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

指令说明

imul 指令表示有符号整数乘法。
imul 指令分为 8 位 imulb、16 位 imulw、32 位 imull、64 位 imulq。
imul 指令可以操作立即数、寄存器、内存。

mul 指令表示无符号整数乘法。
mul 指令分为 8 位 mulb、16 位 mulw、32 位 mull、64 位 mulq。
mul 指令可以操作寄存器、内存。

语法格式 `imul cc` 表示 `rax * cc = rdx : rax` 。
语法格式 `imul ee, ff` 表示 `ee * ff = ff` 。
语法格式 `imul ee, ff, gg` 表示 `ee * ff = gg` 。
语法格式 `mul cc` 表示 `rax * cc = rdx : rax` 。

单操作数模式 `imul cc` 和 `mul cc`，结果使用 rdx 和 rax。
乘法操作导致位数膨胀。比如，16 位数字乘以 16 位数字，结果最大为 32 位数字。
把结果分为高位、低位。高位放在 rdx，低位放在 rax。

简单的乘法

编写代码： mul.s

```
.data

num_int32 :
    .long 0x0

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

str_int64 :
    .string " int64 = %lld \n"

.text
.global main

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

    # 64 位。操作寄存器
    movq $333333333333333, %r8
```

```
imulq $2 , %r8 ,  %r9       # r9 = 2 * r8
movq $str_int64 , %rdi
movq %r9, %rsi
callq printf

# 32 位。操作寄存器
movl $3, -4(%rbp)
imull $2 , -4(%rbp) , %r9d   # r9d = 2 * (-4(%rbp))
movq $str_int32 , %rdi
movl %r9d, %esi
callq printf

# 32 位。操作寄存器
movl $3, %eax
movl $5, %r8d
imull %r8d           # eax = eax * r8d
movq $str_int32 , %rdi
movl %eax, %esi
callq printf

# 32 位。操作栈
movl $3, %eax
movl $6, -4(%rbp)
imull -4(%rbp)        # eax = eax * (-4(%rbp))
movq $str_int32 , %rdi
movl %eax, %esi
callq printf

# 32 位。操作数据段
movl $3, %eax
movl $7, num_int32(%rip)
imull num_int32(%rip)   # eax = eax * (num_int32(%rip))
movq $str_int32, %rdi
movl %eax, %esi
callq printf

# 16 位。操作寄存器
movw $3, %ax
movw $8, %r8w
imulw %r8w           # ax = ax * r8w
movq $str_int32 , %rdi
movzwl %ax, %eax
movl %eax, %esi
callq printf

# 8 位。操作寄存器
movb $3, %al
movb $9, %r8b
imulb %r8b           # al = al * r8b
```

```
    movq $str_int32 , %rdi
    movzbl %al, %eax
    movl %eax, %esi
    callq printf

    # 32 位。操作寄存器。双操作数。
    movl $3, %ecx
    movl $-11, %edx
    imull %edx, %ecx    # ecx = ecx * edx
    movq $str_int32, %rdi
    movl %ecx, %esi
    callq printf

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

编译代码：

```
gcc mul.s -o mul
```

运行代码：

```
[root@local int]# ./mul
 int64 = 666666666666666
 int32 = 6
 int32 = 15
 int32 = 18
 int32 = 21
 int32 = 24
 int32 = 27
 int32 = -33
```

分析结果：

汇编代码	结果和分析
# 64 位。操作寄存器 movq $333333333333333, %r8 imulq $2 , %r8 , %r9 # r9 = 2 * r8	int64 = 666666666666666 把 64 位 333333333333333 写到寄存器 r8。 把 r8 乘以 2 写到 r9 。 三操作数模式，imul 有 3 个参数。 333333333333333 * 2 = 666666666666666
# 32 位。操作寄存器 movl $3, -4(%rbp) imull $2 , -4(%rbp) , %r9d # r9d = 2 * (-4(%rbp))	int32 = 6 把 32 位 3 写到栈内存-4(%rbp)。 再乘以 2，写到 r9d。 三操作数模式，imul 有 3 个参数。 3 * 2 = 6

# 32 位。操作寄存器 	
movl $3, %eax
movl $5, %r8d
imull %r8d   # eax = eax * r8d
	int32 = 15
	
	把 32 位 3 写到寄存器 eax。
	把 32 位 5 写到寄存器 r8d。

	单操作数模式，imul 有 1 个参数。 3 * 5 = 15

# 32 位。操作栈 	
movl $3, %eax
movl $6, -4(%rbp)
imull -4(%rbp) # eax = eax * (-4(%rbp))
	int32 = 18
	
	把 32 位 3 写到寄存器 eax。
	把 32 位 6 写到栈内存-4(%rbp)。
	单操作数模式，imul 有 1 个参数。
	3 * 6 = 18

# 32 位。操作数据段 	
movl $3, %eax
movl $7, num_int32(%rip)
imull num_int32(%rip) # eax = eax * (num_int32(%rip))
	int32 = 21
	
	把 32 位 3 写到寄存器 eax。
	把 32 位 7 写到变量 num_int32。
	单操作数模式，imul 有 1 个参数。
	3 * 7 = 21

# 16 位。操作寄存器 	
movw $3, %ax
movw $8, %r8w
imulw %r8w # ax = ax * r8w
	int32 = 24
	
	把 16 位 3 写到寄存器 ax。
	把 16 位 8 写到变量 r8w。
	单操作数模式，imul 有 1 个参数。
	3 * 8 = 24

# 8 位。操作寄存器 	
movb $3, %al
movb $9, %r8b
imulb %r8b # al = al * r8b
	int32 = 27
	
	把 8 位 3 写到寄存器 al。
	把 8 位 9 写到变量 r8b。
	单操作数模式，imul 有 1 个参数。
	3 * 9 = 27

# 32 位。操作寄存器。双操作数。 	
movl $3, %ecx
movl $-11, %edx
imull %edx, %ecx # ecx = ecx * edx
movq $str_int32, %rdi
movl %ecx, %esi
callq printf
	int32 = -33
	
	把 32 位 3 写到寄存器 ecx。
	把 32 位-11 写到寄存器 edx。
	双操作数模式，imul 有 2 个参数。
	3 * -11 = -33

复杂的乘法

编写代码： mul_hard.s

```
.data

str_r9 :
    .string "int64  r9 = %#llX  rdx = %#X \n"

str_int32 :
    .string "int32  edx = %#X  eax = %#X \n"

str_int64 :
```

```
    .string "int64  rdx = %#llX  rax = %#llX \n"

.text
.global main

main :
    pushq %rbp
    movq %rsp, %rbp

    # 64 位，三操作数，溢出
    movq $0x11111111, %rdx          # 写值，64 位，rdx 占位
    movq $0x3333333322222222, %r9   # 写值，64 位，r9
    movq $0x55667788AABBCCDD, %r8   # 写值，64 位，r8
    imulq $16, %r8, %r9             # r9 = 16 * r8
    movq $str_r9, %rdi
    movq %r9, %rsi
    movq %rdx, %rdx
    callq printf

    # 32 位，三操作数，溢出
    movq $0x11111111, %rdx          # 写值，64 位，rdx 占位
    movq $0x3333333322222222, %r9   # 写值，64 位，r9
    movl $0xAABBCCDD, %r8d          # 写值，32 位，r8d
    imull $16 , %r8d , %r9d         # r9d = 16 * r8d 。高位清零
    movq $str_r9, %rdi
    movq %r9 , %rsi
    movq %rdx, %rdx
    callq printf

    # 32 位，单操作数，溢出
    movl $0x22222222, %edx      # 写值，32 位，edx 占位
    movl $0x55667788, %eax      # 写值，32 位，eax
    movl $1, %r8d
    shll $12, %r8d              # 16 进制，4 位一组
    imull %r8d                  # 结果在 edx eax
    movq $str_int32, %rdi
    movl %edx, %esi
    movl %eax, %edx
    callq printf

    # 32 位，单操作数，有符号乘法，负数，溢出
    movl $0x22222222, %edx     # 写值，32 位，edx 占位
    movl $0xF6E6D6C6, %eax     # 写值，32 位，eax，负数
    movl $1, %ecx
    shll $12, %ecx             # 16 进制，4 位一组
    imull %ecx                 # 结果在 edx eax
    movq $str_int32, %rdi
    movl %edx, %esi
    movl %eax, %edx
```

```
    callq printf

    # 32 位，单操作数，无符号乘法，溢出
    movl $0x22222222, %edx      # 写值，32 位，edx 占位
    movl $0xF6E6D6C6, %eax      # 写值，32 位，eax，负数
    movl $1, %ecx
    shll $12, %ecx              # 16 进制，4 位一组
    mull %ecx                   # 结果在 edx eax
    movq $str_int32, %rdi
    movl %edx, %esi
    movl %eax, %edx
    callq printf

    # 64 位，单操作数，溢出
    movq $0x2222222222222222, %rdx  # 写值，64 位，rdx 占位
    movq $0x33445566778899AA, %rax  # 写值，64 位，rax
    movq $1, %r8
    shlq $12, %r8           # 16 进制，4 位一组
    imulq %r8               # 结果在 rdx rax
    movq $str_int64, %rdi
    movq %rdx, %rsi
    movq %rax, %rdx
    callq printf

    popq %rbp
    retq
```

编译代码：

```
gcc mul_hard.s -o mul_hard
```

运行代码：

```
[root@local int]# ./mul_hard
int64  r9 = 0X5667788AABBCCDD0  rdx = 0X11111111
int64  r9 = 0XABBCCDD0  rdx = 0X11111111
int32  edx = 0X556  eax = 0X67788000
int32  edx = 0XFFFFFF6E  eax = 0X6D6C6000
int32  edx = 0XF6E  eax = 0X6D6C6000
int64  rdx = 0X334  rax = 0X45566778899AA000
```

分析结果：

汇编代码	结果和分析

# 64 位，三操作数，溢出 	
movq $0x11111111, %rdx # 写值，64 位，rdx 占位
movq $0x3333333322222222, %r9 # 写值，64 位，r9
movq $0x55667788AABBCCDD, %r8 # 写值，64 位，r8
imulq $16, %r8, %r9 # r9 = 16 * r8
	int64 r9 = 0X5667788AABBCCDD0 rdx = 0X11111111
	
	把 0x11111111 写到 rdx，占位。
	把 0x3333333322222222 写到 r9。
	乘法 16 * 0x55667788AABBCCDD ，结果写到 r9。
	r9 的值为 0X5667788AABBCCDD0 。
	rdx 的值没有变化。

# 32 位，三操作数，溢出	int64 r9 = 0XABBCCDD0 rdx = 0X11111111

movq $0x11111111, %rdx　　# 写值，64 位，rdx 占位 movq $0x3333333322222222, %r9　# 写值，64 位，r9 movl $0xAABBCCDD, %r8d　　# 写值，32 位，r8d imull $16 , %r8d , %r9d　　# r9d = 16 * r8d 。高位清零	把 0x11111111 写到 rdx，占位。 把 0x3333333322222222 写到 r9。 乘法 16 * 0xAABBCCDD，结果写到 r9d。 r9 的值为 0XABBCCDD0，并且高位清零。 rdx 的值没有变化。
# 32 位，单操作数，溢出 movl $0x22222222, %edx　　# 写值，32 位，edx 占位 movl $0x55667788, %eax　　# 写值，32 位，eax movl $1, %r8d shll $12, %r8d　　# 16 进制，4 位一组 imull %r8d　　# 结果在 edx eax	int32　edx = 0X556　eax = 0X67788000 把 0x22222222 写到 edx，占位。 把 0x55667788 写到 eax。 乘法 0x55667788 *(1<<12) 。 edx 的值为 0X556，表示结果的高位。 eax 的值为 0X67788000，表示结果的低位。
# 32 位，单操作数，有符号乘法，负数，溢出 movl $0x22222222, %edx　　# 写值，32 位，edx 占位 movl $0xF6E6D6C6, %eax　　# 写值，32 位，eax，负数 movl $1, %ecx shll $12, %ecx　　# 16 进制，4 位一组 imull %ecx　　# 结果在 edx eax	int32　edx = 0XFFFFFF6E　eax = 0X6D6C6000 把 0x22222222 写到 edx，占位。 把 0xF6E6D6C6 写到 eax。 imull 有符号乘法 0xF6E6D6C6 *(1<<12) 。 edx 的值为 0XFFFFFF6E，表示结果的高位。 eax 的值为 0X6D6C6000，表示结果的低位。 结果是负数，edx 的高位有许多 F。

# 32 位，单操作数，无符号乘法，溢出 	
movl $0x22222222, %edx　　# 写值，32 位，edx 占位
movl $0xF6E6D6C6, %eax　　# 写值，32 位，eax，负数
movl $1, %ecx
shll $12, %ecx　　# 16 进制，4 位一组
mull %ecx　　# 结果在 edx eax
movq $str_int32, %rdi
movl %edx, %esi
movl %eax, %edx
callq printf
	int32　edx = 0XF6E　eax = 0X6D6C6000
	
	把 0x22222222 写到 edx，占位。
	把 0xF6E6D6C6 写到 eax。
	mull 无符号乘法 0xF6E6D6C6 *(1<<12) 。
	edx 的值为 0XF6E，表示结果的高位。
	eax 的值为 0X6D6C6000，表示结果的低位。
	因为使用无符号乘法，edx 的高位没有多余的 F。

# 64 位，单操作数，溢出 movq $0x2222222222222222, %rdx　# 写值，64 位，rdx 占位 movq $0x33445566778899AA, %rax　# 写值，64 位，rax movq $1, %r8 shlq $12, %r8　　# 16 进制，4 位一组 imulq %r8　　# 结果在 rdx rax	int64　rdx = 0X334　rax = 0X45566778899AA000 把 0x2222222222222222 写到 rdx，占位。 把 0x33445566778899AA 写到 rax。 乘法 0x33445566778899AA *(1<<12) 。 rdx 的值为 0X334　，表示结果的高位。 rax 的值为 0X45566778899AA000，表示结果的低位。

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。

相同的乘法语句 0xF6E6D6C6 *(1<<12) ，使用有符号乘法指令、无符号乘法指令，结果不相同。
imull 指令的结果为 int32　edx = 0XFFFFFF6E　eax = 0X6D6C6000 。
mull 指令的结果为 int32　edx = 0XF6E　eax = 0X6D6C6000 。
有符号乘法指令，使得 edx 的高位符号扩展，包含多个 F。

64 位乘法 0x33445566778899AA *(1<<12) ，结果为 0x33445566778899AA000 。
以 64 位为单位，拆分 0x33445566778899AA000 ，写到 rdx、rax。

0x334 写到 rdx，表示结果的高位。
0x45566778899AA000 写到 rax，表示结果的低位。