Ch2: Data Representation

数据的机器级表示

第一讲 数值数据的表示

第二讲 非数值数据表示及
数据的宽度、存储排列、纠/检错

第一讲：数值数据的表示

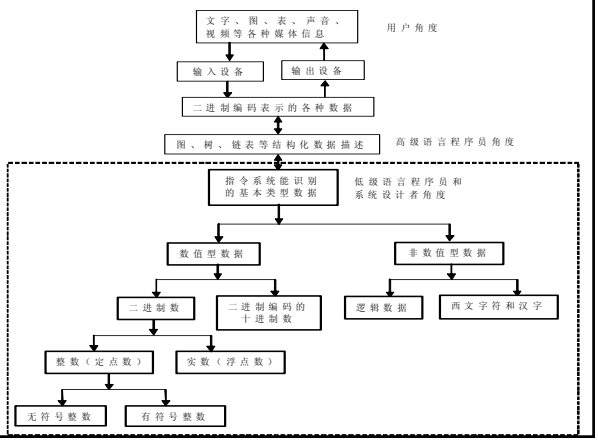
主 要 内 容

- 定点数的表示
  - 进位计数制
  - 定点数的二进制编码
    - 原码、补码、移码表示
  - 定点整数的表示
    - 无符号整数、带符号整数
- 浮点数格式和表示范围
- 浮点数的规格化
- IEEE754浮点数标准
  - 单精度浮点数、双精度浮点数
  - 特殊数的表示形式
- C语言程序中的整数类型、浮点数类型
- 十进制数表示

信息的二进制编码

- 计算机的外部信息与内部机器级数据
  - 数值数据：无符号整数、带符号整数、浮点数（实数）、十进制数
  - 非数值数据：逻辑数（包括位串）、西文字符和汉字
- 计算机内部所有信息都用二进制（即：0和1）进行编码
- 用二进制编码的原因：
  - 制造二个稳定态的物理器件容易
  - 二进制编码、计数、运算规则简单
  - 正好与逻辑命题对应，便于逻辑运算，并可方便地用逻辑电路实现算术运算
- 真值和机器数
  - 机器数：用0和1编码的计算机内部的0/1序列
  - 真值：机器数真正的值，即：现实中带正负号的数

首先考虑数值数据的表示



数值数据的表示

- 数值数据表示的三要素
  - 进位计数制
  - 定、浮点表示
  - 如何用二进制编码

  即：要确定一个数值数据的值必须先确定这三个要素。

  例如，机器数 01011001的值是多少？ 答案是：不知道！
- 进位计数制
  - 十进制、二进制、十六进制、八进制数及其相互转换
- 定/浮点表示（解决小数点问题）
  - 定点整数、定点小数
  - 浮点数（可用一个定点小数和一个定点整数来表示）
- 定点数的编码（解决正负号问题）
  - 原码、补码、反码、移码 （反码很少用）

Sign and Magnitude （原码的表示）

Decimal	Binary	Decimal	Binary
0	0000	-0	1000
1	0001	-1	1001
2	0010	-2	1010
3	0011	-3	1011
4	0100	-4	1100
5	0101	-5	1101
6	0110	-6	1110
7	0111	-7	1111

- 容易理解，但是：
  - 0 的表示不唯一，不利于程序员编程
  - 加、减运算方式不统一
  - 需额外对符号位进行处理，不利于硬件设计
  - 特别当 a<b时，实现 a- b比较困难

从 50年代开始，整数都采用补码来表示
但浮点数的尾数用原码定点小数表示

Excess (biased) notion- 移码表示

° 什么是“excess (biased) notation-移码表示”?

将每一个数值加上一个偏置常数（ Excess / bias）

° 一般来说，当编码位数为 n时，bias取 2^{n-1}

Ex. n=4: $E_{biased} = E + 2^3$ （bias= $2^3 = 1000_2$）

-8 (+8) ~ 0000_2

-7 (+8) ~ 0001_2 0的移码表示惟一

…

0 (+8) ~ 1000_2 移码和补码仅第一位不同

…

+7 (+8) ~ 1111_2 移码主要用来表示浮点数的阶码！

° 为什么要用移码来表示指数（阶码）?

便于浮点数加减运算时的对阶操作

例：$1.01 \times 2^{-1} + 1.11 \times 2^3$ 简化比较 $1.01 \times 2^{-1+4} + 1.11 \times 2^{3+4}$

补码：1111< 0011 ? 移码：0011< 0111

(-1) (3) (3) (7)

Back to last

回顾：补码特性 - 模运算（modular运算）

重要概念：在一个模运算系统中，一个数与它除以“模”后的余数等价。

时钟是一种模-12系统

假定钟表时针指向10点，要将它拨向 6 点， 则有两种拨法：

① 倒拨4格：10-4=6

② 顺拨8格：10+8=18≡6 (mod 12)

模12系统中： 10-4 ≡ 10+8 (mod 12)

-4≡ 8 (mod 12)

则，称8是- 4对模12的补码。

同样有 -3 ≡ 9 （mod 12）

-5 ≡ 7 （mod 12）等

结论1： 一个负数的补码等于模减该负数的绝对值。

结论2： 对于某一确定的模，某数减去小于模的另一数，总可以用该数加上另一数的补码来代替。

补码（modular运算）：+ and – 的统一

模运算系统举例

例1：“钟表”模运算系统

假定时针只能顺拨，从10点倒拨4格后是几点？

10- 4=10+(12- 4)=10+8=6 （mod 12）

例2：“4位十进制数” 模运算系统

假定算盘只有四档，且只能做加法，则在算盘上计算

9828-1928等于多少？

9828-1928=9828+(10^4-1928)

=9828+8072

= 1 7900

=7900（mod 10^4） 取模的含义就是只留余数，高位的“1”被丢弃！相当于只有低4位留在算盘上。

运算器是一个模运算系统，适合用补码表示和运算

计算机中的运算器只有有限位，假定为n位，则运算结果只能保留低n位，可看成是个只有n档的二进制运算算盘。所以，其模为2^n。

补码的定义 假定补码有n位，则：

① 定点整数：$[X]_{补} = 2^n + X$ （$-2^n \leqslant X < 2^n$，mod 2^n）

② 定点小数：$[X]_{补} = 2 + X$ （$-1 \leqslant X < 1$，mod 2）

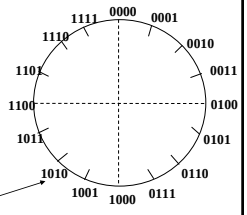
特殊数的补码（假定有n位）

① $[-2^{n-1}]_{补} = 2^n - 2^{n-1} = 10\ldots0$（n-1个0） （mod 2^n）

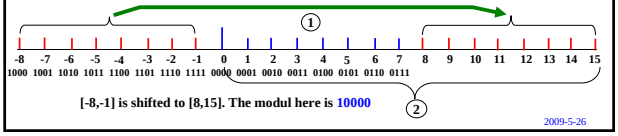
② $[-1]_{补} = 2^n - 0\ldots01 = 11\ldots1$（n个1） （mod 2^n）

③ $[-1.0]_{补} = 2 - 1.0 = 1.00\ldots0$（n-1个0） （mod 2）

④ $[+0]_{补} = [-0]_{补} = 00\ldots0$（n个0）



当n=4时，共有16个机器数：0000 ~ 1111，可看成是模为2^4 的钟表系统。真值范围为：-8 ~ +7



[-8,-1] is shifted to [8,15]. The modul here is 10000

Two’s Complement （补码的表示）

- 正数：符号位（sign bit）为0，数值部分不变
- 负数：符号位为1，数值部分“各位取反，末位加1”

变形（模4）补码：双符号，用于存放可溢出的中间结果。

Decimal	补码	变形补码	Decimal	Bitwise Inverse	补码	变形补码
0	0000	00000	-0	1111	0000	00000
1	0001	00001	-1	1110	1111	11111
2	0010	00010	-2	1101	1110	11110
3	0011	00011	-3	1100	1101	11101
4	0100	00100	-4	1011	1100	11100
5	0101	00101	-5	1010	1011	11011
6	0110	00110	-6	1001	1010	11010
7	0111	00111	-7	1000	1001	11001
8	1000	01000	-8	0111	1000	11000

+0和-0表示唯一

值太大，用4位补码无法表示，故“溢出”！但用变形补码可保留符号位和最高数值位。

如何求补码的值

根据补码各位上的“权”，可以求出一个补码的值

8-bit 2's complement example:

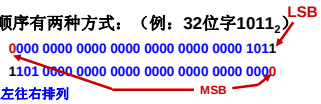
$11010110 = -2^7 + 2^6 + 2^4 + 2^2 + 2^1 = -128 + 64 + 16 + 4 + 2 = -42$

当N=4时，范围为：$-2^3 \sim 2^3 - 1$ （即：-8 ~ +7）

当N=32时，范围为：$-2^{31} \sim 2^{31} - 1$

Unsigned integer(无符号整数)

- 机器中字的位排列顺序有两种方式：（例：32位字1011_2）
  - 高到低位从左到右：0000 0000 0000 0000 0000 0000 0000 1011
  - 高到低位从右到左：1101 0000 0000 0000 0000 0000 0000 0000
  - MIPS采用高到低从左往右排列
  - Leftmost和rightmost这两个词有歧义，故用LSB(Least Significant Bit)来表示最低有效位，用MSB来表示最高有效位



- 若一个字为n位，则可表示的不同模式的字有2^n个
  - N=4时，16种模式为0000 ~ 1111
- 一般在全部是正数运算且不出现负值结果的场合下，可使用无符号数表示。例如地址运算
- 无符号数的各位编码中没有符号位
- 在字长相同的情况下，它的表示范围大于有符号数
- 无符号数总是整数，所以很多时候就简称为“无符号数”
- 最大8位无符号整数是11111111B，其值为255

Signed integer（带符号整数）

- 计算机必须能处理正数(positive) 和负数(negative)，MSB表示数符
- 有三种表示方式
  - Signed magnitude （原码）
    用来表示浮点（实）数的尾数
  - One’s complement （反码）
    现已不用
  - Two’s complement （补码）
    50年代以来，所有计算机都用补码来表示定点（整）数
- 为什么用补码表示带符号整数？
  - 补码运算系统是模运算系统，加、减运算统一
  - 数0的表示惟一，方便使用
  - 比原码和反码多表示一个最小负数
  - 与移码相比，其符号位和真值的符号对应关系清楚

带符号数和无符号数的比较

- 扩充操作有差别
  - 例如，MIPS提供了两种加载指令
    - 无符号数：lbu $t0, 0($s0) ；$t0的高24位补0 （称为0扩展）
    - 带符号数： lb $t0, 0($s0) ；$t0的高24位补符号 （称为符号扩展）
- 数的比较有差异
  - 无符号数：MSB为1的数比MSB为0的数大
  - 带符号数： MSB为1的数比MSB为0的数小
  - 例如，MIPS中提供了不同的比较指令，如：
    - 无符号数：sltu $t0, $s0, $s1 （set less than unsigned）
    - 带符号数： slt $t1, $s0, $s1 （set less than）

  假定： $s0=1111 1111 1111 1111 1111 1111 1111 1111
  $s1=0000 0000 0000 0000 0000 0000 0000 0001
  则：$t0和$t1分别为多少？
  答案：$t0和$t1分别为0和1。
- 溢出判断有差异（无符号数根据最高位是否有进位来判断溢出）
  - MIPS规定：无符号数运算溢出时，不产生“溢出异常”

C语言程序中的整数

无符号数：unsigned int (short / long)；带符号数： int (short / long)
常在一个数的后面加一个“u”或“U”表示无符号数
若运算中同时有无符号数和有符号整数，则C编译器隐含将有符号数强制转换为无符号数

假定以下关系表达式在32位用补码表示的机器上执行，结果是什么？

关系表达式	运算类型	结果	说明
0 = = 0U			
-1 < 0			
-1 < 0U			
2147483647 > -2147483647 - 1			
2147483647U > -2147483647 - 1			
2147483647 > (int) 2147483648U			
-1 > -2			
(unsigned) -1 > -2			

C语言程序中的整数

无符号数：unsigned int (short / long)；带符号数： int (short / long)
常在一个数的后面加一个“u”或“U”表示无符号数
若运算中同时有无符号数和有符号整数，则C编译器隐含将有符号数强制转换为无符号数

假定以下关系表达式在32位用补码表示的机器上执行，结果是什么？

关系表达式	运算类型	结果	说明
0 = = 0U	无符号整数	1	00…0B = 00…0B
-1 < 0	有符号整数	1	11…1B (-1) < 00…0B (0)
-1 < 0U	无符号整数	0*	11…1B (2^{32}-1) > 00…0B(0)
2147483647 > -2147483647 - 1	有符号整数	1	011…1B (2^{31}-1) > 100…0B (-2^{31})
2147483647U > -2147483647 - 1	无符号整数	0*	011…1B (2^{31}-1) < 100…0B(2^{31})
2147483647 > (int) 2147483648U	有符号整数	1*	011…1B (2^{31}-1) > 100…0B (-2^{31})
-1 > -2	有符号整数	1	11…1B (-1) > 11…10B (-2)
(unsigned) -1 > -2	无符号整数	1	11…1B (2^{32}-1) > 11…10B (2^{32}-2)

科学计数法(Scientific Notation)与浮点数

Example:

mantissa (尾数) — *exponent(阶码、指数)*

6.02×10^{21}

decimal point — *radix (base，基)*

- Normalized form（规格化形式）：小数点前只有一位非0数
- 同一个数有多种表示形式。例：对于数 1/1,000,000,000
  - Normalized (唯一的规格化形式): 1.0×10^{-9}
  - Unnormalized（非规格化形式不唯一）：0.1×10^{-8}, 10.0×10^{-10}

for Binary Numbers:

mantissa（尾数） — *exponent（指数）*

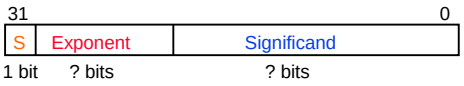
$1.011_{two} \times 2^{-10}$

binary point — *基为2*

只要对尾数和指数分别编码，就可表示一个浮点数（即：实数）

浮点数的表示

- Normal format（规格化数形式）：
  +/-1.xxxxxxxxx$_{two}$ x $2^{Exponent}$
- 32-bit 规格化数：



S 是符号位（Sign）

Exponent用 excess (or biased) notation(移码/增码)来表示

Significand 表示 xxxxxxxxxxxx，尾数部分

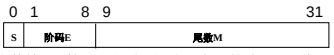
(基可以是 2/ 4 / 8 / 16，约定信息，无需显式表示)

- 早期的计算机，各自定义自己的浮点数格式

问题：浮点数表示不统一会带来什么问题？

浮点数(Floating Point)的表示范围

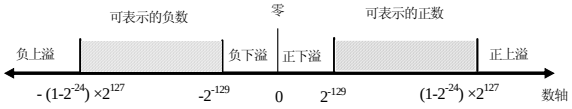
例：画出下述32位浮点数格式的表数范围。



第0位为数符S；第1～8位为8位移码表示的阶码E（偏置常数为128）；第9～31位为24位二进制原码小数表示的尾数M。规格化尾数的第一位总是1，故规定第一位默认的"1"不明显表示出来。这样可用23个数位表示24位尾数。

最大正数： $0.11\ldots1 \times 2^{11\ldots1} = (1-2^{-24}) \times 2^{127}$ **最小正数：** $0.10\ldots0 \times 2^{00\ldots0} = (1/2) \times 2^{-128}$

因为原码和移码都是对称的，所以其表示范围是关于原点对称的。



机器0：阶码为0 或 落在下溢区中的数

浮点数范围比定点数大，但数的个数没有变多，故数之间更稀疏，且不均匀

"Father" of the IEEE 754 standard

直到80年代初，各个机器内部的浮点数表示格式还没有统一
因而相互不兼容，机器之间传送数据时，带来麻烦

1970年代后期, IEEE成立委员会着手制定浮点数标准

1985年完成浮点数标准IEEE754的制定

现在所有计算机都采用IEEE754来表示浮点数

This standard was primarily the work of one person, UC Berkeley math professor William Kahan.

www.cs.berkeley.edu/~wkahan/ieee754status/754story.html

Prof. William Kahan

IEEE 754 Floating Point Standard

Single Precision： (Double Precision is similar)

- **Sign bit: 1 表示negative ; 0表示 positive**
- **Exponent（阶码 / 指数）：** **全0和全1编码要用来表示特殊的值！**
  - **SP规格化数阶码范围为0000 0001(-126) ~ 1111 1110(127)**
  - **bias为127 (single), 1023(double)**
- **Significand（尾数）：**
  - **规格化尾数最高位总是1，所以隐含表示，省1位**
  - **1 + 23 bits （ single），1 + 52 bits （double）**

为什么用127？若用128，则阶码范围为多少？

0000 0001(-127) ~ 1111 1110(126)

SP: $(-1)^S \times (1 + Significand) \times 2^{(Exponent-127)}$

DP: $(-1)^S \times (1 + Significand) \times 2^{(Exponent-1023)}$

Ex: Converting Binary FP to Decimal

BEE00000H is the hex. Rep. Of an IEEE 754 SP FP number

1011 1110 1110 0000 0000 0000 0000 0000

$(-1)^S \times (1 + Significand) \times 2^{(Exponent-127)}$

- **Sign: 1 => negative**
- **Exponent:**
  - **$0111\ 1101_{two} = 125_{ten}$**
  - **Bias adjustment: 125 - 127 = -2**
- **Significand:**

  $1 + 1\times2^{-1} + 1\times2^{-2} + 0\times2^{-3} + 0\times2^{-4} + 0\times2^{-5} + \ldots$

  $=1+2^{-1}+2^{-2} = 1+0.5+0.25 = 1.75$
- **Represents: $-1.75_{ten} \times 2^{-2} = -0.4375$ $(= -4.375\times10^{-1})$**

Ex: Converting Decimal to FP

-1.275×10^1

1. **Denormalize: -12. 75**
2. **Convert integer part:**
   $12 = 8 + 4 = 1100_2$
3. **Convert fractional part:**
   $.75 = .5 + .25 = .11_2$
4. **Put parts together and normalize:**
   $1100.11 = 1.10011 \times 2^3$
5. **Convert exponent: $127 + 3 = 128 + 2 = 10000010_2$**

1100 0001 0100 1100 0000 0000 0000 0000

The Hex rep. is C14C0000H

Normalized numbers（规格化数）

前面的定义都是针对规格化数（normalized form）

How about other patterns?

Exponent	Significand	Object
1-254	anything implicit leading 1	Norms
0	0	?
0	nonzero	?
255	0	?
255	nonzero	?

Representation for 0

How to represent 0?

exponent: all zeros

significand: all zeros

What about sign? Both cases valid.

+0: 0 00000000 00000000000000000000000

-0: 1 00000000 00000000000000000000000

Representation for +∞/-∞

In FP, 除数为0的结果是 +/- infinity, 不是overflow.

为什么要这样处理?

- **可以利用+∞/-∞作比较。 例如：X/0 > Y 可作为一个有效比较**

How to represent +∞/-∞?

- **Exponent : all ones (11111111B=255)**
- **Significand: all zeros**

+∞ : 0 11111111 00000000000000000000000

-∞ : 1 11111111 00000000000000000000000

Operations

5 / 0 =+∞, -5 / 0 =-∞

5+(+∞)=+∞, (+∞)+(+∞)=+∞

5 - (+∞)=-∞, (-∞) - (+∞)= -∞ etc

Representation for "Not a Number"

Sqrt (- 4.0) = ? 0/0 = ?

- **Called Not a Number (NaN) - "非数"**

How to represent NaN

Exponent = 255

Significand: nonzero

NaNs can help with debugging

Operations

sqrt(-4.0)=NaN 0/0=NaN

op(NaN,x) = NaN +∞+(-∞)=NaN

+∞-(+∞)=NaN ∞/∞=NaN

etc.

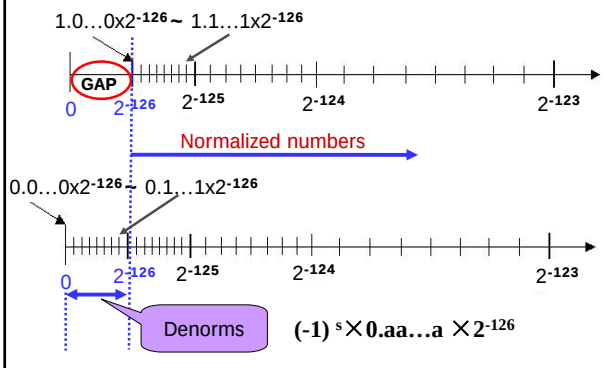
Representation for Denorms(非规格化数)

What have we defined so far? (Single Precision)

Exponent	Significand	Object
0	0	+/-0
0	nonzero	Denorms
1-254	anything implicit leading 1	Norms
255	0	+/- infinity
255	nonzero	NaN

Used to represent Denormalized numbers

Representation for Denorms



Questions about IEEE 754

- What's the range of representable values?
  The largest number for single: +1.11…1x2^{127} 约 +3.4 x 10^{38}
  How about double? 约 +1.8 x 10^{308}
- What about following type converting: not always true!

```
if ( i == (int) ((float) i) ) {      How about double?   True!
    printf ("true");
}
if ( f == (float) ((int) f) ) {      How about double?   Not always true!
    printf ("true");
}
```

- How about FP add associative? FALSE!
  x = – 1.5 x 10^{38}, y = 1.5 x 10^{38}, z = 1.0
  (x+y)+z = (–1.5x10^{38}+1.5x10^{38}) +1.0 = 1.0
  x+(y+z) = –1.5x10^{38}+ (1.5x10^{38}+1.0) = 0.0

十进制数的表示

- 数值数据（numerical data）的两种表示
  Binary (二进制数)
  - 定点整数：Fixed-point number (integer)
  - 浮点数：Floating-point number (real number)

  Decimal (十进制数)
  - 用ASCII码表示
  - 用BCD（Binary coded Decimal）码表示
- 计算机中为什么要用十进制数表示数值？
  - 日常使用的都是十进制数，所以，计算机外部都使用十进制数。在一些有大量数据输入/出的系统中，为减少二进制数和十进制数之间的转换，在计算机内部直接用十进制数表示数值。

用ASCII码表示十进制数

- 前分隔数字串
  - 符号位单独用一个字节表示，位于数字串之前。
  - 正号用"+"的ASCII码(2BH)表示；负号用"-"的ASCII码(2DH)表示
  - 例：十进制数+236表示为: 2B 32 33 36H
    0010 1011 0011 0010 0011 0011 0011 0110B
    十进制数-2369表示为: 2D 32 33 36 39H
    0010 1101 0011 0010 0011 0011 0011 0110 0011 1001B
- 后嵌入数字串
  - 符号位嵌入到最低一位数字的ASCII码的高4位中。比前分隔方式省一个字节。
  - 正数不变；负数高4位变为0111.
  - 例：十进制数+236表示为: 32 33 36H
    0011 0010 0011 0011 0011 0110B
    十进制数-2369表示为: 32 33 36 79H
    0011 0010 0011 0011 0011 0110 0111 1001B

缺点：占空间大，且需转换成二进制数或BCD码才能计算。

用BCD码表示十进制数

- 编码思想
  每个十进制数位必须至少有4位二进制位来表示。而4位二进制位可以组合成16种状态，去掉10种状态后还有6种冗余状态。
- 编码方案
  1. 十进制有权码
     - 指表示每个十进制数位的4个二进制数位（称为基2码）都有一个确定的权。8421码是最常用的十进制有权码。也称自然BCD（NBCD）码。
  2. 十进制无权码
     - 指表示每个十进制数位的4个基2码没有确定的权。在无权码方案中，用的较多的是余3码和格雷码。
  3. 其他编码方案 （5中取2码、独热码等）
- 符号位的表示：
  - "+"：1100； "-"：1101
  - 例：+236=$(1100\ 0010\ 0011\ 0110)_{8421}$ （占2个字节）
    - 2369=$(1101\ 0000\ 0010\ 0011\ 0110\ 1001)_{8421}$ （占3个字节）
      补0以使数占满一个字节

第一讲小结

- 计算机内所有信息都用二进制编码，在机器内部编码后的数称为机器数，其值称为真值
- 定义数值数据有三个要素：进制、定点/浮点、编码
- 定点数编码：原码、补码、移码
- 定点数：
  - 定点整数：表示整数或浮点数中的指数（阶码）；定点小数：表示浮点数中的尾数
- 整数的表示
  - 无符号数：正整数，用来表示地址等；带符号整数：用补码表示
- C语言中的整数
  - 无符号数：unsigned int (short / long)；带符号数： int (short / long)
- 浮点数的表示
  - 符号；尾数：定点小数；指数（阶）：定点整数（基不用表示）
- 浮点数的范围
  - 不可表示区域：正上溢、正下溢、负上溢、负下溢；与阶码的位数和基的大小有关
- 浮点数的精度
  - 与尾数的位数和是否规格化有关；规格化操作：左规、右规
- 浮点数的表示（IEEE754标准）：单精度SP（float）和双精度DP（double）
  - 规格化数(SP)：阶码1~254，尾数最高位隐含为1
  - "零" (阶为全0，尾为全0)
  - ∞ (阶为全1，尾为全0)
  - NaN (阶为全1，尾为非0)
  - 非规格化数 (阶为全0，尾为非0)
- 十进制数的表示：用ASCII码、BCD码表示

第二讲 非数值数据、数据排列、纠/检错

主 要 内 容

- 非数值数据的表示
  - 逻辑数据
  - 西文字符
  - 汉字
- 数据的宽度
- 数据的存储排列
  - 大端方式
  - 小端方式
- 数据的纠错和检错
  - 奇偶校验
  - 海明校验
  - 循环冗余校验

逻辑数据的编码表示

- 表示
  - 用一位表示 真：1 / 假：0
  - N位二进制数可表示N个逻辑数据，或一个位串
- 运算
  - 按位进行
  - 如:按位与 / 按位或 / 逻辑左移 / 逻辑右移 等
- 识别
  - 逻辑数据和数值数据在形式上并无差别，也是一串0/1序列，机器靠指令来识别。

西文字符的编码表示

- 特点
  - 是一种拼音文字，用有限几个字母可以拼写出所有单词
  - 只要对有限个少量字母和一些数学符号、标点符号等辅助字符进行编码
  - 所有西文字符集的字符总数不超过256个，所以使用7或8个二进位可表示
- 表示（常用编码为7位ASCII码，必须熟悉数字、字母和空格(SP)的表示）
  - 十进制数字：0/1/2…/9
  - 英文字母：A/B/…/Z/a/b/…/z
  - 专用符号：+/-/%/*/&/……
  - 控制字符（不可打印或显示）
- 操作
  - 字符串操作，如:传送/比较 等

汉字及国际字符的编码表示

- 特点
  - 汉字是表意文字，一个字就是一个方块图形。
  - 汉字数量巨大，总数超过6万字，给汉字在计算机内部的表示、汉字的传输与交换、汉字的输入和输出等带来了一系列问题。
- 编码形式
  - 有以下几种汉字代码：

    输入码：对每个汉字用相应的按键进行编码表示，用于输入。

    内码：用于在系统中进行存储、查找、传送等处理。

    字模点阵码或轮廓描述：描述汉字的字模点阵或轮廓，用于显示或打印。

问题：西文字符有没有输入码？有没有内码？有没有字模点阵码或轮廓描述？

汉字的输入码

- 向计算机输入汉字的方式：

  ① 手写汉字联机识别输入，或者是印刷汉字扫描输入后自动识别，这两种方法现均已达到实用水平。

  ② 用语音输入汉字，虽然简单易操作，但离实用阶段还相差很远。

  ③ 利用英文键盘输入汉字：

  每个汉字用一个或几个键表示，这种对每个汉字用相应的按键进行的编码表示称为汉字的“输入码”，又称外码。输入码的码元为键盘上的按键。这是最简便、最广泛采用的汉字输入方法。

  常用的方法有：五笔字型、智能ABC、微软拼音等
- 使用汉字输入码的原因：

  ① 键盘面向西文设计，一个或两个西文字符对应一个按键，非常方便。

  ② 汉字是大字符集，专门的汉字输入键盘由于键多、查找不便、成本高等原因而几乎无法采用。

字符集与汉字的内码

问题：西文字符常用的内码是什么？

其内码就是ASCII码。

对于汉字内码的选择，必须考虑以下几个因素：

① 不能有二义性，即不能和ASCII码有相同的编码。

② 尽量与汉字在字库中的位置有关系，便于汉字查找和处理。

③ 编码应尽量短。

国标码（国际交换码）

1981年我国颁布了《信息交换用汉字编码字符集·基本集》(GB2312—80)。该标准选出6763个常用汉字，为每个汉字规定了标准代码，以供汉字信息在不同计算机系统之间交换使用。

可在汉字国标码的基础上产生汉字机内码

GB2312-80字符集

- 由三部分组成：

  ① 字母、数字和各种符号，包括英文、俄文、日文平假名与片假名、罗马字母、汉语拼音等共687个；

  ② 一级常用汉字，共3755个，按汉语拼音排列；

  ③ 二级常用汉字，共3008个，因不太常用，所以按偏旁部首排列。
- 汉字的区位码
  - 码表由94行、94列组成，行号称为区号，列号称为位号，各占7位
  - 区位码指出了该汉字在码表中的位置。区位码共14位，区号在左、位号在右
- 汉字的国标码
  - 每个汉字的区号和位号各自加上32（20H），得到它的“国标码”
  - 国标码中区号和位号各占7位。在计算机内部，为方便处理与存储，前面添一个0，构成一个字节。

汉字内码

- 至少需2个字节才能表示一个汉字内码，为什么？
- 可在GB2312国标码的基础上产生汉字内码
  - 将国标码的两个字节的第一位置“1”后得到内码
    例如，汉字“大”的国标码为：3473h（0011 0100 0111 0011B），前面的34h和字符“4”的ACSII码相同，后面的73h和字符“s”的ACSII码相同，将每个字节的最高位各设为“1”后，就得到其内码：B4F3h (1011 0100 1111 0011B)。
- 国际字符集的必要性
  - 不同地区使用不同的字符集内码，如中文GB2312/Big5、日文Shift-JIS/EUC-JP等。一台安装了中文系统的计算机，若打开一个日文文件，便会出现乱码。为使所有国际字符都能互换，必须创建一种涵盖全部字符的多字符集。通过对已有各种地区性字符集规定使用范围来唯一定义各字符的编码。
- 国际多字符集，例如：
  - 国际标准ISO/IEC 10646提出了一种包括全世界现代书面语言文字所使用的所有字符的标准编码，每个字符用4个字节编码(UCS-4)和2字节编码(UCS-2)。
  - 我国（包括香港、台湾地区）与日本、韩国联合制订了一个统一的汉字字符集（CJK编码），共收集了上述不同国家和地区的共约2万多汉字及符号，采用2字节编码(即：UCS-2)，现已被批准为国家标准(GB13000)。
  - 微软公司新板的Windows操作系统(中文版)中已采用中西文统一编码，收集了中、日、韩三国常用的约2万汉字，称为“Unicode”，采用2字节编码，与UCS-2一致。

汉字的字模点阵码和轮廓描述

- 为便于汉字的打印、显示等，每个汉字的字形都必须预先存在机内
- 一套汉字所有字符的形状描述信息集合在一起称为字形信息库，简称字库(font)
- 不同字体(如宋体、仿宋、楷体、黑体等)对应不同字库
- 输出汉字时，先到字库中找到字形描述信息，然后送相应设备输出
- 字形主要有两种描述方法：
  - 字模点阵描述（图像方式）
  - 轮廓描述（图形方式）
    - 直线向量轮廓
    - 曲线轮廓（True Type字形）

数据的基本宽度

- 比特（bit）是计算机中处理、存储、传输信息的最小单位
- 在计算机内部，二进制信息的计量单位是“字节”(Byte)，也称“位组”
  - 现代计算机中，主存按字节编址
  - 字节是最小可寻址单位*(addressable* unit *)*
- 除了比特和字节之外，还经常使用“字”(word)作为单位
- “字”和 “字长”的概念不同
  - “字长”指数据通路的宽度。

    （数据通路指CPU内部的数据流经的路径以及路径上的部件，主要是CPU内部进行数据运算、存储和传送的部件，这些部件的宽度基本上要一致，才能相互匹配。因此，“字长”应该等于CPU内部总线的宽度、运算器的位数、通用寄存器的宽度等。）
  - “字”用来表示被处理信息的单位，用来度量各种数据类型的宽度。
  - 字和字长的宽度可以一样，也可不同。

    例如，Intel微处理器从386开始就是32位字长，但其定义的“字”的宽度为16位

数据量的度量单位

- 存储二进制信息时的度量单位要比字节或字大得多
- 经常使用的单位有：
  - “千字节”(KB)，1KB=2^{10}字节=1024B
  - “兆字节”(MB)，1MB=2^{20}字节=1024KB
  - “千兆字节”(GB)，1GB=2^{30}字节=1024MB
  - “兆兆字节”(TB)，1TB=2^{40}字节=1024GB
- 在描述计算机通信中的带宽时，也会遇到上述信息单位，但其值的大小与上述给出的值不同，这种情况下的值为：
  - “千字节”(KB)，1KB=10^3字节=1000B
  - “兆字节”(MB)，1MB=10^6字节=1000KB
  - “千兆字节”(GB)，1GB=10^9字节=1000MB
  - “兆兆字节”(TB)，1TB=10^{12}字节=1000GB

程序中数据类型的宽度

- 高级语言支持多种类型、多种长度的数据
  - 例如，C语言中Char类型的宽度为1个字节，可表示一个字符（非数值数据），也可表示一个8位的整数（数值数据）
  - 不同机器上表示的同一种类型的数据可能宽度不同
- 必须能够提供相应的机器级数据表示和相应的处理指令
  (在第五章指令系统介绍具体指令)

C语言中数值数据类型的宽度 (单位：字节)

C声明	典型32位机器	Compaq Alpha机器
char	1	1
short int	2	2
int	4	4
long int	4	8
char*	4	8
float	4	4
double	8	8

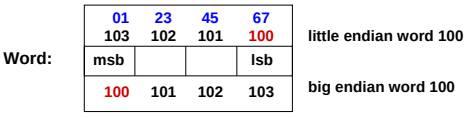
从表中看出：同类型数据并不是所有机器都采用相同的宽度，分配的字节数随机器的字长和编译器的不同而不同。

Compaq Alpha是一个针对高端应用的64位机器，即：字长为64位。

数据的存储和排列顺序

- BYTE Addressing: (字节编址)
  - 80年代开始，几乎所有机器都用字节编址
- ISAs设计时要考虑的两个问题：
  - 如何从一个字节地址中取到一个32位的字？- 字的存放问题
  - 一个字能否存放在任何字节边界？- 字的边界对齐问题

例如，若 int i = 0x01234567，存放在内存100号单元，则用“取数”指令访问100号单元取出 i 时，必须清楚i的4个字节是如何存放的。



大端方式（Big Endian）：MSB所在的地址是数的地址
e.g. IBM 360/370, Motorola 68k, MIPS, Sparc, HP PA
小端方式（Little Endian）：LSB所在的地址是数的地址
e.g. Intel 80x86, DEC VAX

BIG Endian versus Little Endian

Example 1: Memory layout of a number ABCDH located in 1000

In Big Endian: ⟶
CD 1001
AB 1000

In Little Endian: ⟶
AB 1001
CD 1000

Example 2: Memory layout of a number 00ABCDEFH located in 1000

In Big Endian: ⟶
00 1000
AB 1001
CD 1002
EF 1003

In Little Endian: ⟶
00 1003
AB 1002
CD 1001
EF 1000

Example 3: Memory layout of a instruction located in 1000

假定小端机器x86中指令：mov AX, 0x12345(BX)
其中操作码mov为40H，寄存器AX和BX分别为0001B和0010B，立即数占32位，则存放顺序为：

40	1	2	45 23 01 00

若在大端机器上，则存放顺序如何？

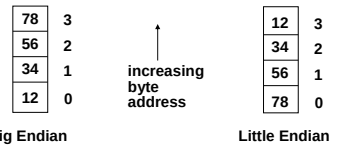
40	1	2	00 01 23 45

MOV	AX	BX	0x12345

00 1005
01 1004
23 1003
45 1002
12 1001
40 1000

2009-5-26

Byte Swap Problem（字节交换问题）



上述存放在0号单元的数据（字）是什么？ 12345678H？ 78563412H？

两个存放方式不同的机器间程序移植或数据通信时，会发生什么问题？

- 每个系统内部都是一致的，但在系统间通信时可能会发生问题！
- 因为顺序不同，需要顺序转换
- 任何像音频、视频和图像等文件格式或处理程序都涉及到字节顺序问题

ex. Little endian: GIF, PC Paintbrush, Microsoft RTF,etc
Big endian: Adobe Photoshop, JPEG, MacPaint, etc

2009-5-26

Alignment(对齐)

Alignment: 要求数据的地址是相应边界地址

- 目前计算机所用数据字长一般为32位或64位，而存储器地址按字节编址
- 指令系统支持对字节、半字、字及双字的运算，也有位处理指令
- 各种不同长度的数据存放时，有两种处理方式:
  - 按边界对齐 （假定字的宽度为32位，按字节编址）
    - 字地址：4的倍数(低两位为0)
    - 半字地址：2的倍数(低位为0)
    - 字节地址：任意
  - 不按边界对齐

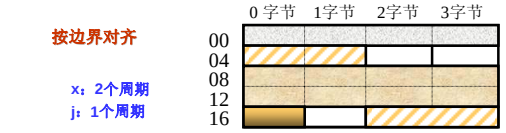
    坏处：可能会增加访存次数！

2009-5-26

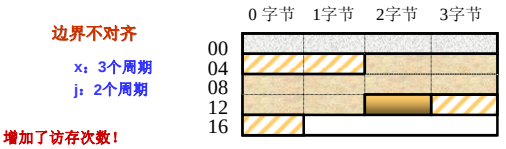
Alignment(对齐)

示例 假设数据顺序：字-半字-双字-字节-半字-……

如：int i, short k, double x, char c, short j,……



则：&i=0; &k=4; &x=8; &c=16; &j=18;……



增加了访存次数！

则： &i=0; &k=4; &x=6; &c=14; &j=15;……

2009-5-26

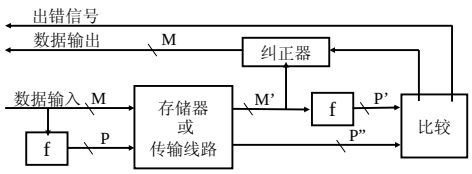
数据的检/纠错（Error Detect/Correct）

- 为什么要进行数据的错误检测与校正？

  计算、存取和传送时，由于元器件故障或噪音干扰等原因会出现差错。措施：

  (1) 从计算机硬件本身的可靠性入手，在电路、电源、布线等各方面采取必要的措施，提高计算机的抗干扰能力；

  (2) 采取相应的数据检错和校正措施，自动地发现并纠正错误。
- 如何进行错误检测与校正？
  - 大多采用“冗余校验”思想，即除原数据信息外，还增加若干位编码，这些新增的代码被称为校验位。



2009-5-26

数据的检/纠错

数据检/校过程：

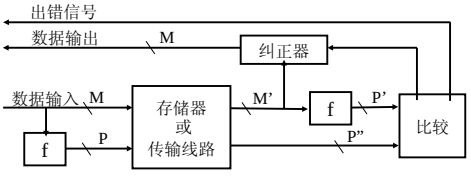
数据被存入存储器或从源部件传输时，对数据M进行某种运算（用函数f 表示），以产生相应的代码P= f (M)，这里P就是校验位。这样原数据信息和相应的校验位一起被存储或传送。当数据被读出或传送到终部件时，和数据信息一起被存储或传送的校验位也被得到，用于检错和纠错。假定读出后的数据为M'，通过同样的运算f 对M'也得到一个新的校验位P'=f (M')，假定原来被存储的校验位P取出后其值为P''，将校验位P''与新生成的校验位P'进行某种比较，根据其比较结果确定是否发生了差错。

比较的结果为以下三种情况之一：

① 没有检测到错误，得到的数据位直接传送出去。

② 检测到差错，并可以纠错。数据位和比较结果一起送入纠错器，将正确数据位传送出去。

③ 检测到错误，但无法确认哪位出错，因而不能进行纠错处理，此时，报告出错情况。



BACK

2009-5-26

码字和码距

- 什么叫码距？
  - 由若干位代码组成的一个字叫“码字”
  - 两个码字中具有不同代码的位的个数叫做这两个码字间的“距离”
  - 一种码制各码字间的最小距离称为“码距”，它就是这个码制的距离。

  问题：“8421”码的码距是几？

  2（0010）和3（0011）间距离为1，“8421”码制的码距为1。
- 数据校验中的“码字”是指数据位和校验位按某种规律排列得到的代码
- 码距与检错、纠错能力的关系

  ① 如果码距d为奇数，则能发现d-1位错，或者能纠正(d-1)/2位错。

  ② 如果码距d为偶数，则能发现d /2位错，并能纠正(d/2-1)位错。
- 常用的数据校验码有：

  奇偶校验码、海明校验码和循环冗余校验码。

奇偶校验码

基本思想：增加一位奇校验位（或偶校验位），然后将原数据和得到的校验位一起进行存储或传送，对存取后或在传送的终部件得到的相应数据和校验位，再进行一次编码，求出新校验位，最后根据得到的这个新校验位的值，确定是否发生了错误。

实现原理：假设将数据B=$b_{n-1}b_{n-2}...b_1b_0$从源部件传送至终部件。在终部件接收到的数据为B'=$b_{n-1}'b_{n-2}'...b_1'b_0'$。

第一步：在源部件求出奇（偶）校验位P。

若采用奇校验，则P=$b_{n-1}\oplus b_{n-2}\oplus...\oplus b_1\oplus b_0\oplus 1$。

若采用偶校验，则P=$b_{n-1}\oplus b_{n-2}\oplus...\oplus b_1\oplus b_0$。

第二步：在终部件求出奇（偶）校验位P'。

若采用奇校验，则P'= $b_{n-1}'\oplus b_{n-2}'\oplus...\oplus b_1'\oplus b_0'\oplus 1$。

若采用偶校验，则P'=$b_{n-1}'\oplus b_{n-2}'\oplus...\oplus b_1'\oplus b_0'$。

第三步：计算最终的校验位P*，并根据其值判断有无奇偶错。

假定P在终部件接受到的值为P''，则P*= P'⊕P''

① 若P*=1，则表示终部件接受的数据有奇数位错。

② 若P*=0，则表示终部件接受的数据正确或有偶数个错。

奇偶校验法的特点

- 特点：
  - 问题：奇偶校验码的码距是几？为什么？
  - 码距d=2。在奇偶校验码中，若两个数中有奇数位不同，则它们相应的校验位就不同；若有偶数位不同，则虽校验位相同，但至少有两位数据位不同。因而任意两个码字之间至少有两位不同。
  - 根据码距和纠/检错能力的关系，它只能发现奇数位出错，不能发现偶数位出错，而且也不能确定发生错误的位置，不具有纠错能力。
- 优点：
  - 开销小
  - 适用于校验一字节长的代码，故常被用于存储器读写检查或按字节传输过程中的数据校验

    因为一字节长的代码发生错误时，1位出错的概率较大，两位以上出错则很少，所以可用奇偶校验。

海明校验码

- 由Richard Hamming于1950年提出，目前还被广泛使用。
- 主要用于存储器中数据存取校验。
- 基本思想：奇偶校验码对整个数据编码生成一位校验位。因此这种校验码检错能力差，并且没有纠错能力。如果将整个数据按某种规律分成若干组，对每组进行相应的奇偶检测，就能提供多位检错信息，从而对错误位置进行定位，并将其纠正。
  - 海明校验码实质上就是一种多重奇偶校验码。
- 处理过程：
  - 最终比较时，按位进行异或操作，根据异或结果，确定是否有差错。
  - 这种异或操作所得到的结果称为故障字（syndrome word）。显然，校验码和故障字的位数是相同。

校验码位数的确定

- 假定数据位数为n，校验码为k位，则故障字的位数也为k位。k位故障字所能表示的状态最多是2^K，每种状态可用来说明一种出错情况。若只有一位错，则结果可能是：
  - 数据中某一位错 (n种可能)
  - 校验码中有一位错 (k种可能)
  - 无错 (1 种可能)

  } 1+n+k种情况

要能对最多一位错的所有结果进行正确表示，则n和k必须满足下列关系：

$$2^K \geqslant 1+n+k, \quad 即：2^K-1 \geqslant n+k$$

- 有效数据位数和校验码位数间的关系
- 从表中可以看出，当数据有8位时，校验码和故障字都应有4位。

说明：4位的故障字最多可以表示16种状态，而单个位出错情况最多只有12种可能（8个数据位和4个校验位），再加上无错的情况，一共有13种。所以，用16种状态表示13种情况应是足够了。

有效数据位数和校验码位数间的关系

	单纠错		单纠错/双检错	
数据位	检查位	增加百分率	检查位	增加百分率
8	4	50	5	62.5
16	5	31.25	6	37.5
32	6	18.75	7	21.875
64	7	10.94	8	12.5
128	8	6.25	9	7.03
256	9	3.52	10	3.91

海明码的分组

- 基本思想： 数据位和校验位按某种方式排列为一个n+k的码字，将该字中每一位的出错位置与故障字的数值建立关系，通过故障字的值确定该码字中哪一位发生了错误，并将其取反来纠正。

根据上述基本思想，按以下规则来解释各故障字的值。

规则1： 若故障字每位全都是0，则表示没有发生错误。

规则2：若故障字中有且仅有一位为1，则表示校验位中有一位出错，因而不需纠正。

规则3：若故障字中多位为1，则表示有一个数据位出错，其在码字中的出错位置由故障字的数值来确定。纠正时只要将出错位取反即可。

海明码的分组

以8位数据进行单个位检错和纠错为例说明。

假定一个8位数据M= $M_8M_7M_6M_5M_4M_3M_2M_1$，其相应的4位校验位为P=$P_4P_3P_2P_1$。根据规则将数据M和校验位P按一定的规律排到一个12位码字中。

据规则1，故障字为0000时，表示无错，因此没有和位置号0000对应的出错情况，所以位置号从0001开始。

据规则2，故障字中有且仅有一位为1时，表示校验位中有一位出错，此时，故障字只可能是0001、0010、0100、1000四种情况，我们将这四种状态分别代表校验位中第P_1、P_2、P_3、P_4位发生错误的情况，因此，校验位P_1、P_2、P_3、P_4应分别位于码字的第1、2、4、8位。

据规则3，将其他多位为1的故障字依次表示数据位M_1~M_8发生错误的情况。因此，数据位M_1~M_8应分别位于码字的第0011(3)、0101(5)、0110(6)、0111(7)、1001(9)、1010(10)、1011(11)、1100(12)位。即码字的排列为：

$M_8M_7M_6M_5P_4M_4M_3M_2P_3M_1P_2P_1$

这样，得到故障字S=$S_4S_3S_2S_1$的各个状态和出错情况的对应关系表，可根据这种对应关系对整个数据进行分组。

海明校验码分组情况

根据故障字$S_4S_3S_2S_1$的值确定哪位出错，因此，某位出错一定会影响与之相对应的故障字中为1的位所在组的奇偶性。

码字：$M_8M_7M_6M_5P_4M_4M_3M_2P_3M_1P_2P_1$

例：若M1出错，则故障字为0011,所以一定会改变S_2和S_1所在的分组。故M1同时被分到第一组和第二组。

问题：若P_1出错，则如何？若M_8出错，则如何？

P_1~0001，分在第一组，
M_8~1100，分在第四组和第三组

序号 / 含义 / 分组	1	2	3	4	5	6	7	8	9	10	11	12	故障字	正确	出错位
	P_1	P_2	M	P_3	M_2	M_3	M_4	P_4	M_5	M_6	M_7	M_8			1 2 3 4 5 6 7 8 9 10 11 12
第4组								√	√	√	√		S_4	0	0 0 0 0 0 0 0 0 1 1 1 1
第3组				√	√	√	√					√	S_3	0	0 0 0 1 1 1 1 0 0 0 0 1
第2组		√	√			√	√			√	√		S_2	0	0 1 1 0 0 1 1 0 0 1 1 0
第1组	√		√		√		√		√		√		S_1	0	1 0 1 0 1 0 1 0 1 0 1 0

BACK

校验位的生成和检错、纠错

- 分组完成后，就可对每组采用相应的奇（偶）校验，以得到相应的一个校验位。
- 假定采用偶校验 (即取校验位Pi，使对应组中有偶数个1)，则得到校验位与数据位之间存在如下关系：

$P_1 = M_1 \oplus M_2 \oplus M_4 \oplus M_5 \oplus M_7$

$P_2 = M_1 \oplus M_3 \oplus M_4 \oplus M_6 \oplus M_7$

$P_3 = M_2 \oplus M_3 \oplus M_4 \oplus M_8$

$P_4 = M_5 \oplus M_6 \oplus M_7 \oplus M_8$

- 海明校验过程：
  - 根据上面公式，可求出每一组对应的校验位Pi (i=1,2,3,4)
  - 数据M和校验位P一起被存储，根据读出数据M'，得到新的校验位P'
  - 读出校验位P''与新校验位P' 按位进行异或操作，得 故障字S = $S_4S_3S_2S_1$
  - 根据S的值确定：无错、仅校验位错、某个数据位错

海明码举例

假定一个8位数据M为：$M_8M_7M_6M_5M_4M_3M_2M_1$= 01101010，根据上述公式求出相应的校验位为：

$P_1 = M_1 \oplus M_2 \oplus M_4 \oplus M_5 \oplus M_7 = 0 \oplus 1 \oplus 1 \oplus 0 \oplus 1 = 1$

$P_2 = M_1 \oplus M_3 \oplus M_4 \oplus M_6 \oplus M_7 = 0 \oplus 0 \oplus 1 \oplus 1 \oplus 1 = 1$

$P_3 = M_2 \oplus M_3 \oplus M_4 \oplus M_8 = 1 \oplus 0 \oplus 1 \oplus 0 = 0$

$P_4 = M_5 \oplus M_6 \oplus M_7 \oplus M_8 = 0 \oplus 1 \oplus 1 \oplus 0 = 0$

假定12位码字($M_8M_7M_6M_5P_4M_4M_3M_2P_3M_1P_2P_1$)读出后的结果是:

(1) 数据位M'=M=01101010，校验位P''=P=0011

(2) 数据位M'= 01111010，校验位P''=P=0011

(3) 数据位M'=M=01101010，校验位P''= 1011

要求分别考察每种情况的故障字。

(1) 数据位M'=M=01101010，校验位P''=P=0011，即：所有位都无错。

这种情况下，因为M'=M，所以P'= P， 因此, S = P'' ⊕ P'=P ⊕ P = 0000。

海明码举例

(2) 数据位M'= 01111010，校验位P''=P=0011，即：数据位第5位(M_5)错。

这种情况下，对M'生成新的校验位P'为：

$P_1' = M_1' \oplus M_2' \oplus M_4' \oplus M_5' \oplus M_7' = 0 \oplus 1 \oplus 1 \oplus 1 \oplus 1 = 0$

$P_2' = M_1' \oplus M_3' \oplus M_4' \oplus M_6' \oplus M_7' = 0 \oplus 0 \oplus 1 \oplus 1 \oplus 1 = 1$

$P_3' = M_2' \oplus M_3' \oplus M_4' \oplus M_8' = 1 \oplus 0 \oplus 1 \oplus 0 = 0$

$P_4' = M_5' \oplus M_6' \oplus M_7' \oplus M_8' = 1 \oplus 1 \oplus 1 \oplus 0 = 1$

故障字S为：

$S_1 = P_1' \oplus P_1'' = 0 \oplus 1 = 1$

$S_2 = P_2' \oplus P_2'' = 1 \oplus 1 = 0$

$S_3 = P_3' \oplus P_3'' = 0 \oplus 0 = 0$

$S_4 = P_4' \oplus P_4'' = 1 \oplus 0 = 1$

根据故障字的数值1001，可以判断出发生错误的位是在12位码字的第1001位(即：第9位)，在这一位上排列的是数据位M_5，所以检错正确，纠错时，只要将码字的第9位（即：第5个数据位）取反即可。

海明码举例

(3) 数据位M'=M=01101010，校验位P''= 1011，

即：校验码第4位(P_4)错。

这种情况下，因为M'=M，所以P'= P，因此故障位S为：

$S_1= P_1' \oplus P_1''=1 \oplus 1=0$

$S_2= P_2' \oplus P_2''=1 \oplus 1=0$

$S_3= P_3' \oplus P_3''=0 \oplus 0=0$

$S_4= P_4' \oplus P_4''=0 \oplus 1=1$

根据故障字的数值1000，可以判断出发生错误的位是在12位 码字的第1000位(即：第8位)，在这一位上排列的是校验位P_4，所以检错正确，不需纠错。

单纠错和双检错码

- 单纠错码（SEC）
  - 问题：上述(n=8/k=4)海明码的码距是几?
  - 码距d=3。因为，若有一位出错，则因该位至少要参与两组校验位的生成，因而至少引起两个校验位的不同。
  - 根据码距与检错、纠错能力的关系，知：这种码制能发现两位错，或对单个位出错进行定位和纠错。这种码称为单纠错码（SEC）。
- 单纠错和双检错码（SEC-DED）
  - 如果校验码同时具有发现两位错和纠正一位错的能力，则称为单纠错和双检错码（SEC-DED）。
  - 一般半导体存储器都采用这种校验码进行数据校验。
  - 若要使上述单纠错码成为SEC-DED，则码距需扩大到d= 4。为此，还需要增加一位校验位P_5，将P_5排列在码字的最前面，即：$P_5M_8M_7M_6M_5P_4M_4M_3M_2P_3M_1P_2P_1$，并使得数据中的每一位都参与三个校验位的生成。从表中可看出除了M_4和M_7参与了三个校验位的生成外，其余位都只参与了两个校验位的生成。因此P_5按下式求值：

$$P_5= M_1 \oplus M_2 \oplus M_3 \oplus M_5 \oplus M_6 \oplus M_8$$

当任意一个数据位发生错误时，必将引起三个校验位发生变化，所以码距为4

SEC-DED的检错／纠错规则

引入P_5后，故障字S也增加了一位，即$S=S_5S_4S_3S_2S_1$，根据$S_5S_4S_3S_2S_1$的取值情况，可按如下规则发现两位错并纠正一位错。

① 当$S_5S_4S_3S_2S_1$为00000时，表明无错。

② 当$S_5S_4S_3S_2S_1$中仅一位不为0时，表明由S指定的位置上那个校验位发生了错误，或是在数据和校验位中有三位同时出错，但后面这种可能性非常小，所以一般认为发生了前一种情况。

③ 当$S_5S_4S_3S_2S_1$中有两位不为0时，表明数据和校验位中有两位同时出错，此时只能发现这种错误，但无法确定是哪两位错。

④ 当$S_5S_4S_3S_2S_1$中有三位不为0时，表明有一个数据位发生了错误，或是三个校验位同时出错，但后面这种可能性非常小，所以一般认为发生了前一种情况。此时，出错的位置由$S_4S_3S_2S_1$的数值指定。

⑤ 当$S_5S_4S_3S_2S_1$中有四位或五位都不为0时，表明出错情况严重，系统可能出现故障，应检查系统硬件的正确性。

循环冗余码

循环冗余校验码（Cyclic Redundancy Check），简称CRC码

- 具很强的检错、纠错能力
- 用于大批量数据存储和传送(如：外存和通信)中的数据校验

奇偶校验码是在每个字符信息后增加一位校验位进行数据校验的，这样对大批量传输数据进行校验时，会增加大量的额外开销，尤其是在网络通信中，传输的数据信息都是二进制比特流，因而没有必要将数据再分解成一个个字符，也就无法采用奇偶校验码，因此，通常采用CRC码进行校验。

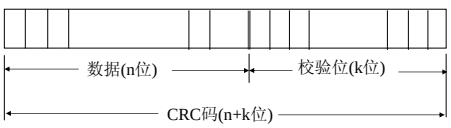
- 通过某种数学运算来建立数据和校验位之间的约定关系。

奇偶校验码和海明校验码都是以奇偶检测为手段的。

CRC码的检错方法

基本思想：

- 假设要进行校验的数据信息M(x)为一个n位的二进制数据，将M(x)左移k位后，用一个约定的"生成多项式"G(x)相除，G(x)是一个k+1位的二进制数，相除后得到的k位余数就是校验位。这些校验位拼接到M(x)的n位数据后面，形成一个n+k位的代码，我们称这个代码为循环冗余校验（CRC）码，也称（n+k,n）码。
- 一个CRC码一定能被生成多项式整除，所以当数据和校验位一起送到接受端后，只要将接受到的数据和校验位用同样的生成多项式相除，如果正好除尽，表明没有发生错误；若除不尽，则表明某些数据位发生了错误。



循环冗余码举例

校验位的生成：用一个例子来说明校验位的生成过程。

- 假设要传送的数据信息为：100011，即报文多项式为：

$M(x)= x^5 + x + 1$。数据信息位数n=6。

- 若约定的生成多项式为：$G(x)= x^3+ 1$，则生成多项式位数为4位，所以校验位位数k=3，除数为1001。
- 生成校验位时，用$x^3M(x)$去除以G(x)，即：100011000÷1001。
- 相除时采用"模 2 运算"的多项式除法。

循环冗余码举例

$X^3 \cdot M(x) \div G(x) = (x^8 + x^4 + x^3) \div (x^3 + 1)$

```
        100111
1001 ) 100011000
       1001
       ----
        0011
        0000
        ----
         0111
         0000
         ----
          1110
          1001
          ----
           1110
           1001
           ----
            1110
            1001
            ----
             111
```

(模 2 运算不考虑加法进位和减法借位，上商的原则是当部分余数首位是 1 时商取 1，反之商取 0。然后按模 2 相减原则求得最高位后面几位的余数。这样当被除数逐步除完时，最后的余数位数比除数少一位。这样得到的余数就是校验位，此例中最终的余数有3位。)

校验位为111，CRC码为100011 111。如果要校验CRC码，则可将CRC码用同一个多项式相除，若余数为0，则说明无错；否则说明有错。例如，若在接收方的CRC码也为100011 111时，用同一个多项式相除后余数为0。若接收方CRC码不为100011 111时，余数则不为0。

第二讲小结

- 非数值数据的表示
  - 逻辑数据用来表示真/假或N位位串，按位运算
  - 西文字符：用ASCII码表示
  - 汉字：汉字输入码、汉字内码、汉字字模码
- 数据的宽度
  - 位、字节、字（不一定等于字长），K/M/G/…有不同的含义
- 数据的存储排列
  - 大端方式：用MSB存放的地址表示数据的地址
  - 小端方式：用LSB存放的地址表示数据的地址
  - 按边界对齐可减少访存次数
- 数据的纠错和检错
  - 奇偶校验：适应于1字节长数据的校验
  - 海明校验：分组后，各组内用奇偶校验，用于内存储器数据的校验
  - 循环冗余校验：用在通信和外存中，适合于大批量数据的校验

附录： Decimal / Binary（十 / 二进制数）

❶ The decimal number 5836.47 in powers of 10:

$$5\times 10^3 + 8\times 10^2 + 3\times 10^1 + 6\times 10^0 + 4\times 10^{-1} + 7\times 10^{-2}$$

❶ The binary number 11001 in powers of 2 :

$$1\times 2^4 + 1\times 2^3 + 0\times 2^2 + 0\times 2^1 + 1\times 2^0 = 16 + 8 + 0 + 0 + 1 = 25$$

❶ 用一个下标表示数的基（ radix / base）

$11001_2 = 25_{10}$

附录： Octal / Hexadecimal (八 / 十六进制数)

2^{11}	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
0	1	1	1	1	1	0	1	0	0	0	0

$= 2000_{10}$

$$v = \sum_{i=0}^{n-1} 2^i b_i$$

2^3=8　　2^4=16

03720　　0x7d0

Octal - base 8	Hexadecimal - base 16	
000 - 0	0000 - 0	1000 - 8
001 - 1	0001 - 1	1001 - 9
010 - 2	0010 - 2	1010 - a
011 - 3	0011 - 3	1011 - b
100 - 4	0100 - 4	1100 - c
101 - 5	0101 - 5	1101 - d
110 - 6	0110 - 6	1110 - e
111 - 7	0111 - 7	1111 - f

计算机用二进制表示所有信息！

为什么要引入 8 / 16进制？

8 / 16进制是二进制的简便表示。便于阅读和书写！

它们之间对应简单，转换容易。

在机器内部用二进制，在屏幕或其他外部设备上表示时，转换为8/16进制数，可缩短长度

附录： Conversions of numbers

(1) R进制数 => 十进制数

按“权”展开 (a power of R)

例1: $(10101.01)_2 = 1\times2^4 + 1\times2^2 + 1\times2^0 + 1\times2^{-2} = (21.25)_{10}$

例2: $(307.6)_8 = 3\times8^2 + 7\times8^0 + 6\times8^{-1} = (199.75)_{10}$

例1: $(3A.1)_{16} = 3\times16^1 + 10\times16^0 + 1\times16^{-1} = (58.0625)_{10}$

(2)十进制数 => R进制数

整数部分和小数部分分别转换

① 整数(integral part)----“除基取余，上右下左”

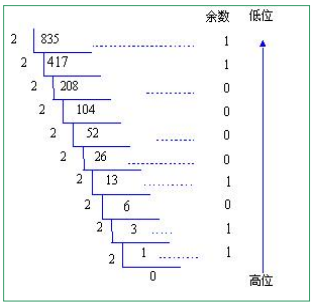
② 小数(fractional part)----“乘基取整，上左下右”

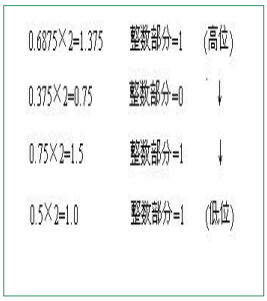
附录： Decimal to Binary Conversions

例1:$(835.6785)_{10} = (1101000011.1011)_2$

整数-----“除基取余，上右下左”

小数-----“乘基取整，上左下右”



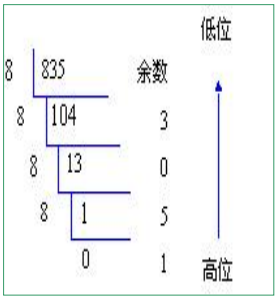


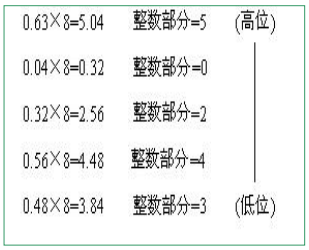
附录：Decimal to Binary Conversions

例2:$(835.63)_{10}=(1503.50243\ldots)_8$

整数----“除基取余，上右下左”　小数----“乘基取整，上左下右”

有可能乘积的小数部分总得不到0 ，此时得到一个近似值。





附录： Conversions of numbers

(3) 二/八/十六进制数的相互转换

① 八进制数转换成二进制数

$(13.724)_8=(\ 001\ 011\ .\ 111\ \ 010\ \ 100\ )_2=(1011.1110101)_2$

② 十六进制数转换成二进制数

$(2B.5E)_{16}=(00101011\ .\ 01011110)_2=(101011.0101111)_2$

③ 二进制数转换成八进制数

$(0.10101)_2=(\ 000\ .\ 101\ \ 010\ )_2=(\ 0.52)_8$

④ 二进制数转换成十六进制数

$(11001.11)_2=(\ 0001\ \ 1001\ .\ 1100\ )_2=(\ 19.C\ )_{16}$

第二章作业

2（1）、2（6）、2（7）、4、5、6、7、8、9、11、13、14、17、18、19、20

其他未列题目请自行选择练习

作业下星期二（3月6号）交