汇编语言与逆向技术

第5章 控制流转移

本章知识点

- 程序动态调试
- 控制流转移指令
  - 重点：CMP、条件跳转、LOOP
- 过程的定义和使用
  - 重点：PROC、ENDP、CALL、RET
- 调用链接库中的函数
  - 难点：PROTO伪指令

控制流转移指令

主观题 10分

设置

有几种CPU控制流转移的方式?

作答

控制转移

- 控制转移（transfer of control）是一种改变汇编语句执行顺序的方法。
  - 无条件转移
  - 条件转移

主观题 10分

设置

CPU的控制流跳转是如何实现的?

作答

南開大學 Nankai University

无条件转移 Unconditional Jump

- 将CPU控制权直接转移到指定的汇编语句
  - 修改EIP为指定的内存地址
  - CPU从EIP指定的内存地址读取下一条机器指令

JMP指令

- JMP 目的地址
- JMP指令实现CPU控制权的无条件跳转
- 目的地址是代码标号
  - 代码标号被汇编器翻译成内存地址
  - CPU看到的是内存地址，不是代码标号

循环

top：

… …

… …

JMP top

无限循环

主观题 10分

设置

条件跳转指令的判断条件存储在哪里?

作答

条件跳转指令 Conditional Jump

- 布尔和比较指令
  - AND、OR、XOR、NOT、TEST、CMP

AND指令

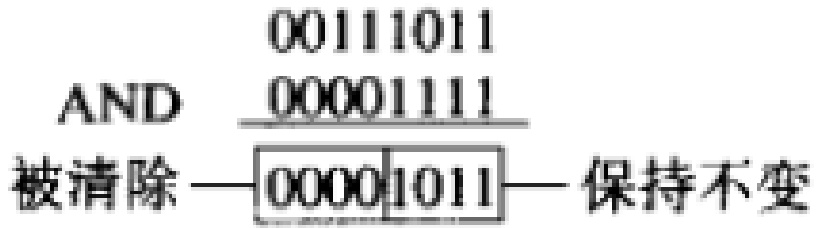
- 按位“与”操作，结果存放在目的操作数中

AND reg, reg
AND reg, mem
AND reg, imm
AND mem, reg
AND mem, imm

常被用于清除选定位，保留其他位

AND指令

常被用于清除选定位，保留其他位




```
mov al,00111011b
and al,00001111b
```

OR指令

- 按位“或” 操作，结果存放在目的操作数中

OR reg, reg

OR reg, mem

OR reg, imm

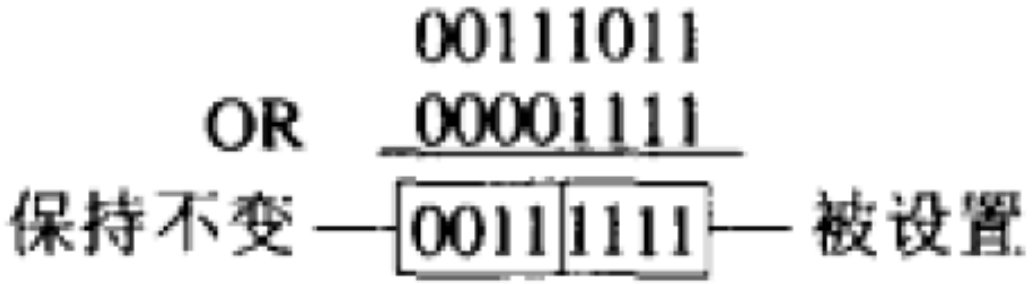
OR mem, reg

OR mem, imm

常被用于设置选定位，保留其他位

OR指令

常被用于设定选定位，保留其他位



填空题 0分

设置

此题未设置答案，请点击右侧设置按钮

```
MOV   EAX, 35H
AND   EAX, 53H
OR    EAX, 35H
```

的结果?

作答

XOR指令

- 按位“异或” 操作，结果存放在目的操作数中。
  - 如果位相同，结果为0；否则为1。
  - 与同样操作数执行两次XOR运算后，其值保持不变。
  - 用于检查奇偶标志

NOT指令

- 单操作数，按位“取反” 操作，结果存放在该操作数中。

NOT reg

NOT mem

填空题 0分

设置

此题未设置答案，请点击右侧设置按钮

```
MOV   EAX, 35H
XOR   EAX, 53H
NOT    EAX, 35H
```

的结果?

作答

TEST指令

- 按位“与”操作，但结果不会存放在目的操作数中（与AND的区别），只会设置相应的标志位。
  - 检查某些位是否为1

```
0 0 1 0 0 1 0 1   <- 输入值
0 0 0 0 1 0 0 1   <- 测试值
0 0 0 0 0 0 0 1   <- 结果：ZF = 0

0 0 1 0 0 1 0 0   <- 输入值
0 0 0 0 1 0 0 1   <- 测试值
0 0 0 0 0 0 0 0   <- 结果：ZF = 1
```

CMP指令

- CMP指令，比较目的操作数和源操作数
  - CMP reg，reg
  - CMP reg，imm
  - CMP mem，reg
  - CMP mem，imm
  - CMP reg，mem

CMP指令

- 执行从源操作数中减掉目的操作数的减法操作
  - 用于条件跳转指令的条件判断
  - 不改变目的操作数和源操作数，只影响eflags的标志位
- 设置相应的标志位
- 标志位：OF、SF、ZF、AF、PF、CF

```
    MOV EAX，100h
    MOV EBX，200h
    CMP EAX, EBX
    JA L1
    INVOKE StdOut，ADDR str1
    JMP L2
L1：
    INVOKE StdOut，ADDR str2 ;
L2：
    INVOKE ExitProcess，0
```

设置标志位技巧

- 设置和清除零标志位ZF

```
test al,0                ; 设置零标志
and  al,0                ; 设置零标志
or   al,1                ; 清除零标志
```

设置符号标志技巧

- 设置和清除符号标志位SF

```
or   al,80h          ;  设置符号标志
and  al,7Fh          ;  清除符号标志
```

设置进位标志技巧

- 设置和清除进位标志位CF

```
STC     ;设置进位标志

CLC     ;清除进位标志
```

设置溢出标志技巧

- 设置和清除溢出标志位OF

```
mov al,7Fh          ; AL = +127
inc al              ; AL = 80h (-128), OF = 1
or eax,0            ; 清除溢出标志
```

条件跳转指令Conditional Jump

- 条件指令j<condition>，通过条件判断来修改CPU控制流

有符号数的条件跳转指令
Conditional jump instructions used on signed data

Instruction	Description	Flags tested
JE/JZ	Jump Equal or Jump Zero	ZF
JNE/JNZ	Jump not Equal or Jump Not Zero	ZF
JG/JNLE	Jump Greater or Jump Not Less/Equal	OF, SF, ZF
JGE/JNL	Jump Greater/Equal or Jump Not Less	OF, SF
JL/JNGE	Jump Less or Jump Not Greater/Equal	OF, SF
JLE/JNG	Jump Less/Equal or Jump Not Greater	OF, SF, ZF

条件跳转指令 Conditional Jump

无符号数的条件跳转指令
Conditional jump instructions used on unsigned data

Instruction	Description	Flags tested
JE/JZ	Jump Equal or Jump Zero	ZF
JNE/JNZ	Jump not Equal or Jump Not Zero	ZF
JA/JNBE	Jump Above or Jump Not Below/Equal	CF, ZF
JAE/JNB	Jump Above/Equal or Jump Not Below	CF
JB/JNAE	Jump Below or Jump Not Above/Equal	CF
JBE/JNA	Jump Below/Equal or Jump Not Above	AF, CF

条件跳转指令 Conditional Jump

标志位和特殊用途的条件跳转指令

conditional jump instructions have special uses and check the value of flags

Instruction	Description	Flags tested
JXCZ	Jump if CX is Zero	none
JC	Jump If Carry	CF
JNC	Jump If No Carry	CF
JO	Jump If Overflow	OF
JNO	Jump If No Overflow	OF
JP/JPE	Jump Parity or Jump Parity Even	PF
JNP/JPO	Jump No Parity or Jump Parity Odd	PF
JS	Jump Sign (negative value)	SF
JNS	Jump No Sign (positive value)	SF

主观题 10分

设置

如何进行跳转条件的判断?

作答

LOOP指令

- LOOP 目的地址
- LOOP指令可以指定循环执行的次数 （loop count）
  - ECX寄存器作为循环计数器
  - LOOP指令执行时，ECX减1
  - 如果ECX不等于0，跳转到目的地址
  - 如果ECX等于0，不跳转，顺序执行

LOOP指令

```
MOV EAX 10h
MOV ECX 10h
L1：
    INC EAX
    LOOP L1
```

填空题 2分

设置

```
MOV EAX 10h
MOV ECX 10h
L1:
    INC EAX
    LOOP L1
```

LOOP循环结束后，EAX寄存器的值为 [填空1]

正常使用填空题需3.0以上版本雨课堂

作答

南開大學 Nankai University

LOOP指令

- LOOP指令先ecx减1，然后判断ecx是否为0.
- LOOP is exactly like dec ecx / jnz

填空题 10分

设置

```
MOV EAX 10h
MOV ECX 0
L1：
    INC EAX
    LOOP L1
```

如果ECX的初始值为0，LOOP会循环执行 [填空1] 次。

正常使用填空题需3.0以上版本雨课堂

作答

南開大學 Nankai University

循环的嵌套

```
.data
    count DWORD 0
.code
    MOV ECX, 100；  L1 循环100次
L1：
    MOV count, ECX
    MOV ECX, 10 ; L2 循环10次
L2:
    … ….
    LOOP L2
    MOV ECX，  count
    LOOP L1
```

主观题 10分

设置

```
.data
    array DWORD 100h, 200h, 300h, 400h
```

编写汇编代码，计算数组array的和

正常使用主观题需2.0以上版本雨课堂

作答

LOOP指令的变种（自学）

- LOOPZ、 LOOPE
- LOOPNZ、 LOOPNE

数组求和

```
.data
    array DWORD 100h, 200h, 300h, 400h
.code
    MOV ECX, LENGTHOF array ; 循环次数
    MOV EDI，OFFSET array；索引
    MOV EAX，0；和
L1：
    ADD EAX，[EDI]
    ADD EDI, TYPE array
    LOOP L1
```

主观题 10分

设置

```
.data
    src BYTE “Hello World”, 0Dh, 0Ah, 0
    dst BYTE SIZEOF src DUP(0), 0
```

使用LOOP指令，将字符串src复制到dst

正常使用主观题需2.0以上版本雨课堂

作答

字符串赋值

```
.data
    src BYTE “Hello World”, 0Dh, 0Ah, 0
    dst BYTE SIZEOF src DUP(0), 0
.code
    MOV ECX, SIZEOF src ；循环次数
    MOV ESI, 0 ；字符索引
L1:
    MOV AL, BYTE PTR src[ESI]
    MOV BYTE PTR dst[ESI], AL
    INC ESI
    LOOP L1
```

主观题 10分

设置

.data
    num BYTE 1, 2, 3, 4, 5, 6, 7, 8, 9, 0
.code
    将数字转换成对应的ASCII字符，输出到命令行窗口
ASCII 编码：
‘0’ 30h
‘1’31h
… …
‘9’39h

正常使用主观题需2.0以上版本雨课堂

作答

南開大學 Nankai University

```
.data
    num BYTE 1，2，3，4，5，6，7，8，9，0，0
.code
    MOV ECX, 10 ；循环次数
    MOV ESI, 0 ； 索引
L1:
    MOV AL, BYTE PTR num[ESI]
    ADD AL，30h
    MOV BYTE PTR num[ESI], AL
    INC ESI
    LOOP L1
```

整数算术指令

移位和循环移位指令

指　令	含　义
SHL	逻辑左移
SHR	逻辑右移
SAL	算术左移
SAR	算术右移
ROL	循环左移
ROR	循环右移
RCL	带进位的循环左移

逻辑移位

- 逻辑**左移\右移**：对无符号数进行移位，以0填充移出的位

SHL\SHR 目的操作数，移位位数

```
SHL  reg,imm8
SHL  mem,imm8

SHL  reg,CL
SHL  mem,CL
```

可以使用CL存放移动的位数，适用于所有的移位操作

逻辑移位

逻辑左移：实现无符号数快速乘法，2^n

逻辑右移：实现无符号数快速除法，2^n

- **实现dl乘以2**

```
mov dl,5
shl dl,1
```

算术移位

- 算术**左移\右移**：对有符号数进行移位，符号位不变，
- 算术左移：以0填充移出的位；算术右移：以符号位填充移出的位

SAL\SAR 目的操作数，移位位数

```
mov   al,0F0h                ; AL = 11110000b (-16)
sar   al,1                   ; AL = 11111000b (-8)   CF = 0
```

填空题 0分

设置

此题未设置答案，请点击右侧设置按钮

请写出

```
MOV    AL, 35h
SHL    AL，3
```

的运算结果

作答

填空题 0分

设置

此题未设置答案，请点击右侧设置按钮

请写出

```
MOV    AL, 0B5h
SAR    AL，2
```

的运算结果

作答

循环移位

- 向左\右循环移位后，把最高\低位同时复制到进位标志和移出位中
  - 循环左移利用**ROL**实现

```
mov  al,40h          ; AL = 01000000b
rol  al,1            ; AL = 10000000b, CF = 0
rol  al,1            ; AL = 00000001b, CF = 1
rol  al,1            ; AL = 00000010b, CF = 0
```

  - 循环右移利用**ROR**实现

```
mov al,01h           ; AL = 00000001b
ror al,1             ; AL = 10000000b, CF = 1
ror al,1             ; AL = 01000000b, CF = 0
```

- 此外，还有**RCL**和**RCR**的变种（自学）

快速二进制乘法

- 可以利用逻辑移位实现快速乘法，相比较MUL\IMUL指令，执行速度更快

例如：实现EAX*36 = EAX*32+ EAX*4

```
.code
mov   eax,123
mov   ebx,eax
shl   eax,5                    ;乘以 2^5
shl   ebx,2                    ;乘以 2^2
add   eax,ebx                  ;积相加
```

主观题 10分

设置

请写出实现EAX*50的汇编代码

作答

无符号乘法指令

- 乘法：MUL无符号乘法指令，乘数与被乘数具有相同尺寸，乘积是乘数/被乘数的2倍
  - 接受一个操作数（即乘数）：寄存器操作数、内存操作数均可，但不能是立即数。另一个隐含操作数，即被乘数是EAX。

MUL r/m8

MUL r/m16

MUL r/m32

被乘数	乘数	积
AL	r/m8	AX
AX	r/m16	DX:AX
EAX	r/m32	EDX:EAX

16\32位乘数，其乘积高位存于DX\EDX中

有符号乘法指令

- 乘法：IMUL有符号乘法指令，用法与MUL相似。 除了单操作数，还支持双、三操作数

- **单操作数：结果存于EAX或EDX:EAX中**

```
IMUL r/m8          ; AX = AL * r/m byte
IMUL r/m16         ; DX:AX = AX * r/m word
IMUL r/m32         ; EDX:EAX = EAX * r/m doubleword
```

- 双操作数：结果存于第1操作数

```
IMUL r16,r/m16     IMUL r32,r/m32
IMUL r16,imm8      IMUL r32,imm8
IMUL r16,imm16     IMUL r32,imm32
```

- 三操作数：结果存于第1操作数

```
IMUL r16,r/m16,imm8      IMUL r32,r/m32,imm8
IMUL r16,r/m16,imm16     IMUL r32,r/m32,imm32
```

无符号除法指令

- 除法：DIV无符号除法指令，与MUL类似，唯一操作数为除数，只能为寄存器或内存。
- 结果分为商和余数，商一般存入EAX，余数存入AH或EDX
-

```
DIV r/m8
DIV r/m16
DIV r/m32
```

被除数	除数	商	余数
AX	r/m8	AL	AH
DX:AX	r/m16	AX	DX
EDX:EAX	r/m32	EAX	EDX

有符号除法指令

- 除法：IDIV有符号除法指令，余数的符号与被除数保持一致
- 符号扩展：需要对被除数进行符号扩展，8位除法扩展至AH，16位除法扩展至DX，32位除法扩展至EDX
- 除法溢出：可能由于商太大导致除法溢出，会触发CPU中断

过程的定义和使用

过程

- C++中的函数定义

```
return_type function_name( parameter list ) {
    body of the function
}
```

过程

- **返回类型**：一个函数可以返回一个值。**return_type** 是函数返回值的数据类型。
- **函数名称**：这是函数的实际名称。函数名和参数列表一起构成了函数签名。

过程

- **参数**：参数就像是占位符。当函数被调用时，向参数传递一个值，这个值被称为实际参数。参数列表包括函数参数的类型、顺序、数量。
- **函数主体**：函数主体包含一组定义函数执行任务的语句。

过程

```
int max(int num1, int num2) {
    // 局部变量声明
    int result;
    if (num1 > num2)
        result = num1;
    else
        result = num2;
    return result;
}
```

过程

- 汇编语言中，一般使用术语“过程”（procedure）表示高级语言中的函数、方法
  - 函数（function），C/C++中称为函数
  - 方法（method），JAVA中称为方法

过程

- 汇编语言早于 “面向对象”、“面向函数”的编程语言
  - 编译器是如何将这些高级语言中的函数、过程翻译成汇编语言的？

过程

- 汇编语言把过程定义为以返回语句结束的命名语句块。
  - 使用PROC和ENDP伪指令来声明过程
  - 必须定义一个过程名字（标识符）

过程

my_proc PROC

... ...

ret

my_proc ENDP

- 除启动过程之外，其它过程以ret指令结束
  - 将CPU控制权转移到过程被调用的地方

启动过程（main）

```
main PROC
    ... ...
main ENDP
```

- 启动过程（main）的返回语句是
  - INVOKE ExitProcess, 0
  - 将CPU的控制权转移给Windows操作系统

过程的定义

- MyProc过程，计算寄存器EAX、EBX、ECX之和

```
MyProc PROC
    ADD EAX，EBX
    ADD EAX，ECX
    RET
MyProc ENDP
```

过程的调用与返回

- CALL指令将CPU的控制权转移到新的内存地址执行指令，实现过程的调用
- RET指令将CPU的控制权返回到程序中过程被调用的地方继续执行

过程的调用与返回

```
main PROC
    MOV EAX, 1000h
    MOV EBX, 1000h
    MOV ECX, 1000h
    CALL MyProc
    INVOKE ExitProcess, 0
main ENDP
MyProc PROC
    ADD EAX, EBX
    ADD EAX, ECX
    RET
MyProc ENDP
```

过程的调用与返回

- 过程返回地址的保存
  - CALL指令调用之后，将过程的返回地址压入堆栈，将过程入口地址赋值给EIP，实现CPU控制权的转移
  - RET指令调用之后，将过程的返回地址赋值给EIP寄存器，实现CPU控制权的转移

过程的调用与返回


00401000 68 00304000 PUSH OFFSET 00403000 ASCII "Hello World",CR,LF
00401005 E8 66000000 CALL 00401070
0040100A B8 00100000 MOV EAX,1000
0040100F BB 00100000 MOV EBX,1000
00401014 B9 00100000 MOV ECX,1000
00401019 E8 1C000000 CALL 0040103A
0040101E 68 0E304000 PUSH OFFSET 0040300E
00401023 50 PUSH EAX
00401024 E8 1B000000 CALL 00401044
00401029 68 0E304000 PUSH OFFSET 0040300E
0040102E E8 3D000000 CALL 00401070
00401033 6A 00 PUSH 0
00401035 E8 E8000000 CALL <JMP.&kernel32.ExitProcess> 跳转至 KERNEL32.ExitProcess
0040103A 03C3 ADD EAX,EBX
0040103C 03C1 ADD EAX,ECX
0040103E C3 RETN
0040103F E8 BCFFFFFF CALL 00401000
00401044 55 PUSH EBP
寄存器 (FPU)
EAX 00003000
ECX 00001000
EDX 0019FF1C
EBX 00001000
ESP 0019FF70
EBP 0019FF80
ESI 0040103F proc.<ModuleEntry
EDI 0040103F proc.<ModuleEntry
EIP 0040101E proc.0040101E
C 0 ES 002B 32Bit 0(FFFFFFFF)
P 1 CS 0023 32Bit 0(FFFFFFFF)
A 0 SS 002B 32Bit 0(FFFFFFFF)
Z 0 DS 002B 32Bit 0(FFFFFFFF)
S 0 FS 0053 32Bit 308000(FFF)
T 0 GS 002B 32Bit 0(FFFFFFFF)
D 0
O 0 LastErr 00000000 ERROR_SU
Stack [0019FF6C]=proc.0040101E
地址 十六进制数据 多字节 (ANSI/OEM - 简体中文
00403000 48 65 6C 6C 6F 20 57 6F 72 6C 64 0D 0A 00 00 00 H e l l o W o r l d
0019FF70 00401044 返回到 proc.00401000 来自 proc.00401
0019FF74 74CB0419 返回到 KERNEL32.74CB0419

调用链接库中的函数

链接库

- 链接库（Link Library）是一个文件，包含已经编译成机器码的过程，其引入过程如下。
  - includelib \masm32\lib\masm32.lib
  - includelib \masm32\lib\kernel32.lib

PROTO伪指令

- PROTO伪指令用于声明链接库中的过程
- include \masm32\include\masm32.inc
- include \masm32\include\kernel32.inc

PROTO伪指令声明举例

- StdOut
  - StdOut PROTO :DWORD
- StdIn
  - StdIn PROTO :DWORD,:DWORD
- ExitProcess
  - ExitProcess PROTO STDCALL :DWORD

库包含文件

- 库包含文件，包含.inc文件，类似头文件
  - include \masm32\include\masm32.inc
  - include \Irvine32\Irvine32.inc

典型的附带链接库文件

- Irvine32和Irvine16链接库，包含了常用的输入输出功能
  - includelib \Irvine32\Irvine32.lib
  - include \Irvine32\Irvine32.inc
- 利用call调用相关过程

```
mov  eax,fileHandle
call CloseFile
```

Irvine32和Irvine16提供的过程

表 5.1 本书链接库中的过程

过程名称	描 述
CloseFile	关闭以前打开的磁盘文件*
Clrscr	清除控制台窗口并将光标定位在左上角
CreateOutputFile	创建一个新的磁盘文件用于输出*
Crlf	在标准输出上输出换行符
Delay	暂停程序 *n* 毫秒
DumpMem	以十六进制数格式在控制台上显示一块内存的内容
DumpRegs	以十六进制数显示 EAX、EBX、ECX、EDX、ESI、EDI、EBP、ESP、EFLAGS 和 EIP 寄存器的内容，同时显示常用的 CPU 状态标志的值
GetCommandTail	把程序的命令行参数复制到一个字节数组中
GetMaxXY	获取控制台窗口缓冲区的行数和列数*
GetMseconds	返回从午夜开始逝去的毫秒数
GetTextColor	返回当前控制台窗口的前景和背景色*
Gotoxy	把光标定位在控制台中指定的行列位置上
IsDigit	如果 AL 寄存器中包含十进制数字（0~9）的 ASCII 码，则设置零标注
MsgBox	显示一个弹出消息框*
MsgBoxAsk	显示一个带一个 Yes 按钮和一个 No 按钮的弹出消息框*
OpenInputFile	打开一个磁盘文件以进行输入*
ParseDecimal32	把无符号整数字符串转换为一个 32 位的二进制数
Random32	生成一个 0~FFFFFFFFh 范围内的伪随机整数
Randomize	以指定值初始化随机数发生器种子
RandomRange	生成指定范围内的伪随机数
ReadChar	等待键盘输入一个字符并返回该字符
ReadDec	读取键盘输入的无符号 32 位十进制整数，以回车键结束
ReadFromFile	把输入文件的内容读到一个缓冲区内*
ReadHex	读取键盘输入的 32 位十六进制整数，以回车键结束
ReadInt	读取键盘输入的 32 位有符号十进制整数，以回车键结束
ReadKey	从键盘输入缓冲区读入一个字符，如果无输入不等待
ReadString	读取键盘输入的一个字符串，以回车键结束
SetTextColor	设置随后所有输出到控制台的文本的前景和背景色
StrLength	返回字符串的长度
WaitMsg	显示一条消息并等待按下一个键
WriteBin	以 ASCII 码二进制数格式在控制台上显示一个 32 位无符号整数
WriteBinB	以 ASCII 码二进制数格式显示整数，可以指定以字节、字或双字的方式显示
WriteChar	在标准输出显示一个字符
WriteDec	以十进制数格式在控制台上显示一个 32 位无符号整数
WriteHex	以十六进制数格式在控制台上显示一个 32 位整数
WriteHexB	以十六进制数格式显示整数，可以指定以字节、字或双字的方式显示
WriteInt	以十进制数格式在控制台上显示一个 32 位有符号整数
WriteString	在控制台上显示一个以空字符结尾的字符串
WriteToFile	把缓冲区的内容写入文件*
WriteWindowsMsg	显示一个描述 MS-Windows 产生的最后一次错误的字符串*

堆栈功能

- 压栈操作（PUSH）

PUSH 指令首先减小 ESP 的值，然后再把一个 16 位或 32 位的源操作数复制到堆栈上。对于 16 位操作数，ESP 值将减 2；对于 32 位操作数，ESP 值将减 4。PUSH 指令有以下三种格式：

```
PUSH r/m16
PUSH r/m32
PUSH imm32
```

堆栈功能

- 出栈操作（POP、POPFD）

POP 指令首先将 ESP 所指的堆栈元素复制到 16 位或 32 位的目的操作数中，然后增加 ESP 的值。如果操作数是 16 位的，ESP 值将加 2；如果操作数是 32 位的，ESP 值将加 4。其格式如下：

```
POP  r/m16
POP  r/m32
```

单选题 2分

设置

执行 PUSH EAX
PUSH BX
后ESP的值变为?

A ESP+6

B ESP+8

C ESP-8

D ESP-6

提交

单选题 2分

设置

执行 PUSH EAX
POP BX
后ESP的值变为?

A ESP+2

B ESP+6

C ESP-2

D ESP-6

提交

堆栈功能

- 标志位的压栈出栈（PUSHFD、POPFD）

PUSHFD 指令在堆栈上压入 32 位的 EFLAGS 寄存器的值，POPFD 指令将堆栈顶部的值弹出并送至 EFLAGS 寄存器：

```
pushfd
popfd
```