



# CPE 221: Computer Organization

O2 Number Systems and Digital Logic  
[rahul.bhadani@uah.edu](mailto:rahul.bhadani@uah.edu)

*Rahul Bhadani*

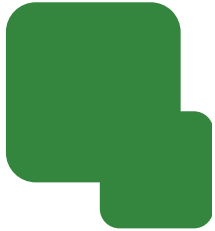


# Announcement

# Homework 01

Due: Jan 24, 2025

Points: 100



# Number Systems

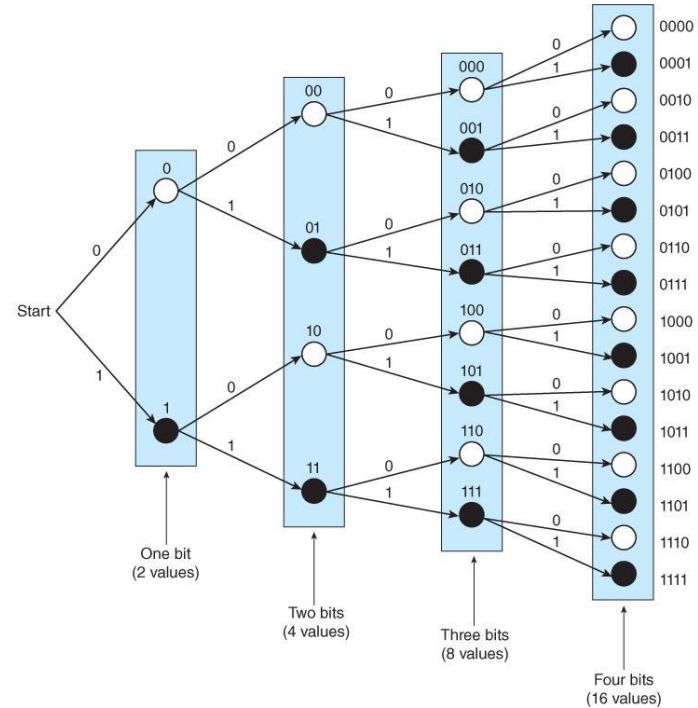
# Data in Modern Computing

- Everything is binary.
- But why?
  - Computers are based on transistors and switches that attain values **ON** and **OFF**. For simplicity, we translated **ON** to **1** and **OFF** to **0** or vice versa.

# Binary, bits and bytes

- The basic unit is the **bit** (binary digit).
- A **byte** is 8 **bits**.
- A word is **variable** length(platform to platform).
- A **64-bit** computer system has **64-bit** words.
- A **4-bit** computer system has **4-bit** words.
- The number of values represented by an **n-bit** word is  $2^n$ .

FIGURE 2.1 The binary tree



There are 10 kinds of  
people.

Those who understand  
binary and those who  
don't.



somee cards  
user card

# Data types

- A collection of bits can be
  - Signed integer                      temperature
  - Unsigned integer                      attending student count
  - Computer Instruction                      plan binary to be decoded
  - Floating Point Number                      partial number values
  - Image                      jpeg,
  - Audio                      mp3, wav,
  - Character                      ascii, Unicode, UTF
  - Pointer (Address)                      Machine's bytes counted in binary



# Bases in Number Systems

- Humans use base **ten**, computers use base **2**.
- Humans use **plus** or **minus** to indicate **sign** but we're lazy so we leave off the **plus sign** and assume unless indicated negative.
- Computers can do this as well, it's called **sign** and **magnitude** representation but it's rarely used.
- What is used is something called **signed 2's complement** representation
- Positive numbers are easier, so we'll start there
  - From base 6 to base 10:  $+1345_6 \square 353_{10}$

Use expanded form to convert a base 6 to base 10

$$(1345)_6 = 1 \times 6^3 + 3 \times 6^2 + 4 \times 6^1 + 5 \times 6^0$$

$$(1345)_6 = (1 \times 216) + (3 \times 36) + (4 \times 6) + (5 \times 1)$$

$$(1345)_6 = (353)_{10}$$

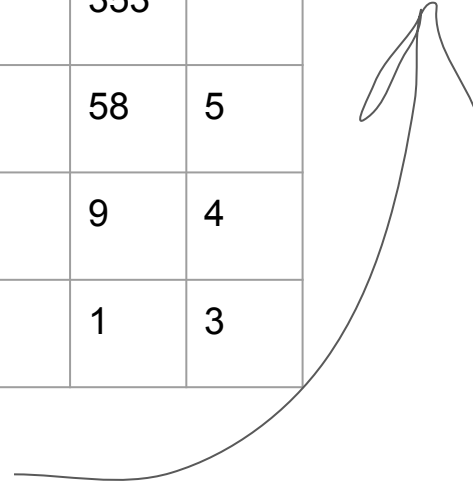
# Converting from Base 10 to Base 6

- $(353)_{10} \square (1345)_6$

6	353	
6	58	5

6	353	
6	58	5
	9	4

6	353	
6	58	5
	9	4
	1	3



# Binary to Decimal ( Positive Unsigned)

From base 2 to base 10:

$$(10111000)_2 =$$

$$1 \times 2^7 + 0 \times 2^6 + 1 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 0 \times 2^0$$

$$128 + 0 + 32 + 16 + 8 + 0 + 0 + 0$$

$$= 184$$

# Binary Addition

	1	1	
1	0	1	1
0	0	1	1

1	1	1	0
---	---	---	---

# Signed Binary

## System 1: Sign/Magnitude

N-bit binary number

Most significant bit as the sign, and remaining N-1 bits as the magnitude.

For example

$$(5)_{10} = (101)_2 \text{ or } (0101)_2$$

$$\text{then } (-5)_{10} = (1101)_2$$

However, with this scheme, we cannot perform addition operations discussed in the previous slide. Try it out!  $(5)_{10} + (-5)_{10} = ?$

# Signed Binary

## System 2: Two's complement

To get two's complement, write numbers in the binary, flip 0 to 1 and 1 to 0, and add 1.

Consider 00011100 which is 28 in the decimal in 8-bit binary.

Step 1: Flip the digits

11100011

Step 2: Add 1.

11100100 which is -28 in the decimal.

# 2's complement

To convert back 2's complement to original, basically know whose number it is negative of

Step 1: Flip 0 and 1.

Step 2: Add 1

$11100100 \Rightarrow 00011011 \Rightarrow 00011100$

# Arithmetic with Two's Complement

With two's complement, circuitry for addition and subtraction can be unified as we will see later.

Add 12 and 21

00001100

00010101

-----

00100001

Add 12 and -21

-21 is

00010101  $\Rightarrow$  11101010

$\Rightarrow$  11101011

00001100

11101011

-----

11110111 which is (-9) in two's complement



# Two's Complement

In 2's complement, the most significant bit tells you whether the number is positive or negative.

- Range  $-2^{(n-1)}$  to  $+2^{(n-1)} - 1$ . if  $n=8$  then range is -128 to +127. ( $n$  is the bit count)\*
- Two's complement facilitate same hardware

\* Bit count of an integer is typically 32 (4 bytes) in most systems



# Decimal and Fractional Number Representation

# Fixed-points numbers

26.5

26 = high word

.5 = low word

$$2 * 10^1 + 6 * 10^0 + 5 * 10^{-1} = 26.5$$

**11010.1<sub>2</sub>**

$$= 1 * 2^4 + 1 * 2^3 + 0 * 2^2 + 1 * 2^1 + 0 * 2^0 + 1 * 2^{-1}$$

$$= 16 + 8 + 2 + 0.5$$

$$= 26.5$$

$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$	$2^{-1}$	$2^{-2}$	$2^{-3}$
...	1	1	0	1	0	1	0	...

# Fixed Point representation of negative number

Consider the number -2.5, fixed<w,b> width = 4 bit, binary point = 1 bit (assume the binary point is at position 1). First, represent 2.5 in binary, then find its 2's complement and you will get the binary fixed-point representation of -2.5.

$$2.5_{10} = 0101_2$$

$$-2.5_{10} = 1010_2 + 1 \text{ (1's complement + 1 = 2's complement)}$$

$$-2.5_{10} = 1011_2$$

# Floating-point Number Representation

IEEE 754 Floating-point standard (1985)

<https://standards.ieee.org/ieee/754/6210/>

<https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8766229>

Consider

$$\underbrace{+}_{\text{sign}} \underbrace{4.1}_{\text{mantissa}} \times \underbrace{2^3}_{\text{exponent}}$$

# Rules for IEEE 754 Floating-point standard

1. 32 bits = 1 bit for sign + 8 bit for exponents + 23 bits for mantissa.
2. 64 bits = 1 bit for sign + 11 bit for exponents + 52 bits for mantissa.
3. In binary representation, the first bit of the mantissa is always 1 (excluded from 23 (or 52) bits).
4. Floating point representation uses bias. That is actual floating point + a constant bias. For a 32 bit floating point, the bias is 127. Hence the exponent of 7 is written as  $7 + 127 = 134 = 10000110_2$ .
5. Special case

Number	Sign	Exponent	Fraction
0	X	00000000	00000000000000000000000
$\infty$	0	11111111	00000000000000000000000
$-\infty$	1	11111111	00000000000000000000000
NaN	X	11111111	Non-zero

# Example

0.085 in Single precision (32 bit) binary format

1. We write 0.085 in base-2 scientific notation, that is we must factor it into a number in the range  $(1 \leq n < 2)$  and a power of 2.

$$\begin{array}{lcl} 0.085 & & 0.085 = (-1)^0(1 + \text{fraction}) \times 2^{\text{power}}, \quad \text{or, equivalently:} \\ / & 2^{\text{power}} & = 1 + \text{fraction} \end{array}$$

$$0.085 / 2^{-1} = 0.17$$

$$0.085 / 2^{-2} = 0.34$$

$$0.085 / 2^{-3} = 0.68$$

$$0.085 / 2^{-4} = 1.36$$

# Example ...

Therefore,  $0.085 = 1.36 \times 2^{-4}$

Now, we find the exponent  
exponent =  
 $-4 + 127 = 123 = 01111011_2$

Then, we write the fraction in  
binary form using successive  
multiplications by 2

But we only have 23 bits.

0.01011100001010001111011

```
0.36 x 2 = 0.72
0.72 x 2 = 1.44
0.44 x 2 = 0.88
0.88 x 2 = 1.76
0.76 x 2 = 1.52
0.52 x 2 = 1.04
0.04 x 2 = 0.08
0.08 x 2 = 0.16
0.16 x 2 = 0.32
0.32 x 2 = 0.64
0.64 x 2 = 1.28
0.28 x 2 = 0.56
0.56 x 2 = 1.12
0.12 x 2 = 0.24
0.24 x 2 = 0.48
0.48 x 2 = 0.96
0.96 x 2 = 1.92
0.92 x 2 = 1.84
0.84 x 2 = 1.68
0.68 x 2 = 1.36
0.36 x 2 = ...
```

Once this process terminates or starts repeating,  
we read the unit's digits from top to bottom  
to reveal the binary form for 0.36:

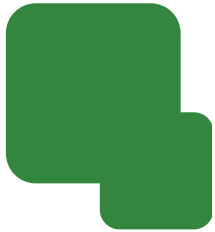
0.010111000010100011110111000...

(at this point the list starts repeating)

So 0.085 in IEEE 754 format is:

0 01111011 01011100001010001111011





# Data Formats

# Some Common Data Representation

Types of Data	Standard(s)
Alphanumeric	Unicode, ASCII
Image (bitmap)	GIF, TIFF, PNG, JPEG
Image (object)	Postscript, SVG
Outline graphics and Fonts	Postscript, TrueType
Page Description	PDF, HTML, XML
Video	MPEG-4, WMV, MP4

# Alphanumeric Codes

1. ASCII (American Standard Code for Information Interchange)
2. Unicode
3. EBCDIC (Extended Binary Coded Decimal Interchange Code): Defunct



















































ASCII and EBCDIC both can be stored using 1 byte. (How many bits in one byte?) Hence, ASCII is limited in what it can represent.

# ASCII Table

<https://www.asciitable.com/>

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	<b>NUL</b> (null)	32	20	040	&#32;	Space	64	40	100	&#64;	@	96	60	140	&#96;	`
1	1	001	<b>SOH</b> (start of heading)	33	21	041	&#33;	!	65	41	101	&#65;	A	97	61	141	&#97;	a
2	2	002	<b>STX</b> (start of text)	34	22	042	&#34;	"	66	42	102	&#66;	B	98	62	142	&#98;	b
3	3	003	<b>ETX</b> (end of text)	35	23	043	&#35;	#	67	43	103	&#67;	C	99	63	143	&#99;	c
4	4	004	<b>EOT</b> (end of transmission)	36	24	044	&#36;	\$	68	44	104	&#68;	D	100	64	144	&#100;	d
5	5	005	<b>ENQ</b> (enquiry)	37	25	045	&#37;	%	69	45	105	&#69;	E	101	65	145	&#101;	e
6	6	006	<b>ACK</b> (acknowledge)	38	26	046	&#38;	&	70	46	106	&#70;	F	102	66	146	&#102;	f
7	7	007	<b>BEL</b> (bell)	39	27	047	&#39;	'	71	47	107	&#71;	G	103	67	147	&#103;	g
8	8	010	<b>BS</b> (backspace)	40	28	050	&#40;	(	72	48	110	&#72;	H	104	68	150	&#104;	h
9	9	011	<b>TAB</b> (horizontal tab)	41	29	051	&#41;	)	73	49	111	&#73;	I	105	69	151	&#105;	i
10	A	012	<b>LF</b> (NL line feed, new line)	42	2A	052	&#42;	*	74	4A	112	&#74;	J	106	6A	152	&#106;	j
11	B	013	<b>VT</b> (vertical tab)	43	2B	053	&#43;	+	75	4B	113	&#75;	K	107	6B	153	&#107;	k
12	C	014	<b>FF</b> (NP form feed, new page)	44	2C	054	&#44;	,	76	4C	114	&#76;	L	108	6C	154	&#108;	l
13	D	015	<b>CR</b> (carriage return)	45	2D	055	&#45;	-	77	4D	115	&#77;	M	109	6D	155	&#109;	m
14	E	016	<b>SO</b> (shift out)	46	2E	056	&#46;	.	78	4E	116	&#78;	N	110	6E	156	&#110;	n
15	F	017	<b>SI</b> (shift in)	47	2F	057	&#47;	/	79	4F	117	&#79;	O	111	6F	157	&#111;	o
16	10	020	<b>DLE</b> (data link escape)	48	30	060	&#48;	0	80	50	120	&#80;	P	112	70	160	&#112;	p
17	11	021	<b>DC1</b> (device control 1)	49	31	061	&#49;	1	81	51	121	&#81;	Q	113	71	161	&#113;	q
18	12	022	<b>DC2</b> (device control 2)	50	32	062	&#50;	2	82	52	122	&#82;	R	114	72	162	&#114;	r
19	13	023	<b>DC3</b> (device control 3)	51	33	063	&#51;	3	83	53	123	&#83;	S	115	73	163	&#115;	s
20	14	024	<b>DC4</b> (device control 4)	52	34	064	&#52;	4	84	54	124	&#84;	T	116	74	164	&#116;	t
21	15	025	<b>NAK</b> (negative acknowledge)	53	35	065	&#53;	5	85	55	125	&#85;	U	117	75	165	&#117;	u
22	16	026	<b>SYN</b> (synchronous idle)	54	36	066	&#54;	6	86	56	126	&#86;	V	118	76	166	&#118;	v
23	17	027	<b>ETB</b> (end of trans. block)	55	37	067	&#55;	7	87	57	127	&#87;	W	119	77	167	&#119;	w
24	18	030	<b>CAN</b> (cancel)	56	38	070	&#56;	8	88	58	130	&#88;	X	120	78	170	&#120;	x
25	19	031	<b>EM</b> (end of medium)	57	39	071	&#57;	9	89	59	131	&#89;	Y	121	79	171	&#121;	y
26	1A	032	<b>SUB</b> (substitute)	58	3A	072	&#58;	:	90	5A	132	&#90;	Z	122	7A	172	&#122;	z
27	1B	033	<b>ESC</b> (escape)	59	3B	073	&#59;	;	91	5B	133	&#91;	[	123	7B	173	&#123;	{
28	1C	034	<b>FS</b> (file separator)	60	3C	074	&#60;	<	92	5C	134	&#92;	\	124	7C	174	&#124;	
29	1D	035	<b>GS</b> (group separator)	61	3D	075	&#61;	=	93	5D	135	&#93;	]	125	7D	175	&#125;	}
30	1E	036	<b>RS</b> (record separator)	62	3E	076	&#62;	>	94	5E	136	&#94;	^	126	7E	176	&#126;	~
31	1F	037	<b>US</b> (unit separator)	63	3F	077	&#63;	?	95	5F	137	&#95;	_	127	7F	177	&#127;	DEL

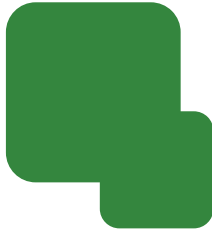
# Unicode

 U+0F3C	 U+0FD1	 U+03B5	 U+3078	 U+0D05	 U+261E	 U+10EB	 U+104E	 U+0E21	 U+2B1B	 U+B208
 U+30FD	 U+03C6	 U+FF10	 U+21E7	 U+D1E1	 U+2021	 U+1F499	 U+134C	 U+03B3	 U+0B27	 U+02CA
 U+11BA	 U+0296	 U+0B67	<p>Everyone in the world should be able to use their own language on phones and computers.</p> <p><a href="#">LEARN MORE ABOUT UNICODE</a></p>							
 U+2661	 U+266A	 U+03C2								
 U+1F923	 U+10F9	 U+30C7	 U+2015	 U+2014	 U+266F	 U+06B1	 U+8DD1	 U+10DB	 U+262E	 U+01D0
 U+270D	 U+2044	 U+FF61	 U+06F6	 U+0665	 U+141B	 U+060F	 U+3008	 U+0964	 U+10E3	 U+056E

<https://home.unicode.org>

# Unicode

- Most common Unicode is UTF(Unicode Transformation Format)-16: uses 16 bits (how many characters it can represent?)
- Unicode is multilingual: has representation for other languages, and emojis.
- Each UTF-16 alphanumeric character is stored using two bytes



# Computer Logic

# Boolean Algebra: A Quick Cheatsheet (otherwise see in your EE 202)

1. Law of Commutation  $A + B = B + A$   
 $A \cdot B = B \cdot A$

2. Law of association  $(A \cdot B) \cdot C = A \cdot (B \cdot C)$   
 $(A + B) + C = A + (B + C)$

3. Law of Distribution  $A \cdot B + A \cdot C = A \cdot (B + C)$   
 $(A + B)(A + C) = A + BC$

4. Law of absorption  $A \cdot (A + B) = A$   
 $A + AB = A$   
 $AB + \bar{B} = A + \bar{B}$   
 $A\bar{B} + B = A + B$

5. Consensus theorem  $AB + \bar{A}C + BC = AB + \bar{A}C$   
 $(A + B)(\bar{A} + C)(B + C) = (A + B)(\bar{A} + C)$

6. Transposition theorem  $AB + \bar{A}C = (A + C)(\bar{A} + B)$

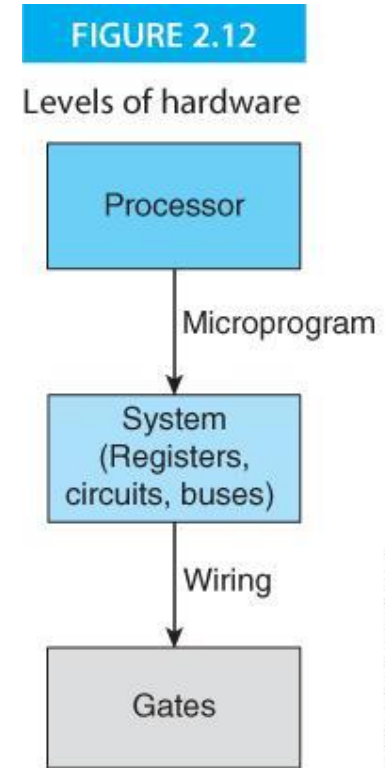
7. De Morgan's theorem-I  $\overline{A + B} = \bar{A} \cdot \bar{B}$

8. De Morgan's theorem-II  $\overline{A \cdot B} = \bar{A} + \bar{B}$



# Computer Logic using Logic Gates (1)

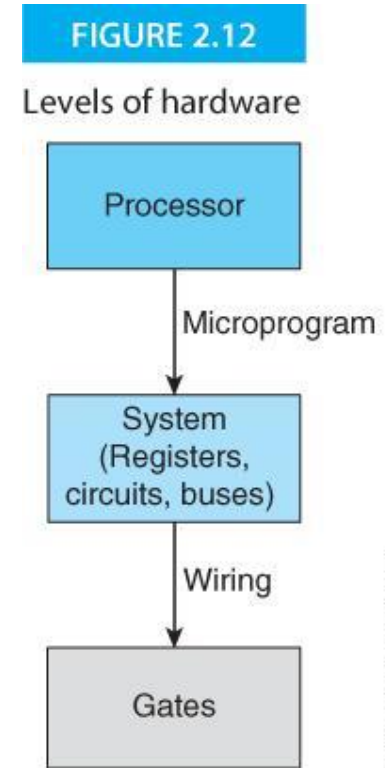
- Computers are constructed from two basic circuit elements
  - Combinational** logic elements (*gates*)
  - Sequential** logic elements (*flip-flops*)
- A combinational logic element is a circuit whose output depends only on its **current inputs**.
- A sequential element is a circuit whose output depends on **present inputs** and **past inputs** (captured as the **state** of the element or a form of memory).
- Sequential elements themselves can be made from simple combinational logic elements.



© Cengage Learning 2014

# Computer Logic using Logic Gates (2)

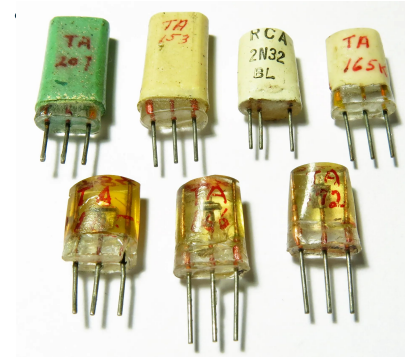
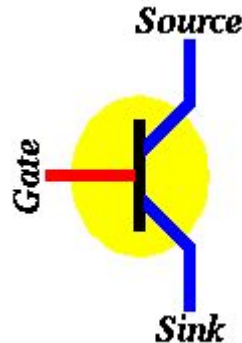
- Any combinational circuit can be made from AND , OR , and NOT gates.
- This set of gates is said to be **functionally complete**.
- Because **flip-flops** can be constructed from **gates**, all computers can be constructed from **gates** alone.
- Moreover, because the **NAND** or **NOR** gate, can be used to synthesize AND, OR, and NOT gates, any computer can be constructed from nothing more than a large number of **NAND** or **NOR** gates.



© Cengage Learning 2014

# Why Logic Gates are called Logic Gates

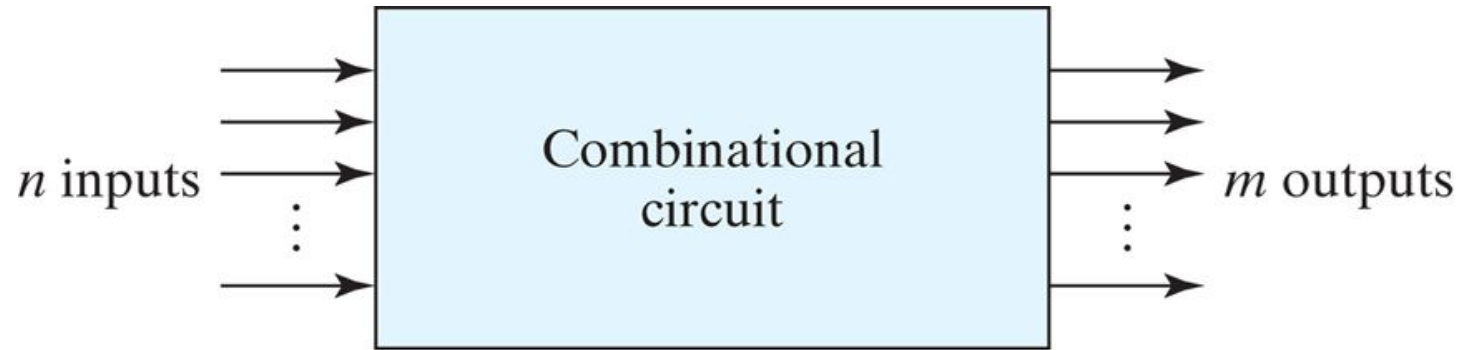
The word “logic” implies decision making. The word “gate” is used in electronics to describe a switch, in the early days of logic a switch comprised discrete transistors. The gate switches the “output” of a “decision” based on the type of gate and the information provided to the gates “input”.



HOW THE FIRST TRANSISTOR WORKED

# Computer Logic Diagram

Combinational logic – This is an abstract level view of a computer architecture. EE's will have classes that study the layers underneath, including the transistor.



Copyright ©2013 Pearson Education, publishing as Prentice Hall

# Transistor

- “Trans Resistor” invented in 1947 in Bell Laboratories, replaced the vacuum tube triode
- Probably the single most important invention to computer and electrical science.
- Works by manipulating resistances
- Optional videos in canvas



# AND Gate

TABLE 2.8

Truth Table for the AND Gate

B	A	$C = A \cdot B$	C	B	A	$D = A \cdot B \cdot C$
0	0	0	0	0	0	0
0	1	0	0	0	1	0
1	0	0	0	1	0	0
1	1	1	0	1	1	0
			1	0	0	0
			1	0	1	0
			1	1	0	0
			1	1	1	1

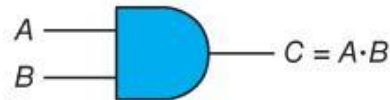
(a) Two-input AND gate

(b) Three-input AND gate

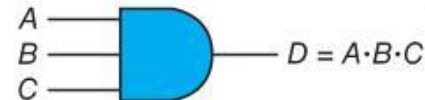
© Cengage Learning 2014

FIGURE 2.14

The symbol for an AND gate



(a) Two-input AND gate



(b) Three-input AND gate

© Cengage Learning 2014

**TABLE 2.9** Truth Table for the OR Gate

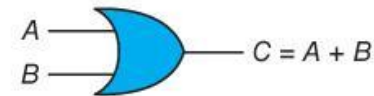
$B$	$A$	$C = A + B$	$C$	$B$	$A$	$D = A + B + C$
0	0	0	0	0	0	0
0	1	1	0	0	1	1
1	0	1	0	1	0	1
1	1	1	0	1	1	1
			1	0	0	1
			1	0	1	1
			1	1	0	1
			1	1	1	1

(a) Two-input OR gate

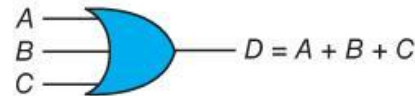
(b) Three-input OR gate

© Cengage Learning 2014

**FIGURE 2.15** The symbol for an OR gate



(a) Two-input OR gate



(b) Three-input OR gate

© Cengage Learning 2014

# Comparing AND and OR Gates

**TABLE 2.10**

Truth Table for AND and OR Gates with Both Constant and Variable Inputs

AND		OR	
Constant	Variable	Constant	Variable
$0 \cdot 0 = 0$	$A \cdot 0 = 0$	$0 + 0 = 0$	$A + 0 = A$
$0 \cdot 1 = 0$	$A \cdot 1 = A$	$0 + 1 = 1$	$A + 1 = 1$
$1 \cdot 0 = 0$	$A \cdot \bar{A} = 0$	$1 + 0 = 1$	$A + \bar{A} = 1$
$1 \cdot 1 = 1$	$A \cdot A = A$	$1 + 1 = 1$	$A + A = A$

© Cengage Learning 2014



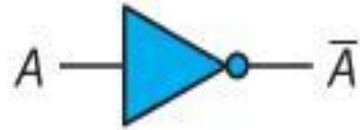
# Points to Note

- Setting one input of an AND to constant 0 PERMANENTLY disables that gate (until that input is allowed to be 1 again), setting its output to constant 0
- Setting one input of an OR gate to constant 1 also permanently disables it, but its output is constant 1

# Inverter or NOT Gate

**FIGURE 2.16**

The symbol and truth table for an inverter



(a) Symbol for inverter

$A$	$\bar{A}$
0	1
1	0

(b) Truth table of inverter

© Cengage Learning 2014

# Question

Which logic statement(s) below is true?

1.  $B \text{ OR } B' = 1$
2.  $B \text{ AND } B' = 0$
3.  $A \text{ OR } 1 = 0$

# NOR, NAND, Exclusive OR (XOR)

**TABLE 2.11** Truth Table for the NOR Gate, NAND Gate, and Exclusive OR Gates

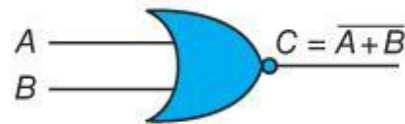
A	C	$C = \overline{A + B}$	..	A	B	$C = \overline{A \cdot B}$	..	A	B	$C = A \oplus B$
0	0	1		0	0	1		0	0	0
0	1	0		0	1	1		0	1	1
1	0	0		1	0	1		1	0	1
1	1	0		1	1	0		1	1	0

(a) The NOR gate

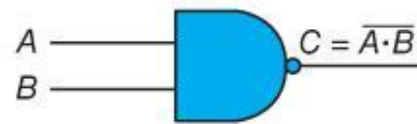
(b) The NAND gate

(c) The XOR gate

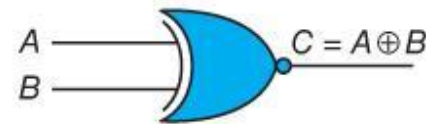
**FIGURE 2.19** Three derived gates



(a) NOR gate



(b) NAND gate



(c) Exclusive OR gate

© Cengage Learning 2014

© Cengage Learning 2014

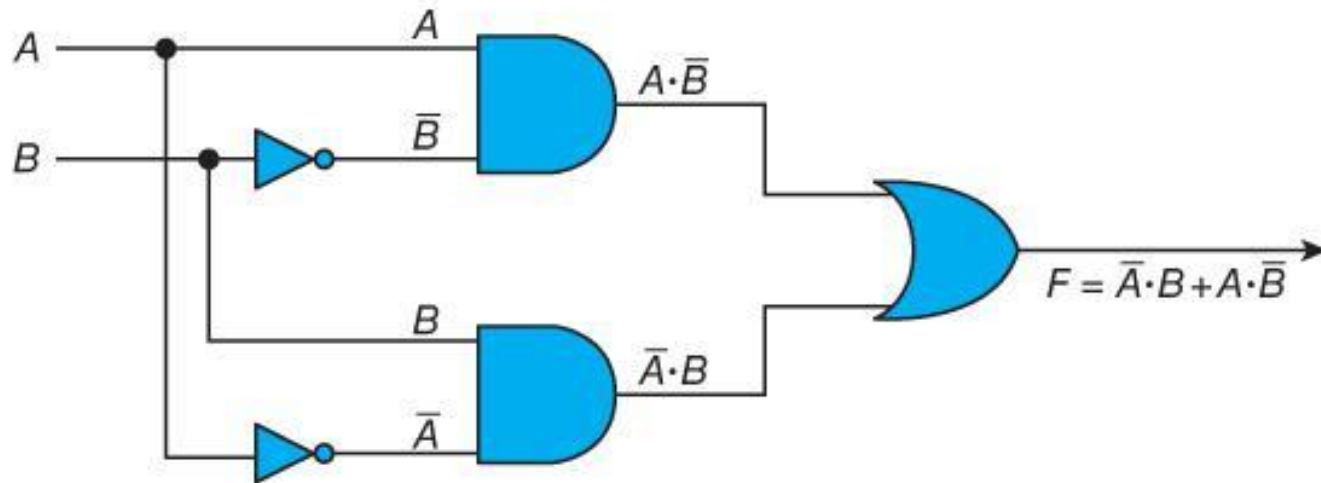
# Question

What is the output of A **NAND** B when  $A = 1$  and  $B = 1$ ? If  $A = 1$  and  $B = 0$ ?

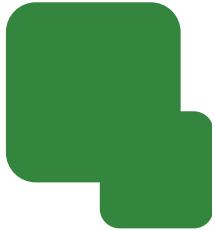
# Making XOR out of AND, OR, and NOT

**FIGURE 2.20**

Constructing an XOR circuit from AND, OR, and NOT gates

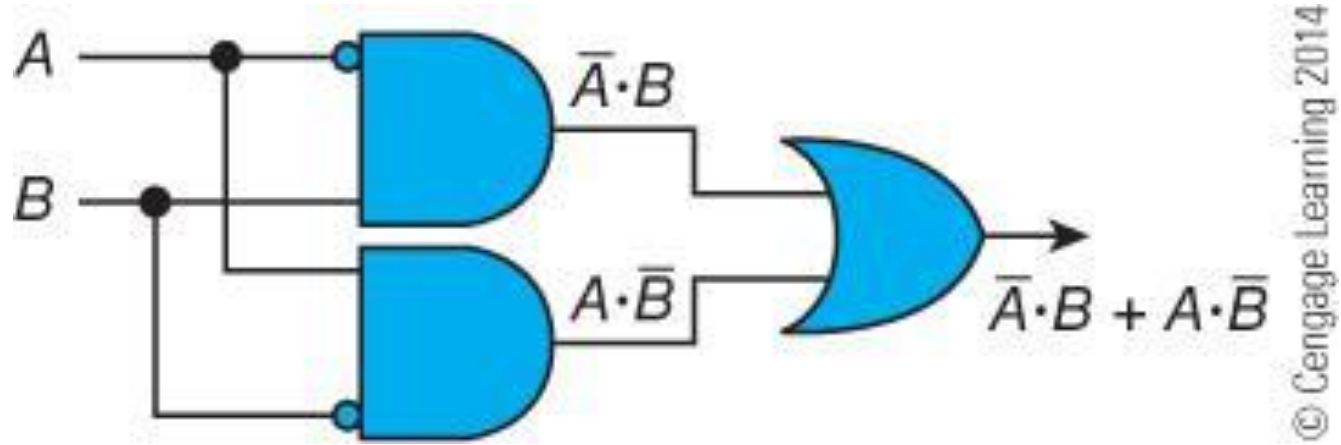


© Cengage Learning 2014



# Combination Circuits

# Inversion Bubbles





# The Half Adder

TABLE 2.13

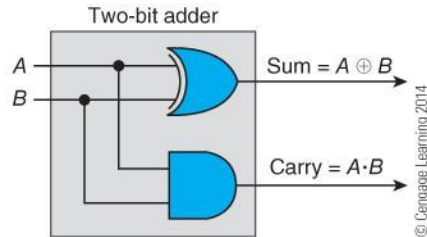
Truth Table of a Half Adder

A	B	Sum	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

© Cengage Learning 2014

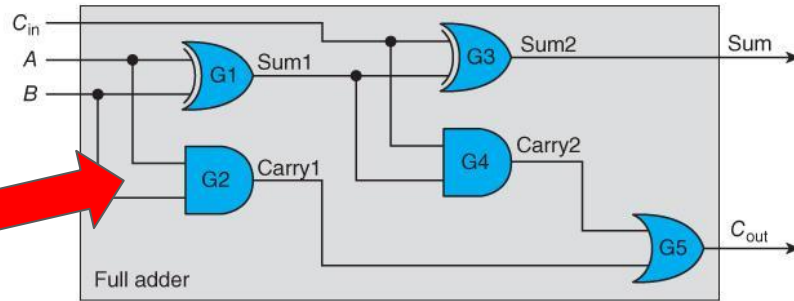
FIGURE 2.22

The two-bit adder (the half adder)



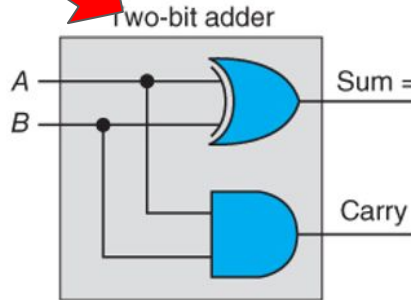
# The Full Adder

FIGURE 2.23 The full adder



© Cengage Learning 2014

Half adder here

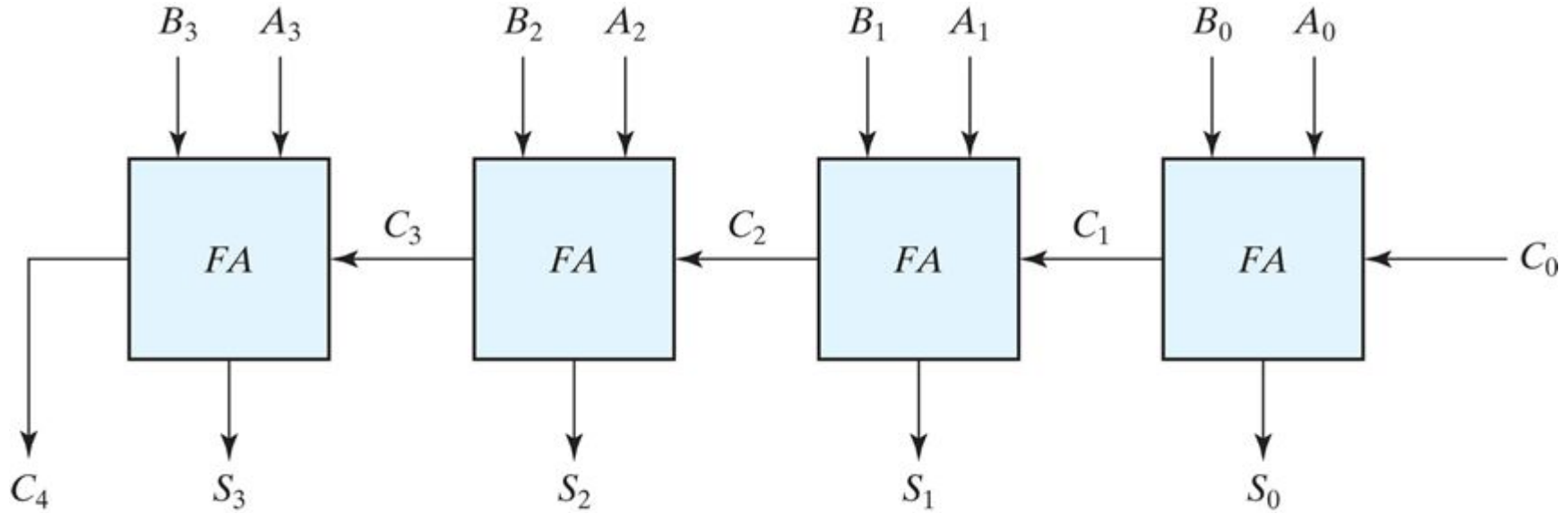


$C_{in}$	$A$	$B$	Sum	$C_{out}$
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

# Question

1. A full adder has inputs A, B, and  $C_{in}$  with the values of (1,1,0). What are the output values for  $C_{out}$  and S?
2. A full adder has inputs A, B, and  $C_{in}$  with the values of (1,1,1). What are the output values for  $C_{out}$  and S?

# Ripple Carry Adder

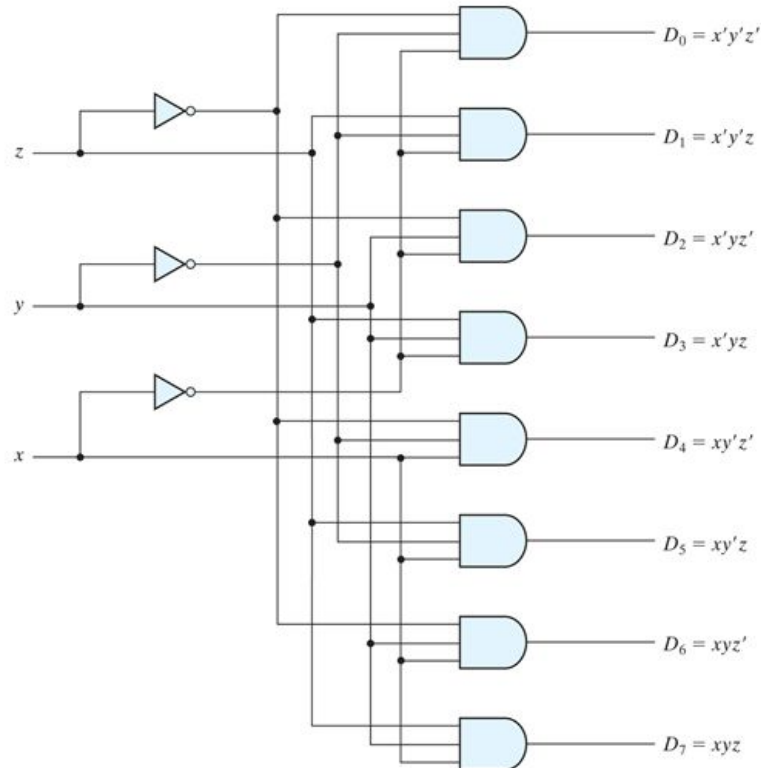


Copyright ©2013 Pearson Education, publishing as Prentice Hall

# Decoding an Instruction

## 3-to-8 Active High Decoder

Should be easier to understand using a specific example of a 3-to-8 decoder.



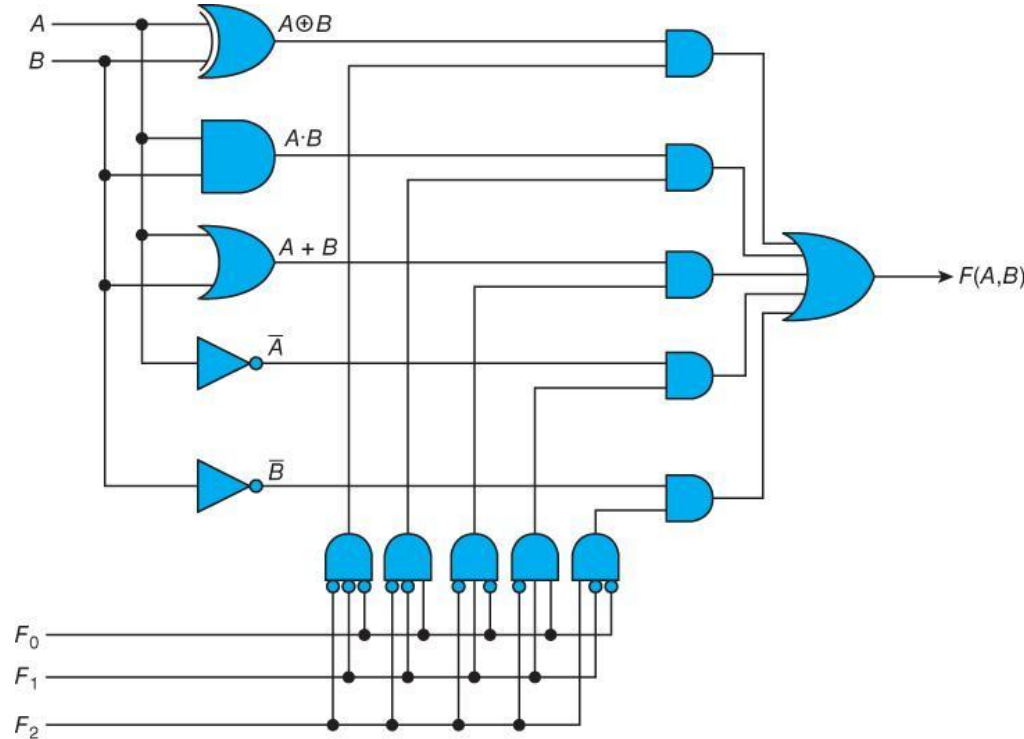
# Decoder Truth Table

**Table 4.6**  
*Truth Table of a Three-to-Eight-Line Decoder*

Inputs			Outputs							
<i>x</i>	<i>y</i>	<i>z</i>	<i>D</i> <sub>0</sub>	<i>D</i> <sub>1</sub>	<i>D</i> <sub>2</sub>	<i>D</i> <sub>3</sub>	<i>D</i> <sub>4</sub>	<i>D</i> <sub>5</sub>	<i>D</i> <sub>6</sub>	<i>D</i> <sub>7</sub>
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

Copyright ©2012 Pearson Education, publishing as Prentice Hall

# One Bit of an ALU (Arithmetic Logical Unit)

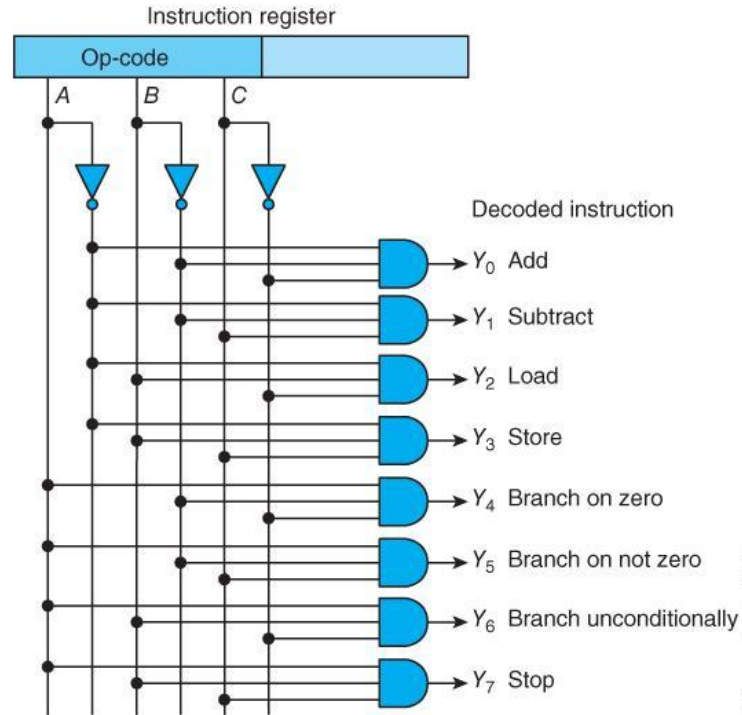


© Cengage Learning 2014

# Decoding an Instruction

FIGURE 2.28

Application of a decoder



© Cengage Learning 2014



# Multiplexer

**Multiplexer** - Combinational logic which takes one of its  $2^n$  inputs and directs it to its only output under control of its  $n$  control or select lines.

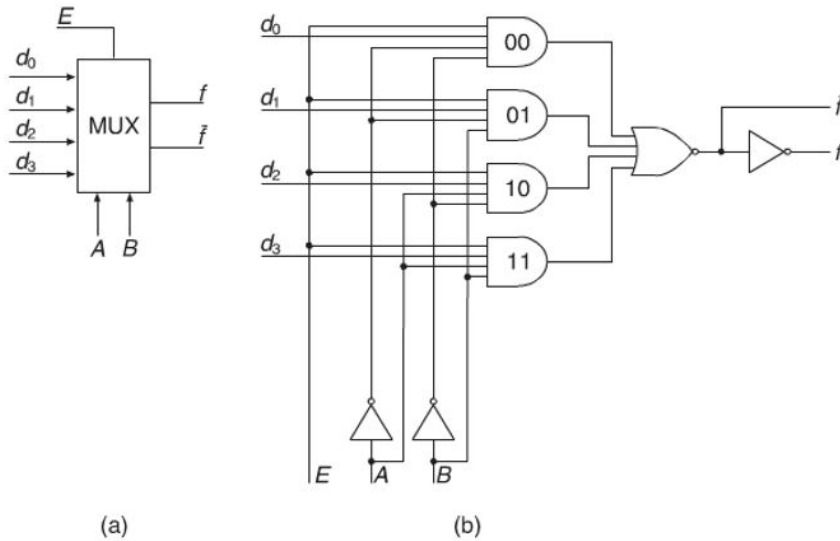
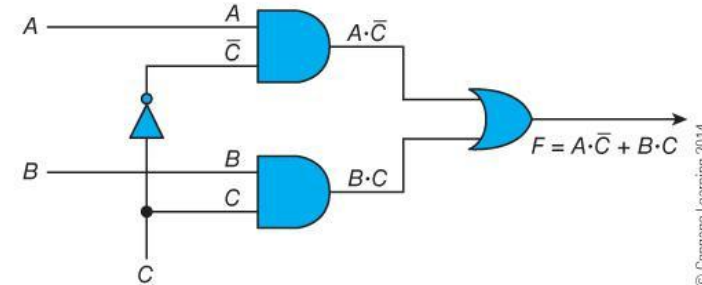


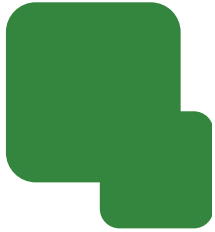
Figure 5.1 (a) Block diagram of a 4-input multiplexer and (b) its gate implementation

FIGURE 2.30

Alternative representation of the two-input multiplexer



© Cengage Learning 2014



# Circuit Simplification

# Sum of Products and Product of Sums

**Sum-of-products (SOP):** A SOP is the logical OR operation of multiple product terms. Each product term is the logical AND operation of binary literals. For example,

$$XY + X\bar{Y} + YZ$$

is a SOP expression.

**Product-of-sum (POS):** A POS is the logical AND operation of multiple ORrd terms. Each sum term is the logical OR operation of binary literals. For example,

$$(X + \bar{Y})(X + Y + Z)(\bar{X} + Y + \bar{Z})$$

is a POS expression.

# Minterm and Maxterm

**Minterm:** A minterm is a special case product (AND) term. A minterm is a product term that contains all of the input variables (each literal no more than once) that makes up a Boolean expression. Example:

XYZ for a three input logic circuit. Denoted by small  $m$ .

**Maxterm:** A maxterm is a special case sum (OR) term. A maxterm is a sum term that contains all of the input variables (each literal no more than once) that make up a Boolean expression. Example:

$X + Y + Z$  for a three input logic circuit. Denoted by capital  $M$ .

$$\begin{aligned}\bar{A} + \bar{B} &= \bar{A}(B + \bar{B}) + (A + \bar{A})\bar{B} \\ &= \bar{A}B + \bar{A}\bar{B} + A\bar{B} + \bar{A}\bar{B} \\ &= \bar{A}B + \bar{A}\bar{B} + A\bar{B} \\ &= 01 + 00 + 10 \\ &= m_1 + m_0 + m_2 \\ &= \sum m(0, 1, 2)\end{aligned}$$

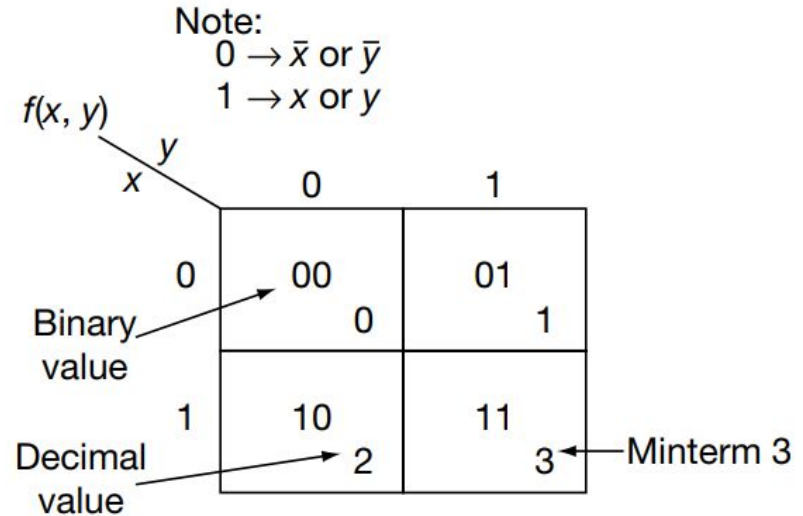
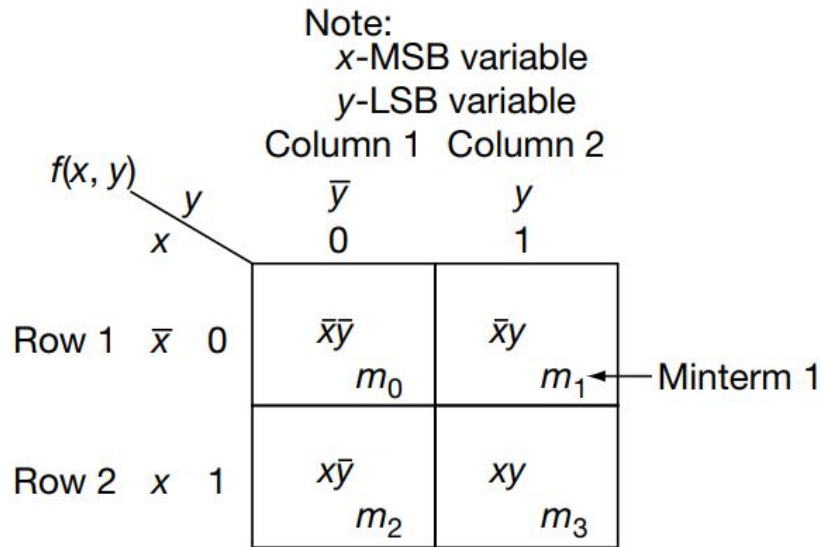
Sum of products

$$\bar{A} + \bar{B} = \prod M(3)$$

Product of Sum

# Karnaugh Map for Circuit Simplification

## Two-variable K-map using minterms



# Two-variable K-map using maxterms

Note:  
x-MSB variable  
y-LSB variable

		Column 1    Column 2	
		y 0	$\bar{y}$ 1
Row 1	x   0	$x + y$ $M_0$	$x + \bar{y}$ $M_1$ ← Maxterm 1
	$\bar{x}$ 1	$\bar{x} + y$ $M_2$	$\bar{x} + \bar{y}$ $M_3$

Note:  
 $1 \rightarrow \bar{x} \text{ or } \bar{y}$   
 $0 \rightarrow x \text{ or } y$

$f(x, y)$

		0	1
0	00 0	01 1	
1	10 2	11 3 ← Maxterm 3	

Binary value

Decimal value

# Three-variable K-map

			Column 1	Column 2	Column 3	Column 4	Note:
$f(x, y, z)$			$\bar{y}\bar{z}$	$\bar{y}z$	$yz$	$y\bar{z}$	$z \rightarrow \text{LSB variable}$
			00	01	11	10	$x \rightarrow \text{MSB variable}$
$x$	0	$\bar{x}$	$\bar{x}\bar{y}\bar{z}$ $m_0$	$\bar{x}\bar{y}z$ $m_1$	$\bar{x}yz$ $m_3$	$\bar{x}y\bar{z}$ $m_2$	Note: $0 \rightarrow \bar{x} \text{ or } \bar{y} \text{ or } \bar{z}$ $1 \rightarrow x \text{ or } y \text{ or } z$
	1	$x$	$x\bar{y}\bar{z}$ $m_4$	$x\bar{y}z$ $m_5$	$xyzz$ $m_7$	$xy\bar{z}$ $m_6$	← Minterm 6

			Column 1	Column 2	Column 3	Column 4	Note:
$f(x, y, z)$			$y + z$	$y + \bar{z}$	$\bar{y} + \bar{z}$	$\bar{y} + z$	$z \rightarrow \text{LSB variable}$
			00	01	11	10	$x \rightarrow \text{MSB variable}$
$x$	0	$x$	$x + y + z$ $M_0$	$x + y + \bar{z}$ $M_1$	$x + \bar{y} + \bar{z}$ $M_3$	$x + \bar{y} + z$ $M_2$	Note: $1 \rightarrow \bar{x} \text{ or } \bar{y} \text{ or } \bar{z}$ $0 \rightarrow x \text{ or } y \text{ or } z$
	1	$\bar{x}$	$\bar{x} + y + z$ $M_4$	$\bar{x} + y + \bar{z}$ $M_5$	$\bar{x} + \bar{y} + \bar{z}$ $M_7$	$\bar{x} + \bar{y} + z$ $M_6$	← Maxterm 6

# Four Variable K-map

$f(w, x, y, z)$		Column 1	Column 2	Column 3	Column 4	Note:
		$\bar{y}\bar{z}$ 00	$\bar{y}z$ 01	$yz$ 11	$y\bar{z}$ 10	
Row 1	$\bar{w}\bar{x}$ 00	$\bar{w}\bar{x}\bar{y}\bar{z}$ $m_0$	$\bar{w}\bar{x}\bar{y}z$ $m_1$	$\bar{w}\bar{x}yz$ $m_3$	$\bar{w}\bar{x}y\bar{z}$ $m_2$	Note: $z \rightarrow$ LSB variable $w \rightarrow$ MSB variable  Note: $0 \rightarrow \bar{w}$ or $\bar{x}$ or $\bar{y}$ or $\bar{z}$ $1 \rightarrow w$ or $x$ or $y$ or $z$
Row 2	$\bar{w}x$ 01	$\bar{w}x\bar{y}\bar{z}$ $m_4$	$\bar{w}x\bar{y}z$ $m_5$	$\bar{w}xyz$ $m_7$	$\bar{w}xy\bar{z}$ $m_6$	
Row 3	$wx$ 11	$wx\bar{y}\bar{z}$ $m_{12}$	$wx\bar{y}z$ $m_{13}$	$wxyz$ $m_{15}$	$wxy\bar{z}$ $m_{14}$	
Row 4	$w\bar{x}$ 10	$w\bar{x}\bar{y}\bar{z}$ $m_8$	$w\bar{x}\bar{y}z$ $m_9$	$w\bar{x}yz$ $m_{11}$	$w\bar{x}y\bar{z}$ $m_{10}$	

Minterm 6

$f(w, x, y, z)$		Column 1	Column 2	Column 3	Column 4	Note:
		$y + z$ 00	$y + \bar{z}$ 01	$\bar{y} + \bar{z}$ 11	$\bar{y} + z$ 10	
Row 1	$w + x$ 00	$w + x + y + z$ $M_0$	$w + x + y + \bar{z}$ $M_1$	$w + x + \bar{y} + \bar{z}$ $M_3$	$w + x + \bar{y} + z$ $M_2$	Note: $z \rightarrow$ LSB variable $w \rightarrow$ MSB variable $0 \rightarrow \bar{w}$ or $\bar{x}$ or $\bar{y}$ or $\bar{z}$ $1 \rightarrow w$ or $x$ or $y$ or $z$
Row 2	$w + \bar{x}$ 01	$w + \bar{x} + y + z$ $M_4$	$w + \bar{x} + y + \bar{z}$ $M_5$	$w + \bar{x} + \bar{y} + \bar{z}$ $M_7$	$w + \bar{x} + \bar{y} + z$ $M_6$	
Row 3	$\bar{w} + \bar{x}$ 11	$\bar{w} + \bar{x} + y + z$ $M_{12}$	$\bar{w} + \bar{x} + y + \bar{z}$ $M_{13}$	$\bar{w} + \bar{x} + \bar{y} + \bar{z}$ $M_{15}$	$\bar{w} + \bar{x} + \bar{y} + z$ $M_{14}$	
Row 4	$\bar{w} + x$ 10	$\bar{w} + x + y + z$ $M_8$	$\bar{w} + x + y + \bar{z}$ $M_9$	$\bar{w} + x + \bar{y} + \bar{z}$ $M_{11}$	$\bar{w} + x + \bar{y} + z$ $M_{10}$	

Maxterm 8

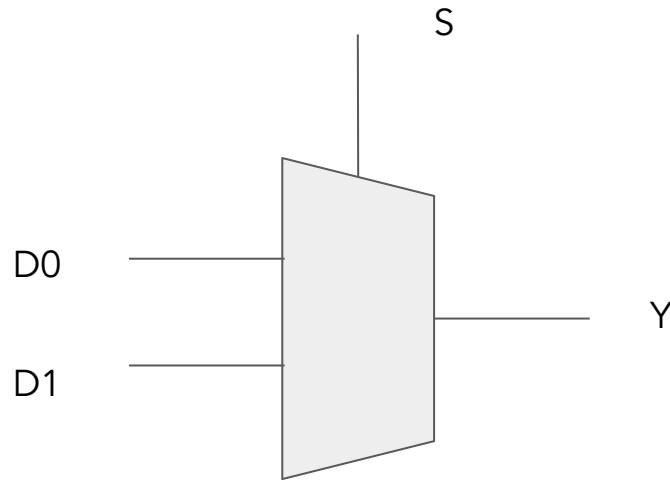


Q1: Simplify  $f(x,y,z) = \sum m(0, 1, 2, 3)$  using Karnaugh Map

Q2. Simplify  $f(x, y, z) = \sum m(2, 3, 5, 7)$  using Karnaugh Map

# Multiplexers: Minimizing a Circuit

They choose an output from several possible inputs based on the value of select signal.



if  $S = 0$

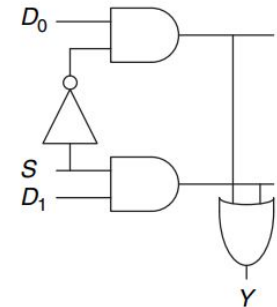
$$Y = D0$$

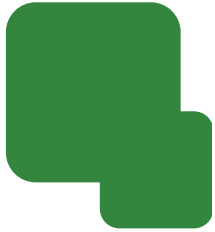
if  $S = 1$

$$Y = D1$$

Y S	D <sub>1:0</sub>			
	00	01	11	10
0	0	1	1	0
1	0	0	1	1

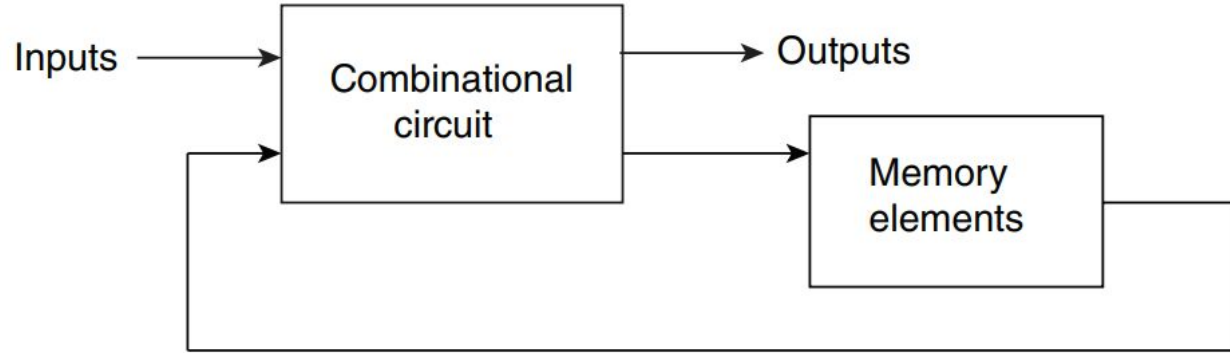
$$Y = D_0 \bar{S} + D_1 S$$





# Sequential Circuits

# Sequential Circuits from Combination Circuits



# Clock



(a) Response to positive level



(b) Positive-edge response

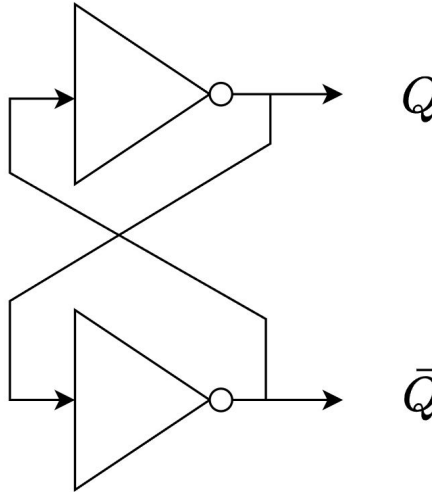
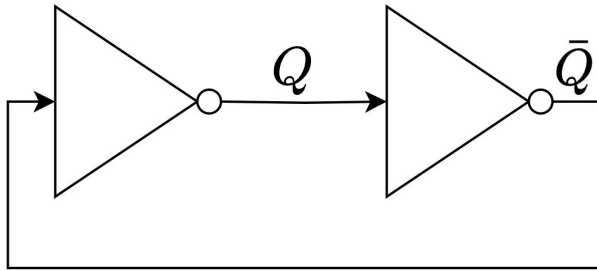


(c) Negative-edge response

Copyright © 2013 Pearson Education, publishing as Prentice Hall

# Latch

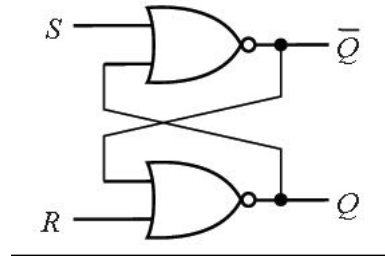
A latch is a combinational (asynchronous) circuit that uses present states to dictate future states. “Memory” is created.



# SR Latch

The simplest latch is the Set-Reset (S-R) latch. You can build one by connecting two **NOR** gates with a cross-feedback loop.

$S = 1$ and $R = 0$	$Q' = 0$ and $Q = 1$ (set)
$S = 0$ and $R = 0$	$Q' = 0$ and $Q = 1$ still (no change)
$S = 0$ and $R = 1$	$Q = 0$ and $Q' = 1$ (reset)
$S = 0$ and $R = 0$	$Q = 0$ and $Q' = 1$ still (no change)
$S = 1$ and $R = 1$	$Q' = 0$ and $Q = 0$ (illegal state)



Input $\bar{S}$	Input $\bar{R}$	Output $Q$
1	1	Previous State
1	0	0
0	1	1
0	0	0 (Invalid)



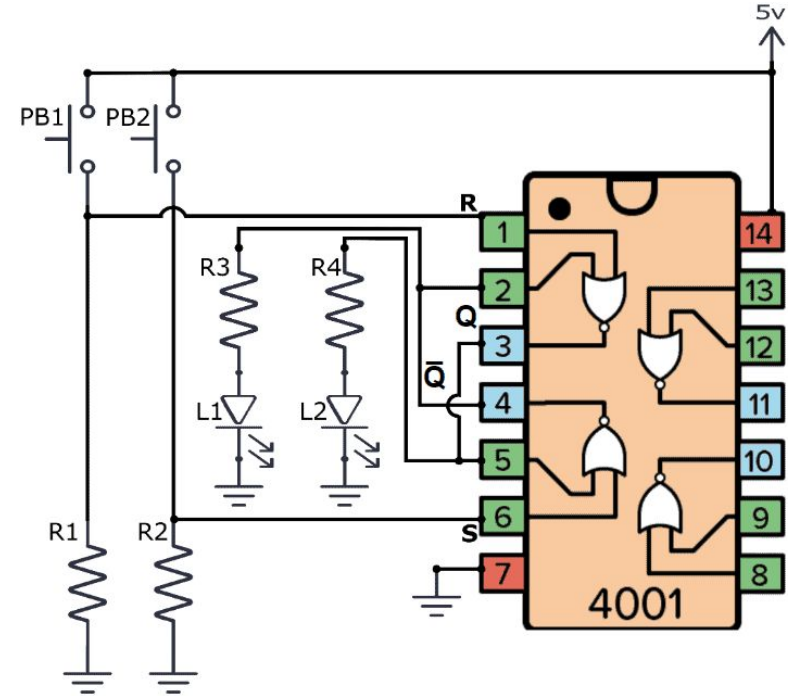
# Circuit for SR Latch

You can build an SR latch using the [CD4001](#) chip.

The CD4001 is a CMOS chip with four NOR gates.

When the button PB2 is pushed, the LED L2 turns on and stays on even after PB2 is released, while the LED L1 remains off. LED L2 turns off when the button PB1 is pressed, whereas LED L1 turns and stays on even after PB1 has been released. To assemble the above circuit you need:

- The CD4001 chip
- Two push buttons (PB1 and PB2)
- Two LEDs
- Two 10 k $\Omega$  resistors (R1 and R2)
- Two 330  $\Omega$  resistors (R3 and R4)

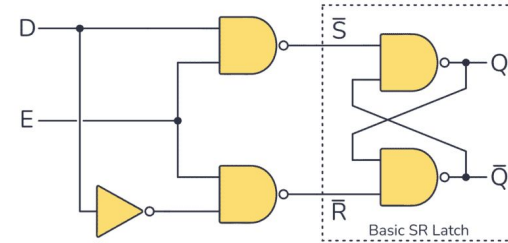


# D-Latch

- A D Latch is a logic circuit that can store one bit of data (1 or 0).
- The D Latch has two inputs: D (Data) and E (Enable).
- The latch will only change its output (Q) when the Enable input is HIGH. If E is HIGH, the output Q will reflect the value of the D input. If E is LOW, the output will remain the same, regardless of the D input.
- The D Latch avoids the “undefined” or “invalid” state problem found in the S-R latch. This is because the inverter at the input of the D Latch makes sure the S and R inputs are always opposites.

- **Function**

- When CLK = 1
  - D passes through to Q (transparent)
- When CLK = 0
  - Q holds its previous value (opaque)
- Avoids invalid case when  $Q \neq \text{NOT } Q$



E	D	Q	Description
0	X	Q	Memory (no change)
1	0	0	Reset Q to 0
1	1	1	Set Q to 1

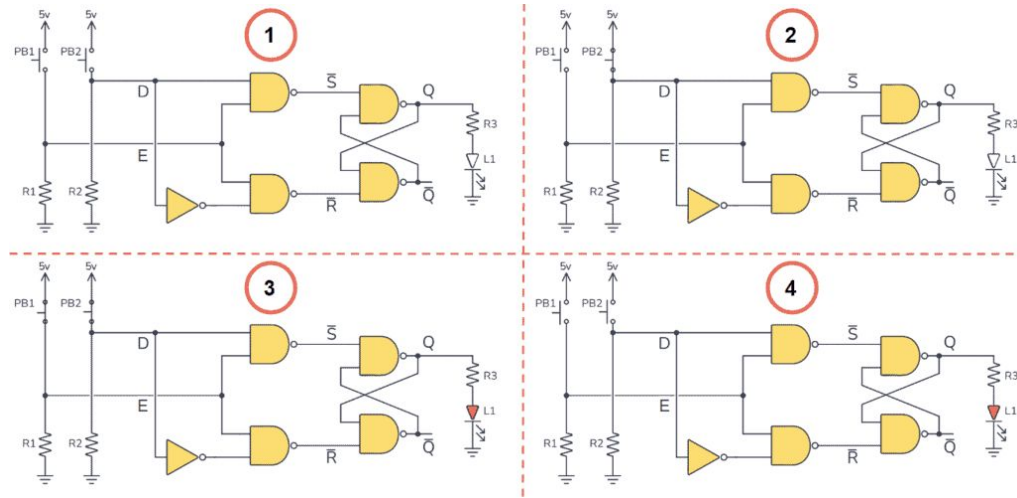
# D-Latch Circuit

1: Q is 0 (LED L1 off), and both PB1 and PB2 are not pressed.

2: PB2 is pushed. You now have a 1 on the D input, but the output Q remains as 0 because the E input hasn't received an enable signal yet.

3: When PB1 is pressed, a 1 on the E input appears and places the bit 1 from D to Q. When Q is 1 it turns on LED L1.

4: PB1 and PB2 return to their original states in section 4, LED L1 remains ON indicating that the Q output has not changed.



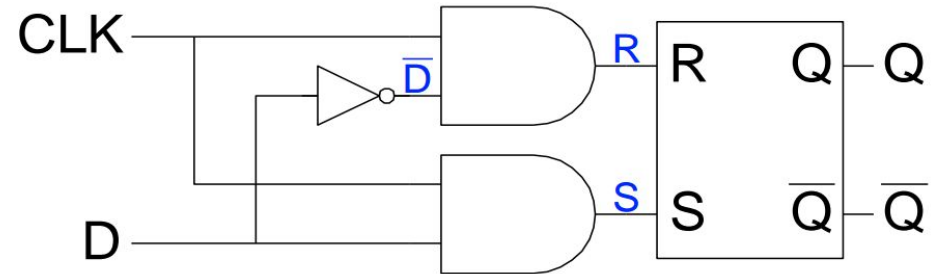
To assemble the above circuit you need:

- Four NAND gates (Ex [CD4011](#))
- One NOT gate (Ex CD4049 or CD4069)
- 2x pushbuttons
- 1x LED
- 2x 10 k $\Omega$  resistors (R1 and R2)
- 1x 330  $\Omega$  resistors (R3)

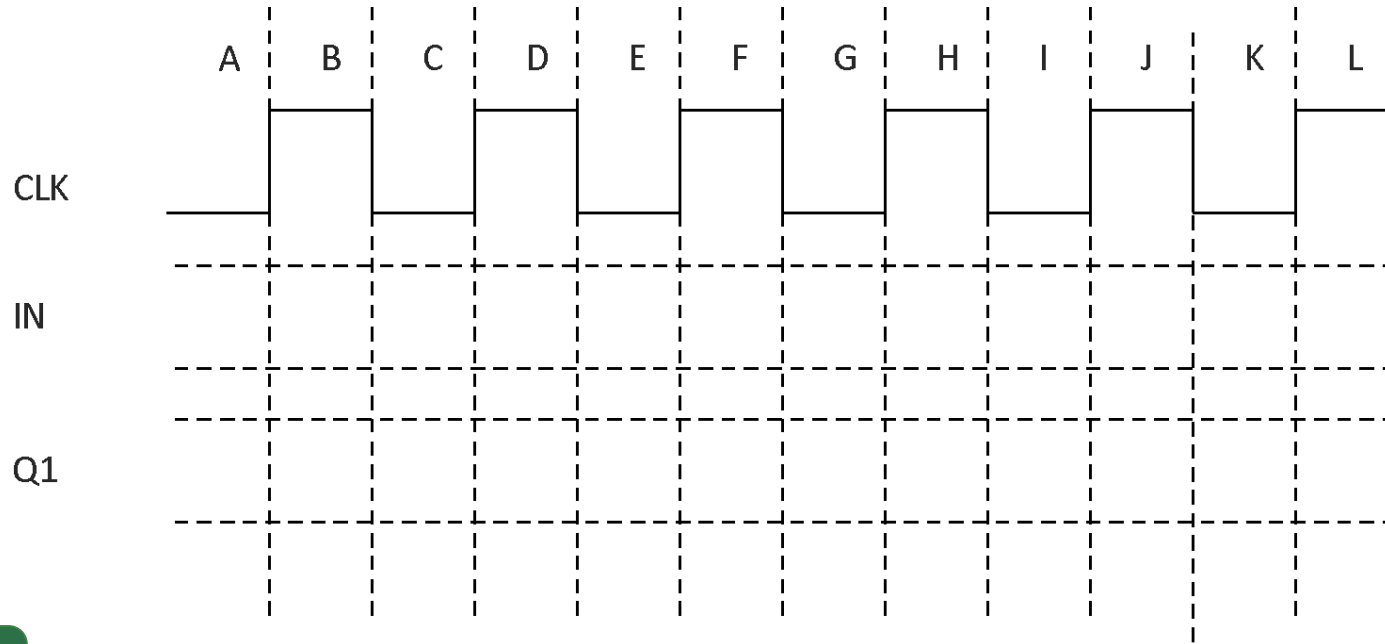
# D-Latch

- Two inputs: CLK, D
  - CLK**: controls when the output changes
  - D** (the data input): controls what the output changes to
- Fixes Issues with SR Latch when both S and R are asserted.
- Function
  - When CLK = 1
    - D passes through to Q (transparent)
  - When CLK = 0
    - Q holds its previous value (opaque)
- Avoids invalid case when  $Q \neq \text{NOT } Q$

CLK	D	$\overline{D}$	S	R	Q	$\overline{Q}$
0	X	X	0	0	$Q_{prev}$	$\overline{Q}_{prev}$
1	0	1	0	1	0	1
1	1	0	1	0	1	0



# Practice with a D Latch



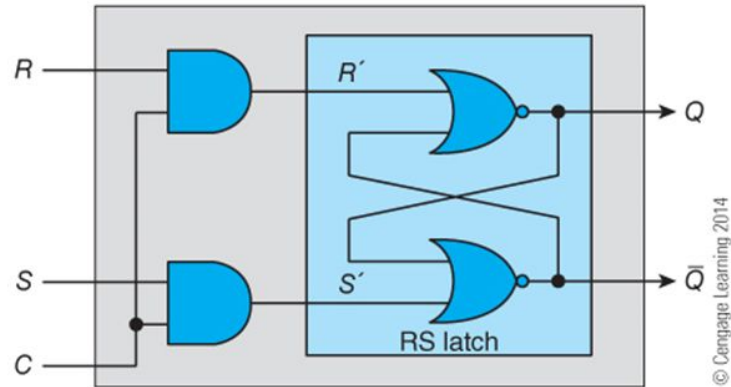
# Flip Flops

A latch can change its output at any time as long as it's enabled, a flip flop is an edge-triggered device that needs a clock transition to change its output.

# SR Flip Flop

FIGURE 2.36

The clocked RS flip-flop

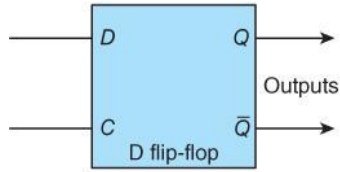


© Cengage Learning 2014

# The D Flip-Flop

FIGURE 2.37

The D flip-flop



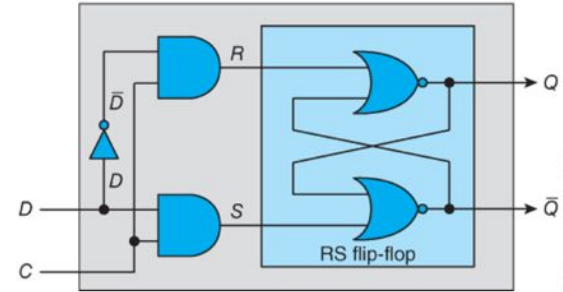
(a) Symbol for D flip-flop

$C$	$D$	$Q^+$
0	0	$Q$
0	1	$Q$
1	0	0
1	1	1

(b) Truth table of D flip-flop

FIGURE 2.38

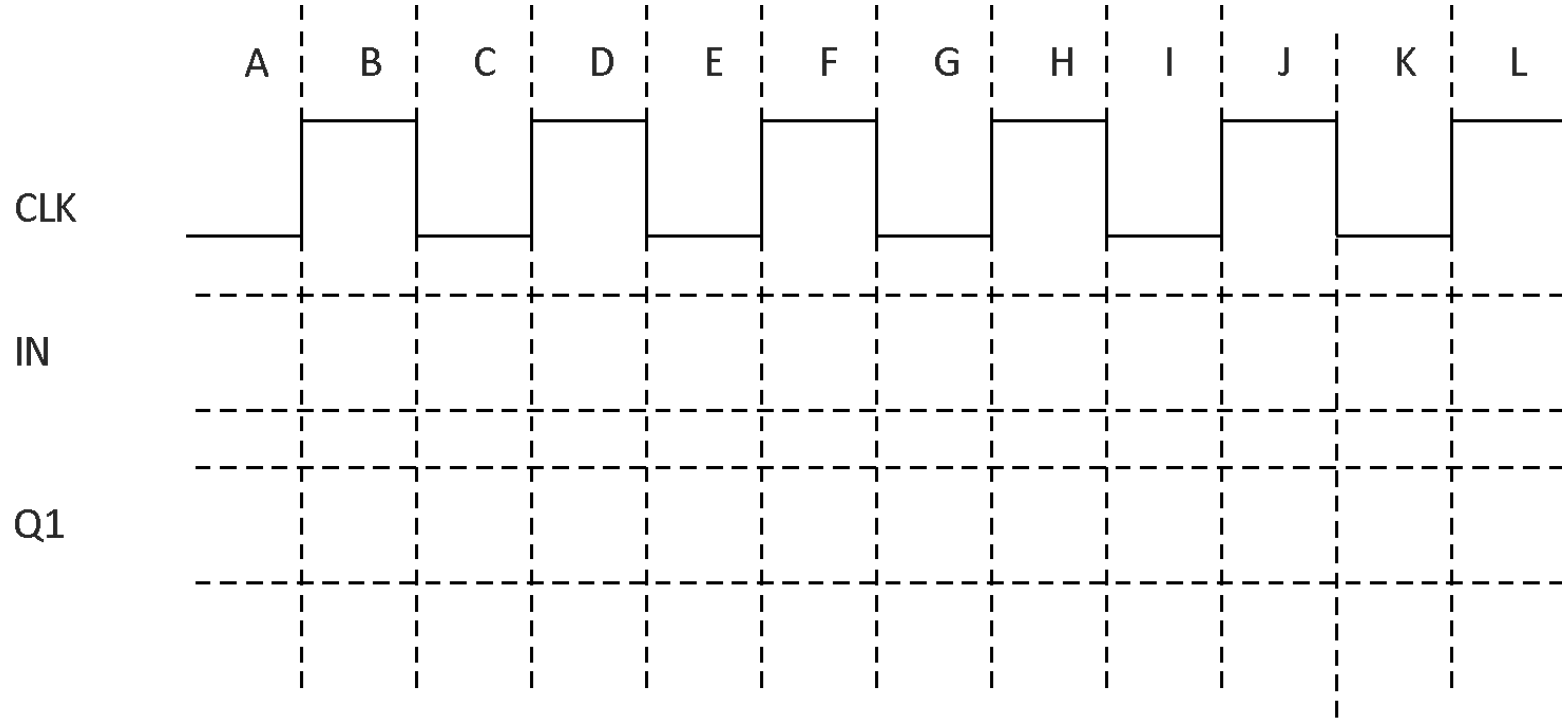
Circuit of a D flip-flop



© Cengage Learning 2014

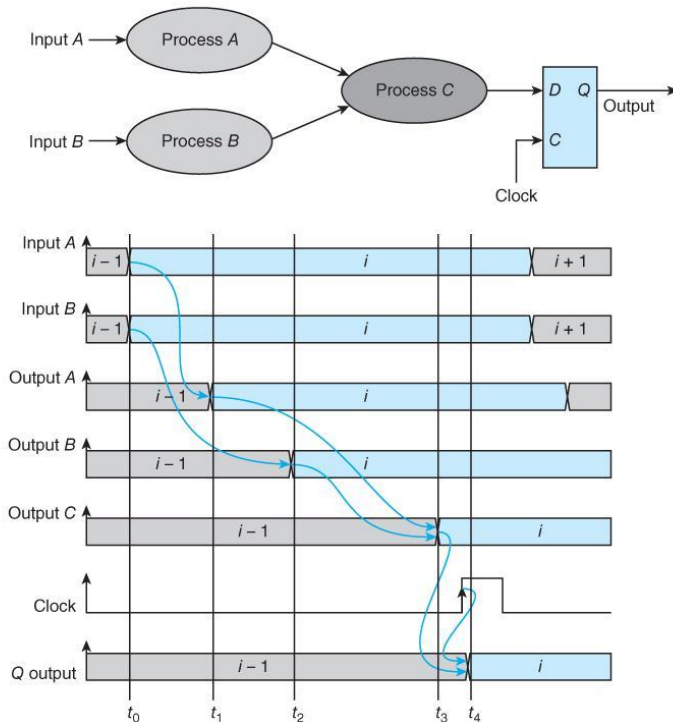


# Practice with a D Flip-Flop



# Timing in Sequential Circuits

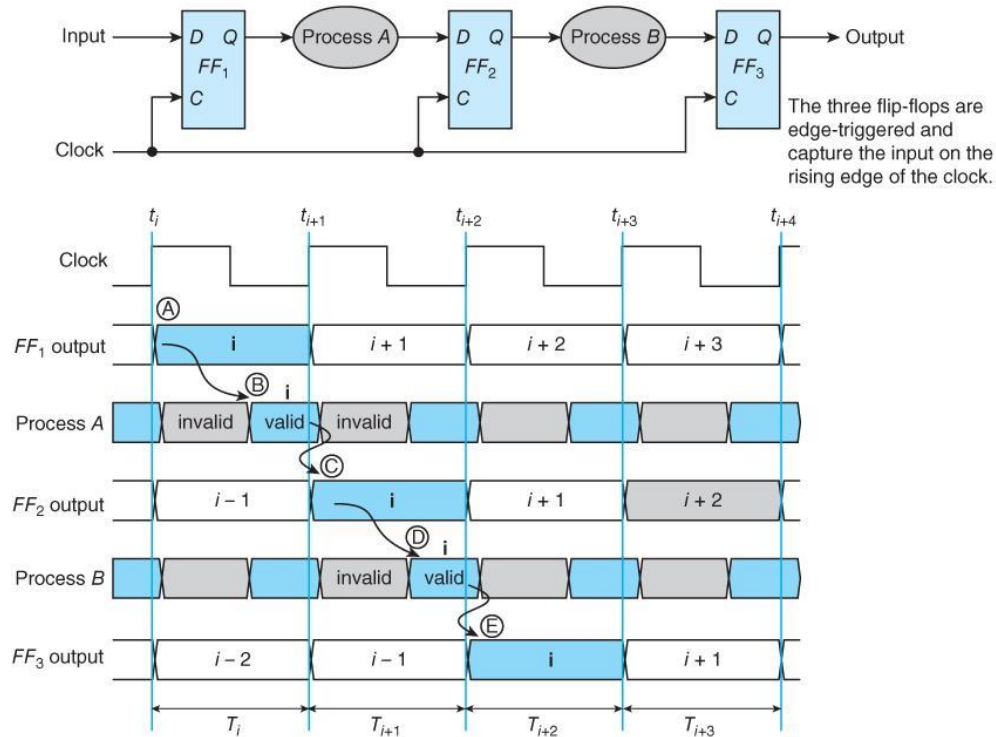
FIGURE 2.40 Capturing the output of a system



© Cengage Learning 2014

# Timing in Sequential Circuits (Pipelining)

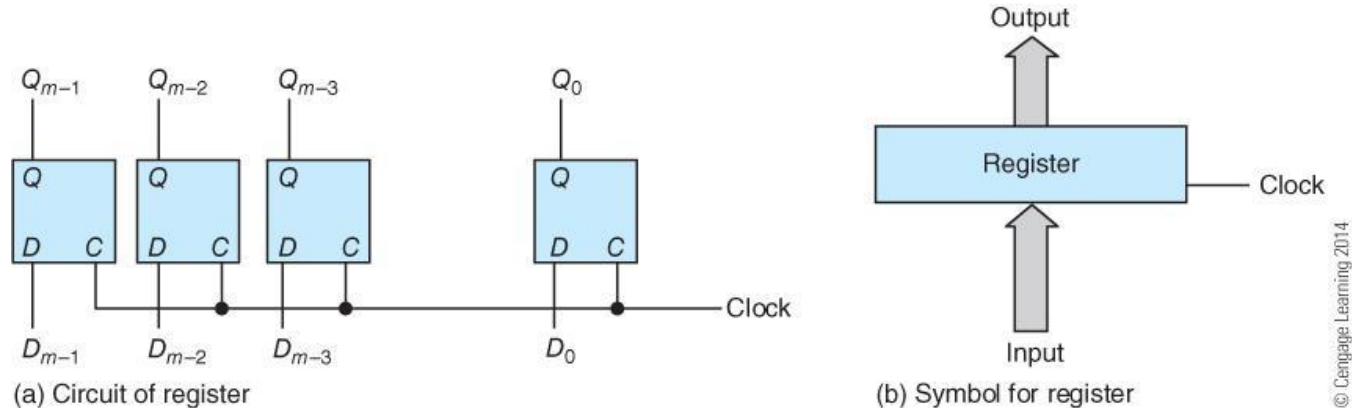
FIGURE 2.41 Pipelining using flip-flops



© Cengage Learning 2014

# Registers

FIGURE 2.44 The register

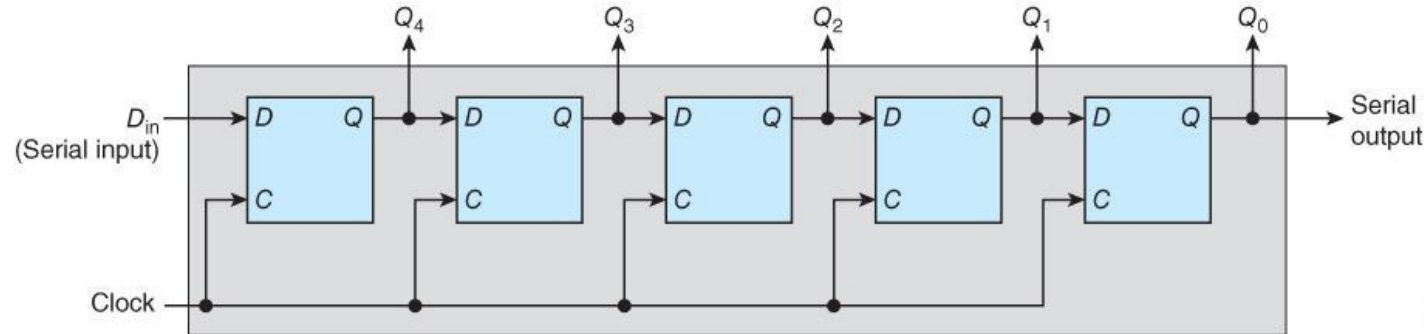


The register is an  $m$ -bit storage element that uses  $m$  flip-flops to store an  $m$ -bit word. The clock inputs of the flip-flops are connected together and all flip-flops are clocked together.

# A Shift Register

This shift register shifts **right** only. A shift register can be designed to shift **left** only, to be capable of shifting left or right, and to have **parallel load**. Shifting is useful to multiply or divide by powers of 2. It can also be used to look at individual bits of the contents of a register so that decisions can be made based on their value.

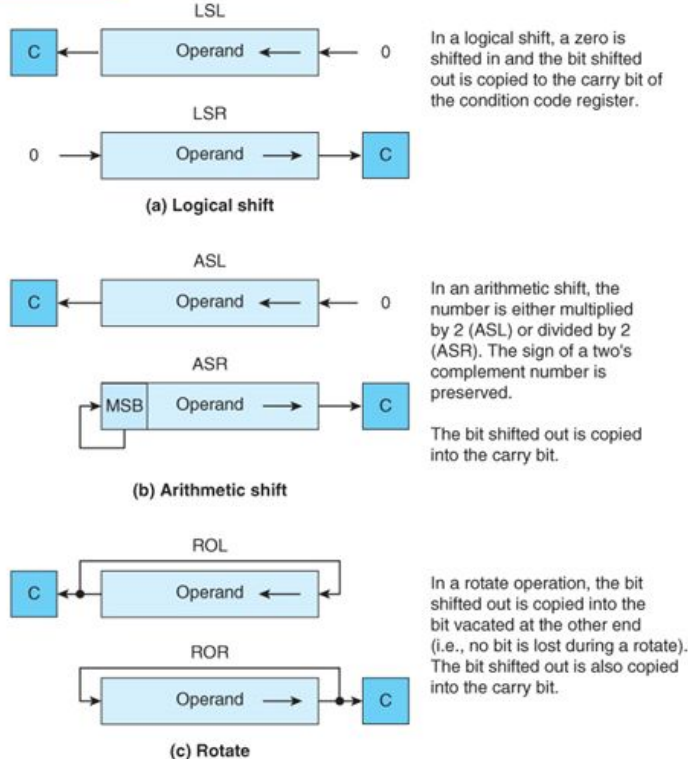
**FIGURE 2.45** The shift register



© Cengage Learning 2014

# Other Shift operations

FIGURE 3.23 Shift operations

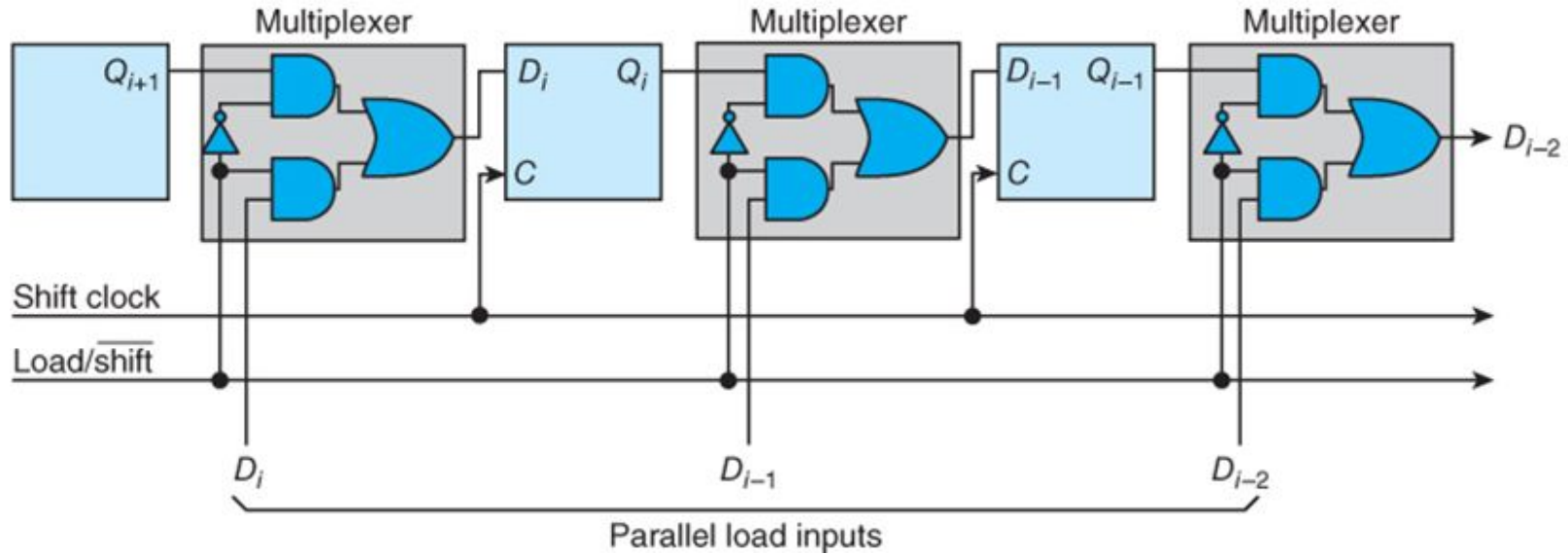


Shift type	Shift Left	Shift Right
Original bit pattern before shift	1101 0111	1101 0111
Logical shift	1010 1110	0110 1011
Arithmetic shift	1010 1110	1110 1011
Circular shift	1010 1111	1110 1011

## Parallel Load Capacity

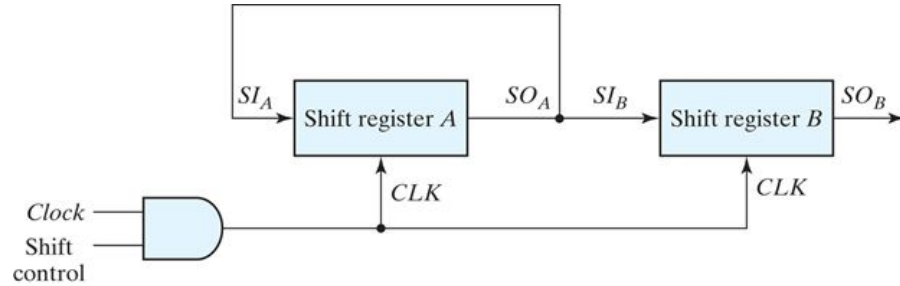
FIGURE 2.48

### Shift register with a parallel load capability

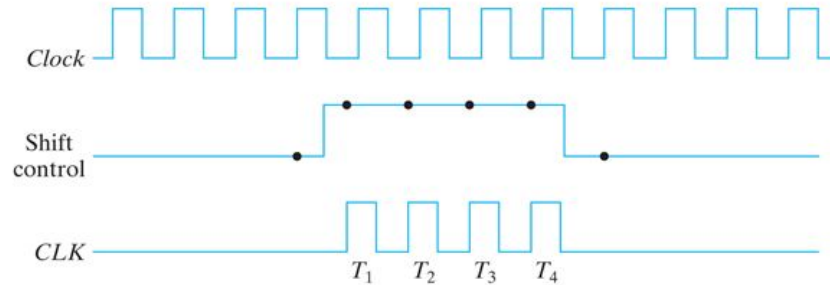


© Cengage Learning 2014

# Serial Input/ Serial Output



(a) Block diagram

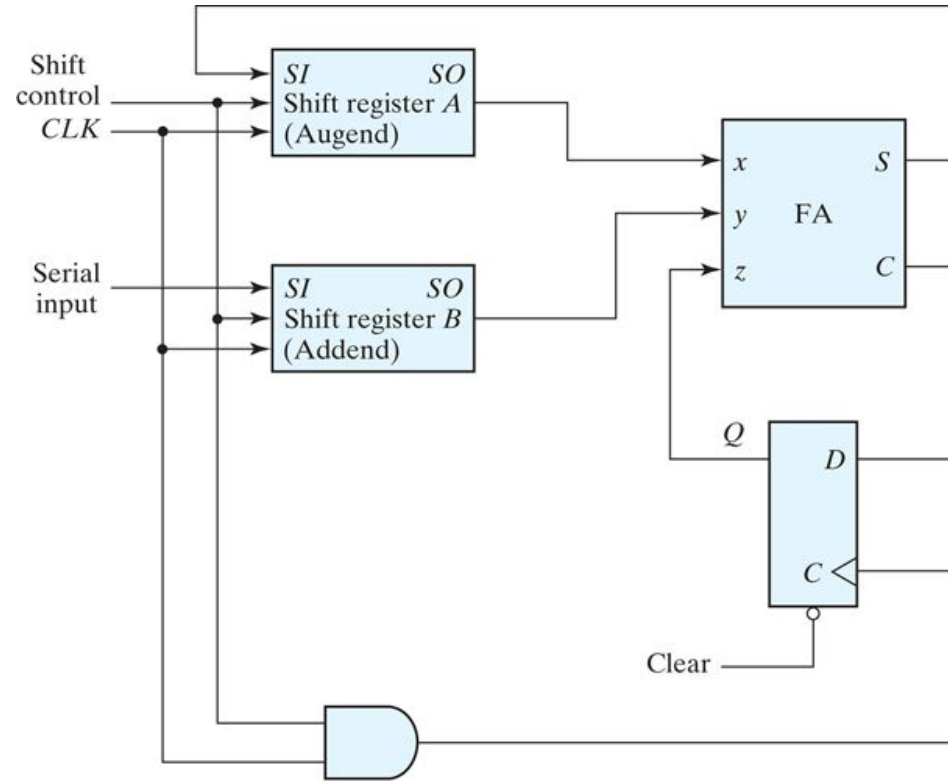


(b) Timing diagram

Copyright © 2013 Pearson Education, publishing as Prentice Hall

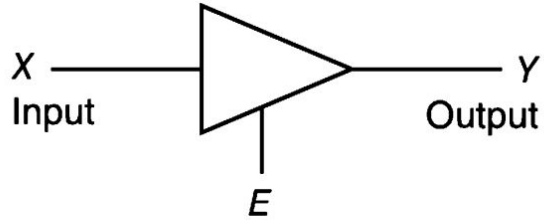


# Serial Adder

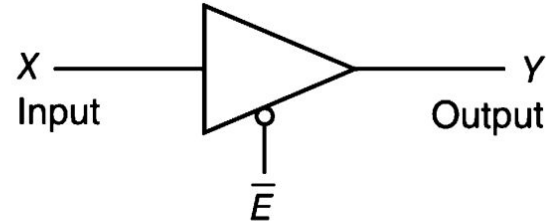


Copyright ©2013 Pearson Education, publishing as Prentice Hall

# Tristate Gates

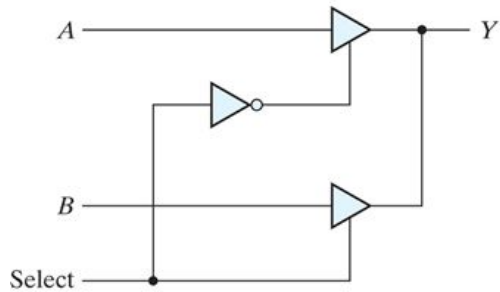


(a) Non-inverting buffer with active-high enable

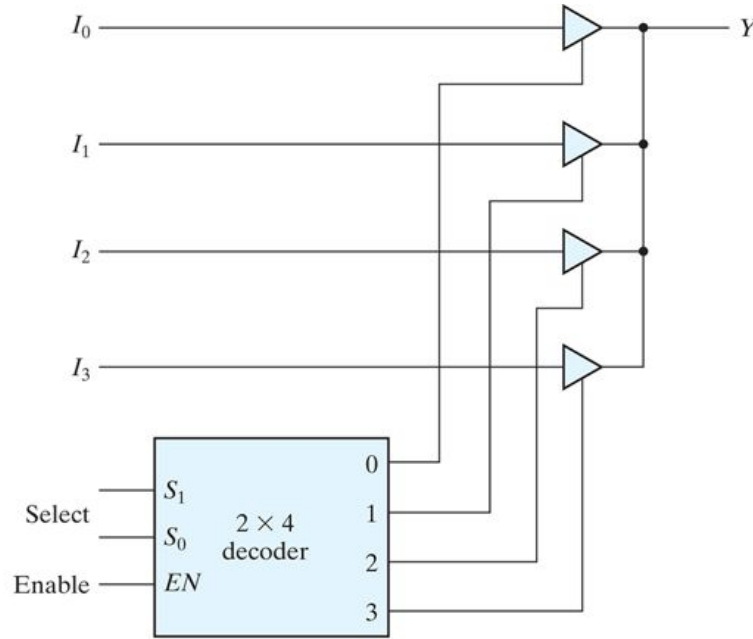


(b) Non-inverting buffer active-low enable

# Tristate Buffers



(a) 2-to-1-line mux



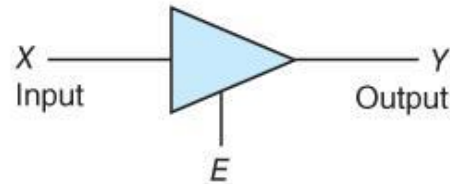
(b) 4-to-1-line mux

Copyright ©2013 Pearson Education, publishing as Prentice Hall

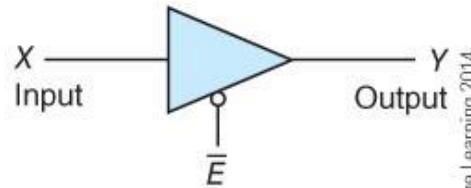
# Buses and Tristate Gates

FIGURE 2.56

The tristate gate (tristate buffer)



(a) Non-inverting buffer with active-high enable



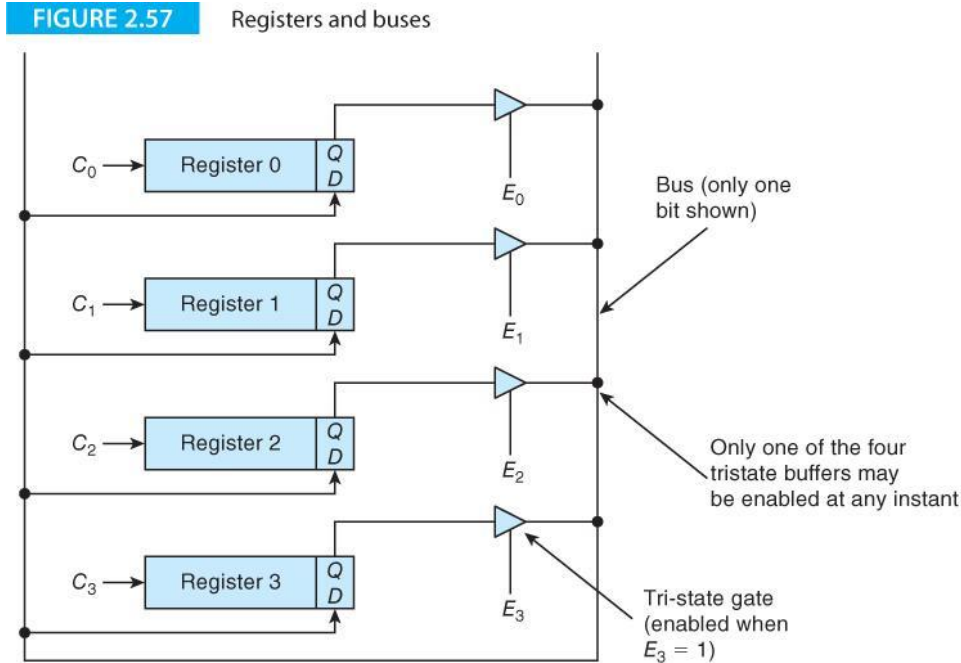
(b) Non-inverting buffer active-low enable

© Cengage Learning 2014

When we connect multiple drivers to a bus, we need a way for only one of them to drive the bus at a time. One way to do this is tristate gates with enable inputs that come from the output of a decoder so that only one tristate is enabled at a time.

# Registers, Buses and Functional Units

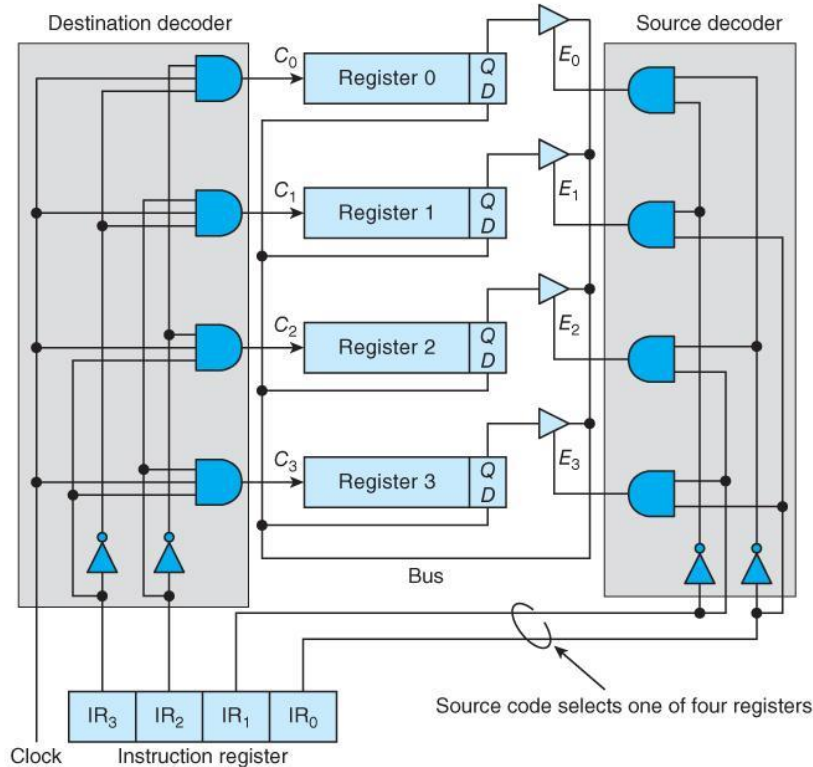
E0 - E3 would come from a decoder



© Cengage Learning 2014

# Implementing a MOVE (Copy) Instruction

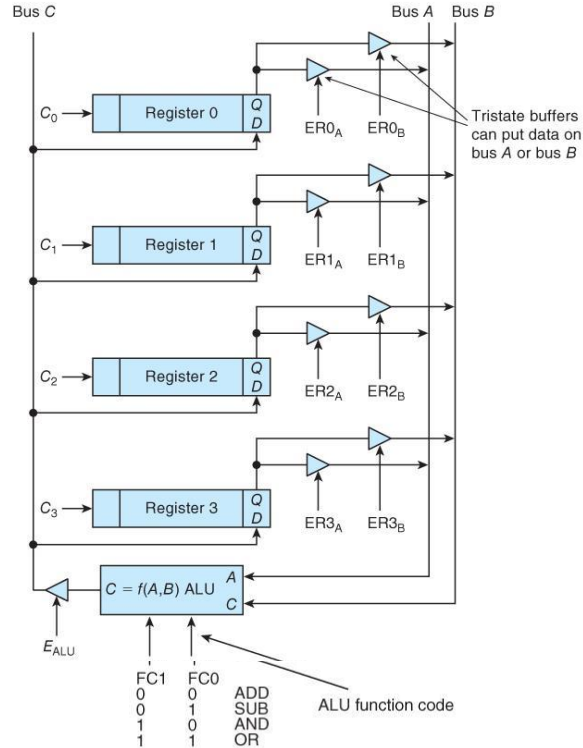
FIGURE 2.58 Controlling the bus



© Cengage Learning 2014

# Multiple Buses and an ALU

FIGURE 2.59 The registers, buses, and ALU



# Video of the day

## The History of the Integrated Circuit

<https://www.youtube.com/watch?v=SYSJefKc7L4>