CSE 231: Operating Systems
Assignment 1, Question 2

Name: Abhik S Basu
Roll No: 2020165
Section: A

```
1   targ: q2
2
3   q2: q2.c
4       nasm -f elf64 -o b.o b.asm
5       gcc -no-pie  q2.c b.o -o q2
6       clear
7       ./q2
8
9   clean:
10      -rm *.o q2
```

To run the program, run the following make command in the terminal -

```
make
```

To clear out the files generated and reset the programme for a new run, run the following command in the terminal -

```
make clean
```

On running make, we will find the output the code in the following flow -

1. We will first be prompted to enter the 64 bit number.
2. On entering the number we will find the output as follows:

```
./q2
In function A right now

Enter the 64 bit number:208043721541

Printing in function B

Printing ASCII first in Little Endian: EC_p0
Printing the integer: 208043721541

In function C now. Exiting now
```

How the programme works (Processes) -

In this programme we have mainly three files. One is q2 which is written in C language. One is ‘b’ which is written in NASM language (assembly for intel x86_64) and one is ‘c’ which is again written in C language.

We have the main in q2.c . From here we redirect control to a function a() located in the q2.c file only. Here we accept a 64 bit integer as a long int and pass it onto the b() function. b() function takes in a int_64t parameter which is the 64 bit number being input by the user.

Now when ‘b’ is invoked we have to technically link the assembly programme with our C programme. This is done with the help of extern keyword. Extern basically has the function declaration which is used for linking to the assembly cod at runtime. So even thought the assembly code is not present in the q2.c file an extern with function declaration tells it that it will be found at the time of linking which has been done in our makefile.

Now after the control has been transferred to b, written in assembly. We do two things. First we show the number in its ASCII character representation and then we also show the integer which is printed as a string (the second functionality from the code was not asked, but I have done it for practice). So firstly in our code we do a few sys calls involving write. Now for write if the write function is such that write(fd,buffer,count). Then the ID of the write function which is 1 in NASM is stored in rax register. The file descriptor is stored in rdi. The buffer in rsi and the count in rdx. The function parameters are stored in rdi,rsi,rdx,and a few other registers respectively. So using this we carry out a few print subroutines using the write sys call whose data is stored in the .data section.

Now in the .bss part we statically create two variables. One is 8 bytes long and other is 64 bytes long. The 8 byte one say strPosStorage is used for printing the number in both ASCII and string form.

To print in ASCII we simply make use of the fact that in assembly int is never used and can never be printed. So anything which is a number is stored in a register or anywhere else in the form of ASCII characters. So we simply need to assign the integer value that was taken as parameter (currently stored in rdi register) and

just assign it to r9 (an idle register). Now this r9 points to the address of strPosStorage and if we print this using the print subroutine using write sys call then we get the ASCII characters as asked in question.

As far as printing the integer as string is concerned, we can simply achieve this by storing it in various addresses in strStorage as an array of characters and then we keep avoiding by 10 and adding to an array. At the end we keep dividing the number by 10 and add the remainder to the array. And store the quotient in the same array and this loop goes on.

say we have 123
123/10=12; 123%10=3
12/10=1;12%10=2
1/10=0;1%10=1

So now we have a 321 which is then reversed using decrement counters as were increment being used initially. However this was done as an additional subroutine only.

Now we need to transfer control to 'c' function present in c.c . For this we need to modify the stack. Now in the code r10 stores a reference to function A and if we call ret, the stack refers to this location and goes here. However if we use extern c in our .text and then push c into the stack and pop r10 . We remove details of A and our function naturally returns to C instead of A. However this could be done without popping r10 but if r10 remains in stack then we have a scope of returning to A which is not needed so we just remove it from stack and push c. This way when ret is called we actually transfer control to function c. Where we print and exit with exit(0) so stack manipulation using push and pop and ret and extern helps us do this. Then we terminate the programme.Stack modification is shown below.