北京理工大學

本科生毕业设计（论文）

汇编语言与接口技术组队实验 - 2048

Assembly and Interface Team Programming Experiment - 2048

学　　院：计算机学院

专　　业：计算机科学与技术

学生姓名：卜梦煜、陈泓旭、傅泽、武智强

学　　号：1120192419、1120192064、1120192062、1120192427

指导教师：张全新

2022 年 6 月 15 日

原创性声明

本人郑重声明：所呈交的毕业设计（论文），是本人在指导老师的指导下独立进行研究所取得的成果。除文中已经注明引用的内容外，本文不包含任何其他个人或集体已经发表或撰写过的研究成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。

特此申明。

本人签名：　　日 期：　　年　月　日

关于使用授权的声明

本人完全了解北京理工大学有关保管、使用毕业设计（论文）的规定，其中包括：①学校有权保管、并向有关部门送交本毕业设计（论文）的原件与复印件；②学校可以采用影印、缩印或其它复制手段复制并保存本毕业设计（论文）；③学校可允许本毕业设计（论文）被查阅或借阅；④学校可以学术交流为目的，复制赠送和交换本毕业设计（论文）；⑤学校可以公布本毕业设计（论文）的全部或部分内容。

本人签名：　　日 期：　　年　月　日

指导老师签名：　　日 期：　　年　月　日

汇编语言与接口技术组队实验 - 2048

摘　要

汇编语言课程设计是对所学课程内容全面、系统的总结、巩固和提高的一项课程实践活动。根据汇编语言的特点，选择制作了 2048 游戏。课程的设计过程中我们熟悉了汇编语言的编程，子程序的段内调用，堆栈技术，寄存器的应用等的相关知识。通过课程设计，一方面提高了运用汇编语言解决实际问题的能力，另一方面使同学们更深入的了解计算机系统内部的相关知识，为以后的学习和系统开发奠定良好的基础。

小组成员分工如下：卜梦煜同学和武智强同学负责界面设计、美化与代码实现工作；陈泓旭同学负责实现随机数生成算法和棋盘数字移动合并算法；傅泽同学负责分数计算部分的编程实现和测试 debug 工作。

Assembly and Interface Team Programming Experiment - 2048

Abstract

The assembly language course design is not only a comprehensive and systematic summary, consolidation and improvement but also a practical activity of the course content.

The game 2048 was created according to the characteristics of assembly language. During the course of the design, we became familiar with assembly language programming, subroutine calls, stacking techniques, register applications, etc. Through the course design, on the one hand, we improved our ability to solve practical problems in assembly language, and on the other hand, we made the students understand more deeply about the internal knowledge of computer systems, so as to lay a good foundation for future study and system development.

The group was divided as follows. Bu Mengyu and Wu Zhiqiang are responsible for UI designing and controlling; Chen Hongxu complete the algorithm of random number generation and block moving and merging algorithm; Fu Ze is responsible for the score calculating part and testing.

Key Words: Assembly; X86 Assembly; Windows API; MASM32

目　录

第 1 章 实验目的和内容

1. 1 实验目的

使用 X86 汇编语言进行团队编程实现 2048 游戏，增进学生对汇编语言基础语法的掌握程度，以及使用 Windows API、C 语言标准库、以及模块化编程等先进工具简化开发流程的能力。

1. 2 实验内容

通过汇编语言编程，实现简单的 2048 游戏。设计的游戏玩法如下：

- 进入游戏后，使用 W、A、S、D 键或箭头键移动一张 4×4 棋盘上的数字方块。相邻且数字一样的方块将会合并，新方块的数值将是原两个方块的数值之和。
- 当棋盘中的数字方块已填满且无合并方案时，弹窗提示玩家失败，游戏自动重新开始。
- 当棋盘中出现 2048 方块时，弹窗提示玩家胜利，玩家可以选择继续游玩或退出游戏。
- 计分板会实时显示玩家当前的最高分和历史最高分。

第 2 章 实验内容与步骤

2.1 界面设计

界面设计总共分成三个部分，一是游戏运行界面，三是游戏结束界面，三是游戏胜利界面。

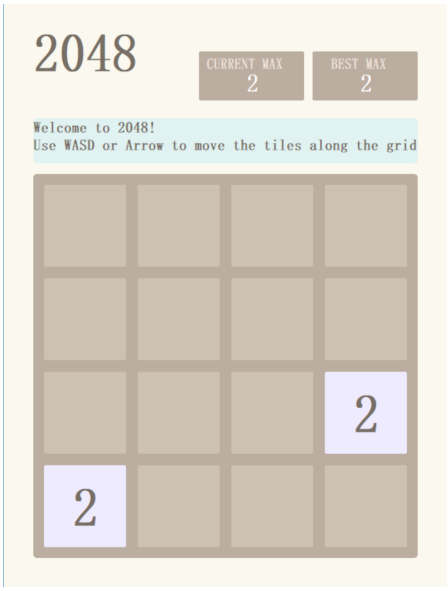

图 2-1 游戏运行界面

图 2-2 游戏结束界面

图 2-3 游戏胜利界面

2.1.1 整体思路

初始化 gameMat 为一个 4*4 的全局数组，每一个数组所对应的四个方格的内的数字，根据数组元素中值的不同，从而对不同方格绘制背景色。具体值对应的颜色 RGB 如下表所示。

表 2-1 统计表

数组的值	RGB
0	空白
2	#EEEBFF
4	#D2CDFF
8	#9A9AEF
16	#CD6090
32	#BC5A32
64	#8B58F8
128	#0093DB
256	#AB3E12
512	#AF143B
1024	#9A9319
2048	#00ABFF

流程图

整个游戏过程中，循环监听消息，并执行相应的回调函数，通过回调函数对 gameMat 数组的修改，并对界面进行重绘。

代码展示

```
invoke GameInit, @hWnd

.while TRUE

    invoke PeekMessage, addr @msg, NULL, 0, 0, PM_REMOVE ;等待消息
    .if  eax == 1
```

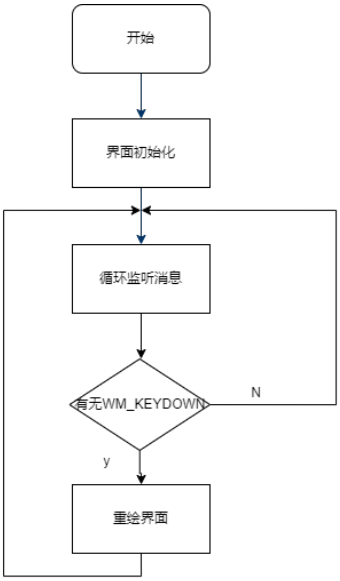

图 2-4 流程图

```
            invoke TranslateMessage, addr @msg
            invoke DispatchMessage, addr @msg
        .endif

        invoke GetTickCount
        sub eax,g_tPre

        .if eax>=50                                    ;每隔50ms更新图片
            ;.if bomb1_flag==1
            ;sub bomb1_time,1
            ;.endif
            invoke GamePaint,@hWnd
        .endif
.endw
```

2.1.2 初始化过程

使用循环初始化数组值，并使用 CreateFont 和 CreateSolidBrush API 创建字体和笔刷。值得注意的是 CreateSolidBrush 的参数传入 RGB 值时，需要注意是小端存储。比如：#EEEBFF invoke CreateSolidBrush, 0FFEBEEh

```
initFmt  proc   far  C uses  eax  esi  ecx  edx

        ;初始化gameMat为0
        mov ecx,16
        mov esi,0
L1:
        mov gameMat[esi*4],0
        inc  esi
        loop  L1

        ;初始化各个值
        mov gameIsEnd,0
        mov gameIsWin,0
        mov gameContinue,0
        mov score,0

        ;gameMat随机生成两个值
        INVOKE random32,dat,max
        INVOKE random32,dat,max
        ;初始化字体
        invoke  CreateFont，105，48，0，0，FW_BOLD, FALSE, FALSE, FALSE,
            DEFAULT_CHARSET, OUT_CHARACTER_PRECIS,
            CLIP_CHARACTER_PRECIS, DEFAULT_QUALITY, DEFAULT_PITCH
            or FF_DONTCARE, addr szFontName
    mov hFont0, eax
    invoke  CreateFont，20，8，0，0，FW_BOLD, FALSE, FALSE, FALSE,
```

```
        DEFAULT_CHARSET, OUT_CHARACTER_PRECIS,
        CLIP_CHARACTER_PRECIS, DEFAULT_QUALITY, DEFAULT_PITCH or
        FF_DONTCARE, addr szFontName
    mov hFont1, eax
    invoke CreateFont，26，11，0，0，FW_NORMAL, FALSE, FALSE, FALSE,
        DEFAULT_CHARSET, OUT_CHARACTER_PRECIS,
        CLIP_CHARACTER_PRECIS, DEFAULT_QUALITY, DEFAULT_PITCH or
        FF_DONTCARE, addr szFontName
    mov hFont2, eax
    invoke CreateFont，20，9，0，0，FW_BLACK, FALSE, FALSE, FALSE,
        DEFAULT_CHARSET, OUT_CHARACTER_PRECIS,
        CLIP_CHARACTER_PRECIS, DEFAULT_QUALITY, DEFAULT_PITCH or
        FF_DONTCARE, addr szFontName
    mov hFont3, eax
    invoke CreateFont，33，15，0，0，FW_BLACK, FALSE, FALSE, FALSE,
        DEFAULT_CHARSET, OUT_CHARACTER_PRECIS,
        CLIP_CHARACTER_PRECIS, DEFAULT_QUALITY, DEFAULT_PITCH or
        FF_DONTCARE, addr szFontName
    mov hFont4, eax
        invoke CreateFont，72，33，0，0，FW_BLACK, FALSE, FALSE, FALSE,
            DEFAULT_CHARSET, OUT_CHARACTER_PRECIS,
            CLIP_CHARACTER_PRECIS, DEFAULT_QUALITY, DEFAULT_PITCH
            or FF_DONTCARE, addr szFontName
    mov hFontDigit1，eax
    invoke CreateFont，59，27，0，0，FW_BLACK, FALSE, FALSE, FALSE,
        DEFAULT_CHARSET, OUT_CHARACTER_PRECIS,
        CLIP_CHARACTER_PRECIS, DEFAULT_QUALITY, DEFAULT_PITCH or
        FF_DONTCARE, addr szFontName
    mov hFontDigit3，eax
    invoke CreateFont，46，21，0，0，FW_BLACK, FALSE, FALSE, FALSE,
```

```
        DEFAULT_CHARSET, OUT_CHARACTER_PRECIS,
        CLIP_CHARACTER_PRECIS, DEFAULT_QUALITY, DEFAULT_PITCH or
        FF_DONTCARE, addr szFontName
    mov hFontDigit4,  eax
        ；初始化笔刷
        invoke  CreateSolidBrush，0FFEBEEh
    mov hBrush_d2, eax
    invoke  CreateSolidBrush，0FFCDD2h
    mov hBrush_d4, eax
    invoke  CreateSolidBrush，0EF9A9Ah
    mov hBrush_d8, eax
    invoke  CreateSolidBrush，09060cdh
    mov hBrush_d16, eax
    invoke  CreateSolidBrush，0325abch
    mov hBrush_d32, eax
    invoke  CreateSolidBrush，00f858bh
    mov hBrush_d64, eax
    invoke  CreateSolidBrush，0db9300h
    mov hBrush_d128, eax
    invoke  CreateSolidBrush，0123eabh
    mov hBrush_d256, eax
    invoke  CreateSolidBrush，03b14afh
    mov hBrush_d512, eax
    invoke  CreateSolidBrush，001939ah
    mov hBrush_d1024, eax
    invoke  CreateSolidBrush，0ffab00h
    mov hBrush_d2048, eax
        ret
initFmt  endp
```

2.1.3 绘制界面

总体流程

首先调用 GetDC 获得设备环境句柄，便于画图；因为绘图对象是位图，需要创建一个与位图句柄相联系的 DC，也就是用函数 CreateCompatibleDC 来创建与目标句柄兼容的临时内存；再创建一个位图句柄 CreateCompatibleBitmap，之后采用 SelectObject 函数选入到临时内存 DC 中，表示之后使用该临时内存对象时都是位图对象。

之后 CreateSolidBrush 选中笔刷，并且使用 Rectangle 绘制矩形背景；背景绘制完毕后，使用 TextOut 绘制文字，文字样式通过 SelectObject 选择相应 hFont。

在计算坐标位置时，使用了大量的 local 变量，使程序可读性增加。

相关代码示例：（这里以绘制 4*4 方格为例）

```
；4∗4数字位置
mov @boardStartX, (WINDOW_WIDTH − BOARD_EDGE) / 2 − WINDOW_BIAS
mov @boardStartY, 225
mov @boardEndX, (WINDOW_WIDTH + BOARD_EDGE) / 2 − WINDOW_BIAS
mov @boardEndY, BOARD_EDGE + 225
invoke  CreateSolidBrush，0b4c1cdh
mov @bghDc, eax

mov eax, @boardStartY
add eax，PATH_WIDTH
mov @cellStartY, eax
add eax，CELL_EDGE
mov @cellEndY, eax

mov ecx, 0
.while  ecx < 4
        mov eax, @boardStartX
        add eax，PATH_WIDTH
```

```
        mov @cellStartX, eax
        add eax, CELL_EDGE
        mov @cellEndX, eax

        push ecx
        mov ecx, 0
        .while ecx < 4
                push ecx
                invoke SelectObject, _hDc, @bghDc
                invoke RoundRect, _hDc, @cellStartX, @cellStartY, @cellEndX,
                    @cellEndY, 3, 3
                pop ecx
                add @cellStartX, CELL_EDGE + PATH_WIDTH
                add @cellEndX, CELL_EDGE + PATH_WIDTH
                inc ecx
        .endw
        pop ecx
        inc ecx
        add @cellStartY, CELL_EDGE + PATH_WIDTH
        add @cellEndY, CELL_EDGE + PATH_WIDTH
.endw

mov esi, offset gameMat
; 生成静态数字
mov @y, 0
        .while @y < 4
                mov @x, 0
                .while @x < 4
                        mov edi, esi
                        mov eax, @y
```

```
                shl  eax，4
                add  edi，eax
                mov eax, @x
                shl  eax，2
                add  edi，eax
                invoke  _GetPosition，@boardStartX, @boardStartY, @x, @y
                invoke  _DrawDigit, _hDc, eax，ebx，DWORD ptr [edi]
                inc  @x
            .endw
            inc  @y
        .endw
invoke  BitBlt，hDc, 0,  0,  WINDOW_WIDTH, WINDOW_HEIGHT, _hDc, 0, 0,
    SRCCOPY
invoke  DeleteObject，@bghDc
invoke  DeleteObject，_cptBmp
invoke  DeleteDC, _hDc
popad
```

工具类子程序 - 获取数字框位置

根据方格左上角的坐标来计算下一个方格的 (X,Y) 坐标，最终保存在 eax（X 坐标）,ebx（Y 坐标）中。具体代码如下：

```
;参数 _StartX，_StartY，_dX, _dY
local  @RetX, @RetY

mov eax, _StartX
mov @RetX, eax
add @RetX, PATH_WIDTH
mov eax, _StartY
mov @RetY, eax
```

```
add @RetY, PATH_WIDTH

.while _dX > 0
        add @RetX, CELL_EDGE
        add @RetX, PATH_WIDTH
        dec _dX
.endw
.while _dY > 0
        add @RetY, CELL_EDGE
        add @RetY, PATH_WIDTH
        dec _dY
.endw

mov eax, @RetX
mov ebx, @RetY
ret
```

工具类子程序 - 绘制数字

根据数字数值不同，绘制不同样式的数字，样式主要指字体颜色，背景颜色，和字体大小，如多位数字之间的间距大小即单字的高宽。字体与背景颜色按照数字来分，2-2048 每种背景颜色均不同；数字间距和单字宽高按照数字位数 1-4 各不同。具体代码如下：

```
;参数_hDc, _cellStartX，_cellStartY，_num

local @cellEndX, @cellEndY
local @digitStartX，@digitStartY
local @szBuffer[10]: byte

. if _num == 0
```

```
            ret
. endif

pushad

mov eax,  _cellStartX
mov @cellEndX, eax
add @cellEndX, CELL_EDGE
mov eax,  _cellStartY
mov @cellEndY, eax
add @cellEndY, CELL_EDGE

; 设置数字背景颜色
. if  _num == 2
            invoke  SelectObject ,  _hDc, hBrush_d2
. elseif  _num == 4
            invoke  SelectObject ,  _hDc, hBrush_d4
. elseif  _num == 8
            invoke  SelectObject ,  _hDc, hBrush_d8
. elseif  _num == 16
            invoke  SelectObject ,  _hDc, hBrush_d16
. elseif  _num == 32
            invoke  SelectObject ,  _hDc, hBrush_d32
. elseif  _num == 64
            invoke  SelectObject ,  _hDc, hBrush_d64
. elseif  _num == 128
            invoke  SelectObject ,  _hDc, hBrush_d128
. elseif  _num == 256
            invoke  SelectObject ,  _hDc, hBrush_d256
. elseif  _num == 512
```

```
        invoke  SelectObject ,  _hDc, hBrush_d512
. elseif  _num == 1024
        invoke  SelectObject ,  _hDc, hBrush_d1024
. else
        invoke  SelectObject ,  _hDc, hBrush_d2048
. endif
invoke  RoundRect, _hDc,  _cellStartX ,   _cellStartY ,  @cellEndX, @cellEndY, 3, 3

; 设置数字字体颜色
. if  _num < 8
        invoke  SetTextColor,  _hDc, hColor_d1
. else
        invoke  SetTextColor,  _hDc, hColor_d2
. endif

mov eax,  _cellStartX
mov @digitStartX,  eax
mov eax,  _cellStartY
mov @digitStartY,  eax

; 设置数字位置
. if  _num < 10
        invoke  SelectObject ,  _hDc, hFontDigit1
        add @digitStartX，37
        add @digitStartY，18
. elseif  _num < 100
        invoke  SelectObject ,  _hDc, hFontDigit1
        add @digitStartX，21
        add @digitStartY，18
. elseif  _num < 1000
```

```
        invoke  SelectObject , _hDc, hFontDigit3
        add @digitStartX，13
        add @digitStartY，24
. else
        invoke  SelectObject , _hDc, hFontDigit4
        add @digitStartX，12
        add @digitStartY，31
. endif

invoke  wsprintf，addr @szBuffer, addr szFormat, _num
invoke TextOut, _hDc, @digitStartX，@digitStartY，addr @szBuffer, eax

popad
ret
```

2. 2 界面控制逻辑

界面绘制函数为 WinMain 函数，游戏界面控制函数为 ProcWinMain 函数。控制逻辑如图所示。

2. 2. 1 WinMain 函数

绘制界面调用了 Windows32 提供的系统 API。首先需要初始化一个主窗口 WinMain，包括初始化窗口参数、注册窗口、创建窗口、显示窗口、更新窗口、循环处理窗口消息。对 Windows32 提供的部分与创建窗体相关的 API 及数据结构的查询信息如图所示。

基于以上资料，编写程序如下：

```
WinMain proc  ;窗口程序
        local  @stWndClass:WNDCLASSEX ;定义了窗口的一些主要属性
        local  @stMsg:MSG                                   ;消息传递

        ;获取应用程序的句柄，把该句柄的值放在hInstance
```

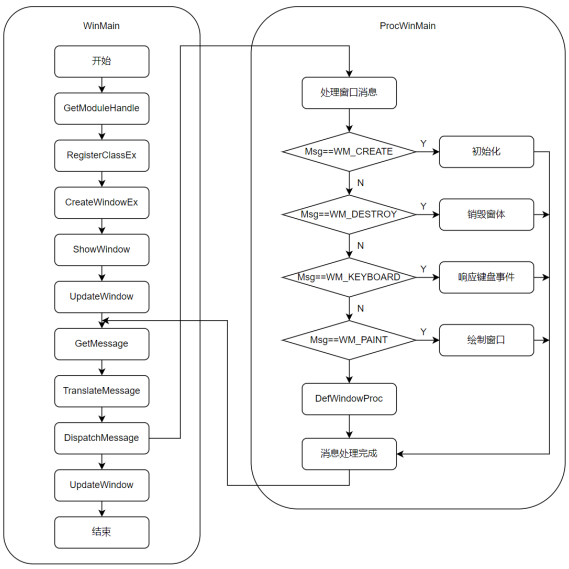

图 2-5 流程图

windowsAPI	类型	功能
GetModuleHandle	函数	获取程序句柄
GetCommandLine	函数	获取程序的命令行参数然后以参数的形式传递给WinMain函数
RegisterClassEx	函数	注册窗体
CreateWindowEx	函数	创建窗体
HINSTANCE	变量	程序句柄
LPSTR	变量	运行时参数，命令行内容
WNDCLASSEX	变量	窗口类，描述、编辑、播报窗口
MSG	变量	获取和设置消息的id号
HWND	变量	检索窗口的窗口句柄

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nShowCmd)

- hinstance - 该参数为Windows 为应用程序生成的实例句柄。实例是用来跟踪资源的指针
- hprevinstance 当前已不再使用该参数，之前用来跟踪应用程序的前一个实例，即程序的父亲的程序实例
- ipcmdline - 一个以NULL结尾的字符串，类似于标准main(int argc,char** argv)中的命令行参数，但没有单独的argc来指示命令行参数数量
- ncmdshow—— 最后一个参数是启动期间传递给应用程序的一个整数，指出如何打开程序的窗口。事实上，基本没用，但ncmdshow的值一般会用于ShowWindows()中使用，用来设置新建子窗口的形式

图 2-6 创建窗体相关 API

```
invoke  GetModuleHandle,NULL
mov hInstance,eax

invoke  RtlZeroMemory,addr @stWndClass,sizeof @stWndClass ;将stWndClass
    初始化全0

;注册窗口类
invoke  LoadCursor,0,IDC_ARROW
mov @stWndClass.hCursor,eax
push  hInstance
pop @stWndClass.hInstance
mov @stWndClass.cbSize,sizeof WNDCLASSEX
mov @stWndClass.style,CS_HREDRAW or CS_VREDRAW
mov @stWndClass.lpfnWndProc,offset _ProcWinMain
mov @stWndClass.hbrBackground,COLOR_WINDOW+1
mov @stWndClass.lpszClassName,offset szClassName

;注册窗口类，注册前先填写参数WNDCLASSEX结构
invoke  RegisterClassEx,addr  @stWndClass

;创建窗口并将句柄放在hWinMain中
invoke  CreateWindowEx,WS_EX_CLIENTEDGE,\
                    offset  szClassName,offset  szCaptionMain,\
                    WS_OVERLAPPEDWINDOW,400,50,WINDOW_WIDTH,
                        WINDOW_HEIGHT,\
                    NULL,NULL,hInstance,NULL
mov hWinMain,eax

;显示窗口
invoke  ShowWindow,hWinMain,SW_SHOWNORMAL
```

```
    ;刷新窗口客户区
    invoke UpdateWindow,hWinMain

    ;进入无限的消息获取和处理的循环
    .while TRUE
        invoke GetMessage,addr @stMsg,NULL,0,0 ;从消息队列中取出第一
            个消息，放在stMsg结构中
        .break .if eax==0
            ;如果是退出消息，eax将会置成0，退出循环
        invoke TranslateMessage,addr @stMsg          ;将基于键盘扫描
            码的按键信息转换成对应的ASCII码，如果消息不是通过键盘
            输入的，这步将跳过
        invoke DispatchMessage,addr @stMsg           ;这条语句的作用
            是找到该窗口程序的窗口过程，通过该窗口过程来处理消息
    .endw
    ret
WinMain endp
```

2. 2. 2 ProcWinMain 函数

创建窗体后需要添加窗体元素，并响应窗体事件，执行不同动作。对 Windows32 提供的部分与窗体事件响应相关的 API 及数据结构的查询信息如图所示。

ProcWinMain 函数需要响应的消息类型包括 WM_CREATE、WM_CLOSE、WM_KEYBOARD、WM_PAINT，函数检测到对应消息后会执行不同动作。

窗口创建时产生 WM_CREATE 消息，初始化相关参数，如创建笔刷、字体，生成随机数，分数、界面清零等等。

窗口关闭时产生 WM_CLOSE 消息，销毁窗体并通知主程序窗口关闭。

敲击键盘时发出 WM_KEYBOARD 消息，将按键值存放在 wParam 中，支持的按键控制为“WASD”、“wasd”、“上下左右”三种。根据不同方向执行不同的 move 函数，并通过 InvalidateRect 函数发出 WM_PAINT 消息绘制更新后的窗口。

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)

窗口过程函数决定了当一个窗口从外界接收到不同的信息时，所采取的不同反应，即主要用于处理发送给窗口的信息

- hwnd是要处理窗口的句柄
- message是消息ID，代表了不同的消息类型
  - WM_DESTROY：窗体摧毁消息
  - WM_KEYDOWN：键盘消息
  - WM_PAINT：客户区重绘消息 BeginPaint
- wParam和lParam代表了消息的附加信息，附加信息会随着消息类型的不同而不同
  - wParam的值为按下按键的虚拟键码
  - lParam则存储按键的相关状态信息
  - 当鼠标消息发出时，wParam值为鼠标按键的信息，而lParam则储存鼠标的坐标，高字节代表y坐标，低字节代表x坐标，g_y =HIWORD(lParam)，g_x = LOWORD(lParam)

图 2-7 窗体事件响应相关 API

需要重新绘制窗口时发出 WM_PAINT 消息。本代码为支持自定义的字体、颜色、背景框等内容，采用绘制窗口的方式更新窗口内容，而不是常见的、利用 CreateWindowEx 函数创建窗体元素并保存句柄，通过句柄访问窗体。收到 WM_PAINT 消息时，程序取出该消息，并调用 DrawGame 函数绘制界面。

除此之外，还需要调用默认处理方法 DefWindowProc 函数响应其他所有信息。

基于以上控制逻辑，设计主窗口代码如下所示。

```
ProcWinMain proc uses ebx  edi   esi ,hWnd,uMsg,wParam,lParam ;窗口过程
        local  @stPs:PAINTSTRUCT
        local  @stRect:RECT
        local  @hDc
        LOCAL @oldPen:HPEN
        local  @hBm

        mov eax,uMsg  ;uMsg是消息类型，如下面的WM_PAINT,WM_CREATE

        . if  eax==WM_CREATE ;创建窗口
                invoke  initFmt
```

```
. elseif  eax==WM_CLOSE ;窗口关闭消息
          invoke  DestroyWindow,hWinMain
          invoke  PostQuitMessage,NULL

. elseif  eax== WM_KEYDOWN  ;WM_CHAR为按下键盘消息，按下的键
    的值存在wParam中
          MOV edx,wParam
          ;如果为W则向上移动
          . if  edx == 'W' ||  edx == 'w' ||  edx == VK_UP

                    invoke  moveW
                    ;如果可以向上移动，在随机位置产生一个2
                    .IF  changedW == 1
                              invoke  random32
                    . endif
                    ;计算分数
                    ;INVOKE getScore
                    ;更新界面
                    ;INVOKE UpdataGame,hWnd
                    invoke   InvalidateRect ,  hWnd, NULL, FALSE ; 更新窗体
          ;同上
          . elseif  edx == 'S' ||  edx == 's' ||  edx == VK_DOWN
                    invoke  moveS

                    .IF  changedS == 1
                              invoke  random32
                    . endif
                    ;INVOKE getScore
                    ;NVOKE UpdataGame,hWnd
```

```
        invoke  InvalidateRect , hWnd, NULL, FALSE
;同上
. elseif  edx ==’A’ || edx == ’a’ || edx == VK_LEFT

        invoke moveA

        .IF changedA == 1
                invoke random32
        . endif
        ;INVOKE getScore

        invoke  InvalidateRect , hWnd, NULL, FALSE
;同上
. elseif  edx == ’D’ || edx == ’d’ || edx == VK_RIGHT
        invoke moveD
        .IF changedD == 1
                invoke random32
        . endif

        invoke  InvalidateRect , hWnd, NULL, FALSE
. endif

;如果游戏还未获胜，gameContinue=0，如果游戏已经获胜过了，
    且玩家选择继续玩，则gameContinue=1，将不会再弹出获胜消
    息
;. if  gameContinue == 0
        ;如果gameIsWin==1，游戏获胜，弹出游戏获胜消息
;       . if  gameIsWin == 1
;               invoke gameWin
;       . endif
```

```
        ;. endif

        ;判断游戏是否结束
        invoke  gameEnd
        ;如果游戏结束，绘制失败弹窗
        . if  gameIsEnd == 1
                invoke  MessageBox,hWinMain,offset szText7, offset   szText6
                    ,MB_OK
                ;重新开始游戏
                . if  eax  == IDOK
                        invoke  initFmt
                        invoke   InvalidateRect ，hWnd, NULL, FALSE
                . endif
        . endif

. elseif   eax==WM_PAINT ;如果想自己绘制客户区，在这里些代码，即第
    一次打开窗口会显示什么信息


        invoke  BeginPaint,hWnd,addr @stPs
        mov @hDc, eax

        ;绘制界面
        invoke  DrawGame,hWnd, @hDc

        invoke  EndPaint,hWnd,addr @stPs
. else    ;否则按默认处理方法处理消息
        invoke  DefWindowProc,hWnd,uMsg,wParam,lParam
        ret
. endif
```

```
        xor  eax,eax
        ret
ProcWinMain endp
```

2.3 随机数生成算法

该部分对应最终代码的 random32 函数，用于在矩阵的随机位置随机生成 2 或 4。

随机数算法采用线性同余法，公式为 $x_{n+1} = (a \times x_n + c)\ mod\ MAX$。迭代使用该公式，每次计算出的 x_{n+1} 即为本次的随机数，$x_0 = seed$。在 InitFmt 函数中调用 GetTickCount 函数，获得当前时间作为随机数种子 seed。Max 是随机数的模数，由于是 4×4 的矩阵，模数取为 16。参数 a 和 c 参考 msvc 运行时环境中的设计，a 取 0343FDH，c 取 269EC3H，使得伪随机数尽量满足随机性要求。随机数算法对应代码中的 lcg: 之后的代码块。

每次使用上述随机数算法都连续调用两次，产生两个随机数，第一个数表示新出现方格的位置，第二个数如果是奇数则在新方格上填 4，如果是偶数则填 2。考虑方格上可能已经有数，需要循环上述过程，直到找到一个空闲的方格为止。循环找空闲方格的函数对应代码中的 loop_lcg: 标号之后的代码块；找到空闲方格后跳转到 inital_mat: 标号位置，在矩阵对应位置填上对应的数。

由于 random32 函数在一次游戏中可能被调用多次，每次调用时应该要从上一次生成的最后一个随机数继续往下生成，因此，每次调用 random32 函数结束时，将当前 eax 的值（即最近一次生成的随机数）保存到 seed 中，下一次再从 seed 开始迭代，就能使得每一次调用都是接着上一次调用继续往下生成。

2.4 方块移动与合并算法

该部分对应最终代码的 moveW, moveS, moveA, moveD 函数。四个函数分别处理向上、向下、向左、向右移动，基本逻辑相似，以 moveW 为例说明。

在处理方格移动的函数中主要需要完成两件事：一是判断方格移动方向上有没有 0，如果有则移动方格；二是判断方格移动方向上相邻的方格是不是与当前方格相同，如果是则合并。需要循环所有 16 个方格对这两个移动条件进行判断，因此需要使用两层循环。对于向上移动来说，第 1 行只需要考虑合并而不需要考虑移动到空方

格上，2-4 行需要先考虑移动到空方格上，在考虑合并的问题，为了复用代码，将合并的过程定义为 w_merge 标号，2-4 行判断完移动空方格之后跳转到 w_merge 所在位置，判断能否合并。判断合并和判断空方格的函数逻辑较为简单，分别是检查相邻方块是否相同以及检查是否为 0。如果能够合并或移动到空方格上，则将 changedW 标志置为 1，其他模块可以根据这一标志位判断需要采取的操作，比如在空方格上出现新的数或者在无法移动时结束游戏。

由于代码的跳转逻辑较为复杂，为了加强代码可读性和可维护性，代码中尽量利用了.while 和.endw 这一组伪指令以及.if，.elseif，.endif 这一组伪指令来实现类似 c 语言的控制结构。

2.5 分数计算算法

该部分对应最终代码中的 getScore 函数。

getScore 函数分析当前“棋盘” gameMat 的局面，计算玩家获得的分数，并以此为依据，判断玩家是否已经成功拼出 2048 方块赢得游戏，为后续显示胜利信息等功能提供所需的信息。鉴于大部分玩家较为关心场上已拼出的分值最高的方块，getScore 计算的正是这个值。该函数在 DrawGame 函数中调用，在每一次需要重绘游戏界面时，都计算一次分数，以防分数运算不及时；运行时，其将依次遍历数据段中定义的“棋盘”——gameMat 的 16 个值，并找出最大的值存储于数据段中的 score 中。接下来，函数对 score 的值进行判断。如果 score 等于 2048，说明玩家已经赢得游戏，将数据段中的 gameIsWin 字段更改为 1，供其他过程引用。

具体代码如下所示。

```
getScore  proc
    push ecx
    push edx
        push  esi
        push  edi

    xor  ecx，ecx
    xor  eax,eax
        mov score，0
```

```
        ;遍历矩阵
        mov ecx, 16  ; 总共有16个格子
        mov edi, 0  ; 初始时，最大值设置为0
        .while ecx > 0
                dec ecx
                mov esi, gameMat[ecx * (sizeof dword)]
                .if esi > edi
                        mov edi, esi
                .endif
        .endw
        mov score, edi
        .if score == 2048
                mov gameIsWin, 1
        .endif

        pop edi
        pop esi
        pop edx
    pop ecx
    ret
getScore        Endp
```

第 3 章 实验感想

本次实验是我们第一次使用汇编语言编写游戏。因为对汇编指令与程序结构不够熟悉，刚开始编写时难以和书本上的知识联系起来。后来在与同学交流和网上查阅资料后，小组成员对汇编指令、汇编函数编写、参数传递机制、汇编调用 C 语言、汇编调用 Windows 提供的界面 API、窗口绘制、消息响应机制等等有了较为深入的理解。本次实验组员们的心得体会主要有以下几点。

- 对汇编语言基础知识有了更深入的理解，包括对汇编指令调用、汇编浮点指令、汇编函数编写、函数参数传递、汇编调用 C 语言函数等等。
- 掌握汇编语言环境配置方法，能够编写 Windows 风格的汇编语言应用程序。
- 掌握其他窗口绘制方法。一种是使用 Windows 提供的 CreateWindowEx 函数创建窗口元素，使用句柄控制，这种方法的缺点无法自定义字体、颜色等格式。另一种是使用窗口绘制方式更新窗口，每次窗口元素变化时重新绘制，这样可以实现对窗口元素的风格化显示。此外，还有位图控制等方式控制窗口。

通过本次实验，小组成员也对整个计算机系统的结构也有了更深刻的认识，从应用软件，操作系统，到硬件的组成相互协调，这是一个密不可分的整体，对其中的每个环节每个层次都有深入的了解学习之后，才能在今后的学习实践中更加得心应手，做出更有质量的成果。

最后，十分感谢老师一学期认真负责的讲授。您清晰的讲解使我们受益匪浅，几个实验也使我们完成了从理论到实践的落实，使我们较好地掌握了汇编语言与接口技术的知识。