

第7章

高级汇编语言技术



设问：

1. 什么是宏？
2. 宏与子程序的相同之处与不同之处是什么？
3. 汇编指令、伪指令、宏指令三者有何不同？
4. 多个代码段下的多个模块如何编写、汇编及连接？

本章重点

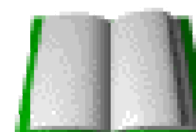
宏功能

结构伪操作

重复汇编和条件汇编

模块化程序结构

宏与多模块



相关知识点

宏是源程序中一段有独立功能的程序代码。
宏也可以称为宏指令、宏操作。

宏的使用需要经过三个步骤：宏定义、宏调用和宏展开。

宏定义

宏定义语句**MACRO**和子程序定义语句**PROC**一样都是伪指令。宏定义需要一对伪指令**MACRO**和**ENDM**完成。

格式：

宏名字 **MACRO** [哑元1， 哑元2， ...]

...

语句串

...

ENDM

说明：宏定义并不产生目标代码，只是用来说明“宏名字”与一段源代码之间的联系。其中哑元1，哑元2，... 是虚拟参数或形式参数，用逗号分隔。虚参或形参可不设置。

例1 定义实现源程序结束功能的宏**RETSYS**

RETSYS MACRO

MOV AH, 4CH

INT 21H

ENDM

注意：起名时，不要和汇编语言的指令名、保留字相同。

例2 定义键盘输入宏指令INPUT

INPUT MACRO

MOV AH, 01H

INT 21H

ENDM

例3 定义两数相加宏指令SUMM

SUMM MACRO X1, X2, RESULT

MOV AX, X1

ADD AX, X2

MOV RESULT, AX

ENDM

在程序中使用宏时，只要写出宏名字[实参]既可。

例1 从键盘输入一个字符，判断是否为“-”号，不是则继续输入，是则结束。（利用前面定义的宏）

```
.MODEL SMALL  
.STACK 100H  
.CODE  
    START:  
        INPUT  
        CMP AL, '-'  
        JNE START  
        RETSYS  
END START
```


例2 调用宏SUMM实现(BX)=34+25

.MODEL SMALL

.CODE

START:

SUMM 34,25,BX

RETSYS

END START

源程序在汇编时，宏指令被汇编程序用相应的程序代码所取代，这个过程称为宏展开。

下面来看一下**8.1.2**宏调用的例子在宏展开后的情况。

例1 展开前：

.MODEL SMALL

.STACK 100H

.CODE

START:

INPUT

1

1

CMP AL, '-'

JNE START

RETSYS

1

1

END START

展开后：

START:

MOV AH, 01H

INT 21H

CMP AL, '-'

JNE START

MOV AH, 4CH

INT 21H

例2 展开前:

.MODEL SMALL

.CODE

START:

SUMM 34,25,BX

RETSYS

END START

展开后:

START:

1 MOV AX, 34

1 ADD AX, 25

1 MOV BX, AX

1 MOV AH, 4CH

1 INT 21H



练习：

- (1) 定义显示一个字符的宏指令**OUTPUT**，要显示的字符用哑元**Z**表示。
- (2) 定义宏指令**KEY_STR**，实现从键盘输入一串字符。
- (3) 定义宏指令**DISPLAY**，显示一串字符。
- (4) 编程序。利用宏指令**INPUT**和**OUTPUT**实现将键入的小写字母变为大写。

求数组中各元素的绝对值

```
data segment
```

```
  x db 2,-5,7,-4
```

```
data ends
```

```
absol macro oper
```

```
    local next
```

```
    cmp oper,0
```

```
    jg next
```

```
    neg oper
```

```
    next:
```

```
endm
```

```
code segment
```

```
  assume cs:code, ds:data
```

```
  main proc far
```

```
start:
```

```
  push ds
```

```
  mov ax,0
```

```
  push ax
```

```
  mov ax,data
```

```
  mov ds,ax
```

```
  absol x
```

```
  absol x+1
```

```
  absol x+2
```

```
  absol x+3
```

```
  ret
```

```
  main endp
```

```
code ends
```

```
end start
```

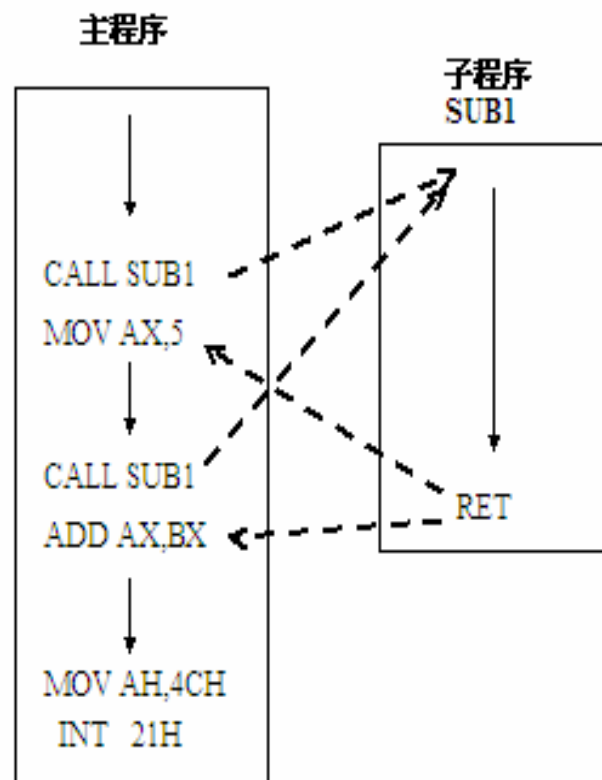
宏与子程序都是编写结构化程序的重要手段，两者各有特色。

●**相同之处：**宏和子程序都可以定义为一段功能程序，可以被其他程序调用。

●**不同之处：**

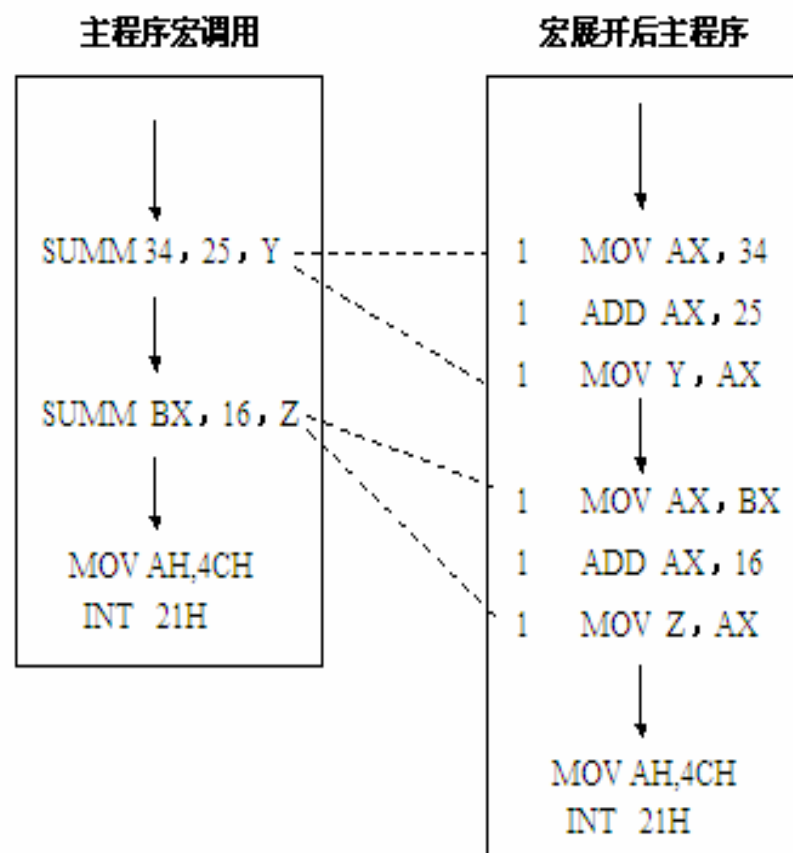
- (1) 宏指令利用哑元和实元进行参数传递。宏调用时用实元取代哑元，避免了子程序因参数传递带来的麻烦。使宏汇编的使用增加了灵活性。
- (2) 实元还可以是指令的操作码或操作码的一部分，在编译汇编的过程中指令可以改变。
- (3) 宏调用的工作方式和子程序调用的工作方式是完全不同的。

如图：子程序调用方式



a 子程序调用方式

宏调用方式



b 宏调用方式

在宏定义的形参表中的参数可以有多种形式，灵活使用这些参数可以实现不同功能。

1. 变元是操作数

例1 定义串传送宏指令**STR_MOV**。

宏定义：

```
STR_MOV MACRO OPR1, OPR2, OPR3  
    PUSH CX  
    MOV CX, OPR1  
    LEA SI, OPR2  
    LEA DI, OPR3  
    CLD  
    REP MOVSB  
    POP CX  
    ENDM
```


宏调用:

STR_MOV 10, MESS1, MESS2

.....

STR_MOV 20, X, Y

宏展开:

1 PUSH CX
1 MOV CX, 10
1 LEA SI, MESS1
1 LEA DI, MESS2
1 CLD
1 REP MOVSB
1 POP CX

.....

1 PUSH CX
1 MOV CX, 2
1 LEA SI, X
1 LEA DI, Y
1 CLD
1 REP MOVSB
1 POP CX

2. 变元是操作码

例2 定义栈操作宏指令STACKM。

宏定义:

```
STACKM MACRO OPR1, OPR2, OPR3  
        PUSH  OPR1  
        OPR2  OPR3  
ENDM
```

宏调用:

```
STACKM AX, POP, BX  
STACKM CX, PUSH, SI
```

宏展开:

```
1      PUSH AX  
1      POP  BX  
  
1      PUSH CX  
1      PUSH SI
```

例3 定义移位宏指令SHIFT。

宏定义：

```
SHIFT MACRO A, B, C
    PUSH CX
    MOV CL, C
    S&A B, CL
    POP CX
ENDM
```

宏调用：

```
SHIFT HR, BX, 1
SHIFT AL, AL, 3
```

宏展开：

```
1      PUSH  CX
1      MOV   CL, 1
1      SHR   BX, CL
1      POP   CX

1      PUSH  CX
1      MOV   CL, 3
1      SAL   AL, CL
1      POP   CX
```

3. 变元是操作码一部分
变元出现在操作码中，要用&作为分隔符。

4. 变元是存储单元

在数据段中用伪指令定义存储单元时，也可以使用宏，使单元内容的设置更灵活。

例4 定义存储单元宏指令DATAS。

宏定义：

```
DATAS MACRO A1, A2, A3  
      X&A1 DB A2 DUP(A3)  
      ENDM
```

宏调用：

```
DATAS 5, 6, 1  
DATAS 2, 3, 4
```

宏展开：

```
1 X5 DB 6 DUP(1)  
1 X2 DB 3 DUP(4)
```

5. 变元是字符串

例5 定义字符串宏指令**MASG**。

宏定义：

```
MASG MACRO S1, S2  
    POSIT&S1 DB 'Number &S2',0AH,0DH,'$'  
ENDM
```

宏调用：

```
MASG 1, ABC  
MASG 2, 2
```

宏展开：

```
1 POSIT1 DB 'Number ABC' , 0AH, 0DH, '$'  
1 POSIT2 DB 'Number 2' , 0AH, 0DH, '$'
```

宏运算是指以特殊运算符实现不同变元的过程。

包括**&**、**<>**、**!**、**%**、**;;** 5种运算符。

1. & 替换运算符

用于将字符串与哑元连接。宏调用时，字符串与相应的实元内容连在一起。

例1 定义字符串宏指令DISTR.。

宏定义：

```
DISTR MACRO SS  
    DB 'Exam: &SS', 0AH, 0DH, '$'  
ENDM
```

宏调用：

```
DISTR book
```

宏展开：

```
1 DB 'Exam: book', 0AH, 0DH, '$'
```

2. < > 传递运算符

在变元为字符串时，如果实元是含有空格的字符串，则实元要用< > 传递运算符括起来。

宏调用：

DISTR < I am a student >

DISTR lamastudent

宏展开：

**1 DB 'Exam: I am a student' , 0AH,
0DH, '\$'**

3. ! 转义运算符

当字符串中含有< 或 >字符时，为避免与传递运算符冲突，在宏调用的实元中用! 表示该字符为普通字符。

宏调用：

DISTR X!<Y

宏展开：

1 DB 'Exam: X<Y' , 0AH, 0DH, '\$'

4. % 表达式运算符

在宏调用的实元中如果有表达式，%运算符将表达式的值作为实元。

宏调用：

DISTR %35+12

宏展开：

1 DB 'Exam: 47' , 0AH, 0DH, '\$'

5. ;;宏注释符

双分号;;宏注释符是在宏定义中使用的注释符。其后的注释在宏调用及宏展开时不出现。

其它宏功能

宏标号

格式: **LOCAL** 标号1 [, 标号2...]

说明: **LOCAL**指定局部标号伪指令只能在宏定义体中使用。并且是宏定义体的第一条语句。**LOCAL**的作用是将给出的标号在多次宏调用时以不同的数字取代标号, 避免标号的重复定义。

例 定义分支宏指令**BRANCH**。

宏定义：

```
BRANCH MACRO B1, B2  
    LOCAL OUT1  
    MOV AL, B1  
    CMP AL, B2  
    JLE OUT1  
    SUB AL, B2  
    OUT1: HLT  
    ENDM
```

宏调用：

```
BRANCH 10, BL  
BRANCH DL, BH
```

宏展开:

```
1    MOV  AL, 10
1    CMP  AL, BL
1    JLE  ??0000
1    SUB  AL, BL
1    ??0000: HLT
```

```
1    MOV  AL, DL
1    CMP  AL, BH
1    JLE  ??0001
1    SUB  AL, BH
1    ??0001: HLT
```

当不需要某个宏时，可以将其删除。

格式：**PURGE** 宏名[, 宏名...]

说明：**PURGE**伪指令在汇编时将该语句中的宏定义名删除。

例 将宏**INPUT**和**SUMM**删除。

PURGE INPUT, SUMM

宏嵌套

在宏定义中可以使用宏调用，称为宏嵌套。宏嵌套可以增加宏的功能，简化宏的操作。

例1 定义判断运算宏指令**DIST_OPER**。

宏定义：

```
DIST_OPER MACRO DD1, DD2  
    LOCAL LET1  
    INPUT          ; 已定义的键盘输入宏指令INPUT  
    CMP AL, '-'  
    JNE LET1  
    NEG DD1  
    LET1:  
    ADD DD1, DD2  
    RETSYS        ; 已定义的源程序结束功能的宏RETSYS  
    ENDM
```

宏调用：

```
DIST_OPER X, BL
```

宏展开:

```
2      MOV AH, 01H
2      INT  21H
1      CMP AL, '-'
1      JNE ??0000
1      NEG X
1      ??0000
1      ADD X, BL
2      MOV AH, 4CH
2 INT 21H
```

宏展开中数字**2**表示是宏嵌套中的指令。

1. 建立宏库

把这些宏的宏定义部分放在一个文本文件中，为其起名并加上扩展名**.MAC**，称为宏库。

例 建立宏库**8-1.mac** 文件。共有**4**个宏。

程序如下：

;8-1.mac 宏库

;1

INPUT macro

mov ah,01H

int 21h

endm

;2

OUTPUT macro x

mov dl,x

mov ah,02h

int 21h

endm

```
;3  
RETSYS macro  
mov ah,4ch  
int 21h  
endm  
;4  
ADDI macro x1,x2,result  
mov ax,x1  
add ax,x2  
mov result,ax  
endm  
;5  
STR_MOV macro opr1,opr2,opr3  
mov cx,opr1  
lea si,opr2  
lea di,opr3  
cld  
rep movsb  
endm
```


2. 调用宏库

在应用程序中使用宏指令之前，用**INCLUDE**伪指令把宏库调入，然后再使用这些宏。

示例8-1 宏库的使用。

;**8-1.asm** 宏库的使用

include 8-1.mac

.model small

.stack 100h

.data

x db 33h,34h

y dw ?

mess1 db 1,2,3,4,5,6,7,8,9,0

mess2 db 10 dup(?)

.code

```

start:
mov ax,@data
mov ds,ax
mov es,ax
;
STR_MOV 10,mess1,mess2      ;mess1传送到mess2
STR_MOV 2,x,y              ;x传送到y
;
INPUT                      ;输入的小写字母变为大写输出
sub al,20h
OUTPUT al
;
ADDI 34,25,y              ;y=34+25
RETSYS                    ;结束，返回DOS
end start

```

结构伪操作

1.结构定义

格式： 结构名 **STRUC**

结构体

结构名 **ENDS**

例 定义结构**CLASS**，存放班级情况。

```
CLASS STRUC
```

```
    NO          DB ?
```

```
    NAME1       DB 'XXXXXX'
```

```
    SEX         DB 'XXXX'
```

```
    AGE         DB ?
```

```
    RESU        DB ?
```

```
CLASS ENDS
```

2.结构预置

结构定义之后还不能使用，要对结构预置后才能把相关信息真正存入存储器。

格式： 结构变量名 结构名 <字段值表>

说明：结构变量名可任意起名，用于在程序中直接引用。

结构名是结构定义时的名字；<字段值表>用于给结构变量赋初值。

结构预置：

```
mem1 class  
<1,'WANG','MAN',18,89>
```

```
mem2 class <2,'LI','MAN',18,76>
```

```
mem3 class  
<3,'JIANG','FMAN',17,92>
```

3.结构引用

结构在定义和预置之后，在程序中可以使用。在引用时，直接写结构变量名。

格式：结构变量名. 结构字段名

说明：“.”表示对字段的访问。在使用时，可以预先将结构变量的起始地址、偏移量送往某个寄存器，再用寄存器间址代替结构变量名。

结构引用：

```
mov si, 0
```

```
mov al,mem1.age[si]
```

；把mem1的年龄age字段值→AL

```
inc si
```

```
mov mem2.name1[si],'U'
```

；把mem2的姓名name1字段的第二个字母改为'U'

重复汇编和条件汇编

重复汇编和条件汇编都是伪操作。利用重复汇编和条件汇编，在编写汇编语言程序时可以简化程序的书写，控制程序代码的生成，为程序员提供方便。

重复汇编

1. 重复次数确定

格式: **REPT** 重复次数n

重复体

ENDM

功能: 将重复体重复n次。

例 建立数字0~9的ASCII码表。

. DATA

N='0'

REPT 10

DB N

N=N+1

ENDM

在数据段中定义了10个单元，存放30H~39H。

2. 重复次数不确定

格式1: **IRP** 哑元, <实元1, 实元2, ...>

重复体

ENDM

功能: 用实元替代哑元, 重复次数由实元的个数决定。

格式2: **IRPC** 哑元, 字符串

功能: 由字符串替代哑元, 重复次数由字符串的字符个数决定。

例 用**IRP**定义子程序现场保护功能。

```
.CODE  
  
IRP  
REG ,<AX,BX,CX,DX,SI,DI,BP>  
  
PUSH REG  
  
ENDM
```

汇编时, 在代码段中连续插入了7条**PUSH**指令, 分别是
PUSH AX~PUSH BP。

条件汇编

格式: **IF** 表达式

代码段1

ELSE

代码段2

ENDIF

例 在程序中控制某条指令是否汇编。

.CODE

...

IF X EQ 0 ; 汇编时, 如果X单元的值等于0,

MOV BX,0 ; 这两条指令加在程序中

MOV AL,[BX]

ELSE ; 否则, 下面两条指令加在程序中

MOV BX,1

MOV DL,[BX]

ENDIF ...

模块化结构特点

编写复杂的大型程序时会有多人参与编程。每个程序员都编写自己的代码段，这就形成了多个代码段、多个模块的大系统。

要想实现多模块汇编，就要事先将各个参数进行说明和定义，使有关的参数能够关联起来。同时，各个模块中的源程序要独立汇编，生成各自的`.OBJ`文件，然后用`LINK`命令连接到一起，最终生成`.EXE`可执行文件。

1. 全局符号定义PUBLIC

在各个模块间共用的变量、符号、标号、过程等要用**PUBLIC**伪指令事先说明为全局变量，以便能被其他模块引用。

格式：**PUBLIC** 符号1[, 符号2,]

功能：将本模块中的符号或过程定义为全局变量，共其它模块使用。

2. 外部符号说明**EXTRN**

EXTRN伪指令用来说明某个变量、符号或过程是其它模块定义的，在本模块中需要引用。

格式：**EXTRN** 符号1：类型 [, 符号2：类型,]

功能：将外部符号和其类型进行说明。

类型为：**BYTE**、**WORD**、**DWORD**、**NEAR**、**FAR**等。符号的类型要与它在定义模块中的一致。

3. 段属性与段组合

由于多个源程序分别在不同的代码段中使用，因此段的属性要设置正确，以便于段组合。在定义代码段时，代码段名相同时要加上**PARA'CODE'**，以使其类别相同；数据段也可以用**PARA'DATA'**加以说明。

在多模块程序设计中，最少定义一个堆栈段，一般在主模块中定义。

主模块的最后一条结束伪指令**END START**必须加上标号

(**START**)，而其它模块的**END**语句不能带标号。

4. 参数传递

多模块之间的参数传递方法与子程序传参类似，也可以用寄存器传参、存储单元传参、堆栈传参。通过对变量的**PUBLIC/EXTRN**的声明，可以实现参数传递。但是要注意段的名字、类别要相同。还可以将数据段定义为共享数据段，即组合类型为**COMMON**，利用公共数据段实现模块间的数据访问。

实例八 宏与多模块

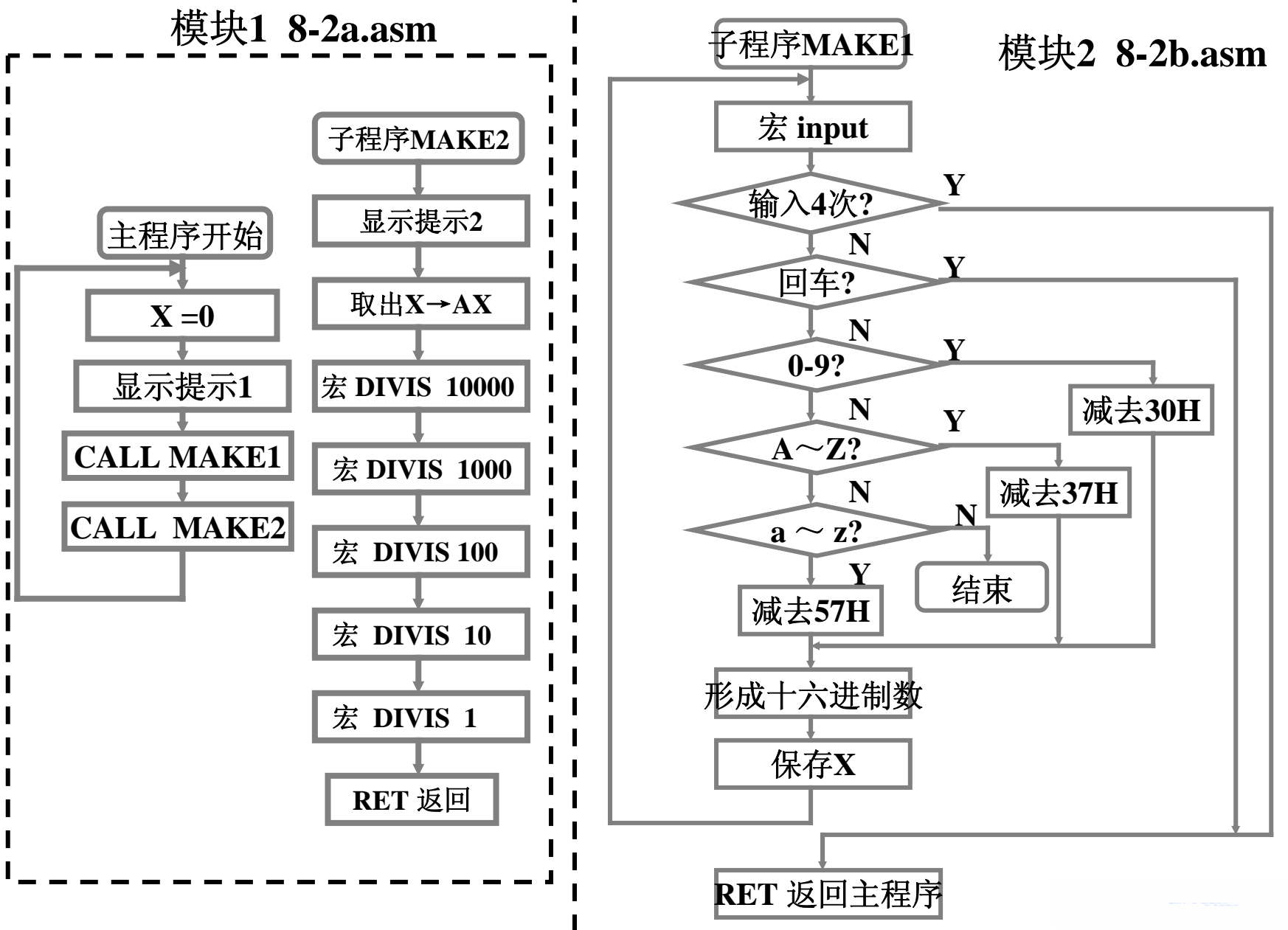
8.6.1 宏与模块化

示例8-2 从键盘输入4位十六进制数，转换成十进制数显示出来。

设计思路:

- (1) 设计一个主程序**make0**、两个子程序**make1**和**make2**；用两个代码段分别保存主程序和子程序；实现远程的访问与调用。
- (2) 将常用的功能设为宏，并用宏库**8-2.mac**保存。宏库中共有**6**个宏。
- (3) 主程序**make0**和子程序**make1**在同一代码段中，保存在同一个模块（**.asm**），作近程调用；另一个子程序**make2**在另外一个代码段中，单独一个模块，作远程调用。
- (4) 主程序**make0**的功能是调用两个子程序；
- (5) 子程序**make1**功能是键盘输入，并把输入的数字变为十六进制数**X**；
- (6) 子程序**make2**功能是通过多次调用宏**DIVIS**查表将十六进制数**X**变为十进制，并显示出来。

程序框图：



程序如下：

(1) 模块1: 8-2a.asm

;8-2a.asm 远程调用模块化程序。

extrn make1:far ;外部符号说明,**make1**子程序是远程的

public x ;定义**x**为公共变量

include 8-2.mac ;宏库

data segment

x dw 0

mess1 db 0dh,0ah,'input hex=\$'

mess2 db 0dh,0ah,'out dec=\$'

dectab db '0123456789'

data ends

stack segment para stack 'stack' ;堆栈段

dw 100h dup(0)

stack ends

code segment para 'code'

assume cs:code,ds:data,ss:stack

```

start:
mov ax,data
mov ds,ax
make0 proc far           ;主程序make0
    mov x,0
    display mess1        ;宏display,显示提示1
    mov bx,0
    call make1
    call make2
    jmp make0
make0 endp
make2 proc               ;子程序make2: 查表, 显示十进制
    display mess2        ;宏display,显示提示2
    mov ax,x
    mov dx,0
    divis 10000           ;宏divis, 做除法并显示。
    divis 1000
    ret
    make2 endp
code ends
end start

```

(2) 模块2: 8-2b.asm

```
; 8-2b.asm  
public make1           ;定义make1子程序为公共类型  
extrn x:word           ;说明另一个模块中的x为字型  
include 8-2.mac        ;调入宏库  
code segment para'code' ;代码段名类别相同  
assume cs:code  
make1 proc far         ;子程序make1: 键盘输入、形成十六进制  
inc bx  
cmp bx,4              ;键入4次?  
jg exit  
input                 ;宏input,键盘输入十六进制数  
cmp al,0dh  
jz exit  
cmp al,'0'            ;判断是否0-9, A-F或a-f  
jl out1               ;是其它字符,转out1  
cmp al,'9'  
jle smal1
```

```

cmp al,'A'
jl out1
cmp al,'F'
jle smal2
cmp al,'a'
jl out1
cmp al,'f'
jg out1
sub al,20h
smal2:
sub al,7
smal1:
sub al,30h
mov ah,0
xchg ax,x
mov cx,16
mul cx
xchg ax,x
add x,ax
jmp make1
exit:ret

```

;小写a-f减去57h
 ;大写字母A-F减去37h

;数字0-9减去30h

;形成十六进制数

```

out1:
retsys      ;宏retsys,结束、返回DOS
make1 endp
code ends
end

```

(3) 宏库8-2.mac

;8-2.mac 宏库

;1

input macro

; 宏input, 键盘输入一个字符

mov ah,01H

int 21h

endm

;2

output macro opr1

; 宏output, 显示一个字符

mov dl,opr1

mov ah,02h

int 21h

endm

;3

retsys macro

; 宏retsys, 结束、返回DOS

mov ah,4ch

int 21h

endm

;4

key_str macro opr1

; 宏key_str， 键盘输入一串字符

mov dx,offset opr1

mov ah,10

int 21h

endm

;5

display macro opr1

; 宏display， 显示一串字符

lea dx,opr1

mov ah,9

int 21h

endm

;6

divis macro x

; 宏divis， 做除法并显示

mov cx,x

mov ah,2

;显示ax中部分商

div cx

int 21h

mov bx,dx

mov ax,bx

mov si,ax

mov dx,0

mov dl,dectab[si]

endm

运行结果:

(1) 先将两个模块分别汇编

```
C:\hb>masm 8-2a;  
Microsoft (R) Macro Assembler Version 5.00  
Copyright (C) Microsoft Corp 1981-1985, 1987. All rights reserved.
```

```
50772 + 434316 Bytes symbol space free
```

```
0 Warning Errors  
0 Severe Errors
```

```
C:\hb>masm 8-2b;  
Microsoft (R) Macro Assembler Version 5.00  
Copyright (C) Microsoft Corp 1981-1985, 1987. All rights reserved.
```

```
50968 + 434120 Bytes symbol space free
```

```
0 Warning Errors  
0 Severe Errors
```

```
C:\hb>_
```

(2) 用LINK命令将两个OBJ文件连接（用+号连接）

```
C:\hb>link 8-2a.obj+8-2b.obj;
```

```
Microsoft (R) Overlay Linker  Version 3.60  
Copyright (C) Microsoft Corp 1983-1987.  All rights reserved.
```

```
C:\hb>
```


(3) 运行EXE文件

```
C:\hb>8-2a.exe
```

```
input hex=12  
out dec=00018  
input hex=64  
out dec=00100  
input hex=F  
out dec=00015  
input hex=3a6  
out dec=00934  
input hex=800  
out dec=02048  
input hex=ffff  
out dec=65535  
input hex=q  
C:\hb>
```


- 在程序设计过程中，有时要编写一些小型程序，要求占用空间少、执行速度快。这样的程序只能有一个段。如果只有一个段，这个段必须是代码段。那么数据、堆栈都要在同一个段中。在这样的程序结构中，数据、堆栈、代码是混杂的，此时尤其要注意指针的改变。

- 在带符号数的运算中，如果从键盘输入负号，要求程序能够判断出“-”，并将数值求补。

 **示例8-4** 从键盘多次输入十进制数，无论正、负数，求出补码并用二进制和十六进制显示。

设计思路:

(1) 主程序**main**调用子程序**subr1**，两次调用子程序**subr2**分别显示二进制和十六进制数。

(2) 子程序**subr1**: 功能为键盘输入，数字键**ASCII**码→十进制数（该十进制数保存为二进制）；判断负号，求出负数的补码；用存储单元**x**传参；

(3) 子程序**subr2**: 取出**x**，用循环左移保留要显示的数值，查**ASCII**表分别显示二进制数和十六进制数；

(4) 利用宏库**8-2.mac**简化程序。

运行结果:

```
C:\hb>8-4

input dec=2
binary=000000000000000010
HEX=0002
input dec=15
binary=0000000000000111
HEX=000F
input dec=123
binary=0000000001111011
HEX=007B
input dec=-1
binary=1111111111111111
HEX=FFFF
input dec=-89
binary=1111111110100111
HEX=FFA7
input dec=-127
binary=1111111110000001
HEX=FF81
input dec=-32768
binary=1000000000000000
HEX=8000
input dec=
C:\hb>
```