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**

QUESTIONS :

PART - A : X86-64 ASSEMBLY LANGUAGE PROGRAMMING ON LINUX PRACTICE QUESTIONS

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
```

```
#               $as  -gstabs  fib.s   -o  fib.o
# To create an executable file after linking
# $ld  -dynamic-linker  /lib64/ld-linux-x86-64.so.2  fib.o   -o fib  -lc
# To execute fib
#   $./fib
# ---------------------------------------------------------------------------

        .global _start
        .data
                format:     .asciz  "%20ld\n"
        .text
_start:
          pushq    %rbx              # we have to save this since we use it
          movq     $90, %rcx         # ecx 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 rax, rbx, and rcx.  Printf may
# destroy rax and rcx so we will save these before the call and  restore
# them afterwards.
# Before calling  printf(format, arg); - parameters are to be stored in
# registers as follows :    format →%rdi,   arg → %rsi,  0 → %rax

print:      pushq    %rax                # caller-save register
            pushq    %rcx                # caller-save register
            movq     $format, %rdi       # set 1st parameter (format)
            movq     %rax, %rsi          # set 2nd parameter (current_number)
            xorq     %rax, %rax          # because printf is varargs
    # Stack is already aligned because we pushed three 8 byte registers
            call     printf          # printf(format, current_number)
            popq     %rcx                # restore caller-save register
            popq     %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

            popq     %rbx            # restore rbx before returning

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

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      #messaage 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.)

PART - B : X86-64 ASSEMBLY LANGUAGE PROGRAMMING ON LINUX PROGRAMMING QUESTIONS

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 **sys_read** 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 atleast 32 different string inputs and Provide screenshots of the Program entered , Inputs given and all the results.)

10. Write an X86-64 assembly language program to read a string of binary number containing maximum 64 bits from the keyboard using system call routine sys_read 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 atleast 32 different string inputs and Provide screenshots of the Program entered, Inputs given and all the results.)

11. Write an X86-64 assembly language program to read a string of characters of any length (maximum 1024 characters) from the keyboard and check whether the given string is a palindrome or not. If the given string is a palindrome then print that it is palindrome otherwise print it is not a palindrome (use System calls). Assemble using **as** assembler, link using **ld** linker and execute on linux system.

For example :

(i) Input : Provide the following prompt message before reading the input
Enter a string : MADAM
Output: output should be printed as follows :
The given string MADAM **is a Palindrome**

(ii) Input :
Enter a string : HELLO GOOD MORNING

Output:
The given string HELLO GOOD MORNING **is not a Palindrome**

(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 **message** (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.

For example :
Input : Provide the following prompt message before reading the input
Enter a message : Hello! How are You?
Output:
Original message : Hello! How are You?
Encrypted message : Qnuux*)QxÇ)j{n)bx~H

(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 **encrypted message** 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.

For example :
Input : Provide the following prompt message before reading the input
Encrypted message : Qnuux*)QxÇ)j{n)bx~H
Output:
Decrypted message : Hello! How are You?

(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:

    a) 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.

    For example :

    Inputs :

    Enter Length of the DATA M= 8, assuming that 8 is the value entered
    Enter 8 bit DATA M= 00111001, this is the DATA bits entered

    OUTPUTS :

    Given DATA BITS M:

D8	D7	D6	D5	D4	D3	D2	D1
0	0	1	1	1	0	0	1

    CHECK BITS REQUIRED K = 4

    CHECK BITS (HAMMING CODE) K :

C8	C4	C2	C1
0	1	1	1

    DATA STORED IN THE MEMORY ALONG WITH CHECK BITS(M+K):

D8	D7	D6	D5	C8	D4	D3	D2	C4	D1	C2	C1
0	0	1	1	0	1	0	0	1	1	1	1

    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

SYNDROME word and print the results considering three possible outcomes of the SYNDROME word.

For Example :
INPUT :
Enter Bit Number : 6 this is the bit number entered
Bit 6 new value (0/1) : 1 this is the bit value entered

OUTPUTS :
M+K BITS WITH SINGLE BIT ERROR

D8	D7	D6	D5	C8	D4	D3	D2	C4	D1	C2	C1
0	0	1	1	0	1	1	0	1	1	1	1

RECOMPUTED CHECK BITS K :

C8	C4	C2	C1
0	0	0	1

SYNDROME WORD :

C8	C4	C2	C1
0	1	1	0

ERROR IN BIT POSITION 0110 – 6 : DATA BIT 3 (D3)

(You need to try out atleast 32 different Data bits of length between 8 to 16 and introduce single bit error in any of the bit positions (Data and Check bit positions)
Provide screenshots of the Program entered, Inputs given and all the results.)

15. Write an X86-64 assembly language program to read a string of characters of any length (maximum 1024 characters) from the keyboard and count the number of alphabets, number of numerals, number of special characters (all characters including space - other than alphabets and numerals), total number of characters and total number of words in the string.

For example :

Input : Provide the following prompt message before reading the input

Enter a string :
#tagIUT Monday – 30th March 2020 – SP LAB ASSIGNMENT 3 - SUBMISSION DEADLINE - INHA !!! University, at Tashkent; Estd. in 2014.

Output: 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 :

a) Convert all the uppercase alphabets in the text to lowercase and print the output on the screen
b) Convert all the lowercase alphabets in the text to uppercase and print the output on the screen
c) 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

For example :
Input :
Enter a string :
As per the INHA UNIVERSITY TASHKENT notification, MIDTERM Examinations will be conducted in the EIGHTH WEEK of the spring semester 2020

a)
Output:
as per the inha university tashkent notification, midterm examinations will be conducted in the eighth week of the spring semester 2020

b)
Output:
AS PER THE INHA UNIVERSITY TASHKENT NOTIFICATION, MIDTERM EXAMINATIONS WILL BE CONDUCTED IN THE EIGHTH WEEK OF THE SPRING SEMESTER 2020

c)
Output:
As Per The Inha University Tashkent Notification, Midterm Examinations Will Be Conducted In The Eighth Week Of The Spring Semester 2020

(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.

For example :

Input : Provide the following prompt message before reading the input
Enter the value for n = 7

Output : then the output should be printed as follows :
n! = 5040

(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 :

Input - Provide the following prompt message before reading the input
Enter value for n : 20
Output - Then the output should be printed as follows:
Prime numbers between 1 and 20 are : 1,3,5,7,11,13,17,19

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

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);
}
```

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

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.

 # Progam 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\n",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;
```

```
for(i=0; i<n; i++)
 if(list[i]==key)
        {
printf("key %d is found at location %d in the list\n",key, i);
count++;
        }
        if(count==0)
                printf("\n Key %d not found in the list \n",key);
  else
                printf("\n Key occurs %d times in the list \n",count);
}
```

b) Modify the program in 20 a) to make it a general program which reads the size of the list n, 'n' number of elements in the list and key to be searched in the list from the keyboard and print the results for different values of n, different values of elements in the list and different value of key in each case.

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

PART - C : SINGLE STEP EXECUTION USING DDD DEBUGGER

21. Create an x86-64 assembly language program (file name **asmddd.s**) containing the following instructions using AT&T terminology which is required to be assembled using GNU **as** Assembler:

```
#Program asmddd.s for debugging with ddd

        .global         _start

        .text
_start: movq    $0x01289AB76EF34567, %rax
        movb    $-1, %al
        movw    $-1, %ax
        movl    $-1, %eax
        movq    $-1, %rax
        movq    $0x01289AB76EF34567, %rbx
        movb    $0xBB, %dl
        movb    %dl, %bl
        movsbq  %dl, %rbx
        movzbq  %dl, %rbx
        movq    $65535, %rbx
```

```
        addq    %rbx, %rax
        subq    %rax, %rbx
        negq     %rbx
        subq     %rbx, %rax
        movq    $60000, %rsi
        movq    $65535, %rdx
        movq    %rsi, %rax
        imulq   %rdx
         cqto
        idivq   %rsi
        testq   %rsi,%rdi
        jnz     down1
up1:    cmpq    %rsi,%rdi
        setl    %al
        movzbq  %al, %rax
        jmp     down2
down1:  pushq   %rax
        pushq   %rbx
        popq    %rax
        popq    %rbx
        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
```

ret

You are required to run the program using **GNU ddd** debugger by setting the **breakpoints** at the first instruction **movq** and the last instruction **ret** initially and then execute each instruction in single step (single instruction execution or Trace mode) mode by clicking the **step button** on the Command Tool window.

The main purpose of this exercise is to understand each instruction clearly by noting down the effect of the instructions on the contents of relevant registers and flag bits or condition codes.

You are required to take snapshots of the ddd debugger main screen showing Source window, Data window, GDB Console and Machine code window, Command tool window, and Register window showing the contents of involved registers and flags before and after the execution of each instruction and also describe the operation performed.

 # program file asmddd.s

Assemble using **as** command -
$as -gstabs asmddd.s -o asmddd.o
Link using the command –
$ld asmddd.o -o asmddd
Run the executable code in trace mode using the command –
$ddd asmddd

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 **as** 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 0x0000000000000009
            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
```

****