第5章

逻辑与运算指令及程序

本章导读：

软件是逻辑、运算与数据的组合，逻辑与运算在单片机软件编程中占有重要地位。单片机没有专用的浮点处理器，数据处理能力较弱，直接进行浮点数等复杂运算非常困难，因而逻辑与运算程序需要更多技巧。

单片机的主要编程语言有汇编语言和C语言，利用汇编语言需要直接编写逻辑与运算程序，能调用的库函数较少，处理计算问题复杂，对编程技巧要求较高，但对于理解单片机应用系统的编程原理、优化程序结构，都有着非常重要的作用。如何掌握单片机汇编语言和C语言的逻辑与运算程序的设计，是本章讨论的主要问题。

本章主要内容：

本章的主要内容为MCS-51系列单片机的逻辑与运算汇编指令、标志位，C语言的逻辑与运算符号、常见的逻辑与运算程序编写、定点数与浮点数的理解。通过本章学习应该达到以下目的：熟悉MCS-51系列单片机的逻辑与运算指令，熟练掌握常见的逻辑与运算程序编写，掌握单片机的程序设计方法，熟悉定点数与浮点数、浮点运算定点化等编程技巧。

5.1 单片机的标志位

一般在程序指令执行后，其运行状态需要保存在状态字寄存器PSW(Program Status Word)中，以便于在之后的逻辑与运算程序中处理。MCS-51 PSW的格式和功能如表5-1所示。

表5-1 MCS-51 PSW的格式和功能

*D*7	*D*6	*D*5	*D*4	*D*3	*D*2	*D*1	*D*0
CY	AC	F0	RS1	RS0	OV	F1	P

各标志位的作用：

- CY：进借位标志，运算中出现进位或者借位时为1。8051中的运算器是一种8位的运算器，表示0～255。如果加减法时超过这个范围，就会使CY＝1，否则CY＝0。该标志位常在多位运算中使用。
- AC：半进位标志或辅助进借位标志。运算中，当运算数的$D3$位向$D4$位产生进位或者借位时AC为1。该标志位主要用于二-十进制数加法的十进制调整。
- F0：用户标志，可由用户自行设定，常用于用户的程序逻辑标志。
- RS1、RS0：工作寄存器组选择位。00、01、10、11分别代表$R0$～$R7$位于内部RAM区的00H～07H(0区)、08H～0FH(1区)、10H～17H(2区)、18H～1FH(3区)，常用于主程序与子程序的工作寄存器组切换。
- OV：溢出标志。若操作结果$D6$有进位进入$D7$但$D7$没有产生进位(CY＝0)，或者$D7$产生进位(CY＝1)而$D6$没有向$D7$进行进位，则置位溢出标志位OV，反之清0。OV＝1可以出现在以下3种情况下：带符号运算结果超过范围(－127～128)、无符号运算结果超过范围(255)或除数为0。该位常用于判断运算是否溢出。
- F1：目前多数产品中，该位可以作为用户标志位使用，大部分用法与F0相同。
- P：奇偶标志，A中有奇数个1时置1，有偶数个1时清0，常用于奇偶校验。

5.2 逻辑及运算指令

MCS-51汇编指令系统用42种操作码助记符来表达33种操作功能，每种操作可以使用一种以上的数据类型，助记符定义所访问的存储空间，一种功能可能有几个助记符。寻址方式同功能助记符组合在一起，共111种指令。MCS-51汇编指令系统具有存储效率高、执行速度快等特点。其指令可以按不同指标分类：按照执行所需要的时间来分类，共有64条单周期指令、45条双周期指令和2条四周期指令(乘与除运算)；按照字节数来分类，则有49条单字节指令、45条双字节指令和17条三字节指令；按功用可以分成5类，即数据传送类(29条)、算术操作类(24条)、逻辑操作类(24条)、控制程序转移类(17条)、布尔变量操作类(17条)。

各类指令的介绍方式将采取先描述该类指令的共同特征，然后按助记符逐条描述指令，包括助记符、操作码、执行的具体内容以及短小的应用例子。第2章与第3章已经介绍了各个传送类指令与转移类指令，本章将着重介绍其他各操作类指令。

5.2.1 算数操作指令

与同类8位微处理器相比，由于寄存器只有8位，所以MCS-51的算术运算指令并没有16位运算指令，但是增加了除法和乘法指令。这一方面不如某些微处理器(如Z80)的指令系统。算术操作类指令共有24条，包括4种基本的加减乘除算术操作指令，这4种指令能对8位的无符号数进行直接运算，还可利用溢出标志位对带符号数进行补码运算。有了进位标志位的帮助，还可实现多精度的加减和环移，同时也可对压缩的BCD数进行运算。算术运算指令执行的结果将使程序状态字PSW的某些位发生置位或复位，但是加1或减1指

令不影响这些标志位。表 5-2 给出了包括一些非算术操作的指令在内的对标志位有影响的所有指令。

表 5-2　影响标志位的指令

指令助记符	影响标志位		
	CY	OV	AC
ADD	*	*	*
ADDC	*	*	*
SUBB	*	*	*
MUL	0	*	
DIV	0	*	
DA	*		
RRC	*		
RLC	*		
SETB C	1		
CLR　C	0		
CPL　C	*		
ANL　C,bit	*		
ANL　C，/bit	*		
ORL　C，bit	*		
ORL　C，/bit	*		
MOV　C，bit	*		
CJNE	*		

注：* 表示根据运行的结果使该标志位置位或复位，1 表示置位，0 表示复位。

算术操作类指令用到的助记符有 ADD、ADDC、INC、DA、SUBB、DEC、MUL 和 DIV 8 种，下面分别介绍。

1. 不带进位位的加法指令

指令格式包括：

```
ADD A, #DATA
ADD A, direct
ADD A, Rn
ADD A, @Ri
```

这组指令的功能是把指令中指定的一个字节与累加器内容相加，最终结果放回到累加器 A 中。在该组指令中，参加运算的均为两个 8 位二进制数。对于程序设计人员来说，这些二进制数可以当成无符号数(0～255)，也可以当成带符号数，即补码数(－128～＋127)，这完全视程序设计情况而定。但计算机在作加法运算时，总是按如下规则进行：求和时，操作数直接相加，不需要做任何变换。

例 5.1　分析下式按照无符号数与带符号数分别计算时的结果。

```
    10110000
+   00001010
------------
    10111010
```

分析：如果认为是无符号数相加，则这个结果代表十进制数186。如果认为是带符号数相加，则它代表十进制数－70。

两个数相加时，当$D6$和$D7$位不同时有进位时，溢出标志位OV将置位(OV＝1)。如果认为处理的是无符号数，那么8位二进制数表示的范围为00H～FFH(0～255)，超过此范围的数就会产生溢出，在这种情况下用CY＝1表示数据有溢出。但如果处理的是带符号数，则数$D7$位用作符号位，表示的数范围为－128～＋127。在这种情况下，数据是否溢出不能用CY表示，而是用OV＝1表示数据有溢出。因此在进行带符号数的加法运算时，OV是一个极其重要的编程标志，MCS-48指令系统中则没有这个标志。辅助进位标志位AC和奇偶标志位P也会受到加法指令的影响。

例5.2 设(A)＝0B9H，(Rn)＝8AH，执行语句“ADD A，Rn”。

解：

```
        10111001
    +   10001010
-------------------
    1)  01000011
```

程序执行后(A)＝43H，CY＝1，OV＝1，AC＝1，P＝1。

2. 带进位位的加法指令

指令格式包括：

```
ADDC A, Rn
ADDC A, direct
ADDC A, @Ri
ADDC A, #data
```

该组指令的功能是将A中的值和其后面的值相加，并且加上进位位CY中的值。由于51系列单片机是一种8位机，只能做8位的数学运算，但8位运算的范围只有0～255，这在实际工作中是远远不够的，因此需要进行扩展。一般是将两个8位的数学运算合起来，成为一个16位的运算，这可以表达的数的范围就可以达到0～65 535。

如何合并呢？其实很简单，先来看一个十进制数的66＋78，这两个数相加，先做低位6＋8，然后再做高位6＋7，做了两次加法。之所以要分成两次来做，是因为这两个数超过了一位数所能表达的范围(0～9)。在做低位时产生了进位，需要在适当的位置记住低位的进位，然后再做高位加法时将进位加进去。实际上，计算机中做16位加法时同样如此，先做低8位，如果两数相加产生了进位，也要在PSW中的进位位CY做个标记，在进行高位加法时将这个CY加进去。

例如：1058H＋10B0H，先做58H＋B0H＝108H，而108H显然超过了0FFH，因此最终保存在A中的是08H，而1则到了PSW中的CY位；换言之，CY就相当于是01H。然后再做10H＋10H＋CY，结果是21H，所以最终的结果是2108H。

例5.3 设(A)＝87H，(20H)＝0F9H，执行语句“ADDC A，20H”。

解：

```
        10000111
        11111001
    +           1
-------------------
CY＝1   10000001
```

程序执行后(A)＝81H，CY＝1，OV＝1，AC＝1，P＝0。

3. 带借位的减法指令

指令格式包括：

```
SUBB A, Rn
SUBB A, direct
SUBB A, @Ri
SUBB A, #data
```

该组指令的功能是从累加器的内容中减去指定的一个变量和借位标志，执行结果放在累加器 A 中。当够减时，进位标志位 CY 复位。当不够减时，发生借位操作，进位标志位 CY 置位。SUBB 指令是带借位减的，它需要考虑前一次运算操作对 CY 的影响。因此在进行多字节减法运算时，在开始 SUBB 运算前，要首先将 CY 复位。没有不带借位的减法指令，如果需要做不带借位的减法指令，那么只要将 CY 清 0 即可。

例 5.4　设(A)＝87H，(20H)＝039H，执行语句“ADDC A，20H”。

解：

```
    10000111
    00111001
 −          1
------------
    01001101
```

程序执行后(A)＝4DH，CY＝0，OV＝1，AC＝1，P＝0。

4. 加 1 和减 1 指令

指令格式包括：

```
INC A
INC Rn
INC direct
INC @Ri
INC DPTR
```

上述命令以 INC(DEC)为操作码，累加器 A、Rn、direct 和@Ri 为操作数。功能是操作数的值加(减)1。除对累加器 A 操作影响 P 标志外，不影响任何标志位。

例 5.5　设(A)＝12H，(R0)＝34H，(21H)＝32H，(34H)＝22H，DPTR＝1234H，分别对上述值执行 INC 指令。

解：

```
INC A          ;执行后结果(A) = 13H
INC R0         ;执行后结果(R0) = 34H
INC 21H        ;执行后结果(21H) = 33H
INC @R0        ;执行后结果(34H) = 23H
INC DPTR       ;执行后结果(DPTR) = 1235H
```

从结果上看，语句“INC A”和语句“ADD A，#1”差不多，但前者是单字节、单周期指令，而后者则是双字节、双周期指令，而且前者不会影响 PSW 位。如(A)＝0FFH，INC A 后(A)＝00H，而 CY 依然保持不变。如果是“ADD A，#1”，则(A)＝00H，而 CY 一定是 1。加 1 指令常用做计数、地址增加等用途。DEC 指令与 INC 指令基本保持一致。

5. 乘法指令

格式为：

```
MUL AB
```

该指令的功能是将 A 和 B 中的两个 8 位无符号数相乘，两数相乘结果一般比较大，因此最终结果用 1 个 16 位数来表达，其中高 8 位放在 B 中，低 8 位放在 A 中。在乘积大于 0FFH(255)时，0V 置 1(溢出)，否则 OV 为 0，而进位标志位 CY 总是 0。

例 5.6 设(A)＝4EH，(B)＝5DH，执行语句“MUL AB”。

解：程序执行后(A)＝56H，(B)＝1CH，CY＝0，OV＝1。

6. 除法指令

格式为：

```
DIV AB
```

该指令的功能是将 A 中的 8 位无符号数除以 B 中的 8 位无符号数(A/B)。除法一般会出现小数，但计算机中无法直接表达小数转而使用商和余数。CY 和 OV 都是 0。如果在做除法前 B 中的值是 00H，也就是除数为 0，那么 OV＝1。

例 5.7 设(A)＝0EH，(B)＝03H，执行语句“MUL AB”。

解：程序执行后(A)＝04H，(B)＝02H，CY＝0，OV＝0。

7. 十进制调整指令

格式为：

```
DA A
```

该指令是对 A 的 BCD 码相加结果进行调整。两个压缩型 BCD 码按二进制数相加之后，必须经过该指令调整方能得到压缩型的 BCD 码的和数。如果累加器 A 的低 4 位数值大于 9，或者第 3 位向第 4 位有进位(即 AC＝1)，则需将 A 的低 4 位内容加 6 进行调整，从而产生正确的低 4 位 BCD 码值。如果调整后，低 4 位产生进位，且高 4 位在进位之前均为 1，则内部加操作将置位 CY。如果累加器 A 的高 4 位值大于 9 或 CY＝1，则高 4 位需加 6 调整，以产生正确的高 4 位 BCD 码值。同样，加 6 调整后若产生最高进位，则对 CY 置位。

例 5.8 设累加器 A 的内容为 0101 0110B，BCD 码为 56，寄存器 R3 的内容为 0110 0111B，为 BCD 码下的 67，CY 为 1。执行如下语句：

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

解：运算过程如下

	(A)＝01010110	
	(R3)＝01100111	
＋)	(CY)＝00000001	
	和 ＝10111110	
	调整＋) 01100110	
	100100100	BCD 码为 124

由于 DA 指令不影响溢出标志位，所以不能用 DA 指令对十进制减法操作的结果进行调整，借助进位标志位可实现多位 BCD 数加法结果的调整，BCD 数存放在累加器 A 中。

在实际使用过程中，常常需要结合数据传送指令和运算操作指令来完成更加复杂多变

的程序，比如双字节的加减法和数码管显示数码的拆分程序。

例 5.9　实现双字节加法运算。设被加数存放于 RAM 的 addr1(低位字节)和 addr2(高位字节)，加数存放于 addr3(低位字节)和 addr4(高位字节)，运算结果和数存于 addr1 和 addr2 中。试编写程序完成上述功能。

解：程序段如下：

```
MOV   R0, #addr1
MOV   R1, #addr3          ;取加数与被加数的低字节
MOV   A, @R0
ADD   A, @R1              ;低字节相加
MOV   @R0, A              ;保存于 addr1 中
MOV   R0, #addr2
MOV   R1, #addr4          ;取加数与被加数的高字节
MOV   A, @R0
ADD   A, @R1              ;高字节相加
MOV   @R0, A              ;保存于 addr2 中
```

例 5.10　实现双字节减法。设被减数存放于 R3(高字节)和 R4(低字节)中，减数存放于 R6(高字节)和 R7(低字节)中，运算结果存放在 R3(高字节)和 R4(低字节)中。试编写程序完成上述功能。

解：程序段如下：

```
MOV    A, R4
CLR    C
SUBB   A, R7     ;低位相减
MOV    R4, A
MOV    A, R3
SUBB   A, R6     ;高位相减
MOV    R3, A
```

5.2.2　逻辑操作类指令

逻辑操作类指令包括与、或、异或、清除、求反、左右移位等逻辑操作，该类指令共有 24 条。逻辑操作类指令用到的助记符有 CLR、CPL、RRC、RR、RLC、RL、XRL、ORL 和 ANL，下面分别详细描述。

1. 对累加器 A 的逻辑操作指令

CLR A：该指令的功能是将 A 中的值清 0，单周期单字节指令，与语句“MOV A，#00H”效果相同，不影响 CY、AC、OV 等标志位。

CPL A：该指令的功能是将 A 中的值按位取反。

RL A：左环移指令，该指令的功能是将 A 中的值逻辑左移一位，位 7 环移入位 0，不影响标志位。

RLC A：带进位左环移指令，该指令的功能是将 A 中的值加上进位标志位一起向左环移一位，位 7 移入进位 CY，CY 移入位 0，不影响其他标志位。

RR A：右环移指令，该指令的功能是将累加器 A 中的内容向右环移一位，位 0 环移入位 7，不影响标志位。

RRC A：带进位右环移指令，该指令的功能是将 A 中的值加上进位位进行逻辑右环移一位，位 0 进入 CY，CY 移入位 7。

SWAP A：累加器 A 半字节交换指令，该指令的功能是将 A 中的值高、低 4 位交换。

左右环移指令示意图如图 5-1 所示。

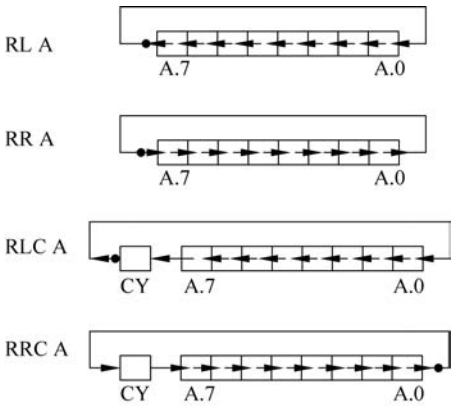

图 5-1　左右环移指令示意图

下面通过一些例子对这些指令的功能进行说明。

例 5.11　(A)＝73H，执行语句“CPL A”。

分析：73H 转换为二进制 0111 0011，逐位取反即为 1000 1100，即 8CH。

例 5.12　A 中的值为 68H，执行语句“RL A”。

分析：68H 转换为二进制 0110 1000，按上面的指令进行移动。0110 1000 转换为 1101 0000，即 D0H。

例 5.13　A 中的值为 68H，C 中的值为 1，执行语句“RLC A”。

分析：执行语句后，结果为 0 1101 0001，也就是 C 进位位的值变成了 0，而(A)则变成了 D1H。

对于指令 RR A 和 RRC A 不再赘述，可以参考上面两个例子自行练习。

例 5.14　(A)＝39H，分析执行语句“SWAP A”前后，A 中值的变化。

分析：执行语句后，A 中的值就是 93H。怎么正好是前后交换呢？因为这是一个十六进制数，每个十六进制位数字代表 4 个二进制位。注意，如果是(A)＝39，则执行语句后不是(A)＝93，要将它转换成二进制再算，39 转换为二进制是 10111，即 0001 0111，高 4 位为 0001，低 4 位为 0111，交换后是为 0111 0001，十六进制表示为 71H，即 113。

利用移位指令实现乘除法，只要乘以或除以一个整数才可以用移位的方法得到结果。前面已经学习了乘法和除法指令，为什么还要用移位指令来实现呢？这是因为移位指令占 2 个机器周期，而乘除法指令占 4 个机器周期，为了提高单片机的效率，这样做就十分必要了。实际使用过程中左移 n 位等于乘以 2^n，右移 n 位等于除以 2^n。若乘除法不是 2 的 n 次方时，如 A 乘以 9，可以拆分成 A×(8＋1)，即 A 乘以 2^3 加上 A，如乘法 A×7 则为 A×(8－1)。

例 5.15　简单移位实现乘除法示例。

```
MOV A, #7EH     ;二进制 01111110,十进制 126
MOV B, #7CH     ;二进制 01111100,十进制 124
```

```
RL A            ;二进制 11111100,十进制 252,相当于 126×2
RRB             ;二进制 00111110,十进制 62,相当于 124/2
```

例 5.16　复杂移位实现乘除法需要采用循环的方法来设计程序,对于 8×8 位的乘法需要循环 8 次,循环次数取决于乘数的位数。尝试根据图 5-2 的 8×8 位乘法写出程序。

```
                1 0 0 1 1 1 0 1
×               0 1 0 1 0 0 1 1
-----------------------------------
                1 0 0 1 1 1 0 1
              1 0 0 1 1 1 0 1
            0 0 0 0 0 0 0 0
          0 0 0 0 0 0 0 0
        1 0 0 1 1 1 0 1
      0 0 0 0 0 0 0 0
    1 0 0 1 1 1 0 1
+ 0 0 0 0 0 0 0 0
-----------------------------------
= 0 1 1 0 0 1 0 1 1 1 0 0 1 1 1
```

图 5-2　二进制乘法竖式计算示例

解：乘数带进位右移→积＋被乘数或 0→积右移,不管数据长度是多少,每一位的操作都是一样的。

程序示例如下(32 位二进制乘法子程序。乘数 1：73H～70H。乘数 2：77H～74H。积：7FH～78H。用到的寄存器如下：循环次数 R7,移位循环次数 R6,间址寄存器 R0,将指定的寄存器内容作为地址,由该地址所指定的单元内容作为操作数)：

```
MULTIPLY:               ;32 位二进制乘法子程序
    PUSH  07H           ;将程序会用到的寄存器
    PUSH  06H           ;R0,R6,R7 压入堆栈保存
    PUSH  00H

    MOV  R7, #32        ;设循环次数,与数据相关

    MOV  7FH, #0
    MOV  7EH, #0
    MOV  7DH, #0
    MOV  7CH, #0
    MOV  7BH, #0
    MOV  7AH, #0
    MOV  79H, #0
    MOV  78H, #0

MULTI0:
    CLR C
    MOV  R0, #73H
    MOV  R6, #4
```

```
MULTI1:
    MOV  A, @R0
    RRC  A
    MOV  @R0, A
    DEC  R0
    DJNZ  R6,MULTI1 ;乘数带进位右移
    JNZ  MULTI2     ;C=0,跳转

    MOV  A, 7CH
    ADD  A, 74H
    MOV  7CH, A
    MOV  A, 7DH
    ADDC  A, 75H
    MOV  7DH, A
    MOV  A, 7EH
    ADDC  A, 76H
    MOV  7EH, A
    MOV  A, 7FH
    ADDC  A, 77H
    MOV  7FH, A     ;积高 4 字节 + 被乘数
MULTI2:
    CLR  C
    MOV  R0, #7FH
    MOV  R6, #8
MULTI3:
    MOV  A, @R0
```

```
RRC  A
MOV  @R0, A
DEC  R0
DJNZ  R6, MULTI3 ;积右移一位
DJNZ  R7, MULTI0 ;循环字节位数次

POP  00H
POP  06H
POP  07H
RET
```

对于多字节除法，可以根据乘法类推，思考应该如何修改程序完成功能。

2. 逻辑与指令

指令格式包括：

```
ANL A, Rn
ANL A, direct
ANL A, @Ri
ANL A, #data
ANL direct, A
ANL direct, #data
```

该组指令的功能是在源操作数和目的操作数所指出的变量之间，按位执行逻辑与操作命令，结果存放在目的操作数中。

例 5.17 71H 和 56H 相与，将两数写成二进制形式。

解： (71H) 0111 0001

(56H) 0010 0110

结果为 0010 0000，即 20H。

两个参与运算的数，同一位中只要其中有一个为 0，这位的结果就是 0，均为 1 时结果才为 1。理解了逻辑与的运算规则，结果自然就出来了。

例 5.18 分析下列语句功能与运算结果。

```
MOV A, #45H              ;(A) = 45H
MOV R1, #25H             ;(R1) = 25H
MOV 25H, #79H            ;(25H) = 79H
ANL A, @R1               ;45H 与 79H 按位与，结果为 41H，送入 A 中，(A) = 41H
ANL 25H, #15H            ;25H 中的值(79H)与 15H 相与，结果为(25H) = 11H
ANL 25H, A               ;25H 中的值(11H)与 A 中的值(41H)相与，结果为(25H) = 11H
```

在知道了逻辑与指令的功能后，逻辑或和逻辑异或的功能就很简单了。逻辑或是按位“或”，即有 1 为 1，全 0 为 0。而异或则是按位“异或”，相同为 0，相异为 1。所有的或指令，就是将与指令中的 ANL 换成 ORL，而异或指令则是将 ANL 换成 XRL。或指令如下，对于异或指令 XRL 不再赘述。

```
ORL A, Rn                ;A 和 Rn 中的值按位"或"，结果送入 A 中
ORL A, direct            ;A 和间址寻址单元@Ri 中的值按位"或"，结果送入 A 中
ORL A, #data             ;A 和 direct 中的值按位"或"，结果送入 A 中
ORL A, @Ri               ;A 和立即数 data 按位"或"，结果送入 A 中
```

```
ORL direct, A              ;direct 中值和 A 中的值按位"或",结果送入 direct 中
ORL direct, #data          ;direct 中的值和立即数 data 按位"或",结果送入 direct 中
```

5.2.3　布尔变量操作类指令

MCS-51 硬件结构中有一个布尔处理器,它实际上是 1 位处理器,即它是以位为单位来进行操作和运算的,而我们学的指令全都是用字节来介绍的,如字节的移动、加法与减法、逻辑运算和移位等。用字节处理一些数学问题,例如,控制冰箱的温度、电视的音量等,比较直观,但用它控制一些布尔事件就不方便了,如开关的开闭、灯的亮灭等。工业中有很多场合需要处理这类开关输出、继电器吸合等事件,用字节处理显得有些麻烦,所以在 MCS-51 系列单片机中的布尔处理器就应运而生了。它用进位位 CY(程序状态字 PSW.7)作为累加器 C,用 RAM 和 SFR 内的位寻址区的位单位作为操作数,通过 C 完成位的传送和逻辑运算。子集共有 17 条指令,包括布尔变量的传送、逻辑运算、控制程序转移等指令。

MCS-51 系列单片机的布尔处理器可以使程序设计变得更加方便灵活,避免不必要的大量冗余数据传送及屏蔽字节等操作。

1. 位传送指令

```
MOV C, BIT
MOV BIT, C
```

这组指令的功能是实现位累加器(CY)和其他位地址之间的数据传递。

例 5.19　分析如下程序执行结果。

```
MOV C, 05H
MOV P1.0, C
```

分析:执行步骤依次为(30H).5→(CY),(CY).5→P1.0,结果为(30H).5→P1.0,注意变量传送必须通过 C 进行。

2. 位修正指令

1) 基本数据类型

```
CLR C                      ;使 CY = 0
CLR bit                    ;使指令的位地址等于 0
```

例如,语句"CLR P1.0",功能为使 P1.0 变为 0。

2) 位置 1 指令

```
SETB C:   使 CY = 1
SETB bit:使指定的位地址等于 1
```

例如,语句"SETB P1.0",使 P1.0 变为 1。

3) 位取反指令

```
CPL C                      ;使 CY 的值与原来的值相反,由 1 变为 0,由 0 变为 1
CPL bit                    ;指定位的值与原来的值相反,由 0 变为 1,由 1 变为 0
```

3. 位逻辑运算指令

```
ANL C, bit                ;CY 与指定的位地址的值相与,结果送回 CY
ANL C, /bit:              ;先将指定的位地址中的值取出后取反,再和 CY 相与,结果送回 CY,但要
                          ;注意指定的位地址中的值本身并不发生变化
```

例 5.20 分析指令 ANL C, /P1.0 执行前后 CY 与 P1.0 的变化。

分析：设执行本指令前,若 CY=1,P1.0=1,则执行完本指令后 CY=0,且 P1.0=1。可用下列程序验证。

```
      ORG  0000H
      AJMP  START
      ORG 30H
START: MOV SP, #5FH
      MOV P1, #0FFH
      SETB C
      ANL C, /P1.0
      MOV P1.1, C          ;将结果传送 P1.1,结果应是 P1.1 上的灯亮,而 P1.0 上的灯灭
```

4. 位或指令

```
ORL C, bit
ORL C, /bit
```

这组指令的功能为：如果原来的布尔值为 1,则置位进位标志位 CY,否则保持原状态。bit 前的斜杠表示对其取反,取反后用作源操作数,但并不改变原来的值。例如,奇偶标志位初始时为 0,进位标志位也为 0,执行语句“ORL C, /P”后,C 修改为 1,但 P 仍为 0。

在实际应用过程中,布尔变量和位操作指令对只有两个状态的事物表示非常方便,如交通灯的亮灭(灯亮为 1,灯灭为 0)。

例 5.21 一个十字路口有两条交汇的甲和乙路口以及对应的两组交通灯,为简化控制程序,假设每一组交通灯只有红灯和绿灯,甲路口的红灯和绿灯控制位分别为 P1.0、P1.1,乙路口的红灯和绿灯控制位分别为 P2.0、P2.1,设计完成红绿灯控制程序。

解：依照题意,在程序执行前需要使所有灯熄灭,且灯为依次循环闪亮,需要用到循环程序。红绿灯的交替变化闪亮,这里需要用到延时子程序以实现灯的持续亮灭。

红绿灯控制程序如下。

```
    SETB  P1.0            ;初始化,甲路口红灯亮
    CLR   P1.1            ;甲路口绿灯灭
    CLR   P2.0            ;乙路口红灯灭
    SETB  P2.1            ;乙路口绿灯亮
LOOP:                     ;循环设计灯的亮灭
    CPL   P1.0            ;位取反,甲路口红灯灭
    LCALL DLAY100MS       ;延时 100ms
    CPL   P1.1            ;甲路口绿灯亮
    LCALL DLAY100MS
    CPL   P2.0            ;乙路口红灯亮
    LCALL DLAY100MS
    CPL   P2.1            ;乙路口绿灯灭
    JMP   LOOP
```

5.3　C51的运算及表达式

5.3.1　基本运算符

C语言中运算符和表达式数量之多，在高级语言中是少见的。正是丰富的运算符和表达式使得C语言的功能十分完善，这也是C语言的主要特点之一。

C语言的运算符不仅具有不同的优先级，而且还有一个特点，就是它的结合性。在表达式中，各运算量参与运算的先后顺序不仅要遵守运算符优先级别的规定，还要受运算符结合性的制约，以便确定是自左向右进行运算还是自右向左进行运算。这种结合性是其他高级语言的运算符所没有的，因此也增加了C语言的复杂性。

1. 运算符的种类

C语言的运算符可分为以下几类。

- **算术运算符**

算术运算符用于各类数值运算，包括加(+)、减(-)、乘(*)、除(/)、求余(或称模运算，%)、自增(++)和自减(--)共7种。

- **关系运算符**

关系运算符用于比较运算，包括大于(>)、小于(<)、等于(==)、大于或等于(>=)、小于或等于(<=)和不等于(!=)共6种。

- **逻辑运算符**

逻辑运算符用于逻辑运算，包括与(&&)、或(||)、非(!)共3种。

- **位操作运算符**

参与运算的量，按二进制位进行运算，包括位与(&)、位或(|)、位非(~)、位异或(^)、左移(<<)和右移(>>)共6种。

- **赋值运算符**

赋值运算符用于赋值运算，分为简单赋值(=)、复合算术赋值(+=、-=、*=、/=、%=)和复合位运算赋值(&=、|=、^=、>>=、<<=)3类共11种。

- **条件运算符**

条件运算符格式为(　?　:　)。这是一个三目运算符，也是C语言中唯一的三目运算符。用于条件求值。

- **逗号运算符**

逗号运算符用于把若干表达式组合成一个表达式(,)。

- **指针运算符**

指针运算符用于取内容(*)和取地址(&)两种运算。

- **求字节数运算符**

求字节数运算符用于计算数据类型所占的字节数(sizeof)。

- **特殊运算符**

特殊运算符有括号()、下标[]、成员(→、.)等。

2. 优先级和结合性

C 语言中,运算符的运算优先级共分为 15 级,1 级最高,15 级最低。在表达式中,优先级较高的先于优先级较低的进行运算。而在一个运算量两侧的运算符优先级相同时,则按运算符的结合性所规定的结合方向处理。C 语言中各运算符的结合性分为两种,即左结合性(自左至右)和右结合性(自右至左)。例如,算术运算符的结合性是自左至右,即先左后右。如有表达式"x－y＋z",则 y 应先与减号结合,执行 x－y 运算,然后再执行＋z 的运算。这种自左至右的结合方向称为"左结合性",而自右至左的结合方向称为"右结合性"。最典型的右结合性运算符是赋值运算符。如 x＝y＝z,由于"＝"的右结合性,应先执行 y＝z 再执行 x＝(y＝z)运算。C 语言运算符中有不少为右结合性,应注意区别以避免理解错误。

3. 算术运算符和算术表达式运算符

该类运算符主要有以下几类。

- 加法运算符(＋):加法运算符为双目运算符,即应有两个量参与加法运算,如 a＋b 及 4＋8 等,具有右结合性。
- 减法运算符(－):减法运算符为双目运算符。但"－"也可作为负值运算符,此时为单目运算,如 －x、－5 等具有左结合性。
- 乘法运算符(＊):双目运算,具有左结合性。
- 除法运算符(/):双目运算,具有左结合性,参与运算量均为整型时,结果也为整型,舍去小数。如果运算量中有一个是实型,则结果为双精度实型。
- 求余运算符(也称模运算符 a%b):双目运算,具有左结合性,要求参与运算的量均为整型。求余运算的结果等于两数相除后的余数。

4. 自增 1 和自减 1 运算符

自增 1 运算符记为"＋＋",其功能是使变量的值自增 1,自减 1 运算符记为"－－",其功能是使变量值自减 1。自增 1 和自减 1 运算符均为单目运算,都具有右结合性。可有以下几种形式:＋＋i 为 i 自增 1 后再参与其他运算,i＋＋反之,－－i 和 i－－同理。

例 5.22 给出下列自加自减运算的结果。

```
void main()
{
  int i = 8;
  printf("%d\n", ++i);
  printf("%d\n", --i);
  printf("%d\n", i++);
  printf("%d\n", i--);
}
```

解:结果依次为 9(i 先加再取值输出),8(i 先加减取值输出),8(i 先取值再加),9(i 先取值再减)。

5.3.2 算术表达式

算术表达式是由算术运算符和括号连接起来的式子,以下是算术表达式的例子。

```
a+b; (a * 2) / c; (x+r) * 8-(a+b) / 7; ++I;  sin(x)+sin(y);  (++i)-(j++)+(k--)
```

赋值运算符、赋值表达式及复合赋值符已经在第 2 章详细介绍过，这里再介绍另几种运算符。

1. 逗号运算符

C 语言中的逗号(,)也是一种运算符，称为逗号运算符。其功能是把两个表达式连接起来组成一个表达式，称为逗号表达式。其中用逗号分开的表达式的值分别结算，但整个表达式的值是最后一个表达式的值。

其一般形式为（表达式 1，表达式 2），其求值过程是分别求两个表达式的值，并以表达式 2 的值作为整个逗号表达式的值。

例 5.23　分析以下语句执行后 x 的结果。

```
int a = 1, b = 2, c = 3, x;
x = (a = a + 1, b = b + a, a + b);
```

解：运算结果为 x＝6，表达式中执行依次为 x＝2，b＝4，执行最后一个表达式后再将计算出的值赋予 x。

例 5.24　分析以下语句执行后 x 的结果。

```
int a1, a2, b = 2, c = 7, d = 5;
a1 = (++b, c--, d + 3);
a2 = ++b, c--, d + 3;
```

解：对于给 a1 赋值的代码，有 3 个表达式，用逗号分开，所以最终的值应该是最后一个表达式的值，也就是(d＋3)的值，为 8，所以 a1 的值为 8。

对于给 a2 赋值的代码，也有 3 个表达式，这时的 3 个表达式为 a2＝＋＋b，c－－，d＋3，这是因为赋值运算符比逗号运算符优先级高。最终表达式的值虽然也为 8，但第 2 行代码运算完时，b＝3，即第 3 行代码运行时，b 的值为 4，所以 a2＝4。

2. 三目运算符

主要用于条件求值。一般形式为：

```
表达式 1? 表达式 2: 表达式 3
```

其求值规则如下：如果表达式 1 的值为真，则以表达式 2 的值作为条件表达式的值，否则以表达式 2 的值作为整个条件表达式的值。条件表达式通常用于赋值语句之中。

例 5.25　分析以下语句执行后 x 的结果。

```
int a, b, c;
C = (a>b ? 6 : 9);
```

解：上面使用了三目运算符号，当 a 大于 b 时 c 的值为 6，当 a 不大于 b 时 c 的值为 9。

注意：当三目运算符嵌套使用时，应先理清三目运算符的运算规则及先后顺序，再依次计算每个表达式的结果后输出。

例如，语句“a＞b ? a : c＞d ? c : d”应理解为“a＞b ? a : (c＞d ? c:d)”。这也就是条件表达式嵌套的情形，即其中的表达式 3 又是一个条件表达式。

例 5.26　考试成绩高于 90 分的同学用 A 表示，60～89 分的用 B 表示，60 分以下的用 C 表示。试利用条件运算符的嵌套编写程序完成以上功能。

```
#include<stdio.h>
```

```
Int main(int argc, const char? * argv[])
{
  float grade;                                          //嵌套的三目运算,输出等级 ABC
  char level;
  printf("请输入同学的分数是:\n");
  scanf("%f", &grade);
  level = (grade>= 90)? 'A' : (grade>60 ? 'B' : 'C');     //此时需要一个字符 a 用来接收
  printf("该同学的成绩表示为:%c\n",a);
  return 0;
}
```

5.4 C 语言和汇编语言混合编程

在单片机应用系统开发中,目前主要使用 C 语言和汇编语言。C 语言和汇编语言各有优缺点。在稍大规模的嵌入式软件中(如含有 OS)大部分的代码都是用 C 编写的,主要是因为 C 语言的结构比较好,便于理解,有大量的支持库,编程效率高。C 语言中数据类型丰富,程序结构清晰,但是在执行速度、精确定时和控制硬件等方面不如汇编语言。

进行单片机编程时经常在 C 语言中嵌入汇编语言,但很多地方还是要用到汇编语言,例如,开机时硬件系统的 CPU 状态设定、中断使能、主频设定,以及 RAM 的控制参数及初始化。在一些精确延时的场合,在对性能非常敏感的代码块,在一些中断处理方面也需要嵌入汇编,因为一条汇编执行是可知的,单片机的运行是可以完全掌握的。而 C 指令执行虽然结果可知,但部分性能是不可知的,因为 C 编译器不一样,编译后的汇编也是不一样的。如果要在各方面都获得满意的结果,那么可以使用 C 语言与汇编语言的混合编程。

本节主要讨论 C 语言和汇编的混合编程,以及它们之间的函数调用。

5.4.1 混合编程的约定规则

在 C 程序和 ARM 汇编程序之间相互调用时必须遵守 ATPCS 规则(ARM-Thumb Procedure Call Standard,ARM-Thumb 过程调用标准),它规定了应用程序的函数如何分开编写和分开编译,最后将它们连接在一起,所以它实际上定义了一套有关过程(函数)调用者与被调用者之间的协议。ATPCS 规定了一些子程序间调用的基本规则、寄存器的使用规则、堆栈的使用规则和参数的传递规则等。

1. 寄存器的使用规则

(1) 子程序之间通过寄存器 R0～R3 来传递参数,当参数多于 4 个时,使用堆栈来传递参数。被调用的子程序在返回前无须恢复寄存器 R0～R3 中的值。

(2) 在子程序中,使用寄存器 R4～R11 保存局部变量,因此当进行子程序调用时要注意对这些寄存器的保存和恢复。在 Thumb 程序中,通常只能使用寄存器 R4～R7 来保存局部变量。R4～R11 可记作 V1～V8。

(3) 寄存器 R12 用作子程序间内部过程调用寄存器,用于保存堆栈指针 SP,当子程序返回时使用该寄存器出栈,记作 IP。

(4) 寄存器 R13 用作堆栈指针,记作 SP,在子程序中不能用作其他用途,它在进入子程序时的值和退出子程序的值必须相等。

(5) 寄存器 R14 称为链接寄存器,记作 LR。该寄存器用于保存子程序的返回地址。如果在子程序中保存了返回地址,寄存器 R14 则可以用作其他用途。

(6) 寄存器 R15 称为程序计数器,记作 PC,不能用作其他用途。

2. 堆栈的使用规则

ATPCS 规定堆栈采用满递减类型(Full Descending,FD),即堆栈通过减小存储器地址而向下增长,堆栈指针指向内含有效数据项的最低地址。对汇编程序来说,若目标文件中包含外部调用,则需满足如下两个条件:

(1) 外部接口的堆栈必须是 8 字节对齐的。

(2) 在汇编程序中使用 PRESERVE8 伪指令告诉连接器,本汇编程序数据是 8 字节对齐的。

3. 参数的传递规则

对于参数可变的子程序,当参数不超过 4 个时,可以使用寄存器 R0～R3 来传递参数;当参数超过 4 个时,还可以使用堆栈来传递参数。在传递参数时,将所有参数看作是存放在连续的内存单元的字数据。然后依次将各字数据传递到寄存器 R0～R3 中。入栈的顺序与参数传递顺序相反,即最后一个字数据先入栈。

当子程序参数个数固定,且结果为浮点数时,通过浮点运算部件的寄存器 F0、D0 或者 S0 返回。如果系统不包含浮点运算的硬件部件,那么浮点参数会通过相应的规则转换成整数参数(若没有浮点参数,则省略),然后依次将各字数据传送到寄存器 R0～R3 中。如果参数多于 4 个,将剩余的字数据传送堆栈中,入栈的顺序与参数顺序相反,即最后一个字数据先入栈。在参数传递时,将所有参数看作是存放在连续的内存子单元的字数据。

4. 子程序结果返回规则

(1) 结果为一个 32 位整数时,可以通过寄存器 R0 返回。

(2) 结果为一个 64 位整数时,可以通过寄存器 R0 和 R1 返回。

(3) 结果为一个浮点数时,可以通过浮点运算部件的寄存器 F0、D0 或 S0 返回。

(4) 结果为复合型浮点数(如复数)时,可以通过寄存器 F0～Fn 或 D0～Dn 来返回。

(5) 对于位数更多的结果,需要通过内存来传递。

5.4.2　在 C 语言中内嵌汇编

用 C 语言调用汇编语言程序时,被调用函数(汇编语言函数)要在调用函数(C 语言函数)所在的文件中说明。C 语言程序中使用 extern 关键字声明外部函数(声明要调用的汇编子程序)。要根据不同情况对函数名进行转换,见表 5-3。

表 5-3　函数名转换规则

函数首部	符号名	说　明
void func(void)	FUNC	无参数传递或不含寄存器参数的函数名不作改变
voidfunc(char)	_FUNC	带寄存器参数的函数名加“_”前缀
voidfunc(void) reentrant	_? FUNC	可重入函数前加“_?”前缀

例 5.27 在 C 语言中调用汇编函数实例。

C 程序：

```
#include<stdio.h>
extern void strcopy(char *d, const char *s)
int main()
{
    const char *srcstr = "First";
    char *dststr = "Second";
    strcopy(dststr,scrstr);
    return 0;
}
```

调用的汇编程序如下：

```
AREA SCopy, CODE, READONLY
ENTEY
EXPORT strcopy
strcopy
LDRB R2, [R1], #1
STRM R2,[R0], #1
BEN strcopy
MOV PC, LR
END
```

在 C 语言中嵌入汇编代码的步骤如下：

(1) 采用汇编代码命令加入汇编代码。

```
#pragma ASM                    ;汇编代码开始
; Assembler Code Here
#pragma ENDASM                 ;汇编代码结束
```

(2) 改变编译选择。

在 Project 窗口中包含源文件上，右击，在弹出的快捷菜单中选择 Options for 命令，单击右边的 Generate Assembler SRC File 和 Assemble SRC File，使检查框由灰变黑(有效状态)。

根据选择的编译模式，把对应的库文件(如 Small 模式时，Keil\C51\Lib\C51S. Lib)加入工程，该文件必须作为工程的最后文件。

(3) 编译生成目标代码。

例 5.28 在 C 语言中嵌入汇编代码实例。

```
#include<reg51.h>
void main()
{
#pragma asm
MOV R7, #10
:MOV R6, #20
```

```
DJNZ R6, $
DJNZ R7,DEL
#pragma endasm
}
```

5.4.3　在汇编程序中内嵌 C 语言函数

在汇编函数中要调用 C 语言的子函数，应该根据 C 语言函数原型所要求的参数类型，分别将参数压入堆栈后，再调用 C 语言函数。调用结束后还须再进行弹栈，以回复调用 C 语言函数前的堆栈指针。

汇编程序的书写要遵循 ATPCS 规则，以保证程序调用时参数正确传递。在汇编程序中调用 C 语言函数的方法为：首先在汇编程序中使用 IMPORT 伪指令事先声明将要调用的 C 语言函数，使其他程序可以调用该子程序，在 C 语言程序中使用。

例 5.29　在汇编程序中调用 C 语言函数的实例。

C 语言函数代码：

```
int cFun(int a, int b, int c)
{
  return (a + b + c)
}
```

调用 C 语言函数的汇编程序代码如下：

```
AREA asmfile, CODE, READONLY
IMPORT cFun
ENTRY
MOV R0, #11
MOV R1, #22
MOV R2, #33
BL cFun
END
```

5.5　简单计算器实验

1. 实验内容

编写程序实现以下功能：一个具有个位数的加、减、乘、除运算功能的简易计算器，并将结果显示在显示器上。

2. 注意事项

(1) 编辑程序并保存，注意对于汇编语言，文件扩展名为 .ASM；对于 C 语言，文件扩展名为 .C。

(2) “有键”判断的条件，需进行延时消抖。

(3) 程序需要具备键盘扫描功能，能知道输入数据；具备显示功能，可以显示结果及设定；具有容错功能，当输入数据大于设定时，能提示或禁止输入；具有拆字和转换编码功

能，能将运算结果转换为显示编码；具有运算功能，能进行简单运算。

3. 汇编程序示例

```
        ORG     0000H
        LJMP    INIT
        ORG     0030H
        LSA     EQU P2.2
        LSB     EQU P2.3
        LSC     EQU P2.4
        ;************** 主程序 **************;
INIT:MOV    A, #0FEH
MAIN:MOV    P3, #0FFH
    CLR     LSA                     ;显示第一位
    CLR     LSB
    CLR     LSC
    MOV     P0, A
    JNB     P3.1, K1
    JNB     P3.0, K2
    JNB     P3.2, K3
    JNB     P3.3, K4
    SJMP    MAIN

    ;********* K1 按键按下处理程序 ********;
K1:ACALL    DELAY10MS               ;延时消抖
    JB      P3.1, MAIN
    MOV     A, 03FH
    MOV     R2, #030H
KEY1_UP:ACALL  DELAY10MS
    DJNZ    R2, KEY1_UP
    LJMP    MAIN

    ;********* K2 按键按下处理程序 *********;
K2:ACALL    DELAY10MS
    JB      P3.0, MAIN
    MOV     A, #06FH
    MOV     R2, #030H
KEY2_UP:ACALL  DELAY10MS
    DJNZ    R2, KEY2_UP
    LJMP    MAIN

    ;********** K3 按键按下处理程序 *********;
K3:ACALL    DELAY10MS
    JB      P3.2, MAIN
    MOV     R2, #030H
KEY3_UP:ACALL  DELAY10MS
    DJNZ    R2, KEY3_UP
    CJNE    A, 03FH, NEXT_K3        ;如果不等于 00H,那么转去检测是否等于 FFH
NEXT3: MOV  A, #06FH
    LJMP    MAIN
NEXT_K3:CJNEA, 0FFH, RL3            ;也不等于 FFH,那么执行左移任务
```

```
    SJMP    NEXT3                    ;如果等于 FFH,那么跳回赋值 FEH
RL3:ADDA
    LJMP    MAIN

    ;********** K4 按键按下处理程序 **********;
K4:ACALL    DELAY10MS
    JB      P3.3, MAIN
    MOV     R2, #030H
KEY4_UP:ACALL  DELAY10MS
    DJNZ    R2, KEY4_UP
    CJNE    A, 03FH, NEXT_K4
NEX4:MOV    A, #06FH
    LJMP    MAIN
NEXT_K4:CJNEA, 0FFH, RR4
    SJMP    NEX4
RR4:SUBA
    LJMP    MAIN

    ;********** 延时程序 ***********;
DELAY10MS:  MOV R6, #015H
DE1:MOV     R7, #0F8H
DE2:DJNZ    R7, DE2
    DJNZ    R6, DE1
    RET

    END
```

5.6　习题

1. 设(A)＝58H,(R0)＝20H,(R1)＝3DH,(R5)＝7AH,(40H)＝2CH,(20H)＝0ECH,请写出下列指令独立执行之后有关寄存器和存储单元的内容,并给出 CY、AC 和 OV 的值。

(1) MOV A, @R0

(2) ANL 40H, 04FH

(3) ADD A, R5

(4) DEC R5

2. 有两个 16 位无符号数,分别存放在内部 RAM 的 40H、41H 和 50H、51H(低位数在低地址单元),请写出:

(1) 两个 16 位数的加法程序,和存于 R3、R1 中(R3 中存放高位数)。

(2) 两个 16 位数的减法程序,差存于 R5、R7 中(R5 中存放高位数)。

3. 在内部 RAM 40H 单元开始的区域内存有 10 个单字节十进制数(压缩型 BCD 码),试求其累加和,并将结果存放在内部 RAM 20H～23H 单元中。

4. 设内部 RAM 40H 单元存放有 8 位无符号的被除数,42H 单元存放有 8 位无符号的

除数，将两数相除之后的商存入 43H 单元，余数存入 45H 单元。

5. 设内部 RAM 40H、41H 单元存放有 16 位无符号的被除数，42H 单元存放有 8 位无符号的除数，将两数相除之后的商存入 43H、44H 单元，余数存入 45H 单元。

6. 设无符号双字节被乘数存放在 40H 和 41H 单元(40H 中为高位数)，乘数存放在 42H 和 43H 单元(43H 中为高位数)，将两数之积存入 44H、45H、46H、47H 中。

7. 给出以下运算表达式的结果。

(1) 整型变量 m、n、a、b、c、d 均为 1，执行语句“(m=a>b) && (n=c>d)”后 m、n 的值是多少？

(2) 设整型变量 i 值为 2，表达式“(++i)+(++i)+(++i)”的结果是什么？

(3) 设 a=1，b=2，c=3，d=4，则表达式“a<b? a:c<d? a:d”的结果是什么？

(4) 设 a=3，则执行了语句“a+=a-=a*=a”后，变量 a 的值是多少？