Due: Smartsite Wed., 4/29, 11:55 p.m.
Names of Files to Submit: **64bitAdd.s, editDist.s , div.c, divAssembly.s, ReadMe.txt**

- If you are working in a group **ALL** members must submit the assignment
- All programs should compile with no warnings when compiled with the -Wall option
- All prompts for input and all output must match my prompts/output. We use a program to grade your work and tiny differences can cause your work to be marked as a 0.
  - The best way to avoid being deducted points is to copy the prompt/unchanging portion of the outputs into your code. Make sure to get the spaces as well.
- You must also submit a file called ReadMe.txt. Include the names of all partners and any trouble you had on the assignment
- An example ReadMe.txt has been included with this assignment
- The input in the examples has been underlined to help you figure out what is input and what is output
- Submit your work on Smartsite by uploading each file separately. Do not upload any folders or compressed files such as .rar, .tar, .targz, .zip etc.
- If you have any questions please post them to Piazza

1. (Time 15 min)Write an assembly program called **64bitAdd.s** that adds two 64 bit numbers together.
   1. The first number will be referenced by the label **num1** and the second number will be referenced by the label **num2**.
   2. The upper 32 bits of the sum should be placed in **EDX** and the lower 32 bits in **EAX**.
   3. **AFTER** the last line of code that you wish to be executed in your program please place the label **done**.
   4. I have included a Makefile for you that will compile your program.
   5. **IT IS OF VITAL IMPORTANCE THAT YOU NAME YOUR LABELS AS SPECIFIED AND MAKE THE APPROPRIATE AMOUNT OF SPACE FOR EACH VARIABLE!** I will be using gdb to test your code and if your labels do not match then the tests will fail. You must also make sure to include the done label AFTER the last line of code you want executed in your program so that I know where to set break points.
   6. The following table shows how the numbers will be laid out in memory.

num1:	Upper 32 bits of num1	Lower 32 bits of num1
num2:	Upper 32 bits of num2	Lower 32 bits of num2

2. (Time 2 hours) Write an assembly program called **editDist.s** that calculates the edit distance between 2 strings. An explanation of what edit distance is can be found here while accompanying pseudo code can be found here.
    1. The label for the first string should be **string1** and the label for the second string should be **string2.**
    2. The edit distance between string1 and string2 should be placed in **EAX**.
    3. For each string please allocate space for 100 bytes.
        1. While you must allocate space for 100 bytes in your final submission you will likely find it easier to work with the .string directive for testing and debugging.
    4. **AFTER** the last line of code that you wish to be executed in your program please place the label **done**.
    5. I have included a C implementation of the edit distance program. I highly recommend translating this solution into assembly as it will make your life much easier.
    6. I have included a Makefile that will compile your program. Your program must be able to be compiled by this Makefile when you submit it
    7. **IT IS OF VITAL IMPORTANCE THAT YOU NAME YOUR LABELS AS SPECIFIED AND MAKE THE APPROPRIATE AMOUNT OF SPACE FOR EACH VARIABLE!** I will be using gdb to test your code and if your labels do not match then the tests will fail. You must also make sure to include the done label AFTER the last line of code you want executed in your program so that I know where to set break points.

3. Write a C program called **div.c** that implements division on two unsigned integers.
    1. Name your executable **div.out**
    2. You cannot use division in your solution
    3. Arguments should be accepted from the command line
        1. The first argument is the dividend
        2. The second argument is divisor
    4. Your program should display the quotient and remainder after doing the division
    5. Your program must complete in O(1) time
        1. This is possible because an integer is 32 bits long and so the loop that does the division should not take longer than 32 iterations.
        2. Because of this restriction the following solution is not acceptable as it does not meet the O requirements

```
void bad_div(unsigned int dividend,
unsigned int divisor,
unsigned int* quotient,
unsigned int *remainder){
  *quotient = 0;

  while(dividend >= divisor){
    (*quotient)++;
    dividend -= divisor;
  }

  *remainder = dividend;

}
```

        3. In order to meet the O requirements you will have to division in base 2 as you would by hand. See these 2 articles for some examples: Dr. Math and Exploring Binary
            1. Hint: use bitwise operators
            2. The one step that they leave out and one that you normally skip when doing division by hand is checking to see how many times the divisor goes into the dividend for the numbers that contain fewer digits than the divisor.
                1. For example: 30 / 15
                2. First we should check how many times does 15 go into 3. The answer is 0.
                3. Then we check how many times does 15 go into 30, which is 2.
                4. So our answer would be 02 R 0
        4. Examples:

```
1. ./div.out 10 5
   10 / 5 = 2 R 0
2. ./div.out 100 17
   100 / 17 = 5 R 15
3.
```

4. Translate your C program from problem 3 into an assembly program called **divAssembly.s.**
    1. The label for the dividend is **dividend**
        1. 4 bytes of space should be made for the dividend
    2. The label for the divisor is **divisor**
        1. 4 bytes of space should be made for the divisor
    3. Place the quotient in **EAX**
    4. Place the remainder in **EDX**
    5. **AFTER** the last line of code that you wish to be executed in your program please place the label **done**.
    6. **IT IS OF VITAL IMPORTANCE THAT YOU NAME YOUR LABELS AS SPECIFIED AND MAKE THE APPROPRIATE AMOUNT OF SPACE FOR EACH VARIABLE!** I will be using gdb to test your code and if your labels do not match then the tests will fail. You must also make sure to include the done label **AFTER** the last line of code you want executed in your program so that I know where to set break points.