Computer Organization and Assembly Language

Project Report

Section: A

Submitted by:

Fiza Manzoor	2023-CS-612
Samar Noor Riaz	2023-CS-630

Submitted to:

Mam. Rimsha Noreen

Dated: 6th May 2025

Department of Computer Science

University of Engineering and Technology Lahore, New Campus

🎶 Assembly-based Hangman Game 🎶

Objective:

The Hangman game project is an educational and interactive application developed in x86 assembly language using the Irvine32 library. The game challenges players to guess a hidden word by suggesting letters, with a graphical hangman figure updating based on incorrect guesses. The project is modular, split across multiple files (hangman.asm, welcome.asm, display.asm, endwithresult.asm, start.asm) and a header file (display.inc), demonstrating assembly programming concepts such as data manipulation, control structures, and procedure calls.

Project Structure:

The project is organized into the following files:

- **hangman.asm**: Main game logic, including game loop, state updates, and user input handling.
- **welcome.asm**: Displays the welcome screen with the game title in ASCII art.
- **display.asm**: Handles the graphical display of the hangman figure based on remaining lives.
- **endwithresult.asm**: Manages the endgame logic, displaying win/lose messages and prompting for replay.
- **start.asm**: Initializes the game state and sets up the initial display.
- **display.inc**: Header file containing procedure prototypes and constants.

Data Section (hangman.asm):

The data section in hangman.asm defines the variables used throughout the game:

- **targetWord BYTE "COMPUTER",0**: The word to guess, null-terminated.
- **wordLength EQU ($ - targetWord) - 1**: Calculates the length of the target word (excluding the null terminator).
- **displayWord BYTE wordLength DUP('_'),0**: Stores the current state of the word with underscores for unguessed letters.
- **livesLeft DWORD 6**: Tracks the number of remaining incorrect guesses.
- **correctGuess BYTE "Correct guess!",0**: Message for correct guesses.
- **incorrectGuess BYTE "Incorrect guess!",0**: Message for incorrect guesses.
- **prompt BYTE "Guess a letter: ",0**: Prompt for user input.
- **winMessage BYTE "Congratulations! You won!",0**: Message displayed on winning.
- **loseMessage BYTE "Game over! The word was: ",0**: Message displayed on losing.
- **alreadyGuessed BYTE "You already guessed that letter!",0**: Message for repeated guesses.
- **guessedLetters BYTE 26 DUP(0)**: Tracks guessed letters (0 = not guessed, 1 = guessed).

- **initiak_col BYTE 20, initial_row BYTE 12**: Initial cursor position for displaying the word.
- **slate BYTE "------------",0**: Decorative line for the game UI.
- **playAgain BYTE 0**: Flag to determine if the player wants to replay.

Macros (hangman.asm)

Macros were added to simplify repetitive tasks and improve code readability:

- **DisplayString MACRO strAddr**: Displays a string at the address strAddr and moves to the next line.
    - Usage: DisplayString OFFSET displayWord
- **SetCursor MACRO row, col**: Sets the cursor position to the specified row and column.
    - Usage: SetCursor initial_row, initiak_col
- **ResetArray MACRO arrAddr, arrSize, val**: Resets an array at arrAddr with size arrSize to the value val.
    - Usage: ResetArray OFFSET displayWord, wordLength, '_'
- **HandleIncorrectGuess MACRO**: Displays the incorrect guess message and decrements livesLeft.
    - Usage: HandleIncorrectGuess

Main Procedure (hangman.asm)

main PROC

The main procedure orchestrates the game flow:

1. **GameRestart**: Resets the game state:
    - Sets livesLeft to 6, initial_row to 12, initiak_col to 20, and playAgain to 0.
    - Resets displayWord to underscores using ResetArray.
    - Resets guessedLetters to 0 using ResetArray.
    - Clears the screen and calls Welcome, Start, and displays the slate.
2. **GameLoop**: The main game loop:
    - Sets the cursor position using SetCursor.
    - Displays the current displayWord using DisplayString.
    - Shows remaining lives and a prompt using DisplayString.
    - Checks win/lose conditions by calling CheckWinCondition and comparing livesLeft.
    - Gets user input via GetUserGuess and updates the game state with UpdateGameState.
    - Calls displayHangman to update the hangman figure.
    - Adjusts the row for the next iteration and loops back.
3. **GameWon**: Handles the win scenario:
    - Displays the win message and calls endgame to check for replay.
    - Restarts if playAgain is 1; otherwise, exits.

4. **GameLost**: Handles the lose scenario:
    - Displays the lose message and the target word, calls endgame, and restarts if playAgain is 1.
5. **ExitGame**: Exits the program.

UpdateGameState PROC

Updates the game state based on the user's guess (letter in AL):

- Checks if the letter was already guessed by indexing into guessedLetters.
- Marks the letter as guessed.
- Searches targetWord for the letter, updating displayWord if found.
- If the letter is not found, calls HandleIncorrectGuess to display the incorrect message and decrement lives.
- If found, displays the correct guess message.

GetUserGuess PROC

Handles user input:

- Displays the prompt and reads a character.
- Converts lowercase letters to uppercase.
- Validates that the input is a letter (A-Z); if not, loops back to prompt again.
- Returns the uppercase letter in AL.

CheckWinCondition PROC

Checks if the player has won:

- Loops through displayWord to check for remaining underscores.
- Returns 1 in EAX if no underscores remain (win); otherwise, returns 0.

Welcome Procedure (welcome.asm)

Welcome PROC

Displays the game's title in ASCII art:

- Uses six strings (hangman1 to hangman6) to form the word "HANGMAN" in a stylized format.
- Positions each line at row 2 to 7, column 14, using Gotoxy.
- Writes each string using WriteString.
- Waits for a key press (ReadChar) before returning.

Display Procedure (display.asm):

displayHangman PROC

Displays the hangman figure based on the number of lives remaining (passed as a parameter):

- **lives = 5**: Displays the gallows pole ("|").
- **lives = 4**: Adds the head ("O").
- **lives = 3**: Adds the neck ("|").
- **lives = 2**: Adds the body ("|").
- **lives = 1**: Adds the arms ("/ ").
- **lives = 0**: Adds the legs ("/ ").
- Each part is positioned at a specific row and column using Gotoxy and displayed using WriteString.

Header File (display.inc):

Defines procedure prototypes and constants:

- **DisplayGameState PROTO**: Prototype for displaying the game state (not implemented in provided code).
- **MAX_GUESSES EQU 6**: Constant for maximum incorrect guesses.
- **displayHangman PROTO, lives:DWORD**: Prototype for displaying the hangman figure.
- **Welcome PROTO, Start PROTO, endgame PROTO, main PROTO**: Prototypes for other procedures.

Additional Files:

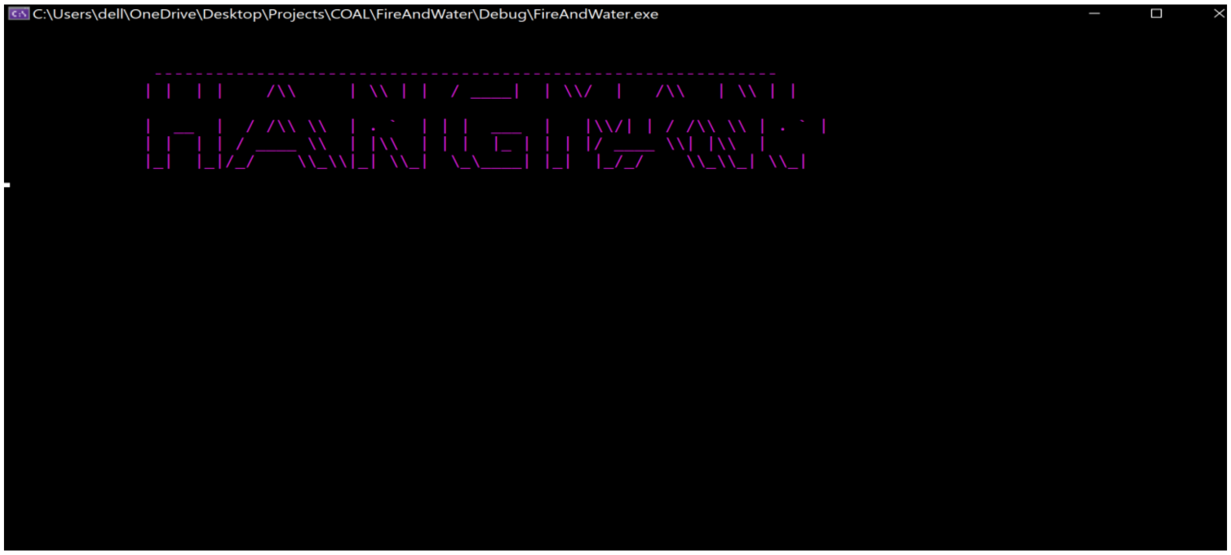
- **endwithresult.asm**: Contains the endgame PROC, which displays the final message (win or lose) and prompts the user to play again, setting the playAgain flag accordingly.
- **start.asm**: Contains the Start PROC, which initializes the game UI and prepares for the game loop.

Assembly Concepts Demonstrated:

- **Data Manipulation**: Use of BYTE, DWORD, and EQU for defining variables and constants.
- **Control Structures**: Loops (loop instruction) and conditional jumps (cmp, je, jne).
- **Procedure Calls**: Passing parameters (e.g., lives to displayHangman) and using INVOKE.
- **String Handling**: Character comparison and array manipulation for guessed letters.
- **Console I/O**: Cursor positioning (Gotoxy), string output (WriteString), and input (ReadChar).
- **Macros**: Reusable code blocks for common tasks like displaying strings and resetting arrays.

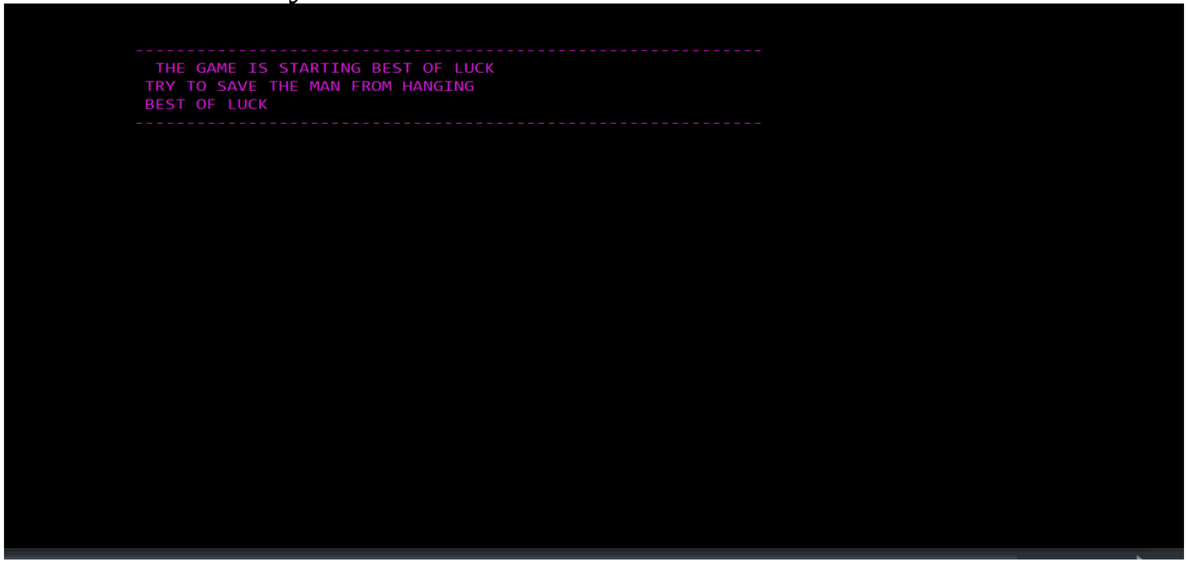
Program Execution Flow:

1. Display Start Game Home Screen:



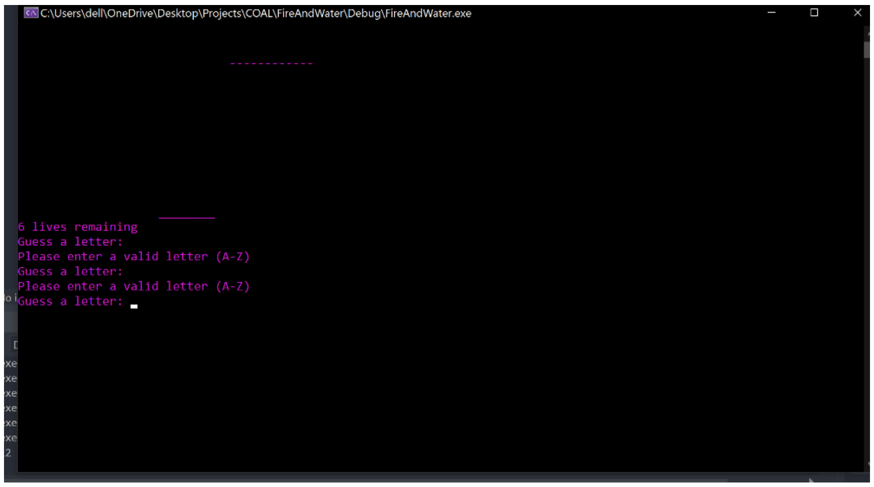
```
C:\Users\dell\OneDrive\Desktop\Projects\COAL\FireAndWater\Debug\FireAndWater.exe

    ------------------------------------------------------------
    | |  | |    /\\     | \\ | |  / ____|  | \\/  |    /\\     | \\ | |
    |  __  |  / /\\ \\   | . ` | | |  __   | |\\/| |  / /\\ \\  | . ` |
    | |  | | / ____ \\  | |\\  | | |__| |  | |  | | / ____ \\ | |\\  |
    |_|  |_|/_/    \\_\\ |_| \\_|  \\_____|  |_|  |_|/_/    \\_\\|_| \\_|
```

2. Game Start Delay:



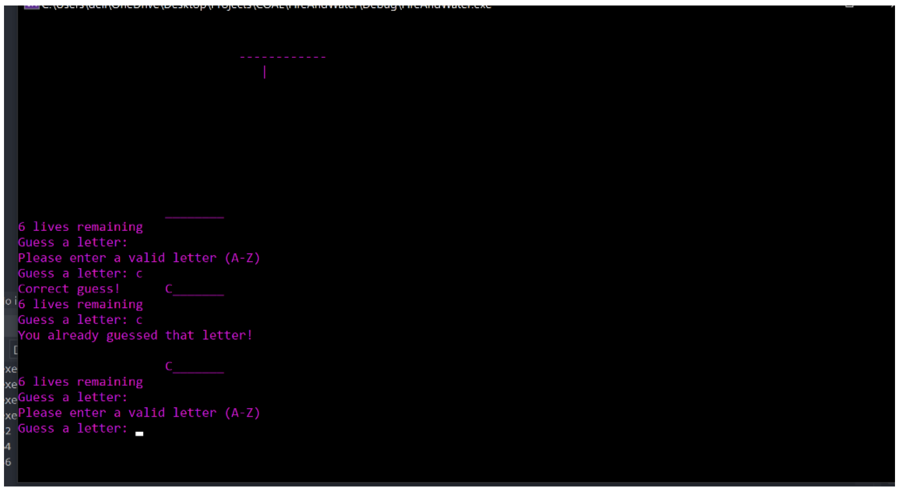
```
    ---------------------------------------------------------------
      THE GAME IS STARTING BEST OF LUCK
     TRY TO SAVE THE MAN FROM HANGING
     BEST OF LUCK
    ---------------------------------------------------------------
```

3. Asking user for valid guess:



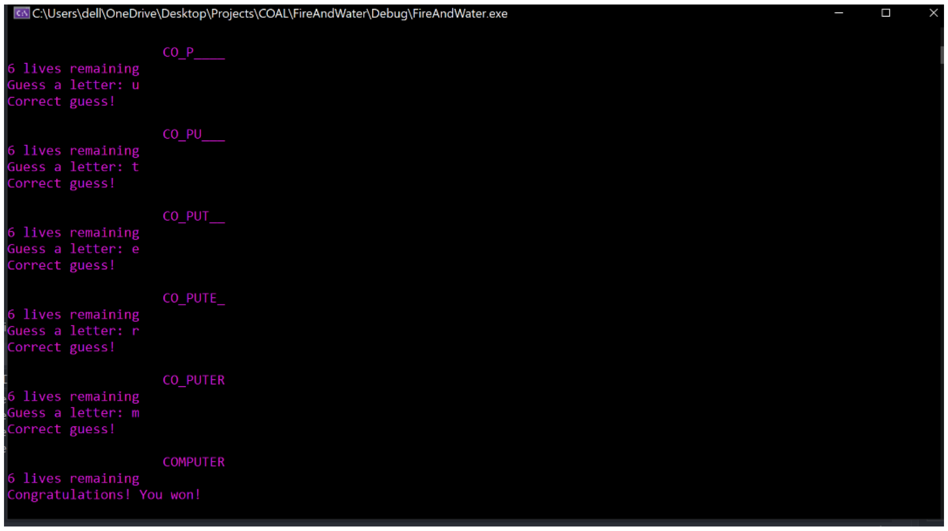
```
C:\Users\dell\OneDrive\Desktop\Projects\COAL\FireAndWater\Debug\FireAndWater.exe

                              -----------

                    _______
6 lives remaining
Guess a letter:
Please enter a valid letter (A-Z)
Guess a letter:
Please enter a valid letter (A-Z)
Guess a letter:
```

4. If letter is already guessed:



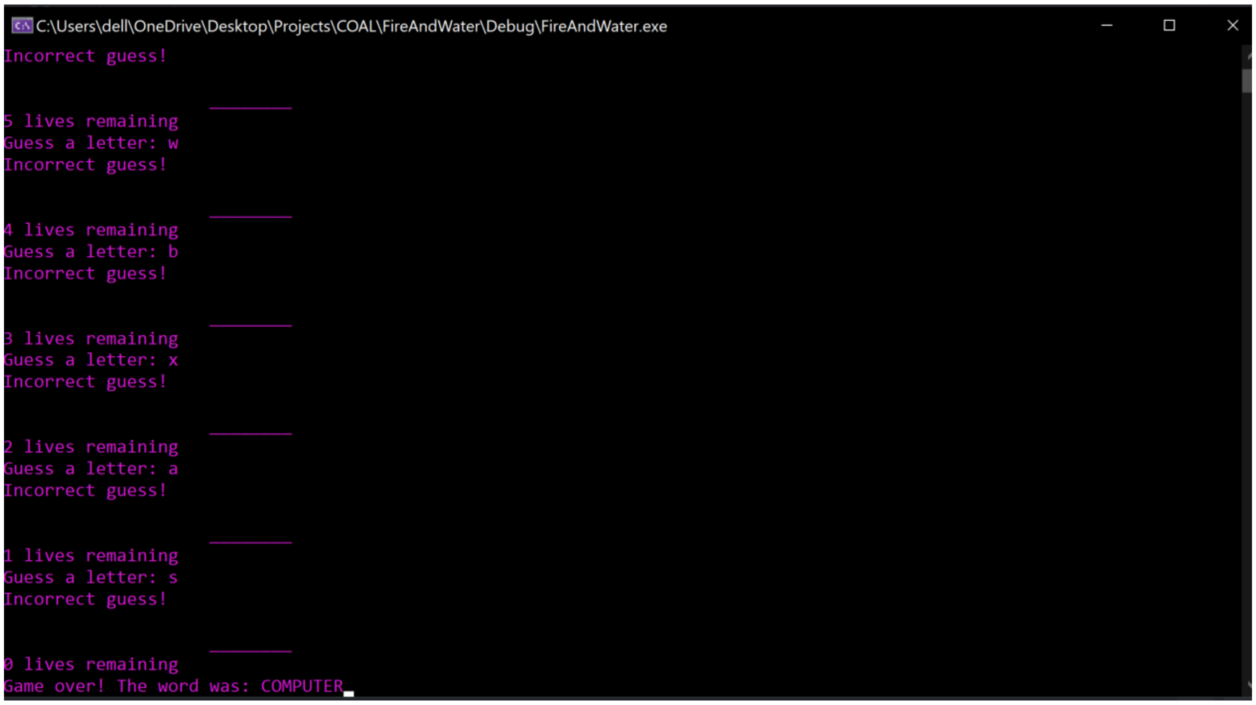
```
                              -----------
                                 |

                    ________
6 lives remaining
Guess a letter:
Please enter a valid letter (A-Z)
Guess a letter: c
Correct guess!      C_______
6 lives remaining
Guess a letter: c
You already guessed that letter!

                    C_______
6 lives remaining
Guess a letter:
Please enter a valid letter (A-Z)
Guess a letter:
```

5. Game Won



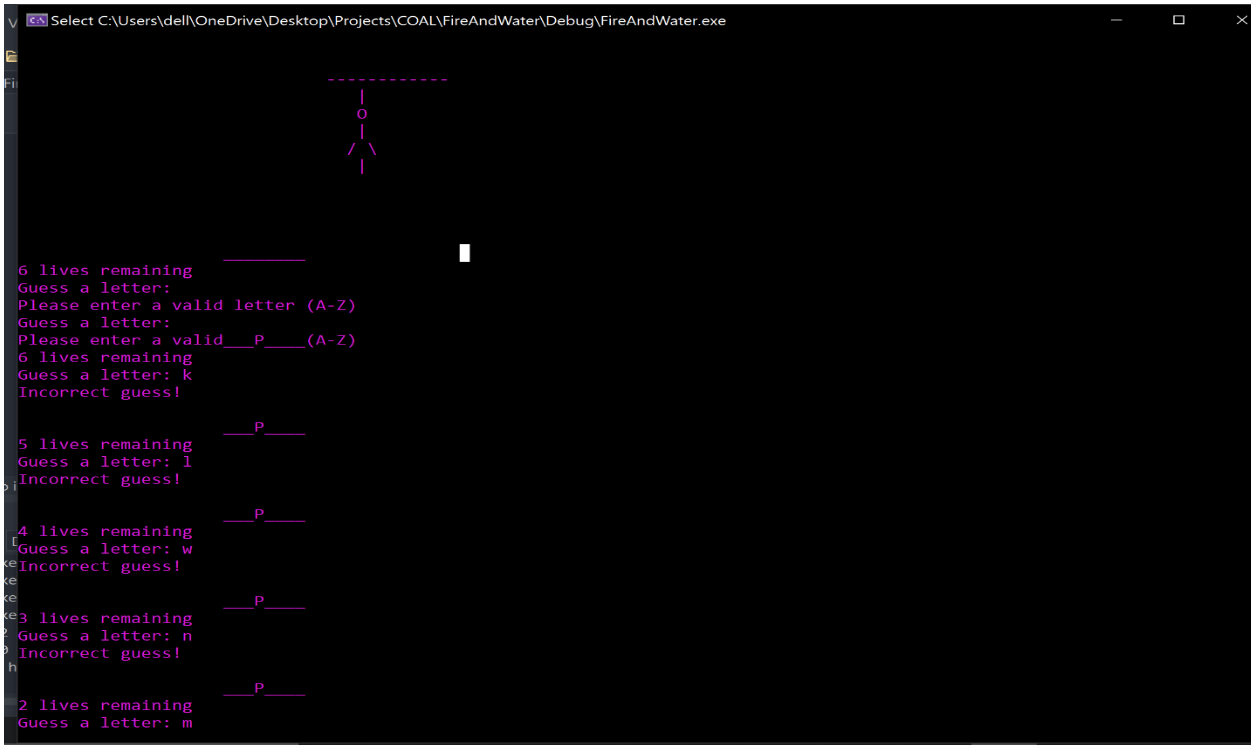
```
C:\Users\dell\OneDrive\Desktop\Projects\COAL\FireAndWater\Debug\FireAndWater.exe

                    CO_P____
6 lives remaining
Guess a letter: u
Correct guess!

                    CO_PU___
6 lives remaining
Guess a letter: t
Correct guess!

                    CO_PUT__
6 lives remaining
Guess a letter: e
Correct guess!

                    CO_PUTE_
6 lives remaining
Guess a letter: r
Correct guess!

                    CO_PUTER
6 lives remaining
Guess a letter: m
Correct guess!

                    COMPUTER
6 lives remaining
Congratulations! You won!
```

6. Game Lost



```
C:\Users\dell\OneDrive\Desktop\Projects\COAL\FireAndWater\Debug\FireAndWater.exe
Incorrect guess!

                    ________
5 lives remaining
Guess a letter: w
Incorrect guess!

                    ________
4 lives remaining
Guess a letter: b
Incorrect guess!

                    ________
3 lives remaining
Guess a letter: x
Incorrect guess!

                    ________
2 lives remaining
Guess a letter: a
Incorrect guess!

                    ________
1 lives remaining
Guess a letter: s
Incorrect guess!

                    ________
0 lives remaining
Game over! The word was: COMPUTER
```

7. Hanging man on losing game:



```
Select C:\Users\dell\OneDrive\Desktop\Projects\COAL\FireAndWater\Debug\FireAndWater.exe

                                ------------
                                   |
                                   O
                                   |
                                  / \
                                   |



                    ________
6 lives remaining
Guess a letter:
Please enter a valid letter (A-Z)
Guess a letter:
Please enter a valid___P____(A-Z)
6 lives remaining
Guess a letter: k
Incorrect guess!

                    ___P____
5 lives remaining
Guess a letter: l
Incorrect guess!

                    ___P____
4 lives remaining
Guess a letter: w
Incorrect guess!

                    ___P____
3 lives remaining
Guess a letter: n
Incorrect guess!

                    ___P____
2 lives remaining
Guess a letter: m
```

Challenges and Solutions:

- **Input Validation**: Ensured only letters are accepted by validating ASCII ranges in GetUserGuess.
- **Modular Design**: Split code into multiple files to improve maintainability, using a header file for prototypes.
- **Graphical Display**: Managed cursor positioning to align the hangman figure correctly across multiple lives.

Appendix:

Header File:

```
; display.inc
DisplayGameState PROTO

MAX_GUESSES EQU 6


displayHangman PROTO, lives:dword

Welcome PROTO
Start PROTO

endgame PROTO,check:dword



main PROTO
```

Main file code:

```
INCLUDE Irvine32.inc
INCLUDE display.inc
.data
targetWord      BYTE "COMPUTER",0       ; The word to guess
wordLength      EQU ($ - targetWord) - 1 ; Length not counting null terminator
displayWord     BYTE wordLength DUP('_'),0
livesLeft       DWORD 6                 ; Number of allowed incorrect guesses
correctGuess    BYTE "Correct guess!",0
incorrectGuess  BYTE "Incorrect guess!",0
prompt          BYTE "Guess a letter: ",0
winMessage      BYTE "Congratulations! You won!",0
loseMessage     BYTE "Game over! The word was: ",0
alreadyGuessed  BYTE "You already guessed that letter!",0
guessedLetters  BYTE 26 DUP(0)          ; Track which letters have been guessed
initiak_col byte 20
initial_row byte 12
slate BYTE "------------",0
playAgain BYTE 0

; Macro to display a string and move to the next line
DisplayString MACRO strAddr
    mov edx, strAddr
    call WriteString
    call Crlf
ENDM

; Macro to set cursor position
SetCursor MACRO row, col
    mov dh, row
    mov dl, col
    call Gotoxy
ENDM

; Macro to reset an array with a given value
ResetArray MACRO arrAddr, arrSize, val
    mov ecx, arrSize
    mov edi, arrAddr
    mov al, val
ResetLoop:
    mov [edi], al
    inc edi
    loop ResetLoop
ENDM

; Macro to decrement lives and display incorrect guess message
HandleIncorrectGuess MACRO
    mov edx, OFFSET incorrectGuess
    call WriteString
    call Crlf
    dec livesLeft
ENDM
.code
main PROC
GameRestart:
    ; Reset game state
    mov livesLeft, 6
    mov initial_row, 12
    mov initiak_col, 20
    mov playAgain, 0
    ; Reset displayWord to underscores
    mov ecx, wordLength
    mov edi, OFFSET displayWord
    mov al, '_'
ResetDisplay:
    mov [edi], al
    inc edi
    loop ResetDisplay
    mov BYTE PTR [edi], 0  ; Null terminator
    ; Reset guessedLetters
    mov ecx, 26
    mov edi, OFFSET guessedLetters
    mov al, 0
ResetGuessed:
    mov [edi], al
    inc edi
    loop ResetGuessed

    call Clrscr
    call Welcome
    call Clrscr
    call Start
    call Clrscr
    mov dh, 2
    mov dl, 30
    call Gotoxy
    mov edx, OFFSET slate
    call WriteString

GameLoop:
    mov dh, initial_row
    mov dl, initiak_col
    call Gotoxy
    ; Display current game state
    mov edx, OFFSET displayWord
    call WriteString
    call Crlf
    ; Display remaining lives
    mov eax, livesLeft
    call WriteDec
    mov edx, OFFSET livesPrompt
    call WriteString
    call Crlf

    ; Check win/lose conditions
    call CheckWinCondition
    cmp eax, 1
    je GameWon

    cmp livesLeft, 0
    je GameLost

    ; Get user input
    call GetUserGuess
    call UpdateGameState

    INVOKE displayHangman, livesLeft
    add initial_row, 5
    jmp GameLoop

GameWon:
    mov edx, OFFSET winMessage
    call WriteString
    call Crlf
    mov eax, 1
    INVOKE endgame, eax
    cmp playAgain, 1
    je GameRestart
    jmp ExitGame

GameLost:
    mov edx, OFFSET loseMessage
    call WriteString
    mov edx, OFFSET targetWord
    call WriteString
    mov eax, 0
    INVOKE endgame, eax
    cmp playAgain, 1
    je GameRestart
    call Crlf

ExitGame:
    exit
main ENDP
;---------------------------------------------------------
UpdateGameState PROC
    ; AL contains the guessed letter (uppercase)
    mov bl, al              ; Save the guessed letter in BL

    ; Check if letter was already guessed
    movzx eax, bl
    sub eax, 'A'            ; Convert to index (0-25)
    cmp guessedLetters[eax], 1
    je AlreadyGuessed2

    ; Mark letter as guessed
    mov guessedLetters[eax], 1

    ; Check if letter is in the word
    mov ecx, wordLength
    mov edi, 0
    mov edx, 0              ; Flag for found letter (0 = not found)

SearchWord:
    mov al, targetWord[edi]  ; Get current letter from target word
    cmp al, bl              ; Compare with guessed letter
    jne NotMatch

    ; Found a match - update displayWord
    mov displayWord[edi], al
    mov edx, 1              ; Set flag to found

NotMatch:
    inc edi
    loop SearchWord

    ; Update lives if letter not found
    cmp edx, 1
    je LetterFound

    ; Letter not found
    mov edx, OFFSET incorrectGuess
    call WriteString
    call Crlf
    dec livesLeft
    jmp UpdateDone

LetterFound:
    mov edx, OFFSET correctGuess
    call WriteString
    call Crlf

UpdateDone:
    ret

AlreadyGuessed2:
    mov edx, OFFSET alreadyGuessed
    call WriteString
    call Crlf
    ret
UpdateGameState ENDP

;----------------------------------------------------------
GetUserGuess PROC
    ; Prompt for and get a letter from the user
    ; Returns uppercase letter in AL
InputLoop:
    mov edx, OFFSET prompt
    call WriteString

    call ReadChar           ; Read character (returns in AL)
    call WriteChar          ; Echo the character
    call Crlf

    ; Convert to uppercase
    cmp al, 'a'
    jl NotLower
    cmp al, 'z'
    jg NotLower
    sub al, 32              ; Convert lowercase to uppercase

NotLower:
    ; Validate input is a letter
    cmp al, 'A'
    jl InvalidInput
    cmp al, 'Z'
    jg InvalidInput

    ret

InvalidInput:
    mov edx, OFFSET invalidInputMsg
    call WriteString
    call Crlf
    jmp InputLoop
GetUserGuess ENDP

;----------------------------------------------------------
CheckWinCondition PROC
    ; Checks if all letters have been guessed
    ; Returns 1 in EAX if game is won, 0 otherwise
    mov ecx, wordLength
    mov esi, 0

CheckLoop:
    mov al, displayWord[esi]
    cmp al, '_'
    je NotWon

    inc esi
    loop CheckLoop

    ; All letters guessed
    mov eax, 1
    ret

NotWon:
    mov eax, 0
    ret
CheckWinCondition ENDP

; Additional data declarations
livesPrompt     BYTE " lives remaining",0
invalidInputMsg BYTE "Please enter a valid letter (A-Z)",0



END main
```

Display:

```
.386
.model flat, stdcall
.stack 4096
INCLUDE Irvine32.inc

.data
    slate BYTE "|",0
    face BYTE "O",0
    neck BYTE "|",0
    body BYTE "|",0
    arms BYTE "/ \",0
    legs BYTE "/ \",0

.code
displayHangman PROC, lives:DWORD
    mov eax, lives
    cmp eax, 5
    je slate1
    cmp eax, 4
    je face1
    cmp eax, 3
    je neck1
    cmp eax, 2
    je body1
    cmp eax, 1
    je arms1
    cmp eax, 0
    je legs1

slate1:
    mov dh,3
    mov dl,33
    call Gotoxy
    mov edx,offset slate
    call WriteString
    ret

face1:
    mov dh, 4
    mov dl, 33
    call Gotoxy
    mov edx, OFFSET face
    call WriteString
    ret

neck1:
    mov dh, 5
    mov dl, 33
    call Gotoxy
    mov edx, OFFSET neck
    call WriteString
    ret

arms1:
    mov dh, 6
    mov dl, 32
    call Gotoxy
    mov edx, OFFSET arms
    call WriteString
    ret

body1:
    mov dh, 7
    mov dl, 33
    call Gotoxy
    mov edx, OFFSET body
    call WriteString
    ret

legs1:
    mov dh, 8
    mov dl, 32
    call Gotoxy
    mov edx, OFFSET legs
    call WriteString
    ret
displayHangman ENDP


END
```