CECS 285: Lab 11

OBJECTIVES:

- Use Rotating bit encryption and decryption with a multi-byte key
- Make this program menu driven through the serial console
- Use given subroutines and define your own subroutines to make this program as modular as possible

ACTIVITY 1: Rotating bit encryption is to be applied in this lab. This is not a true symmetrical encryption technique because the same exact algorithm will not encrypt and decrypt a message. The only difference between the encryption and decryption algorithms in this case are that the directions of rotation are opposite. For our purposes, we will rotate bits to the right to encrypt and rotate bits to the left to decrypt.

The same key that was used to encrypt a message must also be used to decrypt the cipher text.

The functional operations performed by this program are:

1. Display a console message requesting plain text
2. Save the plain text to 8051 RAM
3. Encrypt the plain text in 8051 RAM
4. Output cipher text from 8051 RAM
5. Display a console message to indicate encryption had completed successfully

At the beginning of this program:

- use the following constant and variable definitions

```
;----------------   constant definitions    ----------------;
null     equ 0x00
cr       equ 0x0d
lf       equ 0x0a
tab      equ 0x09
keyBytesRAMaddress  EQU 0x22 ;  symbolic constant for base address of
;                               encryption key in RAM
txtRAMaddress       EQU 0x30 ;  symbolic constant for base address of
;                               encryption key in RAM

;----------------   variable definitions    ----------------;
keyvalIndex         EQU 0xe0        ;   variable to index the keyval constant array
keyLength           equ 0x20    ;   variable to track length of key
txtLength           equ 0x21    ;   variable to track length of key
charIndex           equ 0xe0    ;   alias for accumulator
choice              equ 0x7F    ;   variable to store selected operation
```

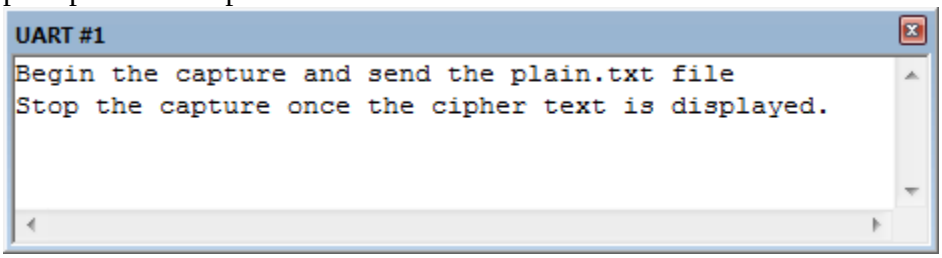
The program will go through a similar initialization and configuration process as lab 10 where configuration of the serial peripheral is performed and the key is loaded from ROM into RAM. This time a subroutine will load key bytes from ROM into RAM. The LoadKeyFromROM subroutine is listed at the beginning of the next page along with a description of its usage:

```
;----------- LoadKeyFromROM ----------;
;Receives ROM location of key array in dptr before it is called
;loads bytes from a constant array of key values into RAM
;returns nothing

LoadKeyFromROM:
        mov     r0,             #keyBytesRAMaddress-1   ;initialize RAM pointer
        mov     keyvalIndex,    #0x00                   ;initialize accumulator
    GetNextKeyByteFromROM:
        inc     r0                                  ;increment RAM pointer
        push    keyvalIndex                         ;preserve keyvalIndex variable
        movc    a,              @a+dptr             ;load byte of key into accumulator
        notNull:
            mov     @r0,        a                   ;put byte of key into ram
            pop     keyvalIndex                     ;restore keyvalIndex variable
            inc     keyvalIndex                     ;increment keyvalIndex
        cjne    @r0, #0x00,     GetNextKeyByteFromROM   ;check for null terminating charac

    LoadDone:
        mov     @r0, #0x00                          ;append null char to string
        ret
```

After the key is loaded into RAM and the main loop program section will be entered, display a prompt to ask for plain text.



The prompts and messages to be displayed in the serial console must be listed at the very end of your program source. This multiline message is defined in software as follows:

```
PromptPT:   db  "Begin the capture and send the plain.txt file",cr,lf
            db  "Stop the capture once the cipher text is displayed.",cr,lf,null
```

Note the usage of symbolic constants in places where carriage return, line feed, and null must be inserted. The WriteString subroutine only needs to be called once to output the entire message since it only stops writing characters once the null is found.

In previous versions of our encryption program the plain text was read, encrypted, and then sent to the console output. For this Lab, the plain text will be stored in 8051 RAM so the length of the message must not be so large that other areas of RAM are overwritten.

Store plain text in 8051 RAM by calling the BufferText subroutine which is defined as follows:

```
;---------- Buffer Text ----------;
;Receives no parameters
;Reads a series of TXT bytes from serial Rx
;writes the bytes to RAM at location indicated by keyBytesRAMaddress
;Returns length of key in the keyLength variable
BufferText:
        mov     r1, #txtRAMaddress      ;initialize pointer
    WaitForTXTChar:
        jnb ri, $                       ;wait to receive char
        call    getchar                 ;char received, get it!
        mov     @r1, a                  ;store character in ram
        inc     r1                      ;increment pointer
        ;cjne   a, #0x00, WaitForTXTChar    ;check for null char
        cjne    a, #0x0D, WaitForTXTChar    ;DEBUG:  check for Enter char for debug
            ret
```

Notice the two cjne instructions shown above where one is commented. When running the program in debug simulation the character 0x0D is being checked so that text can be entered manually and buffering of characters stops when the enter key is pressed. When running the program on 8051 hardware use null character terminated string in your text files.

After the plain text has been buffered it is ready to be encrypted.

In our previous programs the encryption functionality was put in line with the rest of the program source. In this lab we are employing modular design so the encryption subroutine will be put into a subroutine named RotationEncrypt.

The RotationEncrypt subroutine will transform each plain text character using the Rotation encryption technique and write each cipher text character back to the RAM location from which it was read. Study the usage of pointers in the subroutine source code to see how this is done.

Debug versions of the compare instructions are shown for use with the Keil debug environment.

The definition of this subroutine is shown as follows.

```
;---------- Rotation Encrypt ----------;
;Receives no parameters
;Encrypts the Plain Text contained in RAM
;Returns nothing
RotationEncrypt:
    mov r0, #keyBytesRAMaddress ;re-initialize key pointer
    mov r1, #txtRAMaddress      ;re-initialize txt pointer
RotationEncryptNextChar:
    mov a,  @r0                 ;initialize rotate loop count
    mov r6, a                   ;must be passed to a before r6
    mov a,  @r1                 ;get char from plain text

    rotateEncrypt:
        rr      a
        djnz    r6, rotateEncrypt

    mov @r1, a                  ;write encrypted character back to RAM

    inc r0                      ;point to next key Byte
    cjne    @r0, #0x00, dontResetRotationEncryptionKeyPtr
;   cjne    @r0, #0x0D, dontResetRotationEncryptionKeyPtr   ;DEBUG:
        mov r0, #keyBytesRAMaddress ;reinitialize key pointer
    dontResetRotationEncryptionKeyPtr:
    inc r1                      ;point to next plain text char
        cjne    @r1,    #0x00, RotationEncryptNextChar
    ;   cjne    @r1, #0x0D, RotationEncryptNextChar ;DEBUG:  check for Enter
        ret                     ;end of string reached
;------------------------------------------------;
```

After the Rotation Encrypt subroutine has completed its task the plain text contained in RAM will have been transformed into cipher text.

The cipher text must now be read from RAM and sent to the serial output.

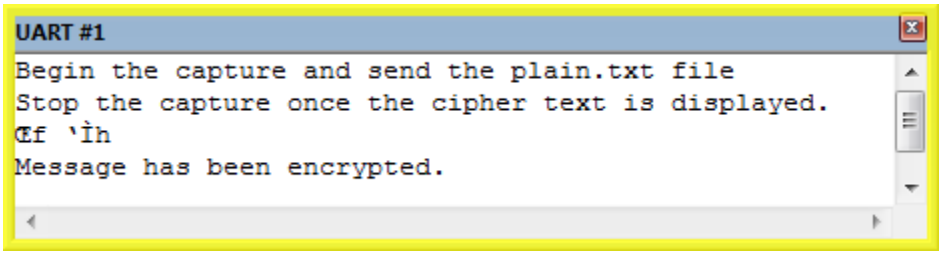
A subroutine named WriteBufferedText will be used to read cipher text from RAM and write the characters to the serial console.

```
;----------- WriteBufferedText ----------;
;receives address of Buffered Text in r1
;sends Buffered Text serially using writechar
;returns nothing
WriteBufferedText:
    mov r1, #txtRAMaddress      ;re-initialize txt pointer
    NextBufChar:
        mov     a, @r1
        call    writeChar
        inc r1
        cjne    @r1,    #null,  NextBufChar
;       cjne    @r1,    #0x0D,  NextBufChar     ;DEBUG
            mov     a,@r1
            call    writeChar
            ret
;------------------------------------------------;
```

Once the cipher text has been shown on the serial console, the file capture operation of hyperterminal can be stopped so the text is dumped. We need to have our program wait without display extra characters at this point in the program's execution. This will be accomplished with a subroutine named WaitForEnterkey which is defined as follows:

```
;----------- Wait For Enter Key ----------;
;Receives no parameters
;Loops until keyboard Enter key press is detected
;Returns nothing
WaitForEnterKey:
    jnb ri, $
        call    getchar
        cjne    a, #0x0d, WaitForEnterKey
        ret
;------------------------------------------------;
```

So after the cipher text has been captured to the output file, the enter key can be pressed and the program can continue to display another prompt and set up for performing other tasks. For this Activity a success message will be displayed. A complete program simulation is shown in the window below:



ACTIVITY 2: After the menu driven encryption program has been completed for ACTIVITY 1, a menu driven decryption program must be made as well.

The decryption algorithm is identical to the encryption algorithm except the rotate left instruction will be used instead of rotate right. Make the decryption algorithm in its own subroutine.

Use your Decryption program to decrypt the cipher text message generated by the program developed in Activity 1 of this lab.

Instead of making a separate decryption program, a menu option can be added at the beginning of the program so that Encryption or Decryption can be chosen. Implementation of this feature will be worth 10 bonus points in this lab.

ACTIVITY 3: Repeat a testing process similar to what was outlined in Lab 9 Activity3 but this time proof of decryption will be shown:

- Encrypt a plain text message of your choosing which must be at least 20 characters long and must be a legitimate message i.e. song, quote, statement, haiku, … not random characters using a **keyvals** array which contains 8 different bytes
- Use your program from ACTIVITY 2 of this lab to decrypt the cipher text.

Take screenshots of the following steps in the process:

- a screenshot showing the contents of your cipher text file
- a screenshot showing the decrypted plain text message displayed in Hyperterminal
- a screenshot showing the contents of your plain text file with NUL character

Deliverables:

- Completed and fully commented program source from Activity 1
- The screenshots specified in Activity 3
- Manual verification that the first 4 characters of your cipher text message were decrypted correctly with your keyvals. To represent this manual verification:
    - Convert each of the first four ascii characters to hex ascii encoded value
    - Convert the raw hex of the first four characters to binary
    - Rotate Left each binary quantity with the corresponding key byte to produce the binary decrypted value that represents the plain text character
    - Convert each binary plain text value to hex
    - Convert the hex value from the previous step to its corresponding ascii character

There is no demo portion for this lab