SCHOOL OF COMPUTER SCIENCE ENGINEERING

Hangman Game Development using Emu8086

Vishal R (19BCE0695)
V. Rajeevvas (19BCE0064)
Rahul Kumar (19BCE0965)
S. Bharath Viswath (19BCE0453)

Project Report
of
CSE2006 – MICROPROCESSOR & INTERFACING

Fall Semester 2021-22

SCHOOL OF COMPUTER SCIENCE ENGINEERING

Hangman Game Development using Emu8086

By:

Vishal R (19BCE0695)   **Rahul Kumar (19BCE0965)**
S. Bharath Viswath (19BCE0453)   **V. Rajeevvas (19BCE0064)**

CSE2006 – Microprocessor & Interfacing
Fall Semester 2021-22

Submitted to

Faculty: Prof. Gautam Narayan
Date: 09/12/2021
Slot:L27+28

CONTENTS	PAGE No.
Abstract	4
Introduction	5
Literature Survey	6
Novelty, Block Diagram	9
Flow chart	10
Implementation	11
Screenshots of the prototype	12
Results	15
Conclusion	15
References	16
Appendix (code)	17
Plagiarism report	20

Abstract

This project describes the realization of the Hangman game in emu8086, using the assembler programming language. Knowledge of the work in the software tool is demonstrated, as well as advanced knowledge of assembler and work with library functions. The obtained game uses less CPU time than its realizations in other higher level programming languages. We have used different registers and functions to make our project less complex and easy to run. Our project runs on very simple, easy to understand and basic assembly language programming. Our project demonstrates the game hangman in which for every wrong guess of the word provided hangman loses its life ultimately if the user guesses the word correctly without taking all lives of hangman user wins else if the user is not able to guess the user lose all the lives and hangman dies. Our project is solely made in emu8086 using assembly language programming.

Introduction

Main aim of our Project to draw a graphic figure piece by piece to represent the number to chances/lives left for the player. And the main objective of the player is to guess correct word in the given number of chances and Hangman Diagram will be drawn respectively for wrong Guesses.

Assembly language is a low-level programming language, and it is specific to a particular processor architecture. In this project, we have decided to use x86 processor for development of the project. To understand our realization of the game, it is necessary to get acquainted with the processor architecture, especially with registers and the way the strings are processed. The game is realized using the already known term which is to be guessed. For our purpose, we chose that the number of terms should be at least 10. At the beginning of the game, the player tries to guess the term by writing letters from the standard input. If the entered letter does not exist in the specified term, then the part of Hangman is added. However, if the letter exists in the term which a player tries to guess, then the letter is written to the appropriate position in the word. If the letter has already been hit, the user is informed necessarily about it, and if the user by accident writes it, then it does not affect the game. Every time a new letter is entered from the standard input, it is necessary to print it on the screen. When a user guesses the current term, they are allowed to hit the new one. If the user misses the letters enough times, the entire Hangman is drawn out, and a player loses the game. The description of the project code and the principle of the game operation are described in the following subsections of the project. This project is made in most simple way possible and in oldest programming language – assembly programming language using the one of the oldest emulators emu8086.

Literature Survey

1. Development of the Game Hangman in Assembly Programming Language

Author - Stefan Tešanović and Predrag Mitrović

Telfor Journal, Vol. 10, No. 2, 2018

This paper describes the realization of the Hangman game in Microsoft Visual Studio, using the assembler programming language, Kip Irvine's and MASM libraries. Knowledge of the work in the software tool is demonstrated, as well as advanced knowledge of assembler and work with library functions. The obtained game uses less CPU time than its realizations in other higher level programming languages. Unlike programming in higher programming languages, where it is necessary to understand the endpoints of the procedural or object-oriented programming, for programming in assembler it is necessary to understand hardware. Assembly language is not portable, because it is designed for a specific processor family. Thus, there are many different assembly languages widely used today, each based on a processor family . Of all the hardware, the most important thing to us is the architecture of the central processor unit (CPU). In the CPU, where all calculations and logical operations take place, there are a limited number of storage locations named registers, a high-frequency clock, a control unit and an arithmetic logic unit. In this paper, we have decided to use x86 processor for development of the project. To understand our realization of the game, it is necessary to get acquainted with the processor architecture, especially with registers and the way the strings are processed

2. Using Assembly Language for Creating Game

Author - Haris Turkmanović, David Vukoje, leksandra Lekić

Date – 18/06/2018

Journal – Icetran

The aim of this paper is to demonstrate some interesting and useful approaches for writing a program in the assembly language. In order to demonstrate the possibilities of the assembly language, a project called "Arkanoid" was created. This project is written in assembly language and it presents few interesting algorithms. Assembly language, which is used for designing the game is x86 Assembly language, which produces object code for the x86 class of processors. As a working environment is chosen Visual Studio 2015, because it gives the useful tools for debugging and testing of the created software (game). Execution of the program results in a "Arkanoid" game, placed in Windows OS Console. The aim of this paper is to demonstrate some interesting and useful approaches for writing a program in the assembly language. In order to demonstrate the possibilities of the assembly language, a project called "Arkanoid" was created. This project is written in assembly language and it presents few interesting algorithms. Assembly language, which is used for designing the game is x86 Assembly language, which produces object code for the x86 class of processors. As a working environment is chosen Visual Studio 2015, because it gives the useful tools for debugging and testing of the created software (game). Execution of the program results in a "Arkanoid" game, placed in Windows OS Console.

3. An Overview of Microprocessors and Assembly Language Programming

Author - Zaman, Md, Monira, Nusrath
Date – 17/12/2017

The microprocessor is a very useful tool for our modern communication. In fact, the performance of any computer is vastly dependent on them. In this paper, we have focused on the evolution of the microprocessors first, and then went for the categorization, organization, operation and some other fundamental things. Discussed the several cycles that a microprocessor goes through and at last, gave some ideas and aspects of assembly language programming. The microprocessor is a very useful tool for our modern communication. In fact, the performance of any computer is vastly dependent on them. In this paper, we have focused on the evolution of the microprocessors first, and then went for the categorization, organization, operation and some other fundamental things. Discussed the several cycles that a microprocessor goes through and at last, gave some ideas and aspects of assembly language programming.

4. A full system x86 simulator for teaching computer organization

Author - Priyadarshini Komala, Michael David Black

This paper describes a new graphical computer simulator developed for computer organization students. Unlike other teaching simulators, our simulator faithfully models a complete personal computer, including an 1386 processor, physical memory, I/O ports, floppy and hard disks, interrupts, timers, and a serial port. It can run PC software such as free DOS, Windows, and Minix, and can run as Java applet Graphical user interfaces allow students to view and modify the processor, memory, disks, and hardware devices at runtime. The simulator includes a processor development utility that allows students to design their own Datapath and control units, and run their custom processor alongside the x86 processor. The paper describes labs where students use the simulator to write x86 assembly programs, device drivers, hardware controllers, and design both simple and pipelined processors.This paper describes a new graphical computer simulator developed for computer organization students. Unlike other teaching simulators, our simulator faithfully models a complete personal computer, including an 1386 processor, physical memory, I/O ports, floppy and hard disks, interrupts, timers, and a serial port. It can run PC software such as free DOS, Windows, and Minix, and can run as Java applet Graphical user interfaces allow students to view and modify the processor, memory, disks, and hardware devices at runtime. The simulator includes a processor development utility that allows students to design their own Datapath and control units, and run their custom processor alongside the x86 processor. The paper describes labs where students use the simulator to write x86 assembly programs, device drivers, hardware controllers, and design both simple and pipelined processors.

5. An analysis of 8086 instruction set usage in MS DOS programs

Authors – T. L. Adams, R. E. Zimmerman
Journal – ACM SIGARCH
Date – 07/09/2011

An architectural evaluation must be based upon real programs in an actual operating environment. The ubiquitous IBM personal computer running MS DOS represents an excellent test bed for architectural evaluation of Intel 8086 systems. There are many programs and tools available to evaluate the performance of IBM Personal Computers and compatibles; these evaluation tools are intended to relate the performance of one machine to another. Very little data is available on dynamic instruction traces in systems using an 8086. This paper reports on dynamic traces of 8086/88 programs obtained using software tracing tools (described below). The objective of this work is to analyse instruction usage and addressing modes used in actual software. The system used to obtain the dynamic instruction frequenc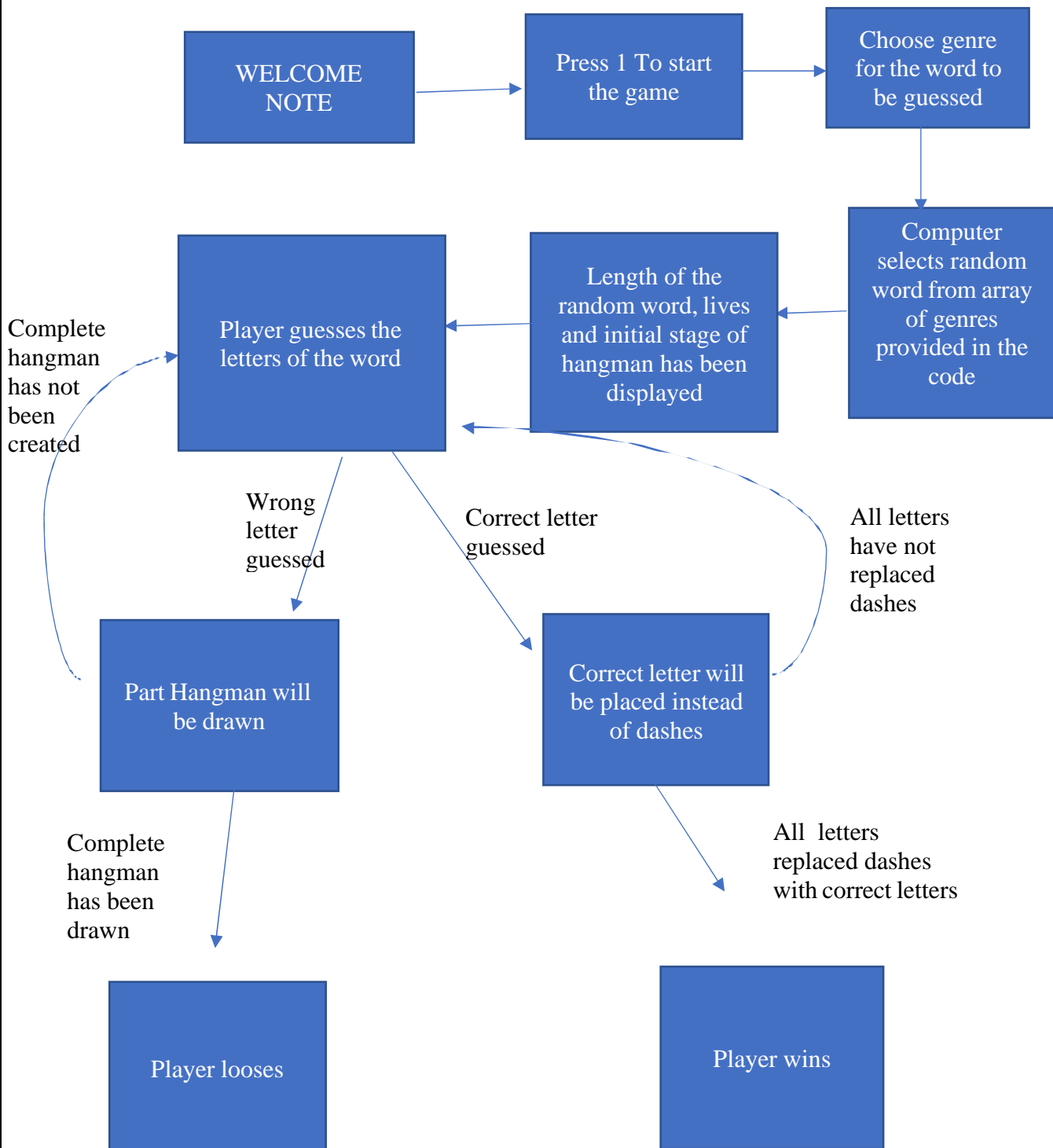ies was a compatible running MS DOS 3.1 with BIOS. An architectural evaluation must be based upon real programs in an actual operating environment. The ubiquitous IBM personal computer running MS DOS represents an excellent test bed for architectural evaluation of Intel 8086 systems. There are many programs and tools available to evaluate the performance of IBM Personal Computers and compatibles; these evaluation tools are intended to relate the performance of one machine to another. Very little data is available on dynamic instruction traces in systems using an 8086. This paper reports on dynamic traces of 8086/88 programs obtained using software tracing tools (described below). The objective of this work is to analyse instruction usage and addressing modes used in actual software. The system used to obtain the dynamic instruction frequencies was a compatible running MS DOS 3.1 with BIOS.
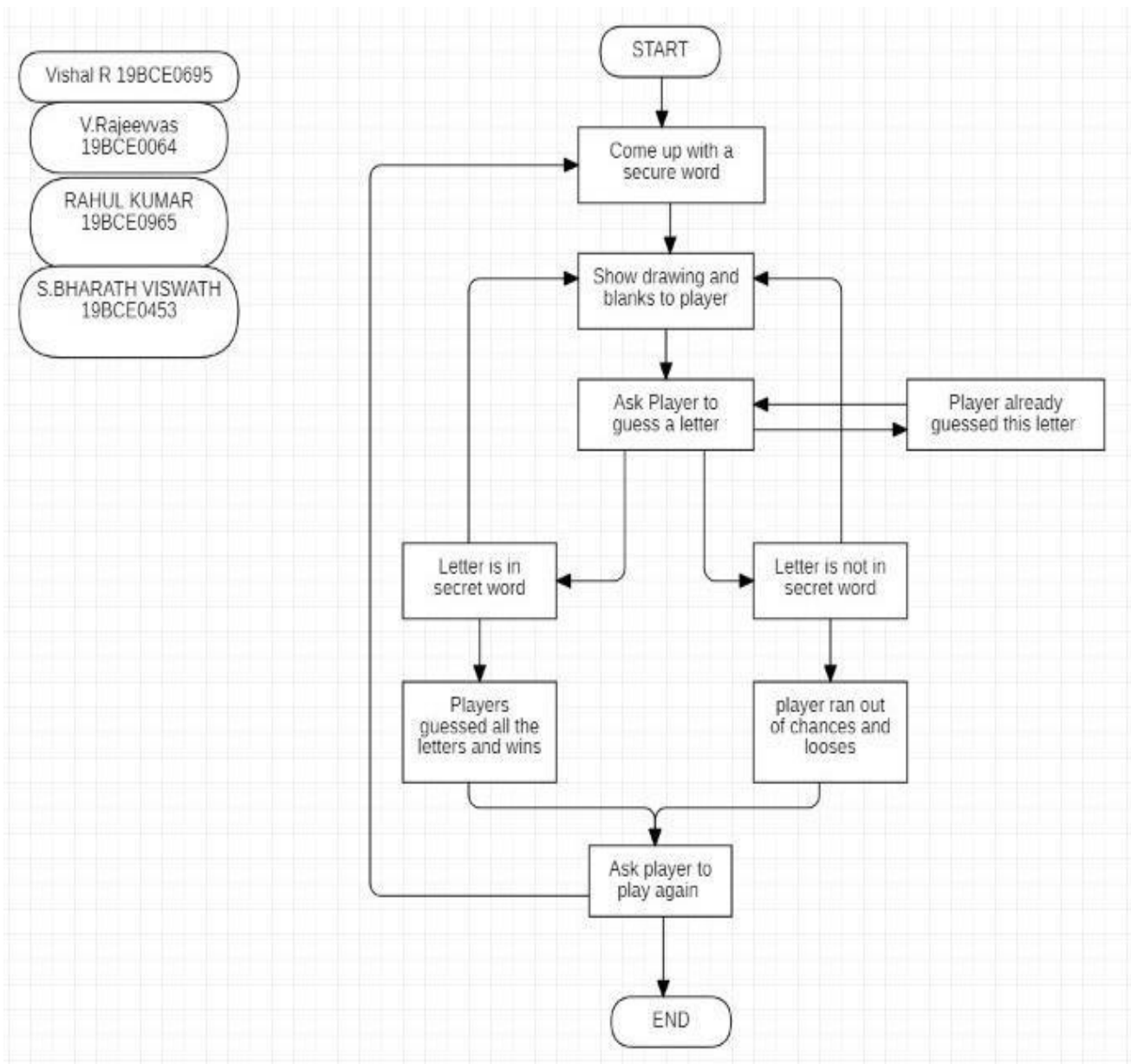
Novelty In Proposed work

We have tried to make our game more interactive and in order to make more interactive we have provided with an in interactive interface. We have the given the user choice to choose the genre from which he wants to guess the word. With every wrong answer given by the user a life will be deducted, and the picture of hangman will be incremented with it's face, body, hands and legs.
We have also successfully done and run everything with the help of 8086 ALP and in emu8086

Block Diagram

```
WELCOME          →    Press 1 To start    →    Choose genre
NOTE                  the game                 for the word to
                                               be guessed
                                                    │
                                                    ▼
Player guesses the  ←  Length of the        ←  Computer
letters of the word    random word, lives       selects random
                       and initial stage of     word from array
                       hangman has been         of genres
                       displayed                provided in the
                                                code
```

Complete hangman has not been created

Wrong letter guessed

Correct letter guessed

All letters have not replaced dashes

Part Hangman will be drawn

Correct letter will be placed instead of dashes

Complete hangman has been drawn

All letters replaced dashes with correct letters

Player looses

Player wins

Flow Chart for Algorithm

Vishal R 19BCE0695

V.Rajeevvas 19BCE0064

RAHUL KUMAR 19BCE0965

S.BHARATH VISWATH 19BCE0453

START

Come up with a secure word

Show drawing and blanks to player

Ask Player to guess a letter

Player already guessed this letter

Letter is in secret word

Letter is not in secret word

Players guessed all the letters and wins

player ran out of chances and looses

Ask player to play again

END

Implementation

The Hangman program randomly selects a secret. word from a list of secret words and the number of letters in the word will be displayed to the user. Then the player will guess a letter. If that letter is in the word(s) then It will write the letter everyplace it appears. If the letter isn't in the word then we cross out the lifelines. The player will continue guessing the letters. until he can either solve the word (or phrase) and WIN or he will end up losing all the lifelines and he will be declared a LOSER.

Explanation of all the instruction sets used in the code

Mov - Moves data from register to register, register to memory, memory to register, memory to accumulator, accumulator to memory, etc.

call - Calls a procedure whose address is given in the instruction and saves their return address to the stack

Inc - Increment Register or memory by 1

Cmp - Compare Immediate data, register or memory with accumulator, register or memory location.

Je - Jump if zero or equal i.e. when ZF = 1

Jmp - Causes the program execution to jump unconditionally to the memory address or label given in the instruction

xor - Performs bit by bit logical XOR operation of two operands and places the result in the specified destination

Div - Unsigned 8-bit or 16-bit division.

Add - Adds data to the accumulator i.e., AL or AX register or memory location.

Sub - Subtract immediate data from accumulator, memory or register

Loop - Jump to defined label until CX = 0

Dec - Decrement register or memory by 1

Ret - Returns program execution from a procedure (subroutine) to the next instruction or main program

Jb - Jump if below, not above, equal or carry i.e. when CF = 0
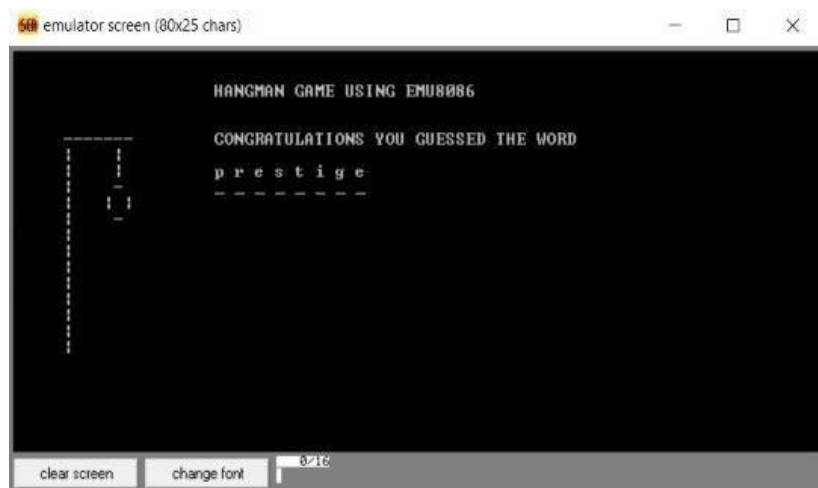
Screenshot of prototype

In this screen the user will get a welcome note and further he/she has to choose the genre, according to which the code will choose a random word to be guessed.
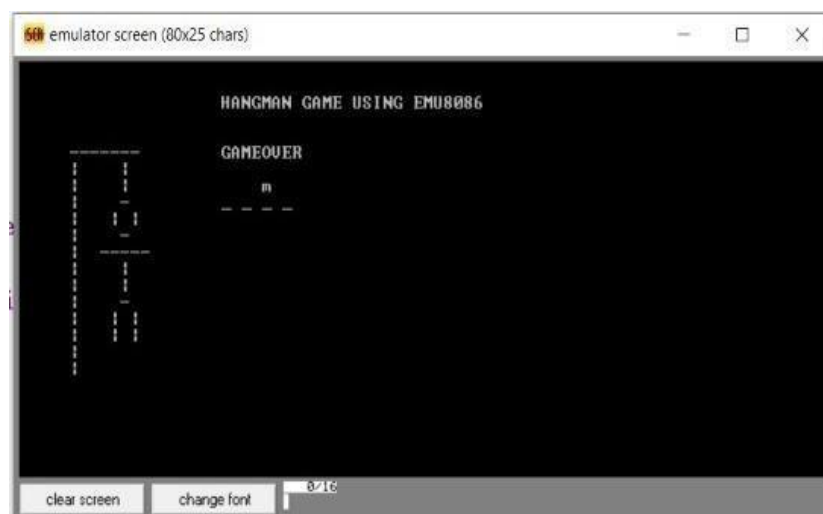

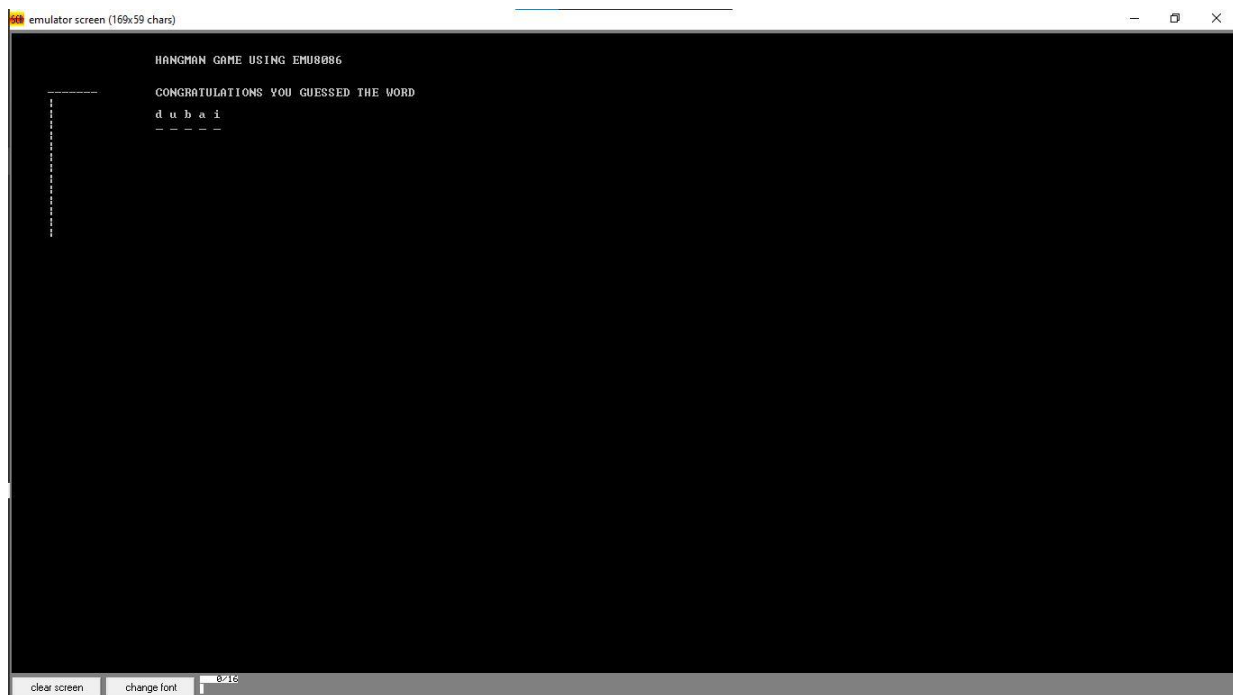User has to guess a random word picked by code according to the genre choosed.
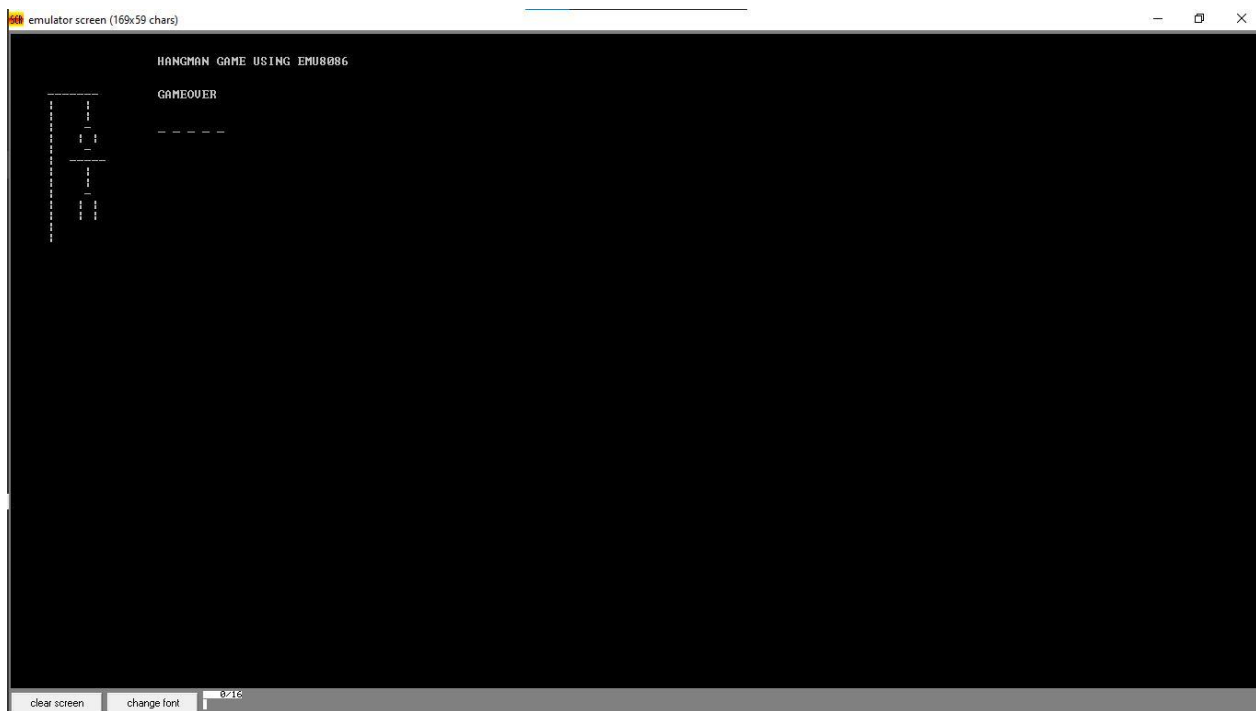
The case where player wins

The case where player looses

```
                    HANGMAN GAME USING EMU8086                                    14

                    GAMEOVER
 _____
|         |
|         |
|        ( )
|         |
 _____
|
|        | |
|        | |
|
```

```
                    HANGMAN GAME USING EMU8086
 _____
|                   CONGRATULATIONS YOU GUESSED THE WORD
|                   d u b a i
|                   _ _ _ _ _
|
|
|
|
|
```

Result

Our project will offer the user an interactive hangman game. For our purpose, we chose that the number of terms should be at least 10. At the beginning of the game, the player tries to guess the term by writing letters from the standard input. If the entered letter does not exist in the specified term, then the part of Hangman is added. However, if the letter exists in the term which a player tries to guess, then the letter is written to the appropriate position in the word. If the letter has already been hit, the user is informed necessarily about it, and if the user by accident writes it, then it does not affect the game. Every time a new letter is entered from the standard input, it is necessary to print it on the screen. When a user guesses the current term, they are allowed to hit the new one. If the user misses the letters enough times, the entire Hangman is drawn out, and a player loses the game. We have succeeded in making the game interactive in emulator 8086 which the user can enjoy.

Conclusion

In this project, we describe a realization of the game Hangman. Although the game is well known and built numerous times, our realization gives an exciting approach to a programming assignment made entirely in assembly programming language. This realization is especially interesting for educational purposes because it provides an exciting way to introduce students to assembly programming and architecture of x86 processors.

References:

1. TEACHING OF 8088/86 PROGRAMMING WITH 8086 ASSEMBLY EMULATOR-Date - 2020/04/11 Journal -2nd Joint International Conference on Science, Technology andInnovation, Mandalay (IcSTIM 2019)
2. Development of the Game in Assembly Programming Language-Date – 2018/11/01 Journal – Telfor journal
3. An Overview of Microprocessors and Assembly Language Programming-Date - 2017/12/01 Journal - Advances in Interconnect Technologies: An International Journal (AITIJ)
4. Using Assembly Language for Creating Games-Date – 2018/06/18 Conference - IcEtran
5. https://en.wikipedia.org/wiki/Hangman_(game)Researchgate.net
6. https://www.gutenberg.org/files/41727/41727-h/41727-h.htm#GameI_50
7. https://stackoverflow.com/questions/17855817/generating-a-random-number-within-range-of-0-9-in-x86-8086-assembly

.

Appendix

Below are some important parts and functions of the code (this is not the full code)

This part of code will choose a random word based upon genre selected by the player from the arrays.

```asm
053  movies:
054      MOV AH, 00h  ; interrupts to get system time
055      INT 1AH      ; CX:DX now hold number of clock ticks since midnight
056      mov  ax, dx
057      xor  dx, dx
058      mov  cx, 5
059      div  cx       ; here dx contains the remainder of the division - from 0 to 4
060      add  dl, '0'  ; to ascii from '0' to '4'
061
062      PRINTN '*Random number choosen:'
063      mov ah, 2h    ; call interrupt to display a value in DL
064      int 21h
065
066      mov bl,dl
067      sub bl,30h    ; now the bl value is from 0h to 4h
068      PRINTN ''
069
070      PRINTN '*Word selected:'
071      cmp bl,00h
072      je movie_first
073      cmp bl,01h
074      je movie_second
075      cmp bl,02h
076      je movie_third
077      cmp bl,03h
078      je movie_fourth
079      cmp bl,04h
080      je movie_fifth
081
082  movie_first:
083      lea di,movie_word[0]
084      mov cx,8
085      mov dx,di
086      call printit
087      jmp getchar
088  movie_second:
089      lea di,movie_word[9]
090      mov cx,5
091      mov dx,di
092      call printit
093      jmp getchar
094  movie_third:
095      lea di,movie_word[15]
096      mov cx,5
097      mov dx,di
098      call printit
099      jmp getchar
100  movie_fourth:
101      lea di,movie_word[21]
102      mov cx,6
103      mov dx,di
104      call printit
105      jmp getchar
106  movie_fifth:
107      lea di,movie_word[28]
108      mov cx,9
109      mov dx,di
110      call printit
```

Print_live function will tell us about how many lives are still remaining

```
288  printlife:
289      mov dh,5
290      mov dl,20
291      mov bh,0
292      mov ah,2h
293      int 10h          ;set cursor position DH = row.DL = column.BH = page number (0..7).
294
295      PRINT 'LIVES:'
296      mov dx,lives     ;printing lives
297      mov ah,02h
298      int 21h
299      ret
300  gameover:
301      mov dh,5
302      mov dl,20
303      mov bh,0
304      mov ah,2h
305      int 10h          ;set cursor position DH = row.DL = column.BH = page number (0..7).
306
307      PRINT 'GAMEOVER'
308      jmp close
309
310  cong:
311      mov dh,5
312      mov dl,20
313      mov bh,0
314      mov ah,2h
315      int 10h          ;set cursor position DH = row.DL = column.BH = page number (0..7).
316
317      PRINT 'CONGRATULATIONS YOU GUESSED THE WORD'
318      jmp close
319
```

These functions collectively will draw the figure of hangman

```
053  movies:
054      MOV AH, 00h  ; interrupts to get system time
055      INT 1AH      ; CX:DX now hold number of clock ticks since midnight
056      mov  ax, dx
057      xor  dx, dx
058      mov  cx, 5
059      div  cx      ; here dx contains the remainder of the division - from 0 to 4
060      add  dl, '0' ; to ascii from '0' to '4'
061
062      PRINTN '*Random number choosen:'
063      mov ah, 2h   ; call interrupt to display a value in DL
064      int 21h
065
066      mov bl,dl
067      sub bl,30h   ; now the bl value is from 0h to 4h
068      PRINTN ''
069
070      PRINTN '*Word selected:'
071      cmp bl,00h
072      je movie_first
073      cmp bl,01h
074      je movie_second
075      cmp bl,02h
076      je movie_third
077      cmp bl,03h
078      je movie_fourth
079      cmp bl,04h
080      je movie_fifth
081
082  movie_first:
083      lea di,movie_word[0]
084      mov cx,8
085      mov dx,di
086      call printit
087      jmp getchar
088  movie_second:
089      lea di,movie_word[9]
090      mov cx,5
091      mov dx,di
092      call printit
093      jmp getchar
094  movie_third:
095      lea di,movie_word[15]
096      mov cx,5
097      mov dx,di
098      call printit
099      jmp getchar
100  movie_fourth:
101      lea di,movie_word[21]
102      mov cx,6
103      mov dx,di
104      call printit
105      jmp getchar
106  movie_fifth:
107      lea di,movie_word[28]
108      mov cx,9
109      mov dx,di
110      call printit
```

The check function checks if the letter given by player comes twice, in that case the loop must not terminate and replace the other dash with the letter also. This function for example is used for words like inception, prestige or tenet.

N_found function will check how many lives are remaining and according print the hangman.

```asm
227
228  check:
229      mov bl,[si]
230      inc pos
231      inc si
232      cmp al,bl
233      je foundit
234      loop check
235
236      cmp gotit,0
237      je nfound
238      mov gotit,0
239
240      jmp goagain
241
242  nfound:
243      dec lives
244      call printlife
245
246      cmp lives,52    ;52->4
247      je rope
248
249      cmp lives,51
250      je head
251
252      cmp lives,50
253      je body
254
255      cmp lives,49
256      je hands
257
258      cmp lives,48
259      je legs
260
261      jmp goagain
262
263  foundit:
264      ;PRININ "found"
265      mov gotit,1
266      inc correct
267      mov dh,7
268      mov dl,18
269      add dl,pos
270      add dl,pos
271      mov bh,0
272      mov ah,2h
273      int 10h          ;set cursor position DH = row.DL = column.BH = page number (0..7).
274
275      mov dl,al        ;printing character
276      mov ah,02h
277      int 21h
278
279      mov dx,len
280      cmp dx,correct
281      je cong
282
283      jmp check
284
```

Plagiarism Report

include 'emu8086.inc' data segment movie_word db "prestige$","venom$","tenet$" ,"avatar$","inception$" city_word db "paris$","london$","bangkok$","dubai$","rome$" len dw 0 pos db 0 correct dw 0 gotit db 0 lives dw 53 ;53->5 ends code segment start: mov ax,data mov ds,ax mov dh,2 call setcursor PRINTN 'Hangman game development using Emu8086' inc dh call setcursor PRINTN 'Press 1 to start playing' inc dh call setcursor mov ah,01h ; getting a input from user which will be stores in AL int 21h PRINTN " cmp al,31h ; ascii value of character 1 is 31h je gameon jmp close gameon: inc dh call setcursor PRINTN 'Choose your GENRE' inc dh call setcursor PRINTN '1.Hollywood movie name' inc dh call setcursor PRINTN '2.Popular city name' inc dh call setcursor mov ah,01h int 21h PRINTN " cmp al,31h je movies cmp al,32h je city movies: MOV AH, 00h ; interrupts to get system time INT 1AH : CX:DX now hold number of clock ticks since midnight mov ax, dx xor dx, dx mov cx, 5 div cx ; here dx contains the remainder of the division - from 0 to 4 add dl, '0' ; to ascii from '0' to '4' PRINTN "Random number choosen.' mov ah, 2h ; call interrupt to display a value in DL int 21h mov bl,dl sub bl,30h ; now the bl value is from 0h to 4h PRINTN " PRINTN "Word selected:' cmp bl,00h je movie_first cmp bl,01h je movie_second cmp bl,02h je movie_third cmp bl,03h je movie_fourth cmp bl,04h je movie_fifth movie_first: lea di,movie_word[0] mov cx,8 mov dx,di call printit jmp getchar movie_second: lea di,movie_word[9] mov cx,5 mov dx,di call printit jmp getchar movie_third: lea di,movie_word[15] mov cx,5 mov dx,di call printit jmp getchar movie_fourth: lea di,movie_word[21] mov cx,6 mov dx,di call printit jmp getchar movie_fifth: lea di,movie_word[28] mov cx,9 mov dx,di call printit jmp getchar PRINTN " PRINTN "Word selected:' cmp bl,00h je city_first cmp bl,01h je city_second cmp bl,02h je city_third cmp bl,03h je city_fourth cmp bl,04h je city_fifth city_first: lea di,city_word[0] mov cx,5 mov dx,di call printit jmp getchar city_second: lea di,city_word[6] mov cx,6 mov dx,di call printit jmp getchar city_third: lea di,city_word[13] mov cx,7 mov dx,di call printit jmp getchar city_fourth: lea di,city_word[21] mov cx,5 mov dx,di call printit jmp getchar city_fifth: lea di,city_word[27] mov cx,4 mov dx,di call printit jmp getchar ;;;;;;;;;;;;;;;;;;;;rev -3;;;;;;;;;;;;;;;;;;;;; printit: mov ah,09h int 21h ret setcursor: mov dl,20 mov bh,0 mov ah,2h int 10h ;set cursor position DH = row.DL = column.BH = page number (0..7). ret getchar: mov len,cx mov al, 02h mov ah, 0 int 10h ;clear screen call rod welcome: mov dh,2 mov dl,20 mov bh,0 mov ah,2h int 10h ;set cursor position DH = row.DL = column.BH = page number (0..7). PRINT 'HANGMAN GAME USING EMU8086' call printlife underscores: mov dh,8 mov dl,20 mov bh,0 mov ah,2h int 10h ;set cursor position DH = row.DL = column.BH = page number (0..7). mov cx,len printunderscores: PRINT '_' inc dl loop printunderscores beg: mov ah,07h ;get input without echo int 21h mov ah,00h lea si,d mov cx,len mov pos,0 check: mov bl,[si] inc pos inc si cmp al,bl je foundit loop check cmp gotit,0 je nfound mov gotit,0 jmp goagain nfound: dec lives call printlife cmp lives,52 ;52->4 je rope cmp lives,51 je head cmp lives,50 je body cmp lives,49 je hands cmp lives,48 je legs jmp goagain foundit: ;PRINTN "found" mov gotit,1 inc correct mov dh,7 mov di,18 add di,pos add dl,pos mov bh,0 mov ah,2h int 10h ;set cursor position DH = row.DL = column.BH = page number (0..7). mov dl,al ;printing character mov ah,02h int 21h mov dx,len cmp dx,correct je cong jmp check goagain: jmp beg printlife: mov dh,5 mov dl,20 mov bh,0 mov ah,2h int 10h ;set cursor position DH = row.DL = column.BH = page number (0..7). PRINT 'LIVES:' mov dx,lives ;printing lives mov ah,02h int 21h ret gameover: mov dh,5 mov dl,20 mov bh,0 mov ah,2h int 10h ;set cursor position DH = row.DL = column.BH = page number (0..7). PRINT 'GAMEOVER' jmp close cong: mov dh,5 mov dl,20 mov bh,0 mov ah,2h int 10h ;set cursor position DH = row.DL = column.BH = page number (0..7). PRINT 'CONGRATULATIONS YOU GUESSED THE WORD' jmp close rod: mov dh,5 mov dl,5 mov bh,0 mov ah,2h int 10h rodtop: PRINT "-" inc dl cmp dl,12 jb rodtop mov dh,5 rodleft: inc dh mov dl,5 mov bh,0 mov ah,2h int 10h PRINTN "|" cmp dh,18 jb rodleft ret rope: mov dh,5 drawrope: inc dh mov dl,10 mov bh,0 mov ah,2h int 10h PRINTN "|" cmp dh,8 jb drawrope jmp goagain head: mov dh,8 mov dl,10 mov bh,0 mov ah,2h int 10h PRINT "-" inc dh mov dl,9 drawhead: mov bh,0 mov ah,2h int 10h PRINT "|" inc dl inc dl cmp dl,12 jb drawhead inc dh mov dl,10 mov bh,0 mov ah,2h int 10h PRINT "-" jmp goagain body: mov