第 1 章

基 础 知 识

本章主要介绍学习汇编语言的重要性及计算机的一些基础知识，包括计算机中数和字符的表示、8086/8088 CPU 的编程结构、存储器及堆栈等内容。

1.1 汇编语言概述

1.1.1 为什么要学习和使用汇编语言

计算机程序设计语言大致可分为高级语言、低级语言、机器语言。汇编语言属于低级语言。高级语言(如 BASIC、PASCAL、C 等)具有‘易用好学’的特点。因为即使你不懂计算机内部的组成，采用高级语言仍然能够设计出很好的程序来。相对来说，用汇编语言来编写程序要困难一些，因为你不但要熟悉机器的指令系统，还要熟悉其内部结构，特别是中央处理器(CPU) 和存储器的结构，既然如此，为什么还要学习汇编语言呢？其原因有以下几点：

(1) 采用高级语言编写的程序，机器是不能直接执行的，需要编译程序或解释程序将它翻译成相应的机器语言程序，机器才能接受。而通过编译或者解释得到的目标代码程序比较冗长，占有存储空间大，执行速度慢，效率低，而汇编语言程序无论是在目标代码长短和程序的执行速度，还是在节省存储空间方面，都要优于高级语言，尤其在实时控制中，常常用汇编语言来编制程序。

(2) 学习和使用汇编语言可以从根本上认识和理解计算机的工作过程。学习汇编语言要熟悉计算机的内部结构，通过汇编程序的编制和运行，可以更清楚地了解计算机是怎样完成各种复杂工作的，以及如何利用计算机的所有硬件特征来直接控制硬件的。

(3) 现在计算机系统中的某些功能只能采用汇编语言来完成。例如，机器的自检、系统的初始化等。

1.1.2 汇编语言的基本概念

1. 机器语言

机器语言是用二进制编码的机器指令的集合及一组使用机器指令的规则。它是 CPU 能直接识别的唯一语言。只有用机器语言书写的程序， CPU 才能直接执行。用机器语言描述的程序称为目的程序或目标程序。用机器语言编写的程序不易为人们理解、记忆和交

流，而且容易出错。一旦出错，也很难发现和纠错。

2. 汇编语言

为了克服机器语言的缺点，人们采用便于记忆并能描述指令功能的符号来表示指令的操作码。这些符号被称为指令助记符。助记符一般是说明指令功能的英语词汇或词汇的缩写，同时也用符号表示操作数，如 CPU 的寄存器、存储单元等。

汇编指令由指令助记符及操作数构成。汇编语言是汇编指令、伪指令的集合及表示和使用这些指令的一组规则。用汇编语言书写的程序称为汇编语言程序或汇编语言源程序。用汇编语言编写的程序要比用机器语言编写的程序容易理解、调试和维护。

3. 汇编程序

由于 CPU 能直接识别的唯一语言是机器语言，所以用汇编语言编写的源程序必须被翻译成用机器语言表示的目标程序后才能由 CPU 执行。把汇编语言源程序翻译成目标程序的过程称为汇编。完成汇编任务的程序叫做汇编程序。汇编过程如图 1-1 所示。

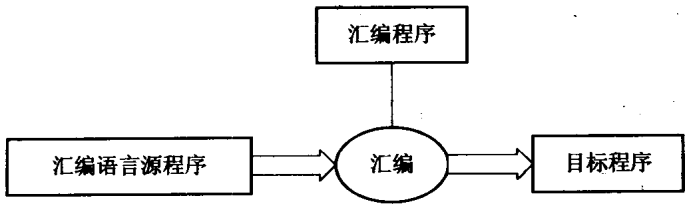


图 1-1　汇编过程示意图

1.2　计算机中数和字符的表示

1.2.1　数制及数制间的转换

1. 二进制计数

(1) 二进制计数只有两个数码，即 0 和 1。

(2) 其进位原则是“逢二进一”。

(3) 二进制基数为 2，位权为 2^k(k 为整数)。

例如：

$$(11011.101)_2 = 1\times2^4+1\times2^3+0\times2^2+1\times2^1+1\times2^0+1\times2^{-1}+0\times2^{-2}+1\times2^{-3}$$
$$= (27.625)_{10}$$

任意一个二进制数 B，可以表示成如下形式：

$$(B)_2 = B_{n-1}\times2^{n-1}+B_{n-2}\times2^{n-2}+\cdots+B_1\times2^1+B_0\times2^0+B_{-1}\times2^{-1}+\cdots+B_{-m+1}\times2^{-m+1}+B_{-m}\times2^{-m}$$

式中，B 为数位上的数码，其取值为 0 和 1；n 为整数位个数；m 为小数位个数；2 为基数，2^{n-1}，2^{n-2}，…，2^{-m}为二进制数的位权。

从上面可以看出，把二进制数转化为十进制数非常简单，只要按权展开相加即可。为了与其他进制相区别，常在二进制数后面写一个字母 B，如 10010101B。

因为二进制数只有两个数码 0 和 1 ，所以二进制数的每一位都可以用任何两个不同稳定状态的物理元件来表示。如电灯的亮和灭，晶体管的截止和导通，脉冲的有无，电位的高低等，只要规定其中一种状态表示“ 1”，另一种状态表示为‘ 0”，就可以用来表示二进制数了。在计算机内部，采用的就是二进制计数。

2. 八进制计数

(1) 八进制计数有 8 个不同的数码。即 0，1，2，3，4，5，6，7。

(2) 进位原则为‘ 逢八进一 ”。

(3) 八进制数基数为 8,位权为 8^k(k 为整数)。

例如：

$$(123.24)_8 = 1\times8^2+2\times8^1+3\times8^0+2\times8^{-1}+4\times8^{-2}$$
$$= (83.3125)_{10}$$

任意一个八进制数 Q，可以表示成如下形式：

$$(Q)_8 = Q_{n-1}\times8^{n-1}+Q_{n-2}\times8^{n-2}+\cdots+Q_1\times8^1+Q_0\times8^0+Q_{-1}\times8^{-1}+\cdots$$
$$+Q_{-m+1}\times8^{-m+1}+Q_{-m}\times8^{-m}$$

式中，Q 为数位上的数码，其取值范围为 0～7；n 为整数位个数；m为小数位个数；8为基数，8^{n-1}，8^{n-2}，…，8^{-m}是八进制数的位权。

为了与其他进制数值区别。一般在八进制数后面加上 Q,例如 723.24Q。

八进制数也是计算机中常用的一种计数方法，它可以弥补二进制数书写位数过长的不足。

3. 十六进制计数

(1)十六进制计数有 16 个不同的数码,即 0～9，A，B，C，D，E，F。

(2) 进位原则是‘ 逢十六进一 ”。

(3)十六进制基数为 16,位权为 16^k(k 为整数)。

例如：

$$(3AB.48)_{16} = 3\times16^2+A\times16^1+B\times16^0+4\times16^{-1}+8\times16^{-2}$$
$$= (939.281\ 25)_{10}$$

任意一个十六进制数 H，可以表示成如下形式：

$$(H)_{16} = H_{n-1}\times16^{n-1}+H_{n-2}\times16^{n-2}+\cdots+H_1\times16^1+H_0\times16^0+H_{-1}\times16^{-1}+\cdots$$
$$+H_{-m}\times16^{-m}$$

其中，H 为数位上的数码，其取值范围为 0～9，A～F；n为整数位个数；m 为小数位个数；16 为基数，16^{n-1}，16^{n-2}，… ，16^{-m}为十六进制的位数。

为了与其他进制数值区别。一般在十六制数后面加上 H,例如 7A.CDH。在汇编语言中，凡是以字母 A～F 打头的十六进制数，都要以‘ 0 ”开头进行书写，以避免与标识符相混淆，例如十六进制数 B7H 应写成 0B7H。

十六进制数是计算机中常用的一种计数方法，它可以弥补二进制数书写位数过长的不足。

总结以上几种计数制，可将它们的特点概括为：

(1) 每一种计数制都有一个固定的基数 J(J 为大于 1 的整数),它的每一位可取 J个不

同的数值。

(2)每一种计数制都有自己的位权,并且遵循‘逢 J 进一 ’的原则。

表 1 - 1 中列出了几种常用进位计数制的表示方法。

表 1 - 1　常用进位计数制的表示方法

十进制	十六进制	二进制	十进制	十六进制	二进制
0	0	0000	9	9	1001
1	1	0001	10	A	1010
2	2	0010	11	B	1011
3	3	0011	12	C	1100
4	4	0100	13	D	1101
5	5	0101	14	E	1110
6	6	0110	15	F	1111
7	7	0111	16	10	10000
8	8	1000			

4. 不同进位计数制间的转换

计算机内部处理的数据都是二进制数，但编程时往往会用到其他一些进制数，这就必然存在各种进制数之间的转换。不同进制数之间的转换，实质上是基数之间的转换，一般转换原则是：如果两个有理数相等，则两数的整数部分和小数部分一定分别相等。因此，各数制之间进行转换时，通常对整数部分和小数部分分别进行转换。

1 非十进制数转换为十进制数

通过前面的例子可以看出，非十进制数转换为十进制数的方法为，把各个二进制数(或八进制数，或十六进制数）写成 2(或 8，或 16)的各次幂之和的形式，然后计算其结果。

【例 1 - 1】把下列二进制数转换成十进制数：

$$(110101)_2 = 1\times2^5+1\times2^4+0\times2^3+1\times2^2+0\times2^1+1\times2^0$$
$$= (53)_{10}$$

$$(1101.101)_2 = 1\times2^3+1\times2^2+0\times2^1+1\times2^0+1\times2^{-1}+0\times2^{-2}+1\times2^{-3}$$
$$= (13.625)_{10}$$

【例 1 - 2】 把下列八进制数转换成十进制数：

$$(305)_8 = 3\times8^2+0\times8^1+5\times8^0 = (197)_{10}$$

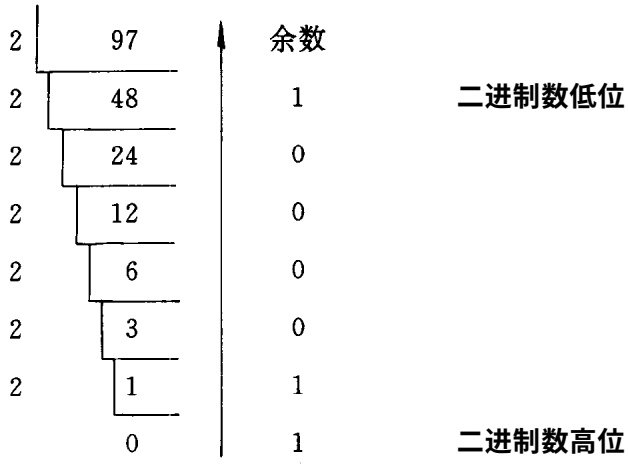
$$(456.124)_8 = 4\times8^2+5\times8^1+6\times8^0+1\times8^{-1}+2\times8^{-2}+4\times8^{-3}$$
$$= (302.1640625)_{10}$$

2）十进制数转换成非十进制数

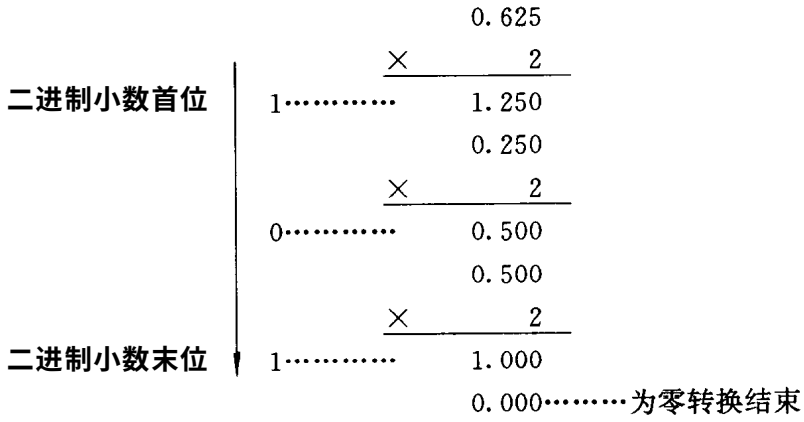
(1）十进制数转换成二进制数。把十进制数转换为二进制数的方法是：整数部分转换用‘除 2 取余法 ”,小数部分转换用‘乘 2 取整法 ”。

【例 1 - 3】 将十进制数 $(97.625)_{10}$转换成二进制数。

整数部分 97 转换如下：



小数部分 0.625 转换如下



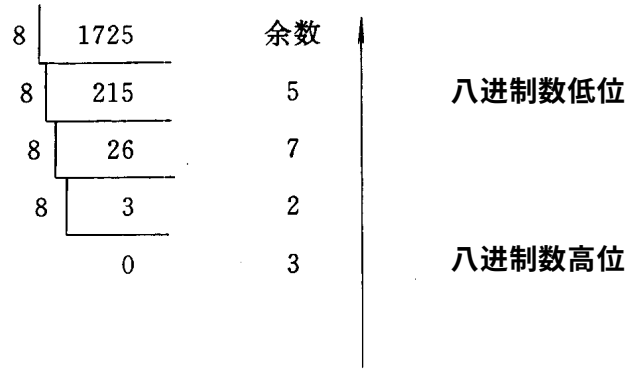
即 $(97.625)_{10}=(1100001.101)_2$

在以上例子中，小数部分经过有限次乘 2 取整过程即告结束。但有些情况下可能转换是无限的，这就要根据转换的精度要求截止。对于八进制和十六进制也有同样的情况。

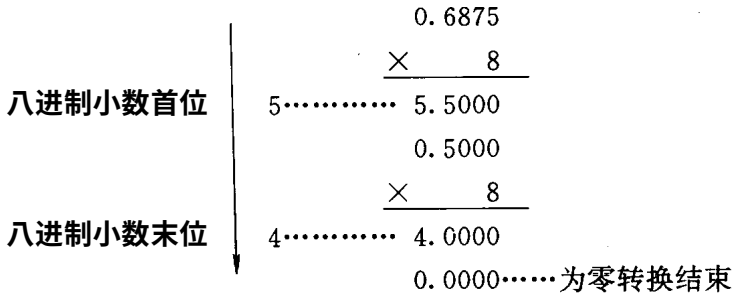
(2) 十进制数转换为八进制数。十进制数转换为八进制数的方法是：整数部分采用“除 8 取余法”，小数部分采用“乘 8 取整法”。

【例 1-4】 将十进制数 $(1725.6875)_{10}$ 转换成八进制数。

整数部分 1725 转换如下：



小数部分 0.6875 转换如下：

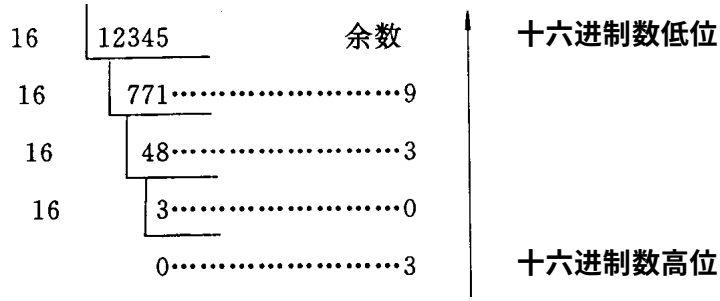


即 $(1725.6875)_{10}=(3275.54)_8$

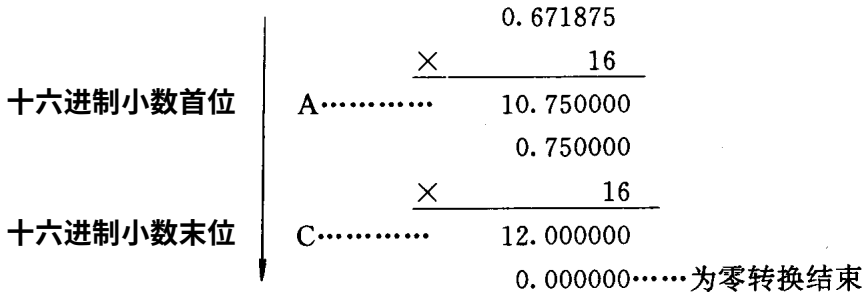
(3) 十进制数转换成十六进制数。将十进制数转换成十六进制数的方法是：整数部分转换采用“除 16 取余法”，小数部分转换采用“乘 16 取整法”。

【例 1-5】将十进制数 $(12\ 345.671\ 875)_{10}$ 转换成十六进制数。

整数部分 12 345 转换过程如下：



小数部分 0.671875 转换过程如下：



即 $(12\ 345.671\ 875)_{10}=(3039.AC)_{16}$

3) 非十进制数之间的相互转换

由于 1 位八进制数相当于 3 位二进制数，只需以小数点为界，向左或向右每 1 位八进制数用相应 3 位二进制数取代即可；反之，二进制数转换成八进制数，只是上述方法的逆过程，即以小数点为界，向左或向右每 3位二进制数用相应 1 位八进制数取代即可，如果不足 3 位，可用零补足。

【例 1-6】将八进制数 $(714.413)_8$ 转换成二进制数。

7	1	4	.	4	1	3
111	001	100	.	100	001	011

即 $(714.413)_8=(111001100.100001011)_2$

【例 1－7】 将二进制数$(1110011101.01)_2$转换成八进制数。

001 110 011 101 . 010

1 6 3 5 . 2

即 $(1110011101.01)_2=(1635.2)_8$

由于 1 位十六进制数相当于 4 位二进制数，因此要将十六进制数转换成相应二进制数，只需以小数点为界，向右每 1 位十六进制数用相应的 4 位二进制数取代即可。反之，要将一个二进制数转换成相应的十六进制数，只是上述过程的逆过程，即将二进制数以小数点为界分成左右两部分，向左或向右每 4 位二进制数用相应 1 位十六进制数取代即可，如果不足 4 位，则以零补足。

【例 1－8】将十六进制数 $(9ACD.6B)_{16}$转换成相应的二进制数。

9 A C D . 6 B

1001 1010 1100 1101 . 0110 1011

即 $(9ACD.6B)_{16}=(1001101011001101.01101011)_2$

【例 1－9】将二进制数 $(101011001.111)_2$ 转换成相应的十六进制数。

0001 0101 1001 . 1110

1 5 9 . E

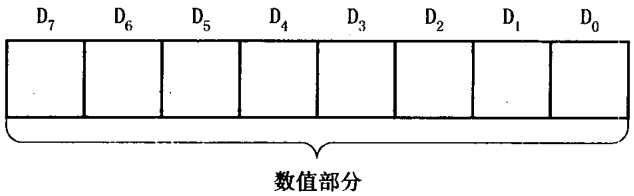
即 $(101011001.111)_2=(159.E)_{16}$

1.2.2 数和字符在计算机中的表示

1. 计算机中数的表示

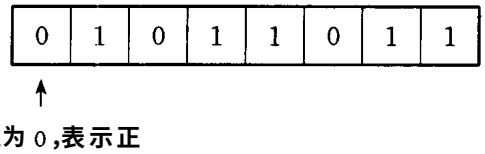
1) 真值与机器数

计算机中的数分为有符号数和无符号数。对于无符号数来说，所有二进制数位都是该数的一部分。例如，8 位无符号数的形式如下：

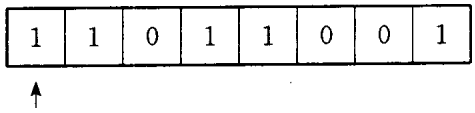


它能表示的最小无符号数是 00000000B，即 0 最大无符数号是 11111111B，即 255D。

对于有符号数，在计算机中只能用数字化的信息表示数的正负。人们规定用“ 0”表示正号，用‘ 1”表示负号。例如，在机器中用 8 位二进制数表示＋91，其格式如下：



用 8 位二进制数表示 −89，其格式如下：



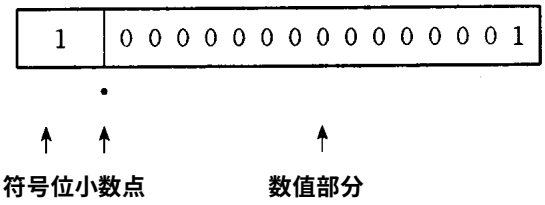
符号位为 1 表示负

在计算机内部，数字和符号都用二进制代码表示，两者合在一起构成数的机内表示形式，称为机器数。而它真正表示的数值称为这个机器数的真值。

2）定点数和浮点数

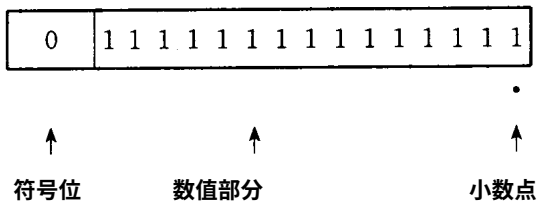
（1）机器数的表示受设备限制。计算机是以字为单位进行数据的处理、存储和传递的。所以运算器中的加法器，累加器以及其他一些寄存器，都选择与字长相同的位数。字长一定，则计算机所能表示数的范围也就确定了。例如，使用 8 位字长的计算机，它可以表示无符号整数的范围在 0～255 。如果运算数值超出机器数所能表示的范围，机器就会停止运算进行处理。这种现象称为溢出。

（2）定点数。计算机中的数，既有整数，也有小数。如何确定小数点的位置呢？通常有两种约定：一种是规定小数点位置固定不变，这时的机器数称为定点数；另一种是小数点位置可以浮动的，这样的机器数称为浮点数。对于定点数，小数点位置可以固定在符号位之后，这样的机器表示的全是定点小数。例如，假定机器字长为 16 位，符号位占 1 位，数值占有 15 位，于是机器数表示为：



其相当于十进制数为 -2^{-15}。

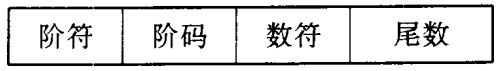
小数点位置固定在数的最后，则该机器表示的全是定点整数。例如，假设机器字长为 16 位，符号位占有 1 位，数值部分占 15 位，下面表示的机器数相当于十进制数为 +32 767 ：



定点表示法表示的数值范围有限，为了扩大定点数的表示范围，可以采用多个字节来表示一个定点数，例如，采用 4 字节或 8 字节来表示。

（3）浮点数。浮点数表示法就是小数点在数中的位置是浮动的。由于定点数表示的数的范围较窄，不能满足实际问题的需要，因此要采用浮点表示法。在同样字长情况下，浮点表示法能表示数的范围扩大了。

浮点表示法包括两部分：一部分是阶码，另一部分是尾数。浮点数在机器中的表示方法如下所示：



由尾数部分隐含的小数点位置可知，尾数总是小于 1 的数，它给出该浮点数的有效数字。尾数部分的数符确定该浮点数的正负。阶码总是整数，它确定小数点浮动的位数。若阶符为正，小数点向右移动；若阶符为负，则向左移动。

当浮点数的尾数为零或者阶码为最小值时，机器通常规定，把该数看作 0，称为“机器零”。在浮点数的表示和运算中，当一个数的阶码大于机器所能表示的最大阶码时，产生“上溢”当一个数的阶码小于机器所能表示的最小阶码时，产生“下溢”。

3）原码，反码和补码

计算机中数的编码通常有原码、反码和补码表示法。

（1）原码表示法。原码表示法是机器数的一种简单表示法，其符号位用 0 表示正号，用 1 表示负号，数值部分用二进制数表示。

【例 1-10】 $X_1=+1011011$

$X_2=-1011011$

则

$[X_1]_{原}=[+1011011]_{原}=01011011$

$[X_2]_{原}=[-1011011]_{原}=11011011$

原码表示法对 0 有两种表示形式：

$[+0]_{原}=00000000$

$[-0]_{原}=10000000$

（2）反码表示法。机器数的反码可由原码得到。对于一个正数，该数的反码与其原码相同，对于一个负数，该数的反码是其原码（符号位除外）各位取反得到的。

【例 1-11】 $X_1=+1010110$

$X_2=-1001010$

则

$[X_1]_{原}=01010110$

$[X_1]_{反}=[X_1]_{原}=01010110$

$[X_2]_{原}=11001010$

$[X_2]_{反}=10110101$

在反码表示中，对 0 有两种表示形式：

$[+0]_{反}=00000000$

$[-0]_{反}=11111111$

（3）补码表示法。机器数的补码可由原码得到。对于一个正数，则该数的补码与其原码相同；对于一个负数，则该数的补码等于它的原码（除符号位外）各位取反并在末尾加 1 而得到。

【例 1－12】 $X_1=+1010110$

$X_2=-1001010$

则

$[X_1]_{原}=01010110$

$[X_1]_{补}=[X_1]_{原}=01010110$

$[X_2]_{原}=11001010$

$[X_2]_{补}=10110101+1=10110110$

在补码表示方法中，0 只有一种表示形式：$[+0]_{补}=[-0]_{补}=00000000$。

在机器内部，负数是以其补码的形式存储和参加运算的，例如，－1、－5 和－23 分别是以其补码即 11111111(0FFH)、11111011B(0FBH)和 11101001B(0E9H)的形式进行存储的。而运算是以 $8-2=8+(-2)=00001000B+11111110B=00000110B=6$ 的方式进行的。

2. 二进制数的运算

1）二进制数的算术运算

(1) 二进制数的加法运算。加法运算有如下三条法则：

$0+0=0$

$0+1=1+0=1$

$1+1=10$

↑

向高位进位

(2) 二进制数的减法运算。减法运算有如下三条法则：

$0-0=1-1=0$

$1-0=1$

$0-1=1$

↑

向高位借 1

(3) 二进制数的乘法运算。乘法运算有如下三条法则：

$0\times0=0$

$0\times1=1\times0=0$

$1\times1=1$

(4) 二进制数的除法运算。二进制运算有如下三条法则：

$0\div0=0$

$0\div1=0$(1÷0 无意义)

$1\div1=1$

【例 1－13】 计算$(110111)_2\div(1001)_2$。

```
                  110  ←商
除数→1001 ) 110111  ←被除数
            1001
            ----
            1001
            1001
            ----
               1  ←余数
```

即$(110111)_2$除以$(1001)_2$，商为$(110)_2$，余数为$(1)_2$。

2）二进制数的逻辑运算

（1）逻辑加法(逻辑或运算)。逻辑加法运算用符号十或∨表示，有如下运算法则：

0 ∨ 0 =0　读成“0或0等于0”

0 ∨ 1 =1　读成“0或1等于1”

1 ∨ 0 =1　读成“1或0等于1”

1 ∨ 1 =1　读成“1或1等于1”

逻辑加法的特点是：只要参加运算的逻辑变量中有一个为1，那么逻辑运算结果就为1。

【例1-14】求二进制数（$10011011)_2$和$(10100101)_2$的逻辑或运算结果。

```
  10011011
∨ 10100101
----------
  10111111
```

即　10011011∨10100101=10111111

（2）逻辑乘法(逻辑与运算)。逻辑乘法运算符号用 ·或∧表示，有如下运算法则：

0 ∧ 0 = 0　读成“0与0等于0”

0 ∧ 1 = 0　读成“0与1等于0”

1 ∧ 0 = 0　读成“1与0等于0”

1 ∧ 1 = 1　读成“1与1等于1”

逻辑乘法的特点是：只要参加运算的逻辑变量中有一个为0，则运算结果为0。

【例1-15】求二进制数（$10101101)_2$和$(00101011)_2$的逻辑与运算结果。

```
  10101101
∧ 00101011
----------
  00101001
```

即　10101101∧00101011=00101001

（3）逻辑否定（逻辑非运算）。逻辑否定的运算符号是在逻辑变量的上方加一横线，例如$\overline{A}$表示对A的否定运算。其运算法则如下：

$0=1$　读成“非0等于1”

$\overline{1}=0$　读成“非1等于0”

（4）逻辑异或。逻辑异或的运算符号通常用“⊕”表示。其运算法则如下：

$0\oplus0=0$　读成“0异或0等于0”

$0\oplus1=1$　读成“0异或1等于1”

$1\oplus0=1$　读成“1异或0等于1”

$1 \oplus 1 = 0$　读成“1 异或 1 等于 0”

逻辑异或的特点是：只要两个参加运算的逻辑变量的值相同，则异或运算结果为 0。两个逻辑变量值不同时，异或运算结果为 1。

【例 1-16】求二进制数 $(10101101)_2$ 和 $(00101011)_2$ 的逻辑异或运算结果。

$$\begin{array}{rr} & 10101101 \\ \oplus & 00101011 \\ \hline & 10000110 \end{array}$$

即　$10101101 \oplus 00101011 = 10000110$

3. 字符编码

在计算机中，数值数据是以原码、反码、补码的形式存放的，而计算机不仅要处理数值数据，还要处理大量的非数值数据，即要对文字和符号进行数字化编码。为了进行信息的交换、表示、处理、存储，人们制定了许多国家标准和国际标准的字符编码。以下是几种常见的编码方式。

1) ASCII 码

ASCII 码即美国信息交换标准码（American Standard Code for Information Interchange)。这种代码用一个字节(8 位二进制代码）来表示一个字符，其中低 7 位为字符的 ASCII 值，最高位一般用作校验位。表 1-2 列出了用十六进制数表示的字符的 ASCII 值。

表 1-2　十六进制表示的 ASCII 码表

字符	ASCII	字符	ASCII	字符	ASCII	字符	ASCII
NULL	00	4	34	M	4D	f	66
BEL	07	5	35	N	4E	g	67
LF	0A	6	36	O	4F	h	68
FF	0C	7	37	P	50	i	69
CR	0D	8	38	Q	51	j	6A
SP	20	9	39	R	52	k	6B
!	21	:	3A	S	53	l	6C
"	22	;	3B	T	54	m	6D
#	23	<	3C	U	55	n	6E
$	24	=	3D	V	56	o	6F
%	25	>	3E	W	57	p	70
&	26	?	3F	X	58	q	71
'	27	@	40	Y	59	r	72
(	28	A	41	Z	5A	s	73
)	29	B	42	[	5B	t	74
*	2A	C	43	\	5C	u	75
+	2B	D	44	]	5D	v	76
,	2C	E	45	↑	5E	w	77
-	2D	F	46	←	5F	x	78
.	2E	G	47	'	60	y	79
/	2F	H	48	a	61	z	7A
0	30	I	49	b	62	{	7B
1	31	J	4A	c	63 |	7C
2	32	K	4B	d	64	}	7D
3	33	L	4C	e	65	~	7E

2）BCD码

BCD 码是用二进制编码来表示十进制数的，而每位十进制数用 4 位二进制数来表示（即 0～9用 0000～1001 表示）。BCD 码在存储器中有两种存储方式，即压缩 BCD 码和非压缩 BCD码。

（1）非压缩 BCD 码。其存储方式是一个字节只存一个 BCD码，如图 1－2 所示。

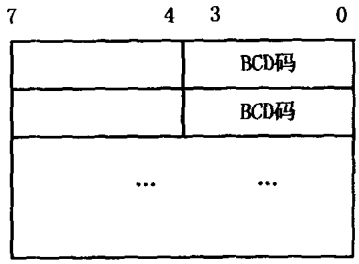


图 1－2　非压缩的 BCD码

（2）压缩的 BCD 码。其存储方式是一个字节存放两个 BCD码，如图 1－3 所示。

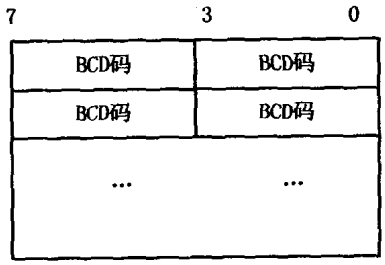


图 1－3　压缩的 BCD码

例如，十进制数 98 用压缩 BCD 码表示为 1001 1000 ，用非压缩 BCD 码表示为 00001001 00001000。

4. 数值及字符的存储

数值及字符编码在计算机内部可以以位、字节、字、双字等形式存在。

1）位（bit）

计算机中最小的一个数据单位是二进制的一个数位，简称位（bit）。位是计算机内部最基本的数据存储单位，一个二进制位可表示两种状态（0或 1）。两个二进制位可表示四种状态（00，01，10，11）。位越多，所表示的状态就越多。

2）字 节（Byte）

一个字节由八个二进制位所组成。字节用 B表示，是计算机中用来表示存储空间大小的最基本的容量单位。一个字节即一个存储单元，每个存储单元都有地址，每个存储单元中存放的数据称为该存储单元的内容。如果用 ××××H 表示某存储单元的地址，则用（××××H）表示该存储单元的内容。例如，（2000H）＝45H，表示地址为 2000H 的单元内存放的内容是 45H。

3）字（Word）

字由若干个字节组成（通常取字节的整数倍），通常意义上的字是由二个字节组成的，即由 16 个二进制位所组成。当把一个字保存在存储器时，一般将高 8 位(一个字节)存储在高地址单元，将低 8 位(一个字节)存储在低地址单元。例如(2000H)＝3456H,则其存放形式如图 1－4所示。

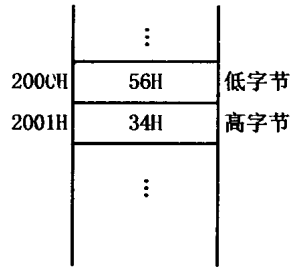


图 1－4　字的存储

4）双字(Dword)

双字是由两个连续存放的相邻的字组成(四个字节)的,其宽度为 32个二进制位。当把一个双字保存到存储器时，一般将高 16 位(一个字)存储在高地址单元,将低 16 位存储在低地址单元。例如存储一个双字 345678ABH ,如图 1－5 所示。

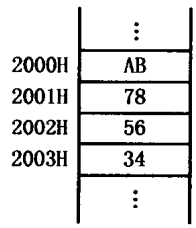


图 1－5　双字的存储

5 字符串

字符串是由字符构成的一个线性数组。 通常每个字符用一个字节表示，但有时每个字符也可用一个字或一个双字来表示。

1.3　8086/8088 的编程结构

1.3.1　8086/8088 的内部结构

8086/8088 CPU 的内部结构如图 1－6 所示。

1. EU 单元的功能和组成

EU 单元的作用主要是进行指令译码和执行。在程序执行过程中， EU 单元所需要的指令代码和数据均由总线接口单元 BIU 通过系统总线存取。

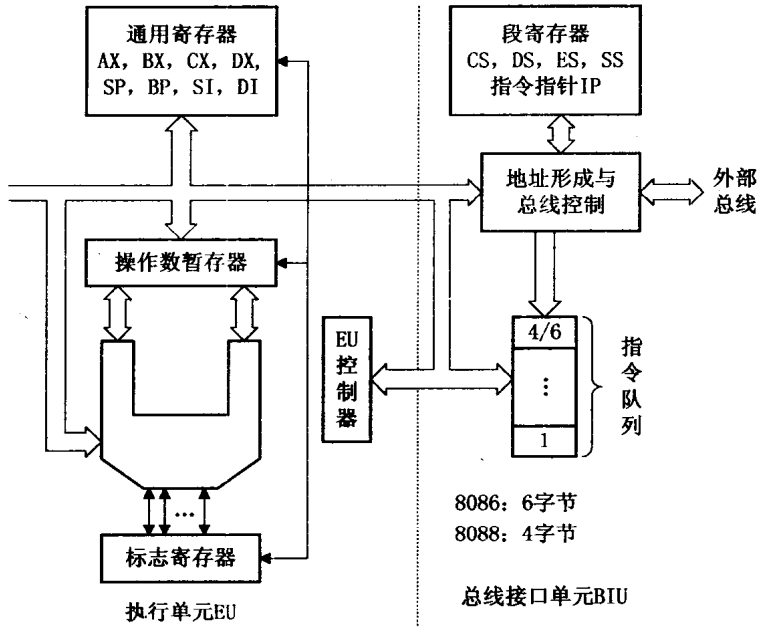


图 1－6　8086/8088 CPU 内部结构

EU 单元主要由以下几部分组成：

（1）一个 16 位的算术、逻辑单元 ALU 可以对参加的 8 位或 16 位二进制数进行算术和逻辑运算。

（2）一个 16 位的状态标志寄存器 FR：用来反映算术或逻辑运算的结果特征或控制 CPU 操作。

（3）8 个 16 位的寄存器：用于存放 8 位或 16 位的地址或数据，可以通过指令访问它们。

（4）EU 控制器：用于对从 BIU 单元获取的指令进行译码及发出相应的内部控制信号。

2. BIU 单元的功能和组成

总线接口单元 BIU 的作用是根据 EU 单元的请求完成与存储器或 I/O 端口之间的数据传送，BIU 单元主要由以下几部分组成：

（1）指令队列寄存器：在程序的执行过程中，用于存放 BIU 单元预取的指令代码，长度为 6 个字节。

（2）4 个 16 位的段基址寄存器 CS、DS、SS 和 ES，一个 16 位的指令指示器 IP 程序执行过程中，这 4 个段寄存器中的段地址与偏移地址一起形成访问存储器的 20 位物理地址。

（3）地址加法器 $\sum$：上述访问存储器的 20 位物理地址可通过这个专用的加法器运算获得。

（4）总线控制电路：将 CPU 的内部总线与外部总线相连，在 CPU 和其他部件之间传送执行指令所需的数据、地址和控制信号。

1.3.2 8086/8088 CPU 的寄存器结构及其用途

从用户角度看，8086/8088 CPU 内部可供程序直接使用的寄存器共有 14 个且为 16 位，其中包括 4 个数据寄存器，4 个指针及变址寄存器。8086/8088 CPU 内部寄存器组如图 1－7所示。

图 1－7　8086/8088 CPU 内部寄存器组

1. 数据寄存器

数据寄存器包括 AX、BX、CX 和 DX 4 个 16 位的通用寄存器组。它们主要用来暂时存放参加运算的操作数及中间结果。这 4 个通用寄存器既可以按字（即 16 位）的形式进行读写，也可以按字节（即 8 位）的形式读写，并分为高字节（高 8 位）和低字节（低 8 位）进行读写。AX、BX、CX、DX 的高低字节分别被命名为 AH、AL，BH、BL，CH、CL，DH、DL。这 4 个寄存器称为通用寄存器，但都有其专用场合。

AX：作为累加器用，常在算术运算中暂存参加运算的数及运算结果。另外，在 I/O 指令中必须使用 AX 寄存器。

BX：在计算存储器地址时，BX 中常存放偏移地址，因此，BX 也称基址寄存器。

CX：在循环指令和串处理指令中，CX 被用作计数器来控制循环的次数和待处理的字节（字）数。

DX：在字操作的乘除指令中要使用 DX 寄存器，另外，在 I/O 指令中须用 DX 来保存端口地址。

2. 指针及变址寄存器

指针及变址寄存器包括 SP、BP、SI 和 DI 4 个 16 位的寄存器。它们可以像数据寄存器一样在运算过程中存放操作数，但只能以字为单位使用。此外，它们常常存放段内寻址时的偏移地址，SP（Stack Pointer）称为堆栈指针寄存器，用来指示栈顶的偏移地址；BP（Base Pointer）称为基址指针寄存器。BP、SP 一般和堆栈段寄存器 SS 一起使用，达到对堆栈区寻址的目的。SI（Source Index，源变址寄存器）、DI（Destination Index，目的变址寄存器）一般

与 DS 联用，达到对数据段寻址的目的。而且，在串操作指令中，SI 和 DI 都可以通过指令被设置为自动增量和自动减量功能。此外，SI 和 DS 联用，DI 和 ES 联用，可达到数据段和附加段寻址的目的。

3. 段寄存器

段寄存器包括CS、DS、SS和 ES 4 个 16 位的段寄存器。关于段寄存器的具体使用，我们将在下一节作详细介绍。

4. 控制寄存器

控制寄存器包括两个寄存器，即 PSW(FR 或 FLAGS)和 IP。

1) PSW(Program Status Word)

PSW 称为程序状态字寄存器，这是一个 16 位寄存器，共设了 9 个标志位。其中，6 位是状态标志位(Status Flags)，用于记录程序中运行结果的状态信息；另外 3 位称为控制标志位(Control Flags)，用于控制 CPU 的操作。用户可通过指令来进行清 0 或置 1，PSW 如下所示：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
				OF	DF	IF	TF	SF	ZF		AF		PF		CF

状态标志位包括以下 6位：

CF(Canny Flag) 进位标志位。在执行算术运算指令后，若结果的最高位(字节操作是 D_7 位，字操作是 D_{15}位)产生进位或借位，则 CF＝1，否则 CF＝0。

PF(Panity Flag)：奇偶标志位。当操作结果中“1”的个数为偶数时，PF＝1，否则 PF＝0。

AF(Auxiliany Flag)：辅助进位标志位。当执行算术运算指令时，如果低 4 位(半个字节)有向高 4 位的进位或借位，则 AF＝1，否则 AF＝0。

ZF(Zero Flag) 零标志位。当运算结果为 0时，ZF＝1，否则 ZF＝0。

SF(Sign Flag) 符号标志位。当运算结果的符号位(字节运算为 D_7 位，字运算为 D_{15} 位)为 1 时，SF＝1，否则 SF＝0。SF 还可以表示符号数运算结果是负数(SF＝1)还是正数(SF＝0)。

OF (Overflow Flag)：溢出标志位。当运算结果超出机器所能表示数的范围时产生溢出，此时 OF＝1，否则 OF＝0。

控制标志位包括以下 3 位。

DF(Direction Flag) 方向标志位。在串处理指令中，DF 决定串处理指令是按增址方向进行还是按减址方向进行的。若 DF＝0，则每次操作后变址寄存器 SI 和 DI 增量，这样使得串处理操作从低地址向高地址进行；DF＝1，则相反。

IF(Interrupt Flag) 中断标志位，当 IF＝1 时，允许 CPU 响应外部可屏蔽中断请求，否则 CPU 不予响应。可用开中断或关中断指令使 IF 置 1 或清 0。有关中断原理将在第 6 章详述。

TF(Trap Flag) 单步标志位。当 TF＝1 时，CPU 每执行完一条即进入内部中断，以便对指令执行结果进行检查；当 TF＝0 时，CPU 处于指令连续执行状态。