



3.6 子程序设计



3.6 子程序设计：子程序的作用

◇ 编程计算下面表达式的值

$$S = \sqrt{2X} + \sqrt{3Y} + \sqrt{150}$$

3.6 子程序设计：子程序概念

- ◇ 把功能相对独立的程序段单独编写和调试，作为一个模块供程序使用，就形成子程序。
- ◇ 子程序可以实现源程序的模块化，可简化源程序结构，可以提高编程效率。
- ◇ 主程序使用**CALL**指令调用子程序。
- ◇ 子程序利用**RET**指令返回主程序。

3.6.1 过程定义和子程序编写

- ◇ 汇编语言中，子程序要用一对过程定义伪指令**PROC**和**ENDP**声明，格式如下：

过程名 **PROC** [**NEAR**|**FAR**]
 ; 过程体
过程名 **ENDP**

- ◇ 可选的参数指定过程的调用属性，没有指定过程属性，则采用默认属性
 - ◆ **NEAR**属性的过程只能被相同代码段的其他程序调用，称为段内近调用。
 - ◆ **FAR**属性的过程可以被相同或不同代码段的程序调用，称为段间远调用。

子程序编写注意事项

wj0316

- (1) 子程序要使用过程定义伪指令声明
- (2) 主程序执行**CALL**指令调用子程序，子程序最后利用**RET**指令返回主程序
- (3) 子程序开始应该保护使用到的寄存器内容，子程序返回前相应进行恢复
- (4) 子程序中对堆栈的压入和弹出操作要成对使用，保持堆栈的平衡
- (5) 子程序定义应安排在代码段的主程序之外，最好放在主程序执行终止后的位置（返回**DOS**后、汇编结束**END**伪指令前），也可以放在主程序开始执行之前的位置

例3.15 用显示器功能调用输出一个字符的子程序


；子程序：显示AL中的字符

```
dpchar  proc           ; 过程定义，过程名为dpchar
        push ax        ; 入栈保护寄存器
        push bx
        mov bx,0
        mov ah,0eh     ; 显示器0EH号输出一个字符功能
        int  10h
        pop  bx        ; 逆序出栈，恢复寄存器
        pop  ax
        ret            ; 子程序返回
dpchar  endp           ; 过程结束
```

；主程序

```
        mov al,'?'     ; 主程序提供显示字符
        call dpchar    ; 调用子程序
```

例3.15完整的源程序




```
; wj0315.asm
.model small
.stack
.code
start:  mov al,'?'           ; 主程序提供显示字符
        call dpchar         ; 调用子程序
        mov ax,4c00h
        int 21h
```

子程序代码可以安排在这个位置



例3.15源程序（续）





```
dpchar  proc                ; 过程定义，过程名为dpchar
        push ax             ; 顺序入栈，保护寄存器
        push bx
        mov bx,0
        mov ah,0eh          ; 显示器0EH号输出一个字符功能
        int 10h
        pop bx              ; 逆序出栈，恢复寄存器
        pop ax
        ret                 ; 子程序返回
dpchar  endp                ; 过程结束
        end start
```

子程序安排在主程序执行终止后的位置




子程序编写注意事项（续）

- 
- (6) 子程序允许嵌套和递归
 - (7) 子程序可以与主程序共用一个数据段，也可以使用不同的数据段（注意修改**DS**），还可以在子程序最后设置数据区（利用**CS**寻址）
 - (8) 子程序的编写可以很灵活，例如具有多个出口（多个**RET**指令）和入口，但一定要保证堆栈操作的正确性
 - (9) 处理好子程序与主程序间的参数传递问题
 - (10) 提供必要的子程序说明信息
- 

子程序说明信息



子程序说明包括以下几项内容：

- ①子程序名；
 - ②子程序功能；
 - ③入口条件；
 - ④出口条件；
 - ⑤受影响的寄存器。
- 

例3.16 显示以“0”结尾字符串的嵌套子程序



```
    ; 数据段
msg  db 'Well, I made it !',0
    ; 代码段（主程序）
mov  si,offset msg  ; 主程序提供字符串地址
call dpstri         ; 调用子程序
```

Wj0316.asm



例3.16 子程序

；子程序dpstri：显示DS:SI指向的字符串（以0结尾）

```
dpstri  proc
        push ax
dps1:   mov al,[si]      ; 取串中字符
        inc si
        cmp al,0         ; 是结尾，则显示结束
        jz done
        call dpchar      ; 调用字符显示子程序
        jmp dps1
done:   pop ax
        ret
dpstri  endp
```

；子程序dpchar：显示AL中的字符（同[例题3.15](#)）

含数据区的子程序举例

；子程序HTOASC：用查表法将AL低四位转换为其十六进制
；形式数值对应的ASCII码。

```
HTOASC proc
    push bx
    mov  bx,offset ASCII
    and  al,0fh
    xlat  CS:ASCII    ; 换码：AL←CS:[BX + AL]
    pop  bx
    ret
```

；数据区

```
ASCII    db 30h,31h,32h,33h,34h,35h,36h,37h,38h,39h
          db 41h,42h,43h,44h,45h,46h
```

```
HTOASC endp
```

Wj0304s.asm

多出口子程序举例

；子程序HTOASC：十六进制数转换为ASCII码

```
HTOASC  proc
        and  al,0fh
        cmp  al,9
        jbe  htoasc1
        add  al,37h    ; 是A ~ F, 加37H
        ret           ; 子程序返回
htoasc1: add  al,30h    ; 是0 ~ 9, 加30H
        ret           ; 子程序返回
HTOASC  endp
```

参数传递

- ◇ 子程序设计的一个主要问题是实现主程序与子程序之间的参数传递
 - ◆ 入口参数（输入参数）：主程序调用子程序时，提供给子程序的参数
 - ◆ 出口参数（输出参数）：子程序执行结束返回给主程序的参数
- ◇ 参数的具体内容
 - ◆ 传数值：传送数据本身
 - ◆ 传地址：传送数据的主存地址
- ◇ 常用的参数传递方法
 - ◆ 寄存器
 - ◆ 共享变量
 - ◆ 堆栈

3.6.2 用寄存器传递参数

- ◇ 最简单和常用的参数传递方法是通过寄存器，只要把参数存于约定的寄存器中就可以了。
- ◇ 由于通用寄存器个数有限，这种方法对少量数据可以直接传递数值，而对大量数据只能传递地址。
- ◇ 采用寄存器传递参数，注意带有出口参数的寄存器不能保护和恢复，带有入口参数的寄存器可以保护、也可以不保护，但最好能够保持一致。



dpchar



dpstri



HTOASC

3.6.2 用寄存器传递参数

- ◇ If you are passing a single parameter to a procedure you should use the following registers for the accompanying data types

Data Size	Pass in this Register
Byte	al
Word	ax
Double Word	dx:ax or eax (if 80386 or better)

- ◇ If you are passing several parameters to a procedure in the 80x86's registers, you should probably use up the registers in the following order

First	Last
ax, dx, si, di, bx, cx	

例3.17 用寄存器传递参数显示字符串

; 数据段

msg db 'Well, I made it !',0

; 代码段（主程序）

mov si,offset msg

;SI寄存器传递参数：字符串地址

call dpstri ;调用子程序

子程序在下一页

例3.17 用寄存器传递参数显示字符串（续）

；代码段（子程序）

dpstri	proc	；显示以0结尾的字符处
	push ax	；入口参数：SI = 字符串地址
	push dx	
dps1:	mov dl,[si]	；通过SI使用参数
	cmp dl,0	
	jz dps2	
	mov ah,2	
	int 21h	
	inc si	
	jmp dps1	
		dps2: pop dx
		pop ax
		ret
		dpstri endp

3.6.2 用寄存器传递参数

例3.18 编写子程序，实现从键盘接收一个有符号十进制数的功能。输入时负数用“-”引导，正数用“+”引导或直接输入数值。设数据范围为-32768 ~ 32767。

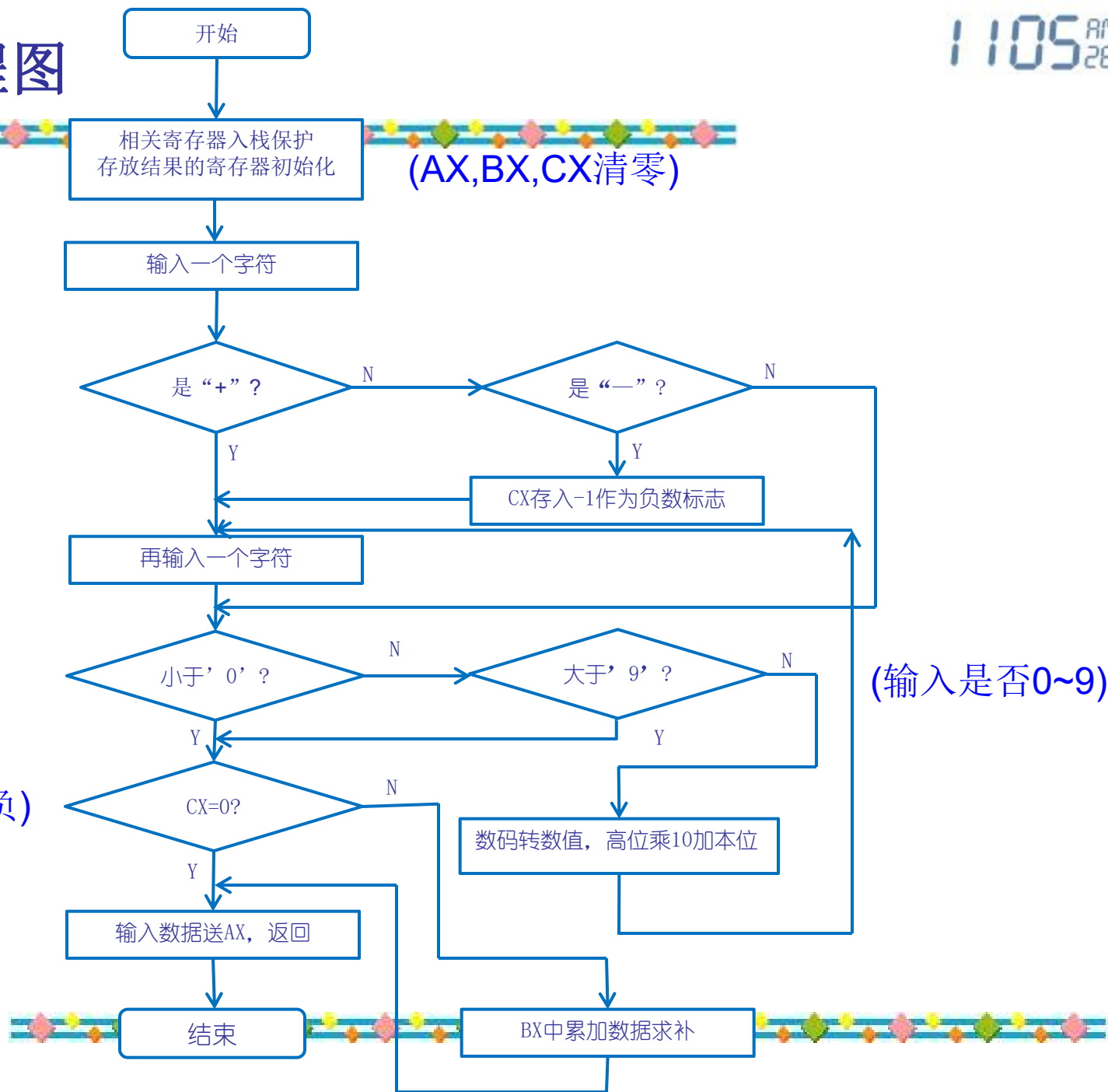
输入数据示例：123， +369， -987

例3.18 算法分析

- ① 首先判断输入的是正数还是负数，并用一个寄存器记录下来，如正数**CX**存入0，负数存入-1；
- ② 接着输入0 ~ 9数字（从键盘得到的是**ASCII**码），减**30H**后转换为二进制数；
- ③ 如果有新数位输入，将前面输入的数值乘以10，并与刚输入的数字相加得到新的数值；
- ④ 重复②、③步，直到输入一个非数字字符结束；
- ⑤ 如果是负数进行求补，转换成补码；否则直接将数值保存。因为数值范围-32768~32767可用16位二进制表示，约定输入数据在**AX**寄存器中。

-	1	2	3
---	---	---	---


子程序流程图



例3.18 从键盘输入有符号十进制数（1/3）


read	proc	;输入有符号十进制数	Wj0320.asm
	<u>push bx</u>	;出口参数: AX	
	push cx	;说明: 负数用“-”引导	+123
	push dx		123
	xor bx,bx	;BX清零, 用来保存中间结果	- 123
	xor cx,cx	;CX存正负标志, 正存0, 负存-1	
	mov ah,1	;开始输入一个字符	
	int 21h		
	cmp al, '+'	;首先进行符号判别, 是“+”吗	
	jz <u>read1</u>	;是“+”则转read1继续输入数字, 如+12	
	cmp al, '-'	;否则判断是否为“-”	
	jnz <u>read2</u>	;也不是负号则认为首次输入为数字符	
	mov cx,-1	;是“-”则向CX中存入-1作为符号标记	

例3.18 从键盘输入有符号十进制数（2/3）




```
read1:  mov ah,1          ;接收数值部分输入直到结束符
        int 21h


read2:  cmp al, '0'       ;判断输入的是数字还是结束符
        jb read3         ;不是0~9之间的字符，输入结束
        cmp al, '9'
        ja read3
        sub al, 30h       ;是0~9之间的字符，转换为二进制数
        ;利用移位指令，实现数值乘10:  $BX \leftarrow BX \times 10$ 
        shl bx,1          ; $bx \times 2$ ，2个时钟周期，MUL 70时钟周期
        mov dx,bx         ; $dx = bx \times 2$ ，2个时钟周期
        shl bx,1          ; $bx \times 4$ 
        shl bx,1          ; $bx \times 8$ 
        add bx,dx         ; $bx \times 8 + bx \times 2 = bx \times 10$ ，3个时钟周期
```




例3.18 从键盘输入有符号十进制数（3/3）



```
mov ah,0    ;
add bx,ax    ;新输入的数位在AL中
             ;已输入数值乘10后，与新输入数值相加
jmp read1    ;继续输入字符
read3: cmp cx,0    ;输入结束符，判数据符号，是"+"吗
      jz read4    ;如果符号是"+", 数据直接送出口寄存器
      neg bx      ;是负数，进行求补
read4: mov ax,bx    ;输入数据送出口寄存器
      pop dx
      pop cx
      pop bx
      ret          ;子程序返回
read  endp
```



例3.18 从键盘输入有符号十进制数的子程序



；主程序调用子程序输入整数，先定义数据段

count = 10 ;常量定义

array dw count dup(0) ;定义整形数组，10个元素

；代码段中主程序部分，循环调用子程序

mov cx,count ;循环次数初始化

mov bx,offset array ;指针变量初始化

again: call read ;调用子程序，输入一个整数


mov [bx],ax ;子程序返回值在ax中，存入数组

inc bx

inc bx


call dpcrlf ;调用回车换行子程序

loop again ;在下一行继续输入




Wj0320.asm

例3.18 从键盘输入有符号十进制数（续4）



```
dpcrlf    proc                ;使光标回车换行的子程序
           push ax
           push dx
           mov ah,2
           mov dl,0dh          ;回车
           int 21h
           mov ah,2
           mov dl,0ah          ;换行
           int 21h
           pop dx
           pop ax
           ret
dpcrlf    endp
```



3.6.3 用共享变量传递参数

- ◇ 利用共享变量(全局变量)进行参数传递是指子程序和主程序使用同一个变量存取数据。
- ◇ 如果变量定义和使用不在同一个源程序中，需要利用**PUBLIC**、**EXTERN**声明。wj0323
- ◇ 如果主程序还要利用原来的变量值，则需要保护和恢复。
- ◇ 利用共享变量传递参数，子程序的通用性较差，但特别适合在多个程序段间、尤其在不同的程序模块间传递数据

例3.19 用共享变量传递参数显示字符串

wj0318

；假设串的首地址用共享变量传递

；数据段

msg db 'Well, I made it !',0

temp dw ? ;共享变量，存串地址

；代码段，主程序

mov si,offset msg

mov temp,si ;共享变量传递参数

call dpstri ;调用子程序

例3.19 用共享变量传递参数显示字符串（续）

1105 AM
26

; 代码段，子程序


dpstri	proc	;显示以0结尾的字符串
	push ax	;入口参数：temp = 字符串地址
	push dx	
	mov si,temp	;通过temp获得参数
	;后同例3.16A程序
	ret	
dpstri	endp	

3.6.3 用共享变量传递参数

例3.20 编写子程序在屏幕上显示一个有符号十进制整数，负数用“-”引导，整数的数值范围为-32768~32767。

编写主程序调用该子程序显示输出10个数据，要求主、子程序之间用共享变量传递参数。

例3.20 子程序算法分析

- 
- ① 首先判断数据是零、正数还是负数，是零显示“0”退出；
 - ② 是负数，则先显示“-”，求数据的绝对值；
 - ③ 接着数据除以10，余数加30H转换为ASCII码压入堆栈；
 - ④ 重复③步，直到商为0结束；
 - ⑤ 依次从堆栈弹出各位数字，进行显示。

Wj0321.asm




例3.20 向显示器输出有符号十进制数（1/4）

;显示有符号10进制数的通用子程序


write	proc	;入口参数：共享变量wtemp
	<u>push ax</u>	; ax存放待处理参数
	push bx	;存放ax数据的备份
	push dx	;2号DOS功能调用使用dx
	mov ax, <u>wtemp</u>	;从共享变量wtemp取出显示数据
	test ax,ax	;判断数据是零、正数或负数
	jnz <u>write1</u>	;不为0，转write1处区分正负
	mov dl,'0'	;是零，显示“0”后退出
	mov ah,2	
	int 21h	
	jmp <u>write5</u>	;write5，子程序返回

以-123的显示为例

例3.20 向显示器输出有符号十进制数（2/4）




```
write1:  jns write2      ;是正数, 转write2
         mov bx,ax       ;是负数,显示"-", AX数据暂存于BX
         mov dl,'-'
         mov ah,2
         int 21h
         mov ax,bx
         neg ax          ;负数求补得到其绝对值
write2:  mov bx,10        ;write2对正数或负数绝对值进行转换
         push bx         ;数的位数不同, 将10压入堆栈, 从堆栈
取出数据进行显示时, 出栈数据为10标志一个数据显示结束
```




例3.20 向显示器输出有符号十进制数（3/4）

1105 AM
26



write3:	<code>cmp ax,0</code>	;AX值（商）为0 ?
	<code>jz write4</code>	;为0表示所有数位已转换为ASCII码
	<code>sub dx,dx</code>	;扩展被除数DX.AX
	<code>div bx</code>	;数据除以10: $DX.AX \div 10$
	<code>add dl,30h</code>	;余数（0 ~ 9）转换为ASCII码
	<code>push dx</code>	;数据各位先低位后高位压入堆栈
	<code>jmp write3</code>	
write4:	<code>pop dx</code>	;数据各位先高位后低位弹出堆栈
	<code>cmp dl,10</code>	;是结束标志10，则退出
	<code>je <u>write5</u></code>	



例3.20 向显示器输出有符号十进制数（4/4）

mov ah,2 ;进行显示

int 21h

jmp write4

write5: pop dx

pop bx

pop ax

ret ;子程序返回

write endp



例3.20 向显示器输出有符号十进制数

; 主程序, 数据段

count = 10

array dw 1234,-1234,0 ...

wtemp dw ?

; 主程序, 代码段

mov cx,count ;循环次数, 数据个数

mov bx,offset array ;指针变量赋初值

again: mov ax,[bx]

mov wtemp,ax ;将入口参数存放到共享变量

call **write** ;调用子程序, 显示一个数据

inc bx

inc bx

call **dpcrlf** ;光标回车换行

loop again

3.6.4 用堆栈传递参数

- ◇ 主、子程序之间还可以通过堆栈传递参数，过程如下。
 - ◆ 主程序将入口参数压入堆栈，子程序从堆栈中取出参数；
 - ◆ 子程序将出口参数压入堆栈，主程序则通过出栈操作取得它们
- ◇ 采用堆栈传递参数是程式化的，它是编译程序处理参数传递、以及汇编语言与高级语言混合编程时的常规方法。

3.6.4 用堆栈传递参数—实例1

例3.21 编制显示以0结尾的字符串的子程序，用堆栈传递参数。

```
msg db 'Well, I made it !',0
```

Wj0319.asm

3.6.4 用堆栈传递参数—实例2

例3.22 编制计算整型数组平均值的子程序，入口参数为数组元素个数和数据缓冲区的首地址，用堆栈传递；出口参数为数组元素的平均值，用AX寄存器传递。

Wj0322.asm

3.6.5 子程序模块

◇ 将子程序单独编写成一个源程序文件，经过汇编之后形成目标模块**OBJ**文件，连接时应用。

(1) 使用伪指令**PUBLIC**和**EXTERN**声明共享

例3.23

;定义标识符的模块使用

PUBLIC 标识符 [,标识符 ...]

;调用标识符的模块使用

EXTERN 标识符:类型 [,标识符:类型 ...]

(2) 子程序在代码段中，但没有开始和结束点。

(3) 子程序与主程序文件的存储模式要一致。

(4) 处理好子程序与主程序之间的参数传递。



例3.23 编制程序，对由键盘输入的多个有符号十进制数求平均值，并将数据及平均值以十进制形式输出。

wj0323.asm, wj0323s.asm



3.6.6 子程序库

- ◇ 利用库管理工具程序LIB.EXE
- ◇ 将子程序模块统一管理，存入子程序库文件（.LIB）
- ◇ 使用子程序库中的子程序
 - ◆ 方法1：在连接过程中指明子程序库
 - ◆ 方法2：主程序使用子程序库文件包含伪指令
INCLUDELIB指明
- ◇ 子程序库中子程序的编写与子程序模块中的要求一样，只是为方便调用，更加严格，最好遵循一致的规则。

3.7 宏汇编

- ◇ 宏是具有宏名的一段汇编语句序列。
- ◇ 宏需要先定义，然后在程序中进行宏调用。
- ◇ 由于形式上类似其他指令，所以常称其为宏指令。
- ◇ 宏指令实际上是一段代码序列的缩写，在汇编时，汇编程序用对应的代码序列替代宏指令。
- ◇ 因为是在汇编过程中实现的宏展开，所以常称为宏汇编。

wj0324.asm, wj0324.mac

1. 宏定义

- ◇ 宏定义由一对宏汇编伪指令**MACRO**和**ENDM**来完成，格式如下：

宏名 **MACRO** [形参表]
 ; 宏定义体
 ENDM

- ◇ 其中宏名是符合语法的标识符，同一源程序中该名字定义唯一。宏定义体中不仅可以是硬指令序列，还可以是伪指令语句序列。
- ◇ 可选的形参表给出了宏定义中用到的形式参数，每个形式参数之间用逗号分隔。

2. 宏调用

- ◇ 宏定义之后就可以使用它，即[宏调用](#)：

宏名 [实参表]

- ◇ 宏调用的格式同一般指令一样：在使用宏指令的位置写下宏名，后跟实体参数；如果有多个参数，应按形参顺序填入实参，也用逗号分隔。
- ◇ 在汇编时，宏指令被汇编程序用对应的代码序列替代，这就是[宏展开](#)。
- ◇ 宏展开的具体过程是：当汇编程序扫描源程序遇到已有定义的宏调用时，即用相应的宏定义体完全替代源程序的宏指令，同时用位置匹配的实参对形参进行取代。

宏的实例1

dispchar macro char ; 宏定义
 mov ah,2 ; 宏定义体
 mov dl,char
 int 21h
 endm

...


dispchar '?' ;宏调用（宏指令）

...

1 mov ah,2 ;宏展开
1 mov dl,'?'
1 int 21h



宏的实例2



```
dispmsg    macro message    ;宏定义
            mov ah,9         ;宏定义体
            lea dx,message
            int 21h
            endm
```

...

```
dispmsg string    ;宏调用（宏指令）
```

...

```
1    mov ah,9    ;宏展开
1    lea dx,string
1    int 21h
```




3. 局部标号

- ◇ 宏定义体中的标号必须用**LOCAL**伪指令声明为局部标号。
- ◇ 局部标号伪指令**LOCAL**只能用在宏定义体内，而且是宏定义**MACRO**语句之后的第一条语句：


LOCAL 标号列表

- ◇ 标号列表由宏定义体内使用的标号组成，用逗号分隔。
- ◇ 每次宏展开时汇编程序将对其中的标号自动产生一个唯一的标识符（其形式为“??0000”到“??FFFF”），避免宏展开后的标号重复。

将十六进制字符转换为十六进制数的宏



```
ASCTOH    macro
            local asctoh1,asctoh2
            cmp al,'9'
            jbe asctoh1        ;'0'~'9', 减去30H
            cmp al,'a'
            jb  asctoh2        ;'A'~'F', 还要减7
            sub al,20h         ;'a'~'f', 再减去20H
asctoh2:   sub al,7
asctoh1:   sub al,30h
            endm
```



将一个字节量数据按十六进制数显示出来的宏

disphex

macro hexdata

local disphex1

push ax ;保护寄存器

push bx

push cx

push dx

mov bx,hexdata


mov cx,0404h

;CH=4, 作为循环次数


;CL=4, 作为循环移位次数

将一个字节数据按十六进制数显示出来的宏（续）

1105 AM 26



```
disphex1:    rol bx,cl           ;高4位循环移位到低4位
              mov al,bl
              and al,0fh
              call htoasc        ;转换成ASCII码
              dispchar al        ;显示该位数值
              dec ch
              jnz disphex1
              pop dx             ;恢复寄存器
              pop cx
              pop bx
              pop ax
              endm
```




4. 文件包含

- ◇ 包含伪指令**INCLUDE**可以将任何文本文件内容插入源程序，与其他部分同时汇编。

INCLUDE 文件名

- ◇ 包含的文件可以是
 - ◆ **.MAC**宏库文件：常用的或有价值的宏定义
 - ◆ **.INC**包含文件：各种常量定义、声明语句等
 - ◆ **.ASM**汇编语言源文件：常用的子程序等
- ◇ 利用**INCLUDE**伪指令包含其他文件，其实质仍然是一个源程序，只不过是分在了几个文件书写；被包含的文件不能独立汇编，是依附主程序而存在的。

例3.24 输入中断向量号，显示其入口地址



```
include wj0324.mac    ;前面4个宏定义
;数据段
msg1 db 'Enter number (XX) : $'
msg2 db 'The Interrupt Program Address: $'
crlf db 0dh,0ah,'$'
;代码段
dispmsg msg1          ;提示输入一个两位十六进制数
mov ah,1              ;接受高位
int 21h
ASCTOH                ;将ASCII码转换为十六进制数
mov bl,al              ;存入BL
```



例3.24 输入中断向量号，显示其入口地址（续1）

1105 AM 26



shl bl,1

shl bl,1

shl bl,1

shl bl,1

mov ah,1

;接受低位

int 21h

ASCTOH

or bl,al


;合成一个字节作为中断向量号

xor bh,bh




例3.24 输入中断向量号，显示其入口地址（续2）

1105 AM 26



dispmsg crlf	;回车换行
dispmsg msg2	;提示输出中断向量（入口地址）
shl bx,1	;中断向量号×4为偏移地址
shl bx,1	
mov ax,0	;中断向量表的段地址是0
mov es,ax	
disphex es:[bx+2]	;显示中断向量的段地址
dispchar ':'	;显示 分隔字符“:”
disphex es:[bx]	;显示中断向量的偏移地址
.....	;后面含有HTOASC子程序



宏与子程序的比较



- ◇ 仅是源程序级的简化：宏调用在汇编时进行程序语句的展开，不需要返回；不减小目标程序，执行速度没有改变。
 - ◇ 还是目标程序级的简化：子程序调用在执行时由CALL指令转向、RET指令返回；形成的目标代码较短，执行速度减慢。
-
- ◇ 通过形参、实参结合实现参数传递，简捷直观、灵活多变。
 - ◇ 需要利用寄存器、存储单元或堆栈等传递参数。

宏与子程序的比较结论



宏与子程序具有各自的特点，程序员应该根据具体问题选择使用那种方法：

- ◆ 当程序段较短或要求较快执行时，应选用宏；
- ◆ 当程序段较长或为减小目标代码时，要选用子程序。



子程序与宏的学习结束
谢谢！



dpchar子程序的参数传递（例3.15）

；主程序

mov al,'?' ；主程序提供显示字符

call dpchar

；子程序：显示AL中的字符

```
dpchar proc
    push ax
    push bx
    mov bx,0
    mov ah,0eh
    int 10h
    pop bx
    pop ax
    ret
dpchar endp
```

入口参数：寄存器**AL**，传数值
出口参数：无



dpstri子程序的传递参数（例3.16）

；子程序dpstri：显示DS:SI指向的字符串

```
dpstri  proc
        push ax
dps1:   mov al,[si]
        inc si
        cmp al,0
        jz dps2
        call dpchar
        jmp dps1
dps2:   pop ax
        ret
dpstri  endp
```

入口参数：寄存器**DS:SI**，传地址
出口参数：无



HTOASC子程序的参数传递

；子程序HTOASC：十六进制数转换为ASCII码

HTOASC proc

push bx

mov bx,offset ASCII

and al,0fh

xlat CS:ASCII

pop bx

ret

ASCII db 30h,31h,32h,33h,34h,35h,36h,37h,38h,39h

db 41h,42h,43h,44h,45h,46h

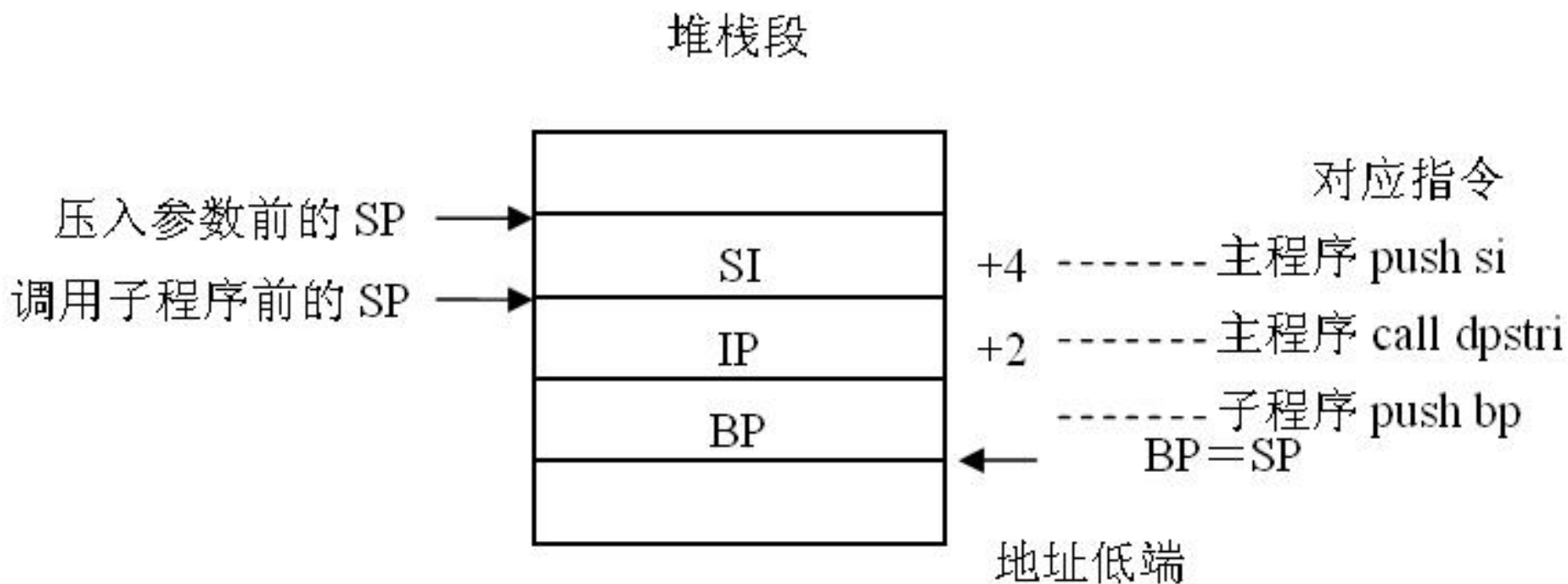
HTOASC endp

入口参数：寄存器**AL**，传数值

出口参数：寄存器**AL**，传数值



堆栈传递参数，例3.21的堆栈



例3.21 用堆栈传递参数显示字符串

; 数据段

msg db 'Well, I made it !',0

; 代码段（主程序）


mov si,offset msg

push si ;** 入口参数压入堆栈


call dpstri ;调用子程序

add sp,2 ;** 平衡堆栈

例3.21 用堆栈传递参数显示字符串（续）



```
dpstri    proc                                ;显示以0结尾的字符处
           push bp                            ;**入口参数：堆栈 = 字符串地址
           mov bp,sp                          ;**通过BP获得堆栈内的参数
           push ax
           push dx
           mov si,[bp+4]                      ;**通过BP指针获得参数
           ...
dps2:     pop dx
           pop ax
           pop bp                            ;**恢复BP寄存器
           ret
dpstri    endp
```



例3.22 计算有符号数平均值

```

count    = 10
array    dw 1234,-1234,0 ...
wmed     dw ?

```

; 数据段

; 代码段 (主程序)

```
mov ax,count
```

```
push ax           ;压入数据个数
```

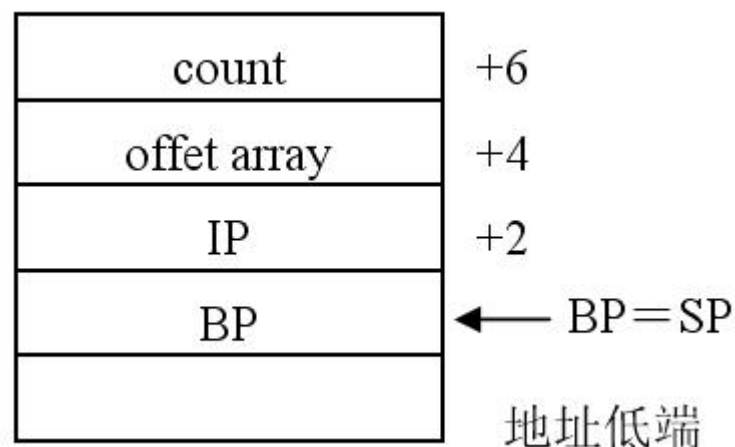
```
mov ax,offset array
```

```
push ax           ;压入数据缓冲区的偏移地址
```

```
call mean         ;调用子程序, 求平均值
```

```
add sp,4          ;平衡堆栈
```

```
mov wmed,ax       ;保存出口参数
```



例3.22 计算有符号数平均值（续1）

mean proc

;计算16位有符号数平均值子程序

;入口参数：顺序压入数据个数和数据缓冲区偏移地址

push bp

;出口参数：AX = 平均值

mov bp,sp

push bx

push cx

push dx

push si

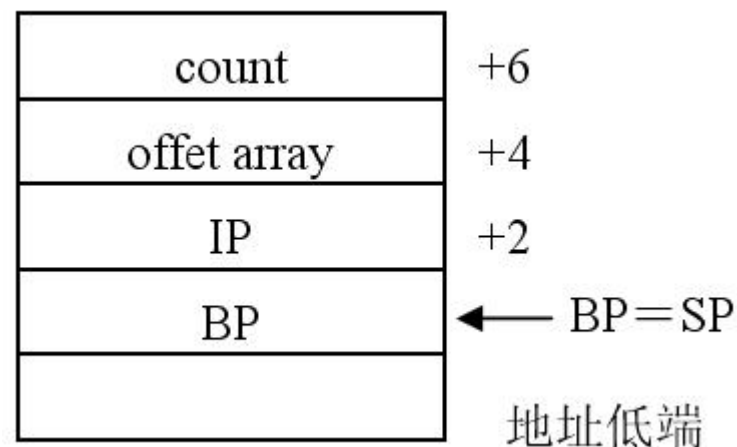
push di

mov bx,[bp+4]


;从堆栈中取出缓冲区偏移地址→BX

mov cx,[bp+6]


;从堆栈中数据个数→CX



例3.22 计算有符号数平均值（续2）



xor si,si	;SI保存求和的低16位值
mov di,si	;DI保存求和的高16位值
mean1: mov ax,[bx]	;取出一个数据→AX
cwd	;符号扩展→DX
add si,ax	;求和低16位
adc di,dx	;求和高16位
inc bx	;指向下一个数据
inc bx	
loop mean1	;循环

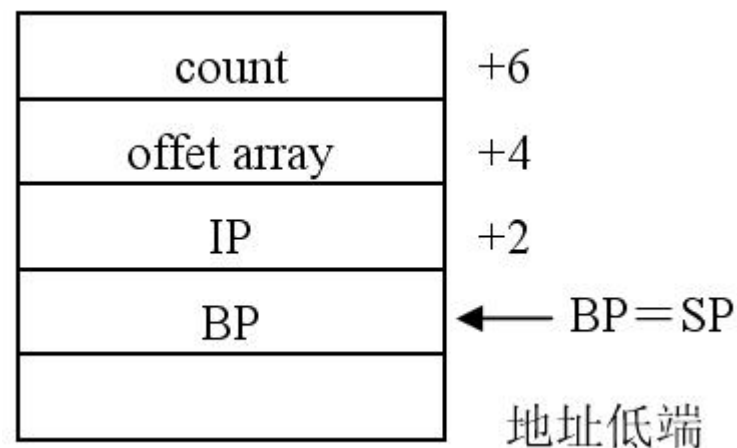


例3.22 计算有符号数平均值（续3）

```

mov ax,si          ;累加和在DX.AX
mov dx,di
mov cx,[bp+6]      ;数据个数在CX
idiv cx            ;有符号数除法, AX = 平均值
pop di            ;恢复寄存器
pop si
pop dx
pop cx
pop bx
pop bp
ret
mean endp

```



例3.23 输入有符号十进制数、求平均值输出

;子程序文件

.model small ;相同的存储模式

public read,write,mean ;子程序共用

extern wtemp:word ;声明外部变量

.code ;代码段

..... ;子程序代码

;主程序文件

.model small ;相同的存储模式

extern read:near,write:near,mean:near

;声明外部子程序

public wtemp ;变量共用

..... ;输入、计算和输出

子程序嵌套调用

1105 AM 26

