

选择语句的机器级表示

程序中的选择语句（分支结构）

每取出一条指令，PC 自动+“1”，指向下一条即将执行的指令

PC
程序计数器

地址	主存
...	...
100	指令1
104	指令2
108	指令3
112	指令4
116	指令5
120	指令6
124	指令7
128	指令8
132	指令9
136	指令10
...	...

```
if (a>b){  
    c=a;  
} else {  
    c=b;  
}
```

分支结构可能改变程序的执行流

注：Intel x86 处理器中

程序计数器 PC （Program Counter）
通常被称为 **IP** （Instruction Pointer）

无条件转移指令 jmp

PC

即将执行的指令

程序默认顺序执行

地址	主存
...	...
100	指令1
104	指令2
108	指令3
112	指令4
116	指令5
120	指令6
124	指令7
128	指令8
132	指令9
136	指令10
...	...

当前执行的指令



哔哩哔哩

无条件转移指令——jmp

无条件转移指令，类似与C语言里的 goto 语句

地址

...

```

100 mov eax, 7
104 mov ebx, 6
108 jmp 116
112 mov ecx, ebx
116 mov ecx, eax
120

```



```

mov eax, 7
mov ebx, 6
jmp NEXT
mov ecx, ebx
NEXT:
mov ecx, eax

```

#用“标号”锚定位置

特征——有冒号，名字可以自己取

无条件转移指令

jmp <地址> #PC 无条件转移至 <地址>

jmp 128 #<地址>可以用常数给出

jmp eax #<地址>可以来自于寄存器

jmp [999] #<地址>可以来自于主存

jmp NEXT #<地址>可以用“标号”锚定



条件转移指令 jxxx

```
cmp a,b
```

#比较a和b两个数

a、b两个数可能来自
寄存器/主存/常量

```
je <地址>      #jump when equal, 若a==b则跳转
jne <地址>      #jump when not equal, 若a!=b则跳转
jg <地址>       #jump when greater than, 若a>b则跳转
jge <地址>      #jump when greater than or equal to, 若a>=b则跳转
jl <地址>       #jump when less than, 若a<b则跳转
jle <地址>      #jump when less than or equal to, 若a<=b则跳转
```

条件转移指令一般要和
cmp 指令一起使用

```
cmp eax,ebx    #比较寄存器eax和ebx里的值
jg NEXT        #若 eax > ebx, 则跳转到 NEXT:
```

正确套路

示例：选择语句的机器级表示

```
if (a>b){
    c=a;
} else {
    c=b;
}
```



else 部分的
逻辑

if 部分的
逻辑

```
mov eax,7      #假设变量a=7,存入eax
mov ebx,6      #假设变量b=6,存入ebx
cmp eax,ebx    #比较变量a和b
jg NEXT        #若a>b, 转移到NEXT:
mov ecx,ebx    #假设用ecx存储变量c, 令c=b
jmp END        #无条件转移到END:

NEXT:
mov ecx,eax    #假设用ecx存储变量c, 令c=a
END:
```

```
if (a>b){
    c=a;
} else {
    c=b;
}
```



if 部分的
逻辑

else 部分的
逻辑

```
mov eax,7      #假设变量a=7,存入eax
mov ebx,6      #假设变量b=6,存入ebx
cmp eax,ebx    #比较变量a和b
jle NEXT       #若a≤b, 转移到NEXT:
mov ecx,eax    #假设用ecx存储变量c, 令c=a
jmp END        #无条件转移到END:

NEXT:
mov ecx,ebx    #假设用ecx存储变量c, 令c=b
END:
```

2. 【2019 统考真题】已知 $f(n) = n! = n \times (n-1) \times (n-2) \times \dots \times 2 \times 1$ ，计算 $f(n)$ 的 C 语言函数 fl 的源程序（阴影部分）及其在 32 位计算机 M 上的部分机器级代码如下：

```

int fl(int n){
    1    00401000    55          f1: push ebp
    ...    ...    ...
    if(n>1)
    11    00401018    83 7D 08 01    cmp dword ptr [ebp+8],1
    12    0040101C    7E 17          jle f1+35h (00401035)
    return n*fl(n-1);
    13    0040101E    8B 45 08    mov eax, dword ptr [ebp+8]
    14    00401021    83 E8 01    sub eax, 1
    15    00401024    50          push eax
    16    00401025    E8 D6 FF FF FF    call fl ( 00401000)
    ...    ...    ...
    19    00401030    0F AF C1    imul eax, ecx
    20    00401033    EB 05          jmp f1+3Ah (0040103a)
    else return 1;
    21    00401035    B8 01 00 00 00    mov eax,1
}
    ...    ...    ...
    26    00401040    3B EC          cmp ebp, esp
    ...    ...    ...
    30    0040104A    C3          ret

```

其中，机器级代码行包括行号、虚拟地址、机器指令和汇编指令，计算机 M 按字节编址，int 型数据占 32 位。请回答下列问题：



写汇编语言代码时，一般会以函数名作为“标号”，标注该函数指令的起始地址

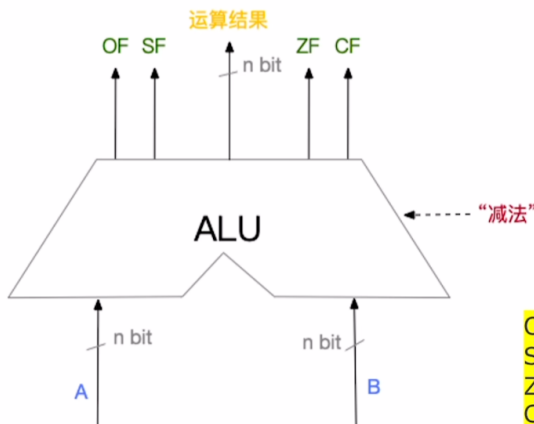
cmp 指令的底层原理



扩展：cmp 指令的底层原理

ALU 每次运算的标志位都自动存入 PSW 程序状态字寄存器（Intel 称其为“标志寄存器”）

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
			OF				SF	ZF							CF



本质上是进行 a-b 减法运算，并生成标志位 OF、ZF、CF、SF

cmp a,b #比较a和b两个数

je <地址> #若a==b则跳转，ZF==1 ?
 jne <地址> #若a!=b则跳转，ZF==0 ?
 jg <地址> #若a>b则跳转，ZF==0 && SF==OF ?
 jge <地址> #若a>=b则跳转，SF==OF ?
 jl <地址> #若a<b则跳转，SF!=OF ?
 jle <地址> #若a<=b则跳转，SF!=OF || ZF==1 ?

OF (Overflow Flag) 溢出标志。溢出时为1，否则置0。
 SF (Sign Flag) 符号标志。结果为负时置1，否则置0。
 ZF (Zero Flag) 零标志，运算结果为0时ZF位置1，否则置0。
 CF (Carry Flag) 进位/借位标志，进位/借位时置1，否则置0。