第3章

单片机汇编语言程序设计

最早应用于51单片机开发与应用的语言是汇编语言。汇编语言的执行速度快，代码短小精炼，且指令的执行周期确定，因此是一种高效的单片机语言。汇编语言也有不足之处：指令复杂、缺乏通用性、不便于程序移植。随着电子技术的发展，汇编语言的使用范围越来越窄，逐渐被C51语言所代替。汇编语言的程序是汇编指令的集合，可以用来控制单片机实现特定的任务。51单片机的汇编语言程序设计与汇编指令集和硬件结构等有很大关系。汇编语言因为操作硬件的能力强而获得了广泛应用。本章对51单片机汇编语言程序设计的基本情况进行简单介绍。

3.1 51系列单片机的汇编指令格式和功能描述符

指令是CPU根据人的意图来执行某种操作的命令。一台计算机所能执行的全部指令的集合称为这个CPU的指令系统。指令系统的功能强弱在很大程度上决定了这类计算机性能的高低。

MCS-51系列单片机指令系统具有功能强、指令短、执行快等特点，用42个助记符代表33种操作功能，共111条指令，其中，有49条单字节指令、45条双字节指令和17条三字节指令；有64条单周期指令、45条双周期指令，只有乘法、除法两条指令为四周期指令。占字节数多的指令不一定执行时间就长，反之亦然。按指令的功能属性，MCS-51系列单片机指令可分为数据传送指令、算术运算指令、逻辑操作指令、控制转移指令和位操作指令。本节将对指令格式和指令功能描述符进行介绍和说明。

3.1.1 指令格式

不同的单片机指令可以实现不同的功能，具体格式也不同。但是从总体上分析，每一条指令通常由操作码和操作数两部分组成。操作码表示计算机执行该指令将进行何种操作，

操作数表示参加操作的数或者操作数所在的地址。MCS-51 单片机汇编语言指令的基本格式为

[标号：]操作码助记符 [操作数 1][,操作数 2][,操作数 3]　{；注释}

方括号内的部分可以根据实际情况取舍，每个字段之间可以用分隔符分隔，可以用作分隔符的符号有空格(用于操作码和操作数之间)、冒号(用于标号之后)、逗号(用于操作数之间)、分号(用于注释之前)等。例如，“LOOP：MOV A，＃7FH；A←7FH”的功能是循环将立即数＃7FH 传送到目的操作数累加器 A 中。

具体说明如下：

(1) 标号是指令的符号地址，通常作为转移指令的操作数。对标号有如下规定：

① 由 1～31 个字符组成，要以非数字字符开头，后跟字母、数字、“－”和“?”等字符。

② 不能用已定义的保留字(如指令助记符、伪指令、寄存器名称和运算符等)。

③ 必须后跟英文冒号“:”。

(2) 操作码助记符表明指令的不同功能，是汇编语句中唯一不能空缺的部分，汇编器在汇编时会将其翻译成对应的二进制代码。不同的指令有不同的助记符，一般用说明其功能的英文缩写表示。例如，“MOV”表示传送，“ADD”表示加法。三操作数指令只有一条，就是比较转移指令“CJNE”，后面的章节会介绍。

(3) 操作数是指令要操作的数据或数据的地址。在一条汇编语句中，操作数可能是空缺的，也可能是多项的。MCS-51 单片机的指令按操作数的多少，可分为无操作数、单操作数、双操作数和三操作数四种指令。例如，“RET”指令是返回调用子程序的下一条指令位置，该指令无操作数；“INC A”的功能是对累加器 A 中的内容加 1，只有一个操作数；而上文提到的“LOOP：MOV A＃7FH ；A←7FH”指令，有两个操作数。操作数的内容可能包含以下几种情况：

① 数据。

二进制数，末尾以 B 标识。如，1111 1000B。

十进制数，末尾以字母 D 标识或将字母 D 省略。如，88D，88。

十六进制数，末尾以字母 H 标识。如，88H，0F8H。此处应该注意，十六进制数以字母 A～F 开头时，应在其前面加上数字 0 引导，以便于汇编器将其与标号或符号名相区分。

ASCII 码以单引号进行标识。如‘A’，‘1234’。

② 符号。可以是符号名、标号或特定的符号“＄”(该指令的存储地址)等。

③ 表达式。由运算符和数据构成的算式。可用的运算符如表 3-1 所示。

(4) 注释是对语句的解释说明，是编程人员根据需要加上去的，用于对指令进行解释和说明，可以增加程序的可读性，也有助于程序的阅读和维护。对于指令本身的功能而言，是可以不添加的。该字段必须以英文“;”开头，当一行书写不下时，可以换行接着书写，但换行时应注意使用“;”开头。

表 3-1 51 汇编器的运算符及其优先级

优先级	运算符	功能	表达式及其结果
高↓低	( )	括号	4*(4+4)即 64
	NOT、HIGH、LOW	取反、取高字节、取低字节	NOT 55H 即 AAH；HIGH 1234H 即 12H
	+、−	正号、负号	+3、−4
	*、/、MOD	乘、除(取商)、取余数	17/6 即 2；17/6 即 5
	+、−	加、减	5+4 即 9；5−4 即 1
	SHL、SHR	左移、右移	2 SHL 2 即 8；8 SHR 2 即 2
	AND、OR、XOR	与、或、异或	45H AND 0FH 即 05H
	<、>、=、<>、<=、>=	比较运算符	MOV A,X>8 若 X>8 为真,则为 MOV A,01H; 若 X>8 为假,则为 MOV A,00H

注：表达式中含有多个相同优先级运算符时,优先级按“从左到右”顺序确定。

3.1.2 指令功能描述符

MCS-51 单片机汇编指令功能描述符常用以下符号表示：

(1) Rn(n=0～7)：表示当前工作寄存器组中的寄存器 R0～R7 之一；

(2) Ri(i=0,1)：表示当前工作寄存器组中的寄存器 R0 或 R1；

(3) A：累加器,又可写成 ACC；

(4) B：专用寄存器,多用于乘法 MUL、除法 DIV 指令中；

(5) C：表示当前工作寄存器组中的寄存器 R0 或 R1；

(6) @：间接寻址或变址寻址前缀；

(7) #data：8 位立即数；

(8) #data16：16 位立即数；

(9) rel：以补码形式表示的 8 位地址偏移量,其值在−128～+127 内；

(10) addr11：11 位直接地址；

(11) addr16：16 位直接地址；

(12) direct：直接寻址的地址(片内 RAM 单元地址及 SFR 地址,可用符号名称表示)；

(13) bit：按位寻址的直接位地址(片内 RAM 位地址及 SFR 中的位地址,可用符号名称表示)；

(14) DPTR：数据指针,作 16 位地址寄存器；

(15) /：表示对该位操作数取反,但不影响该位的原值；

(16) →或←：表示数据传送方向；

(17) (×)：表示×地址单元或寄存器的内容；

(18) ((×))：表示×地址单元或寄存器内容为地址所指定单元的内容；

(19) ↔：表示数据交换；

(20) $：指令自身的首地址,用作跳转指令的自循环地址标号。

3.2　51 系列单片机指令的寻址方式

指令由操作码和操作数组成，操作数又分为源操作数和目地操作数。操作数总是存放在某一存储单元中，寻找操作数实际上是寻找操作数所在的单元地址，称为寻址方式，寻址方式取决于单片机自身的硬件结构。总体来说，寻址方式越丰富，单片机的指令功能越强，编程灵活性越大，指令系统也越复杂。寻址方式所要解决的主要问题就是如何在整个存储器和寄存器的寻址空间内，灵活方便、快速地找到指令的操作数。

MCS-51 单片机中，操作数的存放范围很宽，可以放在片外 ROM/RAM 中，也可以放在片内 ROM/RAM 以及 SFR 中。为了适应在操作数范围内的寻址，MCS-51 单片机指令系统使用了 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、相对寻址和位寻址。

3.2.1　立即寻址

操作数是 1 字节或 2 字节的常数，直接在指令中给出，用“＃”符号作为前缀，以区别直接寻址方式。在指令编码中，该操作数紧跟在操作码后面，与操作码一起存放在指令代码段(ROM)中，可以立即得到并执行，不需要经过别的途径去寻找，被称为立即数，该寻址方式被称为立即寻址。例如，

```
MOV  A,＃18H
```

该指令的功能是把立即数 18H 送到累加器 A，其中操作数 18H 为源操作数，是立即数。指令执行后累加器 A 中的内容为 18H。该条指令的操作码为 74H，存储和执行过程如图 3-1 所示。

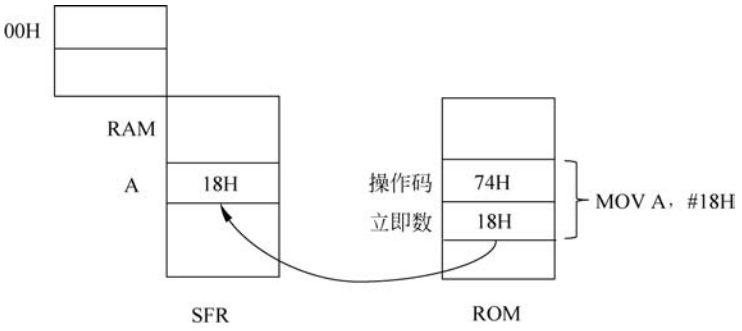

图 3-1　立即寻址方式示意图

在 MCS-51 单片机指令系统中，仅有一条包含 16 位立即数的指令，形式为“MOV DPTR,＃data16”，其中“＃data16”表示 16 位立即数。例如：指令“MOV　DPTR,＃1234H”，其功能是把 16 位立即数“1234H”传送到寄存器 DPTR 中，其中高 8 位“12H”送到 DPH，低 8 位“34H”送到 DPL。

因为立即数直接存放在 ROM 中，所以立即寻址对应的寻址空间为 ROM 空间。

3.2.2 直接寻址

指令中直接给出操作数所在存储单元的地址。在指令编码中，操作数所在存储单元的地址紧跟在操作码之后，与操作码一起存放在指令代码段中，而操作数本身则存放在该地址所指示的存储单元中。例如，

```
MOV  A,18H
```

该指令的功能是把内部 RAM 中 18H 地址的内容传送到累加器 A 中。如果指令执行前片内数据存储器 18H 单元的内容为 56H，则执行后累加器 A 的内容为 56H。该条指令的操作码为 E5H，存储和执行过程如图 3-2 所示。

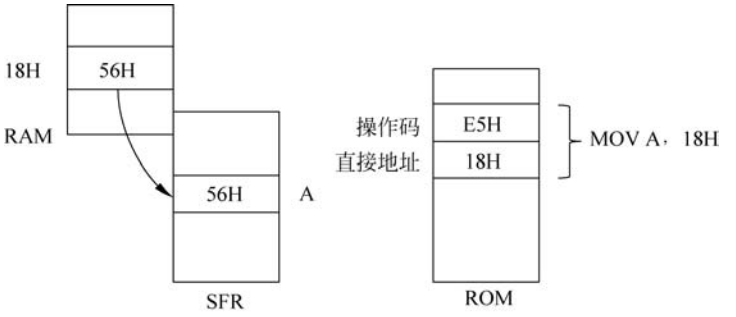

图 3-2 直接寻址方式示意图

直接寻址可访问内部 RAM 低 128 个单元及 SFR 区域。需要注意的是，片内 RAM 高 128 个单元(增强型)必须采用寄存器间接寻址方式访问。对于 SFR，在指令中通常采用寄存器符号来表示操作数所在单元的地址。例如，“MOV A，80H”可以直接写成“MOV A，P0”。这里的“P0”和“80H”是等效的。

3.2.3 寄存器寻址

操作数存放在寄存器中，指令中直接给出该寄存器的名称。存放操作数的寄存器在指令代码中不占据单独的字节，而是包含在操作码字节中，因此该寻址方式的指令为一字节指令。由于寄存器在单片机 CPU 内部，因此采用寄存器寻址的速度相比于其他几种寻址方式要快，可以使程序具有较好的运算处理速度。

在 MCS-51 系列单片机中，寄存器寻址方式针对的寄存器只能是 R0～R7 这 8 个通用工作寄存器和部分特殊功能寄存器(如累加器 A、寄存器 B、数据指针寄存器 DPTR 和位累加器 CY)。在汇编指令中，寄存器寻址在指令中直接提供寄存器的名称，如 R0、R1、A 等。例如，

```
MOV  A,R0
```

该指令的功能是把 R0 中的内容传送到累加器 A 中。如果指令执行前 R0 中的内容为 56H，则指令执行后累加器 A 中的内容为 56H。该条指令的操作码为 E8H，存储和执行过程如图 3-3 所示。

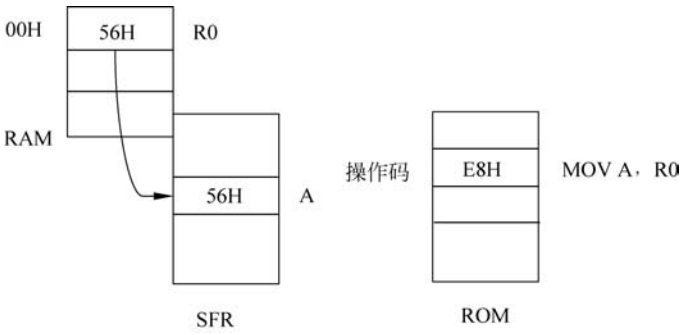

图 3-3　寄存器寻址方式示意图

另外，对于第 0 组工作寄存器，R0 的物理地址为 00H，因此指令“MOV A，00H”和“MOV A，R0”实现的功能是一样的，都是将 00H 单元的内容传送到累加器 A 中。但是两者也是有区别的，前者属于直接寻址，机器码为 E5H、00H，这条指令占用 2 字节；后者属于寄存器寻址，机器码为 E8H，这条指令占用 1 字节。由此可见，同一个功能可以采用不同的指令来实现，而且对于同一个存储单元，若采用不同的形式表示，所对应的寻址方式是不同的。

3.2.4　寄存器间接寻址

指令中给出的寄存器中存放的不是操作数，而是操作数所在单元的地址，类似于 C 语言的指针，即操作数是通过指令中给出的寄存器间接得到的。在 MCS-51 系列单片机中，只能使用寄存器 R0、R1、SP 和 DPTR 作为间接寻址的寄存器。为了区别于寄存器寻址方式，在寄存器的名称前加前缀标志“@”来表示寄存器间接寻址。例如，

```
MOV  A,@R0
```

该指令的功能是把以 R0 中的内容作为地址的片内 RAM 单元的数据传送到累加器 A 中。如果指令执行前 R0 中的内容为 80H，片内 RAM80H 地址单元的内容为 56H，则指令执行后累加器 A 中的内容为 56H。该条指令的操作码为 E6H，存储和执行过程如图 3-4 所示。

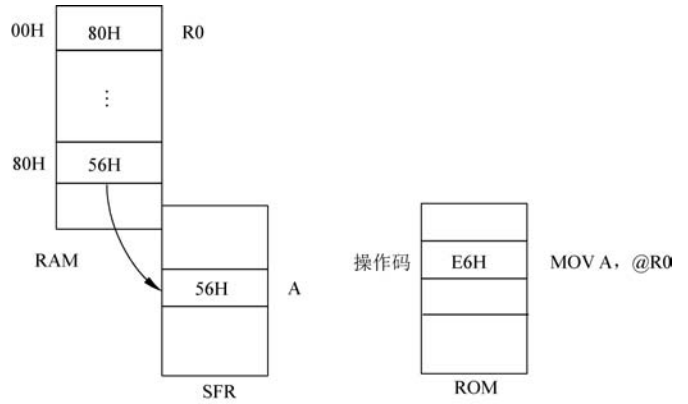

图 3-4　寄存器间接寻址方式示意图

寄存器间接寻址方式的寻址空间为RAM，并且，

(1) 内部RAM低128单元，应使用R0或者R1作为间接寻址寄存器，其通用形式为“@R0”或“@R1”。

(2) 对外部RAM单元的间接寻址，一般采用两种方式：采用Ri(i=0或1)作为间接寻址寄存器，可以寻址256个单元；或者采用16位数据指针DPTR作为间接寻址寄存器，可以寻址外部RAM完整64KB地址空间。

(3) 堆栈操作指令PUSH和POP使用堆栈指针SP作为间接寻址寄存器对堆栈区进行间接寻址。

3.2.5 变址寻址

以DPTR或PC作为基址寄存器，以累加器A作为变址寄存器，并以两者内容相加形成操作数所在单元的地址。在MCS-51系列单片机中，用变址寻址方式只能访问ROM，寻址范围为0000H～FFFFH，最大存储容量为64KB。该寻址方式通常用于访问ROM中的表格型数据，表首单元的地址为基址，放于基址寄存器，访问的单元相对于表首的位移量为变址，放于变址寄存器，通过变址寻址可得到ROM相应单元的数据。例如，

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

该指令的功能是把DPTR和累加器A的内容相加得到的ROM中某单元的地址对应单元中的内容传送到累加器A中。如果指令执行前，数据指针寄存器DPTR的值为2000H，累加器A中的值为05H，ROM 2005H单元的内容为56H，则指令执行后累加器A中的内容为56H。该条指令的操作码为93H，存储和执行过程如图3-5所示。

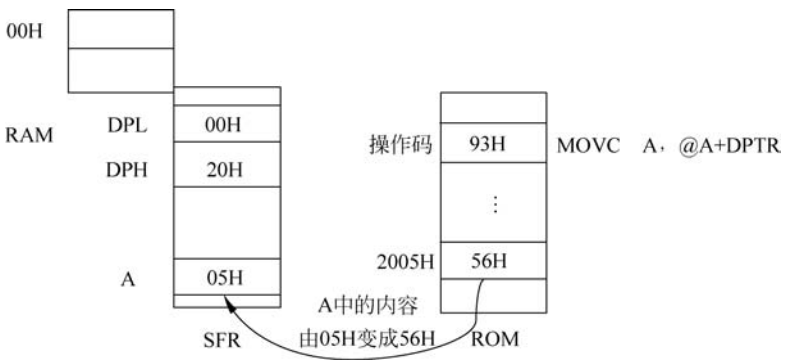

图3-5 变址寻址方式示意图

需要说明的是，变址寻址的指令只有3条：

(1) MOVC A,@A+DPTR

(2) MOVC A,@A+PC

(3) JMP @A+PC

其中，前两条指令是ROM读指令，最后一条是无条件跳转指令。

3.2.6 相对寻址

程序计数器PC的当前值加上指令中给出的偏移量rel，所得结果作为转移地址送入PC

中，即跳向一个新的地址来执行程序。采用该寻址方式进行操作时修改的是 PC 值，因此这种寻址方式用于实现程序的分支跳转。

使用相对寻址时需要注意以下两点，

(1) PC 的当前值为读出 2 字节或者 3 字节的跳转指令后，PC 指向的下一条指令的地址，即 PC 当前地址＝相对转移指令所在存储单元的地址＋指令字节数。

例如，“JZ rel”是一条累加器 A 为 0 就转移的双字节指令。若该指令的存储地址为 2020H，则执行该指令时的 PC 当前值为 2022H。

(2) 偏移量 rel 是一个有符号的 8 位二进制补码数，以补码的形式置于操作码之后存放，范围是－128～＋127。负数表示向地址减小的方向转移，正数表示向地址增加的方向转移，目标地址＝当前 PC 值＋rel＝指令存储地址＋指令字节数＋rel。

例如，rel 为 75H，PSW.7 为 1，指令“JC　rel”存放在 ROM 地址 1000H 处开始的单元，即 PC 中的值为 1000H，则执行“JC　rel”(指令的机器码为 4075H)指令后，程序跳转到 1077H 单元取指令并执行，执行过程如图 3-6 所示。目标地址＝当前 PC 值(1000H＋02H)＋偏移量(75H)＝1077H。

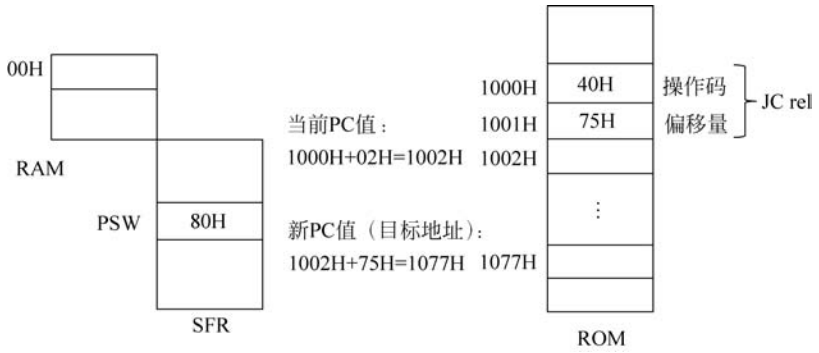

图 3-6　相对寻址方式示意图

实际编写程序时，程序中只需要在转移指令中给出地址标号，汇编过程中编译器会自动计算偏移量 rel，如果 rel 值超越规定的有效范围，会自动给出错误提示。此情况下需要利用跳转范围较大的绝对转移指令(如 AJMP)转移到所需的地址标号。例如，

```
SJPM  LOOP        ; LOOP 为要转向的目标地址标号
```

3.2.7　位寻址

MCS-51 系列单片机中，有一个独立的位处理器，能够进行各种位运算，位运算的操作对象为片内 RAM 的位寻址区和某些可位寻址的 SFR 内的位数据。指令中给出位数据的位地址的寻址方式称为位寻址。位寻址方式属于位的直接寻址，与直接寻址方式不同的是，位寻址只给出位地址，而不是字节地址。需要注意的是，位地址和字节地址的表示形式完全一样，位地址和字节地址是根据指令功能来区分的。如果是位操作指令，则所操作的是位地址，属于字节地址中的某一位；如果是字节操作指令，则所操作的是字节地址，属于整个字节。例如，

```
MOV  C,00H  ; 对位累加器 C 操作,00H 是位地址(属于字节地址 20H 的最低位)
MOV  A,00H  ; 对累加器 A 操作,00H 是字节地址(属于 0 区工作寄存器的 R0)
```

MCS-51 系列单片机内部 RAM 中有两个区域可以位寻址：

(1) 片内 RAM 的 20H～2FH 单元是可以进行位寻址的区域,共 16×8＝128 位,它们的位地址为 00H～7FH。例如,20H 单元的 0～7 位的位地址为 00H～07H。该区域的可寻址位有两种表示方法：

① 直接使用位地址表示；②使用单元地址加位序号表示。

例如,位地址 00H 和 20H.0 指的都是片内 RAM 中 20H 单元的第 0 位。编程中常用“位地址”表示方法。

(2) SFR 的可寻址位,可提供位寻址的特殊功能寄存器有 11 个,字节地址能被 8 整除的特殊功能寄存器可以位寻址。该区域的可寻址位有 4 种表示方法。① 使用位名称表示；②直接使用位地址表示；③使用单元地址加位序号表示；④使用 SFR 符号加位序号表示。

例如,终端允许寄存器 IE 可寻址位如下：

	D7	D6	D5	D4	D3	D2	D1	D0
(A8H)	AFH	AEH	ADH	ACH	ABH	AAH	A9H	A8H
IE	EA			ES	ET1	EX1	ET0	EX0

其中,最高位是中断允许位,位名称是 EA,直接位地址是 0AFH,单元地址加位序号是 0A8H.7,SFR 符号加位序号是 IE.7。编程中常用“位名称”表示方法。

本节介绍的七种寻址方式及它们所对应的寄存器和寻址空间如表 3-2 所示。

表 3-2　七种寻址方式所对应的寄存器和寻址储空间

寻 址 方 式	使用的变量	寄存器或寻址空间
立即寻址	直接给出数字,无变量	ROM
直接寻址	直接给出变量,无地址	片内 RAM 低 128 字节；SFR
寄存器寻址	R0～R7、A、B、DPTR、位累加器 C	工作寄存器 R0～R7；A、AB、DPTR 和 C；部分 SFR
寄存器间接寻址	@R0、@R1、SP 或@DPTR	片内 RAM 或片外 RAM
变址寻址	@A＋DPTR、@A＋PC	ROM
相对寻址	PC＋偏移变量	ROM
位寻址	直接给出位地址或位符号	片内 RAM 的位寻址区,SFR 的可寻址位

3.3　51 系列单片机的指令系统

51 系列单片机的指令系统共有 111 条指令,按功能不同可以分为 5 大类：

(1) 数据传送指令(29 条)：实现存储器赋值、数据转移等功能；

(2) 算术运算指令(24 条)：实现数值的加、减、乘、除等运算功能；

(3) 逻辑运算指令(24 条)：实现逻辑与、或、异或、移位等功能；

(4) 控制转移指令(17 条)：实现程序条件转移、无条件转移等功能；

(5) 0 位操作指令(17 条)：实现位清 0、置 1、判断等功能，由 51 单片机内部特有的布尔处理器完成。

3.3.1　数据传送指令

51 系列单片机中，数据传送是最基本也是最主要的操作。该操作可以在片内 RAM 单元和 SFR 中进行，也可以在累加器 A 和片外 RAM 之间进行，还可以到 ROM 中进行查表，并且除了以累加器 A 为目的操作数的指令会对 PSW 中的奇偶标志位 P 有影响外，其余指令执行时均不会影响任何标志位。因此，数据传送指令是指令系统中数量最多、使用也最频繁的一类指令。数据传送指令共 29 条，采用 7 个助记符 MOV、MOVX、XCH、XCHD、SWAP、PUSH 和 POP 表示。

数据传送指令可分为三组：普通数据传送指令、数据交换指令和堆栈操作指令。

1. 普通数据传送指令

普通数据传送指令的功能是将源操作数的内容复制到目的操作数所在单元，而源操作数的内容不变。该指令以助记符 MOV 为基础，根据访问对象的不同，分为片内数据存储器传送指令、片外数据存储器传送指令和程序存储器传送指令。

1) 片内数据存储器传送指令 MOV

指令格式如下：

```
MOV 目的操作数,源操作数
```

其中，源操作数可以是 A、Rn、@Ri、direct、#data[16]，目的操作数可以为 A、Rn、@Ri、direct、DPTR。将目的操作数和源操作数按照不同的寻址方式进行组合，组合起来总共 16 条，如图 3-7 所示。

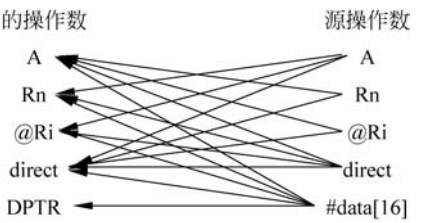

图 3-7　操作数组合

基于此，片内数据存储器传送指令按目的操作数的寻址方式，可划分为 5 组。

(1) 以 A 为目的操作数(4 条)：

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

上述指令的功能是将源操作数所指定的工作寄存器 Rn(即 R0～R7)的内容、直接寻址或寄存器 Ri(即 R0 或 R1)寻址所得的片内 RAM 单元或特殊功能寄存器中的内容以及立即数传送到累加器 A 中。

上述操作不影响源字节和任何别的寄存器内容，只影响 PSW 的 P 标志位。

例如，

```
MOV    A,#55H            ; A←55H
MOV    A,55H             ; A←(55H)
MOV    A,R0              ; A←(R0)
MOV    A,@R0             ; A←((R0))
```

(2) 以 Rn 为目的操作数(3 条)：

```
MOV    Rn,A              ; Rn←A
```

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

上述指令的功能是将源操作数所指定的内容传送到当前工作寄存器组 R0～R7 中的某个寄存器。源操作数有寄存器寻址、直接寻址和立即数寻址 3 种方式。

注意,51 指令系统中没有“MOV　Rn,Rn”传送指令。

例如,

```
MOV    R7,A             ; R7←A
MOV    R7,55H           ; Rn←(55H)
MOV    R7,#55H          ; Rn←#55H
```

(3) 以直接地址 direct 为目的操作数(5 条):

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

上述指令的功能是将源操作数指定的内容送到由直接地址 direct 所指定的片内存储单元中。源操作数的寻址方式分别为寄存器寻址、直接寻址、寄存器间接寻址和立即寻址。

注意,“MOV　direct,direct”指令中,源地址在前,目的地址在后。

例如,

```
MOV    30H,A            ; (30H)←A
MOV    30H,R0           ; (30H)←R0
MOV    30H,55H          ; (30H)←(55H)
MOV    30H,@R0          ; (30H)←(R0)
MOV    30H,#55H         ; (30H)←#55H
```

(4) 以@Ri 为目的操作数(3 条):

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

上述指令的功能是将源操作数所指定的内容传送到 Ri(R0 或 R1)内容所指向的地址单元中,源操作数的寻址分别为寄存器寻址、直接寻址和立即寻址。

例如,

```
MOV    @R0,A            ; (Ri)←A
MOV    @R0,55H          ; (Ri)←(55H)
MOV    @R0,#55H         ; (Ri)←#55H
```

(5) 以 DPTR 为目的操作数(1 条):

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

该条指令是 51 单片机指令系统中唯一的一条 16 位数据传送指令,该指令的功能是将 16 位立即数传送到数据指针 DPTR 中。由于 DPTR 是由 DPH 和 DPL 组成的,因此这条

指令执行后要把 data16 的高 8 位数据传送给 DPH，而把低 8 位数据传送给 DPL。例如，

```
MOV  DPTR,＃2345H
```

该指令执行后，DPH 中的值为 23H，DPL 中的值为 45H。

为了实现此功能，我们也可以分别向 DPH 和 DPL 传送数据，即用下面的两条指令实现上述功能。

```
MOV  DPH,＃23H
MOV  DPL,＃45H
```

此时，DPH 和 DPL 是特殊功能寄存器，属于 direct 类型。

综合以上，可以得出 51 单片机 MOV 传送指令中各操作数之间的关系，如图 3-8 所示。

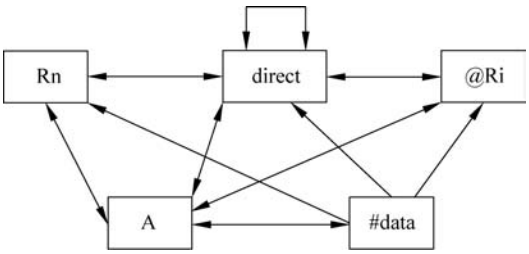

图 3-8　MOV 指令中各操作数之间的关系

需要说明的是，片内数据存储器传送指令 MOV 在使用时应注意，源操作数和目的操作数中的 Rn 和@Ri 不能相互配对，不允许“MOV　Rn,Rn”“MOV　@Ri,Rn”这样的指令。在 MOV 指令中，不允许在一条指令中同时出现工作寄存器，无论它是寄存器寻址还是寄存器间接寻址。

【例 3-1】　设(70H)＝60H，(60H)＝20H，P1 为输入口，状态为 0B7H，执行如下程序：

```
MOV    R0,＃70H
MOV    A,@R0
MOV    R1,A
MOV    B,@R1
MOV    @R0,P1
```

【解】　程序执行后，(R0)＝70H，(A)＝60H，(R1)＝60H，(B)＝20H，(70H)＝0B7H。

【例 3-2】　把存放在片内 RAM 30H 单元中的数据 00H 送到 60H 单元中(要求最终指令以@Ri 为目的操作数)。

【解】　程序 1：

```
MOV    R1,＃60H          ; (R1) = 60H
MOV    @R1,30H           ; ((R1)) = (60H) = (30H) = 00H
```

程序 2：

```
MOV    R1,＃60H          ; (R1) = 60H
MOV    A,30H             ; (A) = (30H) = 00H
MOV    @R1,A             ; ((R1)) = (60H) = (A) = (30H) = 00H
```

2）片外数据存储器传送指令 MOVX

51 指令系统中，只能通过累加器 A 与片外数据存储器进行数据传送，且采用@Ri 和@DPTH 寄存器间接寻址的方式完成。该类指令的助记符为 MOVX，共有 4 条，分别为

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

其中，前两条指令通过@DPTR 间接寻址，可以对整个 64KB 片外数据存储器访问。高 8 位地址放在 DPH 中，由 P2 口输出，低 8 位地址放在 DPL 中，由 P0 口输出。后两条指令通过@Ri 间接寻址，只能对片外数据存储器的低 256 字节访问，高 8 位地址放在 Ri(R1 或 R0)中，由 P0 口输出，此时如果访问超过 256 字节的外部 RAM 空间，需利用 P2 口确定高 8 位地址(也称页地址)。

值得说明的是，片外扩展的 I/O 接口也要利用这 4 条指令进行数据输入和输出。

【例 3-3】 试写出完成以下功能的程序：

① 将片内 RAM 60H 单元中的内容送入片外 RAM 40H 单元中。

② 将片外 RAM 2000H 单元中的内容送到片内 RAM 20H 单元中。

③ 将片外 RAM 2010H 单元中的内容送到片外 RAM 2020H 单元中。

【解】 程序如下：

```
①  MOV      A,60H
   MOV      R0,#40H
   MOVX     @R0,A
②  MOV      DPTR,#2000H
   MOVX     A,@DPTR
   MOV      20H,A
③  MOV      P2,#20H
   MOV      R0,#10H
   MOVX     A,@R0
   MOV      R1,#20H
   MOVX     @R1,A
```

3）程序存储器传送指令 MOVC

通常 ROM 中可以存放两类内容：一是单片机执行的程序代码；二是一些固定不变的常数(如表格数据、字符代码等)。访问 ROM 实际就是读取 ROM 常数表中的数据，简称查表，而访问 ROM 的数据传送指令被称为查表指令。该指令必须通过累加器 A 采用变址寻址的方法来完成，共有两条指令：

```
MOVC   A,@A+DPTR        ; A←(A+DPTR)
MOVC   A,@A+PC          ; A←(A+PC)
```

此两条指令主要用于对存放在 ROM 中常数表的查找。由于偏移量在 A 中，故常数表长度不超过 256B。利用上面第 1 条指令查表时，表头地址由 DPTR 指定，所以常数表易于放置在 64KB 的任意位置，称为远程查表指令；利用第 2 条指令查表时，常数表放在该指令后的 256B 空间查表，称为近程查表指令。

对于近程查表指令应该注意，PC 的内容是指取出该条指令字节后的 PC 值，与远程查表指令相比，该指令无须 DPTR 参与（节省资源），也没有影响 PC 的内容。

2. 数据交换指令

普通数据传送指令实现将源操作数的数据传送到目的操作数，指令执行后，源操作数不变，数据传送是单向的。数据交换指令是把数据双向传送，传送后，前一个操作数原来的内容传送到后一个操作数中，后一个操作数原来的内容传送到前一个操作数中。数据交换指令又分为字节交换指令和半字节交换指令两种。

（1）字节交换指令包含以下 3 条：

```
XCH   A,Rn                ;A<=>Rn
XCH   A,direct            ;A<=>(direct)
XCH   A,@Ri               ;A<=>(Ri)
```

这 3 条指令的功能是将累加器 A 中的数据与源操作数中的数据进行交换。

（2）半字节交换指令包含以下 2 条：

```
XCHD  A,@Ri               ;A0~3<=>(Ri)0~3
SWAP  A                   ;A0~3<=>A4~7
```

第 1 条指令的功能是将累加器 A 的低 4 位（低半字节）与间接寄存器所指向的地址单元中的低 4 位内容互换，而各自的高 4 位（高半字节）内容保持不变。

第 2 条指令的功能是将累加器 A 内部的高 4 位与低 4 位的内容互换。

值得说明的是，数据交换指令要求第一个操作数必须为累加器 A。

【例 3-4】　已知 R0＝30H，(30H)＝4AH，A＝28H，试分析下列指令执行的结果：

```
① XCH  A,@R0
② XCHD A,@R0
③ SWAP A
```

【解】　①执行后 A＝4AH，(30H)＝28H；②执行后 A＝2AH，(30H)＝48H；③执行后 A＝82H。

3. 堆栈操作指令

堆栈是片内 RAM 中按“先进后出，后进先出”原则设置的专用存储区。此区域一端固定，称为栈底；另一端是活动的，称为栈顶，栈顶的位置（地址）由指针 SP 指示（即 SP 的内容是栈顶的地址）。在 MCS-51 单片机中，堆栈设置在片内 RAM 的低 128 字节单元，且生长方向是向上（地址增大）的。系统复位时，SP 的内容为 07H。通常，用户应在系统初始化时对 SP 重新设置。SP 的值越小，堆栈的深度越深。

堆栈主要用于子程序调用时保护返回地址，或者用于保护子程序调用之前的某些重要数据（即保护现场）。

堆栈操作指令有 2 条：

```
PUSH  direct              ;SP←SP+1,SP←(direct)
POP   direct              ;(direct)←SP,SP←SP-1
```

其中，PUSH 为入栈指令，POP 为出栈指令。操作时以字节为单位。入栈时，SP 指针先加

1，再入栈；出栈时，内容先出栈，SP 指针再减 1。通常情况下，入栈指令和出栈指令是成对出现的。用堆栈保护数据时，先入栈的内容后出栈；后入栈的内容先出栈。例如，

若入栈保存时，入栈的顺序为

```
PUSH  A
PUSH  B
```

则出栈的顺序为

```
POP   B
POP   A
```

【例 3-5】 用堆栈指令实现 RAM 10H 和 20H 中的内容交换，设(10H)＝12H，(20H)＝34H。

【解】 程序如下：

```
MOV   SP,#6FH          ;设置堆栈指针指向 6FH
PUSH  10H              ;SP←SP + 1,SP = 70H,(71H) = 12H
PUSH  20H              ;SP←SP + 1,SP = 71H,(71H) = 34H
POP   10H              ;(10H) = 34H,SP←SP - 1,SP = 70H
POP   20H              ;(20H) = 12H,SP←SP - 1,SP = 6FH
```

3.3.2 算术运算指令

算术运算指令可以完成加、减、乘、除、加 1、减 1 和十进制调整，共 24 条。这些指令的操作数都是 8 位无符号数，不能直接对有符号数和 16 位数据进行运算。算术运算指令的执行一般会影响程序状态字 PSW 中的一些标志位。

1. *加法指令*

加法指令有不带进位的加法指令、带进位的加法指令和加 1 指令。

(1) 不带进位的加法指令 ADD

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

这 4 条 8 位二进制数加法指令的一个加数总是来自累加器 A，而另一个加数可由寄存器寻址、直接寻址、寄存器间接寻址和立即数寻址等不同的寻址方式得到，相加结果总是放在累加器 A 中。

使用指令时要注意累加器 A 中的运算结果对各个标志位的影响。

① 如果位 7 有进位，则进位标志 CY 置 1，否则 CY 清 0；

② 如果位 3 有进位，则辅助进位标志 AC 置 1，否则 AC 清 0；

③ 如果位 6 有进位，而位 7 没有进位，或者位 7 有进位，而位 6 没有进位，则溢出标志位 OV 置 1，否则 OV 清 0；

④ 累加器 A 中有奇数个“1”时，P＝1，反之 P＝0。

值得说明的是，溢出标志位 OV 的状态，只有在进行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，运算结果是错误的，否则运算是正确的，即无溢出产生。

在实际应用中，编程者应该确保单字节无符号数运算结果不超过 255。如果结果可能大于 255，就应将数据用多字节形式表示；单字节的有符号数运算结果不要超过－128～127，如果结果可能超过这个范围，就应该将数据用多字节表示或在程序运算中对 PSW 寄存器中的状态标志进行判断，并根据判断情况进行相应处理。

【例 3-6】　分析如下程序段执行后，累加器 A 及 PSW 相关标志的结果。

```
MOV   A,＃10010010B
ADD   A,＃11001001B
```

【解】　指令执行后，(A)＝01011011B＝5BH，CY＝1，AC＝0，OV＝1，P＝1。

【例 3-7】　(A)＝53H，(R0)＝FCH，执行指令

```
ADD   A,R0
```

【解】　运算结果为(A)＝4FH，CY＝1，AC＝0，OV＝0，P＝1(位 6 和位 7 同时进位，所以 OV＝0)。

(2) 带进位的加法指令 ADDC

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

这组加法指令的特点是进位标志位 CY 参加运算，指令中不同寻址方式所指定的加数、进位标志与累加器 A 中内容相加，结果存在累加器 A 中，并根据指令执行情况，重新对 CY、AC、OV、P 等标志位置 1 或清 0。

带进位的加法指令常用于多字节数加法运算中。

【例 3-8】　已知当前的 CY＝1，分析下列指令的执行结果。

```
MOV   A,＃85H
ADDC  A,＃97H
```

【解】　指令执行结果为(A)＝1DH，CY＝1，AC＝0，OV＝1，P＝0。

【例 3-9】　试把存放在 R1R2 和 R3R4 中的两个 16 位数相加，结果存于 R5R6 中。

【解】　处理时，R2 和 R4 用一般的加法指令 ADD，结果存放于 R6 中；R1 和 R3 用带进位的加法指令 ADDC，结果存放于 R5 中，程序如下：

```
MOV   A, R2
ADD   A, R4
MOV   R6,A
MOV   A, R1
ADDC  A, R3
MOV   R5,A
```

(3) 加 1 指令 INC

```
INC   A                 ;A←A＋1，影响 P 标志
```

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

这组指令是单字节指令，其功能是把指令中所指出的变量加 1，且不影响除 P 以外其余 PSW 中的任何标志位。若变量原来为 FFH，加 1 后将溢出为 00H(仅指前 4 条指令)，标志也不会受到影响。最后一条指令是 16 位数加 1 指令，首先对低 8 位指针 DPL 的内容加 1 操作，当产生溢出时，就对 DPH 的内容进行加 1 操作，并不影响 CY 的状态。

Ri 和 DPTR 常用作指针并指向一片存储区域的起始地址，将它们的内容加 1 可以指向下一单元的地址，编程时常利用这两个指针对一片存储区域进行操作。

2. 减法指令

减法指令有带借位减法指令和减 1 指令。

(1) 带借位减法指令 SUBB

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

这组指令的功能是从累加器 A 中的内容减去指定的变量和进位标志 CY 的值，结果存放在累加器 A 中，常用于多字节减法运算中。MCS-51 单片机中，只提供了一种带借位的减法指令。不带借位的减法操作可以通过先对 CY 标志清零，然后再执行带借位的减法来实现。

使用指令时要注意累加器 A 中的运算结果对各个标志位的影响。

① 如果最高位有借位，则 CY 置 1，否则 CY 清 0；

② 如果低 4 位向高 4 位有借位时，AC＝1，否则 AC＝0；

③ 如果位 6 和位 7 不同时，产生借位时，OV＝1，否则 OV＝0；

④ 累加器 A 中有奇数个“1”时，P＝1，反之 P＝0。

(2) 减 1 指令 DEC

```
DEC   A               ;A←A - 1,影响 P 标志
DEC   Rn              ;Rn←Rn - 1
DEC   direct          ;(direct)←(direct) - 1
DEC   @Ri             ;((Ri))←((Ri)) - 1
```

这组指令是单字节指令，其功能是把指令中所指出的变量减 1，且不影响除 P 以外其余 PSW 中的任何标志位。若变量原来为 00H，减 1 后将溢出为 FFH，标志也不会受到影响。

【例 3-10】 已知(A)＝95H，(R0)＝63H，(CY)＝1，执行指令“SUBB A，R0”的结果。

【解】 结果为：(A)＝31H，(CY)＝0，AC＝0，OV＝1，P＝1。

3. 乘法指令

在 MCS-51 单片机中，乘法指令只有 1 条：

```
MUL   AB
```

该指令的功能是将累加器 A 和寄存器 B 中的两个 8 位无符号数相乘，乘积为 16 位，高

8 位放在 B 中，低 8 位放在 A 中。对于 PSW 中标志位的影响情况如下：

① CY 总是被清零；

② 若乘积大于 FFH(255)，则 OV＝1，否则 OV＝0；

③ 累加器 A 中有奇数个“1”时，P＝1，反之 P＝0。

值得注意的是，在 51 单片机的指令系统中，乘法和除法指令的目的操作数和源操作数必须是 A 和 B，且目的操作数和源操作数之间没有“,”分割符。

4. 除法指令

在 MCS-51 单片机中，除法指令也只有 1 条：

```
DIV   AB
```

该指令的功能是将累加器 A 中的 8 位无符号二进制数除以寄存器 B 中的 8 位无符号二进制数，商的整数部分存放在累加器 A 中，余数部分存放在寄存器 B 中。对于 PSW 中标志位的影响情况如下：

① CY 总是被清 0；

② 当除数为 0 时，OV＝1；

③ 累加器 A 中有奇数个“1”时，P＝1，反之 P＝0。

5. 十进制调整指令

在 MCS-51 单片机中，十进制调整指令只有 1 条：

```
DA    A
```

该指令的功能是在进行 BCD 码加法运算时，跟在 ADD 或 ADDC 后，对两个 BCD 码相加后存放在累加器 A 中的运算结果进行十进制调整。两个压缩的 BCD 码按二进制相加后，必须经过调整方能得到正确的压缩 BCD 码的和。十进制调整指令执行后，PSW 中的 CY 表示结果的百位值。调整的具体过程如下：

(1) 若累加器 A 的低 4 位为十六进制的 A～F 或辅助进位 AC 为 1，则累加器 A 中的内容加 06H 调整。

(2) 若累加器 A 的高 4 位为十六进制的 A～F 或辅助进位标志 CY 为 1，则累加器 A 中的内容加 60H 调整。

因为指令是对 BCD 码进行修正，所以该指令也称为 BCD 码修正指令。两个压缩 BCD 码按二进制数相加后，必须经本指令的调整才能得到正确的结果。

【例 3-11】　在 R3 中有十进制数 67，在 R2 中有十进制数 85，用十进制运算，运算的结果放于 R5 中。

【解】　程序为

```
MOV   A,R3
ADD   A,R2
DA    A
MOV   R5,A
```

程序中 ADD 指令运算出来的结果放于累加器 A 中，DA 指令对该结果进行调整。调整后，累加器 A 中的内容为 52H，CY 为 1，结果为 152，最后放于 R5 中的内容为 52H(十进制数 52)。

3.3.3 逻辑操作指令

逻辑运算指令完成字节的逻辑与、或、异或、清零、取反和左右移位等操作。当以 A 为目的操作数时，对 PSW 寄存器中的 P 标志有影响，循环指令是对 A 的循环。

1. 逻辑“与”指令 ANL

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

这组指令的功能是将源操作数单元的内容与目的操作数单元的内容按位相与，结果存放到目的操作数单元中，而源操作数单元中的内容不变。指令运行时仅影响 P 标志位。

逻辑与运算指令常用于将某些位屏蔽，只要将需要屏蔽的位和“0”相与，要保留的位和“1”相与即可。另外，如果使用该指令修改一个输出口时，作为原始数据的值将从单片机的输出数据锁存器(P0～P3)读入，而不是读引脚的状态。例如，

```
ANL   P2,#0F            ;屏蔽 P2 口的高 4 位,低 4 位保持不变
```

2. 逻辑“或”指令 ORL

```
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
```

这组指令的功能是将源操作数单元的内容与目的操作数单元的内容按位相或，结果存放到目的操作数单元中，而源操作数单元的内容不变。指令运行时仅影响 P 标志位。

逻辑或指令常用于将某些位置位，即使之为“1”，只要将需要置位的位与“1”相或，要保留的位与“0”相或即可。另外，如果使用该指令修改一个输出口时，作为原始数据的值将从单片机的输出数据锁存器(P0～P3)读入，而不是读引脚的状态。

3. 逻辑“异或”指令 XRL

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

这组指令的功能是将源操作数单元的内容与目的操作数单元的内容相异或，结果存放到目的操作数单元中，而源操作数单元中的内容不变。指令执行时仅影响 P 标志位。

逻辑异或指令常用于将某些位取反，即使“0”位变为“1”，使“1”位变为“0”。只要将需要取反的位与“1”相异或，要保留的位与“0”相异或即可。另外，如果使用该指令修改一个输出

口时，作为原始数据的值将从单片机的输出数据锁存器（P0～P3）读入，而不是读引脚的状态。

4. 清零指令和求反指令

（1）清零指令

```
CLR   A                ;A←0
```

（2）求反指令

```
CPL   A                ;A← Ā
```

在 MCS-51 系统中，只能对累加器 A 中的内容进行清零和求反，如果要对其他的寄存器或存储器单元进行清零和求反，则需要放在累加器 A 中进行，运算后再放回原位置。

【例 3-12】　写出对 R1 寄存器内容求反的程序段。

【解】　程序为

```
MOV   A,R1
CPL   A
MOV   R1,A
```

5. 循环移位指令

MCS-51 系统有 4 条对累加器 A 的循环移位指令：

（1）循环左移

```
RL    A
```

指令的功能是累加器 A 的 8 位向左循环移位，位 7 循环移入位 0，不影响标志位，如图 3-9(a)所示。

（2）循环右移

```
RR    A
```

指令的功能是累加器 A 的 8 位向右循环移位，位 0 循环移入位 7，不影响标志位，如图 3-9(b)所示。

（3）带进位的循环左移

```
RLC   A
```

指令功能是将累加器 A 中的内容和进位标志位 CY 一起向左循环移位 1 位，位 7 移入

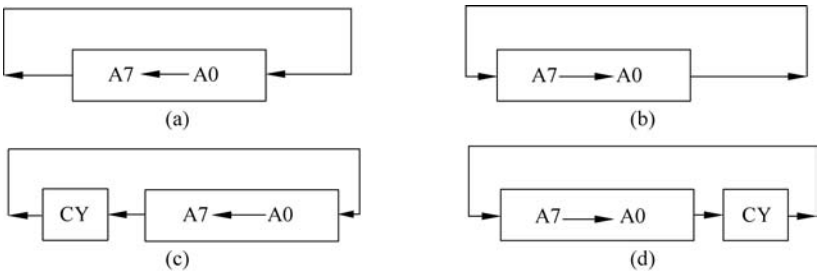

图 3-9　循环指令移位示意图

进位为 CY，CY 移入位 0，不影响标志位，如图 3-9(c)所示。

值得注意的是，“累加器 A 内容乘 2”的任务可以利用该指令方便地完成。

(4) 带进位的循环右移

```
RRC   A
```

指令功能是将累加器 A 中的内容和进位标志位 CY 一起向左循环移位 1 位，位 0 移入进位为 CY，CY 移入位 0，不影响标志位，如图 3-9(d)所示。

3.3.4 控制转移指令

通常情况下，程序的执行是顺序进行的，但也可以根据需要改变程序的执行顺序，称为程序转移。控制程序的转移要利用转移指令。MCS-51 单片机的控制转移指令包括无条件转移指令、条件转移指令及子程序调用与返回指令。

1. 无条件转移指令

无条件转移指令是指当执行该指令后，程序将无条件地转移到指令指定的地方。无条件转移指令包括长转移指令、绝对转移指令、相对转移指令和间接转移指令。

(1) 长转移指令(LJMP)

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

该指令为 3 字节指令，提供了 16 位的转移目标地址，指令执行时将该 16 位地址送给程序指针 PC，程序无条件地转移到 addr16 指出的目标地址。该指令可以使程序在 64KB 范围内跳转，且不影响标志位，使用方便，但执行时间长，字节数多。

【例 3-13】 执行下段程序后，求 PC 的值。

```
1000H Table:       MOV    A,＃21H
                   …
                   LJMP   Table
```

【解】 结果：PC＝1000H

指令中 addr16 及后面 LJMP 指令中的 rel 常用目的地址的地址标号来代替，由汇编程序自动换成 16 位或 8 位相对地址值。例如例 3-13 中的“Table”代表的就是 1000H。

(2) 绝对转移指令(AJMP)

```
AJMP   addr11              ;PC←PC＋2,PC10～0←addr11,PC15～11 不变
```

该指令为双字节指令，指令提供 11 位地址，指令执行时，先将 PC 的内容＋2(这时 PC 指向的是 AJMP 的下一条指令)，然后把指令中 11 位地址传送到 $PC_{10\sim0}$，而 $PC_{15\sim11}$ 保持不变。因为该指令只提供低 11 位地址，高 5 位地址为原 $PC_{15\sim11}$ 的值，所以程序转移位置以当前指令位置为基准，向前或向后转移 2KB 以内的范围，该指令不影响标志位。

【例 3-14】 1030H：AJMP　100H

【解】 目的地址：PC＝1032H 的高 5 位＋100H 的低 11 位＝1100H

(3) 相对转移指令(SJMP)

```
SJMP   rel                ;PC←PC + 2 + rel
```

该指令为双字节指令,指令执行时,先将程序指针 PC 的值+2(该指令的长度),然后再将 PC 的值与指令中的位移偏移量 rel 相加得到目的地址。rel 是一个有符号的 8 位二进制补码,取值范围是-128～+127(00H～7FH 表示 0～+127,80H～FFH 表示-128～-1),负数表示反向转移,正数表示正向转移。与 LJMP 指令相同,SJMP 指令中的相对地址 rel 常常用目的地址的标号(符号地址)表示。

在单片机程序设计中,为等待中断或程序结束,常有使用程序"原地踏步"的需要,可使用 SJMP 指令实现,即

```
LOOP: SJMP   LOOP
```

或者

```
LOOP: SJMP $              ;指令机器码为 80H,"$"表示 PC 的当前值
```

(4) 间接转移指令(JMP)

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

该指令是一条单字节指令,其转移地址由数据指针 DPTR 的 16 位数和 A 的 8 位数进行无符号数相加形成,并直接装入 PC,可以实现 64KB 范围内的条件转移。指令执行后不改变 DPTR 及 A 中原来的内容,也不影响标志位。

该指令以 DPTR 的内容为基地址,A 的内容作变址,因此只要把 DPTR 的值固定,然后赋予 A 不同的值,即可实现程序的多分支转移。间接转移指令因为可以代替众多的判别跳转指令,又称为散转指令。

【例 3-15】　已知累加器 A 中存放着控制程序转向的编号 0～3,ROM 中存有起始地址为 TABLE 的双字节绝对转移指令表,试编写程序使单片机能按照累加器 A 中的编号转去执行相应的命令程序,即当 A=00H 时,执行 TAB0 分支程序; 当 A=01H 时,执行 TAB1 分支程序; A=02H 时,执行 TAB2 分支程序; A=03H 时,执行 TAB3 分支程序。

【解】　参考代码:

```
        CLR     C
        RLC     A
        MOV     DPTR,#TABLE       ;将 TABLE 地址送入 DPTR 中
        JMP     @A + DPTR         ;程序转到地址为 A + DPTR 的地址中执行
TABLE:  AJMP    TAB0              ;当 A = 0 时,执行 TAB0 分支程序
        AJMP    TAB1              ;当 A = 1 时,执行 TAB1 分支程序
        AJMP    TAB2              ;当 A = 2 时,执行 TAB2 分支程序
        AJMP    TAB3              ;当 A = 3 时,执行 TAB3 分支程序
        …
```

因为表格中的 AJMP 是双字节指令,所以执行查表指令之前应将累加器 A 的内容乘以 2,以形成正确的偏移量。

2. *条件转移指令*

条件转移指令是指当条件满足时,程序转移到指定位置,条件不满足时,程序将继续顺

次执行。在 MCS-51 系统中，条件转移指令有三种：累加器 A 判零条件转移指令、比较转移指令和减 1 不为零转移指令。

（1）累加器 A 判零转移指令（JZ/JNZ）

```
JZ      rel     ;若 A = 0,则 PC←PC + 2 + rel,否则 PC←PC + 2
JNZ     rel     ;若 A≠0,则 PC←PC + 2 + rel,否则 PC←PC + 2
```

这两条指令都是双字节指令，rel 为相对地址偏移量，当各自条件满足时，程序转移的目标地址为 PC＋2＋rel，指令中的相对地址 rel 常常用目的地址的标号（符号地址）表示。该指令的转移范围在以 PC 当前值为起始地址的－128～＋127B 范围内。

【例 3-16】 把片外 RAM 30H 单元开始的数据块传送到片内 RAM 的 40H 开始的位置，直到出现零为止。

【解】 参考程序如下：

```
        MOV     R0,＃30H
        MOV     R1,＃40H
LOOP:   MOVX    A,@R0
        MOV     @R1,A
        INC     R1
        INC     R0
        JNZ     LOOP
        SJMP    $
```

（2）比较转移指令（CJNE）

```
CJNE    A,＃data,rel        ;当 A≠data,则 PC←PC + 3 + rel,否则 PC←PC + 3
CJNE    Rn,＃data,rel       ;当 Rn≠data,则 PC←PC + 3 + rel,否则 PC←PC + 3
CJNE    @Ri,＃data,rel      ;当(Ri)≠data,则 PC←PC + 3 + rel,否则 PC←PC + 3
CJNE    A,direct,rel        ;当(direct)≠data,则 PC←PC + 3 + rel,否则 PC←PC + 3
```

这组指令是三字节指令，功能是对两个规定的操作数进行比较，并且根据比较的结果决定是否转移。若两个操作数相等，则程序顺序执行；若两个操作数不相等，则程序转移到目标地址 PC＋3＋rel。指令中的地址 rel 通常也用目的地址的标号（符号地址）表示。

因为指令执行过程中通过减法操作（不保存两数的差）实现操作数比较，所以可以通过标志位 CY 反映操作数的大小：如果目的操作数的内容大于源操作数的内容，则 CY＝0；如果目的操作数的内容小于源操作数的内容，则 CY＝1。所以，在程序转移后可以利用标志位 CY 做进一步判断，可实现三分支转移。

【例 3-17】 已知工作寄存器 R0 中存放着一个无符号数 X，试编写程序求出下式的函数值 Y，并存入工作寄存器 R1 中。

$$Y=\begin{cases}\text{AAH} & X>20\text{H}\\ 00\text{H} & X=20\text{H}\\ \text{FFH} & X<20\text{H}\end{cases}$$

【解】 参考程序如下：

```
MOV   A,R0
CJNE  A,＃20H,L1
MOV   R1,＃0
```

```
        LJMP L3
L1:     JC   L2          ;若 CY = 1,则跳转值 L2,否则顺序执行
        MOV  R1,#0AAH
        LJMP L3
L2:     MOV  R1,#0FFH
L3:     LJMP L3
```

（3）减 1 不为零转移指令(DJNZ)

```
DJNZ    Rn,rel;Rn←Rn - 1,若 Rn≠0,则 PC←PC + 2 + rel
DJNZ    direct,rel;(direct)←(direct) - 1,若(direct)≠0,则 PC←PC + 2 + rel
```

该组指令将减 1 和条件转移结合到一起，执行指令时，首先将操作数的内容减 1，并将结果存在第一操作数中，然后判断结果是否为 0。若不为 0，转移到目标地址，否则顺序执行。

该组指令对于构成循环程序是十分有用的，可以指定任何一个工作寄存器作为程序循环计数器，每循环一次，这种指令被执行一次，计数器就减 1，预定的循环次数不到，计数器不会为 0，继续执行循环操作；当到达预定的循环次数，计数器就被减至 0，顺序执行下一条指令，结束循环。

【例 3-18】　统计片内 RAM 中 32H 单元开始的 10 个数据中 0 的个数，放于 R7 中。

【解】　参考程序：

```
        MOV  R0,#32H
        MOV  R2,#10H
        MOV  R7,#0
LOOP:   MOV  A,@R0
        CJNE A,#0,NEXT
        INC  R7
NEXT:   INC  R0
        DJNZ R2,LOOP
        SJMP $
```

3. 子程序调用与返回指令

在一个程序中经常遇到反复多次执行某段程序的情况，如果重复执行编写这个程序段，会使程序变得冗长而杂乱。而子程序结构，可以将重复的程序片段编写为一个子程序，通过主程序调用而使用，既能减少编程的工作量，也能缩短程序的长度。主程序和子程序之间的调用关系如图 3-10 所示。

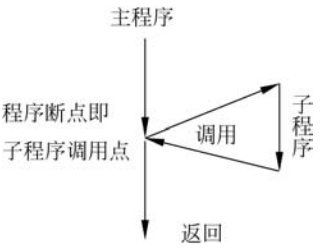

图 3-10　主程序和子程序之间的调用关系

子程序调用及返回指令有 4 条：2 条程序调用指令和 2 条程序返回指令。调用和返回指令必须成对使用。调用指令具有把断点地址保护到堆栈以及把子程序入口地址自动送入 PC 的功能，返回指令则具有把堆栈中的断点地址自动恢复到 PC 的功能。

（1）长调用指令 LCALL

```
LCALL   addr16      ;PC←PC + 3,SP←SP + 1,(SP)←PC7~0,
                    SP←SP + 1,(SP)←PC15~8,PC10~0←addr16
```

该指令为三字节指令，指令提供16位目标地址，可调用64KB范围内的子程序。执行指令时，首先修改PC←PC+3，以获得下一条指令地址；然后将这16位地址（断点值，即返回到LCALL指定的下一条指令地址）压入堆栈（先压入$PC_{7\sim0}$低位字节，后压入$PC_{15\sim8}$高位字节），堆栈指针SP加2指向栈顶；接着将16位目标地址addr16送入程序计数器$PC_{10\sim0}$，从而使程序转向目标地址去执行被调用的子程序。

（2）绝对调用指令ACALL

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

该指令为双字节指令，指令提供11位目标地址，只能调用与PC在同一个2KB绝对地址范围内的子程序。执行指令时，首先修改PC←PC+2，作为下一条指令地址，并把修改后的PC内容（先低字节，后高字节）压入堆栈保护，堆栈指针SP加2指向栈顶；接着将11位目标地址addr11送入程序计数器$PC_{10\sim0}$，从而使程序转向目标地址去执行被调用的子程序。

值得说明的是，两条调用指令执行时要将返回地址入栈，初始化时必须设置合适的SP值（SP默认值为07H）。在汇编程序中，调用指令后面通常带转移地址的标号。

（3）子程序返回指令RET

```
RET    ;PC15~8←(SP),SP←SP - 1,PC7~0←(SP),SP←SP - 1
```

该指令通常放在子程序的最后一条指令位置，用于实现返回到主程序。指令执行时，将子程序调用指令压入堆栈的地址出栈，第一次出栈的内容送到PC的高8位，第二次出栈的内容送到PC的低8位。指令执行完后，程序将返回到调用指令的下一条指令执行。

（4）中断返回指令

```
RETI   ;PC15~8←(SP),SP←SP - 1,PC7~0←(SP),SP←SP - 1
```

该指令作为中断服务子程序的最后一条指令，用于返回主程序中断的断点位置，继续执行断点位置后面的指令。该指令的执行过程与RET基本相同，只是RETI在执行后，在转移前将先清除中断的优先级触发器。

MCS-51系统中，中断都是硬件中断，没有软件中断调用指令。硬件中断时，由一条长转移指令使程序转移到中断服务程序的入口位置，在转移之前，由硬件将当前的断点地址压入堆栈保存，以便以后通过中断返回指令返回到断点位置后继续执行。

值得注意的是，虽然这两条指令均是把堆栈中的断点地址恢复到PC中，从而使单片机回到断点处执行程序，但二者具有如下区别：

① RET指令为子程序的最后一条指令，而RETI为中断服务程序的最后一条指令，二者不能互换使用。

② RETI指令除了恢复断点地址外，还恢复CPU响应中断时硬件自动保护的现场信息，如将清除中断响应时所置1的优先级状态触发器，使得已申请的同级或低级中断申请可以响应，而RET指令只能恢复返回地址。

③ 对开发者而言，RET指令返回的地址是确定的，而RETI指令返回的地址是未知的。

3.3.5　位操作指令

位操作又称为布尔变量操作，以位(bit)为单位进行运算和操作，由单片机中一个布尔处理机实现，借用进位标志 CY 作为累加器。位操作指令共 17 条，可以实现位传送、位逻辑运算、位控制转移等操作。

1. 位传送指令

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

该组指令的功能是实现 CY 和其他位地址之间的相互数据传送。在程序中，CY 记做 C。

两个位地址间不能直接进行数据传送，可以通过 CY 间接实现。

【例 3-19】　把片内 RAM 中位寻址区的 30H 位的内容传送到 20H 位。

【解】　参考程序：

```
MOV    C,30H
MOV    20H,C
```

2. 位逻辑操作指令

位逻辑操作指令包括清 0、置 1、取反、与和或，共 10 条指令。

(1) 位清 0

```
CLR    C              ;CY←0
CLR    bit            ;(bit)←0
```

(2) 位置 1

```
SETB   C              ;CY←1
SETB   bit            ;(bit)←1
```

(3) 位取反

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

(4) 位与

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

(5) 位或

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

指令中位地址 bit 若为 00H～7FH，则位地址在片内 RAM(20H～2FH)中，共 128 位，bit 若为 80H～FFH，则位地址在 11 个特殊功能寄存器中。其中，有 4 个 8 位的并行 I/O 口，每位均可单独进行操作。因此，布尔 I/O 口共有 32 个，分别为 P0.0～P0.7、P1.0～P1.7、P2.0～P2.7 和 P3.0～P3.7。

【例 3-20】

```
MOV    P1.0,C          ;P1.0←CY,把标志位 CY 的内容送入 P1 口最低位 P1.0 中
```

【例 3-21】

```
SETB   P3.0            ;把 P3 口的最低位置 1
CLR    07H             ;把 20H 字节的最高位清 0
```

区分：CLR　A ；只有唯一的一条清累加器 A 的指令(整个字节清 0)，因此，上述 07H 必定属于位地址。

【例 3-22】　利用位逻辑运算指令编程实现如图 3-11 所示的硬件逻辑电路的功能。

【解】　参考程序：

```
MOV    C,P1.0
ANL    C,P1.1
MOV    F0,C
MOV    C,P1.1
ORL    C,P1.2
ANL    C,F0
CPL    C
```

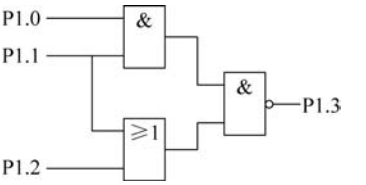

图 3-11　硬件逻辑电路图

3. 位转移指令

位转移指令有以 C 为条件的转移指令和以 bit 为条件的转移指令，共 5 条。

(1) 以 C 为条件的转移指令

```
JC     rel             ;若 CY = 1,则转移,PC←PC + 2 + rel;否则程序继续执行
JNC    rel             ;若 CY = 0,则转移,PC←PC + 2 + rel;否则程序继续执行
```

该组指令是相对转移指令，以 CY 的值来判决程序是否需要转移，常常和比较条件转移指令 CJNE 连用，以便根据 CJNE 指令执行过程中形成的 CY 进一步判决程序的流向或形成三分支模式。

(2) 以 bit 为条件的转移指令

```
JB     bit,    rel     ;若(bit) = 1,则转移,PC←PC + 3 + rel;否则程序继续执行
JNB    bit,    rel     ;若(bit) = 0,则转移,PC←PC + 3 + rel;否则程序继续执行
JBC    bit,    rel     ;若(bit) = 1,则转移,PC←PC + 3 + rel,且(bit)←0;否则程序继续执行
```

【例 3-23】　如图 3-12 所示，P3.2 和 P3.3 上各接有一个按键，要求它们分别按下时(P3.2＝0 或 P3.3＝0)，分别使 P1 口输出 0 或 FFH。

【解】　因为 P3 口是准双向口，所以在读取按键状态之前，要先通过程序使 P3 口输出高电平，然后再一次读取 P3.2 和 P3.3 的状态并进行判断。

参考程序：

```
       MOV     P3,#0FFH
L1:    JNB     P3.2,L2
       JNB     P3.3,L3
       LJMP    L1
L2:    MOV     P1,#00H
       LJMP    L1
```

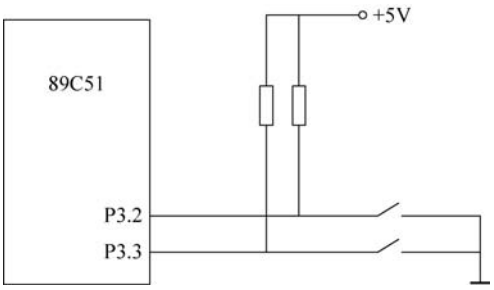

图 3-12　例 3-23 电路图

```
L3:   MOV     P1,#0FFH
      LJMP    L1
```

【例 3-24】　从片外 RAM 20H 单元开始有 50 个数据，统计其中正数、0 和负数的个数，分别放于 R5、R6 和 R7 中。

【解】　设 R2 作计数器，用 DJNZ 指令对 R2 减 1 转移进行循环控制，在循环体外设置 R0 指针，指向片外 RAM 20H，对 R5、R6 和 R7 清零，在循环体中用指针 R0 依次取出片外 RAM 中的 50 个数据，然后判断，若大于 0，则 R5 中的内容加 1；若等于 0，则 R6 中的内容加 1；若小于 0，则 R7 中的内容加 1。

参考程序：

```
      MOV     R2,#50
      MOV     R0,#20H
      MOV     R5,#0
      MOV     R6,#0
      MOV     R7,#0
LOOP: MOVX    A,@R0
      CJNE    A,#0,NEXT1
      INC     R6
      SJMP    NEXT3
NEXT1:CLR     C
      SUBB    A,#80H
      JC      NEXT2
      INC     R7
      SJMP    NEXT3
NEXT2:INC     R5
NEXT3:INC     R0
      DJNZ    R2,LOOP
      SJMP    $
```

4. 空操作指令

空操作指令只有 1 条单字节指令：

```
NOP   ;PC←PC+1
```

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。该指令要占用一个机器周期，常用来产生延迟，构造延时程序。

3.4 51系列单片机汇编程序常用伪指令

3.3节介绍51系列单片机的汇编语言指令系统。汇编语言源程序是用汇编语言编写的程序，必须翻译成机器代码才能运行，而汇编就是翻译的过程。伪指令是放在汇编语言源程序中，对汇编过程进行相应的控制和说明的指示性命令。该类指令可以被汇编器识别，但是没有对应的可执行机器码，汇编产生的目标程序中不会再出现伪指令。伪指令通常用于定义数据、分配存储空间、控制程序的输入输出等。MCS-51汇编语言源程序常用的伪指令有以下几条：

1. 状态控制伪指令

(1) 起始地址设定伪指令ORG

格式：

```
ORG    表达式
```

该伪指令放在一段源程序或数据的前面，其功能是说明下面紧接的代码或数据存放的起始地址。表达式通常为十六进制地址，也可以是已定义的地址标号。

【例3-25】 ORG伪指令的使用。

```
      ORG      2000H
START:MOV      A,#40H
```

【解】 该段程序的机器码从地址2000H单元开始存放。

通常情况下，每个汇编源程序的开始，都需要利用ORG来指明该程序存放的起始位置。若省略，则默认该程序存放的起始地址为0000H。

(2) 汇编器结束汇编伪指令END

格式：

```
END
```

该伪指令放于程序的最后，用于控制汇编器结束汇编。一个源程序只能有一个END命令，否则就有一部分指令不能被汇编。

2. 符号伪指令

(1) 定义常值为符号名伪指令EQU

格式：

```
符号名   EQU    常值表达式
```

该伪指令的功能是定义一个指定的符号名。在汇编过程中，汇编器会将源程序中出现的该符号用定义的常值来取代。值得注意的是，标号只代表地址，而符号名可以代表地址、常数、段名或字符串、寄存器或位名等，符号名不后接冒号。

【例 3-26】　EQU 伪指令的使用。

```
L      EQU     12
SUM    EQU     41H
BLOCK  EQU     42H
       CLR     A
       MOV     R1,#L
       MOV     R0,#BLOCK
LOOP:  ADD     A,@R0
       INC     R0
       DJNZ    R1,LOOP
       MOV     SUM,A
```

【解】　该程序段功能是把 22H 单元开始存放的 10 个无符号数求和，并将结果存放入 41H 单元。程序中的符号 L、SUM 和 BLOCK 在汇编后将分别由常值 12、41H 和 42H 取代。

(2) 定义地址符号名伪指令 BIT

格式：

```
符号名  BIT    位地址表达式
```

该伪指令的功能是将位地址赋予指定的符号名。位地址表达式可以是绝对地址，也可以是符号地址。

值得注意的是，用 EQA 或 BIT 定义的“符号名”一经定义便不能重新定义和改变。

【例 3-27】　BIT 伪指令的使用。

```
ST      BIT    P1.0            ;将 P1.0 的位地址赋予符号名 ST
CF      BIT    0D7H            ;将位地址 D7H 的位定义为符号名 CF
```

3. *存储器空间初始化伪指令*

(1) 定义字节数据表伪指令 DB

格式：

```
[标号:]DB  字节数据表
```

该伪指令的功能是从标号指定的地址单元开始，在程序存储器中定义字节数据表，可以定义一个字节，也可以定义多个字节。定义多个字节时，两量之间用逗号间隔，定义的多个字节在存储器中是连续存放的。定义的字节可以是一般常数、字符或者字符串。字符和字符串以引号括起来，字符数据在存储器中以 ASCII 码形式存放。

【例 3-28】　DB 伪指令的使用。

```
      ORG    0100H
TAB1: DB     56H,78H
      DB    '4','a','ABC'
```

【解】　汇编后，各个数据在存储单元中的存放情况如图 3-13 所示。

(2) 定义字数据表伪指令 DW

格式：

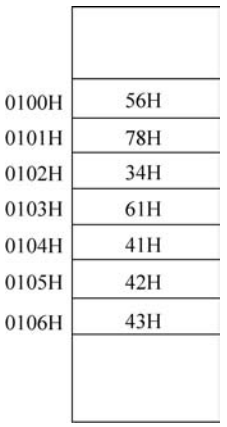

图 3-13　DB 数据分配情况

[标号:]DW　字数据表

该伪指令与 DB 类似,用于定义字数据。项或项表所定义的一个字节在存储器中占 2 字节。汇编时,机器自动按高字节在前、低字节在后存放,即高字节存放在低地址单元,低字节存放在高地址单元。该伪指令常用于在 ROM 中定义双字节的跳转地址。

【例 3-29】　DW 伪指令的使用。

```
      ORG     0100H
TAB2:DW   1234H,5678H
```

【解】　汇编后,各个数据在存储单元中的存放情况如图 3-14 所示。

4. 保留空间伪指令

(1) DS 伪指令

格式:

[标号:]DS　　数值表达式

该伪指令用于在存储器中保留一定数量的字节单元,保留字节单元的数目由表达式的值决定。

【例 3-30】　DS 伪指令的使用。

```
ORG     0100H
DB      34H,56H
DS      4H
DB      '7'
```

【解】　汇编后,各个数据在存储单元中的存放情况如图 3-15 所示。

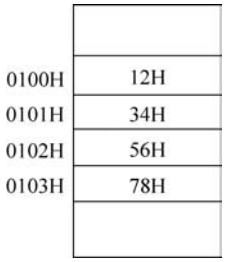

图 3-14　DW 数据分配情况

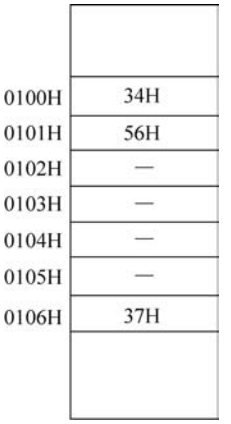

图 3-15　DS 数据分配情况

(2) DBIT 伪指令

格式:

[标号:]DBIT　数值表达式

该伪指令与 DS 类似,以位为单位保留存储空间。

5. 其他伪指令

为了支持模块化设计,51 汇编器还提供了一些其他伪指令,如表 3-3 所示。

表 3-3　其他伪指令

分　类	伪 指 令	功　能	示　例
段及类型定义	SEGMENT	段名定义	bedbin　SEGMENT　ODE
	RSEG	可重新定位段说明	RSEG　edbin
	CODE	程序代码空间	KEY　CODE　0H
	DATA	直接寻址数据空间	one　DATA　30H
	IDATA	间接寻址数据空间	two　IDATA　80H
	XDATA	扩展的外部数据空间	SUM　XDATA　1000H
地址指定	BSEG AT	定位位寻址段	BSEG　AT　20H
	CSEG AT	定位代码段	CSEG　AT　0000H
	DSEG AT	定位内部 RAM 直接寻址数据段	DSEG　AT　40H
	ISEG AT	定位内部 RAM 间接寻址数据段	ISEG　AT　80H
	XSEG AT	定位外部 RAM 数据段	XSEG　AT　1000H
模块连接	PUBLISH	符号在本模块定义,其他模块引用	PUBLISH　_BIN2BCDB
	EXTRN	符号在本模块引用,其他模块定义	EXTRN CODE(_BIN2BCDB)
其他	USING	指定当前工作寄存器组	USING　1

3.5　51 单片机汇编语言程序设计举例

3.5.1　概述

用汇编语言设计程序与用高级语言设计程序有相似之处,设计过程大致可以分为以下几个步骤:

(1) 明确课题的具体内容。明确对程序的功能、运算精度、执行速度等方面的要求及硬件条件。

(2) 编制框图,绘出流程图。把复杂问题分解为若干个模块,确定各模块的处理方法,画出程序流程图。对复杂问题可分别画出分模块流程图和总的流程图。

(3) 存储器资源分配。根据系统的工作需要,合理选择并分配内存单元及工作寄存器。

(4) 编写源程序。根据程序流程图精心选择合适的指令和寻址方式来编制源程序。

(5) 对程序进行汇编、调试和修改。对编制好的源程序进行汇编,检查修改程序中的错误,执行目标程序,对程序运行结果进行分析,直至正确为止。

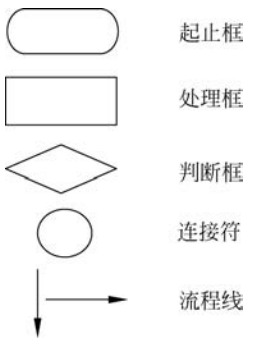

图 3-16 几何图形符号图

另外，用汇编语言进行程序设计时，对于程序、数据在存储器的存放位置，对工作寄存器、片内数据存储单元、堆栈空间等的安排都要由编程者亲自做，编写过程中要特别注意。

画流程图是指用各种图形、符号、指向线等来说明程序设计的过程，常用几何图形符号如图 3-16 所示。

国际通用的图形和符号说明如下：

(1) 椭圆框：起止框，在程序的开始和结束时使用。

(2) 矩形框：处理框，表示要进行的各种操作。

(3) 菱形框：判断框，表示条件判断以决定程序的流向。

(4) 圆圈：连接符，表示不同页之间的流程连接。

(5) 指向线：流程线，表示程序执行的流向。

3.5.2 顺序程序设计

顺序结构程序是一种最简单、最基本的程序，所以有时也称为简单程序设计，是一种无分支的直线型结构，即按照程序的编写顺序依次执行每条指令。

【例 3-31】 设片内 RAM 的 21H 单元存放一个十进制数据十位的 ASCII 码，22H 单元存放该数据个位的 ASCII 码。编写程序将该数据转换成压缩 BCD 码存放在 20H 单元。

【解】 由于 ASCII 码 30H～39H 对应 BCD 码的 0～9，所以只要保留 ASCII 码的低 4 位，而将高 4 位清 0 即可。流程图如图 3-17 所示。

参考程序：

```
        ORG     0000H
        LJMP    START
        ORG     0040H
START:  MOV     A,21H           ;取十位 ASCII 码
        ANL     A,#0FH          ;保留低半字节
        SWAP    A               ;移至高半字节
        MOV     20H,A           ;存于 20H 单元
        MOV     A,22H           ;取个位 ASCII 码
        ANL     A,#0FH          ;保留低半字节
        ORL     20H,A           ;合并到结果单元
        SJMP    $
        END
```

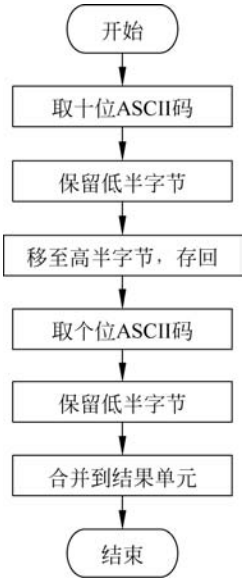

图 3-17 例 3-31 程序流程图

【例 3-32】 已知两个压缩 BCD 码分别放在内部 RAM 的 31H、30H 和 33H、32H 这 4 个单元中，试编程求和，结果存入 R4、R3、R2 中。

【解】 参考程序：

```
        ORG     0000H           ;程序开始
        LJMP    MAIN            ;跳转到 MAIN 标号
        ORG     0040H           ;MAIN 程序段的起始命令
MAIN:   MOV     A,30H           ;将 30H 单元的内容送给 A
        ADD     A,32H           ;将 32H 单元的内容与 A 的内容相加
        DA      A               ;进行 BCD 码调整
```

```
        MOV     R2,A            ;将 A 的内容送 R2 保存,保存低位的和
        MOV     A,31H           ;将高位 31H 的内容送给 A
        ADDC    A,33H           ;33H 的内容与 A 的内容相加,同时加上低位的进位
        DA      A               ;进行 BCD 码调整
        MOV     R3,A            ;将 A 的内容送 R3,保存高位和
        CLR     A               ;清 A 的内容
        MOV     ACC.0,C         ;保存高位和的进位
        MOV     R4,A            ;在 R4 中保存高位的进位
HERE:   SJMP    HERE            ;程序在此运行
        END                     ;程序结束
```

3.5.3　分支程序设计

在实际应用过程中,往往需要单片机对某些条件进行判断,从而实现不同的程序流向,这样,就会产生一个或多个分支程序。分支程序设计可以分为单分支程序设计和多分支程序设计。

【例 3-33】　设变量 x 以补码的形式存放在片内 RAM 的 30H 单元,变量 y 与 x 的关系是:当 x 大于 0 时,$y=x$;当 $x=0$ 时,$y=20H$;当 x 小于 0 时,$y=x+5$。编制程序,根据 x 的大小求 y 并送回原单元。流程图如图 3-18 所示。

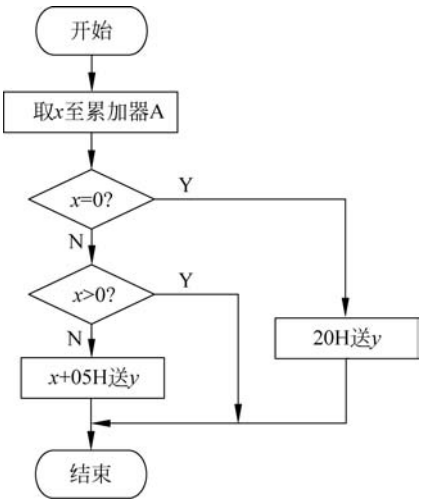

图 3-18　例 3-33 程序流程图

【解】　参考代码:

```
        ORG     0000H       ;程序开始
        LJMP    START       ;跳转到 START 标号
        ORG     0040H       ;START 程序段的起始命令
START:  MOV     A,30H       ;取 x 至累加器 A
        JZ      NEXT        ;x = 0,转 NEXT
        ANL     A,#80H      ;否,保留符号位
        JZ      DONE        ;x > 0,转结束
        MOV     A,#05H      ;x < 0,处理
        ADD     A,30H
        MOV     30H,A       ;x + 05H 送 y
        SJMP    DONE
NEXT:   MOV     30H,#20H    ;x = 0,20H 送 y
DONE:   SJMP    DONE
        END
```

【例 3-34】　假定在外部 RAM 中,有 ST1、ST2 和 ST3 共 3 个连续单元,其中 ST1、ST2 单元中分别存放两个 8 位无符号数,要求找出其中的大数并存入 ST3 单元。

【解】　比较两个无符号数的大小,可利用两数相减是否有借位来判断。

参考代码:

```
START:  CLR     C               ;清 C 标志
        MOV     DPTR,#ST1       ;将立即数 ST1 送 DPTR
        MOVX    A,@DPTR         ;取外部数据送 A
```

```
        MOV    R7,A              ;将 A 中数据送 R7
        INC    DPTR              ;DPTR 自动加 1
        MOVX   A,@DPTR           ;再取下一个外部数据
        SUBB   A,R7              ;对外部数据进行减法运算
        JC     B1                ;若 C 为 1,即(A)<(R7),程序转向 B1
        MOVX   A,@DPTR           ;再次将外部数据送 A
        SJMP   B2                ;程序转向 B2
B1:     XCH    A,R7              ;交换 A 和 R7 的值
B2:     INC    DPTR              ;DPTR 加 1
        MOVX   @DPTR,A           ;(A)送外部数据寄存器
        SJMP   $                 ;程序在此跳转
```

【例 3-35】 设变量 X 存放于 30H 单元，函数值 Y 存放于 31H 单元。若 $X>0$，则 $Y=1$；若 $X=0$，则 $Y=0$；若 $X<0$，则 $Y=-1$。

【解】 X 是有符号数，判断符号位是 0 还是 1，可利用 JB 或 JNB 指令。判断 X 是否等于 0，则直接可以使用累加器 A 的判 0 指令。流程图如图 3-19 所示。

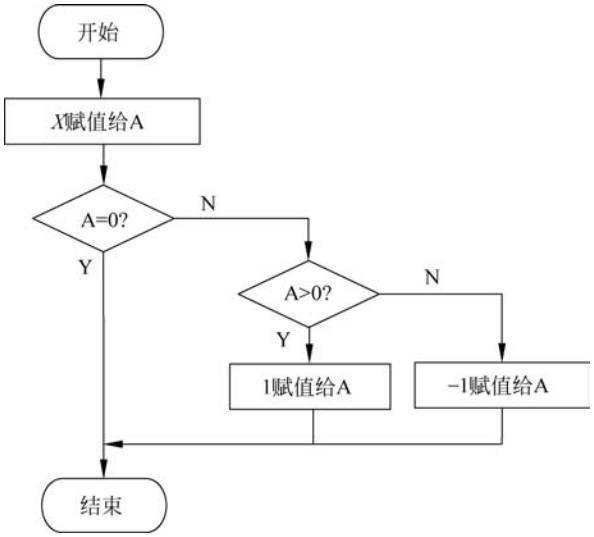

图 3-19 例 3-35 程序流程图

参考程序：

```
START:  MOV    A,30H
        JZ     OVER
        JNB    ACC.7,LAB1
        MOV    A,#0FFH
        SJMP   OVER
LAB1:   MOV    A,#1
OVER:   MOV    31H,A
        SJMP   $
```

注意，对于分支不多的应用程序，常采用的指令有 JZ、JNZ、JC、JNC、JB、JNB、CJNE 等。对于多分支的程序，需要采用 JMP @A+DPTR 指令。

3.5.4　循环程序设计

所谓循环程序，是指按照某种规律重复执行的程序，分为“先执行后判断”和“先判断后执行”两种结构。前者的特点是，先执行循环处理部分，然后根据循环控制条件判断是否结束循环，如图 3-20(a)所示；后者的特点是，先根据循环控制条件判断是否结束循环，若不结束，则执行循环处理部分，否则结束处理，退出循环，如图 3-20(b)所示。

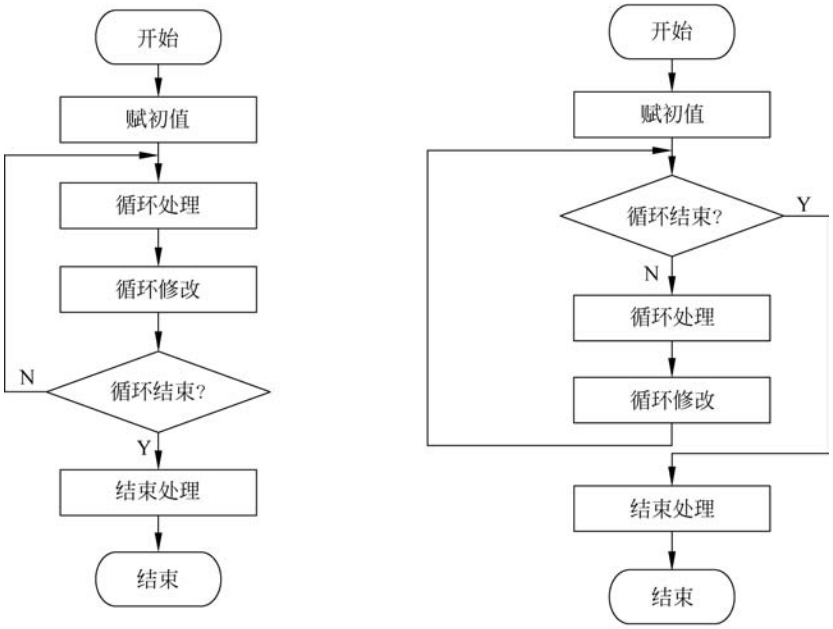

图 3-20　循环程序结构图

【例 3-36】　计算 50 个 8 位二进制数(单字节)之和。要求：50 个数存放在 30H 开头的内部 RAM 中以及 R6R7 中。

【解】　采用 DJNZ 循环体的程序流程图，如图 3-21 所示，在参考程序中，R0 为数据地址指针，R2 为减法循环计数器。在使用 DJNZ 控制时，循环计数器初值不能为 0，当为 0 时，第一次进入循环执行到 DJNZ 时，减 1 使 R2 变为 FFH，循环次数成了 256。

参考程序：

```
START:   MOV    R6,#0
         MOV    R7,#0
         MOV    R2,#50
         MOV    R0,#30H
LOOP:    MOV    A,R7
         ADD    A,@R0
         MOV    R7,A
```

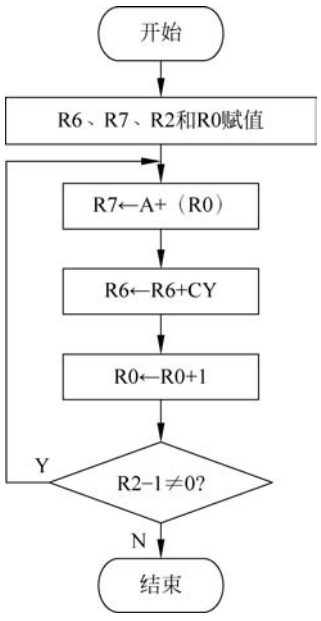

图 3-21　例 3-36 的循环程序结构图

```
        CLR   A
        ADDC  A,R6
        MOV   R6,A
        INC   R0
        DJNZ  R2,LOOP
        SJMP  $
```

【例 3-37】 编写程序,将内部 RAM 的 30H～3FH 单元初始化为 00H。

【解】 参考程序:

```
        ORG   0000H
        LJMP  MAIN
        ORG   0040H
MAIN:   MOV   0,#30H          ;置初值
        MOV   A,#00H
        MOV   R7,#16
LOOP:   MOV   @R0,A           ;循环处理
        INC   R0
        SJMP  $               ;结束处理
        END
```

3.5.5 查表程序设计

查表程序设计中用于查表的指令有两条:

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

当使用 DPTR 作为基址寄存器时,查表的步骤分为三步:

(1) 基址(表格首地址)送 DPTR 数据指针;

(2) 变址值(在表中的位置是第几项)送累加器 A;

(3) 执行查表指令 MOVC A,@A+DPTR,进行读数,查表结果送回到累加器 A 中。

当使用 PC 作为基址寄存器时,由于 PC 本身是一个程序计数器,与指令的存放地址有关,查表时的操作有所不同。查表的步骤也分三步:

(1) 变址值(在表中的位置是第几项)送累加器 A;

(2) 偏移量(查表指令的下一条指令的首地址到表格首地址之间的字节数)+A→A(修正);

(3) 执行查表指令 MOVC A,@A+PC。

【例 3-38】 编写 2 位十六进制数与 ASCII 码的转换程序。设数值在 R2 中,转换结果的低位存在 R2 中,高位存在 R3 中。

【解】

(1) 利用 DPTR 作基址的参考程序。

```
HEXASC:   MOV   DPTR,#TABIE
          MOV   A,R2
          ANL   A,#0FH
          MOVC  A,@A+DPTR              ;查表
```

```
        XCH     R2,A
        ANL     A,#0F0H
        SWAP    A
        MOVC    A,@A+DPTR               ;查表
        MOV     R3,A
        RET
TABLE:  DB      30H,31H,32H,33H,34H     ;ASCII 表
        DB      35H,36H,37H,38H,39H
        DB      41H,42H,43H,44H,45H,46H
```

（2）利用 PC 作基址的参考程序。

```
HEXASC: MOV     A,R2
        ANL     A,#0FH
        ADD     A,#9
        MOVC    A,@A+PC                 ;查表
        XCH     R2,A
        ANL     A,#0F0H
        SWAP    A
        ADD     A,#9
        MOVC    A,@A+PC                 ;查表
        MOV     R3,A
        RET
TABLE:  DB      30H,31H,32H,33H,34H     ;ASCII 表
        DB      35H,36H,37H,38H,39H
        DB      41H,42H,43H,44H,45H,46H
```

3.5.6　子程序设计

将需要多次使用、完成相同的某种基本运算或操作的程序段从整个程序中独立出来，单独编成一个程序段，需要时通过子程序调用指令进行调用。这样的程序段称为子程序。

采用子程序能使整个程序结构简单，缩短程序的设计时间，减少占用的程序存储空间。调用子程序的程序称为主程序或调用程序。

在编写子程序时应注意以下问题：①子程序的第一条指令地址称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来实现的；③子程序结构中使用堆栈保护断点和现场保护，且堆栈只能存于片内 RAM 中，不能存在于片外 RAM 中；④子程序的最后一条指令必须是 RET 指令，它的功能是把堆栈中的断点地址弹出。

典型的子程序的基本结构如下：

```
MAIN:   …                               ; MAIN 为主程序入口标号
        …
        LCALL   SUB                     ;调用子程序 SUB
        …
SUB:    PUSH    PSW                     ;现场保护
        ACALL   Acc
```

子程序处理程序段：

```
POP     Acc                          ;现场恢复,注意要先进后出
POP     PSW                                                    } 子程序
RET                                  ;最后一条指令必须为 RET
```

值得说明的是,上述子程序结构中,现场保护与现场恢复不是必需的,要根据实际情况而定。

【例 3-39】 利用子程序实现 $c=a^2+b^2$。设 a、b、c 分别存于内部 RAM 的 20H、21H 和 22H 三个单元中。

【解】

子程序入口:(A)=预平方数;

子程序出口:(A)=平方值。

子程序如下:

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

验证程序段如下:

```
MAIN:   MOV    20H,#4
        MOV    21H,#5
        MOV    A,20H                 ;利用累加器 A 向子程序传递参数 a
        ACALL  SQR
        MOV    R1,A                  ; a² 暂存于 R1 中
        MOV    A,21H                 ;利用累加器 A 向子程序传递参数 b
        ACALL  SQR
        ADD    A,R1                  ; a²+b² 存于 A 中
        MOV    22H,A                 ;A 中的结果存入 22H 单元
        SJMP   $
```

结果:$4^2+5^2=29$H

在程序存储器的一片存储单元中建立起该变量的平方表。用数据指针 DPTR 指向平方表的首址,则变量与数据指针之和的地址单元中的内容就是变量的平方值。

采用 MOVC A,@A+PC 指令实现查表功能时,无须 DPTR 参与(系统资源占用少),但表格只能存放在 MOVC 指令后的 256 B 内,即表格存放地点和空间有一定的限制。

3.6 本章小结

本章主要介绍 51 单片机的汇编指令和汇编语言程序设计,重点了解并熟悉汇编语言的基本知识;熟悉各指令的功能、寻址方式、寻址范围、对标志位的影响以及指令执行的机器周期;掌握伪指令和单片机指令的使用,并理解二者的区别;掌握基本程序结构和典型程序设计。

习题

一、简答题

1. 在 MCS-51 单片机中，寻址方式有几种？其中对片内 RAM 可以用哪几种寻址方式？对片外 RAM 可以用哪几种寻址方式？访问特殊功能寄存器区可以用哪几种寻址方式？访问片外部程序存储器可以采用哪些寻址方式？

2. 在对片外 RAM 单元寻址中，用 Ri 间接寻址与用 DPTR 间接寻址有什么区别。

3. 在位处理中，位地址的表示方式有哪几种？

4. 什么是伪指令？常用的伪指令功能如何？

5. 子程序调用时，参数的传递方法有哪几种？

6. 区分下列指令有什么不同？

```
(1)  MOV    A,30H     和    MOV    A,#30H
(2)  MOV    A,@R0     和    MOVX   A,@R0
(3)  MOV    A,R0      和    MOV    A,@R0
(4)  MOVX   A,@R0     和    MOVX   A,@DPTR
(5)  MOVX   A,@DPTR   和    MOVC   @A+DPTR
```

二、阅读程序写结果

1. 若 R1＝30H，A＝40H，(30H)＝60H，(40H)＝08H。试分析执行下列程序段后上述各单元内容的变化。

```
MOV  A,@R1
MOV  @R1,40H
MOV  40H,A
MOV  R1,#7FH
```

2. 若(50H)＝40H，试写出执行以下程序段后累加器 A、寄存器 R0 及内部 RAM 的 50H、41H、42H 单元中的内容各为多少？

```
MOV  A,50H
MOV  R0,A
MOV  A,#00H
MOV  @R0,A
MOV  A,#3BH
MOV  41H,A
MOV  42H,41H
```

3. 设(70H)＝60H，(60H)＝20H。P1 口为输入口，当输入状态为 B7H 时，执行下列程序，试分析(70H)、B、(R1)、(R0)的内容是什么？

```
MOV  R0,#70H
MOV  A,@R0
MOV  R1,A
MOV  B,@R1
```

```
MOV  P1,#0FFH
MOV  @R0,P1
```

4. 若 A=E8H,R0=40H,R1=20H,R4=3AH,(40H)=2CH,(20H)=0FH,试写出下列各指令独立执行后有关寄存器和存储单元的内容。若该指令影响标志位,试指出 CY、AC 和 OV 的值。

```
MOV  A,@R0
ANL  40H,#0FH
ADD  A,R4
SWAP A
DEC  @R1
XCHD A,@R1
```

5. 设 A=83H,R0=17H,(17H)=34H,分析当执行完下面指令段后累加器 A、R0、17H 单元中的内容。

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

6. 下列程序段汇编后,从 1000H 单元开始的单元内容是什么?

```
     ORG   1000H
TAB: DB    12H,34H
     DS    3
     DW    5567H,87H
```

三、程序设计

1.完成某种操作可以采用几条指令构成的指令序列实现,试写出完成以下每种操作的指令序列。

(1) R0 的内容送到 R1 中。

(2) 内部 RAM 60H 单元的内容传送到寄存器 R2。

(3) 外部 RAM 1000H 单元的内容传送到寄存器 R2。

(4) 片内 RAM 20H 单元的内容送到片内 RAM 的 40H 单元中。

(5) 片内 RAM 20H 单元的内容送到片外 RAM 的 60H 单元中。

(6) 片内 RAM 50H 单元的内容送到片外 RAM 的 1000H 单元中。

(7) 片外 RAM 1000H 单元的内容送到片外 RAM 的 20H 单元中。

(8) 片外 RAM 1000H 单元的内容送到片外 RAM 的 4000H 单元中。

(9) ROM 1000H 单元的内容送到片内 RAM 的 50H 单元中。

(10) ROM 1000H 单元的内容送到片外 RAM 的 1000H 单元中。

2. 写出完成下列要求的指令,要求不得改变未涉及的位的内容。

(1) 使累加器 A 第 0 位置位。

(2) 累加器 A 的高 2 位置 1,其余位不变。

(3) 清除累加器 A 的高 4 位。

(4) 清除 ACC.3,ACC.4,ACC.5,ACC.6。

(5) 累加器 A 第 0 位、2 位、4 位、6 位取反。

3. 试编写一段程序,将片内 RAM 的 1000～101FH 单元的内容依次存入片外 RAM 的 20H～3FH 单元中。

4. 设被加数存放在内部 RAM 的 20H、21H 单元,加数存放在 22H、23H 单元,若要求和存放在 24H、25H 中,试编写出 16 位无符号数相加的程序(采用大端模式存储)。

5. 试编写程序,将内部 RAM 的 40H、41H 单元的两个无符号数相乘,结果存放在 R2、R3 中,R2 中存放高 8 位,R3 中存放低 8 位。

6. 试编写程序,将 R1 中的低 4 位数与 R2 中的高 4 位数合并成一个 8 位数,并将其存放在 R1 中。

7. 用查表的方法实现一位十六进制数转换成 ASCII 码。

8. 编写程序,求内部 RAM 中 40H～49H 10 个单元内容的平均值,并存放在 6AH 单元中。