第 5 章　指令与指令系统

本章学习目标

- 详细了解计算机指令系统。
- 熟悉计算机指令设计的基本方法。
- 掌握基本计算机指令的编写。

本章将详细介绍计算机指令系统的发展过程，指令的分类、各种指令格式、寻址方式以及功能、指令集的优化设计、指令周期和计算机 ARM 指令集结构，同时简单地介绍了 RISC 指令级的并行处理。

5.1　指令系统的发展

计算机系统主要由硬件(Hardware)和软件(Software)两部分组成。所谓硬件就是由中央处理器(CPU)、存储器以及外部设备等组成的实际装置。软件则是为便于用户使用计算机而编写的各种程序，最终转换成一系列机器指令后在计算机上执行。

计算机的性能与它所设置的指令系统有很大的关系，而指令系统的设置又与机器的硬件结构密切相关。通常性能较好的计算机都设置有功能齐全、通用性强、指令丰富的指令系统，但这需要复杂的硬件结构来支持。

在 20 世纪 50 年代和 60 年代早期，由于计算机采用分立元件(电子管或晶体管)，其体积庞大，价格昂贵，因此，大多数计算机的硬件结构比较简单。其所支持的指令系统一般只有定点加减运算、逻辑运算、数据传送和移动等十几至几十条最基本的指令，而且寻址方式简单。到 20 世纪 60 年代中后期，随着集成电路的出现，计算机的价格不断下降，硬件功能不断增强，指令系统也越来越丰富。除了具有以上最基本的指令以外，还设置了乘除法运算指令、浮点运算指令、十进制运算指令以及字符串处理指令等，指令数多达一二百条，寻址方式也趋于多样化。

随着集成电路的发展和计算机应用领域的不断扩大，计算机的软件价格相对不断提高。为了继承已有的软件，减少软件的开发费用，人们迫切希望各机器上的软件能够兼容，以便在旧机器上编制的软件也能在新的、性能更好的机器上正确运行，因此，在 20 世纪 60 年代出现了系列(Series)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，如 IBM 370 系列，VAX-11 系列，IBM PC(XT/AT/286/386/486/Pentium)微机系列等。一个系列往往有多种型号，由于推出的时间不同，所采用的器件也不同，因此在结构和性能上可以有很大差异。通常新推出的机种在性能和价格方面要比早推出的机种优越。系列机能解决软件兼容问题的必要条件是该系列的各机种拥有共同的指令集，而且新推出的机种的指令系统一定包含旧机种的所有指令，因此在旧机种上运行的各种软件可以不加任何修改地在新机种上运行。

计算机发展至今，其硬件结构随着超大规模集成电路(VLSI)技术的飞速发展而越来越复杂化，所支持的指令系统也趋于多用途、强功能化。指令系统的改进是围绕着缩小指令与高级语言的语义差异以及有利于操作系统的优化而进行的。例如，高级语言中的实数计算是通过浮点运算实现的，因此对用于科学计算的计算机来讲，如能设置浮点运算指令就能显著提高运算速度；另外在高级语言程序中经常用到 IF 语句、DO 语句等，为此设置功能较强的条件转移指令是有好处的；为了便于程序嵌套，设置了调用指令(CALL)和返回指令(RET)等。上述这些措施都是为了便于高级语言程序编译以及提高机器的运行速度而采取的，这对简化汇编语言程序设计也是很有利的。为了便于操作系统的实现和优化，还设置有控制系统状态的特权指令、管理多道程序和多处理机系统的专用指令等。然而，指令结构太复杂也会带来一些不利的因素，如设计周期长，正确性难以保证且不易维护等；此外，实验证明，在如此庞大的指令系统中，只有诸如算术逻辑运算、数据传送、转移和子程序调用等几十条最基本的指令才是经常使用的，需要大量硬件支持的大多数较复杂的指令却利用率很低，造成硬件资源的极大浪费。为了解决这个问题，在 20 世纪 70 年代末人们提出了便于 VLSI 实现的精简指令系统计算机，简称 RISC(见 5.5 节)。

5.2 指令系统

指令系统是计算机硬件的语言系统，也叫机器语言，它是软件和硬件的主要界面，从系统结构的角度看，它是系统程序员看到的计算机的主要属性。因此指令系统表征了计算机的基本功能，决定了机器所要求的能力，也决定了指令的格式和机器的结构。对不同的计算机在设计指令系统时，应对指令格式、类型及操作功能给予应有的重视。

5.2.1 指令系统的介绍

指令系统是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的指令种类和数目也不同。一般均包含算术运算型、逻辑运算型、数据传送型、判定和控制型、输入和输出型等指令。指令系统是表征一台计算机性能的重要因素，它的格式与功能不仅直接影响到机器的硬件结构，而且也直接影响到系统软件，影响到机器的适用范围。

不同系列的计算机有不同的指令系统，指令系统的设计是依据计算机 CPU 的硬件特点而研究出来的，和 CPU 性能有着密不可分的关系。为了实现更多更好的性能，人们采用各种先进技术设计 CPU，但最终都是通过设计一套性能极高的指令系统而展现计算机的强大功能的；相反，如果指令系统太庞杂，也会带来很多不利因素，因为在庞杂的指令系统中存在一些不经常使用的指令，这些指令需要很多硬件支持，这样就会降低性能价格比。为了解决这些问题，人们提出精简指令系统。

5.2.2 指令的分类

常见指令按功能可划分为

(1) 数据处理指令：包括算术运算指令、逻辑运算指令、移位指令、比较指令等。

(2) 数据传送指令：包括寄存器之间、寄存器与主存储器之间的传送指令等。

(3) 程序控制指令：包括条件转移指令、无条件转移指令、转子程序指令等。

(4) 输入/输出指令：包括各种外围设备的读、写指令等。有的计算机将输入/输出指令包含在数据传送指令类中。

(5) 状态管理指令：包括诸如实现存储保护、中断处理等功能的管理指令。

随着计算机系统结构的发展，有些计算机还不断引入新指令，如“测并置”指令是为在多机系统和多道程序中防止重入公用子程序而设置的。指令先测试标志位以判断该子程序是否正在使用。如未被使用，则转入子程序并置该标志位，以防其他进程重入。后来又出现功能更强的信号(PV 操作)指令。有的计算机还设置“执行”指令。“执行”指令执行由地址域所确定的存储单元中的指令。其目的是避免用程序直接修改程序中的指令。这对程序的检查和流水线等技术的应用均有好处。有的计算机采用堆栈实现程序的调用指令和返回指令。调用时将返回地址和各种状态、参数压入堆栈顶部，这样就能较好地实现子程序的嵌套和递归调用，并可使子程序具有可重入性。另外，一些计算机使不少复杂的操作固定化，形成诸如多项式求值、队列插项、队列撤项和各种翻译、编辑等指令。

那么计算机的指令系统究竟是怎样构成的呢？下面的章节将详细介绍这个问题。

5.3 指令格式

指令语言是面向机器的语言，它的程序目标代码短，执行速度快，计算机通过解释每条指令的含义，从而执行各种预定操作，设计一种合理有效的指令格式是设计指令系统的关键。

指令一般由两部分组成：操作码和操作数。操作码指示计算机的有关部件执行什么操作；操作数指明参加本指令运算的数据。一般指令格式如图 5.1 所示。

操作码(OP)	操作数

图 5.1 一般指令格式

在有些指令中可以没有操作数，例如 NOP 指令；操作数也可以直接出现在指令中，如 MOV AL 10H；有些指令中的操作数则存放在某些地址码中，地址码不同，操作数的存储位置也就不同。总之，一条指令一般应包含以下信息：

(1) 说明操作的性质，即计算机要具体实施什么样的动作；

(2) 说明操作数的位置，给出相应的地址码，即如何找到操作数本身；

(3) 说明下一条指令的地址。

一般说来，操作码字段一经确定，就很少再变化了，但地址码的形式是多样的，指令的地址码反映了操作数在计算机中的存储位置，根据地址的个数，指令可分为零地址指令、单地址指令、双地址指令、三地址指令和多地址指令等多种形式，下面对地址码字段做具体介绍。

5.3.1 零地址指令

在指令中只有操作码，没有操作数地址，其格式为

操作码(OP)

这种指令用于不需要操作数的场合，例如空操作 NOP、开中断指令 STI、十进制修正指令 DAA 和暂停指令 HLT 等。另外指令中没有操作数地址也并不意味着没有地址，而是产

生的地址比较特殊，采用默认方式，如在堆栈的存储单元中存放数据经常采用这种方式，参加运算的操作数放于堆栈中，运算结果也存放在堆栈中。

5.3.2 单指令地址

指令中只有一个操作数地址，其格式为

操作码(OP)	操作数地址A1

操作数地址 A1 有两种含义，既是操作数地址，又是指令执行后的结果存放地址，如 Intel 8086 系列计算机中的加 1 指令 INC、减 1 指令 DEC 等。还有一种方式为第一个操作数的地址在指令中给出，另一个操作数在默认的累加器 A 中，结果仍放回累加器 A 中。

单地址指令长度较短，可以节省内存空间，在微型计算机中经常采用这种指令。

5.3.3 双地址指令

指令中明确说明两个操作数地址，其格式为

操作码(OP)	操作数地址A1	操作数地址A2

A1 为源操作数地址，A2 为目的操作数地址，执行指令时将源操作数地址 A1 的内容和目的操作数地址 A2 的内容按规定操作后，将结果存入目的操作数地址 A2 中，执行结果将替代目的操作数地址中原有的内容。双地址指令长度较短，执行速度快，硬件实现简单，因此是较常用的一种方式。

5.3.4 三地址指令

指令中明确指出三个操作数地址和存放结果的地址，其格式为

操作码(OP)	操作数地址A1	操作数地址A2	操作数地址A3

A1 为第一操作数地址，A2 为第二操作数地址，A3 为结果地址。按此方式执行时，将地址 A1 中的操作数和地址 A2 中的操作数执行规定的操作后，结果送入地址 A3 中。三地址指令的指令长度较长，一般适用于字长较长的大中型计算机。

5.3.5 多地址指令

在多地址指令中包含多个地址码，其格式为

操作码(OP)	操作数地址A1	…	操作数地址An

这种方式可能指出下一条指令的地址单元，比较直观，但是指令长度较长，操作速度相对较慢，因此这种指令使用率不高。

以上介绍的各种指令方式在使用中需要综合考虑，如果为了减少成本，简化硬件设计，或者减少访问次数和指令长度，则单地址指令为首选指令；如果为了节省内存空间，就得尽量缩短指令长度，这时选用三地址指令较好，也方便用户使用，但是指令长度较长。在相同条件下，用三地址指令编写的程序较短，但指令长度较长，用双地址、单地址和零地址指令编

程，程序的长度一个比一个长，但指令的长度一个比一个短，为此对这几种指令方式用表5.1作比较。

表5.1　各种指令方式的比较

地址个数	程序长度	占用内存空间	执行速度	应用场合
三地址	短	最多	一般	矩阵、向量运算等
双地址	一般	很多	很慢	很少用
单地址	较长	较多	较快	硬件电路简单、连续运算等
零地址	最长	最小	最慢	递归、嵌套等问题

5.4　寻址方式

在执行指令的过程中，需要确定操作数的位置，并指出操作结果送到什么地方，这就涉及操作数的寻址方式。指令中提供操作数或操作数地址的方式称为寻址方式。寻址方式与计算机硬件结构密切相关，不同计算机有不同的寻址方式，但基本原理大致相同。寻址方式设计的好坏直接关系到编程人员的编程效率，一套好的寻址方式，可以使指令中的地址字段尽量短，减少占用的内存空间，提高工作效率。计算机中常用的寻址方式有很多种，下面介绍使用率较高的几种基本寻址方式：立即寻址、寄存器寻址、直接寻址、变址寻址和相对寻址。

5.4.1　立即数寻址

指令的操作数直接出现在指令中，这种寻址方式称为立即数寻址，其格式为

操作码(OP)	立即数Im

指令的功能是对立即数Im进行操作码OP规定的操作，操作数在指令中直接取得，这就意味着只要取出指令，也就同时取得了操作数，这种寻址方式不必访问存储单元，免去了执行总线周期的时间，缩短了可执行指令的时间。因此这类寻址方式的显著特点就是速度快。但是立即数只能作为源操作数，不能作为目的操作数。

例5.1　MOV AX,1000H

如图5.2所示，操作码OP和操作数1000H分别放在不同的存储单元中，执行指令后，1000H这个操作数就被送到AX寄存器中，小地址内容00H放入AL中，大地址内容10H放入高字节AH中。

5.4.2　寄存器寻址

指令中出现在地址码位置的是CPU内部寄存器，操作数本身存放在指令指定的寄存器中，这种寻址方式称为寄存器寻址，其格式如图5.3所示。

例5.2　MOV AX,BX，设(AX)＝2326H，(BX)＝8086H，则执行指令后AX和BX的内容是多少？

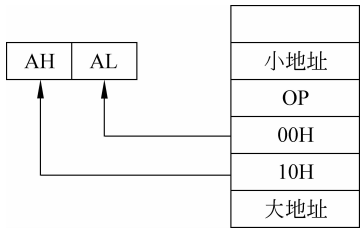


图 5.2　立即数寻址操作过程

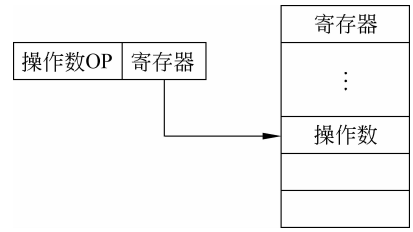
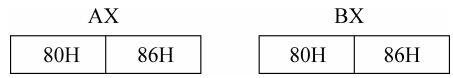


图 5.3　寄存器寻址操作过程

指令执行前为

指令执行后为



由于计算机中寄存器的数目较少，比存储器要少得多，而且寄存器在 CPU 内部，不用执行总线周期，所以这种寻址方式可以缩短指令长度，提高指令执行速度，是一种广泛采用的寻址方式。

5.4.3　直接寻址

有效地址(Effective Address，EA)直接出现在命令中，称为直接寻址。有效地址即为存储器的某个存储单元地址，操作数存在指令中出现的地址码中，所以直接寻址是访问存储器时普遍采用的最简单的寻址方式，其格式如图 5.4 所示。

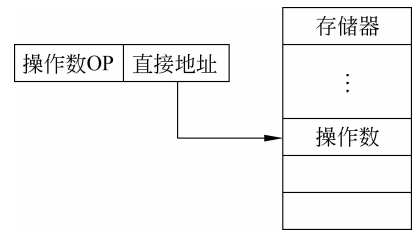


图 5.4　直接寻址操作流程

例 5.3　在 80X86 系列某计算机中，设 DS＝2000H，(21000H)＝12H，(21001H)＝34H，执行指令 MOV AX，[1000H]后，AX 的内容是什么？

解：有效地址EA＝1000H，则物理地址为

＝DS×16＋1000H

＝2000H×16＋1000H＝21000H

执行指令后 21000H 和 21001H 两个存储单元的内容被取出来送到寄存器 AX 中，大地址内容送高位字节 AH 中，小地址内容送低位字节 AL 中，所以指令完成后 AX 的内容为 3412H。

5.4.4　间接寻址

间接寻址方式中，在指令的地址码中给出的不是操作数的真正地址，而是一个形式地址。形式地址可能是寄存器，也有可能是存储器地址，因此间接寻址可分成两类：寄存器间接寻址和存储器间接寻址。下面就分别介绍这两种间接寻址方式。

1. 寄存器间接寻址

寄存器间接寻址是指在寄存器中给出的不是一个操作数，而是操作数的地址，是最基本、最常用的寻址方式之一，其格式如图 5.5 所示。

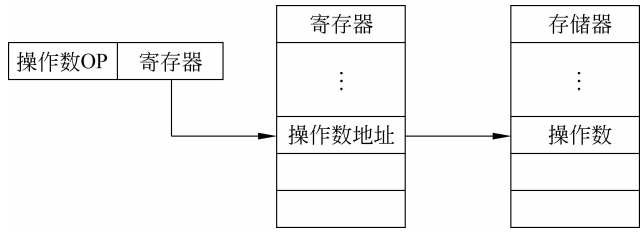


图 5.5　寄存器间接寻址操作过程

例 5.4　在 80X86 系列某计算机中，设 DS＝3000H，(BX)＝4000H，(34000H)＝56H，(34001H)＝78H，执行指令 MOV AX,[BX]后，AX 的内容是什么？

解：有效地址EA＝4000H，则物理地址为

＝DS×16＋4000H＝3000H×16＋4000H＝34000H

执行指令后，34000H 和 34001H 两个存储单元的内容被取出来送到寄存器 AX 中，大地址内容送高位字节 AH 中，小地址内容送低位字节 AL 中，所以指令完成后 AX 的内容为 7856H。

由此可见，寄存器间接寻址的操作数最终还是从存储器中取出来的。这种寻址方式可以应用在处理表格中，执行完一条指令后，如果想取出表格中的下一个内容，只要修改寄存器的内容即可。

2. 存储器间接寻址

在存储器间接寻址方式中，地址码字段中出现的不是操作数的有效地址，而是操作数的地址，其格式如图 5.6 所示。

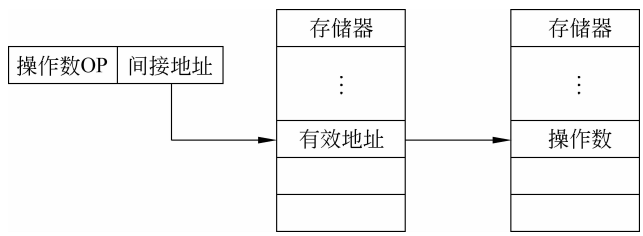


图 5.6　存储器间接寻址操作过程

间接寻址是相对直接寻址而言的，因此它可以是多级间接寻址，方法不再赘述，不论是一级还是多级寻址方式，最终操作数一定在存储器中取得。

5.4.5　变址寻址

指令中出现在地址码中的地址和变址寄存器中的内容相加，所得的结果作为操作数的地址，这种寻找方式叫做变址寻址，其格式如图 5.7 所示。

由图 5.7 可知，如果把地址码 A 的值作为基址，把变址寄存器 X 提供的地址作为偏移量，偏移量就是一个带有方向性(即有正有负)的值，两者相加构成的地址即为有效地址。

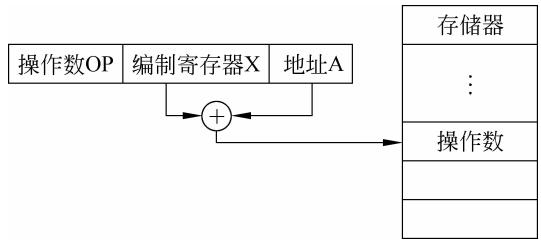


图 5.7　变址寻址操作过程

变址寻址方式是目前普遍应用的一种寻址方式，尤其是在需要频繁修改地址时，无须修改指令，只要修改变址寄存器中的内容就可以了。如在数组的应用中，可以把数组的首地址作为基址，让变址寄存器的内容改变，就可方便地访问数组中的各个元素。这种方式非常灵活，是计算机中广泛应用的一种寻址方式。

例 5.5　假定(BX)＝637DH，(SI)＝2A9BH，位移量 D＝3237H，试确定在变址寻址方式下的有效地址是什么？

解：变址寻址的有效地址＝(BX)＋(SI)＝637DH＋2A9BH＝8E18H。

5.4.6　相对寻址

指令中的地址码中出现的形式地址与程序计数器(PC)的内容之和形成操作数的有效地址，称为相对寻址，其格式如图 5.8 所示。

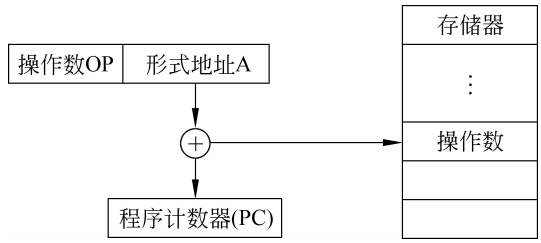


图 5.8　相对寻址操作过程

相对寻址方式中，操作数的地址是浮动的，随着程序计数器(PC)的内容变化，有效地址也在不断地变化，这样程序员在编写程序时就不用考虑指令的绝对地址，即编写的程序可以存放在存储器中的任意位置。

例 5.6　假设(DS)＝2900H，(ES)＝2100H，(SS)＝1500H，(SI)＝00A0H，(BX)＝0100H，(BP)＝0010H，数据段中变量名 VAL 的偏移地址值为 0050H，试指出源操作数字段 MOV AX,[BX＋10] 的寻址方式是什么？其物理地址值是多少？

解：在指令 MOV AX,[BX＋10] 中，源操作数字段的寻址方式是相对寻址，其物理地址值为

$$(DS)\times 10H+(BX)+0AH = 29000H+100H+0AH = 2910AH$$

5.5 复杂指令系统和精简指令系统

5.5.1 复杂指令系统

随着计算机系统结构的逐步完善和性能的不断提高，指令系统也在相应地不断变化。早期的计算机指令系统较小，一般只包含一些功能很简单的指令，功能比较弱，这是由于当时的硬件水平比较低。随着集成电路技术的飞速发展，计算机的硬件性能不断提高，成本逐渐下降，软件成本却不断上升。为了适应不同应用领域的需求以及尽量使软件兼容，人们不得不靠增加指令来提高系统性能，这就使得指令系统越来越庞杂（指令系统往往达到几百条指令），机器结构也越来越复杂，因而称这类计算机为复杂指令系统计算机（Complex Instruction Set Computer，CISC）。例如，VAX-11/780 计算机的指令条数达到 303 条，寻址方式也有 18 种之多，这是典型的 CISC。另外，Intel 80x86 系列计算机也属于 CISC。

CISC 的指令系统具有下列特点：

（1）指令集丰富，通常在 200 条以上，指令格式和寻址方式一般多于 4 种。

（2）采用微程序控制技术，各种指令均可访问存储器，指令执行需要较多的时钟周期。

（3）程序一般按照串行顺序执行，控制简单，执行速度慢。

（4）软件系统开发时间较短。

（5）指令字长变化，指令的执行时间相差较大。

（6）高级语言的编译指令不是唯一的，使得高级语言的编译很难优化。

CISC 的指令集向下兼容性好，新的系统只需在原来的基础上增加指令，对编译器的要求也不高，开发时间短。但是缺点也很多，指令系统规模过大，指令格式长短相差悬殊，导致计算机的硬件结构控制复杂，使得 CPU 的效率跟不上集成电路的发展要求。

为了解决这类问题，1975 年，IBM 的设计师约翰·科克（John Cocke）提出了一种全新的思想，即精简指令系统计算机（Reduced Instruction Set Computer，RISC）的想法。

5.5.2 精简指令系统

RISC 设计师们曾经做过这样的测试，发现占指令总数 20％的常用简单指令，使用频率却高达 80％，而指令总数的另外 80％的复杂指令，使用频率却只有 20％，这样结构的指令系统，不但增加了调试和维护的难度，也使得计算机的研制周期增长，并且降低了系统的执行速度，所以精简指令势在必行。

RISC 技术设计的成功，使得指令系统又有了一个新的突破性进展，它不但可以简化指令，而且可以简化计算机的指令译码器和控制器，加快计算机的执行速度。

RISC 的特点为：

（1）简化了指令系统，指令数一般在 100 条以下，寻址方式和指令格式大多在 4 种以下。

（2）指令的操作大多在寄存器间完成，仅有少数几条指令可以访问存储器。

（3）控制器大多采用硬布线控制逻辑，少用或不用微程序控制。

（4）采用流水线控制技术，提高指令执行速度。

(5) 用简单有效的方式支持操作系统和高级语言,开发软件的周期比较长。

(6) 采用优化的编译程序,能够支持高级语言。

RISC 技术和 CISC 技术各有优缺点,也各自有自己的应用领域,但是随着集成技术的不断发展,RISC 将会逐渐占据优势地位。

5.6 指令集结构的功能设计

大多数指令集结构所支持的操作可以被归纳为如表 5.2 所示的一些类型。指令集结构一般都会对三种类型的操作提供相应的指令。各种指令集结构对系统功能的支持程度会随着结构的不同而有较大的差异,但是所有的指令集结构都必须对基本的系统功能提供一些指令支持。另外,大多数具有浮点计算功能的指令集结构均提供了支持浮点操作的浮点指令。对最后三种类型的操作而言,有些指令集结构可能根本不会对其提供任何指令支持,而有的指令集结构可能提供了和这些操作相对应的指令。

表 5.2 指令集结构中操作的分类

操作类型	实 例
算术和逻辑运算(Arithmetic Logical)	整数的算术与逻辑运算,加减乘除,与或非
数据传输(Data Transfer)	LOAD/STORE
控制(Control)	分支,跳转,过程调用和返回,自陷
系统(System)	操作系统调用,虚拟存储器管理
浮点(Float)	浮点数算术运算
十进制(Decimal)	BCD 的加减乘除
字符串(String)	字符串移动,比较,查找等
图形(Graphics)	像素操作,压缩/解压操作等

那么,一种指令集结构中的指令到底要支持哪些类型的操作呢? 这就是所谓的指令集结构功能设计问题。在这一问题的处理上,当前有两种截然不同的技术方向,一个方向是强化指令功能,实现软件功能向硬件功能转移,基于这种指令集结构而设计实现的计算机系统称为复杂指令集计算机(CISC)。另一个方向是 20 世纪 80 年代发展起来的精简指令集计算机(RISC),其目的是尽可能地降低指令集结构的复杂性,以达到简化实现、提高性能的目的,这也是当今指令集结构功能设计的一个主要趋势。

下面分别从这两个方向讨论指令集结构功能设计的一些问题。另外,由于分支和跳转等控制操作的行为在上述操作类型中较为特殊,所以在本节的最后将深入讨论支持控制操作的控制流指令。

5.6.1 CISC 指令集结构的功能设计

CISC 指令集结构和 RISC 指令集结构的重要区别就在于其指令的功能强弱上。一般来说,CISC 指令集结构追求的目标是强化指令功能,减少程序的指令条数,以达到提高性能