# Data Encryption

Microprocessor Project



By:- Aanchal

# Introduction

Assembly language is a programming language which allows user to communicate with the computer at a more fundamental level. An assembly (or assembler) language, often abbreviated asm, is a low-level programming language for a computer, or other programmable device, in which there is a very strong (generally one-to -one) correspondence between the language and the architecture’s machine code instructions. Each assembly language is specific to particular computer architecture. In contrast, most high-level Programming languages are generally portable across multiple architectures but require interpreting or compiling. Assembly language may also be called symbolic machine code. Assembly language is converted into executable machine code by a utility program referred to as an assembler. The conversion process is referred to as assembly, or assembling the source code. Assembly time is the computational step where an assembler is run. Assembly language uses a mnemonic to represent each low-level machine instruction or opcode, typically also each architectural register, flag, etc.

Many operations require one or more operands in order to form a complete instructions and most assembler can take expressions of numbers and named constants as well as registers and labels as operands, freeing the programmer from tedious repetitive calculations. Depending on architecture, these elements are also being combined for specific instructions or addressing mode using offsets or other data as well as fixed addresses. Many assemblers offer additional mechanisms to facilitate program development to control the assembly process, and to aid debugging.

# What is Data Encryption?

Cryptography, from the Greek word kryptos, meaning *"hidden"*, deals with the science of hiding a message from eyes you do not want to understand the contents of the message. There are many ways to keep something secret. One way is to physically hide the document. Another way is to encrypt the text you wish to remain secret.

Encrypting a file requires an input message, called the ***"plain text"*,** and an encryption algorithm. An encryption algorithm transforms "plain text" into ***"cipher text"****.* Just like a door needs a key to lock and unlock it, an encryption algorithm often requires a key to encrypt and decrypt a message. Just like a homeowner can selectively give the key to the front door to only those he wants to allow unaccompanied access, the author of a message can selectively give the encryption key to only those he wants to be able to read the message. In order for someone to read the encrypted message, he has to decrypt the cipher text, which usually requires\_the\_key.  
  
For example, suppose the plain text message is **hello world**. An encryption algorithm would produce the cipher text **axppsgsupn**. If someone saw **axppsgsupn**, he would have no idea what it meant (unless, of course, he could figure it out, or had the key to decrypt it). The key to encrypt in this case is according to the following table:-

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

**Purpose:-** The purpose of data encryption is to protect digital data confidentiality as it is stored on computer systems and transmitted using the internet or other computer networks.

# Algorithm

**1. Get Input:**

- Program will print a message “Enter the string: “

- INPUT A MESSAGE .WHEN DONE, PRESS <ENTER>

* The user will input a character string(with or without spaces) from the keyboard, terminating the message with the <Enter> key.
* Program will store the message,

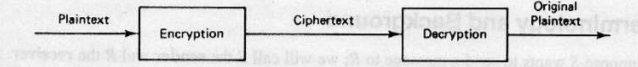
**2. Print Outcome:**

-When the <Enter> key is hit, the program will show:

(i) Encrypted message

(ii) Decrypted message

in the new line (without spaces as present in the original string).

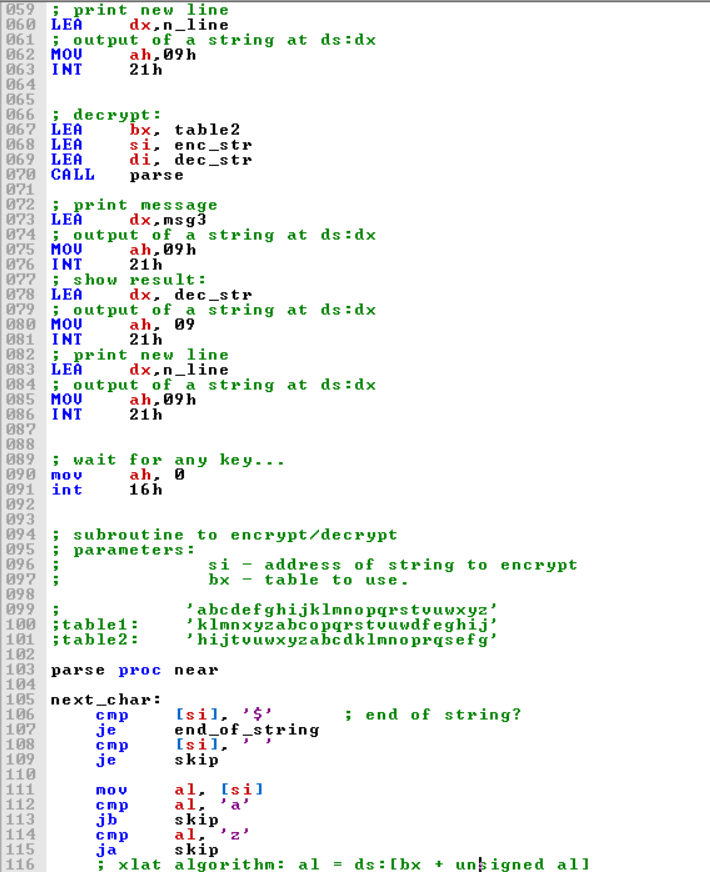
****

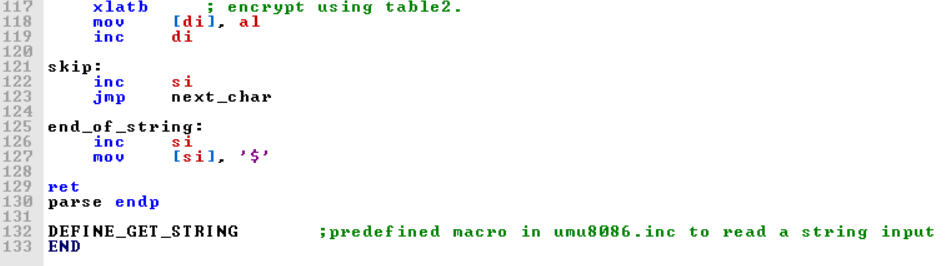
# Code Explanation

|  |  |
| --- | --- |
| include ‘emu8086.inc’ | Library of common functions |
| org 100h | Assembly directive. Tells where the machine code is to be placed |
| jmp start | Control transferred to start |
| table1 db 97 dup (‘’),‘klmn.’ | Reserve an array of 97 bytes and give it a name table1 |
| msg1 db ‘Enter the string: ’, ‘$’ | Reserve bytes of memory equal to the no. of chars in the string named msg1 |
| n\_line db 0DH,0AH, ‘$’ | For new line |
| Str db 256 DUP(‘$’) | A string str of 256 spaces having value ‘$’ |
| Start: | |
| LEA dx,msg1 | Load address of msg1 to dx |
| MOV ah,09h  INT 21h | Print String at ds:dx:- “Enter the string: ” |
| PUSH CS | Storing the value of CS in stack |
| POP DS | Setting DS to the original value of CS |
| LEA DI,str | Get address of str in di |
| MOV DX,00FFH | Move 00ffh to dx |
| CALL GET\_STRING | String is called in to the buffer ds:di |
| LEA dx,n\_line  MOV ah,09h  INT 21h | This part of code will bring the cursor pointing to the new line. (next content will printed in new line) |
| LEA bx,table1 | Get address of table1 in bx for encryption |
| LEA si,str | Get address of str to be encrypted in si |
| LEA di, enc\_str | Get address of enc\_str(where ans to be stored) in di |
| CALL parse | Calling the procedure parse |
| LEA dx,msg2 | Load address of msg2 to dx |
| MOV ah,09h  INT 21h | Print String:-“Encrypted string: ” |
| LEA dx, enc\_str | Get address of encrypted string in dx |
| MOV ah,09h  INT 21h | Print the encrypted string formed from the string entered by the user |
| LEA dx,n\_line  MOV ah,09h  INT 21h | This part of code will bring the cursor pointing to the new line. (next content will printed in new line) |
| LEA bx, table2 | Get address of table2 in bx for decryption |
| LEA si, enc\_str | Get address of encrypted string to be decrypted in si |
| LEA di, dec\_str | Get address of dec\_str(where ans to be stored) in di |
| CALL parse | Calling the procedure named parse |
| LEA dx,msg3 | Load address of msg3 to dx |
| MOV ah,09h  INT 21h | Print String:- “Decrypted string: ” |
| LEA dx,dec\_str | Get address of decrypted string in dx |
| MOV ah,09h  INT 21h | Print the decrypted string formed by the decoding of encrypted string |
| LEA dx,n\_line  MOV ah,09h  INT 21h | This part of code will bring the cursor pointing to the new line. (next content will printed in new line) |
| MOV ah, 0  INT 16h | This instruction is used to read character from the buffer and wait if it is empty |
| parse proc near | |
| next\_char: | |
| cmp [si], '$' | Compare with ‘$’ |
| je end\_of\_string | If equal to ‘$’ jump to end of the string |
| cmp [si], ' ' | Compare with ‘ ’ |
| je skip | If equal to ‘ ’ skip that character |
| mov al, [si] | Copy content present at memory location si to al |
| cmp al, 'a' | Compare al with a, ascii value will be compared |
| jb skip | If below than a(ascii value less than a) go to skip |
| cmp al, 'z' | Compare al with z, ascii values will be compared |
| Ja skip | If above than z(ascii value greater than z) go to skip |
| xlatb | Else encode that using the above table |
| mov [di], al | Copy content present at al to the memory location at di |
| inc di | Increment di by 1 |
| skip: | |
| inc si | Increment si by 1 |
| jmp next\_char | Go back to next character |
| end\_of\_string: | |
| inc si | Increment si by 1 |
| mov [si], '$' | Copy ‘$’ to the memory location of si |
| ret | Return to the operating system |
| parse endp | Ending of the procedure parse |
| DEFINE\_GET\_STRING | Procedure to get a NULL terminated string from user |
| END | End of the program |

# ALP:

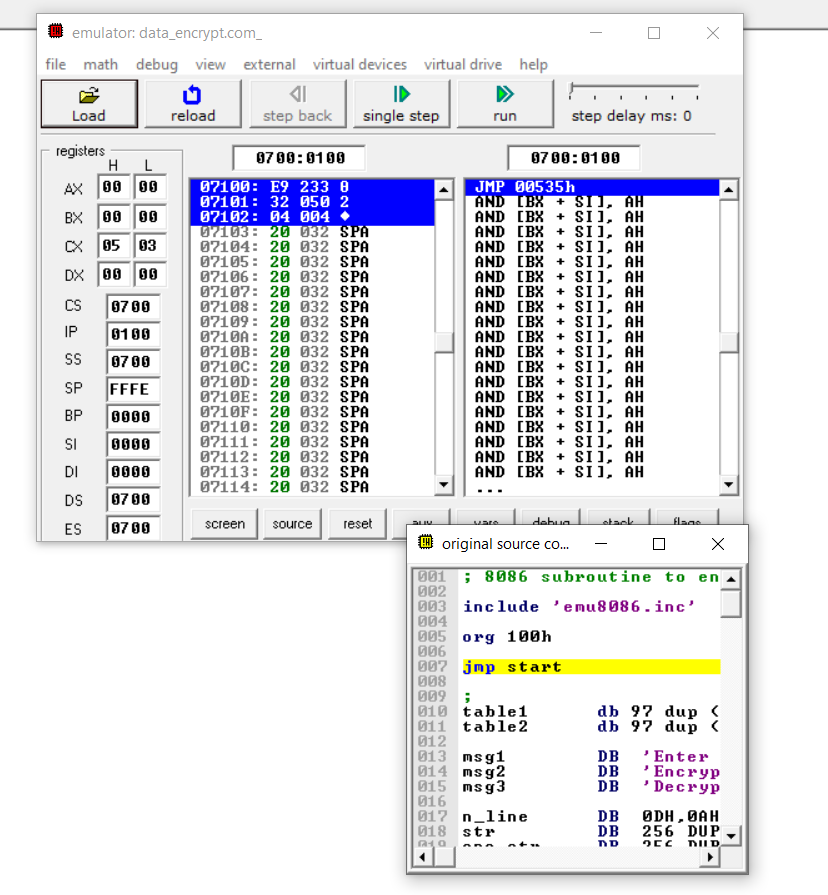






# Results:

1. Before starting emulation:



ii) Emulator screen:



# Conclusion:

The project run smoothly on emu8086 and is able to successfully encode the string provided by the user and then taking the same string as input it is able to decode it.

# Future scope:

In today’s world the protection of sensitive data is one of the most critical concerns for organizations and their customers.  As a result cryptography is emerging as the foundation for enterprise data security and compliance, and quickly becoming the foundation of security best practice. Encryption is the most reliable way to secure the data. As of now, this project deals with alphabets only and takes a fixed string size. Despite of these limitations, it helps in understanding more complex problems. And at the end of the day, we need to protect our data and this project is able to do so**.**

# References:

1. *Advanced Microprocessors and Peripherals*

-KM Bhurchandi and A K Ray

*2. Microprocessors and Interfacing*

-A.P. Godse and D.A. Godse

3. *DOS interrupts* from the site given below:

<http://www2.ift.ulaval.ca/~marchand/ift17583/dosints.pdf>

# Thank you!