第八章 数据处理

数据处理的两个基本问题

- 处理的数据在什么地方
- 处理的数据有多长

bx， si， di和bp寄存器

`bx, si, di`放在`[...]`中来进行内存单元的寻址。`bp`寄存器也有相同的作用，不过有点用法的上需要注意的地方。

单独使用都是可以的，但是组合的情况下需要注意，

- `si, di`不可以同时出现。
- `bx, bp`不可以同时出现。

如果在`[...]`中使用到了`bp`寄存器，那么段地址默认是在`ss`寄存器中。

数据在哪儿

```
1 mov ax, 1
2 mov ax, [1]
3 mov ax, bx
```

- 立即数(idata)

  对于直接包含在指令中的数据，执行前在cpu的指令缓冲器中。在汇编语言中被成为立即数。如上面的`mov ax, 1`

- 寄存器

  要处理的数据在寄存器中。如上面的`mov ax, bx`

- 段地址和偏移地址

  要处理的数据在内存中，在汇编语言中可以使用[X]的格式给出偏移地址(EA)，其段地址(SA)在某个段地址寄存器中。

  如`mov ax, [1]`，`mov ax, [di]`，`mov ax, [bx + si]`这些指令的段地址都在`ds`寄存器中，而`mov ax, [bp]`，`mov ax, [bp + di]`这些指令的段地址都在`ss`寄存器中。

  存放段地址的寄存器也**可以显式的给出**，如：

  `mov ax, ss:[bx]`，`mov ax, ds:[bp]`，`mov ax, es:[di]`

数据有多长

- 通过寄存器指明数据的尺寸
  - `mov ax, 1`这里的1就是一个字的大小。
  - `mov al, 1`这里的1就是一个字节大小。
- 没有寄存器，使用X ptr指明内存单元的长度

  常见的是`byte ptr`表示处理的是字节数据，`word ptr`表示处理的是字数据。

  - `mov word ptr ds:[0], 1`
  - `mov byte ptr ds:[2], 2`
  - `inc word ptr [bx]`
- 通过指令

  `push/pop`指令只进行字操作。

  `push [1000H]`不要写成`push word ptr [100H]`。

div指令

div指令的是除法指令，格式如下。

`div reg 或者 div 内存单元`

需要注意一下的问题。

- 除数 => 有8位和16位两种，在一个reg或者内存单元中。
- 被除数 => 默认放在`ax`或`dx和ax`中。如果除数为8位那么被除数为16位，如果除数为16位，被除数为32位（在dx和ax中存放，dx中存放高16位，ax存放低16位）
- 结果 => 如果除数为8位，al为商，ah为余。如果除数为32位，ax为商， dx为余。

例子：

```
div byte ptr ds:[0]
al中存放商，ah中存放余
div word ptr es:[2]
ax中存放商，dx中存放余
```

例：计算 100001/100

将100001化为16进制为186a1h，需要使用ax和dx两个寄存器存放。此时被除数为32位，除数需为16位。

```
mov ax, 86a1h ; 存放低八位
mov dx, 1h      ; 存放高八位

mov bx, 100   ; 使用一个16位的寄存器存储除数
div bx
```

程序执行结果： ax中为`03e8h`（1000）， dx 中为1

例：计算1001/100

1001<65535(ffffh)，可以使用ax寄存器存放。此时被除数为16位，除数为8位。

```
mov ax, 1001
mov bl, 100
div bl
```

程序执行结果：al中为0ah(10)，ah中为1。

伪指令dd

- `db: define byte`
- `dw:  define word`
- `db: define dword`

```
data segment
        db 1 ; 一个字节
        dw 1 ; 两个字节
        db 1 ; 四个字节
data ends
```

伪指令dup

`dup`是`duplicate`的缩写，译为复制，重复。

之前我们定义栈段的时候使用如下的方式。

```
stack segment
        dw 0, 0, 0, 0, 0, 0, 0, 0 ;定义了16个字节的栈段
stack ends
```

如果我们需要定义160个字节的栈段又该如何呢？？

`dup`指令通常和`db, dw, dd`这些指令一起使用。如：

```
db 3 dup (0)
相当于
db 0, 0, 0

db 3 dup (1, 2, 3)
相当于
db 1, 2, 3, 1, 2, 3, 1, 2, 3

db 3 dup ('abc, 'ABC')
相当于
db 'abcABCabcABCabcABC'
```

上面我们需要定义160字节的栈段，就可以使用

```
stack segment
        db 160 dup (0)
stack ends
```