**2021/01/08_32位汇编_第1课_32位汇编与16位汇编的区别、调试器的使用**

**笔记本:** 32位汇编
**创建时间:** 2021/1/8 星期五 10:46
**作者:** ileemi



# 32位汇编和16位汇编的区别

masm615集成开发工具：ml.exe 是一个32位程序，link.exe 是一个16位程序。

## 不分段

> 8086最大寻址范围是1M，而超过64K的内存区域访问要靠切换段基址。当Intel的CPU发展到32位后，内部有32条内存总线，寻址空间达到了4G。所以使用32位CPU及以后编写汇编代码不在需要分段。在访问内存数据的时候不用考虑段寄存器，直接访问内存即可（`[00401000H]`）。

原16位CPU中的段寄存器进行了保留（CS、DS、ES、SS），同时又扩展了两个段寄存器**FS和GS（给操作系统用的）**。保留之前的段寄存器是为了向下兼容16位CPU以及16位汇编程序。保留以及扩展的段寄存器供操作系统使用（做一些额外的管理作用）。

新代CPU也可以跑16位的汇编代码（目的就是向下兼容），但是操作系统不一定。

## 寄存器的变化

> 32位寄存器：源16位寄存器标号（除短寄存器外）前都加 'e'（扩展的意思）。例如：eax，ebx, ecx, edx，esi，edi，ebp，esp，eip，eflag。原来16位CPU的通用寄存器现在分别为32位CPU的通用寄存器的低16位（ax为eax的低16位）。

扩展的寄存器供操作系统进行使用：CR0~CR4、DR0~DR7。源16位寄存器不变。

## 代码分区

32位汇编不在需要分段，编写程序时，为了方便管理软，件数据和代码也应该进行拆分，写在一起不可取。所以就提出了分区的概念。

> 代码分区（有内存保护属性，防止误操作）：代码区（执行、读）、数据区（读、写）、常量区（只读）、堆栈区。

发明内存保护属性的根本原因是保护系统安全（通过中断可以修改操作系统的代码，有了内存保护属性就可以将操作系统的代码放到代码区就可以避免危险操作）。代码分区由操作系统进行，需要编译器支持。

**CPU运行模式：**

- **实地址模式**：只能运行16位汇编，英特尔的x86系列处理器为兼容早期的8086处理器，在上电后处于这个模式。8086有20位地址线，1M的线性地址空间。
- **保护模式**：只能运行32位汇编（由操作系统定义内存的访问属性，在由硬件进行检测），一个受保护并支持多任务的环境。大部分OS运行在这个模式下。80386有32位地址线，4G线性地址空间。 80386把4G只分为一个段，段基址0x00000000,段长0xFFFFFFFF(4G)。
- **虚拟8086模式**：模拟一个8086的环境来运行程序，解决了在保护模式下，运行16位汇编编写的程序。

> 保护模式（内存属性的检测）由操作系统和CPU共同完成的（操纵系统定义那块内存可读可写，由CPU进行检测）。在32汇编编程时，就需要知道内存的哪些位置可读，哪些位置可写（不能直接告诉操作系统，可以先告诉编译器，然后由编译器将其记录到文件中，之后操作系统读取文件时就知道哪些地址可读、可写）。

# 分区

> 编写32位汇编时需要提前进行分区，32位汇编的文件格式：PE文件格式（记录分区的信息）。
>
> 微软官方伪指令相关文档

代码示例：

```
; 指明指令集，告诉编译器编译32位汇编程序（386位CPU的型号）
.386
```

```
; model --> 初始化程序内存模型，flat --> 平坦模式（不分段，不使用段寄存器，使用分区）
; stdcall --> 默认调用约定（编写函数的时候不在需要给调用约定）
.model flat, stdcall
OPTION CASEMAP: none ; 开启伪指令大小写敏感

; 栈区 --> 可以不初始化，编译器会默认生成4kb左右的空间，使用时就需要在链接时给出栈的大小
; 如果对栈的大小有要求，就需要定义对应空间大小的栈
.stack

; 常量区
.const
MY_MSG db "Hello World", 0
MY_TITLE db "标题", 0

; 初始化数据区
.data
  NUM1 db 1 ; 初始化为1，不得不将其保存到内存中，会多占用文件字节
  NUM1 db ? ; 初始化一个随机数，不会多占用文件大小

; 不初始化数据区，类似高级语言中的全局变量
.data?
  NUM2 db ? ; 默认初始化为0，不会多占用文件大小

; 函数声明
MessageBoxA proto hWnd:dword, lpText:dword, lpCaption :dword,
uType:dword
; 退出进程
ExitProcess proto uExitCode:dword

; 代码区
.code
START:
  ;push 0
  ;push 0
  ;push 0
  ;push 0
  ;call MessageBoxA

  invoke MessageBoxA, 0, offset MY_MSG, offset MY_TITLE, 0

  ;退出进程
  invoke ExitProcess, 0
  ret
end START
```

> 32位汇编程序可调用API，不在需要调用中断。调用API，需要使用 PROTO（原型函数或过程。可以使用 INVOKE 指令通过 PROTO 指令调用函数原型）进行函数声明（向前声明，API的函数声明需要自己声明），调用MessageBoxA API前，对其声明示例：
> MessageBoxA proto, hWnd:dword, lpText:dword, lpCaption :dword, uType:dword

汇编程序中还可以设置是否开启伪指令大小写字符敏感，通过使用 OPTION CASEMAP: none 告诉编译器开启大小写敏感（汇编指令关键字不区分大小写）。

## 编译链接

通过ml、link编译链接时，分别需要指定生成的文件格式（MZ：omf、PE：coff）以及子系统的类型（窗口、）。coff：停用对象文件格式，Microsoft实现叫做可移植可执行（PE）文件格式。

> 编译：ml /coff /c hello.asm（生成的 .obj 文件为32位）
> 链接：link /subsystem:windows hello.obj user32.lib kernel32.lib

注意：程序中使用了API，在链接时就需要添加链接选项（将user32.lib拷贝到程序的目录下并一同链接或者将库的路径添加到环境变量中），在汇编程序中 ret 不代表结束进程，所以就需要手动结束进程，调用API：ExitProcess （在kernel32.lib中）

## 支持添加头文件

- 出代码区外所有的代码都可以存放到 .inc 文件中，使用时在 .asm 文件中使用 include 包含头文件即可。
- API所需要的 .lib 文件也可以通过 includelib 进行声明（在 .asm 文件头文件下使用即可，链接时不在需要添加链接选项）。

示例代码如下：

```
include hello.inc ; hello.inc为头文件
includelib user32.lib
includelib kernel32.lib

; 代码区
.code
START:
  invoke MessageBoxA, 0, offset MY_MSG, offset MY_TITLE, 0

  ; 退出进程
  invoke ExitProcess, 0
  ret
end START
```

编译链接命令：

```
ml /coff /c hello.asm
link /subsystem:windows hello.obj
pause
```

# 寻址方式的差异

32位汇编：访问内存不在需要使用基址寄存器（基址寄存器可有任何寄存器取代），**增加比例因子寻址方式。**例如：

```
mov eax, [eax]
mov eax, [eax + ebx + 1]
mov eax, [esp] ; 访问堆栈
mov eax, [esp + 1]
; 增加比例因子寻址方式（指令的限制，最多乘以8（内部利用移位进行乘法运算））
mov eax, [ebx + eax * 2] ; ebx 数组首地址
mov eax, [ebx + eax * 4 + 2] ; ebx 数组首地址
mov eax, [esp + ebx * 8 + 4] ; ebx 数组首地址
```

# 新指令

16位的汇编指令在32位汇编中都支持：

```
mov eax, ebx
mov ax, bx
```

## pushad、popad

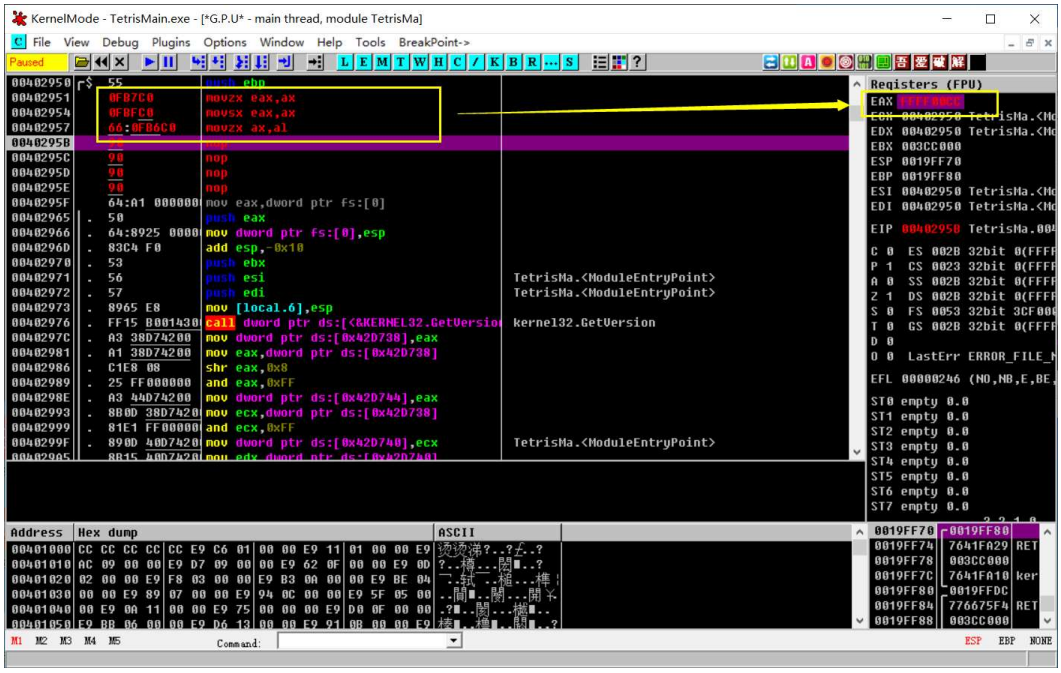
pushad：将所有通用寄存器依次全部入栈
popad：将所有通用寄存器依次从堆栈中弹出

## movzx、movsx

movzx：将一个16位数扩展为32位，高位补零（无符号）
movsx：将一个16位数扩展为32位，高位补符号位（有符号）

代码示例：

```
eax = 0019FFCC
movzx eax, ax ; eax = 0000FFCC
movsx eax, ax ; eax = FFFFFFCC
movzx ax, al ; eax = FFFF00CC
```




# 32位程序调试器

> Windbg：debug的改进版（win10上可在应用商店下载Windbg Preview）
> OllyDBG：TD的增强版
> x64dbg：支持调试32位、64位的程序

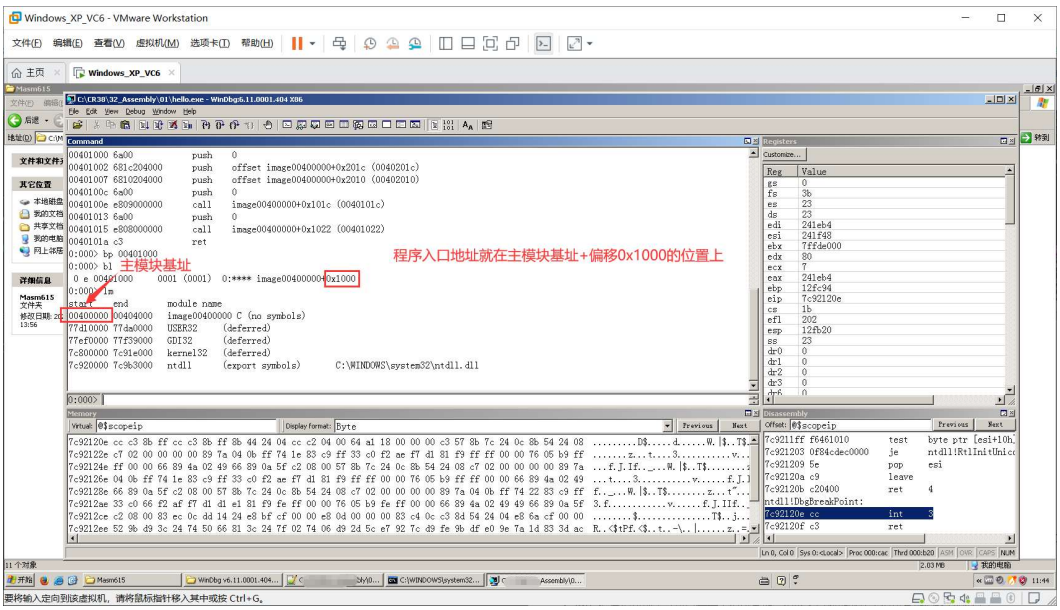
## Windbg

> 有32位、64位版本

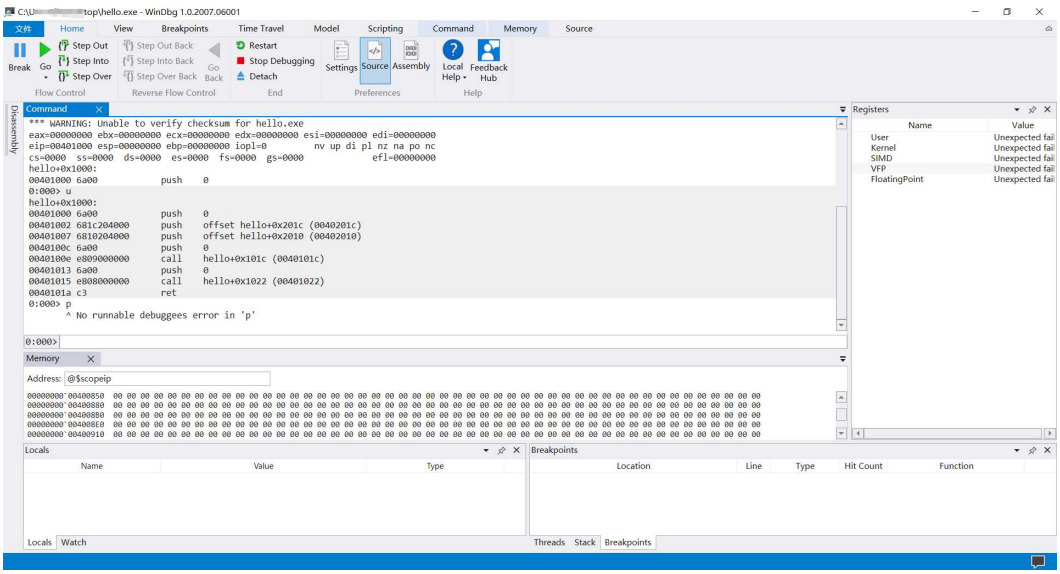
操作：

- g $entry：在程序入口处下断点（操作系统运行程序之前还会执行一些初始化的代码），没有反应可以试着输入：`u 00401000` （编译器将程序的入口地址固定放在这个地址上）。
- bp 00401000：在地址 00401000 处下断点。
- bl：查看已经下过断点的信息
- lm：遍历模块列表（可以查看主模块基址，程序入口地址就在主模块基址+偏移0x1000的位置）
- 汇编窗口支持快捷键调试

win_xp：




win_10：



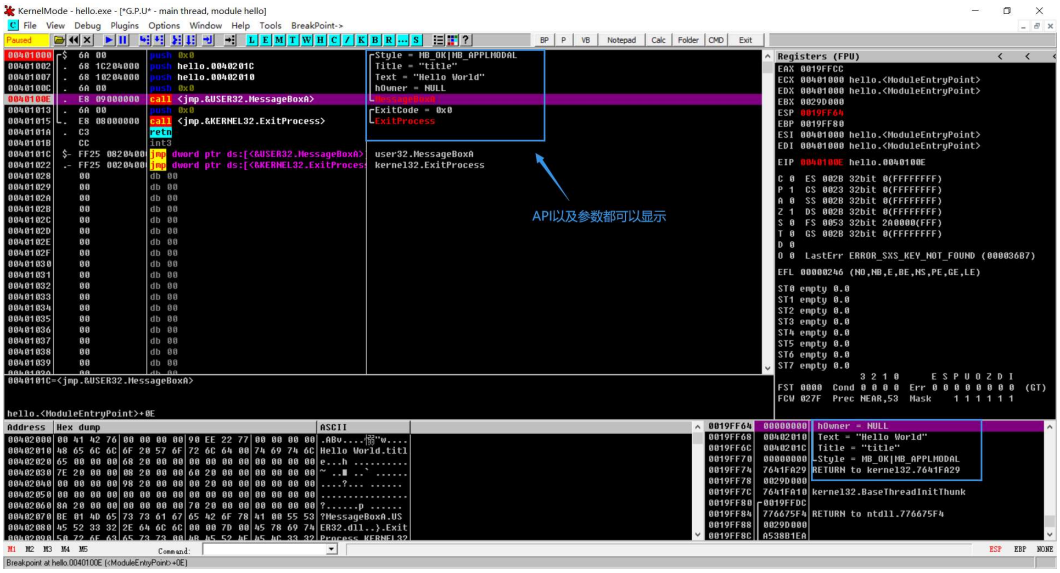
# OllyDBG

> 原版已经停止维护（存在一些bug），现在多用逆向爱好者维护的改进版本，只支持调试32位程序。

快捷键：

- Ctrl+G：搜索内存地址
- F2：下断点
- F9：运行

使用示例：




# x64dbg

> 开源，支持调试32位、64位程序。