

A scenic landscape featuring a calm lake in the foreground, a dense forest of trees with autumn foliage in the middle ground, and rolling mountains in the background under a clear blue sky with scattered white clouds. The text '上海交通大学' is overlaid in the upper center.

上海交通大学

网络学院

微机原理与应用

The Principle & Application of Microcomputer

王春香 副教授

wangcx@sjtu.edu.cn

第二章 微机运算基础

主要内容

2.1 进位计数制

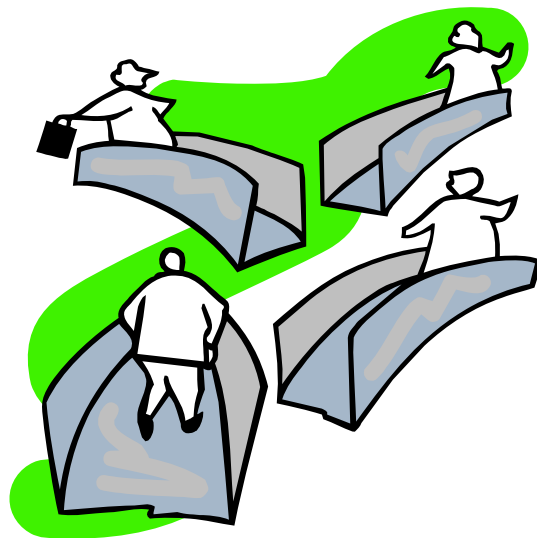
2.2 进位数制之间的转换

2.3 二进制编码

2.4 二进制数的运算

2.5 数的定点与浮点表示

2.6 带符号数的表示法



第二章 微机运算基础

■ 学习要求

- ◆ 理解进位记数制的基本特点。
- ◆ 熟练掌握各种进位记数制之间相互转换的方法。
- ◆ 掌握常用的8421 BCD编码和ASCII编码。
- ◆ 熟练掌握二进制数的各种算术运算与逻辑运算方法。
- ◆ 理解数的定点和浮点表示法。
- ◆ 理解和熟练掌握补码及其运算与溢出。

2.1 进位计数制



□ 基本概念

进位计数制(简称进位制): 利用符号按照**进位原则**来计数的方法, 一种进位计数制包含一组**数码符号**和两个基本因素(**基数, 权**)。

数码(Number): 用不同的**数字符号**来表示一种数制的数值, 这些数字符号称为“数码”。

例如: 十进制数码 (0, 1, 2, ..., 9)

基数(Radix, 也称底数): 数制中所使用的**数码个数**称为该计数制的“基数”。

例如: 十进制有10个数码, 因此基数为10, 逢十进一

2.1 进位计数制



□ 基本概念

位权(Weight): 某数制中，每一位所具有的值称为“位权”，用基数的n次幂来表示。

例如：十进制中位权表示为， 10^{-2} (百分位)， 10^{-1} (十分位)， 10^0 (个位)， 10^1 (十位)

结论：在各种进位计数制中，十进制是人们最熟悉的，二进制在计算机内使用，八进制和十六进制则可看成二进制的压缩形式。



2.1 进位计数制

□ 十进制 (Decimal Number)

数码: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

基数: 10

位权: 10^i , $i = \dots, 3, 2, 1, 0, -1, -2, -3, \dots$

规则: 逢十进一

表示: $(999.99)_{10}$, 或者 $(999.99)D$, 或者 999.99

例如:

$$(999.99)_{10} = 9 \times 10^2 + 9 \times 10^1 + 9 \times 10^0 + 9 \times 10^{-1} + 9 \times 10^{-2}$$

$$(15)_{10} = ?$$



2.1 进位计数制

□ 二进制 (Binary Number)

数码: 0, 1

基数: 2

位权: 2^i , $i = \dots, 3, 2, 1, 0, -1, -2, -3, \dots$

规则: 逢二进一

表示: $(1101.11)_2$, 或者 $(1101.11)_B$

例如: 二进制 转换成 十进制

$$\begin{aligned}(1101.11)_2 &= 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-2} \\ &= 8 + 4 + 0 + 1 + 0.5 + 0.25 = 13.75_{10}\end{aligned}$$

$$(1111)_2 = ?$$

结论: 计算机内使用的是二进制编码(也称为基2码), 容易实现、规则简单、运算方便。



2.1 进位计数制

□ 八进制 (Octale Number)

数码: 0, 1, 2, 3, 4, 5, 6, 7

基数: 8

位权: 8^i , $i = \dots, 3, 2, 1, 0, -1, -2, -3, \dots$

规则: 逢八进一

表示: $(257)_8$, 或者 $(257)O$, 或者 $(257)Q$

例如: 八进制 转换成 十进制

$$\begin{aligned}(257)_8 &= 2 \times 8^2 + 5 \times 8^1 + 7 \times 8^0 \\ &= 128 + 40 + 7 = (175)_{10}\end{aligned}$$

$$(17)_8 = ?$$



2.1 进位计数制

□ 十六进制 (Hexadecimal Number)

数码: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9,
A, B, C, D, E, F

基数: 16

位权: 16^i , $i = \dots, 3, 2, 1, 0, -1, -2, -3, \dots$

规则: 逢十六进一

表示: $(257)_{16}$, 或者 $(257)H$

例如: 十六进制 转换成 十进制

$$\begin{aligned}(257)_{16} &= 2 \times 16^2 + 5 \times 16^1 + 7 \times 16^0 \\ &= 2 \times 256 + 5 \times 16 + 7 = (599)_{10}\end{aligned}$$

$$(F)_{16} = ?$$



2.1 进位计数制

□ 小结1

数 N 按照位权展开的一般通式为：

$$N = \pm \sum_{i=n-1}^{-m} (k_i \cdot b^i)$$

其中： k_i — 第 i 位的数码；

b — 为基数；

b^i — 为第 i 位的权；

n — 为整数的总位数；

m — 为小数的总位数。

例如：十六进制数 **FFFE.A3 H**

$$\begin{aligned} (\text{FFFE.A3})_{16} &= \text{F} \times 16^3 + \text{F} \times 16^2 + \text{F} \times 16^1 + \text{E} \times 16^0 + \text{A} \times 16^{-1} + 3 \times 16^{-2} \\ &= ? \end{aligned}$$



2.1 进位计数制

□ 小结2

各种数制对照表

十六进制	十进制	八进制	二进制
0	0	0	0000
1	1	1	0001
2	2	2	0010
3	3	3	0011
4	4	4	0100
5	5	5	0101
6	6	6	0110
7	7	7	0111
8	8	10	1000
9	9	11	1001
A	10	12	1010
B	11	13	1011
C	12	14	1100
D	13	15	1101
E	14	16	1110
F	15	17	1111



2.2 进位数制之间的转换

□ 二进制数转换为十进制数

- ◆ 方法1：按权展开多项式和的形式
- ◆ 方法2：整数部分、小数部分分别转换

➤ 整数部分（从最高位开始，连续乘2）

假设5位二进制整数N，表示为

$$\begin{aligned} N &= K_4 K_3 K_2 K_1 K_0 \\ &= K_4 \times 2^4 + K_3 \times 2^3 + K_2 \times 2^2 + K_1 \times 2^1 + K_0 \times 2^0 \\ &= \{[(K_4 \times 2 + K_3) \times 2 + K_2] \times 2 + K_1\} \times 2 + K_0 \end{aligned}$$



2.2 进位数制之间的转换

□ 二进制数转换为十进制数

例如：二进制整数10111，转化为十进制数为

$$\begin{aligned} N &= \{[(K_4 \times 2 + K_3) \times 2 + K_2] \times 2 + K_1\} \times 2 + K_0 \\ &= \{[(1 \times 2 + 0) \times 2 + 1] \times 2 + 1\} \times 2 + 1 \\ &= 23D \end{aligned}$$

结果：二进制整数10111，转化为十进制数23



2.2 进位数制之间的转换

□ 二进制数转换为十进制数

➤ 小数部分（从最低位开始，连续除2）

假设4位二进制小数N，表示为

$$\begin{aligned} N &= K_{-1}K_{-2}K_{-3}K_{-4} \\ &= K_{-1} \times 2^{-1} + K_{-2} \times 2^{-2} + K_{-3} \times 2^{-3} + K_{-4} \times 2^{-4} \\ &= 2^{-1} \times \{K_{-1} + 2^{-1} \times [K_{-2} + 2^{-1} \times (K_{-3} + 2^{-1} \times K_{-4})]\} \end{aligned}$$



2.2 进位数制之间的转换

□ 二进制数转换为十进制数

例如：二进制小数 .1011，转化为十进制小数为

$$N = 2^{-1} \times \{1 + 2^{-1} \times [0 + 2^{-1} \times (1 + \underbrace{2^{-1} \times 1}_{0.5})]\}$$

Diagram illustrating the conversion of the binary fraction .1011 to the decimal fraction 0.6875 using a nested multiplication method. The diagram shows the following steps:

- Innermost term: $2^{-1} \times 1 = 0.5$
- Next level: $0 + 2^{-1} \times 0.5 = 0.25$
- Next level: $1 + 2^{-1} \times 0.25 = 1.125$
- Outermost term: $2^{-1} \times 1.125 = 0.5625$

The final result is 0.6875, which is the sum of the contributions from each bit: $0.5 + 0.125 + 0.0625 + 0.00625 = 0.6875$.

结果：二进制小数 .1011，转化为十进制小数0.6875



2.2 进位数制之间的转换

□ 十进制数转换为二进制数

➤ 整数部分（除2逆取余）

2		175		
2		87		1
2		43		1
2		21		1
2		10		1
2		5		0
2		2		1
2		1		0
		0		1

余数

最低位

最高位

所以：175 D = 10101111 B



2.2 进位数制之间的转换

□ 十进制数转换为二进制数

➤ 小数部分（乘2顺取整）

最高位	整数	0.625
		x 2
	1	<u> </u>
		.250
		x 2
	0	<u> </u>
		.500
		x 2
最低位	1	<u> </u>
		.000

所以：0.625 D = 0.101 B



2.2 进位数制之间的转换

□ 八进制数与二进制数之间的转换

◆ 二进制转换为八进制

➤ 整数部分:

从小数点左边第一位开始，每3位一组，最高位不足补0。

例如：二进制整数 10101001B，转化为八进制数为

$$\begin{array}{cccc} \textcircled{0}10 & 101 & 001 & \text{B} \\ \hline 2 & 5 & 1 & \text{Q} \end{array}$$

所以：10101001 B = 251 Q



2.2 进位数制之间的转换

□ 八进制数与二进制数之间的转换

◆ 二进制转换为八进制

➤ 小数部分:

从小数点右边第一位开始，每3位一组，最低位不足补0。

例如：二进制小数 .01101011B，转化为八进制数为

$$\begin{array}{ccc} \underline{.011} & \underline{010} & \underline{11\textcircled{0}} \\ 3 & 2 & 6 \end{array} \text{ B}$$

所以：0.01101011 B = 0.326 Q



2.2 进位数制之间的转换

□ 八进制数与二进制数之间的转换

◆ 八进制转换为二进制

将八进制数的每1位，用3位二进制数替代，去掉无意义的零。

例如：八进制整数 352.14 Q，转化为二进制数为

$$\begin{array}{ccccccc} \underline{3} & \underline{5} & \underline{2} & . & \underline{1} & \underline{4} & \text{Q} \\ \textcircled{0}11 & 101 & 010 & . & 001 & 1\textcircled{00} & \text{B} \end{array}$$

所以：352.14 Q = 11101010.0011 B



2.2 进位数制之间的转换

□ 八进制数与十进制数之间的转换

◆ 八进制数转换为十进制数

按权位展开，然后相加

例如：八进制数 372.01 Q，转化为十进制数为

$$\begin{aligned} 372.01Q &= 3 \times 8^2 + 7 \times 8^1 + 2 \times 8^0 + 0 \times 8^{-1} + 1 \times 8^{-2} \\ &= 192 + 56 + 2 + 0.015625 \\ &= 250.015625 \quad D \end{aligned}$$



2.2 进位数制之间的转换

□ 八进制数与十进制数之间的转换

◆ 十进制数转换为八进制数

➤ 整数部分（除8逆取余）

8		175		余数	
8		21		7	最低位
8		2		5	
		0		2	最高位

所以：175 D = 257 Q



2.2 进位数制之间的转换

□ 八进制数与十进制数之间的转换

◆ 十进制数转换为八进制数

➤ 小数部分（乘8顺取整）

	整数	0.315
最高位		$\begin{array}{r} \times 8 \\ \hline 8 \end{array}$
	2	$\begin{array}{r} .520 \\ \times 8 \\ \hline 8 \end{array}$
	4	$\begin{array}{r} .160 \\ \times 8 \\ \hline 8 \end{array}$
	1	$\begin{array}{r} .280 \\ \times 8 \\ \hline 8 \end{array}$
最低位	2	$\begin{array}{r} .240 \end{array}$

所以：0.315 D = 0.2412 Q



2.2 进位数制之间的转换

□ 十六进制数与二进制数之间的转换

◆ 二进制转换为十六进制

➤ 整数部分：

从小数点左边第1位开始，每4位一组，最高位不足补0。

例如：二进制整数110100110 B，转化为十六进制数为

$$\begin{array}{ccc} \underline{0001} & \underline{1010} & \underline{0110} & \text{B} \\ 1 & A & 6 & \end{array}$$

所以：110100110 B = 1A6 H



2.2 进位数制之间的转换

□ 十六进制数与二进制数之间的转换

◆ 二进制转换为十六进制

➤ 小数部分

从小数点右边第1位开始，每4位一组，最低位不足补0。

例如：二进制小数 .110101B，转化为十六进制数为

$$\begin{array}{cc} \underline{.1101} & \underline{01\textcircled{00}} & \text{B} \\ D & 4 & \end{array}$$

所以：0.110101 B = 0.D4 H



2.2 进位数制之间的转换

□ 十六进制数与二进制数之间的转换

◆ 十六进制转换为二进制

将十六进制数的每1位，用4位二进制数替代，去掉无意义的0。

例如：十六进制数 C8F.49 H，转化为二进制数为

<u>C</u>	<u>8</u>	<u>F</u>	.	<u>4</u>	<u>9</u>	H
1010	1000	1111	.	0100	1001	B

所以： C8F.49 H
= 1010 1000 1111. 0100 1001 B



2.2 进位数制之间的转换

□ 十六进制数与十进制数之间的转换

◆ 十进制数转换为十六进制数

➤ 整数部分（除16逆取余）

		余数	
16	<u>1192</u>		
16	<u>74</u>	8	最低位
16	<u>4</u>	A	
	0	4	最高位

所以：1192 D = 4A8 H



2.2 进位数制之间的转换

□ 十六进制数与十进制数之间的转换

◆ 十进制数转换为十六进制数

➤ 小数部分（乘16顺取整）

	整数	0.9032
		x 16
最高位	E	.4512
		x 16
	7	.2192
		x 16
	3	.5072
		x 16
最低位	8	.1152

所以：0.9032 D = 0.E738 H



2.2 进位数制之间的转换

□ 小结1

1. 二进制、八进制、十六进制数**转换为**十进制数，只需按照**位权展开**，然后**求和**即可。
2. 十进制数**转换为**二进制数(或者八进制、十六进制数)，
整数部分采用“除2(或8、16)逆取余”方法，即第一个余数为最低位，最后一个余数为最高位。
小数部分采用“乘2(或8、16)顺取整”方法，即第一个整数为最高位，最后一个整数为最低位。

注意：小数转换不一定能算尽，只能算到一定精度的位数为止，故要产生一些误差。不过当位数足够多时，这个误差就很小了。



2.2 进位数制之间的转换

□ 小结2

3. 二进制数**转换为**八进制数(或十六进制数)，以**小数点为分界线**，3位(或4位)分为一组，最左与最右一组不足3位(或4位)时补零，然后每3位(或4位)写成对应的八进制数(或十六进制数)即可。

八进制数(或十六进制数)**转换为**二进制数，每1位用相应的3位(或4位)二进制数代替即可，去除最高位前面和最低位后面多余的零。



2.3 二进制编码

□ 二进制编码的十进制 (BCD – Binary Coded Decimal)

BCD编码表

十进制	BCD码	十进制	BCD码
0	0000	10	0001 0000
1	0001	11	0001 0001
2	0010	12	0001 0010
3	0011	13	0001 0011
4	1000	14	0001 0100
5	1001	15	0001 0101
6	1010		
7	0111		
8	1000		
9	1001		

BCD码：用二进制代码对十进制数进行编码，它既具有二进制码的形式(四位二进制码)，又有十进制数的特点(每四位二进制码是一位十进制数)。

二进制与BCD码之间的转换，需经过**十进制**。

2.3 二进制编码



□ 二进制编码的十进制 (BCD – Binary Coded Decimal)

例1：十进制数 256，BCD码为

$$(256)_D = (0010\ 0101\ 0110)_{BCD}$$

例2：十进制数 0.764，BCD码为

$$(0.764)_D = (0.0111\ 0110\ 0100)_{BCD}$$

2.3 二进制编码



□ 二进制编码的十进制 (BCD – Binary Coded Decimal)

例3: BCD码转换为十进制数

$$(0110\ 0010\ 1000.1001\ 0101\ 0100)\ \text{BCD} = (628.954)\text{D}$$

例4: 二进制数转换为BCD码

$$\begin{aligned}(1011.01)\ \text{B} &= (1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2})\text{D} \\ &= 11.25\ \text{D} \\ &= (0001\ 0001.0010\ 0101)\ \text{BCD}\end{aligned}$$

2.3 二进制编码



□ 二进制编码的十进制 (BCD – Binary Coded Decimal)

8421码：编码值与 ASCII 码字符0到9的的低4位码相同，易于实现人机联系。但比纯二进制编码效率低。

余3码：是在8421码的基础上，把每个代码都加0011码而形成的。它的主要优点是执行十进制数相加时，能正确地产生进位信号，而且还给减法运算带来了方便。

格雷码：循环码中的一种，任何两个相邻的代码只有一个二进制位的状态不同，有利于抗干扰。

2.3 二进制编码



□ 字母与字符的编码

美国国家信息交换标准码，ASCII码

(American national Standard Code for Information Interchange)

7位 ASCII 码表示 $2^7=128$ 种不同的字符，包括

可显示字符(94个): 阿拉伯数字(10个): 0~9

英文大小写字母(52个): A~Z, a~z

西文符号(32个): 如!, <, },等

控制符(34个): 如 NUL(空白), CR(回车), ...等



2.3 二进制编码

□ 7位ASCII码表

高位 \ 低位		0	1	2	3	4	5	6	7
		000	001	010	011	100	101	110	111
0	0000	NUL 空白	DLE 数据链换码	SP 空格	0	@	P	,	p
1	0001	SOH 标题开始	DC1 设备控制1	!	1	A	Q	a	q
2	0010	STX 文本开始	DC2 设备控制2	”	2	B	R	b	r
3	0011	ETX 文本结束	DC3 设备控制3	#	3	C	S	c	s
4	0100	EOT 传输结束	DC4 设备控制4	\$	4	D	T	d	t
5	0101	ENQ 询问	NAK 否定应答	%	5	E	U	e	u
6	0110	ACK 应答	SYN 空转同步	&	6	F	V	f	v
7	0111	BEL 报警符	ETB 信息组传输结束	'	8	G	W	g	w
8	1000	BS 退一格	CAN 删去符	(	8	H	X	h	x
9	1001	HT 横向列表	EM 信息结束	)	9	I	Y	i	y
A	1010	LF 换行	SUB 减	*	:	J	Z	j	z
B	1011	VT 垂直列表	ESC 换码	+	;	K	[	k	{
C	1100	FF 走纸控制	FS 文件分隔符	,	<	L	\	l	
D	1101	CR 回车	GS 组分分隔符	-	=	M	]	m	}
E	1110	SO 移位输出	RS 记录分隔符	.	>	N	↑	n	-
F	1111	SI 移位输入	US 单元分隔符	/	?	O	←	o	DEL 作废字符

2.4 二进制运算



□ 二进制的算术运算

一种数制的基本算术运算：**加法**和**减法**。

利用加法和减法可进行**乘法**、**除法**以及**其它**数值运算。

2.4 二进制运算



□ 二进制的算术运算

◆ 二进制加法

运算法则：

$$0+0=0$$

$$0+1=1$$

$$1+1=10 \text{（产生了进位1）}$$

$$1+1+1=11 \text{（产生了进位1）}$$

2.4 二进制运算



◆ 二进制加法实例1

例1：1101和1011相加

1 1 1 1	进位
1 1 0 1	被加数
+ 1 0 1 1	加数
1 1 0 0 0	和

结论：两个二进制数相加时，每一位是被加数、加数和低位的进位三个数的相加。

2.4 二进制运算



◆ 二进制加法实例2

例2: 10001111B和10110101B相加

$$\begin{array}{r} 1\ 0\ 1\ 1\ 1\ 1\ 1\ 1 \\ 1\ 0\ 1\ 1\ 0\ 1\ 0\ 1 \\ +\ 1\ 0\ 0\ 0\ 1\ 1\ 1\ 1 \\ \hline 1\ 0\ 1\ 0\ 0\ 0\ 1\ 0\ 0 \end{array}$$

进位

被加数

加数

和

2.4 二进制运算



□ 二进制的算术运算

◆ 二进制减法

运算法则：

$$0-0=0$$

$$1-1=0$$

$$1-0=1$$

$$0-1=1 \text{ (产生了借位1)}$$

2.4 二进制运算



◆ 二进制减法实例1

例1: 11011B 和1101B 相减

	0	10	10	1	1	借位后的被减数
	1	1	0	1	1	被减数
—	0	1	1	0	1	减数
		1	1	1	0	差

结论: 两个二进制数相减时, 每一位是**被减数**、**减数**和低位来的**借位**三个数的相减。

首先求被减数与借位的差, **再用**这个差当作被减数, 从中减去减数。



2.4 二进制运算

◆ 二进制减法实例2

例2: 11000100B和00100101B1011相减

	1	0	1	1	1	10	1	10	借位后的被减数
	1	1	0	0	0	1	0	0	被减数
—	0	0	1	0	0	1	0	1	减数
	1	0	0	1	1	1	1	1	差

2.4 二进制运算



□ 二进制的算术运算

◆ 二进制乘法（边乘、边加的方法）

运算法则：

$$0 \times 0 = 0$$

$$1 \times 1 = 1$$

$$1 \times 0 = 0$$

$$0 \times 1 = 0$$

2.4 二进制运算



◆ 二进制乘法（边乘、边加的方法）实例1

例1：1111B 和1101B 相乘

$$\begin{array}{r} 1111 \text{ — 被乘数} \\ \times 1101 \text{ — 乘数} \\ \hline 1111 \text{ — 第1次部分积} \\ 0000 \\ \hline 01111 \text{ — 第2次部分积} \\ 1111 \\ \hline 1001011 \text{ — 第3次部分积} \\ 1111 \\ \hline 11000011 \text{ — 第4次部分积} \end{array}$$

结论：从乘数的低位开始，用乘数的每一位分别去乘被乘数，所得的各中间结果的最低有效位与相应的乘数位对齐，最后把这些中间结果同时相加即得到最后乘积。

2.4 二进制运算



□ 二进制的算术运算

◆ 二进制乘法（被乘数左移+部分积）

运算法则：

$$0 \times 0 = 0$$

$$1 \times 1 = 1$$

$$1 \times 0 = 0$$

$$0 \times 1 = 0$$



2.4 二进制运算

◆ 二进制乘法（被乘数左移+部分积）实例

例如：1111B 和1101B 相乘

乘数	被乘数	部分积	
1101	1111	0000	部分积初值
		+ 1111	被乘数
	11110	1111	部分积
	111100		
		1111	部分积
	1111000		
		1111000	左移2次的被乘数
	1111000	1001011	部分积
		1111000	左移3次的被乘数
		11000011	

乘数为1，把被乘数加至部分积(初值为0)上，然后被乘数左移1位

乘数为0，不加被乘数，把被乘数左移1位

乘数为1，把被乘数加至部分积上，然后被乘数左移1位

乘数为1，把被乘数加至部分积上，得到最终乘积。

结论：在计算机中，乘法运算是用移位和相加的操作来实现的，需要 $2n$ 个加法器。



2.4 二进制运算

◆ 二进制乘法（部分积右移+被乘数）实例

例如：1111B 和1101B 相乘

乘数	被乘数	部分积	
1101	1111	0000	部分积初值
		+ 1111	被乘数
		1111	部分积
		0111 1	右移的部分积
		0011 11	右移的部分积
		+ 1111	被乘数
		10010 11	部分积
		1001 011	右移的部分积
		+ 1111	被乘数
		11000 011	部分积
		1100 0011	最终乘积

乘数最低位为1，把被乘数加至部分积(初值为0)上，然后部分积右移1位

乘数为0，不加被乘数，部分积右移1位

乘数为1，把被乘数加至部分积上，然后部分积右移1位

乘数为1，把被乘数加至部分积上，然后部分积右移1位，得到最终乘积。

结论：在计算机中，乘法运算是用移位和相加的操作来实现的，需要n个加法器。



2.4 二进制运算

□ 二进制的算术运算

◆ 二进制除法：方法1

应用乘法规则可实现除法运算，从被除数**最高位**开始，找到足以减去除数的位数商1，再从被除数**减去除数**，依次除下去……

例如：100011B 除以101B

		0 0 0 1 1 1	商
除数 101	)	1 0 0 0 1 1	被除数
		1 0 1	
		1 1 1	余数
		1 0 1	
		1 0 1	余数
		1 0 1	
		0	余数



2.4 二进制运算

□ 二进制的算术运算

◆ 二进制除法：方法2

余数（最初为被除数）左移1位，减除数；
余数大于等于除数，商为1，否则商为0

例如：100011B 除以101B

1 0 0 0 1 1	被除数	商	最高位
1 0 0 0 1 1	被除数左移一位		
— 1 0 1	减去除数，够减商为	1	
1 1 1 1	余数		
1 1 1 1	余数左移一位		
— 1 0 1	减去除数，够减商为	1	
1 0 1	余数		
1 0 1	余数左移一位		
— 1 0 1	减去除数，够减商为	1	
0			最低位

2.4 二进制运算



□ 二进制数的逻辑运算

计算机中，0和1两种取值表示的变量称之为**逻辑变量**，代表所研究问题的两种状态或可能性。

3种逻辑运算：逻辑加法(或运算)，逻辑乘法(与运算)，逻辑否定(非运算)

逻辑运算只在对应位之间进行运算。

2.4 二进制运算



□ 二进制数的逻辑运算

◆ 与运算

与运算表示符号：“ $\times$ ”或“ $\bullet$ ”或“ $\wedge$ ”

运算法则：

$$0 \times 0 = 0$$

$$1 \times 1 = 1$$

$$1 \times 0 = 0$$

$$0 \times 1 = 0$$

结论：只有参加运算的逻辑变量都取值为1时，其**与运算**的结果才等于1。

2.4 二进制运算



□ 二进制数的逻辑运算

◆ 或运算

或运算表示符号：“+”或“ $\vee$ ”

运算法则：

$$0 + 0 = 0$$

$$0 + 1 = 1$$

$$1 + 0 = 1$$

$$1 + 1 = 1$$

结论：只要参加运算的逻辑变量中有一个为1，其或运算的结果就为1。

2.4 二进制运算



□ 二进制数的逻辑运算

◆ 非运算

非运算又称为逻辑否定。逻辑变量上方加一横线表示。

运算法则：

$$\bar{0} = 1$$

$$\bar{1} = 0$$

2.4 二进制运算



□ 二进制数的逻辑运算

◆ 异或运算

异或运算表示符号 $\oplus$ 。

运算法则：

$$0 \oplus 0 = 0$$

$$1 \oplus 1 = 0$$

$$0 \oplus 1 = 1$$

$$1 \oplus 0 = 1$$

结论：参加运算的两个逻辑变量**相同**时，异或运算的结果等于**0**。

当两个逻辑变量**不相同**时，异或运算的结果为**1**。

2.5 数的定点与浮点表示



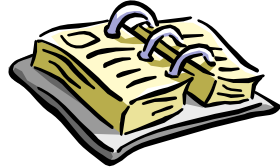
在计算机中，用二进制表示一个带小数点的数有两种方法，即**定点**表示和**浮点**表示。

相应地，计算机按数的表示方法不同也可以分为定点计算机和浮点计算机两大类。

所谓**定点**表示，就是小数点在数中的位置是**固定**的；

所谓**浮点**表示，就是小数点在数中的位置是**浮动**的。

2.5 数的定点与浮点表示



□ 定点表示 (Fixed Point Number)

定点数：小数点固定在数的某个位置，即**阶码是固定值**。
计算机中没有专门表示小数点的位，小数点的位置是**约定的**。

任意一个二进制数可表示为：纯小数或纯整数与一个2的整数次幂的乘积，即

$$N = 2^P \times S$$

- S – 数 N 的**尾数**，表示了数 N 的全部有效数字
- P – 数 N 的**阶码**，确定了小数点的位置
- 2 – 阶码的**底**

2.5 数的定点与浮点表示



□ 定点表示 (Fixed Point Number)

如假定 $P = 0$ ，且尾数 S 为纯小数时，这时定点数只能表示小数。

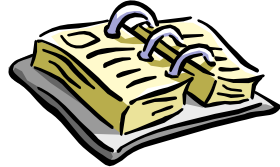


如假定 $P = 0$ ，且尾数 S 为纯整数时，这时定点数只能表示整数。



定点数的两种表示法，在计算机中均有采用。究竟采用哪种方法，均是事先约定的。如用纯小数进行计算时，其运算结果要用适当的比例因子来折算成真实值。

2.5 数的定点与浮点表示



□ 定点表示 (Fixed Point Number)

计算机中，数的正负是用**0 (正)** 和**1 (负)** 来表示。

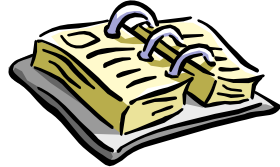
例如：8位二进制数，最左边第1位表示符号(称为符号位)，其余7位可用来表示尾数。

定点纯整数表示范围：

无符号时，00000000 ~ 11111111，即 0~255；

有符号时，-11111111 ~ +11111111，即 -127~+127

2.5 数的定点与浮点表示



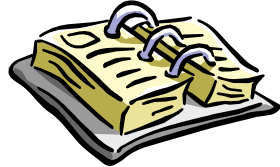
□ 定点表示 (Fixed Point Number)

定点纯小数表示范围：

$$\underbrace{2^{-n}}_{\substack{0.0000\dots01 \\ n-1\text{个}0}} \leq |N| \leq \underbrace{1-2^{-n}}_{\substack{0.1111\dots1 \\ n\text{位}}}$$

结论：定点数表示法简单直观，但是数值表示的范围太小，运算时容易产生溢出。

2.5 数的定点与浮点表示



□ 浮点表示 (Floating Point Number)

浮点数：小数点的位置可以变动的，即**阶码可以取不同的值**。

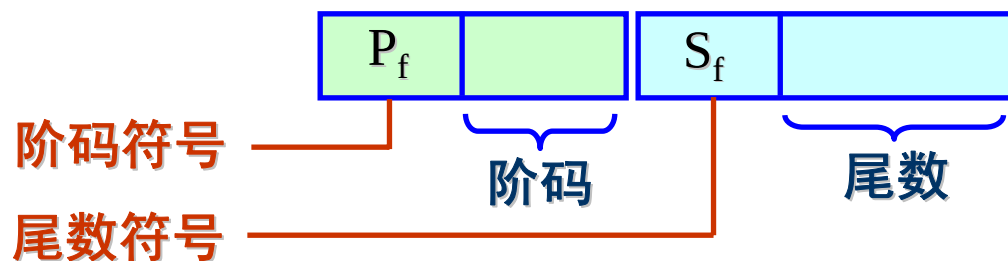
浮点表示法类似于十进制中的科学计数法。

计算机中表示一个浮点数，要分为**阶码和尾数**两个部分来表示。

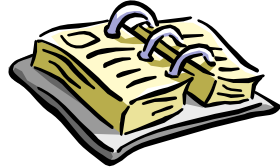
阶码P：二进制整数表示，可为正数和负数， P_f 表示阶码符号；

尾数S：二进制表示，可为正数和负数， S_f 表示尾数符号。

$$N = 2^P \times S$$



2.5 数的定点与浮点表示



□ 浮点表示 (Floating Point Number)

浮点数可以表示成多种形式：

$$0.110 \times 2^6 = 1.10 \times 2^5 = 0.000110 \times 2^9 \dots\dots$$

为了不丢失有效数字，提高运算精度，采用二进制**浮点规格化数**。

浮点规格化：尾数S的绝对值小于1而大于或等于1/2，
即**小数点后面的一位必须是1**。

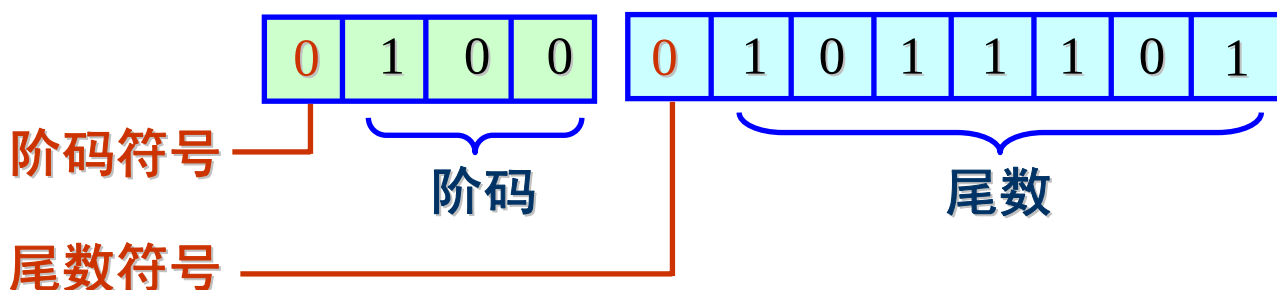
$$N = 2^P \times S \quad \Rightarrow \quad N = \pm 0.1bbb\dots bb \times 2^P$$

2.5 数的定点与浮点表示



□ 浮点表示 (Floating Point Number)

例：二进制数 $+1011.101$ ，可写成 $2^{+100} \times 0.1011101$ (相当于十进制数 11.625)，其浮点数表示为



浮点表示和定点表示相比，多了一个阶码部分。

浮点表示范围(m 位阶码, n 位尾数):

$$2^{-(2^m-1)} \cdot 2^{-n} \leq |N| \leq 2^{+(2^m-1)} \cdot (1-2^{-n})$$

阶码最小值 $2^{-(2^m-1)}$ 阶码最大值 $2^{+(2^m-1)}$



2.6 带符号数的表示法

□ 机器数与真值

符号数码化：将符号用“0正1负”表示，并以二进制数的最高位(D7位)作为符号位。

D7	D6	D5	D4	D3	D2	D1	D0
----	----	----	----	----	----	----	----

符号位

机器数：数据在计算机中连同数码化的符号位一起表示的编码数。

真值：把机器数实际代表的数称为机器数的真值。

机器数 { 无符号数：没有符号位
 { 有符号数： { 原码
 { 反码
 { 补码



2.6 带符号数的表示法

□ 机器数的种类和表示方法

◆ 原码

D7位作为符号位（0正1负），D6~D0位为原来的二进制数值位。

例1：正数 $X=+105$ 的原码表示：

$$[X]_{\text{原}} = \underset{\substack{\downarrow \\ \text{符号位}}}{0} \underbrace{1101001}_{\text{数值本身}}$$

例2：负数 $X=-105$ 的原码表示：

$$[X]_{\text{原}} = \underset{\substack{\downarrow \\ \text{符号位}}}{1} \underbrace{1101001}_{\text{数值本身}}$$



2.6 带符号数的表示法

□ 机器数的种类和表示方法

◆ 原码

例3: 0 的原码表示为

$$[+0]_{\text{原}} = 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0$$

$$[-0]_{\text{原}} = 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0$$

8位二进制，原码表示范围为： $+(127)D \sim -(127)D$

结论：原码表示简单易懂，与真值的转换很方便。但在计算机中进行加减运算时比较麻烦。

◆ 反码

例： $[+0]_{\text{反}} = \underset{\downarrow}{\textcolor{brown}{0}} \underbrace{000000}_{\text{数值本身}}$

$[+4]_{\text{反}} = 0 \quad 0000100$
 ↓
 符号位 数值本身

$[+127]_{\text{反}} = 0 \underbrace{1111111}_{\text{数值本身}}$
 ↓
 符号位



2.6 带符号数的表示法

□ 机器数的种类和表示方法

◆ 反码

负数的反码：将它的正数按位(包括符号位)取反形成的。

例： $[-0]_{\text{反}} = \underset{\substack{\downarrow \\ \text{符号位}}}{1} \underbrace{1\ 1\ 1\ 1\ 1\ 1\ 1}_{\text{数值本身}}$

$[-4]_{\text{反}} = \underset{\substack{\downarrow \\ \text{符号位}}}{1} \underbrace{1\ 1\ 1\ 1\ 0\ 1\ 1}_{\text{数值本身}}$

$[-127]_{\text{反}} = \underset{\substack{\downarrow \\ \text{符号位}}}{1} \underbrace{0\ 0\ 0\ 0\ 0\ 0\ 0}_{\text{数值本身}}$



2.6 带符号数的表示法

□ 机器数的种类和表示方法

◆ 反码结论

- “0”的反码有两种表示法：
00000000表示“+0”，11111111表示“-0”。
- 8位二进制反码的数值范围：
 $+(127)D \sim -(127)D$
- 一个带符号数用反码表示时，最高位为符号位。

2.6 带符号数的表示法



□ 机器数的种类和表示方法

◆ 补码（微机中采用补码表示法）

同一加法电路即可实现有符号数的相加，也可实现无符号数的相加，且可通过加法来实现减法运算；

简化逻辑运算，提高速度，降低成本。

2.6 带符号数的表示法



□ 机器数的种类和表示方法

例：假设目前正确时间为6点整，有一钟表停在10点整，如何校准？

逆时针拨4格，即 $10 - 4 = 6$

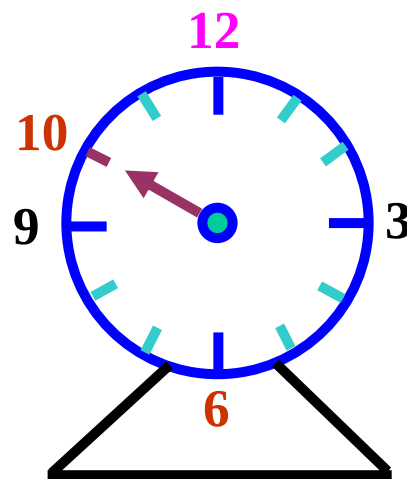
顺时针拨8格，到12点后从0开始重新计时，

即 $10 + 8 = 12(\text{自动丢失}) + 6 = 6$

模：循环计数系统中所表示的最大数

(-4) 与 $(+8)$ 对模12互为补数，同余数

结论：对于某一确定的模(12)，某数(10)减去绝对值小于模的另一个数(4)，总可以用某数(10)加上“另一数的负数(-4)与其模(12)之和(8)”(即补数)来代替。





2.6 带符号数的表示法

□ 机器数的种类和表示方法

◆ 补码（微机中采用补码表示法）

字长为8位的二进制数制中，模为 $2^8=256\text{ D}$ 。

例： $64-10 = \underline{64+(-10)} = 64+[256-10] = 64+246$
 $\quad\quad\quad = \underline{256+54} = 54$

0100 0000	64	0100 0000	64
- 0000 1010	-10	+ 1111 0110	+246
<u> </u>		<u> </u>	
0100 0000	54	1 0011 0110	54

自动丢失

- 结论：
- ① (-10) 与 $(+246)$ 对模256互为补数，同余数；
 - ② $246\text{ D} = 1111\ 0110\text{ B}$ 就是 (-10) 的补码表示；
 - ③ 负数表示为它的补码，减法转换为加法。



2.6 带符号数的表示法

□ 机器数的种类和表示方法

◆ 补码（微机中采用补码表示法）

正数的补码：与其原码相同，即符号位用“0”表示正，数字位为数值本身。

例： $[+0]_{\text{补}} = \underset{\substack{\downarrow \\ \text{符号位}}}{0} \underbrace{0000000}_{\text{数值本身}}$

$[+4]_{\text{补}} = \underset{\substack{\downarrow \\ \text{符号位}}}{0} \underbrace{0000100}_{\text{数值本身}}$

$[+127]_{\text{补}} = \underset{\substack{\downarrow \\ \text{符号位}}}{0} \underbrace{1111111}_{\text{数值本身}}$



2.6 带符号数的表示法

□ 机器数的种类和表示方法

◆ 补码（微机中采用补码表示法）

负数的补码：反码加1。连同符号位，按位取反再加1。

例： $[-4]_{\text{补}} = \underset{\substack{\downarrow \\ \text{符号位}}}{1} \underbrace{1111100}_{\text{数值本身}}$

$[-31]_{\text{补}} = \underset{\substack{\downarrow \\ \text{符号位}}}{1} \underbrace{1100001}_{\text{数值本身}}$

$[-127]_{\text{补}} = \underset{\substack{\downarrow \\ \text{符号位}}}{1} \underbrace{0000001}_{\text{数值本身}}$



2.6 带符号数的表示法

□ 机器数的种类和表示方法

◆ 补码结论

① $[+0]_{\text{补}} = [-0]_{\text{补}} = 00000000$

② 8位二进制补码所能表示的数值为 $-128 \sim +127$
最小的负数 $10000000\text{B} (-128\text{D})$

③ 当1个带符号数用8位二进制补码表示时，最高位为符号位，其余7位为数字位。

例： $[X]_{\text{补}} = 1\ 0011011\ \text{B}$ 表示一个负数

将 0011011 按位取反后，再加1，得到 1100101

$X = -1100101 = -101\ \text{D}$



2.6 带符号数的表示法

□ 补码的加法运算

符号位与数字位一起参加运算，运算结果也是补码；

$$\begin{aligned}[X]_{\text{补}} + [Y]_{\text{补}} &= (2^n + X) + (2^n + Y) \\ &= 2^n + (X + Y) \\ &= [X + Y]_{\text{补}}\end{aligned}$$

结论：两数补码之和，等于两数和的补码。



2.6 带符号数的表示法

□ 补码的加法运算

例1: $X = +10000000$, $Y = +0001000$, 求两数的补码之和。

解: $[X]_{\text{补}} = 01000000$, $[Y]_{\text{补}} = 00001000$

$[X]_{\text{补}} = 01000000$	$+64$
$+) \quad [Y]_{\text{补}} = 00001000$	$+) \quad + 8$
$[X]_{\text{补}} + [Y]_{\text{补}} = 01001000$	$+ 72$

结论: 两数和为正, 正数的补码等于原码, 即

$$[X+Y]_{\text{补}} = [X+Y]_{\text{原}} = 01001000, \text{真值为} +72$$



2.6 带符号数的表示法

□ 补码的加法运算

例2: $X = +0000111$, $Y = -0010011$, 求两数的补码之和。

解: $[X]_{\text{补}} = 00000111$, $[Y]_{\text{补}} = 11101101$

$[X]_{\text{补}} = 00000111$	$+ 7$
$+) [Y]_{\text{补}} = 11101101$	$+) - 19$
$[X]_{\text{补}} + [Y]_{\text{补}} = 11110100$	$- 12$

结论: 两数和为负, 将负数的补码还原为原码, 即

$$[X+Y]_{\text{原}} = [(X+Y)_{\text{补}}]_{\text{补}} = 10001100, \text{ 真值为 } -12$$



2.6 带符号数的表示法

□ 补码的加法运算

例3: $X = -0011001$, $Y = -0000110$, 求两数的补码之和。

解: $[X]_{\text{补}} = 11100111$, $[Y]_{\text{补}} = 11111010$

自动 丢失	$[X]_{\text{补}} =$	11100111	- 25
	$+) [Y]_{\text{补}} =$	11111010	$+) - 6$
	$[X]_{\text{补}} + [Y]_{\text{补}} =$	1 11100001	- 31

结论: 两数和为负, 将负数的补码还原为原码, 即

$$[X+Y]_{\text{原}} = [(X+Y)_{\text{补}}]_{\text{补}} = 10011111, \text{ 真值为 } -31$$



2.6 带符号数的表示法

□ 补码的减法运算

两数补码之差，等于两数差的补码。

$$\begin{aligned}[X]_{\text{补}} + [Y]_{\text{补}} &= [X]_{\text{补}} + [-Y]_{\text{补}} \\ &= (2^n + X) + [2^n + (-Y)] \\ &= 2^n + (X - Y) \\ &= [X - Y]_{\text{补}}\end{aligned}$$



2.6 带符号数的表示法

□ 补码的减法运算

例1: $X = +10000000$, $Y = +0001000$, 求两数的补码之差。

解: $[X]_{\text{补}} = 01000000$, $[-Y]_{\text{补}} = 11111000$

	$[X]_{\text{补}} =$	01000000	+64
自动 丢失	$+)$	$[-Y]_{\text{补}} =$	11111000
		$[X]_{\text{补}} + [-Y]_{\text{补}} =$	1 00111000
			+ 56

结论: 两数差为正, 正数的补码等于原码, 即

$$[X-Y]_{\text{补}} = [X-Y]_{\text{原}} = 00111000, \text{真值为} +56$$



2.6 带符号数的表示法

□ 补码的减法运算

例2: $X = +0000111$, $Y = -0010011$, 求两数的补码之差。

解: $[X]_{\text{补}} = 00000111$, $[-Y]_{\text{补}} = 00010011$

$[X]_{\text{补}} = 00000111$	$+ 7$
$+) \quad [-Y]_{\text{补}} = 00010011$	$-) \quad - 19$
$[X]_{\text{补}} + [-Y]_{\text{补}} = 00011010$	$+ 26$

结论: 两数差为正, 正数的补码为原码, 即

$$[X-Y]_{\text{原}} = [X-Y]_{\text{补}} = 00011010, \text{真值为} +26$$



2.6 带符号数的表示法

□ 补码的减法运算

例3: $X = -0011001$, $Y = -0000110$, 求两数的补码之差。

解: $[X]_{\text{补}} = 11100111$, $[-Y]_{\text{补}} = 00000110$

	$[X]_{\text{补}} = 11100111$	- 25
+) $[-Y]_{\text{补}} = 00000110$		-) - 6
	$[X]_{\text{补}} + [-Y]_{\text{补}} = 11101101$	- 19

结论: 两数差为负, 将负数的补码还原为原码, 即

$$[X-Y]_{\text{原}} = [(X-Y)_{\text{补}}]_{\text{补}} = 10010011, \text{ 真值为 } -19$$



2.6 带符号数的表示法

□ 溢出及其判断方法

◆ 溢出：指带符号数的补码运算溢出。

带符号数字长为 n ，最高位表示符号，其余 $n-1$ 位表示数值，补码运算范围为 $-2^{n-1} \sim +2^{n-1}-1$ ，如果运算结果超出此范围，称为补码溢出(简称溢出)。溢出时，将造成运算错误。

字长 $n=8$ ，二进制数补码运算范围为：

$-2^{8-1} \sim +2^{8-1}-1$ ，即 $-128 \sim +127$ ，

如果运算结果超出此范围，则产生溢出。

字长 $n=16$ ，补码运算范围为：

$-2^{16-1} \sim +2^{16-1}-1$ ，即 $-32768 \sim +32767$ ，

如果运算结果超出此范围，则产生溢出。



2.6 带符号数的表示法

□ 溢出及其判断方法

例1: $X = +1000000$, $Y = +1000001$, 求两数的补码之和。

解: $[X]_{\text{补}} = 01000000$, $[Y]_{\text{补}} = 01000001$

$$\begin{array}{r} [X]_{\text{补}} = 01000000 \\ +) [Y]_{\text{补}} = 01000001 \\ \hline [X]_{\text{补}} + [Y]_{\text{补}} = 10000001 \end{array} \qquad \begin{array}{r} + 64 \\ +) + 65 \\ \hline + 129 \end{array}$$

$$[X+Y]_{\text{补}} = 10000001, \quad X+Y = -1111111 (-127)$$

结论: 两个正数相加, 结果应该为正数+129, 但运算结果却为负数-127, 发生错误的原因是运算时产生了溢出。



2.6 带符号数的表示法

□ 溢出及其判断方法

例2: $X = -1111111$, $Y = -0000010$, 求两数的补码之和。

解: $[X]_{\text{补}} = 10000001$, $[Y]_{\text{补}} = 11111110$

	$[X]_{\text{补}} =$	10000001		-127
自动 丢失	+) $[Y]_{\text{补}} =$	11111110		+) - 2
	$[X]_{\text{补}} + [Y]_{\text{补}} =$	1 01111111		- 129

$$[X+Y]_{\text{补}} = 01111111, \quad X+Y = +1111111 (+127)$$

结论: 两个负数相加, 结果应该为负数-129, 但运算结果却为负数+127, 发生错误的原因是运算时产生了溢出。



2.6 带符号数的表示法

□ 溢出及其判断方法

◆ 判断溢出的方法

- ① 根据参加运算的两个数的符号及运算结果的符号判断；
- ② 利用双进位状态来判断，即符号位相加的进位状态 $\oplus$ 数值部分的最高位相加的进位状态。

$$V = D_{7C} \oplus D_{6C}$$

$V=1$ ，表示有溢出； $V=0$ ，表示无溢出

2.6 带符号数的表示法



□ 溢出及其判断方法

◆ **溢出与进位**：不同性质的概念，二者之间无必然联系。

进位：运算结果的最高位向更高位的进位。

如有进位： $D_{7C}=1$

若 $D_{6C}=1$ ，则 $V=D_{7C} \oplus D_{6C}=1 \oplus 1=0$ ，表示**无**溢出；

若 $D_{6C}=0$ ，则 $V=D_{7C} \oplus D_{6C}=1 \oplus 0=1$ ，表示**有**溢出；

如无进位： $D_{7C}=0$

若 $D_{6C}=1$ ，则 $V=D_{7C} \oplus D_{6C}=0 \oplus 1=1$ ，表示**有**溢出；

若 $D_{6C}=0$ ，则 $V=D_{7C} \oplus D_{6C}=0 \oplus 0=0$ ，表示**无**溢出；



第二章结束！