ICT 409

13 Dec 2020

Data Representation

Md. Sharif Hossen, Assistant Professor, Dept. of ICT, Comilla University

Outline

- Floating Point Number Representation
- Shifting Operation
- Fixed Point Number Representation
- Data Representation
  - Basic Formats
  - Word lengths
  - Storage Order
    - Big-endian
    - Little-endian

Floating Point Number Representation

Consider a binary number $\mathbf{110101_2}$ which represents the value:

$1 * 2^5 + 1 * 2^4 + 0 * 2^3 + 1 * 2^2 + 0* 2^1 + 1 * 2^0$

$= 32 + 16 + 4 + 1$

$= \mathbf{53_{10}}$

Divide the number 53 by 2, the result would be **26.5**.

How do we represent it if we only had integer representations?

Floating Point Number Representation

Consider a binary number $\mathbf{110101_2}$ which represents the value:

$1 * 2^5 + 1 * 2^4 + 0 * 2^3 + 1 * 2^2 + 0 * 2^1 + 1 * 2^0$

$= 32 + 16 + 4 + 1$

$= \mathbf{53_{10}}$

Divide the number 53 by 2, the result would be **26.5**.

How do we represent it if we only had integer representations?

The **key to represent fractional numbers**, like 26.5 above, is the concept of *binary point*. A binary point is like the decimal point in a decimal system. It acts as a **divider** between the integer and the fractional part of a number.

Floating Point Number Representation

In a decimal system, a decimal point denotes the **position in a numeral** that the **coefficient should multiply** by $10^0 = 1$. For example, in the numeral 26.5, the coefficient 6 has a weight of $10^0 = 1$.

$$2 * 10^1 + 6 * 10^0 + 5 * 10^{-1} = 26.5$$

As in the decimal system, a binary point represents the coefficient of the term $2^0 = 1$. All digits (or bits) to the **left of the binary point** carries a weight of 2^0, 2^1, 2^2, and so on. Digits (or bits) on the **right of binary point** carries a weight of 2^{-1}, 2^{-2}, 2^{-3}, and so on. So $\mathbf{11010.1_2}$ represents the value

$1 * 2^4 + 1 * 2^3 + 0 * 2^2 + 1 * 2^1 + 0* 2^0 + 1 * 2^{-1}$

$= 16 + 8 + 2 + 0.5 = 26.5$

Shifting Operation

Used Instead of Binary Point

In the case of **53_{10}**, there is "no" binary point. Alternatively, we can say the binary point is located at the far right, *at position 0*. (Think in decimal, 53 and *53.0* represents the same number.)

2^5	2^4	2^3	2^2	2^1	2^0	Binary Point	2^{-1}	2^{-2}	2^{-3}
1	1	0	1	0	1	.	0	0	0

In the case of **26.5_{10}**, binary point is located one position to the *left* of 53_{10}. **Shifting an integer to the right by 1 bit position** is equivalent to dividing the number by 2.

2^5	2^4	2^3	2^2	2^1	2^0	Binary Point	2^{-1}	2^{-2}	2^{-3}
0	1	1	0	1	0	.	1	0	0

Fixed Point Number Representation

To represent a real number in computers (or any hardware in general), we can define a fixed point number type simply by **implicitly *fixing* the binary point to be at some position** of a numeral. W we need are two parameters:

- width (w) of the number representation, and
- binary point position (b) within the number

Hence, the notation is **fixed<w,b>**

For example, **fixed<8,3>** denotes a 8-bit fixed point number, of which 3 right most bits are fractional. Therefore, the bit pattern:

0	0	0	1	0	1	1	0

represents a real number:

$00010.110_2 = 1 * 2^1 + 1 * 2^{-1} + 1 * 2^{-2}$

$= 2 + 0.5 + 0.25 = 2.75$

Fixed Point Number Representation

0	0	0	1	0	1	1	0

A bit pattern can represent anything. Therefore the same bit pattern, if we "cast" it to another type, such as a **fixed<8,5>** type, will represents the number:

$000.10110_2 = 1 * 2^{-1} + 1 * 2^{-3} + 1 * 2^{-4}$

$= 0.5 + 0.125 + 0.0625 = 0.6875$

If we treat this bit patter as **integer**, it represents the number:

$00010110_2 = 1 * 2^4 + 1 * 2^2 + 1 * 2^1$

$= 16 + 4 + 2 = 22$

H.T. Write a program in C++ to represent fixed point number.

Data Representation

- Basic Formats

Figure 3.15 shows the fundamental division of **information into instructions** (operation or control words) and **data** (operands).

Data can be further subdivided into **numerical and nonnumerical**. **Two** main number **formats** have evolved: fixed-point and floating-point.

The **binary fixed-point format** takes the form $b_A b_B b_C \ldots b_K$, where each b_i is 0 or 1 and a binary point is present in some fixed but implicit position.

A floating-point number, on the other hand, consists of a pair of fixed-point numbers M, E, which denote the number $M \times B^E$, where B is a predetermined base.

Data Representation

Nonnumerical data usually take the form of **variable-length character strings** encoded in one of several standard codes, such as ASCII code.

Non-numerical data represents characteristics such as a person's gender, marital status, hometown, ethnicity or the types of movies people like. An example is non-numerical data representing the colors of flowers in a yard: yellow, blue, white, red, etc.

Data Representation

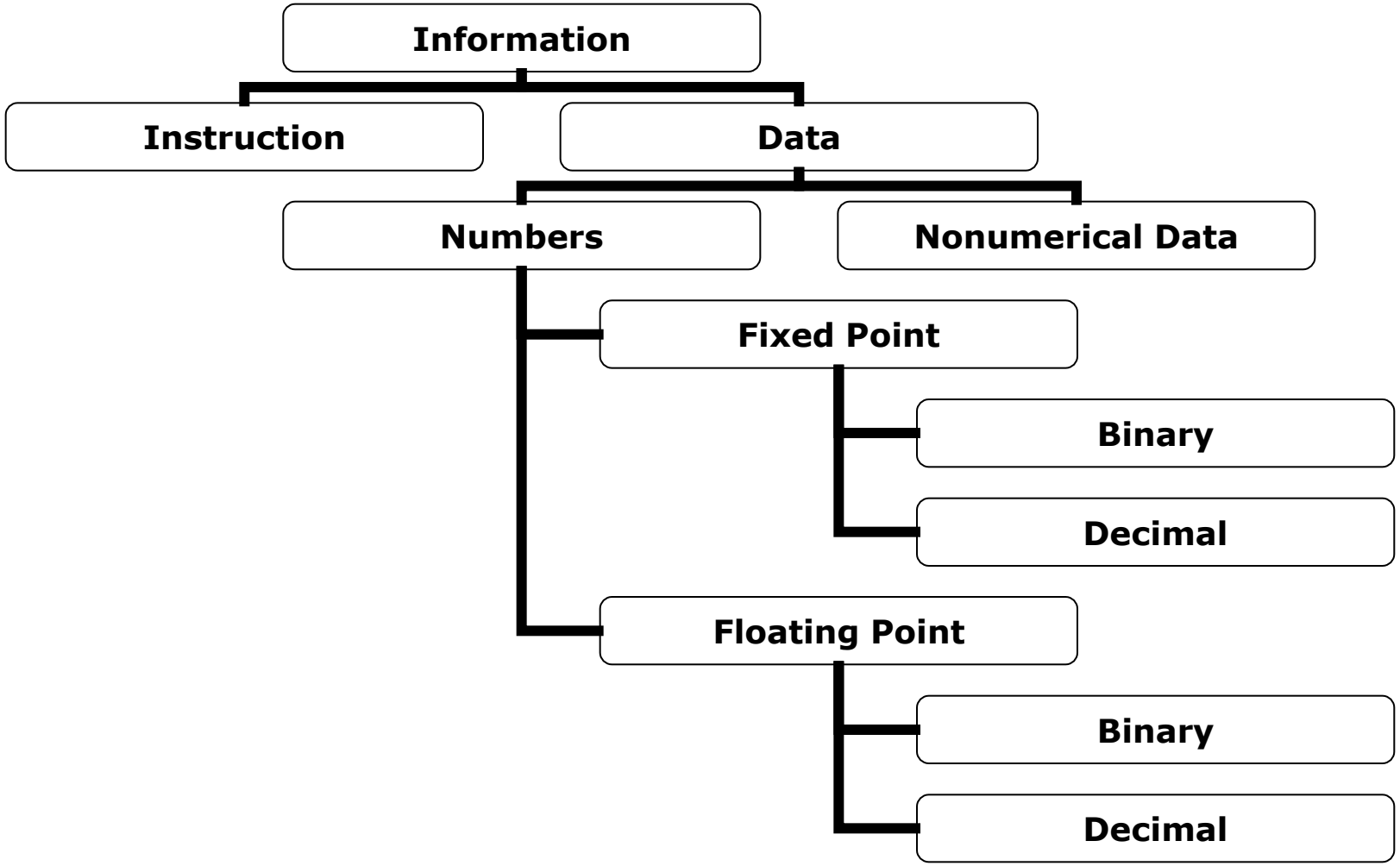

Figure 3.15 The basic information types

Data Representation

- Word Length

Information is represented in a digital computer by means of **binary words**, where a word is a **unit of information of some fixed length n**. An n-bit word allows up to **2^n different items to be represented**. For example, with n = 4, we can encode the 10 decimal digits as follows:

0 = 0000 1 = 0001 2 = 0010 3 = 0011 4 = 0100

5 = 0101 6 = 0110 7 = 0111 8 = 1000 9 = 1001

Number of bits AC can store at a time. Word size is typically a multiple of 8, common CPU word sizes being 8, 16, 32, and 64 bits.

Data Representation

- Storage Order

A small but important aspect of data representaion is the way in which the **bits of a word are indexed** as shown in Fig.1, where the **right-most bit is assigned the index 0** and the bits are **labeled in increasing order** from right to left.

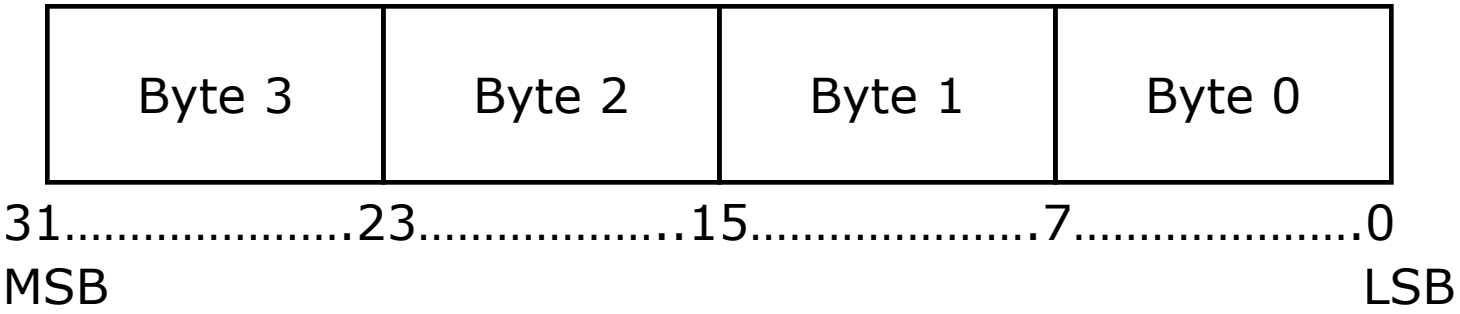

Fig 1. Indexing convention for the bits and bytes of a word.

The advantage of this convention is that when the word is interpreted as an unsigned binary integer, the **low order indexes** correspond to the numerically **less significant bits** and the high-order indexes more significant bits.

Data Representation

Suppose a sequence $W_0, W_1,\ldots,W_m$ of m 4-byte number words is to be stored. Suppose further that $\mathbf{W_i}$ as $\mathbf{B_{i,3}}$, $\mathbf{B_{i,2}}$, $\mathbf{B_{i,1}}$, $\mathbf{B_{i,0}}$. Hence the entire sequence can be written as

$$W_0,W_1,\ldots W_m = B_{0,3},B_{0,2},B_{0,1},B_{0,0},B_{1,3},B_{1,2},B_{1,1},B_{1,0},\ldots\ldots, B_{m,3},B_{m,2},B_{m,1},B_{m,0}$$

This storage sequence is called **big-endian** and shown in Fig 3.18 a. It is *so named* because the **most significant byte $\mathbf{B_{i,3}}$ of word $\mathbf{W_i}$ is assigned the lowest address** and the **least significant byte $\mathbf{B_{i,0}}$ is assigned the highest address**. In other words, the big-endian scheme assigns the highest address to byte 0.

Data Representation

The alternative byte-storage scheme called **little-endian assigns the lowest address to byte 0**. This corresponds to

$$W_0, W_1, \ldots W_m = B_{0,0}, B_{0,1}, B_{0,2}, B_{0,3}, B_{1,0}, B_{1,1}, B_{1,2}, B_{1,3}, \ldots\ldots, B_{m,0}, B_{m,1}, B_{m,2}, B_{m,3}$$

and is illustrated by Fig. 3.18 b.

For example, the Motorola 680X0 uses the big-endian method, whereas the Intel 80X86 series is little-endian.

Data Representation

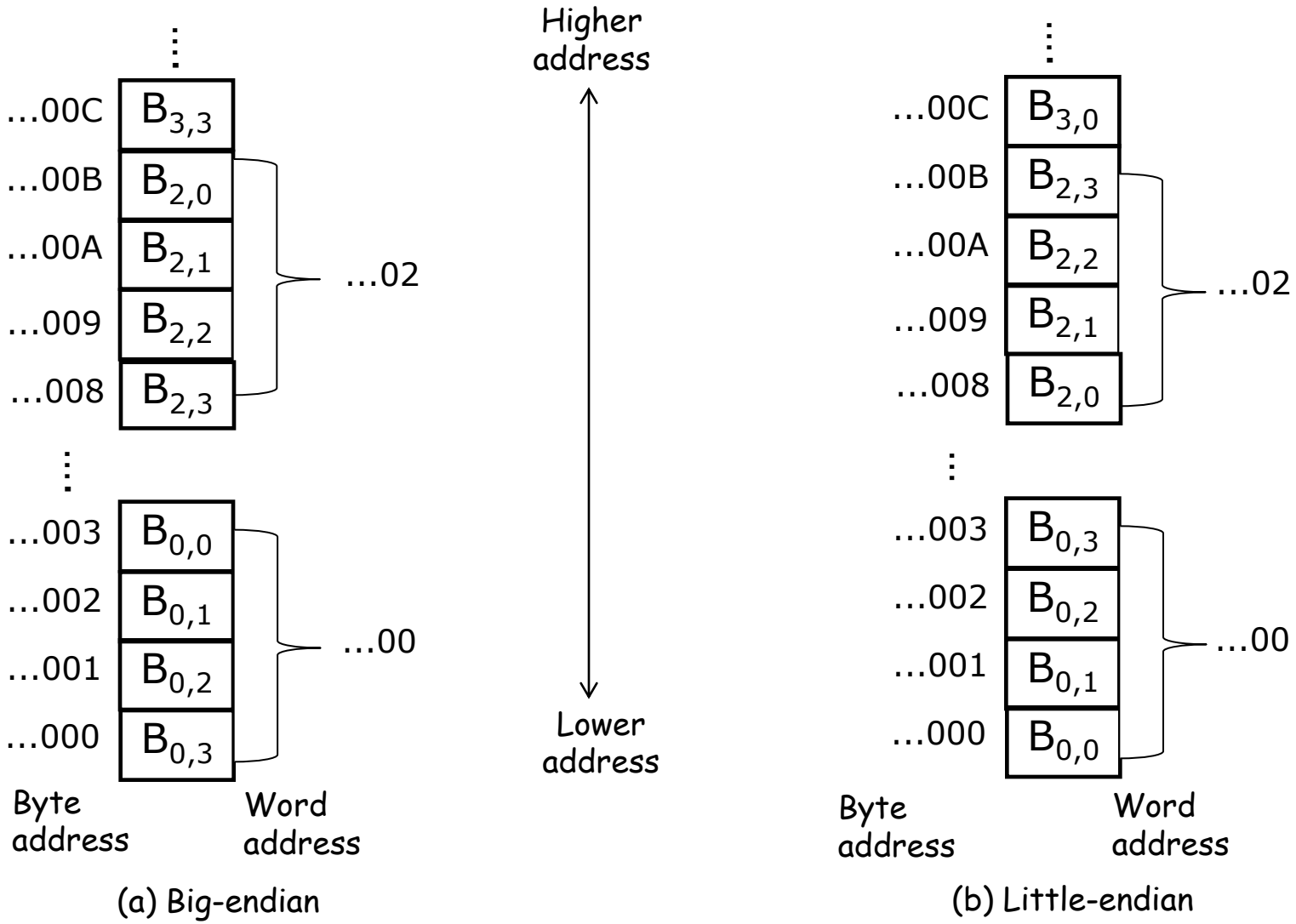

Fig. 3.18 Basic byte storage methods