

第七章 高级汇编语言技术

- 7.1 宏汇编
- 7.2 重复汇编
- 7.3 条件汇编
- 7.4 高级语言结构
- 7.5 ARM宏汇编

本章目标

1. 掌握宏汇编

- 定义、调用、展开

2. 掌握重复汇编

- 读程序、写结果

3. 了解条件汇编

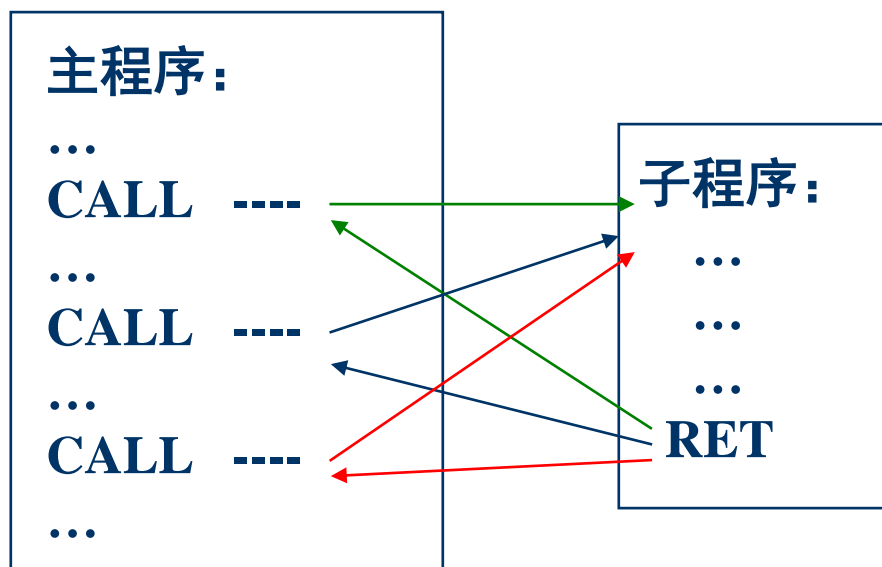
- 调用、展开

7.1 宏汇编

宏：源程序中一段有独立功能的程序代码。

宏指令：用户自定义的指令。在编程时，将多次使用的功能用一条宏指令来代替。

汇编语言程序 { 指令
伪指令（伪操作）
宏指令



优: 模块化
省内存

缺: 开销大

宏定义:

```

Q MACRO x,y
...
ENDM

```

主程序:

```

...
Q a, b
...
Q c, d
...
Q e, f
...

```

目标程序:

```

...
[Macro Expansion Block]
...
[Macro Expansion Block]
...
[Macro Expansion Block]
...

```

优:

参数传送
简单, 执行效率高

缺:

占用内存
空间大

◆ 子程序

■ 优点

- 节省存储空间及程序设计所花的时间
- 提供模块化程序设计的条件
- 便于程序的调试及修改

■ 缺点

- 转子、返回，保存、恢复寄存器，参数的传送等，增加了程序的额外开销（操作所消耗的时间、占用的存储空间）

◆ 宏汇编的用途

- 当子程序本身较短或者需要传送的参数较多的情况下，使用宏汇编更加便利
- 为用户提供更加容易、更加灵活、更加向高级语言靠拢的汇编工具

```
multiply macro opr1,opr2,result
    push dx
    push ax
    mov ax, opr1
    imul opr2
    mov result, ax
    mov result+2, dx
    pop ax
    pop dx
endm

;
data segment
var dw ?
xyz dw ?,?
data ends

;
cseg segment
assume cs:cseg,ds:data
start proc far

;
    mov ax, data
    mov ds, ax
    multiply cx, var, xyz[bx]

;
exit:    mov ax,4c00h
        int 21h

start    endp
cseg     ends

end start
```

源程序.ASM文件

-u			
1425:0000	B82414	MOV	AX,1424
1425:0003	8ED8	MOV	DS,AX
1425:0005	52	PUSH	DX
1425:0006	50	PUSH	AX
1425:0007	8BC1	MOV	AX,CX
1425:0009	F72E0000	IMUL	WORD PTR [0000]
1425:000D	89870200	MOV	[BX+0002],AX
1425:0011	89970400	MOV	[BX+0004],DX
1425:0015	58	POP	AX
1425:0016	5A	POP	DX
1425:0017	B8004C	MOV	AX,4C00
1425:001A	CD21	INT	21

宏展开后
机器指令

```
Microsoft (R) Macro Assembler Version 5.00    4/18/9    Page  1-1

multiply    macro  opr1, opr2, result
    push dx
    push ax
    mov ax, opr1
    imul opr2
    mov result, ax
    mov result+2, dx
    pop ax
    pop dx
endm

;
0000                                data segment
0000  ????                          var dw ?
0002  ????  ????                    xyz dw ?,?
0006                                data ends

;
0000                                cseg segment
                                assume cs:cseg, ds:data
0000                                start proc far

;
0000 B8 ---- R                      mov ax, data
0003 8E D8                          mov ds, ax
                                multiply cx, var, xyz[bx]

0005 52                             1 push dx
0006 50                             1 push ax
0007 8B C1                           1 mov ax, cx
0009 F7 2E 0000 R                    1 imul var
000D 89 87 0002 R                    1 mov xyz[bx], ax
0011 89 97 0004 R                    1 mov xyz[bx]+2, dx
0015 58                             1 pop ax
0016 5A                             1 pop dx

;
0017 B8 4C00                          exit: mov ax,4c00h
001A CD 21                          int 21h
001C                                start endp
001C                                cseg ends
                                end start
```

汇编后的.LIST文件

7.1.1 宏定义、宏调用和宏展开

宏定义：

macro_name **MACRO** [哑元表] ; 形参/虚参

[LOCAL 标号表]

.....
.....

; 宏定义体

ENDM

宏调用：（必须先定义后调用）

macro_name [实元表] ; 实参

宏展开：汇编程序把宏调用展开

宏定义体 \longrightarrow 复制到宏指令位置, 实参代虚参

LOCAL中的标号 \longrightarrow ??0000~??ffff

◆ 宏定义

■ 格式:

macro_name **MACRO** [哑元表]

[LOCAL 标号表]

..... (宏定义体)

ENDM

```
multiply MACRO opr1,opr2,result  
    mov ax, opr1  
    imul opr2  
    mov result, ax  
    mov result+2, dx  
ENDM
```

- * **MACRO**、 **ENDM**是一对宏定义伪操作
- * 哑元表给出形式参数（虚参）
- * 宏定义体：一组有独立功能的程序段
- * 如果宏定义体有一个或多个**标号**，则必须用**LOCAL**伪操作列出所有的标号

宏定义

```
macro_name MACRO [哑元表]  
          [ LOCAL 标号表 ]  
          ..... (宏定义体)  
ENDM
```

- ◆ **宏调用：**定义了宏指令，就可以在程序中多次调用它

- **格式：**

macro_name [实元表] ; 实参

- * 实元表中的实元与哑元表中的哑元在位置上一一对应
- * 若实元数>哑元数，则多余的实元无效
- * 若实元数<哑元数，则多余的哑元作“空(NUL)”处理
- * 对宏指令的调用：**必须先定义后调用**
- * **实元：**可以是常数、寄存器、存储单元名、地址、表达式；也可以是操作码或操作码的一部分

汇编程序主要功能:

- 1、检查程序
- 2、测出源程序中的语法错误, 并给出错误信息
- 3、展开宏指令
- 4、产生源程序的机器语言目标程序, 并给出列表文件, 同时列出汇编语言和机器语言, XXX.LST

宏定义

```
macro_name MACRO [哑元表]  
[LOCAL 标号表]  
..... (宏定义体)  
ENDM
```

宏调用

```
macro_name [实元表]
```

◆ 宏展开:

- 源程序被汇编时, 汇编程序将对每个宏调用作宏展开
 - 用**宏定义体**替换宏指令名, 把**宏定义体**复制到调用宏指令的位置上, 同时用实元取代哑元
- 由LOCAL定义的标号也由 ??0000~??FFFF 偏移量替代 (其实质是自动给了一个新标号)
 - 多次宏调用展开时, 解决标号冲突问题

例7.1 两个16位的字操作数相乘

宏定义：（编程时）

multiply **MACRO** **opr1**, **opr2**, **result**

push **dx**

push **ax**

mov **ax**, **opr1**

imul **opr2**

mov **result**, **ax**

mov **result+2**, **dx**

pop **ax**

pop **dx**

ENDM

宏调用：（编程时）

multiply **cx**, **var**, **xyz[bx]**

宏展开：（汇编时）

1 **push** **dx**
1 **push** **ax**
1 **mov** **ax**, **cx**
1 **imul** **var**
1 **mov** **xyz[bx]**, **ax**
1 **mov** **xyz[bx]+2**, **dx**
1 **pop** **ax**
1 **pop** **dx**

替代

1 表示这些指令由宏展开，同时也表示第一层展开结果，较早版本用 + 表示

7.1.2 宏定义中的参数

- **哑元**：实质上只是一个字符串构成的符号
- **实元**：可以是常数、寄存器、存储单元、地址、表达式；也可以是操作码或操作码的一部分
- 哑元和实元统称变元

例7.2 宏定义无变元

例7.3 变元是操作码

例7.4 变元是操作码的一部分

例7.6 变元是字符串

例7.7 变元是表达式

例7.2 保存寄存器

宏定义可以无变元

宏定义：

savereg **MACRO**

push ax

push bx

push cx

push dx

push si

push di

ENDM

宏展开：

1 push ax

1 push bx

1 push cx

1 push dx

1 push si

1 push di

宏调用：

savereg

例7.3 变元可以是操作码

宏定义：


```
FOO MACRO P1, P2, P3
    MOV P3, P1
    P2 P3
ENDM
```

宏调用：

```
FOO WORD_VAR, INC, AX
```

宏展开：

```
1 MOV AX, WORD_VAR
1 INC AX
```



汇编程序汇编生成.OBJ文件时，生成这2条机器指令，并替代源程序中的宏调用指令

宏汇编操作符 & :: % : REQ ::=

◆ 符号1&符号2

- 文本替换操作符。宏展开时, 合并前后两个符号形成一个符号
- 符号可以是操作码、操作数或是一个字符串

例7.4 变元是操作码的一部分

宏定义: (源程序中定义)

```
leap macro cond, lab  
    j&cond lab  
endm
```

宏调用: (源程序中调用)

```
leap z, there  
.....  
leap nz, here
```

宏展开: (汇编时展开)

```
1 jz there  
.....  
1 jnz here
```

◆ :: 注释

- 宏注释。宏展开时，若注释以一个分号开始，则该注释在宏扩展时出现。若注释以两个分号开始，则::后面的注释不予展开

例: Q **MACRO** m

 ; display a message

 ;; m is a string

ENDM

每次展开保留此注释

每次展开不保留此注释

◆ %表达式

- 表达式操作符。汇编程序将%后面的表达式立即求值转换为数字，并在展开时用这个数取代哑元，宏调用时使用

例7.7

宏定义:

```
MSG      MACRO  COUNT, STRING
MSG&COUNT DB STRING
ENDM
ERRMSG   MACRO  TEXT
CNTR=CNTR+1
MSG      %CNTR, TEXT
ENDM
```

宏调用:

```
.....
CNTR=0
ERRMSG 'SYNTAX ERROR'
.....
ERRMSG 'INVALID OPPERAND'
```

宏展开:

```
.....
CNTR=0
2 MSG1 DB 'SYNTAX ERROR'
.....
2 MSG2 DB 'INVALID OPPERAND'
.....
```

◆ : REQ

- 指定某个变元必须有。调用时必须有对应的实元，否则汇编时出错

例7.8: 宏定义

```
DIF  MACRO A, B
    DB B-A
    ENDM
```

```
DIF1 MACRO A:REQ, B:REQ
    DB B-A
    ENDM
```

```
DIF2 MACRO A:REQ, B
    DB B-A
    ENDM
```

P267

◆ :=

- 为宏变元提供缺省值。

例7.9:

宏定义:

```
DIF3 MACRO A:=<10>,B:=<12>
    DB B-A
    ENDM
```

宏调用:

```
DIF3
```

宏展开:

```
1 DB 12-10
```

宏调用:

```
DIF3 5, 8
```

宏展开:

```
1 DB 8-5
```

注意：宏指令名与指令助记符或伪操作名相同时，宏指令定义的优先级最高，即同名的助记符或伪操作名被汇编程序认为是宏指令。

例：

宏定义：

```
add    MACRO  opr1, opr2, result
        .....
        ENDM
```

宏调用：

```
        .....
add     xx, yy, zz
purge   add                ; 取消宏定义
        .....
```

建议宏指令名与指令助记符或伪操作名尽量不要相同

宏定义

```
macro_name MACRO [哑元表]  
          [ LOCAL 标号表 ]  
          ..... (宏定义体)  
ENDM
```

7.1.3 LOCAL伪操作

- ◆ 当在宏定义体中使用了标号，多次调用该宏定义时，则展开后会出现标号的多重定义，这是不允许的。
 - LOCAL伪操作可以解决这个问题
- ◆ LOCAL伪操作格式： **LOCAL 局部标号表**
 - 局部标号表中的每个符号，在汇编时每扩展一次便建立一个唯一的标号，形如??xxxx（xxxx的值在0000~FFFF之间），以保证汇编时生成符号名字的惟一性
 - LOCAL伪操作只能用在宏定义体内，必须是MACRO伪操作后的第一个语句，在MACRO和 LOCAL伪操作之间，不允许有注释和分号标志

例7.10 求绝对值(使用LOCAL伪操作)

宏定义:

absol

MACRO

oper

LOCAL

next

cmp oper, 0

jge next

neg oper

next:

ENDM

宏展开:

.....

1 **cmp var, 0**

1 **jge ??0000**

1 **neg var**

1 **??0000:**

.....

宏调用:

.....

absol var

.....

absol bx

.....

1 **cmp bx, 0**

1 **jge ??0001**

1 **neg bx**

1 **??0001:**

.....

例：定义延时程序的宏指令，在同一个程序中两次被调用的扩展情况

宏定义：

```
DELAY  MACRO
        LOCAL  LOP
        MOV    CX, 2801
LOP:    LOOP   LOP
        ENDM
```

宏调用：

```
DELAY
DELAY
```

汇编时宏扩展如下：

```
MOV CX, 2801
1  ??0000: LOOP ??0000
MOV CX, 2801
1  ??0001: LOOP ??0001
```

7.1.4 在宏定义内使用宏

◆ 宏指令嵌套有两种情况：

- 宏定义体中含有宏调用
 - 必须先定义后调用
- 宏定义体中含有宏定义

例7.12

宏定义:

```
INT21 MACRO FUNCTION
    MOV AH, FUNCTION
    INT 21H
ENDM

DISP MACRO CHAR
    MOV DL, CHAR
    INT21 02H
ENDM
```

宏调用:

DISP ‘?’

宏展开:

```
1  MOV      DL, ‘?’
2  MOV      AH, 02H
2  INT      21H
```


例7.13

宏定义:

```
DEFMAC MACRO MACNAM, OPERATOR
    MACNAM MACRO X, Y, Z
        PUSH AX
        MOV AX, X
        OPERATOR AX, Y
        MOV Z, AX
        POP AX
    ENDM
ENDM
```

用宏调用形成加法宏定义:

```
DEFMAC ADDITION, ADD
```

形成加法宏定义:

```
ADDITION MACRO X, Y, Z
    PUSH AX
    MOV AX, X
    ADD AX, Y
    MOV Z, AX
    POP AX
ENDM
```

宏调用:

```
ADDITION VAR1, VAR2, VAR3
```

宏展开:

```
1  PUSH AX
1  MOV  AX, VAR1
1  ADD  AX, VAR2
1  MOV  VAR3, AX
1  POP  AX
```

7.1.5 列表伪操作

(自学 *p271*)

列表伪操作:

.LALL: 在LST清单中列出宏展开后的全部语句
(包括注释)。

.SALL: 在LST清单中不列出任何宏展开后的语句。

.XALL: 缺省的列表方式, 只列出宏体中产生目标
代码的语句。

列表伪指令只影响列表文件, 并不影响目标码的生成

7.1.6 宏库的建立与调用 (自学 P274)

建立宏库:

>EDIT MACRO.MAC

```
macro1 MACRO [哑元表]
```

```
.....  
ENDM
```

```
macro2 MACRO [哑元表]
```

```
.....  
ENDM
```

```
.....
```

```
macroN MACRO [哑元表]
```

```
.....  
ENDM
```

调用宏库:

>EDIT EXP.ASM

```
include MACRO.MAC
```

```
.....  
macro1 [实元表]
```

```
.....  
macro2 [实元表]
```

```
.....  
macroN [实元表]
```

```
.....
```

7.1.7 删除宏定义

◆ 格式:

PURGE **macro_name** [, **macro_name**, ...]

- 删除不再使用的宏定义，使该宏定义为空
- 删除后，汇编程序再遇到该宏调用指令将忽略，也不会指示出错

7.2 重复汇编

用于连续产生完全相同或基本相同的一组代码。

重复伪操作 REPT

REPT 表达式



;重复块

ENDM

不定重复伪操作 IRP/IRPC

IRP 哑元, <自变量表>



;重复块

ENDM

IRPC 哑元, 字符串



;重复块

ENDM

7.2.1 重复伪操作

◆ 重复伪操作 REPT

格式: REPT 表达式
 ; 重复块
 ENDM

- 表达式: 重复次数, 结果应该是无符号常数
- 不一定要用在宏定义中, 程序其他段中也可以用

例7. 15

```
X=0  
REPT 10  
X=X+1  
DB X  
ENDM
```

汇编展开后:

```
1 DB 1  
1 DB 2  
1 DB 3  
.....  
1 DB 10
```

例7.16 把字符 ‘A’到 ‘Z’ 的 ASCII 码填入数组TABLE

```
CHAR='A'  
TABLE LABEL BYTE  
      REPT 26  
      DB CHAR  
      CHAR=CHAR+1  
      ENDM
```

汇编展开后:

```
TABLE LABEL BYTE  
      1 DB 41H  
      1 DB 42H  
      1 DB 43H  
      .....  
      1 DB 5AH
```


7.2.2 不定重复伪操作

□ 不定重复伪操作 IRP / IRPC

1. IRP伪操作：用实际参量替换哑元

格式： IRP 哑元，〈自变量表〉
..... ； 重复块

ENDM

展开时，重复块中变量不定，无规律

- 每次重复用自变量表中的一项取代哑元，直到用完为止
- 重复次数由自变量的个数决定

例7. 20

```
IRP  REG, <AX,BX,CX,DX>  
    PUSH  REG  
ENDM
```

汇编展开后：

```
1  PUSH  AX  
1  PUSH  BX  
1  PUSH  CX  
1  PUSH  DX
```

例：在数据段产生字符区array，包括5个字符串 ‘NO.K’

```
data segment
array label byte
    IRP K, <1,2,3,4,5>
        db 'NO.&K'
    ENDM
data ends
```

汇编展开后

```
data segment
array label byte
1 db 'NO.1'
1 db 'NO.2'
1 db 'NO.3'
1 db 'NO.4'
1 db 'NO.5'
data ends
```

7.2.2 不定重复伪操作

2. IRPC伪操作：用字符串替换哑元

格式： IRPC 哑元，字符串
 ； 重复块
 ENDM

- 每次重复用字符串中的一个字符取代哑，直到用完为止
- 重复次数等于字符串中的字符数

例： 在数据段产生字符区array，包括5个字符串 ‘NO. K’

```
data segment
    array label byte
    IRPC K, 12345
    db 'NO.&K'
    ENDM
data ends
```

汇编展开后：





```
data segment
    array label byte
    1 db 'NO.1'
    1 db 'NO.2'
    1 db 'NO.3'
    1 db 'NO.4'
    1 db 'NO.5'
data ends
```

7.3 条件汇编

根据条件把一段源程序包括在汇编语言程序内或者排除在外。

一般格式：

IF ×× 自变量	；××为条件
	；自变量满足条件则汇编此块
[ELSE]	
	；自变量不满足条件则汇编此块
ENDIF	

条件伪操作：

{	IF	表达式	;表达式$\neq 0$，则汇编
	IFE	表达式	;表达式$= 0$，则汇编
{	IF1		;在第一遍扫视期间满足条件
	IF2		;在第二遍扫视期间满足条件
{	IFDEF	符号	;符号已定义，则汇编
	IFNDEF	符号	;符号未定义，则汇编
{	IFB	<自变量>	;自变量为空，则汇编
	IFNB	<自变量>	;自变量不为空，则汇编
{	IFIDN	<字符串1>,<字符串2>	;串1与串2相同
	IFDIF	<字符串1>,<字符串2>	;串1与串2不同

例7.24 求3个变元中最大值放入AX，且变元数不同时产生不同的程序段

宏展开:

宏定义:

```
MAX MACRO K,A,B,C
    LOCAL NEXT,OUT
    MOV AX,A
    IF K-1 ; 如果k-1≠0, 条件为真
    IF K-2
    {
        CMP C,AX
        JLE NEXT
        MOV AX,C
    }
    ENDIF
    NEXT: CMP B,AX
    JLE OUT
    MOV AX,B
    ENDIF
    OUT:
ENDM
```

宏调用:

```
MAX 1,P
MAX 2,P,Q
MAX 3,P,Q,R
```

1 MOV AX,P
1 ??0001:

1 MOV AX,P
1 ??0002: CMP Q,AX
1 JLE ??0003
1 MOV AX,Q
1 ??0003:

1 MOV AX,P
1 CMP R,AX
1 JLE ??0004
1 MOV AX,R
1 ??0004: CMP Q,AX
1 JLE ??0005
1 MOV AX,Q
1 ??0005:


```

MAX MACRO K,A,B,C
    LOCAL NEXT,OUT
    MOV AX,A
    IF K-1 ;如果k-1≠0, 条件为真
        IF K-2
            CMP C,AX
            JLE NEXT
            MOV AX,C
        ENDIF
    NEXT:    CMP B,AX
            JLE OUT
            MOV AX,B
        ENDIF
    OUT:
    ENDM

;
data segment
P        dw ?
Q        dw ?
R        dw ?
data ends
;
cseg      segment
assume cs:cseg,ds:data
start     proc far
;
        mov ax,data
        mov ds,ax
;
        MAX 1,P
        MAX 2,P,Q
        MAX 3,P,Q,R
;
exit:     mov ax,4c00h
        int 21h
start     endp
cseg      ends
end start

```

```

MAX MACRO K,A,B,C
    LOCAL NEXT,OUT
    MOV AX,A
    IF K-1 ;如果k-1≠0, 条件为真
        IF K-2
            CMP C,AX
            JLE NEXT
            MOV AX,C
        ENDIF
    NEXT:    CMP B,AX
            JLE OUT
            MOV AX,B
        ENDIF
    OUT:
    ENDM

;
0000      data segment
0000      ????      P      dw ?
0002      ????      Q      dw ?
0004      ????      R      dw ?
0006      data ends
;
0000      cseg      segment
        assume cs:cseg,ds:data
        start     proc far
;
0000      B8 ---- R      mov ax,data
0003      8E D8      mov ds,ax
;
0005      A1 0000 R      1      MAX 1,P
0008      1      1      ??0001:    MOV AX,P
;
0008      A1 0000 R      1      MAX 2,P,Q
000B      39 06 0002 R    1      MOV AX,P
000F      7E 03      1      ??0002:    CMP Q,AX
0011      A1 0002 R      1      JLE ??0003
0014      1      1      MOV AX,Q
;
0014      A1 0000 R      1      MAX 3,P,Q,R
0017      39 06 0004 R    1      MOV AX,P
001B      7E 03      1      CMP R,AX
001D      A1 0004 R      1      JLE ??0004
0020      39 06 0002 R    1      MOV AX,R
0024      7E 03      1      ??0004:    CMP Q,AX
0026      A1 0002 R      1      JLE ??0005
0029      1      1      MOV AX,Q
;
0029      B8 4C00      exit:      mov ax,4c00h
002C      CD 21      int 21h
002E      start     endp
002E      cseg      ends
end start

```

例7.25 根据跳转距离生成不同跳转指令

宏定义:

```
BRANCH  MACRO  X
    IF    ($-X) LT 128
        JMP SHORT X
    ELSE
        JMP NEAR PTR X
    ENDIF
ENDM
```

宏调用:

```
BRANCH AA
```

宏展开:

如果相对于AA距离小于
128, 宏展开

```
1  JMP SHORT AA
```

否则产生

```
1  JMP NEAR PTR AA
```

例7.26 在宏定义的递归调用中，使用条件伪操作结束宏递归

X和 2^N 相乘，即X左移N次

宏定义：

```
POWER MACRO X, N
    SAL X, 1
    COUNT=COUNT+1
    IF COUNT-N
    POWER X, N
    ENDIF
ENDM
```

宏调用：

```
COUNT=0
POWER AX, 3
```

宏展开：

```
1  SAL AX, 1
2  SAL AX, 1
3  SAL AX, 1
```

例 7.28 (p285):

divide **macro** dividend,divisor,quotient

local comp, out

cnt=0

ifndef dividend

cnt=1

endif

ifndef divisor

cnt=1

endif

ifndef quotient

cnt=1

endif

if cnt

exitm

endif

mov ax, dividend

mov bx, divisor

sub cx, cx

comp:

cmp ax, bx

jb out

sub ax, bx

inc cx

jmp comp

out:

mov quotient, cx

endm

7.4 高级语言结构

(自学 *p293*)

- ◆ MASM 6.0引入了几种更接近高级语言编程的高级语言结构，如以下标准宏指令
 - .IF/.ELSEIF/.ELSE/.ENDIF
 - .WHILE/.ENDW
 - .REPEAT/.UNTIL
 - .REPEAT/.UNTILECXZ
 - .BREAK
 - .CONTINUE
- ◆ 汇编程序将按标准指令段展开，形成一串指令完成特定操作

. IF/. ELSEIF/. ELSE/. ENDIF

格式:

```
. IF expression1  
  (汇编语言语句组1)  
. ELSEIF expression2  
  (汇编语言语句组2)  
. ELSEIF expression3  
  (汇编语言语句组3)  
  ⋮  
. ELSE  
  (汇编语言语句组n)  
. ENDIF
```

.IF宏指令在汇编时会产生比较(CMP)和条件跳转两条指令

汇编时这里产生一条**JMP**指令

```
.IF AL==“A”  
CALL DISP  
.ENDIF
```

```
CMP AL, “A”  
JNZ  NOTA  
CALL DISP
```

高级语言结构中使用的表达式

◆ 表达式中的操作符

机器指令中有对应的操作

- == 相等
- != 不等
- > 大于
- >= 大于或等于
- < 小于
- <= 小于或等于
- & 位测试
- ! 逻辑非
- && 逻辑与
- || 逻辑或

◆ 表达式格式

1) 测试条件码的值 (=1时表达式值为真)

- ZERO?
- CARRY?
- OVERFLOW?
- SIGN?
- PARITY?

机器指令中有对应的条件转移

2) 操作数构成的表达式

- reg op reg
- reg op memory
- reg op constant
- memory op constant

机器指令中有对应的操作数表示形式

.IF reg == constant



CMP reg, constant
JNZ xxxxx

- 高级语言语句在汇编时汇编程序将生成标准的机器指令串替代高级语言语句

更加易于编程和可读性

```
.IF AL=="A"  
    CALL DISP  
.ELSEIF AL=="B"  
    CALL DISP  
.ELSE  
    MOV AL, "N"  
    CALL DISP  
.ENDIF
```

汇编展开

```
CMP AL, "A"  
JNZ NOTA  
CALL DISP  
JMP DONE  
NOTA: CMP AL, "B"  
JNZ NOTB  
CALL DISP  
JMP DONE  
NOTB: MOV AL, "N"  
CALL DISP  
DONE:
```

例 7.32 (p294)

练习举例

P301 7.1

7.1 定义宏指令CLRB，完成用空格符将一字符区中的字符取代的工作。字符区的首地址及其长度为变元。（两种实现方案）

```
CRLB1  MACRO  ADDR, CUNT
        LOCAL  NEXT
        MOV    SI, OFFSET  ADDR
        MOV    AL, 20H  ;
        ‘ ’ =20H
        MOV    CX, CUNT
NEXT:   MOV    [SI], AL
        INC    SI
        LOOP   NEXT
        ENDM
```

```
CRLB2  MACRO  ADDR, CUNT
        LEA    DI, ADDR
        MOV    AL, ‘ ’
        CLD                                ; DF=0
        MOV    CX, CUNT
        REP    STOSB
        ENDM
```

7.5.1 ARM64宏定义和调用

◆ 宏定义格式:

```
// 定义了一个名为mac_name的宏  
.macro  mac_name  
    ...  
.endm
```

```
// 定义了一个名为mac_name2的宏  
// 并且有2个输入参数  
.macro  mac_name2 p1, p2  
    ...  
.endm
```

◆ 说明:

- .macro和.endm是ARM中定义宏的伪指令
- 宏定义中，可以带有参数。
 - 参数之间用“,”或空格分开
 - 参数可以通过“=deflt”的格式提供缺省值
 - 若参数为必须提供，可以在参数后使用“:req”
 - 当参数数量可变时，最后一个参数可使用“:vararg”
- 宏定义体中使用参数时要在参数前加“\”

7.5.1 ARM64宏定义和调用

```
.macro  resv_str p1=0, p2
    ...
.endm
```

◆ 宏定义举例：

- 宏resv_str带有2个参数
- 参数p1有一个缺省值为0，第2个参数没有提供缺省值
- 在宏定义体中通过“\p1”的方式使用第1个参数p1

◆ 宏调用

- 宏调用时，参数按宏定义时的参数顺序一一对应。也可以使用键值对的方式指令参数值
 - 在宏调用 resv_str 9, 17 中，参数p1=9, p2=17
 - 在宏调用 resv_str p2=17, p1=9 中，参数p1=9, p2=17

◆ 宏定义体中参数值传送

- 如果有宏调用 resv_str x, y, 则在宏定义体内的参数“\p1”会被x替代，“\p2”会被y替代
- 而宏调用 resv_str , y后，“\p1”会被缺省值0替代

7.5.1 ARM64宏定义和调用

◆ store宏定义（编程时）：

- 类似于x86，把一个寄存器的值存储到数据段定义的一个双字变量单元

```
// store宏定义。p1为64位寄存器
// p2为数据段定义的变量
.macro store p1, p2
    adr    x29, \p2
    str    \p1, [x29]
.endm
```

◆ 宏调用（编程时）：

```
store    x1, Z    //Z在数据段定义
```

◆ 宏展开（汇编时）：

```
12                                store x0,Z
12 0010 1D000010                >  adr x29,Z
12 0014 A00300F9                >  str x0,[x29]
```

- “>” 表示宏展开1次
- 宏展开2次用 “>>” 表示

7.5.1 ARM64宏定义和调用

◆ multiply、store宏举例完整代码

```
.macro store p1, p2
    adr x29, \p2
    str \p1, [x29]
.endm
// multiply宏, 使用了store宏
.macro multiply p1, p2, result
    ldr x1, \p1
    ldr x2, \p2
    mul x0, x1, x2
    store x0, \result
.endm

.data
X: .dword 12
Y: .dword 15
Z: .dword 0
```

```
// 主程序, 演示宏的展开
.text
.global main
main:
    stp x29, x30, [sp, #-16]!
    // 直接用数据段定义的变量
    // 实现  $Z = X * Y$  操作
    multiply X, Y, Z
    ldp x29, x30, [sp], #16
    ret
```

◆ 宏举例完整.lst文件内容

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14 0000 0C000000
14      00000000
15 0008 0F000000
15      00000000
16 0010 00000000
16      00000000
17
18
19
20
21 0000 FD7BBFA9
22
22 0004 01000058
22 0008 02000058
22 000c 207C029B
22
22 0010 1D000010
22 0014 A00300F9
23 0018 FD7BC1A8
24 001c C0035FD6

        .macro store p1, p2
            adr x29, \p2
            str \p1, [x29]
        .endm

        .macro multiply p1, p2, result
            ldr x1, \p1
            ldr x2, \p2
            mul x0, x1, x2
            store x0, \result
        .endm

        .data
            X: .dword 12
            Y: .dword 15
            Z: .dword 0

        .text
        .global main
main:
        stp x29, x30, [sp, #-16]!
        multiply X, Y, Z
> ldr x1,X
> ldr x2,Y
> mul x0,x1,x2
> store x0,Z
>> adr x29,Z
>> str x0,[x29]
        ldp x29, x30, [sp], #16
        ret
```

7.5.2 ARM64宏定义中的参数

◆ (1) 宏定义可以无变元:

- enter_func宏, 将X29, X30寄存器压入堆栈
- exit_func宏, 从堆栈中恢复X29, X30寄存器的值, 并且执行ret指令

```
// 进入函数后, 保存X29, X30寄存器
.macro enter_func
    stp x29, x30, [sp, #-16]!
.endm
// 函数返回前, 恢复X29, X30寄存器
.macro exit_func
    ldp x29, x30, [sp], #16
    ret
.endm
```

◆ 宏展开后(.lst文件)

```
11          main:
12          enter_func
12 0000 FD7BBFA9  > stp x29,x30,[sp,#-16]!
13 0004 200080D2    mov x0, #1
14          exit_func
14 0008 FD7BC1A8  > ldp x29,x30,[sp],#16
14 000c C0035FD6  > ret
```

7.5.2 ARM64宏定义中的参数

◆ (2) 变元可以是操作码或操作码的一部分：

■ 数据定义INIT_ARRAY宏

```
.macro INIT_ARRAY name, suffix, dtype, first, value:vararg
    \name\suffix: .\dtype \first, \value
.endm
```

```
.data
    INIT_ARRAY arr, 0, word, 0, 0x1234, 0x246
    INIT_ARRAY arr, 1, byte, 2, 4
```

◆ 宏展开后(.lst文件)

```
5          .data
6          INIT_ARRAY arr, 0, word, 0, 0x1234, 0x246
6 0000 00000000      > arr0:.word 0,0x1234,0x246
6          34120000
6          46020000
7          INIT_ARRAY arr, 1, byte, 2, 4
7 000c 0204          > arr1:.byte 2,4
```


7.5.2 ARM64宏定义中的参数

◆ (2) 变元可以是操作码或操作码的一部分:

- 类似x86的jmp宏，根据条件进行跳转
- 第1个参数是条件码，与b指令组成ARM64的条件跳转指令
- 第2个参数是跳转的目标地址

◆ 宏展开后(.lst文件)

```
.macro jmp cc dest
    b\cc \dest
.endm
```

```
.text
...
mov x0, #0
mov x1, 0x0
cmp x0, x1
jmp eq next
jmp gt big
mov x0, #-1
b    next
...
```

```
11 000c 1F0001EB          cmp x0, x1
12                        jmp eq next
12 0010 A0000054      >    beq next
13                        jmp gt big
13 0014 6C000054      >    bgt big
14 0018 00008092          mov x0, #-1
15 001c 02000014          b    next
```

7.5.3 ARM64宏定义中的标号

- ◆ 当在宏定义体中使用了标号，多次调用该宏定义时，则展开后会出现标号的多重定义，这是不允许的。
- ◆ 在使用GNU as对ARM进行汇编时，可以使用纯数字格式定义标号，“N:”，其中N代表任意非负整数。
 - 如要引用某个标号的之前最近一次定义，应使用“Nb”，N应与标号在定义时的数字相同；
 - 如要引用某个标号的下一次定义，应使用“Nf”。
 - ‘b’表示“向后引用”，‘f’表示“向前引用”。

```
1: branch 1f  
2: branch 1b  
1: branch 2f  
2: branch 1b
```

等价于

```
label_1: branch label_3  
label_2: branch label_1  
label_3: branch label_4  
label_4: branch label_3
```

在宏定义中，只能使用数字格式的标号。

7.5.3 ARM64宏定义中的标号

- ◆ 举例：NUMABS宏，求一个数的绝对值

核心源代码

```
.macro NUMABS oper
    cmp \oper, #0
    bge 1f
    neg \oper, \oper
1:
.endm

.text
...
mov x0, #-4
NUMABS x0
mov x1, 0x12
NUMABS x1
mov x2, #-128
NUMABS x2
...
```

对应的.lst代码

```
12 0004 60008092          mov x0, #-4
13                          NUMABS x0
13 0008 1F0000F1    >    cmp x0,#0
13 000c 4A000054    >    bge 1f
13 0010 E00300CB    >    neg x0,x0
13                          > 1:
14 0014 410280D2          mov x1, 0x12
15                          NUMABS x1
15 0018 3F0000F1    >    cmp x1,#0
15 001c 4A000054    >    bge 1f
15 0020 E10301CB    >    neg x1,x1
15                          > 1:
16 0024 E20F8092          mov x2, #-128
17                          NUMABS x2
17 0028 5F0000F1    >    cmp x2,#0
17 002c 4A000054    >    bge 1f
17 0030 E20302CB    >    neg x2,x2
17                          > 1:
```

7.5.4 ARM64重复汇编

- ◆ 重复伪操作 `.rept`
- ◆ 格式: `.rept count`
 ... // 重复块
 `.endr`
- ◆ 例如, 将长度为3的数组arr中各元素初始化为0x1234

```
.data
arr:
    .rept 3
        .word 0x1234
    .endr

.text
...
ldr w0, arr
ldr w1, arr+4
ldr w2, arr+8
```

```
1      .data
2      arr:
3          .rept 3
3 0000 34120000 > .word 0x1234
3          >
3 0004 34120000 > .word 0x1234
3          >
3 0008 34120000 > .word 0x1234
4          .word 0x1234
5          .endr
6
7      .text
```

7.5.4 ARM64重复汇编

- ◆ 重复伪操作 `.rept`
- ◆ 例如，在数据段定义一个长度为26的ASCII大写字母表/数组，数组内数据依次为字符 'A'、... 'Z'

```
.data
table:
    ch = 'A'
    .rept 26
        .byte ch
        ch = ch + 1
    .endr
.text
    adr x0, table
    ...
```

```
1      .data
2      table:
3          ch = 'A'
4          .rept 26
4 0000 41      > .byte ch
4          > ch = ch+1
4          >
4 0001 42      > .byte ch
4          > ch = ch+1
4          >
4 0019 5A      > .byte ch
4          > ch = ch+1
5              .byte ch
6              ch = ch + 1
7          .endr
8
9      .text
13 0004 00000010 adr x0, table
    ...
```

26个

- ◆ 用gdb显示数据段内存分配

```
(gdb) x/26xb $x0
```

0x420028:	0x41	0x42	0x43	0x44	0x45	0x46	0x47	0x48
0x420030:	0x49	0x4a	0x4b	0x4c	0x4d	0x4e	0x4f	0x50
0x420038:	0x51	0x52	0x53	0x54	0x55	0x56	0x57	0x58
0x420040:	0x59	0x5a						

7.5.4 ARM64重复汇编

- ◆ 不定重复伪操作 .irp
- ◆ 格式: **.irp** param, <语句序列>
... // 重复块

语句序列中的元素间
使用 ‘,’ 或空格分开

.endr

- ◆ 如果没有列出值, 语句序列将被汇编一次, 并将符号设置为空字符串。要引用语句序列中的符号, 使用\param。
- ◆ 例如, 给X0, X1, X2, 赋初始值0x1234
- ◆ 源代码如右图:

```
.text
.global main
main:
    stp x29, x30, [sp, #-16]!
    .irp param, 0, 1, 2
    mov X\param, 0x1234
    .endr
    ldp x29, x30, [sp], #16
    ret
```

7.5.4 ARM64重复汇编

- ◆ 不定重复伪操作 .irp
- ◆ 例如，给X0, X1, X2, 赋初始值0x1234

- ◆ .lst文件

```
4                               main:
5 0000 FD7BBFA9                stp x29, x30, [sp, #-16]!
6                               .irp param, 0, 1, 2
6 0004 804682D2                > mov X0,0x1234
6                               >
6 0008 814682D2                > mov X1,0x1234
6                               >
6 000c 824682D2                > mov X2,0x1234
7                               mov X\param, 0x1234
8                               .endr
9 0010 FD7BC1A8                ldp x29, x30, [sp], #16
10 0014 C0035FD6               ret
```

- ◆ 用gdb反汇编后的代码

```
<+0>:    stp    x29, x30, [sp, #-16]!
<+4>:    mov    x0, #0x1234
<+8>:    mov    x1, #0x1234
<+12>:   mov    x2, #0x1234
```

7.5.4 ARM64重复汇编

- ◆ 不定重复伪操作 `.irpc`
- ◆ 格式: `.irpc param, <字符串>`
... // 重复块

`.endr`

- ◆ 对于值中的每个字符, 将符号设置为该字符, 然后汇编语句序列。
- ◆ 如果没有列出值, 语句序列将被汇编一次, 并将符号设置为空字符串。要引用语句序列中的符号, 使用 `\param`。
- ◆ 例如, 给 `X0`, `X1`, `X2`, 赋初始值 `0x1234`
- ◆ 源代码如右图:
 - ◆ 对应的 `.lst` 文件和反汇编后的代码和上页 `.irp` 的例子类似

```
.text
.global main
main:
    stp x29, x30, [sp, #-16]!
    .irpc param, 012
    mov X\param, 0x1234
    .endr
    ldp x29, x30, [sp], #16
    ret
```


7.5.5 ARM64条件汇编

- ◆ 条件伪操作能使汇编程序根据条件把一段源程序包括在目标程序之内，或者把它排除在外
- ◆ 条件伪操作的一般格式：

.if <逻辑表达式>

... ; 逻辑表达式为真则汇编此块

{.else

... ; 逻辑表达式为假则汇编此块}

.endif

- .if后的逻辑表达式决定是否对后面的指令进行汇编，如果逻辑表达式为假，就会对.else后的指令进行汇编。
- .endif标志着条件汇编的结束。

7.5.5 ARM64条件汇编

◆ 举例：X和 2^N 相乘，即X左移N次

◆ 源代码：

- power宏为嵌套调用，每调用1次，将X左移1位，实现乘2的效果
- 当power宏的调用次数（用count统计）等于n后，表达式(n-count)等于0，即条件汇编.if的表达式为假，停止嵌套调用。

```
.macro power reg, n
    lsl \reg, \reg, #1
    count = count + 1
    .if (\n - count)
        power \reg, \n
    .endif
.endm

.text
.global main
main:
    stp x29, x30, [sp, #-16]!
    count = 0
    mov x0, #4
    power x0, 3
    ldp x29, x30, [sp], #16
    ret
```

7.5.5 ARM64条件汇编

- ◆ 举例：X和 2^N 相乘，即X左移N次
- ◆ 对应的.lst文件：

```
11                                     main:
12 0000 FD7BBFA9                    stp x29, x30, [sp, #-16]!
13                                count = 0
14 0004 800080D2                    mov x0, #4
15                                power x0, 3
15 0008 00F87FD3                    > lsl x0,x0,#1
15                                > count =count+1
15                                > .if (3-count)
15                                > power x0,3
15 000c 00F87FD3                    >> lsl x0,x0,#1
15                                >> count =count+1
15                                >> .if (3-count)
15                                >> power x0,3
15 0010 00F87FD3                    >>> lsl x0,x0,#1
15                                >>> count =count+1
15                                >>> .if (3-count)
15                                >>> power x0,3
15                                >>> .endif
15                                >> .endif
15                                > .endif
16 0014 FD7BC1A8                    ldp x29, x30, [sp], #16
17 0018 C0035FD6                    ret
```

7.5.5 ARM64条件汇编

- ◆ 举例：X和 2^N 相乘，即X左移N次
- ◆ 使用gdb对该可执行文件进行调试：
- ◆ 使用disas反汇编指令，得到main函数对应的反汇编代码如下图所示。
 - 条件汇编指令仅在汇编时对汇编器有用，没有生成机器指令。

```
(gdb) disas
Dump of assembler code for function main:
=> 0x00000000004005d4 <+0>:      stp     x29, x30, [sp, #-16]!
    0x00000000004005d8 <+4>:      mov     x0, #0x4
    0x00000000004005dc <+8>:      lsl     x0, x0, #1
    0x00000000004005e0 <+12>:     lsl     x0, x0, #1
    0x00000000004005e4 <+16>:     lsl     x0, x0, #1
    0x00000000004005e8 <+20>:     ldp     x29, x30, [sp], #16
    0x00000000004005ec <+24>:     ret
End of assembler dump.
```