Ministerul Educației, Culturii și Cercetării al Republicii Moldova
Universitatea Tehnică a Moldovei
Facultatea Calculatoare, Informatică și Microelectronică
Departamentul Ingineria Software și Automatică

Raport

pentru lucrare de laborator Nr. 2
la cursul Sisteme de Operare
“Work with keyboard”

A efectuat: Arteom KALAMAGHIN, FAF-211
A verificat: Rostislav CĂLIN

Chișinău - 2023

Subject: *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 then the input should be stopped and in this case only "backspace" or "enter" keys should be accepted.

Implementation:

* In order to compile the assembly code and create an image that is possible to run in a VM from floppy I use the following bash script:

```
#!/bin/bash

if [ $# -ne 1 ]; then
   echo "Usage: $0 <filename>"
   exit 1
fi

filename="$1"
ext1=".asm"
ext2=".com"

cp "$filename$ext1" backup/
rm -f "floppy.img"

nasm -f bin -o "$filename$ext2"
"$filename$ext1"
truncate -s 1474560 "$filename$ext2"
mv "$filename$ext2" floppy.img
```

The main process looks like that we read the key pressed, if that is Backspace or Enter we handle it appropriately, otherwise we save the character in the buffer and print it on the screen …

```
typing:
   mov     AH, 00h             ; Set AH register to 00h - read keyboard input
   int     16h                 ; Call the interruption to get the key press

   cmp     AL, 08h             ; Compare the value in AL with Backspace (08h) ...
   je      handle_backspace    ; If equal handle Backspace

   cmp     AL, 0Dh             ; Compare the value in AL with Enter (0Dh) ...
   je      handle_enter        ; If equal handle Enter

   cmp     SI, input + 256     ; Compare SI with the end of the buffer ...
   je      typing              ; If 256 characters were inserted, leave only Enter
                               ; and Backspace as options

   mov     [SI], AL            ; Store the character in AL in the buffer at [SI]
   add     SI, 1               ; Increment SI to point to the next buffer location

   mov     AH, 0eh             ; Set AH to 0eh - write character as TTY
   int     10h                 ; Call the interruption to print on the screen

   jmp     typing              ; Continue typing
```

On the following pages, you will see the code for the *appropriate handling* mentioned.

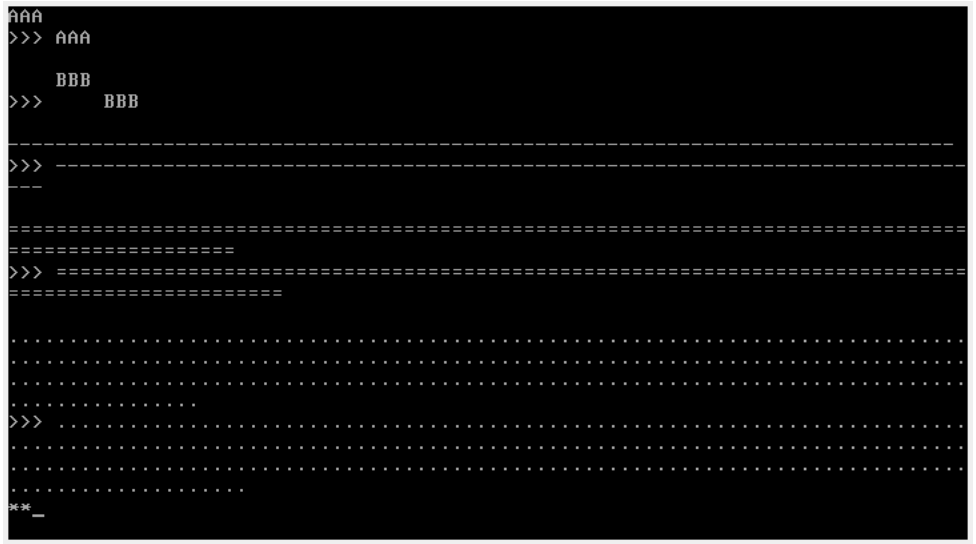
```
handle_backspace:
   cmp     SI, input           ; Compare SI with the start of the buffer
   je      typing              ; If SI is at the start - line is empty, we skip

   dec     SI                  ; Decrement SI to point to the previous buffer location
   mov     byte [SI], 0        ; Erase the character in the buffer at [SI]

   mov     AH, 03h             ; Set AH to 03h - query cursor pos. and size
   mov     BH, 0               ; From the first page ...
   int     10h                 ; Call the interruption to get the cursor information

   cmp     DL, 0               ; Prev. interruption saved the cursor column to DL ...
   jz      previous_line       ; If cursor is at the start of the line - return to the
                               ; prev. line

   ; Otherwise, print a blank space to effectively erase the last typed character

   mov     AH, 02h             ; Set AH to 02h - set cursor position
   dec     DL                  ; Decrement DL to return the cursor one column back
   int     10h                 ; Call the interruption to move the cursor

   mov     AH, 0eh             ; Set AH to 0eh - write character as TTY
   mov     AL, 20h             ; 20h for the blank space character
   int     10h                 ; Call the interruption to print on the screen

   ; TTY advanced the cursor automatically so we need the return it once more

   mov     AH, 02h             ; Set AH to 02h - set cursor position
   int     10h                 ; Call the interruption to move the cursor

   jmp     typing              ; Continue typing

previous_line:
   mov     AH, 02h             ; Set AH to 02h - set cursor position
   mov     DL, 79              ; Set DL to 79 (last column)
   dec     DH                  ; Decrement DH to return one row back (up)
   int     10h                 ; Call the interruption to move the cursor

   ; There is a character on this position we need to erase

   mov     AH, 0eh             ; Set AH to 0eh - write character as TTY
   mov     AL, 20h             ; 20h for the blank space char.
   int     10h                 ; Call the interruption to print on the screen

   ; TTY advanced the cursor automatically so, to end up in the last
   ; column of the prev. row, we need to move it one column back

   mov     AH, 02h             ; Set AH to 02h for the set cursor function
   int     10h                 ; Call the interruption to move the cursor

   jmp     typing              ; Continue typing
```

I guess these explicit comments are enough to explain what happens. On the next page you may find the code for enter handling.

```
handle_enter:
   mov     AH, 03h             ; Set AH to 03h - query cursor pos. and size
   mov     BH, 0               ; From the first page ...
   int     10h                 ; Call interrupt 10h to get cursor information

   sub     SI, input           ; Calculate the number of characters in the buffer
   je      move_curs_down      ; If SI == 0 (no characters were in the buffer), just \
                               ; advance one row down

   cmp     DH, 24              ; Compare DH with 24 (the max. row val)...
   jmp     print_buffer        ; If DH is less than 24, print the buffer

   ; Else it is possible to scroll the screen down to fit another line ...

print_buffer:
   mov     BH, 0               ; On the first page ...
   inc     DH                  ; Increment DH - from the next row

   ; A short ">>> " in front of the buffer output to indicate the echo part of displayed
   ; text …

   ; Need to get and set the cursor position to prevent the buffer display from
   ; overwriting the ">>> " ...

   ; Print the buffer contents

   mov     AX, 0               ; Clear AX register
   mov     ES, AX              ; Set ES register to 0 for video memory
   mov     BP, input           ; Send reference to the start of the buffer
   mov     BL, 07h             ; Set BL to 07h - print in light-gray
   mov     CX, SI              ; Set CX to the number of characters in the buffer
                               ; (stored in SI after line 89)
   mov     AX, 1301h           ; Set AH to 1300h - display string and advance the cursor
   int     10h                 ; Call the interrupt to display the string

move_curs_down:
   mov     AH, 03h             ; Set AH to 03h - query cursor pos. and size
   mov     BH, 0               ; From the first page ...
   int     10h                 ; Call interrupt 10h to get cursor information

   mov     AH, 02h             ; Set AH to 02h - set cursor position
   mov     BH, 0               ; On the first page ...
   add     DH, 1               ; Increment DH - on the next row
   mov     DL, 0               ; Set DL to 0 - from the start of the line
   int     10h                 ; Call the interruption to move the cursor

   mov     SI, input           ; Reset the buffer pointer to be ready to read a new line
                               ; of characters and

   jmp     typing              ; Continue typing
```

Simply advance the row or echo everything typed and move the cursor over everything displayed - just a piece of cake, again explicitly commented.

And here are some results:



Conclusion:

In this lab exercise, I was tasked with creating an assembly program that implements keyboard input and echoing functionality. The primary objectives were to echo characters typed on the keyboard, provide special handling for the backspace key, and handle the enter key by printing the entered string on the screen. In conclusion, this lab work provided a practical experience in low-level programming and operating system development. It taught me how to handle keyboard input, manipulate the output and direct execution flow of the program. Additionally, the exercise emphasized the importance of documentation which I consulted extensively during the implementation by exploring all the codes needed to achieve the desirable result. By creating a bootable floppy image, I was able to test my program in a controlled environment, and the successful execution of the program on the virtual machine demonstrated my ability to create a functional piece of software from scratch. Overall, this lab work enhanced my understanding of assembly programming and its application in building simple operating system components.