1. 基本概念和基础知识
   1. 机器语言
   2. 汇编语言的产生
   3. 汇编语言的组成
   4. 汇编语言的特点
   5. 进位记数制与不同基数的数之间的转换
   6. 数和字符的表示
   7. 逻辑运算
2. 80x86计算机组织
   1. 80x86微处理器
   2. 基于微处理器的计算机系统构成
   3. 中央处理机
   4. 存储器
   5. 外部设备
3. 寻址方式与指令系统
   1. 指令的格式
   2. 寻址方式
   3. 指令系统
4. 基本概念和基础知识
   1. 机器语言

说到汇编语言的产生，首先要讲一下机器语言。机器语言是机器指令的集合。机器指令展开来讲就是一台机器可以正确执行的命令。电子计算机的机器指令是一列二进制数字。计算机将之转变为一列高低电平，以使计算机的电子器件受到驱动，进行运算。

上面所说的计算机指的是可以执行机器指令，进行运算的机器。这是早期计算机的概念。现在，在我们常用的PC机中，有一个芯片来完成上面所说的计算机的功能。这个芯片就是我们常说的CPU（Central Processing Unit，中央处理单元），CPU是一种微处理器。以后我们提到的计算机是指由CPU和其他受CPU直接或间接控制的芯片、器件、设备组成的计算机系统，比如我们最常见的PC机。

每一种微处理器，由于硬件设计和内部结构的不同，就需要用不同的电平脉冲来控制，使它工作。所以每一种微处理器都有自己的机器指令集，也就是机器语言。

早期的程序设计均使用机器语言。程序员们将用0、1数字编成的程序代码打在纸带或卡片上，1打孔，0不打孔，再将程序通过纸带机或卡片机输入计算机，进行运算。

* 1. 汇编语言的产生

早期的程序员们很快就发现了使用机器语言带来的麻烦，它是如此难于辨别和记忆，给整个产业的发展带来了障碍。于是汇编语言产生了。

汇编语言的主体是汇编指令。汇编指令和机器指令的差别在于指令的表示方法上。汇编指令是机器指令便于记忆的书写格式。

例如∶机器指令100010011011000表示把寄存器BX的内容送到AX中。汇编指令则写成mov ax，bx。这样的写法与人类语言接近，便于阅读和记忆。

操作∶寄存器BX的内容送到AX中

机器指令∶1000100111011000

汇编指令∶mov ax，bx

（寄存器，简单地讲是CPU中可以存储数据的器件，一个CPU中有多个寄存器。AX 是其中一个寄存器的代号，BX 是另一个寄存器的代号。更详细的内容我们在以后的课程中将会讲到。）

此后，程序员们就用汇编指令编写源程序。可是，计算机能读懂的只有机器指令，那么如何让计算机执行程序员用汇编指令编写的程序呢?这时，就需要有一个能够将汇编指令转换成机器指令的翻译程序，这样的程序我们称其为编译器。程序员用汇编语言写出源程序，再用汇编编译器将其编译为机器码，由计算机最终执行。图1.1描述了这个工作过程。

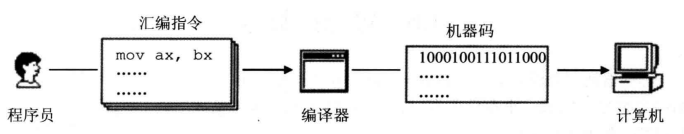


图1.1 用汇编语言编写程序的工作过程

* 1. 汇编语言的组成

汇编语言发展至今，有以下3类指令组成。

1. 汇编指令∶机器码的助记符，有对应的机器码。
2. 伪指令∶没有对应的机器码，由编译器执行，计算机并不执行。
3. 其他符号∶如+、一、\*、/等，由编译器识别，没有对应的机器码。汇编语言的核心是汇编指令，它决定了汇编语言的特性。
   1. 汇编语言的特点

汇编语言是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。

汇编语言保持了机器语言的优点，具有直接和简捷的特点。

汇编语言可有效地访问、控制计算机的各种硬件设备,如磁盘、存储器、CPU、I/O端口等。

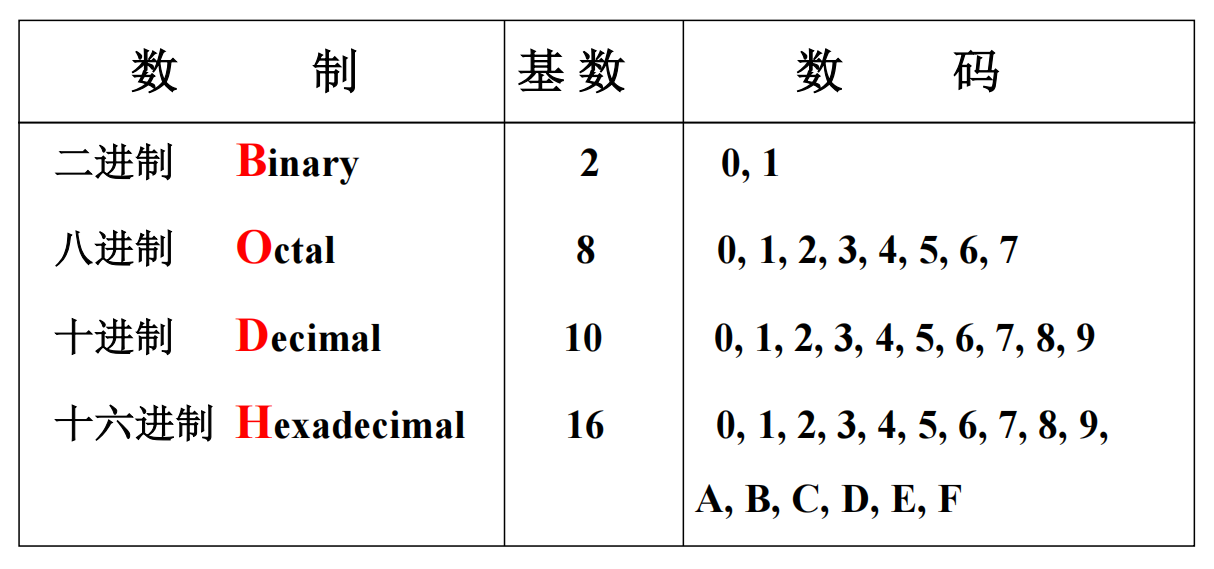
汇编语言目标代码简短，占用内存少，执行速度快，是高效的程序设计语言。

汇编语言经常与高级语言配合使用，应用十分广泛。

* 1. 进位记数制与不同基数的数之间的转换
     1. 二进制数

进位计数制是一种计数的方法,习惯上最常用的是十进制计数法。

计算机中为便于存储及计算的物理实现，采用了二进制数.二进制数的基数为2，只有0 两个数码，并遵循逢二进一的规则，它的各位权是以2表示的，



* + 1. 二进制数和十进制数之间的转换
       1. 二进制数转换为十进制数

各位二进制数码乘以与其对应的权之和即为与该二进制数相对应的十进制数。

* + - 1. 十进制数转换为二进制数

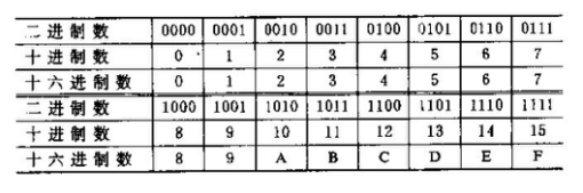
十进制数转换为二进制数的方法很多，这里只说明比较简单的降寒法及除法两种。

降幂法：首先写出要转换的十进制数，其次写出所有小于此数的各位二进制权值，然后用要转换的十进制数减去与它最相近的二进制权值，如够减用减去并在相应位记以1；如不够减则在相应位记以0并跳过此位；如此不断反复，直到该数为0为止。

* + 1. 十六进制数及其与二进制数、十进制数之间的转换
       1. 十六进制数的表示

计算机中存储信息的基本单位为一个二进制位（Bit），它可以用来表示0和1两个数码。此外，由于计算机中常用的字符是采用由8位二进制数组成的一个字节（Byte）来表示的，因此字节也成为计算机中存储信息的单位。计算机的字长一般都选为字节的整数倍，如16位、32位。64位等。一个字节由8位组成，它可以用两个四位组（又称半字节）来表示，所以用十六进制数来表示二进制数是比较方便的。

十六进制数的基数为16，共有16个数码，它们是0，1，2，3，4，5，6，7，8，9，A，B，C，D，E，F，其中A表示十进制的10，余类推。它们与二进制和十进制数的对应关系如下∶



* + - 1. 十六进制数和二进制数之间的转换

由于十六进制数的基数是2的幂，所以这两种数制之间的转换是十分容易的。一个二进制数，只要把它从低位到高位每4位组成一组，直接用十六进制数来表示就可以了。

例

0011 0101 1011 1111

3 5 B 0

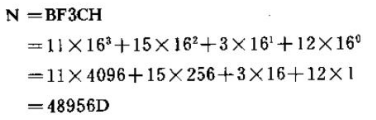
亦即 001101011011111B=35BFH

反之，把十六进制数中的每一位用4位二进制数表示，就形成相应的二进制数了。

* + - 1. 十六进制数和十进制数之间的转换

各位十六进制数与其对应权值的乘积之和即为与此十六进销数相对应的十进制数。

例



十进制数转换为十六进制数也可使用降幕法和除法。

降幂法：首先写出要转换的十进制数，其次写出小于该数的十六进制权值，然后找出该数中包含多少个最接近它的权值的倍数，这一倍数即对应位的值，用压数减去此倍数与相应位权值的乘积得到一个差值，再用此差值去找低一位的权值的倍数，如此反复直到差值为0为止。

除法：把要转换的十进制数的整数部分不断除以16，并记下余数，直到商为止。

* 1. 数和字符的表示
     1. 数的补码表示

计算机中的数是用二进制来表示的，数的符号也是用二进制表示的，把一个数连同其符号在内在机器中的表示加以数值化，这样的数称为机器数）一般用最高有效值来表示数的符号，正数用0表示，负数用1表示。机器数可以用不同的码制来表示，常用的有原码、补码和反码表示法。由于多数机器的整数采用补最表示法，IBMPC机也是这样，所以我们在这里只介绍补码表示法。

补码表示法中，正数采用符号-绝对值表示，即数的最高有效位为0表示符号为正，数的其余部分则表示数的绝对值。例如，假设机器字长为8位，则[＋1]=00000001，[＋127]=01111111，[+0]=000000000。

当用补码表示法来表示负数时则要麻烦一些。负数X用一|X|来表示，其中n为机器的字长。当n=8时，[-1]=-1=1=11111111，而[-127]=-127=10000001，显然，最高有效位为1表示该数的符号为负。应该注意，[-0]==00000000，所以在补码表示法中0只有一种表示，即00000000。对于10000000这个数，在补码表示法中被定义为-128。这样，8位补码能表示数的范围为-128～＋127。

我们可以用一种比较简单的办法来写出一个负数的补码表示∶先写出与该负数相对应的正数的补码表示（用符号-绝对值法），然后将其按位求反（即0变为1，1变为0），最后在末位（最低位）加1，就可以得到该负数的补码表示了。

* + 1. 补码的加法和减法

我们知道，对一个正数的补码表示按位求反后再在末位加1，可以得到与此正数相应的负数的补码表示。我们把这种对一个二进制数按位求反后在末位加1的运算称为求补运算，可以证明补码表示的数具有以下特性∶



在这里，只用例子来说明。由下例可见∶

[117]=0075H

[-117]=FF8BH

现对 [-117] 作求补运算∶

[-117]为 1111 111 1000 1011

按位求反后得 0000 0000 0111 0100

末位加1后得00000000000001101

此数正是[＋117]=0075H。

这一特性在补码的加、减法运算中很有用。

补码的加法规则是∶

[X+Y]=[X]+[Y]

补码的减法规则是∶

[X-Y]=[X]+[-Y]

其中的[-Y]只要对[Y]求补就可得到。对于这两个规则我们只用例子来说明。读者可以从下面的例子中认识到由于用补码表示数，使计算机中的加、减法运算十分简便，它不必判断数的正负，只要符号位参加运算能自动地得到正确的结果。

* + 1. 无符号整数

在某些情况下，要处理的数全是正数，此时再保留符号位就没有意义了。我们可以把最高有效位也作为数值处理，这样的数称为无符号整数。16位无符号数的表数范围是0≤N≤65535，8位无符号数的表数范围是0≤N≤255。

在计算机中最常用的无符号整数是表示地址的数。此外，如双精度数的低位字也是无符号整数等。在某些情况下，帮符号的数（在机器中用补码表示）与无符号数的处理是有差别的，读者在处理数时，应注意它们的区别。

* + 1. 字符表示法

计算机中处理的信息并不全是数，有时需要处理字符或字符串，例如从键盘输入的信息或打印输出的信息都是字符方式输入输出的，因此，计算机必须能表示字符。字符包括∶

字母A、B、…、Z，a、b、…、Z；

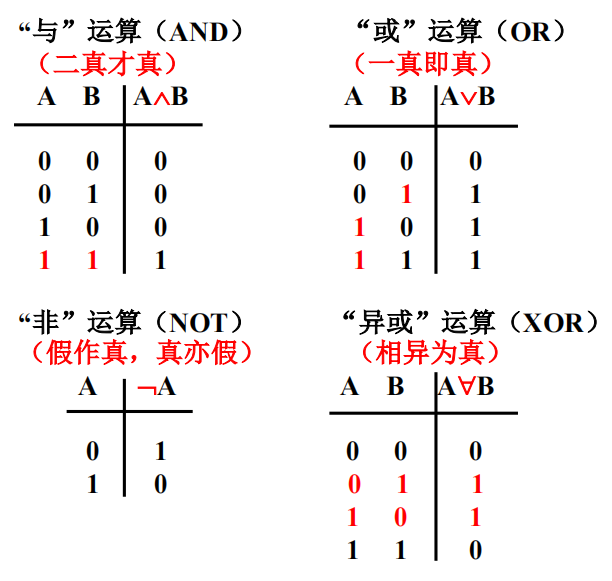
数字∶0、1、…、9；

专用字符∶＋、-、\*、/、半、SP（space空格）····

非打印字符∶BEL（Bell响铃）、LF（Line Feed换行），CR（Carriage Return回车）、……

这些字符在机器里必须用二进制数来表示。IBM PC机采用目前最常用的美国信息交换标准代码ASCII（American Standard Code for Information Interchange）来表示。这种代码用一个字节（8位二进制码）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。下表列出了用十六进制数表示的部分常用字符的ASCII值。

* 1. 逻辑运算



* 1. 存储单元

存储器被划分成若干个存储单元，每个存储单元从0开始顺序编号，例如一个存储器有128个存储单元，编号从0～127，如图1.2所示。

那么一个存储单元能存储多少信息呢?我们知道电子计算机的最小信息单位是 bit（音译为比特），也就是一个二进制位。8个bit组成一个Byte，也就是通常讲的一个字节。微型机存储器的存储单元可以存储一个 Byte，即8个二进制位。一个存储器有128个存储单元，它可以存储128个Byte。

微机存储器的容量是以字节为最小单位来计算的。对于拥有 128 个存储单元的存储器，我们可以说，它的容量是128个字节。

对于大容量的存储器一般还用以下单位来计量容量（以下用B来代表Byte）∶

1KB = 1024B = 210B （KiloByte、千字节）

1MB = 1024KB = 220B （MegaBate、兆字节）

1GB = 1024MB = 230B （GigaByte、千兆字节）

1TB = 1024GB = 240B （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

磁盘的容量单位同内存的一样，实际上以上单位是微机中常用的计量单位。

1. 80x86计算机组织
   1. 80x86微处理器
   2. 基于微处理器的计算机系统构成
   3. 中央处理机

CPU的任务是执行存放在存储器里的指令序列。它由运算器和控制器两部分组成。

从图中可以看出，CPU由三部分组成∶

1.算术逻辑部件ALU（Arith-metic Logic Unit）用来进行算术和逻辑运算。

2.控制逻辑负责对全机的控制工作，包括从存储器取出指令，对指令进行译码分析，从存储器取得操作数，发出执行指令的所有命令，

把结果存入存储器，以及对总线及I/O传送的控制等。

3.工作寄存器在计算机中起着重要的作用，每一个寄存器相当于运算器中的一个存储单元，但它的存取速度比存储器要快得多。

* 1. 存储器
     1. 存储器

CPU是计算机的核心部件，它控制整个计算机的运作并进行运算。要想让一个 CPU 工作，就必须向它提供指令和数据。指令和数据在存储器中存放，也就是我们平时所说的内存。在一台PC机中内存的作用仅次于CPU。离开了内存，性能再好的CPU也无法工作。这就像再聪明的大脑，没有了记忆也无法进行思考。磁盘不同于内存，磁盘上的数据或程序如果不读到内存中，就无法被CPU使用。要灵活地利用汇编语言编程，我们首先要了解CPU是如何从内存中读取信息，以及向内存中写入信息的。

* + 1. 存储单元的地址和内容

计算机存储信息的基本单位是一个二进制位，一位可存储一个二进制数∶0或1，每8位组成一个字节。

那么一个存储单元能存储多少信息呢?我们知道电子计算机的最小信息单位是 bit（音译为比特），也就是一个二进制位。8个bit组成一个Byte，也就是通常讲的一个字节。微型机存储器的存储单元可以存储一个 Byte，即8个二进制位。一个存储器有128个存储单元，它可以存储128个Byte。

微机存储器的容量是以字节为最小单位来计算的。对于拥有 128 个存储单元的存储器，我们可以说，它的容量是128个字节。

对于大容量的存储器一般还用以下单位来计量容量（以下用B来代表Byte）∶

1KB = 1024B = 210B （KiloByte、千字节）

1MB = 1024KB = 220B （MegaBate、兆字节）

1GB = 1024MB = 230B （GigaByte、千兆字节）

1TB = 1024GB = 240B （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

磁盘的容量单位同内存的一样，实际上以上单位是微机中常用的计量单位。

* 1. 外部设备

计算机运行时的程序和数据都要通过输入设备送入机器，程序运行的结果要通过输出设备送给用户，所以输入、输出设备是计算机必不可少的组成部分。大容量的外存储器（如磁盘）能存储大量信息，也是现代计算机不可缺少的一部分，对于外部设备的管理是汇编语言的重要使用场合之一。

外部设备与主机（CPU和存储器）的通信是通过外设接口进行的，每个接口包括一组寄存器。一般说来，这些寄存器有三种不同的用途∶

1. 数据寄存器∶用来存放要在外设和主机间传送的数据，这种寄存器实际上起缓冲器的作用。

2.状态寄存器∶用来保存外部设备或接口的状态信息，以便CPU在必要时测试外设状态，了解外设的工作情况。例如，每个设备都有忙闲位用来标志设备当前是否正在工作，是否有空接受CPU给予的新任务等。

3.命令寄存器∶CPU给外设或接口的控制命令通过此寄存器送给外部设备。例如，CPU要启动磁盘工作，必须发出启动命令等。

各种外部设备都有以上三种类型的海存器A只是每个接口所配备的寄存器数量是根据设备的需要确定的。例如，工作方式较简单、速度又慢的键盘只有一个8位的数据寄存器，并把状态和命令寄存器合二为一个控制寄存器。又如工作速度快、工作方式又比较复杂的磁盘则需要多个数据、状态和命令寄存器。

1. 寻址方式与指令系统
   1. 指令的格式

计算机中的指令由操作仍字段和操作数字段两部分组成。操作码字段指示计算机所要执行的操作，而操作数字段则指出在指令执行操作的过程中所需要的操作数。例如，加法指令除需要指定做加法操作外，还需提供加数和被加数。操作数字段可以是操作数本身。也可以是操作数地址或是地址的一部分，还可以是指向操作数地址的指针或其他有关操作数的信息。

指令的格式一般是∶

YB`K36{9L{RG[OT4PN2}1K2

操作数字段可以有一个、二个或三个，通常称为一地址、二地址或三地址指令。例如，单操作数指令就是一地址指令，它只需要指定一个操作数，如加1指令只需要指出需要加1的操作数。大多数运算型指令都是双操作数指令。对于这种指令，有的机器使用三地址指令∶除给出参加运算的两个操作数外，还指出运算结果的存放地址。近代多数机器则使用二地址指令，此时分别称两个操作数为源操作数和目的操作数。尽管在指令执行前这两个操作数都是输入楼作数，但指令执行后将把运算结果存放到目的操作数的地址之中，也就是说，经过运算后，参加运算的一个操作数将会丢失，但一般不关心这个问题。如果在以后的运算中还会用到这个操作数的话，则应在运算之前先为它准备一个副本（即提前存储起来）。80x86的大多数运算型指令就采用这种二地址指令，少数采用三地址命令。

指令的操作码字段在机器里的表示比较简单，只需对每一种操作指定确定的二进制代码就可以了。指令的操作数字段的情况就比较复杂了，如果操作数存放在寄存器中，则由于寄存器的数量较少，因而需要指定的操作数地址的位数就较少；但如果操作数存放在存储器里，那么一个存储单元的地址就需要20位，怎样设法使它在指令的操作数字段的表示中减少位数呢?另外，从程序运行时的数据结构来看，操作数常常不是单个的数，往往是成组的以表格或数组的形式存放在存储器的某一区中，在这种情况下，指令用什么方式来指定操作数地址更好呢?这在计算机的设计中是一个很重要的问题，它会影响机器运行的速度和效率。下一节将专门讨论这个问题。

* 1. 寻址方式
     1. 与数据有关的寻址方式

1. 立即寻址方式

操作数的值直接存储在指令的操作数字段中，作为指令的一部分存放在代码段里，这种操作数称为立即数。

立即寻址方式用来表示常数。它经常用于给寄存器赋初值，并且只能用于源操作数字段，不能用于自的操作教学段。

1. 寄存器寻址方式

操作数在寄存器中，指令指定寄存器号。对于16位操作数，寄存器可以是AX、BX、CX、DX、SI、DI、SP和BP等；对于8位操作数，寄存器可以是AL、AH、BL、BH、CL、CH、DL和DH。这种寻址方式由于操作数就在寄存器中，不需要访间存储器来取得操作数，因而可以取得较高的运算速度。

1. 直接寻址方式

指令的操作数字段是操作数的地址。具体将就是段内的偏移地址或者位移量。

1. 寄存器间接寻址方式

指令中出现的是数据地址的存储地址。为了加快程序的执行，我们规定将数据地址存放在寄存器中，指令中指定该寄存器进行寻址。有：操作数的有效地址在基址寄存器BX、BP或者变址寄存器SI、DI中，而操作数则在存储器中。

1. 寄存器相对寻址方式

有效地址是一个基址或变址寄存器的内容和指令中指定的一个8位或者16位位移量（displacement)之和。

1. 基址变址寻址方式

操作数的有效地址是一个基址寄存器和一个变址寄存器的内容之和。两个寄存器均由指令指定。

1. 相对基址变址寻址方式

操作数的有效地址是一个基址寄存器与一个变址寄存器的内容和指令中指定的位移量之和，所以有效地址由三种成分组成。

* + 1. 与数据有关的寻址方式

1. 段内直接寻址

转向的有效地址是当前IP寄存器的内容和指令中指定的-8-位或16位位移量之和。

这种方式的转向有效地址用相对于当前IP值的位移量来表示，所以它是一种相对寻址方式。指令中的位移量是转向的有效地址与当前IP值之差，所以当这一程序段在内存中的不同区域运行时，这种寻址方式的转移指令本身不会发生变化，这是符合程序的再定位要求的。这种寻址方式适用于条件转移及无条件转移指令，但是当它用于条件转移指令时，位移量只允许8位。无条件转移指令在位移量为8位时称为短跳转，位移量为16位时则称为近跳转。

1. 段内间接寻址

转向有效地址是一个寄存器或是一个存储单元的内容。这个寄存器或存储单元的内容可以用数据寻址方式中除立即数以外的任何一种寻址方式取得，所得到的转向的有效地址用来取代IP 寄存器的内容。

1. 段间直接寻址

指令中直接提供了转向段地址和偏移地址，所以只要用指令中指定的偏移地址取代IP寄存器的内容，用指令中指定的段地址取代CS寄存器的内容就完成了从一个段到另一个段的转移操作。

1. 段间间接寻址

用存储器中的二个相继字的内容来取代IP和CS寄存器中的原始内容以达到段间转移的目的。这里存储单元的地址是由指令指定除立即数方式和寄存器方式以外的任何一种数据寻址方式取得。

* 1. 指令系统
     1. 数据传送指令

1. 通用数据传送指令

MOV（Move）传送

PUSH（Push onto the stack）进栈

POP（Pop from the stack）出栈

XCHG（Exchange）交换

1. MOV传送指令

格式为∶MOV DST，SRC

执行操作∶（DST）<-（SRC）

其中DST表示目的操作数，SRC表示源操作数。

1. PUSH进栈指令

格式为∶ PUSH SRC

执行操作:

(SP)<-(SP)-2

((SP)+1,(SP))<-(SRC)

1. POP出栈指令

格式为∶ POP DST

执行操作

(DST)<-((SP)+1,(SP))

(SP)<-(SP)+2

1. XCHG 交换指令

格式为∶ XCHG OPR1，OPR2

执行操作∶（OPR1）<->（OPR2）

其中OPR表示操作数。该指令的两个操作数中必须有一个在寄存器中。

1. 累加器专用传送指令

IN（Input） 输入

OUT（Output）输出

XLAT（Translate） 换码

这组指令只限于使用累加器AX或AL传送信息。

（1）IN 输入指令

长格式为：

IN AL，PORT（字节）

IN AX，PORT（字）

执行的操作∶

（AL）<-（PORT）（字节）

（AX）<-（PORT+1，PORT）（字）

短格式为∶

IN AL，DX（字节）

IN AX，DX（字）

执行的操作：

（AL）<-（（DX））（字节）

（AX）<-（（DX）+1，（DX））（字）

（2）OUT 输出指令

长格式为∶

OUT PORT，AL（字节）

OUT PORT，AX（字）

执行的操作：

（PORT）+（AL）（字节）

（PORT+1，PORT）+（AX）（字）

短格式为∶

OUT DX，AL（字节）

OUT DX，AX（字）

执行操作为：

（（DX））<-（AL）（字节）

（（DX）+1，（DX）<-（AX）（字）

（3）XLAT 换码指令

格式为∶

XLAT OPR

或XLAT

执行的操作∶（AL）<-（（BX）+（AL））

1. 地址传送指令

LEA（Load effective address）有效地址送寄存器

LDS（Load DS with Polnter）指针送寄存器和DS

LES（Load ES with Pointer）指针送寄存器和ES

（1）LEA 有效地址送寄存器指令

格式为∶LEA REG,SRC

执行的操作；（REG）<-SRC

指令把源操作数的有效地址送到指定的寄存器中。

（2）LDS 指针送寄存器和DS指令

格式为：LDS REG,SRC

执行的操作∶（REG）<-（SRC）

(DS)<-(SRC+2)

* + 1. 算术指令

1. 加法指令

ADD 加法指令

ADC（add with carry）带进位加法

INC（increment）加1

1. ADD加法指令

格式：ADD DST,SRC

执行的操作∶（DST）<-（SRC）+（DST）

1. ADC带进位加法指令

格式∶ADC DST,SRC

执行的操作∶（DST）<-（SRC）+（DST）+CF

其中CF为进位位的值。

（3）INC 加1指令

格式：INC OPR

执行的操作∶（OPR）<-（OPR）＋1

1. 减法指令

SUB（subtract）减法指令

SBB（subtract with borrow）带借位减法

DEC（Decrement）减1

NEG（Negate）求补

CMP（Compare）比较

1. SUB指令

格式：SUB DST,SRC

执行的操作∶（DST）<-（DST）-（SRC）

（2）SBB 带借位减法指令

格式∶SBB DST,SRC

执行的操作∶（DST）<-（DST）-（SRC）-CF

其中CF为进位位的值。

（3）DEC减1指令

格式∶DEC OPR

执行的操作∶（OPR）<-（OPR）-1

（4）NEG求补指令

格式∶NEG OPR

执行的操作∶（OPR）<- -（OPR）

亦即把操作数按位求反后末位+1，因而执行的操作也可表示为∶

(OPR)<-0FFFFH--(OPR)+1

（5）CMP比较指令

格式∶CMP OPR1,OPR2

执行的操作∶（OPR1）-（OPR2）

1. 乘法指令

MUL（Unsigned Multiple）无符号数乘法

IMUL（Signed Multiple）带符号数乘法

（1）MUL 无符号数乘法指令

格式：MUL SRC

执行的操作：

字节操作数∶（AX）<-（AL）\*（SRC）

字操作数∶（DX，AX）<-（AX）\*（SRC）

1. IMUL 带符号数乘法指令

格式：IMUL SRC

执行的操作∶与MUL相同，但必须是带符号数，而MUL是无符号数。

1. 除法指令

DIV（Unsigned divide）无符号数除法

IDIV（Signed divide）带符号数除法

1. DIV无符号数除法指令

格式为∶ DIV SRC

执行的操作∶

字节操作∶16位被除数在AX中，8位除数为源操作数，结果的8位商在AL中，8 位余数在AH中。表示为∶

（AL）<-（AX）/（SRC）的商

（AH）<-（AX）/（SRC）的余数

字操作∶32位被除数在DX，AX中，其中DX为高位字∶16位除数为源操作数，结果的∶6位商在AX中，16位余数在DX中，表示为∶

（AX）<-（DX.AX）/（SRC）的商

（DX）<-（DX，AX）/（SRC）的余数。

（2）IDIV 带符号数除法指令

格式：IDIV SRC.

抉行的操作∶与DIV相同，但操作数必须是带符号数，商和余数也均为带符号数，且余数的筹号和被除数的符号相同。

* + 1. 逻辑指令
       1. 逻辑运算指令

AND(and) 逻辑与

OR(or) 逻辑或

NOT(not) 逻辑非

XOR（exclusive or） 异或

TEST (test) 测试

逻辑运算指令可以对字或字节执行逻辑运算。由于逻辑运算是按位操作的，因此一般说来，其操作数应该是位串而不是数。

（1）AND 逻辑与指令

格式∶AND DST,SRC

执行的操作∶（DST）<-（DST）（SRC）

（2）OR 逻辑或指令

格式∶OR DST,SRC

执行的操作∶（DST）<-（DST）（SRC）

（3）NOT 逻辑非指令

格式∶NOT OPR

执行的操作∶（OPR）<-（）

（4）OR 异或指令

格式∶XOR DST,SRC

执行的操作∶（DST）<-（DST）（SRC）

（5） TEST 测试指令

格式∶TEST OPR1,OPR2

执行的操作∶（OPR1）（OPR2）

* + 1. 串处理指令
    2. 控制转移指令
    3. 处理机控制与杂项操作指令

参考资料：

《IBM-PC汇编语言程序设计》（第2版）