# CS 271 Computer Architecture and Assembly Language
# Programming Assignment #4

**Objectives:**

1) Designing and implementing procedures
2) Designing and implementing loops
3) Writing nested loops
4) Understanding data validation

**Problem Definition:**

Write a program to calculate composite numbers. First, the user is instructed to enter the number of composites to be displayed, and is prompted to enter an integer in the range [1 .. 400]. The user enters a number, *n*, and the program verifies that $1 \leq n \leq 400$. If *n* is out of range, the user is re-prompted until s/he enters a value in the specified range. The program then calculates and displays all of the composite numbers up to and including the $n^{th}$ composite. The results should be displayed 10 composites per line with at least 3 spaces between the numbers.

**Requirements:**

1) The programmer's name must appear in the output.
2) The counting loop (1 to *n*) must be implemented using the MASM *loop* instruction.
3) The *main* procedure must consist (mostly) of procedure calls. It should be a readable "list" of what the program will do.
4) Each procedure will implement a section of the program logic, i.e., each procedure will specify how the logic of its section is implemented. The program must be modularized into at least the following procedures and sub-procedures :
   - *introduction*
   - *getUserData*
     - *validate*
   - *showComposites*
     - *isComposite*
   - *farewell*
5) The upper limit should be defined and used as a constant.
6) Data validation is required. If the user enters a number outside the range [1 .. 400] an error message should be displayed and the user should be prompted to re-enter the number of composites.
7) The usual requirements regarding documentation, readability, user-friendliness, etc., apply.
8) Submit your text code file (*.asm*) to Canvas by the due date.

**Notes:**

1) For this program, you may use global variables instead of passing parameters. This is a one-time relaxation of the standards so that you can get accustomed to using procedures.
2) A number *k* is composite if it can be factored into a product of smaller integers. Every integer greater than one is either prime or composite. Note that this implies that
   a. 1 is not composite.
   b. The number must be positive.
3) There are several ways to make your *isComposite* procedure efficient. (I recommend discussing this in your groups!)
4) See next page for an example execution

**Example** (user input in ***italics***):

```
Composite Numbers     Programmed by Euclid

Enter the number of composite numbers you would like to see.
I'll accept orders for up to 400 composites.

Enter the number of composites to display [1 .. 400]: 501
Out of range.  Try again.
Enter the number of composites to display [1 .. 400]: 0
Out of range.  Try again.
Enter the number of composites to display [1 .. 400]: 31

4   6   8   9   10   12   14   15   16   18
20   21   22   24   25   26   27   28   30   32
33   34   35   36   38   39   40   42   44   45
46

Results certified by Euclid.  Goodbye.
```

**Extra Credit:**

1) Align the output columns.
2) Display more composites, but show them one page at a time. The user can "Press any key to continue …" to view the next page. Since length of the numbers will increase, it's OK to display fewer numbers per line.
3) One way to make the program more efficient is to check against only prime divisors, which requires saving all of the primes found so far (numbers that fail the composite test). It's easy in a high-level language, but you will have to look ahead in the textbook to figure out how to do it in assembly language.

To ensure you receive credit for any extra credit options you did, you must add one print statement to your program output PER EXTRA CREDIT which describes the extra credit you chose to work on. You will not receive extra credit points unless you do this. The statement must be formatted as follows...

```
--Program Intro--
**EC: DESCRIPTION

--Program prompts, etc—
```