

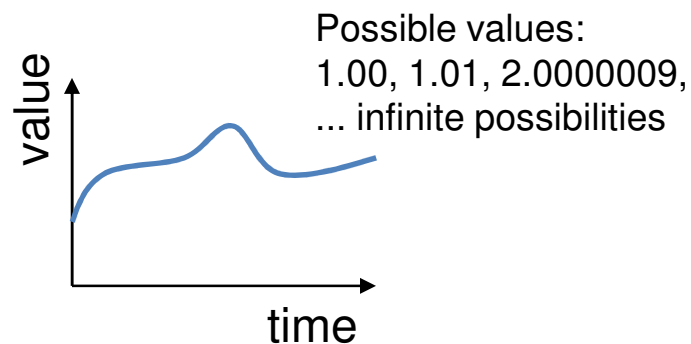
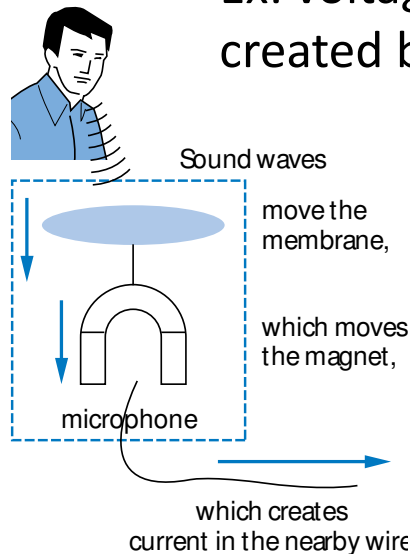
Digital Design 3e, Morris Mano
Chapter 1 – Binary Systems

NUMBER SYSTEMS & DIGITAL LOGIC

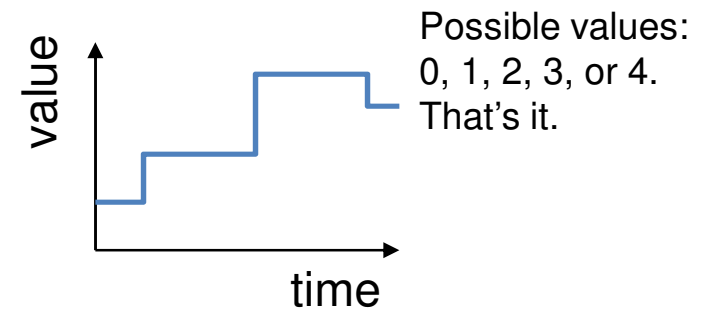
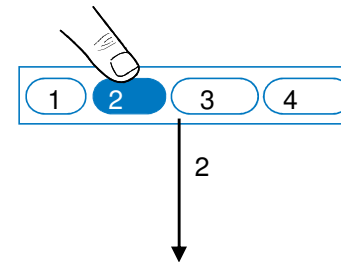
DR. ENG KAMALANATH SAMARAKOON

What Does “digital” Mean?

- Analogue signal
 - Infinite possible values
 - Ex: voltage on a wire created by microphone



- Digital signal
 - Finite possible values
 - Ex: button pressed on a keypad



Benefits of Digitization

- Analogue signal (e.g., audio) may lose quality
 - Voltage levels not saved/copied/transmitted perfectly
- Digitized version enables near-perfect save/cpy/trn.
 - “Sample” voltage at particular rate, save sample using bit encoding
 - Voltage levels still not kept perfectly
 - But we can distinguish 0s from 1s
- Digitized audio can be compressed
 - e.g., MP3s
 - A CD can hold about 20 songs uncompressed, but about 200 compressed
- Compression also done on digitized pictures (jpeg), movies (mpeg), and more
- Digitization has many other benefits too, which you will see later

Binary: Digital Signals with 2 Values

- **Binary** digital signal -- only two possible values
 - Typically represented as **0** and **1**
 - One binary digit is a **bit**
 - **We'll only consider binary digital signals**
 - Binary is popular because
 - Transistors, the basic digital electric component, operate using two voltages (more later)
 - Storing/transmitting one of two values is easier than three or more

Implementing Digital Systems

1. Programming Microprocessors

- Microprocessors a common choice to implement a digital system
- Easy to program
- Cheap (as low as \$1)
- Available now

2. Designing Digital Circuits

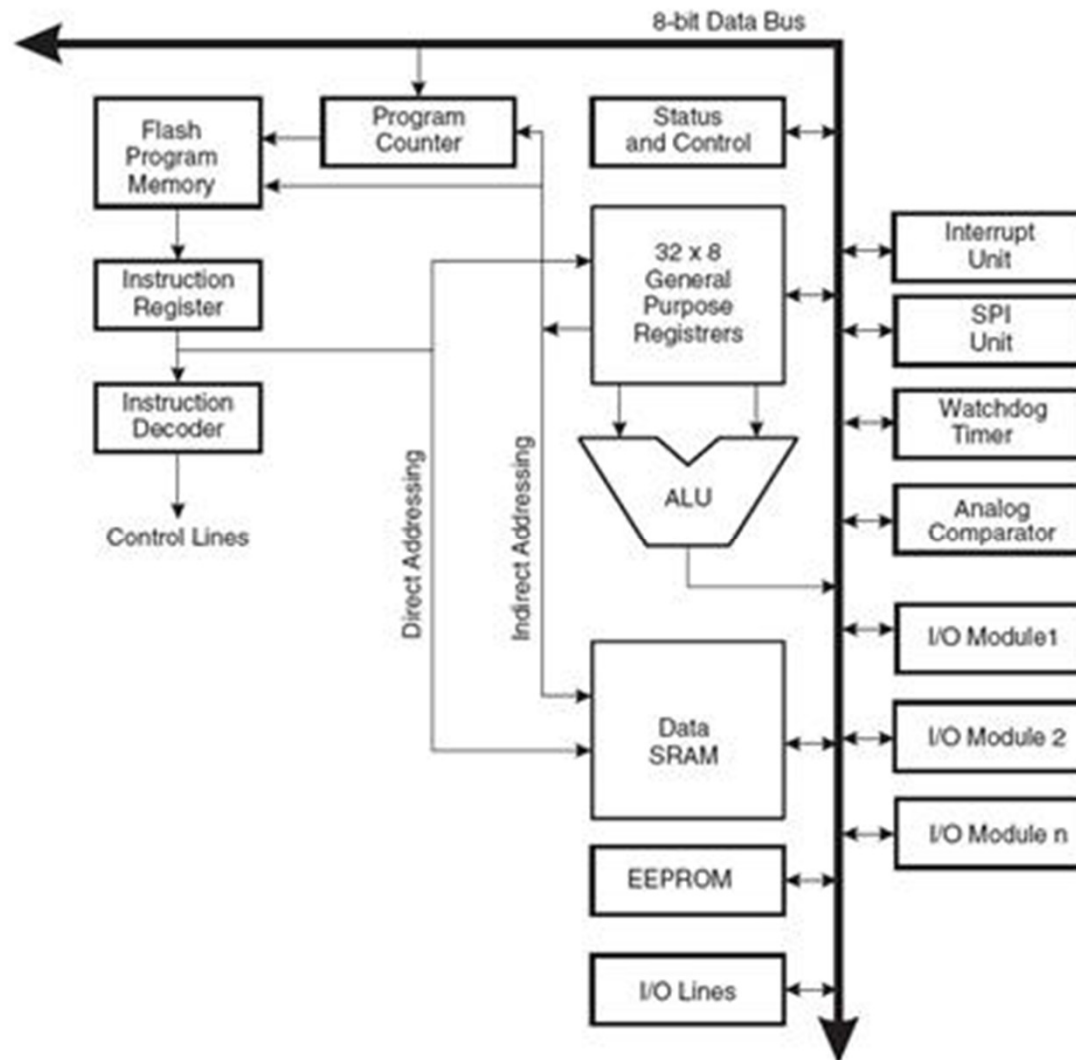
- With microprocessors so easy, cheap, and available, why design a digital circuit?
- Microprocessor may be too slow
- Or too big, power hungry, or costly
- **When Microprocessors Aren't Good Enough**

Implementing Digital Systems (2)

- Sample digital camera task execution times (in seconds) on a microprocessor versus a digital circuit:

Task	Microprocessor	Custom Digital Circuit
Read	5	0.1
Compress	8	0.5
Store	1	0.8

Building block of complex systems such as Microcontrollers and Microprocessors



Topics Covered in CO221

- **Introduction to Digital Logic:**
Digital signals, Digital Logic, Computers and Digital Systems , Purpose and role of digital logic in computer engineering, CMOS logic circuits
- **Number Systems and Digital logic:**
Binary number system, , Number Base Conversions, A Representation of Negative Numbers, Binary arithmetic, Levels of Description of Logic Networks, Basic logic gates, Positive and negative logic

- **Combinational logic circuits:**

Boolean algebra, Boolean laws and theorems, Sum-of-products and Product-of-sums methods, Simplifications of Boolean expressions, Truth tables, Karnaugh Maps, Quine Mc-clusky method, Don't care combinations, Elimination of timing Hazards

-
- Modular design of combinational circuits
 - Introduction of levels of integration
 - Multiplexers, De-multiplexers
 - Encoders and Decoders
 - ROM and PLA

Decimal Numbers

Uses 10 digits [0, 1, 2, . . . 9]

Positional Number Notation

Weight of digit determined by its position.

Example:

$$\begin{aligned} 246 &= 200 + 40 + 6 \\ &= 2 \times 10^2 + 4 \times 10^1 + 6 \times 10^0 \end{aligned}$$

In general:

$$N = \sum N_i \times 10^i \quad \text{where } N_i \in [0, 1, 2, \dots, 9],$$

where N_i are the weights

Note: decimal fractions occur when i is negative

$$0.35 = 3 \times 10^{-1} + 5 \times 10^{-2}$$

Binary Numbers

Uses 2 digits [0 & 1]

Positional Number Notation

Example:

$$\begin{aligned} 111001_2 &= 100000_2 + 10000_2 + 1000_2 + 000_2 + 00_2 + 1_2 \\ &= 1 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 \\ &= 1 \times 32 + 1 \times 16 + 1 \times 8 + 0 \times 4 + 0 \times 2 + 1 \times 1 \\ &= 57_{10} \end{aligned}$$

In general:

$$N = \sum N_i \times 2^i \quad \text{where } N_i \in [0, 1].$$

Note: binary fractions occur when i is negative

$$0.11_2 = 1 \times 2^{-1} + 1 \times 2^{-2} = 0.5_{10} + 0.25_{10} = 0.75_{10}$$

Hexadecimal (Hex) Numbers

Uses 16 digits [0, 1, 2, . . . 9, A, B, C, D, E, F]

Positional Number Notation

Note: A - F represent the decimal values of 10 - 15, respectively.

Example:

$$\begin{aligned} 89AB_{16} &= 8000_{16} + 900_{16} + A0_{16} + B_{16} \\ &= 8 \times 16^3 + 9 \times 16^2 + A \times 16^1 + B \times 16^0 \\ &= 8 \times 4096 + 9 \times 256 + 10 \times 16 + 11 \times 1 = 35243_{10} \end{aligned}$$

In general:

$$N = \sum N_i \times 16^i \quad \text{where } N_i \in [0, 1, 2, \dots, 9, A, B, C, D, E, F].$$

Note: hexadecimal fractions occur when i is negative

$$0.9A_{16} = 9 \times 16^{-1} + 10 \times 16^{-2} = 0.5625_{10} + 0.0390625_{10} = 0.6015625_{10}$$

Exercise

- Determine the base of the numbers in each cases for the following operations to be correct:
 - (a) $14/2 = 5$
 - (b) $54/4 = 13$
 - (c) $24+17 = 40$

Conversion Among Bases

1. In general, with positional number notation and the known decimal weights for each position in any arbitrary base, it is the **easiest to convert other bases to decimal**.
2. This was demonstrated in each previous example where the decimal value was found using the equation for base B:

$$\sum N_i \times B^i \quad \text{where } N_i \in [0, 1, 2, \dots, B-1]$$

and substituting the equivalent decimal weight for the digits N_i and the decimal value of B^i .

Hexadecimal \Leftrightarrow Binary

Example: convert $10011000111001101_2 \Leftrightarrow$ hexadecimal

Group by 4s, starting at the right \Leftrightarrow expand into 4 bit groups

$$1\ 0011\ 0001\ 1100\ 1101_2 \Leftrightarrow 1\ 3\ 1\ C\ D_{16}$$

Hex	Binary	Decimal	Hex	Binary	Decimal
0	0000	0	1	0001	1
2	0010	2	3	0011	3
4	0100	4	5	0101	5
6	0110	6	7	0111	7
8	1000	8	9	1001	9
A	1010	10	B	1011	11
C	1100	12	D	1101	13
E	1110	14	F	1111	15

Decimal \Rightarrow Binary

Successive Division: an easy way to convert Decimal to Binary

Based on the remainders after dividing the decimal number by higher powers of 2.

If the decimal number is even, the corresponding binary number will end in a 0, and if it is odd, the binary number will end in a 1, and so on.

Example:

$57 / 2 = 28$, remainder = **1** (binary number will end with 1)

$28 / 2 = 14$, remainder = **0**

$14 / 2 = 7$, remainder = **0**

$7 / 2 = 3$, remainder = **1**

$3 / 2 = 1$, remainder = **1**

$1 / 2 =$ **0**, remainder = **1** (binary number will start with 1)

Therefore, collecting the remainders, $57_{10} = 111001_2$

$$((((0 \times 2 + 1) \times 2 + 1) \times 2 + 1) \times 2 + 0) \times 2 + 0) \times 2 + 1 = 1 \times 32 + 1 \times 16 + 1 \times 8 + 0 \times 4 + 0 \times 2 + 1 \times 1 = 57$$

(check: $32 + 16 + 8 + 0 + 0 + 1 = 57$)

Decimal \Rightarrow Hex

Successive Division: an easy way to convert Decimal to Hexadecimal

Based on the remainders after dividing the decimal number by higher powers of 16.

Example:

$35243 / 16 = 2202$, remainder = 11 \rightarrow B (hex number will end with B)

$2202 / 16 = 137$, remainder = 10 \rightarrow A

$137 / 16 = 8$, remainder = 9

$8 / 16 = \underline{0}$, remainder = 8 (hex number will start with 8)

Therefore, collecting the remainders, $35243_{10} = 89AB_{16}$

(check: $8 \times 4096 + 9 \times 256 + 10 \times 16 + 11 = 35243$)

Hex \Rightarrow Decimal

Multiplication: an easy way to convert Hexadecimal to Decimal

Multiply the hexadecimal weights by powers of 16.

Example:

$$\text{BA89} \rightarrow \text{B} \times 16^3 + \text{A} \times 16^2 + 8 \times 16 + 9$$

(**CAUTION:** mixed bases)

$$\begin{aligned} &= 11 \times 4096 + 10 \times 256 + 8 \times 16 + 9 \\ &= 45056 + 2560 + 128 + 9 \\ &= 47753 \end{aligned}$$

We can check by converting back to hex:

$$47753 / 16 = 2984 + \text{remainder, } 9$$

$$2984 / 16 = 186 + \text{remainder, } 8$$

$$186 / 16 = 11 + \text{remainder, } 10 \rightarrow \text{A}$$

$$11 / 16 = \underline{0} + \text{remainder, } 11 \rightarrow \text{B}$$

$$\therefore 47753_{10} \rightarrow \text{BA89}_{16}$$

Binary \Rightarrow Decimal

Multiplication: an easy way to convert Binary to Decimal

We can multiply the binary weights by powers of 2.

However, sometimes it is just as easy to convert the binary number to hexadecimal first and then into decimal.

Compare:

$$\begin{aligned} 11010110_2 &\rightarrow 2^7 + 2^6 + 2^4 + 2^2 + 2 \text{ (exploiting the binary weights)} \\ &= 128 + 64 + 16 + 4 + 2 \\ &= 214_{10} \end{aligned}$$

$$11010110_2 \rightarrow D6_{16} \rightarrow 13 \times 16 + 6 = 208 + 6 = 214_{10}$$

Exercise

- Convert the decimal number 345 to binary in two ways :
 - Convert directly to binary
 - Convert first to hexadecimal, then from hexadecimal to binary.
- Which method is faster ?

Exercise

- Do the following conversion problems :
 - (a) Convert decimal 34.4375 to binary.
 - (b) Calculate the binary equivalent of $1/3$ out to 8 places.
 - (c) Then convert from binary to decimal. How close is the result to $1/3$?
 - (d) Convert the binary result in (b) into hexadecimal. Then convert the result to decimal. Is the answer the same ?

Answer (a)

(a)

34.4375

34

0.4375

$$34:2=17 \quad r=0$$

$$17:2=8 \quad r=1$$

$$8:2=4 \quad r=0$$

$$4:2=2 \quad r=0$$

$$2:2=1 \quad r=0$$

$$0.4375*2=0.875 \quad r=0$$

$$0.875*2=1.75 \quad r=1$$

$$0.75*2=1.5 \quad r=1$$

$$0.5*2=1.0 \quad r=1$$

$$0*2=0 \quad r=0$$

$$0.4375=(0.01110)_2$$

$$34=(100010)_2$$

$$34.4375=(100010.01110)_2$$

Binary Arithmetic

- Computers have circuits that do binary arithmetic.
- You already know the rules for decimal addition and subtraction (how to handle sums, carries, differences, and borrows).
- Analogously, we will develop the rules for binary addition and subtraction.

Decimal Addition

Refresher

$$\begin{array}{r} \boxed{\begin{array}{r} 95_{10} \\ + 16_{10} \end{array}} \Rightarrow \begin{array}{r} 9 \times 10^1 + 5 \times 10^0 \\ + 1 \times 10^1 + 6 \times 10^0 \\ \hline 10 \times 10^1 + 11 \times 10^0 \end{array} \\ \downarrow \quad \swarrow \quad \searrow \quad \downarrow \\ 111_{10} = 1 \times 10^2 + (0+1) \times 10^1 + 1 \times 10^0 \end{array}$$

Summary

11 ← Column carries

$$\begin{array}{r} 95_{10} \\ + 16_{10} \\ \hline 111_{10} \end{array}$$

Binary Addition

This table calculates the sum for pairs of binary numbers

$$0 + 0 = 0$$

$$0 + 1 = 1$$

$$1 + 0 = 1$$

$$1 + 1 = 0 \text{ with a carry of } 1$$

Also known as the Half Adder Table

Binary Addition with Carry

This table shows all the possible sums for binary numbers with carries

carry		addend		augend	sum	
0	+	0	+	0	=	0
0	+	0	+	1	=	1
0	+	1	+	0	=	1
0	+	1	+	1	=	0 with a carry of 1
1	+	0	+	0	=	1
1	+	0	+	1	=	0 with a carry of 1
1	+	1	+	0	=	0 with a carry of 1
1	+	1	+	1	=	1 with a carry of 1

Also known as the Full Adder Table

Binary Addition

Similar to the decimal case

Example: Add 5 and 3 in binary

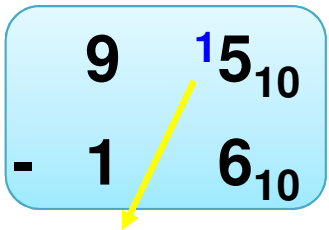
The diagram illustrates the binary addition of 5 and 3. The numbers are aligned by their least significant bits. The first row is 101 (5 in decimal) and the second row is 011 (3 in decimal). A horizontal line separates the addends from the result. The result is 1000 (8 in decimal). Blue arrows and the number '1' indicate the carry propagation from right to left: a carry of 1 is generated from the first column (1+1), carried to the second column; a carry of 1 is generated from the second column (0+1+1), carried to the third column; and a carry of 1 is generated from the third column (1+0+1), carried to the fourth column. The text '(carries)' is written in blue to the right of the arrows.

$$\begin{array}{r} 101_2 = 5_{10} \\ + 011_2 = 3_{10} \\ \hline 1000_2 = 8_{10} \end{array}$$

(carries)

Decimal Subtraction

Refresher


$$\begin{array}{r} 9 \quad 15_{10} \\ - 1 \quad 6_{10} \\ \hline 7 \quad 9_{10} \end{array}$$
$$95 = 9 \times 10^1 + 5 \times 10^0 = 9 \times 10^1 + 15 \times 10^0$$
$$-16 = -1 \times 10^1 + -6 \times 10^0 = -1 \times 10^1 + -6 \times 10^0$$

borrow = -1×10^1

$$\begin{array}{r} 7 \times 10^1 + 9 \times 10^0 \end{array}$$

Note: borrows are shown as explicit subtractions.

Binary Subtraction

This table calculates the difference for pairs of binary numbers

$$0 - 0 = 0$$

$$0 - 1 = 1 \text{ with a borrow of 1}$$

$$1 - 0 = 1$$

$$1 - 1 = 0$$

Also known as the Half Subtractor Table

Binary Subtraction with Borrow

This shows all the possible differences for binary numbers with borrows

minuend subtrahend borrow difference

$$0 - 0 - 0 = 0$$

$$0 - 0 - 1 = 1 \text{ with a borrow of 1}$$

$$0 - 1 - 0 = 1 \text{ with a borrow of 1}$$

$$0 - 1 - 1 = 0 \text{ with a borrow of 1}$$

$$1 - 0 - 0 = 1$$

$$1 - 0 - 1 = 0$$

$$1 - 1 - 0 = 0$$

$$1 - 1 - 1 = 1 \text{ with a borrow of 1}$$

Also known as the Full Subtractor Table

Binary Subtraction

Similar to the decimal case

Example: Subtract 3 from 5 in binary

$$\begin{array}{r} 101_2 = 5_{10} \\ - 11_2 = 3_{10} \\ \hline 010_2 = 2_{10} \end{array}$$

(borrows)

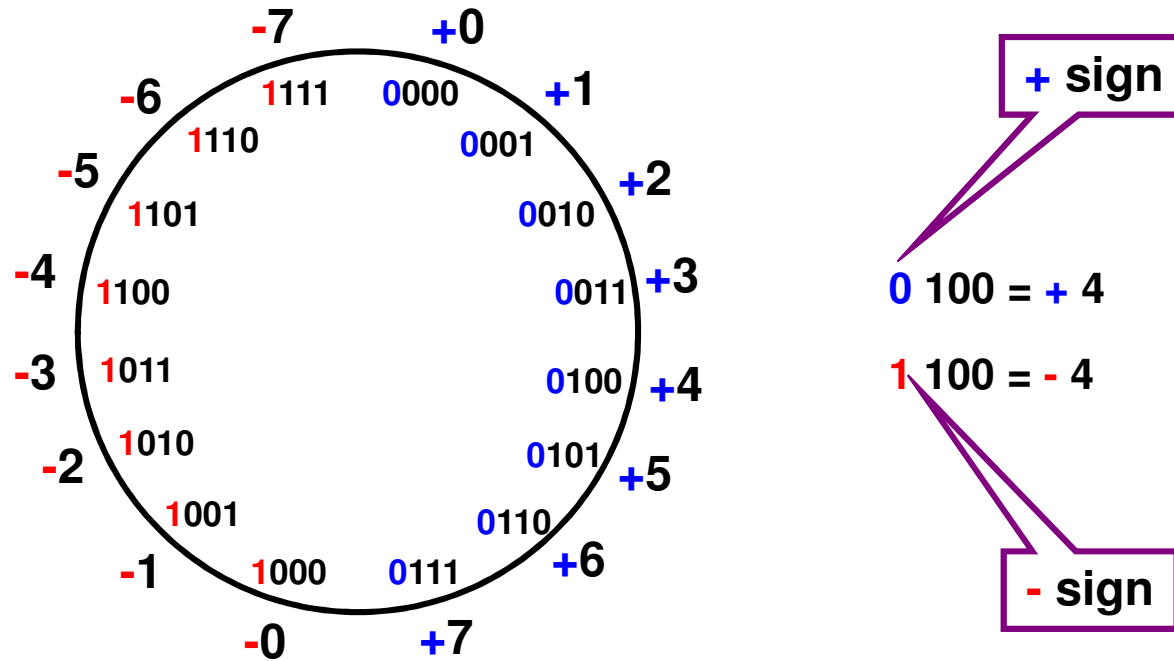
Number Systems

- **Representation of Integers**

- Positive numbers are same in most systems
- Major differences are in how negative numbers are represented
- Three major schemes:
 - (1) sign & magnitude (2) one's complement (3) two's complement
- If we assume a 4 bit machine word
 - 16 different values can be represented
 - roughly half are positive, half are negative

(1) Sign & Magnitude Representation

4 bit example



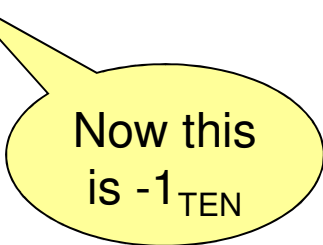
- Sign equals the High order bit: 0 = positive or zero (non-negative)
1 = negative
- Magnitude equals the three low order bits: 0 = 000 thru 7 = 111
- The number range = ± 7 for 4 bit numbers; for n bits, $\pm 2^{n-1} - 1$
- Two Representations for 0 (redundant & problematic)

(2) One's Complement Representation

- Example:
 - $+9_{\text{ten}} \rightarrow 00001001_{\text{two}}$
 - $-9_{\text{ten}} \rightarrow 11110110_{\text{two}}$
- Positive numbers have leading zeros and negative numbers have leading ones
- Shortcomings of one's complement
 - Complicated arithmetic circuit
 - Again two zeros
 - $00000000 \rightarrow +0$
 - $11111111 \rightarrow -0$

(3) Two's Complement Representation

- The **most common representation** for signed binary integers
- The idea came from subtracting a large unsigned number from a small one
 - We would try to borrow from the string of leading zeros and therefore the result would be a string of leading ones
 - $4 - 5 \rightarrow 00000100 - 00000101 = 11111111$



Now this
is -1_{TEN}

(3) Two's complement – Contd.

- Example:

$$+1_{\text{ten}} = 0000\ 0001_{\text{two}}$$

$$\begin{aligned} -1_{\text{ten}} &= \text{inverse}(1_{\text{ten}}) + 1_{\text{two}} \\ &= 1111\ 1110_{\text{two}} + 1_{\text{two}} \\ &= 1111\ 1111_{\text{two}} \end{aligned}$$

$$\begin{aligned} -8_{\text{ten}} &= \text{inverse}(8_{\text{TEN}}) + 1_{\text{two}} \\ &= \text{inverse}(0000\ 1000_{\text{two}}) + 1_{\text{two}} \\ &= 1111\ 0111_{\text{two}} + 1_{\text{two}} \\ &= 1111\ 1000_{\text{two}} \end{aligned}$$

Signed Number System - Summary

- All the 4-bit numbers in the different signed number systems are given
- Positive numbers are the same in all three representations
- Signed magnitude and one's complement have two ways of representing 0. This makes things more complicated
- Two's complement has asymmetric ranges; there is one more negative number than positive number. Here, you can represent -8 but not +8
- However, two's complement is preferred because it has only one 0, and its addition algorithm is the simplest

Decimal	S.M.	1's comp.	2's comp.
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	1000	1111	—
-1	1001	1110	1111
-2	1010	1101	1110
-3	1011	1100	1101
-4	1100	1011	1100
-5	1101	1010	1011
-6	1110	1001	1010
-7	1111	1000	1001
-8	—	—	1000

Binary Codes

- **Binary Coded Decimals (BCD)**
 - decimal numbers are encoded with each digit is represented by its own binary sequence
 - Check http://en.wikipedia.org/wiki/Binary-coded_decimal
- **Gray Code**
 - two successive values differ in only one bit
 - Produces quantities that are continuous
 - Check http://en.wikipedia.org/wiki/Gray_code
- **ASCII Character Code**
 - American Standard Code for Information Interchange
 - character-encoding scheme based on the ordering of the English alphabet
 - Check <http://en.wikipedia.org/wiki/ASCII>

Error Detecting Code

- Using parity bit
 - Usually added to a data transmission or storage to verify the correctness of data
 - Example:
 - Data without parity bit -> 0100 100
 - Data with parity bit -> 0010 0100 (even parity)
 - Data with parity bit -> 1010 0100 (odd parity)
 - Can we correct errors using this parity bit?

Acknowledgement

- Dr. Swarnalatha Radakrshnan