汇编语言编程基础

汇编语言基本元素

整数常量

基数后缀（Radix）：h十六进制、q/o八进制、d十进制、b二进制、r编码实数

如果整数常量后面没有基数后缀，默认是十进制整数

以字母开头的十六进制常量前面必须加0

整数表达式

包含整数值和算数运算符的数学表达式，表达式的结果不能超过32bits的表示范围。

实数常量

十进制实数常量：由符号sign、整数、小数点、小数和指数组成
，至少要有一个数字和一个小数点。如：-1.11E-5、2.、+3.0、2.E5

编码（十六进制）实数：以十六进制数表示一个实数

字符常量

字符串常量

嵌套引号

保留字

- 指令助记符：MOV、ADD
- 伪指令：INCLUDE、PROC
- 属性：BYTE、WORD
- 预定义符号：$、？

标识符

标识符是程序员选择用来标识变量、常量、过程、代码的标号

- 包含1~247个字符
- 大小写不敏感（MASM默认）
- 第一个字符必须是字母、下划线、@、？或$

- 第一个字符不能是数字（对比十六进制整数）

伪指令

伪指令内嵌在汇编语言源代码中，由汇编器识别、执行相应动作的命令，用于定义变量、段、过程、汇编器选项等。伪指令在程序运行时并不执行。

DWORD伪指令告知汇编器要在程序中给一个双字变量保留空间。MOV指令在运行时真正执行，把myVar的内容复制到EAX寄存器：

```
myVar DWORD 26
mov   ear, myVar
```

- 定义段（Segment）：.data、.code、.stack
- 定义过程（Procedure）：PROC、ENDP
- 允许或禁止汇编器的某些特性： OPTION、.386、.MODEL

指令

一条汇编指令包括四个部分：标号、指令助记符（必需）、操作数（通常必需）、注释。

标号

标号是充当指令或数据位置标记的标识符。分为数据标号和代码标号。

数据标号

标识变量地址，位在代码中引用该变量提供方便。

```
count DWORD 100
```

相对.data数据段在内存中的偏移

OFFSET：获取数据标号的内存偏移地址

```
.data
str_hello BYTE "Hello World! ", 0
.code
mov eax, OFFSET str_hello
```

汇编器为每个标号分配一个数字地址。在一个标号中定义多个数据项是可以的。下例中，array标识了第一个数字（1024）的位置，其他在内存中相邻数字紧接其后。

```
array DWORD 1024,2048
      DWORD 4096,8912
```

代码标号

程序代码区中的标号必须以冒号结尾。代码标号通常用做跳转和循环指令的目标地址。

```
target:
        mov ax,bx
        ...
        jmp target
```

指令助记符

用以表示一条指令

- mov 将一个值移动到另一个，前者为目的操作数，后者为源操作数
- add
- sub 从一个值中减去一个值
- mul
- jmp
- call 调用一个过程

操作数

操作数是指令的操作对象，可以是寄存器、内存、常量、I/O端口

```
inc eax  ;eax加1
```

注释

单行注释：分号

块注释: COMMENT伪指令和用户定义的符号

```
COMMENT !
This is a comment
!
```

NOP指令

最安全的指令是NOP(nooperation)，一条NOP指令占用一个字节的存储，什么也不做。有时编译器或汇编器使用NOP指令把代码对齐到偶数地址边界。在下面的例子中，第一个MOV指令生成三个机器字节码，NOP指令将第三条指令的地址对齐到双字(4的倍数)边界上。

```
00000000 66 8B C3 mov ax,bx
00000003 90    nop   ;对齐下一条指令
00000004 8B D1 mov edx,ecx
```

IA-32处理器从偶数双字地址处加载代码和数据更加快速。

HELLOWORLD

```
.386
;表示程序使用的指令集，允许汇编80386处理器的非特权指令，禁用其后处理器引入的汇编指令。
.model flat, stdcall
;初始化程序的内存模式，使用平坦内存模式（4GB内存空间）并使用stdcall调用习惯，即API调用时右边的参数先入栈。
option casemap :none
;编译器程序中变量名和子程序名对大小写敏感。
include \masm32\include\windows.inc
include \masm32\include\kernel32.inc
include \masm32\include\masm32.inc
; include跟在其后的文件名所指定的文件在编译时将插入在该处。这三条语句得到函数的常量和声明。
includelib \masm32\lib\kernel32.lib
includelib \masm32\lib\masm32.lib
;链接库

.data   ;定义已初始化数据段的开始。
str_hello BYTE "Hello World!", 0
;byte定义字符串，命名为str_hello，0表示字符串的结尾。
.code   ;定义代码段的开始
start:    ;指令标号，标记指令地址
invoke StdOut, addr str_hello
;StdOut为masm32.inc中定义的函数，将内存数据输出到命令行窗口。addr用来把标号的地址传递给被调用的函数。
invoke ExitProcess, 0
;ExitProcess为kernel32.inc中定义的函数，退出程序执行
END start   ;标记模块的结束，指定程序的入口点是start
```

```
.386
.model flat, stdcall
option casemap :none
include \masm32\include\windows.inc
include \masm32\include\kernel32.inc
include \masm32\include\user32.inc
includelib \masm32\lib\kernel32.lib
includelib \masm32\lib\user32.lib
.data
str_hello BYTE "Hello World!", 0
.code
start:
invoke MessageBox, NULL, addr str_hello, addr str_hello, MB_OK
;调用MessageBox，用来弹出一个对话框，标题和显示的内容均为str_hello，包含一个确定按钮。
invoke ExitProcess, 0
END start
```

```
;命令行输出

\masm32\bin\ml /c /Zd /coff hello_console.asm

;用汇编程序（\masm32\bin\ml.exe）对hello_console.asm进行汇编，形成目标文件（.obj）。其中：/c是只汇编、不链接的指令，/Zd是在目标文件中生成行号信息，即目标文件指令与源代码中代码行的对应关系，/coff是生成microsoft公共目标文件格式的文件。

\masm32\bin\link /SUBSYSTEM:CONSOLE hello_console.obj

;用链接程序（\masm32\bin\link.exe）对hello_console.obj进行链接，形成可执行文件（.exe）。/SUBSYSTEM:CONSOLE是生成命令行程序的指令。

;窗口输出

“\masm32\bin\ml /c /Zd /coff hello_window.asm

;对hello_window.asm进行汇编，汇编生成hello_window.obj文件。其中：/c是只汇编、不链接的指令，/Zd是在目标文件中生成行号信息，即目标文件指令与源代码中代码行的对应关系，/coff是生成microsoft公共目标文件格式的文件。

\masm32\bin\Link /SUBSYSTEM:WINDOWS hello_window.obj

;用链接程序（\masm32\bin\link.exe）对hello_window.obj进行链接，形成可执行文件（.exe）。/SUBSYSTEM:WINDOWS是生成窗口程序的指令。
```