Ministry of Education and Investigation Republic of Moldova

Technical University of Moldova

Faculty of Computers, Informatics and Microelectronics

REPORT

Laboratory work nr.2

on the course “Operating Systems”

Executed by:

st. gr. FAF-212 Ciumac Alexei

Verified by:

prof. univ. Rostislav Calin

Chişinău - 2023

Topic: scanning input from keyboard and outputting itself back;

Tasks:

Each ASCII character 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 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, then when is pressed Backspace in the first column, then 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)

Implementation and results:

Tasks were implemented in a single file. The program starts with initializing character counter, buffer and jumping to "read_char":

```
org 7c00h
; nasm -f bin lab2.asm -o  lab2.bin
section .data
  charCounter db 0


section .bss
  buffer resb 255


section .text
  global _start
```

```
_start:
  ; initialize the buffer and its counter
  mov si, buffer
  mov byte [charCounter], 0


  jmp ReadChar
```

“read char” accepts any char inputted via 16h 00h interruption. By comparing the input segment al with Backspace and Enter ASCII, it decides whether it will jump to “handle_backspace”, “handle_enter” or will go further. Next comparison checks if string length hasn’t passed value of 256, and otherwise jumps back to this function recursively. If no condition was satisfied it goes further writes current character to the current pointer at buffer segment moves it to one further and increases char_counter. Further it displays character, moves cursor one place further and jumps back recursively.

```
ReadChar:
  ; read character
  mov ah, 00h
  int 16h


  ; check if the ENTER key was introduced
  cmp al, 0dh
  je HoldEnter


  ; check if the BACKSPACE key was introduced
  cmp al, 08h
  je HoldBackspace
```

```
; check if the buffer limit is reached
cmp byte [charCounter], 255
je ReadChar


; add character into the buffer and increment its pointer
mov [si], al
inc si
inc byte [charCounter]


; display character
mov ah, 0ah
mov bh, 0x00
mov cx, 1
int 10h


; move cursor
mov ah, 02h
inc dl
int 10h


jmp ReadChar
```

In cases when inputted character is Backspace the program checks if the cursor is on the left border of the window and if yes it jumps on the “prev_line”, where it checks if cursor is on the top line and if yes jumps back to _start, otherwise it changes cursor position and also jumps back to _start. If no conditions were met it decreases buffer pointer, decreases current string length, goes back one symbol and prints blank instead of a character. In the end it jumps back to “read_char”.

```
HoldBackspace:
  cmp dl, 0
  je PrevLine


  ; clear last buffer char
  dec si
  dec byte [charCounter]


  ; move cursor to the left
  mov ah, 02h
  dec dl
  int 10h


  ; print space instead of the cleared char
  mov ah, 0ah
  mov al, ' '
  mov bh, 0x00
  mov cx, 1
  int 10h


  jmp ReadChar
```

Check Cursor Position: Compares the current column position (dl) with 0. If the cursor is at the beginning of the line, it jumps to the newline label.

Check if Buffer is Empty: Compares the character counter (char_counter) with 0. If the buffer is empty, it also jumps to the newline label.

Clear Buffer: Sets the value at the memory location pointed to by si to 0, effectively clearing the character buffer. Then, it resets the buffer pointer si to the beginning of the buffer.

Move Cursor to Next Line: Uses BIOS interrupt 10h to set the cursor position to the second line below the current one (inc dh twice), with the column position (dl) set to 0.

Jump to Print Buffer: Jumps to the print_buffer label, which displays the characters in the buffer on the new line.

Print Buffer: This part of the code, labeled print_buffer, prints the characters in the buffer one by one, moving the cursor right after each character.

Newline: The newline label is a target for jumping after handling Enter key presses. It moves the cursor to the beginning of the next line using BIOS interrupt 10h.

```
HoldEnter:
  cmp dl, 0
  je Newline


  cmp byte [charCounter], 0
  je Newline


  ; clear the character buffer
  mov byte [si], 0
  mov si, buffer


  ; move cursor to the second next line
  mov ah, 02h
  inc dh
  inc dh
  mov dl, 0
  int 10h
```

```
  jmp PrintBuffer


PrintBuffer:
  lodsb; load character form edi into al

  test al, al
  jz Newline

  ; display character
  mov ah, 0ah
  mov bh, 0x00
  mov cx, 1
  int 10h

  ; move cursor
  mov ah, 02h
  inc dl
  int 10h

  jmp PrintBuffer


; move cursor to the beginning of the new line
Newline:
```

```
  mov ah, 02h
  inc dh
  mov dl, 0
  int 10h

  jmp _start


PrevLine:
  cmp dh, 0
  je _start

  mov ah, 02h
  dec dh
  mov dl, 79
  int 10h

  jmp _start
```

The result of the above program is presented on the screenshot below:

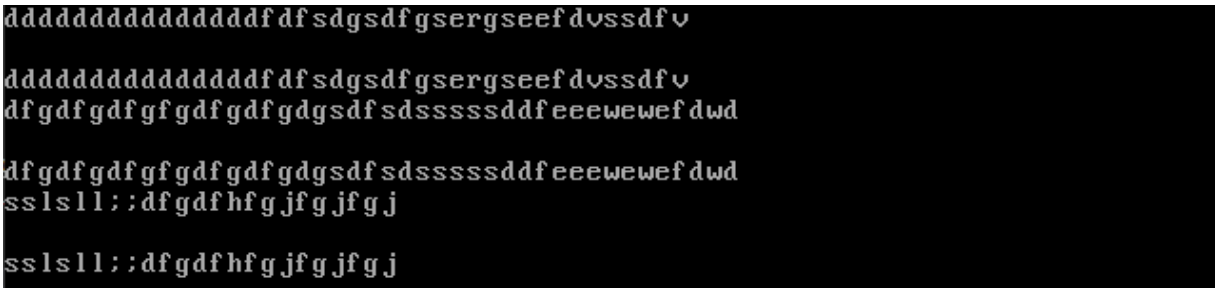

Figure 1 - Output of the program

Conclusions:

In this laboratory assignment, the assembly code successfully implemented crucial features, including robust handling of Enter and Backspace keys, dynamic screen updates, and efficient management of a character buffer. The utilization of BIOS interrupts, particularly interrupt 16h for keyboard input and interrupt 10h for screen output, highlighted the seamless integration of hardware-level interactions within the program.

The creation of an echo program provided valuable insights into the nuances of assembly language, underscoring the significance of adeptly managing memory, registers, and interrupts. This hands-on experience in developing a program that interfaces with low-level hardware components deepened our comprehension of computer architecture and the intricate execution flow of a basic assembly program.