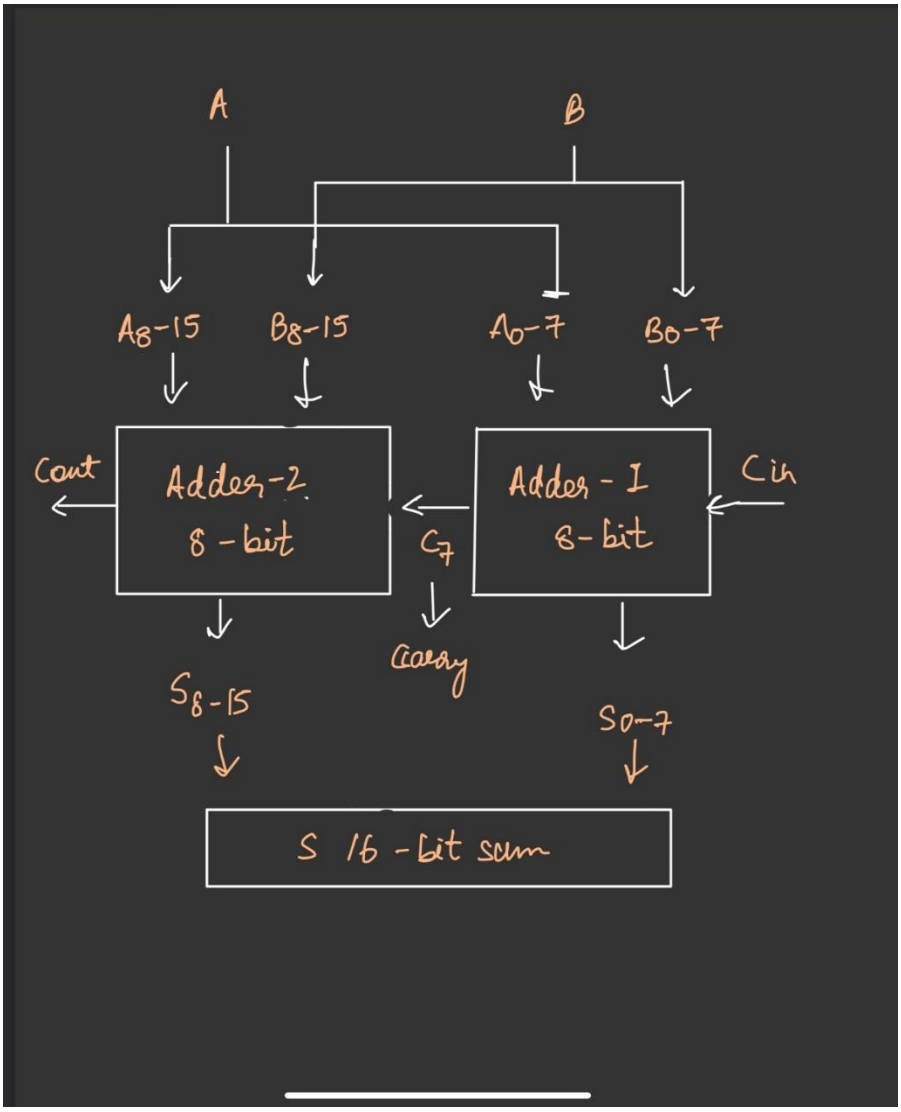
Assignment 1: Digital Logic and Representations - 2GA3

Jasraj Singh Johal 400434346

Task 1: Digital Logic (4 points)

a) Using two 8-bit full adders, a 16-bit addition is intended to be accomplished. 16-bit numbers are divided into their low and high 8-bits to do this. Then, the two full adders are connected in series, with the carry-in for the second adder (which handles the upper 8 bits) replacing the carry-out from the first adder (which handles the lower 8 bits). The carry-in and carry-out signals are controlled by additional NAND or NOR gates. The main goal is to ensure effective 16-bit addition with the fewest additional parts.



b) A decoder is a digital component that translates binary inputs into multiple output signals, selecting one output line from several based on the binary input. With an n-bit input, it typically has 2^n outputs, where only one output is active for each input combination. When used as a demultiplexer, a decoder routes an additional input to the selected output. When paired with a clock and counter, it can guide a circuit through a sequence of steps, all achievable using Boolean gates.
For a decoder with n input lines, there will be 2^n output lines. If n = 3, then the decoder will have $2^3 = 8$ output pins.

c)

Input	Binary Output	Integer Output
1	001	1
1	001	1
0	001	1
0	001	1
1	010	2
0	010	2
1	011	3
0	011	3
0	011	3
1	100	4

Task 2: Data Representation (3 points)

a) Base Case:
Let k = 1, number of distinct values that can be represented by two values: 0 and 1, so $2^1 = 2$ which is true.

Inductive Hypothesis:
Let's assume for a positive integer k = m, 2^m distinct values can be represented by m bits.

Inductive Step:
To prove that k + 1 bit can represent 2^{k+1} distinct values.
We will show that, if k = m + 1 bits then k1, k2, k3, … , k(m+1). Using Inductive Hypothesis, the first m bits can represent 2^m bits.

The final bit k(m + 1) has 0, 1representation. Therefore, all the 2^m remaining representations can have 2 representations.

$2^{(m+1)} = 2 * 2^m$ (2^m values can be represented by n bits from Inductive Hypothesis)

Hence proved k + 1 bits can represent $2^{(k+1)}$ distinct values.

b) For a 4-bit byte:

Maximum Value:

The largest positive value is when the most significant bit (MSB) is 0 (indicating a positive number) and the remaining bits are 1s.

That is: 0111.

This represents $2^2 + 2^1 + 2^0 = 4 + 2 + 1 = 7$

Minimum Value:

The largest negative value is when the MSB is 1 (indicating a negative number) and the remaining bits are 0s.

That is: 1000

In two's complement, this represents $-2^{(n-1)}$ where n is the number of bits. Here, n=4.

This represents $-2^3 = -8$

For an unsigned integer:

Maximum Value:

All bits are 1s.

That is: 1111

This represents $2^3 + 2^2 + 2^1 + 2^0 = 8 + 4 + 2 + 1 = 15$

Minimum Value:

All bits are 0s.

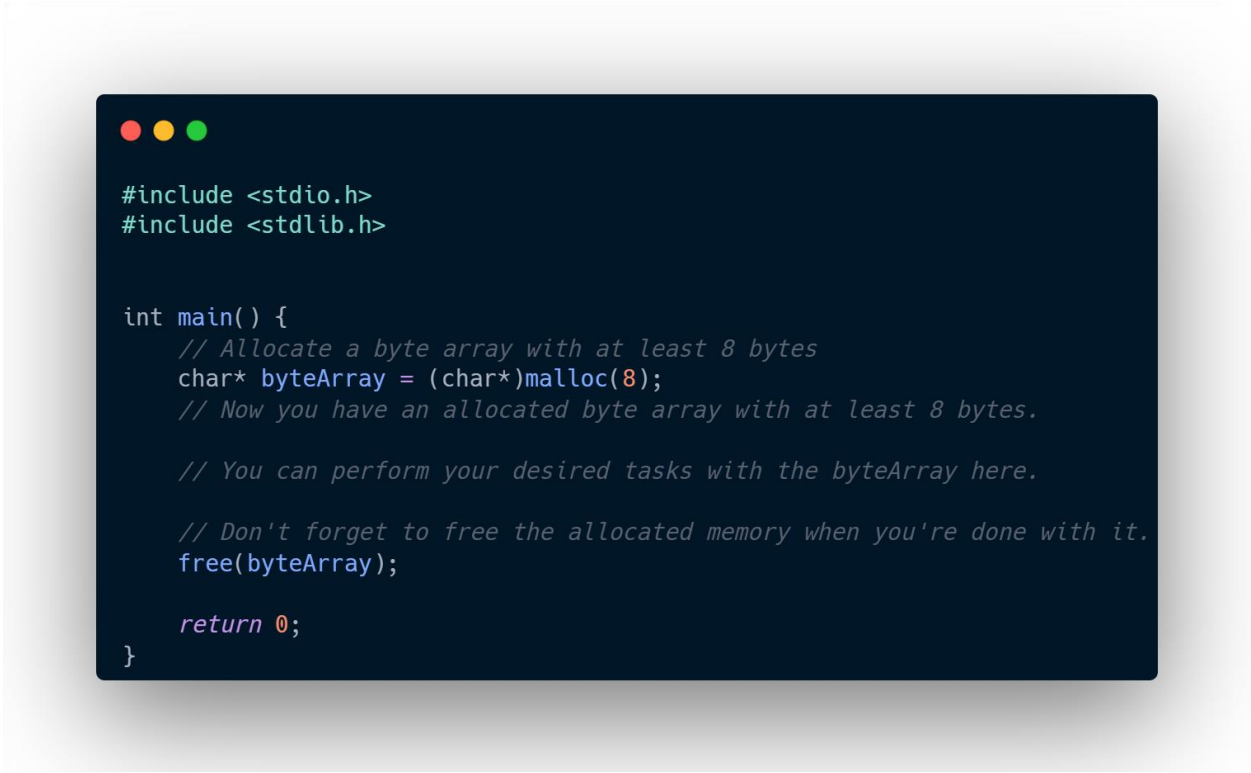
That is: 0000

This represents 0.

In summary, for a 4-bit two's complement system, the range is from -8 to 7. For an unsigned 4-bit integer, the range is from 0 to 15.

Task 3: Integer Representations (4 points)

a)



```c
#include <stdio.h>
#include <stdlib.h>


int main() {
    // Allocate a byte array with at least 8 bytes
    char* byteArray = (char*)malloc(8);
    // Now you have an allocated byte array with at least 8 bytes.

    // You can perform your desired tasks with the byteArray here.

    // Don't forget to free the allocated memory when you're done with it.
    free(byteArray);

    return 0;
}
```

b)

```c
#include <stdio.h>
#include <math.h>

char toFloat(float arg) {
    int exponent = (int) log2f(arg);

    if (exponent + 7 >= 15) {
        return (char) 0x7F; // Maximum representable value
    } else if (exponent + 7 <= 0) {
        return (char) 0;
    } else {
        exponent += 7;
    }

    int mantissa = (int)((arg / powf(2, exponent - 7)) * powf(2, 4));
    char result = (char)((exponent << 4) | mantissa);
    return result;
}

int main() {
    float input = 5.25; // Example input
    char result = toFloat(input);
    printf("Original float: %f\n", input);
    printf("8-bit floating point representation: %d\n", result);
    return 0;
}
```

c)

```c
#include <stdio.h>

// Function to check integer representation
void checkIntegerRepresentation() {
    char testNum = (char)0b1111111; // Binary 1111111 is decimal 127

    // Compare with 127 and -129
    if (testNum == 127) {
        printf("The architecture uses sign magnitude integer representation.\n");
    } else if (testNum == -129) { // Corrected to -129 for two's complement
        printf("The architecture uses two's complement integer representation.\n");
    } else {
        printf("Unknown integer representation.\n");
    }
}

int main() {
    checkIntegerRepresentation();
    return 0;
}
```

Task 4: Floating-point representation (5 points)

a) Largest:

Sign = 0 (positive)

Exponent = 110 (which is 6 in decimal)

Mantissa = 1111 (which is a fraction)

Decoding the given representation:

Largest number = 1.1111 (binary) x 2^3

= 1111.1 (binary) = 15.5 (decimal)

Smallest:

You can assume that the smallest value is -15.5

Due to a sign change

Smallest Increment:

The smallest increment is determined by the least significant bit of the mantissa. Which means the smallest fraction it can represent is 2^{-6}.

b)

```c
#include <stdio.h>
#include <math.h>

char toFloat(float arg) {
    int exponent = (int) log2f(arg);

    if (exponent + 7 >= 15) {
        return (char) 0x7F; // Maximum representable value
    } else if (exponent + 7 <= 0) {
        return (char) 0;
    } else {
        exponent += 7;
    }

    int mantissa = (int)((arg / powf(2, exponent - 7)) * powf(2, 4));
    char result = (char)((exponent << 4) | mantissa);
    return result;
}

int main() {
    float input = 5.25; // Example input
    char result = toFloat(input);
    printf("Original float: %f\n", input);
    printf("8-bit floating point representation: %d\n", result);
    return 0;
}
```

c)

```c
#include <math.h>

char toFloat(float arg) {

    char sign = (arg < 0) ? 1 : 0;

    arg = fabsf(arg);

    int expo = (int)log2f(arg);

    if (expo < -3) {
        expo = -3;
    } else if (expo > 4) {
        expo = 4;
    }

    // Normalize the mantissa
    float mantissa = arg / powf(2, expo);

    // Convert mantissa to 4-bit representation
    char bits = (char)(mantissa * 16);

    // Combine the sign, expo, and mantissa bits
    char result = (sign << 7) | ((expo + 3) << 4) | bits;

    return result;
}
```