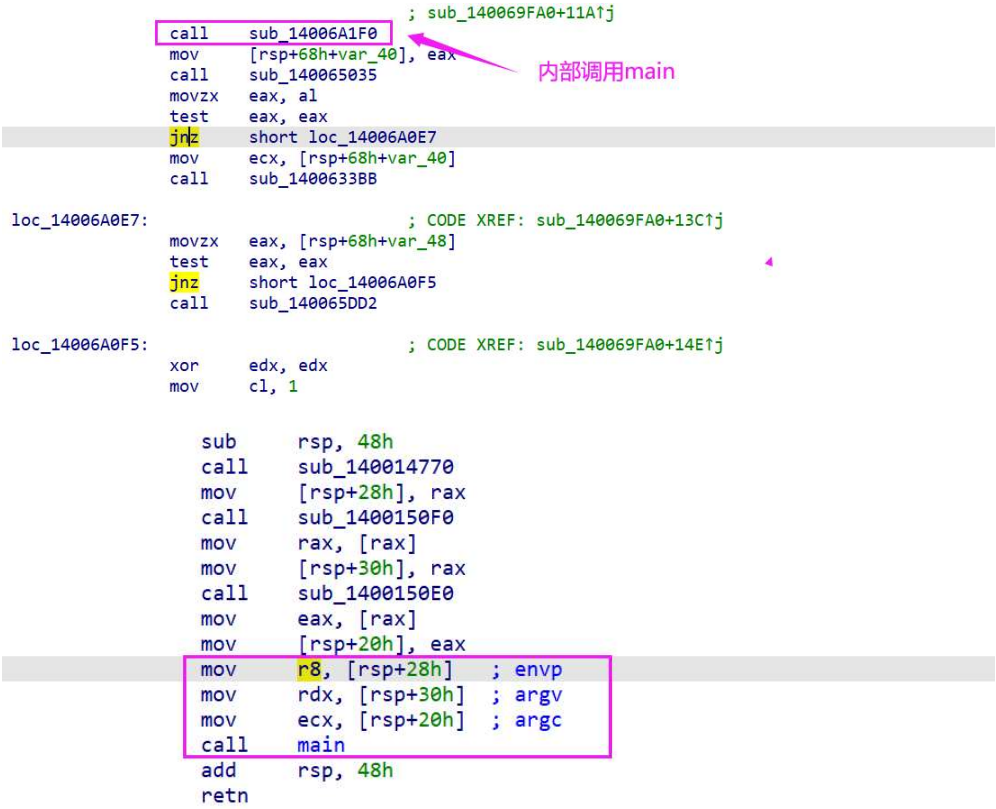
2021/05/25_x64汇编与逆向_第2课_表达式(加减乘除等)

笔记本： x64汇编与逆向
创建时间： 2021/5/25 星期二 15:06
作者： ileemi

定位入口

> 静态库编译的程序且完全没有符号的情况定位入口：

- Debug：两个相邻call，两个判断。
- Release：main之前有三个mov传参（R8、RDX、RCX）。



所有平台识别main入口的特征（x32、x64）：

- main函数执行完会调用exit退出进程，离exit（exit可使用sig文件进行识别）最近的call可能就是main（其它平台一般是exit函数上一个call，高版本的VS比较特殊，在exit上面在第二个call）。
- 当程序种也使用调用了exit函数，快速定位方法：从start开始进去，定位第一个exit，之后进行call辨别。注意：main函数有三个参数，WinMain有四个参数，都使用寄存器传参。

```
mov     [rsp+28h], rax
mov     r9d, [rsp+20h]
mov     r8, [rsp+28h]   ; envp
xor     edx, edx        ; argv
lea     rcx, cs:140000000h ; argc
call    j_main
add     rsp, 38h
retn
```

表达式的识别

加减法

> Release版优化：可能会使用 lea 进行比例因子寻址代替 add、sub，和x86基本一样，减法可能直接减法或者优化成加补码。

```
push    rbx
sub     rsp, 20h
mov     ebx, ecx
lea     edx, [rcx+3]
lea     rcx, Format     ; "n1 = %d\n"
call    printf
mov     edx, 3
lea     rcx, Format     ; "n1 = %d\n"
sub     edx, ebx
call    printf
```

乘法

> 在x86上优化成比例因子寻址或者左移，x64会直接用 imul 指令，因为64位的乘法指令延迟（latency）较小，指令执行速度较快。

```
40 53                   push    rbx
48 83 EC 20             sub     rsp, 20h
6B D1 0F                imul    edx, ecx, 15
8B D9                   mov     ebx, ecx
48 8D 0D BE 11+         lea     rcx, Format     ; "n1 * 15 = %d\n"
00 00
E8 89 FF FF FF          call    printf
8B D3                   mov     edx, ebx
48 8D 0D C0 11+         lea     rcx, aN116D     ; "n1 * 16 = %d\n"
00 00
C1 E2 04                shl     edx, 4
E8 78 FF FF FF          call    printf
BA 04 00 00 00          mov     edx, 4
48 8D 0D BC 11+         lea     rcx, a22D       ; "2 * 2 = %d\n"
00 00
E8 67 FF FF FF          call    printf
8B D3                   mov     edx, ebx
48 8D 0D BE 11+         lea     rcx, aN245D     ; "n2 * 4 + 5 = %d\n"
00 00
C1 E2 05                shl     edx, 5
83 C2 05                add     edx, 5
E8 53 FF FF FF          call    printf
0F AF DB                imul    ebx, ebx
48 8D 0D C1 11+         lea     rcx, aN1N2D     ; "n1 * n2 = %d\n"
00 00
8B D3                   mov     edx, ebx
E8 42 FF FF FF          call    printf
```

除法

> 识别：特征（容易受到编译器、指令集的影响）、公式

除法还原公式：注意最后值的精度（一般小数点后9位都是9），精度不够，直接还原成公式。n为移位总次数，在x86下，指数为32。在x64下，指数为64。

正数还原除数：2^n / M
负数还原除数：2^n / (2^32 - M)
溢出还原除数：2^n / (2^32 + M)

1. 无符号除数为2的幂：x / 2^n = x >> n，直接右移

```
// 变量除以常量，常量为无符号2的幂
printf("argc / 16 = %u", (unsigned)argc / 16);
// 对应的汇编代码
shr ecx, 4
mov edx, ecx
lea rcx, Format ; "argc / 16 = %u"
call printf
```

```
shr     ecx, 4
mov     edx, ecx
lea     rcx, Format     ; "argc / 16 = %u"
call    printf
```

2. 无符号除数为非2的幂（情况1：没有溢出）：

- 32位除法：x / imm = x * M >> 32 >> n
- 64位除法：x / imm = x * M >> 64 >> n

```
// 变量除以常量，常量为无符号非2的幂
printf("argc / 3 = %u", (unsigned)argc / 3);
// 对应的汇编代码
movsxd rbx, ecx
mov eax, 2863311531
mul ebx ; edx.eax
lea rcx, Format ; "argc / 3 = %u"
shr edx, 1 ; >> 32+1
call printf

// 变量除以常量，常量为无符号非2的幂
printf("argc / 3 = %u", (unsigned long long)argc / 3);
// 对应的汇编代码
mov rax, 12297829382473034411
lea rcx, Format ; "argc / 3 = %u"
mul rbx ; rdx.rax
shr rdx, 1 ; >> 64+1
call printf
```

上述优化方式是vs编译器，对于clang编译器会将右移值进行合并，一次性右移。

3. 无符号除数为非2的幂（情况2：溢出）：**乘减移加移**
   x / imm = (((x - (x * M >> 32)) >> n1) + (x * M >> 32)) >> n2

```
//变量除以常量，常量为无符号非2的幂
printf("argc / 7 = %u", (unsigned long long)argc / 7);
// 对应的汇编代码
movsxd rbx, ecx
mov eax, 613566757
mul ebx
mov eax, ebx
lea rcx, Format ; "argc / 7 = %u"
sub eax, edx
shr eax, 1
add edx, eax
shr edx, 2
call printf

//变量除以常量，常量为无符号非2的幂
printf("argc / 7 = %u", (unsigned long long)argc / 7);
// 对应的汇编代码
mov rax, 2635249153387078803
lea rcx, Format ; "argc / 7 = %u"
mul rbx
mov r8, rbx
sub r8, rdx
shr r8, 1
add rdx, r8
shr rdx, 2
call printf
```

4. 有符号除数为2的幂：

- x >= 0：x / 2^n = x >> n
- x < 0：x / 2^n = (x + (2^n - 1)) >> n

代码示例：

```
//变量除以常量，常量为无符号非2的幂
printf("argc / -8 = %u", argc / -8);
// 对应的汇编代码
mov eax, ecx
lea rcx, Format ; "argc / -8 = %u"
cdq ; 判断正负，进行无分之求解
and edx, 7 ; 负:于7,负:于0
add edx, eax
```

```
sar edx, 3
neg edx
call printf

无分支除法：
mov edx, ecx
add edx, 7
cmp ecx, 0
cmovlt edx, ecx
sar edx, 3

mov edx, ecx
sar edx, 31 ; edx = 0/1
and edx, 7 ; edx = 0/7
add ecx, edx
sar ecx, 2
```

5. 有符号除数为负2的幂：
   x / -2^n = -(x / 2^n)

6. 有符号除数为正非2的幂

- x >= 0：x / imm = （x * M >> n） + 0
- x < 0：x / imm = （x * M >> n） + 1

```
// 对应的汇编代码
mov eax, 66666667h
mov ebx, ecx
imul ecx
lea rcx, Format ; "argc / 5 = %u"
sar edx, 1
mov eax, edx
shr eax, 1Fh
add edx, eax
call printf
```

正数还原除数：2^n / M

7. 有符号除数为正非2的幂（特殊情况：MagicNumber最高位为1）

- x >= 0：x / imm = ((x * M >> 32 + x)) >> n + 0
- x < 0：x / imm = ((x * M >> 32 + x)) >> n + 1

```
printf("argc / 7 = %u", argc / 7);
// 对应的汇编代码
mov eax, 92492493h
imul ecx
add edx, ecx
```

```
lea rcx, Format ; "argc / 7 = %u"
sar edx, 2
mov eax, edx
shr eax, 1Fh
add edx, eax
call printf
```

8. 有符号除数为负非2的幂

- x >= 0：x / imm = (x * M >> n) + 0
- x < 0：x / imm = (x * -M >> n) + 1

```
printf("argc / -5 = %u", argc / -5);
// 对应的汇编代码
mov eax, 99999999h
lea rcx, aArgc5U_0 ; "argc / -5 = %u"
imul ebx
sar edx, 1
mov eax, edx
shr eax, 1Fh
add edx, eax
call printf
```

负数还原除数：2^n / (2^32 - M)

9. 有符号除数为负非2的幂（特殊情况：MagicNumber最高位为0）
    - x >= 0：x / imm = ((x * M >> 32 - x)) >> n + 0
    - x < 0：x / imm = ((x * M >> 32 - x)) >> n + 1

取模

1. **模2的幂**
    - x >= 0
      x % 2^n = x & (2^n - 1)
    - x < 0
      32位（老公式--vs、gcc编译器）：x % 2^n = (x & (2^n - 1)) - 1 | (~(2^n - 1)) + 1
      64位（新公式-- vs、gcc编译器）：x % 2^n = (x + (2^n - 1) & (2^n - 1)) - (2^n - 1)
      新公式示例：x % 8 = (x + 7 & 7) - 7
      clang编译器：
      x >=0：x-(x+0&(~7))
      x < 0：x-(x+7&(~7))

代码示例：

```
printf("%d", argc % 8);
// 对应的汇编代码
  movsxd  rbx, ecx
  mov edx, ebx
  and edx, 80000007h
  jge short loc_14000108A
  dec edx
  or edx, 0FFFFFFF8h
  inc edx
loc_14000108A: ; CODE XREF: main+11↑j
  lea rcx, Format ; "%d"
  call printf

printf("%d", (unsigned long long)argc % 8);
// 对应的汇编代码
mov rax, rbx
lea rcx, Format     ; "%d"
cqo
and edx, 7
add rax, rdx
and eax, 7
sub rax, rdx
mov rdx, rax
call printf
```

2. **模非2的幂**：x % y = x - x / y * y，余数 = 被除数 - 商 * 除数

代码示例：

```
printf("%d", argc % 3);
// 对应的汇编代码
mov eax, 1431655766
mov ebx, ecx
imul ecx
mov eax, edx
shr eax, 31
add edx, eax ; x / 3
lea eax, [rdx+rdx*2] ; x / 3 * 3
mov edx, ecx
sub edx, eax ; x - x / 3 * 3
lea rcx, Format ; "%d"
call printf

printf("%d", (unsigned long long)argc % 3);
// 对应的汇编代码
mov eax, 2863311531
```

```
lea rcx, Format ; "%d"
mul ebx
shr edx, 1
lea r8d, [rdx+rdx*2]
mov edx, ebx
sub edx, r8d
call printf
```

三目运算

- x == 0 ? 8 : 9 这种两值相差一的，利用 setz/setnz reg 条件设置指令
- 其他情况利用 cmovz 条件传送指令

除法优化公式

> 根据生成的汇编代码公式识别除法（主要记住4种），无符号不需要+1操作

- 无符号：
  公式1：(x + 2^n - 1) >> n
  公式2：((x - (x * M >> 32)) >> n1 + (x * M >> 32))) >> n2 -- **溢出还原除数**
- 有符号：看M决定正负数
  公式1：(x * M >> 32 >> n) + 1 （+1有符号，不加1无符号）
  公式2：(x * M >> 32 + x)>> n + 1 正数
  公式3：(x * M >> 32 - x)>> n + 1 负数

被除数还原公式

正数还原除数: 2^n / M
负数还原除数: 2^n / (2^32 - M)
溢出还原除数: 2^n / (2^32 + M)