

# **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**

<b>Serial No.</b>	<b>Chapter No.</b>	<b>Title of the Chapter</b>	<b>Page No.</b>
1.	1	Introduction	1
2.	2	Problem Statement	2
3.	3	Methodology	3-4
4.	4	Implementation	5-8
5.	5	Results and interpretation	9-10
6.	6	Conclusion	11
7.		References	12
8.		Appendices	13-14

## 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

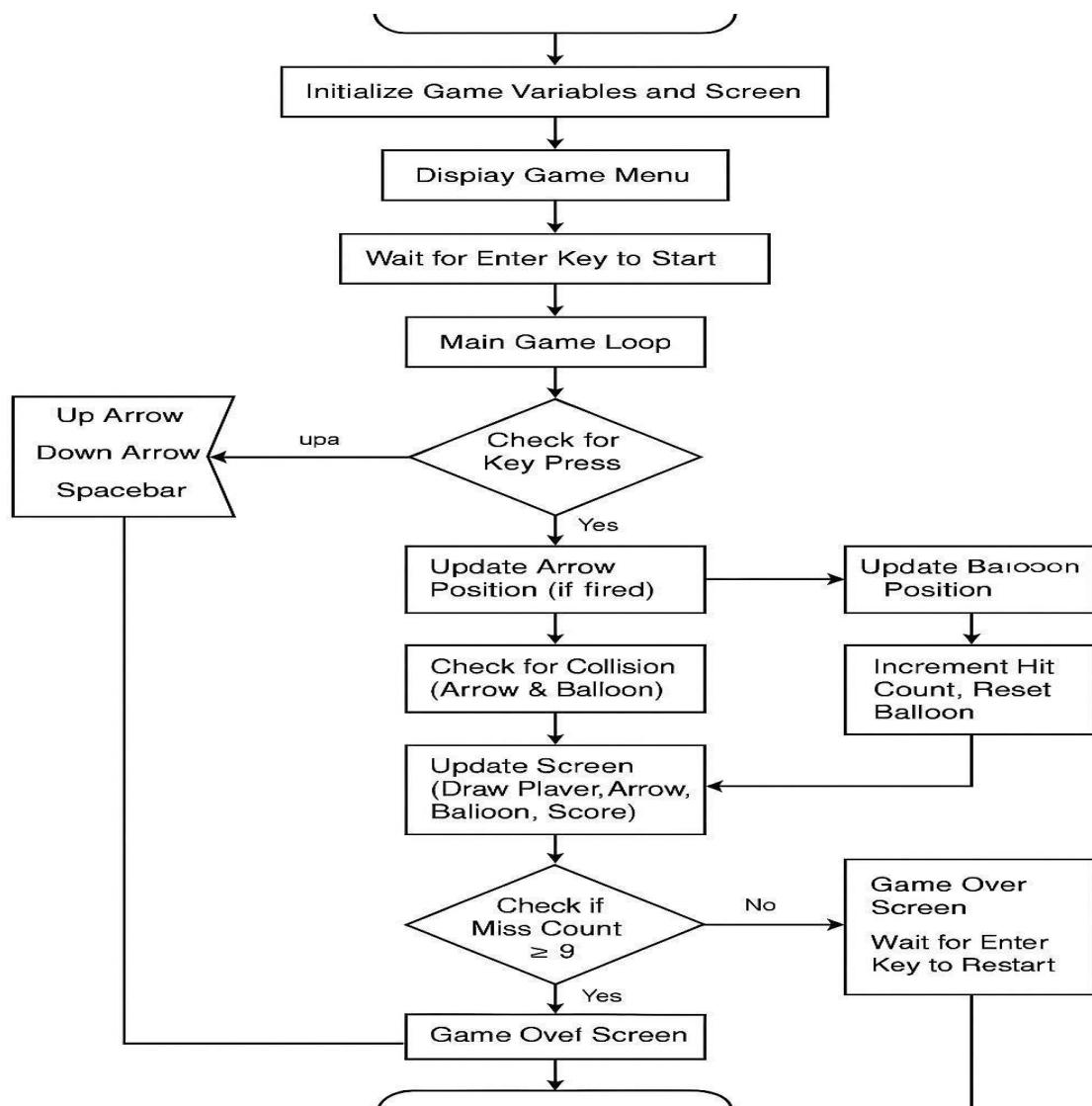
## 6.Score Display

- 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.

The screenshot shows the emu8086 assembler interface with the following assembly code:

```
001 |
002 |
003 org 100h
004
005 ;=====
006 ; Balloon Shooting Game
007 ; Md. Rezve Hasan
008 ;=====
009
010 .model large           ; Use large memory model <separate code/data>
011 .data
012
013 exit db 0              ; Game exit flag
014 player_pos dw 1760d    ; Player's starting position on screen
015
016 arrow_pos dw 0d         ; Current position of arrow
017 arrow_status db 0d       ; 0 = arrow is ready, 1 = in air
018 arrow_limit dw 22d      ; Arrow travel limit (used to hide it)
019
020 loon_pos dw 3860d      ; Balloon's starting position
021 loon_status db 0d        ; Balloon state (currently unused)
022
023 direction db 0d        ; Player direction: up = 8, down = 2, idle = 0
024
025 state_buf db '00:0:0:0:0:0:0:00$' ; Buffer to display score
026 hit_num db 0d            ; Not used
027 hits dw 0d               ; Hit count
028 miss dw 0d               ; Missed balloon count
029
030 game_over_str dw ' ',0ah,0dh ; Game over screen ASCII art
031 dw ' ',0ah,0dh
032 dw ' ',0ah,0dh
033 dw ' ',0ah,0dh
034 dw ' ',0ah,0dh
035 dw ' ',0ah,0dh
036 dw ' ',0ah,0dh
037 dw ' ',0ah,0dh
038 dw ' ',0ah,0dh
039 dw ' ',0ah,0dh
040 dw ' ',0ah,0dh
041 dw ' ',0ah,0dh
042 dw ' ',0ah,0dh
043 dw ' ',0ah,0dh
044 game_start_str dw ' ',0ah,0dh ; Start screen ASCII art
045 dw ' ',0ah,0dh
046 dw ' ',0ah,0dh
047 dw ' ',0ah,0dh
048 dw ' '
049 dw ' '
050 dw ' '
051 dw ' '
052 dw ' '
053 dw ' '
054 dw ' '
055 dw ' '
056 dw ' '
057 dw ' '
058 dw ' '
```

The code defines variables for game states (exit, player position, arrow position, balloon position, direction, etc.) and buffers for displaying the game over and start screens. It also includes a loop for the game logic.

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

```

new open examples save compile emulate calculator convertor options help about
061 dw ',,' ; Initialize data segment
062 dw '$',0ah,0dh
063 dw '$',0ah,0dh
064
065 .code
066 main proc
067     mov ax,0edata
068     mov ds,ax
069
070     mov ax,0B800h ; Video memory segment for text mode
071     mov es,ax
072
073     jmp game_menu ; Jump to game menu screen
074
075 main_loop: ; Main game loop
076     mov ah,1h ; Check if a key is pressed
077     int 16h
078     jnz key_pressed ; If yes, handle it
079     jmp inside_loop ; If not, continue game logic
080
081 inside_loop: ; If 9 balloons missed
082     cmp miss,9
083     jge game_over ; End game if miss limit reached
084
085     mov dx,arrow_pos ; Check collision between arrow and balloon
086     cmp dx,loon_pos
087     je hit ; If collision, register hit
088
089     cmp direction,8d ; Move player up
090     je player_up
091     cmp direction,2d ; Move player down
092     je player_down
093
094     mov dx,arrow_limit ; Hide arrow if past limit
095     cmp arrow_pos,dx
096     jge hide_arrow
097
098     cmp loon_pos,0d ; If balloon left screen, it's a miss
099     jle miss_loon ; Otherwise, update balloon position
100     jne render_loon
101
102 hit: ; Play a sound <simulate>
103     mov ah,2
104     mov dx,7d
105     int 21h
106
107     inc hits ; Increment hit counter
108
109     lea bx,state_buf ; Update score on screen
110     call show_score
111     lea dx,state_buf
112     mov ah,09h
113     int 21h
114
115     mov ah,2 ; Print newline
116     mov dl,0dh
117     int 21h
118

```

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

```

new open examples save compile emulate calculator convertor options help about
118
119     jmp fire_loon ; Create new balloon
120
121 render_loon: ; Erase old balloon
122     mov cl,','
123     mov ch,1111b
124     mov bx,loon_pos
125     mov es:[bx],cx
126
127     sub loon_pos,160d ; Move balloon upward
128     mov cl,15d ; New balloon character
129     mov ch,1101b
130     mov bx,loon_pos
131     mov es:[bx],cx
132
133     cmp arrow_status,id ; If arrow in air, update it
134     je render_arrow
135     jne inside_loop2 ; Otherwise, skip
136
137 render_arrow: ; Erase old arrow
138     mov cl,','
139     mov ch,1111b
140     mov bx,arrow_pos
141     mov es:[bx],cx
142
143     add arrow_pos,4d ; Move arrow upward
144     mov cl,26d ; Arrow display character
145     mov ch,1001b
146     mov bx,arrow_pos
147     mov es:[bx],cx
148
149 inside_loop2: ; Player character
150     mov cl,125d
151     mov ch,1100b
152     mov bx,player_pos
153     mov es:[bx],cx
154
155     cmp exit,0 ; Continue loop
156     je main_loop
157     jmp exit_game ; Exit game
158
159 player_up: ; Erase old player
160     mov cl,','
161     mov ch,1111b
162     mov bx,player_pos
163     mov es:[bx],cx
164
165     sub player_pos,160d ; Move player up
166     mov direction,0
167     jmp inside_loop2
168
169 player_down: ; Add player_pos,160d ; Move player down
170     mov cl,','
171     mov ch,1111b
172     mov bx,player_pos
173     mov es:[bx],cx
174
175     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 converter options help about

172    mov bx,player_pos
173    mov es:[bx], cx
174
175    add player_pos,160d      ; Move player down
176    mov direction, 0
177    jmp inside_loop2
178
179 key_pressed:
180    mov ah,0
181    int 16h                  ; Read the pressed key
182
183    cmp ah,48h
184    je upKey                ; Up arrow key
185    cmp ah,50h
186    je downKey              ; Down arrow key
187
188    cmp ah,39h
189    je spaceKey             ; Spacebar
190
191    cmp ah,4Bh
192    je leftKey              ; Debug <simulate miss>
193
194    jmp inside_loop
195
196 leftKey:
197    inc miss                 ; Increase miss count
198    lea bx,state_buf
199    call show_score
200    lea dx,state_buf
201    mov ah,09h
202    int 21h
203
204    mov ah,2
205    mov dl, 0dh
206    int 21h
207    jmp inside_loop
208
209 upKey:
210    mov direction, 8d         ; Set direction to up
211    jmp inside_loop
212
213 downKey:
214    mov direction, 2d         ; Set direction to down
215    jmp inside_loop
216
217 spaceKey:
218    cmp arrow_status, 0
219    je fire_arrow            ; Only shoot if arrow is not already in air
220    jmp inside_loop
221
222 fire_arrow:
223    mov dx,player_pos
224    mov arrow_pos, dx
225    mov dx,player_pos
226    mov arrow_limit, dx
227    add arrow_limit, 22d
228    mov arrow_status, 1d
229    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 converter options help about

229    jmp inside_loop
230
231 miss_loon:
232    add miss,1                 ; Increase miss count
233    lea bx,state_buf
234    call show_score
235    lea dx,state_buf
236    mov ah,09h
237    int 21h
238    mov ah,2
239    mov dl, 0dh
240    int 21h
241    jmp fire_loon             ; Generate new balloon
242
243 fire_loon:
244    mov loon_status, 1d
245    mov loon_pos, 3860d
246    jmp render_loon
247
248 hide_arrow:
249    mov arrow_status, 0
250    mov cl, '
251    mov ch, 1111b
252    mov bx,arrow_pos
253    mov es:[bx], cx
254
255    cmp loon_pos, 0d
256    jle miss_loon
257    jne render_loon
258
259    jmp inside_loop2
260
261 game_over:
262    mov ah,09h
263    mov dx, offset game_over_str
264    int 21h
265
266    mov cl, '
267    mov ch, 1111b
268    mov bx,arrow_pos           ; Erase last arrow
269
270    mov cl, '
271    mov ch, 1111b
272    mov bx,player_pos          ; Erase player
273
274    ; Reset game state
275    mov miss, 0d
276    mov hits, 0d
277    mov player_pos, 1760d
278    mov arrow_pos, 0d
279    mov arrow_status, 0d
280    mov arrow_limit, 22d
281    mov loon_pos, 3860d
282    mov loon_status, 0d
283    mov direction, 0d
284
285    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

283     mov direction, 0d
284
285 input:
286     mov ah,1
287     int 21h
288     cmp al,13d
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
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
320             ; Set exit flag
321
322 main endp
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     nov [bx+7], 's'
338     nov [bx+8], ':'
339     nov [bx+9], dx
340
341     nov dx, miss
342     add dx,48d
343     nov [bx+10], ' '
344     nov [bx+11], 'M'
345     nov [bx+12], 'i'
346     nov [bx+13], 's'
347     nov [bx+14], ' '
348     nov [bx+15], ' '
349     nov [bx+16], dx
350     ret
351 show_score endp
352
353 ;-----;;
354 ; Clear screen by switching mode ;;
355
356 clear_screen proc near
357     nov ah,0
358     nov al,3
359     int 10h
360             ; Switch to text mode to clear screen
361     ret
362 clear_screen endp
363
364 end main
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!”

The screenshot shows the emu8086 debugger interface. On the left, the 'original source code' window displays the assembly code for the Balloon Shooting Game. The code includes declarations for memory models, variables like player\_pos, arrow\_pos, and balloon\_pos, and ASCII art for the game's title and start screen. It also defines game\_over and game\_start strings. The main window shows the 'emulator screen [94x28 chars]' with the game's title 'Balloon Shooting Game' and instructions: 'Use up and down key to move player and space button to shoot' and 'Press Enter to start'. Below the screen is a status bar with 'waiting for input'. The bottom half of the interface contains the 'registers' window showing CPU register values (AX=01, BX=00, CX=09, DX=02, SI=0000, DI=0000) and assembly instructions, and the 'stack' window showing the current stack state.

```
001
002
003 org 100h
004
005 ;-----+
006 ;-----+ Balloon Shooting Game
007 ;-----+
008
009 .model large           ; Use large memory model (separate code/data)
010 .data
011
012 exit db 0               ; Game exit flag
013 player_pos dw 1760d     ; Player's starting position on screen
014
015 arrow_pos dw 0d          ; Current position of arrow
016 arrow_status db 0d        ; 0 = arrow is ready, 1 = in air
017 arrow_limit dw 22d        ; Arrow travel limit (used to hide it)
018
019 balloon_pos dw 3860d      ; Balloon's starting position
020 balloon_status db 0d       ; Balloon state (currently unused)
021
022 direction db 0d          ; Player direction: up = 8, down = 2, idle = 0
023
024 state_buf db '00:00:00:00:00:00' ; Buffer to display score
025 hit_num db 0d              ; Not used
026 hits dw 0d                 ; Hit count
027 miss dw 0d                ; Missed balloon count
028
029
030 game_over_str dw '$',0ah,0dh ; Game over screen ASCII art
031 du ',0ah,0dh
032 du ',0ah,0dh
033 du ',0ah,0dh
034 du ',0ah,0dh
035 du ',0ah,0dh
036 du ',0ah,0dh
037 du ',0ah,0dh
038 du ',0ah,0dh
039 du ',0ah,0dh
040 du ',0ah,0dh
041 du ',0ah,0dh
042
043 game_over_str dw 'Press Enter to start again$',0ah,0dh
044
045 game_start_str dw '$',0ah,0dh ; Start screen ASCII art
046 du ',0ah,0dh
047 du ',0ah,0dh
048 du ',0ah,0dh
049 du ',0ah,0dh
050 du ',0ah,0dh
051 du ',0ah,0dh
052 du ',0ah,0dh
053 du ',0ah,0dh
054 du ',0ah,0dh
055 du ',0ah,0dh
056 du ',0ah,0dh
057 du ',0ah,0dh
058 du ',0ah,0dh
059 du ',0ah,0dh
060 du ',0ah,0dh
061 du ',0ah,0dh
062 du ',0ah,0dh
063 du '$',0ah,0dh
```

original source code

```

076: mov ah,1h
077: int 1fh
078: cmp al,0ffh
079: jne inside_loop
080: ; If key is pressed
081: ; If not, continue game logic
082: inc dx
083: cmp miss,9
084: jge game_over
085: mov dx,arrow_pos
086: cmp dx,loon_pos
087: je hit
088: ; If collision, register hit
089: cmp direction,8d
090: jne up
091: cmp direction,2d
092: je player_down
093: inc dx
094: cmp dx,arrow_limit
095: cmp arrow_pos,dx
096: jge hide_arrow
097: ; Hide arrow if past limit
098: cmp loon_pos,8d
099: jle miss_loon
100: jne render_loon
101: inc hits
102: int 21h
103: mov ah,2
104: mov dx,7d
105: int 21h
106: inc hits
107: inc hits
108: lea bx,state_buf
109: call show_score
110: lea bx,state_buf
111: add bx,109h
112: int 21h
113: int 21h
114: mov ah,2
115: mov dl,0dh
116: int 21h
117: jmp fire_loon
118: ; Create new balloon
119: render_loon:
120: mov cl,111b
121: mov es,111b
122: mov ds,111b
123: mov es:[bx],cx
124: sub loon_pos,160d
125: mov cl,15d
126: mov es:[bx],cl
127: mov bx,loon_pos
128: mov cx,1101b
129: mov es:[bx],cx
130: mov es:[bx],cx
131: mov es:[bx],cx
132: mov arrow_status,1d
133: cmp arrow_status,1d
134: jne render_arrow
135: jmp inside_loop2
136: ; If arrow in air, update it
137: ; Otherwise, skip
138: render_arrow:
139: mov cl,-1
140: ; Erase old arrow
141: 
```

emulator screen (94x25 chars)

Registers

	H	L	
AX	01	00	076F1: 24 115 t
BX	06	E0	076F2: 03 003 v
CX	0C	70	076F3: E9 233 0
DX	00	16	076F4: 00 000 NULL
SI	07	00	076F5: EB 233 6
DI	00	00	076F6: EB 233 6
IP	07	02	076F7: 03 131 t
SP	FFFE		076F8: 3E 062 >
BP	0000		076F9: 25 037 6
CS	07	00	076FA: 00 000 TAB
DS	07	00	076FB: 2C 124 1
SS	0700		076FC: EB 233 6
FS			076FD: 03 147 6
GS			076FE: EB 233 6
SI	0000		07700: 93 147 6
DI	0000		07701: 0B 139 v
IP	0702		07702: 0B 139 t
SP	0703		07703: 16 022 -

File math debug view external virtual devices virtual drive help

Load reload step back single step STOP step delay ms: 0

emulator: balloon blaster with description.com

registers

	H	L	
AX	07	00	0780: 07 02
BX	00	00	0781: 00 000 NULL
CX	00	00	0782: 00 000 NULL
DX	00	00	0783: 00 000 NULL
SI	0000		0784: 00 000 NULL
DI	0000		0785: 00 000 NULL
IP	0702		0786: 00 000 NULL
SP	0700		0787: 00 000 NULL
BP	0000		0788: 00 000 NULL
CS	0700		0789: 00 000 NULL
DS	0700		078A: 00 000 NULL

File math debug view external virtual devices virtual drive help

Load reload step back single step STOP step delay ms: 0

emulator screen (80x37 chars)

Registers

	H	L	
AX	01	00	F400: FF 255 RES
BX	06	E0	F4001: FF 255 RES
CX	0F	20	F4002: CD 205 t
DX	01	27	F4003: 21 033 v
SI	F400		F4004: CF 267 t
DI	0200		F4005: 00 000 NULL
IP	0200		F4006: 00 000 NULL
SP	0700		F4007: 00 000 NULL
BP	FFF8		F4008: 00 000 NULL
CS	F400		F4009: 00 000 NULL
DS	F400		F400A: 00 000 NULL
SS	0700		F400B: 00 000 NULL
FS			F400C: 00 000 NULL
GS			F400D: 00 000 NULL

File math debug view external virtual devices virtual drive help

Load reload step back single step STOP step delay ms: 0

emulator: balloon blaster with description.com

Registers

	H	L	
AX	01	00	F400: 02 00
BX	06	E0	F4001: FF 255 RES
CX	0F	20	F4002: CD 205 t
DX	01	27	F4003: 21 033 v
SI	F400		F4004: CF 267 t
DI	0200		F4005: 00 000 NULL
IP	0200		F4006: 00 000 NULL
SP	0700		F4007: 00 000 NULL
BP	FFF8		F4008: 00 000 NULL
CS	F400		F4009: 00 000 NULL
DS	F400		F400A: 00 000 NULL
SS	0700		F400B: 00 000 NULL
FS			F400C: 00 000 NULL
GS			F400D: 00 000 NULL

File math debug view external virtual devices virtual drive help

Load reload step back single step STOP step delay ms: 0

## 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](http://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