重命名和注释

- 变量重命名 快捷键(y)，若想恢复则更名为空白名称即可
- 方法/寄存器重命名 快捷键(N)
- 注释：常规注释(:)/可重复注释(;),「;」表示该行为注释

如果你所分析的文件类型与常见编译器生成的普通二进制可执行文件相差甚大，你可能需要对反汇编分析和显示过程进行更多的控制。在分析采用自定义文件格式（IDA无法识别）的模糊代码或文件时，情况更是如此。

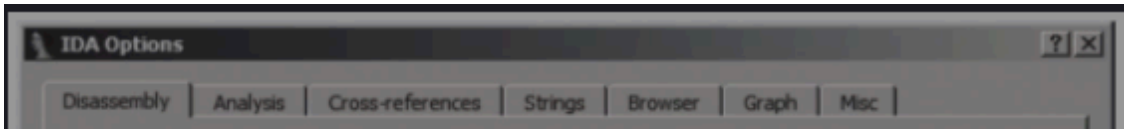
此时便需要手动帮助IDA分辨代码和数据，通常分为以下几类：

- 数据转换为代码
- 代码转换为数据
- 指定一个指令序列为函数
- 更改现有函数的起始或结束地址
- 更改指令操作数的显示格式

代码显示选项

在转换之前，可以通过一些设置来让IDA展示更多反编译信息帮助我们判断和决策。

常规的一行反汇编行中包含来标签、助记符和操作数，我们也可以通过options -> general打开IDA Options设置，选择「Disassembly」选项卡，为反汇编行选择其他显示的部分信息。



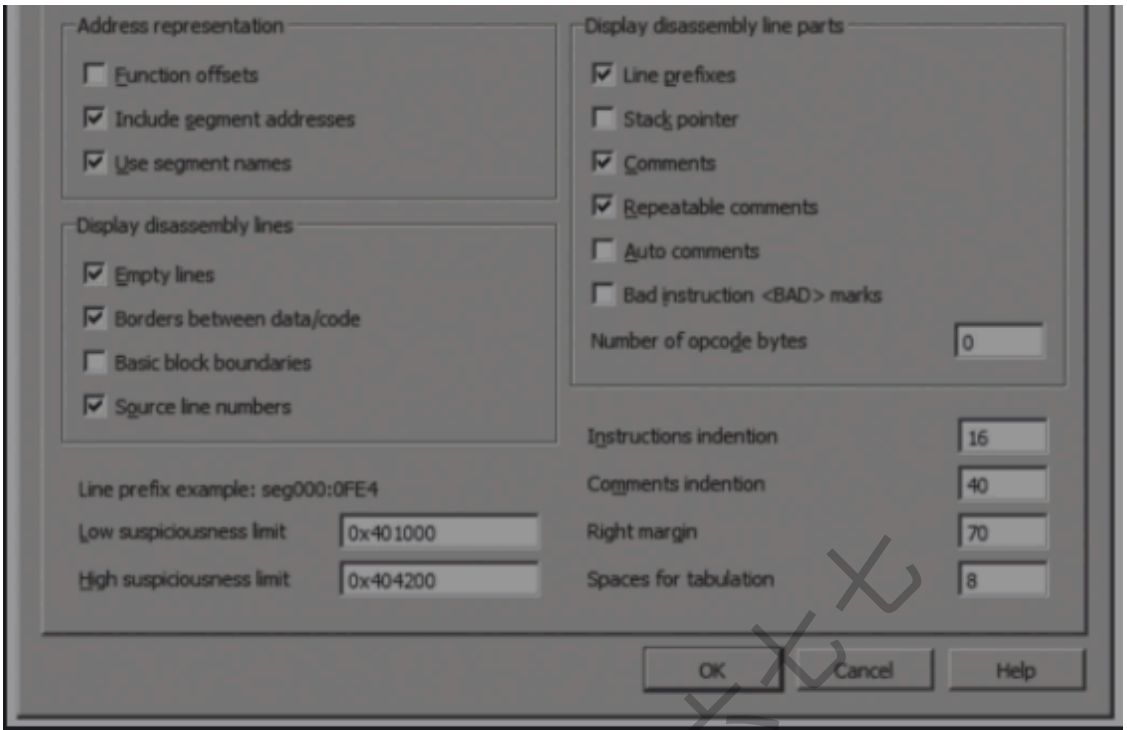


图7-4 反汇编行显示选项

- Line prefixes 行前缀，展示样式为：section:address
- **Stack Pointer（栈指针）** 这个不必多说，从前一章可以知道栈帧中栈指针多么重要。 选中栈指针选项，IDA将会显示栈指针在每个函数执行过程中的相对变化。这样做有助于识别调用约定方面的差异（例如，IDA可能不知道某个特殊的函数使用的是stdcall调用约定），或者确定对栈指针的不寻常操纵。任何时候，如果IDA遇到一个函数返回语句，并检测到栈指针的值不为0，这时，**IDA将标注一个错误条件**，并将相关指令以红色显示。有时候，这样做可能是有意阻挠自动分析。这时候就可以手动调整栈指针来正确反汇编。
- Numbers of opcode bytes（操作码字节数）以指定IDA应为每个指令显示的机器语言字节的数量，选择性地查看与汇编语言指令混杂在一起的机器语言字节。

栈指针调整

如前所述，IDA会尽其所能跟踪函数内每一条指令上的栈指针的变化。IDA跟踪这种变化的准确程度，在很大程度上影响着函数的栈帧布局的准确程度。如果IDA无法确定一条指令是否更改了栈指针，你就需要手动调整栈指针。

如果一个函数调用了另一个使用stdcall调用约定的函数，就会出现上述情况，这是最简单的一种情况。如果被调用的函数位于IDA无法识别的共享库中，那么，IDA并不知道该函数使用了stdcall调用约定，也就无法认识到：被调用的函数会将栈指针

修改后返回。因此，IDA会为函数的剩余部分提供一个错误的栈指针值。

如何进行调整：

首先开启栈指针选项，假设有以下几行指令：

028 call some_imported_func

028 mov ebx,eax

mov指令处的栈指针理论上应为01C，因为在函数some_imported_func调用返回时会清除栈中的3个参数（stdcall调用约定），但IDA不知道这是一个stdcall，因此不会进行清除，导致栈指针不变，此时我们可以在call指令处
Alt+K（Edit▶Functions▶Change Stack Pointer），然后指定栈指针更改的字节数，在本例中为12。

但这样只能处理这一处的调用，假设有其他地方调用了some_imported_func，我们可以在该导入函数的导入表条目中设置「已删除字节的数量」,通过编辑这个函数，你可以指定它在返回时从栈中删除的字节数，IDA将会“扩散”这一信息，将其应用于调用该函数的每一个位置，立即纠正每个位置的栈指针计算错误。

格式化指令操作数

为了使反汇编代码更具可读性，IDA尽可能地使用符号名称，而非数字。因此我们在IDA的反汇编窗口中，基本看到的都是符号名称而非数字常量来代表全局变量、栈帧中的偏移量，但是也存在数字常量的使用情况，IDA此时会将常量格式化成十六进制表示。

如果我们可以确定某个常量是个标准符号常量，我们可以右键常量 -> Use standard symbolic constant然后将常量用字符串替换，例如0x0AH在X.25网络连接中可以替换为AF_CCITT，此时mov [ebp+var_60],0Ah就会被替换为mov [ebp+var_60],AF_CCITT。

更改函数的起始或结束地址

新建函数

在某些情况下，你可能需要在没有函数的地方创建新函数。新函数可以由已经不属于某个函数的现有指令创建，或者由尚未被IDA以任何其他方式定义（如双字或字符串）的原始数据字节创建。

操作方式：Edit▶Functions▶Create Function，即可创建一个新函数。在必要时，IDA会将数据转换成代码。接下来，它会向前扫描，分析函数的结构，并搜索返回语句。如果IDA能够找到正确的函数结束部分，它将生成一个新的函数名，分析栈帧，并以函数的形式重组代码。如果它无法找到函数的结束部分，或者发现任何非法指令，则这个操作将以失败告终。

删除函数

你可以使用Edit▶Functions▶Delete Function命令删除现有函数。

更改起始和结束地址

在Edit Function窗口中，可以设置方法的：

- start address
- end address
- local variables area 函数局部变量使用的栈字节数
- saved registers 函数为调用房保护寄存器所使用的字节数。
- BP Based frame 这个特性表示函数利用了一个帧指针。多数情况下，可以通过分析函数的“序言”来自动确定这一点。但是，如果通过分析无法确定给定的函数是否使用了帧指针，就可以手动选择这个特性。如果你手动选择了这个特性，一定要相应地调整保存的寄存器的大小

数据与代码互相转换

在自动分析阶段，字节有时可能被错误地归类。数据字节可能被错误地归类为代码字节，并被反汇编成指令；而代码字节可能被错误地归类为数据字节，并被格式化成数据值。

首先重新格式化反汇编代码：右击你希望取消定义的项目，在结果上下文菜单中选择Undefine（也可使用Edit▶Undefine命令或热键U），即可取消函数、代码或数据的定义。使用“单击并拖动”操作选择一个地址范围，可以取消大范围内的定义。

然后重新反汇编该区域字节：要反汇编一组未定义的字节，右击其中的第一个字节，在上下文菜单中选择Code（也可使用Edit ▶Code或热键C）。

指定数据大小

IDA提供了许多数据大小/类型说明符。最常见的说明符包括db、dw和dd，分别代表

1字节、2字节和4字节数据。

Options▶ Setup Data Types（选项▶设置数据类型）对话框中可以设置数据转盘，然后通过快捷键D来不断切换选中地址所在的数据项的数据类型。

如果你缩小某个项，例如，由dd（4字节）转换成db（1字节），则额外的字节（这里为3字节）将变成未定义字节。如果你增大某个项，且该项之后的字节为已定义字节，这时，IDA会委婉地提醒你：是否希望取消下一个项的定义，以扩大当前的项。这时，IDA显示的消息为：“直接转换成数据吗？”通常，这条消息表示IDA会取消随后足够多的项目的定义，以满足你的要求。例如，将字节数据（db）转换为双字数据（dd）时，还需要另外3字节才能构成新的数据项。

IDA中的数组

数组是一串连续的地址空间，在IDA中不会直接提供数组大小方面的消息，如何识别一个数组也是IDA中需要学习的知识点。

```
.rdata:00402060 unk_402060      db    0    ; DATA XREF: sub_401350+8↑o
.rdata:00402060                            ; sub_401350+18↑o
.rdata:00402061                 db    0
.rdata:00402062                 db    0
.rdata:00402063                 db    0
.rdata:00402064                 db    0
.rdata:00402065                 db    0
.rdata:00402066                 db    0
.rdata:00402067                 db    0
.rdata:00402068                 db    0
.rdata:00402069                 db    0
.rdata:0040206A                 db    0
```


例如上述汇编代码中，unk_402060之后的数据声明其中只有第一个项被指令引用，表明它可能是某个数组中的第一个元素。通常，数组中的其他元素并不直接引用，而是需要经过更加复杂的索引计算，通过其与数组开头之间的偏移量来引用。

IDA提供一些工具，可将连续的数据定义结合起来，组成一个单独的数组定义。要创建数组，首先选择数组中的第一个元素（这里我们选择的是unk_402060），然后通过Edit▶Array命令打开“创建数组”对话框。然后设置相关的数组信息，就可以变成类似:

byte_402060 db 1A0h dup(0)

的简单数组声明，它是一个名为byte_402060的字节数组（db），由416（1A0h）个0值构成。