CS 271 Computer Architecture and Assembly Language
Programming Assignment #5

Objectives:

1. using indirect addressing
2. passing parameters
3. generating "random" numbers
4. working with arrays

Description:

Write and test a *MASM* program to perform the following tasks:

1. Introduce the program.
2. Get a user *request* in the range [*min* = 10 .. *max* = 200].
3. Generate *request* random integers in the range [*lo* = 100 .. *hi* = 999], storing them in consecutive elements of an *array*.
4. Display the list of integers before sorting, 10 numbers per line.
5. Sort the list in descending order (i.e., largest first).
6. Calculate and display the median value, rounded to the nearest integer.
7. Display the sorted list, 10 numbers per line.

Requirements:

1. The title, programmer's name, and brief instructions must be displayed on the screen.
2. The program must validate the user's request.
3. *min*, *max*, *lo*, and *hi* must be declared and used as global constants. Strings may be declared as global variables or constants.
4. The program must be constructed using procedures. At least the following procedures are required:
    A. main
    B. introduction
    C. get data {parameters: *request* (reference)}
    D. fill array {parameters: *request* (value), *array* (reference)}
    E. sort list {parameters: *array* (reference), *request* (value)}
        i. exchange elements (for most sorting algorithms): {parameters: *array*[i] (reference), *array*[j] (reference), where *i* and *j* are the indexes of elements to be exchanged}
    F. display median {parameters: *array* (reference), *request* (value)}
    G. display list {parameters: *array* (reference), *request* (value), *title* (reference)}
5. Parameters must be passed by value or by reference on the system stack as noted above.
6. There must be just one procedure to display the list. This procedure must be called twice: once to display the unsorted list, and once to display the sorted list.
7. Procedures (except main) should not reference .data segment variables by name. *request*, *array*, and titles for the sorted/unsorted lists should be declared in the .data segment, but procedures must use them as parameters. Procedures may use local variables when appropriate. Global constants are OK.
8. The program must use appropriate addressing modes for array elements.
9. The two lists must be identified when they are displayed (use the *title* parameter for the *display* procedure).
10. The program must be fully documented. This includes a complete header block for the program and for each procedure, and a comment outline to explain each section of code.
11. The code and the output must be well-formatted.
12. Submit your text code file (.asm) to Canvas by the due date.

Extra Credit (Be sure to describe your extras in the program header block)**:**

1. Display the numbers ordered by column instead of by row.
2. Use a recursive sorting algorithm (e.g., *Merge Sort*, *Quick Sort*, *Heap Sort*, etc.).
3. Implement the program using floating-point numbers and the floating-point processor.
4. Generate the numbers into a file; then read the file into the array.
5. Others?

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—
```

Notes:

1. The Irvine library provides procedures for generating random numbers. Call *Randomize* once at the beginning of the program (to set up so you don't get the same sequence every time), and call *RandomRange* to get a pseudo-random number. (See the documentation in Lecture slides.)
2. The *Selection Sort* is probably the easiest sorting algorithm to implement. Here is a version of the descending order algorithm, where *request* is the number of *array* elements being sorted, and *exchange* is the code to exchange two elements of *array*:

```
for(k=0; k<request-1; k++) {
   i = k;
   for(j=k+1; j<request; j++) {
      if(array[j] > array[i])
         i = j;
   }
   exchange(array[k], array[i]);
}
```

3. The median is calculated after the array is sorted. It is the "middle" element of the sorted list. If the number of elements is even, the median is the average of the middle two elements (may be rounded).

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

```
Sorting Random Integers               Programmed by Road Runner
This program generates random numbers in the range [100 .. 999],
displays the original list, sorts the list, and calculates the
median value.  Finally, it displays the list sorted in descending order.

How many numbers should be generated? [10 .. 200]: 9
Invalid input
How many numbers should be generated? [10 .. 200]: 16

The unsorted random numbers:
680   329   279   846   123   101   427   913   255   736
431   545   984   391   626   803

The median is 488.

The sorted list:
984   913   846   803   736   680   626   545   431   427
391   329   279   255   123   101
```