信息的表示和处理(1)

谢旻晖

2025.9

Outline

- Bit和Byte
- 数字的机器表示

Why Bit?

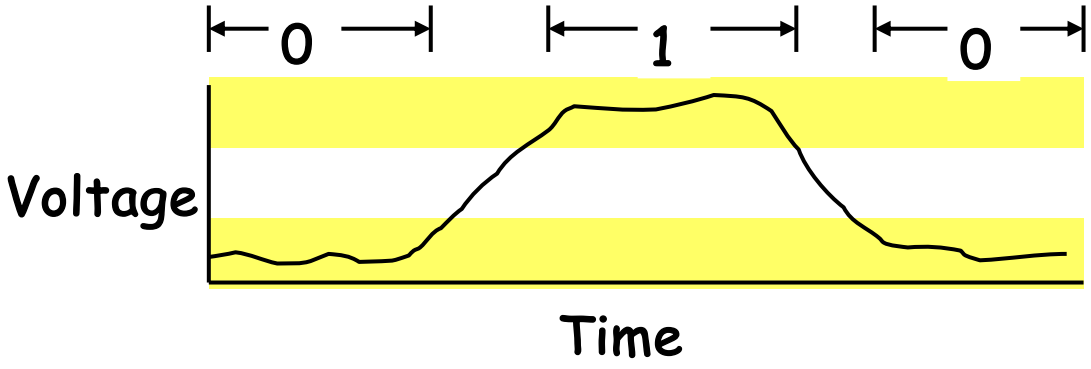
- Modern computers store and process
  - Information represented as two-valued signals
  - These lowly binary digits are ***bits***
- Bits form the basis of the digital revolution

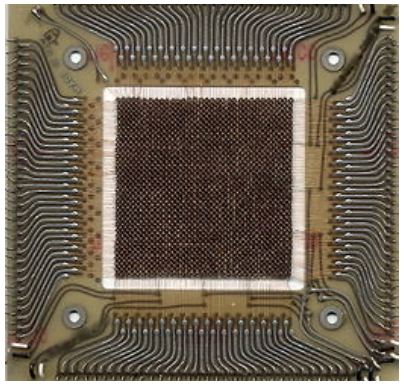
Why Bit?

- 人类更习惯10进制（Decimal）
  - Base-10，10根手指
  - 已经使用超过1000年
  - 印度->阿拉伯->东西方

- 计算机使用二进制表示信息
  - Bit: binary digit
  - binary values work better when building machines
  - store and process information

Why Bit?



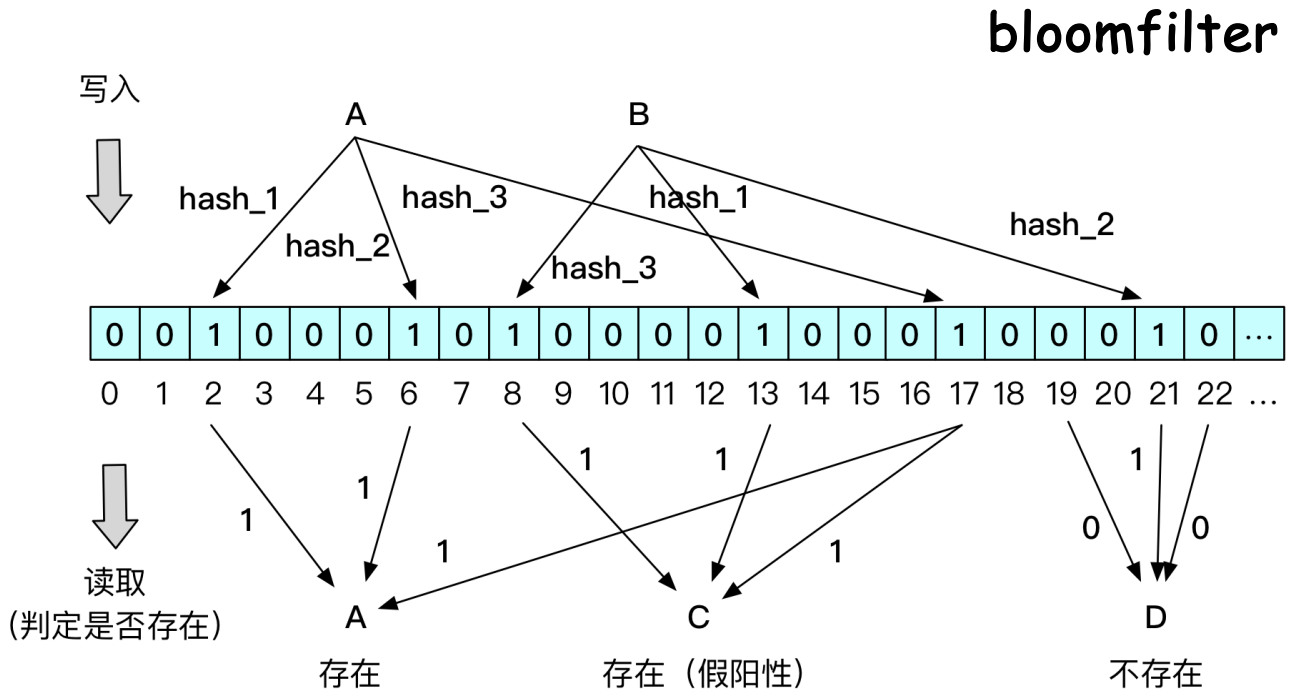


- 在计算机中表示信息方面，二进制比十进制更优秀
  - 对器件的要求低，
  - 简单、低成本、可靠
  - 提高电路密度
  - e.g., PCM, SLC->MLC

Group Bits

- 单个的bit没太大用（bitmap, bloomfilter）
  - 因为alphabet太小（只有0和1两个符号）
  - 英文的alphabet包括26(或52)个符号，单个符号的表达力更强
- 但英文还是有大量的词汇（符号组合）
- 因此，我们也可以用bits(而不是bit)来表示信息

bitmap和bloomfilter



Group Bits

- 具体做法：
  - 首先把bits分成组
  - 然后给可能的bit组合不同的解释，赋予其一定含义
- 8-bit组成a byte
  - Dr. Werner Buchholz in July 1956
  - 在IBM Stretch计算设计的早期阶段

维纳·布赫霍尔兹

Value of Bits

二进制串的值：

Bits　　01010

Value　　$0*2^4+1*2^3+0*2^2+1*2^1+0*2^0 = 10$

Group bits as numbers — Three encodings

- **Unsigned encoding**
  - 表示大于等于0的整数
  - 采用传统的二进制数字表达的规则
  - unsigned short, unsigned int, unsigned long
- **Two's-complement encoding**（二进制补码）
  - 可表示正、负整数（有符号数）
  - 是最常见的一种编码（主流计算机的默认整型编码）
  - short, int, long
- **Floating point encoding**
  - 近似表示实数，无法表示所有实数（很大/很小都不行）
  - 底为2的科学计数法方式
  - float, double

‘int’ is not integer

- Overflow（溢出）
  - 200*300*400*500 = -884,901,888
  - 乘积超过了整数的表示范围
- 满足*交换律*和*结合律*
  - (500 * 400) * (300 * 200)
  - ((500 * 400) * 300) * 200
  - ((200 * 500) * 300) * 400
  - 400 * (200 * (300 * 500))

'float' is not real number

- Overflow and Underflow
- 结合律可能不成立
  - (3.14+1e20)-1e20 = 0.0
  - 3.14+(1e20-1e20) = 3.14

Example Data 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
`long double`	–	–	10/16
pointer	4	8	8

进制转换

二进制转十进制：

二进制数$(N)_2$按权展开再相加，可计算得到该数的十进制表示。

$$(1101.0101)_2 = (1\cdot2^3 + 1\cdot2^2 + 0\cdot2^1 + 1\cdot2^0 + 0\cdot2^{-1} + 1\cdot2^{-2} + 0\cdot2^{-3} + 1\cdot2^{-4})_{10}$$

$$= (8+4+0+1+0+0.25+0+0.0625)_{10}$$

$$= (13.3125)_{10}$$

进制转换

十进制转二进制

```
Value          102(1100110)
Bits           102 = 51*2 + 0 (0)
               51  = 25*2 + 1 (1)
               25  = 12*2 + 1 (1)
               12  =  6*2 + 0 (0)
                6  =  3*2 + 0 (0)
                3  =  1*2 + 1 (1)
                1  =  0*2 + 1 (1)
```

进制转换

十进制数转换成二进制数

•将十进制数转换成二进制的方法为：整数部分和小数部分别转换，各自得出结果后再合并。

•对整数部分，采用**除2取余数法**。其规则如下：

- 将十进制数除以2，所得余数（0或1）即为对应二进制数最低位的值。然后对上次所得商除以2，所得余数即为二进制数次低位的值，如此进行下去，直到商等于0为止，最后得的余数是所求二进制数最高位的值。

•对小数部分，采用**乘2取整数法**。其规则如下：

- 将十进制数乘以2，所得乘积的整数部分即为对应二进制小数最高位的值，然后对所余数的小数部分部分乘以2，所得乘积的整数部分为次高位的值，如此进行下去，直到乘积的小数部分为0，或结果已满足所需精度要求为止。

进制转换

十进制数转换成二进制数

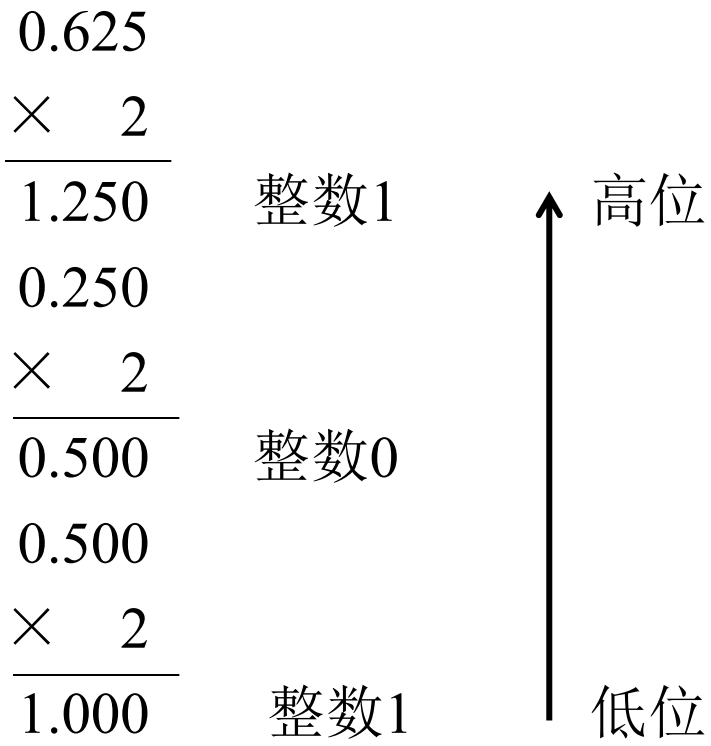
例如：将$(57.625)_{10}$转换成二进制。

•整数部分的转换：

所以得出：$(57)_{10} = (111001)_2$

进制转换

•小数部分的转换：



所以得出： $(0.625)_{10} = (0.101)_2$

总后得出： $(57.625)_{10} = (111001.101)_2$

Hexadecimal

- 16进制的Alphabet中包含16个符号： '0' to '9' and 'A' to 'F'
- Write $FA1D37B_{16}$ in C as
  - `0x`FA1D37B or
  - `0x`fa1d37b or
  - FA1D37B`H` or
  - fa1d37b`H`

Hexadecimal

- Byte = 8 bits
  - Binary 00000000_2 to 11111111_2
  - Decimal: 0_{10} to 255_{10}
  - Hexadecimal 00_{16} to FF_{16}

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

Hexadecimal vs. Binary

```
0x173A4C
Hexadecimal    1    7    3    A    4    C
Binary      0001 0111 0011 1010 0100 1100

111100101011010110011
Binary           11 1100 1010 1101 1011 0011
Hexadecimal       3    C    A    D    B    3
0x3CADB3
```

Hexadecimal vs. Decimal

```
Hexadecimal     0xA7
Decimal         10*16+7 = 167

Decimal         314156 = 19634*16 + 12 (C)
                 19634 =  1227*16 +  2 (2)
                  1227 =    76*16 + 11 (B)
                    76 =     4*16 + 12 (C)
                     4 =     0*16 +  4 (4)
Hexadecimal     0x4CB2C
```

Hexadecimal vs. Binary

- 0x39A7F8 ->
  0011 1001 1010 0111 11111 000
- 1100 1001 0111 1011 -> C97B
- 0xD5E4C -> 1101 0101 1110 0100 1100
- 10 0110 1110 0111 1011 0101 -> 2 6 E 7 B 5

Decimal, Hexadecimal, Binary

Decimal	Binary	Hexadecimal
167	1010 0111	0xA7
62	0011 1110	0x3E
188	1011 1100	0xBC
3*16+7=55	0011 0111	0x37
8*16+8=136	1000 1000	0x88
15*16+3=243	1111 0011	0xF3
5*16+2=82	0101 0010	0x52
10*16+12=172	1010 1100	0xAC
14*16+7=231	1110 0111	0xE7

Octal

八进制

•八进制数$(N)_8$按权展开再相加，可计算得到该数的十进制表示。

•例如:

$$(15.24)_8 = (1\bullet 8^1 + 5\bullet 8^0 + 2\bullet 8^{-1} + 4\bullet 8^{-2})_{10}$$

$$= (8+5+0.25+0.0625)_{10}$$

$$= (13.3125)_{10}$$

Octal

二进制数与八进制数之间的转换

•因为$2^3=8$，所以3位二进制数与1位八进制数有直接对应关系，即3位二进制数可以直接写为1位八进制数， 1位八进制数也可以直接写为3位二进制数。

•将二进制数转换为八进制数的方法是：对于一个兼有整数和小数部分的二进制数，以小数点为界，整数部分自右至左每3位分一组，最后不足3位时左边用0补足；小数部分自左至右每3位分一组，最后不足3位时右边用0补足。

•将八进制数转换为二进制数的方法：将八进制数的每一位用等值的3位二进制数代替。

例：$(1101.0101)_2=(001\ 101.\ 010\ 100)_2=(15.\ 24)_8$

例：$(47.3)_8=(100111.011)_2$

设置

进制转换（二进制小数位最多保留4位）

$(157)_{10}$ = ([填空1])$_2$

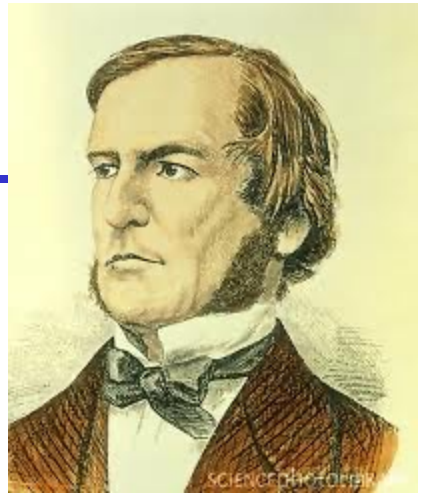
$(102.675)_{10}$ = ([填空2])$_2$

$(36.15)_8$ = ([填空3])$_{16}$

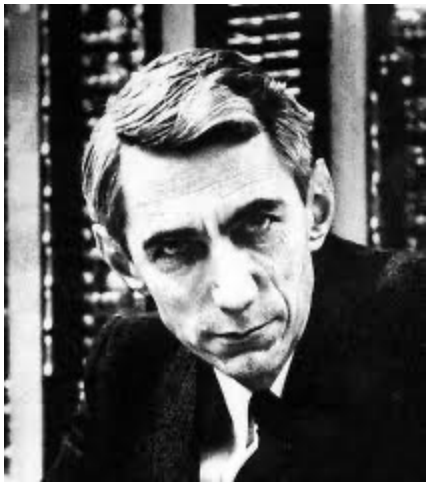
作答

Bit-level operations

Boolean Algebra



- George Boole(1815-1864)发明
  - 逻辑的代数表示：
    - Encode "True" as 1
    - Encode "False" as 0
- Claude Shannon(1916-2001)建立了信息论
  - 建立布尔代数和数字逻辑之间的关联
- 在数字电路设计和分析中起到最重要的作用



Boolean Algebra

And

A&B = 1 when both A=1 and B=1

&	0	1
0	0	0
1	0	1

Or

A|B = 1 when either A=1 or B=1

|	0	1
0	0	1
1	1	1

Not

~A = 1 when A=0

~	
0	1
1	0

Exclusive-Or (Xor)

A^B = 1 when either A=1 or B=1, but not both

^	0	1
0	0	1
1	1	0

General Boolean Algebras

- Operate on Bit Vectors
  - Operations applied bitwise

```
  01101001      01101001      01101001
& 01010101    | 01010101    ^ 01010101    ~ 01010101
  01000001      01111101      00111100      10101010
```

General Boolean Algebras

- 集合的形式表达
- 用宽度为w bit的向量表示子集 {0, …, *w*-1}
  - $a_j = 1$ if $j \in A$
    - 01101001 { 0, 3, 5, 6 }
    - 01010101 { 0, 2, 4, 6 }
  - & Intersection 01000001 { 0, 6 }
  - | Union 01111101 { 0, 2, 3, 4, 5, 6 }
  - ~ Complement 10101010 { 1, 3, 5, 7 }
  - ^ Symmetric difference 00111100 { 2, 3, 4, 5 }

Bit-Level Operations in C

- Operations &, |, ~, ^ Available in C
  - 可以应用于任何整数数据类型
    - long, int, short, char
  - 将参数视为bit vectors
  - 参数中每一位对应去做位运算（并行）

Bit-Level Operations in C

- Examples (Char data type)
  - ~0x41 → 0xBE
    - $\sim 01000001_2 \rightarrow 10111110_2$
  - ~0x00 → 0xFF
    - $\sim 00000000_2 \rightarrow 11111111_2$
  - 0x69 & 0x55 → 0x41
    - $01101001_2 \;\&\; 01010101_2 \rightarrow 01000001_2$
  - 0x69 | 0x55 → 0x7D
    - $01101001_2 \mid 01010101_2 \rightarrow 01111101_2$

Cool Stuff with Xor

- Bitwise Xor 可以用来做加法
  - 0+0 = 0
  - 1+1 = 0 (有额外进位计算逻辑)
  - 1+0=0+1=1
- 每个bit都是自己的additive inverse（加法逆元，相加本位得0）
  - A ^ A = 0

Cool Stuff with Xor

- Xor用于比较两个bit group是否相等
  - 每个bit都相等，则得到0
  - 有一个bit不等，则得到非0

Cool Stuff with Xor

```
int  inplace_swap(int *x, int *y)
{
   *x = *x ^ *y;    /* #1 */
   *y = *x ^ *y;    /* #2 */
   *x = *x ^ *y;    /* #3 */
}
```

Step	*x	*y
Begin	A	B
1	A^B	B
2	A^B	(A^B)^B = A^(B^B) = A^0 = A
3	(A^B)^A = (B^A)^A = B^(A^A) = B^0 = B	A
End	B	A

A^0=A
A^1=~A

Cool Stuff with Xor

```
1  void reverse_array(int a[], int cnt) {
2          int first, last;
3          for (first = 0, last = cnt-1;
4                  first <= last;
5                  first++,last--)
6              inplace_swap(&a[first], &a[last]);
7  }
```

Mask Operations

- Bit pattern
  - 0xFF
    - Having 1s for the least significant eight bits
    - Indicates the lower-order byte of a word
- Mask Operation
  - X = 0x89ABCDEF
  - X & 0xFF =?

Mask Operations

- Write C expressions that work for any word size $w \geq 8$
- For $x = 0x87654321$, with $w = 32$
- The least significant byte of x, with all other bits set to 0
  - [0x00000021].

Mask Operations

- Bit pattern
  - 0xFF
- Mask Operation
  - X = 0x89ABCDEF
  - X | 0xFF =?

Mask Operations

- Bit pattern
  - 0xFF
- Mask Operation
  - X = 0x89ABCDEF
  - X ^ 0xFF =?

Mask Operations

- 掩码操作功能总结
  - 与操作
    - 保留某些位，其他位设为0
  - 或操作
    - 某些位强制置1
  - 异或操作
    - 某些位取反 (mask的对应位为1时)

- DEC公司的VAX计算机：没有And和Or指令，只有bis和bic指令：Set result z to x and modify it
- z = bis (int x, int m) (bit set)
  - Set result z to 1 at each bit position where m is 1
- z = bic(int x, int m) (bit clear)
  - set result z to 0 at each bit position where m is 1
- Use bis and bic to implement
  - Or(int x, int y)
  - Xor(int x, int y)

作答

Logical Operations in C

- Logical Operators
  - &&, ||, !
    - 将0视为"False"
    - 任何非0值都被视为"True"
    - 总是return 0 or 1
    - 提前终止 (短路表达式)

Logical Operations in C

- Examples (char data type)
  - `!0x41  -->  0x00`
  - `!0x00  -->  0x01`
  - `!!0x41 -->  0x01`
  - `0x69 && 0x55  -->  0x01`
  - `0x69 || 0x55  -->  0x01`

Short Cut in Logical Operations

- a && 5/a
  - If a is zero, the evaluation of 5/a is stopped
  - avoid division by zero
- 练习：Using only bit-level and logical operations
  - Implement x == y
  - it returns 1 when x and y are equal, and 0 otherwise

Shift Operations in C

- Left Shift: x << y
  - 将bit-vector x左移y位
    - 左边多出来的bits丢掉
    - 右边补0

Argument x	01100010
<< 3	00010*000*

Argument x	10100010
<< 3	00010*000*

Shift Operations in C

- Right Shift: x >> y
  - 将bit-vector x向右移动y位
    - 丢掉右边额外的bits
  - **逻辑移位**（无符号数）
    - 左边补0
  - **算数移位**（有符号数）
    - 左边总是补入最左边的符号位
    - 在补码表示时很有用（移动后保持数字符号不变，正数->正数，负数->负数）
  - 未定义的行为
    - Shift amount < 0 or ≥ word size

Argument x	01100010
Log. >> 2	*00*011000
Arith. >> 2	*00*011000

Argument x	10100010
Log. >> 2	*00*101000
Arith. >> 2	*11*101000

Shift Operations in C

- What happens ?
  - int lval = 0xFEDCBA98 << 32;
  - int aval = 0xFEDCBA98 >> 36;
  - unsigned uval = 0xFEDCBA98u >> 40;
- It may be
  - lval 0xFEDCBA98 (0)
  - aval 0xFFEDCBA9 (4)
  - uval 0x00FEDCBA (8)
- Be careful about
  - 1<<2 + 3<<4 means 1<<(2 + 3)<<4

课堂练习

写出代码实现以下函数：

```
/* * Generate mask indicating leftmost 1 in x. Assume w=32
* For example, 0xFF00 -> 0x8000, and 0x6000 -> 0x4000.
* If x = 0, then return 0 */
int leftmost_one(unsigned x);
```

可以假设x是32位的int类型，代码最多包含15个算数运算、位运算或逻辑运算。

提示：现将x转为[0…01…1]的位向量

```c
/*
 * Generate mask indicating leftmost 1 in x. Assume w=32
 * For example, 0xFF00 -> 0x8000, and 0x6000 -> 0x4000.
 * If x = 0, then return 0
 */
int leftmost_one(unsigned x) {
  /*
   * first, generate a mask that all bits after leftmost one are one
   * e.g. 0xFF00 -> 0xFFFF, and 0x6000 -> 0x7FFF
   * If x = 0, get 0
   */
  x |= x >> 1;
  x |= x >> 2;
  x |= x >> 4;
  x |= x >> 8;
  x |= x >> 16;
  /*
   * then, do  mask & (~mask >> 1), reserve leftmost bit one
   * that's we want
   */
  return x & (~x >> 1);
}
```

课堂练习

- 写出代码实现如下函数：
  - /* Return 1 when x contains an even number of 1s; 0 otherwise.
  - Assume w=32 */
  - int even_ones (unsigned x);
  - 你的代码最多只能包括12个算术运算、位运算和逻辑运算。
  - C语言中的位运算：&, |, ~, ^ (与、或、非、异或)；移位运算<<，>>

课堂练习（答案）

```
int even_ones (unsigned x) {
        x=x^(x>>16);
        x=x^(x>>8);
        x=x^(x>>4);
        x=x^(x>>2);
        x=x^(x>>1);
        return !(x&1);
}
```

课堂练习：bitCount

- Returns number of 1's a in word
- Examples: bitCount(5) = 2, bitCount(7) = 3
- Legal ops: ! ~ & ^ | + << >>
- Max ops: 40

Sum 8 groups of 4 bits each

```
int bitCount(int x) {
    int m1 = 0x11 | (0x11 << 8);
    int mask = m1 | (m1 << 16);
    int s = x & mask;
    s += x>>1 & mask;
    s += x>>2 & mask;
    s += x>>3 & mask;
```

Combine the sums

```
  /* Now combine high and low order sums */
   s = s + (s >> 16);

   /* Low order 16 bits now consists of 4 sums.
      Split into two groups and sum */
   mask = 0xF | (0xF << 8);
   s = (s & mask) + ((s >> 4) & mask);
   return (s + (s>>8)) & 0x3F;
}
```

Homework1

- Homework1提交
  - 时间：9.30（可能微调）
  - 形式：pdf文件，上传obe.ruc.edu.cn

总结

- Bit, Group of Bits (Byte)
- 进制转换
  - 10 ←→ 2、8、16
- 位运算
  - 布尔代数
  - C语言的位运算操作
  - 利用并行，提升效率

Lab

- Lab1: DataLab
  - 用C语言的位运算、移位运算实现一些操作
  - 运算符数量有严格的限制
    - 充分利用bit的"并发性"
  - Rating: 1~4
  - 20道题目左右
  - 时间：2周