第九讲　WIN32 编程（P238）

WIN32 程序：基于 **Windows** 运行环境的 **32 位段**程序。

一、走进 WIN32 程序的基本思路

；-------这是在 **DOS** 下显示“**How are you!**”的程序---------

```
.386
STACK  SEGMENT  STACK  USE16
       DB   200  DUP(0)
STACK  ENDS
DATA  SEGMENT  USE16
hello  DB   'How are you! $'
DATA  ENDS
CODE  SEGMENT    USE16
        ASSUME  CS:CODE, DS:DATA, SS:STACK
BEGIN:  MOV  AX,  DATA
        MOV  DS,  AX
        LEA  DX,  hello
        MOV  AH,  9
        INT  21H
        MOV  AH, 4CH
        INT  21H
CODE  ENDS
        END  BEGIN
```

（1）如何改成 **32** 位段程序？

（2）如何改成在 **Windows** 下运行的 **32** 位程序？

```
.386
    ;EXTERN MessageBoxA:NEAR
    ;EXTERN ExitProcess:NEAR    ;汇编可以通过，但 LINK 时在库
                                   中的解析信息不够
    MessageBoxA PROTO STDCALL :DWORD,:DWORD,:DWORD,:DWORD
    ExitProcess PROTO STDCALL :DWORD
    includelib  user32.lib
    includelib  kernel32.lib
DATA    SEGMENT
hello   DB "HOW ARE YOU!",0
dir     DB "问候",0
DATA    ENDS
STACK   SEGMENT STACK
        DB  1000 DUP(0)
STACK   ENDS
CODE    SEGMENT
        ASSUME CS:CODE, DS:DATA, SS:STACK
_START: PUSH 0
         PUSH OFFSET dir
         PUSH OFFSET hello
         PUSH 0
         CALL MessageBoxA
         PUSH 0
         CALL ExitProcess
CODE    ENDS
        END _START              ;起始地址的标号前带下划线
```

问题（1）：32 位段使程序可以直接访问所有内存空间，一般不

必用定义多个段的方法获得较大的空间，因此，完整的段定义方法显得复杂不必要。

解决办法：采用简化段定义的方法，定义三个基本种类的段（CODE、STACK、DATA）。

.CODE [段名]

.DATA

另：常数（只读）数据段定义：**.CONST**；

变量是未初始化数据段定义：**.DATA?**

.STACK [堆栈字节数]；（省略时为 **1024**）

但在源文件中所有其它段定义伪指令之前写上一个且只能使用一次伪指令 **MODEL**。

格式： .MODEL 存储模型 [,语言类型][,系统类型][,堆栈选项]

功能：“存储模型”指定内存管理模式

存储 模型	段的大小	代码访问范围	数据访问范围	备注
TINY	16 位	NEAR	NEAR	代码和数据全部放在同一个 64K 段内，常用于生成.COM 程序
SMALL	16 位	NEAR	NEAR	代码和数据在各自的 64K 段内,代码总量和数据总量均不超过 64K
COMPACT	16 位	NEAR	FAR	代码总量不超过 64K，数据总量可以超过 64K
MEDIUM	16 位	FAR	NEAR	代码总量可超过 64K，数据总量不超过 64K
LARGE	16 位	FAR	FAR	代码和数据总量均可超过 64K，但单个数组不超过 64K
HUGE	16 位	FAR	FAR	代码和数据总量均可超过 64K，单个数组可超过 64K
FLAT	32 位	NEAR	NEAR	代码和数据全部放在同一个 4G 空间内

“语言类型”指定了函数命名、调用和返回的方法，例如 C、PASCAL 或 STDCALL 等。

STDCALL 类型：采用堆栈法传递参数，参数进栈次序为：函数原型描述的参数中最右边的参数最先入栈、最左边的最后入栈；由被调用者在返回时清除参数占用的堆栈空间

FLAT 下的程序启动时，DS、ES、SS 已经被设为同一个有效值。

例：

```
.386
.MODEL FLAT, STDCALL
    MessageBoxA PROTO STDCALL :DWORD,:DWORD,:DWORD,:DWORD
    ExitProcess PROTO STDCALL :DWORD
    includelib  user32.lib
    includelib  kernel32.lib
.DATA
   hello  DB "HOW ARE YOU!",0
   dir    DB "问候",0
.STACK 200
.CODE
START:  PUSH 0
        PUSH OFFSET dir
        PUSH OFFSET hello
        PUSH 0
        CALL MessageBoxA
        PUSH 0
        CALL ExitProcess
END START
```

问题（2）：**WIN-API：高级语言函数形式，放在子程序库中；参数多，取值复杂。例：**

```
int MessageBoxA(
    HWND      hWnd,        //所属窗口的句柄，=0 没有所属窗口
    LPCTSTR   lpText,      //待显示字符串（结束符为 0）的首地址
    LPCTSTR   lpCaption, //消息框标题字符串（结束符为 0）的首
```

地址指针；为 0 时显示标题“Error”

```
    UINT uType          //指定消息框的样式
    )
    对应
MessageBoxA  PROTO
     hWnd       :DWORD,
     lpText     :DWORD,
     lpCaption  :DWORD,
     uType      :DWORD
```

其中，uType 指定了在消息框中要显示哪些按钮和图标，例：

=0，仅显示“确认”按钮（MB_OK EQU 0）

=1，同时显示“确认”和“取消”按钮（MB_OKCANCEL=1）

=3， 同时显示“是”、“否”和“取消”按钮（MB_YESNOCANCEL EQU 3）

解决办法：（1）将这些符号常量、函数原型的定义集中放到头文件中，用“INCLUDE 头文件”的方法使用。windows.inc

kernel32.inc， kernel32.lib；

user32.inc，user32.lib；

gdi32.inc，gdi32.lib

（2）使用结构 STRUCT 等数据类型表达复杂的参数。

（3）提供新的子程序定义、调用等方法，使 WIN-API 调用过程更加直观，不易出错。

伪指令 PUBLIC、EXTRN、PROC 和指令 CALL。

新的伪指令 PROTO 和 INVOKE，使子程序的说明/调用形式类似于高级语言。

①原型说明伪指令 PROTO

格式：函数名 PROTO [函数类型][语言类型][[参数名]:参数类型],[[参数名]:参数类型]...

功能：用于说明本模块中要调用的过程或函数。

例如：

```
RADIX_S PROTO FAR C LpResult:WORD,Radix:DWORD,:DWORD
MessageBoxA PROTO hWnd:DWORD，lpText:DWORD，lpCaption :DWORD，
                uType:DWORD
```

“函数类型”:WIN32 应用程序中一般为 NEAR 类型。

“语言类型”.MODEL 语句后指定了语言类型，此处就可省略。

参数名可以省略，但冒号和参数类型不能省略。

②完整的函数定义伪指令 PROC

格式：函数名 PROC [函数类型][语言类型][USES 寄存器表][,参数名[:类型]]...

功能：定义一个新的函数（函数体应紧跟其后）。

参数名是用来传递数据的，可以在程序中直接引用，故不能省略。

USES 后面所列的寄存器是需要在子程序中入栈保护的（由汇编程序自动加入入栈保护和出栈恢复的指令语句）。

局部变量的定义（紧跟在 PROC 语句之后）：

LOCAL 变量名[[数量]][：类型]

局部变量所指的单元在堆栈中。

```
RADIX_S PROC NEAR STDCALL USES EBX EDX SI,LpResult,Radix:DWORD,
                                                NUM:DWORD
        LOCAL COUNT:WORD    ；说明局部变量，用于计数（取代原来程
                              序中的 CX 寄存器）
        MOV  COUNT,0        ；局部变量（计数器）赋初值 0
```

```
        MOV   EAX,NUM          ；从堆栈中取传递过来的待转换数（偏移
                                  值由汇编程序计算）
        MOV   SI,lpResult      ；从堆栈中取传递过来的缓冲区指针
        MOV   EBX,Radix        ；从堆栈中取传递过来的基数 P
LOP1:   MOV   EDX,0            ；以下返回之前部分除 CX 换成 COUNT 外，
                                  同原来的程序
        …
        MOV   AX,SI            ；设置返回参数
        RET
RADIX_S   ENDP
```

③ 函数调用伪指令 INVOKE

格式：INVOKE 函数名 [,参数]...

功能：调用函数。其中，参数可以是各种表达式。

INVOKE 必须在 PROTO 语句或完整的 PROC 语句说明之后使用。

例如：

```
INVOKE   RADIX_S,SI,10,EAX
INVOKE   MessageBoxA，0，OFFSET dir，OFFSET hello，MB_OK
```

INVOKE 与CALL 都完成了子程序的调用；

ADDR 与 OFFSET；

ADDR 只能在 INVOKE 中用，可以取之前定义的局部变量的 EA。

```
INVOKE   MessageBoxA，0，addr dir，ADDR hello，MB_OK
```

二、 WIN32 程序的结构

标准 WIN32 程序

基于窗口的应用程序结构可以简单地划分为四个部分：

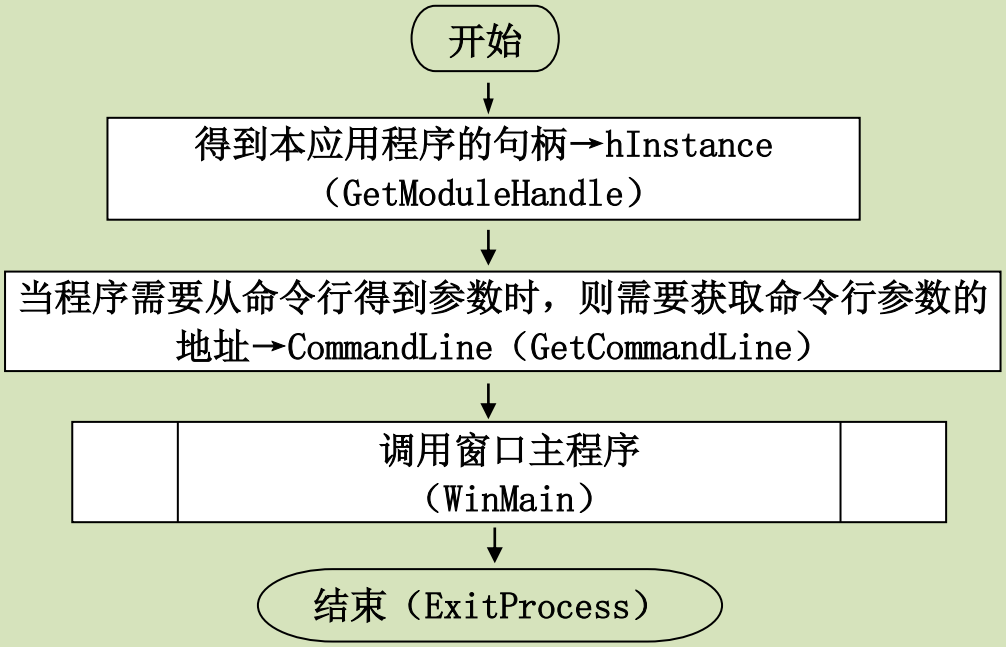
主程序： OS 首先执行“主程序”，获得与本程序有关的基本信息后再调用“窗口主程序”，

窗口主程序： 创建指定窗口后，将该窗口收到的消息通过 OS 转发到“窗口消息处理程序”

窗口消息处理程序： 判断收到的消息种类，决定应该调用“用户处理程序”中的哪一个或几个函数完成相应的功能

用户处理程序： 完成用户实际需求的各种函数的集合

“主程序”流程图：



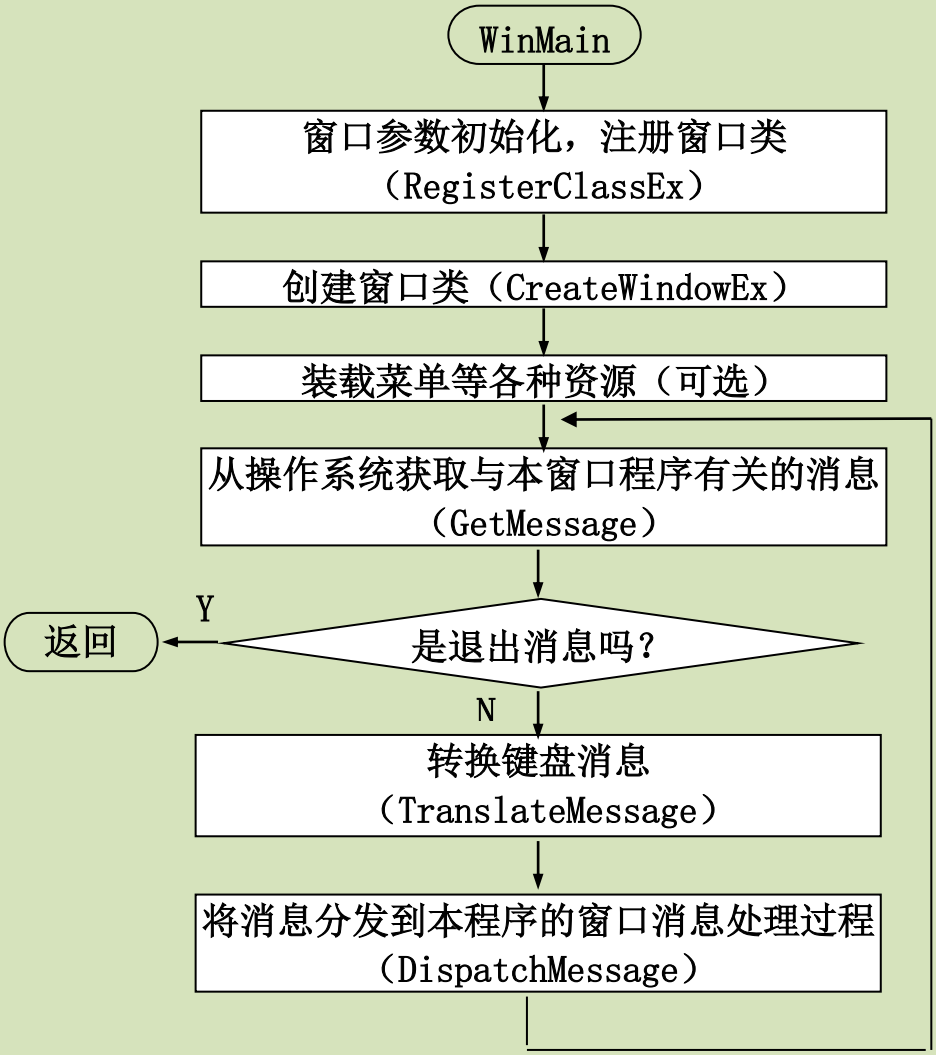
“窗口主程序”原型说明的形式为：

```
WinMain PROTO hInst：DWORD，   ；应用程序的实例句柄
        hPrevInst：DWORD，     ；前一个实例句柄（恒为 NULL）
```

lpCmdLine：DWORD，　；命令行指针，指向以 0 结束的命令行字符串

nCmdShow：DWORD　　；指出如何显示窗口

“窗口主程序”的流程图：



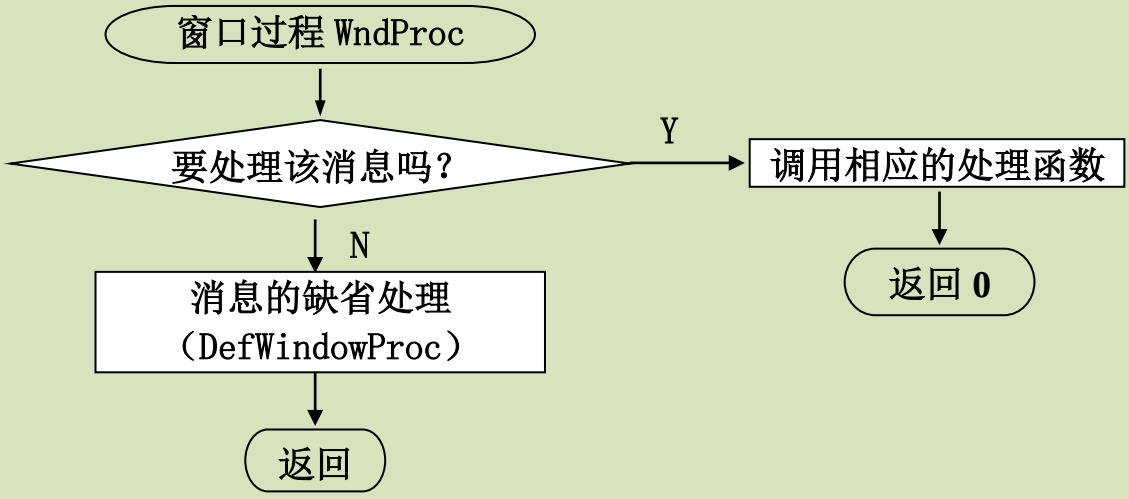
“窗口主程序”中没有直接调用“窗口消息处理程序”

“窗口消息处理程序”（或称窗口过程，也是一个函数体形式）的主要功能是对接收到的消息进行判断，以便分类处理。其程序结构是一个典型的分支程序

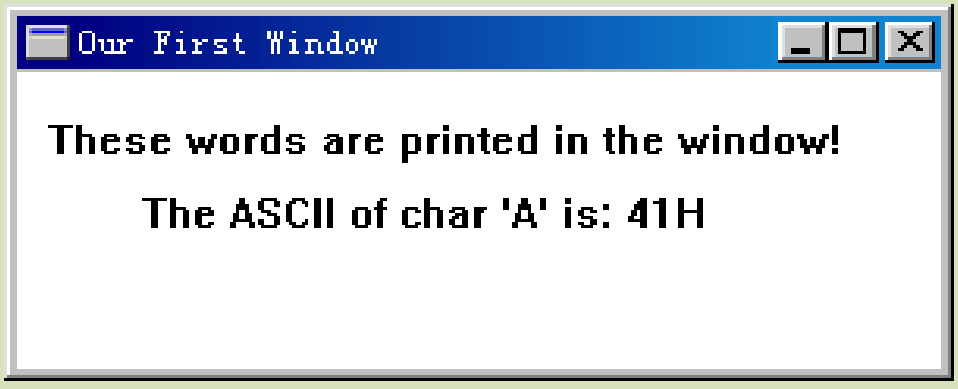
“窗口消息处理程序”的原型说明必须满足如下的形式：

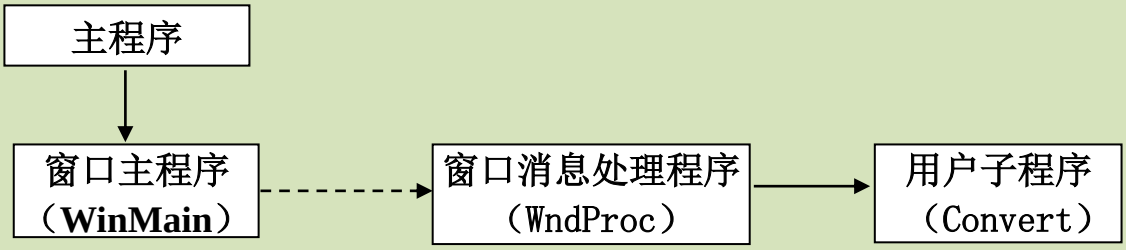
```
WndProc PROTO hWin    :DWORD,      ；窗口句柄
            uMsg      :DWORD,      ；消息号，指明消息的种类，是
                                     分支判断的主要依据
            wParam    :DWORD,      ；该消息的附加信息。若是子消
                                     息号，则是嵌套分支判断的依据
            lParam    :DWORD       ；该消息的附加信息
```

流程图：



例： 在屏幕上创建一个窗口，窗口的标题为：“Our First Window”。当按下一个键时，在窗口中显示字符串“These words are printed in the window!”，并将该键的 ASCII 码以十六进制的形式显示出来。显示的格式为：





模块结构图

```
.386
.MODEL FLAT,STDCALL
OPTION CASEMAP:NONE

WinMain PROTO :DWORD,:DWORD,:DWORD,:DWORD
WndProc PROTO :DWORD,:DWORD,:DWORD,:DWORD
Convert PROTO :BYTE, :DWORD

INCLUDE     windows.inc          ; 头文件说明
INCLUDE     gdi32.inc
INCLUDE     user32.inc
INCLUDE     kernel32.inc
INCLUDELIB gdi32.lib             ; 引入库说明
INCLUDELIB user32.lib
INCLUDELIB kernel32.lib
.DATA
    ClassName    DB  "TryWinClass",0                          ; 窗口类名
    AppName      DB  "Our First Window ",0                    ; 窗口标题
    hInstance    DD   0                                       ; 实例句柄
    CommandLine DD   0
   OurText      DB  "These words are printed in the window!" ; 窗口中待显示
                                                               的字符串
   num1         =   $-OurText
   OurStr       DB  "The ASCII of char '?' is:",20H          ; 20H 为空格符
   szASCII      DB  "00H",20H,20H                           ;后面加了 2 个空格
   num2         =   $-OurStr

.CODE
; ----------------------主程序-----------------------------------------
START:
    INVOKE  GetModuleHandle, NULL                ; 获得并保存本程序的句柄
    MOV     hInstance,EAX
    INVOKE  GetCommandLine
```

```
    MOV     CommandLine,EAX
    INVOKE  WinMain, hInstance,NULL,CommandLine, SW_SHOWDEFAULT ; 调用窗口主程序
    INVOKE  ExitProcess,EAX                                     ; 退出本程序，返回 Windows

; ------------------------窗口主程序-----------------------------------------
WinMain PROC hInst:DWORD,hPrevInst:DWORD,CmdLine:DWORD,CmdShow:DWORD
        LOCAL   wc    :WNDCLASSEX       ;创建主窗口时所需要的信息由该结构说明
        LOCAL   msg   :MSG              ; 消息结构变量用于存放获取的消息
        LOCAL   hwnd  :HWND             ; 存放窗口句柄
  ; 给 WNDCLASSEX 结构变量 wc 的各字段赋值
   MOV     wc.cbSize,SIZEOF WNDCLASSEX          ; WNDCLASSE 结构类型的字节数
   MOV     wc.style, CS_HREDRAW or CS_VREDRAW   ; 窗口风格(当窗口高度和宽度变化时则重画窗口)
   MOV     wc.lpfnWndProc, OFFSET WndProc       ; 本窗口过程的入口地址（偏移地址）
   MOV     wc.cbClsExtra,NULL                   ; 不使用自定义数据则不需 OS 预留空间，置 NULL
   MOV     wc.cbWndExtra,NULL                   ; 同上
   PUSH    hInst                                ; 本应用程序句柄→wc.hInstance
   POP     wc.hInstance
   MOV     wc.hbrBackground,COLOR_WINDOW+1      ; 窗口的背景颜色为白色
   MOV     wc.lpszMenuName,NULL                 ; 窗口上不带菜单，置为 NULL
   MOV     wc.lpszClassName,OFFSET ClassName    ; 窗口类名"TryWinClass"的地址
   INVOKE  LoadIcon,NULL,IDI_APPLICATION        ; 装入系统默认的图标
   MOV     wc.hIcon,EAX                         ; 保存图标的句柄
   MOV     wc.hIconSm,0                         ; 窗口不带小图标
   INVOKE  LoadCursor,NULL,IDC_ARROW            ; 装入系统默认的光标
   MOV     wc.hCursor,EAX                       ; 保存光标的句柄
  INVOKE RegisterClassEx, ADDR wc                       ; 注册窗口类
  INVOKE  CreateWindowEx,NULL,ADDR ClassName, ; 建立"TryWinClass"类窗口
                ADDR AppName,                   ; 窗口标题"Our First Window"的地址
                WS_OVERLAPPEDWINDOW+ WS_VISIBLE,; 创建可显示的窗口
                CW_USEDEFAULT,CW_USEDEFAULT,    ; 窗口左上角坐标默认值
                CW_USEDEFAULT,CW_USEDEFAULT,    ; 窗口宽度，高度默认值
                NULL,NULL,                      ; 无父窗口，无菜单
                hInst,NULL                      ; 本程序句柄，无参数传递给窗口
   MOV     hwnd,EAX                             ; 保存窗口的句柄
StartLoop:                                      ; 进入消息循环
   INVOKE GetMessage, ADDR msg,NULL,0,0         ; 从 Windows 获取消息
   CMP     EAX, 0                               ; 如果（EAX）不为 0 则要转换并分发消息
   JE      ExitLoop                             ; 如果（EAX）为 0 则转 ExitLoop
   INVOKE TranslateMessage, ADDR msg            ; 从键盘接受按键并转换为消息
   INVOKE DispatchMessage, ADDR msg             ; 将消息分发到窗口的消息处理程序
   JMP     StartLoop                            ; 再循环获取消息
ExitLoop:
```

```
        MOV     EAX,msg.wParam          ；设置返回（退出）码
        RET
WinMain ENDP
；--------------------窗口消息处理程序-----------------------------------------
WndProc PROC hWnd:DWORD,uMsg:DWORD,wParam:DWORD,lParam:DWORD
   LOCAL hdc:HDC                        ；存放设备上下文句柄

  .IF uMsg == WM_DESTROY                ；收到的是销毁窗口消息
      INVOKE PostQuitMessage,NULL       ；发退出消息
  .ELSEIF uMsg == WM_ CHAR              ；收到的是在窗口中按键的消息
      INVOKE GetDC,hWnd                 ；根据窗口句柄确定设备句柄
     MOV    hdc,EAX                     ；保存设备上下文句柄
     MOV    eax,wParam                  ；将按键的 ASCII 码送到 AL 中
     MOV    szASCII-7,AL                ；用按键的 ASCII 码替换 OurStr 串中的？
     INVOKE Convert,AL,ADDR szASCII     ；将(AL)转换成 16 进制形式的显示码→
                                          szASCII 中
     INVOKE TextOut ,hdc,10,15,ADDR OurText,num1 ；从窗口坐标（10，15）开
                                                   始显示 OurText 指向的串
     INVOKE TextOut ,hdc,40,40,ADDR OurStr,num2  ；从窗口坐标（40，40）开
                                                   始显示 OurStr 指向的串
  .ELSE
      INVOKE DefWindowProc,hWnd,uMsg,wParam,lParam ；不是本程序要处理的消
                                                     息作其它缺省处理
      RET
  .ENDIF
      XOR    EAX,EAX
      RET
WndProc ENDP
；--------------------用户处理程序-----------------------------------------
；子程序名：Convert
；功能：将 8 位数按 16 进制形式转换成 ASCII 码的子程序
；原型：Convert PROTO
；              bChar:BYTE,     ；待转换的 8 位二进制数
；              lpStr:DWORD     ；转换结果的存放地址
；受影响的寄存器：AL，ESI
Convert PROC bChar:BYTE, lpStr:DWORD
        MOV  AL,bChar
        MOV  ESI,lpStr
        SHR  AL,4
        CMP  AL,10
```

```
        JB    L1
        ADD   AL,7
L1:     ADD   AL,30H
        MOV   [ESI],AL
        MOV   AL,bChar
        AND   AL,0FH
        CMP   AL,10
        JB    L2
        ADD   AL,7
L2:     ADD   AL,30H
        MOV   [ESI+1],AL
        RET
Convert ENDP
        END START
```