CPE 325: Intro to Embedded Computer System

Lab01

Introduction to Code Composer Studio

Submitted by: Esther Shore

Date of Experiment: May 31, 2023

Report Deadline: June 6, 2023

Demonstration Deadline: June 12, 2023

**Theory Topics**

1. CCS Tools/Features
   1. Memory window

The memory window displays the contents stored in memory on the target device, which in our case is the MSP430. It is useful for monitoring what values are being read/written and where those values are stored in memory according to their address, particularly during debugging. By default, the addresses are given in hexadecimal. In the search box, a specific address can be entered to jump the view to begin at the given address. When stepping through a program, changed values show as red.

* 1. Console window

The console window displays readable text information such as build output, program output, or serial port information. It can be used to examine any build errors or view the results of any printf statements included in the code. This is where we output the results for the programs in this lab.

* 1. Variable window

The variable window displays and tracks the names, types, values, and location of variables as a program is executed. If applicable, the location column contains the memory addresses, which can be used in conjunction with the memory window to find exactly where that variable is being stored in memory.

* 1. Breakpoints

Breakpoints can be added when executing a program to essentially pause it at a specific line or state. This functionality is crucial for debugging and also allows for more specific stepping through code to inspect where an issue may be occurring. When used in conjunction with the variable and/or memory window, values at specific points in the code can be examined.

1. CCS Commands

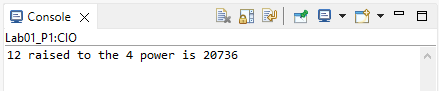
The major commands in Code Composer Studio that are used to run through a program are resume, step into, step over, and step return. Resume begins or resumes the execution from the current execution point indicated by an arrow in the left margin. It only stops when it reaches a breakpoint or finishes execution. Step into allows for stepping through a program line by line. Step over is similar to step into but treats functions as a single step rather than stepping into a function. Step return continues until the current function is done executing or until it returns.

**Program 1**

***Program Description:***

This program is intended to output the result of a given base raised to a given power using a for loop. The total clock cycles for program execution was also recorded to be compared with the provided recursive solution. To accomplish this task, an answer is initialized to 1 since it is a product. Then, a for loop is iterated power number of times to multiply the answer by the base, where the base and power are hardcoded. A flow of control can be viewed in Figure 2 and the code in Table 1 of the Appendix. The output for 12^4 as was specified in the demo code is displayed in Figure 1 below.

***Program Output:***



**Figure 1:** Program 1 Output

***Report Questions:***

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

For 12^4, the recursive solution took 17290 clock cycles. For 12^8, 19778 clock cycles. For 6^4, 16735 cycles.

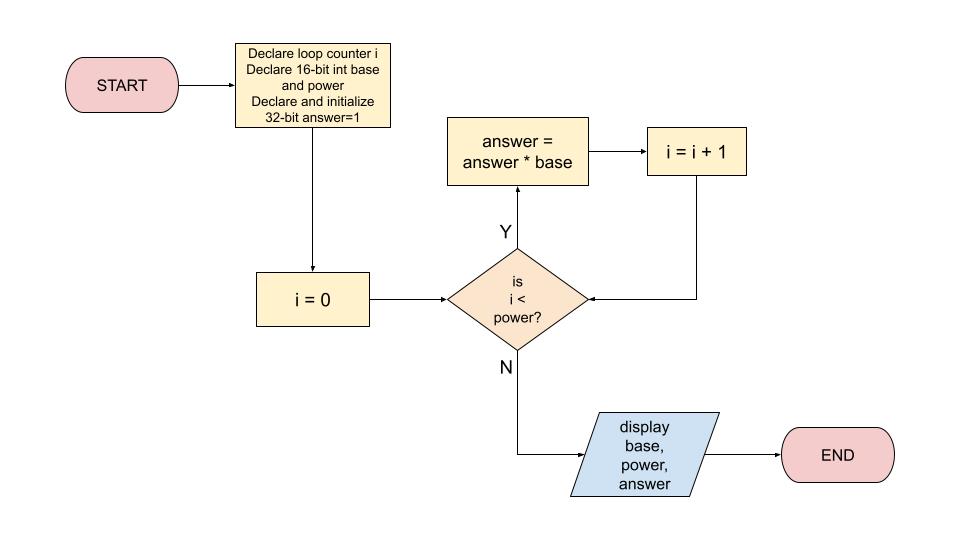
1. How many clock cycles does the other implementation take?

For 12^4, the for loop solution took 17236 clock cycles. For 12^8, 19620 clock cycles. For 6^4, 16681 cycles.

1. Explain the difference

The for loop implementation consistently took less clock cycles for the same high level operation because there is less overhead associated with the iterative solution compared the recursive solution involving multiple function calls. When the power was increased, the difference in clock cycles increased. When the base was changed, the difference in clock cycles remained unchanged. This makes sense because increasing the power, increases the number of recursive function calls that the sample solution takes, requiring greater stack manipulation and incurring more clock cycles than adding iterations to the for loop.

***Program Flowchart:***



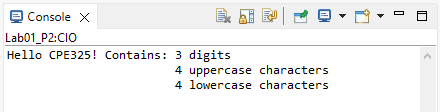
**Figure 2:** Program 1 Flowchart

**Program 2**

***Program Description:***

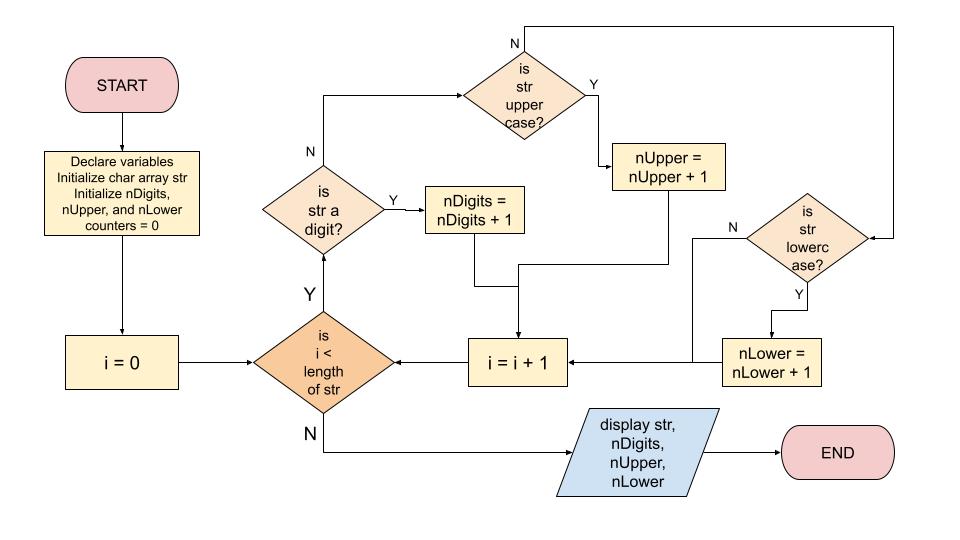
The goal of this program is to output the number of digits, uppercase letters, and lowercase letters in a string. In order to accomplish this, the functions isdigit, isupper, and is lower were used from the ctype.h header to characterize each character in the string. Iterating through a for loop based on the length of the string, if the character satisfies one of the conditions then the appropriate counter is incremented. The output for the provided string is shown in Figure 3 below, and the flowchart for this program is exhibited in Figure 4 below.

***Program Output:***



**Figure 3:** Program 2 Output

***Program Flowchart:***



**Figure 4:** Program 2 Flowchart

***Appendix:***

**Table 1:** Program 1 Source Code

| /\*------------------------------------------------------------------------------  \* File: Lab01\_P1.c  \* Function: This C code will calculate the power of a given base.  \* Description: This program calculates the power of a given  \* base using a for loop.  \* Input: None  \* Output: Base raised to the given power.  \* Author(s): Esther Shore  \* Date: June 6, 2023  \* ---------------------------------------------------------------------------\*/  #include <msp430.h>  #include <stdio.h>  int main() {  WDTCTL = WDTPW + WDTHOLD; // stop watchdog timer  int i; // step counter for for loop  int base = 4;  int power = 2; // exponent, must be greater than 1  long int answer = 1; // initialize product to 1  for (i = 0; i < power; i++) {  answer \*= base; // multiply answer by base  }  printf("%d raised to the %d power is %ld", base, power, answer);  return 0;  } |
| --- |

**Table 2:** Program 2 Source Code

| /\*------------------------------------------------------------------------------  \* File: Lab01\_P2.c  \* Function: Calculates num of digits, upper, and lower letters in a string.  \* Description: This program calculates the number of digits, uppercase, and  \* lowercase letters in a given string.  \* Input: None  \* Output: Console print results.  \* Author(s): Esther Shore  \* Date: June 6, 2023  \* ---------------------------------------------------------------------------\*/  #include <msp430.h>  #include <stdio.h>  #include <ctype.h> // isdigit, isupper, islower  #include <string.h> // strlen  int main() {  WDTCTL = WDTPW + WDTHOLD; // stop watchdog timer  int i; // loop counter  char str[] = "Hello CPE325!"; // given string declaration  int nDigits = 0;  int nUpper = 0;  int nLower = 0; // declare and initial counts  // for each char in str, increment appropriate counter  for (i = 0; i < strlen(str); i++) {  if (isdigit(str[i])) {  nDigits++;  } else if (isupper(str[i])) {  nUpper++;  } else if (islower(str[i])) {  nLower++;  }  }  printf("%s Contains: ", str);  printf("%d digits\n", nDigits);  printf("\t\t\t%d uppercase characters\n", nUpper);  printf("\t\t\t%d lowercase characters\n", nLower);  return 0;  } |
| --- |