2021/03/15_x86逆向_第8课_三目运算

笔记本: x86逆向-C
创建时间: 2021/3/15 星期一 9:59
作者: ileemi

三目运算

> 三目运算各种情况的优化

1. A2 或者 A3，不是常量，jxx
2. A1是等值判断，neg/sbb原型和推广
3. A1是区间判定，cmp/setxx 原型和推广 (1、2、3 VC++6.0环境下)
4. VS20xx新增 cmovxx（条件传送）

表达式为常量的等值比较

> 表达式为常量时，三目运算的原型如下:

VC++ 6.0，Debug:
示例1:

```
int main(int argc, char* argv[])
{
  return argc == 0 ? 0 : -1;
}
// 对应的汇编代码
mov eax, [ebp+argc]
neg eax
sbb eax, eax

// 原型:
mov reg, argc
neg reg // reg = 0, CF = 0; reg != 0, CF = 1
sbb reg, reg // reg = reg - reg - CF = 0 - CF
```

```
// CF = 0，reg = 0 - 0 = 0
// CF = 1，reg = 0 -1 = -1
```

> 求补的优化是取反加一，原型是 "0-操作数"。neg 指令会影响 "CF" 标志位，reg = 0, CF = 0; reg != 0, CF = 1。

优化条件：表达式2、3为常量。不是常量转换成分支（if else）结构。
示例2：

```
int main(int argc, char* argv[])
{
  return argc == 0 ? 0 : 1;
}
// 对应的汇编代码
xor eax, eax
cmp [ebp+argc], 0
setne al
// setne 指令检查 ZF 标志位，当 ZF == 1时，al = 0，反之，al = 1
```

示例3：

```
int main(int argc, char* argv[])
{
  return argc == 0 ? 7 : 1;
}
// 对应的汇编代码
mov eax, [ebp+argc]
neg eax
sbb eax, eax // 0 或者 -1
and al, 0FAh // -6
add eax, 7

定式：
mov reg, xxx
neg reg
sbb reg, rag
and reg, A3 - A2
add reg, A2
```

示例4：表达式1大于0，所以个sub指令，小于0时，所以个add指令

```
int main(int argc, char* argv[])
{
  return argc == 999 ? 21 : 25;
}
// 对应的汇编代码
```

```
mov eax, [ebp+argc]
sub eax, 3E7h // 表达式1小于0时，使用add指令
neg eax
sbb eax, eax
and eax, 4 // 25 - 21
add eax, 15h // 21
```

表达式为常量的区间比较

原型定式：

```
xor eax, eax
cmp reg, xxx
setxx, al
```

代码示例：

```
int main(int argc, char* argv[])
{
  return argc > 999 ? 21 : 25;
}
// 对应的汇编代码，写法1
xor eax, eax
cmp [ebp+argc], 3E7h
setle al // 小于等于（不大于）
dec eax
and al, 0FCh // -4
add eax, 19h // 25
// 验证
argc > 999，eax = 0，eax = -1，eax = -4，eax = -4 + 25 = 21
argc <= 999，eax = 1，eax = 0，eax = 0，eax = 0 + 25 = 25

// 对应的汇编代码，写法2
xor eax, eax
cmp [ebp+argc], 3E7h
setg al // 大于
dec eax
and al, 4
add eax, 21
// 验证
argc > 999，eax = 1，eax = 0，eax = 0，eax = 0 + 21 = 21
argc <= 999，eax = 0，eax = -1，eax = 4，eax = 4 + 21 = 25
```

定式：

```
xor r32, r32
cmp reg, xxx
setxx r8 // 采用反条件
dec r32
and r32, A3 - A2 // A -- 表达式
add reg, A2
```

表达式为变量

> 表达式为变量（一个或多个变量）时的三目运算产生的汇编代码会出现跳转分支结构。

代码示例：

```
int main(int argc, char* argv[])
{
  return argc > 999 ? 21 : argc;
}
// 对应的汇编代码 Debug
  cmp [ebp+argc], 999
  jle short loc_40103A
  mov [ebp+var_4], 21
  jmp short loc_401040
loc_40103A:
  mov eax, [ebp+argc]
  mov [ebp+var_4], eax
loc_401040:
  mov eax, [ebp+var_4]

 // Release
  mov eax, [esp+argc]
  cmp eax, 999
  jle short locret_401010
  mov eax, 21
locret_401010:
  retn
```

cmovxx

原型：

```
mov reg, A2
cmp reg, xxx // reg 和 xxx 作比较
```

```
cmovle reg, A3
```

代码示例:

```
return argc > 999 ? 21 : 220;
// 对应的汇编代码
cmp [ebp+argc], 999
mov eax, 220
mov ecx, 21
cmovg eax, ecx // 大于就进行覆盖 eax = 21，反之 eax = 220
```