原码

补码

移码

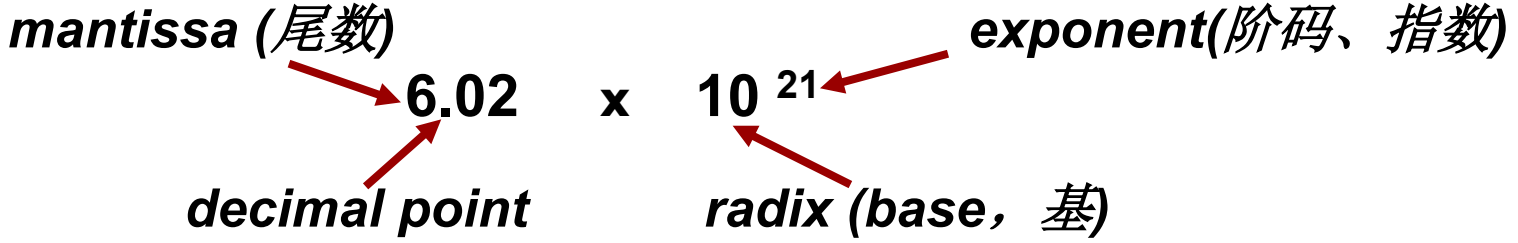
定点整数 { 无符号整数 二进制

带符号整数 补码

小数？实数？

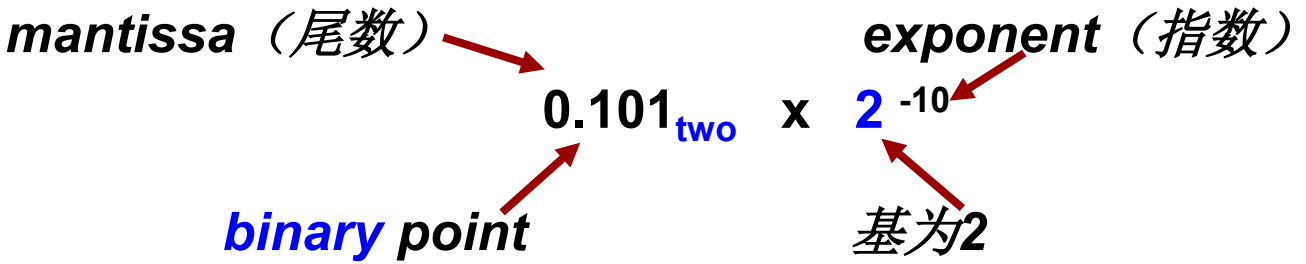
科学计数法(Scientific Notation)与浮点数

十进制的例子:



- Normalized form（规格化形式）：尾数的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}

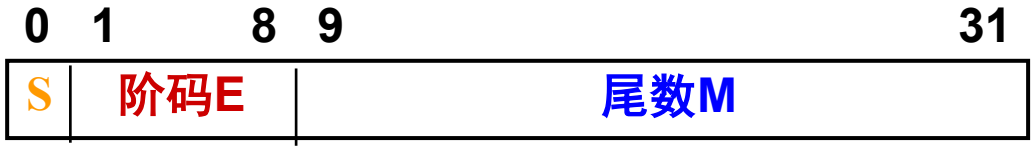
二进制的科学计数法表示：



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(Floating Point)的第1个例子

例：画出下述32位浮点数格式的规格化数的表示范围。规定如下：



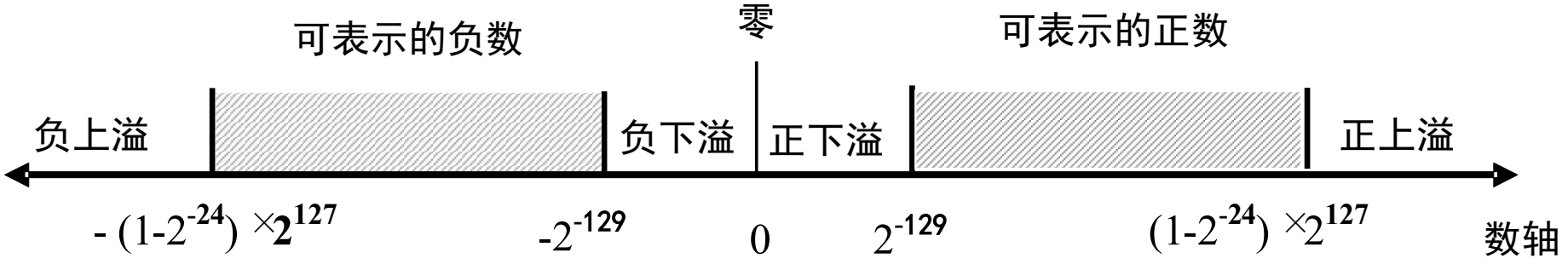
$+/-0.1xxxxx \times 2^{E}$

第0位数符S；第1～8位为8位移码表示阶码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。规格化要求尾数必须是0.1xxxx形式，也就是小数点后第一位总是1，且约定第一位默认的“1”不明显表示出来。最终就用23个数位表示24位尾数。

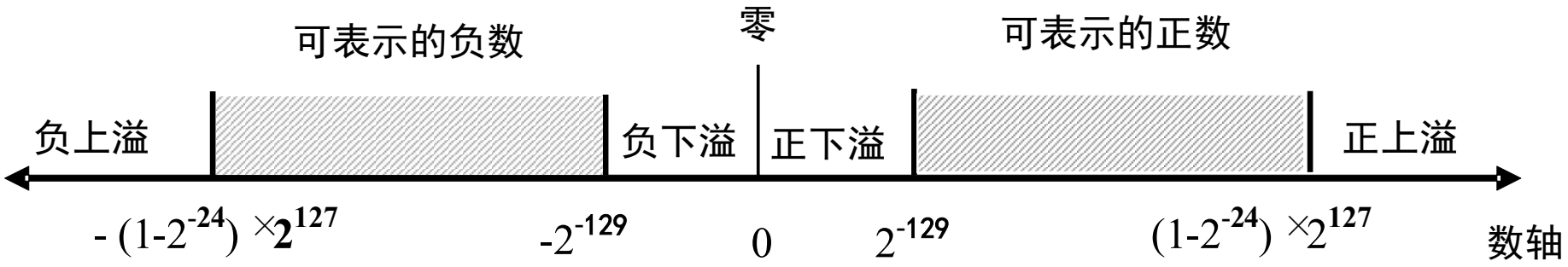
最大正数：$0.11\ldots1 \times 2^{11\ldots1} = (1-2^{-24}) \times 2^{127}$

最小正数：$0.10\ldots0 \times 2^{00\ldots0} = (1/2) \times 2^{-128}$

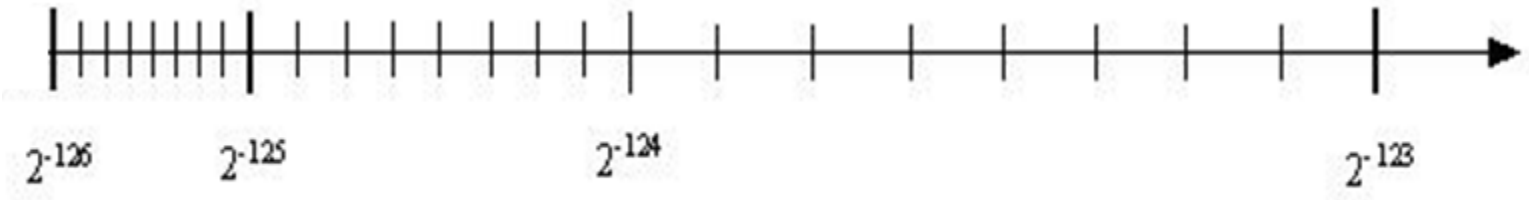
因为原码是对称的，所以其表示范围关于原点对称。



浮点数(Floating Point)的第1个例子（续）



- 机器0：尾数为0 或 落在下溢区中的数
- 浮点数范围比定点数大，但数的个数没变多，故数之间更稀疏且不均匀，也不连续（比如第二小的正数是0.10……1x$2^{00\cdots0}$ ）



浮点数还能怎么表示？

规定：小数点前总是“1”，故可隐含表示

注意：和前面例子的规定不太一样，这里更合理！（多表示了一位尾数的有效数字）

- Normal format（规格化数形式）：

  +/-1.xxxxxxxxxx × $2^{Exponent}$

- 32-bit 规格化数：

S 是符号位（Sign）

Exponent用移码来表示

Significand 表示 xxxxxxxxxxxxx，不含尾数部分小数点前面的1

- 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？

“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IEEE 754的制定

现在所有计算机都采用IEEE 754来表示浮点数

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

1989
ACM Turing
Award Winner!

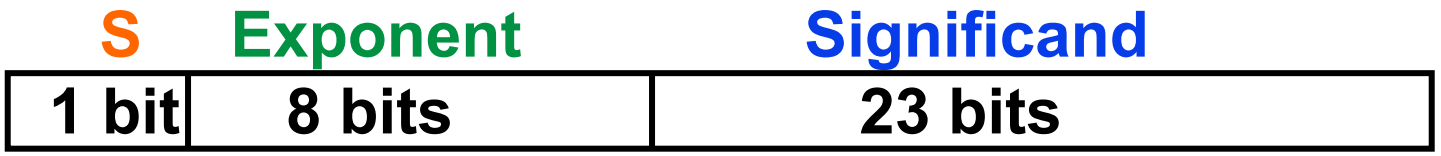
www.cs.berkeley.edu/~wkahan/ieee754status/754story.html

Prof. William Kahan

IEEE 754 浮点数标准

规格化数：+/-1.xxxxxxxxxx$_{two}$ x $2^{Exponent}$

Single Precision单精度：（双精度Double Precision 类似）



- S (符号位)：1 表示negative；0表示 positive
- Exponent（阶码 / 指数）：

  全0和全1用来表示特殊值！

  - SP规格化数阶码范围为0000 0001 (-126) ~ 1111 1110 (127)
  - bias为127 (single), 1023 (double)

  若偏置常数用128，则阶码范围为多少？

- 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）

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

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

0000 0001 (-127)

~

1111 1110 (126)

例：二进制浮点数转换为十进制真值

BEE00000H 是一个 IEEE 754 单精度浮点数的机器数表示

1	0111 1101	110 0000 0000 0000 0000 0000

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

- **Sign: 1 => negative**
- **Exponent:**
  - **$0111\ 1101_{two} = 125_{ten}$**
  - 按偏置常数计算真实指数: **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: $-1.75_{ten}\times2^{-2} = -0.4375$**

例：十进制真值转换为二进制浮点数

-12.75

1. 初始真值: -12.75
2. 整数部分转换:

   $12 = 8 + 4 = 1100_2$
3. 小数部分转换:

   $.75 = .5 + .25 = .11_2$
4. 规格化:

   $1100.11 = 1.10011 \times 2^3$
5. 阶码计算: $127 + 3 = 128 + 2 = 1000\ 0010_2$

1	1000 0010	100 1100 0000 0000 0000 0000

最终浮点数的16进制表示为：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	?

怎么表示0？

exponent: 全0

significand: 全0

sign? 1和0都行

+0: 0 00000000 00000000000000000000000

-0: 1 00000000 00000000000000000000000

怎么表示+∞/-∞ ？

∞ ： infinity

In FP, 除数为0的结果是 +/- ∞, 不是溢出异常.（整数除0为异常）

为什么要这样处理?

- 可以利用+∞/-∞作比较。 例如：X/0>Y可作为有效比较

+∞/-∞的表示：

- Exponent : 全1(11111111B = 255)
- Significand: 全0

+∞ : 0 11111111 00000000000000000000000

-∞ : 1 11111111 00000000000000000000000

可能的运算：

5.0 / 0 = +∞, -5.0 / 0 = -∞

5+(+∞) = +∞, (+∞)+(+∞) = +∞

5 - (+∞) = -∞, (-∞) - (+∞) = -∞ etc

怎么表示非数（“Not a Number”）？

Sqrt (- 4.0) = ？　　0/0 = ？

- **Called Not a Number (NaN) - “非数”**

NaN的表示：

Exponent = 255

Significand: 任意非0值

NaNs 对编程调试有帮助

可能的运算：

sqrt (-4.0) = NaN　　0/0 = NaN

op (NaN,x) = NaN　　+∞+(-∞) = NaN

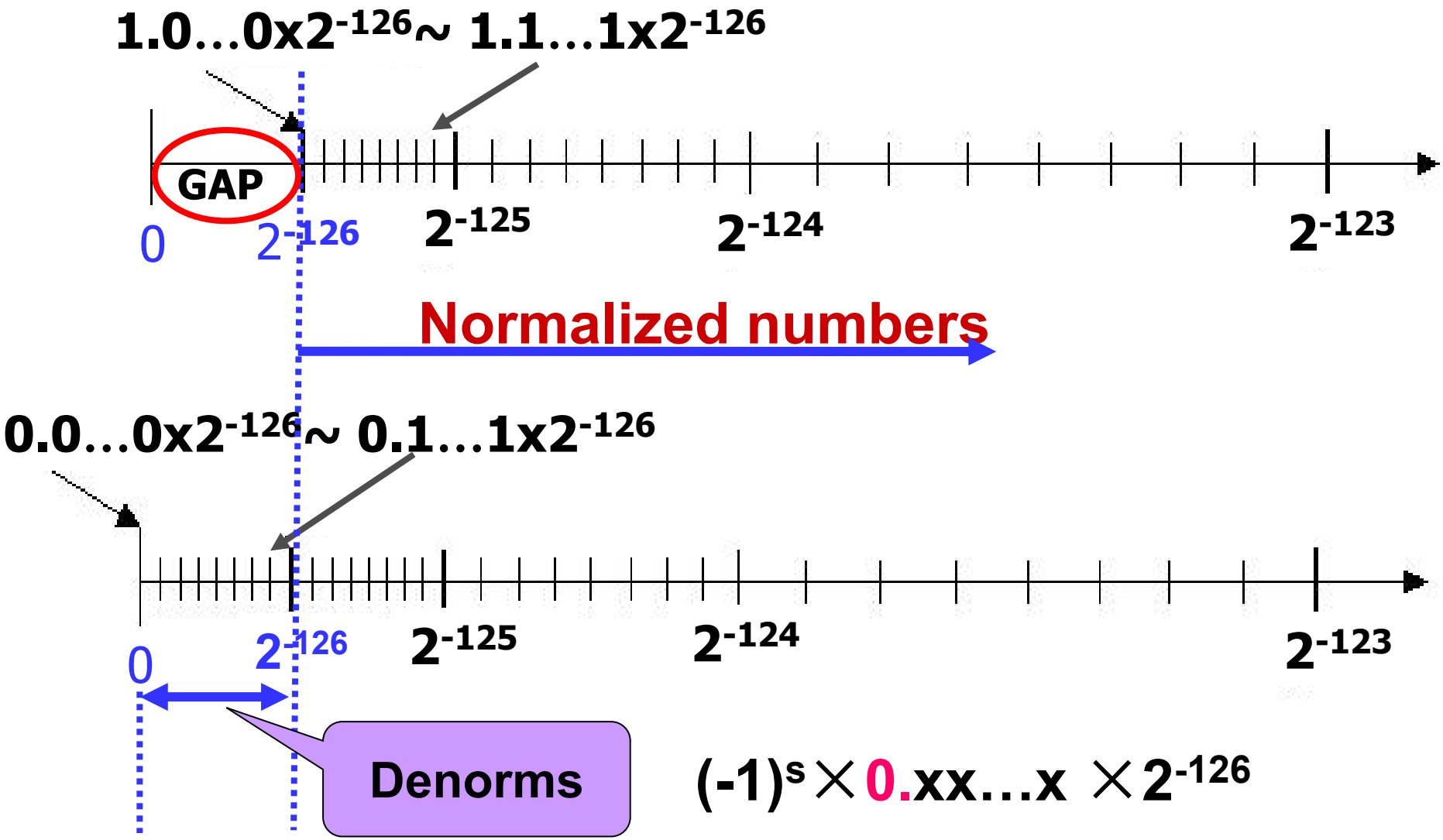
+∞- (+∞) = NaN　　∞/∞ = NaN

etc.

怎么表示Denorms(非规格化数)

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

非规格化数的表示



$(-1)^s \times 0.xx\ldots x \times 2^{-126}$

Exponent： 全0

Significand: 任意非0值

IEEE 754表示的一些问题

- 表数范围?

  单精度可表示最大正数: $+1.11\ldots1 \times 2^{127}$　　约 $+3.4 \times 10^{38}$

  双精度呢?　　约 $+1.8 \times 10^{308}$

- 数据转换时可能发生的问题? i是32位补码，f是float，d是double

  i 和 (int) ((float) i))　　不一定相等

  i 和 (int) ((double) i))　　相等

  f 和 (float) ((int) f))　　不一定相等

  d 和 (double) ((int) d))　　不一定相等

- FP参与加法时的不同计算顺序可能带来的问题?

  $x = -1.5 \times 10^{38}$,　$y = 1.5 \times 10^{38}$,　$z = 1.0$

  $(x+y)+z = (-1.5\times10^{38}+1.5\times10^{38}) +1.0 = 1.0$

  $x+(y+z) = -1.5\times10^{38}+ (1.5\times10^{38}+1.0) = 0.0$

用BCD码表示十进制数

- 编码思想： 每个十进数位（0-9）至少有4位二进制表示。而4位二进制位可组合成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
     - 每个十进制数字的4个二进制位（称为基2码）都有确定的权。
       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
     - 每个十进制数位的4个基2码没有确定的权。
     - 用的较多的是余3码和格雷码。
     - 余3码：由8421码加上0011形成。当两个十进制数字之和是10时，其二进制编码的值正好是16，而且0和9，1和8，…，5和4的余3码互为反码（各二进制位都相反）。
     - 格雷码（Gray Code）：任意两个相邻的编码只有一位二进位不同。格雷码有多种编码形式。
  3. 其他编码方案 （5中取2码、独热码等）

原码
补码
移码

定点整数 { 无符号整数 二进制
带符号整数 补码

浮点数(IEEE754)

{ 尾数（原码）
阶码（移码）

十进制数Decimal

- BCD（Binary coded Decimal）码

第三讲小结

10在计算机中有几种可能的表示？
-10呢？

- 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
- 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
- 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
  - 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
- 浮点数的表示（IEEE 754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，隐藏位为0)
- 十进制数的二进制表示（BCD码）
  - 有权BCD码（8421码）、无权BCD码（余3码、格雷码等）

第四讲　非数值数据、数据的排列和存储

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

逻辑数据的编码表示

- 表示
  - 用一位表示 。例如，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

$$\begin{array}{r} 0101 \\ 1001 \\ \hline 0001 \end{array} \qquad \begin{array}{r} 0101 \\ 1001 \\ \hline 1101 \end{array} \qquad \begin{array}{l} 0101 \\ 1010 \end{array} \qquad \begin{array}{l} 0101 \\ 0010 \end{array}$$

  - 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。
- 位串
  - 用来表示若干个状态位或控制位（OS中使用较多）

西文字符的编码表示

- **特点**
  - 是一种拼音文字，用有限几个字母可拼写出所有单词
  - 只对有限个字母和数学符号、标点符号等辅助字符编码
  - 所有字符总数不超过256个，使用7或8个二进位可表示
- **表示（常用编码为7位ASCII码）**
  - 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）

了解对应的ASCII码！

- **操作**
  - 字符串操作，如:传送/比较　等

ASCII 码表

	$b_6b_5b_4$ =000	$b_6b_5b_4$ =001	$b_6b_5b_4$ =010	$b_6b_5b_4$ =011	$b_6b_5b_4$ =100	$b_6b_5b_4$ =101	$b_6b_5b_4$ =110	$b_6b_5b_4$ =111
$b_3b_2b_1b_0$=0000	NUL	DLE	SP	0	@	P	`	p
$b_3b_2b_1b_0$=0001	SOH	DC1	!	1	A	Q	a	q
$b_3b_2b_1b_0$=0010	STX	DC2	“	2	B	R	b	r
$b_3b_2b_1b_0$=0011	ETX	DC3	#	3	C	S	c	s
$b_3b_2b_1b_0$=0100	EOT	DC4	$	4	D	T	d	t
$b_3b_2b_1b_0$=0101	ENQ	NAK	%	5	E	U	e	u
$b_3b_2b_1b_0$=0110	ACK	SYN	&	6	F	V	f	v
$b_3b_2b_1b_0$=0111	BEL	ETB	‘	7	G	W	g	w
$b_3b_2b_1b_0$=1000	BS	CAN	(	8	H	X	h	x
$b_3b_2b_1b_0$=1001	HT	EM	)	9	I	Y	i	y
$b_3b_2b_1b_0$=1010	LF	SUB	*	:	J	Z	j	z
$b_3b_2b_1b_0$=1011	VT	ESC	+	;	K	[	k	{
$b_3b_2b_1b_0$=1100	FF	FS	,	<	L	\	l	|
$b_3b_2b_1b_0$=1101	CR	GS	-	=	M	]	m	}
$b_3b_2b_1b_0$=1110	SO	RS	.	>	N	^	n	~
$b_3b_2b_1b_0$=1111	SI	US	/	?	O	_	o	DEL

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：对汉字用相应按键进行编码表示，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：描述汉字字模的点阵或轮廓，用于输出

问题：西文字符有没有输入码？有没有内码？
有没有字模点阵或轮廓描述？

汉字内码、字模点阵码(轮廓描述)

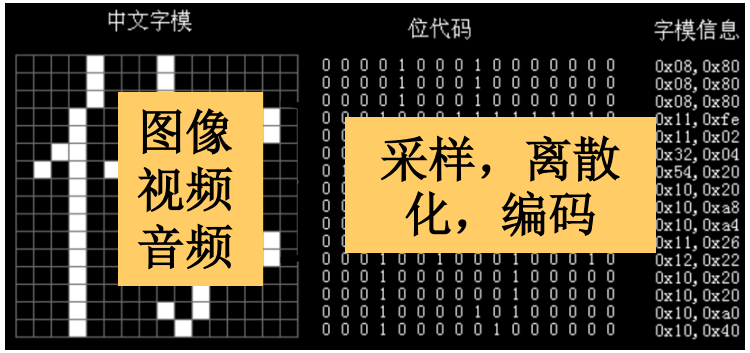
- 至少需2个字节才能表示一个汉字内码。为什么？ $2^{16}=65536$
  - 由汉字的总数决定！
- 可在GB2312国标码的基础上产生汉字内码
  - 为与ASCII码区别，将国标码的两个字节的第一位置“1”后得到一种汉字内码

区位码→国标码→内码

- 为便于打印、显示汉字，汉字字形必须预先存在机内
  - 字库 (font)：所有汉字形状的描述信息集合
  - 不同字体 (如宋体、仿宋、楷体、黑体等) 对应不同字库
  - 从字库中找到字形描述信息，然后送设备输出

  问题：如何知道到哪里找相应的字形信息？

  汉字内码与其在字库中的位置有关！！
- 字形主要有两种描述方法：
  - 字模的点阵描述（图像方式）
  - 字模的轮廓描述（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

- 字节 (Byte)，二进制信息的计量单位，也称“位组”
  - 1个字节=8个bit
  - 现代计算机中，存储器按字节编址
  - 字节是最小可寻址单位

  (addressable unit)

地址编号	存储内容
0	0010 0110
1	1010 1100
……	……
n	1101 0011

- 字 (word) 也经常用来作为数据的长度单位
  - 1个字一般来说是16个bit

“字”（数据的宽度）和 “字长”

- **“字”和 “字长”的概念不同**
  - **“字长”指定点运算数据通路的宽度。**

    （CPU内部有进行数据运算、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”等于CPU内部定点运算部件的位数、通用寄存器的宽度等 ）
  - **“字”表示被处理信息的单位，用来度量数据类型的宽度。**
  - **字和字长的宽度可以一样，也可不同。**

    例如，x86体系结构定义“字”的宽度为16位

    但从386开始字长就是32位了。

    再比如：IA-32中的“字”有16位，字长32位

    DWORD ： 32位、QWORD：64位 （都是数据的宽度单位）

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
- 主存容量经常使用的单位，如：
  - “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
- 主频和带宽使用的单位，如：
  - “千比特/秒”(kb/s)，1kbps=10^3 b/s=1000 bps
  - “兆比特/秒”(Mb/s)，1Mbps=10^6 b/s =1000 kbps
  - “千兆比特/秒”(Gb/s)，1Gbps=10^9 b/s =1000 Mbps
  - “兆兆比特/秒”(Tb/s)，1Tbps=10^{12} b/s =1000 Gbps
- 硬盘和文件使用的单位
  - 不同的硬盘制造商和操作系统用不同的度量方式，因而比较混乱
  - 为避免歧义，国际电工委员会（IEC）在1998年给出了表示2的幂次的二进制前缀字母定义

数据量的度量单位

- **硬盘和文件使用的单位**
  - **不同的硬盘制造商和操作系统用不同的度量方式，因而比较混乱**
  - **为避免歧义，国际电工委员会（IEC）给出了二进制前缀字母定义，可用不同的前缀表示所采用的度量方式**

十进制前缀			IEC 定义的二进制前缀			值差
单词	前缀	值	单词	前缀	值	（%）
kilobyte	KB/kB	10^3	kibibyte	KiB	2^{10}	2%
megabyte	MB	10^6	mebibyte	MiB	2^{20}	5%
gigabyte	GB	10^9	gibibyte	GiB	2^{30}	7%
terabyte	TB	10^{12}	tebibyte	TiB	2^{40}	10%
petabyte	PB	10^{15}	pebibyte	PiB	2^{50}	13%
exabyte	EB	10^{18}	exbibyte	EiB	2^{60}	15%
zettabyte	ZB	10^{21}	zebibyte	ZiB	2^{70}	18%
yottabyte	YB	10^{24}	yobibyte	YiB	2^{80}	21%

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
- 不同机器上表示的同一种类型的数据可能宽度不同
- 程序中的数据有相应的机器级表示方式和相应的处理指令
  (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

- 机器数的位排列顺序有两种方式： （例：32位字: $0\ldots01011_2$）
  - 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
  - 用LSB(Least Significant Bit)来表示最低有效位
  - 用MSB来表示最高有效位

LSB

MSB

- 但是——
  - 80年代开始，几乎所有通用机器都用字节编址
  - ISA设计时要考虑的两个问题：
    - 如何根据一个地址取到一个32位的字？ - 字的存放问题
    - 一个字能否存放在任何地址边界？ - 字的边界对齐问题

- 如果以字节为一个排列单位，则LSB(Least Significant Byte)表示最低有效字节，MSB表示最高有效字节

数据的存储和排列顺序

例如，若 int i = -65535，存放在内存100号单元（即占100# ~ 103#），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

0000 0000 0000 0000 1111 1111 1111 1111

1111 1111 1111 1111 0000 0000 0000 0001

$65535=2^{16}-1$

$[-65535]_{补}$=FFFF0001H

Word:

FF	FF	00	01
103	102	101	100
MSB			LSB
100	101	102	103

little endian word 100#

big endian word 100#

大端方式（Big Endian）：MSB所在的地址是数的地址

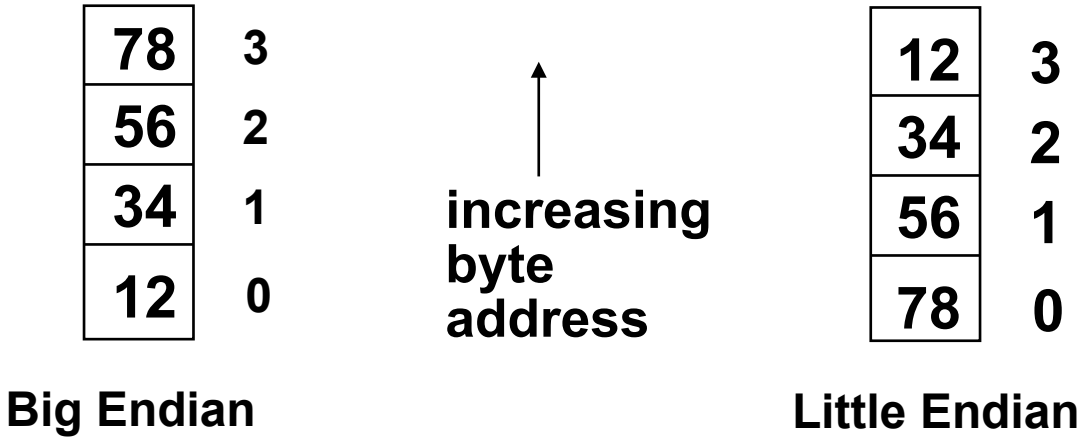
e.g. IBM 360/370, Motorola 68k, MIPS, Sparc, HP PA

小端方式（ Little Endian）：LSB所在的地址是数的地址

e.g. Intel 80x86, DEC VAX

有些机器两种方式都支持，可通过特定控制位来设定采用哪种方式。

Byte Swap Problem（字节交换问题）



上述存放在0号单元的数据（占4个字节）是什么？ 12345678H

存放方式不同的机器间程序移植或数据通信时需要进行顺序转换

音、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

第四讲小结

- 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/K/M/G/…有不同的含义
- 数据的存储排列
  - 数据的地址：连续若干单元中最小的地址，即：从小地址开始存放数据
    - 若一个short型（16位）数据si存放在单元0x0100和0x0101中，那么si的地址是什么——0x0100
  - 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

本章作业

- **教材第1章习题：3、4、5、6、7、8、9、17**
- **作业提交截止时间：9.23**
- **https://cslab.nju.edu.cn/main.htm**