**西安邮电大学**

**汇编语言程序设计竞赛报告**

|  |  |
| --- | --- |
| **专业班级** | 计科1604 |
| **学生姓名** | 喻润洋 马倩茹 郭欣 邢锐锋 张俊松 |
| **学号** | 05166115 01164047 05168178 02162028 04161127 |
| **时间** | **2018年12月2日** |

**一、整体设计结构框图**

map.asm绘制游戏界面

G\_Snake.asm完善功能

sMA.asm让小蛇响应方向后自动移动

sMove.asm让小蛇响应对应的键盘中断自动移动

**二、各子模块设计框图**

1. map.asm绘制游戏界面

绘制游戏初始边框

设置边框颜色

设置左右边框

设置上下边框

画蛇

设置蛇颜色

绘制小蛇图形

绘制游戏界面

2．sMove.asm 是让小蛇响应对应的键盘中断自动移动(在map.asm的基础上)

左下角显示方向（U,D,L,R）

控制小蛇移动的方向

接受键盘的输入

无键盘输入小蛇停止移动

重新绘制小蛇

3. sMA.asm 是让小蛇响应方向后自动移动

左下角显示方向（U,D,L,R）

控制小蛇移动的方向

接受键盘的输入

无键盘输入小蛇按照原方向自动移动

重新绘制小蛇

4.G\_Snake.asm 是最终程序

控制游戏结束

右下角显示得分

接受键盘输入绘制小蛇

显示小蛇运动方向

随机出现食物

**三、各个子模块完成情况**

程序分为四个步骤：map.asm,sMove.asm,sMA.asm以及G\_Snake.asm.

map.asm部分完成了绘制游戏界面的功能，包括画出小蛇，设定游戏界面的上下左右边界，确定小蛇和边界的颜色，以及确定游戏开始时小蛇的初始位置。

sMove.asm部分使小蛇能够响应对应的键盘中断，读入键盘输入的方向键，并且更新小蛇结点中点的位置，重新绘制小蛇，使小蛇在有键盘输入的情况下完成在游戏界面中的移动。

sMA.asm部分完成在读取响应方向后自动移动的功能，即读取一次键盘方向，在下一次键盘输入之前，小蛇都会按照已经读取的方向自动更新位置。

G\_Snake.asm是最终程序，在调用以上三个子程序的基础上，增加了计分功能，在界面下方显示分值以及读取的方向，游戏界面内随机出现食物，小蛇越界游戏结束，恢复9h键盘中断正常退出，显示Game over.

注：游戏运行中有可能会卡住不出现食物，这时候是程序通过获取cmos芯片中的秒数来计算得出的食物位置不合理，正在重新获取新的秒数计算新的食物位置；很快就会恢复。

**四、任务管理模块的实现**

当多任务操作系统使用某种任务调度策略允许两个或更多进程并发共享一个处理器时，事实上处理器在某一时刻只会给一件任务提供服务。因为任务调度机制保证不同任务之间的切换速度十分迅速，因此给人多个任务同时运行的错觉。多任务系统中有3个功能单位：任务、进程和线程。

系统里必须支持一个任务管理器，功能是为每个任务分配时间片，处理各个任务的共享，互锁，饿死 等问题，并且使用特定算法决定不同任务的的分配时间片长度，在任务切换时保留每个任务的特定信息。多任务的要用到什么结构根据要实现的功能而定.

增加事件驱动机制和消息排队技术, 使不同级中断模拟按照优先级排队,同级多中断源按事件触发次序排队, CPU 按中断优先级处理中断响应, 对于同级多中断源, 按 FIFO 方式依次处理, 并赋予每个排队的中断源被处理/ 时间片0, 保证 CPU 均能响应任何中断请求, 不丢失服务事件。事件触发标志寄存器用来记录每个中断是否发生, 1 标志位表示发生, CPU 尚未处理消息, 只有处理完毕, 才清除。任务调度程序把各中断模块看成子任务, 周期性( 定时器定时) 检查队列是否空, 不空, 用队列头指针( 任务指针) 取出消息块首址给任务程序进行处理。对于按 FIFO 结构排队的消息队列, 每处理完一个消息块, 立即处理下一个消息块, 使每个任务都能按执行频率要求得到及时处理。

首先, 设计一个队列 RAM 缓冲器, 每个元素占用 2 个字节, 存放消息块首址, 通常用DW 伪指令创建。这个队列元素按任务优先级排队, 值得注意的是, 对同级多中断源( 即公用个中断向量) 创建一个子队列, 具有 FIFO 属性。其次, 设计一个任务指针, 初始化指向队列首址, 再设计一个头指针和尾指针初始化均指向 FIFO子队列首址。任务指针和头指针用于取消息块首址, 接着, 设计一个事件标志寄存器, 初始化清零。再设计一个同级多中断源事件计数器, 初始化清零。每当发生一次中断, 事件计数器加 1, 尾指针存放待处理消息块首后加 2; CPU 处理完一个消息, 头指针加 2, 事件计数器减 1, 当减为 0, 头指针和尾指针重新初始化.

**五、（喻润洋部分）**

**六、程序运行结果截图**

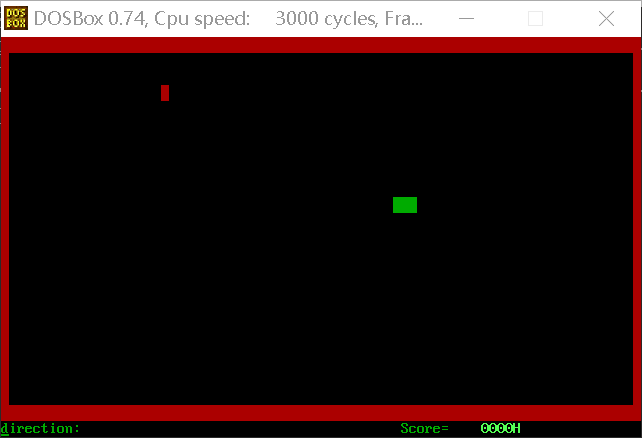


图1 游戏开始界面截图

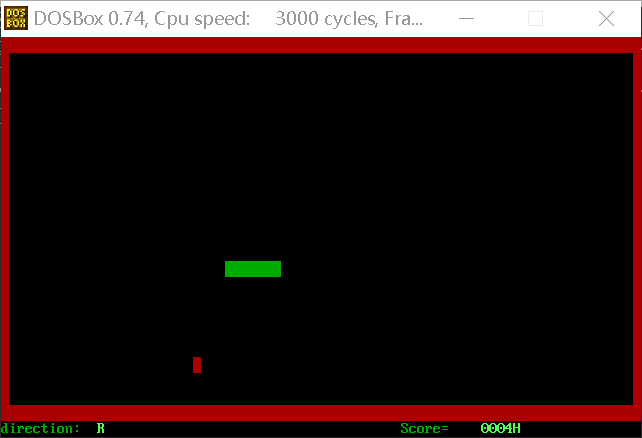


图2 游戏中界面截图

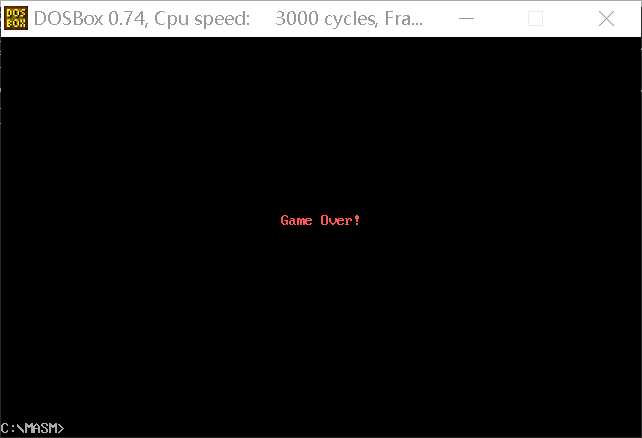


图3 游戏结束界面截图

**七、程序源代码（见附录）**

附录：

assume cs:code , ds:data , ss:stack

data segment

;db 256 dup(0) ; 代码一点一点写，每个调用都测试一下，现在要写真正程序用到的，就注释掉这里

BOUNDARY\_COLOR dw 4431h ; 直接定址法，颜色=0100100b,字符31h=数字1

NEXT\_ROL dw 0A0h ; a0=160

SNAKE\_HEAD dw 0

SNAKE\_BODY dw 6

SNAKE\_STERN dw 12

SNAKE dw 200 dup (0,0,0) ; 三个数是来记录前一个节点，中间点的位置，记录下一个点在内存中的相对偏移

;00 00 00 00 00 00

SNAKE\_COLOR dw 2201h ; 颜色00100010b,字符01h='☺'

UP db 48h

DOWN db 50h

LEFT db 4Bh

RIGHT db 4Dh

SCREEN\_COLOR dw 0700h

NEXT\_ROW dw 160

DIRECTION dw 3 ; 设置自动移动

DIRECTION\_FUN dw offset isMoveUp - offset greedy\_snake + 7e00h ; [0]

dw offset isMoveDown - offset greedy\_snake + 7e00h ; [2]

dw offset isMoveLeft - offset greedy\_snake + 7e00h ; [4]

dw offset isMoveRight - offset greedy\_snake + 7e00h ; [6]

FOOD\_LOCATION dw 160\*3 + 20\*2 ; 先人为的设定一个位置

FOOD\_COLOR dw 4439h ; 39h=字符'9' , 44h=01000100b 红色

NEW\_NODE dw 18 ; 后面吃到食物那要用到，18就是我们初始化小蛇后蛇尾的下一个位置

GAME\_OVER db 'Game Over!'

GAME\_DIR db 'direction: '

SCORE\_STR db 'Score='

SCORE\_CHAR db '0123456789ABCDEF'

SCORE dw 0h

SCORE\_POSITION dw 160\*24+60\*2

data ends

stack segment

db 128 dup(0)

stack ends

code segment

start:

mov ax , stack

mov ss , ax

mov sp , 128

call cpy\_greedy\_snake

call sav\_old\_int9

call set\_new\_int9

mov bx , 0

push bx

mov bx , 7e00h

push bx

retf ; pop ip , pop cs ; CS:IP=0:7e00

mov ax , 4c00h

int 21h

greedy\_snake:

call init\_reg

call clear\_screen

call init\_screen

call init\_food

call init\_snake ; 画出蛇的图形

nextMove:

call delay

cli

call isMoveDirection

sti

jmp nextMove

testA: ; 无限循环,测试

mov ax , 1000h

jmp testA

mov ax , 4c00h

int 21h

;-----------------------------------------

init\_food:

mov di , FOOD\_LOCATION

push FOOD\_COLOR

pop es:[di]

ret

;-----------------------------------------

isMoveDirection:

mov bx , DIRECTION

add bx , bx

call word ptr ds:DIRECTION\_FUN[bx]

ret

;-----------------------------------------

delay:

push ax

push dx

mov dx , 3h

sub ax , ax

delaying:

sub ax , 1

sbb dx , 0

cmp ax , 0

jne delaying

cmp dx , 0

jne delaying

pop dx

pop ax

ret

; 双向链表数据结构

;-----------------------------------------

init\_snake:

mov bx , offset SNAKE

add bx , SNAKE\_HEAD

mov si , 160\*10+40\*2 ; 屏幕上的位置(0690h)

mov dx , SNAKE\_COLOR ; 2201h

mov word ptr ds:[bx+0] , 0 ; ds:[bx+0] = 0

mov ds:[bx+2] , si

mov es:[si] , dx ; 设置蛇的颜色

mov word ptr ds:[bx+4] , 6

sub si , 2

add bx , 6

mov word ptr ds:[bx+0] , 0 ; ds:[bx+0] = 0

mov ds:[bx+2] , si

mov es:[si] , dx ; 设置蛇的颜色

mov word ptr ds:[bx+4] , 12

sub si , 2

add bx , 6

mov word ptr ds:[bx+0] , 6 ; ds:[bx+0] = 0

mov ds:[bx+2] , si

mov es:[si] , dx ; 设置蛇的颜色

mov word ptr ds:[bx+4] , 18

ret

;-----------------------------------------

init\_screen:

mov dx , BOUNDARY\_COLOR ; 设置游戏界面边框的颜色

call show\_up\_down\_line ; 画出游戏上下边界的边框

call show\_left\_right\_line ; 画出游戏左右边界的边框

call show\_score ; 初始化界面下面输出字符串score=

call output\_score ; 输出成绩，初始为 0

call show\_direction ; 显示dierction

ret

;-----------------------------------------

show\_direction:

mov si , offset GAME\_DIR

mov di , 160\*24+0\*2

mov cx , 11

showDirection:

mov al , ds:[si]

mov es:[di] , al

mov byte ptr es:[di+1] , 00000010b

inc si

add di , 2

loop showDirection

ret

;-----------------------------------------

show\_score:

mov si , offset SCORE\_STR

mov di , 160\*24+50\*2

mov cx , 6

showScore:

mov al , ds:[si]

mov es:[di] , al

mov byte ptr es:[di+1] , 00000010b

inc si

add di , 2

loop showScore

ret

;-----------------------------------------

show\_up\_down\_line:

mov bx , 0

mov cx , 80

showUpDownLine:

mov es:[bx] , dx ; dx=4431h,字符31h=数字1，颜色=01000100b

mov es:[bx+160\*23] , dx

add bx , 2

loop showUpDownLine

ret

;-----------------------------------------

output\_score:

mov si , offset SCORE

mov ax , ds:[si]

mov si , SCORE\_POSITION

mov dx , ax

mov al , ah

mov ah , 0

mov cx , 4

shr al , cl ; 得到最高位

mov bx , ax

mov al , ds:SCORE\_CHAR[bx]

mov byte ptr es:[si] , al

mov byte ptr es:[si+1] , 00001010b

add si , 2

mov ax , dx

mov al , ah

mov ah , 0

mov cx , 4

shl al , cl

shr al , cl ; 次高位

mov bx , ax

mov al , ds:SCORE\_CHAR[bx]

mov byte ptr es:[si] , al

mov byte ptr es:[si+1] , 00001010b

add si , 2

mov ax , dx

mov ah , 0

mov cx , 4

shr al , cl ; 次次高位

mov bx , ax

mov al , ds:SCORE\_CHAR[bx]

mov byte ptr es:[si] , al

mov byte ptr es:[si+1] , 00001010b

add si , 2

mov ax , dx

mov ah , 0

mov cx , 4

shl al , cl

shr al , cl ; 最低位

mov bx , ax

mov al , ds:SCORE\_CHAR[bx]

mov byte ptr es:[si] , al

mov byte ptr es:[si+1] , 00001010b

add si , 2

mov byte ptr es:[si] , 'H'

mov byte ptr es:[si+1] , 00001010b

ret

;-----------------------------------------

show\_left\_right\_line:

mov bx , 0

mov cx , 23

showLeftRightLine:

mov es:[bx] , dx

mov es:[bx+79\*2] , dx

;add bx , 160

add bx , NEXT\_ROW ; 优化写法

loop showLeftRightLine

ret

;-----------------------------------------

init\_reg:

mov bx , 0b800h

mov es , bx

mov bx , data

mov ds , bx

ret

;-----------------------------------------

clear\_screen:

mov bx , 0

mov dx , SCREEN\_COLOR

mov cx , 2000

clearScreen:

mov es:[bx] , dx ; (dx)=0700,es:[0]~es:[1]=00,es:[2]~es:[3]=07,颜色属性是07，RGB=白色

add bx , 2

loop clearScreen

ret

;-------------------------------------------

new\_int9:

push ax

call clear\_buff ; 清空键盘缓冲区

in al , 60h ; 60h号端口读取的是键盘扫描码

pushf

call dword ptr cs:[200h]

cmp al , UP ; 记录方向键扫描码

je isUp

cmp al , LEFT

je isLeft

cmp al , RIGHT

je isRight

cmp al , DOWN

je isDown

cmp al , 3bh ; 字符 ‘F1’ 的扫描码

jne int9Ret

call change\_screen\_color ;写来测试看，我写的9号中断处理键盘缓冲区能不能正常工作

int9Ret:

pop ax

iret

;-------------------------------------------

isUp:

mov di , 160\*24 + 12\*2 ; 每行最下面显示出按下的按键

mov byte ptr es:[di] , 'U'

mov byte ptr es:[di+1] , 00001010b

cmp DIRECTION , 1 ; 自动移动加入的时候，按下按键要进行判断，如果水平方向相反，就不用移动了

; 下面写的移动函数，在移动前会判断将要移动到的位置的字符信息是不是背景(也就是没显示其他的东西)，如果是

; 就直接返回了，这么写是代码风格更好。效率也高

je int9Ret

call isMoveUp

jmp int9Ret

isDown:

mov di , 160\*24 + 12\*2 ; 每行最下面显示出按下的按键

mov byte ptr es:[di] , 'D'

mov byte ptr es:[di+1] , 00001010b

cmp DIRECTION , 0

je int9Ret

call isMoveDown

jmp int9Ret

isLeft:

mov di , 160\*24 + 12\*2 ; 每行最下面显示出按下的按键

mov byte ptr es:[di] , 'L'

mov byte ptr es:[di+1] , 00001010b

cmp DIRECTION , 3

je int9Ret

call isMoveLeft

jmp int9Ret

isRight:

mov di , 160\*24 + 12\*2 ; 每行最下面显示出按下的按键

mov byte ptr es:[di] , 'R'

mov byte ptr es:[di+1] , 00001010b

cmp DIRECTION , 2

je int9Ret

call isMoveRight

jmp int9Ret

;-------------------------------------------

isMoveUp:

mov bx , offset SNAKE

add bx , SNAKE\_HEAD

mov si , ds:[bx+2] ; bx+2 获得节点中间的值，这个值记录的是该节点的位置

sub si , NEXT\_ROW ; 向上走了一步，所以中间节点的位置也要偏移160，刚好是一行

cmp byte ptr es:[si] , 0 ; 向上走一步如果es:[si]=1,表示到显示区的边界了，不能移动

jne noMoveUp

call draw\_new\_snake ; 移动，重新绘制蛇

mov DIRECTION , 0 ; 在按下相应按键，蛇能动了之后，就要设置自动移动了

ret

noMoveUp:

call isFood

ret

;-------------------------------------------

isMoveDown:

mov bx , offset SNAKE

add bx , SNAKE\_HEAD

mov si , ds:[bx+2]

add si , NEXT\_ROW

cmp byte ptr es:[si] , 0

jne noMoveDown

call draw\_new\_snake

mov DIRECTION , 1

ret

noMoveDown:

call isFood ;下一步不能走，加入食物后也要判断下是不是食物

ret

;-------------------------------------------

isMoveLeft:

mov bx , offset SNAKE

add bx , SNAKE\_HEAD

mov si , ds:[bx+2]

sub si , 2

cmp byte ptr es:[si] , 0

jne noMoveDown

call draw\_new\_snake

mov DIRECTION , 2

ret

noMoveLeft:

call isFood

ret

;-------------------------------------------

isMoveRight:

mov bx , offset SNAKE

add bx , SNAKE\_HEAD

mov si , ds:[bx+2]

add si , 2

cmp byte ptr es:[si] , 0

jne noMoveDown

call draw\_new\_snake

mov DIRECTION , 3

ret

noMoveRight:

call isFood

ret

;-------------------------------------------

isFood:

cmp byte ptr es:[si] , '9' ; 我们前面设置了食物的字符是9,前景背景都是红色

jne noFood

call eat\_food

call set\_new\_food ; 吃掉一个要再生成一个

ret

noFood:

call clear\_screen

call recover\_int9Ret

call end\_game

call return\_dos

ret

;-------------------------------------------

return\_dos:

mov ax , 4c00h

int 21h

;-------------------------------------------

recover\_int9Ret:

push es

mov bx , 0

mov es , bx

push es:[200h]

pop es:[9\*4]

push es:[202h]

pop es:[9\*4+2]

pop es

ret

;-------------------------------------------

end\_game:

mov si , offset GAME\_OVER

mov di , 160\*12+35\*2

mov cx , 10

endGame:

mov al , ds:[si]

mov byte ptr es:[di] , al

mov byte ptr es:[di+1] , 00001100b

inc si

add di , 2

loop endGame

ret

;-------------------------------------------

set\_new\_food:

mov al , 0

out 70h , al

in al , 71h

mov dl , al

and dl , 00001111b ; dl中是个位数的数字

push cx

mov cl , 4

mov ch , 0

shr al , cl ; al中是十位数的数字

pop cx ; 之前这里没有pop,导致出错，调试半天

mov bl , 10

mul bl ; ax=al\*bl

add al , dl ; 得到秒数

mul al ; 如果al是奇数,得到的肯定也是一个奇数;25\*80\*2=4000种显示位置,最后一行是3840~4000

; 按描述算，60\*60=3600种位置，所以食物不可能随机到最后一行，也不会到达游戏界面的下边界，导致蛇吃不到食物

shr al , 1 ; 二进制，右移一位去掉产生奇数的1

shl al , 1 ; 控制误差，再左移一位，这样误差就为1

mov bx , ax ; 得到下一个食物出现的位置

cmp byte ptr es:[bx] , 0 ; 如果得到的位置不是空闲的

jne set\_new\_food ; 这里有一个问题，如果生成的食物位置不行，到这里要跳转，又要进行下次执行，但是又要发生键盘中断

push FOOD\_COLOR

pop es:[bx]

ret

;-------------------------------------------

eat\_food:

push NEW\_NODE ; 记录新节点的位置，吃到食物，新食物变新的头结点

pop ds:[bx+0]

; 以前头结点的前驱指向食物，新食物的后继指向以前的头结点

mov bx , offset SNAKE

add bx , NEW\_NODE

mov word ptr ds:[bx+0] , 0 ;食物节点变成头结点，蛇变长一截

mov ds:[bx+2] , si

push SNAKE\_COLOR

pop es:[si]

push SNAKE\_HEAD

pop ds:[bx+4]

push NEW\_NODE

pop SNAKE\_HEAD ; 设置新的头

; 00 02 04 06 08 10 12 14 16 18 20 22

;最开始小蛇相关信息在内存中存放(18 si 06 | 00 si 12 | 06 si 18 | 00 si 00)

; ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑

; pre pos next pre pos next pre pos next pre pos next

;这个图是初始小蛇走了一步，然后吃了食物后在内存中的存储状态，第四个节点就变成新的蛇头节点,前面三个节点分别变成蛇的第二个节点，第三个节点和第四个节点

;头结点变了，以前的头结点也要做相应的修改

;尾节点没变，还是在内存偏移12的位置

add NEW\_NODE , 6

inc SCORE

call output\_score

ret

;-------------------------------------------

draw\_new\_snake:

push SNAKE\_STERN ; 蛇尾巴进栈保存，SNAKE\_STERN=12

pop ds:[bx+0] ; bx接上面是指向蛇头的,向上走一步，蛇头节点的第一个存储区就要修改，指向前一个节点

; 我们的做法就是我蛇尾的那个节点放到蛇头前面，这要中间的就可以不用修改，只修改第一个和最后一个

mov bx , offset SNAKE

add bx , SNAKE\_STERN ; 找到记录最后一个节点的内存(蛇尾)

push ds:[bx+0] ; 先保存最后一个节点中的pre,06

mov word ptr ds:[bx+0] , 0 ; 升级成蛇头了，节点中的pre变成0

mov di , ds:[bx+2] ; 中间节点保存的是位置信息，现在要改变颜色

push SCREEN\_COLOR

pop es:[di]

mov ds:[bx+2] , si ; 修改完颜色，现在最后蛇尾节点变蛇头，要重新修改位置

; si 上面是拿到了最开始蛇头的si-160

push SNAKE\_COLOR

pop es:[si] ; 在对应位置画出来

push SNAKE\_HEAD ; SNAKE\_HEAD=0

pop ds:[bx+4] ; 最后修改节点中最后一个信息，就是下一个节点的位置(这时候应该指向原来的头结点)

push SNAKE\_STERN ; 曾经的头部现在已经变成尾巴了，SNAKE\_STERN=12

pop SNAKE\_HEAD ; SNAKE\_HEAD=12

; 因为这些点是记录在内存的，所以要保存他们的位置，才能够正确访问到

; 现在SNAKE\_HEAD变成12，下次访问+SNAKE\_HEAD就找到真正的头结点在内存中的位置，

; 初始SNAKE\_HEAD=0，应该最开始我们在SNAKE中第一个节点存放头，所以只要偏移0就找到真正的头结点了

pop SNAKE\_STERN ; 先保存最后一个节点中的pre，现在修改了蛇尾放到头部前面，pre就是新的尾部

ret

; 00 02 04 06 08 10 12 14 16

;最开始小蛇相关信息在内存中存放(00 si' 06 | 00 si 12 | 06 si 18) ; 这里每个数字都是dw，我写成对应10进制表示的数了

; ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑

; pre pos next pre pos next pre pos next

;所以SNAKE\_HEAD=0,SNAKE\_STERN=12 , 所以要找到对应

;第一个头节点前面什么东西也没有，所有pre=0

;head body STERN ;最开始小蛇

;▅ ▅ ▅

;向上一步，原来的尾巴变成头，在修改对应信息就好了，比如原来最后尾巴节点颜色改成背景色

; 00 02 04 06 08 10 12 14 16

;向上一步后相关信息在内存中存放(12 si' 06 | 00 si 12 | 00 si'-1 00)

; ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑

; pre pos next pre pos next pre pos next(偏移00就可以找到第二个节点在内存中的位置，取出相关信息)

;所以SNAKE\_HEAD=12,SNAKE\_STERN=06 , 原来的头结点变成第二个节点，尾巴节点变成头，第二个节点变成尾巴

;STERN head body STERN ;向上一步走

;▅ ▅ ▅ ▅✘(修改颜色为背景色)

;-------------------------------------------

clear\_buff:

mov ah , 1

int 16h

jz clearBuffRet ; ZF=1,键盘无输入

mov ah , 0

int 16h

jmp clear\_buff

clearBuffRet:

ret

;-------------------------------------------

change\_screen\_color:

push bx

push cx

push es

mov bx , 0b800h

mov es , bx

mov bx , 1

mov cx , 2000

changeScreen:

inc byte ptr es:[bx]

add bx , 2

loop changeScreen

pop es

pop cx

pop bx

ret

greedy\_snake\_end:

nop

;-------------------------------------------

set\_new\_int9:

mov bx , 0

mov es , bx

cli

mov word ptr es:[9\*4] , offset new\_int9 - offset greedy\_snake + 7e00h

mov word ptr es:[9\*4+2] , 0

sti

ret

;-------------------------------------------

sav\_old\_int9: ; 保存原来的9h号处理键盘中断

mov bx , 0

mov es , bx

cli ; IF=0，不允许其它外中断

push es:[9\*4]

pop es:[200h]

push es:[9\*4+2]

pop es:[202h]

sti

ret

;-------------------------------------------

cpy\_greedy\_snake:

mov bx , cs

mov ds , bx

mov si , offset greedy\_snake

mov bx , 0

mov es , bx

mov di , 7e00h ;

mov cx , offset greedy\_snake\_end - offset greedy\_snake

cld

rep movsb

ret

code ends

end start