第 3 章　汇编语言程序设计

计算机程序设计语言分为机器语言、汇编语言和高级语言。高级语言非常接近人类自然语言，是通用于各种计算机、给出问题求解过程的程序设计语言。但是，高级语言程序不能直接控制计算机的硬件，并且执行速度慢，占用的存储空间大。汇编语言和机器语言都是面向某型号计算机的程序设计语言。

汇编语言是一种符号化的机器语言（符号语言），与机器语言是一一对应的。汇编语言程序可以直接控制计算机的硬件和 I/O 接口，实时性能好，并且执行速度快，占用的存储空间小，运行效率高。所以，汇编语言常被用来编写计算机系统程序、实时通信程序、实时控制程序等。

本章主要介绍汇编语言的指令系统、汇编语言程序，以及汇编语言程序设计。

3.1　汇编语言的指令系统

众所周知，当计算机要解决一个计算问题或者处理信息问题时，必须把解决问题的步骤转换成计算机能识别和执行的操作命令。

3.1.1　指令和指令系统

1．指令

计算机的指令是根据 CPU 硬件结构特点设计的、能直接执行的基本操作命令。一条指令对应计算机一条基本操作，如加、减、传送、移位等。

指令由操作码（OP）和操作数（OD）两部分组成。操作码是指令执行的操作功能，操作数是指令操作的数据（操作对象）。

机器指令是一串由 0 和 1 组成的二进制编码。机器指令由于难理解、难记忆、易出错，通常用一些助记符号来描述，这些助记符号被称为汇编语言符号指令，简称指令语句。所以，指令语句是一种符号化的机器指令，与机器指令是一一对应的。指令语句的操作码用英文单词的缩写描述，如传送指令操作码用“MOV”描述，加法指令操作码用“ADD”描述等；指令语句的操作数用操作对象存放的地方（寻址方式）描述。

2．指令系统

能直接执行的全部指令的集合称为计算机的指令系统。实际上，一个计算机的全部指令，加上不同的寻址方式，再加上不同的数据形式（如字节、字、双字等）的组合，可构成上千种

基本操作命令。由此可见，指令系统可以体现计算机的性能。

指令系统是计算机硬件和软件之间的桥梁，也是汇编语言程序设计的基础。

3.1.2 8086/8088 指令语句

1．8086/8088 指令语句的格式

8086/8088 指令语句由标号、操作符、操作数和注释 4 项组成，其格式如下：

[< 标号 >:] < 操作符 > [< 操作数 >] [;< 注释 >]

其中，带方括号的为可选项，可根据需要取舍。各项之间用空格或 Tab 键符分隔。

标号项是一个自定义的、以“:”结束的符号串，表示该指令语句在程序中的地址。通常在需要表明转移到此处时给出标号描述。

操作符项是指令的功能名，是系统提供的该指令操作的助记符，为指令语句的关键字，必须记住，并要正确使用。

操作数项是指令语句的操作对象。操作数可以是操作数据，也可以是转移地址。操作数根据不同指令有 0 个（无）操作数、1 个（单）操作数和 2 个（双）操作数之分。如果是双操作数，操作数之间必须用“,”分隔。

注释项（开始于“;”）是说明几条指令语句或一段程序功能的文字信息。

2．8086/8088 寻址方式

微机的寻址方式是指执行指令的操作数（操作对象）存放的地方——地址。8086/8088 的操作数可以在指令中直接给出，称为立即数，也可以指明存放的寄存器，或存储器（内存），或 I/O 接口，甚至可以约定存放的地方，即隐含寻址。

由于 8086/8088 指令的操作数（操作对象）可以是操作数据，也可以是转移地址，所以其寻址方式可分为与操作数据有关的寻址方式和与转移地址有关的寻址方式两大类。本节只介绍与操作数据有关的寻址方式，与转移地址有关的寻址方式将在 3.3.2 节中介绍。

与操作数据有关的寻址方式包括立即数寻址方式、寄存器寻址方式和存储器寻址方式，其中存储器寻址方式又分为直接寻址方式、寄存器间接寻址方式、寄存器相对寻址方式、基址变址寻址方式、基址变址相对寻址方式。故与操作数据有关的寻址方式共有 7 种。

为了能举例说明与操作数据有关的寻址方式，这里先简单介绍一条数据传送指令——MOV 指令。MOV 指令有 2 个（双）操作数，分别称为源操作数和目的操作数，其功能是把源操作数传送到目的操作数，数据类型有字节（8 位）数和字（16 位）数两种。MOV 指令的格式如下：

MOV < 目的操作数 >,< 源操作数 >

（1）立即数寻址方式：操作数以常量形式（立即数）直接在指令中给出。

例如，

```
MOV     CX, 9                   ;CX ← 9
MOV     AX, 5807H               ;AX ← 5807H
MOV     AL, 42H                 ;AL ← 42H
MOV     AH, 11010011B           ;AH ← 11010011B（0D3H）
MOV     AL, 1000                ;错误，1000 超过了字节数的范围
```

注意，立即数只能作为 MOV 指令中的源操作数。

（2）寄存器寻址方式：操作数存放在一个字节/字寄存器中。

上例中的所有目的操作数的寻址方式都是寄存器寻址方式。源操作数和目的操作数的寻址方式都可以是寄存器寻址方式。

例如，

```
MOV     AX, CX          ;AX ← CX
MOV     BL, AL          ;BL ← AL
MOV     AX, CL          ;错误，寄存器类型不匹配
```

注意，由于 CS:IP 是由 DOS 控制的，IP 寄存器不能用，CS 寄存器只可“读”，不可“写”，即不能改变其内容。

（3）存储器寻址方式（5 种）：操作数存放在存储器中。

存储器操作数是用逻辑地址，即< 段址 >:< 偏移地址 >描述的。8086/8088 的地址加法器自动形成 20 位的物理地址：

$$\text{物理地址} = \langle \text{段址} \rangle \times 16 + \langle \text{偏移地址} \rangle$$

存储器操作数的段址存放在段寄存器中，一般是隐含规定的。存储器操作数的偏移地址（EA）有直接寻址、寄存器间接寻址、寄存器相对寻址、基址变址寻址、基址变址相对寻址 5 种寻址方式。

① 直接寻址方式——指令中直接给出操作数的偏移地址（EA）。

例如，

```
MOV     AL, [1000H]     ;(DS : 1000H)的字节数→AL
MOV     AX, [1000H]     ;(DS : 1000H)的字数→AX
MOV     [2000H], BX     ;BX →(DS : 2000H)
```

又如，

```
N2=1000H                ;伪指令定义符号数据 N2=1000H
MOV     AX, N2          ;N2 是立即数寻址方式
MOV     AX, [N2]        ;[N2]，即[1000H]，是直接寻址方式
```

注意：存储器寻址方式的描述一定要用方括号标明；如果有 2 个操作数，则这 2 个操作数的寻址方式不可以都是存储器寻址方式。

② 寄存器间接寻址方式——用一个 16 位寄存器存放操作数偏移地址（EA）。

EA=基址/变址寄存器数据

存储器寻址方式中使用的寄存器，只能是 BX、BP、SI、DI 中的一个，其中 BX、BP 为基址寄存器，SI、DI 为变址寄存器。

例如，

```
MOV     AX, [BX]        ;(DS : BX)的字数→AX
MOV     AX, [CX]        ;错误，CX 寄存器不能用于存储器寻址
```

又如，

```
MOV     AX, SI          ;SI 是寄存器寻址方式
MOV     [SI], AX        ;[SI]是寄存器间接寻址方式
```

③ 寄存器相对寻址方式——偏移地址（EA）是寄存器数据与位移量之和。

EA=(基址/变址寄存器数据)+< 位移量 >

位移量是相对于基址/变址寄存器数据的一个 8 位或 16 位有符号数（补码）。

例如，

```
MOV     AX, [BX-100]        ;(DS :(BX-100))的字数→AX
MOV     [BP+2], BX          ;BX →(SS :(BP+2))
```

④ 基址变址寻址方式——偏移地址（EA）是基址寄存器数据与变址寄存器数据之和。

EA=(基址寄存器数据)+(变址寄存器数据)

进行基址变址寻址必须将 BX 和 BP 中的一个寄存器与 SI 和 DI 中的一个寄存器组合。

例如，

```
MOV     AX, [BX+SI]         ;(DS :(BX+SI))的字数→AX
MOV     [BP+SI], BX         ;BX →(SS :(BP+SI))
MOV     AX, [SI+DI]         ;错误，两个变址寄存器不能组合寻址
```

⑤ 基址变址相对寻址方式——偏移地址（EA）是基址寄存器数据、变址寄存器数据、位移量三者之和。

EA=(基址寄存器数据)+(变址寄存器数据)+< 位移量 >

例如，

```
MOV     AL, [BX+SI+10]      ;(DS :(BX+SI+10))的字节数→AL
MOV     [BP+DI-6], CX       ;CX →(SS :(BP+DI-6))
```

3．存储器寻址方式的段寄存器

存储器寻址方式中段寄存器的隐含规定：如果直接寻址，或者使用 BX、SI、DI 中某个寄存器间接寻址，那么段址均取自 DS 段寄存器；如果使用 BP 寄存器间接寻址，那么段址取自 SS 段寄存器。

除按照上述隐含规定操作以外，还可使用换段前缀方法改变隐含规定中的段寄存器。

例如，

```
MOV     AX, [1000H]         ;直接寻址方式，段寄存器是 DS
MOV     AX, [BP]            ;寄存器间接寻址方式，段寄存器是 SS
MOV     [BX+10], AX         ;寄存器相对寻址方式，段寄存器是 DS
MOV     ES: [BX+10], AX     ;寄存器相对寻址方式，段寄存器是 ES
MOV     AX, SS: [BX+SI]     ;基址变址寻址方式，段寄存器是 SS
```

3.1.3 8086/8088 指令系统

80x86 系列高档微机的指令系统完全兼容 8086/8088 的指令，所以 8086/8088 指令系统是 80x86 系列微机指令系统的基础。

8086/8088 指令系统有 133 条指令，按功能分为数据传送类指令、算术运算类指令、逻辑运算和移位类指令、处理器控制类指令、控制转移类指令，以及串操作类指令。

本节主要介绍数据传送类指令、算术运算类指令、逻辑运算和移位类指令及处理器控制类指令。控制转移类指令将在 3.3.2 节、3.3.3 节和 3.3.4 节中介绍。对于串操作类指令，本书不做介绍。

下面从格式、操作、操作数寻址方式 3 个方面，分类给出每条指令语句的描述。为了描述的简约性，数据传送方向用“→”标明，操作数的类别和寻址方式用英文词的缩写符号标明。例如，

dst（目的操作数）　src（源操作数）　opr（操作数）　lab（标号）
imm（立即数）　reg（寄存器）　segreg（段寄存器）　mem（存储器）

1．数据传送类指令

数据传送类指令共有 14 条，如表 3.1 所示。

表 3.1　数据传送类指令简表

指令符	功　能	指令符	功　能	指令符	功　能
MOV	数据传送	PUSH	压入堆栈	POP	弹出堆栈
XCHG	数据交换	XLAT	查表换码	LES	取偏移地址和 ES
LEA	取偏移地址	LDS	取偏移地址和 DS		
PUSHF	标志寄存器压入栈	POPF	标志寄存器弹出栈		
LAHF	标志寄存器低 8 位→AH	SAHF	AH→标志寄存器低 8 位		
IN	端口输入（读）	OUT	端口输出（写）		

数据传送类指令的数据类型是字节和字，绝大多数是双操作数，两个操作数类型必须一致。数据传送类指令的寻址方式与 MOV 指令的寻址方式基本上相同，除 POPF 和 SAHF 以外，数据传送类指令的执行均不影响标志位。

1）数据传送指令

格式：MOV　dst, src

操作：dst ← src

操作数寻址方式（dst、src 寻址配对有以下组合关系）：

dst	src
reg	reg/ mem/ imm / segreg
mem	reg/ imm/ segreg
segreg	reg/ mem

例如，

```
MOV     BX, 1000H          ;BX=2000H
MOV     AL, [1000H]        ;AL=(DS:1000H)
MOV     [2000H], [BX]      ;错误，源/目的操作数不能都是存储器寻址方式
MOV     DS, 2000H          ;错误，立即数不能直接传送给段寄存器
```

上述错误语句的功能可以用下列语句实现：

```
MOV     AX, 2000H          ;AX= 2000H
MOV     DS, AX             ;AX→DS，即 DS= 2000H
```

2）堆栈操作指令

堆栈是一个“先进后出”的内存数据区。堆栈的地址指针是 SS：SP，始终指向堆栈栈顶单元。堆栈操作是指从堆栈栈顶压入/弹出数据，压入/弹出的数据必须是字类型数据。

① 压入堆栈（入栈）指令。

格式：PUSH　　src

操作：SP – 2 → SP

　　src → (SS : SP)

操作数寻址方式：src = reg / segreg / mem

② 弹出堆栈（出栈）指令。

格式：POP　　dst

操作：(SS : SP) → dst

　　SP + 2 → SP

操作数寻址方式：同 PUSH 指令。

例如，

```
PUSH    AX                  ;AX→(SS : SP)
PUSH    [BX]                ;(DS : BX)→(SS : SP)
POP     CX                  ;(SS : SP)→CX
PUSH    CL                  ;错误，堆栈操作数据必须是字类型数据
POP     200                 ;错误，立即数不能是堆栈操作数据
```

3）数据交换指令

格式：XCHG　opr1, opr2

操作：opr1 ↔ opr2

操作数寻址方式：opr1= reg / mem　　opr2= reg / mem

例如，

```
XCHG    [2000H], [BX]       ;错误，两个存储器数据不可以直接交换
```

上述错误语句的功能可以用下列语句实现：

```
MOV     AX, [2000H]         ;(DS : 2000H)→AX
XCHG    AX, [BX]            ;AX 和(DS : BX)交换，即 AX =(DS : BX)
MOV     [2000H], AX         ;AX→(DS : 2000H)
```

4）查表换码指令

格式：XLAT

操作：AL ← (DS :(BX+AL))

操作数寻址方式：AL 是隐含寄存器寻址，(BX+AL)是隐含存储器寻址

数据表最大容量为 256 字节，BX 是数据表头的偏移地址（EA），AL 是距离数据表头的位移量（0～255）。例如，

```
MEM   DB    'ABCDEFGHIJKLMNOPQRSTUVWXYZ'    ;定义 MEM 数据表
      MOV   BX, OFFSET MEM      ;BX 取 MEM 数据表头的 EA
      MOV   AL, 2               ;AL= 2
      XLAT                      ;AL= 43H（‘C’的 ASCII 码值）
```

5）地址传送指令

① 偏移地址传送指令。

格式：LEA　dst,src

操作：src 的偏移地址（EA）→ dst

操作数寻址方式：dst = reg　　src = mem

例如，

```
LEA     BX, [2080H]          ;BX = 2080H
```

② 段址和偏移地址传送指令。

格式：LDS　dst,src

　　LES　dst,src

操作：(src) → dst

　　(src+2) → DS / ES

操作数寻址方式：同 LEA 指令

地址传送指令的 src 的寻址方式必须是存储器寻址方式。LDS、LES 指令的操作数是内存双字（4 字节）数据。例如，

```
LEA     BX,[2080H]           ;BX = 2080H
LDS     BX,[2080H]           ;BX =(DS: 2080H)，DS =(DS: 2082H)
```

上面 XLAT 指令的例子，也可以用如下语句来实现：

```
MEM  DB     'ABCDEFGHIJKLMNOPQRSTUVWXYZ'     ;定义 MEM 数据表
     LEA    BX, MEM              ;BX 取 MEM 数据表头的 EA
     MOV    AL, [BX+2]           ;AL= 43H（'C'的 ASCII 码值）
```

6）标志（Flag）寄存器传送指令

① 标志寄存器入/出栈。

格式：PUSHF　　;标志寄存器压入堆栈

　　POPF　　;标志寄存器弹出堆栈

② 标志寄存器低位字节传送。

格式：LAHF　　;标志寄存器低 8 位 → AH

　　SAHF　　;AH → 标志寄存器低 8 位

标志寄存器传送指令的操作数，即标志寄存器和 AH 寄存器是隐含寻址的。

7）I/O 端口数据传送指令

格式：IN　AL/AX,<端口地址>　;读输入端口数据（字节/字）

　　OUT　<端口地址>,AL/AX　;写输出端口数据（字节/字）

I/O 端口数据传送指令的寄存器只能是 AL（字节数据）或 AX（字数据），端口地址是一个 16 位（0～0FFFFH）地址值。如果端口地址值是 8 位（0～0FFH）的，则可以直接给出；如果端口地址值是 16 位的，则必须用（也只能用）DX 寄存器间接给出。

例如，

```
IN      AL, 80H              ;读取 80H 端口的数据→AL
OUT     20H, AL              ;AL→送 20H 端口输出
MOV     DX, 100H             ;DX=100H
OUT     DX, AL               ;AL→送（DX）端口，即送到 100H 端口输出
IN      AL, [80H]            ;错误，端口寻址决不能加方括号
MOV     AL, [80H]            ;正确，[80H]是内存直接寻址方式，(DS:0080H)→AL
```

2．算术运算类指令

算术运算类指令可分为加法指令、减法指令、乘法指令、（整）除法指令和 BCD 码调整

指令，本节仅介绍加法、减法、乘法、除法的 14 条指令，如表 3.2 所示。

表 3.2　算术运算类指令简表

指 令 符	功　能	指 令 符	功　能	指 令 符	功　能
ADD	加法	ADC	进位加法	INC	加 1
SUB	减法	SBB	借位减法	DEC	减 1
CMP	比较	NEG	求补		
MUL	无符号乘法	IMUL	有符号乘法		
DIV	无符号除法	IDIV	有符号除法		
CBW	字节符号扩展	CWD	字符号扩展		

绝大多数算术运算类指令的操作数是双操作数，可为字节/字类型，寻址方式与 MOV 指令的寻址方式基本相同。算术运算类指令一般都是根据运算结果设置标志位（ZF，SF，CF，OF）的。

1）加法相关指令

① 加法指令。

格式：ADD　dst,src

操作：(dst)+(src) → dst

操作数寻址方式：dst = reg/mem　　src = reg/mem/imm

② 进位加法指令。

格式：ADC　dst,src

操作：(dst)+(src)+ CF → dst

操作数寻址方式：同 ADD 指令

③ 加 1 指令。

格式：INC　dst

操作：(dst)+ 1 → dst

操作数寻址方式：　dst = reg/ mem

2）减法相关指令

① 减法指令。

格式：SUB　dst,src

操作：(dst)−(src) → dst

操作数寻址方式：同 ADD 指令

② 借位减法指令。

格式：SBB　dst，src

操作：(dst)−(src)− CF → dst

操作数寻址方式：同 ADD 指令

③ 减 1 指令。

格式：DEC　dst

操作：(dst)−1→ dst

操作数寻址方式：同 INC 指令

④ 比较指令。

格式：CMP　dst,src

操作：(dst)−(src)（不取运算结果，仅根据减法运算结果设置标志位）

操作数寻址方式：同 ADD 指令

⑤ 求补指令。

格式：NEG　dst

操作：0−(dst) → dst　　;求(dst)的互补码

操作数寻址方式：同 INC 指令

例如，求 2 个 32 位（双字）数之和，即 12345678H + 80A7FD28H。

```
MOV     DX, 1234H
MOV     AX, 5678H           ;DX|AX= 12345678H
ADD     AX, 0FD28H
ADC     DX, 80A7H           ;DX|AX=12345678H + 80A7FD28H= 92DC53A0H
```

3）乘法指令

① 无符号数乘法指令。

② 有符号数乘法指令。

格式：MUL / IMUL　src

操作：如果 src 是字节数，则(AL)×(src) → AX（字，16 位）

　　如果 src 是字数，则(AX)×(src) → DX|AX（双字，32 位）

操作数寻址方式：src = reg/ mem

乘法指令的被乘数（AL 或 AX）、乘积（AX 或 DX|AX）是固定的，隐含寻址，只需要给出一个操作数，即乘数（src），并根据乘数的数据类型确定是字节乘法还是字乘法，字节乘法的乘积一定是字类型，字乘法的乘积一定是双字类型。例如，

```
MUL     AH                  ;无符号数 (AL)×(AH)→AX
MUL     BX                  ;无符号数 (AX)×(BX)→DX|AX
IMUL    CX                  ;有符号数 (AX)×(CX)→DX|AX
```

有符号数乘法指令和无符号数乘法指令的运行结果是不一样的。例如，

无符号数字节乘法：0FFH×1= 00FFH（255×1 = 255）。

有符号数字节乘法：0FFH×1= 0FFFFH（−1×1 = −1）。

例如，计算 31×(−4)，

```
MOV     AL, 31              ;AL= 31（1FH）
MOV     CL, −4              ;CL= −4（0FCH）
IMUL    CL                  ;AX= −124（0FF84H）
```

4）除法指令

① 无符号数除法指令。

② 有符号数除法指令。

格式：DIV / IDIV src

操作：如果 src 是字节数，则(AX)/(src) → AL（商）,AH（余数）

　　如果 src 是字数，则(DX|AX)/(src) → AX（商）,DX（余数）

操作数寻址方式：src = reg / mem

除法指令的被除数（AX 或 DX|AX）、商（AL 或 AX）和余数（AH 或 DX）是固定的，

隐含寻址，只需要给出一个操作数——除数（src）。例如，

```
DIV     BL                  ;无符号数 (AX) / (BL) → AL（商）,AH（余数）
IDIV    BX                  ;有符号数 (DX|AX) / (BX) → AX（商）,DX（余数）
```

除法运算可能出现两种错误情况：① 0 作为除数的错误；② 除法溢出错误，即“商”超出了规定的数值范围。如果 AX= 600，BL=2，则

```
DIV     BL                  ;错误，商为 300，超出字节数范围，属于除法溢出错误
```

有符号数除法的余数与被除数的符号相同。

如果 AX = 0010H（+16），BL= 0FDH（−3），则

```
IDIV    BL                  ;商 AL= 0FBH（−5），余数 AH= 1
```

如果 AX=0FFF0H（−16），BL= 03H（3），则

```
IDIV    BL                  ;商 AL= 0FBH（−5），余数 AH= 0FFH（−1）
```

5）符号扩展指令

① 字节符号扩展指令。

② 字符号扩展指令。

```
格式：CBW                   ;AL 字节数符号扩展成 AX 字数
      CWD                   ;AX 字数符号扩展成 DX|AX 双字数
```

符号扩展指令的操作数 AL/AX/DX 是隐含寻址的。

如果 AL=56H，则 CBW 指令使 AX=0056H。

如果 AL=86H，则 CBW 指令使 AX=0FF86H。

符号扩展指令常用在 IDIV 指令之前，做有符号被除数的数据类型扩展，即 AL 扩展成 AX，或 AX 扩展成 DX|AX。

例如，计算（−104）除以 25。

```
MOV     AL, -104            ;AL= −104（98H）
CBW                         ;AL 扩展成 AX（0FF98H）
MOV     BL, 25              ;BL= 25
IDIV    BL                  ;AL= −4（商），AH= −4（余数）
```

3．逻辑运算和移位类指令

逻辑运算和移位类指令是以二进制数位为单位的“位操作”指令。逻辑运算类指令有 5 条（逻辑非、逻辑与、逻辑或、逻辑异或、位测试），移位类指令有 8 条（逻辑左/右移、算术左/右移、循环左/右移、带进位循环左/右移）。

绝大多数逻辑运算和移位类指令的操作数是双操作数，为字节/字类型，在多数情况下会影响标志位。逻辑运算类指令的寻址方式与算术运算类指令的寻址方式基本相同。

1）逻辑运算类指令

① 逻辑非指令。

② 逻辑与指令。

③ 逻辑或指令。

④ 逻辑异或指令。

⑤ 位测试指令。

逻辑运算类指令的格式和操作如表 3.3 所示。

表 3.3　逻辑运算类指令的格式和操作

逻辑运算类指令	格　　式	操　　作
逻辑非指令	NOT　dst	～(dst) → dst
逻辑与指令	AND　dst, src	(dst)∧(src) → dst
逻辑或指令	OR　　dst, src	(dst)∨(src) → dst
逻辑异或指令	XOR　dst, src	(dst)⊕(src) → dst
位测试指令	TEST　　dst,src	(dst)∧(src)（不取运算结果，仅根据逻辑运算结果设置标志位）

操作数寻址方式：dst = reg/mem　　src = imm/reg/mem

逻辑运算类指令的标志位 ZF 和 SF 分别取自结果，OF 和 CF 均为 0。

例如，

```
AND     AL, 50H          ;AL=(AL)∧50H
OR      AX, [8080H]      ;AX=(AX)∨(DS : 8080H)
AND     AL, 0FH          ;AL 高 4 位清 0，低 4 位保留
OR      AL, 0FH          ;AL 高 4 位保留，低 4 位置 1
XOR     AL, 0FH          ;AL 高 4 位保留，低 4 位取反
```

又如，

```
ADD     AL, 50H          ;AL=(AL)+ 50H
TEST    AL, 80H          ;AL∧80H，设置标志位（测试 AL 的 D7 位）
JNZ     P1               ;ZF 标志位“不等于 0”（D7 为 1），转 P1 标号
```

2）移位类指令

① 逻辑左/右移指令。

② 算术左/右移指令。

③ 循环左/右移指令。

④ 带进位循环左/右移指令。

格式：< 指令符 >　　dst,cnt

操作数寻址方式：dst 是移位的对象，dst = reg/mem

cnt 是移位的位数，cnt = 1/CL

移位类指令的标志位（ZF 和 SF）根据移位结果设置。CF，左移取自 dst 的最高位，右移取自 dst 的最低位。

移位类指令的操作图解如图 3.1 所示。

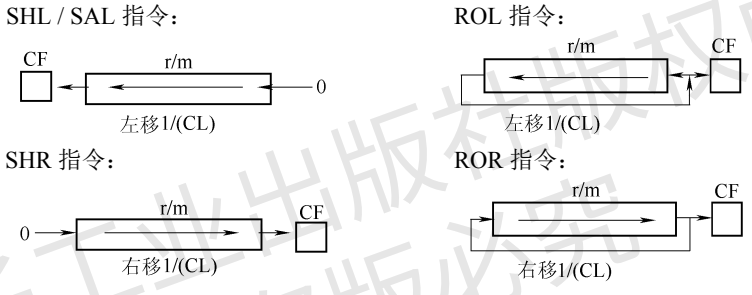

图 3.1　移位类指令的操作图解

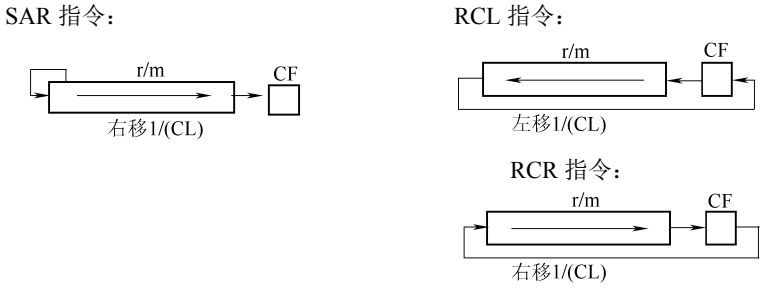

图 3.1　移位类指令的操作图解（续）

算术/逻辑左移一位，相当于“乘以 2”；算术/逻辑右移一位，相当于“除以 2”。所以，在做 2 的倍数的乘/除法时，常用移位类指令来实现。

例如，将双字（DX|AX）算术右移 2 位，即做有符号双字数（DX|AX）除以 4。

```
MOV     DX, 8FF9H
MOV     AX, 8000H          ;DX|AX= 8FF98000H
SAR     DX, 1
RCR     AX, 1              ;(DX|AX)除以 2
SAR     DX, 1
RCR     AX, 1              ;再除以 2，即除以 4，DX|AX= 0E3FE6000H
```

4．处理器控制类指令

处理器控制指令类可分为控制标志位（CF、DF、IF）设置指令和微处理器控制指令，如表 3.4 所示。

表 3.4　处理器控制类指令简表

指　令　符	功　　能	指　令　符	功　　能
CLC	CF=0	STC	CF=1
CMC	CF 取反	STD	DF=1
CLD	DF=0	STI	IF=1
CLI	IF=0	HLT	暂停（等外部中断）
NOP	空操作	WAIT	等待（TEST 信号）
LOCK	封锁总线		

3.2　汇编语言程序

8086/8088 汇编语言程序由执行（符号）指令语句、汇编指示（伪）指令语句、宏指令语句组成。

1）执行（符号）指令语句

执行指令语句是提供给汇编程序的、机器能直接执行的指令语句。

2）汇编指示（伪）指令语句

汇编指示指令语句是汇编程序自身提供的、对汇编过程起控制作用的指令语句。汇编指

示指令可用于分配数据存储单元、给标号赋值、控制汇编过程结束等。相对于执行指令，这类指令是“非执行”的，所以汇编指示指令常被称为伪指令。

3）宏指令语句

宏（Macro）指令语句是提供给汇编程序的、“功能宏大”的扩展指令语句。宏指令语句是有唯一命名、按一定语法规则定义、具有独立功能的一个指令语句序列。宏指令实际是功能扩展的“高级”指令。本书不对宏指令的使用进行介绍。

3.2.1　汇编语言程序的语句格式及汇编表达式

1．汇编语言程序的语句格式

汇编语言程序的执行指令和伪指令语句由 4 项组成，其格式如下。

[< 名字 >]　< 操作 >　< 操作数 >　[;< 注释 >]

1）名字项

名字项是自定义的一个标识符串，可以是标号名（结束于“:”）、符号常数名、变量名、段名、过程（子程序）名等。

名字项的标识符由字母 A～Z / a～z（大/小写字母通用）、数字 0～9、特殊字符“@”“_”“.”“?”等可打印字符组成。名字的标识符不能多于 31 个字符（超过的字符将被省略），第 1 个字符不能是数字符 0～9，“.”只能是第 1 个字符。

有效的名字项如 NEXT_A、LOOP1、START、FFH、.386、AP???等。

无效的名字项如 0FFH（十六进制数）、'ABC'（字符串数据）、2ab（数字打头）、S.asm（文件名）等。

2）操作项

操作项是执行指令名或伪指令名，是系统提供的指令操作的功能助记符，是指令语句的关键字，必须正确使用。

3）操作数项

操作数项是指令具体操作的对象，可以是操作数据，也可以是转移地址。如果是多个操作数，操作数之间用“,”分隔。操作数项可以用常数和汇编表达式描述。

在名字项中定义的名字，可以在操作数项中使用。如果是标号名、过程名，则可作为转移地址使用；如果是变量名，则可作为内存单元的偏移地址（EA）直接寻址使用；如果是符号常数名，则可作为立即数使用；如果是段名，则可作为段址立即数使用。

2．汇编表达式

操作数项可以用汇编表达式描述。汇编表达式是由常数（整数）、寄存器、标号、变量，以及规定的运算符组成的，能被汇编程序识别，并能计算出结果的操作数表达式。

汇编表达式根据其计算结果是数值，还是地址，可分为数值表达式和地址表达式两种。

1）数字常数

数字常数，即立即数，可以是直接给出的二进制数、十进制数、十六进制数、ASCII 码字符数值（用单引号括起来的字符）、名字项定义的符号常数等。

例如，11001010B、0A080H、255、‘A’（41H）、‘ok’（6F6BH）。

2）数值表达式

数值表达式由常数和数值运算符组成，计算结果是字节/字（整数）数据的表达式。

数值表达式中常使用的数值运算符如下。

① 算术运算符：+、−、*、/（整除）、MOD（取余）。

② 逻辑运算符：NOT（非）、AND（与）、OR（或）、XOR（异或）、SHL（左移）、SHR（右移）。

③ 关系运算符：EQ（=）、NE（≠）、GT（>）、GE（≥）、LT（<）、LE（≤），关系运算结果真值为−1，即全 1，假值为 0。

例如，19/7（=2），19 MOD 7（=5），80H OR 78H（=0F8H），88H SHL 2（=20H），

100 NE 102（= −1），100 GT 102（=0）。

3）地址表达式

地址表达式由常量、变量、标号、[BP]、[BX]、[SI]、[DI]，以及地址运算符组成，计算结果为内存地址值的表达式。

内存地址有段址、偏移地址、地址类型三种属性。地址类型分为 BYTE（字节）、WORD（字）、DWORD（双字）、NEAR（段内）、FAR（段间）5 种。

地址表达式中常用的地址运算符如下。

① 地址算术运算符：+、−（加/减偏移地址的相对值）。

② 属性定义运算符：<段寄存器>:（换段前缀）、PTR（类型运算）。

③ 分析运算符：SEG（取段址值）、OFFSET（取偏移地址值）、TYPE（取地址类型值，1/2/4/−1/−2）、LENGTH（取变量单元数）、SIZE（取变量总字节数）。

例如，

```
MEM   DB      10H, 20H, 30H, 40H      ;伪指令 DB 定义 MEM 字节变量
      MOV     AX, SEG MEM             ;AX= MEM 的段址
      MOV     DS, AX
      MOV     BX, OFFSET MEM          ;BX= MEM 的偏移地址 EA
      MOV     AL, [BX+2]              ;AL=30H
      MOV     AX, WORD PTR MEM        ;定义 MEM 为字变量类型，AX=2010H
      MOV     AL, TYPE MEM            ;取 MEM 变量类型值，AL=1
```

3.2.2 伪指令

汇编语言程序中指示汇编操作的指令称为伪指令。常用的汇编语言程序伪指令有符号定义伪指令、内存变量定义伪指令、段定义伪指令、过程（子程序）定义伪指令（在子程序设计中介绍）和程序模块定义伪指令 5 组，如表 3.5 所示。

表 3.5 常用的伪指令简表

伪 指 令 符	功 能	伪 指 令 符	功 能
EQU	符号等值	=	等号
DB	字节变量定义	DW	字变量定义
SEGMENT	段开始	ENDS	段结束
ASSUME	段说明	ORG	段内偏移地址指针$设置
PROC	过程（子程序）开始	ENDP	过程（子程序）结束
NAME	程序模块开始	END	程序模块结束

1．符号定义伪指令

1）符号等值伪指令

格式：<符号名>　EQU　<符号对象>

2）等号伪指令

格式：<符号名>＝<表达式>

注意：EQU 的符号名不可重复定义，符号对象可以是任何符号。

“=”的符号名可以重复定义，“=”的表达式只能是合法的汇编表达式。

例如，

```
Count      EQU      19                ;count =19
b=20
b=b+10                                ;b 重新定义，b = 30
d=(count+4)*2                         ;d = 46
fnum       EQU      123456H           ;正确，123456H 为符号对象
gnum=123456H                          ;错误，123456H 超过了 16 位二进制的数值范围
addr       EQU      ES:[BX+SI]
```

如果使用了 addr 符号定义的指令语句，则

```
           MOV      AX, addr          ;即汇编为 MOV    AX, ES:[BX+SI]
```

2．内存变量定义伪指令

内存变量定义伪指令有 DB（字节）、DW（字）、DD（双字）、DQ（8 字节）、DT（10 字节）等，其中常用的是 DB 和 DW。

格式：[<变量名>]　DB / DW / DD　<数据表>

操作：定义内存变量、类型（BYTE/ WORD/ DWORD），通过数据表分配内存单元，并存放初始数据。

数据表给出了顺序存放在内存单元中的数据，多个数据之间用“,”分隔。数据表的数据可以是数值表达式（8/16/32 位值）、地址表达式（16/32 位值）、ASCII 码值——8 位值、？和数据重复定义子句。

？——仅分配内存单元，不给出初始数据。

数据重复定义子句——重复定义一批数据，并可嵌套使用。

格式：　<重复次数>　DUP <数据表>

例如，

```
DA1        DB       'DATA SEGMENT'
DA2        EQU      $-DA1                ;DA2=12（$ 为当前偏移地址指针）
DA3        DB       6DH, 62, 15H, 28
DA4        DB       10 dup (0, 5 dup (1, 2), 0)
DA5        DB       '12345'
DA6        DW       7, 9, 298, 1967
DA7=DA6 -DA4                             ;DA7=125
DA8= $-DA4                               ;DA8=133
```

3．段定义伪指令

1）段开始伪指令

格式：<段名>　SEGMENT　[<段属性表>]

段属性表为可选项，一般在多模块的大规模程序中使用，这里不讨论。

2）段结束伪指令

格式：<段名>　ENDS

同一个段的 SEGMENT 语句与对应的 ENDS 语句的段名必须一致。

例如，定义了一个数据段，则

```
DATA    SEGMENT                    ;DATA 数据段开始
STR1    DB  100  dup  (?)          ;分配了 100 字节单元
DATA    ENDS                       ;DATA 数据段结束
```

3）段说明伪指令

格式：ASSUME　segreg: <段名>，…

segreg 是 CS、DS、ES 和 SS 中的一个，段名是已定义的段名。ASSUME 伪指令必须用在程序段中，说明已定义的各个段的类型（数据/代码/堆栈/附加段）。

4．程序模块定义伪指令

1）程序模块开始伪指令

格式：NAME　<模块名>

NAME 伪指令标识程序模块名，可省略不用。如果省略 NAME 伪指令，则将程序文件名作为程序模块名。

2）程序模块结束伪指令

格式：END　[<主程序入口>]

程序模块一定要以 END 伪指令结束，不可省略。一般，主程序模块结束要给出程序入口标号或主程序的过程名。

5．汇编语言程序的段结构

8086/8088 汇编语言程序是由一个程序模块组成的，而程序模块又是以标准的段结构形式组织的，所以段结构体现了程序模块化设计思想。用段定义、程序模块定义等伪指令可以组成汇编语言程序的段结构。下面给出一个 8086/8088 汇编语言程序段结构示例。该例有一个数据段和一个代码段。

```
DATA    SEGMENT                    ;定义 DATA 数据段
        〈 数据定义语句序列 〉
DATA    ENDS                       ;DATA 数据段结束
CODE    SEGMENT                    ;定义 CODE 代码段
        ASSUME  CS:CODE，DS:DATA
START:
        〈 指令语句序列 〉
        MOV     AX,4C00H
```

```
        INT     21H                 ;程序结束，返回 DOS
CODE    ENDS                        ;CODE 代码段结束
        END     START               ;汇编结束，START 为入口标号
```

3.2.3　汇编过程

汇编语言程序必须经过一个汇编过程，把汇编语言程序“翻译”成机器语言程序才能被机器执行。系统软件的编辑程序（EDIT 或记事本）、汇编程序（MASM 或 TASM）、连接程序（LINK 或 TLINK）、调试程序（DEBUG 或 TD）承担了把汇编语言程序“翻译”成机器语言程序的功能，其“翻译”过程就是汇编过程。由于汇编语言程序与机器语言程序是一一对应的，所以汇编过程是“一对一”的翻译过程。

汇编语言程序的设计是通过一系列有序操作步骤（编辑、汇编、连接、调试、运行）的上机过程来实现的。汇编语言程序设计的上机过程如图 3.2 所示。

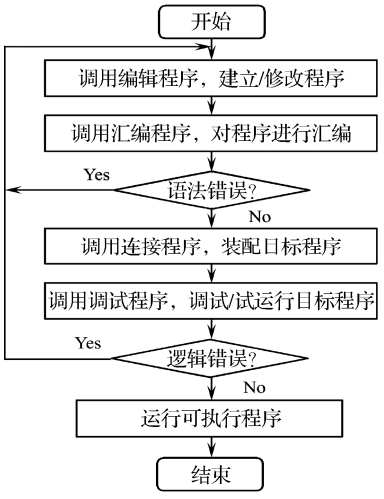

图 3.2　汇编语言程序设计的上机过程

汇编语言程序设计的上机过程必须有编辑程序、汇编程序、连接程序和调试程序等程序的支持。

① 用编辑程序建立扩展名为.ASM 汇编语言源程序文件。
② 用汇编程序将.ASM 文件汇编成扩展名为.OBJ 的二进制目标文件。
③ 用连接程序将.OBJ 文件连接成扩展名为.EXE 的可执行文件。
④ 在 DOS 环境下，可直接执行.EXE 程序，或者通过调试程序调试/试运行.EXE 程序。

3.3　汇编语言程序设计

汇编语言程序设计的步骤和高级语言程序设计的步骤一样，都包括分析问题、确定解决问题的方法（算法）、用流程图表示算法、编写程序、上机调试运行（汇编/连接/调试/运行）等。汇编语言程序采用结构化程序设计技术设计。结构化程序设计体现了程序结构定理化，

即采用三种基本结构（顺序结构、分支结构和循环结构）编写程序。

由顺序结构、分支结构、循环结构的任意组合和嵌套构成的结构化的程序只有一个入口和一个出口，这使得程序结构清晰、易于理解、易于修改、易于调试，充分显示了程序模块化设计的优点。

本节主要介绍 8086/8088 汇编语言的 3 种基本结构（顺序、分支、循环）的程序设计，子程序设计，以及系统提供的中断调用子程序的使用。

3.3.1 顺序程序

顺序结构一种是最基本的程序结构，也是最简单的汇编语言程序设计结构。它是一个完全按顺序逐条执行的指令序列，即指令指针线性增加，程序一直顺序往下执行，中途没有任何分支和出口。顺序程序设计的例子最多的是做数据运算。

【例 3.1】计算 $S=[8000-(X\times Y+Z)]/X$，其中 X、Y、Z、S 均是有符号数字变量。

```
;数据段
DATA    SEGMENT
   X    DW      600
   Y    DW      25
   Z    DW      -2000
   S    DW      ?, ?                ;存放商和余数
DATA    ENDS
;计算 S 算术表达式程序段
        MOV     AX, X
        IMUL    Y                   ;DX|AX =X×Y
        MOV     BX, AX
        MOV     CX, DX              ;CX|BX= X×Y
        MOV     AX, Z
        CWD                         ;Z 扩展成双字 DX|AX
        ADD     BX, AX
        ADC     CX, DX              ;CX|BX= X×Y+Z
        MOV     AX, 8000
        CWD                         ;字数 8000，扩展成双字 DX|AX
        SUB     AX, BX
        SBB     DX, CX              ;DX|AX= 8000-(X×Y+Z)
        IDIV    X                   ;[8000-(X×Y+Z)]/X
        MOV     S, AX               ;商（AX）存放到 S 单元
        MOV     S+2, DX             ;余数（DX）存放到 S+2 单元
```

【例 3.2】把一个字节的压缩 BCD 码，转换成两个字节的 ASCII 码。

```
DATA    SEGMENT
BCD     DB      48H
ASC     DB      ?, ?
```

```
DATA    ENDS
CODE    SEGMENT
        ASSUME  CS:CODE, DS:DATA
START:  MOV     AX, DATA
        MOV     DS, AX          ;设置 DS 数据段址
        MOV     AL, BCD
        MOV     BL, AL          ;AL、BL 取 BCD 数据（48H）
        AND     AL, 0FH         ;AL=08H
        OR      AL, 30H         ;AL=38H
        MOV     ASC, AL         ;转换的 38H 存放到 ASC 单元
        MOV     CL, 4
        SHR     BL, CL          ;BL=04H
        OR      BL, 30H         ;BL=34H
        MOV     ASC+1, BL       ;转换的 34H 存放到 ASC+1 单元
        MOV     AX, 4C00H
        INT     21H             ;返回 DOS
CODE    ENDS
        END     START
```

注意，本书中给出的汇编语言程序设计例题多数把段结构形式省略了，仅给出了相关数据变量定义和主要的程序功能段。

3.3.2　分支程序

一个实际应用程序，往往需要根据处理过程中出现的不同条件做出逻辑判断，从而决定程序的走向。每个逻辑判断有“是”“否”两种结果。程序必须在逻辑判断处出现两种走向的情况下做出选择，即程序出现了分支，构成了分支程序。

汇编语言程序设计可以通过以下两种情况实现分支结构。

① 使用条件转移指令，当指定的条件满足时，改变程序走向；否则，程序顺序执行。

② 利用影响状态标志位的指令，如算术运算类指令、逻辑运算和移位类指令、位测试类指令提供的标志位测试条件给出逻辑判断，确定是否转移。

1．有/无条件转移指令

有/无条件转移指令的操作数是指示转移的目标地址，寻址方式是地址寻址方式。因为要转移，所以必然涉及 CS 和 IP 的值。如果是段内转移，则仅涉及 IP 的值；如果是段间转移，则涉及 CS 和 IP 的值。

1）无条件转移指令

格式：JMP　dst

操作数寻址方式：dst = lab/reg/mem。

操作：计算或得到转移目标处的地址，转移到目标处。

无条件转移指令的操作数寻址方式有段内/段间直接转移、段内/段间间接转移，共 4 种。

① 段内/段间直接转移：直接给出段内/段间目标地址的标号。

格式：JMP　lab

段内/段间直接转移有以下 4 种形式。

```
JMP     lab                 ;段内直接转移，IP = lab 偏移地址（EA）
JMP     SHORT  lab          ;段内短转移，IP= IP + <8 位位移量>
JMP     NEAR PTR  lab       ;段内近转移，IP= IP + <16 位位移量>
JMP     FAR PTR  lab        ;段间直接转移，CS= lab 段址，IP= lab 偏移地址
```

例如，

```
JMP     PP1                 ;段内直接转移，IP = PP1 偏移地址
JMP     SHORT  PP2          ;段内短转移，IP = PP2 偏移地址
JMP     NEAR PTR  PP3       ;段内近转移，IP= PP3 偏移地址
JMP     FAR PTR  PP4        ;段间直接转移，CS = PP4 段址，IP = PP4 偏移地址
```

② 段内/段间间接转移：从寄存器或内存（字/双字）单元中得到目标地址。

格式：JMP　reg/mem

例如，

```
JMP     BX                  ;段内间接转移，IP =(BX)
JMP     WORD PTR [BX]       ;段内间接转移，IP =(DS:BX)
JMP     DWORD PTR [BX]      ;段间间接转移，IP =(DS:BX)，CS=(DS:(BX+2))
```

2）有条件转移指令

格式：Jxx　lab

操作数寻址方式：lab 为段内短转移，即 lab 偏移地址的位移量必须在字节数（8 位）范围内。

操作：IP = IP + <8 位位移量>（补码）。

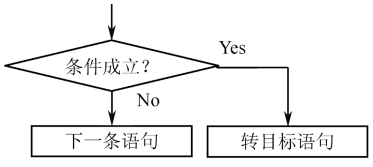

图 3.3　有条件转移指令的功能

有条件转移指令通常在设置或改变了标志位的指令（如算术运算类指令等）之后使用。有条件转移指令的功能如图 3.3 所示。

有条件转移指令共有 19 条。根据测试标志的情况，有条件转移指令可分为单个标志测试的条件转移指令、多个标志综合测试的条件转移指令。多个标志综合测试又分为有符号数比较测试和无符号数比较测试。有条件转移指令简表如表 3.6 所示。

表 3.6　有条件转移指令简表

分　类	助 记 符	测试标志条件	功　能
（单个标志测试）有条件转移指令	JZ（JE）	ZF=1	为零转移
	JNZ（JNE）	ZF=0	非零转移
	JC	CF=1	有进/借位转移
	JNC	CF=0	无进/借位转移
	JS	SF=1	符号位为 1 转移
	JNS	SF=0	符号位为 0 转移
	JO	OF=1	有溢出转移
	JNO	OF=0	无溢出转移
	JP	PF=1	“1”偶数个转移
	JNP	PF=0	“1”奇数个转移
	JCXZ	CX=0	CX=0 转移

续表

分　类	助 记 符	测试标志条件	功　能
（有符号数比较测试）有条件转移指令	JG	(SF⊕OF)∨ZF=0	比较结果为大于转移
	JGE	SF⊕OF=0	比较结果为大于或等于转移
	JL	SF⊕OF=1	比较结果为小于转移
	JLE	(SF⊕OF)∨ZF=1	比较结果为小于或等于转移
（无符号数比较测试）有条件转移指令	JA	CF∨ZF=0	比较结果为高于转移
	JAE	CF=0	比较结果为高于或等于转移
	JB	CF=1	比较结果为低于转移
	JBE	CF∨ZF=1	比较结果为低于或等于转移

2．分支程序的结构

1）二分支结构

汇编语言程序的二分支结构是用有条件转移指令实现的。有条件转移指令相当于高级语言的 if-then 语句，即

if ＜ 单条件 ＞ then ＜ 标号 ＞

【例 3.3】把有符号字节变量 *X* 和 *Y* 中的较大者送入变量 *Z*。

```
;字节变量 X 和 Y 比较程序段
    MOV     AL, X           ;AL= X
    CMP     AL, Y           ;对 AL（X）和 Y 进行比较
    JGE     YG              ;X≥Y，转 YG
    MOV     AL, Y           ;X<Y，AL=Y
YG: MOV     Z, AL           ;将较大数存放到 Z 单元
```

2）多分支结构

汇编语言程序的多分支结构相当于高级语言的 if-then-else 语句或 case 语句，即

if ＜ 复合条件 ＞ then ＜ 标号 1＞ else ＜ 标号 2＞

case ＜ 多选条件 ＞ do＜ 标号 1＞,＜ 标号 2＞,…,＜ 标号 n＞

汇编语言程序的多分支结构设计，是把 if-then-else 语句的复合条件项或者 case 语句的多选条件项，分解成多个单选条件项，用多个有条件转移指令的有效组合实现多分支结构。

3．分支程序设计

分支程序设计一般由“产生条件”“测试”“定向”“转移标号”四部分组成。特别要处理好分支结构的“入/出口”，即分支程序的转移标号（入口）和分支程序处理完成（出口），避免某个分支程序错误地进入另一个分支程序。在线性语句序列描述中，分支程序的“出口”常用 JMP 指令实现。

当然，二分支结构是最简单、最基本的分支程序结构，但实际应用的分支程序结构大多数是多分支结构。多分支程序一定要结构清晰、易读、易理解。

多（*n*）分支程序的设计一般采用逻辑分解法和地址跳转表法。

1）逻辑分解法

逻辑分解法是把 *n*（*n*>2）分支结构逐步分解成 *n*−1 个单分支结构。这种方法适用于程序分支数不多的应用场合。

【例 3.4】求 X 字节变量数据的符号函数（3 分支）。

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

```
;求字节变量符号函数程序段
        MOV     AL, X
        CMP     AL, 0           ;对 AL（X）与 0 进行比较
        JZ      ZERO            ;为 0，转 ZERO
        JS      NEGA            ;为负，转 NEGA
        MOV     AL, 1           ;为正，AL= 1
        JMP     OK              ;转公共出口 OK
ZERO:   MOV     AL, 0           ;AL= 0
        JMP     OK              ;转公共出口 OK
NEGA:   MOV     AL, 0FFH        ;AL=−1
OK:     MOV     Y, AL           ;符号函数值存放到 Y 单元
```

2）地址跳转表法

n 分支程序设计还可以根据分支号进行计算，选择某分支号的“地址跳转表”（连续存放 *n* 个分支转移的地址数据）地址，从中取得分支入口地址，进而实现转移。这种方法适用于程序分支数比较多的应用场合。

应用地址跳转表法是分支程序设计的一个技巧，主要任务是设计地址跳转表，以及给出由分支号得到分支入口地址的算法。

地址跳转表有多种形式，下面举例介绍最简单的偏移地址跳转表多分支程序设计。

【例 3.5】利用键盘输入一个成绩等级字符 A～D，显示对应的分数段字符串（4 分支）。

```
;数据变量定义
CHAR    DB      ?                       ;存放键盘输入的字符（ASCII 码）
TABL    DW      ENT_A, ENT_B, ENT_C, ENT_D      ;4 个入口标号的 EA 跳转表
;地址跳转表法程序段
P1:     ……                              ;利用键盘输入 A、B、C、D 字符，放入 CHAR 单元
        MOV     AL, CHAR
        SUB     AL, 41H                 ;转换成数字 0～3（分支号）
        CMP     AL, 3
        JA      P1                      ;>3（非 A～D），重新输入字符
        CMP     AL, 0
        JS      P1                      ;<0（非 A～D），重新输入字符
        MOV     AH, 0                   ;扩展成 AX
        SHL     AX, 1                   ;(AX)×2
        MOV     BX, AX
        JMP     TABL[BX]                ;转到对应的分支
        ……
```

```
ENT_A:  ……                      ;输出 “ A : 100～85 ! ” 字符串
        JMP     QUIT            ;转公共出口 QUIT
ENT_B:  ……                      ;输出“ B : 84～70 !” 字符串
        JMP     QUIT            ;转公共出口 QUIT
ENT_C:  ……                      ;输出 “ C : 69～60 !” 字符串
        JMP     QUIT            ;转公共出口 QUIT
ENT_D:  ……                      ;输出 “ D : 59～0 !” 字符串
QUIT:   MOV     AX, 4C00H
        INT     21H             ;返回 DOS
```

3.3.3　循环程序

在实际应用中，经常需要连续地重复执行一些相同的操作，这时适宜使用循环程序。

1．循环程序的结构

汇编语言的循环程序的结构一般由循环初始化、循环体（有限次重复执行的操作）、修改循环控制变量和循环结束判断四部分组成，循环程序的基本结构如图 3.4 所示。

① 循环初始化：它是循环程序的准备部分，设置循环程序的初始状态，如循环变量初值、地址指针初值、寄存器或存储单元初值等。循环初始化部分只执行一次。

② 循环体：它是循环程序中需要重复执行的部分，是循环结构的功能程序段。循环体至少执行一次，且必须有限次地执行。

③ 修改循环控制变量：它和循环体协调配合，对参加运算的数据或地址进行恰当的修改，以保证下一次循环操作能正确地取到操作数或存储操作结果。

④ 循环结束判断：它保证循环程序按预定的循环次数或按预定的条件循环，能控制循环程序在有限次执行后正常退出循环结构。如果无限次地执行循环体，就会发生“死”循环，循环控制失败。

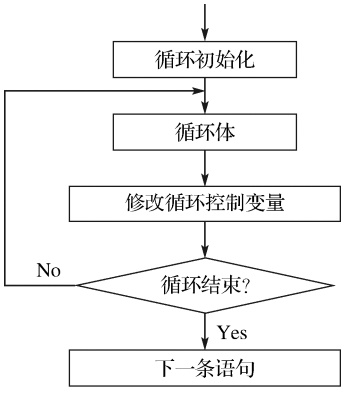

图 3.4　循环程序的基本结构

2．循环指令

实现循环结构的指令有循环控制指令和串操作类（串传送、串存储、串装入、串比较、串扫描）指令。这里不介绍串操作类指令。

循环控制指令有 3 条，即 LOOP 指令、LOOPZ 指令、LOOPNZ 指令。

格式：LOOPxx　lab

操作数寻址方式：lab 为段内短转移，即 lab 偏移地址的位移量必须在字节数（8 位）范围内。循环控制指令的寻址方式与有条件转移指令的寻址方式相同，即段内短转移。

操作：IP = IP + <8 位位移量>（补码）。

循环控制指令简表如表 3.7 所示。

表 3.7　循环控制指令简表

指　令　符	操作和循环测试条件	功　　能
LOOP	CX−1，CX≠0	循环控制
LOOPZ	CX−1，CX≠0 且 ZF=1	零标志循环控制
LOOPNZ	CX−1，CX≠0 且 ZF=0	非零标志循环控制

循环控制指令隐含涉及 CX 的减 1 操作，用于进行循环次数的计数控制。所以，CX 也常被称为循环计数器。

例如，

```
        LOOP    PP3             ;CX−1，CX≠0 转 PP3
```

上例相当于以下两条指令：

```
        DEC     CX
        JNZ     PP3
```

【例 3.6】把 BLKS 数据区的 *N* 个字节数据“搬家”到 BLKD 数据区。

```
;数据变量定义
BLKS    DB      'THIS IS A PROGRAM FOR STRING MOVING'     ;“搬家”的源数据串
N       EQU     $-BLKS          ;N 为数据个数
BLKD    DB      N  DUP  (?)     ;存放“搬家”的目的数据串
;“搬家”程序段
        MOV     AX, DATA
        MOV     DS, AX          ;设置 DS 数据段址
        LEA     SI, BLKS        ;SI 取源数据串 EA
        LEA     DI, BLKD        ;DI 取目的数据串 EA
        MOV     CX, N           ;CX 取数据个数
LOP1:   MOV     AL, [SI]        ;取一个数
        MOV     [DI], AL        ;“搬”一个数
        INC     SI
        INC     DI              ;SI 和 DI 分别做+1 修改
        LOOP    LOP1            ;CX−1≠0，继续“搬”数
```

3．循环程序设计

汇编语言循环程序设计有两种循环控制方法，即计数控制法和条件控制法。

1）计数控制循环程序设计

用计数实现循环控制是最常用的循环程序设计方法，这种方法适用于已知循环次数的应用场合。计数控制法循环程序一般用 CX 作为计数器，初始化时设置计数初值，与 LOOP 指令配合，采用“倒计数”进行计数控制。

【例 3.7】计算 *N*!。

如果设定 *N*!＜65535，即 *N*!不超出一个字（16 位）数据范围，则 *N* 只能取值 1～8。

```
;数据变量定义
N       EQU     x               ;x 为 1～8 中的数之一
```

```
ANS     DW      ?               ;存放 N！单元
;计算 N！程序段
        MOV     AX, 1
        MOV     CX, N           ;循环初始化（AX=1，CX= N）
NEXT:   MUL     CX              ;循环体，即 AX= N×(N−1)×…×1
        LOOP    NEXT            ;CX−1≠0，转 NEXT
        MOV     ANS, AX         ;结果值存放到 ANS 单元
```

【例 3.8】计算 SUM = $a_1b_1+ a_2b_2+\cdots+ a_{10}b_{10}$。

```
;数据变量定义
A       DB      89,5,−56,80,19,−5,76,80,100,12      ;A 数组 10 个数据
B       DB      8,−29,102,38,−5,62,30,−10,52,12     ;B 数组 10 个数据
SUM     DW      ?
；计算 SUM 程序段
        MOV     DX, 0           ;DX=0（计算结果初值）
        MOV     SI, 0           ;SI=0（数组下标初值）
        MOV     CX, 10          ;CX=10（计数初值）
LOP1:   MOV     AL, A[SI]
        IMUL    B[SI]           ;AX = ajbj
        ADD     DX, AX
        INC     SI              ;下标值+1
        LOOP    LOP1            ;CX−1≠0，继续计算
        MOV     SUM, DX         ;结果值存放到 SUM 单元
```

2）条件控制循环程序设计

如果循环程序的循环次数不确定，但能根据设定的某个条件成立与否做有限次的循环控制，则这种循环程序设计方法就是条件控制法。

【例 3.9】求满足 $\Sigma i < 8000$ 的最大数 X。

```
;数据变量定义
X       DW      ?               ;存放 X
CONS = 8000
;求最大数 X 程序段
        MOV     AX, 0           ;AX=0（Σi 初值）
        MOV     BX, 0           ;BX=0（X 初值）
NEXT:   INC     BX
        ADD     AX, BX          ;求Σi
        CMP     AX, CONS        ;Σi 与 8000 比较
        JB      NEXT            ;小于 8000，继续循环
        DEC     BX
        MOV     X, BX           ;结果值存放到 X 单元
```

3）多重循环程序设计

如果一个循环结构的循环体中包含另一个循环结构，就构成了多重循环结构。下面以二

重循环程序设计为例，给出多重循环程序的设计思路。

【例 3.10】统计 BUF 数据区 64 个字节数中“1”数位的个数，并将结果存放到 COUNT 单元。

该统计程序采用二重循环结构设计。外循环采用计数控制法，CX=64（初值）；内循环采用条件控制法。内循环的条件控制程序设计要点如下。

① AL 中为统计的字节数，AL 逻辑左移 1 位，最低位补 0，最高位移入 CF 标志位。

② 用“ADC　BX, 0”指令加 CF 标志值，即(BX)+1 或(BX)+0，做统计。

③ 判断“AL = 0 ？”，如果 AL = 0，则不再统计该字节数，退出内循环。

```
;统计程序段
        MOV     BX, 0           ;BX=0（统计初值）
        LEA     SI, BUF         ;SI 取 BUF 数据区 EA
        MOV     CX, 64          ;CX=64（外循环初值）
EX1:    MOV     AL, [SI]
IN1:    SHL     AL, 1           ;左移 1 位，最高位移入 CF 标志位
        ADC     BX, 0           ;统计，BX 加 CF 标志（0/1）
        CMP     AL, 0
        JNZ     IN1             ;内循环不结束，继续统计该字节数
        INC     SI              ;SI+1
        LOOP    EXl             ;CX−1≠0，继续统计下一个字节数
        MOV     COUNT, BX       ;统计结果存放到 COUNT 单元
```

【例 3.11】ARR 数据区有 *N* 个有符号字节数（ARR 数组）。求 ARR 数组的最大值、最小值、数组元素之和，以及数据平均值。

```
DATA    SEGMENT
ARR     DB      34, −45, 12, 66, −89, 26, 90, 67, −22, 120, 50, 70, 10, 0, −44, 55
        DB      67, 39, −82, −67, 20, −38, 23, −88, 0, −110, 98, 20, −55, 45
N       EQU     $−ARR           ;N=ARR 数组的数据个数
MAX     DB      −128            ;预先放最小值
MIN     DB      127             ;预先放最大值
SUM     DW      0               ;预先放求和初值 0
PING    DB      ?               ;存放平均值
DATA    ENDS
CODE    SEGMENT
        ASSUME  CS: CODE, DS: DATA
START:  MOV     AX, DATA
        MOV     DS, AX
        LEA     BX, ARR         ;BX 取 ARR 数组的 EA
        MOV     CX, N           ;取数据个数
PP1:    MOV     AL, [BX]
        CBW
        ADD     SUM, AX         ;求和
```

```
        CMP     MAX, AL
        JGE     P1
        MOV     MAX, AL         ;求最大值
P1:     CMP     MIN, AL
        JLE     P2
        MOV     MIN, AL         ;求最小值
P2:     INC     BX
        LOOP    PP1             ;循环结束判断
        MOV     AX, SUM
        MOV     CL, N
        IDIV    CL              ;求平均值
        MOV     PING, AL        ;存放平均值
        MOV     AX, 4C00H
        INT     21H             ;返回 DOS
CODE    ENDS
        END     START
```

3.3.4　子程序设计和系统功能调用

汇编语言子程序设计是实现程序模块化设计的重要手段之一。本节主要介绍子程序（过程）及其结构和设计，以及系统提供的软件中断服务子程序的调用（系统功能调用）。

1．子程序

子程序（过程）是具有一定功能的、独立的指令序列程序段，可以在需要时被一次或多次调用。被调用的程序段称为子程序，调用子程序的程序段称为主程序。

1）子程序定义

8086/8088 子程序的定义是由 PROC 和 ENDP 这一对伪指令实现的。

① 过程定义伪指令。

格式：<过程名>　PROC　[<过程类型>]

过程类型有 NEAR（段内过程类型）和 FAR（段间过程类型）两种，默认为 NEAR。

② 过程结束伪指令。

格式：<过程名>　ENDP

一个子程序的 PROC 语句和 ENDP 语句的过程名必须一致。

2）子程序调用指令和返回指令

子程序的调用是用 CALL 指令实现的。当执行到子程序中的 RET 指令时，返回 CALL 指令调用的下一条指令继续执行（主）程序。

① 子程序调用指令。

格式：CALL　<过程名>　　;直接调用（定义的过程类型）

CALL 指令的寻址方式和 JMP 指令的寻址方式相同，有段内直接调用、段间直接调用、段内间接调用、段间间接调用 4 种调用形式。

```
CALL   NEAR PTR  <过程名>      ;段内（近）直接调用（IP）
CALL   FAR PTR   <过程名>      ;段间（远）直接调用（CS : IP）
CALL   WORD PTR  reg/mem       ;段内（近）间接调用（IP）
CALL   DWORD PTR  mem          ;段间（远）间接调用（CS : IP）
```

操作：

a）把返回地址压入堆栈保存（段内：返回 IP 入栈。段间：返回 CS 和 IP 入栈）。

b）转向被调子程序（段内：设置 IP。段间：设置 CS 和 IP）。

例如，

```
CALL   SUB1                    ;根据 SUB1 定义的过程类型调用
CALL   NEAR  PTR  SUB2         ;段内直接调用，IP = SUB2 偏移地址
CALL   FAR  PTR  SUB3          ;段间直接调用，CS = SUB3 段址，IP = SUB3 偏移地址
CALL   WORD  PTR  [BX]         ;段内间接调用，IP =(DS:BX)
CALL   DWORD PTR  [BX]         ;段间间接调用，IP =(DS:BX)，CS=(DS:(BX+2))
```

② 子程序返回指令。

格式：

```
RET                            ;返回主程序
RET    n                       ;返回主程序，并清除堆栈顶 n 个字节（n 为偶数）
```

操作：

如果是段内过程类型，从堆栈弹出 IP；如果是段间过程类型，从堆栈弹出 IP 和 CS。如果是“RET n”指令，除弹出返回地址以外，还要执行(SP)+ n→SP。

在使用 RET 指令时，一定要注意以下两点。

a）当子程序结束时，必须执行到 RET 指令。

b）当执行 RET 指令时，SP 要指向 CALL 指令调用时保存的返回地址，以便能正确返回主程序。

2．子程序结构

子程序结构必须在代码段结构之内。子程序只能通过 CALL 指令进入，也只能通过 RET 指令返回，其他进/出子程序的方式都是错误的。

【例 3.12】8086/8088 汇编语言子程序设计结构示例。

```
CODE    SEGMENT                ;定义 CODE 代码段
        ASSUME  CS : CODE, DS : DATA
START:
        <调用前程序段>
        CALL    PRO1           ;调用子程序 PRO1
        <调用后程序段>
        MOV     AX，4C00H
        INT     21H            ;返回 DOS
PRO1    PROC                   ;定义子程序 PRO1
        <保护现场程序段>
        <子程序功能段>
```

```
        <恢复现场程序段>
        RET                         ;子程序返回
PRO1    ENDP                        ;子程序 PRO1 结束
CODE    ENDS                        ;CODE 代码段结束
        END     START               ;汇编结束，START 为执行入口标号
```

3．子程序设计

在遵循上述结构形式的基础上，子程序设计一般包括以下处理环节。

```
<过程名>    PROC        [NEAR/FAR]
            保护现场
            处理入口参数
            子程序功能段
            处理出口参数
            恢复现场
            RET 返回
<过程名>    ENDP
```

1）保护/恢复现场

子程序和主程序中都使用到的寄存器和存储单元称为现场。为了使子程序能被正确地一次或多次调用和返回，必须在进入子程序前或者在子程序中设置保护现场环节，并且在子程序返回主程序前或者返回主程序后设置恢复现场环节。

保存/恢复现场最常用的方法是利用堆栈特性，保护现场时将“现场”压入堆栈，恢复现场时将“现场”弹出堆栈。所以，子程序设计的最大特点是要合理、正确地设计堆栈操作。

【例 3.13】子程序保护现场和恢复现场示例。

```
SUB1    PROC
        PUSH    AX              ;保护现场（AX, BX, CX 入栈）
        PUSH    BX
        PUSH    CX
        ……                      ;子程序功能段
        POP     CX              ;恢复现场（CX, BX, AX 出栈）
        POP     BX
        POP     AX
        RET
SUB1    ENDP
```

2）子程序的参数传递

汇编语言子程序常需要在主程序和子程序之间传递入/出口参数。除可以利用寄存器、存储单元传送参数以外，还可以利用堆栈传递参数。子程序参数传递的方法有以下 3 种。

① 利用寄存器传递参数：用寄存器传递子程序入/出口参数。这种传递参数的方法最简单、最通用，但由于寄存器数目的限制，传递的参数个数有限。

② 利用存储单元传递参数：用存储单元传递子程序入/出口参数。这种传递参数的方法一般适用于传递的参数个数较多且参数是连续存放在一个数据缓冲区的场合。

③ 利用堆栈传递参数：把传递的入口参数依次压入堆栈，然后调用子程序；子程序取出堆栈中的参数；使用“RET　n”指令返回主程序，弹出堆栈的入/出口参数。

子程序的参数传递要特别注意入/出口参数在堆栈中的位置，并正确使用堆栈指针，保证参数“先进后出”的堆栈操作。

【例 3.14】求一个无符号数的十进制、二进制、十六进制的各数位值的和。

例如，104（01101000B，68H）十进制、二进制、十六进制的数位和分别是 5、3、14。

```
;数据变量定义
NUM     DW      x               ;x 是一个无符号字数据
SUM     DW      ?               ;存放数位和
;主程序
        ……
        MOV     AX, NUM         ;AX 取数 NUM
        MOV     CX, 10          ;CX 取数制 10（或 2 或 16）
        CALL    SUBR            ;AX, CX 为入口参数，BX 为出口参数（数位和）
        MOV     SUM, BX         ;数位和存放到 SUM 单元
        ……
        MOV     AX, 4C00H
        INT     21H             ;返回 DOS
;求数位之和子程序 SUBR
SUBR    PROC
        PUSH    DX              ;保护现场 DX
        MOV     BX, 0           ;BX=0（数位和初值）
  LOP:  CMP     AX, 0
        JZ      OK              ;AX= 0 结束，转 OK（有条件结束循环）
        MOV     DX, 0           ;被除数高 16 位 DX 清 0
        DIV     CX              ;除以 10（或 2 或 16），余数存放在 DX
        ADD     BX, DX          ;求数位和
        JMP     LOP
   OK:  POP     DX              ;恢复现场 DX
        RET
SUBR    ENDP
```

【例 3.15】将例 3.14 中用寄存器传递的参数改为用存储单元传递。

```
;数据变量定义
NUM     DW      x               ;x 为一个无符号数
DECI    DW      10              ;取数制 10（或 2 或 16）
SUM     DW      0               ;存放数位和初值 0
;主程序
        ……
        CALL    SUBR            ;求 NUM 的数位和
```

```
        ……
        MOV     AX, 4C00H
        INT     21H             ;返回 DOS
;求数位之和子程序 SUBR
SUBR    PROC
        PUSH    DX              ;保护现场 DX
        MOV     AX, NUM         ;AX 取 NUM 数
        MOV     CX, DECI        ;CX 取数制 10（或 2 或 16）
  LOP:  CMP     AX, 0
        JZ      OK              ;AX= 0 结束，转 OK
        MOV     DX, 0
        DIV     CX              ;除以 10（或 2 或 16）
        ADD     SUM, DX         ;数位和在 SUM 单元
        JMP     LOP
   OK:  POP     DX              ;恢复现场 DX
        RET
SUBR    ENDP
```

4．系统功能调用

IBM PC 系统提供了一些对 I/O 设备、文件和内存进行管理的中断服务子程序，供用户用软件中断指令调用，这称为系统功能调用。

1）软件中断指令和中断返回指令

① 软件中断指令。

格式：INT　n

其中，n 为中断类型号，数值范围为一个字节数（0～0FFH）。该指令的功能相当于“CALL　< n 号中断服务子程序 >”指令的功能，即实现 n 号中断服务子程序的段间调用。

② 中断返回指令。

格式：IRET

IRET 指令的功能和 RET 指令的功能相同，即返回 INT 指令调用处。只不过 IRET 是专用的中断服务子程序的返回指令。

2）系统功能调用

系统功能调用分为 BIOS 功能调用和 DOS 功能调用，可以用“INT　n”指令实现系统功能调用。

BIOS 是系统提供的基本 I/O 中断服务子程序，驻留在系统的 ROM 中。BIOS 主要有系统加电自检、引导装入、主要 I/O 设备驱动和接口控制等系统功能，是操作系统与外设之间的“软接口”，处于系统软件的底层。常用的 BIOS 功能调节是中断类型号为 10H～1CH 的功能调用，如 10H 号是显示器 I/O，16H 号是键盘 I/O。

DOS 功能调用是 MS-DOS 提供的、中断类型号为 21H 的功能调用。DOS 功能调用有 90 多个子功能，入口参数 AH 寄存器用于设置子功能号。

DOS 功能调用可以分为 5 类管理：字符 I/O 管理、文件管理、内存管理、作业管理、其他资源管理。

3）常用的 DOS 功能调用方法

① 返回 DOS 功能调用，入口参数 AH=4CH，AL=0（返回码 0）。

```
MOV     AX, 4C00H
INT     21H
```

② 字符 I/O 的 DOS 功能调用。

a）01H 子功能：读一个字符到 AL，并回显，入口参数 AH=01H，出口参数 AL。

```
MOV     AH, 01H
INT     21H
```

b）08H 子功能：读一个字符到 AL，不回显，入口参数 AH=08H，出口参数 AL。

```
MOV     AH, 08H
INT     21H
```

c）02H 子功能：显示 DL 的字符，入口参数 AH=02H，DL。

```
MOV     DL, 'A'
MOV     AH, 02H
INT     21H                 ;屏幕上显示 A
```

d）09H 子功能：显示 DS：DX 指向的终止于“$”的字符串，入口参数 AH=09H，DS，DX。

```
STRING  DB      '12345asdfghjkZXCVBNM67890$'
        MOV     AX, SEG STRING
        MOV     DS, AX
        MOV     DX, OFFSET STRING
        MOV     AH, 09H
        INT     21H
```

e）0AH 子功能：读字符串到 DS : DX 指向的数据缓冲区，入口参数 AH=0AH，DS，DX。

存放键盘输入数据的缓冲区格式：

字节 0	预设缓冲区的字节个数 *n*
字节 1	键盘实际接收的字符个数
字节 2～*n*+1	接收输入的字符串（如果多于 *n* 个字符则丢弃）

例如，

```
STRING  DB      20, ?, 20 DUP (?)
        MOV     AX, SEG STRING
        MOV     DS, AX
        MOV     DX, OFFSET STRING
        MOV     AH, 0AH
        INT     21H
```

【例 3.16】从键盘读取一个字符串，把其中的小写字母转换成大写字母显示。

```
;程序段
AGAIN:  MOV     AH, 8
```

```
        INT     21H             ;AL 读键盘字符（不回显）中断调用
        CMP     AL, 0DH         ;回车键的 ASCII 码值是 0DH
        JZ      EXIT            ;是回车键转 EXIT，程序结束
        CMP     AL, 'a'
        JB      OK              ;是小于'a'的字符，转换成字符显示
        CMP     AL, 'z'
        JA      OK              ;是大于'z'的字符，转换成字符显示
        SUB     AL, 20H         ;是'a'～'z'的字符，转换成大写字母显示
OK:     MOV     DL, AL
        MOV     AH, 2
        INT     21H             ;DL 字符显示中断调用
        JMP     AGAIN           ;继续读一个字符
EXIT:   MOV     AX, 4C00H
        INT     21H             ;返回 DOS
```

习　题　3

3.1　根据题意填空。

（1）可以做存储器操作数寻址的寄存器是_____、_____、_____、_____。

（2）8086/8088 状态标志位有_____、_____、_____、_____、_____、_____，控制标志位有_____、_____、_____。

（3）一个逻辑地址为 2000H: 12A7H 存储单元的物理地址是_____。

（4）堆栈数据操作的特性是_____；队列数据操作的特性是_____。

（5）在同一段程序中，可以连续重复执行的程序段是_____结构。

3.2　请指出下列语句中寻址方式的错误。

（1）MOV　CX, [AX]　　（2）ADD　BL, 500

（3）POP　AL　　（4）MUL　AL, BL

（5）AND　100, AX　　（6）SUB　[BX], [DI]

（7）MOV　DS, 2000H　　（8）RCL　BX, 4

（9）XCHG　BX, 4078H　　（10）INC　[SI]

3.3　按题意给出合适的 1～2 条指令。

（1）将 AX 和 250 做有符号数乘法。

（2）将 AX 和 SI 寄存器间接寻址的存储器操作数相加，和放在 AX 中。

（3）将 AL 的高 4 位数据保留，低 4 位数据置 1。

（4）将 AX 的高 8 位和低 8 位数据互换。

3.4　设 AX=8B6AH，BX=00FFH，分别给出以下指令执行后 AX 的值。

（1）ADD　AX, BX　　;AX= _____H

（2）SUB　AX, BX　　;AX= _____H

（3）AND　　AX, BX　　;AX= _____H

（4）OR　　AX, BX　　;AX= _____H

（5）XOR　　AX, BX　　;AX= _____H

3.5　给出执行以下计算的指令序列，其中 *X*、*Y*、*Z*、*W*1～*W*3 均为 16 位无符号数内存变量。

（1）$W1 \leftarrow 2\times X-3\times Y+Z$

（2）$W2 \leftarrow (X\times Y-215)/Z+4\times X$

（3）$W3 \leftarrow (X \wedge Y) \oplus Z$

3.6　说明下列程序段的功能。程序段执行后，HEX 单元的值是什么？

```
;数据变量定义
ASC     DB      41H
HEX     DB      ?
;程序段
        MOV     AL, ASC
        CMP     AL, 39H
        JBE     NEXT
        SUB     AL, 7
NEXT:   SUB     AL, 30H
        MOV     HEX, AL
        HLT
```

3.7　编写能完成以下功能的程序段。

（1）利用加法和移位指令将无符号字节数 *A* 乘以 6，将积送到 B 单元。

（2）求有符号字数 *A*、*B* 的绝对值，将较小的绝对值送到 C 单元。

（3）设 A 数组有 50 个字节数，将数组元素之和送到 B 单元。

（4）求 *X* 字数据的符号函数，将符号函数值送到 Y 单元。

3.8　统计 BUF 数据区 256 个字节数据中零、正数、负数的个数，分别存放到 ZERO、POSI、NEGA 单元。