Green University of Bangladesh

Department of Computer Science and Engineering (CSE)
Semester: (Fall, Year: 2023), B.Sc. in CSE (Day)

Rapid Roll Game

Course Title: Microprocessors and Microcontrollers Lab
Course Code: CSE-304
Section: 221 D1

Students Details

Name	ID
Tanvir Ahmed	221002461
-	-

Submission Date: 09/01/2024
Course Teacher's Name: Mr. Montaser Abdul Quader

[For teachers use only: Don't write anything inside this box]

Lab Project Status	
Marks:	**Signature:**
Comments:	**Date:**

Contents

Chapter 1

Introduction

1.1 Overview

The Rapid Roll game is a classic mobile game that involves controlling a rolling ball through an endless descending environment. The player's objective is to keep the ball on the screen by moving it left or right, avoiding falling off the edges or colliding with obstacles. The game progressively increases in speed, challenging the player's reflexes and coordination. And I am building this game using assembly language.

1.2 Motivation

The primary motivation behind developing the Rapid Roll game in assembly language is to gain an understanding of this low-level programming language, helping us to understand the complexities of programming and computer architecture Assembly language provides a direct connection to the machine's hardware, allowing us to optimize code execution and achieve unparalleled performance. Mastering assembly language will equip us with the skills necessary to tackle complex programming challenges and develop high-efficiency applications. [**?**].

1.3 Problem Definition

1.3.1 Problem Statement

The problem statement for the Rapid Roll game development project is to create a fully functional game using assembly language, 8086 assembly. The game should feature responsive player input handling, smooth ball movement, dynamic obstacle generation, accurate scorekeeping, and visually engaging graphical elements. The project should demonstrate the capabilities of assembly language in handling graphics, player input, and game logic, showcasing its relevance in modern game development.

1.3.2 Complex Engineering Problem

Because of the inherent challenges of low-level programming, developing the Rapid Roll game in assembly language presents a complex engineering problem. Memory management, hardware interactions, and instruction-level programming all require attention in assembly language. The mechanics of the game, such as player input handling, ball physics, obstacle generation, and graphical rendering, must be efficiently implemented within the constraints of assembly language. Furthermore, optimizing the game's performance and ensuring smooth gameplay adds a new level of complexity to the project.

Table 1.1: Summary of the attributes touched by the mentioned projects

Name of the P Attributess	**Explain how to address**
P1: Depth of knowledge required	Deep understanding of assembly language
P2: Range of conflicting requirements	—-
P3: Depth of analysis required	Requires Abstract thinking and creativity and assembly language concepts
P4: Familiarity of issues	Familiarity with assembly language programming and its applications
P5: Extent of applicable codes	—-
P6: Extent of stakeholder involvement and conflicting requirements	—-
P7: Interdependence	Several interdependencies, between the assembly language code, the target application, and the application's requirements

1.4 Design Goals/Objectives

- Gain a profound understanding of assembly language programming.
- Apply assembly language knowledge to create a fully functional game.
- Develop insights into memory management, hardware interactions, and instruction-level programming.
- Gain hands-on experience in optimizing code execution and achieving real-time responsiveness.

1.5 Application

The assembly language code developed for the Rapid Roll game can be applied to various domains beyond game development. This project's skills, such as low-level memory management, hardware interactions, and instruction-level programming, can be applied

to other areas of software development. For example, developing high-performance embedded systems and device drivers requires the ability to optimize code execution and achieve real-time responsiveness. Understanding assembly language can also be used for reverse engineering and vulnerability analysis.

Chapter 2

Design/Development/Implementation of the Project

2.1 Introduction

In this project, we will develop a Rapid Roll game in assembly language. This will allow us to gain a deeper understanding of assembly language and how computers work and how games are programmed.

2.2 Project Details

The Rapid Roll game that we will develop will have the following features:

- A small cube that falls down a screen.
- Blocks that work like a parking spot.
- Points that the cube can collect.
- Different levels of difficulty.
- A score display.
- A game over screen.

2.3 Implementation(Code)

```
.MODEL SMALL
.STACK 100H

.DATA
 XCUBE DW 36
```

```
X1CUBE DW ?
X2CUBE DW ?
YCUBE DW 56
Y1CUBE DW ?
Y2CUBE DW ?

XLINE DW 70
X1LINE DW ?
X2LINE DW ?
YLINE DW 120
Y1LINE DW ?
Y2LINE DW ?

NXLINE DW 150
NX1LINE DW ?
NX2LINE DW ?
NYLINE DW 190
NY1LINE DW ?
NY2LINE DW ?

NNXLINE DW 10
NNX1LINE DW ?
NNX2LINE DW ?
NNYLINE DW 170
NNY1LINE DW ?
NNY2LINE DW ?

CHECK DW 1

randnumber dw 0

TIK DW ?

CHECK_UND DB 0

NCHECK_UND DB 0
NNCHECK_UND DB 0

RATE DW 1

SCOREMSG DB 'Score: $'
SCORE DW 0

BORDERX DW 2
BORDERY DW 13

COUNT DW 3
COUNT1 DW 3
```

```
 MMM DB 0AH,0DH,'                    RAPID ROLL$'
 MMM1 DB 0AH,0DH,0AH,0DH,0AH,0DH,0AH,0DH,0AH,0DH,0AH,0DH,0AH,0DH,'
PRESS 1 TO START THE GAME$'
 MMM2 DB 0AH,0DH,0AH,0DH,'          PRESS 2 FOR HELP$'
 MMM3 DB 0AH,0DH,0AH,0DH,'          PRESS 3 TO EXIT THE GAME$'
 MMM4 DB 0AH,0DH,0AH,0DH,0AH,0DH,0AH,0DH,0AH,0DH,0AH,0DH,0AH,0DH,0AH,0D
CSE 304 PROJECT$'
 MMM5 DB 0AH,0DH,0AH,0DH,' PREPARED BY TANVIR AHMED :D$'


 IMM1 DB 0AH,0DH,' You are a small box that falls along'
      db 0AH,0DH,' through the screen. By continuing in'
      db 0AH,0DH,' your descent through levels of the '
      db 0AH,0DH,' game,you gain points'
      db 0AH,0DH,0AH,0DH,0AH,0DH,' Controls:'
      db 0AH,0DH,' Press A to move left'
      db 0AH,0DH,' Press D to move right'
      db 0AH,0DH,0AH,0DH,0AH,0DH,0AH,0DH,0AH,0DH,0AH,0DH,0AH,0DH,0AH,0D

  GAMEOVERMSG DB 0AH,0DH,0AH,0DH,0AH,0DH,0AH,0DH,0AH,0DH,0AH,0DH,'
GAME OVER'
              DB 0AH,0DH,0AH,0DH,0AH,0DH,0AH,0DH,'
YOUR SCORE: $'

  GAMEOVERMSG1 DB 0AH,0DH,0AH,0DH,0AH,0DH,0AH,0DH,0AH,0DH,0AH,0DH,0AH,0
PRESS P TO PLAY AGAIN$'

  SCORE_ARRAY DB 2,2,3,0,0

.CODE


PLAY_AGAIN PROC
 MOV XCUBE , 36
 MOV YCUBE , 56

 MOV XLINE , 70
 MOV YLINE , 120

 MOV NXLINE , 150
 MOV NYLINE , 190

 MOV NNXLINE , 10
 MOV NNYLINE , 170
 MOV SCORE,0

     RET
```

```
ENDP  PLAY_AGAIN

UPRMVCUBE  PROC
        MOV  BX,X2CUBE
        MOV  XCUBE,BX

        MOV  BX,XCUBE
        MOV  X1CUBE,BX
        MOV  X2CUBE,BX
        ADD  X1CUBE,7

        MOV  BX,Y2CUBE
        MOV  YCUBE,BX

L1:
        MOV  AH,0CH
        MOV  AL,0
        MOV  CX,XCUBE
        MOV  DX,YCUBE
        INT  10H
        INC  XCUBE
        MOV  BX,X1CUBE
        CMP  XCUBE,BX
        JLE  L1

        MOV  BX,X2CUBE
        MOV  XCUBE,BX
        DEC  YCUBE

        MOV  BX,Y1CUBE
        CMP  YCUBE,BX
        JNE  L1

        MOV  BX,X2CUBE
        MOV  XCUBE,BX
        MOV  BX,Y2CUBE
        MOV  YCUBE,BX

        RET

ENDP  UPRMVCUBE

DOWNRMVCUBE  PROC
        MOV  BX,X2CUBE
        MOV  XCUBE,BX

        MOV  BX,XCUBE
        MOV  X1CUBE,BX
```

```
        MOV  X2CUBE,BX
        ADD  X1CUBE,7

        MOV  BX,Y2CUBE
        MOV  YCUBE,BX

L1DOWN:
        MOV  AH,0CH
        MOV  AL,0
        MOV  CX,XCUBE
        MOV  DX,YCUBE
        INT  10H
        INC  XCUBE
        MOV  BX,X1CUBE
        CMP  XCUBE,BX
        JLE  L1DOWN
        MOV  BX,X2CUBE
        MOV  XCUBE,BX
        INC  YCUBE
        MOV  BX,Y1CUBE
        CMP  YCUBE,BX
        JNE  L1DOWN

        MOV  BX,X2CUBE
        MOV  XCUBE,BX
        MOV  BX,Y2CUBE
        MOV  YCUBE,BX

        RET

ENDP  DOWNRMVCUBE

UPDRAWCUBE  PROC
        MOV  BX,XCUBE
        MOV  X1CUBE,BX
        MOV  X2CUBE,BX
        ADD  X1CUBE,7

        MOV  BX,YCUBE
        MOV  Y1CUBE,BX
        MOV  Y2CUBE,BX
        SUB  Y1CUBE,7

LUP:
        MOV  AH,0CH
        MOV  AL,11
        MOV  CX,XCUBE
        MOV  DX,YCUBE
```

```
        INT  10H
        INC  XCUBE
        MOV  BX,X1CUBE
        CMP  XCUBE,BX
        JLE  LUP
        MOV  BX,X2CUBE
        MOV  XCUBE,BX
        DEC  YCUBE
        MOV  BX,Y1CUBE
        CMP  YCUBE,BX
        JNE  LUP

        RET

ENDP  UPDRAWCUBE

DOWNDRAWCUBE PROC
        MOV  BX,XCUBE
        MOV  X1CUBE,BX
        MOV  X2CUBE,BX
        ADD  X1CUBE,7

        MOV  BX,YCUBE
        MOV  Y1CUBE,BX
        MOV  Y2CUBE,BX
        ADD  Y1CUBE,7

LDOWN:
        MOV  AH,0CH
        MOV  AL,11
        MOV  CX,XCUBE
        MOV  DX,YCUBE
        INT  10H
        INC  XCUBE
        MOV  BX,X1CUBE
        CMP  XCUBE,BX
        JLE  LDOWN

        MOV  BX,X2CUBE
        MOV  XCUBE,BX
        INC  YCUBE
        MOV  BX,Y1CUBE
        CMP  YCUBE,BX
        JNE  LDOWN

        RET

ENDP  DOWNDRAWCUBE
```

```
DELAY PROC
      MOV AX,00H
      INT 1AH
      MOV TIK,DX
      ADD TIK,1H
      DELAY1:
      MOV AX,00H
      INT 1AH
      CMP TIK,DX
      JGE DELAY1

      CMP CHECK,0
      JE DDD
      MOV AH,7
      INT 21H
      DEC CHECK
      DDD:
      RET

ENDP DELAY

DRAWLINE PROC

      MOV BX,XLINE
      MOV X1LINE,BX
      MOV X2LINE,BX
      ADD X1LINE,40

      MOV BX,YLINE
      MOV Y1LINE,BX
      MOV Y2LINE,BX
      SUB Y1LINE,3

      LINE:
      MOV AH,0CH
      MOV AL,13
      MOV CX,XLINE
      MOV DX,YLINE
      INT 10H
      INC XLINE
      MOV BX,X1LINE
      CMP XLINE,BX
      JLE LINE

      MOV BX,X2LINE
      MOV XLINE,BX
      DEC YLINE
```

```
        MOV BX,Y1LINE
        CMP YLINE,BX
        JNE LINE

        RET
ENDP DRAWLINE

RMVLINE PROC

        MOV BX,XLINE
        MOV X1LINE,BX
        MOV X2LINE,BX
        ADD X1LINE,40

        MOV XLINE,BX
        MOV BX,Y2LINE
        MOV YLINE,BX

LINE1:
        MOV AH,0CH
        MOV AL,0
        MOV CX,XLINE
        MOV DX,YLINE
        INT 10H
        INC XLINE
        MOV BX,X1LINE
        CMP XLINE,BX
        JLE LINE1

        MOV BX,X2LINE
        MOV XLINE,BX
        DEC YLINE
        MOV BX,Y1LINE
        CMP YLINE,BX
        JNE LINE1

        MOV BX,X2LINE
        MOV XLINE,BX
        MOV BX,Y2LINE
        MOV YLINE,BX

        RET

ENDP RMVLINE

DRAWNLINE PROC

        MOV BX,NXLINE
```

```
        MOV  NX1LINE,BX
        MOV  NX2LINE,BX
        ADD  NX1LINE,40

        MOV  BX,NYLINE
        MOV  NY1LINE,BX
        MOV  NY2LINE,BX
        SUB  NY1LINE,3

        NLINE:
        MOV  AH,0CH
        MOV  AL,13
        MOV  CX,NXLINE
        MOV  DX,NYLINE
        INT  10H
        INC  NXLINE
        MOV  BX,NX1LINE
        CMP  NXLINE,BX
        JLE  NLINE

        MOV  BX,NX2LINE
        MOV  NXLINE,BX
        DEC  NYLINE
        MOV  BX,NY1LINE
        CMP  NYLINE,BX
        JNE  NLINE



        RET
ENDP  DRAWNLINE

RMVNLINE  PROC

        MOV  BX,NXLINE
        MOV  NX1LINE,BX
        MOV  NX2LINE,BX
        ADD  NX1LINE,40

        MOV  NXLINE,BX
        MOV  BX,NY2LINE
        MOV  NYLINE,BX

NLINE1:
        MOV  AH,0CH
        MOV  AL,0
        MOV  CX,NXLINE
        MOV  DX,NYLINE
```

```
        INT  10H
        INC  NXLINE
        MOV  BX,NX1LINE
        CMP  NXLINE,BX
        JLE  NLINE1

        MOV  BX,NX2LINE
        MOV  NXLINE,BX
        DEC  NYLINE
        MOV  BX,NY1LINE
        CMP  NYLINE,BX
        JNE  NLINE1

        MOV  BX,NX2LINE
        MOV  NXLINE,BX
        MOV  BX,NY2LINE
        MOV  NYLINE,BX

        RET

ENDP  RMVNLINE

DRAWNNLINE  PROC

        MOV  BX,NNXLINE
        MOV  NNX1LINE,BX
        MOV  NNX2LINE,BX
        ADD  NNX1LINE,40

        MOV  BX,NNYLINE
        MOV  NNY1LINE,BX
        MOV  NNY2LINE,BX
        SUB  NNY1LINE,3

        NNLINE:
        MOV  AH,0CH
        MOV  AL,13
        MOV  CX,NNXLINE
        MOV  DX,NNYLINE
        INT  10H
        INC  NNXLINE
        MOV  BX,NNX1LINE
        CMP  NNXLINE,BX
        JLE  NNLINE

        MOV  BX,NNX2LINE
        MOV  NNXLINE,BX
        DEC  NNYLINE
```

```
        MOV  BX,NNY1LINE
        CMP  NNYLINE,BX
        JNE  NNLINE



        NNDIDI:

        RET
ENDP  DRAWNNLINE

RMVNNLINE  PROC

        MOV  BX,NNXLINE
        MOV  NNX1LINE,BX
        MOV  NNX2LINE,BX
        ADD  NNX1LINE,40

        MOV  NNXLINE,BX
        MOV  BX,NNY2LINE
        MOV  NNYLINE,BX

NNLINE1:
        MOV  AH,0CH
        MOV  AL,0
        MOV  CX,NNXLINE
        MOV  DX,NNYLINE
        INT  10H
        INC  NNXLINE
        MOV  BX,NNX1LINE
        CMP  NNXLINE,BX
        JLE  NNLINE1

        MOV  BX,NNX2LINE
        MOV  NNXLINE,BX
        DEC  NNYLINE
        MOV  BX,NNY1LINE
        CMP  NNYLINE,BX
        JNE  NNLINE1

        MOV  BX,NNX2LINE
        MOV  NNXLINE,BX
        MOV  BX,NNY2LINE
        MOV  NNYLINE,BX

        RET

ENDP  RMVNNLINE
```

```
GENERATE_RANDOM_NUMBER PROC

     pushall macro
     push ax
     push bx
     push cx
     push dx
endm

popall macro
     pop dx
     pop cx
     pop bx
     pop ax
endm

getrand macro cur
     pushall
     mov ah, 0
     int 1ah

     mov ax, dx
     mov dx, cx   ;dx:ax contains system time

     mov bx, 7261
     mul ax
     add ax, 1
     mov dx, 0
     mov bx, 200
     div bx

     mov cur, dx
     popall
endm
     mov cx, 0
     getrand randnumber

     RET
ENDP GENERATE_RANDOM_NUMBER

CHECK_UP_OR_DOWN PROC

     MOV BX,YLINE
     SUB BX,4
     CMP YCUBE,BX
     JE NEXTPHASE
     MOV CHECK_UND,0
```

```
        JMP  DID

        NEXTPHASE:
        MOV  BX,XLINE
        ADD  BX,42
        CMP  XCUBE,BX
        JL   NEXTPHASE1
        MOV  CHECK_UND,0
        JMP  DID

        NEXTPHASE1:
        MOV  BX,XLINE
        SUB  BX,9
        CMP  BX,XCUBE
        JL   LASTPHASE
        MOV  CHECK_UND,0
        JMP  DID

        LASTPHASE:
        MOV  CHECK_UND,1

        DID:
        RET
ENDP  CHECK_UP_OR_DOWN

NCHECK_UP_OR_DOWN  PROC

        MOV  BX,NYLINE
        SUB  BX,4
        CMP  YCUBE,BX
        JE   NNEXTPHASE
        MOV  NCHECK_UND,0
        JMP  NDID

        NNEXTPHASE:
        MOV  BX,NXLINE
        ADD  BX,42
        CMP  XCUBE,BX
        JL   NNEXTPHASE1
        MOV  NCHECK_UND,0
        JMP  NDID

        NNEXTPHASE1:
        MOV  BX,NXLINE
        SUB  BX,9
        CMP  BX,XCUBE
        JL   NLASTPHASE
        MOV  NCHECK_UND,0
```

```
        JMP  NDID

        NLASTPHASE:
        MOV  NCHECK_UND,1

        NDID:
        RET
ENDP  NCHECK_UP_OR_DOWN

NNCHECK_UP_OR_DOWN  PROC

        MOV  BX,NNYLINE
        SUB  BX,4
        CMP  YCUBE,BX
        JE  NNNEXTPHASE
        MOV  NNCHECK_UND,0
        JMP  NNDID

        NNNEXTPHASE:
        MOV  BX,NNXLINE
        ADD  BX,42
        CMP  XCUBE,BX
        JL  NNNEXTPHASE1
        MOV  NNCHECK_UND,0
        JMP  NNDID

        NNNEXTPHASE1:
        MOV  BX,NNXLINE
        SUB  BX,9
        CMP  BX,XCUBE
        JL  NNLASTPHASE
        MOV  NNCHECK_UND,0
        JMP  NNDID

        NNLASTPHASE:
        MOV  NNCHECK_UND,1

        NNDID:
        RET
ENDP  NNCHECK_UP_OR_DOWN

NEXT_XLINE  PROC

        CMP  YLINE,15
        JGE  NOCHANGE
        MOV  YLINE,196
        MOV  Y2LINE,196
```

```
        MOV BX,RANDNUMBER
        MOV XLINE,BX
        MOV X1LINE,BX

        NOCHANGE:
        RET
ENDP NEXT_XLINE

NEXT_NXLINE PROC

        CMP NYLINE,15
        JGE NNOCHANGE
        MOV NYLINE,196
        MOV NY2LINE,196

        MOV BX,RANDNUMBER
        MOV NXLINE,BX
        MOV NX1LINE,BX

        NNOCHANGE:
        RET
ENDP NEXT_NXLINE

NEXT_NNXLINE PROC

        CMP NNYLINE,15
        JGE NNNOCHANGE
        MOV NNYLINE,196
        MOV NNY2LINE,196

        MOV BX,RANDNUMBER
        MOV NNXLINE,BX
        MOV NNX1LINE,BX

        NNNOCHANGE:
        RET
ENDP NEXT_NNXLINE

GET_THE_SCORE PROC

        MOV AH,0
        MOV AL,2
        INT 10H

        MOV DI,0

        MOV AX,SCORE
        MOV BX,10
```

```
        SCL:
        CMP AX,0
        JE SCL2
        DIV BX
        PUSH DX
        MOV DX,0
        INC COUNT
        JMP SCL

        SCL2:
        MOV BX,COUNT
        MOV COUNT1,BX
        SCL1:
        CMP COUNT,0
        JE SCL3
        DEC COUNT
        POP DX
        ADD DX,48
        MOV SCORE_ARRAY[DI],DL
        INC DI
        JMP SCL1

        SCL3:
        MOV AX,13H
        INT 10H
        MOV DI,0

        RET
ENDP GET_THE_SCORE


BORDER PROC

        BOR:
        MOV AH,0CH
        MOV AL,15
        MOV CX,BORDERX
        MOV DX,BORDERY
        INT 10H
        INC BORDERX
        CMP BORDERX,319
        JE NNP
        JMP BOR

        NNP:
        MOV BORDERX,2
        MOV BORDERY,195
```

```
        BOR1:
        MOV AH,0CH
        MOV AL,15
        MOV CX,BORDERX
        MOV DX,BORDERY
        INT 10H
        INC BORDERX
        CMP BORDERX,319
        JE NNP1
        JMP BOR1

        NNP1:
        MOV BORDERX,2
        MOV BORDERY,13
        BOR2:
        MOV AH,0CH
        MOV AL,15
        MOV CX,BORDERX
        MOV DX,BORDERY
        INT 10H
        INC BORDERY
        CMP BORDERY,195
        JE NNP2
        JMP BOR2

        NNP2:
        MOV BORDERX,319
        MOV BORDERY,13
        BOR3:
        MOV AH,0CH
        MOV AL,15
        MOV CX,BORDERX
        MOV DX,BORDERY
        INT 10H
        INC BORDERY
        CMP BORDERY,196
        JE DADA
        JMP BOR3

        DADA:
        MOV BORDERX,2
        MOV BORDERY,13
        RET
ENDP BORDER

MAIN_MENU PROC
        MPR:
        MOV AH,9
```

```
        LEA  DX,MMM
        INT  21H
        LEA  DX,MMM1
        INT  21H
        LEA  DX,MMM2
        INT  21H
        LEA  DX,MMM3
        INT  21H
        LEA  DX,MMM4
        INT  21H
        LEA  DX,MMM5
        INT  21H

        LOOPP:
        MOV  AH,7
        INT  21H
        CMP  AL,'1'
        JE   STG
        CMP  AL,'2'
        JE   INSTRUC
        CMP  AL,'3'
        JE   STG
        JMP  LOOPP


        INSTRUC:
        CALL  RESET_THE_SCREEN

        MOV  AH,9
        LEA  DX,IMM1
        INT  21H

        MOV  AH,1
        INT  21H

        CALL  RESET_THE_SCREEN

        JMP  MPR

        STG:
        CALL  RESET_THE_SCREEN
        JMP  GAME
        RET
ENDP  MAIN_MENU


RESET_THE_SCREEN  PROC
        MOV  AH,0
```

```
        MOV  AL,2
        INT  10H
        MOV  AX,13H
        INT  10H

        RET

ENDP  RESET_THE_SCREEN

GAME_OVER  PROC

        CALL  RESET_THE_SCREEN

        MOV  AH,9
        LEA  DX,GAMEOVERMSG
        INT  21H

        MOV  DI,0
        MOV  BX,COUNT1
        MOV  COUNT,BX
        GLO:
        CMP  COUNT,0
        JE    GLO1
        DEC  COUNT
        MOV  AH,2
        MOV  DL,SCORE_ARRAY[DI]
        ADD  DL,48
        INT  21H
        INC  DI
        JMP  GLO


        GLO1:
        MOV  AH,9
        LEA  DX,GAMEOVERMSG1
        INT  21H

        AGA:
        MOV  AH,7
        INT  21H
        CMP  AL,'X'
        JE  GGG
        CMP  AL,'x'
        JE  GGG
        CMP  AL,'P'
        JE  DIDA
        JMP  AGA
```

```
        DIDA:
        CALL RESET_THE_SCREEN
        CALL PLAY_AGAIN
        JMP GAME

        GGG:
        RET
ENDP GAME_OVER

MAIN PROC
        MOV AX,@DATA
        MOV DS,AX

        ; Graphic mode
        MOV AX,13H
        INT 10H
        CALL MAIN_MENU

        ;LEA DX,SCOREMSG
        ;MOV AH,9
        ;INT 21H
        CALL GET_THE_SCORE
GAME:

        MOV AH,1
        INT 16H

        JZ NOKEYPRESS
        JNZ KEYPRESS

        NOKEYPRESS:
        CALL NEXT_XLINE
        CALL NEXT_NXLINE
        CALL NEXT_NNXLINE

        CALL GENERATE_RANDOM_NUMBER

        CALL BORDER
        CALL UPDRAWCUBE
        CALL DRAWLINE
        CALL DRAWNLINE
        CALL DRAWNNLINE

        CALL DELAY

        CALL UPRMVCUBE
        CALL RMVLINE
        CALL RMVNLINE
```

```
CALL  RMVNNLINE
JMP  AGAIN

KEYPRESS:
MOV  AH,0
INT  16H
CMP  AL,'A'
JE  MOVELEFT
CMP  AL,'a'
JE  MOVELEFT
CMP  AL,'D'
JE  MOVERIGHT
CMP  AL,'d'
JE  MOVERIGHT
JMP  AGAIN

MOVELEFT:
SUB  XCUBE,2
SUB  X2CUBE,2
JMP  AGAIN

MOVERIGHT:
ADD  XCUBE,2
ADD  X2CUBE,2

AGAIN:
CALL  CHECK_UP_OR_DOWN
CALL  NCHECK_UP_OR_DOWN
CALL  NNCHECK_UP_OR_DOWN
CMP  CHECK_UND,1
JE  AGAIN1
CMP  NCHECK_UND,1
JE  AGAIN1
CMP  NNCHECK_UND,1
JE  AGAIN1

INC  YCUBE
INC  SCORE
AGAIN2:
DEC  YLINE
DEC  NYLINE
DEC  NNYLINE
CMP  YCUBE,198
JE  EXIT
CMP  YCUBE,19
JE  EXIT
;CALL  GET_THE_SCORE
;CALL  RESET_THE_SCREEN
```

```
        JMP  GAME

        AGAIN1 :
        DEC  YCUBE
        JMP  AGAIN2
        EXIT :
        CALL  GAME_OVER
        MOV  AH, 1
        INT  21H

        MOV  AH, 0
        MOV  AL, 2
        INT  10H

        MOV  AH, 4CH
        INT  21H
        MAIN  ENDP
END  MAIN
```

2.4 Algorithms

- Initialization:
  - Display the menu
  - After starting the game, Create initial positions for the cube and blocks.
- Get player input:
  - Check for key presses (A, D).
  - Update the cube's horizontal position based on input.
- Move blocks:
  - Move all blocks up the screen at a constant speed.
  - If a block reaches the top, remove it and generate a new one at the bottom.
- Check for collisions:
  - Check if the cube intersects with any blocks.
  - If a collision occurs, move the cube up at a constant speed of the block.
- Delay:
  - Introduce a short delay to control game speed and make it visually smooth.
- Game over:
  - Display a game over screen with the final score.
  - Prompt the player to play again.

Chapter 3

Performance Evaluation

3.1 Simulation Environment/ Simulation Procedure

Since we're working with a game in assembly language and aiming for optimal performance, using DOSBox as the environment is a great choice. However, emu8086 might indeed be too slow for smooth gameplay here.

3.2 Results Analysis/Testing

Figure 3.1: Interface

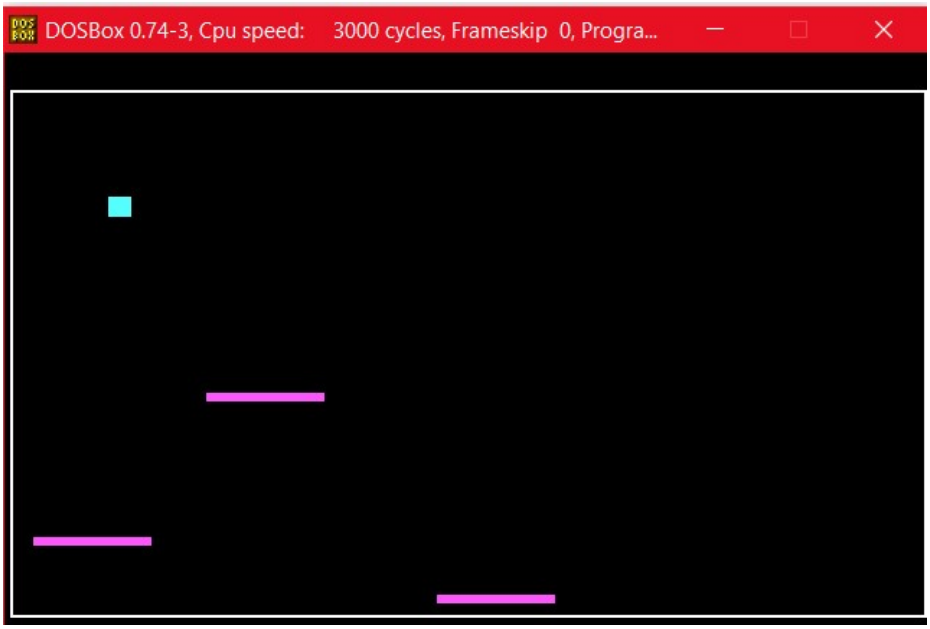

Figure 3.2: In-game Interface

Figure 3.3: Scoreboard

3.3 Results Overall Discussion

We can see in the above picture, that we have a menu, after starting the game we draw the cube and three blocks. Pressing any key the game will start and can be controlled by the keys A and D. If the cube touches the above border or down border, the game will be over and last it displays the scoreboard.

3.3.1 Complex Engineering Problem Discussion

- User Interface and Animation: Maintaining smooth animation for the falling cube, moving blocks and score updates requires efficient control of graphical operations and timer interrupts.
- Collision detection: Accurately determining collisions between the cube and the moving blocks can involve complex calculations and optimization to avoid performance bottlenecks.
- Responsive controls: Handling user input for moving the cube left and right needs timely detection and translation into game actions, ensuring a smooth and responsive experience.
- Dynamic block generation: Randomly generating the moving blocks with varying positions and speeds adds challenge and replayability to the game.

Chapter 4

Conclusion

4.1 Discussion

The Rapid Roll game successfully implements game mechanics with smooth animation, responsive controls, and dynamic block generation. Random block generation adds replayability value, ensuring each playthrough feels unique. Last we get the score perfectly.

4.2 Limitations

- The current design relies on keyboard input for controlling the cube, which may limit accessibility for players with specific needs.
- The scoring system only reflects the distance traveled, not factoring in the complexity of blocks passed.
- The lack of additional features like achievements or power-ups restricts the game's long-term engagement potential.
- The reliance on random line generation can sometimes lead to repetitive gameplay patterns.

4.3 Scope of Future Work

- Expanding the game by adding different levels with unique themes, obstacles, and challenges could enhance player engagement and provide a sense of progression.
- Implementing online multiplayer functionality would allow players to compete against each other or collaborate in co-op mode, significantly increasing the game's social and competitive aspects.