External Project Report on Computer Organization and Architecture (EET 2211)

Balloon Blaster: A Game in 8086 Assembly

Submitted by

Name 01 : Kunal Routray **Regd no: 2341018202**
Name 02 : Samikshya Sanskruti swain **Regd no:2341019634**
Name 03 : Priti Rani Maity **Regd no:2341013065**
Name 04 : Prabeen kumar Pradhan **Regd no :2341016346**

B.TECH CSE (AI & ML)4th SEMESTER(23412C3)

INSTITUTE OF TECHNICAL EDUCATION AND RESEARCH
(FACULTY OF ENGINEERING)

SIKSHA 'O' ANUSANDHAN (DEEMED TO BE UNIVERSITY), BHUBANESWAR, ODISHA

Declaration

We, the undersigned students of B. Tech. of **CSE (AI & ML)** Department hereby declare that we own the full responsibility for the information, results etc. provided in this PROJECT titled "**(Design any 8086-based game system.)**" submitted to **Siksha 'O' Anusandhan Deemed to be University, Bhubaneswar** for the partial fulfillment of the subject **Computer Organization and Architecture (EET 2211)**. We have taken care in all respect to honor the intellectual property right and have acknowledged the contribution of others for using them in academic purpose and further declare that in case of any violation of intellectual property right or copyright we, as the candidate(s), will be fully responsible for the same.

Kunal Routray

Registration No.: 2341018202

Samikshya Sanskruti Swain

Registration No.: 2341019634

Priti Rani Maity

Registration No.: 2341013065

Prabeen Kumar Pradhan

Registration No.:2341016346

DATE:

PLACE:

Abstract

This project presents the development of a simple yet engaging **Balloon Shooting Game** using **8086 Assembly Language** on the **emu8086 emulator**. The game involves player-controlled movement and shooting mechanics, where balloons descend and the player must hit them with arrows.

The game logic is implemented using low-level programming constructs, including **direct memory access**, **keyboard interrupt handling**, and **text-mode graphics rendering** through **video memory (0B800h)**. It demonstrates real-time interaction, dynamic object rendering, and efficient control flow.

Through this project, we explore the fundamental aspects of system-level programming, showcasing how interactive applications can be built from the ground up in Assembly. This not only strengthens understanding of computer architecture and hardware interaction but also highlights the potential of minimalistic game design using limited computing resources.

- Real-time **player movement** and **arrow shooting**
- **Balloon collision detection** and **score tracking**
- Direct use of **video memory (0B800h)** for text graphics
- **Keyboard input handling** with game logic control

Contents

1. Introduction

This project presents a simple **Balloon Shooting Game** developed using **8086 Assembly Language** and designed to run in **text mode on DOS (emu8086)**. The game simulates a vertical shooter where the player controls a character that moves up and down using the arrow keys and shoots arrows using the spacebar to pop moving balloons.

The game showcases fundamental principles of low-level programming including:

- **Direct memory access to video memory** (using segment B800h) for rendering characters on the screen.
- **Keyboard interrupt handling (INT 16h)** for capturing user input.
- **Text-based animation** through manipulation of screen memory positions.
- **Game logic** involving score tracking, collision detection between arrows and balloons, and managing game states such as start, play, and game over.

The core features of the game include:

- **Player Movement** (Up and Down)
- **Arrow Firing Mechanism**
- **Balloon Popping & Collision Detection**
- **Score Display (Hits & Misses)**
- **Game Over and Restart Options**
- **User-Friendly Game Menu**

2. Problem Statement

"How to create a functional and interactive balloon shooting game using 8086 Assembly Language that can respond to real-time user input, display animated gameplay in text mode, and provide a basic scoring system — all within the constraints of DOS architecture and limited system resources?"

Key Problems Explored:

- **Real-Time Input Handling:** Capturing and processing keyboard inputs (arrow keys and spacebar) using BIOS interrupts without halting program flow.
- **Game State Management:** Managing multiple game states such as Start Menu, Active Gameplay, and Game Over, entirely through memory-based control structures.
- **Screen Rendering in Text Mode:** Creating visual feedback and animations (player, arrows, balloons) using only text mode (video memory segment B800h) instead of graphical libraries.
- **Score Tracking and Display:** Implementing logic to track and display the number of hits and misses in a readable format during gameplay.
- **Memory Constraints and Optimization:** Operating within the limited memory model of the 8086 processor while maintaining game performance and responsiveness.
- **Restart Mechanism:** Allowing players to restart the game without exiting, requiring a reset of game state and memory variables.

3. Methodology

1.Planning & Design

- Designed game logic flow: player movement, shooting, collisions, scoring.
- Defined game states: Start Screen → Game Loop → Game Over → Restart.
- Chose text-mode rendering to match 8086 and DOS limitations.

---

2. Memory & Segment Setup

- Used .model large for better segment organization.
- Allocated .data segment for:
    - Game variables (player position, arrow status, balloon status).
    - UI strings (Game Over, Start Menu, Score display).
- Initialized video memory segment (B800h) for screen output.

---

3. User Input Handling

- Used **BIOS interrupt INT 16h** for real-time keyboard input.
- Mapped:
    - Arrow Up → Move Player Up
    - Arrow Down → Move Player Down
    - Spacebar → Shoot Arrow
    - Enter → Start/Restart Game

4.Game Logic Implementation

- **Main Loop**: Continuously updated screen based on:
    - Player direction
    - Arrow and balloon movement
    - Collision detection
- **Collision Detection**
    - Incremented hits or misses accordingly

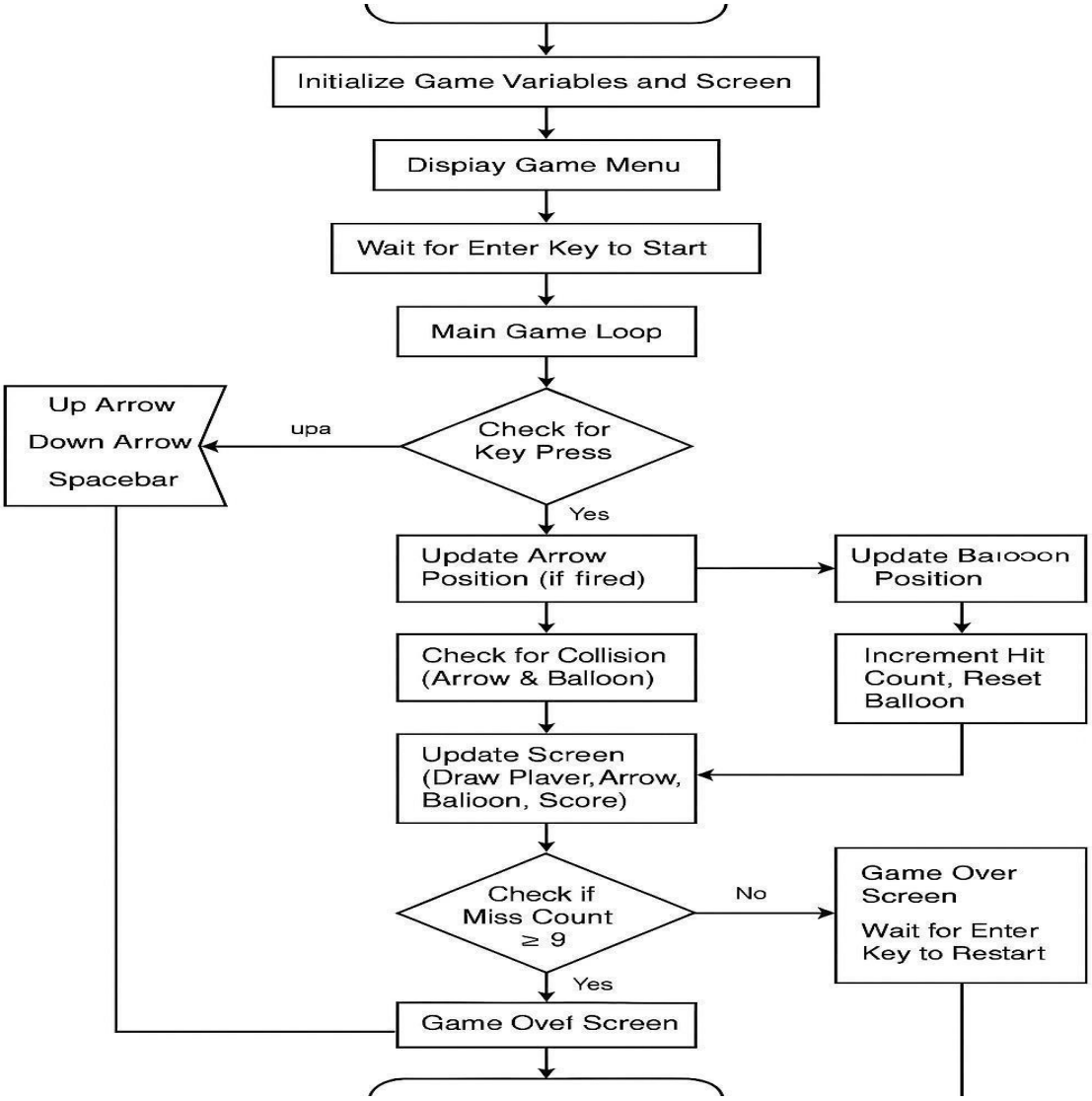
6.Score **Display**

- Created a score buffer state to store and update text.
- Used INT 21h to print score dynamically during the game.

7.Testing & Debugging

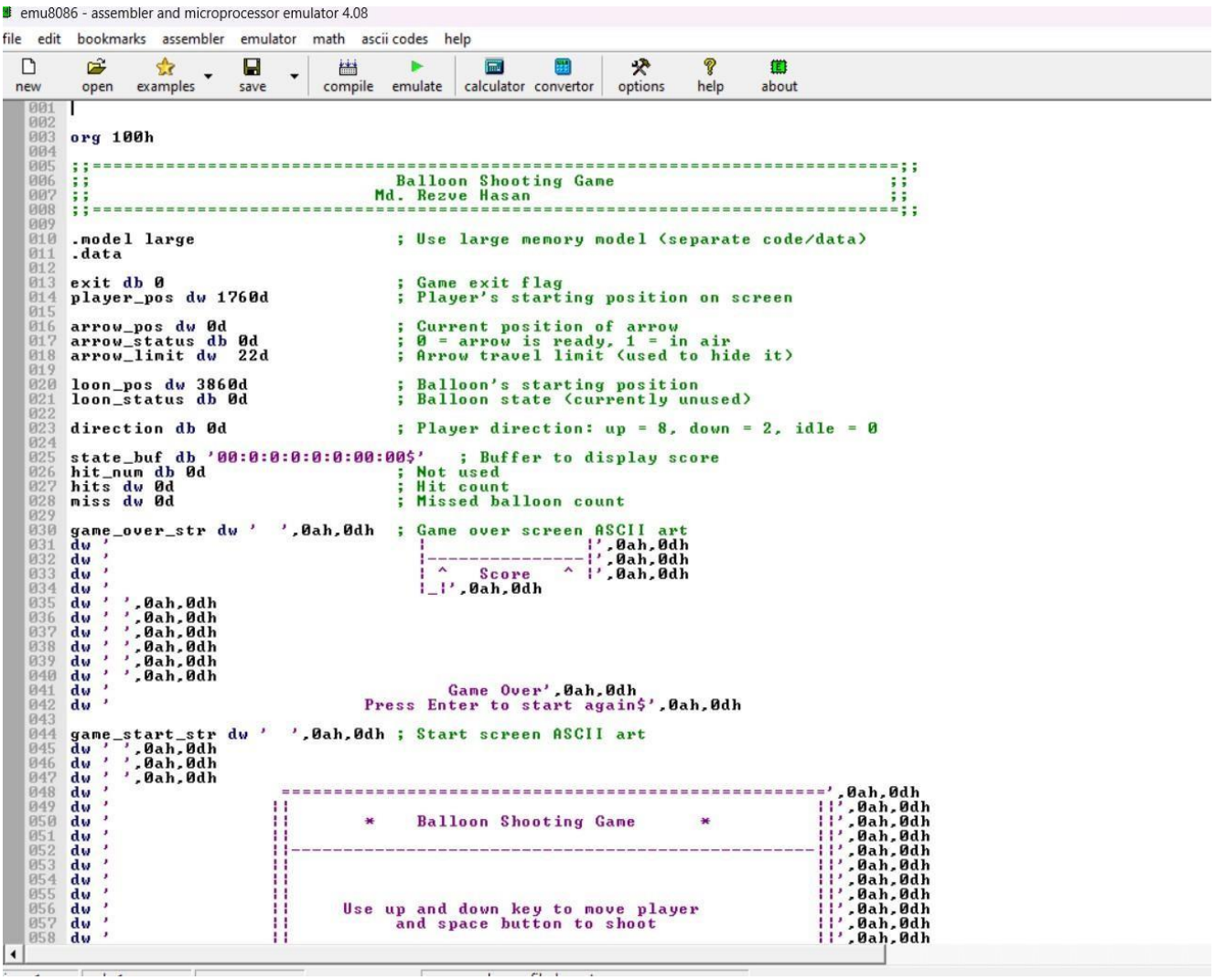
- Debugged movement, rendering, and scoring using key triggers.
- Added test functions like left Key to simulate hits/misses.
- Validated restart logic and game flow after Game Over.

FLOWCHART



4. Implementation

- Developed in 8086 Assembly using DOS interrupts and direct VGA text memory access.
- Player moves up/down with arrow keys; spacebar fires an arrow.
- Arrow moves horizontally, balloon moves left; collision increases score.
- Miss count tracked; game ends after 9 misses with a game over screen.
- Score and game state displayed dynamically on screen.
- Simple beep sound on balloon hit.
- Clear, modular code with procedures for score display and screen clearing.



edit: E:\emu8086\MySource\ballon blaster with description.asm

file edit bookmarks assembler emulator math ascii codes help

new open examples save compile emulate calculator convertor options help about

```
dw '                !!                                                  !!',0ah,0dh
dw '                ==================================================',0ah,0dh
dw '$',0ah,0dh

.code
main proc
    mov ax,@data                ; Initialize data segment
    mov ds,ax

    mov ax, 0B800h              ; Video memory segment for text mode
    mov es,ax

    jmp game_menu               ; Jump to game menu screen

main_loop:                      ; Main game loop
    mov ah,1h
    int 16h                     ; Check if a key is pressed
    jnz key_pressed             ; If yes, handle it
    jmp inside_loop             ; If not, continue game logic

inside_loop:
    cmp miss,9                  ; If 9 balloons missed
    jge game_over               ; End game if miss limit reached

    mov dx,arrow_pos            ; Check collision between arrow and balloon
    cmp dx, loon_pos
    je hit                      ; If collision, register hit

    cmp direction,8d
    je player_up                ; Move player up
    cmp direction,2d
    je player_down              ; Move player down

    mov dx,arrow_limit
    cmp arrow_pos, dx
    jge hide_arrow              ; Hide arrow if past limit

    cmp loon_pos, 0d
    jle miss_loon               ; If balloon left screen, it's a miss
    jne render_loon             ; Otherwise, update balloon position

hit:
    mov ah,2                    ; Play a sound (simulate)
    mov dx, 7d
    int 21h

    inc hits                    ; Increment hit counter

    lea bx,state_buf
    call show_score             ; Update score on screen
    lea dx,state_buf
    mov ah,09h
    int 21h

    mov ah,2                    ; Print newline
    mov dl, 0dh
    int 21h
```

edit: E:\emu8086\MySource\ballon blaster with description.asm

file edit bookmarks assembler emulator math ascii codes help

new open examples save compile emulate calculator convertor options help about

```
    jmp fire_loon               ; Create new balloon

render_loon:
    mov cl, ' '                 ; Erase old balloon
    mov ch, 1111b
    mov bx,loon_pos
    mov es:[bx], cx

    sub loon_pos,160d           ; Move balloon upward
    mov cl, 15d                 ; New balloon character
    mov ch, 1101b
    mov bx,loon_pos
    mov es:[bx], cx

    cmp arrow_status,1d
    je render_arrow             ; If arrow in air, update it
    jne inside_loop2            ; Otherwise, skip

render_arrow:
    mov cl, ' '                 ; Erase old arrow
    mov ch, 1111b
    mov bx,arrow_pos
    mov es:[bx], cx

    add arrow_pos,4d            ; Move arrow upward
    mov cl, 26d                 ; Arrow display character
    mov ch, 1001b
    mov bx,arrow_pos
    mov es:[bx], cx

inside_loop2:
    mov cl, 125d                ; Player character
    mov ch, 1100b
    mov bx,player_pos
    mov es:[bx], cx

    cmp exit,0
    je main_loop                ; Continue loop
    jmp exit_game               ; Exit game

player_up:
    mov cl, ' '                 ; Erase old player
    mov ch, 1111b
    mov bx,player_pos
    mov es:[bx], cx

    sub player_pos, 160d        ; Move player up
    mov direction, 0
    jmp inside_loop2

player_down:
    mov cl, ' '
    mov ch, 1111b
    mov bx,player_pos
    mov es:[bx], cx

    add player_pos,160d         ; Move player down
```

line: 3 col: 10 drag a file here to open

edit: E:\emu8086\MySource\ballon blaster with description.asm

file edit bookmarks assembler emulator math ascii codes help

new open examples save compile emulate calculator convertor options help about

```
    mov bx,player_pos
    mov es:[bx], cx

    add player_pos,160d       ; Move player down
    mov direction, 0
    jmp inside_loop2

key_pressed:
    mov ah,0
    int 16h                   ; Read the pressed key

    cmp ah,48h
    je upKey                  ; Up arrow key
    cmp ah, 50h
    je downKey                ; Down arrow key

    cmp ah,39h
    je spaceKey               ; Spacebar

    cmp ah,4Bh
    je leftKey                ; Debug (simulate miss)

    jmp inside_loop

leftKey:
    inc miss                  ; Increase miss count
    lea bx,state_buf
    call show_score
    lea dx,state_buf
    mov ah,09h
    int 21h

    mov ah,2
    mov dl, 0dh
    int 21h
    jmp inside_loop

upKey:
    mov direction, 8d         ; Set direction to up
    jmp inside_loop

downKey:
    mov direction, 2d         ; Set direction to down
    jmp inside_loop

spaceKey:
    cmp arrow_status,0
    je  fire_arrow            ; Only shoot if arrow is not already in air
    jmp inside_loop

fire_arrow:
    mov dx, player_pos
    mov arrow_pos, dx         ; Set arrow start from player position
    mov dx,player_pos
    mov arrow_limit, dx
    add arrow_limit, 22d      ; Define arrow's flight range
    mov arrow_status, 1d      ; Arrow is now in air
    jmp inside_loop
```

edit: E:\emu8086\MySource\ballon blaster with description.asm

file edit bookmarks assembler emulator math ascii codes help

new open examples save compile emulate calculator convertor options help about

```
    jmp inside_loop

miss_loon:
    add miss,1                ; Increase miss count
    lea bx,state_buf
    call show_score
    lea dx,state_buf
    mov ah,09h
    int 21h
    mov ah,2
    mov dl, 0dh
    int 21h
    jmp fire_loon             ; Generate new balloon

fire_loon:
    mov loon_status, 1d
    mov loon_pos, 3860d
    jmp render_loon

hide_arrow:
    mov arrow_status, 0
    mov cl, ' '
    mov ch, 1111b
    mov bx,arrow_pos
    mov es:[bx], cx

    cmp loon_pos, 0d
    jle miss_loon
    jne render_loon

    jmp inside_loop2

game_over:
    mov ah,09h
    mov dx, offset game_over_str
    int 21h

    mov cl, ' '
    mov ch, 1111b
    mov bx,arrow_pos          ; Erase last arrow

    mov cl, ' '
    mov ch, 1111b
    mov bx,player_pos         ; Erase player

    ; Reset game state
    mov miss, 0d
    mov hits,0d
    mov player_pos, 1760d
    mov arrow_pos, 0d
    mov arrow_status, 0d
    mov arrow_limit, 22d
    mov loon_pos, 3860d
    mov loon_status, 0d
    mov direction, 0d
```

edit: E:\emu8086\MySource\ballon blaster with description.asm

file edit bookmarks assembler emulator math ascii codes help

new open examples save compile emulate calculator convertor options help about

```
283     mov direction, 0d
284
285 input:
286     mov ah,1
287     int 21h
288     cmp al,13d                  ; Wait for Enter key
289     jne input
290     call clear_screen
291     jmp main_loop
292
293 game_menu:
294     mov ah,09h
295     mov dh,0
296     mov dx, offset game_start_str
297     int 21h
298
299 input2:
300     mov ah,1
301     int 21h
302     cmp al,13d                  ; Wait for Enter key
303     jne input2
304     call clear_screen
305
306     lea bx,state_buf
307     call show_score
308     lea dx,state_buf
309     mov ah,09h
310     int 21h
311
312     mov ah,2
313     mov dl, 0dh
314     int 21h
315
316     jmp main_loop
317
318 exit_game:
319     mov exit,10d                ; Set exit flag
320
321 main endp
322
323 ;;-------------------------------------;;
```

edit: E:\emu8086\MySource\ballon blaster with description.asm

file edit bookmarks assembler emulator math ascii codes help

new open examples save compile emulate calculator convertor options help about

```
337     mov [bx+7], 's'
338     mov [bx+8], ':'
339     mov [bx+9], dx
340
341     mov dx, miss
342     add dx,48d
343     mov [bx+10], ' '
344     mov [bx+11], 'M'
345     mov [bx+12], 'i'
346     mov [bx+13], 's'
347     mov [bx+14], 's'
348     mov [bx+15], ':'
349     mov [bx+16], dx
350     ret
351 show_score endp
352
353 ;;-----------------------------------;;
354 ;; Clear screen by switching mode    ;;
355 ;;-----------------------------------;;
356 clear_screen proc near
357     mov ah,0
358     mov al,3
359     int 10h                     ; Switch to text mode to clear screen
360     ret
361 clear_screen endp
362
363 end main
364
365
366 ret
367
368
369
370
371
```

5. Results & Interpretation

🎮 Result & Interpretation

☑ Game Outcome:

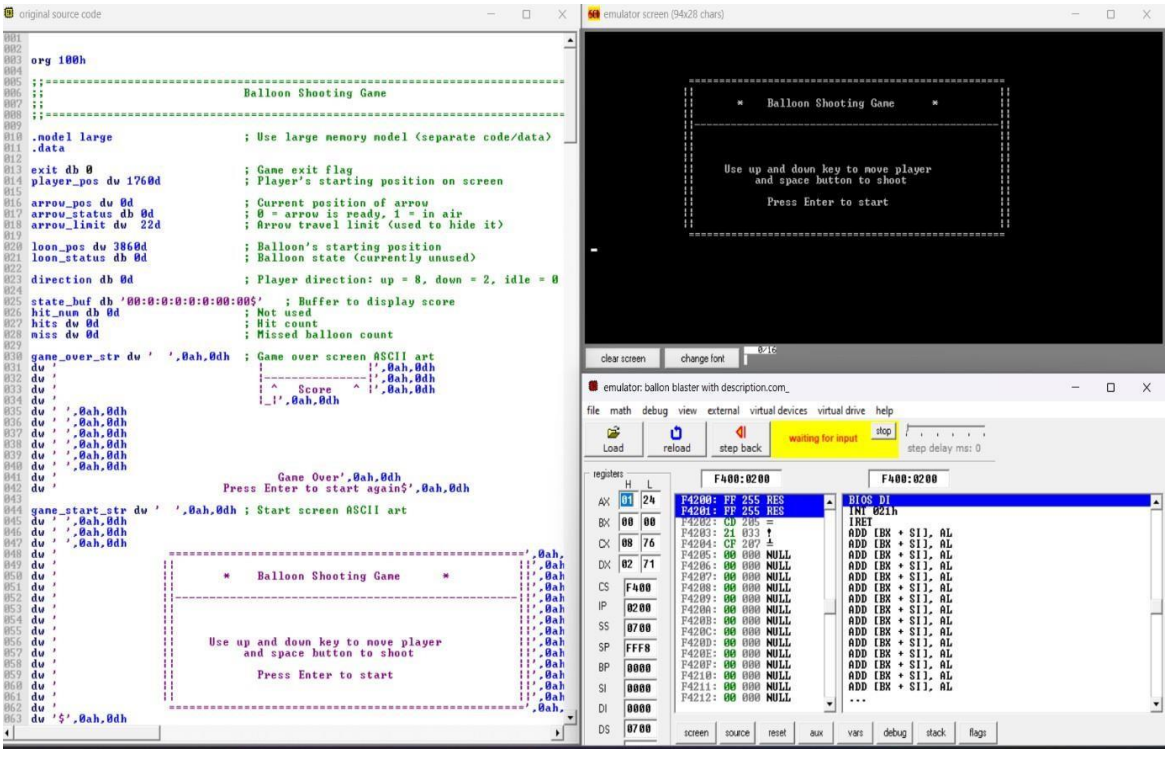
- Player moves using ↑ / ↓, shoots with **Spacebar**
- **Real-time hit/miss score tracking**
- Game ends after **9 missed balloons**
- **"Game Over" & Restart option (Enter key)**

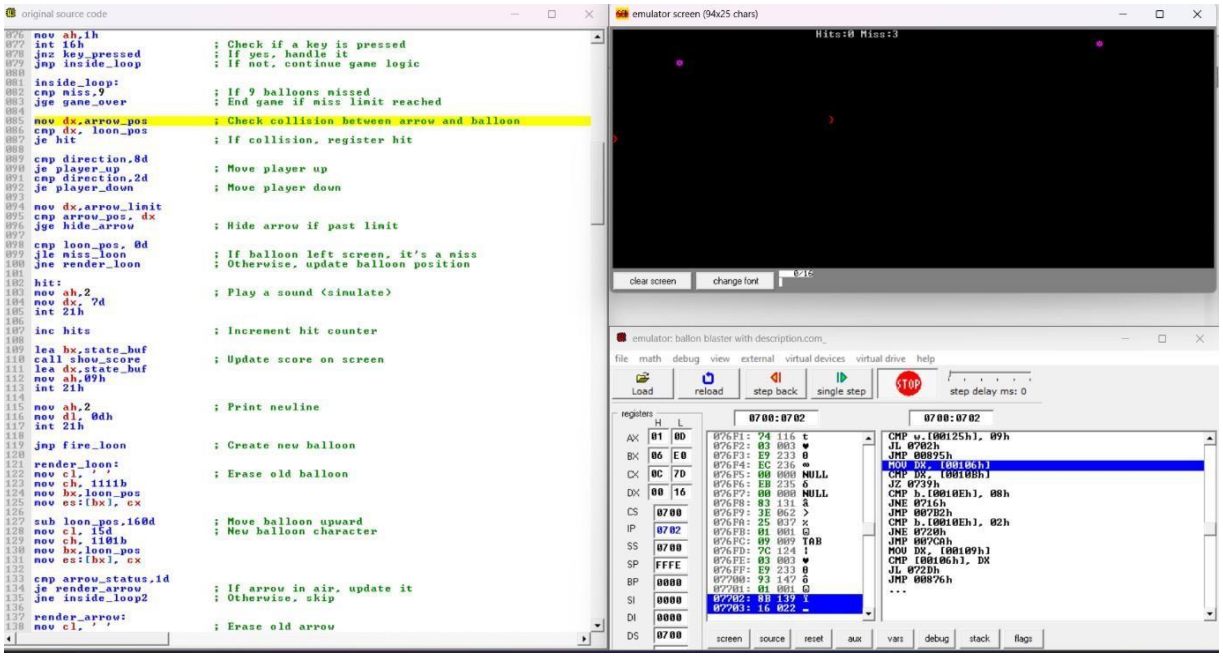
🔍 Key Highlights:

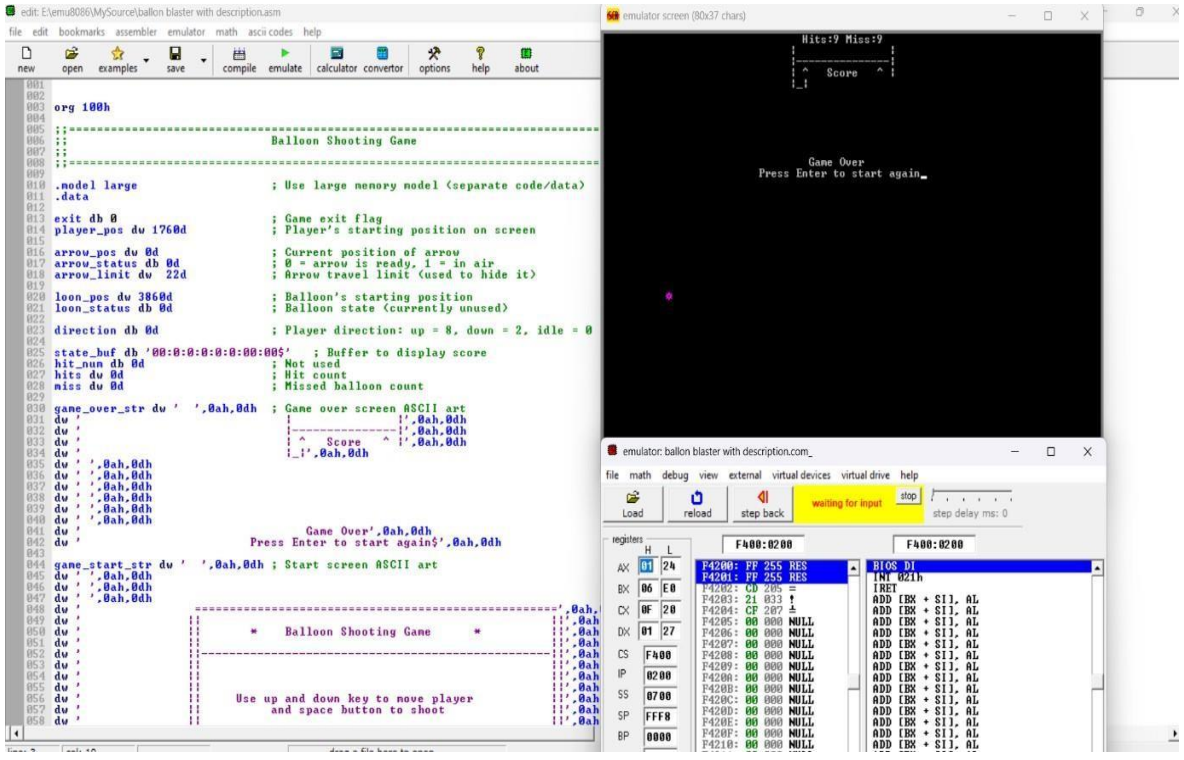
- Developed in **8086 Assembly** using **emu8086**
- Uses **video memory (0B800h)** for screen rendering
- Implements **real-time user interaction**
- Demonstrates **keyboard handling**, **graphics**, and **game logic** at a low level

💡 Interpretation:

"Even with limited resources, low-level programming can build real-time, interactive applications—proving the power of Assembly!"




original source code
mov ah,1h
int 16h ; Check if a key is pressed
jnz key_pressed ; If yes, handle it
jmp inside_loop ; If not, continue game logic
inside_loop:
cmp miss,9 ; If 9 balloons missed
jge game_over ; End game if miss limit reached
mov dx,arrow_pos ; Check collision between arrow and balloon
cmp dx, loon_pos
je hit ; If collision, register hit
cmp direction,8d
je player_up ; Move player up
cmp direction,2d
je player_down ; Move player down
mov dx,arrow_limit
cmp arrow_pos, dx
jge hide_arrow ; Hide arrow if past limit
cmp loon_pos, 0d
jle miss_loon ; If balloon left screen, it's a miss
jne render_loon ; Otherwise, update balloon position
hit:
mov ah,2 ; Play a sound (simulate)
mov dx, 7d
int 21h
inc hits ; Increment hit counter
lea bx,state_buf
call show_score ; Update score on screen
lea dx,state_buf
mov ah,09h
int 21h
mov ah,2 ; Print newline
mov dl, 0dh
int 21h
jmp fire_loon ; Create new balloon
render_loon:
mov cl, ' '
mov ch, 1111b ; Erase old balloon
mov bx,loon_pos
mov es:[bx], cx
sub loon_pos,160d ; Move balloon upward
mov cl, 15d ; New balloon character
mov ch, 1101b
mov bx,loon_pos
mov es:[bx], cx
cmp arrow_status,1d
je render_arrow ; If arrow in air, update it
jne inside_loop2 ; Otherwise, skip
render_arrow:
mov cl, ' ' ; Erase old arrow
emulator screen (94x25 chars)
Hits:0 Miss:3
clear screen
change font
emulator: ballon blaster with description.com_
file math debug view external virtual devices virtual drive help
Load
reload
step back
single step
STOP
step delay ms: 0
registers
0700:0702
0700:0702
AX 01 0D
BX 06 E0
CX 0C 7D
DX 00 16
CS 0700
IP 0702
SS 0700
SP FFFE
BP 0000
SI 0000
DI 0000
DS 0700
screen
source
reset
aux
vars
debug
stack
flags


edit: E:\emu8086\MySource\ballon blaster with description.asm
file edit bookmarks assembler emulator math ascii codes help
new
open
examples
save
compile
emulate
calculator
convertor
options
help
about
org 100h
;; Balloon Shooting Game
.model large ; Use large memory model (separate code/data)
.data
exit db 0 ; Game exit flag
player_pos dw 1760d ; Player's starting position on screen
arrow_pos dw 0d ; Current position of arrow
arrow_status db 0d ; 0 = arrow is ready, 1 = in air
arrow_limit dw 22d ; Arrow travel limit (used to hide it)
loon_pos dw 3860d ; Balloon's starting position
loon_status db 0d ; Balloon state (currently unused)
direction db 0d ; Player direction: up = 8, down = 2, idle = 0
state_buf db '00:0:0:0:0:0:00:00$' ; Buffer to display score
hit_nun db 0d ; Not used
hits dw 0d ; Hit count
miss dw 0d ; Missed balloon count
game_over_str dw ' ',0ah,0dh ; Game over screen ASCII art
Game Over',0ah,0dh
Press Enter to start again$',0ah,0dh
game_start_str dw ' ',0ah,0dh ; Start screen ASCII art
Balloon Shooting Game
Use up and down key to move player
and space button to shoot
emulator screen (80x37 chars)
Hits:9 Miss:9
Score
Game Over
Press Enter to start again_
emulator: ballon blaster with description.com_
file math debug view external virtual devices virtual drive help
Load
reload
step back
waiting for input
stop
step delay ms: 0
registers
F400:0200
F400:0200
AX 01 24
BX 06 E0
CX 0F 20
DX 01 27
CS F400
IP 0200
SS 0700
SP FFF8
BP 0000
F4200: FF 255 RES
F4201: FF 255 RES
F4202: CD 205 =
F4203: 21 033 !
F4204: CF 207 ±
F4205: 00 000 NULL
F4206: 00 000 NULL
F4207: 00 000 NULL
F4208: 00 000 NULL
F4209: 00 000 NULL
F420A: 00 000 NULL
F420B: 00 000 NULL
F420C: 00 000 NULL
F420D: 00 000 NULL
F420E: 00 000 NULL
F420F: 00 000 NULL
F4210: 00 000 NULL
BIOS DI
INT 021h
IRET
ADD [BX + SI], AL

6. Conclusion

Balloon Shooting Game in 8086 Assembly

📌 **What We Achieved:**

- **Created a fully interactive game using Assembly Language**
- **Controlled player movement and arrow shooting**
- **Implemented real-time score tracking and collision logic**
- **Used direct video memory (0B800h) for graphics**
- **Handled keyboard inputs and game flow effectively**

Final Thoughts:

"This project proves that even with low-level resources, high-level creativity can bring games to life. It's not just a game—it's a lesson in system-level programming."

Skill demonstrated

- **Assembly Language Programming**
- **Memory & Register Manipulation**
- **Game Logic & Flow Control**
- **Hardware Interaction**

Thank You!

References

1. Intel 8086 Microprocessor Programming Manual

- **Publisher: Intel Corporation**
- **Description: The official reference for 8086 architecture, instruction set, registers, memory segmentation, and interrupt handling.**
- **Relevance: Used to understand instruction behavior like INT 10h, INT 21h, and control flow logic.**

---

2. Emu8086 Documentation

- **Source: www.emu8086.com**
- **Description: Official guide and help files for the Emu8086 microprocessor emulator.**
- **Relevance: Emu8086 was the main platform used to write, compile, and test the game. The documentation helped with memory addressing (like B800h for video), keyboard input, and interrupt simulation.**

---

3. Kip R. Irvine – Assembly Language for x86 Processors

- **Publisher: Pearson Education**
- **Description: A well-known academic textbook that covers x86 Assembly concepts, string processing, I/O operations, and real-mode programming in DOS.**
- **Relevance: Provided foundational knowledge of structured programming in Assembly, especially helpful in writing procedures (show_score, clear_screen) and managing memory.**

Appendices

Appendices

Appendix A: Project Structure

- **Main Modules:**
  - **main_loop: Core game loop handling rendering and logic.**
  - **key_pressed: Keyboard input handler.**
  - **render_loon & render_arrow: Visual rendering for balloons and arrows.**
  - **show_score: Displays score on-screen.**
  - **game_menu & game_over: UI screens for game state.**
- **Procedures Used:**
  - **show_score, clear_screen**

Appendix B: Tools & Environment

- **Emulator Used: Emu8086**
- **Processor Mode: Real Mode (16-bit)**
- **Display Memory Segment: 0B800h (Text Mode Video Memory)**
- **Interrupts Used:**
  - **INT 10h – Screen control**
  - **INT 21h – DOS services (printing, input)**
  - **INT 16h – Keyboard input**

Appendix C: Gameplay Controls

Key	Function
↑ (Up)	Move player up
↓ (Down)	Move player down
Space	Shoot arrow
Enter	Start game / Restart

Appendix D: Scoring Logic

- **Hits: Incremented when arrow hits balloon**

- **Misses: Incremented when balloon escapes screen**
- **Game Over: Triggered when misses reach 9**

Appendix E: Memory Mapped Video Rendering

- **Game uses direct video memory access:**
  - **es:[bx] is used to write character + color byte at a screen position**
  - **Balloon, player, and arrow are drawn using ASCII symbols and color attributes**

Appendix F: Game Flow Summary

1. **Show start menu (waiting for Enter)**
2. **Start main game loop**
3. **Handle movement, arrow firing, collision detection**
4. **Track score and display**
5. **Show Game Over screen and reset on Enter**