CPEN 3710 **Lab Exercise 9**

Determining Whether Or Not Numbers Are Prime; Analyzing Program Performance

Required Materials:

- Your textbook, *Assembly Language for x86 Processors* (7th edition)
- Removable or network device (Flash drive, memory card, MyMocsNet account mapped to a drive letter, etc.) for storage of your programs
- These instructions
- Intel-compatible, Windows-based personal computer (like the ones in EMCS 306) with text editor, MASM, and Microsoft Visual Studio or CodeView (if needed for debugging)

Preparation for Laboratory:

Read the material on *multiplication and division* in section 7.3, pages 255-269 of your textbook. Also read the material on the use of stack frames in procedures in Chapter 8, pages 287-333.

Instructions:

Write an assembly language program which will prompt the user for a **non-negative decimal integer n** and determine whether or not **n is prime**. (You will need to do a bit of research regarding algorithms that can be used to test numbers to see if they are prime.) The program must display the number input and the result of the test on the screen, then prompt the user to enter another number. For example, if the user entered the numbers 7 and 100, the screen output produced by your program should appear as follows:

```
Enter a non-negative integer (0 to exit): 7
7 is a prime number.
Enter a non-negative integer (0 to exit): 100
100 is not a prime number; it is divisible by 2.
```

The input number may obviously consist of varying numbers of digits; leading zeroes may be entered by the user, but are *not* to be displayed on your output line. **The input number is to be stored as an *unsigned* 32-bit (doubleword) value** (legal input is any decimal integer value between 0 and 2^{32}-1). The program should terminate when the user enters 0, and should generate an error message if the user enters non-numeric input or a negative number. *The prime number test must be done in a separate procedure* that is called from the main program; the single argument (the number to be checked for primality) must be passed to this procedure *on the stack*, and a *result code must be returned to the caller in the EAX register*. **The value returned** in EAX must be **0 if the user's number is prime**, and must be **a divisor of the user's number if it is composite** (if the number has multiple divisors, any one of them may be returned as proof of non-primality). **In the special case that the user enters the number 1, return 1** (note that the number 1 is <u>not</u> prime by definition, since it does not have two *distinct* divisors.) No other registers may be changed by the "prime check" procedure, so use the stack to save and restore any registers your procedure uses, other than EAX. Your called procedure must create a *base pointer* which is used to reference its input parameter as well as the local variables (if any) that it creates, and must restore the caller's base pointer before returning.

Your program must also measure the elapsed time, in milliseconds, taken to check each (legal) input value for primality. You should do this by calling Irvine's GetMseconds procedure twice: once after the input is error-checked, but before the prime number procedure is called; and again just after the prime number procedure returns. The difference between the second value and the first will give you the run time in milliseconds, assuming that you did not commence your program run shortly before midnight (correction for “midnight rollover” is a desirable, but not required, feature of your program). ***In addition to outputting whether or not the number is prime, your program should also display the run time in milliseconds for each computation.*** For example:

```
Enter a non-negative integer (0 to exit): 1046527
1046527 is a prime number.
This computation took 15 milliseconds.
Enter a non-negative integer (0 to exit): 4
4 is not a prime number; it is divisible by 2.
This computation took 0 milliseconds.
```

To Hand In: *(due no later than 3:00 p.m. Tuesday, November 13)*

1. Turn in a printed copy of the *thoroughly commented .LST file* for your program. **Be sure to follow the guidelines given in the programming style and documentation handout.**

2. Submit the *results* of your program as follows: run it several times from the command prompt with *at least* the following test cases as input (more testing is encouraged):

   -11, 1, 2, 3, 13, 43, 48, 169, 240, 241, 487, 489, 65535, 65536, 333667, 3626149, 10619863, 715827883, 715827887, 4000000001, 4000000007, 0.

   In each case, capture a “screen shot” of the results; print out and include all these screen shots in your report. (Since the program input and output only require only a total of two or three lines, it may be possible to capture several inputs and outputs on the same screen.)

3. Make a table showing the (valid) inputs provided to the program and the run time associated with each. **What trends do you observe?** Is the time to check a number for primality dependent on whether or not it is actually prime, or does that not matter? For prime numbers, is the run time linearly proportional to the size of the number (*e.g.* does it take 100 times longer to check a ten-digit prime number vs. an 8-digit number)? If the relationship is not linear, can you identify it as being logarithmic, exponential, or some other defined mathematical relationship? (You may wish to create a graph with the numbers being checked on one axis and the run time on the other axis in order to depict and help explain the relationship.) **Write *at least* a good, solid paragraph (more is encouraged) analyzing the run-time performance of the program vs. the number input by the user.** *Type this up in a word processor and submit your table, analysis, and any other materials such as graphs along with the rest of your lab report.*

4. Have me check the operation of your program and sign in the space below when you have demonstrated its operation. **Submit *this signed sheet*** verifying that I have seen you demonstrate your working program.

Instructor’s signature: ____

Staple your program listing, results, performance analysis, and this signed sheet together. Submit these items by the date and time specified above. Late submissions will be penalized substantially.