# **­­­­­Assignment 6**

**CS2340­­­**

**Encryption and Decryption**

**It is often important to be able to communicate with someone without anyone else being able to understand the message, should it fall into the wrong hands. Your task is to write an encryption and decryption program in MIPS assembly language. Here are the details:**

1. **The program must have a menu with three numbered options. 1: Encrypt the file. 2: Decrypt the file. 3. Exit. This implies that you have a loop that displays the menu and shows the three options until the user exits.**
2. **For options 1 and 2, request an input file name from the user. File names have a maximum length of 255 characters. If the file does not exist, show an error message and return to the main menu. If the file does exist, request the “key” from the user. This key is a string up to 60 characters long which will be used to encrypt or decrypt. If it is zero length, show an error message and return to the menu. (You can request the key before you open the file, to make things easier.)**
3. **For encryption, the extension of the input file name should be .txt. You will create an output file of the same name with the extension .enc. For decryption, your input name should have the extension .enc and you will create an output file with the same name and the extension .txt. Overwrite existing output files. You may assume that the extension is followed by the first period in the file name. For example, c:\mips\test\input.txt has the period just before the extension and there are no other periods in the string. However, remember that the input need not be actual text; that is only the extension. Your program should be able to encrypt anything, from a text file to an executable.**
4. **Apply the encryption or decryption algorithm, described below.**
5. **Return to the menu.**

**Encryption algorithm:**

1. **Read a block of up to 1024 bytes from the text file. If you read zero characters, you are done, close both files and return.**
2. **For each byte in the buffer, add the corresponding byte of the key using unsigned addition.**
3. **Store the new value back into the buffer.**
4. **If you come to the end of the key but still have characters to encrypt (you will know this because the character in the key is a \n) start over at the beginning of the key. Do not use the \n character to encrypt.**
5. **When you finish encrypting a block, write it to the output file, then return to step 1.**

**Decryption is identical to encryption except that instead of adding in step 2, subtract. I used the unsigned addition and subtraction instructions, so you must use those too.**

**Note: The extensions of the input files can be anything, but the extension of the output encrypted file is always the same as the input file with .enc as the extension. The extension of the output decrypted file is always .txt.**

**Good design means having separate functions for various pieces of this program. But do encryption and decryption need to be separate functions?**

**Test using the key “Computer Science” without the quotes. Note that capitalization is important here.**

**You will be provided two test files. One is plain text, which you are to encrypt. The other is encrypted using the key given, which you will decrypt, again using the given key. Do not worry that your encrypted file looks like garbage if you edit it with notepad.**

**If you hand in multiple files make sure you have .globl main and the label main: defined in the one in which execution starts.**

**To hand in through eLearning: A Zip file named CS2340-Asg6-<netID>.zip where you replace <netid> with your netID. For example, if I were to hand it in, the name would be CS2340-Asg6-jxc064000.zip. The names of your files within the Zip file are up to you but please make them meaningful. This must contain ALL files necessary to assemble and run your program except the MARS simulator. Also include the original test files I provided along with encrypted and decrypted test files created by your program.**

|  |  |
| --- | --- |
| **Grading Criteria** | |
| **Comments and variable names** | **15** |
| **Correctly encrypts the file** | **25** |
| **Correctly decrypts the file** | **25** |
| **Program structure and modularity** | **25** |
| **Does not crash for valid input** | **10** |
| **Total** | **100** |

**Additional grading criteria:**

1. **Does not correctly handle file extensions. -3**
2. **Does not correctly encrypt the test file. -25**
3. **Does not correctly decrypt the test file. -25**
4. **Spaghetti code. -2 to -15**
5. **Use of labels like “loop” and “while”. -1 to -5**

**#unfillfn:**

**# sb $zero,regfn($t1) #regfn: + counter2($t1) = 0**

**# sb $zero,encrypfn($t1) #encrypfn: + counter2($t1) = 0**

**# sb $zero,decrypfn($t1) #decrypfn: + counter2($t1) = 0**

**# addi $t1,$t1,1 #counter2($t1)+=1**

**# blt $t1,$t0,unfillfn #if counter2($t1)<counter($t0) branch to unfillfn:**

**# li $t1,0 #counter2($t1)=0**

**#unfillk:**

**# sb $zero,regfn($t1) #regfn: + counter2($t1) = 0**

**# sb $zero,encrypfn($t1) #encrypfn: + counter2($t1) = 0**

**# sb $zero,decrypfn($t1) #decrypfn: + counter2($t1) = 0**

**# sb $zero,key($t1) #decrypfn: + counter2($t1) = 0**

**# addi $t1,$t1,1 #counter2($t1)+=1**

**# blt $t1,$t0,unfillk #if counter2($t1)<counter($t0) branch to unfillfn:**