

Ministerul Educatiei, Culturii și Cercetarii al Republicii Moldova Universitatea Tehnic**ă** a Moldovei

Facultatea Calculatoare, Informatic**ă ş**i Microelectronic**ă** Departamentul Ingineria Software și Automatica

Report

for laboratory work No. 1

course "Operational Systems"

Done by: Felicia LUPAȘCU, gr. FAF-212

Checked by:  
 Rotislav Călin

Chișinău – 2023

**Subject:** ECHO1 (work with keyboard)

**Tasks:**

Create a program in assembler which will "echo" what is typed from the keyboard. Each ASCII character which will be pressed from the keyboard should appear on the screen and the cursor should move to the next position. Special actions need to be implemented only for 2 special keys from the keyboard:

● "backspace key" - in this case the symbol from the left side of the cursor should disappear and the cursor should be moved one position back (If the cursor already is in the first position, then nothing should happen. Special case is if the cursor is on the next line, than when is pressed Backspace in the first column, than cursor should move to the previous line in last column);

● "enter key" - in this case all previously introduced string should be printed to the screen starting with the next line and after one "empty" line (but if "enter key" will be pressed as the first key, in this case NO "empty" line should be added and the action should just go to the next line)

(OPTIONAL)

The maximum length of input string should not exceed 256 characters. If the user wants to input more than 256 characters than the input should be stopped and in this case only "backspace" or "enter" keys should be accepted.

> Compiled program should be used in order to create a floppy image and it should be bootable. Use this image to boot the OS in a VirtualBox VM and the text which you intended to print should appear on the screen.

> You can use any assembly compiler.

> Students should be able to modify the code, to recompile it and to boot the VM with a new version of the program.

> In order to use documentation from TechHelp/XView DOS application, students can install DosBox

**Implementation:**

In order to compile the assembly code and create images in order to run in the Virtual Machine from floppy I used the following script:

import os  
size = os.path.getsize("lab2test.bin")  
small = open("lab2test.bin", "rb")  
big = open("big2test.img", "wb")  
big.write(small.read())  
bytes = b'\x00' \* (1474560 - size)  
big.write(bytes)

This “**typing**” loop continually waits for a key press, and based on the pressed key, it takes different actions. If it's the Backspace key, it jumps to handle Backspace; if it's the Enter key, it jumps to handle Enter; if it's a space or printable character, it jumps to handle printing. If none of these conditions are met, it continues waiting for keyboard input.

typing: ; Label for the main input loop.  
 mov AH, 0 ; Clear AH register.  
 int 0x16 ; Wait for a keyboard key press and store it in AL.  
  
 cmp AH, 0x0e ; Check if the Backspace key was pressed.  
 je backspace\_key ; If so, jump to the "backspace\_key" label.  
  
 cmp AH, 0x1c ; Check if the Enter key was pressed.  
 je enter\_key ; If so, jump to the "enter\_key" label.  
  
 cmp AL, 0x20 ; Check if the entered character is a space or a printable character.  
 jge print\_character; If yes, jump to the "print\_character" label.  
  
 jmp typing ; Otherwise, continue reading user keyboard inputs.

**Backspace Section:**

This code handles the Backspace key by checking if the buffer is empty, clearing the character at the current position in the buffer, and adjusting the cursor position accordingly. If the cursor is at the beginning of the line, it moves to the previous line; otherwise, it moves to the previous character position.

backspace\_key: ; Label for handling the Backspace key.  
 cmp SI, buffer ; Check if the buffer is empty.  
 je typing ; If empty, go back to the input loop.  
 dec SI ; Decrement SI to move the pointer back.  
 mov byte [SI], 0 ; Clear the character at the current position in the buffer.  
  
 mov AH, 03h ; Get cursor position.  
 mov BH, 0 ; Set BH to 0 (video page).  
 int 0x10 ; Get the current cursor position.  
  
 cmp DL, 0 ; Check if DL (column) is at the beginning of the line.  
 jz last\_line ; If so, move the cursor to the previous line.  
 jmp last\_character ; Otherwise, move the cursor to the previous character position.

last\_character moves the cursor to the previous character position by decrementing the column, and last\_line moves the cursor to the previous line by decrementing the row and setting the column to the end of the line.

last\_character: ; Label for moving the cursor to the previous character position.  
 mov AH, 02h ; Set cursor position.  
 dec DL ; Decrement DL (column) to move left.  
 int 0x10 ; Set the new cursor position.  
 jmp space\_character ; Jump to space character handling.  
  
last\_line: ; Label for moving the cursor to the previous line.  
 mov AH, 02h ; Set cursor position.  
 mov DL, 79 ; Set DL (column) to 79 (end of line).  
 dec DH ; Decrement DH (row) to move up one line.  
 int 0x10 ; Set the new cursor position.

**Space Section**  
When a space character is encountered, this code displays the space character on the screen and then continues to wait for additional user input in the main input loop.

space\_character: ; Label for handling space character.  
 mov AH, 0Ah ; Set AH to 0xa (teletype output).  
 mov AL, 0x20 ; Set AL to 0x20 (space character).  
 mov CX, 1 ; Set CX to 1 (number of repetitions).  
 int 0x10 ; Display the space character.  
 jmp typing ; Continue reading user input.

**Enter Section**

This code handles the Enter key by getting the cursor position, checking if the cursor is in the last row, and performing scrolling if necessary. If the cursor is not in the last row, it proceeds to a label (print\_echo) to handle printing. If the cursor is in the last row, it scrolls the window up and sets the cursor to the last visible row.

enter\_key: ; Label for handling the Enter key.  
 mov AH, 03h ; Get cursor position.  
 mov BH, 0 ; Set BH to 0 (video page).  
 int 0x10 ; Get the current cursor position.  
  
 sub SI, buffer ; Calculate the difference between SI and the buffer address.  
 jz insert\_line ; If SI equals the buffer address, jump to insert\_line.  
  
 mov AH, 03h ; Get cursor position.  
 mov BH, 0 ; Set BH to 0 (video page).  
 int 0x10 ; Get the current cursor position.  
  
 cmp DH, 24 ; Check if DH (row) is 24 (last row).  
 jl print\_echo ; If not, jump to print\_echo.  
  
 mov AH, 0x06 ; Set AH to 0x06 (scroll window up).  
 mov AL, 1 ; Set AL to 1 (number of lines to scroll).  
 mov BH, 0x07 ; Set BH to 0x07 (attribute for blank lines).  
 mov CX, 0 ; Set CX to 0 (upper left corner).  
 mov DX, 0x184f ; Set DX to the bottom right corner.  
 int 0x10 ; Scroll the window up.  
 mov DH, 0x17 ; Set DH to 0x17 (last visible row).

*1. insert\_line:* Checks if the cursor is in the last row. If true, it prepares to get the cursor position, and if the cursor is not in the last row, it jumps to a label (scroll\_down) to handle scrolling the window up and moving the cursor down.

*2. scroll\_down:* Handles scrolling the window up, ensuring that the content moves smoothly on the screen when the Enter key is pressed, creating a new line.

*3. cursor\_down:* Moves the cursor down by one line, adjusting the cursor position to the beginning of the line if necessary. This is part of the mechanism to facilitate the movement of the cursor when a new line is inserted.

**Print Echo Section**

This code sets up the necessary registers and parameters for teletype output, specifying the color, buffer position, and cursor position, and then triggers an interrupt to display the character on the screen and update the cursor. This is part of the overall mechanism to print characters to the screen.

print\_echo: ; Label for printing characters to the screen.  
 mov BH, 0 ; Set BH to 0 (video page).  
 mov AX, 0 ; Clear AX register.  
 mov ES, AX ; Set ES to 0 (extra segment).  
 mov BP, buffer ; Set BP to point to the buffer.  
 mov BL, 0Ch ; Set BL to 0Ch (color: red on black).  
 mov CX, SI ; Set CX to the current buffer position.  
 inc DH ; Increment DH to move to the next row.  
 mov DL, 0 ; Set DL to 0 (column).  
 mov AX, 1301h ; Set AX to 1301h (teletype output, update cursor).  
 int 0x10 ; Display the character and update the cursor.

**Print Character Section**

In summary, this code segment checks if the entered character is a control character or if the buffer is at its maximum capacity. If neither condition is met, it stores the character in the buffer, displays it on the screen, and then continues to read more user input in the main input loop.

print\_character: ; Label for printing a character.  
 cmp AL, 0x7f ; Check if the entered character is a control character.  
 jge typing ; If yes, jump back to reading user input.  
 cmp SI, buffer + 256 ; Check if SI is at the end of the buffer.  
 je typing ; If so, jump back to reading user input.  
 mov [SI], AL ; Store the entered character in the buffer.  
 inc SI ; Increment SI to move to the next buffer position.  
 mov AH, 0xe ; Set AH to 0xe (teletype output).  
 int 10h ;Display the entered character.  
 jmp typing ; Continue reading user input.

**Output:**

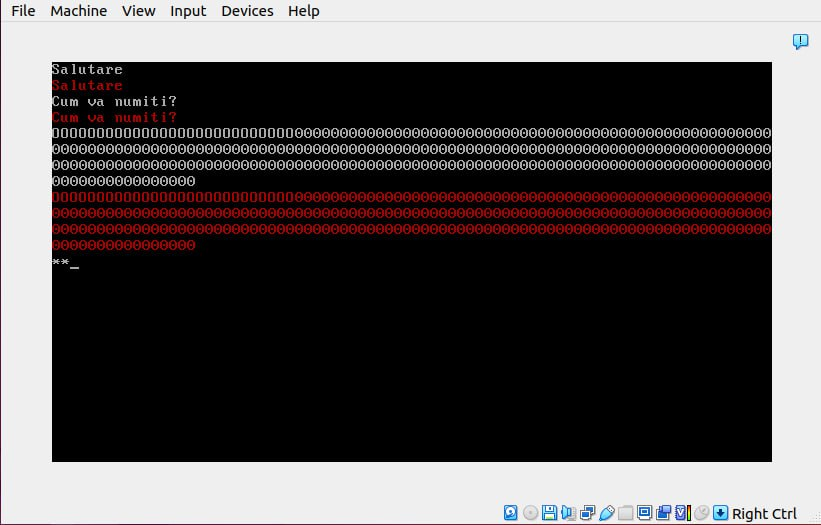


Figure 1 Output

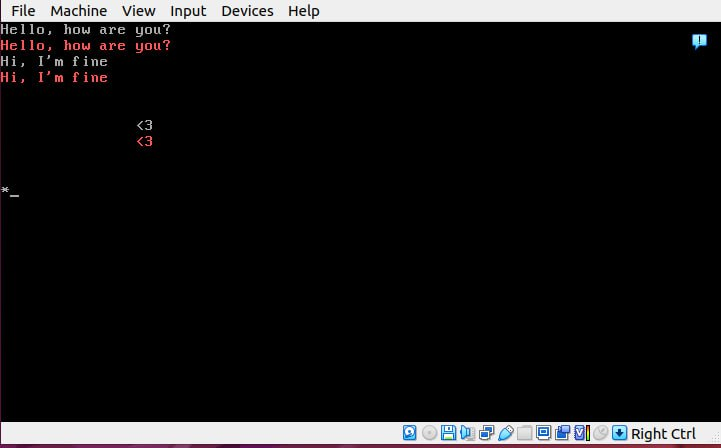


Figure 2 Example of how the limit of characters works

When a user runs the program, he has the possibility to enter the message and the program will echo it back. Also, the program doesn’t allow you to enter a message which is bigger than 256 characters.

**Conclusion:**

In this laboratory work, I successfully implemented a simple text input program in assembly language. The program efficiently captures keyboard inputs, providing real-time feedback on the screen. I incorporated functionalities to handle special keys, such as Backspace and Enter, ensuring a smooth user experience.

<https://github.com/feliciaL3/SO_Labs/blob/main/Lab2/lab2test.asm>