CPE 325: Intro to Embedded Computer System

Lab02

Laboratory Assignment #2

Submitted by: Anshika Sinha

Date of Experiment:____1/15/2025__________________

Report Deadline:________1/21/2025_________________

Demonstration Deadline: _____1/27/2025_____________

Theory

Topic 1: Different data types

a) Char: char is used to store a single character, like 'A' or 'z'. It takes 1 byte of memory.

b) Int: int is used to store positive and negative whole numbers, such as 42 or -5. Its size is usually 4 bytes.

c) Float: float is used for decimal numbers with single precision. It typically takes 4 bytes.

d) Double: double provides twice the precision as float, allowing for more accuracy with numbers like 3.14159265359. It usually uses 8 bytes.

Topic 2: Size limit of data types: The size of a data type determines the range of values it can store.

a) A char is 1 byte and can hold values from -128 to 127, or from 0 to 255 if unsigned.
b) An int is typically 4 bytes, so it can store values between -2,147,483,648 and 2,147,483,647 , or double that range for unsigned numbers.
c) A float, being 4 bytes, can represent decimal numbers approximately between $\pm 3.4 \times 10^{-38}$ and $\pm 3.4 \times 10^{38}$.
d) A double uses 8 bytes, so its range is larger, roughly $\pm 1.7 \times 10^{-308}$ to $\pm 1.7 \times 10^{308}$.

Topic 3: Endianness: Endianness describes the order in which bytes are stored in memory. It is important when data is shared between systems with different memory storage orders.

a) Big Endian: In big-endian systems, the most significant byte is stored first at the lowest memory address.
b) Little Endian: In little-endian systems, the least significant byte is stored at the lowest memory address.

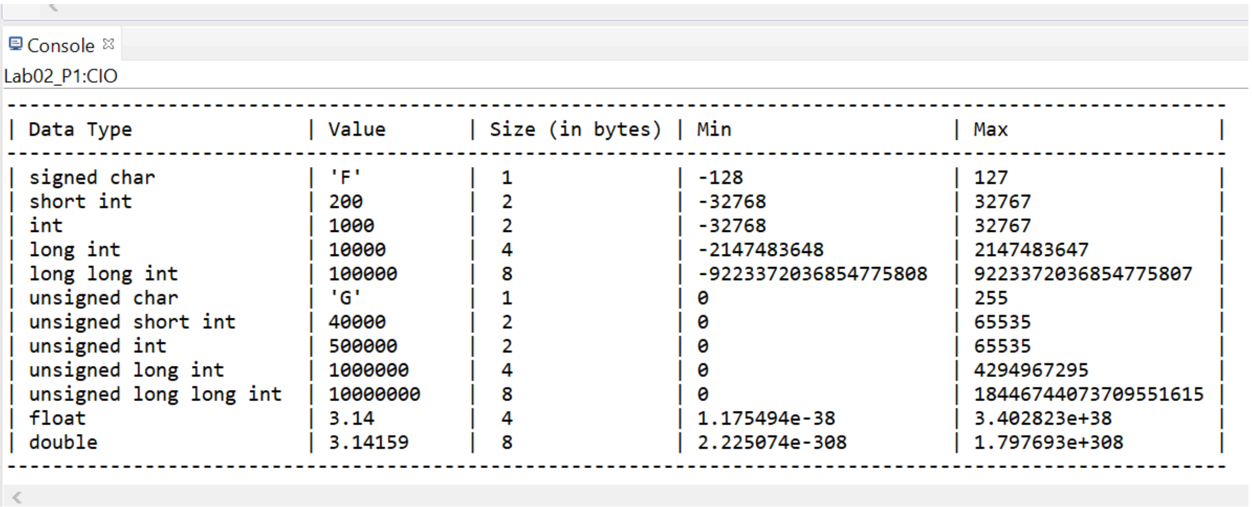
Results & Observation

Program 1:

Program Description:

This C program displays the sizes and ranges of various common data types in a tabular format. It uses the standard libraries limits.h and float.h to retrieve predefined constants for the minimum and maximum values of both integral and floating-point types, ensuring portability across different platforms. The table covers data types such as 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. For each data type, the program displays its size in bytes (determined using the sizeof operator), and its range, including minimum and maximum values.

Program Output:



```
-------------------------------------------------------------------------------------------------------------
| Data Type                | Value       | Size (in bytes) | Min                    | Max                    |
-------------------------------------------------------------------------------------------------------------
| signed char              | 'F'         |  1              | -128                   | 127                    |
| short int                | 200         |  2              | -32768                 | 32767                  |
| int                      | 1000        |  2              | -32768                 | 32767                  |
| long int                 | 10000       |  4              | -2147483648            | 2147483647             |
| long long int            | 100000      |  8              | -9223372036854775808   | 9223372036854775807    |
| unsigned char            | 'G'         |  1              | 0                      | 255                    |
| unsigned short int       | 40000       |  2              | 0                      | 65535                  |
| unsigned int             | 500000      |  2              | 0                      | 65535                  |
| unsigned long int        | 1000000     |  4              | 0                      | 4294967295             |
| unsigned long long int   | 10000000    |  8              | 0                      | 18446744073709551615   |
| float                    | 3.14        |  4              | 1.175494e-38           | 3.402823e+38           |
| double                   | 3.14159     |  8              | 2.225074e-308          | 1.797693e+308          |
-------------------------------------------------------------------------------------------------------------
```

Figure 01: Program 1 Output

Program 2:

Program Description:

Compute the maximum and minimum values of a data-type whose size is 4 bytes by hand. Perform this computation considering the data-type to be (a) unsigned and (b) signed.

Maximum and minimum values of a 4 byte data-type:

(a) unsigned:

Maximum = $2^n - 1 = 2^{32} - 1 = 4294967295$

Minimum = 0

(b) signed:

Maximum = $2^{n-1} - 1 = 2^{32-1} - 1 = 2147483647$

Minimum = $-2^{n-1} = -2^{n-1} = -2147483648$

In the list of data types that you printed in Q1, which data types are 4-bytes. Does your maximum and minimum values match with your output in Q1?

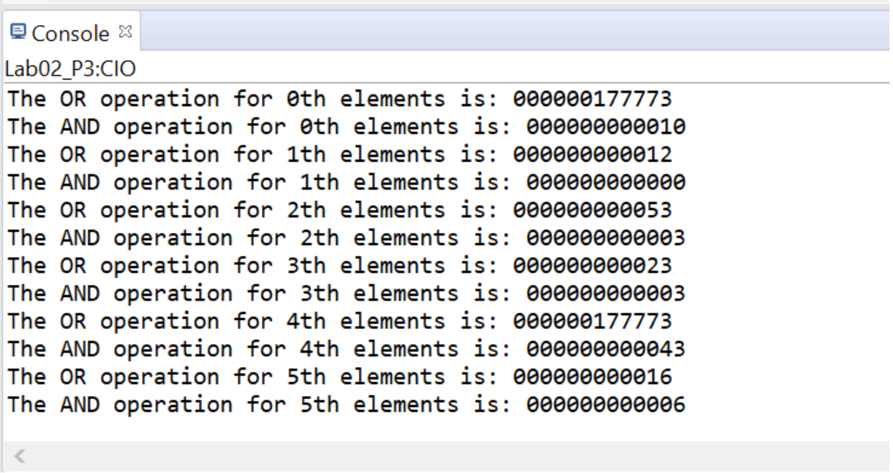
From Q1, long int, unsigned long int, and float are 4 bytes in size. Yes, the maximum and minimum values match with the output in Q1.

Program 3:

Program Description:

This C program performs bitwise operations on two arrays of integers initialized with hexadecimal values. The two arrays, X and Y, are defined with six elements each, including both positive and negative hexadecimal integers to test a variety of cases. A loop iterates through the arrays, performing bitwise OR and AND operations on corresponding elements, and the results are formatted in octal notation.

Program Output:



```
The OR operation for 0th elements is: 000000177773
The AND operation for 0th elements is: 000000000010
The OR operation for 1th elements is: 000000000012
The AND operation for 1th elements is: 000000000000
The OR operation for 2th elements is: 000000000053
The AND operation for 2th elements is: 000000000003
The OR operation for 3th elements is: 000000000023
The AND operation for 3th elements is: 000000000003
The OR operation for 4th elements is: 000000177773
The AND operation for 4th elements is: 000000000043
The OR operation for 5th elements is: 000000000016
The AND operation for 5th elements is: 000000000006
```

Figure 02: Program 3 Output

Program 3 Flowchart:

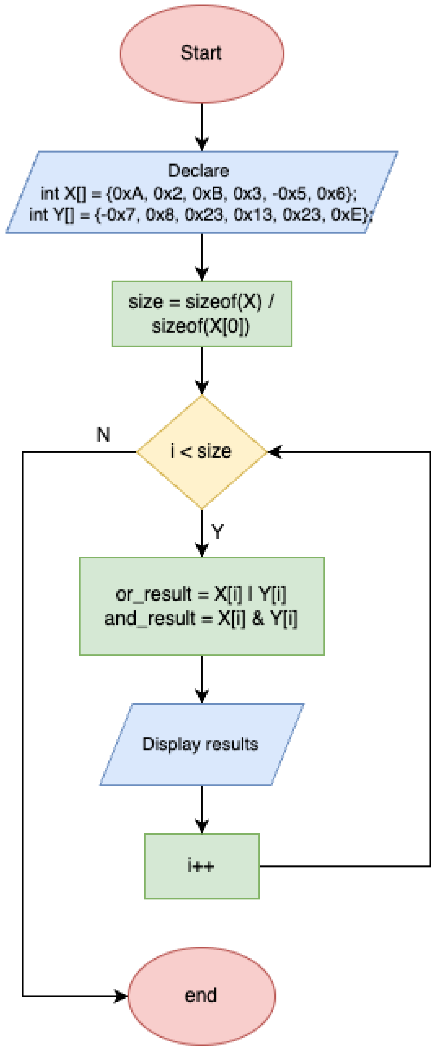

Start
Declare
int X[] = {0xA, 0x2, 0xB, 0x3, -0x5, 0x6};
int Y[] = {-0x7, 0x8, 0x23, 0x13, 0x23, 0xE};
size = sizeof(X) / sizeof(X[0])
N
i < size
Y
or_result = X[i] I Y[i]
and_result = X[i] & Y[i]
Display results
i++
end

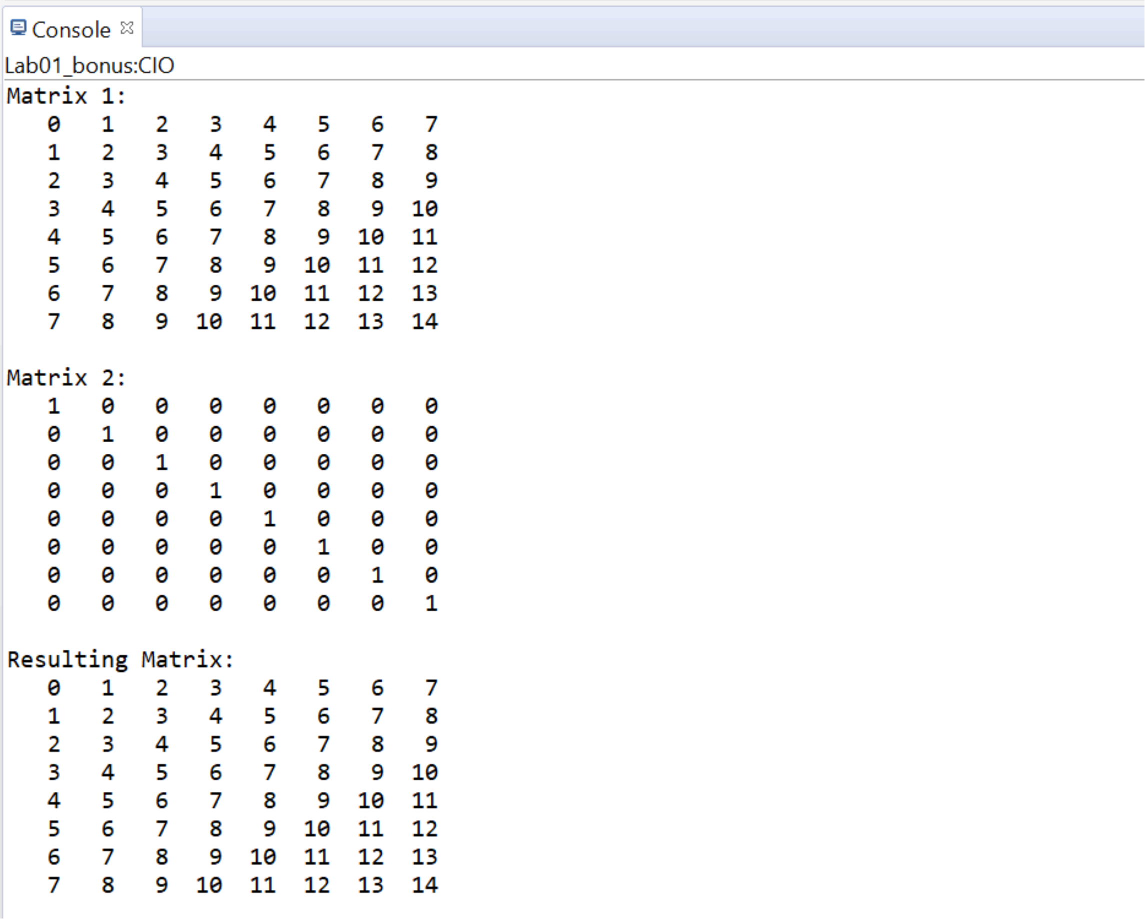
Figure 03: Program 3 Flowchart

Bonus Program:

Program Description:

This C program performs matrix multiplication on two 8x8 matrices and displays the input matrices and their resultant matrix. It initializes the first matrix with values based on the sum of row and column indices and the second matrix as an identity matrix. Using nested loops, the program calculates the product of the matrices, storing the results in a new matrix. Modular functions are included for displaying matrices (displayMatrix) and performing the multiplication (multiplyMatrices). The program highlights fundamental matrix operations and outputs all matrices in a formatted layout.

Program Output:



```
Matrix 1:
   0   1   2   3   4   5   6   7
   1   2   3   4   5   6   7   8
   2   3   4   5   6   7   8   9
   3   4   5   6   7   8   9  10
   4   5   6   7   8   9  10  11
   5   6   7   8   9  10  11  12
   6   7   8   9  10  11  12  13
   7   8   9  10  11  12  13  14

Matrix 2:
   1   0   0   0   0   0   0   0
   0   1   0   0   0   0   0   0
   0   0   1   0   0   0   0   0
   0   0   0   1   0   0   0   0
   0   0   0   0   1   0   0   0
   0   0   0   0   0   1   0   0
   0   0   0   0   0   0   1   0
   0   0   0   0   0   0   0   1

Resulting Matrix:
   0   1   2   3   4   5   6   7
   1   2   3   4   5   6   7   8
   2   3   4   5   6   7   8   9
   3   4   5   6   7   8   9  10
   4   5   6   7   8   9  10  11
   5   6   7   8   9  10  11  12
   6   7   8   9  10  11  12  13
   7   8   9  10  11  12  13  14
```

Figure 04: Bonus program output

Report Questions:

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

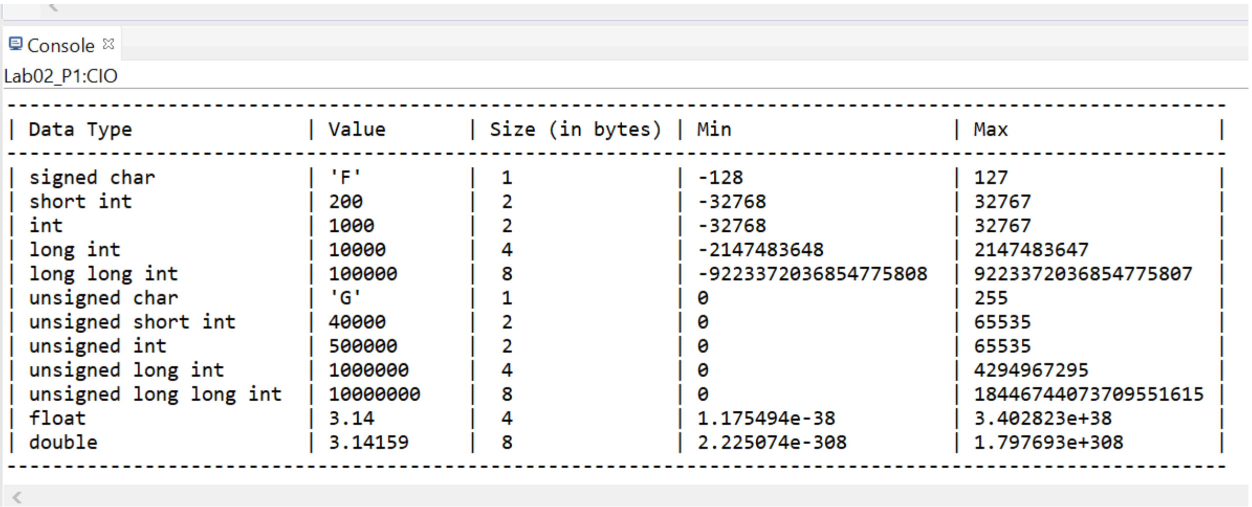
In the program, format specifiers are used within the printf function to control how the program formats and outputs various types of data in the table. Each format specifier corresponds to a specific type of data and ensures that the data is printed correctly. They also handle type-specific behavior, such as preventing negative values for unsigned types and properly handling large ranges for floating-point types.

The %d specifier is used to print signed integers, such as the minimum and maximum values for types like int, short int, and long int. The %u specifier is used for unsigned integers. For larger integer types, %ld and %lld are used to print long int and long long int values, respectively, while %lu and %llu handle their unsigned counterparts. Floating-point values, like float and double, are printed using the %e specifier in scientific notation. It reserves 20 character spaces for the output and specifies 10 digits of precision after the decimal point. The program also uses %2zu to display the sizes of smaller values correctly, with a minimum field width of 2 characters.

2. How are you calculating the output in Q2?

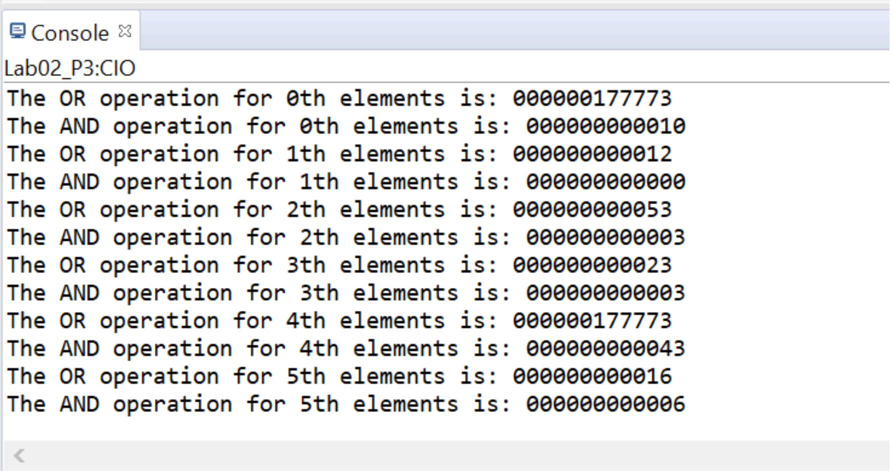
The output in Q2 is calculated using the formula for the ranges 0 to 2^{n} - 1 for unsigned numbers and - 2^{n-1} to 2^{n-1} - 1 for signed numbers.

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



```
Lab02_P1:CIO
----------------------------------------------------------------------------------------------------------
| Data Type               | Value       | Size (in bytes) | Min                   | Max                 |
----------------------------------------------------------------------------------------------------------
| signed char             | 'F'         |  1              | -128                  | 127                 |
| short int               | 200         |  2              | -32768                | 32767               |
| int                     | 1000        |  2              | -32768                | 32767               |
| long int                | 10000       |  4              | -2147483648           | 2147483647          |
| long long int           | 100000      |  8              | -9223372036854775808  | 9223372036854775807 |
| unsigned char           | 'G'         |  1              | 0                     | 255                 |
| unsigned short int      | 40000       |  2              | 0                     | 65535               |
| unsigned int            | 500000      |  2              | 0                     | 65535               |
| unsigned long int       | 1000000     |  4              | 0                     | 4294967295          |
| unsigned long long int  | 10000000    |  8              | 0                     | 18446744073709551615|
| float                   | 3.14        |  4              | 1.175494e-38          | 3.402823e+38        |
| double                  | 3.14159     |  8              | 2.225074e-308         | 1.797693e+308       |
----------------------------------------------------------------------------------------------------------
```

Figure 05: Q1 Output



```
Lab02_P3:CIO
The OR operation for 0th elements is: 000000177773
The AND operation for 0th elements is: 000000000010
The OR operation for 1th elements is: 000000000012
The AND operation for 1th elements is: 000000000000
The OR operation for 2th elements is: 000000000053
The AND operation for 2th elements is: 000000000003
The OR operation for 3th elements is: 000000000023
The AND operation for 3th elements is: 000000000003
The OR operation for 4th elements is: 000000177773
The AND operation for 4th elements is: 000000000043
The OR operation for 5th elements is: 000000000016
The AND operation for 5th elements is: 000000000006
```

Figure 06: Q3 Output

Appendix

Table 01: Program 1 source code

```
/*--------------------------------------------------------------------------
 * File:        Lab02_P1.c
 * Function:    Data types and their sizes
 * Description: This program displays the sizes and ranges of various data
types.
 * Input:       None
 * Output:      The sizes and ranges of the various data types
 * Author(s):   Anshika Sinha
 * Date:        01/15/2025
 *------------------------------------------------------------------------*/
#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               | Value       | Size (in bytes) | Min
| Max                  |\n");

printf("----------------------------------------------------------------------
------------------------------------\n");

    printf("| signed char             | 'F'         | %2zu              | %d
| %d                  |\n", sizeof(signed char), SCHAR_MIN, SCHAR_MAX);
    printf("| short int               | 200         | %2zu              | %d
| %d                |\n", sizeof(short int), SHRT_MIN, SHRT_MAX);
    printf("| int                     | 1000        | %2zu              | %d
| %d                |\n", sizeof(int), INT_MIN, INT_MAX);
    printf("| long int                | 10000       | %2zu              | %ld
| %ld           |\n", sizeof(long int), LONG_MIN, LONG_MAX);
    printf("| long long int           | 100000      | %2zu              | %lld
| %lld  |\n", sizeof(long long int), LLONG_MIN, LLONG_MAX);
    printf("| unsigned char           | 'G'         | %2zu              | 0
| %u                  |\n", sizeof(unsigned char), UCHAR_MAX);
    printf("| unsigned short int      | 40000       | %2zu              | 0
| %u                |\n", sizeof(unsigned short int), USHRT_MAX);
    printf("| unsigned int            | 500000      | %2zu              | 0
| %u                |\n", sizeof(unsigned int), UINT_MAX);
    printf("| unsigned long int       | 1000000     | %2zu              | 0
| %lu           |\n", sizeof(unsigned long int), ULONG_MAX);
    printf("| unsigned long long int  | 10000000    | %2zu              | 0
| %llu |\n", sizeof(unsigned long long int), ULLONG_MAX);
    printf("| float                   | 3.14        | %2zu              | %e
| %e         |\n", sizeof(float), FLT_MIN, FLT_MAX);
    printf("| double                  | 3.14159     | %2zu              | %e
| %e        |\n", sizeof(double), DBL_MIN, DBL_MAX);
printf("----------------------------------------------------------------------
------------------------------------\n");
    return 0;
}
```

Table 02: Program 3 source code

```
/*---------------------------------------------------------------------------
 * File:        Lab02_P3.c
 * Function:    Perform bitwise OR and AND operations on elements of two
arrays
 * Description: This program performs bitwise OR and AND operations on the
corresponding elements of two arrays.
 * Input:       Two arrays with at least 5 elements and at least one element
in each array
 *              should be a negative integer and at least two elements in each
array should
 *              be greater than or equal to 0xA.
 * Output:      The result of bitwise OR and AND operations on the
corresponding elements of the two arrays.
 * Author(s):   Anshika Sinha
 * Date:        01/16/2025
 *-------------------------------------------------------------------------*/
#include <msp430.h>
#include <stdio.h>

int main() {
    WDTCTL = WDTPW + WDTHOLD;       // Stop watchdog timer

    // Initialize the arrays with hexadecimal values
    int X[] = {0xA, 0x2, 0xB, 0x3, -0x5, 0x6};
    int Y[] = {-0x7, 0x8, 0x23, 0x13, 0x23, 0xE};

    // Determine the size of the arrays
    int size = sizeof(X) / sizeof(X[0]);

    // Loop through the arrays and perform bitwise operations
    int i = 0;
    for (i = 0; i < size; i++) {
        int or_result = X[i] | Y[i];
        int and_result = X[i] & Y[i];

        // Print the results in octal format
        printf("The OR operation for %dth elements is: %012o\n", i,
or_result);
        printf("The AND operation for %dth elements is: %012o\n", i,
and_result);
    }

    return 0;
}
```

Table 03: Bonus Program source code

```
/*-------------------------------------------------------------------------
 * File:        Lab02_PB.c
 * Function:    Multiply two 8x8 matrices
 * Description: This program multiplies two 8x8 matrices and displays the
result.
 * Input:       Two matrices of size 8x8
 * Output:      The resulting matrix after multiplying the two input matrices
 * Author(s):   Anshika Sinha
 * Date:        01/19/2025
 *-----------------------------------------------------------------------*/
#include <msp430.h>
#include <stdio.h>

#define SIZE 8

void displayMatrix(int matrix[SIZE][SIZE], const char *name);

void multiplyMatrices(int m1[SIZE][SIZE], int m2[SIZE][SIZE], int
result[SIZE][SIZE]);

int main()
{
    WDTCTL = WDTPW + WDTHOLD;       // Stop watchdog timer
    int m1[SIZE][SIZE];
    int m2[SIZE][SIZE];
    int result[SIZE][SIZE];

    // Initialize the matrices
    int i = 0;
    int j = 0;
    for (i = 0; i < SIZE; i++)
    {
        for (j = 0; j < SIZE; j++)
        {
            m1[i][j] = i + j;                // Example values for matrix1
            m2[i][j] = (i == j) ? 1 : 0; // Identity matrix for matrix2
        }
    }

    // Display input matrices
    displayMatrix(m1, "Matrix 1");
    displayMatrix(m2, "Matrix 2");

    // Perform matrix multiplication
    multiplyMatrices(m1, m2, result);

    // Display the result matrix
    displayMatrix(result, "Resulting Matrix");

    return 0;
}

void displayMatrix(int matrix[SIZE][SIZE], const char *name)
{
    printf("%s:\n", name);
    int i = 0;
    int j = 0;
    for (i = 0; i < SIZE; i++)
    {
        for (j = 0; j < SIZE; j++)
        {
            printf("%4d", matrix[i][j]);
        }
        printf("\n");
    }
    printf("\n");
}

void multiplyMatrices(int m1[SIZE][SIZE], int m2[SIZE][SIZE], int 
result[SIZE][SIZE])
{
    int i = 0;
    int j = 0;
    int k = 0;
    for (i = 0; i < SIZE; i++)
    {
        for (j = 0; j < SIZE; j++)
        {
            result[i][j] = 0;
            for (k = 0; k < SIZE; k++)
            {
                result[i][j] += m1[i][k] * m2[k][j];
            }
        }
    }
}
```

Make sure to submit a single pdf file only