Intro to X86 Assembly

This assignment is a gentle introduction to x86 assembly language programming on a 64-bit Linux platform.

On the Sakai assignment page for this assignment, you will find a file named assign1.asm. For this assignment, you will copy this file onto a 64-bit Linux system, build the file using nasm and ld, run the resulting executable with and without gdb and provide answers to the questions that you find at the end of the this document.

Read through the following link for information on using the GNU debugger to step through your program:

http://www.cs.umbc.edu/~chang/cs313.s02/gdb_help.shtml
also in Sakai
Resources/GDB Survival Guide

You may download a bare bones 64-bit Ubuntu 18.04 virtual machine here:
https://nps.box.com/s/hbvo3afs6rfo4zkio0m30ng61cq08lfk

Unzip the VM, then open/import the ovf file.

After copying assign1.asm into your virtual machine, you need to assemble and link it to create an executable.

Assemble with nasm:

```
$ nasm -f elf64 assign1.asm
```

This command invokes the `nasm` assembler to assemble the input file assign1.asm. With the '`-f elf64`' option we asking nasm to generate a 64-bit ELF object file. Successful execution of this command should yield a file named `assign1.o`.

Linking with ld:

```
$ ld -o assign1 -m elf_x86_64 assign1.o
```

This command invokes the linker (ld) to create a 64-bit executable file by linking the object file `assign1.o` into a proper ELF executable. Successful execution of this command should yield a file named simply `assign1.`

Running the program:

Read through the questions at the end of this document to familiarize yourself with what is expected of you. Within the source code for the program, you will observer a number of labels such as _start, msg, and Q1. The questions below ask you to observe the state of the running process as the program reaches specific labels when it executes.

Before jumping into gdb you may wish to run the program at least once to observe its very simple behavior. After successfully creating the executable using the steps above, you can run the program with the following simple command line:

```
$ ./assign1
```

When you are ready to answer the questions you will need to run the program under gdb control:

```
$ gdb ./assign1
```

Once in gdb, you will need to use the `break` (b) command to set breakpoints at labels Q1 through Q7. For example:

```
(gdb) b Q1
```

Begin execution with the `run` (r) command:

```
(gdb) run
```

The program will stop as you reach the breakpoints that you set previously. With the questions at the end of the document in mind, gather the required information to complete the assignment.

Resume execution from a breakpoint using the `continue` (cont or c) command:

Reading the contents of memory is most often accomplished use the `x` (for examine) command. Choose an appropriate form of the x command based on the information each question asks you to collect. Ask gdb for help with the x command using:

```
(gdb) help x
```

The contents of CPU registers may be viewed using either the `print` command or the `info reg` command

```
(gdb) print $rax
...
(gdb) info reg
...
```

Note that when the program asks for your NPS user name, you should enter your actual NPS username.

Questions:

1. How many times does your program stop at breakpoint location Q1?
2. What is the string stored in memory beginning at the address contained in rsp when the program stops at breakpoint location Q2? What are the hex values of the 8 bytes stored in memory beginning ate the address contained in rsp (provide your answer as a space separated list of 2 digit hex values, e.g. 12 34 56 78 9a bc de f0)?
3. When the program stops at breakpoint location Q3, what is the hex value contained in the ebx register?
4. When the program stops at breakpoint location Q4, what are the hex values of the 8 bytes stored in memory beginning ate the address contained in rsp (provide your answer as a space separated list of 2 digit hex values, e.g. 12 34 56 78 9a bc de f0)?
5. When the program stops at breakpoint location Q5, what are the hex values of the 8 bytes stored in memory beginning ate the address contained in rsp (provide your answer as a space separated list of 2 digit hex values, e.g. 12 34 56 78 9a bc de f0)?
6. When the program stops at breakpoint location Q6, what is the hex value contained in the rax register?
7. When the program stops at breakpoint location Q7, what is the hex value contained in the rip register?
8. What is the output of the program (you may include the username that you entered if you wish, though it is not, strictly speaking, output)?

Deliverables

Paste your answers into the project submission form field for assignment 1 in Sakai. DO NOT upload your answers as a separate file into Sakai.