寄存器传递参数

子程序设计

- 子程序的编写方法与主程序一样
- 但需要留意几个问题：
  - ✓利用过程定义，获得子程序名和调用属性
  - ✓压入和弹出操作要成对使用，保持堆栈平衡
  - ✓开始保护寄存器，返回前相应恢复
  - ✓……

难点是参数传递

参数传递

- 主程序与子程序间通过参数传递建立联系
  - 入口参数（输入参数）：主程序→子程序
  - 出口参数（输出参数）：子程序→主程序
- 参数的具体内容
  - 数据本身（传递数值）
  - 数据的存储地址
    （传递地址，传递引用）

参数传递方法

- ✓通用寄存器
- ✓共享变量
- ✓堆栈

寄存器传递参数

- 最简单和常用的参数传递方法
- 把参数存于约定的寄存器
  - 少量数据直接传递数值
  - 大量数据只能传递地址
- 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

十六进制显示

- 要求：将一个双字数据，以十六进制形式显示
  - 主程序：提供要显示的数据
  - 子程序：将1位十六进制数转换为ASCII码
- 寄存器传递参数
  - 入口参数：AL＝1位十六进制数（对应二进制4位）
  - 出口参数：AL＝对应数码的ASCII码

少量数据直接传递数值

十进制、二进制和十六进制对应关系表

十进制	二进制	十六进制	十进制	二进制	十六进制
0	0000	0	8	1000	8
1	0001	1	9	1001	9
2	0010	2	10	1010	A
3	0011	3	11	1011	B
4	0100	4	12	1100	C
5	0101	5	13	1101	D
6	0110	6	14	1110	E
7	0111	7	15	1111	F

十六进制显示主程序—1

```
        ;代码段，主程序
        mov eax,1234abcdh ;假设一个数据
        xor ebx,ebx       ;相对寻址访问字符串
        mov ecx,8         ;8位十六进制数
again:  rol eax,4         ;高4位循环移位进入低4位
        push eax          ;也可以用mov  edx, eax
        call htoasc       ;调用子程序htoasc
```

AL低4位传递入口参数

AL传递出口参数
主程序进行保护

十六进制显示主程序—2

```
mov regd+4[ebx],al      ;保存转换后的ASCII码
pop eax                 ;也可以用mov  eax, edx
inc ebx
loop again
mov eax,offset regd
call dispmsg            ;显示
```

AL传递出口参数
主程序进行恢复

```
        ;数据段
regd  byte 'EAX=',8 dup(0),'H',0
```

+0　　+4

regd+4[ebx]

EAX=1234ABCDH

运行结果

十六进制显示原理

- 子程序（HTOASC）实现原理
  - 数值0～9：加30H
    成为字符'0'～'9'的ASCII码
  - 数值10～15：加30H，再加7
    成为字符'A'～'F'的ASCII码

ASCII表（部分）

	3	4
0	0	@
1	1	A
2	2	B
3	3	C
4	4	D
5	5	E
6	6	F
7	7	G
8	8	H
9	9	I

十六进制显示子程序—1

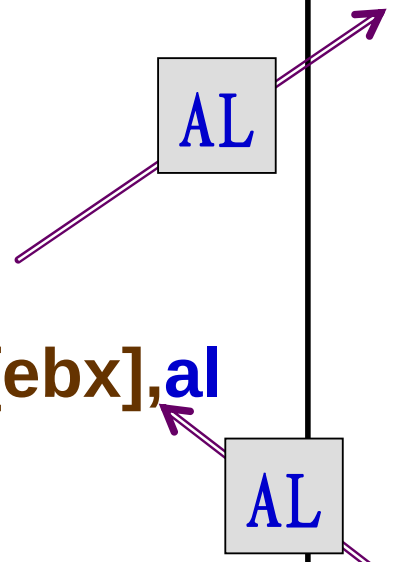
```
        ;代码段，子程序（解1）
htoasc  proc
        and al,0fh      ;只取AL的低4位
        or al,30h       ;AL高4位变成3
        cmp al,39h      ;是0～9，还是A～F
        jbe htoend
        add al,7        ;是A～F，ASCII码再加上7
htoend: ret             ;子程序返回
htoasc  endp
```

十六进制显示子程序—2

```
        ;代码段，子程序（解2）
htoasc  proc
        and eax,0fh             ;取AL低4位
        mov al,ASCII[eax]       ;换码
        ret
        ;子程序的局部数据（只读）
ASCII   byte '0123456789ABCDEF'
htoasc  endp
```

寄存器传递参数

```
        ;代码段，主程序
        ...
again:  rol eax,4
        push eax
        call htoasc
        mov regd+4[ebx],al
        pop eax
        inc ebx
        loop again
```




```
        ;代码段，子程序
htoasc  proc
        and al,0fh
        or al,30h
        cmp al,39h
        jbe htoend
        add al,7
htoend: ret
htoasc  endp
```

共享变量传递参数

参数传递

- 主程序与子程序间通过参数传递建立联系
  - 入口参数（输入参数）：主程序→子程序
  - 出口参数（输出参数）：子程序→主程序
- 参数的具体内容
  - 数据本身（传递数值）
  - 数据的存储地址
    （传递地址，传递引用）

参数传递方法

- ✓通用寄存器
- ✓共享变量
- ✓堆栈

共享变量传递参数

- 子程序和主程序使用同一个变量名存取数据
- 如果变量定义和使用不在同一个程序模块中，需要利用**PUBLIC**、**EXTREN**声明
- 共享变量传递参数，子程序的通用性较差
- 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

共享变量对应高级语言的全局变量

二进制输入

- 要求：从键盘以二进制形式输入若干32位数据
  - 主程序：提供保存输入数据的存储空间，保存数据
  - 子程序：输入一个32位二进制数
- 共享变量传递参数
  - 入口参数：无
  - 出口参数：共享变量temp＝输入的32位数据

少量数据直接传递数值

二进制输入主程序—1

```
        ;数据段
count   = 5
array   dword count dup(0)
temp    dword ?                 ;共享变量
        ; 代码段，主程序
        mov ecx,count           ;输入count个数据
        mov ebx,offset array    ;EBX间接寻址访问数组
```

二进制输入主程序—2

```
again:  call rdbd          ;调用子程序，输入一个数据
        mov eax,temp       ;获得出口参数（共享变量）
        mov [ebx],eax      ;存放到数据缓冲区
        add ebx,4
        loop again
```

使用共享变量直接传递返回值

二进制输入原理

- 子程序（RDDB）实现原理
  - 输入一个字符，判断是“0”或“1”，减30H，转换成为数值0或1
  - 重复输入字符并转换，每次输入后将前一次数值左移1位，并与新输入数值合并
  - 输入非0或1的字符、或超出位数，提示错误，重新输入

二进制输入子程序—1

```
        ;代码段，子程序：输入32位二进制数
rdbd    proc                    ;出口参数：共享变量TEMP
        push eax                ;寄存器保护
        push ebx
        push ecx
rdbd1:  xor ebx,ebx             ;EBX用于存放二进制结果
        mov ecx,32              ;限制输入字符的个数
```

二进制输入子程序—2

```
rdbd2:  call readc          ;输入一个字符
        cmp al,'0'          ;检测键入字符是否合法
        jb rderr            ;不合法则转到出错处理
        cmp al,'1'
        ja rderr
        sub al,'0'          ;对输入的字符进行转化
        shl ebx,1           ;EBX的值乘以2（左移1位）
        or bl,al            ;BL和AL相加（或）
        loop rdbd2          ;循环输入字符
```

二进制输入子程序—3

```
mov temp,ebx        ;把二进制结果存放TEMP返回
call dispcrlf       ;分行
pop ecx
pop ebx
pop eax
ret
```

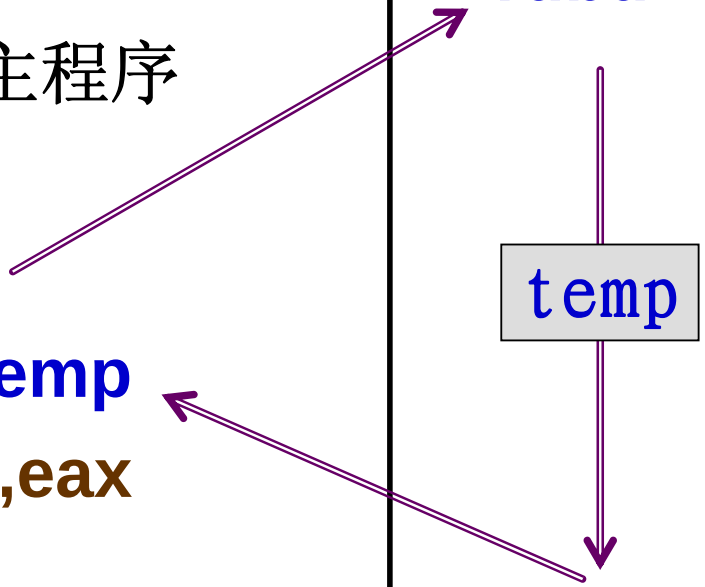
使用共享变量直接传递返回值

二进制输入子程序—4

```
rderr:    mov eax,offset errmsg   ;显示错误信息
          call dispmsg
          jmp rdbd1               ;重新输入
errmsg    byte 0dh,0ah,'Input error, enter again: ',0
rdbd      endp
```

共享变量传递参数

```
        ;数据段，共用
temp    dword ?
        ;代码段，主程序
        ...
again:  call rdbd
        mov eax,temp
        mov [ebx],eax
        add ebx,4
        loop again
```




```
        ;代码段，子程序
rdbd    proc
        push eax
        ...
        mov temp,ebx
        ...
        pop eax
        ret
        ...
rdbd    endp
```

堆栈传递参数

参数传递

- 主程序与子程序间通过参数传递建立联系
  - 入口参数（输入参数）：主程序→子程序
  - 出口参数（输出参数）：子程序→主程序
- 参数的具体内容
  - 数据本身（传递数值）
  - 数据的存储地址
    （传递地址，传递引用）

参数传递方法

- ✓通用寄存器
- ✓共享变量
- ✓堆栈

堆栈传递参数

- 主程序将入口参数压入堆栈
  - 子程序从堆栈中取出参数
- 采用堆栈传递参数常是程式化的
  - 子程序设置**EBP**等于当前**ESP**
  - 利用**EBP**相对寻址访问堆栈中的参数
- 出口参数通常不使用堆栈传递

高级语言函数调用的参数实质是堆栈传递参数

计算有符号数平均值

- 要求：将数组元素求和，除以元素个数，求得平均值
  - 主程序：提供数组地址和元素个数，显示结果
  - 子程序：求平均值，返回结果
- 堆栈传递入口参数
  - 压入堆栈元素个数（传数值）
  - 压入堆栈数组地址（传地址）
- 寄存器传递出口参数
  - EAX＝平均值（传数值）

堆栈传递数值和地址

计算有符号数平均值主程序

```
        ;数据段
array   dword 675,354,-34,198,267,0,9,2371,-67,4257
        ;代码段，主程序
        push lengthof array ;压入数据个数(4个字节)
        push offset array   ;压入数组地址(4个字节)
        call mean           ;调用求平均值子程序
        add esp,8           ;主程序平衡堆栈(弹出8个字节)
        call dispsid        ;显示平均值EAX(出口参数)
```

计算有符号数平均值子程序—1

```
        ;代码段，子程序：计算32位有符号数平均值
mean    proc                ;入口参数：顺序压入个数和地址
        push ebp            ;出口参数：EAX＝平均值
        mov ebp,esp
        push ebx            ;保护寄存器
        push ecx
        push edx
```

采用堆栈传递参数可以程式化

- 子程序设置EBP等于当前ESP
- 利用EBP相对寻址访问堆栈中的参数

计算有符号数平均值子程序—2

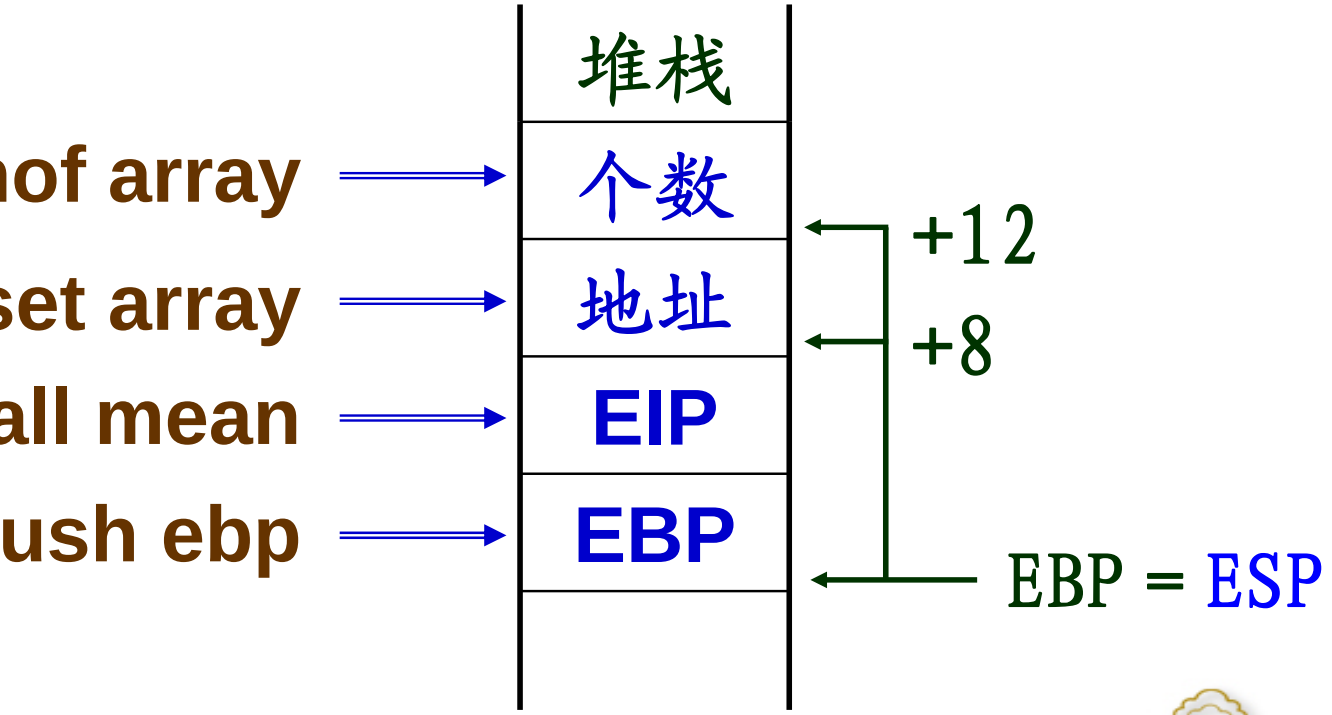
```
mov ebx,[ebp+8]      ;EBX＝取出的数组地址
mov ecx,[ebp+12]     ;ECX＝取出的数据个数
```

主程序： push lengthof array

主程序： push offset array

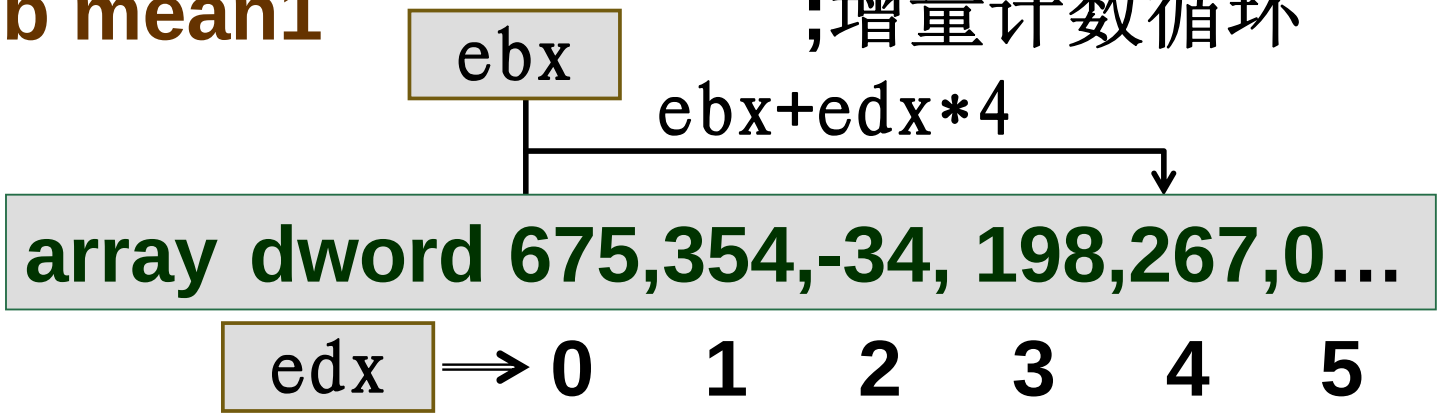
主程序： call mean

子程序： push ebp



计算有符号数平均值子程序—3

```
        xor eax,eax                ;EAX保存和值
        xor edx,edx                ;EDX＝指向数组元素
mean1:  add eax,[ebx+edx*4]        ;求和
        add edx,1                  ;指向下一个数据
        cmp edx,ecx                ;比较个数
        jb mean1                   ;增量计数循环
```



计算有符号数平均值子程序—4

```
cdq          ;将累加和EAX符号扩展到EDX
             ;将EAX最高位填入EDX所有位
idiv ecx     ;有符号数除法，EAX＝平均值
```

```
; cdq替代指令
mov edx,eax
sar edx,31
```

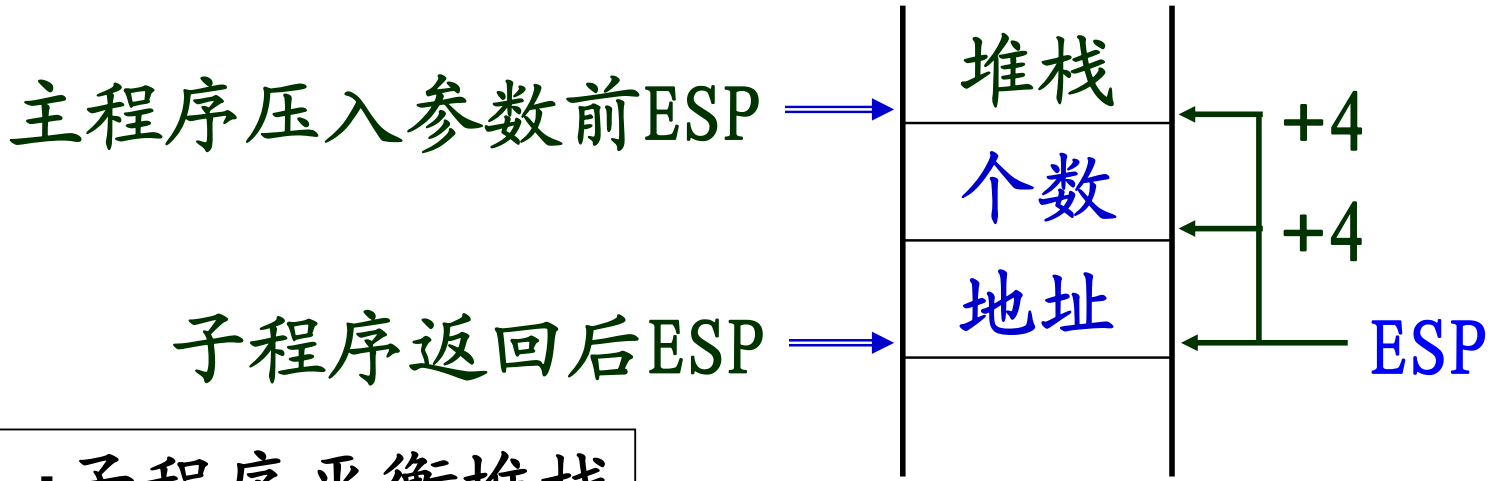
```
IDIV r32/m32    ;32位除法指令
;EAX = EDX.EAX ÷ r32/m32的商
;EDX = EDX.EAX ÷ r32/m32的商
```

计算有符号数平均值子程序—5

```
        pop edx          ;恢复寄存器
        pop ecx
        pop ebx
        pop ebp
        ret
mean    endp
```

堆栈平衡了吗？

```
add esp,8   ;主程序平衡堆栈
```




```
ret  8          ;子程序平衡堆栈
```

堆栈传递参数

```
;代码段，主程序
...
push lengthof array
push offset array
call mean
add esp,8
call dispsid
```

主程序平衡堆栈

EAX


```
;代码段，子程序
mean    proc
        push ebp
        mov ebp,esp
        ...
        mov ebx,[ebp+8]
        mov ecx,[ebp+12]
        ...
        ret
mean    endp
```