第 5 章　指令系统与控制器组成


课堂练习

中央处理器（central processing unit，CPU）是计算机的中枢，是计算机系统的运算核心和控制核心，主要由运算器（arithmetic logic unit，ALU）、控制器（control unit，CU）、一组寄存器及相关总线组成。了解计算机的工作原理的关键是搞清 CPU 的工作原理。关于运算器的简单原理已经在 1.3.2 节中介绍，本章主要对控制器进行逻辑分析。控制器的核心功能是分析指令，设计依据是指令系统。

5.1　处理器的外特性——指令系统

5.1.1　指令

1. 指令的概念

在计算机中，指令（instruction）是要求计算机完成某个基本操作的命令。计算机程序就是由一组指令组成的代码序列。这里所说的“基本”是针对具体的 CPU 而言的，不同的 CPU 所指的“基本”二字的意义不同，所能执行的基本操作的数量和种类也不同。但是，任何 CPU 都必须满足最小完备性原则，即它所能执行的基本操作，必须能组成该 CPU 所承担的全部功能。也就是说，有的 CPU 的基本操作多一些、复杂一些，有的 CPU 的基本操作少一些、简单一些，但它们的组合效果应当相同。例如，有的 CPU 将乘法作为基本操作之一；有的 CPU 没有乘法操作，但可以使用加法和移位操作组成乘法操作。

在 1.4.4 节中已经介绍过，一条指令要由操作码和地址码两部分组成，并且可以按照地址的数量把指令分为 3 地址指令、2 地址指令、1 地址指令和 0 地址指令。

2. 指令的长度与操作码扩展

一条指令的长度由操作码和各地址码的长度决定。指令地址码的长度由指令的寻址空间——存储器的容量决定。例如，一个 10b 的地址码的寻址空间为 2^{10}。一个 1GB 的内存，地址码需要 30b；对于 3 地址指令，指令的长度就需要 100b 左右，这样的指令就太长了。但是多地址指令比少地址指令编写出来的程序长度要小，并且执行速度要快。

操作码的长度由操作的种类决定，一个包含 n 位的操作码最多能表示 2^n 种操作。不同的计算机中，可以采用定长操作码——操作码占有固定长度的位数，也可以采用变长操作码——操作码的长度不固定。变长操作码可以用扩展窗口将部分地址作为操作码使用，以增加指令条数。例如：

（1）原始操作码占 4b，用 0000～1110 定义 15 条 3 地址指令，留下 1111 作为扩展窗口，与 A3 一起组成扩展操作码。

（2）由于长码中不可出现短码，所以只能用 1111 0000～1111 1110 定义 2 地址指令，共 15 条。留下 1111 1111 作为 1 地址指令的扩展窗口，与 A2 组成扩展操作码。

（3）同理，用 1111 1111 0000～1111 1111 1110 定义 15 条 1 地址指令。

（4）最后，用 1111 1111 1111 0000～1111 1111 1111 1111 定义 16 条 0 地址指令。

例 5.1 某计算机字长为 16b，操作码占 4b，有 3 个 4b 的地址码，试说明如何扩展该机器的指令系统。

解：在定长操作码系统中，操作码的长度与地址码是冲突的，即地址码越长，操作码就要越短。而操作码短了，指令系统中的指令数就少了。例如在本题中，操作码只有 4 位，指令系统中最多只能有 16 条指令，这常常是不够的。为了解决这个矛盾，可以采用变长操作码，即减少地址数，扩展操作码。

在进行操作码扩展时，要特别注意的是，长码中不可出现短码。因为编译器会首先从短码开始区分不同的指令类型。短码至少要留出一位用于连接扩展位，称为扩展窗口位。

5.1.2 指令系统及其描述

1. 指令系统及其意义

一个 CPU 所能承担的全部基本操作由一组对应的指令描述。这组完整地描述该 CPU 的指令就称为该 CPU 的指令系统（command system，instruction system，command set，instruction set）。指令系统表明了 CPU 能执行哪些基本操作，由此也称指令系统决定了 CPU 的外特性。用户要使用一台计算机，就要用该机器的 CPU 指令系统中的指令与其交互，用不同的指令，指示其执行不同的操作。所以，对于一台计算机来说，它的 CPU 的指令系统就是该计算机的机器语言。

另一方面，功能模拟和结构模拟是研究、制造任何一种机器的两个最重要的途径。20 世纪 60 年代人们从程序员的角度观察机器属性，开始用计算机体系结构来统一功能和结构这两方面。但是，功能和结构二者是存在差别的，通常把计算机的功能方面称为外（宏）体系结构，把计算机的结构方面称为内（微）体系结构。由于 CPU 的功能是取指令—分析指令—执行指令，所以一个 CPU 设计所依据的功能也来自指令系统，即指令系统是 CPU 设计的基本依据。要设计一个 CPU，要先为它设计指令系统。

2. 指令系统的品质要求

指令系统的品质要求如下。

（1）最小完备性。完备性是指对于一台通用计算机来说，用它的 CPU 提供的指令系统直接提供指令，在编写各种程序时应当足够使用，而不必用别的软件补充。

（2）有效性。有效性是指利用该指令系统所提供的指令编写的程序能够高效率运行。

（3）规整性。规整性是指指令系统的对称性、匀齐性，指令格式和数据格式的一致性。

（4）兼容性。兼容性是指计算机的体系结构设计基本相同，计算机之间具有相同的基本结构、数据表示和共同的基本指令集合，因此指令系统是兼容的，即同一个软件可以不加修改就在其他系统结构相同的机器上使用。

3. 指令系统的描述

在表现形式上，机器语言就是用 0、1 码描述的指令系统。用它编写程序，难读、难记、难查错，给程序设计和计算机的推广、应用、发展造成极大困难。面对这一不足，人们最先是采用一些符号来代替 0、1 码指令，如用 ADD 代替“加”操作码等。这种语言称为符号语言。下面是几条符号指令与其对应的机器指令代码的例子：

```
MOV    AH，01H          ; 机器指令代码：B401H
XOR    AH，AH           ; 机器指令代码：34E2H
MOV    AL，[SI+0078H]   ; 机器指令代码：8A847800H
MOV    BP，[0072H]      ; 机器指令代码：8B2E7200H
DEC    DX               ; 机器指令代码：4AH
IN     AL，DX           ; 机器指令代码：ECH
```

符号语言方便编程，用它编写程序的效率高，写出的程序易读性好，提高了程序的可靠性。但是，符号语言是不能直接执行的，必须将之转换为机器语言才能执行。

符号语言程序转换为机器语言程序的方法是查表，这是非常简单的工作。为了将这种查表工作自动化，除了正常的指令外，还需要添加一些对查表进行说明的指示指令——伪指令，这种查表工作称为汇编。为了进行自动查表，还需要一些指示性指令。用符号语言描述并增加了指示性指令的指令系统称为汇编语言。汇编语言为程序员提供了极大方便，也提高了程序的可靠性。通常在介绍指令系统时，采用的都是汇编语言。

汇编语言指令的一般形式如下。

```
标号：操作码  地址码(操作数)；注释
```

下面是一段用 Intel 8086 汇编语言描述的计算 $A=2+3$ 的程序。

```
        ORG  C0H       ; C0H 为程序起始地址
START:  MOV  AX，2     ; 2→AX，AX 为累加器，START 为标号
        ADD  AX，3     ; 3 + (AX)→AX
        HALT           ; 停
        END START      ; 结束汇编
```

由于汇编语言比机器语言有较好的易读性，又与机器语言一一对应，所以机器指令都可按汇编语言符号形式给出。

4. 汇编语言的基本语法

下面以 Intel 8086 汇编语言为例，介绍汇编语言中的几个基本语法。

1）*数据类型*

Intel 8086 汇编语言中允许使用如下形式的数值数据。

（1）二进制数据，后缀为 B，如 10101011B。

（2）十进制数据，后缀为 D，如 235D。

（3）八进制数据，后缀为 Q（本应是 O，为避免与数字 0 相混，用 Q 代），如 235Q。

（4）十六进制数据，后缀为 H，如 BAC3H。

加后缀的目的是便于区分，较多的是采用十六进制，有时也允许用名字来表示数据，如用 PI 代表 3.141593 等。

用引号作为起止界符的一串字符称为字符串常量，如'A'（等价于 41H）、'B'（等价于 42H）、'AB'（等价于 4142H）等。

2）运算符

（1）算术运算符：+、－、*、/。

（2）关系运算符：EQ（相等）、NE（不相等）、LT（小于）、GT（大于）、LE（小于或等于）、GE（大于或等于）。

（3）算符：AND（与）、OR（或）、NOT（非）。

3）操作码

可以用算术运算符，也可以用英文单词。例如，用 SUB 表示相减，用 ADD 表示相加等。

4）地址码及寻址方式

指令中的地址码可以用十六进制、十进制表示，也可以用寄存器名或存储器地址名表示。为了能用有限的地址码访问大容量的存储器，人们研究出了多种寻址方式。

5）标号与注释

汇编语言还允许使用标号与注释，以增加可读性。这部分与机器语言没有对应关系，仅用于使程序容易理解。

5.1.3 RISC 与 CISC

RISC（reduced instruction set computer，精简指令系统计算机）和 CISC（complex instruction set computer，复杂指令系统计算机）是目前设计指令系统的两种不同思路。

1．冯·诺依曼语义差距问题

在指令执行过程中，要求每个部件所完成的基本操作称为微操作。所以，指令就是计算机微操作的组合，不同的微操作组合序列构成了形形色色的指令。每条指令的语义由它所规定的微操作序列定义，如微操作的种类和数量越多，指令的功能越强，使用起来越灵活。而一台机器的指令系统越丰富，这台机器的功能也越强。从使用者的角度来说，指令系统功能越强，编出的程序越简单。但是从计算机设计制造的角度来说，指令系统越简单，CPU 越简单；指令系统越复杂，CPU 越复杂。

随着计算机技术的发展，高级语言不断向人类自然语言方面靠拢。这就使得冯·诺依曼语义差距不断加大。克服这种差距有两种办法：一是增强编译程序的功能；二是扩大处理机指令系统的功能，设法用一条指令代替一段程序。人们传统地认为，指令系统越复杂，含有能用一条指令代替一段程序的指令越多，编译越简单（因为一条高级语言命令要用一段机器语言程序实现），越有利于缩小语义差距。于是，一旦一种机器被设计好之后，不久便又要在其基础上增加一些新的指令和寻址方式，但又不取消旧的指令和寻址方式。这样

做的好处是，通过向上兼容和向后兼容，既可吸引更多的用户，又可降低新产品的开发周期和代价，于是出现了系列机的概念。

在系列机的发展中所增添的新的指令和寻址方式，往往是综合性的复杂指令，是添加更复杂的功能，向高级语言靠拢。这样，指令系统便不断膨胀。例如，PDP-11 机仅有 70 多条指令，而 VAX-11 机具有 16 种寻址方式、9 种数据格式、330 条指令。又如，32b 的 68020 微型计算机的指令种数比 68000 多两倍，寻址方式多了 11 种，达 18 种之多，指令长度变化多端，可从 1 个字（16b）到 16 个字。

很长一段时间内，人们一方面用增加复杂指令的方法缩短语义差距，另一方面又主要靠提高时钟频率和指令解释执行的并行性来提高机器的性能。随着计算机的不断升级，由于复杂的指令系统中指令的差异性增大，流水线及超标量流水线的采用更导致 CISC 结构的 CPU 设计的复杂度按几何级数增长，硬件变得十分庞大。这一方面限制了内部高速缓存的扩大，另一方面使得芯片运行频率的提高非常困难，容易导致芯片工作不稳定。一个令人难忘的例子是 1975 年 IBM 公司投资 10 亿美元研制高速 FS 机，最终却以“复杂结构不宜构成高速计算机”的结论宣告失败。

2．8020 规律

事实让人们从对复杂结构的追求中清醒，开始对指令使用频度进行统计分析。表 5.1 为对 Intel 8088 处理器各类指令使用频度进行分析的结果。可以看出，6 类指令中，数据传输、算术运算和转移指令这 3 类指令的总平均使用频度达 91.82%，而另外 3 类指令的总平均使用频度只有 8.49%。这个结论与 20 世纪 70 年代 HP 公司研究对 IBM 370 高级语言程序中指令使用频度的分析结果，以及 Marathe 在 1978 年对 PDP-11 机在 5 种不同应用领域中的指令混合测试的结果，有类似的结论：典型程序中的 80%只使用处理机中 20%的指令，并且这些指令是简单指令（加、取数、转移等）。也就是说，人们付出极大的代价所增添的复杂指令，只有不到 20%的使用概率，而由于这些低使用频度的复杂指令的存在，使得使用频度高的简单指令的速度也无法提高。例如，希望增加一条指令替换一个子程序，使该项功能的速度提高 10 倍。若子程序的使用频度不超过 1%，替换后的效率提高为（100–（99+1/10））%=0.9%，而另一方面由于复杂性可以使基本时钟频率下降 3%，总的效率反会降低 2.1%。再说，用增加复杂指令达到缩小语义差距的方法，可能对某种高级语言适用，而对其他语言不一定适用。

表 5.1　Intel 8088 处理器各类指令使用频度统计（%）

指令类型	应用程序号							
	F1	F2	F3	F4	F5	F6	F7	平均
数据传输	34.25	35.85	28.84	20.12	25.05	24.33	34.31	30.25
算术运算	24.97	22.34	45.32	43.65	45.72	45.42	28.28	36.24
逻辑运算/位操作	3.40	4.34	7.63	7.49	6.38	3.97	4.89	5.44
字符串处理	2.42	4.22	2.72	2.01	2.10	2.35	2.10	2.86
转移指令	34.84	32.99	15.34	7.63	20.52	25.72	30.29	25.33
处理器控制	0.13	0.26	0.15	0.10	0.24	0.19	0.14	0.19

3. RISC 的基本思想

8020 规律启发了人们的某种思想：只留下最常用的 20%的简单指令，通过优化硬件设计，把时钟频率提得很高，实现整个系统的高性能。这就是 RISC 技术的基本思想。

假定一个程序总的执行时间为

$$T=N \cdot C \cdot S$$

式中，N 是要执行的指令的总条数，C 是每条指令的平均 CPU 周期，S 是每个 CPU 周期的时间。为缩短 T，CISC 技术主要依靠减少 N，但同时要付出提高 C 的代价，也可能还要增加 S；RISC 技术主要是减少 C 和 S，但要付出增加 N 的代价。表 5.2 是对 RISC 和 CISC 的 N、C、S 的对照统计结果。

表 5.2 RISC 与 CISC 的 *N*、*C*、*S* 统计比较

指令系统	*C*	*T*	*N*
RISC	1.3～1.7	<1	1.2～1.4
CISC	4～10	1	1

当 RISC 与 CISC 在相同的时钟周期下比较时，RISC 的每条指令所占用的时钟周期为 1.3～1.7，而 CISC 为 4～10，故这时 RISC 指令执行速度仍为 CISC 的 3～6 倍。尽管 RISC 的 N 因子比 CISC 多 20%～40%，但折算下来，RISC 的性能仍为 CISC 的 2～5 倍。

由于计算机的硬件和软件在逻辑上的等效性，使得指令系统的精简成为可能。早在 1956 年就有人证明，只要用一条“把主存中指定地址的内容同累加器中的内容求差，把结果留在累加器中并存入主存原来地址中”的指令，就可以编出通用程序。有人提出，只要用一条“条件传送”（CMOVE）指令就可以做出一台计算机。1982 年，国外某大学就做出这样一台 8b 的 CMOVE 系统结构的样机，称为 SIC（单条指令计算机）。也有人认为，指令系统是控制器设计的依据，它精简的部分可以通过其他部件及软件（编译程序）的功能来代替。

4．RISC 的外体系结构

指令系统是控制器设计的蓝本。精简指令系统是 RISC 最基本的追求。通过精简指令系统，可以简化控制器，减少控制器面积，进而带来提高处理速度（降低 C 和 T），增加通用寄存器等一系列好处。RISC 的指令系统有如下一些特点。

1）指令种数较少（一般希望少于 100 种）

由于指令系统精简，在机器设计时就要仔细选择指令系统，力求很好地支持高级语言。例如，RISCⅡ的 39 条指令可以分为如下 4 类。

（1）寄存器-寄存器操作：移位、逻辑、整数运算等 12 条。

（2）取/存数指令：取存字节、半字、字等 16 条。

（3）控制转移：条件转移、调用、返回共 6 条。

（4）其他：存取程序状态字和程序计数器等 5 条。

应该说，用这些指令并在硬件系统的辅助下，足以实现其他指令的功能。例如，RISC 机中没有寄存器之间的数据传送指令（MOVE），可以用加法指令 $R_s+R_o \rightarrow R_d$ 代替，其中 R_o

是一个恒为零的寄存器。还有取负可以用 R_o-R_s→R_d 代替，清除寄存器可以用 R_o+R_o→R_d 代替等。

此外，转移指令可由两个算术操作来完成：一个用于进行比较，设置条件；另一个用于计算目标地址。地址计算由单独提供的加法器完成，这样可以提高布尔表达式的求值效率，而无须设置条件码。

当然，对于商品化的 RISC 机，因用途各异，一般会增加一些指令，扩充指令系统，但不外乎如下 4 种。

（1）浮点指令。

（2）特权指令。

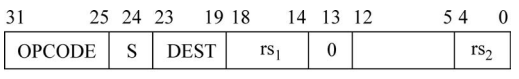
（3）寄存器间数据交换等。

（4）一些专用简单指令。

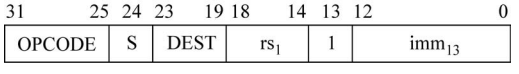
2）指令格式种类少（希望只有 1～2 种）且要求长度一致

RISCⅡ的指令格式只有两种：短立即数格式和长立即数格式，并且指令字中的每个字段都有固定的位置。寄存器-寄存器操作指令仅有短立即数一种格式，有些指令（如条件转移指令）则有两种格式。指令格式少且要求长度一致，便于实现简单而又统一的译码。

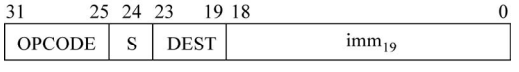
如图 5.1 所示，在 32 位的指令字中，第 25～31 位为操作码；第 24 位为是否置状态位的标志（RISCⅡ只有 4 个状态位：Z、N、V、C）；第 19～23 位为 DEST 字段（对寄存器-寄存器指令，DEST 为目标地址，以 rd 表示；在条件转移指令中，DEST 中的第 23 位无用，其他 4 位表示转移条件）；第 0～18 位为操作数字段。短立即数格式适于算术逻辑指令，需要两个源操作数，它的第 0～18 位分成 3 部分：第 14～18 位为第一源操作数（用 rs1 指出）；第 13 位的 *i* 用于指明第二源操作数是在用 rs_2 指出的寄存器中（*i* = 0，只用 0～4 位，5～12 位不用），还是一个立即数 imm_{13}（*i* = 1，使用第 0～12 位，共 13 位）。长立即数格式用于只需一个源操作数的指令，这时操作数用一个 imm_{19} 表示；对转移指令，imm_{19} 是一个 19 位的相对位移量，表示转移地址相对于程序计数器（PC）的位移量。



(a) 第二源操作数在寄存器中的短立即数格式



(b) 第二源操作数为imm_{13}的短立即数格式



(c) 长立即数格式

图 5.1　RISCⅡ的立即数格式

因为复杂的寻址方式要付出计算有效地址的代价。统计说明，导致增加周期数目的主要环节是访存指令。为消除这一影响，RISC 机器中规定，只有取数（load）和存数（store）两条指令可以访存，以缓解对主存带宽的要求，即 RISC 的大部分指令用于在寄存器间进行数据传送。

5. RISC 与 CISC 的比较

表 5.3 对 RISC 与 CISC 进行了比较。

表 5.3 RISC 与 CISC 比较

比较内容	RISC	CISC
指令数目	一般少于 100 条	一般大于 200 条
寻址方式种类	少	多
指令格式种类	少	多
通用寄存器数量	多	少
指令字长	定长	变长
执行速度	高	较低
不同指令的执行时间	多数在一个周期内	相差甚远
各种指令的使用频率	接近	相差甚远
设计及制造的方便性	高	低
可靠性	逻辑简单，可靠性高	逻辑复杂，可靠性低
控制器实现分时	组合逻辑电路	微程序

5.2 CISC 的成功案例：80x86

5.2.1 80x86 技术概述

Intel 公司是世界上最早生产微型计算机 CPU 的大型微电子厂商。它于 1971 年 11 月 15 日推出了全球第一款 4b 商用微处理器 4004，一年后推出了全球第一款 8b 商用微处理器 8008，1978 年推出了首枚 16b 微处理器 8086。8086 被认为是一款非常成功的微处理器架构，以此为基础经过不断改进和扩充，以后连续推出了 16b 的 80186 和 80286、32b 的 80386 和 80486。这个系列就被通称为 80x86 架构（简称 x86 架构）。由于数字并不能作为注册商标，因此 Intel 公司及其竞争者均在新一代处理器使用可注册的名称，如 Pentium。现时 Intel 公司把 x86-32 称为 IA-32，全名为“Intel Architecture, 32-bit”。

80x86 架构具有如下特点。

（1）80x86 架构最重要的特点是采用了可变指令长度的 CISC。字组（现为 4B 的长度）是以低位字节在前的顺序存储在主存中，因而允许对主存采取不对齐访问。

（2）80x86 使用的是 Intel 和其他几家公司处理器所支持的一组机器指令系统，并且从 8086 到 80186、80286、80386、80486，再到后来的 Pentium 系列，以及现在的多核技术，都是使用一脉相承的 80x86 指令集。这并非说它不发展，它的架构在每次升级中都要引进一些新的思想和机能，如在主存管理和虚拟化方面的硬件支持，在较新的微架构中，还考虑把 80x86 指令转换为更像 RISC 的微指令再予执行，从而获得可与 RISC 比拟的超标量性能。但它每次升级都保持了向前后兼容的特性。这样的指令系统发展及产品系列内部的兼容性大大扩展了 80x86 体系架构的应用范围，将个人用户与企业用户、便携式计算机和超级计算机都包括进来。这是它取得成功的一个经验。

（3）80x86 采用 CISC 架构，具有大量的复杂指令、可变的指令长度、多种的寻址方式这些 CISC 的特点，也是 CISC 的缺点，大大增加了解码的难度。为此，它设置了解码器，用来把长度不定的 80x86 指令转换为长度固定的类似于 RISC 的指令，再交内核执行。解码分为硬件解码和微解码，对于简单的 80x86 指令只要硬件解码即可，速度较快，而遇到复

杂的 80x86 指令则需要进行微解码，并把它分成若干条简单指令，速度较慢且很复杂。

（4）80x86 只有 8 个通用寄存器。这样，CPU 执行时大多数时间是在访问存储器中的数据，而不是寄存器中的数据。这就拖慢了整个系统的速度。

（5）它的寻址范围小，约束了用户需要。

80x86 在 CISC 盛行的时代，由于其灵活性等，取得了辉煌的成功。在 RISC 风行以后，显示出一些不足。但是，目前人们普遍使用的个人计算机基本都是 80x86 架构 CPU。下面粗略地介绍一下它的指令系统，让读者对其有所体会。

5.2.2　8086 的寻址方式

在设计一个 CPU 时，往往要根据用户需求、技术条件和成本的折中，确定字长以及指令格式。指令格式一经确定，指令中各个字段的布局就基本确定，地址字段的长度也就基本确定。一般说来，指令中地址字段的长度是非常有限的。指令设计的一个重要任务就是使用非常有限的地址字段在尽可能大的范围内寻找操作数。于是，就变幻出许多寻址方式。

下面结合 8086 介绍几种常用的寻址方式。

1. 立即寻址

操作数直接包含在指令中，不需要再到寄存器内存中寻址方式称为立即寻址。图 5.2 为 8086 指令 ADD AX 3165H 的存储及执行示意图，这条指令的操作是将 AX 的内容与立即数 3165H 相加，结果存入 AX 中。

2. 寄存器直接寻址

指令需要的操作数放在 CPU 中的某个寄存器中，在指令中只要指定寄存器代号，就可以取得所需的操作数，这种寻址方式称为寄存器直接寻址（简称寄存器寻址）。例如，8086 指令

```
MOV AX，BX
```

是将 BX 中的内容传送到 AX 中。BX 为源地址，AX 为目标地址。

寄存器设在 CPU 内部，存取的速度远高于存储器，所以使用寄存器存放中间结果可以提高程序的运行效率。为了编出高效率的程序，必须先熟悉有哪些寄存器可供编程使用。例如，8086 中有 8 个 8b 的通用寄存器：AL、BL、CL、DL、AH、BH、CH、DH。这 8 个 8b 的通用寄存器也可以合并成 4 个 16b 的通用寄存器：AX（AL 与 AH）、BX（BL 与 BH）、CX（CL 与 CH）、DX（DL 与 DH）。它们的结构及习惯用法如图 5.3 所示。

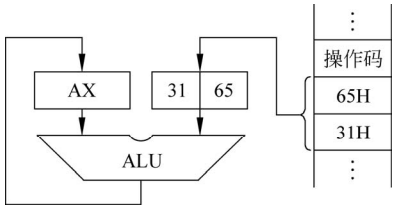


图 5.2　8086 指令 ADD AX 3165H 的存储及执行

图 5.3　8086 的通用寄存器

寄存器寻址指令简单。巧妙地使用寄存器，是提高汇编程序设计的一个关键。

3. 存储器直接寻址

若指令的操作对象存放在内存的某单元中，在指令中直接给出存放操作数的单元地址，就可以取出所需的操作数进行计算，这种寻址方式称为存储器直接寻址。存储器寻址有直接寻址、间接寻址、变址寻址和基址寻址等。图 5.4 为存储器直接寻址的示意图。其中，MOV 是操作码，△是寻址方式，AX 是寄存器代码，3056 是直接地址。这条指令的功能是把 3056H 单元的内容 A3CEH 取出来，送到累加器 AX 中。

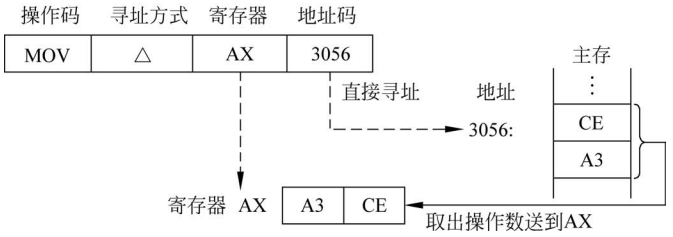


图 5.4 存储器直接寻址示意图

存储器直接寻址灵活性较差，并且受指令长度的限制，寻址范围很有限。例如，一条 2B 指令，操作码占 6b 后，地址码最多能占 10b，寻址范围只有不足 1K。

4. 存储器间接寻址

采用存储器间接寻址方式，由地址码从存储器取出的数不是操作数本身，而是操作数的地址。还需要再以该地址从存储器取出一个数，这个数才是操作数。也就是说，需要两次访问存储器，故称这种寻址方式为存储器间接寻址方式。图 5.5 为存储器间接寻址的示意图。这时 0B58 单元也被称为间址器或地址指针。这条指令的具体操作是从 0B58 单元中取出一个地址，再从该地址 (1A3C) 中取出操作数送到寄存器 AX 中。

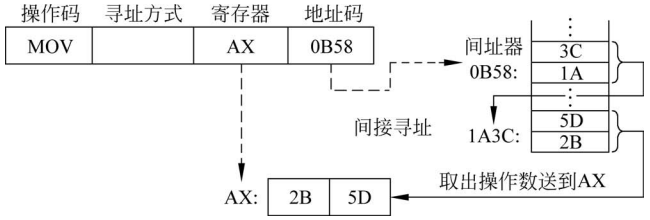


图 5.5 存储器间接寻址示意图

间址器好像是一个“地址寻问处”。这种寻址方式增加了指令的灵活性，只要改变间址器的内容，不改变指令，也可以访问到不同的操作数。同时，全部字长都可以用于存放操作数地址，能扩大寻址范围。但是，由于要多次访问主存，降低了指令的执行速度。提高速度的一个办法是用寄存器作为间址器。

8086 中的间接寻址就是寄存器间接寻址方式。它规定可以用 BX、BP、SI 或 DI 作为（地址）指针。例如指令