北京理工大學

汇 编 语 言 与 接 口 技 术

组 队 实 验 报 告

2048 游戏实验报告

Report on the experiment of 2048 game

学　　院：计算机学院

专　　业：计算机科学与技术

班　　级：07112105、07112106

07812101

姓　　名：范曾 1120212696

洪子翔 1820211062

张栋梁 1120213591

何秉翰 1820211061

指导教师：张全新

目录

第一章 实验目的

通过实验，学习32位汇编程序的指令系统，充分了解数据运算指令、程序控制指令、处理机控制指令等指令的使用方式。

第二章 实验内容

采用32位汇编程序编写一个游戏程序，如俄罗斯方块、贪吃蛇、扫雷、简单射击类游戏等。本小组选择实现“2048”小游戏。本游戏规则很简单，每次可以选择上下左右其中一个方向去滑动，每滑动一次，所有的数字方块都会往滑动的方向靠拢外，系统也会在空白的地方乱数出现一个数字方块，相同数字的方块在靠拢、相撞时会相加。系统给予的数字方块不是2就是4，玩家要想办法在这小小的16格范围中凑出“2048”这个数字方块。

第三章 实验环境

操作系统：Windows10

编辑器：Visual Studio 2019

编译器：masm32

第四章 游戏介绍

4.1. 游戏规则介绍

游戏主界面是一个4×4的方块，界面中存在任意数值为 2 的整数次幂的方块， 玩家可以通过WASD 四个按键来使得界面中所有的方块同时向上下左右中的某一个方向移动，而两个数值相同的方块碰到一起则会合并成一个新的方块，其数值是原来两个数值的加和，也就是原来的2倍，并且每次移动都会在空闲地方产生一个新的2方块。如果玩家最终合成了数值为2048的方块， 那

么玩家取得了胜利；如果玩家填满了所有的16个空闲块，且没有方块能被消除，以及没有产生2048块，那么我们认为玩家游戏失败。

4.2. 空白游戏界面

图 1 空白游戏界面展示

游戏界面采用渐变色设计，并加入手写2048，使得界面富有趣味性，右上角清晰显示当前分数，在分数下方简单介绍规则，方便玩家快速上手。

4.3. 游戏图标展示

图 2 游戏图标

卡通游戏图标，富有趣味性。

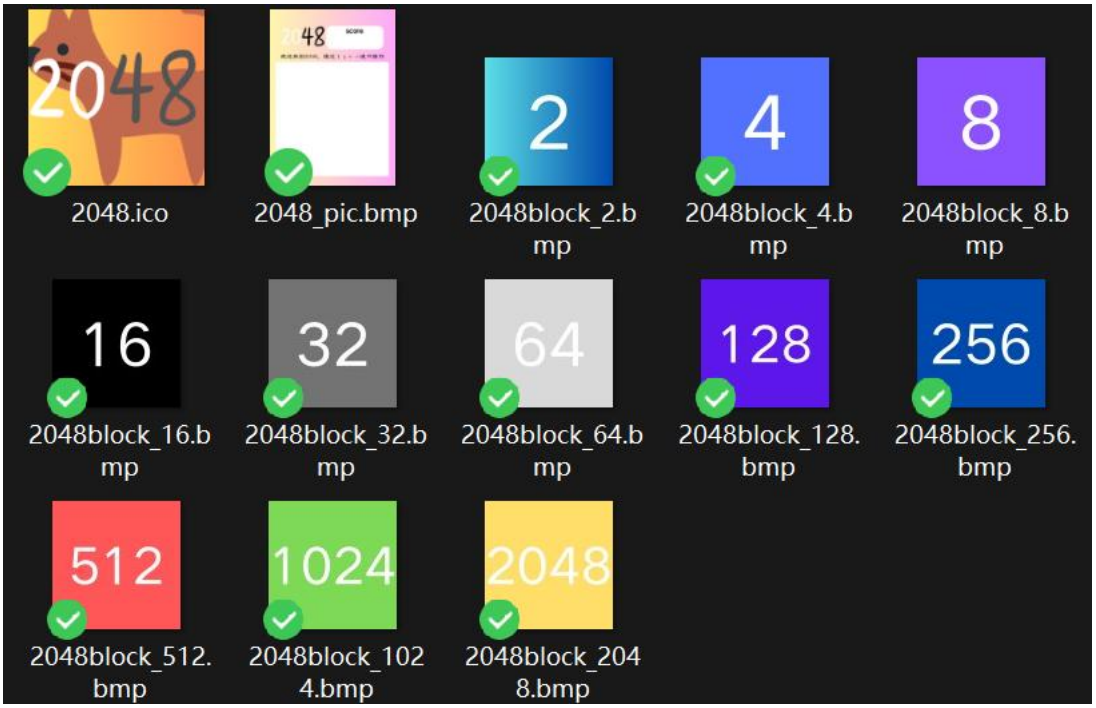

图 3 游戏方块设计

4.4. 游戏主界面

图 4 游戏初始主界面

游戏主界面中，有4×4个方格，采用白色条纹区分，每个方块均有不同颜色，方便玩家醒目区分。

图 5 各个方块

4.5. 游戏操控

玩家控制← ↑ ↓ →或者WASD来操控游戏，控制方块整体向左、向右、向上、向下移动。

图 6 某次操作前

在对以上进行向下、向左操作后，如下图所示：

图 7 某次操作后

图三中横着两个2因为向左操作被合并，竖着两个2因为向下操作被合并。

4.6. 游戏胜利

图 8 游戏胜利——出现2048

此时弹出对话框，玩家可选择是否继续游戏，或者退出游戏。

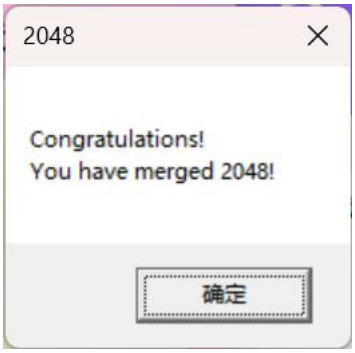

图 9 胜利对话框-1

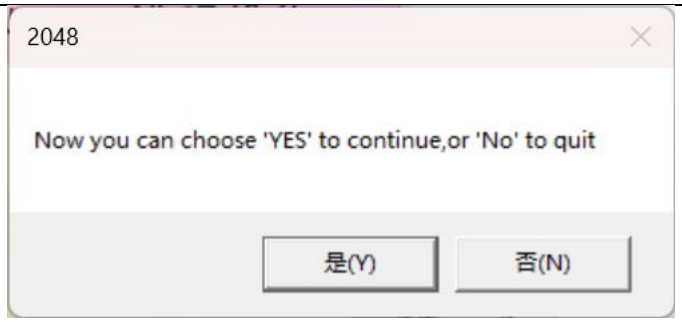

图 10 胜利对话框-2

图 11 选择继续游玩

选择继续游玩，可以继续完成挑战。

4.7. 游戏失败

图 12 游戏失败

此时弹出对话框，显示游戏失败，并且自动重新开始新的一局。

第五章 实验设计

5.1. 总体设计

本项目代码由以下几部分组成：初始化函数，随机生成方块，WASD 移动函数，判断成功/失败函数，窗口回调函数，绘制界面函数，窗口显示函数，重新启动游戏函数等等。

5.2. main函数

```
main proc
invoke ReStartGame
call _WinMain  ; 主程序调用窗口程序和结束程序
invoke ExitProcess,NULL
ret
main endp
```

Main函数负责调用窗口程序，以及结束程序。

5.3. gameWin函数

```
gameWin         PROC
 INVOKE   MessageBox,hWinMain, OFFSET  szWinText, OFFSET szText6,
   MB_OK
 .IF          EAX == IDOK
 INVOKE   MessageBox,hWinMain, OFFSET szText12, OFFSET szText6,
   MB_YESNO
 .IF          EAX == IDYES
 MOV          gameContinue, 1
 .ELSEIF      EAX == IDNO
 INVOKE   DestroyWindow,hWinMain
 INVOKE   PostQuitMessage,NULL
.ENDIF
.ENDIF
RET
gameWin         ENDP
```

这段代码实现了gameWin的过程，游戏胜利时，gameWin置1，弹出游戏胜利提示消息，玩家可选择继续游戏或结束游戏。该函数绘制游戏胜利消息弹窗，同时弹出选则框以供玩家选择是否继续游戏。调用了user32中的MessageBox函数，通过控制“MB_OK”参数以及“MB_YESNO”参数控制弹出框提供的按钮类型。对于选择框而言，若玩家选择了 Yes 按钮，则将 gameContinue 置为1。之后销毁窗口，结束函数。

5.4. WinMain函数

```
_WinMain proc  ; 窗口程序
```

```
2.  local @stWndClass :WNDCLASSEX  ; 定义WNDCLASSEX型结构变量，定义了窗口的一些主要属性
3.  local @stMsg:MSG   ; 定义stMsg，类型是MSG，用来传递消息
4.
5.  invoke GetModuleHandle,NULL  ; 得到应用程序的句柄
6.  mov hInstance,eax  ; 把句柄的值存入hInstance
7.  invoke RtlZeroMemory,addr @stWndClass,sizeof @stWndClass  ; 将stWndClass初始化为0
8.
9.  invoke LoadCursor,0,IDC_ARROW
10. mov @stWndClass.hCursor,eax
11. INVOKE   LoadIcon, hInstance, 108
12. MOV          @stWndClass.hIcon, EAX
13. MOV          EAX, 250 + 248 * 100H + 239 * 10000H
14. INVOKE   CreateSolidBrush, EAX
15. MOV          @stWndClass.hbrBackground, EAX
16. push hInstance
17. pop @stWndClass.hInstance
18. mov @stWndClass.cbSize,sizeof WNDCLASSEX        ; 初始化stWndClass结构中表示窗口的各种属性的值
19. mov @stWndClass.style,CS_HREDRAW or CS_VREDRAW
20. mov @stWndClass.lpfnWndProc,offset _ProcWinMain ; 指定该窗口
21. mov @stWndClass.lpszClassName,offset szClassName
22. invoke RegisterClassEx,addr @stWndClass  ; 使用完成初始化的stWndClass注册窗口
23.
24. invoke CreateWindowEx,WS_EX_CLIENTEDGE, \   ; 建立窗口
25.       offset szClassName, offset szCaptionMain, \
26.       WS_OVERLAPPEDWINDOW, 400, 200, 500, 710, \
27.       NULL,NULL,hInstance, NULL
28.       ; szClassName是建立窗口使用的类名字符串指针，此处为“MyClass”
29. mov hWinMain,eax  ; 将窗口句柄存入hWinMain
30.
31.    invoke ShowWindow,hWinMain,SW_SHOWNORMAL  ; 使用窗口的句柄显示窗口
32. invoke UpdateWindow,hWinMain  ; 刷新窗口
33.
34. .while TRUE  ; 进入消息获取和处理的循环
35.    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
36.    .break .if eax==0  ; 如果是退出消息，eax将会置成0，退出循环
```

```
37.    invoke TranslateMessage,addr @stMsg  ; 将获取的键盘输入转换为
    ASCII码
38.    invoke DispatchMessage,addr @stMsg  ; 调用该窗口程序的窗口过程
    处理消息
39. .endw
40. ret
41._WinMain endp
```

该函数作用为注册主窗口。首先是分别定义了两个变量，WNDCLASSEX型结构变量，定义了窗口的一些主要属性，定义stMsg，类型是MSG，用来传递消息。调用系统函数，获得应用程序的句柄并存入， 然后初始化上面的第一个结构变量，即初始化stWndClass结构中表示窗口的各种属性的值，再指定窗口程序为我们的ProWinMain函数。

5.5. ProcWinMain函数

```
1. _ProcWinMain proc uses ebx edi esi,hWnd,uMsg,wParam,lParam;窗口
    回调函数，处理窗口消息
2.  LOCAL     @stPs :PAINTSTRUCT
3.  LOCAL     @hBm :DWORD
4.  LOCAL     @hDc :DWORD
5.
6.  mov     eax, uMsg ; uMsg是消息类型，如下面的WM_PAINT,WM_CREATE
7.
8.  .IF eax == WM_PAINT   ; 自定义绘制客户区，即第一次打开窗口会显示什么
    信息
9.      mov       ebx, wParam
10.     .if       ebx != 1
11.         invoke  BeginPaint, hWinMain, ADDR @stPs
12.         invoke  GetDC, hWnd
              ; 首先获取窗口DC
13.         mov      @hDc, eax
14.         invoke  CreateCompatibleDC, @hDc
              ; 创建兼容窗口DC的缓存dc
15.           mov     hdcIDB_BITMAP1, eax
16.           invoke  CreateCompatibleBitmap, @hDc, 480, 670
                      ; 创建位图缓存
17.           mov     hbmIDB_BITMAP1, eax
18.           invoke  SelectObject, hdcIDB_BITMAP1, hbmIDB_BITMAP1
                   ; 将hbm与hdc绑定
```

```
            invoke  LoadBitmap, hInstance, 107
                 ; 载入位图到位图句柄中
            mov     @hBm, eax
            invoke  CreatePatternBrush, @hBm
              ; 创建以位图为图案的画刷
            push eax
            invoke  SelectObject,hdcIDB_BITMAP1, eax
                 ; 以画刷填充缓存DC
            invoke  PatBlt, hdcIDB_BITMAP1, 0, 0, 480, 670,
    PATCOPY                ; 按照PATCOPY的方式
            pop     eax
            invoke  DeleteObject, eax
              ; 删除画刷
            invoke  BitBlt, @hDc, 0, 0, 480, 670, hdcIDB_BITMAP1
    , 0, 0, SRCCOPY ; 在主窗口DC上绘制位图dc
            invoke  DeleteDC, @hDc
            invoke  DeleteDC, hdcIDB_BITMAP1
            invoke  DeleteObject, hbmIDB_BITMAP1
            invoke  DeleteObject, @hBm
            invoke  EndPaint, hWnd, ADDR @stPs
     .endif
     invoke   updateBlock

.elseif eax == WM_CLOSE  ; 窗口关闭消息
    invoke DestroyWindow, hWinMain
    invoke PostQuitMessage, NULL
.elseif eax == WM_CREATE  ; 创建窗口
    ; 绘制界面
    invoke DrawGame, hWnd
.elseif eax == WM_KEYDOWN    ;WM_KEYDOWN为按下键盘消息，按下的键的
    值存在wParam中
    mov edx, wParam
    ; 如果为W或方向键上则向上移动
    .if edx == "W" || edx == VK_UP
       invoke movUP
       ; 如果可以移动，在随机位置产生一个2
       .if changedUp == 1
          invoke randomPlace2, dat, max
       .endif
       ; 更新界面
       invoke UpdateGame, hWnd
    .elseif edx == "S" || edx == VK_DOWN
```

```
54.       invoke movDOWN
55.       .if changedDown == 1
56.          invoke randomPlace2, dat, max
57.       .endif
58.       invoke UpdateGame, hWnd
59.    .elseif edx =="A" || edx == VK_LEFT
60.       invoke movLEFT
61.       .if changedLeft == 1
62.          invoke randomPlace2, dat, max
63.       .endif
64.       invoke UpdateGame, hWnd
65.    .elseif edx == "D" || edx == VK_RIGHT
66.       invoke movRIGHT
67.       .if changedRight == 1
68.          invoke randomPlace2, dat, max
69.       .endif
70.       invoke UpdateGame, hWnd
71.    .endif
72.
73.    ; 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
74.    .if gameContinue == 0
75.             invoke checkWin
76.       ; 如果gameIsWin==1，游戏获胜，弹出游戏获胜消息
77.       .if gameIsWin == 1
78.          invoke gameWin
79.       .endif
80.    .endif
81.
82.    ; 移动完毕之后，判断游戏是否结束，如果游戏结束，绘制失败弹窗
83.    invoke gameEnd
84.    .if gameIsEnd == 1
85.       invoke MessageBox, hWinMain, offset szText7, offset szText6,
    MB_OK
86.       ; 重新开始游戏
87.       .if eax == IDOK
88.          invoke ReStartGame
89.          INVOKE UpdateGame, hWnd
90.       .endif
91.    .endif
92. .else
93.    invoke DefWindowProc, hWnd, uMsg, wParam, lParam
```

```
94.         ret
95.     .endif
96.
97. xor eax,eax
98. ret
99._ProcWinMain endp
```

该函数作用是处理窗口信息。得判断窗口信息类型，包括绘制信息， 关闭信息，创建信息以及按键信息。绘制信息包括自定义绘制客户区，窗口显示及相关信息。关闭信息，即窗口关闭功能。创建信息，则是创建窗口的功能的实现，其具体代码实现在 DrawGame中。键盘反馈信息，判断键盘内容↑ ↓ ← → WASD，根据不同的值采取不同的措施。

5.6. checkWin函数

```
1. checkWin proc far C;检查游戏是否胜利，游戏胜利则修改gameIsWin=1
2.     push esi
3.
4.     mov gameIsWin, 0;置零
5.     xor esi, esi;清零
6.     .while esi < 16;遍历16格
7.
8.     .if gameMat[esi * 4] == 2048;有2048，游戏胜利
9.     mov gameIsWin, 1
10.    .break
11.    .endif
12.
13.    inc esi
14.    .endw
15.
16.    pop esi
17.    ret
18.checkWin endp
```

此函数的功能是检查游戏是否胜利。实现逻辑为遍历16个格子，如果存在2048则游戏胜利，将变量gameIsWin设置为1。

5.7. gameEnd函数

```
gameEnd proc far C;检查游戏是否失败，若无路可走则修改gameIsEnd=1
    push esi
    push ecx
    push edx
    push eax

    xor esi, esi
    mov ecx, 16
check0:
    cmp gameMat[esi*4], 0
    je endL;存在空格，游戏继续
    inc esi
    loop check0

    xor esi, esi
    mov row, 0
checkrow:
    mov eax, row
    imul eax, 4
    mov esi, eax;esi = 4*row

    mov edx, gameMat[esi*4]
    mov ecx, 3;一行4个，比3次即可
    Lrow:
        inc esi
        cmp edx, gameMat[esi*4];相邻两数是否相同
        je endL;相同游戏未结束
        mov edx, gameMat[esi*4]
        loop Lrow

    inc row
    cmp row, 4
    jb checkrow

    xor esi, esi
    mov col, 0
checkcol:
    mov esi, col

```

```
40.     mov edx, gameMat[esi*4]
41.     mov ecx, 3;一行4个，比3次即可
42.     Lcol:
43.         add esi, 4
44.         cmp edx, gameMat[esi*4];相邻两数是否相同
45.         je endL;相同游戏未结束
46.         mov edx, gameMat[esi*4]
47.         loop Lcol
48.
49.     inc col
50.     cmp col, 4
51.     jb checkcol
52.
53.     mov gameIsEnd, 1;无路可走，游戏结束
54. endL:
55.     pop eax
56.     pop edx
57.     pop ecx
58.     pop esi
59.     ret
60. gameEnd Endp
```

此函数功能是检查游戏是否失败。实现主要分成两部分，第一步遍历16格，如果发现格子为空，即数组内容为0，则还有路可走，游戏继续；第二步遍历每一行和每一列，如果发现任意相邻的两格有相同的数，则还有路可走，游戏继续。当上述两步都不满足，则无路可走，游戏结束，将变量gameIsEnd设置为1。

5.8. num2byte函数

```
1. num2byte proc far C num:dword;将数字转为字符存储到数组Data中
2.      push eax
3.      push ecx
4.      push edx
5.      push ebx
6.      push edi
7.
8.      mov eax, num
9.      mov ecx, 10;被除数
10.
11.     xor edx, edx
```

```
12.     xor ebx, ebx
13.     .while eax > 0
14.         inc ebx
15.         idiv ecx
16.         add edx, 30H;余数转化为字符
17.         push edx;低位先入栈
18.         xor edx, edx
19.     .endw
20.
21.     mov edi, 0
22.     .while ebx > 0
23.         dec ebx
24.         pop eax
25.         mov byte ptr Data[edi], al;低8位
26.         inc edi
27.     .endw
28.
29.     mov Data[edi], 0
30.
31.     pop edi
32.     pop ebx
33.     pop edx
34.     pop ecx
35.     pop eax
36.     ret
37. num2byte endp
```

此函数功能是将数字类型数据转化成字符类型数据存到数组中，此类函数在大数相乘实验中就使用过。获得字符的方法是写一个循环将待转换数不断除10，然后从寄存器edx中获取余数，将其存入栈中。然后再写一个循环，利用栈先进后出的特性将每一位按续存入数组中。

5.9. movUP函数

```
1. movUP        proc far C uses eax ebx ecx edx
2.
3.           ; 初始化changedUP，row，col
4.           mov    changedUP, 0
5.           mov    row, 1
6.           mov    col, 1
```

```
7.
8.          ; 从左到右遍历，从上到下遍历
9. columnLoop:      cmp    col, 4              ; 遍历列
10.         ja     endLoop                ; col > 4，结束外循环
11.
12.         mov    row, 1              ; 初始化row
13.rowLoop:     cmp    row, 4              ; 遍历当前列的行
14.         jbe     L1                 ; row <= 4
15.         inc    col                 ; row > 4，跳出内循环
16.         jmp    columnLoop
17.
18.L1:          cmp    row, 1              ; 第1行无法向上移动，循环
    continue
19.         jne    notFirst
20.         inc    row
21.         jmp    rowLoop
22.
23.         ; 计算当前位置的格子索引eax，当前位置的上方格子索引ebx，索引
    从0开始，row和col从1开始
24.notFirst:    mov    eax, row
25.         shl    eax, 2
26.         add    eax, col
27.         sub    eax, 5              ; eax = 4 * row + col - 5，row
    行col列的当前格索引
28.         mov    edx, gameMat[eax * 4] ; 当前格的数值保存到edx
29.         lea    ebx, [eax - 4]         ; 当前格的上方格子索引ebx
30.
31.         ; 如果上方格子为空，当前格子持续往上移动到边界
32.         mov    ecx, row
33.         sub    ecx, 1              ; 初始化循环计数器，ecx = row -
    1，确保不会移动越界
34.moveUpLoop:      cmp    gameMat[ebx * 4], 0
35.         jne    mergeCheck             ; 若上方格子有方块，判断当前
    格子(若有方块)是否能与之合并
36.         mov    changedUP, 1           ; 标记已经向上移动
37.         mov    gameMat[ebx * 4], edx
38.         mov    gameMat[eax * 4], 0
39.         mov    eax, ebx            ; 更新eax索引
40.         sub    ebx, 4              ; 更新ebx索引
41.         loop   moveUpLoop
42.
```

```
43. mergeCheck:     cmp    ecx, 0              ; ecx = 0，当前格子已经
    移动到边界处
44.           jz     skipMerge
45.           cmp    edx, 0              ; edx = 0，当前格子为空，而上一
    个格子有方块
46.           jz     skipMerge
47.           cmp    edx, gameMat[ebx * 4] ; 当前格子的方块是否和上一个
    方块相同
48.           jne    skipMerge
49.
50.           ; 执行合并操作
51.           shl    edx, 1              ; edx = edx * 2
52.           add    score, edx
53.           mov    gameMat[ebx * 4], edx
54.           mov    gameMat[eax * 4], 0
55.           mov    changedUP, 1
56.
57. skipMerge:      inc    row
58.           jmp    rowLoop
59.
60. endLoop:     mov    eax, 1
61.           ret
62. movUP         endp
```

这段代码实现了方块向上移动时的操作，通过循环判断格子能否移动，在移动后能否合并，实现了整体方块的向上移动。其余三个move函数与该函数类似，不再做说明。

5.10. randomPlace2函数

```
1. randomPlace2 proc uses eax ebx ecx edx esi, lowerBound :dword, sz
    :dword;基于输入的种子与限定的随机数最大值产生32位随机数，随后初始化矩阵
2.
3.           ; 产生随机数的公式: randData = (randSeed * X + Y) mod Z,
    X和Y至少有一个是素数
4.           invoke GetTickCount          ; 获取系统时间，取得随机数种
    子randSeed
5.           mov    ecx, 23           ; X = 23
6.           mul    ecx               ; randSeed * 23
7.           add    eax, 7               ; randSeed * 23 + 7
8.           mov    ecx, sz           ; Z = sz
```

```
9.          xor     edx, edx            ; edx清零，准备做除法
10.         div     ecx                 ; (randseed * 23 + 7) mod Z （
    余数在edx）
11.         add     edx, lowerBound
12.         mov     randData, edx         ; 产生一个[ lowerBound,
    (lowerBound + sz) )区间内的随机数randData
13.
14.         ; 索引为randData的格子位置产生滑块2，无冲突时跳转
15.         cmp     gameMat[edx * 4], 0
16.         je      place2
17.
18.         ; 冲突处理
19.         xor    ebx, ebx             ; ebx清零，用于存放gameMat指针
20.         xor    esi, esi             ; esi清零，用于存放tmpMat指针
21.
22.         mov    ecx, 16                ; 执行16次
23.L1:          cmp    gameMat[ebx * 4], 0       ; 遍历每一个格子
24.         jnz    L2
25.         mov    tmpMat[esi * 4], ebx   ; 记录所有等于0的格子位置
26.         inc    esi
27.L2:          inc    ebx
28.         loop   L1
29.
30.         mov    eax, randData
31.         xor    edx, edx             ; edx清零，准备做除法
32.         div    esi                  ; esi存放着tmpMat的长度，(余数
    在edx)
33.         mov    eax, tmpMat[edx * 4]   ; edx记录着tmpMat的下标
34.         mov    edx, eax
35.
36.place2:         mov    gameMat[edx * 4], 2
37.
38.         ret
39.randomPlace2 endp
```

该函数的主要功能是基于输入的种子和限定的随机数最大值生成一个随机数，并将该随机数作为索引，在游戏矩阵中找到一个空的位置放置滑块2。如果找不到空的位置，则会进行冲突处理。

5.11. RefreshGameBoard函数

```
1. RefreshGameBoard   PROC C USES EDX, windowHandle
2.  ; 触发绘制消息以更新界面，特别是重新绘制数字块。
3.  INVOKE   SendMessage,windowHandle, WM_PAINT, 1, 0
4.  INVOKE   num2byte, score
5.  ; 在窗口中设置分数值
6.  INVOKE   SetWindowText, hGame[72], OFFSET Data
7.  RET
8. RefreshGameBoard   ENDP
```

该函数的主要功能是发送绘制消息以更新游戏界面，将分数值设置到窗口的指定位置，使用SendMessage函数发送一个绘制消息（WM_PAINT）到指定的窗口（由windowHandle参数指定），以更新界面。这个消息会触发窗口的重新绘制，特别是重新绘制数字块。调用了一个名为num2byte的函数，将分数（score）转换为字节形式。这可能是为了将分数值以字节形式传递给后续的操作。第6行使用SetWindowText函数，在窗口的指定位置（hGame[72]）设置分数值。

5.12. RenderGame

```
1. RenderGame PROC C USES EAX, windowHandle :DWORD
2.     ; 绘制分数框
3.     INVOKE num2byte, score
4.     INVOKE CreateWindowEx, WS_EX_RIGHT, OFFSET edit, OFFSET Data,
    \
5.            WS_CHILD OR WS_VISIBLE OR WS_DISABLED, \
6.            296, 95, 132, 28, windowHandle, 18, hInstance, NULL
7.     MOV hGame[72], EAX
8.     RET
9. RenderGame ENDP
```

该函数实现更新分数，并且显示在分数框上。该函数创建并渲染一个用于显示分数的窗口，并将窗口句柄存储到指定位置。

5.13. RestartGame函数

```
1. RestartGame PROC FAR C USES EAX ESI ECX EDX
2.     ; 调用 DrawScoreBoard
3.     MOV ECX, 16
4.     MOV ESI, 0
5.     ; 清空 gameMat
6.     .WHILE ECX > 0
7.         MOV gameMat[ESI * 4], 0
8.         INC ESI
9.         DEC ECX
10.    .ENDW
11.    ; 初始化
12.    MOV gameIsEnd, 0
13.    MOV gameIsWin, 0
14.    MOV gameContinue, 0
15.    MOV score, 0
16.    MOV state, 0
17.    ; 随机初始化 gameMat
18.    INVOKE random32, dat, max
19.    INVOKE random32, dat, max
20.    RET
21.RestartGame ENDP
```

该函数的主要功能是重新开始游戏，包括清空游戏矩阵、初始化游戏状态和分数，并随机初始化游戏矩阵的一些值。通过循环判断，将游戏内容清空并初始化。

5.14. UpdateBlock函数

```
1. UpdateBlock PROC USES EAX EBX ESI
2.     LOCAL @stPs :PAINTSTRUCT
3.     LOCAL @bmpId :DWORD
4.     LOCAL @hDc :DWORD, hDcBmp :DWORD, hBmp :DWORD, @hBmp :DWORD
5.     INVOKE BeginPaint, hWinMain, ADDR @stPs
6.     XOR ESI, ESI
7.     .WHILE ESI < 16
8.         MOV EAX, gameMat[ESI * 4]
9.         .IF EAX == 0
10.            MOV EBX, 110
11.        .ELSEIF EAX == 2
```

```
12.             MOV EBX, 111
13.         .ELSEIF EAX == 4
14.             MOV EBX, 112
15.         .ELSEIF EAX == 8
16.             MOV EBX, 113
17.         .ELSEIF EAX == 16
18.             MOV EBX, 114
19.         .ELSEIF EAX == 32
20.             MOV EBX, 115
21.         .ELSEIF EAX == 64
22.             MOV EBX, 116
23.         .ELSEIF EAX == 128
24.             MOV EBX, 117
25.         .ELSEIF EAX == 256
26.             MOV EBX, 118
27.         .ELSEIF EAX == 512
28.             MOV EBX, 119
29.         .ELSEIF EAX == 1024
30.             MOV EBX, 120
31.         .ELSEIF EAX == 2048
32.             MOV EBX, 121
33.         .ENDIF
34.         MOV @bmpId, EBX
35.         INVOKE GetDC, hWinMain
36.         MOV @hDc, EAX
37.         INVOKE CreateCompatibleDC, @hDc
38.         MOV hDcBmp, EAX
39.         INVOKE CreateCompatibleBitmap, @hDc, 96, 96
40.         MOV hBmp, EAX
41.         INVOKE SelectObject, hDcBmp, hBmp
42.         INVOKE LoadBitmap, hInstance, @bmpId
43.         MOV @hBmp, EAX
44.         INVOKE CreatePatternBrush, @hBmp
45.         PUSH EAX
46.         INVOKE SelectObject, hDcBmp, EAX
47.         INVOKE PatBlt, hDcBmp, 0, 0, 96, 96, PATCOPY
48.         POP EAX
49.         INVOKE DeleteObject, EAX
50.         INVOKE BitBlt, @hDc, posXMat[ESI * 4], posYMat[ESI * 4],
    96, 96, hDcBmp, 0, 0, SRCCOPY
51.         INVOKE DeleteObject, @hBmp
52.         INVOKE DeleteDC, @hDc
```

```
53.         INVOKE DeleteDC, hDcBmp
54.         INVOKE DeleteObject, hBmp
55.         INC ESI
56.     .ENDW
57.     INVOKE EndPaint, hWinMain, ADDR @stPs
58.     RET
59. UpdateBlock ENDP
```

该函数的主要功能是根据游戏矩阵的状态，更新并绘制游戏界面中每个方块的位图。它通过循环遍历游戏矩阵中的每个方块，根据方块的值选择相应的位图标识符，并将相应的位图绘制到游戏界面上的指定位置。

函数首先通过调用BeginPaint函数获取与窗口相关的设备上下文句柄和绘制信息。然后，使用一个循环来遍历游戏矩阵中的每个方块。

在循环中，首先根据方块的值确定位图的标识符，并将其存储在局部变量@bmpId中。然后，函数创建与窗口设备上下文兼容的设备上下文，并创建与之兼容的位图。接下来，函数将位图对象选入设备上下文，并加载相应的位图资源。然后，函数创建一个图案刷子，并将其选入设备上下文。

接下来，函数使用PatBlt函数在设备上下文中绘制矩形区域，以填充位图的内容。然后，函数删除图案刷子，并使用BitBlt函数将设备上下文中的图像复制到窗口设备上下文的指定位置。

最后，函数删除位图对象和设备上下文，并通过调用EndPaint函数结束绘制过程。

第六章 实验人员分工

范曾	总体架构、文档编写、Main函数、WinMain函数、ProcWinMain函数、gameWin函数
洪子翔	UI设计及美工、ReStartGame函数、RefreshGameBoard函数、RenderGame函数
张栋梁	代码整合、测试运行、num2byte函数、gameEnd函数、checkWin函数
何秉翰	UI设计及美工、movX函数、randomPlace2函数

表 1 人员分工表

第七章 实验总结

在这次实验中，我们小组选择了“2048”这款游戏，2048游戏逻辑简单明确，通过写出c语言，再转换成汇编后，即可完成了代码实现。我们着重了ui设计，通过简单大方的卡通设计，简单的规则说明，让玩家快速上手。这个游戏的难点在于windows接口，在处理好这部分逻辑后，整体汇编代码得以实现。

本次实验锻炼了我们利用汇编语言开发实际应用的能力，增强了我们对于汇编语言的理解和运用。通过此次实验，还能更好的了解计算机工作具体原理，更好地深入理解计算机系统。