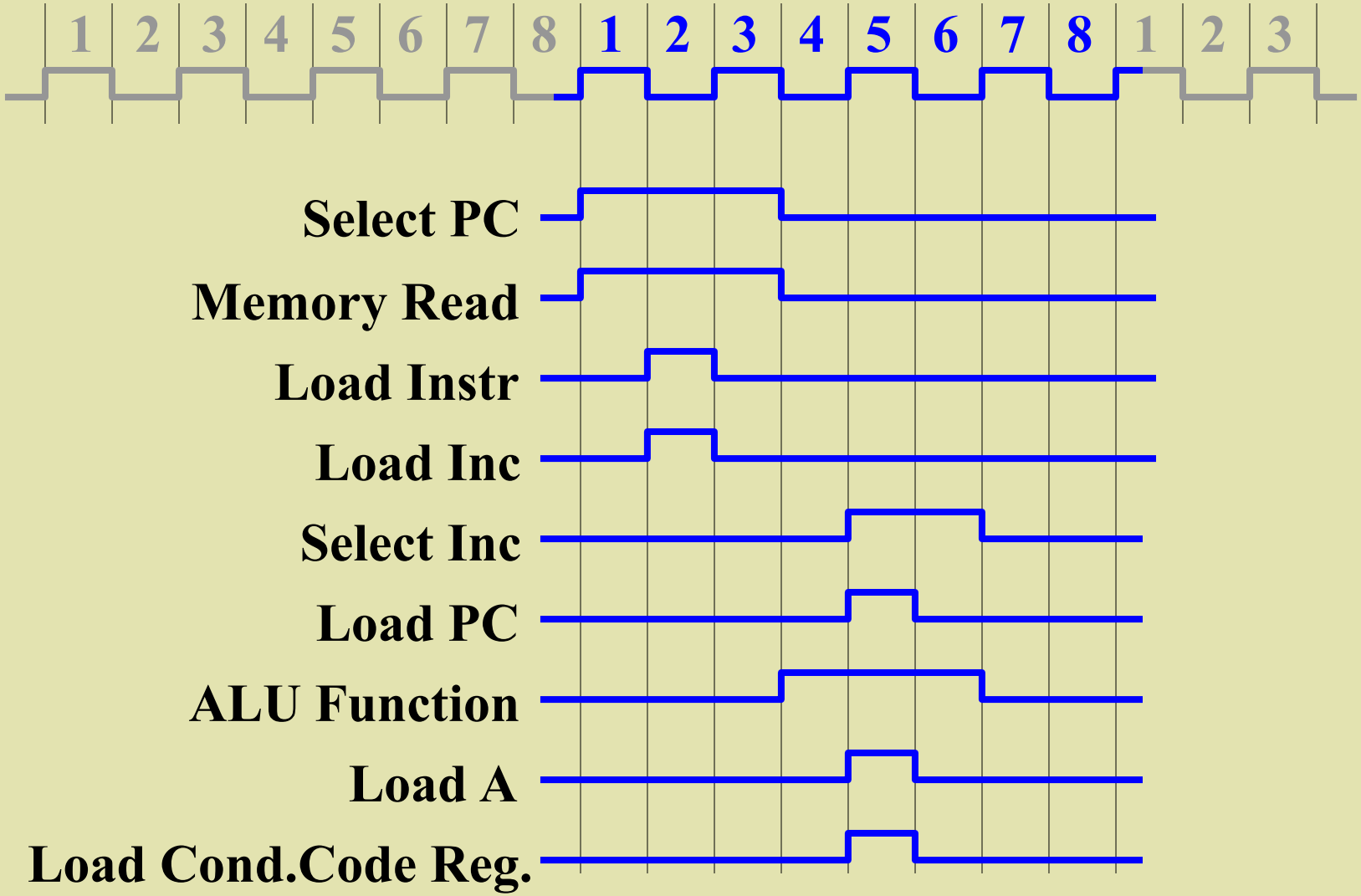
Finite State Machine



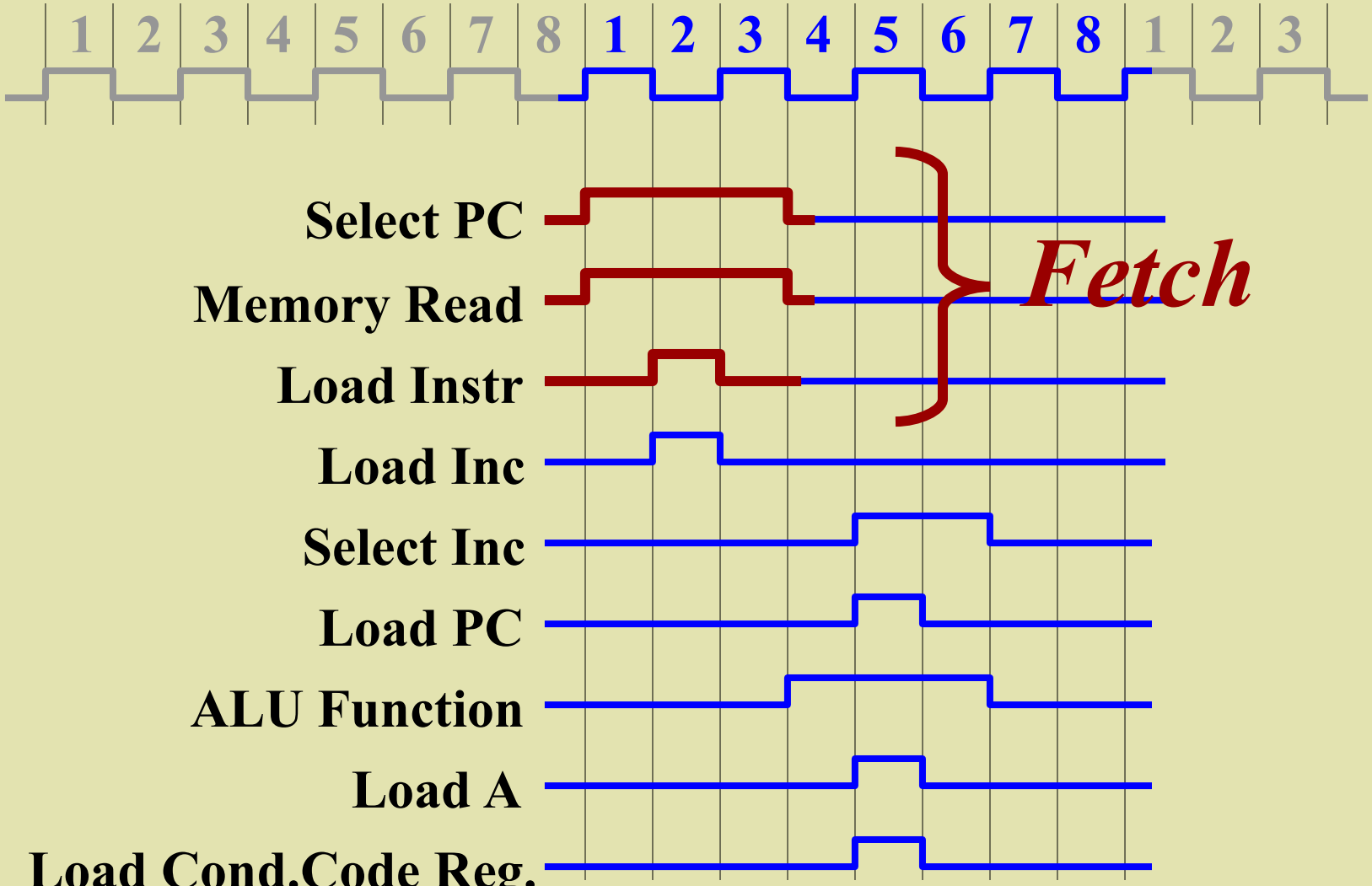
Output from FSA



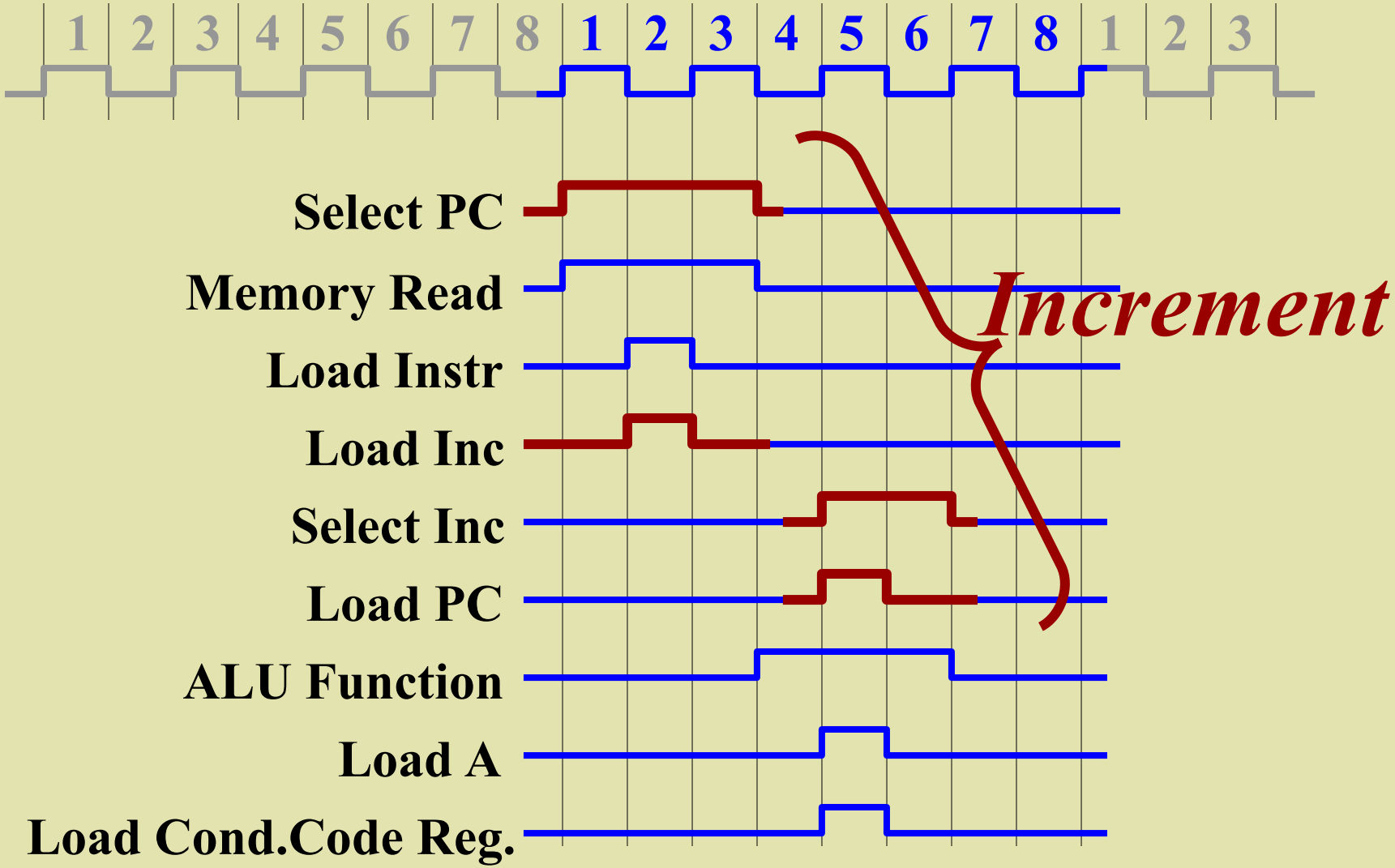
Instruction Timing - ALU Instruction



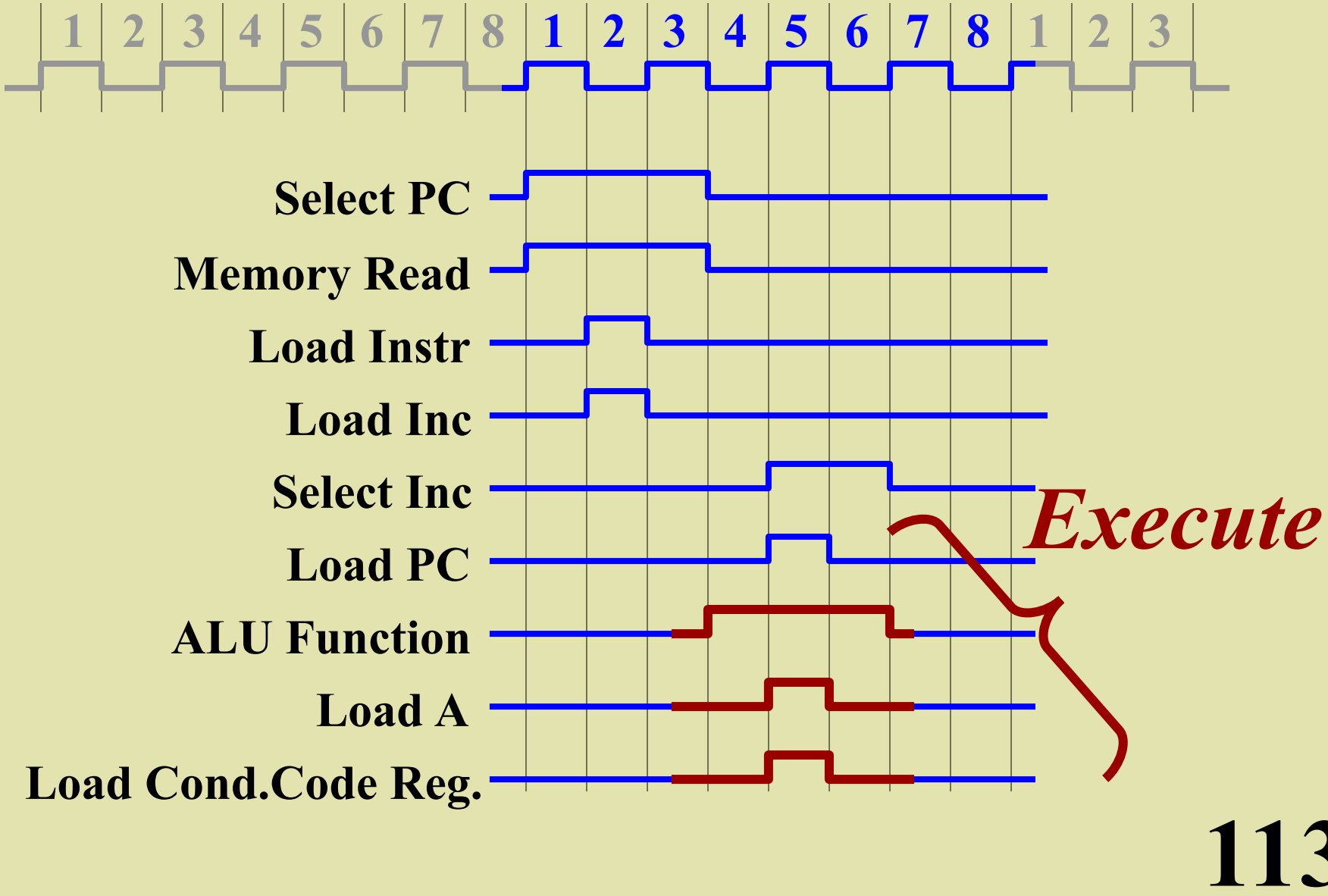
Instruction Timing - ALU Instruction



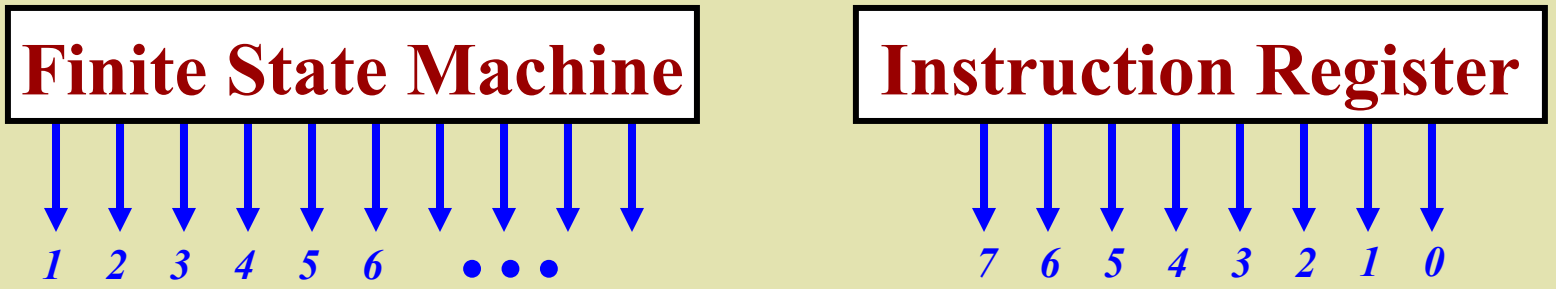
Instruction Timing - ALU Instruction



Instruction Timing - ALU Instruction



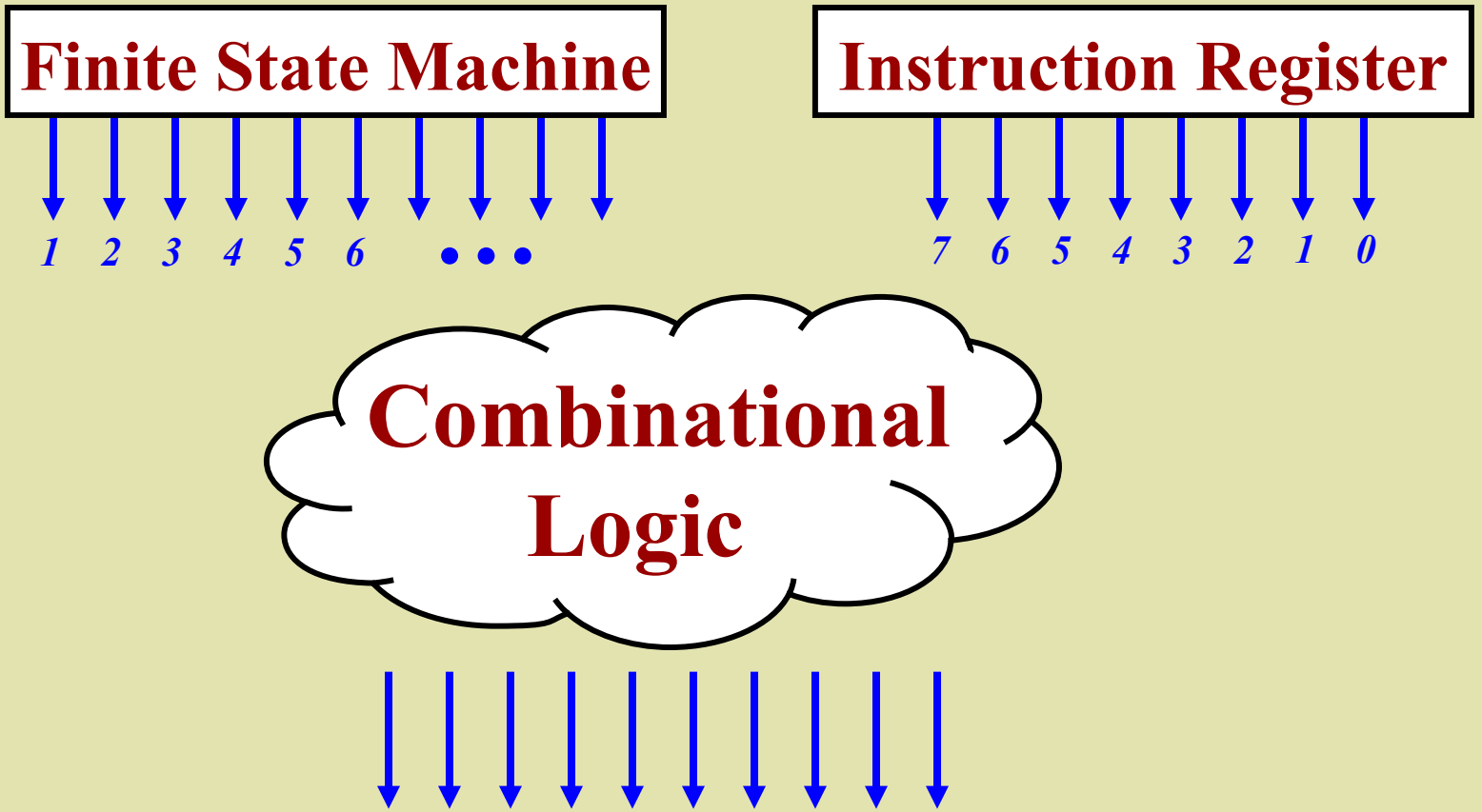
Instruction Decoding



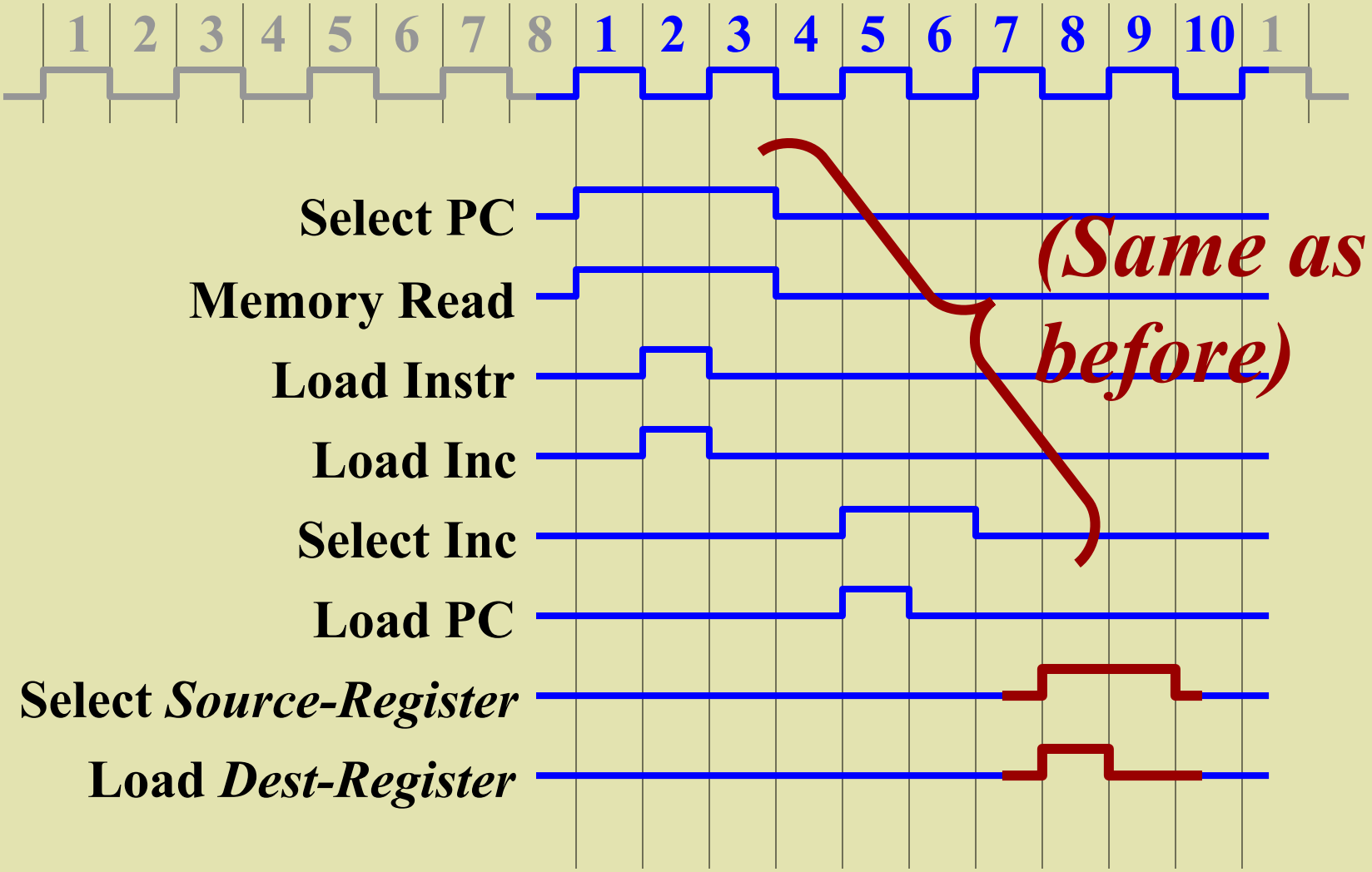
Control Signals

(Load, Select, Mem-Read, etc.)

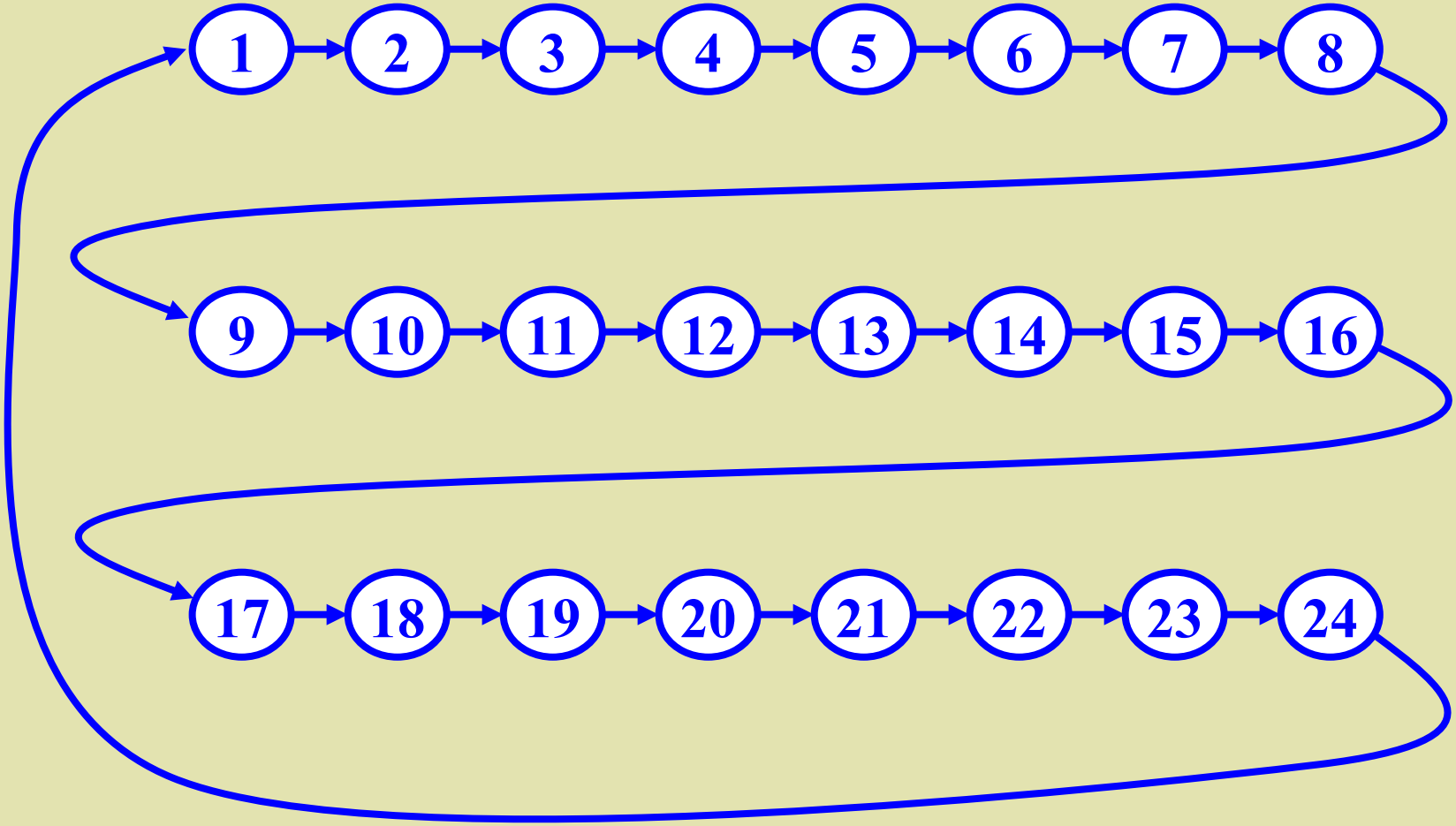
Instruction Decoding



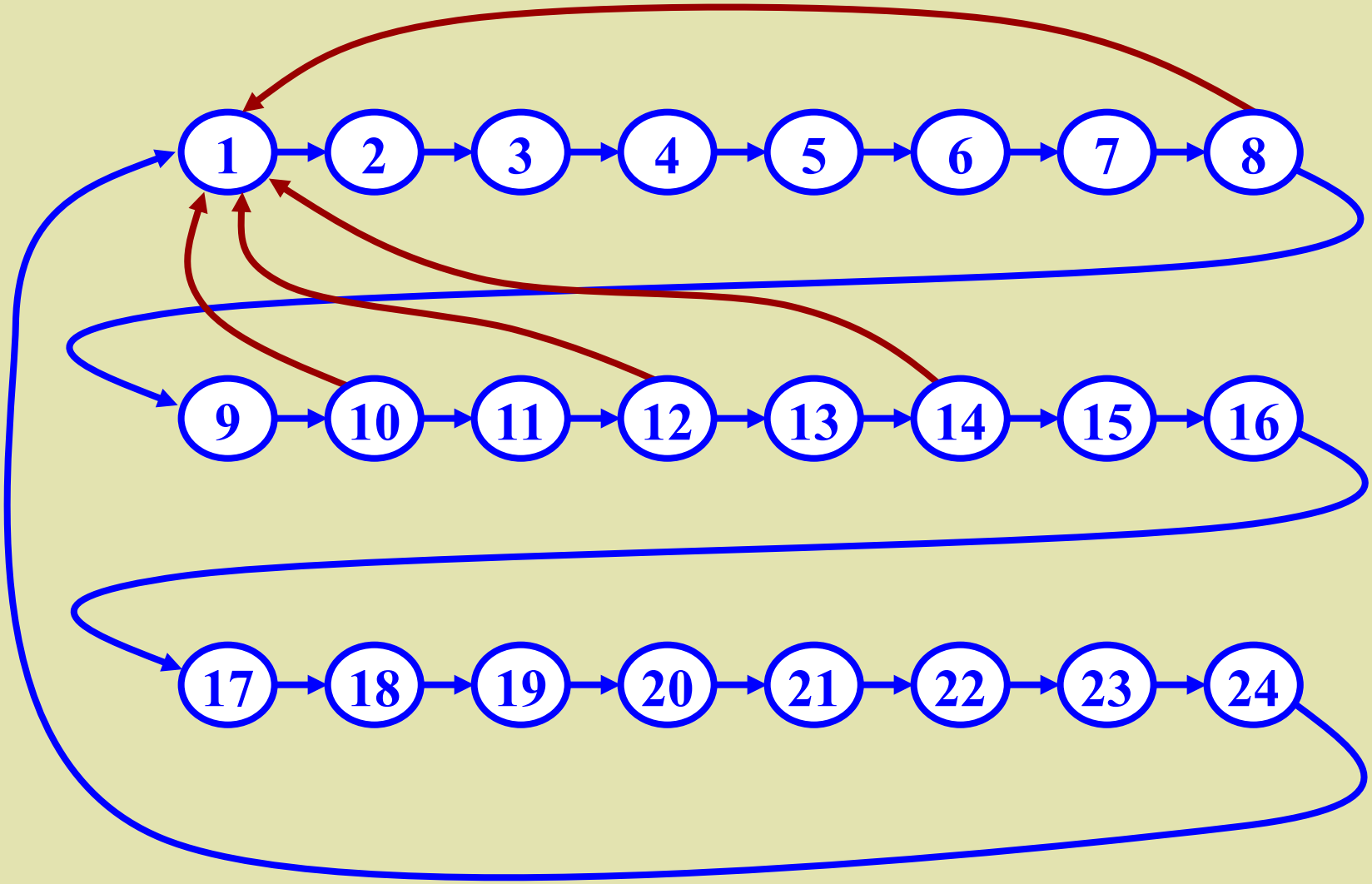
Instruction Timing - 16-bit Move



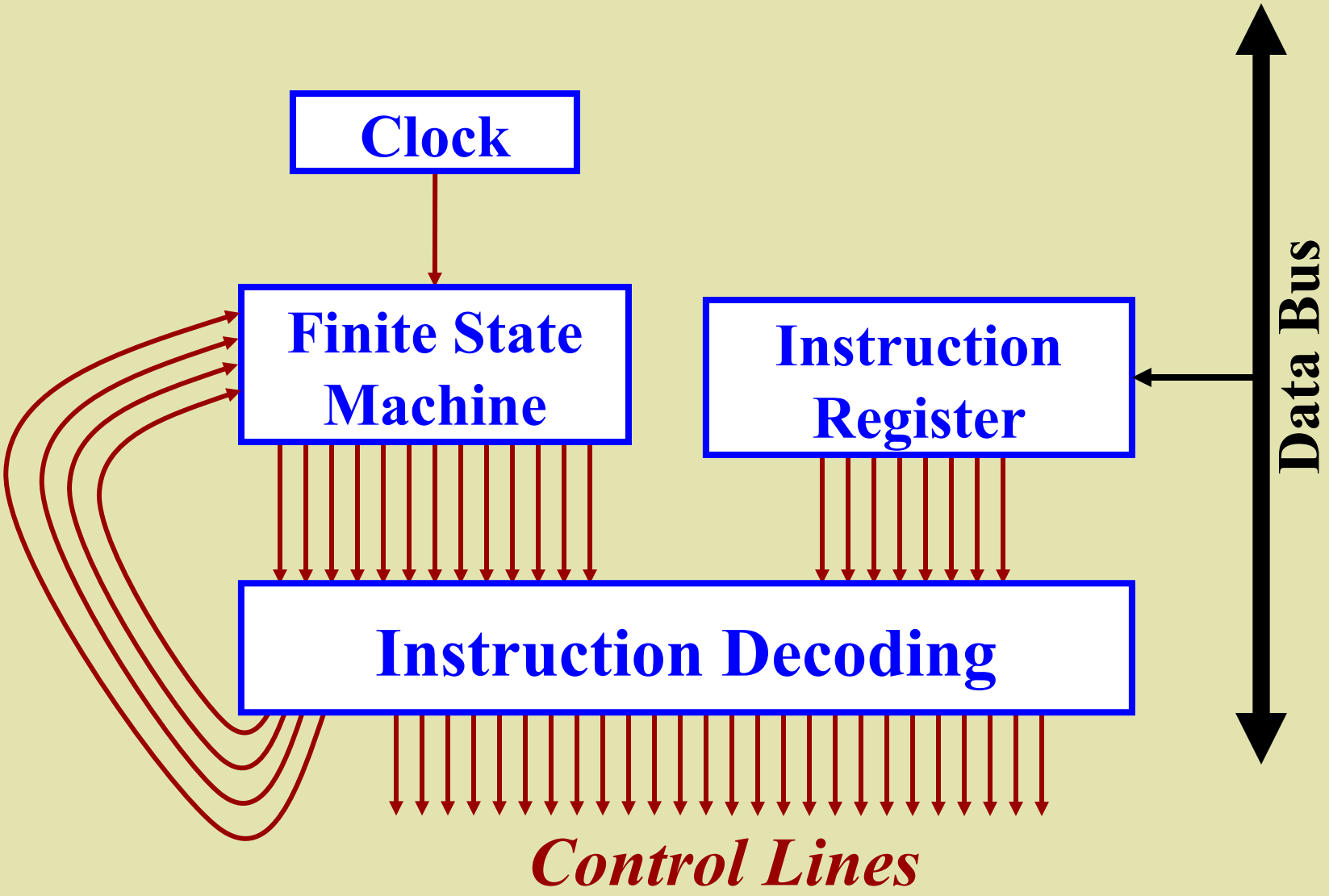
Finite State Machine



Finite State Machine



Instruction Decoding and Control

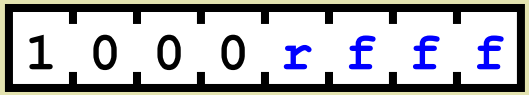


The Instruction Set (1)

Move

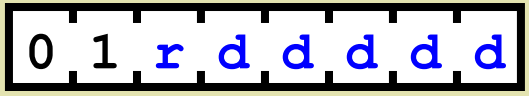
`ddd` = destination register
`sss` = source register
(A, B, C, D, M_1, M_2, X or Y)

ALU



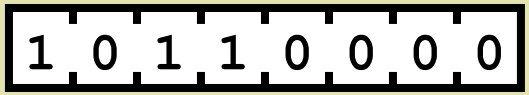
`r` = destination register (A or D)
`fff` = function code
(add, inc, and, or, xor, not, shl)

Load Immediate



`r` = destination register (A or B)
`ddddd` = value (-16..15)

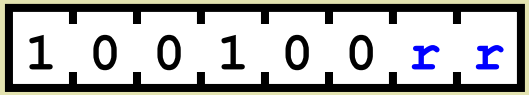
16-bit Increment



XY ← XY + 1

The Instruction Set (2)

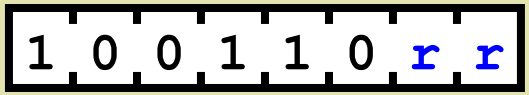
Load



`rr` = destination register (A, B, C, D)

reg ← [M]

Store



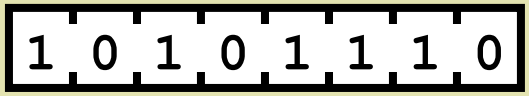
`rr` = source register (A, B, C, D)

[M] ← reg

Load 16-bit Immediate

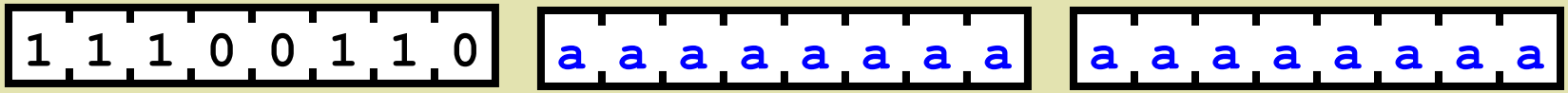
Load the immediate value into M (i.e., M_1 and M_2)

Halt



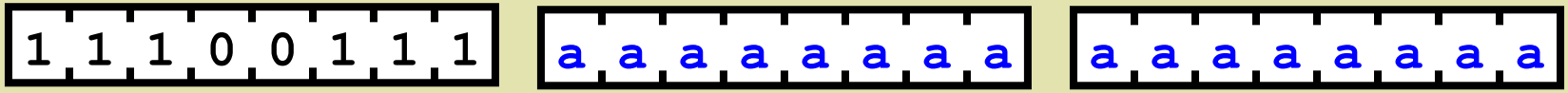
The Instruction Set (3)

Goto



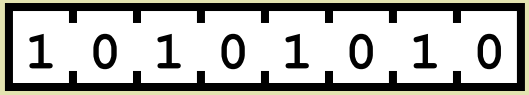
Branch to the given address

Call



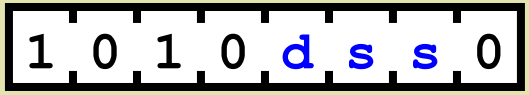
Branch to the given address
Save return location in XY register

Return / Branch Indirect



PC ← XY

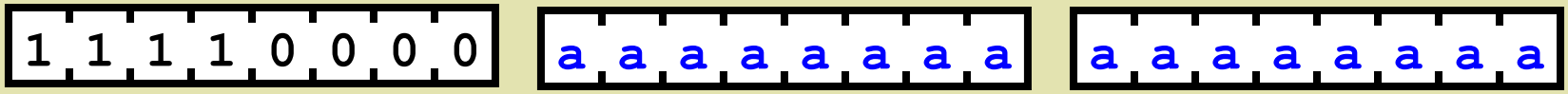
16-Bit Move



`d` = destination register (PC or XY)
`ss` = source register (M, XY or J)

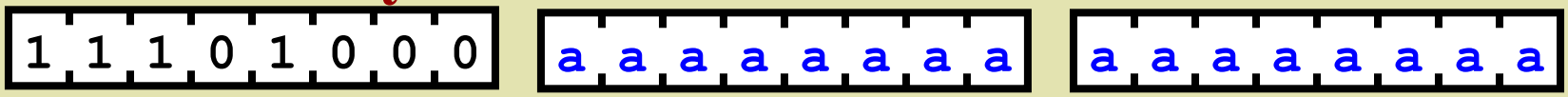
The Instruction Set (4)

Branch If Negative



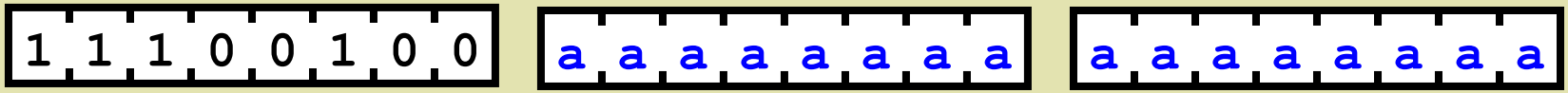
Branch to the given address if S = 1

Branch If Carry



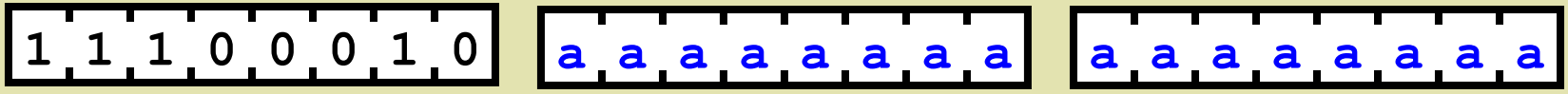
Branch to the given address if Cy = 1

Branch If Zero



Branch to the given address if Z = 1

Branch If Not Zero



Branch to the given address if Z = 0

An Example Program

address	instr	assembly	comment
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