Register

Laboratory 2

Title of the Laboratory Exercise: Arithmetic and Logical Operations

1. Introduction and Purpose of Experiment

   Students will be able to perform all arithmetic and logical operations using assembly instructions

2. Aim and

   Objectives Aim

   To develop assembly language program to perform all arithmetic operations.

   Objectives

   At the end of this lab, the student will be able to

   - Identify the appropriate assembly language instruction for the given arithmetic operations
   - Perform all arithmetic operations using assembly language instructions
   - Understand different data types and memory used
   - Get familiar with assembly language program by developing simple programs

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 a laboratory report documenting the work

4. Questions

1. Consider the following source code fragment

```
Int a,b,c,d;
a= (b + c)/d
e= (b*c)  -
d;
e=(((a*b)/c)*d)
```

- Assume that q,*b, c, d,e* are in registers. Develop an assembly language program to perform this assignment statements.

  Int a,b,c,d;

  a= (b + c)/d

  *e= (b*c) -*

  d;

  *e=(((a*b)/c)*d)*

- Assume that *b,e is* in registers and *c*, *d,a* in memory. Develop an assembly language program to perform this assignment statements.

  Int a,b,c,d;

  a= (b AND c) XOR d;

  a=(b XOR c) OR d;

  Assume that a,*b, c, d* are in registers. Develop an assembly language program to perform this assignment statements.

  Int a,b,c,d;

  a= (b AND c) XOR d;

  a=(b XOR c) OR d;

  Assume that *b, c* are in registers and a,*d* in memory. Develop an assembly language program to perform this assignment statements.

5. Calculations/Computations/Algorithms

6. Presentation of Results

THE CODE

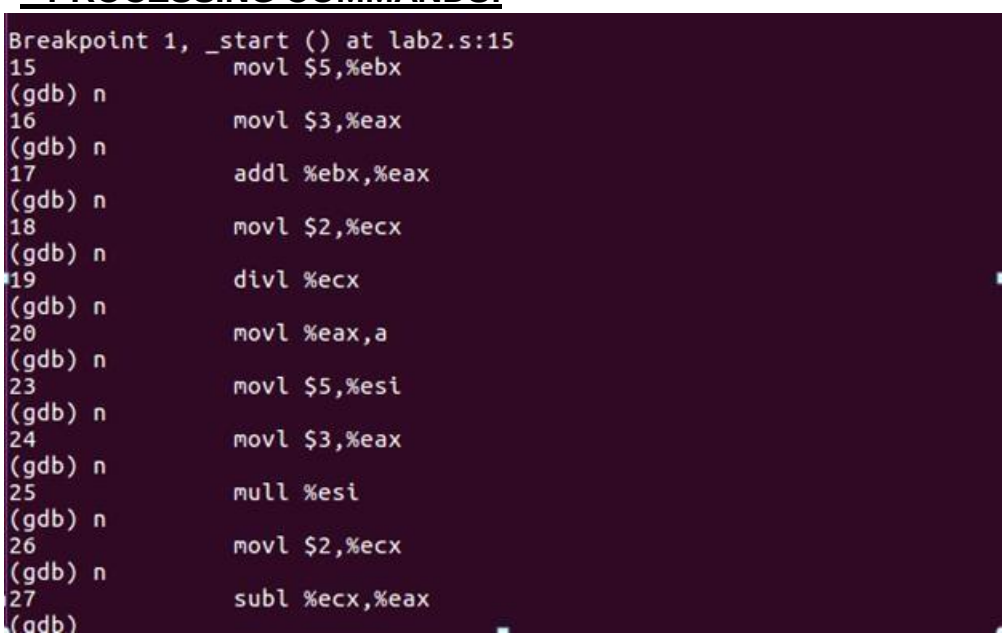
```
.section .data
        c:
                .int 3
        d:
                .int 2
        a:
                .int 4
        e:
                .int 0
.section .text
.globl _start
_start:
        /* a=4 b=5 c=3 d=2 */
        /* a=(b+c)/d */
        movl $5,%ebx
        movl $3,%eax
        addl %ebx,%eax
        movl $2,%ecx
        divl %ecx
        movl %eax,a
```

```
/* e=(b*c)-d */
movl $5,%esi
movl $3,%eax
mull %esi
movl $2,%ecx
subl %ecx,%eax
movl %eax,e

/* e=(((a*b)/c)*d) */
movl $4,%eax
movl $5,%edx
mull %edx
movl $3,%ecx
divl %ecx
movl $2,%ecx
mull %ecx
movl %eax,e

/* b,e are in registers . c,d, a are in memory /
/* a=(b+c)/d */
/* a=4 b=5 c=3 d=2 */
movl $5,%eax
addl c,%eax
movl d,%ecx
divl %ecx
movl %eax,a

/* e=(b*c) -d */
movl $5,%eax
movl c,%edi
mull %edi
movl d,%ebx
subl %ebx,%eax
movl %eax,e

/* e=(((a*b)/c)*d) */
movl a,%eax
movl $5,%ebx
mull %ebx
movl c,%ecx
divl %ecx
movl d,%edi
mull %edi
movl %eax,e

/* boolean functions */
/* a=(b AND c) XOR d */
/* b=5 c=15 d=20 */
movl $5,%eax
movl $15,%ebx
movl $20,%ecx
andl %eax,%ebx
xorl %ebx,%ecx
movl %ecx,a
```

```
/* boolean functions */
/* a=(b XOR c) OR d */
/* b=5 c=15 d=20 */
movl $5,%eax
movl $15,%ebx
movl $20, %ecx
xorl %eax,%ebx
orl %ebx,%ecx
movl %ecx, a

/* boolean functions */
/* a=(b AND c) XOR d */
/* b,c are in registers . a,d are in memory (a=4 ,d=2 ,b=20 ,c=43 )*/
movl $20,%edx
movl $43,%ebx
andl %edx,%ebx
movl d,%eax
xorl %ebx,%eax
movl %eax, a

/* boolean functions */
/* a=(b XOR c) OR d */
/* b,c are in registers . a,d are in memory (a=4 ,d=2 )*/
movl $6,%eax
movl $3,%ebx
xorl %eax,%ebx
movl d,%edi
orl %ebx,%edi
movl %edi, a

movl $1,%eax
movl $0,%ebx
int $0X80
```

Register

PROCESSING COMMANDS:



Register

```
(gdb) n
28              movl %eax,e
(gdb) n
31              movl $4,%eax
(gdb) n
32              movl $5,%edx
```

```
(gdb) n
33              mull %edx
(gdb) n
34              movl $3,%ecx
(gdb) n
35              divl %ecx
(gdb) n
36              movl $2,%ecx
(gdb) n
37              mull %ecx
(gdb) n
38              movl %eax,e
(gdb) n
43              movl $5,%eax
(gdb) n
44              addl c,%eax
(gdb) n
45              movl d,%ecx
(gdb) n
46              divl %ecx
(gdb) n
47              movl %eax,a
(gdb) n
50              movl $5,%eax
(gdb) n
51              movl c,%edi
(gdb) n
52              mull %edi
(gdb) n
53              movl d,%ebx
(gdb) n
54              subl %ebx,%eax
```

```
55              movl %eax,e
(gdb) n
58              movl a,%eax
(gdb) n
59              movl $5,%ebx
(gdb) n
60              mull %ebx
(gdb) n
61              movl c,%ecx
(gdb) n
62              divl %ecx
```

```
(gdb) n
63              movl d,%edi
(gdb) n
64              mull %edi
(gdb) n
65              movl %eax,e
(gdb) n
70              movl $5,%eax
(gdb) n
71              movl $15,%ebx
(gdb) n
72              movl $20,%ecx
(gdb) n
73              andl %eax,%ebx
(gdb) n
74              xorl %ebx,%ecx
(gdb) n
75              movl %ecx,a
(gdb) n
80              movl $5,%eax
(gdb) n
81              movl $15,%ebx
(gdb) n
82              movl $20, %ecx
(gdb) n
83              xorl %eax,%ebx
(gdb) n
84              orl %ebx,%ecx
(gdb) n
85              movl %ecx, a
(gdb) n
90              movl $20,%edx
(gdb)
```

```
90              movl $20,%edx
(gdb) n
91              movl $43,%ebx
(gdb) n
92              andl %edx,%ebx
(gdb) n
93              movl d,%eax
(gdb) n
94              xorl %ebx,%eax
(gdb) n
95              movl %eax, a
(gdb) n
100             movl $6,%eax
(gdb) n
101             movl $3,%ebx
(gdb) n
102             xorl %eax,%ebx
(gdb) n
103             movl d,%edi
(gdb) n
104             orl %ebx,%edi
(gdb) n
105             movl %edi, a
(gdb) n
107             movl $1,%eax
(gdb) n
108             movl $0,%ebx
(gdb) n
109             int $0X80
(gdb) n
[Inferior 1 (process 3579) exited normally]
(gdb)
```

REGISTER BEFORE THE OPERATION A=(B+C)/D

Register

```
Reading symbols from lab2...done.
(gdb) break lab2.s:11
Breakpoint 1 at 0x8048074: file lab2.s, line 11.
(gdb) run
Starting program: /home/ubuntu/amith147/lab2

Breakpoint 1, _start () at lab2.s:15
15              movl $5,%ebx
(gdb) info register
eax            0x0      0
ecx            0x0      0
edx            0x0      0
ebx            0x0      0
esp            0xbffff080       0xbffff080
ebp            0x0      0x0
esi            0x0      0
edi            0x0      0
eip            0x8048074        0x8048074 <_start>
eflags         0x202    [ IF ]
cs             0x73     115
ss             0x7b     123
ds             0x7b     123
es             0x7b     123
fs             0x0      0
gs             0x0      0
```

REGISTER AFTER A=(B+C)/D OPERATION:

```
(gdb) n
16              movl $3,%eax
(gdb) n
17              addl %ebx,%eax
(gdb) n
18              movl $2,%ecx
(gdb) n
19              divl %ecx
(gdb) n
20              movl %eax,a
(gdb) n
23              movl $5,%esi
(gdb) print a
$1 = 4
(gdb) info register
eax            0x4      4
ecx            0x2      2
edx            0x0      0
ebx            0x5      5
esp            0xbffff080       0xbffff080
ebp            0x0      0x0
esi            0x0      0
edi            0x0      0
eip            0x804808c        0x804808c <_start+24>
eflags         0x202    [ IF ]
cs             0x73     115
ss             0x7b     123
ds             0x7b     123
es             0x7b     123
fs             0x0      0
gs             0x0      0
```

REGISTER AFTER E=(B*C)-D:

Register

```
(gdb) n
24              movl $3,%eax
(gdb) n
25              mull %esi
(gdb) n
26              movl $2,%ecx
(gdb) n
27              subl %ecx,%eax
(gdb) n
28              movl %eax,e
(gdb) n
31              movl $4,%eax
(gdb) print e
$2 = 13
(gdb) info registers
eax            0xd      13
ecx            0x2      2
edx            0x0      0
ebx            0x5      5
esp            0xbffff080       0xbffff080
ebp            0x0      0x0
esi            0x5      5
edi            0x0      0
eip            0x80480a4        0x80480a4 <_start+48>
eflags         0x202    [ IF ]
cs             0x73     115
ss             0x7b     123
ds             0x7b     123
es             0x7b     123
fs             0x0      0
gs             0x0      0
```

REGISTER AFTER E=(((A*B)/C)*D)

```
33              mull %edx
(gdb) n
34              movl $3,%ecx
(gdb) n
35              divl %ecx
(gdb) n
36              movl $2,%ecx
(gdb) n
37              mull %ecx
(gdb) n
38              movl %eax,e
(gdb) n
43              movl $5,%eax
(gdb) print e
$3 = 12
(gdb) info registers
eax            0xc      12
ecx            0x2      2
edx            0x0      0
ebx            0x5      5
esp            0xbffff080       0xbffff080
ebp            0x0      0x0
esi            0x5      5
edi            0x0      0
eip            0x80480c3        0x80480c3 <_start+79>
eflags         0x206    [ PF IF ]
cs             0x73     115
ss             0x7b     123
ds             0x7b     123
es             0x7b     123
fs             0x0      0
gs             0x0      0
```

REPEATING THE ABOVE 3 OPERATIONS BY STORING DATA IN MEMORY AND REGISTERS

REGISTER AFTER A=(B+C)/D OPERATION:

Register

```
44              addl c,%eax
(gdb) n
45              movl d,%ecx
(gdb) n
46              divl %ecx
(gdb) n
47              movl %eax,a
(gdb) n
50              movl $5,%eax
(gdb) print a
$4 = 4
(gdb) info register
eax            0x4      4
ecx            0x2      2
edx            0x0      0
ebx            0x5      5
esp            0xbffff080       0xbffff080
ebp            0x0      0x0
esi            0x5      5
edi            0x0      0
eip            0x80480db        0x80480db <_start+103>
eflags         0x202    [ IF ]
cs             0x73     115
ss             0x7b     123
ds             0x7b     123
es             0x7b     123
fs             0x0      0
gs             0x0      0
```

REGISTER AFTER E=(B*C)-D:

```
(gdb) n
51              movl c,%edi
(gdb) n
52              mull %edi
(gdb) n
53              movl d,%ebx
(gdb) n
54              subl %ebx,%eax
(gdb) n
55              movl %eax,e
(gdb) n
58              movl a,%eax
(gdb) print e
$5 = 13
(gdb) info register
eax            0xd      13
ecx            0x2      2
edx            0x0      0
ebx            0x2      2
esp            0xbffff080       0xbffff080
ebp            0x0      0x0
esi            0x5      5
edi            0x3      3
eip            0x80480f5        0x80480f5 <_start+129>
eflags         0x202    [ IF ]
cs             0x73     115
ss             0x7b     123
ds             0x7b     123
es             0x7b     123
fs             0x0      0
gs             0x0      0
```

REGISTER AFTER E=(((A*B)/C)*D)

```
59              movl $5,%ebx
(gdb) n
60              mull %ebx
(gdb) n
61              movl c,%ecx
(gdb) n
62              divl %ecx
(gdb) n
63              movl d,%edi
(gdb) n
64              mull %edi
(gdb) n
65              movl %eax,e
(gdb) n
70              movl $5,%eax
(gdb) print e
$6 = 12
(gdb) info register
eax            0xc      12
ecx            0x3      3
edx            0x0      0
ebx            0x5      5
esp            0xbffff080       0xbffff080
ebp            0x0      0x0
esi            0x5      5
edi            0x2      2
eip            0x8048116        0x8048116 <_start+162>
eflags         0x206    [ PF IF ]
cs             0x73     115
ss             0x7b     123
ds             0x7b     123
es             0x7b     123
fs             0x0      0
```

REGISTER AFTER BOOLEAN FUNCTION A=(B AND C) XOR D

```
(gdb) n
71              movl $15,%ebx
(gdb) n
72              movl $20,%ecx
(gdb) n
73              andl %eax,%ebx
(gdb) n
74              xorl %ebx,%ecx
(gdb) n
75              movl %ecx,a
(gdb) n
80              movl $5,%eax
(gdb) print a
$7 = 17
(gdb) info register
eax            0x5      5
ecx            0x11     17
edx            0x0      0
ebx            0x5      5
esp            0xbffff080       0xbffff080
ebp            0x0      0x0
esi            0x5      5
edi            0x2      2
eip            0x804812f        0x804812f <_start+187>
eflags         0x206    [ PF IF ]
cs             0x73     115
ss             0x7b     123
ds             0x7b     123
es             0x7b     123
fs             0x0      0
gs             0x0      0
```

REGISTER AFTER BOOLEAN FUNCTIN A=(B XOR C) OR D

```
(gdb) n
81              movl $15,%ebx
(gdb) n
82              movl $20, %ecx
(gdb) n
83              xorl %eax,%ebx
(gdb) n
84              orl %ebx,%ecx
(gdb) n
85              movl %ecx, a
(gdb) n
90              movl $20,%edx
(gdb) print a
$8 = 30
(gdb) info register
eax            0x5      5
ecx            0x1e     30
edx            0x0      0
ebx            0xa      10
esp            0xbffff080       0xbffff080
ebp            0x0      0x0
esi            0x5      5
edi            0x2      2
eip            0x8048148        0x8048148 <_start+212>
eflags         0x206    [ PF IF ]
cs             0x73     115
ss             0x7b     123
ds             0x7b     123
es             0x7b     123
fs             0x0      0
gs             0x0      0
```

REPEATING THE ABOVE 2 BINARY FUNCTIONS BY STORING DATA IN MEMORY AND REGISTERS

REGISTER AFTER BOOLEAN FUNCTION A=(B AND C) XOR D

Register

```
(gdb) n
91              movl $43,%ebx
(gdb) n
92              andl %edx,%ebx
(gdb) n
93              movl d,%eax
(gdb) n
94              xorl %ebx,%eax
(gdb) n
95              movl %eax, a
(gdb) n
100             movl $6,%eax
(gdb) print a
$9 = 2
(gdb) info register
eax            0x2        2
ecx            0x1e       30
edx            0x14       20
ebx            0x0        0
esp            0xbffff080         0xbffff080
ebp            0x0        0x0
esi            0x5        5
edi            0x2        2
eip            0x8048160          0x8048160 <_start+236>
eflags         0x202      [ IF ]
cs             0x73       115
ss             0x7b       123
ds             0x7b       123
es             0x7b       123
fs             0x0        0
gs             0x0        0
```

REGISTER AFTER BOOLEAN FUNCTIN A=(B XOR C) OR D

```
(gdb) n
101             movl $3,%ebx
(gdb) n
102             xorl %eax,%ebx
(gdb) n
103             movl d,%edi
(gdb) n
104             orl %ebx,%edi
(gdb) n
105             movl %edi, a
(gdb) n
107             movl $1,%eax
(gdb) print a
$10 = 7
(gdb) info register
eax            0x6        6
ecx            0x1e       30
edx            0x14       20
ebx            0x5        5
esp            0xbffff080         0xbffff080
ebp            0x0        0x0
esi            0x5        5
edi            0x7        7
eip            0x804817a          0x804817a <_start+262>
eflags         0x202      [ IF ]
cs             0x73       115
ss             0x7b       123
ds             0x7b       123
es             0x7b       123
fs             0x0        0
gs             0x0        0
```

ENDING AFTER COMPLETION OF ALL OPERATIONS

Register

```
(gdb) n
108             movl $0,%ebx
(gdb) n
109             int $0X80
(gdb) n
[Inferior 1 (process 3405) exited normally]
(gdb)
```

7. ANALYSIS AND DISCUSSIONS

8. CONCLUSIONS

9. COMMENTS

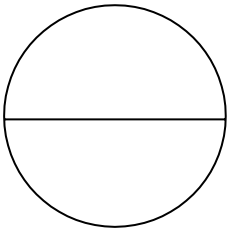
1. Limitations of the experiment
2. Limitations of the result

3. Learning happened

4. Recommendations

Signature and date

Marks



Register

Laboratory 3

Title of the Laboratory Exercise: Controlling execution flow using conditional instructions

1. Introduction and Purpose of Experiment

   Students will be able to perform control flow operations using conditional instructions

2. Aim and

   Objectives Aim

   To develop assembly language program to perform control flow operations using conditional instructions.

   Objectives

   At the end of this lab, the student will be able to

   - Identify the appropriate assembly language instruction for the given conditional operations
   - Perform all conditional operations using assembly language instructions
   - Get familiar with assembly language program by developing simple programs

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 a laboratory report documenting the work

4. Questions

   Develop an assembly language program to perform the following

   1. Create an array of 10 elements and initialized the array elements with random values

      a. Assign array[5]=0 and array[8]=0 (array is the name of the array, you can create array with any name)

      b. Perform the computation: (value is a memory variable)

      Value=array[3]+
      array[6]
      Array[7]=value+array[7
      ]

c. Compare array[5] with array[8] , show whether both the values are equal or not.

d. Read the 7th element of the array and check whether it's odd or even number

2. Given: sides of triangle

Find out the type of triangle: isosceles, scalene, equilateral triangle

5. Calculations/Computations/Algorithms

6. Presentation of Results

3.a) CODE :

```
.section .data
        value:
                .int 2
        a:
                .int 10,20,30,40,60,70,80,90,95,100
        side1:
                .int 32
        side2:
                .int 40
        side3:
                .int 62
.section .text
.globl _start
_start:
        /replacing the value of elements at index 5 and 8/
        movl $5,%eax
        movl $8,%ebx
        movl $0,a( ,%eax,4)
        movl $0,a( ,%ebx,4)

        /performing value=array(3)+array(6)/
        movl $3,%ebx
        movl $6,%ecx
```

```
        /performing value=array(3)+array(6)/
        movl $3,%ebx
        movl $6,%ecx
        movl a( ,%ebx,4),%ebx
        movl a( ,%ecx,4),%eax
        addl %ebx,%eax
        movl %eax,value

        /array(7)=value+array(7)/
        movl $7,%edi
        movl a( ,%edi,4),%edx
        addl value,%edx
        movl %edx,a( ,%edi,4)

        /comapring array[5] and array[8]/
        movl $5,%ecx
        movl $8,%eax
        movl a( ,%ecx,4),%edi
        movl a( ,%eax,4),%esi
        cmp %edi,%esi
        je theyareequal
        jne theyarenotequal
theyareequal:
        movl $3,%ebx
theyarenotequal:
        movl $4,%ebx
        /to find the number is even or odd/
        movl $0,%edx
        movl $0,%eax
        movl $6,%esi
        movl a( ,%esi,4),%eax
        movl $2,%ecx
        movl $0,%ebx
        divl %ecx
        cmp %edx,%ebx
        je eeven
        jne oddd
eeven:
        movl $222,%eax
oddd:
        movl $111,%eax

        /to find the type of triangle/
        /case1 when it is a equilateral triangle/
        movl side1,%edx
```

```
        jne oddd
eeven:
        movl $222,%eax
oddd:
        movl $111,%eax

        /to find the type of triangle/
        /case1 when it is a equilateral triangle/
        movl side1,%edx
        movl side2,%edi
        movl side3,%esi
        cmp %edx,%edi
        je equal
        jne notequal
equal:
        cmp %edi,%esi
        je equal1
        jne notequal1
equal1:
        movl $111,%eax
notequal1:
        movl $121,%eax
```

```
        jne notequal1
equal1:
        movl $111,%eax
notequal1:
        movl $121,%eax
notequal:
        cmp %edi,%esi
        je equal2
        jne notequal2
equal2:
        movl $121,%eax
notequal2:
        cmp %edx,%esi
        je equal3
        jne notequal3
equal3:
        movl $121,%eax
notequal3:
        movl $123,%eax

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

3b. : even odd

The program of even and odd

Register

```
theyareequal:
        movl $3,%ebx
theyarenotequal:
        movl $4,%ebx
        /to find the number is even or odd/
        movl $0,%edx
        movl $0,%eax
        movl $6,%esi
        movl a( ,%esi,4),%eax
        movl $2,%ecx
        movl $0,%ebx
        divl %ecx
        cmp %edx,%ebx
        je eeven
        jne oddd
eeven:
        movl $222,%eax
oddd:
        movl $111,%eax

        /to find the type of triangle/
        /case1 when it is a equilateral triangle/
        movl side1,%edx
```

OUTPUTS:

Output:
1,2,3&
4

```
Reading symbols from l3...done.
(gdb) break l3.s:11
Breakpoint 1 at 0x8048074: file l3.s, line 11.
(gdb) run
Starting program: /home/ubuntu/amith147/l3

Breakpoint 1, _start () at l3.s:16
16              movl $5,%eax
(gdb) n
17              movl $8,%ebx
(gdb) n
18              movl $0,a( ,%eax,4)
(gdb) n
19              movl $0,a( ,%ebx,4)
(gdb) n
22              movl $3,%ebx
(gdb) n
```

Register

```
23              movl $6,%ecx
(gdb) n
24              movl a( ,%ebx,4),%ebx
(gdb) n
25              movl a( ,%ecx,4),%eax
(gdb) n
26              addl %ebx,%eax
(gdb) n
27              movl %eax,value
(gdb) n
30              movl $7,%edi
(gdb) print value
$1 = 120
(gdb) n
31              movl a( ,%edi,4),%edx
(gdb) n
32              addl value,%edx
(gdb) n
33              movl %edx,a( ,%edi,4)
(gdb) n
36              movl $5,%ecx
(gdb) print $edi
$2 = 7
(gdb) print $edx
```

```
37              movl $8,%eax
(gdb) n
38              movl a( ,%ecx,4),%edi
(gdb) n
39              movl a( ,%eax,4),%esi
(gdb) n
40              cmp %edi,%esi
(gdb) n
41              je theyareequal
(gdb) n
44              movl $3,%ebx
(gdb) n
46              movl $4,%ebx
(gdb) n
48              movl $0,%edx
(gdb) n
49              movl $0,%eax
(gdb) n
50              movl $6,%esi
(gdb) n
51              movl a( ,%esi,4),%eax
(gdb) n
52              movl $2,%ecx
(gdb) n
```

v

Register

OUT PUT TO FIND THE TYPE OF TRIANGLE:
If the value in eax register is :

111 : it means the triangle is equilateral triangle 121 : it means the triangle is isosceles triangle 123 : it means the triangle is skeletal triangle

```
55              cmp %edx,%ebx
(gdb) n
56              je eeven
(gdb) n
59              movl $222,%eax
(gdb) n
61              movl $111,%eax
(gdb) n
65              movl side1,%edx
(gdb) info register
eax            0x6f     111
ecx            0x2      2
edx            0x0      0
ebx            0x0      0
esp            0xbffff020       0xbffff020
ebp            0x0      0x0
esi            0x6      6
edi            0x0      0
eip            0x8048126        0x8048126 <oddd+5>
eflags         0x246    [ PF ZF IF ]
cs             0x73     115
ss             0x7b     123
ds             0x7b     123
es             0x7b     123
```

```
es             0x7b     123
fs             0x0      0
gs             0x0      0
(gdb) n
66              movl side2,%edi
(gdb) print a@10
$4 = {10, 20, 30, 40, 60, 0, 80, 210, 0, 100}
(gdb) n
67              movl side3,%esi
(gdb) n
68              cmp %edx,%edi
(gdb) n
69              je equal
(gdb) n
```

Register

```
(gdb) n
70              jne notequal
(gdb) n
80              cmp %edi,%esi
(gdb) n
81              je equal2
(gdb) n
82              jne notequal2
(gdb) n
86              cmp %edx,%esi
(gdb) n
87              je equal3
(gdb) n
88              jne notequal3
(gdb) n
92              movl $123,%eax
(gdb) n
94              movl $0,%eax
(gdb) info register
eax            0x7b     123
ecx            0x2      2
edx            0x20     32
ebx            0x0      0
esp            0xbffff020       0xbffff020
ebp            0x0      0x0
esi            0x3e     62
edi            0x28     40
eip            0x8048169        0x8048169 <notequal3+5>
eflags         0x206    [ PF IF ]
cs             0x73     115
ss             0x7b     123
ds             0x7b     123
es             0x7b     123
fs             0x0      0
gs             0x0      0
(gdb) n
95              movl $1,%ebx
(gdb) n
96              int $0x80
(gdb) n
0x08048175 in ?? ()
(gdb) n
Cannot find bounds of current function
(gdb) q
```

1. Analysis and Discussions

2. Conclusions

Value in eax register is 123 there fore it is a scalene triangle

Register

3. Comments

1. Limitations of Experiments

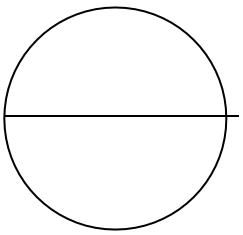
2. Limitations of Results

3. Learning happened

4. Recommendations

Signature and date

Marks



LABORATORY 4:

Title of the Laboratory Exercise: Controlling execution flow using conditional instructions

1. Introduction and Purpose of Experiment

   Students will be able to perform control flow operations using conditional instructions

2. Aim and Objectives

   Aim

   To develop assembly language program to perform control flow operations using conditional instructions.

   Objectives

   At the end of this lab, the student will be able to

   - Identify the appropriate assembly language instruction for the given conditional operations
   - Perform all conditional operations using assembly language instructions
   - Get familiar with assembly language program by developing simple programs

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 a laboratory report documenting the work

4. Questions

   Develop an assembly language program to perform the following

   1. Create an uninitialized array and print 'n' natural numbers in the created array.
   2. Develop an assembly language program to insert a given element in an array at the specified position.

      Input Array: 10, 20,30,40,50

      Element to be inserted: 60

Position: 3

Output: 10, 20, 30,60,40,50

3. Develop an assembly language program to generate the first n numbers in Fibonacci series.

5. Calculations/Computations/Algorithms

i)

```
// To print n natural numbers
.section .bss
        .lcomm array,40
.section .text
.globl _start:
_start:
        movl $0,%ecx
        movl $1,%edx
        loop:
             movl %edx,array( ,%ecx,4)
             addl $1,%ecx
             addl $1,%ebx
             cmpl $10,%ecx
             jne loop
        movl $1,%eax
        movl $0,%ebx
        int $0x80
```

ii)

```
//To insert an element at specific index in an array
.section .data
array:
      .int 10,20,30,40,50
.section .text
.globl _start:
_start:
       movl $5,%eax
       movl $4,%ebx
       loop:
            movl array( ,%ebx,4),%edx
            movl %edx,array( ,%eax,4)
            subl $1,%eax
            subl $1,%ebx
            cmpl $3,%eax
            jne loop
            movl $60,%ecx
            movl %ecx,array( ,%eax,4)
            movl $1,%eax
            movl $0,%ebx
            int $0x80
```

iii)

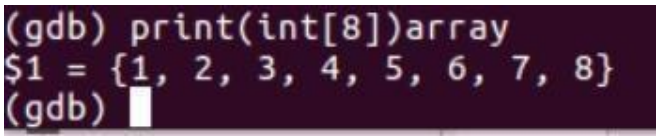
```
 1 .section .bss
 2         .lcomm array,40
 3 .section .text
 4 .globl _start
 5 _start:
 6         movl $0,%eax
 7         movl $1,%ebx
 8         movl $0,%ecx
 9         movl %eax,array( ,%ecx.4)
10         addl $1,%ecx
11         movl %ebx,array( ,%ecx,4)
12         loop:
13               addl %ebx,%eax
14               movl array( ,%ecx,4),%ebx
15               addl $1,%ecx
16               movl %eax,array( ,%ecx,4)
17               cmpl $10,%ecx
18               jne loop
19               je exit
20       exit:
21              movl $1,%eax
22              movl $0,%ebx
23              int  $0x80
```

5) Presentation of results

i)

```
Reading symbols from l6...done.
(gdb) break l6.s:6
Breakpoint 1 at 0x8048074: file l6.s, line 6.
(gdb) run
```

```
Breakpoint 1, _start () at l6.s:7
7               movl $0,%ecx
(gdb) n
8               movl $1,%edx
(gdb) n
10                   movl %edx,array( ,%ecx,4)
(gdb) n
11                   addl $1,%ecx
(gdb) n
12                   addl $1,%edx
(gdb) n
13                   cmpl $10,%ecx
(gdb) n
14                   jne loop
(gdb) n
10                   movl %edx,array( ,%ecx,4)
(gdb) n
11                   addl $1,%ecx
(gdb)
```


(gdb) print(int[8])array
$1 = {1, 2, 3, 4, 5, 6, 7, 8}
(gdb)

ii)

```
Reading symbols from l62...done.
(gdb) break l62.s:6
Breakpoint 1 at 0x8048074: file l62.s, line 6.
(gdb) run
```

```
Breakpoint 1, _start () at l62.s:7
7               movl $5,%eax
(gdb) n
8               movl $4,%ebx
(gdb) n
10                  movl array( ,%ebx,4),%edx
(gdb) n
11                  movl %edx,array( ,%eax,4)
(gdb) n
12                  subl $1,%eax
(gdb) n
13                  subl $1,%ebx
(gdb) n
14                  cmpl $3,%eax
(gdb)
```

```
(gdb) print(int[5])array
$1 = {10, 20, 30, 60, 40}
(gdb) print(int[6])array
$2 = {10, 20, 30, 60, 40, 50}
(gdb)
```

iii)

```
Reading symbols from l65...done.
(gdb) break l65.s:4
Breakpoint 1 at 0x8048074: file l65.s, line 4.
(gdb) run
Breakpoint 1, _start () at l65.s:6
6               movl $0,%eax
(gdb) n
7               movl $1,%ebx
(gdb) n
8               movl $0,%ecx
(gdb) n
9               movl %eax, array( ,%ecx,4)
(gdb) n
10              addl $1,%ecx
(gdb) n
11              movl %ebx,array( ,%ecx,4)
(gdb) n
13                    addl %ebx,%eax
(gdb) n
14                    movl array( ,%ecx,4),%ebx
(gdb)
```

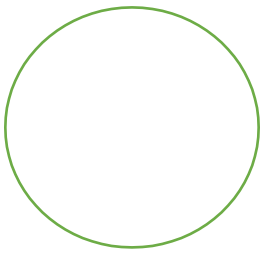
```
(gdb) print(int[10])array
$1 = {0, 1, 1, 2, 3, 5, 8, 13, 21, 34}
(gdb)
```

6) Conclusion

In this assembly laboratory program, we implemented basic mathematical programs using loops and array.

Signature and date

marks



Register

Laboratory 5

Title of the Laboratory Exercise: String manipulation

1. Introduction and Purpose of Experiment

   Students will be able to perform all string manipulations in assembly language

2. Aim and Objectives

   Aim

   To develop assembly language program to perform all string operations like inserting a byte, deleting a byte and copying a string as a sub-string

   Objectives

   At the end of this lab, the student will be able to

   - Identify instructions for performing string manipulation
   - Use indexed addressing mode
   - Apply looping instructions in assembly language
   - Use data segment to represent arrays

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 a laboratory report documenting the work

4. Questions

1. Develop an assembly language program to perform the following

   Copy the contents of STRING 1 to STRING 2

2. Develop an ALP to count the number of vowels in the given string

   Input: Hi Welcome Output: 4

3. Develop an assembly language program to remove first character of each word in the given string. Input: Hi Welcome All
   Output: i elcome ll

Register

4.Develop an assembly language program to compare two strings and print a message “Equal” if they are equal, “Not Equal” if they are not equal.

3. Calculations/Computations/Algorithms

5.1CODE:

```
.section .data
input_string:
        .asciz "A    "
.section .bss
        .lcomm output_string,15

.section .text
.globl _start
_start:
        movl $0,%ecx

loop:
        movb input_string(,%ecx,1),%al
        movb %al,output_string(,%ecx,1)
        addl $1,%ecx
        cmpl $10,%ecx
        Jne loop

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

Register

OUTPUT :

```
16              cmpl $10,%ecx
(gdb) n
17              Jne loop
(gdb) n
13              movb input_string(,%ecx,1),%al
(gdb) n
14              movb %al,output_string(,%ecx,1)
(gdb) n
15              addl $1,%ecx
(gdb) n
16              cmpl $10,%ecx
(gdb) n
17              Jne loop
(gdb) n
19              movl $1,%eax
(gdb) n
20              movl $0,%ebx
(gdb) n
21              int $0x80
```

```
(gdb) x/s &output_string
0x80490a8 <output_string>:      "A    "
(gdb)
```

5.2)CODE :

Code:

```
.section .data
input_string:
        .asciz "Hi Welcome"
vowels:
        .asciz "aeiou"
.section .bss
        .lcomm output_string,15

.section .text
.globl _start
_start:
        movl $0,%ecx
        movl $0,%edx
        movl $0,%edi

loop:
        movb input_string(,%ecx,1),%al
        movb vowels(,%edx,1),%bl
```

```
loop:
        movb input_string(,%ecx,1),%al
        movb vowels(,%edx,1),%bl
        cmpb %al,%bl
        jne increment_vowel_loop_index
        addl $1,%edi
        movl $0,%edx
        addl $1,%ecx
        cmpl $10,%ecx
        jne loop
        jmp Exit

increment_vowel_loop_index:
        addl $1,%edx
        cmpl $5,%edx
        Jne loop
        jmp Exit
```

```
increment_vowel_loop_index:
        addl $1,%edx
        cmpl $5,%edx
        Jne loop

increment_input_string_loop_index:
        movl $0,%edx
        addl $1,%ecx
        cmpl $10,%ecx
        jne loop
Exit:
        movl $1,%eax
        movl $0,%ebx
        int $0x80
```

Output:

```
Reading symbols from l5...done.
(gdb) break l5.s:2
Breakpoint 1 at 0x8048074: file l5.s, line 2.
(gdb) run
```

```
(gdb) x/s &vowels
0x80490d3:      "aeiou"
(gdb) info register
eax            0x1      1
ecx            0xa      10
edx            0x0      0
ebx            0x0      0
esp            0xbffff020       0xbffff020
ebp            0x0      0x0
esi            0x0      0
edi            0x4      4
eip            0x80480c6        0x80480c6 <Exit+10>
eflags         0x246    [ PF ZF IF ]
cs             0x73     115
ss             0x7b     123
ds             0x7b     123
es             0x7b     123
fs             0x0      0
gs             0x0      0
(gdb) n
[Inferior 1 (process 2992) exited normally]
(gdb) q
```

5.3)CODE :

```
.section .data
msg1:
        .ascii "HELLO !! PEOPLE !!"
.section .bss
.lcomm msg2,15
.section .text
.globl _start
_start:
        cld
        leal msg1+10, %esi
        leal msg2,%edi
        loop:
                movsb
                jmp loop
        jmp exit
exit:
        movl $1,%eax
        movl $0,%ebx
        int $0x80
```

OUTPUT :

```
Breakpoint 1, _start () at lab63.s:9
9               cld
(gdb) n
10              leal msg1+10, %esi
(gdb) n
11              leal msg2,%edi
(gdb) n
13                      movsb
(gdb) n
14                      jmp loop
(gdb) n
13                      movsb
(gdb) n
14                      jmp loop
```

```
(gdb) n
13                      movsb
(gdb) n
14                      jmp loop
(gdb) n
13                      movsb
(gdb) n
14                      jmp loop
(gdb) x/s &msg2
0x80490a8 <msg2>:       "EOPLE !!"
(gdb)
```

5.4)CODE :

```
.section .data
string1:
        .asciz "Hello"
Equal_Result:
        .asciz "Strings are Equal"
NE_Result:
        .asciz " Strings are Not Equal"
string2:
        .asciz "Hell"

.section .text
.globl _start
_start:
        movl $0,%ecx

loop:
        movb string1(,%ecx,1),%al
        movb string2(,%ecx,1),%bl
        cmpb %al,%bl
        je increment_loop_index
        Leal NE_Result,%esi
        jmp exit
```

Register

```
loop:
        movb string1(,%ecx,1),%al
        movb string2(,%ecx,1),%bl
        cmpb %al,%bl
        je increment_loop_index
        Leal NE_Result,%esi
        jmp exit

increment_loop_index:
        addl $1,%ecx
        cmpl $10,%ecx
        Jne loop

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

Register

OUTPUT :

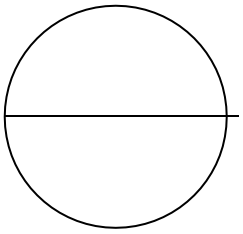
```
27              Jne loop
(gdb) n
17              movb string1(,%ecx,1),%al
(gdb) n
18              movb string2(,%ecx,1),%bl
(gdb) n
19              cmpb %al,%bl
(gdb) n
20              je increment_loop_index
(gdb) n
21              Leal NE_Result,%esi
(gdb) n
22              jmp exit
(gdb) n
31              movl $1,%eax
(gdb) n
32              movl $0,%ebx
(gdb) x/s $esi
0x80490c5:      " Strings are Not Equal"
(gdb) n
33              int $0x80
(gdb) n
[Inferior 1 (process 2907) exited normally]
(gdb)
```

4.Conclusion

In this assembly laboratory program , we implemented basic string operation programs.

Signature and date

marks



Laboratory 6

Title of the Laboratory Exercise: Searching an element in an array

1. Introduction and Purpose of Experiment

   Students will be able to perform search operations in an array of integers or characters

2. Aim and Objectives

   Aim

   To develop assembly language program to perform search operations in an array

   Objectives

   At the end of this lab, the student will be able to

   - Identify instructions to be used in assembly language
   - Perform search operations 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 a laboratory report documenting the work

4. Questions

   Develop an assembly language program to perform the following:

   1. Searching an element in an array of 'n' numbers
   2. Read a sentence with at least one special character and search for the special character and print it. E.g., consider the input {youremailid@msruas.ac.in }

      Output: @, .

5. Source Code

Part 1

```
.section .data
    array:
        .int 5, 6, 7, 8, 0, 1, 2, 3, 4, 9  # length 10

    search_element:
        .int 4

    found_at_index:
        .int -1



.section .text
.global _start
_start:
    # searching an element in an array of  'n'  numbers
    movl $0, %ecx       # index

search_loop:
    movl array(, %ecx, 4), %eax
    addl $1, %ecx   # index++

    cmpl search_element, %eax # if array[index] == search_element
    je update_found_index

    cmpl $10, %ecx      # exit if index > size
    jne search_loop
    jmp exit

update_found_index:
    subl $1, %ecx
    movl %ecx, found_at_index

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

Part 2

```
.section .data
    input:
        .ascii "21ETMC412011@msruas.ac.in" # length 25

.section .bss
```

```
    .lcomm output 25

.section .text
.global _start
_start:
    movl $0, %ecx
    movl $0, %ebx

loop:
    movb input(, %ecx, 1), %al
    jmp L1
continue_loop:
    addl $1, %ecx
    cmpl $25, %ecx
    jne loop
    jmp print

L1:
    cmpb $96, %al
    jle L2
    cmpb $122, %al
    jle continue_loop
L2:
    cmpb $64, %al
    jle L4
    cmpb $90, %al
    jle continue_loop
L4:
    cmpb $47, %al
    jle L5
    cmpb $57, %al
    jle continue_loop
L5:
    movb input(, %ecx, 1), %al
    movb %al, output(, %ebx, 1)
    addl $1, %ebx
    jmp continue_loop

print:
    movl $4, %eax
    movl $1, %ebx
    leal (output), %ecx
    movl $25, %edx
    int $0x80
```

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

6. Presentation of Results

```
Reading symbols from ./lab7a.elf...
(gdb) b 1
Breakpoint 1 at 0x8049000: file ./lab7a.s, line 17.
(gdb) b exit
Breakpoint 2 at 0x8049027: file ./lab7a.s, line 35.
(gdb) r
Starting program:

Breakpoint 1, _start () at ./lab7a.s:17
17          movl $0, %ecx       # index
(gdb) p (int[10])array
$1 = {5, 6, 7, 8, 0, 1, 2, 3, 4, 9}
(gdb) p (int)search_element
$2 = 4
(gdb) c
Continuing.

Breakpoint 2, exit () at ./lab7a.s:35
35          movl $1, %eax
(gdb) p (int)found_at_index
$3 = 8
(gdb) c
Continuing.
[Inferior 1 (process 3461) exited normally]
```

```
Reading symbols from ./lab7b.elf...
(gdb) b 1
Breakpoint 1 at 0x8049000: file ./lab7b.s, line 11.
(gdb) b exit
Breakpoint 2 at 0x804905f: file ./lab7b.s, line 52.
(gdb) r
Starting program:
```

```
11                  movl $0, %ecx
(gdb) p (char[25]) input
$1 = "21ETMC412011@msruas.ac.in"
(gdb) c
Continuing.
@..
```

```
50                  movl $1, %eax
(gdb) p (char[25]) output
$2 = "@..", '\000' <repeats 21 times>
(gdb) c
Continuing.
[Inferior 1 (process 3382) exited normally]
```

7. Conclusions

   The program executed successfully.

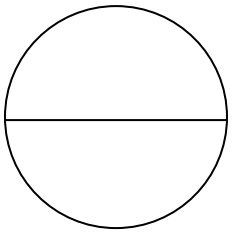
8. Comments

   1. Learning happened

   Learnt to work with strings and use complex algorithms

Signature and date

Marks



Register

Laboratory 7

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

   • Develop an assembly language program to sort all the elements in the given array (ascending order).

   •Develop an assembly language program to compute the second smallest number in the given array.

5. Calculations/Computations/Algorithms

Register

1.1) CODE :

```
.section .data
array:
        .int 4,2,1,6,8

b:
        .int 0
.section .bss
        .lcomm out,20


.section .text
.globl _start
_start:
        movl $0,%ecx
        movl $0,%edx
        movl $0,%esi
        movl $4,%edi

        loop1:
                movl array( ,%ecx,4),%eax
                loop2:
                        movl array( ,%edx,4),%ebx
                        addl $1,%edx
                        cmp %ebx,%eax
                        jl loop3
                        cmp $5,%edx
                        je loop6
                        jmp loop2
                loop3:
                        addl $1,%esi
                        cmp $5,%edx
                        je loop4
                        jmp loop2

                loop4:

                        subl %esi,%edi
                        movl %eax,out( ,%edi,4)
                        cmp $5,%ecx
                        je loop5
                        addl $1,%ecx
                        movl $0,%edx
                        movl $0,%esi
                        movl $4,%edi
                        jmp loop1
```

Register

OUTPUT :

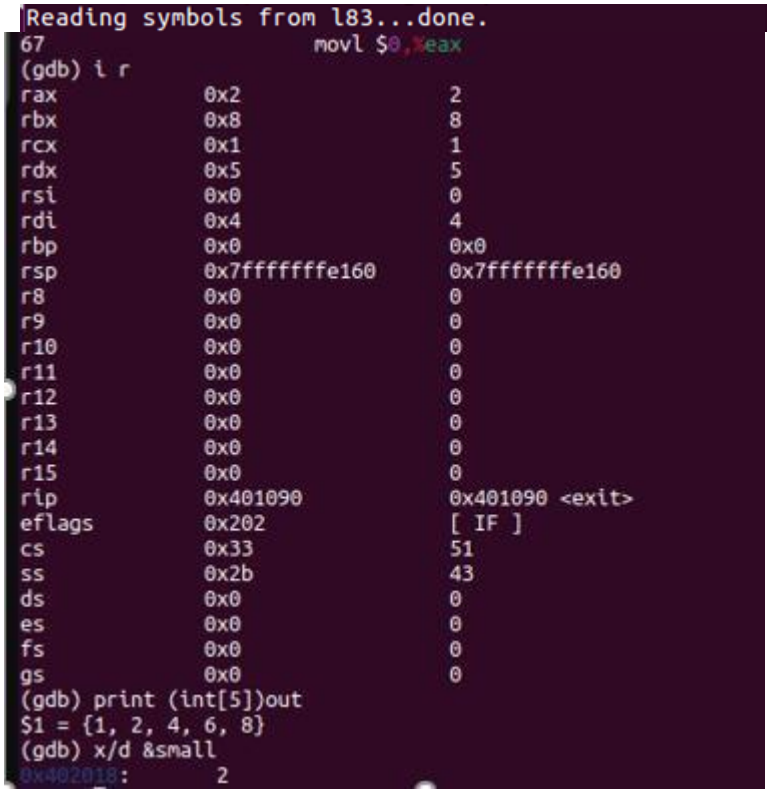
```
Reading symbols from l81...done.
(gdb) break l81.s:4
Breakpoint 1 at 0x8048074: file l81.s, line 4.
(gdb) run
Starting program: /home/ubuntu/amith147/l81
```

```
59                      movl $0,%eax
(gdb) i r
rax            0x8                 8
rbx            0x8                 8
rcx            0x4                 4
rdx            0x5                 5
rsi            0x0                 0
rdi            0x4                 4
rbp            0x0                 0x0
rsp            0x7fffffffe160      0x7fffffffe160
r8             0x0                 0
r9             0x0                 0
r10            0x0                 0
r11            0x0                 0
r12            0x0                 0
r13            0x0                 0
r14            0x0                 0
r15            0x0                 0
rip            0x401075            0x401075 <exit>
eflags         0x202               [ IF ]
cs             0x33                51
ss             0x2b                43
ds             0x0                 0
--Type <RET> for more, q to quit, c to continue without paging--c
es             0x0                 0
fs             0x0                 0
gs             0x0                 0
(gdb) print (int[5])out
$1 = {1, 2, 4, 6, 8}
```

1.2) CODE :

```
.section .data
array:
        .int 4,2,1,6,8

b:
        .int 0
small:
        .int 0
.section .bss
        .lcomm out,20


.section .text
.globl _start
_start:
        movl $0,%ecx
        movl $0,%edx
        movl $0,%esi
        movl $4,%edi

        loop1:
                movl array( ,%ecx,4),%eax
                loop2:
                        movl array( ,%edx,4),%ebx
                        addl $1,%edx
                        cmp %ebx,%eax

                        jl loop3
                        cmp $5,%edx
                        je loop6
                        jmp loop2
                loop3:
                        addl $1,%esi
                        cmp $5,%edx
                        je loop4
                        jmp loop2

                loop4:

                        subl %esi,%edi
                        movl %eax,out( ,%edi,4)
                        cmp $5,%ecx
                        je loop5
                        addl $1,%ecx
                        movl $0,%edx
                        movl $0,%esi
                        movl $4,%edi
                        jmp loop1

                loop5:
                        addl $1,%ecx
                        cmp $5,%ecx
                        je exit
                        jne loop1
                loop6:
                        subl %esi,%edi
                        movl %eax,out( ,%edi,4)
                        jmp loop7

                loop7:
                        movl $1,%ecx
                        movl $0,%eax
                        movl out( ,%ecx,4),%eax
                        movl %eax,small
                        jmp exit
        exit:
                movl $0,%eax
                movl $1,%ebx
                int $0x80
```

OUTPUT :



6. Conclusion

We learned to develop assembly program to arrange array in ascending and descending order.

Signature and date

Marks