计算机组成原理

Computer Organization

2023 · 秋

西安交通大学 计算机科学与技术学院

计算机组成原理课程组

http://corg.xjtu.edu.cn

第五章 数据表示与运算

数字化编码的基本原则

- 二要素
  - (1) 少量、简单的基本符号；
  - (2) 一定的组合规则。
- 用以表示大量、复杂、多样的信息

例如十进制数：基本符号0～9十个数码等，通过特定合规则表示出庞大的数值体系。

冯·诺依曼：电子元件的机器应采用二进制

- **电子数字计算机中，广泛采用二进制编码（基二码）来表示各种不同的信息。**

基二码（二进制编码）的特点

(1) 易于物理实现：两个基本符号1和0，个数最少，电子元件的**双稳态**和**开关**特性正好适用。

(2) 运算的简易性：编码、计数和算术运算的规则**最简单**，执行基本算术运算**最快**。

(3) 适合逻辑代数应用：与二值逻辑的“真”和“假”对应，有利于采用**逻辑代数工具**来分析、设计和简化机内逻辑线路。

(4) 很高的经济性：使用的**器材更少**，**更经济**。

计算机中常用的信息类型

计算机中，“数据”泛指所有需要进行加工处理的信息。

常见类型：**数值、逻辑数、字符、汉字、音频、视频等。在计算机内部，都必须用数字化编码的形式存储、加工和传送。**

音频、视频信号的表示通常归结为一种专门信息类别，称为多媒体。本课程不讨论。

计算机中常用信息综览

- 数值数据
  - 无符号数
  - 带符号数
    - 定点数
      - 整数
      - 小数
    - 浮点数
  - 十进制数
- 非数值数据
  - 逻辑数
  - 字符
  - 汉字

非数值数据的表示

逻辑数

- 二值：真 —— 1，假 —— 0；正好用二进制两个符号表示
- 特点：**按位操作**，位与位之间相互独立，无位权、无进位和借位等数值关系，可执行与、或、非等基本逻辑运算。
- 识别方法：通过不同的指令类型来进行识别
  - **逻辑指令**的操作数被默认为是**逻辑数**
  - **算术运算指令**的操作数则被默认为是**数值数据**
- 实际意义：控制系统中表示控制信号的**有(1)和无(0)**。

非数值数据的表示

字符的表示

字符——字母、数字、专用符号等西文信息，是人与计算机交互的重要媒介。

机内表示的关键：**二进制**编码，每个字符都有**唯一**的编码值，作为识别的依据。

❍字符交换码标准

ASCII码——美国标准信息交换码（American Standard Code for Information Interchange），当前国际通用。

- ✧ ASCII码字符集：**7位二进制编码表示一个特定的字符，总共定义了128种字符，**95种显示字符，33种控制字符。

ASCII码的机内表示

ASCII码在计算机中用一个**字节**来进行表示，ASCII码只占用了字节的低7位，字节的**最高位**取值需要确定。

处理方法：

(1) 恒为“0”，可利用这一位作为ASCII码**识别标志**

(2) 奇偶校验位，根据奇偶校验技术的需要取值(1/0)

(3) 扩展ASCII码方案，最高位用作字符编码(1/0)

- ASCII码位数扩展到8位
- 字符集扩大到256种字符

字符串的表示

字符串——连续一串字符组成的数据类型。

机内表示的关键问题：在主存中的存放方式。

向量表示法：将字符串存放在主存连续多个字节单元中，每个字节保存串中一个字符的ASCII码。

在同一个主存单元中的存放顺序，取决于不同计算机主存单元的编址顺序。

- ✧可按从低字节单元向高字节单元的顺序存放
  (适合小端方式)
- ✧也可按从高字节单元向低字节单元顺序存放
  (适合大端方式)

向量表示法

例：字符串：“if（A＜B）READ（C）”

从高到低字节单元的存放方案

HB LB

i	f		(
A	<	B	)
	R	E	A
D	(	C	)

32位

字符存放顺序示意

HB LB

69	66	20	28
41	3C	42	29
20	52	45	41
44	28	43	29

32位

机内存放顺序

（ASCII码用16进制表示）

向量表示法

字符串："if （A＜B） READ（C）"

HB LB

（		f	i
)	B	<	A
A	E	R	
)	C	(	D

32位

HB LB

28	20	66	69
29	42	3C	41
41	45	52	20
29	43	28	44

32位

从低到高字节单元的存放方案

向量表示法

表征参数：串首地址和串长（或结束符）。
作用：在存取时对字符串进行定位。

特点：

- 最简单、最节省存储空间的字符串存放方法。
- 在进行删除、插入操作时，对被操作字符后面的剩余字符串部分需要全部重新分配存储空间。字符串较长时，将花费较多的时间。

调用方法：指令系统中提供串操作类型的机器指令，如Intel 80X86指令系统中的串处理指令MOVS、CMPS、SCAS等等。

汉字的表示

为解决汉字**输入**、**表示**、**处理**、**输出**，有几种不同编码方案。

汉字输入码（汉字机外码、外码）

- 为直接使用西文标准键盘把汉字输入到计算机而设计
- 常用汉字输入码方案：四类

(1) 数字编码：用数字输入，无重码，转换方便，难记，不易推广。**电报码**、国标区位码等。

(2) 字音编码：**拼音输入法**。简单易学，重码率高，影响输入速度。微软拼音输入法、智能ABC输入法等。

(3) 字形编码：字形基本笔划输入，重码少、速度快，编码规则不易掌握。**五笔字型**输入法等。

(4) 形音编码：音、形结合，吸取(2)和(3)的优点，规则简化、重码减少，不易掌握。

汉字交换码标准

信息交换用汉字编码字符集:用于汉字处理、汉字通信等系统之间的信息交换。

- GB2312-80 标准，**基本集**

  1981 年颁布，共收入6763个常用汉字。称为国标码，国标交换码 。按使用频度分为:

  - **一级汉字3755个，常用字，按汉语拼音排序**
  - **二级汉字3008个，次常用字，按偏旁部首排序**
  - **常用的字母、数字和符号，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**

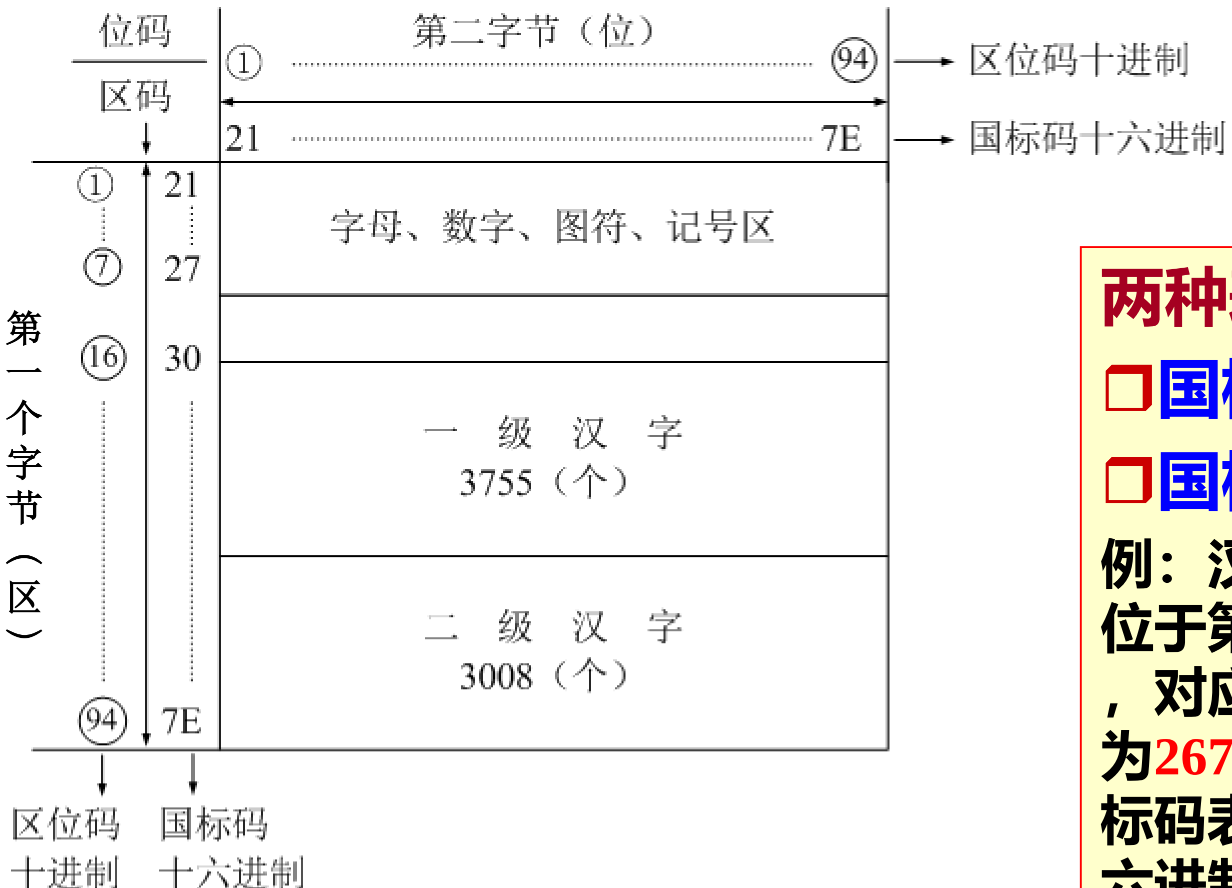
- **GB18030-2005，超大型中文编码字符集标准**

  共收录7万多个汉字，汉字为主，包含多种少数民族文字。

  - **藏、彝、蒙古、朝鲜、维吾尔文等**
  - **采用单字节、双字节、四字节三种方式编码，扩展了字符集容量**

汉字交换码标准（GB2312-80）

代码表



两种表示方式

- **国标码**
- **国标区位码**

例：汉字“红”位于第26区76位，对应的区位码为2676，而用国标码表示则为十六进制的3A6CH

汉字机内码（内码）

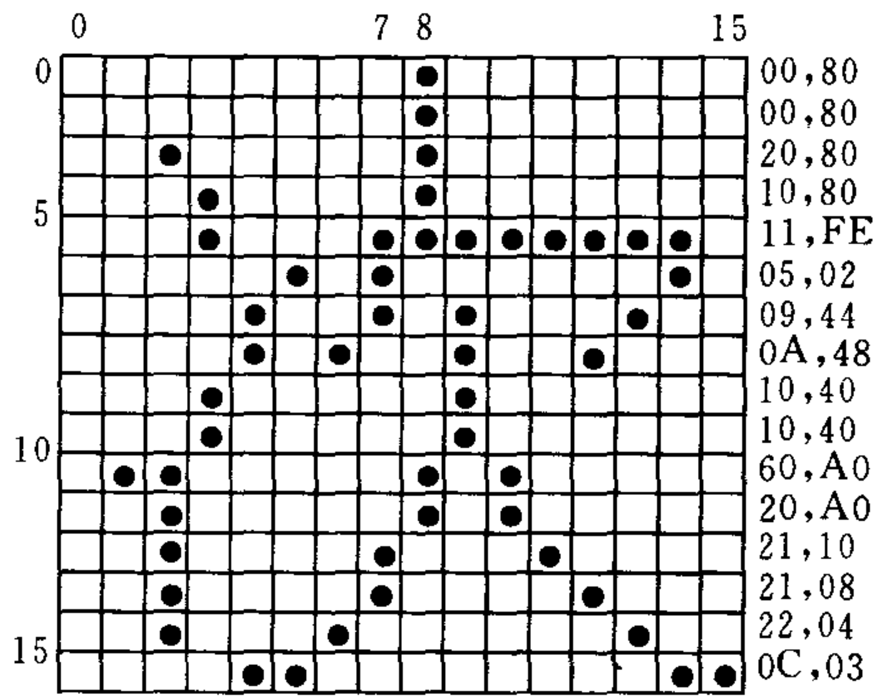
- 汉字在机内存储、处理、传送采用的编码。**常用国标码**
- 表示关键：**机内中西文信息常混合处理，汉字机内码需要特别标识，才能与ASCII机内码相区别。**
- 常用的汉字机内码设定方法如下：

(1) 若ASCII机内码最高位恒为“0”，可用两字节表示一个汉字，两字节最高位均为“1”，每个字节低7位为国标码，最多编码数量128×128。

(2) 若ASCII机内码最高位用于奇偶校验位或扩展ASCII码，计算机中需要三个字节表示汉字，其中第一个字节作为汉字的标识符使用。最多编码数量256×256。

汉字字模码

- ❑ 用于在输出设备上输出汉字而设计的字形编码
- ❑ 两种描述方法：点阵描述、轮廓描述
  - ❍ 点阵描述
    将汉字的字形用“点”组成的方阵表示，方阵中**有笔划的点位**点上黑点，用“**1**”表示
  - ❍ 点阵规模
    至少**16×16**，如果希望更好看，可用更大点阵



十进制数据的编码表示

- 用二进制编码表示**1位**十进制数，至少需**4位**
- **16**个编码状态选其中**10**个，多种编码方案，统称为二进制编码的十进制数（**Binary Coded Decimal**），简称**BCD**码或二—十进制编码。
- 分类
  - 有权码：四个二进制位均有指定的位权，**8421**码（**NBCD**码）
  - 无权码：二进制编码各位无指定的位权，余3码
  - 自补码：按位求反可得该数相对于**9**的补码，**2421**码
  - 循环码：相邻编码只有**1**位状态不同，格雷码

十进制有权码

十进制数	8421码	2421码	5211码	4311码
0	0000	0000	0000	0000
1	0001	0001	0001	0001
2	0010	0010	0011	0011
3	0011	0011	0101	0100
4	0100	0100	0111	1000
5	0101	1011	1000	0111
6	0110	1100	1010	1011
7	0111	1101	1100	1100
8	1000	1110	1110	1110
9	1001	1111	1111	1111

自补码

十进制无权码

十进制数	余3码	格雷码（1）	格雷码（2）
0	0011	0000	0000
1	0100	0001	0100
2	0101	0011	0110
3	**0110**	0010	0010
4	0111	0110	1010
5	1000	1110	1011
6	**1001**	1010	0011
7	1010	1000	0001
8	1011	1100	1001
9	1100	0100	1000

自补码

循环码

十进制数串的表示

- 主要有两种表示形式

  字符串形式

  - **将十进制数串看成一串字符串，常用于非数值计算领域**
  - **又可以分为两种形式**

    (1) 前分隔数字串：**同书写顺序，数符单独用一个字节。正号“+”的ASCII 码2BH，负号用“-”（2DH）。**

    (2) 后嵌入数字串：**数符不单独用一个字节，嵌入到最低一位数的ASCII码中。正数最低一位数的ASCII码不变；负数最低一位数的ASCII码高4位由0011变为0111。**

压缩的十进制数串形式

- 串中每位十进制数的ASCII码高4位压缩掉，只用其低4位，相当于每位十进制数用8421码表示，占用半个字节。
- 符号位：选用8421码不用的6种冗余编码表示，并放在最低数位之后。通常用1100(CH)表示正号，1101(DH)表示负号。
- **规定数值位加符号位之和必须为偶数，若串中总位数连同符号位为奇数时，在最高位前补0。**
- 表征参数：串首地址和位长（不含符号位）。位长可变，常为0~31，甚至更长。位长为0的数其值也为0。
- **压缩的十进制数串比字符串表示方式节省一倍左右的存储空间，又便于直接完成十进制数的算术运算，是较为理想的表示方法。**

5.2 定点数的表示

5.2.1 真值与机器数

5.2.2 常用机器码表示

真值与机器数

无符号数：不考虑正负号的数，相当于数的绝对值。

整个机器字长的全部二进制位均可用来表示数值。

例如： $X_1=(0100\ 1010)_2$ ——无符号数74

$X_2=(1100\ 1101)_2$ ——无符号数205

- 无符号数表示范围

  当机器字长为n位，无符号数表示范围是0 ~（2^n-1）

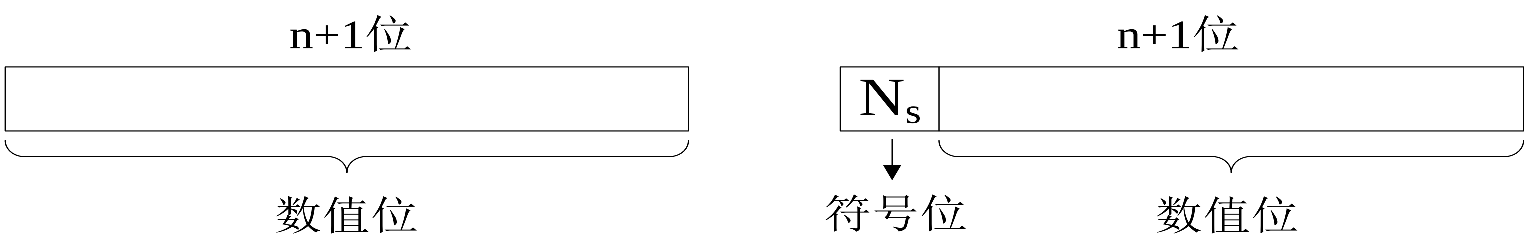
  - 例：机器字长8位，则范围为0~255

- **调用方法：通过**无符号数的运算和处理指令。

  - 如 **Intel 8086** 中的 **MUL**和 **DIV**指令——无符号数乘、除法指令等。

数符的机内表示

- 带符号数：即带有正、负号的数
- 问题：数学符号“+”、“-”在计算机中无法直接表示
- 解决思路：把符号二进制数码化，“0”、“1”两种代码正好可用来表示数的正、负两种符号。
- 具体方法：在最高数值位之前再安排一位，即二进制的最高位专门用来表示数的符号，称为“符号位”



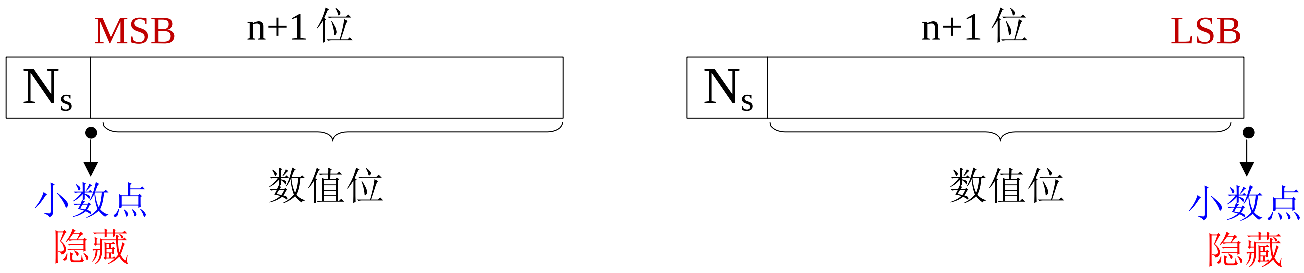
a）无符号数　　　　**b**）带符号数

小数点的机内定位

Technology
Science
Computer

- 表示难点：**小数点在数据格式中无法再用二进制代码表示**
- 表示方法：**隐含表示。数据格式中并不明显地给出小数点，事先约定好小数点的位置，按照约定识别数值。**
- 两种形式：**定点表示法和浮点表示法。浮点表示法在本章6.5节中将进行专门的阐述。**
  - 定点表示法：指小数点位置在计算机中是事先约定好且固定不变的。两种定位方式：
    - 定点整数表示法：**小数点位置定在最低数值位之后**
    - 定点小数表示法：**小数点的位置定在符号位之后**

小数点的机内定位



a）定点小数　　　　b）定点整数

定点表示法的特点：表示方法简单，但表示范围较小，（纯整数、纯小数）。原始数据需乘上一定的比例因子，变为整数或小数形式，定点计算机内部才能接受。对于非常大或非常小的数据，合适的比例因子不好找。

真值与机器数

真值——指用一般书写形式表示的数。

特点：数的符号用“+”、“-”号表示，根据符号的形状区分正、负。

机器数——指计算机内数据的形式。

特点：数的符号数字化了，用二进制代码“0”、“1”表示数的符号，数值部分与数符按照一定的编码规律进行组合。

常见的机器数编码方案（码制）有：原码、反码、补码、移码。

原码表示法

小数的原码表示：设X为真值

定义：
$$[X]_{原} = \begin{cases} X & 0 \le X < 1 \\ 1-X = 1+|X| & -1 < X \le 0 \end{cases}$$

实例：X = +0.10110　　　-0.10110

$[X]_{原}$= 0.10110　　　1.10110

整数的原码表示

定义：
$$[X]_{原} = \begin{cases} X & 0 \leqslant X < 2^n \\ 2^n - X = 2^n + |X| & -2^n < X \leqslant 0 \end{cases}$$

实例：X = + 10110　　　-10110

$[X]_{原}$= 0,10110　　　1,10110

原码表示法

零的原码表示

由于正、负域中都包含“0”，造成原码有“+0”和“-0”两种零的表示形式。

例：求X=±0.000 0000的原码

解：当X = +0.000 0000时，$[X]_{原}$= X =0.000 0000

当X = -0.000 0000时， $[X]_{原}$=1-X=1.000 0000

如果将±0代入定点**整数**的原码定义式，同样可以得到$[+0]_{原}$和$[-0]_{原}$两种形式。

特点：原码表示法**直观**；

与二进制真值之间的**转换方便**；

原码**乘除运算**的规则简单；加减运算复杂；

零的表示不唯一，给判“0”操作带来麻烦。

补码表示法

补码表示的基本原理

计算机内运算的特点：**模运算**。

“模”的概念：模运算系统中编码表示范围有上限，运算时一旦达到此值即自动丢掉，称为“**模溢出**”。

模运算数轴呈头、尾封闭的圆周形，日常生活中关于模运算最典型的例子是**钟表系统**。

补码表示法

小数的补码表示（模2补码）

定义：$[X]_{补} = \begin{cases} X & 0 \leq X < 1 \\ 2+X & -1 \leq X < 0 \end{cases}$ Mod 2

实例：X = +0.10110　　-0.10110

$[X]_{补}$ = 0.10110　　1.01010

整数的补码表示

定义：$[\,X\,]_{补} = \begin{cases} X & 0 \leqslant X < 2^n \\ 2^{n+1} + X & -2^n \leqslant X < 0 \end{cases}$ Mod 2^{n+1}

定点整数补码与定点小数补码的形式完全相同，运算特性和转换规则也一样，差别仅表现在小数点的位置不同

补码表示法

补码的转换方法

负数真值求补码的方法：

① 按照定义做减法 ——最基本

② 按位变反+1LSB ——变减为加，符号位“0正1负”

③ 按位变反‖10…0 ——不加减，符号位“0正1负”

除符号外，上述方法②、③也适用于负数补码求真值的逆转换

补码与真值的转换关系式：

若 $[X]_{补} = X_0.X_1X_2 \ldots X_n$ （Mod 2）

则

$$X = -X_0 + \sum_{i=1}^{n} X_i \times 2^{-i}$$

通过此关系式可直接将补码转换成真值形式（包括符号），这在补码运算公式推导时很有用。

补码表示法

- 例1： $X = +0.101\ 10$ ， $[X]_{补}=0.101\ 10$ ——用定义求
- 例2： $X= -0.101\ 10$ ，求 $[X]_{补}$
  - ① $[X]_{补}=10.000\ 00-0.101\ 10= 1.010\ 10$ ——用定义求
  - ② $[X]_{补}=1.010\ 01+0.000\ 01=1.010\ 10$ ——变反+1LSB
  - ③ $[X]_{补}=1.010 \parallel 10=1.010\ 10$ ——高位变反，低位不变

- 例3： $[X]_{补}=0.101\ 10$ ， $X=-0+0.101\ 10=+0.101\ 10$ ——转换公式
- 例4： $[X]_{补}=1.010\ 10$ ，求X
  - ① $X=-1+0.010\ 10=-(1- 0.010\ 10)=- 0.101\ 10$ ——转换公式
  - ② $X= -(0.101\ 01+0.000\ 01)= - 0.101\ 10$ ——变反+1LSB
  - ③ $X= -(0.101 \parallel 10)= - 0.101\ 10$ ——高位变反，低位不变

补码零

- 补码的正负定义域不对称，负数定义域不包括“0”，“0”只包括在正数定义域中。
- 补码“0”的表示唯一，不再有“+0”、“-0”之分
- 减少了计算机中判“0”操作的麻烦。
- 例：求X=±0.000 0000的补码。
- 解：当X=+0.000 0000时，[X]补=X=0.000 0000；

  当X=-0.000 0000时，

  $[X]_{补}=2-|X|=10.000\ 0000-0.000\ 0000$

  $=0.000\ 0000-0.000\ 0000 \pmod 2$

  $=0.000\ 0000 \pmod 2$

则 $[0]_{补}=[+0]_{补}=[-0]_{补}=0.00……0$

补码表示的下限

- 由于补码“0”的表示唯一，只需一个码点，比原码省出了一个码点。
- 利用省出的这个码点，补码可以多表示一个数。
- 因此，负数补码表示的**下限可达（-1.00……0）**

- 例：求X=-1.000 0000的补码。
- 解：当X=-1.000 0000时，

  $[X]_{补}=2-|X|=10.000\ 0000-1.000\ 0000=1.000\ 0000$

- 此时补码符号位的“1”有**双重**含义：

  ① **表示负号**

  ② **表示数值“1”**

- **注意**：此时“-1”仍属于定点小数模2补码表示范围。

变形补码

- 双符号位补码，设置两个符号位，称为“变形补码”
  正数符号位取值“00”，负数符号位取值“11”
- 意义：变形补码的“模” 扩大了一倍，变成模 4 补码。则模 4 补码的表示范围也比模 2 补码扩大了一倍。利用这个特性可实现补码运算的溢出判断。
- 模4补码定义如下：

  若设X=±0.X_1X_2……X_n代表 *n* 位小数的真值，则

$$[X]_{补} = \begin{cases} X & 0 \le X < 2 \\ 4+X = 4-|X| & -2 \le X < 0 \end{cases} \quad (\mathrm{mod}\ \ 4)$$

- 模4补码除了具有双符号位以外，其它特性和转换方法均与模 2 补码类似。

反码表示法

小数的反码表示（1的补码）

定义：$[X]_{反} = \begin{cases} X & 0 \le X < 1 \\ (2-2^{-n}) + X & -1 < X \le 0 \end{cases}$ Mod $(2 - 2^{-n})$

实例：$X = 0.10110$ $\quad -0.10110$

$[X]_{反}=0.10110$ $\quad 1.01001$

整数的反码表示

$[X]_{反} = \begin{cases} X & 0 \leqslant X < 2^n \\ (2^{n+1}-1) + X & -2^n < X \leqslant 0 \end{cases}$ Mod $(2^{n+1}-1)$

实例：$X = 10110$ $\quad -10110$

$[X]_{反}=0,10110$ $\quad 1,01001$

反码表示法

❑反码“0”的表示：

$$[+0]_{反} = +0 = 0.00\ldots0$$

$$[-0]_{反} = (2-2^{-n}) + (-0)$$

$$=1.11\ldots11-0.00\ldots0=1.11\ldots11$$

分+0（ **0.00…0**）、-0（**1.11…11**）两种表示方式。

❑ 特点

- 零的表示不唯一，给机内判 0 带来不便；
- 与补码有许多相似性质，同样是模运算、利用反码也可以将减法转换为加法来做等；
- 反码的模不是2的整幂，在运算发生模溢出（丢掉的是2的整幂）时多丢掉了1LSB，需要将丢掉的1LSB再加回结果的末位去（循环进位），导致反码运算不如补码运算方便。

三种机器码的比较

- 整数原、补、反码与小数表示基本相同，仅小数点位置不同
- 符号位按0正1负设置，原码、补码、反码均相同；
- 正数 原、补、反码均相同，符号位为 0，数值位同数的真值
- 原码和反码零的表示不唯一，补码零的表示唯一；
- 负数的原码、补码、反码表示均不同，符号位为 1，
  - 数值位：原码为数的绝对值；
  - 反码为绝对值按位取反；
  - 补码高位部分同反码，低位部分同原码；
- 补码数值位与原、反码转换规则双向适用，符号位不需转换
- 原、反码定义域相同，表示范围零对称。补码正、负定义域不对称，正数表示范围同原码，负数较正数能多表示1个数
- 补码和反码的符号位和数值位一起参加运算；原码符号位不能参加运算，必须单独进行处理。

三种机器码的比较

二进制代码	原码 对应的真值	反码 对应的真值	补码 对应的真值
0.000 0000	+0	+0	0
0.000 0001	+1/128	+1/128	+1/128
0.000 0010	+1/64	+1/64	+1/64
……	……	……	……
0.111 1101	+125/128	+125/128	+125/128
0.111 1110	+63/64	+63/64	+63/64
0.111 1111	+127/128	+127/128	+127/128
1.000 0000	−0	−127/128	−1
1.000 0001	−1/128	−63/64	−127/128
1.000 0010	−1/64	−125/128	−63/64
……	……	……	……
1.111 1101	−125/128	−1/64	−3/128
1.111 1110	−63/64	−1/128	−1/64
1.111 1111	−127/128	−0	−1/128

定点表示法的特点

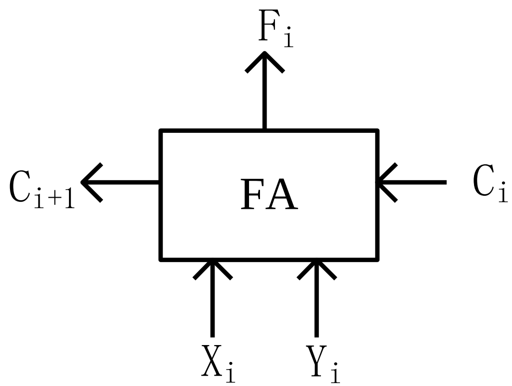
- 表示方法简单，对应的运算方法也简单，因此使得定点机结构简单，硬件代价低；
- 由于计算机的字长有限，定点数的表示范围较小，运算结果容易超出其表示范围，导致出错（溢出）。
- 只能表示纯小数或纯整数，需要选择合适的“比例因子”将原始数据变为纯小数或纯整数形式，不好选且比较麻烦。
- 如果想要有效地扩大机器数的表示范围，常用的方法是采用浮点表示法，将在6.5节讨论。

5.3 定点运算

运算部件的基本结构

一位二进制加法单元

全加器的逻辑符号



全加器真值表

输 入			输 出	
X_i	Y_i	C_i	F_i	C_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

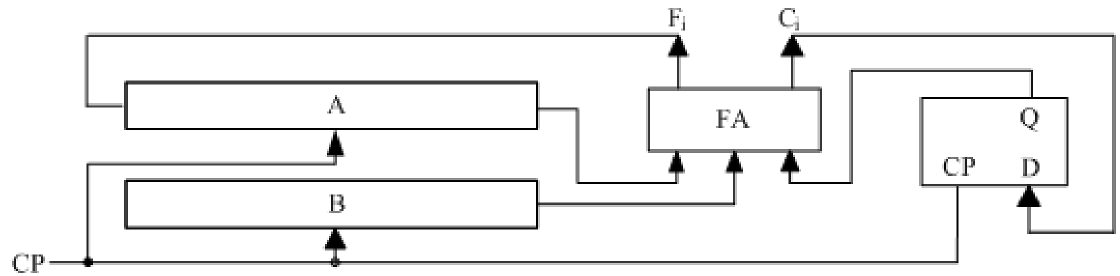
全加器的典型逻辑表达式

$$\left.\begin{array}{l} F_i = X_i \oplus Y_i \oplus C_i \\ C_{i+1} = X_iY_i + (X_i \oplus Y_i)C_i,\ (\text{或}C_{i+1} = X_iY_i + (X_i + Y_i)C_i) \end{array}\right\}$$

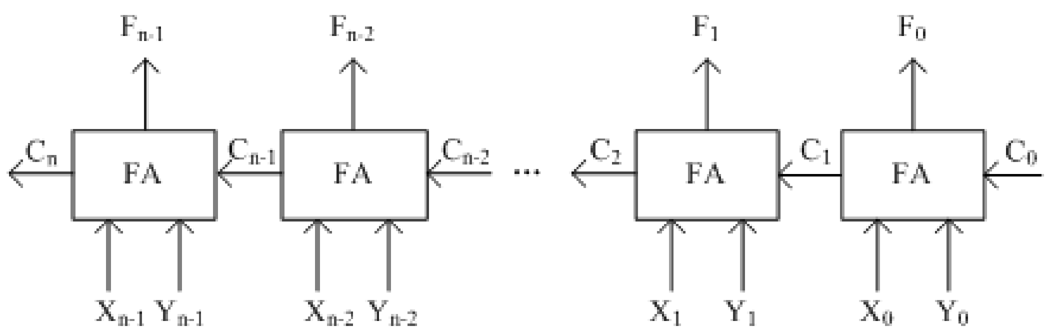
运算部件的基本结构

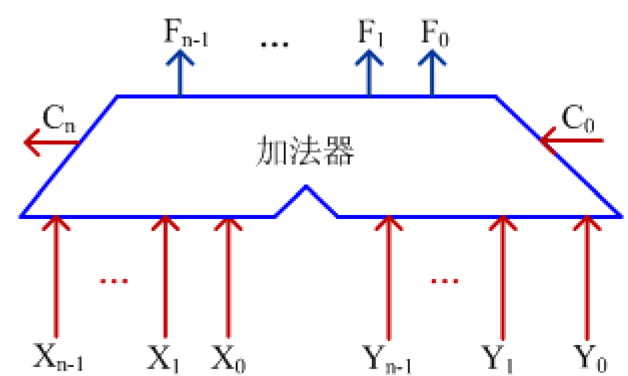
n位加法器的基本结构

1）串行加法器



2）并行加法器





定点补码加减运算

补码运算的基本特点

- 参与运算的操作数均用补码表示；
- 按补码运算规则进行运算；
- 补码的符号位与数值位视为整体一起参加运算；
- 结果的符号位由运算自动产生；
- 运算过程中符号位向高位产生的进位自动丢掉（**模溢出**）；
- 补码运算的结果亦为补码。

补码加减运算基本公式

$[X \pm Y]_补 = [X]_补 \pm [Y]_补$ （mod 2）

即：$[X+Y]_补 = [X]_补 + [Y]_补$ （mod 2）

$[X-Y]_补 = [X]_补 + [-Y]_补$ （mod 2）

求$[-Y]_补$，可通过对 $[Y]_补$逐位取反,再在最低位加 1完成。

例：已知真值X、Y，试用补码加减算法求X ± Y=？

X=0.1011011，Y=-0.0010010

解：$[X]_补$=X=0.1011011；$[Y]_补$＝1.1101110；$[-Y]_补$＝0.0010010

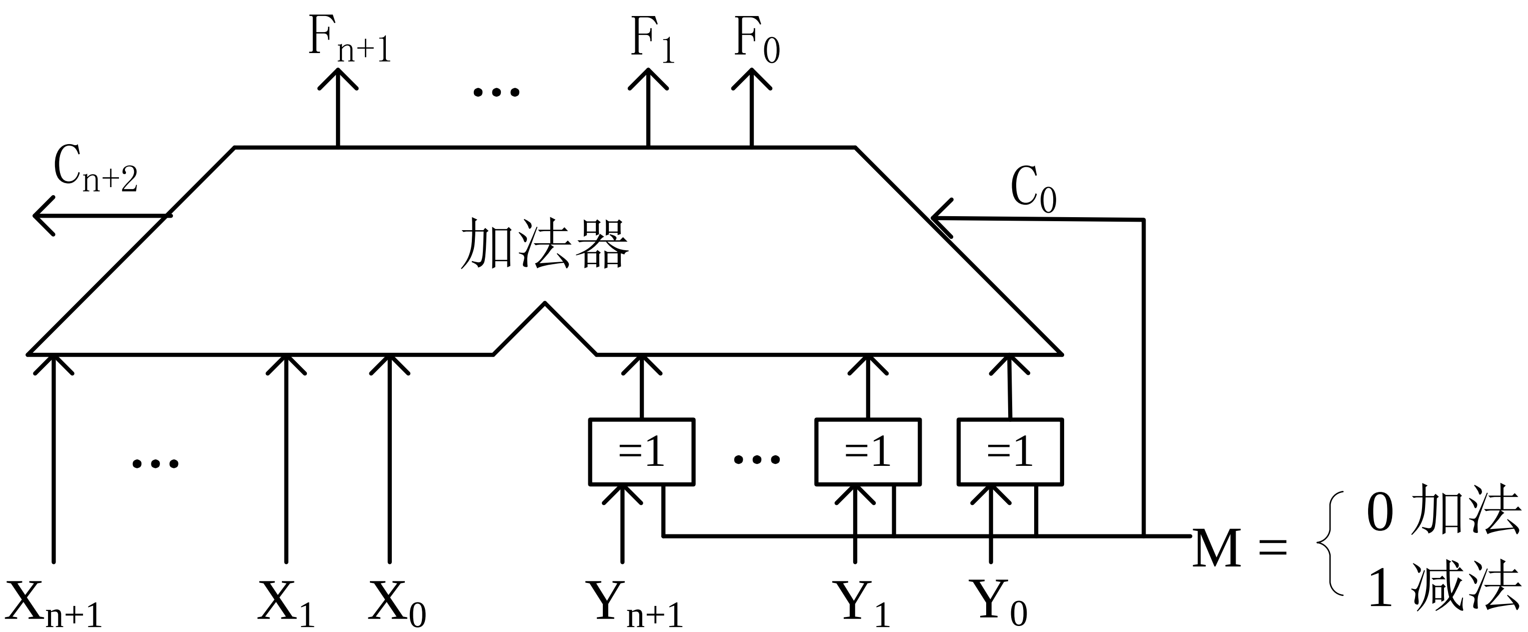
$[X+Y]_补$=0.1011011＋1.1101110

=10. 1001001=0. 1001001 （mod 2） （模溢出，自动丢2）

$[X-Y]_补$=0.1011011＋ 0.0010010 =0. 1101101

X+Y=[X＋Y]补=0. 1001001； X-Y =0. 1101101

二进制补码加/减法器逻辑框图



特点：

- ❑ **在加法器的一个输入端添加一组异或门实现求补运算；**
- ❑ **在异或门的输入端设置加/减控制信号M进行运算选择。**

补码加减法的溢出判别

溢出（Overflow）：运算结果超出数据的表示范围。分为正溢出和负溢出。

例：(1) X=-0.1101100，Y=-0.1011011，求X+Y

(2) X=0.1101100，Y=-0.1011011，求X-Y

解：(1) $[X]_{补}$= 1.0010100；$[Y]_{补}$＝1. 0100101；$[-Y]_{补}$=0.1011011

$[X+Y]_{补}$=1.0010100+1. 0100101=10.0111001=0.0111001（mod 2）

分析：① 模溢出，自动丢2（不影响结果正确性）；

② 负加负结果应为负，但现在结果符号位为0，出错（负溢出）

(2) $[X]_{补}$= 0.1101100；$[Y]_{补}$＝1. 0100101；$[-Y]_{补}$=0.1011011

$[X-Y]_{补}$=0.1101100＋0.1011011=1.1000111

分析：正数减去负数结果应为正，但现在结果符号位为1，出错（正溢出）

溢出的后果：结果的最高数值位侵入符号位，使符号位遭到破坏。在计算机中溢出是作为出错处理的。

补码加减法的溢出判别

溢出判断：同一件事实，三种不同的判别方式。

(1) **正+正得负；负+负得正；**

$$V=(\overline{X_s}\cdot\overline{Y_s}\cdot F_s+X_s\cdot Y_s\cdot\overline{F_s})\overline{M}+(\overline{X_s}\cdot Y_s\cdot F_s+X_s\cdot\overline{Y_s}\cdot\overline{F_s})M$$

M = 0 , 加法； M = 1 , 减法

(2) 最高数值位与符号位向更高位的进位不同时产生； $V=C_s \oplus C_{MSB}$

(3) 双符号位的值为 01或10。溢出时，**高位符号保持正确符号。** （双符号位补码也称**变形补码**，**模=4**）

$$V= F_{s1} \oplus F_{s2}$$

补码加减法运算规则——模4补码为例

- 两个操作数均用双符号位补码表示；
- 两个符号位与数值位作为整体一起参加运算；
- 若作加法，两数补码直接相加；
  若作减法，减数求补后再与被减数相加；
- 模溢出进位信号自动丢掉，不影响运算的正确性；
- 结果为双符号位补码，两符号位均由运算自动产生；
- 若结果的两个符号位相同，无溢出，运算结束；
  若结果的两个符号位相异，有溢出，转溢出处理；
  最高符号位为结果的正确符号。

- 定点整数补码加减算法规则与定点小数补码加减运算基本相同，只是补码的模变了，小数点在最低数值位之后。

补码加减法运算举例

X = 0.1011； Y = -0.0101；

用模 4 补码（双符号位）运算

$[X]_{补}$ = 00.1011， $[Y]_{补}$ = 11.1011

$[-Y]_{补}$ = 00.0101

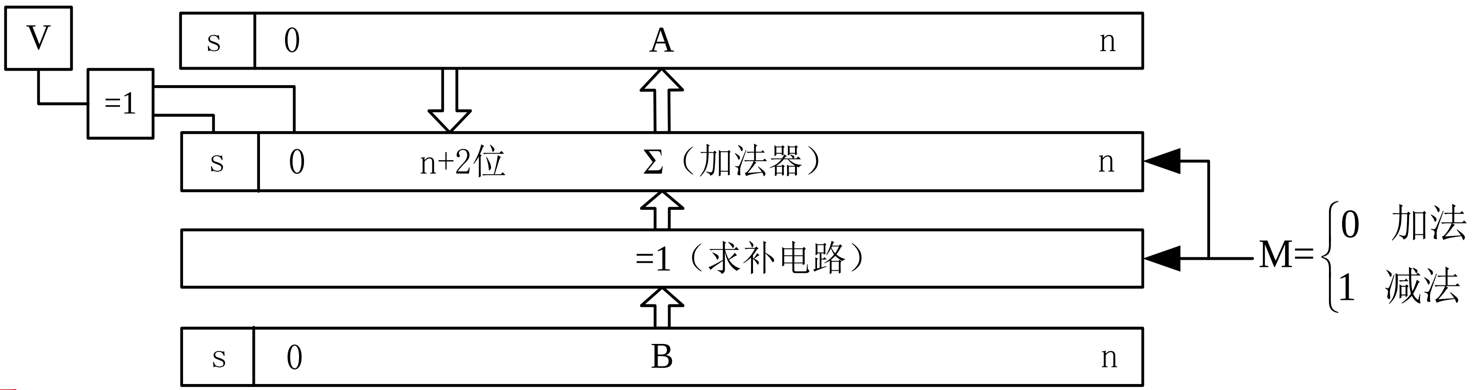
$$
\begin{array}{r}
00.1011 \\
+\ 11.1011 \\
\hline
00.0110
\end{array}
\qquad
\begin{array}{r}
00.1011 \\
+\ 00.0101 \\
\hline
01.0110
\end{array}
$$

X+Y，去掉最高位进位

X-Y，结果正溢出

定点补码加减法的硬件配置

模4补码加减运算器逻辑框图



特点：

- 数值位为n位，运算器的总位数应达到n+2位；
- 补码加减运算器主要由一个加/减法器和两个寄存器组成；
- 寄存器A为累加器，用来存放运算结果，下一次运算可继续使用；
- 加上溢出判别、操作控制等辅助逻辑电路，完善加减运算功能；

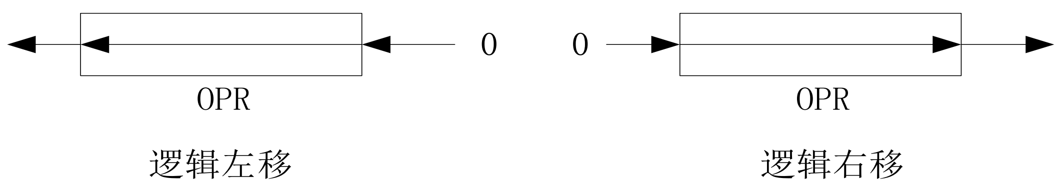
- 整数补码加减与定点小数补码加减法运算原理一样，硬件线路也完全相同，只是小数点的默认位置在操作数的最低位（LSB）之后

移位运算

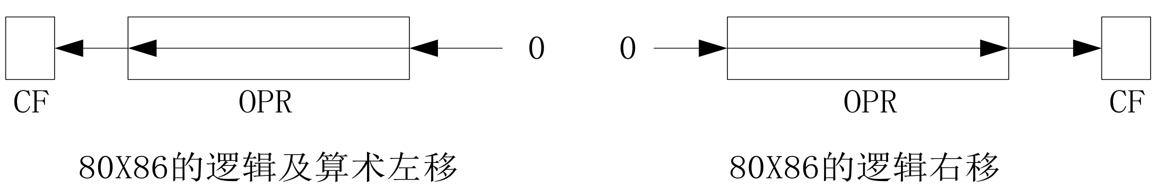
- 移位运算又叫移位操作，用于提高某些运算的速度或作为乘除法运算的子运算。分为算术移位、逻辑移位和循环移位三类
  - 逻辑移位：不考虑数据符号位含义的移位操作，用于对无符号数（或逻辑数）进行的移位。包括逻辑左移、逻辑右移。
  - 循环移位：将机器字首尾相接进行的移位；用来满足位测试等功能的操作需求，也分循环左移和循环右移等几种形式。
  - 算术移位：对带符号数进行的移位，包括算术左移和算术右移，在移位的过程中需要特别注意保持符号位不发生改变。
- 这三类移位运算规则上的区别主要反映在符号位操作及对移位后空出位填补值不同这两方面。

逻辑移位

- **逻辑移位规则**
  - **逻辑左移时，高位移丢，低位补0；**
  - **逻辑右移时，低位移丢，高位补0。**
- **逻辑移位示意**



（a）逻辑移位操作示意



（b）逻辑移位在80x86中的实现

循环移位

❒ 循环移位规则

移位通路构成一个封闭的环路，因此不管是循环左移还是循环右移，其移出位都会自动填补到空出位中

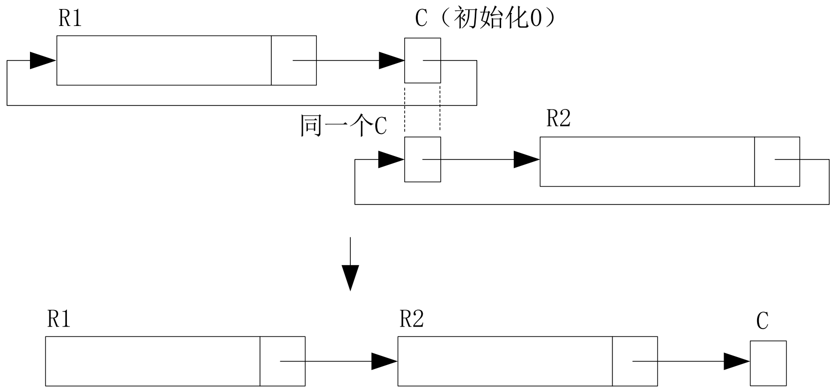
- 循环左移：最高位移出填入最低位；
- 循环右移：最低位移出填入最高位。

（a） 循环移位操作示意

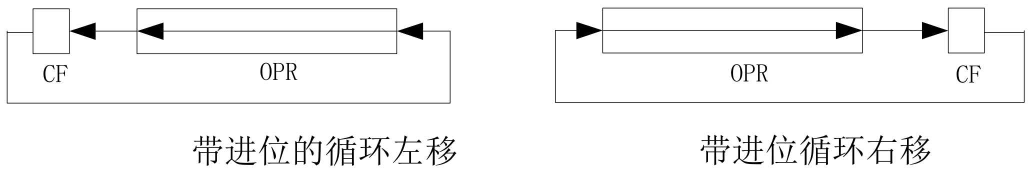
（b） 循环移位在80x86 指令集中的实现

带进位的循环移位

- 带进位循环移位操作将进位标志位加入移位循环回路中，利用进位标志位可以实现双字长移位时**移出位在两个寄存器之间的传递**；也可用于机器字中某一位的位操作



(a)双字长移位的过程



（b）80X86带进位循环移位指令

算术移位

规则：

(1) 移位后符号位不变；

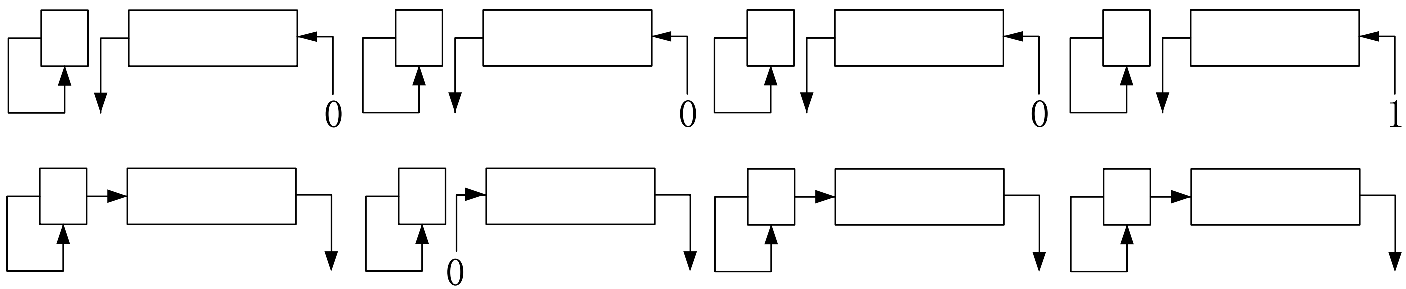
(2) 不同码制机器数移位后空位填补值要符合所用的机器码编码规律。

原码、补码、反码的算术移位空位填补规则：

	码　制	右移填补代码	左移填补代码
正数	原码、补码、反码	0	0
负数	原　码	0	0
	补　码	1	0
	反　码	1	1

算术移位

❑ 算术左移和右移操作的硬件示意框图



(a)正数　　（b）负数原码　　（b）负数补码　　（b）负数反码

❑ 移位后对值的影响

正数： 左移时最高数位丢1，结果溢出出错；
右移时最低数位丢1，影响精度。

负数： 原码左移时，高位丢1，结果溢出出错；
右移时，低位丢1，影响精度；
反码左移时，高位丢0，结果溢出出错；
右移时，低位丢0，影响精度；
补码左移时，高位丢0，结果溢出出错；
右移时，低位丢1，影响精度；

补码算术移位的符号延伸特性

补码算术移位时的符号延伸（扩展）特性

- 右移后对**最高数值位（MSB）的填充值等于符号位的值**，相当于右移时符号位在自身保持不变的前提下，同时移入空出的MSB位。这个特性称为补码的**符号延伸特性**
- 补码右移运算公式：
  - 若 $[X]_{补} = X_0.X_1X_2…X_n$　　Mod 2 或 Mod 2^{n+1}
  - 则 $[1/2X]_{补}=X_0.X_0X_1 … X_{n-1}$　　X_n移出丢掉

    $[1/4X]_{补}=X_0.X_0X_0X_1 … X_{n-2}$　　$X_{n-1}X_n$移出丢掉

    $[1/8X]_{补}=X_0.X_0X_0X_0X_1 … X_{n-3}$　　$X_{n-2}X_{n-1}X_n$移出丢掉

    ……符号可一直延伸到所需右移的位数为止

补码算术移位的符号延伸特性

求证：(以定点小数为例）

若 $[X]_{补} = X_0.X_1X_2 \ldots X_n$ Mod 2

则 $[1/2X]_{补} = X_0.X_0X_1 \ldots X_{(n-1)}$

证明：因为 $[X]_{补} = X_0.X_1X_2 \ldots X_n$ Mod 2

所以 $$X = -X_0 + \sum_{i=1}^{n} X_i \times 2^{-i}$$

$$1/2X = -1/2X_0 + 1/2\sum_{i=1}^{n} X_i \times 2^{-i}$$

$$= -X_0 + \sum_{i=1}^{n} X_i \times 2^{-(i+1)}$$

则 $[1/2X]_{补} = X_0.X_0X_1 \ldots X_{(n-1)}$ （设字长不变）

用相同的方法，还可以进一步证出$[1/4X]_{补}$、$[1/8X]_{补}$……

补码算术移位的符号延伸特性

- ❑ **补码的符号延伸特性也可以反过来用于补码位数的扩展**

移位前：	X_0	X_1			X_n
右移一位后：	X_0	X_0	X_1		X_{n-1}
右移二位后：	X_0	X_0	X_0	X_1	X_{n-2}

字长n+1位不变

右移时的符号延伸

X_0	X_1	X_7	扩展前8位

,

X_0	$X_0 \cdots\cdots X_0$	X_0	X_1	X_7	扩展成16位

,

X_0	$X_0 \cdots\cdots X_0$	$X_0 \cdots\cdots X_0$	X_0	$X_0 \cdots\cdots X_0$	X_0	X_1	X_7	扩展成32位

,

字长转换时的符号扩展（定点整数）

算术右移误差的舍入处理

舍入——为了提高运算结果的精度，在舍去多余位时，并按照一定规则对保留的数值位进行调整。

硬件支持：需要在有效数据字长之外多设若干位（保护位），先把运算过程中右移出的前几位数暂时保存起来，以供舍入判断之用。保护位一般通过增设保护位寄存器实现。

主要舍入方法：

1）截断法：也叫截尾，无条件地丢掉结果最低位之后超出部分的数值，即“恒舍”，实现起来最简单。

- **单向误差，每次舍入后可能产生的误差都是负误差（对结果精确值起减小作用）。单次误差< - 1LSB；**
- **由于是单向误差，累积误差可能会很大，所以对运算结果的精度影响较大。**

算术右移误差的舍入处理

2）**末位恒置1法：**在舍去结果最低位之后数值的同时，将机器数末位置1。

- **双向误差，当机器数末位为0时，可能产生正误差；当机器数末位本来就为1时，则可能产生负误差。单次误差< ±1LSB**
- **不会产生累积误差，对精度影响比截断法小得多。**
- **在具体实施时，为了进一步减小误差，常采用：**

当机器数最低位=1，或移出去的位中有1，则末位置1；否则，舍去移出去的各位，不做末位置1的操作。

0舍1入法

- 基本思想：用移出部分的最高位（舍取位）作判断标志

  如果该位为0，则“舍”——操作同截断法；

  如果该位为1，则“入”——在丢掉移出部分的位数后，对数值保留部分的最低位+1。
- 双向误差，单次误差<±1/2LSB，显然比前两种舍入方法的单次误差小一倍；
- 没有累积误差，精确度最高，但操作规则比前两种舍入方法复杂，需要考虑采用的是哪一种码制。
- 当用原码表示，或用补码表示的正数时，

  若移出部分的最高位=0，舍去；

  若移出部分的最高位=1，在舍去时，将机器数末位+1。
- 此时“舍”使数据的绝对值变小；“入”使其绝对值变大。

0舍1入法

- 当补码负数舍入时，

 若移出部分的最高位为0，舍去；

 若移出部分的最高位为1，其余各位全为0，舍去；

 若移出的最高位为1，其余各位不全为0，舍去时，末位+1。

- 此时“舍”使绝对值变大；“入”使绝对值变小。

- “0舍1入”法虽然误差最小，但“入”的操作需要进行一次加法运算，且可能会引起数据溢出，所以处理较麻烦且影响速度。

误差的舍入处理举例

例：设X_1=0.0111000010，Y_1=-0.0111000010；
X_2=0.0111001100，Y_2=-0.0111001100；

分别用**原码和补码**表示，如果只要求**8位字长**，请分别采用**截断法、恒置1法和0舍1入法**对每一个操作数进行舍入，并对舍入结果进行比较。

解：先将真值X_1、X_2、Y_1、Y_2表示成机器码形式，再进行舍入，为方便比较，舍入结果用表格列出（见下页）。

注意相同下标的X_i、Y_i互为相反数，LSB^*则表示误差方向是相对于最低有效位LSB的绝对值而言。

误差的舍入处理举例

舍入前（11位）	舍入后（8位）	丢掉位	结果真值	误差分析
$[X_1]_{原}=[X_1]_{补}=X_1$ =0.0111000010	截断=0舍1入 =0.0111000 (舍) 恒置1=0.0111001 (入)	010	0.0111 0.0111001	-1/4LSB* +3/4LSB*
$[Y_1]_{原}$=1.0111000010	截断=0舍1入 =1.0111000 (舍) 恒置1=1.0111001 (入)	010	-0.011 1 -0.0111001	-1/4LSB* +3/4LSB*
$[Y_1]_{补}$=1.1000111110	截断=恒置1 =1.1000111 (入) 0舍1入=1.1001000 (舍)	110	-0.0111001 -0.0111	+3/4LSB* -1/4LSB*
$[X_2]_{原}=[X_2]_{补}=X_2$ =0.0111001100	截断=恒置1 =0.0111001 (舍) 0舍1入=0.0111010 (入)	100	0.0111001 0.011101	-1/2LSB* +1/2LSB*
$[Y_2]_{原}$=1.0111001100	截断=恒置1 =1.0111001 (舍) 0舍1入=1.0111010 (入)	100	-0.0111001 -0.011101	-1/2LSB* +1/2LSB*
$[Y_2]_{补}$=1.1000110100	截断=0舍1入 =1.1000110 (入) 恒置1=1. 1000111 (舍)	100	-0.011101 -0.0111001	+1/2LSB* -1/2LSB*

移位运算的硬件实现

1）移位寄存器

- 使用具有移位功能的寄存器部件实现移位；
- 特点：每次只能移一位，当左移或右移n位时，需要n个打入时钟周期时间才能完成。移位速度慢，移位和加减运算不能同时进行。

2）组合逻辑的移位器

- 组合逻辑的移位器电路实际上相当于一个组合逻辑的多路选择器，三选一的多路选择器可实现将运算结果按位直送、左斜一位传送（左移一位）、右斜一位传送（右移一位）的移位器功能，而五选一的多路选择器可在此基础上再加两路传送，即左斜两位传送（左移二位）、右斜两位传送（右移二位）。
- 特点：可实现多位并行移位，移位操作和加减运算可在同一个时钟周期内完成，广泛用于计算机的运算器中。

桶形移位器（barrel shifter）

- 一种并行度极高的快速移位器，可在数据通路宽度范围内将一个数据移动任意位。例如，当机器字长32位时，一个32位的桶形移位器即可以实现数据左移一位、右移一位的普通移位操作，也可以实现左移10位、右移10位，或左移32位、右移32位的操作。
- 实现方法：仍然基于组合逻辑多路选择器的使用。
- 例如：32位桶形移位器的设计，可用一个32选1的多路选择器实现将32位数据中的任意一位（包括不移位直送方式）移到一个特定位的操作，32位字长一共需要32个32选1的多路选择器。
- 具体应用：32选1规模的多路选择器并不常见，可用多个较小规模的多路选择器多级连接构成。在目前LSI技术支持下，桶形移位器在计算机数据通路中得到了广泛应用。

本章第一次作业（总第8次作业）

- 5.12、5.17、 5.18、5.19

5.19仅做原码加减交替除法

课内测试

CRT字符显示器可显示128种ASCII字符，每帧可显示64字×25排；每个字符字形采用7×8点阵，即横向7点，字间间隔1点（为方便起见和点阵一起存在ROM中），纵向8点，排间间隔6点；帧频100Hz，行频49KHz，点频29.792MHz，采用逐行扫描方式，问：

（1）缓存容量至少有多大?

（2）字符发生器（ROM）容量至少有多大?

（3）缓存中存放的是ASCII代码还是点阵信息?

（4）设置哪些计数器以控制缓存访问与屏幕扫描之间的同步？它们的分频关系如何?