Objective

The objective of this programming project is for you to practice writing an assembly language subroutine and to gain more experience with loop control.

Assignment

For this assignment, you will write an assembly language subroutine that prints out a 64-bit unsigned number to standard output in hexadecimal format. The 64-bit value will be first interpreted from the user's ASCII input (maximum of 20 characters allowed), then converted into a 64-bit value. This 64-bit value is to be converted into HEX, character by character by passing the character into a C subroutine (cfunc.c) through the RDI register. Your subroutine must preserve all registers (i.e., any registers that you use must be saved on the stack and restored before you return). Your subroutine should print the given number in hexadecimal and then do nothing else.

To determine the values of each of the base 16 digits in the hexadecimal representation of the number, you will use the division method (don't use repeated subtraction). The DIV instruction will divide the 64-bit value stored in RDX:RAX after converting the user's input. Note that RDX should be initialized to 0 here. After the DIV instruction is executed, the quotient is stored in the RAX register and the remainder is stored in the RDX register. (See further discussion of the DIV instruction in Implementation Notes below.)

You are provided with a skeleton code 'hexConverter.asm' and a C code 'cfunc.c' (you will not modify the provided c function).

The project is split into two files. You will be implementing your algorithm in the hexConverter.asm file. The entry point into the C subroutine/function must be named **`printhex`** and must be externally referenced using **`extern`** (as it already is in the skeleton project). The printf function/subroutine has also been declared **`extern`** for you in the skeleton project. You can review the expected test output in the test cases provided on Blackboard. If the input is less than 20 characters or more than 21 characters, the user input is no longer 64-bit. Validity checks must be implemented to ensure that the user inputs only numbers (no alphabets or special characters allowed).

To build your project :

```
nasm -g -f elf64 -F dwarf hexConverter.asm

gcc -g hexConverter.o cfunc.c -o converter.out
```

To test your project:

```
./converter.out
```

Implementation Notes

1. In the division method for converting a number to base 16, you will get the one's place first (the least significant digit), then the 16's place, then the 256's place, ... This is in the opposite order that the hexadecimal number will be printed. The easiest way to handle this situation is to prepare a string starting at a higher numbered address and loop backwards storing each character at a lower numbered address.

2. The number of characters that you will print will depend on the value passed in the RAX register. You will need to determine or keep count of the length of the hexadecimal number to be printed.

3. If the value in the RAX register is 0, then your subroutine should print out the string 0x0.

4. When you prepare you assembly file, you must tell the NASM assembler that the printhex routine is from a different file. Otherwise, it will only look for printhex in the current file (and not find it). To tell NASM this, include the declaration:

   ```
   extern printhex
   ```

   The extern declarations are also typically made just after the SECTION .text declaration.

5. There are a couple of quirks about the DIV instruction. First, since the dividend is always RDX:RAX, it is not specified in the instruction. For example, to divide RDX:RAX by RBX, we simply say:

   ```
   div rbx
   ```

6. Another quirk is that the DIV instruction does not support immediate operands. So, you cannot divide by 16 by saying:

   ```
   div 16 ; this is WRONG
   ```

7. Since you are converting a 64-bit value into hexadecimal, the dividend always fits in the lower 64 bits. Thus, you must zero out the RDX register before each DIV instruction.

8. We are always dividing by 16 in this program. So, we know that the remainder in RDX will always be less than 16. This value fits in 8 bits, which is quite rather convenient. Why?

9. Remember that the bytes that you are printing to the screen will be interpreted as ASCII values.

Turning in your program

Use the UNIX submit command on the GL system to turn in your project. You must name your assembly file “hexConverter.asm”. The ‘cfunc.c’ is to be submitted ‘as is’ with no modifications. The class name for submit is CMSC313_Deepak. The name of the assignment name is proj3. The UNIX command to do this should look exactly like:

```
submit CMSC313_Deepak proj3 hexConverter.asm cfunc.c typescript readme.txt
```