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 divl、64 位 divq。b、w、l、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 idivl、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

```
.data

str_int32 :
    .string "int32   shang = %d   yushu = %d \n"

str_int64 :
    .string "int64   shang = %lld   yushu = %lld \n"
```

```
.text
.global main

main :
    pushq %rbp
    movq %rsp, %rbp
    subq $64, %rsp

    # 64 位。操作寄存器。有符号除法
    movq $17, %rax      # 被除数，rax
    cqto                # 符号扩展 rax > rdx:rax
    movq $-3, %r8       # 除数，r8
    idivq %r8           # rax / r8 = rax, rdx
    movq $str_int64, %rdi
    movq %rax, %rsi     # 商
    movq %rdx, %rdx     # 余数
    callq printf

    # 64 位。操作寄存器。无符号除法
    movq $17, %rax      # 被除数，rax
    movq $0, %rdx       # 零扩展，rdx，清零
    movq $3, %r8        # 除数，r8
    divq %r8            # rax / r8 = rax, rdx
    movq $str_int64, %rdi
    movq %rax, %rsi     # 商
    movq %rdx, %rdx     # 余数
    callq printf

    # 32 位。操作寄存器。
    movl $9, %eax       # 被除数，eax
    cltd                # 符号扩展 eax > edx:eax
    movl $-2, %r8d      # 除数，r8d
    idivl %r8d          # eax / r8d = eax, edx
    movq $str_int32, %rdi
    movl %eax, %esi     # 商
    movl %edx, %edx     # 余数
    callq printf

    # 32 位。操作栈。
    movl $-17, %eax     # 被除数，eax
```

```
    cltd                # 符号扩展 eax > edx:eax
    movl $3, -4(%rbp)   # 除数，-4(%rbp)
    idivl -4(%rbp)      # eax / (-4(%rbp)) = eax, edx
    movq $str_int32, %rdi
    movl %eax, %esi     # 商
    movl %edx, %edx     # 余数
    callq printf

    # 16 位。操作寄存器。
    movl $0xFFFFFFFF, %eax  # 占位，eax
    movw $13, %ax           # 被除数，ax
    movswl %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
    movl %eax, %esi
    movl %edx, %edx
    callq printf

    # 8 位。操作寄存器。
    movb $-16, %al      # 被除数，al
    movsbl %al, %eax    # 符号扩展，32 位
    movb $5, %cl        # 除数，cl
    idivb %cl           # al / cl = al , ah
    movb %ah, %bl       # 余数
    movsbl %al, %eax    # 商
    movsbl %bl, %ebx
    movq $str_int32, %rdi
    movl %eax, %esi
    movl %ebx, %edx
    callq printf

    addq $64, %rsp
    popq %rbp
    retq
```

编译代码：

```
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 	
movq $17, %rax # 被除数，rax
cqto # 符号扩展 rax > rdx:rax
movq $-3, %r8 # 除数，r8
idivq %r8 # rax / r8 = rax, rdx
movq $str_int64, %rdi
movq %rax, %rsi # 商
movq %rdx, %rdx # 余数
	int64 shang = -5 yushu = 2
	
	有符号除法，使用 idivq。
	符号扩展使用 cqto，表示 rax > rdx:rax 。
	除数在寄存器 r8。
	17 / (-3) = -5 , 2
	商在 rax，余数在 rdx。

# 64 位。操作寄存器。无符号除法 	
movq $17, %rax # 被除数，rax
movq $0, %rdx # 零扩展，rdx，清零
movq $3, %r8 # 除数，r8
divq %r8 # rax / r8 = rax, rdx
movq $str_int64, %rdi
movq %rax, %rsi # 商
movq %rdx, %rdx # 余数
	int64 shang = 5 yushu = 2
	
	无符号除法，使用 divq。
	零扩展使用 movq $0, %rdx 。
	除数在寄存器 r8。
	17 / 3 = 5 , 2
	商在 rax，余数在 rdx。

# 32 位。操作寄存器。 	
movl $9, %eax # 被除数，eax
cltd # 符号扩展 eax > edx:eax
movl $-2, %r8d # 除数，r8d
idivl %r8d # eax / r8d = eax, edx
movq $str_int32, %rdi
movl %eax, %esi # 商
movl %edx, %edx # 余数
	int32 shang = -4 yushu = 1
	
	有符号除法，使用 idivl。
	符号扩展使用 cltd，表示 eax > edx:eax 。
	除数在寄存器 r8d。
	9 / (-2) = -4 , 1
	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 movl $3, -4(%rbp)　　# 除数，-4(%rbp) idivl -4(%rbp)　　# eax / (-4(%rbp)) = eax, edx movq $str_int32, %rdi movl %eax, %esi　　# 商 movl %edx, %edx　　# 余数	有符号除法，使用 idivl。 符号扩展使用 cltd，表示 eax > edx:eax 。 除数在栈内存-4(%rbp)。 -17 / 3 = -5 , -2 商在 eax，余数在 edx 。
# 16 位。操作寄存器。 movl $0xFFFFFFFF, %eax　# 占位，eax movw $13, %ax　　# 被除数，ax movswl %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	int32　shang = -2　yushu = 3 有符号除法，使用 idivw。 符号扩展使用 cltd，表示 eax > edx:eax 。 除数在寄存器 r8w。 13 / (-5) = -2 , 3 商在 ax，余数在 dx。
# 8 位。操作寄存器。 movb $-16, %al　　# 被除数，al movsbl %al, %eax　　# 符号扩展，32 位 movb $5, %cl　　# 除数，cl idivb %cl　　# al / cl = al , ah movb %ah, %bl　　# 余数 movsbl %al, %eax　　# 商	int32　shang = -3　yushu = -1 有符号除法，使用 idivb。 符号扩展 movsbl %al, %eax ，8 位扩展到 32 位。 -16 / 5 = -3 , -1 商在 al，余数在 ah。

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
    movl $0x80556677, %eax  # 被除数，eax。负数
    cltd                    # 符号扩展 eax > edx:eax
    movl $1, %r8d           # 除数，r8d
    shll $12, %r8d          # 左移 12 位
    idivl %r8d              # eax / r8d = eax, edx
    movq $str_int32, %rdi
    movl %eax, %esi         # 商
    movl %edx, %edx         # 余数
    callq printf

    # 32 位。操作寄存器。无符号除法
    movl $0x80556677, %eax  # 被除数，eax。
    movl $0, %edx           # 零扩展。edx 置 0
    movl $1, %r8d           # 除数，r8d
    shll $12, %r8d          # 移动 12 位
    divl %r8d               # eax / r8d = eax, edx
    movq $str_int32, %rdi
    movl %eax, %esi         # 商
    movl %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 	
movl $0x80556677, %eax # 被除数，eax。负数
cltd   # 符号扩展 eax > edx:eax
movl $1, %r8d   # 除数，r8d
shll $12, %r8d   # 左移 12 位
idivl %r8d   # eax / r8d = eax, edx
movq $str_int32, %rdi
movl %eax, %esi   # 商
movl %edx, %edx   # 余数
	int32 shang = 0XFFF80557 yushu = 0XFFFFF677
	
	有符号除法，使用 idivl。
	符号扩展使用 cltd，表示 eax > edx:eax 。
	0x80556677 / (1<<12) = 0XFFF80557 , 0XFFFFF677
	商为 0XFFF80557。
	余数为 0XFFFFF677。

# 32 位。操作寄存器。无符号除法 	
movl $0x80556677, %eax # 被除数，eax。
movl $0, %edx   # 零扩展。edx 置 0
movl $1, %r8d   # 除数，r8d
shll $12, %r8d   # 移动 12 位
divl %r8d   # eax / r8d = eax, edx
movq $str_int32, %rdi
movl %eax, %esi   # 商
movl %edx, %edx   # 余数
	int32 shang = 0X80556 yushu = 0X677
	
	无符号除法，使用 divl。
	零扩展使用 movl $0, %edx 。
	0x80556677 / (1<<12) = 0X80556 , 0X677
	商为 0X80556。
	余数为 0X677。

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 = 0XFFFFF677 。
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 。