SRM Institute of Science and Technology
College of Engineering and Technology

SCHOOL OF COMPUTING

MINI PROJECT REPORT

ODD Semester: 2024-2025

Lab code & Sub Name : 21CSS201T & Computer Organization and Architecture

Year & Semester : II & III

Project Title :

Team Members :
1. Arunava Mondal (RA2311003010858)
2. Gowtham K (RA2311003010799)
3. Naghul Pranav K R (RA2311003010802)

Particulars	**Max. Marks**	**Marks Obtained**
		Name:
		Register No:
Review 1 and 2	05	
Demo verification &viva	03	
Project Report	02	
Total	**10**	

Date : **22 October 21, 2024**

Staff Name : **Divya Mohan Ma'am**

Signature :

SNAKE GAME USING 8086 PROCESSOR

OBJECTIVE:

To develop a classic Snake Game using the 8086 processor in assembly language, handling snake movement, user input, and score tracking through efficient use of the processor's capabilities.

ABSTRACT:

This project involves creating a Snake Game using the 8086 microprocessor, programmed in assembly language.

The game allows players to control the movement of the snake using keyboard input, with the goal of consuming food items and growing the snake while avoiding collisions. The game logic handles real-time input processing, score calculation, and game termination when the snake hits a boundary or itself. By leveraging the 8086 processor's instruction set and hardware interrupts, this project demonstrates efficient resource management and low-level hardware interaction, providing a practical understanding of microprocessor-based game development.

INTRODUCTION:

The Snake Game is a popular arcade game where players navigate a growing snake to consume food while avoiding obstacles. This project aims to implement the Snake Game using the 8086 microprocessor, a widely used microcontroller in early computing.

Written in assembly language, the game demonstrates the principles of low-level programming, utilizing the 8086-instruction set, registers, and interrupts to handle the game's logic, input control, and display. Through this project, we explore fundamental concepts such as memory management, hardware interfacing, and the real-time processing capabilities of the 8086 processor.

HARDWARE/SOFTWARE REQUIREMENTS:

- Hardware:
  - 8086 Microprocessor.
  - Standard PC with support for DOSBox.
  - Keyboard for user input.

- Software:
  - IDE: Visual Studio Code (VS Code) for writing and editing assembly code.
  - Assembler: NASM (Netwide Assembler) for assembling the 8086 assembly code.
  - Emulator: DOSBox for simulating the DOS environment to run and test the 8086 code.
  - Operating System: DOSBox running on Windows for executing the game.

CONCEPTS/WORKING PRINCIPLE

The Snake Game on the 8086 processor operates using the following key concepts:

- Assembly Language Programming:
  The game logic, movement, and input handling are programmed in assembly language using the 8086-instruction set, providing direct control over the processor's registers, memory, and hardware.

- Real-Time User Input:
  The game uses keyboard interrupts to capture real-time input for controlling the snake's movement. Specific interrupt services are used to read and process the key presses for up, down, left, and right directions.

- Snake Movement and Collision Detection:
  The snake's position is stored in memory and updated continuously based on user input. Collision detection is implemented by checking if the snake hits the boundaries of the game area or itself, which results in game termination.

- Graphics Handling:
  The game screen is drawn using simple character-based graphics. The snake, food, and boundary are represented by specific characters on the console, which are updated as the game progresses.

- Score and Game Over Logic:
  The game tracks the player's score based on the number of food items consumed by the snake. When a collision occurs, the game displays the final score and terminates, indicating the end of the session.

By utilizing interrupts, memory management, and direct hardware manipulation, this project demonstrates key aspects of programming at the hardware level using the 8086 processor.

APPROACH

- **Design the Game Structure:**
  - Outline the main components of the game, including the snake, food, boundaries, and score.
  - Define the game states: running, paused, and game over.

- **Set Up the Development Environment:**
  - Install NASM for assembling the assembly code.
  - Use Visual Studio Code for code editing, ensuring syntax highlighting and debugging features are available.
  - Configure DOSBox to create an appropriate environment for running the 8086-assembly code.

- **Implement Keyboard Input Handling:**
  - Use interrupts to capture keyboard input in real-time.
  - Map specific keys (e.g., arrow keys) to control the snake's direction.

- **Develop Game Logic:**
  - Create functions for initializing the game, updating the snake's position, checking for collisions, and generating food.
  - Implement score tracking by incrementing the score whenever the snake consumes food.

- **Render Graphics:**
  - Use character graphics to represent the snake, food, and game boundaries on the screen.
  - Continuously refresh the display to reflect changes in the game state, such as snake movement and food consumption.

- **Collision Detection:**
  - Implement logic to check if the snake's head collides with the boundaries or its own body.
  - Handle game over conditions by displaying the final score and providing options to restart or exit.

- **Testing and Debugging:**
  - Run the game in DOSBox to test functionality, performance, and user experience.
  - Debug any issues encountered during gameplay, adjusting the code as necessary to ensure smooth operation.

- **Documentation:**
  - Document the code for clarity and future reference, including explanations of key functions and logic used throughout the game development.

By following this structured approach, the project aims to effectively create a functional and engaging Snake Game while enhancing understanding of assembly language programming and microprocessor architecture.

METHODOLOGY:

- **Requirements Analysis:**
  Identify key features of the Snake Game, including user input handling, snake movement, collision detection, and score tracking.

- **System Design:**
  Create a flowchart to outline the game logic and design data structures for the snake’s position, food coordinates, and score.

- **Coding:**
  Write assembly code using NASM to implement game functionalities, organizing the code into modular functions for clarity and maintainability.

- **Testing:**
  Conduct unit and integration testing to verify the functionality of individual components and the overall game performance, ensuring stability and responsiveness.

- **Documentation:**
  Document the development process, including design decisions, code explanations, and user instructions, to provide clarity for future reference and user guidance.

CODE SNIPPETS:

Here are a few code snippets used in the program to run certain tasks:

- **Initialization of VGA Graphics:**

Initialization of VGA Graphics

```
init13h:
    MOV AX,0013H            ; set
320x200 8bit video mode
    INT 10H
    RET
```

- **Keyboard Interrupt Handler:**

Keyboard Interrupt Handler

```
keyboard_int:
    STI
    PUSH AX
    IN AL,60H               ; get key
from keyboard port
    MOV [CS:lastkey],AL
    POP AX
    ; ... (handling key press logic)
    IRET
```

- **Moving the Snake:**

Moving the Snake

```
move_snake:
    PUSH AX
    PUSH BX
    CMP BYTE [snake_vector],0
    JE move_snake_end
    ; ... (logic to move the snake
based on direction)
move_snake_end:
    POP BX
    POP AX
    RET
```

- **Drawing a Pixel:**

Pixel

```
putpixel:
    PUSH AX
    PUSH BX
    PUSH CX
    ; ... (calculating pixel position)
    MOV [ES:DI],CL          ; set pixel
color
    POP CX
    POP BX
    POP AX
    RET
```

- **Collision Detection:**

```
Collision

detect_collisions:
    PUSH AX
    PUSH BX
    ; ... (checking for collisions with
food and walls)
    JMP detect_collisions_end
detect_collisions_end:
    POP BX
    POP AX
    RET
```

- **Game Over Logic:**

```
Game Over

game_over:
    MOV CL,128
    MOV AX,320 / 2 - 9 * 8 / 2
    MOV BX,95
    MOV DX,text_game_over
    CALL draw_text
    CALL draw_screen
    CALL wait_for_any_key
    JMP exit
```

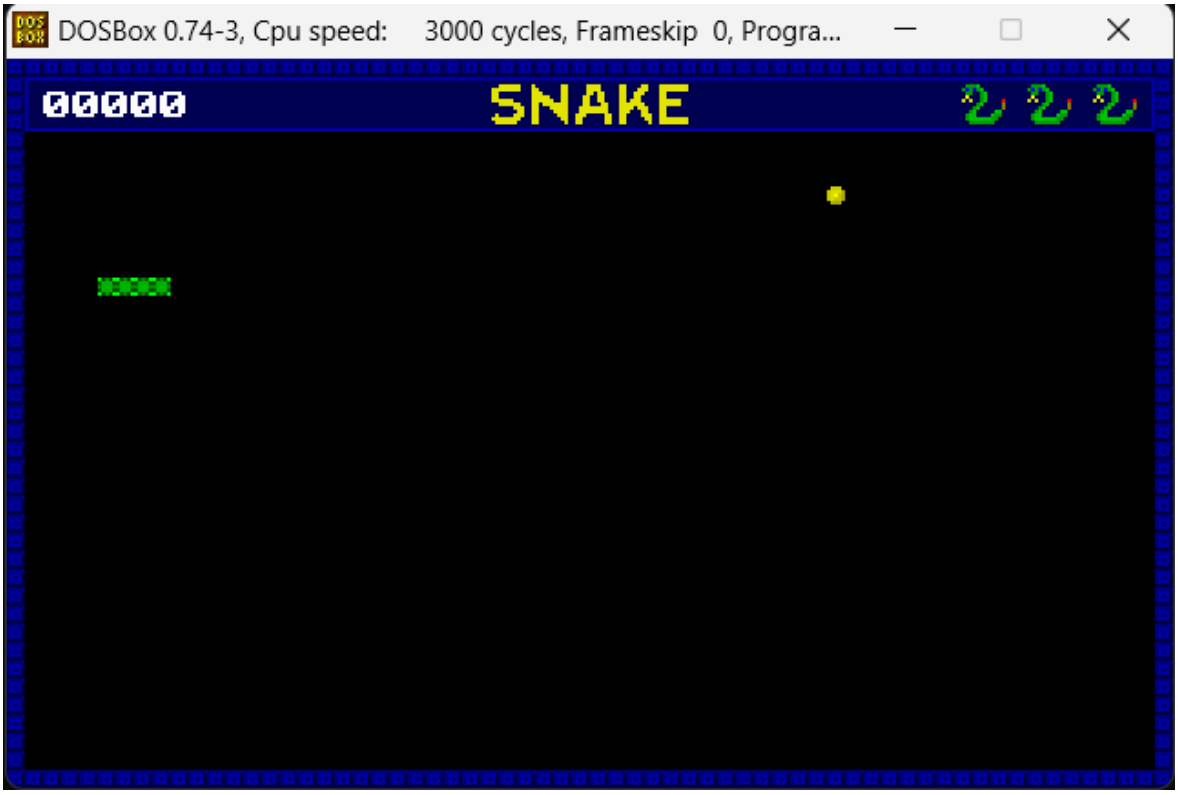
- **Random Number Generation:**

Random

```
random_gen:
    MOV CX,[random_seed]
    IMUL CX,13A7H
    INC CX
    MOV [random_seed],CX
    MOV CL,CH
    MOV CH,0
    RET
```

OUTPUT:

The Snake Game for the 8086 processor, developed using NASM in an emulated DOSBox environment, is a classic console-based game where the player controls a growing snake. Each time the snake consumes food, it increases in length, making navigation progressively more challenging. The objective is to keep the snake moving without colliding with the walls or its own body, providing an engaging experience that demonstrates efficient memory and control handling on the 8086 architecture. This project showcases fundamental assembly language concepts and low-level programming skills in creating interactive gameplay.



CONCLUSIONS:

In conclusion, this Snake Game project effectively demonstrates the fundamentals of assembly programming on the 8086 processor. It highlights key concepts such as memory management, control structures, and real-time input handling, offering a practical application of COA principles in creating an interactive, low-level game.

REFERENCES:

- **"The 8086/8088 Microprocessor: Programming, Interfacing, Software, Hardware, and Applications"** by Walter A. Triebel and Avtar Singh.
- **NASM Documentation** – Official documentation for the Netwide Assembler (NASM), useful for syntax, commands, and understanding assembly language structures: NASM Documentation.
- **"IBM PC Assembly Language and Programming"** by Peter Abel – A classic resource for learning assembly language programming, including examples and explanations relevant to the 8086 architecture.
- **DOSBox Wiki** – The DOSBox emulator is widely used to run and test assembly programs on modern systems: DOSBox Wiki.
- **Online Assembly Language Tutorials** – Websites like **TutorialsPoint** and **GeeksforGeeks** offer beginner-friendly guides to assembly language, covering foundational concepts for building projects like the Snake Game.