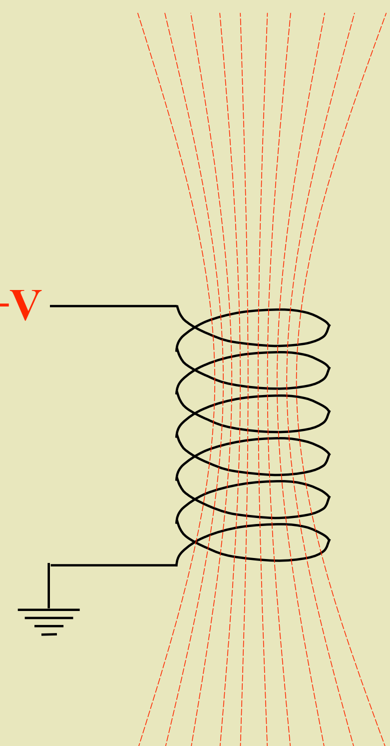
Slides to Accompany a Video Tutorial on

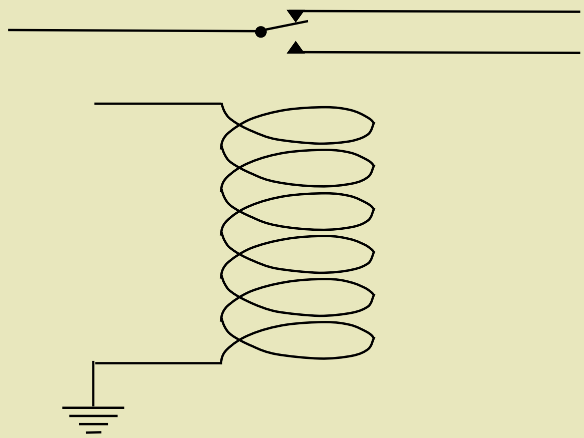
Harry Porter's Relay Computer

Harry Porter, Ph.D.
Portland State University

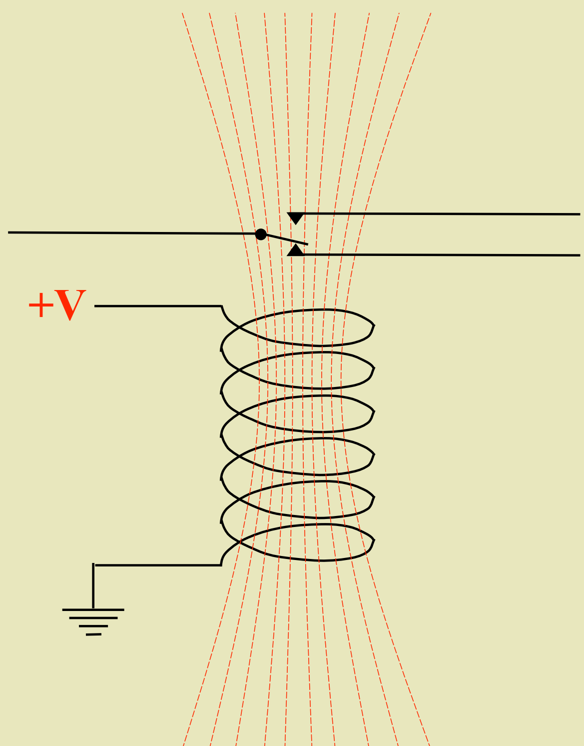
November 7, 2007



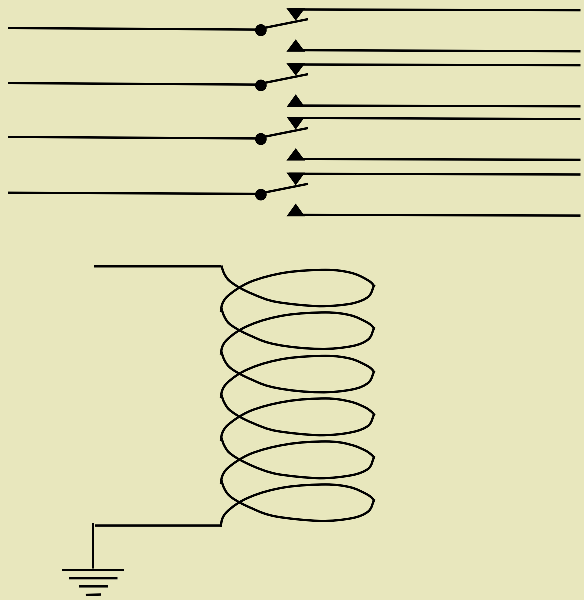
Double Throw Relay



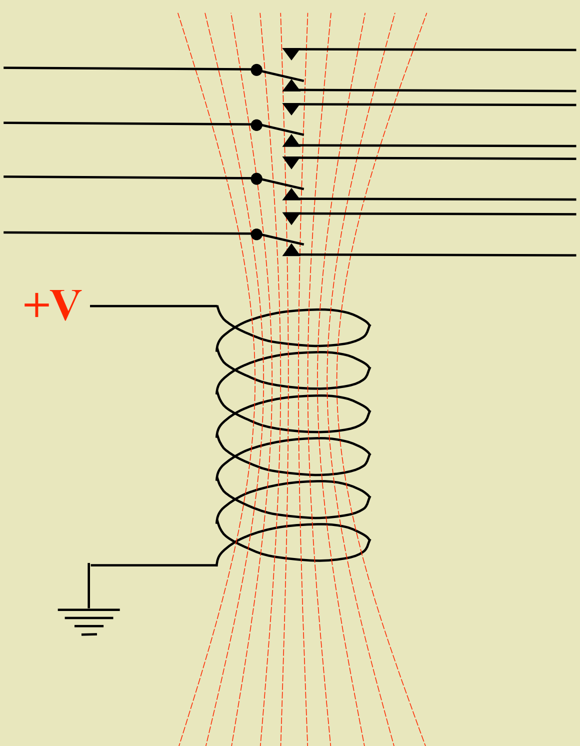
Double Throw Relay

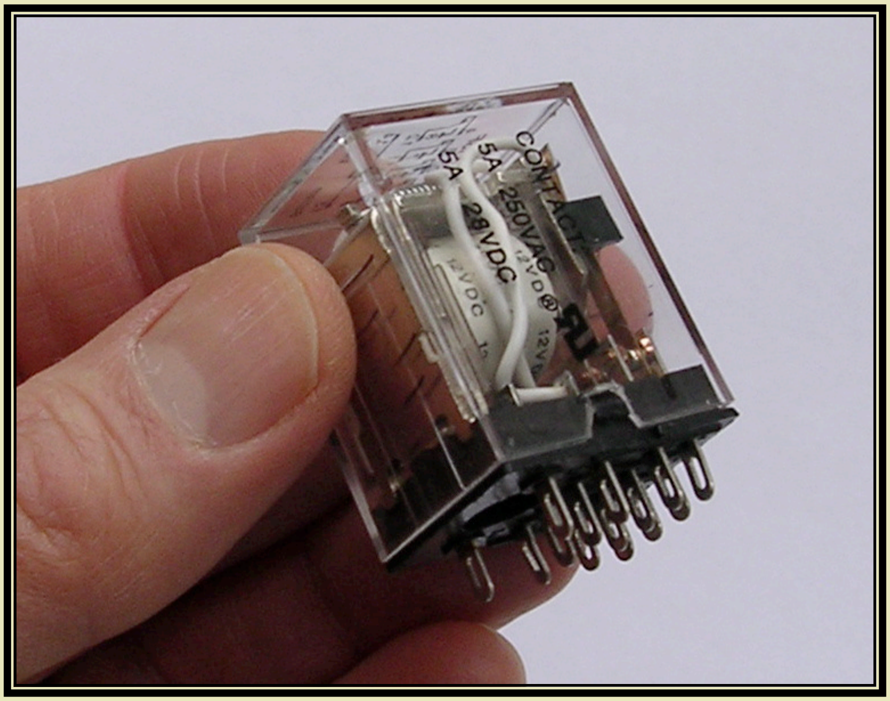


Four Pole, Double Throw Relay

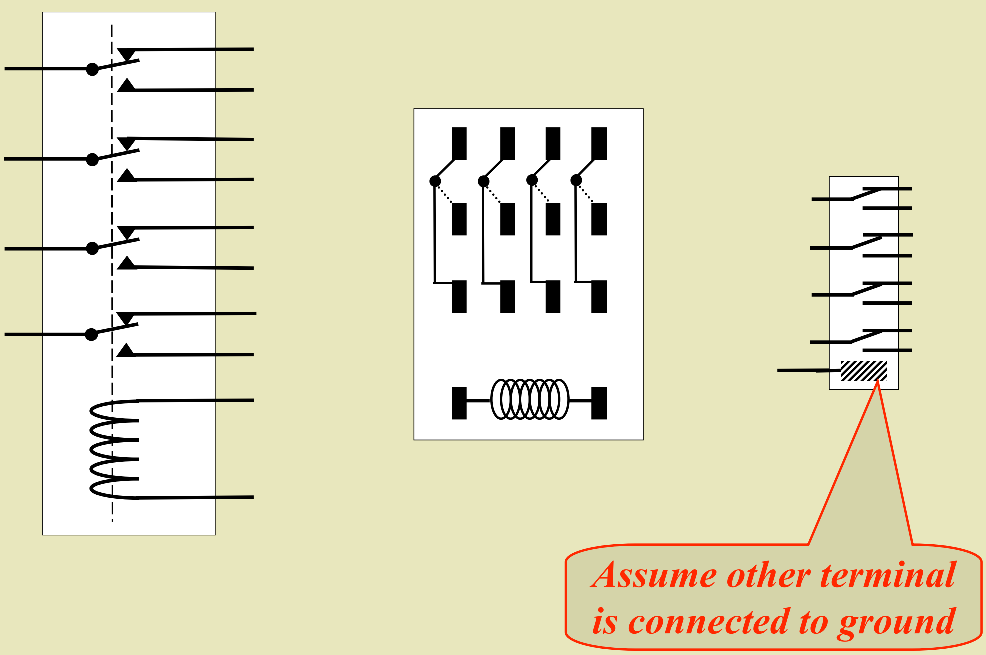


Four Pole, Double Throw Relay

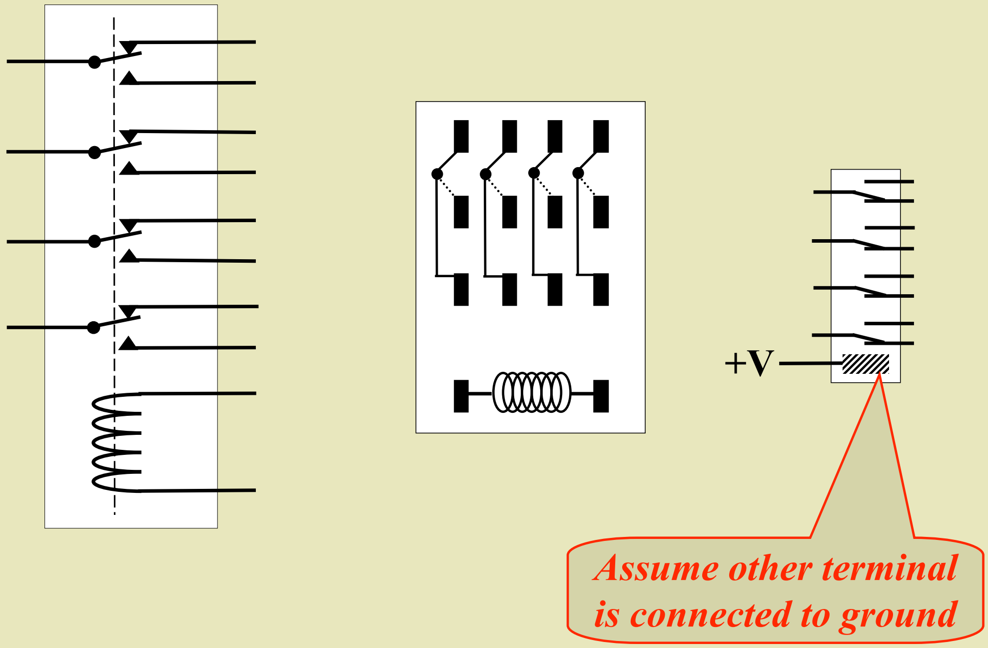




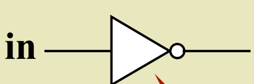
Schematic Diagrams



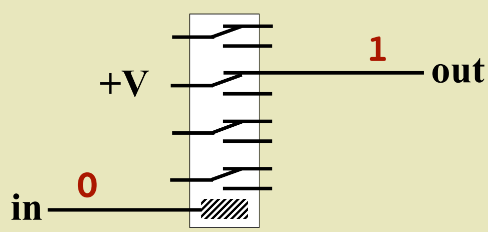
Schematic Diagrams



The “NOT” Circuit



in	out
0	1
1	0

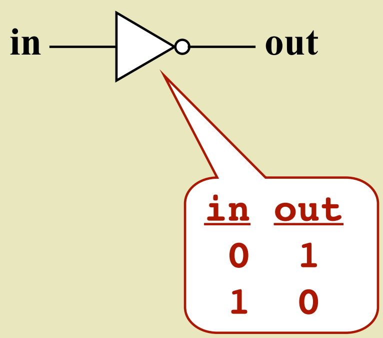


Convention:

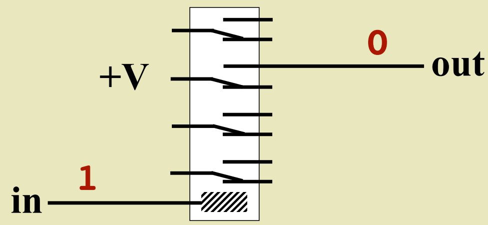
“1” = +12V

“0” = not connected

The “NOT” Circuit

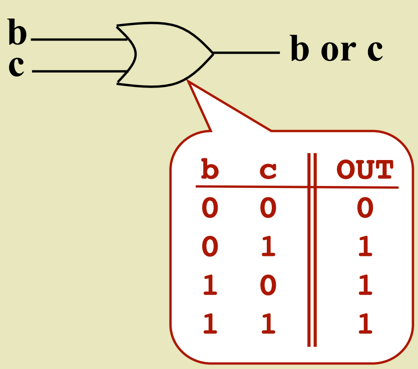


in	out
0	1
1	0

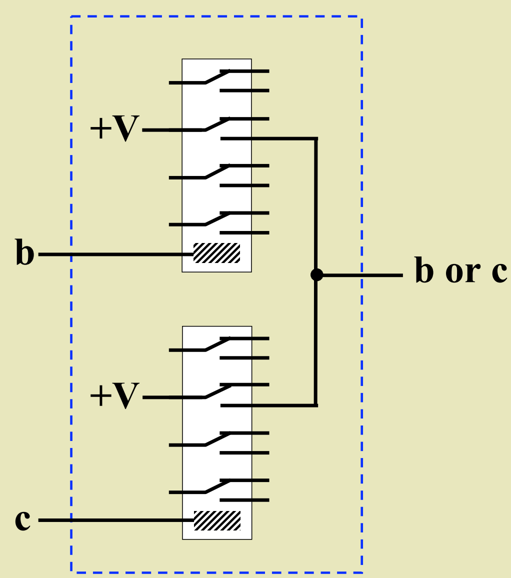


Convention:
“1” = +12V
“0” = not connected

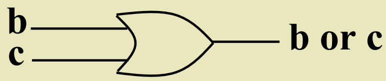
The “OR” Circuit

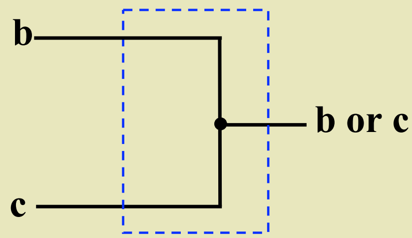


b	c	OUT
0	0	0
0	1	1
1	0	1
1	1	1

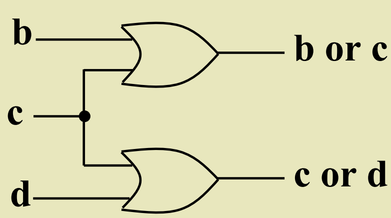


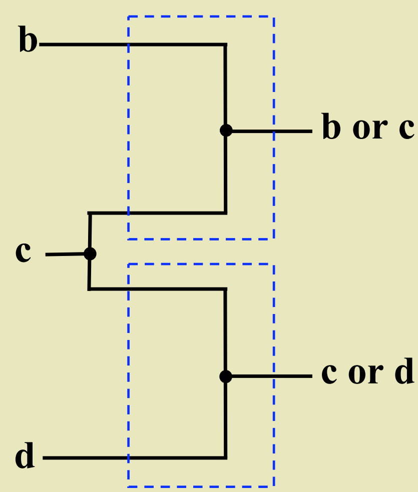
An Optimization?



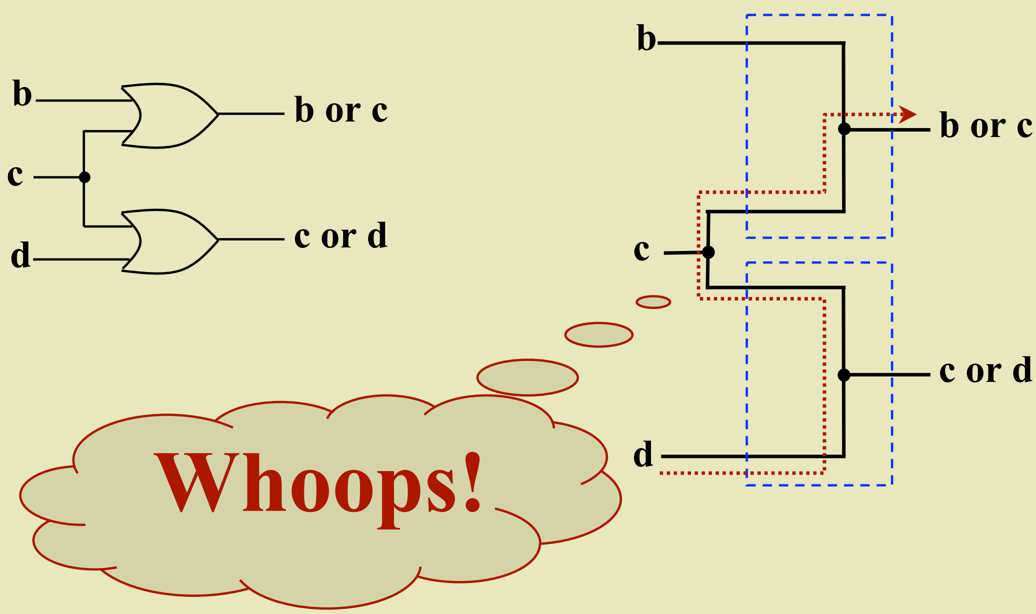


An Optimization?

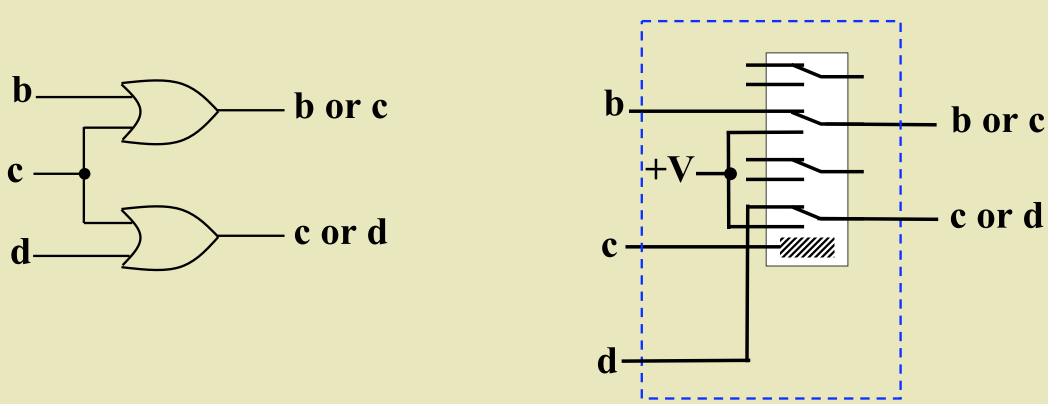




An Optimization?

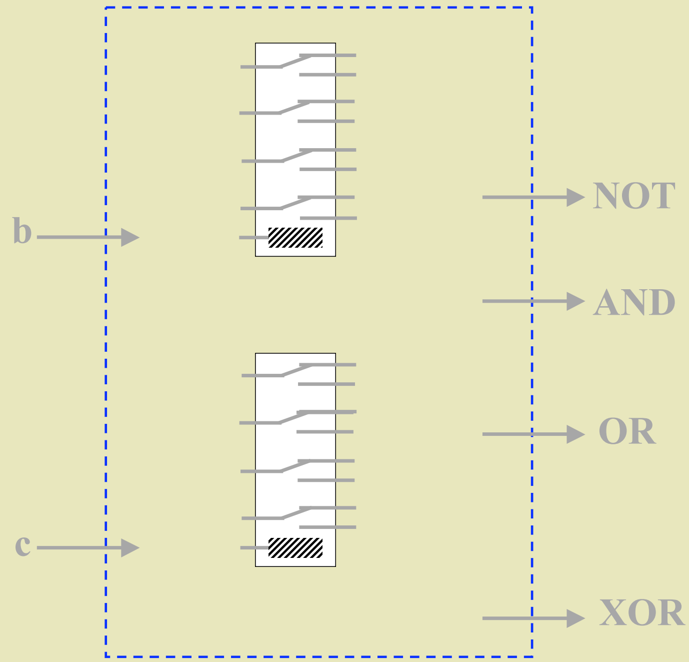


An Optimization?



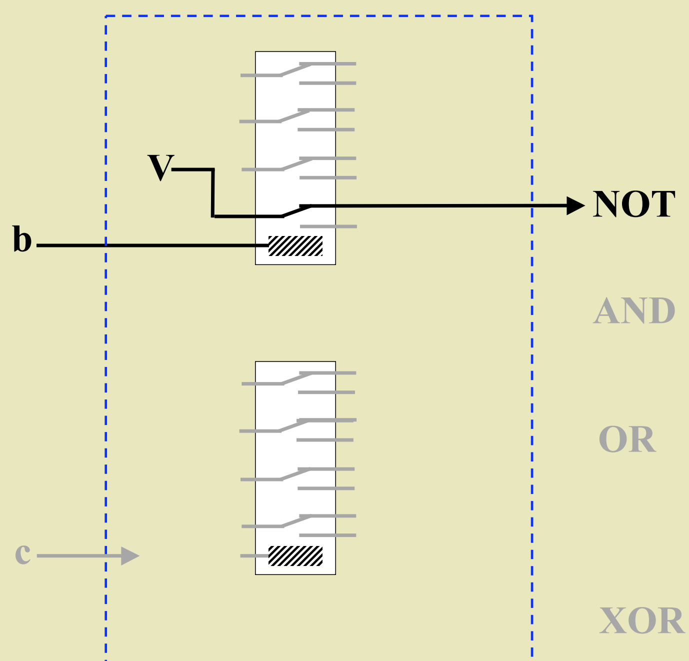
1-Bit Logic Circuit

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



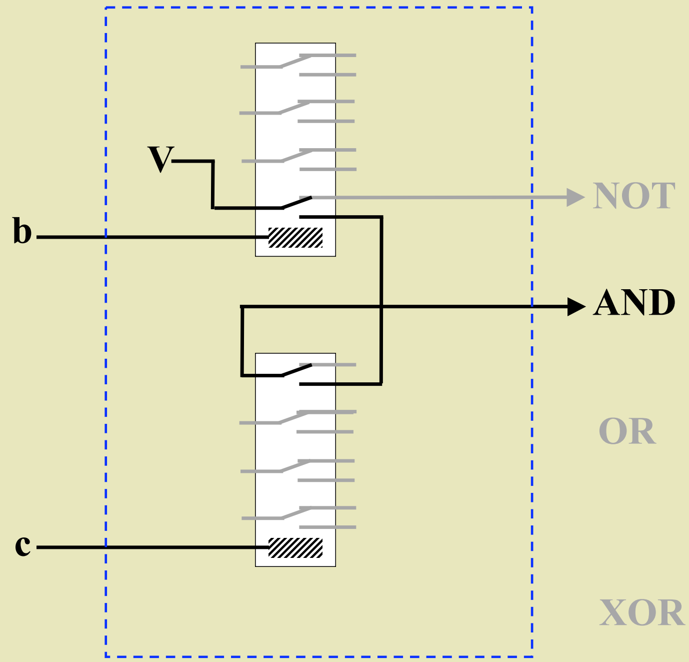
1-Bit Logic Circuit

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



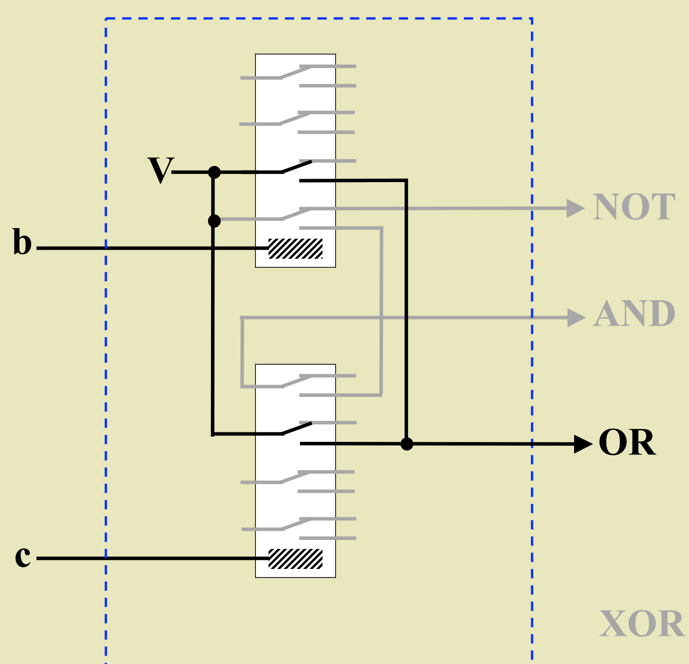
1-Bit Logic Circuit

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



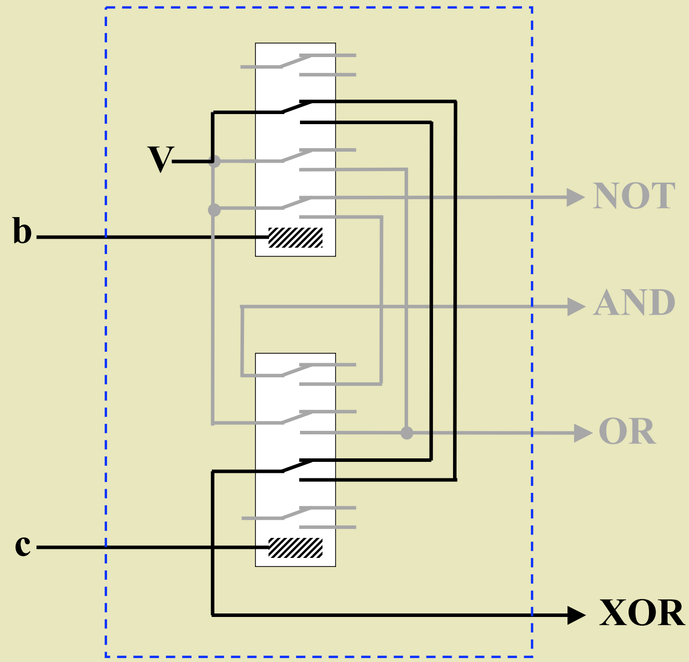
1-Bit Logic Circuit

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



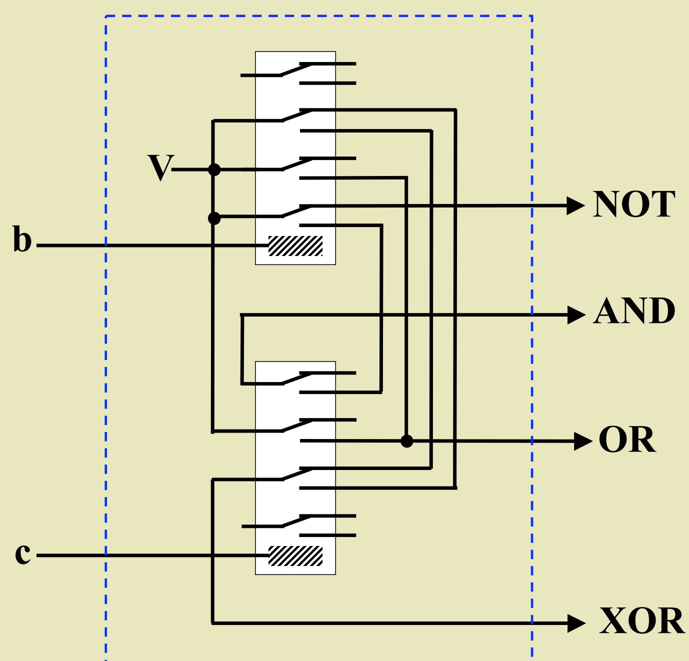
1-Bit Logic Circuit

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



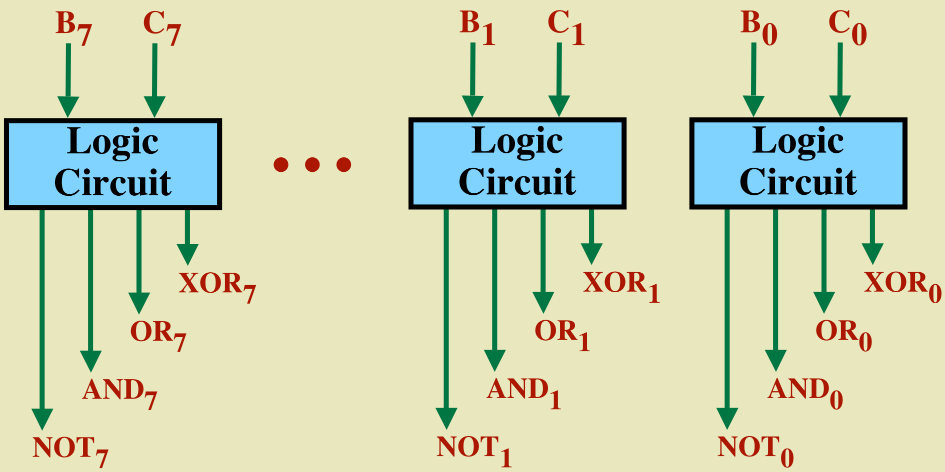
1-Bit Logic Circuit

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



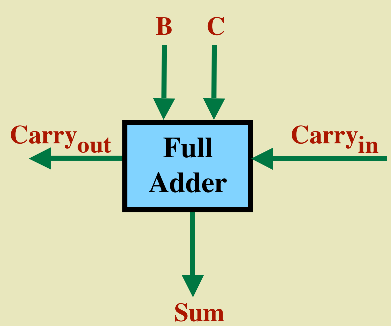
Eight 1-bit circuits can be combined to build an...

8-Bit Logic Circuit



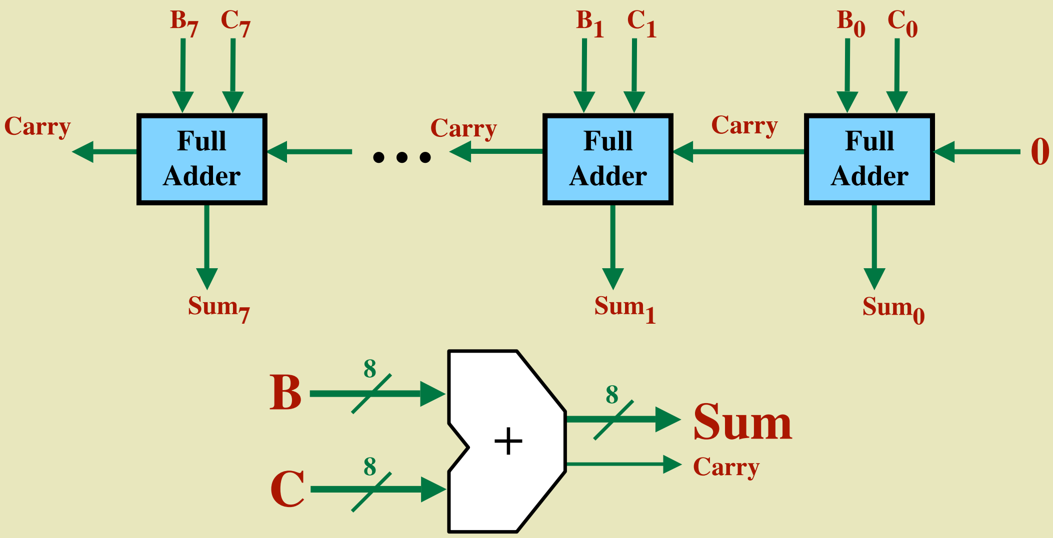
The “Full Adder” Circuit

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

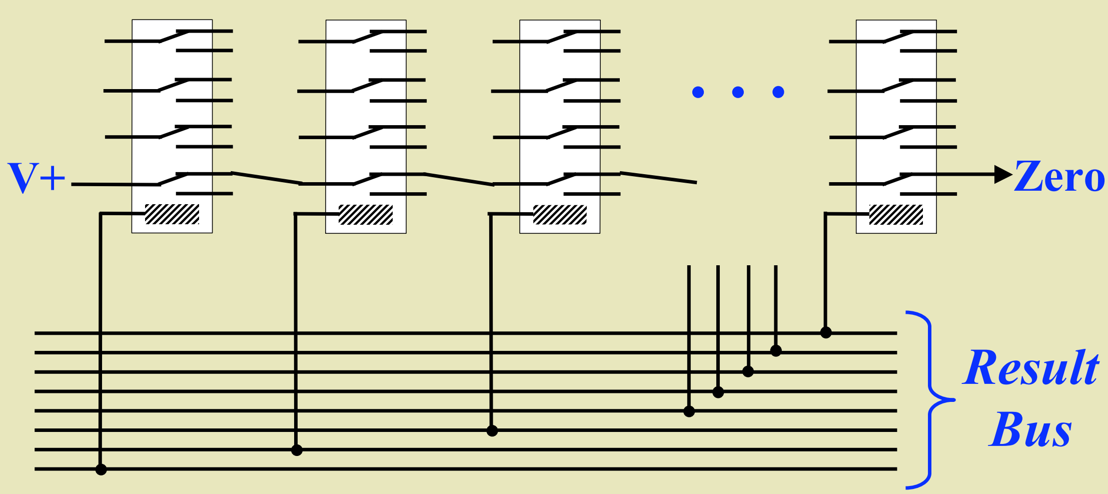


8 full-adders can be combined to build an...

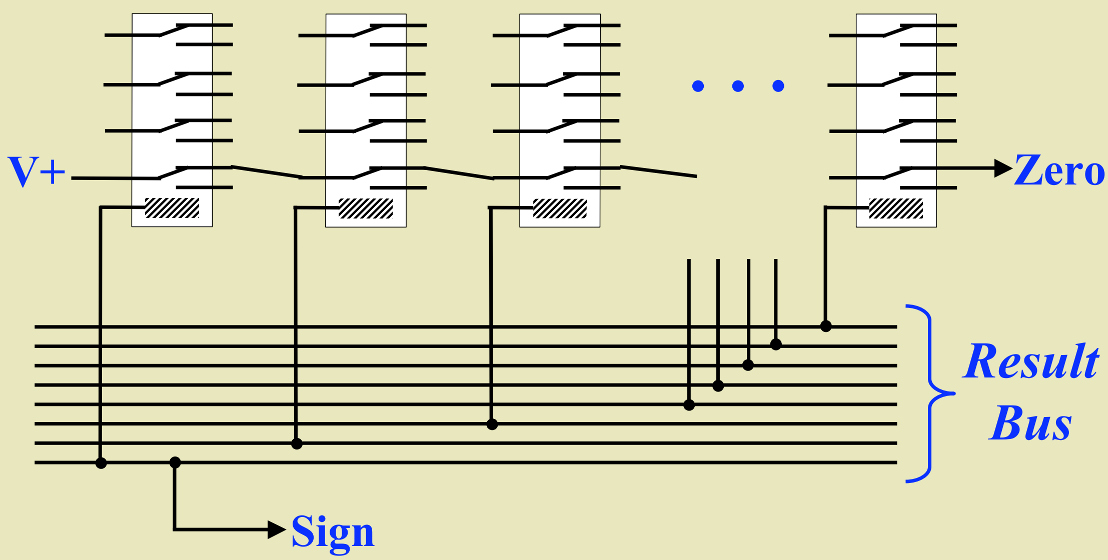
8-Bit Adder



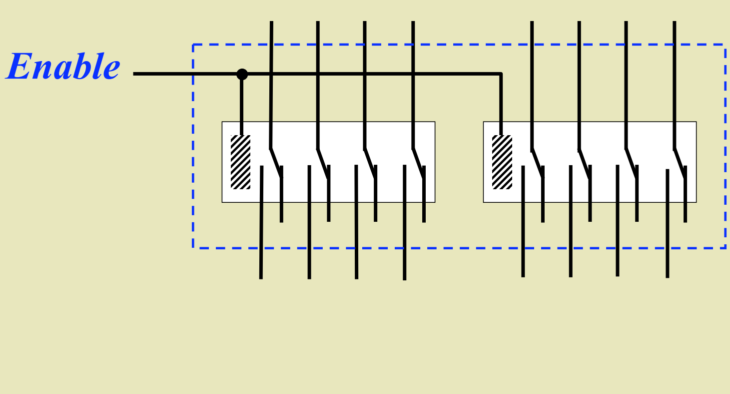
The Zero-Detect Circuit

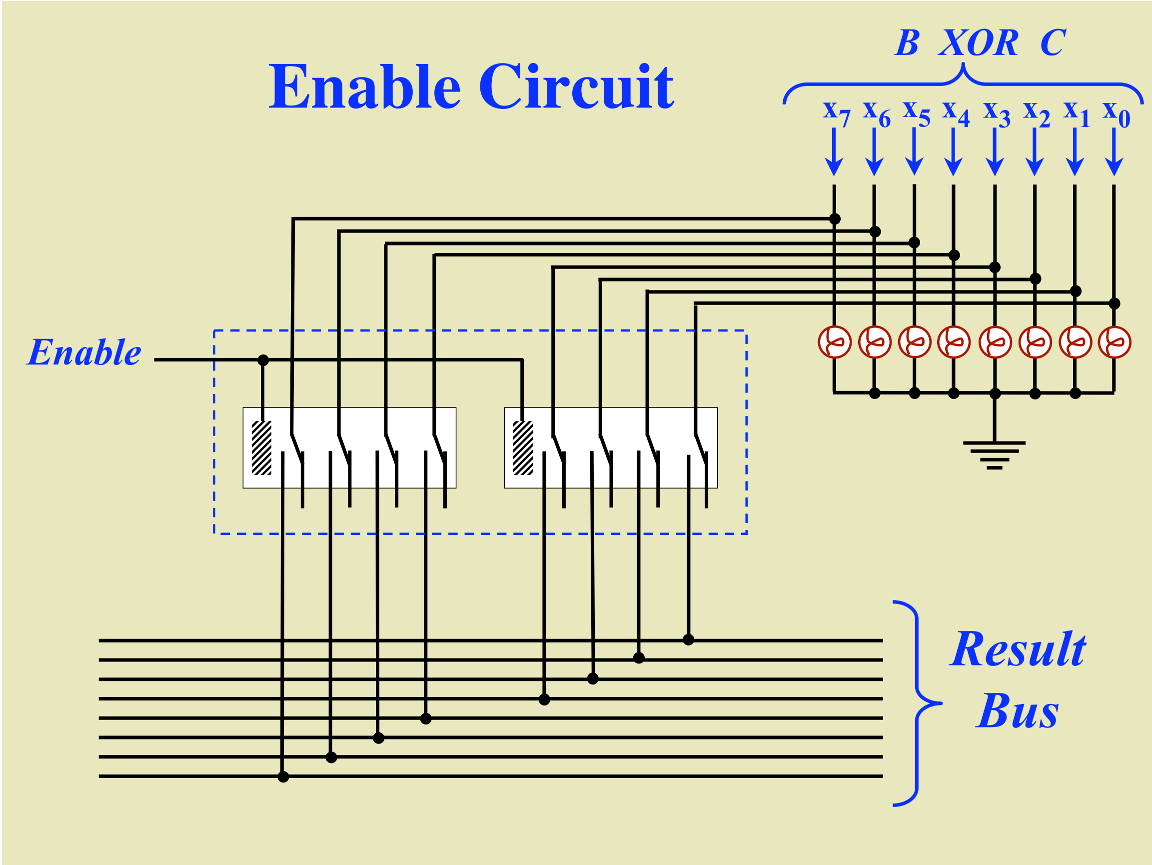


The Zero-Detect Circuit

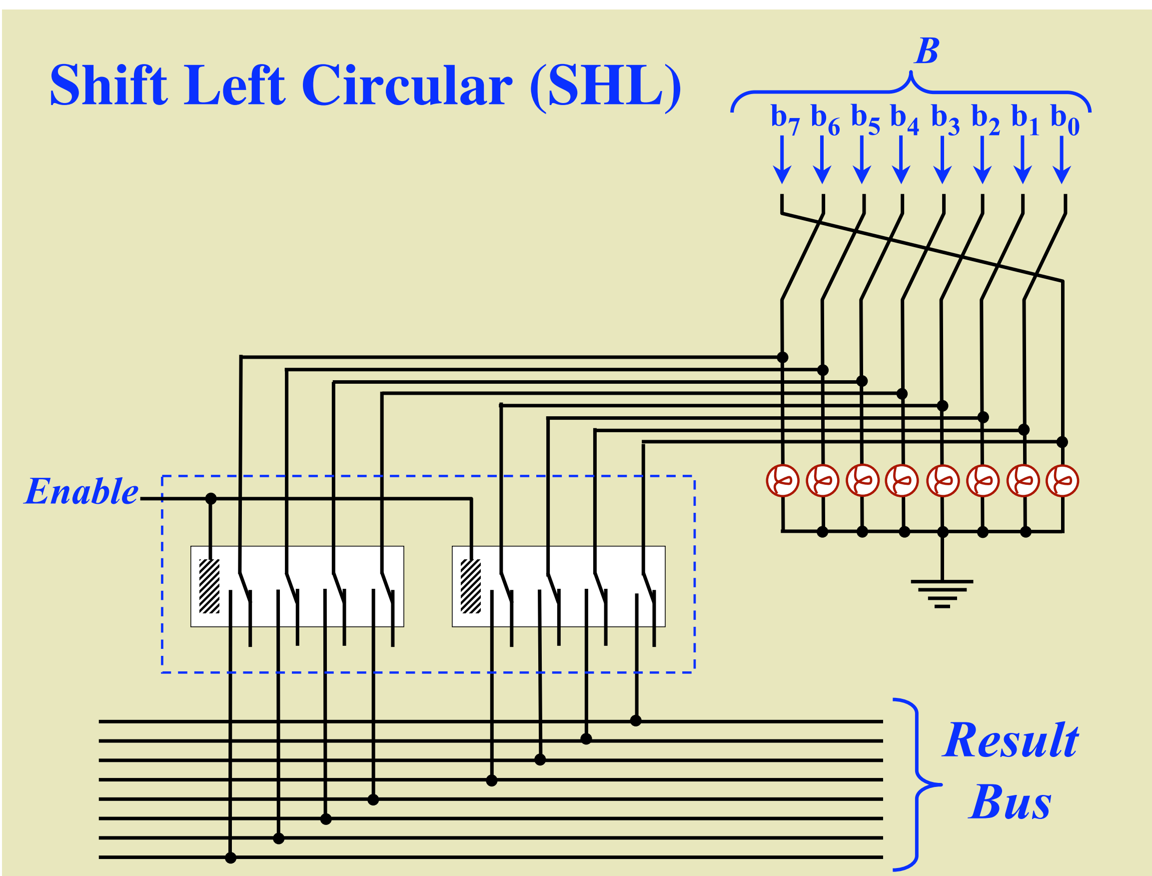


Enable Circuit





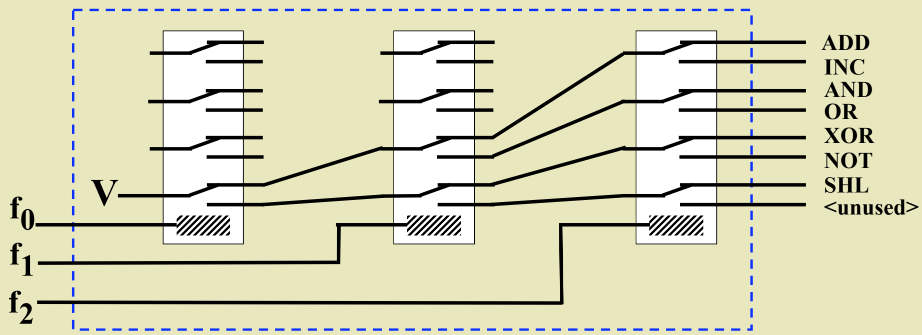
Enable Circuit
B XOR C
x7 x6 x5 x4 x3 x2 x1 x0
Enable
Result
Bus



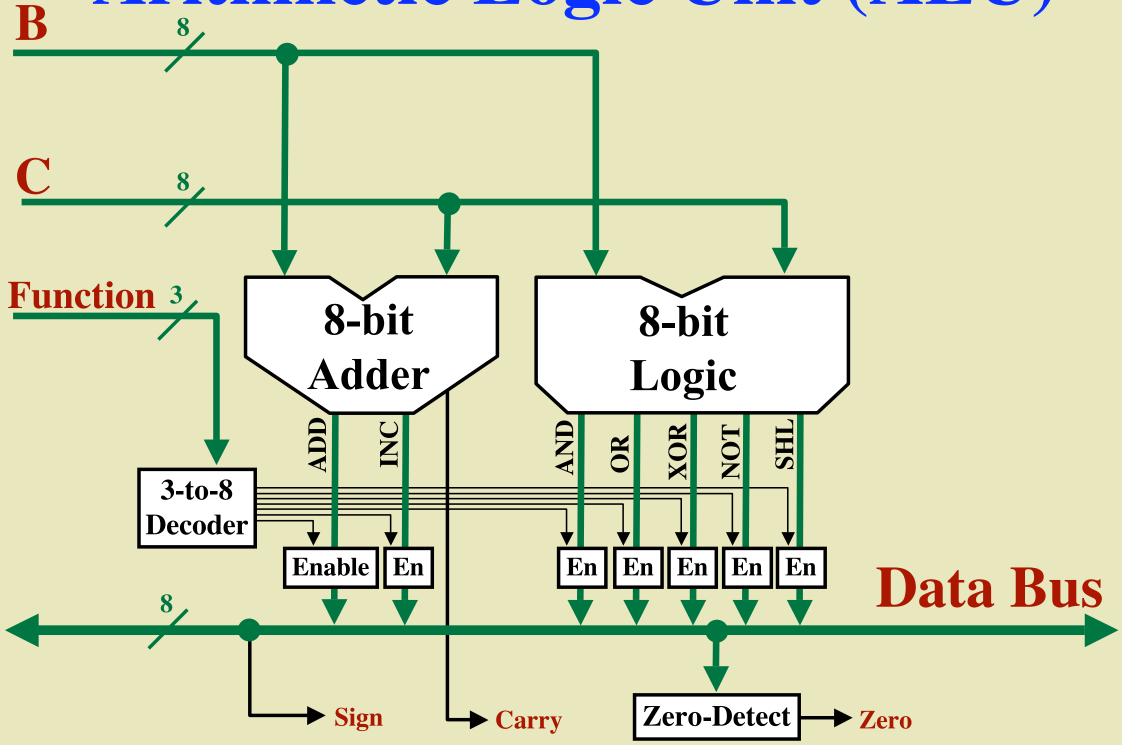
Shift Left Circular (SHL)
B
b7 b6 b5 b4 b3 b2 b1 b0
Enable
Result
Bus

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



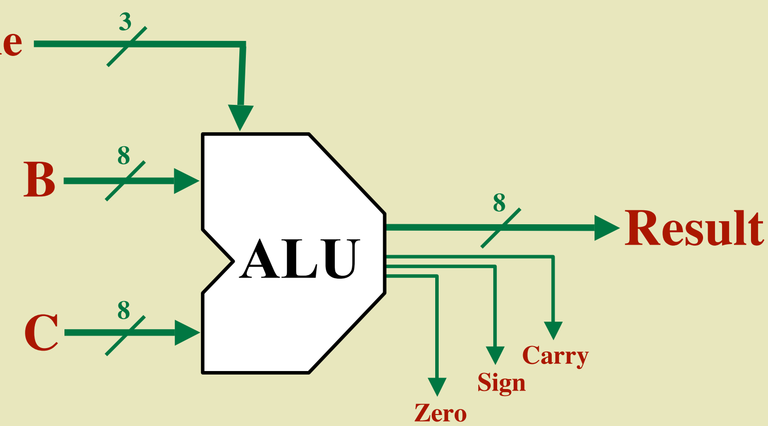
Arithmetic Logic Unit (ALU)



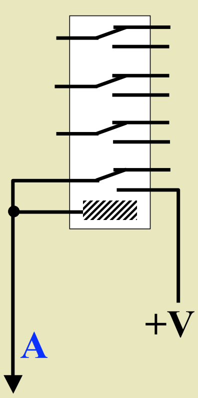
An 8-Bit ALU

Function Code

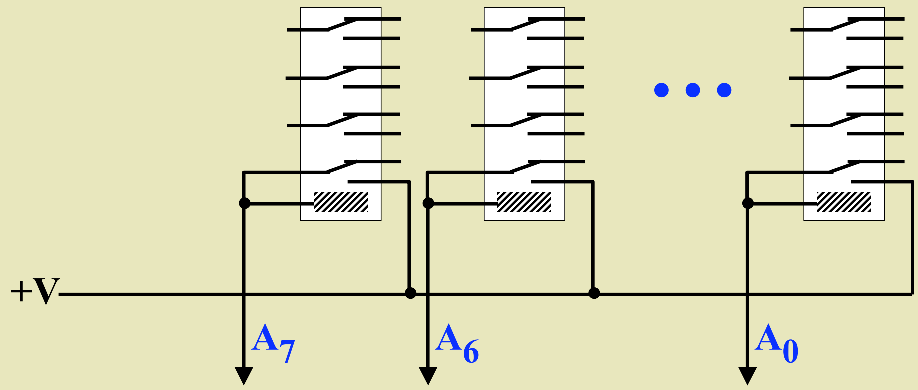
```
000 add
001 inc
010 and
011 or
100 xor
101 not
110 shl
111 <nop>
```



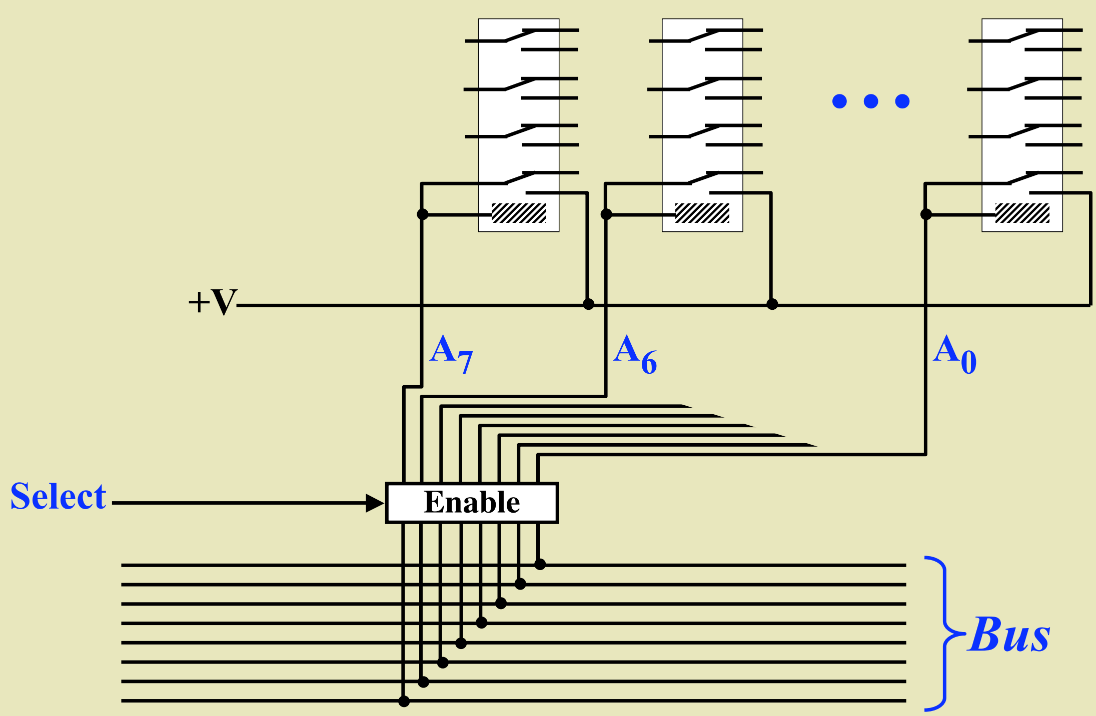
Register Storage



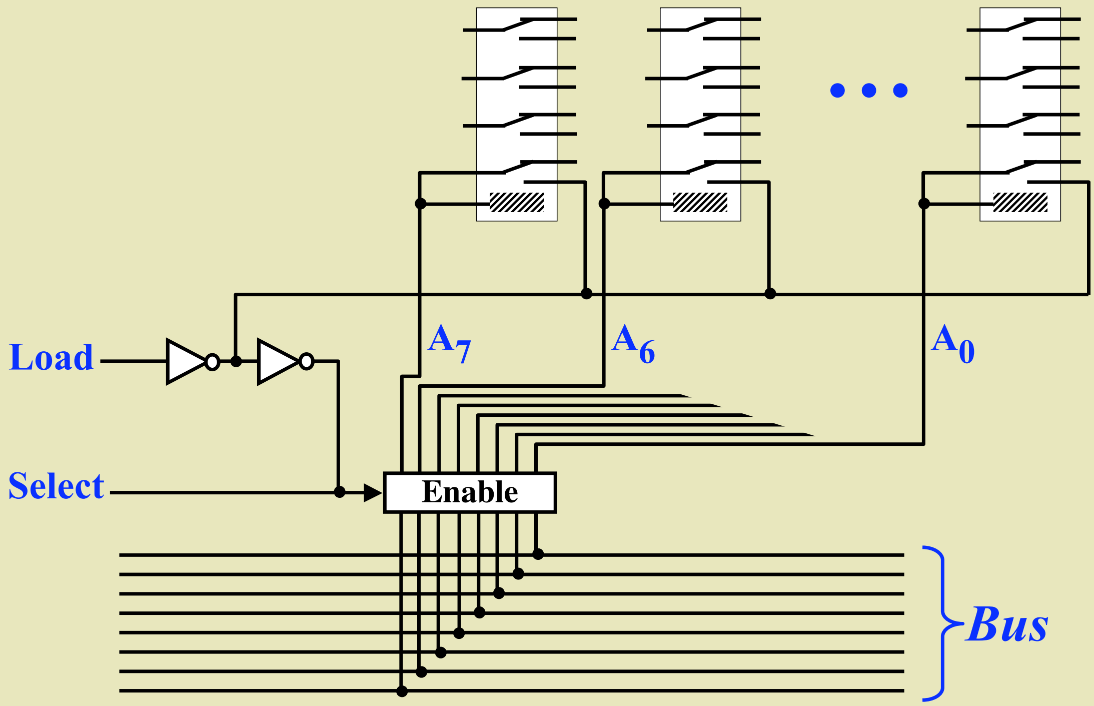
Register Storage



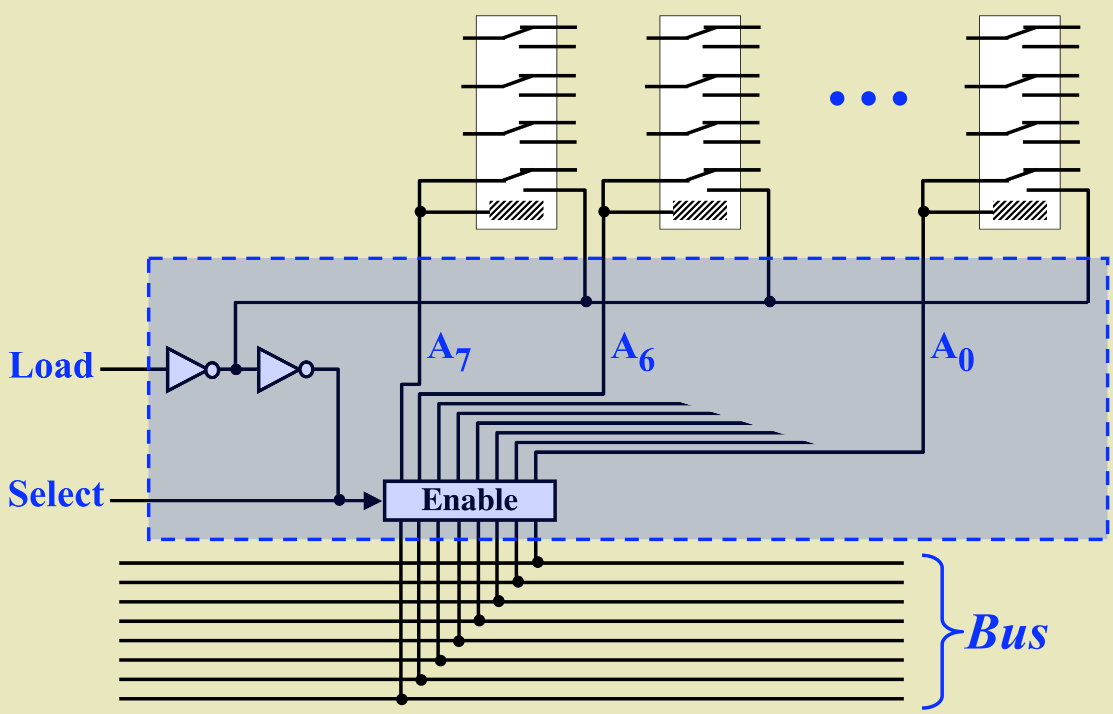
Register Storage



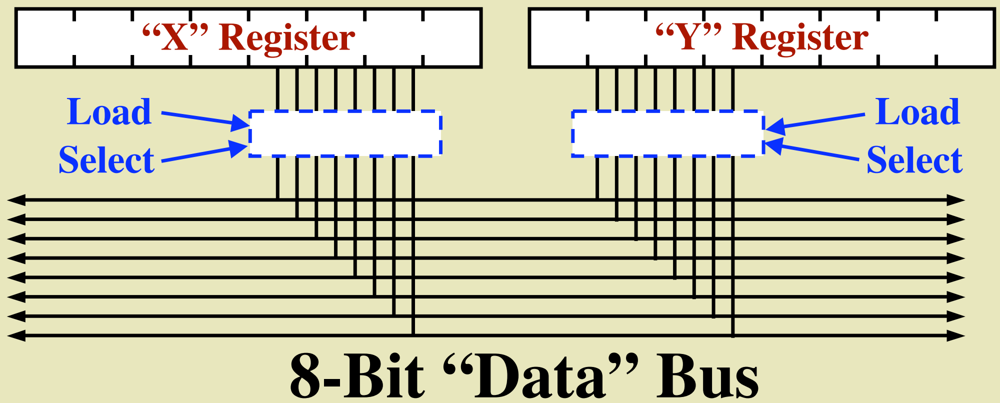
Register Storage

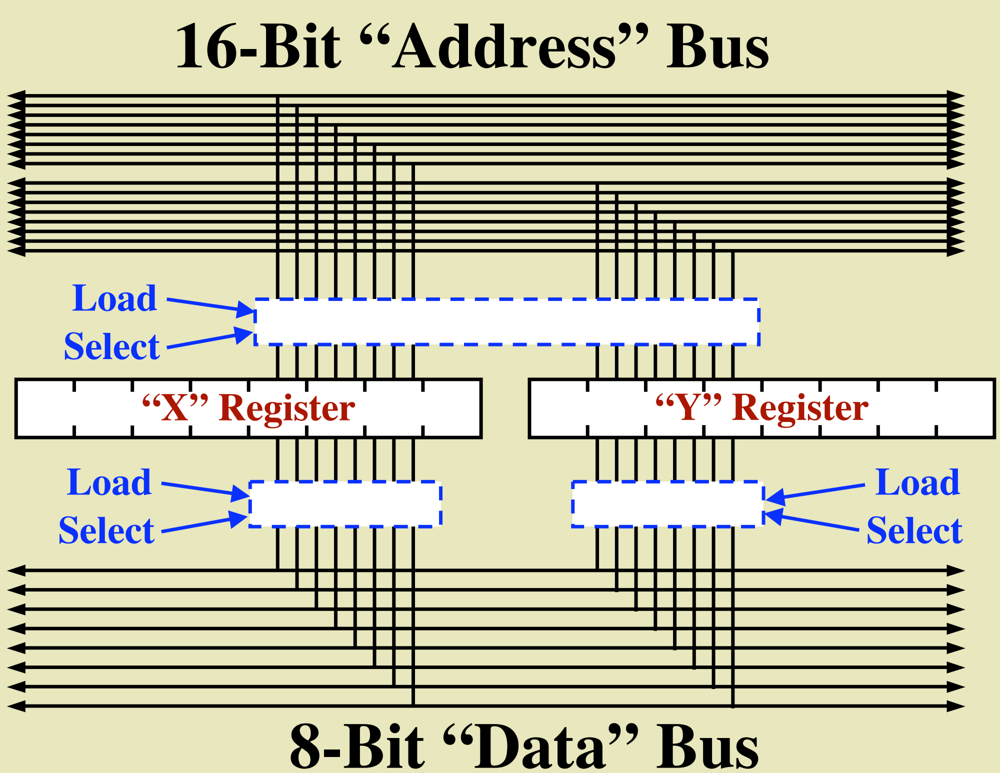


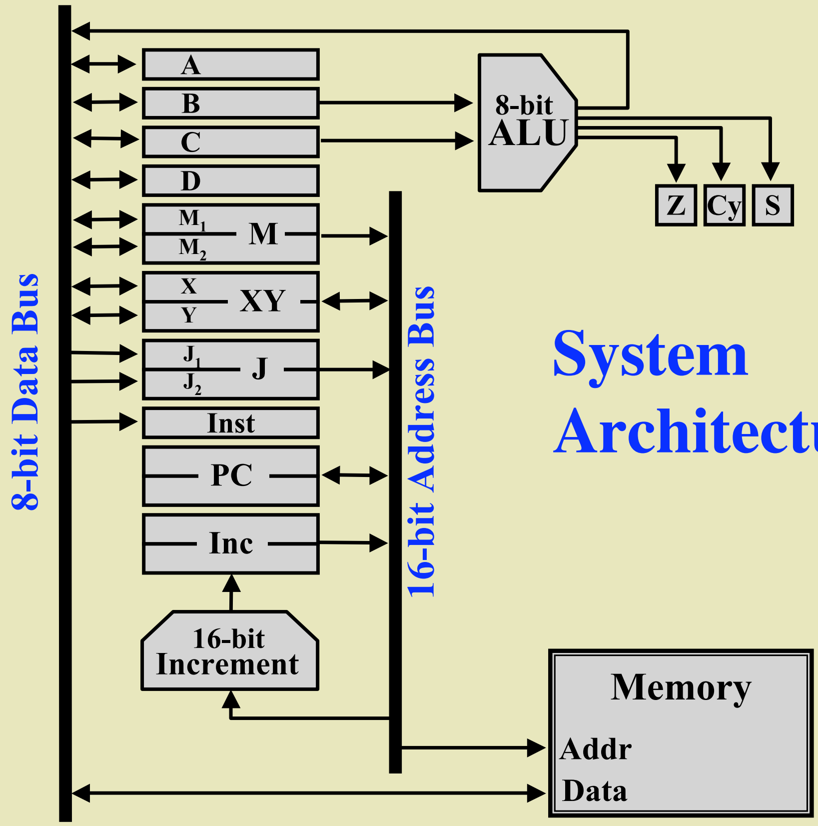
Register Storage



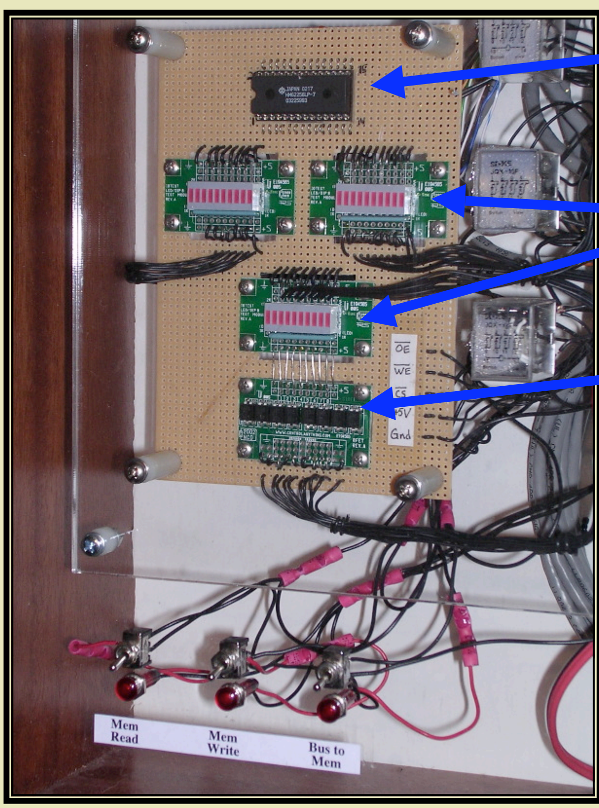
8-Bit Registers







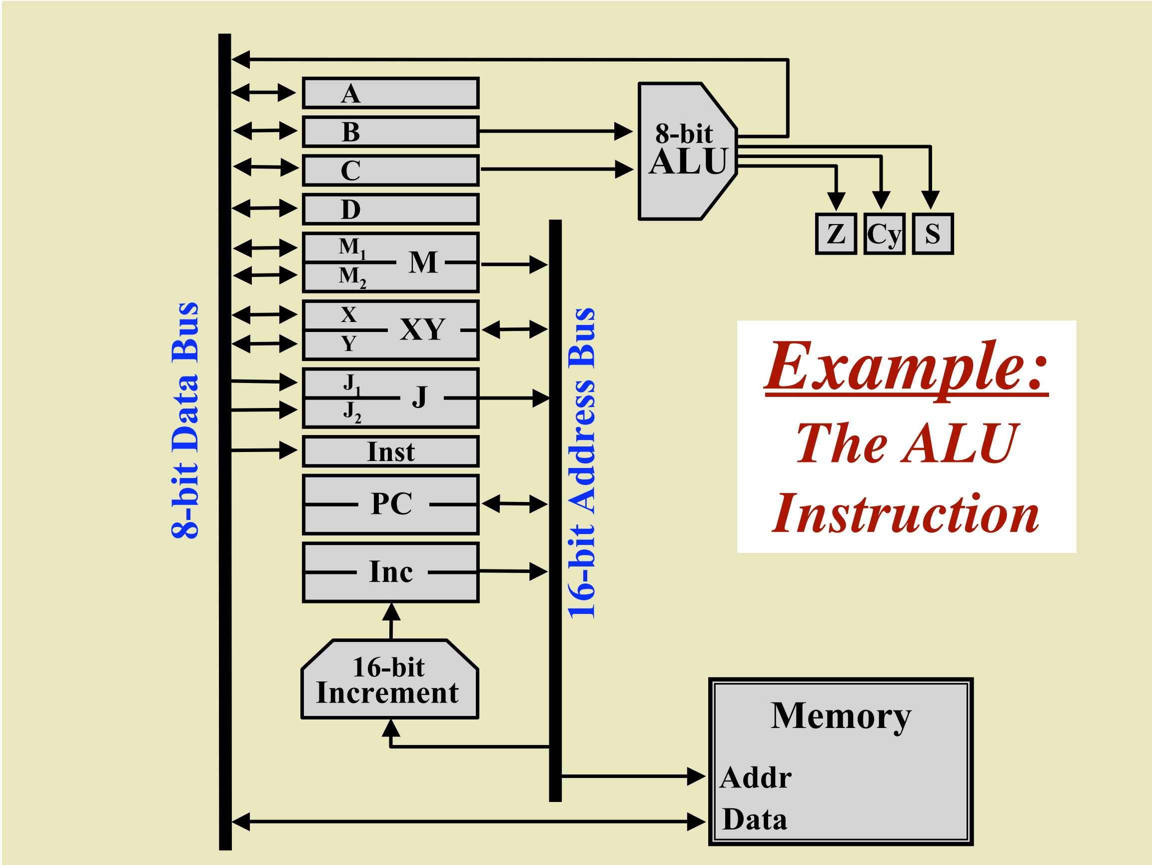
System Architecture



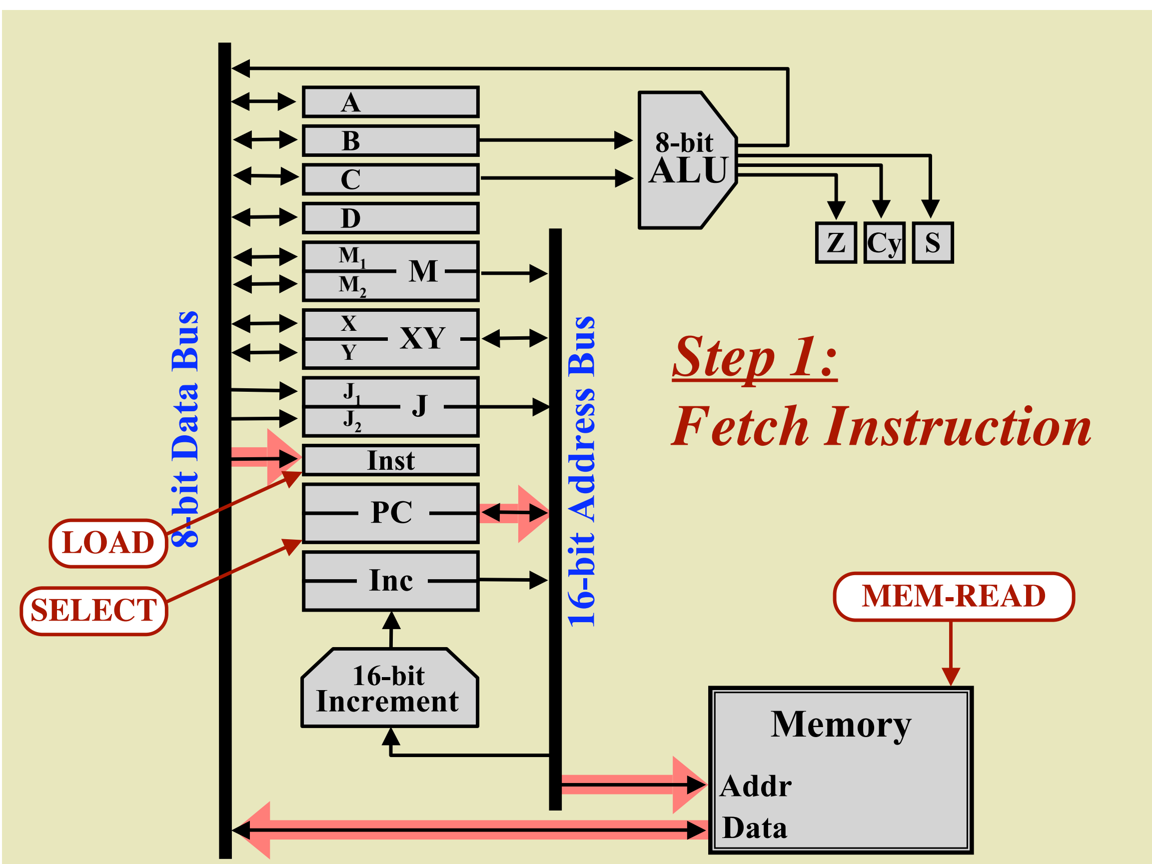
32K Byte Static RAM Chip

LEDs

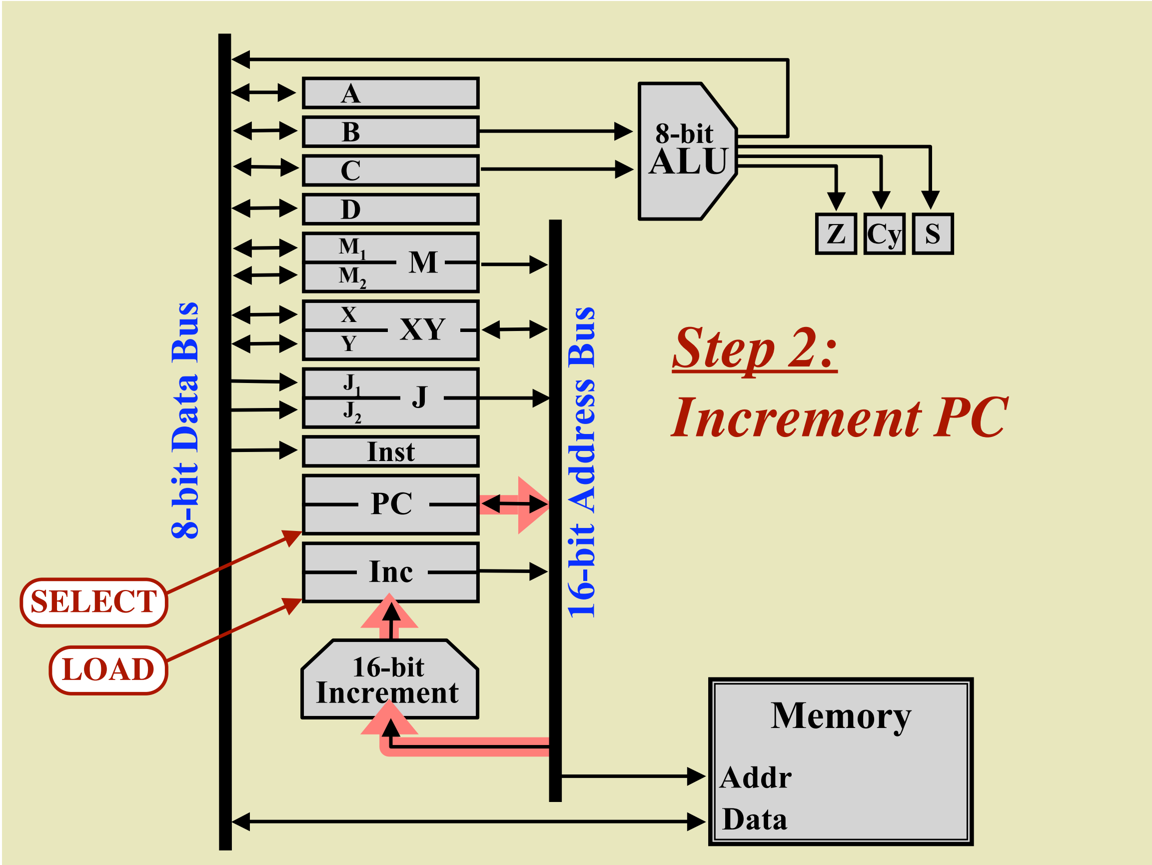
8 FET Power Transistors
(to drive relays during a memory-read operation)

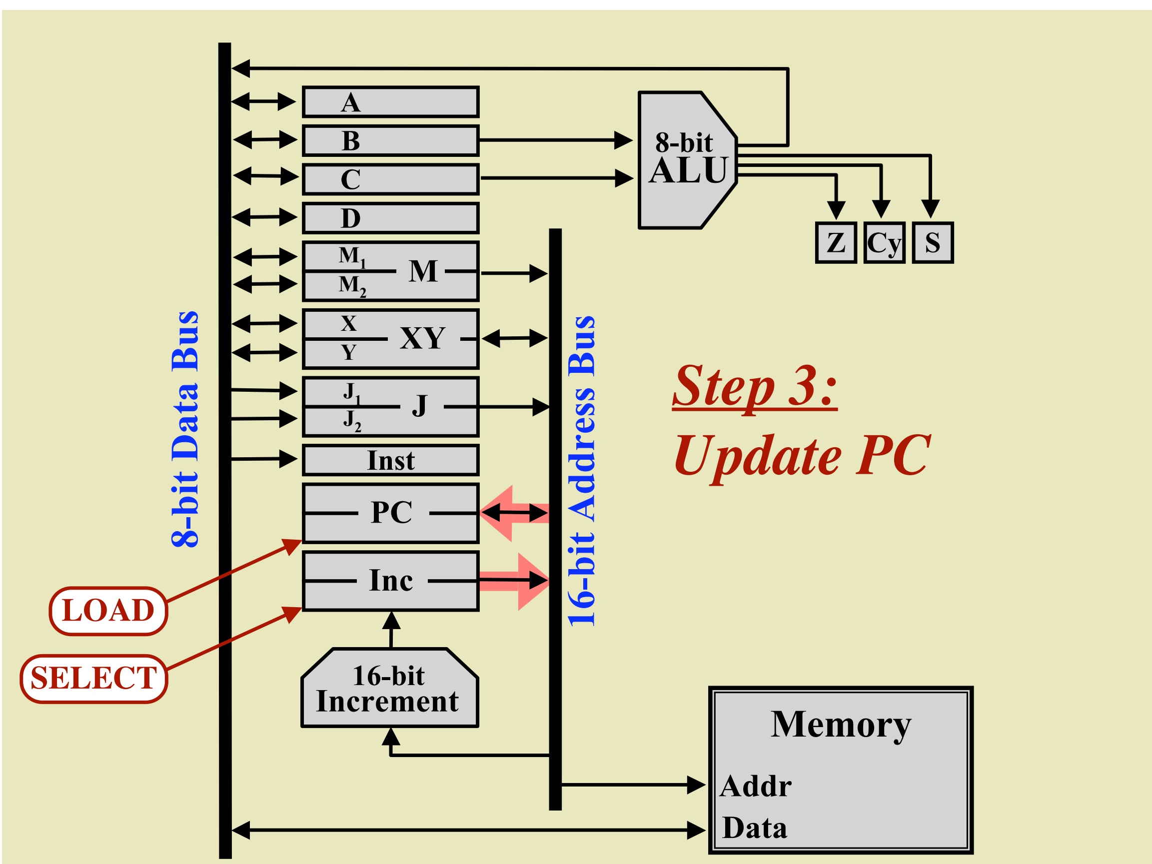


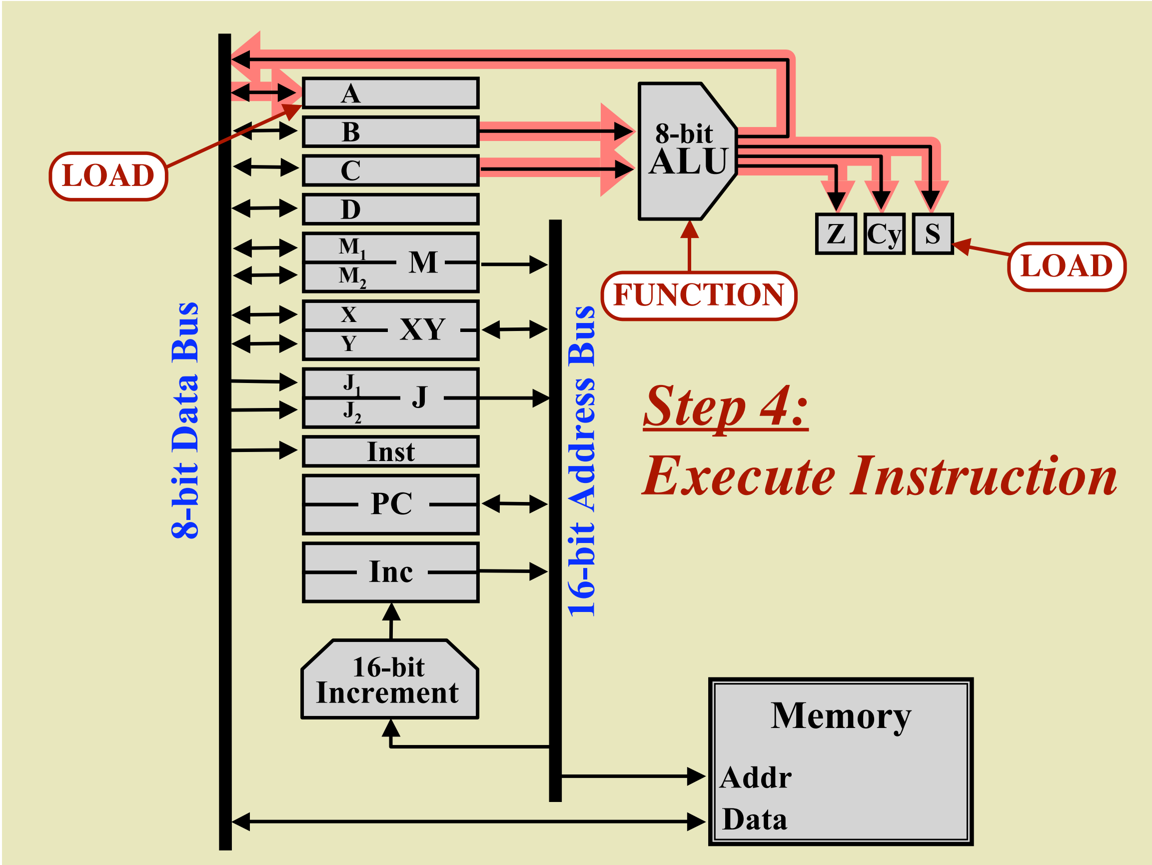
8-bit Data Bus
A
B
C
D
M1
M2
M
X
Y
XY
J1
J2
J
Inst
PC
Inc
16-bit Increment
16-bit Address Bus
8-bit ALU
Z
Cy
S
Example:
The ALU Instruction
Memory
Addr
Data



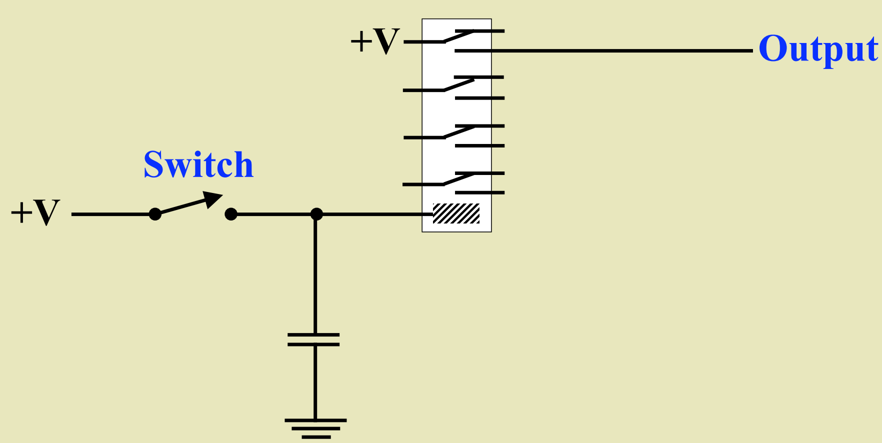
8-bit Data Bus
A
B
C
D
M1
M2
M
X
Y
XY
J1
J2
J
Inst
PC
Inc
16-bit Increment
16-bit Address Bus
8-bit ALU
Z
Cy
S
Step 1:
Fetch Instruction
LOAD
SELECT
MEM-READ
Memory
Addr
Data


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
Step 2:
Increment PC
SELECT
LOAD
16-bit
Increment
Memory
Addr
Data

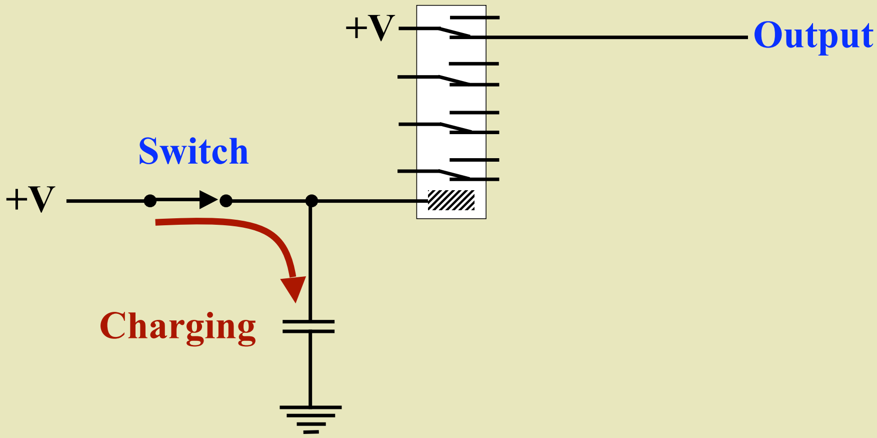

A
B
C
D
M1
M2
M
X
Y
XY
J1
J2
J
Inst
PC
Inc
8-bit
ALU
Z
Cy
S
8-bit Data Bus
16-bit Address Bus
Step 3:
Update PC
LOAD
SELECT
16-bit
Increment
Memory
Addr
Data


A
B
C
D
M1
M2
M
X
Y
XY
J1
J2
J
Inst
PC
Inc
16-bit Increment
8-bit Data Bus
16-bit Address Bus
8-bit ALU
LOAD
FUNCTION
Z
Cy
S
LOAD
Memory
Addr
Data
Step 4:
Execute Instruction

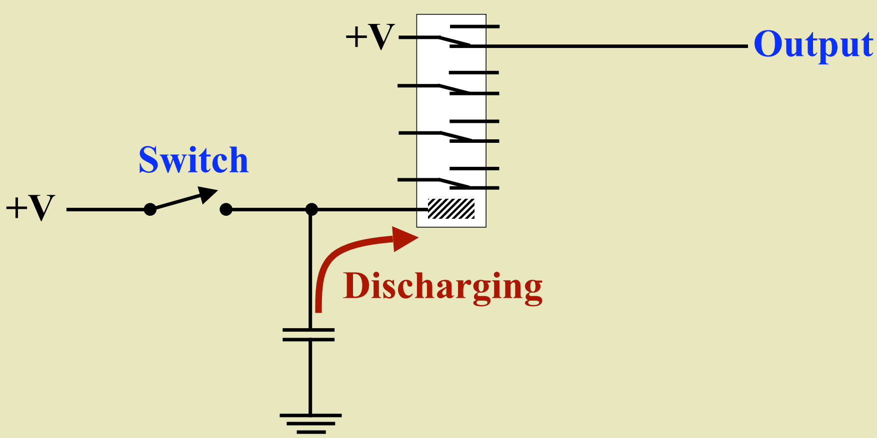
Clock


+V
Output
Switch
+V

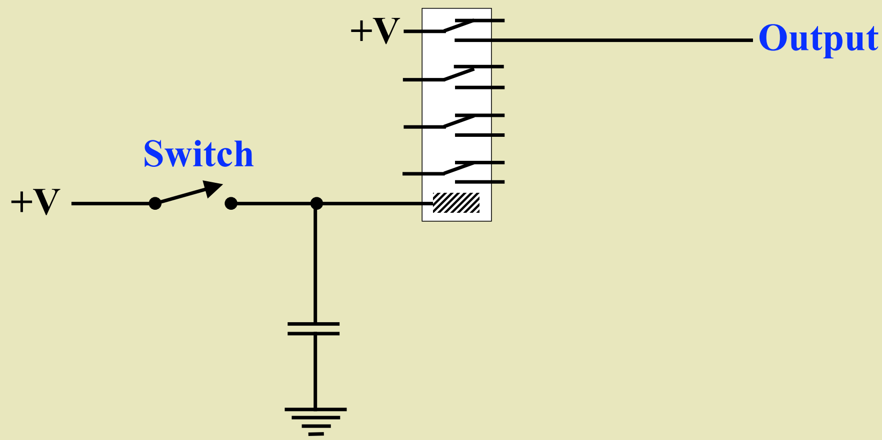
Clock



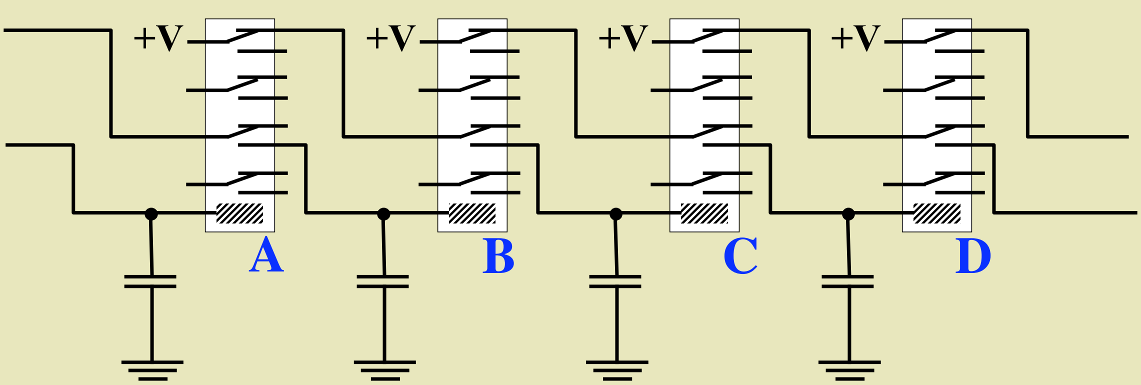
Clock



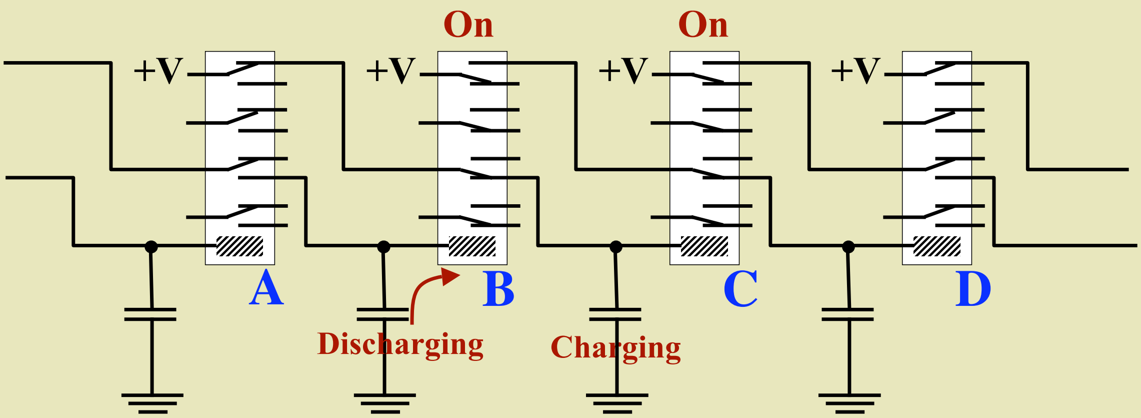
Clock



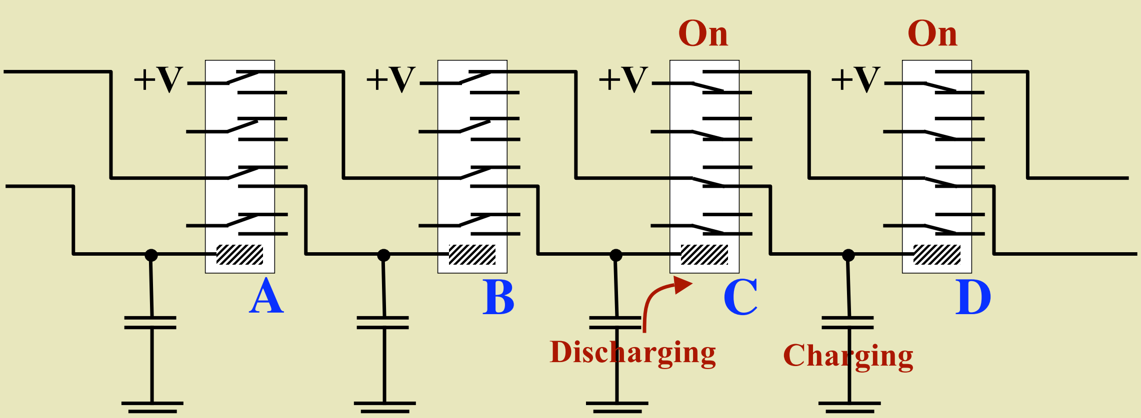
Clock



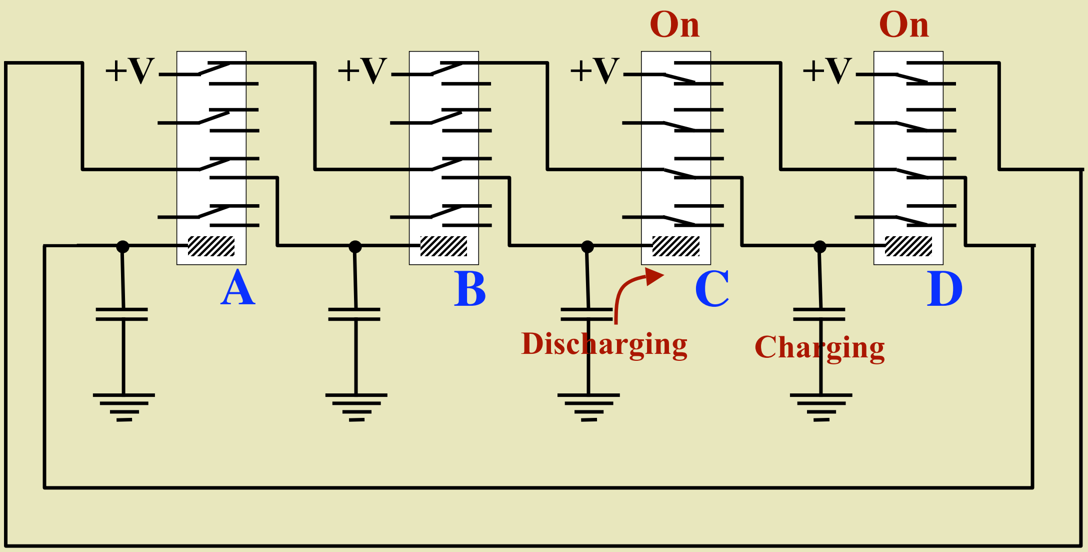
Clock


On
On
+V
+V
+V
+V
A
B
C
D
Discharging
Charging

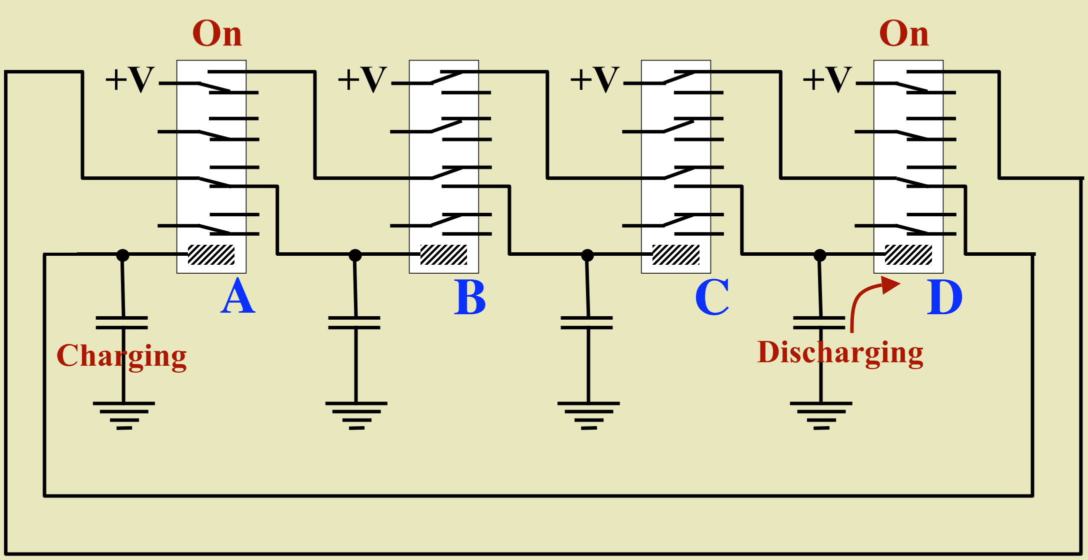
Clock


On
On
+V
+V
+V
+V
A
B
C
D
Discharging
Charging

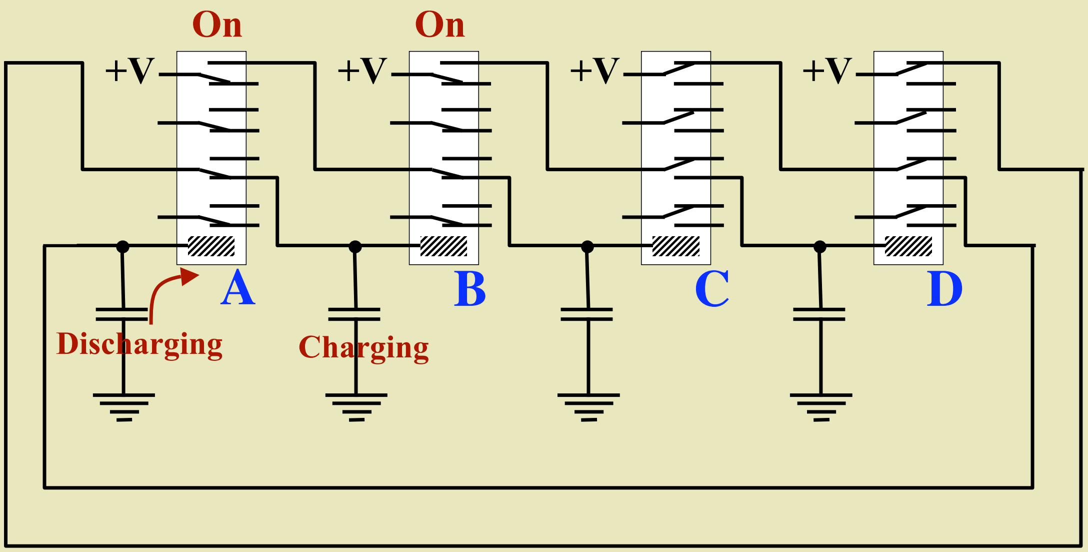
Clock


On
On
+V
+V
+V
+V
A
B
C
D
Discharging
Charging

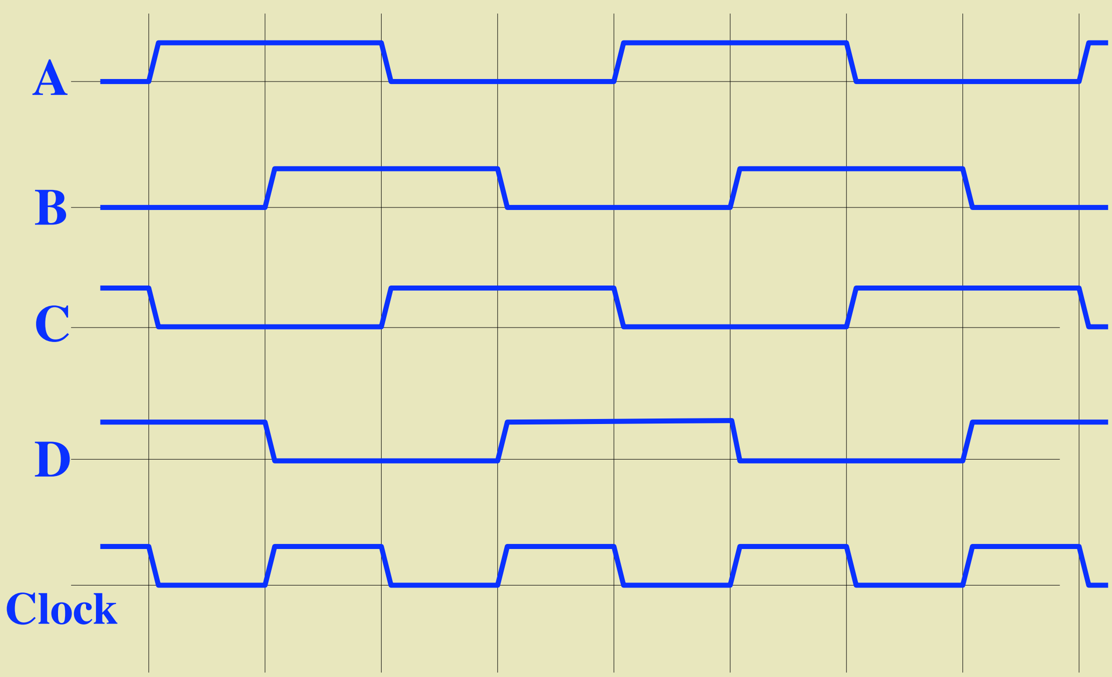
Clock


On
On
+V
+V
+V
+V
A
B
C
D
Charging
Discharging

Clock

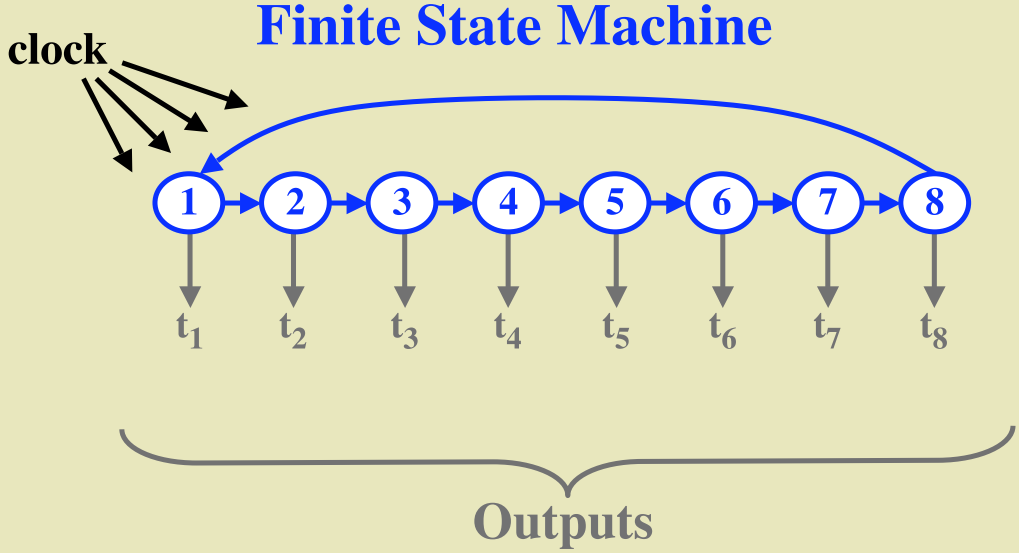


Clock Timing Diagram

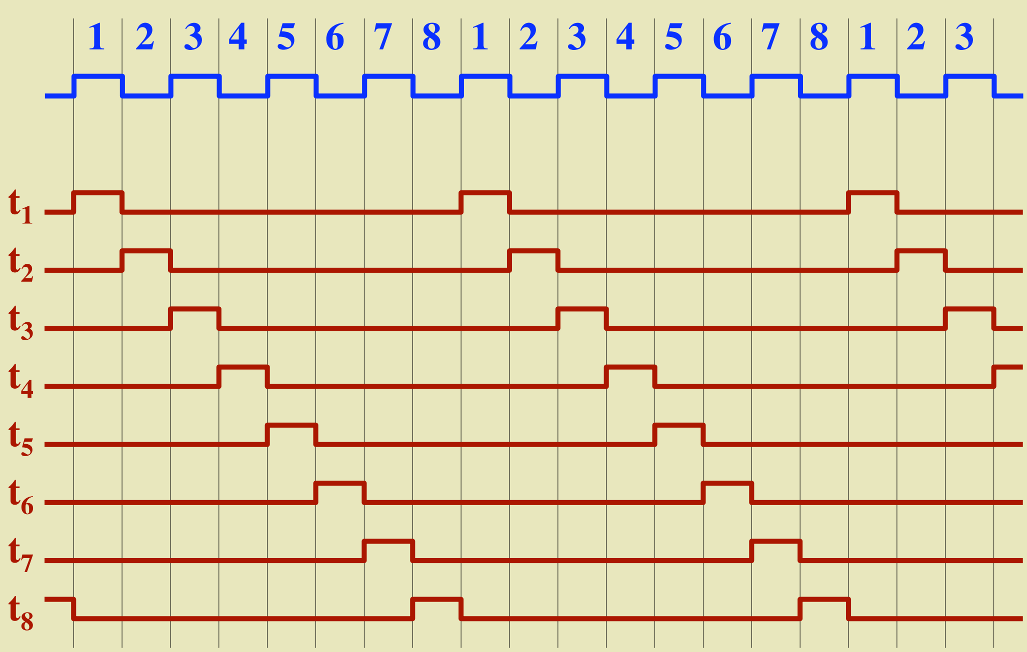


Clock = (A and B) or (C and D)

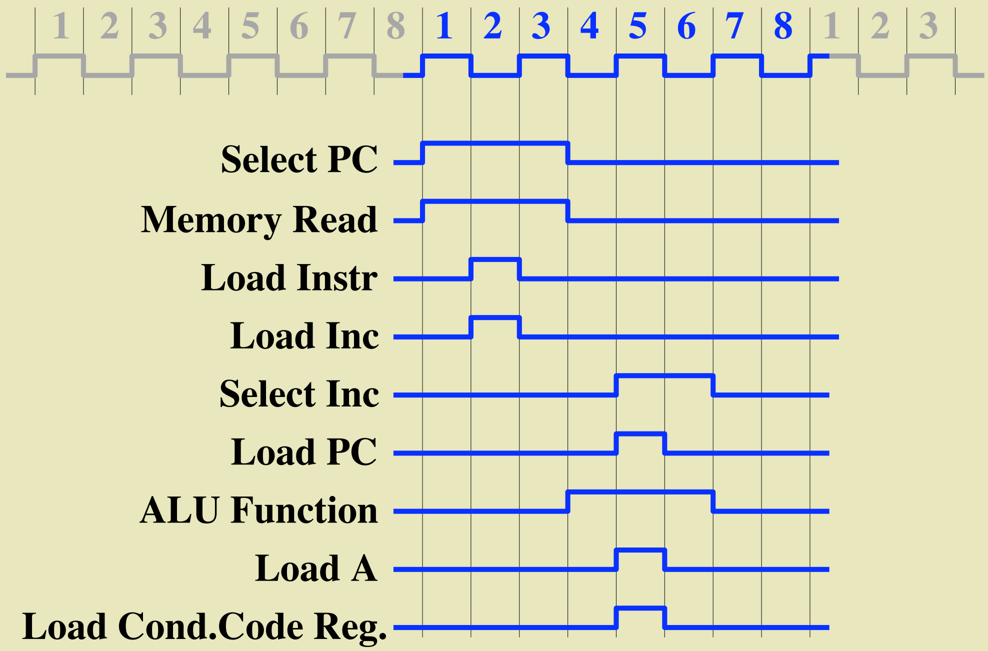
Finite State Machine



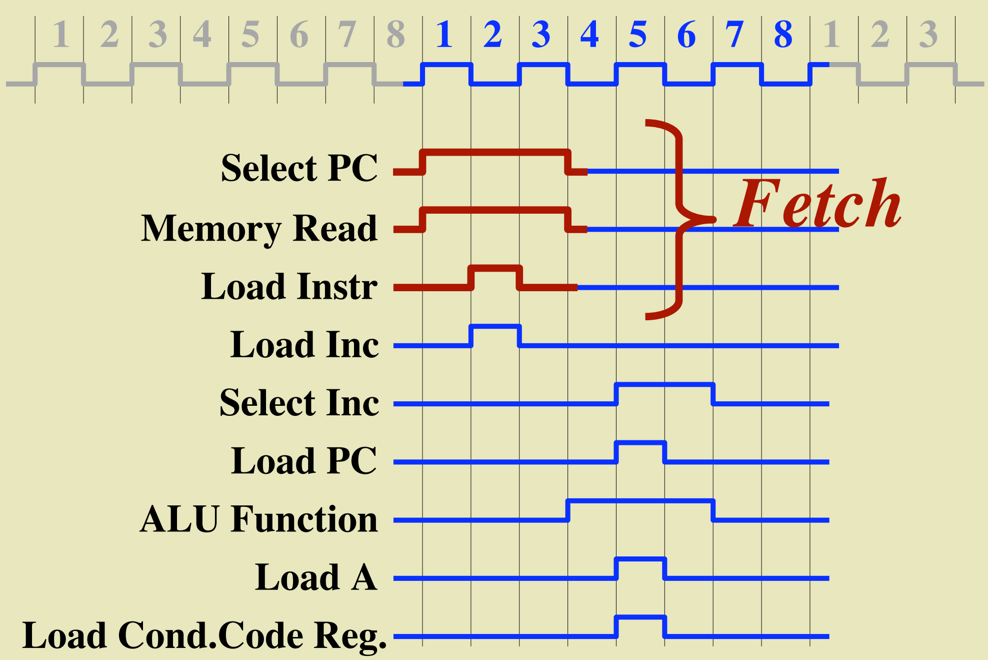
Output from FSA



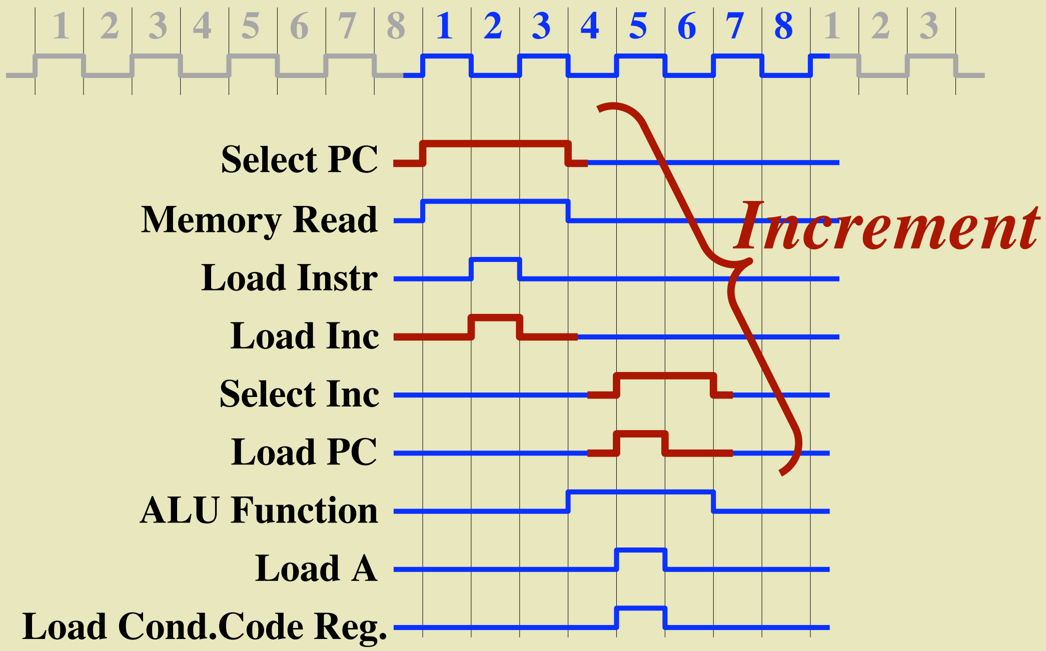
Instruction Timing - ALU Instruction



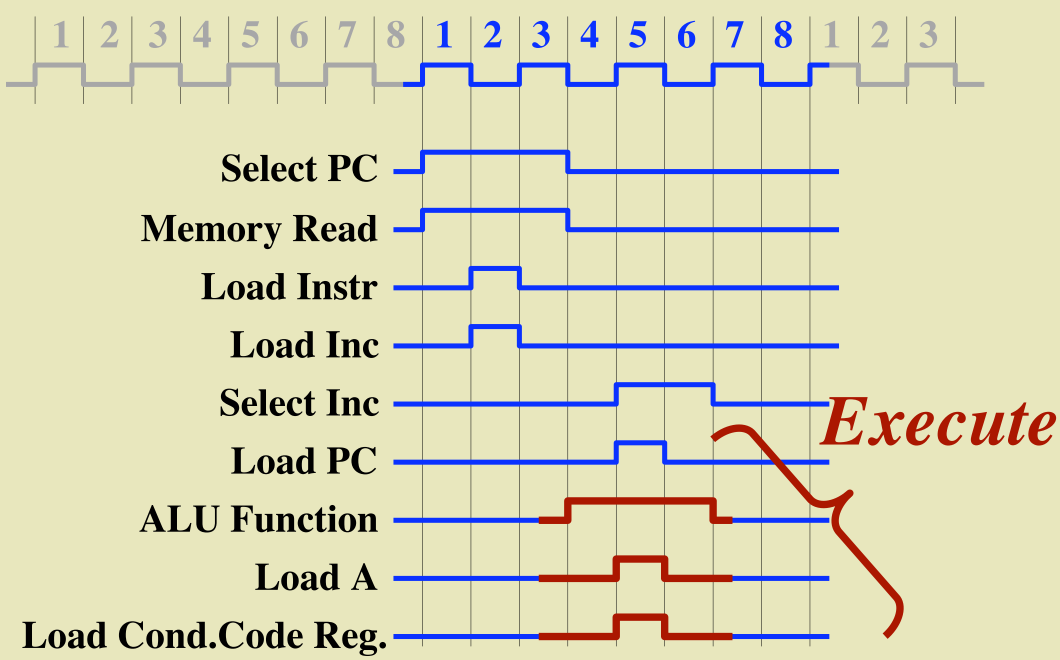
Instruction Timing - ALU Instruction



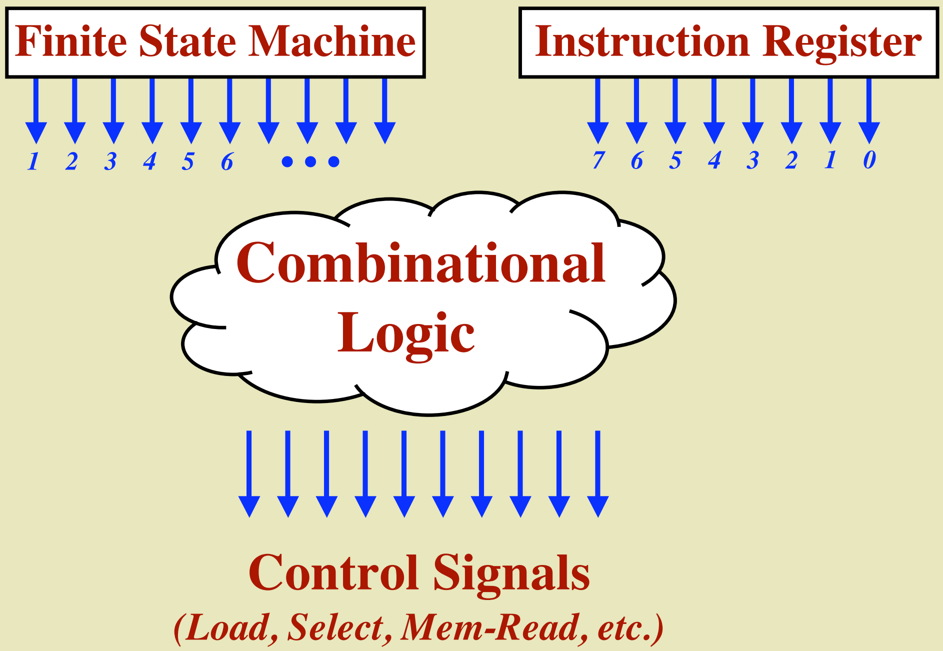
Instruction Timing - ALU Instruction



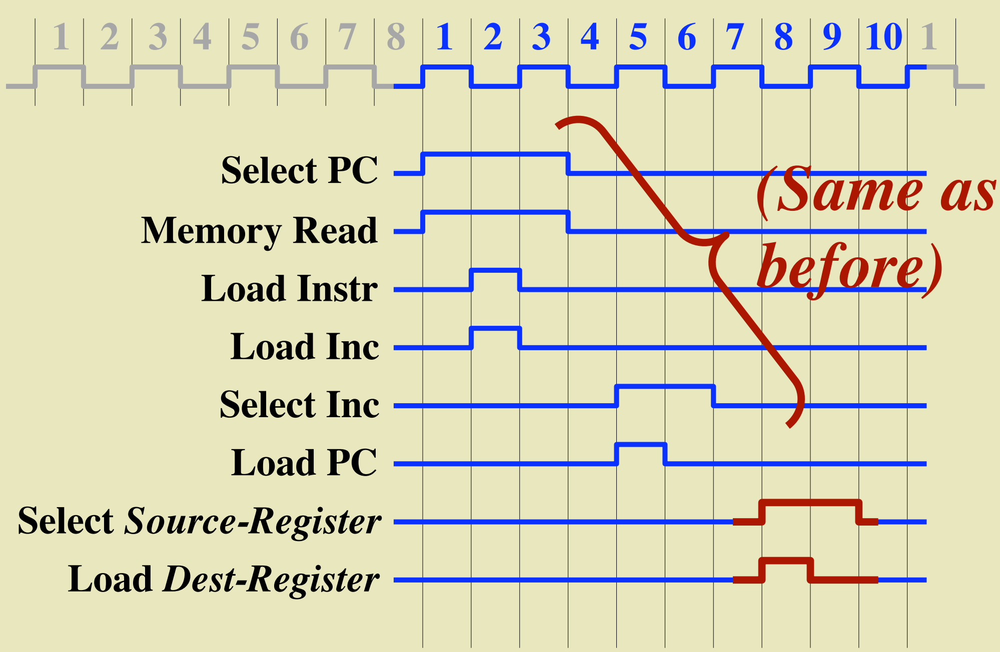
Instruction Timing - ALU Instruction



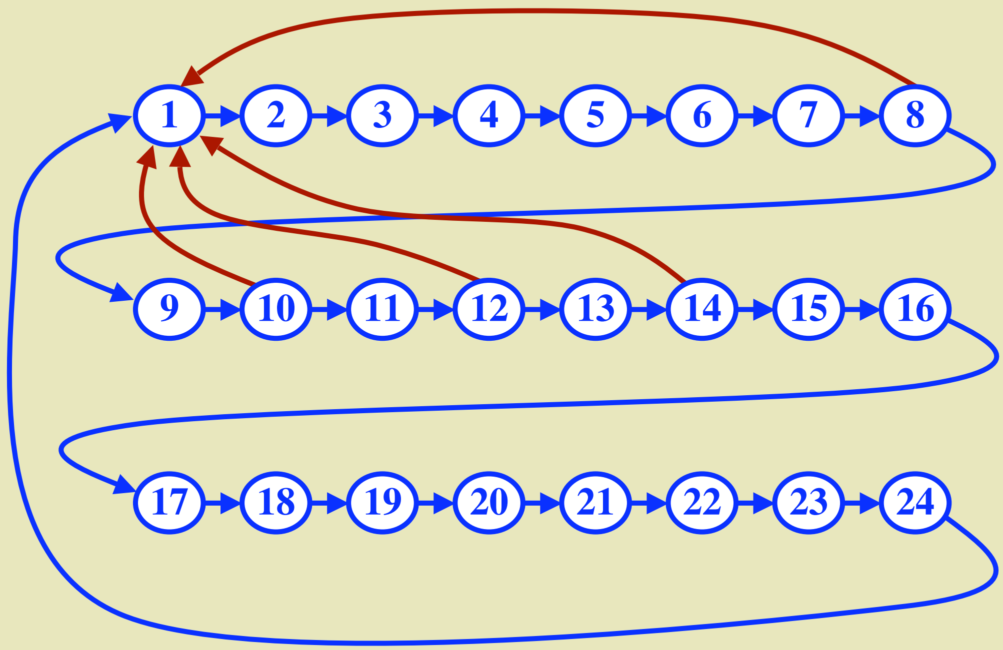
Instruction Decoding



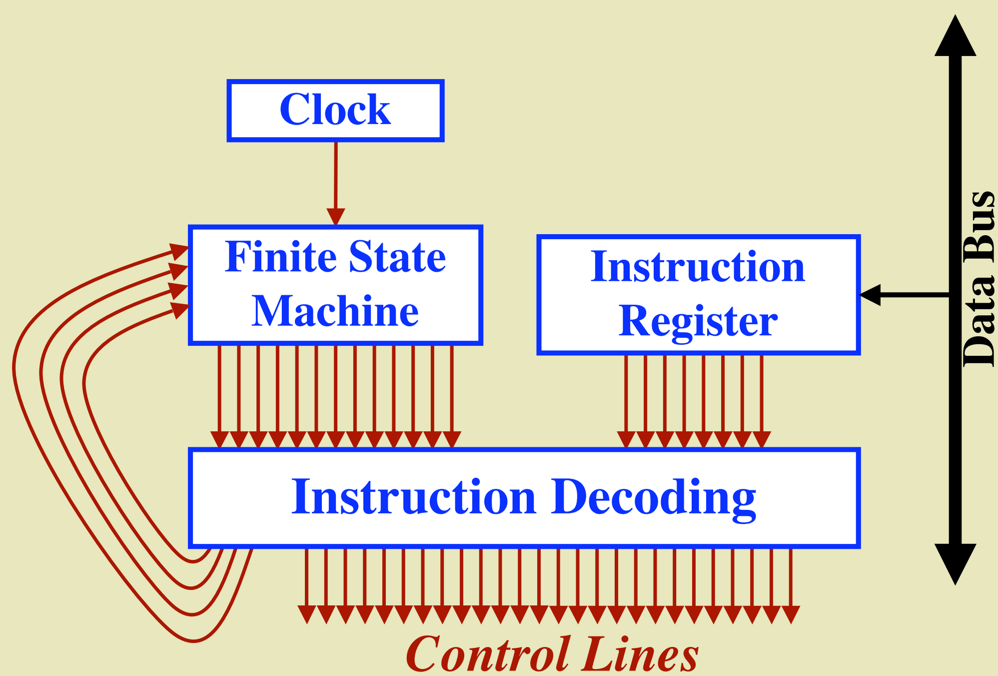
Instruction Timing - 16-bit Move



Finite State Machine



Instruction Decoding and Control



The Instruction Set

Move

0	0	d	d	d	s	s	s

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

ALU

1	0	0	0	r	f	f	f

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

Load Immediate

0	1	r	d	d	d	d	d

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

16-bit Increment

1	0	1	1	0	0	0	0

XY ← XY + 1

The Instruction Set

Load

1	0	0	1	0	0	r	r

`rr` = destination register (A, B, C, D)
reg ← [M]

Store

1	0	0	1	1	0	r	r

`rr` = source register (A, B, C, D)
[M] ← reg

Load 16-bit Immediate

1	1	0	0	0	0	0	0

v	v	v	v	v	v	v	v

v	v	v	v	v	v	v	v

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

Halt

1	0	1	0	1	1	1	0

The Instruction Set

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 ← 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

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

```
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
```