

## 1. **ss:sp** 及堆栈段

**ss: stack segment**

**sp: stack pointer** 堆栈指针, 表示堆栈顶端的偏移地址, 而 **ss** 则用来表示堆栈的段地址。

但是不能用 **[sp]** 或 **[sp+常数或其它寄存器]** 的形式来引用某个变量。为了弥补 **sp** 不能表示间接地址的缺陷, 同时也为了凑齐 4 个偏移地址寄存器, 于是从通用寄存器中找来 **bx** 作为 **sp** 的替补, 于是就有了以下 4 个偏移地址寄存器:

**bx, bp, si, di**

即这 4 个寄存器都可以放在 **[]** 中表示变量的偏移地址。

**[ax], [cx], [dx]** 均是错误的。

**push abc[0]** 的过程:

①  $sp = sp - 2 = 200h - 2 = 1FEh$

② 把 **push** 后面所跟的值保存到 **ss:sp** 当前指向的内存单元中

**ss:1FE 34h** ←

**ss:1FF 12h**

**pop abc[2]** 的过程:

① 把当前 **ss:sp** 指向的字取出来, 保存到 **pop** 后面所跟的变量中

②  $sp = sp + 2$

堆栈段的定义及使用。例如: <http://10.71.45.100/bhh/ss.asm>

代码中 **push word ptr ds:[0]** 时,

$sp = sp - 2 = 200h - 2 = 1FE$

再把 **ds:[0]** 里面的值 **1234h** 保存到 **ss:1FE** 里面

而 **pop word ptr ds:[2]** 时, 先取出当前 **ss:sp** 指向的 16 位值即 **1234h** 并保存到 **ds:[2]** 中, 再做  $sp = sp + 2$

## 2. **es**

**es: extra segment** 附加段, 它跟 **ds** 类似, 可以用来表示一个数据段的段址。例如: <http://10.71.45.100/bhh/2seg.asm>

### 3. FL 标志寄存器

FL 共 16 位, 但只用其中 9 位, 这 9 位包括 6 个状态标志和 3 个控制标志, 如下所示:

11	10	9	8	7	6	4	2	0							
O	D	I	T	S	Z	A	P	C							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
×	×	×	×	OF	DF	IF	TF	SF	ZF	×	AF	×	PF	×	CF
0	0	0	0							0		0		1	

CF: 进位标志 (carry flag)

```
mov ah, 0FFh
```

```
add ah, 1; AH=0, CF=1 产生了进位
```

```
add ah, 2; AH=2, CF=0
```

```
sub ah, 3; AH=0FFh, CF=1 产生了借位
```

与 CF 相关的两条跳转指令: jc, jnc

ZF: 零标志 (zero flag)

```
sub ax, ax; AX=0, ZF=1
```

```
add ax, 1; AX=1, ZF=0
```

```
add ax, 0FFFFh; AX=0, ZF=1, CF=1
```

```
jz is_zero; 会发生跳转, 因为当前 ZF==1
```

与 jz 相反的指令是 jnz, jnz 是根据 ZF==0 作出跳转

注意: mov 指令不影响任何标志位, 例如:

```
mov ax, 1234h
mov bx, 1234h
sub ax, bx
mov bx, 1; 此 mov 不影响 sub 指令产生的 ZF 状态
jz iszero
```

```
mov ax, 1234h
mov bx, 1234h
sub ax, bx
jz iszero
```

```
mov bx, 0
jmp done; 与左边相比
```

```

                                ; 这里多出一条
                                ; jmp 指令
mov bx, 0                      iszero: ; 故左边写法更好
iszero:                        mov bx, 1
                                done:

```

与 ZF 相关的跳转指令除了 jz, jnz 还有 je, jne

其实 je≡jz, jne≡jnz

设 ax=1234h, bx=1234h

cmp ax, bx; ZF=1, 因为 cmp 指令内部做了减法会  
; 影响 ZF 的状态

je is\_equal; 写成 jz is\_equal 效果一样

SF: 符号标志(sign flag)

```
mov ah, 7Fh
```

```
add ah, 1; AH=80h=1000 0000B, SF=1
```

```
sub ah, 1; AH=7Fh=0111 1111B, SF=0
```

```
jns positive; 会发生跳转, 因为 SF==0
```

与 jns 相反的指令为 js, js 是根据 SF==1 作出跳转

OF: 溢出标志(overflow flag)

```
mov ah, 7Fh
```

```
add ah, 1; AH=80h, OF=1, ZF=0, CF=0, SF=1
```

```
mov ah, 80h
```

```
add ah, 0FFh; AH=7Fh, OF=1, ZF=0, CF=1, SF=0
```

```
mov ah, 80h
```

```
sub ah, 1; AH=7Fh, OF=1, ZF=0, CF=0, SF=0
```

OF 也有两条相关的指令: jo, jno