Introduction to Digital Logic

Prof. Nizamettin AYDIN

naydin@yildiz.edu.tr

naydin@ieee.org

1

Course Outline

1. Digital Computers, Number Systems, Arithmetic Operations, Decimal, Alphanumeric, and Gray Codes
2. Binary Logic, Gates, Boolean Algebra, Standard Forms
3. Circuit Optimization, Two-Level Optimization, Map Manipulation, Multi-Level Circuit Optimization
4. Additional Gates and Circuits, Other Gate Types, Exclusive-OR Operator and Gates, High-Impedance Outputs
5. Implementation Technology and Logic Design, Design Concepts and Automation, The Design Space, Design Procedure, The major design steps
6. Programmable Implementation Technologies: Read-Only Memories, Programmable Logic Arrays, Programmable Array Logic,Technology mapping to programmable logic devices
7. Combinational Functions and Circuits
8. Arithmetic Functions and Circuits
9. Sequential Circuits Storage Elements and Sequential Circuit Analysis
10. Sequential Circuits, Sequential Circuit Design State Diagrams, State Tables
11. Counters, register cells, buses, & serial operations
12. Sequencing and Control, Datapath and Control, Algorithmic State Machines (ASM)
13. Memory Basics

2

Introduction to Digital Logic

Lecture 1

Digital Computers and Information

3

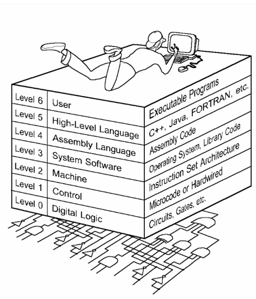
Overview

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

4

The Computer Level Hierarchy

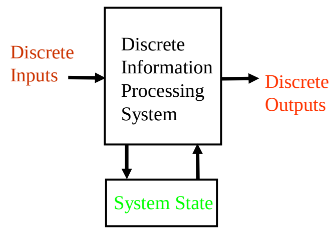
- **Each virtual machine layer is an abstraction of the level below it.**
- **The machines at each level execute their own particular instructions, calling upon machines at lower levels to perform tasks as required.**
- **Computer circuits ultimately carry out the work.**



5

Digital System

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



6

Types of Digital Systems

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

    => Synchronous Sequential System
  - State updated at any time

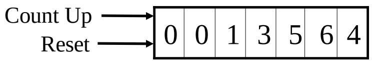
    =>Asynchronous Sequential System
  - State = Function (State, Input)
  - Output = Function (State)

    or Function (State, Input)

7

Digital System Example:

A Digital Counter

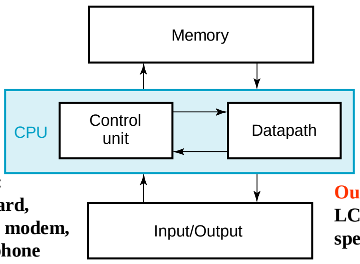


Inputs: Count Up, Reset
Outputs: Visual Display
State: "Value" of stored digits

Synchronous or Asynchronous?

8

A Digital Computer Example



Inputs: Keyboard, mouse, modem, microphone

Outputs: CRT, LCD, modem, speakers

Synchronous or Asynchronous?

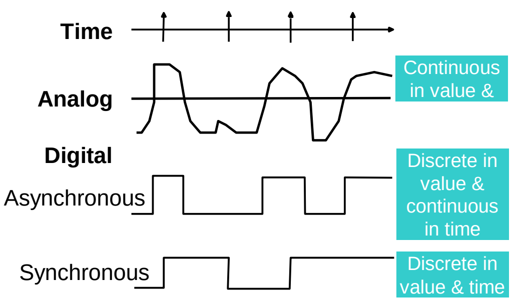
9

Signal

- An information variable represented by physical quantity.
- For digital systems, the variable takes 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

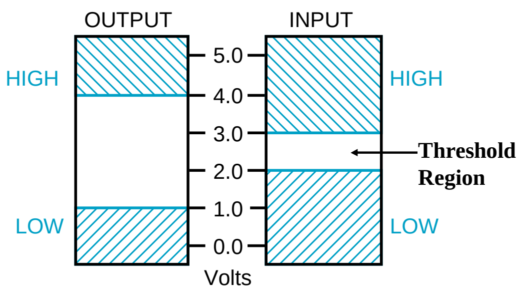
10

Signal Examples Over Time



11

Signal Example – Physical Quantity: Voltage



12

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**

13

Number Systems – Representation

- Positive radix, positional number systems
- A number with *radix* r is represented by a string of digits:

$$A_{n-1}A_{n-2}\dots A_1A_0 \cdot A_{-1}A_{-2}\dots A_{-m+1}A_{-m}$$

in which $0 \le A_i < r$ and $\cdot$ 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)

14

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

15

Special Powers of 2

- **2^{10} (1024) is Kilo, denoted "K"**
- **2^{20} (1,048,576) is Mega, denoted "M"**
- **2^{30} (1,073, 741,824) is Giga, denoted "G"**
- **2^{40} (1,099,511,627,776) is Tera, denoted "T"**

16

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

17

Converting Binary to Decimal

- **To convert to decimal, use decimal arithmetic to form Σ (digit × respective power of 2).**
- **Example:Convert 11010_2 to N_{10}:**

$$1\times 2^4 + 1\times 2^3 + 0\times 2^2 + 1\times 2^1 + 0\times 2^0 = 26$$

18

Converting Decimal to Binary

- Method 1
  - Subtract the largest power of 2 that gives a positive remainder and record the power.
  - Repeat, subtracting from the prior remainder and recording the power, until the remainder is zero.
  - Place 1's in the positions in the binary result corresponding to the powers recorded; in all other positions place 0's.
- Example: Convert 625_{10} to N_2
  - $625 - 512 = 113 = N_1$ $\quad 512 = 2^9$
  - $113 - 64 = 49 = N_2$ $\quad 64 = 2^6$
  - $49 - 32 = 17 = N_3$ $\quad 32 = 2^5$
  - $17 - 16 = 1 = N_4$ $\quad 16 = 2^4$
  - $1 - 1 = 0 = N_5$ $\quad 1 = 2^0$

$(625)_{10} = 1 \cdot 2^9 + 0 \cdot 2^8 + 0 \cdot 2^7 + 1 \cdot 2^6 + 1 \cdot 2^5 + 1 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0$

$= (1001110001)_2$

Commonly Occurring Bases

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 (in addition to the 10 integers) in hexadecimal represent:

A→10, B→11, C→12, D→13, E→14, F→15

Numbers in Different Bases

- Good idea to memorize!

Decimal (Base 10)	Binary (Base 2)	Octal (Base 8)	Hexadecimal (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

Conversion Between Bases

- Method 2
- To convert from one base to another:
  1) Convert the Integer Part
  2) Convert the Fraction Part
  3) Join the two results with a radix point

Conversion Details

- To Convert the Integer Part:

  Repeatedly divide the number by the new radix and save the remainders. The digits for the new radix are the remainders in *reverse order* of their computation. If the new radix is > 10, then convert all remainders > 10 to digits A, B, …
- To Convert the Fractional Part:

  Repeatedly multiply the fraction by the new radix and save the integer digits that result. The digits for the new radix are the integer digits in *order* of their computation. If the new radix is > 10, then convert all integers > 10 to digits A, B, …

Example: Convert 46.6875_{10} To Base 2

- Convert 46 to Base 2:
  - $(101110)_2$
- Convert 0.6875 to Base 2:
  - $(0.1011)_2$
- Join the results together with the radix point:
  - $(101110.1011)_2$

Additional Issue - Fractional Part

- Note that in the conversion in the previous slide, the fractional part became 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: 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.

25

Checking the Conversion

- To convert back, sum the digits times their respective powers of *r*.
- From the prior conversion of 46.6875_{10}

$101110_2 = 1{\cdot}2^5 + 0{\cdot}2^4 + 1{\cdot}2^3 + 1{\cdot}2^2 + 1{\cdot}2^1 + 0{\cdot}2^0$

$=1{\cdot}32 + 0{\cdot}16 + 1{\cdot}8 + 1{\cdot}4 + 1{\cdot}2 + 0{\cdot}1$

$= 32 + 8 + 4 + 2$

$= 46$

$0.1011_2 = 1/2 + 1/8 + 1/16$

$= 0.5000 + 0.1250 + 0.0625$

$= 0.6875$

26

Octal to Binary and Back

- Octal to Binary:
  - Restate the octal as three binary digits starting at the radix point and going both ways.
- Binary to Octal:
  - Group the binary digits into three 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 digit.

27

Hexadecimal to Binary and Back

- Hexadecimal to Binary:
  - Restate the hexadecimal as four binary digits starting at the radix point and going both ways.
- Binary to Hexadecimal:
  - Group the binary digits into 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 four bits to a hexadecimal digit.

28

Octal to Hexadecimal via Binary

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

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

$(\underline{110}\ \underline{011}\ \underline{101}\ .\ \underline{001}\ \underline{111}\ \underline{111})_2$

$(\underline{0001}\ \underline{1001}\ \underline{1101}\ .\ \underline{0011}\ \underline{1111}\ \underline{1000})_2$

$(1\ \ 9\ \ D\ .\ 3\ \ F\ \ 8)_{16}$

- Why do these conversions work?

29

A Final Conversion Note

- You can use arithmetic in other bases if you are careful:
- Example: Convert 101110_2 to Base 10 using binary arithmetic:

Step 1 101110 / 1010 = 100 r 0110

Step 2 100 / 1010 = 0 r 0100

Converted Digits are 0100_2 | 0110_2

or $(4\ \ 6)_{10}$

30

Binary Numbers and Binary Coding

- Flexibility of representation
  - Within constraints below, can assign any binary combination (called a code word) to any data as long as data is uniquely encoded.
- Information Types
  - 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

31

Non-numeric Binary Codes

- Given n binary digits (called bits), a binary code is a mapping from a set of represented elements to a subset of the 2^n binary numbers.
- Example: A binary code for the seven colors of the rainbow
- Code 100 is not used

Color	Binary Number
Red	000
Orange	001
Yellow	010
Green	011
Blue	101
Indigo	110
Violet	111

32

Number of Bits Required

- Given M elements to be represented by a binary code, the minimum number of bits, n, needed, satisfies the following relationships:

  $2^n > M > 2^{(n-1)}$

  $n = \lceil \log_2 M \rceil$ where $\lceil x \rceil$, called the *ceiling function,* is the integer greater than or equal to x.
- Example: How many bits are required to represent decimal digits with a binary code?
  - 4 bits are required ($n = \lceil \log_2 9 \rceil = 4$)

33

Number of Elements Represented

S1-2-28Eylul

- Given n digits in radix r, there are r^n distinct elements that can be represented.
- But, you can represent m elements, $m < r^n$
- Examples:
  - You can represent 4 elements in radix r = 2 with n = 2 digits: (00, 01, 10, 11).
  - You can represent 4 elements in radix r = 2 with n = 4 digits: (0001, 0010, 0100, 1000).
  - This second code is called a "one hot" code.

34

Binary Codes for Decimal Digits

- The usual way of expressing a decimal number in terms of a binary number is known as *pure binary coding*
- There are over 8,000 ways that you can chose 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	0001
2	0010	0101	0110	0011
3	0011	0110	0101	0010
4	0100	0111	0100	0110
5	0101	1000	1011	0111
6	0110	1001	1010	0101
7	0111	1010	1001	0100
8	1000	1011	1000	1100
9	1001	1100	1111	1101

35

Binary Coded Decimal (BCD)

- In the 8421 Binary Coded Decimal (BCD) representation each decimal digit is converted to its 4-bit pure binary equivalent
- This code is the simplest, most intuitive binary code for decimal digits and uses the same powers of 2 as a binary number, but only encodes the first ten values from 0 to 9.

  For example: $(57)_{dec} \rightarrow (?)_{bcd}$

  $(\ 5 \quad 7\ )_{dec}$

  $= (\underline{0101}\ \underline{0111})_{bcd}$

36

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?**

37

Binary to Gray Code Conversion

- **What special property does the Gray code have in relation to adjacent decimal digits?**
- To convert binary to a Gray-coded number then follow this method :

1. **The binary number and the Gray-coded number will have the same number of bits**
2. **The Gray code MSB (left-hand bit) and binary MSB will always be the same**
3. **To get the Gray code next-to-MSB (i.e. next digit to the right) add the binary MSB and the binary next-to-MSB. Record the sum, ignoring any carry.**
4. **Continue in this manner right through to the end.**

Decimal	8421	Gray
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

38

Gray Code to Binary conversion

- To convert a Gray-coded number to binary then follow this method :

1. **The binary number and the Gray-coded number will have the same number of bits**
2. **The binary MSB (left-hand bit) and Gray code MSB will always be the same**
3. **To get the binary next-to-MSB (i.e. next digit to the right) add the binary MSB and the gray code next-to-MSB. Record the sum, ignoring any carry.**
4. **Continue in this manner right through to the end.**

Decimal	8421	Gray
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

39

Binary to Gray Conversion Example

- **Question:** Convert the binary number **11101101** to its Gray code equivalent.
- **Answer:**

Binary	11101101	**Gray CodeDigit 1**	= 1	(same as binary)
Binary	11101101	**Gray Code Digit 2**	= 1 + 1 = 0	(carry 1)
Binary	11101101	**Gray Code Digit 3**	= 1 + 1 = 0	(carry 1)
Binary	11101101	**Gray Code Digit 4**	= 1 + 0 = 1	
Binary	11101101	**Gray Code Digit 5**	= 0 + 1 = 1	
Binary	11101101	**Gray Code Digit 6**	= 1 + 1 = 0	(carry 1)
Binary	11101101	**Gray Code Digit 7**	= 1 + 0 = 1	
Binary	11101101	**Gray Code Digit 8**	= 0 + 1 = 1	

$\mathbf{11101101_{bin} = 10011011_{gray}}$

40

Gray to Decimal Conversion

- **Question:** Convert the Gray coded number **10011011** to its binary equivalent.
- **Answer:**

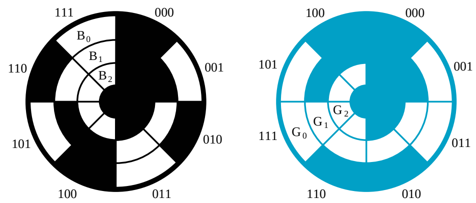
Gray Code	10011011	**Binary Digit 1**	= 1	(same as Gray code)
Gray Code	10011011	**Binary Digit 2**	= 0 + 1 = 1	
Gray Code	10011011	**Binary Digit 3**	= 0 + 1 = 1	
Gray Code	10011011	**Binary Digit 4**	= 1 + 1 = 0	(carry 1)
Gray Code	10011011	**Binary Digit 5**	= 1 + 0 = 1	
Gray Code	10011011	**Binary Digit 6**	= 0 + 1 = 1	
Gray Code	10011011	**Binary Digit 7**	= 1 + 1 = 0	(carry 1)
Gray Code	10011011	**Binary Digit 8**	= 1 + 0 = 1	

$\mathbf{10011011_{gray} = 11101101_{bin}}$

41

Gray Code (Continued)

- **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

42

Gray Code (Continued)

- **How does the shaft encoder work?**
- **For the binary code, what codes may be produced if the shaft position lies between codes for 3 and 4 (011 and 100)?**
- **Is this a problem?**

43

Gray Code (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?**

44

4221 BCD Code

- The 4221 BCD code is another binary coded decimal code where each bit is weighted by 4, 2, 2 and 1 respectively. Unlike BCD coding there are no invalid representations.
- The 1's complement of a 4221 representation is important in decimal arithmetic. In forming the code remember the following rules
- Below decimal 5 use the right-most bit representing 2 first
- Above decimal 5 use the left-most bit representing 2 first
- Decimal 5 = 2+2+1 and not 4+1

Decimal	4221	1's complement
0	0000	1111
1	0001	1110
2	0010	1101
3	0011	1100
4	1000	0111
5	0111	1000
6	1100	0011
7	1101	0010
8	1110	0001
9	1111	0000

45

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)**

46

Binary Arithmetic

- **Single Bit Addition with Carry**
- **Multiple Bit Addition**
- **Single Bit Subtraction with Borrow**
- **Multiple Bit Subtraction**
- **Multiplication**
- **BCD Addition**

47

Single Bit Binary Addition with Carry

Given two binary digits (X,Y), a carry in (Z) we get the following sum (S) and carry (C):

Carry in (Z) of 0:

Z	0	0	0	0
X	0	0	1	1
+ Y	+ 0	+ 1	+ 0	+ 1
C S	0 0	0 1	0 1	1 0

Carry in (Z) of 1:

Z	1	1	1	1
X	0	0	1	1
+ Y	+ 0	+ 1	+ 0	+ 1
C S	0 1	1 0	1 0	1 1

48

Multiple Bit Binary Addition

- Extending this to two multiple bit examples:

Carries	00000	10110 0
Augend	01100	10110
Addend	+10001	+10111
Sum	11101	101101

- Note: The 0 is the default Carry-In to the least significant bit.

49

Single Bit Binary Subtraction with Borrow

- Given two binary digits (X,Y), a borrow in (Z) we get the following difference (S) and borrow (B):
- Borrow in (Z) of 0:

Z	0	0	0	0
X	0	0	1	1
- Y	-0	-1	-0	-1
BS	0 0	1 1	0 1	0 0

- Borrow in (Z) of 1:

Z	1	1	1	1
X	0	0	1	1
- Y	-0	-1	-0	-1
BS	11	1 0	0 0	1 1

50

Multiple Bit Binary Subtraction

- Extending this to two multiple bit examples:

Borrows	00000	0011 0
Minuend	10110	10110
Subtrahend	- 10010	- 10011
Difference	00100	00011

- Notes: The 0 is a Borrow-In to the least significant bit. If the Subtrahend > the Minuend, interchange and append a – to the result.

51

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

52

BCD Arithmetic

- 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

53

BCD Addition Example

- Addition is analogous to decimal addition with normal binary addition taking place from right to left. For example,

6 0110 BCD for 6	42 0100 0010 BCD for 42
+3 0011 BCD for 3	+27 0010 0111 BCD for 27
1001 BCD for 9	0110 1001 BCD for 69

54

BCD Addition Example-2

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

	0111	0011	0011	1110
	0001	1000	1001	0111
+	0010	1001	0000	0101
	0100	10010	1010	1100
		0110	0110	0110
		1000	10000	10010
	4	8	0	2

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

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

ASCII Character Codes

- American Standard Code for Information Interchange (Refer to Table 1-4 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

Data types

- Our first requirement is to find a way to represent information (data) in a form that is mutually comprehensible by human and machine.
  - Ultimately, we will have to develop schemes for representing all conceivable types of information - language, images, actions, etc.
  - We will start by examining different ways of representing *integers*, and look for a form that suits the computer.
  - Specifically, the devices that make up a computer are switches that can be on or off, i.e. at high or low voltage. Thus they naturally provide us with two symbols to work with: we can call them *on* & *off*, or (more usefully) *0* and *1*.

61

Decimal Numbers

- "decimal" means that we have ten digits to use in our representation (the symbols 0 through 9)
- What is 3546?
  - it is *three* thousands plus *five* hundreds plus *four* tens plus *six* ones.
  - i.e. $3546 = 3.10^3 + 5.10^2 + 4.10^1 + 6.10^0$
- How about negative numbers?
  - we use two more symbols to distinguish positive and negative:

    \+ and -

62

Unsigned Binary Integers

$Y = \text{"abc"} = a.2^2 + b.2^1 + c.2^0$

(where the digits a, b, c can each take on the values of 0 or 1 only)

N = number of bits

Range is:
$0 \leq i < 2^N - 1$

	3-bits	5-bits	8-bits
0	000	00000	00000000
1	001	00001	00000001
2	010	00010	00000010
3	011	00011	00000011
4	100	00100	00000100

Problem:
- How do we represent *negative* numbers?

63

Signed Magnitude

- Leading bit is the sign bit

$Y = \text{"abc"} = (-1)^a (b.2^1 + c.2^0)$

Range is:
$-2^{N-1} + 1 < i < 2^{N-1} - 1$

Problems:
- How do we do addition/subtraction?
- We have two numbers for zero (+/-)!

-4	10100
-3	10011
-2	10010
-1	10001
-0	10000
+0	00000
+1	00001
+2	00010
+3	00011
+4	00100

64

One's Complement

- Invert all bits

If msb (most significant bit) is 1 then the number is negative (same as signed magnitude)

Range is:
$-2^{N-1} + 1 < i < 2^{N-1} - 1$

Problems:
- Same as for signed magnitude

-4	11011
-3	11100
-2	11101
-1	11110
-0	11111
+0	00000
+1	00001
+2	00010
+3	00011
+4	00100

65

Two's Complement

- Transformation
  - To transform a into -a, invert all bits in a and add 1 to the result

Range is:
$-2^{N-1} < i < 2^{N-1} - 1$

Advantages:
- Operations need not check the sign
- Only one representation for zero
- Efficient use of all the bits

-16	10000
…	…
-3	11101
-2	11110
-1	11111
0	00000
+1	00001
+2	00010
+3	00011
…	…
+15	01111

66

Limitations of integer representations

- Most numbers are not integer!
  - Even with integers, there are two other considerations:
- Range:
  - The magnitude of the numbers we can represent is determined by how many bits we use:
    - e.g. with 32 bits the largest number we can represent is about +/- 2 billion, far too small for many purposes.
- Precision:
  - The exactness with which we can specify a number:
    - e.g. a 32 bit number gives us 31 bits of precision, or roughly 9 figure precision in decimal repesentation.
- We need another data type!

67

Real numbers

- Our decimal system handles non-integer *real* numbers by adding yet another symbol - the decimal point (**.**) to make a *fixed point* notation:
  - e.g. $3456.78 = 3.10^3 + 4.10^2 + 5.10^1 + 6.10^0 + 7.10^{-1} + 8.10^{-2}$
- The *floating point*, or scientific, notation allows us to represent very large and very small numbers (integer or real), with as much or as little precision as needed:
  - Unit of electric charge $e = 1.602\ 176\ 462 \times 10^{-19}$ Coulomb
  - Volume of universe = 1×10^{85} cm^3
    - the two components of these numbers are called the mantissa and the exponent

68

Real numbers in binary

- We mimic the decimal floating point notation to create a "hybrid" binary floating point number:
  - We first use a "binary point" to separate whole numbers from fractional numbers to make a fixed point notation:
    - e.g. $00011001.110 = 1.2^4 + 1.10^3 + 1.10^1 + 1.2^{-1} + 1.2^{-2} \Rightarrow 25.75$
      ($2^{-1} = 0.5$ and $2^{-2} = 0.25$, etc.)
  - We then "float" the binary point:
    - $00011001.110 \Rightarrow 1.1001110 \times 2^4$
      mantissa = 1.1001110, exponent = 4
  - Now we have to express this without the extra symbols (x, 2, .)
    - by convention, we divide the available bits into three fields:
      sign, mantissa, exponent

69

IEEE-754 fp numbers - 1

s	biased exp.	fraction
1	8 bits	23 bits

32 bits:

$$N = (-1)^s \times 1.\mathbf{fraction} \times 2^{(\text{biased exp.} - 127)}$$

- Sign: 1 bit
- Mantissa: 23 bits
  - We "normalize" the mantissa by dropping the leading 1 and recording only its fractional part (why?)
- Exponent: 8 bits
  - In order to handle both +ve and -ve exponents, we add 127 to the actual exponent to create a "biased exponent":
    - 2^{-127} => biased exponent = 0000 0000 (= 0)
    - 2^0 => biased exponent = 0111 1111 (= 127)
    - 2^{+127} => biased exponent = 1111 1110 (= 254)

70

IEEE-754 fp numbers - 2

- Example: Find the corresponding fp representation of 25.75
  - $25.75 \Rightarrow 00011001.110 \Rightarrow 1.1001110 \times 2^4$
  - sign bit = 0 (+ve)
  - normalized mantissa (fraction) = 100 1110 0000 0000 0000 0000
  - biased exponent = 4 + 127 = 131 => 1000 0011
  - so 25.75 => 0 1000 0011 100 1110 0000 0000 0000 0000 => x41CE0000
- Values represented by convention:
  - Infinity (+ and -): exponent = 255 (1111 1111) and fraction = 0
  - NaN (not a number): exponent = 255 and fraction ≠ 0
  - Zero (0): exponent = 0 and fraction = 0
    - note: exponent = 0 => fraction is *de-normalized*, i.e no hidden 1

71

IEEE-754 fp numbers - 3

- Double precision (64 bit) floating point

64 bits:

s	biased exp.	fraction
1	11 bits	52 bits

$$N = (-1)^s \times 1.\mathbf{fraction} \times 2^{(\text{biased exp.} - 1023)}$$

- **Range & Precision:**
  - **32 bit:**
    - mantissa of 23 bits + 1 => approx. 7 digits decimal
    - $2^{+/-127}$ => approx. $10^{+/-38}$
  - **64 bit:**
    - mantissa of 52 bits + 1 => approx. 15 digits decimal
    - $2^{+/-1023}$ => approx. $10^{+/-306}$

72

Another use for bits: Logic

- Beyond numbers
  - *logical variables* can be *true* or *false*, *on* or *off*, etc., and so are readily represented by the binary system.
  - A logical variable A can take the values *false = 0* or *true = 1* only.
  - The manipulation of logical variables is known as Boolean Algebra, and has its own set of operations - which are not to be confused with the arithmetical operations.
  - Some basic operations: NOT, AND, OR, XOR

73

Basic Logic Operations

- Truth Tables of Basic Operations

NOT

A	A'
0	1
1	0

AND

A	B	A.B
0	0	0
0	1	0
1	0	0
1	1	1

OR

A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1

- Equivalent Notations
  - not A = A' = $\overline{A}$
  - A and B = A.B = A∧B = A intersection B
  - A or B = A+B = A∨B = A union B

74

More Logic Operations

XOR

A	B	A⊕B
0	0	0
0	1	1
1	0	1
1	1	0

XNOR

A	B	(A⊕B)'
0	0	1
0	1	0
1	0	0
1	1	1

– Exclusive OR (XOR): either A or B is 1, not both

– A⊕B = A.B' + A'.B

75