
VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

# SCHOOL OF COMPUTER SCIENCE ENGINEERING

## Hangman Game Development using Emu8086

By:

**Divyanshu Gupta (19BCE2093)**

## CSE2006 – Microprocessor & Interfacing
## Fall Semester 2021-22

Submitted to

**Faculty: Dr. Ragunath G**
**Date: 07/12/2021**
**Slot: E1 + TE1**

## Table of Contents

## 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
   Journal - Advances in Interconnect Technologies: An International Journal (AITIJ)

   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
   SIGCSE '11: Proceedings of the 42nd ACM technical symposium on Computer
   March 2011

   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 frequencies 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.

## Drawbacks in existing work

- Only limited number of words and genres can be added. The user experience and greatly improve if the code can contain lot of words and genres but which would take more space and increase the complexity, thus increasing the processing time.

- Existing used various platforms for interface for hangman which used much more disk space than we are using.

- Complex code containing lot of pointers and functions makes the program difficult to extend and roll new features.

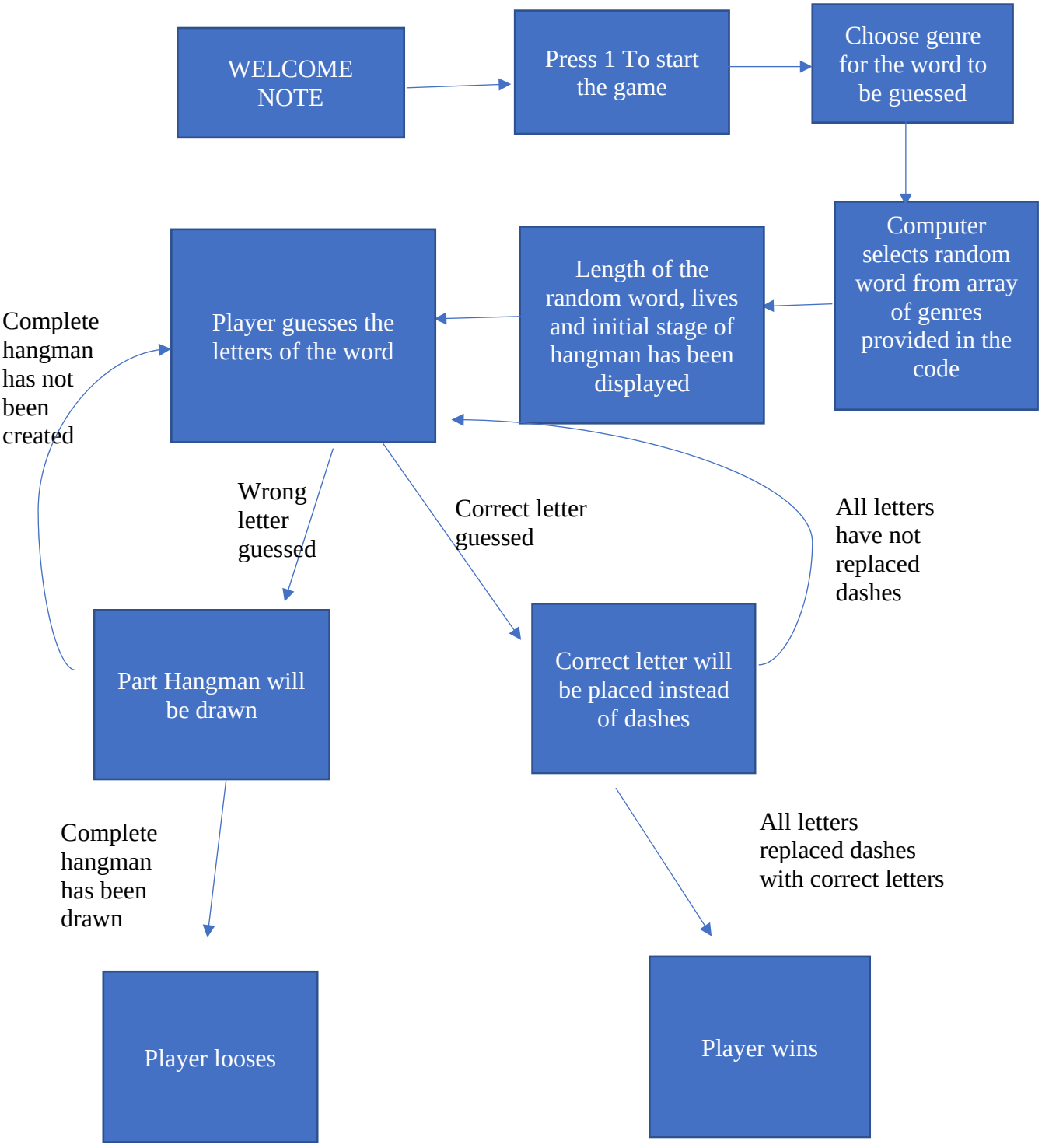
- Limited hints provided to the user mainly the genre and number of letters in the word. The user experience could increase if the interface can have more hints and interactive ways to provide them to the user.

- A function by which the user can decide difficulty and adjust it according to his or her taste and skills. The functionality will greatly improve upon how the player wants to play the game.

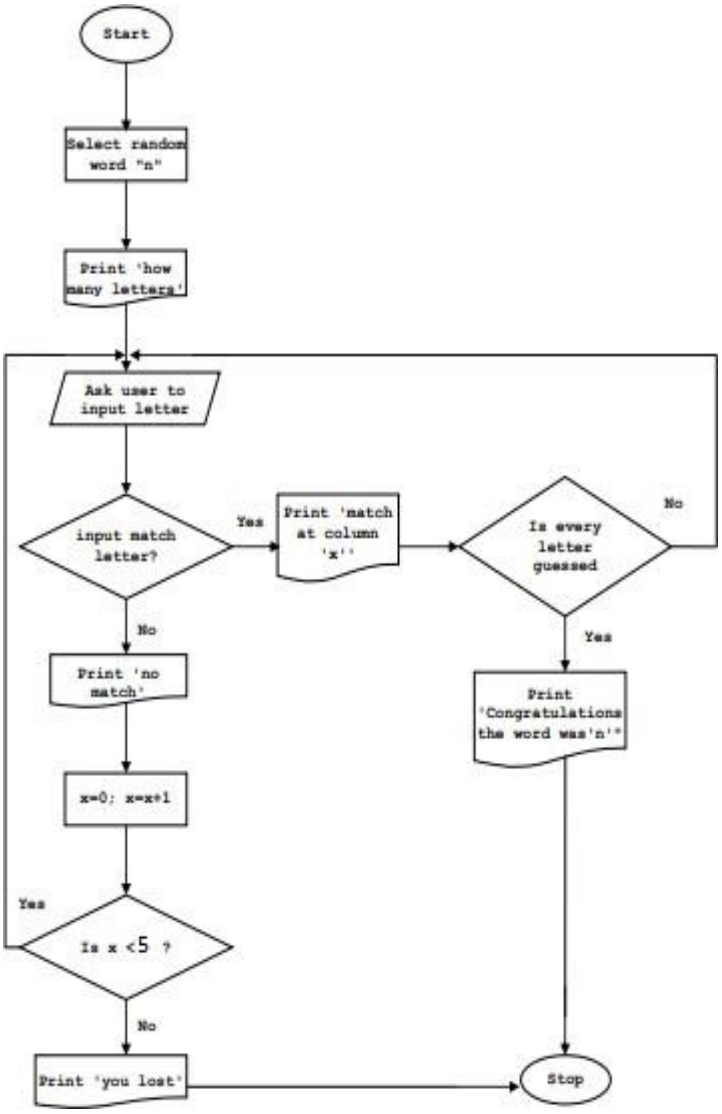
## Proposed work

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.

## Block Diagram



## Flow Chart for Algorithm



## 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

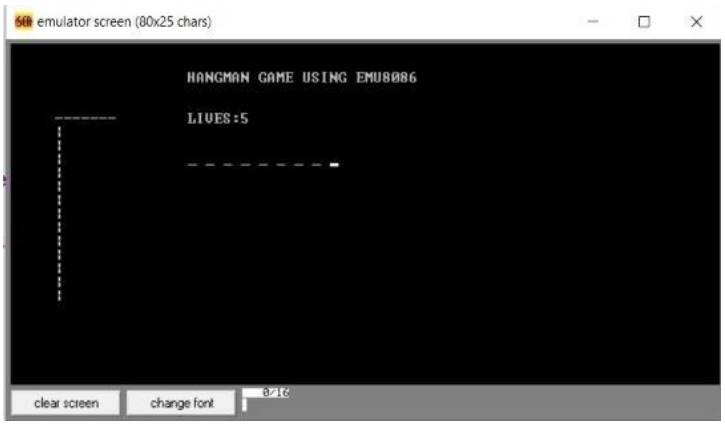
**Ret** - Returns program execution from a procedure (subroutine) to the next instruction or main program

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

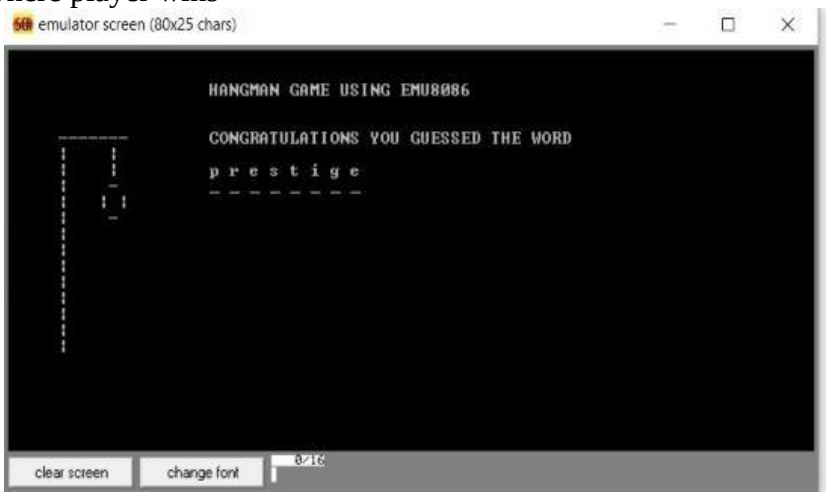
## Screenshot of prototype

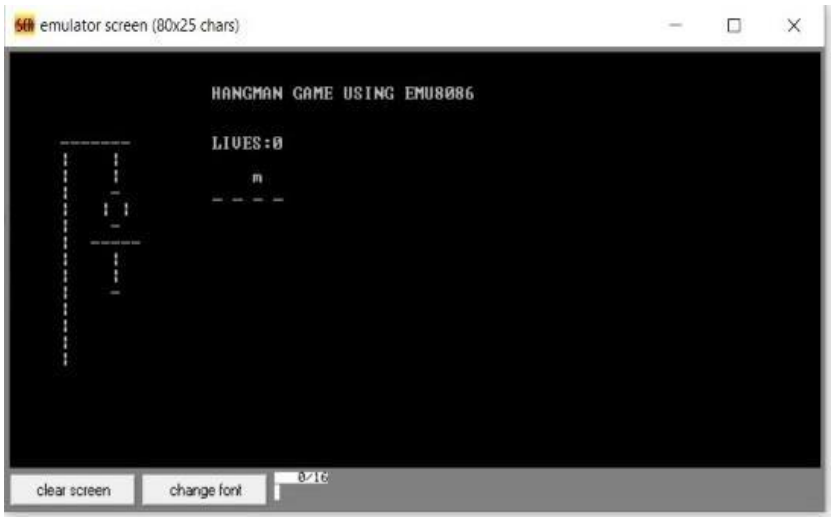
In this screen the user will get a welcome note and further he/she has to choose the genre, according to which the code will choose a random word to be guessed.


User has to guess a random word picked by code according to the genre choosed.

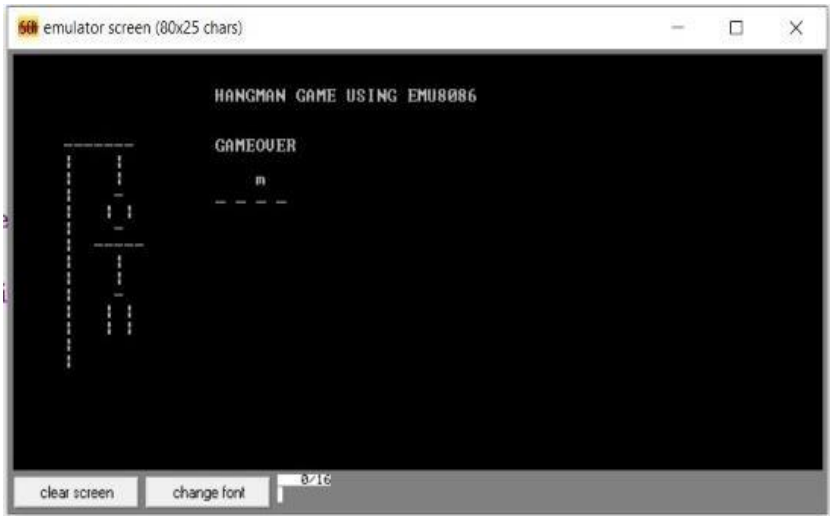


The case where player wins




The case where player looses



## 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.

## Individual contribution:

**Divyanshu Gupta:** Coding the drawing of hangman graphic, randomize function coding part for selection of words, integration of hangman graphic along with lives left, coding part if the letters entered by player are not in the word (nfound function), coding of integration of check and traversal of array functions, documentation and ppts.

**Shubham Kaushik:** Generation of array part and genre selection, coding of part if letter entered by the user comes twice in the word selected, coding of beginning function, coding of lose and win function, coding of integration of check and traversal of array functions, coding of setcursor and get_corrected function to determine the initial position of displaying output. documentation and ppts.

**Royal:** Hangman graphic Coding the drawing of hangman graphic, randomize function coding part for selection of words, integration of hangman graphic along with lives left, coding part if the letters entered by player are not in the word (nfound function), coding of integration of check and traversal of array functions, documentation and ppts.

## References


1. https://en.wikipedia.org/wiki/Hangman_(game)
2. Carter, Paul A. PC Assembly Language. Lulu.com, 2007.
3. Irvine, Kip R. Assembly language for Intel-based computers. Prentice Hall, 2003
4. Prokin, M. Računarska elektronika. Akademska misao, 2005
5. Hyde, R. The art of assembly language. No Starch Press, 2010.
6. http://kipirvine.com/asm/index6th.htm
7. http://www.asmirvine.com/
8. http://programming.msjc.edu/asm/help/index.html?page=source%2 Fabout.htm
9. Kuo, Sen M., Bob H. Lee, and Wenshun Tian. Real-time digital signal processing: fundamentals, implementations and applications. John Wiley & Sons, 2013.
10. S. Tešanović and P. Mitrović, Development of the Game Hangman in Assembly Programming Language, 2017 25th Telecommunications Forum (TELFOR), Belgrade, 2017, pp. 912- 915


.

## 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.

```
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
```

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

```
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
```

These functions collectively will draw the figure of hangman

```
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
```

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.

```
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 dl,18
   add dl,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
```