汇编常见指令

---

add指令

1. add 寄存器，数据 例：add ax,8
2. add 寄存器，寄存器 例：add ax,bx
3. add 寄存器，内存单元 例：add ax,[0]
4. add 内存单元，寄存器 例：add [0],ax

sub指令

1. sub 寄存器，数据 例：sub ax,8
2. sub 寄存器，寄存器 例：sub ax,bx
3. sub 寄存器，内存单元 例：sub ax,[0]
4. sub 内存单元，寄存器 例：sub [0],ax

mul乘法指令

两个相乘的数，或者全为8位，或者全为16位；

若为8位，一个默认放置于AL中，另一个置于8位寄存器或内存单元中，结果默认放置于AX中；

若为16位，一个默认放置于AX中，另一个置于16位寄存器或内存单元中，结果高位默认置于DX，低位置于AX；

无符号数乘法指令： mul 通用寄存器/内存单元
有符号数乘法指令： imul 通用寄存器/内存单元

在这里插入图片描述

div除法指令

【在除号前面的是被除数，除号后面的是除数。】

除法分为8位和16位的运算，有被除数和除数

除数： 有8位和16位两种，在一个寄存器或内存单元中

被除数： 默认放置在AX或DX和AX中，除数为8位，被除数为16位，默认在AX中放置；

若除数为16位，被除数为32位，在DX和AX中放置，DX存放高16位，AX存放低16位；

结果： 如果除数为8位，则AL存储除法操作的商，AH存储除法操作的余数

如果除数为16位，则AX存储除法操作的商，DX存储除法操作的余数

无符号数除法指令： div 通用寄存器/内存单元
有符号数除法指令： idiv 通用寄存器/内存单元

inc（自增）（即C语言++）

只有一个操作数：寄存器或存储单元

对操作数加1（增量）再将结果返回原处，用于计数器和地址指针的调整

取ax，默认值为5

```
inc ax   // 则ax值变为6
```

dec（自减）（即–）

只有一个操作数：寄存器或存储单元

```
dec ax     // 则ax值变4
```

push（入栈）

```
push 寄存器：将一个寄存器中的数据入栈

push 段寄存器：将一个段寄存器中的数据入栈

push 内存单元：将一个内存单元处的字入栈
```

pop（出栈）

```
pop 寄存器：用一个寄存器接受出栈的数据。

pop 段寄存器：用一个段寄存器接受出栈的数据

pop 内存单元：出栈，用一个内存字单元接收出栈的数据
```

and（与）

逻辑与运算，按位进行与运算
通过该运算符可将操作对象的相应位设为0，其他位不变

例:

```
mov  ax,01100001B
and  ax,10111111B
```

结果ax变为00100001B

or（或）

逻辑或运算，按位进行或运算

例:

```
mov  ax,01100001B
or  ax,00001100B
```

结果ax变为01101101B

Test

作用与and指令相似，进行逻辑于运算，但不会保存结果，Test命令的两个操作数不会被改变。运算结果在设置过相关标记位后会被丢弃，仅对标志寄存器【ZF】进行修改

```
test  ax,ax
```

CMP（比较）

cmp相当于是sub指令，不保存结果，仅对相应标志寄存器进行修改。

```
 sub ax,bx
```

常见的传输类汇编指令

mov指令

```
mov 寄存器，数据
mov 寄存器，寄存器
mov 寄存器，内存单元
mov 内存单元，寄存器
mov 段寄存器，寄存器
mov 寄存器，段寄存器
```

call指令

1. call 标号【将当前的IP压栈后，转到标号处执行指令】

```
(1) (sp) = (sp) -2
            ((ss) * 16 + (sp)) = (IP)
(2) (IP) = (IP) + 16位位移
```

16位位移 = “标号”处的地址 - call指令后的第一个字节的地址
16位位移由编译程序在编译时算出。

CPU执行“ call 标号 ”时，相当于进行：

```
push IP
jmp near ptr  标号【转到标号处执行指令】
```

2. call far ptr 标号

```
(1) sp = sp -2
        ss * 16 + sp = cs
        sp = sp -2
        ss * 16 + sp = IP
(2) CS = 标号所在的段地址
        IP = 标号所在的偏移地址
```

CPU执行“ call far ptr 标号 ”时，相当于进行：

```
push CS
push IP
jmp far ptr 标号
```

3. call 16位寄存器

```
sp = sp -2
ss * 16 + sp = IP
IP = 16位寄存器
```

CPU执行“ call 16位寄存器”时，相当于进行：

```
push IP
jmp 16位寄存器
```

4. call word ptr 内存单元地址

```
push IP
jmp word ptr 内存单元地址
```

5. call dword ptr 内存单元地址

```
push CS
push IP
jmp dword ptr 内存单元地址
```

ret指令

ret指令用栈中的数据，修改IP的内容，从而实现近转移。

CPU进行ret指令时，进行下面两步操作：

```
 (1)  (IP) = ((ss) * 16 + (sp))
(2)  (sp) = (sp) + 2
```

相当于 pop IP

jmp指令

```
jmp 段地址:偏移地址
```

同时修改CS和IP，用指令中给出的段地址修改CS，偏移地址修改IP。

```
jmp 某一合法寄存器
```

则是仅修改IP。如jmp ax类似于mov IP,ax。

```
jmp 2AE3:3     即物理地址被修改为2AE33
jmp 3:0B16     0003:0B16,即物理地址被修改为00B46
```

常用控制类指令

转移指令

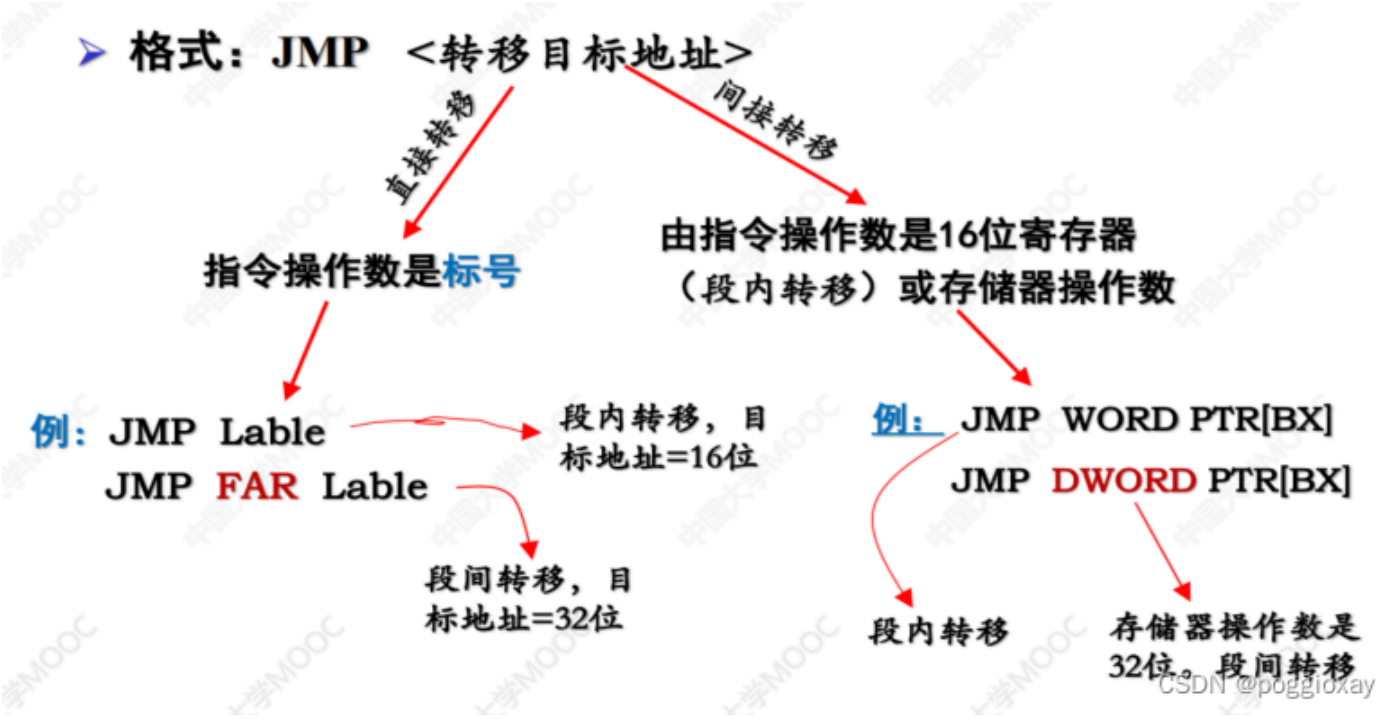
1. 无条件转移指令

直接和间距的区别就是，前者是直接给地址，后者通过寄存器或者其他
段间和段内的区别就是，前者32位（CS:IP 故32位），后者是16位（就ip，所以16位）

```
段内直接：jmp 地址，目标地址16位
段内间接：jmp 寄存器
```

段间直接转移就是我们在指令里面直接给出 32 位目标地址（CS:IP）
段间间接转移就是需要通过 32 位的存储器操作数（注意不能是寄存器了）给出目标地址。

```
段间直接：jmp 地址，目标地址32位
段间间接：jmp  DWORD PTR[BX]，[BX]指向的是存储器操作数
```



2. 有条件转移指令

根据单个条件标志的设置情况转移：这种转移指令常常用于适用于测试某一次运算的结果并根据其不同特征产生程序分支不同的处理的情况

指令	英文	含义	格式	测试条件
JZ/JE	jump if zero/equal	结果为零/相等则转移	JZ/JE OPR	ZF=1
JNZ/JNE	jump if not zero/equal	结果不为零/不相等则转移	JNZ/JNE OPR	ZF=0
JS	jump if sign	结果为负则转移	JS OPR	SF=1
JNS	jump if not sign	结果为正则转移	JNS OPR	SF=0
JO	jump if overflow	溢出则转移	JO OPR	OF=1
JNO	jump if not overflow	不溢出则转移	JNO OPR	OF=0
JP/JPE	jump if parity/parity even	奇偶位为1则转移	JP/JPE OPR	PF=1
JNP/JNPE	jump if not parity/parity even	奇偶位为0则转移	JNP/JNPE OPR	PF=0
JB/JNAE/JC	jump if below/not above、not equal/carry	低于/不高于或不等于/进位为1则转移	JB/JNAE/JC OPR	CF=1
JNB/JAE/JNC	jump if not below/ above、equal/not carry	不低于/高于或等于/进位为零则转移	JNB/JAE/JNC OPR	CF=0

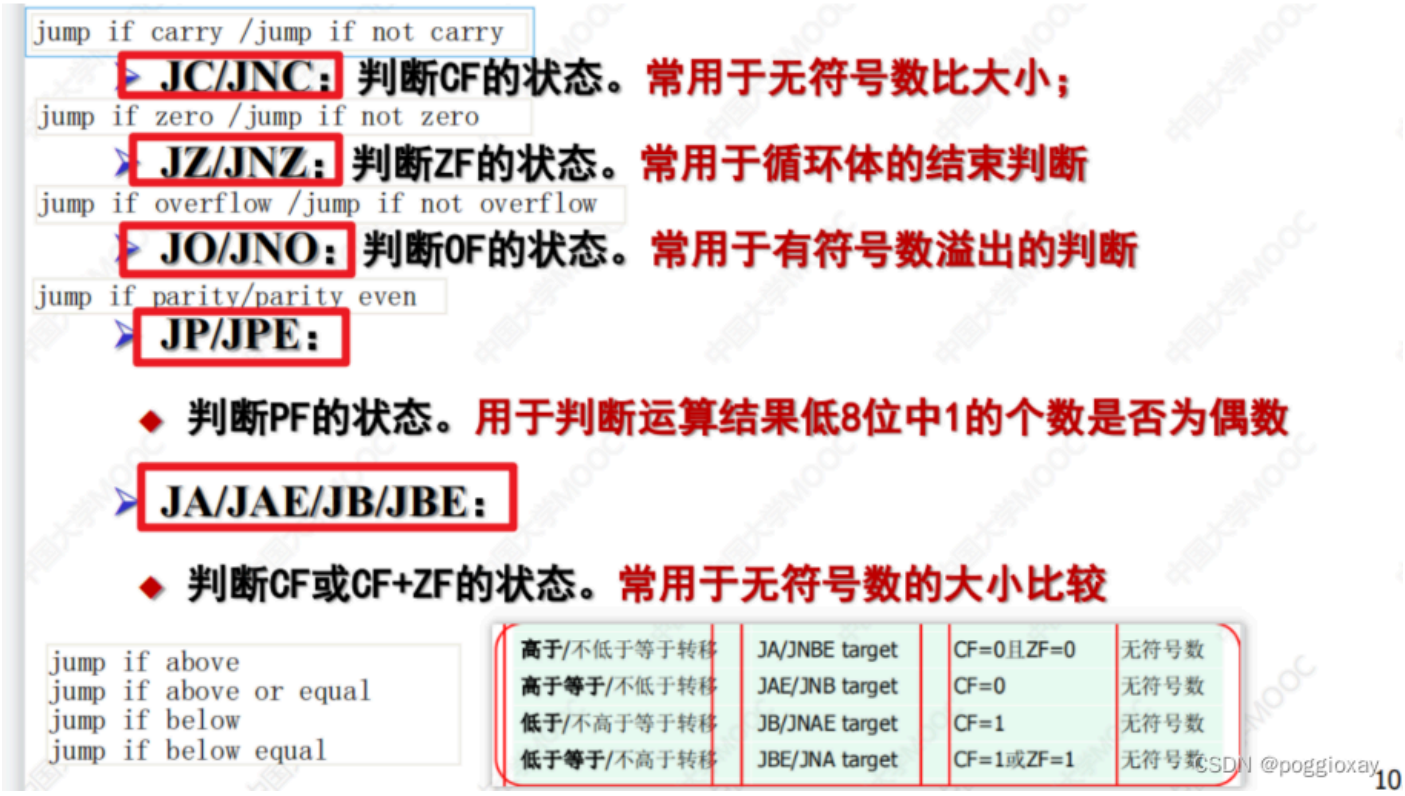
比较两个无符号数，并根据比较的结果转移

指令	英文	含义	格式	测试条件	等价于
JB/JNAE/JC	jump if below/not above、not equal/carry	低于/不高于或不等于/进位为1则转移	JB/JNAE/JC OPR	CF=1	<
JNB/JAE/JNC	jump if not below/ above、equal/not carry	不低于/高于或等于/进位为零则转移	JNB/JAE/JNC OPR	CF=0	≥
JBE/JNA	jump if below/equal、not above	低于/等于、不高于则转移	JBE/JNA OPR	CF并ZF=1	≤
JNBE/JA	jump if not below/not equal、above	不低于/不等于、高于则转移	JNBE/JA OPR	CF并ZF=0	>

比较两个带符号数，并根据比较的结果转移

指令	英文	含义	格式	测试条件	等价于
JL/JNGE	jump if less、not greater/equal	小于、不大于/不等于则转移	JL/JNGE OPR	SF异或CF=1	<
JNL/JGE	jump if not less、greater/equal	不小于、大于/等于则转移	JNL/JGE OPR	SF异或CF=0	≥
JLE/JNG	jump if less/equal、not greater	小于/等于、不大于则转移	JLE/JNG OPR	(SF异或CF)并ZF=1	≤
JNLE/JG	jump if not less/not equal、 greater	不小于/不等于、大于则转移	JNLE/JG OPR	(SF异或CF)并ZF=0	>

常用：



JZ（Jump if Zero）是此前的运算结果为0时跳转。
若此前运算结果不为0，则不跳转，执行JZ指令后面的下一条指令。

判断结果是否为零，靠的是ZF标志位状态。
若结果是0，则ZF=1
若结果不是0，则ZF=0

所以，JZ指令是在ZF=1时跳转，ZF=0时不跳转。

循环转移指令

1. 无条件循环指令

要使用loop循环时要提前给CX赋值，给CX赋的值就是你要进行的循环次数，因为每执行一次loop循环CX中存储的值减一，循环结束的标准是CX==0。

因此汇编语言的循环写法大致是：

```
mov cx, 循环次数
s:
循环体（要循环执行的内容）
loop s
```

循环的条件是：当 CX ≠ 0时。

2. 条件循环指令

功能：先使得 CX -1，再根据 CX 的值以及 ZF 的值去决定是否循环。

```
LOOPZ（相等则循环）：当 CX ≠ 0，且 ZF = 1时循环
LOOPNZ（不相等则循环）：当 CX ≠ 0，且 ZF = 0时循环
```

因此，条件循环指令前面需要跟能够改变 ZF 状态的指令，用以控制循环

过程调用指令

转移类指令，是程序运行到某个地方之后，就跳转到另外一段代码，不会再回到原处了；
但是对于过程调用指令，我们跳转到子程序运行完了之后，是需要回到原来的地方继续执行主程序的。

```
调用指令：CALL〈子过程的入口地址〉
返回指令：RET
```

中断控制指令

响应中断，即针对某个随机或异常事件执行一段处理程序，称为中断服务程序，本质上是一种特殊的过程调用，且全部是远过程调用。

指令格式：INT n，n=0 ~ 255
中断返回指令

格式：IRET
中断服务程序的最后一条指令，负责恢复断点 、恢复标志寄存器内容