第3章 MCS-51单片机的指令系统及程序设计

CHAPTER 3

指令是程序的基本单元，它是控制、指挥 CPU 的命令。一台计算机所有指令的集合称为指令系统。指令是由计算机的硬件特性决定的，不同类型的计算机的指令系统是不兼容的。指令系统展示了计算机的操作功能，它是表征计算机性能的一个重要指标。MCS-51 单片机指令系统共有 111 条指令，提供了多种灵活的寻址方式，指令代码短、功能强、执行快；另外，它提供了位操作指令，允许直接对位进行逻辑和传送操作。本章将介绍 MCS-51 单片机指令系统、寻址方式、指令的功能和使用方法。

3.1 指令格式

指令(Instruction)是人们给计算机的命令，是芯片制造厂家提供给用户使用的软件资源。一台计算机所有指令的集合称为指令系统。由于计算机只能识别二进制数和二进制编码，而对于用户来说，二进制编码可读性差，难以记忆和理解，因此，一条指令有两种表示方式：一种是计算机能够识别的机器码(二进制编码的指令代码)——机器语言(Machine Language)，另一种是采用人们容易理解和记忆的助记符(Mnemonics)形式——汇编语言(Assembly Language)。汇编语言便于用户编写、阅读和识别，但不能直接被计算机识别和理解，必须汇编成机器语言才能被计算机执行。汇编语言可以汇编为机器语言，机器语言也可以反汇编为汇编语言，它们之间一一对应。汇编和反汇编可以由编译系统自动完成，也可以由用户通过人工查表的方法手工完成。在本章主要介绍指令的汇编语言形式。

MCS-51 单片机的指令由标号、操作码助记符、操作数和注释 4 部分组成，格式如下。

[标号：]　操作码助记符 [操作数] ;[注释]

(1) 标号：表示该指令代码的第一字节所在单元地址。标号由用户自行定义，必须是英文字母开头。在汇编语言程序中，标号可有可无。程序由编译系统汇编或手工汇编时，把标号替换成该指令代码的第一字节所在单元地址。

(2) 操作码助记符：规定指令所执行的操作，描述指令的功能。在指令中不可缺少。

(3) 操作数：参与操作的数据信息。

(4) 注释：用户对指令的操作说明，便于阅读和理解程序。注释部分可有可无。

编制程序时，一般标号后带冒号(:)，与操作码助记符之间应有空若干空格；操作码助记符和操作数用空格隔开；如果指令中包含多个操作数，操作数之间用逗号(,)隔开；注释

与指令之间采用分号(;)隔开,一般情况下,在程序中,分号(;)之后的一切信息均为说明注释部分,编译系统汇编时不予处理。

下面为一段汇编语言程序:

```
【标号】     【操作码助记符】     【操作数】        【注释】
START :      MOV                  A ,   #20H ;      把数 20H 送入累加器 A 中
             INC                  A ;               A.加一
```

MCS-51 单片机汇编语言指令有以下几种形式。

(1) 没有操作数,如: RET,RETI,NOP。

(2) 有一个操作数,如: INC A,DEC 20H,CLR C,SJMP NEXT。

(3) 有两个操作数,如: MOV R7, #DATA; ADD A, R0; DJNZ R2,LOOP。

(4) 有三个操作数,如: CJNE A, #20H, NEQ。

从机器语言的指令代码长度来看,MCS-51 单片机汇编语言指令有以下 3 种形式。

(1) 单字节指令: 指令机器代码为一字节,占用一个单元。如:

```
INC DPTR          (指令机器代码: A3)
ADD A, R7         (指令机器代码: 2F)
```

(2) 双字节指令: 指令机器代码为两字节,占用两个单元。如:

```
SUBB A, 2BH       (指令机器代码: 95 2B)
ORL C, /27H       (指令机器代码: A0 27)
```

(3) 三字节指令: 指令机器代码为三字节,占用三个单元。如:

```
MOV 20H, #00H     (指令机器代码: 75 20 00)
LJMP  2000H       (指令机器代码: 02 20 00)
```

3.2　MCS-51 单片机的寻址方式

所谓寻址方式(Addressing Mode)就是 CPU 执行指令时获取操作数的方式。寻址方式的多少是反映指令系统优劣的主要指标之一。寻址方式隐含在指令代码中,寻址方式越多,灵活性越大,指令系统越复杂。MCS-51 单片机提供了 7 种不同的寻址方式: 立即寻址、直接寻址、寄存器寻址、寄存器间接寻址、变址寻址、位寻址和相对寻址。

1. 立即寻址方式

立即寻址方式也称为立即数(Immediate Constants)寻址。立即寻址方式是在指令中直接给出了参与运算的操作数,CPU 直接从指令中获取操作数。这种由指令直接提供的操作数叫立即数,它是一个常数。指令中操作数前面加有“#”号,它的作用是告知汇编系统,其后是一个常数。例如,如图 3.1 所示,“MOV A, #20H”表示把立即数 20H 送入累加器 A 中。

2. 直接寻址方式

直接寻址方式(Direct Addressing)是在指令中给出了参与运算的操作数所在单元的地址或所在位的位地址,操作数存储在指定的单元或位中。例如,“MOV A,20H”表示把 20H 单元的内容送到累加器 A 中,如图 3.2 所示。

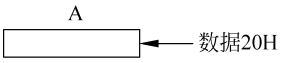


图 3.1　立即寻址方式

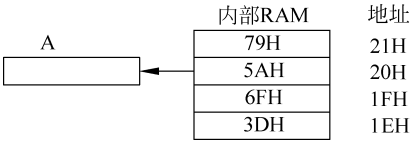


图 3.2　直接寻址方式

直接寻址方式可以访问以下 3 种地址空间。

(1) 内部 RAM：00～7FH。

(2) 21 个特殊功能寄存器，对这些特殊功能寄存器，CPU 只能采用直接寻址方式。

(3) 位寻址空间。

3. 寄存器寻址方式

寄存器寻址(Register Addressing)方式是在指令中指出了参与运算的操作数所在的寄存器，操作数存储在寄存器中。例如，“MOV A，R0”表示把工作寄存器 R0 中的数送到累加器 A 中，如图 3.3 所示。

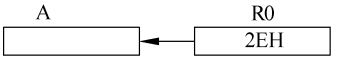


图 3.3　寄存器寻址方式

寄存器寻址方式中的寄存器位工作寄存器 R0～R7、DPTR、累加器 A、寄存器 B(仅在乘除法时)和布尔累加器 C。

4. 寄存器间接寻址方式

寄存器间接寻址(Indirect Addressing)方式是在指令中用地址寄存器指出存放操作数的单元地址。地址寄存器(Address Register)的内容是操作数所在单元的地址。操作数是通过指令中给出的地址寄存器内容间接得到的。在指令中，作为地址寄存器的寄存器只有 R0，R1，DPTR，表示为@R0，@R1，@DPTR。

例如，已知 PSW 的内容是 00H，寄存器 R0 的内容为 41H，指令“MOV A，@R0”是把地址寄存器 R0 的内容 41H 作为操作数所在单元地址，把该单元中的内容 5AH 送到累加器 A 中，如图 3.4 所示。

图 3.4　对内部 RAM 单元的寄存器间接寻址

例如，采用指令“MOVX A，@DPTR”把外部 RAM 的 4001H 单元的内容送到累加器 A 中，如图 3.5 所示，它是由 DPTR 的内容指出要访问的外部 RAM 单元地址 4001H。

图 3.5　对外部 RAM 单元的寄存器间接寻址

寄存器间接寻址方式的寻址范围如下。

(1) 内部 RAM：00H～7FH，由地址寄存器@R0 和@R1 指出操作数所在单元的地址。

（2）外部 RAM 和外部 I/O 口：0000H～FFFFH，由 16 位地址寄存器@DPTR 指出操作数所在单元或 I/O 口的地址，也可由 8 位地址寄存器@R0 和@R1 指出操作数所在单元的低 8 位地址，此时，高 8 位地址由 P2 口提供。

另外，还有一个隐含的 8 位地址寄存器 SP，用于与堆栈操作相关的指令，由 SP 内容间接指出操作数所在单元的地址。

5. 变址寻址方式

变址寻址（Indexed Addressing）方式也称为基址寄存器加变址寄存器间接寻址方式，存放操作数单元的地址为基址寄存器和变址寄存器二者内容之和。变址寻址方式中，操作数所在单元的地址以基址寄存器与变址寄存器内容之和的形式在指令中指出。这种寻址方式只适用于程序存储器。在 MCS-51 单片机中，只有两个 16 位寄存器 DPTR 和 PC 可以作为基址寄存器，而可作为变址寄存器的只有累加器 A。从程序存储器中读取操作数的指令有如下两种：

```
MOVC A , @A + DPTR;
MOVC A , @A + PC;
```

基址寄存器 PC 或 DPTR 与累加器 A 两者内容相加得到的 16 位地址作为操作数所在单元的地址，把该地址对应单元的内容取出送给累加器 A。

另外一种变址寻址方式的指令用于程序散转指令。在这种情况下，程序转移的目标地址（Destination Address）由基址寄存器 DPTR 与累加器 A 内容之和确定，把二者内容之和传送给程序计数器 PC，使程序执行的顺序发生改变，从而实现程序转移。指令如下：

```
JMP @A + DPTR;
```

6. 位寻址方式

位寻址（Bit Addressing）方式是在指令中指出了参与运算的操作数（一位）所在的位的位地址或位寄存器（仅有位累加器 C）。位寻址方式是 MCS-51 单片机特有的一种寻址方式。

在指令中位地址通常以下列几种形式之一表示。

（1）被操作位用直接位地址表示：

```
CLR  07H;     MOV  22H, C
```

（2）被操作位用该位在单元或特殊功能寄存器的相对位置表示，常用点操作符方式：

```
SETB  ACC.6 ;     ANL  C, 25H.5
```

（3）被操作位用特殊功能寄存器中规定的位名称表示：

```
CPL  RS0 ;     JBC  TF0, OVER;
```

位寻址方式的适用范围为 MCS-51 单片机的位寻址空间。

实际上，如果位寻址方式在指令中指出参与操作的位的位地址时，可以理解为直接寻址方式；而在指令中指出参与操作的位所在的位寄存器（位累加器 C）时，可以认为是寄存器寻址方式。

7. 相对寻址方式

相对寻址（Relative Addressing）方式是在指令中给出了程序转移的目标地址与当前地

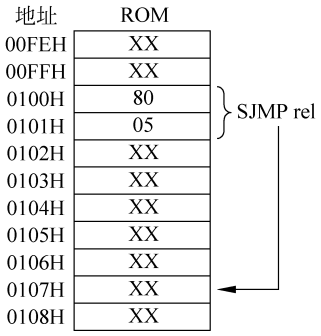


图 3.6　相对寻址方式

址之间的相对偏移量。它是为解决程序转移而专门设置的，用于控制转移类指令。相对偏移量为一字节的补码，在指令代码中用 rel 表示，取值为－128～＋127，相对偏移量 rel 大于 0 则程序向下转移，偏移量 rel 小于 0 则程序向上转移。程序转移的目标地址为当前地址与偏移量 rel 之和，该值送给程序计数器(PC)，则程序转移到目标地址处执行。

如指令“SJMP rel”，其对应的机器码为“80 rel”，如图 3.6 所示。当“SJMP rel”被 CPU 取走后，PC 指向 0102H 单元，假设 rel＝05H，CPU 执行该指令计算出的目标地址是

(PC)＋ rel＝0102H ＋05H＝0107H

则 CPU 把该地址赋给 PC，这样程序就转移到 0107H 处。

在程序设计时，通常在 SJMP 指令之后以标号或单元地址的形式给出目标地址，在程序汇编时，偏移量 rel 由汇编系统自动算出。下面一段程序是带有条件判别的程序，程序汇编后，被存储在程序存储器 0200H 开始的区域。在本程序中，如果进位 Cy 的状态为 1，则需要程序转移到 CARRY 处执行。程序中“JC　CARRY”的指令代码存储的起始地址为 0204H，指令代码中给出的偏移量为 03H，JC 指令代码为 2 字节，则 CPU 取走该指令后，(PC)＝0204H＋02H＝0206H；该指令执行后，CPU 计算出的目标地址是(PC)＝0206H＋03＝0209H。程序会转移到 0209H 单元执行，即标号 CARRY 处，与程序设计意图是一致的。

【标号】	【指令】	【注释】	【单元地址】	【指令代码】
MYPROG:	MOV A, 20H;	取 20H 单元内容到 A	0200H	E5　20
	ADD A, 30H;	两个单元内容相加	0202H	25　30
	JC　CARRY;	有进位时，转移	0204H	40　03
	MOV 20H, A;	存结果	0206H	F5　20
	RET;		0208H	22
CARRY:	SETB 28H.0;	标志位置 1	0209H	D2　40
	…		020BH	…

3.3　指令系统分析

3.3.1　指令的分类

MCS-51 指令系统中共有 111 条指令。

按指令代码的字节数划分，可分为以下 3 类。

(1) 单字节指令(49 条)。

(2) 双字节指令(45 条)。

(3) 三字节指令(17 条)。

按指令执行的时间划分，可分为以下 3 类。

(1) 单机器周期指令(64 条)。

(2) 双机器周期指令(45 条)。

(3) 4 机器周期指令(2 条)。

按指令功能进行划分,可分为以下 5 类。

(1) 数据传送类指令(29 条)。

(2) 算术运算类指令(24 条)。

(3) 逻辑运算类指令(24 条)。

(4) 控制转移类指令(17 条)。

(5) 位操作类指令(17 条)。

本章将按指令功能详细介绍 MCS-51 单片机的指令系统,主要以汇编语言指令为主,指令的机器码(指令代码)可查阅附录 A 的 MCS-51 单片机指令及其指令代码表。为了说明指令的功能,下面定义一些在汇编指令中用到的符号和字段。

- Rn:n=0～7,表示当前工作寄存器的 8 个工作寄存器 R0～R7。
- @Ri:i=0,1;表示作为地址寄存器的工作寄存器 R0 和 R1。
- direct:表示一个单元地址,8 位二进制数,取值范围为 00H～FFH。它可表示内部 RAM 的单元地址(00H～7FH)或特殊功能寄存器(SFR)的地址(80H～FFH)。
- data:表示一个 8 位二进制常数,取值范围为 00H～FFH。
- data16:表示一个 16 位二进制常数,取值范围为 0000H～FFFFH。
- addr16:表示一个 16 位单元地址,取值范围为 0000H～FFFFH。
- addr11:表示一个 11 位单元地址,取值范围为 000 0000 0000H～111 1111 1111B。
- bit:表示一个位地址,8 位二进制数,取值范围为 00H～FFH,bit 可表示内部 RAM(20H～2FH)或是特殊功能寄存器(SFR)中的可寻址位的位地址。
- rel:表示偏移量,8 位带符号二进制数,补码,取值范围为－128～＋127。
- (direct):表示由地址 direct 指定的寄存器或单元的内容。
- [(AddReg)]:表示由地址寄存器 AddReg 内容所指定的存储单元的内容。
- LABEL:程序中指定的标号。

在 MCS-51 单片机系统中,除了 16 位寄存器(PC 和 DPTR)和布尔累加器 C(位处理器)之外,单元或寄存器内容为 8 位二进制数,数值范围为 00H～FFH,不管它属于内部 RAM、外部 RAM、特殊功能寄存器,还是程序存储器。存储在位寻址空间上的信息称为状态,一位的状态有两种取值:0 和 1。另外,由于在指令中常采用十六进制数表示单元地址或常数,若以 A～F 打头表示一个十六进制数时,在指令中采用前面加 0 的方式以区别于字符,如 0AAH。

3.3.2　数据传送指令

数据传送(Data Transfers)类指令共有 29 条,是 MCS-51 单片机指令系统中种类最多、程序中使用最频繁的一类指令。数据传送类指令分为以下 5 种类型。

(1) 通用传送指令。

(2) 堆栈操作指令。

(3) 交换指令。

(4) 访问程序存储器的数据传送指令。

(5) 访问外部 RAM 和外部 I/O 口的数据传送指令。

1. 通用传送指令

通用传送指令的助记符为 MOV，指令的一般形式为：

```
MOV  目的操作数,源操作数
```

指令的功能是把源操作数送给目的操作数，源操作数保持不变。除非是 PSW 作为目的操作数，执行指令一般不影响标志位 Cy，AC，OV；但累加器 A 作为目的操作数时，将会对奇偶校验位 P 产生影响。这类指令操作在单片机内部 RAM 或特殊功能寄存器区。下面分别以累加器 A、工作寄存器 R0～R7、某一个单元为目的操作数，介绍这类指令的功能。

1）以累加器 A 为目的操作数的传送指令

```
MOV  A,源操作数;
```

源操作数复制给累加器 A，表示为：源操作数→(A)，“→”表示数据的传递方向。有以下 4 种形式：

```
MOV  A,Rn              ;(Rn)→(A),n=0～7
MOV  A,direct          ;(direct)→(A)
MOV  A,@Ri             ;[(Ri)]→(A),i=0,1
MOV  A,#data           ;data→(A)
```

如：

```
MOV  A,R2              ;(R2)→(A),把寄存器 R2 的内容复制给累加器 A.
MOV  A,30H             ;(30H)→(A),把内部 RAM 地址为 30H 单元的内容复制给累加器 A.
MOV  A,@R0             ;[(R0)]→(A),把地址寄存器 R0 的内容指定的内部 RAM 单元内容复制
                       ;给累加器 A.也就是把内部 RAM 的一个单元的内容送给累加器 A,该单
                       ;元的地址是由地址寄存器 R0 的内容指定的
MOV  A,#0A6H           ;把常数 0A6H 存放在累加器 A 中
```

2）以工作寄存器 Rn 为目的操作数的传送指令

```
MOV   Rn,源操作数;
```

源操作数送给工作寄存器 Rn，n＝0～7，有 3 种形式：

```
MOV  Rn,A              ;(A)→(Rn),n=0～7
MOV  Rn,direct         ;(direct)→(Rn),n=0～7
MOV  Rn,#data          ;data→(Rn),n=0～7
```

如：

```
MOV  R0,A              ;(A)→(R0),把累加器 A 的内容送到寄存器 R0 中
MOV  R3,30H            ;(30H)→(R3),把内部 RAM 的 30H 单元的内容复制给寄存器 R3
MOV  R7,#36H           ;36H→(R7),把常数 36H 存放在寄存器 R7 中
MOV  R1,#30            ;30→(R1),把十进制数 30 送到 R1 中,(R1)=1EH
MOV  R6,#01101100B     ;把二进制数 01101100B 送到 R6 中,(R6)=6CH
```

3）以直接地址为目的操作数的传送指令

```
MOV   direct,源操作数;
```

把源操作数送到指定单元 direct 中，这是通用传送指令中操作形式最为丰富的一组指令，支持任意两个单元之间的数据传送。这组指令有 5 种形式：

```
MOV  direct,A          ;(A)→(direct)
MOV  direct,Rn         ;(Rn)→(direct),n=0～7
```

```
MOV  direct,direct1     ;(direct1)→(direct)
MOV  direct,@Ri         ;[(Ri)]→(direct),i=0,1
MOV  direct,#data       ;data→(direct)
```

如：

```
MOV  30H,A              ;(A)→(30H),把累加器 A 的内容复制给内部 RAM 为 30H 的单元
MOV  P1,R2              ;(R2)→(P1),把 R2 的内容从 P1 口输出
MOV  38H,60H            ;(60H)→(38H),把 60H 单元的内容复制给 38H 单元
MOV  TL0,@R1            ;[(R1)]→(TL0),把 R1 内容指定单元的内容送到计数器 TL0 中
MOV  58H,#36H           ;36H→(58H),把常数 36H 写入内部 RAM 的 58H 单元
MOV  PSW,#00011000B     ;把 00011000B 写入 PSW
```

需要指出的是，CPU 访问 SFR 只能采取直接寻址方式，因此，上述指令中对 P1 和 TL0 的访问可以写成下列形式：

```
MOV  P1,R2 → MOV 90H, R2
MOV  TL0,@R1 → MOV 8CH, @R1
MOV  PSW,#00011000B → MOV 0D0H, #00011000B
```

不管哪种形式，汇编时它们的指令代码是相同的。对于编程者来说，第一种使用 SFR 名称的形式更方便。

4）以间接地址为目的操作数的传送指令

```
MOV  @Ri,源操作数;
```

这是一组以某一个单元为目的操作数传送指令，与上一组指令不同的是，单元地址不是直接给出的，而是由地址寄存器 R0 或 R1 的内容间接给出的，有以下 3 种形式：

```
MOV  @Ri,A              ;(A)→[(Ri)],i=0,1
MOV  @Ri,direct         ;(direct)→[(Ri)],i=0,1
MOV  @Ri,#data          ;data→[(Ri)],i=0,1
```

如：

```
MOV  @R0,A              ;(A)→[(R0)],把 A 的内容送给地址寄存器 R0 内容指定的单元
MOV  @R1,36H            ;(36H)→[(R1)],把 36H 单元的内容送给另一个单元,该单元
                        ;的地址由地址寄存器 R1 内容指定
MOV  @R0,SBUF           ;(SBUF)→[(R0)],把串行口接收缓冲器 SBUF 的内容送给地址寄存
                        ;器 R0 内容指定的单元
MOV  @R0,#0D6H          ;给地址寄存器 R0 内容指定的单元设置常数 0D6H
```

例 3.1　在内部 RAM 中，30H 单元的内容为 40H，40H 单元的内容为 10H，当前从 P1 口输入数据 11001010B，分析下列程序的执行结果：

```
MOV  R0, #30H;
MOV  A,  @R0;
MOV  R1,  A;
MOV  B,  @R1;
MOV  @R1, P1;
MOV  P2,  P1;
```

分析如下：

```
MOV  R0, #30H           ;30H→(R0),(R0)=30H
MOV  A,  @R0            ;[(R0)]→(A),即[30H]→(A),(A)=40H
MOV  R1,  A             ;(A)→(R1),(R1)=40H
```

```
MOV  B,  @R1          ;[(R1)]→(B),即[40H]→(B),(B) = 10H
MOV  @R1, P1          ;(P1)→[(R1)],即(P1)→[40H],(40H ) = 11001010B = 0CAH
MOV  P2,  P1          ;(P2) = 0CAH
```

执行结果：(30H)＝40H,(R0)＝30H,(R1)＝40H,(A)＝40H,(B)＝10H,(40H)＝0CAH,(P2)＝0CAH。

例 3.2 设计程序把单片机 P1 口引脚当前的状态从 P3 口输出。

```
MOV  P1, #0FFH        ;P1 口全写 1,置 P1 口为输入
MOV  A,P1             ;读取 P1 口引脚的状态
MOV  P3,A             ;从 P3 口输出
```

也可用下面的程序实现：

```
MOV  P1, #0FFH        ;P1 口全写 1,置 P1 口为输入
MOV  P3,P1            ;读取 P1 口引脚的状态,并从 P3 口输出
```

5）十六位数据传送指令

```
MOV  DPTR,#data16     ;data8～15→(DPH),data0～7→(DPL).
```

这是 MCS-51 单片机指令系统中唯一的一条设置 16 位二进制常数的指令。该指令操作等同于分别对 DPH 和 DPL 的操作：

```
MOV  DPH,#data8～15
MOV  DPL,#data0～7
```

两种实现方法的区别在于：前者为三字节指令，指令周期为两个机器周期；而后者为两条指令共 3 字节，需要 4 个机器周期才能执行完毕。如：

```
MOV  DPTR,#2368H      ;(DPTR) = 2368H;(DPH) = 23H,(DPL) = 68H,
MOV  DPTR,#35326      ;(DPTR) = 35326 = 89FEH
```

在使用通用数据传送指令时，应注意以下几点。

（1）通用数据传送指令不支持工作寄存器 R0～R7 之间的数据直接传送。如把 R2 的内容传递给 R5，但可以采用下面的方法实现：

```
MOV  40H, R2
MOV  R5, 40H
```

如果知道此时 CPU 使用的当前工作寄存器组，可用传送指令实现。假设当时的 RS0＝0，RS1＝1，则 R2 和 R5 的地址分别是 12H 和 15H，把 R2 的内容传递给 R5 可以采用：

```
MOV      15H, R2
```

或

```
MOV      R5, 12H
```

（2）通用数据传送指令不支持工作寄存器 R0～R7 内容直接传送给由地址寄存器内容指定的单元，或由地址寄存器内容指定单元的内容送给工作寄存器 R0～R7，如果在程序中需要这样的数据传送，可以采用其他方式间接实现。例如，把地址寄存器 R1 内容指定的单元内容传送给工作寄存器 R5，可以采用：

```
MOV  A, @R1
MOV  R5, A
```

把 R7 内容送给地址寄存器 R0 内容指定的单元，可以用下面的方法：

```
MOV  30H, R7
MOV  @R0, 30H
```

(3) 数据传送指令中，地址寄存器只有 R0 和 R1 可以担当，其他工作寄存器无此功能。

(4) 虽然 MCS-51 单片机有两个 16 位的寄存器：PC 和 DPTR，但只有 DPTR 用户可以用指令方式直接设置其内容。

2. 堆栈操作指令

堆栈是在内部 RAM 中开辟的一个先进后出(后进先出)的区域，用来保护 CPU 执行程序的现场，如 CPU 响应中断和子程序调用时的返回地址、重要单元和寄存器的内容等。其中重要单元和寄存器的内容采用堆栈操作指令完成。在保护时，先把它们传送到堆栈区暂时保存，待需要时再从堆栈区取出送回原来的单元和寄存器。堆栈的操作有两种：入栈和出栈。

1) 入栈指令

```
PUSH  direct;
```

入栈指令的功能是把指定单元 direct 的内容压入堆栈，指令执行时不影响标志位 Cy，AC，OV，P。CPU 操作过程如下。

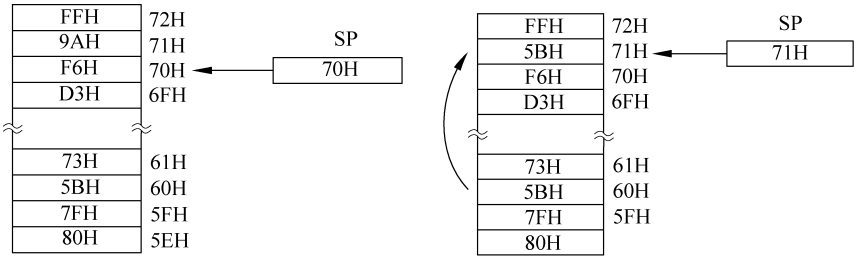
(1) (SP)+1→(SP)，修改堆栈指针。

(2) (direct)→[(SP)]，指定单元 direct 的内容入栈，即把该单元的内容送到由堆栈指针 SP 内容所指的单元。

例 3.3　把内部 RAM 的 60H 单元入栈。

```
MOV  SP,  #70H          ;(SP) = 70H,栈区开辟在 71H 开始的内部 RAM 区域
PUSH 60H                ;(SP) + 1→(SP),则(SP) = 71H;
                        ;(60H)→[(SP)],则(60H)→(71H),60H 单元内容被保存到栈
                        ;区的 71H 单元中,60H 单元的内容没有改变
```

例 3.3 的程序的执行过程如图 3.7 所示。



(a) PUSH指令执行前　　(b) PUSH指令执行后

图 3.7　PUSH 指令的执行过程

2) 出栈指令

```
POP  direct
```

出栈指令的功能是把堆栈中由(SP)所指单元的内容传送到指定的 direct 单元。指令执行时不影响标志位 Cy，AC，OV，P。CPU 操作过程如下。

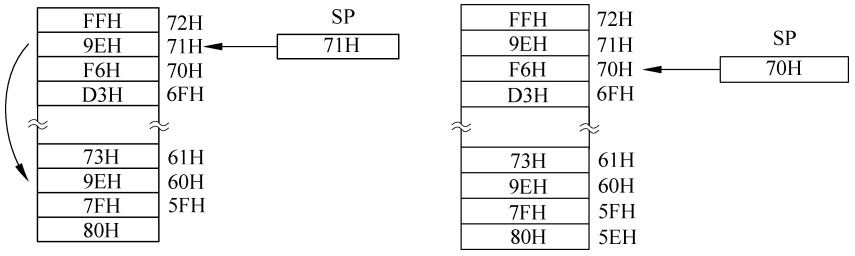
(1) [(SP)]→(direct)，出栈，把堆栈中由(SP)所指单元的内容传送到 direct 单元。

（2）（SP）－1→（SP），修改堆栈指针。

例 3.4 把存储在堆栈区 71H 单元的内容恢复到内部 RAM 的 60H 单元。

```
MOV  SP,  #71H          ;(SP) = 71H,目前的栈顶为 71H 单元
POP  60H                ;出栈,[(SP)]→(60H),即(71H)→(60H)
                        ;修改栈顶指针(SP) - 1→(SP),(SP) = 70H
```

上述程序的执行过程如图 3.8 所示。



(a) 数据出栈　　(b) 堆栈指针减1

图 3.8　POP 指令的执行过程

在使用堆栈时，应注意以下几点。

（1）PUSH 和 POP 指令的操作数必须是单元地址。PUSH 指令中指定的单元地址是被保护单元的地址（源操作数），指令隐含了目的操作数；而 POP 指令中指定的单元地址是内容要恢复的单元地址（目的操作数），指令隐含了源操作数。

（2）MCS-51 单片机的堆栈建在内部 RAM 中，单片机复位后，（SP）＝07H，从 08H 单元开始的区域均为栈区。在应用系统中，一般把栈区开辟在内部 RAM 的 30H～7FH 这一区域，栈区最好靠近内部 RAM 的末端，以避免堆栈向上增长时覆盖有效数据。

（3）在使用堆栈操作指令时，入栈指令 PUSH 和出栈指令 POP 应成对出现，保护指定单元内容时，必须遵循先进后出的原则，否则，单元内容在出栈恢复时会发生改变。

（4）MCS-51 单片机不支持对工作寄存器 R0～R7 直接使用堆栈操作指令。如果要用堆栈操作保护寄存器 Rn（n＝0～7）的内容，可对该工作寄存器对应的单元操作。如当 RS1 和 RS0 为 10 时，把 R5 的内容入栈，可用“PUSH 15H”；出栈用“POP 15H”即可恢复 R5 的内容。

例 3.5 设当前堆栈指针（SP）的内容为 70H，20H 单元的内容为 0FFH。下列程序执行后，DPTR 的内容是多少？

```
MOV  DPTR,  #0123H
MOV  R0, #20H
PUSH DPH
PUSH DPL
MOV  A,  @R0
MOV  DPL,  A
MOV  DPH,  #50H
MOV  R0,  #00H
POP  DPL
POP  DPH
```

程序执行过程分析如下：

```
MOV   DPTR, #0123H      ;(DPTR) = 0123H
MOV   R0, #20H          ;(R0) = 20H
PUSH  DPH               ;(SP) + 1→(SP),(SP) = 71H,DPH 的内容入栈,则(71H) = 01H
PUSH  DPL               ;(SP) + 1→(SP),(SP) = 72H,DPL 的内容进栈,则(72H) = 23H,
                        ;此时,DPTR 的内容全部入栈保护
MOV   A,  @R0           ;[(R0)]→(A),即[20H]→(A),(A) = 0FFH
MOV   DPL,  A           ;(DPL) = 0FFH
MOV   DPH,  #50H        ;(DPH) = 50H,此时,(DPTR) = 50FFH
MOV   R0,  #00H         ;(R0) = 00H
POP   DPL               ;恢复 DPTR 内容.[(SP)]→(DPL),即[72H]→(DPL),
                        ;则(DPL) = 23H.(SP) - 1→(SP),(SP) = 71H
POP   DPH               ;[(SP)]→(DPH),即[71H]→(DPH),则(DPH) = 01H
                        ;(SP) - 1→(SP),(SP) = 70H.此时,(DPTR) = 0123H,内容恢复
```

在例 3.5 中,在 DPTR 内容入栈保护之后,DPTR 被释放,对它进行了其他操作,操作完成之后,又采用出栈操作恢复了 DPTR 内容。这种方法常用于子程序模块中。由于单片机中,存储单元和寄存器是唯一的,如果在子程序中使用了相同的资源,但又不希望子程序运行时影响这些资源在调用之前的状态,在进入子程序时首先把它们入栈保护,释放资源,待调用子程序结束时,再利用出栈操作把它们恢复到子程序调用之前的状态,使调用前后它们保持相同的状态。如果交换 DPH 和 DPL 出栈的顺序,结果会是什么呢?

3. 交换指令

这是一组需要累加器 A 参与完成的指令,数据交换在内部 RAM 存储单元与累加器 A 之间或累加器 A 的高低 4 位之间进行,可以实现整字节或半字节的数据交换。

1) 字节交换指令

```
XCH   A,源操作数;
```

将源操作数与累加器 A 的内容互换,源操作数必须是工作寄存器、SFR 或内部 RAM 的存储单元。这组指令有 3 种形式:

```
XCH   A,Rn              ;(A)↔(Rn),n = 0～7
XCH   A,direct          ;(A)↔ (direct)
XCH   A,@Ri             ;(A)↔ [(Ri)],i = 0,1
```

例 3.6　将内部 RAM 的 20H 单元的内容与 40H 单元的内容交换。

方法一:

```
MOV   A,20H
XCH   A,40H
MOV   20H,  A
```

方法二:

```
MOV   A,20H
MOV   20H, 40H
MOV   40H, A
```

2) 半字节交换指令

```
XCHD   A,@Ri            ;(A)0～3↔ [(Ri)]0～3,i = 0,1
```

把指定单元内容的低 4 位与累加器 A 的低 4 位互换,而二者的高 4 位保持不变,如图 3.9 所示。

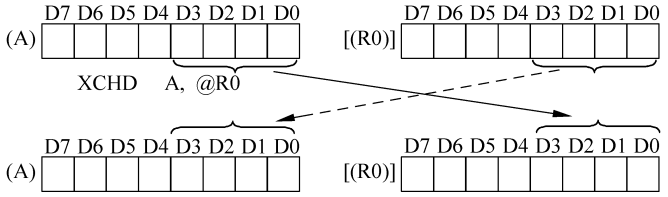


图 3.9 XCHD A,@R0 指令执行过程

例 3.7 编程实现把内部 RAM 的 20H 单元的内容低 4 位与 R7 单元的内容低 4 位交换。

程序如下：

```
MOV  R0,  #20H
MOV  A,  R7
XCHD A,  @R0          ;20H 单元内容的低 4 位与累加器 A 的低 4 位互换
MOV  R7,  A           ;累加器 A 内容送 R7,完成题目要求
```

3）高低 4 位互换指令

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

将累加器 A 的高 4 位和低 4 位互换，如图 3.10 所示。只有累加器 A 能够实现高 4 位和低 4 位互换。

例 3.8 把内部 RAM 存储单元 7BH 的内容高低 4 位互换。

```
MOV  A,  7BH
SWAP A
MOV  7BH,  A
```

例 3.9 8 位十进制数以压缩 BCD 码的形式依次存储在内部 RAM 以 2BH 单元开始的区域中，先存高位，设计程序把该数右移 2 位（即除以 100），并存储在原来的位置。

分析：十进制数右移 2 位，相当于这个数除以 100，假设 $X=12345678$，右移 2 位后，$X=123456$，移位前后存储在内部 RAM 中的映射如图 3.11 所示。显然，可以采用数据传送的方式实现上述要求，也可以采用单元内容交换的方法实现。

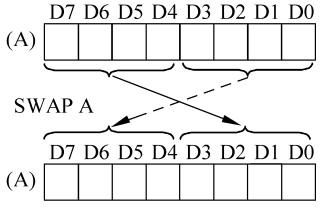


图 3.10 SWAP A 执行过程

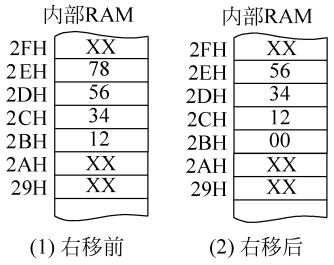


图 3.11 X 右移前后存储单元的内容

方法一：采用传送指令

```
MOV  2EH, 2DH
MOV  2DH, 2CH
MOV  2CH, 2BH
MOV  2BH, #0          ;存储最高位单元清零
```

方法二：采用交换指令

```
CLR   A
XCH   A, 2BH
XCH   A, 2CH
XCH   A, 2DH
XCH   A, 2EH
```

这两种方法的区别是：第一种方法在程序存储器中需要 12 个单元存储指令代码，执行时需要 8 个机器周期；而后者只需要 9 个单元，执行时间为 5 个机器周期。

4. 访问程序存储器的数据传送指令

MCS-51 单片机的程序存储器主要用于存放程序指令代码(Code)，还可用来存放程序中需要的常数。这些常数存储在程序存储器的一个区域，由若干连续单元构成，通常称为表或表格(Table)。因此，访问程序存储器的数据传送指令也叫查表(Lookup Table)，它的功能是从程序存储器某个单元读取一字节的常数。指令有两种形式：

```
MOVC  A,@A+DPTR        ;[(A)+(DPTR)]→(A),常数所在存储单元的地址由 DPTR 和累加
                       ;器 A 的内容之和确定
MOVC  A,@A+PC          ;(PC)+1→(PC),CPU 取指令代码;[(A)+(PC)]→(A),常数所在
                       ;存储单元的地址由程序计数器(PC)和累加器 A 的内容之和确定
```

例 3.10　采用查表方法获取一个数 x(0≤x≤15)的平方值。

首先在程序存储器中建立一个 0≤x≤15 的平方表，定义从 5000H 开始的连续 16 个单元中分别存有 0～15 的平方值。x 存放在累加器 A 中，x 取值为 00H～0FH。程序执行后，得到 x 的平方值在累加器 A 中。分别用上述两种指令设计查表程序。

(1) 采用“MOVC A,@A+DPTR”指令。

```
MOV   DPTR,#5000H      ;表的首地址送 DPTR
MOVC  A,@A+DPTR        ;查表获得的值送累加器 A
RET                    ;返回
ORG   5000H            ;0≤x≤15 的平方表从 5000H 存放
  DB  00H              ;0^2,DB: Define a byte,伪指令,在此定义 5000H 单元的内容是常
                       ;数而非指令代码,在程序中起说明作用
  DB  01H              ;1^2 存放在 5001H 单元
  DB  04H              ;2^2 存放在 5002H 单元
…
  DB  0E1H             ;15^2 = 225,存放在 500FH 单元
```

说明：若(A)=00H，程序执行后，[(A)+(DPTR)]→(A)，即(00+5000)=(5000H)→(A)，则(A)=00H。若(A)=02H，则[(A)+(DPTR)]→(A)，即(5002H)→(A)，(A)=04H。

实际上，如果平方表放在程序存储器的其他地方，如存放在 3700H，只要在程序中修改表的首地址，“MOV DPTR,#3700H”，程序运行的结果是相同的。

(2) 采用“MOVC A,@A+PC”指令。

由于这条指令执行时，其结果与 PC 有关，为了分析方便，把每条指令及其代码同时给出，程序从程序存储器的 1000H 单元开始存放，同样，x 存放在累加器 A 中，x 取值为 00H～0FH。程序执行后得到 x 的平方值在累加器 A 中。

【标号】	【指令】	【注释】	【单元地址】	【指令代码】
CHECKUP:	INC A	;(A)的内容+1	1000H	04
	MOVC A,@A+PC	;(PC)+1→(PC),	1001H	83

```
                        ;[(A)+(PC)]→(A)
RET                     ;返回                1002H        22
DB 00H                  ;0²                  1003H        00H
DB 01H                  ;1²                  1004H        01H
DB 04H                  ;2²                  1005H        04H
…                                                        …
DB 0E1H                 ;15²                 1012H        0E1H
```

说明：如果(A)＝02H时，执行此程序，首先(PC)＝1000H，由于“INC A”为单字节指令，(PC)＋1→(PC)，则(PC)＝1001H，CPU取指令代码04，解释执行后(A)被加1，其内容变为03H。接着，由于(PC)＝1001H，(PC)＋1→(PC)，则(PC)＝1002H，CPU取代码83解释执行：[(A)＋(PC)]→(A)，即[03＋1002]→(A)，则(A)的内容为04H，即2的平方。

如果去掉指令“INC A”，执行的结果正确吗？从程序代码在程序存储器中的存储位置来看，查表指令与平方表首地址之间有1字节的偏移量，给累加器加1正是为了弥补这个偏移量而设置的。如果没有加1处理，执行结果是不对的。

MCS-51单片机中从程序存储器中获取常数只有通过累加器A，虽然上述两条指令的功能是相同的，但二者在使用时查表的范围是不一样的。

(1) “MOVC A,@A＋DPTR”指令中，累加器A和DPTR的内容用户可以通过指令直接设定。16位数据指针DPTR为基址寄存器，取值范围为0000H～0FFFFH，也就是说常数表可以放置在程序存储器64KB空间的任何位置，而且表的最大长度可以接近64KB。

(2) “MOVC A,@A＋PC”是以程序计数器(PC)作为基址寄存器，该指令的执行结果与PC的内容有关，指令执行时PC的内容无法用传送指令指定，只有累加器A是可改变的，它取值范围为00H～0FFH。因此，使用这条指令时，常数表必须紧跟该指令存放，且长度不能大于256字节。

5. 访问外部RAM和外部I/O口的数据传送指令

MCS-51单片机系统中，扩展的外部RAM和外部I/O口是统一编址的，CPU访问外部RAM和外部I/O口时使用相同的指令，助记符为MOVX。CPU访问外部RAM和外部I/O口时采用间接寻址方式，外部RAM单元地址或外部I/O口地址由地址寄存器的内容指出，其中DPTR，R0，R1可作为地址寄存器，因此，有两种形式的指令。

1) 以DPTR为地址寄存器的访问外部RAM和外部I/O口的指令

(1) 读(输入)指令。

```
MOVX  A,@DPTR        ;[(DPTR)]→(A);
```

在读控制信号$\overline{RD}$为0时，把DPTR内容指出的外部RAM单元的内容或外部I/O口的状态读到累加器A中。DPTR内容指出16位地址，寻址范围为0000H～0FFFFH，即64KB。CPU执行读外部数据存储器和外部I/O口指令的时序如图3.12所示。CPU读数据时，单元地址分别由P0和P2口输出，P0输出DPL(低8位地址)，P2输出DPH(高8位地址)。

例3.11 把外部RAM的2000H单元的内容存入单片机内部RAM的30H单元。

```
MOV   DPTR, #2000H  ;把外部 RAM 单元的地址放入 DPTR
MOVX  A,   @DPTR    ;把外部 RAM 单元存储的数据读入 CPU
MOV   30H, A        ;存储读取的数据到 30H 单元
```

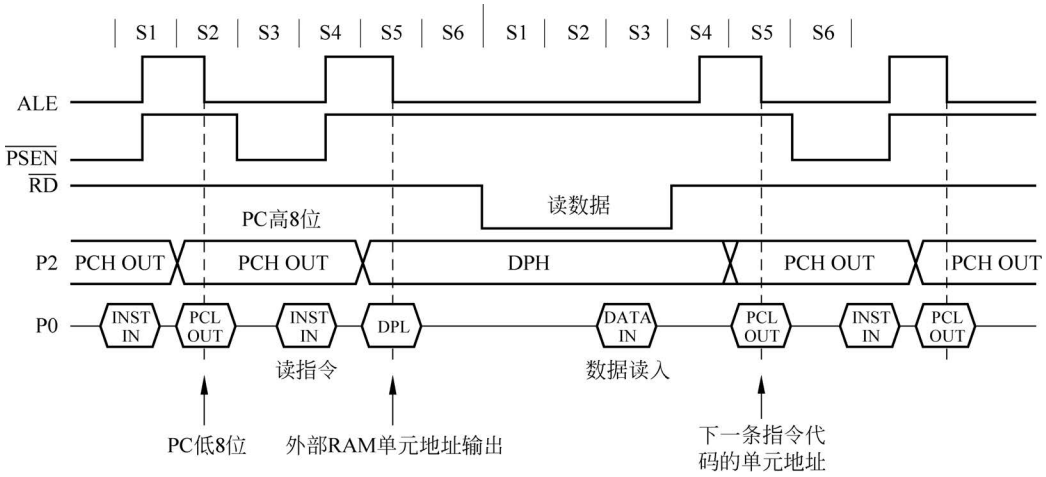


图 3.12　CPU 执行读外部数据存储器和外部 I/O 口指令的时序

(2) 写(输出)指令。

```
MOVX   @DPTR,A          ;(A)→[(DPTR)];
```

在写控制信号 $\overline{WR}$ 为 0 时,把单片机累加器 A 的内容输出到 DPTR 内容指出的外部 RAM 单元或外部 I/O 口,DPTR 内容指出 16 位地址,寻址范围为 0000H～0FFFFH,即 64K。CPU 执行写外部数据存储器和外部 I/O 口指令的时序如图 3.13 所示。

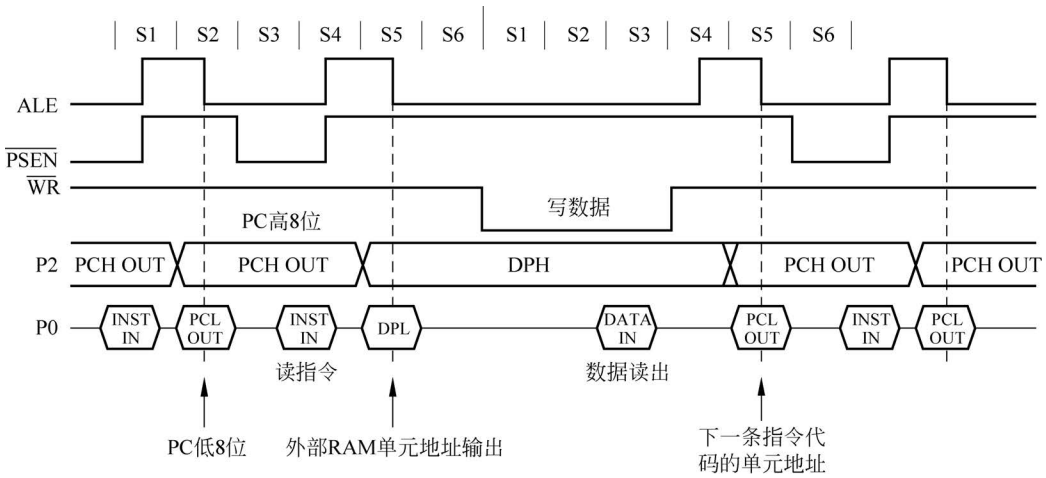


图 3.13　CPU 执行写外部数据存储器和外部 I/O 口指令的时序

例 3.12　把单片机内部 RAM 的 20H 单元的内容转存到外部 RAM 的 8000H 单元。

```
MOV    DPTR,    #8000H    ;把外部 RAM 单元的地址放入 DPTR
MOV    A,       20H       ;从内部 RAM 的 20H 单元读取要写入的数据
MOVX   @DPTR,   A         ;把数据写到外部 RAM 的 8000H 单元
```

2) 以 R0 和 R1 为地址寄存器的访问外部 RAM 和外部 I/O 口的指令

(1) 读(输入)指令。

```
MOVX   A,@Ri            ;[(P2)(Ri)]→(A),i=0,1
```

以 R0 或 R1 内容作为低 8 位地址，由 P0 口送出，寻址范围为 00H～0FFH，即 256 个单元的地址空间，高 8 位由当前的 P2 口状态提供。在控制信号 $\overline{RD}$(P3.7)为 0 时，把指定单元或 I/O 口的内容读入累加器 A。

(2) 写(输出)指令。

```
MOVX   @Ri,A          ;(A)→[(P2)(Ri)],i=0,1
```

以 R0 或 R1 内容作为低 8 位地址，由 P0 口送出，寻址范围为 256 个单元的地址空间，高 8 位由当前的 P2 口状态提供。在控制信号 $\overline{WR}$(P3.6)为 0 时，把累加器 A 的内容输出到指定的单元或 I/O 口。

上述两种指令的操作时序与图 3.12 和图 3.13 相同。值得一提的是，采用 R0 或 R1 作为地址寄存器指出的是外部 RAM 和外部 I/O 口的低 8 位地址，当扩展的数据存储器单元和 I/O 口的空间不大于 256 个时，P2 口可以作为 I/O 口使用。

例 3.13 256 个单元的 RAM 芯片与 8051 单片机的连接如图 3.14 所示，外部 RAM 的地址范围为 00H～0FFH，此时系统中 P1 和 P2 口为 I/O 口，设 R0 和 R1 的内容分别为 12H 和 34H，外部 RAM 的 34H 单元的内容为 56H，分析下列指令序列的执行结果。

```
MOVX   A,   @R1
MOVX   @R0,   A
MOV    @R0,   A
```

分析如下：

```
MOVX   A,   @R1       ;外部 RAM 的 34H 单元内容送给累加器 A
MOVX   @R0,   A       ;累加器 A 内容送给外部 RAM 的 12H 单元,外部 RAM(12H) = 56H
MOV    @R0,   A       ;累加器 A 内容送给内部 RAM 的 12H 单元,内部 RAM(12H) = 56H
```

上述指令序列执行后，外部 RAM 的 34H 单元的内容分别被送到内部 RAM 的 12H 和外部 RAM 的 12H 单元。另外，CPU 在执行“MOVX @Ri，A”和“MOVX A，@Ri”指令时，特殊功能寄存器 P2 的内容会保持在 P2 口上。

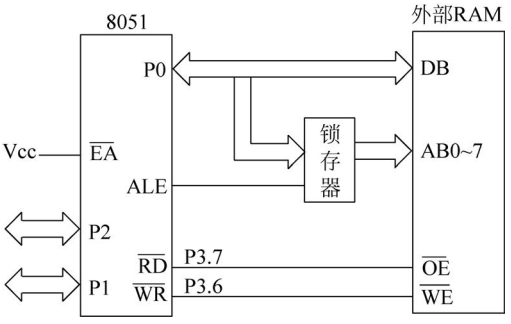


图 3.14 外部 RAM 芯片与 8051 单片机的连接

CPU 对外部 RAM 和外部 I/O 口的读写必须通过累加器 A，外部 RAM 和外部 I/O 口的地址为 16 位，内部 RAM 的地址为 8 位，不属于同一个地址空间，它们之间不能直接进行数据传送。另外，读写控制信号 $\overline{RD}$ 和 $\overline{WR}$ 仅在执行 MOVX 时才会有效，系统中没有对外部 RAM 和外部 I/O 口读写时，P3.6 和 P3.7 可作为 I/O 口使用。

3.3.3　算术运算指令

算术运算类指令(Arithmetic Instruction)共有 24 条,实现加、减、乘、除、加 1、减 1 及十进制调整等运算。MCS-51 单片机指令系统仅提供两个单字节无符号二进制数的算术运算,带符号或多字节二进制数的算术运算,需要通过设计算法把它们转化为单字节无符号二进制数的算术运算才能实现。

1. 加法指令

1) 不带进位位的加法指令

不带进位位的 8 位二进制数加法指令的一般形式:

```
ADD    A,源操作数      ;(A)+源操作数→(A)
```

这组指令实现两个无符号的 8 位二进制数加法运算,运算结果存储在累加器 A 中,运算过程影响标志位 Cy,AC,OV,P,有以下 4 种指令形式:

```
ADD    A,#data         ;(A)+ data→(A)
ADD    A,Rn            ;(A)+(Rn)→(A),n=0～7
ADD    A,direct        ;(A)+ (direct)→(A)
ADD    A,@Ri           ;(A)+[(Ri)]→(A),i=0,1
```

加法指令执行过程与标志位之间的关系如图 3.15 所示;若最高位 D7 在运算过程中产生进位,则(Cy)=1;若低半字节(低 4 位)在运算过程中向高半字节(高 4 位)进位位时,则(AC)=1;D6 与 D7 两位在运算过程中其中一位有进位位,而另一位没有,则(OV)=1,否则,(OV)=0。运算结果(A)中 1 的个数为偶数,(P)=0,否则,(P)=1。

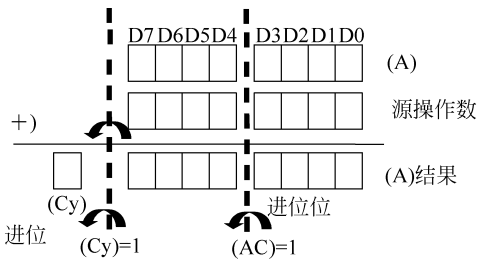


图 3.15　加法指令执行过程与标志位之间的关系

例 3.14　设(A)=0C3H,(20H)=0AAH,执行指令"ADD A,20H",执行过程如图 3.16 所示,则(A)=6DH,(Cy)=1,(OV)=1,(AC)=0,(P)=1。

例 3.15　单字节二进制加法。已知 x 存放在 20H 单元,y 存放在 21H 单元,求 $z=x+y$(设 z 小于 0FFH)。

图 3.17 是两个单字节相加的过程,程序如下:

```
MOV    A,20H
ADD    A,21H
MOV    22H,A           ;结果存在 22H 单元
```

```
  1100 0011
+ 1010 1010
-----------
1 0110 1101
```

图 3.16　例 3.14 的执行过程

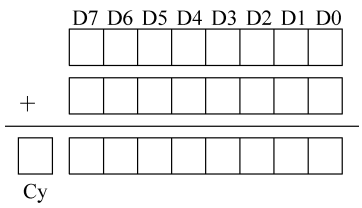


图 3.17　两个单字节相加的过程

2）带进位位的加法指令

8位二进制带进位位加法指令的一般形式：

```
ADDC  A,源操作数      ;(A)+源操作数+(Cy)→(A)
```

这组指令实现两个无符号的8位二进制数的加法运算，并且在运算时，把当前的进位位Cy的状态计入运算结果，运算结果存储在累加器A中，运算过程影响标志位Cy，AC，OV，P，有以下4种指令形式：

```
ADDC  A,#data        ;(A)+ data+(Cy)→(A)
ADDC  A,Rn           ;(A)+ (Rn) +(Cy)→(A),n=0～7
ADDC  A,direct       ;(A)+ (direct) +(Cy)→(A)
ADDC  A,@Ri          ;(A)+[(Ri)] +(Cy)→(A),i=0.1
```

例3.16 单字节二进制加法：已知x存放在20H单元，y存放在21H单元，求$z=x+y$。

两个任意的8位二进制数单字节相加，结果会是几字节呢？可以事先估计一下，然后给计算结果分配适度的存储单元。因为是无符号数，一字节最大的二进制数为0FFH，最小为00H。如果x，y都是0FFH，$x+y$的结果为1FEH。因为计算机中的数据是以字节形式存放的，所以需两个单元存储。给z分配两个单元：22H和23H，前者存放z的高8位，后者存储z的低8位。程序的实现算法如图3.18所示。

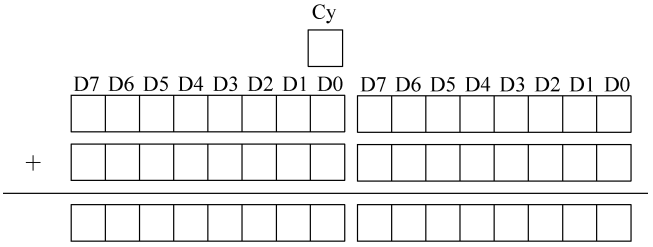


图3.18 两个单字节相加实现算法

程序如下：

```
MOV   A,  20H
ADD   A,  21H
MOV   23H,  A        ;和的低8位
MOV   A,  #00
ADDC  A,  #00        ;处理进位
MOV   22H,  A        ;和的高8位
```

3）加1指令

加1(Increment)指令是把指定单元或寄存器的内容加1，指令的一般形式：

```
INC   源操作数        ;源操作数+1→源操作数
```

这组指令有以下形式：

```
INC   A              ;(A)+1→(A),该指令执行时,不影响标志位Cy、AC和OV
INC   Rn             ;(Rn)+1→(Rn),n=0～7.该指令执行时,不影响标志位
INC   direct         ;(direct)+1→(direct),该指令执行时,不影响标志位
INC   @Ri            ;[(Ri)]+1→[(Ri)],i=0, 1.该指令执行时,不影响标志位
INC   DPTR           ;(DPTR)+1→(DPTR),该指令执行时,不影响标志位
```

例 3.17　设 R0 的内容为 7EH，内部 RAM 的 7EH 和 7FH 单元的内容分别为 0FFH 和 40H，P1 口的内容为 55H，执行下列指令后，R0，P1，7EH 和 7FH 单元的内容分别是多少？

```
INC   @R0
INC   R0
INC   @R0
INC   7FH
INC   P1
```

分析：

```
INC   @R0          ;[(R0)] + 1→[(R0)]，即[7EH] + 1→[7EH]，所以，(7EH) = 00H
INC   R0           ;(R0) + 1→(R0)，所以，(R0) = 7FH
INC   @R0          ;[(R0)] + 1→[(R0)]，即[7FH] + 1→[7FH]，所以，(7FH) = 41H
INC   7FH          ;(7FH) + 1→(7FH)，所以，(7FH) = 42H
INC   P1           ;(P1) + 1→(P1)，(P1) = 56H，并从 P1 口输出
```

程序执行结束后，(R0)＝7FH，(P1)＝56H，(7EH)＝00H，(7FH)＝42H。

例 3.17 中，(7EH)＝0FFH，该单元内容加 1 变成了 00H，这种现象称为上溢。类似地，(DPTR)＝0FFFFH 时，“INC DPTR”指令执行后，DPTR 内容为 0000H。在程序中使用 INC 类指令时，应注意上溢现象。

在例 3.17 中，对 P1 口进行加 1 操作“INC P1”时，CPU 进行了三步操作：第一，读 P1 寄存器；第二，P1 寄存器加 1 计算，写 P1 寄存器和引脚输出；第三，P1 寄存器内容刷新、引脚状态重置。指令执行改变了 P1 口的输出状态和对应的 SFR 的内容。因此，用 INC 类指令对 P0，P1，P2，P3 口操作时，参与运算的是 I/O 口对应的寄存器的内容，而不是来自于引脚的状态，但最终的运算结果将从 I/O 口的引脚输出，并修改对应寄存器的内容。

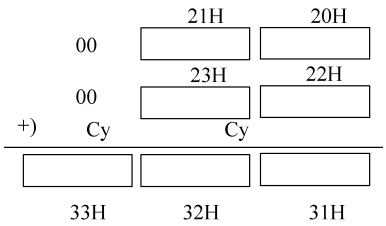


图 3.19　两个双字节相加实现算法

例 3.18　双字节二进制加法。

双字节加法的实现算法如图 3.19 所示，设 x 存放在 21H，20H 单元（高 8 位在 21H 单元），y 存放在 23H，22H 单元，$z=x+y$ 存放在 33H，32H，31H 单元。程序如下：

```
MOV   R0,   #20H     ; 指向被加数的低 8 位
MOV   R1,   #22H     ;指向加数的低 8 位
MOV   A,    @R0
ADD   A,    @R1
MOV   31H,   A       ;结果的低 8 位
INC   R0             ;修改单元地址
INC   R1
MOV   A,    @R0
ADDC  A,    @R1
MOV   32H,   A       ;结果的中 8 位
MOV   A,    #00
ADDC  A,    #00      ;处理进位
MOV   33H,   A       ;结果的高 8 位
```

4）十进制加法调整指令

十进制数采用 BCD 码表示时，用 4 位二进制编码来表示 1 位十进制数，采用紧凑格式

存储，一个单元可存放两位十进制数，通常叫作压缩 BCD 码格式(Packed-BCD format)。在 MCS-51 单片机中，不论是不带进位的加法指令，还是带进位的加法指令，它们仅支持两个 8 位二进制数的运算。两个十进制数相加时也必须借助于加法指令实现，用二进制数的加法运算法则处理 BCD 码的加法运算，但其运算结果会产生错误。如计算十进制 99 与 23 之和，它们的压缩 BCD 码分别是 10011001 和 00100011，在单片机中它们相加的计算过程如图 3.20 所示。加法指令执行结果为 10111100B，即 0BCH，显然“B”和“C”不是十进制数的数符，运算结果是不正确的。因此，必须对上述结果进行调整。十进制加法调整指令的功能就是在用加法指令完成 BCD 码加法运算之后，对运算结果进行处理，把运算结果转换为 BCD 码形式。指令如下：

```
1001 1001
+ 0010 0011
1011 1100
```

图 3.20　BCD 码 99 与 23 相加的运算过程

```
DA    A;
```

CPU 执行“DA　A”的流程如图 3.21 所示。若$(A)_{0\sim3}>9$ 或(AC)＝1，则(A)＋06H→(A)；若$(A)_{4\sim7}>9$ 或(Cy)＝1，则(A)＋60H→(A)，指令执行时影响标志位 Cy，AC，OV 和 P。使用“DA　A”指令时，必须注意以下几点。

(1) 该指令的前提是进行了两个 2 位十进制数(BCD 码)的加法，需要对加法运算的结果进行调整，使结果变为十进制数，即将累加器 A 中的和调整为 BCD 码。

(2) 必须与加法指令联合使用。

(3) 单独使用该指令时，不能保证累加器 A 中的数据正确地转换为 BCD 码，因为“DA A”的调整结果不仅依赖于累加器 A 的内容，而且与标志位 Cy 和 AC 的状态有关。

例 3.19　两个 4 位十进制数以紧凑 BCD 码格式存储在内部 RAM 中，编程实现求这两个数的和。

十进制数在计算机中以 BCD 码存储，一个单元可以存储一个 2 位十进制数，即两个 BCD 码，存储 4 位十进制数需两个单元。实现算法如图 3.22 所示。

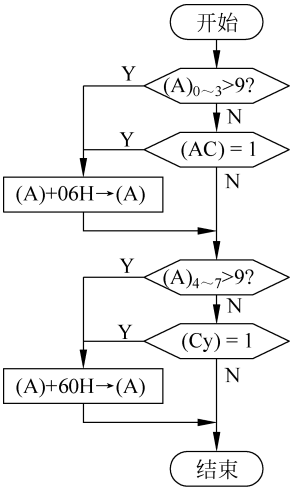


图 3.21　CPU 执行“DA　A”的流程

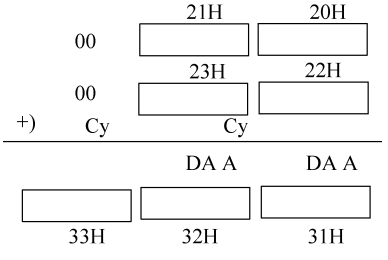


图 3.22　两个 4 位十进制数相加

程序如下：

```
MOV    R0,    #20H      ;指向被加数的低 2 位
```

```
MOV   R1,  #22H     ;指向加数的低 2 位
MOV   A,   @R0
ADD   A,   @R1
DA    A
MOV   31H,  A       ;结果的低 2 位; 十位和个位
INC   R0            ;修改单元地址
INC   R1
MOV   A,   @R0
ADDC  A,   @R1
DA    A
MOV   32H,  A       ;结果的中 2 位; 十位和百位
MOV   A,   #00
ADDC  A,   #00      ;处理进位
MOV   33H,  A       ;结果的高 2 位,最高 2 位无须调整,其结果只有 00 或 01 两种可能
```

2. 减法指令

1）带借位的减法指令

MCS-51 单片机没有不带借位的减法指令，带借位的减法指令一般形式：

```
SUBB  A,源          ;(A) - 源 - (Cy)→(A)
```

这组指令是把两个无符号的 8 位二进制数相减后，再减去当前的借位位 Cy 的状态，运算结果在累加器 A 中；指令执行时影响标志位 Cy，AC，OV，P。指令有以下 4 种形式：

```
SUBB  A,#data       ;(A) - data - (Cy)→(A)
SUBB  A,Rn          ;(A) - (Rn) - (Cy)→(A)
SUBB  A,direct      ;(A) - (direct) - (Cy)→(A)
SUBB  A,@Ri         ;(A) - [(Ri)] - (Cy)→(A)
```

例 3.20　设累加器 A 的内容为 0C9H，寄存器 R2 的内容为 5AH，当前 Cy 的状态为 1，执行指令“SUBB A，R2”后，累加器 A 和标志位 Cy，AC，OV，P 的状态如何？

指令“SUBB A，R2”的执行过程如图 3.23 所示。累加器 A 的内容为 6EH，(Cy)＝0，(AC)＝1，(OV)＝0，(P)＝1。

```
   111 11        借位
  1100 1001      (A)
  0101 1010      (R2)
-        1       (Cy)
  0110 1110      (A)  差
```

图 3.23　“SUBB A，R2”的执行过程

0C9H－5AH＝6FH，而例 3.20 中累加器 A 的内容为 6EH，这是因为减法指令执行时，当前的进位位 Cy 参与了运算。在进行减法时，如果不能确定进位位 Cy 的状态，在应用减法指令时，必须对进位位 Cy 清零（用指令“CLR Cy ”或“CLR C”），以保证正确的运算结果。

2）减 1 指令

减 1 指令的一般形式：

```
DEC   源            ;源操作数 - 1→源操作数
```

源操作数必须是一个寄存器或存储单元，有以下 4 种形式：

```
DEC   A             ;(A) - 1→(A),该指令执行时,不影响标志位 Cy,AC 和 OV
DEC   Rn            ;(Rn) - 1→(Rn),n = 0～7,该指令执行时,不影响标志位
DEC   direct        ;(direct) - 1→(direct),该指令执行时,不影响标志位
DEC   @Ri           ;[(Ri)] - 1→[(Ri)],i = 0, 1,该指令执行时,不影响标志位
```

若原来寄存器或单元的内容为 00H，减 1 运算后，其内容变为 0FFH，即向下溢出。与

INC 类指令类似，对 I/O 口操作时，参与减 1 运算的是 I/O 口对应的寄存器的内容，而不是来自于引脚的状态，但最终的运算结果将从 I/O 口的引脚输出，并修改寄存器的内容。

例 3.21 设 R0 的内容为 7EH，内部 RAM 的 7DH 和 7EH 单元的内容分别为 00H 和 40H，P1 口寄存器的内容为 55H，执行下列指令后，R0，P1 和 7EH 单元的内容分别是多少？

```
DEC   @R0
DEC   R0
DEC   @R0
DEC   7EH
DEC   P1
```

分析：

```
DEC   @R0          ;[(R0)]－1→[(R0)]，即[7EH]－1→[7EH]，所以，(7EH)＝3FH
DEC   R0           ;(R0)－1→(R0)，所以，(R0)＝7DH
DEC   @R0          ;[(R0)]－1→[(R0)]，即[7DH]－1→[7DH]，所以，(7DH)＝0FFH
DEC   7EH          ;(7EH)－1→(7EH)，所以，(7EH)＝3EH
DEC   P1           ;(P1)－1→(P1)，(P1)＝54H，并从 P1 口输出
```

上述指令执行后，(R0)＝7DH，(7EH)＝3EH，(P1)＝54H，P1 口输出 54H。

例 3.22 双字节二进制减法：x 存放在 20H，21H 单元(高 8 位在 20H 单元)，y 存放在 22H，23H 单元(高 8 位在 22H 单元)，$x \geqslant y$，求 $z=x-y$。

双字节二进制减法算法如图 3.24 所示。程序如下：

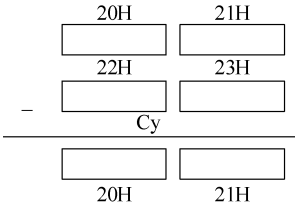


图 3.24 双字节二进制减法算法

```
MOV   R0,  #21H    ;指向被减数的低 8 位
MOV   R1,  #23H    ;指向减数的低 8 位
MOV   A,   @R0
CLR   Cy
SUBB  A,   @R1
MOV   @R0,  A      ;存结果的低 8 位
DEC   R0           ;修改单元地址
DEC   R1
MOV   A,   @R0
SUBB  A,   @R1
MOV   @R0,  A      ;存结果的高 8 位
```

例 3.23 2 位十进制数减法。

MCS-51 单片机没有十进制数减法指令。为了实现十进制数减法，引入十进制数补码，采用补码相加的方法实现。例如：

$$67-34=67+[-34]_{补}$$

在二进制求反码时，数符 1 的反码为 0，0 的反码为 1。按照这样的规律类比，十进制时，数符 0 的反码为 9，1 的反码为 8，2 的反码为 7，……，9 的反码为 0，那么，有

$$[-34]_{反}=65$$

$$[-34]_{补}=[-34]_{反}+1=66$$

这样，$67-34=67+[-34]_{补}=67+66=133$，如果将最高位丢弃，运算结果是准确无误的。实际上，2 位十进制数的补码还可以采用另外一种简捷的方法，如：

$$[-34]_{补}=100-34=66$$

上述过程还可以写成：

$$[-34]_{补}=100-34=99-34+1=\ 65+1=66$$

进一步也可写成：

$$[-34]_{补}=99+1-34\Rightarrow 9A-34=66$$

通过以上分析，可以得到 2 位十进制数减法的算法。设 2 位十进制数 x 和 y 分别为被减数和减数，$x\geqslant y$，差为 z，那么：

$$z=x-y=x+[-y]_{补}=x+100-y=x+99+1-y\Rightarrow x+9A-y$$

这个算法分两步，第一步是求补码，实质上是一个二进制数减法；第二步是十进制数加法。程序如下：

```
MOV   A,   #9AH
CLR   C              ;清进位位
SUBB  A,   R5        ;减数 y 存放在 R5 中,求出的补码在累加器 A 中
ADD   A,   R4        ;被减数 x 存放在 R4 中
DA    A              ;调整十进制数加法运算的结果
MOV   R6,  A         ;差 z 存放在 R6 中
```

3. 乘法指令

```
MUL   AB ;
```

乘法指令实现(A)×(B)，乘积的高 8 位存储在寄存器 B 中，低 8 位在累加器 A 中，若乘积大于 255，即寄存器 B 内容非 0 时，则溢出标志 OV 置 1；若乘积小于 255，即寄存器 B 内容为 0 时，则溢出标志 OV 清零。不论在哪种情况下，乘法指令执行结束时，Cy 总是被清零的。

乘法指令实现的是两个 8 位无符号二进制数相乘，结果是两字节。只有累加器 A 和寄存器 B 具有实现乘法的功能，其他寄存器和单元不能直接实现。

例 3.24　已知(A)=4EH，(B)=5DH，执行指令“MUL AB”后的结果是：(B)=1CH，(A)=56H，(OV)=1，(Cy)=0。

例 3.25　已知 x 存放在 20H 单元，y 存放在 21H 单元，求 x×y。

```
MOV   A,   20H       ;取被乘数 x
MOV   B,   21H       ;取乘数 y
MUL   AB
MOV   22H,  A        ;乘积的低 8 位
MOV   23H,  B        ;乘积的高 8 位
```

例 3.26　已知 x 存放在 R2 和 R3 中，高 8 位在 R2 中，y 存放在 R1，求 x×y，乘积结果存在 R4，R5，R6 中。

图 3.25 是从多位十进制数乘法类比推理得到的多字节乘以单字节的实现算法。多字节乘法算法与手算十进制数乘法的方法是相同的，只不过用一字节代替了十进制数的一位。

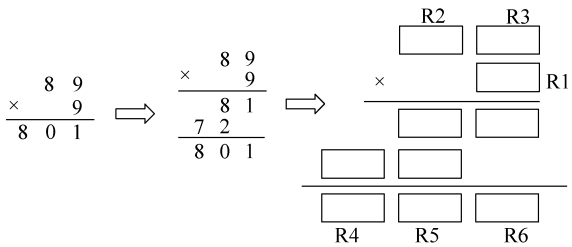


图 3.25　多字节乘以单字节的实现算法

程序如下：

```
MOV   A,  R3        ;x的低8位
MOV   B,  R1        ;乘数y
MUL   AB
MOV   R6,  A        ;乘积的低8位
MOV   R5,  B        ;暂存中间结果
MOV   A,  R2        ;x的高8位
MOV   B,  R1        ;乘数y
MUL   AB
ADD   A,  R5        ;求乘积的中8位
MOV   R5,  A        ;存储乘积的中8位
MOV   A,  #00
ADDC  A,  B         ;计算乘积的高8位,把中8位运算产生的进位计入高8位
MOV   R4,  A        ;存储乘积的高8位
```

4. 除法指令

```
DIV   AB
```

除法指令实现累加器A内容除以寄存器B的内容。指令执行后，商在累加器A中，余数在寄存器B中。若除数寄存器B的内容为0，则标志位OV置1；若寄存器B的内容不为0，则标志位OV清零；除法指令执行后，Cy被清零。

除法指令实现的是两个8位无符号的二进制数相除，商和余数也是8位无符号二进制整数。只有累加器A和寄存器B具有实现除法的功能，其他寄存器和单元不能直接实现。

例3.27 已知x存放在20H单元，y存放在21H单元，求x/y。

```
MOV   A,  20H       ;取被除数x
MOV   B,  21H       ;取除数y
DIV   AB
MOV   22H,  A       ;商存在22H单元
```

例3.28 把R7中的二进制数转换为十进制数，并以压缩BCD码的格式存放到R4和R5中。

一字节无符号二进制数的取值范围为00H～0FFH，对应的十进制数为0～255，以二进制数0FEH为例，它除以100，商即为其十进制数的百位“2”，余数为36H(54)。然后，余数再除以10，由本次运算的商和余数就可以得到其十进制数的十位和个位。压缩BCD码存储格式是用一个单元存储2位十进制数，组装上述运算得到的十进制数各位，结果为：百位“02”存在R4中，十位与个位“54”存在R5中。程序如下：

```
MOV   A,  R7        ;取被转换的二进制数
MOV   B, #100       ;
DIV   AB            ;被转换数除以100,商为百位数
MOV   R4,  A        ;转换的百位数存到R4
MOV   A,  B         ;取余数
MOV   B,  #10       ;
DIV   AB            ;被余数除以10,商为十位数,余数为个位数
SWAP  A             ;
ADD   A,  B         ;变换成压缩BCD码格式
MOV   R5,  A        ;十进制数的十位、个位
```

例3.29 4位十进制数以压缩BCD码的形式存储在R4和R5中，高位在R4中，把该

数转化为分离式 BCD 码的形式，存储在 30H 单元开始的区域。

BCD 码是十进制数符的 4 位二进制编码，如 23，它的压缩 BCD 码是 0010 0011，写成分离式 BCD 码形式，即用一字节表示一位十进制数，其结果是 0000 0010，00000011。因此，一字节的压缩 BCD 码分离方法是：用该字节除以 16，商为该字节高位对应的 BCD 码，余数为该字节低位对应的 BCD 码。程序如下：

```
MOV   R0,#30H       ;设置分离式 BCD 存储区首地址，千位
MOV   A,R4          ;取千位、百位分离
MOV   B,#10H
DIV   AB
MOV   @R0,A         ;存千位
INC   R0            ;修改存储单元地址
MOV   @R0,B         ;存百位
INC   R0            ;修改存储单元地址
MOV   A,R5          ;取十位、个位分离
MOV   B,#10H
DIV   AB
MOV   @R0,A         ;存十位
INC   R0
MOV   @R0,B         ;存个位
```

3.3.4　逻辑运算指令

逻辑运算指令可以完成与、或、异或、清零、求反和左右移位等操作。

1. 由累加器 A 实现的逻辑操作指令

1）累加器 A 清零指令

```
CLR   A
```

把累加器 A 的内容清零，只影响标志位 P。与“MOV A，#00H”的执行结果相同。

2）累加器 A 取反指令

```
CPL   A
```

累加器 A 的内容按位取反，不影响任何标志位。

例 3.30　设(A)＝56H (01010110B)，执行命令“CPL　A”，结果为 A9H(10101001B)。

3）累加器 A 循环左移指令

```
RL    A
```

把累加器 A 的内容循环左移 1 位，最高位移入最低位，指令操作不影响标志位。操作如图 3.26 所示，当(A)≤ 07FH 时，左移 1 位相当于(A)乘以 2。

4）累加器 A 带进位位循环左移指令

```
RLC   A
```

把累加器 A 的内容连同进位位 Cy 左移 1 位，累加器 A 的最高位移入进位位 Cy，而 Cy 原来的内容被移入累加器 A 的最低位，如图 3.27 所示。该指令每次只移动 1 位，影响标志位 Cy 和 P。

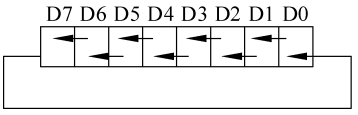


图 3.26 “RL A”操作

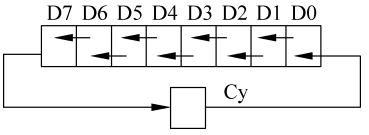


图 3.27 “RLC A”操作

例 3.31 设 x 存在于 33H 单元中，采用移位指令求 2x。

在二进制中，最低位补 0 左移一位，其结果为原数的 2 倍。程序如下：

```
MOV   A,  33H        ;取 x
CLR   Cy
RLC   A
MOV   20H,  A        ;结果的低 8 位
CLR   A;
RLC   A              ;处理进位
MOV   21H,  A        ;结果的高 8 位
```

5）累加器 A 循环右移指令

```
RR    A
```

把累加器 A 的内容右移 1 位，最低位移入最高位，如图 3.28 所示，该指令操作不影响任何标志位，当(A)为偶数时，右移 1 位相当于(A)除以 2。

6）累加器 A 带进位位循环右移指令

```
RRC   A
```

累加器 A 的内容连同进位位 Cy 被右移 1 位，最低位移入 Cy，而 Cy 原来的状态被移入累加器 A 的最高位，如图 3.29 所示。指令执行时影响标志位 Cy 和 P。

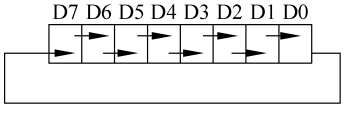


图 3.28 “RR A”操作

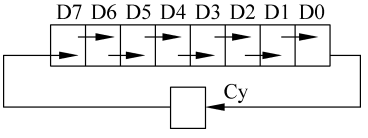
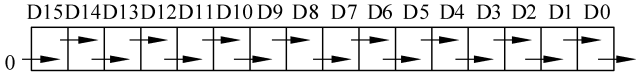


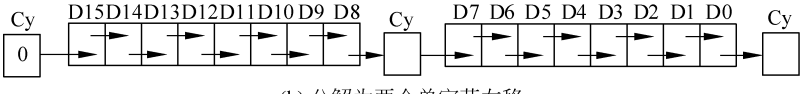
图 3.29 “RRC A”操作

例 3.32 多字节二进制数除以 2。

在二进制中，最高位补 0 右移一位，其结果为原数的 1/2。图 3.30 是用指令“RRC A”实现 16 位二进制数除以 2 的算法。因为“RRC A”仅能实现 8 位右移，因此，需要将多字节分解成多个单字节。



(a) 最高位补0右移一位，其结果为原数的1/2



(b) 分解为两个单字节右移

图 3.30 用带进位位循环右移“RRC A”实现除以 2 的算法

设二进制数存放在 R5 和 R6 中，结果仍存放在原处，程序如下：

```
MOV   A,   R5          ;高 8 位
CLR   C
RRC   A
MOV   R5,   A          ;商的高 8 位
MOV   A,   R6          ;低 8 位
RRC   A
MOV   R6,   A          ;商的低 8 位
```

实际上，程序执行完后，(Cy)为余数。

2. 与逻辑运算指令

与逻辑运算指令的一般形式：

```
ANL   目的操作数,源操作数;
```

这组指令实现两个 8 位二进制数的与运算，除了累加器 A 可作为目的操作数，单元也可以作为目的操作数。

1）以累加器 A 为目的操作数的与逻辑运算指令

```
ANL   A,#data          ;(A)∧ data→(A)
ANL   A,Rn             ;(A)∧(Rn)→(A), n=0～7
ANL   A,direct         ;(A)∧(direct)→(A)
ANL   A,@Ri            ;(A)∧[(Ri)]→(A), i=0,1
```

这是两个 8 位二进制数相与的运算，运算结果存储在累加器 A 中，由于与运算不产生进位，这 4 条指令执行时仅影响标志位 P。

2）以某个单元为目的操作数的与逻辑运算指令

```
ANL   direct,#data  ;(direct)∧data→(direct)
ANL   direct,A      ;(direct)∧(A)→(direct)
```

这组指令的特点在于：进行与运算时，一个单元作为目的操作数，因此，指令执行时不会影响任何标志位。这种操作方式只支持指定单元与 8 位二进制数和累加器 A 之间的运算。

设 $d_i(i=7\sim0)$为 8 位二进制数的 1 位数，进行与运算时：

$$d_i \wedge 0=0$$

$$d_i \wedge 1=d_i$$

因此，与运算常用于使某些位清零，实现屏蔽操作。如果要屏蔽某位，就把该位和 0 相与，要保留，则和 1 相与。

例 3.33　在单片机应用系统中，希望把从 P1 口读取的数据的 0,3,5,6 位状态保留，其他位状态屏蔽。

根据题意，保留 P1 口的 0,3,5,6 位的状态的屏蔽码为 01101001B，程序如下：

```
MOV   P1, #0FFH          ;P1 口全写 1,设置为输入状态
MOV   20H, P1            ;把从 P1 口地区的状态存入 20H 单元
ANL   20H, #01101001B    ;屏蔽无用位,保留 0,3,5,6 位
```

例 3.34　在单片机应用系统中，希望保留 P1 口的 0,3,5,6 位状态，其他位状态屏蔽。

```
ANL   P1, #01101001B     ;屏蔽无用位,保留 0,3,5,6 位,运算结果写入 P1 寄存器并从 P1 输出
```

上述指令执行时，首先读取 P1 寄存器的状态，再进行与运算，然后再把运算结果写入寄存器 P1，并从 P1 口输出，P1.1，P1.2，P1.4，P1.7 被清零了，输出低电平。

例 3.35 已知 2 位十进制数以压缩 BCD 码格式存放在 30H 单元，把 2 位数分开，以分离 BCD 码形式分别存放在两个单元 20H 和 21H 中。

设 21H，20H 单元分别存十位数和个位数的 BCD 码，根据题目要求，程序如下：

```
MOV   A,  30H
ANL   A,  #0F0H          ;取十位
SWAP  A                  ;把 BCD 码转移到低 4 位
MOV   21H,  A            ;存十位的 BCD 码
MOV   A,  30H
ANL   A,  #0FH           ;取个位
MOV   20H,  A            ;存个位的 BCD 码
```

3. 或逻辑运算指令

或逻辑运算指令的一般形式：

```
ORL   目的操作数,源操作数
```

这组指令实现两个 8 位二进制数的或运算，除了累加器 A 可作为目的操作数，单元也可以作为目的操作数。

1）以累加器 A 为目的操作数的或逻辑运算指令

```
ORL   A,#data            ;(A)∨data→(A)
ORL   A,Rn               ;(A)∨(Rn)→(A),n=0～7
ORL   A,direct           ;(A)∨(direct)→(A)t
ORL   A,@Ri;             ;(A)∨[(Ri)]→(A),i=0,1
```

这是两个 8 位二进制数相或的运算，运算结果存储在累加器 A 中，由于或运算不产生进位，指令执行时仅影响标志位 P。

2）以某个单元为目的操作数的或逻辑运算指令

```
ORL   direct,#data       ;(direct)∨data→(direct)
ORL   direct,A           ;(direct)∨(A)→(direct)
```

这组指令在进行或运算时，一个单元作为目的操作数，指令执行时不会影响任何标志位。这种操作方式只支持指定单元与 8 位二进制数和累加器 A 之间的运算。

设 $d_i(i=7\sim0)$为 8 位二进制数的 1 位数，进行或运算时：

$$d_i \vee 0 = d_i$$

$$d_i \vee 1 = 1$$

或运算常用于使某些位置 1，实现置位操作。如果要使某位置位，就把它与 1 相或。

例 3.36 把累加器 A 的低 4 位由 P1 口的低 4 位输出，并且保持 P1 口的高 4 位不变。

根据题目要求，程序如下：

```
ANL   A, #00001111B      ;提取 A 的低 4 位
MOV   R7,  A             ;暂存 A 的低 4 位
MOV   A,  P1             ;读取 P1 口
ANL   A,  #11110000B     ;保留 P1 高 4 位,屏蔽低 4 位
ORL   A,  R7             ;合并
MOV   P1,  A             ;输出
```

4. 异或逻辑运算指令

异或逻辑运算指令的一般形式：

```
XRL    目的操作数,源操作数
```

这组指令实现两个 8 位二进制数的异或运算，除了累加器 A 可作为目的操作数，单元也可以作为目的操作数。

1）以累加器 A 为目的操作数的异或逻辑运算指令

```
XRL    A,#data          ;(A)⊕data→(A)
XRL    A,Rn             ;(A) ⊕(Rn)→(A),n=0～7,
XRL    A,direct         ;(A)⊕(direct)→(A)
XRL    A,@Ri;           ;(A)⊕[(Ri)]→(A),i=0, 1
```

这是两个 8 位二进制数的异或运算，运算结果存储在累加器 A 中，由于异或运算不产生进位，因此，指令执行时仅影响标志位 P。

2）以某个单元为目的操作数的异或逻辑运算指令

```
XRL    direct,#data     ;(direct) ⊕data→(direct)
XRL    direct,A         ;(direct) ⊕(A)→(direct)
```

这组指令把一个单元作为目的操作数，指令执行时不会影响任何标志位。这种操作方式只支持指定单元与 8 位二进制数和累加器 A 之间的运算。

设 $d_i(i=7\sim0)$为 8 位二进制数的 1 位数，进行异或运算时：

$$d_i \oplus 0 = d_i$$

$$d_i \oplus 1 = \overline{d}_i$$

异或运算常用于使某些位取反。如果某位与 1 相异或，就把该位取反；与 0 相异或，则可以保持该位原来的状态不变。

例 3.37　已知一个负数的原码存放在 30H 单元，求它的补码。

负数求补码的步骤是：先求该数的反码，然后反码加 1。求反码可以采用以下 3 种方法。

（1）采用异或指令，最高位与 0 异或以保留符号位，数值位与 1 异或取反。

（2）采用取反指令，所有位均取反，然后最高位置 1 以实现保持符号位不变的目的。

（3）采用算术方法，先把负数取绝对值，即最高位清零，然后用 0FFH 减去它。

综上所述，求存储在 30H 单元负数补码的程序如下：

程序 1：

```
MOV    A,   30H
XRL    A,   #01111111B     ;求反码：保留符号位,数值位按位取反
ADD    A,   #01            ;补码=反码+1
MOV    30H,  A             ;存补码
```

程序 2：

```
MOV    A,   30H
CPL    A                   ;求 A 的各位全部取反
ORL    A,   #10000000B     ;恢复符号位
ADD    A,   #01H           ;补码=反码+1
MOV    30H,  A             ;存补码
```

程序 3：

```
ANL   30H, #01111111B       ;求绝对值
MOV   A, #0FFH
CLR   C
SUBB  A, 30H                ;求反码
ADD   A,#01H                ;求补码
MOV   30H,  A               ;存补码
```

3.3.5 位操作指令

位操作指令支持对位的直接操作，包括位传送、位逻辑运算以及位控制转移指令，为逻辑处理提供了一种高效的方法，可使逻辑电路软件化，减少系统中元器件的数量，提高系统可靠性。本节介绍位传送和位运算指令，位控制转移指令将在 3.3.6 节介绍。

1. 位数据传送指令

```
MOV   C,bit                 ;(bit)→(C)
MOV   bit,C                 ;(C)→(bit)
```

位传送指令仅支持某 1 个指定位 bit 与布尔处理器 C 之间的状态传送，两位之间不能直接进行状态传送，必须通过 C 来进行。

例 3.38 把单片机内部 RAM 中的标志位状态从 P1.2 引脚输出，设标志位存储在 28H.0 位。

程序如下：

```
MOV   C,  28H.0             ;取标志位的状态
MOV   P1.2,  C              ;输出,P1.2 的状态取决于标志位的状态
```

2. 位修正指令

1）清零

```
CLR   C                     ;0→(C)
CLR   bit                   ;0→(bit)
```

这组指令把位累加器 C 或指定位 bit 的状态清零。因为 C 是 PSW 的最高位 Cy，也可用“CLR Cy”指令清零。

2）置位

```
SETB  C                     ;1→(C)
SETB  bit                   ;1→(bit)
```

这组指令把位累加器 C 或指定位 bit 的状态置 1。

3）取反

```
CPL   C                     ;(C̄)→(C)
CPL   bit                   ;(bit̄)→(bit)
```

这组指令是把位累加器 C 或指定位 bit 的状态取反，操作如图 3.31 所示，功能上相当于非门。

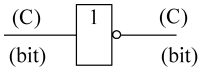


图 3.31 取反指令的功能

3. 位逻辑运算指令

1）位逻辑与运算指令

```
ANL   C,bi                  ;(C)∧(bit)→(C)
```

该指令把位累加器 C 与指定位 bit 的状态相与，运算结果存储在 C 中。指令操作如图 3.32 所示。

```
ANL    C,/bit              ;(C)∧(bit)→(C)
```

该指令把位累加器 C 与指定位 bit 的非状态相与，运算结果存储在 C 中。指令中斜线“/”表示对指定位 bit 的状态逻辑取反。指令的操作形式如图 3.33 所示。值得注意的是，指令执行并不改变指定位 bit 的状态。

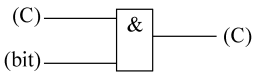


图 3.32　“ANL　C,bit”指令的功能

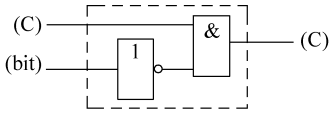


图 3.33　“ANL　C,/bit”指令的功能

在与运算指令中，只有 C 能作为该指令的目的操作数，两个位的状态不能直接相与。

2）位逻辑或运算指令

```
ORL    C,bit               ;(C) ∧(bit)→(C)
```

该指令把 C 与指定位 bit 的状态相或，运算结果存储在 C 中。指令操作如图 3.34 所示。

```
ORL    C,/bit              ;(C)∧( / bit)→(C)
```

该指令把 C 与指定位 bit 的非状态相或，运算结果存储在位累加器 C 中，指令的执行不改变指定位 bit 的状态。指令操作如图 3.35 所示。

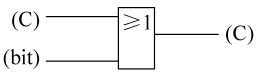


图 3.34　“ORL　C,bit”指令的功能

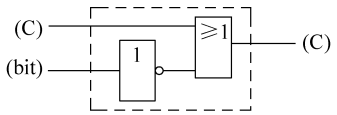


图 3.35　“ORL　C,/bit”指令的功能

在或运算指令中，同样只有 C 能作为目的操作数，两个位的状态是不能直接相或的。

例 3.39　已知逻辑表达式：$Q=U(V+W)+\overline{X\bar{Y}}+\bar{Z}$，设 U 为 P1.1，V 为 P1.2，W 为 P1.3，X 为 27H.1，Y 为 27H.0，Z 为 TF0，Q 为 P1.5。采用位操作指令实现该逻辑表达式。

根据逻辑表达式可得图 3.36 所示的电路，它也是程序设计的框图，框图设计时考虑了指令和逻辑门电路的关系。程序如下：

```
MOV    C,   P1.2           ;取 V
ORL    C,   P1.3           ;(V + W)
ANL    C,   P1.1           ;U(V + W)
MOV    20H.0,   C          ;暂存中间结果 U(V + W)于 20H 单元的第 0 位
MOV    C,   27H.1          ;取 X
ANL    C,   /27H.0         ;X·Ȳ
CPL    C                   ;(X·Ȳ)̄
ORL    C,   /TF0           ;(X·Ȳ)̄ + Z̄
ORL    C,   20H.0          ;U(V + W) + (X·Ȳ)̄ + Z̄
MOV    P1.5,   C           ;输出
```

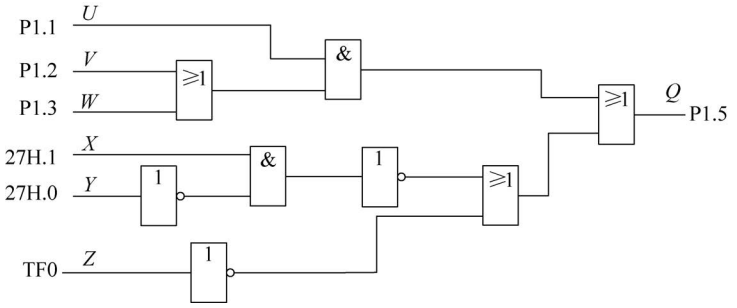


图 3.36　程序设计框图

3.3.6　控制转移指令

在工程应用中，自上而下的顺序模式程序只能实现一些简单的、用途有限的功能；通常程序总是伴随着逻辑判断，由判别结果决定下一步做什么。逻辑判断有两种结果：条件成立或条件不成立，这样，程序就有两种执行顺序。另外，为了实现某种意图，在程序中需要强制 CPU 转移到指定的模块去执行程序，改变程序执行的顺序。改变程序执行顺序是由控制转移指令实现的。MCS-51 单片机提供了丰富的控制转移指令，包括无条件转移指令、条件转移指令，以及子程序调用及返回指令。

1. 无条件转移类指令

CPU 在执行程序的过程中，碰到该类型指令将“无条件”地根据指令的类型改变 PC 的内容，从而实现转移。共有 4 种不同类型，分别叙述如下。

1）转移指令

```
AJMP   addr11                ;addr11——反映在指令代码中的 11 位地址
```

指令代码为两字节，CPU 执行过程如下：

（1）CPU 取指令：(PC)＋2→(PC)；

（2）执行指令：获取目标地址并转移，$(PC)_{15\sim11}$ 作为目标地址的高 5 位，addr11→$(PC)_{10\sim0}$ 作为目标地址的低 11 位，即将 16 位目标地址送给 PC，程序转移到目标地址处执行。

该指令在指令代码中仅提供 11 位转移地址，因此，CPU 执行程序的转移范围为本条指令上下 2KB 的空间。CPU 执行程序时碰到该指令会立即转移到目标地址处。在程序中，该指令的使用方式：

```
AJMP   LABEL
```

LABEL 在编程时指定目标标号或目标地址，要求程序无条件地转移到 LABEL 处。

2）长转移指令

```
LJMP   addr16                ;addr16——反映在指令代码中的 16 位地址
```

指令代码为三字节，CPU 执行过程如下：

（1）CPU 取指令：(PC)＋3→(PC)；

（2）执行指令：获取目标地址，addr16→(PC)，将指令中给定的 16 位目标地址 addr16 送给 PC，程序转移到目标地址 addr16 处执行。

CPU 执行程序时，碰到该指令立即转移到指令指定的目标地址处执行程序。该指令提供 16 位转移地址，转移范围为 64KB。该指令直接指出了要转移到的 16 位目标地址，因此，CPU 可以转移到程序存储器 64KB 地址空间的任何单元。在程序中该指令的使用方式：

```
LJMP  LABEL
```

程序转移到标号 LABEL 处。

3）短转移指令

```
SJMP  rel                    ;rel——反映在指令代码中的转移相对偏移量,补码
```

指令代码为两字节，CPU 执行过程如下：

（1）CPU 取指令：(PC)＋2→(PC)；

（2）执行指令：获取目标地址并转移，(PC)＋rel→(PC) 作为目标地址送给 PC，程序转移到目标地址处执行。

指令代码中给定的转移相对偏移量 rel 为 8 位二进制补码，因此，该指令的转移范围是本条指令上方最远 128B，下方最远 127B。在程序中该指令的使用方式：

```
SJMP  LABEL
```

程序转移到指定的标号 LABEL 处。

上述 3 种指令的功能是相同的，其功能与高级语言中的 GOTO 语句类似。3 条指令的区别在于它们的转移范围：LJMP 指令的转移范围为 64KB，可以转移到程序存储器的任何地方，AJMP 指令的转移范围为该指令上方和下方 2KB，而 SJMP 指令为本指令上方 128B、下方 127B。编程时只需在指令的助记符 LJMP、AJMP、SJMP 之后以标号或 16 位地址的形式指定目标地址，汇编系统在汇编时会把正确的目标地址格式添加在指令的指令代码中。如果给出的目标地址超出所用指令的转移范围，汇编系统会提示错误信息，用较大转移范围的指令替换原来的指令即可。

值得一提的是，目前大多数汇编系统支持“JMP LABEL”形式，它不是 MCS-51 的标准指令，是 LJMP/AJMP/SJMP 的一般形式，汇编系统汇编时会按照默认的指令方式（3 种指令中的一种）汇编程序，并把正确的目标地址格式添加在指令的指令代码中。

4）间接转移指令

```
JMP  @A + DPTR
```

指令代码为 1 字节，CPU 执行过程如下：

（1）CPU 取指令：(PC)＋1→(PC)；

（2）执行指令：获取目标地址并转移，(DPTR)＋(A)→(PC) 作为目标地址送给 PC，程序转移到目标地址处执行。

该指令转移到的目标地址是由累加器 A（8 位无符号数）和数据指针 DPTR（16 位无符号数）的内容相加形成的。它可以根据运算结果（累加器 A 的内容）的不同，把程序转移到不同的位置，执行不同功能的程序，具有多分支转移功能，即散转功能，又叫散转指令。该指令执行时，不改变累加器 A 和 DPTR 的内容，也不影响任何标志位。

例 3.40　设应用系统的操作键盘上定义了 5 个功能键：FUN0～FUN4，它们对应的键值分别为 00H～04H，FUN0 按下时，执行处理程序 P_FUN0，FUN1 按下时，执行处理程序 P_FUN1，以此类推，如图 3.37 所示。

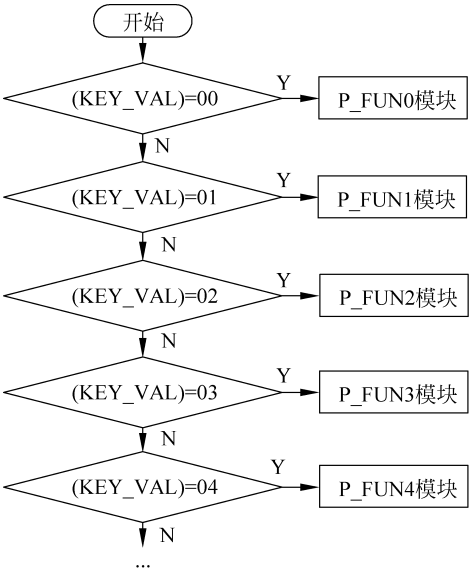


图 3.37　例 3.40 的执行过程流程图

“JMP @ A + DPTR”具有散转功能，类似于高级语言的 CASE 或 SWITCH 语句，它可以根据变量或表达式的值，使程序转移到指定的标号处。设键按下后，键值存放在 KEY_VAL 单元，采用 AJMP 指令使程序转移到指定处理程序，为了便于分析，把指令代码与程序一起给出，程序从 0200H 单元开始存放。程序如下：

```
【地址】【指令代码】【标号】    【指令】                【注释】
0200H   90 02 27              MOV DPTR,#JMP_TABLE     ;设置转移表首地址
0203H   E5 40                 MOV  A, KEY_VAL         ;KEY_VAL 为 40H
0205H   23                    RL   A                  ;AJMP 指令代码为双字节,因
                                                      ;此键值乘以 2
0206H   73                    JMP @ A + DPTR
0207H   61 00       JMP_TABLE: AJMP  P_FUN0           ;P_FUN0 模块入口地址为 0300H
0209H   81 00                 AJMP  P_FUN1            ;P_FUN1 模块入口地址为 0400H
020BH   A1 00                 AJMP  P_FUN2            ;P_FUN2 模块入口地址为 0500H
020DH   C1 00                 AJMP  P_FUN3            ;P_FUN3 模块入口地址为 0600H
020FH   E1 00                 AJMP  P_FUN4            ;P_FUN4 模块入口地址为 0700H
```

上述程序在运行过程中动态地确定程序转移的分支。设计程序时，事先把需要散转的分支建成一个由转移指令组成的表格，分支的选择由键值确定。由于 AJMP 指令代码为两字节，在检索分支时，把键值乘以 2 使程序能够正确地转移到指定的分支。

也可以采用 LJMP 指令使分支转移的范围更大一些，由于 LJMP 指令为 3 字节，计算转移目标地址时，键值应乘以 3。程序如下：

```
                MOV  DPTR, #JMP_TABLE      ;设置转移表首地址
                MOV  A, KEY_VAL            ;KEY_VAL 内容为键值
                RL   A
                ADD  A, KEY_VAL            ;LJMP 指令代码为 3 字节,键值乘以 3
                JMP  @ A + DPTR
JMP_TABLE:      LJMP P_FUN0                ;P_FUN0 模块入口
                LJMP P_FUN1                ;P_FUN1 模块入口
                LJMP P_FUN2                ;P_FUN2 模块入口
                LJMP P_FUN3                ;P_FUN3 模块入口
                LJMP P_FUN4                ;P_FUN4 模块入口
```

2. 条件转移指令

CPU 执行条件转移指令时，当满足给定条件时，程序转移到目的地址处执行；否则，顺序执行转移指令的下一条指令。

1）以累加器 A 的内容为条件的转移指令。

（1）以累加器 A 的内容等于零为条件的转移指令。

```
JZ  rel
```

指令代码为 2 字节，CPU 执行过程如下：

① 取指令：(PC)＋2→(PC)；

② 执行并获取目标地址：当(A)＝0 时，(PC)＋rel→(PC)，转移；当(A)≠0 时，顺序执行下一条指令。

CPU 执行过程流程图如图 3.38(a)所示。编写程序时，该指令的使用方式：

```
JZ  LABEL
```

编程使用方式的流程图如图 3.38(b)所示。

（2）以累加器 A 内容不等于零为条件的转移指令。

```
JNZ  rel
```

指令代码为 2 字节，CPU 的执行过程如下：

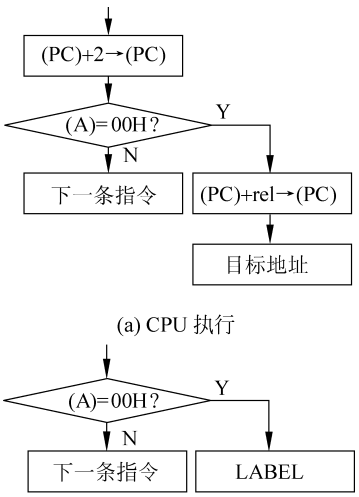
① 取指令：(PC)＋2→(PC)；

② 执行并获取目标地址：当(A)≠0 时，(PC)＋rel→(PC)，转移；当时(A)＝0，顺序执行下一条指令。

CPU 执行过程的流程图如图 3.39(a)所示。编写程序时，它的使用方式：

```
JNZ  LABEL
```

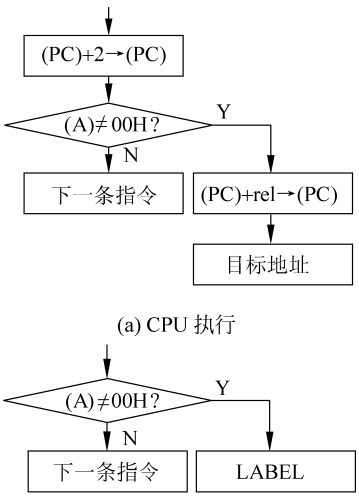
编程使用方式的流程图如图 3.39(b)所示。



(a) CPU 执行

(b) 编程使用

图 3.38　JZ 指令的流程图



(a) CPU 执行

(b) 编程使用

图 3.39　JNZ 指令的流程图

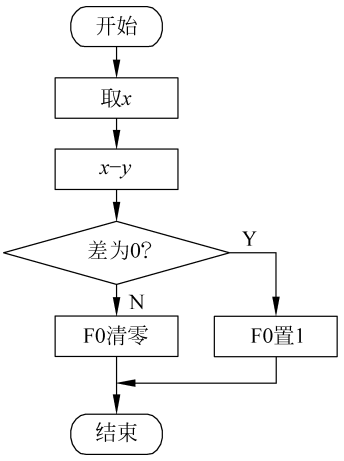


图 3.40　比较两个数是否相等的程序流程图

例 3.41　设无符号数 x 存放于 20H 单元，y 存放于 21H 单元，比较两个数 x、y 是否相等，若相等置标志位 F0 位为 1，否则，F0 清零。

解：比较两个数是否相等，最简单的方法是把两个数相减，若差为 0，则二者相等；否则，不相等。上述方法的程序流程图见图 3.40，程序如下：

```
        MOV  A,   20H        ;取 x
        CLR  C
        SUBB A,   21H        ;x - y,产生比较条件
        JZ   EQUX            ;x - y = 0,相等
        CLR  F0              ;不相等,清标志位 F0(PSW.5)
        RET                  ;返回
EQUX:   SETB F0              ;相等,标志位 F0 置 1
        RET                  ;返回
```

例 3.41 也可以用 JNZ 指令实现，此时的判断条件是差不为 0。这两种指令的判别对象为 A 的内容，判断条件是 A 的内容为 0 或不为 0，程序设计时，建立判断条件的途径如下。

（1）数据传送，累加器 A 作为目的操作数的指令。

（2）算术运算，加、减、乘、除指令。

（3）逻辑运算，与累加器 A 有关的与、或、异或指令。

（4）移位指令，与累加器 A 有关的移位指令。

2）比较转移指令

（1）累加器 A 与指定单元比较的转移指令。

```
CJNE  A,direct,rel
```

该指令的指令代码为 3 字节，CPU 执行过程：

① 取指令：(PC)＋3→(PC)；

② 执行并获取目标地址：若(A)＞(direct)，则(PC)＋rel→(PC)，且 0→(Cy)；若(A)＜(direct)，则(PC)＋rel→PC，且 1→(Cy)；若(A)＝(direct)，则顺序执行该指令的下一条指令，且 0→(Cy)。CPU 执行过程流程图如图 3.41(a)所示。

编写程序时，它的使用方式：

```
CJNE  A,direct,LABEL
```

编程使用方式的流程图如图 3.41(b)所示。

这条指令支持累加器 A 与指定单元的内容之间的比较，若二者不相等则转移，并且通过标志位 Cy 的状态指出了两个操作数的大小信息；如果二者相等，则顺序执行程序。比较

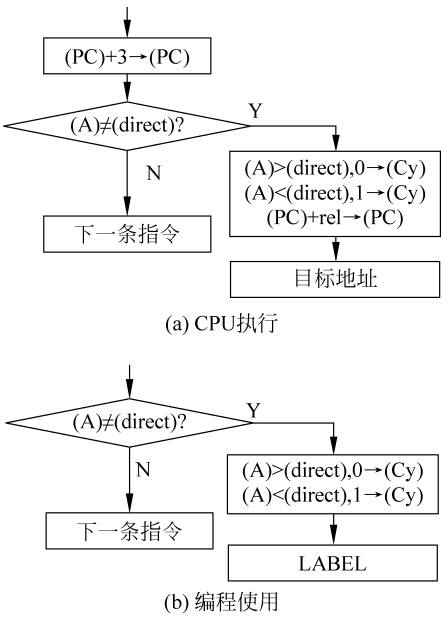


图 3.41　“CJNE　A,direct,ret”指令的流程图

的结果不传送，即不影响累加器 A 和指定单元的内容，但是指令执行时影响标志位 Cy。

重新用 CJNE 指令实现例 3.41，程序如下：

```
        MOV  A,   20H
        CJNE A,   21H,   NEQ        ;比较 x 和 y,不相等则转移到 NEQ
        SETB F0
        RET
NEQ:    CLR  F0
        RET
```

下列 3 种指令与前面介绍的比较指令的功能相似，不同的是它们支持工作寄存器或存储单元与给定的常数进行比较。

(2) 累加器 A 的内容与常数比较的转移指令。

```
CJNE  A,#data,rel
```

该指令把累加器 A 的内容与给定的 8 位二进制常数比较，指令代码为 3 字节，CPU 执行过程为：

① 取指令：(PC)＋3→(PC)；

② 执行并获取目标地址：

若(A)＞data，则(PC)＋rel→(PC)，且 0→(Cy)；

若(A)＜data，则(PC)＋rel→(PC)，且 1→(Cy)；

若(A)＝data，则顺序执行，且 0→(Cy)。

在程序中使用方式：

```
CJNE  A,#data,LABEL
```

(3) 工作寄存器内容与常数比较的转移指令。

```
CJNE  Rn,#data,rel
```

该指令把工作寄存器的内容与给定的 8 位二进制常数比较，指令代码为 3 字节，CPU 执行过程：

① 取指令：(PC)＋3→(PC)；

② 执行并获取目标地址：

若(Rn)＞data，则(PC)＋rel→(PC)，且 0→(Cy)；

若(Rn)＜data，则(PC)＋rel→(PC)，且 1→(Cy)；

若(Rn)＝data，则顺序执行，且 0→(Cy)。

在程序中，该指令的使用方式：

```
CJNE  Rn,#data,LABEL
```

(4) 指定单元内容与常数比较的转移指令。

```
CJNE  @Ri,#data,rel        ;i=0,1
```

该指令把地址寄存器指定的内部 RAM 单元的内容与给定的 8 位二进制数比较。指令代码为 3 字节，CPU 执行过程：

① 取指令：(PC)＋3→(PC)；

② 执行并获取目标地址：

若[(Ri)]>data，则(PC)+rel→(PC)，且 0→(Cy)；
若[(Ri)]<data，则(PC)+rel→(PC)，且 1→(Cy)；
若[(Ri)]=data，则顺序执行，且 0→(Cy)。
在程序中的使用方式：

```
CJNE  @Ri,#data,LABEL        ;i=0,1
```

例 3.42 从内部 RAM 的 30H 单元开始连续存储有 20 个无符号 8 位二进制数。统计这一组数据中 00H 的个数，结果存入 60H 单元。

CJNE 类比较指令实现两个无符号的 8 位二进制数的比较，分析题目要求，得到的程序流程图如图 3.42 所示。程序如下：

```
        MOV  A,   #20            ;数据长度
        MOV  60H,  #00H          ;统计个数清零
        MOV  R0,   #30H          ;设置数据块首地址
NEXT:   CJNE @R0, #00H, GOON    ;逐个取单元并比较
        INC  60H                 ;统计单元内容为 00H 的个数
GOON:   INC  R0                  ;修改地址
        DEC  A                   ;数据长度减 1
        JNZ  NEXT                ;比较完否?
        RET
```

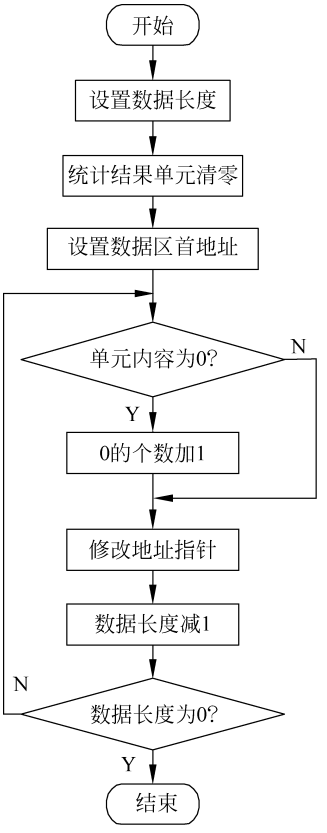


图 3.42 例 3.42 的程序流程图

3）以进位位 Cy 状态为判别条件的转移指令

（1）以 Cy 状态是 1 为判别条件的转移指令。

```
JC  rel
```

该指令的指令代码为 2 字节，CPU 执行过程：

① 取指令：(PC)+2→(PC)；

② 执行并获取目标地址：若(Cy)=1，则(PC)+rel→(PC)；若(Cy)=0，则顺序向下执行。

CPU 执行过程流程图如图 3.43(a)所示。

该指令在程序中的使用方式：

```
JC  LABEL
```

编程使用方式的流程图如图 3.43(b)所示。

（2）以 Cy 状态是 0 为判别条件的转移指令。

```
JNC  rel
```

该指令的指令代码为 2 字节，CPU 执行过程：

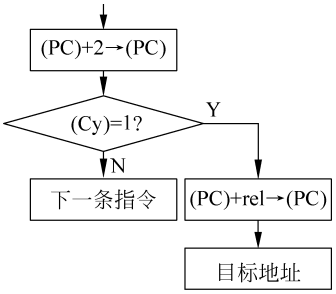
① 取指令：(PC)+2→(PC)；

② 执行并获取目标地址：

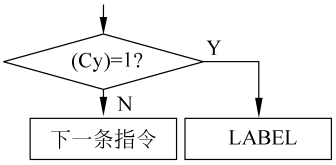
若(Cy)=0，则(PC)+rel→(PC)；

若(Cy)=1，则顺序向下执行。

CPU 执行过程流程图如图 3.44(a)所示。

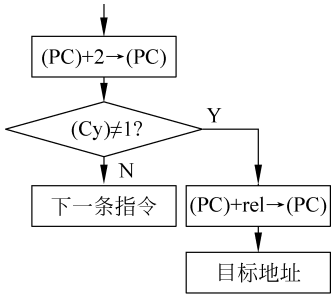


(a) CPU执行

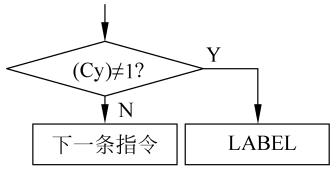


(b) 编程使用

图 3.43　JC 指令的流程图



(a) CPU执行



(b) 编程使用

图 3.44　JNC 指令的流程图

该指令在程序中的使用方式：

```
JNC   LABEL
```

编程使用方式的流程图如图 3.44(b) 所示。

以上两种以进位标志位 Cy 的状态为判断条件，满足条件则转移到目标地址处。在程序设计时，建立判断条件的途径如下。

(1) 位传送：MOV C，bit。

(2) 算术运算(加、减法指令)：ADD/ADDC/SUBB。

(3) 带进位移位的指令：RLC A，RRC A。

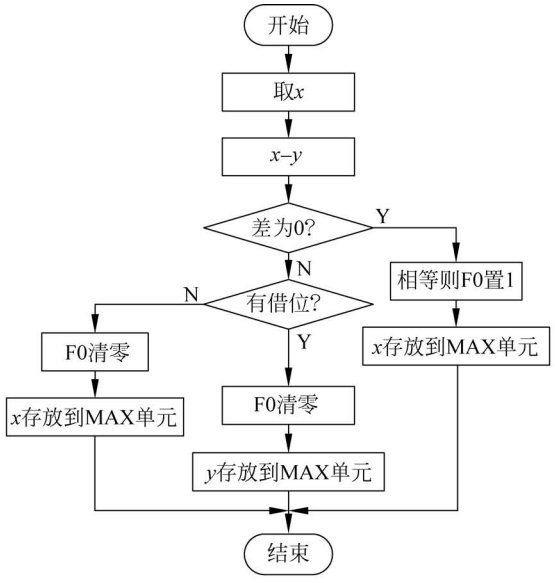
(4) 位逻辑运算：与、或运算。

例 3.43　比较两个 8 位二进制无符号数 x、y 的大小，并将大数存放在 MAX 单元，若相等，置标志位 F0 为 1；否则，F0 清零。

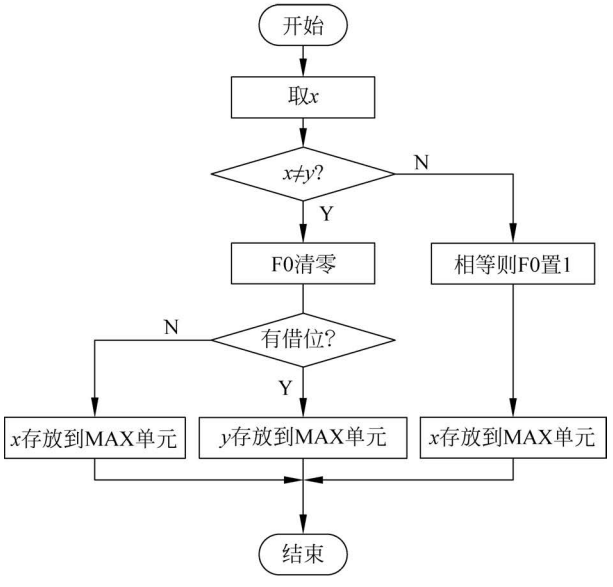
根据题意，可采用两种方法比较 x、y 的大小，程序流程图如图 3.45 所示。设 x 和 y 分别存储在 20H 和 21H 单元。

方法一：采用减法比较大小。

```
        MOV   A,   20H              ;取 x
        CLR   C
        SUBB  A,   21H              ;减法
        JZ    EQUXY                 ;差为 0,相等
        CLR   F0                    ;不相等
        JNC   GRT                   ;没有借位,x 大于 y
        MOV   MAX,  21H             ;y 大于 x,存大数
        RET                         ;返回
EQUXY:  SETB  F0                    ;x 和 y 相等
GRT:    MOV   MAX,  20H             ;存大数
        RET
```



(a) 方法一



(b) 方法二

图 3.45　程序流程图

方法二：采用比较指令。

```
        MOV   A,   20H              ;取 x
        CJNE  A,   21H,   NEQ       ;比较 x 和 y 是否相等
        SETB  F0                    ;相等
        MOV   MAX, A                ;存大数
        RET                         ;返回
NEQ:    CLR   F0                    ;不相等，F0 清零
        JC    LESS                  ;(Cy) = 1,y 大于 x
        MOV   MAX, A                ;存大数
```

```
        RET
LESS:   MOV   MAX,   21H                ;y 大于 x
        RET
```

4）以位状态为判别条件的转移指令

（1）以位状态为 1 作为判别条件转移指令。

```
JB   bit,rel
```

该指令的指令代码为 3 字节，CPU 执行过程：

① 取指令：(PC)＋3→(PC)；

② 执行并获取目标地址：

若(bit)＝1，则(PC)＋rel→(PC)；

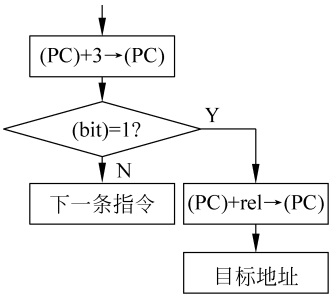
若(bit)＝0，则顺序向下执行。

CPU 执行过程流程图如图 3.46(a)所示。

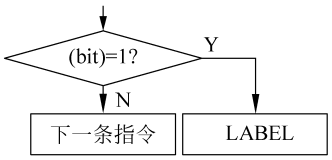
该指令在程序中的使用方式：

```
JB   bit, LABEL
```

编程使用方式的流程图如图 3.46(b)所示。



(a) CPU执行

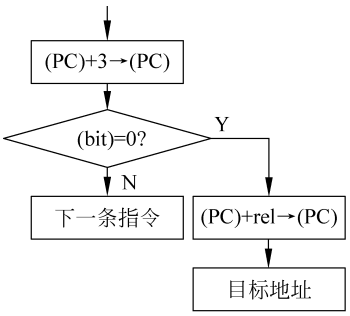


(b) 编程使用

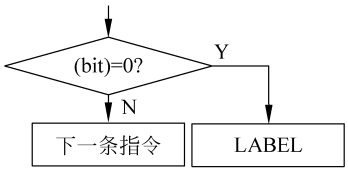
图 3.46　JB 指令的流程图

（2）以位状态为 0 作为判别条件转移指令。

```
JNB   bit,rel
```



(a) CPU执行



(b) 编程使用

图 3.47　JNB 指令的流程图

该指令的指令代码为 3 字节，CPU 执行过程：

① 取指令：(PC)＋3→(PC)；

② 执行并获取目标地址：

若(bit)＝0，则(PC)＋rel→(PC)；

若(bit)＝1，则顺序向下执行。

CPU 执行过程流程图如图 3.47(a)所示。

该指令在程序中的使用方式：

```
JNB   bit, LABEL
```

编程使用方式的流程图如图 3.47(b)所示。

例 3.44　利用标志位实现控制键的多重定义。单片机应用系统如图 3.48 所示。在系统运行过程中，要求按下按钮 S，电机 M 启动，再次按下它时，电机 M 停机，能够重复实现。

用 20H.7 位的状态标记电机的状态，(20H.7)为 0，电机处于停机状态，反之，电机处于开机状态。程序设计流程图如图 3.49 所示，程序如下：

```
        SETB   P1.0              ;置 P1.0 为输入口
        CLR    P1.3              ;关电机
        CLR    20H.7             ;电机为停机状态
NO_PRESS:  JB   P1.0,   NO_PRESS ;判断开关 S 是否按下
        JNB    20H.7, ON;        ;P1.0 为低电平，S 按下，电机启动
        CLR    P1.3              ;(20H.7) = 1，电机在运转，则停机
        CLR    20H.7             ;更改电机运行状态：停机
```

```
        SJMP   NO_PRESS          ;等待 S 按下启动
ON:     SETB   P1.3              ;电机启动,(P1.3) = 1,KA 得电,电机启动运行
        SETB   20H.7             ;更改电机运行状态：启动
        SJMP   NO_PRESS          ;等待 S 按下停机
```

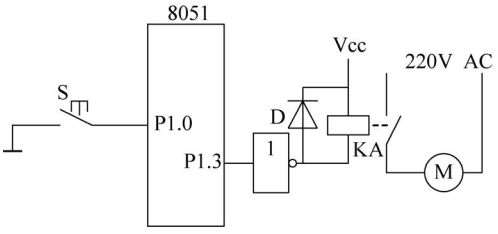


图 3.48　单片机应用系统

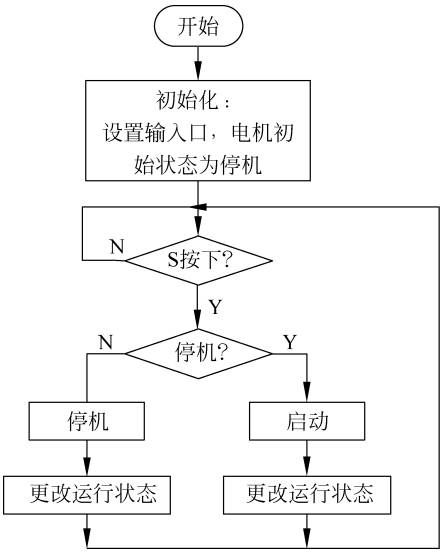


图 3.49　例 3.44 的程序设计流程图

（3）判别位状态并清零的转移指令。

```
JBC   bit,rel
```

该指令的指令代码为 3 字节，CPU 执行过程：

① 取指令：(PC)＋3→(PC)；

② 执行并获取目标地址：

若(bit)＝1，则 bit 位被清零，(PC)＋rel→(PC)；

若(bit)＝0，则顺序向下执行。

该指令执行时，位状态测试和清零一起完成；指令执行完毕，测试位 bit 的状态总为 0。CPU 执行过程流程图如图 3.50(a)所示。

该指令在程序中的使用方式：

```
JBC   bit, LABEL;
```

编程使用方式的流程图如图 3.50(b)所示。

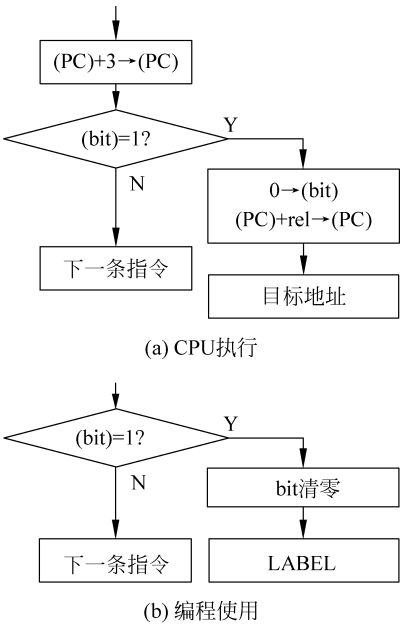


图 3.50　JBC 指令的流程图

例 3.45　已知累加器 A 的内容为 56H(01010110B)，执行下列指令序列：

```
JBC   ACC.3, LABEL1
JBC   ACC.2, LABEL2
```

程序将转移到 LABEL2 处，并且累加器 A 的内容变为 52H(01010010B)。

5）循环控制转移指令

（1）以工作寄存器内容作为循环控制变量的转移指令。

```
DJNZ   Rn,rel                          ;n = 0～7
```

该指令的指令代码为 2 字节，CPU 执行过程：

① 取指令：(PC)＋2→(PC)；

② 执行并获取目标地址：寄存器 Rn 的内容减 1，(Rn)－1→(Rn)；判断 Rn 的内容是否为 0，若(Rn)≠0，则(PC)＋rel→(PC)；若(Rn) ＝ 0，则结束循环，顺序执行。

CPU 执行过程流程图如图 3.51(a)所示。

该指令在程序中的使用方式：

```
DJNZ   Rn,LABEL
```

编程使用方式的流程图如图 3.51(b)所示。

(2) 以单元的内容作为循环控制变量的转移指令。

```
DJNZ   direct,rel
```

该指令的指令代码为 3 字节，CPU 执行过程：

① 取指令：(PC)＋3→(PC)；

② 执行并获取目标地址：

单元内容减 1，(direct)－1→(direct)；判断单元内容是否为 0：

若(direct)≠0，则(PC)＋rel→(PC)；

若(direct)＝0，则结束循环，顺序执行。

在程序中，该指令的使用方式：

```
DJNZ   direct,LABEL
```

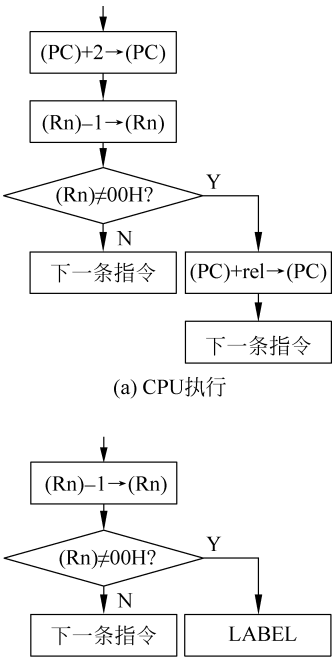


图 3.51　DJNZ 指令的流程图

除了指令代码是 3 字节以外，上述指令的执行过程和编程使用方式与前一条指令相同。这两组指令含有减 1 运算，使用时应注意循环控制变量的下溢现象。

例 3.46　把内部 RAM 从 20H 单元开始的 20 个单元清零。

解：根据题意，程序设计流程图如图 3.52 所示。程序如下：

```
        MOV   R0,   ＃20H          ;设置数据区首地址
        MOV   R5,   ＃20           ;数据个数
DO:     MOV   @R0,  ＃00H          ;单元内容清零
        INC   R0                   ;修改地址指针
        DJNZ  R5,   DO             ;循环结束否?若否,继续清零
        RET
```

例 3.47　把外部数据 RAM 中的从 ADDRESS_X 单元开始存储的 LEN 字节数据块传送到内部数据 RAM。在内部数据 RAM 中数据块从 BUFFER 单元开始存放。

解：根据题目要求，程序设计流程图如图 3.53 所示，程序如下：

```
        MOV   DPTR,   ＃ADDRESS_X      ;源数据块首地址
        MOV   R0,   ＃BUFFER           ;目的存储区首地址
        MOV   20H, ＃LEN               ;数据长度
TRANSFER:  MOVX A,   @DPTR             ;取数据
        MOV   @R0,   A                 ;存数据
        INC   DPTR                     ;修改源地址指针
```

```
INC   R0                    ;修改目的地址指针
DJNZ  20H, TRANSFER         ;传送结束
RET                         ;返回
```

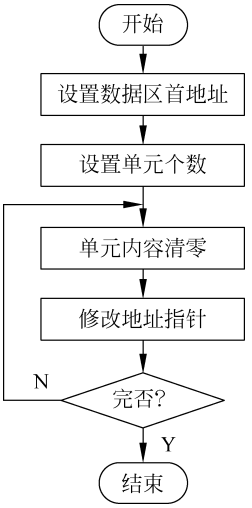


图 3.52　例 3.46 的程序设计流程图

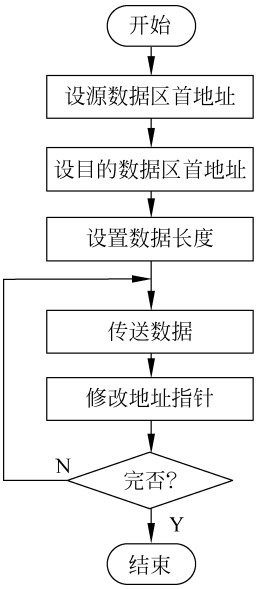


图 3.53　例 3.47 的程序设计流程图

3. 子程序调用及返回指令

在程序设计中，经常会遇到某种相同的计算和操作需要进行多次，除了参与运算的数据不同之外，其他完全相同。这种相同的程序段，如果每用一次编写一次，既麻烦，又使程序变得冗长而杂乱。冗长的程序不仅浪费了程序存储器的存储空间，而且增加了程序出错的概率。为了克服上述缺点，程序设计采用子程序(Subroutine)的概念，把程序中多次使用的程序段独立出来，单独编成一个程序，使其标准化，存储起来以备需要时调出使用，这样的程序称为子程序。与子程序相对的是主程序(Main Routine)，它是使用子程序的程序。主程序使用子程序称为调用。

主程序调用子程序是通过调用指令实现的。CPU 在执行主程序的过程中，遇到子程序调用指令，它将转移去执行子程序，当子程序执行结束后，再返回主程序，从子程序调用指令的下一条指令开始继续向下执行。由子程序返回到主程序是通过子程序中的一条指令——返回指令实现的。主程序调用和子程序返回过程如图 3.54 所示。

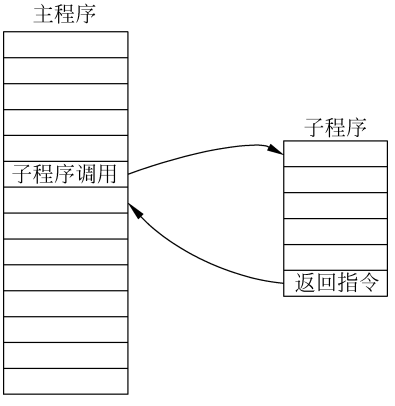


图 3.54　主程序调用和子程序返回过程

为了使 CPU 能够正确地返回到主程序中的子程序调用指令的下一条指令，在 CPU 调用子程序之前，必须把调用指令的下一条指令的地址保存起来，这个地址为返回地址，被保护在堆栈中。子程序执行结束时，通过返回指令把返回地址重新赋给 PC，这样 CPU 将执行子程序调用指令的下一条指令。

下面介绍 MCS-51 单片机的子程序调用及返回指令。

1）调用指令

（1）长调用指令。

```
LCALL   addr16
```

该指令的指令代码为 3 字节，CPU 执行过程：

① 取指令：(PC)＋3→(PC)；

② 保护返回地址：

(SP)＋1→(SP)，$(PC)_{0\sim7}$→[(SP)]；

(SP)＋1→(SP)，$(PC)_{8\sim15}$→[(SP)]。

取子程序入口地址，调用子程序：addr16→(PC)。

由于指令给出的是 16 位地址，该指令转移范围为整个程序存储器空间，即 64KB。在程序中，该指令的使用方式：

```
LCALL   SUBROUTINE
```

标号 SUBROUTINE 是子程序名，或者是子程序的入口地址，一般是指主程序转入子程序时要执行的第一条指令所在单元的地址。

（2）短调用指令。

```
ACALL   addr11
```

该指令的指令代码为 2 字节，CPU 执行过程：

① 取指令：(PC)＋2→(PC)；

② 保护返回地址：

(SP)＋1→(SP)，$(PC)_{0\sim7}$→[(SP)]；

(SP)＋1→(SP)，$(PC)_{8\sim15}$→[(SP)]。

获取子程序入口地址：addr11→$(PC)_{0\sim10}$，$(PC)_{11\sim15}$ 不变，构成子程序的入口地址，调用子程序。

在 ACALL 的指令代码中给出 11 位地址，该指令的转移范围为本条指令上下 2KB。在程序中，该指令的使用方式：

```
ACALL   SUBROUTINE
```

子程序调用指令 ACALL，LCALL 在改变 PC 内容的方式上与转移指令 AJMP、LJMP 是一样的，因此也可分别称其为短调用和长调用。它们的区别在于指令 ACALL、LCALL 在实现调用前，先把下一条指令的地址推入堆栈保存，以便执行子程序返回指令 RET 时能找到返回地址，实现正确返回。而转移指令 AJMP、LJMP 指令不需要保护返回地址。

值得一提的是，目前大多数汇编器支持“CALL　SUBROUTINE”形式，它不是 MCS-51 的标准指令，而是 LCALL/ACALL 的一般形式，汇编时，汇编系统将按照默认的指令方式汇编程序，并把正确的目标地址格式添加到指令的指令代码中。

2）返回指令

（1）子程序返回指令。

```
RET
```

该指令的指令代码为1字节，CPU执行过程：

① 取指令：(PC)+1→(PC)；

② 从堆栈中取返回地址：

[(SP)]→$(PC)_{8\sim15}$，(SP)−1→(SP)；

[(SP)]→$(PC)_{0\sim7}$，(SP)−1→(SP)。

返回指令的功能是：控制程序从当前执行的子程序返回到主程序本次调用指令(ACALL/LCALL)的下一条指令处。CPU执行RET的目的就是要从堆栈中取出这条指令的地址，该地址称为返回地址。

在程序中，该指令的使用方式：

```
RET
```

在程序设计时，子程序的最后一条指令必须是RET，它标志子程序结束。

子程序是为了实现一些公用功能而编写的程序。子程序具有通用性，它既可以被一个程序多次调用，也可以被多个不同的程序调用。另外，子程序可以存储在程序存储器的任何地方。主程序调用子程序时，事先应该把子程序需要的有关参数存放在约定的位置(存储单元、寄存器、可寻址位)，子程序执行时，从约定位置取得运算所需的参数，当子程序执行完毕后，将执行结果也存入事先约定的位置，返回主程序后，主程序就可以从约定位置上取得所需要的结果，这个过程称为参数传递。

为了便于调用，编写子程序时，一般应提供以下信息。

- 子程序名称，即入口地址或标号。
- 子程序功能描述。
- 输入输出参数，也称为子程序的入口条件和出口条件。
- 子程序中所用寄存器、存储单元和可寻址位。
- 子程序中所调用的其他子程序。

另外，有时还包含该子程序的调用示例。

主程序对子程序调用时，一般包括以下几个步骤：保护现场，调用子程序，恢复现场。由于主程序每次调用子程序的工作是事先安排的，根据实际情况，有时保护现场和恢复现场的步骤可以省略。

例3.48 编写内部RAM多个单元清零的子程序，并把从20H单元开始的20个单元清零。

解：根据例3.46的思路，编写一个内部RAM多个单元清零的子程序。

① 子程序入口条件：R0中存放待清零的内部RAM区首地址，R2中存放待清零的单元个数。

② 出口条件：无。

③ 子程序功能：把从固定起始单元开始的多个单元清零。

程序如下：

```
CLR_RAM:    MOV   @R0,   #00H        ;单元内容清零
            INC   R0                 ;修改地址指针
            DJNZ  R2,    CLR_RAM     ;循环结束否?若否,则继续清零
            RET
```

主程序：

```
MOV   R0,   #20H          ;设置数据区首地址
MOV   R2,   #20           ;单元个数
ACALL  CLR_RAM
RET
```

(2) 中断返回指令。

```
RETI
```

该指令的指令代码为 1 字节，CPU 执行过程：

① 取指令：(PC)+1→(PC)；

② 从堆栈中取返回地址：

[(SP)]→$(PC)_{8\sim15}$，(SP)−1→(SP)；

[(SP)]→$(PC)_{0\sim7}$，(SP)−1→(SP)。

在程序中的使用方式：

```
RETI
```

该指令专用于中断处理程序，是中断处理结束的标志。每一个中断处理程序的最后一条指令必须是 RETI 指令。RETI 指令与 RET 指令的区别在于 RETI 指令在实现中断返回的同时，重新开放中断使 CPU 能够接收同优先级的另外一个中断请求。在应用系统中不包含中断处理时，二者的作用是相同的。

4. 空操作指令

```
NOP
```

该指令的指令代码为 1 字节，CPU 执行过程：

取指令：(PC)+1→(PC)。

这是一条单字节指令，执行时间(指令周期)为 1 个机器周期(T_M)。该指令执行时，不做任何操作(即空操作)，仅将程序计数器(PC)的内容加 1，使 CPU 指向下一条指令继续执行程序。这条指令常用来产生一个机器周期的时间延迟。

例 3.49　一个能延时 1s 的软件延时子程序。假设系统的晶体振荡器频率为 6MHz。

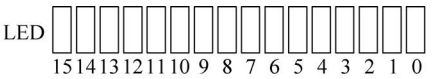
```
DELAY:  MOV R2,#250          ;指令周期为 1T_M
DELY1:  MOV R3,#250          ;1T_M
DELY2:  NOP                  ;1T_M
        NOP                  ;1T_M
        NOP                  ;1T_M
        NOP                  ;1T_M
        NOP                  ;1T_M
        NOP                  ;1T_M
        DJNZ  R3,  DELY2     ;指令周期为 2T_M
        DJNZ  R2,  DELY1     ;指令周期为 2T_M
        RET                  ;指令周期为 2T_M
```

由于晶振的频率为 6MHz，则 $T_M=2\mu s$，总延时时间为

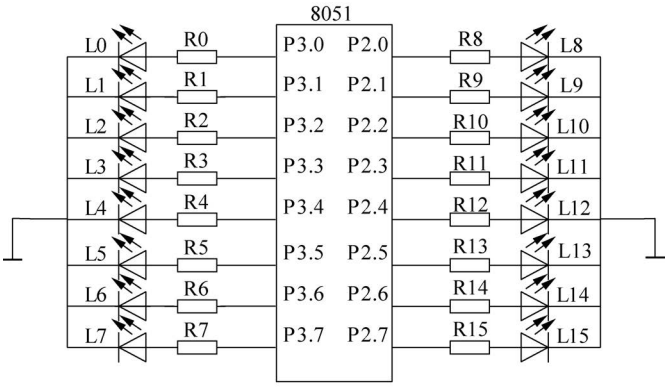
$$T=T_M+[250\times(2T_M+6\times T_M)+3T_M]\times 250+2T_M=1001ms\approx 1s$$

例 3.50　LED 阵列的灯位以图 3.55(a)的方式布置，图 3.55(b)为 LED 阵列控制电路原理图。工作时，要求 LED 从右向左逐个点亮并保持，阵列中所有的 LED 全亮后保持一段

时间后，从左向右依次逐个熄灭，如此循环。系统晶振频率为 12MHz。



(a) LED灯位布置



(b) LED阵列控制电路

图 3.55　LED 阵列控制电路原理图

根据图 3.55(b)可知，当输出口输出 1 时，LED 亮，输出 0 时，LED 熄灭。LED 阵列从右向左依次点亮时，可采用“RLC A”指令，令 Cy 为 1，每移入一次 Cy 的状态，向左点亮一个 LED，当所有 LED 全亮时，输出口 P2 和 P3 输出全为 1，即可停止左移，进入保持阶段。从左向右熄灭 LED 的实现过程与点亮相似。程序流程图见图 3.56。

用 R2 和 R3 分别存储 P3 和 P2 口的控制码，程序如下：

```
START:  MOV R2,#00H              ;初始状态,自右向左
        MOV R3,#00H
REDO:   SETB C                   ;置 1,逐个点亮
        MOV A,R2
        RLC A
        MOV R2,A
        MOV A,R3
        RLC A
        MOV R3,A                 ;移位,产生控制码
        MOV P3,R2                ;输出显示
        MOV P2,R3
        ACALL DELMS              ;延时
        XRL A,R2                 ;若 LED 全亮,(R2) = (R3) = 0FFH
        JNZ REDO                 ;没全亮,则继续
        MOV R7,#50;
HOLD:   ACALL DELMS              ;保持全亮延时,约 5s
        DJNZ R7,HOLD
REDO1:  CLR C                    ;逐个熄灭
        MOV A,R3
        RRC A
        MOV R3,A
        MOV A,R2
```

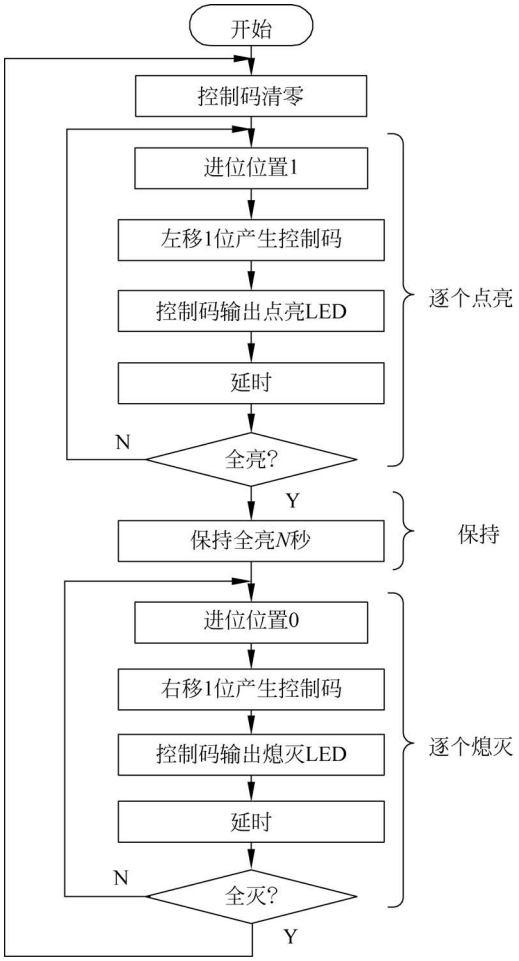


图 3.56　例 3.50 程序流程图

```
        RRC A
        MOV R2,A                ;移位产生控制码
        MOV P3,R2               ;输出显示
        MOV P2,R3
        ACALL DELMS             ;延时
        XRL A,R3                ;若 LED 熄灭,(R2) = (R3) = 00H
        JNZ REDO1               ;没全熄灭, 则继续
        LJMP START
;延时子程序:
DELMS:  MOV R5,#100
DELX0:  MOV R6,#250
D00:    NOP                     ;1T_M
        NOP                     ;1T_M
        DJNZ R6,D00             ;2T_M, 250×4μs = 1000μs = 1ms
        DJNZ R5,DELX0           ;约 100×1ms = 100ms
        RET
```

3.4 汇编语言程序设计

3.4.1 伪指令

伪指令(Pseudo Instruction)是汇编语言中起解释说明的命令，它不是单片机的指令。在集成开发环境中，伪指令向编译系统说明程序在程序存储器的哪个区域、到何处结束、变量所表示的单元地址或数值等。汇编时伪指令不会产生目标代码，不影响程序的执行。

不同的编译系统使用的伪指令种类不同，常用的有以下几种伪指令。

1. 设置起始地址伪指令 ORG

```
ORG   xxxxH
```

设置程序从程序存储器的 xxxxH 单元开始存放。在一个汇编语言源程序中，可以多次定义 ORG 伪指令，但要求规定的地址由小到大安排，各段之间地址不允许重叠。如：

```
        ORG       0100H
SUB:    MOV       R0,#30H
…
```

程序汇编后，子程序 SUB 的代码从 0100H 单元开始存放，也就是“MOV R0，#30H”指令代码的第一字节存放在程序存储器的 0100H 单元中。

2. 赋值伪指令 EQU

```
变量代号 EQU 数值
```

EQU 指令用来给变量代号赋值。在同一个源程序中，任何一个变量代号只能赋值一次。赋值以后，变量代号在整个源程序中的值是固定的，不可改变。变量代号可以表示一个单元地址或者一个立即数。EQU 指令后面的数值可以是 8 位或 16 位的二进制数，也可以是事先定义的表达式，在有的向编译系统中，数值的形式也可以为位地址。如：

```
LEN       EQU 20        ;在程序中变量 LEN 的值为 20
Xdata     EQU 4F8BH     ;在程序中变量 Xdata 的值为 4F8BH
FLAGX     EQU 20H.7     ;在程序中变量 FLAGX 表示 20H.7
```

3. 定义字节数据伪指令 DB

```
[单元地址代号:]  DB     data
```

DB 用来说明程序存储器单元的内容是一字节的常数 data，而非指令代码。单元地址代号可以省略。如：

```
ADDR1:DB 30H
```

ADDR1 单元的内容设置为 30H。DB 也可用来定义多个连续单元为常数，如：

```
ORG   1000H
DB     30H, 31H, 32H, 33H, 34H, 35H, 36H, 37H, 38H, 39H, 2EH, 0DH
```

向编译系统说明从程序存储器的 1000H 单元开始存储了 12 字节的常数。

4. 定义双字节数据伪指令 DW

```
[单元地址代号: ] DW data16
```

DW 用来定义程序存储器相邻两个单元的内容为常数。如：

```
ADDR2: DW 0FDE1H
```

编译系统把 0FDE1H 的高 8 位 0FDH 放在 ADDR2 单元，低 8 位 0E1H 放在 ADDR2＋1 单元。

```
    ORG  0400H
XTABLE:DW 1345, 2241, 34556
```

向编译系统说明常数表格 XTABLE 从 0400H 单元开始存放。

5. 位地址赋值伪指令 BIT

```
变量代号  BIT  位地址
```

BIT 用于定义有位地址的位，把位地址赋予指定的变量代号。如：

```
CS      BIT    P2.0
FLAG    BIT    20H.6
```

6. 汇编结束伪指令 END

```
END
```

END 是用来告诉编译系统，源程序到此结束。在一个程序中，只允许出现一条 END 伪指令，而且必须安排在源程序的末尾。

3.4.2　程序设计举例

1. 算术运算程序

在 MCS-51 单片机指令系统中，算术运算指令仅支持两个无符号的 8 位二进制数的运算，二进制数算术运算是按字节的方式进行的。

例 3.51　多字节二进制加法。

以 3 字节无符号二进制数为例，算法如图 3.57 所示，图中 1 个方框代表 1 个单元，Cy 为进位。最低字节运算时，若令 Cy 为 0，那么，完成 3 字节的加法运算进行了 3 次相同的加法操作，因此，可采用循环结构实现两个 3 字节数据的加法运算。

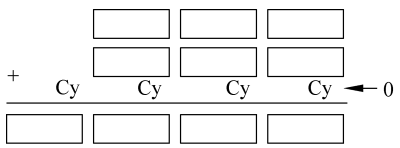


图 3.57　多字节二进制加法

(1) 采用循环结构设计的多字节二进制加法程序。

设两个数分别存在内部 RAM 的 20H 和 30H 单元开始的区域，低 8 位在前，程序如下：

```
        MOV  R0, #20H          ;被加数低 8 位存储单元地址
        MOV  R1, #30H          ;加数低 8 位存储单元地址
        MOV  R5, #03H          ;字节数
        CLR  C                 ;首次加法运算(Cy)清零
DOAD1:  MOV  A, @R0;           ;取(被)加数
        ADDC A, @R1            ;取加数
        MOV  @R0, A            ;存和
        INC  R0                ;修改存储地址
        INC  R1
        DJNZ R5, DOAD1         ;运算结束?
        CLR  A                 ;处理高 8 位运算的进位
        ADDC A, #00H
```

```
        MOV  @R0, A
        RET
```

（2）把单字节加法操作提取出来作为一个子程序。

```
; 单字节加法子程序 BIN_ADD
; 入口条件：R0 指出被加数所在单元的地址；R1 指出加数所在单元的地址
; 出口条件：R0 指出和所在单元的地址，进位在 Cy 中
BIN_ADD: MOV  A, @R0;
        ADDC A, @R1
        MOV  @R0, A
        INC  R0
        INC  R1
        RET
```

那么，3 字节二进制数加法程序如下：

```
        MOV  R0, #20H
        MOV  R1, #30H
        MOV  R5, #03H
        CLR  C
DOAD:   ACALLBIN_ADD
        DJNZ R5, DOAD
        CLR  A
        ADDC A, #00H
        MOV  @R0, A
        RET
```

例 3.52　多字节二进制减法。

多字节二进制减法与多字节二进制加法相似，图 3.58 为 3 字节二进制减法的算法。假设两个 3 字节数据分别存放在内部 RAM 的 20H 和 30H 单元开始的区域，低 8 位在前，程序如下：

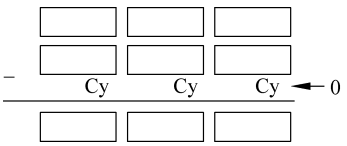


图 3.58　二进制数减法算法

（1）单字节减法子程序 BIN_SUB：

```
;入口条件：R0 指出被减数所在单元的地址；R1 指出减数所在单元的地址
;出口条件：R0 指出差所在单元的地址，借位在 Cy 中
BIN_SUB: MOV  A, @R0;
        SUBB A, @R1
        MOV  @R0, A
        INC  R0
        INC  R1
        RET
```

（2）3 字节无符号二进制数减法程序：

```
        MOV  R0, #20H
        MOV  R1, #30H
        MOV  R5, #03H
        CLR  C
DOSUB:  ACALL  BIN_SUB
        DJNZ R5, DOSUB
        RET
```

例 3.53　多位十进制数加法。

十进制数在计算机中可以采用BCD码的形式存放。采用压缩式(或紧凑形式)BCD码格式存放十进制数时，一个存储单元可以存储2位。MCS-51单片机仅支持二进制加法运算，采用ADD和ADDC指令的结果是二进制数，因此，两个以BCD码形式存储的数据，在用ADD和ADDC运算之后，必须对其运算结果进行调整。多位十进制数加法的算法与多字节二进制数加法的算法相似，如图3.59所示。6位十进制数加法程序如下：

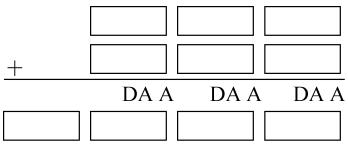


图3.59　十进制数加法算法

(1) 2位十进制数加法子程序SH_ADD：

```
;入口条件：R0指出被加数所在单元的地址；R1指出加数所在单元的地址
;出口条件：R0指出和所在单元的地址，进位在Cy中
SH_ADD:  MOV  A, @R0;
         ADDC A, @R1
         DA   A                        ;结果调整为十进制数
         MOV  @R0, A
         INC  R0
         INC  R1
         RET
```

(2) 6位十进制数加法程序：

```
         MOV  R0, #20H
         MOV  R1, #30H
         MOV  R5, #03H
         CLR  C
DOAD:    ACALLSH_ADD
         DJNZ R5, DOAD
         CLR  A
         ADDC A, #00H
         MOV  @R0, A
         RET
```

例3.54　多位十进制减法。

2位十进制数减法算法如下：X－Y＝X＋100－Y→X＋9AH－Y。把十进制减法变换成二进制减法(求十进制减数的补码)和十进制加法两步进行。多位十进制数减法也采用了同样的算法，在进行高位减法运算时考虑了低位的借位状态。设被减数存放在20H开始的内部RAM存储单元，减数存放在30H开始的存储单元，6位十进制数减法的程序如下。

(1) 2位十进制数减法子程序：

```
;入口条件：R0指出被减数所在单元的地址；R1指出减数所在单元的地址
;出口条件：R0指出差所在单元的地址，借位在Cy中
SH_SUB:  MOV  A, #9AH
         SUBB A, @R1
         ADD  A, @R0
         DA   A
         MOV  @R0, A
         INC  R0
         INC  R1
         CPL  C
         RET
```

(2) 6位十进制数减法程序：

```
        MOV  R0, #20H
        MOV  R1, #30H
        MOV  R5, #03H
        CLR  C
DOSUB:  ACALL SH_SUB
        DJNZ R5, DOSUB
        RET
```

例 3.55 多字节无符号二进制数乘法。

多字节无符号二进制数乘法算法与十进制数乘法相似。以两个 2 字节二进制数相乘为例介绍多字节数的乘法算法，如图 3.60 所示。图中被乘数为 X，其高 8 位和低 8 位分别存储在 XH 和 XL 单元，乘数为 Y，其高 8 位和低 8 位分别存储在 YH 和 YL 单元。算法分两步进行：首先，分别用乘数的高 8 位和低 8 位与被乘数相乘求出部分积，分别存储在 XYH3～XYH1 和 XYL3～XYL1 单元，乘法运算可以把例 3.25 作为乘法子程序来调用。第二步，采用加法运算求出乘积并存储在 XY4～XY1 单元。读者可参考图 3.60 的流程图编写程序。

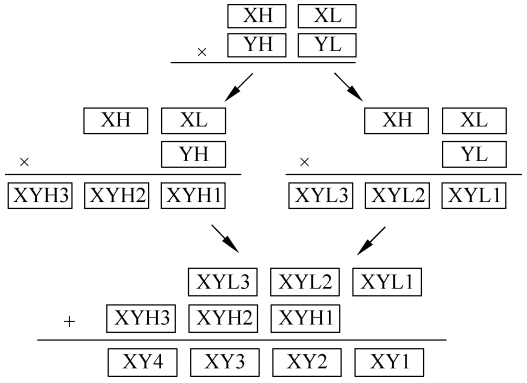


图 3.60 2 字节二进制数乘法算法

例 3.56 多字节无符号二进制数除法。

两个多字节无符号二进制数的除法是采用移位和减法运算实现的，实现过程与进行十进制数除法相似，每次进行除法运算时先试商，如果余数大于减数则商 1，否则，商 0。图 3.61 为 16 位二进制数除以 8 位二进制数的程序流程图。该算法要求被除数的高 8 位数据必须小于除数，否则，作为溢出处理，子程序把标志位 OV 的状态置为 1 并返回。

16 位无符号二进制数除以 8 位无符号二进制数子程序如下：

```
;入口条件：被除数存储在 R4、R5 中，除数存储在 R7 中
;出口条件：(OV) = 0 时，商在 R3 中；(OV) = 1 时，溢出
;子程序执行时，使用了单片机的 PSW，A，R3～R7
DIV21:  CLR  C
        MOV  A,R4
        SUBB A,R7
        JC   DV50
        SETB OV                    ;商溢出
        RET
DV50:   MOV  R6,#8                 ;(R4R5/R7 - →R3)
DV51:   MOV  A,R5
        RLC  A
        MOV  R5,A
```

```
        MOV   A,R4
        RLC   A
        MOV   R4,A
        MOV   F0,C
        CLR   C
        SUBB  A,R7
        ANL   C,/F0
        JC    DV52
        MOV   R4,A
DV52:   CPL   C
        MOV   A,R3
        RLC   A
        MOV   R3,A
        DJNZ  R6,DV51
        MOV   A,R4                    ;四舍五入
        ADD   A,R4
        JC    DV53
        SUBB  A,R7
        JC    DV54
DV53:   INC   R3
DV54:   CLR   OV
        RET
```

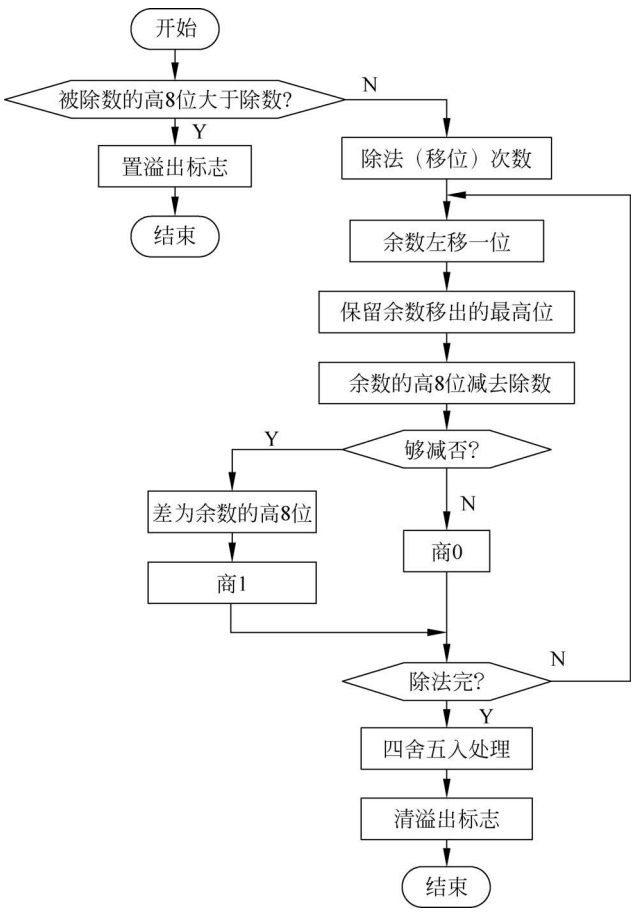


图 3.61　16 位二进制数除以 8 位二进制数的程序流程图

2. 循环程序的设计

1）循环程序的组成

循环程序由 4 部分组成：初始化部分、循环处理部分、循环控制部分和结束部分。循环结构组成见图 3.62。

（1）初始化部分用来设置循环处理之前的初始状态，如循环次数、变量初值、地址指针的设置等。

（2）循环处理部分又称为循环体，是重复执行的处理程序段，是循环程序的核心部分。

（3）循环控制部分用来控制循环继续与否。

（4）结束部分是对循环程序全部执行结束后的结果进行分析、处理和保存。

程序设计中常见的典型循环结构如图 3.62 和图 3.63 所示，前者为先处理后判断的结构，后者为先判断后处理的结构。根据循环程序也可分为单重循环和多重循环。程序设计时，若循环次数已知，可用循环次数计数器控制循环；若循环次数是未知的，则需按条件控制循环。

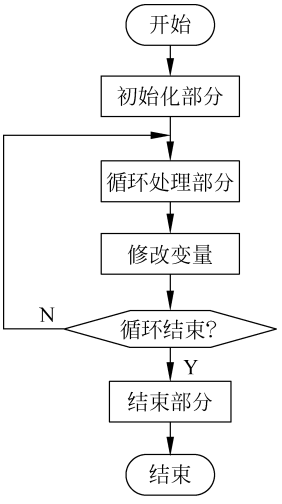


图 3.62　循环结构组成

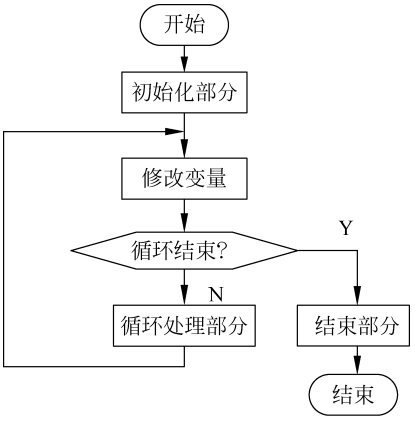


图 3.63　典型循环结构

2）循环程序设计举例

例 3.57　设单片机系统采集的 8 字节数据存储在内部 RAM 的 30H 开始的单元中，求它们的均值。

计算一组数据平均值的公式为 $\bar{x}=\sum_{i=1}^{N}x_i/N$，其中，$x_i$ 为第 i 个数据，N 为数据的个数。因此，要计算出平均值需要进行两种运算：求数据的总和及数据总和除以数据个数。

（1）求数据的总和。

设 S 为数据的总和，求多个数据总和的算法如下：

$$S=0 \qquad i=0$$

$$S=S+x_i \qquad i=1,2,\cdots,N$$

该算法的程序流程图见图 3.64。设总和 S 存放在寄存器 R5 和 R6 中，R5 存高 8 位，则求总和子程序如下：

```
SIGMA:   MOV  R1,＃30H          ;数据区首地址
         MOV  R5,＃00H          ;存放总和的单元清零
         MOV  R6,＃00H          ;
         MOV  R4,＃08H          ;数据个数
SIGMA1:  MOV  A,@R1             ;取数据
         ADD  A,R6              ;求和
         MOV  R6,A
         CLR  A
         ADDC A,R5
         MOV  R5,A
         INC  R1
         DJNZ R4,SIGMA1
         RET
```

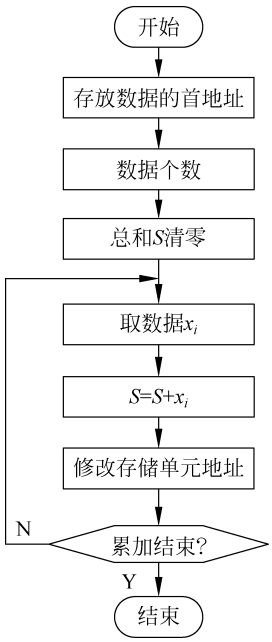


图 3.64　多个数据求总和的程序流程图

(2) 求均值。

在汇编语言设计时，除数为 2^n 时，除法可以采用移位的方法实现，这样做效率更高。$S/8=((S/2)/2)/2$，即通过调用 3 次右移除以 2 过程即可。采用移位方法求均值的程序如下：

```
MEAN:    MOV  R4,＃03H
DIV2:    MOV  A, R5          ;R5 和 R6 中存放总和,R5 存放高 8 位
         CLR  C
         RRC  A
         MOV  R5, A          ;商的高 8 位
         MOV  A, R6          ;低 8 位
         RRC  A
         MOV  R6, A          ;商的低 8 位
         DJNZ R4,DIV2
         RET
```

子程序返回时，均值存储在 R6 中。

(3) 8 个单字节数据求均值的主程序如下：

```
ACALL SIGMA
ACALL MEAN
RET
```

例 3.58　一个字符串从内部 RAM 的 40H 单元开始存放，以回车符(ASCII 码为 0DH)为结束标志，编写程序测试字符串长度。

这是一个循环次数未知的循环程序设计例题。为了测试字符串的长度，字符串中的每个字符依次与回车符(0DH)比较，如果比较不相等，则字符串长度计数器加 1，继续测试；否则，该字符为回车符，则检测结束，长度计数器的值就是字符串的长度。程序流程图如图 3.65 所示。设 R7 为字符串长度计数器，程序如下：

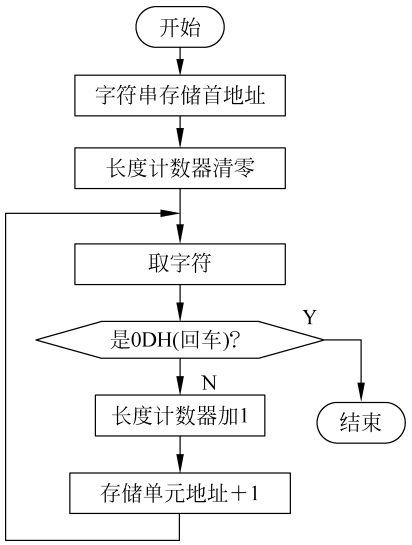


图 3.65　测试字符串长度的程序流程图

```
         MOV  R7,＃00H          ;设置长度计数器初值
         MOV  R0,＃40H;
LOOP:    MOV  A,@R0
```

```
        CJNE A, #0DH, NON
        RET                          ;如果是回车符,则字符串结束
NON:    INC  R7                      ;长度计数器加 1
        INC  R0                      ;修改存储单元地址
        SJMP LOOP
```

3. 查表程序的设计

查表程序是单片机应用系统中常用的一种程序，例如，显示输出时，利用它提取字型编码，数值运算时，利用它可以避免进行复杂的数值运算，实现插补、修正、计算、转换等功能。查表程序简单、执行速度快。查表就是根据自变量 x 在表中找出 y。在计算机中，把一组数据按照某种关系连续地存放在程序存储器中就形成了常数表，通过查表指令提取常数，设计的主要问题是建立自变量 x 与存储数据 y 的单元地址之间的关系，x 通常是 y 在表中的存储顺序。

例 3.59 设字符 0～9、A～F 的 ASCII 码存储在程序存储器中，编写子程序由 x（$0 \leqslant x \leqslant F$）查找其对应的 ASCII 码。

ASCII 码为 7 位二进制编码，一个单元也可存储一个字符的 ASCII 码。如果 ASCII 码表存放在以 ASC_TAB 单元开始的区域，则存储 ASCII 码的单元地址与 x 的关系为：ASC_TAB+x。设 x 存储在寄存器 R2 中，从子程序返回时 ASCII 码存储在 R2 中，子程序如下：

```
CHECHUP: MOV  DPTR, #ASC_TAB         ;设置表的首地址
         MOV  A,R2                   ;取 x
         MOVC A,@A + DPTR            ;查表取 ASCII 码
         MOV  R2,A                   ;存储查到的 ASCII 码
         RET
ASC_TAB: DB  30H, 31H, 32H, 33H, 34H, 35H, 36H, 37H, 38H, 39H
         DB  41H, 42H, 43H, 44H, 45H, 46H
```

例 3.60 一个 16 路的巡回检测报警系统，把每路的报警阈值（2 字节）存放在一个表格中，系统运行时，需要根据巡检回路号取出报警阈值，与采样值进行比较，以判断采样值是否超过限位。编写获取回路报警阈值的查表程序。

设巡检回路号为 x，$0 \leqslant x \leqslant F$（16 个回路），每个回路的报警限位阈值被存储在两个相邻的单元，由于 MCS-51 单片机的查表指令每次操作只能从程序存储器中取出一个单元的内容，因此，2 字节的阈值需要两次查表操作才能得到。设报警阈值存储在 LIM_TAB 开始的区域，阈值第一字节的存储单元地址为 LIM_TAB+$2x$，第二字节为 LIM_TAB+$2x$+1。设回路号存放在 R2 中，回路报警限位值存入 R3 和 R4，子程序如下：

```
CHECHUP: MOV  DPTR, #LIM_TAB         ;阈值表的首地址
         MOV  A, R2                  ;取 x
         ADD  A, R2                  ;计算 2x
         MOV  R2, A                  ;2x 暂存于 R2 中
         MOVC A, @A + DPTR           ;取阈值的第一字节
         MOV  R3, A                  ;阈值第一字节存于 R3 中
         INC  R2                     ;2x + 1
         MOV  A, R2
         MOVC A, @A + DPTR           ;取阈值的第二字节
         MOV  R4, A                  ;阈值的第二字节存于 R4
         RET
```

```
LIM_TAB:  DW    3233, 26, 1020, 2435, 423, 267, 200, 435
          DW    130, 86, 11345, 2400, 4230, 32267, 220, 352
```

例 3.61　在一个压力测量仪表中，传感器输出电压由 10 位 A/D 转换器转换为二进制数送入单片机，仪表显示器以 4 位十进制数形式显示压力值。通过实验得到了 A/D 转换值与压力的对应关系，并把它存储在单片机中。设计由 A/D 转换值获取十进制数压力值的程序。

由于 A/D 转换值 x 与压力值的对应关系已知，建立的常数表包含了 $2^{10}=1024$ 个压力值，若以压缩 BCD 码形式存储，4 位十进制数压力值需两个单元存储。若表从 PRS_TAB 单元开始存储，高 2 位存储在单元 PRS_TAB+$2x$ 中，低 2 位存储在单元 PRS_TAB+$2x+1$ 中。设将 A/D 转换值 x 存到 R2 和 R3，压力值放在 R4 和 R5 中，子程序如下：

```
CONVT:    MOV   DPTR, #PRS_TAB          ;表的首地址
          MOV   A,R3
          ADD   A,R3
          MOV   R3,A
          MOV   A,R2
          ADDC  A,R2
          MOV   R2,A                    ;计算 2x,并暂存于 R2 和 R3 中
          MOV   A,DPL
          ADD   A,R3
          MOV   DPL,A
          MOV   A,R2
          ADDC  A,DPH
          MOV   DPH,A                   ;计算 PRS_TAB + 2x,结果存于 DPTR
          CLR   A
          MOVC  A,@A + DPTR             ;取高 2 位
          MOV   R4,A                    ;存高 2 位
          INC   DPTR                    ;计算 PRS_TAB + 2x + 1,结果存于 DPTR
          CLR   A
          MOVC  A,@A + DPTR
          MOV   R5,A                    ;存低 2 位
          RET
PRS_TAB:  DW    0304H, 0420H, 0523H, …
```

4. 检索程序的设计

数据检索的任务是查找关键字，通常有两种方法：顺序检索和对分检索。本节介绍前者，对分检索可参阅相关资料。

例 3.62　设有一单字节无符号数的数据块，存储在内部 RAM 以 30H 单元为首地址的区域中。长度为 50 字节，试找出其中最小的数，并放在 20H 单元。

程序流程图如图 3.66 所示。首先把第一个数据取出作为最小数，然后依次取出其余的数据与其比较，如果小于指定的最小数，则替换；否则，继续比较。

```
          MOV   R7, #50                 ;设置比较次数
          MOV   R0, #30H                ;设置数据块首地址
          MOV   A,@R0                   ;
          MOV   20H, A                  ;取第一个数作为最小数
LOOP1:    INC   R0                      ;修改存储单元地址
          MOV   A,@R0                   ;取数
          CJNE A,20H, LOOP              ;与最小数比较
LOOP:     JNC   LOOP2                   ;若取出的数不小于最小数,则继续
          MOV   20H, A                  ;取出的数据小于最小数,则替换原来的最小数
```

```
LOOP2:   DJNZ R7,LOOP1              ;比较完否?
         RET
```

例 3.63 一个 ASCII 码字符串存放在 20H 单元开始的区域，以‘EOT’为结束标志。从其中找字符‘A’，若找到，把标志位 F0 置 1，否则，把 F0 清零。

程序流程图如图 3.67 所示。对于要检索的字符串，只要发现一个字符‘A’，则停止检索，把标志位 F0 置 1。若整个字符串没有发现‘A’，则标志位清零。程序如下：

```
         EOT  EOU 041-1             ;EOT 的 ASCII 码
INDEX:   MOV  R1,#20H
         CLR  F0
NEXT:    MOV  A,@R1
         CJNE A,#'EOT',GOON
         RET
GOON:    CJNE A, #'A', NON
         SETB F0
         RET
NON:     INC R1
         SJMP NEXT
```

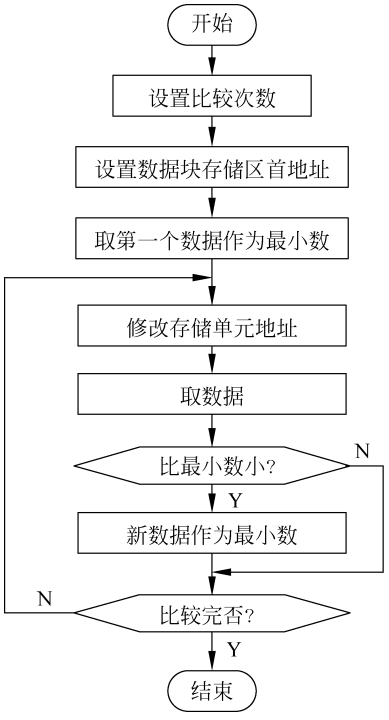


图 3.66 例 3.62 的程序流程图

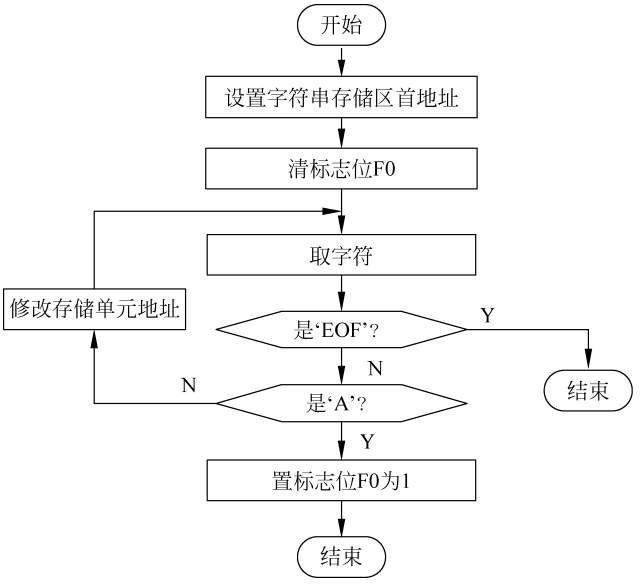


图 3.67 例 3.63 的程序流程图

5. 分支程序的设计

分支程序主要是根据判断条件的成立与否来确定程序的走向，可组成单分支结构和多分支结构。

单分支结构一般为两者选一的处理，程序的判断部分仅有两个出口。通常用条件判断指令来确定分支的出口。这类单分支选择结构有 3 种典型的形式，见图 3.68。

(1) 如果条件满足，执行程序段 2；否则，执行程序段 1，结构如图 3.68(a)所示。

（2）如果条件满足，则不执行程序段 1，仅执行程序段 2；否则，先执行程序段 1，再执行程序段 2，结构如图 3.68(b)所示。

（3）当条件不满足时，重复执行程序段 1，只有当条件满足时，才停止执行程序段 1，开始执行程序段 2，结构如图 3.68(c)所示。

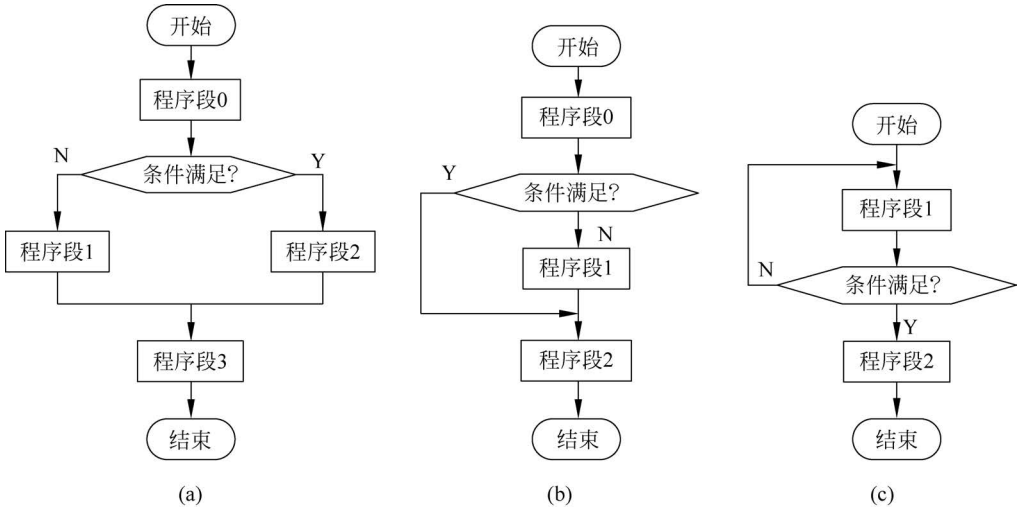


图 3.68　单分支选择结构

多分支选择结构是指程序的判别部分有两个以上的出口流向，如图 3.69 所示。

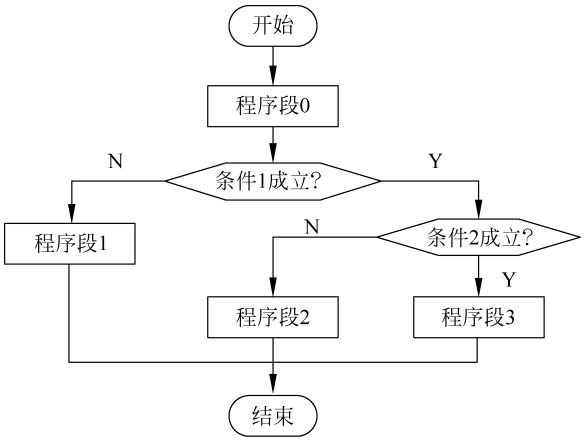


图 3.69　多分支选择结构

例 3.64　x 和 y 为两个带符号单字节数据，以原码方式存放，编写程序求它们的乘积。

MCS-51 单片机的乘法指令支持两个 8 位无符号二进制数相乘，两个带符号二进制数相乘的方法程序流程图如图 3.70 所示，若符号相同，乘积符号为正，数值为两个数绝对值之积；若符号相异，乘积符号为负，数值为两个数绝对值之积。

设 x 和 y 分别存放在 40H 和 41H 单元，乘积存放的 R4、R3 中，程序如下：

```
MOV   A,40H                ;取 x
XRL   A,41H                ;符号运算
JB    ACC.7,DIFF           ;符号相异,转移
MOV   A,40H
```

```
        ANL   A,#01111111B          ;x取绝对值
        MOV   B,41H
        ANL   B,#01111111B          ;y取绝对值
        MUL   AB
        MOV   R3,A
        MOV   R4,B                  ;存乘积
        RET
DIFF:   MOV   A,40H                 ;
        ANL   A,#01111111B          ;x取绝对值
        MOV   B,41H                 ;
        ANL   B,#01111111B          ;y取绝对值
        MUL   AB
        MOV   R3,A                  ;存乘积的低8位
        ORL   B,#10000000B          ;符号相异,乘积符号为负,置符号位为1
        MOV   R4,B                  ;存乘积的高8位
        RET
```

例 3.65 设变量 x 存放在内部 RAM 的 30H 单元中，求解下列函数式，并将 y 存入 40H 单元。

$$y=\begin{cases} x-1, & x<10 \\ 0, & 10\leqslant x<100 \\ x+1, & x\geqslant 100 \end{cases}$$

程序流程图如图 3.71 所示。程序如下：

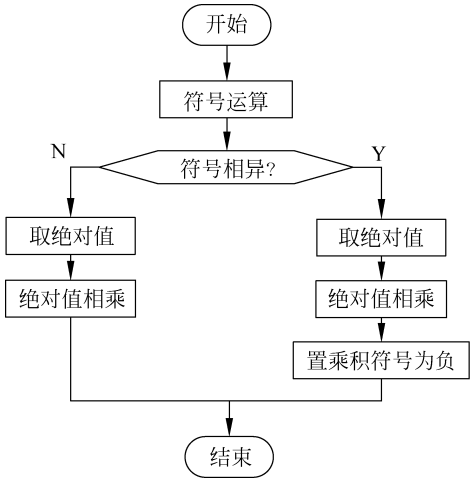


图 3.70 带符号二进制乘法的程序流程图

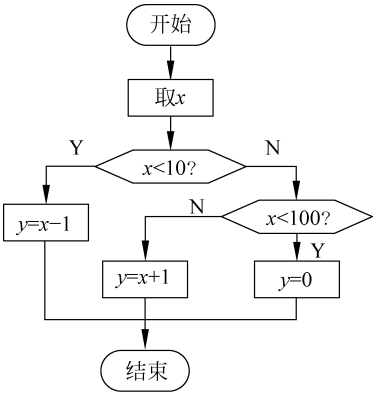


图 3.71 例 3.65 的程序设计框图

```
        MOV   A,30H             ;取x
        CLR C
        SUBB A,#10
        JNC   GT10              ;x大于或等于10,转移
        MOV   A,30H
        DEC   A                 ;x-1
        MOV   40H,A
        RET
GT10:   MOV   A,30H
        SUBB A,#100
```

```
        JC   LS100          ;x 小于 100,转移
        MOV  A,30H
        INC  A
        MOV  40H,A          ;x + 1
        RET
LS100:  MOV  40H, #00
        RET
```

6. 码制转换程序的设计

码制转换程序是单片机应用系统常用程序，如 CPU 计算、存储是采用二进制形式，而人机界面常采用十进制，需要码制转换；设备之间交换信息有时采用 ASCII 码，CPU 处理时也需要转换。本节主要介绍常用的不同进制数之间的转换程序设计方法。

1）二进制数与十进制数(BCD 码)之间的转换程序设计

例 3.66　设 4 位十进制数(BCD 码)存储在 R2 和 R3 中，R2 存放千位和百位，R3 存放十位和个位，把该数转换为二进制数。

设 4 位十进制数为 $x=d_3d_2d_1d_0$，它可以表示为

$$\begin{aligned} x &= d_3\times10^3+d_2\times10^2+d_1\times10^1+d_0\times10^0 \\ &= (((d_3\times10+d_2)\times10)+d_1)\times10+d_0 \end{aligned} \tag{3.1}$$

也可以表示为

$$x=(d_3\times10+d_2)\times10^2+(d_1\times10^1+d_0) \tag{3.2}$$

式(3.1)和式(3.2)是两种转换算法。显然，式(3.2)的算法比较简单。$x=d_3d_2d_1d_0$ 以 BCD 码形式存储时，d_3d_2 存放在一个单元，而 d_1d_0 存放在一个单元。设计程序时，只要设计 2 位 BCD 码转换的子程序，在高两位 d_3d_2 转换完乘以 100 之后，再加上低两位 d_1d_0 的转换结果即可得到转换结果。2 位 BCD 码转换为二进制数的子程序如下：

```
        ;入口条件: 待转换的 2 位十进制(BCD)码整数在累加器 A 中
        ;出口条件: 转换后的单字节十六进制整数仍在累加器 A 中
        ;影响寄存器: PSW、A、B、R4; 堆栈需求: 2 字节
BCDH:   MOV  B, #10H        ;分离十位数和个位数
        DIV  AB
        MOV  R4, B          ;商为十位数,余数为个位数,暂存个位数于 R4
        MOV  B, #10         ;将十位数转换成二进制数
        MUL  AB             ;d1 × 10 + d0
        ADD  A, R4          ;转换结果在 A 中
        RET
```

下面为 4 位十进制数转换为二进制数子程序，转换结果仍然存储在 R2 和 R3 中：

```
BCD2BN: MOV  A, R3          ;将个位、十位转换成十六进制
        LCALL BCDH          ;调用子程序 d1 × 10 + d0
        MOV  R3, A          ;存个位、十位的二进制数转换结果
        MOV  A, R2          ;将千位和百位转换成二进制
        LCALL BCDH          ;d3 × 10 + d2
        MOV  B, #100        ;(d3 × 10 + d2) × 100
        MUL  AB
        ADD  A, R3          ;x = (d3 × 10 + d2) × 10^2 + (d1 × 10^1 + d0)
        MOV  R3, A
        CLR  A
        ADDC A, B
```

```
        MOV  R2, A
        RET
```

例 3.67 设 16 位二进制数存储在 R6 和 R7 中，R6 中存放高 8 位，把该数转换为 BCD 码形式，并把结果存储在 R3，R4 和 R5 中。

16 位二进制数可以转换为 5 位 BCD 码，因此需要 3 个单元存放。二进制数转换为十进制数的方法为按权展开，设 16 位二进制数 $x=d_{15}d_{14}\cdots d_1d_0$，则对应的十进制数为

$$\begin{aligned}x_{10} &= d_{15}\times 2^{15}+d_{14}\times 2^{14}+\cdots+d_1\times 2^1+d_0\times 2^0\\ &= (\cdots+(d_{15}\times 2+d_{14})\times 2+d_{13})\times 2+\cdots+d_1)\times 2+d_0\end{aligned} \tag{3.3}$$

式(3.3)为二进制数转换为十进制数的算法。转换时，乘以 2 可以采用左移方法实现，从最高位 d_{15} 开始，逐位加到 BCD 码存储单元的最低位，并进行十进制加法调整，然后左移，当最低位 d_0 加入后，转换完成。程序流程图如图 3.72 所示。程序如下：

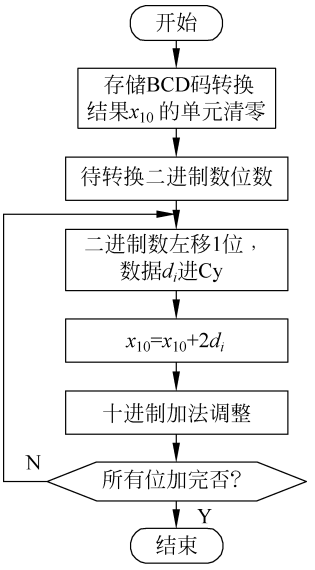


图 3.72 程序流程图

```
HB2:    CLR  A              ;存放转换结果的单元清零
        MOV  R3,A           ;x10 存储在(R3)(R4)(R5)
        MOV  R4,A
        MOV  R5,A           ;x10 = 0
        MOV  R2,#10H        ;转换 16 位二进制数
HB3:    MOV  A,R7           ;把高位移入 Cy 中
        RLC  A
        MOV  R7,A
        MOV  A,R6
        RLC  A
        MOV  R6,A           ;高位已移入 Cy 中
        MOV  A,R5           ;x10 × 2 + d16-i ⇒ x10，第 1 次：0 × 2 + d15 ⇒ x10
        ADDC A,R5
        DA   A              ;十进制调整
        MOV  R5,A
        MOV  A,R4
        ADDC A,R4
        DA   A
        MOV  R4,A
        MOV  A,R3
        ADDC A,R3
        MOV  R3,A           ;双字节十六进制数的万位数不超过 6,不调整
        DJNZ R2,HB3 ;
        RET
```

2）ASCII 代码与十六进制数之间的转换程序设计

例 3.68 把两个 ASCII 码表示的十六进制数转换成一字节的十六进制数。

在 ASCII 码表中，数符‘0’～‘9’的 ASCII 码是 30H～39H，与其代表的十六进制数值相差 30H；数符‘A’～‘F’的 ASCII 码为 41H～46H，与其代表的十六进制数值相差 37H。因此，1 位十六进制数的 ASCII 码转换为十六进制数时，当 ASCII 码减去 30H 的差小于 0AH 时，其差值就是转换结果；否则，差值还应再减去 07H 才能得到转换结果。设 1 位十

六进制数的 ASCII 码存储在 R1 中，其转换结果也存储在 R1 中，子程序如下：

```
ASC2HEX:  MOV  A,R1          ;取操作数
          CLR  C             ;清进位标志位 C
          SUBB A,#30H        ;ASCII 码减去 30H
          MOV  R1,A          ;暂存结果
          SUBB A,#0AH        ;结果是否>9?
          JC   DONE          ;若差≤9，则转换结束
          XCH  A,R1
          SUBB A,#07H        ;若>9,再减 07H
          MOV  R1,A
DONE:     RET
```

通过两次调用子程序 ASC2HEX，然后把转换结果组装成一字节的十六进制数，即可实现题目要求。设两个 ASCII 码分别存储在 R5 和 R6 中，转换结果存储在 R4 中，程序如下：

```
ASCNT:    MOV   A,R5         ;取第一个十六进制数的 ASCII,高位
          MOV   R1,A
          LCALL ASC2HEX
          MOV   A,R1         ;第一个十六进制数的 ASCII 码的转换结果
          SWAP  A
          MOV   R4,A         ;作为 2 位十六进制数的高位
          MOV   A,R6         ;取第二个十六进制数的 ASCII,低位
          MOV   R1,A
          LCALL ASC2HEX
          MOV   A,R1         ;第二个十六进制数的 ASCII 码的转换结果
          ORL   A,R4         ;组装 2 位十六进制数
          MOV   R4,A
          RET
```

十六进制数转换为 ASCII 码的方法比较简单：数符‘0’～‘9’加上 30H，数符‘A’～‘F’加上 37H。读者可以根据以上思路编写程序。

3）ASCII 代码与十进制数(BCD 码)之间的转换程序设计

十进制数符‘0’～‘9’对应的 ASCII 码是 30H～39H，因此，‘0’～‘9’的 BCD 码加上 30H(或者与 30H 相或)就是它所对应的 ASCII 码；反之，数符‘0’～‘9’的 ASCII 码减去 30H(或者与 00001111B 相与)就是它的 BCD 码。

3.5　本章小结

(1) 指令是人们给计算机的命令。一台计算机所有指令的集合称为指令系统。指令有两种表示方式：机器语言和汇编语言。

(2) 寻址方式是 CPU 执行指令时获取操作数的方式。MSC-51 单片机具有 7 种寻址方式：立即寻址、直接寻址、寄存器寻址、寄存器间接寻址、变址寻址、位寻址和相对寻址。

(3) MCS-51 单片机共有 111 条指令。按功能可分为 5 大类：数据传送类指令、算术运算类指令、逻辑运算类指令、控制转移类指令和位操作类指令。

(4) 数据传送类指令分为：通用传送指令、堆栈操作指令、交换指令、访问程序存储器的数据传送指令、访问外部 RAM 和外部 I/O 口的数据传送指令。

通用传送指令：

```
MOV  目的操作数，源操作数
```

作为目的操作数的可以是寄存器（累加器 A，工作寄存器 R0～R7）、特殊功能寄存器和内部 RAM 的存储单元，其中内部 RAM 存储单元的地址在指令中有两种方式：直接给出单元地址，或者由地址寄存器@R0 或@R1 间接指出。除了寄存器（累加器 A，工作寄存器 R0～R7）、特殊功能寄存器和内部 RAM 的存储单元可作为源操作数外，还有 8 位二进制常数。

单片机有一条十六位二进制常数的操作指令：

```
MOV DPTR, #data16
```

堆栈操作指令有两种：

```
PUSH/POP direct
```

MCS-51 单片机的交换指令有字节交换、半字节交换和一字节高低 4 位互换 3 种形式，完成交换必须有累加器 A 参与。

① 字节交换指令：

```
XCH  A,源操作数
```

实现累加器 A 与指定寄存器或单元（直接和间接指出）的内容进行交换。

② 半字节交换指令：

```
XCHD  A,@Ri
```

只能实现累加器 A 与地址寄存器@R0 或@R1 间接指出的存储单元的低 4 位互换。

③ 高低 4 位互换指令：

```
SWAP  A
```

它是累加器 A 独有的一种运算，把 A 的内容高 4 位和低 4 位互换。

MCS-51 单片机访问数据存储器或外部 I/O 口只能通过累加器 A 实现，被访问的存储单元或外部 I/O 口的地址必须通过地址寄存器间接给出。有以下两种操作：

```
读(输入): MOVX  A,源操作数        ; 源操作数为@DPTR、@Ri
写(输出): MOVX  目的操作数,A      ;目的操作数为@DPTR、@Ri
```

MCS-51 单片机访问程序存储器也只能通过累加器 A 实现，仅有读操作。CPU 读取程序存储器某个单元的内容的（查表）指令：

```
MOVC  A,  @A + PC
MOVC  A,  @A + DPTR
```

前者在使用时，常数表应紧跟该指令，最大长度不大于 256B，后者常数表可以放在程序存储器的任何地方。

（5）MCS-51 单片机支持两个无符号的 8 位二进制数的算术运算：

① 加、减运算指令：

```
ADD
ADDC    A,源操作数; 源操作数 { #data  8 位二进制常数
SUBB                          { Rn  工作寄存器
                              { direct  内部 RAM 单元或 SFR
                              { @Ri  地址寄存器指定的内部 RAM 单元
```

② 乘、除运算指令：MUL/DIV AB。

③ 十进制数（BCD 码）加法调整指令：DA A。

以上运算必须有累加器 A 参与，指令执行影响标志位。

④ 加 1、减 1 指令：

INC / DEC　源操作数；源操作数：
- 累加器 A
- Rn　工作寄存器
- direct　内部 RAM 单元或 SFR
- @Ri　地址寄存器指定的内部 RAM 单元

使用时应注意上溢和下溢现象。另外，还有 1 条十六位二进制数加 1 指令：INC DPTR。

(6) 逻辑运算指令可以分成两类：

一类是针对累加器 A 的逻辑操作指令：

① 清零：CLR A；

② 取反：CPL A；

③ 左、右移位：RL/RLC A；RR/RRC A。

另一类为逻辑运算指令：与、或、异或。它们的指令格式：

ANL / ORL / XRL　A,　源操作数；源操作数：
- #data　8 位二进制常数
- Rn　工作寄存器
- direct　内部 RAM 单元或 SFR
- @Ri　地址寄存器指定的内部 RAM 单元

ANL / ORL / XRL　direct,　源操作数；源操作数：
- #data　8 位二进制常数
- A　累加器

通常，与运算用于屏蔽，或运算用于置位，而异或运算用于取反。

(7) 控制转移指令改变了程序的执行顺序，MCS-51 单片机的控制转移指令有无条件转移指令、条件转移指令、循环控制转移指令、子程序调用及返回指令。

无条件转移指令包含 LJMP/AJMP/SIMP　LABEL，它们的功能相同，转移范围不同。

条件转移指令根据判别条件有以下 4 组。

① 以累加器 A 的内容为判别条件：JZ/JNZ　LABEL。

② 以进位位的状态为判别条件：JC/JNC　LABEL。

③ 以可寻址位的状态为判别条件：JB/JNB/JBC　LABEL。

④ 以 2 字节数据比较为判别条件的指令：

```
CJNE  A, direct, LABEL
CJNE  A, #data, LABEL
CJNE  Rn, #data, LABEL
CJNE  @Ri, #data, LABEL
```

CJNE 比较操作不改变两个操作数的状态，但影响标志位 Cy 的状态。若(Cy)＝0，则第一操作数大于第二操作数；若(Cy)＝1，则第一操作数小于第二操作数。

循环控制转移指令：

```
DJNZ Rn/direct, LABEL
```

DJNZ 指令执行时，指定寄存器和单元内容先减 1，然后再判断减 1 之后的内容是否为 0。

调用指令：ACALL/LCALL SUBROTINE；两种调用指令的功能相同，调用范围不同。

返回指令：RET，表示子程序到此结束，由此处返回主程序。

中断返回指令：RETI，表示中断处理到此结束，由此处返回主程序，返回主程序后，CPU 可以响应新的中断请求。

空操作指令：NOP，延时一个机器周期。

伪指令是汇编语言中起解释说明的命令，它不是单片机的指令、汇编时，不会产生目标代码，不影响程序的执行。

常用的伪指令有：(1)设置程序块的起始地址：ORG；(2)赋值：EQU；(3)定义字节数据：DB；(4)定义双字节数据：DW；(5)位地址赋值：BIT；(6)源程序汇编结束：END。

3.6 复习思考题

一、选择题

1. 指令“MOV A,@R0”的寻址方式是(　　)。

A. 寄存器寻址　　B. 寄存器间接寻址　　C. 直接寻址　　D. 立即寻址

2. 指令“MOV R0,＃75”的寻址方式是(　　)。

A. 寄存器寻址　　B. 寄存器间接寻址　　C. 直接寻址　　D. 立即寻址

3. 下列对工作寄存器操作的指令中，正确的是(　　)。

A. MOV R1,R7　　B. MOV R1,@R1

C. MOV R2,@DPTR　　D. MOV R1,B

4. 要把内部 RAM 的一个单元内容取到累加器中，下面指令中正确的是(　　)。

A. MOV A,@R2　　B. MOVX A,@R1

C. MOV A,@R1　　D. MOV A,@DPTR

5. 下面交换指令中正确的是(　　)。

A. XCH A,@R1　　B. XCH A,＃3AH

C. XCH 20H,R1　　D. XCH R1,20H

6. 下面半字节交换指令中正确的是(　　)。

A. XCHD A,@R1　　B. XCHD A,R1

C. XCHD A,＃23H　　D. XCHD A,20H

7. CPU 要取外部 RAM 的一个单元内容，下面指令中正确的是(　　)。

A. MOV @R1,A　　B. MOVX @DPTR,A

C. MOV A,@R1　　D. MOVX A,@DPTR

8. CPU 执行“MOV PSW,＃38H”后，当前工作寄存器组(　　)。

A. 保持不变　　B. 切换到 BANK0

C. 切换到 BANK1　　D. 切换到 BANK2

E. 切换到 BANK3

9. 累加器 A 的内容为 0BCH，执行指令“ADD A,＃2DH”后，OV 和 P 的状态是(　　)。

A. 0,0　　B. 0,1　　C. 1,0　　D. 1,1

10. 累加器 A 的内容为 0BCH，Cy 当前状态为 1，执行指令“SUBB A,＃0D7H”后，Cy 和 AC 的状态是(　　)。

A. 0,0　　B. 0,1　　C. 1,0　　D. 1,1

11. 下列哪条减法指令是正确的？(　　)

A. SUBB R7,#05H　　B. SUBB 30H,@R1

C. SUBBC A,#30H　　D. SUBB A,@R1

12. “MUL AB”指令执行后,若积超过255,则(　　)。

A. (Cy)=1　　B. (AC)=1　　C. (OV)=1　　D. (P)=1

13. “DIV AB”指令执行后,(　　)。

A. (Cy)=0　　B. (AC)=1　　C. (OV)=1　　D. (P)=1

14. 要滤除掉20H单元的最低位和最高位,正确的操作是(　　)。

A. XRL 20H,#81H　　B. ANL 20H,#7EH

C. ORL 20H,#81H　　D. SUBB 20H,#81H

15. 要把累加器A的第3、4位取反,正确的操作是(　　)。

A. XRL A,#18H　　B. ANL A,#18H

C. ORL A,#18H　　D. SUBB A,#18H

16. 要把特殊功能寄存器IE的第3、4位置1,正确的操作是(　　)。

A. XRL IE,#18H　　B. ANL IE,#18H

C. ORL IE,#18H　　D. ADD IE,#18H

17. 下列指令中,能实现对20H单元内容取反的是(　　)。

A. CPL 20H　　B. ANL 20H,#00H

C. XRL 20H,#0FFH　　D. XRL 20H,#00H

18. 下列指令中,指令执行影响标志位的是(　　)。

A. CPL　A　　B. RLC A　　C. RL A　　D. RR A

19. 把20H单元的最低位送入累加器A对应的位置,正确的做法是(　　)。

A. MOV A,20H.0　　B. MOV ACC.0,20H.0

C. MOV 0E0H,00H　　D. 不能直接传送

20. 已知(2FH.7)=0,执行“ANL C,/2FH.7”指令后,(2FH.7)的状态是(　　)。

A. 0　　B. 1

C. 不能确定　　D. 取决于C当前的状态

21. 下列指令中错误的是(　　)。

A. CLR A　　B. CLR 27H.5　　C. CLR R7　　D. CLR C

22. LJMP指令的转移范围是(　　)。

A. 本条指令上、下方2KB

B. 64KB

C. 本条指令上方128B、下方127B

D. 本条指令上、下方128B

23. JZ指令的判断条件是(　　)。

A. 某一单元的内容为0

B. 工作寄存器Rn的内容为0,n=0～7

C. 累加器A的内容为0

D. 一个 SFR 的内容为 0

24. CJNE 指令执行时，影响的标志位是（　　）。

A. OV　　B. Cy

C. AC　　D. P

25. 指令“JBC 28H.5，GOON”执行后，28H.5 的状态是（　　）。

A. 1　　B. 0　　C. 不确定　　D. 与 C 相同

26. 已知(R7)＝78H，执行指令“DJNZ R7，NEXT”后，R7 的内容是（　　）。

A. 79H　　B. 78H　　C. 不确定　　D. 77H

27. 对于子程序调用指令 ACALL 来说，子程序在程序存储器中的放置范围是（　　）。

A. 本条指令上、下方 2KB　　B. 64KB

C. 本条指令上方 128B、下方 127B　　D. 本条指令上、下方 128B

28. 对于子程序调用指令 LCALL 来说，子程序在程序存储器中的放置范围是（　　）。

A. 本条指令上、下方 2KB　　B. 64KB

C. 本条指令上方 128B、下方 127B　　D. 本条指令上、下方 128B

29. 关于 RET 指令，不正确的描述是（　　）。

A. 放置在子程序的最后，标志一个子程序的结束

B. CPU 执行该指令的目的是获取返回地址

C. RET 指令执行时包含了出栈操作

D. RET 指令是程序结束标志，CPU 执行程序时，遇到 RET 指令，终止执行的程序

30. 关于 NOP 指令，下面说法正确的是（　　）。

A. CPU 什么也不做，原地踏步

B. CPU 处于等待状态，需要消耗一定时间

C. CPU 不做任何操作，只把 PC 的内容加 1，产生一个机器周期的延时

D. 无用指令

二、思考题

1. 什么是寻址方式？在 MCS-51 单片机有哪几种寻址方式？

2. 设内部 RAM 中 59H 单元的内容为 50H，CPU 执行下列程序段后，寄存器 A、R0 和内部 RAM 中 50H、51H 单元的内容是多少？

```
MOV  A, 59H
MOV  R0, A
MOV  A, #00H
MOV  @R0, A
MOV  A, #25H
MOV  51H, A
MOV  52H, #70H
```

3. 已知 4EH 和 4FH 单元的内容分别为 20H 和 5FH，执行下列指令后，DPTR 的内容是多少？

```
MOV  A, 4EH
MOV  R0, #4FH
XCH  A, @R0
SWAP A
```

```
XCHD A,@R0
MOV  DPH,@R0
MOV  DPL,A
```

4. CPU 执行下列程序后，A 和 B 寄存器的内容是多少？

```
MOV  SP, #3AH
MOV  A, #20H
MOV  B, #30H
PUSH ACC
PUSH B
POP  ACC
POP  B
```

5. 设外部 RAM 的 2000H 单元内容为 80H，CPU 执行下列程序后，A 的内容是多少？

```
MOV  P2, #20H
MOV  R0,  #00H
MOVX A,  @R0
```

6. 指令 XCH、XCHD 和 SWAP 有什么区别？

7. 指令“MOVC A,@A+DPTR”与“MOVC A,@A+PC”有什么不同？

8. 假定累加器 A 的内容为 30H，CPU 执行指令下列后 CPU 把程序存储器的哪个单元的内容送到了累加器 A 中？

```
1000H: MOVC A ,@A + PC
```

9. 假定 DPTR 的内容为 8100H，累加器的内容为 40H，CPU 执行下列指令后，读取的是程序存储器哪个单元的内容？

```
1000H: MOVC A,@A + DPTR
```

10. 假定(SP)=60H，(ACC)=30H，(B)=70H，CPU 执行下列程序后，SP，60H，61H、62H 的内容各是多少？

```
PUSH ACC
PUSH B
```

11. 假定(SP)=62H，(61H)=50H，(62H)=7AH，CPU 执行下列程序后，SP，60H，61H，62H 及 DPTR 的内容各是多少？

```
POP  DPH
POP  DPL
```

12. 假定(A)=85H，(R0)=20H，(20H)=0AFH，CPU 执行指令

```
ADD  A,  @R0
```

累加器 A 及 Cy，AC，OV，P 的内容是多少？

13. 假定(A)=85H，(20H)=0FEH，(Cy)=1，执行指令

```
ADD  A,  20H
```

累加器 A 的内容及 Cy，AC，OV，P 的内容是多少？

14. 假定(A)=0FFH，(R3)=0FH，(30H)=0F0H，(R0)=40H，(40H)=00H，CPU 执行下列指令后，上述寄存器和存储单元的内容是多少？

```
INC  A
INC  R3
INC  30H
INC  @R0
```

15. 假定(A)＝56H,(R5)＝67H,CPU 执行下列指令后 A 和 Cy 的内容是多少?

```
ADD  A ,R5
DA   A
```

16. 分析下面的程序,指出是对哪几个单元进行了加法运算,结果存在哪个单元?

```
MOV  A, 20H
MOV  R0, #30H
ADD  A,@R0
INC  R0
ADD  A,@R0
MOV  @R0,A
```

17. ADD 指令和 ADDC 指令有什么不同?

18. DA 指令起什么作用? 它如何使用?

19. 假定(A)＝0FH,(R7)＝19H,(30H)＝00H,(R1)＝40H,(40H)＝0FFH,CPU 执行下列指令后,上述寄存器和存储单元的内容是多少?

```
DEC  A
DEC  R7
DEC  30H
DEC  @R1
```

20. 分析下面的程序,参与加减法运算的单元有哪些? 结果存在哪个单元?

```
MOV  A, 20H
MOV  R0, #30H
CLR  C
SUBB A,@R0
DEC  R0
ADD  A,@R0
MOV  @R0,A
```

21. 假定(A)＝50H,(B)＝0A0H。CPU 执行指令“MUL　AB”后,寄存器 B 和累加器 A 的内容各是多少? Cy 和 OV 的状态是什么?

22. 假定(A)＝0FBH,(B)＝12H。执行指令“DIV　AB”后,寄存器 B 和累加器 A 的内容各是多少? Cy 和 OV 各是什么状态?

23. 已知(A)＝83H,(R0)＝17H,(17H)＝34H。CPU 执行完下列程序段后 A 的内容是多少?

```
ANL  A,#17H
ORL  17H, A
XRL  A, @R0
CPL  A
```

24. 设(A)＝55H,(R5)＝0AAH,如果 CPU 分别执行下列指令,A 和 R5 的内容是多少?

```
(1) ANL  A,R5
```

(2) ORL　A,R5
(3) XRL　A,R5

25. 分析下列指令序列,写出它所实现的逻辑表达式。

```
MOV  C,  P1.0
ANL  C,  P1.1
ORL  C,  /P1.2
MOV  P3.0, C
```

26. 指令“LJMP PROG”和“LCALL PROG”有什么区别?

27. 已知(20)=00H,执行下列程序段后,程序将如何执行?

```
DJNZ 20H, REDO
MOV  A, 20H
```

28. CPU 分别执行指令“JB ACC.7,LABEL”和“JBC ACC.7,LABEL”后,它们的结果有什么不同?

29. RET 和 RETI 指令有什么区别?

30. 当系统晶振为 12MHz 时,计算下列子程序的执行时间。

```
SUBRTN:MOV  R1,#125
REDO:  PUSH ACC
       POP  ACC
       NOP
       NOP
       DJNZ R1,REDO
       RET
```

三、程序设计

1. 把内部 RAM 的 20H,21H,22H 单元的内容依次存入 2FH,2EH 和 2DH 中。

2. 把外部 RAM 的 2040H 单元内容与 3040H 单元内容互换。

3. 把内部 RAM 的 40H 单元与 5000H 单元的低 4 位互换。

4. 已知一个二维数据表格如下,存储在程序存储器中,编程实现自动查表。

X	0	1	2	3	4	…	0B	0C	0D	0E	0F
Y	11	12	01	AD	DD	…	AB	24	4B	7C	AA

5. 已知二进制数 X 和 Y,X 被存放在 20H(高 8 位)和 21H(低 8 位)单元,Y 被存放在 22H,编程实现 X+Y。

6. 已知 8 位十进制数 X 和 Y 以压缩 BCD 的格式存储,X 被存放在 20H～23H 单元,Y 被存放在 40H～43H 单元,编程实现 X+Y。

7. 已知十进制数 X 和 Y 以压缩 BCD 码的格式存储,X 被存放在 20H(高位)和 21H 单元,Y 被存放在 22H 和 23H 单元,编程实现 X−Y。

8. 已知二进制数 X 被存放在 20H 单元,编程实现 X^3。

9. 已知二进制数 X 被存放在 20H(高 8 位),21H,22H 单元,Y 被存放在 30H 单元,编程实现 X×Y。

10. 二进制数 X 被存放在 20H(高 8 位),21H 单元,用移位方法实现 2X。

11. 4 位十进制数 X 以压缩 BCD 的格式存储在内部 RAM 中,编程实现 X 乘以 10。

12. 二进制数 X 被存放在 20H(高 8 位)、21H 单元，用移位方法实现 X/2。

13. 4 位十进制数 X 以压缩 BCD 的格式存储在内部 RAM 中，编程实现 X/10，并把小数部分存储在 R6 中。

14. 非正数 X 被存放在 20H(高 8 位)、21H 单元，求该数的补码。

15. X 是二进制数，编程实现下列要求：X＝0 时，执行程序 PROG1；X＝1 时，执行程序 PROG2；X＝2 时，执行程序 PROG3；X＝3 时，执行程序 PROG4。

16. 求出无符号单字节数 X，Y，Z 中的最大数，并把它存放在 50H 单元。

17. 请把内部 RAM 的 20H～2FH 连续 16 个单元的内容转移到外部 RAM 的 2000H 单元开始的区域中。

18. 假设 U 为 P1.1，V 为 P1.2，W 为 P3.3，X 为 28H.1，Y 为 2EH.0，Z 为 TF0，Q 为 P1.5，编制程序实现下列逻辑表达式：$Q=\overline{U}+V+W+\overline{\overline{X}\overline{Y}}\cdot\overline{Z}$。

19. 一批 8 位二进制数据存放在单片机内部 RAM 以 20H 单元开始的区域，数据长度为 100 个，编程统计该批数据中数值为 65H 的数据的个数，将统计结果存放在 R7 中。

20. 一批 8 位二进制数据存放在单片机内部 RAM 以 10H 单元开始的区域，数据长度为 50 个，编制程序统计该批数据中的偶数，并把偶数存放在内部 RAM 以 50H 开始的区域。

21. 编程查询外部 RAM 的 3000H 单元中 0 和 1 的个数，把结果存储在 R5 和 R6 中。

22. 4 位十进制数以压缩 BCD 码格式被存放在 20H(高位)和 21H 单元，请将该数转换为分离式 BCD 码形式，并将结果存在 30H，31H，32H，33H 单元。用调用子程序的方法实现。

23. 用 P1 口驱动图 3.73 所示的 LED 显示装置，设计驱动电路并编制程序实现下列要求：LED 依次顺时针点亮——逆时针灭——全亮若干秒全灭，周而复始地重复上述过程。系统晶振为 12MHz。

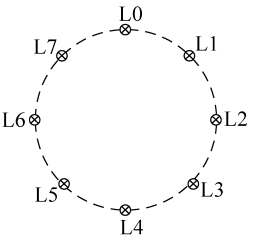


图 3.73　显示装置

24. 已知 a、b 为 8 位无符号二进制数，分别存在 data 和 data＋1 单元，编写程序计算 5a＋b。

25. 已知 16 位二进制数以补码形式存放在 data 和 data＋1 单元，求其绝对值并将结果存储在原单元。(提示：求出原码后，再求绝对值)

26. 在单片机内部 RAM 中从 20H 单元开始存储 50 个数据，请编写一个程序统计其中正数的个数，并将统计结果存放于 70H 单元。

27. 从内部 RAM 的 20H 单元开始存一批带符号的 8 位二进制数据，数据长度存放在 1FH 单元中，统计其中大于 0、小于 0、等于 0 的个数，并把统计结果分别存放在 ONE，TWO，THREE 单元。

28. 从内部 RAM 的 20H 单元开始存放 30 个带符号的 8 位二进制数据，编写一个程序，分别把正数和负数存放在 51H 和 71H 开始的区域，并统计正数和负数的个数，分别存放在 50H 和 70H 单元。

29. 搜索一串 ASCII 码字符串中的最后一个非空格字符，该字符串从外部 RAM 的 8000H 单元开始存放，以回车符(ASCII 码为 ODH)结束。编程实现搜索，并将搜索到的最

后一个非空格字符的单元地址存放在40H和41H单元。

30. 5个双字节无符号数求和，数据存放在外部RAM的5000H单元开始的区域，把结果存放在以SUM开始的内部RAM单元中。

31. 把外部RAM中BLOCK1为首地址的数据块传送到内部RAM中以BLOCK2为首地址开始的区域，数据长度为length。

32. 把长度为LENGTH的字符串从内部RAM的BLOCK1单元开始传送到外部RAM的以BLOCK2单元开始的区域，在传送过程中如果碰到回车符(CR)时，传送即刻结束。

33. 某一应用系统数据缓冲区开辟在外部RAM中，用于存储单字节数据，缓冲区从BUFFER单元开始，长度为100个单元，为了某种统计需要，要求缓冲区的非负数存储在单元地址为BLOCK1开始的区域，其余的数存储在单元地址为BLOCK2开始的区域，这两个缓冲区也设置在外部RAM中。

34. 已知无符号数二进制数 x 存放于20H单元，y 存放于21H单元，编写程序实现下列表达式：

$$y=\begin{cases}x/2, & x<5\\ 5x-7, & 5\leqslant x<15\\ 30, & x\geqslant 15\end{cases}$$

35. 已知逻辑表达式 $Q=(\overline{W+V})+\overline{U}(\overline{DE})+X$，其中，Q为P1.5，X为P1.0，U为P1.1，V为P1.2，W为22H.0，D为22H.5，E为定时计数器T0的溢出标志TF0，请编写程序实现上述逻辑功能。

36. 在20H，21H和22H单元存储了一个6位十进制数，把该数转换成ASCII码并存放到30H单元开始的区域。

37. 编写程序把6位十进制数转换为二进制数。