Ramzey Ghanaim

rghanaim@ucsc.edu

ID: 1395938

Lab 7: Shifters and Caesar Cipher

Due: 6/5/2015

CMPE-12/L: Section 5

Introduction

In this lab there were two tasks. The first task was to take the binary number generated from lab 6, and shift the bits to the right and left by a specified amount by the user. This involved the use of subroutines. This program had two additional procedures from lab 6. These were right shift and left ship. Lastly the value of the sifted binary number was printed. In part B of the lab I created an encryption algorithm called Creaser Cipher. This allows the user to input a string that can have the program decrypt or encrypt the string based off an offset cipher that the user also enters.

Part A

In this lab I simply took what I had in Lab 6 and expanded on it. The basic algorithm for shifting bits to the left was achieved by simply adding the bit to itself. If the user entered a shift of 3 bits, then the program added the number to itself three times to achieve the desired left shift. The right sift on the other hand was a bit more complicated. In order to get the bits to shift to the right I had to divide the number by two. Since there is no division on the LC-3 I had to keep track of how many times I could subtract two from the digit. If the subtraction ended up giving me a negative number I had to take away 1 from the counter. And after this, the counter was the final output. Once again if the user entered a shift of 3, I would repeat this division process three times, with the counter of the last calculation to be the number that is divided by 2 in the next iteration of the shift. Once the computation is complete the answer was outputted to the screen. Lastly, I wanted to note that the process for right shifting and left shifting were called upon as a subroutine. Therefore, each time I wanted to perform a right or left shift I just used a JSR to jump to the desired subroutine.

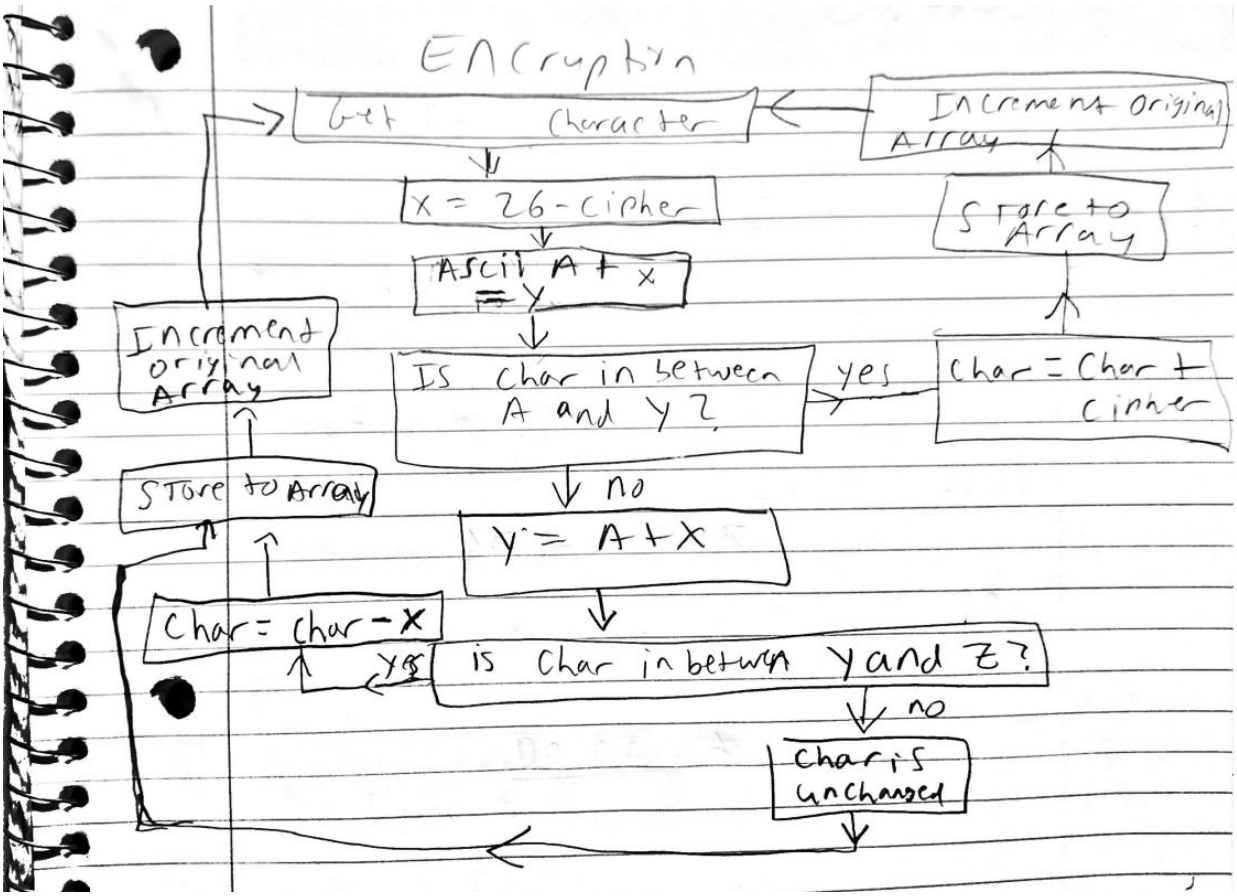
Part B

This part of the lab was slightly more complicated since it involved mapping a 2D array onto a 1D physical memory. I had to create a 2x200 array. I did this by creating an array with 400 elements, reserving the first 200 spots for the input string and the last 200 spots for the output strings. Each block of 200 memory lactations was considered as one row. Since I had two rows, I had 200*2 memory locations or 400. From here I started the program by asking the user to choose to decrypt or encrypt a string. The user then enters a number to determine how far to move each letter by. Then the user was able to enter a string. This string consists of up to 200 characters. This character limit is the result for the array size for the input and output being 200 each. My next step was now to either encrypt or decrypt the message. Like in part A I had two subroutines. One for encrypting the string and the other for decrypting it. For the encrypting and decrypting, each time the user entered a character, that character

was stored In the array, starting at the base address. I incremented this address in between each character input so I could store the next character as the next element of the array. Once the user hit the enter key, the program moves on to determine whether or not to encrypt or decrypt the given string, based off of what the user had earlier stated.

Encryption Algorithm

For the encryption algorithm I started by loading the string I then I have a bunch of checks to determine if the character the user entered was a capital letter, lowercase letter, or another character. Each character is represented as a number thanks to the ASCII standard. The capital alphabet is represented from numbers 65 through 90. And the lower case alphabet is represented from 97 to 122. I checked to see if the value in the address was somewhere in these interval listed above. If not, then the character is not a letter and its representation will be unchanged in the second row of the array. I computed this logic just by subtracting the upper and lower bounds of the interval and determined if the number was a letter or not based off of whether or not the result of the subtracted numbers was negative, positive or zero. Now for the actual algorithm of changing characters. This flow chart best demonstrates this:

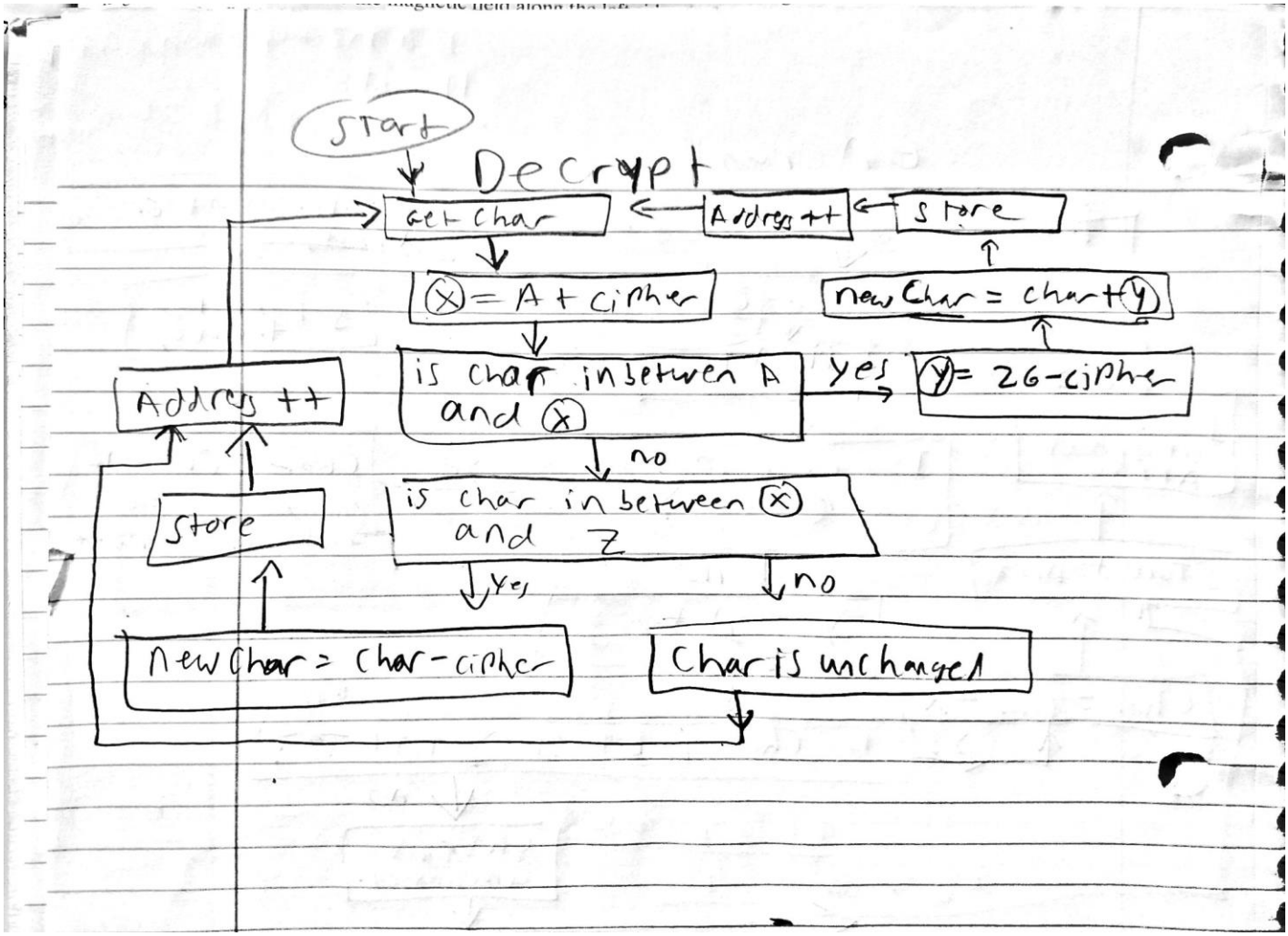


This algorithm is god because it allows me to shift each letter by a specified cipher without getting characters outside off the a-z ASCII codes. Therefore no random characters were generated. From here the storing of the new value back into the array is the same for both values. First, I took the address of the original character, and simply added 200 to it to get the value to be in the “same column” but in the

"second" virtual array. I quote around some of these words because there is not any real 2D mapping of a 2D array to memory. I then stored the new character to the location of the new address. This is computed for each character, incrementing the address every time, starting at the base address. After this, the second row is displayed on the screen with the correct encrypted value. Keep in mind this algoritm works with both capital and lower case letters. I just had to change the values of A and Z depending on if it was checking a capital or lower case letter.

Decrypting Algorithm

The algorithm for decrypting an input string is the same exact thing except, the algorithm will be different. Here is the flow chart for my algorithm:



From here the second row in the 2D array will be printed. This algorithm is similar to the encryption. It makes sure that the new character lies within the bounds of A and Z. Once again this same algorithm applies for capital and lower case letters, it's just that the values for A and Z are different, which I have stored from memory and load them into registers at the appropriate time.

Conclusion

For me, the main thing I learned from this lab was the ability to take complex ideas like arrays and dividing and shifting and convert them to simple mathematical representations of adding and subtracting (by adding). This lab also allowed me to understand and give me a hands on experience of how jumps work in the LC-3. I learned to save Register 7 whenever I was printing or changing the value of the register so I can return back to where the JSR came from.