2021/01/11_32位汇编_第2课_32位汇编宏和结构体的定义、使用_masm32的使用

笔记本:	32位汇编
创建时间:	2021/1/11 星期一 10:46
作者:	ileemi

加载程序时，程序的入口地址OllyDBG会自动向操作系统获取，获取不到可以通过OllyDBG从文件中获取。

汇编宏的定义

> 使用宏伪指令可提高程序的可读性

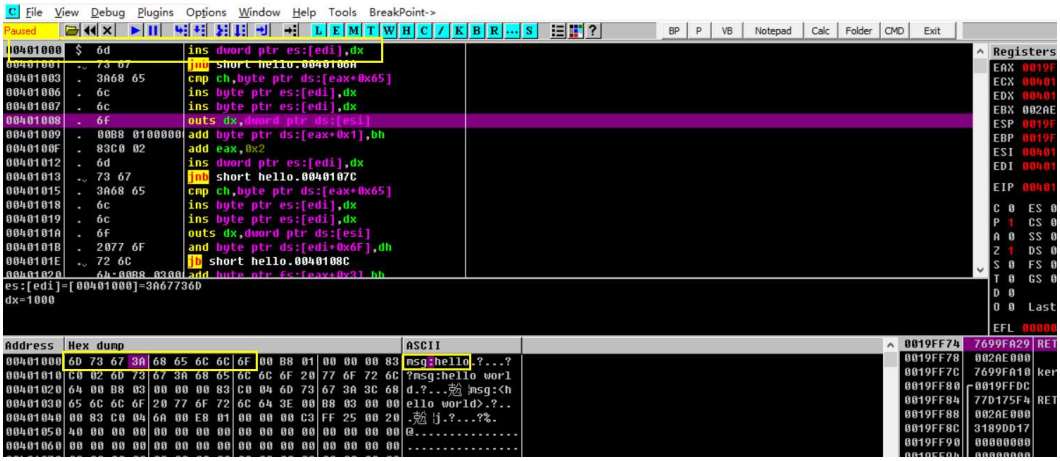
- **无参宏**：**equ（代码替换）**
  ;宏(无参宏)
  NULL EQU 0
  MB_OK EQU 0
- **带参宏**：**macro**（和多次调用函数相比，带参宏（空间换时间）这种方法指令周期要比函数的指令周期高（函数省内存空间，时间换空间））

> 现在编程中，当一个 "方法" 的汇编代码所占的内存较大时（所占用的字节数）就使用函数，占用内存较少时使用带参宏，类似高级语言的**内敛函数（编译器根据 "方法（函数）"的实现代码自动进行分析，所占内存较少的时候就当作宏展开，反之当作函数）**。

使用无有参宏传递字符串的时候，在宏内使用参数替换时需要在参数前后添加 "&" 字符（表示使用参数替换），不添加编译器将参数当作字符串。传递的字符串中间有空格的时候，需要将传递参数时，将参数前后分别添加 "<>" 字符。汇编中的尖括号（<>）转移字符为 "!"：<!<hello World!>>。函数写在代码区内。
代码示例：

```
include  hello.inc
includelib  user32.lib
includelib  kernel32.lib

MyAdd2  macro  n1,  n2,  n3
    ;  使用传递的参数n3需要在前后添加  "&"表示使用参数替换
    ;  防止参数当作字符串
    db  "msg:&n3&",  0
```

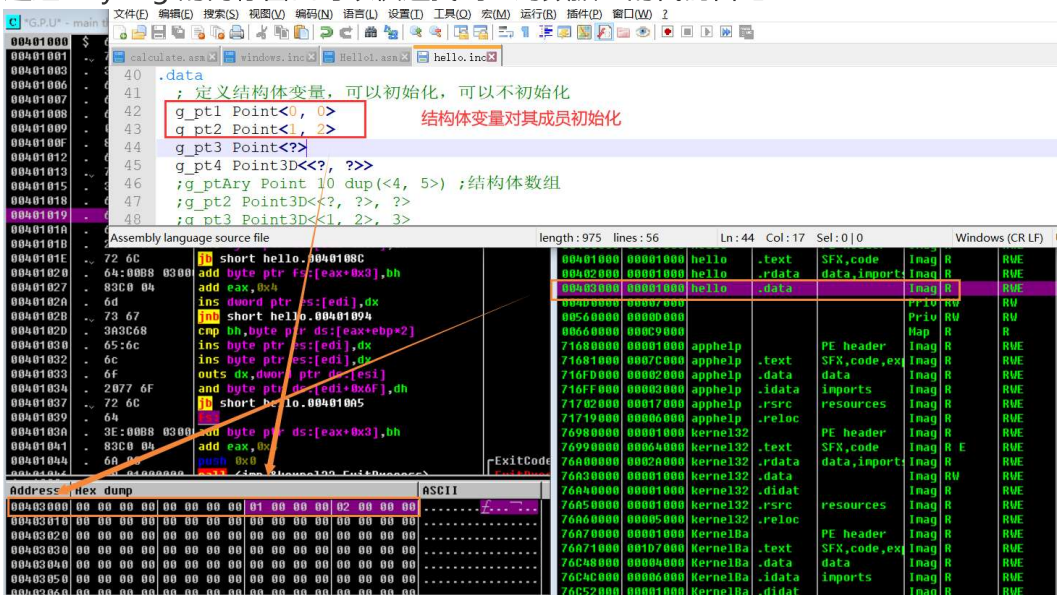
```
    mov   eax,   n1
    add   eax,   n2
endm

; 代码区
.code

START   proc
    MyAdd2   1,   2,   hello
    MyAdd2   3,   4,   <hello   world> ; 传递的字串中有空格使用 "<>" 号将字符串括起来
    MyAdd2   3,   4,   <!<hello   world!>>   ; 特殊符号需要转义
    invoke   ExitProcess,   0
    RET
START   endp
end     START
```



汇编结构体的定义

> 使用结构体伪指令可自动计算结构体中成员的偏移量:
> 伪指令：**struct**

通过 ollydbg 的内存窗口可以快速找到全局数据区的代码首地址：



结构体的定义，代码示例：

```
  .386     ;指令集
.model  flat,  stdcall  ;平坦模式（不分段),默认调用约定
OPTION  CASEMAP:none                    ;大小写敏感

;宏(无参宏)
NULL  EQU  0
MB_OK  EQU  0

;代码宏(有参宏)
MyMsg  MACRO  p1
    Mov  eax,  eax
endm

;常量区
.const
;结构体
Point  struct  4
    x  dword  ?    ;  定义成员，注意字节对齐(成员不初始化)
    y  dword  ?
Point  ends
Point3D  struct  4
    pt  Point<?>
    Z    dword  ?
Point3D  ends

;初始化数据区
.data
    ;  定义结构体变量，可以初始化，可以不初始化
    ;  g_pt1  Point<>          ;  成员使用默认值
    g_pt1  Point<0,  0>
```

```
    g_pt2  Point<1,  2>
    g_pt3  Point<?,  2>    ;  第一个成员不初始化，一个  "?"对应一个成员
    g_pt4  Point3D<<?,  ?>>
    ;g_ptAry  Point  10  dup(<4,  5>)  ;结构体数组
    ;g_pt2  Point3D<<?,  ?>,  ?>    ;结构体中套结构体
    ;g_pt3  Point3D<<1,  2>,  3>
    ;g_pt4  Point3D<<?,  2>,  3>
```

在代码区中使用结构体变量时，需要获取结构体变量的地址，写法：

- 一般写法：

```
;  参数多的时候偏移值不方便计算
;  成员位置发生变化的时候，也需要改动代码
mov  ebx,  offset  g_pt2
mov  eax,  dword  ptr  [ebx+0]  ; pt.x -- 手动计算偏移
mov  eax,  dword  ptr  [ebx+4]  ; pt.y
```

- 使用假设 **assume**：assume 伪指令允许将结构体的地址交给一个通用寄存器（将通用寄存器假设为结构体的首地址，一般使用 "**ebx, ebp**"），该寄存器就可以当作结构体来使用，通过该寄存器可以直接访问结构体数据成员，不使用的时候使用 nothing 进行取消。代码示例：

```
;  假设,编译器自动计算编译
ASSUME  ebx:ptr  Point    ;  把ebx当作Point结构体指针来使用
mov  eax,  [ebx].x          ;  编译时,  编译器会自动计算偏移
mov  eax,  [ebx].y
ASSUME  ebx:nothing       ;  取消假设后,  eax通用寄存器
```

> 在汇编中局部变量只能定义在函数或者宏中才能使用。没有保存 ebp 在代码段进行使用会有问题，需要在代码区 "START" 开始处添加 "proc"（修改为过程函数）。

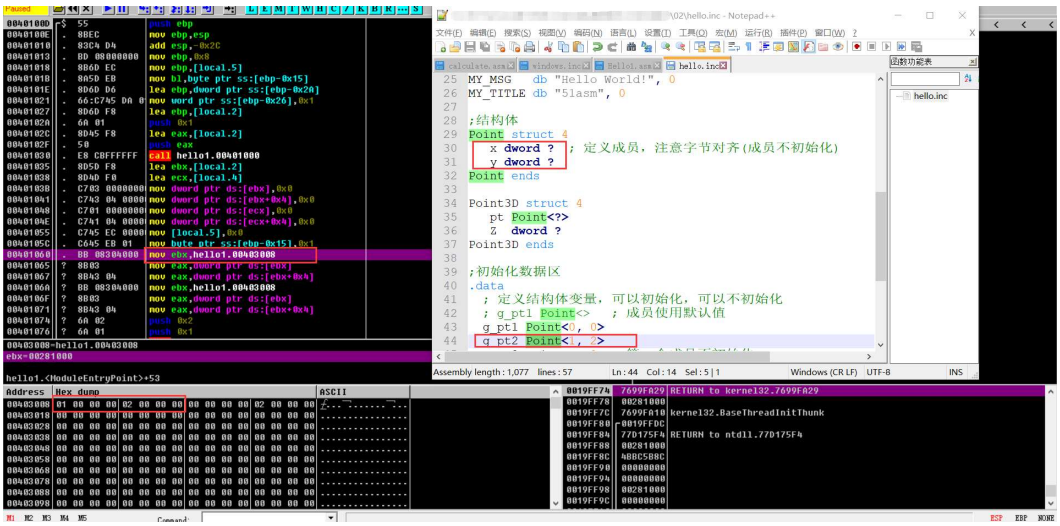
- 获取局部变量对应字节数的数值，代码如下：

```
    ;  源操作数写局部变量的标号就是取出对应的字节数
    mov  ebp,  Point        ;  将结构体的字节大小给ebp
    ;mov  ebp,  @pt         ;  编译不通过,结构体需要取总字节数，寄存器放不下
    mov  ebp,  @num         ;  取变量num的值  mov  ebp,  dword  ptr  [ebp-c]
    ;mov  ax,  @num2 ; 编译不通过
    mov  bl,  @num2
```

- 通过使用 addr可以将局部变量的地址当作参数传递给函数（在invoke中使用），代码示例：

```
include  hello.inc
includelib  user32.lib
includelib  kernel32.lib

MyAdd2  macro  n1,  n2,  n3
    ;使用传递的参数n3需要在前后添加  "&"表示使用参数替换
    db  "&n3&",  0
    mov  eax,  n1
    add  eax,  n2
endm

;代码区
.code
MyAdd1  proc  n1:dword,  n2:dword
    mov  eax,  n1
    add  eax,  n2
    ret
MyAdd1  endp

START  proc
    local  @pt:Point
    local  @pt2:Point
    local  @num:dword
    local  @num2:byte
    local  @ary[10]:word

    ;  源操作数写局部变量的标号就是取出对应的字节数
    mov  ebp,  Point          ;  将结构体的字节大小给ebp
    ;mov  ebp,  @pt           ;  编译不通过,结构体需要取总字节数，寄存器
放不下
    mov  ebp,  @num           ;  取变量num的
值  mov  ebp,  dword  ptr  [ebp-c]
    ;mov  ax,  @num2
    mov  bl,  @num2

    ;访问局部变量数组
    lea  ebp,  @ary           ;  获取数组首地址
    ;mov  @ary[2  *  type  @ary[0]],  1
    mov  [@ary  +  2  *  2],  1     ;  访问对应元素

    ;  获取局部变量地址
    lea  ebp,  @pt
    ;  使用invoke时，需要将局部变量的地址进行传递可使用  addr
    invoke  MyAdd1,  addr  @pt,  1

    lea  ebx,  @pt     ;  mov  ebp,  dword  ptr  [ebp-8]
```

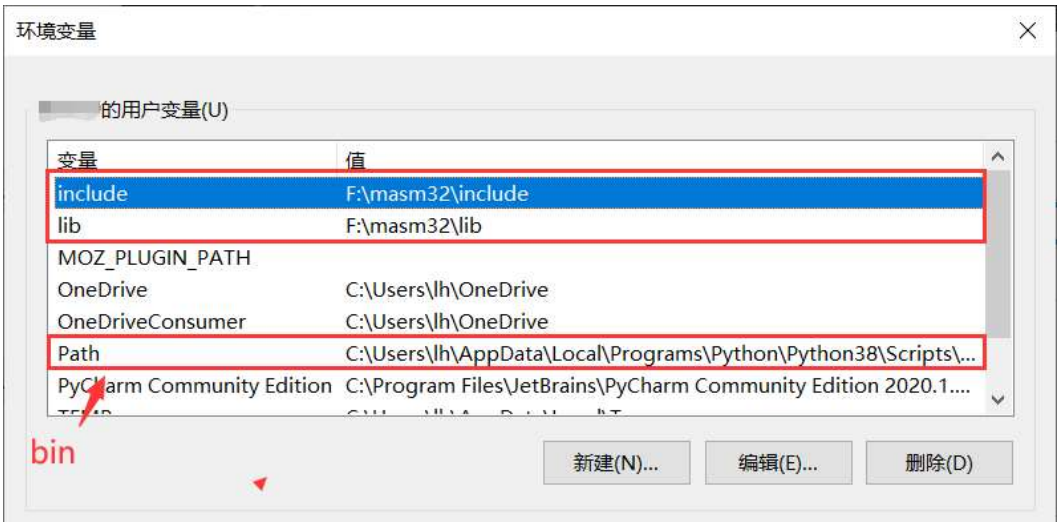
```
    lea  ecx,  @pt2    ;  mov  ebp,  dword  ptr  [ebp-8]
    assume  ebx:ptr  Point
    assume  ecx:ptr  Point
    mov  [ebx].x,  0
    mov  [ebx].y,  0
    mov  [ecx].x,  0
    mov  [ecx].y,  0
    assume  ebx:nothing  ;取消
    assume  ecx:nothing  ;取消
    mov  @num,  0
    mov  @num2,  1
    mov  ebx,  offset  g_pt2  ;  讲g_pt2地址赋值给ebx
    ;假设,编译器自动计算编译
    assume  ebx:ptr  Point    ;  把ebx当作Point结构体指针来使用
    mov  eax,  [ebx].x          ;  编译时,  编译器会自动计算偏移
    mov  eax,  [ebx].y
    assume  ebx:nothing       ;  取消假设后,  eax通用寄存器

    assume  eax:ptr  Point3D

    ;  参数多的时候偏移值不方便计算
    ;  成员位置发生变化的时候，也需要改动代码
    mov  ebx,  offset  g_pt2
    mov  eax,  dword  ptr  [ebx+0]  ;pt.x
    mov  eax,  dword  ptr  [ebx+4]  ;pt.y

    invoke  MyAdd1,  1,  2
    invoke  MyAdd1,  1,  2
    ;MyAdd2<1,2>
    ;MyAdd2<1,2>
    ;MyMsg<MB_OK>
    ;MyMsg<MB_OK>
    ;MyAdd2  1,  2,  <hello  world>
    ;MyAdd2  3,  4,  <hello  world>
    ;MyAdd2  3,  4,  <!<hello  world!>>  ;转义
    invoke  ExitProcess,  0
    RET
START  endp
end    START
```



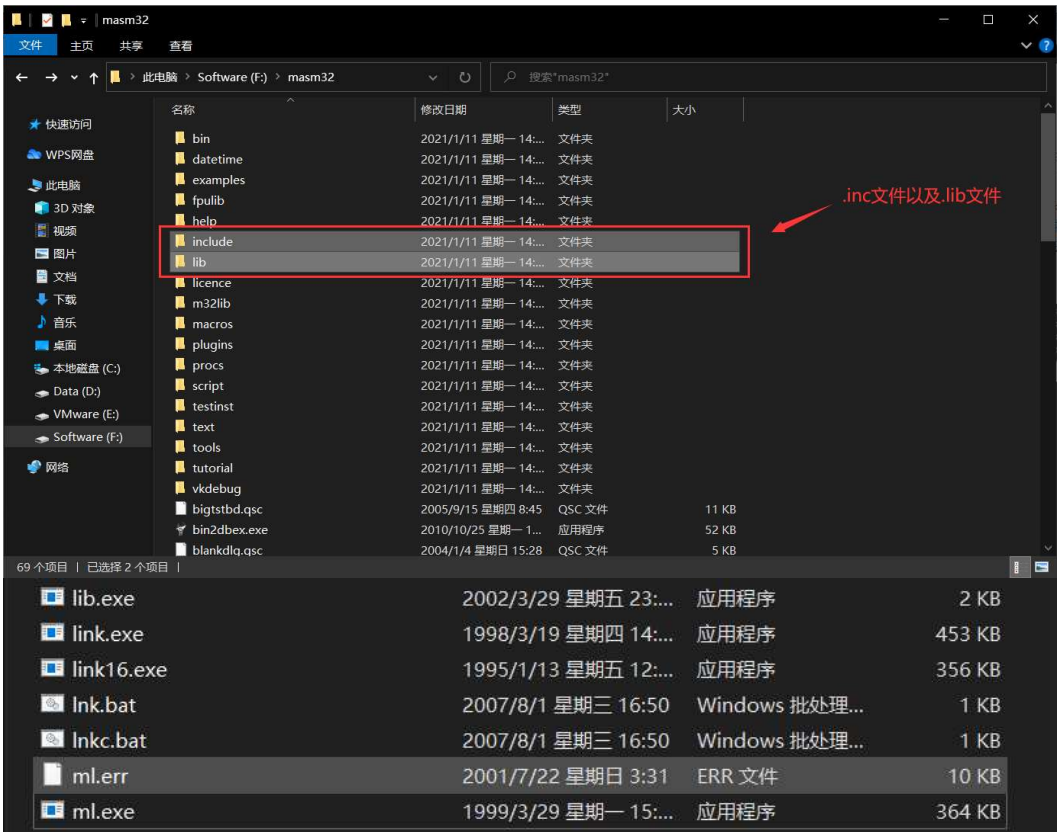
masm32

将.lib文件中的API通过解析格式生成对应的.inc文件（结构体函数声明，API分Unicode和ASCII版本）。之后在程序中需要使用API的时候只需要添加.inc文件即可，不在需要去写API对应的函数声明（内部同时集成了开发环境）。安装完成后，需要添加环境变量，如下图示例：

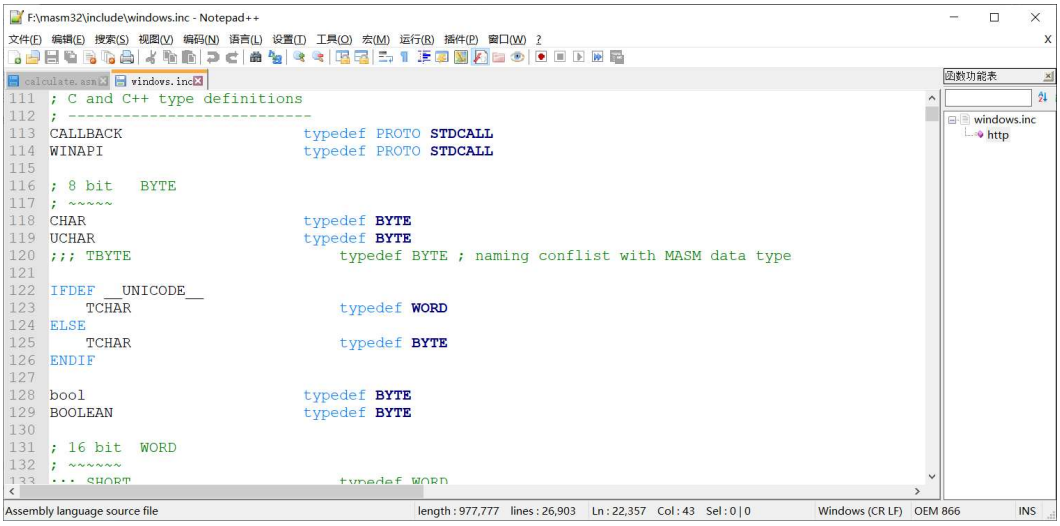
环境变量：



目录格式：



解析后的格式：



> "masm32\tools\l2inc" 下的 "l2inc.exe" 程序可以单独的将 ".lib" 文件生成对应的 ".inc" 文件（主要用于官方跟新库中的函数后，需要使用新库中的函数时可以使用该工具）。