CSE-3232 (Microprocessor and Assembly Language Lab)

Table of Contents:

Some Basic Terminology for String Operation in Assembly Language

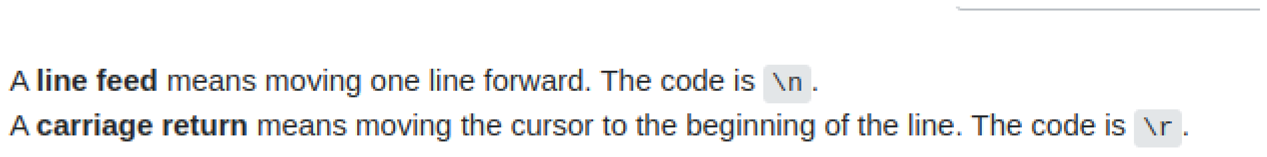
Note: File `./Basics.pdf` contains some basic information with proper examples.

- Hello world
- Service Routine
- Input and Output
- Loop (Simple and Nested)
- Jump (Both Conditional and Unconditional)

Take String Input and Display Output

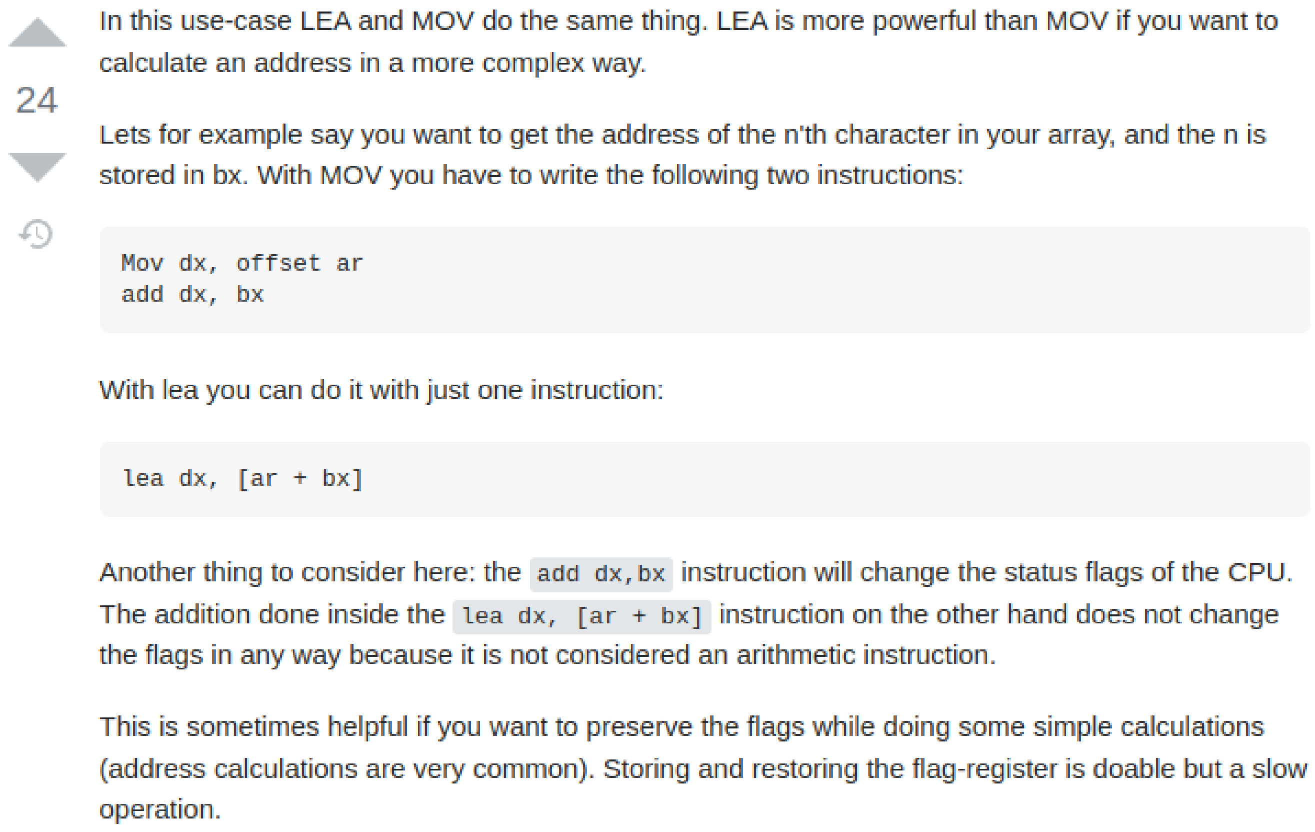
```
; read string
.model small
.stack 100h

.data
; msg is a array variable with length 50, we places '$' in dup() because in assembly
; string ends with character '$'

msg db 50 dup('$')
newline db 10, 13, '$'    ; 10 -> line feed and 13 carriage return

.code
main proc
    ; initialize data segment
    mov ax, @data
    mov ds, ax

    ; below both statement are same
    ;mov si, offset msg
    ; To access array we need to define the si register with start offset address of the variable
    lea si, msg                  ; mov starting offset address into the si register
    ; read string and store into the variable msg
    loop_1:
        mov ah, 01                ; read single character and by default it store on the al register
        int 21h

        ; if user press enter key the program jumpped into the display_and_exit where we display
        ; what we store on the msg variable
        cmp al, 13 ; 13 enter key
        je display_and_exit


        mov [si], al
        ; increment offset
        inc si
    loop loop_1


    display_and_exit:
        ; we need to print newline because if we don't print newline
        ; the result is shown above the input ... you getting confused.
        mov dx, offset newline
        ; lea dx, newline
        mov ah, 09         ; print string until find '$' sign
        int 21h

        ; lea dx,  msg
        mov dx, offset msg
        mov ah, 09
        int 21h

        mov ah, 4ch
        int 21h

main endp
end main
```

"Line Feed" Vs "Carriage Return"



A **line feed** means moving one line forward. The code is `\n`.
A **carriage return** means moving the cursor to the beginning of the line. The code is `\r`.

`lea` and `offset`



In this use-case LEA and MOV do the same thing. LEA is more powerful than MOV if you want to calculate an address in a more complex way.

Lets for example say you want to get the address of the n'th character in your array, and the n is stored in bx. With MOV you have to write the following two instructions:

```
Mov dx, offset ar
add dx, bx
```

With lea you can do it with just one instruction:

```
lea dx, [ar + bx]
```

Another thing to consider here: the `add dx,bx` instruction will change the status flags of the CPU. The addition done inside the `lea dx, [ar + bx]` instruction on the other hand does not change the flags in any way because it is not considered an arithmetic instruction.

This is sometimes helpful if you want to preserve the flags while doing some simple calculations (address calculations are very common). Storing and restoring the flag-register is doable but a slow operation.

> If you are already fimiliar with take string input and display it on the monitor then you are ready for perform various string operation.

Experiment-1 (String Reverse)