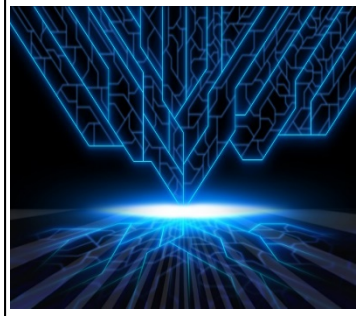
第8讲
算术和逻辑操作



吴海军
南京大学计算机科学与技术系

算术和逻辑运算指令

指令		效果	描述
leaq	S, D	$D \leftarrow \&S$	加载有效地址
INC	D	$D \leftarrow D+1$	加1
DEC	D	$D \leftarrow D-1$	减1
NEG	D	$D \leftarrow -D$	取负
NOT	D	$D \leftarrow \sim D$	取补
ADD	S, D	$D \leftarrow D+S$	加
SUB	S, D	$D \leftarrow D-S$	减
IMUL	S, D	$D \leftarrow D*S$	乘
XOR	S, D	$D \leftarrow D \hat{} S$	异或
OR	S, D	$D \leftarrow D \mid S$	或
AND	S, D	$D \leftarrow D \& S$	与
SAL	k, D	$D \leftarrow D << k$	左移
SHL	k, D	$D \leftarrow D << k$	左移（等同于SAL）
SAR	k, D	$D \leftarrow D >>_A k$	算术右移
SHR	k, D	$D \leftarrow D >>_L k$	逻辑右移

一元操作，只有1个操作数

二元操作，第二个操作数既是源操作数又是目的操作数。

两个操作数不能同时是存储器的位置。

移位操作，先给出移位量，单字节，可以使用立即数，或存放在寄存器%cl中。算术右移SAR，左端补充符号位。

加载有效地址指令leaq

- **leaq** *Src*, *Dest* 加载有效地址，实际上是movq指令的变形。
  - 形式上是从内存读数据到寄存器，
  - 实际上只是把有效地址写入目的操作数（寄存器）
  - 执行简便的算术运算，形如x + k*y的算术表达式

```
Leaq 7(%rdx,%rdx,4), %rax
```

```
%rax=5%rdx+7
```

```
long scale(long x, long y, long z)
 {
     long t = x + 4 * y + 12 * z;
     return t;
}
```

```
long scale(long x, long y, long z)
 x in%rdi , y in%rsi , z in%rdx
scale:
 leaq     (%rdi,%rsi,4), %rax
 leaq     (%rdx,%rdx,2), %rdx
 leaq     (%rax,%rdx,4), %rax
 ret
```

一些算术操作

- 一元操作数指令，只有一个操作数既是源又是目的

格式	运算
`inc` *Dest*	*Dest* = *Dest* + 1
`dec` *Dest*	*Dest* = *Dest* – 1
`neg` *Dest*	*Dest* = – *Dest*
`not` *Dest*	*Dest* = ~ *Dest*

div *Dest*　　EDX：EAX除以Dest，商在EAX中，余数在EDX中

mul *Dest*　Dest 乘以%EAX，乘积的高位在EDX中，低位在EAX中

Pushq *Dest*

Popq *Dest*

一些算术操作

- 二元操作指令，第2个操作数既是源又是目的

格式		运算	
add	*Src,Dest*	*Dest = Dest + Src*	
sub	*Src,Dest*	*Dest = Dest - Src*	
imul	*Src,Dest*	*Dest = Dest * Src*	
sal	*Src,Dest*	*Dest = Dest << Src*	*也叫做*shll
sar	*Src,Dest*	*Dest = Dest >> Src*	算术右移
shr	*Src,Dest*	*Dest = Dest >> Src*	逻辑右移
xor	*Src,Dest*	*Dest = Dest ^ Src*	
and	*Src,Dest*	*Dest = Dest & Src*	
or	*Src,Dest*	*Dest = Dest | Src*	
Cmp	*Src,Dest*	*Flags= Src cmp Dest*	
Xchg	*Src,Dest*	*Src <->Dest*	

算术运算示例

- 假设下面的值存放在指定的内存地址和寄存器中

地址	值
0x100	0xFF
0x108	0xAB
0x110	0x13
0x118	0x11

寄存器	值
%rax	0x100
%rcx	0x1
%rdx	0x3

指令	目的	值
addq %rcx, (%rax)	0x100	0x100
subq %rdx, 8 (%rax)	0x108	0xA5
imulq $16,(%rax,%rdx,8)	0x118	0x110
incq 16 (%rax)	0x110	0x14
decq %rcx	%rcx	0x0
subq %rdx, %rax	%rax	0xFD

移位操作

- 先给出移位量， 然后第二项给出的是要移位的数。
- 移位量可以是一个立即数，或者放在单字节寄存器%cl中。
- 1个字节的移位量的编码范围可以达到2^8-1=255。
- x86-64中，移位操作对n 位长的数据值进行操作，移位量是由%cl寄存器的低m位决定的，这里2^m=n, 高位被忽略。
  - 例如当寄存器%cl的十六进制值为0xFF时，指令salb会移7位，salw会移15 位，sall会移 31位，而salq会移63位。
- 左移指令 SAL和SHL,两者的效果是一样的。
- 右移指令SAR执行算术移位（填上符号位），而SHR执行逻辑移位（填 上0)。
- 移位操作的目的操作数可以是一个寄存器或是一个内存位置

移位操作示例

- 根据C函数，完成汇编代码

```
long shift_left4_rightn(long x, long n)
{
    x<<=4;
    x>>=n;
    return x
}
```

参数 x和n分别存放在寄存器%rdi和%rsi中。

```
shift_left4_rightn:
    movq %rdi, %rax      // Get x
    Salq   $4,%rax       // x<<=4
    movl %esi, %ecx      // get n
    Sarq   %cl,%rax      // x>>=n
```

汇编指令功能示例

- C函数汇编代码的功能

```
long arith(long x, long y, long z)
 {
     long t1= x^y;
     long t2 = z * 48;
     long t3 = t1 & 0x0F0F0F0F;
     long t4 = t2 -t3;
     return t4;
}
```

```
// x in %rdi, y in %rsi, z in%rdx
arith:
   xorq    %rsi, %rdi
   leaq    (%rdx,%rdx,2), %rax
   salq    $4, %rax
   andl    $252645135, %edi
   subq    %rdi, %rax
   ret
```

说出每一行汇编指令的功能

乘法指令

- imulq指令有两种不同的形式
  - 双操作数，从两个64位操作数产生一个64位的乘积
  - 单操作数，从两个64位操作数产生一个128位的乘积
- X86-64指令集提供了对128位数的操作支持。

指令	效　果	描述
imulq S Mulq S	R[%rdx]: R[%rax] ← S×R[%rax] R[%rdx]: R[%rax] ← S×R[%rax]	有符号全乘法 无符号全乘法
cqto	R[%rdx]: R[%rax] ← 符号扩展R[%rax]	转换为八字
idivq S	R[%rdx] ← R[%rdx]: R[%rax] mod S R[%rax] ← R[%rdx]: R[%rax] ÷ S	有符号除法
divq S	R[%rdx] ← R[%rdx]: R[%rax] mod S R[%rax] ← R[%rdx]: R[%rax] ÷ S	无符号除法

乘法指令

- imulq指令单操作数运算，隐含要求一个参数必须在寄存器%rax中。乘积存放在寄存器%rdx（高64位）和%rax（低64位）中。
- 两个无符号64位数字乘法示例

```
#include <inttypes.h>
typedef unsigned __ int128 uint128_t;
void store_uprod(uint128_t *dest, uint64_t x, uint64_t y)
{
    *dest = x *(uint128_t) y;
}
```

参数 dest in %rdi, x in %rsi, y in %rdx

```
store_uprod:
    movq    %rsi, %rax
    mulq    %rdx
    movq    %rax, (%rdi)      存储低8个字节
    movq    %rdx, 8(%rdi)     存储高8个字节
    ret
```

除法指令

- 单操作数除法指令idivq将寄存器%rdx(高64位)和%rax(低64位)中的128位数作为被除数，除数作为指令的操作数给出
- 指令将商存储在寄存器%rax中，余数存储在寄存器%rdx中。
- cqto指令，没有操作数，读出%rax的符号位，并复制到%rdx的所有位。

```
void remdiv(long x, long y, long *qp, long *rp)
 {
    long q = x/y;
    long r = x%y;
    *qp = q;
    *rp=r;
}
```

```
//  x in %rdi , y in %rsi , qp in %rdx, rp in %rcx
 remdiv:
    movq      %rdx, %r8
    movq      %rdi, %rax
    cqto
    idivq      %rsi
    movq       %rax, (%r8)
    movq       %rdx, (%rcx)
    ret
```

把参数qp保存到另一个寄存器中

准备被除数， 复制并符号扩展x

寄存器%rax 中的商被保存在qp

寄存器%rdx 中的余数被保存在rp