**Ministry of Education and Investigation Republic of Moldova**

**Technical University of Moldova**

**Faculty of Computers, Informatics and Microelectronics**

# REPORT

Laboratory work nr.4

on the course “Operating Systems”

Executed by:

st. gr. FAF-212 Alexei Ciumac

Verified by:

prof. univ. Rostislav Calin

Chişinău - 2023

**Topic:** Boot Loader

**Tasks:**

Create an assembly language application that will act as a Boot Loader and do the following:

1. It will display a greeting message that will include the author's name and will wait for the keyboard input of the "source" address on the floppy, from where to read the kernel (or other compiled code that is desired to be loaded and executed). The address will be entered in the format SIDE, TRACK, SECTOR and the address must be strictly within the range reserved for the student author as it was in Lab3.

2. It will wait for the keyboard to read the "destination" address of the RAM where to load the data block read from the floppy. The RAM address must be in the format XXXXh:XXXXh, the same as it was for Lab3.

3. It will transfer the data FLOPPY ==> RAM and display the error code with which the given operation completed.

4. It will prompt you to type a key and launch the kernel (or execute the code you want to execute).

5. After completion of kernel execution or executed code, it will display a message to type a key and run Boot Loader repeatedly

**Implementation and results:**

***colorflag.asm***

```
section .text
    global _start


_start:
    ; receive segment:offset pair from the bootloader
```

```
mov [add1], ax
mov [add2], bx

mov si, [add1]
mov ds, [add2]

mov byte [video_mode], 13
mov byte [pixel_color], 0

mov byte [line_length], 0
mov byte [stripe_width], 0

mov word [left_indent], 10
mov word [stripe_indent], 0

mov word [line_number], 10

mov byte [stripes], 0
mov word [stripe_height], 0

mov byte [char_counter], 0
mov byte [result], 0

mov byte [page], 0
```

```
    mov byte [c], 0

    jmp menu


menu:
    mov byte [page], 0
    mov word [line_number], 10

    ; set text video mode
    mov ah, 00h
    mov al, 2
    int 10h

    ; print command disclaimer
    call find_current_cursor_position

    mov ax, [add2]
    mov es, ax
    mov bh, [page]
    mov bl, 07h
    mov cx, disclaimer_length

    mov ax, disclaimer
```

```
add ax, word [add1]
mov bp, ax

mov ax, 1301h
int 10h

call newline

; print reboot option
; print command disclaimer
call find_current_cursor_position

mov ax, [add2]
mov es, ax
mov bh, [page]
mov bl, 07h
mov cx, reboot_prompt_length

mov ax, reboot_prompt
add ax, word [add1]
mov bp, ax

mov ax, 1301h
int 10h
```

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

cmp al, 'r'
je reboot

call newline

; input stripe width
call find_current_cursor_position

mov ax, [add2]
 mov es, ax
mov bh, [page]
 mov bl, 07h
mov cx, stripe_width_prompt_length

mov ax, stripe_width_prompt
add ax, word [add1]
 mov bp, ax

 mov ax, 1301h
```

```
int 10h

mov byte [result], 0
call clear_buffer
call read_buffer

mov al, [result]
mov byte [stripe_width], al

call newline

; input stripe height
call find_current_cursor_position

mov ax, [add2]
 mov es, ax
mov bh, [page]
 mov bl, 07h
mov cx, stripe_height_prompt_length

mov ax, stripe_height_prompt
add ax, word [add1]
 mov bp, ax
```

```
mov ax, 1301h
int 10h

mov byte [result], 0
call clear_buffer
call read_buffer

mov al, [result]
mov byte [stripe_height], al

call newline


; input stripe indent
call find_current_cursor_position

mov ax, [add2]
mov es, ax
mov bh, [page]
mov bl, 07h
mov cx, stripe_indent_prompt_length

mov ax, stripe_indent_prompt
add ax, word [add1]
```

```
mov bp, ax

mov ax, 1301h
int 10h

mov byte [result], 0
call clear_buffer
call read_buffer

mov al, [result]
mov byte [stripe_indent], al

call newline
call draw_colorful_line

; read character
mov ah, 00h
int 16h

call change_page_number
jmp menu

jmp end
```

```
reboot:
    call change_page_number

    ; set text video mode
    mov ah, 00h
    mov al, 2
    int 10h

    jmp 0000h:7c00h


read_buffer:
    read_char:
        ; read character
        mov ah, 00h
        int 16h

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

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

```
        je handle_backspace

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

        ; display character as TTY
        mov ah, 0eh
        mov bl, 07h
        int 10h

        jmp read_char

    handle_enter:
        mov byte [si], 0
        mov si, buffer
        call convert_input_int
        jmp end_read_buffer

    handle_backspace:
        call find_current_cursor_position
```

```
    cmp byte [char_counter], 0
    je read_char

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

    ; move cursor to the left
    mov ah, 02h
    mov bh, 0
    dec dl
    int 10h

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

    jmp read_char

end_read_buffer:
```

```
    ret


clear_buffer:
    mov byte [char_counter], 0
    mov byte [si], 0
    mov si, buffer

    ret


draw_colorful_line:
    ; set graphic video mode
    mov ah, 00h
    mov al, [video_mode]
    int 10h

    mov al, byte [stripe_height]
    mov byte [stripes], al
    mov byte [pixel_color], 14
    call draw_stripe

    dec byte [stripe_width]
    dec byte [stripe_height]
```

```
mov al, byte [stripe_height]
mov byte [stripes], al
mov byte [pixel_color], 20
call draw_stripe

dec byte [stripe_width]
dec byte [stripe_height]
mov al, byte [stripe_height]
mov byte [stripes], al
mov byte [pixel_color], 1
call draw_stripe

dec byte [stripe_width]
dec byte [stripe_height]
mov al, byte [stripe_height]
mov byte [stripes], al
mov byte [pixel_color], 3
call draw_stripe

dec byte [stripe_width]
dec byte [stripe_height]
mov al, byte [stripe_height]
mov byte [stripes], al
mov byte [pixel_color], 13
```

```
    call draw_stripe

    dec byte [stripe_width]
    dec byte [stripe_height]
    mov al, byte [stripe_height]
    mov byte [stripes], al
    mov byte [pixel_color], 28
    call draw_stripe

    dec byte [stripe_width]
    dec byte [stripe_height]
    mov al, byte [stripe_height]
    mov byte [stripes], al
    mov byte [pixel_color], 2
    call draw_stripe

    ret


draw_stripe:

    stripe_loop:
        mov al, byte [stripe_width]
        mov byte [line_length], al
```

```
        mov al, byte [stripe_indent]
        mov byte [left_indent], al
        mov cx, [left_indent]
        call draw_line

        cmp byte [stripes], 0
        je end_stripe_loop

        dec byte [stripes]
        jmp stripe_loop

    end_stripe_loop:

    ret


draw_line:

    draw_pixel:
        mov ah, 0ch
        mov bh, byte [page]
        mov al, [pixel_color]
        mov dx, [line_number]
```

```
        int 10h

        inc cx

        dec byte [line_length]
        cmp byte [line_length], 0
        jne draw_pixel

    inc word [line_number]

    ret


convert_input_int:
    xor ax, ax
    xor bx, bx

    convert_digit:
        lodsb

        sub al, '0'
        xor bh, bh
        imul bx, 10
        add bl, al
```

```
        mov [result], bl

        dec byte [char_counter]
        cmp byte [char_counter], 0
        jne convert_digit

    ret


change_page_number:
    inc byte [page]
    mov ah, 05h
    mov al, [page]
    int 10h

    ret


find_current_cursor_position:
    mov ah, 03h
    mov bh, byte [page]
    int 10h

    ret
```

```
newline:
    call find_current_cursor_position

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

    ret

end:



section .data
      disclaimer db "Welcome to the rainbow command! Remember, the page has the size 320x200!"
    disclaimer_length equ 72

     reboot_prompt db "Press r to reboot or any other key to continue: "
    reboot_prompt_length equ 47
```

```
    stripe_width_prompt db "Stripe width: "
    stripe_width_prompt_length equ 14

    stripe_indent_prompt db "Stripe indent: "
    stripe_indent_prompt_length equ 15

    stripe_height_prompt db "Stripe height: "
    stripe_height_prompt_length equ 15


section .bss
    video_mode resb 1
    pixel_color resb 1

    line_length resb 1
    stripe_width resb 1

    left_indent resb 2
    stripe_indent resb 2

    line_number resb 2

    stripes resb 1
    stripe_height resb 2
```

```
    char_counter resb 1
    result resb 1

    page resb 1
    c resb 1

    add1 resb 2
    add2 resb 2
    buffer resb 100
```

***bootloader.asm***

```
org 7d00h

mov byte [page_number], 0
jmp main

main:
    mov byte [marker], 0

    ; print initial prompt
    mov si, prompt
    call print

    ; read character
```

```
mov ah, 00h
int 16h

call newline

mov si, hts_prompt
call print
call newline


; print sector count prompt
mov ah, 0eh
mov al, '>'
mov bl, 07h
int 10h

mov byte [result], 0
call clear
call read_buffer

mov al, [result]
mov byte [sc], al

call newline
```

```
; print head prompt
mov ah, 0eh
mov al, '>'
mov bl, 07h
int 10h

mov byte [result], 0
call clear
call read_buffer

mov al, [result]
mov byte [h], al

call newline


; print track prompt
mov ah, 0eh
mov al, '>'
mov bl, 07h
int 10h
```

```
mov byte [result], 0
call clear
call read_buffer

mov al, [result]
mov byte [t], al

call newline


; print sector prompt
mov ah, 0eh
mov al, '>'
mov bl, 07h
int 10h

mov byte [result], 0
call clear
call read_buffer

mov al, [result]
mov byte [s], al

call newline
```

```
call newline

inc byte [marker]

; print ram address prompt
mov si, so_prompt
call print
call newline

; print segment prompt
mov ah, 0eh
mov al, '>'
mov bl, 07h
int 10h

call clear
call read_buffer

mov ax, [hex_result]
mov [add1], ax

call newline

; print offset prompt
```

```
mov ah, 0eh
mov al, '>'
mov bl, 07h
int 10h

call clear
call read_buffer

mov ax, [hex_result]
mov [add2], ax

call newline

call load_kernel

; print a prompt to load the kernel
mov si, kernel_start
call newline
call print

; read option
mov ah, 00h
int 16h
```

```
    ; display character as TTY
    mov ah, 0eh
    mov bl, 07h
    int 10h

    call newline
    call newline

    ; remember segment and offset in ax:bx
    mov ax, [add1]
    mov bx, [add2]

    ; jump to the loaded NASM script
    add ax, bx
    jmp ax


load_kernel:
    mov ah, 0h
    int 13h

    mov ax, [add2]
    mov es, ax
    mov bx, [add1]
```

```
    ; load the NASM script into memory
    mov ah, 02h
    mov al, [sc]
    mov ch, [t]
    mov cl, [s]
    mov dh, [h]
    mov dl, 0

    int 13h

    ; print error code
    mov al, '0'
    add al, ah
    mov ah, 0eh
    int 10h

    call newline

    ret


read_buffer:
    mov byte [c], 0
```

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

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

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

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

        ; display character as TTY
        mov ah, 0eh
        mov bl, 07h
        int 10h
```

```
    jmp read_char

hdl_enter:
    cmp byte [c], 0
    je tomain

    mov byte [si], 0
    mov si, buffer

    cmp byte [marker], 0
    je atoi_jump
    jmp atoh_jump

hdl_backspace:
    call cursor

    cmp byte [c], 0
    je read_char

    ; clear last buffer char
    dec si
    dec byte [c]
```

```
    ; move cursor to the left
    mov ah, 02h
    mov bh, 0
    dec dl
    int 10h

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

    jmp read_char

atoi_jump:
    call atoi
    jmp end_read_buffer

atoh_jump:
    call atoh
    jmp end_read_buffer

end_read_buffer:
```

```
    ret


tomain:
    call change_page_number
    jmp main


atoi:
    xor ax, ax
    xor bx, bx

    atoi_d:
        lodsb

        sub al, '0'
        xor bh, bh
        imul bx, 10
        add bl, al
        mov [result], bl

        dec byte [c]
        cmp byte [c], 0
```

```
        jne atoi_d

    ret


atoh:
    xor bx, bx
    mov di, hex_result

    atoh_s:
        xor ax, ax
        mov al, [si]

        cmp al, 65
        jg atoh_l
        sub al, 48
        jmp continue

        atoh_l:
            sub al, 55
            jmp continue

        continue:
            mov bx, [di]
```

```
            imul bx, 16
            add bx, ax
            mov [di], bx

            inc si

        dec byte [c]
        jnz atoh_s

    ret


print:
    call cursor

    print_char:
        mov al, [si]
        cmp al, '$'
        je end_print

        mov ah, 0eh
        int 10h
        inc si
        jmp print_char
```

```
    end_print:
        ret


clear:
    mov byte [c], 0
    mov byte [si], 0
    mov si, buffer

    ret


cursor:
    mov ah, 03h
    mov bh, 0
    int 10h

    ret


change_page_number:
    inc byte [page_number]
    mov ah, 05h
```

```
    mov al, [page_number]
    int 10h

    ret



newline:
    call cursor

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

    ret



section .data:
     prompt db 'Welcome to the "Rainbow", Alexei! Press any key to draw a colorful flag: $'
    hts_prompt db "Input NumberHeadTrackSector script$"
    so_prompt db "Type OFF:SEGM RAM$"
     kernel_start db "Press any key to load the kernel: $"
```

```
    sc db 0
    h db 0
    t db 0
    s db 0

    c db 0
    result db 0

    page_number db 0
    marker db 0


section .bss:
    hex_result resb 2
    add1 resb 2
    add2 resb 2
    buffer resb 2

dw 0AA55h
```

***mini_boot.asm***

```
org 7c00h

mov ah, 00
```

```
int 13h

mov ax, 0000h
mov es, ax
mov bx, 7d00h

mov ah, 02h
mov al, 2
mov ch, 0
mov cl, 2
mov dh, 0
mov dl, 0
int 13h

jmp 0000h:7d00h

times 510-($-$$) db 0
dw 0AA55h
```

***boot_floppy_script.sh***

```
#!/bin/bash

binary_file="colorflag.bin"
bootloader="bootloader.asm"
```

```
mini_boot="mini_boot.asm"


# Check if the binary file exists
if [ ! -f "$binary_file" ]; then
     echo "Error: Binary file '$binary_file' does not exist."
    exit 1
fi


nasm -f bin $mini_boot -o mini_boot.bin
nasm -f bin $bootloader -o bootloader.bin


# Create an empty floppy disk image (1.44MB size)
floppy_image="floppy.img"
truncate -s 1474560 mini_boot.bin
mv mini_boot.bin $floppy_image


dd if="bootloader.bin" of="$floppy_image" bs=512 seek=1 conv=notrunc
dd if="$binary_file" of="$floppy_image" bs=512 seek=3 conv=notrunc


echo "Binary file '$binary_file' successfully added to floppy image '$floppy_image'."
```

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

```
Welcome to the "Rainbow", Alexei! Press any key to draw a colorful flag:
Input NumberHeadTrackSector script
>2
>0
>0
>4

Type OFF:SEGM RAM
>8000
>0000
0

Press any key to load the kernel:
```

**Figure 1.** Initiating kernel

```
Welcome to the rainbow command! Remember, the page has the size 320x200!
Press r to reboot or any other key to continue:
Stripe width: 250
Stripe height: 15
Stripe indent: 15_
```

**Figure 2.** Start of the program loaded

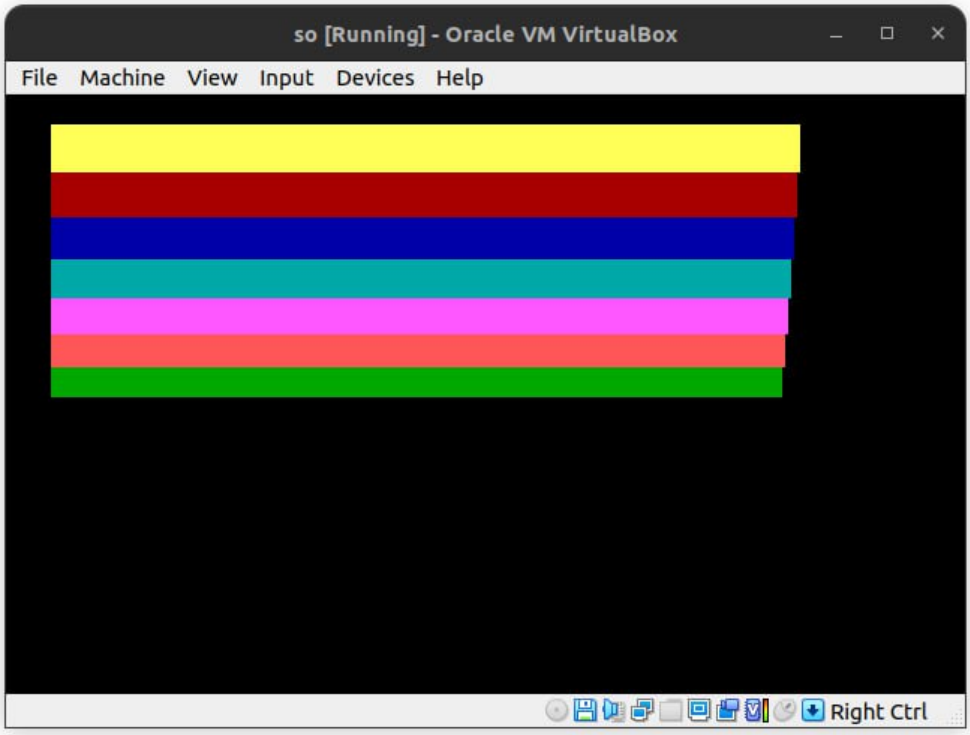


**Figure 3.** Output of the program

**Conclusions:**

During this laboratory work, our exploration delved into the development of a boot loader for a kernel using NASM. Our emphasis was on delving into low-level programming to initiate system initialization and smoothly transfer control to the kernel. Importantly, we acquired the skill of crafting bootable disk images through the utilization of Linux shell commands, enhancing the efficiency of the development workflow. This hands-on encounter serves as a cornerstone for comprehending the intricacies of operating system bootstrapping and the creation of practical image files.