2021/03/16_x86逆向_第9课_流程控制(if、if-elses、witch-case)

笔记本: x86逆向-C
创建时间: 2021/3/16 星期二 10:58
作者: ileemi

、[TOC]

流程控制

if

> 常量判定时，编译器会判定符合条件的语句是否执行，条件结果为假时，编译器会对其进行优化（简值优化）。判断条件表达式为常量会生成顺序结构，当有变量时，生成的汇编代码有条件跳转。

代码示例:

```
int main(int argc, char* argv[])  {
  if (argc > 5) {
    printf("Hello World!\n");
  }
  return 0;
}
// 对应的汇编代码
  cmp [esp+argc], 5
  jle short loc_401014
  push offset aHelloWorld ; "Hello World!\n"
  call sub_401020
  add esp, 4
loc_401014:
  xor eax, eax
  retn
```

> 单分支结构的汇编代码定式:

```
  jxx IF_END // jxx 是高级语言判定的反条件
  ...
IF_END:
  ...
```

> if 语句就是单分支结构，jxx跳转语句后跟一个标号，标号上面没有其它的跳转语句。

if else

> VC++6.0，工程设置中点击 "C/C++" --> "Optimizations"，将原来的 "Maximize Speed" 改为 "Minimize Size"调整编译器会程序的优化方案。之后再使用IDA打开分析分支结构。

代码示例：

```
int main(int argc, char* argv[])  {
  if (argc > 5) {
    printf("z\n");
  }
  else {
    printf("World!\n");
  }
  return 0;
}
// 对应的汇编代码 (Release)
  cmp [esp+argc], 5
  jle short loc_40100E
  push offset aHello   ; "Hello\n"
  jmp short loc_401013
loc_40100E:
  push offset aWorld   ; "World!\n"
loc_401013:
  call sub_40101C
  pop ecx // 类似 add esp, 4
  xor eax, eax
  retn
```

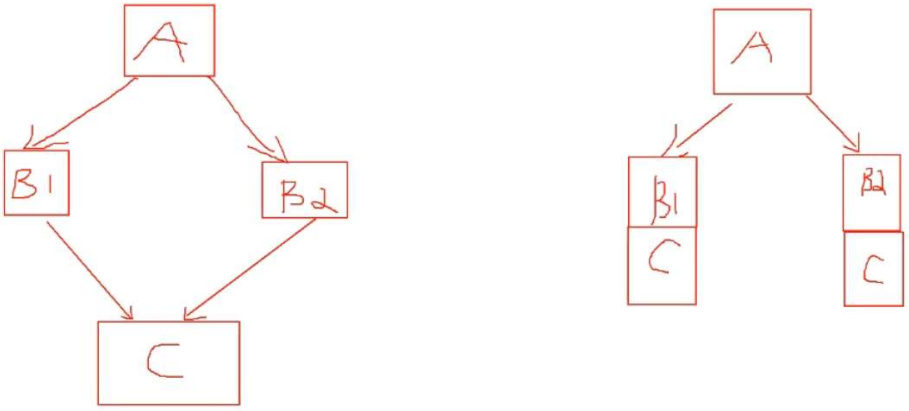
> 双分支结构的汇编代码定式：

```
  jxx ELSE_BEGIN // 该地址为 else 语句块的首地址
IF_BEGIN:
  ... // if 语句块内的执行代码
IF_END:
  jmp ELSE_END
ELSE_BEGIN:
  ...
ELSE_END
```

> if else 语句就是双分支结构，jxx跳转语句后跟一个标号，标号上面有另一个跳转语句，跳转到第一个标号的后面。

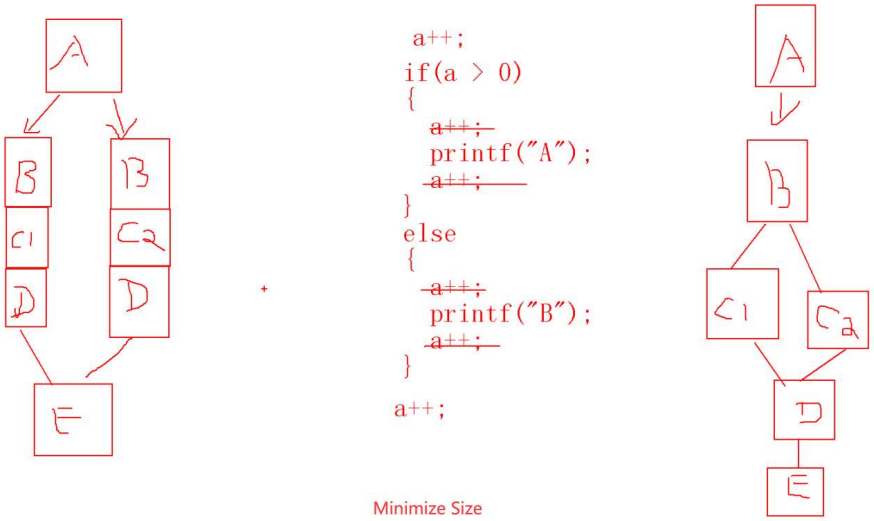
优化方案：

- Maximize Speed（速度优先）：add esp, 4 会使用 pop reg 进行代替。



Maximize Speed

- Minimize Size（空间优化）



Minimize Size

if else if

> if else if 为多分支结构，每个分支下都有一个跳转指令（公共出口）。分支的优化支持空间优化以及时间优化。

代码示例：

```
int main(int argc, char* argv[]) {
  if (argc > 5) {
    printf("argc > 5\n");
  }
  else if (argc > 3) {
    printf("argc > 3!\n");
  }
```

```
  else if (argc > 2) {
    printf("argc > 2!\n");
  }
  else {
    printf("world!\n");
  }
  return 0;
}
// 对应的汇编代码
  mov eax, [esp+argc]
  cmp eax, 5
  jle short loc_401010
  push offset aArgc5   ; "argc > 5\n"
  jmp short loc_40102D // loc_40102D 为公共出口
loc_401010:
  cmp eax, 3
  jle short loc_40101C
  push offset aArgc3   ; "argc > 3!\n"
  jmp short loc_40102D
loc_40101C:
  cmp eax, 2
  jle short loc_401028
  push offset aArgc2   ; "argc > 2!\n"
  jmp short loc_40102D
loc_401028:
  push offset aWorld   ; "world!\n"
loc_40102D:
  call sub_401036
  pop ecx
  xor eax, eax
  retn
```

> 多分支结构对应的汇编代码定式：

```
// 会影响标志为的指令
  jxx ELSE_IF_BEGIN // 跳转到下一条 else if 语句块的首地址
IF_BEGIN:
  ... // if 语句块内的执行代码
IF_END:
  jmp END // 公共出口
ELSE_IF_BEGIN: // else if 语句块的起始地址
  jxx ELSE_BEGIN
  ... // else if 语句块的执行代码
ELSE_IF_END: // else if 语句结尾处
  jmp END // 跳转到多分枝结构的结尾地址
ELSE_BEGIN: // else 语句块的起始地址
```

```
  ... // else 语句块的起内的执行代码
END: // 多分支结构的结尾处
```

switch case

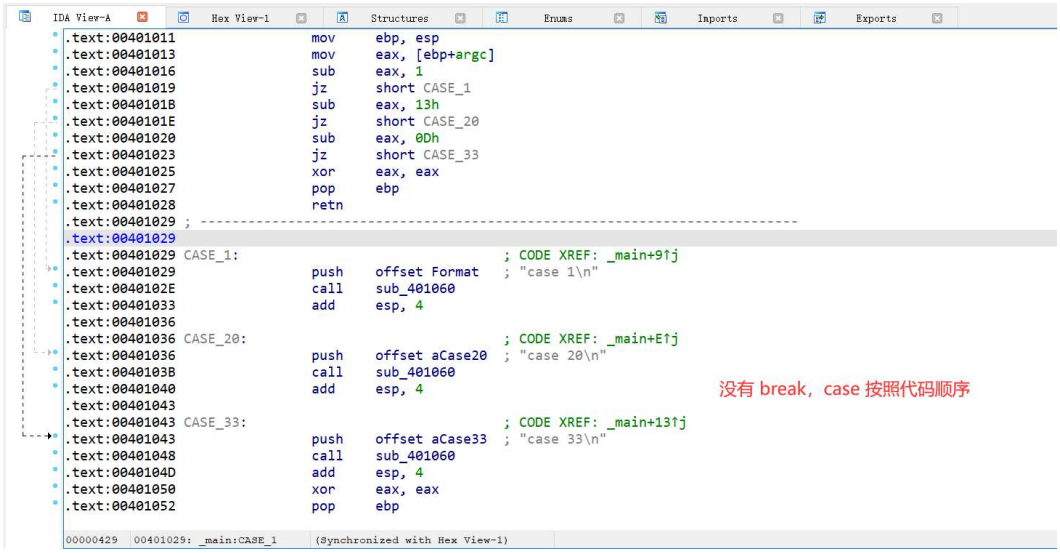
情况1：case语句块在3条以内时（3条分支以内），switch 语句使用对应次数的条件跳转，分别进行比较。

代码示例：

```
int main(int argc, char* argv[]) {
  switch(argc) {
  case 1:
    printf("case 1\n");
    break;
  case 2:
    printf("case 2\n");
    break;
  case 3:
    printf("case 1\n");
    break;
  }
  return 0;
}
// 对应的汇编部分代码 （Debug）
mov eax, [ebp+argc]
mov [ebp+var_4], eax
cmp [ebp+var_4], 1
jz short loc_401042
cmp [ebp+var_4], 2
jz short loc_401051
cmp [ebp+var_4], 3
jz short loc_401060
jmp short loc_40106D // 引导部分

//  实际执行部分
...
```

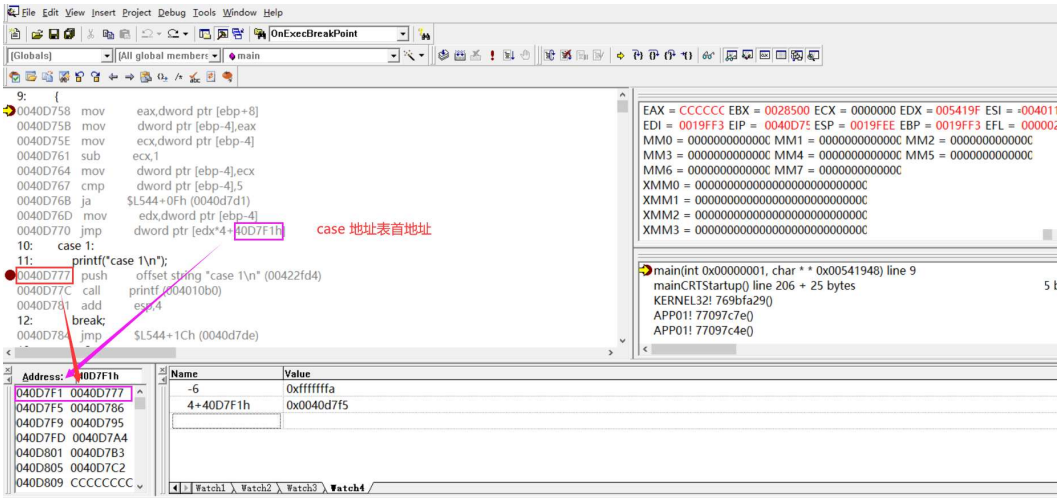
示例2：

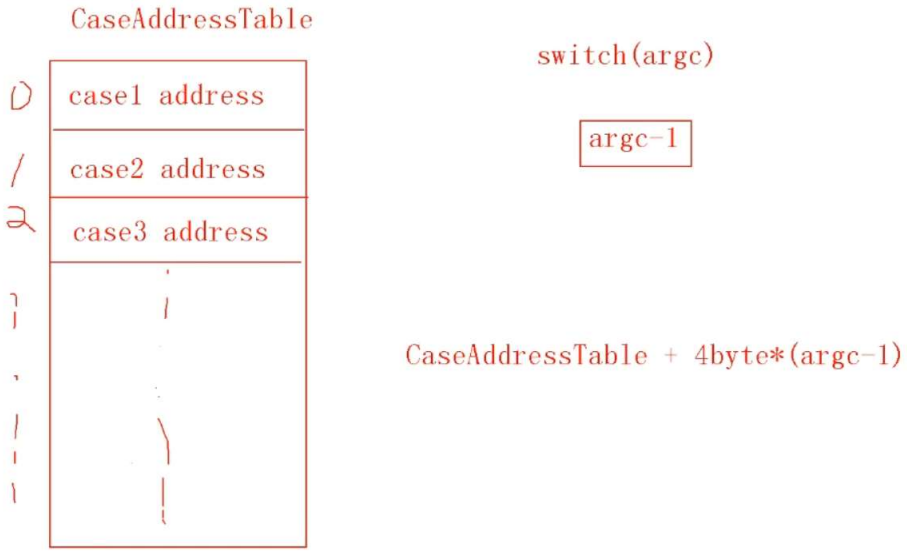


情况2：case 语句的标号为一个数值且为有序序列（1~7）。为了降低比较的次数，提高效率，编译器在编译时，会将每个 case 语句块的地址预先保存到数组中。通过 switch 的参数，依此查询 case 语句块地址的数组，得到对应的 case 语句块的首地址。

代码示例：

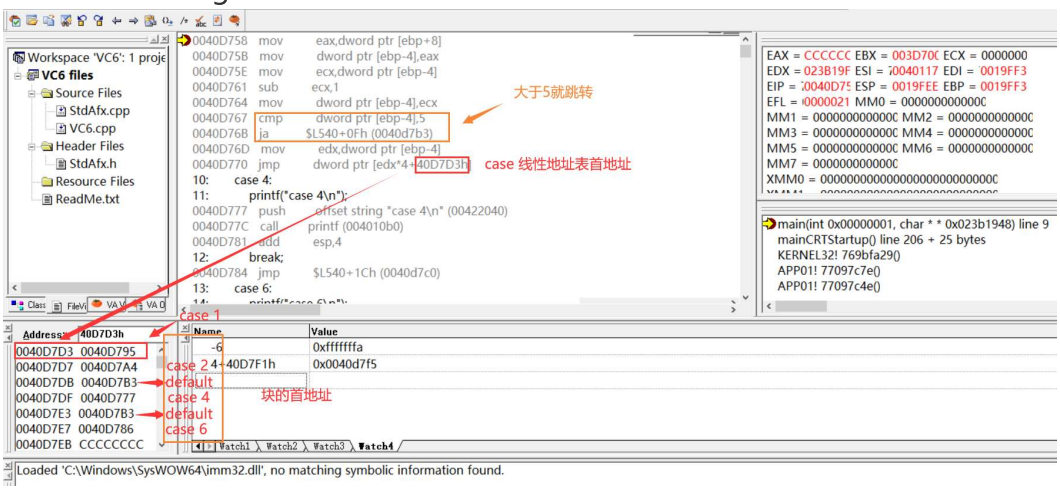
```
int main(int argc, char* argv[]) {
  switch(argc) {
  case 1:
    printf("case 1\n");
    break;
  case 2:
    printf("case 2\n");
    break;
  case 3:
    printf("case 3\n");
    break;
  case 4:
    printf("case 4\n");
    break;
  case 5:
    printf("case 5\n");
    break;
  case 6:
    printf("case 6\n");
    break;
  default:
    printf("default 1\n");
  }
  return 0;
}
// 对应的汇编代码 VC++ 6.0 Release
```





> 编译器回在编译过程中对 case 线性地址表进行排序，如果 case 的顺序为 4、6、1、2，在 case 线性表中，编译器会将它们的语句块的首地址进行排序。将case 1语句块的首地址放在 case 线性地址表的第0项。case 值不连续的情况下，case 线性地址表中会将不连续case 值的中间填充 default 语句块的首地址。

VC++ 6.0 Debug：



VC++ 6.0 Release：

```
.text:00401036                call    sub_401080
.text:0040103B                add     esp, 4
.text:0040103E                xor     eax, eax
.text:00401040                retn
.text:00401041 ; ---------------------------------------------------------------------------
.text:00401041
.text:00401041 CASE_2:                                ; CODE XREF: _main+A↑j
.text:00401041                                        ; DATA XREF: .text:jpt_40100A↓o
.text:00401041                push    offset aCase2   ; jumptable 0040100A case 2
.text:00401046                call    sub_401080
.text:0040104B                add     esp, 4
.text:0040104E                xor     eax, eax
.text:00401050                retn
.text:00401051 ; ---------------------------------------------------------------------------
.text:00401051
.text:00401051 DEFAULT:                               ; CODE XREF: _main+8↑j
.text:00401051                                        ; _main+A↑j
.text:00401051                                        ; DATA XREF: ...
.text:00401051                push    offset aDefault ; jumptable 0040100A default case, cases 3,5
.text:00401056                call    sub_401080
.text:0040105B                add     esp, 4
.text:0040105E
.text:0040105E SWITCH_END:
.text:0040105E                xor     eax, eax
.text:00401060                retn
.text:00401060 _main          endp
```

情况3：对于非线性的 switch 结构，可以采用制作索引表的方法来进行优化，索引表优化需要两张表，case 语句块地址表、case 语句块索引表。

- 地址表中的每一项保存一个 case 语句块的首地址。有几个 case 语句块就有几项（包括 default 语句块，没有保存 switch 结束地址）。
- 索引表中保存地址表的编号，它的大小等于最大 case 值和最小 case 值的差。当差值大于 255 时，这种优化方案也会浪费空间（可使用树方式进行优化）。当差值小于或等于255时，表中每一项为一个字节大小，保存的数据为 case 语句块地址表中的索引编号。

示例，如下图所示：

```
.text:00401000
.text:00401000 ; int __cdecl main(int argc, const char **argv, const char **envp)
.text:00401000 _main          proc near               ; CODE XREF: start+AF↓p
.text:00401000
.text:00401000 argc           = dword ptr  4
.text:00401000 argv           = dword ptr  8
.text:00401000 envp           = dword ptr  0Ch
.text:00401000
.text:00401000                mov     eax, [esp+argc]
.text:00401004                add     eax, -14        ; switch 78 cases
.text:00401007                cmp     eax, 77         ; case 最小值0-(-14) = 14,最大值为77+14 = 91
.text:0040100A                ja      short def_401014 ; jumptable 00401014 default case, cases 15-24,26-39,41-61,63-72,74-90
.text:0040100C                xor     ecx, ecx
.text:0040100E                mov     cl, ds:byte_4010A8[eax] ; 下标表
.text:00401014                jmp     ds:jpt_401014[ecx*4] ; switch jump
.text:0040101B ; ---------------------------------------------------------------------------
.text:0040101B
.text:0040101B CASE_40:                               ; CODE XREF: _main+14↑j
.text:0040101B                                        ; DATA XREF: .text:jpt_401014↓o
.text:0040101B                push    offset aCase40  ; jumptable 00401014 case 40
.text:00401020                call    sub_401100
.text:00401025                add     esp, 4
.text:00401028                xor     eax, eax
.text:0040102A                retn
.text:0040102B ; ---------------------------------------------------------------------------
.text:0040102B
```

地址表

下标表

```
.text:0040108A _main          endp
.text:0040108A
.text:0040108A ; ---------------------------------------------------------------------------
.text:0040108B                align 4
.text:0040108C jpt_401014     dd offset CASE_14       ; DATA XREF: _main+14↑r
.text:0040108C                dd offset CASE_25       ; jump table for switch statement
.text:0040108C                dd offset CASE_40
.text:0040108C                dd offset CASE_62
.text:0040108C                dd offset CASE_73
.text:0040108C                dd offset CASE_91
.text:0040108C                dd offset def_401014
.text:004010A8 byte_4010A8    db 0,6,6,6,6,6,6,6,6,6  ; DATA XREF: _main+E↑r
.text:004010A8                db 6,1,6,6,6,6,6,6,6,6  ; indirect table for switch statement
.text:004010A8                db 6,6,6,6,6,6,2,6,6,6
.text:004010A8                db 6,6,6,6,6,6,6,6,6,6
.text:004010A8                db 6,6,6,6,6,6,6,6,3,6
.text:004010A8                db 6,6,6,6,6,6,6,6,6,4
.text:004010A8                db 6,6,6,6,6,6,6,6,6,6
.text:004010A8                db 6,6,6,6,6,6,6,6,6,5
.text:004010F6                align 10h
.text:00401100
.text:00401100 ; =============== S U B R O U T I N E =======================================
.text:00401100
.text:00401100
.text:00401100 sub_401100     proc near               ; CODE XREF: _main+20↑p
.text:00401100                                        ; _main+30↑p ...
```

地址表

下标表

情况3出现的条件：

- max_case - min_case <= 255
- VC++6.0 下，case之间不连续且差值要大于12（微软平台，小于12会更换方案2）