Logic and Computer Design Fundamentals

Chapter 1 – Digital Systems and Information

Yueming Wang (王跃明)
Professor
ymingwang@zju.edu.cn
2018

College of Computer Science, Zhejiang University

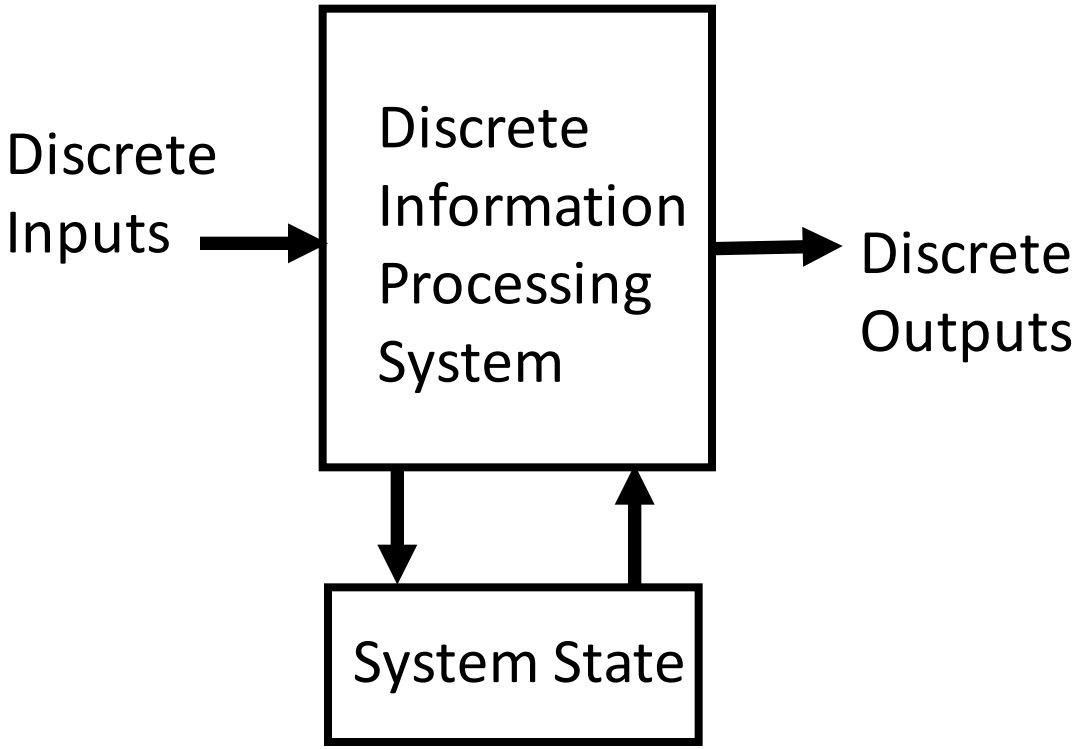
Qiushi Academy for Advanced Studies, Zhejiang University

Overview

- **Digital Systems and Computers**
- **Information Representation**
- **Number Systems** [binary, octal and hexadecimal]
  - **Arithmetic Operations**
  - **Base Conversion**
- **Coding**
  - **Decimal Codes** [BCD (binary coded decimal)]
  - **Gray Codes**
  - **Parity Bit**
  - **Alphanumeric Codes**

DIGITAL & COMPUTER SYSTEMS - Digital System

- **Takes a set of discrete information inputs and discrete internal information (system state) and generates a set of discrete information outputs.**



Types of Digital Systems

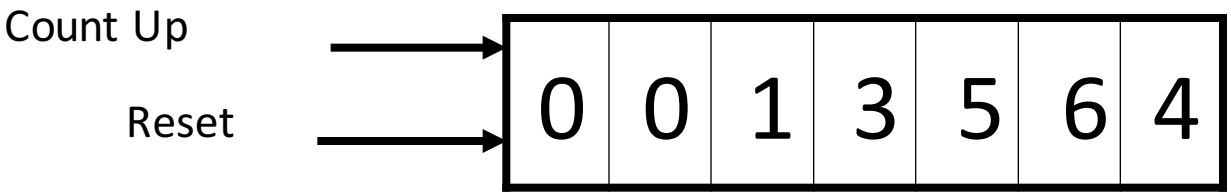
- **Combinational Logic System**
  - **No state present**
  - **Output = Function(Input)**
- **Sequential System**
  - **State present**
  - **State updated at discrete times**

    => Synchronous Sequential System
  - **State updated at any time**

    =>Asynchronous Sequential System
  - **State = Function (State, Input)**
  - **Output = Function (State)**
    or Function (State, Input)

Digital System Example:

A Digital Counter (e. g., odometer):



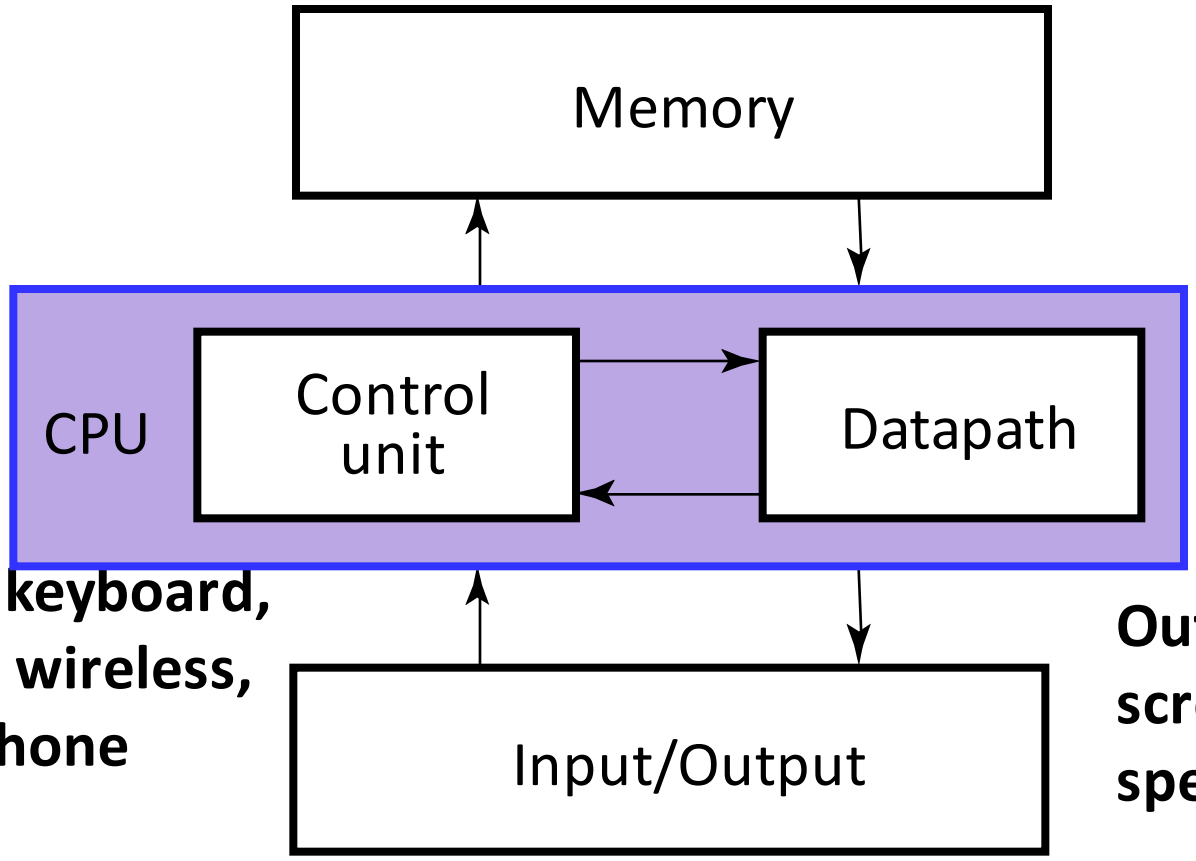
Inputs: **Count Up, Reset**

Outputs: **Visual Display**

State: **"Value" of stored digits**

Synchronous or Asynchronous?

Digital Computer Example



Inputs: keyboard, mouse, wireless, microphone

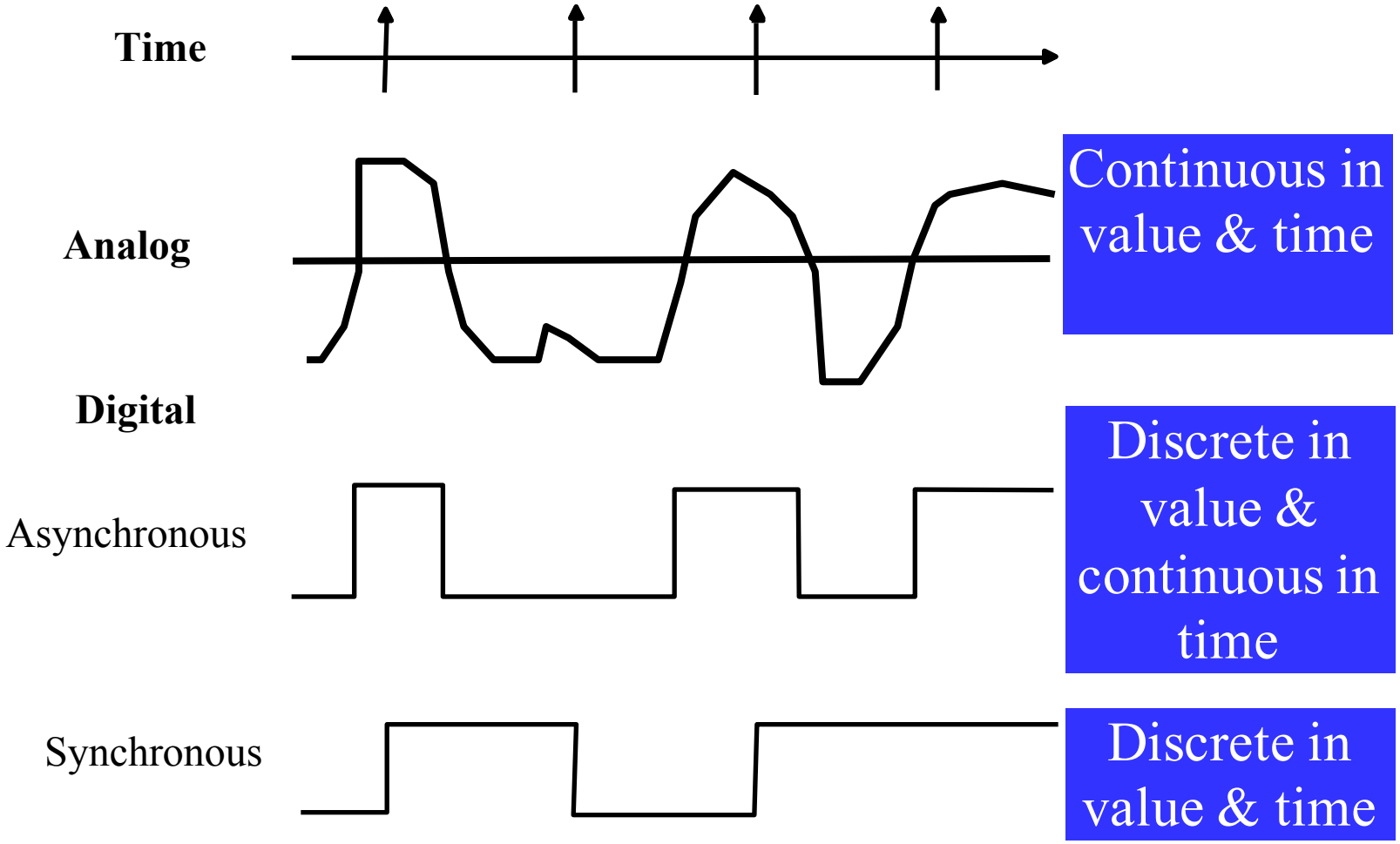
Outputs: LCD screen, wireless, speakers

Synchronous or Asynchronous?

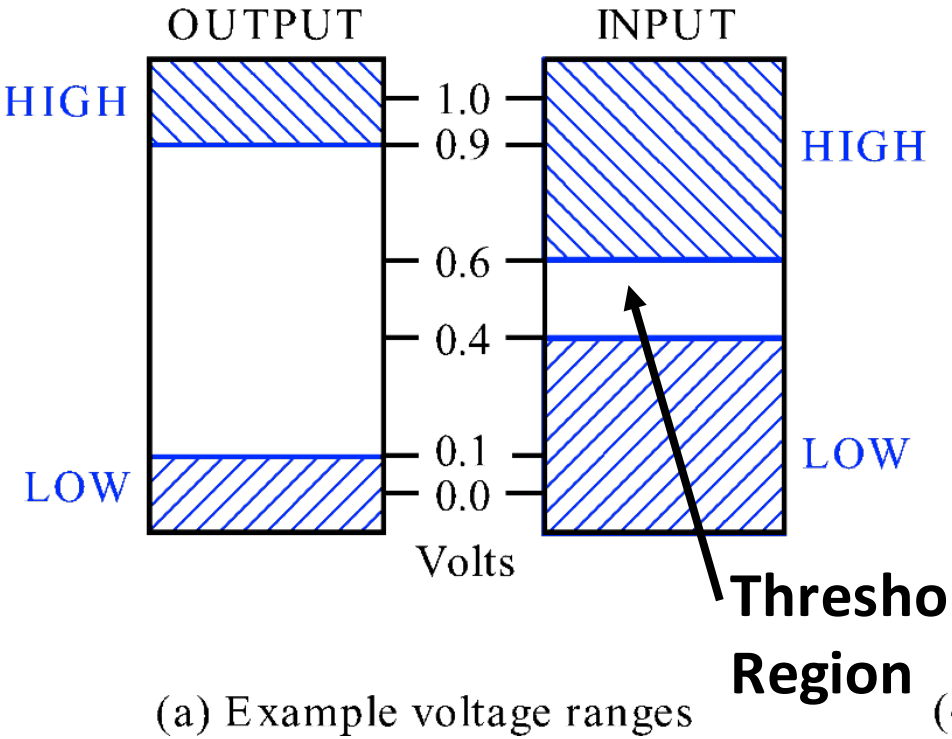
Information Representation - Signals

- **Information variables represented by physical quantities.**
- **For digital systems, the variables take on discrete values.**
- **Two level, or binary values are the most prevalent values in digital systems.**
- **Binary values are represented abstractly by:**
  - **digits 0 and 1**
  - **words (symbols) False (F) and True (T)**
  - **words (symbols) Low (L) and High (H)**
  - **and words On and Off.**
- **Binary values are represented by values or ranges of values of physical quantities**

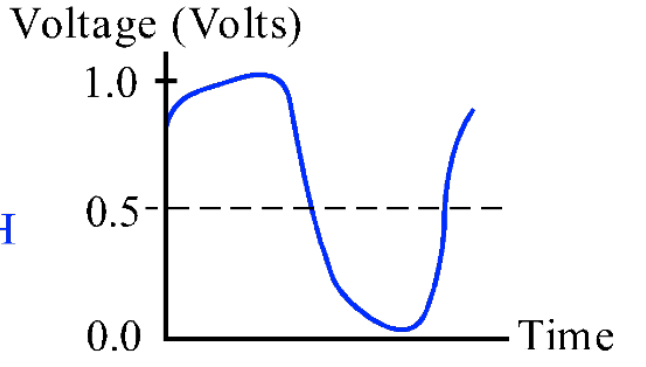
Signal Examples Over Time



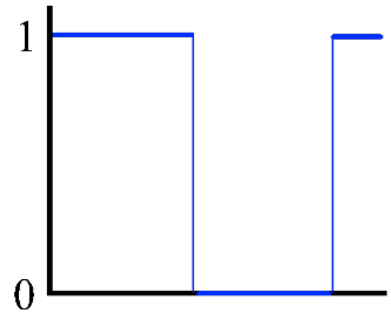
Signal Example – Physical Quantity: Voltage



(a) Example voltage ranges



(b) Time-dependent Voltage



(c) Binary model of time-dependent voltage

Binary Values: Other Physical Quantities

- **What are other physical quantities represent 0 and 1?**
  - **CPU** — Voltage
  - **Disk** — Magnetic Field Direction
  - **CD** — Surface Pits/Light
  - **Dynamic RAM** — Electrical Charge

NUMBER SYSTEMS – Representation

- Start from decimal numbers, 223.45
  - 2 100 plus 2 10 plus 3 1 plus 4 0.1 plus 5 0.01

    $(223.45)_{10}=2\cdot10^{2}+2\cdot10^{1}+3\cdot10^{0}+4\cdot10^{-1}+5\cdot10^{-2}$
  - Convention: 223.45, a string only with digits , infer the powers from the positions
- Generally,

$$(A_{n-1}A_{n-2},...,A_0.A_{-1},...,A_{-m})_{10}=(\sum_{i=-m}^{n-1}A_i10^i)_{10}=$$

$$A_{n-1}\cdot10^{n-1}+A_{n-2}\cdot10^{n-2}+\ldots+A_1\cdot10^{1}+A_0\cdot10^{0}+A_{-1}\cdot10^{-1}+\ldots+A_{-m}\cdot10^{-m}$$

NUMBER SYSTEMS – Representation

- More general form, positional number systems
  - A number with *radix (base)* ***r*** is represented by a string of digits:

    $$A_{n-1}A_{n-2}\cdots A_1A_0 . A_{-1}A_{-2}\cdots A_{-m+1}A_{-m}$$

    in which $\mathbf{0 \le A_i < r}$ and **.** is the *radix point*.
  - The string of digits represents the power series:

$$(\text{Number})_r = \left(\sum_{i=0}^{i=n-1} A_i \cdot r^i\right) + \left(\sum_{j=-m}^{j=-1} A_j \cdot r^j\right)$$

(Integer Portion) + (Fraction Portion)

  - r^j: weight; A_{n-1}: Most significant digit (msd); A_{-m}: Least significant digit (lsd)

The decimal number system

Radix: r = 10
Digits: 0, 1, 2, 3, …, 9
Weight: $W_i = 10^i$
Expressed as:

$$(N)_{10} = \left(\sum_{i=-m}^{n-1} A_i 10^i\right)_{10} =$$

$$A_{n-1} \cdot 10^{n-1} + A_{n-2} \cdot 10^{n-2} + \ldots + A_1 \cdot 10^1 + A_0 \cdot 10^0 + A_{-1} \cdot 10^{-1} + \ldots + A_{-m} \cdot 10^{-m}$$

Example: $(123.45)_{10} = 1 \cdot 10^2 + 2 \cdot 10^1 + 3 \cdot 10^0 + 4 \cdot 10^{-1} + 5 \cdot 10^{-2}$

The binary number system

Radix: r = 2
Digits: 0, 1
Weight: $W_i = 2^i$
Expressed as:

$$(N)_2 = \left(\sum_{i=-m}^{n-1} A_i 2^i\right)_2$$

$$A_{n-1} \cdot 2^{n-1} + A_{n-2} \cdot 2^{n-2} + \ldots + A_1 \cdot 2^1 + A_0 \cdot 2^0 + A_{-1} \cdot 2^{-1} + \ldots\ldots + A_{-m} \cdot 2^{-m}$$

Example: $(1011.101)_2 = 1\cdot2^3+0\cdot2^2+1\cdot2^1+1\cdot2^0+1\cdot2^{-1}+0\cdot2^{-2}+1\cdot2^{-3}$

Number Systems – Examples

		General	Decimal	Binary
Radix (Base)		**r**	**10**	**2**
Digits		**0 => r - 1**	**0 => 9**	**0 => 1**
Powers of Radix	**0**	r^0	**1**	**1**
	1	r^1	**10**	**2**
	2	r^2	**100**	**4**
	3	r^3	**1000**	**8**
	4	r^4	**10,000**	**16**
	5	r^5	**100,000**	**32**
	-1	r^{-1}	**0.1**	**0.5**
	-2	r^{-2}	**0.01**	**0.25**
	-3	r^{-3}	**0.001**	**0.125**
	-4	r^{-4}	**0.0001**	**0.0625**
	-5	r^{-5}	**0.00001**	**0.03125**

The octal number system

Radix: r = 8
Digits: 0, 1, 2, 3, …, 7
Weight: $W_i = 8^i$
Expressed as:

$$(N)_8 = \left(\sum_{i=-m}^{n-1} A_i 8^i\right)_8 =$$

$$A_{n-1} \cdot 8^{n-1} + A_{n-2} \cdot 8^{n-2} + \ldots + A_1 \cdot 8^1 + A_0 \cdot 8^0 + A_{-1} \cdot 8^{-1} + \ldots\ldots + A_{-m} \cdot 8^{-m}$$

Example: $(567.125)_8$

The hexadecimal number system

Radix: r = 16

Digits: 0, 1, 2, …, 9, A, B, …, F

Weight: $W_i = 16^i$

Expressed as:

$$(N)_{16} = \left(\sum_{i=-m}^{n-1} A_i 16^i\right)_{16} =$$

$$A_{n-1} \cdot 16^{n-1} + A_{n-2} \cdot 16^{n-2} + \ldots + A_1 \cdot 16^1 + A_0 \cdot 16^0 + A_{-1} \cdot 16^{-1} + \ldots + A_{-m} \cdot 16^{-m}$$

Example: $(5AF.9B)_{16}$

Numbers in Different Bases

- **Good idea to memorize!**

Decimal (Base 10)	Binary (Base 2)	Octal (Base 8)	Hexa decimal (Base 16)
00	00000	00	00
01	00001	01	01
02	00010	02	02
03	00011	03	03
04	00100	04	04
05	00101	05	05
06	00110	06	06
07	00111	07	07
08	01000	10	08
09	01001	11	09
10	01010	12	0A
11	01011	13	0B
12	01100	14	0C
13	01101	15	0D
14	01110	16	0E
15	01111	17	0F
16	10000	20	10

Binary Addition

Carries	00000	01100
Augend	01100	10110
Addend	+10001	+10111
Sum	11101	101101

Binary Subtraction

Borrows	00000	00110
Minuend	**10110**	**10110**
Subtrahend	**- 10010**	**- 10011**
Difference	00100	00011

- **Notes: If the Subtrahend > the Minuend, interchange and append a – to the result.**

Binary Multiplication

The binary multiplication table is simple:

0 * 0 = 0 | 1 * 0 = 0 | 0 * 1 = 0 | 1 * 1 = 1

Extending multiplication to multiple digits:

Multiplicand	**1011**
Multiplier	**x 101**
Partial Products	**1011**
	0000 -
	1011 - -
Product	**110111**

Converting Binary to Decimal

- **To convert to decimal, use decimal arithmetic to form Σ (digit × respective power of 2).**
- **Example:**

$$(110\ 0101.101)_2 = 1*2^6+1*2^5+0*2^4+0*2^3+$$
$$+1*2^2+0*2^1+1*2^0+1*2^{-1}+0*2^{-2}+1*2^{-3}$$
$$=(101.625)_{10}$$

Converting Decimal to Binary

- **Convert the Integer Part**
- **Convert the Fraction Part**
- **Join the two results with a radix point**

Conversion Details

- **To Convert the Integral Part:**

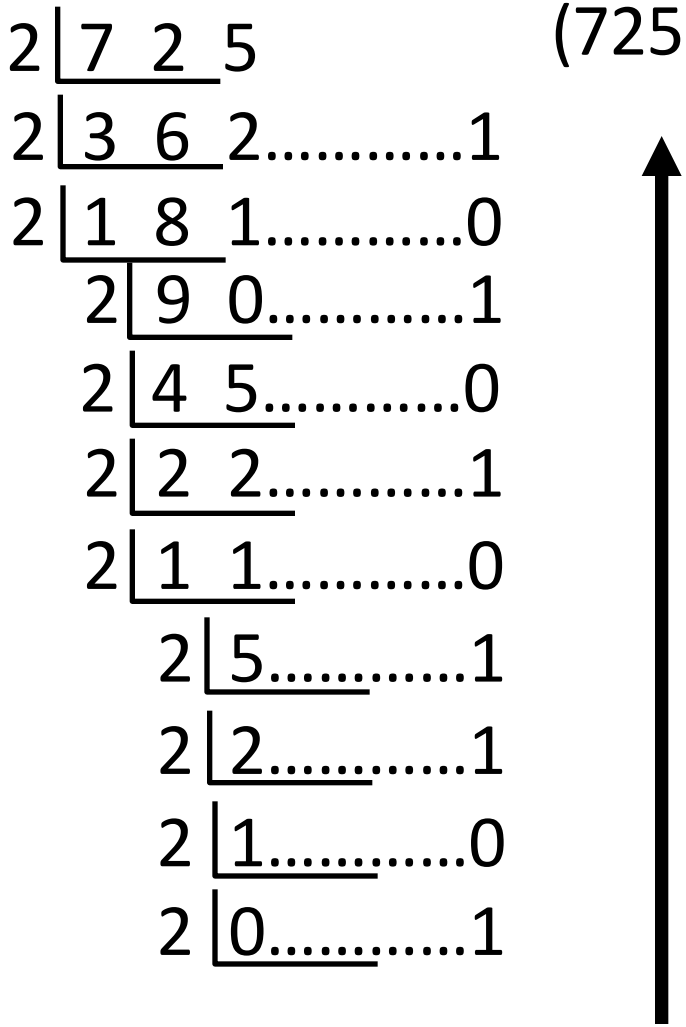
  Repeatedly divide the number by 2 and save the remainders. The digits for Binary are the remainders in *reverse order* of their computation.

- **To Convert the Fractional Part:**

  Repeatedly multiply the fraction by 2 and save the integer digits that result. The digits for Binary are the integer digits in *order* of their computation.

Example: Convert 725_{10} To Base 2

$(725)_{10} = (010\ 1101\ 0101)_2$



Example: Convert 0.678_{10} To Base 2

$(0.678)_{10} = (0.1010\ 1101\ 1001)_2$

$2 \times 0.678\ldots\ldots = 1.356$

$2 \times 0.356\ldots\ldots = 0.712$

$2 \times 0.712\ldots\ldots = 1.424$

$2 \times 0.424\ldots\ldots = 0.848$

$2 \times 0.848\ldots\ldots = 1.696$

$2 \times 0.696\ldots\ldots = 1.392$

$2 \times 0.392\ldots\ldots = 0.784$

$2 \times 0.784\ldots\ldots = 1.568$

$2 \times 0.568\ldots\ldots = 1.136$

$2 \times 0.136\ldots\ldots = 0.272$

$2 \times 0.272\ldots\ldots = 0.544$

$2 \times 0.544\ldots\ldots = 1.088$

Example: Convert 46.6875_{10} To Base 2

- **Convert 46 to Base 2**

- **Convert 0.6875 to Base 2:**

- **Join the results together with the radix point:**

Additional Issue - Fractional Part

- **Note that in this conversion, the fractional part can become 0 as a result of the repeated multiplications.**
- **In general, it may take many bits to get this to happen or it may never happen.**
- **Example Problem: Convert 0.65_{10} to N_2**
  - **0.65 = 0.1010011001001 …**
  - **The fractional part begins repeating every 4 steps yielding repeating 1001 forever!**
- **Solution: Specify number of bits to right of radix point and round or truncate to this number.**

Octal (Hexadecimal) to Binary and Back

- **Octal (Hexadecimal) to Binary:**
  - **Restate the octal (hexadecimal) as three (four) binary digits starting at the radix point and going both ways.**
- **Binary to Octal (Hexadecimal):**
  - **Group the binary digits into three (four) bit groups starting at the radix point and going both ways, padding with zeros as needed in the fractional part.**
  - **Convert each group of three bits to an octal (hexadecimal) digit.**

Octal to Binary and Back

$(67.731)_8 =$（110 111 .111 011 001)$_2$

$(312.64)_8 =$(011 001 010 . 110 1)$_2$

$(11\ 111\ 101\ .\ 010\ 011\ 11)_2 = (375.236)_8$

$(10\ 110.11)_2 = (26.6)_8$

Hexadecimal to Binary and Back

$(3AB4.1)_{16}$ = $(0011\ 1010\ 1011\ 0100\ .0001)_2$

$(21A.5)_{16}$=$(0010\ 0001\ 1010.0101)_2$

$(1001101.01101)_2$= $(\ 0100\ 1101.0110\ 1000\ )_2$=$(4D.68)_{16}$

$(111\ 1101\ .\ 0100\ 1111)_2$=$(7D.4F)_{16}$

$(110\ 0101.101)_2$=$(65.A)_{16}$

Octal to Hexadecimal via Binary

- **Convert octal to binary.**
- **Use groups of <u>four bits</u> and convert as above to hexadecimal digits.**
- **Example: Octal to Binary to Hexadecimal**

$6\ 3\ 5\ .\ 1\ 7\ 7_8$

- **Why do these conversions work?**

BASE CONVERSION - Positive Powers of 2

- Useful for Base Conversion

Exponent	Value
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1024

Exponent	Value
11	2,048
12	4,096
13	8,192
14	16,384
15	32,768
16	65,536
17	131,072
18	262,144
19	524,288
20	1,048,576
21	2,097,152

Coding - Decimal Code

- **By the binary number system, binary numbers can represent decimal numbers. However,**
- **Computers input and output data used by most people need to be in the decimal system. Working in the manner of the decimal system is highly important.**
- **Binary coding**
  - **assign any binary combination (called a code word) to any data as long as data is uniquely encoded.**
  - **Numeric**
    - **Must represent range of data needed**
    - **Very desirable to represent data such that simple, straightforward computation for common arithmetic operations permitted**
    - **Tight relation to binary numbers**
  - **Non-numeric**
    - **Greater flexibility since arithmetic operations not applied.**
    - **Not tied to binary numbers**

DECIMAL CODES - Binary Coded Decimal (BCD)

There are over 8,000 ways that you can choose 10 elements from the 16 binary numbers of 4 bits. A few are useful:

Decimal	8,4,2,1	Excess3	8,4,-2,-1	Gray
0	0000	0011	0000	0000
1	0001	0100	0111	0100
2	0010	0101	0110	0101
3	0011	0110	0101	0111
4	0100	0111	0100	0110
5	0101	1000	1011	0010
6	0110	1001	1010	0011
7	0111	1010	1001	0001
8	1000	1011	1000	1001
9	1001	1100	1111	1000

8421 BCD Codes

- **Use 4 bits to encode the first ten values from 0 – 9 by their binary form.**

Decimal	0	1	2	3	4	5	6	7	8	9
8421	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001

  - **This code is the simplest, most intuitive binary code for decimal digits.**
- **BCD is a *weighted* code, and 8, 4, 2, 1 are weights**
- **Example: $(93)_{(10)} = (1001\ 0011)_{8421BCD}$**
- **How many "invalid" code words are there?**
- **What are the "invalid" code words?**

8421 BCD Addition

- **Given a BCD code, we use binary arithmetic to add the digits:**

8	1000	Eight
+5	+0101	Plus 5
13	1101	is 13 (> 9)

Note that the result is MORE THAN 9, so must be represented by two digits!

- **To correct the digit, subtract 10 by adding 6 modulo 16.**

8	1000	Eight
+5	+0101	Plus 5
13	1101	is 13 (> 9)
	+0110	so add 6
carry = 1	0011	leaving 3 + cy
0001 |	0011	Final answer (two digits)

- **If the digit sum is > 9, add one to the next significant digit**

BCD Addition Examples

- **Add 2905_{BCD} to 1897_{BCD} showing carries and digit corrections.**

1	**1**	**1**	**0**
0001	**1000**	**1001**	**0111**
+ 0010	**1001**	**0000**	**0101**
0100	**10010**	**1010**	**1100**
+0000	**+0110**	**+0110**	**+0110**
0100	**1000**	**0000**	**0010**

Excess 3 Code and 8, 4, –2, –1 Code

Decimal	Excess 3	8, 4, –2, –1
0	0011	0000
1	0100	0111
2	0101	0110
3	0110	0101
4	0111	0100
5	1000	1011
6	1001	1010
7	1010	1001
8	1011	1000
9	1100	1111

- **What interesting property is common to these two codes?**

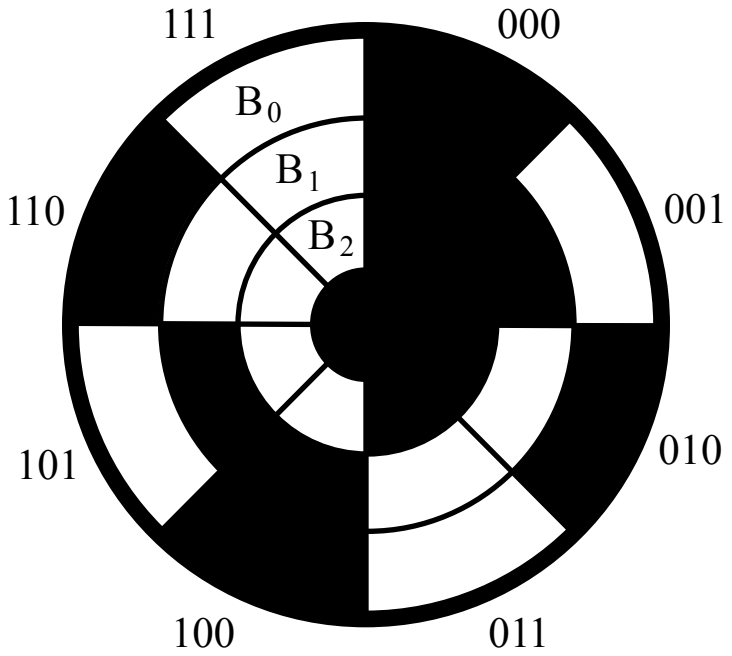
GRAY CODE – Decimal

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

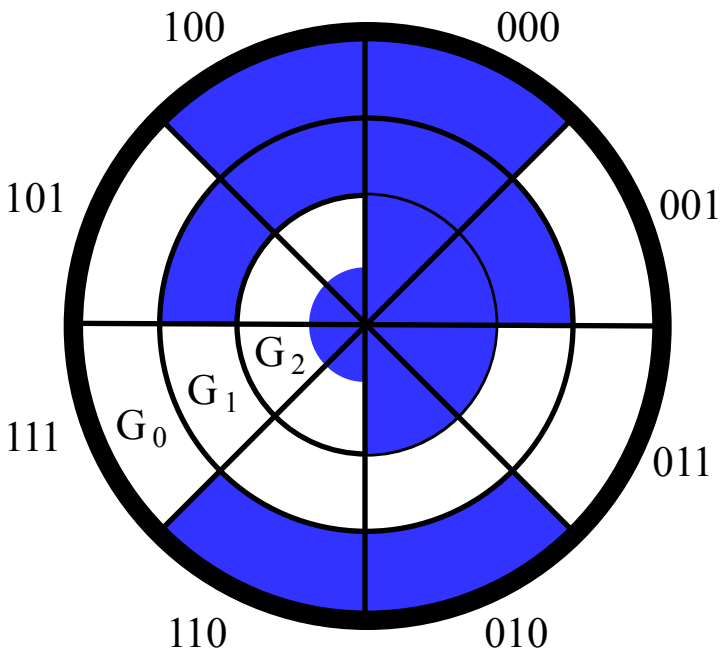
- **What special property does the Gray code have in relation to adjacent decimal digits?**

Optical Shaft Encoder

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



(a) Binary Code for Positions 0 through 7



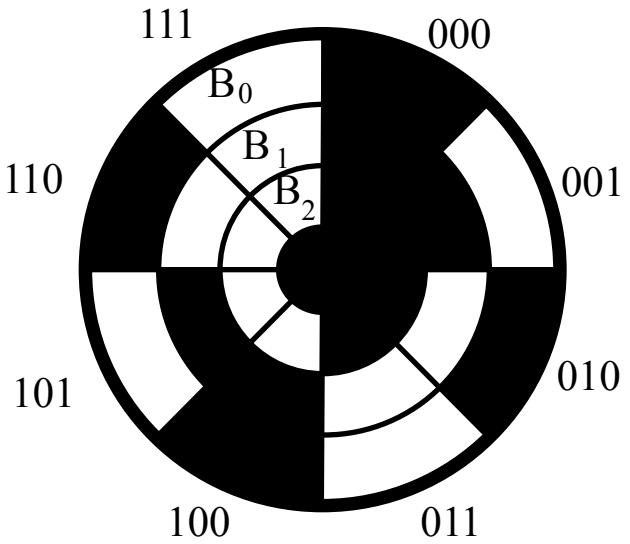
(b) Gray Code for Positions 0 through 7

Shaft Encoder (Continued)

- **For the binary code, what codes may be produced if the shaft position lies between codes for 3 and 4 (011 and 100)?**
- **Answer:**

  011 100 000 010

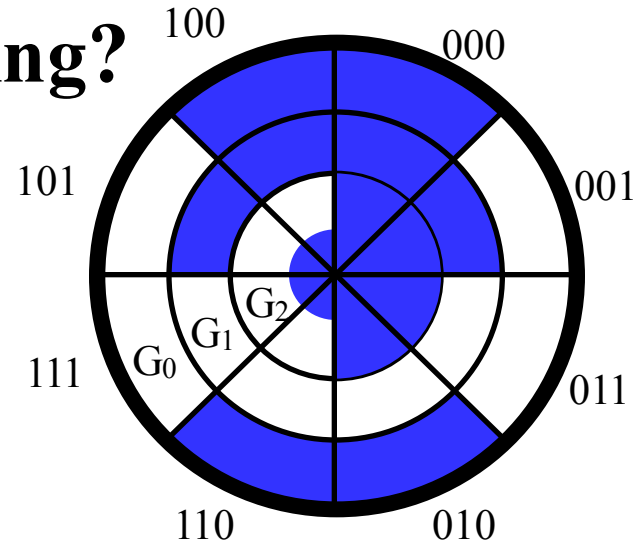
  001 110 101 111
- **Is this a problem?**



(a) Binary Code for Positions 0 through 7

Shaft Encoder (Continued)

- **For the Gray code, what codes may be produced if the shaft position lies between codes for 3 and 4 (010 and 110)? Is this a problem?**
- **Does the Gray code function correctly for these borderline shaft positions for all cases encountered in octal counting?**



(b) Gray Code for Positions 0 through 7

Conversion btw binary and gray code

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

Parity Bit Error-Detection Codes

- **Redundancy (e.g. extra information), in the form of extra bits, can be incorporated into binary code words to detect and correct errors.**
- **A simple form of redundancy is parity, an extra bit appended onto the code word to make the number of 1's odd or even. Parity can detect all single-bit errors and some multiple-bit errors.**
- **A code word has even parity if the number of 1's in the code word is even.**
- **A code word has odd parity if the number of 1's in the code word is odd.**

4-Bit Parity Code Example

- **Fill in the even and odd parity bits:**

Even Parity		Odd Parity	
Message	Parity	Message	Parity
000	0	000	1
001	1	001	0
010	1	010	0
011	0	011	1
100	-	100	-
101	-	101	-
110	-	110	-
111	-	111	-

- **The codeword "1111" has <u>even parity</u> and the codeword "1110" has <u>odd parity</u>. Both can be used to represent 3-bit data.**

Warning: Conversion or Coding?

- **Do NOT mix up conversion of a decimal number to a binary number with coding a decimal number with a BINARY CODE.**
- **$13_{10} = 1101_2$ (This is conversion)**
- **13 $\Leftrightarrow$ 0001\|0011 (This is coding)**

ALPHANUMERIC CODES - ASCII Character Codes

- **American Standard Code for Information Interchange (Refer to Table 1 -5 in the text)**
- **This code is a popular code used to represent information sent as character-based data. It uses 7-bits to represent:**
  - **94 Graphic printing characters.**
  - **34 Non-printing characters**
- **Some non-printing characters are used for text format (e.g. BS = Backspace, CR = carriage return)**
- **Other non-printing characters are used for record marking and flow control (e.g. STX and ETX start and end text areas).**

ASCII Properties

ASCII has some interesting properties:

Digits 0 to 9 span Hexadecimal values 30_{16} to 39_{16}.

Upper case A -Z span 41_{16} to $5A_{16}$.

Lower case a -z span 61_{16} to $7A_{16}$

- **Lower to upper case translation (and vice versa) occurs by flipping bit 6.**

Delete (DEL) is all bits set, a carryover from when punched paper tape was used to store messages. Punching all holes in a row erased a mistake!

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**

Assignment

- **1-3, 1-9, 1-12, 1-13, 1-16, 1-18, 1-19, 1-28**