**PROJECT – SNAKE GAME**

**Muhammad Tayyab Khan - 22F-3440**

**Azam Rafique - 22F-8766**

3-E **Dated:** 12/5/2023 **Submitted To:** Sir Ali Hamza

**Game Development Documentation: Assembly Language Game**

**1. Introduction**

The purpose of this game development project is to leverage the knowledge gained in Computer Organization and Assembly Language (COAL) to create a meaningful and functional game. This document outlines the general structure and requirements applicable to any game developed in assembly language.

The game has been developed using the following softwares/tools:

* DOSBOX
* NASM

**2. Objectives**

The primary objectives of the Snake Game project are:

* Demonstrate understanding of Instruction Set Architecture.
* Implement memory addressing, branching, bit manipulation, stack operations, subroutines, string operations, and interrupts in assembly language.
* Develop a playable Snake Game adhering to specified requirements.

**3. Game Choice**

Game chosen for our project is: **Snake Game**

**4. Documentation Elements**

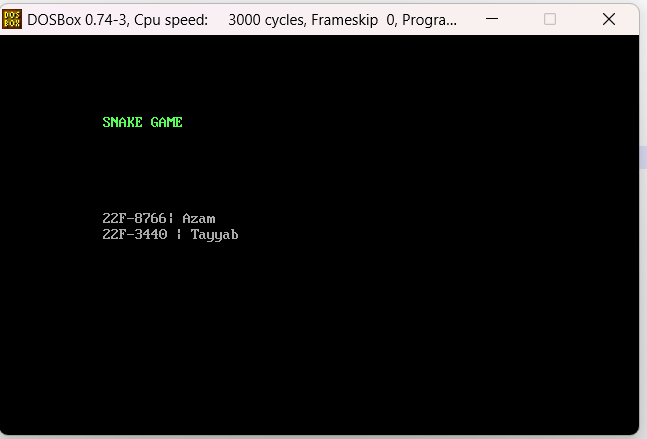
**Introduction Message:**

The chosen game type for this project is the classic Snake Game. This game involves controlling a snake to collect food and grow in length while avoiding collisions with the boundaries of the game area.

* The game begins with **Welcome Message, Instructions,** and **Actual Gameplay.**
* The Welcome Message displays the names of the project members and in our case it is (22F-3440 Tayyab, 22F8766 Azam)
* The Instructions are displayed after this screen.
* The gameplay begins after the instruction’s Game.

**Game Menu:**

Instructions: User will use the arrow keys (UP, DOWN, LEFT, RIGHT) to move the snake. User will hit enter on the instructions screen to move to the gameplay screen.

A screenshot of a computer

Description automatically generated

**Game Screen:**

The core game mechanics in the provided assembly code for the Snake Game involve several key elements, including player input, collision detection, and other relevant aspects. Let's break down these components:

**1. Player Input**

Player input is handled in the **play\_game** subroutine. The game waits for user input and responds to arrow key presses, which control the snake's movement. The relevant portion of the code is as follows:

**repeat:**

**mov ah, 01h ; Check for keyboard input**

**int 16h**

**jz noKey ; If no key is pressed, jump to 'noKey' label**

**mov ah, 0 ; Clear the keyboard buffer**

**int 16h**

**cmp ah, 0x48 ; Up arrow key**

**je up**

**cmp ah, 0x4B ; Left arrow key**

**je left**

**cmp ah, 0x4D ; Right arrow key**

**je right**

**cmp ah, 0x50 ; Down arrow key**

**je down**

**cmp ah, ' '**

**jne repeat ; Repeat loop if the key pressed is not space (not used for movement)**

**mov ah, 0x4c**

**int 21h**

**je exit2 ; Jump to 'exit2' label if the key is 'Enter'**

**noKey:**

**cmp byte[direction], 0**

**je up**

**cmp byte[direction], 1**

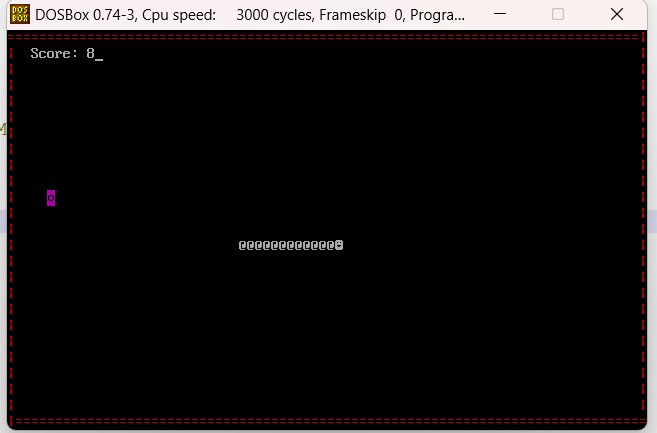
**je down**

**cmp byte[direction], 2**

**je left**

**cmp byte[direction], 3**

**je right**

****

**2. Collision Detection**

Collision detection is implemented in the **check\_death** subroutine. It checks whether the snake collides with the boundaries of the game area or with itself. If a collision is detected, the **over** subroutine is called to end the game.

**check\_death:**

**push ax**

**push di**

**push cx**

**mov ax, [snake\_locations]**

**cmp ax, 160**

**jb finished**

**; Check collision with the top and bottom borders**

**mov di, 160**

**mov cx, 24**

**check1:**

**cmp ax, di**

**je finished**

**add di, 158**

**cmp ax, di**

**je finished**

**add di, 2**

**loop check1**

**mov di, 3840**

**cmp ax, di**

**ja finished**

**jmp else**

**finished:**

**call over**

**else:**

**pop cx**

**pop di**

**pop ax**

**ret**

**3. Other Relevant Aspects**

**3.1 Snake Movement**

The snake's movement is determined by the **move\_snake\_up**, **move\_snake\_down**, **move\_snake\_left**, and **move\_snake\_right** subroutines. These subroutines handle updating the snake's position based on the current direction.

**3.2 Drawing the Snake**

The **draw\_snake** subroutine is responsible for rendering the snake on the screen. It uses video memory (**0xb800**) to display the snake's characters at the specified locations.

**3.3 Displaying Food**

The **displayFood** subroutine randomly places food on the screen, updating the food's position in video memory.

**3.4 Game Over Message**

When a collision is detected, the **over** subroutine is called to display the "Game Over" message along with the final score.

**Score/Progress Display**:

Updates and displays the player's score or relevant game progress during gameplay.

The display score mechanism in the provided assembly code involves updating and showing the player's score on the game screen. The relevant subroutine is **printScore.**

**printScore:**

**push bp**

**mov bp, sp**

**pusha**

**s:**

**mov ah, 0x13 ; BIOS interrupt service 13 - print string**

**mov al, 1 ; Subservice 01 – update cursor**

**mov bh, 0 ; Output on page 0**

**mov bl, 7 ; Normal attribute**

**mov dx, 0x0103 ; Row 10, Column 3**

**mov cx, 7 ; Length of the score string**

**push cs ; Push code segment to get the segment of the string**

**pop es ; Pop it into ES**

**mov bp, score ; Offset of the score string**

**int 0x10 ; Call BIOS interrupt to print the score string**

**mov dx, 0x010A ; Row 01, Column 10**

**mov cx, 1 ; Length of the currScore string**

**push cs ; Push code segment to get the segment of the string**

**pop es ; Pop it into ES**

**mov bp, currScore ; Offset of the currScore string**

**int 0x10 ; Call BIOS interrupt to print the currScore string**

**popa**

**pop bp**

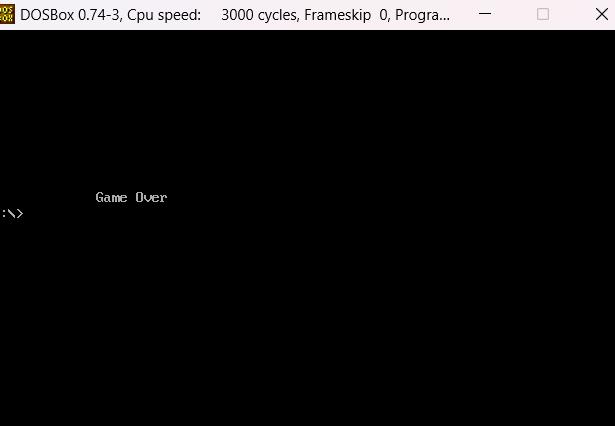
**ret 2 ; Return, removing 2 bytes from stack**

**A screenshot of a computer

Description automatically generated**

**Game Over/Win Message:**

Displays a message when the game ends or the player achieves victory, providing feedback to the player.



**5. Deadline**

The final submission deadline is non-extendable, set for December 5th. Partners are responsible for both online code submission and the in-person viva presentation on December 6th.

**CODE:**

[org 0x0100]

jmp start

x\_pos: dw 0

y\_pos: dw 0

foodposition: dw 0

gameName: db 'SNAKE GAME'

GroupMember1: db '22F-8766| Azam'

GroupMember2: db'22F-3440 | Tayyab'

GAMEINSTUCT: db 'Instructions: Use Arrows Key to Move Snake, Press Enter to Continue'

GameOver: db 'Game Over'

currScore: db '0'

score: db 'Score: '

snake: db 02,'@','@','@','@'

snake\_length: dw 5

direction: db 2

delaytime: dd 0

;/////////////////

clearscreen:

push es

push ax

push di

push cx

mov ax,0xb800 ; video memory address

mov es,ax

mov ax,0x0720 ; color code and space ASCII

mov di,0

nextchar:

mov [es:di],ax

add di,2

cmp di,4000

jne nextchar

;popping all values

pop cx

pop di

pop ax

pop es

ret

;=======PRINT WELCOME MSSG======

welcomeMsg:

pusha

mov ah, 0x13 ; service 13 - print string

mov al, 1 ; subservice 01 – update cursor

mov bh, 0 ; output on page 0

mov bl, 10 ; normal attrib

mov dx, 0x050D ; row 10 column 3

mov cx, 10 ; length of string

push cs

pop es ; segment of string

mov bp, gameName ; offset of string

int 0x10

mov ah, 0x13 ; service 13 - print string

mov al, 1 ; subservice 01 – update cursor

mov bh, 0 ; output on page 0

mov bl, 7 ; normal attrib

mov dx, 0x0B0D ; row 10 column 3

mov cx, 14 ; length of string

push cs

pop es ; segment of string

mov bp, GroupMember1 ; offset of string

int 0x10

mov dx, 0x0C0D

mov cx, 17

mov bp, GroupMember2

int 0x10

popa

ret

;///////////////////

Instruction:

pusha

mov ah, 0x13 ; service 13 - print string

mov al, 1 ; subservice 01 – update cursor

mov bh, 0 ; output on page 0

mov bl, 7 ; normal attrib

mov dx, 0x0A03 ; row 10 column 3

mov cx, 67 ; length of string

push cs

pop es ; segment of string

mov bp, GAMEINSTUCT ; offset of string

int 0x10

popa

ret

;//////////////////////////////////////////

; draw\_snake: Draws the snake on the screen

; Input:

; [bp + 6]: Address of the snake characters

; [bp + 4]: Address of snake\_locations

; Output:

; Updates the screen with the snake representation

draw\_snake:

push bp

mov bp, sp

pusha

mov si, [bp + 6]

mov cx, 5

mov di, 1500

mov ax, 0xb800

mov es, ax

mov bx, [bp +4]

mov ah, 0x04

snake\_next\_part:

mov al, [si]

mov [es:di], ax

mov [bx], di

inc si

add bx, 2

add di, 2

loop snake\_next\_part

popa

pop bp

ret 6

;////////////////////////

printScore:

push bp

mov bp, sp

push ax

push bx

push cx

push si

push di

push es

s:

mov ah, 0x13 ; BIOS interrupt service 13 - print string

mov al, 1 ; Subservice 01 – update cursor

mov bh, 0 ; Output on page 0

mov bl, 7 ; Normal attribute

mov dx, 0x0103 ; Row 10, Column 3

mov cx, 7 ; Length of the score string

push cs ; Push code segment to get the segment of the string

pop es ; Pop it into ES

mov bp, score ; Offset of the score string

int 0x10 ; Call BIOS interrupt to print the score string

mov dx, 0x010A ; Row 01, Column 10

mov cx, 1 ; Length of the currScore string

push cs ; Push code segment to get the segment of the string

pop es ; Pop it into ES

mov bp, currScore ; Offset of the currScore string

int 0x10 ; Call BIOS interrupt to print the currScore string

pop es ; Restore registers from the stack

pop di

pop si

pop cx

pop bx

pop ax

pop bp

ret 2 ; Return, removing 2 bytes from the stack (arguments)

;///////////////////////////////////////////////////////

move\_snake\_left:

push bp

mov bp, sp

push ax

push bx

push cx

push dx

push es

push di

push si

;snake itself parts colision check

mov bx, [bp + 4] ; snake locations

mov dx, [bx] ; snake head

mov cx, [bp + 8]; len of snake

dec cx

sub dx, 2 ; dx = 1978

check\_left\_colision:

cmp dx, [bx]

je no\_left\_movement

add bx, 2

loop check\_left\_colision

left\_movement:

mov si, [bp + 6] ;snake

mov bx, [bp + 4] ;snake location

mov dx, [bx]

sub dx, 2

mov di, dx

mov ax, 0xb800

mov es, ax

mov ah, 0x07

mov al, [si]

mov [es:di],ax ;snake head placed

mov cx, [bp + 8]

mov di, [bx]

inc si

mov ah, 0x07

mov al, [si]

mov [es:di],ax

left\_location\_sort:

mov ax, [bx]

mov [bx], dx

mov dx, ax

add bx, 2

loop left\_location\_sort

mov di, dx

mov ax, 0x0720

mov [es:di], ax

jmp end1

no\_left\_movement:

call exitGame

end1:

pop si

pop di

pop es

pop dx

pop cx

pop bx

pop ax

pop bp

ret 6

;///////////////////////////////////////////////

move\_snake\_up:

push bp

mov bp, sp

push ax

push bx

push cx

push dx

push es

push di

push si

;snake\_parts colision detection

mov bx, [bp + 4] ;snake location

mov dx, [bx]

mov cx, [bp + 8]

dec cx

sub dx, 160

check\_up\_colision:

cmp dx, [bx]

je no\_up\_movement

add bx, 2

loop check\_up\_colision

upward\_movement:

mov si, [bp + 6] ;snake

mov bx, [bp + 4] ;snake location

mov dx, [bx]

sub dx, 160

mov di, dx

mov ax, 0xb800

mov es, ax

mov ah, 0x07 ;snake color for head

mov al, [si]

mov [es:di],ax ;snake head placed

mov cx, [bp + 8]

mov di, [bx]

inc si

mov ah, 0x07 ;snake color

mov al, [si]

mov [es:di],ax

up\_location\_sort:

mov ax, [bx]

mov [bx], dx

mov dx, ax

add bx, 2

loop up\_location\_sort

mov di, dx

mov ax, 0x0720

mov [es:di], ax

jmp end2

no\_up\_movement:

call exitGame

end2:

pop si

pop di

pop es

pop dx

pop cx

pop bx

pop ax

pop bp

ret 6

;//////////////////////////////////////////////

move\_snake\_down:

push bp

mov bp, sp

push ax

push bx

push cx

push dx

push es

push di

push si

;snake\_parts colision detection

mov bx, [bp + 4] ;snake location

mov dx, [bx]

mov cx, [bp + 8]

dec cx

add dx, 160

check\_down\_colision:

cmp dx, [bx]

je no\_down\_movement

add bx, 2

loop check\_down\_colision

downward\_movement:

mov si, [bp + 6] ;snake

mov bx, [bp + 4] ;snake location

mov dx, [bx]

add dx, 160

mov di, dx

mov ax, 0xb800

mov es, ax

mov ah, 0x07

mov al, [si]

mov [es:di], ax ;snake head placed

mov cx, [bp + 8] ;snake length

mov di, [bx]

inc si

mov ah, 0x07

mov al, [si]

mov [es:di],ax

down\_location\_sort:

mov ax, [bx]

mov [bx], dx

mov dx, ax

add bx, 2

loop down\_location\_sort

mov di, dx

mov ax, 0x0720

mov [es:di], ax

jmp end3

no\_down\_movement:

call exitGame

end3:

pop si

pop di

pop es

pop dx

pop cx

pop bx

pop ax

pop bp

ret 6

;-///////////////////////////////////

move\_snake\_right:

push bp

mov bp, sp

push ax

push bx

push cx

push dx

push es

push di

push si

;snake\_parts colision detection

mov bx, [bp + 4] ;snake location

mov dx, [bx]

mov cx, [bp + 8]

dec cx

add dx, 2

check\_right\_colision:

cmp dx, [bx]

je no\_right\_movement

add bx, 2

loop check\_right\_colision

right\_movement:

mov si, [bp + 6] ;snake

mov bx, [bp + 4] ;snake location

mov dx, [bx]

add dx, 2

mov di, dx

mov ax, 0xb800

mov es, ax

mov ah, 0x07

mov al, [si]

mov [es:di], ax ;snake head placed

mov cx, [bp + 8] ;snake length

mov di, [bx]

inc si

mov ah, 0x07

mov al, [si]

mov [es:di],ax

right\_location\_sort:

mov ax, [bx]

mov [bx], dx

mov dx, ax

add bx, 2

loop right\_location\_sort

mov di, dx

mov ax, 0x0720

mov [es:di], ax

jmp end4

no\_right\_movement:

call exitGame

end4:

pop si

pop di

pop es

pop dx

pop cx

pop bx

pop ax

pop bp

ret 6

;//////////////////////////////////////

check\_death:

push ax

push di

push cx

mov ax, [snake\_locations]

cmp ax, 160

jb finished

mov di, 160

mov cx, 24

check1:

cmp ax, di

je finished

add di, 158

cmp ax, di

je finished

add di, 2

loop check1

mov di,3840

cmp ax, di

ja finished

jmp else

finished:

call exitGame

else:

pop cx

pop di

pop ax

ret

;////////////////////////////////////

; Play Game Routine

play\_game:

; Initial setup

call clearscreen ; Clear the screen

call welcomeMsg ; Display welcome message

mov ah,00 ; Set interrupt AH to 00

int 16h ; Wait for user input

call clearscreen ; Clear the screen again

call Instruction ; Display game instructions

mov ah,00 ; Set interrupt AH to 00

int 16h ; Wait for user input

; Game initialization

call clearscreen

call draw\_border ; Draw the game border

; Draw initial snake and food

push word [snake\_length]

mov bx, snake

push bx

mov bx, snake\_locations

push bx

call draw\_snake ; Draw the snake

call displayFood ; Display food on the screen

; Main game loop

repeat:

push 164 ; Print the score

call printScore

cmp byte[currScore],'5' ; Check if current score is 5 or more

jae leve1 ; Jump to leve1 if true

mov dword[delaytime],120000 ; Set delay time

leve1:

mov dword[delaytime],120000 ; Set delay time

delay:

dec dword[delaytime] ; Delay loop

cmp dword[delaytime],0

jne delay

mov ah, 01h ; Check for keyboard input

int 16h

jz noKey ; Jump to noKey if no key pressed

mov ah,0

int 16h ; Read the key

; Check arrow key input

cmp ah,0x48 ; up arrow

je up

cmp ah,0x4B ; left arrow

je left

cmp ah,0x4D ; right arrow

je right

cmp ah,0x50 ; down arrow

je down

cmp ah, ' '

jne repeat ; Repeat loop if key is not space

mov ah,0x4c

int 21h ; Exit program

; Handle no key pressed scenario

noKey:

cmp byte[direction], 0 ; Check current direction

je up

cmp byte[direction], 1 ; Check current direction

je down

cmp byte[direction], 2 ; Check current direction

je left

cmp byte[direction], 3 ; Check current direction

je right

; Move snake based on direction

up:

mov byte[direction],0

push word [snake\_length]

mov bx, snake

push bx

mov bx, snake\_locations

push bx

call move\_snake\_up

jmp new

down:

mov byte[direction],1

push word [snake\_length]

mov bx, snake

push bx

mov bx, snake\_locations

push bx

call move\_snake\_down

jmp new

left:

mov byte[direction],2

push word [snake\_length]

mov bx, snake

push bx

mov bx, snake\_locations

push bx

call move\_snake\_left

jmp new

right:

mov byte[direction],3

push word [snake\_length]

mov bx, snake

push bx

mov bx, snake\_locations

push bx

call move\_snake\_right

new:

call check\_death ; Check for snake collision with itself or borders

push ax

mov ax,word[foodposition]

cmp ax,[snake\_locations]

jne f

call displayFood ; Display new food

add word[snake\_length],1 ; Increase snake length

add byte[currScore],1 ; Increase the score

f:

pop ax

jmp repeat ; Repeat the main game loop

exit:

pop bx

pop ax

ret

;/////////////////////////////////////////

displayFood:

push bp

push bx

push ax

push cx

push dx

push es

push di

l1:

MOV AH, 00h ; interrupts to get system time

INT 1AH ; CX:DX now hold number of clock ticks since midnight

mov ax, dx

xor dx, dx

mov cx, 25

div cx

mov word[x\_pos],dx

MOV AH, 00h ; interrupts to get system time

INT 1AH ; CX:DX now hold number of clock ticks since midnight

mov ax, dx

xor dx, dx

mov cx, 80

div cx ; here dx contains the remainder of the division - from 0 to 9

mov word[y\_pos],dx

mov ax,[x\_pos]

mov bx,80

mul bx

add ax,[y\_pos]

shl ax,1

cmp ax,3840

jg l1

cmp ax,190

jb l1

mov word[foodposition],ax

mov di,ax

mov ax,0xb800

mov es,ax

mov ax,0x050A

mov [es:di],ax

pop di

pop es

pop dx

pop cx

pop ax

pop bx

pop bp

ret

;/////////////////////////////

draw\_border:

push ax

push bx

push es

push di

push si

push cx

mov ax,0xb800

mov es,ax

mov di,0

mov cx,80

mov ah,0x04

mov al,'='

top\_border:

mov [es:di],ax

add di,2

loop top\_border

mov cx,80

mov di,3840

mov al,'='

bottom\_border:

mov [es:di],ax

add di,2

loop bottom\_border

mov cx,24

mov al,'|'

mov di,160

left\_border:

mov [es:di],ax

add di,160

loop left\_border

mov cx,24

mov al,'|'

mov di,158

right\_border:

mov [es:di],ax

add di,160

loop right\_border

pop cx

pop si

pop di

pop es

pop bx

pop ax

ret

;////////////////////////////////////////

exitGame:

call clearscreen

pusha

mov ah, 0x13 ; BIOS interrupt service 13 - print string

mov al, 1 ; Subservice 01 – update cursor

mov bh, 0 ; Output on page 0

mov bl, 7 ; Normal attribute

mov dx, 0x0A0D ; Row 10, Column 3

mov cx, 9 ; Length of the Game Over string

push cs ; Push code segment to get the segment of the string

pop es ; Pop it into ES

mov bp, GameOver ; Offset of the Game Over string

int 0x10 ; Call BIOS interrupt to print the Game Over string

popa

mov ax, 0x4c00

int 0x21

ret

;//////////////////////////////////////////

start:

call play\_game

mov ax,0x4c00

int 0x21

snake\_locations: dw 0