# 第5章　指令系统

指令是控制计算机执行某种操作的命令，是计算机硬件实体直接表征控制信息的语言。指令系统是一台计算机所能执行的全部指令的集合，是计算机体系结构的核心，是计算机系统硬件、软件的主要连接面。它既是计算机硬件设计的主要依据，又是软件编程的基石。指令系统是表征一台计算机性能的重要因素。指令系统设计的是否合理，直接关系到计算机硬件结构的复杂程度，也关系到程序设计的支持程度和效率。设计指令系统首先要确定计算机的哪些功能由一条基本指令实现（即硬件实现），哪些功能由程序实现（即软件实现），然后确定指令的编码规则。

本章简要介绍指令系统的发展和性能要求，以及指令系统中的常用功能的指令，重点介绍指令系统设计的各种要素及寻址方式，简要说明 RISC 的概念和基本特征。

## 5.1　概　　述

计算机程序是人们在解决实际问题时，按一定逻辑排列的一串指令或语句序列。从计算机组成的层次结构来说，计算机的指令有微指令、机器指令和宏指令之分。微指令是微程序控制器实现机器指令功能的命令，它完全属于计算机硬件，是计算机硬件设计人员使用的。宏指令或高级语言语句的功能是要通过若干条机器指令组成的程序段来完成，属于软件。机器指令是计算机硬件能够直接完成的基本操作命令，又可以被系统程序员使用，它介于硬件和软件之间，是设计一台计算机硬件和底层软件（系统软件）的接口。

### 5.1.1　指令系统的发展

指令系统的发展经历了从简单到复杂的演变过程。

20 世纪 50 年代，计算机大多数采用分立元件的晶体管或电子管组成，体积庞大、价格昂贵，计算机的硬件结构比较简单，所支持的指令系统只有十几至几十条最基本的指令，寻址方式也十分简单。到 20 世纪 60 年代中期，随着集成电路的出现，计算机的功耗、体积、价格等不断下降，硬件功能不断增强，指令系统也越来越丰富。除了以上基本指令外，还设置了乘除运算、浮点运算、十进制运算、字符串处理等功能的指令。指令数目多达一二百条，寻址方式也开始多样化。

随着集成电路的发展和计算机应用领域的不断扩大，20 世纪 60 年代后期开始出现系列计算机。所谓系列计算机，是指基本指令系统相同、基本体系结构相同的一系列计算机。例如 Pentium 系列是当前流行的一种个人计算机系列。系列机由于推出的时间不同，采用的器件不同，它们在结构和性能上有所差异，新推出机种的指令系统要包含所有旧机种的全部指令。这样，系列机解决了各机种间的软件兼容问题，大大减少了软件开发的费用。

20 世纪 70 年代，高级语言已成为大、中、小型机的主要程序设计语言，计算机应用日益普及。由于软件的发展超过了软件设计理论的发展，复杂的软件系统设计一直没有很好的

理论指导，导致软件质量无法保证，从而出现了所谓的“软件危机”。人们认为，缩小机器指令系统与高级语言的语义差距，为高级语言提供很多的支持，是缓解软件危机有效和可行的办法。计算机设计者们利用当时已经成熟的微程序技术和飞速发展的 VLSI 技术，增设各种各样复杂的、面向高级语言的指令。指令系统的规模越来越庞大，大多数计算机的指令系统多达几百条指令。按这种思想设计的计算机系统称为复杂指令系统计算机(Complex Set Instruction Computer，CISC)。但是如此庞大的指令系统不但使计算机的研制周期变长，而且由于采用了大量使用频率很低的复杂指令，造成硬件资源的浪费，产生指令集所谓的“二八”现象，即最常使用的简单指令仅占指令总数的 20%，它们在程序中出现的频率却达 80%。相对应，另外 80%的指令在程序中出现的频率只有 20%。为此，人们提出了便于 VLSI 技术实现的精简指令系统计算机(Reduced Instruction Set Computer，RISC)。

### 5.1.2 指令系统的性能要求

指令系统的性能决定了计算机的基本功能，它的设计直接关系到计算机的硬件结构和用户的需要。一个完善的指令系统应满足如下 4 个方面的要求。

(1) 完备性。完备性是指用汇编语言编写各种程序时，指令系统直接提供的指令足够使用。完备性要求指令系统丰富、功能齐全、使用方便。

一台计算机最基本的指令并不多，许多功能都可用这些指令编程来实现。例如，乘除运算、浮点运算功能可设置为一条专用指令，即硬件直接实现，也可以用基本指令编写的程序来实现。指令系统中不设置此类指令的目的是提高程序执行速度，为用户编写程序提供方便。

(2) 有效性。有效性是指利用该指令系统所编写的程序能够高效率地运行。高效率主要表现在程序占据存储空间小、执行速度快。一般来说，指令系统的功能越强大、越完善，其有效性会越好。

(3) 规整性。规整性包括指令系统的对称性、匀齐性、指令格式和数据格式的一致性。

对称性是指在指令系统中所有的寄存器和存储器单元都可同等对待，所有的指令都可使用各种寻址方式。例如，传送指令可以实现寄存器之间的数据传送，也应该可以在存储器之间传送数据。

匀齐性是指一种操作性质的指令可以支持各种数据类型。机器语言操作的数据类型包括定点的字节型、字型、双字型和浮点数据。

指令格式和数据格式的一致性是指指令长度和数据长度有一定的关系，以方便处理和存取。一般来说，指令长度和数据长度是字节长度的整数倍。

(4) 兼容性。系列机之间由于具有相同的基本结构和共同基本指令集，但新推出的机种在结构和性能上会有差异，会在旧机种的基础上增强一些功能指令，因此做到所有软件在不同机种之间完全兼容是不现实的，但要能做到“向上兼容”，即低档机上运行的软件可以在高档机上运行。

在设计指令系统，确定指令系统的规模、功能以及格式时，要综合考虑上述 4 个方面，一般要满足兼容性，尽可能做到完备性和有效性。对于规整性，要综合考虑指令格式、指令字长和执行速度多方面，适当进行取舍。

### 5.1.3 低级语言与硬件结构的关系

实现计算机程序的语言有高级语言和低级语言之分。高级语言又称算法语言，它侧重于程序的算法，其语句与具体机器的指令系统无关。低级语言又分机器语言和汇编语言，机器语言中的语句是用指令的二进制编码，书写、阅读和调试程序很不方便；汇编语言是一种符号语言，将机器语言的指令编码用助记符的形式表示，例如 Intel 8086 的汇编语句

```
MOV AX,BX
```

表示“将 AX 寄存器的内容传送给 BX 寄存器”，较二进制指令编码容易理解多了。汇编指令与机器指令一一对应，和机器的指令系统有关，都是面向机器的语言。

高级语言程序中的语句要翻译为机器语言语句后才能由计算机的硬件执行，这个翻译由相应的软件（如编译程序）完成。一般来说，一条高级语言的语句要翻译成一段机器指令代码。汇编语言源程序也要通过汇编程序翻译为机器语言程序，才能由计算机硬件直接执行。

指令系统中的每条指令都能够由计算机的硬件直接执行完成其功能，它与计算机的硬件结构有密切的关系。指令系统确定后，完成指令操作的相应硬件功能基本确定，因此它直接影响着计算机的硬件结构设计。同样，计算机的硬件结构一定，其指令系统的功能也基本确定。二者相互依赖，相互影响。机器语言和汇编语言可以直接对计算机的硬件资源进行访问，例如直接访问主存单元或寄存器，而高级语言不可以。为了克服这一缺陷，一些高级语言（如 C 语言）提供了与汇编语言之间的调用接口，可以在高级语言程序中嵌入汇编语言程序段。

## 5.2 机器指令的设计

计算机设计的一个重要方面就是指令系统的设计。指令系统对计算机系统有多方面的影响，其设计也涉及多个方面。指令系统定义了 CPU 应完成的多数功能，对 CPU 的实现有很大的影响；另一方面，程序员是通过指令系统控制 CPU 的，设计指令系统还要考虑程序设计的要求。

设计指令系统首先要确定指令系统所实现的基本功能，然后确定指令的具体格式、类型以及对操作数的访问方式。

（1）操作指令表是指应提供多少、什么样的操作以及操作的复杂程度。

（2）数据类型是指所支持数据的类型。

（3）指令格式是指指令的（位）长度、操作数地址的数目、各个字段的大小等。

（4）寄存器是指能被指令访问的寄存器数目以及它们的用途。

（5）寻址方式是指指定操作数的产生方式。

这几个方面是紧密相关的，设计指令系统时要综合考虑。

本节先简要介绍操作指令设计的一般思想，然后重点介绍指令的基本格式及操作码和地址码设计中的一般问题和相关技术。寻址方式的相关概念和技术在 5.3 节介绍。

### 5.2.1 指令操作

设计指令系统的操作功能首先要考虑到其完备性。下面通过一个简单的例子说明。

在高级语言中,一个简单的赋值语句:

$$X=X+Y$$

其功能为,变量 $X$ 的值和 $Y$ 的值相加,并将结果存入变量 $X$ 中。机器语言如何完成它呢?

假定 $X$ 和 $Y$ 变量对应的存储单元为 2000H 和 2001H,一般的指令系统中,算术运算指令的两个操作数不能都在存储单元中,那么完成上述赋值语句需要以下 2 条指令:

(1) 将 2001H 存储单元的内容装入一个寄存器。

(2) 将 2000H 存储单元的内容加上述寄存器的内容,并将结果存入 2000H 存储单元。

这样,指令系统就必须有"存储器传送到寄存器"和"存储器与寄存器相加,结果存入存储器"的功能指令。

上述简单例子启示,一个具体的计算机中,必须有供用户表达任何数据处理任务的一组基本指令。考虑到任何高级语言编写的程序,最终都必须转换为机器语言才能执行。因此机器指令的集合必须充分、完备,足以表达任何高级语言语句的功能。

一般来说,指令系统应该包含以下 6 种类型的指令。

(1) 数据处理类,提供处理数值型或其他类型数据的各种算术、逻辑运算及移位操作指令。

(2) 数据存储类,实现数据在 CPU(寄存器)与存储器之间的传送。

(3) 数据传送类,完成 CPU 内部寄存器之间数据的传递。

(4) 输入输出类,完成外围设备接口与主机之间的数据交换。

(5) 程序控制类,实现程序执行顺序的控制。

(6) 其他系统类指令。

各种类型指令的功能详细介绍见 5.4 节。

### 5.2.2 机器指令的基本格式

指令是控制计算机执行某种操作的命令,一条指令就是机器语言的一个语句,它是一组有意义的二进制编码。CPU 执行程序就是将指令的二进制编码从内存中取出后,由控制器产生完成该指令功能的控制信号,控制计算机各功能部件完成指令的执行过程。因此组成指令的二进制编码中一般应包含这样两部分信息:一部分告诉 CPU 指令要进行何种操作,即操作的性质及功能;另一部分给出要操作的对象所在的位置。前一部分人们称之为操作码字段,后一部分称为地址码字段。即指令的一般格式如图 5-1 所示。


图 5-1 指令的一般格式

指令操作的对象包括操作的数据或下条要执行的指令的地址,统称为操作数。操作数可能存放的位置一般有以下 3 种。

(1) 操作数在指令的代码中直接包含,即地址码字段直接给出操作数。如果指令长度是一个存储字长,CPU 在取指令的代码时一起将操作数读入 CPU。如果指令长度是多个

存储字长，连续取多个存储字得到操作数。

(2) 操作数在 CPU 的寄存器中。一般来说，CPU 中总有一个或多个能被机器指令访问的寄存器。如果只有一个寄存器，对它的引用可以是隐式的，若不只一个，则每个寄存器要有一个对应的编号。由指令的地址码部分提供操作数所在寄存器的编号。

(3) 操作数在内存中。指令的地址码部分提供操作数所在存储单元地址的相关信息。

(4) 操作的数据还可能来自外围设备接口的端口寄存器，此时地址码部分是端口地址的有关信息。

合理的指令格式应该给出足够的信息，其长度又尽可能与机器字长匹配，以节省存储空间，缩短取指令时间。指令格式尽量规整，减少硬件译码的复杂程度。

**说明**：不同计算机，即使是指令功能相同，其二进制编码也会不同。为了叙述方便，特别是后续寻址方式举例时，指令格式中的操作码或寄存器等均采用助记符表示，即汇编指令格式。例如操作码部分，使用 ADD 表示“加”，SUB 表示“减”，MOV 表示“传送”等。

### 5.2.3 指令字长

指令字长是指一条指令中包含的二进制编码的位数。指令的长度与操作码的长度、操作对象的个数和位数、存储单元的地址长度等多种因素有关。为了充分利用存储空间，节省取指令的时间，指令的长度通常也设计为字节长度的整数倍。

指令字的长度与机器的字长没有固定的关系。机器字长是指运算器能一次直接处理的二进制数的位数，为了处理字符数据，尽可能利用存储空间，一般机器字长都是字节长度的整数倍，即 8，16，32 或 64。指令字长可以是机器字长的一半、1 倍或多倍。例如，Intel 8086 为 16 位字长，其指令格式有 8 位、16 位、24 位、32 位等多种格式。采用多字长指令，便于解决访问内存任何单元的寻址问题，但是程序占用的内存空间增加，取指令时间较长，也影响程序执行速度。

如果一个指令系统中各种指令的长度是相同的，称为等长指令字结构。如果指令功能不同，相应的指令字长不同，例如 Intel 8086 的指令系统就是变长指令字结构。变长指令字结构便于充分利用指令长度，但指令的控制较复杂。现代计算机广泛采用变长指令字结构，指令的长度能短则短，需长则长。

### 5.2.4 地址码结构

地址码用来描述指令的操作对象。地址码的长度与操作对象的个数、位数及存储单元的地址长度、寻址方式等多种因素有关。

不同操作的指令所需要的操作对象的个数不同，地址码字段所需要的地址数不同。算术和逻辑运算指令要求的操作数最多。一般的双操作数运算指令，其地址数最多会有 4 个。后续叙述中，用 $A_1$ 表示第一操作数地址，用 $A_2$ 表示第二操作数地址，用 $A_3$ 表示操作结果存放地址，用 $A_4$ 表示给出下条要执行指令的地址。

这些信息可以在指令中明显给出，称为显地址。也可以依照某种约定，用隐含的方式给出，称为隐地址。

**说明**：这里所说的操作数地址是广义的概念，对于操作对象所在位置不同，它可能直接是操作的数据、寄存器的编码或者存储单元的地址。

根据地址码所涉及的地址数量，常见的指令格式有以下几种。

(1) 四地址指令。4 个地址信息都在地址码字段中显式给出。其指令格式如图 5-2 所示。

| OP | $A_1$ | $A_2$ | $A_3$ | $A_4$ |
|---|---|---|---|---|

图 5-2　四地址指令的格式

指令的含义：$A_3 \leftarrow (A_1)\mathrm{OP}(A_2)$，$A_4$ 为要执行的下条指令的地址。

**说明**：一般用 $A_i$ 表示地址，$(A_i)$表示存放在该地址中的内容。

这种格式的主要优点是直观，但指令字的长度太长，如果每个地址为 16 位，整个地址码字段就要长达 64 位，所以这种格式是不切实际的。

(2) 三地址指令。正常情况下，大多数指令按顺序依次从内存取出来执行，只有遇到转移指令时程序的执行顺序才会改变，需要专门给出要取下一条指令的地址。因此，通常在 CPU 中设置一个程序计数器(Program Counter，PC)存放指令的地址，每取一条指令，PC 就自动加 1(假设每条指令只占一个存储单元)。因此，指令格式中一般不再给出 $A_4$。指令格式变成三地址，如图 5-3 所示。

| OP | $A_1$ | $A_2$ | $A_3$ |
|---|---|---|---|

图 5-3　三地址指令的格式

这种格式的指令字仍比较长，一般只用在大、中型计算机中，小型或微型计算机中很少使用。

(3) 二地址指令。三地址指令执行完后，参与操作的两个操作数均不会被破坏，可供再次使用。通常情况下，并不需要完整的保留两个操作数，可以让第一个操作数地址兼做存放结果的地址，这样得到二地址格式，如图 5-4 所示。

| OP | $A_1$ | $A_2$ |
|---|---|---|

图 5-4　二地址指令的格式

其中，$A_1$ 称为目的操作数地址，$A_2$ 称为源操作数地址。该指令的含义为 $A_1 \leftarrow (A_1)\mathrm{OP}(A_2)$。

(4) 单地址指令。单地址指令的地址码字段只有一个显地址。其格式如图 5-5 所示。

| OP | $A_1$ |
|---|---|

图 5-5　单地址指令的格式

单地址指令可能只有一个操作数，例如取非、移位等某些单操作数指令；也可能是双操作数指令，$A_1$ 显式给出一个，另一个操作数(或操作结果)约定存放在某个寄存器中，即隐含寻址。因此，该指令的含义为 $A_1 \leftarrow \mathrm{OP}(A_1)$或 $A_2 \leftarrow (A_2)\mathrm{OP}(A_1)$。

例如 Intel 8086 中，乘法指令格式中只显示提供一个操作数，另一个操作数在累加器 AL 或 AX 中。助记符为 MUL $R$，功能为(AL)×($R$)→AX，其中 $R$ 为通用寄存器。

(5) 零地址指令。顾名思义，零地址指令格式中只有操作码，没有地址码。这种格式用于一些不需要操作数的指令，或者操作数隐含的指令。例如，暂停、空操作等一些系统控制指令、标志位的测试或设置指令，无操作数；而 Intel 8086 的指令系统中的 CBW 指令，其功能是将 AL 寄存器的字节扩展到 AH 寄存器，形成 16 位的字存放在 AX 寄存器中，其操作数是隐含的。

指令地址数的选择是指令格式设计的一个重要因素。直觉上，地址数越少，指令的长度越短，所占存储空间少。但是，实际上由于指令地址数太少，在完成多操作数操作功能的程序中，总指令条数可能大大增加，使得程序变长且复杂。对于多地址指令，如果操作数均在存储器中，指令执行时取操作数花费时间较长，指令的执行速度大大受到影响。因此，在设计时要综合考虑多种因素。一般来说，设计指令时，尽量不用四地址指令，很少使用三地址指令。

随着计算机技术和集成电子技术的迅速发展，许多计算机中设置了大量的通用寄存器，使用它们来存放操作数或运算结果，既缩短了指令字的长度，又兼顾到了执行速度。因此，对于二地址指令，大多数运算指令的操作数中至少一个操作数在寄存器中，尽可能两个操作数都在寄存器中，只有极少数指令的两个操作数均在存储器中。

二地址指令中，根据操作数的存储位置又分为寄存器—寄存器(RR)型、存储器—存储器(SS)型和存储器—寄存器(SR)型 3 种。

在很多计算机中，其指令系统中指令字的长度和地址结构并不是单一的，往往多种格式混合使用，以增强指令系统的性能。例如 Intel 8086/8088 指令系统的指令格式有 6 种格式，每种格式中地址码的含义有很大区别，如图 5-6 所示。

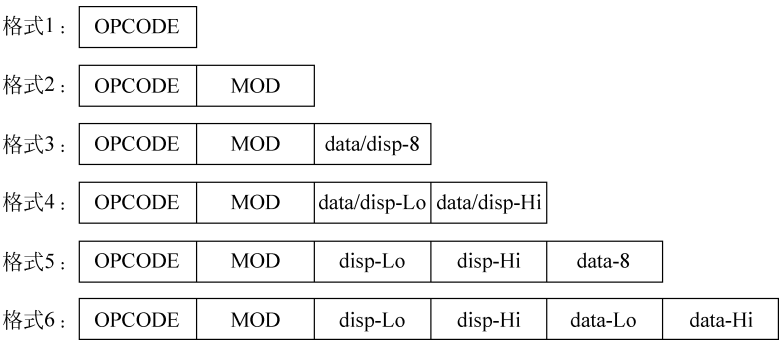



图 5-6　Intel 8086/8088 的指令格式

### 5.2.5　操作码设计

操作码是机器指令代码中不可或缺的组成部分。它用来指明操作的类型以及源操作数、目的操作数的引用方式。一台计算机可能有几十条至几百条指令，每一条指令都对应有一个操作码，CPU 的控制器通过对该操作码进行译码分析来确定指令的功能，产生相应的控制信号，控制计算机的各功能部件完成指令的操作。

指令系统中的每条指令都有一个唯一确定的操作码，设计时希望操作码字段的位数尽可少。常用的操作码编码方法有定长和变长两种方式。

**1. 定长操作码**

这是一种最简单的编码方法，其操作码字段的位数固定。操作码的位数与指令系统的规模有关。假设指令系统有 $m$ 条指令，指令中操作码字段的位数为 $N$ 位，则它们之间有如下关系：

$$m \leqslant 2^N \quad 或 \quad N \geqslant \mathrm{lb}m$$

定长操作码的指令译码规整，电路简单，简化了硬件设计，对提高指令执行速度有利，在字长较长的大、中型计算机及超级小型计算机中广泛使用，一些 CISC 的微型计算机中也有使用。例如 IBM 370 系列(32 位字长)，无论什么指令，指令长度是多少位，其操作码字段均

为 8 位。又如前述例子图 5-1 所示的 Intel 8086/8088 CPU 指令格式中，其操作码字段的位数也是固定的 8 位。8 位操作码最多可以表示 256 条指令，实际上 IBM 370 系列和 Intel 8086/8088 都没有那么多条指令，存在一定的信息冗余。

**2. 变长操作码**

顾名思义，变长操作码格式中操作码的位数是不固定的。例如 PDP-11（16 位字长）就采用了这种指令格式，如图 5-7 所示。对于单字长指令，其操作码的位数占 4～16 位不等。这种在基本操作码的基础上进行扩展的变长操作码编码方法也叫扩展操作码法。

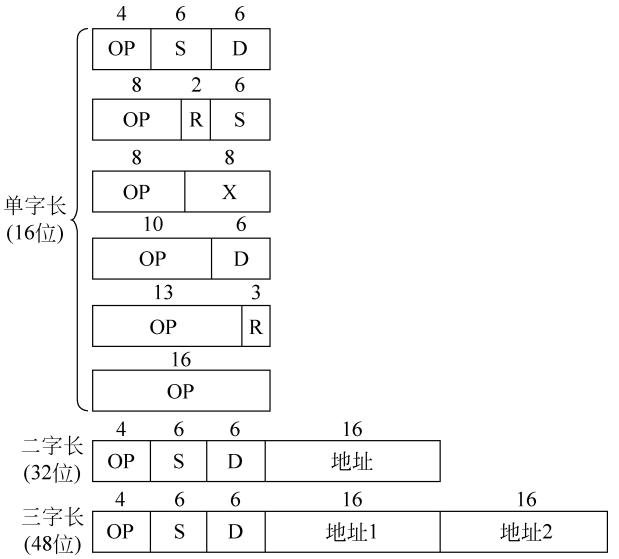



图 5-7　PDP-11 机的指令格式

事实上，当指令长度一定时，地址码和操作码的长度是相互制约的。如果操作数的个数较多，地址码所占位数较多，操作码的位数就要短些；相反，操作数的地址较短，操作码的位数就可以多些。采取这样的变长操作码方法，在不增加指令长度的情况下，通过扩展操作码的位数，可以表示更多的指令。这种方式在一些字长较短的小型机、微型机中广泛采用。

在操作码长度不固定时，一般会考虑将最短的操作码分配给地址码信息最多的指令。实际应用中，会考虑给常用指令分配最短的操作码，而不常用的指令分配长的操作码，以使程序平均指令长度达到最短。因此，指令格式的设计需要综合权衡多种因素。

综上，已经介绍了机器指令设计的基本要素。下面通过一些实例进一步理解指令格式的概念以及操作码、地址码的编码方法。

**【例 5-1】** 指令格式如图 5-8 所示，其中 OP 为操作码，试分析指令格式的特点。

| OP（6 位） | —（2 位） | 源寄存器（4 位） | 目标寄存器（4 位） |
|---|---|---|---|

图 5-8　例 5-1 的指令格式

**解**：该指令格式的特点为

① 单字长、定长操作码、二地址。

② 操作码为 6 位，最多可指定 64 条指令。

③ 源操作数和目标操作数都是通用寄存器，可以指定 16 个寄存器。

【例 5-2】 指令格式如图 5-9 所示，其中 OP 为操作码，试分析指令格式的特点。

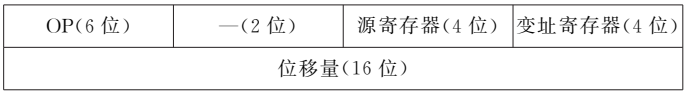



图 5-9 例 5-2 的指令格式

**解**：该指令格式的特点为

① 双字长、定长操作码、二地址。

② 操作码为 6 位，最多可指定 64 条指令。

③ 源操作数是通用寄存器(可以指定 16 个)；目标操作数在存储器中，由变址寄存器和位移量决定存储单元的地址。

【例 5-3】 设指令字长为 16 位，其中 4 位为基本操作码字段(OP)，每个操作数地址编码 4 位，最多可以有 3 个操作数。现对指令的操作码进行等长(等于操作数地址位数)扩展，要求设计的指令系统中：三地址指令 15 条，二地址指令 15 条，一地址指令 15 条，零地址指令 16 条。试写出各种指令的操作码。

**解**：指令的一般格式如图 5-10 所示。


图 5-10 例 5-3 的指令格式

三地址指令，由于 3 个地址各占 4 位，操作码只能 4 位。15 条三地址指令占用 4 位 OP 的 15 个编码，假设 15 个编码为 0000～1110。编码 1111 与后续地址结合进行操作码扩展。

二地址指令，$A_1$ 字段可以扩展为操作码，即操作码最多可以达到 8 位。但操作码的最高 4 位只能是 1111，只能通过 8 位操作码中的低 4 位进行定义。二地址指令有 15 条，占用 15 个编码，其操作码是 OP 字段中的 1111 与 $A_1$ 字段的 15 个编码(0000～1110)结合形成。

同理，将 OP、$A_1$ 字段结合后的 8 位编码中的 11111111，再与后续 $A_2$ 地址字段进行操作码扩展，得到一地址和零地址指令的操作码。其结果如图 5-11 所示。

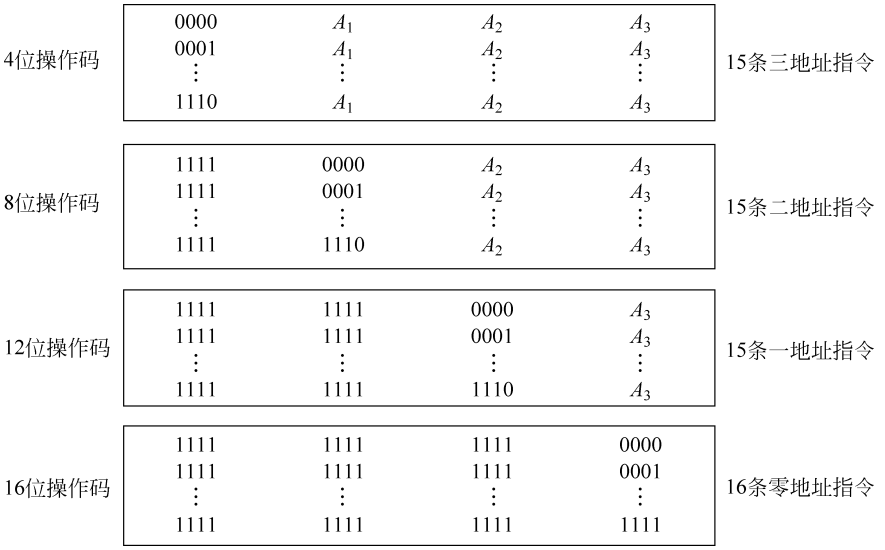



图 5-11 操作码扩展的编码

## 5.3 寻址方式

寻址方式属于指令系统的一部分，它与计算机的硬件结构紧密相关，它的设计和使用对机器的性能和程序的质量有很大影响。

CPU 要执行程序，需要找到指令以及指令所要操作的数据。在指令的代码中，要给出相关的编码信息，以便于 CPU 根据这些信息查找。广义来说，根据指令格式中给出的信息编码，找到要执行的指令或操作的数据的过程称为寻址。寻址方式是指寻找操作数或指令所在位置的方式。完善的寻址方式可为用户编程和使用数据提供方便。

指令一般存放在主存中，而操作的数据则可能存放在 CPU 的寄存器、主存单元或 I/O 端口中。主存的访问有地址指定、相联存储、堆栈等多种方式。一般来说，主存多数情况都采用地址指定方式。因此，狭义地说，寻址方式是形成操作数地址或指令地址的方式。

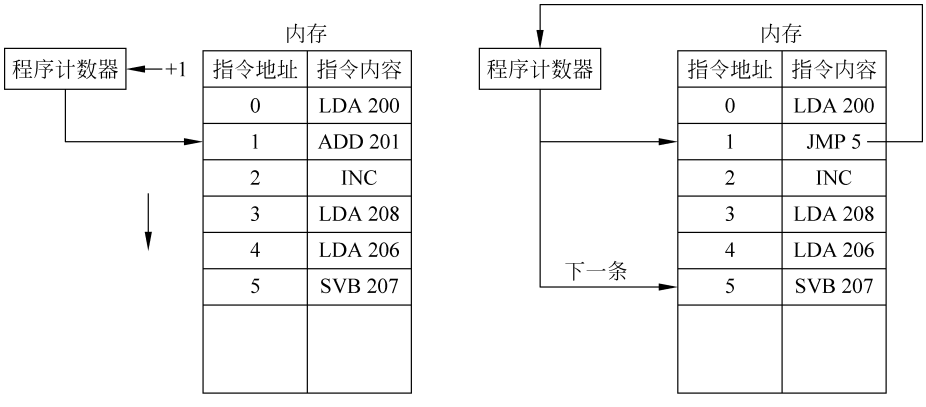
每种计算机的指令系统都有着自己的一套寻址方式。不同的计算机，虽然有着相同的寻址方式，其基本原理也相同，但其名称也并不一定一样。本节仅从一般概念上介绍几种典型又常用的寻址方式。

### 5.3.1 指令的寻址方式

指令的寻址方式指的是形成下一条将要执行指令的地址的方式，分为顺序寻址和跳跃寻址两种。

**1. 顺序寻址方式**

通常，构成程序的指令在内存中是按地址大小顺序存放的。当执行一段程序时，从存储器中按顺序一条指令接一条指令地取出并执行。这种顺序执行指令的过程，称为指令的顺序寻址方式。为此，一般在 CPU 的控制器中设置一专门的程序计数器(PC)，又称指令指针寄存器，其初值为将要执行的一条指令所在内存单元的地址。每取出一条指令，PC 会自增，例如自加 1，形成下一条指令的地址。图 5-12(a)为指令顺序寻址方式的示意图。若 PC 初值为 0，在取出第一条指令 LDA 200 后，PC 的内容自动加 1，指向第二条指令 ADD 201。




(a) 顺序寻址　　(b) 跳跃寻址

图 5-12　指令的寻址方式

**2. 跳跃寻址方式**

当程序不按照存储的先后顺序执行，而是跳到其他地址的指令，下条指令的地址不能由程序计数器直接产生，而是根据本条指令的地址码字段提供的信息形成。这种形成下条指令地址的方式称为跳跃寻址。跳跃寻址后，形成的指令地址装入程序计数器，程序按新地址开始顺序执行。

例如，图 5-12(b)中，在取出地址 1 号单元的 JMP 5 指令后，PC 自加 1 变为 2，正常情况下次顺序取 2 号单元的指令。由于 JMP 5 指令执行时，改变了 PC 的内容，将 5 装入 PC 中，下次要取 5 号单元的指令。即执行完 JPM5 指令后，跳到指令 SVB 207 执行。

跳跃寻址的目标指令地址的形成方式常用的有 3 种：直接(绝对)寻址、相对寻址和间接寻址，它们与下面要介绍的数据寻址方式中对应的寻址方式类似，只是寻址的存储单元中存放的是指令，这里不再赘述。

采用跳跃寻址方式，可以实现程序转移或构成循环程序，从而能缩短程序长度，或将某些程序作为公共程序引用。指令系统中的各种条件转移或无条件转移指令，其寻址方式都是跳跃寻址。

## 5.3.2 数据的寻址方式

数据的寻址方式是指根据指令中地址码字段的信息找到真实操作数据的方式。

现代计算机中，主存的容量越来越大，如果指令格式中的地址码字段直接给出内存单元地址，指令字的长度太长。为了缩短指令字的长度，同时也为了提高编程的灵活性，通常数据的寻址方式会设置多种。

无论是什么寻址方式，指令格式中的地址码字段均为二进制编码，这个二进制编码代表的含义，由寻址方式决定。因此，在允许多种寻址方式的指令格式中，地址码字段中要包含一定的信息或增加扩展操作码来表征寻址方式，这部分信息称为寻址方式特征位。例如，图 5-1 Intel 086/8088 的指令格式中，操作码后面的 MOD 字段，就是用来说明指令的操作数类型或寻址方式类型的。

多种寻址方式中，指令格式中的地址码字段并不代表操作数的真实地址。为了叙述的方便，把它称为形式地址，用 $A$ 表示。而操作数所在单元的实际地址称为有效地址，用 EA 表示，它由寻址方式和形式地址共同确定。即指令格式如图 5-13 所示。

| 操作码 | 寻址特征 | 形式地址 $A$ |
|---|---|---|

图 5-13　寻址指令的一般格式

**1. 立即寻址**

立即寻址是一种特殊的寻址方式，指令的地址码字段中给出的不是操作数的地址，而是操作数本身，即数据包含在指令中。在取出指令时就得到其操作数，因此称为立即寻址。其指令格式如图 5-14 所示。

| OP | 寻址特征 | 立即数 $D$ |
|---|---|---|

图 5-14　立即寻址指令的格式

立即寻址方式的优点是指令执行时间很短，因为取指令后不需要再次访问内存取操作数。但是，因为操作数是指令的一部分，其大小会受指令字长的限制。另外，程序写好之后

立即数不能再被修改，因此编程的灵活性较差。通常只有一些数据传送指令或少数算术运算指令设置这种寻址方式，用来对寄存器或存储单元赋初值。

例如，在 Intel 8086 中，指令

```
MOV AX,2000H
```

表示将立即数 2000H 传送给 AX 寄存器。源操作数采用的是立即寻址方式。当然，立即数只能作为源操作数，不能作为目的操作数。

**2. 寄存器寻址**

寄存器寻址方式的操作数在 CPU 的通用寄存器中，而不在内存中。此时指令格式中的形式地址是通用寄存器组中某个寄存器的编号。其寻址过程如图 5-15 所示。

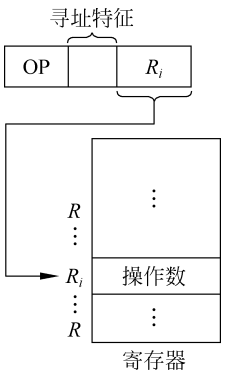



图 5-15　寄存器寻址

例如，指令

```
MOV AX,2000H
```

的目的操作数采用的是寄存器寻址方式。

**说明**：一条指令中，两个操作数可能采用不同的寻址方式，在说指令的寻址方式时，要针对指定的操作数。

寄存器寻址方式有两大优点。

(1) 指令字的长度较短。一般来说，CPU 中的通用寄存器的数量不会太多，相对于内存单元地址来说，用来指定寄存器的编码位数较少，因此指令中地址码部分占位较少，相应的指令字长度较短。

(2) 指令执行速度快。由于寄存器寻址方式的操作数在寄存器中，不需要访问主存获取操作数，因此指令的执行速度较快。

由于 CPU 中通用寄存器的数量不会太多，所以不能用来存储大量的数据。一般来说，大量数据存放在内存中。在程序设计时，对于数据处理类指令中反复使用的数据或中间结果，采用寄存器寻址，以提高整个程序的执行速度。

**3. 主存寻址**

对于操作数在主存的情况，寻址方式有多种。常用的有直接寻址、间接寻址、寄存器间接寻址和偏移寻址等。

(1) 直接寻址。直接寻址是主存寻址中最基本的一种寻址方法，其指令地址码字段中直接给出操作数所在内存单元的地址，即形式地址就是有效地址，EA＝$A$。其寻址过程如图 5-16 所示。

例如，在 Intel 8086 指令系统中，指令

```
MOV AX,[2000H]
```

的第二个操作数采用的是直接寻址，指令的功能是将数据段中偏移地址为 2000H 单元的内容送给 AX 寄存器。

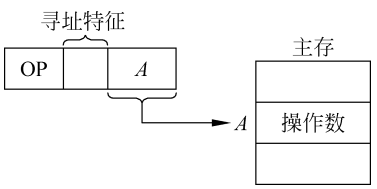



图 5-16　直接寻址

直接寻址方式简单直观，无须地址变换，硬件实现也简单；但可寻址的存储器地址空间受指令中地址码的长度限制，特别是二地址、三地址指令中，这个矛盾更加突出。另外，直接寻址的地址码是指

令的一部分，不能随程序的需要而动态改变，不能满足循环等程序的需要。

该寻址方式也可用于指令的寻址，形式地址直接指向要执行的下一条指令。

例如，在 Intel 8086 的指令系统中，指令

```
JMP 2000H
```

的功能是执行完这条指令后跳转到程序段中地址为 2000H 的指令并继续执行。该指令的形式地址就是要执行的下一条指令的地址。不过，在 Intel 8086 的指令系统中，这种转移指令的寻址方式叫做绝对寻址；相对应，后续的 PC 加形式地址（偏移量）的寻址方式叫相对寻址。

（2）间接寻址。间接寻址是相对直接寻址而言的。在间接寻址方式下，指令地址码字段中的形式地址不是操作数的真实地址，形式地址对应的存储单元的内容才是操作数的有效地址：EA=(*A*)。其寻址过程如图 5-17 所示。

由于间接寻址的操作数的地址，可以由存储单元的内容给出，所以不受指令地址码长度的限制，而与存储单元的字长相关。因此，一般来说间接寻址方式寻址的内存范围比直接寻址大。但是，由于间接寻址需要在取得形式地址后，多一次（一次间址）或多次（多次间址）访问存储器来取操作数地址，指令的执行时间延长。个别计算机设置一次间址，一般不设置多次间址方式。

（3）寄存器间接寻址方式。寄存器间接寻址方式与寄存器寻址方式的区别在于：形式地址编码指定的寄存器中，存放的不是操作数，而是操作数的地址。真正的操作数在该地址指定的内存单元中。即 EA=($R_i$)。其寻址过程如图 5-18 所示。

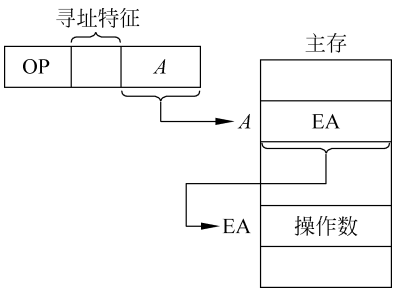



图 5-17　间接寻址

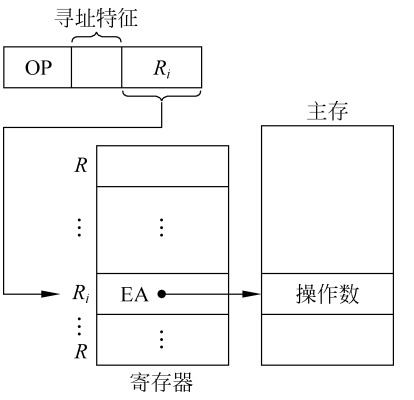



图 5-18　寄存器间接寻址

寄存器间接寻址的操作数不在寄存器中，而是在内存中。其指令的执行速度比寄存器寻址方式慢，但比间接寻址速度快些。由于用于寻址的寄存器的位数一般等于机器字长，大于或等于存储器字长，所以寄存器间接寻址比直接寻址的寻址范围大。此外，由于寄存器的编码位数比内存单元地址位数少，寄存器间接寻址的地址码占位较少，有利于缩短指令字的长度。更重要的是，由于寄存器间接寻址的操作数地址在寄存器中，非常方便实现循环程序中的地址的修改，大大增加了编程的灵活性。

下面是一段 Intel 8086 CPU 的汇编语言程序：

```
        MOV BX, 2000H          ; 取操作数的起始地址,送寄存器 BX 中
        MOV CX, 10             ; CX 做循环计数器,初值为 10
L1:     MOV AL, [BX]           ; 间接寻址取一个操作数
        ADD AL, 20H            ; 操作数加 20H
        MOV [BX], AL           ; 存回原存储单元
        INC BX                 ; 地址加 1
        DEC CX                 ; 循环次数减 1
        JRNZ L1                ; 循环未结束,转 L1 处继续
```

该程序的功能是将连续的 10 个存储单元存放的英文大写字符串转换成小写字符串。其中循环处理连续多个存储单元数据时,采用寄存器间接寻址很方便。

(4) 相对寻址方式。相对寻址是程序计数器 PC 的内容加形式地址 $A$ 形成操作数有效地址的寻址方式,即 EA=(PC)+$A$。程序计数器的内容是当前指令的地址,"相对"寻址就是相对于当前指令的地址偏移了 $A$ 个地址单元,此时的形式地址也叫相对位移量(记作 $D$)。位移量 $D$ 是有符号数,以补码形成表示。其寻址过程如图 5-19 所示。

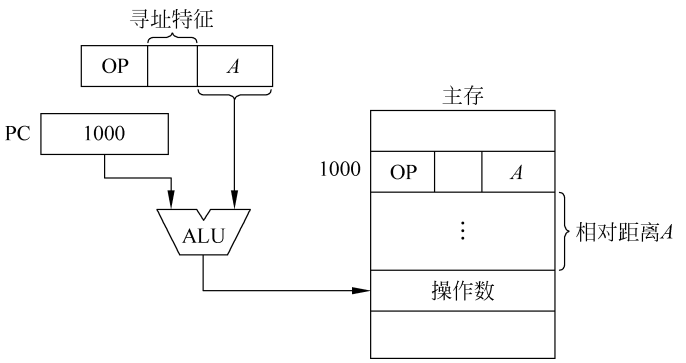



图 5-19　相对寻址

采用相对寻址方式的好处是程序员无须用指令的绝对地址编程,因而所编程序可以放在内存的任何地方。例如,上述程序段示例中,指令 JRNZ L1 采用的就是指令的相对寻址。

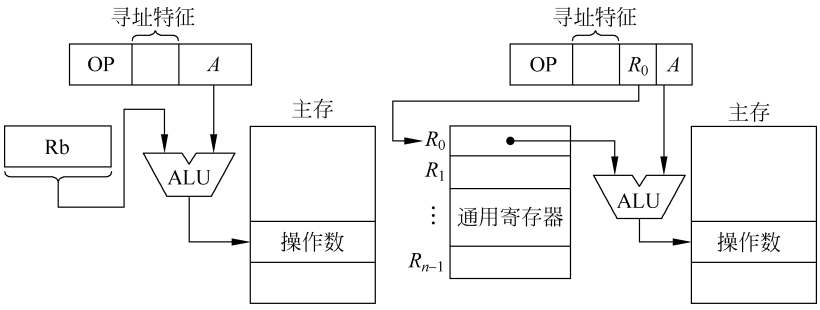
**说明**:相对寻址指令的地址码中的偏移量,是汇编程序将汇编源程序翻译成机器语言代码时计算出来的,程序员无须关注。

(5) 基址寻址方式。基址寻址方式是将 CPU 中基址寄存器的内容,加上形式地址 $A$,形成的操作数的有效地址。即 EA=(Rb)+$A$,此时 $A$ 也称为位移量 $D$。其寻址过程如图 5-20 所示。

图 5-20(a)是 CPU 内部有一个专用的基址寄存器的情况。此时,指令地址码部分可以不显式指出基址寄存器。若通用寄存器中的任一个都可以作为基址寻址的寄存器,则需要地址码部分给出基址寻址寄存器的编码。如图 5-20(b)所示。

基址寻址的优点是可以扩大寻址能力,因为与形式地址相比,基址寄存器的位数较多,从而可以在较大的存储空间中寻址。

(6) 变址寻址方式。变址寻址方式与基址寻址方式计算有效地址的方法很相似,它是把 CPU 中变址寄存器的内容与形式地址 $A$(位移量 $D$)相加,来形成操作数的有效地址,即 EA=($Rx$)+$D$。其寻址过程与基址寻址类似,这里不再图示。




(a) 专用基址寄存器Rb　　(b) 通用寄存器作基址寄存器

图 5-20　基址寻址

基址寻址与变址寻址形成有效地址的算法相同，而且在一些计算机中用相同的硬件实现两种寻址方式，但是两者应用有较大的区别。

变址寻址不以扩大寻址访问为目的，主要用于实现程序块的规律变化。一般来说，变址寻址中的变址寄存器提供修改量(可变的)，指令中提供基准值(固定的)。当变址寄存器的内容按一定的规律变化(自加 1、减 1 或乘比例系数)，而指令本身不改变，可以使有效地址按变址寄存器的内容实现有规律的变化。变址寻址一般面向用户，用来处理字符串、向量和数组等成批数据。

基址寻址中基址寄存器一般提供基准地址(固定的)，指令提供位移量(可变的)。主要是面向系统，用于逻辑地址和物理地址的变换，解决程序在主存中的再定位或扩大寻址空间等问题。

在某些大型机中，基址寄存器只能由特权指令来管理，用户无权操作和修改。在某些小型、微型计算机中，基址寻址和变址寻址实际上合二为一。

变址寻址还可以和其他寻址方式结合使用。例如：

(1) 变址寻址和基址寻址结合，其有效地址为 EA=$(R_b)+(R_x)+A$。

(2) 变址寻址与间接寻址结合。按寻址方式操作的先后顺序又可以分前变址和后变址两种。前者是先变址后间址，有效地址 EA=$((R_x)+A)$；后者是先根据形式地址进行间址然后再变址，有效地址 EA=$(R_x)+(A)$。

例如，Intel 8086 中设置了基址加变址的寻址方式。

**4. 隐含寻址**

隐含寻址是一种特殊的寻址方式，其指令格式中没有明显地给出操作数所在位置的编码，隐含在表示指令功能的操作码中。

例如，一些双操作数指令，其指令格式中只明显地给出一个操作数的地址，另一个操作数约定在累加寄存器中。此时，累加寄存器对单地址指令格式来说是隐含地址。例如 Intel 8086 指令系统中的乘法(MUL)指令和除法(DIV)指令，其被乘数或被除数都是隐含。例如 MUL BL，其被乘数隐含在 AL 寄存器中，乘积隐含存入 AX 寄存器中。

当然也有一些单操作数指令，其指令格式中没有地址码字段，其操作数也是隐含的。例如 Intel 8086 中的对累加器 AL 中的内容进行十进制调整的指令，其汇编语句格式为 DAA。指令编码中只有表示操作功能的操作码，没有表示操作数位置的地址码部分，操作

数隐含在 AL 寄存器中。

### 5.3.3 堆栈及堆栈寻址

堆栈是一种按照“先进后出”或“后进先出”的方式进行存取的存储区。它可以是主存中指定的一块区域，称为软堆栈。在 CPU 寄存器数量较多而堆栈容量较小时，也可以用一组寄存器构成堆栈，称为硬堆栈。

**说明**：软堆栈的含义是指堆栈的大小是用户可以设置的。而硬堆栈是 CPU 中的寄存器堆栈，在 CPU 设计好后，硬堆栈的大小是不可更改的。

一般来说，堆栈主要用来暂存中断断点、子程序的返回地址、状态标志，及中断调用或子程序调用时需要保护的一些寄存器中的数据。

**1. 寄存器堆栈**

寄存器堆栈的结构如图 5-21 所示。位于顶端的寄存器（称为栈顶）是固定的，寄存器组的各寄存器相互连接，它们之间具有对应位自动推移的功能。

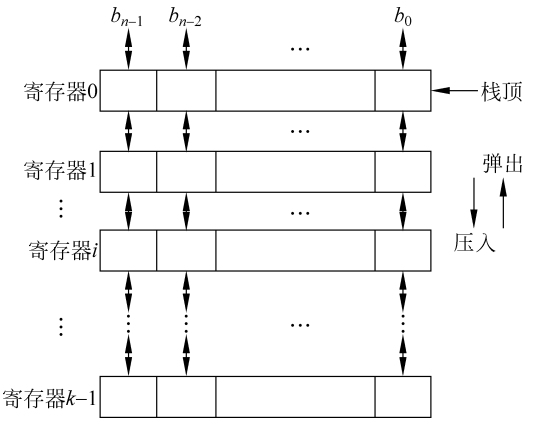



图 5-21　寄存器堆栈结构

在执行入栈操作时，一个压入信号使所有寄存器的内容依次向下推移一个位置，即寄存器 $i$ 的内容被送到寄存器 $i+1$ 中，新的内容进入栈顶（寄存器 0）中。相反，在执行弹出操作时，一个弹出信号把所有寄存器的内容依次向上推移一个位置，栈顶寄存器的内容被弹出。

**2. 存储器堆栈**

存储器堆栈是在主存中划出一段区域作堆栈。其大小可变，栈底固定，栈顶浮动，设置一个专门的寄存器作为栈顶指示器，简称栈指针（Stack Point，SP），指向栈顶的存储单元。操作数只能从栈顶的存储单元进行存或取。堆栈寻址也可以视为一种隐含寻址，其操作数的地址被隐含在 SP 中。当然，它本质上也是寄存器间接寻址。

图 5-22 给出了压栈（PUSH A）的寻址过程，其中 A 为寄存器或存储单元。压栈的具体操作包括如下两步：

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

图 5-23 给出了出栈(POP $A$)的寻址过程,其中 $A$ 为寄存器或存储单元。出栈的具体操作也包括两步:

```
SP+ 1→SP          修改栈指针
(SP)→A            将栈顶单元的内容弹出到 A 中
```

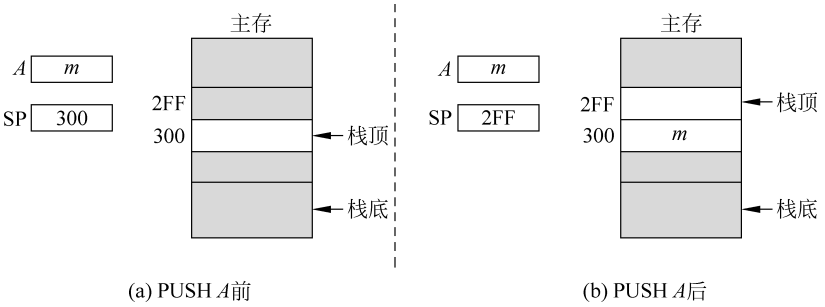



图 5-22 压栈操作(PUSH $A$)过程

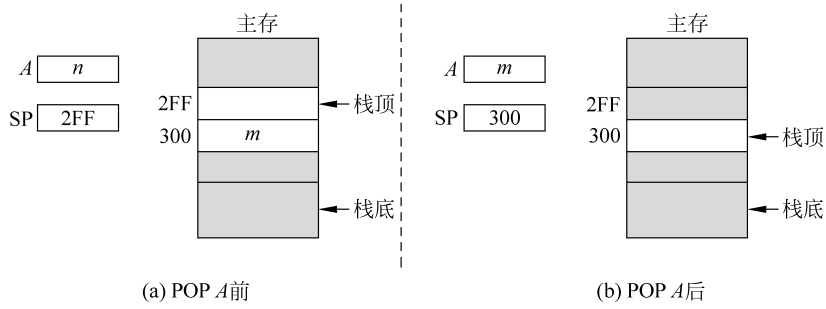



图 5-23 出栈操作(POP $A$)的寻址过程

软堆栈的容量可以很大,而且可以在整个主存中浮动,但是每访问一次堆栈就要访问一次主存,相比于硬(寄存器)堆栈,速度比较慢。在一些大型计算机系统中,希望堆栈的容量大、速度快,往往将前述两种堆栈组合起来使用。在压入、弹出操作时,直接对硬堆栈进行操作。当硬堆栈满后,每向硬堆栈压入一个数据,其栈底寄存器中的数据压入软堆栈中;同样,数据出栈时,不断将软堆栈栈顶的内容移至硬堆栈的栈底寄存器中。相当于将堆栈的总容量进行了有效扩充。这样,既保证速度又扩大了容量。只是在控制上稍复杂些,但又是可以实现的。

### 5.3.4 相联存储方式

相联存储是根据存储单元的内容寻找存储单元的一种寻址方式。它不是面向用户,是面向系统的操作。例如主存地址到高速缓存的地址映射中,主存块复制到高速缓存中时,块标记存放在一个存储表中,该表中单元的寻址过程就是相联存储方式。详细过程见 4.4 节,这里不再赘述。

上述介绍了多种寻址方式。但由于计算机种类繁多,仍有一些计算机的寻址方式并未在此提到,读者使用时自行分析。

不同计算机中采用的寻址方式有所差异,但大都有立即寻址、直接寻址、寄存器寻址、寄

存器间接寻址、变址寻址等基本寻址方式。一台计算机提供多种寻址方式，便于缩短指令长度，扩大寻址空间，提高编程灵活性。

熟悉机器指令的寻址方式，对于汇编语言编程是必须的。当然，设计 CPU 的指令系统，确定指令格式时，更需要了解各种寻址方式。另一方面，如果掌握了指令的寻址方式，会加深对机器内信息流动及整机工作的理解。

下面通过一些例子进一步理解指令格式的设计和寻址方式的概念。

**【例 5-4】** 某机主存容量为 4M×16 位，指令字长等于存储字长。若该机指令系统中一地址指令能完成 70 种操作，操作码位数固定，具有直接、间接、变址、基址、相对、立即 6 种寻址方式。

(1) 画出一地址指令格式，并指出各字段的作用。

(2) 分析其直接寻址的最大寻址范围。

(3) 给出一次间址和多次间址的最大寻址范围。

(4) 给出相对寻址的位移量的范围。

(5) 指出立即寻址的数据范围。

(6) 分析上述 6 种寻址方式的指令哪一种格式执行时间最短？哪一种执行时间最长？哪一种便于用户编制处理数组问题的程序？哪一种便于程序浮动？为什么？

(7) 如何修改指令格式，使指令的直接寻址范围可扩大到 4M？

(8) 为使一条转移指令能转移到主存的任意位置，可采取什么措施？

**解：**

(1) 指令字长与存储字长相等，为 16 位。指令系统中一地址指令共 70 种，因操作码位数固定，所以 OP 字段最少为 7 位。由于具有 6 种寻址方式，寻址方式特征位最少为 3 位；形式地址码位数应尽可能多。

综上分析，一地址指令格式如图 5-24 所示。

| OP(7 位) | $M$(3 位) | $A$(6 位) |
|---|---|---|

图 5-24　例 5-4 的指令格式

(2) 对于直接寻址方式，形式地址就是操作数所在的存储单元地址，$A$ 是 6 位，因此直接寻址的最大范围是 $2^6=64$ 个单元，单元地址范围是 00H～3FH。

(3) 一次间接寻址方式的操作数的地址在形式地址 $A$ 指向的存储单元中，而存储单元是 16 位的，即实际操作数的地址可以有 $2^{16}=64$K 个，单元地址范围是 0000H～FFFFH。

对于多次间址，一般将找到的存储单元的最高位设置为是否继续间接寻址，因此寻址范围为 $2^{15}=32$K 个单元，即单元地址范围为 0000H～7FFFH。

(4) 相对寻址方式时，形式地址 $A$ 表示的偏移量，且是有符号数，6 位偏移量的取值范围为－32～31。

(5) 对于立即寻址，形式地址为 $A$ 为立即数。立即数是有符号数，所以立即数的范围为－32～31。

(6) 对于上述 6 种寻址方式来说：

由于立即寻址是取指令时直接得到的，无须再次访问存储器取操作数，所以执行速度是最快的。

对于间接寻址，每间址一次就要比直接寻址(以及基址、变址寻址)多访问一次存储器，因此执行速度是最慢的。

对于变址寻址，变址寄存器的内容可由用户给定，且程序执行时可以再修改，而形式地址始终不变，因此特别适合数组问题的处理。

相对寻址中操作数的有效地址与当前指令地址之间存在一定的位移量，与直接寻址相比，更有利于程序的浮动。

(7) 要想扩大直接寻址的范围，只能增加形式地址的位数。可以将单字长指令扩展为双字长。指令改为如图 5-25 所示的格式。

| OP(7 位) | $M$(3 位) | $A$ 高位(6 位) |
|---|---|---|
| $A$ 低位(16 位) | | |

图 5-25　例 5-4 扩展为双字长的指令格式

形式地址 $A$ 有 22 位，寻址范围可以达到 $2^{22}=4$M。

(8) 对于实现指令转移功能的寻址方式，即跳跃寻址，具体可以通过相对寻址、基址寻址或变址寻址实现。为使一条转移指令转移到主存任意位置，即寻址范围达到 4M。除了采用(7)中的方法，将形式地址增加到 22 位外，还可以将基址寄存器或变址寄存器配置为 22 位。

**【例 5-5】** 某模型机字长为 16 位，存储器容量为 1MB(按字节编址)。该机指令系统共有 50 种操作指令，其操作码位数固定，采用一地址或二地址格式，可以寄存器寻址、直接寻址和相对寻址(位移量为－128～127)。如果其算术运算和逻辑运算的双操作数全部在寄存器中，结果也在寄存器中；取数/存数指令在通用寄存器和存储器之间进行。试设计算术运算和逻辑运算指令、存数/取数指令和相对转移指令的格式，并简述理由。假设 CPU 内部有 16 个通用寄存器。

**解：**

由于该机共 50 中操作，且操作码位数固定，所以指令中操作码位数设为 6 位。

16 个通用寄存器，每个寄存器的编码占用 4 位；

由于共 3 种寻址方式，且不同指令的寻址方式有特殊要求，寻址方式特征位(M)可以设为 2 位，其中：00 表示寄存器—寄存器，01 表示寄存器—存储器(直接寻址)，10 表示相对寻址。

(1) 对于算术运算和逻辑运算指令，两个操作数都是寄存器寻址，每个寄存器的编码为 4 位，因此可以设置为单字长指令(16 位)格式。具体格式如图 5-26 所示。

| OP(6 位) | 0 0 | 目的 $R$(4 位) | 源 $R$(4 位) |
|---|---|---|---|

图 5-26　例 5-5 的指令格式

(2) 对于存数/取数指令，有一个操作数在存储器中，且采用直接寻址，要在整个存储器(1M 单元)内寻址，形式地址必须提供 20 位的存储单元地址，所以存数/取数指令需要双字长(32 位)。具体格式如图 5-27 所示。

| OP(6 位) | 0 1 | $R$(4 位) | 内存地址的高位 4 位 |
|---|---|---|---|
| 内存地址的低位 16 位 | | | |

图 5-27　例 5-5 扩展为双字长后的指令格式

(3) 转移指令是一地址指令，指令中需提供要转移目标地址的信息。相对转移需采用相对寻址方式提供目标地址，相对位移量为－128～127，形式地址需要 8 位。所以相对转移

指令格式可以是单字长(16 位)。具体格式如图 5-28 所示。

| OP(6 位) | 1 0 | A(8 位) |
|---|---|---|

图 5-28　相对转移指令的格式

## 5.4　典 型 指 令

指令系统是指计算机所能执行的全部指令的集合,它描述了计算机内全部的控制信息和“逻辑判断”能力。不同计算机的指令系统包含指令的种类和数目不尽相同,但一般均包含算术运算类、逻辑运算类、数据传送类、判定和控制类、移位操作类、位(位串)操作类、输入和输出类等功能的指令。这里简单介绍常用类型指令的功能及应用。

### 5.4.1　数据传送类指令

数据传送类指令是计算机指令系统中最基本的指令类型,主要实现计算机内部各部件之间的数据传送。包括 CPU 与内存、CPU 与外设端口、CPU 内部寄存器之间、内存单元之间的数据传送。根据传送的位置或方式不同,又具体分为以下几种。

**1. 一般传送指令**

主要实现 CPU 内部寄存器之间、寄存器与存储单元之间以及内存单元之间的数据传送。一般的助记符采用 MOV,也有些计算机把寄存器到存储单元的传送用 STORE(存数),而将存储单元到寄存器的传送用 LOAD(取数)。

需要说明的是,数据传送是复制性的,即将要传送的数据复制一份送到指定的位置。

**2. 堆栈操作指令**

堆栈指令是一种特殊的数据传送指令,有进栈(PUSH)和出栈(POP)操作两种,具体操作过程在 5.3.3 小节已介绍。一般来说,在程序设计时,进栈和出栈操作会成对出现。

**3. 数据交换指令**

一般的数据传送是单方向的,而数据交换是双向的,将两个不同位置的数据进行互换,为程序设计中的变量值互换提供方便。如果没有此类指令,互换数据需要 3 条指令实现。

**4. 输入输出指令**

输入输出指令主要实现 CPU 与外围设备之间的操作,包括 CPU 与外围设备之间数据的传送,对外围设备的启动与控制,检查测试外围设备的工作状态。

不同计算机实现输入输出指令的方式不同。如果外围设备的端口寄存器的地址与主存单元的地址统一编排,可以不再单独设置输入输出指令,采用访问一般存储单元的指令(取数、存数指令)访问外围设备。如果外围设备的端口寄存器的地址与主存单元的地址各自独立编排,端口寄存器的地址和主存单元的地址有可能是相同的地址,需要通过设置不同的指令(即操作码不同)加以识别。即需要有专用的输入输出指令。一般输入输出指令的助记符为 IN(输入)和 OUT(输出)。

### 5.4.2　运算类指令

运算类指令主要实现计算机的算术逻辑运算功能。具体又分算术运算指令、逻辑运算指令和移位指令 3 类。

**1. 算术运算类指令**

算术运算指令主要实现定点和浮点运算。包括加(ADD)、减(SUB)、乘(MUL)、除(DIV)运算,整数的加 1(INC)、减 1(DEC)及比较(CMP)指令等。为了实现多字节或多字长数据的加减运算,一般还设置带进位的加(ADC)、带借位的减(SBB)指令。有的计算机还设置实现十进制运算的调整指令。

绝大多数运算类指令都会对状态寄存器中的进位、溢出、符号、奇偶、零标志等的状态有一定的影响。程序设计时可以根据这些标志的状态不同,对后续程序进行相应的处理。

**2. 逻辑运算类指令**

一般计算机都有“与”“或”“非”和“异或”等逻辑运算指令。这类指令的特点是按位操作,即两个操作数相对应的位进行逻辑运算。可以实现数据位的检测、对某些位进行置位或清“0”等功能。例如,某数据与 00010000B 相与,如果结果为 0,说明数据的 $D_4$ 位为 0,否则 $D_4$ 位不为 0,即可以进行位测试。同样某数与 11110111B 相与,可以用来将 $D_3$ 位进行清“0”,与 00010000B 相或,可以将 $D_4$ 位进行置“1”。

**3. 移位类指令**

移位类指令可分算术移位、逻辑移位和循环移位 3 种。通过移位指令可以实现位的检测或拼字操作。另外,算术左(右)移可以实现定点有符号数的乘(除)2 运算,逻辑左(右)移可以实现定点无符号数的乘(除)2 运算。由于乘(除)法指令实现乘(除)2 运算的速度比左(右)移指令慢很多,因此一般乘(除)2 的次方的运算常采用移位指令实现。

### 5.4.3 程序控制类指令

程序控制类指令主要实现程序执行顺序的控制。主要包括转移指令、子程序调用和返回指令、中断返回指令等。

转移指令又分无条件转移和有条件转移 2 种。无条件转移又称必转,助记符一般为 JMP。条件转移将受到条件的约束,当条件满足时程序转移,否则仍顺序执行。转移的条件一般是前述运算结果对某些状态标志的影响。例如 JRNC 表示没有进位(即进位标志为 0)时转移,JRNZ 表示不为 0(零标志为 0)时转移。

### 5.4.4 特权指令

特权指令是指具有特殊权限的指令。一般只用于操作系统等系统软件的开发,不直接提供给用户。主要用于系统资源的分配和管理,包括改变系统工作方式,检测用户的访问权限,修改虚拟存储器管理的段表、页表,完成任务的创建与切换等。在多用户、多任务的计算机系统中,特权指令是必不可少的。

### 5.4.5 其他指令

包括开中断 EI、关中断 DI、状态标志位进行置位/复位的指令,暂停 HALT、空操作 NOP 系统控制类的指令,等等。HLAT 指令让机器处于动态停机的状态,主要用来等待某事件发生再使机器进入正常工作状态。NOP 指令执行时不进行任何操作,只是因为执行一条指令而使程序计数器的值增加,它对程序的调试和修改提供很大方便。

CISC 的指令系统一般多达两三百条,不同计算机指令系统的指令功能和表述方式也有

很大差别。这里从教学的角度，给出一个一般计算机都会有的简单操作功能的指令集合。如表 5-1 所示。

**表 5-1　基本操作功能的指令**

| 指令类型 | 指令 | 操作名称 | 说　　明 |
|---|---|---|---|
| 数据传送 | MOV | 传送 | 由源向目标传送字，源和目标是寄存器 |
| | STO | 存数 | 由 CPU 向存储器传送字 |
| | LAD | 取数 | 由存储器向 CPU 传送字 |
| | EXC | 交换 | 源和目标交换内容 |
| | CLA | 清“0” | 传送全 0 字到目标 |
| | SET | 置“1” | 传送全 1 字到目标 |
| | PUSH | 进栈 | 由源向堆栈传送字 |
| | POP | 退栈 | 由堆栈向源传送字 |
| 算术运算 | ADD | 加法 | 计算两个操作数的和 |
| | SUB | 减法 | 计算两个操作数的差 |
| | MUL | 乘法 | 计算两个操作数的积 |
| | DIV | 除法 | 计算两个操作数的商 |
| | ABS | 绝对值 | 以其绝对值代替操作数 |
| | NEG | 变负 | 改变操作数的符号 |
| | INC | 增量 | 操作数加 1 |
| | DEC | 减量 | 操作数减 1 |
| 逻辑运算 | AND | 与 | 按位完成指定的逻辑操作 |
| | OR | 或 | |
| | NOT | 求反 | |
| | EOR | 异或 | |
| | TES | 测试 | 测试指令的条件，根据结果设置标志 |
| | COM | 比较 | 对两个操作数进行逻辑或算数比较，根据结果设置标志 |
| | | 设置控制变量 | 为保护、中断管理、时间控制等设置的指令 |
| | SHI | 移位 | 左(右)移位操作数，一端引入常数 |
| | ROT | 循环移 | 左(右)移位操作数，两端环绕无条件转移，以指定地址装入 PC |
| 控制传递 | JMP | 无条件移 | 无条件转移，以指定地址装入 PC |
| | JMPX | 条件移 | 根据测试条件，将指定地址装入 PC，或什么也不做 |
| | JMPC | 转子 | 将当前程序控制信息放到一个已知位置，转移到指定地址 |
| | RET | 返回 | 由已知位置的内容替代 PC 和其他寄存器的内容 |

## 5.5　RISC

RISC(Reduced Instruction Set Computer，精简指令系统计算机)与其对应的是 CISC(Complex Instruction Set Computer，复杂指令系统计算机)。

### 5.5.1　RISC 的产生

在早期的计算机中，存储器是一个很昂贵的资源，因此希望指令系统能支持生成最短的

程序，希望执行程序时所需访问的数据总数越少越好，从而提高执行效率。如果一条高级语言的语句能被转换成一条机器语言指令，可使编译软件的编写变得非常容易。这种发展趋向导致了复杂指令系统(CISC)设计风格的形成，即计算机性能的提高主要依靠增加指令功能及其复杂性。CISC 指令系统主要存在如下 3 方面的问题。

(1) CISC 中各种指令的使用频度相差很悬殊，大量的统计数字表明，大约有 20%的指令使用频度比较高，占据了 80%的处理机时间。换句话说，有 80%的指令只在 20%的处理机运行时间内才被用到。

(2) CISC 处理机中，大量使用微程序技术以实现复杂的指令系统，给 VLSI 工艺造成很大困难。VLSI 集成度的迅速提高，使得生产单芯片处理机成为可能。在单芯片处理机内，可以采用硬布线控制逻辑，以提高 CPU 的工作速度。

(3) 复杂指令简化了目标程序，缩小了高级语言与机器指令之间的语义差距，但是会增加了硬件的复杂程度，使指令的执行周期大大加大，有可能导致整个程序的执行时间反而增加。

由于 CISC 技术在发展中出现了问题，计算机系统结构设计的先驱者们尝试从另一条途径来支持高级语言，以适应 VLSI 的技术特点。1975 年，IBM 公司 John Cocke 提出了精简指令系统的设想。1979 年，由美国加州大学伯克莱分校 Patter son 教授领导的研究组，首先提出了 RISC 的概念，并先后研制了 RISC-Ⅰ和 RISC-Ⅱ计算机。1981 年，由美国斯坦福大学的 Hennessy 教授领导的研究小组研制出了 MIPS RISC 计算机。该机通过高效的流水和采用编译方法进行流水调度。由此，RISC 技术设计风格得到很大补充和发展。

### 5.5.2 RISC 指令系统的特点

RISC 是计算机体系结构的一种设计思想，是近代计算机体系结构发展史中的一个里程碑。20 世纪 90 年代初，IEEE 的 Michael Slater 对 RISC 的定义是，RISC 处理器的指令系统应能使流水线处理高效率执行，能使编译器生成优化代码。RISC 指令系统的主要特点如下。

(1) 选取使用频率最高的一些简单指令，指令条数少。

(2) 指令长度固定，指令格式种类少，寻址方式种类少。

(3) 只有存数和取数指令可以访问存储器，其余指令的操作都在寄存器之间进行。

(4) 使用简单且格式统一的指令译码。

### 5.5.3 RISC 指令系统实例

本节以 Power PC 为例说明。Power PC 是 Motorola 公司和 Apple 公司联合开发的高性能 32 位 RISC 微处理器，共有 64 条指令。其指令类型与指令格式如图 5-29 所示。

Power PC 的指令有 5 类：

(1) 整数算术运算、逻辑运算、移位、旋转(循环移位)指令；

(2) 浮点算术运算指令；

(3) 取数、存数指令；

(4) 条件寄存器逻辑指令；

(5) 转移指令。

| 6位 | 5位 | 5位 | 16位 | | | |
|---|---|---|---|---|---|---|
| 算术 | 目标寄存器 | 源寄存器 | 源寄存器 | O | 加、减等 | R |
| 与、或等 | 目标寄存器 | 源寄存器 | 有符号立即值 | | | |
| 逻辑 | 源寄存器 | 目标寄存器 | 源寄存器 | 与、或、异或等 | | R |
| 与、或等 | 源寄存器 | 目标寄存器 | 无符号立即值 | | | |
| 旋转 | 源寄存器 | 目标寄存器 | 移位总量 | 屏蔽开始 | 屏蔽结束 | R |
| 旋转或移位 | 源寄存器 | 目标寄存器 | 源寄存器 | 移位类型或屏蔽 | | R |
| 旋转 | 源寄存器 | 目标寄存器 | 移位总量 | 屏蔽 | XO S | R |
| 旋转 | 源寄存器 | 目标寄存器 | 源寄存器 | 屏蔽 | XO | R |
| 移位 | 源寄存器 | 目标寄存器 | 移位总量 | 移位类型或屏蔽 | S | R |

(a) 整数算术、逻辑、移位/旋转指令

| 浮点单/双精度 | 目标寄存器 | 源寄存器 | 源寄存器 | 源寄存器 | Fadd等 | R |
|---|---|---|---|---|---|---|

(b) 浮点算术指令

| Ld/st间接 | 目标寄存器 | 基址寄存器 | 偏移 | | |
|---|---|---|---|---|---|
| Ld/st间接 | 目标寄存器 | 基址寄存器 | 变址寄存器 | 大小、符号、修改 | / |
| Ld/st间接 | 目标寄存器 | 基址寄存器 | 偏移 | | XO |

(c) 取数/存数指令

| CR | 目标位 | 源位 | 源位 | 与、或、异或等 | / |
|---|---|---|---|---|---|

(d) 条件寄存器逻辑指令

| 转移 | 长立即数 | | | A | L |
|---|---|---|---|---|---|
| 条件转移 | 选项 | CR位 | 转移偏移 | A | L |
| 条件转移 | 选项 | CR位 | 通过链或计数寄存器间接进行 | | L |

(e) 转移指令

A=绝对或PC相对　　O=XER中记录溢出　　XO=操作码扩展
L=链接到子程序　　R=CRI中记录条件　　S=移位总量域的部分

图 5-29　Power PC 指令类型与指令格式

指令字是等长的 32 位，并有规整的格式。格式中的高 6 位为操作码，在某些情况下其他部分有操作码的扩展，用于指定操作的细节，如图 5-29 格式中低 16 位的阴影部分所示。大多数指令采用寄存器寻址，寄存器的编码占 5 位，即可以寻址 32 个寄存器。

所有算术运算和移位（包括循环移位）指令的操作数都在寄存器中；逻辑运算中除了少数有一个操作数是立即数外，大多也都是寄存器操作数；取数/存数指令的存储单元地址采用寄存器间接寻址或偏移寻址方式。为指令执行速度的提高奠定基础。

转移指令包括一个链接（L）位，它指示此转移指令后的那条指令的有效地址是否放入链接寄存器。转移指令中的 A 位，用来指示寻址方式是绝对寻址还是 PC 相对寻址。对于条件转移指令，CR 位字段指定条件寄存器中被测试的位，选项字段指向转移发生的条件，例如：无条件转移、计数＝0 转移、计数≠0、条件是真转移、条件是假转移等。

大多数运算指令（算术、逻辑、浮点算术）中一般也有一个 R 位，指定运算结果是否记录

在条件寄存器中，这对于转移预测处理很有用。

指令系统的发展是伴随计算机硬件和软件的发展而发展和演变的，RISC 的设计目标从原来的设法减少指令的数量和种类，变成设法降低执行每条指令所需的时钟周期数。近几年，大凡稍高档点的中央处理器都采用 RISC 技术。RISC 是高性能 CPU 的发展方向。

## 本章小结

指令系统是计算机硬件的语言系统，表征了一台计算机最基本的硬件功能，为程序设计人员呈现了计算机的主要属性。它的格式与功能不仅直接影响到机器的硬件结构，而且也影响到系统软件的开发。

指令系统的设计应满足完备性、有效性、规整性和兼容性的要求。典型的指令系统一般都包括数据传送类、算术逻辑运算类、程序控制类、输入输出类和系统控制类指令。

指令格式是指令字的二进制码表示的结构形式，通常由操作码和地址码两大部分组成。操作码部分表征指令的操作特性和功能，其位数与指令系统的规模有关。地址码部分提供操作数所在位置的有关信息，其长度与操作数所在的位置，以及存储器的单元地址长度、寄存器的个数等多种因素有关。指令的长度称为指令字长，它可以是单字长、半字长和多字长。为了缩短指令字的长度，操作码编码时采用扩展操作码技术。

寻址方式指执行指令时寻找操作数所在位置的方式。根据操作的对象不同，有指令寻址和数据寻址两大类。指令寻址有顺序和跳跃两种。数据的寻址方式，操作数可以在寄存器、内存、I/O 端口和指令中。为了缩短指令的长度，提高编程的灵活性，一般计算机数据的寻址方式都设置很多种。不同计算机中，其数据的寻址方式各不相同，一般都有立即寻址、寄存器寻址、直接寻址、间接寻址、寄存器间接寻址、相对寻址、基址寻址和变址寻址等。堆栈是一种特殊的数据寻址方式，采用“先进后出”的原则。根据堆栈的结构的不同，有寄存器堆栈和存储器堆栈。

## 习 题 5

### 一、基础题

1. 计算机系统中，硬件能够直接识别的编程语句是(　　)。

   A. 机器指令　　B. 汇编语言指令

   C. 高级语言指令　　D. 特权指令

2. 在一地址指令格式中，下面论述正确的是(　　)。

   A. 仅能有一个操作数，它由地址码提供

   B. 一定有两个操作数，另一个是隐含的

   C. 可能有一个操作数，也可能有两个操作数

   D. 如果有两个操作数，另一个操作数是本身

3. 在指令的地址码字段中直接给出操作数本身的寻址方式，称为(　　)。

   A. 隐含地址　　B. 立即寻址　　C. 寄存器寻址　　D. 直接寻址

4. 寄存器寻址方式中，操作数在(　　)中。

A. 寄存器　　B. 堆栈栈顶　　C. 累加器　　D. 主存单元

5. 操作数地址存放在寄存器中的寻址方式叫(　　)。

A. 相对寻址　　B. 变址寻址

C. 寄存器寻址　　D. 寄存器间接寻址

6. 直接、间接、立即 3 种寻址方式指令的执行速度，由快至慢的排序是(　　)。

A. 直接、立即、间接　　B. 直接、间接、立即

C. 立即、直接、间接　　D. 立即、间接、直接

7. 无条件转移指令的功能是将指令中的地址码送入(　　)中。

A. 累加器　　B. 地址寄存器　　C. PC　　D. 存储器

8. 变址寻址方式中，操作数的有效地址等于(　　)。

A. 基值寄存器的内容加上形式地址(位移量)

B. 堆栈指示器的内容加上形式地址(位移量)

C. 变址寄存器的内容加上形式地址(位移量)

D. 程序记数器的内容加上形式地址(位移量)

9. 程序转移类指令的功能是(　　)。

A. 进行主存与 CPU 之间的数据传输

B. 进行 CPU 与 I/O 设备之间的数据传输

C. 进行系统控制

D. 改变程序的执行顺序

10. 下列几项中，不符合 RISC 指令系统的特点是(　　)。

A. 指令长度固定，指令种类少

B. 寻址方式种类尽量多，指令功能尽可能强

C. 增加寄存器的数目，以尽量减少访存次数

D. 选取使用频率最高的一些简单指令以及很有用但不复杂的指令

11. 什么是指令字长、机器字长和存储字长？指令长度和机器字长有什么关系？

12. 零地址指令是否有操作数？一地址指令一定只有一个操作数吗？

13. 某机器字长为 16 位，其指令系统只有单地址指令和双地址指令 2 种。若每个地址字段均为 6 位，双地址指令有 $x$ 条，试求单地址指令最多可以有多少条？

14. 一种二地址 RS 型指令格式如图 5-30 所示。

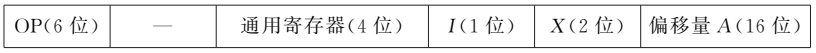



图 5-30　第 14 题的指令格式

其中 $I$ 为间接特征，$X$ 为寻址模式，$A$ 为形式地址，通过 $I$、$X$、$A$ 组合可构成表 5-2 所示的寻址方式。请写出表 5-2 中 6 种寻址方式的名称。

**表 5-2　6 种寻址方式**

| 寻址方式 | $I$ | $X$ | 有效地址 EA 的算法 | 说　明 |
|---|---|---|---|---|
| ① | 0 | 00 | EA=$A$ | — |
| ② | 0 | 01 | EA=(PC) +$A$ | PC 为程序计数器 |

续表

| 寻址方式 | $I$ | $X$ | 有效地址 EA 的算法 | 说　明 |
|---|---|---|---|---|
| ③ | 0 | 10 | EA=($R_2$) +$A$ | $R_2$ 为变址寄存器 |
| ④ | 1 | 11 | EA=($R_2$) | — |
| ⑤ | 1 | 00 | EA=($A$) | — |
| ⑥ | 0 | 11 | EA=($R_1$)+$A$ | $R_1$ 为基址寄存器 |

15. 某计算机字长 32 位，主存容量 4MB，其指令长度与字长相等。指令系统共有 80 种操作的指令，寻址方式有立即寻址、直接寻址、寄存器间接寻址、变址寻址 4 种。试给出单地址指令的格式(假设采用定长操作码)，并说明当采用不同的寻址方式时，形式地址的含义及寻址的范围。

**二、提高题**

1. 指令系统中采用不同寻址方式的目的主要是(　　)。

A. 可直接访问外存

B. 实现存储程序和程序控制

C. 扩展操作码，降低指令译码难度

D. 缩短指令长度，扩大寻址空间，提高编程灵活性

2. 单地址指令中为了完成两个操作数运算，地址码指明一个操作数，另一个操作数需采用(　　)。

A. 立即寻址　　B. 直接寻址　　C. 间接寻址　　D. 隐含寻址

3. 操作数在内存中，为了缩短指令地址码的位数，同时指令的执行时间又相对短，则有效的寻址方式是(　　)。

A. 立即寻址　　B. 寄存器寻址

C. 直接寻址　　D. 寄存器间接寻址

4. 指令的寻址方式有顺序和跳跃两种，采用跳跃寻址，可以实现(　　)。

A. 堆栈寻址　　B. 程序的有条件转移

C. 程序的无条件转移　　D. 程序的无条件转移或有条件转移

5. 采用寄存器间接寻址方式的指令中，形式地址给出的是(　　)。

A. 立即数　　B. 寄存器的编码　　C. 主存单元地址　　D. 有效地址

6. 扩展操作码是(　　)。

A. 指令格式中不同字段设置的操作码

B. 增加操作码字段的位数

C. 操作码字段以外的辅助操作码字段的代码

D. 一种指令优化技术，即让操作码的长度随地址数的减少而增加，不同地址数的指令可以具有不同的操作码长度

7. 下列描述汇编语言特性有错误的是(　　)。

A. 对程序员的训练来说，需要硬件知识

B. 汇编语言对机器的依赖性高

C. 汇编语言源程序通常比高级语言源程序短小

D. 汇编语言编写的程序执行速度一般比高级语言快

8. 某机为定长指令字结构，指令长度 32 位，每个操作数的地址码长 9 位，指令分为零地址、单地址和二地址 3 种格式。若二地址指令已有 $K$ 种，零地址指令已有 $L$ 种。

(1) 采用扩展操作码时，单地址指令最多可能有多少种？

(2) 采用扩展操作码时，上述 3 类指令各自允许的最大指令条数是多少？

9. 若某机存储器按字节编址，其相对寻址的转移指令长度为 2B，首字节是操作码，第二字节是相对位移量，用补码表示。现假设当前转移指令首字节所在地址为 2010H，且 CPU 每取一字节 PC 自加 1。试问 JMP * +8 指令和 JMP * −9 指令(* 为相对寻址特征)的第二字节的内容分别是多少？转移的目标地址各是什么？

10. 设机器字长、指令字长、存储字长相同。举例说明哪几种寻址方式除取指令外不访问存储器？哪几种寻址方式除取指令外要访问一次存储器？哪种寻址方式包括取指令在内要访问 4 次存储器？

11. 设某机字长为 32 位，存储器字长 16 位，CPU 有 16 个 32 位通用寄存器，一个 32 位变址寄存器。若该机指令系统共有 120 种操作，有直接寻址，间接寻址、立即寻址、变址寻址等 6 种寻址方式，采用单字长指令格式。试设计一个二地址指令格式，要求其中一个操作数在寄存器中。并分析回答：

(1) 采用直接寻址的最大寻址空间为多少？

(2) 若采用间接寻址，可寻址的最大存储空间为多少？

(3) 如果采用变址寻址，可寻址的最大存储空间为多少？

(4) 若立即数为带符号的补码整数，写出立即寻址中立即数范围。

12. 某机的指令格式如图 5-31 所示。

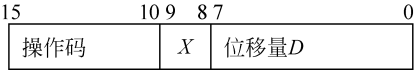



图 5-31　第 12 题的指令格式

其中：$X$ 为寻址特征位，$X=00$ 表示直接寻址；$X=01$ 表示用变址寄存器 $R_{X1}$ 寻址；$X=10$ 表示用变址寄存器 $R_{X2}$ 寻址；$X=11$ 表示相对寻址。

设(PC)=1234H，$(R_{X1})$=0037H，$(R_{X2})$=1122H(H 代表十六进制数)，请确定下列指令编码中的操作数的有效地址：

(1) 4420H；

(2) 2244H；

(3) 1322H；

(4) 3521H。