第 3 章

CHAPTER 3

MCS-51 指令系统及汇编程序设计

本章讨论 MCS-51 单片机的指令系统及汇编语言程序设计。内容主要有寻址方式、分类指令、伪指令和汇编语言程序设计基础。

通过学习指令系统和汇编语言，能够更深刻理解计算机的工作原理。本章是单片机程序设计的基础，虽然现在多以 C 语言编程为主，但对某些要求较高的部分，还是需要用汇编语言来写；另外在使用 Keil C 调试、分析程序时，经常需要阅读反汇编窗口程序。

3.1 汇编语言概述

3.1.1 指令和机器语言

指令是计算机中 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。指令系统的强弱，决定了计算机智能的高低。MCS-51 单片机指令系统功能很强，有乘、除法指令、丰富的条件跳转指令、位操作指令等，并且使用方便、灵活。

要使计算机按照人们的要求完成一项工作，就必须让 CPU 按顺序执行预设的操作，即逐条执行人们编写的指令。这种按照人们要求所编排的指令操作的序列，称为程序。编写程序的过程叫程序设计。

程序设计语言就是编写程序的一整套规则和方法，是实现人机交换信息的基本工具，分为机器语言、汇编语言和高级语言。

机器语言用二进制编码表示每条指令，是计算机能够直接识别和执行的语言。用机器语言编写的程序，称为机器语言程序或机器码程序。因为机器只能够识别和执行这种机器码程序，所有语言程序最终都需要翻译成机器码程序，所以机器码程序又称为目标程序。

MCS-51 单片机是 8 位机，其机器语言以 8 位二进制码为单位(字节)，有单字节、双字节和 3 字节指令。

例如，要做“13＋25”的加法，在 MCS-51 单片机中机器码程序为：

```
01110100        00001101        (把 13 放到累加器 A 中)
00100100        00011001        (A 加 25,结果仍放回 A 中)
```

为了便于书写和记忆,可采用十六进制表示指令码。上面这两条指令可写为:

```
74H  0DH
24H  19H
```

显然,用机器语言编写程序不易理解、不易查错、不易修改、不易记忆。

3.1.2 汇编语言

直接用机器语言编写程序非常困难,为了克服机器语言编程中的问题,人们发明了用符号代替机器码的编程方法。这种符号就是助记符,一般采用相关的英文单词或其缩写来表示。这就是汇编语言。

汇编语言是用助记符、符号、数字等来表示指令的程序语言,相对于机器语言来说,汇编语言容易理解和记忆。它与机器语言是一一对应的。汇编语言不像高级语言(如 C 语言)那样具有通用性,而是属于某种 CPU 所独有的,与 CPU 内部硬件结构密切相关。用汇编语言编写的程序叫汇编语言程序。

例如,上面的"13+25"的例子可写成:

```
汇编语言程序              机器语言代码
MOV  A,#0DH               74H  0DH
ADD  A,#19H               24H  19H
```

汇编语言和机器语言都属于低级语言。尽管汇编语言相对机器语言有不少优点,但它仍然存在着机器语言的某些缺点,如与 CPU 的硬件结构紧密相关,不同的 CPU 其汇编语言不同。这使得汇编语言不能够移植,使用不便;其次,要用汇编语言进行程序设计,必须了解所使用的 CPU 的硬件结构与性能,对程序设计人员有较高的要求。所以又出现了对 MCS-51 单片机编程的高级语言,如 PL/M、BASIC、C 语言等,现在 PL/M、BASIC 等语言已经被淘汰,主要使用 C 语言。

3.1.3 汇编语言格式

MCS-51 汇编语言指令由四部分组成,其一般格式如下:

```
[标号:]    操作码    [操作数]    [;注释]
```

格式中的方括号表示可以没有相应部分,可见,可以没有标号、操作数和注释,但至少要有操作码。其操作数部分最多可以是三项:

```
[操作数1]    [,操作数2]    [,操作数3]
```

操作数 1 常称为目的操作数,操作数 2 称为源操作数,操作数 3 多为跳转的目标。例如:

```
START: MOV    A,#23H    ;23H→A
```

START 为标号,MOV 为操作码,A、#23H 为操作数,23H→A 为注释。

标号是相应指令的标记,便于查找,用于程序入口、循环等。

操作码规定了指令所要执行的操作,由 2～5 个英文字母表示。例如,MOV、ADD、RRC、JZ、DJNZ、CJNE、LCALL 等。

操作数指出了参与操作的数据来源、操作结果存放的地方，以及跳转的目标位置。操作数可以是一个数(立即数)，也可以是数据所在的空间地址，即在执行指令时从指定的空间地址读取或写入数据。

注释主要使程序容易阅读。

操作码和操作数都有对应的二进制代码，指令代码由若干字节组成。对于不同的指令，指令的字节数不同。在MCS-51指令系统中，有单字节指令、双字节指令和3字节指令。下面分别加以说明。

1. 单字节指令

单字节指令中的8位二进制代码，既包含操作码的信息，也包含操作数的信息。这种指令有两种情况。

(1) 指令码中隐含着对某一个寄存器的操作。

例如，INC　A、MUL　AB、RL　A、CLR　C、INC　DPTR等指令，都属于这一类，只需要一个字节就可以表示出执行什么操作、操作数是哪个。如数据指针DPTR增1指令INC　DPTR，其8位二进制指令代码为A3H，格式为：

1	0	1	0	0	0	1	1

(2) 由指令码中的r r r或i指定操作数。

例如，ADD　A，Rn、INC　Rn、ANL　A，@Ri、MOV　@Ri，A等指令，都属于这一类。如累加器A向工作寄存器传送数据指令MOV　Rn，A，其指令格式为：

1	1	1	1	1	r	r	r

其中高5位为操作码内容，指出作传送数据操作，低3位的rrr的不同组合编码，用来表示向哪一个寄存器(R0～R7)传送数据，故一字节就够了。

MCS-51单片机共有49条单字节指令。

2. 双字节指令

用一个字节表示操作码，另一个字节表示操作数或操作数所在的地址。其指令格式为：

操作码	立即数或地址

MCS-51单片机共有45条双字节指令。

3. 3字节指令

用一个字节表示操作码，另外两个字节表示操作数或操作数所在的地址。其指令格式为：

操作码	立即数或地址	立即数或地址

MCS-51单片机共有17条3字节指令。

3.2　MCS-51单片机寻址方式

所谓寻址方式，是指CPU寻找参与运算的操作数的方式，或者寻找数据保存位置的方式。寻址方式是汇编语言程序设计中最基本的内容之一，必须要十分熟悉。

MCS-51 单片机有 7 种寻址方式：立即数寻址、寄存器寻址、直接寻址、寄存器间接寻址、变址寻址、位寻址和指令寻址。可以分为两类：操作数寻址和指令寻址，在 7 种寻址方式中，除了指令寻址之外，其余 6 种都属于操作数寻址。

3.2.1 立即数寻址

立即数寻址也叫立即寻址、常数寻址。其操作数就在指令中，是指令的一部分，紧跟在操作码后面，用“#”符号作前缀，以区别地址。访问的是 code 区域。例如：

```
MOV     A,#2CH          ; 2CH→A
MOV     A,2CH           ; (2CH)→A
```

前者表示把 2CH 这个数送给累加器 A，后者表示把片内 RAM 中地址为 2CH 单元的内容送给累加器 A。

立即数也可以是 16 位的，如：

```
MOV     DPTR,#1234H
MOV     TL2,#2345H
MOV     RCAP2L,#3456H
```

对于第 2 条指令，立即数的低 8 位送给了 TL2，高 8 位送给了 TH2；对于第 3 条指令，立即数的低 8 位送给了 RCAP2L，高 8 位送给了 RCAP2H。

3.2.2 寄存器寻址

寄存器寻址就是由指令指出寄存器组 R0～R7 中某一个或寄存器 A、B、DPTR 和 C(位处理器的累加器)的内容作为操作数。例如：

```
MOV     A,R7            ; (R7)→A
MOV     36H,A           ; (A)→36H
ADD     A,R0            ; (A)+(R0)→A
```

指令中给出的操作数是一个寄存器名，在此寄存器中存放着真正被操作的对象。工作寄存器的识别由操作码的低 3 位完成。其对应关系如表 3-1 所示。

表 3-1 低 3 位操作码与寄存器 Rn 的对应关系

低 3 位 r r r	000	001	010	011	100	101	110	111
寄存器 Rn	R0	R1	R2	R3	R4	R5	R6	R7

例如，INC Rn 的机器码格式为 00010rrr。若 rrr＝010B，则 Rn＝R2，即

```
INC     R2              ; (R2)+1→R2
```

对于工作寄存器组的操作必须注意，要考虑 PSW 中 RS1、RS0 的值，即要确定当前使用的是哪一组寄存器，然后对其值进行操作。设(R2)＝23H，使用第 2 组(RS1 RS0＝10B)寄存器，则该指令的执行过程如图 3-1 所示。

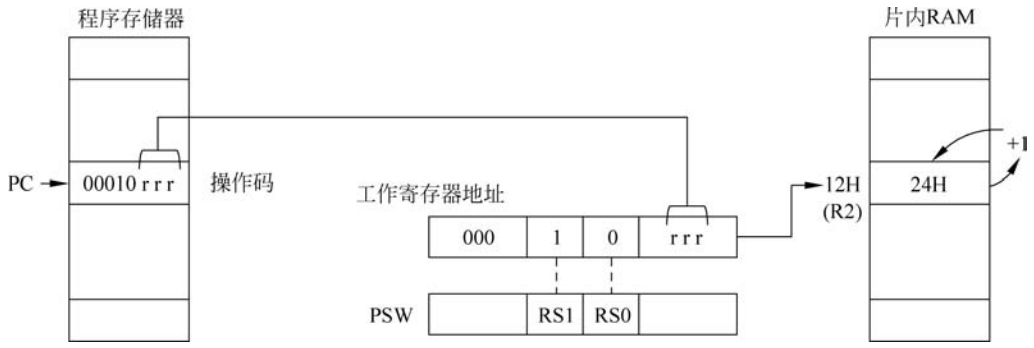


图 3-1　寄存器寻址示意图

3.2.3　直接寻址

直接寻址是指操作数存放在片内 RAM 中，指令中给出 RAM 中的地址。例如：

```
MOV     A,38H           ; (38H)→A
```

即片内 RAM 中 38H 单元的内容送入累加器 A。

设(38H)＝6DH，该指令的执行过程如图 3-2 所示。

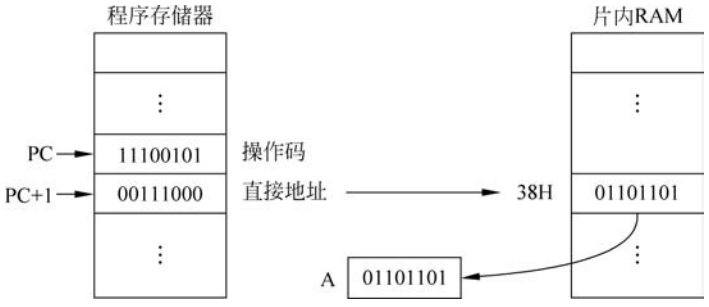


图 3-2　直接寻址示意图

在 MCS-51 单片机中，直接寻址方式可以访问片内 RAM 的低 128 字节(data 区域)和所有的特殊功能寄存器(sfr 区域)，而不能够直接寻址访问片内 RAM 的高 128 字节，高 128 字节只能够间接访问。

对于特殊功能寄存器，既可以使用地址，也可以使用 SFR 名。例如：

```
MOV     A,P1            ; (P1)→A
```

是把 SFR 中 P1 口引脚的数据送给累加器 A，也可以写成：

```
MOV     A,90H
```

其中，90H 是 P1 口的地址。

直接寻址的地址占一字节，所以，一条直接寻址方式的指令至少占用内存两个单元。

3.2.4　寄存器间接寻址

寄存器间接寻址是指操作数存放在片内或片外 RAM 中，操作数的地址存放在寄存器

中,在指令执行时,通过指令中的寄存器内的地址,间接地访问操作数。存放地址的寄存器称为间址寄存器,指令中在寄存器前面加前缀“@”表示。

MCS-51 单片机规定只使用 Ri(i=0、1,即指 R0、R1)、SP 和 DPTR 作间址寄存器。寄存器间接寻址的空间和范围有以下几种情况。

1. 使用 Ri、SP 间接访问片内 RAM 空间

这种情况间接访问的范围是片内 RAM 的 256 字节(idata 区域),包括低 128 字节和高 128 字节,但不包括特殊功能寄存器。例如:

```
MOV     A,@Ri           ; ((Ri))→A
ADD     A,@Ri           ; (A)+((Ri))→A
```

上面(Ri)表示 Ri 指向的单元,即单元的地址,((Ri))表示 Ri 指向单元中的数据。其操作如图 3-3 所示。

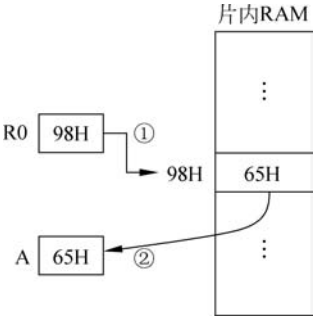


图 3-3 间接寻址(MOV A,@R0)示意图

对使用 SP 间接访问片内 RAM,仅用在堆栈操作中,见后面指令系统。

2. 使用 Ri 间接访问片外 RAM 空间

这种情况间接访问的范围是片外 RAM 的 64KB 全空间。其指令只有两条:

```
MOVX    A,@Ri           ; ((P2)(Ri))→A
MOVX    @Ri,A           ; (A)→(P2 Ri)
```

P2 中的值作为高 8 位地址,Ri 中的值作为低 8 位地址。P2 为 0 时,访问的区域为 pdata。

3. 使用 DPTR 间接访问片外 RAM 空间

这种情况间接访问的范围是片外 RAM 的 64KB 全空间(xdata 区域)。其指令也只有两条:

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

DPTR 为 16 位地址。

3.2.5 变址寻址

变址寻址实际上是基址加变址的间接寻址,就是操作数的地址由基址寄存器的地址,加上变址寄存器的地址得到。

基址寄存器使用 DPTR 或程序计数器(PC),累加器 A 则为变址寄存器。因为变址寻址也是间接寻址,因此在地址寄存器前面要加上前缀“@”。例如:

```
MOVC    A,@A+DPTR       ; ((A)+(DPTR))→A
```

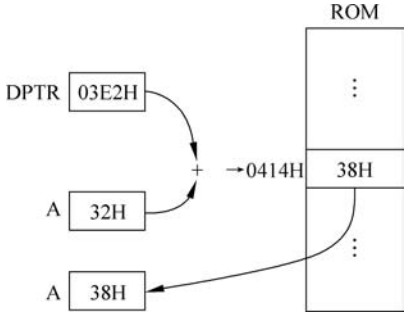


图 3-4 变址寻址示意图

该指令的操作过程如图 3-4 所示。

变址寻址的空间为程序存储器。其范围为:若使用 DPTR 为基址寄存器,寻址范围为

64KB；若使用PC为基址寄存器，寻址空间在PC之后256字节范围内。变址寻址访问的为code区域。变址寻址主要用于查表操作。

3.2.6　位寻址

所谓位寻址，是指操作数是二进制位的地址。指令中给出的是操作数的位地址，位地址可以是片内RAM中20H～2FH(bdata区域)中的某一位，也可以是特殊功能寄存器(sfr区域)中能够按位寻址的某一位。位地址在指令中用bit表示。例如：

```
SETB    bit
MOV     C,bit
```

在MCS-51单片机中，位地址可以用以下4种方式表示：

(1) 直接位地址(00H～FFH)。如32H。

(2) 字节地址带位号。如20H.1，表示20H单元的第1位。

(3) 特殊功能寄存器名带位号。如P1.7，表示P1口的第7位。

(4) 位符号地址。可以是特殊功能寄存器位名，也可以是用位地址符号命令BIT定义的位符号(如，flag　BIT　01H)。如TR0、flag，TR0表示定时器/计数器0的运行控制位，flag表示01H位。

3.2.7　指令寻址

指令寻址使用于控制转移指令中，其操作数给出转移的目标位置的地址，访问的是code区域。在MCS-51指令系统中，目标位置的地址的提供有两种方式，分别对应两种寻址方式。

1. 绝对寻址

绝对寻址是在指令的操作数中，直接提供目标位置的地址或地址的一部分。在MCS-51指令系统中，长转移和长调用指令给出的是16位地址，寻址范围为64KB全空间。例如：

```
LJMP    SER_INT_T1          ; 无条件跳转到T1中断服务程序SER_INT_T1处
LCALL   SUB_SORT            ; 调用排序子程序SUB_SORT
```

2. 相对寻址

相对寻址是以当前程序计数器(PC)值(为所执行指令的下一条指令的地址)为基地址，加上指令中给出的偏移量rel，得到目标位置的地址，即：

```
目标地址 = PC + rel
rel = 目标地址 - PC
```

偏移量rel为8位补码，其值为－128～＋127。rel<0表明目标地址小、源地址大，程序向回跳转；rel>0，程序向前跳转。例如：

```
JZ      FIRST           ; (A) = 0,跳转到FIRST
DJNZ    R7,LOOP         ; (R7) - 1≠0,跳转到LOOP
```

注意：在实际编程中，不需要计算rel，由编译器自动计算(过去手工编译时需要程序员

计算 rel 值)；当跳转范围超出了 rel 范围，编译器会提示，对程序做适当调整即可。

3.2.8 寻址空间及指令中符号注释

1. 各寻址方式的寻址空间

表 3-2 给出了各种寻址方式所使用的操作数、寻址空间及范围。

表 3-2 操作数寻址方式、寻址空间及范围

寻址方式	操作数寻址空间及范围	示例指令
立即数寻址	在程序存储空间，随指令读入	MOV A,#46H
直接寻址	片内 RAM 中；低 128 字节和 SFR	MOV A,46H
寄存器寻址	使用 R0～R7、A、B、C、DPTR	MOV A,R2
寄存器间接寻址	片内 RAM：使用@Ri、SP；范围为 256B，不含 SFR 片外 RAM：使用@Ri、@DPTR；范围为 64KB	MOV A,@R0 MOVX @DPTR,A
变址寻址	使用@A+PC、@A+DPTR；在程序存储器中； 范围分别为 PC 之后 256B 之内和 64KB 全空间	MOVC A,@A+DPTR MOVC A,@A+PC
位寻址	使用位地址；在位寻址空间；RAM 的 20H～2FH 和 SFR	SETB 36H
指令绝对寻址	操作数是目标地址；在程序存储空间； 范围为 64KB 全空间	LJMP SECON
指令相对寻址	操作数是相对地址；在程序存储空间；范围－128～127	SJMP LOOP

2. 指令中常用符号注释

Rn：n＝0～7。当前选中的工作寄存器 R0～R7。它们的具体地址由 PSW 中的 RS1、RS0 确定，可以是 00H～07H(第 0 组)、08H～0FH(第 1 组)、10H～17H(第 2 组)或 18H～1FH(第 3 组)。

Ri：i＝0、1。当前选中的工作寄存器组中可作为地址指针的 R0 和 R1。

＃data：8 位立即数。

＃data16：16 位立即数。

direct：8 位片内 RAM 单元地址，包括低 128B 和 SFR，但不包括 RAM 的高 128B。

addr16：程序存储空间的 16 位目的地址，用于 LCALL 和 LJMP 指令中。目的地址在 64KB 程序存储空间的任意位置。

rel：补码形式的 8 位地址偏移量。以下一条指令的第一个字节为基地址，地址偏移量在－128～＋127。

bit：片内 RAM 或 SFR 中的直接寻址位地址。

@：在间接寻址方式中，间址寄存器的前缀符号。

(×)：表示×中的内容。

((×))：表示由×中指向的地址单元的内容。

∧：逻辑与。

∨：逻辑或。

⊕：逻辑异或。

←、→：指令操作流程，将内容送到箭头指向的地方。

3.3　MCS-51 单片机指令系统

MCS-51 单片机指令系统有 111 条指令，其中单字节指令 49 条，双字节指令 45 条，3 字节指令 17 条。从指令执行的时间来看，单周期指令 64 条，双周期指令 45 条，只有乘、除两条指令执行时间为 4 个周期。

MCS-51 单片机指令系统按其功能分，可以分为 5 大类：

- 数据传送指令(29 条)。
- 算术运算指令(24 条)。
- 逻辑操作指令(24 条)。
- 控制程序转移指令(17 条)。
- 位(布尔)操作指令(17 条)。

虽然有 111 条指令，但由于没有复杂的寻址方式，没有难理解的指令，并且助记符只有 42 种，所以 MCS-51 单片机的指令系统容易理解、容易记忆、容易掌握。

3.3.1　数据传送指令

在通常的应用程序中，数据传送指令往往占有较大的数量，数据传送是否灵活、迅速，对整个程序的编写和执行都有很大的影响。

所谓传送，就是把源地址单元的内容传送到目的地址单元中去，而源地址单元中的内容不变。

数据传送指令共 29 条，是指令中数量最多、使用最频繁的一类指令。这类指令一般不影响程序状态字，只有目的操作数是累加器 A 时，才会影响标志位 P。这类指令可以分为三组：普通传送指令、数据交换指令和堆栈操作指令。

1. 普通传送指令

普通传送指令以助记符 MOV 为基础，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：MOV　目的操作数，源操作数

其中：源操作数可以是 A、Rn、@Ri、direct、#data，目的操作数可以是 A、Rn、@Ri、direct、DPTR。以目的操作数的不同可以分为五组，共 16 条指令。

(1) 以 A 为目的操作数。

汇编指令格式		操作	机器码(H)
MOV	A,Rn	;(Rn)→A	E8～EF
MOV	A,direct	;(direct)→A	E5 direct
MOV	A,@Ri	;((Ri))→A	E6、E7
MOV	A,#data	;data→A	74 data

指令中的 Rn，对应工作寄存器的 R0～R7。Ri 为间接寻址寄存器，i＝0 或 1，即 R0 或 R1。

本组 4 条指令都影响 PSW 中的 P 标志位。

(2) 以 Rn 为目的操作数。

汇编指令格式	操作	机器码(H)
MOV Rn,A	; (A)→Rn	F8～FF
MOV Rn,direct	; (direct)→Rn	A8～AF direct
MOV Rn,#data	; data→Rn	78～7F data

本组指令都不影响 PSW 中的标志位。

(3) 以直接地址 direct 为目的操作数。

汇编指令格式	操作	机器码(H)
MOV direct,A	; (A)→direct	F5 direct
MOV direct,Rn	; (Rn)→direct	88～8F direct
MOV direct2,direct1	; (direct1)→direct2	85 direct1 direct2
MOV direct,@Ri	; ((Ri))→direct	86、87direct
MOV direct,#data	; data→direct	75 direct data

本组指令都不影响 PSW 中的标志位。

(4) 以间接地址@Ri 为目的操作数。

汇编指令格式	操作	机器码(H)
MOV @Ri,A	; (A)→(Ri)	F6、F7
MOV @Ri,direct	; (direct)→(Ri)	A6、A7 direct
MOV @Ri,#data	; data→(Ri)	76、77 data

本组指令都不影响 PSW 中的标志位。

(5) 以 DPTR 为目的操作数。

汇编指令格式	操作	机器码(H)
MOV DPTR,#data16	; dataH→DPH,dataL→DPL	90 data15～8 data7～0

该指令不影响 PSW 中的标志位。后面的指令不再给出机器码,其机器码可以参考附录 A 表的第 3 列。

【例 3-1】 设片内 RAM(30H)=40H,(40H)=10H,(10H)=00H,(DPL)=CAH,分析以下程序执行后各单元及寄存器、P2 口的内容。

```
MOV    R0,#30H;          ; 30H→R0
MOV    A,@R0             ; ((R0))→A
MOV    R1,A              ; (A)→R1
MOV    B,@R1             ; ((R1))→B
MOV    @R1, DPL          ; (DPL)→(R1)
MOV    P2, DPL           ; (DPL)→P2
MOV    10H,#20H          ; 20H→10H
```

执行上述指令后的结果为:(R0)=30H,(R1)=(A)=40H,(B)=10H,(40H)=CAH,(DPL)=(P2)=CAH,(10H)=20H。

2) 片外数据存储器传送指令 MOVX

MCS-51 单片机对片外 RAM 或 I/O 口进行数据传送,采用的是寄存器间接寻址的方法,通过累加器 A 完成。这类指令共有以下 4 条单字节指令。

```
汇编指令格式                    操作
MOVX  A,@Ri                     ; ((P2)(Ri))→A
MOVX  @Ri,A                     ; A→(P2, Ri)
MOVX  A,@DPTR                   ; ((DPTR))→A
MOVX  @DPTR,A                   ; A→(DPTR)
```

这4条指令都是执行总线操作，第1条和第3条指令是执行总线读操作，读控制信号$\overline{RD}$有效；第2条和第4条指令是执行总线写操作，写控制信号$\overline{WR}$有效。

这组指令中第1、3两条指令影响P标志位，其他两条指令不影响任何标志位。

对前两条指令要特别注意：①间址寄存器Ri提供低8位地址，隐含的P2提供高8位地址，在执行操作之前，必须先对P2和Ri分别赋高8位和低8位地址值；②这两条指令的访问范围都是整个片外RAM，64KB全空间。

【例3-2】 设片外RAM空间(0203H)＝6FH，分析执行下面指令后的结果。

```
MOV   DPTR,＃0203H              ; 0203H→DPTR
MOVX  A,@DPTR                   ; ((DPTR))→A
MOV   30H,A                     ; A→30H
MOV   A,＃0FH                   ; 0FH→A
MOVX  @DPTR,A                   ,A; A→(DPTR)
```

执行结果为：(DPTR)＝0203H，(30H)＝6FH，(0203H)＝(A)＝0FH。

3) 程序存储器传送指令MOVC

访问程序存储器的数据传送指令又称为查表指令，经常用于查表。查表指令采用基址加变址的间接寻址方式，把放在程序存储器中的表格数据读出，传送给累加器A。这类指令只有以下两条单字节指令。

```
汇编指令格式                    操作
MOVC  A,@A＋DPTR                ; ((A)＋(DPTR))→A
MOVC  A,@A＋PC                  ; ((A)＋(PC))→A
```

前一条指令采用DPTR作基址寄存器，因此，可以很方便地把一个16位地址送到DPTR，实现在整个64KB程序存储空间任一单元到累加器A的数据传送。称为远程查表指令，即数据表格可以存放到程序存储空间的任何地方。

后一条指令以PC作为基址寄存器，其PC值是下一条指令的地址。另外，累加器A的内容为8位无符号数，所以查表范围限于256字节之内，称为近程查表指令。使用该条指令，关键要准确计算从本指令到数据所在地址的地址偏移量A。但在实际应用中，往往给出的是数据表的首地址和数据在表内的偏移量，因此，需要先计算出表首偏移量，其计算关系为：

表首偏移量＝表首地址－PC

数据地址偏移量A与表首偏移量、表内偏移量的关系为：

数据地址偏移量A＝表首偏移量＋表内偏移量

这组指令都影响P标志位。

【例3-3】 从片外程序存储器2000H单元开始存放0～9的平方值，以PC作为基址寄存器，执行查表指令得到6的平方值，并且送到片内RAM中的30H单元。

解：设MOVC指令所在的地址为1FA0H，则表首偏移量＝2000H－(1FA0H＋1)＝

5FH,表内偏移量为6。

相应的程序为:

```
MOV    A,#5FH
ADD    A,#06H
MOVC   A,@A+PC
MOV    30H,A
```

执行结果为:(PC)=1FA1H,(A)=(30H)=24H=36D。

如果使用以DPTR为基址寄存器的查表指令,其程序如下:

```
MOV    DPTR,#2000H
MOV    A,#6
MOVC   A,@A+DPTR
MOV    30H,A
```

通过本例对两条查表指令进行比较可以看出,以DPTR为基址寄存器的查表指令使用简单、方便。

2. 数据交换指令

普通数据传送指令完成的是把源操作数传送给目的操作数,指令执行后源操作数不变,数据传送是单向的。而数据交换指令则对数据作双向传送,传送后,前一个操作数传送到了后一个操作数所保存的地方,后一个操作数传送到了前一个操作数所保存的地方。

数据交换指令要求第一个操作数必须为累加器A。共5条指令,分为字节交换和半字节交换。

1) 字节交换指令

```
汇编指令格式            操作
XCH    A,Rn             ; (A)←→(Rn)
XCH    A,direct         ; (A)←→(direct)
XCH    A,@Ri            ; (A)←→((Ri))
```

这3条指令都影响P标志位。

2) 低半字节交换指令

```
汇编指令格式            操作
XCHD   A,@Ri            ; (A0~3)←→((Ri)0~3)
```

这条指令影响P标志位。

3) A自身半字节交换指令

```
汇编指令格式            操作
SWAP   A                ; (A0~3)←→(A4~7)
```

这条指令不影响任何标志位。

【例3-4】 设(R0)=30H,(30H)=4AH,(A)=28H,则分别执行XCH A,@R0、XCHD A,@R0、SWAP A后各单元的内容。

解:

```
执行: XCH    A,@R0          ; 结果为(A) = 4AH,(30H) = 28H
```

```
执行：XCHD   A,@R0          ；结果为(A)=48H,(30H)=2AH
执行：SWAP   A              ；结果为(A)=84H,(30H)=2AH
```

R0 中的内容一直未变，(R0)＝30H。

3. 堆栈操作指令

堆栈操作有进栈和出栈，常用于保存和恢复现场。堆栈操作指令有两条。

```
汇编指令格式                操作
PUSH    direct              ;先(SP)+ 1→SP,
                            ; 后(direct)→(SP)
POP     direct              ; 先((SP))→direct,
                            ; 后(SP)－1→SP
```

PUSH 为进栈操作。进栈时，堆栈指针 SP 先加 1，指向栈顶的一个空单元，然后将直接地址(direct)单元的内容压入 SP 所指向的空栈顶中。本指令不影响任何标志位。

POP 为出栈操作。出栈时，先将栈顶的内容弹出送给直接地址 direct 单元，然后堆栈指针 SP 减 1，使 SP 指向堆栈中有效的数据。本指令有可能影响 P 标志位，当操作数是累加器 A 时。

【例 3-5】 若在程序存储器中 2000H 单元开始的区域依次存放着 0～9 的平方值，用查表指令读取 3 的平方值并存于片内 RAM 中 30H 单元，要求操作后保持 DPTR 中原来的内容不变。

解：为了使用 DPTR，并且保持原来的内容不变，应该在使用 DPTR 前使其进栈，使用后再出栈恢复其原来内容。程序如下：

```
PUSH    DPH
PUSH    DPL
MOV     DPTR,＃2000H
MOV     A,＃3
MOVC    A,@A＋DPTR
MOV     30H,A
POP     DPL
POP     DPH
```

注意：

(1) 进栈与出栈必须成对使用，否则会出现意想不到的问题，如在子程序中操作，会使子程序不能够正确返回；

(2) 先进栈的必须后出栈，后进栈的必须先出栈，否则会出现 DPL 与 DPH 内容互换。

3.3.2　算术运算指令

算术运算类指令共有 24 条，包括加法、减法、乘法、除法、BCD 码调整等指令。MCS-51 单片机的算术/逻辑运算部件只能执行无符号二进制整数运算，可以借助于溢出标志位，实现有符号数的补码运算。借助于进位标志，可以实现高精度加、减运算。

算术运算结果会影响进位标志 CY、半进位标志 AC、溢出标志 OV，但加 1 和减 1 指令不影响这些标志位。如果累加器 A 为目的操作数，还要影响奇偶标志位 P。

算术运算指令多数以累加器 A 作为第一操作数，第二操作数可以是工作寄存器 Rn、直

接地址数据、间接地址数据和立即数。为了便于讨论，按运算将其分为5组。

1. 加法指令

加法指令分为不带进位加法指令、带进位加法指令和加1指令。

1) 不带进位加法指令 ADD

```
汇编指令格式          操作
ADD     A,Rn          ; (A) + (Rn)→A
ADD     A,direct      ; (A) + (direct)→A
ADD     A,@Ri         ; (A) + ((Ri))→A
ADD     A,#data       ; (A) + data→A
```

这组指令的执行影响标志位CY、AC、OV和P，溢出标志OV只对有符号运算有意义。

【例3-6】 设(A)=0C3H，(R0)=0AAH，试分析执行ADD A,R0后的结果及各标志位的值。

执行结果为：(A)=6DH。

各标志位为：CY=1，AC=0，P=1，OV=1。

(A)：	1100	0011
+(R0)：	1010	1010
1	0110	1101

溢出标志OV为第7位与第6位的进位C7、C6的异或，即OV=C7⊕C6。

2) 带进位加法指令 ADDC

```
汇编指令格式          操作
ADDC    A,Rn          ; (A) + (Rn) + CY→A
ADDC    A,direct      ; (A) + (direct) + CY→A
ADDC    A,@Ri         ; (A) + ((Ri)) + CY→A
ADDC    A,#data       ; (A) + data + CY→A
```

这组指令的执行影响标志位CY、AC、OV和P，溢出标志OV只对有符号运算有意义。

【例3-7】 试编写程序，把R1R2和R3R4中的两个16位数相加，结果存放在R5R6中。

解：对于相加的两个数的低8位R2和R4使用不带进位的加法指令ADD，其和存放于R6中，对于高8位的R1和R3，使用带进位的加法指令ADDC，其和存放于R5中。程序段如下：

```
MOV     A,R2          ; (R2)→A
ADD     A,R4          ; (A) + (R4)→A
MOV     R6,A          ; (A)→R6
MOV     A,R1          ; (R1)→A
ADDC    A,R3          ; (A) + (R3) + CY→A
MOV     R5,A          ; (A)→R5
```

3) 加1指令 INC

```
汇编指令格式          操作
INC     A             ; (A) + 1→A
INC     Rn            ; (Rn) + 1→Rn
INC     direct        ; (direct) + 1→direct
INC     @Ri           ; ((Ri)) + 1→(Ri)
INC     DPTR          ; (DPTR) + 1→DPTR
```

这组指令除了第一条影响标志位 P 之外，其他指令都不影响标志位。

2. 减法指令

减法指令分为带借位减法指令和减 1 指令。

1）带借位减法指令 SUBB

```
汇编指令格式              操作
SUBB    A,Rn              ; (A)-(Rn)-CY→A
SUBB    A,direct          ; (A)-(direct)-CY→A
SUBB    A,@Ri             ; (A)-((Ri))-CY→A
SUBB    A,#data           ; (A)-data -CY→A
```

这组指令影响标志位 CY、AC、OV 和 P，溢出标志 OV 只对有符号数运算有意义。

由于 MCS-51 单片机没有不带借位的减法指令，对于不带借位的减法运算，可以先对 CY 清 0(用 CLR　C)，然后再用 SUBB 命令操作。

【例 3-8】 试编写实现 R2-R1→R3 功能的程序。

解：程序段如下：

```
MOV     A,R2
CLR     C
SUBB    A,R1
MOV     R3,A
```

2）减 1 指令 DEC

```
汇编指令格式          操作
DEC  A                ; (A)-1→A
DEC  Rn               ; (Rn)-1→Rn
DEC  direct           ; (direct)-1→direct
DEC  @Ri              ; ((Ri))-1→(Ri)
```

这组指令除了第一条影响标志位 P 之外，其他指令都不影响标志位。

3. 乘法指令 MUL

在 MCS-51 单片机中，乘法指令只有一条。

```
汇编指令格式              操作
MUL  AB                   ; (A)×(B)→B(高字节)、A(低字节)
```

该指令的操作是：把累加器 A 和寄存器 B 中两个 8 位无符号数相乘，所得的 16 位积的高字节存放在 B 中，低字节存放在 A 中。若乘积大于 0FFH，OV 置 1，说明高字节 B 中不为 0，否则 OV 清 0，即 B 中为 0。该指令还影响 P 标志位，并且对 CY 总是清 0。

4. 除法指令 DIV

在 MCS-51 单片机中，除法指令也只有一条。

```
汇编指令格式              操作
DIV  AB                   ; (A)/(B),商→A、余→B
```

该指令的操作是：累加器 A 的内容除以寄存器 B 的内容，两个都是 8 位无符号整数，所得结果的整数商存放在 A 中，余数存放在 B 中。如果除数(B)＝0，则标志位 OV 置 1，否则清 0。该指令还影响 P 标志位，并且 CY 总是被清 0。

5. 十进制调整指令 DA

在 MCS-51 单片机中,十进制调整指令只有一条。

```
汇编指令格式                    操作
DA   A                          ; 调整累加器 A 内容为 BCD 码
```

该指令用于 ADD 或 ADDC 指令后,且只能用于压缩的 BCD 码相加结果的调整,目的是使单片机能够实现十进制加法运算功能。

调整过程如下:

(1) 若累加器 A 的低 4 位为十六进制的 A～F,或者半进位标志位 AC 为 1,则累加器 A 的内容作加 06H 调整。

(2) 若累加器 A 的高 4 位为十六进制的 A～F,或者进位标志位 CY 为 1,则累加器 A 的内容作加 60H 调整。

该指令影响标志位 CY、AC 和 P,但不影响 OV。

【例 3-9】 试编写程序,对两个十进制数 76、58 相加,并且保持其结果为十进制数,把结果存于 R3 中。

解:程序段如下:

```
MOV     A,#76H
ADD     A,#58H
DA      A
MOV     R3,A
```

程序执行后,R3 中的内容为 34H(十进制数 34),进位标志 CY 为 1,则最后结果为 134。

在编写程序时,对 BCD 码的写法必须注意:要按十进制数格式写,然后在其后面加上 H。

3.3.3 逻辑操作指令

逻辑操作指令共有 24 条,包括与、或、异或、清 0、求反、移位等操作指令。

参与逻辑操作的操作数可以是累加器 A、工作寄存器 Rn、直接地址数据、间接地址数据和立即数。

逻辑操作指令对标志位的影响:如果累加器 A 为目的操作数,会影响奇偶标志 P;带进位移位操作,会影响进位标志 CY。

为了便于讨论,将其分为 5 组。

1. 逻辑与指令 ANL

```
汇编指令格式                    操作
ANL   A,Rn                      ; (A)∧(Rn)→A
ANL   A,direct                  ; (A)∧(direct)→A
ANL   A,@Ri                     ; (A)∧((Ri))→A
ANL   A,#data                   ; (A)∧data→A
ANL   direct,A                  ; (direct)∧(A)→direct
ANL   direct,#data              ; (direct)∧data→direct
```

这组指令的前 4 条影响奇偶标志位 P,后 2 条指令不影响任何标志位。

逻辑与操作往往用于使某些位清 0。

2. 逻辑或指令 ORL

汇编指令格式	操作
ORL　A,Rn	;(A)∨(Rn)→A
ORL　A,direct	;(A)∨(direct)→A
ORL　A,@Ri	;(A)∨((Ri))→A
ORL　A,#data	;(A)∨data→A
ORL　direct,A	;(direct)∨(A)→direct
ORL　direct,#data	;(direct)∨data→direct

这组指令的前四条影响奇偶标志位 P,后两条指令不影响任何标志位。

逻辑或操作往往用于使某些位置 1。

3. 逻辑异或指令 XRL

汇编指令格式	操作
XRL　A,Rn	;(A)⊕(Rn)→A
XRL　A,direct	;(A)⊕(direct)→A
XRL　A,@Ri	;(A)⊕((Ri))→A
XRL　A,#data	;(A)⊕data→A
XRL　direct,A	;(direct)⊕(A)→direct
XRL　direct,#data	;(direct)⊕data→direct

这组指令的前四条影响奇偶标志位 P,后两条指令不影响任何标志位。

逻辑异或操作往往用于使某些位取反。

【例 3-10】 写出完成以下各功能的指令:

(1) 对累加器 A 中的 1、3、5 位清 0,其余位不变。

(2) 对 A 中的 2、4、6 位置 1,其余位不变。

(3) 对 A 中的 0、1 位取反,其余位不变。

解:对应指令如下:

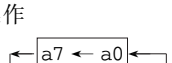
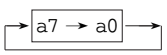
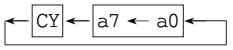
```
ANL　A,#11010101B
ORL　A,#01010100B
XRL　A,#00000011B
```

4. 累加器 A 清 0 和求反指令

汇编指令格式	操作
CLR　A	;0→A
CPL　A	;$\overline{(A)}$→A

前一条指令是对 A 清 0,该指令影响奇偶标志位 P。后一条指令是对 A 求反,不影响任何标志位。

5. 循环移位指令

指令名称	指令格式	操作
A 循环左移	RL　A	; 
A 循环右移	RR　A	; 
A 带进位循环左移	RLC　A	; 

```
A带进位循环右移          RRC  A                    ; →[CY]→[a7 → a0]→
```

前两条指令不影响任何标志位，后两条指令影响进位位 CY 和奇偶标志位 P。

注意：①这 4 条指令，每执行一次只移动 1 位；②左移一次相当于乘以 2，右移一次相当于除以 2。常用移位的方式进行乘除运算，因为移位指令比乘除指令速度快。

【例 3-11】 试编写程序，对 8 位二进制数 01100101B＝65H＝101D 乘以 2。

解：程序段如下：

```
MOV  A,＃65H
CLR  C
RLC  A
```

程序执行后的结果为：(A)＝CAH，CY＝0。CAH＝202D＝101D×2。

3.3.4 控制程序转移指令

计算机功能的强弱，主要取决于转移类指令的多少与功能，特别是条件转移指令。MCS-51 单片机有 17 条转移类指令，包括无条件转移指令、条件转移指令、子程序调用及返回指令等。

这类指令只有比较转移指令影响进位标志 CY，其他指令不影响标志位。

为了便于讨论，将其分为 4 组。

1. 无条件转移指令

无条件转移指令是指，当程序执行该指令后，程序无条件地转移到指令所指定的地址去执行。无条件转移指令包括短转移、长转移和间接转移 3 条。

1) 短转移指令(相对转移指令)SJMP

```
汇编指令格式              操作
SJMP  rel                 ; (PC) + rel→PC
```

指令的实际编写形式为："SJMP 目标地址标号"。

指令的操作数是相对地址，rel 是一个有符号字节数，其范围为－128～127，负数表示向回跳转，正数表示向前跳转。在使用时并不需要计算和写出 rel 值，看下面例子。

【例 3-12】 程序中有一无条件转移指令 SJMP　RELOAD，已知本指令的地址为 0100H，标号 RELOAD 的地址为 0123H，试计算相对地址偏移量 rel。

```
rel = 0123H-(PC) = 0123H-(0100H + 2) = 21H
```

对于 rel 值，在手工汇编时需要计算，并且要把 rel 值写到该指令码的第 2 字节，指令码为 8021H。现在都是计算机进行汇编，并不需要计算 rel 值，所以该指令的编写形式为："SJMP　目标地址标号"。对于后面所有的转移指令，由于使用计算机汇编，其编写形式均是如此。

2) 长转移指令 LJMP

```
汇编指令格式              操作
LJMP  addr16              ; addr16→PC
```

指令的实际编写形式为："LJMP 目标地址标号"。

指令提供16位目标地址，执行时，直接将16位地址送给程序计数器(PC)，程序无条件跳转到指定的目标地址去执行。

由于程序的目标地址是16位，因此程序可以跳转到64KB程序存储器空间的任何地方。

3）间接转移指令JMP

汇编指令格式	操作
JMP　@A+DPTR	;(A)+(DPTR)→PC

该指令转移的目标地址是由数据指针寄存器DPTR的内容与累加器A的内容相加得到，DPTR的内容一般为基址，A的内容为相对偏移，在64KB范围内无条件转移。DPTR一般为确定的值，累加器A为变值，根据A的值转移到不同的地方，因此该指令也叫作散转指令。在使用中，往往与一个转移指令表一起实现多分支转移。

【例3-13】 分析下面多分支转移程序段，程序中，根据累加器A的值0、1、2、3转移到相应的TAB0～TAB3分支去执行。

```
    MOV    B,#3
    MUL    AB
    MOV    DPTR,#TABLE              ;转移表首地址送DPTR
    JMP    @A+DPTR                  ;根据A值转移
TABLE:
    LJMP   TAB0                     ;初始(A)=0时转到TAB0执行
    LJMP   TAB1                     ;初始(A)=1时转到TAB1执行
    LJMP   TAB2                     ;初始(A)=2时转到TAB2执行
    LJMP   TAB3                     ;初始(A)=3时转到TAB3执行
    …
```

2. 条件转移指令

条件转移指令是指，当指令中条件满足时，程序转到指定位置执行，条件不满足时，程序顺序执行。这类指令都属于相对转移，转移范围均为－128～127，负数表示向回跳转，正数表示向前跳转。

在MCS-51系统中，条件转移指令有三种：累加器A判0转移指令、比较转移指令、循环转移指令，共8条。需要注意的是，注释中的PC值，均为指向下一条指令的地址值。

1）判0转移指令

指令名称	指令格式	操作
判A为0转移	JZ　rel	;(A)=0,(PC)+rel→PC; ;(A)≠0,顺序执行
判A非0转移	JNZ　rel	;(A)≠0,(PC)+rel→PC; ;(A)=0,顺序执行

指令的实际编写形式分别为："JZ　目标地址标号"和"JNZ　目标地址标号"。

【例3-14】 试编写程序，把片外RAM地址从2000H开始的数据，传送到片内RAM地址从30H开始的单元，直到出现0为止。

解：程序段如下：

```
    MOV    DPTR,#2000H              ;用DPTR指向片外RAM的2000H单元
```

```
    MOV     R0,#30H                ;用R0指向片内RAM的30H单元
LOOP:
    MOVX    A,@DPTR                ;从片外RAM中DPTR指向的单元读数据给A
    MOV     @R0,A                  ;把A中数据存于用R0指向的片内RAM的单元
    INC     R0                     ;片内RAM指针R0增1
    INC     DPTR                   ;片外RAM指针DPTR增1
    JNZ     LOOP                   ;(A)≠0跳转到LOOP去执行;否则顺序向下执行
    SJMP    $                      ;程序无休止地执行本指令,停留到此
```

2）比较转移指令 CJNE

比较转移指令功能较强，共有 4 条指令，它的一般格式为：

```
CJNE    操作数1,操作数2,rel     ;3字节指令
```

指令的功能是，两个操作数进行比较(操作数 1 减操作数 2，置标志位，不保存结果)，若不等则转移，否则顺序执行。该类指令影响进位标志位 CY，而不改变两个操作数。

具体指令形式如下：

```
汇编指令格式                    操作
CJNE  A,direct,rel              ;若(A)>(direct),则(PC)+rel→PC,0→CY
                                ;若(A)>(direct),则(PC)+rel→PC,1→CY
                                ;若(A)=(direct),则顺序执行,0→CY
CJNE  A,#data,rel               ;若(A)>data,则(PC)+rel→PC,0→CY
                                ;若(A)>data,则(PC)+rel→PC,1→CY
                                ;若(A)=data,则顺序执行,0→CY
CJNE  Rn,#data,rel              ;若(Rn)>data,则(PC)+rel→PC,0→CY
                                ;若(Rn)>data,则(PC)+rel→PC,1→CY
                                ;若(Rn)=data,则顺序执行,0→CY
CJNE  @Ri,#data,rel             ;若((Ri))>data,则(PC)+rel→PC, 0→CY
                                ;若((Ri))>data,则(PC)+rel→PC,1→CY
                                ;若((Ri))=data,则顺序执行,0→CY
```

指令的实际编写形式分别为：

```
CJNE  A,direct,目标地址标号
CJNE  A,#data,目标地址标号
CJNE  Rn,#data,目标地址标号
CJNE  @Ri,#data,目标地址标号
```

3）循环转移指令 DJNZ

循环转移指令同样功能很强，共有两条指令。

```
汇编指令格式                    操作
DJNZ    Rn,rel                  ;(Rn)-1→Rn
                                ;若(Rn)≠0,则(PC)+rel→PC
                                ;若(Rn)=0,则顺序执行
DJNZ    direct,rel              ;(direct)-1→direct
                                ;若(direct)≠0,则(PC)+rel→PC
                                ;若(direct)=0,则顺序执行
```

指令的实际编写形式分别为：

```
DJNZ    Rn,目标地址标号
DJNZ    direct,目标地址标号
```

【例 3-15】 试编写程序，统计片内 RAM 中从 40H 单元开始的 20 个单元中 0 的个数，结果存于 R2 中。

解：用 R0 作间址寄存器读取数据，R7 作循环变量，用 JNZ 或 CJNE 判断数据是否为 0，用 DJNZ 指令和 R7 控制循环。

程序段一：

```
    MOV     R0,#40H
    MOV     R7,#20
    MOV     R2,#0
LOOP:
    MOV     A,@R0
    JNZ     NEXT
    INC     R2
NEXT:
    INC     R0
    DJNZ    R7,LOOP
```

程序段二：

```
    MOV     R0,#40H
    MOV     R7,#20
    MOV     R2,#0
LOOP:
    CJNE    @R0,#0,NEXT
    INC     R2
NEXT:
    INC     R0
    DJNZ    R7,LOOP
```

3. 子程序调用和返回指令

这类指令有 3 条，一条子程序调用指令，两条程序返回指令。

1）子程序调用（长调用）指令

汇编指令格式	操作
LCALL addr16	;(SP)+1→SP、($PC_{7\sim0}$)→(SP), ;(SP)+1→SP、($PC_{15\sim8}$)→(SP), ;addr16→PC

本指令提供 16 位目标地址，因此可以调用 64KB 范围内任何地方的子程序。

指令的实际编写形式为："LCALL　目标地址标号或子程序名"。

2）子程序返回指令

汇编指令格式	操作
RET	;((SP))→$PC_{15\sim8}$、(SP)-1→SP, ;((SP))→$PC_{7\sim0}$、(SP)-1→SP

子程序返回时，只需要将堆栈中的返回地址弹出送给 PC，程序就自动接着调用前的程序继续执行。从堆栈中先弹出高 8 位地址，后弹出低 8 位地址。

3）中断服务程序返回指令

汇编指令格式	操作
RETI	;((SP))→$PC_{15\sim8}$、(SP)-1→SP, ;((SP))→$PC_{7\sim0}$、(SP)-1→SP

中断服务程序返回指令 RETI，除了具有"RET"指令的功能外，还将开放中断逻辑。

4. 空操作指令

汇编指令格式	操作
NOP	;无任何操作

这是一条单字节指令,执行时,不做任何操作(即空操作),仅将程序计数器(PC)值加1,使CPU指向下一条指令继续执行,它要占用一个机器周期,常用来产生时间延迟和程序缓冲。

细心的读者会发现,以上只有15条指令,还少两条指令,这两条指令是AJMP和ACALL,称为绝对转移(也叫短转移)指令和绝对子程序调用(也叫短调用)指令,这两条指令的转移范围是绝对划定的2KB范围之内,用不好会出现错误,并且其编码也不好理解(见附录A),唯一的优点只是比LJMP和LCALL指令少一个字节。在存储器容量变大、价格低廉的今天,其唯一的优点也没有了意义,所以没有必要使用这两条指令。

3.3.5 位操作指令

位操作指令又叫布尔处理指令。MCS-51单片机有一个位处理器(布尔处理器),它具有一套处理位变量的指令集,有位数据传送指令、位逻辑操作指令、控制程序转移指令。

在进行位操作时,位累加器为C,即进位标志CY。位地址是片内RAM字节地址20H～2FH单元中连续的128个位(位地址为00H～7FH)和部分SFR,累加器A和寄存器B(位地址E0H～E7H和F0H～F7H)中的位与00H～7FH位一样,都可以作软件标志或位变量。

在汇编语言中,位地址可以用以下4种方式表示:

(1) 直接位地址(00H～FFH)。如18H。

(2) 字节地址带位号。如20H.0,表示20H单元的第0位。

(3) 特殊功能寄存器名带位号。如P2.3,表示P2口的第3位。

(4) 位符号地址。可以是特殊功能寄存器位名,也可以是用位地址符号命令BIT定义的位符号,如flag(flag应在这之前定义过,如flag　BIT　05H)。

例如,用上述4种方式都可以表示PSW(D0H)中的第2位,分别为:D2H、D0H.2、PSW.2、OV。

MCS-51单片机共17条位操作指令,为了讨论方便,将其分成三组。

1. 位传送指令

位传送指令有两条,实现位累加器C与一般位之间的数据传送。

```
汇编指令格式                操作
MOV  C,bit                  ; (bit)→C
MOV  bit,C                  ; (C)→bit
```

【例3-16】 编写程序,把片内RAM中07H位的数值,传送到ACC.0位。

解:程序段如下:

```
MOV  C,07H
MOV  ACC.0,C
```

注意:一般位之间不能够直接传送,必须借助于C。

2. 位逻辑操作指令

位逻辑操作指令包括位清0、位置1、位取反、位与、位或,共10条指令。

1）位清 0 指令

汇编指令格式	操作
CLR　C	; 0→C
CLR　bit	;0→bit

2）位置 1 指令

汇编指令格式	操作
SETB　C	; 1→C
SETB　bit	; 1→bit

3）位取反指令

汇编指令格式	操作
CPL　C	; $\overline{(C)}$→C
CPL　bit	; $\overline{(bit)}$→bit

4）位与指令

汇编指令格式	操作
ANL　C,bit	; (C) ∧ (bit)→C
ANL　C,$\overline{bit}$	;(C) ∧ $\overline{(bit)}$→C

5）位或指令

汇编指令格式	操作
ORL　C,bit	; (C) ∨ (bit)→C
ORL　C,$\overline{bit}$	; (C) ∨ $\overline{(bit)}$→C

3. 位条件转移指令

位条件转移指令是以 C 或 bit 为判断条件的转移指令，共 5 条指令。

1）以 C 为条件的转移指令

汇编指令格式	操作
JC　rel	; 若(C) = 1,则(PC) + rel→PC
	; 若(C) = 0,则顺序向下执行
JNC　rel	; 若(C) = 0,则(PC) + rel→PC
	; 若(C) = 1,则顺序向下执行

2）以 bit 为条件的转移指令

汇编指令格式	操作
JB　bit,rel	; 若(bit) = 1,则(PC) + rel→PC
	; 若(bit) = 0,则顺序向下执行
JNB　bit,rel	; 若(bit) = 0,则(PC) + rel→PC
	; 若(bit) = 1,则顺序向下执行
JBC　bit,rel	; 若(bit) = 1,则(PC) + rel→PC,且 0→bit;
	; 若(bit) = 0,则顺序向下执行

【例 3-17】 编写程序，利用位操作指令，实现图 3-5 所示的硬件逻辑电路功能。

解：程序段如下：

```
MOV     C,P1.1          ; (P1.1)→C
ORL     C,P1.2          ; (C)∨(P1.2)→C
CPL     C
ANL     C,P1.0          ; (C)∧(P1.0)→C
CPL     C
MOV     F0,C            ; (C)→F0 位
MOV     C,P1.3          ; (P1.3)→C
ANL     C,/P1.4         ; (C)∧(/P1.4)→C
CPL     C
ORL     C,F0            ; (C)∨(F0)→C
MOV     P1.5,C          ; (C)→P1.5
```

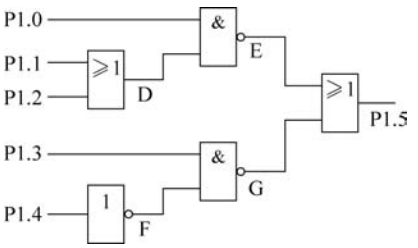


图 3-5　硬件逻辑电路

3.4　MCS-51 单片机伪指令

伪指令是汇编程序中用于指示汇编程序如何对源程序进行汇编的指令。伪指令不同于指令，在汇编时并不翻译成机器代码，只是在汇编过程进行相应的控制和说明。

伪指令通常在汇编程序中用于定义数据、分配存储空间、控制程序的输入/输出等。在MCS-51 系统中，常用的伪指令有以下 7 条。

1. 汇编地址设置伪指令 ORG

ORG 常用于汇编语言某程序段或某个数据块的开始，指明其汇编地址。一般格式为：

```
[标号:]    ORG  16 位地址
```

其标号为可选项。例如：

```
  ORG    0040H
MAIN:
  MOV    SP,#0DFH
  MOV    30H,#00H
    ……
```

此处的 ORG 伪指令指明后面的程序从 0040H 单元开始存放。

2. 结束汇编伪指令 END

END 伪指令用于汇编语言程序段的末尾，指示源程序在 END 处结束汇编，即便是 END 后面还有程序，也不做处理。一般格式为：

```
    ……
END
```

3. 符号定义伪指令 EQU

EQU 也称为赋值伪指令，其一般格式为：

```
符号名  EQU      项(常数、常数表达式、字符串或地址标号)
```

EQU 的功能是将右边的项赋给左边。在汇编过程中，遇到 EQU 定义的符号名，就用其右边的项代替符号名。需要注意的是，EQU 只能先定义后使用。例子如下。

```
HOUR  EQU    30H                    ;定义变量 HOUR 的地址为 30H
```

```
MINU  EQU   31H                ;定义变量 MINU 的地址为 31H
REGI  EQU   R7                 ;定义字符串 R7
DISP  EQU   0800H              ;定义变量 DISP 的地址为 0800H
      MOV   HOUR,#09H          ;变量 HOUR 赋值 9
      MOV   R0,#HOUR           ;使指针 R0 指向 30H 单元
      INC   R0                 ;指针 R0 增 1
      MOV   @R0,#25            ;变量 MINU 赋值 25
      MOV   REGI,A             ;(A)→R7
      LCALL DISP               ;调用首地址为 0800H 处的子程序
```

4. 变量定义伪指令 DATA

DATA 伪指令称为数据地址符号伪指令。其一般格式为：

```
符号名  DATA  常数或常数表达式
```

DATA 的功能与 EQU 相似，是将右边的项赋值给左边。在汇编过程中遇到 DATA 定义的符号名，就用其右边的项代替符号名。该伪指令用于定义片内数据区变量。

与 DATA 类似的还有 IDATA、XDATA、CODE 等伪指令，分别用于定义其他数据区的变量。

注意：DATA 可以后定义先使用，当然也可以先定义后使用。例如：

```
HOUR  DATA  30H                ;定义变量 HOUR 的地址为 30H
MINU  DATA  31H                ;定义变量 MINU 的地址为 31H
      MOV   HOUR,#09H          ;变量 HOUR 赋值 9
      MOV   R0,#HOUR           ;使指针 R0 指向 30H 单元
      INC   R0                 ;指针 R0 增 1
      MOV   @R0,#25            ;变量 MINU 赋值 25
```

5. 位变量定义伪指令 BIT

BIT 伪指令称为位地址符号伪指令。其格式为：

```
符号名  BIT   位地址
```

BIT 伪指令的功能是把右边的地址赋给左边的符号名。位地址可以是前面所述的 4 种形式中的任一种。例如：

```
FLAGRUN   BIT  00H
FLAGMUS   BIT  01H
FLAGKEY   BIT  02H
FLAGALAR  BIT  P1.7
```

6. 字节数据定义伪指令 DB

DB 伪指令的一般格式为：

```
[标号:]  DB   项(字节数据、字节数表或字符、字符串)
```

它的功能是从指定单元开始定义并存储若干字节的数据或字符、字符串，字符或字符串需要用引号（单引号或双引号均可）括起来，即用 ASCII 码表示。其中标号是可选的。例如：

```
TABLE: DB  32,24H,'A',"B",'abcd',"EFGH"
```

7. 字数据定义伪指令 DW

DW 伪指令的一般格式为：

```
[标号:]  DW     字数据或字数据表
```

DW 伪指令的功能与 DB 相似，是从指定单元开始定义并存储若干字数据，每个数据都占 2 个字节，而用 DB 伪指令定义的数据只占一个字节。其中标号是可选的。例如：

```
ORG      1000H
TABLE2:  DW       32,24H,1234H
```

上面这两行程序汇编后，从 1000H 单元开始，依次存放如下数据：

```
(1000H) = 00H
(1001H) = 20H
(1002H) = 00H
(1003H) = 24H
(1004H) = 12H
(1005H) = 34H
```

注意：高字节存放在前面(低地址)，低字节存放在后面(高地址)。

3.5 汇编语言程序设计

3.5.1 简单程序设计

程序的简单和复杂是相对而言的，这里所说的简单程序，是指顺序执行的程序。简单程序从第一条指令开始，依次执行每一条指令，直到程序执行完毕，之间没有任何转移和子程序调用指令，整个程序只有一个入口和一个出口。这种程序虽然在结构上简单，但它是复杂程序的基础。

1. 数据拆分

【例 3-18】 片内 RAM 的 30H 单元内存放着一压缩的 BCD 码，编写程序，将其拆开并转换成两个 ASCII 码，分别存入 31H 和 32H 单元中，高位在 31H 中。

解：数字 0～9 的 ASCII 码为 30H～39H，因此，将 30H 中的两个 BCD 码拆开后，分别加上 30H 即可。相应程序段如下：

```
MOV     R0,＃30H          ; 用间址寄存器 R0 存取数据
MOV     A,@R0             ; 取原 BCD 码数据
PUSH    ACC               ; 原 BCD 码数据进栈暂存
SWAP    A                 ; 将高位数交换到低 4 位
ANL     A,＃0FH           ; 先作高位转换,截取高位数
ORL     A,＃30H           ; 高位转换成 ASCII 码
INC     R0                ; 使 R0 指向 31H 单元
MOV     @R0,A             ; 保存高位 ASCII 码
POP     ACC               ; 原 BCD 码数据出栈
ANL     A,＃0FH           ; 作低位转换,截取低位数
ORL     A,＃30H           ; 低位转换成 ASCII 码
```

```
INC     R0              ; 使 R0 指向 32H 单元
MOV     @R0,A           ; 保存低位 ASCII 码
SJMP    $               ; CPU 停留于此处
```

2. 数制转换

【例 3-19】 片内 RAM 的 30H 单元内存放着一 8 位二进制数，编写程序，将其转换成压缩的 BCD 码，分别存入 30H 和 31H 单元中，高位在 30H 中。

解：其方法是用除法实现。原数除以 10，余数为个位数，其商再除以 10，所得新商为百位数，新余数为十位数。对应程序段如下：

```
MOV     A,30H           ; 取数据
MOV     B,#10
DIV     AB              ; 除以 10 后，个位在 B，百位和十位在 A
MOV     31H,B           ; 保存个位于 31H 中的低 4 位
MOV     B,#10
DIV     AB              ; 除以 10 后，十位在 B，百位在 A
MOV     30H,A           ; 保存百位数
MOV     A,B             ; 十位数送 A
SWAP    A               ; 十位数被交换到高 4 位
ORL     31H,A           ; 将十位数存于 31H 中的高 4 位
SJMP    $
```

3.5.2　分支程序设计

在许多情况下，程序会根据不同的条件，转向不同的处理程序，这种结构的程序称为分支程序。使用条件转移指令、比较转移指令和位条件转移指令，可以实现程序的分支处理。

在汇编语言程序中，分支结构是比较麻烦的，初学时应特别注意。

1. 一般分支程序

【例 3-20】 片内 RAM 的 30H、31H 单元存放着两个无符号数，编写程序比较其大小，将其较大者存于 30H 中，较小者存于 31H 单元中。

解：用减法判断，两个数相减后，通过借位标志位 CY 来判断。程序段如下：

```
    MOV     A,30H
    CLR     C
    SUBB    A,31H
    JNC     L1              ; (30H)≥(31H)则转
    MOV     A,30H           ; (30H)中数小，两个数交换
    XCH     A,31H
    MOV     30H,A
L1: SJMP    $
```

【例 3-21】 片内 RAM 的 30H 单元内存放着一有符号二进制数变量 X，其函数 Y 与变量 X 的关系为：

$$Y=\begin{cases} X+5 & X>20 \\ 0 & 20\geqslant X\geqslant 10 \\ -5 & X<10 \end{cases}$$

编写程序，根据变量值，将其对应的函数值送入 31H 中。

解：这是一个三分支的条件转移程序，可以使用CJNE、JC、JNC等指令进行判断。程序流程图如图3-6所示，程序段如下：

```
    MOV     A,30H
    CJNE    A,#10,L1
L1:
    JNC     L2              ; X≥10 转 L2
    MOV     31H,#0FBH       ; X<10,Y = -5
    SJMP    L4              ; X≥10
L2:
    ADD     A,#5            ; 先按 X>20 处理,Y = X + 5
    MOV     31H,A
    CJNE    A,#26,L3
L3: JNC     L4              ; X>20,转
    MOV     31H,#0          ; 20≥X≥10,Y = 0
L4: SJMP    $
```

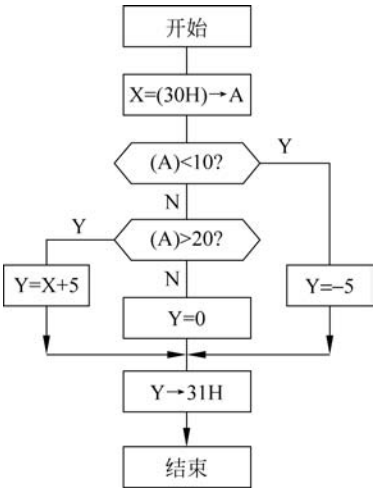


图3-6　例3-21程序流程图

2. 多分支程序

利用间接转移指令JMP　@A+DPTR，可以实现多分支转移，即实现散转。可以参考例3-13，不再举例。

3.5.3　循环程序设计

在实际应用中，循环结构程序使用得非常多，必须要熟练掌握。循环程序一般由以下几个部分组成：

(1) 循环初始化部分。这一部分位于循环程序的开始，用于对循环变量、其他变量和常量赋初值，做好循环前的准备工作。

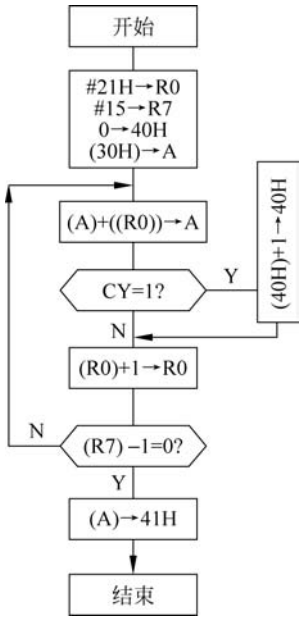


图3-7　例3-22程序流程图

(2) 循环体部分。这一部分由重复执行部分和循环控制部分组成。重复执行部分需要根据具体功能编写，要求尽可能简洁，以提高执行的效率。循环控制部分由修改循环控制变量和条件转移语句等组成，用于控制循环的次数。

(3) 循环结束部分。这一部分用于存放循环结果、恢复所占用寄存器或内存的数据等。

循环程序的关键是对循环变量的修改和控制，特别是循环次数的控制。在循环次数已知的情况下用计数的方法控制循环，在循环次数未知的情况下，往往需要根据给出的某种条件，判断是否结束循环。

1. 单层循环程序

【例3-22】 在片内RAM的20H～2FH单元，存放着16个无符号字节数据，编写程序，计算这16个数的和。

解：16个字节数的和不会超过两个字节，将和存于40H、41H中，高字节在40H中。用R0作取加数指针，R7作控制循环计数变量。流程图如图3-7所示，程序段如下。

```
  MOV    R0,#21H          ; R0 指向 21H 单元
  MOV    R7,#15           ; 控制循环次数初值
  MOV    40H,#0           ; 高字节清 0
  MOV    A,20H            ; 取第一个加数
LOOP:
  ADD    A,@R0            ; 低字节加上一个数
  JNC    NEXT             ; 无进位跳转
  INC    40H              ; 有进位高字节加 1
NEXT:
  INC    R0               ; 指针增 1
  DJNZ   R7,LOOP          ; R7 减 1 不为 0 继续循环
  MOV    41H,A            ; 保存低字节数据
  SJMP   $
```

2. 双层循环程序

【例 3-23】 设计一软件延时 xms 的子程序。设晶振频率为 12MHz。

解：机器周期为 1μs。子程序如下：

```
DELAYxMS:                       ; 机器周期数
;      MOV    R7,#x             ; 1.本句在调用程序中
LP1:   MOV    R6,#249           ; 1
       NOP                      ; 1
LP2:   NOP                      ; 1
       NOP                      ; 1
       DJNZ   R6,LP2           ; 2
       DJNZ   R7,LP1           ; 2
       RET                      ; 2
```

延时时间：

$$[1+1+4\times249+2]\times x+2=(1000x+2)\mu s=xms+2\mu s$$

延时时间为 xms 多 2μs。如果考虑到子程序调用指令 LCALL 及其前面的 R7 赋值指令，分别需要用两个和一个机器周期，实际延时时间仅多出 5μs，并且此误差与延时的 ms 数 x 无关。误差是非常小的。

需要注意的是，此延时子程序的延时范围为 1～255ms。

3.5.4　子程序设计

子程序是指完成某一确定任务，并且能够被其他程序反复调用的程序段。采用子程序，可以简化程序，提高编程效率。而且从程序结构上看，逻辑关系简单、清晰，便于阅读和调试，实现程序模块化。

子程序在结构上有一定的要求，编写时需要注意：

① 子程序第一条指令的地址称为入口地址，该指令前必须要有标号，其标号一般要能够说明子程序的功能。

② 子程序末尾一定要有返回指令。而调用子程序的指令应该在其他程序中。

③ 在子程序中，要注意保护在主调程序中使用的寄存器和存储单元中的数据，必要时在子程序的开始使其进栈保护，在子程序返回前再出栈恢复原来值。

④ 在子程序中，要明确指出“入口参数”和“出口参数”，入口参数就是在调用前需要给

子程序准备的数据,出口参数就是子程序的返回值。

参数的传递有以下几种方式:

(1) 通过寄存器 R0～R7 或累加器 A。

(2) 传递地址、入口参数和出口参数的数据存放在存储器中,使用 R0、R1 或 DPTR 传递指向数据的地址。

(3) 通过堆栈传递参数。

1. 用寄存器传递参数

【例 3-24】 试编写程序,把存放在 30H、31H 和 40H、41H 中的两个双字节压缩的 BCD 码数相减,结果回存到被减数的 30H、31H 中。高位数在 30H、40H 中。要求使用子程序。

解:由于计算机内部加减都是按照二进制数进行的,所以对 BCD 码数据相减后,需要进行十进制数调整,为了实现十进制数调整,将减法运算转变为加法运算。在子程序中完成 BCD 码相减,通过寄存器传递参数,程序如下:

```
    MOV    R0,31H
    MOV    R1,41H
    CLR    C
    LCALL  BCDSUB              ; 计算低字节差值
    MOV    31H,A               ; 保存低字节差值
    MOV    R0,30H
    MOV    R1,40H
    LCALL  BCDSUB              ; 计算高字节差值
    MOV    30H,A               ; 保存高字节差值
    SJMP   $
    ; BCD 码减法子程序
    ; 入口参数: R0 被减数; R1 减数; C 借位位
    ; 出口参数: A 差值,为 BCD 码; C 借位位
BCDSUB:
    MOV    A,#9AH
    SUBB   A,R1                ; 把减数转变成十进制数的补码
    ADD    A,R0                ; 被减数加上减数的补码
    DA     A                   ; 做 BCD 码调整
    CPL    C
    RET
```

2. 用堆栈传递参数

【例 3-25】 编写程序,把片内 RAM 的 30H 单元中的 8 位二进制数转换成 ASCII 码,分别存放到 31H、32H 中,31H 中存放高位 ASCII 码。要求使用子程序。

解:在子程序中通过查表完成转换,主调程序与子程序的参数通过堆栈进行传递。程序如下:

```
    MOV    SP,#0DFH            ; 设置堆栈指针,把堆栈放在片内 RAM 高端
    MOV    DPTR,#TAB           ; DPTR 指向数表的首地址
    MOV    A,30H
    SWAP   A                   ; 先对高位进行转换
    PUSH   ACC                 ; 高位数据进栈
```

```
    LCALL   HEX_ASC              ; 调用转换子程序
    POP     31H                  ; 转换结果出栈并保存在 31H 中
    PUSH    30H                  ; 低位数据进栈
    LCALL   HEX_ASC              ; 调用转换子程序
    POP     32H                  ; 转换结果出栈并保存在 32H 中
    SJMP    $

    ; 十六进制数转换 ASCII 码子程序
    ; 入口参数: 栈顶之下第 2 单元(对主调程序来说是栈顶)
    ; 出口参数: 存放在栈顶之下第 2 单元(对主调程序来说是栈顶)
HEX_ASC:
    MOV     R0,SP                ; R0 指针指向栈顶
    DEC     R0                   ; 修改指针使其指向栈顶之下第 2 单元
    DEC     R0
    MOV     A,@R0                ; 从堆栈中读取参数
    ANL     A,#0FH               ; 屏蔽高 4 位
    MOVC    A,@A+DPTR            ; 查表读取 ASCII 码
    MOV     @R0,A                ; 将转换结果保存到栈顶之下第 2 单元
    RET
TAB: DB     "0123456789ABCDEF"
```

思考题与习题

(1) 简述 MCS-51 汇编指令格式。

(2) 何谓寻址方式? MCS-51 单片机有哪些寻址方式,是怎样操作的? 各种寻址方式的寻址空间和范围是什么?

(3) 访问片内 RAM 低 128 字节可使用哪些寻址方式? 访问片内 RAM 高 128 字节使用什么寻址方式? 访问 SFR 使用什么寻址方式?

(4) 访问片外 RAM 使用什么寻址方式?

(5) 访问程序存储器使用什么寻址方式? 指令跳转使用什么寻址方式?

(6) 分析下面指令是否正确,并说明理由。

```
MOV     R3,R7
MOV     B,@R2
DEC     DPTR
MOV     20H,F0H
PUSH    DPTR
CPL     36H
MOV     PC,#0800H
```

(7) 分析下面各组指令,区分它们的不同之处。

```
MOV     A,30H       与      MOV     A,#30H
MOV     A,R0        与      MOV     A,@R0
MOV     A,@R1       与      MOVX    A,@R1
MOVX    A,@R0       与      MOVX    A,@DPTR
MOVX    A,@DPTR     与      MOVC    A,@A+DPTR
```

(8) 已知单片机的片内 RAM 中(30H)=38H、(38H)=40H、(40H)=48H、(48H)=90H。请说明下面各是什么指令和寻址方式,每条指令执行后目的操作数的结果。两段程序是独立的。

程序段一:

```
MOV    P1,#0FH
MOV    40H,30H
MOV    P0,48H
MOV    48H,#30H
MOV    DPTR,#1234H
```

程序段二:

```
MOV    A,40H
MOV    R0,A
MOV    @R0,30H
MOV    R0,38H
MOV    A,@R0
```

(9) 已知单片机中(A)=23H、(R1)=65H、(DPTR)=1FECH,片内 RAM 中(65H)=70H, ROM 中(205CH)=64H。试分析下列各条指令执行后目标操作数的内容。

```
MOV    A,@R1
MOVX   @DPTR,A
MOVC   A,@A+DPTR
XCHD   A,@R1
```

(10) 已知单片机中(R1)=76H、(A)=76H、(B)=4、CY=1,片内 RAM 中(76H)=0D0H、(80H)=6CH。试分析下列各条指令执行后目标操作数的内容和相应标志位的值。

```
ADD    A,@R1
SUBB   A,#75H
MUL    AB
DIV    AB
ANL    76H,#76H
ORL    A,#0FH
XRL    80H,A
```

(11) 已知单片机中(A)=83H、(R0)=17H,片内 RAM 中(17H)=34H。试分析当执行完下面程序段后累加器 A、R0、17H 单元的内容。

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

(12) 阅读下面程序段,说明该段程序的功能。

```
    MOV    R0,#40H
    MOV    R7,#10
    CLR    A
LOOP:
    MOV    @R0,A
    INC    A
    INC    R0
    DJNZ   R7,LOOP
    SJMP   $
```

(13) 阅读下面程序段,说明该段程序的功能。

```
    MOV    R0,#50H
    MOV    R1,#00H
    MOV    P2,#01H
    MOV    R7,#20
LOOP:
    MOV    A,@R0
    MOVX   @R1,A
    INC    R0
    INC    R1
    DJNZ   R7,LOOP
    SJMP   $
```

(14) 阅读下面程序段,说明该段程序的功能。

```
MOV    R0,#40H
MOV    A,@R0
INC    R0
ADD    A,@R0
MOV    43H,A
CLR    A
ADDC   A,#0
MOV    42H,A
SJMP   $
```

(15) 编写程序,用位处理指令实现“P1.4=P1.0∨(P1.1∧P1.2)∨P1.3”的逻辑功能。

(16) 编写程序,若累加器 A 的内容分别满足下列条件,则程序转到 LABEL 存储单元,否则顺序执行。设 A 中存放的是无符号数。

① A≥10;　② A>10;　③ A≤10。

(17) 编写程序,把片外 RAM 从 0100H 开始存放的 16 字节数据,传送到片内从 30H 开始的单元中。用 Keil C 编译并调试运行,观察、对比两个储存器中的数据。

(18) 片内 RAM30H 和 31H 单元中存放着一个 16 位的二进制数,高位在前,低位在后。编写程序对其求补,并存回原处。

(19) 片内 RAM 的 30H 到 33H 单元中存放着两个 16 位的无符号二进制数,高位在前,低位在后,将其相加,其结果保存到 30H、31H 单元,高位放在前面。用 Keil C 编译并调试运行,观察、分析储存器中数据的变化情况。

(20) 片内 RAM 的 30H 到 33H 单元中存放着两个 16 位的无符号二进制数,高位在前,低位在后,将其相减(前面数减去后面数),其结果保存到 30H、31H 单元,高位放在前面。

(21) 片内 RAM 中有两个 4 字节压缩的 BCD 码形式存放的十进制数,一个存放在 30H～33H 单元中,另一个存放在 40H～43H 单元中,高位数在低地址。编写程序将它们相加,结果的 BCD 码存放在 30H～33H 中。用 Keil C 编译并调试运行,观察、分析储存器中的数据。

(22) 编写程序,查找片内 RAM30H～50H 单元中是否有 55H 这一数据,若有,则 51H 单元置为 FFH;若未找到,则将 51H 单元清 0。用 Keil C 编译并调试运行,观察、分析 51H

中的数据是否正确。

(23) 编写程序,查找片内 RAM 的 30H～50H 单元中出现 0 的次数,并将查找的结果存入 51H 单元。用 Keil C 编译并调试运行,观察、分析 51H 中的数据是否正确。

(24) 编写程序,将程序存储区地址从 0010H 开始的 20 个字节数据,读取到片内 RAM 从 30H 单元开始的区域,然后用冒泡法从大到小进行排序。用 Keil C 编译并调试运行,观察、对比两个储存器中的数据,分析是否正确。