模块三　80C51的指令系统和程序设计

（1）了解 80C51 指令系统的构成。
（2）掌握 80C51 单片机的寻址方式。
（3）掌握 80C51 单片机的各种指令及其应用。
（4）掌握汇编语言程序设计方法，为以后实际控制系统的程序设计奠定基础。

3.1　80C51 指令系统概述

对于任何一台计算机，如果只有硬件（称为裸机），而没有软件（即程序）的支持是不能工作的。单片机也不例外，它必须在各种各样的软件支持下才能发挥其运算和控制功能。而程序的最基本单位就是指令，不同的 CPU 具有不同的指令系统，要编写适用于 80C51 单片机的程序就要先学习 80C51 的指令系统。

所谓指令就是 CPU 所能进行的操作。每条指令都对应 CPU 的一种操作，CPU 有多少种操作就应该有多少条指令。把所有指令的集合称为 CPU 的指令系统，80C51 单片机的指令系统共包括 111 条指令。

3.1.1　计算机常用的编程语言

程序是完成某项任务的计算机指令的有序集合。设计程序就要用到程序设计语言，从使用的角度看，程序设计语言可分为如下 3 种：

（1）机器语言。机器语言是计算机唯一能“懂”的语言，直接用二进制代码表示，这种编码称为机器码，或者称为机器指令。只有内存中的机器码能被计算机直接执行。然而，不同类型的 CPU 其机器语言必然不同，且直接用机器语言来编程，很难懂，极易出错，很难用它来进行程序设计。

（2）汇编语言。为了克服使用机器语言编程的缺点，人们采用一些符号来代表地址或数据，用简单明了的助记符（英文缩写）表示指令的操作码，这就形成了汇编语言。用汇编语言编制程序时，程序的每一个语句都与计算机的某一具体操作相对应，因此汇编语言是面向机器的语言，编程者必须熟悉机器的硬件结构。但汇编语言毕竟不是机器语言，计算机不能直接识别，必须将这种符号代码翻译成计算机可识别的机器代码，这种翻译过程称为汇编。汇编工作通常由计算机通过一种具有“翻译”功能的程序——汇编程序来完成（称为机器汇编），也可通过查指令表来完成汇编（称为手工汇编）。

（3）高级语言。上述两种语言都针对计算机硬件，程序的可读性和可移植性都比较差。

为了克服这些缺点，人们又逐步创造出许多面向用户，适合于不同机器、不同场合的高级语言，如 BASIC、FORTRAN、COBOL、Pascal、C 语言以及目前非常流行的可视化程序设计语言 Visual Basic、Visual C++等。这些高级语言更接近人的自然语言，从而使编制的程序直观、简练、易读，且具有较强的通用性。然而，高级语言是更符号化的语言，计算机不能直接接受，需要经过复杂的编译程序或解释程序的“翻译”才能转化为机器代码，因此它比前两种语言占用更多的内存和更长的执行时间。

由此可以看出，汇编语言的性能介于机器语言和高级语言之间。它与机器语言相比，易学、易记、好理解、使用方便；与高级语言相比，占用存储空间少、执行速度快，特别是其具有直接针对输入/输出端口的操作指令，便于自控及检测系统中数据的采集及发送，而许多高级语言却无此功能。因此，对于要求反应灵敏与控制及时的工控、检测等实时控制系统，以及要求体积小、系统小的许多“电脑化”产品，采用汇编语言的优越性是显而易见的。目前单片机系统普遍采用汇编语言编程，当然高级用户也常采用 C51 编程。

3.1.2 汇编语言指令格式

80C51 单片机的汇编指令由操作码和操作数两大部分组成，指令格式如下：

[标号:] 操作码 [操作数 1,][操作数 2,][操作数 3][;注释]

例如：

```
LB:MOV A,R2                    ;把寄存器 R2 中的数送入 A
```

（1）标号：是该指令的符号地址（如上述指令中的“LB”）。并非每条指令都需要标号。当一条指令含有标号时，那么在其他指令的操作数中就可以引用该标号，以便控制程序的转移或寻址。不同的汇编程序对标号有不同的要求，一般的汇编程序规定标号受下列条件约束：

1）标号必须是由大小写英文字母打头，1～8 个字母、数字或下划线组成的字符串。

2）标号不能与寄存器、端口及指令助记符重名。

标号与其后面的操作码之间用半角冒号“：”隔开，也可用半角冒号“：”加若干个空格隔开，冒号“：”的左边不能有空格。

（2）操作码：即指令助记符（如上述指令中的“MOV”），是表示指令功能的英文缩写。

这是指令中唯一不可缺少的核心部分，其作用是规定 CPU 应该完成什么样的操作（如加法、减法、数据传送等）。操作码与操作数之间用若干个空格分开。

（3）操作数：即参与操作的对象（如上述指令中的“A”和“R2”）。它与操作码一起确定指令所要执行的具体操作。操作数可以是常数、寄存器、地址及表达式等。大多数指令有一个或两个操作数，个别指令有 3 个操作数，但也有几条指令的操作数隐含在操作码中，形式上没有操作数（如 RET 指令）。如果一条指令有两个或 3 个操作数，则操作数之间要用半角逗号“,”隔开，也可用半角逗号“,”加若干个空格隔开，但逗号的左边不能有空格。

（4）注释：是对指令功能的解释（如上述指令中的“A←(R2)”）。注释可有可无，其主要作用是增强程序的可读性，但它不是汇编语言源程序的功能部分，所以汇编时并不生成目标代码，对机器的工作没有影响。

注释与其前面的指令主体之间用半角分号“;”隔开，也可用若干个空格加半角分号“;”隔开。

辩论

计算机常用的程序设计语言有汇编语言和高级语言。

正方：汇编语言比高级语言使用方便，编程容易，所以目前单片机系统普遍采用汇编语言编程。

反方：高级语言比汇编语言使用方便，编程容易，高级用户常采用高级语言编程。

马上行动

你认为正、反哪一方的论据站得住脚？赶快发表你自己的见解吧！

现场研讨

1．80C51 单片机汇编语言的特点是什么？汇编语句的格式如何？

2．你入学后，学过几种计算机语言，各有哪些优缺点？

3.2　指令的寻址方式

指令的操作对象大多是各类数据，而数据在寄存器、存储器中可以用多种方式存取。指令执行过程中寻找操作数的方式，称为指令的寻址方式。

原理分析

1．寻址的概念

寻址方式的“寻址”是指寻找参与操作的数据所在存储器单元的地址。80C51 的大部分指令在执行时都需要使用操作数，因此也就存在着到哪里去取得操作数的问题。因为在计算机中只要给出数据所在存储器单元的地址，就能找到所需要的数据。因此，所谓寻址，其实质就是如何确定操作数所在单元地址的问题。

2．寻址方式

80C51 指令系统有以下 7 种寻址方式：

（1）立即寻址方式。

所谓立即寻址，就是在指令中直接给出参与操作的数据本身，即操作数直接出现在指令中。这种形式的操作数被称为立即数，以在数据前加“#”号表示。

例如：

```
MOV A,#40H                    ;将立即数 40H 送到 A 中
MOV DPTR,#2345H               ;将立即数 2345H 送到 DPTR 中
```

（2）直接寻址方式。

所谓直接寻址，就是在指令中直接给出参与操作的数据所在内部 RAM 单元的地址，即操

作数直接以单元地址的形式出现在指令中。直接寻址方式适用于内部 RAM 的低 128 字节和特殊功能寄存器。

例如：

MOV 40H,A　　;将 A 中的内容传送到内部 RAM 的 40H 单元

例如 MOV A,P0 等同于 MOV A,80H，因为 80H 就是 P0 口的地址。

（3）寄存器寻址方式。

所谓寄存器寻址，就是在指令中直接以寄存器的形式给出参与操作的数据，即操作数放在指定的寄存器中。寄存器寻址中的寄存器包括 R0～R7、A、B、AB、DPTR 等。

例如：

MOV A,R3　　;将当前工作寄存器 R3 中的内容传送到累加器 A

若 R3 的内容为 4FH，则执行该指令后累加器 A 的内容也变为 4FH。

（4）寄存器间接寻址方式。

所谓寄存器间接寻址，就是在指令中所给定的寄存器中存放的不是操作数本身，而是操作数所在内部 RAM 或外部 RAM 单元的地址，即操作数通过寄存器间接获得。为了与寄存器寻址相区别，在寄存器名称前加上符号“@”表示间接寻址。可用于间接寻址的寄存器有 R0、R1、SP 及 DPTR，其中的 SP 在间接寻址方式中以隐含方式出现。

假设 R0 的内容为 65H，则指令“MOV A,@R0　;将 R0 中的值作为地址，到这个地址中取数，然后送到 A 中去”的功能是以寄存器 R0 的内容“65H”为地址，将与该地址对应的内 RAM 单元中的数据送到累加器 A。若内部 RAM 65H 单元的内容为 79H，则执行该指令后，累加器 A 的内容即为 79H。

（5）变址寻址（基址+变址寄存器间接寻址）方式。

基址+变址寄存器间接寻址方式又称变址寻址方式，在指令中以“@A+DPTR”或“@A+PC”的形式出现，表示以数据指针 DPTR 或程序计数器 PC 的内容为基地址（16 位），以累加器 A 中的内容为地址偏移量（8 位），二者之和即为操作数所在程序存储器单元的物理地址。

假设累加器 A 的内容为 05H，DPTR 的内容为 0400H，程序存储器 0405H 单元的内容为 2DH，则指令“MOVC A, @A+DPTR”执行后累加器 A 的内容为 2DH。

这种寻址方式只适用于程序存储器。

（6）相对寻址方式。

所谓相对寻址，就是以 PC 的当前值为基准，加上指令中给出的相对偏移量“rel”而形成有效的物理地址。这种寻址方式出现在转移指令中，用来指定程序转移的目标地址。目前在单片机程序设计中，一般采用机器汇编，通常用标号来表示目标位置，基本不需要人工计算相对寻址的值，因此学生不需要详细分析相对寻址的原理。

（7）位寻址方式。

位寻址是指寻找某一位地址的状态。采用位寻址指令，其操作数是 8 位二进制数中的某一位，指令中给出的是位地址。

位寻址的寻址空间为内部 RAM 中位寻址区 20H～2FH 单元的 128 位（位地址为 00H～7FH）及 11 个特殊功能寄存器的 83 个可寻址位。

例如：

MOV C,90H

该指令的功能是将 Pl.0 口（地址为 90H）的状态送位累加器 C。

区分位地址与字节地址主要看指令是位操作指令还是字节操作指令。如果是位操作指令，则操作数中的地址一定是位地址，反之为字节地址。

80C51 指令系统的各种寻址方式有不同的寻址空间，读者在使用时要特别注意，在不同的存储区中应采用不同的寻址方式。

3．指令中的操作数标记

80C51 单片机的指令系统采用了多种记忆符号（助记符），要真正理解每一条指令，必须先了解各种助记符的意义。

- Rn（n=0～7）：表示当前工作寄存器区的 8 个寄存器 R0～R7。
- @Ri(0,1)：表示当前工作寄存器中可用于间接寻址的两个寄存器 R0 和 R1。
- direct：表示内部 RAM 单元的地址，取值为 00H～7FH 时对应内部 RAM 的低 128 字节单元；取值为 80H～0FFH 时，表示特殊功能寄存器。
- @DPTR：表示以间接寻址方式出现的数据指针 DPTR。
- #data：出现在指令中的 8 位常数，也称 8 位立即数。
- #data16：出现在指令中的 16 位常数，也称 16 位立即数。
- bit：可直接寻址位的位地址。
- /bit：在位操作指令中，表示对该位（bit）先取反，然后再参与运算，但不改变指定位（bit）的原值。
- (x)：表示 x 寄存器的内容或地址为 x 的存储单元中的内容。
- ((x))：表示由间接寄存器 x 所间接寻址的存储单元的内容。
- $：代表当前指令的首地址，通常用在相对转移指令中，可以组成表达式，如$-3 指以当前指令首地址为基础向前查第 3 个单元。

另外在注释中，常用箭头“←”表示数据传送方向；箭头右边为源操作数，一般表示到数字；箭头左边为目的操作数，表示到存储器单元或寄存器，不能表示到数字。

4．7 种寻址方式所对应的地址空间

7 种寻址方式所对应的地址空间如表 3-1 所示。

表 3-1　寻址方式所对应的地址空间

寻址方式	地址空间
立即寻址	立即数
直接寻址	片内 RAM 低 128 字节或特殊功能寄存器
寄存器寻址	工作寄存器 R0～R7、A、AB、DPTR
寄存器间接寻址	片内 RAM 和片外数据空间
变址寻址	程序空间
相对寻址	程序空间
位寻址	片内 RAM（20H～2FH）位寻址空间和可位寻址的特殊功能寄存器位地址 C

1．访问内部 RAM 单元、外部 RAM 单元、外部程序存储器、特殊功能寄存器 SFR 可以

分别采用哪些寻址方式？

2．如何区分操作数的地址是位地址还是字节地址。

动手演练

分析下面各条指令，说明源操作数的寻址方式。

```
MOV A,40H
MOV R0,A
MOV P1,#0F0H
MOV @R0,20H
MOV 50H,R0
MOV A,@R0
MOV P2,P1
MOVC A,@A+DPTR
```

3.3 数据传送类指令

数据传送类指令是指令系统中最基本的、编程时使用最频繁的一类指令。

数据传送指令的基本助记符为“MOV”，通用格式如下：

MOV(目的操作数),(源操作数)

数据传送指令的功能就是把由“源操作数”所指定的数据传送（实际上是复制）到“目的操作数”所指定的存储单元或寄存器中，而“源操作数”所指定的数据不变。

此类指令不影响 Cy、AC 及 OV 标志，只影响奇偶标志 P。

案例分析

案例 1 用仿真软件学习数据传送类指令

马上行动

（1）在 PC 机上双击 Keil uVision 3 图标，进入 Keil 调试环境，选择串行口，单击“确定”按钮。

（2）单击“项目”→“新项目”命令，输入项目名，选择目标 MCU，如 Intel 89C52。

（3）编辑文件：单击“文件”→“新建”命令，在文本编辑器中输入以下程序：

```
        ORG 0000H
START:  MOV R3,#50H
        MOV A,#35H
        MOV R0,#45H
```

```
        MOV DPTR,#1234H
        MOV 30H,A
        MOV P1,A
        MOV 45H,#7CH
        MOV R1,A
        MOV A,@R0
        MOV @R1,#29H
        MOV P2,#11010011B
LOOP0:  MOV A,R3
        MOV R7,#0AH
LOOP:   SJMP START
        END
```

（4）文件保存：单击“文件”→“保存”命令，在对话框中输入文件名。

（5）为项目添加文件：在项目工作区中右击“源代码组 1”，在弹出的快捷菜单中选择“添加文件到组‘源代码组 1’”选项，在弹出的对话框中选中刚保存的文件，单击“确定”按钮完成项目文件的添加。

（6）为项目设置通信口：在项目工作区中右击“目标 1”，在弹出的快捷菜单中选择“为目标‘目标 1’设置选项”→“调试”选项。在新窗口中选择使用 Keil monitor-51 Driver，在出现的窗口中设置串口和波特率（注：波特率必须为 38400）。

（7）文件编译、连接、装载：单击“项目”→“重建所有目标文件”命令，系统自动进行编译，并出现信息窗口。若有语法错误，则需要重新修正，并再次执行“项目”→“重建所有目标文件”命令；若无语法错误，单击“调试”→“启动”→“停止调试”命令进入调试状态。

（8）设置观察窗口：单击“视图”→“存储口窗口”命令，在地址栏中输入 CPU 内部存储区地址，回车后出现地址为 30H 的数据显示于窗口中。

（9）单步执行程序：按 F11 键一条一条地执行下去，注意观察左边寄存器区中相应的寄存器或者 CPU 内部存储区中相应的数据寄存器单元的数据变化。

（10）连续运行：单击“外围设备”→“复位 CPU”命令，使 PC 指向 0000H，单击“调试”→“运行”命令后，程序开始连续运行。如需暂停，单击“调试”→“停止运行”命令。

（11）断点运行：要使程序执行到某条指令处暂停，例如希望程序执行到 LOOP0 处暂停，可按如下操作：将光标移到 LOOP0 处双击即可设置断点，单击“调试”→“运行”命令，程序将在 LOOP0 行停止运行。

（12）复位：单击“外围设备”→“复位 CPU”命令，强迫 PC 指向 0000H。

输入程序，用 Keil 软件仿真，进行数据传送指令学习。

原理分析

3.3.1　内部 RAM 单元之间的数据传送指令

内部 RAM 单元之间的数据传送通常是通过 MOV 数据传送指令完成。这类指令称为一般

数据传送指令。

1. 以累加器 A 为目的操作数的传送指令

```
MOV A,Rn                    ;A←(Rn)
MOV A, direct               ;A←(direct)
MOV A, @Ri                  ;A←((Ri))
MOV A, #data                ;A←data
```

这组指令的功能是把源操作数所指定的数据送入累加器 A，源操作数可以采用寄存器寻址、寄存器间接寻址、直接寻址和立即寻址方式。

【例 3-1】假设(R0)=30H，内部 RAM 中(30H)=0F7H，(68H)=66H，给出执行每条指令后 A 的内容。

解：将每条指令的执行结果写在注释部分：

```
MOV A,R0                    ;(A)=(R0)=30H
MOV A,@R0                   ;(A)=((R0))=(30H)=0F7H
MOV A,68H                   ;(A)=(68H)=66H
MOV A,#18                   ;(A)=18
```

2. 以工作寄存器 Rn（n=0～7）为目的操作数的传送指令

```
MOV Rn,A                    ;Rn←(A)
MOV Rn,direct               ;Rn←(direct)
MOV Rn,#data                ;Rn←data
```

这组指令的功能是把源操作数所指定的数据送入当前工作寄存器区的某个寄存器。

【例 3-2】假设(A)=2FH，内部 RAM(36H)=0E6H，给出执行每条指令后的 Rn 的内容。解：将每条指令的执行结果写在注释部分：

```
MOV R1,A                    ;(R1)=(A)=2FH
MOV R7,36H                  ;(R7)=(36H)=0E6H
MOV R4,#96H                 ;(R4)=96H
```

3. 以间接寄存器为目的操作数的传送指令

```
MOV @Ri, A                  ;(Ri)←(A)
MOV @Ri, direct             ;(Ri)←(direct)
MOV @Ri, #data              ;(Ri)←data
```

这组指令的功能是把源操作数所指定的数据送入 R0 或 Ri 所指向的内部 RAM 单元。

【例 3-3】假设(A)=2FH，内部 RAM(36H)=0E6H，(R0)=30H，(R1)=32H，给出每条指令的执行结果。

解：将每条指令的执行结果写在注释部分：

```
MOV @R1,A                   ;内部 RAM(32H)=(A)=2FH
MOV @R1,36H                 ;内部 RMA (32H)=(36H)=0E6H
MOV @R0,#56                 ;内部 RAM (30H)=56=38H
```

4. 以直接地址为目的操作数的传送指令

```
MOV direct, A               ;direct←(A)
MOV direct, Rn              ;direct←(Rn)
```

MOV direct, @Ri　　;direct←((Ri))
MOV direct1,direct2　　;direct1←(direct2)
MOV direct, #data　　;direct←data

这组指令的功能是把源操作数所指定的数据送入直接地址所对应的内部 RAM 单元或 SFR。这里再次强调 direct 表示内部 RAM 或 SFR 的字节地址。

【例 3-4】分别用 3 种方法将内部 RAM 中 30H 单元的内容传送到 40H 单元。

解：（1）直接将 30H 单元的内容传送到 40H 单元，只需一条指令：

MOV 40H,30H　　;(40H)=(30H)

（2）用间接寄存器指向源地址，指令序列如下：

MOV R0,#30H　　;(R0)=30H
MOV 40H,@R0　　;(40H)=((R0))=(30H)

（3）用间接寄存器指向目标地址，指令序列如下：

MOV R0,#40H　　;(R0)=40H
MOV @R0,30H　　;(40H)=(30H)

5. 以 DPTR 为目的操作数的传送指令

MOV DPTR,#data16　　;DPTR←data16

这条指令的功能是把 16 位立即数送入 DPTR，这是整个指令系统中仅有的一条 16 位数据传送指令，使用该指令可为外部程序存储器或外部数据存储器建立一个地址指针。16 位寄存器 DPTR 由 DPH 和 DPL 组成。这条指令执行的结果是把高 8 位立即数送入 DPH，低 8 位立即数送入 DPL。

3.3.2　栈操作指令

在 80C51 内部 RAM 区可以设定一个先进后出的区域作为一个堆栈，在特殊功能寄存器中有一个堆栈指针 SP，它始终指向栈顶位置。引入堆栈的目的，就是在程序调用或中断时保护和恢复现场数据及断点地址。在指令系统中有两条栈操作指令：进栈指令（PUSH）和出栈指令（POP）。

1. 进栈指令

PUSH direct　　;SP←(SP)+1（先变指针），(SP)←(direct)（再压栈）

这条指令先对指针 SP 加 1，然后将直接寻址的内部 RAM 单元或 SFR 的内容传送到 SP 所指向的内部 RAM 单元。

2. 出栈指令

POP direct　　;direct←((SP))（先出栈），SP←(SP)-1（再变指针）

这条指令先将堆栈指针 SP 指向的内部 RAM 单元的内容送入直接寻址的内部 RAM 单元或 SFR，再对栈指针 SP 减 1。

【例 3-5】假设(SP)=30H，(DPTR)=0123H，分析连续执行下列指令序列后 DPTR 及 SP 的内容。

解：先分析每条指令执行后的结果：

PUSH　DPL　　;(SP)=30H+1=31H，(31H)=(DPL)=23H
PUSH　DPH　　;(SP)=31H+1=32H，(32H)=(DPH)=01H

```
POP     DPL                ;(DPL)=(32H)=01H，(SP)=32H-1=31H
POP     DPH                ;(DPH)=(31H)=23H，(SP)=31H-1=30H
```

所以程序之行后，(SP)=30H，(DPTR)=2301H。

本例说明当进栈和出栈的顺序违反“后进先出”的原则后，将不能正确恢复原数据。

3.3.3 数据交换指令

数据交换主要在内部 RAM 和累加器 A 之间进行，有整字节和半字节两种交换指令。

1. 整字节交换指令

```
XCH A,Rn                   ;(A)↔(Rn)
XCH A,direct               ;(A)↔(direct)
XCH A,@Ri                  ;(A)↔((Ri))
```

这组指令的功能是将累加器 A 的内容与源操作数所指定的数据进行交换。

2. 低半字节交换指令

```
XCHD A,@Ri                 ;(A.3～A.0)↔((Ri.3～Ri.0))
```

这条指令的功能是将累加器 A 的低半字节（D3～D0）与由@Ri 指定的内部 RAM 单元的低半字节（D3～D0）交换，高半字节（D7～D4）保持不变。

3. 高、低半字节互换指令

```
SWAP A                     ;(A.3～A.0)↔(A.7～A.4)
```

这条指令的功能是将累加器 A 的高半字节（D7～D4）和低半字节（D3～D0）交换。

【例 3-6】假设(A)=12H，(R0)=34H，内部 RAM(34H)=56H，分析每条指令的执行结果。

解：将每条指令的执行结果写在注释部分：

```
XCH A,@R0                  ;(A)=((R0))=(34H)=56H，(34H)=(A)=12H
XCHD A,@R0                 ;(A)=16H，(34H)=52H
SWAP A                     ;(A)=21H
```

3.3.4 累加器 A 与外部 RAM 的数据传送指令

外部数据存储器与内部数据存储器的数据传送只能通过累加器 A 进行，且只能采用间接寻址方式。

1. 读外部数据存储器或外部 I/O 口的指令

```
MOVX A, @Ri                ;A←((Ri))
MOVX A, @DPTR              ;A←((DPTR))
```

2. 写外部数据存储器或外部 I/O 口的指令

```
MOVX @Ri,A                 ;(Ri) ←(a)
MOVX @DPTR,A               ;(DPTR)←(A)
```

这组指令功能是访问外部 RAM，源操作数采用寄存器间接寻址或寄存器寻址。

【例 3-7】若(DPTR)=3020H，外部 RAM(3020H)=48H，执行指令 MOVX A, @DPTR 后，(A)=48H。

3.3.5　累加器 A 与 ROM 的数据传送指令（查表指令）

程序存储器属于只读类型，因此，有关程序存储器的指令只能实现单方向的数据传送，且只能采用基址+变址寄存器间接寻址方式。

1．基址寄存器为 PC 的查表指令

MOVC A, @A+PC　　;A←((A)+(PC))

2．基址寄存器为 DPTR 的查表指令

MOVC A, @A+DPTR　　;A←((A)+(DPTR))

这组指令的功能是读程序存储器 ROM。执行结果只和 PC 或 DPTR 及累加器 A 的内容有关，与该指令存放的地址及数据表格存放的地址无关，因此表格的大小和位置可以在 64KB 程序存储器中任意安排，表格可以被多个程序模块共享。源操作数的寻址方式采用变址寻址。

【例 3-8】若(DPTR)=3000H，(A)=20H，执行指令 M0VC A, @A+DPTR 后，程序存储器 3020H 单元的内容送入 A。

1．MOVC A,@DPTR 与 MOVX A,@DPTR 指令有什么不同？

2．访问特殊功能寄存器 SFR 可以采用哪些寻址方式？

3．访问内部 RAM 单元可以采用哪些寻址方式？

4．访问外部 RAM 单元可以采用哪些寻址方式？

5．访问外部程序存储器可以采用哪些寻址方式？

1．试编写程序，将内部 RAM 的 2FH、2EH 和 2DH 三个连续单元的内容依次存入 25H、24H、23H 单元。

2．假定(SP)=60H，(A)=30H，(B)=70H，执行下列指令：

PUSH A

PUSH B

后，SP、61H 单元及 62H 单元的内容各是多少？

3．假定(SP)=62H，(61H)=30H，(62H)=70H，执行下列指令：

POP DPH

POP DPL

后，SP、DPTR 的内容各是多少？

4．若(A)=40H，(R0)=50H，(50H)=60H，(40H)=08H，试分析执行下列程序段后上述各单元内容的变化。

MOV A,@R0

MOV @R0,40H

MOV 40H,A
MOV R0,#F7H

5．设片内 RAM 中的(40H)=50H，写出执行下列程序段后寄存器 A 和 R0，以及片内 RAM 中 50H 和 51H 单元的内容为何值？

MOV A,40H
MOV R0,A
MOV A,#00
MOV @R0,A
MOV A,#30H
MOV 51H,A
MOV 52H,#70H

6．设堆栈指针(SP)=60H，片内 RAM 中的(30H)=24H，(31H)=10H，执行下列程序段后，61H、62H、30H、31H、DPTR 及 SP 中的内容将有何变化？

7．在 8051 的片内 RAM 中，已知(20H)=30H，(30H)=40H，(40H)=50H，(50H)=55H，分析下面各条指令，说明源操作数的寻址方式，分析按顺序执行各条指令后的结果。

MOV A,40H
MOV R0,A
MOV P1,#0F0H
MOV @R0,20H
MOV 50H,R0
MOV A,@R0
MOV P2,P1

8．试写出完成以下每种操作的指令序列。

（1）将 R1 的内容传送到 R0。
（2）外部 RAM 单元 1000H 的内容传送到内部 RAM 单元 50H。
（3）内部 RAM 单元 50H 的内容传送到寄存器 R1。
（4）外部 RAM 单元 2000H 的内容传送到寄存器 R1。

3.4 算术运算类指令

算术运算类指令共 24 条，包括加、减、乘、除 4 种最基本的算术操作。这类指令多数以 A 为源操作数之一，同时又使 A 为目的操作数。借助溢出标志，可对带符号数进行补码运算；借助进位标志，可实现多字节加减运算；借助调整指令，也可对压缩 BCD 码进行运算。

案例 2 用仿真软件学习算术运算类指令

马上行动

（1）在 PC 机上双击 KEIL uvision 3 图标，进入 KEIL 调试环境，编辑文件，在文本编辑器中输入以下程序：

```
        ORG 0000H
START:  MOV R3,#50H
        MOV A,#35H
        ADD A,#45H
        ADD A,R3
        ADDC A,#79H
        INC A
        SUBB A,#23H
        DEC A
        MOV B,#29H
        MUL AB
        MOV A,#0FBH
        MOV B,#18
        DIV AB
LOOP:   SJMP START
        END
```

（2）保存文件。文件编译、连接、装载。若有语法错误，则需要重新修正，若无语法错误，进入调试状态。

（3）设置观察窗口，单步执行程序，按 F11 键一条一条地执行下去，注意观察左边寄存器区中相应的寄存器或者 CPU 内部存储区中相应的数据寄存器单元的数据变化。验证运算结果是否正确。

原理分析

3.4.1　加法指令

80C51 的指令系统只有 8 位数据的加法运算指令，包括半加、全加及增量运算。

1. 不带进位的加法指令（半加）

不带进位的加法运算相当于数字电路中的半加器，参与运算的数据只有被加数和加数，而不考虑低位的进位，指令格式如下：

```
ADD A,Rn          ;A ←(A)+(Rn)
ADD A,direct      ;A ←(A)+(direct)
ADD A,@Ri         ;A ←(A)+((Ri))
ADD A,#data       ;A ←(A)+data
```

这组指令常用于多字节无符号数据的加法运算。上面 4 条指令是将累加器 A 的内容和工作寄存器 Rn 的内容、直接地址单元中的内容、由 Ri 间接寻址内部 RAM 单元的内容以及立即数相加，所得的和再存入到累加器 A 中。

上述指令将影响 AC、CY、OV 和 P 标志位。例如在执行上述指令时，第 3 位向第 4 位有进位，则将 AC 置 1，否则清零；第 7 位向上有进位，则将 CY 置 1，否则清零；最高位向上的进位和次高位向最高位的进位相异或若为 1，则置 OV 为 1，否则清零；同样对 P 标志的影响也是如此，若相加后 A 中 1 的个数为奇数，则将 P 置 1，否则清零。

2. 带进位加法指令（4 条）

这 4 条指令除与 ADD 功能相同外，在进行加法运算时还需要考虑进位问题。

```
ADDC A,direct          ;(A)+(direct)+(C)→(A)
ADDC A,#data           ;(A)+ data +(C)→(A)
ADDC A,Rn              ;(A)+(Rn)+(C)→(A)
ADDC A,@Ri             ;(A)+((Ri))+(C)→(A)
```

这组指令的功能是把源操作数与累加器 A 中的内容、连同进位位相加，结果存放在目的操作数 A 中。

这组指令的操作影响程序状态字 PSW 中的 OV、C、AC 和 P 标志。

【例 3-9】设内部 RAM 30H～32H 有 3 个单字节的无符号数，求它们的和，并将和的低字节送入 33H 单元，高字节送入 34H 单元。

解：程序清单如下：

```
MOV A,30H
ADD A, 31H             ;(A)←(30H)+(31H)，请注意标志位 Cy
MOV 33H,A              ; 前两位相加之和的非进位部分暂存于 33H
MOV A,#00H
ADDC A,#00H            ;前两位相加时进位放入 A 中
MOV 34H, A             ;两数相加时的进位暂存于 34H
MOV A,33H              ;前两数相加之和的非进位部分放入 A 中
ADD A,32H              ;加第三个字节数,请注意标志位 Cy
MOV 33H,A              ;和的低字节存入 33H 单元
MOV A,34H              ;将前两数相加时的进位放入 A 中
ADDC A,#00H            ;将 3 数相加时的进位值求出并放入 A 中
MOV 34H,A              ;和的高字节存入 34H 单元
```

3. 加 1 指令

加 1 运算又称增量运算。

```
INC A                  ;A ←(A)+1
INC Rn                 ;Rn ←(Rn)+1
INC direct             ;direct ←(direct)+1
INC @Ri                ;(Ri) ←((Ri))+1
INC DPTR               ;DPTR ←(DPTR)+1
```

这组指令的功能是对累加器、寄存器、内部 RAM 单元或数据指针进行加 1 操作。加 1 指

令的操作不影响标志位的状态（除“INC A”影响 P 标志外）。

【例 3-10】设(R0)=7EH，(DPTR)=10FEH，内部 RAM(7EH)=0FFH，(7FH)=38H，分析下列指令顺序执行的情况。

解：顺序执行指令时，前一条指令的结果当作当前指令的条件，执行结果请看注释。

```
INC @R0                 ;(R0)=7EH，(7EH)=0FFH+1=00H
INC R0                  ;(R0)=7EH+1=7FH
INC @R0                 ;(R0)=7FH，(7FH)=38H+1=39H
INC DPTR                ;(DPTR)=10FEH+1=10FFH
INC DPTR                ;(DPTR)=10FFH+1=1100H
INC DPTR                ;(DPTR)=1100H+1=1101H
```

3.4.2　减法指令

1. 带借位减法指令

指令格式如下：

```
SUBB A,Rn               ;A←(A)-(Rn)-(Cy)
SUBB A, @Ri             ;A←(A)-((Ri))-(Cy)
SUBB A,direct           ;A←(A)-(direct)-(Cy)
SUBB A,#data            ;A←(A)-data-(Cy)
```

这组指令的功能是从累加器 A 中减去由不同寻址方式确定的操作数及进位标志，结果仍在累加器 A 中。指令执行后将影响 Cy、AC、OV 及 P 标志，根据这些标志可以分析两数差值情况：

（1）两个无符号数相减时，若(Cy)=1，表明被减数小于或等于减数，此时必须将累加器 A 中的值连同借位一并考虑才能得到正确结果。

（2）两个带符号数相减时，若(OV)=0，表明没有发生溢出；若(OV)=1，表明发生溢出。

【例 3-11】设(A)=43H，(R1)=6AH，(Cy)=1，分析指令“SUBB A,R1”执行后的结果。

解：计算过程如下：

```
   0100  0011B
   0110  1010B
 -          1B
 ------------------
 1 1101  1000B
```

则结果为：(Cy)=1，(AC)=1，(OV)=0，(P)=0，(A)=0D8H。

2. 减 1 指令

减 1 又称减量运算，减量包括 4 条指令：

```
DEC A                   ;A←(A)-1
DEC Rn                  ;Rn←(Rn)-1
DEC @Ri                 ;@←((Ri))-1
DEC direct              ;direct←(direct)-1
```

这组指令的功能是对累加器、寄存器以及内部 RAM 单元的内容减 1。该组指令不影响标志位（除“DEC A”影响 P 标志外）。

3.4.3 乘法指令

MCS-51 的指令系统只有一条乘法运算指令，且只能进行 8 位无符号数的乘法运算，指令格式如下：

MUL AB ;A←(A)×(B)低字节，B←(A)×(B)高字节

该指令的功能是把累加器 A 和寄存器 B 中的两个 8 位无符号数相乘，乘积又送回到 A、B，其中 B 中存放积的高位字节，A 中存放积的低位字节。

若乘积大于 0FFH，则溢出标志 OV=l。乘法运算总对 Cy 清零，但不影响其他标志位。

3.4.4 除法指令

DIV AB ;A←(A)/(B)（商），B←(A)/(B)（余数）

该指令是把 A 中的 8 位无符号数除以 B 中的 8 位无符号数，商存放在 A 中，余数存放在 B 中。若除数为 0，执行该指令后结果不定，并将 OV 置 l。Cy 在除法运算中总为 0。

【例 3-12】设(A)=50H，(B)=32H，分析下面每条指令的执行结果。

（1）MUL AB

（2）DIV AB

解：每条指令的执行结果写于各自的注释段内，计算过程如下：

```
        01010000  被乘数
  ×     00110010  乘数
 ------------------
        00000000
       01010000
      00000000
     00000000
    0l010000
  + 01010000
 ------------------
   0111110100000    乘积
```

```
          00000001   商
除数 00110010/ 01010000   被除数
       - 00110010
 ------------------
          00011110   余数
```

（1）MUL AB ;(A)=0A0H，(B)=0FH

（2）DIV AB ;(A)=1，(B)=1EH

现场研讨

1．ADD 与 ADDC 指令有什么区别？

2．在进行单字节减法运算时，应该注意什么？

3．已知两乘数分别存放在 R1 和 R0 中，试编程求其积，并存入 R3 和 R2。

动手演练

1．有两个双字节数求和，两个数分别放在 20H～23H，结果放在 30H 和 31H。试编写程

序实现之。

2．分析程序并写出结果。

已知(R0)=20H，(20H)=10H，(P0)=30H，(R2)=20H，执行如下程序段后，(40H)=_____。

```
MOV @R0, #11H
MOV A, R2
ADDC A, 20H
MOV PSW, #80H
SUBB A, P0
MOV 40H, A
```

3．试编写程序，将内部 RAM 的 20H、21H 单元的两个无符号数相乘，结果存放在 R2、R3 中，R2 中存放高 8 位，R3 中存放低 8 位。

3.5　逻辑运算与循环类指令

案例 3　用仿真软件学习逻辑运算与循环类指令

（1）在 PC 机上双击 Keil uVision 3 图标，进入 Keil 调试环境，编辑文件，在文本编辑器中输入以下程序：

```
        ORG 0000H
START:  MOV A,#35H
        ANL A,#47H
        MOV R3,#5BH
        ORL A,R3
        XRL A,#79H
        CPL A
        RR A
        RL A
        RRC A
        RLC A
        SWAP A
LOOP:   SJMP START
        END
```

（2）保存文件。文件编译、连接、装载。若有语法错误，则需要重新修正；若无语法错误，进入调试状态。

（3）设置观察窗口，单步执行程序，按 F11 键一条一条地执行下去，注意观察左边寄存器区中相应的寄存器或者 CPU 内部存储区中相应的数据寄存器单元的数据变化。验证程序运算结果是否正确

逻辑运算和移位指令共有 25 条，有与、或、异或、求反、左右移位、清零等逻辑操作，有直接、寄存器和寄存器间址等寻址方式。这类指令一般不影响程序状态字（PSW）标志。

3.5.1 循环移位指令（4 条）

这 4 条指令的作用是将累加器中的内容循环左移或右移一位，后两条指令是连同进位位 CY 一起移位的。

```
RL A        ;累加器 A 中的内容左移一位（RL，Rotate Left）
RR A        ;累加器 A 中的内容右移一位（RR，Rotate Right）
RLC A       ;累加器 A 中的内容连同进位位 CY 左移一位（RLC，Rotate left carry）
RRC A       ;累加器 A 中的内容连同进位位 CY 右移一位（RRC，Rotate right carry）
```

【例 3-13】阅读下面的程序，说明其所实现的功能。

解：指令的执行情况见每条指令的注释部分。

```
MOV R0,#27H             ;送立即数 27H 到 R0
MOV A,@R0               ;取内部 RAM“27H”单元的内容到累加器
RL A                    ;(27H)×2
MOV R1,A                ;(R1)=(27H)×2
RL A                    ;(27H)×4
RL A                    ;(27H)×8
ADD A,R1                ;(A)=(27H)×2+(27H)×8=(27H)×10
MOV @R0,A               ;保存结果
```

该程序的功能是对内部 RAM“27H”单元的内容乘 10。

3.5.2 求反指令（一条）

这条指令将累加器中的内容按位取反。

```
CPL A                   ;累加器中的内容按位取反
```

【例 3-14】对 40H 单元的内容求补。

解：求补和求补码是不同的，求补即求取反加 1，不分符号位和数字位，不考虑正负；求补码要分符号位和数字位，还要考虑正负。程序清单如下：

```
MOV A, 40H              ;取待求数据
CPL A                   ;取反
INC A                   ;加 1
MOV 40H，A              ;保存结果
```

3.5.3　清零指令（一条）

这条指令将累加器中的内容清零。

CLR A　　; 0→(A)，累加器中的内容清零

3.5.4　逻辑与操作指令（6 条）

这组指令的作用是将两个单元中的内容执行逻辑与操作。

ANL A,Rn　　;A←(A)∧(Rn)
ANL A,@Ri　　;A←(A)∧((Ri))
ANL A,direct　　;A←(A)∧(direct)
ANL A, #data　　;A←(A)∧data
ANL direct,A　　;direct←(direct)∧(A)
ANL direct,#data　　;direct←(direct)∧data

与运算的规则是：相应位“见 0 为 0，全 1 为 1”。

3.5.5　逻辑或操作指令（6 条）

这组指令的作用是将两个单元中的内容执行逻辑或操作。

ORL A,Rn　　;A←(A)∨(Rn)
ORL A, @Ri　　;A←(A)∨((Ri))
ORL A,direct　　;A←(A)∨(direct)
ORL A,#data　　;A←(A)∨data
ORL direct,A　　;direct←(direct)∨(A)
ORL direct,#data　　;direct←(direct)∨data

或运算的规则是：相应位“见 1 为 1，全 0 为 0”。

3.5.6　逻辑异或操作指令（6 条）

这组指令的作用是将两个单元中的内容执行逻辑异或操作。

XRL A,Rn　　;A←(a) ⊕ (Rn)
XRL A,@Ri　　;A←(a) ⊕ ((Ri))
XRL A,direct　　;A←(a) ⊕ (direct)
XRL A,#data　　;A←(a) ⊕ data
XRL direct,A　　;direct←(direct) ⊕ (A)
XRL direct,#data　　;direct←(direct) ⊕ data

异或运算的运算规则是：对应位“相同为 0，不同为 1”，据此可用异或指令判断两数是否相等。

【例 3-15】假设(A)=0D4H，分析每条指令的执行结果。

解：指令执行结果见每条指令注释部分，计算过程如下：

	1101	0100B			1101	0100B			1101	0100B
∧	0000	1111B		∨	0000	1111B		⊕	0000	1111B
	0000	0100B			1101	1111B			1101	1011B

（1）ANL A,#0FH ;用 0 屏蔽 A 的高 4 位，即“高 4 位清零，低 4 位不变”，所以(A)=04H

（2）ORL A,#0FH ;用 1 屏蔽 A 的低 4 位，即“高 4 位不变，低 4 位置 1”，所以(A)=0DFH

（3）XRL A,#0FH ;(A)=0DBH，相当于“高 4 位不变，低 4 位求反”。

所以，想给某位清零，可使之与“0”相与；想给某位置 l，可使之与“l”相或；想对某位求反，可使之与 l 相异或。

【例 3-16】编写程序，将 A 的低 5 位传送到 Pl 的低 5 位，但保持 Pl 的高 3 位不变，A 的内容也保持不变。

解：要保持 A 的内容可以采用堆栈，也可以将 A 的内容暂存到其他寄存器；要传送 A 的低 5 位到 Pl 的低 5 位，最简单最通用的办法就是采用逻辑操作指令。当然，也可以采用位传送指令，只是比较烦琐。采用逻辑运算指令的程序清单如下：

```
PUSH ACC                ;保留 A 的原值
ANL A,#00011111B        ;屏蔽 A 的高 3 位，低 5 位保持不变
ANL Pl,#11100000B       ;将 P1 的高 3 位保持不变，屏蔽低 5 位
ORL Pl,A                ;保持 P1 的高 3 位不变，将 A 的低 5 位传送到 P1 的低 5 位
POP ACC
```

如何使某位清零，如何使某位置 l，如何对某位求反？

1．试编写程序，将 R0 中的低 4 位数与 R1 中的高 4 位数合并成一个 8 位数，并存放在 R0 中。

2．以下程序段执行后，(A)=_____，(30H)=_____。

```
MOV 30H,#0A4H
MOV A,#0D0H
MOV R0,#30H
MOV R2,#5EH
ANL A,R2
ORL A,@R0
SWAP A
CPL A
XRL A,#0FEH
ORL 30H,A
```

3.6　控制转移类指令

原理分析

计算机的程序一般是按顺序执行的（由 PC 自动加 1 实现），但也可以根据需要用控制转移类指令改变程序的执行顺序。

MCS-51 的控制转移类指令共 17 条，包括条件转移指令、无条件转移指令及子程序调用与返回指令。

3.6.1　无条件转移指令

1. 长转移指令

LJMP addr16　　;PC←addr16

指令执行时把 16 位目标地址（addr16）直接送给 PC，从而实现程序转移。本转移指令可覆盖整个程序存储器空间，即 64KB。

在实际使用时，addr16 常用符号地址表示。

2. 短转移指令

AJMP addr11　　;PC←(PC)+2，$PC_{10\sim0}$←addr11，$PC_{15\sim11}$不变

指令执行时，PC（加 2 修改后的值）的高 5 位与指令中的 11 位地址拼接在一起，共同形成 16 位的目标地址，从而达到在现行地址所在 2KB 的范围内转移。

在实际使用时，addr11 常用符号地址来代替。

3. 相对转移指令

SJMP rel　　;PC←(PC)+2，PC←(PC)+rel

该指令的功能是根据指令中给出的相对偏移量 rel 计算出程序将要转移的目标地址：

目标地址=源地址（本指令所在地址）+2+rel

在实际使用时，rel 常用符号地址来表示。

例如，在程序存储器 0100H 单元开始存储有下列程序段，则程序执行完 SJMP 指令后自动转向 LOOP 处执行。

```
0100H SJMP LOOP
0102H MOV A,#10H
…
0130H ADD A,R0
LOOP: A,#40H
```

在这里 LOOP 代表指令的地址是 0130H+1（ADD 指令的字节数）=0131H，所以可求出相对偏移量为：rel=0131H-0100H-2=2FH（只取低字节）。

再如，在实际应用中常使用该指令完成程序“原地踏步”功能，指令形式如下：

SJMP $

其中的“$”表示当前指令的地址，该指令等同于指令：

LOOP: SJMP LOOP

4. 间接转移指令

JMP @A+DPTR ;PC←(A)+(DPTR)

这条指令的功能是把累加器中的 8 位无符号数与数据指针 DPTR 的 16 位数相加，结果作为目标地址，指令执行后不改变 A 和 DPTR 的内容，也不影响标志位。

该指令可根据 A 的内容进行跳转，而 A 的内容又可随意改变，故可形成程序分支。本指令跳转范围为 64KB。

【例 3-17】设某单片机系统的键盘扫描程序所得键值存放在 R2 中，试编写程序，当 R2 为 0、1、2 时，能够使程序分别转向去执行与按键对应的功能。

解：题目属于一个多分支程序结构，应采用 JMP 指令来实现，程序清单如下：

```
        MOV A,R2          ;取控制变量
        RL A              ;对 A 乘 2 修正，因为 AJMP 为 2B 指令
        MOV DPTR,#TABLE
        JMP @A+DPTR
TABLE:  AJMP ROUT0
        AJMP ROUT1
        AJMP ROUT2
```

当 R2=0 时，程序将转到 ROUT0 处执行；当 R2=1 时，程序将转到 ROUT1 处执行；当 R2=2 时，程序将转到 ROUT2 处执行。

3.6.2 条件转移指令

1. 累加器判零转移指令

```
JZ rel      ;若(A)=0，转移；否则顺序执行本指令的下一条指令
JNZ rel     ;若(A) 不等于 0，转移；否则顺序执行本指令的下一条指令
```

这两条指令以累加器 A 的内容是否为 0 作为转移的条件。指令的执行结果不影响任何一个操作数的内容，也不影响任何标志位。

2. 比较不相等转移指令

```
CJNE A,#data, rel       ;若(A)=data，顺序执行；否则转移
                        ;如果(A)>data，C=0；否则 C=1
CJNE A,#direct, rel     ;若(A)=(direct)，顺序执行；否则转移
CJNE Rn,#data,rel       ;若(Rn)=data，顺序执行；否则转移
CJNE @Ri,#data,rel      ;若((Ri))=data，顺序执行；否则转移
```

这组指令的功能是比较两个操作数（无符号数）是否相等，若两数不相等则转移；若相等则顺序执行下一条指令。如果第一操作数大于或等于第二操作数，则(Cy)=0；反之，若第一操作数小于第二操作数，则(Cy)=1。指令的执行结果不影响任何一个操作数的内容。

3. 减 1 不为 0 转移指令

```
DJNZ Rn,rel             ;Rn←(Rn)-1
                        ;若(Rn)≠0，则转移，继续循环
                        ;若(Rn)=0，则结束循环，程序往下执行
```

```
DJNZ direct,rel          ;direct←(direct)-1
                         ;若(direct)≠0，则转移，继续循环
                         ;若(direct)=0，则结束循环，程序往下执行
```

这组指令对控制已知循环次数的循环过程十分有用：指定任何一个工作寄存器 Rn 或 RAM 单元 direct 为循环变量，对循环变量赋予初值以后，每完成一次循环，循环变量自动减 1，直到循环变量减为 0 时循环结束。

【例 3-18】编写程序，将内部 RAM 从 20H 开始的 20 个数据传送到外部 RAM 从 2000H 开始的对应单元中。

解：要实现题目要求，一种方法是使用顺序程序结构，但比较烦琐；另一种方法是采用循环程序结构，程序清单如下：

```
        MOV R1,#20H          ;取源数据首地址
        MOV DPTR,#2000H      ;取目标单元首地址
        MOV R7,#20           ;置循环变量
LOOP:   MOV A,@R1            ;取数据
        MOVX @DPTR,A         ;送数据到目标单元
        INC R1               ;修改地址，指向下一个源数据单元
        INC DPTR             ;修改地址，指向下一个目标单元
        DJNZ R7,LOOP         ;修改循环变量，并判断数据是否处理完
        SJMP $               ;程序踏步
```

3.6.3　子程序调用与返回指令

子程序是程序设计中常用的一种结构，为了缩短程序、节省存储空间，常常把逻辑上相对独立或具有通用意义的某段程序编写成子程序，当某个程序需要引用该子程序时，只需要通过子程序调用指令即可转向子程序，而当子程序执行完毕，又需要通过子程序返回指令把程序的执行流程引导到调用程序中继续执行。

为了实现子程序的调用与返回，MCS-51 提供了两条调用指令和两条返回指令。

（1）短调用指令：

ACALL addr11

（2）长调用指令：

LCALL addr16

（3）返回指令：

RET

（4）中断返回指令：

RETI

第一条指令是短调用指令，实际书写程序时常这样写：

ACALL SUB

即用标号的形式来指出调用的位置。

第二条指令是长调用指令，实际书写程序时常这样写：

LCALL SUB

同样用标号的形式来指出调用的位置。

RET 指令和 RETI 指令为子程序的最后一条指令。

3.6.4 空操作指令

空操作指令是唯一的一条不使 CPU 产生任何操作的控制指令，其格式为：

```
NOP                    ;PC←(PC)+1
```

NOP 指令的功能是使程序计数器 PC 加 1，在执行时间上消耗 12 个时钟周期，因此常用 NOP 指令实现等待或延时。

1．控制已知循环次数的循环过程通常用什么指令完成？

2．SJMP、AJMP 和 LJMP 指令在功能上有什么不同？

1．试编写程序，将 R0 中的低 4 位数与 R1 中的高 4 位数合并成一个 8 位数，并存放在 R0 中。

2．若单片机的主频为 12MHz，试用循环转移指令编写延时 20ms 的延时子程序，并说明这种软件延时方式的优缺点。

3.7 位操作类指令

位操作又称为布尔操作，它是以位为单位进行的各种操作。在布尔处理机中，借用进位标志 Cy 来存放逻辑运算结果，大部分操作都涉及 Cy，因此它相当于布尔处理机的“累加器”，称为“位累加器”，用符号 C 表示。

3.7.1 位传送指令

```
MOV C,bit              ; Cy←(bit)
MOV bit,C              ; bit←(Cy)
```

这组指令其中的一个操作数必须是进位标志 C，另一个可以是位地址（用 bit 表示）。

例如：

```
MOV C,ACC.7            ;将 ACC 中的最高位送给 Cy
```

再如，设(Pl)=00111101B，(P3)=11000101B，执行下面两条指令：

```
MOV C,P3.3
MOV Pl.2,C
```

则结果为：(P3)=11000101B，状态未变；(Cy)=0，(Pl)=00111001B。

3.7.2　位置位和位清零指令

SETB C　　;Cy←1
SETB bit　　;bit←1
这两条指令可以实现地址单元与位累加器的置位。
CLR C　　;CY←0
CLR bit　　;bit←0
这两条指令可以实现地址单元与位累加器的清零。
例如：
SETB P1.0　　;可使 P1.0 置 1
CLR P1.0　　;可使 P1.0 清零

3.7.3　位逻辑运算指令

位逻辑运算指令包括“与”、“或”、“非”3 种，下面介绍它们的指令格式。

1. 位逻辑“与”指令

ANL C,bit　　;CY←(CY)∧(bit)
ANL C,/bit　　;CY←(CY)∧(bit)取反

这两条指令是将 CY 标志位中的内容与位地址单元的内容或其内容的反码进行逻辑与操作，并将相与的结果再送入到 CY 标志位中。

2. 位逻辑“或”指令

ORL C, bit　　;CY←(CY)∨(bit)
ORL C, /bit　　;CY←(CY)∨(bit)取反

这两条指令是将 CY 标志中的内容和直接位地址中的内容或其内容的反码进行逻辑或操作，并将相或的结果送入到 CY 标志位中。

3. 位逻辑“非”指令

CPL C　　;CY←(CY)取反
CPL bit　　;bit←(bit)取反

这两条指令是将 CY 标志位中的内容或直接位地址中的内容取反，再放回原单元中去。

3.7.4　位条件转移指令

1. 位累加器 Cy 状态判断转移指令

JC rel　　;若(Cy)=1，转移
JNC rel　　;若(Cy)=0，转移

这组指令通常与 CJNE 指令一起使用，可以比较出两个数的大小，从而形成大于、小于、等于 3 个分支。

2. 位状态判断转移指令

JNB bit,rel　　;若(bit)=0，转移
JBC bit,rel　　;若(bit)=1，转移，且 bit←0

JB bit,rel ;若(bit)=1，转移

这几条指令可以实现位测试条件转移。其中 JBC 指令除了判断位地址的内容外，还将被测位清零。

为什么说布尔处理功能是 8051 单片机的重要特点？

1．试用位操作指令实现下列逻辑操作，要求不得改变未涉及的位的内容。

（1）使 ACC.0 置位。

（2）清除累加器高 4 位。

（3）清除 ACC.0、ACC.1、ACC.2、ACC.3。

2．若(CY)=1，(P1)=10100011B，(P3)=01101100B，试指出执行下列程序段后，CY、P1 口及 P3 口内容的变化情况。

```
MOV P1.3,C
MOV P1.4,C
MOV C,P1.6
MOV P3.6,C
MOV C,P1.0
MOV P3.4,C
```

3.8 程序设计实例

指令系统是熟悉单片机功能、合理应用单片机的基础。掌握单片机指令的关键在于多看多练，多上机练习，然后在现有程序的基础上进行模仿性编程。

汇编语言程序有 4 种结构形式，即顺序结构、分支结构、循环结构和子程序结构。下面举例说明这 4 种程序结构及编程方法。

3.8.1 顺序结构程序设计

案例 4 拆字程序

把 8000H 地址上的内容拆开，高位送 8001H 地址的低位，低位送 8002H 地址的低位，8001H、8002H 地址的高位清零。本程序通常在把数据送显示缓冲区时使用。

程序流程图如图 3-1 所示，程序清单如下：

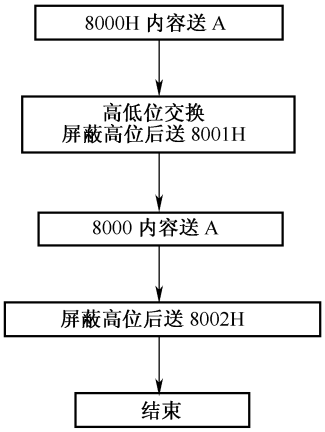

图 3-1　程序流程图

```
        ORG 0000H
        MOV DPTR,#8000H          ;指定的字节
        MOVX A,@DPTR
        MOV B,A                  ;暂存
        SWAP A                   ;交换
        ANL A,#0FH               ;屏敝高位
        INC DPTR
        MOVX @DPTR,A
        INC DPTR
        MOV A,B
        ANL A,#0FH               ;指定字节的内容屏敝高位
        MOVX @DPTR,A
LOOP:   SJMP LOOP
        END
```

案例 5　拼字程序

把 8000H、8001H 两个字节的低位分别送入 8002H 的高位和低位。本程序一般用于把显示缓冲区中的数据取出拼装成一个字节。

程序流程图如图 3-2 所示，程序清单如下：

```
        ORG   0000H
        MOV DPTR,#8000H
        MOVX A,@DPTR
        ANL A,#0FH               ;屏敝高位
        SWAP A
```

```
        MOV B,A                 ;保存
        INC DPTR
        MOVX A,@DPTR
        ANL A,#0FH
        ORL A,B                 ;合并
        INC DPTR
        MOVX @DPTR,A            ;送 8002H 存放
LOOP:   SJMP LOOP
        END
```

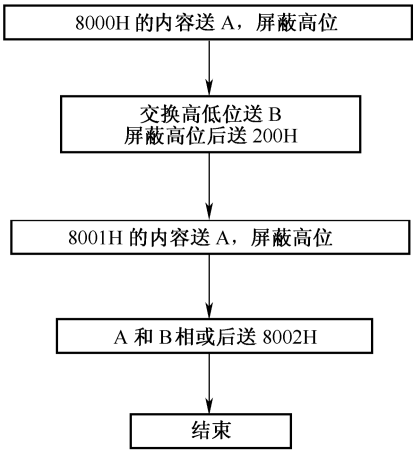

图 3-2　程序流程图

案例 6　查表程序

求 R1 中的数（0～15 之间）平方，结果仍放回到 R1 中。

查表是计算机解决复杂问题所采取的一种简单而有效的方法。它根据变量 x 的值，在表格中查找 y，使 y=f(x)。由于查表程序结构简单，执行速度快，因而广泛应用于数值计算、转换、补偿、显示、打印等功能程序中。

设计查表程序的关键是善于组织表格，使其具有一定的规律性，便于编程查找。程序清单（采用 DPTR 当基址寄存器）如下：

```
        ORG 0100H
TAB2:   PUSH DPH                ;保存 DPTR 的原值
        PUSH DPL
        MOV DPTR, #TAB          ;取平方表首地址
        MOV A, R1
        MOVC A, @A+DPTR         ;查平方表
        MOV R1, A
        POP DPL                 ;恢复 DPTR 的原值
        POP DPH
        RET
```

```
TAB:  DB 0,1,4,9,16,25,39          ;平方表
      DB 49,64,81,100,121          ;续表
      DB 144,169,196,225           ;续表
```

为了保护调用程序中 DPTR 的值，在子程序中可以用堆栈进行保护，子程序返回之前应正确恢复。

3.8.2　分支结构程序设计

程序分支是通过条件转移指令实现的，即根据条件对程序的执行进行判断，满足条件则进行程序转移，不满足条件则顺序执行程序。

案例 7　无符号数的比较（单分支结构）

比较外部 RAM 的 Data1 和 Data2 单元的两个无符号数，并将大数存入 Data3 单元（Data1、Data2、Data3 为 3 个连续的单元）。

解：该问题属于单分支结构，程序流程图如图 3-3 所示，程序清单如下：

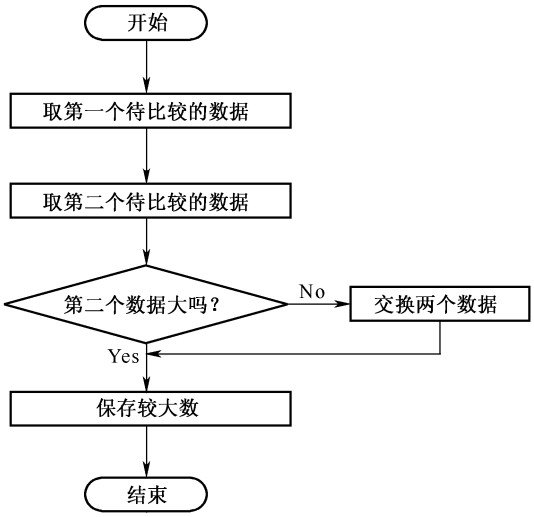

图 3-3　无符号数的比较

```
      ORG 1000H
COM1: MOV DPTR,#DATA1        ;设置外部数据存储器指针
      MOV A,@DPTR            ;取第一个数
      MOV B,A                ;暂存第一个数到 B
      INC DPTR               ;修改指针，指向第二个数所在的单元
      MOVX A,@DPTR           ;取第二个数到 A
      CJNE A,B,L1            ;两数比较，产生 Cy 标志
L1:   JC BIG1                ;若第二个数小于第一个数，则转向 BIG1；修改
```

```
                                ;指针，指向 Data3 单元
BIG:    INC DPTR
        MOVX @DPTR,A            ;存大数
        RET
BIG1:   XCH A,B                 ;将大数转移到 A
        SJMP BIG
        END
```

案例 8 码制转换（双分支结构程序）

将 R2 中的一位十六进制数转换为 ASCII 码，结果仍放在 R2 中。

解：查 ASCII 码表可知数字 0～9 的 ASCII 码分别是 30H～39H；英文大写字母 A～F 的 ASCII 码分别是 41H～46H。可见该十六进制数若＜10，要转换为 ASCII 码应加 30H；若≥10，则加 37H。

该问题属于双分支结构，程序流程图如图 3-4 所示，程序清单如下：

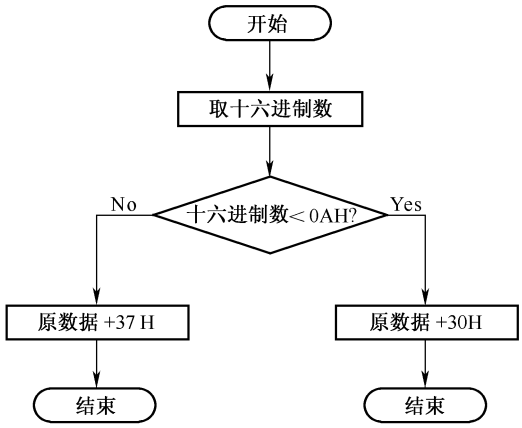

图 3-4 十六进制数转换为 ASCII 码

```
        ORG 0000H
HEXASC: MOV A, R2               ;将该十六进制数暂存于 A 中
        CJNE A, #0AH, L1
L1:     JNC ADD_37              ;判断是否小于 0AH
ADD_30: ADD A, #30H             ;小于 0AH，则加 30H
        MOV R2, A               ;保存结果
        RET                     ;子程序返回
ADD_37: ADD A, #37H             ;大于或等于 0AH，则加 37H
        MOV R2, A               ;保存结果
        RET
```

该程序使用了两个子程序返回指令，这也是双分支程序的一个特点。

案例 9 带符号数的比较（多分支结构程序）

比较内部 RAM 的 Data1 和 Data2 单元内以补码形式表示的两个带符号数，并将大数存入

BIG 单元，小数存入 SMALL 单元；若两数相等，则建立起标志位 F0。

解：要比较两个带符号数，不能依据 Cy 标志位判定其大小，其判定方法为：

（1）若 X、Y 两数符号相同，且(X-Y)为正，表明 X>Y。

（2）若 X、Y 两数符号不同，则根据符号判定大小。

程序流程图如图 3-5 所示，程序清单如下：

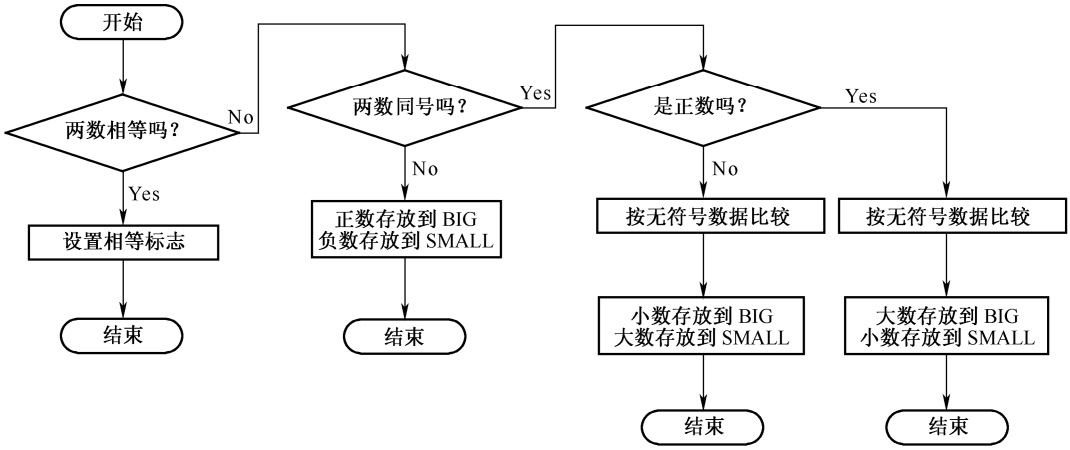

图 3-5　比较两个带符号数

```
        DATA1 EQU 40H
        DATA2 EQU 41H
        BIG EQU 30H
        SMALL EQU 31H
        ORG 0000H
COM2:   MOV A, 40H
        XRL A, 41H
        JNZ STEP1               ;两数不等，转 STEP1
        SETB F0                 ;两数相等，F0 置位
        RET
STEP1:  JB ACC.7, TEST          ;两数异号，转 TEST
        XRL A, 41H              ;恢复 40H
        SUBB A, 41H             ;比较
        JB ACC.7, STEP3         ;40H 小，转 STEP3
STEP2:  MOV BIG, 40H
        MOV SMALL, 41H
        RET
TEST:   XRL A, 41H
        JNB ACC.7, STEP2        ;40H 为正，转 STEP2
STEP3:  MOV SMALL, 40H
```

```
        MOV BIG, 41H
        RET
```

该程序为多分支结构程序。

3.8.3 循环结构程序设计

当需要反复执行某段程序时，采用循环程序最为适宜。它可以简化程序的编制，使程序结构清晰，并大大缩短程序所占用的内存单元。循环又分单重循环、双重循环和多重循环。

案例 10 计算 $\sum_{x=1}^{10} X$ 并将结果保存在内部 RAM 的 Y 单元中

解：程序清单如下：

```
        Y EQU 20H
        ORG 1000H
        CLR A                   ;清部分和
        MOV R0,#01              ;置循环初值
NEXT:   CJNE R0,#10,AD          ;判断是否完成
        MOV Y,A                 ;保存结果
        RET                     ;结束
AD:     ADD A,R0                ;计算部分和
        INC R0                  ;循环修改
        SJMP NEXT               ;继续累加下一个数据
```

该程序属于“先判断后执行”结构的单循环程序。

案例 11 求最大数

设有一组数存放在内部 RAM 从 Data 开始的连续单元中，这组数的长度存放在 Data1 单元中，试编写程序找出将其中的最大数，并存入 Data2 单元。

解：根据题意，可先假设第一个数为最大数并放入 DATA2 单元，然后逐个取出其他数据，并与 DATA2 单元中的数据进行比较，若当前数大于 DATA2 单元中的数据，则将当前数存入 DATA2 单元；否则继续比较下一个数，直至比较结束。

程序清单如下：

```
        Data EQU 42H
        Data1 EQU 41H
        Data2 EQU 40H
        ORG 0000H
        MOV R0, #Data           ;设置地址指针，并指向存放数据的起始单元
        MOV Data2, @R0          ;将第一个数当作当前最大数
        DEC Data1               ;修改剩余的数据个数
```

```
NEXT:  INC R0                ;指向下一个数据单元
       MOV A, @R0            ;取当前数据
       CJNE A, Data2, COMP   ;比较当前数与当前最大数
COMP:  JC CON                ;若当前数小于当前最大数，则继续
       MOV Data2, A          ;否则，把当前数当作最大数
CON:   DJNZ Data1, NEXT      ;判断数据是否比较完毕
       RET                   ;比较结束
```

该程序属于“先执行后判断”结构的单循环程序。

3.8.4　子程序设计

对于一般的单片机应用系统，通常采用模块化程序设计方法来设计应用程序。所谓模块化程序设计，就是把一个复杂的程序分成多个功能上相对独立的程序模块（即子程序），分别编制、调试，然后连接在一起形成一个完整的程序。为此，对那些具有独立功能的或通用的或需要多次重复使用的程序段，常编写成子程序，需要执行这段程序时，就用 ACALL 或 LCALL 指令调用该子程序，子程序执行完毕再由返回指令 RET 返回到调用程序继续执行。子程序调用与返回过程如图 3-6 所示。

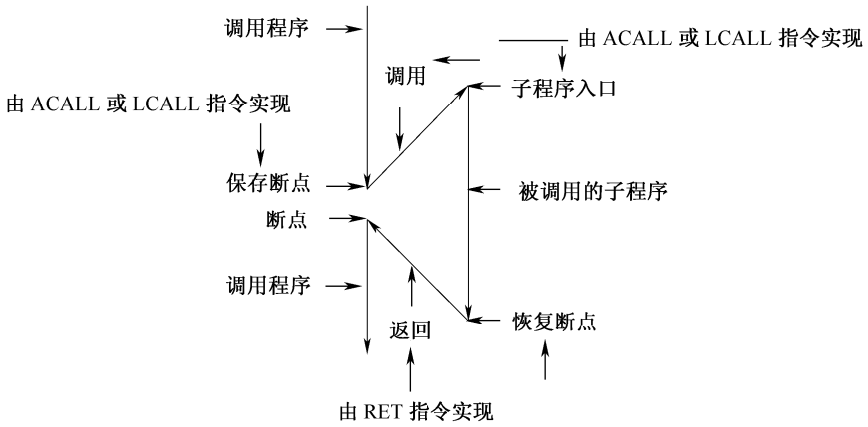

图 3-6　子程序的调用与返回

案例 12　一位数码管显示

P1 接七段数码管段数据口，P3.4 接七段数码管位数据口，编写程序，使一位数码管 0～9 循环点亮。数码管为共阴极数码管。

程序流程图如图 3-7 所示，程序清单如下：

```
       ORG 0000H
START: CLR P3.4
       MOV P1,#3FH           ;显示“0”
```

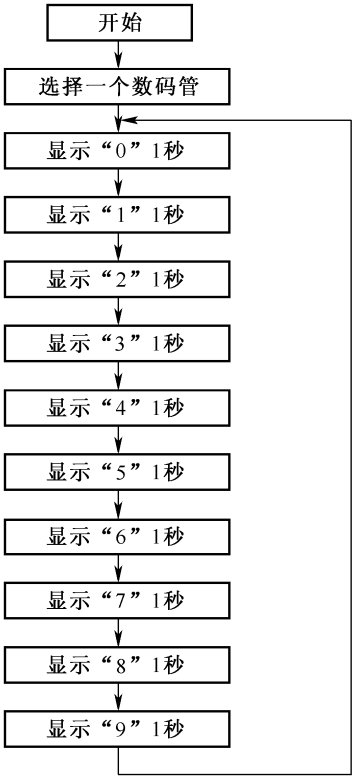

图 3-7

```
LCALL DELAY
MOV P1,#06H          ;显示“1”
LCALL DELAY
MOV P1,#5BH          ;显示“2”
LCALL DELAY
MOV P1,#4FH          ;显示“3”
LCALL DELAY
MOV P1,#66H          ;显示“4”
LCALL DELAY
MOV P1,#6DH          ;显示“5”
LCALL DELAY
MOV P1,#7DH          ;显示“6”
LCALL DELAY
MOV P1,#07H          ;显示“7”
LCALL DELAY
MOV P1,#7FH          ;显示“8”
LCALL DELAY
MOV P1,#67H          ;显示“9”
```

```
        LCALL DELAY
        LJMP START
DELAY:                          ;延时子程序
        MOV R4,#10
D1:     MOV R5,#200
D2:     MOV R6,#126
D3:     DJNZ R6,D3
        DJNZ R5,D2
        DJNZ R4,D1
        RET
        END
```

现场研讨

1．常用的程序结构有哪几种？特点如何？

2．什么是伪指令？常用的伪指令功能如何？

动手演练

1．求 8 个无符号数的平均值，这 8 个无符号数存放在内部 RAM 以 20H 开始的 8 个单元中，结果保存在 30H 中。

2．设内部 RAM 从 21H 单元开始存有一组带符号数，数据长度存于 20H 单元，要求把该组数据中的正数、负数分别求和，并存放在寄存器 R6 和 R7 中。

3．设内部 RAM 从 LIST 单元开始存有一组无符号数据，数据个数为 30，编程找出其中的最大数，并存入 BIG 单元中。

4．设内部 RAM 从 20H 单元开始存有 40H 个无符号数，试编制程序将它们按从大到小的顺序排列，结果仍存放在原存储区域内。

5．试编写程序，将内部 RAM 的 20H、21H 单元的两个无符号数相乘，结果存放在 R2、R3 中，R2 中存放高 8 位，R3 中存放低 8 位。

6．已知自变量 X 存放在 4000H 单元中，函数值 Y 存放在 4001H 单元中，编写程序计算分支函数的值。

$$Y=\begin{cases}5X & ;X>10H\\ 4X+50H & ;X=10H\\ 3X-30H & ;X<10H\end{cases}$$

7．编写排序程序，将地址为 30H～3FH 的片内数据存储器中的数据降序排序。

8．设被加数存放在内部 RAM 的 20H、21H 单元，加数存放在 22H、23H 单元，若要求和存放在 24H、25H 中，试编写出 16 位数相加的程序。

模块总结

指令系统功能的强弱决定了计算机性能的高低。80C51 单片机的指令系统共 111 条指令，其指令的执行时间短、字节少，位操作指令极为丰富。

指令由操作码和操作数组成。寻址方式是寻找存放操作数的地址并将其提取出来的方法。80C51 单片机有 7 种基本的寻址方式。

本模块以举例方式，对于初学者经常用到的各类指令和简单的程序类型进行了较详细的注释。数据传送类程序的编写是初学者首先应掌握的，而数据传送空间是内部 RAM、寄存器及外部数据空间及程序空间。因此，内部硬件结构及存储器结构应是初学者首先应该掌握的。只有软件指令功能和硬件映像相结合，才能更好地掌握汇编源程序的编写。初学者可以结合本模块中的例题和习题进行程序编写的练习。