

CPE 325: Intro to Embedded Computer System

Lab01

Introduction to CCS features and functions.

Submitted by: Dan Otieno

Date of Experiment: 01/14/2022

Report Deadline: 01/17/2022

Demonstration Deadline: 01/17/2022.

Introduction

This Lab involves testing and learning the basics of TI Code Composer Studio (CCS) which we will use for all projects throughout the semester. In this report, I will explain some of the debugging tools/features in CCS (Memory window, Console window, Variable window, and Breakpoints) and provide source code and program analysis. This report will also contain a total of 3 source codes, two of which I will analyze in comparison, and one that will analyze a simple program by itself, along with their respective flowcharts.

Theory

Topic 1: SOME TOOLS/FEATURES OF CCS:

- a) **MEMORY WINDOW:** Can also be referred as the Memory browser view window. The memory window displays the contents of a target memory and can be accessed by clicking on the "View" option on the CCS menu bar across the top of the screen; However, this view is always open by default after debugging the program. Data in a specific memory address can be viewed in the window and will mostly be in hexadecimal format but can be changed to other data types. It is also worth noting that memory contents can be edited/changed by double-clicking them, and the window includes a tool bar on the top right corner to help with other instructions. To view data in a memory location, user can click on the location search space, type the memory they want to go, press "enter", and the contents will be displayed, as shown below:



Fig 1: A view of the memory browser window.

- b) **CONSOLE WINDOW:** The console window is also displayed when debugging the program in CCS. It shows details during the debug session, for instance the size of memory used, can also display program output when the run command is used (as in the case of printf() lines), errors and/or warnings, among other status information. This window also contains a toolbar for functions to help clear the contents, maximize, or minimize the window, word wrap to wrap long lines of text, other editing commands, among other instructions. An example below shows the console

view displaying the output from a printf() statement in a code:



Fig 2: Console View window.

- c) **VARIABLE WINDOW:** The variable window displays the variables in the program, usually the variables in a function (e.g., variables in main(void)). When stepping over a program, we can notice that some variables in the window will be highlighted in yellow as they change. Variables values can be modified in the window. There are four columns in the variable window, Name, Type, Value and Location. The Name column displays the assigned variable name, the Type column displays the data type, the Value column shows its value (for instance for x = 1, 1 would be the value for variable x), and the Location shows the memory location the variable is in, or a register. An example of the variable window is shown below:

A screenshot of the 'Variables' window in a debugger. It shows a table with four columns: Name, Type, Value, and Location. The variables listed are 'digits' (int, 20353, 0x004418), 'i' (int, 36, 0x00441A), 'lowers' (int, 34, 0x004416), 'uppers' (int, 20097, 0x004414), and 'x' (unsigned char[14], [4 '\x04', 18 '\x12', 49 '1', -128 '\x...', 0x004406]).

Name	Type	Value	Location
digits	int	20353	0x004418
i	int	36	0x00441A
lowers	int	34	0x004416
uppers	int	20097	0x004414
x	unsigned char[14]	[4 '\x04', 18 '\x12', 49 '1', -128 '\x...',	0x004406

Fig 3: Variable window view.

- d) **BREAKPOINTS:** Breakpoints are used to stop the execution of program instructions at a specific line in the source code. They are indicated by a blue dot or circle at the left hand side of the code editor, where the line numbers are displayed. They can be toggled on/off by right-clicking on the line number section on the left side of the editor. When a breakpoint is set, the debugger will check the debugger memory map to determine if the specified location is writable (RAM). If so, a software breakpoint will be used. If it is determined that the location is not writable (Flash,

ROM), then a hardware breakpoint will be used. An example is shown below:

```
31 {  
32     uppers++; // If found, upper case character counter  
33 }  
34 else if(x[i] >= 97 && x[i] <= 122) // ASCII character #s for lower case letters  
35 {  
36     lowers++; // If found, lower case character counter  
37 }  
38 } // End Loop.  
39  
40 printf("Hello CPE325! contains: \n %d digits \n %d uppercase characters \n %d lowercase characters \n",  
41 digits, uppers, lowers);  
42 return 0;
```

Fig 4: Breakpoint (Note the blue icon on line 36).

Results & Observation

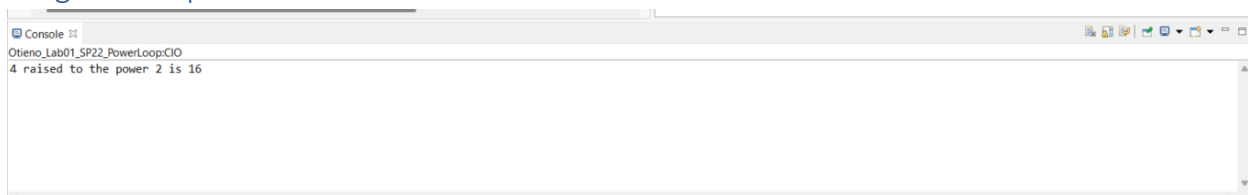
Program 1:

Program Description:

Explain your approach in solving the problem.

My approach towards solving this problem was to re-enact the same setup used in the D2 recursion code, that is, to setup my loop inside a function outside of main and then call the function in main to perform the intended instructions. In the source code, my function prototype is defined first, right below the headers, and then we have the main function, and then the exponential function, which holds the For Loop iteration with base and power as parameters of integer datatypes.

Program Output:




```
Console  
Otieno_Lab01_SP22_PowerLoop.CIO  
4 raised to the power 2 is 16
```

Figure 5: Console window output for program 1.

Report Questions:

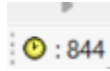
1. How many clock cycles does the code with a recursive function take to complete?

The Recursive function took 857 clock cycles to complete (printf() statement was commented

out before measuring the cycles).  : 857

2. How many clock cycles does the other implementation take?

The Loop function took 844 clock cycles to complete, similarly, printf() was commented out.



3. Explain the difference

The input values used to read these clock cycle measurements were a base of 4 and power of 10. The recursive function gives us a higher count because the function has to be recalled recursively, as opposed to the Loop.

Program Flowchart:

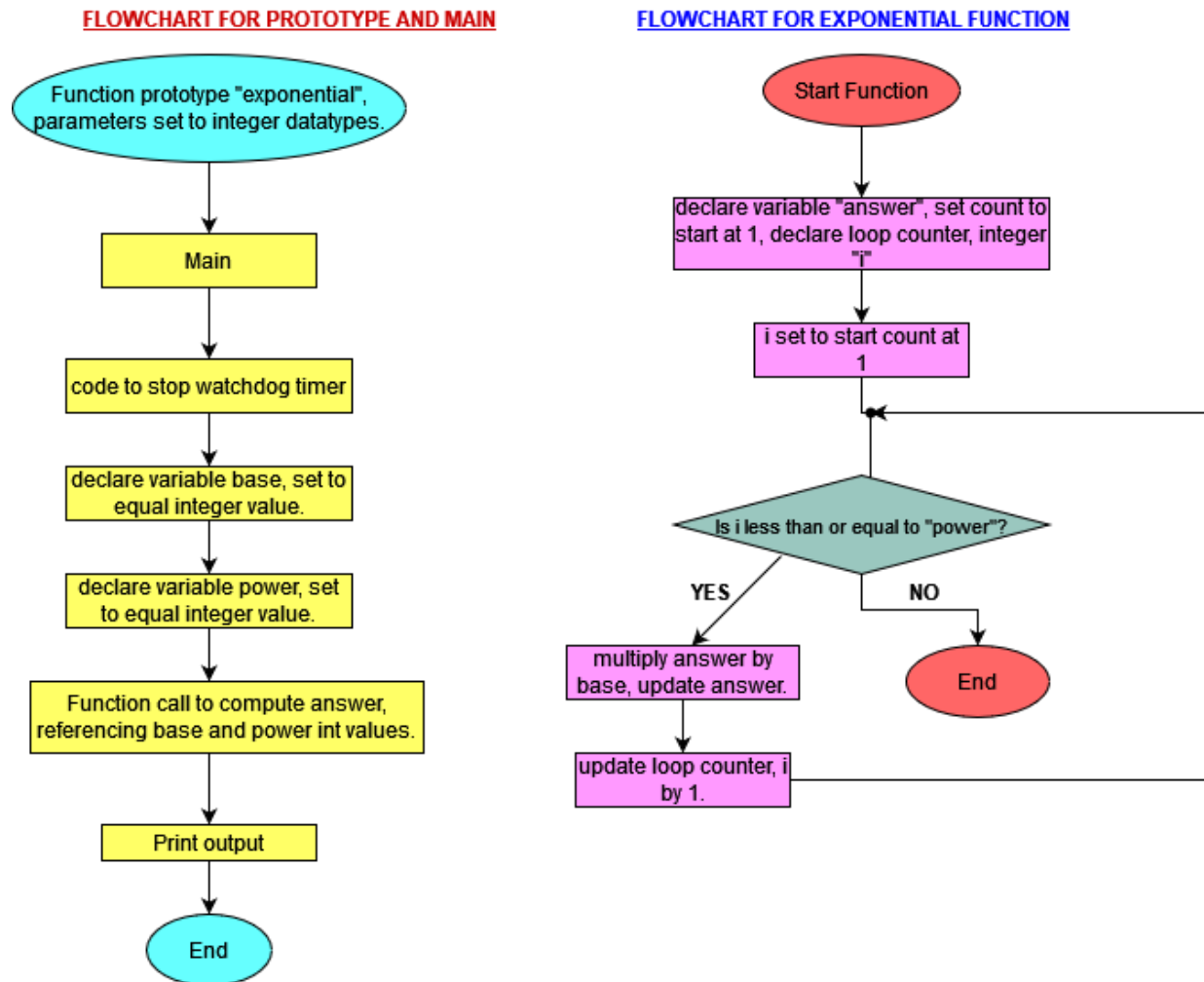


Figure 6: Flowcharts for Program 1.

Program 2:

Program Description:

For this problem, I decided to write everything inside main. I went with a “For” loop and If-Else statements. First, I set up my character counters for alphabetical characters and digits, and then set them to start counting at 0. Next, I defined a char type variable called “x” and set up an array to hold the characters in the string, and then compared ASCII number values corresponding to the characters using If-Else statements inside the loop, provided the conditions were met.

Program Output:

A screenshot of a console window titled "Console". The window displays the output of a program. The first line is "Otieno_Lab01_SP22_CharCountsC10". The second line is "Hello CPE325I contains:". The third line is "3 digits". The fourth line is "4 uppercase characters". The fifth line is "4 lowercase characters".

```
Console
Otieno_Lab01_SP22_CharCountsC10
Hello CPE325I contains:
3 digits
4 uppercase characters
4 lowercase characters
```

Figure 7: Output for program 2 as displayed in console window.

Program Flowchart:

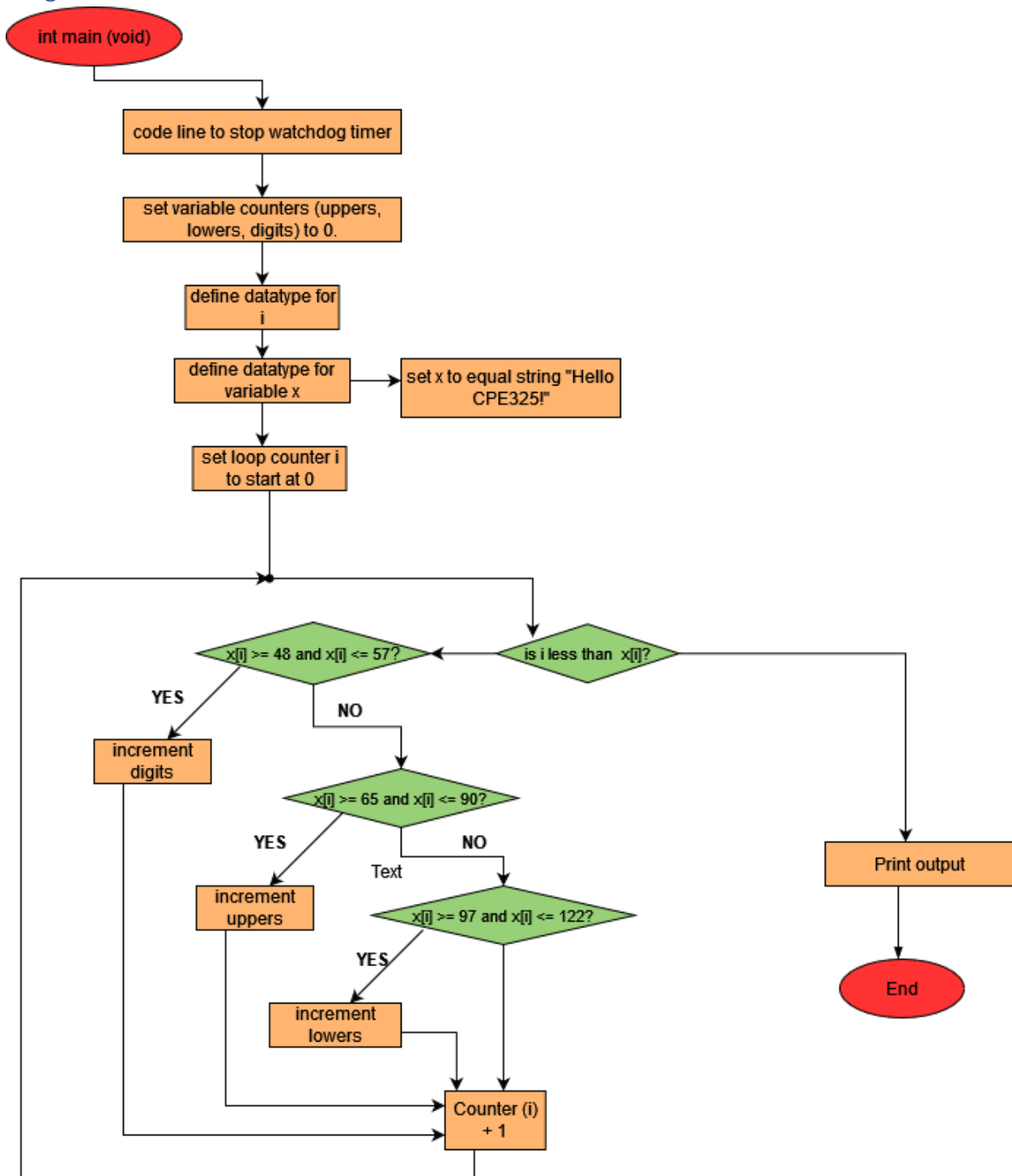


Figure 8: Flowchart for Program 2.

Conclusion

For my conclusion, it is worth mentioning that I was only able to get higher clock cycle counts for the recursive function in question 1 when I used higher values for the exponents. When I used lower values, my iteration loop always returned a higher clock cycle count. In any case, I was able to observe the differences between writing a program in two different ways, and learned that we can count characters in strings using loops and if-else statements comparing ASCII codes!!

Appendix

Table 1: Program 1 source code

```
/*-----  
 * File:      Lab01_PowerLoop.c  
 * Function:   This C code will calculate the power of a given base.  
 * Description: This program calculates the power of a given  
 *             base using iterative "For" loop.  
 * Input:      None.  
 * Output:     Base raised to the given power.  
 * Author(s):  Dan Otieno, dpo0002@uah.edu  
 * Date:       January 14th, 2022.  
 * -----*/  
  
#include <stdio.h>  
#include <math.h>  
#include "msp430.h"  
  
long int exponential(int, int); // Prototype of exponential  
int main(void)  
{  
    WDTCTL = WDTPW + WDTHOLD; // Stop watchdog timer.  
    int base = 4;             // Set base value.  
    int power = 2;            // Set power value.  
  
    long int answer = exponential(base, power); // Call exponential function in  
    main to calculate exponent by referencing variables.  
    printf("%d raised to the power %d is %d", base, power, answer); // Print  
    output.  
    return 0; // End program.  
}  
// This function uses an iterative loop to calculate the exponential of given base  
// and power values.  
long int exponential(int base, int power)  
{  
    long int answer = 1, i;  
    for(i = 1; i<=power; i++) //Loop to calculate exponent; condition is  
    counter must be less than or equal to power variable.  
    {  
        answer = answer*base; // If condition is met, calculation is  
    completed inside of loop.  
    }  
  
    return answer;  
}
```


Table 2: Program 2 source code

```

/*-----
 * File:      Lab01_CharCounts.c
 * Function:   This C code will count the letters and digits in a given string.
 * Description: This program counts and returns the number of alphabetical
characters and digits in a given string
 *             using ASCII character comparison method.
 * Input:      None.
 * Output:     Number of alphabetical characters (upper, lower case) and digits.
 * Author(s):  Dan Otieno, dpo0002@uah.edu
 * Date:       January 14th, 2022.
 * -----*/
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <msp430.h>

int main(void)
{
    WDTCTL = WDTPW | WDTHOLD; // stop watchdog timer.
    int uppers=0, lowers=0, digits=0; // Define variables for alphabetical
characters, digits and loop counter, set counters to start at 0.
    int i; // Define loop counter.
    char x[] = "Hello CPE325!"; // Define variable to hold the string
characters in an array.
    for(i=0; i<x[i]; i++) // Start of loop; loop counter starts at 0; compare
counter with length of character string.
    {
        if(x[i] >= 48 && x[i] <= 57) // ASCII character #s for digits; 48-
57; Program checks for characters in that range.
        {
            digits++; // If found, digit character counter is incremented
by 1.
        }
        else if(x[i] >= 65 && x[i] <= 90) // ASCII character #s for upper
case letters; 65-90; Program checks for characters in that range.
        {
            uppers++; // If found, upper case
character counter is incremented by 1.
        }
        else if(x[i] >= 97 && x[i] <= 122) // ASCII character #s for lower
case letters; 97-122; Program checks for characters in that range.
        {
            lowers++; // If found, lower case character counter is
incremented by 1.
        }
    } // End Loop.

    printf("Hello CPE325! contains: \n %d digits \n %d uppercase characters \n
%d lowercase characters", digits, uppers, lowers); // Print results.

    return 0;
}

```