

1. 除法溢出

(1) 除法溢出的两种情形：

①

```
mov ax, 1234h
```

```
mov bh, 0
```

```
div bh; 此时因为除以 0, 所以会发生除法溢出
```

②

```
mov ax, 123h
```

```
mov bh, 1
```

```
div bh; 此时由于商无法保存到 AL 中, 因此也会发生溢出。
```

(2) 除法溢出时会发生什么？

```
mov ax, 123h
```

```
mov bh, 1
```

```
;除法溢出时会在此处插入 int 00h 并执行
```

```
;在 dos 系统下, int 00h 会显示溢出信息并终止程序运行
```

```
div bh; 此处发生除法溢出
```

```
mov ah, 4Ch; \ 这 2 条指令将不可能被执行到
```

```
int 21h      ; /
```

2. 逻辑运算和移位指令

(1) 逻辑运算指令：AND, OR, XOR, NOT, TEST

```
mov ax, 9234h
```

```
test ax, 8000h; ZF=0, AX=9234h
```

```
jnz msb_is_one; most significant bit 最高位
```

test 和 and 的关系相当于 cmp 和 sub 的关系。

判断某个寄存器是否为 0 的几种方法：

```
test cl, cl
or cl, cl
and cl, cl
or cl, 0
cmp cl, 0
```

上述每条指令后面都可以跟 `jz` 或 `jnz` 来判断 CL 是否为 0。

(2) 移位指令

```
shl, shr, sal, sar, rol, ror, rcl, rcr
```

`rcl`: rotate through carry left 带进位循环左移

`rcr`: rotate through carry right 带进位循环右移

```
mov ah, 0B6h
```

```
stc; CF=1
```

```
rcl ah, 1; CF=1 AH=1011 0110 移位前
```

```
          ; CF=1 AH=0110 1101 移位后
```

```
mov ah, 0B6h
```

```
stc; CF=1
```

```
rcr ah, 1; AH=1011 0110 CF=1 移位前
```

```
          ; AH=1101 1011 CF=0 移位后
```

例如：要把 1234ABCDh 逻辑左移 3 位，结果保存在 `dx:ax`

解法 1：

设 `ax=0ABCDh`

```
and ax, 0E000h
```

```
mov dx, 1234h
```

```
mov ax, 0ABCDh
```

```
mov cl, 3
shl dx, cl
mov bx, ax
shl ax, cl
mov cl, 13
shr bx, cl
or dx, bx
```

解法 2:

```
mov dx, 1234h
mov ax, 0ABCDh
mov cx, 3
next:
shl ax, 1
rcl dx, 1
dec cx
jnz next
```

sal: shift arithmetic left 算术左移

sar: shift arithmetic right 算术右移

sal 及 **sar** 是针对符号数的移位运算, 对负数右移的时候要在左边补 1, 对正数右移的时候左边补 0, 无论对正数还是负数左移右边都补 0。显然 **sal**≡**shl**。

shl 及 **shr** 是针对非符号数的移位运算, 无论左移还是右移, 空缺的部分永远补 0。

shl, **shr**, **rol**, **ror**, **rcl**, **rcr** 最后移出去的那一位一定在 **CF** 中。

假定要把 **AX** 中的 16 位值转化成二进制输出:

解法 1:

```
mov cx, 16
next:
shl ax, 1
jc is_1
is_0:
mov dl, '0'
jmp output
is_1:
mov dl, '1'
output:
push ax
mov ah, 2
int 21h
pop ax
dec cx
jnz next
```

解法 2:

```
mov cx, 16
next:
shl ax, 1
mov dl, '0'
adc dl, 0
output:
push ax
mov ah, 2
int 21h
pop ax
dec cx
```

```
jnz next
```

C 语言有 2 个库函数用来做循环左移及右移:

```
unsigned int _rotr(unsigned int x, int n)
```

```
unsigned int _rotr(unsigned int x, int n)
```

```
unsigned int _rotr(unsigned int x, int n)
```

```
{
```

```
    return x << n | x >> sizeof(x)*8-n;
```

```
}
```

3. 字符串操作指令

(1) 字符串传送指令: MOVSB, MOVSW, MOVSD

`rep movsb`

其中 `rep` 表示 `repeat`, `s` 表示 `string`, `b` 表示 `byte`
在执行此指令前要做以下准备工作:

- ① `ds:si` → 源字符串 (`si` 就是 `source index`)
- ② `es:di` → 目标字符串 (`di` 就是 `destination index`)
- ③ `cx` = 移动次数
- ④ `DF=0` 即方向标志设成正方向 (用指令 `cld`)

`rep movsb` 所做的操作如下:

```
again:
if(cx == 0)
    goto done;
byte ptr es:[di] = byte ptr ds:[si]
if(df==0)
    {si++; di++;}
else
    {si--; di--;}
cx--
goto again
done:
```

单独 `movsb` 指令所做的操作如下:

```
byte ptr es:[di] = byte ptr ds:[si]
if(df==0)
    {si++; di++;}
else
    {si--; di--;}
```

例子: 要把以下左侧 4 个字节复制到右侧

1000:0000	'A'	2000:0000	'A'
1000:0001	'B'	2000:0001	'B'

1000:0002 'C'	2000:0002 'C'
1000:0003 00	2000:0003 00

则程序可以这样写:

mov ax, 1000h	
mov ds, ax	
mov si, 0	; mov si, 3
mov ax, 2000h	
mov es, ax	
mov di, 0	; mov di, 3
mov cx, 4	
cld	; std
rep movsb	
循环结束时	循环结束时
si=4	si=FFFF
di=4	di=FFFF
cx=0	cx=0

rep movsw 的操作过程:

```
again:
if(cx == 0)
    goto done;
word ptr es:[di] = word ptr ds:[si]
if(df==0)
    {si+=2; di+=2;}
else
    {si-=2; di-=2;}
cx--
goto again
done:
```

rep movsd 的操作过程:

```
again:
if(cx == 0)
    goto done;
dword ptr es:[di] = dword ptr ds:[si]
if(df==0)
    {si+=4; di+=4;}
else
```

```

    {si-=4; di-=4;}
cx--
goto again
done:

```

在 32 位系统下，假定 `ds:esi`→源内存块，`es:edi`→目标块，`DF=0`，则当要复制的字节数 `ecx` 不是 4 的倍数时，可以做如下处理：

```

push ecx
shr ecx, 2
rep movsd
pop ecx
and ecx, 3; 相当于 ecx = ecx % 4
rep movsb

```

(2) 字符串比较指令：CMPSB, CMPSW, CMPSD

① `cmps`

比较 `byte ptr ds:[si]` 与 `byte ptr es:[di]`

当 `DF=0` 时，`SI++`，`DI++`

当 `DF=1` 时，`SI--`，`DI--`

② `repe cmpsb`; (若本次比较相等则继续比较下一个)

`again:`

`if(cx == 0) goto done;`

比较 `byte ptr ds:[si]` 与 `byte ptr es:[di]`

当 `DF=0` 时，`SI++`，`DI++`

当 `DF=1` 时，`SI--`，`DI--`

`cx--`

若本次比较相等则 goto again

done:

③repne cmpsb

(若本次比较不等则继续比较下一个)

设 ds=1000h, si=10A0h, es=2000h, di=20F0h

cx=6, DF=0

1000:10A0	'A'	2000:20F0	'A'
1000:10A1	'B'	2000:20F1	'B'
1000:10A2	'C'	2000:20F2	'C'
1000:10A3	'1'	2000:20F3	'4'
1000:10A4→	'2'	2000:20F4→	'5'
1000:10A5	'3'	2000:20F5	'6'

repe cmpsb

je equal; 全等

dec si

dec di

...

equal:

(3) 字符串扫描指令: scasb, scasw, scasd

scasb:

cmp al, es:[di]

di++; 当 DF=1 时, 为 di--

repne scasb:

next:

if(cx == 0) goto done;

cmp al, es:[di]

```

        di++; 当 DF=1 时, 为 di--
        cx--
    je done
    goto next
done:

```

例子：假定从地址 1000:2000 开始存放一个字符串，请计算该字符串的长度并存放到 CX 中。假定字符串以 ASCII 码 0 结束，字符串长度不包括 0。

```

mov ax, 1000h
mov es, ax
mov di, 2000h; ES:DI→目标串
mov cx, 0FFFFh; CX=最多找 FFFF 次
mov al, 0; AL=待找的字符
cld          ; DF=0, 表示正方向
repne scasb; again:
not cx       ; 相当于 cx=FFFF-cx
dec cx
;上述两条指令也可以替换成以下两条指令:
;inc cx
;not cx

```

repe scasb

假定从地址 1000:0000 起存放以下字符串：
 "###ABC"，现要求跳过前面的#，把后面剩余的
 全部字符复制到 2000:0000 中。

假定 es=1000h, di=0, cx=7, 则

```

mov al, '#'
cld
repe scasb
dec di; ES:DI->"ABC"
inc cx; CX=4
push es
pop ds; DS=ES
push di
pop si; SI=DI

```

```
mov ax, 2000h
mov es, ax
mov di, 0
rep movsb
```