DIGITAL LOGIC

Lecture 1
Number Systems

2025 Fall

baiyh@sustech.edu.cn

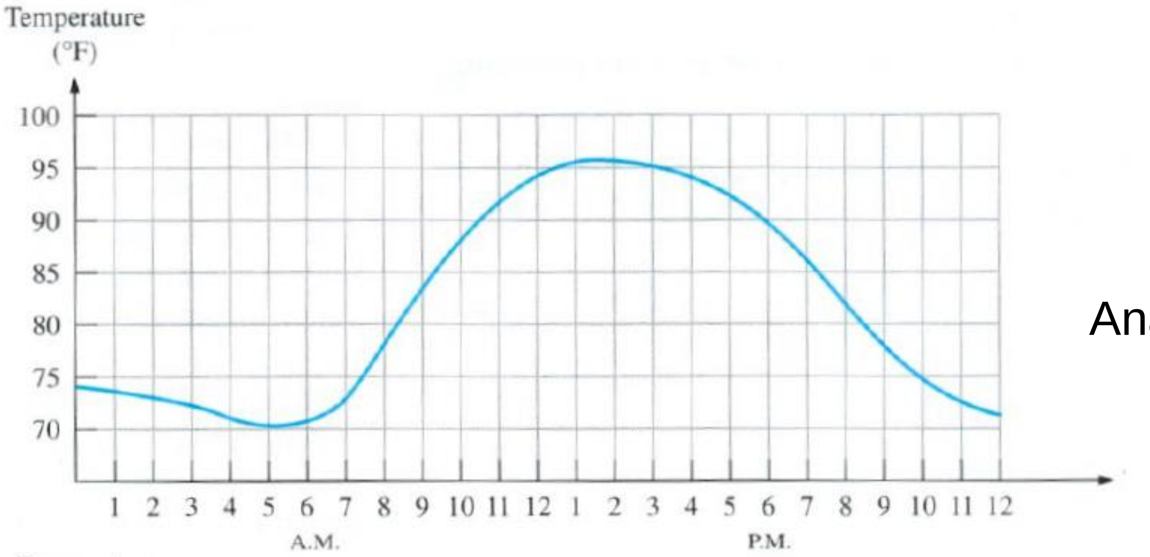
Outline

- **Digital Number Systems**
- Data Representation
- Binary Codes and Logic

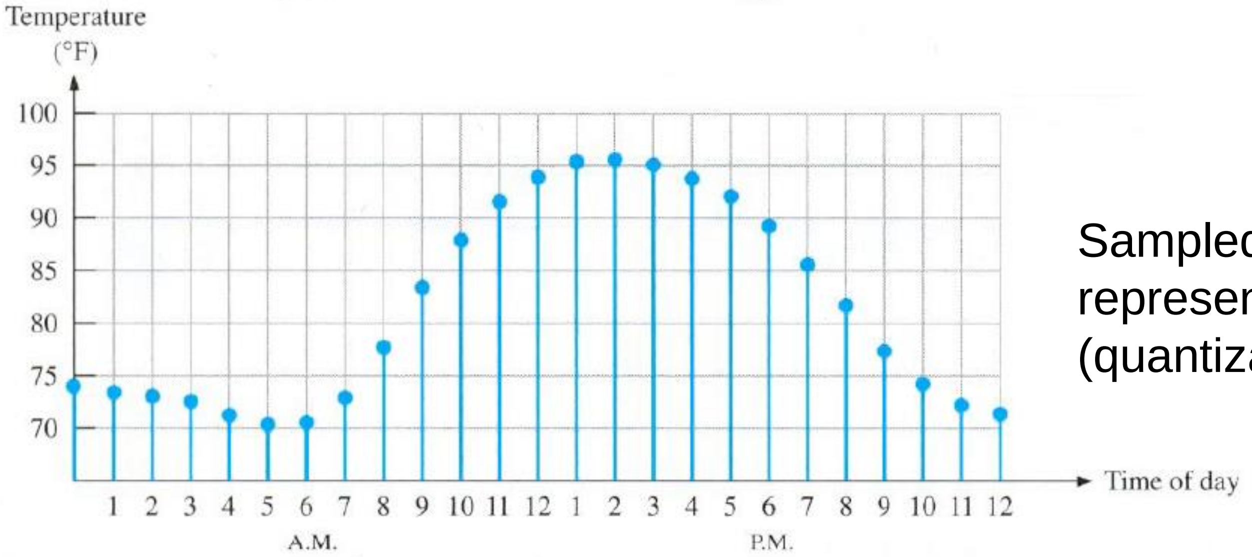
- Related material: Textbook Chapter 1

Analog vs. Digital Signals

- Signal definition
  - Quantity that can represent and convey information
  - Passed between devices to send and receive information
- Analog signals
  - Converts information into waves of varying amplitude and frequency
  - Continuous waveform
- Digital signals
  - ON (1) or OFF (0) pulses (i.e. binary)
  - Discrete waveform



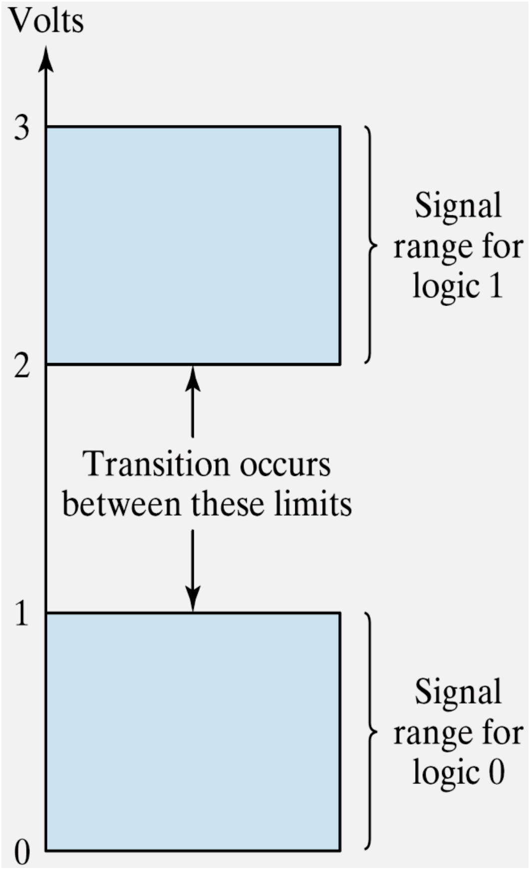
Analog quantity



Sampled-value representation (quantization)

Binary Digits and Logic Levels

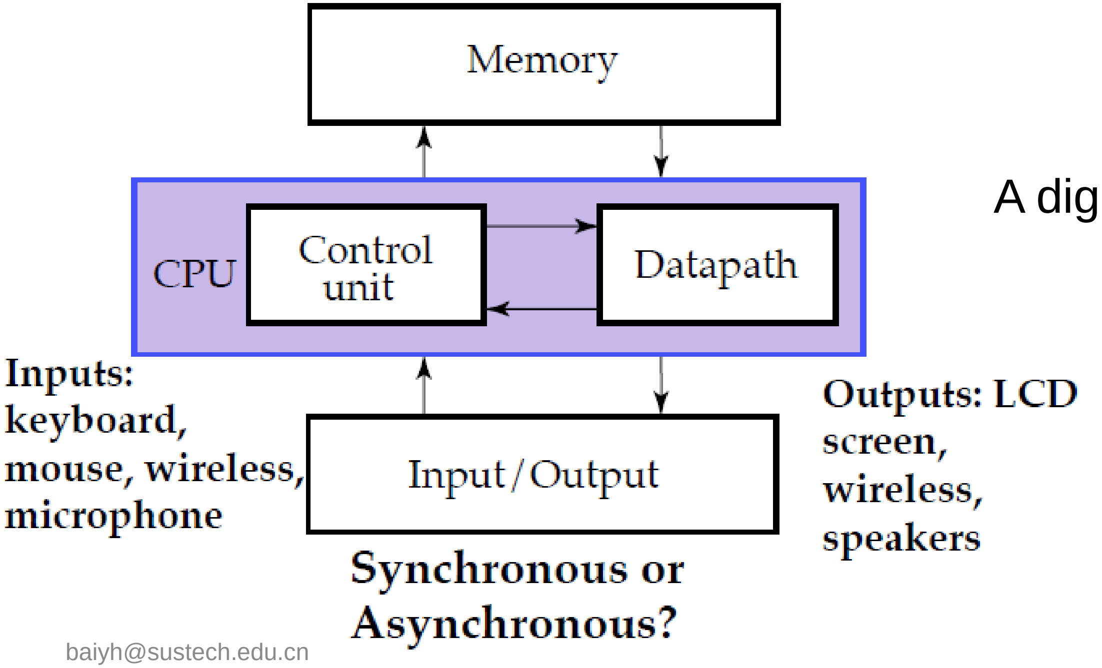
- Bit: binary digit
  - 1: HIGH (TRUE)
  - 0: LOW (FALSE)
- Codes: group of bits (combinations of 1s and 0s)
  - Used to represent numbers, letters, symbols, instructions, and anything else required in a given application
- Logic levels



Digital Systems

- A digital system is a system that processes digital signals or data. It operates on discrete values and performs operations such as logic, arithmetic, and data storage in a binary format.
- Digital systems are prevalent in modern electronics, including computers, smartphones, and digital communication devices, due to their reliability and ease of processing.

Understanding digital systems: first step toward more advanced topics like Computer Organization.



A digital computer

Common Number Systems

- It is natural for human to use **decimal system**(十进制)
- In a digital world, we think in **binary**(二进制)
- The **octal** (八进制) and **hexadecimal** (十六进制) numbers are shorter forms for representing binary numbers.

Decimal (base 10)	Binary (base 2)	Octal (base 8)	Hexadecimal (base 16)
0	0000	00	0x0
1	0001	01	0x1
2	0010	02	0x2
3	0011	03	0x3
4	0100	04	0x4
5	0101	05	0x5
6	0110	06	0x6
7	0111	07	0x7
8	1000	010	0x8
9	1001	011	0x9
10	1010	012	0xA
11	1011	013	0xB
12	1100	014	0xC
13	1101	015	0xD
14	1110	016	0xE
15	1111	017	0xF

Outline

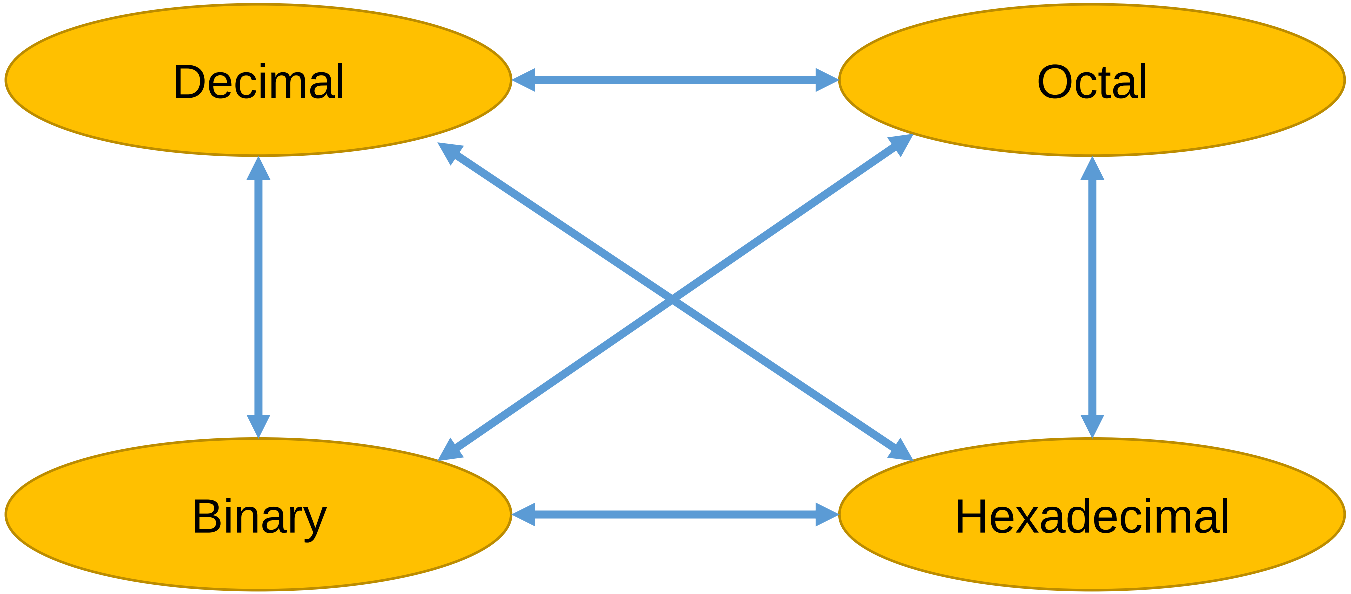
- Digital Number Systems
- **Data Representation**
  - Number Systems
  - Conversion to Decimal
  - Conversion from Decimal
  - Other Conversions
  - Binary arithmetic
  - Complement
  - Signed numbers
- Binary Codes and Logic

Conversion among Bases

- A quick example:
  - $25_{10} = 11001_2 = 31_8 = 19_{16}$

Base or Radix

- The possibilities:



Radix-r to Decimal Conversion

- We use Positional Number Systems: Let r be the radix (or base), then the (n+m)-digit number

$$D = d_{n-1}d_{n-2}\ldots d_1 d_0 . d_{-1}d_{-2}\ldots d_{-m} \quad 0 \leqslant d < r$$

has the value

Decimal Value

radix point

$$D = d_{n-1}r^{n-1} + d_{n-2}r^{n-2} + \ldots + d_1 r + d_0 + d_{-1}r^{-1} + d_{-2}r^{-2} + \ldots + d_{-m}r^{-m}$$

Most-significant Digit (MSD)

Least-significant Digit (LSD)

$$D = \sum_{i=-m}^{n-1} d_i r^i$$

Radix-r to Decimal Conversion

- **Decimal** Number System: Base (radix) r = 10

2	1	0		-1	-2
5	1	2	•	7	4

  - Coefficients $D=(d_2 d_1 d_0 . d_{-1} d_{-2}) = (512.74)_{10}$
  - $(512.74)_{10} = 5 \times 10^2 + 1 \times 10^1 + 2 \times 10^0 + 7 \times 10^{-1} + 4 \times 10^{-2}$

- **Binary** Number System: Base (radix) r = 2

2	1	0		-1	-2
1	0	1	•	0	1

  - Coefficients $D=(b_2 b_1 b_0 . b_{-1} b_{-2}) = (101.01)_2$
  - $(101.01)_2 = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2} = (5.25)_{10}$

Exercise:

$1010.101_2 = ?_{10}$

$22.22_4 = ?_{10}$

$12.5_8 = ?_{10}$

$A.A_{16} = ?_{10}$

Radix-r to Decimal Conversion

$$D = d_{n-1}r^{n-1} + d_{n-2}r^{n-2} + \dots + d_1 r + d_0 + d_{-1}r^{-1} + d_{-2}r^{-2} + \dots + d_{-m}r^{-m}$$

Most-significant Digit (MSD)

Least-significant Digit (LSD)

Exercise:

$1010.101_2 = 1*2^3 + 0*2^2 + 1*2^1 + 0*2^0 + 1*2^{-1}+0*2^{-2}+1*2^{-3} = 10.625_{10}$

$22.22_4 = 2*4^1+2*4^0+2*4^{-1}+2*4^{-2} = 10.625_{10}$

$12.5_8 = 1*8^1+2*8^0+5*8^{-1} = 10.625_{10}$

$A.A_{16} = 10*16^0+10*16^{-1} = 10.625_{10}$

Decimal to Radix-r Conversion

- Integer part: Successive divisions by r and observe the remainders
- Fraction: Successive multiplications by r and observe the integer part

Decimal to Binary Conversion (1)

- For Integer part
- Divide the number by the ‘Base’ (=2)
- Take the remainder (either 0 or 1) as a coefficient
- Take the quotient and repeat the division

Example: $(13)_{10}$

	Quotient	Remainder	Coefficient
13 / 2 =	6	1	$a_0 = 1$
6 / 2 =	3	0	$a_1 = 0$
3 / 2 =	1	1	$a_2 = 1$
1 / 2 =	0	1	$a_3 = 1$

Answer: $(13)_{10} = (a_3\ a_2\ a_1\ a_0)_2 = (1101)_2$

MSB LSB

Decimal to Binary Conversion (2)

- For Fraction part, the computation is reversed again
- Multiply the number by the 'Base' (=2)
- Take the integer (either 0 or 1) as a coefficient
- Take the resultant fraction and repeat the division

Example: $(0.625)_{10}$

	Integer		Fraction	Coefficient
0.625 * 2 =	1	.	25	$a_{-1} = 1$
0.25 * 2 =	0	.	5	$a_{-2} = 0$
0.5 * 2 =	1	.	0	$a_{-3} = 1$

Answer: $(0.625)_{10} = (0.a_{-1}\ a_{-2}\ a_{-3})_2 = (0.101)_2$

MSB LSB

Decimal to Binary Conversion (3)

- Easier method

Example: (35.375)$_{10}$

2	35		
2	17	…	1
2	8	…	1
2	4	…	0
2	2	…	0
2	1	…	0
	0	…	1

	0.375		
x	2		
	$\boxed{\cancel{0}}$.750	…	0
x	2		
	$\boxed{\cancel{1}}$.50	…	1
x	2		
	$\boxed{\cancel{1}}$.0	…	1

- $(100011.011)_2$

Decimal to Octal Conversion

Example: $(175.3125)_{10}$

	Quotient	Remainder	Coefficient
175 / 8 =	21	7	$a_0 = 7$
21 / 8 =	2	5	$a_1 = 5$
2 / 8 =	0	2	$a_2 = 2$

Integer part: $(175)_{10} = (a_2\, a_1\, a_0)_8 = (257)_8$

	Integer		Fraction	Coefficient
0.3125 * 8 =	2	.	5	$a_{-1} = 2$
0.5 * 8 =	4	.	0	$a_{-2} = 4$

Fraction part: $(0.3125)_{10} = (0.a_{-1}\, a_{-2}\, a_{-3})_8 = (0.24)_8$

Answer: $(175.3125)_{10} = (a_2\, a_1\, a_0.a_{-1}\, a_{-2}\, a_{-3})_8 = (257.24)_8$

Radix-r to Radix-r Conversion

- Binary − Octal
  - Each group of 3 bits represents an octal digit starting from radix point
  - Works both ways (Binary to Octal & Octal to Binary)

Assume Zeros

$$(1\,0\,1\,1\,0\,.\,0\,1)_2$$

$$(2\quad 6\;.\;2)_8$$

- Binary − Hexadecimal
  - Each group of 4 bits represents a hexadecimal digit starting from radix point
  - Works both ways (Octal to Hex & Hex to Octal)

Assume Zeros

$$(1\,0\,1\,1\,0\,.\,0\,1)_2$$

$$(1\quad 6\;.\;4)_{16}$$

Radix-r to Radix-r Conversion (2)

- Octal − Hexadecimal
- Convert to Binary as an intermediate step

$$(2\ \ 6\ .\ 2)_8$$

Assume Zeros

$$(0\,1\,0\,1\,1\,0\,.\,0\,1\,0)_2$$

Assume Zeros

$$(1\ \ 6\ .\ 4)_{16}$$

Common Notions

- Bits(位/位元)

$\underbrace{1}_{\text{most significant bit}}001011\underbrace{0}_{\text{least significant bit}}$

- Bytes(字节)
  - 1 byte = 8 bits

$\overbrace{10010110}^{\text{byte}}$

- Hexadecimal representation
  - two hexadecimal digit represent one byte value
  - often used to represent memory address

$0X\underbrace{CE}_{\text{most significant byte}}BF9A\underbrace{D7}_{\text{least significant byte}}$

Power	Meaning	Prefix	Symbol
2^{10}	1024	Kilo	K
2^{20}	1024^2	Mega	M
2^{30}	1024^3	Giga	G
2^{40}	1024^4	Tera	T
2^{50}	1024^5	Peta	P
2^{60}	1024^6	Exa	E
2^{70}	1024^7	Zetta	Z

e.g. 1MB = 1024KB

Binary Addition

- Same rules as for decimal numbers
- Column Addition

carry

$(10)_2 = (2)_{10}$

	1	1	1	1	1	1			
		1	1	1	1	0	1	= 61	augend
+			1	0	1	1	1	= 23	addend
	1	0	1	0	1	0	0	= 84	sum

Binary Subtraction

- Same rules as for decimal numbers
- Borrow a “Base” when needed

borrow

$$
\begin{array}{cccccccc|l|l}
 & & 1 & & & 2 & & & & \\
 & 0 & \cancel{2} & 2 & 0 & 0 & 2 & & = (10)_2 & \\
 & \cancel{1} & 0 & 0 & \cancel{1} & \cancel{1} & 0 & 1 & = 77 & \text{minuend} \\
- & & & 1 & 0 & 1 & 1 & 1 & = 23 & \text{subtrahend} \\
\hline
 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & = 54 & \text{difference}
\end{array}
$$

Complements

- When human do subtraction, we use “borrow” concept to borrow a 1 from a higher significant position.
- It is hard for circuits to design “borrow”. So complements are used to implement subtraction.
    - Simplify the subtraction operation.
    - Simpler, less expensive circuits.
- Two types for radix-r system
    - Diminished radix complement (反码) ((r-1)’s-complement)
    - Radix complement (补码) (r’s-complement)
- Examples:
    - For a binary system: 1’s complement and 2’s complement.
    - For a decimal system: 9’s complement and 10’s complement.
- Question:
    - For a radix-7 system: ?

Complements for decimal system

- Diminished radix complement
  - **r^n - 1 – x, assuming r is the radix, n is number of bits, x is the value**
  - 9's-complement of 540 = 999 - 540 = 459
  - 9's-complement of 12 = 999 - 012 = 987 (is it equal to 87?)
- Radix complement
  - **r^n – x, assuming r is the radix, n is number of bits, x is the value**
  - 10's-complement of 540 = 1000 – 540 = 460
  - 10's-complement of 12= 1000 – 012 = 988 (is it equal to 88?)
  - **Easier method 1**: Calculate the diminished radix complement, then plus one
    - 10's-complement of 540 = 999 - 540 + 1 = 460
  - **Easier method 2**: use *r* minus the least significant non-zero digit, and $r - 1$ minus digits on the left
    - The least significant non-zero digit of 540 is 4: 10 − 4 = 6;
    - Digits on the left is 5: 9 − 5 = 4;
    - The 10's complement of 540 is 460.

Complements for binary system

- 1's Complement (*Diminished Radix* Complement) for binary
  - All '0's become '1's
  - All '1's become '0's
  - just like a bitwise not(按位取反)

  Example $(10110000)_2$

  ⇨ $(01001111)_2$

- 2's Complement: 1's complement, then plus one:

	1 0 1 1 0 0 0 0
	0 1 0 0 1 1 1 1
+	1
	0 1 0 1 0 0 0 0

Another way to find 2's complement: leave the first non-zero LSB unchanged, and then replacing 1's with 0's and 0's with 1's in the other MSBs:

1 0 1 1 0 0 0 0

first non-zero LSB

0 1 0 1 0 0 0 0

Subtraction with Complements

- Algorithm:
- Subtraction of two k-digit unsigned numbers M – N
  - Replace subtraction with addition
  - M and N both have k-digit
- M-N = M + r’s complement of N
  - If M >= N, the sum will produce an end **carry** r^k which is **discarded**, and what is left is the result M – N
  - If M < N, the sum does not produce an end carry. It is equal to the r's complement of (M - N). The correct answer is generated by
    1. taking the r's **complement** of the answer
    2. then adding a **negative sign** to the front
- Pay attention to align the number of digits for two operands

Subtraction with 10's Complement

- Example with M>=N
  - Using 10's complement, subtract 72532 – 3250.

$$
\begin{array}{rr}
M = & 72532 \\
\text{10's complement of } N = & +\underline{96750} \\
\text{Sum} = & 169282 \\
\text{Discard end carry } 10^5 = & -\underline{100000} \\
\text{Answer} = & 69282
\end{array}
$$

- Example with M < N
  - Using 10's complement, subtract 3250 – 72532.

$$
\begin{array}{rr}
\text{M} = & 03250 \\
\text{10's complement of N} = & +\underline{27468} \\
\text{Sum} = & 30718
\end{array}
$$

There is no end carry.
Answer = – (10's complement of 30718)
= – 69282.

Note: When working with paper and pencil, we can change the answer to a signed negative number in order to put it in a familiar form.

Subtraction with 2’s Complement

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

$$
\begin{array}{lrr}
\text{(a)} & \text{X} = & 1010100 \\
 & \text{2's complement of Y} = & \underline{+0111101} \\
 & \text{Sum} = & 10010001 \\
 & \text{Discard end carry } 2^7 = & \underline{-10000000} \\
 & \text{Answer. X} - \text{Y} = & 0010001
\end{array}
$$

$$
\begin{array}{lrr}
\text{(b)} & \text{Y} = & 1000011 \\
 & \text{2's complement of X} = & +\ \underline{0101100} \\
 & \text{Sum} = & 1101111
\end{array}
$$

There is no end carry.
Answer. Y – X
= – (2's complement of 1101111)
= – 0010001.

Signed Binary Numbers

- To represent Signed numbers in real life:
  - We have + and −
- To represent signed number in digital system
  - everything in binary digit
- **Three types** of representations of signed binary numbers:
  - Sign-magnitude representation
  - Signed-1’s complement representation
  - Signed-2’s complement representation
- In Signed binary system, the convention is to make the sign bit (MSB)
  - **0 for positive**
  - and **1 for negative**

3 Types of Signed Binary Numbers

- Example, assume 9-bits number representation:
- $(105)_{10}$?
- $105_{10}=1101001_2$, represent in 9 bits
  - Signed-magnitude representation of 105: 001101001
  - Signed-1's-comrepresentation of 105: 001101001
  - Signed-2's-complement representation plement of 105: 001101001

MSb $\overbrace{\qquad\qquad\qquad}^{\text{8bits}}$

S	7	6	5	4	3	2	1	0
0	0	1	1	0	1	0	0	1
0	0	1	1	0	1	0	0	1
0	0	1	1	0	1	0	0	1

- $(-105)_{10}$?
- Magnitude of -105 is 1101001, represent in 9 bits
  - Signed-magnitude representation of -105: 101101001
  - Signed-1's-complement representation of -105: 110010110
  - Signed-2's-complement representation of -105: 110010111

MSb $\overbrace{\qquad\qquad\qquad}^{\text{8bits}}$

S	7	6	5	4	3	2	1	0
1	0	1	1	0	1	0	0	1
1	1	0	0	1	0	1	1	0
1	1	0	0	1	0	1	1	1

Which Signed Binary Number to Choose?

- All possible 4-bit signed binary numbers in the three representations.
- Which one is the best? Why?
- Noted:
  - To unified representation,
  - for positive number, the complement is the same as the original form
  - for negative number, the complement is obtained by finding the 1's complement of its absolute value representation, then add 1

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	—	—

Which Signed Binary Number to Choose?

	Addition	Representation of 0	Range
Sign-magnitude	Doesn't work → -6+6 1110 + 0110 10100 (wrong!)	Two representations 0000 +0 1000 -0	$[-(2^{N-1}-1), 2^{N-1}-1]$
Signed-1's complement	Doesn't work → -3+6 1100 + 0110 10010 (wrong!)	Two representations 0000 +0 1111 -0	$[-(2^{N-1}-1), 2^{N-1}-1]$
Signed-2's complement	Works → -3+6 1101 + 0110 10011 (correct!)	Only one 0000 ±0 1000 is -8	$[-2^{N-1}, 2^{N-1}-1]$

Outline

- **Digital Number Systems**
- Data Representation
- Binary Codes and Logic
  - BCD Code
  - Gray Code
  - ASCII Code
  - Error Detection Code

BCD Codes

- BCD Code
  - Four bits are required to code each decimal number.
    - Decimal 396 is represented in BCD with 12bits as 0011 1001 0110, with each group of 4 bits representing one decimal digit.
  - Also known as 8-4-2-1 code, as 8, 4, 2, and 1 are the weights of the four bits of BCD.
  - The binary combinations 1010 through 1111 are not used and have no meaning in 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

BCD Addition

- First add the two numbers using normal rules for binary addition.
- If the 4-bit sum is equal to or less than 9, it becomes a valid BCD number.
- If the 4-bit sum is **greater than 9**, or if a **carry** out of the group is generated, it is an invalid result.
  - In such a case, add $(0110)_2$ or $(6)_{10}$ to the 4-bit sum in order to skip the six invalid states and return the code to BCD. If a carry results when 6 is added, add the carry to the next 4-bit group.
- Example: Consider the addition of 184 + 576 = 760 in BCD:

BCD	1	1		
	0001	1000	0100	184
	+ 0101	0111	0110	+576
Binary sum	0111 valid result	carry ~~1~~0000	1010 greater than 9	
Add 6		0110	0110	
BCD sum	0111	0110	~~1~~0000	760

Question: Why adding 6 if greater than 9 or carry occurs?

BCD Subtraction

- Same as in the binary case:
- Take the 10's complement of the subtrahend and add it to the minuend.
- Example: Consider the subtraction of 109 - 132 = -23 in BCD:
  - Take 10's comp of 132 = 868
  - Convert difference into 10's complement

Subtraction		1		
	0001	0000	1001	109
	+1000	0110	1000	+868
			carry	
Binary sum	1001	0111	10001	
Add 6	0000	0000	0110	
Difference	1001	0111	0111	977
10's complement				-23

Gray Code

- Gray Code(格雷码)
  - Minimum change code: A number changes by only one bit as it proceeds from one number to the next.
    - Error detection.
    - Representation of analog data.
    - Low power design.

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

ASCII Codes

- American Standard Code for Information Interchange (ASCII) Character Code
  - Many applications of the computer require not only handling of numbers, but also of letters.
  - To represent letters it is necessary to have a binary code for the alphabet.
  - Seven bits to code 128 characters.

	$b_7b_6b_5$							
$b_4b_3b_2b_1$	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

Error-Detecting Code

- Error-Detecting Code （奇偶校验码）
  - To detect errors in data communication and processing, an extra (eighth in this example) bit is sometimes added to the ASCII character to indicate its parity.
  - A parity bit (校验位) is an extra bit included with a message to make the total number of 1's either even or odd.
- Example:

	With even parity	With odd parity
ASCII A = 1000001	$\boxed{0}$1000001	$\boxed{1}$1000001
ASCII T = 1010100	$\boxed{1}$1010100	$\boxed{0}$1010100

  - Suppose we use even parity

Original code	With even parity	sender	receiver	Parity check Passed?
1000001	01000001	01000001	01000001	yes
1000001	01000001	01000001	01001001	No
1000001	01000001	01000001	01001101	Yes but fails for double errors

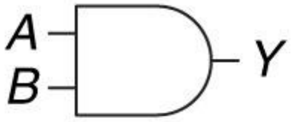
Binary Logic

- Binary logic deals with binary variables(e.g. can have two values,“0”and “1”)
- Binary variables can undergo three basic logical operators AND, OR and NOT
  - AND is denoted by a dot (•) z = x • y or z = xy.
  - OR is denoted by a plus (+) z = x + y.
  - NOT is denoted by a single quote mark (’) after the variable, or an overbar (‾) above the variable.
    - x’y is pronounced as "x prime y" or "x complement y.
- Binary logic resembles binary arithmetic.
  - However, binary logic should not be confused with binary arithmetic.
  - An arithmetic variable designates a number that may consist of many digits.
  - A logic variable is always either 0 or 1.

Binary Logic

- Truth Tables, Boolean Expressions, and Logic Gates

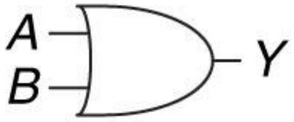
AND



$Y = AB$

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

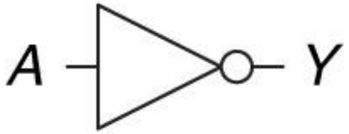
OR



$Y = A + B$

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

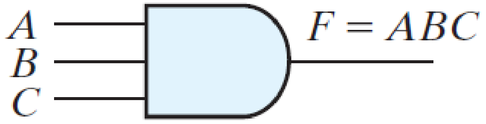
NOT

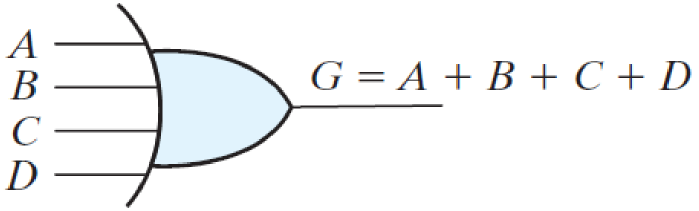


$Y = \overline{A}$

A	Y
0	1
1	0

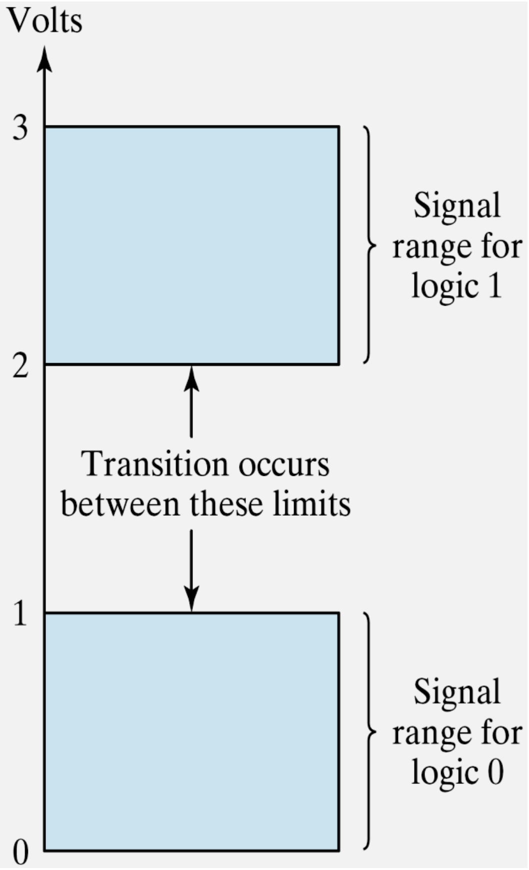
- It is fine to have more than two inputs for AND/OR



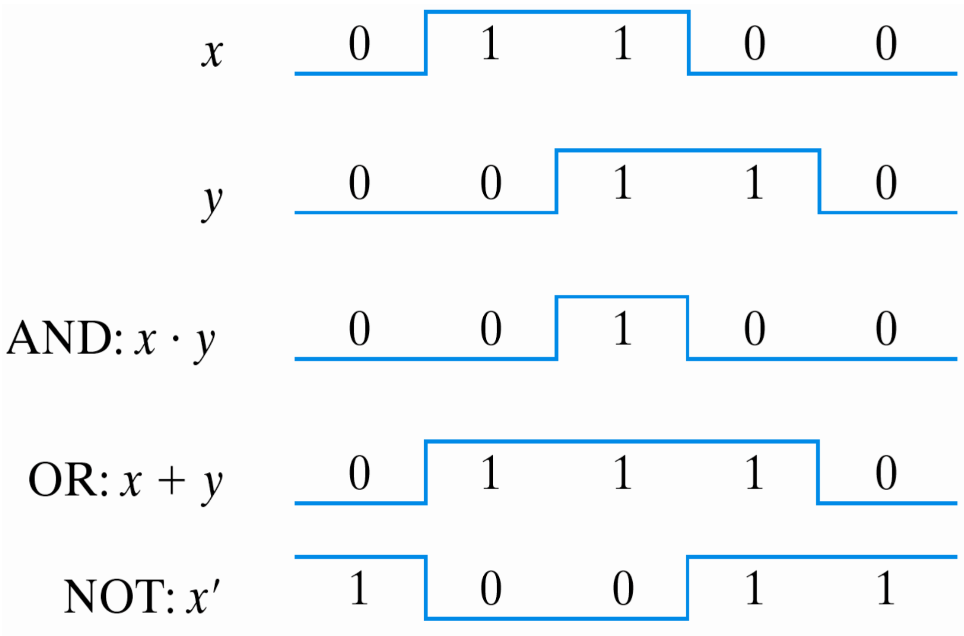


Binary Logic

- Voltage-operated, though on a range, interpreted to be either of the two values



Signal levels for binary logic values



Input–output signals for gates

Summary

- Number Systems
  - Binary, Octal, Hexadecimal
  - Conversion: Use weighted sums (to Decimal) and division/multiplication (from Decimal).
- Binary Arithmetic
  - Addition/Subtraction: Column-wise with carry/borrow.
  - Complements: Simplify subtraction.
    - 1's Comp: Bitwise NOT.
    - 2's Comp: (1's Comp + 1); preferred method.
- Signed Numbers
  - MSB is Sign Bit: 0 = Positive, 1 = Negative.
  - 2's Complement is Standard: Uniquely represents zero, simplifies arithmetic.
- Binary Codes
  - BCD: 4 bits per decimal digit.
  - Gray Code: Adjacent values differ by 1 bit.
  - Parity Bit: Single-bit error detection.
- Binary Logic
  - Gates (AND, OR, NOT): Building blocks.
  - Logic Levels: HIGH (1) / LOW (0) voltage ranges.