

INHA UNIVERSITY TASHKENT
DEPARTMENT OF CSE & ICE
SPRING SEMESTER 2020
SOC 2040 - SYSTEMS PROGRAMMING
LAB ASSIGNMENT 3

INSTRUCTIONS :

- ALL LAB ASSIGNMENTS ARE TO BE COMPLETED IN GROUPS OF 4 TO 6 STUDENTS.
- LAB ASSIGNMENT REPORT SHOULD BE PREPARED USING THE LAB ASSIGNMENT 3 REPORT TEMPLATE PROVIDED.
- ONE HARD COPY OF THE LAB ASSIGNMENT REPORT OF EACH GROUP SHOULD BE HANDED IN AT THE OFFICE BY THE GROUP LEADER.
- EVERY MEMBER OF THE TEAM MUST UPLOAD THE SOFTCOPY OF THE REPORT AT THE E- CLASS PORTAL.
- FOR ALL PART-A ASSEMBLY LANGUAGE PROGRAMMING PRACTICE QUESTIONS, YOU NEED TO PROVIDE SCREENSHOTS OF PROGRAM ENTERED, INPUTS GIVEN AND RESULTS.
- FOR ALL PART-B ASSEMBLY LANGUAGE PROGRAMMING QUESTIONS, YOU NEED TO PROVIDE SCREENSHOTS OF ALGORITHM, PROGRAM ENTERED, INPUTS GIVEN AND RESULTS.
- FOR ALL PART-C QUESTIONS, YOU NEED TO PROVIDE DDD DEBUGGER SCREENSHOTS AFTER EXECUTION OF EACH INSTRUCTION IN TRACE MODE BY SHOWING THE INVOLVED REGISTER CONTENTS, MEMORY CONTENTS AND STACK CONTENTS IN CASE OF STACK OPERATIONS.
- ALL FILES CONTAINING ASSEMBLY LANGUAGE PROGRAMS WRITTEN AND EXECUTED FOR ALL THE PART-A, PART-B AND PART-C QUESTIONS SHOULD BE UPLOADED AT THE E-CLASS PORTAL ALONG WITH THE LAB ASSIGNMENT 3 REPORT.
- LAST DATE FOR SUBMISSION OF THE LAB ASSIGNMENT REPORT IS 30th MARCH 2020
- LATE SUBMISSIONS ARE NOT ENTERTAINED, ADHERE TO THE DEADLINE STRICTLY
- READ THE QUESTIONS CORRECTLY & CAREFULLY

[illegible]

ENTER THE FOLLOWING PROGRAMS USING gedit EDITOR, ASSEMBLE THEM USING GNU as ASSEMBLER, LINK THEM USING ld TO CREATE EXECUTABLE CODE AND EXECUTE THEM.

1. Program to print Hello World

```
# program helloworld.s
# To create object file using GNU assembler as
#      $as -gstabs helloworld.s -o helloworld.o
# To create an executable file after linking
#      $ld helloworld.o -o helloworld
# To execute helloworld
#      $./helloworld

.global  _start
.data
    message: .ascii "Hello World\n"

.text
_start:
    movq $1,%rax      #system call 1 is write
    movq $1,%rdi      #file handle 1 is stdout
    movq $message,%rsi #address of string in memory to output
    movq $13,%rdx     #number of bytes
    syscall           #invoke operating system to print

    movq $60,%rax     #system call 60 is exit
    xorq %rdi,%rdi    #return code 0
    syscall           #invoke operating system to exit
```

2. Program – nextascii.s

**#The following program takes each character in the string stored at message
#in memory and converts it to its next ascii character, stores it in buffer
#memory at nxtasciim :**

```
# To create object file using GNU assembler as
# $as -gstabs nextascii.s -o nextascii.o
```

```

# To create an executable file after linking
#          $ld nextascii.o -o nextascii
# To execute nextascii
#          $./nextascii

```

```

.global _start
.data
message: .asciz "SYSTEM PROGRAMMING Lab Assignment 3 \n"
nxtasciim: .space 80

```

```

.text
_start:  movq $1, %rax
         movq $1, %rdi
         movq $message, %rsi
         movq $39, %rdx
         syscall

         movq $37, %rcx
         movq $nxtasciim, %rdi
         movq $message, %rsi
up:      movb (%rsi), %al
         addb $1, %al
         movb %al, (%rdi)
         incq %rsi
         incq %rdi
         decq %rcx
         jnz up
         movb (%rsi), %al
         movb %al, (%rdi)
         incq %rsi
         incq %rdi
         movb (%rsi), %al
         movb %al, (%rdi)

         movq $1, %rax
         movq $1, %rdi
         movq $nxtasciim, %rsi
         movq $37, %rdx
         syscall

         movq $60, %rax
         xorq %rdi, %rdi
         syscall

```

3. Program - echoline.s

**#The following program just reads one line from stdin, using sys_read, then
echoes it to stdout:**

**# To create object file using GNU assembler as
\$as -gstabs echoline.s -o echoline.o
To create an executable file after linking
\$ld echoline.o -o echoline
To execute rwstring
\$./echoline**

```
.global _start  
.data  
    buf: .skip 1024  
  
.text  
_start:  
    movq $0, %rdi      # stdin file descriptor  
    movq $buf, %rsi    # buffer  
    movq $80, %rdx     # buffer length  
    movq $0, %rax      # sys_read  
    syscall  
  
# The sys_read function returns the number of bytes that were read in the rax  
# register, so we use that as the message length in the call to sys_write.  
  
    movq $1, %rdi      # stdout file descriptor  
    movq $buf, %rsi    # message to print  
    movq %rax, %rdx    # message length  
    movq $1, %rax      # sys_write  
    syscall  
  
    movq $0, %rdi      # exit with return code =0  
    movq $60, %rax  
    syscall             #sys_exit
```

**(You need to try out atleast 32 different string inputs and Provide
screenshots of the Program entered, Inputs given and all the
results.)**

4. Program - upper2lower.s

**#The following program reads a string from keyboard, stores it in memory
#starting at location givenstr and then takes each uppercase alphabet in
#that string and converts it to its lowercase alphabet and stores it in memory
#at location lowerstr :**

**# To create object file using GNU assembler as
\$as -gstabs upper2lower.s -o upper2lower.o
To create an executable file after linking
\$ld upper2lower.o -o upper2lower
To execute upper2lower
\$./upper2lower**

```
.global _start  
.data  
    message: .asciz "ENTER A STRING :"  
    givenstr: .skip 100  
    lowerstr: .space 100  
.text  
_start:    movq $1, %rax        #sys_write  
          movq $1, %rdi  
          movq $message, %rsi  
          movq $16, %rdx  
          syscall  
  
          movq $0, %rdi        #sys_read  
          movq $givenstr, %rsi  
          movq $100, %rdx  
          movq $0, %rax  
          syscall  
  
          movq %rax, %r8  
          movq %r8, %rcx  
          decq %rcx  
          movq $givenstr, %rsi  
          movq $lowerstr, %rdi  
up:       movb (%rsi), %al  
          cmp $65, %al  
          jl down1  
          cmp $90, %al  
          jg down1  
          addb $32, %al  
down1:    movb %al, (%rdi)
```

```

incq %rsi
incq %rdi
decq %rcx
jnz up
movb (%rsi), %al
movb %al, (%rdi)

movq $1, %rax
movq $1, %rdi
movq $lowerstr, %rsi
movq $r8, %rdx
syscall

movq $60, %rax
xorq %rdi, %rdi
syscall

```

(You need to try out atleast 32 different string inputs and Provide screenshots of the Program entered , Inputs given and all the results.)

5. Program - prevascii.s

**#The following program reads a string from keyboard, stores it in memory
#at location givenstr and then takes each character from that location and
#converts it to its previous ascii character stores it in buffer prevchars:
To create object file using GNU assembler as
\$as -gstabs prevascii.s -o prevascii.o
To create an executable file after linking
\$ld prevascii.o -o prevascii
To execute nextchar
\$./prevascii**

```

.global _start
.data
    message: .asciz "ENTER A STRING :"
    givenstr: .skip 100
    prevchars: .space 100
.text
_start:  movq $1, %rax           #sys_write
        movq $1, %rdi
        movq $message, %rsi
        movq $16, %rdx
        syscall

```

```

movq $0, %rdi                                #sys_read
movq $givenstr, %rsi
movq $60, %rdx
movq $0, %rax
syscall

movq %rax, %r8
movq %r8, %rcx
decq %rcx
movq $givenstr, %rsi
movq $prevchars, %rdi
up: movb (%rsi), %al
addb $1, %al
movb %al, (%rdi)
incq %rsi
incq %rdi
decq %rcx
jnz up
movb (%rsi), %al
movb %al, (%rdi)

movq $1, %rax                                #sys_write
movq $1, %rdi
movq $prevchars, %rsi
movq $r8, %rdx
syscall

movq $60, %rax                                #sys_exit
xorq %rdi, %rdi
syscall

```

(You need to try out atleast 32 different string inputs and Provide screenshots of the Program entered, Inputs given and all the results.)

6. Program - intlist.s

```

# -----
# A 64-bit Linux application that reads and prints list of n integers
# The input value n is to be read from the keyboard.
# Functions called : puts, scanf, printf
# This routine needs to be linked with C library functions
# To create object file using GNU assembler as
# $as -gstabs intlist.s -o intlist.o

```

```
# To create an executable file after linking
# $ld -dynamic-linker /lib64/ld-linux-x86-64.so.2 intlist.o -o intlist -lc
# Execute the code listint generated using the following command :
# $ ./intlist
```

```
# -----
```

```
.global _start
```

```
# Data Declaration Section
```

```
.data
n:          .quad 0
listn:      .space 800
format1:    .asciz "ENTER LIST OF n=%ld NUMBERS\n"
            # asciz puts a 0 byte at the end
format2:    .asciz "THE GIVEN LIST OF n=%ld NUMBERS\n"
message:    .asciz "ENTER A VALUE FOR n : "
f1:         .string "%ld"
f2:         .string "%ld\n"
```

```
# Program Text (code) section
```

```
.text
```

```
_start:
```

```
# put message by calling puts
movq $message, %rdi #First message address (or pointer) parameter in %rdi
call puts          # puts(message)
```

```
# read n, number of integers
```

```
movq $0, %rax
movq $f1, %rdi # put scanf 1st parameter (format f1) in %rdi
movq $n, %rsi # put scanf 2nd parameter - pointer to location n in %rsi,
call scanf    # scanf(f, pointer n)
```

```
# print message by calling printf
```

```
movq n, %rax
pushq %rcx # caller-save register
movq $format1, %rdi # put printf 1st parameter (format1) in %rdi
movq %rax, %rsi #put printf 2nd parameter ( n ) in %rsi, n is the value to be printed
xorq %rax, %rax
call printf    # printf(format1, n)
popq %rcx     # restore caller-save register
```

```
#Read list elements from Keyboard by calling scanf
```

```
movq $listn, %rdx
movq n, %rcx
```

```
up:
```

```
pushq %rdx
pushq %rcx
pushq %rbp
movq $0, %rax
movq $f1, %rdi # put scanf 1st parameter (format f1) in %rdi
movq %rdx, %rsi # put scanf 2nd parameter - pointer to list element location in %rsi
call scanf    # scanf(f1, pointer to list element)
```



```

popq %rbp
popq %rcx
popq %rdx
addq $8, %rdx
decq %rcx
jne  up

```

Print message by calling printf

```

movq n, %rax
pushq %rcx                # caller-save register
movq $format2, %rdi        # put printf 1st parameter (format2) in %rdi
movq %rax, %rsi            # put printf 2nd parameter ( n ) in %rsi, n is the value to be printed
xorq %rax, %rax
call printf                # printf(format2, n)
popq %rcx                 # restore caller-save register

```

Print list of n elements read from keyboard

```

movq $listn, %rdx
movq n, %rcx

```

up1:

```

pushq %rdx
pushq %rcx
pushq %rbp
movq $0, %rax
movq $f2, %rdi            # put printf 1st parameter (format f2) in %rdi
movq (%rdx), %rsi         # put printf 2nd parameter - list element value in %rsi
call printf                # printf(f2, list element value)
popq %rbp
popq %rcx
popq %rdx
addq $8, %rdx
decq %rcx
jne  up1

```

```

movq $60, %rax            # syscall to return 0
xorq %rdi, %rdi
syscall

```

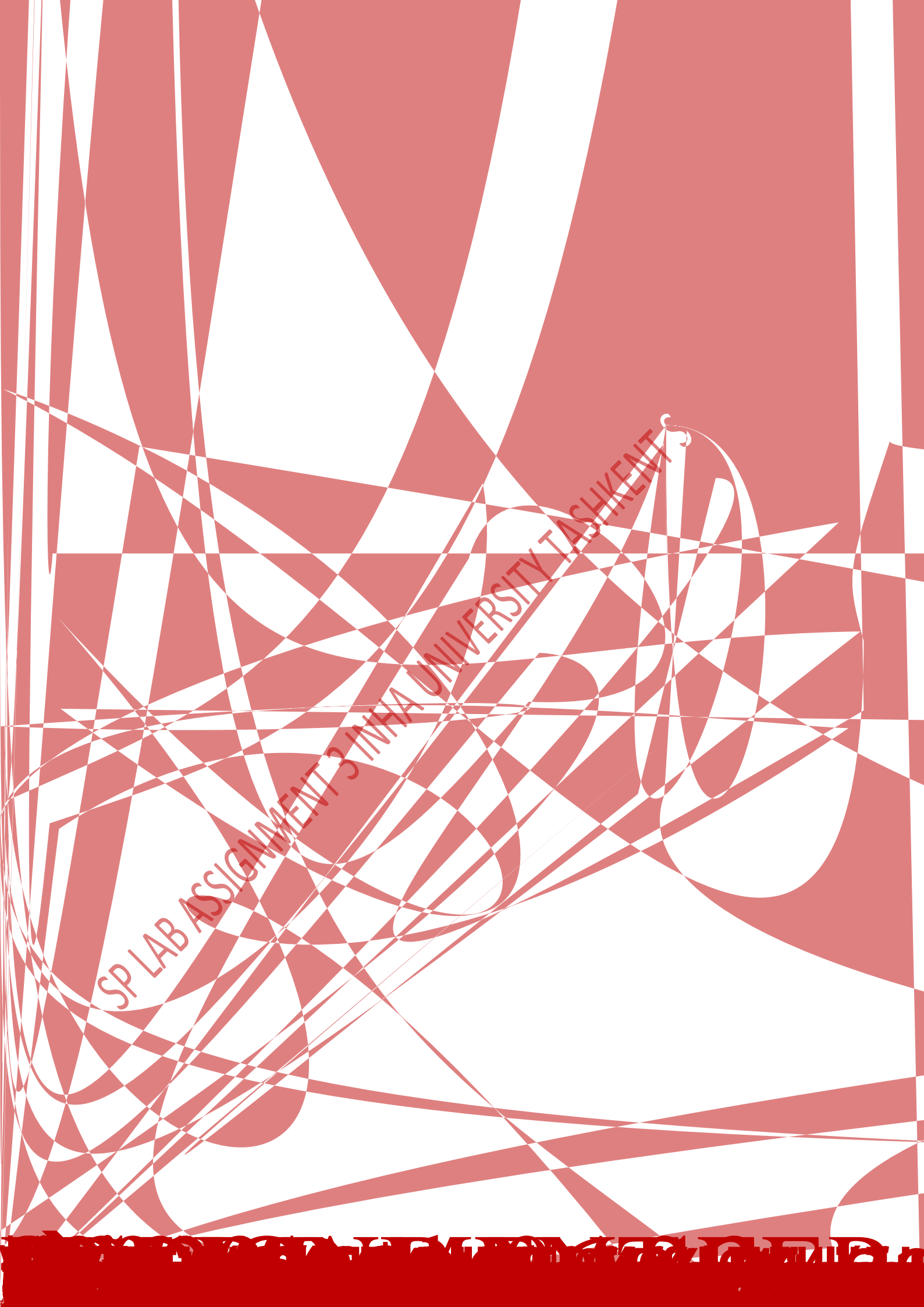
(You need to try out atleast 32 different string inputs and Provide screenshots of the Program entered , Inputs given and all the results.)

7. Program - fib.s

```

# -----
# A 64-bit Linux application that writes the first 90 Fibonacci numbers.
# It needs to be linked with a C library.
# To create object file using GNU assembler as

```



8. Program - fibio.s

```
# -----  
# A 64-bit Linux application that writes the first n Fibonacci numbers.  
# The input value n is to be read from the keyboard.  
# Functions called : puts, scanf, printf  
# This routine needs to be linked with C library functions  
# To create object file using GNU assembler as  
#          $as -gstabs fibio.s -o fibio.o  
# To create an executable file after linking  
# $ld -dynamic-linker /lib64/ld-linux-x86-64.so.2 fibio.o -o fibio -lc  
# To execute fibio  
#          $./fibio  
#-----  
  
    .global _start  
    .data  
        message: .asciz "ENTER A VALUE FOR n:"  
                                     # asciz puts a 0 byte at the end  
        format:  .asciz "%20ld\n"  
        format1: .asciz "THE LIST OF FIRST n=%8ld FIBONACCI NUMBERS\n"  
        f:       .string "%d"  
        x:       .quad 0  
  
    .text  
_start:  
    pushq    %rbx    # save this register content since we will use it  
  
# To call C function puts(message pointer); first store message pointer in #  
# register %rdi as follows          message pointer → %rdi  
  
    movq    $message, %rdi    #message address (or pointer) parameter in %rdi  
    call    puts              # puts(message)  
  
#READ VALUE OF n FROM KEYBOARD AND STORE IT IN MEMORY LOCATION x  
    pushq    %rbp  
# To call C function scanf(format, &x); put the parameters in registers  
# as follows #format →%rdi, &x → %rsi, 0 → %rax  
  
    movq    $0, %rax  
    movq    $f, %rdi    # put scanf 1st parameter (format f - see .data section) in %rdi  
    movq    $x, %rsi    # put scanf 2nd parameter (pointer to location x - see .data section) in %rsi  
                    # value n read from keyboard will be stored in location x  
    call    scanf        # scanf(f, pointer x)  
    popq    %rbp
```

PRINT VALUE OF n in x on the screen

```
    movq    x, %rax # %rax ← n , i.e. Contents of memory location x
    pushq   %rax    # caller-save register
    pushq   %rcx    # caller-save register
# To call C function printf(format, arg); put the parameters in registers as
# follows format → %rdi, arg → %rsi, 0 → %rax
```

```
    movq    $format1, %rdi # put printf 1st parameter (format1 - see .data section) in %rdi
    movq    %rax, %rsi    # put printf 2nd parameter ( n ) in %rsi, n is the value to be printed
    xorq    %rax, %rax
    call     printf        # printf(format1, current_number)
    pop     %rcx          # restore caller-save register
    pop     %rax          # restore caller-save register
```

#COMPUTING FIBONACCI NUMBERS

```
    movq    %rax, %rcx    # rcx will countdown to 0
    xorq    %rax, %rax    # rax will hold the current number
    xorq    %rbx, %rbx    # rbx will hold the next number
    incq    %rbx          # rbx is originally 1
# We need to call printf, but we are using eax, ebx, and ecx. printf may
# destroy eax and ecx so we will save these before the call and restore them
# afterwards
```

print:

```
    push    %rax          # caller-save register
    push    %rcx          # caller-save register
    movq    $format, %rdi # put printf 1st parameter (format1 - see .data section) in %rdi
    movq    %rax, %rsi
# put printf 2nd parameter (current_number - currently generated
# Fibonacci number ) in %rsi; currently generated fibonacci number is the
# value to be printed
```

```
    xorq    %rax, %rax
    call     printf        # printf(format, current_number)
    pop     %rcx          # restore caller-save register
    pop     %rax          # restore caller-save register
    movq    %rax, %rdx    # save the current number
    movq    %rbx, %rax    # next number is now current
    addq    %rdx, %rbx    # get the new next number
    decq    %rcx          # count down
    jnz     print         # if not done counting, jump to print
    pop     %rbx          # restore rbx before returning to operating system
```

```
    movq    $60, %rax     # syscall to return 0
    xorq    %rdi, %rdi
```

syscall

(You need to try out atleast 32 different input values for n and Provide screenshots of the Program entered, Inputs given and all the results.)

UF WY²G⁷P=; 2 9% XXJR GQ %Q SL ZFL J^UWTL WF RRLSL %T S^QSZJ
UWT LWF RRLSL %V ZJXYNT SX

9. Write an X86-64 assembly language program to read a string of signed decimal number containing maximum 10 digits from the keyboard using system call routine **d j f i** and convert this to integer form and store it in a register `%rax` and print this using `printf` with format specifier `%ld`.
(You need to try out at least 32 different string inputs and Provide screenshots of the Program entered , Inputs given and all the results.)

(ii) Input :
Js j %% rsl %? HELLO GOOD MORNING

Output:
Ymj %n j s% rsl %?HELLO GOOD MORNING% %t %%f qsi t r j

(You need to try out atleast 32 different input strings and Provide screenshots of the Program entered, Inputs given and all the results.)

12. Write an X86-64 assembly language program to read a r j fl j (string of characters of any length (maximum 1024 characters)) from the keyboard and perform the following:
ENCRYPT the message by adding 9 to every character in the message and store this encrypted message in a separate memory area. Now print the original message and the encrypted message on the screen.

Kt %b fr uq %?
Nu %? Provide the following prompt message before reading the input
Js j %% j fl j %?Mj qt %Mt % j %t D
T u ?
T h r s f q % j fl j % ?Mj qt %Mt % j %t D
Jsh u ji % j fl j %Vs /.V .o s.g M

(You need to try out atleast 32 different messages of varying length and Provide screenshots of the Program entered, Inputs given and all the results.)

13. Write an X86-64 assembly language program to read jsh u ji % j fl j generated in Q12) from the keyboard and perform the following:
DECRYPT the message by subtracting 9 from every character in the message and store this decrypted message in a separate memory area. Now print the Encrypted message and the Decrypted message on the screen.

Kt %b fr uq %?
Nu %? Provide the following prompt message before reading the input
Jsh u ji % j fl j %Vs /.V .o s.g M
T u ?
l jh u ji % j fl j %?Mj qt %Mt % j %t D

(You need to try out atleast 32 different messages of varying length and Provide screenshots of the Program entered, Inputs given and all the results).

14. Write an X86-64 assembly language program to implement Hamming code algorithm suggested by Hamming to detect and correct single bit errors in given Data. You are required to read a Data of M bits (M bits can be 8 to 16 bits). Length(number of bits) of the number must be read from the keyboard first and then read the number containing the specified bits. Your program should determine the number of Check bits (K) required, then compute the check bits values using the hamming code algorithm. You are required to perform the following:
- Store the Data in the memory along with the Check bits (i.e. M + K bits) in the same order as specified in the algorithm. Then print the output in the following way.

Kt % fr ug %8%

№ 2

J s j %Qj sl n%k/mj%F YF %R B%56665561% r rsl %nf %a %mj%f q j %s j ji
J s j %a %n%F YF %R B%56665561%am %a %mj%F YF %n %s j ji

TZYUZYX°/8

Ln j s%FYF%GNX%R?

$\frac{7}{7} = \frac{0}{100} < \frac{0}{100}$; $\frac{7}{100} = \frac{7}{100}$ 9% 8% 7% 6%
 $\frac{0}{7} = \frac{0}{100}$ 5% 5% 6% 6% 5% 6% 5% 6%

HMJHP%GNX%WUVZNUI %B%B%A

HMJHP%GNX%MFRRSL%HTI J.%P%P%

99%79%6

[illegible]

F Y F % X Y T W I % S % M J % R J R T W % Q T S L % N Y M % H M J H P % G N X - R O P . ?

[illegible][illegible]

- b) Now introduce a SINGLE BIT ERROR in the data bits or check bits by asking the user to enter the bit number (any value between 1 to $M+K$ bits of DATA with Hamming code (CHECK bits)). Then recompute the CHECK BITS for the modified $M+K$ bits and then determine the

T u ? Number of alphabets : 82
 Number of numerals : 11
 Number of special characters : 34
 Total number of characters in the string : 127
 Total number of words in the string : 23

(You need to try out atleast 32 different string inputs and Provide screenshots of the Program, Inputs given and all the results.)

16. Write an X86-64 assembly language program to read a text (string of characters of any length (maximum 1024 characters)) from the keyboard and perform the following operations on the text :
- Convert all the uppercase alphabets in the text to lowercase and print the output on the screen
 - Convert all the lowercase alphabets in the text to uppercase and print the output on the screen
 - Convert first letter of every word in the text to uppercase if it is in lowercase and all other letters in every word to lower case and print the output on the screen

```

Kt % fr uq %?%
Nu %?
Js j %%% rsl %?%
F %uj %mj %NMF%ZSN JWKY %FXMPJSY%st rhf rt s?RN YJWR%
J fr rsf rt s %mj %ht si h ji %s %mj %JNL MYM% JJP tk %mj %u rsl %
jr j j %575
  
```

a)

```

T u ?%
f %uj %nj %rsnf% snj n % f mpjs %st rhf rt s?r n j r %
j fr rsf rt s %mj %ht si h ji %s %mj %n mn% jjp tk %mj %u rsl %
jr j j %575
  
```

b)

```

T u ?%
FX%UJW%YMJ%NMF%ZSN JWKY %YFXMPJSY%STYKNHFYNT S?%
RN YJWR%J JFRN SFYNT SX% NQ%GJ %HT SI ZHYJI %S %YMJ %JNL MYM%
\ JJP TK %YMJ %XUW %L %XJR JXYJW %575
  
```

c)

```

T u ?
F %Uj %Ymj %Nmf %Zsn j n %Yf mpjs %St rhf rt s?%Rn j r %
J fr rsf rt s %mj %ht si h ji %s %mj %n mn% jjp tk %mj %X u rsl %
Xjr j j %575
  
```

(You need to try out atleast 32 different text messages and Provide screenshots of the Program, Inputs given and all the results.)

17. Write an X86-64 assembly language program to find the factorial of a given number n. The input 'n' should be read from the keyboard and the output should be printed on the screen.

Kt %b fr uq %b

Provide the following prompt message before reading the input
Js j %mj %f q j %t %s %b

T u %b then the output should be printed as follows:
s %b 595

(You need to try out atleast 32 different input values for n and to Provide screenshots of the Program, Inputs given and all the results.)

18. Write an X86-64 assembly language program to generate all prime numbers between 1 and n. The input 'n' should be read from the keyboard and the output should be printed on the screen.

For example :

Provide the following prompt message before reading the input
Js j %f q j %t %s %b 5

T u - Then the output should be printed as follows:

U r j %s r g j %g j j s %6 %si %5 %j %26 1 1 < 166 168 16 < 16 %

% t %s j j i %t % % % q f %87 % m k j j s %f q j %t %s u %s %
f si %U t n j %h j j s nt %k %m j %U t l fr %s j j i 1 %s u %
l n j s %si %q %m j %j q 3

19. You are given the following 'C' program containing recursive function to count the number of 1's in a given number x. Write the equivalent recursive function in x86-64 assembly language and Run the program for different values of x and display the result on the screen.

```
int main()
{
    long int x, onesc;
    scanf("%ld", &x);
```

```

    onesc= rcount1s(x);
    printf("Number of 1s in the given number %ld = %ld\n", x, onesc);
}

```

```

long rcount1s(long y)
{
    if(y == 0)
        return 0;
    else
        return ( y & 0x1) + rcount1s(y >>= 1);
}

```

```

- t %s j j i %t % %t %f q f %87% m j j s %f q j %t %s u % %f s i %
U t n j %h j j s n t %k %m j %U t l f r %s j j i %s u %n j s %s i %q %
m j %j q 3

```

20. a) You are given the following 'C' program to find the sum and average of the given list of integers specified in the data segment. Also you are required to find the number of occurrences of a given key in the list. Translate the program to extended C (if – goto version) and then write the equivalent function in x86-64 assembly language. Run the program and display the result on the screen.

```

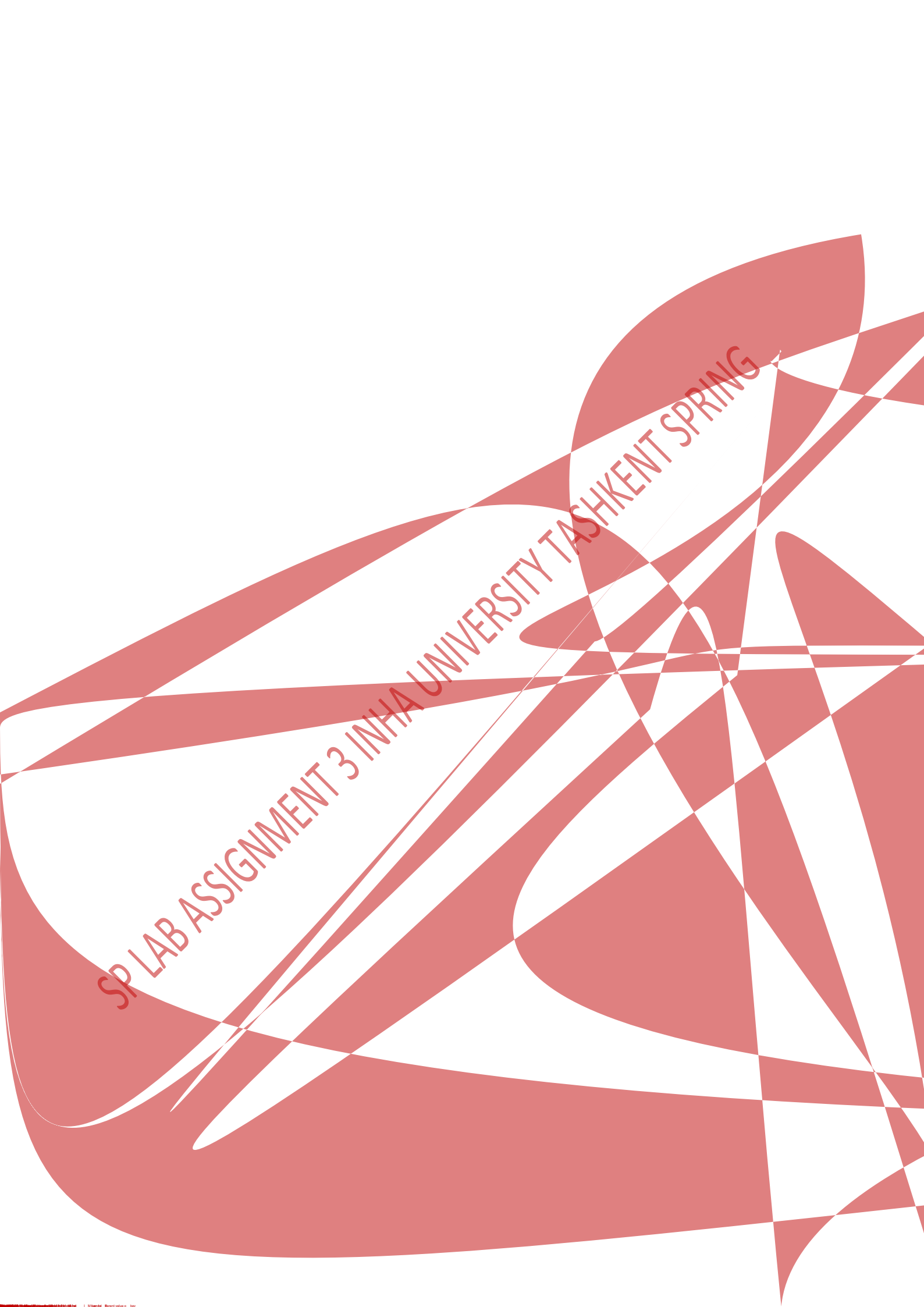
# Program file name : program.c
int main()
{
    int i, sum, count;
    float avg;
    int n=20;
    int list[]={12,23,32,84,121,34,23,32,93,22,56,32,948,123,99, 23,32,289,99,34};
    int key=32;

    printf("Elements of the list : \n");
    for(i=0; i<n; i++)
        printf("%d\t", list[i]);

    sum=0;
    for(i=0; i<n; i++)
        sum += list[i];
    printf("sum of elements in the list= %d \n", sum);
    avg = (float)sum/ (float)n;
    printf("Average of elements in the list= %f \n", avg);

    // To search for a key in the above list
    count =0;
}

```



SP LAB ASSIGNMENT 3 INHA UNIVERSITY TASHKENT SPRING

```

f i i v      g 1% f
gv          f 1% g
%6s j l v %6s g
%6s gv %6s g 1% f
%6s t v      ; 55551% n
r t v %6s ; :: 8: 1% i
r t v %6s r 1% f %
r r q %6s i
%av t
n n v %6s n
j v %6s r i n
os      i t s6
u6? hr uv %6s r i n %6s
j q %6s f q %6s
r t gv %6s f q %6s f %6s
or u      i t s7
i t s6? u mv      f
u mv      g
ut uv      f
ut uv      g
movq      %rdi, %rax
subq      %rsi, %rax
movq      %rsi, %rdx
subq      %rdi, %rdx
cmpq      %rsi, %rdi
cmovle    %rdx, %rax
cmp      %rdx, %rax
jz      up1
down2: xorq      %rax, %rbx
xorq      %rbx, %rax
xorq      %rax, %rbx
andq      %rsi, %rax
orq      %rsi, %rax
xorq      %rax, %rax
notq      %rax
negq      %rax
incq      %rax
decq      %rax
addq      $99, %rax
leaq      (%rax, %rax, 8), %rax
leaq      (%rax, %rax, 4), %rax
salq      $3, %rax
negq      %rax
sarq      %rax

```

j

You are required to run the program using **LSZ%i i** debugger by setting the **g j fput rs** at the first instruction **r t v** and the last instruction **j %** initially and then execute each instruction in single step (single instruction execution or Trace mode) mode by clicking the **j u % g t s** on the Command Tool window.

```
Ymj%r frs%u ut j%t k% m%j j hn j%a %t% si j f si %j fh n%  
rs h r t s%a q f q %g %t rsl %t s%mj %k j h %k %mj %s h r t s %s %  
mj %t s j s %k %j q f s %j l n j %si %q l %g n % %t si n r t s %t i j 3%
```

```
t % j %j v n j i %t %f p j %s f u m t %k %mj %i i %j g l l j %f r s %  
h j j s %m t rsl %X t h j %r s i t 1 % f f %r s i t 1 % l G %h t s t q %s i %  
R f h m s j %h t i j %r s i t 1 % H t r r f s i %t t q %r s i t 1 % f s i %W j l n j %  
r s i t %m t rsl %mj %t s j s %k %s t q j i %j l n j %si %q l %g j k t j %  
f s i %k j %mj %j h r t s %k %f h n %s h r t s %f s i %q t %j h r g j %mj %  
t u j f r t s %j k t r j i 3%
```

```
(% t l fr %r g % r i i i 3 %
```

```
# Assemble using f command -
```

```
f %2 fg % r i i i 3 %2 % r i i i 3 %
```

```
# Link using the command -
```

```
q %r i i i 3 %2 % r i i i
```

```
# Run the executable code in trace mode using the command -
```

```
i i i % r i i i
```

22. You are given the following x86-64 assembly language program containing recursive function to find the factorial of a given number n. Assemble the program using GNU **f** assembler, link and run the executable code for the given value of n using the GNU ddd debugger in step mode and trace the stack structure for all the recursive calls and returns.

```
# program file ffact.s
```

```
# Assemble using as command - $as -gstabs ffact.s -o ffact.o
```

```
# Link using the command - $ld ffact.o -o ffact
```

```
# Run the executable code in trace mode using the command - $ddd ffact
```

```

.global    _start

.data
    n:      .quad 0x00000000000000009
    nfact:   .quad 0x0000000000000000

.text
_start:

    movq    n, %rbx
    movq    %rbx,%rax
    call    ffact
    movq    $nfact, %rdi
    movq    %rax, (%rdi)
    movq    $60, %rax    # system call 60 for exit
    xorq    %rdi, %rdi    # return code 0 syscall
    syscall

ffact:
    cmpq    $1, %rbx
    je      L1
    decq    %rbx
    pushq   %rbx
    call    ffact
    popq    %rbx
    mulq    %rbx
L1:
    ret

```
