CPE 325: Intro to Embedded Computer System

Lab02

Output specifiers, Array manipulation, Data-Types

Submitted by: Nolan Anderson

Date of Experiment: 09/06/2020

Report Deadline: 09/07/2020

Introduction

Lab two is a overview of how to output data correctly, a focus on the different data types, and array / matrix multiplication. Problem one has you output the data type, its size, along with its minimum and maximum value. Need to understand how to include headers, use output specifiers, and printf. The third problem covers a simple for loop to find the dot product of two, minimum 5 element array's.

Theory

Problem #1:

For problem #1, the main focus is to get more comfortable with printf format specifiers and brush up on the C data types, their sizes, as well as their minimum and maximum values. We also learn to better understand built in header files such as limits.h and float.h for things like INT_MIN. This also helps us to work with printf to make it output how we want to in a nice laid out format.

Problem #2:

For this problem, we are asked to compute the minimum and maximum value of a 2 byte data type. The objective of this is to learn the formulas for signed and unsigned data types to calculate sizes, and similarly how to convert from Bytes to bits.

Problem #3:

Lastly, problem #3 focuses on array's, for loops, data types, format specifiers, and array implementation. To get the output to function, it is necessary to understand how to declare an array, create a for loop, use format specifiers to correctly output the data, and how to do operations on array values.

Endianess

When looking at endianness, it is very important to know what your computer is running. There is little endian and big endian, and the MSP430 is little endian. What this means is that the most significant byte is stored at the highest address of the store region. Essentially, Big Endian would store it at 0A, and Little Endian would store it at 0D.

Size limits of data types

The size limit of a data type is very important when you are coding. For example, if you have limited memory, you would want to consider using the correct data type. I.e. using short int instead of long int for example 253. If you declared 253 as a long int, regardless, it is going to take up 4 bytes instead of just two on a short int. This will speed up your code and optimize it to work the best for the data you are working with.

Results & Observation

Copy the question from the assignment here:

Questions:

1. How are format specifiers used in your Q1 program?

For problem #1, I used format specifiers to correctly output my data for each data type. Using the correct format specifier is very important because if you use the wrong one, your data will display incorrectly. For example, I would not want to use %d for an unsigned number because it will output a negative number, which is not in the range of an unsigned number.

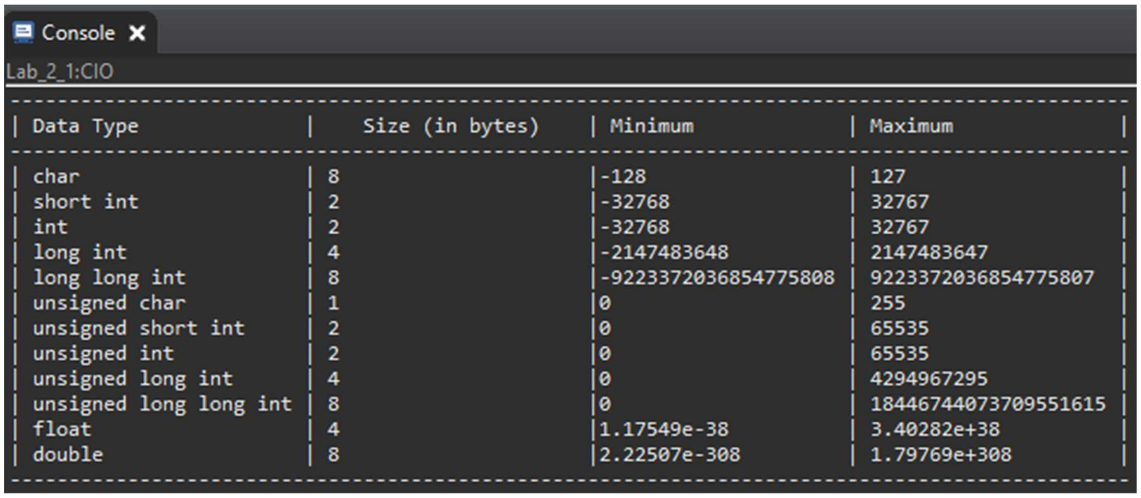
2. What is dot product? How do you implement dot-product in Q3?

The dot product is a way to find the sum of the product of two array's. To implement this, you must use a for loop. In this for loop, you multiple the current index of array x and array y. So, if i = 1, and your array is x = [1, 3] and y = [4, 5], then you would multiply 3 * 5 and add it to the running sum.

3. Show console output for both the questions Q1 and Q3.

Results Screenshots/Pictures:

Lab 2 Problem 1 console output:

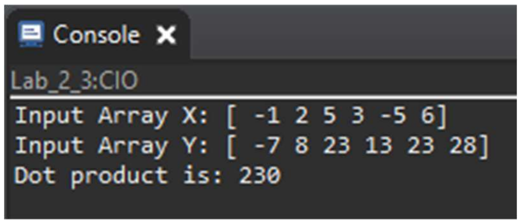


Console

Lab_2_1:CIO

```
---------------------------------------------------------------------------------------------
| Data Type              |    Size (in bytes)   | Minimum              | Maximum              |
---------------------------------------------------------------------------------------------
| char                   | 8                    |-128                  | 127                  |
| short int              | 2                    |-32768                | 32767                |
| int                    | 2                    |-32768                | 32767                |
| long int               | 4                    |-2147483648           | 2147483647           |
| long long int          | 8                    |-9223372036854775808  | 9223372036854775807  |
| unsigned char          | 1                    |0                     | 255                  |
| unsigned short int     | 2                    |0                     | 65535                |
| unsigned int           | 2                    |0                     | 65535                |
| unsigned long int      | 4                    |0                     | 4294967295           |
| unsigned long long int | 8                    |0                     | 18446744073709551615 |
| float                  | 4                    |1.17549e-38           | 3.40282e+38          |
| double                 | 8                    |2.22507e-308          | 1.79769e+308         |
---------------------------------------------------------------------------------------------
```

Lab 2, Problem 3 console output:



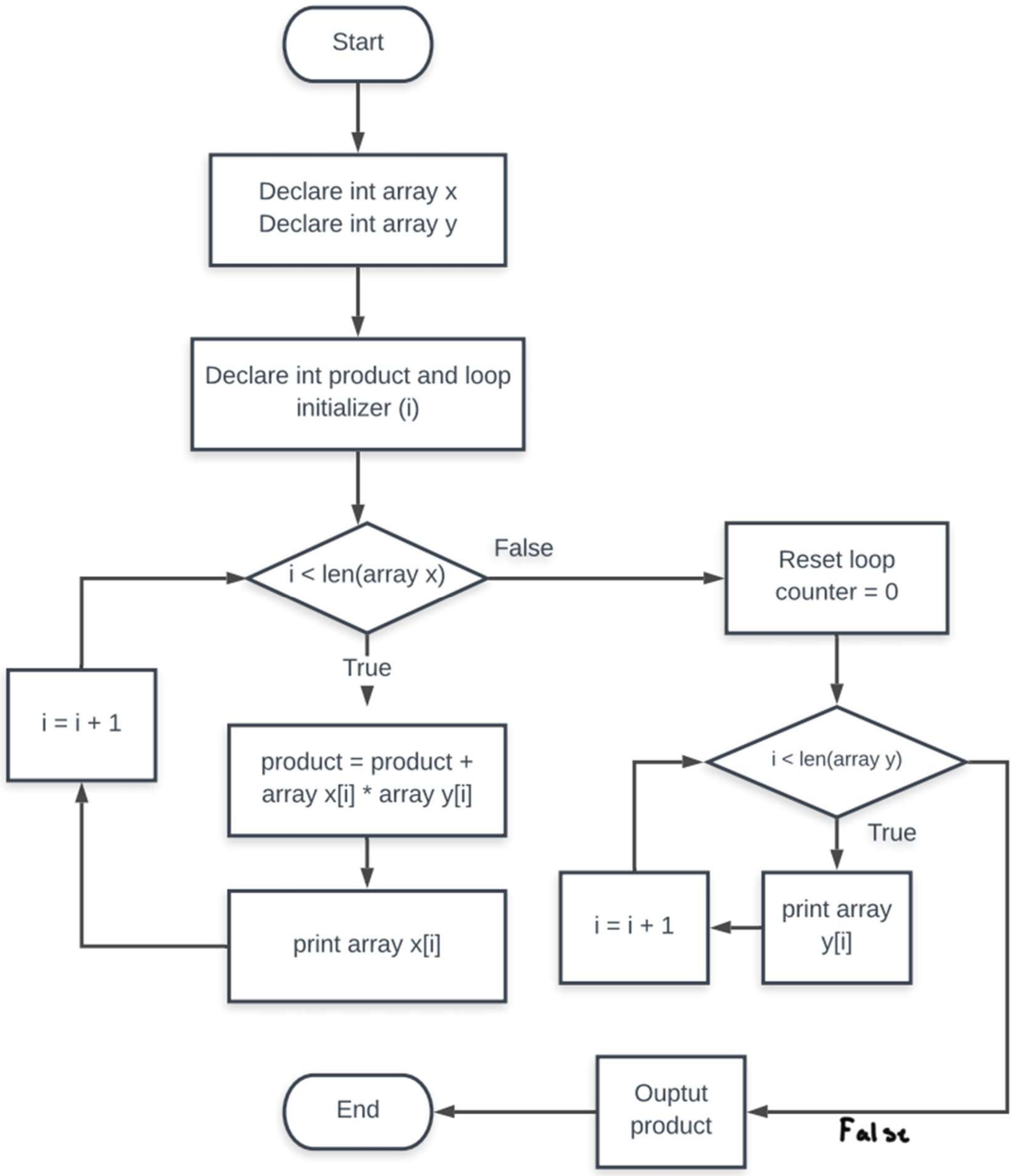
Console

Lab_2_3:CIO

```
Input Array X: [ -1 2 5 3 -5 6]
Input Array Y: [ -7 8 23 13 23 28]
Dot product is: 230
```

Flow Charts:

Lab 2, problem #3 flow chart.



Lab 2, Problem 2 written work:

Size of data type: 2 bytes

Size	Minimum	Maximum
Signed	-32,768	32,767
Unsigned	0	65,535

Signed

Min: $-1 \times 2^{N-1}$; $N=16 \rightarrow -1 \times 2^{15} = -32,768$

Max: $2^{N-1}-1$; $N=16 \rightarrow 2^{15}-1 = 32,767$

Unsigned

Min = 0

Max: $2^{N}-1 = 2^{16}-1 = 65,535$

Part 2: There are 4 that are 2 bytes.

1. Signed short
2. signed int
3. unsigned short
4. Unsigned int

} all four are the same range as I calculated here.

Observations:

-During this lab, I noticed how important output specifiers are and how diverse they can be. You cannot simply put %d or %u, there is usually a specific sequence of characters to get your output correct. It is also difficult and time consuming especially running in ccs.

-The size of data types is also important, especially to optimize your code well for speed. Coding with efficiency in mind should always be at the forefront because you never know what type of computer you will be running your code on. For our phones and PC's it is find, but something like the MSP430 and other embedded systems have low memory by design. Accounting for this is very important.

Conclusion

Overall, this lab was a lot more difficult than the first but the concepts were straight forward and easy to understand. The hardest part of this lab for me was just getting the output specifiers to work how I wanted them to. That took the majority of the time. Otherwise, I learned a lot from this lab and how data types work and how the MSP430 handles them in the c language.

Link to your video recordings/demo.

Direct video link:

https://drive.google.com/file/d/19yx7R57Np_j-kVVPV9kIxX3aIMWjQU-W/view?usp=sharing

Folder link:

https://drive.google.com/drive/folders/1_Y3ABMDhCUxc9phtQ8JKHCM8LDOK7k7J?usp=sharing

Appendix

Appendix 1

```c
/*------------------------------------------------------------------------------
* Student:      Nolan Anderson
* Program:      Lab_2_1.c
* Date:         Aug 21, 2121
* Input:        None
* Output:       Prints the sizes of common c data types
* Description:  This c program prints the sizes and ranges of common data types:
*               char, short int, int, long int, long long int, unsigned char,
*               unsigned short int, unsigned int, unsigned
                long int, unsigned long long int, float, and double.
*-----------------------------------------------------------------------------*/
#include <msp430.h>
#include <stdio.h>
#include <limits.h>
#include <float.h>
int main()
{
    WDTCTL = WDTPW + WDTHOLD;                               // Stop WatchDogTimer
    int unsignedmin = 0;
    printf("-----------------------------------------------------------------------------------------------\n");
    printf("| Data Type               |    Size (in bytes)    | Minimum             | Maximum              |\n");
    printf("-----------------------------------------------------------------------------------------------\n");
    printf("| char                   | %d                     |%-21hhi| %-21hhi|\n", CHAR_BIT, SCHAR_MIN, SCHAR_MAX);
    printf("| short int              | %d                     |%-21hd| %-21hd|\n",   sizeof(short int), SHRT_MIN, SHRT_MAX);
    printf("| int                    | %d                     |%-21d| %-21d|\n",     sizeof(int), INT_MIN, INT_MAX);
    printf("| long int               | %d                     |%-21ld| %-21ld|\n",   sizeof(long int), LONG_MIN, LONG_MAX);
    printf("| long long int          | %d                     |%-21lld| %-21lld|\n", sizeof(long long int), LLONG_MIN, LLONG_MAX);
    printf("| unsigned char          | %d                     |%-21hhu| %-21hhu|\n", sizeof(unsigned char), CHAR_MIN, UCHAR_MAX);
    printf("| unsigned short int     | %d                     |%-21hu| %-21hu|\n",   sizeof(unsigned short int), unsignedmin, USHRT_MAX);
    printf("| unsigned int           | %d                     |%-21u| %-21u|\n",     sizeof(unsigned int),unsignedmin, UINT_MAX);
    printf("| unsigned long int      | %d                     |%-21u| %-21lu|\n",    sizeof(unsigned long int), unsignedmin, ULONG_MAX);
    printf("| unsigned long long int | %d                     |%-21u| %-21llu|\n",   sizeof(unsigned long long int), unsignedmin, ULLONG_MAX);
    printf("| float                  | %d                     |%-21.2E| %-21.2E|\n", sizeof(float), -FLT_MAX, FLT_MAX);
    printf("| double                 | %d                     |%-21g| %-21g|\n",     sizeof(double), -DBL_MAX, DBL_MAX);
    printf("-----------------------------------------------------------------------------------------------\n");
    return 0;
}
```

Appendix 2

```c
/*------------------------------------------------------------------------------
* Student:      Nolan Anderson
* Program:      Lab_2_3.c
* Date:         Aug 21, 2121
* Input:        None
* Output:       Takes the dot product of two minimum 5 element arrays
* Description:  This program takes in two input arrays and performs a dot
*               product of the two. Then it outputs the dot product of the two
*               array's.
*-----------------------------------------------------------------------------*/
#include <msp430.h>
#include <stdio.h>
int main()
{
    WDTCTL = WDTPW + WDTHOLD;
    int array[] = {-1, 2, 5, 3, -5, 6};                 // First array
    int array2[] = {-7, 8, 23, 13, 23, 28};             // Second Array
    int product = 0;                                    // Product initializer
    int i = 0;                                          // Loop Initializer
    printf("Input Array X: [");                         // Printing out first statement
    for(i = 0; i < 6; i++)                              // Loop through the values
    {
        product = product + array[i] * array2[i];       // Perform the dot product on the two arrays
        printf(" %d", array[i]);                        // Output first array values
    }
    printf("]\nInput Array Y: [");                      // Output start for second array
    for(i = 0; i < 6; i++)
    {
        printf(" %d", array2[i]);                       // Output values for second array
    }
    printf("]\nDot product is: %d", product);           // Last line to output product.

    return 0;
}
```