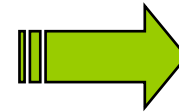


IBM—PC汇编语言 程序设计(第3版)

计算机语言的发展

机器语言

高级语言



FORTRAN

BASIC

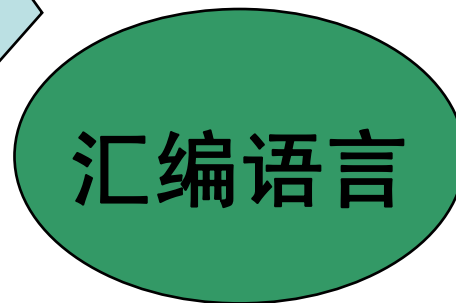
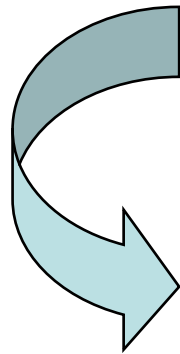
COBOL

PASCAL

C/C++

JAVA

...



汇编语言的特点

- 面向机器的低级语言，通常是为特定的计算机或计算机系列专门设计的。
- 保持了机器语言的优点，具有直接和简捷的特点。
- 可有效地访问、控制计算机的各种硬件设备，如磁盘、存储器、CPU、I/O端口等。
- 目标代码简短，占用内存少，执行速度快，是高效的程序设计语言。
- 经常与高级语言配合使用，应用十分广泛。

编程实现 $c = a + b$ ，并在屏幕上显示出结果。

例1

```
#include "stdafx.h"
#include "stdio.h"
int main(int argc, char* argv[])
{   int a,b,c;
    a=1;
    b=2;
    c=a+b;
    printf("c=%d\n",c);
    return 0;
}
```

编译后的目标文件达到3.59KB

例 2. $c = a + b$

```
a      db      ?  
b      db      ?  
c      db      ?  
string db      'c=$'
```

```
main    proc far  
        assume cs:code,  
ds:data, es:data
```

```
        push    ds  
        sub     ax, ax  
        push    ax  
        mov     ax, data  
        mov     ds, ax  
        mov     es, ax
```

```
mov     a, 1  
mov     b, 2  
mov     al, a  
add     al, b  
mov     c, al  
lea     dx, string  
mov     ah, 09  
int     21h  
add     c, 30h  
mov     dl, c  
mov     ah, 2  
int     21h  
mov     dl, 0ah  
int     21h  
mov     dl, 0dh  
int     21h  
ret  
main    endp
```

汇编后的目标文件只有208字节

汇编语言的应用

系统程序、效率代码、I/O驱动程序

- 70%以上的系统软件是用汇编语言编写的。
- 某些快速处理、位处理、访问硬件设备等
高效程序是用汇编语言编写的。
- 高级绘图程序、视频游戏程序一般是用汇编语言编写的。

学习建议

目标：

- 了解汇编语言的特性及其编程技术，
- 建立起“机器”和“程序”、“空间”和“时间”的概念。

建议：

- 充分注意“汇编”课实践性强的特点，
- 多读程序，多写程序，多上机调试程序，
- 熟悉PC机的编程结构，
- 掌握汇编语言及其程序设计的基本概念、方法和技巧。

参考资料

1. 80X86汇编语言程序设计

沈美明 温冬婵 清华大学出版社

2. IBM PC 汇编语言与程序设计

PETER ABEL 编著

第4版 清华大学出版社. PRENTICE HALL

第5版 人民邮电出版社

3. IBM PC 汇编语言与程序设计例题习题集

温冬婵 沈美明 清华大学出版社

教学日

课程名称：汇编语言程序设计				
所在班级： 计算机		考试周次： 17周		
周次	讲课内容	时数	实验	时数
1	第一章 概论	2		
1	第二章 Inter 80x86计算机组织	3		
2	第三章 指令系统及寻址方式	8		
5	第四章 汇编语言程序上机	2	熟悉TC, MASM, LINK, DEBUG等工具	2
7	第五章 直接程序设计	3		
8	第六章 分支程序设计	3	分支程序设计	2
10	第七章 循环程序设计	6	循环程序设计	2
13	第八章 子程序设计及系统调用	6	子程序设计	2
15	第九章 宏技术	3		
16	第十章 输入输出程序设计	4		

第1章 基础知识

- 数制
- 数制之间的转换
- 运算
- 数和字符的表示

预 备 知 识

存储容量

1KB = 1024B = 2^{10} B (KiloByte、千字节)

1MB = 1024KB = 2^{20} B (MegaBate、兆字节)

1GB = 1024MB = 2^{30} B (GigaByte、千兆字节)

1TB = 1024GB = 2^{40} B (TeraByte、兆兆字节)

1个二进制位: **bit** (位、比特)

8个二进制位: **Byte** (字节)

1Byte = 8bit

2个字节: **Word** (字)

1Word = 2Byte = 16bit

1. 数 制

数 制	基 数	数 码
二进制 B inary	2	0, 1
八进制 O ctal	8	0, 1, 2, 3, 4, 5, 6, 7
十进制 D ecimal	10	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
十六进制 H exadecimal	16	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

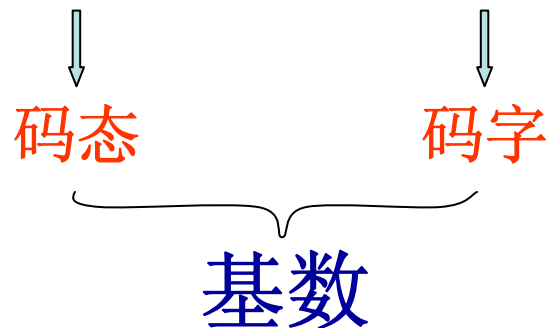
例如：

177D, **101100110B**, **0074H**

进位数制基本原理

1、十进制：对日常常用的进位计数制十进制，总结有：

1) 每个计数位上最多有0, 1, 2, 3, 4, 5, 6, 7, 8, 9这十种基本状态
(注：应理解为有十种基本状态，以0~9来表示)



2) 进位计数过程中，满“十”向临高位进“一”；该数量比最大的基数码态多一个数量单位，表示为 **10**，称 (**10**) 态为**满进制数**。

3) 相同的基数码态在不同的数位上所代表的量值是不一样的。如 **1000**，**100**中的1，分别代表了有1个“一 千”和1个“一百”。以**位权**作为其**量值大小**的表达方式

故而，从上述总结可得一个R进制计数制，其计数特点或原理有：

1. **基数**：基本态有几个？表示方式是什么？

如前十进制中基态有十个，按照由小到大顺序分别是...

每一种进制都允许使用固定个数的数码

2. **逢多少进位**：满进制数是如何产生的？表达方式是什么

如前十进制数中满进制数为 (10) D

逢R进一

3. **位权**：以各位作为计数点，各个位如下表示

$a_n, a_{n-1} \dots a_3, a_2, a_1, a_0 . b_1, b_2, b_3 \dots$

则 a_i 的位权是 10^i ，

即满进制数的i次幂。

采用位权表示法

常用进位计数制

日常生活中的（自然）计数法是十进制，为人们熟练使用和理解，但是由于该计数法中有十个基数态，目前自然物质中无法简单实现其表示，不适合计算机内部的数据信息表示，从而产生了一种适合计算机的计数法则----二进制

1、十进制（**Dec**）：

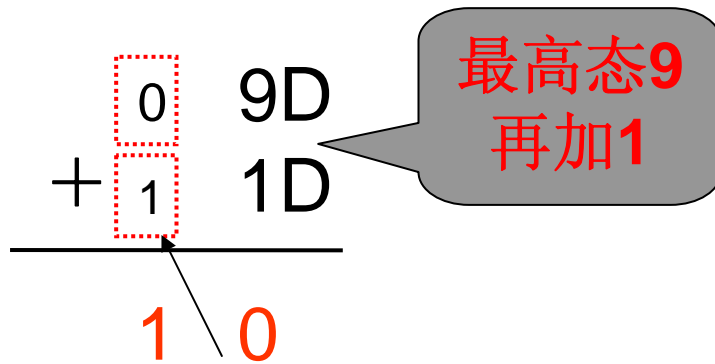
- a) 基数：0,1,2,3,4,5,6,7,8,9
- b) 逢“十”进一：“十进制”表示为：**D**
- c) 位权： 10^k

2、二进制（**Bin**）：

- a) 基数：0,1
- b) 逢“二”进一：“二进制”表示为：**B**
- c) 位权： 2^k

常用进位计数制

进位法则：计数过程中，当本位为最高稳定状态再加上一个计量单位（1）时，本位恢复到最低态，同时向临高位产生进位1进行相加。例：



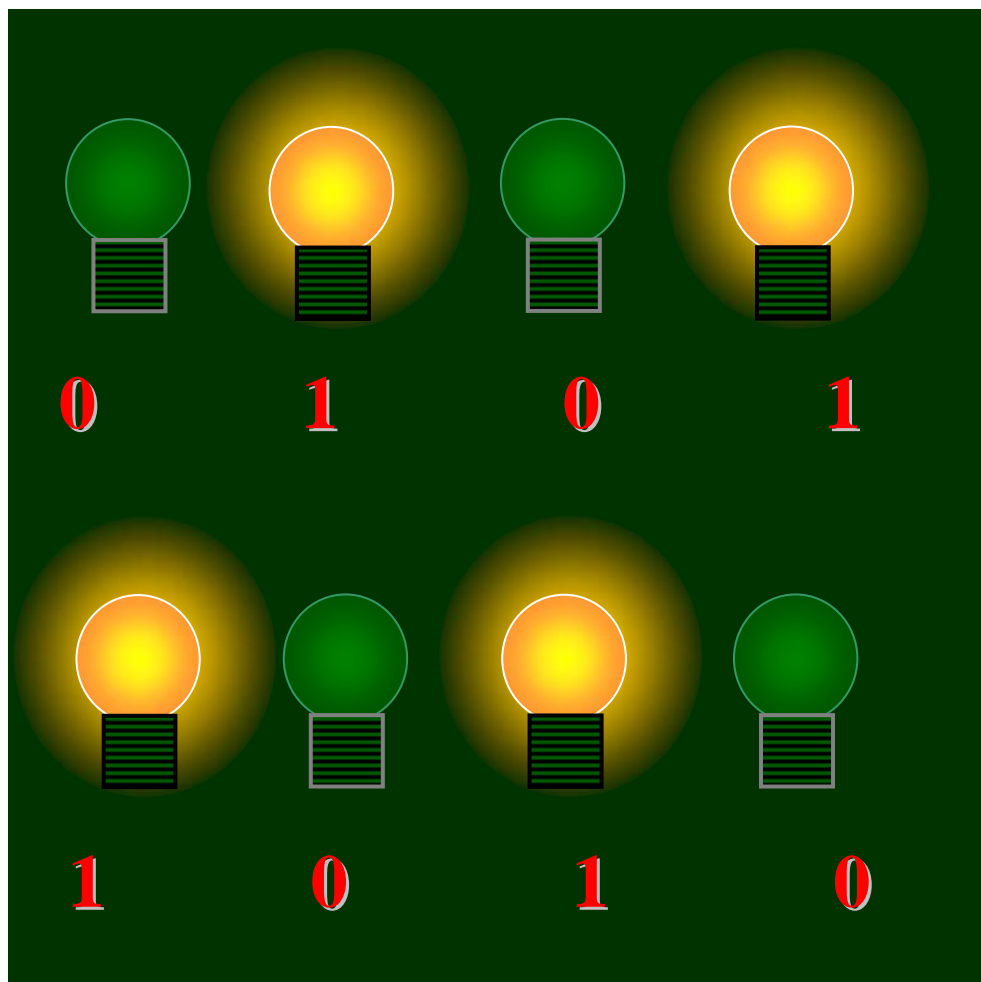
Dec	Bin
0 +1	0 +1
1 +1	1 +1
⋮ +1	10 +1
9 +1	11 +1
10 +1	100
⋮	
99 +1	
100	

计算机为什么使用二进制

♣ 二进制在物理上容易实现

♣ 二进制运算规则简单

用一排灯表示一个二进制数，明为数码“1”，
暗为数码“0”



= 十进制的“5”

= 十进制的“10”

由于二进制数相对与十进制数其对同一个数量值表示时所需的位数要多很多，书写长度大，为了便于交流同时体现计算机内部信息的存储形式，派生了八进制和十六进制两种书面计数制。

3、八进制（**Oct**）：

a) 基数：0,1,2,3,4,5,6,7

b) 逢“八”进一：“八进制”表示为：**O**

c) 位权： 8^k

4、十六进制（**Hex**）：

a) 基数：0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F

b) 逢“十六”进一：“十六进制”表示为：**H**

c) 位权： 16^k

不同数制间数的互相转换

- 任意进制数转化为十进制
- 十进制数转化为任意进制数
- 二进制、八进制、十六进制数之间的相互转化

任意进制数转化为十进制

任意进制数转化为十进制数只要写出该进制数的按权展开式，进行乘法和加法运算，得出结果即可。

$$(101101)_2 = 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 \\ = 32 + 0 + 8 + 4 + 0 + 1 = (45)_{10}$$

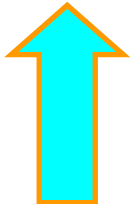
$$(710)_8 = 7 \times 8^2 + 1 \times 8^1 + 0 \times 8^0 = 448 + 8 + 0 = (456)_{10}$$

$$(1B6)_{16} = 1 \times 16^2 + 11 \times 16^1 + 6 \times 16^0 \\ = 256 + 176 + 6 = (438)_{10}$$

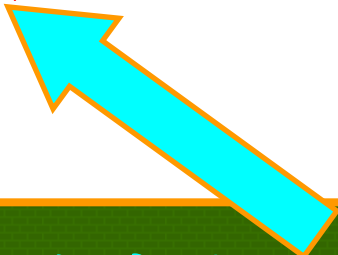
十进制数转化为任意进制数

将十进制数转化为任意进制数需对

整数部分和小数部分分别进行转化



采用“除基数取余法”，即用基数多次除被转换的十进制数，直到商为0，每次相除所得的余数，按逆序排列便是对应的进制数



小数部分的转换采用“乘基数取整法”，即用基数多次乘被转换的十进制数的小数部分，每次相乘后，所得乘积的整数部分按正序排列就是对应的二进制数

十进制化成二进制举例： $(185)_{10} = (?)_2$

$$(185)_{10} = (?)_2$$

2	185	余数
2	92	1
2	46	0
2	23	0
2	11	1
2	5	1
2	2	1
2	1	0
	0	1


$$(185)_{10} = (10111001)_2$$

十进制化成二进制举例

$$(0.8125)_{10} = (\quad?)_2$$

$$(0.8125)_{10} = (0.1101)_2$$


	$\times \quad \frac{0.8125}{2}$	整数
	<hr/>	
	1.6250 ...	1
	$\times \quad \frac{0.6250}{2}$	
	<hr/>	
	1.2500...	1
	$\times \quad \frac{0.2500}{2}$	
	<hr/>	
	0.5000...	0
	$\times \quad \frac{0.5000}{2}$	
	<hr/>	
	1.0000...	1



十进制化成八进制举例

$$(185)_{10} = (\quad ? \quad)_8$$

8		185	余数
8		231
8		27
		02

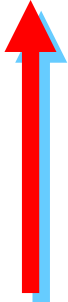


$$(185)_{10} = (271)_8$$

十进制化成十六进制举例

$$(3981)_{10} = (?)_{16}$$

16		3 9 8 1	余数
16		2 4 8	13 (D)
16		1 5	8
		0	15 (F)



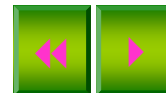
$$(3981)_{10} = (\mathbf{F8D})_{16}$$

二进制、八进制、十六进制数之间的相互转化

- ➡ 二进制数转化为八进制数
- ➡ 八进制数转化为二进制数
- ➡ 二进制数转化为十六进制数
- ➡ 十六进制数转化为二进制数

二进制数转化为八进制数

二进制数转换成八进制数，概括为“三位合一”、即：以小数点为基准，整数部分从右至左，小数部分从左至右，每三位一组，不足三位时，整数部分在高端补齐，小数部分在低端补齐。然后，把每一组二进制数用一位相应的八进制数表示，小数点位置不变，即得到八进制数



举例

$(1\ 011\ 010\ 101\ 110)_2$

1 3 2 5 6

为八进制的13256

$(11\ 011\ 111.011\ 100)_2$

3 3 7 . 3 4

为八进制的337.34



八进制数转化为二进制数

八进制数转换成二进制数，概括为“一位拆三位”，即把一位八进制数写成对应的三位二进制数，然后按权连接即可

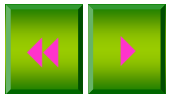
例如：

$(5\ 4\ 2\ 7\ 0)_8$ $(1\ 6\ 3\ .\ 6\ 4)_8$
 $101\ 100\ 010\ 111\ 000$ $001\ 110\ 011\ .\ 110\ 100$



二进制数转化为十六进制数

二进制数转换成十六进制数，概括为“四位合一”、即：以小数点为基准，整数部分从右至左，小数部分从左至右，每四位一组，不足四位时，整数部分在高端补齐，小数部分在低端补齐。然后，把每一组二进制数用一位相应的十六进制数表示，小数点位置不变，即得到十六进制数



举例

$(1\ 0110\ 1010\ 1110)_2$

1 6 A E

为十六进制的16AE

$(110\ 1001\ 1111.0111\ 1000)_2$

6 9 F . 7 8

为十六进制的69F.78



十六进制数转化为二进制数

十六进制数转换成二进制数，概括为“一位拆四位”，即把一位十六进制数写成对应的四位二进制数，然后按权连接即可

例如：

(B		4		F		7)	₁₆	(C		2		.		A		8)	₁₆
1011		0100		1111		0111		1100		0010		.		1001		1000					



- 二进制 \longleftrightarrow 十六进制

→ 0011 0101 1011 1111

↓ ↓ ↓ ↓

3 5 B F

∴ 0011,0101,1011,1111B = 35BFH

← A 1 9 C

↓ ↓ ↓ ↓

1010 0001 1001 1100

∴ A19CH = 1010,0001,1001,1100B

- 二进制 \rightleftarrows 十进制

$$1011\text{B} = 2^3 + 2^1 + 2^0 = 11\text{D}$$

降幂法

例: $27\text{D} = ? \text{B}$

32 16 8 4 2 1

能减记1，不能减记0

$$27 - 16 = 11 \quad (\text{能减, 记1})$$

$$11 - 8 = 3 \quad (\text{能减, 记1})$$

$$3 - 4 \quad (\times \text{ 枪毙, 记0})$$

$$3 - 2 = 1 \quad (\text{能减, 记1})$$

$$1 - 1 = 0 \quad (\text{能减, 记1})$$

(为0, 结束)

$$\therefore 27\text{D} = 11011\text{B}$$



转换秘籍: “8421” 葵花宝典

仅口述

二进制转换为十进制:

(1)1010

(2)0101

二进制转换为十六进制:

(1)1010

(2)11 1011 1100

十六进制转换为二进制:

(1)1234

(2)F9B5

3. 运算（算术运算和逻辑运算）

- 算术运算

二进制数：逢二进一 借一为二

加法规则

$$0 + 0 = 0$$

$$0 + 1 = 1$$

$$1 + 0 = 1$$

$$1 + 1 = 0 \text{ (进位1)}$$

乘法规则

$$0 \times 0 = 0$$

$$0 \times 1 = 0$$

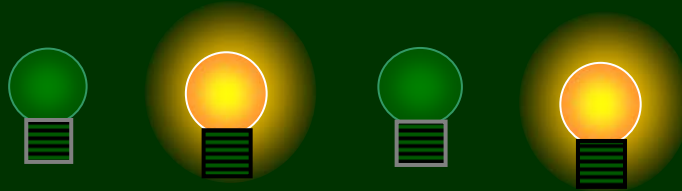
$$1 \times 0 = 0$$

$$1 \times 1 = 1$$

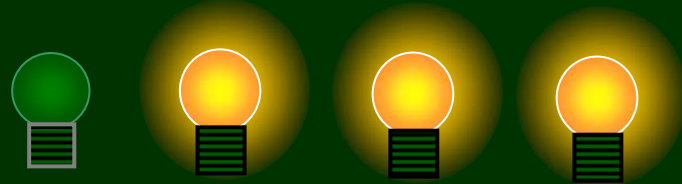
二进制加法运算的简单示例

求 $C = A + B = 5 + 7$

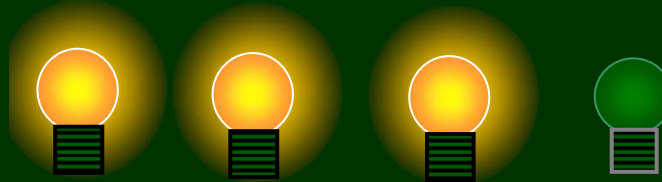
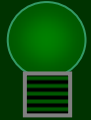
A:



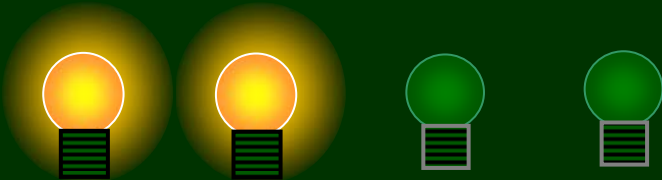
B:



进位
标志



C:



PageDown演示各位的计算

运算规则:

$$0+0=0$$

$$1+0=1$$

$$0+1=1$$

$$1+1=10$$

A的当前位 + B的当前位 + 进位标志 = 新进位标志和C的当前位

十六进制数：逢十六进一 借一为十六

	0	5	C	3	H		3	D ⁻¹	2	5	H	
+	3 ₁	D	2	5	H		-	0	5	C	3	H
<hr/>							<hr/>					
	4	2	E	8	H			3	7	6	2	H

- 逻辑运算（按位操作）

“与”运算（AND）

（二真才真）

A	B	$A \wedge B$
0	0	0
0	1	0
1	0	0
1	1	1

“或”运算（OR）

（一真即真）

A	B	$A \vee B$
0	0	0
0	1	1
1	0	1
1	1	1

“非”运算（NOT）

（假作真，真亦假）

A	$\neg A$
0	1
1	0

“异或”运算（XOR）

（相异为真）

A	B	$A \nabla B$
0	0	0
0	1	1
1	0	1
1	1	0

例: $X=00FFH$ $Y=5555H$, 求 $Z=X \vee Y = ?$

$X = 0000 \ 0000 \ 1111 \ 1111 \ B$

$\vee \ Y = 0101 \ 0101 \ 0101 \ 0101 \ B$

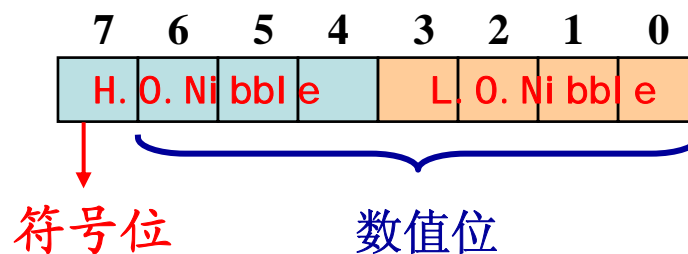
$Z = 0101 \ 0101 \ 1010 \ 1010 \ B$

$\therefore Z = 55AAH$

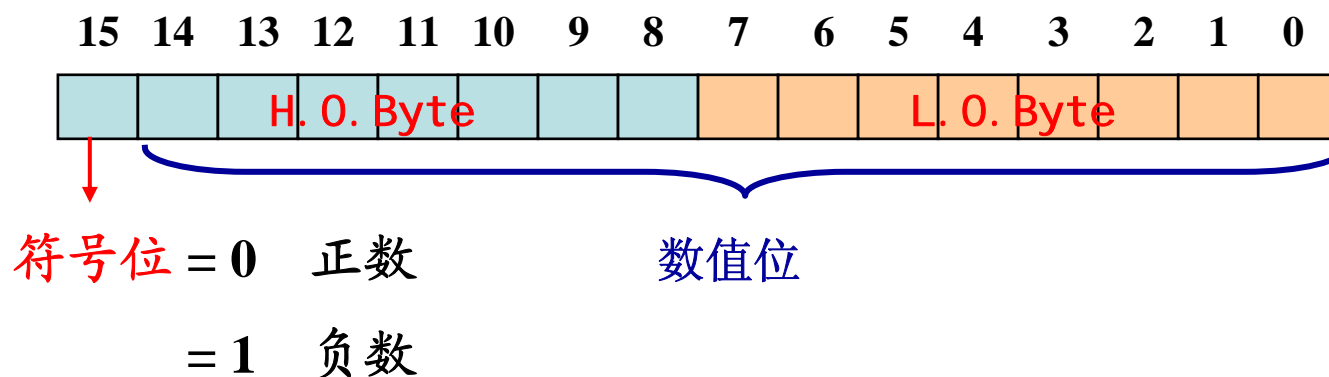
4. 数和字符的表示

- 计算机中正负数的表示

假设机器字长为8位：



假设机器字长为16位：



数的常用表示法 —— 原码 反码 补码

原码表示法: 符号 + 绝对值

例: $n = 8\text{bit}$

$$[+3]_{\text{原码}} = 0\ 000,0011 = 03\text{H}$$

$$[-3]_{\text{原码}} = 1\ 000,0011 = 83\text{H}$$

$$[+0]_{\text{原码}} = 0\ 000,0000 = 00\text{H}$$

$$[-0]_{\text{原码}} = 1\ 000,0000 = 80\text{H}$$

$\therefore 0$ 的表示不惟一

反码表示法: 正数的反码同原码, 负数的反码数值位与原码相反

例: $n = 8\text{bit}$

$$[+5]_{\text{反码}} = 0\ 000,0101 = 05\text{H}$$

$$[-5]_{\text{反码}} = 1\ 111,1010 = \text{FAH}$$

$$[+0]_{\text{反码}} = 0\ 000,0000 = 00\text{H}$$

$$[-0]_{\text{反码}} = 1\ 111,1111 = \text{FFH}$$

$\therefore 0$ 的表示不惟一

补码 (Two's Complement) 表示法:

正数的补码: 同原码

负数的补码: (1) 写出正数的补码
(2) 按位求反
(3) 末位加一

例: 机器字长8位, $[-46]_{\text{补码}} = ?$

$[46]_{\text{补码}} = 00101110$
按位求反
11010001
末位加一
11010010 = D2H

机器字长16位, $[-46]_{\text{补码}} = \text{FFD2H}$

$[+0]_{\text{补码}} = 00000000$
按位求反
11111111
末位加一
00000000 = $[-0]_{\text{补码}}$

$\therefore 0$ 的表示惟一

补码的符号扩展

- 8位扩展到16位，16位扩展到32位
- 正数的符号扩展，前面补0
- 负数的符号扩展，前面补1

$$[+46]_{\text{补}} = 0010\ 1110 \rightarrow 0000\ 0000\ 0010\ 1110 \\ = 002D\text{H}$$

$$[-46]_{\text{补}} = 1101\ 0010 \rightarrow 1111\ 1111\ 1101\ 0010 \\ = \text{FFD2H}$$

n位二进制补码的表数范围： $-2^{n-1} \leq N \leq 2^{n-1}-1$

十进制	二进制	十六进制	十进制	十六进制
n=8			n=16	
+127	0111 1111	7F	+32767	7FFF
+126	0111 1110	7E	+32766	7FFE
...
+2	0000 0010	02	+2	0002
+1	0000 0001	01	+1	0001
0	0000 0000	00	0	0000
-1	1111 1111	FF	-1	FFFF
-2	1111 1110	FE	-2	FFFE
...
-126	1000 0010	82	-32766	8002
-127	1000 0001	81	-32767	8001
-128	1000 0000	80	-32768	8000

无符号整数的表数范围： $0 \leq N \leq 2^n-1$

补码的加法和减法：

求补运算 \Rightarrow ： 对一个二进制数按位求反、末位加一

$$[X]_{\text{补码}} \Rightarrow [-X]_{\text{补码}} \Rightarrow [X]_{\text{补码}}$$

加法规则： $[X+Y]_{\text{补码}} = [X]_{\text{补码}} + [Y]_{\text{补码}}$

减法规则： $[X-Y]_{\text{补码}} = [X]_{\text{补码}} + [-Y]_{\text{补码}}$

补码减法可转换为补码加法

例：

$$\begin{array}{r} 64 \\ + (-46) \\ \hline 18 \end{array}$$

$$\begin{array}{r} 0100 \ 0000 \\ + 1101 \ 0010 \\ \hline 0001 \ 0010 \end{array}$$

原码、反码、补码的特点

- 如果 x 为**正**数，则有 $[x]_{\text{原}}=[x]_{\text{反}}=[x]_{\text{补}}$
- 如果 x 为负数，则 $[x]_{\text{原}}$ ， $[x]_{\text{反}}$ ， $[x]_{\text{补}}$ 有不同的表示
- 如果 x 为0，则 $[x]_{\text{补}}$ 有唯一编码， $[x]_{\text{原}}$ 和 $[x]_{\text{反}}$ 都有两个不同的编码
- $[x]_{\text{原}}$ 、 $[x]_{\text{反}}$ 、 $[x]_{\text{补}}$ 表示的机器数，最高一位表示符号位，**正**数用“0”表示，负数用“1”表示

- 进位和溢出

进位： 由于运算结果超出了位数,最高有效位向前的进位，这一位自然丢失，一般不表示结果的对错。

溢出： 表示结果超出了字长允许表示的范围，一般会造成结果出错。

例：

(-64)	1100 0000
+ 64	0100 0000
<hr/>	
0	1 0000 0000

进位

127	0111 1111
+ 1	0000 0001
<hr/>	
128	1000 0000

溢出

- 字符的表示

ASCII码: 用一个字节来表示一个字符，低7位为字符的**ASCII**值，最高位一般用作校验位。

例:

字符	ASCII值
‘A’	41H
‘a’	61H
‘1’	31H
换行	0AH
回车	0DH
空格	20H