# Snake Game in Assembly Language (8086)

## Introduction

This project is a simple implementation of the classic Snake Game using the 8086 assembly language. It is designed to run in text mode on emu8086, a popular emulator for learning low-level x86 programming. The game showcases basic game loop logic, input handling, screen rendering, and state management using low-level interrupts. The snake is displayed as a single moving 'O' character on the screen that reacts to user input via WASD keys. A '\*' represents the food. Although this is a simplified version without collision detection or snake growth, it effectively demonstrates core principles of interactive console-based game development in 8086 assembly.

## Logic Diagram

1. Initialize data segment and set video text mode (03h).  
2. Draw initial food on screen.  
3. Enter game loop:  
 ├── Draw snake at current position  
 ├── Introduce delay  
 ├── Erase snake from current position  
 ├── Check for keypress (non-blocking)  
 ├── Update snake position based on direction  
 └── Repeat.

## Block Diagram Explanation (Microprocessor)

The block diagram of the microprocessor for this game includes the following interactions:  
  
1. \*\*CPU Registers (AX, BX, CX, DX):\*\* Used for storing and processing positions, directions, and function parameters.  
2. \*\*Memory Segment (DS):\*\* Contains variables for snake head coordinates, food position, and direction.  
3. \*\*INT 10h (BIOS Video Services):\*\* Handles screen cursor movement and character display.  
4. \*\*INT 16h (Keyboard Input):\*\* Handles keyboard input detection and character reading.  
5. \*\*Main Game Loop:\*\* Runs continuously and performs the draw-delay-erase-move sequence to animate the snake.

## Assembly Code (8086 - emu8086 Compatible)

.model small  
.stack 100h  
.data  
snakeHead db 24, 10  
foodX db 30  
foodY db 10  
direction db 0 ; 0=right, 1=down, 2=left, 3=up  
  
.code  
main:  
 mov ax, @data  
 mov ds, ax  
  
 call SetTextMode  
 call DrawFood  
  
MainLoop:  
 call DrawSnake  
 call Delay  
 call EraseSnake  
 call GetKeyPress  
 call MoveSnake  
 jmp MainLoop  
  
SetTextMode:  
 mov ah, 0  
 mov al, 03h  
 int 10h  
 ret  
  
DrawSnake:  
 mov ah, 02h  
 mov bh, 0  
 mov dh, [snakeHead+1]  
 mov dl, [snakeHead]  
 int 10h  
  
 mov ah, 09h  
 mov al, 'O'  
 mov bh, 0  
 mov bl, 2  
 mov cx, 1  
 int 10h  
 ret  
  
EraseSnake:  
 mov ah, 02h  
 mov bh, 0  
 mov dh, [snakeHead+1]  
 mov dl, [snakeHead]  
 int 10h  
  
 mov ah, 09h  
 mov al, ' '  
 mov bh, 0  
 mov bl, 0  
 mov cx, 1  
 int 10h  
 ret  
  
MoveSnake:  
 cmp direction, 0  
 je moveRight  
 cmp direction, 1  
 je moveDown  
 cmp direction, 2  
 je moveLeft  
 cmp direction, 3  
 je moveUp  
 ret  
  
moveRight:  
 mov bx, offset snakeHead  
 inc byte ptr [bx]  
 ret  
  
moveDown:  
 mov bx, offset snakeHead  
 inc byte ptr [bx+1]  
 ret  
  
moveLeft:  
 mov bx, offset snakeHead  
 dec byte ptr [bx]  
 ret  
  
moveUp:  
 mov bx, offset snakeHead  
 dec byte ptr [bx+1]  
 ret  
  
GetKeyPress:  
 mov ah, 1  
 int 16h  
 jz noKey  
  
 mov ah, 0  
 int 16h  
 cmp al, 'w'  
 je setUp  
 cmp al, 'a'  
 je setLeft  
 cmp al, 's'  
 je setDown  
 cmp al, 'd'  
 je setRight  
 cmp al, 27  
 je Exit  
 jmp noKey  
  
setUp:  
 mov direction, 3  
 jmp noKey  
setLeft:  
 mov direction, 2  
 jmp noKey  
setDown:  
 mov direction, 1  
 jmp noKey  
setRight:  
 mov direction, 0  
  
noKey:  
 ret  
  
DrawFood:  
 mov ah, 02h  
 mov bh, 0  
 mov dh, [foodY]  
 mov dl, [foodX]  
 int 10h  
  
 mov ah, 09h  
 mov al, '\*'  
 mov bh, 0  
 mov bl, 4  
 mov cx, 1  
 int 10h  
 ret  
  
Delay:  
 mov cx, 0FFFFh  
delayLoop:  
 nop  
 loop delayLoop  
 ret  
  
Exit:  
 mov ah, 4Ch  
 int 21h  
end main