LABORATORY 6

TITLE OF THE LABORATORY EXERCISE: SORTING

1. INTRODUCTION AND PURPOSE OF EXPERIMENT

   STUDENTS WILL CREATE ASSEMBLY CODE WITH SORTING TECHNIQUES AND NESTED LOOPS

2. AIM AND OBJECTIVES

   AIM

   TO DEVELOP ASSEMBLY LANGUAGE PROGRAM TO PERFORM SORTING USING NESTED LOOP STRUCTURES

   OBJECTIVES

   AT THE END OF THIS LAB, THE STUDENT WILL BE ABLE TO

   - USE NESTED LOOPS IN ASSEMBLY
   - PERFORM SORTING IN ASCENDING/ DESCENDING ORDER
   - BUILD COMPLEX LOOPING LOGIC IN ASSEMBLY LANGUAGE

3. EXPERIMENTAL PROCEDURE

   1. WRITE ALGORITHM TO SOLVE THE GIVEN PROBLEM

   2. TRANSLATE THE ALGORITHM TO ASSEMBLY LANGUAGE CODE

   3. RUN THE ASSEMBLY CODE IN GNU ASSEMBLER

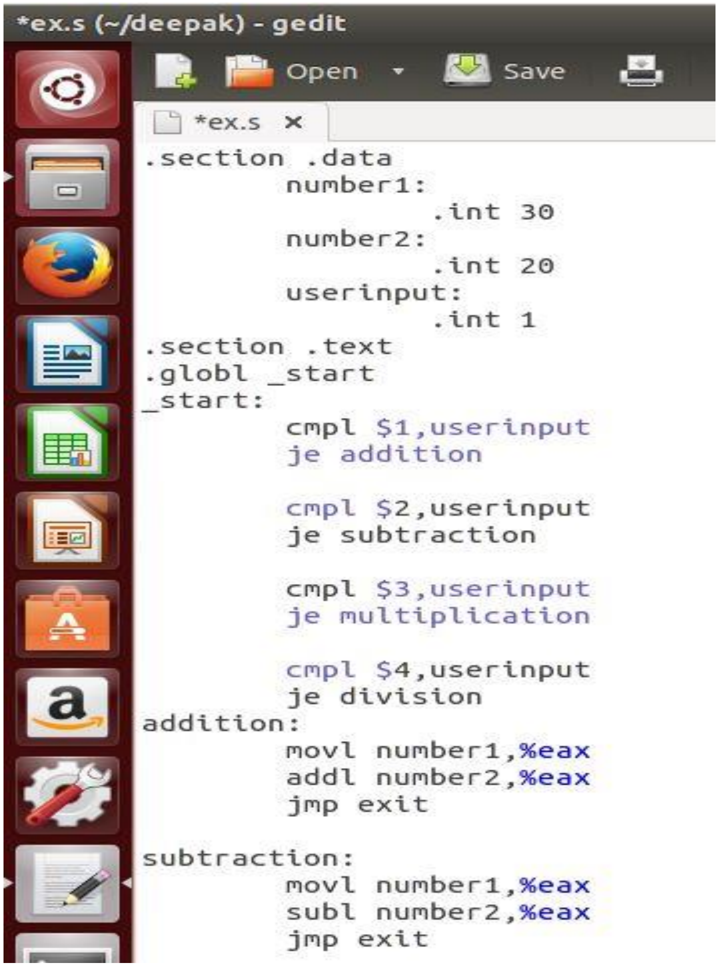
   4. CREATE LABORATORY REPORT DOCUMENTING THE WORK

4. QUESTIONS

   DEVELOP AN ASSEMBLY LANGUAGE PROGRAM TO PERFORM THE FOLLOWING

   1. TO DESIGN CALCULATOR TO PERFORM ALL ARITHMETIC OPERATIONS BASED ON INPUT GIVEN BY USER.
   2. TO PERFORM SWAP OPERATION USING LOGICAL INSTRUCTIONS
   3. TO COMPUTE FACTORIAL OF A NUMBER.
   4. TO FIND SECOND SMALLEST NUMBER IN AN UNSORTED ARRAY.

A. CALCULATIONS/COMPUTATIONS/ALGORITHMS



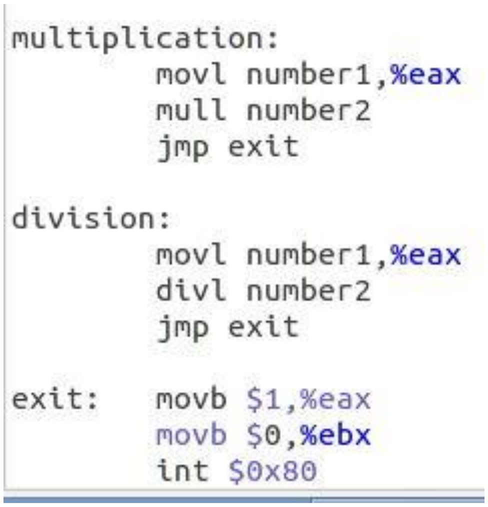
```
.section .data
        number1:
                .int 30
        number2:
                .int 20
        userinput:
                .int 1
.section .text
.globl _start
_start:
        cmpl $1,userinput
        je addition

        cmpl $2,userinput
        je subtraction

        cmpl $3,userinput
        je multiplication

        cmpl $4,userinput
        je division
addition:
        movl number1,%eax
        addl number2,%eax
        jmp exit

subtraction:
        movl number1,%eax
        subl number2,%eax
        jmp exit
```

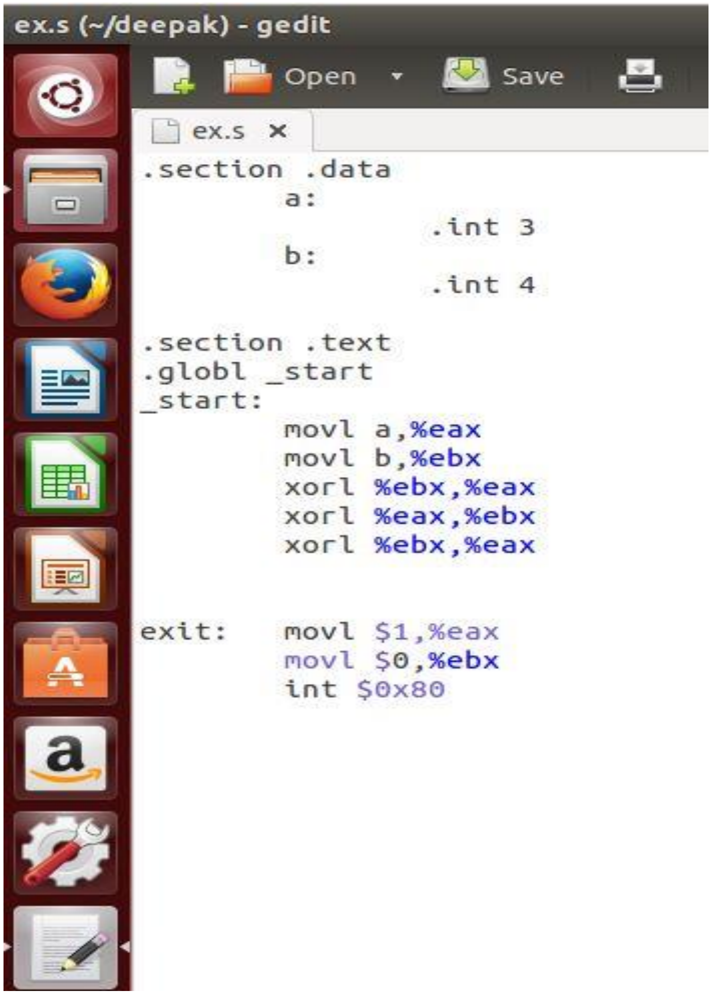


```
multiplication:
        movl number1,%eax
        mull number2
        jmp exit

division:
        movl number1,%eax
        divl number2
        jmp exit

exit:   movb $1,%eax
        movb $0,%ebx
        int $0x80
```

FIG 1 PROGRAM TO DESIGN CALCULATOR TO PERFORM ALL ARITHMETIC OPERATIONS BASED ON INPUT GIVEN BY USER.

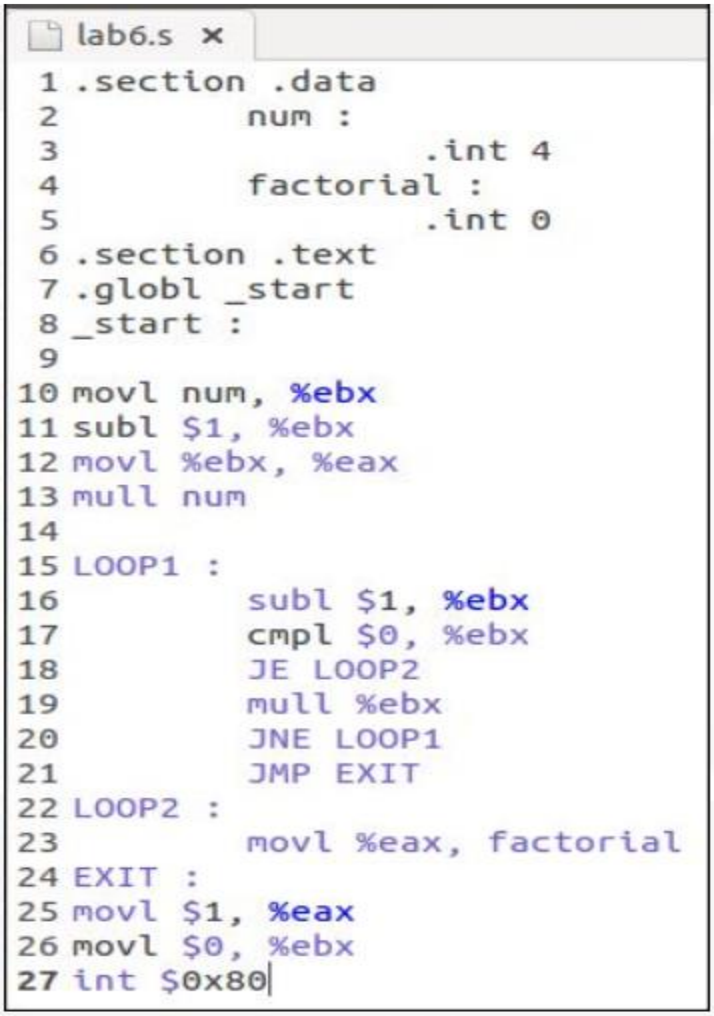


```
.section .data
        a:
                .int 3
        b:
                .int 4

.section .text
.globl _start
_start:
        movl a,%eax
        movl b,%ebx
        xorl %ebx,%eax
        xorl %eax,%ebx
        xorl %ebx,%eax


exit:   movl $1,%eax
        movl $0,%ebx
        int $0x80
```

FIG 2 PROGRAM TO PERFORM SWAP OPERATION USING LOGICAL INSTRUCTIONS

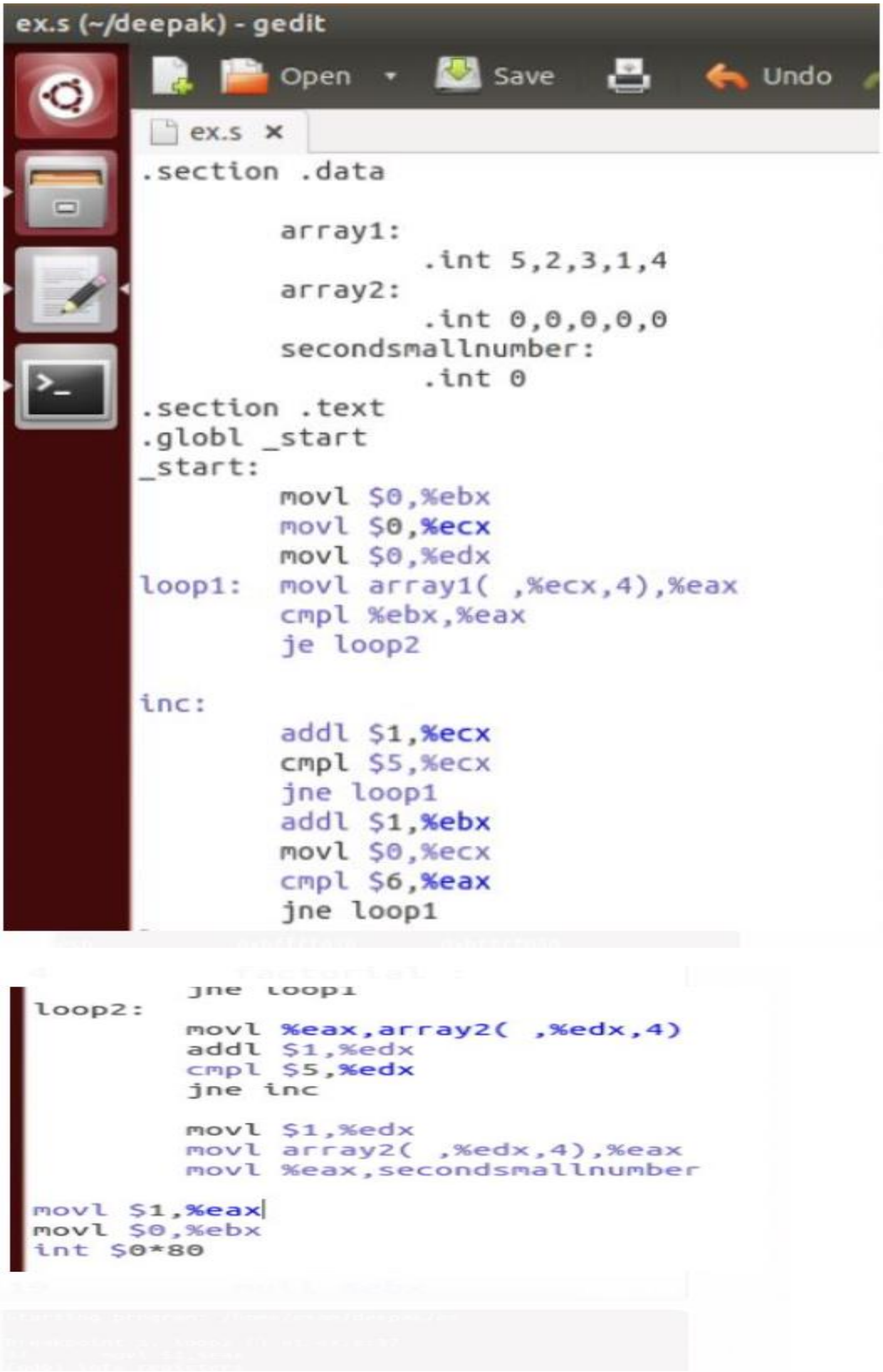


```
.section .data
        num :
                .int 4
        factorial :
                .int 0
.section .text
.globl _start
_start :

movl num, %ebx
subl $1, %ebx
movl %ebx, %eax
mull num

LOOP1 :
        subl $1, %ebx
        cmpl $0, %ebx
        JE LOOP2
        mull %ebx
        JNE LOOP1
        JMP EXIT
LOOP2 :
        movl %eax, factorial
EXIT :
movl $1, %eax
movl $0, %ebx
int $0x80
```

FIG 3 PROGRAM TO COMPUTE FACTORIAL OF A NUMBER.



```
.section .data

        array1:
                .int 5,2,3,1,4
        array2:
                .int 0,0,0,0,0
        secondsmallnumber:
                .int 0
.section .text
.globl _start
_start:
        movl $0,%ebx
        movl $0,%ecx
        movl $0,%edx
loop1:  movl array1( ,%ecx,4),%eax
        cmpl %ebx,%eax
        je loop2

inc:
        addl $1,%ecx
        cmpl $5,%ecx
        jne loop1
        addl $1,%ebx
        movl $0,%ecx
        cmpl $6,%eax
        jne loop1
```

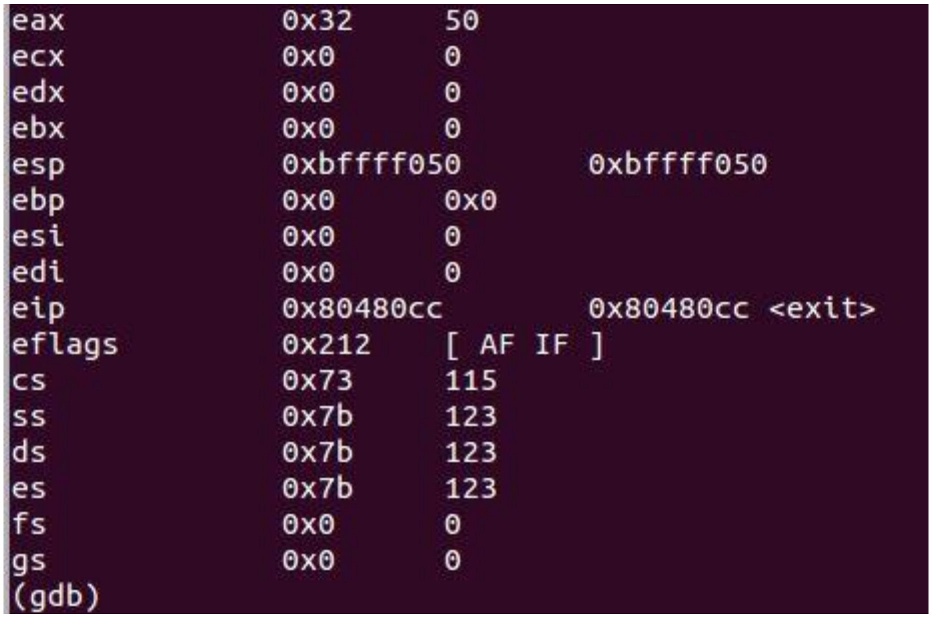
```
        jne loop1
loop2:
        movl %eax,array2( ,%edx,4)
        addl $1,%edx
        cmpl $5,%edx
        jne inc

        movl $1,%edx
        movl array2( ,%edx,4),%eax
        movl %eax,secondsmallnumber

movl $1,%eax
movl $0,%ebx
int $0*80
```

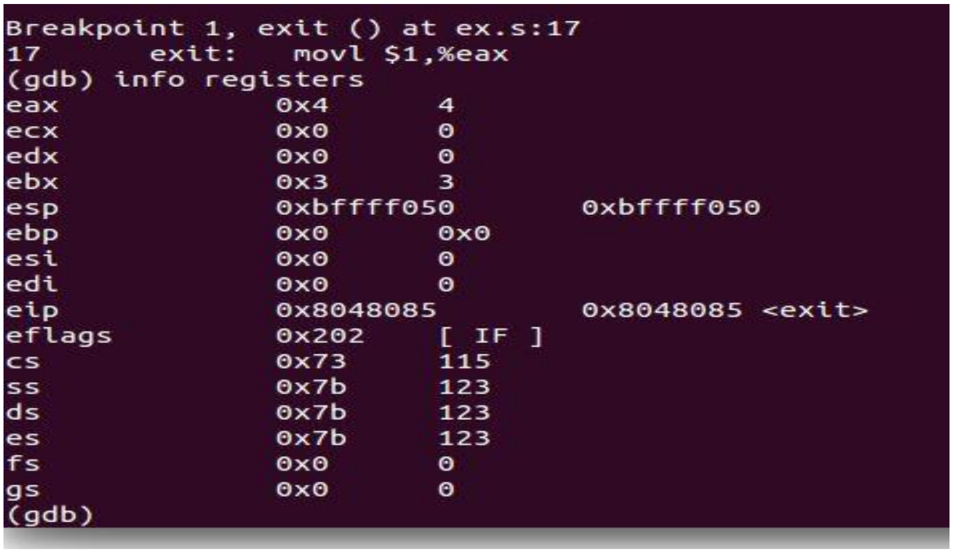
FIG 4 PROGRAM TO FIND SECOND SMALLEST NUMBER IN AN UNSORTED ARRAY.

5. Presentation of Results



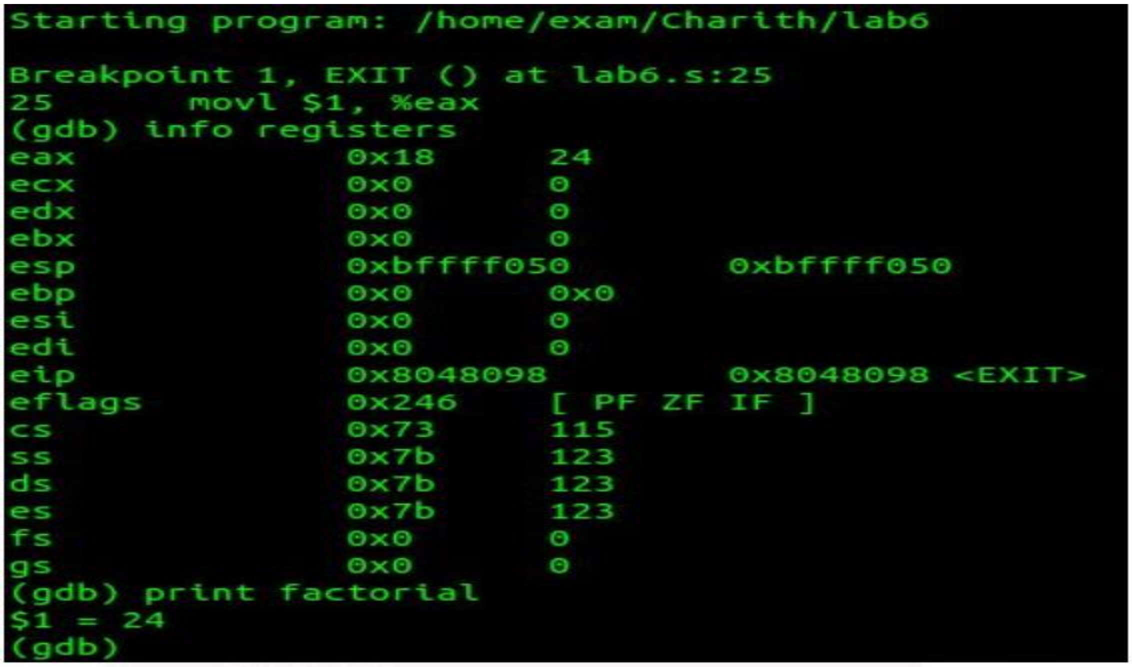
```
eax            0x32     50
ecx            0x0      0
edx            0x0      0
ebx            0x0      0
esp            0xbffff050       0xbffff050
ebp            0x0      0x0
esi            0x0      0
edi            0x0      0
eip            0x80480cc        0x80480cc <exit>
eflags         0x212    [ AF IF ]
cs             0x73     115
ss             0x7b     123
ds             0x7b     123
es             0x7b     123
fs             0x0      0
gs             0x0      0
(gdb)
```

FIG 5 RESULT OF PROGRAM TO PERFORM ADDITION OF TWO NUMBERS



```
Breakpoint 1, exit () at ex.s:17
17      exit:   movl $1,%eax
(gdb) info registers
eax            0x4      4
ecx            0x0      0
edx            0x0      0
ebx            0x3      3
esp            0xbffff050       0xbffff050
ebp            0x0      0x0
esi            0x0      0
edi            0x0      0
eip            0x8048085        0x8048085 <exit>
eflags         0x202    [ IF ]
cs             0x73     115
ss             0x7b     123
ds             0x7b     123
es             0x7b     123
fs             0x0      0
gs             0x0      0
(gdb)
```

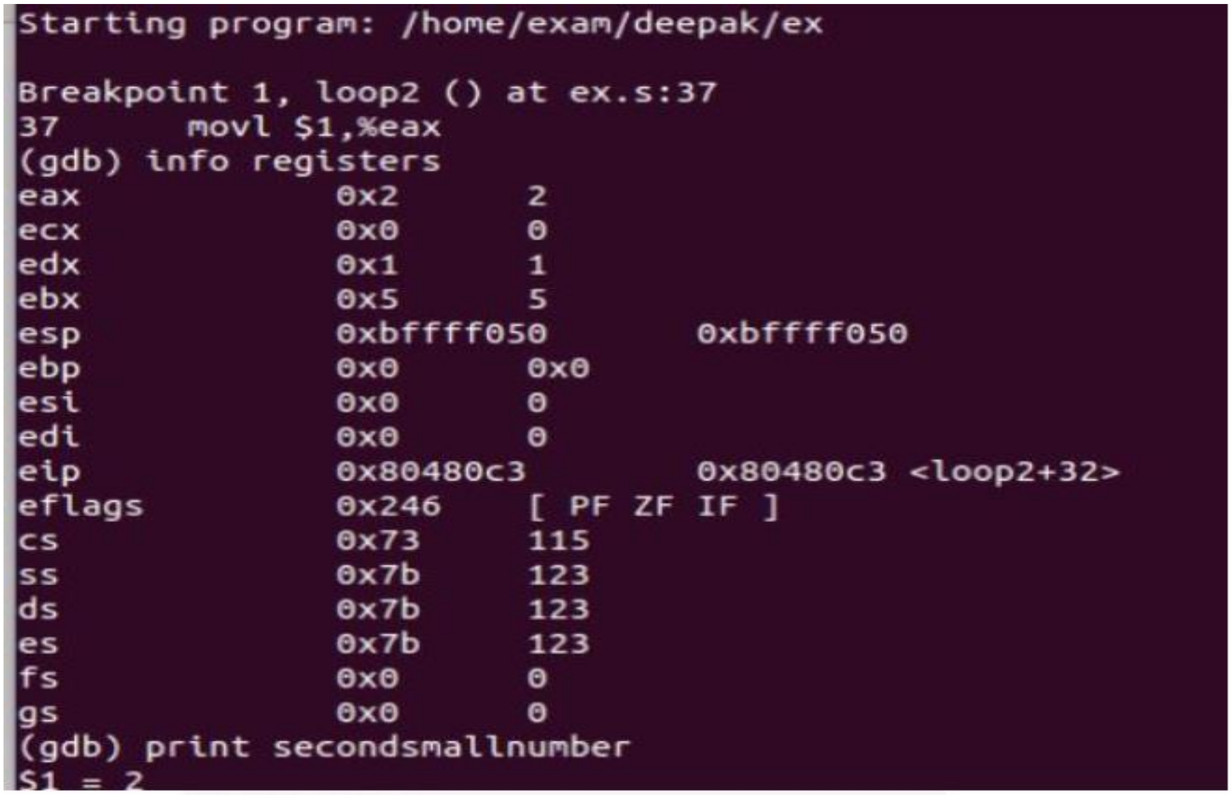
FIG 6 RESULT OF PROGRAM TO PERFORM SWAP OPERATION USING LOGICAL INSTRUCTIONS



```
Starting program: /home/exam/Charith/lab6

Breakpoint 1, EXIT () at lab6.s:25
25       movl $1, %eax
(gdb) info registers
eax            0x18     24
ecx            0x0      0
edx            0x0      0
ebx            0x0      0
esp            0xbffff050       0xbffff050
ebp            0x0      0x0
esi            0x0      0
edi            0x0      0
eip            0x8048098        0x8048098 <EXIT>
eflags         0x246    [ PF ZF IF ]
cs             0x73     115
ss             0x7b     123
ds             0x7b     123
es             0x7b     123
fs             0x0      0
gs             0x0      0
(gdb) print factorial
$1 = 24
(gdb)
```

FIG 7 RESULT OF PROGRAM TO COMPUTE FACTORIAL OF A NUMBER.



```
Starting program: /home/exam/deepak/ex

Breakpoint 1, loop2 () at ex.s:37
37       movl $1,%eax
(gdb) info registers
eax            0x2      2
ecx            0x0      0
edx            0x1      1
ebx            0x5      5
esp            0xbffff050       0xbffff050
ebp            0x0      0x0
esi            0x0      0
edi            0x0      0
eip            0x80480c3        0x80480c3 <loop2+32>
eflags         0x246    [ PF ZF IF ]
cs             0x73     115
ss             0x7b     123
ds             0x7b     123
es             0x7b     123
fs             0x0      0
gs             0x0      0
(gdb) print secondsmallnumber
$1 = 2
```

FIG 8 RESULT OF PROGRAM TO FIND SECOND SMALLEST NUMBER IN AN UNSORTED ARRAY.

1. ANALYSIS AND DISCUSSIONS

THE ARRAY HERE IS SORTED USING VARIOUS CONDITIONAL STATEMENTS BY COMBINATION OF COMPARE AND JUMP INSTRUCTION, TO SORT THEM WE ARE USING INSERTION SORT ALGORITHM, BUT A LITTLE UNOPTIMIZED VERSION OF IT, WE KEEP ON SWAPPING THE CURRENT ELEMENT WITH EVERY ELEMENT THAT IS SMALLER THAN IT, WE KEEP ON DOING THIS UNTIL WE REACH THE END OF THE ARRAY, THE TIME COMPLEXITY FOR SUCH A SORTING ALGORITHM IS $O(n^2)$.

2. CONCLUSIONS

ARRAYS CAN BE SORTED BY USING A COMBINATION OF JUMP, COMPARE AND LABELS. THESE HAVE TO BE CAREFULLY DESIGNED AS TO AVOID INFINITE LOOPS AND ARRAY INDEX OUT OF BOUNDS ERRORS, WHICH CAUSES SEGMENTATION FAULTS.

3. COMMENTS

1. LIMITATIONS OF EXPERIMENTS

COMPLEX SORTING ALGORITHMS SUCH AS TIM SORT, RADIX SORT, OPTIMIZED QUICK SORT, ARE VERY COMPLEX TO IMPLEMENT IN ASSEMBLY ALTHOUGH THEY WOULD PROVIDE PERFORMANCE BENEFITS.

2. LIMITATIONS OF RESULTS

THE CODE WRITTEN FOR SORTING THE ELEMENTS IS UNOPTIMIZED AND WOULD PERFORM WORSE WHEN THE DATA GIVEN TO IT IS IN HUGE AMOUNT.

3. LEARNING HAPPENED

ELEMENTS OF AN ARRAY CAN BE SORTED IN ASSEMBLY, THOUGH WITH QUITE A LOT OF WRITTEN CODE.

4. RECOMMENDATIONS

THE LOOP STATEMENTS SHOULD BE CAREFULLY WRITTEN TO AVOID INFINITE LOOPS.

SIGNATURE AND DATE

MARKS