## university_of_smiu_logo .jpg

## PROJECT REPORT

|  |  |
| --- | --- |
| Group Members: | Khuzaima Tabraiz (CSC-18F-155) & Haris Hanif (CSC-18F-161) |
| Section: | CS – IV B |
| Course Name: | Computer Organization & Assembly Language |
| Course Code: | CEN202 |
| Course Supervisor: | Miss Sehar Jumani |

## SNAKE GAME

## Introduction to Assembly Language:

## This is a brief introduction to assembly language. Assembly language is the most basic programming language available for any processor. With assembly language, a programmer works only with operations implemented directly on the physical CPU. Assembly language lacks high-level conveniences such as variables and functions, and it is not portable between various families of processors. Nevertheless, assembly language is the most powerful computer programming language available, and it gives programmers the insight required to write effective code in high-level languages. Learning assembly language is well worth the time and effort of every serious programmer.

**Source Code:**

## left equ 0r

## top equ 2

## row equ 15

## col equ 40

## right equ left+col

## bottom equ top+row

## .model small

## .data

## msg db "Welcome to the snake game!!",0

## instructions db 0AH,0DH,"Use w, a, s and d to control your snake",0AH,0DH,"Use q anytime to quit",0DH,0AH, "Press any key to continue$"

## quitmsg db "Thanks for playing! hope you enjoyed",0

## gameovermsg db "OOPS!! your snake died! :(", 0

## scoremsg db "Your Score is: ",0

## head db '^',10,10

## body db '\*',10,11, 3\*15 DUP(0)

## segmentcount db 1

## fruitactive db 1

## fruitx db 8

## fruity db 8

## gameover db 0

## quit db 0

## delaytime db 5

## .stack

## dw 128 dup(0)

## .code

## main proc far

## mov ax, @data

## mov ds, ax

## 

## mov ax, 0b800H

## mov es, ax

## 

## ;clearing the screen

## mov ax, 0003H

## int 10H

## 

## lea bx, msg

## mov dx,00

## call writestringat

## 

## lea dx, instructions

## mov ah, 09H

## int 21h

## 

## mov ah, 07h

## int 21h

## mov ax, 0003H

## int 10H

## call printbox

## 

## 

## mainloop:

## call delay

## lea bx, msg

## mov dx, 00

## call writestringat

## call shiftsnake

## cmp gameover,1

## je gameover\_mainloop

## 

## call keyboardfunctions

## cmp quit, 1

## je quitpressed\_mainloop

## call fruitgeneration

## call draw

## 

## ;TODO: check gameover and quit

## 

## jmp mainloop

## 

## gameover\_mainloop:

## mov ax, 0003H

## int 10H

## mov delaytime, 100

## mov dx, 0000H

## lea bx, gameovermsg

## call writestringat

## call delay

## jmp quit\_mainloop

## 

## quitpressed\_mainloop:

## mov ax, 0003H

## int 10H

## mov delaytime, 100

## mov dx, 0000H

## lea bx, quitmsg

## call writestringat

## call delay

## jmp quit\_mainloop

## 

## 

## quit\_mainloop:

## ;first clear screen

## mov ax, 0003H

## int 10h

## mov ax, 4c00h

## int 21h

## 

## delay proc

## 

## 

## mov ah, 00

## int 1Ah

## mov bx, dx

## 

## jmp\_delay:

## int 1Ah

## sub dx, bx

## ;there are about 18 ticks in a second, 10 ticks are about enough

## cmp dl, delaytime

## jl jmp\_delay

## ret

## 

## delay endp

## 

## 

## fruitgeneration proc

## mov ch, fruity

## mov cl, fruitx

## regenerate:

## 

## cmp fruitactive, 1

## je ret\_fruitactive

## mov ah, 00

## int 1Ah

## ;dx contains the ticks

## push dx

## mov ax, dx

## xor dx, dx

## xor bh, bh

## mov bl, row

## dec bl

## div bx

## mov fruity, dl

## inc fruity

## 

## 

## pop ax

## mov bl, col

## dec dl

## xor bh, bh

## xor dx, dx

## div bx

## mov fruitx, dl

## inc fruitx

## 

## cmp fruitx, cl

## jne nevermind

## cmp fruity, ch

## jne nevermind

## jmp regenerate

## nevermind:

## mov al, fruitx

## ror al,1

## jc regenerate

## 

## 

## add fruity, top

## add fruitx, left

## 

## mov dh, fruity

## mov dl, fruitx

## call readcharat

## cmp bl, '\*'

## je regenerate

## cmp bl, '^'

## je regenerate

## cmp bl, '<'

## je regenerate

## cmp bl, '>'

## je regenerate

## cmp bl, 'v'

## je regenerate

## 

## ret\_fruitactive:

## ret

## fruitgeneration endp

## dispdigit proc

## add dl, '0'

## mov ah, 02H

## int 21H

## ret

## dispdigit endp

## 

## dispnum proc

## test ax,ax

## jz retz

## xor dx, dx

## ;ax contains the number to be displayed

## ;bx must contain 10

## mov bx,10

## div bx

## ;dispnum ax first.

## push dx

## call dispnum

## pop dx

## call dispdigit

## ret

## retz:

## mov ah, 02

## ret

## dispnum endp

## ;sets the cursor position, ax and bx used, dh=row, dl = column

## ;preserves other registers

## setcursorpos proc

## mov ah, 02H

## push bx

## mov bh,0

## int 10h

## pop bx

## ret

## setcursorpos endp

## draw proc

## lea bx, scoremsg

## mov dx, 0109

## call writestringat

## 

## 

## add dx, 7

## call setcursorpos

## mov al, segmentcount

## dec al

## xor ah, ah

## call dispnum

## 

## lea si, head

## draw\_loop:

## mov bl, ds:[si]

## test bl, bl

## jz out\_draw

## mov dx, ds:[si+1]

## call writecharat

## add si,3

## jmp draw\_loop

## out\_draw:

## mov bl, 'F'

## mov dh, fruity

## mov dl, fruitx

## call writecharat

## mov fruitactive, 1

## 

## ret

## 

## 

## 

## draw endp

## ;dl contains the ascii character if keypressed, else dl contains 0

## ;uses dx and ax, preserves other registers

## readchar proc

## mov ah, 01H

## int 16H

## jnz keybdpressed

## xor dl, dl

## ret

## keybdpressed:

## ;extract the keystroke from the buffer

## mov ah, 00H

## int 16H

## mov dl,al

## ret

## readchar endp

## 

## 

## 

## 

## 

## keyboardfunctions proc

## 

## call readchar

## cmp dl, 0

## je next\_14

## 

## ;so a key was pressed, which key was pressed then solti?

## cmp dl, 'w'

## jne next\_11

## cmp head, 'v'

## je next\_14

## mov head, '^'

## ret

## next\_11:

## cmp dl, 's'

## jne next\_12

## cmp head, '^'

## je next\_14

## mov head, 'v'

## ret

## next\_12:

## cmp dl, 'a'

## jne next\_13

## cmp head, '>'

## je next\_14

## mov head, '<'

## ret

## next\_13:

## cmp dl, 'd'

## jne next\_14

## cmp head, '<'

## je next\_14

## mov head,'>'

## next\_14:

## cmp dl, 'q'

## je quit\_keyboardfunctions

## ret

## quit\_keyboardfunctions:

## ;conditions for quitting in here please

## inc quit

## ret

## 

## keyboardfunctions endp

## 

## 

## 

## 

## 

## 

## shiftsnake proc

## mov bx, offset head

## 

## ;determine the where should the head go solti?

## ;preserve the head

## xor ax, ax

## mov al, [bx]

## push ax

## inc bx

## mov ax, [bx]

## inc bx

## inc bx

## xor cx, cx

## l:

## mov si, [bx]

## test si, [bx]

## jz outside

## inc cx

## inc bx

## mov dx,[bx]

## mov [bx], ax

## mov ax,dx

## inc bx

## inc bx

## jmp l

## 

## outside:

## 

## 

## 

## pop ax

## 

## push dx

## 

## 

## lea bx, head

## inc bx

## mov dx, [bx]

## 

## cmp al, '<'

## jne next\_1

## dec dl

## dec dl

## jmp done\_checking\_the\_head

## next\_1:

## cmp al, '>'

## jne next\_2

## inc dl

## inc dl

## jmp done\_checking\_the\_head

## 

## next\_2:

## cmp al, '^'

## jne next\_3

## dec dh

## 

## 

## jmp done\_checking\_the\_head

## 

## next\_3:

## ;must be 'v'

## inc dh

## 

## done\_checking\_the\_head:

## mov [bx],dx

## ;dx contains the new position of the head, now check whats in that position

## call readcharat ;dx

## ;bl contains the result

## 

## cmp bl, 'F'

## je i\_ate\_fruit

## 

## ;if fruit was not eaten, then clear the last segment,

## ;it will be cleared where?

## mov cx, dx

## pop dx

## cmp bl, '\*' ;the snake bit itself, gameover

## je game\_over

## mov bl, 0

## call writecharat

## mov dx, cx

## 

## 

## 

## 

## 

## ;check whether the snake is within the boundary

## cmp dh, top

## je game\_over

## cmp dh, bottom

## je game\_over

## cmp dl,left

## je game\_over

## cmp dl, right

## je game\_over

## 

## 

## 

## ;balance the stack, number of segment and the coordinate of the last segment

## 

## ret

## game\_over:

## inc gameover

## ret

## i\_ate\_fruit:

## ; add a new segment then

## mov al, segmentcount

## xor ah, ah

## 

## 

## lea bx, body

## mov cx, 3

## mul cx

## 

## pop dx

## add bx, ax

## mov byte ptr ds:[bx], '\*'

## mov [bx+1], dx

## inc segmentcount

## mov dh, fruity

## mov dl, fruitx

## mov bl, 0

## call writecharat

## mov fruitactive, 0

## ret

## shiftsnake endp

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## ;Printbox

## printbox proc

## ;Draw a box around

## mov dh, top

## mov dl, left

## mov cx, col

## mov bl, '\*'

## l1:

## call writecharat

## inc dl

## loop l1

## 

## mov cx, row

## l2:

## call writecharat

## inc dh

## loop l2

## 

## mov cx, col

## l3:

## call writecharat

## dec dl

## loop l3

## mov cx, row

## l4:

## call writecharat

## dec dh

## loop l4

## 

## ret

## printbox endp

## 

## 

## 

## 

## 

## 

## 

## 

## 

## ;dx contains row, col

## ;bl contains the character to write

## ;uses di.

## writecharat proc

## ;80x25

## push dx

## mov ax, dx

## and ax, 0FF00H

## shr ax,1

## shr ax,1

## shr ax,1

## shr ax,1

## shr ax,1

## shr ax,1

## shr ax,1

## shr ax,1

## 

## 

## push bx

## mov bh, 160

## mul bh

## pop bx

## and dx, 0FFH

## shl dx,1

## add ax, dx

## mov di, ax

## mov es:[di], bl

## pop dx

## ret

## writecharat endp

## 

## 

## 

## 

## 

## 

## 

## 

## 

## ;dx contains row,col

## ;returns the character at bl

## ;uses di

## readcharat proc

## push dx

## mov ax, dx

## and ax, 0FF00H

## shr ax,1

## shr ax,1

## shr ax,1

## shr ax,1

## shr ax,1

## shr ax,1

## shr ax,1

## shr ax,1

## push bx

## mov bh, 160

## mul bh

## pop bx

## and dx, 0FFH

## shl dx,1

## add ax, dx

## mov di, ax

## mov bl,es:[di]

## pop dx

## ret

## readcharat endp

## ;dx contains row, col

## ;bx contains the offset of the string

## writestringat proc

## push dx

## mov ax, dx

## and ax, 0FF00H

## shr ax,1

## shr ax,1

## shr ax,1

## shr ax,1

## shr ax,1

## shr ax,1

## shr ax,1

## shr ax,1

## 

## push bx

## mov bh, 160

## mul bh

## 

## pop bx

## and dx, 0FFH

## shl dx,1

## add ax, dx

## mov di, ax

## loop\_writestringat:

## 

## mov al, [bx]

## test al, al

## jz exit\_writestringat

## mov es:[di], al

## inc di

## inc di

## inc bx

## jmp loop\_writestringat

## 

## 

## exit\_writestringat:

## pop dx

## ret

## 

## 

## writestringat endp

## 

## 

## 

## 

## 

## 

## main endp

## 

## end main

## FLOW CHART :

## project-on-snake-game-in-c-language-7-638.jpg

## CONCLUSION :

## This is really an interesting game. Anyone can spend their time by playing this game. We also gather a huge knowledge about Assembly Language by completing this project. We also hope that in future we can make more project in Assembly Language.

## THANK YOU