Chapter 1: Digital Systems and Binary Numbers

1.1 Digital Systems

1.2 Binary Numbers

1.3 Number-Base Conversions

1.4 Octal and Hexadecimal Numbers

1.5 Complements of Numbers

1.6 Signed Binary Numbers

1.7 Binary Codes

1.8 Binary Storage and Registers

1.9 Binary Logic

Digital Systems

- Digital age:
  - digital systems play a prominent role in everyday life; used in communication, business transactions, traffic control, spacecraft guidance, medical treatment, weather monitoring, the Internet, and many other commercial, industrial, and scientific enterprises
- Digital system examples:
  - mobile phone
  - digital camera
  - digital versatile disc (DVD), personal digital assistance (PDA), MP3, …
  - digital TV
  - most, if not all, of these devices have an embedded ***special-purpose digital computer***
- Digital computers:

  What you will learn in the micro-processor course.
  - generality or flexibility
  - users can specify and change the ***program***, a sequence of instructions, or the data according to the specific need
  - ***general-purpose*** digital computers can perform a variety of information-processing tasks that range over a wide spectrum of applications

Signal

- Digital systems manipulate discrete elements of information represented in a digital system by physical quantities called signals.
- ***Voltages*** and ***currents*** are the most common electrical signals.
- The signals in most present-day electronic digital systems use just two discrete values or ***binary*** values
- Binary values are represented abstractly by:
  - digit: 0 and 1 ➔ A binary digit is called a ***bit.***
  - logic: False (F) and True (T)
  - level: Low (L) and High (H)
  - word: Yes and No
  - state: On and Off.
- Discrete elements of information are represented with groups of bits called ***binary codes***. For example: $0111_2 = 7_{10}$
- *Analog-to-digital converters* perform *quantization* process on analog (continuous) signals to form a digital (discrete) signals.

General-purpose Digital Computer

- The major parts of a computer are a *memory* unit, a *central processing unit* (*CPU*), and *input–output* units (*I/O*). [check these in your *Computer* course]
- Memory unit stores programs as well as input, output, and intermediate data.
- CPU performs arithmetic and data-processing operations as specified by the program.
- The program and data prepared by a user are transferred into memory by means of an input device such as a keyboard.
- An output device, such as a printer, receives the results of the computations, and the printed results are presented to the user.
- Voice, image, finger touch, gesture, brain wave, …, also can be used as input or output.
- Communication unit provides interaction with other users through the Internet.
- A digital computer can perform not only arithmetic computations and logical operations but also can make decisions based on internal and external conditions.

Hardware Description Language (HDL)

- A digital system is an interconnection of digital *modules*.
- To understand the operation of each digital module, it is necessary to have a basic knowledge of digital circuits and their logical function.
- A major trend in digital design methodology is the use of a ***hardware description language*** (***HDL***) to describe and *simulate* the functionality of a digital circuit.
- An HDL resembles a programming language and is suitable for describing digital circuits in textual form, simulating a digital system to *verify* its operation before hardware is built, and automating the design process in conjunction with *logic synthesis* tools.
- Ignorance of industry's practices on HDL modeling will lead to cute, but worthless, HDL models that may simulate a phenomenon, but that cannot be synthesized by design tools, or to models that waste silicon area or synthesize to hardware that cannot operate correctly. [take the *Digital System Design* course for more learning on HDL]

Decimal Numbers

- Decimal number (*base*-10 or *radix*-10)

$$\cdots a_5 a_4 a_3 a_2 a_1 a_0 . a_{-1} a_{-2} a_{-3} \cdots$$

Decimal point

a_j — digit; j — Power

$$\cdots + 10^5 a_5 + 10^4 a_4 + 10^3 a_3 + 10^2 a_2 + 10^1 a_1 + 10^0 a_0 + 10^{-1} a_{-1} + 10^{-2} a_{-2} + 10^{-3} a_{-3} + \cdots$$

Example:

$$7,329 = 7 \times 10^3 + 3 \times 10^2 + 2 \times 10^1 + 9 \times 10^0$$

- General form of *base-r* system

$$a_n \cdot r^n + a_{n-1} \cdot r^{n-1} + \cdots + a_2 \cdot r^2 + a_1 \cdot r^1 + a_0 + a_{-1} \cdot r^{-1} + a_{-2} \cdot r^{-2} + \cdots + a_{-m} \cdot r^{-m}$$

Coefficient: $a_j = 0$ to $r - 1$

Base-r Numbers

Example: Base-2, *binary*

$$(11010.11)_2 = (26.75)_{10}$$

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

Example: Base-5

$$(4021.2)_5$$

$$= 4\times 5^3 + 0\times 5^2 + 2\times 5^1 + 1\times 5^0 + 2\times 5^{-1} = (511.5)_{10}$$

Example: Base-8, *octal*

$$(127.4)_8$$

$$= 1\times 8^3 + 2\times 8^2 + 1\times 8^1 + 7\times 8^0 + 4\times 8^{-1} = (87.5)_{10}$$

Example: Base-16, *hexadecimal*, (A, B, C, D, E, and F are used for the digits 10 ~ 15, respectively)

$$(\mathrm{B65F})_{16} = 11\times 16^3 + 6\times 16^2 + 5\times 16^1 + 15\times 16^0 = (46,687)_{10}$$

Binary to Decimal Conversion

Example:

$$(110101)_2 = 32 + 16 + 4 + 1 = (53)_{10}$$

Special Powers of 2

- $2^{10} \approx 10^3$ (1024) is *Kilo*, denoted “K”
- $2^{20} \approx 10^6$ (1,048,576) is *Mega*, denoted “M”
- $2^{30} \approx 10^9$ (1,073,741,824) is *Giga*, denoted “G”
- $2^{40} \approx 10^{12}$ (1,099,511,627,776) is *Tera*, denoted “T”

Powers of Two

Table 1.1
Powers of Two

n	2^n	n	2^n	n	2^n
0	1	8	256	16	65,536
1	2	9	512	17	131,072
2	4	10	1,024 (1K)	18	262,144
3	8	11	2,048	19	524,288
4	16	12	4,096 (4K)	20	1,048,576 (1M)
5	32	13	8,192	21	2,097,152
6	64	14	16,384	22	4,194,304
7	128	15	32,768	23	8,388,608

- A *byte* is equal to eight bits and usually used in computer capacity.
- A computer hard disk with four gigabytes of storage has a capacity of $4G = 2^{32}$ bytes (approximately 4 billion bytes).
- A terabyte is 1024 gigabytes, approximately 1 trillion bytes.

Binary Arithmetic Operations

- **Addition**

Augend:	101101
Addend:	+100111
Sum:	1010100

- **Subtraction**

Minuend:	101101
Subtrahend:	–100111
Difference:	000110

- **Multiplication**

The binary multiplication table is simple:

$$0 \times 0 = 0,\ \ 1 \times 0 = 0,\ \ 0 \times 1 = 0,\ \ 1 \times 1 = 1$$

Extending multiplication to multiple digits:

Multiplicand	1011
Multiplier	× 101
Partial Products	1011
	0000-
	1011--
Product	110111

Number-Base Conversion

Name	Radix	Digits
Binary	2	0,1
Octal	8	0,1,2,3,4,5,6,7
Decimal	10	0,1,2,3,4,5,6,7,8,9
Hexadecimal	16	0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F

- The six letters A, B, C, D, E, and F in hexadecimal represent: 10, 11, 12, 13, 14, and 15, respectively.

- Base-*r* to decimal conversion is done by expanding the number in a power series and adding all the terms.
- Decimal to base-*r* conversion is done by dividing the number and all successive quotients by *r* and accumulating the remainders.

Decimal to Binary Conversion

Example1.1

Convert decimal 41 to binary. The quotient-divided-2 process is continued until the integer quotient becomes 0.

	Integer Quotient	Remainder	Coefficient
41/2 =	20	1	$a_0 = 1$
20/2 =	10	0	$a_1 = 0$
10/2 =	5	0	$a_2 = 0$
5/2 =	2	1	$a_3 = 1$
2/2 =	1	0	$a_4 = 0$
1/2 =	0	1	$a_5 = 1$

$$(41)_{10} = (a_5a_4a_3a_2a_1a_0)_2 = (101001)_2$$

Long Division

♣ The arithmetic process can be manipulated more conveniently as follows:

÷ 2

Integer	Remainder
41	
20	1
10	0
5	0
2	1
1	0
0	1

$41 = 20 \times 2 + 1$

$= (10 \times 2 + 0) \times 2 + 1$

$= ((5 \times 2 + 0) \times 2 + 0) \times 2 + 1$

$= (((2 \times 2 + 1) \times 2 + 0) \times 2 + 0) \times 2 + 1$

$= ((((1 \times 2 + 0) \times 2 + 1) \times 2 + 0) \times 2 + 0) \times 2 + 1$

$= (((1 \times 2 + 0) \times 2 + 1) \times 2 + 0) \times 2^2 + 0 \times 2^1 + 1 \times 2^0$

$= ((1 \times 2 + 0) \times 2 + 1) \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$

$= (1 \times 2 + 0) \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$

$= 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$

101001 = answer

Decimal to Octal Conversion

Example 1.2

Convert decimal 153 to octal. The required base *r* is 8.

Integer	Remainder
153	
19	1
2	3
0	2

$= (231)_8$

$$153 = 19 \times 8 + 1$$
$$= (2 \times 8 + 3) \times 8 + 1$$
$$= 2 \times 8^2 + 3 \times 8^1 + 1 \times 8^0$$

Decimal Fraction Number to Binary

Example1.3

Convert $(0.6875)_{10}$ to binary. The multiplied-by-2 process is continued until the fraction becomes 0 or until the number of digits has sufficient accuracy.

	Integer		Fraction	Coefficient
$0.6875 \times 2 =$	1	+	0.3750	$a_{-1} = 1$
$0.3750 \times 2 =$	0	+	0.7500	$a_{-2} = 0$
$0.7500 \times 2 =$	1	+	0.5000	$a_{-3} = 1$
$0.5000 \times 2 =$	1	+	0.0000	$a_{-4} = 1$

$$(0.6875)_{10} = (0.a_{-1}a_{-2}a_{-3}a_{-4})_2 = (0.1011)_2$$

- To convert a decimal fraction to a base-*r* number, a similar procedure is used. However, multiplication is by *r* instead of 2, and the coefficients found from the integers may range in value from 0 to $r - 1$ instead of 0 and 1.

Decimal Fraction Number to Octal

Example1.4

Convert $(0.513)_{10}$ to octal.

$$0.513 \times 8 = 4.104$$
$$0.104 \times 8 = 0.832$$
$$0.832 \times 8 = 6.656$$
$$0.656 \times 8 = 5.248$$
$$0.248 \times 8 = 1.984$$
$$0.984 \times 8 = 7.872$$

$$(0.513)_{10} = (0.406517\ldots)_8$$

- The conversion of decimal numbers with both integer and fraction parts is done by converting the integer and the fraction separately and then combining the two answers.
  - ♣ **From Examples 1.1 and 1.3:** $(41.6875)_{10} = (101001.1011)_2$
  - ♣ **From Examples 1.2 and 1.4:** $(153.513)_{10} = (231.406517)_8$

Numbers with Different Bases

♣ $2^3 = 8$ & $2^4=16$

→each octal/hexadecimal digit corresponds to three/four binary digits

Table 1.2
Numbers with Different Bases

Decimal (base 10)	Binary (base 2)	Octal (base 8)	Hexadecimal (base 16)
00	0000	00	0
01	0001	01	1
02	0010	02	2
03	0011	03	3
04	0100	04	4
05	0101	05	5
06	0110	06	6
07	0111	07	7
08	1000	10	8
09	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

Octal and Hexadecimal Numbers

$2^3 = 8, 2^4 = 16$

- Conversion from binary to octal can be done by positioning the binary number into groups of **three** digits each, starting from the binary point and proceeding to the left and to the right.

$(10$	110	001	101	011	$\cdot$	111	100	000	$110)_2 = (26153.7406)_8$	
2	6	1	5	3		7	4	0	6	

- Conversion from binary to hexadecimal is similar, except that the binary number is divided into groups of **four** digits:

$(10$	1100	0110	1011	$\cdot$	1111	$0010)_2 = (2C6B.F2)_{16}$	
2	C	6	B		F	2	

- Conversion from octal or hexadecimal to binary is done by reversing the preceding procedure.

$(673.124)_8 = (110$	111	011	$\cdot$	001	010	$100)_2$	
	6	7	3		1	2	4

$(306.D)_{16} = (0011$	0000	0110	$\cdot$	$1101)_2$	
	3	0	6	D	

Complements of Numbers

- Complements are used in digital systems to simplify the subtraction operation
- There are two types of complements for each base-r system: the radix complement and diminished radix complement.

  ⟹ the r's complement and the second as the $(r-1)$'s complement.

Diminished Radix Complement

Given a number N in case r having n digits, the $(r-1)$'s complement of N is defined as $(r^n - 1) - N$. For decimal numbers, $r = 10$ and $r - 1 = 9$, so the 9's complement of N is $(10^n - 1) - N$.

Example:

The 9's complement of 546700 is $999999 - 546700 = 453299$.

The 9's complement of 012398 is $999999 - 012398 = 987601$.

- For binary numbers, $r = 2$ and $r - 1 = 1$, so the 1's complement of N is $(2^n - 1) - N$.

Example:

The 1's complement of 1011000 is 0100111

The 1's complement of 0101101 is 1010010

Complements of Numbers

Radix Complement

The r's complement of an n-digit number N in base r is defined as $r^n - N$ for $N \neq 0$ and as 0 for $N = 0$. Comparing with the $(r - 1)$'s complement, we note that the r's complement is obtained by adding 1 to the $(r - 1)$'s complement, since $r^n - N = [(r^n - 1) - N] + 1$.

Example: Base-10

The 10's complement of 012398 is 987602
The 10's complement of 246700 is 753300

Example: Base-2

The 2's complement of 1101100 is 0010100
The 2's complement of 0110111 is 1001001

Subtraction with Complements

- The subtraction of two n-digit unsigned numbers $M - N$ in base r can be done as follows:
  - Add the minuend M to the r's complement of the subtrahend N.

    → $M + (r^n - N) = M - N + r^n$
  - $M \geq N$ → the sum will produce an end carry r^n, which can be discarded, what is left is the result M-N
  - $M < N$ → the sum does not produce an end carry, and is equal to $r^n - (N - M)$

10’s Complement Subtraction

Example 1.5

Using 10's complement, subtract 72532 – 3250.

	$M =$	72532
10's complement of	$N =$	$+\,96750$
	Sum =	169282
	Discard end carry 10^5 =	$-\,100000$
	Answer =	69282

Example 1.6

Using 10's complement, subtract 3250 – 72532

	M =	03250
10's complement of	N =	$+\,27468$
	Sum =	30718

⇒ There is no end carry.

⇒ Therefore, the answer is – (10's complement of 30718) = – 69282.

2's Complement Subtraction

Example 1.7

Given the two binary numbers $X = 1010100$ and $Y = 1000011$, perform the subtraction (a) X – Y and (b) $Y - X$ by using 2's complement.

(a)		
	X =	1010100
	2's complement of Y =	+0111101
	Sum =	10010001
	Discard end carry 2^7 =	–10000000
	Answer. X – Y =	0010001

(b)		
	Y =	1000011
	2's complement of X =	+ 0101100
	Sum =	1101111

There is no end carry. Therefore, the answer is $Y - X$ = – (2's complement of 1101111) = – 0010001.

1's Complement Subtraction

- Subtraction of unsigned numbers can also be done by means of the $(r-1)$'s complement. Remember that the $(r-1)$'s complement is one less then the r's complement.

Example 1.8

Repeat Example 1.7, but this time using 1's complement.

(a) $X - Y = 1010100 - 1000011$

$X =$	1010100
1's complement of $Y =$	± 0111100
Sum =	10010000
End-around carry =	+ 1
Answer. $X - Y =$	0010001

(b) $Y - X = 1000011 - 1010100$

$Y =$	1000011
1's complement of $X =$	+ 0101011
Sum =	1101110

There is no end carry, Therefore, the answer is $Y - X = -$ (1's complement of 1101110) = – 0010001.

Signed Binary Numbers

- ♣ To represent negative integers, we need a notation for negative values.
- ♣ It is customary to represent the sign with a bit placed in the leftmost position of the number.
- ♣ The convention is to make the sign bit 0 for positive and 1 for negative.

Example: Although there is only one way to represent +9, there are three different ways to represent -9 with eight bits:

Signed-magnitude representation:	10001001
Signed-1's-complement representation:	11110110
Signed-2's-complement representation:	11110111

- ♣ **Table 3** lists all possible four-bit signed binary numbers in the three representations.

Table 1.3
Signed Binary Numbers

Decimal	Signed-2's Complement	Signed-1's Complement	Signed Magnitude
+7	0111	0111	0111
+6	0110	0110	0110
+5	0101	0101	0101
+4	0100	0100	0100
+3	0011	0011	0011
+2	0010	0010	0010
+1	0001	0001	0001
+0	0000	0000	0000
−0	—	1111	1000
−1	1111	1110	1001
−2	1110	1101	1010
−3	1101	1100	1011
−4	1100	1011	1100
−5	1011	1010	1101
−6	1010	1001	1110
−7	1001	1000	1111
−8	1000	—	—

Arithmetic Addition with Signed Binary Numbers

- The addition of two numbers in the signed-magnitude system follows the rules of ordinary arithmetic.
  - ✓ If the signs are the same, add the two magnitudes and give the sum the common sign.
  - ✓ If the signs are different, subtract the smaller magnitude from the larger and give the difference the sign if the larger magnitude.
- The addition of negative numbers represented in signed-2's-complement form is obtained from the addition of the two numbers, including their sign bits.
- A carry out of the sign-bit position is discarded.

Example: signed-2's-complement arithmetic

+ 6	00000110	− 6	11111010
+13	00001101	+13	00001101
+ 19	00010011	+ 7	00000111
+ 6	00000110	− 6	11111010
−13	11110011	−13	11110011
− 7	11111001	− 19	11101101

Arithmetic Subtraction

♣ **In 2's-complement form:**

1. Take the 2's complement of the subtrahend (including the sign bit) and add it to the minuend (including sign bit).
2. A carry out of sign-bit position is discarded.

$$(\pm A)-(+B)=(\pm A)+(-B)$$
$$(\pm A)-(-B)=(\pm A)+(+B)$$

Example:

(– 6) – (– 13) ⇒ (11111010 – 11110011)

⇒ (11111010 + 00001101)

⇒ 00000111 (+ 7)

➢ Since binary numbers in signed-complement system are added and subtracted by the same basic addition and subtraction rules as unsigned numbers, computers need only one common hardware circuit to handle both types of arithmetic.

Binary Codes

- An *n*-bit binary code can have up to 2^n distinct combinations of 1's and 0's, with each combination representing one element of the information set that is being coded.
- A set of four elements can be coded with two bits: 00, 01, 10, 11.
- A set of eight elements requires a 3-bit code and a set of 16 elements requires a 4-bit code. (What are these codes?)
- The bit combination of an *n*-bit code is the count in binary from 0 to 2^n - 1.
- Each element must be assigned a unique binary bit combination, and no two elements can have the same value; otherwise, the code assignment will be ambiguous.
- Although the *minimum* number of bits required to code 2^n distinct quantities is *n*, there is no *maximum* number of bits that may be used for a binary code. For example, the 10 decimal digits can be coded with 10 bits, and each decimal digit can be assigned a bit combination of nine 0's and a 1. In this particular binary code, the digit 6 is assigned the bit combination 0001000000. (It is called ***one-hot*** code.)

Binary-Coded Decimal (BCD)

Table 1.4
Binary-Coded Decimal (BCD)

Decimal Symbol	BCD Digit
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

- A number with *k* decimal digits will require 4*k* bits in BCD.
- Decimal 396 is represented in BCD with 12 bits as 0011 1001 0110, **with each group of 4 bits representing one decimal digit.**
- A decimal number in BCD is the same as its equivalent binary number only when the number is between 0 and 9.
- A BCD number greater than 10 looks different from its equivalent binary number, even though both contain 1's and 0's.
- **The binary combinations 1010 through 1111 are not used and have no meaning in BCD.**

BCD Conversion and Addition

Example:

Consider decimal 185 and its corresponding value in BCD and binary:

$$(185)_{10} = (0001\ 1000\ 0101)_{BCD} = (10111001)_2$$

BCD Addition

4	0100	4	0100	8	1000
+5	+0101	+8	+1000	+9	+1001
9	1001	12	1100	17	10001
			+0110		+0110
			10010		10111

If the decimal addition has carry, correct the results by adding 6 = 0110.

BCD Addition

Example:

Consider the addition of 184 + 576 = 760 in BCD:

BCD	1	1		
	0001	1000	0100	184
	+0101	0111	0110	+576
Binary sum	0111	10000	1010	
Add 6		0110	0110	
BCD sum	0111	0110	0000	760

- Consider the addition (+375) + (-240) = +135, done in the signed-complement system:

0	375
+9	760
0	135

9760 is the 10's complement of 0240.

Other Decimal Codes

Table 1.5
Four Different Binary Codes for the Decimal Digits

Decimal Digit	BCD 8421	2421	Excess-3	8, 4, −2, −1
0	0000	0000	0011	0000
1	0001	0001	0100	0111
2	0010	0010	0101	0110
3	0011	0011	0110	0101
4	0100	0100	0111	0100
5	0101	1011	1000	1011
6	0110	1100	1001	1010
7	0111	1101	1010	1001
8	1000	1110	1011	1000
9	1001	1111	1100	1111
	1010	0101	0000	0001
Unused	1011	0110	0001	0010
bit	1100	0111	0010	0011
combi-	1101	1000	1101	1100
nations	1110	1001	1110	1101
	1111	1010	1111	1110

Gray Code

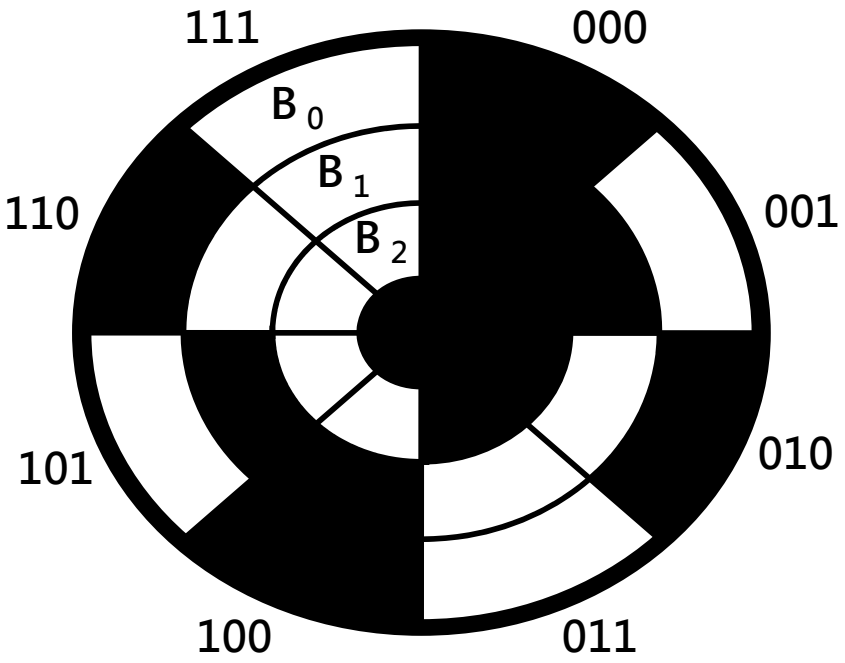
Table 1.6
Gray Code

Gray Code	Decimal Equivalent
0000	0
0001	1
0011	2
0010	3
0110	4
0111	5
0101	6
0100	7
1100	8
1101	9
1111	10
1110	11
1010	12
1011	13
1001	14
1000	15

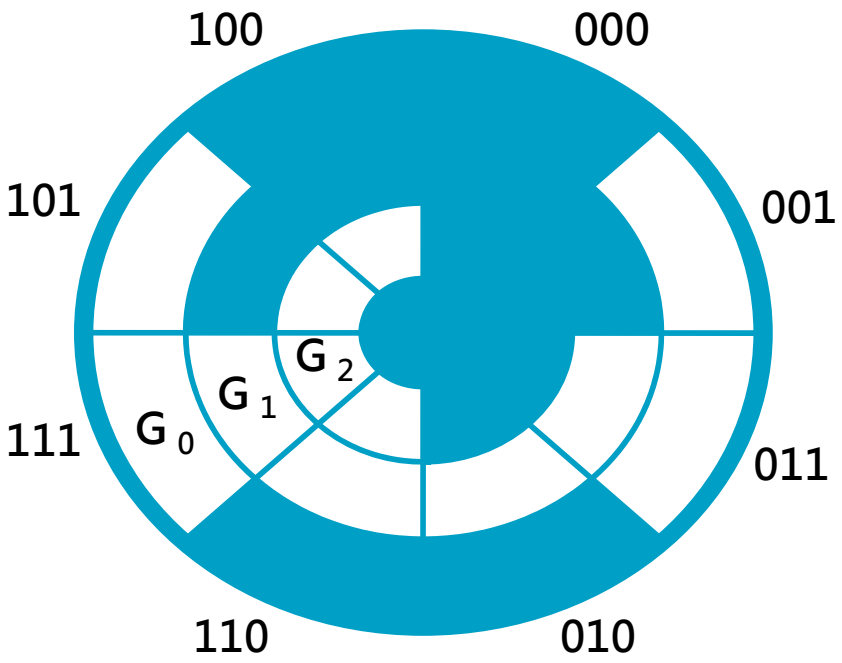
- ✓ Gray code changes only one bit in the code group going from one number to the next.
- ✓ For example, going from 7 to 8, the Gray code changes from 0100 to 1100. Only the first bit changes, from 0 to 1; the other three bits remain the same.
- ✓ If binary numbers are used, a change, for example, from 0111 to 1000 may produce an intermediate erroneous number 1001 if the value of the rightmost bit takes longer to change than do the values of the other three bits.

Gray Code (Continued)

- Does this special Gray code property have any value?
- Example: Optical Shaft Encoder



(a) Binary Code for Positions 0 through 7



(b) Gray Code for Positions 0 through 7

Gray Code Generation

- Gray codes

0	00	000	0000
1	01	001	0001
	11	011	0011
	10	010	0010
		110	0110
		111	0111
		101	0101
		100	0100
			1100
			1101
			1111
			1110
			1010
			1011
			1001
			1000

ASCII Character Code

- Digital computers handle not only the numbers, but also other characters or symbols, such as the letters of the alphabet.
- An alphanumeric character set is a set of elements that includes the 10 decimal digits, the 26 letters of the alphabet, and a number of special characters. Such a set contains between 36 and 64 elements if only capital letters are included, or between 64 and 128 elements if both uppercase and lowercase letters are included.
- The ***American Standard Code for Information Interchange*** (**ASCII**) uses seven bits to code 128 characters, as shown in Table 1.7 .
- The 7 bits of the code are designated by b_1 through b_7, with b_7 the most significant bit.
- The letter *A*, for example, is represented in ASCII as 1000001 (column 100, row 0001).
- The ASCII code also contains 94 graphic characters that can be printed and 34 nonprinting characters used for various control functions.
- The graphic characters consist of the 26 uppercase letters (A through Z), the 26 lowercase letters (a through z), the 10 numerals (0 through 9), and 32 special printable characters, such as %, *, and $.

ASCII Character Set

Table 1.7
American Standard Code for Information Interchange (ASCII)

	$b_7b_6b_5$							
$b_4b_3b_2b_1$	000	001	010	011	100	101	110	111
0000	NUL	DLE	SP	0	@	P	`	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	“	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	‘	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	\	l	|
1101	CR	GS	–	=	M	]	m	}
1110	SO	RS	.	>	N	^	n	~
1111	SI	US	/	?	O	–	o	DEL

ASCII Control Characters

Control Characters

NUL	Null	DLE	Data-link escape
SOH	Start of heading	DC1	Device control 1
STX	Start of text	DC2	Device control 2
ETX	End of text	DC3	Device control 3
EOT	End of transmission	DC4	Device control 4
ENQ	Enquiry	NAK	Negative acknowledge
ACK	Acknowledge	SYN	Synchronous idle
BEL	Bell	ETB	End-of-transmission block
BS	Backspace	CAN	Cancel
HT	Horizontal tab	EM	End of medium
LF	Line feed	SUB	Substitute
VT	Vertical tab	ESC	Escape
FF	Form feed	FS	File separator
CR	Carriage return	GS	Group separator
SO	Shift out	RS	Record separator
SI	Shift in	US	Unit separator
SP	Space	DEL	Delete

UNICODE

- UNICODE extends ASCII to 65,536 universal characters codes
  - For encoding characters in world languages
  - Available in many modern applications
  - 2 byte (16-bit) code words
  - See Reading Supplement – Unicode on the Companion Website http://www.prenhall.com/mano

Error-Detecting Code

- To detect errors in data communication and processing, an eighth bit is sometimes added to the ASCII character to indicate its parity.
- A ***parity*** bit is an extra bit included with a message to make the total number of 1's either even or odd.

Example:

Consider the following two characters and their even and odd parity:

	With even parity	With odd parity
ASCII A = 1000001	01000001	11000001
ASCII T = 1010100	11010100	01010100

parity bit

What kind of errors can be detected?

Can the error be corrected?

Memory: Binary Storage and Registers

Registers

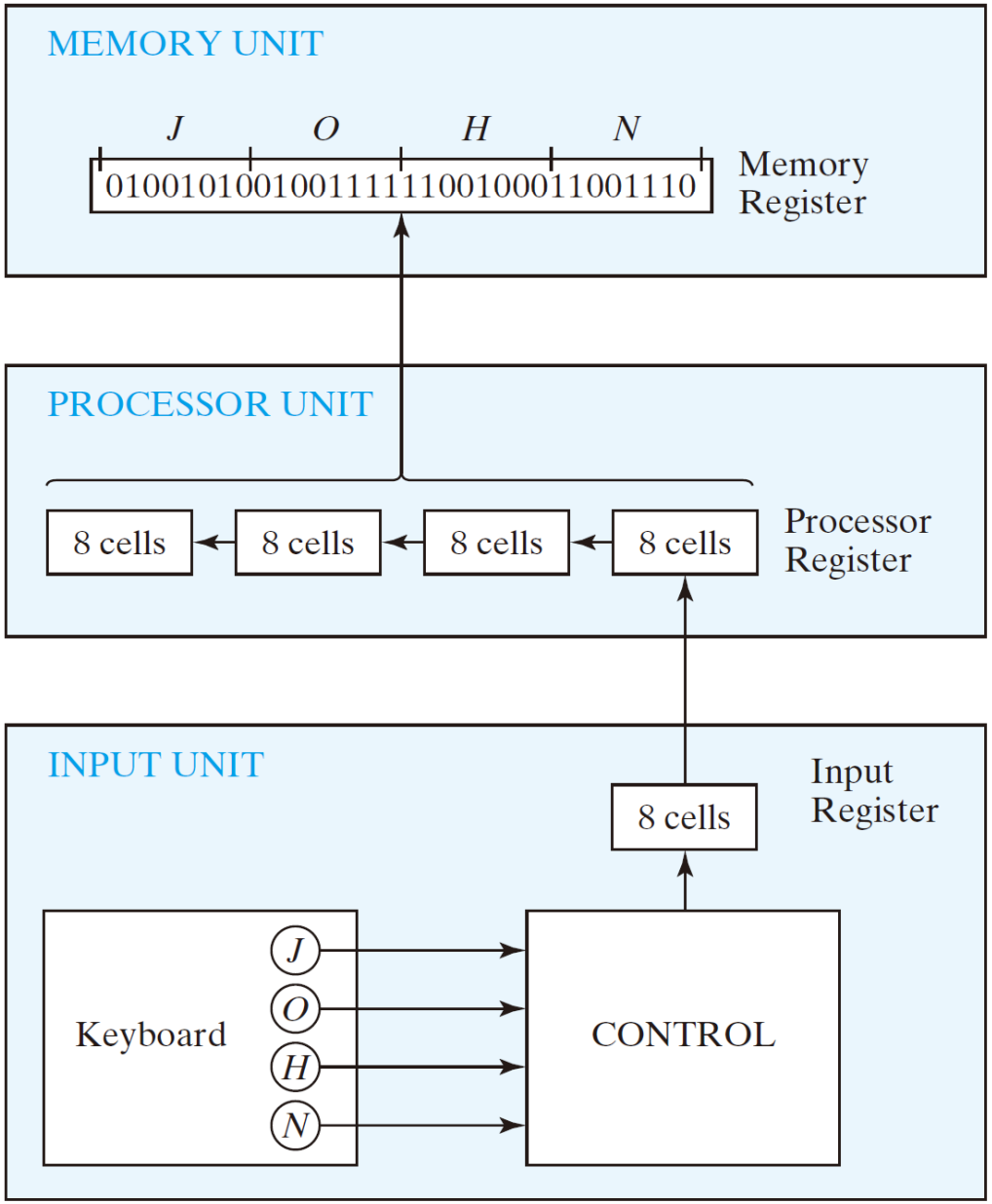
- The binary information in a digital computer must have a physical existence in some medium for storing individual bits.
- A *binary cell* is a device that possesses two stable states and is capable of storing one bit of information (0 or 1).
- A *register* is a group of binary cells.
- A register with *n* cells can store any discrete quantity of information that contains *n* bits.

***n* cells**  **2^n possible states**

- The content of a register (the information stored in it) is interpreted differently depending on the application of CPU.

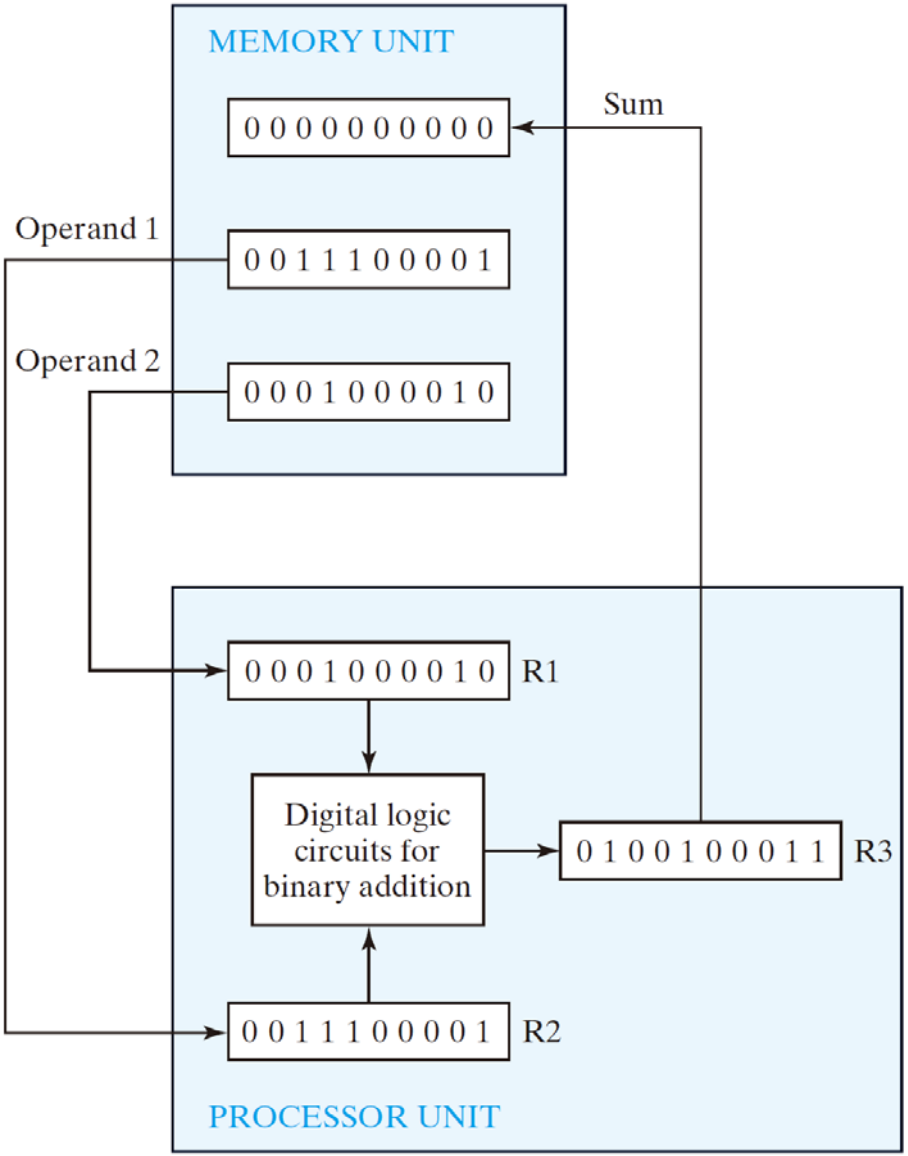
Enumerate the kinds of memory you know.

Register Transfer of information



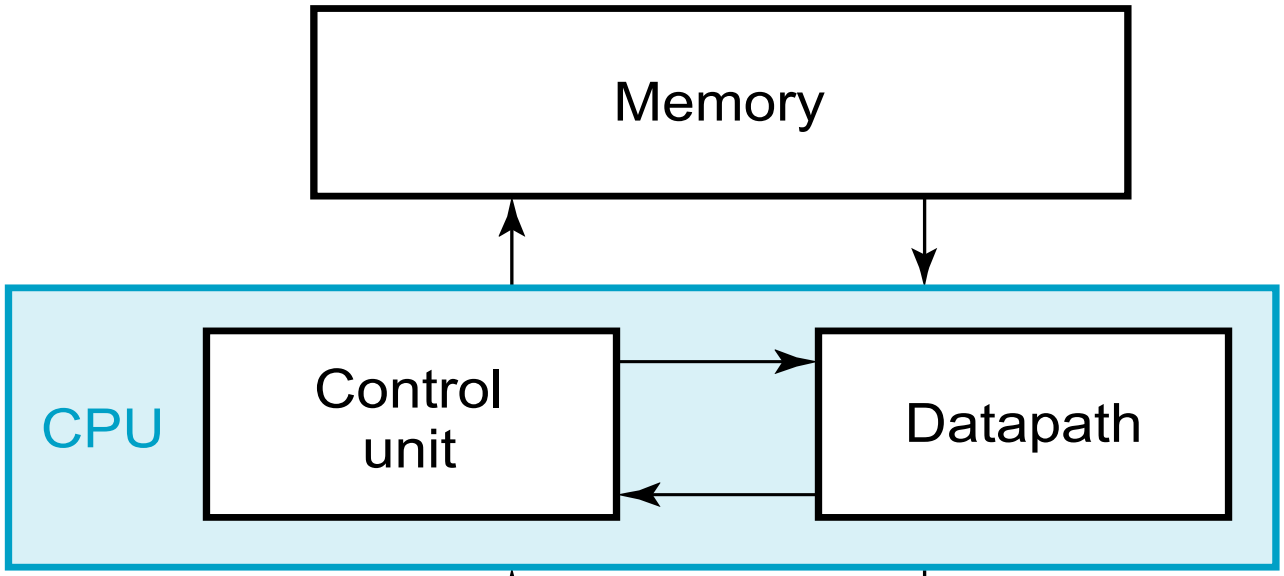
- The transfer of binary information from a keyboard into a register in the memory unit.

Binary Information Processing



- ✓ Binary variables are manipulated by means of digital logic circuits.
- ✓ Data is commonly hold in registers.
- ✓ Digital logic circuits and registers are covered in Chapters 2 through 6.
- ✓ Memory unit is explained in Chapter 7.
- ✓ The description of register operations and the design of digital systems are covered in Chapter 8 .

A Digital Computer Example



Inputs:

Keyboard, mouse, modem, microphone

Outputs:

LCD, modem, speakers

Synchronous or Asynchronous?

Binary Logic

- Binary logic consists of binary variables and a set of logical operations.
- The variables are designated by letters of the alphabet, such as *A*, *B*, *C*, *x*, *y*, *z*, etc, with each variable having two and only two distinct possible values: 1 and 0.
- There are three basic logical operations: AND, OR, and NOT.

AND: represented by a dot or by the absence of an operator.

- $x \cdot y = z$ or $xy = z$ is read "*x* AND *y* is equal to *z*."
- $z = 1$ if and only if $x = 1$ and $y = 1$; otherwise $z = 0$. (Remember that *x*, *y*, and *z* are binary variables and can be equal either to 1 or 0, and nothing else.)

OR: represented by a plus sign.

- $x + y = z$ is read "*x* OR *y* is equal to *z*," meaning that $z = 1$ if $x = 1$ or if $y = 1$ or if both $x = 1$ and $y = 1$.
- If both $x = 0$ and $y = 0$, then $z = 0$.

NOT: represented by a prime (sometimes by an overbar).

- $x' = z$ (or $x = \overline{z}$) is read "not *x* is equal to *z*," meaning that *z* is what *x* is not.
- If $x = 1$, then $z = 0$, but if $x = 0$, then $z = 1$.
- also referred to as the *complement* operation, since it changes a 1 to 0 and a 0 to 1.

Truth Table of Logic Operation

Table 1.8
Truth Tables of Logical Operations

AND			OR			NOT	
x	y	$x \cdot y$	x	y	$x + y$	x	x'
0	0	0	0	0	0	0	1
0	1	0	0	1	1	1	0
1	0	0	1	0	1		
1	1	1	1	1	1		

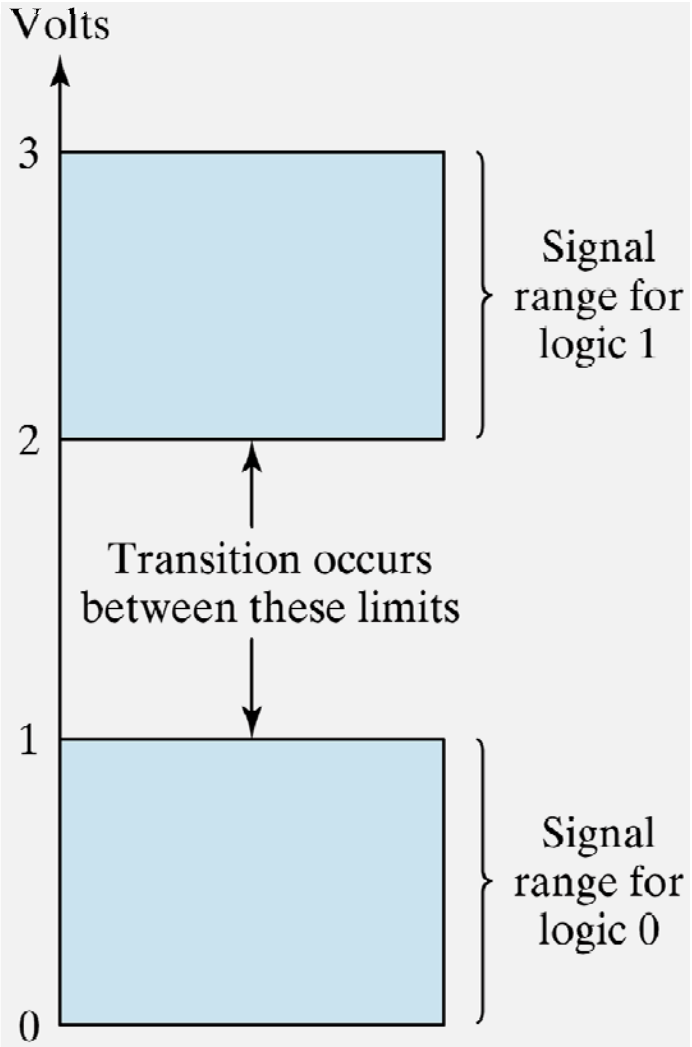
A logic variable is always either 1 or 0.

Binary Logic Levels

- Binary logic levels have evolved from a higher voltage to lower for saving power consumption.
- 5 V ➔ 3 V ➔ 1.8 V ➔ …
- Typically,

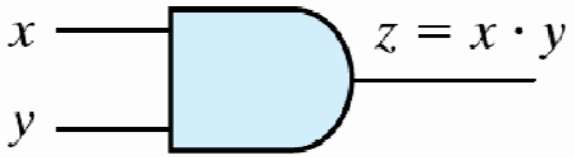
  a high voltage region (V_{DD}) represents logic 1

  a low voltage region (GND) represents logic 0
- The transition region is forbidden for proper logic operation.
- The required signal range for logic is different at the input and output sides of logic elements. The signal range at output side is more stringent than that at input side. The difference is called *noise margin*.

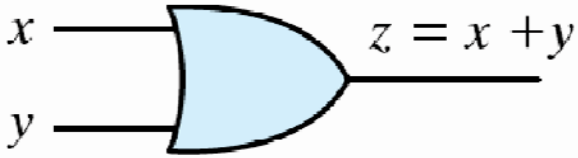


Basic Logic Gates

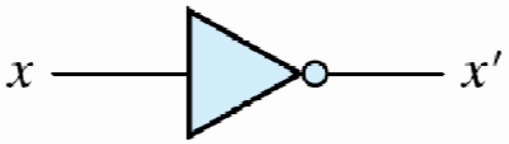
♣ Graphic Symbols and Input-Output Signals for Logic gates:



(a) Two-input AND gate

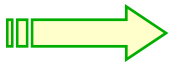


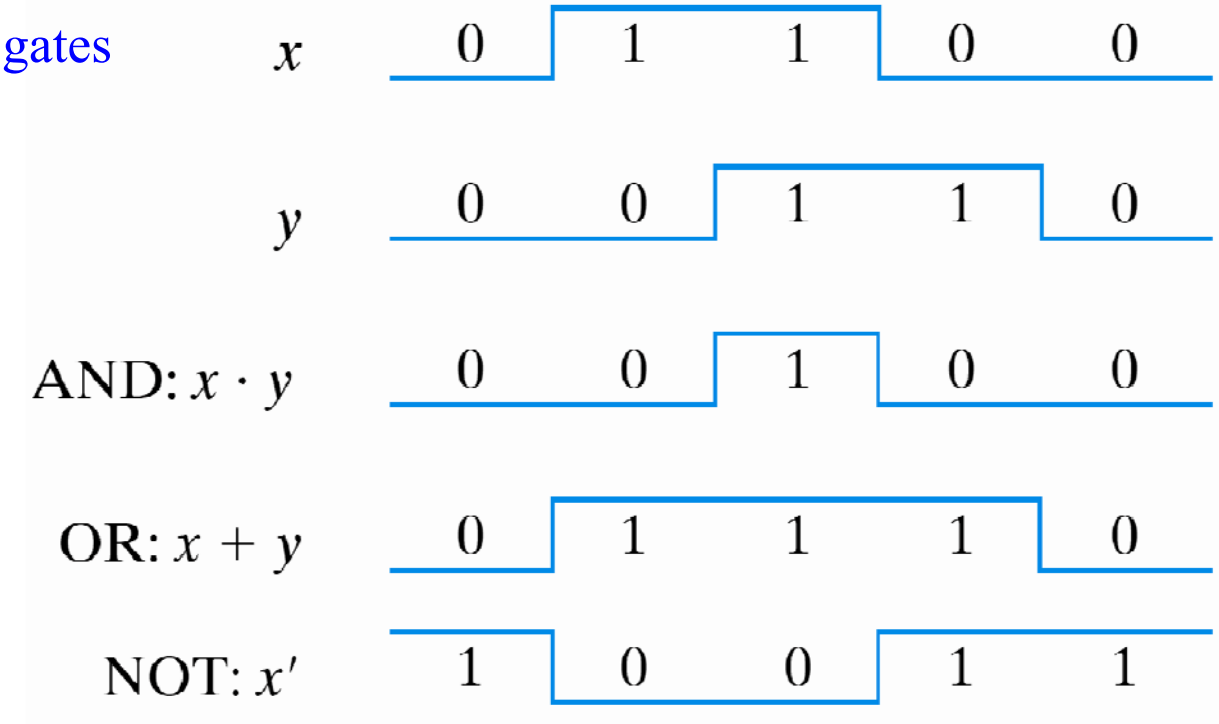
(b) Two-input OR gate



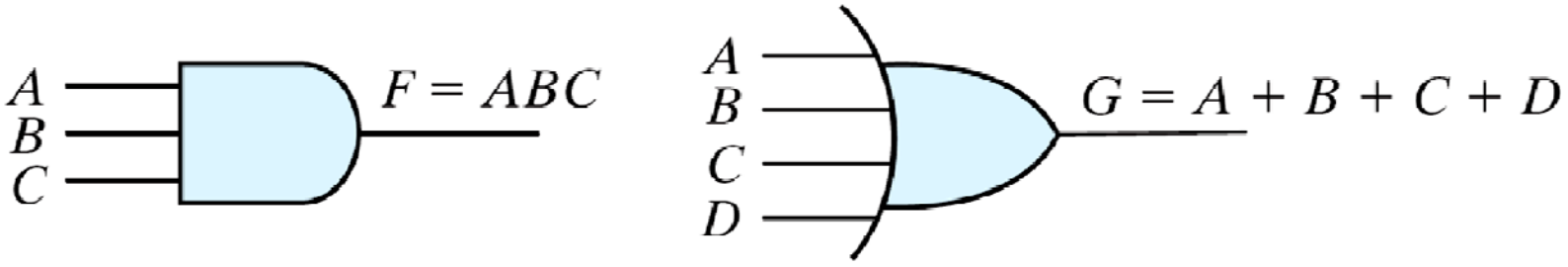
(c) NOT gate or inverter

➢Input-Output signals for logic gates



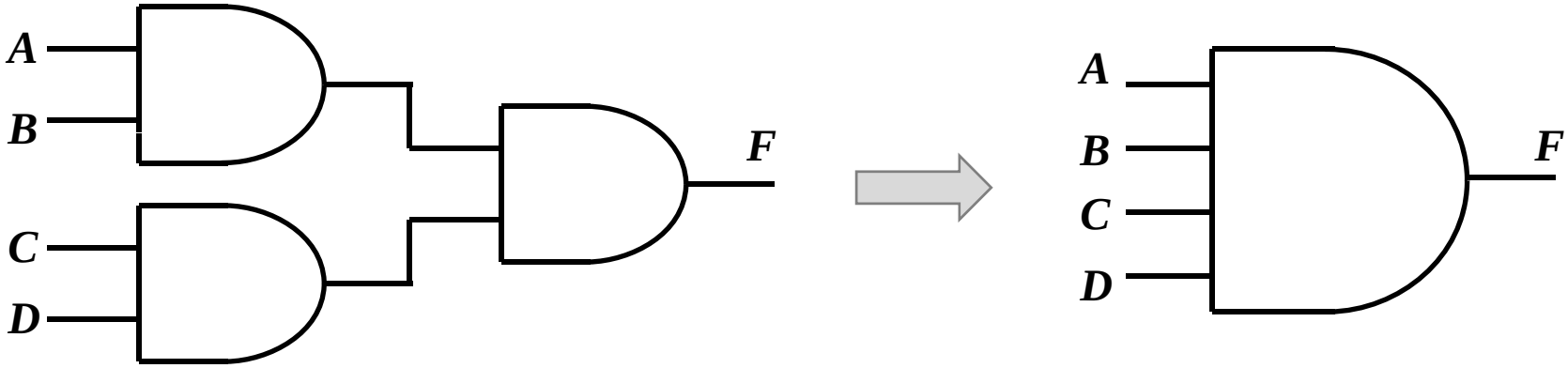


Multiple-Input Gates



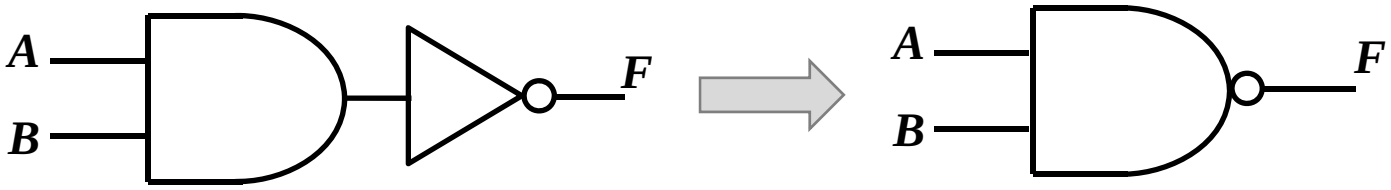
(a) Three-input AND gate (b) Four-input OR gate

$$F = (AB)(CD) = ABCD$$



Inverting Gates

NOT + AND = NAND



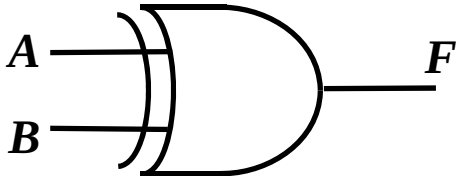
A	B	F
0	0	1
0	1	1
1	0	1
1	1	0

NOT + OR = NOR

A	B	F
0	0	1
0	1	0
1	0	0
1	1	0

Exclusive OR/NOR Gates

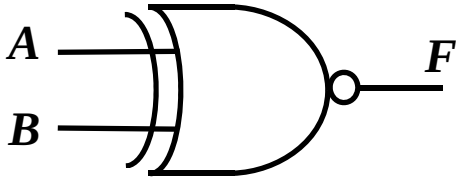
XOR



$F = A \oplus B$

A	B	F
0	0	0
0	1	1
1	0	1
1	1	0

XNOR



$F = A \odot B$

A	B	F
0	0	1
0	1	0
1	0	0
1	1	1

Homework #1

1.4

1.8

1.18

1.22

1.36

Due: one week