# 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

1. 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

1. 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

1. 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: @, .

1. 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.

1. 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 ecxth 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 ecxth 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

1. Presentation of Results

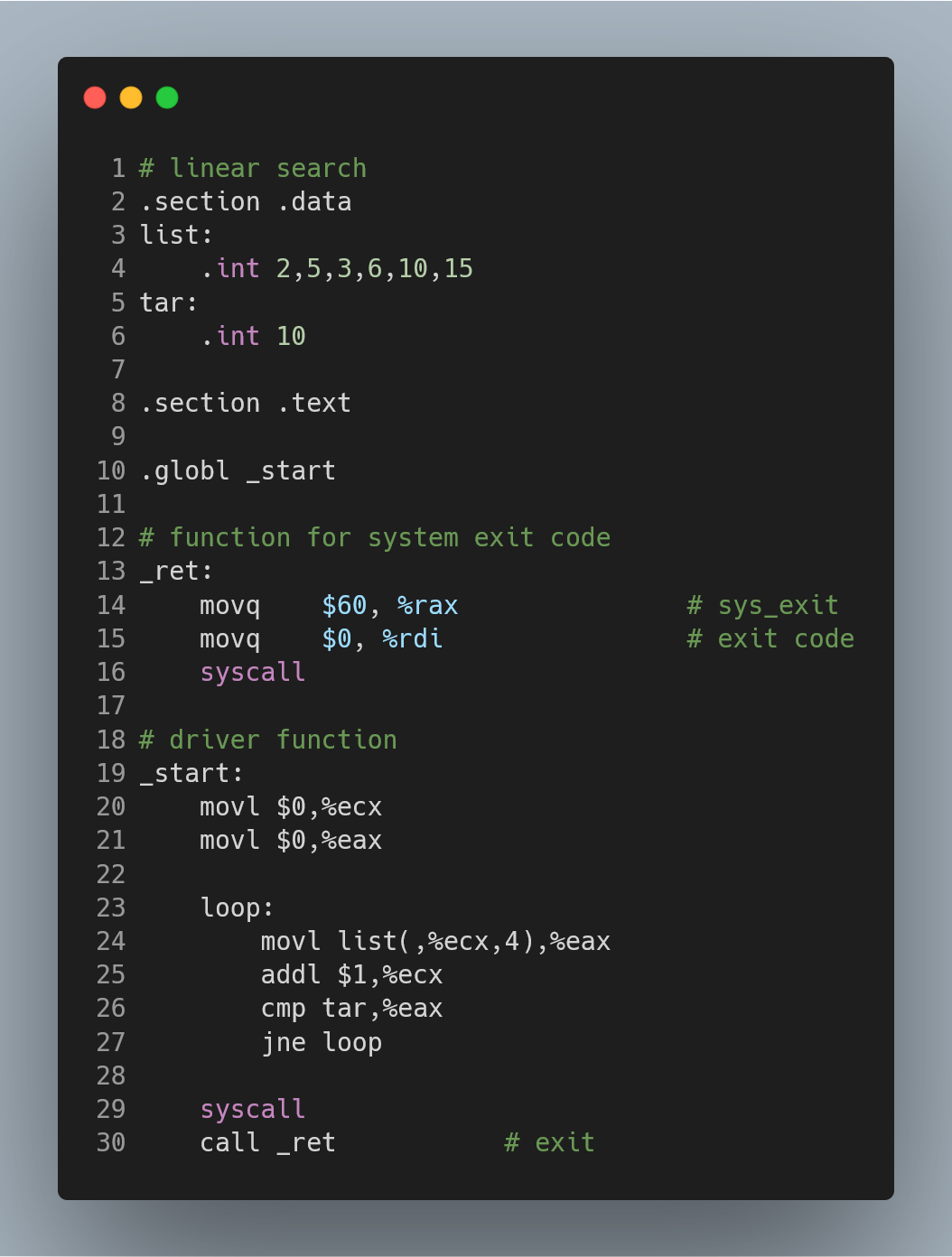


Figure 1 ASM code of Searching an element in an array of ‘n’ numbers

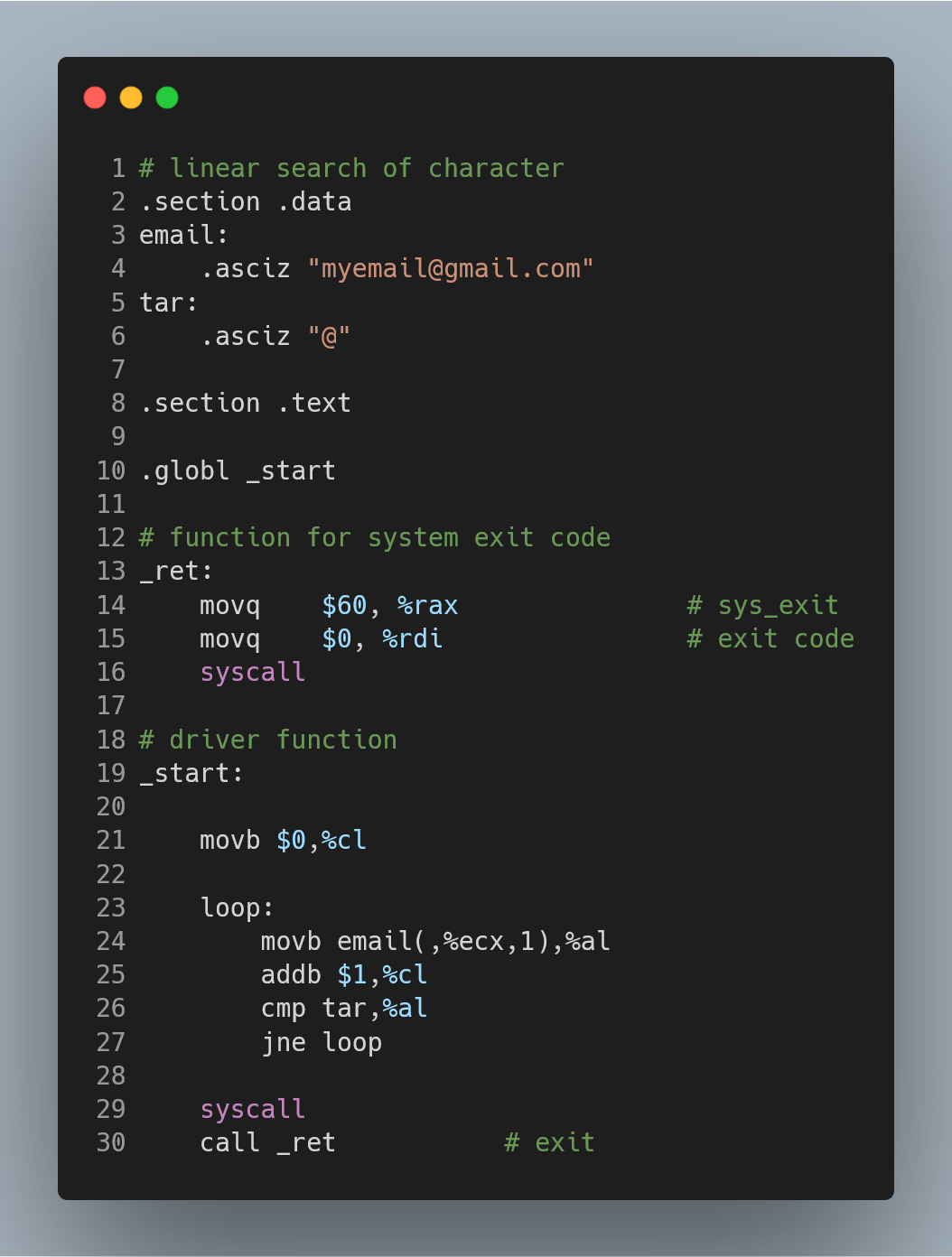


Figure 2 ASM code of program to Read a sentence with at least one special character and search for the special character



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:

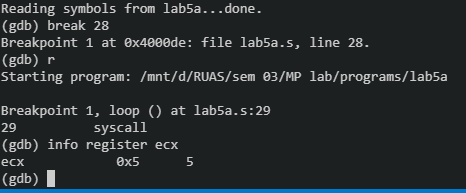


Figure 4 '5' is the position of target in the array

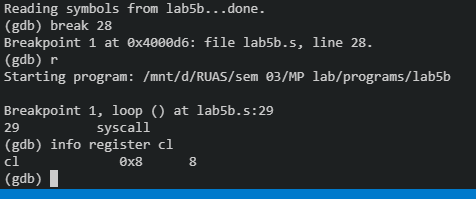


Figure 5 '8' is the position of target '@' in the string

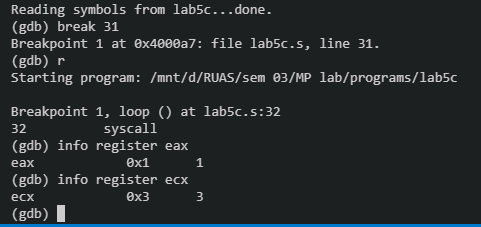


Figure 6 '3' is the number of ones present in hexa-decimal number (7). 1 is parity(odd parity) .

1. 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). |

1. 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.

1. 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