CSEC 201, LAB #11 BASIC ASSEMBLY CODING	DUE: 11/10/2021 @ 7 PM

Learning Outcome: Students will gain experience using the Immunity debugger to analyze binary code and writing assembly code that involves comparisons and jumps.

1. (25 points) Working with Binary Files

Use the Immunity debugger to analyze and modify the provided **Password.exe**.

1) Modify assembly code so that it always prints "Correct password!" no matter what the user enters.
2) Find the password stored in the stack frame of the **password** function.
3) Reopen Password.exe if needed. Replace the actual password with "Apple." And modify an assembly instruction so that when the user enters the new password (i.e., Apple), "password!" is printed on the console instead of "Correct password!".

The **main** function of the Password program is provided as follows:

```
int main() {
      char userin[20];
      printf("Enter your password: ");
      scanf("%s", userin);
      int result = password(userin);
      if (result == 1)
            printf("Correct password!");
      else
            printf("Incorrect password!");
}
```

Create a document file named **lab11_q1.doc** and include a screenshot for each:

1) A screenshot that shows the original password you found using Immunity.
2) A screenshot that shows how/where you modified in the Immunity Debugger.
3) A screenshot that shows both the Immunity assembly code you modified and the console displaying the user input and the modified message "password!".

You are required to write x86 assembly code for each of the following questions.

2. (25 points) Comparisons and Loop

Convert the following C code to assembly code.

```
      char str1[20] = "Hello World!";
      char str2[20];
      printf("Enter your string: ");
      gets(str2);  //Reads a line from stdin and stores it into
[str2]
      if (strcmp(str1, str2) == 0) {
            printf("%s and %s are the same", str1, str2);
      }
      else {
            printf("%s and %s are not the same", str1, str2);
```

```
}
```

- Use the .data section to declare `str1` and `str2`.
- You will not use the **strcmp** function in your assembly code. Use a loop to implement the string comparison functionality.
- Don't forget to declare the **gets** function, i.e., `extern gets:NEAR`

3. (25 points) Maximum of three
   Write an x86 assembly program to compute the maximum among the three numbers entered by the user. For example, if the user entered 10, 20, and 5, your program should display "Maximum is 20".

4. (25 points) Working with an integer array
   Declare an array of integers named `myarray` in the data section. In the code section, fill and print the array as indicated in the following C code:

```
int myarray[20];

for (int i = 0; i<20; i++){
    myarray[i] = (i*i + 5)%20;
    printf("%d ", myarray[i]);
}
printf("\n");
```

And then find the maximum value in the array and print it at the end.
Expected output:

```
5 6 9 14 1 10 1 14 9 6 5 6 9 14 1 10 1 14 9 6
max = 14
```

Deliverables:
You must upload the document file containing screenshots and working code to the Assignment11 repository before 11/10/2021 at 7 PM.

CSEC 201, LAB #11 BASIC ASSEMBLY CODING	DUE: 11/10/2021 @ 7 PM

NAME:
Each signature is worth 25% of your lab grade.

1. **The student could use Immunity Debugger to find the password and modify the assembly code as required.**

2. **The student could write a loop and compare two strings.**

3. **The student could use the comparison and jump instructions to compute the maximum among three numbers.**

4. **The student could use integer arrays, initialize elements, and find maximum in the array.**