第 1 章　Windows 汇编语言程序设计基础

Windows 汇编语言程序分为控制台编程和图形界面编程两种，控制台编程相对简单一些。为了由浅入深，本书从控制台编程开始讲解。

读者总希望用最快的速度掌握书中的概貌，为此从一个最简单的程序开始。一些汇编语言语法也结合程序进行讲解，有些指令和语法用注解的方法说明。

1.1　第一个完整的 Windows 汇编语言程序

Windows 汇编语言程序有自己的编程规范，它的编程规范比 Visual C 要简单得多，调试也很方便。更重要的是系统把重要的东西都呈现给读者，使读者更能掌握其中的本质。

用一条一条的汇编语言指令很难写出大程序，Windows 汇编语言程序也是调用系统提供的 API 来写程序。因而，用 Windows 汇编语言同样可写出大程序。以下是一个最简单的 Windows 程序。

```
;程序功能：显示一个信息框。
;ex1.asm(e:\masm\base)      ;程序名
;编译链接方法：
;ml /c /coff  ex1.asm
;link /subsystem:console ex1.obj
    .386                ;指明指令集
    .model flat,stdcall ;程序工作模式，flat为Windows程序使用的模式(代码和数据
                        ;使用同一个4GB段),stdcall为API调用时右边的参数先入栈
    option casemap:none ;指明大小写敏感

include windows.inc
include user32.inc
includelib user32.lib
include kernel32.inc
includelib kernel32.lib

        .data           ;数据段
szCaption   db  '抬头串',0
szText      db  'Hello! ',0

        .code           ;代码段
start:
```

```
    invoke MessageBox,      ;显示信息框
        NULL,               ;父窗口句柄
        offset szText,      ;正文串的地址
        offset szCaption,   ;抬头串的地址
        MB_OK               ;按钮

    invoke ExitProcess,     ;终止一个进程
        NULL                ;退出代码

    end start               ;指明程序入口点
```

图 1-1

程序运行结果见图 1-1。

说明：程序调用了两个 Windows 提供的 API。invoke 是汇编语言中的伪指令，该指令的使用方法见 1.4 节。

1.2 编译、链接和运行

1.2.1 创建编译链接环境

（1）安装 MASM615 调试工具。

（2）建立一个 VAR.BAT 文件，内容如下。

```
@echo off
rem 请根据 Masm32 软件包的安装目录修改下面的 Masm32Dir 环境变量!
set Masm32Dir=c:\Masm32
set include=%Masm32Dir%\Include;
c:\Program Files\Microsoft Visual Studio\VC98\Include;(本行应接在上行后)
Program Files\Microsoft Visual Studio\VC98\MFC\Include;(本行应接在上行后)
%include%(本行应接在上行后)
set lib=%Masm32Dir%\lib;%lib%
set w2k=%Masm32Dir%\Incluse\w2k;%Include\w2k%
set path=%Masm32Dir%\Bin;%Masm32Dir%\Include;%Masm32Dir%\Include\w2k;
%Masm32Dir%\lib;%Masm32Dir%;%PATH%(本行应接在上行后)
set Masm32Dir=
echo on
```

编译链接程序前，需要切换到命令提示符方式，并运行该文件（设置好环境），然后方可进行编译链接。

1.2.2 编译链接和运行

以下以编译链接 ex1.asm 为例：

（1）编译。

```
ML /Zi /c /Fl /coff ex1.asm
```

ML 参数说明（注意参数大小写）：

```
/Zi -- 加符号调试信息
/c -- 连接前的编译
/Fl -- F1[file]产生列表文件
/coff -- 产生COFF格式目标文件
```

编译的更多参数说明，可用命令 ML /?查阅。

（2）链接。

```
LINK /SUBSYSTEM:console ex1.obj
```

其中 console 指明是控制台编程，如果是 Windows 窗口编程，则将 console 改为 Windows。

（3）运行。

在 Windows 下双击 ex1.exe 或在 DOS 命令提示符下键入 ex1 回车。

1.2.3　建立编译链接批命令文件

可以把编译链接过程写成批命令文件，以减少键盘输入量。例如：

```
MLEXE.BAT
ML /Zi /c /Fl /coff %1.asm
LINK /subsystem:console %1.obj
del %1.obj
dir %1.*
```

如果要编译链接 ex1.asm，则只需输入：

```
MLEXE ex1   回车
```

1.3　将 Windows 汇编语言程序反汇编后的程序原形

将可执行程序用 IDA 反汇编工具反汇编后，程序的代码部分可直接使用，程序的其他部分稍作修改后，即可再编译链接成可执行程序。具体修改部分见程序尾的说明。

```
;iex1.asm,本程序为ex1.exe反汇编后的程序。
;iex1.asm(e:\masm\base)
;编译链接方法:
;ml /c /coff iex1.asm
;link /subsystem:Windows iex1.obj
    .386
    .model flat,stdcall
```

```
    option casemap:none
    .data
include windows.inc
include user32.inc
includelib user32.lib
include kernel32.inc
includelib kernel32.lib

Caption     db '抬头串',0
Text        db 'Hello!',0
    .code
        public start
start       proc near
        push    0               ;uType
        push    offset Caption  ;"抬头串"
        push    offset Text     ;"Hello!"
        push    0               ;hWnd
        call    MessageBoxA
        push    0               ;uExitCode
        call    ExitProcess
start       endp
        end start
```

说明：API 调用时右边的参数先入栈。用反汇编工具 IAD 反汇编后，保留代码段不变，将数据段中的数据搬入代码段，将其余部分删除，再加入包含文件和程序中的前 4 条指令，即可再编译链接成可执行程序。

将可执行文件反汇编成汇编语言程序，经过适当修改后，再编译链接成可执行文件，这是十分有意义的。

1.4 invoke 伪指令的使用格式、变量及数据段 data 和 data? 的区别

1.4.1 invoke 伪指令的使用格式

invoke 伪指令的使用格式为：

```
invoke 函数名[,参数1][,参数2]...
```

参数的个数不定，可以没有，也可以有多个。如果 invoke 与某个函数的参数个数不匹配（少或多），则编译时报错。如果参数个数少，则报错“error A2137:too few arguments to INVOKE”；如果参数个数多，则报错“error A2137:too many arguments to INVOKE”。

1.4.2　变量

1．变量的命名规则

变量由大写字母 A，B，…，Z，小写字母 a，b，…，z，数字 0，1，2，…，9，下划线，符号@、$和?组成，且变量的第一个符号不能是数字。变量的长度不能超过 240 个字符，不能使用指令名关键字，在同一个作用域内不能重名。应该养成良好的命名习惯，如表 1-1 所示。

表　1-1

缩　写	含　义	缩　写	含　义
sz	表示以 0 结尾的字符串（ASCIIZ）	lp	表示指针 long point
h	表示句柄 handle	lpsz	表示指向 ASCIIZ 的指针
b	表示字节 byte	f	表示浮点数 float
w	表示字 word	st	表示结构体 struct
dw	表示双字 double word		

例如：

```
hWin          表示窗口句柄
lpArray       表示指向数组的指针
szString      以0结尾的字符串
stWndClass    WNDCLASS结构
bNumber       以字节定义的数
dwNumber      以双字定义的数
wNumber       以字定义的数
```

2．全局变量

全局变量的作用域为整个程序。在　.data 和　.data？段内定义的变量为全局变量。全局变量的定义格式为：

```
变量名 类型 初始值
变量名 类型 重复数量 dup(初始值)
```

例如：

```
count dw 0
array db 10 dup(0)
```

3．局部变量

局部变量的作用域为一个程序内。局部变量的定义格式为：

```
local 变量名1[重复数量][：类型],变量名2[重复数量][：类型]...
```

局部变量要放在子程序的开始位置，并且没有初始值。例如：

```
        .386
```

```
        .model flat, stdcall
        option casemap :none
include     windows.inc
include     kernel32.inc
includelib  kernel32.lib
    .code
SubProc proc,x:byte,y:byte

    local a:byte            ;定义局部变量
    local b:byte            ;定义局部变量

    mov al,x
    mov a,al
    mov al,y
    mov b,al
    ret
SubProc endp
main    proc
    invoke SubProc,1,2      ;调用子程序（右边参数先入栈）
    invoke ExitProcess,0    ;退出进程
main    endp
    end main
```

4．局部变量在栈中的位置

将以上程序用 IDA 反汇编后的程序如下：

```
sub_401000  proc near

var_2       = byte ptr -2
var_1       = byte ptr -1
arg_0       = byte ptr  8
arg_4       = byte ptr  0Ch

        push    ebp
        mov ebp, esp
        add esp, 0FFFFFFFCh     ;(-4)的补码=0FFFFFFFCh
        mov al, [ebp+arg_0]     ;x=1
        mov [ebp+var_1], al
        mov al, [ebp+arg_4]     ;y=2
        mov [ebp+var_2], al
        leave
        retn    8               ;将入口参数（x,y)退栈
sub_401000  endp
        public start
start   proc near
        push    2
        push    1
        call    sub_401000
```

```
        push    0
        call    $+5
        jmp ds:ExitProcess
start   endp
```

说明：参数在栈中的位置见图 1-2。

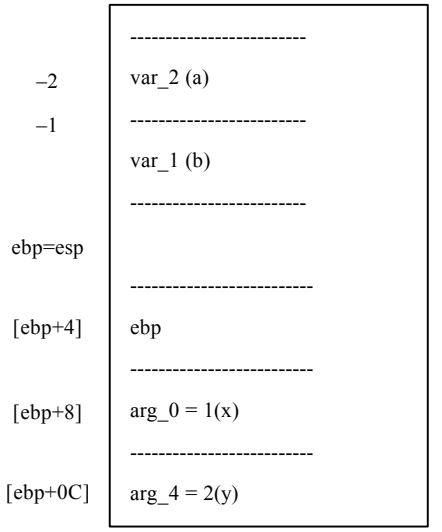

图 1-2　栈区示意图

1.4.3　数据段 data 和 data?的区别

程序的一般结构为：

```
.data
定义变量并初始化（有初始值）
.data?
定义变量（变量的初始值为'? '）
.const
定义常量
.code
```

定义在　.data？段中的变量的初始值只能是'?'。定义在　.data？段中的变量不占用磁盘空间，即不增加　.exe 文件的大小。这是用　.data？段的优点。例如，在　.data？中定义变量：

```
.data?
sum dw  ?
array 10 db dup(?)
```

在实际应用中，上述变量的初始值为 0。

汇编语言中的类型如表 1-2 所示。

表 1-2

缩　写	全 名 写 法	名称和字节数
db	byte	字节（1 字节）
dw	word	字（2 字节）
dd	dword	双字（4 字节）
df	fword	三字（6 字节）
dq	qword	四字（8 字节）
dt	tbyte	10 字节 BCD 码
	sbyte	有符号字节（1 字节）
	sword	有符号字（2 字节）
	sdword	有符号双字（4 字节）
	real4	单精度实型数（4 字节）
	real8	双精度实型数（8 字节）
	real10	10 字节实型数（10 字节）

1.4.4　高级语法 while-endw 的使用

while-endw 的格式为：

```
.while(条件)
  循环体（条件满足时执行）
.endw
```

```
;ex2.asm(e:\masm\base) 高级语法while-endw的使用示例。
;程序功能：用while循环给字节数组赋值并计算。
    .386
    .model flat, stdcall
    option casemap :none
 include    windows.inc
 include    kernel32.inc
 includelib kernel32.lib
 include    user32.inc
 includelib user32.lib

    .data
 a      db  10 dup(0)        ;定义字节数组
 buffer db  10 dup(0)
 CapMsg db  '输出',0
 szFmt  db  '结果是:%d',0
 i      db  0
 sum    db  0
```

```
    .code
start:
    mov edi,0
    .while(i<10)
      mov al,i
      mov a[edi],al
      inc i
      inc edi
    .endw

    mov i,0
    mov edi,0
    .while(i<10)
     mov al,a[edi]
     add sum,al
     inc i
     inc edi
    .endw

    xor eax,eax
    mov al,sum
    invoke  wsprintf,       ;格式化信息串
        addr buffer,        ;信息串格式化后的存放地
        addr szFmt,         ;信息串的格式: "结果是:%d",0
        eax                 ;二进制数
    invoke  MessageBox,     ;显示信息框
        NULL,               ;父窗口句柄
        offset buffer,      ;正文串 "结果是:%d",0
        offset CapMsg,      ;抬头串 "输出",0
        MB_OK               ;按钮"确定"

    INVOKE ExitProcess,0    ;结束进程
    end start
```

图　1-3

运行结果见图 1-3。

1.4.5　高级语法 repeat-until 的使用

repeat-until 的格式为：

```
    .repeat
      循环体（条件不满足时执行）
    .until(条件)
```

```
;ex3.asm(e:\masm\base) 高级语法repeat-until的使用。
```

```
;程序功能：用repeat循环给双字数组赋值并计算。
    .386
    .model flat, stdcall
    option casemap :none
include    windows.inc
include    kernel32.inc
includelib kernel32.lib
include    user32.inc
includelib user32.lib

    .data
a       dd  10 dup(0)   ;定义双字数组
buffer  db  10 dup(0)
CapMsg  db  '输出',0
szFmt   db  '结果是:%d',0
i       dd  0
sum     dd  0

    .code
start:
    mov edi,0
    .repeat
      mov eax,i
      mov a[edi],eax
      inc i
      add edi,4
    .until (i>=10)

    mov i,0
    mov edi,0
    .repeat
      mov eax,a[edi]
      add sum,eax
      inc i
      add edi,4
    .until (i>=10)

    mov eax,sum
    invoke  wsprintf,       ;格式化信息串
        addr buffer,        ;信息串格式化后的存放地
        addr szFmt,         ;信息串的格式："结果是:%d",0
        eax                 ;二进制数
    invoke  MessageBox,     ;显示信息框
```

```
        NULL,               ;父窗口句柄
        offset buffer,      ;正文串 "结果是:%d",0
        offset CapMsg,      ;抬头串 "输出",0
        MB_OK               ;按钮"确定"

    INVOKE ExitProcess,0    ;结束进程
    end start
```

运行结果见图 1-3。

1.4.6　高级语法 if-elseif-endif 的使用

if-elseif-endif 的格式为：

```
.if 条件1
  指令（条件1满足时执行）
[.elseif 条件2]
       指令(条件2满足时执行)
    [.elseif 条件3]
          指令(条件3满足时执行)
        M
        endif
```

```
;ex4.asm(e:\masm\base) 高级语法if-elseif-endif的使用。
;程序功能:输出"number是正数"或"number是负数"。
    .386
    .model flat, stdcall
    option casemap :none
 include    windows.inc
 include    kernel32.inc
 includelib kernel32.lib
 include    user32.inc
 includelib user32.lib

    .data
 CapMsg  db '输出',0
 szFmt   db 'number是正数:%d',0
 szFmt2  db 'number是负数:%d',0
 number  dd -9
 buffer  db     80 dup(0)
    .code
 start:
    mov eax,number
    invoke  wsprintf,       ;格式化信息串
```

```
        addr buffer,          ;信息串格式化后的存放地
        addr szFmt,           ;'number是正数',0
        eax                   ;二进制数

    mov eax,number
    .if eax == -9
       invoke  wsprintf,      ;格式化信息串
           addr buffer,       ;信息串格式化后的存放地
           addr szFmt2,       ;'number是负数',0
           eax                ;二进制数
    .endif

    invoke  MessageBox,       ;显示信息框
        NULL,                 ;父窗口句柄
        offset buffer,        ;正文串 "结果是:%d",0
        offset CapMsg,        ;抬头串 "输出",0
        MB_OK                 ;按钮"确定"

    INVOKE ExitProcess,0      ; 结束进程
    end start
```

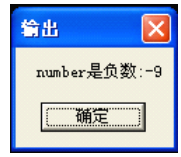

图 1-4

运行结果见图 1-4。

1.4.7 条件运算符

汇编语言中的条件运算符和高级语言的条件运算符类似，这给编写程序带来了极大的方便。条件运算符如表 1-3 所示。

表 1-3

条件运算符	功　能	说　明
==	等于	变量和操作数之间的比较
!=	不等于	变量和操作数之间的比较
<	小于	变量和操作数之间的比较
>	大于	变量和操作数之间的比较
<=	小于等于	变量和操作数之间的比较
>=	大于等于	变量和操作数之间的比较
&&	逻辑与	两个表达式的结果
||	逻辑或	两个表达式的结果
&	按位与	变量和操作数的结果
!	取反	变量或表达式的结果

条件运算时，非零为真，零为假。条件运算时，要注意表达式的左边应为寄存器或变量，不能为常数，表达式两边可以同时为寄存器，但不能同时为变量。

标志测试伪指令：

```
    CARRY?       表示进位标志是置位
    ZERO?        表示零标志是置位
    SIGN?        表示符号标志是置位
    PARITY?      表示奇偶标志是置位
    OVERFLOW?    表示溢出标志是置位
```

```
;ex5.asm(e:\masm\base) 条件运算符的使用。
    .386
    .model flat, stdcall
    option casemap :none
 include    windows.inc
 include    kernel32.inc
 includelib kernel32.lib
 include    user32.inc
 includelib user32.lib

    .data
 CapMsg  db '输出',0
 szFmt   db 'eax=%d(eax==1满足), ebx=%d（置进位标志）, edx=%d（edx为零满足）',0
 buffer  db  80 dup(0)

    .code
start:
    mov eax,1
    .if eax==1
      mov eax,2
    .endif

    mov ebx,10
    stc                 ;置进位标志
    .if CARRY?          ;进位标志是置位?
      mov ebx,20
    .endif

    xor edx,edx         ;使零标志置位(为1)
    .if !ZERO?          ;零标志不是置位?
      mov edx,8
    .endif

    invoke  wsprintf,   ;格式化信息串
        addr buffer,    ;信息串格式化后的存放地
```

```
        addr szFmt,      ;格式化串
        eax,             ;数
        ebx,
        edx

    invoke  MessageBox,      ;显示信息框
        NULL,                ;父窗口句柄
        offset buffer,       ;正文串
        offset CapMsg,       ;抬头串 "输出",0
        MB_OK                ;按钮"确定"

    INVOKE ExitProcess,0     ;结束进程
    end start
```

运行结果见图 1-5。

图 1-5

1.4.8 高级语法 continue 的使用

continue 用在循环 while-endw 和 repeat-until 中，它的功能是结束本次循环（continue 语句后的语句不执行），进入下一次循环。continue 的语法为：

```
    .continue
```

```
;ex6.asm (e:\masm\base)高级语法 continue 的使用。
;程序功能：计算1+3+5+...+99的和。
    .386
    .model flat, stdcall
    option casemap :none
 include    windows.inc
 include    kernel32.inc
 includelib kernel32.lib
 include    user32.inc
 includelib user32.lib

    .data
```

```
array  dw  100 dup(0)      ;定义字数组
buffer db  80 dup(0)
CapMsg db  '输出',0
szFmt  db  '结果是:%d',0
i      dw  1
sum    dd  0

   .code
start:
   mov edi,0
   .while(i<=100)
     mov ax,i
     mov array[edi],ax     ;给数组赋值
     inc i                 ;计数
     add edi,2             ;指向下一个
   .endw

   mov i,1
   mov edi,0
   .while(i<=100)
     movzx eax,word ptr array[edi]
     inc i                 ;计数
     add edi,2             ;指向下一个
     mov edx,eax
     ror edx,1             ;循环右移一位
     .if !CARRY?           ;进位标志没有置位?(eax是偶数)
       .continue           ;结束本次循环（此语句后的语句不执行）
     .endif
     add sum,eax           ;奇数求和
   .endw

   invoke  wsprintf,       ;格式化信息串
           addr buffer,    ;信息串格式化后的存放地
           addr szFmt,     ;信息串的格式: "结果是:%d",0
           sum             ;二进制数
   invoke  MessageBox,     ;显示信息框
           NULL,           ;父窗口句柄
           offset buffer,  ;正文串 "结果是:%d",0
           offset CapMsg,  ;抬头串 "输出",0
           MB_OK           ;按钮"确定"

   INVOKE ExitProcess,0    ;结束进程
   end start
```

运行结果见图 1-6。

图 1-6

1.4.9 高级语法 break if 的使用

break if 语句用在循环 while-endw 和 repeat-until 中，其功能是退出循环。

break if 语句的语法是：

```
.break .if退出条件
```

```
;ex7.asm(e:\masm\base) 高级语法 break if 的使用。
;程序功能：计算1+2+3+...+50的和。
    .386
    .model flat, stdcall
    option casemap :none
 include    windows.inc
 include    kernel32.inc
 includelib kernel32.lib
 include    user32.inc
 includelib user32.lib

    .data
 array  dw  100 dup(0)  ;定义字数组
 buffer db  80 dup(0)
 CapMsg db  '输出',0
 szFmt  db  '结果是:%d',0
 i      dw  1
 sum    dd  0

    .code
 start:
    mov edi,0
    .while(i<=100)
      mov ax,i
      mov array[edi],ax ;给数组赋值
      inc i             ;计数
      add edi,2         ;指向下一个
    .endw

    mov i,1
    mov edi,0
    .while(i<=100)
      movzx eax,word ptr array[edi]
      .break .if eax==51
      add sum,eax       ;求和
      inc i             ;计数
      add edi,2         ;指向下一个
```

```
    .endw

    invoke  wsprintf,    ;格式化信息串
        addr buffer,     ;信息串格式化后的存放地
        addr szFmt,      ;信息串的格式："结果是:%d",0
        sum              ;二进制数
    invoke  MessageBox,  ;显示信息框
        NULL,            ;父窗口句柄
        offset buffer,   ;正文串 '结果是:%d",0
        offset CapMsg,   ;抬头串 "输出",0
        MB_OK            ;按钮"确定"

    INVOKE ExitProcess,0    ;结束进程
    end start
```

运行结果见图 1-7。

图　1-7

1.4.10　结构体

汇编语言中的结构体和共用体与 C 语言中的基本类似。
结构体的定义格式如下：

```
名字 struct
    成员列表
名字 ends
```

例如（windows.inc 文件中包含以下结构）：

```
    COORD STRUCT
        x  WORD   ?
        y  WORD   ?
    COORD ENDS

SYSTEMTIME STRUCT
  WYear             WORD      ?      ;年(4位)
  WMonth            WORD      ?      ;月(1~12)
  WDayOfWeek        WORD      ?      ;星期（0~6） 0=星期天，1=星期一...
  wDay              WORD      ?      ;日(1~31)
  wHour             WORD      ?      ;时(0~23)
  wMinute           WORD      ?      ;分(0~59)
  wSecond           WORD      ?      ;秒(0~59)
  wMilliseconds     WORD      ?      ;毫秒(0~999)
SYSTEMTIME ENDS

 ;ex8.asm(e:\masm\base) 结构体应用示例。
    .386
```

```
    .model flat, stdcall
    option casemap :none

  include   windows.inc
  include   kernel32.inc
  includelib    kernel32.lib
  include       user32.inc
  includelib    user32.lib

    .data
;COORD STRUCT              ;windows.inc 文件中有定义
;  x  WORD   ?
;  y  WORD   ?
;COORD ENDS

 buffer db  80 dup(0)
 CapMsg db  '输出',0
 szFmt  db  'stPos.x 的和 = %d, stPos.y 的和 = %d',0

 stPos1 COORD <4,8>         ;定义结构体变量并初始化
 stPos2 COORD <>            ;定义结构体变量（取结构体原初始值）
 stPos3 COORD {2,4}         ;定义结构体变量并初始化
 stPos4 COORD {,40}         ;定义结构体变量并初始化

 stPos COORD 10 dup(<0,0>)  ;定义结构体数组

    .code
 start:
    mov stPos2.x, 10        ;域的使用
    mov stPos2.y, 20

    mov ecx,10
    mov edi,offset stPos    ;edi指向结构体变量
    mov eax,1
    mov ebx,11
@@:
    mov (COORD ptr[edi]).x, ax
    mov (COORD ptr[edi]).y, bx
    add edi,type COORD      ;结构体的大小
    inc ax
    inc bx
    loop short @B

    xor eax,eax
    xor ebx,ebx
```

```
    mov ecx,10
    mov edi,offset stPos    ;edi指向结构体变量
@@:
    add ax,(COORD ptr[edi]).x
    add bx,(COORD ptr[edi]).y
    add edi,type COORD      ;结构体的大小
    loop short @B

    invoke  wsprintf,       ;格式化信息串
            addr buffer,    ;信息串格式化后的存放地
            addr szFmt,     ;信息串的格式
            eax,
            ebx

    invoke  MessageBox,     ;显示信息框
            NULL,           ;父窗口句柄
            offset buffer,  ;正文串
            offset CapMsg,  ;抬头串
            MB_OK           ;按钮"确定"

     INVOKE ExitProcess,0   ;结束进程
    end start
```

图　1-8

运行结果见图 1-8。

```
;ex9.asm(e:\masm\base) 显示系统时间。
    .386
    .model flat, stdcall
    option casemap :none
 include    windows.inc
 include    kernel32.inc
 includelib kernel32.lib
 include    user32.inc
 includelib user32.lib

    .data
sysTime SYSTEMTIME <>       ;定义结构体变量（结构体由系统定义）

CapMsg  db  '系统当前时间',0
szFmt   db  '%d年%2d月%2d日',0Dh,0Ah,0Dh,0Ah
        db  '%2d:%2d:%2d',0
buffer  db  80 dup(0)

        .code
start:
    invoke GetLocalTime,offset sysTime      ;获取系统时间（从结构体返回）
```

```
movzx esi,sysTime.wYear     ;年(4位)
movzx edi,sysTime.wMonth    ;月(1~12)
movzx edx,sysTime.wDay      ;日(1~31)
movzx eax,sysTime.wHour     ;时(0~23)
movzx ebx,sysTime.wMinute   ;分(0~59)
movzx ecx,sysTime.wSecond   ;秒(0~59)

invoke  wsprintf,           ;格式化信息串
        addr buffer,        ;信息串格式化后的存放地
        addr szFmt,         ;格式化串
        esi,
        edi,
        edx,
        eax,
        ebx,
        ecx

invoke  MessageBox,         ;显示信息框
        NULL,               ;父窗口句柄
        offset buffer,      ;正文串
        offset CapMsg,      ;抬头串
        MB_ICONQUESTION     ;显示问号图标

invoke ExitProcess,0        ;结束进程
end start
```

图 1-9

运行结果见图 1-9。

1.4.11 语句的不同书写方法

```
;ex10.asm(e:\masm\ex1\ex10)
;-----------------------------------
;del %1.exe
;ml /c /coff %1.asm
;link /subsystem:windows %1.obj
;del %1.obj
;dir %1.*
 ;----------------------------------
.386
.model flat, stdcall
option casemap :none

include windows.inc
include user32.inc
```

```
  include kernel32.inc

  includelib user32.lib
  includelib kernel32.lib
.code
```