

CPE 325: Intro to Embedded Computer System

Lab01

Caesar Cipher and Prime Factorization

Submitted by: Caleb Keller

Date of Experiment: 08/24/2022

Report Deadline: 08/24/2022

Demonstration Deadline: 08/25/2022

Introduction

Write a brief discussion on what the lab is about. (Use the tutorial and write in your own words. DO NOT copy text).

In the first part of this lab, I am implementing the Caesar cipher algorithm to encrypt a message using the programming language 'C' through CCS. The encrypted message shifts each upper and lowercase letter 3 times to the right. In the second part of the lab, I am finding the errors within given source code of a prime factorization program. The errors include a syntax error, and a logical error.

Theory

Write short notes on each topic discussed in lab.

Topic 1: Discuss the following tools/features from Code Composer Studio

- a) **Memory window:** The memory window in CCS shows how much memory is being allocated in specific memory locations.

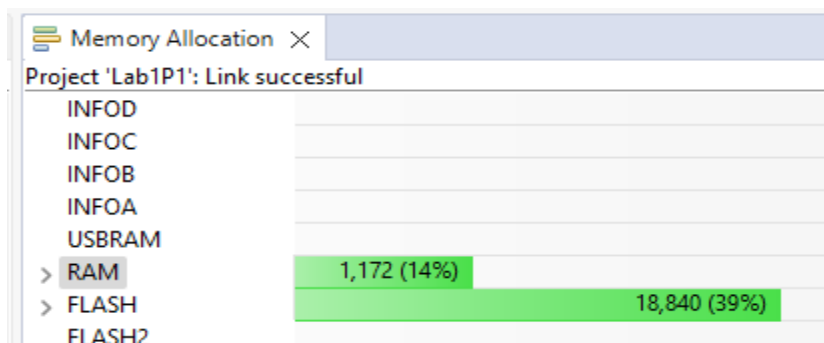


Figure #1: the memory window

- b) **Console Window:** The console window in CCS shows the results of the program after it is executed, as well as what is happening during execution.



Figure #2: the console window

- c) **Variable Window:** The variable window in CCS allows the user to see which variables the program is using, as well as their type, value, and location in memory. Using the step-over function allows the user to see the changes in variables' attributes line-by-line.

(x)= Variables × Expressions			
Name	Type	Value	Location
(x)= key	int	22	0x00441E
(x)= message	int	18817	0x004420
> msg	unsigned char[29]	[7 '\x07',18 '\x12',6 '\x06',18 '\x12'...	0x004400

Figure #3: the variable window

- d) **Breakpoints:** Breakpoints in CCS can be added to stop the program from running once it reaches the line of code that harbors the breakpoint. Breakpoints can be added on the left side of a line of code (in front of the line number). Breakpoints can be useful to allow users to see how a program executes up to a certain point.

```

25     return 0;
26 }
27
28
29 void Caesar(char* msg, int message, int key)
30 {
31     int i = 0;
32
33     for(i = 0; i <= message; i++)
34     {
35         if(msg[i] == ' ')
36         {
37             continue; // if it encounters a space, just keep going
38         }
39         if(msg[i] >= 'A' && msg[i] <= 'Z')

```

Figure #4: breakpoint example

Topic 2: What commands in Code Composer Studio can you use to run through your program?

- a) **Debug:** debug executes the program and reveals any warnings or errors that are found.



Figure #5: the debug button

- b) **Step Over:** step over allows user to step through program after debugging and is particularly useful when trying to pinpoint where an error in the code takes place.



Figure #6: the step over button

Results & Observation

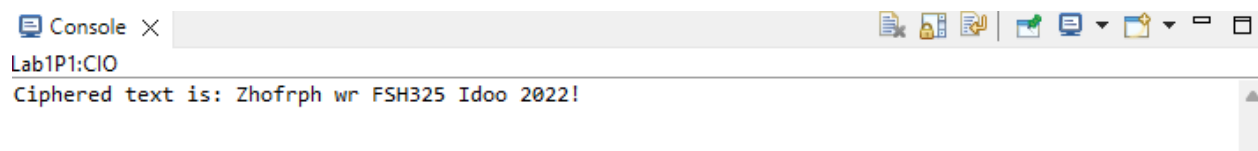
Program 1:

Program Description:

Explain your approach in solving the problem.

To perform the Caesar cipher, the program must go through the string of text and must change each character to another character in the ASCII code. To solve this problem, I began by creating the string and creating a variable to hold the string length. After, I created a function to implement the algorithm. Within the function is a for-loop that goes through each character of the string and changes the character depending on if the character is uppercase or lowercase (because capitalization matters in ASCII!). To avoid spaces, an if-statement determines if there is a space, and if so, it continues past it.

Program Output:

A screenshot of a console window titled "Console" with a close button. The window shows the output of a program. The first line is "Lab1P1:CIO" and the second line is "Ciphred text is: Zhofrph wr FSH325 Idoo 2022!". The console has a standard toolbar with icons for file operations and a scroll bar on the right.

```
Lab1P1:CIO
Ciphred text is: Zhofrph wr FSH325 Idoo 2022!
```

Figure #7: Program 1 output

Report Questions:

- 1. How many clock cycles does the code with a recursive function take to complete?**
Part 1: 14, 816 clock cycles according to breakpoint being set at end of main
- 2. How many clock cycles does the other implementation take?**
Part 2: 27, 890 clock cycles according to breakpoint being set at end of main

3. Explain the difference

The first program had only 1 recursive for loop, whereas the second program contains several while loops, as well as a recursive for loop which created a much larger clock cycle.

Program Flowchart:

Lab 1 P1

x Caleb Keller

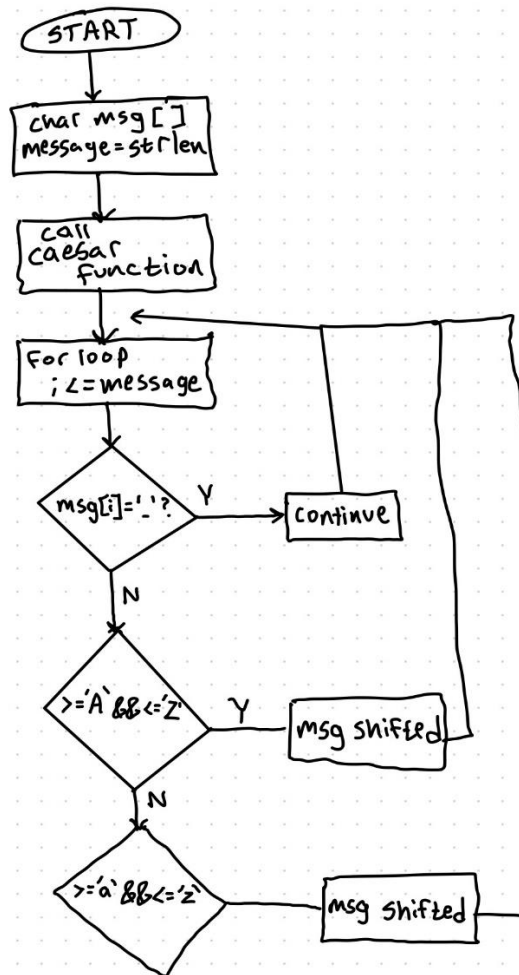


Figure #8: Program 1 Flowchart

Program 2:

Program Description:

Explain your approach in solving the problem.

For the syntax error, I began by building and debugging the program. When I did, the console window revealed a syntax error stating that a line in the file was missing a semicolon, in which I fixed immediately. The other issue was the logic error. I debugged the code and saw that it didn't print out all the factors. I scanned through the code until I found an arithmetic issue and corrected it. Once these two issues were fixed, the program ran correctly.

Program Output:

```
Console X
Lab1P2:CIO
Input value:
84
Output factors:
2 2 3 7
```

Figure #8: Program 2 output

Program Flowchart:

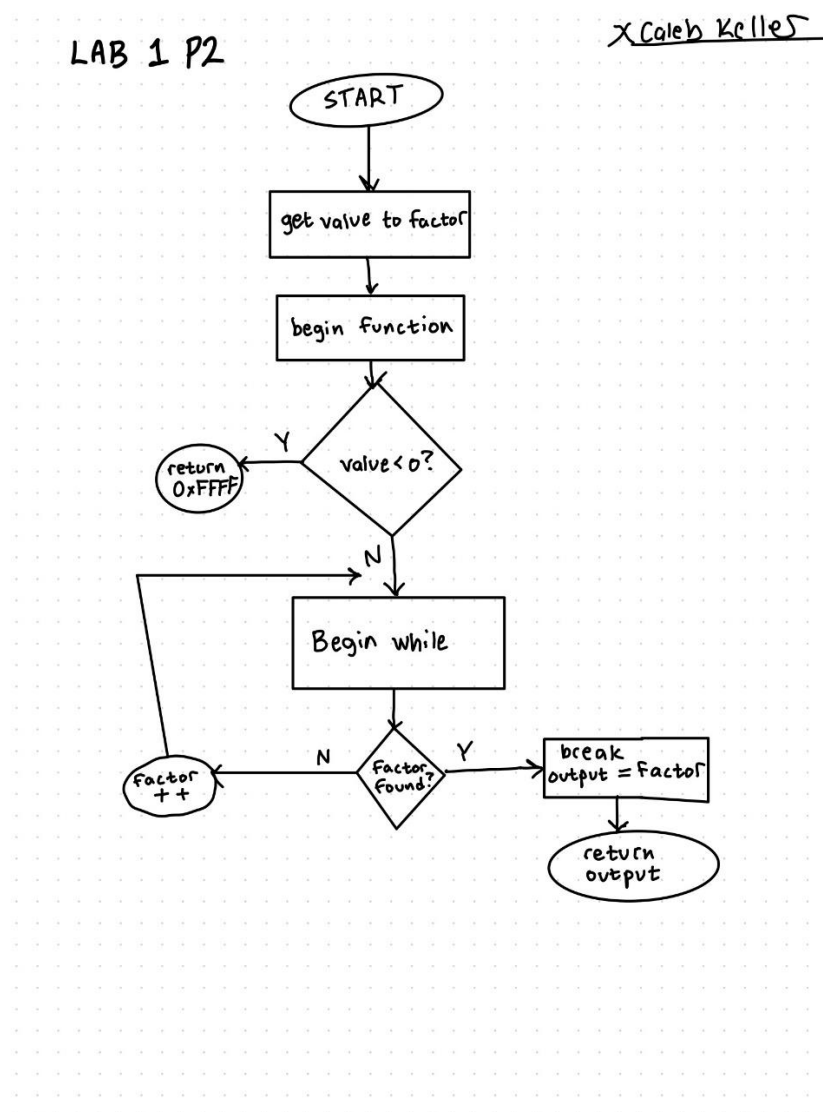


Figure #9: Program 2 Flowchart

Conclusion

While working through these programs, I encountered several issues in which I corrected. For example, I had an issue creating a loop to determine which ASCII character was in a string, and finally after trying to use a while loop—I implemented a for loop that fixed my program immediately! I learned numerous things about assembly language and C, as well as how to navigate CCS. On top of that, I have learned more about how microprocessors work, and how memory changes throughout run-time. Also, I am now more familiar with how many clock cycles a program performs as it executes a program.

Appendix

Your first code goes here, if any. Make sure you use a 1X1 table for this.

(Note: Make sure the code is readable, have comments. Also reduce spacing between lines to avoid lengthy reports.

Table ##: Program 1 source code

```
#include <msp430.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/*
 *
 * Title: Lab1 P1 Caesar Cipher encoding program
 * Author: Caleb Keller
 * Written: 08/23/2022
 * Description: This program encodes a given message by shifting each letter to the
right 3 times
 */

/**
 * main.c
 */

void Caesar(char* msg, int message, int key);

int main(void)
{
    WDTCTL = WDTPW | WDTHOLD; // stop watchdog timer

    const int key = 3; // key for shifting 3 times
    char msg[] = "Welcome to CPE325 Fall 2022!"; // hard-coded message to be
encoded
    int message = strlen(msg); // setting integer message to equal the length of
the message

    Caesar(msg, message, key); // calling caesar function to perform encoding

    return 0;
}

void Caesar(char* msg, int message, int key)
{
    int i = 0;

    for(i = 0; i <= message; i++) // for as long as it is <= end of the message
length
    {
        if(msg[i] == ' ')
        {
            continue; // if it encounters a space, just keep going
        }
        if(msg[i] >= 'A' && msg[i] <= 'Z')
```



```

    {
        msg[i] = ((msg[i] - 65 + key) % 26) + 65; // if the character is
        capital ASCII, perform cipher
    }
    if(msg[i] >= 'a' && msg[i] <= 'z')
    {
        msg[i] = ((msg[i] - 97 + key) % 26) + 97; // if the character is
        lowercase ASCII, perform cipher
    }
}
printf("%s", msg); // print the ciphered message
}

```

Your next code goes here, if any.

Table ##: Program 2 source code

```

/*-----
-
* Title: Lab1 P2
* Edited by Caleb Keller on August 23, 2022
*
* File:      Lab01_S2.c
* Description: This program finds the prime factorization of a hardcoded value
*
* Board:      5529
* Input:      Hardcoded short int number
* Output:     Factors of input value printed to console (ordered low to high)
* Author:     Douglas Marr
* Date:      August 16, 2022
*-----*/

#include <msp430.h>
#include <stdio.h>

#define true 1
#define false 0

// This function finds all of the factors in `value`
// .. Factors of `value` are output as elements of `factors`
// .. Function return value is number of factors found (0xFFFF for error)
int get_prime_factors(int *factors, int value)
{
    int output_factor_num = 0;
    int factor;
    char factor_found;

    if (value < 0)
        return 0xFFFF;

    // Loop while remaining value is not prime

```

```

while (true) {
    factor_found = false;
    factor = 2;

    // Get lowest remaining factor of `value`
    while (factor <= value) { // DELETED THE /2 FOUND HERE BECAUSE IT WAS
UNECESSARY
        if (value % factor == 0) { // Is `factor` a factor of `value`?
            // Factor found
            factor_found = true;
            break;
        }

        // Factor not found
        factor++;
    }

    if (!factor_found)
        break;

    // Add factor to array, and divide out of working value
    factors[output_factor_num] = factor;
    output_factor_num++;
    value /= factor;
}

return output_factor_num;
}

int main(void)
{
    WDTCTL = WDTPW | WDTHOLD; // stop watchdog timer

    // Input value for factorization
    const int INPUT_VALUE = 84;

    // Output array
    // Array must have at least as many elements as factors of INPUT_VALUE
    int prime_factors[10] = {};

    int num_factors;
    int i;

    // Calculate prime factors, and check for function error
    num_factors = get_prime_factors(prime_factors, INPUT_VALUE);
    if (num_factors == 0xFFFF)
        return 1;

    // Print input value & output prime factors separated by spaces
    printf("Input value: \n%d\n", INPUT_VALUE);
    printf("Output factors: \n");
    for (i = 0; i < num_factors; i++)
        printf("%d ", prime_factors[i]);

    printf("\n");

```

```
fflush(stdout);  
  
    return 0;  
}
```

Make sure to submit a single pdf file only