CPE 325: Intro to Embedded Computer System

Lab02

Data Types

Submitted by: Esther Shore

Date of Experiment: May 31, 2023

Report Deadline: June 6, 2023

Demonstration Deadline: June 12, 2023

**Introduction**

This lab examines common data types in C for the MSP430. The size in bytes, minimum value, and maximum value are recorded for each data type to characterize and highlight the differences. The concept of endianness is discussed, and the use and manipulation of arrays is practiced.

**Theory Topics**

1. Different data types

There are 10 main data types examined in this lab: char, short int, int, long int, unsigned int, unsigned long int, long long int, unsigned long long int, float, and double. They differ in the typical number of bytes they occupy in memory as well as their possible range of values. For these reasons, different data types have varying use cases. The data type char is often used to represent ASCII characters whereas float is used to represent decimal numbers with limited precision.

1. Size limit of data types

Let n be the number of bits. A byte is 8 bits. For signed integers, the range goes from -2^(n-1) to 2^(n-1) - 1. For unsigned data types, the range goes from 0 to 2^n - 1. Long data types generally double the number of bytes occupied. The size and range of data types can vary based on language and hardware implementation.

1. Endianness

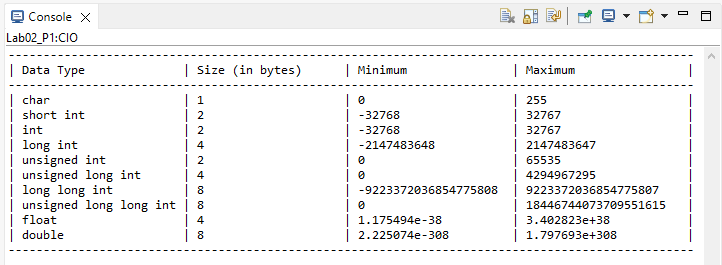
There are two primary types of endianness, which describes how multi-byte objects are stored in memory: Big Endian and Little Endian. The MSP430 uses Little Endian, meaning the least significant byte (LSB) is stored at a lower address or first in memory. Following the Big Endian policy, the most significant byte (LSB) would be stored at a lower address or first in memory.

**Program 1**

***Program Description:***

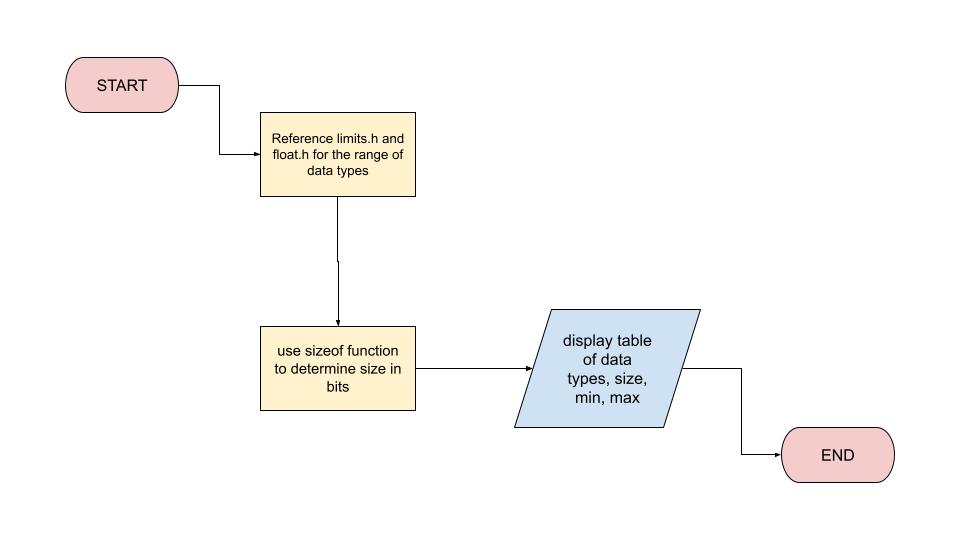
This program displays a table of data types with their size in bytes, minimum value, and maximum value. The sizeof function was used to determine the number of bytes for each data type. The constants from the limits.h and float.h header files were referenced to establish the minimum and maximum values. The flowchart is displayed in Figure 2. For unsigned data types, the minimum must be zero. The results can be seen below in Figure 1. For float and double, the minimums as defined in the header file were the normalized positive minimums, and scientific notation was used to improve readability. The results make sense following the theory topics discussed prior where unsigned gets rid of the sign bit and therefore increases the maximum by a power of two in comparison to its signed counterpart. Furthermore, long data types occupy twice as much space in memory and therefore exponentially expand the range.

***Program Output:***



**Figure 1:** Program 1 Output

***Program Flowchart:***



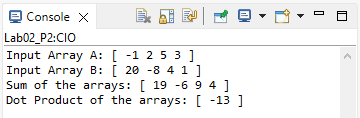
**Figure 2:** Program 1 Flowchart

**Program 2**

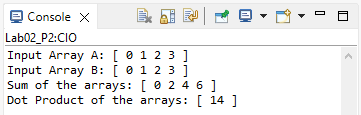
***Program Description:***

This program is designed to calculate and display resultant arrays for the sum and dot product of two given arrays. The key considerations while solving this question were the array size, calculating and storing the output arrays, and displaying the arrays. In order to maintain consistency, the array size was defined as a constant. Then, the sum and dot product were calculated in the same for loop, iterating through the elements at each index of the input arrays. The flowchart is exhibited in Figure 5. For the sake of cleaner and reusable code, a dedicated function to print an array of a specified size was also declared and utilized to display each of the four arrays. The given inputs result in the expected output as shown in Figure 3, but the program also works with other data values as shown in Figure 4. To verify the Figure 4 example, sum = {0+0 ,1+1, 2+2, 3+3} = {0,2,4,6} and dot product = 0\*0 + 1\*1 + 2\*2 + 3\*3 = 14.

***Program Output:***

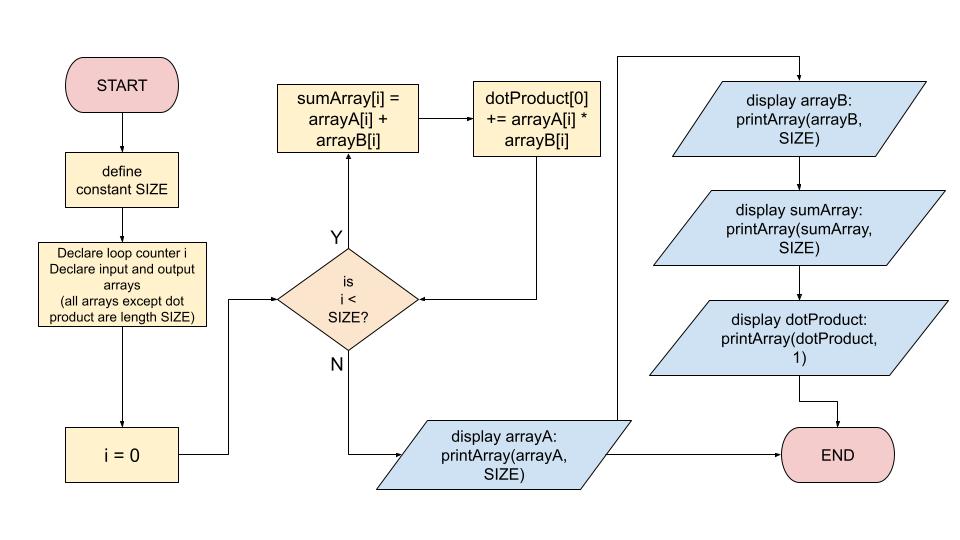


**Figure 3:** Program 2 Output

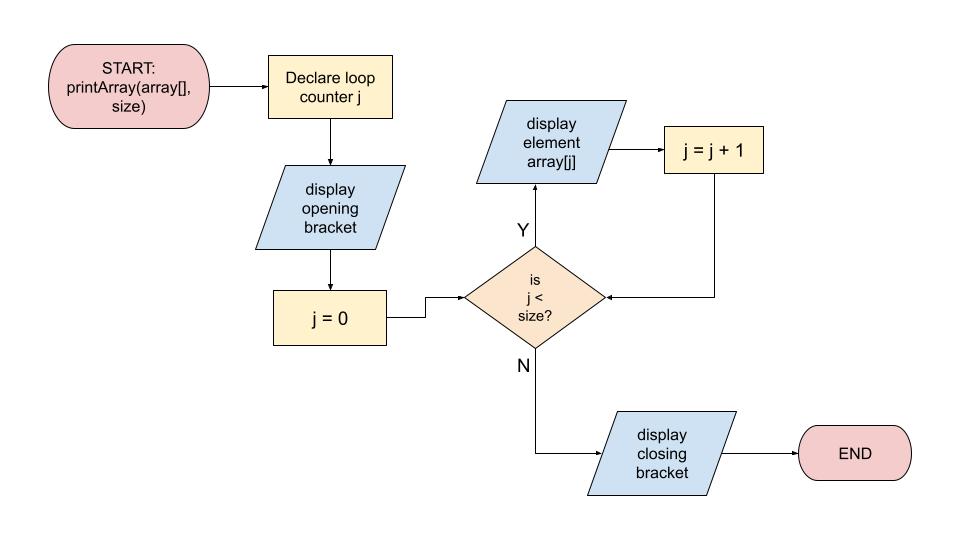


**Figure 4:** Program 2 Sample Output

***Program Flowchart:***



**Figure 5:** Program 2 Flowchart



**Figure 6:** Function printArray Flowchart

**Conclusion**

This lab introduced working with different data types and the differences between them. It also increased exposure and experience using Code Composer Studio as well as the C programming language on the MSP430. At first, some of the range numbers were off due to attempting to display all the values as standard integers. The problem was fixed by matching the data types to reflect the number trying to be displayed accordingly. For instance, one warning dealt with using the sizeof function, which returns a type size\_t and corresponds to the string format specifier %zu. Another development issue encountered was the last printf statement seemingly not outputting to the console until resume was pressed again after reaching the breakpoint at the end of main. This issue was resolved by ensuring a newline character at the end of each line to flush the output buffer.

***Appendix:***

**Table 1:** Program 1 Source Code

| /\*------------------------------------------------------------------------------  \* File: Lab02\_P1.c  \* Function: Determine the size and range of data types in C.  \* Description: This program determines the size in bytes, minimum value, and  \* maximum value for common data types in C using definitions  \* given in the limits.h and float.h header files. Displays the  \* results in table format.  \* Input: None  \* Output: Console print statements.  \* Author(s): Esther Shore  \* Date: June 6, 2023  \* ---------------------------------------------------------------------------\*/  #include <msp430.h>  #include <stdio.h>  #include <limits.h>  #include <float.h>  int main() {  WDTCTL = WDTPW + WDTHOLD; // stop watchdog timer  printf("--------------------------------------------------------------------------------------------------\n");  printf("| Data Type | Size (in bytes)\t| Minimum\t\t| Maximum\t\t |\n");  printf("--------------------------------------------------------------------------------------------------\n");  printf("| char | %zu\t\t\t| %d\t\t\t| %d\t\t\t |\n", sizeof(char), CHAR\_MIN, CHAR\_MAX);  printf("| short int | %zu\t\t\t| %d\t\t| %d\t\t\t |\n", sizeof(short int), SHRT\_MIN, SHRT\_MAX);  printf("| int | %zu\t\t\t| %d\t\t| %d\t\t\t |\n", sizeof(int), INT\_MIN, INT\_MAX);  printf("| long int | %zu\t\t\t| %ld\t\t| %ld\t\t |\n", sizeof(long int), LONG\_MIN, LONG\_MAX);  printf("| unsigned int | %zu\t\t\t| %d\t\t\t| %u\t\t\t |\n", sizeof(unsigned int), 0, UINT\_MAX);  printf("| unsigned long int | %zu\t\t\t| %d\t\t\t| %lu\t\t |\n", sizeof(unsigned long int), 0, ULONG\_MAX);  printf("| long long int | %zu\t\t\t| %lld\t| %lld\t |\n", sizeof(long long int), LLONG\_MIN, LLONG\_MAX);  printf("| unsigned long long int | %zu\t\t\t| %d\t\t\t| %llu\t |\n", sizeof(unsigned long long int), 0, ULLONG\_MAX);  printf("| float | %zu\t\t\t| %e\t\t| %e\t\t |\n", sizeof(float), FLT\_MIN, FLT\_MAX);  printf("| double | %zu\t\t\t| %e\t\t| %e\t\t |\n", sizeof(double), DBL\_MIN, DBL\_MAX);  printf("--------------------------------------------------------------------------------------------------\n");  return 0;  } |
| --- |

**Table 2:** Program 2 Source Code

| /\*------------------------------------------------------------------------------  \* File: Lab02\_P2.c  \* Function: Calculate the sum and dot product of two given arrays.  \* Description: This program creates a sum array and dot product array for two  \* given arrays and displays the results. A printArray function  \* is used to condense repeated code.  \* Input: None  \* Output: Console print statements of arrays.  \* Author(s): Esther Shore  \* Date: June 6, 2023  \* ---------------------------------------------------------------------------\*/  #include <msp430.h>  #include <stdio.h>  #define SIZE 4  void printArray(int[], int); // function prototype for printArray  int main() {  WDTCTL = WDTPW + WDTHOLD; // stop watchdog timer  int i; // declare loop counter  int arrayA[SIZE] = {-1, 2, 5, 3};  int arrayB[SIZE] = { 20, -8, 4, 1}; // declare and initialize input arrays  int sumArray[SIZE];  int dotProduct[1] = {0}; // declare output arrays, initialize dotProduct bc it's a sum  for (i = 0; i < SIZE; i++) {  sumArray[i] = arrayA[i] + arrayB[i]; // sum corresponding elements  dotProduct[0] += arrayA[i] \* arrayB[i]; // add product of corresponding elements  }  printf("Input Array A: ");  printArray(arrayA, SIZE);  printf("Input Array B: ");  printArray(arrayB, SIZE);  printf("Sum of the arrays: ");  printArray(sumArray, SIZE);  printf("Dot Product of the arrays: ");  printArray(dotProduct, 1);  return 0;  }  void printArray(int array[], int size) {  int j; // declare loop counter  printf("[ ");  for (j = 0; j < size; j++) {  printf("%d ", array[j]); // print each element of the array  }  printf("]\n");  } |
| --- |