



南京大學  
NANJING UNIVERSITY



# IA-32中的控制转移指令

南京大学

计算机科学与技术系

袁春风

email: [cfyuan@nju.edu.cn](mailto:cfyuan@nju.edu.cn)

2015.6

# IA-32常用指令类型

---

## (4) 控制转移指令

指令执行可**按顺序**或**跳转到转移目标指令处**执行

- 无条件转移指令

JMP DST : 无条件转移到目标指令DST处执行

- 条件转移

Jcc DST : cc为条件码, 根据标志(条件码)判断是否满足条件, 若满足, 则转移到目标指令DST处执行, 否则按顺序执行

- 条件设置

SETcc DST : 按条件码cc判断的结果保存到DST(是一个8位寄存器)

- 调用和返回指令(用于过程调用)

CALL DST : 返回地址RA入栈, 转DST处执行

RET : 从栈中取出返回地址RA, 转到RA处执行

- 中断指令(详见第7、8章)

# IA-32的标志寄存器

31-22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
保留	ID	VIP	VIF	AC	VM	RF	0	NT	IOPL		0	D	I	T	S	Z	0	A	0	P	1	C

← 80286/386
← 8086 →

- 6个条件标志

- OF、SF、ZF、CF各是什么标志（条件码）？
- AF：辅助进位标志（BCD码运算时才有意义）
- PF：奇偶标志

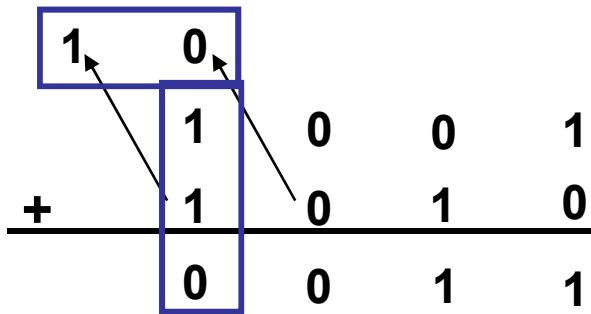
- 3个控制标志

- DF（Direction Flag）：方向标志（自动变址方向是增还是减）
- IF（Interrupt Flag）：中断允许标志（仅对外部可屏蔽中断有用）
- TF（Trap Flag）：陷阱标志（是否是单步跟踪状态）

- .....

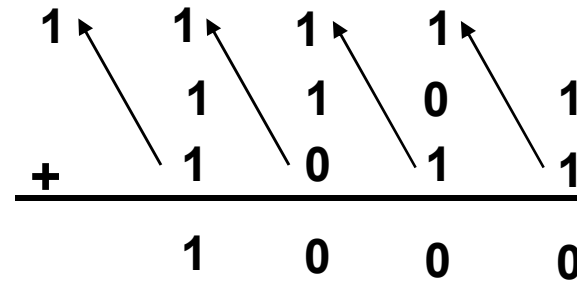
# 回顾：整数减法举例

$$\begin{array}{l} -7 - 6 = -7 + (-6) = +3 \text{ x} \\ 9 - 6 = 3 \checkmark \end{array}$$



OF=1、ZF=0  
SF=0、借位CF=0

$$\begin{array}{l} -3 - 5 = -3 + (-5) = -8 \checkmark \\ 13 - 5 = 8 \checkmark \end{array}$$



OF=0、ZF=0、  
SF=1、借位CF=0

## 可利用条件标志进行大小判断

做减法以比较大小，规则：  
Unsigned: CF=0时，大于  
Signed : OF=SF时，大于

验证：9>6，故CF=0；13>5，故CF=0

验证：-7<6，故OF≠SF  
-3<5，故OF≠SF

## 分三类：

### (1)根据单个标志的值转移

### (2)按无符号整数比较转移

### (3)按带符号整数比较转移

序号	指令	转移条件	说明
1	jc label	CF=1	有进位/借位
2	jnc label	CF=0	无进位/借位
3	je/jz label	ZF=1	相等/等于零
4	jne/jnz label	ZF=0	不相等/不等于零
5	js label	SF=1	是负数
6	jns label	SF=0	是非负数
7	jo label	OF=1	有溢出
8	jno label	OF=0	无溢出
9	ja/jnbe label	CF=0 AND ZF=0	无符号整数 $A > B$
10	jae/jnb label	CF=0 OR ZF=1	无符号整数 $A \geq B$
11	jb/jnae label	CF=1 AND ZF=0	无符号整数 $A < B$
12	jbe/jna label	CF=1 OR ZF=1	无符号整数 $A \leq B$
13	jg/jnle label	SF=OF AND ZF=0	带符号整数 $A > B$
14	jge/jnl label	SF=OF OR ZF=1	带符号整数 $A \geq B$
15	jl/jnge label	SF $\neq$ OF AND ZF=0	带符号整数 $A < B$
16	jle/jng label	SF $\neq$ OF OR ZF=1	带符号整数 $A \leq B$

# 例子：程序的机器级表示与执行

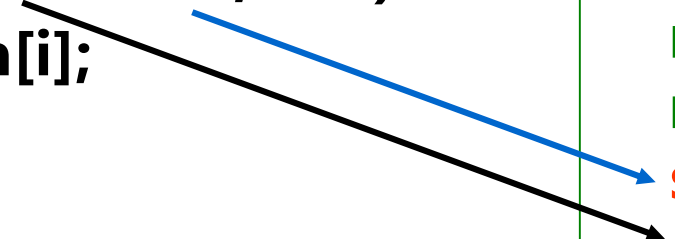
```
int sum(int a[ ], unsigned len)
{
    int i, sum = 0;
    for (i = 0; i <= len-1; i++)
        sum += a[i];
    return sum;
}
```

当参数len为0时，返回值应该是0，但是在机器上执行时，却发生了存储器访问异常。 **Why?**

**i 和 len 分别在哪个寄存器中？**

**i : %eax ; len : %edx**

```
sum:
    ...
.L3:
    ...
    movl -4(%ebp), %eax
    movl 12(%ebp), %edx
    subl $1, %edx
    cmpl %edx, %eax
    jbe .L3
    ...
```



**第一次循环，执行结果是什么？**

**%eax: 0000 ..... 0000**

**%edx: 0000 ..... 0000**

**subl 指令的执行结果是什么？**

**cmpl 指令的执行结果是什么？**

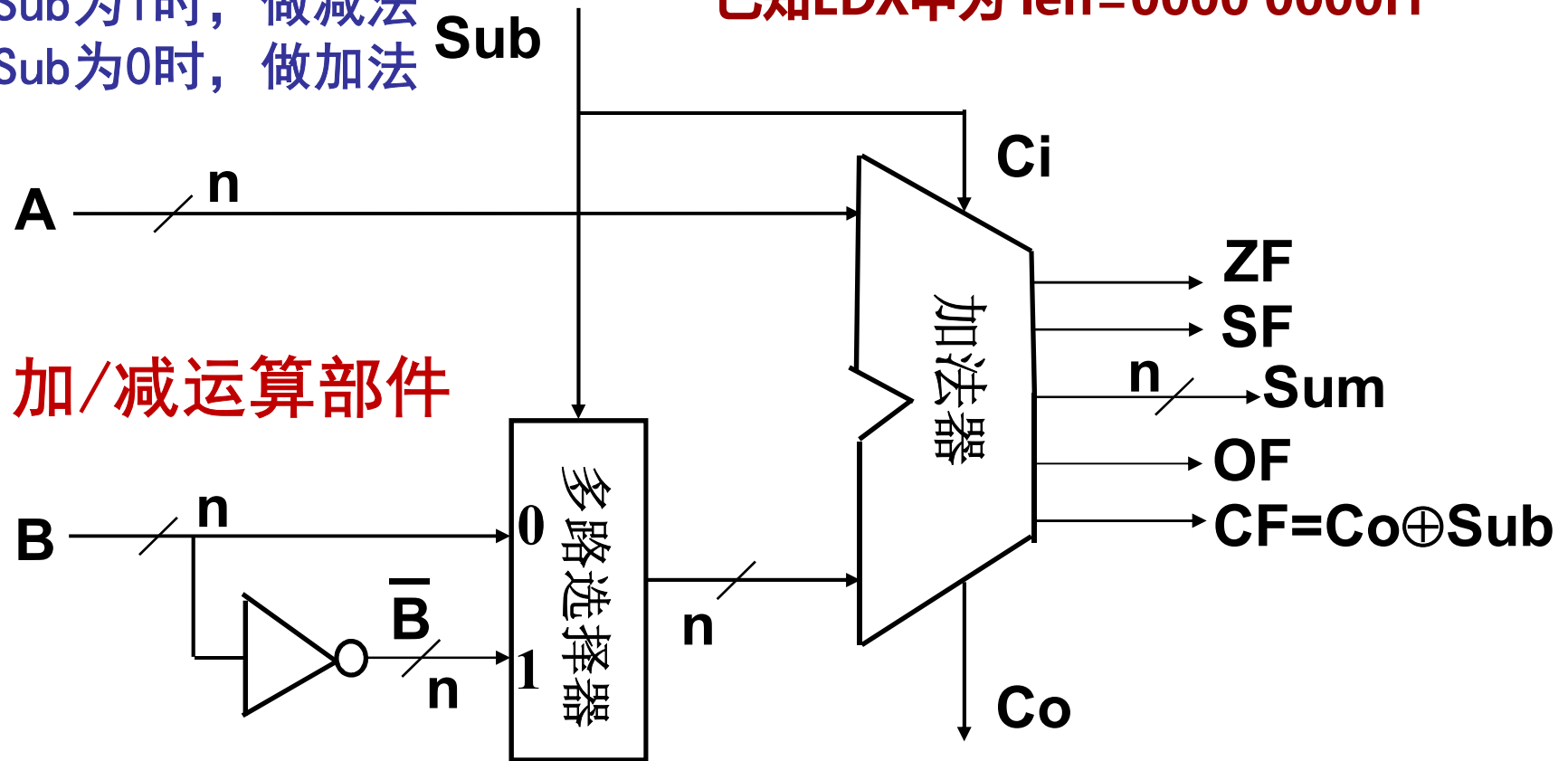
# subl \$1, %edx指令的执行结果

当Sub为1时，做减法  
当Sub为0时，做加法

Sub

已知EDX中为 len=0000 0000H

加/减运算部件



“subl \$1, %edx” 执行时：A=0000 0000H，B为0000 0001H，  
Sub=1，因此Sum是32个1，即R[edx]=FFFFFFFFH=0xffffffff

完全等价的两种不同写法！

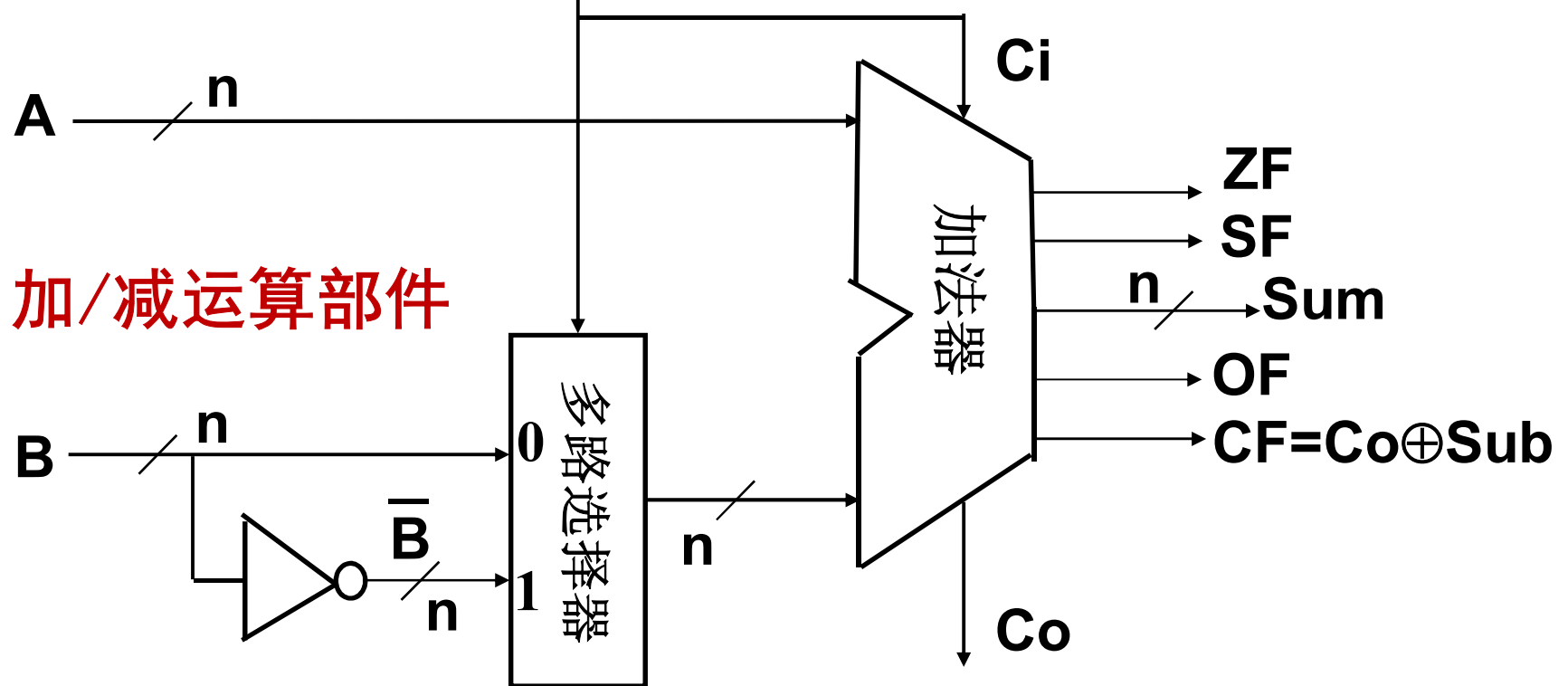
# cpml %edx,%eax指令的执行结果

当Sub为1时，做减法  
当Sub为0时，做加法

Sub

已知EDX中为 len-1=FFFF FFFFH

EAX中为 i=0000 0000H



“cpml %edx,%eax” 执行时：A=0000 0000H，B为FFFF FFFFH，Sub=1，因此Sum是0...01，CF=1，ZF=0，OF=0，SF=0



## jbe .L3指令的执行结果

指令	转移条件	说明
JA/JNBE label	CF=0 AND ZF=0	无符号数A > B
JAE/JNB label	CF=0 OR ZF=1	无符号数A ≥ B
JB/JNAE label	CF=1 AND ZF=0	无符号数A < B
JBE/JNA label	CF=1 OR ZF=1	无符号数A ≤ B
JG/JNLE label	SF=OF AND ZF=0	带符号整数A > B
JGE/JNL label	SF=OF OR ZF=1	带符号整数A ≥ B
JL/JNGE label	SF≠OF AND ZF=0	带符号整数A < B
JLE/JNG label	SF≠OF OR ZF=1	带符号整数A ≤ B

“**cmpl %edx,%eax**” 执行结果是 **CF=1, ZF=0, OF=0, SF=0** ,  
因此 , 在执行 “**jbe .L3**” 时满足条件 , 应转移到.L3执行 !

# 例子：程序的机器级表示与执行

```
int sum(int a[ ], unsigned len)
{
    int i , sum = 0;
    for (i = 0; i <= len-1; i++)
        sum += a[i];
    return sum;
}
```

当参数len为0时，返回值应该是0，但是在机器上执行时，却发生了存储器访问异常。 **Why?**

```
sum:
    ...
.L3:
    ...
    movl -4(%ebp), %eax
    movl 12(%ebp), %edx
    subl $1, %edx
    cmpl %edx, %eax
    jbe .L3
    ...
```

“**cmpl %edx,%eax**” 执行结果是 **CF=1, ZF=0, OF=0, SF=0**，说明满足条件，应转移到.L3执行！显然，对于每个i都满足条件，因为任何无符号数都比32个1小，因此循环体被不断执行，最终导致数组访问越界而发生存储器访问异常。

# 例子：程序的机器级表示与执行

例：

```
int sum(int a[ ], int len)
{
    int i, sum = 0;
    for (i = 0; i <= len-1; i++)
        sum += a[i];
    return sum;
}
```

正确的做法是将参数len声明为int型。 **Why?**

```
sum:
    ...
.L3:
    ...
    movl -4(%ebp), %eax
    movl 12(%ebp), %edx
    subl $1, %edx
    cmpl %edx, %eax
    jle .L3
    ...
```

“sub \$1,%edx” 和 “cmpl %edx,%eax” 执行结果与前面一样！  
执行到 “jle .L3” 指令时，也是 **CF=1, ZF=0, OF=0, SF=0**！

## jle .L3指令的执行结果

指令	转移条件	说明
JA/JNBE label	CF=0 AND ZF=0	无符号数A > B
JAE/JNB label	CF=0 OR ZF=1	无符号数A ≥ B
JB/JNAE label	CF=1 AND ZF=0	无符号数A < B
JBE/JNA label	CF=1 OR ZF=1	无符号数A ≤ B
JG/JNLE label	SF=OF AND ZF=0	带符号整数A > B
JGE/JNL label	SF=OF OR ZF=1	带符号整数A ≥ B
JL/JNGE label	SF≠OF AND ZF=0	带符号整数A < B
JLE/JNG label	SF≠OF OR ZF=1	带符号整数A ≤ B

“**cmpl %edx,%eax**” 执行结果是 **CF=1, ZF=0, OF=0, SF=0** ,  
因此, 在执行 “**jle .L3**” 时不满足条件, 应跳出循环执行, 使得  
执行结果正常。

# 例子：C表达式类型转换顺序

unsigned long long

↑

long long

↑

unsigned

↑

int

↑

(unsigned) char  
(unsigned) short

```
#include <stdio.h>
void main()
{
    unsigned int a = 1;
    unsigned short b = 1;
    char c = -1;
    int d;

    d = (a > c) ? 1:0;
    printf("%d\n", d);
    d = (b > c) ? 1:0;
    printf("%d\n", d);
}
```

猜测：各用哪种条件设置指令？

条件设置指令：

SETcc DST：按条件码cc判断的结果保存到DST

0804841c <main>:

804841c: 55 push %ebp

#include <stdio.h>

void main()

{

unsigned int a = 1;

unsigned short b = 1;

char c = -1;

int d;

d = (a > c) ? 1:0;

printf("%d\n",d);

d = (b > c) ? 1:0;

printf("%d\n",d);

}

804846c: 0f 9f c0 movsb %al,%eax

804846f: 0f b6 c0 movsb %al,%eax

8048472: 89 44 24 14 mov %eax,0x14(%esp)

8048476: 8b 44 24 14 mov %eax,0x14(%esp)

804847a: 89 44 24 04 mov %eax,0x4(%esp)

804847e: c7 04 24 20 85 04 08 movl \$0x8048520,(%esp)

8048485: e8 76 fe ff ff call 8048300 <printf@plt>

804848a: c9 leave

804848b: c3 ret

push %ebp

mov %esp,%ebp

and \$0xfffffffff0,%esp

sub \$0x20,%esp

00 00 movl \$0x1,0x1c(%esp)

01 00 movw \$0x1,0x1a(%esp)

movb \$0xff,0x19(%esp)

movsbl 0x19(%esp),%eax

cmp 0x1c(%esp),%eax

setb %al

无符号

movzbl %al,%eax

mov %eax,0x14(%esp)

mov 0x14(%esp),%eax

mov %eax,0x4(%esp)

04 08 movl \$0x8048520,(%esp)

call 8048300 <printf@plt>

movzwl 0x1a(%esp),%edx

movsbl 0x19(%esp),%eax

cmp %eax,%edx

setg %al

带符号

movzbl %al,%eax

mov %eax,0x14(%esp)

mov 0x14(%esp),%eax

mov %eax,0x4(%esp)

movl \$0x8048520,(%esp)

call 8048300 <printf@plt>

leave

ret

执行结果是？

0

1