Implement single adder - 10/02/21

To implement the single adder I created a main program that would read in the values to use and output the calculated values.

```
int main()
{
    bool A;
    bool B;
    bool c_in;
    sum Q;

    cout << "Input your first binary number: ";
    cin >> A;
    cout << "Input your second binary number: ";
    cin >> B;
    cout << "Input your value for C_In: ";
    cin >> c_in;

    Q = full_adder(A, B, c_in);

    cout << endl << "Sum is: " << Q.Q << endl;
    cout << "Carry out is: " << Q.c_out;

    return 0;
}
```

To implement the full adder I used the flowchart that I prepared in the prep to write a function that would call the half adder function twice to create a full adder.

```
sum full_adder(bool A, bool B, bool c_in)
{
    sum fa;
    //half add A & B
    sum ha_1 = half_adder(A, B);
    //half add Q & c_in
    sum ha_2 = half_adder(ha_1.Q, c_in);

    //set the output
    fa.Q = ha_2.Q;
    fa.c_out = func_or(ha_1.c_out, ha_2.c_out);

    return fa;
}
```

Testing my implementation I found that I could correctly sum two single binary bits.

```
PS G:\1204_Advanced_Programming\P2_Adders_&_subtractors\3.1_Single_adder>
.\single.exe
Input your first binary number: 0
Input your second binary number: 0
Input your value for C_In: 1

Sum is: 1
Carry out is: 0
PS G:\1204_Advanced_Programming\P2_Adders_&_subtractors\3.1_Single_adder>
.\single.exe
Input your first binary number: 0
Input your second binary number: 1
Input your value for C_In: 1

Sum is: 0
Carry out is: 1
```

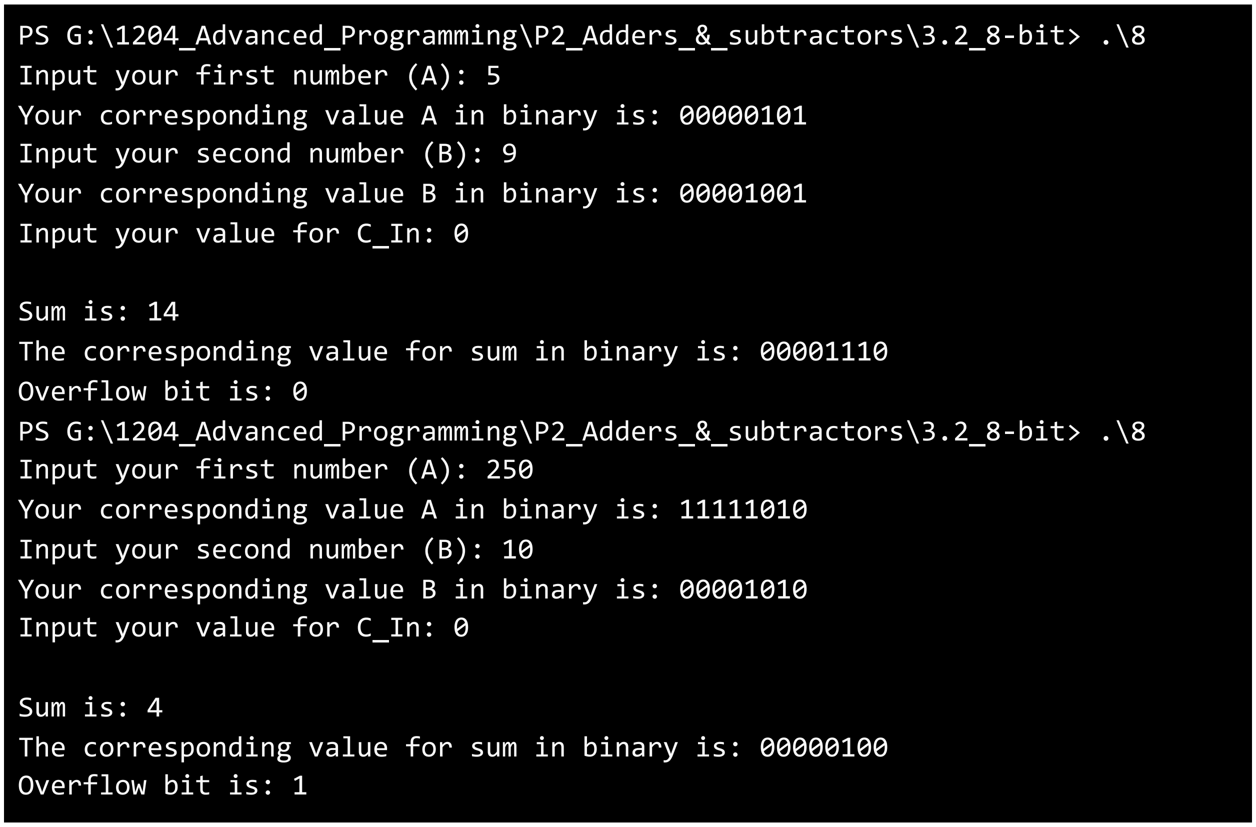
Implement 8-bit adder - 10/02/21

To implement the 8-bit adder I read in the ints of the two values. I then had to convert the value to an array of bools so my register function could access the information in binary. When printing the final number back out I had to implement the reverse function.

```
void conv_bin(int num, bool array[])
{
    for (int i = REG_SIZE - 1, j = 0; i >= 0; i--, j++)
    {
        if (num >= pow(2, i))
        {
            num -= pow(2, i);
            array[j] = 1;
        }
        else
        {
            array[j] = 0;
        }
        cout << array[j];
    }
}
int conv_dec(bool array[])
{
    int sum = 0;

    for(int i = REG_SIZE - 1, j = 0; i >= 0; i--, j++)
    {
        if (array[j] == 1)
        {
            sum += pow(2, i);
        }
    }

    return sum;
}
```

My register adder function followed the flowchart that I prepared in the prep. It would loop over the bits in the array completing the adder on the input registers and then update the carry in.

```c
sum_arr reg_adder(bool A[], bool B[], bool c_in)
{
    sum_arr output;
    sum temp;

    //full adder each bit of the registers.
    for(int i = REG_SIZE - 1; i >= 0; i--)
    {
        temp = full_adder(A[i], B[i], c_in);
        output.Q[i] = temp.Q;
        c_in = temp.c_out;
    }

    //set c_out to notify of an overflow
    output.c_out = c_in;
    return output;
}
```



```
PS G:\1204_Advanced_Programming\P2_Adders_&_subtractors\3.2_8-bit> .\8
Input your first number (A): 5
Your corresponding value A in binary is: 00000101
Input your second number (B): 9
Your corresponding value B in binary is: 00001001
Input your value for C_In: 0

Sum is: 14
The corresponding value for sum in binary is: 00001110
Overflow bit is: 0
PS G:\1204_Advanced_Programming\P2_Adders_&_subtractors\3.2_8-bit> .\8
Input your first number (A): 250
Your corresponding value A in binary is: 11111010
Input your second number (B): 10
Your corresponding value B in binary is: 00001010
Input your value for C_In: 0

Sum is: 4
The corresponding value for sum in binary is: 00000100
Overflow bit is: 1
```

Subtraction - 10/02/21

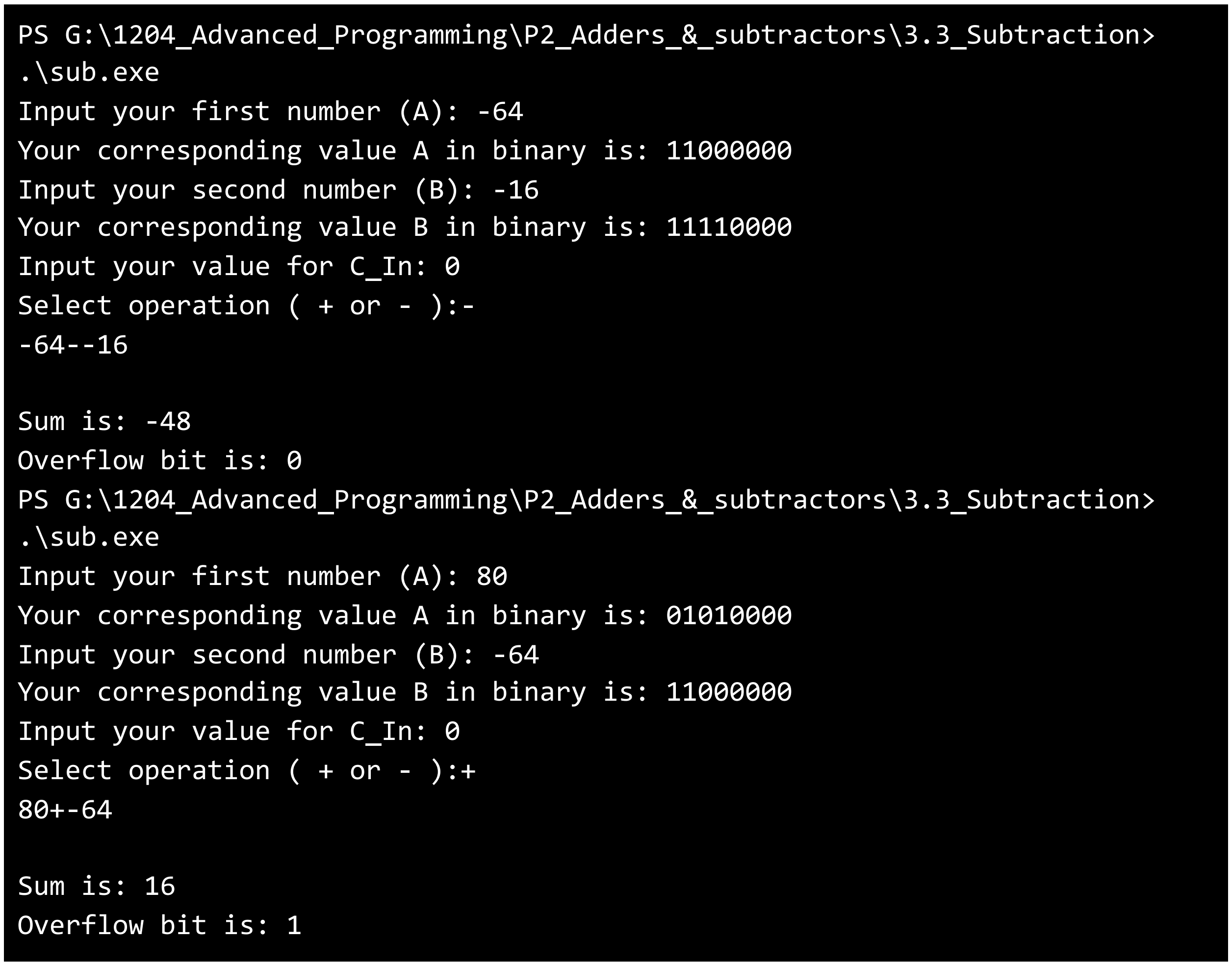
As with the previous section I had to update my main function so that it could select the operation to perform.

```
    cout << "Select operation ( + or - ):";
    cin >> op;
    cout << num_1 << op << num_2 << endl;
    if ((int)op == 45)
    {
        num_2 *= -1;
        conv_bin(num_2, B);
    }
```

When converting two and from binary I had to implement a new function that would convert the array two and from two's complement. This would loop over the array “flipping the bits” then using the register adder it would add one.

```
void conv_two(bool array[])
{
    bool empty[REG_SIZE] = {0};

    for (int i = 0; i < REG_SIZE; i++)
    {
        if (array[i] == 1)
        {
            array[i] = 0;
        }
        else
        {
            array[i] = 1;
        }
    }

    sum_arr temp = reg_adder(array, empty, 1);
    for (int i = 0; i < REG_SIZE; i++)
    {
        array[i] = temp.Q[i];
    }
}
```

Testing the subtractor and two's complement implementation I found that my functions operated correctly.



```
PS G:\1204_Advanced_Programming\P2_Adders_&_subtractors\3.3_Subtraction>
.\sub.exe
Input your first number (A): -64
Your corresponding value A in binary is: 11000000
Input your second number (B): -16
Your corresponding value B in binary is: 11110000
Input your value for C_In: 0
Select operation ( + or - ):-
-64--16

Sum is: -48
Overflow bit is: 0
PS G:\1204_Advanced_Programming\P2_Adders_&_subtractors\3.3_Subtraction>
.\sub.exe
Input your first number (A): 80
Your corresponding value A in binary is: 01010000
Input your second number (B): -64
Your corresponding value B in binary is: 11000000
Input your value for C_In: 0
Select operation ( + or - ):+
80+-64

Sum is: 16
Overflow bit is: 1
```