第 1 章　概 述

1946 年世界上第一台现代电子计算机在美国诞生，标志着人类进入了计算机时代。计算机是一种能够按照人们预先存放在其中的一系列命令连续高速地进行数据处理的电子机器。能够把人的命令告诉计算机的一套符号系统及其使用规则称为“计算机语言”。到目前为止，计算机语言已经由低级到高级经历了机器语言、汇编语言、高级语言、 第四代语言的发展过程。其中汇编语言是一种能够充分利用计算机硬件特性的低级语言，它与计算机的结构有非常紧密的联系。不同的计算机有各自的汇编语言，本书介绍 Intel 8086/8088 的汇编语言。

1.1　计算机语言是人机交流工具

1.1.1　机器语言

计算机的所有操作都是在指令的控制下进行的。能够直接控制计算机完成指定动作的是机器指令。一条机器指令是一个由 0 和 1 组成的二进制代码序列，不同的机器指令对应的二进制代码序列也各不相同。一条机器指令通常由操作码和操作数两部分构成，操作码在前，操作数在后。

操作码	操作数

操作码部分用来指出这条指令做什么样的操作，是做加法，做减法，还是完成数据传送，亦或是其它的操作；操作数部分或者给出操作对象的值，或者指出操作对象在什么地方。下面的二进制代码序列就是一条 8088 的机器指令：

10000000,00000110,01100100,00000000,00010010

二进制序列中的逗号是为了阅读方便而加上的，并不是机器指令的一部分。这条指令的前 16 位是操作码部分，含义是要求计算机做两个数的加法操作；后 24 位是操作数部分，分别指出第一个加数在内部存储器的编号为 100 的那个字节中，另一加数在指令中，是 18。

对于同样的二进制序列，不同型号的 CPU 对它的“理解”是不一样的。比如上面的那一行指令代码在 8088CPU 看来是要求做加法，换到另一种 CPU 中完全可能被当作是另一种操作，甚至是错误的指令，所以机器代码与机器本身有着紧密的联系。每一种计算机（准

确地说是每一种 CPU）都有自己的一套指令，一种机型的所有机器指令的集合就是它的指令系统。指令系统及其使用规则构成这种计算机的机器语言。选择指令系统中的指令排列起来，可以构成一个指令序列，用以告诉计算机完成一连串的动作，就是一个机器语言程序。

1.1.2 自然语言与汇编语言的对比

机器语言是计算机的“母语”，这是绝大多数人都不懂也很难学会的一种语言，正如前面给出的一条机器指令的例子会令试图学习机器语言的人望而生畏。现实社会中，人们使用汉语、英语、法语等各种不同的自然语言，任何一种自然语言对于当代计算机来说都是无法领会的。因而，人与计算机之间进行信息交流有很大的困难。比较好的解决方法是“找”一种双方都能够也容易学会的语言作为中间媒介，汇编语言以及后来的高级语言、第四代语言都扮演着这样的中介角色。

一个已掌握有自己的母语的人，如果要学习一种新的语言，他该学些什么呢？不妨想像一下中国人学英语的过程：大概所有把英语作为外语来学习的人都是从字母开始的，以后是单词、简单的句子，再发展到用若干连贯的句子描述一件简单的事情，最后是熟练地写英语文章。在学习过程中，从单词的拼写到句子的组织，再到文章的连贯，都会穿插着相应的语法知识。汇编语言既然是一种语言，学习过程也大致如此。表 1.1 中列举了自然语言与汇编语言的对照关系，一方面说明在学习这两种语言时有很多共同之处，另一方面也表明汇编语言需要学习的主要内容。

表 1.1 自然语言与汇编语言的对照

语言 / 对比项目	自然语言（英语）	汇 编 语 言
基本符号	字母表	字母、专用符号
词	单词	保留字、标识符
句	句子	完整的指令、伪指令
段	段落	子程序
章	文章	程序
语法	拼写、句法、文法	指令、子程序、程序格式及使用规则
技巧	句子正确，文理通顺	指令正确，程序精简，易读性好，结构化好

汇编语言是介于自然语言和机器语言之间的一种人机交流媒介。人可以发挥自己的聪明才智学会这一类新的语言，但计算机又如何去“学会”呢？这是利用汇编语言到机器语言的固定翻译机制实现的。编写好的汇编语言程序可以通过一种固定的模式翻译成机器语言。这种翻译工作如果由人来完成同样是非常困难的，而且出错的可能性很大；再说，这

种翻译很枯燥、很机械，倒是非常适合由计算机按人们指定的方法自动进行，因此计算机专家们已编制了一些翻译程序供汇编语言的编程人员使用，这种翻译程序称为“汇编程序”。

1.1.3 汇编程序和连接程序

汇编程序是一种计算机软件，属于软件分类中的系统软件部分，它能够把人们编写的汇编语言程序（称为源程序，一般以 ASM 作为文件扩展名）翻译成机器语言，这种翻译操作称为“汇编”。由于不同的计算机有不同的机器语言，因而也需要有不同的翻译器——汇编程序，MASM.EXE 是一种专门用于把 Intel 8086/8088 的汇编语言源程序翻译成相应的机器语言程序的翻译器，是 8086/8088 汇编语言编程人员必备的基本工具之一。

汇编程序还具有语法检查的功能，交给汇编程序进行处理的源程序在翻译之前都必须经过语法检查这一关。如果汇编程序发现源程序中有违背汇编语言语法的地方，将不进行翻译工作，而是指出错误的位置以及类型。从这个角度来说，汇编程序决定了汇编语言的语法，不同厂家、不同版本的汇编程序在语法规定上可能有细微的差别，本书后面章节中都以 MicroSoft 公司的 MASM.EXE V5.0 作为汇编程序，如果读者所使用的汇编程序不是这个版本，请参照有关说明书。

汇编程序翻译的结果已具备机器语言的形式，称为“目标程序”，一般以 OBJ 作为文件扩展名。但是，目标程序还不能直接交给计算机去执行，它还需要通过连接程序(LINK.EXE)的装配才具备可执行的形式，装配结果称为“执行文件”，一般以 EXE 作为文件扩展名。另一方面，连接程序还具有把多个目标程序装配在一起的功能，或者把目标程序与预先编写好的一些放在子程序库中的子程序连接在一起，构成较大的执行文件。汇编语言源程序、汇编程序、目标程序、连接程序、执行文件的关系如图 1.1 所示。

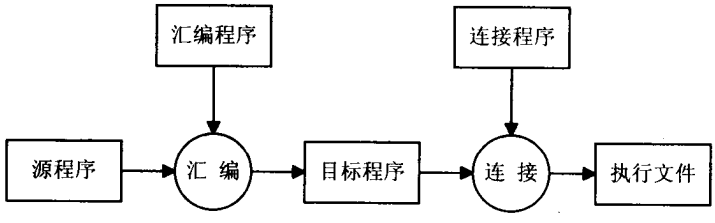

图 1.1 由汇编语言源程序到执行文件的处理过程

1.1.4 汇编语言的构成

汇编语言是较早发明的一种介于自然语言与机器语言之间的程序设计语言。为了使汇编语言到机器语言的翻译比较简单，汇编语言用大量的语法规则对从指令到程序的书写加以限制。与后来的高级语言、第四代语言相比，汇编语言更接近于机器语言，用汇编语言编写的源程序还保留了很多机器语言的影子。比如：

机器指令中的操作码部分在汇编语言中用与该指令的功能相关的一个符号表示，例如加法指令就用 ADD 表示，数据传送用 MOV 表示，这类符号称为“助记符”。

对于一个放在内存当中的操作数，如果要求编程人员记住每一个操作数在内存中的存

放位置则是一个巨大的负担。在汇编语言中，减轻这种负担的方式是用变量存放操作数，程序员只要记住变量的名字即可。

跳转是程序设计中不可避免的一个问题。在机器语言中，跳转的目的地是用指令所在的位置（即在内存的哪一个字节）来表示的，而汇编语言中的跳转则是在目的地做一个称为“标号”的标记。

除了与机器语言有直接对应关系的助记符、变量、标号外，为了能让汇编程序正确地完成翻译工作，必须要告诉汇编程序变量需要占据多少字节的内存、程序到何处结束、整个程序的第一条指令在什么地方等等问题。因此，源程序中应该有一些告诉汇编程序如何进行翻译操作的“说明”，这类说明在翻译结果中并没有对应的机器代码，所以称为“伪指令”。

指令助记符、数据和存放数据的变量、标号、伪指令以及相应的使用规则构成了汇编语言的全部内容。

1.1.5 汇编语言的特点

与机器语言相比，汇编语言易于理解和记忆，编写的源程序可读性较强。汇编语言中还提供了一些分工协作并相互通讯的方法，可以实现初步的结构化编程。源程序翻译成机器语言后的执行文件在存储空间、执行速度方面与机器语言编写的程序大致相当。

高级语言和第四代语言在科学计算、事务处理等方面比汇编语言有巨大的优势，但用高级语言编写的程序，在翻译成机器语言后，程序代码冗长，占用存储空间大，执行速度慢。如果用高级语言来编写接口控制、设备通讯等方面的程序则不太合适，相反这样的情况下汇编语言更容易发挥其长处：目标程序或执行文件简短，执行速度快，效率高，特别是汇编语言能直接控制计算机的内存和外设，这些特点是高级语言和第四代语言望尘莫及的。

可见高级语言、第四代语言适合于编写应用软件，而对于系统软件，尤其是涉及内存管理、硬件控制方面问题时汇编语言则比较合适。可以说，汇编语言程序设计是从事计算机研究与应用的重要手段。

1.2 预备知识

1.2.1 数制及其转换

人们在日常生活中每天都要与数打交道，我们通常书写的数是十进制数，而计算机内部使用的却是由 0 和 1 两种符号构成的二进制数。二进制数在书写时显得过于冗长、麻烦，所以在与计算机打交道时还会经常使用八进制数和十六进制数。同一个数值可以用不同的数制写出，不同数制之间可以相互转换。汇编语言中只有整数可以直接处理，所以，在此仅讨论整数在不同数制间的转换方法。

1.2.1.1 数制

任何一个数制都涉及下面三个问题：

1．计数符号

这是用于书写数值的各个符号，所有计数符号构成的集合称作数符集。 k 进制的数符集中必然包含 k 个符号。比如：

二进制的数符集中有两个符号： 0 和 1；

八进制的数符集中有 8 个符号：0, 1, 2, 3, 4, 5, 6, 7；

十进制的数符集中有 10 个符号：0, 1, 2, 3, 4, 5, 6, 7, 8, 9；

十六进制的数符集中有 16 个符号：0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F。

显然，任意进制的各个计数符号是有顺序的，二进制、八进制和十进制的每个计数符号就是它的序号值，十六进制的后 6 个计数符号 A、B、C、D、E、F 的序号值依次是 10、11、12、13、14、15。

2．基数和权

如果把用 k 进制书写的一个整数从右往左依次记作第 0 位、第 1 位、…、第 n 位，则第 i 位上的数符 a_i 所代表的含义是 $a_i \times k^i$。在此，我们把 k 称为一个数制的基数，而把 k^i 称为 k 进制数第 i 位的权。

3．计数规则

简单地说，就是“逢 k 进 1，借 1 当 k”。

1.2.1.2 把非十进制数转换成十进制数

从上面关于数制的描述中不难看出，用 k 进制书写的一个数，其含义是：

$$a_n a_{n-1} \ldots a_1 a_0 = a_n \times k^n + a_{n-1} \times k^{n-1} + \ldots + a_1 \times k^1 + a_0 \times k^0 \qquad (1.1)$$

依照计数符号的次序不难把 k 进制的一个符号 a_i 转换成一个十进制数，然后把式 1.1 的右边按照十进制运算规则进行计算，求得的结果就是相应的十进制数。

1.2.1.3 把十进制数转换成非十进制数的通用方法

从式（1.1）中不难看出，如果能把一个十进制数写成式（1.1）右边的形式，当缺少某一权值项 k^j 时，就在它应在的位置写上 $0 \times k^j$，并且保证每个权值项 k^i 前面的系数 a_i 都是小于 k 的非负整数，就可以先把每个系数转换成 k 进制数相应的数符，然后从最大的一个权起，按照权从大到小的次序，依次记录下各系数对应的数符，即可得到相应的 k 进制数。

【例 1.1】把十进制数 15370 转换成十六进制数。

【解】十进制数 15370 可以写成如下形式：

$$15370 = 3 \times 16^3 + 12 \times 16^2 + 10$$

按十六进制数符的排列次序知：十进制数 12 对应十六进制数数符 C，十进制数 10 对应十六进制数数符 A，对于式中缺少的权 16^1，以 0 作为它前面的系数，而最后一项 10 可看作 10×16^0。所以与十进制数 15370 相等的十六进制数是 3C0A。

1.2.1.4　把十进制数转换成二进制数

从式（1.1）中也不难推导出把十进制整数转换成二进制数的“除 2 取余”法，并且该方法可以推广到十进制数到八进制数或十六进制数的转换中，很多计算机教材上已有详细的描述，在此仅举两例。

【例 1.2】把十进制数 137 转换成二进制数。

【解】

```
2 | 137 …… 1
  2 | 68 …… 0
  2 | 34 …… 0
  2 | 17 …… 1
    2 | 8 …… 0
    2 | 4 …… 0
    2 | 2 …… 0
    2 | 1 …… 1
      0
```

所以，与十进制数 137 等值的二进制数是 10001001。

【例 1.3】把十进制数 1234 转换成十六进制数。

【解】

```
16 | 1234 …… 2
   16 | 77 …… 13 = D
      16 | 4 …… 4
         0
```

所以，与十进制数 1234 等值的十六进制数是 4D2。

.2.1.5　二进制数与十六进制数的相互转换

在十进制以外的其它数制之间，如果要把一个 k 进制数转换成 r 进制数，一般是以十进制作为中间媒介。先把 k 进制数转换成十进制数，再把等值的十进制数转换成 r 进制数。但是，在二进制数和十六进制数之间存在非常简便的转换方法。

二进制数转换成十六进制数比较容易，具体方法是：

(1) 把二进制数自右向左每 4 位分成一组，最左边不足 4 位的左补 0；

(2)把每组 4 位的二进制数转换成 1 位十六进制数；

(3) 按从左到右的次序写出转换结果。

【例 1.4】把二进制数 101100110101111 转换成十六进制数。

【解】分组，左补一个 0:　0101，1001，1010，1111

转换：　　5　　9　　A　　F

所以，等值的十六进制数是 59AF。

十六进制数转换成二进制数就更简单，只需从左到右把每位十六进制数符写成相应的 4 位二进制数，不足 4 位则左补 0 凑齐 4 位，并把结果写在一起即可。

【例 1.5】把十六进制数 3BD 转换成二进制数。

【解】等值的二进制数是 001110111101，去掉最左边没有意义的 0，得 1110111101。

表 1.2 中列出了 0～15 之间的十进制数在二进制、八进制和十六进制下的对应值。为了加快数制转换的速度，这张表中的内容应该熟记于心。二进制、十六进制下多位数的加减法也应作为基本技能熟练掌握。

表 1.2　0～15 在各数制下的表示

十进制	二进制	八进制	十六进制	十进制	二进制	八进制	十六进制
0	0000	00	0	8	1000	10	8
1	0001	01	1	9	1001	11	9
2	0010	02	2	10	1010	12	A
3	0011	03	3	11	1011	13	B
4	0100	04	4	12	1100	14	C
5	0101	05	5	13	1101	15	D
6	0110	06	6	14	1110	16	E
7	0111	07	7	15	1111	17	F

1.2.1.6　数的书写方法

由于计算机中经常使用的数制有二进制、八进制、十进制和十六进制，所以写出一个数有时会无法分辨到底是使用的哪一种数制，比如 1011。为此，需要在书写形式上加以区分。一般来说，书写方法有三种：下标法、前导法、后缀法。

1．下标法

这是日常书写的一种常用方法，是在写完表示数值的符号序列之后，在其右下角以角标的形式写上所使用的数制的基数，比如：

1001_2 表示二进制数 1001；

377_8 表示八进制数 377；

377_{16} 表示十六进制数 377。

对于日常使用的十进制数，可以按上述方法加下标表示，也可以不加；反之，如果一个数在书写时没有加上任何标记，则表示是十进制数。

2．前导法

有些程序设计语言中使用前导法，即在表示数值的数符序列前面加上特定的前导符号以区分不同的数制。比如，在 C 语言中就用 0x 作为十六进制数的前导符号，在 Turbo Pascal 中用 $作为十六进制数的前导符号。十六进制数 123 在 C 语言中写作 0x123，在 Turbo Pascal 中则写作 $123。

3．后缀法

与前导法相反的一种做法是，在写完表示数值的数符序列后，加上一个特殊的符号表示不同的数制。汇编语言中就采取这种做法。二进制、八进制、十进制和十六进制的后缀符号分别是 B、Q、D 和 H 比如：

1011B 表示二进制数 1011；

1011H 表示十六进制数 1011。

特别的是，如果一个十六进制数以字母数符开头，会与后面章节中所说的变量名、标号等标识符相混淆，为了区分这样的情况，在所有字母数符开头的十六进制数的前面再加上一个 0。我们知道，在一个整数的最高位的前面加 0是不影响数值本身的。比如，十六进制数 ABC 必须写成 0ABCH。

当然，在不加任何后缀的情况下书写的一个数是十进制的。

本书后面的章节中都采用后缀法，而在已有适当的文字清楚地说明数制，不至于造成混淆的情况下，则省略后缀和十六进制前导的 0。

1.2.2 无符号数与带符号数

负数是程序设计中必须面对的问题。但是，计算机内部没有正负号，只有 0 和 1。计 算机内部区分正负数的方法是，在存放数的若干个二进制位（一般是 8位、16 位或 32 位)中，用最高位作为符号位，这一位为 1表示该数是负数，而这一位为 0则表示非负。

但是，计算机中并没有严格规定一定要把最高位用作符号位，也就是说，需要处理的数据允许有负数时，就用最高位表示数的正负情况，当处理的数据不可能有负数时，就用最高位与其它位一起存放数值。最高位上的 0 或者 1 究竟作何用途，从存储的数据本身是无法区分的。比如，在计算机内部以二进制存放的数据 10011100，如果是在一个人数统计的程序中用来表示人数就应该当作 156，而在数值计算的程序中用以表示两个数相差多少时就应该当作一个负值看待。这一点只有根据程序本身所完成的功能以及该数所表示的具体含义加以区分。

我们把最高位用作表示符号的数称为“带符号数”或“有符号数”，而把最高位用来表示数值的数称为“无符号数”。这两种数据是汇编语言能够直接处理的两种数据。存放在计算机内部的数是否带符号是人为看待的问题，而不是数据本身所具备的属性。

1.2.3 原码和补码

原码和补码是表示带符号数的两种最常用的方法，在现代计算机中，数据都是用补码表示的。

1..2.3.1 原码

用原码表示带符号数，首先要确定用以表示数据的一进制位数，一般是用 8 位或 16 位，把其中的最高位用来表示数据的正负符号，剩余位表示该数据的绝对值。比如，用 8 位二进制表示 +12 就是 00001100B，而 -12 是 10001100B，用 16 位二进制表示 +1024 是 0000010000000000B，而 -1024 则是 1000010000000000B。

原码表示法的特点是简便、直观，懂得数制转换的人可以很快计算出其表示的数在十进制中究竟是多少，它的缺点之一是 0 的表示有两种，即 +0 和 -0，这对计算机来说可不是件好事。另一缺点是运算比较麻烦。比如两个原码表示的数据相加，首先需要判断两数的符号位，以决定到底是做加法还是做减法，然后用它们的绝对值进行计算；还需要判断计算结果的正负情况，最后把计算结果在最高位上填上正确的符号位。这种做法尽管在计

算机上可以实现，但还有更好的方法。

1.2.3.2 补码

补码是在原码的基础之上，为简化运算而发展出来的另一种表示带符号二进制数的方法，具体方法是：

(1) 确定表示数据的二进制位数，通常是 8 位或 16 位；

(2) 如果被表示的数据是非负的，则用其原码表示；

(3) 如果被表示的数据是负数，则把该数的绝对值表示成 8 位或 16 位二进制数，然后对每一位取反，即原位上是 0 就改写成 1，原位上是 1 则改写成 0，再把取反后的结果加 1。

【例 1.6】把 15 和 -27 转换成 8 位补码表示，把 345 和 -32768 转换成 16 位补码表示。

【解】按照上述补码转换规则，有：

(1) 因为 15>0，所以直接用 8 位原码表示，即

$$15 = 00001111B = 0FH$$

(2) 因为 -27<0，所以先把其绝对值 27 转换成 8 位二进制数，再取反加 1，即

-27 => 27 的 8 位二进制表示 00011011B

=> 各位取反，得 11100100B

=> 再加 1 得 11100101B = 0E5H

(3) 因为 345>0，所以直接用 16 位原码表示，即

$$345 = 0000000101011001B = 0159H$$

(4) 因为 -32768<0 所以先把其绝对值 32768 转换成 16 位二进制数，再取反加 1，即

-32768 => 32768 的 16 位二进制表示 1000000000000000B

=> 各位取反，得 0111111111111111B

=> 再加 1，得 1000000000000000B = 8000H

用补码表示二进制数的好处在于，对于两个带符号数进行加法或减法运算时，符号位直接参与运算，不需要判断符号，而计算结果的最高位仍然表示符号。现在的电子计算机中都使用补码表示带符号数，8086/8088 当然也不例外。

【例 1.7】用 8 位补码表示 15 和 -27，并计算两数的和与差。

【解】利用例 1.6 的结果，有：

```
  00001111  ……  15
+ 11100101  ……  -27
------------------
  11110100  ……  -12

  00001111  ……  15
- 11100101  ……  -27
------------------
  00101010  ……  42
```

那么，计算结果 11110100B 又是如何还原成十进制的 -12 的呢？这也不难，从最高位的 1 知结果是负数，然后对每一位取反，得 00001011B，再加 1，得 00001100B。这就是计

算结果的绝对值，转换成十进制就是 12，不要忘记原数据的最高位的 1 表示它是负的，所以结果是 -12。当然，如果一个补码表示的数最高位是 0，就如同例 1.7 中两数相减的结果，这表示结果是非负数，则直接转换成相应的十进制数即可得 42。

细心的读者不难发现，例 1.7 中做减法时其实是不够减的。这时被减数的最高位需要再向前借位，即向补码表示的 8 位之外去借。这在计算机中是允许的，并有一定的机制记录减法是否出现了这种向外的借位，详见第 4 章 CF 标志位的设置情况。

另外，一个 8 位补码与一个 16 位补码表示的二进制数是不能进行加减法运算的，这时需要把 8 位补码表示的二进制数转换成 16 位补码表示形式，方法是 如果 8 位补码是非负的，则在其前面加 8 个 0，如果是负数，则在其前面加 8 个 1。类似的方法可以应用于把 16 位补码转换成 32 位补码。这种把 8 位补码转换成 16 位补码，或把 16 位补码转换成 32 位补码的操作称为符号扩展。

1.2.3.3 求补函数

上一节中两次提到了把二进制数按每一位的情况取反，再加 1，这是一种操作，或者看作一种运算，运算的结果仍然是一个二进制数，这种处理刚好符合数学上一元函数的特点。

【定义】设 x 是 8 位二进制整数，把对 x 进行如下处理记做 $N_8(x)$：对二进制表示的 x 的各位取反，再加 1，加 1 时如果最高位向前有进位则忽略。 $N_8(x)$称作求补函数。

类似地，可以定义 16 位求补函数 $N_{16}(x)$。并且，在不考虑位数时，把求补函数统一写作 N(x)。

上述定义中并没有涉及二进制数的最高位是否表示符号，也就是说，不论一个数是无符号数还是带符号数，都可以作为 N(x)的处理对象，而处理结果也不考虑其最高位的含义，N(x) 本身只代表一种处理方法。当然，由于数制间是可转换的， x 也可以用各种数制写出。

不难看出，若 y 是负数，则 y 的补码表示就是 N(|y|)。

求补函数 N(x)还具有两个很重要的性质：

【性质 1】把 x 当作 8 位或 16 位无符号数看待，则有

$$N_8(x) = 256-x = 2^8-x$$

$$N_{16}(x) = 65536-x = 2^{16}-x$$

【证明】对于 8 位求补函数 $N_8(x)$，因为

$$256 = 255+1 = 11111111B+1$$

所以

$$256-x = (11111111B-x)+1$$

设 $x=x_7x_6x_5x_4x_3x_2x_1x_0$，即把 x 的第 i 位表示为一个二进制数符 x_i（i=0,1,...,7），记 $11111111B-x_7x_6x_5x_4x_3x_2x_1x_0$ 的结果为 $y_7y_6y_5y_4y_3y_2y_1y_0$，根据二进制数减法运算规则有 $y_i=\overline{x_i}$，即 y_i 是二进制数符集中与 x_i 不同的另一个符号，所以(11111111B-x) 的值就等于对 x 按位取反的结果。

类似地可以证明 $N_{16}(x)$的情况。

这一性质实际上告诉我们另一种求补码的方法。从补码本身还可以推出：记 x′ 是把 x 当作有符号数时的对应值，则

$$N(x) = 0-x'$$

【性质 2】N(N(x))=x。

这是从性质 1 可以简单推导出的结论。

性质 2 说明，对 x 连续两次求补，则结果可以还原。 1.2.2.2 节中所描述的一个补码表示的负数转换成日常表示的方法，正是利用了这个性质。

1.2.4 逻辑运算

能够进行逻辑运算是电子计算机的一大特色，相信读者已从有关课程中对此有所了解，在此仅列出与、或、非和异或 4 种逻辑运算的基本运算规则。

用二进制的 1 表示逻辑值“真”，用 0 表示逻辑值“假”，用“×”表示逻辑与运算，用“+”表示逻辑或运算，用“—”表示逻辑非运算，用“⊙”表示逻辑异或运算，则

$$
\begin{aligned}
&0\times 0 = 0\\
&0\times 1 = 0\\
&1\times 0 = 0\\
&1\times 1 = 1\\
&0+0 = 0\\
&0+1 = 1\\
&1+0 = 1\\
&1+1 = 1\\
&\overline{0} = 1\\
&\overline{1} = 0\\
&0\odot 0 = 0\\
&0\odot 1 = 1\\
&1\odot 0 = 1\\
&1\odot 1 = 0
\end{aligned}
$$

1.2.5 8086/8088 支持的数据类型及其内部表示

8086/8088 机器指令中可以直接处理的数据类型只有三种，分别是 8 位二进制数的字节型、16 位的字型和 32 位的双字型。但是根据其具体含义及写法的不同，在汇编语言中又有多种变化。表 1.3 列出了几种常用的数据类型及其表示范围。

表 1.3 各种数据类型的有效范围

数据类型	有效范围
字节型无符号数	0~255
字型无符号数	0~65535
双字型无符号数	0~4294967295
字节型带符号数	-128~+127
字型带符号数	-32768~+32767
双字型带符号数	-2147483648~2147483647

在 8086/8088 汇编语言中，数据的书写方法比较丰富，不同的书写形式在计算机内部可能有相同的存储结果，而存储器中的同一个数据也可能因为使用的方法不同而有不同的外部表现形式。所以内部存储形式相同的数据之间不存在类型转换问题，而是一个使用方法的问题。

【例 1.8】把下面几种形式书写的数据转换成字节型内部存储形式。

0E3H, 227　-29　-11101B

【解】经转换，内部形式完全相同，都是 11100011B。

汇编语言中数据还有一种字符形式的写法，如'A'、'0'等。在字符的两边加单引号，其含义是以该字符的 ASCII 值作为数据，所以'A'相当于 41H，而'0'相当于 30H。至于这样的数据究竟是字节型还是字型则需要看使用的场合，从数据本身是无法确定的。

【例 1.9】设存储器中有一个字节型数据 01100010B，试说明该数据在不同的使用环境下几种可能的含义。

【解】作为字符的 ASCII 值用于向屏幕输出，该数表示字符'b';

作为无符号数，该数表示 98;

作为带符号数，该数表示+98。

数据是计算机处理的对象，数据的各种书写方法为编程提供了便利，而同一数据可以具有不同的含义又给学习汇编语言造成一定的困难。

本 章 要 点

汇编语言是一种低级程序设计语言，它由指令助记符、数据和变量、标号、伪指令以及语法规则构成。指令助记符对应于机器指令中的操作码，数据和变量对应于操作数，标号表明指令所在的位置，伪指令指出翻译成机器语言时的处理方法。汇编语言编写的源程序经过汇编和连接处理后形成可执行的机器语言程序文件。汇编语言适合于编写与计算机结构联系紧密、需要直接控制硬件设备的程序，用汇编语言编写的程序执行速度快、效率高。学习汇编语言与学习自然语言有很多相似之处，可以借鉴后者的学习方法。

汇编语言中的数据有无符号数、带符号数、字符等形式，不同表现形式的数据可以在计算机内部的存储形式相同；根据使用方法不同，一个在计算机内存放的数据也可以有不同的含义。

计算机内部使用二进制，汇编语言中则可以用各种数制书写数据，不同数制之间存在固定的转换关系。补码是一种表示带符号数的编码方法，用补码表示的数据在进行加减法运算时比较方便。

习　题　一

11　解释下列名词：

汇编语言　汇编程序　汇编　目标程序　机器语言　伪指令　助记符

12　把下列十进制数转换成二进制数。

100　　49　　23　　67

1.3 把下列十进制数转换成十六进制数。

1111　　5493　　32074　　9843

1.4 把下列数据转换成十进制数。

10111B　　365Q　　327H　　128H

1.5 把下列二进制数转换成十六进制数。

10110011111101B　　1101101111101111B

1110010001001B　　1110111101011001B

1.6 把下列十六进制数转换成二进制数。

0F8H　　37CH　　1BDH　　7AEH

1.7 把下列十进制数转换成 8 位二进制补码表示。

+33　　+100　　－33　　－100

1.8 把下列十进制数转换成 16 位二进制补码表示。

+330　　+8379　　－5428　　－3504

1.9 用十六进制完成下列计算。

21CH + 0ABH　　7A0BH + 6F3H

6B2CH－9AFH　　289EH－0BFAH

1.10 用 8 位补码完成下列计算，并把结果转换回十进制数。

(+ 43) + (+ 27)　　(+ 25) + (－ 49)

(－ 92) + (+ 33)　　(－ 76) + (－ 18)

(+ 43) － (+ 27)　　(+ 25) － (－ 49)

(－ 92) － (+ 33)　　(－ 76) － (－ 18)

1.11 用 16 位补码完成下列计算，并用十进制写出相应的带符号数。

(－ 432) + (+ 1000)　　(－ 789) + (－ 12345)

(+ 3290) － (－ 742)　　(－ 12078) － (+ 934)

第 2 章微型计算机的内部结构

汇编语言是一种可以直接控制计算机硬件设备的计算机语言，掌握一些计算机硬件知识是学习汇编语言的必要前提，其中最重要的是了解计算机各部件的基本结构和逻辑连接关系，尤其是核心部件 CPU 的内部结构。

2.1 微型计算机的构成

电子计算机由中央处理器(简称 CPU)、内部存储器和输入输出设备(简称 I/O 设备)三大部件构成。CPU 是计算机的核心，人们交给计算机的命令总是由 CPU 解释并执行，CPU 还负责产生各种控制信号，令各部件协调工作，使整个系统构成一个有机整体。内部存储器（简称内存）是高速的数据存储部件，它与 CPU 一起构成了计算机的主机部分。输入输出设备用来完成计算机内部与系统外部的数据交换。 I/O 设备中，用来把人的控制命令、外部采集到的数据传递到计算机内部的设备称为输入设备，典型的输入设备有键盘、鼠标、扫描仪等；把计算机内部数据送出来变成人可以理解的形式，或者变成控制工业机器的信号的设备称为输出设备，典型的输出设备有显示器、打印机、绘图仪等；用于长期保存大量数据的设备是外部存储器（简称外存），磁盘、光盘是典型的外存。

CPU 的规模是划分计算机档次的一个决定性指标，我们把集成在一个芯片上的 CPU 又称为微处理器，简称 MPU，8088 和 8086 芯片是 Intel 公司较早生产的两种同档次的 MPU，在此之后，该公司又陆续推出人们熟知的 80286、80386、80486 和 Pentium 一系列性能越来越好的 MPU。由一个 MPU 芯片为核心而构成的计算机称为微型计算机或者微电脑。由 8088或 8086 为 MPU 构成的微机就是 20 世纪 80 年代初出现并流行一时的 PC/XT 型个人电脑，当今正在走入家庭的微电脑大多数则是以 Pentium 作为主芯片。

计算机的协调工作是靠三大部件之间有机连接并相互配合来实现的。三大部件之间的逻辑连接关系可以用图 2.1 来表示。

图 2.1 中的双线是各部件之间进行数据传递的公共通道，是计算机的系统总线。 MPU 和内存直接连接在系统总线上。 D_1、D_2、…、 D_n表示若干个外部设备，各个外设性能各异，控制外设的信号也各不相同。为了分担 MPU 的工作，在每个外设与总线之间往往需要一个称为“ 接口 ”的连接部件。

通常情况下，MPU 掌握总线的控制权，可以选择连接在总线上的另外一个部件与之进行数据传递，数据传递的方向也由 MPU 决定。

系统总线从功能上划分为数据总线、地址总线和控制总线三部分，每一部分又包含多

根信号线。

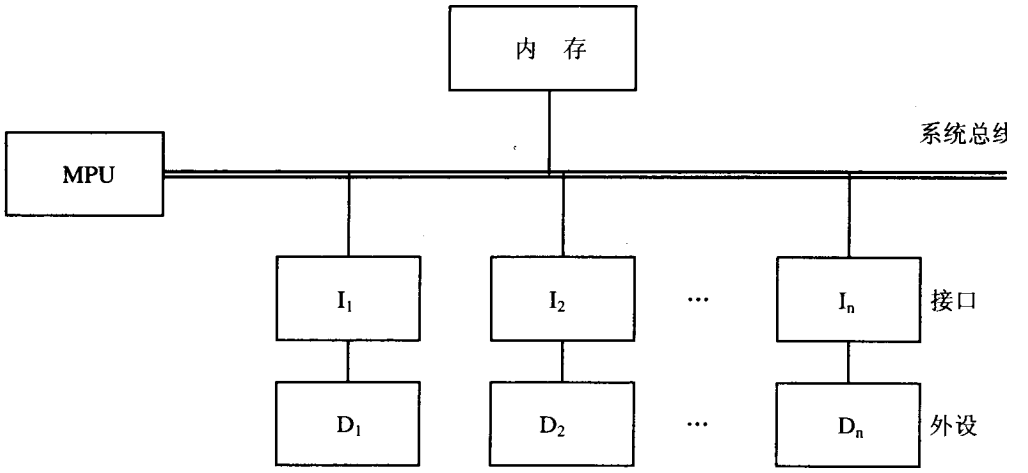

图 2.1 微型计算机各部件的连接

2.2 8086/8088 MPU 的内部结构

微型计算机的核心部件 MPU 从总体功能上讲由三部分构成：运算器、控制器、寄存器。运算器完成算术运算和逻辑运算，寄存器临时存放数据，控制器负责整个系统的协调。8086/8088 微处理器的内部结构如图 2.2 所示。

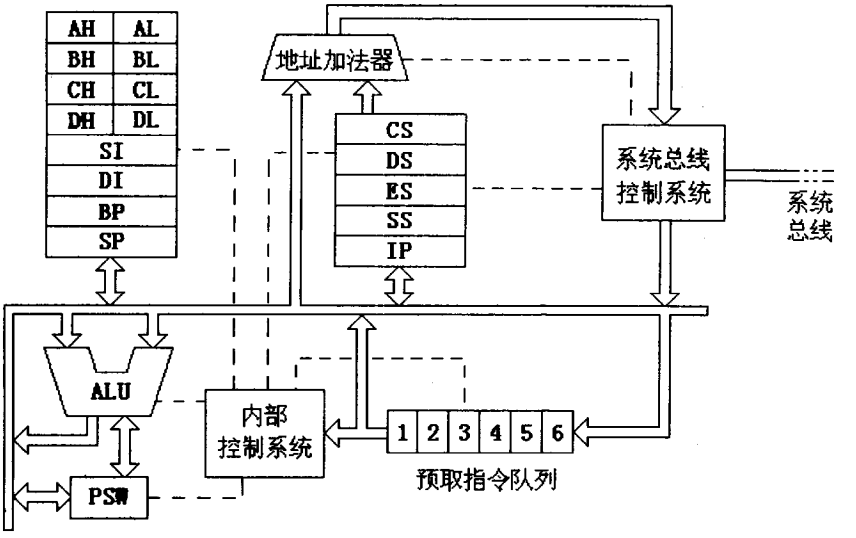

图 2.2 8086/8088 MPU 的内部构成

图 2.2 中宽线表示由多根线组成的 MPU 内部数据总线，可同时传送多位数据；箭头表示数据传递的方向；虚线是内部控制线；内部控制系统和系统总线控制系统构成 MPU 的控制器。

2.2.1 运算器

运算器(Arithmatical and Logical Unit)是 CPU 中完成算术运算和逻辑运算的部件，它受内部控制系统发出的信号控制，决定究竟做哪一种操作。操作中涉及的数据可以由各部件通过内部总线送来，运算结果一方面通过内总线送到某一部件，另一方面运算结果的特征还送往标志寄存器。

具体地说，8086/8088 的运算器可以完成以下操作：

(1) 加减法。两个字节型数据相加减，或者两个字型数据相加减，计算结果的类型与操作数类型相同。

(2) 乘法。两个字节型数据相乘，得到字型的乘积；或者两个字型数据相乘，得到双字型的乘积。

(3) 除法。字型的被除数除以字节型的除数，计算结果的商和余数都是字节型；或者双字型的被除数除以字型的除数，计算结果的商和余数都是字型。

(4) 逻辑与、逻辑或、逻辑异或。两个字节型数据进行对应位的逻辑运算，结果仍然是字节型数据；或者两个字型数据运算，结果仍然是字型。

(5) 逻辑非。对一个字节型或字型数据的每个二进制位进行逻辑非运算。

2.2.2 通用寄存器组

寄存器是 CPU 内部临时存放数据的部件，它的存取速度比内存更快，可以把数据通过内部总线送往运算器进行运算，或者接收来自运算器的结果。充分利用 CPU 的内部寄存器可以加快程序的执行速度。8086/8088 有 8 个 16 位的字型通用寄存器，各个寄存器都有自己特定的功能，有些指令规定使用这些功能，而不执行这类指令时，这些寄存器都可以用来临时存放数据。

各寄存器的名称及特定功能是：

(1) AX——累加器。这是算术运算的主要寄存器，用于存放操作数或运算结果；另外，外设操作指令都使用 AX 存放数据。

(2) BX——基地址寄存器。用于存放操作数的偏移地址。

(3) CX——计数寄存器。用于循环指令、移位指令及串操作指令的计数控制。

(4) DX——数据寄存器。在乘除法运算中，与 AX 组合在一起存放双字型数据；DX 还用于存放外设操作的端口地址。

(5) SI—— 源变址寄存器。作为串操作的源操作数地址指针，也可用于存放操作数的偏移地址。

(6) DI—— 目的变址寄存器。作为串操作的目的操作数地址指针，也可用于存放操作数的偏移地址。

(7) BP——栈基地址寄存器。用于存放堆栈中的操作数的偏移地址。

(8) SP——栈顶指针。用于记载栈顶的当前位置（偏移地址）。

上述每个寄存器都是 16 位的，可以存放一个字型数据。但 8086/8088 还支持 8 位数据

操作，因而需要有临时存放 8 位二进制数的字节型寄存器。通用寄存器中的 AX、BX、CX 和 DX，每一个又可以拆成两个 8 位寄存器使用，代号分别是 AH、AL、BH、BL、CH、CL、DH、DL。往 AX 中放一个数据将对 AH 和 AL 都有影响，反之，对 AH 的赋值将改变 AX 的高 8 位部分而不影响 AL，对 AL 的赋值会改变 AX 的低 8 位但不影响 AH。

以上各寄存器的说明中涉及一些后续章节中将要讲述的专用术语，读者目前不必记忆每个寄存器的特定用法，只需知道其代号和名称即可，对下面的段寄存器和标志寄存器等部件亦然。

2.2.3 标志寄存器

标志寄存器又称作程序状态字（Program Status Word，简记作 PSW），共 16 位，一般把每一位分别使用，8086/8088 使用其中的 9 位，用于存放当前程序执行的状况和运算结果的特征，各标志位的分布如图 2.3 所示。

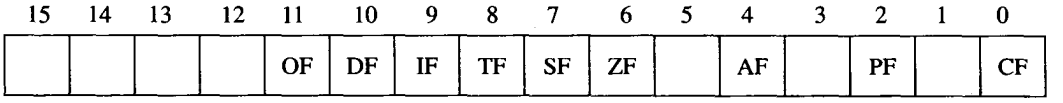

图 2.3 标志寄存器中各标志位的分布

在 8086/8088 中使用的 9 个标志位可分为两组：一组为条件标志位，记载算术运算或者逻辑运算结果的特征，包括 CF、ZF、SF、OF、PF、AF 等 6 个标志位；另一组是控制标志位，记载 MPU 当前工作状态，包括 TF、IF、DF 等 3 个标志位。

1. 条件标志位

（1）CF——进位标志。记录加法运算的进位值，或者减法运算的借位值。

（2）ZF——零标志。记录运算结果是否为 0。

（3）SF——符号标志。记录运算结果的正负情况。

（4）OF——溢出标志。记录运算结果是否超出有符号数的表示范围。

（5）PF——奇偶标志。记录字节型运算结果中 1 的个数的奇偶性。

（6）AF——辅助进位标志。记录加／减法运算中最后 4 位向前有无进／借位。

其中前 4 个标志位至关重要，与汇编语言中分支和循环结构程序设计有密切的联系，这 4 个标志位的具体设定和使用方法见第 4 章；后两个标志位对于汇编语言的普通编程并没有很大的价值，后面不再详细介绍。

2. 控制标志位

（1）TF——单步中断允许标志。又称跟踪标志，表示系统当前是否允许单步中断。

（2）IF——外中断屏蔽标志。表示系统当前是否允许可屏蔽外中断。

（3）DF——方向标志。表示串操作按增量方向还是按减量方向进行。

这 3 个标志位与 8086/8088 的运行状态密切相关，TF 和 IF 涉及中断系统，将在第 8 章中介绍，DF 影响串操作指令的执行方式，将在本书第 7 章中加以说明。