Name: subhendu maji Reg No.: 18ETCS002121

Laboratory 5

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: @, .
   3. Develop an assembly language program to compute the parity of a hexadecimal number stored in the Register1. If Register1 has odd number of ones, update Register2 with 0x01. If Register1 has even number of ones, update Register2 with 0x00.

      Note: Register1 and Register2 can be any General Purpose Registers.

5. Calculations/Computations/Algorithms

5.1 Searching an element in an array of ‘n’ numbers

```
Step 1: start

Step 2: declare (array)list 2,5,3,6,10,15

Step 3: declare target (tar) 10

Step 4: move 0 to ecx

Step 5: move 0 to eax

Step 6: start a loop

     6.1 move ecx^th  value of list to eax
     6.2 add 1 to ecx
     6.3 compare target to eax
     6.4 jump if not equal

Step 7: the position of target is stored in ecx

Step 9: stop
```

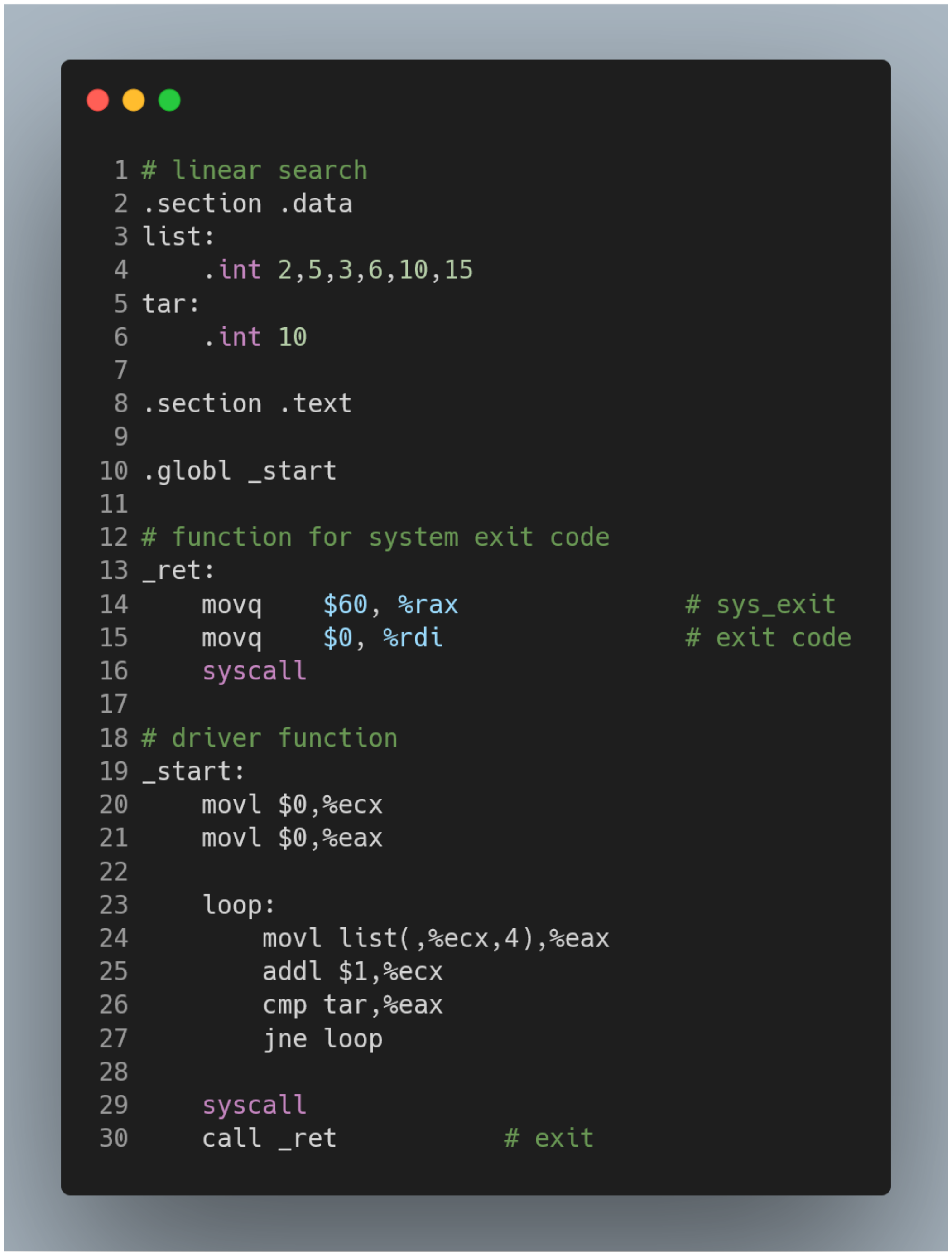
5.2 Read a sentence with at least one special character and search for the special character

```
Step 1: start

Step 2: declare (asciz) email = myemail@gmail.com

Step 3: declare target (tar) @

Step 4: move 0 to cl

Step 5: start a loop

     5.1 move ecx^th  value of email to al
     5.2 add 1 to cl
     5.3 compare target to al
     5.4 jump if not equal

Step 6: the position of target is stored in ecx
Step 7: stop
```

5.3 program to compute the parity of a hexadecimal number stored in the Register1. If Register1 has odd number of ones, update Register2 with 0x01. If Register1 has even number of ones, update Register2 with 0x00.

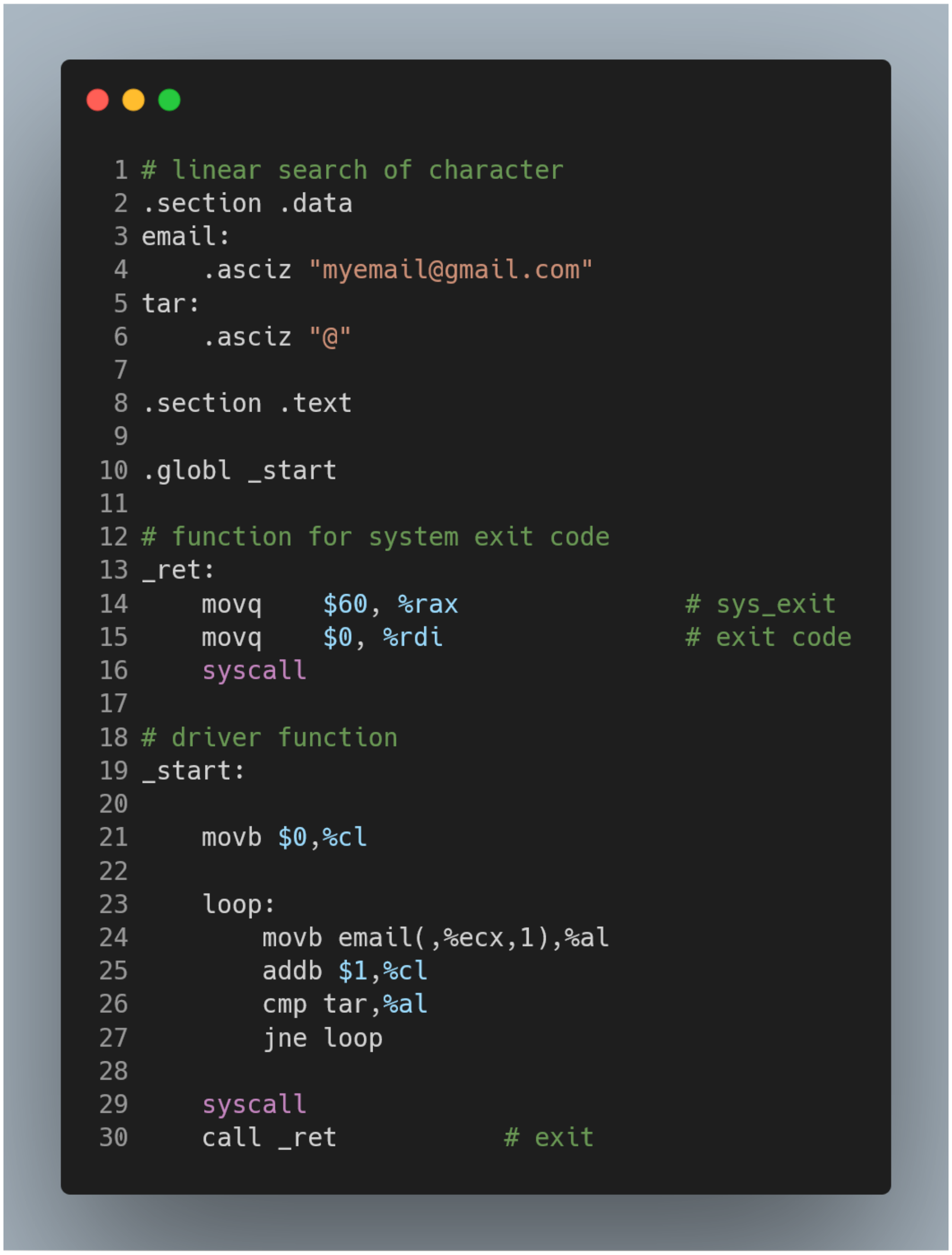
```
Step 1: start
Step 2: move 0x07(hexa-decimal 7) to ebx
Step 3: move 0 to ecx
Step 4: start a loop
        4.1 move 1 to eax
        4.2 perform and operation between ebx and eax
        4.3 add eax and ecx
        4.4 shift 1 bit right of eax
        4.5 compare 0 to ebx
        4.6 jump if not equal
Step 5: move ecx to eax
Step 6: perform and operation between 1 and eax
(0 & 1 = 0 / 1 & 1 = 1)
Step 7: stop
```

6. Presentation of Results



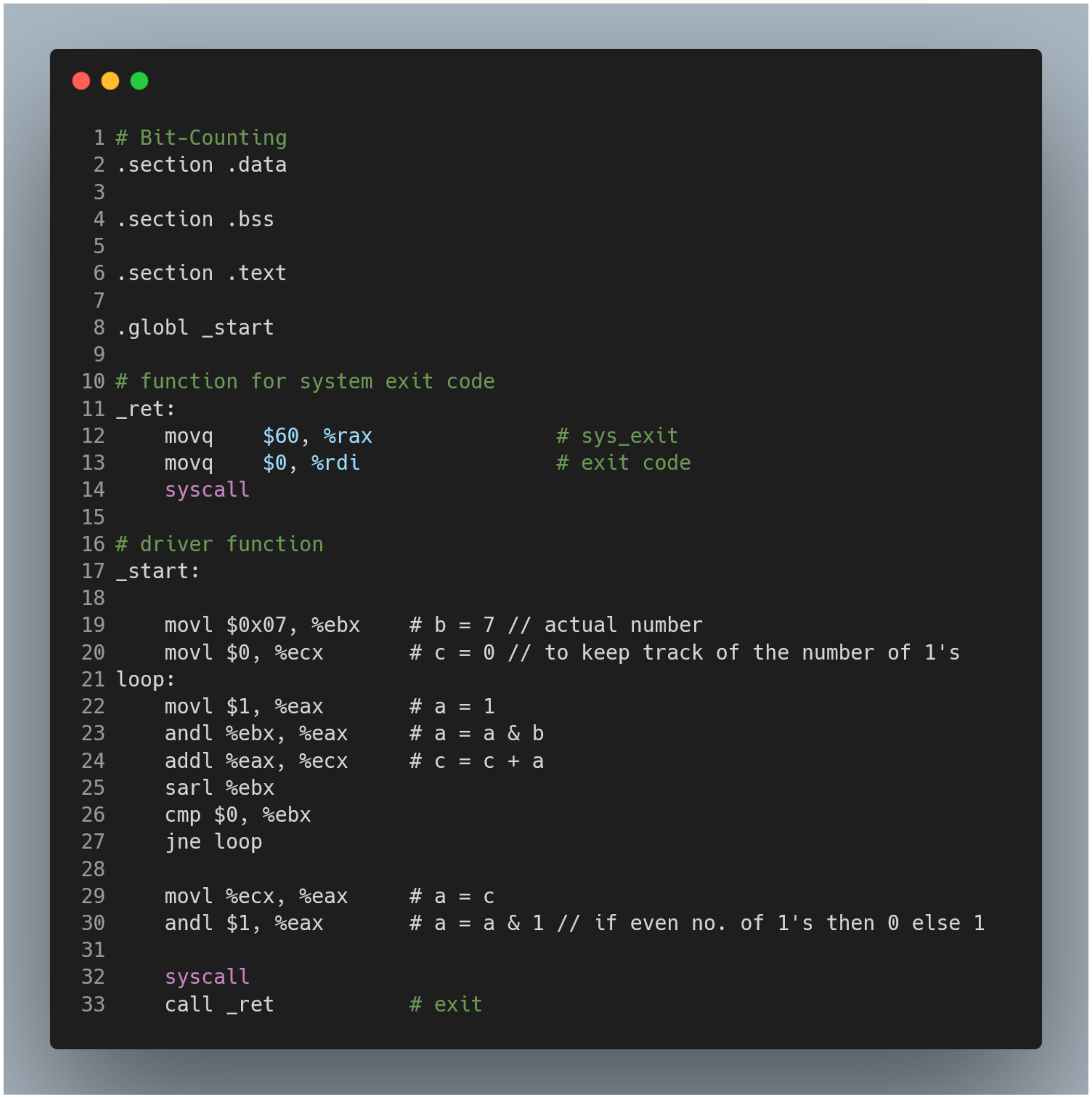
```
# linear search
.section .data
list:
    .int 2,5,3,6,10,15
tar:
    .int 10

.section .text

.globl _start

# function for system exit code
_ret:
    movq    $60, %rax               # sys_exit
    movq    $0, %rdi                # exit code
    syscall

# driver function
_start:
    movl $0,%ecx
    movl $0,%eax

    loop:
        movl list(,%ecx,4),%eax
        addl $1,%ecx
        cmp tar,%eax
        jne loop

    syscall
    call _ret           # exit
```

Figure 1 ASM code of Searching an element in an array of ‘n’ numbers



```
# linear search of character
.section .data
email:
    .asciz "myemail@gmail.com"
tar:
    .asciz "@"

.section .text

.globl _start

# function for system exit code
_ret:
    movq    $60, %rax               # sys_exit
    movq    $0, %rdi                # exit code
    syscall

# driver function
_start:

    movb $0,%cl

    loop:
        movb email(,%ecx,1),%al
        addb $1,%cl
        cmp tar,%al
        jne loop

    syscall
    call _ret           # exit
```

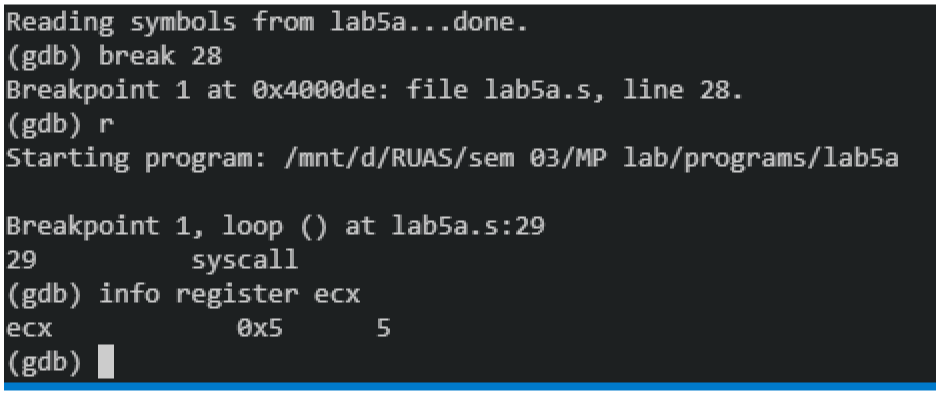
Figure 2 ASM code of program to Read a sentence with at least one special character and search for the special character



```
# Bit-Counting
.section .data

.section .bss

.section .text

.globl _start

# function for system exit code
_ret:
    movq    $60, %rax               # sys_exit
    movq    $0, %rdi                # exit code
    syscall

# driver function
_start:

    movl $0x07, %ebx    # b = 7 // actual number
    movl $0, %ecx       # c = 0 // to keep track of the number of 1's
loop:
    movl $1, %eax       # a = 1
    andl %ebx, %eax     # a = a & b
    addl %eax, %ecx     # c = c + a
    sarl %ebx
    cmp $0, %ebx
    jne loop

    movl %ecx, %eax     # a = c
    andl $1, %eax       # a = a & 1 // if even no. of 1's then 0 else 1

    syscall
    call _ret           # exit
```

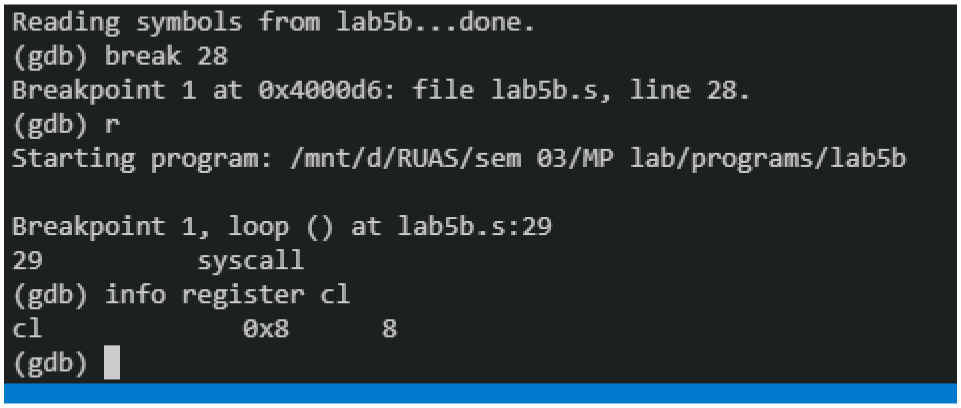
Figure 3 ASM code of program to compute the parity of a hexadecimal number stored in the Register1. If Register1 has odd number of ones, update Register2 with 0x01. If Register1 has even number of ones, update Register2 with 0x00.

Output of the programs:



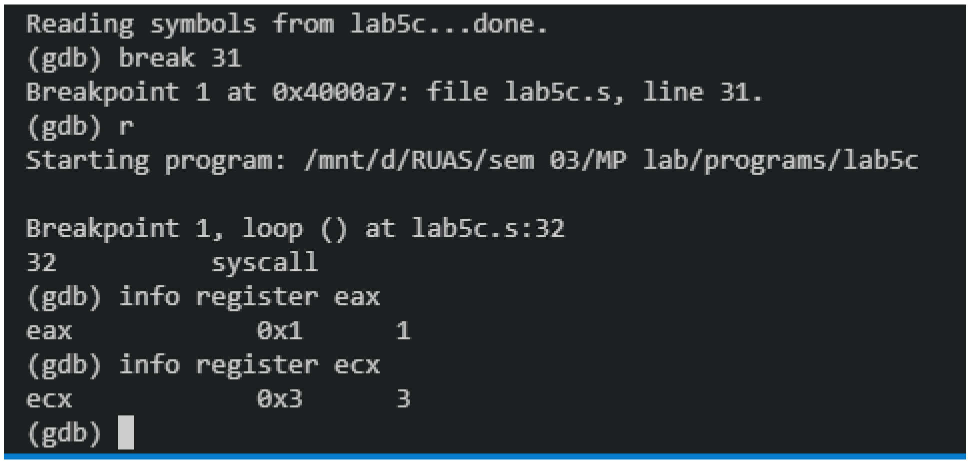
```
Reading symbols from lab5a...done.
(gdb) break 28
Breakpoint 1 at 0x4000de: file lab5a.s, line 28.
(gdb) r
Starting program: /mnt/d/RUAS/sem 03/MP lab/programs/lab5a

Breakpoint 1, loop () at lab5a.s:29
29          syscall
(gdb) info register ecx
ecx            0x5      5
(gdb)
```

Figure 4 '5' is the position of target in the array



```
Reading symbols from lab5b...done.
(gdb) break 28
Breakpoint 1 at 0x4000d6: file lab5b.s, line 28.
(gdb) r
Starting program: /mnt/d/RUAS/sem 03/MP lab/programs/lab5b

Breakpoint 1, loop () at lab5b.s:29
29          syscall
(gdb) info register cl
cl             0x8      8
(gdb)
```

Figure 5 '8' is the position of target '@' in the string



```
Reading symbols from lab5c...done.
(gdb) break 31
Breakpoint 1 at 0x4000a7: file lab5c.s, line 31.
(gdb) r
Starting program: /mnt/d/RUAS/sem 03/MP lab/programs/lab5c

Breakpoint 1, loop () at lab5c.s:32
32          syscall
(gdb) info register eax
eax            0x1      1
(gdb) info register ecx
ecx            0x3      3
(gdb)
```

Figure 6 '3' is the number of ones present in hexa-decimal number (7). 1 is parity(odd parity) .

7. Analysis and Discussions

Code	`jcc address`
Example	`jne loop`
Explanation	Performs: Jumps to the address location if the condition is met Here cc = ne, e, ge, g, etc. Description: Checks the state of one or more of the status flags in the EFLAGS register (CF, OF, PF, SF, and ZF) and, if the flags are in the specified state (condition), performs a jump to the target instruction specified by the destination operand. A condition code (cc) is associated with each instruction to indicate the condition being tested for. If the condition is not satisfied, the jump is not performed and execution continues with the instruction following the Jcc instruction.

Code	`cmp op1 op2`
Example	`cmp $0, %eax`
Explanation	Performs: Compares the two operands Description: Compares the first source operand with the second source operand and sets the status flags in the EFLAGS register according to the results. The comparison is performed by subtracting the second operand from the first operand and then setting the status flags in the same manner as the SUB instruction. When an immediate value is used as an operand, it is sign-extended to the length of the first operand.

Code	`and <source> <destination>`
Example	`andl $20, %ebx`
Explanation	Performs: `Destination = Destination AND Source` Description: Performs a bitwise AND operation on the destination (first) and source (second) operands and stores the result in the destination operand location. The source operand can be an immediate, a register, or a memory location; the destination operand can be a register or a memory location.

Code	`sal <shift_amt> <destination>`
	`sar <shift_amt> <destination>`

Example	`sal $2, %ebx`
	`sar $7, %ebx`

Explanation	Performs: `Destination = bitwise shift destination shit_amt times, either to the left or to the right, depending upon usage of sal or sar respectively.` Description: The shift arithmetic left (SAL) and shift logical left (SHL) instructions perform the same operation; they shift the bits in the destination operand to the left (toward more significant bit locations). The shift arithmetic right (SAR) and shift logical right (SHR) instructions shift the bits of the destination operand to the right (toward less significant bit locations).

8. Conclusions

Execution Flow can be controlled by using conditional instructions, which includes a cmp instruction followed by a jump instruction, a cmp instruction compares the two operands and updates the flag register, this is then used with jump instruction to go to some other part of the program.

9. Comments

1. Limitations of Experiments

Although looping structures can be formed using the cmp, jcc instructions but recursive structures are complex to form using just these instructions.

2. Limitations of Results

None.

3. Learning happened

We learnt the use of compare, unconditional jump and conditional jump instructions to form looping structures and conditional statements.

4. Recommendations

Since a program can contain numerous loop labels, each label should be carefully names, and the programmer must keep track of which parts of the program jump to where, else there might be chances of forming infinite loops.

Signature and date Marks