x86-64的基本指令

南京大学
计算机科学与技术系
袁春风
email: cfyuan@nju.edu.cn
2015.6

传送指令

- 数据传送指令（助记符“q”表示操作数长度为四字（即64位））

  movabsq I, R：将64位立即数送64位通用寄存器

  movq：传送一个64位的四字

  movsbq、movswq、movslq：将源操作数进行符号扩展并传送到一个64位寄存器或存储单元中

  movzbq、movzwq：将源操作数进行零扩展后传送到一个64位寄存器或存储单元中

  movl：的功能相当于movzlq指令

  pushq S：R[rsp]←R[rsp]-8; M[R[rsp]] ←S

  popq D： D← M[R[rsp]]; R[rsp]←R[rsp]-8

传送指令

- **数据传送指令举例**

以下函数功能是将类型为source_type的参数转换为dest_type型数据并返回

```
dest_type  convert(source_type x) {
        dest_type y = (dest_type) x;
        return y;
}
```

根据参数传递约定知，x在RDI对应的适合宽度的寄存器（RDI、EDI、DI和DIL）中，y存放在RAX对应的寄存器（RAX、EAX、AX或AL）中，填写下表中的汇编指令以实现convert函数中的赋值语句

source_type	dest_type	汇 编 指 令
char	long	问题：每种情况对应的汇编指令各是什么？
int	long	
long	long	
long	int	
unsigned int	unsigned long	
unsigned long	unsigned int	
unsigned char	unsigned long	

传送指令

- **数据传送指令举例**

以下函数功能是将类型为source_type的参数转换为dest_type型数据并返回

```
dest_type  convert(source_type x) {
        dest_type y = (dest_type) x;
        return y;
}
```

根据参数传递约定知，x在RDI对应的适合宽度的寄存器（RDI、EDI、DI和DIL）中，y存放在RAX对应的寄存器（RAX、EAX、AX或AL）中，填写下表中的汇编指令以实现convert函数中的赋值语句

source_type	dest_type	汇 编 指 令
char	long	movsbq %dil, %rax
int	long	movslq %edi, %rax
long	long	movq %rdi, %rax
long	int	movslq %edi, %rax //符号扩展到 64 位 movl %edi, %eax //只需x的低32位
unsigned int	unsigned long	movl %edi, %eax //零扩展到 64 位
unsigned long	unsigned int	movl %edi, %eax //零扩展到 64 位
unsigned char	unsigned long	movzbq %dil, %rax //零扩展到 64 位

算术逻辑指令

- **常规的算术逻辑运算指令**

  只要将原来IA-32中的指令扩展到64位即可。例如：

  - **addq（四字相加）**
  - **subq（四字相减）**
  - **incq（四字加1）**
  - **decq（四字减1）**
  - **imulq（带符号整数四字相乘）**
  - **orq（64位相或）**
  - **salq（64位算术左移）**
  - **leaq（有效地址加载到64位寄存器）**

算术逻辑指令

以下是C赋值语句“x=a*b+c*d;”对应的x86-64汇编代码

已知x、a、b、c和d分别在寄存器RAX(x)、RDI(a)、RSI(b)、RDX(c)和RCX(d)对应宽度的寄存器中

根据以下汇编代码，推测x、a、b、c和d的数据类型

```
movslq %ecx, %rcx
imulq  %rdx, %rcx
movsbl %sil, %esi
imull %edi, %esi
movslq %esi, %rsi
leaq (%rcx, %rsi), %rax
```

d从32位符号扩展为64位，故d为int型

在RDX中的c为64位long型

在SIL中的b为char型

在EDI中的a是int型

在RAX中的x是long型

算术逻辑指令

- **特殊的算术逻辑运算指令**

  对于x86-64，还有一些特殊的算术逻辑运算指令。例如：

  imulq S：R[rdx]:R[rax]← S * R[rax] (64位*64位带符号整数)

  mulq S：R[rdx]:R[rax]← S * R[rax] (64位*64位无符号整数)

  cltq：R[rax] ← SignExtend(R[eax]) (将EAX内容符号扩展为四字)

  clto： R[rdx]:R[rax]← SignExtend(R[rax]) (符号扩展为八字)

  idivq S：R[rdx] ← R[rdx]:R[rax] mod S (带符号整数相除、余数)

  R[rax] ← R[rdx]:R[rax]÷S (带符号整数相除、商)

  divq S：R[rdx] ← R[rdx]:R[rax] mod S (无符号整数相除、余数)

  R[rax] ← R[rdx]:R[rax]÷S (无符号整数相除、商)

  上述功能描述中，R[rdx]:R[rax]是一个128位的八字（oct word）

算术逻辑指令

不同长度操作数混合运算时，编译器必须选择正确的指令的组合。

```
long samp(int x, int y)
{
    long t1=(long) x+y;
    long t2=(long) (x+y);
    return t1 | t2;
}
```

计算t1：先符号扩展为64位，再进行64位加法

计算t2：先进行32位加法，再符号扩展为64位

对应x86-64汇编代码如下（x在EDI中，y在ESI中）：

```
leal  (%rdi,%rsi), %eax
cltq
movslq %esi, %rsi
movslq %edi, %rdi
addq %rsi, %rdi
orq  %rdi, %rax
```

EDI和ESI中内容相加，低32位送EAX

将EAX符号扩展为四字，送RAX（t2）

将ESI符号扩展为四字，送RSI

将EDI符号扩展为四字，送RDI

RDI和RSI中内容相加，送RDI（t1）

RAX和RDI中内容相或，送RAX

比较和测试指令

- **比较和测试指令**

  与IA-32中比较和测试指令类似。例如：

  cmpq S2, S1 : S1-S2 (64位数相减进行比较)

  testq S2, S1 : S1∧S2 (64位数相与进行比较)

  条件转移指令、条件传送指令、条件设置指令都根据上述比较指令和测试指令生成的标志进行处理

x86-64逆向工程举例

根据汇编代码填写C语句，说明功能

```
long test(unsigned long x)
{
    long val=0;
    int i;
    for ( ①  ; ②  ; ③ ){
        ④ ;
    }
    ⑤ ;
    return ⑥ ;
}
```

①处是: i=0

```
movl    $0, %ecx        R[ecx]=val
movl    $0, %edx        R[edx]=i
movabsq $7234017283807673, %rsi
.L1                     R[rsi]=0x0101010101010101
movq    %rdi, %rax
andq    %rsi, %rax
addq    %rax, %rcx
shrq    %rdi
addl    $1, %edx
cmpl    $8, %edx
jne     .L1
movq    %rcx, %rax
sarq    $32, %rax
addq    %rcx, %rax
movq    %rax, %rdx
sarq    $16, %rdx
addq    %rax, %rdx
movq    %rdx, %rax
sarq    $8, %rax
addq    %rdx, %rax
andl    $255, %eax
ret
```

④处是:
val+=x&0x01...01;
x>>=1;

③处是: i++

②处是: i<8

⑤处是以下语句：
val+=(val>>32);
高、低32位相加
val+=(val>>16);
高、低16位相加
val+=(val>>8);
高、低8位相加

⑥处是: val&0xFF;

GCC生成的x86-64汇编代码如右

$2^{56}+2^{48}+2^{40}+2^{32}+2^{24}+2^{16}+2^{8}+2^{0}$
=7234017283807673

for循环：val各字节记录x中对应字节内1的个数

函数功能：计算x中1的个数。

因最大值为64，故最终析取低8位