

**微机原理与汇编语言课程设计**

学院名称： 计算机工程学院

专 业： 计算机科学与技术

班 级： 18计3z

学 号： 2018141524

姓 名： 刘展鹏

指导教师： 傅中君

2020年6月18日

目录

[一、 摘要 2](#_Toc437021299)

[二、 课程设计目的 2](#_Toc622899553)

[三、 设计任务及要求 2](#_Toc90008146)

[四、 程序总体结构及基本算法 2](#_Toc938422334)

[五、 概要设计 3](#_Toc944263970)

[1 各模块设计思想 3](#_Toc340392460)

[1.1 数据变量的设计 3](#_Toc79639612)

[1.2 实验程序的整体架构 3](#_Toc620646803)

[1.3 游戏地图显示结构 3](#_Toc882744542)

[1.4 输入程序设计 3](#_Toc1470483918)

[1.5 更新游戏地图程序设计 3](#_Toc1181400150)

[1.6 判断游戏输赢的结构设计 3](#_Toc158520888)

[2 程序各模块流程图 3](#_Toc2108851670)

[六、 程序详细设计 5](#_Toc1399907602)

[1.数据的定义 5](#_Toc416230282)

[2.输出菜单 6](#_Toc1228111295)

[3.输入数字选择 7](#_Toc1401203748)

[4.更新游戏地图 8](#_Toc725719634)

[5.判断游戏是否结束 9](#_Toc1610257325)

[七、 测试数据及bug分析 13](#_Toc1872244083)

[八、 收获与体会 14](#_Toc1875907137)

[九、 附录(源代码) 15](#_Toc1208946388)

1. 摘要

**摘要**：用汇编语言作为自己程序设计的主流语言是不现实的，当代码量达到一定程度后，代码的维护会变困难，甚至不可能实现。但我们仍要学习汇编语言，但学习汇编的目的不是用它去进行实实在在的程序设计，而是充分获得底层的编程体验和深刻理解机器运行程序的机理。另外，对于一个人来说，不能没有常识，尽管常识不能直接挣钱吃饭，但它影响谈吐，影响你的判断力和决断力，决定着你接受新事物和新知识的程度。相应的，汇编语言就是计算机语言里面的常识和基础。

1. 课程设计目的

1.了解汇编语言程序设计的过程；熟悉DEBUG的基本调试命令；掌握应用DEBUG调试汇编语言程序的基本方法，熟悉顺序、分支、循环、子程序等设计的方法。

2.熟练掌握子程序的参数传递的方式、子程序的嵌套及其设计方法和技巧

3.培养学生创新精神和独立运用所学基本知识和技能分析问题和解决问题的能力。

4.进行汇编语言程序设计应用的综合性训练；掌握基于DOS/BIOS功能调用的标准控制台输入/输出程序的设计方法；灵活应用调试手段，实现标准控制台输入/ 输出程序的调试。

1. 设计任务及要求

本程序的目的是设计一款井字棋游戏，即有一个井字格，一共有9个格子。当有任意一方下的棋子中有三个可以连起来就胜利。

1. 程序总体结构及基本算法

程序中大致的分成以下一个结构：

* 输入结构，根据输入的数字，来将对应方颜色的棋子放在相应的位置上；
* 输出游戏地图结构，即将游戏的面板输出；
* 判断游戏结构，即实时的判断游戏是否结束；
* 主程序架构，即简单的循环；

1. 概要设计

1 各模块设计思想

* 1. 数据变量设计

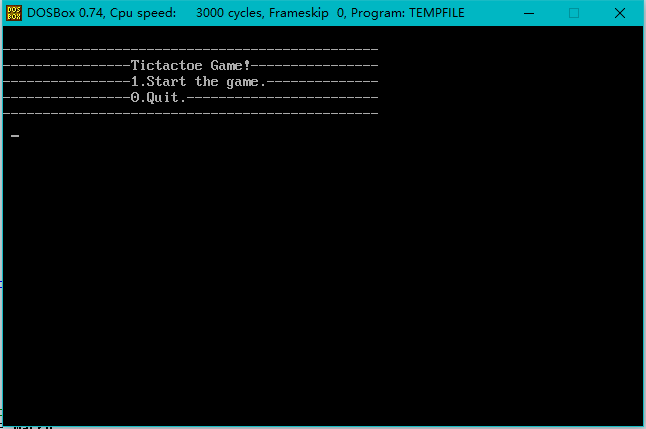
数据变量，本实验中一共设计了num变量，用来存储各个方格的状态：是被红色方占据，还是被蓝色方占据，或者是什么都没有，几乎本程序中该数据结构很重要，因为我们需要实时的维护该数据结构，根据该数据结构的值来显示对应的游戏进度；

还有的话，就是flag,用来存储当前轮到哪一方下棋。

1.2 实验程序的整体架构

主函数所做的事情就是一直循环输出功能列表，用户选择相应的功能之后开始执行相关的操作。

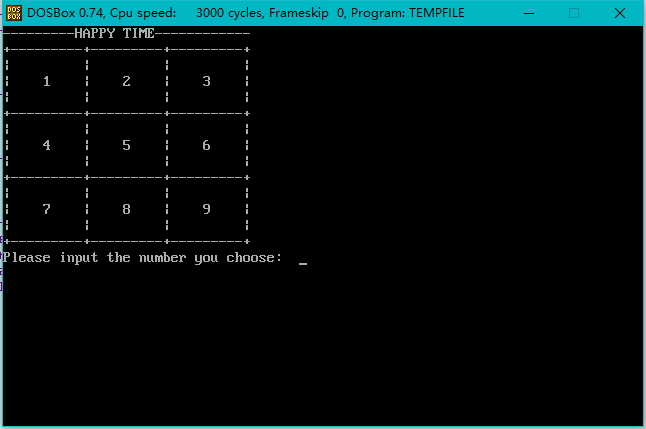
该部分很简单：就是首先屏幕会输出一个列表。上面是功能介绍，本次实验的具体功能有：



每一次会提示用户输入一个字符，接着会根据输入的具体的字符进行判断而去执行不同的子程序。

1.3 游戏地图显示结构

地图显示为下图所示：



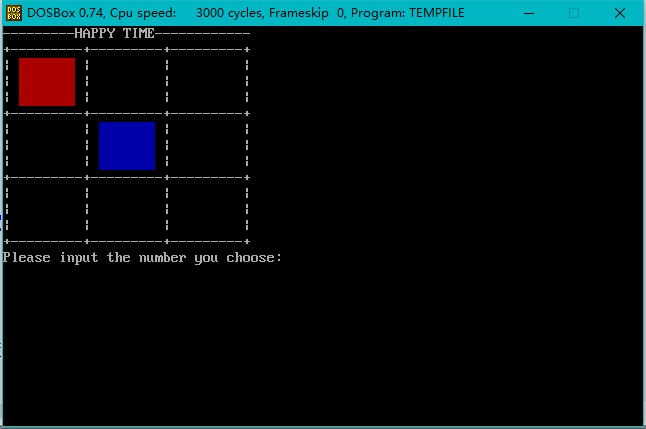
1.4 输入程序设计

本程序中输入数据之后，需要做的就是先取出对应方格标志位进行判断，如果该方格为空的话，就可以下棋，但是如果有人的话，就会提示输入失败，重新输入，然后维护相关的标志数组。

1.5 更新游戏地图程序设计

该部分根据我们提前在内存中维护的标志数组，进行显示；如果是红色方（一开始默认是红色方下棋）的话，就将对应的格子修改为红色，反之修改蓝色。

如下图所示：

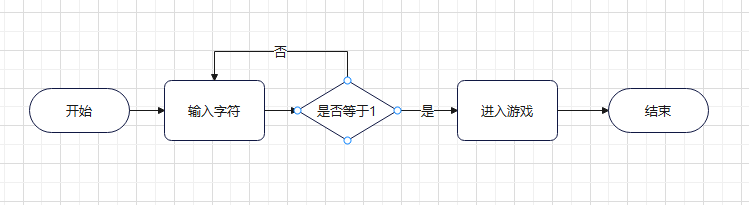


1.6 判断游戏输赢的结构设计

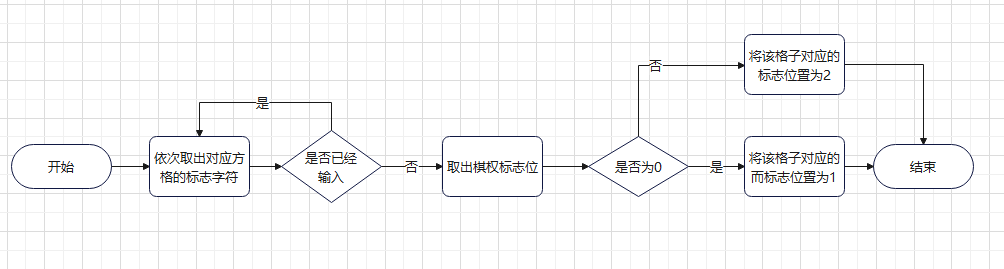
游戏的输赢判断就是判断某一个方向上是否有连续的三个颜色相同的格子，如果有三个相同的的格子的话（除了黑颜色外），就判定对应颜色方胜利；如果9个格子已经满了，说明游戏为平局。

2 程序各模块流程图

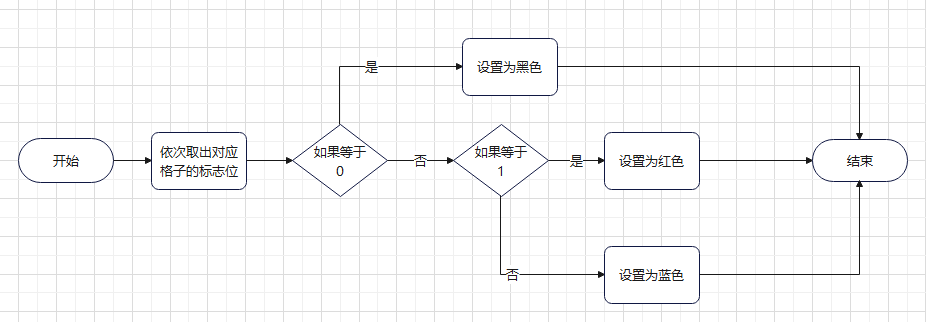
程序的整体架构：



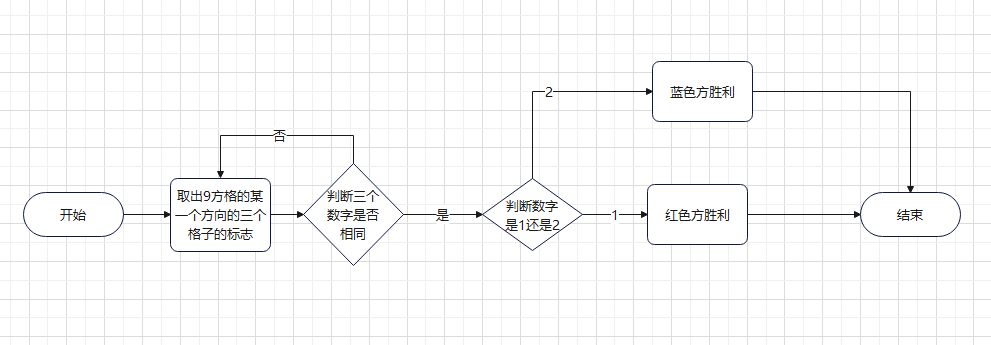
输入程序流程图：



显示游戏地图功能：



判断游戏是否结束：



1. 程序详细设计
2. 数据的定义

; The purpose of this program is to implement a simple tic-tac-toe!   
assume ds:data,cs:code,ss:stacks   
data segment   
 num db 9 dup(0)   
; The purpose of the parameter flag is to transfer the right of chess   
; if flag == 0 , it's red's turn to play chess   
; if flag == 1 , it's blue's turn to play chess   
 flag db 0   
 sum db 0   
 menu db "-----------------------------------------------",13,10   
 db "----------------Tictactoe Game!----------------",13,10   
 db "----------------1.Start the game.--------------",13,10   
 db "----------------0.Quit.------------------------",13,10   
 db "-----------------------------------------------",13,10,'$'   
 game db "---------HAPPY TIME------------",13,10   
 db "+---------+---------+---------+",13,10   
 db "| | | |",13,10   
 db "| 1 | 2 | 3 |",13,10   
 db "| | | |",13,10   
 db "+---------+---------+---------+",13,10   
 db "| | | |",13,10   
 db "| 4 | 5 | 6 |",13,10   
 db "| | | |",13,10   
 db "+---------+---------+---------+",13,10   
 db "| | | |",13,10   
 db "| 7 | 8 | 9 |",13,10   
 db "| | | |",13,10   
 db "+---------+---------+---------+",13,10   
 message db "Please input the number you choose: ",'$'   
 error db "Input Error,the location has no space!",'$'   
 red db "Congratulations to the red team!",'$'   
 blue db "Congratulations to the blue team!",'$'   
 \_no\_ db "Come out even!",'$'   
   
 CRLF db 13,10,'$'   
data ends

1. 输出菜单

; show game menu   
show\_map proc near   
 push dx   
 push ax   
 mov dx,offset game   
 mov ah,09h   
 int 21H   
   
 pop ax   
 pop dx   
 ret   
show\_map endp

1. 输入数字选择

; flag : num db 9 dup(0)   
; input characters and change the corresponding flag bit   
; if the red side is playing chess,the flag position is 1   
; if the blue side is playing chess, the flag position is 2   
; if there is nothing in the position, it will be 0   
; if the position is already occipied by someone else ,an error message will be displayed,and the input should be re-entered   
input\_number proc near   
 push ax   
 push dx   
input:   
 mov ah,01h   
 int 21h   
 block0   
 xor ah,ah   
 sub ax,48   
 cmp ax,0   
 je input\_error   
   
 mov di,offset num   
 sub ax,1   
 cmp ax,0   
 je find\_ed ; Find the flag corresponding to the square   
 mov cx,ax   
find\_ing:   
 inc di   
 loop find\_ing   
find\_ed:   
   
 ; check if the square has already been placed   
 mov al,ds:[di]   
 xor ah,ah   
 push di   
 cmp ax,0   
 je set\_ok   
 jmp input\_error   
 ; this square is not played   
set\_ok:   
 inc si   
 push di   
 push ax   
   
 mov di,offset sum   
 mov al,ds:[di]   
 inc al   
 mov ds:[di],al   
 pop ax   
 pop di   
 mov di,offset flag   
 mov al,ds:[di]   
 xor ah,ah   
 cmp ax,0   
 je input\_red   
 pop di   
 mov al,2   
 mov ds:[di],al   
 jmp input\_return   
   
input\_red:   
 pop di   
 mov al,1   
 mov ds:[di],al   
 jmp input\_return   
   
input\_error:   
 mov dx,offset error   
 mov ah,09h   
 int 21h   
 call delay   
 jmp input   
   
input\_return:   
 pop dx   
 pop ax   
 ret   
input\_number endp

1. 更新游戏地图

; update the first square   
set\_color\_1:   
 mov al,ds:[di]   
 xor ah,ah   
 cmp ax,0   
 je \_white\_1   
 cmp ax,1   
 je \_red\_1   
 set\_color 2,2,8,4,16 ; blue   
 jmp set\_color\_2

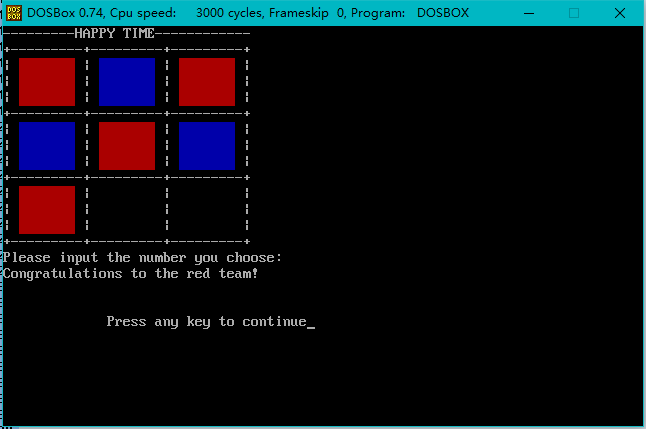
此处代码有些臃肿，所以仅仅拿出一个格子来说明，第一个格子，首先取出对应的标志位，如果是0的话，就正常显示；如果是1的话，显示红色；反之显示蓝色。

1. 判断游戏是否结束

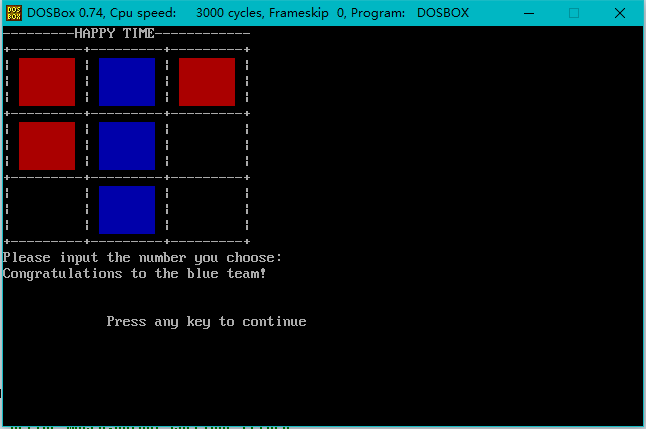
judge\_0:   
 ; take the first row of three numbers for comparison   
 mov di,offset num   
 mov al,ds:[di]   
 xor ah,ah   
 inc di   
 mov bl,ds:[di]   
 xor bh,bh   
 inc di   
 mov cl,ds:[di]   
 xor ch,ch   
 cmp ax,bx   
 jne judge\_1   
 cmp bx,cx   
 jne judge\_1

本子程序的代码臃肿度较高，仅仅拿出其中第一行的判断输赢来说明，即取出对应的三个数字，然后进行比较，如果三个数字全部相等的话，进行判断，如果是0，则进行其他比较，如果是1的话，说明是红色方胜利；如果是2的话，则蓝色方胜利。

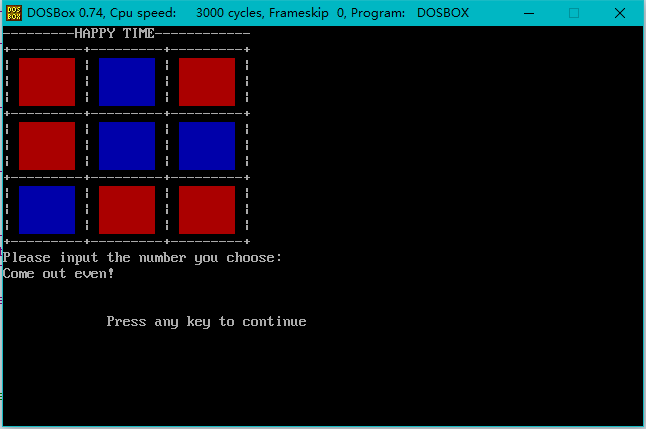
1. 测试数据及bug分析
2. 红色方胜利：



1. 蓝色方胜利：



3、平局：



1. 收获与体会

本大作业程序还有很多缺点如下：

1. 没有优化具体的结构，使得代码的臃肿度较高；
2. 整体的程序结构感觉设计的还是有所不足；

学习计算机知识，学习的是计算机整套的系统知识,当然这其中包括的内容就多了:数据结构,操作系统，微机原理，编译原理，各种高级语言，甚至更高层次的应用,如此众多的知识对于一个初次接触计算机的学习者来说深奥,茫然.这就需要一个好的教学体系与之对应，让学生能更好的理解计算机知识.而这套教学体系的开启项就应该是汇编语言原因是由汇编语言本身的特性决定的:它充分接触硬件，理解计算机运行原理，可以让操作者感知计算机的运行过程和原理，从而能够对计算机硬件和应用程序之间的联系和交互形成一个清晰的认识.这也是最能够锻炼编程者编程思维逻辑的,只有这样，学习者才能形成一个软.硬兼备的编程知识体系，这是任何高级语言都无法给予的!相对于繁复的高级语言，汇编语言指令集合更简约，指令操作更直接,从汇编开始学习更符合循序渐进的学习原理!最根本的是让学习者知道计算机到底是什么东西，抹去计算机学习者对计算机神秘,深奥的感觉，同时增加学习者的信心。

1. 附录(源代码)
2. ; The purpose of this program is to implement a simple tic-tac-toe!   
      
   assume ds:data,cs:code,ss:stacks   
      
   data segment   
    num db 9 dup(0)   
    ; The purpose of the parameter flag is to transfer the right of chess   
    ; if flag == 0 , it's red's turn to play chess   
    ; if flag == 1 , it's blue's turn to play chess   
    flag db 0   
    sum db 0   
    menu db "-----------------------------------------------",13,10   
    db "----------------Tictactoe Game!----------------",13,10   
    db "----------------1.Start the game.--------------",13,10   
    db "----------------0.Quit.------------------------",13,10   
    db "-----------------------------------------------",13,10,'$'   
    game db "---------HAPPY TIME------------",13,10   
    db "+---------+---------+---------+",13,10   
    db "| | | |",13,10   
    db "| 1 | 2 | 3 |",13,10   
    db "| | | |",13,10   
    db "+---------+---------+---------+",13,10   
    db "| | | |",13,10   
    db "| 4 | 5 | 6 |",13,10   
    db "| | | |",13,10   
    db "+---------+---------+---------+",13,10   
    db "| | | |",13,10   
    db "| 7 | 8 | 9 |",13,10   
    db "| | | |",13,10   
    db "+---------+---------+---------+",13,10   
    message db "Please input the number you choose: ",'$'   
    error db "Input Error,the location has no space!",'$'   
    red db "Congratulations to the red team!",'$'   
    blue db "Congratulations to the blue team!",'$'   
    \_no\_ db "Come out even!",'$'   
      
    CRLF db 13,10,'$'   
   data ends   
      
   ; define macro,output carriage return   
   \_output\_CRLF macro   
    mov dx,offset CRLF   
    mov ah,09h   
    int 21h   
   endm   
      
   ; define macro, output the game menu   
   show\_menu macro   
    mov dx,offset menu   
    mov ah,09h   
    int 21H   
   endm   
   ; define macro, switch the right of chess   
   change macro   
    push ax   
    mov di,offset flag   
    mov al,ds:[di]   
    xor ah,ah   
    cmp ax,0   
    je set\_blue   
    mov al,0   
    mov ds:[di],al   
    jmp change\_ret   
   set\_blue:   
    mov al,1   
    mov ds:[di],al   
   change\_ret:   
    pop ax   
   endm   
      
      
      
   ; define macro, place the cursor at the specified position   
   post\_c macro y,x   
    mov bh,0   
    mov ah,2   
    mov dh,y   
    mov dl,x   
    int 10h   
   endm   
      
   ; define macro, Clear the screen of the specified range   
   clear macro a,b,c,d   
    mov al,0   
    mov bh,7   
    mov ch,a   
    mov cl,b   
    mov dh,c   
    mov dl,d   
    mov ah,6   
    int 10h   
   endm   
      
   ; change the square corresponding to the coordinate to corresponding color   
   set\_color macro a,b,c,d,e   
    push ax   
    push bx   
    push cx   
    push dx   
    mov al,0   
    ; 16 is blue   
    ; 64 is red   
    mov bh,e   
    mov ch,b   
    mov cl,a   
    mov dh,d   
    mov dl,c   
    mov ah,6   
    int 10h   
    pop dx   
    pop cx   
    pop bx   
    pop ax   
   endm   
      
   ; define macro, blocking input program   
   block macro   
    push ax   
    push dx   
   block\_loop:   
    mov ah,0   
    int 16h   
    cmp ah,1ch   
    jne block\_loop   
    \_output\_CRLF   
    pop dx   
    pop ax   
   endm   
      
   block0 macro   
    push ax   
    push dx   
   block0\_loop:   
    mov ah,0   
    int 16h   
    cmp ah,1ch   
    jne block0\_loop   
    \_output\_CRLF   
    pop dx   
    pop ax   
   endm   
      
   stacks segment stack   
    dw 20 dup(?)   
   stacks ends   
      
   code segment   
   start:   
    mov ax,data   
    mov ds,ax   
    mov ax,stacks   
    mov ss,ax   
   main\_loop:   
    xor si,si   
    clear 0,0,24,79   
    post\_c 0,0   
    \_output\_CRLF   
    show\_menu   
      
    mov ah,1   
    int 21h   
    ; input   
    block   
    xor ah,ah   
    cmp ax,0030h   
    je main\_return   
    cmp ax,0031h   
    je \_game\_loop   
    jmp main\_loop   
      
    ; into the game   
   \_game\_loop:   
    ; clear the screen   
    clear 0,0,24,79   
    ; place the cursor at (0,0)   
    post\_c 0,0   
    ; display the game-menu   
    call show\_map   
   \_game\_ing:   
    call input\_number   
    call update\_map   
    change   
    call judge\_game   
    cmp bx,1   
    je \_red\_suc   
    cmp bx,2   
    je \_blue\_suc   
      
    mov di,offset sum   
    mov al,ds:[di]   
    xor ah,ah   
    cmp si,9   
    je \_no\_team   
    jmp \_game\_ing   
   \_red\_suc:   
    post\_c 15,0   
    mov dx,offset red   
    mov ah,09h   
    int 21h   
    jmp main\_return   
   \_no\_team:   
    post\_c 15,0   
    mov dx,offset \_no\_   
    mov ah,09h   
    int 21h   
    jmp main\_return   
   \_blue\_suc:   
    post\_c 15,0   
    mov dx,offset blue   
    mov ah,09h   
    int 21h   
    ; The coordinates corresponding to the 9 squares   
    ; 1: 2,2,8,4   
    ; 2: 12,2,18,4   
    ; 3: 22,2,28,4   
    ; 4: 2,6,8,8   
    ; 5: 12,6,18,8   
    ; 6: 22,6,28,8   
    ; 7: 2,10,8,12   
    ; 8: 12,10,18,12   
    ; 9: 22,10,28,12   
      
   main\_return:   
    mov ah,4CH   
    int 21H   
      
   ; show game menu   
   show\_map proc near   
    push dx   
    push ax   
    mov dx,offset game   
    mov ah,09h   
    int 21H   
      
    pop ax   
    pop dx   
    ret   
   show\_map endp   
      
   ; flag : num db 9 dup(0)   
   ; input characters and change the corresponding flag bit   
   ; if the red side is playing chess,the flag position is 1   
   ; if the blue side is playing chess, the flag position is 2   
   ; if there is nothing in the position, it will be 0   
   ; if the position is already occipied by someone else ,an error message will be displayed,and the input should be re-entered   
   input\_number proc near   
    push ax   
    push dx   
   input:   
    mov ah,01h   
    int 21h   
    block0   
    xor ah,ah   
    sub ax,48   
    cmp ax,0   
    je input\_error   
      
    mov di,offset num   
    sub ax,1   
    cmp ax,0   
    je find\_ed ; Find the flag corresponding to the square   
    mov cx,ax   
   find\_ing:   
    inc di   
    loop find\_ing   
   find\_ed:   
      
    ; check if the square has already been placed   
    mov al,ds:[di]   
    xor ah,ah   
    push di   
    cmp ax,0   
    je set\_ok   
    jmp input\_error   
    ; this square is not played   
   set\_ok:   
    inc si   
    push di   
    push ax   
      
    mov di,offset sum   
    mov al,ds:[di]   
    inc al   
    mov ds:[di],al   
    pop ax   
    pop di   
    mov di,offset flag   
    mov al,ds:[di]   
    xor ah,ah   
    cmp ax,0   
    je input\_red   
    pop di   
    mov al,2   
    mov ds:[di],al   
    jmp input\_return   
      
   input\_red:   
    pop di   
    mov al,1   
    mov ds:[di],al   
    jmp input\_return   
      
   input\_error:   
    mov dx,offset error   
    mov ah,09h   
    int 21h   
    call delay   
    jmp input   
      
   input\_return:   
    pop dx   
    pop ax   
    ret   
   input\_number endp   
      
   ; delay the program ,let the error message disappear after a while   
   delay proc near   
    push ax   
    push dx   
    mov dx,0012h   
    mov ax,0   
   s1:   
    sub ax,1   
    sbb dx,0   
    cmp ax,0   
    jne s1   
    cmp dx,0   
    jne s1   
    pop dx   
    pop ax   
    ret   
   delay endp   
      
   ; update map   
   update\_map proc near   
    push ax   
    push dx   
    push bx   
      
    clear 0,0,20,38   
    post\_c 0,0   
    mov dx,offset game   
    mov ah,09h   
    int 21H   
      
    mov di,offset num   
      
   ; update the first square   
   set\_color\_1:   
    mov al,ds:[di]   
    xor ah,ah   
    cmp ax,0   
    je \_white\_1   
    cmp ax,1   
    je \_red\_1   
    set\_color 2,2,8,4,16 ; blue   
    jmp set\_color\_2   
   \_white\_1:   
    set\_color 2,2,8,4,0 ; white   
    jmp set\_color\_2   
   \_red\_1:   
    set\_color 2,2,8,4,64 ; red   
      
    ; update the second square   
   set\_color\_2:   
    inc di   
    mov al,ds:[di]   
    xor ah,ah   
    cmp ax,0   
    je \_white\_2   
    cmp ax,1   
    je \_red\_2   
    set\_color 12,2,18,4,16 ; blue   
    jmp set\_color\_3   
   \_white\_2:   
    set\_color 12,2,18,4,0 ; white   
    jmp set\_color\_3   
   \_red\_2:   
    set\_color 12,2,18,4,64 ; red   
      
    ; update the third square   
   set\_color\_3:   
   inc di   
    mov al,ds:[di]   
    xor ah,ah   
    cmp ax,0   
    je \_white\_3   
    cmp ax,1   
    je \_red\_3   
    set\_color 22,2,28,4,16 ; blue   
    jmp set\_color\_4   
   \_white\_3:   
    set\_color 22,2,28,4,0 ; white   
    jmp set\_color\_4   
   \_red\_3:   
    set\_color 22,2,28,4,64 ; red   
      
    ; update the forth square   
   set\_color\_4:   
    inc di   
    mov al,ds:[di]   
    xor ah,ah   
    cmp ax,0   
    je \_white\_4   
    cmp ax,1   
    je \_red\_4   
    set\_color 2,6,8,8,16 ; blue   
    jmp set\_color\_5   
   \_white\_4:   
    set\_color 2,6,8,8,0 ; white   
      
    jmp set\_color\_5   
   \_red\_4:   
    set\_color 2,6,8,8,64 ; red   
      
    ; set the fifth square   
   set\_color\_5:   
    inc di   
    mov al,ds:[di]   
    xor ah,ah   
    cmp ax,0   
    je \_white\_5   
    cmp ax,1   
    je \_red\_5   
    set\_color 12,6,18,8,16 ; blue   
    jmp set\_color\_6   
   \_white\_5:   
    set\_color 12,6,18,8,0 ; white   
    jmp set\_color\_6   
   \_red\_5:   
    set\_color 12,6,18,8,64 ; red   
      
    ; set the sixth square   
   set\_color\_6:   
    inc di   
    mov al,ds:[di]   
    xor ah,ah   
    cmp ax,0   
    je \_white\_6   
    cmp ax,1   
    je \_red\_6   
    set\_color 22,6,28,8,16 ; blue   
    jmp set\_color\_7   
   \_white\_6:   
    set\_color 22,6,28,8,0 ; white   
    jmp set\_color\_7   
   \_red\_6:   
    set\_color 22,6,28,8,64 ; red   
      
    ; set the seventh square   
   set\_color\_7:   
    inc di   
    mov al,ds:[di]   
    xor ah,ah   
    cmp ax,0   
    je \_white\_7   
    cmp ax,1   
    je \_red\_7   
    set\_color 2,10,8,12,16 ; blue   
    jmp set\_color\_8   
   \_white\_7:   
    set\_color 2,10,8,12,0 ; white   
    jmp set\_color\_8   
   \_red\_7:   
    set\_color 2,10,8,12,64 ; red   
      
    ; set the eighth square   
   set\_color\_8:   
    inc di   
    mov al,ds:[di]   
    xor ah,ah   
    cmp ax,0   
    je \_white\_8   
    cmp ax,1   
    je \_red\_8   
    set\_color 12,10,18,12,16 ; blue   
    jmp set\_color\_9   
   \_white\_8:   
    set\_color 12,10,18,12,0 ; white   
    jmp set\_color\_9   
   \_red\_8:   
    set\_color 12,10,18,12,64 ; red   
      
    ; set the ninth square   
   set\_color\_9:   
    inc di   
    mov al,ds:[di]   
    xor ah,ah   
    cmp ax,0   
    je \_white\_9   
    cmp ax,1   
    je \_red\_9   
    set\_color 22,10,28,12,16 ; blue   
    jmp update\_return   
   \_white\_9:   
    set\_color 22,10,28,12,0 ; white   
    jmp update\_return   
   \_red\_9:   
    set\_color 22,10,28,12,64 ; red   
      
   update\_return:   
    pop bx   
    pop dx   
    pop ax   
      
    ret   
   update\_map endp   
      
   ; determine if the game is over   
   ; the return value is stored in register BX and SI   
   ; if si == 9, draw   
   ; if bx == 1, red side wins   
   ; if bx == 2, blue side wins   
   judge\_game proc near   
      
   judge\_0:   
    ; take the first row of three numbers for comparison   
    mov di,offset num   
    mov al,ds:[di]   
    xor ah,ah   
    inc di   
    mov bl,ds:[di]   
    xor bh,bh   
    inc di   
    mov cl,ds:[di]   
    xor ch,ch   
    cmp ax,bx   
    jne judge\_1   
    cmp bx,cx   
    jne judge\_1   
    ; the three numbers are the same, indicating that one side may wins   
    cmp ax,0   
    je judge\_1   
    cmp ax,1   
    je judge\_red   
    jmp judge\_blue   
   judge\_1:   
    ; take the second row of three numbers for comparison   
    inc di   
    mov al,ds:[di]   
    xor ah,ah   
    inc di   
    mov bl,ds:[di]   
    xor bh,bh   
    inc di   
    mov cl,ds:[di]   
    xor ch,ch   
    cmp ax,bx   
    jne judge\_2   
    cmp bx,cx   
    jne judge\_2   
    ; the three numbers are the same, indicating that one side may wins   
    cmp ax,0   
    je judge\_2   
    cmp ax,1   
    je judge\_red   
    jmp judge\_blue   
   judge\_2:   
    ; take the third row of three numbers for comparison   
    inc di   
    mov al,ds:[di]   
    xor ah,ah   
    inc di   
    mov bl,ds:[di]   
    xor bh,bh   
    inc di   
    mov cl,ds:[di]   
    xor ch,ch   
    cmp ax,bx   
    jne judge\_3   
    cmp bx,cx   
    jne judge\_3   
    ; the three numbers are the same, indicating that one side may wins   
    cmp ax,0   
    je judge\_3   
    cmp ax,1   
    je judge\_red   
    jmp judge\_blue   
   judge\_3:   
    ; get the three numbers in the first column   
    mov di,offset num   
    mov al,ds:[di]   
    xor ah,ah   
      
    add di,3   
    mov bl,ds:[di]   
    xor bh,bh   
      
    add di,3   
    mov cl,ds:[di]   
    xor ch,ch   
    cmp ax,bx   
    jne judge\_4   
    cmp bx,cx   
    jne judge\_4   
    ; the three numbers are the same, indicating that one side may wins   
    cmp ax,0   
    je judge\_4   
    cmp ax,1   
    je judge\_red   
    jmp judge\_blue   
   judge\_4:   
    ; get the three numbers in the second column   
    mov di,offset num   
    inc di   
    mov al,ds:[di]   
    xor ah,ah   
      
    add di,3   
    mov bl,ds:[di]   
    xor bh,bh   
      
    add di,3   
    mov cl,ds:[di]   
    xor ch,ch   
    cmp ax,bx   
    jne judge\_5   
    cmp bx,cx   
    jne judge\_5   
    ; the three numbers are the same, indicating that one side may wins   
    cmp ax,0   
    je judge\_5   
    cmp ax,1   
    je judge\_red   
    jmp judge\_blue   
      
   judge\_5:   
    ; get the three numbers in the third column   
    mov di,offset num   
    add di,2   
    mov al,ds:[di]   
    xor ah,ah   
      
    add di,3   
    mov bl,ds:[di]   
    xor bh,bh   
      
    add di,3   
    mov cl,ds:[di]   
    xor ch,ch   
    cmp ax,bx   
    jne judge\_6   
    cmp bx,cx   
    jne judge\_6   
    ; the three numbers are the same, indicating that one side may wins   
    cmp ax,0   
    je judge\_6   
    cmp ax,1   
    je judge\_red   
    jmp judge\_blue   
   judge\_6:   
    ; take out the three diagonal numbers   
    mov di,offset num   
    mov al,ds:[di]   
    xor ah,ah   
      
    add di,4   
    mov bl,ds:[di]   
    xor bh,bh   
      
    add di,4   
    mov cl,ds:[di]   
    xor ch,ch   
    cmp ax,bx   
    jne judge\_7   
    cmp bx,cx   
    jne judge\_7   
    ; the three numbers are the same, indicating that one side may wins   
    cmp ax,0   
    je judge\_7   
    cmp ax,1   
    je judge\_red   
    jmp judge\_blue   
      
   judge\_7:   
    ; take out the three diagonal numbers   
    mov di,offset num   
    add di,2   
    mov al,ds:[di]   
    xor ah,ah   
      
    add di,2   
    mov bl,ds:[di]   
    xor bh,bh   
      
    add di,2   
    mov cl,ds:[di]   
    xor ch,ch   
    cmp ax,bx   
    jne judge\_no   
    cmp bx,cx   
    jne judge\_no   
    ; the three numbers are the same, indicating that one side may wins   
    cmp ax,0   
    je judge\_no   
    cmp ax,1   
    je judge\_red   
    jmp judge\_blue   
      
   judge\_no:   
    xor bx,bx   
    mov bx,0   
    jmp judge\_game\_return   
   judge\_red:   
    xor bx,bx   
    mov bx,1   
    jmp judge\_game\_return   
   judge\_blue:   
    xor bx,bx   
    mov bx,2   
   judge\_game\_return:   
    ret   
   judge\_game endp   
      
   code ends   
   end start