Bits and Bytes, Integers, Floating Point

1、 bits and bytes

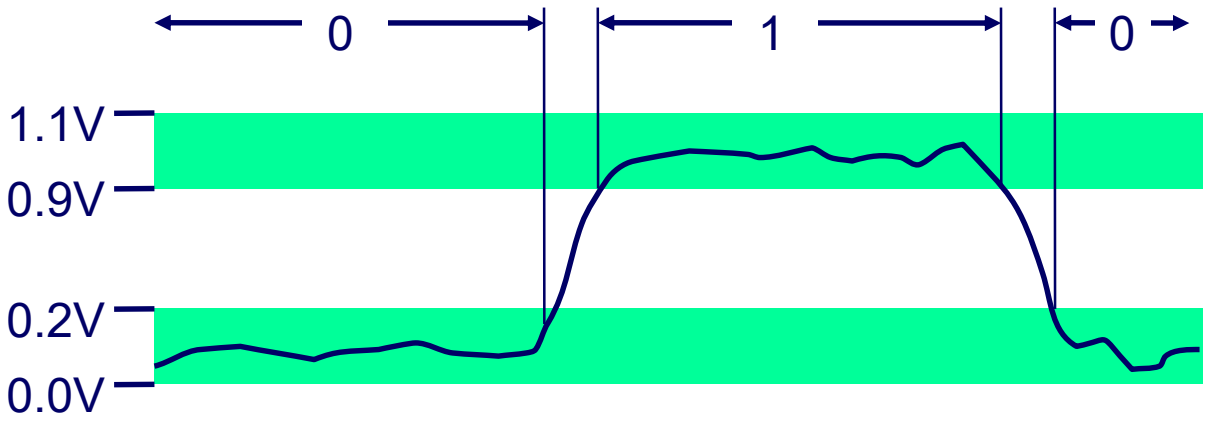
- **Representing information as bits**

①each bit is 0 or 1.

②By encoding / interpreting sets of bits in various ways

- Computer determine what to do.
- Manipulate number, string, sets……

③类似于工作原理？



在计算机读取 010101010 的时候会有电压的变化，通过电压的变化来发信号。

④Binary, Decimal, Hexadecimal（二进制，十进制，十六进制

- Byte ＝ 8 bits
- Binary 0000 0000 to 1111 1111
- Decimal 0 to 255
- Hexadecimal 0x00 to 0xff

Example: 15213: 0011 1011 0110 1101

3 B 6 D

⑤Data Representations

C Data Type	Typical 32-bit	Typical 64-bit	x86-64
char	1	1	1
short	2	2	2
int	4	4	4
long	4	8	8
float	4	4	4
double	8	8	8
pointer	4	8	8

（在 32 位操作系统中只有 double 数据类型是 8 个 byte，因为他必须是 Int 的两倍）

- **Bit-level manipulations**

①Algebraic representation of logic

- encode “true” as 1 and “false” as 0.
- A&B = 1 when both A=1 and B=1 (只要有 0 就是 0
- A|B = 1 when either A=1 or B=1 (有 1 就是 1
- ~A = 1 when A=0 (1 变成 0，0 变成 1
- A^B = 1 when either A=1 or B=1, but not both (都是 0 或者都是 1 的情况下为 0

（看 PPT 上的例子，一般十六进制的符号运算都是先转换成二进制然后计算

②Logic Operations：**&&, ||, !**

- View 0 as “false”.
- Anything nonzero as “true”.

- Always return 0 or 1.
- Early termination (

Example: !0x00 || !0 || !0000 --- > 1

!0x41 || !1 || !0001 --- > 0

③shift Operations (移位运算

- Left shift：x << y

  将 x 向左移 y 位，舍弃溢出位，在右边补 0

  （左移位不分逻辑和算数，只在右边补 0，舍弃溢出位

- Right shift：x >> y

  Logic Right shift (逻辑右移

  将 x 向右移 y 位，舍弃溢出位，在左边补 0

  （和左移相似

  Arithmetic Right shift (算数右移

  如果最左位是 1，则在左边补 1；

  如果最左边是 0，则在左边补 0；

- Integers

① unsigned and signed

- Unsigned

$$B2U(X) = \sum_{i=0}^{w-1} x_i \times 2^i$$

(无符号数二进制计算公式，十六进制同理

- Signed

$$B2T(X) = -x_{w-1} \times 2^{w-1} + \sum_{i=0}^{w-2} x_i \times 2^i$$

（有符号数

第一位是符号位，是 1 就是负数，0 就是正数

如果要取 x 的相反数，运算为：~x + 1

Unsigned if have “U” as suffix

0U, 4294967259U

example:

-16 8 4 2 1

-10 = 1 + 0 + 1 + 1 + 0 (凑数

16 位数值

	Decimal	Hex	Binary
UMax	65535	FF FF	11111111 11111111
TMax	32767	7F FF	01111111 11111111
TMin	-32768	80 00	10000000 00000000
-1	-1	FF FF	11111111 11111111
0	0	00 00	00000000 00000000

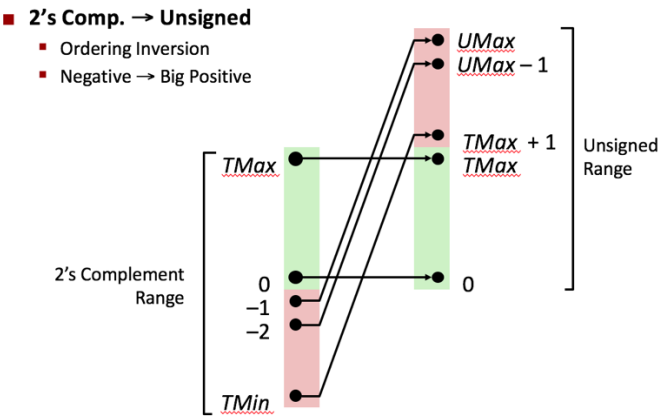
不同位的不同值

	W			
	8	16	32	64
UMax	255	65,535	4,294,967,295	18,446,744,073,709,551,615
TMax	127	32,767	2,147,483,647	9,223,372,036,854,775,807
TMin	-128	-32,768	-2,147,483,648	-9,223,372,036,854,775,808

$|TMin| = TMax + 1$ $UMax = 2 * TMax + 1$

②Conversion and casting

Bits	Signed		Unsigned
0000	0	=	0
0001	1		1
0010	2		2
0011	3		3
0100	4		4
0101	5		5
0110	6		6
0111	7		7
1000	-8		8
1001	-7		9
1010	-6		10
1011	-5	+/- 16	11
1100	-4		12
1101	-3		13
1110	-2		14
1111	-1		15



③Expanding and truncating

- Sign Extension

  将 w 位的二进制数变为 w + b 位的，并保持原值不变。

  方法：在前面扩展出 k 位的相同数字。（用上一节的 example 即可证）

  比如下面一段程序：

```
short int x =  15213;
int       ix = (int) x;
short int y = -15213;
int       iy = (int) y;
```

实际原理：（int 为 4 个 byte 4*8 = 32bits

	decimal	binary
x	15213	0011 1011 0110 1101
ix	15213	0000 0000 0000 0000 0011 1011 0110 1101
y	--15213	1100 0100 1001 0011
iy	--15213	1111 1111 1111 1111 1100 0100 1001 0011

- Truncation

  将 w + b 位的二进制有符号数或者无符号数变成 w 位的。

  方法：直接删去前面 b 位的数。（老实说感觉很不靠谱

④Addition，negation，multiplication，shifting

- Unsigned Addition

  Example：

  1110 + 1101 = 11011 -----→ 1011 (舍弃一位，先算二进制在转化成 16 和 10 进制

- Two‘s Complement Addition

  规则和上面差不多？舍多位

- Unsigned Multiplication

  也是正常算，然后舍弃前面多的位数

- Power-of-2 Multiply with Shift

  Example：**u << 3 == u * 8**

  (u << 5) – (u << 3) == u * 24

- x + ~x = -1

- 在 ppt 的倒数第二页有很好的习题就不贴出来了

2、 float

- IEEE Floating Point

  ①Floating point representation

  Example:

  $15213_{10} = (-1)^0 \times 1.1101101101101_2 \times 2^{13}$

  Function：$(-1)^s M \ 2^E$

  S 是确定这个数是正数还是负数。

  M 区间在 1.0-2.0，表示小数部分（m 的小数点后面部分是 frac，少的补 0）

  E 表示幂次（小数点移位 （exp － Bias

  如图为组成

  - 32bits 中 A ＝ 8 B ＝ 23 Bias ＝ 127
  - 64bits 中 A ＝ 11 B ＝ 52 Bias ＝ 1023

  e 的计算是 E ＝ Exp（Decimal） － Bias（Decimal）

  frac 为 m 小数点之后的部分，如果位数不够用 0 补齐。

  - Example：

  float F = 15213.0;

  - 当 exp 全为 0 时，e ＝ 1 － Bias，m 为 0.xxxx

- Rounding, addition, multiplication

  - Rounding

    Towards zero 向 0 取整

    Round down 向负无穷取整（向下取整

    Round up 向正无穷取整（向上取整

    Nearest Even 如果小数部分大于等于 0.5，向偶数部分取整

    Example：1.40 ——》 1.0，1.6———》2.0，2.50 ——》2.0

  - Float point multiplication

    Function：$(-1)^{s1}$ M1 2^{E1} x $(-1)^{s2}$ M2 2^{E2}

S: s1 x s2　　m: m1 x m2　　e: e1 + e2

Example: 4 bit mantissa:

$1.010*2^2$ x $1.110 * 2^3 = 10.0011 * 2^5 = 1.00011 * 2^6 = 1.001 * 2^6$

- Float point addition

Example: $1.010*2^2 + 1.110*2^3 = (0.1010 + 1.1100)*2^3$

$= 10.0110 * 2^3 = 1.00110 * 2^4 = 1.010 * 2^4$