

1

Logic Design

Chia-Tso Chao 趙家佐

Dept. of Electronics Engineering

National Chiao Tung University

Welcome to Logic Design Class

2

© Iris H.-R. Jiang

- **Time/location : Tuesday CD and Friday B @ ED 220**
- **Course webpage : <http://tiger.ee.nctu.edu.tw/Course.php>**
- **Office our : Thursday EF (made by appointment)**
- **Office : ED 601, Ext: 31671**
- **Text Book:**
 - ▣ C. H. Roth, Jr. and L. L. Kinney, Fundamentals of Logic Design, 7th edition, Cengage Learning, 2014.
- **Teaching Assistant: (Lab ED 612, ext. 54178)**
 - ▣ Zak (張玟翔), o0000032@yahoo.com.tw
 - ▣ Benjamin (胡鈺邦), a40112a40112@yahoo.com.tw
 - ▣ Victor (洪維澤), victor@ictek.net
- **Grading:**
 - ▣ Quiz 30%
(there are 14 quizzes, but only 12 higher scores are taken)
 - ▣ Midterm 35%
 - ▣ Final 35%

UNIT 1

NUMBER SYSTEMS AND CONVERSIONS



Spring 2011

Number Systems and Conversions

□ Contents

- ▣ Number systems and conversion
- ▣ Binary arithmetic
- ▣ Representation of negative numbers
 - Addition of two's complement numbers
 - Addition of one's complement numbers
- ▣ Binary codes

□ Readings

- ▣ Unit 1.2~1.5

Number Systems (1/2)

- **Positional notation:** Each digit is multiplied by an appropriate **power of base** depending on its **position** in the number
 - ▣ The **point** separates the positive and negative powers of base
 - ▣ e.g., decimal (base 10) numbers

$$\begin{array}{ccccccc} & \uparrow & \uparrow & \uparrow & & \uparrow & \uparrow \\ 9 & 5 & 3 & . & 7 & 8 & _{10} \end{array} = 9 \times 10^2 + 5 \times 10^1 + 3 \times 10^0 + 7 \times 10^{-1} + 8 \times 10^{-2}$$

(2 1 0 . -1 -2): powers of base

- ▣ A positive number N with **base (radix)** R (**positive integer, $R > 1$**):

$$\begin{aligned} N &= (a_4 a_3 a_2 a_1 a_0 . a_{-1} a_{-2} a_{-3})_R \\ &= a_4 \times R^4 + a_3 \times R^3 + a_2 \times R^2 + a_1 \times R^1 + a_0 \times R^0 + a_{-1} \times R^{-1} + a_{-2} \times R^{-2} + a_{-3} \times R^{-3} \end{aligned}$$

- ▣ **Base** is also called **radix**
- ▣ Base is indicated as subscript

- ▣ Q: Why do people use the decimal number system?

Number Systems (2/2)

□ Examples:

▣ Decimal (base 10) numbers

$$953.78_{10} = 9 \times 10^2 + 5 \times 10^1 + 3 \times 10^0 + 7 \times 10^{-1} + 8 \times 10^{-2}$$

▣ Binary (base 2) numbers

$$1011.11_2 = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-2} = 11.75_{10}$$

▣ Octal (base 8) numbers

$$147.3_8 = 1 \times 8^2 + 4 \times 8^1 + 7 \times 8^0 + 3 \times 8^{-1} = 103.375_{10}$$

▣ Hexadecimal (base 16) numbers

$$A2F_{16} = 10 \times 16^2 + 2 \times 16^1 + 15 \times 16^0 = 2607_{10}$$

Common Number Systems

7

© Iris H.-R. Jiang

Name	Decimal	Binary	Octal	Hexadecimal
Radix	10	2	8	16
Digits	0, 1, 2, 3, 4, 5, 6, 7, 8, 9	0, 1	0, 1, 2, 3, 4, 5, 6, 7	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
First 17 positive integers	0	0	0	0
	1	1	1	1
	2	10	2	2
	3	11	3	3
	4	100	4	4
	5	101	5	5
	6	110	6	6
	7	111	7	7
	8	1000	10	8
	9	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
	16	10000	20	10

Conversion of a Decimal Integer

□ **Convert a decimal **integer** to base R using **division**:**

- $N = (a_n a_{n-1} \dots a_2 a_1 a_0)_R = a_n R^n + a_{n-1} R^{n-1} + \dots + a_2 R^2 + a_1 R^1 + a_0$
- $N/R = a_n R^{n-1} + a_{n-1} R^{n-2} + \dots + a_2 R^1 + a_1 = Q_1 \dots \dots \dots \text{remainder } a_0$
- $Q_1/R = a_n R^{n-2} + a_{n-1} R^{n-3} + \dots + a_3 R^1 + a_2 = Q_2 \dots \dots \dots \text{remainder } a_1$
- $Q_2/R = a_n R^{n-3} + a_{n-1} R^{n-4} + \dots + a_3 = Q_3 \dots \dots \dots \text{remainder } a_2$
- ...
- Continued until we finally obtain $\dots \dots \dots \text{remainder } a_n$
- e.g., convert 53_{10} to binary

2	53		
2	26	 remainder = 1 = a_0	(LSB)
2	13	 remainder = 0 = a_1	
2	6	 remainder = 1 = a_2	$53_{10} = 110101_2$
2	3	 remainder = 0 = a_3	
2	1	 remainder = 1 = a_4	
	0	 remainder = 1 = a_5	(MSB)

Conversion of a Decimal Fraction (1/2)

□ **Convert a decimal **fraction** to base R using **multiplication**:**

$$\square F = (.a_{-1}a_{-2}a_{-3}\dots a_{-m})_R = a_{-1}R^{-1} + a_{-2}R^{-2} + a_{-3}R^{-3} + \dots + a_{-m}R^{-m}$$

$$\square FR = a_{-1} + a_{-2}R^{-1} + a_{-3}R^{-2} + \dots + a_{-m}R^{-m+1} = a_{-1} + F_1$$

$$\square F_1R = a_{-2} + a_{-3}R^{-1} + \dots + a_{-m}R^{-m+2} = a_{-2} + F_2$$

integer fraction

$$\square F_2R = a_{-3} + \dots + a_{-m}R^{-m+3} = a_{-3} + F_3$$

□ ...

□ Continued until we finally obtain a sufficient number of digits

□ e.g., convert $.625_{10}$ to binary

$$\begin{array}{r} .625 \\ \times \quad 2 \\ \hline (1) .250 \quad (a_{-1}=1, \text{ MSB}) \\ \times \quad 2 \\ \hline (0) .500 \quad (a_{-2}=0) \\ \times \quad 2 \\ \hline (1) .000 \quad (a_{-3}=1, \text{ LSB}) \end{array}$$

$$.625_{10} = .101_2$$

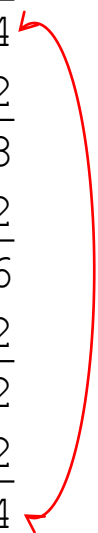
Conversion of a Decimal Fraction (2/2)

10

© Iris H.-R. Jiang

□ Sometimes, the result is a repeating fraction

□ e.g., convert 0.7_{10} to binary


$$\begin{array}{r} .7 \\ \times 2 \\ \hline (1) .4 \\ \times 2 \\ \hline (0) .8 \\ \times 2 \\ \hline (1) .6 \\ \times 2 \\ \hline (1) .2 \\ \times 2 \\ \hline (0) .4 \\ \times 2 \\ \hline (0) .8 \end{array}$$

Start repeating!
.4 was previously obtained
 $0.7_{10} = .1 \underline{0110} \underline{0110} \underline{0110} \dots_2$

Conversion between Two Bases (1/2)

11

© Iris H.-R. Jiang

□ Convert between two bases R_1 and R_2 other than decimal

▣ Base $R_1 \Rightarrow$ base 10 $\Rightarrow$ base R_2

▣ e.g., convert 231.3_4 to base 7

$$231.3_4 = 2 \times 4^2 + 3 \times 4^1 + 1 \times 4^0 + 3 \times 4^{-1} = 45.75_{10}$$

integer

fraction

$$\begin{array}{r} 7 \overline{) 45} \\ 7 \overline{) 6} \quad \text{..... rem. } 3 \\ 0 \quad \text{..... rem. } 6 \\ \Rightarrow 63_7 \end{array}$$

$$\begin{array}{r} .75 \\ \times 7 \\ \hline (5) .25 \\ \times 7 \\ \hline (1) .75 \\ \times 7 \\ \hline (5) .25 \\ \times 7 \\ \hline (1) .75 \end{array} \quad \begin{array}{l} \text{Repeat!} \\ \Rightarrow .\underline{51}_7 \end{array}$$

integer

fraction

$$231.3_4 = 45.75_{10} = 63.\underline{51}_7$$

Conversion between Two Bases (2/2)

12

© Iris H.-R. Jiang

□ Convert between binary and **octal/hexadecimal** by **inspection**

1. Start at the binary point
2. Divide bits into groups of **three/four**, adding 0's if necessary
3. Replace each group by an **octal/hexadecimal** digit
4. And vice versa

■ Binary $\Leftrightarrow$ **octal**

$$1001101.010111_2 = \frac{001}{1} \frac{001}{1} \frac{101}{5} . \frac{010}{2} \frac{111}{7}_2 = 115.27_8$$

■ Binary $\Leftrightarrow$ **hexadecimal**

$$1001101.010111_2 = \frac{0100}{4} \frac{1101}{D} . \frac{0101}{5} \frac{1100}{C}_2 = 4D.5C_{16}$$

← adding 0's →

13

Binary Arithmetic

Binary Arithmetic -- Addition

14

© Iris H.-R. Jiang

□ Addition table:

- $0 + 0 = 0$
- $0 + 1 = 1$
- $1 + 0 = 1$
- $1 + 1 = 0$ (and carry 1 to the next column)
- e.g., add 13_{10} and 11_{10} in binary

$$\begin{array}{r} 1111 \leftarrow \text{carries} \\ 13_{10} = 1101 \\ 11_{10} = 1011 \\ \hline 11000 = 24_{10} \end{array}$$

Binary Arithmetic -- Subtraction

15

© Iris H.-R. Jiang

□ Subtraction table:

- ▣ $0 - 0 = 0$
- ▣ $1 - 0 = 1$
- ▣ $1 - 1 = 0$
- ▣ $0 - 1 = 1$ (and borrow 1 from the next column)
 - Borrow 1 from the next column
 - ≡ have -1 at the next column
 - = obtain R at the current column
- ▣ e.g., subtract 19_{10} from 29_{10} in binary

$$\begin{array}{r} \phantom{29_{10}} 1 \leftarrow \text{borrow} \\ 29_{10} = 11101 \\ 19_{10} = \underline{10011} \\ \phantom{29_{10}} 1010 = 10_{10} \end{array}$$

Binary Arithmetic -- Multiplication

16

© Iris H.-R. Jiang

□ Multiplication table:

- $0 \times 0 = 0$
- $0 \times 1 = 0$
- $1 \times 0 = 0$
- $1 \times 1 = 1$
- e.g., multiply 13_{10} by 11_{10} in binary

$$\begin{array}{r} 1101 = 13_{10} \text{ multiplicand} \\ \times 1011 = 11_{10} \text{ multiplier} \\ \hline 1101 \\ 1101 \leftarrow \text{copy of multiplicand if "1"} \\ 0000 \\ 1101 \\ \hline 10001111 = 143_{10} \end{array}$$

Binary Arithmetic -- Division

17

© Iris H.-R. Jiang

□ Is similar to (easier than) decimal division

▣ e.g., divide 145_{10} by 11_{10} in binary

$$\begin{array}{r} \text{divisor } 11_{10} = \underline{1011} \quad \overline{10010001} = 145_{10} \text{ dividend} \\ \phantom{\text{divisor } 11_{10} = } \underline{1011} \\ \phantom{\text{divisor } 11_{10} = } 1110 \\ \phantom{\text{divisor } 11_{10} = } \underline{1011} \\ \phantom{\text{divisor } 11_{10} = } 1101 \\ \phantom{\text{divisor } 11_{10} = } \underline{1011} \\ \phantom{\text{divisor } 11_{10} = } 10 = \text{remainder} \end{array}$$

$1101 = \text{quotient}$

Representation of Negative Numbers

Addition of two's complement numbers

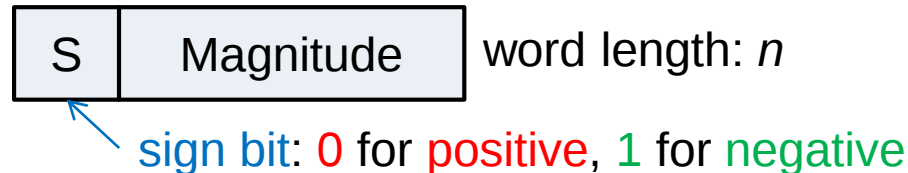
Addition of one's complement numbers

Negative Numbers

19

© Iris H.-R. Jiang

- **Sign and Magnitude:** 1-bit sign + $(n-1)$ -bit magnitude



- e.g., $0011_{sm} = +3_{10}$, $1011_{sm} = -3_{10}$
- Is common for people but awkward for computers
- **1's complement:** $N_- = (2^n - 1) - N$
 - Complement + N bit-by-bit
 - e.g., $+3 = 0011$, $3_- = 1100$
- **2's complement:** $N^* = 2^n - N = N_- + 1$
 - Complement + N bit-by-bit and then add 1
 - Or complement all bits to the left of the rightmost 1
 - e.g., $+3 = 0011$, $3^* = 1101$

Signed Binary Integers

20

© Iris H.-R. Jiang

Word length: $n = 4$

+N	Positive integers (all systems)	-N	Negative integers		
			Sign and magnitude	2's complement N^*	1's complement N_-
+0	0000	-0	1000	----	1111
+1	0001	-1	1001	1111	1110
+2	0010	-2	1010	1110	1101
+3	0011	-3	1011	1101	1100
+4	0100	-4	1100	1100	1011
+5	0101	-5	1101	1011	1010
+6	0110	-6	1110	1010	1001
+7	0111	-7	1111	1001	1000
		-8	----	1000	----

- For word length n , there are 2^n different permutations
 - ▣ SM and N_- : [+7, ..., +0, -0, ..., -7] (redundant -0)
 - ▣ N^* : [+7, ..., +0, ..., -8] ($-2^{n-1} \sim +2^{n-1}-1$)
- View the first bit as the sign bit: **positive/negative** $\Leftrightarrow$ **0/1**

Warning: Unsigned or Signed Numbers

- For word length n , there are 2^n different permutations
 - ▣ Unsigned:
 - $[0, 2^n-1]$
 - ▣ Signed
 - SM (sign and magnitude) and N_- (1's complement) :
 $[-2^{n-1}+1, +2^{n-1}-1]$ (redundant -0)
 - N^* (2's complement) : $[-2^{n-1}, +2^{n-1}-1]$
- e.g., what is 1110?

Addition of Two's Complement Numbers

22

© Iris H.-R. Jiang

1. Add just as if all numbers were positive

2. Discard the carry from the sign bit

□ Why to **discard the carry**? (i.e., **subtract 2^n**)

1. Add($-A$, $+B$), $B > A$

$$\blacksquare A^* + B = (2^n - A) + B = 2^n + (B - A)$$

$$\blacksquare \Rightarrow -2^n \Rightarrow (B - A)$$

2. Add($-A$, $-B$), $A+B \leq 2^{n-1}$

$$\blacksquare A^* + B^* = (2^n - A) + (2^n - B) = 2^n + 2^n - (A + B) = 2^n + (A + B)^*$$

$$\blacksquare \Rightarrow -2^n \Rightarrow (A + B)^*$$

□ e.g.,

Add(+A, +B) $A+B < 2^{n-1}$		Add(+A, -B) $B > A$		Add(-A, +B) $B > A$		Add(-A, -B) $A+B \leq 2^{n-1}$	
+3	0011	+5	0101	-5	1011	-3	1101
+4	0100	-6	1010	+6	0110	-4	1100
+7	0111	-1	1111	+1	(1) 0001	-7	(1) 1001

Exception: Overflow in Two's Complement

23

© Iris H.-R. Jiang

- An **overflow** means out of range of representation
 - ▣ When the word length is n bits, the correct representation of a number (**including sign**) requires more than n bits
- **How to detect overflow?**
 - ▣ Check **sign**!
 - $(+) + (+) \Rightarrow (-)$ or $(-) + (-) \Rightarrow (+)$
- **e.g.,**

Add(+A, +B) $A+B \geq 2^{n-1}$		Add(-A, -B) $A+B > 2^{n-1}$	
+5	0101	-5	1011
+6	0110	-6	1010
	1011	(1)	0101
Wrong answer! Overflow: +11 requires 5 bits including sign		Wrong answer! Overflow: -11 requires 5 bits including sign	

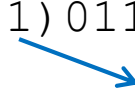
Addition of One's Complement Numbers

24

© Iris H.-R. Jiang

- **End-around carry:** Add the carry out back to the rightmost bit
 - ▣ Why to take **end-around carry**? (i.e., **subtract 2^n & add 1**)
 1. Add($-A$, $+B$), $B > A$
 - $A_ + B = (2^n - 1 - A) + B = 2^n + (B - A) - 1$
 - $\Rightarrow -2^n \text{ \& } +1 \Rightarrow (B - A)$
 2. Add($-A$, $-B$), $A+B \leq 2^{n-1}$
 - $A_ + B_ = (2^n - 1 - A) + (2^n - 1 - B) = 2^n + 2^n - 1 - (A + B) - 1$
 $= 2^n + (A + B)_ - 1$
 - $\Rightarrow -2^n \text{ \& } +1 \Rightarrow (A + B)_$

□ e.g.,

Add(+A, +B) $A+B < 2^{n-1}$		Add(+A, -B) $B > A$		Add(-A, +B) $B > A$		Add(-A, -B) $A+B \leq 2^{n-1}$	
+3	0011	+5	0101	-5	1010	-3	1100
+4	0100	-6	1001	+6	0110	-4	1011
<u>+7</u>	<u>0111</u>	<u>-1</u>	<u>1110</u>	<u>+1</u>	(1) 0000	<u>-7</u>	(1) 0111
					 1		 1
					0001		1000

Exception: Overflow in One's Complement

25

© Iris H.-R. Jiang

- **How to detect overflow?**
 - ▣ Check **sign**!
 - $(+) + (+) \Rightarrow (-)$ or $(-) + (-) \Rightarrow (+)$
- **e.g.,**

Add(+A, +B) $A+B \geq 2^{n-1}$		Add(-A, -B) $A+B \geq 2^{n-1}$	
+5	0101	-5	1010
+6	0110	-6	1001
	1011		(1) 0011
			
			1
			0100
Wrong answer! Overflow: +11 requires 5 bits including sign		Wrong answer! Overflow: -11 requires 5 bits including sign	

26

Binary Codes

Decimal Digits to Binary Codes

27

© Iris H.-R. Jiang

- Input/output interface generally uses decimal numbers
- How to represent decimal numbers using binary codes?

- ▣ Choose 10 elements from 16 binary numbers of 4 bits

- ▣ e.g., 937.25 $\Rightarrow$ BCD: 1001 0011 0111 . 0010 0101

Many options!

Decimal digit	8-4-2-1 code	6-3-1-1 code	Excess-3 code	2-out-of-5 code	Gray code
0	0000	0000	0011	00011	0000
1	0001	0001	0100	00101	0001
2	0010	0011	0101	00110	0011
3	0011	0100	0110	01001	0010
4	0100	0101	0111	01010	0110
5	0101	0111	1000	01100	0111
6	0110	1000	1001	10001	0101
7	0111	1001	1010	10010	0100
8	1000	1011	1011	10100	1100
9	1001	1100	1100	11000	1101
Property	Weighted a.k.a. BCD	Weighted	= BCD+3	Only 2 bits are one $\Rightarrow$ For error checking	Only 1 bit changes as counting up/down $\Rightarrow$ For analog quantities

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**

- e.g.,

$13_{10} \Leftrightarrow 1101_2$ (This is conversion)

$13 \Leftrightarrow 0001\ 0011$ (This is coding)

Text to Binary Codes

29

© Iris H.-R. Jiang

□ ASCII

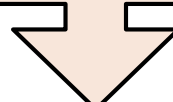
- ▣ American Standard Code for Information Interchange
- ▣ English alphanumeric symbols
- ▣ 7 bits

□ Big-5 code

- ▣ Traditional Chinese characters
- ▣ Double-byte coding (16 bits)



```
!"#$%&'()*+,-./
0123456789:;<=>?
@ABCDEFGHIJKLMNO
PQRSTUVWXYZ[\]^_
`abcdefghijklmnopqrstuvwxyz{|}~
```



94 printable
characters are
numbered 32_{10} to
 126_{10} in ASCII code