



Course Code: EE-2003	Course Name: Computer Organization & Assembly Language
Course Teacher:	Assignment No: 03

Instructions:

- You are required to Submit Assignment scanned word document on Google classroom.
- The deadline for submission is **December 07, 2022**
- Copying is not allowed at all. Any similarities among the submitted files of any student will result in zero marks.

Total Points: 100

Question 1:

Create a procedure that returns the sum of all array elements falling within the range j...k (inclusive). Write a test program that calls the procedure twice, passing a pointer to a signed doubleword array, the size of the array, and the values of j and k. Return the sum in the EAX register and preserve all other register values between calls to the procedure. Display both the results.

Note: Do include the documentation of your user-defined procedure

Hint: You can use the USES operator for saving registers. Don't use pushad/ popad here as result of procedure will be stored in EAX

Assume:

Array values: 30, -40, 20, 65, 80, 45

j = 20 and k = 50 for 1st call

j = 35 and k = 90 for 2nd call

Question 2:

•Selectsort method for sorting an array

To sort an array A of N elements, we proceed as follows:

Pass 1: Find the largest element among A[1] ... A[N]. Swap it and A[N]. Because this puts the largest element in position N, we need only sort A[1] ... A[N-1] to finish.

Pass 2: Find the largest element among A[1] ... A[N-1]. Swap it and A[N-1] This places the next-to-largest element in its proper position.

▪
▪
▪
▪

Pass N-1: Find the largest element among A[1], A[2]. Swap it and A[2]. At this point A[2]... A[N] are in their proper positions, so A[1] is as well, and the array is sorted.

Selectsort algorithm

i = N

For N-1 times DO

```
Find the position k of the largest element
among A[1]... A[i]
(*)Swap A[k] and A[i]
i = i-1
END_FOR
```

Step (*) will be handled by a procedure SWAP.

Write a program that sorts the elements of an array = {60,4,17,45,7} using the selectsort algorithm. Swapping operation should be performed by user-defined procedure named SWAP.

Question 3:

•To sort an array A of N elements by the bubblesort method, we proceed as follows:

Pass 1. For $j = 2 \dots N$, if $A[j] < A[j-1]$ then swap $A[j]$ and $A[j-1]$. This will place the largest element in position N.

Pass 2. For $j = 2 \dots N-1$, If $A[j] < A[j-1]$ then swap $A[j]$ and $A[j-1]$. This will place the second largest element in position N-1.

-
-
-

Pass N - 1. If $A[2] < A[1]$, then swap $A[2]$ and $A[1]$. At this point the array is sorted.

Write a procedure BUBBLE to sort a byte array by the bubblesort algorithm. The procedure receives the offset address of the array in ESI and the number of elements in EBX. Write a program that takes 10 single-digit numbers as input from user, calls BUBBLE to sort them, and then displays the sorted list on the console window.

Question 4:

Create a procedure FACTORIAL that will compute $N!$ for a positive integer N. The procedure should receive N in ECX and return $N!$ in EAX. Suppose that overflow does not occur. Write a test program that takes N as input from the user, calls the procedure FACTORIAL and displays the output on the console window.

Question 5:

Write a program that prompts the user to enter a character, and on subsequent lines prints its ASCII code in binary, and the number of 1 bits in its ASCII code.

-
-

Sample execution:

TYPE A CHARACTER: A

THE ASCII CODE OF A IN BINARY IS 01000001

THE NUMBER OF 1 BITS IS 2

Question 6:

Write a procedure named CountMatches that receives pointers to two arrays of signed doublewords, and a third parameter that indicates the length of the two arrays. For each element in the first array, if the corresponding in the second array is equal, increment a count. At the end, return a count of the number of matching array elements in EAX. Write a test program that calls your procedure and passes pointers to two different pairs of arrays. Use the INVOKE statement to call your procedure and pass stack parameters. Create a PROTO declaration for CountMatches. Save and restore any registers (other than EAX) changed by your procedure.

Question 7:

Create a procedure named Extended_Sub that subtracts two binary integers of arbitrary size. The storage size of the two integers must be the same, and their size must be a multiple of 32 bits. Write a test program that passes several pairs of integers, each at least 10 bytes long.

Question 8:

Create a procedure named Extended_Add that subtracts two binary integers of arbitrary size. The storage size of the two integers must be the same, and their size must be a multiple of 32 bits. Write a test program that passes several pairs of integers, each at least 10 bytes long.

Question 9:

Write a recursive implementation of Euclid's algorithm for finding the greatest common divisor (GCD) of two integers. High-level language code of this algorithm is given below.

```
int gcd(int a, int b)
{
    // Everything divides 0
    if (a == 0)
        return b;
    if (b == 0)
        return a;

    // base case
    if (a == b)
        return a;

    // a is greater
    if (a > b)
        return gcd(a-b, b);
    return gcd(a, b-a);
}
```

Write a test program that calls your GCD procedure thrice, using the following pairs of integers: (5,20), (24,18) and (432,226). After each procedure call, display the GCD.

Question 10:

Write a procedure named **CountNearMatches** that receives pointers to two arrays of signed doublewords, a parameter that indicates the length of the two arrays, and a parameter that indicates the maximum allowed difference (called diff) between any two matching elements. For each element in the first array, if the difference between it and the corresponding in the second array is less than or equal to diff, increment a count. At the end, return a count of the number of nearly matching array elements in EAX.

Write a test program that calls **CountNearMatches** and passes pointers to two different pairs of arrays. Use the **INVOKE** statement to call your procedure and pass stack parameters. Create a **PROTO** declaration for **CountNearMatches**. Save and restore any registers (other than EAX) changed by your procedure.