作者: 代兴 邮箱: 503268771@qq.com Vx 公众号: 东方架构师

指令说明

div 指令表示无符号整数除法。div 全称为 divide,即除法含义。div 指令分为 8 位 divb、16 位 divw、32 位 div1、64 位 divq。b、w、1、q 分别表示 byte、word、long、quad,即 1 字节、2 字节、4 字节、8 字节。(下同)div 指令可以操作寄存器、内存。

idiv 指令表示有符号整数除法。idiv 全称为 signed divide,即除法带上符号。idiv 指令分为 8 位 idivb、16 位 idivw、32 位 idiv1、64 位 idivq。idiv 指令可以操作寄存器、内存。

语法格式 div cc 表示 rax / cc = rax, rdx 。例外, 8 位除法 divb cc 表示 al / cc = al, ah。 语法格式 idiv cc 表示 rax / cc = rax, rdx 。例外, 8 位除法 idivb cc 表示 al / cc = al, ah。

结果 rax, rdx 表示商在寄存器 rax, 余数在寄存器 rdx。

除法的结果通常使用 2 个寄存器 (rax/rdx), 所以需要先给被除数执行符号扩展或零扩展, 把 1 个寄存器的值扩展到 2 个寄存器。

比如,不能直接用32位除以32位,需要先把被除数扩展为64位,再执行除法。

有符号整数除法,符号扩展使用 cltd、cqto。

cltd 把 eax 扩展为 edx:eax , cltd 可以理解为 convert long to doubleword。
cqto 把 rax 扩展为 rdx:rax , cqto 可以理解为 convert quadword to octoword。

无符号整数除法,零扩展使用 mov。比如, movq \$0, %rdx。

简单的除法

编写代码: div.s

```
. text
.global main
main:
   pushq %rbp
   movq %rsp, %rbp
   subq $64, %rsp
   #64位。操作寄存器。有符号除法
                     #被除数, rax
   movq $17, %rax
                     # 符号扩展 rax > rdx:rax
   cqto
   movq $-3, %r8
                     # 除数, r8
   idivq %r8
                     \# rax / r8 = rax, rdx
   movq $str_int64, %rdi
   movq %rax, %rsi
                     # 商
                     # 余数
   movq %rdx, %rdx
   callq printf
   #64位。操作寄存器。无符号除法
                 #被除数, rax
   movq $17, %rax
   movq $0, %rdx
                     #零扩展,rdx,清零
                     # 除数, r8
   movq $3, %r8
                     \# rax / r8 = rax, rdx
   divq %r8
   movq $str_int64, %rdi
   movq %rax, %rsi
                     # 商
   movq %rdx, %rdx
                     # 余数
   callq printf
   #32位。操作寄存器。
   mov1 $9, %eax
                     #被除数,eax
   cltd
                     # 符号扩展 eax > edx:eax
   mov1 $-2, %r8d
                     # 除数, r8d
   idivl %r8d
                     \# eax / r8d = eax, edx
   movq $str int32, %rdi
                     # 商
   mov1 %eax, %esi
   mov1 %edx, %edx
                     # 余数
   callq printf
   # 32 位。操作栈。
                     #被除数,eax
   mov1 $-17, %eax
```

```
c1td
                  # 符号扩展 eax > edx:eax
                  # 除数, -4(%rbp)
mov1 $3, -4(%rbp)
idivl -4(%rbp)
                  \# \text{ eax } / (-4 (\% \text{rbp})) = \text{eax, edx}
movq $str_int32, %rdi
mov1 %eax, %esi
                  # 商
mov1 %edx, %edx
                  # 余数
callq printf
# 16 位。操作寄存器。
#被除数,ax
movw $13, %ax
movsw1 %ax, %eax
                    # 符号扩展, 32 位
cltd
                     # 符号扩展 eax > edx:eax
movw $-5, %r8w
                   # 除数, r8w
idivw %r8w
                     \# eax / r8d = eax, edx
movswl %ax, %eax
                    # 商
movswl %dx, %edx
                     # 余数
movq $str int32, %rdi
mov1 %eax, %esi
mov1 %edx, %edx
callq printf
#8位。操作寄存器。
movb $-16, %al
                  # 被除数, al
movsb1 %a1, %eax
                  # 符号扩展, 32 位
movb $5, %c1
                  # 除数, c1
                  \# al / cl = al , ah
idivb %cl
movb %ah, %bl
                  # 余数
movsb1 %a1, %eax
                  # 商
movsb1 %b1, %ebx
movq $str int32, %rdi
mov1 %eax, %esi
mov1 %ebx, %edx
callq printf
addq $64, %rsp
popq %rbp
retq
```

gcc div.s -o div

运行代码:

[root@local int]# ./div int64 shang = -5 yushu = 2 int64 shang = 5 yushu = 2 int32 shang = -4 yushu = 1 int32 shang = -5 yushu = -2 int32 shang = -2 yushu = 3 int32 shang = -3 yushu = -1

分析结果:

汇编代码		结果和分析	
# 64 位。操作寄存器。有符号除法		int64 shang = -5 yushu = 2	
movq \$17, %rax # Ř	被除数,rax		
cqto # 彳	符号扩展 rax > rdx:rax	有符号除法,使用 idivq。	
movq \$-3, %r8 # \$	除数,r8	符号扩展使用 cqto, 表示 rax >	
idivq %r8 # r	rax / r8 = rax, rdx	rdx:rax 。	
movq \$str_int64, %rdi		除数在寄存器 r8。	
movq %rax, %rsi # #	商	17 / (-3) = -5 , 2	
movq %rdx, %rdx # 5	余数	商在 rax,余数在 rdx。	
# 64 位。操作寄存器。无	符号除法	int64 shang = 5 yushu = 2	
movq \$17, %rax # Ř	被除数,rax		
movq \$0, %rdx # \$	零扩展, rdx, 清零	无符号除法,使用 divq。	
movq \$3, %r8 # \$	除数,r8	零扩展使用 movq \$0, %rdx 。	
divq %r8 # r	rax / r8 = rax, rdx	除数在寄存器 r8。	
movq \$str_int64, %rdi		17 / 3 = 5 , 2	
movq %rax, %rsi # #	商	商在 rax,余数在 rdx。	
movq %rdx, %rdx # 5	余数		
# 32 位。操作寄存器。		int32 shang = -4 yushu = 1	
mov1 \$9, %eax # Ř	被除数,eax		
cltd # 7	符号扩展 eax > edx:eax	有符号除法,使用 idivl。	
mov1 \$-2, %r8d # \$	除数,r8d	符号扩展使用 cltd, 表示 eax >	
idivl %r8d # e	eax / r8d = eax, edx	edx:eax 。	
movq \$str_int32, %rdi		除数在寄存器 r8d。	
mov1 %eax, %esi # #	商	9 / (-2) = -4 , 1	
mov1 %edx, %edx # 5	余数	商在 eax,余数在 edx。	
# 32 位。操作栈。		int32 shang = -5 yushu = -2	

movl \$-17, %eax # 被除数, eax cltd # 符号扩展 eax > edx:eax 有符号除法,使用 idivl。 mov1 \$3, -4(%rbp) # 除数, -4(%rbp) 符号扩展使用 cltd, 表示 eax > idivl -4(%rbp) # eax / (-4(%rbp)) = eax, edxedx:eax . movg \$str int32, %rdi 除数在栈内存-4(%rbp)。 -17 / 3 = -5 , -2mov1 %eax, %esi # 商 # 余数 mov1 %edx, %edx 商在 eax, 余数在 edx。 # 16 位。操作寄存器。 shang = -2 yushu = 3int32 mov1 \$0xFFFFFFFF, %eax # 占位, eax movw \$13, %ax #被除数,ax 有符号除法,使用 idivw。 符号扩展使用 cltd, 表示 eax > movsw1 %ax, %eax # 符号扩展, 32 位 # 符号扩展 eax > edx:eax edx:eax . cltd movw \$-5, %r8w # 除数, r8w 除数在寄存器 r8w。 idivw %r8w # eax / r8d = eax, edx 13 / (-5) = -2 , 3 movswl %ax. %eax # 商 商在ax, 余数在dx。 movswl %dx, %edx # 余数 #8位。操作寄存器。 int32 shang = -3yushu = -1movb \$-16, %al #被除数,al movsb1 %a1, %eax # 符号扩展, 32 位 有符号除法,使用 idivb。 符号扩展 movsbl %al, %eax , 8位扩展 movb \$5, %c1 # 除数, cl idivb %cl # al / cl = al , ah 到 32 位。 movb %ah, %bl # 余数 -16 / 5 = -3 , -1商在 al, 余数在 ah。 movsbl %al, %eax # 商

位数	被除数	商	余数
64 位	rax	rax	rdx
32 位	eax	eax	edx
16 位	ax	ax	dx
8位	al	al	ah

8位除法特殊,结果不使用寄存器 rdx。商和余数,都在寄存器 rax。商使用 al,余数使用 ah。

复杂的除法

编写代码: div hard.s

.data

```
str int32 :
   .string "int32 shang = %#X yushu = %#X \n"
. text
.global main
main:
   pushq %rbp
   movq %rsp, %rbp
   subq $64, %rsp
   # 32 位。操作寄存器。有符号除法
   mov1 $0x80556677, %eax # 被除数, eax。负数
                        # 符号扩展 eax > edx:eax
   cltd
   mov1 $1, %r8d
                        # 除数, r8d
   sh11 $12, %r8d
                       # 左移 12 位
   idivl %r8d
                        \# eax / r8d = eax, edx
   movq $str int32, %rdi
   mov1 %eax, %esi
                        # 商
                    # 余数
   mov1 %edx, %edx
   callq printf
   # 32 位。操作寄存器。无符号除法
   mov1 $0x80556677, %eax # 被除数, eax。
   mov1 $0, %edx
                        # 零扩展。edx 置 0
   mov1 $1, %r8d
                        # 除数, r8d
                        # 移动 12 位
   sh11 $12, %r8d
   div1 %r8d
                        \# eax / r8d = eax, edx
   movq $str int32, %rdi
   mov1 %eax, %esi
                         # 商
   mov1 %edx, %edx
                        # 余数
   callq printf
   addq $64, %rsp
   popq %rbp
   retq
```

编译代码:

gcc div hard.s -o div hard

运行代码:

[root@local int]# ./div hard

int32 shang = 0XFFF80557 yushu = 0XFFFFF677

int32 shang = 0X80556 yushu = 0X677

分析结果:

汇编代码		结果和分析		
# 32 位。操作寄存器。有	符号除法	int32 shang = OXFFF80557 yushu =		
mov1 \$0x80556677, %eax # 被除数, eax。负		0XFFFFF677		
数				
cltd	# 符号扩展 eax >	有符号除法,使用 idivl。		
edx:eax		符号扩展使用 cltd,表示 eax > edx:eax 。		
mov1 \$1, %r8d	# 除数, r8d	0x80556677 / $(1<<12)$ = $0XFFF80557$,		
shll \$12, %r8d	# 左移 12 位	0XFFFFF677		
idivl %r8d	# eax $/$ r8d = eax,	商为 0XFFF80557。		
edx		余数为 OXFFFFF677。		
movq \$str_int32, %rdi				
mov1 %eax, %esi	# 商			
movl %edx, %edx	# 余数			
# 32 位。操作寄存器。无符号除法		int32 shang = 0X80556 yushu = 0X677		
mov1 \$0x80556677, %eax	#被除数,eax。			
mov1 \$0, %edx	# 零扩展。edx 置 0	无符号除法,使用 divl。		
mov1 \$1, %r8d	# 除数, r8d	零扩展使用 mov1 \$0, %edx 。		
sh11 \$12, %r8d	# 移动 12 位	0x80556677 / (1 << 12) = 0X80556, $0X677$		
div1 %r8d	# eax $/$ r8d = eax,	商为 0X80556。		
edx		余数为 0X677。		
movq \$str_int32, %rdi				
mov1 %eax, %esi	# 商			
mov1 %edx, %edx	# 余数			

使用 16 进制与移位指令,方便查看结果。

单个 16 进制数字,表示 4个二进制数字。二进制数字左移 12位,表示 16 进制数字左移 3位。

相同的除法语句 0x80556677 / (1<<12) ,使用有符号除法指令、无符号除法指令,结果不相同。 idivl 指令的结果为 int32 shang = 0XFFF80557 yushu = 0XFFFF677 。

divl 指令的结果为 int32 shang = 0X80556 yushu = 0X677 。

32 位数字 0x80556677, 最高位是 1。如果表示有符号整数,则是负数。

有符号除法指令,使得商和余数的高位符号扩展,包含多个F。

无符号除法指令,使得商和余数的高位是0。