第 3 章 指令系统

3.1 基本内容摘要

- 指令格式
  - 机器指令的基本格式
  - 地址码结构

    三、二、一、零地址指令的特点。
  - 指令的操作码

    规整型编码；

    非规整型编码(扩展操作码)。
- 寻址技术
  - 编址方式

    编址与编址单位；

    指令中地址码的位数。
  - 指令寻址和数据寻址
  - 基本的数据寻址方式

    立即寻址；

    寄存器寻址；

    直接寻址；

    间接寻址；

    寄存器间接寻址；

    变址寻址；

    基址寻址；

    相对寻址；

    页面寻址。
  - 变型或组合寻址方式
- 堆栈与堆栈操作
  - 堆栈结构

    寄存器堆栈；

    存储器堆栈。

- ◆ 堆栈操作
- 指令类型
  - ◆ 数据传送类指令
  - ◆ 运算类指令
  - ◆ 程序控制类指令

    转移指令；

    子程序调用指令；

    返回指令。
  - ◆ 输入输出类指令

    独立编址的输入输出；

    统一编址的输入输出。
- 指令系统的发展
  - ◆ 从复杂指令系统到精简指令系统
  - ◆ VLIW 和 EPIC

3.2 重点难点梳理

1. 机器指令的长度

一条机器指令通常可以分成操作码和地址码两部分。指令的长度取决于操作码的长度、地址码的个数以及每个地址的长度。一条指令构成一个指令字。指令字长与机器字长是两个不同的概念，两者之间没有固定的关系。早期的计算机多采用定长指令字结构，即所有指令的长度都是相等的，并且指令字长和机器字长相同。随着计算机技术的发展，现代计算机多采用变长指令字结构，即指令字长既可以与机器字长相等，也可以大于或小于机器字长。指令字长等于机器字长的指令称为单字长指令，指令字长等于半个机器字长的指令称为半字长指令，指令字长等于两个机器字长的指令称为双字长指令……例如，Intel 8086 的机器字长为 16 位，指令字长为 1～6 字节，最短的指令只有 8 位(半字长指令)，最长的指令为 48 位(3 字长指令)。

2. 不同地址数指令的区别

对于双操作数运算类指令来说，每条指令中都需要包含以下 4 个地址信息：

(1) 第一操作数地址 A_1；

(2) 第二操作数地址 A_2；

(3) 操作结果存放地址 A_3；

(4) 下一条指令的地址 A_4。

这些地址信息可以明确地给出，称为显地址；也可以依照某种事先的约定，用隐含的方式给出，称为隐地址。

大多数计算机中用程序计数器(PC)指出要执行的下一条指令的地址，其余 3 个地址的处理方式有下面 4 种(以双操作数的加法指令为例)。

1) 三地址指令

三地址指令表示指令中有 3 个显地址，指令的含义为

$$(A_1)+(A_2)\rightarrow A_3$$

执行一条三地址的加法指令需要访问 4 次主存。第一次取操作码，第二次取被加数，第三次取加数，第四次保存结果。

2）二地址指令

二地址指令表示指令中有两个显地址，让第一操作数地址同时兼作结果存放地址（目的地址），指令的含义为

$$(A_1)+(A_2)\rightarrow A_1$$

执行一条二地址的加法指令同样需要访问 4 次主存。

3）一地址指令

一地址指令只有一个显地址，参加运算的另一个操作数来自累加寄存器 Acc。指令的含义为

$$(\mathrm{Acc})+(A_1)\rightarrow \mathrm{Acc}$$

执行一条一地址的加法指令只需要访问两次主存。第一次取指令本身，第二次取被加数。由于加数和结果都放在累加寄存器中，所以读取加数和存入结果都不需要访问主存。

4）零地址指令

零地址指令中只有操作码，没有显地址。零地址的加法指令仅用在堆栈计算机中，堆栈计算机没有一般计算机中必备的通用寄存器，操作数和结果都保存在堆栈中。通常，参加加法运算的两个操作数隐含地从堆栈顶部弹出，送到运算器中进行运算，运算的结果再隐含地压入堆栈。

如果采用存储器堆栈，执行一条零地址的加法指令仍需要访问 4 次主存，这是因为存储器堆栈就是主存的一部分；而如果采用寄存器堆栈，执行一条零地址的加法指令只需要在取指令的时候访问一次主存即可。

3. 二地址指令的分类

前面提到的地址都是指存储器的地址，如二地址 M-M 型指令，两个操作数均在主存中，执行一条运算类指令需要多次访问主存，且指令字长比较大，所以 M-M 型指令在通用计算机中并不适用。事实上，参与运算的两个操作数并不一定都在主存中，往往有一个或两个在通用寄存器中，此时的二地址指令就是 R-M 或 R-R 型指令。表 3-1 给出了不同类型二地址指令的区别。

表 3-1　不同类型二地址指令的区别

二地址指令类型	名　　称	操作数物理位置	执行速度	访问主存次数（取操作码除外）
M-M	存储器-存储器	主存	最慢	多次
R-R	寄存器-寄存器	寄存器	最快	不访问
R-M	寄存器-存储器	寄存器-主存	在 M-M 型和 R-R 型之间	一次

4. 指令系统中操作数的位置

按照 CPU 中操作数的存储位置，指令系统可分为堆栈型、累加器型和通用寄存器型 3 类，其相应的机器分别称为堆栈型机器、累加器型机器和通用寄存器型机器。注意，CPU 中操作数的存储位置是针对指令系统中运算类指令来分类的。在堆栈型机器中，运算指令

的操作数地址是隐含的,操作数在栈顶,即前面所描述的零地址指令;在累加器型机器的运算指令中有一个操作数地址是隐含的,这个操作数在累加器中,即前面所描述的一地址指令;在通用寄存器型机器的运算指令中,操作数全部是显式给出的,或者为寄存器地址,或者为主存地址,也就是前面所描述的二地址指令或三地址指令。

堆栈型机器的主要优点是表达式计算简单,指令短,但是堆栈不能被随机访问,并且栈顶容易形成瓶颈。累加器型机器的主要优点是机器的内部状态简单,指令短,实现容易,但是累加器同样容易成为计算机的瓶颈。在通用寄存器型机器中,编译器可以有效地计算表达式的值,可以减少访存次数,提高程序执行速度,但是指令中的操作数需要显式给出,导致指令字较长。

通用寄存器型机器根据运算指令中存储器操作数的个数,又可以进一步分为3种类型:寄存器-寄存器(R-R)型,寄存器-存储器(R-M)型以及存储器-存储器(M-M)型。从表3-1中可以看出,R-R型指令系统运算类指令中不包含存储器操作数,因此运算速度快,目前的RISC计算机均为这种指令系统。R-M型指令系统运算类指令中既有寄存器操作数又有存储器操作数,运算速度居中;而M-M型指令系统由于要多次访存,运算速度最慢。当然,M-M型指令系统并不是所有运算类指令中的操作数均存放在存储器中,因为这与该指令系统仍然属于通用寄存器型指令系统是违背的。M-M型指令系统中只有部分运算指令将操作数都存放在存储器中。在目前的指令系统中,通常只允许指令中最多具有一个存储器操作数。

5. 扩展操作码法

指令操作码可以采用规整型和非规整型两种编码方式,最常用的非规整型编码方式是扩展操作码法。如果指令长度一定,则地址码与操作码字段的长度是相互制约的。为了解决这一矛盾,让操作数地址个数多的指令(三地址指令)的操作码短些,操作数地址个数少的指令(一或零地址指令)的操作码长些,这样就能在不增加指令长度的情况下扩展操作码的位数,能表示更多的指令。假设某机的指令长度为16位,操作码为4位,有3个4位的地址码,如果按照定长编码的方法,最多只能表示16条不同的三地址指令。

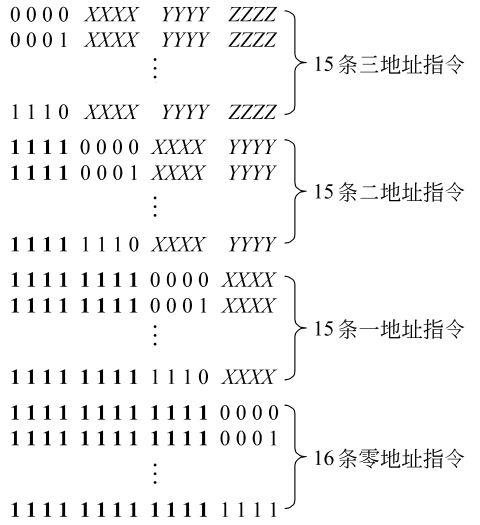


图3-1 扩展操作码举例

利用扩展操作码法可以在指令长度不变的情况下,使指令的总数远远大于16条。例如,指令系统中要求有15条三地址指令、15条二地址指令、15条一地址指令和16条零地址指令,共61条指令。扩展的方法如图3-1所示。

在上述扩展过程中,每种类型的指令都只留下一种编码作为扩展窗口。实际上可以有许多种不同的扩展方案。例如,也可以形成15条三地址指令、14条二地址指令、31条一地址指令和16条零地址指令,共76条指令,具体如下:

(1) 4位操作码的编码0000~1110定义15条三地址指令,留下1111作为扩展窗口,与下一个4位组成一个8位的操作码字段;

(2) 8 位操作码的编码 11110000～11111101 定义 14 条二地址指令，留下 11111110 和 11111111 作为扩展窗口，与下一个 4 位组成一个 12 位的操作码字段；

(3) 12 位操作码的编码 111111100000～111111111110 定义 31 条一地址指令，扩展窗口为 111111111111，与下一个 4 位组成 16 位的操作码字段；

(4) 最后，由 16 位操作码的编码 1111111111110000～1111111111111111 定义 16 条零地址指令。

不论采用何种方案，必须注意以下两点：

- 不允许短码是长码的前缀，即短操作码不能与长操作码的前面部分的代码相同，否则将无法保证解码的唯一性和实时性。
- 各条指令的操作码一定不能重复，而且各类指令的格式安排应统一规整。

6. 字编址和字节编址

字编址是实现起来最容易的一种编址方式，这是因为每个编址单位与访问单位一致，即每个编址单位所包含的信息量(二进制位数)与访问一次寄存器、主存所获得的信息量相同。早期的大多数计算机都采用这种编址方式。

在采用字编址的计算机中，每执行一条指令，程序计数器加 1；每从主存中读出一个数据，地址寄存器加 1。这种控制方式实现起来简单，地址信息没有任何浪费；其主要缺点是不支持非数值应用。

目前使用最普遍的编址方式是字节编址，这是为了适应非数值应用的需要。字节编址方式使编址单位与信息的基本单位(字节)相一致，但主存的访问单位是若干字节。

在采用字节编址的计算机中，如果指令长度是 32 位，那么每执行完一条指令，程序计数器要加 4。如果数据字长是 32 位，当连续访问存储器时，每读写完一个数据字，地址寄存器要加 4。由此可见，字节编址方式存在着地址信息的浪费。

7. 地址码位数与主存容量和最小寻址单位的关系

指令格式中每个地址码的位数是与主存容量和最小寻址单位(即编址单位)相关联的。主存容量越大，访问全部存储空间所需的地址码位数就越长。对于相同的存储容量来说，如果以字节为最小寻址单位，所需的地址码的位数就要长些，但是可以方便地对每一个字符进行处理；如果以字为最小寻址单位(假定字长为 16 位或更长)，所需的地址码的位数可以减少，但对字符操作比较困难。例如，设某机主存容量为 64MB，机器字长 32b(位)，若最小寻址单位为字节(按字节编址)，其地址码应为 26b($2^{26}=64$MB)；若最小寻址单位为字(按字编址)，其地址码只需 24b。这是因为 32b 的一个字(其单位符号写为 W)等于 4 字节(B)，则有

$$64\text{MB} = \frac{64}{4}\text{MW} = 16\text{MW}$$

从减少指令长度的角度看，最小寻址单位越大越好；而从对字符或位的操作是否方便的角度看，最小寻址单位越小越好。

8. 指令寻址和数据寻址

寻址可以分为指令寻址和数据寻址。寻找下一条将要执行的指令地址称为指令寻址，寻找操作数的地址称为数据寻址。

指令寻址比较简单，它又可以分为顺序寻址和跳跃寻址。顺序寻址可通过程序计数器加 1 自动形成下一条指令的地址。需要说明的是，这里所说的加 1 中的 1 并不是数字 1 本

身,而是逻辑上的递增,即表示程序计数器将指向下一条待执行的指令地址,具体PC值加多少取决于指令的长度和编址方式。跳跃寻址则需要通过程序转移类指令实现,其转移地址的形成方式有3种:直接(绝对)寻址、相对寻址和间接寻址。

数据寻址方式是根据指令中给出的地址码寻找真实操作数地址的方式。数据寻址的种类较多,如立即寻址、寄存器寻址、直接寻址、间接寻址、变址寻址等,其最终目的都是寻找指令执行时所需要的操作数。

9. 各种数据寻址方式的速度区别

数据寻址的最终目的是寻找指令执行时所需要的操作数。操作数可以在主存中,也可以在寄存器中,甚至可以在堆栈中。各种不同的寻址方式获取操作数的速度是不相同的,其中速度最快的是立即寻址,最慢的是多级间接寻址。

立即寻址是一种特殊的寻址方式,指令中在操作码后面的部分不是通常意义上的地址码,而是操作数本身,也就是数据就包含在指令中,只要取出指令,也就取出了可以立即使用的操作数,不必再次访问主存,从而提高了指令的执行速度。

寄存器寻址获取操作数的速度仅次于立即寻址,因为操作数在通用寄存器中,所以不需要访问主存就可以获得数据。

直接寻址、寄存器间接寻址、变址寻址、基址寻址、相对寻址和页面寻址等寻址方式获取一个操作数都只需要访问一次主存,根据得到有效地址(EA)的难易程度,寻址速度由快至慢依次为直接寻址、寄存器间接寻址、页面寻址、变址寻址(基址寻址、相对寻址)。

间接寻址指令中给出的形式地址A不是操作数的地址,而是操作数地址的地址。这就意味着为获取一个操作数至少需要两次访问主存,第一次得到操作数的有效地址,第二次才能得到操作数。间接寻址允许多级寻址,多级间接寻址为获取操作数需要多次访问主存。

基本寻址方式的比较如表3-2所示。其中的偏移寻址包括变址寻址、基址寻址和相对寻址3种方式。

表3-2 基本寻址方式的比较

寻址方式	规 则	主要优点	主要缺点
立即寻址	操作数=A	无须访问主存	操作数取值范围受限
寄存器寻址	EA=R	无须访问主存	寻址空间受限
直接寻址	EA=A	简单	寻址空间受限
间接寻址	EA=(A)	寻址空间大	多次访问主存
寄存器间接寻址	EA=(R)	寻址空间大	比寄存器寻址多访问一次主存
偏移寻址	EA=(R)+A	灵活	复杂

10. 寄存器寻址的特点

寄存器寻址指令在执行过程中所需要的操作数在通用寄存器中,运算结果也写回通用寄存器中,这种寻址方式在所有的RISC计算机以及大部分CISC计算机中得到广泛应用。

寄存器寻址方式主要有以下优点:

(1) 指令字长短。由于通用寄存器的数量一般为几十个,在指令中只需很少几位就能表示一个操作数的地址。

(2) 指令执行速度快。由于访问通用寄存器的速度很快,与访问主存相比,访问时间几乎可以忽略不计。

(3) 支持向量、矩阵运算。当通用寄存器的数量比较多时,可以把一个向量或向量的一部分放在通用寄存器内,从而提高运算速度。

寄存器寻址方式也有明显的缺点,主要有以下几点:

(1) 不利于优化编译。由于通用寄存器分配是否合理将直接影响程序的执行速度,所以通常要把那些连续使用或用得比较频繁的变量分配在通用寄存器中,这就给编译器的优化设计造成了很大的困难。

(2) 现场切换困难。在程序运行过程中,当发生调用、中断、分时切换等情况时,要把有关通用寄存器中的内容都保存到主存中,在程序返回时,再全部恢复这些通用寄存器中的内容。通用寄存器的数量越多,保存和恢复所需要的时间就越长。为了解决这一问题,目前多数处理机都设置了两套或两套以上的通用寄存器,程序员只能看到其中一套通用寄存器,当发生现场切换时,硬件自动切换到另一套通用寄存器。

(3) 硬件复杂。一方面,在处理机中设置大量的通用寄存器,包括程序员能看到的通用寄存器和程序员看不到的通用寄存器,需要增加硬件;另一方面,这些通用寄存器的读、写及现场切换等控制也相当复杂。

11. 直接寻址与寄存器间接寻址

直接寻址与寄存器间接寻址都属于主存寻址,它们是所有计算机中都普遍采用的寻址方式。

直接寻址又称为绝对寻址,它是在指令中直接给出参加运算的操作数及运算结果所存放的主存地址,即在指令中直接给出有效地址。直接寻址在早期生产的计算机及目前某些专用计算机中用得比较多。随着主存容量的不断扩大及虚拟存储器的普及,这种寻址方式暴露出了许多致命的弱点。首先,它需要很长的地址码,特别是在二地址及三地址指令中,这一矛盾更为突出。其次,为了实现程序循环及高效处理数组运算等,在程序设计中通常需要修改数据的地址,而采用直接寻址方式编写的程序,如果要修改数据地址,就必须修改程序中的指令,采用这种方法编写的程序将没有再入性,这是现代程序设计不能接受的。

寄存器间接寻址指令所指定的寄存器中存放着操作数的有效地址,而操作数则存放在主存中。这种方式既利用了寄存器寻址方式指令字长短、执行速度快的优点,又避免了直接寻址方式只要修改数据地址就必须修改指令本身的弱点。

12. 间接寻址与变址寻址

对于数组运算,通常要用一个循环结构对数组中的各个元素进行操作,这时必须通过修改操作数的地址才能实现。间接寻址方式与变址寻址方式的设计目标都是为了解决操作数地址的修改问题。它们都可以在程序设计过程中对操作数的地址进行修改,而不必修改程序中的指令本身。

原则上,在一个计算机系统中,仅需设置间接寻址与变址寻址两种方式之一即可。例如,在 IBM 公司生产的大型、中型计算机系统中,只有变址寻址方式,没有间接寻址方式;在一些小型及微型计算机系统中,只有间接寻址方式,没有变址寻址方式。当然,也有一些计算机系统同时具有间接寻址方式和变址寻址方式。

这两种寻址方式的区别如下:

(1) 实现的难易程度不同。间接寻址实现起来很容易,只需要增加一条从主存的数据寄存器到地址寄存器的数据通路即可。而变址寻址的实现则需要增加较多的硬件,例如需要一个加法器、一个和多个变址寄存器(也可以与通用寄存器合用)。

(2) 指令的执行速度不同。采用间接寻址方式编写的程序执行速度较慢,读写一个操作数至少需要访问两次主存,第一次读取有效地址,第二次才是读或写操作数,如果采用多级间接寻址,则访问主存的次数还会增多。而采用变址寻址方式编写的程序执行速度较快,有效地址通过加法器直接产生,不需要访问主存。

(3) 对数组运算的支持不同。变址寻址方式比较好,间接寻址方式比较差,这是因为变址寻址方式可以修改变址值。

13. 变址寻址与基址寻址

变址寻址是将变址寄存器的内容 R_x 与指令的形式地址 A 相加,形成操作数有效地址,即 EA=(R_x)+A。R_x 称为变址值。

基址寻址是将基址寄存器的内容 R_b 与指令的形式地址(位移量)相加,形成操作数有效地址,考虑到位移量实际上是一个可正可负的带符号数,故用 D 表示,即 EA=(R_b)+D。R_b 称为基址值。

变址寻址和基址寻址在形成有效地址时所用的算法相同,但使用它们的目的却有所不同。例如,在数组运算中使用时,通常变址寄存器的内容是可变的,而基址寄存器的内容却保持不变。如果需要对一个数组中的每一个元素或者一部分连续存放的元素进行操作时,则可将该数组的首地址作为指令中的形式地址,而将需要操作的第一个元素的序号置入变址寄存器中,于是这条指令执行一次可得到数组中的一个元素;然后将变址寄存器的内容加1,同一条指令的下一次执行就可以得到数组中的下一个元素;重复上述操作,可以顺序取得数组中的全部或部分元素。如果需要从一个数组中找到某个元素,则可将该数组的首地址置入基址寄存器中,所需元素的序号作为指令中的形式地址,于是可以采用基址寻址方式方便地获取所需的元素。

在有些计算机中还设置了基址加变址寻址方式,若将数组的首地址置入基址寄存器,将数组中元素的序号置入变址寄存器,便可方便地实现对该数组中的多个元素进行某种操作。

14. 偏移寻址中的相对寻址

将直接寻址和寄存器间接寻址方式相结合,可以得到几种寻址方式。因为它们提供操作数有效地址的机制相同,都是将指定寄存器的内容与指令中的地址码字段相加,所以统称为偏移寻址。常见的偏移寻址有相对寻址和上述的变址寻址、基址寻址3种。

相对寻址是基址寻址的一种变通,由程序计数器提供基准地址,即 EA=(PC)+D。

由于大多数访存的位置都相对靠近正在执行的指令位置,则使用相对寻址可节省指令中的地址码位数,而且采用相对寻址方式编制程序时,不需要指定绝对地址,只需确定程序内部的相对距离,因而可以使用浮动地址,这样给编程带来了方便。

15. 页面寻址

页面寻址相当于将整个主存空间分成若干个大小相同的区,每个区称为一页,每页有若干个主存单元。每页都有自己的编号,称为页面地址;页面内的每个主存单元也有自己的编号,称为页内地址。这样,存储器的有效地址就被分为两部分:前部为页面地址,后部为页内地址。页面地址的位数取决于主存划分页数的多少,页内地址的位数取决于每一页中存

储单元的多少。页面寻址根据页面地址的形成方式又可以分成3种形式：

(1) 基页寻址，又称零页寻址。由于页面地址全等于0，所以有效地址EA=0//A(//在这里表示简单拼接)，操作数S一定在0页中。基页寻址实际上就是直接寻址。

(2) 当前页寻址。页面地址就等于程序计数器的高位部分的内容，所以有效地址EA=$(PC)_H$//A，操作数S与指令本身一定处于同一页面中。$(PC)_H$代表程序计数器高位部分的内容。当前页寻址示意图参见图3-2。

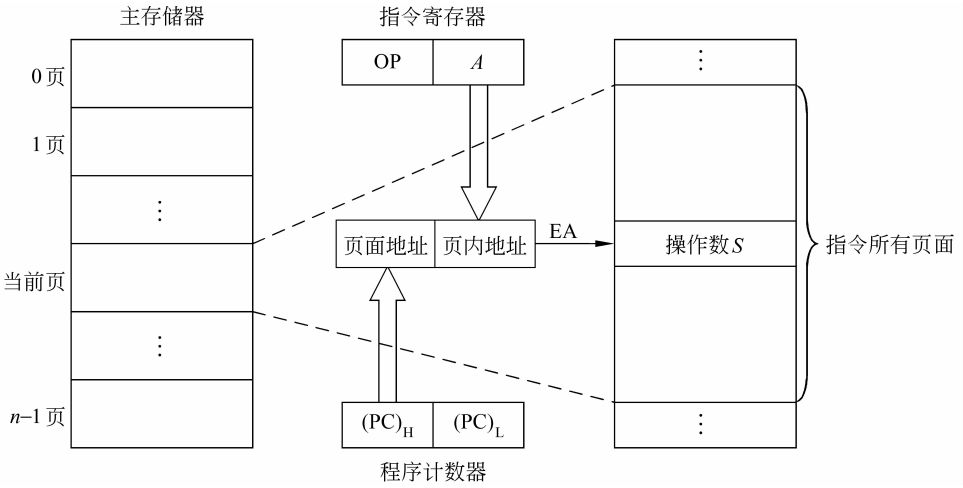


图3-2　当前页寻址示意图

(3) 页寄存器寻址。页面地址取自页寄存器，与形式地址相拼接，形成有效地址。

通常所说的页面寻址主要是指当前页寻址。

当采用页面寻址方式时，页内地址由指令的地址码部分自动直接提供，它与页面地址通过简单的拼接就可得到有效地址，无须像偏移寻址那样进行加法计算，所以可以迅速地得到有效地址。

16. 堆栈寻址

大多数计算机都设置了堆栈，寻找堆栈中的数据称为堆栈寻址方式。堆栈寻址方式的地址是隐含的，在指令中不必给出操作数的地址。参加运算所需要的操作数从堆栈顶端弹出，如果需要两个或多个操作数，则依次从堆栈顶端弹出，运算结果压入堆栈顶端。

自20世纪60年代开始，出现了一批以堆栈寻址方式为主的堆栈计算机，堆栈计算机具有如下特点：

(1) 支持高级语言，有利于编译程序。因为一般的算术表达式很容易转化成逆波兰表达式(后缀表达式)，而逆波兰表达式能够直接形成由堆栈指令组成的程序，这样就简化了编译程序。

(2) 程序的总存储量最小。由于堆栈指令不需要地址码，指令的长度很短，与以寄存器寻址方式和以主存寻址方式的计算机系统相比，虽然堆栈计算机中的程序本身的指令条数没有减少，但程序的总存储量减小许多。

(3) 支持程序的嵌套和递归调用，支持中断处理。在程序调用过程中，要保存返回地址，保存程序现场，并向子程序传送参数。在堆栈计算机中，可以把这些信息都压入堆栈；当

从子程序返回时,可以直接从堆栈中弹出所需要的信息。这样可以减少大量的辅助操作,加快运算速度。

堆栈计算机的主要缺点是运算速度比较慢,这是由于堆栈与处理机之间的信息传送量大造成的。

17. 存储器堆栈栈指针的修改

从主存中划出一段区域来作堆栈是最合算且最常用的方法,这种堆栈又称为软堆栈。堆栈的大小可变,栈底固定,栈顶浮动,需要一个专门的硬件寄存器作为堆栈栈顶指针 SP(简称栈指针),栈指针所指定的存储单元就是堆栈的栈顶,当堆栈中的数据发生变化时,栈指针会改变。

对于自底向上生成的堆栈,其栈底地址大于栈顶地址,如果栈指针始终指向栈顶的满单元,则进栈和出栈的两种操作如下。

进栈时,栈指针的内容需要先自动减量,然后再将数据压入堆栈,例如:

(SP)-1→SP	修改栈指针
(A)→(SP)	将 A 中的内容压入栈顶单元

出栈时,需要先将堆栈中的数据弹出,然后栈指针的内容再自动增量,例如:

((SP))→A	将栈顶单元内容弹出,送入 A 中
(SP)+1→SP	修改栈指针

在上述两种操作中,A 为寄存器或存储单元地址;(SP)表示栈指针的内容,即栈顶单元地址;((SP))表示栈顶单元的内容。式中的 1 代表栈指针增加或减少的量,它是由压入堆栈的数据字长决定的,若存储器按字节编址,压入堆栈的数据为 32 位,则修改栈指针时应该是(SP)－4→SP。

对于自顶向下生成的堆栈(栈底地址小于栈顶地址),修改栈指针时的增、减量正好与上述操作相反。如果栈指针始终指向栈顶的空单元,则进栈和出栈两种操作的顺序也要颠倒过来。

堆栈操作既不是在堆栈中移动它所存储的内容,也不是把已存储在堆栈中的内容从堆栈中抹掉,而是通过调整栈指针来给出新的栈顶位置,以便对位于栈顶单元的数据进行操作。

18. 程序控制类指令的特点

程序控制类指令用于控制程序的执行顺序,主要包括转移指令、转子指令和返回指令等。

转移指令又分为条件转移指令和无条件转移指令两种。无论是条件转移还是无条件转移,都需要给出转移地址。若采用相对寻址方式,转移地址为当前指令地址(即 PC 的值)和指令中给出的位移量之和,即(PC)＋位移量→PC;若采用绝对寻址方式,转移地址由指令的地址码字段直接给出,即 A→PC。

子程序是一组可以公用的指令序列,只要知道子程序的入口地址就能调用它。转子(子程序调用)指令安排在主程序中需要调用子程序的地方。转子指令需要给出转移的目的地址(子程序的入口地址),所以它必定是一地址指令。

转移指令和转子指令都可以改变程序的执行顺序,但两者的主要差别如下:

(1) 转移指令使程序转移到新的地址后继续执行指令,不存在返回的问题,所以没有返回地址;而转子指令要考虑返回问题,所以必须以某种方式保存返回地址(转子指令的下一条指令的地址),以便返回时能找到原来的位置。

(2) 转移指令用于实现同一个程序内的转移;而转子指令转去执行一段子程序,实现的是程序段与程序段之间的转移。

子程序可以再调用别的子程序,这称为子程序的嵌套调用,子程序还可以调用自己,这称为子程序的递归调用。直接调用自己的递归调用称为直接递归调用,间接调用自己的递归调用称为间接递归调用。

从子程序转向主程序的指令称为返回指令,子程序的最后一条指令一定是返回指令。返回地址存放的位置决定了返回指令的格式,返回地址通常保存在堆栈中,所以返回指令常是零地址指令。对于没有堆栈的计算机,返回地址也可以保存在其他地方。例如,可以用子程序的第一个字单元来保存返回地址。转子指令把返回地址存放在子程序的第一个字单元中,子程序从第二个字单元开始执行。返回时将第一个字单元地址作为间接地址,采用间址寻址方式返回主程序,此时的返回指令必须是一条一地址指令。

19. 条件转移指令的转移条件

绝大多数算术运算指令都会影响状态标志位。通常的状态标志位有进位/借位标志(CF)、零标志(ZF)、符号标志(SF)、溢出标志(OF)和奇偶标志(PF)等。

运算类指令除常见的加、减、乘、除指令外,还包括比较指令。比较指令(CMP)与减法指令(SUB)都执行减法操作,但前者不保留运算结果,只是改变状态标志位,而后者不仅要保留运算结果,也要改变标志位。

表 3-3 给出了在无符号数比较和带符号数比较两种情况下其状态标志位反映的两数大小关系。从表 3-3 中可以看出,对无符号数和带符号数,根据状态标志位来判断两数大小的条件是不同的:前者依据 CF 和 ZF 进行判断,后者则依据 ZF、SF 和 OF 进行判断。例如,要判断 $A<B$ 是否成立,无符号数所用的条件是 ZF=0,CF=1,而带符号数所用的条件是 ZF=0,OF+SF=1。

表 3-3 状态标志反映的两数关系

两数比较结果		受影响的状态标志位			
		CF	ZF	SF	OF
$A=B$(Equal)		0	1	0	0
无符号数	$A<B$(Below)	1	0	—	—
	$A>B$(Above)	0	0	—	—
带符号数	$A<B$(Less)	—	0	1	0
		—	0	0	1
	$A>B$(Greater)	—	0	1	1
		—	0	0	0

CMP 指令常用于比较两个数,后跟条件转移指令进行程序控制转移。条件转移必须受

到条件的约束,若条件满足时才执行转移,否则程序仍按指令顺序执行。但需要注意,正因为判断无符号数和带符号数比较结果的条件不同,所以条件转移指令也分无符号数和带符号数两类,见表3-4。

表3-4 条件转移指令

指令类型	转移条件	含义
无符号数条件转移指令	CF=1	有进位转移(与低于/不高于或等于转移重叠)
	CF=0	无进位转移(与高于或等于/不低于转移重叠)
	PF=1	奇偶位为1转移
	PF=0	奇偶位为0转移
	CF=ZF=0	高于/不低于或等于转移
	CF=0	高于或等于/不低于转移
	CF=1	低于/不高于或等于转移
	CF=1 或 ZF=1	低于或等于/不高于转移
	ZF=1	等于/为零转移
	ZF=0	不等于/非零转移
带符号数条件转移指令	OF=1	溢出转移
	OF=0	无溢出转移
	SF=1	为负数转移
	SF=0	为正数转移
	ZF=0 且 SF=OF	大于/不小于或等于转移
	SF=OF	大于或等于/不小于转移
	SF≠OF	小于/不大于或等于转移
	ZF=1 或 SF≠OF	小于或等于/不大于转移
	(CX)=0	CX寄存器为0转移

20. 输入输出指令的特点

输入输出指令用来实现主机与外部设备之间的信息交换,包括输入输出数据、主机向外设发控制命令或外设向主机报告工作状态等。各种不同计算机的输入输出操作的实现方式差别很大,通常有下面两种方式。

1) 独立编址

外设寄存器和主存单元分别独立编址。在这种方式下,使用专门的输入(IN)指令和输出(OUT)指令,指令中必须给出外设的编号(端口地址)。

2) 统一编址

外设寄存器和主存单元统一编址。在这种方式下,不需要专门的输入输出指令,就用一般的数据传送类指令来实现输入输出操作。一个外设通常至少有两个寄存器(数据寄存器和命令/状态寄存器),每个外设寄存器都有唯一的地址,CPU可以像访问主存一样去访问

外设的寄存器。

3.3 典型例题详解

【例 3.1】 比较寄存器-寄存器型指令与存储器-存储器型指令的优缺点。

解：在寄存器-寄存器(R-R)型指令系统中，运算类指令中不含存储器型操作数。机器在执行这类指令的过程中，只对寄存器中的操作数进行操作，从寄存器中取操作数，结果也放在寄存器中，不需要访问存储器，因此速度很快。但是，由于所有数据来源于存储器并且最终要存放到存储器中，故寄存器-寄存器型指令的执行需要使用访问存储器指令来从存储器中存取操作数。

在存储器-存储器(M-M)型指令系统中，部分运算类指令的操作数都存放在存储器中。两个以上的操作数存放在存储器中，使得指令的执行需要 4 次以上的访问存储器(1 次取指，2 次读操作数，1 次写操作数)，这导致指令执行速度缓慢。这种指令可以不使用寄存器，从这个角度看，CPU 的实现代价相对较低。

【例 3.2】 某机字长 32 位，指令为单字长，指令系统中具有二地址指令、一地址指令和零地址指令各若干条。已知每个地址码长 12 位，采用扩展操作码方式。该指令系统中的二地址指令、一地址指令、零地址指令各最多能有多少条？

解：扩展操作码的指令字长是固定的，但指令中操作码的位数不固定。设指令字长为 32 位，每个地址码为 12 位，因此，零地址指令的操作码占 32 位，一地址指令的操作码占 20 位，二地址指令的操作码占 8 位。

对于二地址指令，至少需要留出一个扩展窗口给一地址指令。显然最多可以有 2^8-1 条二地址指令。

对于一地址指令，最多的情况是指令系统中只有一条二地址指令，并要为零地址指令留出一个扩展窗口。显然最多可以有 $(2^8-1)\times 2^{12}-1$ 条一地址指令。

对于零地址指令，最多的情况是指令系统中只有一条二地址指令和一条一地址指令，其余均为零地址指令。零地址指令的操作码为 32 位，共有 2^{32} 种编码，其中 2^{24} 种编码用作表示某条二地址指令，2^{12} 种编码用作表示某条一地址指令。因此零地址指令最多可以有 $2^{32}-2^{24}-2^{12}$ 条。

【例 3.3】 指令字长为 12 位，每个地址码为 3 位，采用扩展操作码的方式，设计 4 条三地址指令、16 条二地址指令、64 条一地址指令和 16 条零地址指令。

(1) 给出一种操作码的扩展方案。

(2) 画出指令译码逻辑。

(3) 计算操作码的平均长度。

解：(1) 操作码的扩展方案如图 3-3 所示。

(2)指令译码逻辑如图 3-4 所示。

(3) 操作码的平均长度＝全部指令的操作码长度÷指令总数＝(4×3＋16×6＋64×9＋16×12)÷

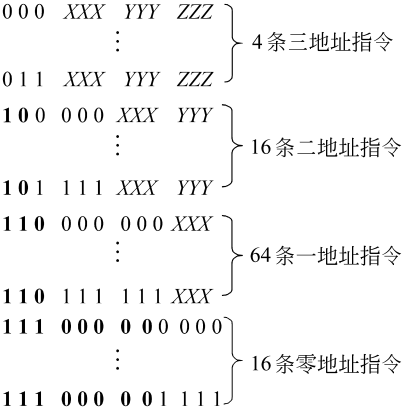


图 3-3 操作码的扩展方案

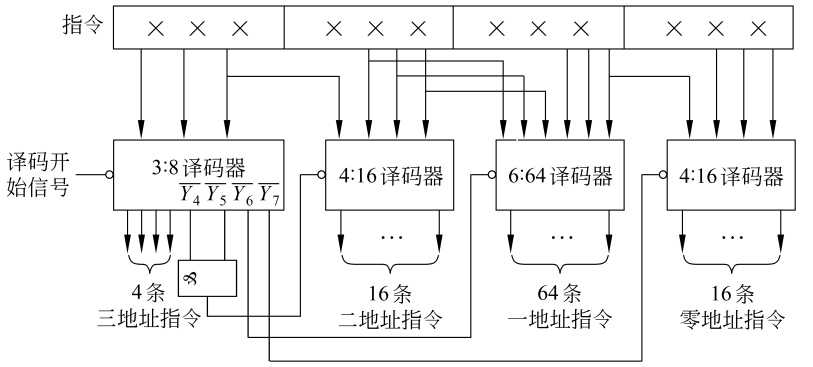


图 3-4　指令译码逻辑

(4+16+64+16)=8.7。

【例 3.4】 在 16 位长的指令系统中,设计一个扩展操作码,能对下列指令进行译码。

(1) 7 条三地址指令。

(2) 225 条一地址指令。

(3) 16 条零地址指令。

令每个地址码为 4 位,分别写出上述 3 种指令的格式,并说明译码过程。

解:(1) 7 条三地址指令:

0000 *XXXX YYYY ZZZZ*

⋮

0110 *XXXX YYYY ZZZZ*

(2) 以 1000 为扩展窗口的一地址指令最多可以有 256 种不同的操作码编码,现在只有 225 条指令,多余的 31 种编码为非法操作码。

	11111111	最后一个操作码编码
−	11111	31 种非法操作码
	11100000	最后一条一地址指令的操作码

225 条单地址指令:

1000 0000 0000 *XXXX*

⋮

1000 1110 0000 *XXXX*

(3) 16 条无地址指令:

1111 0000 0000 0000

⋮

1111 0000 0000 1111

上述 3 种指令的格式如图 3-5 所示。为了译码的方便,特别安排三地址指令的识别标志是指令的最高位为 0,接着再对剩余的 3 位操作码译码,可区别出 7 条不同的三地址指令。一地址和零地址指令的最高位都为 1。如果次高位为 0,则是一地址指令,接着将指令的中间 8 位作为操作码译码,可以区分出 225 条一地址指令;如果次高位仍为 1,则是零地

址指令，接着对指令的最低 4 位进行译码，可以区分出 16 条零地址指令。

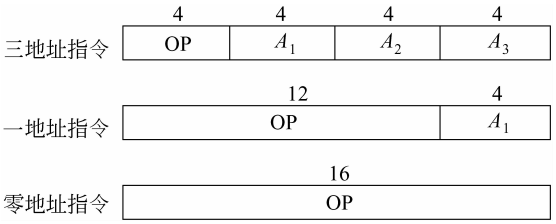


图 3-5　3 种指令的格式

【例 3.5】 假定一个 32 位微处理器的指令字长为 32 位，每条指令由两部分组成，其中第一个字节为操作码，剩余的为立即数或操作数地址。

（1）可直接访问的最大主存空间是多少？

（2）讨论下列两种情况对系统速度的影响：微处理器总线使用 32 位局部地址总线和 16 位局部数据总线；微处理器总线使用 16 位局部地址总线和 32 位局部数据总线。

（3）程序计数器和指令寄存器各需要多少位？

解：（1）因为 32 位指令中有 8 位为操作码，其余 24 位为操作数地址，所以可直接访问的最大主存空间为 2^{24}。

（2）若采用 32 位局部地址总线和 16 位局部数据总线，则需要两个访存周期才能读取一个字的指令和数据；若采用 16 位局部地址总线和 32 位局部数据总线，则需要两个时钟周期才能把地址送出去。以上两种情况都会导致系统速度下降。

（3）指令寄存器的位数与指令字长相同，应为 32 位。而程序计数器的位数与可访问的主存空间相对应，应为 24 位。

【例 3.6】 某计算机的字长为 16 位，存储器按字编址，访存指令格式如图 3-6 所示。其中，OP 是操作码，MOD 定义寻址方式（见表 3-5），A 为形式地址。设 PC 和 R_x 分别为程序计数器和变址寄存器，字长为 16 位。

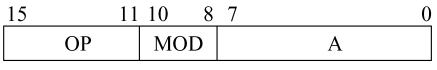


图 3-6　访存指令格式

（1）该格式能定义多少种指令？

（2）各种寻址方式的寻址范围为多少字？

（3）写出各种寻址方式的有效地址 EA 的表达式。

解：（1）由图 3-6 可知，指令格式中的高 5 位为操作码，总的指令种类为 $2^5=32$ 种。

（2）假设存储器按字编址，各种寻址方式的寻址范围见表 3-6。

表 3-5　寻址方式定义

MOD	寻址方式
0	立即寻址
1	直接寻址
2	间接寻址
3	变址寻址
4	相对寻址

表 3-6　各寻址方式的寻址范围

寻址方式	寻址范围
立即寻址	指令字本身
直接寻址	256 个字（主存中最前边的 256 个字）
间接寻址	64K 个字
变址寻址	64K 个字
相对寻址	256 个字（PC 值代表的存储单元附近的 256 个字）

(3) 各种寻址方式的有效地址 EA 的表达式如下：

寻址方式　　有效地址表达式

立即寻址　　EA=(PC)，即操作数在指令中(指令字的低位部分)

直接寻址　　EA=A

间接寻址　　EA=(A)

变址寻址　　EA=(R_x)+A

相对寻址　　EA=(PC)+A

【例 3.7】 某机字长 32 位，共有机器指令 100 条，指令单字长，等长操作码，CPU 内部有通用寄存器 32 个，可作变址寄存器用。存储器按字节编址，指令采用直接寻址、间接寻址、变址寻址和相对寻址 4 种方式。

(1) 分别画出采用 4 种不同寻址方式的一地址指令的指令格式；

(2) 采用直接寻址和间接寻址方式时，可寻址的存储器空间各是多少？

(3) 写出 4 种方式下有效地址 EA 的表达式。

解：(1) 由于系统共有 100 条指令，满足这个条件的最小指令操作码位数为 7 位；系统中允许使用 4 种不同的寻址方式，寻址方式字段需要 2 位。

指令长度为 32 位，剩余位数为 32−7−2=23 位，即为地址码长度。但在变址寻址时，还需要有 5 位寄存器编码，所以真正的地址码只有 18 位。4 种不同寻址方式的一地址指令的指令格式如图 3-7 所示。

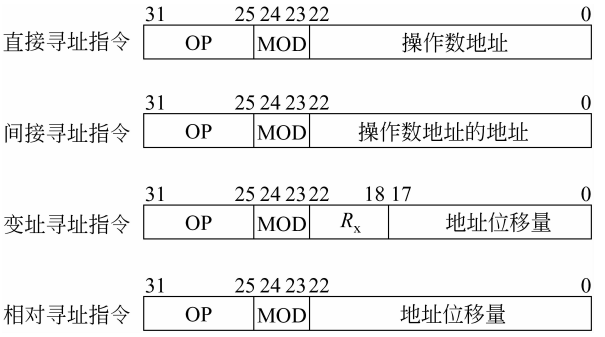


图 3-7　一地址指令的指令格式

(2) 存储器按字节编址，采用直接寻址时，可寻址空间为 8MB(2^{23}B)；采用间接寻址时，由于机器的字长为 32 位，所以可寻址空间为 4GB(2^{32}B)。

(3) 在上述 4 种寻址方式下，有效地址 EA 的表达式如下：

寻址方式　　有效地址表达式

直接寻址　　EA=A

间接寻址　　EA=(A)

变址寻址　　EA=(R_x)+A

相对寻址　　EA=(PC)+A

【例 3.8】 某机字长为 16 位，采用一地址格式的指令系统，允许直接寻址、间接寻址、变址寻址、基址寻址，变址寄存器和基址寄存器均为 16 位。

(1) 若采用单字指令，共能完成 108 种操作，写出指令格式，并指出直接寻址和一次间

址的寻址范围各为多少？

（2）若采用双字指令，操作码位数和寻址方式不变，给出指令可直接寻址的主存空间，并画出指令格式。

（3）字长不变，可采用什么方法访问容量为8MB的主存任一地址单元？说明理由。

解：（1）由于系统的指令集有108条指令，满足这个条件的最小指令操作码位数为7位；系统中允许有4种不同的寻址方式，寻址方式字段需要2位。故采用单字指令时，地址码的位数为7位。设主存按字编址，直接寻址的主存空间为2^7个字。采用一次间址寻址时，由于地址码中是操作数地址的地址，操作数的有效地址在主存内，该地址是16位的，因此，间址寻址的主存空间为2^{16}个字。指令格式如图3-8(a)所示。

（2）当采用双字指令时，指令中地址码的长度增加，所以可寻址的主存空间也加大了，为$2^{16+7}=2^{23}$个字(记为8MW)。指令格式如图3-8（b)所示。

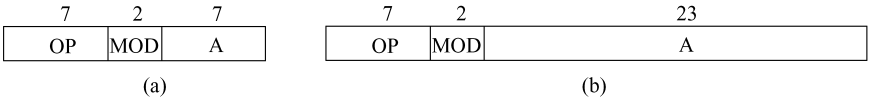


图3-8　某机的指令格式

（3）采用双字指令的直接寻址方式即可以访问8MW的主存空间，但这样每条指令需要占用两个存储字，处理上比较复杂且代价高。也可以使用变址寻址或基址寻址来访问比较大的主存空间，因为8MB=4MW，主存按字编址，物理空间需要22位地址。由于变址寄存器或基址寄存器只有16位，变址寻址或基址寻址的有效地址EA=(*R*)+*A*，可以将变址寄存器或基址寄存器中的内容左移6位之后再与*A*相加。

【例3.9】 某计算机有变址寻址、间接寻址和相对寻址等寻址方式。设当前指令的地址码部分为001AH，正在执行的指令所在地址为1F05H，变址寄存器中的内容为23A0H。

（1）当执行取数指令时，如为变址寻址方式，则取出的数为多少？

（2）当执行取数指令时，如为间接寻址，取出的数为多少？

（3）当执行转移指令时，转移地址为多少？

已知部分存储单元地址及相应内容，见表3-7。

表3-7　部分存储单元地址与内容

地　址	内　容
001AH	23A0H
1F05H	241AH
1F1FH	2500H
23A0H	2600H
23BAH	1748H

解：（1）这是一种数据寻址(变址寻址)，操作数$S=((R_x)+A)=$(23A0H+001AH)=(23BAH)=1748H。

（2）这也是一种数据寻址(间接寻址)，操作数$S=((A))=((001AH))=$(23A0H)=2600H。

（3）这是一种指令寻址，转移指令使用相对寻址，转移地址=(PC)+*A*=1F05H+001AH=1F1FH。

因为在本例的题目中没有指出指令的长度，故未考虑PC值的更新。

【例3.10】 一条双字长的LOAD指令存储在地址为200和201的存储单元，该指令将

指定的内容装入累加器(AC)中。指令的第一个字指定操作码和寻址方式,第二个字是地址码。寄存器与主存内容示意图如图3-9所示。

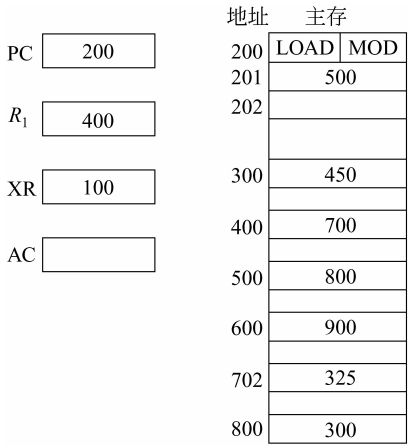


图3-9 寄存器与主存内容示意图

指令的寻址方式字段可指定任何一种寻址方式。在以下几种寻址方式中,装入AC的值是多少?

(1) 直接寻址;

(2) 立即寻址;

(3) 间接寻址;

(4) 相对寻址;

(5) 变址寻址;

(6) 寄存器 R_1 寻址;

(7) 寄存器 R_1 间接寻址。

解:(1) 在直接寻址方式下,有效地址是指令中的地址码500,装入AC的操作数是800。

(2) 在立即寻址方式下,指令的地址码是操作数而不是地址,所以将500装入AC。

(3) 在间接寻址方式下,操作数的有效地址存储在地址为500的存储单元中,由此得到有效地址是800,操作数是300。

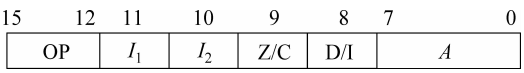
(4) 在相对寻址方式下,有效地址EA=(PC)+A=202+500=702,所以操作数是325。这是因为指令是双字长,在该指令的执行阶段,PC已经加2,更新为下一条指令的地址202。

(5) 在变址寻址方式下,有效地址EA=(XR)+A=100+500=600,所以操作数是900。

(6) 在寄存器寻址方式下,R_1 的内容400装入AC。

(7) 在寄存器间接寻址方式下,有效地址是 R_1 的内容400,装入AC的操作数是700。

【例3.11】 假定指令格式如下:



其中:

I_1=1表示变址寄存器 I_1 寻址;

I_2=1表示变址寄存器 I_2 寻址;

Z/C=1表示当前页寻址;

D/I=0表示直接寻址,D/I=1表示间接寻址。

主存容量为 2^{16} 个存储单元,分为 2^8 个页面,每个页面有 2^8 个字。

设有关寄存器的内容为

$$(I_1)=35A7H \quad (I_2)=1B28H \quad (PC)=46C9H$$

计算下列指令的有效地址:

(1) D4C1H;

(2) 780BH;

(3) E253H;

(4) C009H。

解:首先将十六进制表示的指令写成二进制形式,并根据有关的标志位确定不同的寻址方式,最后计算出有效地址。

(1) D4C1H=1101010011000001B。

I_2=1,变址寄存器 I_2 寻址,EA=(I_2)+A=1B28H+C1H=1BE9H。

(2) 780BH=0111100000001011B。

I_1=1,变址寄存器 I_1 寻址,EA=(I_1)+A=35A7H+0BH=35B2H。

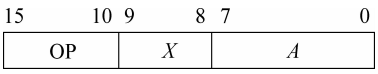
(3) E253H=1110001001010011B。

Z/C=1,当前页寻址。因为主存被分为 2^8 个页面,每个页面有 2^8 个字,故页面地址8位,来自PC的高8位,页内地址8位,来自指令中的形式地址 A。EA=$(PC)_H$ // A=46 // 53=4653H。

(4) C009H=1100000000001001B。

有关的标志位均为0,直接寻址,EA=A=0009H。

【例 3.12】 某机的指令格式如下:



其中,X 为寻址特征位。X=0 时不变址;X=1 时用变址寄存器 X_1 进行变址;X=2 时用变址寄存器 X_2 进行变址;X=3 时为相对寻址。设(PC)=1234H,(X_1)=0037H,(X_2)=1122H。确定下列指令的有效地址(均用十六进制表示)。

(1) 4420H;

(2) 2244H;

(3) 1322H;

(4) 352BH。

解:(1) 指令 4420H 写成二进制为 0100 0100 0010 0000。

X=00,不变址,即直接寻址,EA=A=0020H。

(2) 指令 2244H 写成二进制为 0010 0010 0100 0100。

X=10,用变址寄存器 X_2 进行变址,EA=(X_2)+A=1122H+44H=1166H。

(3) 指令 1322H 写成二进制为 0001 0011 0010 0010。

X=11,为相对寻址,EA=(PC)+A=1234H+22H=1256H(未考虑PC值的更新)。

(4) 指令 352BH 写成二进制为 0011 0101 0010 1011。

X=01,用变址寄存器 X_1 进行变址,EA=(X_1)+A=0037H+2BH=0062H。

【例 3.13】 设相对寻址的转移指令占4字节,其中,第1、2字节是操作码,第3、4字节是相对位移量(用补码表示)。

(1) 设当前PC的内容为2003H,要求转移到200AH的地址,则该转移指令第3、4字节的内容应为多少?

(2) 设当前PC的内容为2008H,要求转移到2001H的地址,则该转移指令第3、4字节的内容应为多少?

解：由于指令占4字节，取指令之后PC加4。

(1) 第3、4字节的内容为200AH－(2003H＋4)＝3(补码表示为0003H)。

(2) 第3、4字节的内容为2001H－(2008H＋4)＝－11(补码表示为FFF5H)。

【例3.14】 以主存地址7EA8H为首地址存放了一条两字节指令。其第一字节为操作码OP，是转移指令；第二字节为相对寻址的位移量D，它是一个8位补码(可正可负)。

(1) 位移量D的表示范围是什么？

(2) 该指令的转移空间范围是什么？

解：(1) 位移量D是8位补码，表示范围为－128～127，用十六进制表示为80H～7FH。

(2) 该指令的转移空间是相对于取出该指令之后的PC值7EA8H＋2＝7EAAH计算的，7EAAH－80H＝7E2AH，7EAAH＋7FH＝7F29H，所以转移空间为7E2AH～7F29H。

【例3.15】 某个自底向上生成的存储器堆栈，栈指针SP始终指向栈顶的满单元。栈底地址为3000H，栈中已压入两个数据a和b。

(1) 画出此时堆栈的示意图。

(2) 若现在将数据c和d按顺序压入堆栈，写出这两个数据进栈的操作步骤，并画出数据进栈之后堆栈的示意图。

(3) 写出数据d出栈的操作步骤。

注：设数据交换通过累加器AC进行。

解：(1) 数据a和b进栈之后堆栈的示意图如图3-10(a)所示。

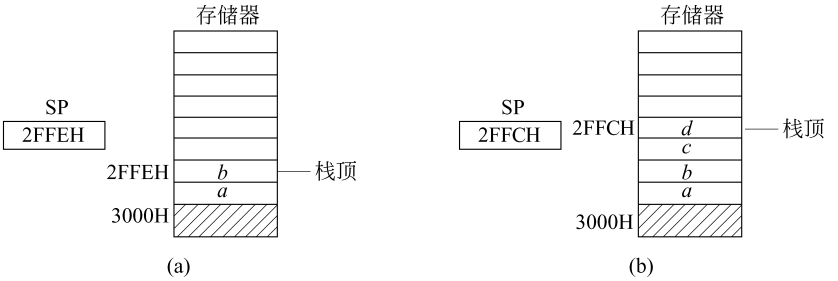


图3-10　堆栈的示意图

(2) 将数据c压入堆栈的操作步骤为

c→AC	数据c放入AC
(SP)－1→SP	栈指针减1
(AC)→(SP)	AC的内容压入栈顶单元

将数据d压入堆栈的操作步骤为

d→AC	数据d放入AC
(SP)－1→SP	栈指针减1
(AC)→(SP)	AC的内容压入栈顶单元

这两个数据进栈后堆栈的示意图如图3-10(b)所示。