CPE 221: Computer Organization

THE UNIVERSITY OF
ALABAMA IN HUNTSVILLE

02 Number Systems and Digital Logic
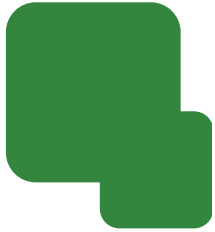rahul.bhadani@uah.edu

Rahul Bhadani

Announcement

Homework 01
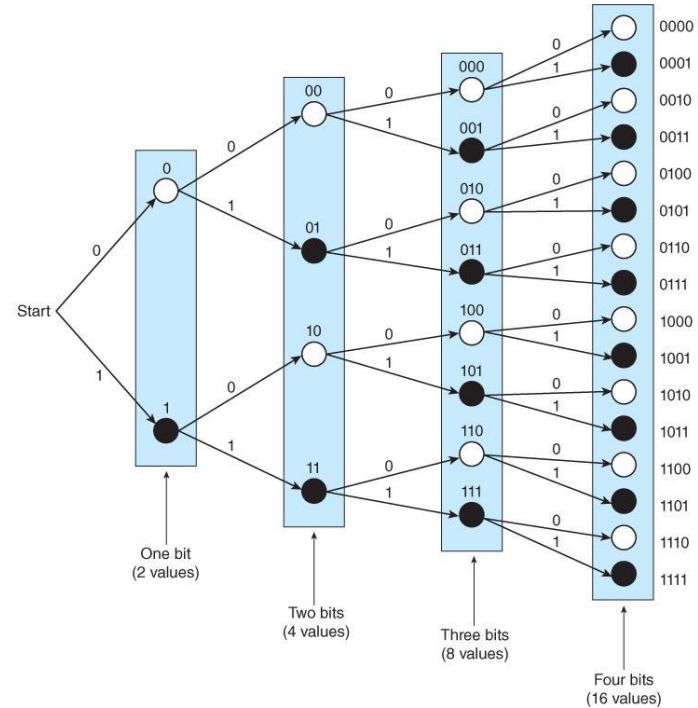
Due: Jan 24, 2025

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.

THE UNIVERSITY OF
ALABAMA IN HUNTSVILLE

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

One bit (2 values)

Two bits (4 values)

Three bits (8 values)

Four bits (16 values)

© Cengage Learning 2014

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

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Rahul Bhadani

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

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

$(1345)_6 = (1 \times 216) + (3 \times 36) + (4 \times 6) + (5 \times 1)$
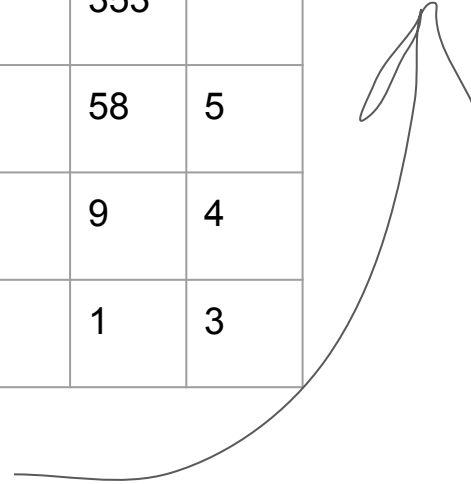
$(1345)_6 = (353)_{10}$

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Converting from Base 10 to Base 6

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

6	353	
6	58	5

6	353	
6	58	5
	9	4

6	353	
6	58	5
	9	4
	1	3

Rahul Bhadani

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Binary to Decimal (Positive Unsigned)

From base 2 to base 10:

$(10111000)_2 =$

$1\text{x}2^7 + 0\text{x}2^6 + 1\text{x}2^5 + 1\text{x}2^4 + 1\text{x}2^3 + 0\text{x}2^2 + 0\text{x}2^1 + 0\text{x}2^0$

$128 \quad + 0 \quad\quad + 32 \quad + 16 \quad + 8 \quad\quad + 0 \quad + 0 \quad\quad + 0$

$= 184$

Rahul Bhadani

Binary Addition

	1	1	
1	0	1	1
0	0	1	1

1	1	1	0

THE UNIVERSITY OF
ALABAMA IN HUNTSVILLE

System 1: Sign/Magnitude

N-bit binary number

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

For example

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

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

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

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.

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

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

$11100100 \Rightarrow 00011011 \Rightarrow 00011100$

Rahul Bhadani

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

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

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

Rahul Bhadani

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

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_2

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

$= 16 + 8 + 2 + 0.5$

$= 26.5$

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	…

THE UNIVERSITY OF
ALABAMA IN HUNTSVILLE

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.

$2.5_{10} = 0101_2$

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

$-2.5_{10} = 1011_2$

THE UNIVERSITY OF
ALABAMA IN HUNTSVILLE

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

$$+ \; 4.1 \; \times \; 2^3$$

sign   mantissa   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
∞	0	11111111	00000000000000000000000
$-\infty$	1	11111111	00000000000000000000000
NaN	X	11111111	Non-zero

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

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.

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

$$
\begin{aligned}
0.085 \ / \ 2^{-1} &= 0.17 \\
0.085 \ / \ 2^{-2} &= 0.34 \\
0.085 \ / \ 2^{-3} &= 0.68 \\
0.085 \ / \ 2^{-4} &= 1.36
\end{aligned}
$$

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        Once this process terminates or starts repeating,
0.08 x 2 = 0.16        we read the unit's digits from top to bottom
0.16 x 2 = 0.32        to reveal the binary form for 0.36:
0.32 x 2 = 0.64
0.64 x 2 = 1.28          0.0101110000101000111101011000...
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 =  ...    (at this point the list starts repeating)
```

So 0.085 in IEEE 754 format is:

0 01111011 01011100001010001111011
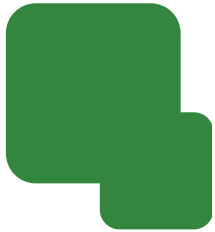
Data Formats

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

THE UNIVERSITY OF
ALABAMA IN HUNTSVILLE

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.

Rahul Bhadani

ASCII Table

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

Spring 2025

Rahul Bhadani

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Unicode

Everyone in the world should be able to use their own language on phones and computers.

▶ LEARN MORE ABOUT UNICODE

https://home.unicode.org

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

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

THE UNIVERSITY OF
ALABAMA IN HUNTSVILLE
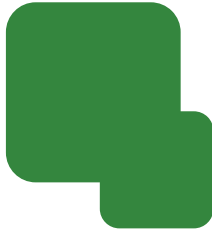
Computer Logic

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
$$AB+\overline{A}C+BC = AB+\overline{A}C$$
$$(A+B)(\overline{A}+C)(B+C)=(A+B)(\overline{A}+C)$$

6. Transposition theore
$$AB+\overline{A}C=(A+C)(\overline{A}+B)$$

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

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

Rahul Bhadani

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

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.

FIGURE 2.12

Levels of hardware

Processor

Microprogram

System (Registers, circuits, buses)

Wiring

Gates

© Cengage Learning 2014

Spring 2025

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Rahul Bhadani
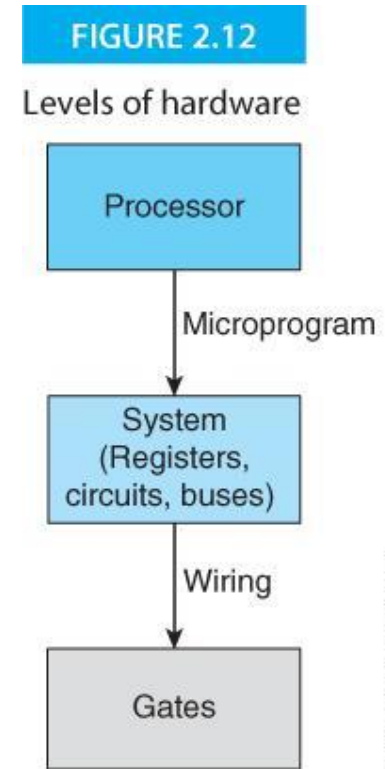
Computer Logic using Logic Gates (2)

- 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.

FIGURE 2.12

Levels of hardware

Processor

Microprogram

System (Registers, circuits, buses)

Wiring

Gates

© Cengage Learning 2014

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE
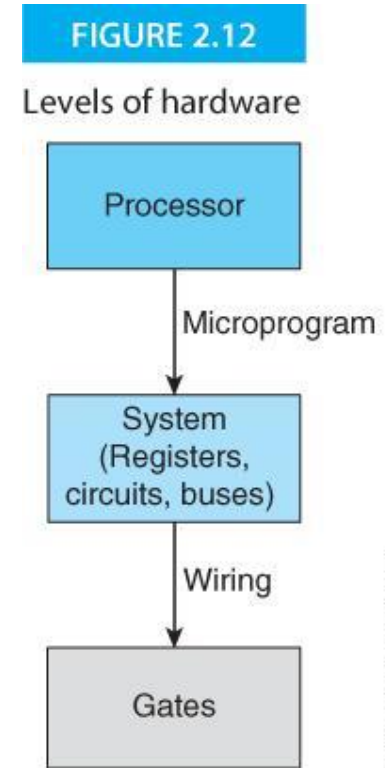
Why Logic Gates are called Logic Gates

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".

Rahul Bhadani
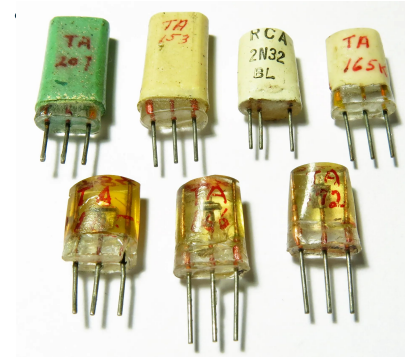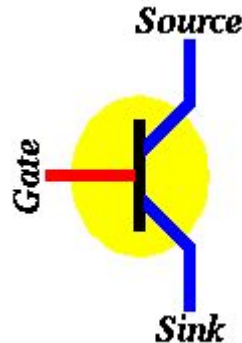
HOW THE FIRST TRANSISTOR WORKED

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Computer Logic Diagram

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

Combinational circuit

n inputs

m outputs

Copyright ©2013 Pearson Education, publishing as Prentice Hall

THE UNIVERSITY OF
ALABAMA IN HUNTSVILLE
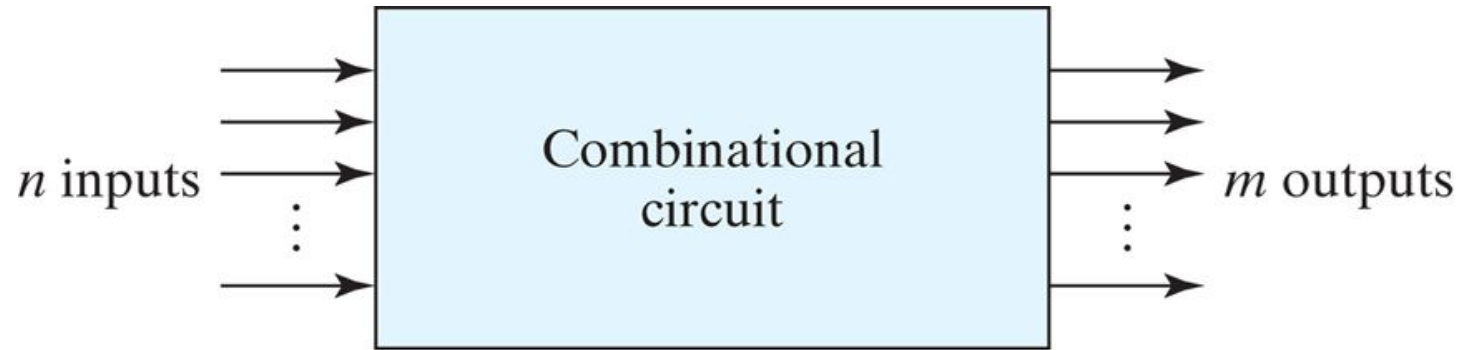
Transistor

- "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

Rahul Bhadani

AND Gate

TABLE 2.8		Truth Table for the AND Gate

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

(a) Two-input AND gate

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

(b) Three-input AND gate

© Cengage Learning 2014

FIGURE 2.14		The symbol for an AND gate

(a) Two-input AND gate $C = A \cdot B$

(b) Three-input AND gate $D = A \cdot B \cdot C$

© Cengage Learning 2014

Rahul Bhadani

OR Gate

TABLE 2.9   Truth Table for the OR Gate

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

(a) Two-input OR gate

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

(b) Three-input OR gate

© Cengage Learning 2014

FIGURE 2.15   The symbol for an OR gate

A
B
$C = A + B$

(a) Two-input OR gate

A
B
C
$D = A + B + C$
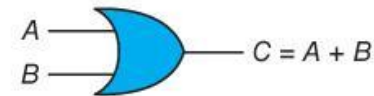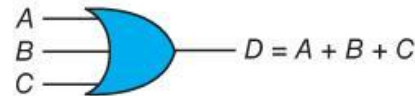
(b) Three-input OR gate

© 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

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

- 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

$$A \longrightarrow \!\!\!\!\!\!\!\triangleright\!\!\circ \longrightarrow \overline{A}$$
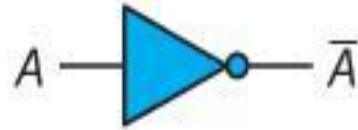
(a) Symbol for inverter

A	$\overline{A}$
0	0
0	1

© Cengage Learning 2014

(b) Truth table of inverter

Which logic statement(s) below is true?

1. B **OR** B' = 1
2. B **AND** B' = 0
3. A **OR** 1 = 0

THE UNIVERSITY OF
ALABAMA IN HUNTSVILLE

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

© Cengage Learning 2014

FIGURE 2.19 Three derived gates

A

B → $C = \overline{A + B}$

(a) NOR gate

A

B → $C = \overline{A \cdot B}$

(b) NAND gate

A

B → $C = A \oplus B$
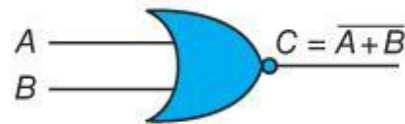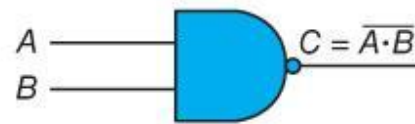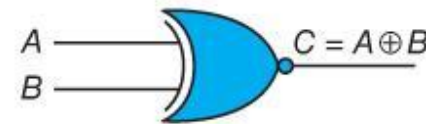
(c) Exclusive OR gate

© Cengage Learning 2014

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, NOT

FIGURE 2.20 Constructing an XOR circuit from AND, OR, and NOT gates

$F = \bar{A} \cdot B + A \cdot \bar{B}$
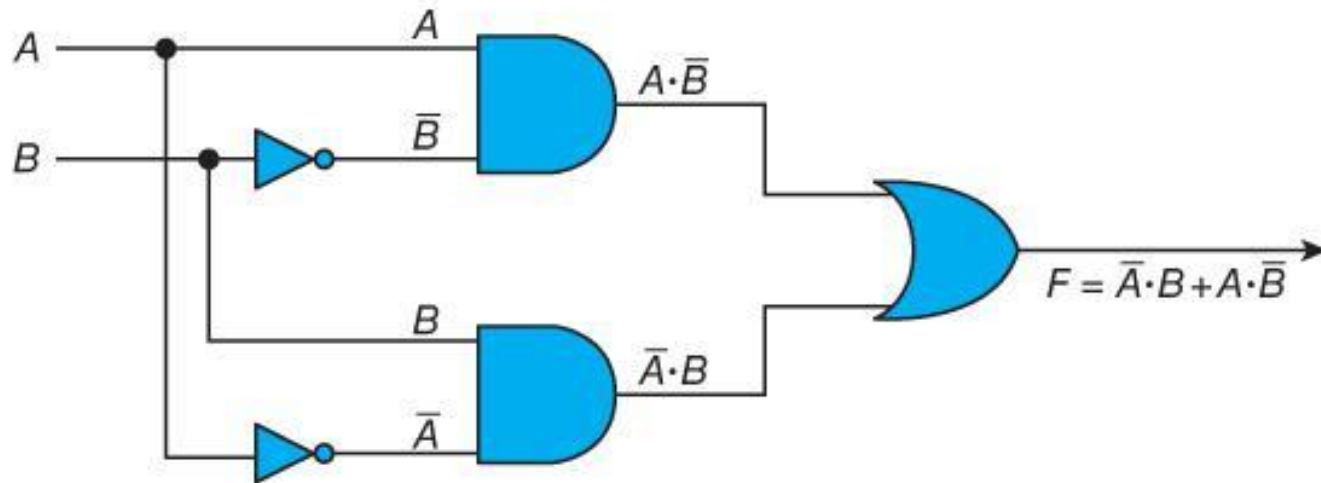
© Cengage Learning 2014
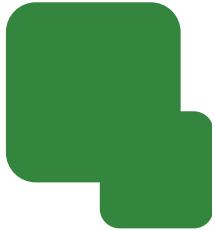
Combination Circuits

Inversion Bubbles

© Cengage Learning 2014
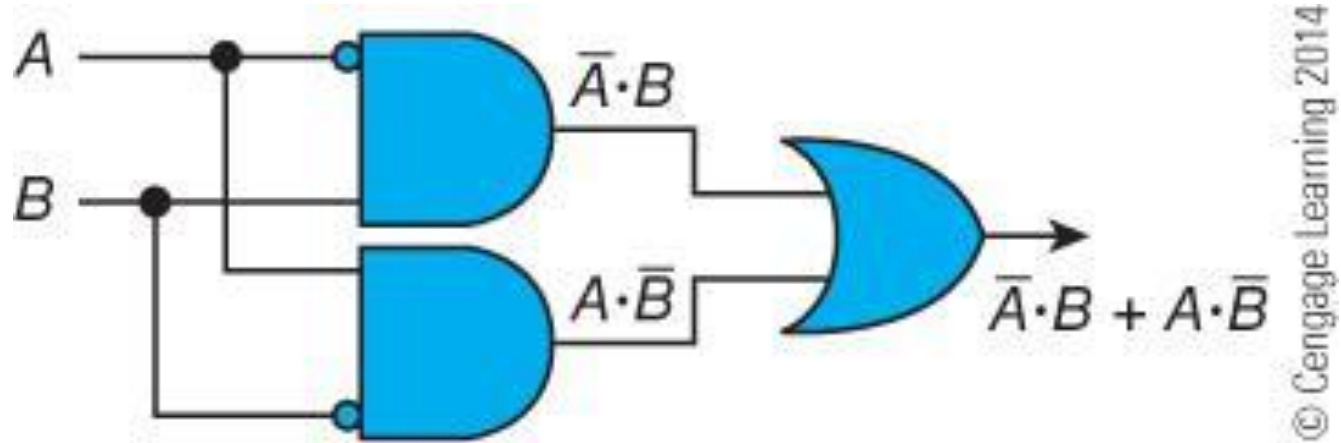
The Half Adder

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

FIGURE 2.22 The two-bit adder (the half adder)

Two-bit adder

$Sum = A \oplus B$

$Carry = A \cdot B$
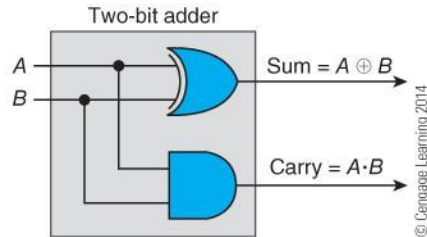
The Full Adder

FIGURE 2.23   The full adder
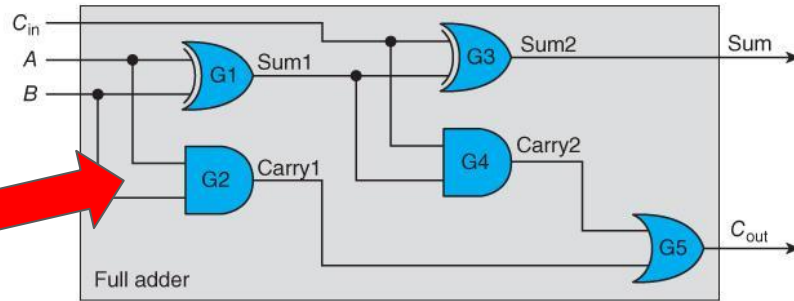
Half adder here
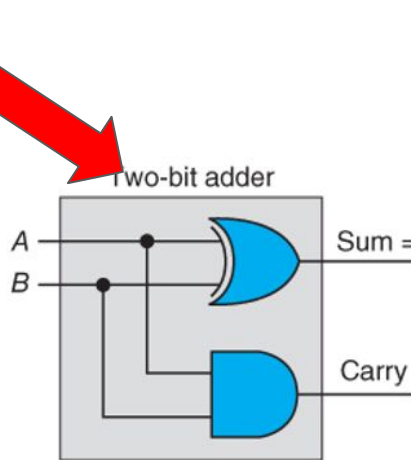
Cin	A	B	Sum	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

1. A full adder has inputs A, B, and Cin with the values of $(1,1,0)$. What are the output values for Cout and S?
2. A full adder has inputs A, B, and Cin with the values of $(1,1,1)$. What are the output values for Cout and S?

THE UNIVERSITY OF
ALABAMA IN HUNTSVILLE

Ripple Carry Adder
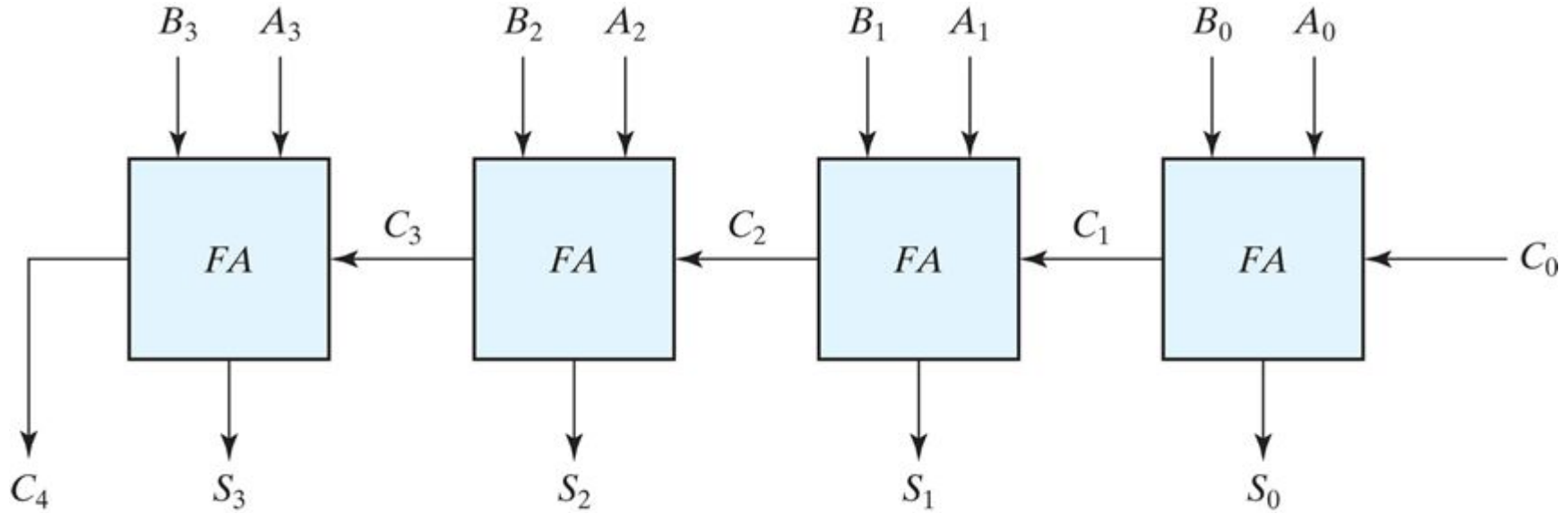
Copyright ©2013 Pearson Education, publishing as Prentice Hall

Decoding an Instruction

3-to-8 Active High Decoder

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

$D_0 = x'y'z'$

$D_1 = x'y'z$

$D_2 = x'yz'$

$D_3 = x'yz$

$D_4 = xy'z'$

$D_5 = xy'z$

$D_6 = xyz'$

$D_7 = xyz$

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE
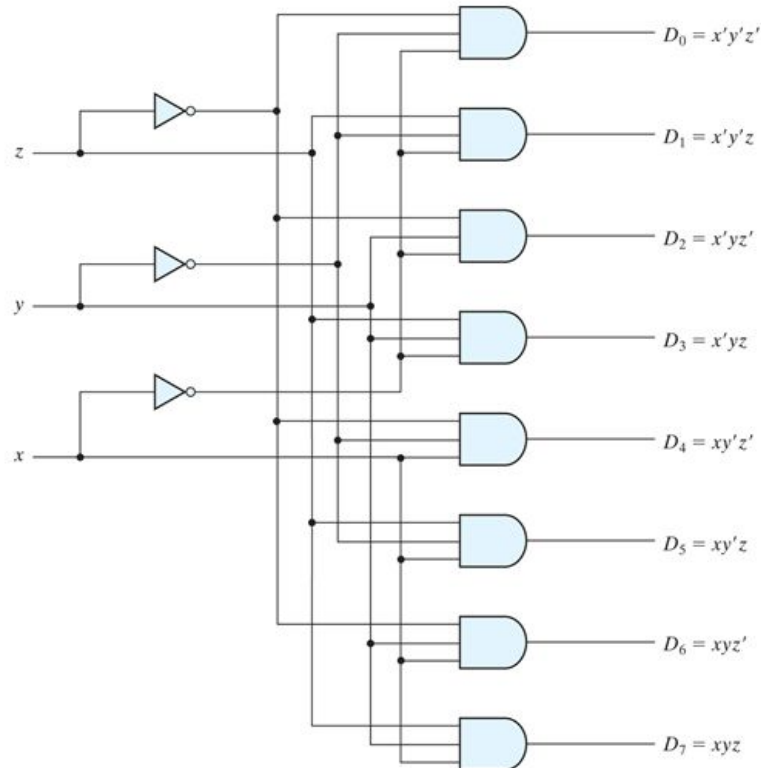
Decoder Truth Table

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

Inputs			Outputs							
x	y	z	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
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
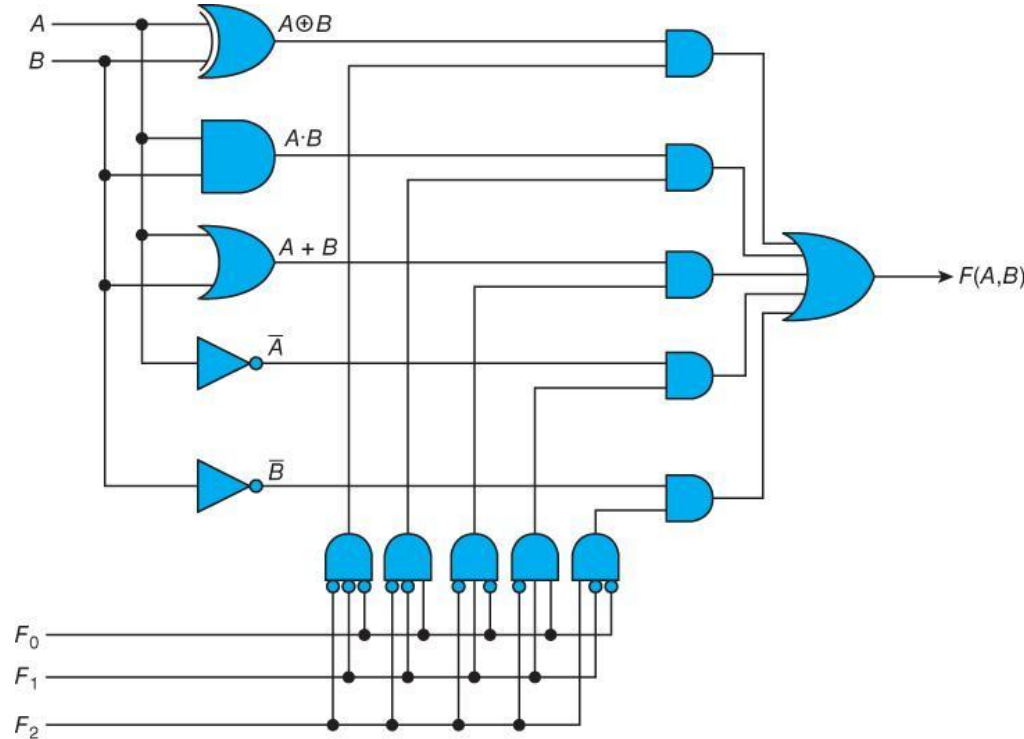
© Cengage Learning 2014

Decoding an Instruction

FIGURE 2.28   Application of a decoder

Instruction register

Op-code

A   B   C

Decoded instruction

Y_0 Add
Y_1 Subtract
Y_2 Load
Y_3 Store
Y_4 Branch on zero
Y_5 Branch on not zero
Y_6 Branch unconditionally
Y_7 Stop

© Cengage Learning 2014

Spring
2025

Rahul Bhadani

THE UNIVERSITY OF
ALABAMA IN HUNTSVILLE
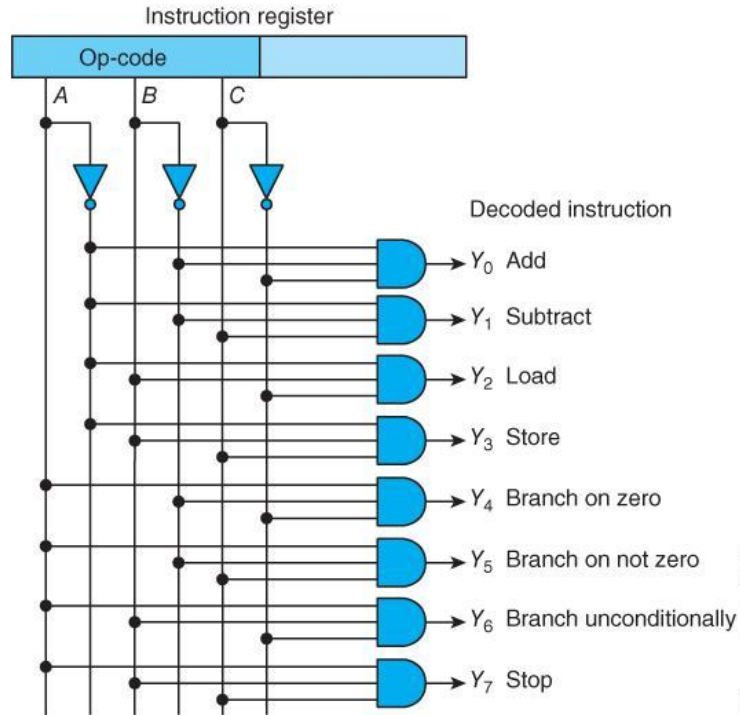
Multiplexer

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.

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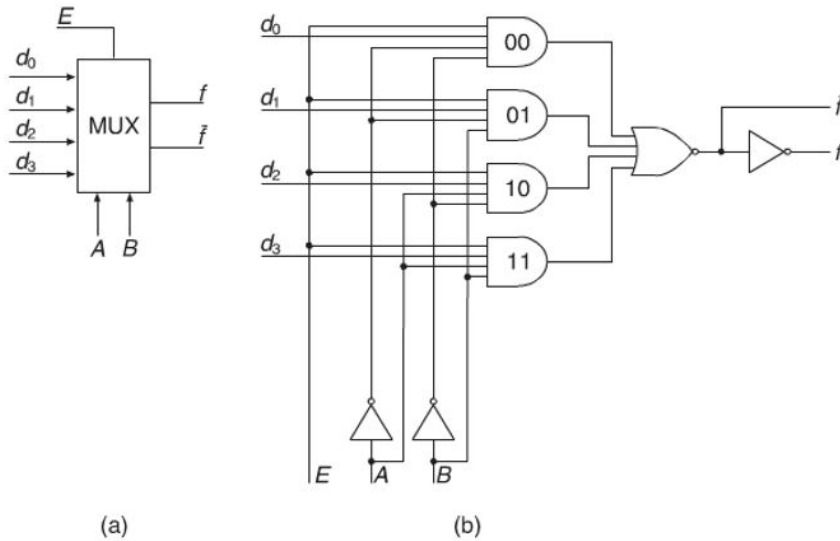
Figure 5.1 (a) Block diagram of a 4-input multiplexer and (b) its gate implementation

FIGURE 2.30 Alternative representation of the two-input multiplexer

$$F = A \cdot \bar{C} + B \cdot C$$

© Cengage Learning 2014

Rahul Bhadani
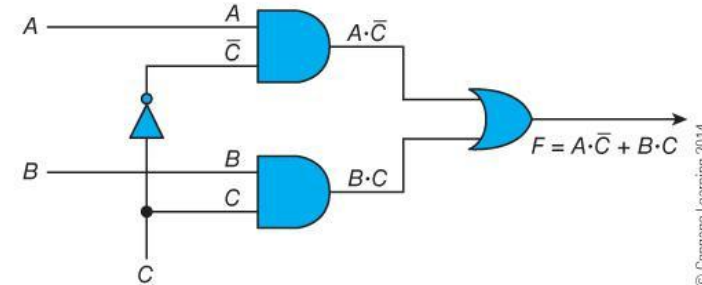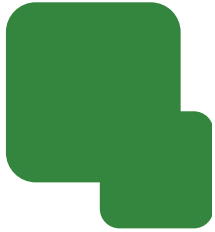
Circuit Simplification

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.

Rahul Bhadani

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.

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

Sum of products

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

Product of Sum

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Two-variable K-map using minterms
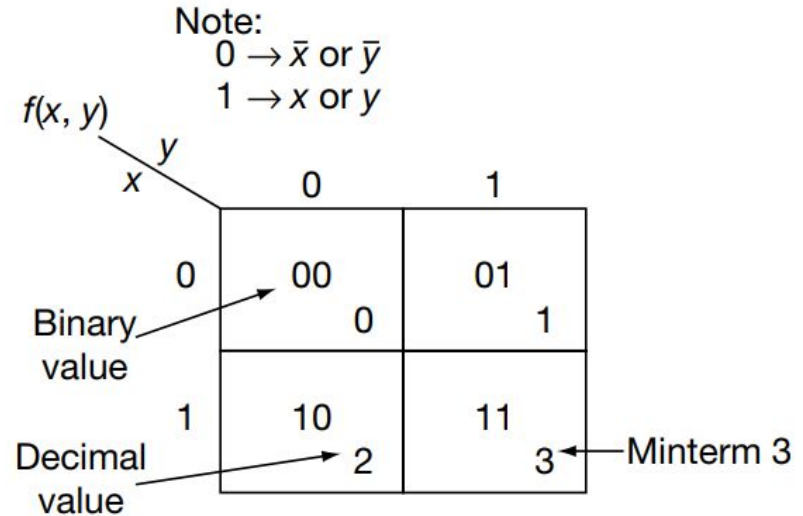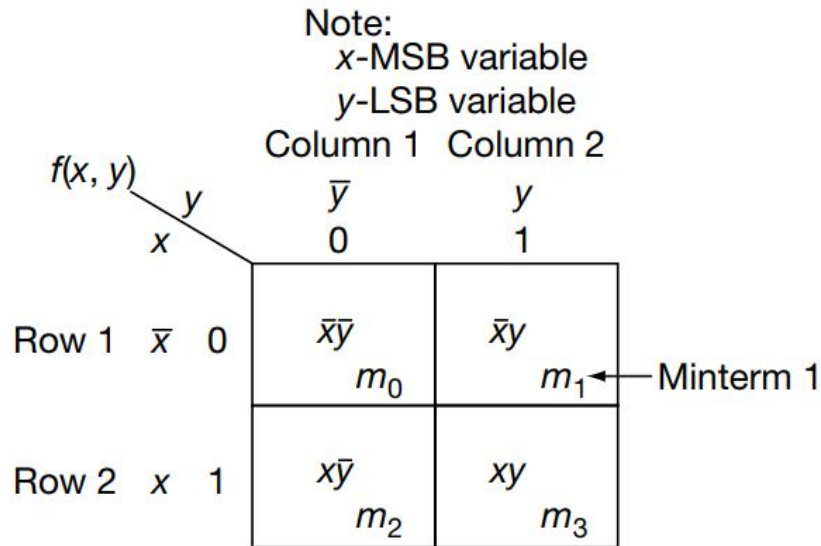
Two-variable K-map using maxterms

Note:
x-MSB variable
y-LSB variable

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

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

$f(x, y)$	y	0	1
x			
Binary value	0	00 0	01 1
Decimal value	1	10 2	11 3 ← Maxterm 3

Three-variable K-map

$f(x, y, z)$		Column 1 $\overline{y}\overline{z}$ 00	Column 2 $\overline{y}z$ 01	Column 3 yz 11	Column 4 $y\overline{z}$ 10
Row 1 $\overline{x}$	0	$\overline{x}\overline{y}\overline{z}$ m_0	$\overline{x}\overline{y}z$ m_1	$\overline{x}yz$ m_3	$\overline{x}y\overline{z}$ m_2
Row 2 x	1	$x\overline{y}\overline{z}$ m_4	$x\overline{y}z$ m_5	$xyzz$ m_7	$xy\overline{z}$ m_6

Note:
$z \rightarrow$ LSB variable
$x \rightarrow$ MSB variable

Note:
$0 \rightarrow \overline{x}$ or $\overline{y}$ or $\overline{z}$
$1 \rightarrow x$ or y or z

$m_6 \longleftarrow$ Minterm 6

$f(x, y, z)$		Column 1 $y + z$ 00	Column 2 $y + \overline{z}$ 01	Column 3 $\overline{y} + \overline{z}$ 11	Column 4 $\overline{y} + z$ 10
Row 1 x	0	$x + y + z$ M_0	$x + y + \overline{z}$ M_1	$x + \overline{y} + \overline{z}$ M_3	$x + \overline{y} + z$ M_2
Row 2 $\overline{x}$	1	$\overline{x} + y + z$ M_4	$\overline{x} + y + \overline{z}$ M_5	$\overline{x} + \overline{y} + \overline{z}$ M_7	$\overline{x} + \overline{y} + z$ M_6

Note:
$z \rightarrow$ LSB variable
$x \rightarrow$ MSB variable

Note:
$1 \rightarrow \overline{x}$ or $\overline{y}$ or $\overline{z}$
$0 \rightarrow x$ or y or z

$M_6 \longleftarrow$ Maxterm 6

Rahul Bhadani

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Four Variable K-map

Left K-map (minterms):

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

Note:
$z \rightarrow$ LSB variable
$w \rightarrow$ MSB variable

Note:
$0 \rightarrow \overline{w}$ or $\overline{x}$ or $\overline{y}$ or $\overline{z}$
$1 \rightarrow w$ or x or y or z

← Minterm 6

Right K-map (maxterms):

Note:
$z \rightarrow$ LSB variable $1 \rightarrow \overline{w}$ or $\overline{x}$ or $\overline{y}$ or $\overline{z}$
$w \rightarrow$ MSB variable $0 \rightarrow w$ or x or y or z

$f(w,x,y,z)$		Column 1 $y+z$ 00	Column 2 $y+\overline{z}$ 01	Column 3 $\overline{y}+\overline{z}$ 11	Column 4 $\overline{y}+z$ 10
Row 1	$w+x$ 00	$w+x+y+z$ M_0	$w+x+y+\overline{z}$ M_1	$w+x+\overline{y}+\overline{z}$ M_3	$w+x+\overline{y}+z$ M_2
Row 2	$w+\overline{x}$ 01	$w+\overline{x}+y+z$ M_4	$w+\overline{x}+y+\overline{z}$ M_5	$w+\overline{x}+\overline{y}+\overline{z}$ M_7	$w+\overline{x}+\overline{y}+z$ M_6
Row 3	$\overline{w}+\overline{x}$ 11	$\overline{w}+\overline{x}+y+z$ M_{12}	$\overline{w}+\overline{x}+y+\overline{z}$ M_{13}	$\overline{w}+\overline{x}+\overline{y}+\overline{z}$ M_{15}	$\overline{w}+\overline{x}+\overline{y}+z$ M_{14}
Row 4	$\overline{w}+x$ 10	$\overline{w}+x+y+z$ M_8	$\overline{w}+x+y+\overline{z}$ M_9	$\overline{w}+x+\overline{y}+\overline{z}$ M_{11}	$\overline{w}+x+\overline{y}+z$ M_{10}

Maxterm 8 →

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

SOLUTION Set up K-map as given in Figure 4.36. Fill 1 for each minterm. There are two quads. One quad is formed by grouping (0, 1, 3, 2) cells and on rolling over K-map another quad is formed by grouping (0, 4, 2, 6) cells/square. These quads are known as prime-implicants as well as essential prime-implicants.

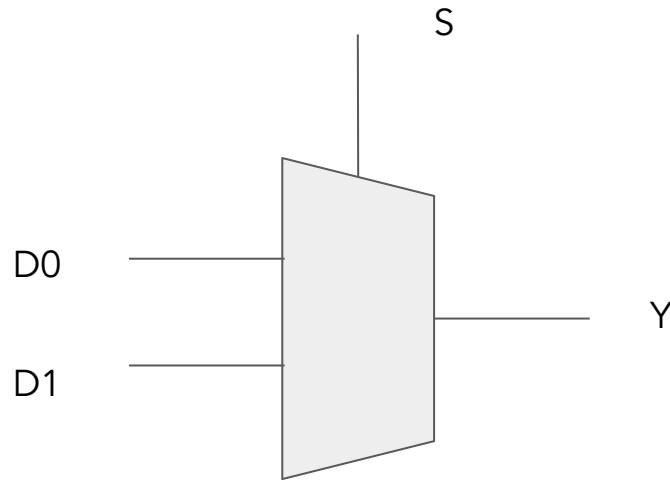*Overlapping:* There is overlapping of Cell 0 and 2 in forming quad (0, 4, 2, 6).
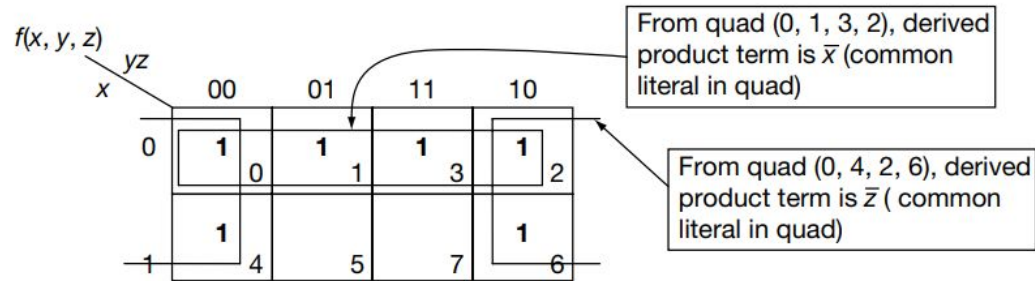
Combine derived literal by using OR operator, to get:

$$f(x, y, z) = \sum m(0, 1, 2, 3, 4, 6) = \bar{x} + \bar{z}$$

Multiplexers: Minimizing a Circuit

They choose an output from several possible inputs based on the value of select signal.
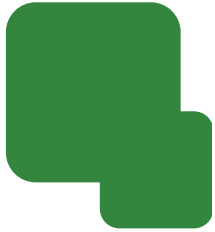
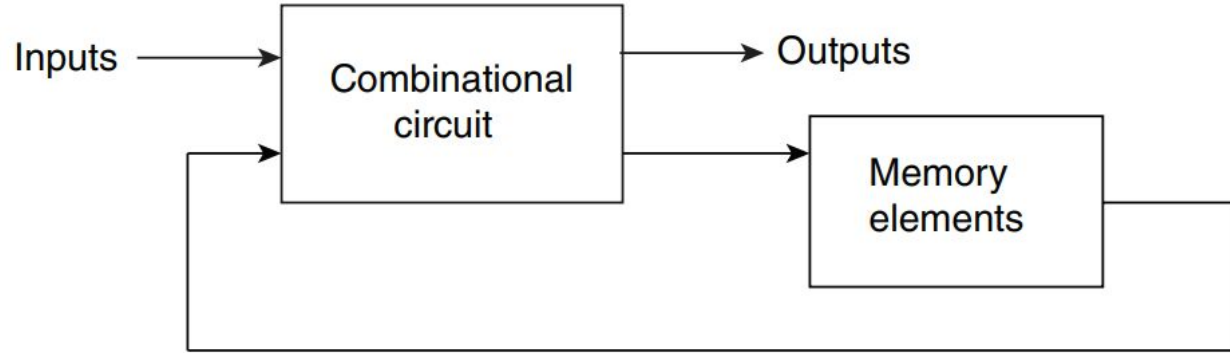if S = 0

    Y = D0

if S = 1

    Y = D1

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

Rahul Bhadani
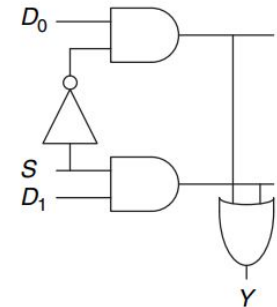
Sequential Circuits

Sequential Circuits from Combination Circuits

Clock

(a) Response to positive level

(b) Positive-edge response

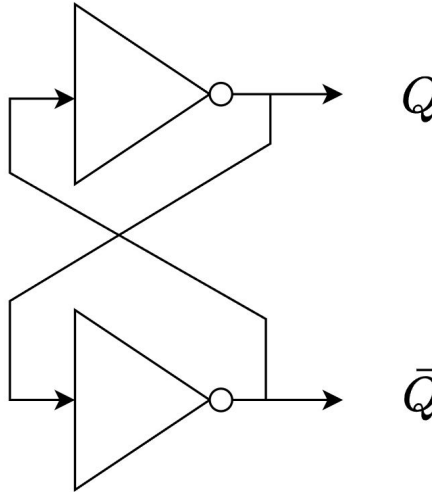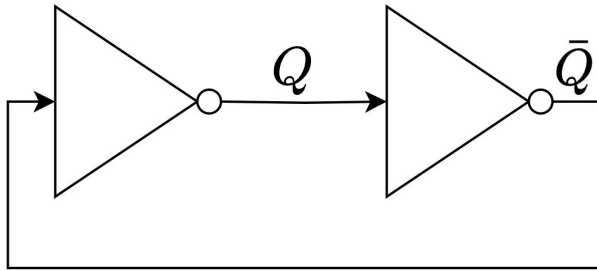(c) Negative-edge response

Copyright ©2013 Pearson Education, publishing as Prentice Hall

Rahul Bhadani

Latch

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

Rahul Bhadani

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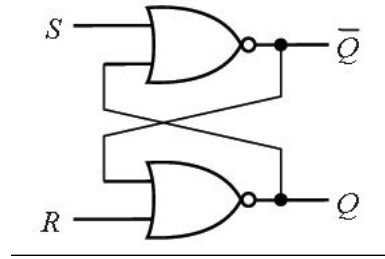
| Input $\bar{S}$	Input $\bar{R}$	Output Q
1	1	Previous State
1	0	0
0	1	1
0	0	0 (Invalid)

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 kΩ resistors (R1 and R2)
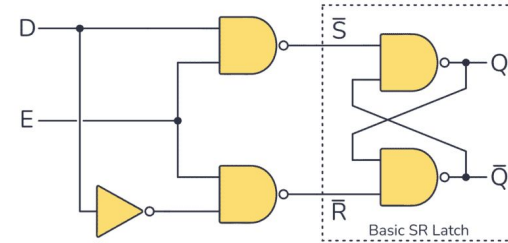- Two 330 Ω resistors (R3 and R4)
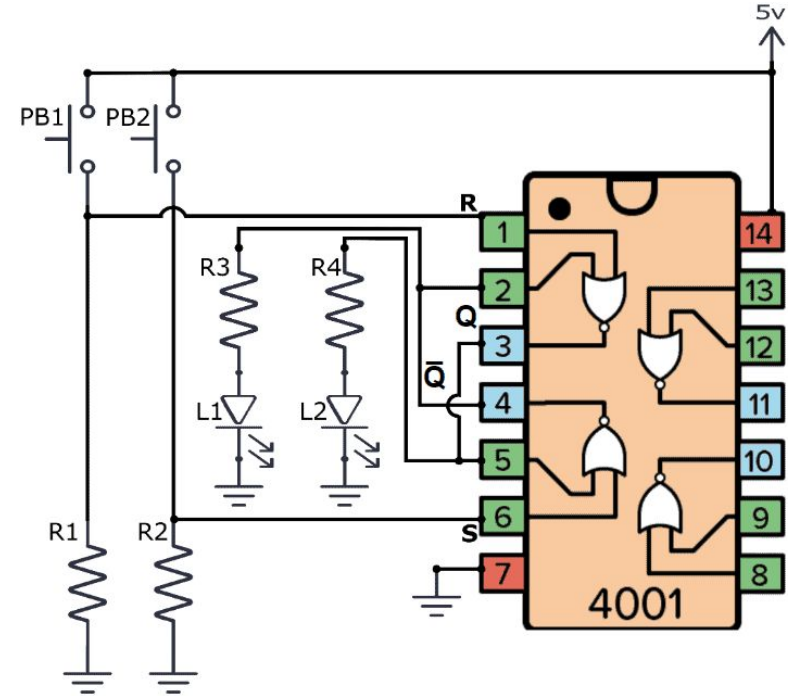
D-Latch

- 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.
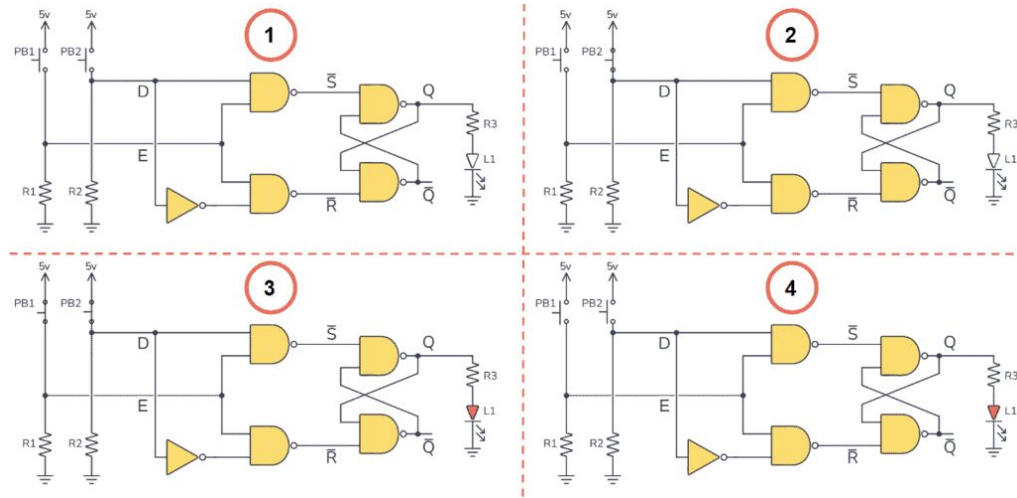
- Function
  - When CLK = 1
    - D passes through to Q (transparent)
  - When CLK = 0
    - Q holds its previous value (opaque)
  - Avoids invalid case when Q ≠ NOT Q

Basic SR Latch

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

Rahul Bhadani

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

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.

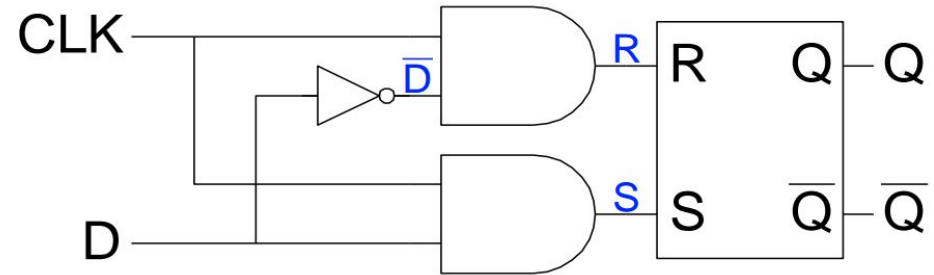4: PB1 and PB2 return to their original states in section 4, LED L1 remains ON indicating that the Q output has not changed.

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 kΩ resistors (R1 and R2)
- 1x 330 Ω 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 ≠ NOT Q

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

Rahul Bhadani

Practice with a D Latch

A   B   C   D   E   F   G   H   I   J   K   L

CLK

IN

Q1

Spring
2025

Rahul Bhadani
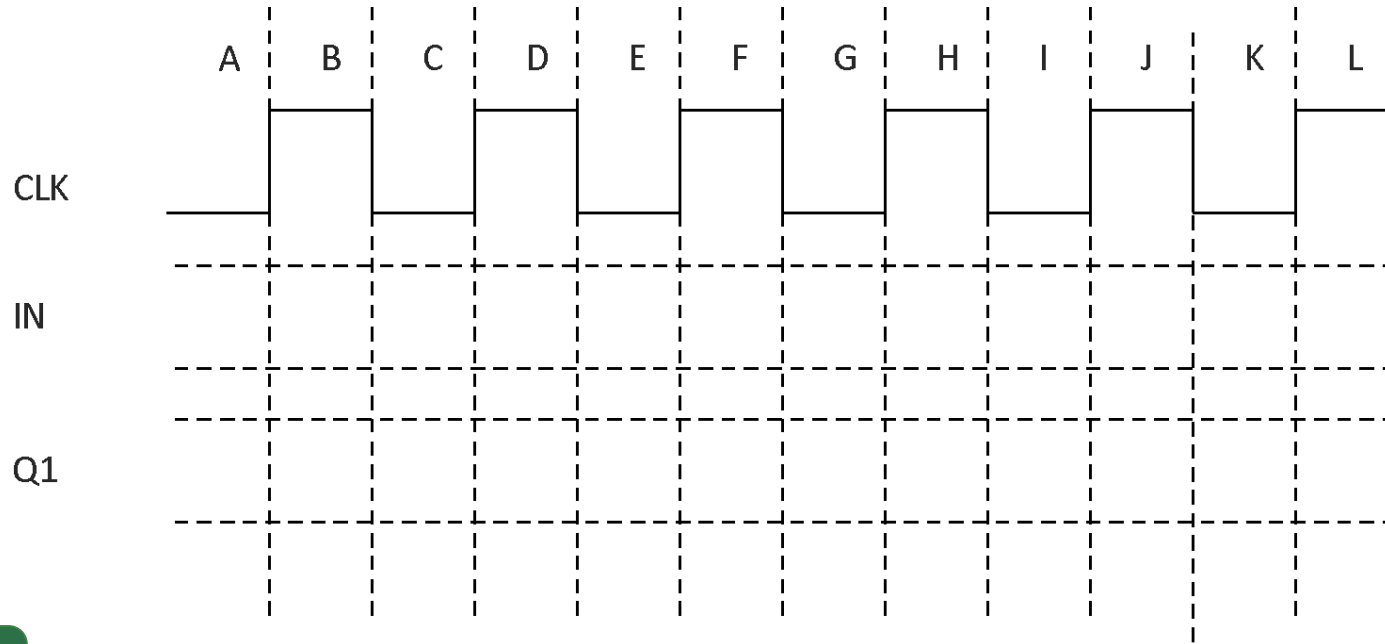
Flip Flops

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

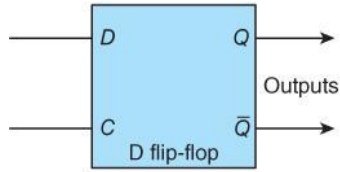FIGURE 2.36   The clocked RS flip-flop
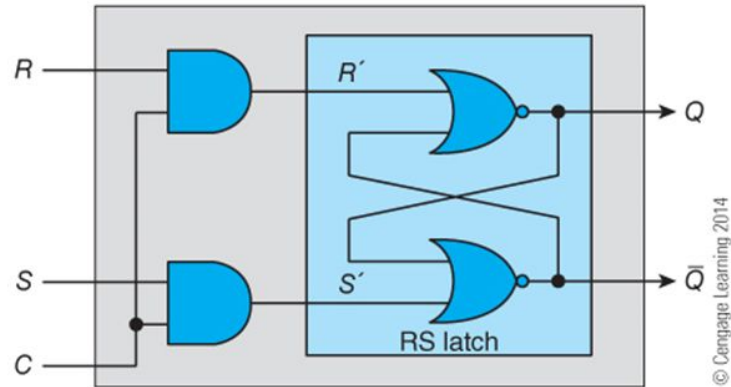
© Cengage Learning 2014
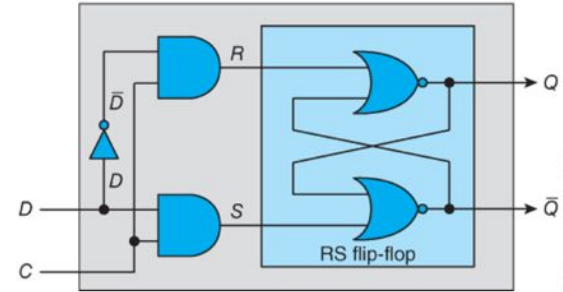
The D Flip-Flop

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

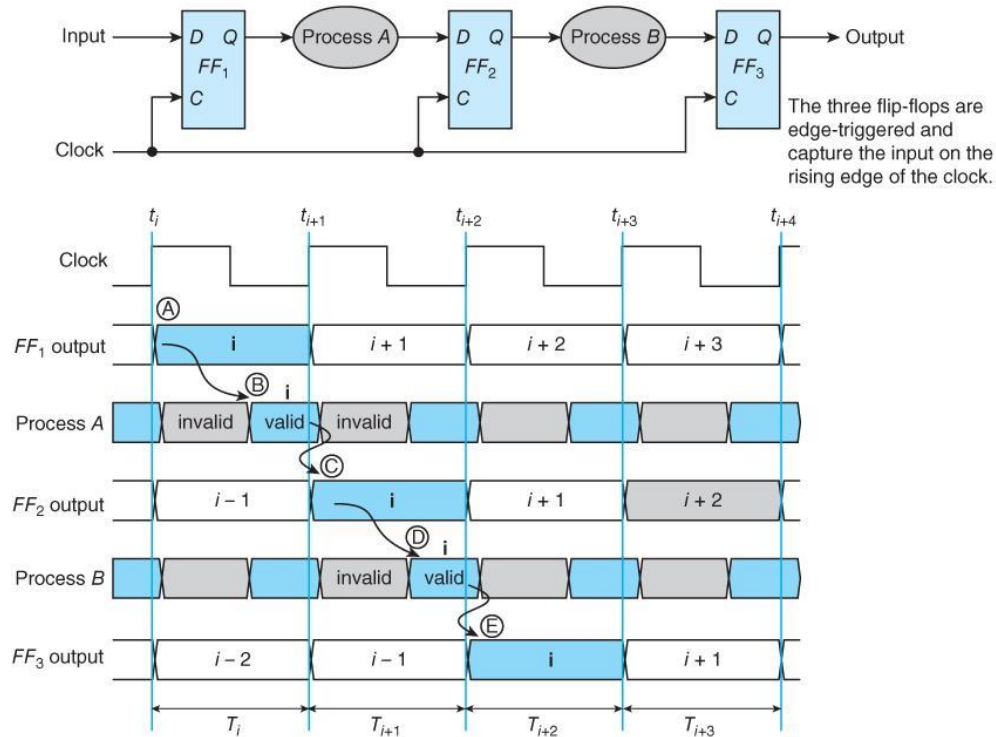# Practice with a D Flip-Flop
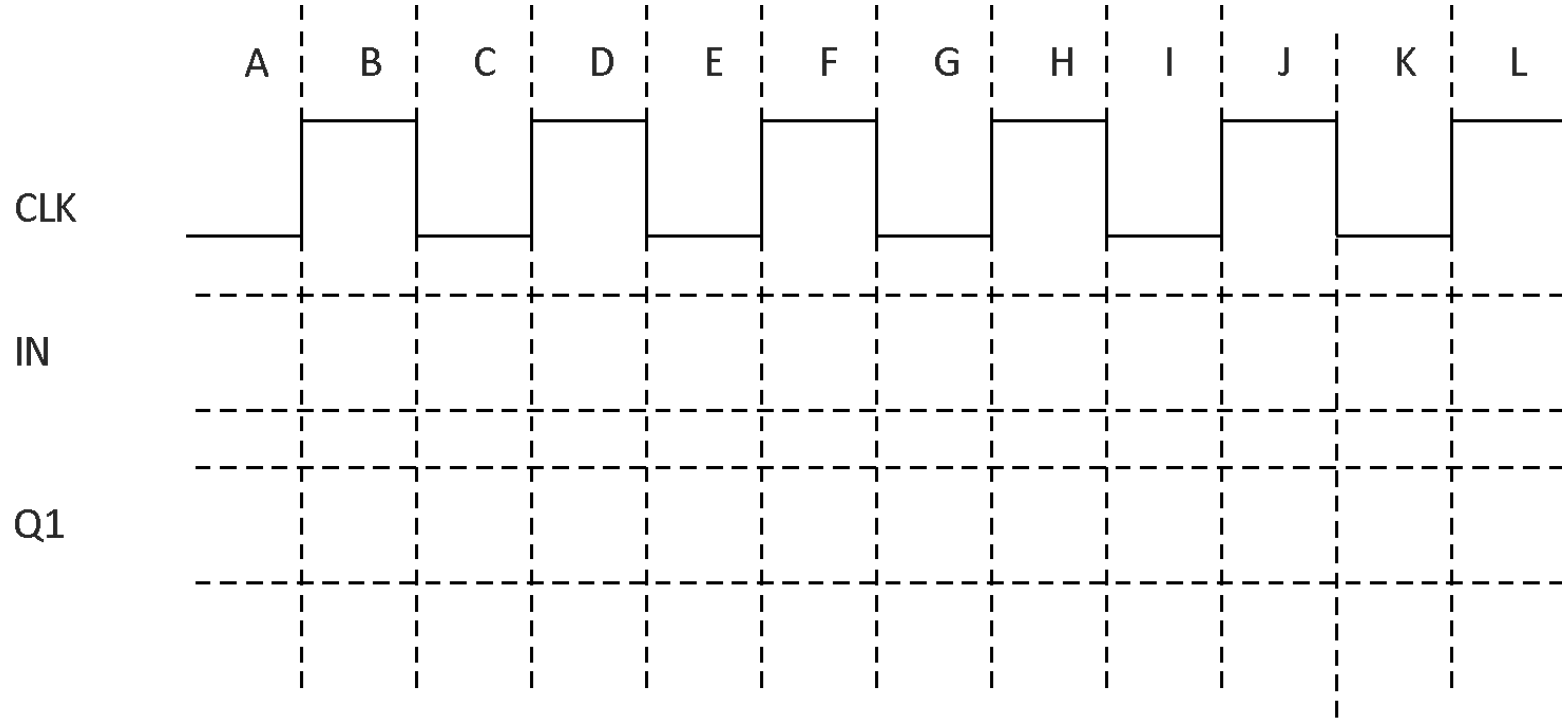
Timing in Sequential Circuits

FIGURE 2.40   Capturing the output of a system
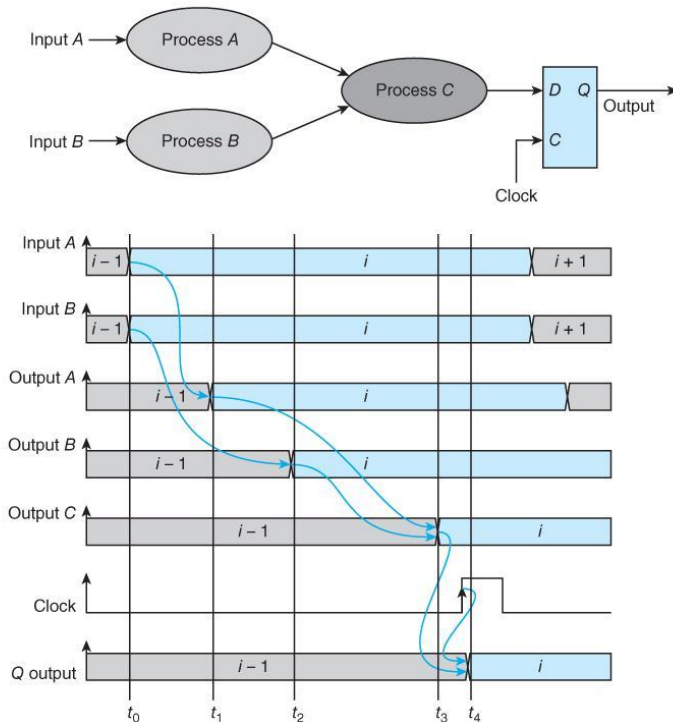
© Cengage Learning 2014

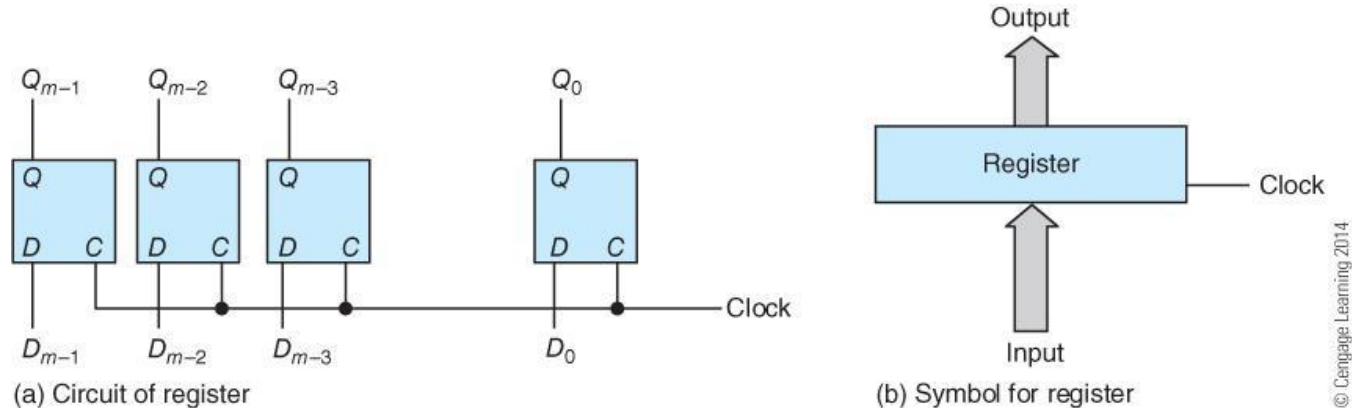# Timing in Sequential Circuits (Pipelining)

FIGURE 2.41 Pipelining using flip-flops

The three flip-flops are edge-triggered and capture the input on the rising edge of the clock.

© Cengage Learning 2014

Registers

FIGURE 2.44 The register

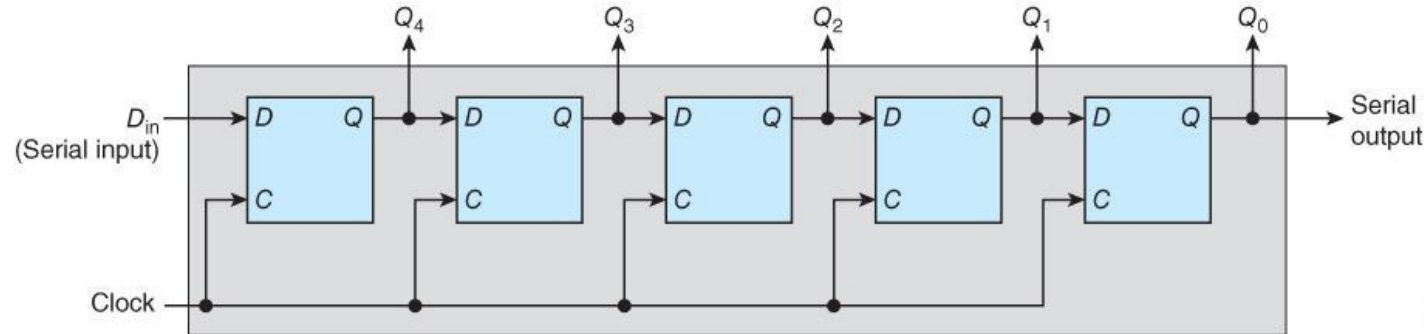(a) Circuit of register

(b) Symbol for register

© Cengage Learning 2014

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.
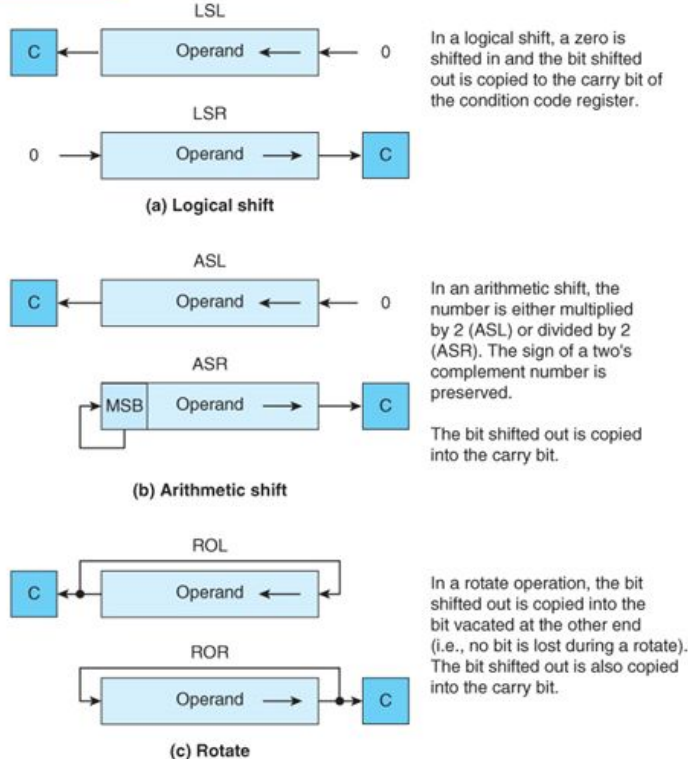

FIGURE 2.45 The shift register

© Cengage Learning 2014

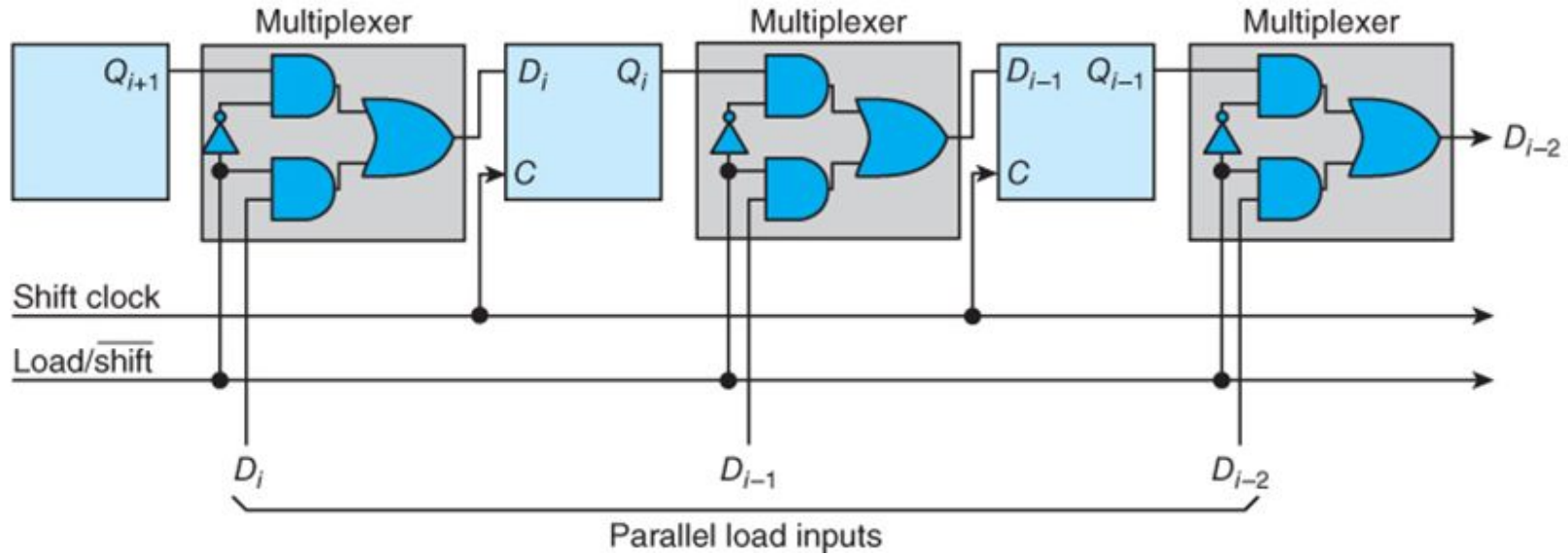# Other Shift operations

FIGURE 3.23   Shift operations

LSL

In a logical shift, a zero is shifted in and the bit shifted out is copied to the carry bit of the condition code register.

LSR

(a) Logical shift

ASL

In an arithmetic shift, the number is either multiplied by 2 (ASL) or divided by 2 (ASR). The sign of a two's complement number is preserved.

The bit shifted out is copied into the carry bit.

ASR

(b) Arithmetic shift

ROL

In a rotate operation, the bit shifted out is copied into the bit vacated at the other end (i.e., no bit is lost during a rotate). The bit shifted out is also copied into the carry bit.

ROR

(c) Rotate

© Cengage Learning 2014

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

Rahul Bhadani

THE UNIVERSITY OF
ALABAMA IN HUNTSVILLE

Parallel Load Capacity

FIGURE 2.48 Shift register with a parallel load capability

Multiplexer Q_{i+1} D_i Q_i C Multiplexer D_{i-1} Q_{i-1} C Multiplexer D_{i-2}

Shift clock

Load/$\overline{\text{shift}}$

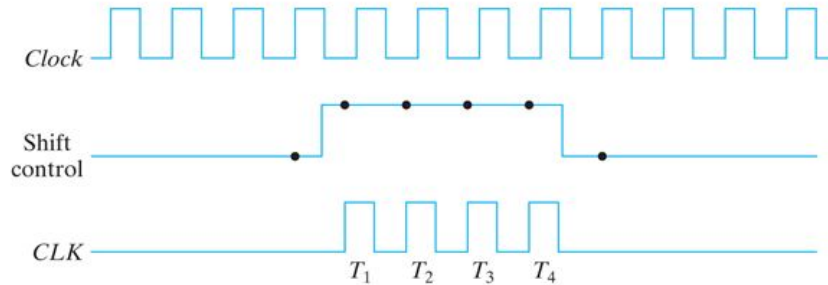D_i D_{i-1} D_{i-2}

Parallel load inputs

© Cengage Learning 2014

Serial Input/ Serial Output
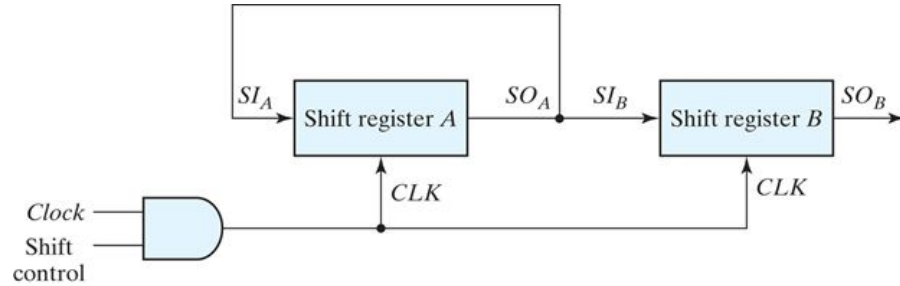
(a) Block diagram

(b) Timing diagram

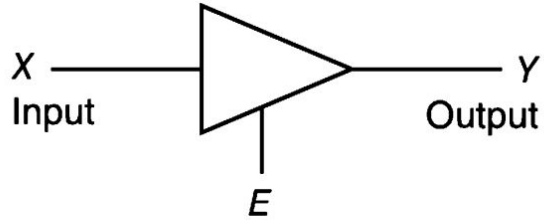# Serial Adder
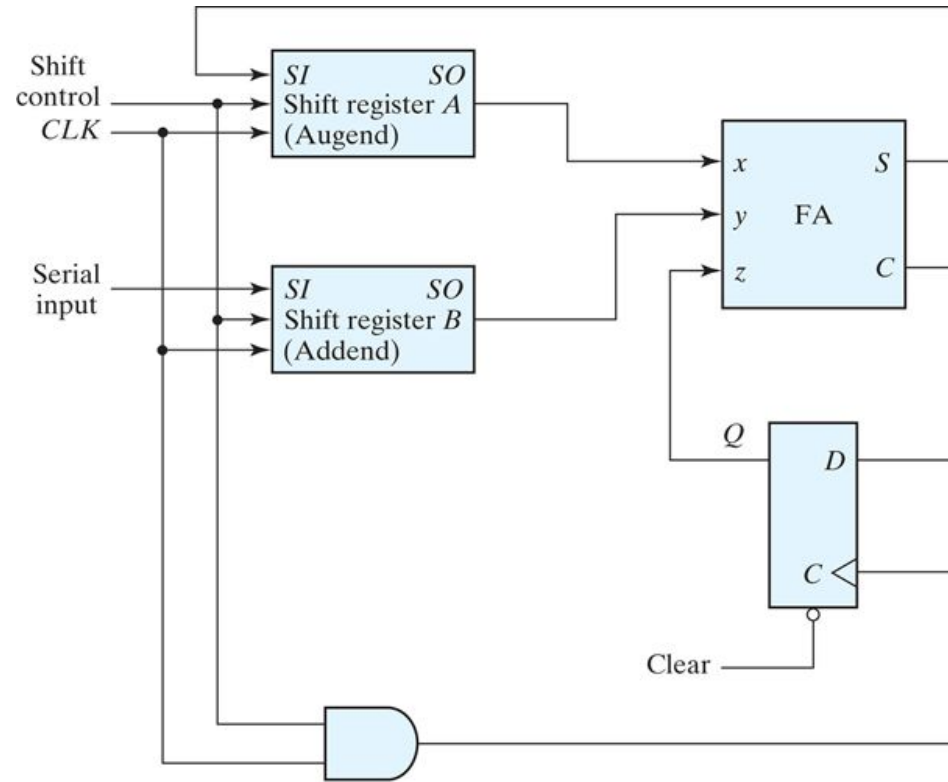
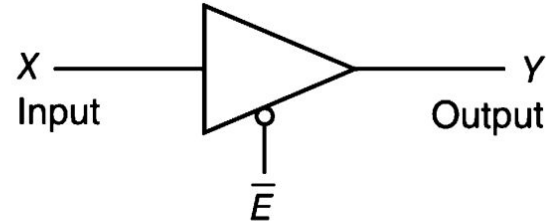Copyright ©2013 Pearson Education, publishing as Prentice Hall

Tristate Gates
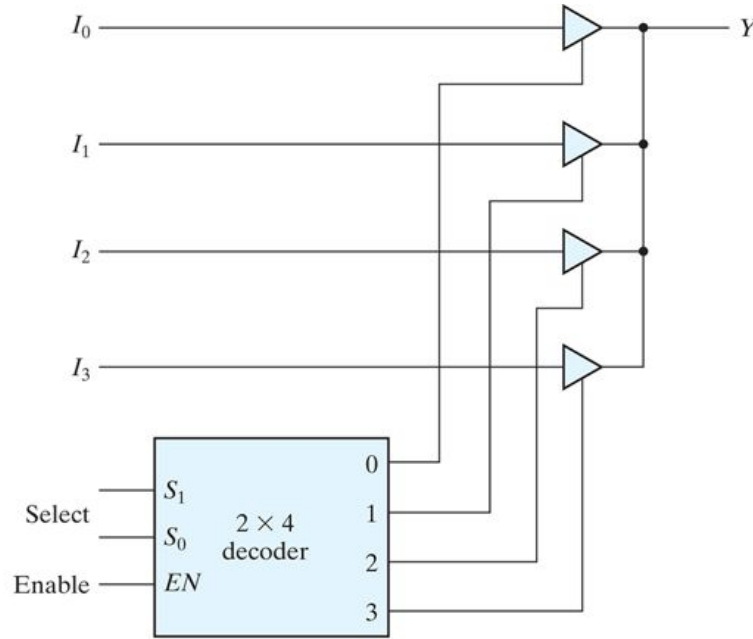

(a) Non-inverting buffer with active-high enable

(b) Non-inverting buffer active-low enable
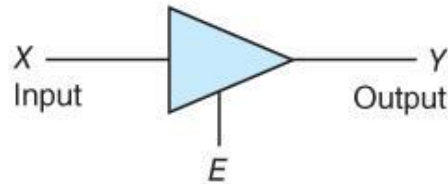
Tristate Buffers
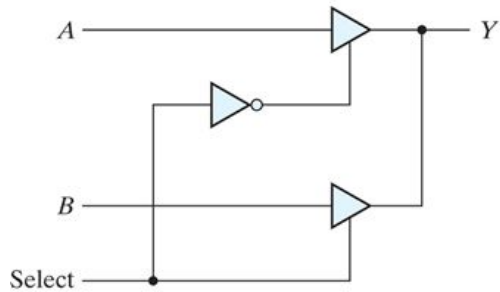
(a) 2-to-1-line mux

(b) 4-to-1-line mux

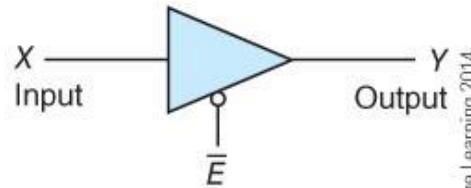Copyright ©2013 Pearson Education, publishing as Prentice Hall

Rahul Bhadani

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.
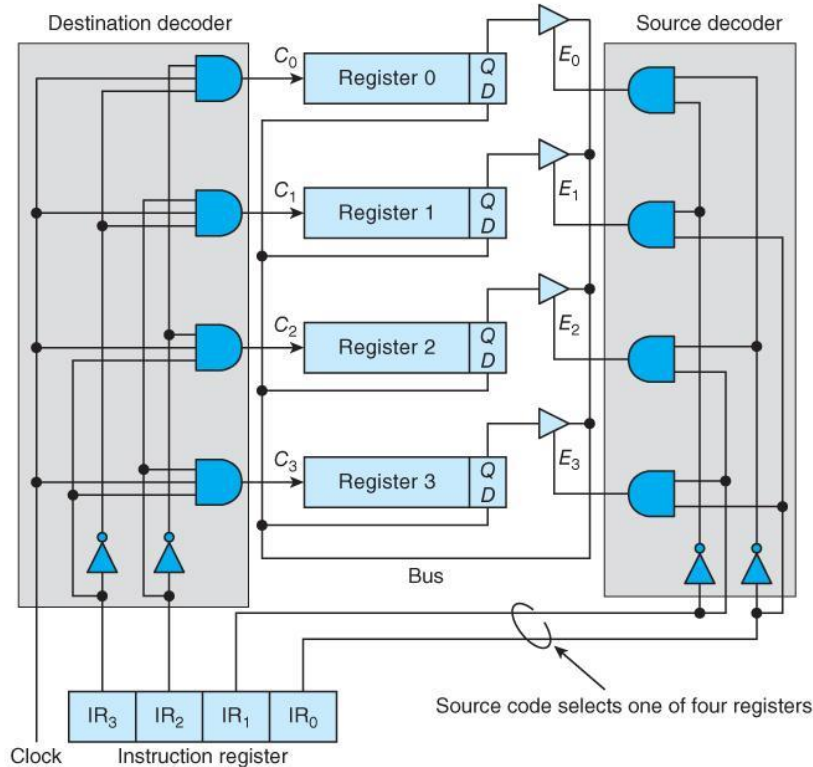
E0 - E3 would come from a decoder
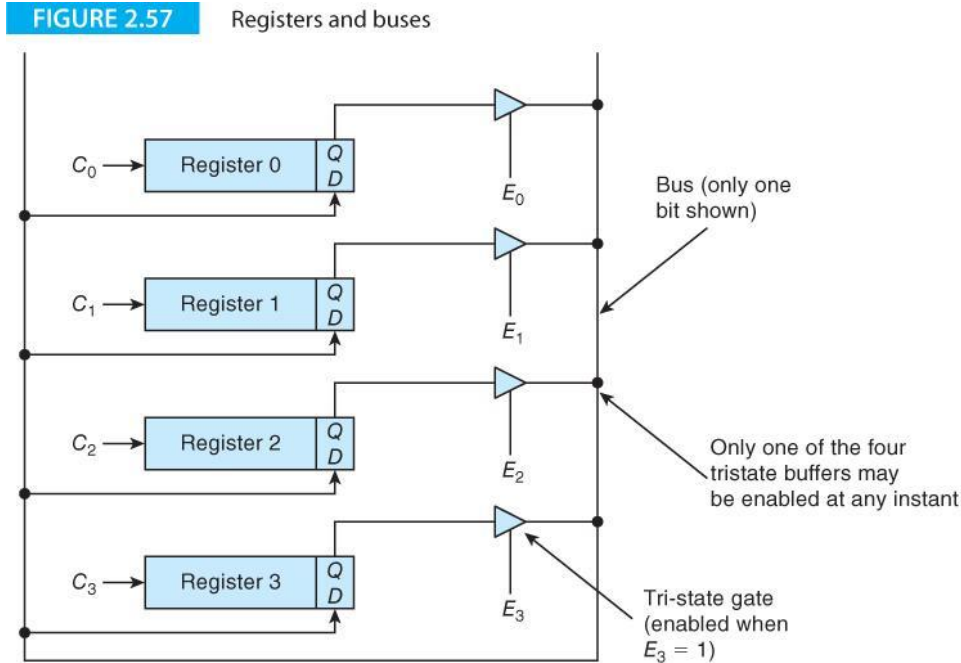
FIGURE 2.57   Registers and buses

Implementing a MOVE (Copy) Instruction

FIGURE 2.58   Controlling the bus

Multiple Buses and an ALU

FIGURE 2.59   The registers, buses, and ALU

Rahul Bhadani
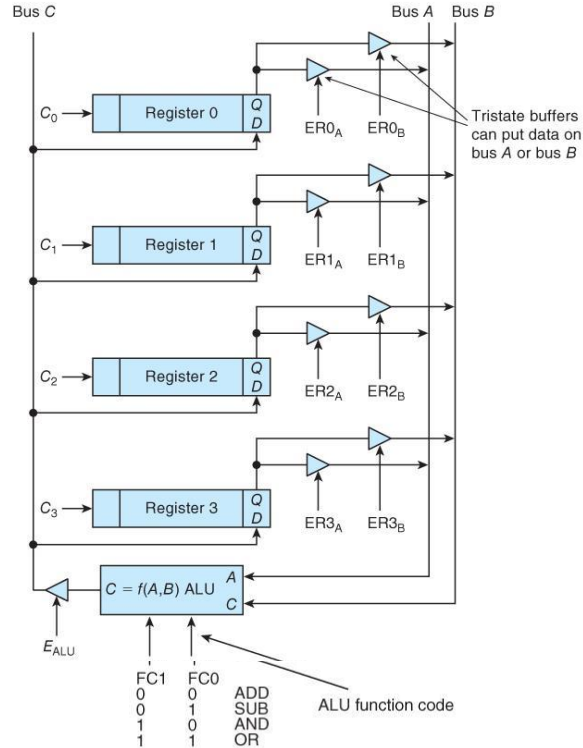
The History of the Integrated Circuit

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

Rahul Bhadani

THE UNIVERSITY OF
ALABAMA IN HUNTSVILLE