Noah Meeker

CS 420

2 / 6 / 2023

Lab 1

In Lab 1, we were task to perform bit manipulation on a decimal number depending on the bit system. Along with providing the bit representation of the decimal number given, we were to provide the one's and two's complement of said number. The limitations for the code were pointer variables only and is to be written in the C language. The main reason for this lab was to have exposure to pointers and memory allocations used in the C language.

To accomplish this, every variable was designated as a pointer. Every pointer was given dynamically allocated memory to hold the necessary information. There are three main pointers. Two integer pointers labeled arr and size. Arr would point to the bit array produced by the code and size would point to the bit system size given by the user. Due to the code needing to support up to 64-bit inputs from the user. The third main pointer is a unsigned long long integer pointer. The scanf() function, provided by C's standard IO library, is used to read in the user's inputs. Based on the variable given to size, the arr pointer is allocated to match that size. These pointer variables are then passed to the following methods.

The first method used in the code is DecToBin(). This method takes in three parameters. Two pointers are instantiated on either side of the bit array and then pointer arithmetic is used to index through the array. At each index, the 1's and 0's are decided by calculating for the remainder of the current value divided by two. The current value is then updated to the current value divided by two. If the current value is no longer positive, then the rest of the array is filled with 0's. This creates a backwards bit representation of the passed in value. After this method, we print the current array for the user to see.

The second method used in the code is OneComp(). As the method name implies, it performs the one's complement on the bit array previously created in the DecToBin() method. This method takes two parameters. Two pointers are instantiated on either side of the bit array and then pointer arithmetic is used to index through the array. The method then checks at each index if there is a one or zero. Depending on what the method sees, it flips the bit. After this method, we print the updated one's complement array for the user to see.

The third method used in the code is TwoComp(). Asa the method name implies, it performs the two's complement on the bit array previously created in the OneComp() method. This method takes two parameters. Two pointers are instantiated on either side of the bit array and then pointer arithmetic is used to index through the array. A third pointer is instantiated with its own memory

allocated space. This third pointer acts as the carry integer to help the method know when to do bit manipulation on the array. At each index, if the bit and carry both hold a bit value of one, then the bit value becomes zero and the carry value is carried to the next bit. If the bit is zero and the carry value is one, then the bit value becomes one and the carry value becomes zero. In any other instance, the bit is left alone. After this method, we print the updated two's complement array for the user to see.

The last method used in the code is a helper method called PrintBin(). This method is used to print the current bit array and takes two parameters. Two pointers are instantiated at either side of the array and then pointer arithmetic is used to index through the array backwards. This is to ensure that the bit array is printed in the corrected orientation since the array was first created backwards.

There are issues with this current version of the code. In order to compensate for 64-bit numbers to be given by the user, the unsigned long long integer pointer was introduced. However, even with this adjustment, there is an overflow issue for when the user inputs a value greater than what the val pointer can hold. Current understanding points this to the use of scanf() to read in user inputs. There is a work around with using strings. As of this version, most errors are covered other than this.

Bitmanipulation.c

```
/*
 * CS 420 | Lab 1
 *
 * bitmanipulation.c
 *
 * @author: Noah Meeker
 * @REDID: 821272563
 *
 */

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

// Header Declaration
void DecToBin(int* arr, unsigned long long* val, int* size);
void OneComp(int* arr, int* size);
void TwoComp(int* arr, int* size);
void PrintBin(int* arr, int* size);

unsigned long long* pow_base2(int* exp);




int main(int argc, char *argv[]) {

    int* arr;
    unsigned long long* val = (unsigned long long*)malloc(sizeof(unsigned long long));
    int* size = (int*)malloc(sizeof(int));

    printf("CS 420: Lab 1\n");
```

```
printf("-------------\n");
printf("Bit Manipulation\n\n");

printf("Input Number: ");

// Check Integer Validation
if (scanf("%llu", *&val) != 1) {
    printf("[ERROR] Incorrect Input. Must Be A Number\n");
    return 1;
}
printf("\n");

printf("Input Bit Size: (ie. 4/8/16/32/64) ");

// Check Integer Validation
if (scanf("%d", *&size) != 1) {
    printf("[ERROR] Incorrect Input. Must Be A Number\n");
    return 1;
}
printf("\n");

// First *size in the below expression is for the case when *size is 0
if (!(*size && !(*size & (*size - 1)))) {
    printf("[ERROR] Desired bit size %d is invalid. \n", *size);
    return 1;
}

if (*val/2 >= *(pow_base2(size))/2) {
    printf("[ERROR] Number %llu does not fit in desired bit system %d.\n", *val, *size);
    return 1;
}

printf("Number: %llu\n", *val);
printf("Bit System: %d\n", *size);
```

```
    arr = (int*)malloc(sizeof(int) * *size); // Allocate array to bit system size

    DecToBin(arr, val, size);

    printf("Number in Binary: ");
    PrintBin(arr, size);

    OneComp(arr, size);

    printf("Number in One's Compliment: ");
    PrintBin(arr, size);

    TwoComp(arr, size);

    printf("Number in Two's Compliment: ");
    PrintBin(arr, size);

    printf("\nGood Bye\n");

    free(val);
    free(size);
    free(arr);

    return 0;
}

/* Convert Decimal to Binary */
void DecToBin(int* arr, unsigned long long* val, int* size) {

    int* arr_index = arr;
    int* arr_end = arr + *size;

    while (arr_index < arr_end) {
```

```
        if (*val > 0) {

            // Add remainder to array
            *arr_index = *val % 2;
            *val = *val / 2;

        } else {

            // Pad out the rest of the array with 0's
            *arr_index = 0;

        }

        arr_index += 1;
    }
}

/* Perform One's Compliment */
void OneComp(int* arr, int* size) {

    int* arr_index = arr;
    int* arr_end = arr + *size;

    while (arr_index < arr_end) {

        // Check value and flip
        switch (*arr_index) {
            case 0:
                *arr_index = 1;
                break;
            case 1:
                *arr_index = 0;
                break;
```

```
        }

        arr_index += 1;
    }

}

/* Perform Two's Compliment */
void TwoComp(int* arr, int* size) {

    int* arr_index = arr;
    int* arr_end = arr + *size;

    int* carry = (int*)malloc(sizeof(int));

    *carry = 1;
    while (arr_index < arr_end) {

        if (*arr_index == 1 && *carry == 1) {
            *arr_index = 0;
        } else if ( *arr_index == 0 && *carry == 1) {
            *arr_index = 1;
            *carry = 0;
        }

        arr_index += 1;
    }

    free(carry);
}

/* Print Method */
void PrintBin(int* arr, int* size) {
```

```
    int* arr_start = arr;
    int* arr_index = arr + (*size - 1);

    while (arr_index >= arr_start) {

        printf("%d", *arr_index);
        arr_index -= 1;

    }
    printf("\n");
}

unsigned long long* pow_base2(int* exp) {

    int* temp = (int*)malloc(sizeof(int));
    int* base = (int*)malloc(sizeof(int));
    unsigned long long* sum = (unsigned long long*)malloc(sizeof(unsigned long long));
    *temp = *exp - 1;
    *base = 2;
    *sum = 1;

    while (*temp > 0) {

        *sum = *sum * (unsigned long long)(*base);
        *temp -= 1;
    }

    free(temp);
    free(base);
    return sum;

}
```

```
(base) noah@pop-os:~/Documents/School_Work/SchoolProjects/CS420/lab1$ gcc -g3 -o main bitmanipulation.c
(base) noah@pop-os:~/Documents/School_Work/SchoolProjects/CS420/lab1$ ./main
CS 420: Lab 1
------------
Bit Manipulation

Input Number: 4

Input Bit Size: (ie. 4/8/16/32/64) 4

Number: 4
Bit System: 4
Number in Binary: 0100
Number in One's Compliment: 1011
Number in Two's Compliment: 1100

Good Bye
```