8086 ASSEMBLY MONOALPHABETIC SUBSTITUTION ENCRYPTION SYSTEM

MINOR PROJECT REPORT

By

SRIJONI CHOWDHURY (RA221002610210)
SAYAL SINGH (RA2211002610218)
MOHMMAD BASITH KALAM RIZVI (RA2211026010219)

Under the guidance of

Dr.R.A.Karthika

In partial fulfilment for the Course

of

21CSS201T– Computer Organization and Architecture

in the Department of Computational Intelligence

FACULTY OF ENGINEERING AND TECHNOLOGY

SCHOOL OF COMPUTING

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR

NOVEMBER 2023

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

(Under Section 3 of UGC Act, 1956)

BONAFIDE CERTIFICATE

Certified that this minor project report for the course **21CSS201T– Computer Organization and Architecture** entitled in "**Monoalphabetic Substitution Encryption System using 8086 Assembly Language**" is the bonafide work of **Srijoni Chowdhury(RA2211026010210)** , **Sayal Singh(RA2211026010218) and Rizvi(RA2211026010219)** who carried out the work under my supervision.

SIGNATURE	**SIGNATURE**
Dr.R.A.Karthika	Dr Annie Uthra R
COA– Course Faculty	**Head of the Department**
Assistant Professor	Professor
Department of Computational Intelligence	Department of Computational Intelligence
SRM Institute of Science and Technology	SRM Institute of Science and Technology
Kattankulathur	Kattankulathur

ABSTRACT

The X86 Assembly Monoalphabetic Substitution Encryption System is a low-level, hardware-dependent cryptographic software designed to secure data through the implementation of a monoalphabetic substitution cipher at the assembly language level on x86 architecture. This system aims to provide a basic yet illustrative example of encryption techniques, offering a fundamental understanding of how encryption works at the machine code level.

The system leverages the x86 assembly language to perform encryption operations, replacing each character in the plaintext with a corresponding character from a predefined substitution key. The substitution key is a mapping of characters from the plaintext alphabet to the ciphertext alphabet, ensuring that each character is replaced consistently. This system relies on the monoalphabetic substitution method, which makes it suitable for educational purposes and limited security applications.

ACKNOWLEDGEMENT

We express our heartfelt thanks to our honorable **Vice Chancellor Dr. C. MUTHAMIZHCHELVAN**, for being the beacon in all our endeavors.

We would like to express my warmth of gratitude to our **Registrar Dr. S. Ponnusamy,** for his encouragement.

We express our profound gratitude to our **Dean (College of Engineering and Technology) Dr. T. V.Gopal,** for bringing out novelty in all executions.

We would like to express my heartfelt thanks to Chairperson, School of Computing **Dr. Revathi Venkataraman,** for imparting confidence to complete my course project

We wish to express my sincere thanks to **Course Audit Professors Dr. C. Malathy,** Professor, Department of Networking and Communication and Course Coordinator **Dr. R.A.Karthika,** Assistant Professor, Department Computational Intelligence for their constant encouragement and support.

We are highly thankful to our Course project Faculty **Dr.R.A.Karthika,** Assistant Professor, Department Computational Intelligence for her assistance, timely suggestion and guidance throughout the duration of this course project.

We extend my gratitude to our **Head of the Department, Dr.R.Annie Uthra, Professor, Department of Computational Intelligence** and my Departmental colleagues for their Support.

Finally, we thank our parents and friends near and dear ones who directly and indirectly contributed to the successful completion of our project. Above all, I thank the almighty for showering his blessings on me to complete my Course project.

TABLE OF CONTENTS

1.INTRODUCTION

1.1Objective

- Low-Level Programming Skills: It provides an opportunity to learn and practice assembly language programming, a valuable skill for understanding and working with computer hardware and embedded systems.

- Cryptography Understanding: Through the implementation of a monoalphabetic substitution cipher, the project helps users comprehend fundamental cryptographic principles. It illustrates the concept of character substitution, which is a core component of many encryption algorithms, and fosters an appreciation for the nuances of secure data communication.

- Algorithm Development: This program encourages users to develop, modify, or enhance the encryption system's algorithm and substitution key. This allows for experimentation and exploration of various aspects of cryptographic design and implementation, fostering creativity and problem-solving skills.

- Source Code Exploration: Users can explore the project's source code to gain insights into how characters are substituted between plaintext and ciphertext. This provides a practical example of how encryption algorithms are coded and implemented in low-level programming languages.

- Security Awareness: While the monoalphabetic substitution cipher is not suitable for secure, real-world applications, the project highlights its weaknesses and reinforces the importance of using stronger encryption techniques for sensitive data. It raises awareness about the limitations and vulnerabilities of simple ciphers.

- Foundation for Further Study: The project can serve as a foundation for further exploration of cryptography and assembly language programming. Users who are interested in more advanced cryptographic concepts or in-depth x86 assembly programming can build upon the knowledge and experience gained from this project.

1.2Introduction

The "8086 Assembly Monoalphabetic Substitution Encryption System" is a fascinating exploration of the convergence of low-level programming and the realm of cryptography. This project delves into the intricate world of encryption, employing 8086 assembly language to implement a monoalphabetic substitution cipher—a fundamental encryption technique. The endeavor is designed to serve as an educational endeavor and a practical exercise, offering enthusiasts and aspiring programmers a unique opportunity to understand the inner workings of encryption at the machine code level.

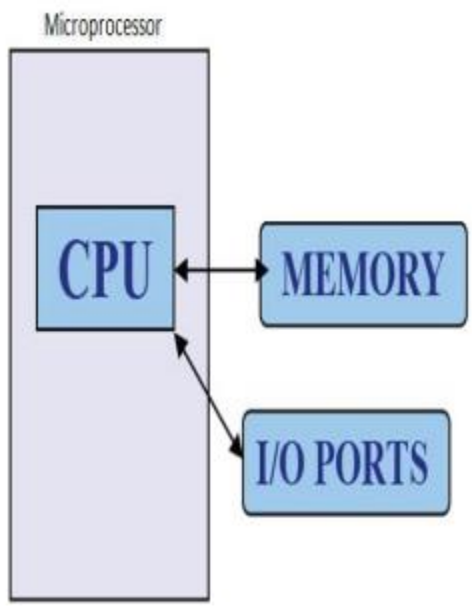
2. SOFTWARE AND HARDWARE REQUIREMENT

- x86 emulator emu8086: EMU8086 is a popular and user-friendly integrated development environment (IDE) and emulator for the x86 architecture assembly language. It is primarily used for programming and testing assembly language code for Intel 8086 and 8088 microprocessors, which are early microprocessor models that were significant in the history of computing.

- Intel 8086 Assembly Language: Intel 8086 assembly language is a low-level programming language used for programming the Intel 8086 and Intel 8088 microprocessors. These processors were part of the x86 family and played a crucial role in the early history of personal computing. Intel 8086 assembly language is known for its close-to-hardware nature, allowing programmers to have precise control over the processor's operations. It's commonly used for educational purposes, for understanding computer architecture, and for tasks that require precise control over hardware. Despite its age, it remains a foundation for understanding the principles of assembly language and low-level programming.

3. CONCEPT/WORKING PRINCIPLE:

Microprocessor: is a CPU on a single chip. Third Generation (16-bit Microprocessor) were introduced in 1978.The 16 - bit processors with a performance like minicomputers. Third Generation Microprocessors can be represented by Intel's 8086, Zilog Z800 and 80286.



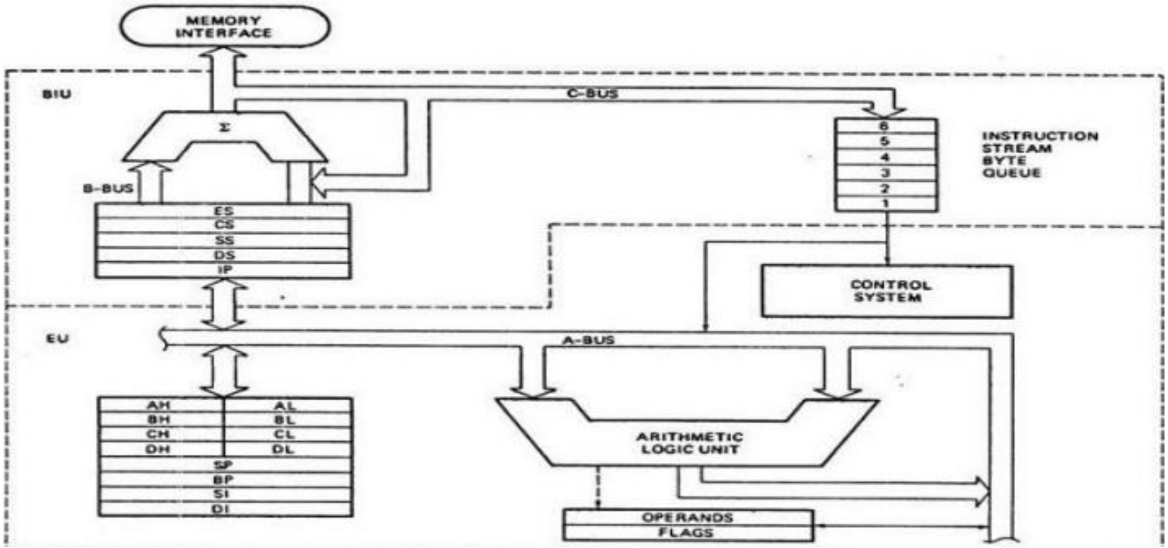
Diag1 - Structure of Microprocessor

8086 Microprocessor

The 8086 microprocessor is an 8-bit/16-bit microprocessor designed by Intel in the late 1970s. It is the first member of the x86 family of microprocessors, which includes many popular CPUs used in personal computers.

The architecture of the 8086 microprocessor is based on a complex instruction set computer (CISC) architecture, which means that it supports a wide range of instructions, many of which can perform multiple operations in a single instruction. The 8086 microprocessor has a 20-bit address bus, which can address up to 1 MB of memory, and a 16-bit data bus, which can transfer data between the microprocessor and memory or I/O devices.

The 8086 microprocessor has a segmented memory architecture, which means that memory is divided into segments that are addressed using both a segment registers and an offset. The segment register points to the start of a segment, while the offset specifies the location of a specific byte within the segment. This allows the 8086 microprocessors to access large amounts of memory, while still using a 16-bit data bus.



Diag2 : Architecture of 8086 Microprocessor

Memory is divided into various sections called segments.

- Code segment : where you store the program.
- Data segment : where the data is stored.
- Extra segment : mostly used for string operations.
- Stack segment : used to push/pop.

General purpose registers are used to store temporary data within the microprocessor.

AX,BX,CX,DX are the General-Purpose Registers. These are 16-bit registers. They are divided into 8bit registers. These registers perform operations on 8-bit instructions.

Microprocessors also consist of Pointer Registers, Index Registers and Segment Registers.

Encryption &Decryption

Encryption: Encryption is a way of scrambling data so that only authorized parties can understand the information. In technical terms, it is the process of converting human-readable plaintext to incomprehensible text, also known as ciphertext. In simpler terms, encryption takes readable data and alters it so that it appears random. Encryption requires the use of a cryptographic key: a set of mathematical values that both the sender and the recipient of an encrypted message agree on.

Decryption: It is the process of transforming data that has been rendered unreadable through encryption back to its unencrypted form. In decryption, the system extracts and converts the garbled data and transforms it to texts and images that are easily understandable not only by the reader but also by the system. Decryption may be accomplished manually or automatically.

4. APPROCH

Encryption can be done in various ways. One of the simplest methods is monoalphabetic cipher. A monoalphabetic cipher is a substitution cipher where each letter of the plain text is replaced with another letter of the alphabet. It uses a fixed key which consists of the 26 letters of a “shuffled alphabet”.

This program implements this method to encrypt/decrypt a string of characters.

1. This program uses the following table as the reference for character substitution.

Plain text	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
Cipher text	q	w	e	r	t	y	u	i	o	p	a	s	d	f	g	h	j	k	l	z	x	c	v	b	n	m

2. The program handles lowercase and uppercase characters.
3. This program omits all spaces in the result.

4.1 PROGRAM

```
ORG 0100H

JMP start.

newline EQU 0AH   ; \n
cret EQU 0DH   ; \r
bcksp EQU 08H   ; \b
;
; Hard-coded string:
;
hardcoded_string DB hi there! This is an encrypted message', cret, newline, '$' input_string DB 259
dup ('$') ; Reserved area for input string (256 chars + \r \n $)
; Messages and dialogues to be displayed in the UI:
;
message_welcome DB newline, 'Welcome to the monoalphabetic encryption system! ', cret,
newline
                DB     'Please choose how you wish to proceed:', cret, newline
                DB     '1- Enter string as input (max: 256 chars)', cret, newline
                DB     '2- Use hard-coded string', cret, newline, '$'

message_using_hc      DB     '================================', cret, newline
                DB     'USING YOUR HARDCODED STRING' , cret, newline
                DB     '================================', cret, newline, '$'

message_using_input    DB     '================================', cret, newline
                DB     'Please enter your string below' , cret, newline
                DB     '================================', cret, newline, '$'

message_try_again      DB     cret, newline, 'Give it one more try? (y/n)', cret, newline, '$'
message_press_key      DB     'Press any key to exit...$'
message_display_org    DB     cret, newline, 'Your original string: $'
message_display_enc    DB     cret, 'Encrypted message: $'
message_display_dec    DB     cret, 'Decrypted message: $'
message_encrypting     DB     'Encrypting...$'
message_decrypting     DB     'Decrypting...$'
; Just for reference --------------------->  'abcdefghijklmnopqrstuvwxyz'
encryption_table_lower DB      97 dup (' '), 'qwertyuiopasdfghjklzxcvbnm'
decryption_table_lower DB      97 dup (' '), 'kxvmcnophqrszyijadlegwbuft'
; We leave 97(61H) blank spaces before the start of the table
; as the ASCII value of 'a' = 61H
encryption_table_upper DB 65 dup (' '), 'QWERTYUIOPASDFGHJKLZXCVBNM'
```

```
decryption_table_upper DB      65 dup (' '), 'KXVMCNOPHQRSZYIJADLEGWBUFT'
; We leave 65(41H) blank spaces before the start of the table
; as the ASCII value of 'A' = 41H
start:          LEA     DX, message_welcome
                MOV     AH, 09
                INT     21H
                MOV     AH, 0
                INT     16H
                CMP     AL, '2'       ; User chose to use the hardcoded string
                JE      use_hc
                CMP     AL, '1'       ; User chose to enter a string
                JNE     start
                CALL    get_input
                JMP     start_process

use_hc:         LEA     DX, message_using_hc
                MOV     AH, 09
                INT     21H
                LEA     SI, hardcoded_string


start_process:
; Display original string
                LEA     DX, message_display_org
                MOV     AH, 09         ; value of AH is adjusted as operation of int 21H depends on its value
                INT     21H            ; at AH = 09, int 21H outputs string at DS:DX
                LEA     DX, SI
                MOV     AH, 09
                INT     21H
; Encrypt:
                LEA     DX, message_encrypting   ; Display message
                MOV     AH, 09
                INT     21H
                MOV     AH, 1          ; value of AH is adjusted as operation of encrypt_decrypt depends on its value
                CALL    encrypt_decrypt ; AH = 1 means monoalphabetic encryption, 0 means monoalphabetic decryption, else do nothing
; Display result on the screen:
                LEA     DX, message_display_enc
                MOV     AH, 09         ; value of AH is adjusted as operation of int 21H depends on its value
                INT     21H            ; at AH = 09, int 21H outputs string at DS:DX
                LEA     DX, SI
                MOV     AH, 09
                INT     21H
; Decrypt:
                LEA     DX, message_decrypting    ; Display message
                MOV     AH, 09
                INT     21H
                MOV     AH, 0          ; AH = 0 means monoalphabetic decryption
                CALL    encrypt_decrypt
```

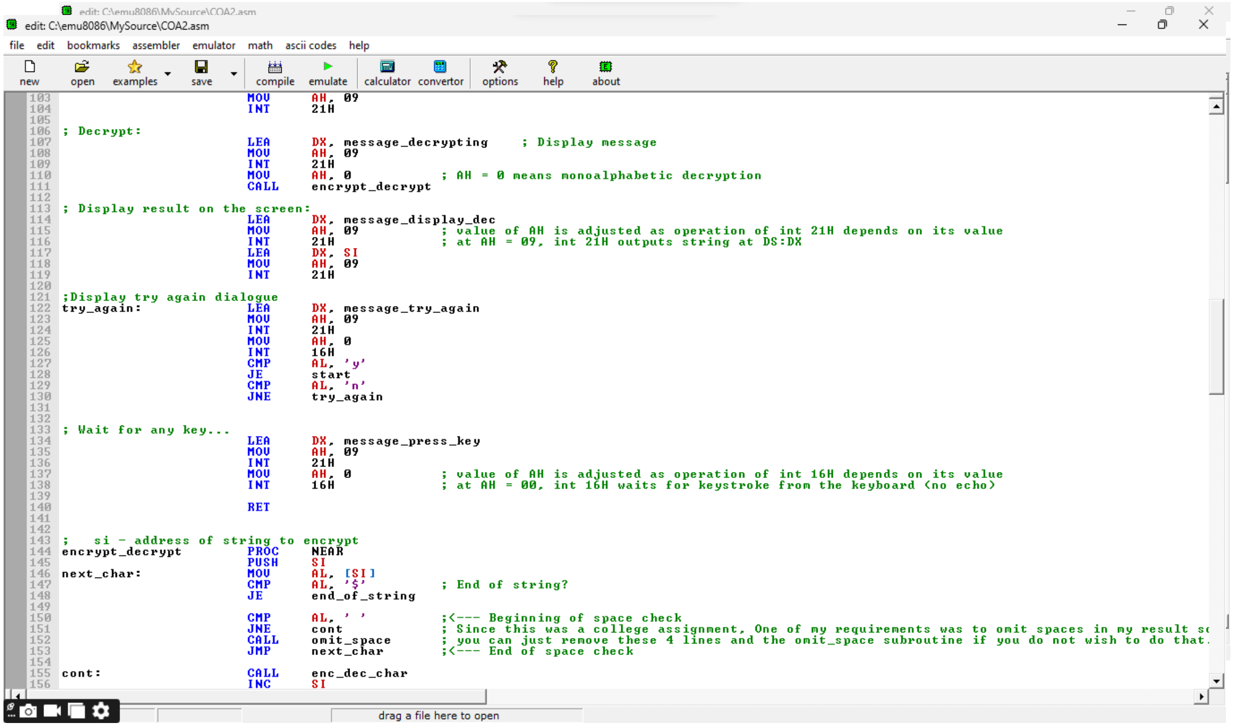
Display result on the screen:

```
                LEA    DX, message_display_dec
                MOV    AH, 09       ; value of AH is adjusted as operation of int 21H depends on
its value
                INT    21H          ; at AH = 09, int 21H outputs string at DS:DX
                LEA    DX, SI
                MOV    AH, 09
                INT    21H
;Display try again dialogue
try_again:      LEA    DX, message_try_again
                MOV    AH, 09
                INT    21H
                MOV    AH, 0
                INT    16H
                CMP    AL, 'y'
                JE     start
                CMP    AL, 'n'
                JNE    try_again



; Wait for any key...
                LEA    DX, message_press_key
                MOV    AH, 09
                INT    21H
                MOV    AH, 0        ; value of AH is adjusted as operation of int 16H depends on
its value
                INT    16H          ; at AH = 00, int 16H waits for keystroke from the keyboard (no
echo)
                RET
    ;   si - address of string to encrypt
encrypt_decrypt     PROC    NEAR
                PUSH   SI
next_char:      MOV AL, [SI]
                CMP  AL, '$'        ; End of string?
                JE  end_of_string
                CMP    AL, ' '      ;<--- Beginning of space check
                JNE    cont         ; Since this was a college assignment, One of my requirements
was to omit spaces in my result so
                CALL   omit_space   ; you can just remove these 4 lines and the omit_space
subroutine if you do not wish to do that.
                JMP    next_char    ;<--- End of space check

cont:           CALL   enc_dec_char
                INC    SI
                JMP    next_char
end_of_string:  POP    SI
                RET
encrypt_decrypt     ENDP

; Subroutine to send space to the end of the string (after '$')
; You can delete this subroutine if you do not need it
;
```

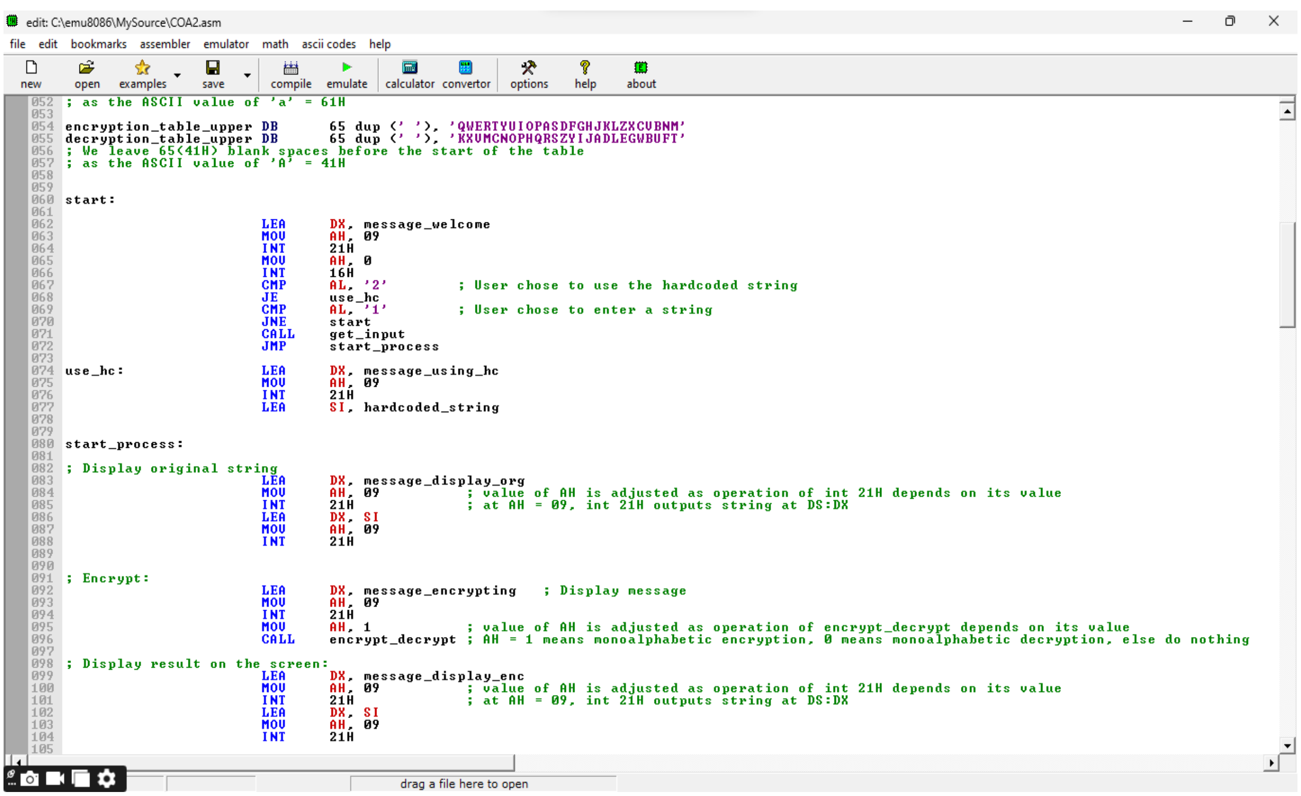
```
omit_space          PROC    NEAR
                PUSH    SI              ; The reason I send the space after the '$'
                                        ; is to handle several consecutive spaces without
omit_space_loop:    MOV     AL, [SI+1]      ; entering an inifnite loop as opposed to just swapping
                MOV     [SI+1], ' '     ; the ' ' character with the character after it
                MOV     [SI], AL
                INC     SI
                CMP     [SI-1], '$'
                JNE     omit_space_loop
                POP     SI
                RET
omit_space          ENDP



; Subroutine to convert character with the
; appropriate table (encrypt/decrypt)(uppercase/lowercase)
enc_dec_char        PROC    NEAR
                PUSH    BX
                CMP     AL, 'a'
                JB      check_upper_char
                CMP     AL, 'z'
                JA      skip_char
                CMP     AH, 1           ; AH = 1 means monoalphabetic encryption, since we are
working with lower case, we use encryption_table_lower
                JE      encrypt_lower_char
                CMP     AH, 0           ; AH = 0 means monoalphabetic decryption, since we are
working with lower case, we use decryption_table_lower
                JNE     skip_char
                LEA     BX, decryption_table_lower
                JMP     translate_char
encrypt_lower_char:     LEA     BX, encryption_table_lower
                    JMP     translate_char
check_upper_char:   CMP     AL, 'A'
                JB      skip_char
                CMP     AL, 'Z'
                JA      skip_char
                CMP     AH, 1           ; AH = 1 means monoalphabetic encryption, since we are
working with upper case, we use encryption_table_upper
                    JE      encrypt_upper_char
                    CMP     AH, 0       ; AH = 0 means monoalphabetic decryption, since we are
working with upper case, we use decryption_table_upper
                JNE     skip_char
                LEA     BX, decryption_table_upper
                JMP     translate_char
encrypt_upper_char:     LEA     BX, encryption_table_upper
translate_char:     XLATB
                    MOV     [SI], AL
skip_char:          POP     BX
                RET
enc_dec_char        ENDP
```

```
; Subroutine to take input string from the user
; The subroutine handles if the user presses backspace: delete char + inc CX
; The subroutine allows the user to enter a maximum of 256 chars
get_input          PROC    NEAR
                   LEA     DX, message_using_input
                   MOV     AH, 09
                   INT     21H
                   LEA     SI, input_string
                   MOV     AH, 1
                   MOV     CX, 255       ; To set a cap for string input to 256 chars
                   JMP     input_loop
backspace_entered:    INC    CX          ; Increment CX in case user presses backspace as a character is deleted
input_loop:        INT     21H
                   MOV     [SI], AL
                   CMP     AL, bcksp
                   JNE     cont_input
                   CMP     SI, offset input_string
                   JE      input_loop    ;If the string is empty just loop again without affecting SI and without incrementing CX
                   MOV     [SI], ' '
                   CALL    omit_space
                   DEC     SI
                   JMP     backspace_entered
cont_input:        INC     SI
                   CMP     AL, cret
                   JE      terminate_string
                   LOOP    input_loop    ; LOOP instead of JMP to incorporate (CX != 0000H) as a jump condition
terminate_string:  MOV     [SI-1], cret
                   MOV     [SI], newline
                   MOV     [SI+1], '$'
                   LEA     SI, input_string
                   RET
get_input          ENDP

end
```

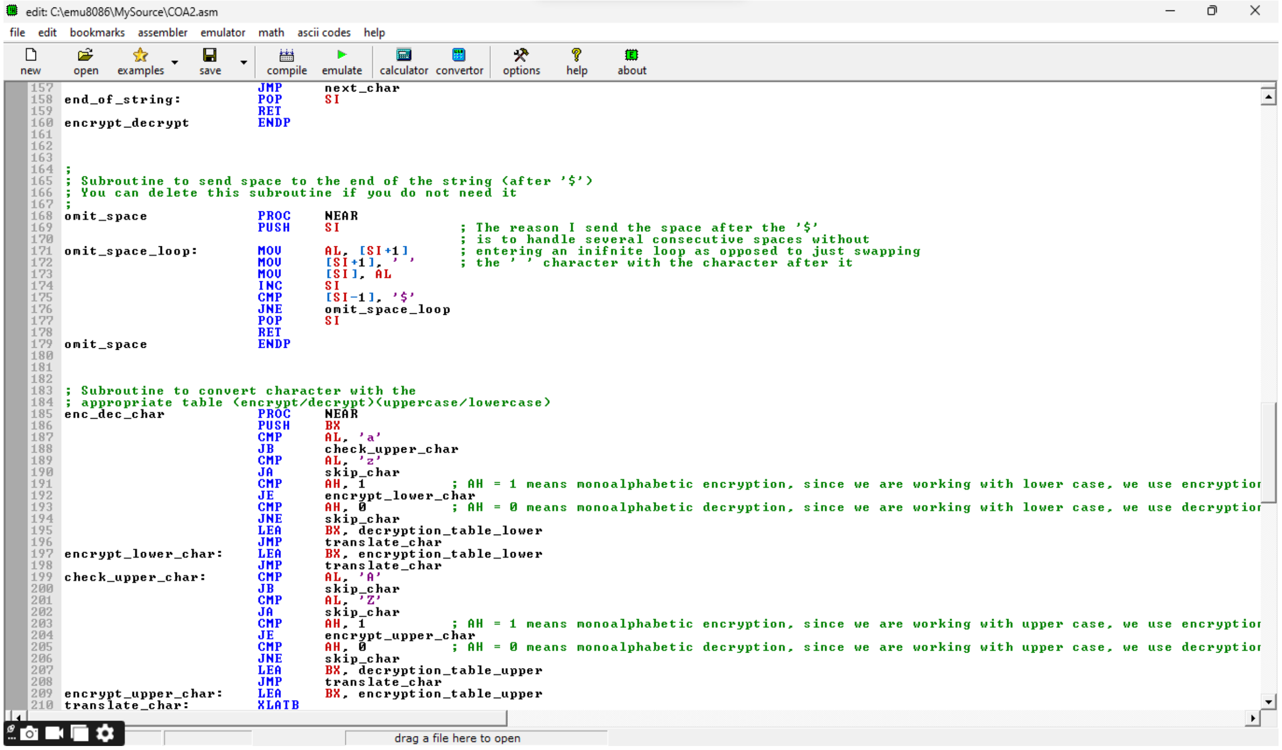
Screenshots of code



Diag3-Code running in emu8086

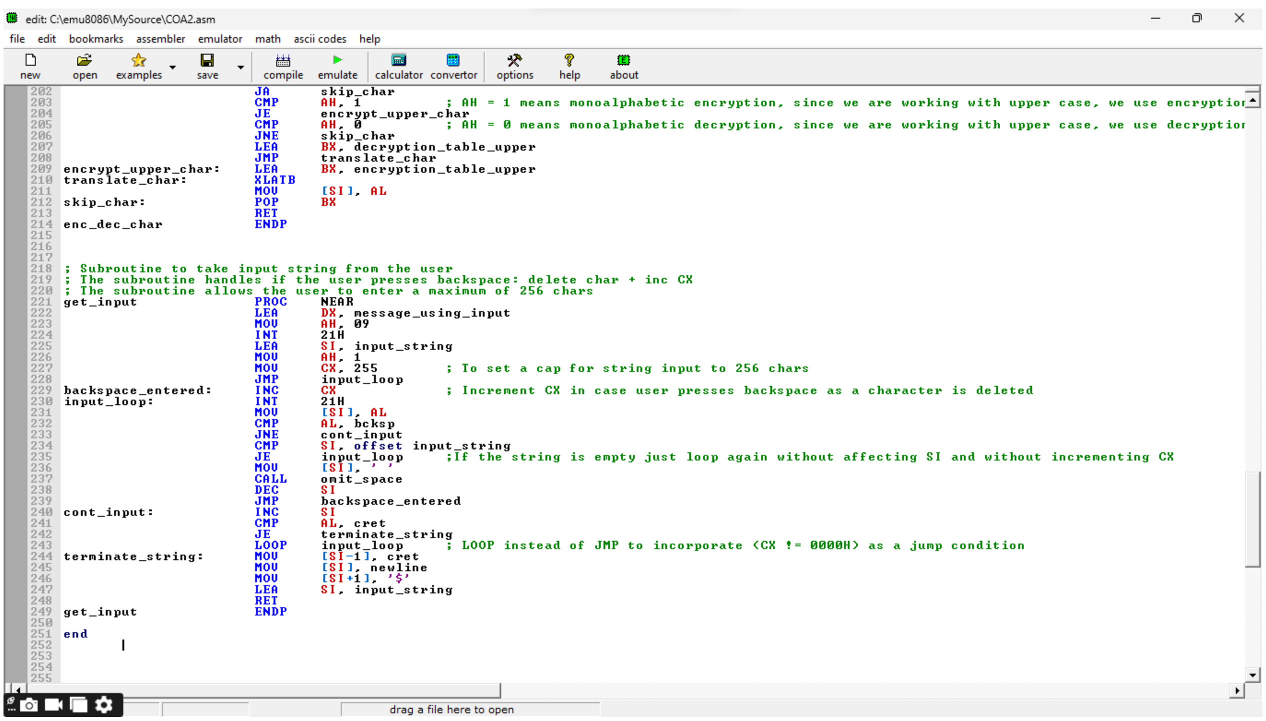


Diag4-Code running in emu8086



```
edit: C:\emu8086\MySource\COA2.asm
file  edit  bookmarks  assembler  emulator  math  ascii codes  help
new  open  examples  save  compile  emulate  calculator  convertor  options  help  about

                        JMP     next_char
end_of_string:          POP     SI
                        RET
encrypt_decrypt         ENDP



;
; Subroutine to send space to the end of the string (after '$')
; You can delete this subroutine if you do not need it
;
omit_space              PROC    NEAR
                        PUSH    SI              ; The reason I send the space after the '$'
                                                ; is to handle several consecutive spaces without
omit_space_loop:        MOV     AL, [SI+1]      ; entering an inifnite loop as opposed to just swapping
                        MOV     [SI+1], ' '     ; the ' ' character with the character after it
                        MOV     [SI], AL
                        INC     SI
                        CMP     [SI-1], '$'
                        JNE     omit_space_loop
                        POP     SI
                        RET
omit_space              ENDP



; Subroutine to convert character with the
; appropriate table (encrypt/decrypt)(uppercase/lowercase)
enc_dec_char            PROC    NEAR
                        PUSH    BX
                        CMP     AL, 'a'
                        JB      check_upper_char
                        CMP     AL, 'z'
                        JA      skip_char
                        CMP     AH, 1          ; AH = 1 means monoalphabetic encryption, since we are working with lower case, we use encryptior
                        JE      encrypt_lower_char
                        CMP     AH, 0          ; AH = 0 means monoalphabetic decryption, since we are working with lower case, we use decryptior
                        JNE     skip_char
                        LEA     BX, decryption_table_lower
                        JMP     translate_char
encrypt_lower_char:     LEA     BX, encryption_table_lower
                        JMP     translate_char
check_upper_char:       CMP     AL, 'A'
                        JB      skip_char
                        CMP     AL, 'Z'
                        JA      skip_char
                        CMP     AH, 1          ; AH = 1 means monoalphabetic encryption, since we are working with upper case, we use encryptior
                        JE      encrypt_upper_char
                        CMP     AH, 0          ; AH = 0 means monoalphabetic decryption, since we are working with upper case, we use decryptior
                        JNE     skip_char
                        LEA     BX, decryption_table_upper
                        JMP     translate_char
encrypt_upper_char:     LEA     BX, encryption_table_upper
translate_char:         XLATB

drag a file here to open
```

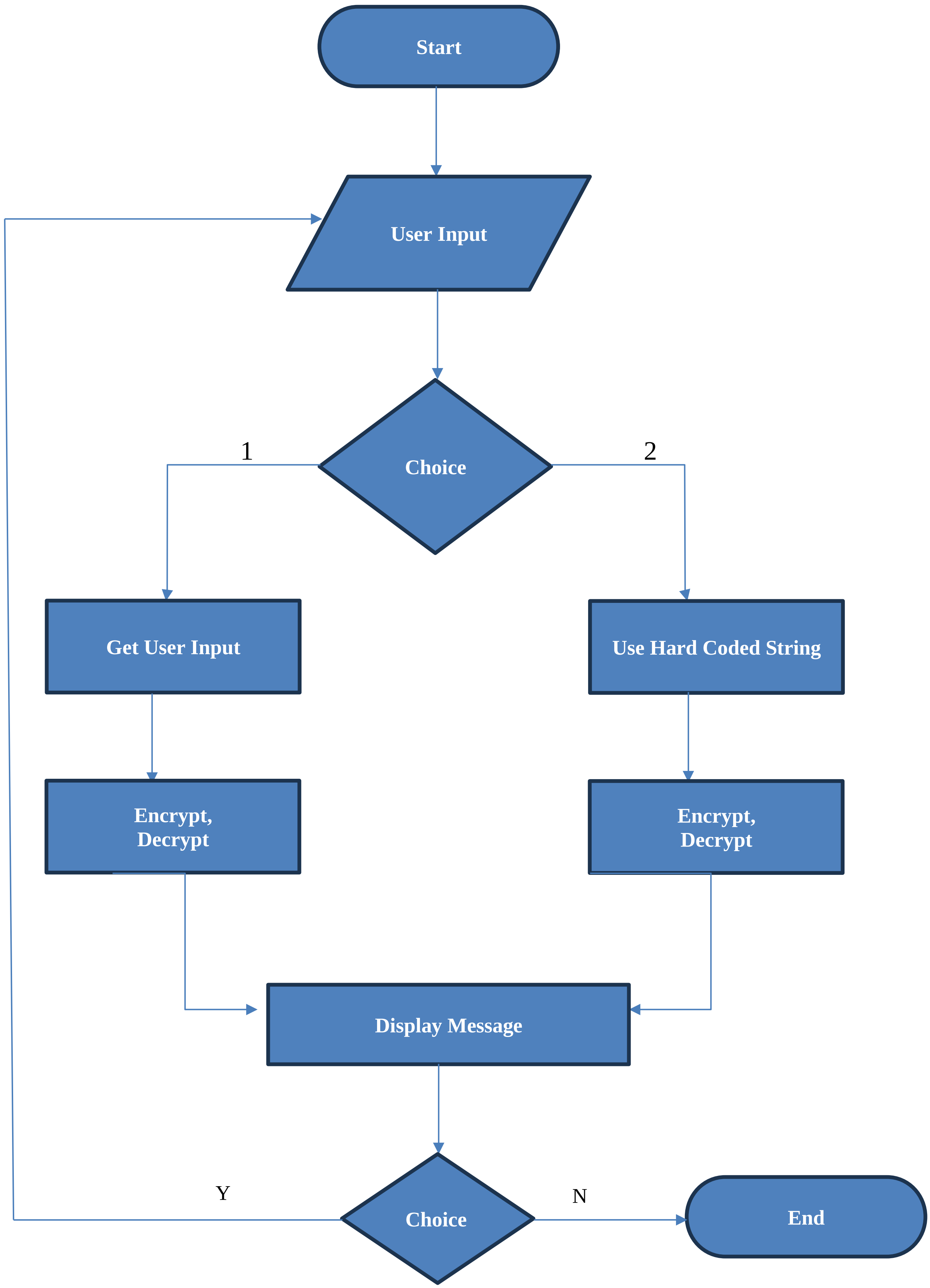
Diag5 -Code running in emu8086



```
edit: C:\emu8086\MySource\COA2.asm
file  edit  bookmarks  assembler  emulator  math  ascii codes  help
new  open  examples  save  compile  emulate  calculator  convertor  options  help  about

                        JA      skip_char
                        CMP     AH, 1          ; AH = 1 means monoalphabetic encryption, since we are working with upper case, we use encryptior
                        JE      encrypt_upper_char
                        CMP     AH, 0          ; AH = 0 means monoalphabetic decryption, since we are working with upper case, we use decryptior
                        JNE     skip_char
                        LEA     BX, decryption_table_upper
                        JMP     translate_char
encrypt_upper_char:     LEA     BX, encryption_table_upper
translate_char:         XLATB
                        MOV     [SI], AL
skip_char:              POP     BX
                        RET
enc_dec_char            ENDP



; Subroutine to take input string from the user
; The subroutine handles if the user presses backspace: delete char + inc CX
; The subroutine allows the user to enter a maximum of 256 chars
get_input               PROC    NEAR
                        LEA     DX, message_using_input
                        MOV     AH, 09
                        INT     21H
                        LEA     SI, input_string
                        MOV     AH, 1
                        MOV     CX, 255        ; To set a cap for string input to 256 chars
                        JMP     input_loop
backspace_entered:      INC     CX             ; Increment CX in case user presses backspace as a character is deleted
input_loop:             INT     21H
                        MOV     [SI], AL
                        CMP     AL, bcksp
                        JNE     cont_input
                        CMP     SI, offset input_string
                        JE      input_loop     ;If the string is empty just loop again without affecting SI and without incrementing CX
                        MOV     [SI], ' '
                        CALL    omit_space
                        DEC     SI
                        JMP     backspace_entered
cont_input:             INC     SI
                        CMP     AL, cret
                        JE      terminate_string
                        LOOP    input_loop     ; LOOP instead of JMP to incorporate (CX != 0000H) as a jump condition
terminate_string:       MOV     [SI-1], cret
                        MOV     [SI], newline
                        MOV     [SI+1], '$'
                        LEA     SI, input_string
                        RET
get_input               ENDP

end

drag a file here to open
```

Diag6-Code running in emu8086

5. FLOWCHART

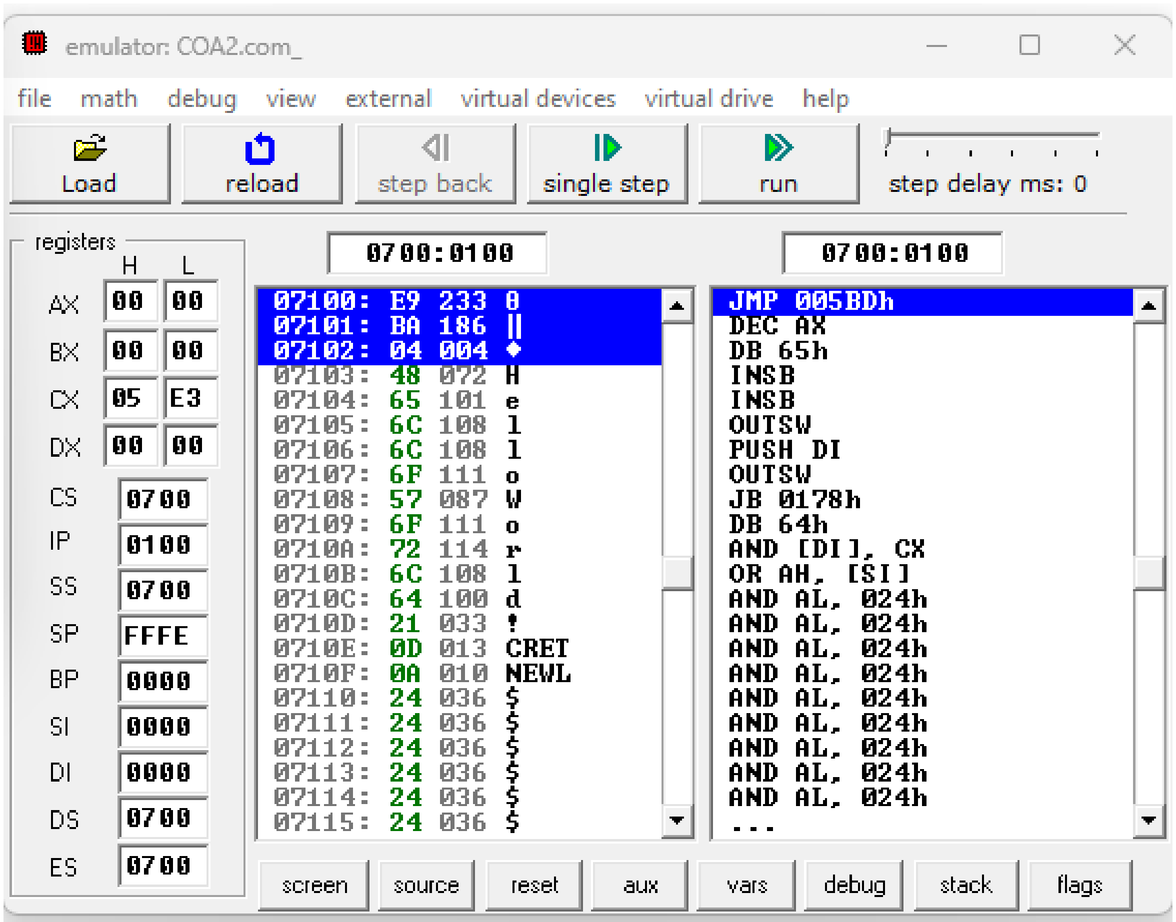

Start
User Input
Choice
1
2
Get User Input
Use Hard Coded String
Encrypt,
Decrypt
Encrypt,
Decrypt
Display Message
Y
N
Choice
End

6. EXPERIMENT RESULTS & ANALYSIS

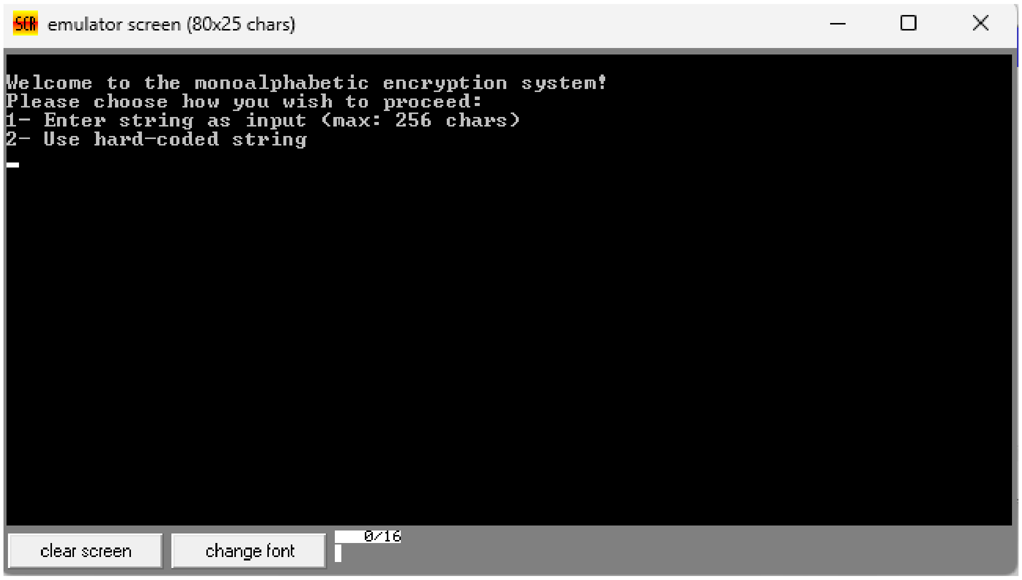
The program lets the user choose whether to enter a string or use the hard-coded one

```
; Hard-coded string:
hardcoded_string        DB      'Welcome To encryption system', cret, newline, '$'

input_string            DB      259 dup ('$')     ; Reserved area for input string (256 chars + \r + \n + $)
```

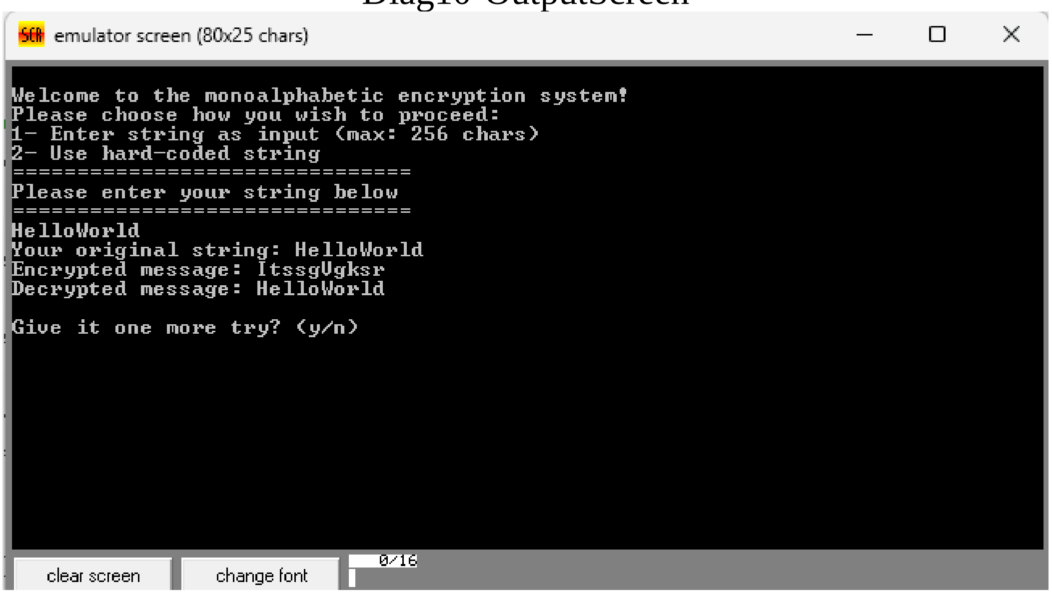
Diag8-Hard Coded String



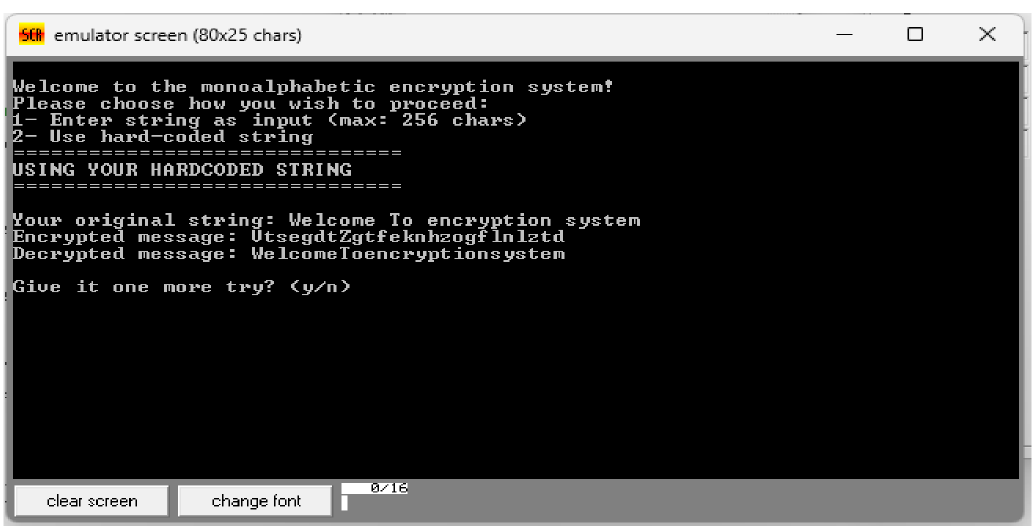
Diag9-Emulator



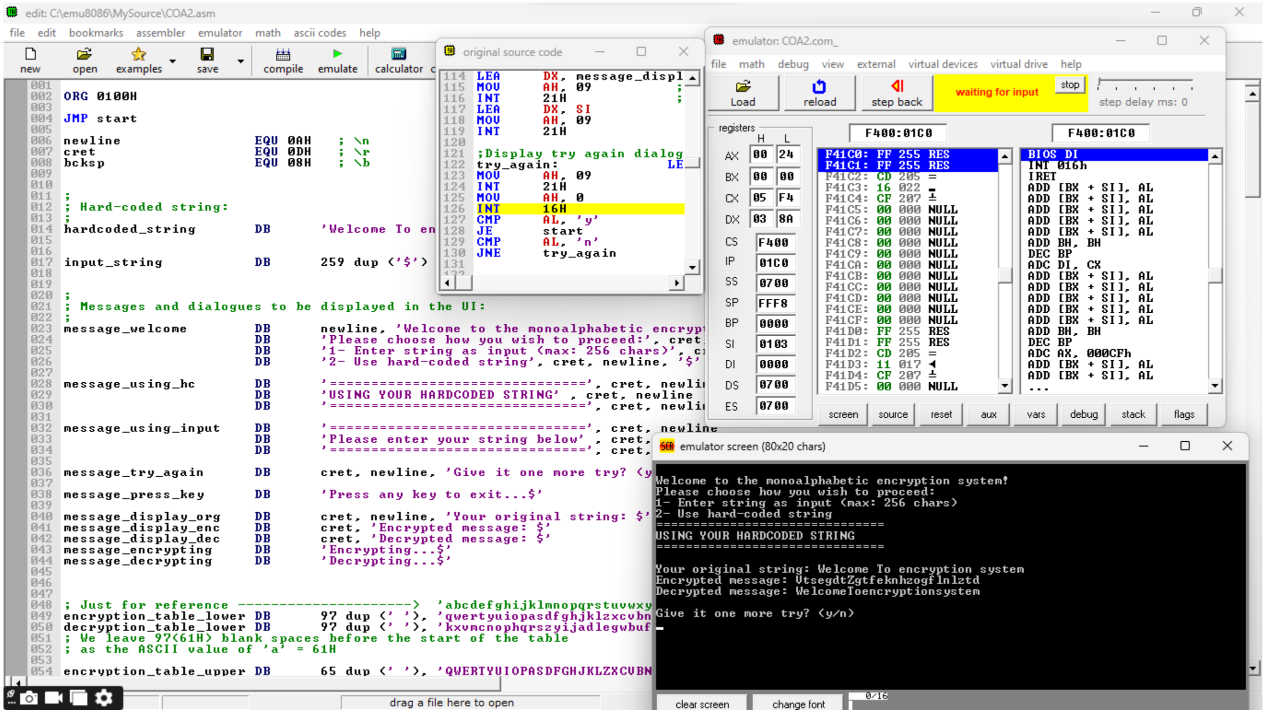
Diag10-OutputScreen



Diag11-Output with UserInput



Diag12-Output with Hard Coded String



Diag13-Program Execution

7. CONCLUSION

The monoalphabetic encryption system project serves as a valuable tool for data security and communication privacy. While this assembly code example provides a basic implementation, it highlights the need for encryption and decryption mechanisms in various real-world applications.
Importance of Encryption:

- Data Protection: In an increasingly digital world, the protection of sensitive information is paramount. Encryption helps safeguard data from unauthorized access and eavesdropping, ensuring confidentiality.
- Privacy: Individuals and organizations need encryption to maintain the privacy of their personal and confidential data, such as financial records, healthcare information, and communication.
- Secure Communication: Secure communication channels are essential, particularly in industries like finance, healthcare, and government, where transmitting sensitive information is routine.
- E-commerce: E-commerce platforms rely on encryption to protect customer data, including credit card information and personal details.
- National Security: Governments employ encryption to safeguard classified information and maintain national security.
- Future Development:The need for encryption and data security will continue to grow as technology evolves. In an era of increasing cyber threats and data breaches, projects like this serve as a foundation for developing advanced encryption methods and secure applications.

In conclusion, the monoalphabetic encryption system project underscores the significance of data security and privacy in our digital age. It not only provides a basic implementation but also encourages further exploration and development in the field of encryption. As technology advances, the demand for secure communication and data protection will remain a crucial consideration for individuals, organizations, and governments.

8.REFERENCES

https://www.geeksforgeeks.org/architecture-of-8086/
Unit2: Basic Structure of a Computer
https://www.philadelphia.edu.jo/academics/qhamarsheh/uploads/emu8086.pdf
https://www.cloudflare.com/learning/ssl/what-is-encryption/