X86 基本指令简介

转移指令

转移指令

指令执行：跳转到转移目标指令处执行

- 无条件转移指令
  - *JMP D*：无条件转移到目标指令D处执行
- 条件转移
  - *Jcc D*：根据cc 状态标志（条件码）判断是否满足条件，若满足，转移到目标指令D处执行，否则按顺序执行

无条件转移指令

- **短转移：** JMP SHORT LABEL
  - IP←IP+8位的位移量（-128~127Byte）
  - 位移量是一个长度为8位的带符号数，为LABEL的偏移地址与当前EIP/IP值之差
- **近转移：** JMP NEAR PTR LABEL
  - IP←IP+16位的位移量（±32KByte）
  - 位移量是一个长度为16位的带符号数，为LABEL的偏移地址与当前EIP/IP值之差
  - 从80386开始，近转移可使用32位的位移量

无条件转移指令

- **远转移：** JMP FAR PTR LABEL
  - IP←LABEL的偏移地址；CS ←LABEL的段基值
- **间接转移：转移目标地址在寄存器中**
  - JMP EAX ； EAX → EIP
- **间接转移：转移目标地址在存储器中**
  - JMP [ESI] ； [ESI] → EIP

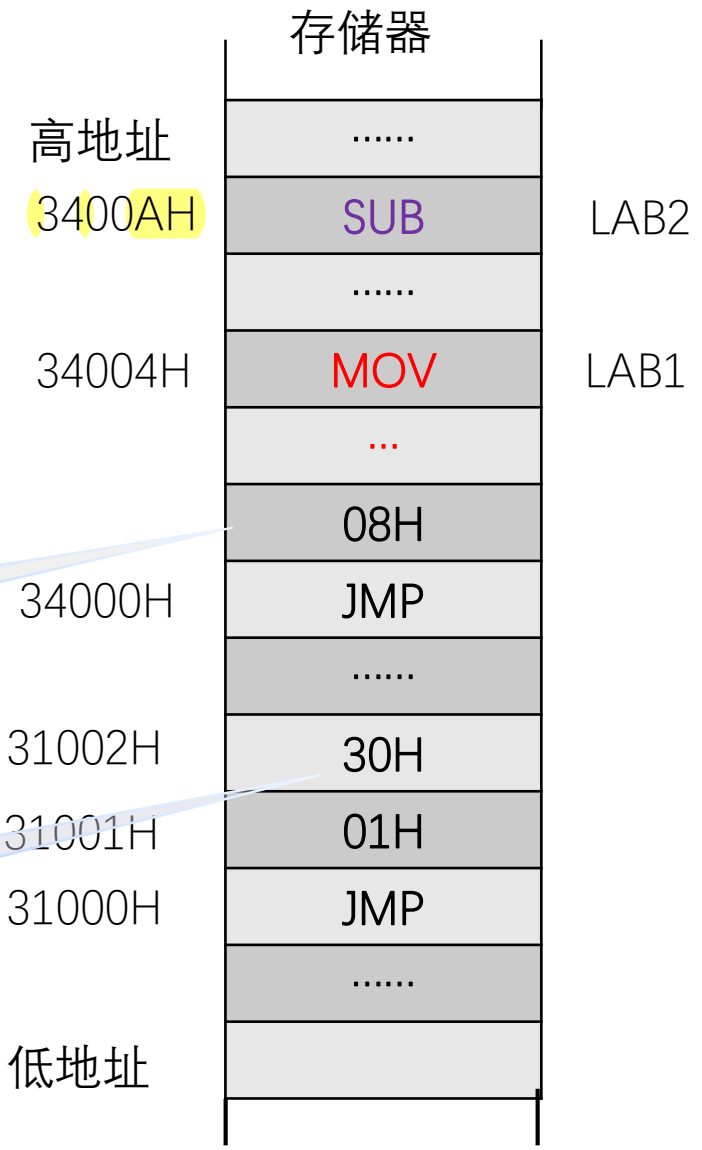
无条件短转移/近转移示例

```
        JMP  NEAR PTR LAB2
         ......
        JMP SHORT LAB1

LAB1:  MOV AX，DX
        ......
LAB2:  SUB   BX，AX
        ......
```

LAB2-IP
= 3400AH-(34000H+2)
=08H

LAB1-IP
= 34004H-(31000H+3)
=3001H



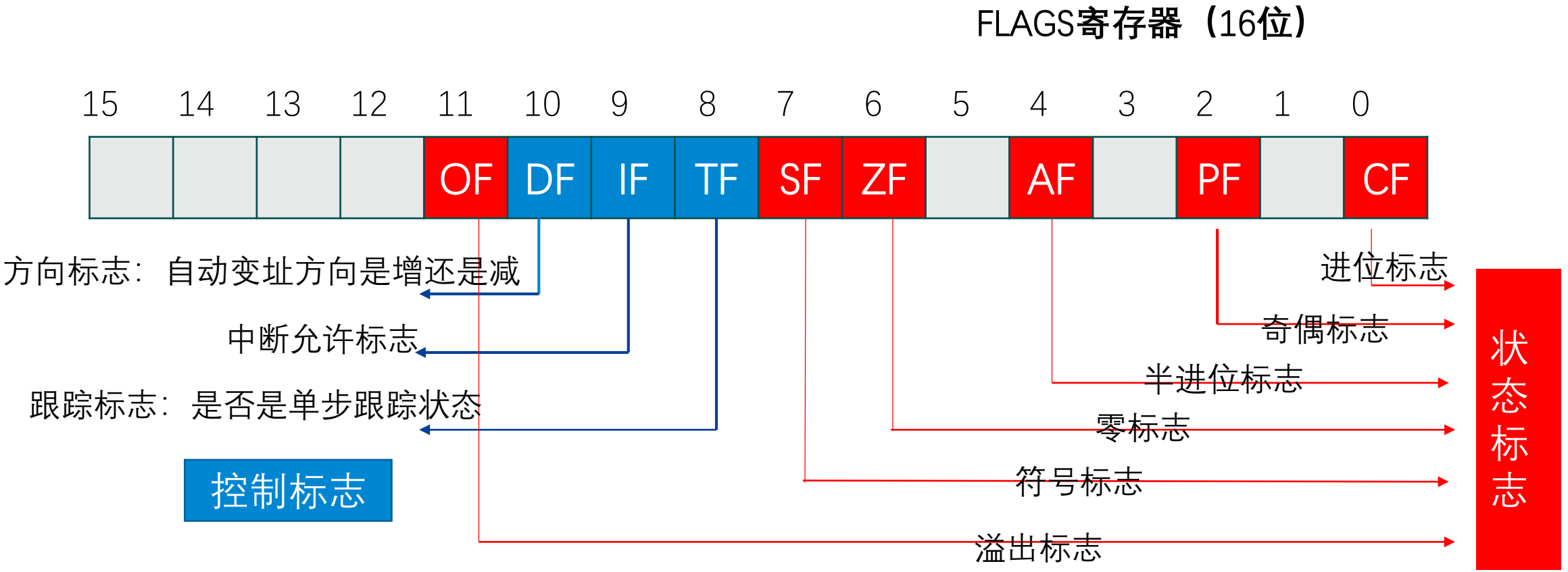
条件转移指令

jX 指令

- 根据条件码的取值，跳转到代码的不同部分

jX 指令	条件	描述
jmp	1	无条件转移
je	ZF	相等/为0 时跳转
jne	~ZF	Not Equal / Not Zero
js	SF	Negative
jns	~SF	Nonnegative
jg	~(SF^OF)&~ZF	Greater (Signed)
jge	~(SF^OF)	Greater or Equal (Signed)
jl	(SF^OF)	Less (Signed)
jle	(SF^OF)|ZF	Less or Equal (Signed)
ja	~CF&~ZF	Above (unsigned)
jb	CF	Below (unsigned)

条件转移：标志寄存器


FLAGS寄存器（16位）
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
方向标志：自动变址方向是增还是减
中断允许标志
跟踪标志：是否是单步跟踪状态
控制标志
进位标志
奇偶标志
半进位标志
零标志
符号标志
溢出标志
状态标志

条件码

CF 进位标志 (对 unsigned)　　**SF** 符号标志 (对 signed)
ZF　　零标志　　**OF** 溢出标志 (对 signed)

- 算术运算操作可以隐含地设置状态标志
- 例如: **addq** *Src*, *Dest*　代表:　**t = a+b**

CF 置1 ： 如果无符号数加法在最高位产生进位、或减法产生借位 (无符号数溢出)

ZF 置1 ： 如果**t == 0**，结果为零置**1**

SF 置1 ： 如果**t < 0** (作为 signed)， 负数置1

OF 置1 ： 如果带符号数，即：2的补码 (带符号数)运算溢出

即: ***(a>0 && b>0 && t<0) || (a<0 && b<0 && t>=0)***

- 注意：leaq 、lea 指令不会设置状态标志

整数运算与条件码

状态寄存器中的状态标志：

CF 进/借位标志（对 unsigned）　　**SF** 符号标志 (对 signed)
ZF 零标志　　**OF** 溢出标志 (对 signed)

-7-5　　9-5

1 0 0 1　(-7)　(9)
\+ 1 0 1 1　(-5)　(-5)

(4) ×　(4) √

1 0 1 0 0

OF=1、　ZF=0、　SF=0、
CF=0（无符号数减法没有借位）

-3-4　　13-4

1 1 0 1　(-3)　(13)
\+ 1 1 0 0　(-4)　(-4)

(-7) √　(9) √

1 1 0 0 1

OF=0、　ZF=0、　SF=1、
CF=0（无符号数减法没有借位）

可利用条件标志进行大小判断：做减法以比较大小，规则如下：

Unsigned	CF=0，ZF≠0 大于	**例如：** 9-5； 13-4； **均** CF=0
signed	OF=SF， ZF≠0 大于	**例如：** -7-5 OF≠SF； -3-4 OF≠ SF

条件码 (用比较指令显示设置)

比较指令:

cmpq *Src2, Src1*
cmpq b,a 进行 **a-b** 运算但不设置运算结果到目标寄存器

- **CF set**: 如果最高位产生进位 (无符号数溢出: unsigned overflow)
- **ZF set**: 如果 **a == b**
- **SF set**: 如果 **(a-b) < 0** (作为 signed)
- **OF set**: 如果2的补码 (带符号数)运算 溢出

  即: **(a>0 && b<0 && (a-b)<0) || (a<0 && b>0 && (a-b)>0)**

条件码 (显式设置：Test指令)

Test 指令：

testq *Src2*, *Src1*
testq b,a 进行 **a&b** 运算但不设置运算结果到目标寄存器

- 当其中一个操作数为掩码时，这条指令很有用
-
- **ZF set** 当 **a&b == 0**
- **SF set** 当 **a&b < 0**

条件转移指令举例

- 编译优化，生成利于调试的可执行代码

```
mylinux > gcc -Og -S -fno-if-conversion control.c
```

```
long absdiff
  (long x, long y)
{
  long result;
  if (x > y)
    result = x-y;
  else
    result = y-x;
  return result;
}
```

```
absdiff:
    cmpq    %rsi, %rdi   # x:y
    jle     .L4
    movq    %rdi, %rax   # %rdi 参数x
    subq    %rsi, %rax   # %rsi 参数y
    ret

.L4:                     # x <= y
    movq    %rsi, %rax
    subq    %rdi, %rax   # %rax 函数返回结果
    ret
```

注：即使在64系统上，有些编译器仍然认为long 是4字节，但所有编译器都认定 long long int 是8字节

条件传送指令（Conditional Move）：变为无跳转指令

```
long absdiff
   (long x, long y)
{
    long result;
    if (x > y)
        result = x-y;
    else
        result = y-x;
    return result;
}
```

```
result = Test ? Then_Expr : Else_Expr;
```

```
result = Then_Expr;
eval = Else_Expr;
nt = !Test;
if (nt) result = eval;
return result;
```

寄存器	用法
%rdi	Argument x
%rsi	Argument y
%rax	Return value

```
absdiff:
    movq    %rdi, %rax   # x
    subq    %rsi, %rax   # result = x-y
    movq    %rsi, %rdx
    subq    %rdi, %rdx   # eval = y-x
    cmpq    %rsi, %rdi   # x:y
    cmovle  %rdx, %rax   # if <=, result = eval
    ret
```

“Do-While” 循环

C 代码

```
long pcount_do
   (unsigned long x) {
   long result = 0;
   do {
     result += x & 0x1;
     x >>= 1;
   } while (x);
   return result;
}
```

Goto 版

```
long pcount_goto
   (unsigned long x) {
   long result = 0;
 loop:
   result += x & 0x1;
   x >>= 1;
   if(x) goto loop;
   return result;
}
```

- 计算参数x 中“1”的个数

“Do-While” 循环编译(64位)

Goto 版

```
long pcount_goto
  (unsigned long x) {
  long result = 0;
 loop:
  result += x & 0x1;
  x >>= 1;
  if(x) goto loop;
  return result;
}
```

使用条件转移指令继续进入或离开循环

```
   movl    $0, %eax    #  result = 0
.L2:                   # loop:
   movq    %rdi, %rdx
   andl    $1, %edx    #  t = x & 0x1
   addq    %rdx, %rax  #  result += t
   shrq    %rdi        #  x >>= 1
   jne     .L2         #  if (x) goto loop
   retq
```

寄存器	用法
%rdi	参数 x
%rax	返回结果 result

小结

- X86-64 控制转移指令