第 3 章　51 系列单片机的汇编指令系统

单片机所能执行的命令的集合就是它的指令系统。凡是具有 8051 内核的单片机均使用 51 系列单片机的汇编语言指令系统。指令常用英文名称或其缩写来作为助记符。

【本章目标】

- 理解汇编指令的寻址方式；
- 掌握 51 系列单片机汇编指令、伪指令；
- 掌握单片机程序设计的基本方法；
- 能够运用汇编语言进行程序设计。

3.1　指令系统概述

3.1.1　51 系列单片机的汇编语言指令及分类

51 系列单片机汇编语言，包含两类不同性质的指令。

(1) 基本指令：即指令系统中的指令。它们都是机器能够执行的指令，每一条指令都有对应的机器代码，该机器代码对应着一种操作。

(2) 伪指令：汇编时用于控制汇编的指令。它们都是机器不能执行的指令，编译后无机器码生成。

下面所讲的汇编语言指令是指基本指令，伪指令有特定的说明。51 系列单片机的基本汇编语言指令共有 111 条。可根据所占字节数、执行时间以及指令功能的不同进行分类。

1. 按照编译后指令在存储器中所占的字节数分类

(1) 单字节指令(49 条)：指令格式由 8 位二进制编码表示，例如：

```
CLR  A→E4H
```

(2) 双字节指令(45 条)：指令格式由两个字节组成，例如：

```
MOV  A, ＃10H→74H  10H
```

(3) 三字节指令(17 条)：指令格式由三个字节组成，例如：

```
MOV  40H, ＃30H→75H 40H 30H
```

2. 按指令的执行时间分类

(1) 1 个机器周期指令(单周期指令)64 条；

(2) 2 个机器周期指令(双周期指令)45 条；

(3) 4 个机器周期指令(四周期指令)2 条。

3. 按功能分类

(1) 算术操作类 24 条；

(2) 数据传送类 28 条；
(3) 逻辑运算类 25 条；
(4) 控制转移类 17 条；
(5) 位操作类 17 条。

3.1.2　汇编语言指令格式

1. 汇编指令的格式

指令格式就是指令的表示方法，指令格式通常包括操作码和操作数。

操作码表示该指令的操作功能，即指令进行什么操作(又被称作操作符、助记符)，操作码是指令的功能部分，不能省略。

操作数是指令要操作的数据信息。操作数是指对什么数进行操作，即指令操作的对象。操作数可能是具体的数或数存放的地址。根据指令的不同功能，操作数的个数有 3、2、1 或没有操作数。

51 系列单片机指令中有单字节、双字节、三字节这些不同长度的指令，指令长度不同，指令的格式也不同：

(1) 单字节指令，指令只有 1 个字节，操作码和操作数都在同一个字节中；
(2) 双字节指令，第一个字节为操作码，第二个字节为操作数；
(3) 三字节指令，第一个字节为操作码，第二、第三个字节为操作数。

2. 汇编语言的语句格式

汇编语言源程序是由汇编语句(即指令)组成的。汇编语言一般由 4 部分组成。每条语句占有一行，典型的汇编语句格式如下：

```
标号:     操作码     操作数      ;注释
START:    MOV        A,30H       ;A←(30H)
```

汇编语句需要注意以下几点：

(1) 标号字段和操作码字段之间要有冒号“：”分隔；
(2) 操作码字段和操作数字段间的分界符是空格；
(3) 双操作数之间用逗号相隔；
(4) 操作数字段和注释字段之间的分界符用分号“;”，
(5) 任何语句都必须有操作码字段，其余各字段为任选项。

1) 标号

标号是指用符号表示的指令地址，即本条语句机器码的第一个字节所在的地址。在程序中可以引用这个标号代表这个地址。标号的基本要求如下：

(1) 必须以“：”结束；
(2) 由 1～8 个英文字母或数字组成，但第一个符号必须是英文字母；
(3) 同一个标号在一个程序中不能重复定义；
(4) 各种寄存器名、指令助记符、伪指令不能用作标号；
(5) 可以没有标号，一般只有被其他语句引用的才赋予标号。

2) 操作码

操作码字段规定了语句执行的操作，操作码是汇编语言指令中唯一不能空缺的部分。

操作码可以是指令助记符，也可以是伪指令。

3）操作数

操作数的个数可以是 0～3 个，可以是数字、标号、寄存器名称，中间用“,”分开，操作数有如下几种格式。

（1）可以是二进制数，例如：

```
MOV   A,   #00100001B
```

（2）可以是十进制数，例如：

```
MOV   A,   #33
```

（3）可以是十六进制数，例如：

```
MOV   A,   #21H
```

以 A～F 开头的十六进制数必须在其前面加上数字 0，即：

```
MOV   A,   #0F8H
```

（4）可以是当前指令地址，常用 $ 表示 PC 当前值，例如：

```
HERE: SJMP   HERE   或   SJMP   $
```

（5）可以是标号，例如：

```
LJMP   NEXT
```

（6）可以是寄存器名，也可以是其地址，例如：

```
MOV   A,P0   或  MOV   A,80H
```

4）注释

注释字段是对注释程序的说明，用“;”开头；不产生任何指令代码。长度不限，一行写不下可以换行，换行后也需要用“;”开头。

3.2 单片机的寻址方式

所谓寻址方式就是如何寻找操作数或操作数存放的地址。或者说通过什么方式找到操作数。由寻址方式指定参与运算的操作数或操作数所在单元的地址。寻址方式越多，计算机寻址能力越强，但指令系统也越复杂。

51 系列单片机有 7 种寻址方式。

1. 立即数寻址

立即数寻址方式是操作数直接在指令中给出，指令操作码后面字节的内容就是操作数本身，为了与直接寻址相区别，立即数前面要加 #。立即数只能作为源操作数，不能当作目的操作数。图 3.1 所示的立即数寻址包括以下操作：

```
MOV   A,     #40H          ;指令代码 74H,40H,把常数 40H 送到累加器 A 中
MOV   DPTR,  #5678H        ;指令代码 90H,56H,78H,把常数 5678H 送到寄存器 DPTR 中
```

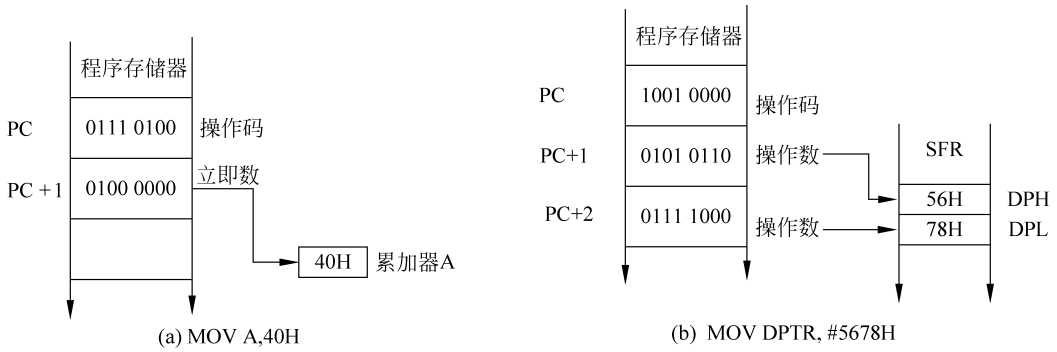


图 3.1　立即数寻址方式示意图

2. 直接寻址

直接寻址方式中操作数是直接以单元地址的形式给出，该地址中的数为操作数。

由于操作数是以存储单元地址的形式给出，因此是 8 位二进制数表示的字节地址。直接寻址的区域为：

(1) 片内 RAM 的 128 个单元

(2) 特殊功能寄存器，可以用单元地址的形式也可以用寄存器符号的形式给出，例如：

```
MOV  A, 40H;   E540H     ;把片内 RAM 字节地址 40H 单元的内容送到累加器 A 中
MOV  40H,A;    F540H     ;把累加器 A 中内容送到片内 RAM 字节地址为 40H 的单元
MOV  50H,60H;  855060H   ;把片内 RAM 字节地址 60H 单元的内容送到 50H 单元中
INC   60H                ;将地址 60H 单元中的内容自加 1。
```

3. 寄存器寻址

寄存器中存放的是操作数，即寄存器中的内容是操作数，因此指定了寄存器就能得到操作数。例如：

```
MOV  A,Rn                ;n 为 0～7,Rn 是当前工作寄存器.把寄存器 Rn 中的内容送到累加器 A 中
MOV  Rn,A                ;把累加器 A 中的内容送到寄存器 Rn 中
```

寄存器寻址范围：

(1) 当前工作寄存器区的 8 个工作寄存器，R0～R7；

(2) 特殊功能寄存器，如 A、B、DPTR 等。

4. 寄存器间接寻址方式

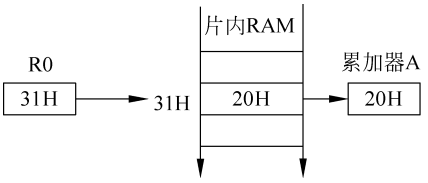
指令中的寄存器中存放的是操作数地址，该地址单元的内容是所需的操作数，这种寻址方式称为寄存器间接寻址，即先通过寄存器找到地址，然后从地址中取出操作数。寄存器间接寻址用符号“@”表示：

```
MOV  A,     @Ri
```

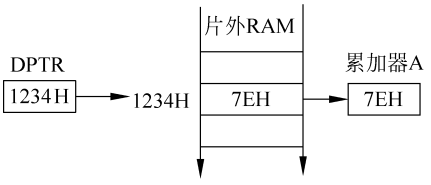
其中，Ri 只能是寄存器 R0 或 R1，例如，根据图 3.2 所示，可有以下指令：

```
MOV  R0,    #31H         ;R0←31H
MOV  A,     @R0          ;A←((R0))
MOV  @R1,   A            ;((R1))←A
```

```
MOV   DPTR,  #1234H     ;DPTR←3456H
MOVX  A,     @DPTR      ;A←((DPTR))是把 DPTR 寄存器所指的外部数据存储器(RAM)的内容传送
                        ;给 A,假设(1234H) = 7EH,指令运行后(A) = 7EH
```



(a) MOV A,@R0



(b) MOVX A,@DPTR

图 3.2　寄存器间接寻址方式示意图

寄存器间接寻址的范围：

(1) 片内 RAM 的 128B(或 256B)中的数据；

(2) 适用于访问外部 RAM,可使用 R0、R1 访问片外 RAM 的低 256 字节；

(3) DPTR 作为地址指针可以访问外部 RAM 的 64KB。

堆栈区 PUSH、POP 相当于以指针 SP 作为间接寄存器的间接寻址方式。

```
SP = 07H
PUSH   ACC              ;SP←SP + 1,08H←(ACC)
POP    ACC              ;ACC←(08H),SP←SP - 1
```

5. 变址寻址

变址寻址也称为基址寄存器＋变址寄存器间接寻址。

这种寻址方式可以读出程序存储器中的数据并送到寄存器 A 中,一般用于访问程序存储器中的常数表格。如图 3.3 所示,它以基址寄存器(DPTR 或 PC)的内容为基本地址,以寄存器 A 为变址寄存器,以两者相加的内容形成的 16 位地址作为操作数地址,访问程序存储器中的数据表格。其基本格式为：

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

变址寻址是专门针对程序存储器的寻址方式,范围达 64KB,该寻址方式只有 3 条指令：

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

这 3 条指令都是单字节指令。

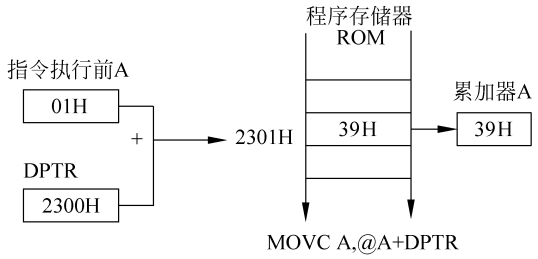


图 3.3　变址寻址方式示意图

6. 相对寻址

指令中给出的操作数为程序转移的偏移量。相对寻址只出现在相对转移指令中，相对转移指令执行时，是以当前的 PC 值加上指令中规定的偏移量 rel 形成实际的转移地址。这里所说的 PC 的当前值是执行完相对转移指令后的 PC 值，一般将相对转移指令操作码所在的地址称为源地址，转移后的地址称为目的地址。

目的地址＝转移指令所在地址＋转移指令字节数＋rel

在 8051 的指令系统中，有许多条相对转移指令，这些指令多数均为两字节指令，只有个别的是三字节的指令。偏移量 rel 是一个带符号的 8 位二进制补码数，所能表示的数的范围是－128～＋127。因此，以相对转移指令的所在地址为基点，向前最大可转移(127＋转移指令字节数)个单元地址，向后最大可转移(128－转移指令字节数)个单元地址。

以当前 PC 的内容为基础，加上指令给出的一字节补码数(偏移量)形成新的 PC 值的寻址方式。相对寻址用于修改 PC 值，主要用于实现程序的分支转移。图 3.4 所示的相对寻址指令为：

```
SJMP   15H              ;PC←PC + 2 + 15H
```

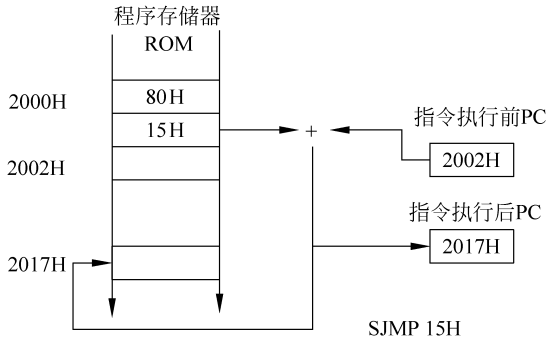


图 3.4　相对寻址方式示意图

7. 位寻址

采用位寻址方式的指令，操作数是 8 位二进制数中的某一位。指令中给出的是位地址，是片内 RAM 某个单元中的某一位的地址。位地址在指令中用 bit 表示。例如：

```
CLR     P1.0            ;(P1.0) ← 0
SETB    ACC.7           ;(ACC.7)← 1
CPL     C               ;( C )← NOT( C )
```

可位寻址的区域包括以下几部分：

(1) 片内 RAM 中的位寻址区：片内 RAM 中的单元地址 20H～2FH，共 16 个单元 128 位为位寻址区，位地址是 00H～7FH。对这 128 位的寻址使用直接位地址表示；

(2) 可位寻址的特殊功能寄存器位：可供位寻址的特殊功能寄存器共有 11 个，有 5 位没有定义，因此寻址位为 83 位。

位地址有 3 种表示方式：

(1) 直接使用位地址表示，对于 20H～2FH 的 16 个单元共 128 位，位地址分布是 00H～7FH。

(2) 对于特殊功能寄存器，可以直接用寄存器名字加位数表示，如 PSW.3 和 ACC.5 等；也可以用单元地址加位的表示方法，例如 PSW 的字节地址为 D0H，PSW 的 bit5 表示为(0D0H).5；或者直接使用位地址表示方法(直接表示为 0D5H)。

(3) 对于定义了位名字的特殊位，可以直接用其位名表示，例如：CY、AC 等。

3.3 51 单片机指令系统分类介绍

在分类介绍指令前，先简单介绍描述指令的一些符号的意义，如表 3.1 所示。

表 3.1 指令符号及意义

符　　号	意　　义
A	累加器 ACC。常用 ACC 表示其地址，用 A 表示其名称
AB	累加器 ACC 和寄存器 B 组成的寄存器对。通常在乘、除法指令中出现
Rn	选定的当前工作寄存器，范围为 R0～R7(n=0～7)
Ri	工作寄存器 R0 或 R1(i=0 或 1)
@	间接寻址符号，简称间址符，常与 Ri 配合用，如@Rl，表示指令对 R1 寄存器间接寻址
#data	8 位立即数，“#”表示后面的 data 是立即数而不是直接地址
direct	表示片内 RAM 存储单元的 8 位直接地址，立即数和直接地址可用二进制码表示，后缀为“B”；也可用十六进制数码表示，后缀为“H”；如果是以字母开头的十六进制数，在其前面应加一个“0”。例如，二进制数码 10101000B 也可转成十六进制码 A8H，但必须写成“0A8H”
@DPTR	以 DPTR 为数据指针的间接寻址，用于对外部 64KB RAM/ROM 寻址
rel	以补码形式表示的 8 位地址偏移量，范围为－128～＋127
Bit	位地址
$	当前指令的地址
←→	取代或替换
(X)	表示 X 寄存器或 X 地址单元中的内容
((X))	表示以 X 寄存器或 X 地址单元中的内容为地址的存储单元中的内容

【例 3-1】 已知数据存储器各单元内容如图 3.5 所示，说明(50H)、(A)、((50H))各为多少？

解：(50H)：表示地址为 50H 存储单元里的内容 01110000B。

(A)：表示累加器 A 中的内容，因 A 的地址为 0E0H，所以(A)为 0010 0001B。

((50H))：以 50H 存储单元的内容 70H 为地址的存储单元内的内容，为 00111001B。

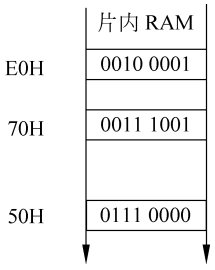


图 3.5　数据表示方式示意图

3.3.1　算术运算类指令

算术运算类指令都是通过算术逻辑运算单元 ALU 进行数据运算处理的指令。它包括各种算术操作，其中有加、减、乘、除四则运算。80C51 单片机还有带借位减法、比较指令。加法类指令包括加法、带进位的加法、加 1 以及二-十进制调整。但 ALU 仅执行无符号二进制整数的算术运算。对于带符号数则要进行其他处理。

除了加 1 和减 1 指令之外，算术运算结果将使进位标志(CY)、半进位标志(AC)、溢出标志(OV)置位或复位。

1. 加法指令

```
ADD   A,#data          ;(A)+data→(A)   (立即数寻址)
ADD   A,direct         ;(A)+(direct)→(A)(直接寻址)
ADD   A,Rn             ;(A)+(Rn)→(A)(寄存器寻址)
ADD   A,@Ri            ;(A)+((Ri))→(A)(寄存器间接寻址)
```

这组指令的功能是将立即数、片内 RAM 单元中的内容、工作寄存器 Rn 中的内容、间接地址存储器中的 8 位无符号二进制数及与累加器 A 中的内容相加，相加的结果仍存放在 A 中。

这类指令将影响标志位 AC、CY、OV、P。

(1) 当和的第 3 位有进位时，将 AC 标志置位，否则为 0；

(2) 当和的第 7 位有进位时，将 CY 标志置位，否则为 0；

(3) 当和的第 7 位与第 6 位中有一位进位而另一位不产生进位时，溢出标志 OV 置位，否则为 0。

溢出标志位 OV 的状态，只有带符号数加法运算时才有意义。当两个带符号数相加时，OV=1，表示两个正数相加，和为负数；或两个负数相加而和为正数的错误结果。表示加法运算超出了累加器 A 所能表示的带符号数的有效范围(−128～+127)，即产生了溢出，表示运算结果是错误的，否则运算是正确的，即无溢出产生。

【例 3-2】　设(A)=74H。(30H)=9CH，执行指令：

```
ADD   A, 30H
```

执行结果：(A)=10H，(CY)=1，(OV)=0，(AC)=1。

【例 3-3】 (A)=85H,(R0)=30H,(30H)=0AFH,执行指令:

```
ADD  A,  @R0
```

执行结果:(A)=34H,(CY)=1,(OV)=1,(AC)=1。

2. 带进位位加法指令

```
ADDC  A,#data          ;(A)←(A) + #data + (CY)(立即数寻址)
ADDC  A,direct         ;(A)←(A) + (direct) + (CY)(直接寻址)
ADDC  A,Rn             ;(A)←(A) + (Rn) + (CY)(寄存器寻址)
ADDC  A,@Ri            ;(A)←(A) + ((Ri)) + (CY)(寄存器间接寻址)
```

这组指令的功能是将立即数、片内 RAM 单元中的内容、工作寄存器 Rn 中的内容、间接地址存储器中的 8 位无符号二进制数及与累加器 A 中的内容相加,再加上进位标志位的内容,相加的结果仍存放在 A 中。这类指令影响标志位 AC、CY、OV、P,同 ADD 指令。

【例 3-4】 (A)=87H, CY=1。执行指令:

```
ADDC  A,  #0FCH
```

执行结果:(A)=84H,(CY)=1,(OV)=0,(AC)=1。

3. 带借位减法指令

```
SUBB  A,#dala          ;(A) - data - (CY)→(A)
SUBB  A,drect          ;(A) - (drect) - (CY)→(A)
SUBB  A,Rn             ;(A) - (Rn) - (CY)→(A)
SUBB  A,@Ri            ;(A) - ((Ri)) - (CY)→(A)
```

这组指令的功能是从 A 中减去进位位 CY 和指定的变量,结果(差)存入 A 中。这类指令影响标志位 AC、CY、OV、P。

(1) 若第 7 位有借位则 CY 置 1,否则 CY 清 0;

(2) 若第 3 位有借位,则 AC 置 1,否则 AC 清 0;

(3) 若第 7 位和第 6 位中有一位需借位而另一位不借位,则 OV 置 1。

OV 位用于带符号的整数减法,OV=1,则表示正数减负数结果为负数,或负数减正数结果为正数的错误结果。

需要注意的是,在 80C51 指令系统中没有不带借位的减法。如果需要的话,可以在 SUBB 指令前,用 CLR C 指令将 CY 先清零。

【例 3-5】 设(A)=0C1H,(R0)=40H,CY=1。执行指令:

```
SUBB  A,  R0
```

执行结果:(A)=80H,(CY)=0,(OV)=1(位 7 无借位,位 6 有借位),(AC)=0。

4. 增 1 指令

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

这组指令是把源操作数加1，当用本指令修改输出口P0～P3时，原始数据的值将从锁存器读入，而不是从引脚读入。这类指令不影响各个标志位。

指令INC　DPTR，16位数增1指令。DPTR寄存器是由DPH和DPL组成的，首先对低8位指针DPL执行加1，当溢出时，就对DPH的内容进行加1，不影响标志CY。

【例3-6】　(DPH)＝23H，(DPL)＝0FFH。执行指令：

```
INC  DPTR
```

执行结果：(DPH)＝24H，(DPL)＝0。

5. 减1指令

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

这组指令的功能是将工作寄存器Rn、片内RAM单元中的内容、间接地址存储器中的8位无符号二进制数和累加器A的内容减1，相减的结果仍存放在原单元中。

这类指令位不影响各个标志。

【例3-7】　(6FH)＝30H。执行指令：

```
DEC  6FH
```

执行结果：(6FH)＝2FH。

6. 乘法指令

```
MUL  AB
```

乘法指令的功能是将A和B中两个无符号8位二进制数相乘，所得的16位积的低8位存于A中，高8位存于B中。如果乘积大于255时，即高位B不为0时，OV置位；否则OV置0。CY总是清0。

【例3-8】　设(A)＝55H(85D)，(B)＝17H(23D)。执行指令：

```
MUL  AB
```

即85×23＝1955＝7A3H。

执行结果：(A)＝0A3H，(B)＝07H，(OV)＝1，(CY)＝0。

7. 除法指令

```
DIV  AB
```

除法指令的功能是将A中无符号8位二进制数除以B中的无符号8位二进制数，所得商的二进制数部分存于A，余数部分存于B中，并将CY和OV置0。当除数(B)＝0时，结果不定，则OV置1。CY总是清0。

【例3-9】　设(A)＝5CH(92D)，(B)＝05H(5)。执行指令：

```
DIV  AB
```

执行结果：(A)＝12H(商为18)，(B)＝2H(余数为2)，(OV)＝0，(CY)＝0。

8. 十进制调整指令

```
DA  A
```

BCD 码采用 4 位二进制数编码，并且只采用了其中的十个编码，即 0000～1001，分别代表 BCD 码 0～9，而 1010～1111 为无效码。当相加结果大于 9，说明已进入无效编码区；当相加结果有进位，说明已跳过无效编码区。凡结果进入或跳过无效编码区时，结果是错误的，相加结果均比正确结果小于 6(差 6 个无效编码)。

十进制调整的修正方法为：

(1) 当累加器低 4 位大于 9 或半进位标志 AC＝1 时，则进行低 4 位加 6 修正

```
(A0～3) + 6→(A0～3),即(A) = (A) + 06
```

(2) 当累加器高 4 位大于 9 或进位标志 CY＝1 时，进行高 4 位加 6 修正

```
(A4～7) + 6→(A4～7),即(A) = (A) + 60H
```

【例 3-10】 设(A)＝0101 0110＝56 BCD，(R3)＝ 0110 0111＝67 BCD，(CY)＝1。执行下述二条指令：

```
ADDC   A, R3
DA     A
```

执行：ADDC A，R3

```
          0 1 0 1 0 1 1 0   (56 BCD)
          0 1 1 0 0 1 1 1   (67 BCD)
                    1
        ---------------
          1 0 1 1 1 1 1 0   (高、低 4 位均大于 9)，
```

再执行：DA　A

```
              0 1 1 0 0 1 1 0   (加 66H 操作)
        ---------------
CY＝1   0 0 1 0 0 0 1 0 0   (124 BCD)
```

即 BCD 码数 56＋67＋1＝124。经十进制调整指令校正后，答案正确。

3.3.2 逻辑运算类指令

1. 逻辑“与”运算指令

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

这条指令对源操作数和目的操作数按位执行逻辑与操作，并将结果存在目的操作数中，结果不影响 PSW 中的标志位。

【例 3-11】 设(A)＝D3H (11010011B)，(R2)＝75H(01110101B)。执行指令：

```
ANL  A,  R2
```

执行结果：(A)＝51H(01010001B)。

【例 3-12】 设 P1＝0FFH。执行指令：

```
ANL  P1,  #10111001B
```

执行结果：P1＝0B9H(10111001B)。

该指令将 P1 口的 bit5，bit2 和 bit1 清零，其余位保持不变，逻辑“与”运算指令用做清除或屏蔽某些位。

2. 逻辑“或”运算指令

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

这条指令对源操作数和目的操作数按位执行逻辑或操作，并将结果存在目的操作数中，结果不影响 PSW 中的标志位。

【例 3-13】 设(A)＝D3H (11010011B)，(R2B)＝75H(01110101B)。执行指令：

```
ORL  A,  R2
```

执行结果：(A)＝0F7H(11110111B)。

【例 3-14】 设 P1＝0FH。执行指令：

```
ORL  P1,    #11000010B
```

执行结果：P1＝OCFH(11001111B)。

这条指令将 P1 口的 bit7，bit6 和 bit1 置 1，其他位保持不变。

3. 逻辑“异或”运算指令

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

该指令功能是将目的地址单元中的数和源地址单元中的数按“位”相“异或”(相同为零，相异为 1)，其结果放回目的地址单元中。

【例 3-15】 设(A)＝C3H(1100 0011B)，(R0)＝ AAH(1010 1010B)。执行指令：

```
XRL  A,  R0
```

$$\begin{array}{rr} & 1\,1\,0\,0\,0\,0\,1\,1 \\ \oplus & 1\,0\,1\,0\,1\,0\,1\,0 \\ \hline & 0\,1\,1\,0\,1\,0\,0\,1 \end{array}$$

执行结果：(A)＝69H(01101001B)。

逻辑异或指令用于对目的操作数的某些位取反，也可以判两个数是否相等，若相等则结

果为 0。

4. 累加器清零与取反操作指令

1) 累加器清零指令

```
CLR  A
```

该指令对累加器 ACC 进行清 0,此操作不影响标志位。

【例 3-16】 设(A)=0C3H,执行指令:

```
CLR  A
```

执行结果:(A)= 00H。

2) 累加器按位取反指令

```
CPL  A
```

该指令对进行累加器 ACC 的内容逐位取反,结果仍存在 A 中。此操作不影响标志位。

【例 3-17】 设(A)=C1H(1100 0001B),执行指令:

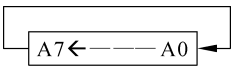
```
CPL  A
```

执行结果:(A)=3EH (0011 1110B)。

5. 循环左移指令

```
RL  A
```

该指令将累加器 A 的内容逐位循环左移一位,并且 A7 的内容移到 A0,此操作不影响标志位,如图 3.6 所示。



RL A

图 3.6 循环左移操作示意图

【例 3-18】 设(A)=43H(01000011B),执行指令:

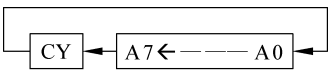
```
RL  A
```

执行结果:(A)=86H(10000110B)。

6. 带进位循环左移指令

```
RLC  A
```

该指令将累加器 A 的内容和进位位一起循环左移一位,并且 A7 移入进位位 CY,CY 的内容移到 A0,此操作不影响 CY 之外的标志位,如图 3.7 所示。



RLC A

图 3.7 带进位循环左移指令操作示意图

【例 3-19】 设(A)＝43H(01000011B),C＝1。执行指令：

```
RLC  A
```

执行结果：(A)＝87H(10000111B),C＝0。

7. 循环右移指令

```
RR  A
```

该指令功能是将累加器 A 的内容逐位循环右移一位,并且 A0 的内容移到 A7,此操作不影响标志位。

【例 3-20】 设(A)＝C3H(11000011B)。执行指令：

```
RR  A
```

执行结果：(A)＝E1H(11100001B)。

8. 带进位循环右移指令

```
RRC  A
```

功能是将累加器 A 的内容和进位位一起循环右移一位,并且 A0 移入进位位 CY,CY 的内容移到 A7,此操作不影响 CY 之外的标志位。

【例 3-21】 设(A)＝43H(01000011B),C＝1。执行指令：

```
RRC  A
```

执行结果：(A)＝A1H(10100001B),C＝1。

9. 累加器半字节交换指令

```
SWAP  A
```

功能是将累加器 A 的高半字节(ACC.7～ACC.4)和低半字节(ACC.3～ACC.0)互换。

【例 3-22】 设(A)＝43H(01000011B)。执行指令：

```
SWAP  A
```

执行结果：(A)＝34H(00110100B)。

3.3.3　数据传送类指令

数据传送类指令用到的助记符有：MOV、MOVX、MOVC、XCH、XCHD、PUSH、POP、SWAP。指令助记符不分大小写。一般数据传送指令的助记符用 MOV 表示。

(1) 格式：MOV　[目的操作数],[源操作数]；

(2) 功能：(目的操作数)←(源操作数中的数据)；

(3) 源操作数可以是：A、＃data、direct、Rn 、@Ri；

(4) 目的操作数可以是：A、direct、Rn、@Ri。

数据传送类指令是把源操作数传送到目的操作数。指令执行之后,源操作数不改变,目的操作数修改为源操作数,所以数据传送类操作属“复制”性质,而不是“搬家”。

本类指令不影响程序状态字 PSW 标志位：CY、AC 和 OV，但影响奇偶标志位 P。

1. 内部 RAM 单元之间的数据传送

1）以累加器为目的操作数的指令

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

该指令把源操作数内容送到累加器 A，源操作数有立即数寻址、直接寻址、寄存器寻址和寄存器间接寻址方式。

【例 3-23】 (R7)＝(38H)，(R0)＝40H，(30H)＝00H，(40H)＝0FFH，执行下列指令及执行结果如下：

```
MOV  A,  R7               ;(A) = 38H 寄存器寻址
MOV  A,  30H              ;(A) = 00H 直接寻址
MOV  A,  @R0              ;(A) = 0FFH 寄存器间接寻址
MOV  A,  ＃30H            ;(A) = 30H 立即数寻址
```

2）以 Rn 为目的操作数的指令

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

该指令把源操作数送入当前寄存器区的 R0～R7 中的某一寄存器。

【例 3-24】 执行下列指令及执行结果如下：

```
MOV  R1,＃53H             ;(R1) = 53H,   立即数寻址
MOV  R3,40H               ;R2←(40H),  直接寻址
MOV  R7,A                 ;R7←(A),    寄存器寻址
```

3）以直接地址为目的操作数的指令

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

该指令把源操作数送入直接地址指定的存储单元。direct 指的是内部 RAM 或 SFR 地址。

4）以寄存器间接地址为目的操作数的指令

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

该指令把源操作数送入寄存器间接寻址指定的存储单元中。

【例 3-25】 若(30H)＝35H,(R1)＝70H,执行指令：

```
MOV  @R1,30H
```

执行结果：RAM(70H)＝35H,同时(30H)＝35H,(R1)＝70H 不变。

5) 16 位数传送指令

```
MOV   DPTR,＃data16        ;(DPTR)←＃data16
```

该指令是把 16 位立即数送入 DPTR,用来设置数据存储器的地址指针。

【例 3-26】 执行指令：

```
MOV   DPTR,＃1234H
```

执行结果：(DPH)＝12H,(DPL)＝34H。

6) 堆栈操作指令

在 51 系列单片机内部 RAM 中可以设定一个后进先出的堆栈,地址为 30H～7FH 或 30H～0FFH 中,堆栈指针 SP 中的内容总是堆栈区中最后一个进栈数据所在的存储单元地址。堆栈操作包括进栈和出栈两种。

(1) 压栈指令：

```
PUSH   direct;    SP←SP＋1(SP)←(direct)
```

这条指令首先将堆栈指针 SP＋1,然后把直接地址里的内容传送到堆栈指针 SP 指出的内部 RAM 存储单元中。

(2) 出栈指令：

```
POP    direct;   ((SP))→direct   SP←SP－1
```

这条指令的功能是将堆栈指针 SP 指出的内部 RAM 单元的内容送入直接地址指出的存储单元中,堆栈指针 SP 减 1。出栈指令用于恢复 CPU 现场。

【例 3-27】 设(SP)＝30H,(ACC)＝60H,(B)＝70H,执行下列指令后结果怎样?

```
PUSH  ACC
PUSH  B
```

操作过程如下：

```
PUSH  ACC                 ;(SP)＋1,31H→SP,(ACC) →31H
PUSH  B                   ;(SP)＋1,32H→SP,(B)→32H
```

执行结果：(31H)＝60H,(32H)＝70H,(SP)＝32H。

【例 3-28】 设(SP)＝32H,(32H)＝70H,(31H)＝60H,执行下述指令：

```
POP DPH                   ;((SP)) →DPH,(SP)－1→SP
POP DPL                   ;((SP)) →DPL,(SP)－1→SP
```

执行结果：(DPH)＝70H,(DPL)＝60H,所以,DPTR＝7060H,(SP)＝30H。

片内 RAM 的数据传输指令源操作数、目的操作数操作示意图如图 3.8 所示。

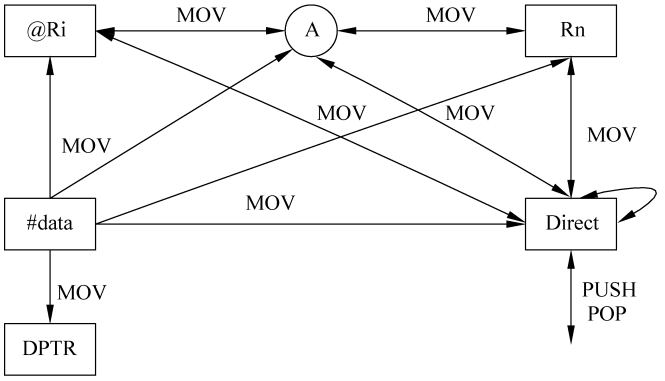


图 3.8　片内 RAM 的数据传输操作示意图

2. 累加器 A 与外部 RAM 单元之间的数据传送

1）读外部 RAM 单元

```
MOVX    A,@DPTR          ;((DPTR))→A,读外部 RAM/IO
MOVX    A,@Ri            ;((Ri))→A,读外部 RAM/IO
```

2）写外部 RAM 单元

```
MOVX    @DPTR,A          ;(A)→((DPTR)),写外部 RAM/IO
MOVX    @Ri,A            ;(A)→((Ri)),写外部 RAM/IO
```

MOV 的后面加“X”,表示访问的是片外 RAM 或 I/O 接口,在执行前读片外 RAM 指令,(P3.7)有效；后两条指令,(P3.6)有效。采用 16 位的 DPTR 间接寻址,可寻址整个 64KB 片外数据存储器空间,高 8 位地址(DPH)由 P2 口输出,低 8 位地址(DPL)由 P0 口输出。采用 Ri(i=0,1)进行间接寻址,可寻址片外 256 个单元的数据存储器。8 位地址由 P0 口输出。

【例 3-29】 (R0)=30H,(R1)=31H,(30H)=12H,(31H)=34H。执行指令：

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

执行结果：(A)=34H,(30H)=34H,(31H)=34H。

3. 查表指令

查表指令有两条,指令的助记符都是在 MOVC,“C”是 CODE 的第一个字母,即表示程序存储器中的代码。

1）以 PC 为基地址查表

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

该指令以 PC 作为基址寄存器,A 的内容(无符号数)和 PC 的当前值(下一条指令的起始地址)相加后得到一个新的 16 位地址,把该地址的内容送到 A。

【例 3-30】 (A)=50H,程序区(1051)=5AH,执行地址 1000H 处的指令。执行指令：

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

执行结果：(A)=5AH,(PC)=1001H。

该指令占用一个字节，下一条指令的地址为1001H，(PC)＝1001H再加上A中的50H，得1051H，结果把程序存储器中1051H的内容送入累加器A。

此指令的优点是不改变特殊功能寄存器及PC的状态，根据A的内容就可以取出表格中的常数。

此指令的缺点是表格只能存放在该指令所在地址的＋256个单元之内，表格大小受到限制，且表格只能被一段程序所用。

2）以DPTR为基地址查表

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

DPTR为基址寄存器，A的内容(无符号数)和DPTR的内容相加得到一个16位地址，把由该地址指定的程序存储器单元的内容送到累加器A。

【例3-31】 (A)＝50H，(DPTR)＝1000H程序区(1050)＝12H。执行指令：

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

执行结果：(A)＝12H，(PC)＝1001H。

将程序存储器中1050H单元内容送入A中。

本指令执行结果只与指针DPTR及累加器A的内容有关，与该指令存放的地址及常数表格存放的地址无关，因此表格的大小和位置可以在64KB程序存储器空间中任意安排，一个表格可以为各个程序块公用。

4．数据交换指令

1）字节交换指令

```
XCH    A,Rn                ;(A) ←→( Rn),n = 0～7
XCH    A,direct            ;(A) ←→(direct)
XCH    A,@Ri               ;(A) ←→(( Ri)),i = 0,1
```

这类指令的功能是将累加器A与源操作数的字节内容互换。

【例3-32】 设(A)＝80H，(R7)＝09H，(40H)＝0F0H，(R0)＝30H，(30H)＝0FEH。求连续执行下列指令后的结果：

执行下列指令　　执行结果：

```
XCH   A,R7                ;(A) = 09H,(R7) = 80H
XCH   A,40H               ;(A) = 0F0H,(40H) = 09H
XCH   A,@R0               ;(A)  = 0FEH,(30H) = 0F0H
```

2）半字交换

```
XCHD   A,@Ri
```

累加器的低4位与Ri间接寻址指定的内部RAM的低4位交换，而它们的高4位内容均不变。

【例3-33】 设(R0)＝20H，(A)＝36H(00110110B)，内部RAM中(20H)＝75H(01110101B)。执行指令：

```
XCHD   A,@R0
```

执行结果：(20H)＝01110110B＝76H，(A)＝00110101B＝35H。

3.3.4 控制转移类指令

1. 无条件转移指令

1）绝对转移指令

```
AJMP    addr11
```

这是2KB范围内的无条件跳转指令，指令构造转移目的地址，功能：PC加2，然后把指令中的11位无符号整数地址addr11（A10～A0）送入PC.10～PC.0，PC.15～PC.11保持不变，形成新的16位转移目的地址。

指令提供11位地址A10～A0（即addr11），其中A10～A8则位于第1字节的高3位，A7～A0在第2字节。操作码只占第1字节的低5位。

第1字节	A10	A9	A8	0	0	0	0	1
第2字节	A7	A6	A5	A4	A3	A2	A1	A0

需注意，目标地址必须与AJMP指令的下一条指令首地址的高5位地址码A15～A11相同，否则将混乱。所以是2KB范围内的无条件跳转指令。

该指令的操作过程可表示为：PC←(PC)＋2；PC10～0←addr11。例如：

```
9A00H    AJMP  LOOP                    ;指令代码：1030H,PC = 9A30H
9A02H
          …
9A30H    LOOP: ADD  A,  ＃30H
9A32H     …
```

2）相对转移指令

```
SJMP  rel
```

该指令先将PC＋2，再把指令中带符号的偏移量加到PC上，得到跳转的目标地址送入PC

$$目标地址＝源地址＋2＋rel$$

该指令为无条件转移，rel为相对偏移量，是一个字节的带符号8位二进制补码数，因此程序转移是双向的。rel如为正，向地址增大的方向转移；rel如为负，向地址减小的方向转移。

编程时，只需写上目的地址标号，相对偏移量由汇编程序自动计算。例如：

```
1000H    LOOP: MOV   A,R6
1002H     …
1030H    SJMP  LOOP
          …
```

汇编时，跳到LOOP处的偏移量由汇编程序自动计算和填入。例如：

```
START:   SJMP  START  等价于SJMP  $
```

$等价于当前语句的地址。

3）长跳转指令

```
LJMP    addr16          ;PC←addr16
```

执行该指令，可以将16位目标地址addr16装入PC，程序无条件转向指定的目标地址。转移指令的目标地址可在64KB程序存储器地址空间的任何地方，不影响任何标志。

4）间接转移指令（散转指令）

```
JMP  @A+DPTR            ;PC←(A)+(DPTR)
```

这是单字节转移指令，目的地址由A中8位无符号数与DPTR的16位无符号数内容之和来确定。以DPTR内容为基址，A的内容作为变址。给A赋予不同值，即可实现多分支转移。

【例3-34】 执行下列指令组后的PC值为多少？执行指令：

```
MOV   A,   #20H
MOV   DPTR,#1000H
JMP   @A+DPTR
```

执行结果：顺序执行完这3条指令后，PC＝1020H。

2．条件转移指令

执行指令时，如条件满足，则转移；不满足，则顺序执行下一指令。转移目的地址在以下一条指令首地址为中心的256B范围内（－128～＋127）。

```
JZ  rel                 ;(A)=0转移，否则顺序执行
JNZ   rel               ;(A)≠0转移，否则顺序执行
```

3．比较转移指令

在MCS-51中没有专门的比较指令，但提供了下面4条比较不相等转移指令；数值比较转移指令是三字节指令：

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

这组指令的功能是对指定的两操作数进行比较，即（操作数1）－（操作数2），但比较结果均不改变两个操作数的值，仅影响标志位CY。

若不等，程序转移到((PC)＋3)加上第三字节带符号的8位偏移量(rel)所指向的目标地址；

若(操作数1)＞(操作数2)，清进位标志(CY)。

若(操作数1)＜(操作数2)，则置位进位标志(CY)。

程序转移的范围是从((PC)＋3)为起始的＋127～－128B的单元地址。

4．减1不为0转移指令

```
DJNZ  Rn,   rel         ;n=0～7,Rn←(Rn)－1≠0转移
DJNZ  direct, rel       ;direct←(direct)－1≠0转移
```

该指令用于控制程序循环。Rn 或 direct 预先装入循环次数，执行该条指令，首先将 Rn 或 direct 里的内容减 1，然后判断 Rn 或 direct 里的内容是否为“0”作为转移条件，为“0”则转移，即实现按次数控制循环。例如：

```
START: MOV    A, #0FFH   ;(A)←0FFH
DL:    MOV    30H,#0AH   ;0AH→(30H)
DL1:   DJNZ   30H,  DL1  ;(30H)-1→(30H),(30H)不为0重复执行
       CPL    A          ;A取反
       MOV    P1,   A    ;送入P1
       AJMP   DL         ;转DL
```

5. 调用及返回指令

1）绝对调用指令

```
ACALL  addr11            ;(PC) ← (PC) + 2, (SP) ← (SP) + 1,
                         ;((SP)) ← (PC7～0), (SP) ← (SP) + 1 ,
                         ;((SP)) ← (PC15～8) ,(PC10～0) ← addr10～0
```

与 AJMP 指令类似，此指令为兼容 MCS-48 的 CALL 指令而设，不影响标志位。格式如下：

第 1 字节	A10	A9	A8	0	1	0	0	1
第 2 字节	A7	A6	A5	A4	A3	A2	A1	A0

2KB 范围内的调用子程序的指令。子程序地址必须与 ACALL 指令下一条指令的 16 位首地址中的高 5 位地址相同，否则将混乱。

【例 3-35】 若(SP)＝60H，在程序存储器地址 0123H 处有程序 ACALL SUBRTN，子程序 SUBRTN 位于 0345H，执行指令：

```
ACALL  SUBRTN
```

执行结果：(SP)＝62H，内部 RAM 中堆栈区内(61H)＝25H，(62H)＝01H，(PC)＝0345H。

2）长调用指令

```
LCALL  addr16            ;(PC) ← (PC) + 3 (SP) ← (SP) + 1
                         ;((SP)) ← (PC7～0) (SP) ← (SP) + 1
                         ;((SP)) ← (PC15～8) (PC) ← addr15～0
```

这条指令实现无条件调用程序存储器 16 位地址(64KB 范围内)的任何一个子程序。

功能先把 PC 加 3 获得下一条指令的地址(断点地址)，并压入堆栈(先低位字节，后高位字节)，堆栈指针加 2。接着把指令的第二和第三字节(A15～A8，A7～A0)分别装入 PC 的高位和低位字节中，然后从 PC 指定的地址开始执行程序。执行后不影响任何标志位。

【例 3-36】 若(SP)＝60H，标号 STRT 所在位置为 0100H，标号 DIR 所在位置为 8100H，执行指令：

```
STRT: LCALL  DIR
```

执行结果：(SP)＝62H,(61H)＝03H,(62H)＝01H,(PC)＝8100H。

3）子程序返回指令

```
RET                    ;((SP))→(PC15～8),然后(SP)－1→SP
                       ;((SP))→(PCL7～0),然后(SP)－1→SP
```

子程序返回指令是把栈顶相邻两个单元的内容弹出送到PC,SP的内容减2,程序返回PC值所指的指令处执行。RET指令通常安排在子程序的末尾,使程序能从子程序返回到主程序。

4）中断返回指令

```
RETI                   ;((SP))→(PC15～8),然后(SP)－1→SP
                       ;((SP))→(PCL7～0),然后(SP)－1→SP
```

这时指令的功能与RET指令相类似,不同之处为该指令恢复了中断逻辑,可以接收与正在进行的中断响应相同优先级的其他中断,其他相同。通常安排在中断服务程序的最后。

5）空操作指令

```
NOP                    ;PC←PC＋1
```

空操作也是CPU控制指令,它不进行任何操作,只消耗一个机器周期的时间,不影响标志位。常用于程序的等待或时间的延迟。例如：

```
CLR P2.7               ;P2.7引脚设为低电平
NOP                    ;消耗1个机器周期的时间
NOP                    ;消耗1个机器周期的时间
NOP                    ;消耗1个机器周期的时间
NOP                    ;消耗1个机器周期的时间
SETB P2.7              ;P2.7引脚设为高电平的时间
```

3.3.5　位操作指令

1. 位数据传送指令

1）位送进位标志位：

```
MOV  C,   bit
```

2）进位标志位送：

```
MOV  bit,   C
```

这两条指令是把源操作数指定的位变量送到目的操作数指定处。一个操作数必须为进位标志,另一个可以是任何直接寻址位。不影响其他寄存器或标志位。

```
MOV    C,06H           ;(20H).6→CY
```

06H是位地址,20H是内部RAM字节地址。06H是内部RAM 20H字节D6的位

地址。

```
MOV   P1.0,C              ;CY→P1.0
```

2. 位变量修改指令

1）进位标志位清零

```
CLR  C                    ;CY←0
```

功能：进位标志位 CY 清零。

2）位清零

```
CLR  bit                  ;bit ←0
```

功能：bit 位清零。

3）进位标志位取反

```
CPL  C                    ;CY←(/CY)
```

功能：进位标志位 CY 取反。

4）位取反

```
CPL  bit                  ;bit ←(/bit)
```

功能：bit 位取反。

5）进位标志位置 1

```
SETB  C                   ;CY←1
```

功能：进位标志位 CY 置位。

6）位置 1

```
SETB  bit                 ;bit←1
```

功能：bit 位置位。

这组指令将操作数指定的位清 0、求反或置 1，不影响其他标志位。例如：

```
CLR  C                    ;CY 位清 0
CLR   03H                 ;0→(20H).3 位
CPL   09H                 ;0→(21H).1 位
SETB  P1.0                ;P1.0 = 1
```

3. 位变量逻辑与指令

1）进位标志位与位逻辑与

```
ANL    C,  bit            ;bit∧CY→CY
```

功能：指令先对直接寻址位与进位标志位 CY 进行“逻辑与”运算，结果送回进位标志位中。

2）进位标志位与位的非逻辑与

```
ANL    C,   /bit          ;/bit∧CY→CY
```

功能：指令先对直接寻址位求反，然后与进位标志位 CY 进行“逻辑与”运算，结果送回到进位标志位中。

4. 位变量逻辑或指令

1）进位标志位与位逻辑或

```
ORL  C,bit                ;bit∨CY→CY
```

指令直接寻址位与进位标志位 CY（位累加器）进行“逻辑或”运算，结果送回到进位标志位中。

2）进位标志位与位的非逻辑或

```
ORL  C,/bit               ;/bit∨CY→CY
```

指令先对直接寻址位求反，然后与位累加器（进位标志位）进行“逻辑或”运算，结果送回到进位标志位中。

5. 位变量条件转移指令

1）C 置位转移

```
JC     rel                ;如进位标志位 CY = 1,则转移 PC←(PC) + 2 + rel,否则顺序执行
```

2）C 清零转移

```
JNC    rel                ;如进位标志位 CY = 0,则转移 PC←(PC) + 2 + rel,否则顺序执行
```

3）位置位转移

```
JB     bit,rel            ;如直接寻址位 = 1,则转移 PC←(PC) + 3 + rel,否则顺序执行
```

4）位清零转移

```
JNB    bit,rel            ;如直接寻址位 = 0,则转移 PC←(PC) + 3 + rel,否则顺序执行
```

5）判位转移并清零

```
JBC    bit,rel            ;如直接寻址位 = 1,转移,并把寻址位清零,否则顺序执行
                          ;PC←(PC) + 3 + rel,bit←0
```

3.4　MCS-51 系列单片机指令汇总

3.4.1　51 系列单片机指令表

3.3 节按功能介绍了 51 系列单片机的指令，下面给出全部的指令助记符及功能简要说明，以及指令长度、执行时间和指令代码（机器代码），如表 3.2 所示。

表 3.2　51 系列单片机指令表

助记符		说明	字节数	执行时间（机器周期）	指令代码（机器代码）
1. 算术运算类					
ADD	A，＃data	立即数加到累加器	2	1	24H，data
ADD	A，direct	直接寻址字节内容加到累加器	2	1	25H，direct
ADD	A，Rn	寄存器内容加到累加器	1	1	28H～2FH
ADD	A，@Ri	间接寻址 RAM 内容加到累加器	1	1	26H～27H
ADDC	A，＃data	立即数加到累加器（带进位）	2	1	34H，data
ADDC	A，direct	直接寻址加到累加器（带进位）	2	1	35H，direct
ADDC	A，Rn	寄存器加到累加器（带进位）	1	1	38H～3FH
ADDC	A，@Ri	间接寻址 RAM 加到累加器（带进位）	1	1	36H～37H
SUBB	A，＃data	累加器减去立即数（带借位）	2	1	94H，data
SUBB	A，direct	累加器内容减去直接寻址字节（带借位）	2	1	95H，direct
SUBB	A，Rn	累加器内容减去寄存器内容（带借位）	1	1	98H～9FH
SUBB	A，@Ri	累加器内容减去间接寻址 RAM（带借位）	1	1	96H～97H
INC	A	累加器增 1	1	1	04H
INC	Rn	寄存器增 1	1	1	08H～0FH
INC	direct	直接寻址字节增 1	2	1	05H，direct
INC	@Ri	间接寻址 RAM 增 1	1	1	06H～07H
INC	DPTR	数据指针增 1	1	2	A3H
DEC	A	累加器减 1	1	1	14H
DEC	Rn	寄存器减 1	1	1	18H～1FH
DEC	direct	直接寻址字节减 1	2	1	15H，direct
DEC	@Ri	间接寻址字节 RAM 减 1	1	1	16H～17H
MUL	AB	累加器和寄存器 B 相乘	1	4	A4H
DIV	AB	累加器除以寄存器 B	1	4	84H
DA	A	累加器十进制调整	1	1	D4H
2. 逻辑操作类					
ANL	A，Rn	寄存器“逻辑与”到累加器	1	1	58H～5FH
ANL	A，direct	直接寻址字节“逻辑与”到累加器	2	1	55H，direct
ANL	A，@Ri	间接寻址 RAM“逻辑与”到累加器	1	1	56H～57H
ANL	A，＃data	立即数“逻辑与”到累加器	2	1	54H，data
ANL	direct，A	累加器“逻辑与”到直接寻址字节	2	1	52H，direct
ANL	direct，＃data	立即数“逻辑与”到直接寻址字节	3	2	53H，direct，data
ORL	A，Rn	寄存器“逻辑或”到累加器	1	2	48H～5FH
ORL	A，direct	直接寻址字节“逻辑或”到累加器	2	1	45H，direct
ORL	A，@Ri	间接寻址 RAM“逻辑或”到累加器	1	1	46H～47H
ORL	A，＃data	立即数“逻辑或”到累加器	2	1	44H，data
ORL	direct，A	累加器“逻辑或”到直接寻址字节	2	1	42H，data
ORL	direct，＃data	立即数“逻辑或”到直接寻址字节	3	2	43H，direct，data
XRL	A，Rn	寄存器“逻辑异或”到累加器	1	2	68H～6FH

续表

助记符		说明	字节数	执行时间（机器周期）	指令代码（机器代码）
2. 逻辑操作类					
XRL	A,direct	直接寻址字节“逻辑异或”到累加器	2	1	65H,direct
XRL	A,@Ri	间接寻址 RAM“逻辑异或”到累加器	1	1	66H～67H
XRL	A,#data	立即数“逻辑异或”到累加器	2	1	64H,dataH
XRL	direct,A	累加器“逻辑异或”到直接寻址字节	2	1	62H,direct
XRL	direct,#data	立即数“逻辑异或”到直接寻址字节	3	2	63H,direct,data
CLR	A	累加器清零	1	2	E4H
CPL	A	累加器求反	1	1	F4H
RL	A	累加器循环左移	1	1	23H
RLC	A	经过进位标志的累加器循环左移	1	1	33H
RR	A	累加器循环右移	1	1	03H
RRC	A	经过进位标志的累加器循环右移	1	1	13H
3. 数据传送类					
MOV	A,#data	立即数传送到累加器	2	1	74H, data
MOV	A,direct	直接寻址字节传到累加器	2	1	E5H, direct
MOV	A,Rn	寄存器内容传到累加器 A	1	1	E8H～EFH
MOV	A,@Ri	间接寻址 RAM 传到累加器	1	1	E6H～E7H
MOV	Rn,#data	立即数传送到 Rn	2	1	78H～7FH, data
MOV	Rn,direct	直接地址内容传送到 Rn	2	2	A8H～AFH,direct
MOV	Rn,A	累加器内容送到寄存器	1	1	F8H～FFH
MOV	direct,A	累加器内容传送到直接寻址字节	2	1	F5H, direct
MOV	direct,#data	立即数传送到直接寻址字节	3	2	75H～E7H
MOV	direct1,direct2	直接寻址字节 2 传送到直接寻址字节 1	3	2	85H,direct1,direct2
MOV	direct,Rn	寄存器内容传送到直接寻址字节	2	2	88H～8FH, direct
MOV	direct,@Ri	间接寻址 RAM 传送到直接寻址字节	2	2	86H～87H, direct
MOV	@Ri,#data	立即数传送到间接寻址 RAM	2	1	76H～77H,data
MOV	@Ri,direct	直接地址传传送到间接寻址 RAM	2	2	A6H～A7H,direct
MOV	@Ri,A	累加器传送到间接寻址 RAM	1	1	F6H～F7H
MOV	DPTR,#data16	16 位常数装入到数据指针	3	2	90H,dataH,dataL
MOVC	A,@A+DPTR	程序存储器代码字节传送到累加器	1	2	93H
MOVC	A,@A+PC	程序存储器代码字节传送到累加器	1	2	83H
MOVX	A,@Ri	外部 RAM(8 地址)传送到 A	1	2	E2H～E3H
MOVX	A,@DPTR	外部 RAM(16 地址)传送到 A	1	2	E0H
MOVX	@Ri,A	累加器传送到外部 RAM(8 地址)	1	2	F2H～F3H
MOVX	@DPTR,A	累加器传送到外部 RAM(16 地址)	1	2	F0H
PUSH	direct	直接寻址字节压入栈顶	2	2	C0H,direct
POP	direct	栈字节弹到直接寻址字节	2	2	D0H,direct
XCH	A,Rn	寄存器和累加器交换	1	1	C8H～CFH
XCH	A, direct	直接寻址字节和累加器交换	2	1	C5H, direct

续表

助记符		说明	字节数	执行时间（机器周期）	指令代码（机器代码）
3. 数据传送类					
XCH	A，@Ri	间接寻址 RAM 和累加器交换	1	1	C6H～C7H
XCHD	A，@Ri	间接寻址 RAM 和累加器交换低半字节	1	1	D6H～D7H
SWAP	A	累加器内高低半字节交换	1	1	C4H
4. 控制转移类					
ACALL	addr11	绝对调用子程序	2	2	a10a9a810001，addr(7～0)
LCALL	addr16	长调用子程序	3	2	12H，addr(15～8)，addr(7～0)
RET		子程序返回	1	2	22H
RETI		中断返回	1	2	32H
AJMP	addr11	绝对转移	2	2	a10a9a810001，addr(7～0)
LJMP	addr16	长转移	3	2	12H，addr(15～8)，addr(7～0)
SJMP	rel	短转移（相对偏移）	2	2	80H，rel
JMP	@A＋DPTR	相对 DPTR 的间接转移	1	2	73H
JZ	rel	累加器为零则转移	2	2	60H，rel
JNZ	rel	累加器为非零则转移	2	2	70H，rel
CJNE	A，direct，rel	比较直接寻址和 A，不相等则转移	3	2	B5H，direct，rel
CJNE	A，＃data，rel	比较立即数和 A，不相等则转移	3	2	B4H，data，rel
CJNE	Rn，＃data，rel	比较寄存器和立即数，不相等则转移	3	2	B8H～BFH，data，rel
CJNE	@Ri，＃data，rel	比较立即数和间接寻址 RAM，不相等则转移	3	2	B6H～B7H，data，rel
DJNZ	Rn，rel	寄存器减 1，不为零则转移	2	2	D8H～DFH，rel
DJNZ	direct，rel	地址字节减 1，不为零则转移	3	2	D5H，direct，rel
NOP		空操作	1	1	00H
5. 位操作类					
CLR	C	进位标志位清“0”	1	1	C3H
CLR	bit	直接寻址位清“0”	2	1	C2H，bit
SETB	C	进位标志位置“1”	1	1	D3H
SETB	bit	直接寻址位置“1”	2	1	D2H，bit
CPL	C	进位标志位取反	1	1	B3H
CPL	bit	直接寻址位取反	2	1	B2H，bit
ANL	C，bit	直接寻址位“逻辑与”到进位标志位	2	2	82H，bit
ANL	C，/bit	直接寻址位的反码“逻辑与”到进位标志位	2	2	B0H，bit
ORL	C，bit	直接寻址位“逻辑或”到进位标志位	2	2	72H，bit

续表

助　记　符		说　　明	字节数	执行时间（机器周期）	指令代码（机器代码）
5. 位操作类					
ORL	C,/bit	直接寻址位的反码“逻辑或”到进位标志位	2	2	A0H,bit
MOV	C,bit	直接寻址位传送到进位标志位	2	1	A2H,bit
MOV	bit, C	进位标志位传送到直接寻址标志位	2	2	92H,bit
JC	rel	进位标志位为1则转移	2	2	40H,rel
JNC	rel	进位标志位为零则转移	2	2	50H,rel
JB	bit,rel	直接寻址位为1则转移	3	2	20H,bit,rel
JNB	bit,rel	直接寻址位为零则转移	3	2	30H,bit,rel
JBC	bit,rel	直接寻址位为1则转移,并清除该位	3	2	10H,bit,rel

3.4.2　指令中关于累加器 A 与 ACC 的区别

累加器可写成 A 或 ACC,区别是什么?

A 代表寄存器寻址的一个特殊功能寄存器；而 ACC 代表直接寻址方式中的 ACC 的地址是 E0H，A 汇编后则隐含在指令操作码中,ACC 汇编后地址是 E0H,例如：

```
INC    A                ;指令代码 04H
INC    ACC              ;相当于 INC direct,指令代码 05H、E0H
MOV    A,    30H        ;指令代码 E5H、30H,目的操作数是寄存器寻址
MOV    ACC,  30H        ;指令代码 85H、E0H、30H 目的操作数是直接寻址
```

因此,在对累加器 A 直接寻址和累加器 A 的某一位寻址要用 ACC,不能写成 A。如压栈和出栈指令是 PUSH Direct 和 POP Direct,因此要对累加器进行压栈和出栈时要用以下指令：

```
PUSH   ACC
POP    ACC
```

而不能用

```
PUSH   A
POP    A
```

3.4.3　指令中关于字节地址和位地址的区分

如何区别指令中出现的字节变量和位变量？例如以下指令：

```
MOV  20H,C
MOV  20H, A
```

这两条指令中的目的操作数 20H 都是以直接地址形式给出的,20H 是字节地址还是位地址,由于前一条指令中的 C 是位变量,因此指令中的 20H 是位地址,后一条指令的一个操

作数是 A 寄存器，是字节变量，所以该条指令中的 20H 为字节地址。

3.5 汇编语言程序设计基础

程序是指令的有序集合。单片机运行就是执行指令序列的过程。编写这一指令序列的过程称为程序设计。本节介绍采用汇编语言进行程序设计。

3.5.1 汇编语言程序设计概述

1. 程序设计语言

常用的编程语言是汇编语言和高级语言。

1）汇编语言

在汇编语言中，可以用于代替机器语言的英文字符被称为助记符。汇编语言就是用助记符表示的指令；汇编语言源程序是指用汇编语言编写的程序。

汇编语言是一种低级语言，它依赖于机器，要求必须对硬件有相当深的了解。它有机器语言的优点，占用内存少，执行速度快，适合于实时控制。但离不开具体的硬件，是面向“硬件”的语言，通用性差。因此，用汇编语言编写程序效率高，占用存储空间小，运行速度快，能编写出最优化的程序，但可读性差，

汇编语言具有以下几个特点：

（1）助记符指令与机器指令是一一对应的，所以用汇编语言编写的程序效率高，占用存储空间小，运行速度快，而且能反映计算机的实际运行情况，所以用汇编语言能编写出最优化的程序。

（2）汇编语言能直接访问存储器、输入与输出接口及扩展的各种芯片（比如 A/D、D/A 等），也可直接处理中断，因此汇编语言能直接管理和控制硬件设备。

（3）汇编语言与机器语言一样，脱离不开具体的机器硬件，都是面向机器的语言，缺乏通用性。

2）高级语言

高级语言采用更接近人类语言和习惯的数学表达式及直接命令的方法来描述算法和过程。高级语言是接近于人的自然语言，面向过程而独立于机器的通用语言。如 C 语言(C51)PL/M 语言用于进行 MCS-51 的程序设计。高级语言易学，通用性强；但程序质量较差，内存占用多，运行速度慢，适用于科学计算和信息管理。

2. 汇编语言源程序的汇编

单片机硬件能够识别的是由“0”“1”代码形式表示的二进制的机器语言，因为机器语言是计算机唯一能理解和执行的语言。汇编语言源程序需转换（翻译）成为二进制代码表示的机器语言程序，才能被识别和执行。因此，将汇编语言转换（翻译）为机器代码的程序称为汇编程序。经汇编程序“汇编”得到的以“0”“1”代码形式表示的机器语言程序称为目标程序。汇编就是把汇编语言翻译成机器代码的过程，汇编分为手工汇编和机器汇编。

通常把人工查表翻译指令的方法称为“手工汇编”。例如

```
MOV A,＃80H 对应着 74H,80H
```

用计算机代替手工汇编称作机器汇编。首先将汇编语言源程序输入到编辑软件中，生成了一个 ASC 码文件，扩展名为“.asm”，通过编译后生成机器语言文件“.hex”。

由一台计算机完成汇编后得到的机器代码在另一台计算机（这里是单片机）上运行，称这种机器汇编为交叉汇编。

在分析现成产品 ROM/EPROM 中的程序时，要将二进制数的机器代码语言程序翻译成汇编语言源程序，该过程称为反汇编。

3.5.2　汇编伪指令

在 3.3 节中介绍了基本的汇编语言指令，每条汇编语言指令都有机器代码与之对应。

在汇编语言源程序中应有向汇编程序发出的指示信息，告诉它如何完成汇编工作，这是通过伪指令来实现。伪指令不属于指令系统中的汇编语言指令，它是程序员发给汇编程序的命令，也称为汇编程序控制命令。只有在汇编前的源程序中才有伪指令。“伪”体现在汇编后，伪指令没有相应的机器代码产生。伪指令具有控制汇编程序的输入/输出、定义数据和符号、条件汇编、分配存储空间等功能。

1. 汇编起始地址命令：ORG

起始地址命令用来说明以下程序段在存储器中存放的起始地址。如果不规定程序存放的起始地址，默认的程序从地址 0 开始存放。例如：

```
ORG    1000H              ;该条语句放在程序存储器地址 1000H 处
START: MOV  A,#10H
       MOV  B,R0
```

在一个程序中可多次用到 ORG 伪指令，但规定由小到大顺序排列，且不应使程序有交叉和重叠。例如：

```
ORG  00H
…
ORG   03H
…
ORG  0BH
…
```

这种顺序是正确的。若按下面顺序的排列则是错误的，因为地址出现了交叉。

```
ORG  00H
…
ORG  0BH
…
ORG  03H
…
```

2. 汇编结束命令：END

END 是汇编语言源程序的结束标志。

汇编时遇到 END 就停止汇编，故该伪指令放在源程序结尾。如果 END 出现在程序中间，其后的源程序，将不进行汇编处理。

3. 赋值，等值指令：EQU

EQU 用于给标号赋值。赋值后，标号值在整个程序有效。

格式：字符　EQU　数值

这个命令使指令中的字符名称等价于给定的数。例如：

```
TAB    EQU   1200H
       MOV   DPTR,#TAB
```

相当于

```
       MOV   DPTR,#1200H
```

EQU 伪指令只能对某标号赋值一次，该标号在同一个程序中不能再一次赋值。

4. 定义数据字节：DB

DB 命令把数据以字节数的形式存放在存储器单元中，通常用于从指定的地址开始，在程序存储器连续单元中定义字节数据。例如：

```
ORG    500H
DB     20H,  'AB',  15
```

汇编后

(500H)＝20H

(501H)＝41H(字符“A”的 ASCII 码)

(502H)＝42H(字符“B”的 ASCII 码)

(503H)＝0EH(十进制的 15)

5. 定义数据字：DW

DW 命令按字的形式把数据存放在存储单元中。与 DB 相似，但 DW 定义的是 16 位数据，占用 2 个字节，汇编时 DW 按高字节在前存放，低字节在后存放。标号也可以，但事先必须赋值。例如：

```
ORG    1000H
DATA   EQU  2316H
TAB:   DW   2104H, 10
       DW   DATA
(1000H) = 21H              ;第 1 个字
(1001H) = 04H
(1002H) = 00H              ;第 2 个字
(1003H) = 0AH
(1004H) = 23H              ;第 3 个字
(1005H) = 16H
```

需要注意这里第 2 个字是 10，它的高字节为 0。

6. 定义存储区：DS

DS 命令从 ROM 的指定地址开始，保留若干个字节作备用。例如：

```
ORG    1000H
DS     05
```

```
DB      88H
```

则 1000H～1004H 这 5 个单元保留，而 1005H 中存放 88H。

7. 位定义：BIT

位定义是指把位地址赋给字符名称。例如：

```
A1      BIT     P1.0
A2      BIT     0H
CLR     A1                  ;P1.0 = 0;
SETB    A2                  ;(20H).0 = 1
```

3.5.3　汇编语言源程序的汇编

汇编是将汇编语言源程序翻译成机器代码的过程，可分为手工汇编和机器汇编两类。

1. 手工汇编

通过查指令的机器代码表(表 3.2)，逐个把助记符指令“翻译”成机器代码，再进行调试和运行。

手工汇编遇到相对转移偏移量的计算时，较麻烦，易出错，只有小程序或受条件限制时才使用。实际中，多采用“汇编程序”来自动完成汇编。

2. 机器汇编

用微型计算机上的软件(汇编程序)来代替手工汇编。在微机上用编辑软件进行源程序编辑，然后生成一个 ASCII 码文件，扩展名为“.ASM”。在微机上运行汇编程序，译成机器码。机器码通过微机的串口(或并口)传送到用户样机(或在线仿真器)，进行程序的调试和运行。有时，在分析某些产品的程序的机器代码时，需将机器代码翻译成汇编语言源程序，称为“反汇编”。

3.6　汇编语言程序设计的基本方法

在单片机的应用程序设计中，要尽量采用模块化的编程方法，把具有相同功能的程序设计成子程序，这种采用子程序和主程序的设计方法便于程序设计和调试，利于程序的优化和分工，提供程序的可读性和可靠性。

功能复杂的程序结构常采用以下几种基本结构：顺序结构、分支结构和循环结构等。这样可以使程序具有结构清晰、可读性好、可移植性强等优点。

3.6.1　顺序结构

顺序结构程序是一种最简单、最基本的程序。特点是程序按编写的顺序依次往下执行每一条指令，直到最后一条，无分支，无循环，不调用子程序，程序流向不变。

【例 3-37】 编写 16 位数加法，一个加数放在 R1(高位)，R0(低位)中，另一个加数放在 R3(高位)，R2(低位)中，和放在 R1(高位)，R0(低位)中。

分析：加法指令是在累加器 A 中完成的，其中一个加数及和都放在 A 中，因此要将一个加数取到 A，加法指令是 ADD，在高位相加的时候要加上低位的进位位，因此用 ADDC。

参考程序如下：

```
        ORG   00H                ;单片机复位时 PC = 0,程序从地址 0 开始执行,
        LJMP  START              ;跳过中断入口区
        ORG   100H               ;实际程序存放的地址
START:  MOV   SP,  #60H          ;堆栈默认为 07H,重新设置堆栈指针,避开寄存器区
        MOV   A,R0               ;取出 R0 中的低位数据
        ADD   A,R2               ;与另一个数据的低位相加
        MOV   R0,A               ;将和存在 R0 中
        MOV   A,R1               ;取出 R1 中的高位数据
        ADDC  A,R3               ;与另一个数据的高位相加
        MOV   R1,A               ;存高位数据
        SJMP $                   ;程序必须是一个循环,防止程序一直运行
        END                      ;汇编结束。
```

【例 3-38】 编写 16 位二进制负数求补程序，数据放在 31H30H 中，结果放在 33H32H 中。

二进制数的求补可归结为“求反加 1”的过程，求反可用 CPL 指令实现；加 1 时应注意，加 1 只能加在低 8 位的最低位上。因为现在是 16 位数，有两个字节，因此要考虑进位问题，即低 8 位取反加 1，高 8 位取反后应加上低 8 位加 1 时可能产生的进位，还要注意这里的加 1 不能用 INC 指令，因为 INC 指令不影响 CY 标志。参考程序如下：

```
        ORG   00H                ;单片机复位时 PC = 0,程序从地址 0 开始执行
        LJMP  START              ;跳过中断入口区
        ORG   100H               ;实际程序存放的地址
START:  MOV   SP, #60H           ;堆栈默认为 07H,重新设置堆栈指针,避开寄存器区
        MOV   R0,  #30H          ;设置存放数据的地址指针
        MOV   A,   @R0           ;取出 30H 的低位数据
        CPL   A                  ;取反
        ADD   A, #1              ;加 1
        INC   R0                 ;指向 31H
        INC   RO                 ;指向 32H
        MOV   @R0, A             ;存低位数据
        DEC   R0                 ;指向 31H
        MOV   A,@R0              ;取出 31H 中存放的高位数据
        CPL   A                  ;取反
        ADDC  A,#0               ;加上低位的进位标志
        INC   R0                 ;指向 32H
        INC   RO                 ;指向 33H
        MOV   @R0, A             ;存高位数据
        SJMP $                   ;程序必须是一个循环,防止程序一直运行
        END                      ;汇编结束。
```

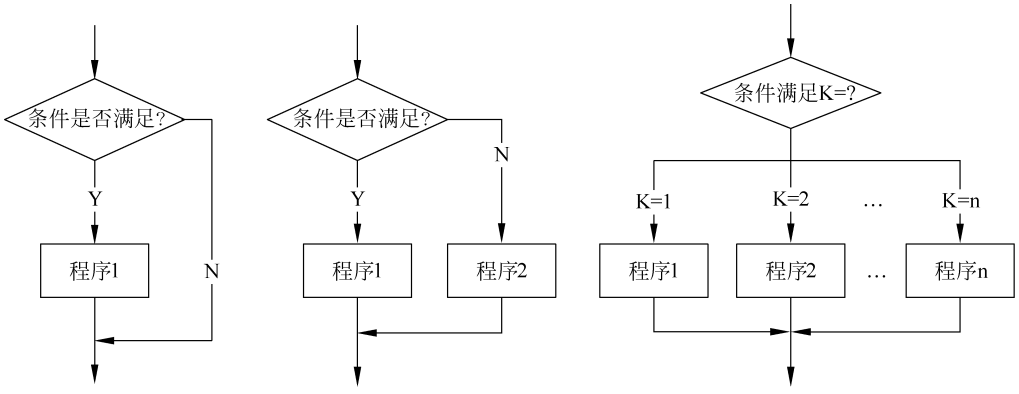
3.6.2 分支结构

程序运行过程中可能需要判断某个条件，当条件满足时执行一段程序，不满足时执行另一端程序，这时候就会用到分支结构程序。分支结构程序包含单分支和多分支结构程序，如图 3.9 所示。分支程序的设计要点如下：

(1) 先建立可供条件转移指令测试的条件；

（2）选用合适的条件转移指令；

（3）在转移的目的地址处设定标号。



(a) 一种单分支结构　(b) 另一种单分支结构　(c) 多分支结构

图 3.9　分支结构程序流程图

在 51 指令系统中条件转移指令有：

（1）判 A 转移指令 JZ、JNZ；

（2）判位转移指令 JB、JNB、JBC、JC、JNC；

（3）比较转移指令 CJNE；

（4）减 1 不为 0 转移指令 DJNZ。

1. 单分支结构

单分支结构仅有两个出口，两者选一。一般根据运算结果的状态标志，用条件判跳指令来选择并转移。

【例 3-39】 设内部 RAM 的 30H 和 31H 中各存放一无符号数，比较大小，将大数存放于 RAM40H 中，小数存放于 RAM41H 中，若两个数相等，则分别存放于这两个单元中。

```
        ORG   00H                    ;复位时 PC = 0,程序从地址 0 开始执行
START:  MOV   SP,   #60H             ;堆栈默认 07H,重新设置堆栈指针,避开寄存器区
        MOV   A, 30H                 ;取第一个数
        MOV   41H, 31H               ;默认第二个数是小的数
        CLR   C
        CJNE  A,31H,LOOP             ;两个数不等跳转,相等的话可以随便存
LOOP:   JNC   BIG
        XCH   A,   41H               ;A 小,存小数
BIG:    MOV   40H,  A                ;存大数
        SJMP   $                     ;暂停
        END
```

2. 多分支选择结构

当程序的判别部分有两个以上的出口时，为多分支选择结构，典型的多分支结构如图 3-9(c)所示。指令系统提供了非常有用的两种多分支选择指令：

（1）比较转移指令

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

```
CJNE  A,  ＃data,rel
CJNE  Rn, ＃data,rel
CJNE  @Ri,＃data,rel
```

（2）间接转移指令

```
JMP    @A＋DPTR
```

4 条比较转移指令 CJNE 能对两个比较的单元内容进行比较，当不相等时，程序实现相对转移；若两者相等，则顺序往下执行，同时第一个操作数大于第二个操作数时，清零 C；第一个操作数小于第二个操作数时置位 C。

间接转移指令 JMP　@A＋DPTR 由数据指针 DPTR 决定多分支转移程序的首地址，由 A 的内容选择对应分支。

【例 3-40】 根据图 3.10 所示的流程图编程实现：设无符号二进制数 X 存在 30H 单元中，比较 30H 中的数与 5AH 的大小，根据：

$$Y=\begin{cases}1 & \text{当 } X>5AH\\ 0 & \text{当 } X=5AH\\ -1 & \text{当 } X<5AH\end{cases}$$

求出 Y 值，将 Y 值存入 31H 单元。

分析：简单的分支转移程序的设计，常采用逐次比较法，就是把所有不同的情况一个一个地进行比较，发现符合就转向对应的处理程序。在编程之前画出程序流程图（见图 3.10），可以分析清楚程序的思路，这道题虽然是三个分支，实际是用两个单分支结构实现了三个分支。

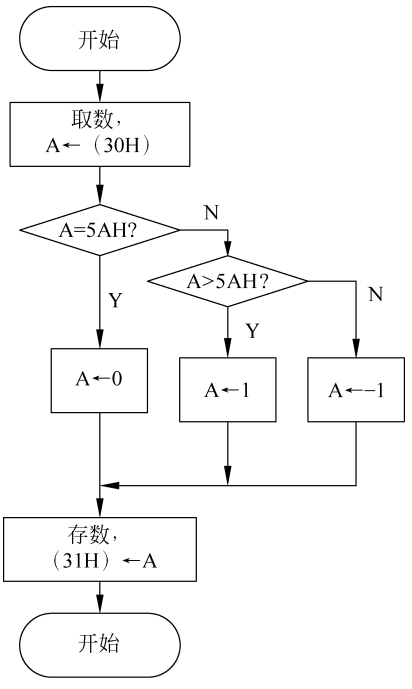


图 3.10　例 3-40 的分支结构程序流程图

```
        ORG   00H                    ;复位时 PC = 0,程序从地址 0 开始执行
START:  MOV   SP,   #60H             ;重新设置堆栈指针,避开寄存器区
        MOV   A,    30H              ;取数
        CJNE  A,    #5AH,NOT         ;不等跳转,且(A)> 5AH,C = 0, (A)< 5AH,C = 1
        MOV   A,    #0
        JZ    SAVE                   ;为零,转存储
NOT:    JC    ACC.7,NEG              ;C = 1,表示(A)< 5AH,转 NEG
        MOV   A,#1H                  ;C = 0 ,表示(A)> 5AH,存 1
        AJMP  SAVE                   ;转到 SAVE,保存数据
NEG:    MOV   A,   #81H              ;Y = - 1
SAVE:   MOV   31H,A                  ;保存数据
        SJMP   $                     ;暂停
        END
```

【例 3-41】 设片内变量 x 是一个 0～3 的无符号数,存在于片内 RAM 20H 中,若 x＝0,执行 R0＝R0＋1,同理 x＝1,执行 R1＝R1＋1,x＝3,执行 R3＝R3＋1,程序执行后将 20H 的内容加一,若 20H 的值大于 3,则重新将其置 0。

解：利用“JMP @A＋DPTR”指令直接给 PC 赋值,实现程序转移,流程图见图 3.11。

```
        ORG   00H
        LJMP  START
START:  ORG   100H
        CLR   A
        MOV   20H,   A
LOOP:   MOV   A,   20H               ;取数
        MOV   B,     #03H            ;每条 LJMP 语句占 3 个字节
        MUL   AB
        MOV   R6,     A              ;暂存低位
        MOV   A,      B              ;取高位
        MOV   DPTR,#TAB              ;转移指令表首地址
        ADD   A,      DPH            ;高位地址加到 DPH
        MOV   DPH,   A               ;存高位地址
        MOV   A ,     R6             ;取暂存的低位地址,进行散转
        JMP   @A + DPTR              ;PC ← A + DPTR
TAB:    LJMP  PRG0                   ;转移指令表
        LJMP  PRG1
        LJMP  PRG2
        LJMP  PRG3
PROG0:  INC   R0
        AJMP  NEXT
PROG1:  INC   R1
        AJMP  NEXT
PROG2:  INC   R2
        AJMP  NEXT
PROG3:  INC   R3
NEXT:   INC   20H
        MOV   A,   20H
        ANL   A,   #3
        MOV   20H,  A
        SJMP  LOOP
        END
```

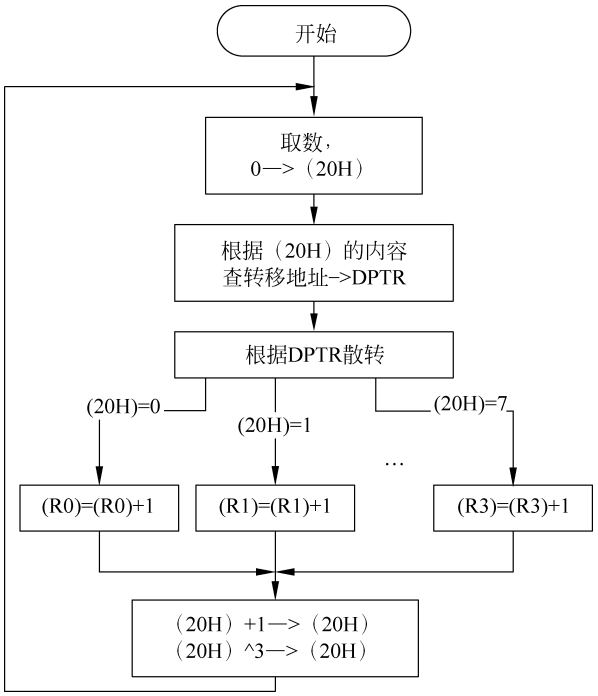


图 3.11　例 3-41 散转程序流程图

3.6.3　循环结构程序设计

在程序设计中，经常需要控制一段指令重复执行若干次，以便用简短的程序完成大量的处理任务，这种按照某种规律执行的程序成为循环程序。程序中含有可以反复执行的程序段，称循环体。例如，求 10 个数的累加和，没必要连续安排 10 条加法指令，用一条加法指令使其循环执行 10 次。因此可缩短程序长度和程序所占的内存单元数量更少，使程序结构紧凑。

1. 循环程序的结构

循环程序的结构主要由以下 4 部分组成。

（1）循环初始化：完成循环前的准备工作。例如，循环控制计数初值的设置、地址指针的起始地址的设置、为变量预置初值等。

（2）循环处理：完成实际处理工作，反复循环执行的部分，故又称循环体。

（3）循环控制：在重复执行循环体的过程中，不断修改循环控制变量，直到符合结束条件，就结束循环程序的执行。循环结束控制方法分为先执行后判断和先判断后执行两种。

（4）循环结束：这部分是对循环程序执行的结果进行分析、处理和存放。

2. 循环结构的控制

循环结构的控制分为先执行后判断和先判断后执行两种，如图 3.12 和图 3.13 所示。循环程序按结构形式可以分为单重循环与多重循环。循环程序的嵌套形式如图 3.14 所示。

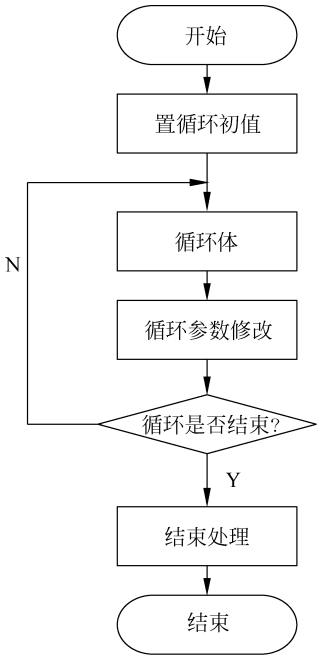


图 3.12　先执行后判断循环结构程序流程图

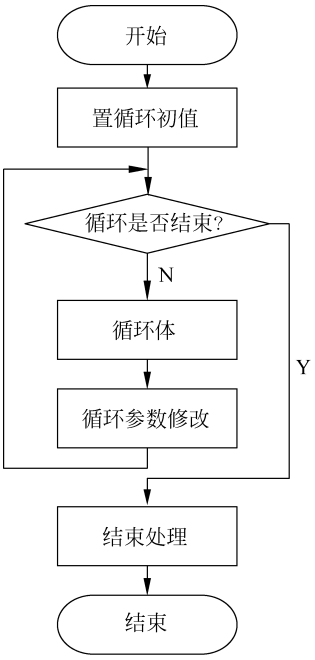
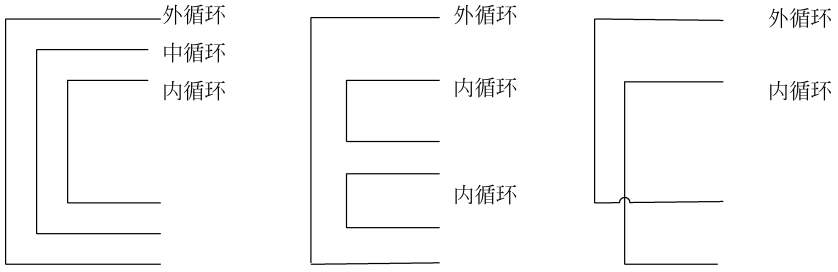


图 3.13　先判断后执行循环结构程序流程图



(a) 正确的嵌套　　(b) 正确的嵌套　　(c) 错误的嵌套

图 3.14　循环结构程序的嵌套形式

【例 3-42】 将片内 RAM 变量 20H～7FH 的内容清零，程序如下。

```
        ORG   00H
        LJMP  START
        ORG   100H
START:  MOV   R0,   #20H
        MOV   R2,   #60H              ;20H～7FH,共 60H 个单元
        CLR   A
LOOP:   MOV   @R0,  A
        INC   R0
        DJNZ  R2,   LOOP
        SJMP  $
        END
```

【例 3-43】 将片内 RAM 变量 30H 开始的数据串传送到片外 RAM 1000H 开始的地址，直到遇到“$”字符停止传送。

由于循环次数不知道，但是循环终止条件已知，该程序可采用先判断后执行的循环控制结构。

```
        ORG   00H
        LJMP  START
        ORG   100H
START:  MOV   R0,    #30H
        MOV   DPTR,  #1000H
LOOP:   MOV   A,     @R0
        CJNE  A,     #24H,   LOOP1      ;"$"字符的 ASCII 码是 24H
        SJMP  THEEND
LOOP1:  MOVX  @DPTR,A
        INC   R0,                       ;指向下一个单元
        INC   DPTR,
        SJMP  LOOP
THEEND: SJMP  $
        END
```

3.6.4 子程序及其调用

在实际应用中，经常会遇到一些通用性的问题，例如数值转换、数值计算等，经常要进行多次，这时可以将其设计成子程序供调用。采用子程序的设计方法可以使程序更紧凑，便于程序的阅读和调试。

子程序调用是暂时中断主程序的执行，而转到子程序的入口地址去执行子程序，子程序执行完再返回主程序执行。子程序的调用指令为 ACALL 和 LCALL，调用子程序应注意：

(1) 子程序第一条指令的地址为子程序的入口地址，该指令前必须有标号；

(2) 主程序调用子程序，是通过主程序或调用程序中的调用指令 ACALL 和 LCALL 实现的；

(3) 注意设置堆栈和保护现场；

(4) 子程序返回，最后一条指令必须是 RET；

(5) 子程序调用时，注意参数传递；

(6) 嵌套调用与递归调用，最多允许 8 层。

1. 现场的保护与恢复

在子程序调用中经常用到寄存器 A、B、DPTR 以及 PSW 等，这些单元主程序也在使用，因此需要进行现场保护，即压栈。在执行完子程序，返回主程序前需要恢复其内容，称为恢复现场，即出栈。保护现场的原则是先进后出，后进先出。保护与恢复现场的方法有两种：

(1) 主程序进行保护与恢复。

```
MAIN:   PUSH  ACC
        PUSH  PSW
        PUSH  B
```

```
        LCALL SUBPROG
        POP   B
        POP   PSW
        POP   ACC
```

(2) 子程序进行保护。

```
SUBPROG: PUSH  ACC
         PUSH  PSW
         PUSH  B
         …
         POP   B
         POP   PSW
         POP   ACC
         RET
```

实际中在子程序中现场保护,程序更规范和清晰,用得较多。

2. 参数的传递

参数的传递主要有以下 3 种方式:

(1) 利用 A 或寄存器进行参数传递;

(2) 利用存储器指针进行参数传递;

(3) 利用堆栈进行参数传递。

【例 3-44】 设 Xi 为单字节数,并且按照 i 顺序存放在 R0 的内容指向的地址开始的单元中,N 个数求和,N 的数量放在 R2 中,求和放在 R4(高位)和 R3(低位)中。

子程序入口:(R0)存放数据存放的首地址

R2 存放求和数据的长度

子程序出口:R4 存放和的高位,R3 存放和的低位

```
        ORG   00H
        LJMP  MAIN
        ORG   40H
MAIN:   MOV   SP,  #70H
        MOV   R0,  #30H              ;求和的数据放在 RAM30H 开始的地址
        MOV   R2,  #10               ;10 个数求和
        LCALL NSUM                   ;调用子程序
        SJMP  $
NSUM:   PUSH  ACC                    ;子程序入口压栈
        PUSH  PSW
        MOV   R3,  #0                ;求和之前清零
        MOV   R4,  #0
NEXT:   MOV   A,  @R0                ;取一个数
        ADD   A,   R3                ;加到和的低位
        MOV   R3,  A                 ;存低位
        CLR   A                      ;A 清零加进位标志位
        ADDC  A,  R4                 ;加高位
        MOV   R4,  A                 ;存高位
        INC   R0                     ;取下一个数做准备
        DJNZ  R2, NEXT               ;数据加完了吗
```

```
        POP   PSW                    ;出栈
        POP   ACC
        RET                          ;子程序返回
        END
```

例 3-44 中,数据长度 R2 是利用寄存器参数传递的,返回值也是用寄存器 R4 和 R3 进行参数传递的,数据存放位置,30H 的地址是利用存储器指针进行参数传递的。

【例 3-45】 编写程序,把片内 RAM 中 20H 单元的 1 字节的十六进制数转换成 ASCII 码,存放在 R0 指向的 2 个单元中。

子程序入口:转换数据放在栈顶;子程序出口:转换结果存在堆栈中。

```
MAIN:     MOV   A,  20H
          SWAP  A,                   ;先查高位
          PUSH  ACC
          ACALL HEXTOASC
          POP   ACC
          MOV   @R0,  A              ;存转换结果的高位
          INC   R0                   ;修改存储地址
          PUSH  20H                  ;低位字节去转换
          ACALL HEXTOASC
          POP   ACC
          MOV   @R0,  A
          SJMP  $
HEXTOASC: MOV   R1, SP
          DEC   R1                   ;跳过返回地址
          DEC   R1
          MOV   A,  @R1              ;取转换数据
          ANL   A,  #0FH             ;保留低位
          ADD   A,  #2               ;表格与查表数据的距离是 2
          MOVC  A,  @A+PC
          XCH   A,  @R1              ;转换结果存在堆栈
          RET
ASCTAB:   DB    30H,31H,32H,33H,34H,35H,36H,37H,38H,39H
          DB    41H,42H,43H,44H,45H,46H
          END
```

3.7 汇编语言程序设计实例

【例 3-46】 用查表法编写程序求平方和的程序 $y=a^2+b^2$,a 和 b 存放在 RAM30H 和 31H 中,和存放在 32H 中。

例题中的求平方采用查表法,采用数据指针指向平方表的首地址,优点是平方表可以放置在程序存储器的任何位置,不受 PC 的限制。

```
Start: MOV   A,   30H
       ACALL SQR
       MOV   R1,  A
       MOV   A,   31H
       ACALL SQR
```

```
        ADD   A,    R1
        MOV   32H,  A
        SJMP  $
SQR:    MOV   DPTR, #TAB
        MOVC  A,    @A+DPTR
        RET
TAB:    DB    0,1,4,9,16,25,36,49,64,81
```

【例 3-47】 编制用软件方法延时 1s 的子程序。

软件延时时间与执行指令的时间有关。如果使用 6MHz 晶振，一个机器周期为 $T_{cy}=2\mu s$。计算出执行每一条指令以及一个循环所需要的时间，根据要求的延时时间确定循环次数，如果单循环时间不够长，可以采用多重循环。程序如下：

```
DEL1S: MOV   R5, #05H              ;1Tcy
DELY0: MOV   R6, #200              ;1Tcy
DELY1: MOV   R7, #248              ;1Tcy
       NOP                         ;1Tcy
DELY2: DJNZ  R7, DELY2             ;2Tcy
       DJNZ  R6, DELY1             ;2Tcy
       DJNZ  R5, DELY0             ;2Tcy
       RET                         ;2Tcy
```

这是一个三重循环程序。前 4 条指令的机器周期数为 1，后 3 条指令的机器周期数为 2。执行内循环所用的机器周期数为 $248\times2=496$，执行中间循环所用的机器周期数为 $(496+4)\times200=100000$；执行外循环所用的机器周期数为 $(100000+3)\times5=500015$，再加上 1(执行第一条指令)就是执行整段程序所用的机器周期数。因此这段程序的延时时间位 $(500015+3)\times2\mu s=1.000036s$。

$$\{1+[1+(1+1+248\times2+2)\times200+2]\times5+2\}\times2\mu s=1.000036s$$

【例 3-48】 片内 RAM 中存放一批数据，查出最大值，并存放于首地址中，设 R0 中存放首地址，R2 中存放字节数。R0 为 30H，R2 为 10H。

```
Start: MOV   R2,    #10H
       MOV   R0,    #30H
       MOV   A,     R0
       MOV   R1,    A
       DEC   R2
       MOV   A,     @R1
LOOP:  MOV   R3,    A
       INC   R1
       CLR   C
       SUBB  A,     @R1
       JNC   LOOP1
       MOV   A,     @R1
       SJMP  LOOP2
LOOP1: MOV   A,     R3
LOOP2: DJNZ  R2,    LOOP
       MOV   @R0,   A
       END
```

【例 3-49】 编写一程序，实现图 3.15 中的逻辑运算电路。其中 P3.1、P1.1、P1.0 分别是单片机端口线上的信息，RS0、RS1 是 PSW 寄存器中的两个标志位，30H、31H 是两个位地址，运算结果由 P1.0 输出。

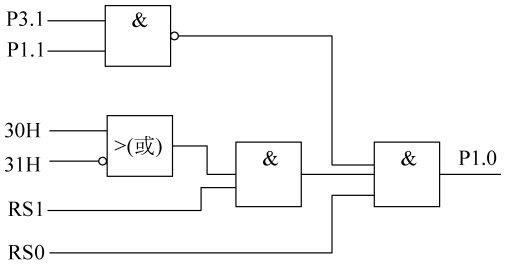


图 3.15　例 3-48 循环结构程序的嵌套形式

程序如下：

```
ORG     0000H
MOV     C,P3.1
ANL     C,P1.1
CPL     C
MOV     20H,C                    ;暂存数据
MOV     C,30H
ORL     C,/31H
ANL     C,RS1
ANL     C,20H
ANL     C,RS0
MOV     P1.0,C                   ;输出结果
SJMP    $
```

本 章 小 结

（1）51 系列单片机的寻址方式有立即数寻址、直接寻址、寄存器寻址、寄存器间接寻址、变址寻址、相对寻址和位寻址七种方式。

（2）本章分类介绍了 51 系列单片机的指令系统，汇编指令是直接针对硬件的指令，指令效率高，因此要掌握单片机的汇编指令集。

（3）本章介绍了伪指令，学会伪指令的使用。

（4）掌握汇编语言程序设计的基本方法。通过采用顺序结构、分支程序、循环程序和子程序的设计方法，提高程序的效率和可读性。子程序设计的过程中注意参数的传递。

本 章 习 题

1. 下列指令中错误的是(　　)。

（A）MOV　A，R4　　（B）MOV　20H，R4

（C）MOV　R4，R3　　（D）MOV　@R4，R3

2. 下列指令中不影响标志位 CY 的指令有(　　)。

(A) ADD A, 20H　(B) CLR　(C) RRC A　(D) INC A

3. LJMP 跳转空间最大可达到(　　)。

(A) 2KB　(B) 256B　(C) 128B　(D) 64KB

4. 设累加器 A 的内容为 0C9H,寄存器 R2 的内容为 54H,CY=1,执行指令 SUBB R2 后结果为(　　)。

(A) (A)=74H　(B) (R2)=74H　(C) (A)=75H　(D) (R2)=75H

5. 设(A)=0C3H,(R0)=0AAH,执行指令 ANL A,R0 后,结果(　　)。

(A) (A)=82H　(B) (A)=6CH　(C) (R0)=82　(D) (R0)=6CH

6. 执行如下三条指令后,30H 单元的内容是(　　)。

```
MOV  R1, #30H
MOV  40H,#0EH
MOV  @R1,  40H
```

(A) 40H　(B) 30H　(C) 0EH　(D) FFH

7. 有如下程序段:

```
MOV  R0,#30H
SETB C
CLR  A
ADDC A,#00H
MOV  @R0,A
```

执行结果是(　　)。

(A) (30H)=00H　(B) (30H)=01H　(C) (00H)=00H　(D) (00H)=01H

8. 从地址 2132H 开始有一条绝对转移指令 AJMP addr11,指令可能实现的转移范围是(　　)。

(A) 2000H～27FFH　(B) 2132H～2832H

(C) 2100H～28FFH　(D) 2000H～3FFFH

9. 对于 JBC bit,rel 指令,下列说法正确的是(　　)。

(A) bit 位状态为 1 时转移　(B) bit 位状态为 0 时转移

(C) bit 位状态为 1 时不转移　(D) bit 位状态为 0 时不转移

(E) 转移时,同时对该位清零

10. 关于指针 DPTR,下列说法正确的是(　　)。

(A) DPTR 是 CPU 和外部存储器进行数据传送的唯一桥梁

(B) DPTR 是一个 16 位寄存器

(C) DPTR 不可寻址

(D) DPTR 的地址 83H

(E) DPTR 是由 DPH 和 DPL 两个 8 位寄存器组成的

11. 对程序存储器的读操作,只能使用(　　)。

(A) MOV 指令　(B) PUSH 指令　(C) MOVX 指令　(D) MOVC 指令

12. LCALL 指令操作码地址是 2000H，执行完子程序返回指令后，PC=(　　)。

(A) 2000H　　(B) 2001H　　(C) 2002H　　(D) 2003H

13. 判断下列(　　)说法正确。

(A) 立即寻址方式是被操作的数据本身在指令中，而不是它的地址在指令中

(B) 指令周期是执行一条指令的时间

(C) 指令中直接给出的操作数称为直接寻址

14. 计算下面子程序执行的时间(晶振频率为 12MHz)。

```
     MOV   R3,#15            ;1 个机器周期
DL1: MOV   R4,#255           ;1 个机器周期
DL2: MOV   P1,R3             ;2 个机器周期
     DJNZ  R4, DL2           ;2 个机器周期
     DJNZ  R3, DL1           ;2 个机器周期
     RET                     ;2 个机器周期
```

15. 写一段程序，把片外 RAM 中 1000H～102FH 的内容传送到内部 RAM 的 30H～5FH 中。

16. 写一个程序，将内部 RAM 中 45H 单元的高 4 位清 0，低 4 位置 1。

17. 试编写程序，查找在内部 RAM 的 30H～50H 单元中是否有 0AAH 这一数据。若有，则将 51H 单元置为“01H”；若未找到，则将 51H 单元置为“00H”。

18. 设 X 存在 30H 单元中，根据下式：

$$Y=\begin{cases}X+2 & \text{当 } X>0\\ 100 & \text{当 } X=0\\ |X| & \text{当 } X<0\end{cases}$$

求出 Y 值，将 Y 值存入 31H 单元。