CSUS

SCHOOL OF ENGINEERING AND COMPUTER SCIENCE

Department of Computer Science

CSC 35

Spring 2021

Dr. Ghansah

**Lab #8: I/O Procedures, Simple Encryption and Decryption.**

**Purpose:** *The purpose of this lab assignment is to get the student more experience with the* ***procedures in Intel Architecture*** *by implementing I/O procedures for a simple encryption and decryption application* ***.***

**Introduction**   
  
This assignment is primarily a symmetric key cryptography assignment using XOR to do encryption and decryption. In symmetric key cryptography, two parties *A* and B that are communicating use a shared secret key *K*. The same key is used for both encryption and decryption. Given a plain text message *m*, *A* will encrypt *m* using key *K* to produce encrypted (aka cipher) text, *c = E(m,K)*. B will decrypt *c* using key *K* and the function *D(c,K)* which should produce *m.* That is, *D(c,K) = m* if *B* knows the secret key *K*. In this assignment, the functions *E* and *D* are both simple *XOR*. Usually, the key should be of the same size as the message. Thus, the message is divided into blocks where each block is of size that is equal to the key size. Each block is encrypted with the same key. This means that a 1 byte key (range, 0..28 – 1) will be used to encrypt each character, a 2 byte key (range 0… 216 – 1) will encrypt two bytes at a time, a 4 byte key (range 0… 232 – 1) will encrypt four bytes at a time, etc. Note that because we are using simple XOR to do the cryptography some keys are not appropriate because they will not help garble the encrypted message. For instance, K=0 will not encrypt, K=2n – 1 which consists of all 1s will simply invert each bit. Also, for 1 byte encryption, K=32 will simply convert the case from lower to upper and vice versa in the case of plain text letters, etc. I am giving you this information to help you test your program properly after it is done.

In this lab, we are going to write several I/O procedures that use specific Irvine Library procedures to perform our input and output functions. Some procedures will also be used to encrypt and decrypt text. For this assignment you will be required to encrypt using *2 byte keys*. Subsequent assignments might require you to use multiple byte keys ( 3, 4, ..). The procedures you are required to use for this assignment are specified below. Some of the procedures are part of the Irvine Libraries. Others have already created in previous lab assignments. For instance, PrintDec has already been written as part of the previous assignment whereas both CrLf and ClrScr are part of the Irvine Library. Please see the attached program file.  
  
**Specifications**   
  
Write the following procedures using only the specified Irvine Library functions including reading single or multiple string of characters/numbers from the keyboard, and writing single or multiple characters to the screen. Characters can be transferred through registers or memory as the case might be as specified below. In the specification below, all parameters are passed via registers. However, you can choose to pass input parameters via the Stack but Output parameters must be placed in the registers specified. If you choose to pass parameters via the stack you must modify parts of my main program to accommodate that.

\*P1. ClrScr; Scroll the whole screen window. No parameters. Irvine Lib

\*P2. WriteChar; Writes character to screen.

Entry: ASCII Character in AL. Irvine Lib

\*P3. WriteString; Writes ASCII string to the screen. The string is in memory and terminated with ASCII NULL (0)

Entry: EDX = Address of String; Returns nothing.

Note: This procedure is the same as to Irvine’s WriteString and will be used in this assignment primarily to display prompts to the screen. Please, see attached main program.

\*P4. CrLf ; moves the cursor to a newline (Carriage Return and Line Feed). No parameters needed. Irvine Lib

\*P5. ReadChar ; Read single character from Keyboard (standard input)

Entry: Nothing, Exit: ASCII Char in AL. It does NOT ECHO to the screen. Irvine Lib

---------------------------------------------------------------------------------------------------------

P6. GetKey (unsigned); Read the encryption/decryption key as multiple Decimal Digits from keyboard (ie ASCII to Binary). Stops when user presses ENTER key (0dh). Returns binary number in AX

Note: This procedure is similar to GetInput procedure from previous labs except that it should NOT ECHO the key to the screen. It should print three ‘\*’ characters on the screen regardless of the size of the key.

P7. GetPlainText; Read string of characters text from keyboard, echo them to the screen, and store them in memory.

Entry: EDX = offset of bytes where data is stored. ECX: max # of characters to be read. Exit: actual count of bytes read in ECX. Stops when user presses ENTER key (0dh)

P8. PrintDec; Print Decimal Digits to Screen (ie. Binary to ASCII). This procedure is used in this assignment to Print the number of plain text characters that are typed in.

Entry: Integer in EAX.

Returns nothing.

Note: This Procedure is the same as OutputW procedure for outputting multiple decimal digits on the screen you created in previous lab assignment.

P9. Crypt2Byte; Used to Encrypt or Decrypt Text from one region in memory and place the results into another region in memory. The key is 2Bytes. Thus, the text to be encrypted/decrypted must divided into 2Byte blocks. Each block is encrypted with the same key. The last block in a plaintext is padded with 0 (ASCII) null if the size of the plaintext is an odd number. This is done prior to encryption.

Entry: ESI = ptr to source, EDI= ptr to destination, ECX= buffer count, AX=Key.

Exit: Nothing

Hint: To encrypt, ESI = ptr to plaintext and EDI= ptr to encrypted text (ciphertext).

To Decrypt, ESI=ptr to ciphertext and EDI= ptr to plaintext.

P10. PrintText; Print text to the screen. Used in this assignment to Print Encrypted or Decrypted Text on the screen.

Entry: EDX= pointer to Text Buffer; ECX= Buffer count.

Exit: Nothing

\* represent procedures you do not have to write because they are Irvine Library Procedures you are allowed to use.

**Requirements**   
  
1. You cannot use any other Irvine library functions except the ones that I have specified. If you use any other external library functions, you will get zero credit for the assignment.   
  
2. A MAIN procedure will be supplied in a separate file. Your program should work perfectly with the supplied MAIN procedure. Please note that the supplied main program is designed to work with the above procedure specification where all parameters are passed via the specified registers. YOU ARE NOT ALLOWED TO CHANGE THE MAIN PROGRAM EXCEPT FOR THE FOLLOWING REASON. If you choose to pass input parameters via the Stack you must change the main program accordingly but the procedures should work perfectly with the modified main program and the session should still work exactly as shown. The supplied main program is also designed with work with MASM.   
3. None of the procedures should use the data segment to store any data directly except through parameters that are passed via the registers as defined in the procedures.

4. As usual, this program must be demonstrated.

5. Your program should have adequate detailed comments (as usual)   
7. None of the procedures should modify any of the registers unless the register is returned as an output parameter. This means you should save registers on stack on entry to the procedure and restore them on exit from the procedure.  
  
**Hints**

1. The following Procedures can be gleaned from Lab 7

GetKey is a special form of GetInput Procedure

PrintDec is same as OutW procedure

2. These following are the only Irvine Libraries you are allowed to use.

ClrScr

WriteString

ReadChar

WriteChar

CrLf

3. Be sure to save all the values in the registers you use when you enter a procedure and restore them when you leave. But be aware that some procedures, by definition, modify certain registers so be careful what you push and pop.

4. Here is the Algorithm for the Crypt2Byte Procedure

1. Push Relevant Registers on stack

2. Check to see if plaintext size is even

3. If yes, no NULL (0) Pad is needed so go to step 6

4. Add a pad ( 0 ) to the end of the plaintext

5. Update length of plaintext to take the pad into account.

6. Get 2 Bytes from plaintext memory buffer

7. Encrypt/Decrypt with key (XOR)

8. Put encrypted 2 bytes in Encrypted text memory buffer

9. Update buffer pointers to point to next block

10. Check to see if we are done with last block

11. If not done Go to step 6.

12. Return count of bytes to output parameter to relevant register.

13. Pop Registers saved on the stack

--------------------------------------------------------------------   
**The Supplied MAIN Procedure and Specification of the Procedures**

**are in a Separate File**   
The main procedure assumes you have all the parts completed. If you want to test a subset of the procedures using the MAIN procedure, you may need to comment parts of it out. Alternatively, you may provide your own main procedure for some parts for testing purposes but **all the parts MUST work with this supplied MAIN procedure**.   
-------------------------------------------------------------------

**EXAMPLE SESSION: See below**

**Testing:** Be sure to test your program before you demonstrate it to your instructor.

**Demonstration**: Demonstrate your program to the instructor by supplying screen shots for your session. It must be a Word Document. The Instructor will check whether all the procedures have been written, they are well-documented, whether they work with the supplied main program under different conditions, and whether you used any Irvine Library procedures illegally.

**Submission:** Submit well documented electronic copy of your program to CANVAS including the source code (.asm ), Output session, and report (.doc, docx). The source code should include the main program and the *5* procedures. The report should include program design (in pseudocode and/or flowchart), lessons learned, problems encountered, how long it took you to complete this assignment, and suggestions (if any) for the future. The output should include sessions you used to test it.

**General Grading Rubric**

Demonstration: 50%

Report: 50% total will be based on

* Source Code -documented with comments: 40%. You will *lose 5% for each global variable* seen in any procedure.
* Output Sessions: 5%
* Pseudocode/Flowchart and Comments (ie how long, problems, lessons learned, suggestions, etc): 5%

**EXAMPLE SESSION**

In the sessions below notice that because the key must be kept secret, three astericks are displayed regardless of the key size. Also, the user is asked to type the key before the encrypted data is decrypted. This is supposed to serve as a test that the decryption works. In a real system the encrypted text will be sent over a network so that an observer will have difficulty figuring out what is being sent. In other applications the encrypted message can be in a file such that any one opening the file will see garbled information until they produce a valid key before the file is decrypted.

In the 2nd session the message is the same but the key is different

In the 3rd session the user failed to type in the correct key.

You can try these and other sessions, including ones where we don’t want to print encrypted text, etc and put it in your output file. If you want me to validate your work include the key you used to produce those sessions. However, you don’t have to because if your program indeed works I should be able to put it my own text and keys to validate it.





