Ministry of Education, Culture and Research of the Republic of Moldova
Technical University of Moldova
Department of Software and Automation Engineering

REPORT

Laboratory work no. 3
Floppy Disk I/O operations

Elaborated:	
st. gr. FAF-213	Botnari Ciprian
st. gr. FAF-213	Guțu Dinu
st. gr. FAF-213	Iațco Sorin

Verified:	
lect. univ.	Rostislav Călin

Chişinău – 2023

Table of Contents

Topic: Floppy Disk I/O operations

Tasks

1. Write at first and last sector on floppy disk for each student in the following format: @@@FAF-21* Prenume NUME###. This string needs to be duplicated 10 times without additional characters.

2. Write a program in Assembly with the following functions:

    a. **(KEYBOARD ==> FLOPPY)**: Read from keyboard a **string** with maximum 256 characters (backspace should work) and write this string **N** times at address **{Head, Track, Sector},** where **N** can be between 1 and 30000. After pressing ENTER, if the length of the string is greater than 0, a new line needs to be displayed and the previous typed string. **N, Head, Track, Sector** should be read from the keyboard input. After the write operation is completed, the error code should be displayed.

    b. **(FLOPPY ==> RAM)**: Read from floppy **N** sectors at address **{Head, Track, Sector},** and transfer this data in **RAM** at address **{XXXX:YYYY}.** After the read operation is completed, the error code should be displayed. After this all contents located at address **{XXXX:YYYY}** should be printed to the screen. If the volume of data exceeds 1 page, pagination needs to be implemented by pressing SPACE key. **N, Head, Track, Sector, address {XXXX:YYYY}** should be read from the keyboard input.

    c. **(RAM ==> FLOPPY)**: Write to floppy **Q** bytes at address **{Head, Track, Sector}** from **RAM** at address **{XXXX:YYYY}.** Data block of **Q** bytes should be displayed and after the write operation to floppy, the error code should be displayed.

Requirements

- After executing any of the functions, program should be ready for another procedure.
- The compiled code shouldn't exceed 512 bytes. Otherwise, it is necessary to implement a workaround for this restriction and finally create a bootable disk image that works in Virtual Box.

Code

NASM Version: 2.16.01

This shell script automates the process of compiling assembly code, adding the bootloader, creating a bootable floppy image, and configuring a VirtualBox virtual machine to use this disk image and finally the virtual machine is then started.

Build script

```
#!/bin/bash
if [ $# -ne 1 ]; then
  echo "Usage: $0 <filename.asm>"
  exit 1
fi

filename_with_extension="$1"
filename="${filename_with_extension%.*}"

bootloader_file="bootloader.asm"
```

```
asm_file="$filename_with_extension"
com_file="$filename.com"
flp_file="$filename.flp"

# Step 1: Compile the assembly code to a .com file
nasm -f bin -o "$com_file" "$asm_file"
if [ $? -ne 0 ]; then
  echo "Compilation failed. Check your assembly code."
  exit 1
fi
echo "Step 1: Compilation completed."

# Step 2: Compile the bootloader code to a .com file
nasm -f bin -o "bootloader.com" "$bootloader_file"
if [ $? -ne 0 ]; then
  echo "Compilation failed. Check your bootloader code."
  exit 1
fi
echo "Step 2: Compilation of boatloader completed."
cat "bootloader.com" "$com_file" > "$flp_file"

# Step 3: Resize the .flp file to 1.44MB
truncate -s 1474560 "$flp_file"
echo "Step 3: Resized $flp_file to 1.44MB."

# Step 4: Close VirtualBox
VM_NAME="BestOS"
VBoxManage controlvm "$VM_NAME" poweroff
echo "Virtual Machine $VM_NAME closed."

sleep 3
# Step 5: Change the storage to $flp_file in VirtualBox
VBoxManage storageattach "$VM_NAME" --storagectl "Floppy" --port 0 --
device 0 --type fdd --medium "$flp_file"
echo "Step 5: Storage in VirtualBox changed to $flp_file."

# Step 6: Start the Virtual Machine
VBoxManage startvm "$VM_NAME"
echo "Step 6: Virtual Machine $VM_NAME started."

echo "All steps completed successfully."
```

Task 1

Here is the floppy space distribution for our team.

Nr	Group	Student	Block	Start	End	Bytes
6	FAF-213	Botnari Ciprian	66	1951	1980	15360
19	FAF-213	Guțu Dinu	79	2341	2370	15360
21	FAF-213	Iațco Sorin	81	2401	2430	15360

Table 1. Floppy space distribution

We applied the following formula to write to floppy:

$$(sector_{number} - 1440)\ /\ 18\ = n.d$$

$$head = \begin{cases} 0, & sector_{number} < 1440 \\ 1, & sector_{number} > 1440 \end{cases}$$

$$where\ 1440 -\ half\ of\ total\ sectors, 18 - total\ tracks, n - track, d - sector$$

Let's apply the formula for $sector_{number} = 1951$

$$(1951 - 1440)\ /\ 18\ = 28.3$$

Thus we obtain the following results:

- Track = 28
- Sector = 3
- Head = 1, since $sector_{number} = 1951 < 1440$

ciprian.asm

```
org 0x7c00

section .data
    message: times 10 db "@@@FAF-213 Ciprian BOTNARI###"

section .text
    global _start

_start:
    ; First sector
    mov ah, 03h          ; Function code for write sectors
    mov al, 1            ; Number of sectors to write
    mov ch, 28           ; Track number
    mov cl, 3            ; Sector number
    mov dh, 1            ; Head number
    mov bx, message      ; Pointer to the string
    int 13h              ; BIOS interrupt

    ; Last sector
    mov ah, 03h          ; Function code for write sectors
    mov al, 1            ; Number of sectors to write
    mov ch, 30           ; Track number
    mov cl, 1            ; Sector number
    mov dh, 1            ; Head number
    mov bx, message      ; Pointer to the string
    int 13h              ; BIOS interrupt

    mov ah, 4ch          ; Function code for program termination
    int 21h              ; DOS interrupt
```

sorin.asm

```
org 0x7c00

section .data
    message: times 10 db "@@@FAF-213 Sorin IATCO###"

section .text
    global _start
```

```
_start:
    ; First sector
    mov ah, 03h          ; Function code for write sectors
    mov al, 1            ; Number of sectors to write
    mov ch, 53           ; Track number
    mov cl, 3            ; Sector number
    mov dh, 1            ; Head number
    mov bx, message      ; Pointer to the string
    int 13h              ; BIOS interrupt

    ; Last sector
    mov ah, 03h          ; Function code for write sectors
    mov al, 1            ; Number of sectors to write
    mov ch, 55           ; Track number
    mov cl, 1            ; Sector number
    mov dh, 1            ; Head number
    mov bx, message      ; Pointer to the string
    int 13h              ; BIOS interrupt

    mov ah, 4ch          ; Function code for program termination
    int 21h              ; DOS interrupt
```

dinu.asm

```
org 0x7c00

section .data
    message: times 10 db "@@@FAF-213 Dinu GUTU###"

section .text
    global _start

_start:
    ; First sector
    mov ah, 03h          ; Function code for write sectors
    mov al, 1            ; Number of sectors to write
    mov ch, 50           ; Track number
    mov cl, 1            ; Sector number
    mov dh, 1            ; Head number
    mov bx, message      ; Pointer to the string
    int 13h              ; BIOS interrupt

    ; Last sector
    mov ah, 03h          ; Function code for write sectors
    mov al, 1            ; Number of sectors to write
    mov ch, 51           ; Track number
    mov cl, 6            ; Sector number
    mov dh, 1            ; Head number
    mov bx, message      ; Pointer to the string
    int 13h              ; BIOS interrupt

    mov ah, 4ch          ; Function code for program termination
    int 21h              ; DOS interrupt
```

Task 2

bootloader.asm

```
org 7c00h

mov ah, 0h
int 13h

mov ax, 0000h
mov es, ax
mov bx, 1000h

mov ah, 02h
mov al, 3
mov ch, 0
mov cl, 2
mov dh, 0
mov dl, 0

int 13h

jmp 0000h:1000h

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

print_string.asm

```
print_string_si:
    push ax

    mov ah, 0x0e
    call print_next_char

    pop ax
    ret

print_next_char:
    mov al, [si]
    cmp al, 0

    jz if_zero

    int 0x10
    inc si

    jmp print_next_char

if_zero:
    ret
```

str_compare.asm

```
compare_strs_si_bx:
    push si
    push bx
    push ax
comp:
    mov ah, [bx]
    cmp [si], ah
    jne not_equal

    cmp byte [si], 0
    je first_zero

    inc si
    inc bx

    jmp comp

first_zero:
    cmp byte [bx], 0
    jne not_equal

    mov cx, 1

    pop si
    pop bx
    pop ax
    ret
not_equal:
    mov cx, 0

    pop si
    pop bx
    pop ax

    ret
```

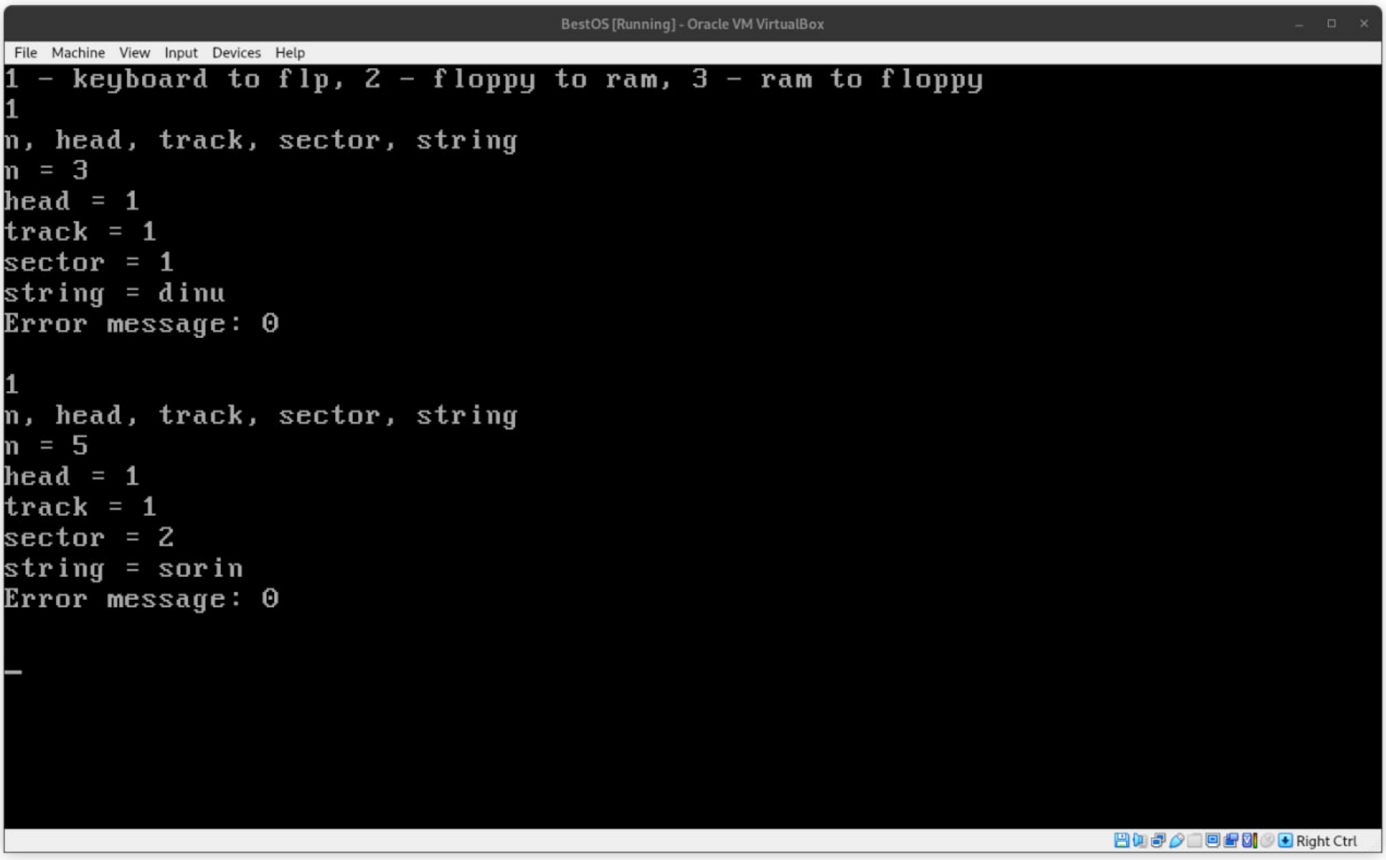
main.asm

```
org 1000h                ; Set the origin of the program to 1000h
bits 16                  ; Set the code generation to 16-bit

jmp start

%include "print_string.asm"
%include "str_compare.asm"

start:                   ; Start of the program
    mov ah, 0x00         ; Set AH register for video services
    mov al, 0x03         ; Set AL register for text mode
    int 0x10             ; Call BIOS video interrupt

    mov sp, 1000h        ; Set the stack pointer
    ; Reset all variables
    mov byte [n], 0
    mov byte [head], 0
    mov byte [track], 0
    mov byte [sector], 0
    mov word [ram_start], 0
    mov word [ram_end], 0
    mov byte [var_flag], 0
    mov byte [ram_flag], 0
    mov byte [q_flag], 0
    mov byte [ram_success], 0
    call clear_buffer

    mov si, help_desc     ; Load the address of help_desc into SI
    call print_string_si

mainloop:                 ; Main loop label
    call get_input
    jmp mainloop

get_input:               ; Subroutine to get user input
    mov bx, 0            ; Initialize BX register

input_processing:        ; Label for input processing loop
    mov ah, 0x0          ; Set AH register for keyboard services
    int 0x16             ; Call BIOS keyboard interrupt

    cmp al, 0x3          ; Compare input with Ctrl+C
    je start

    cmp al, 0x0d          ; Compare input with Enter key
    je check_the_input

    cmp al, 0x8           ; Compare input with Backspace
    je backspace_pressed

    mov ah, 0x0e          ; Set AH register for teletype output
    int 0x10              ; Call BIOS video interrupt

    mov [input+bx], al    ; Store the input character in the buffer
    inc bx                ; Increment the buffer index

    cmp bx, 255           ; Check if the buffer is full
    je check_the_input

    jmp input_processing

backspace_pressed:       ; Subroutine for processing Backspace key
    cmp bx, 0             ; Check if the buffer is empty
    je input_processing

    mov ah, 0x0e          ; Set AH register for teletype output
    int 0x10              ; Call BIOS video interrupt
    mov al, ' '           ; Print a space to erase the character
    int 0x10              ; Call BIOS video interrupt

    mov al, 0x8           ; Move the cursor back (Backspace)
    int 0x10              ; Call BIOS video interrupt

    dec bx                ; Decrement the buffer index
    mov byte [input+bx], 0 ; Set the removed character to null

    jmp input_processing

check_the_input:         ; Label for checking the input
    inc bx                ; Increment the buffer index
    mov byte [input+bx], 0 ; Set the end of the string

    mov si, new_line      ; Load the address of new_line into SI
    call print_string_si

    ; Q processing
    cmp byte [q_flag], 1  ; Check if Q flag is set
    je q_processing

    ; RAM processing
    cmp byte [ram_flag], 2 ; Check if RAM flag is set to 2
    je segment_processing

    cmp byte [ram_flag], 3 ; Check if RAM flag is set to 3
    je address_processing


    ; Option 1 processing
    cmp byte [var_flag], 1 ; Check if var_flag is set to 1
    je n_processing

    cmp byte [var_flag], 2 ; Check if var_flag is set to 2
    je head_processing

    cmp byte [var_flag], 3 ; Check if var_flag is set to 3
    je track_processing

    cmp byte [var_flag], 4 ; Check if var_flag is set to 4
    je sector_processing

    cmp byte [var_flag], 5 ; Check if var_flag is set to 5
    je string_processing

    mov si, help_command   ; Load the address of help_command into SI
    mov bx, input          ; Load the address of input into BX
    call compare_strs_si_bx
    cmp cx, 1              ; Compare the result of string comparison
    je equal_help

    ; Option 1
    mov si, option_1       ; Load the address of option_1 into SI
    mov bx, input          ; Load the address of input into BX
    call compare_strs_si_bx ; Compare strings in SI and BX
    cmp cx, 1              ; Compare the result of string comparison
    je equal_option_1

    ; Option 2
    mov si, option_2       ; Load the address of option_2 into SI
    mov bx, input          ; Load the address of input into BX
    call compare_strs_si_bx
    cmp cx, 1              ; Compare the result of string comparison
    je equal_option_2

    ; Option 3
    mov si, option_3       ; Load the address of option_3 into SI
    mov bx, input          ; Load the address of input into BX
    call compare_strs_si_bx
    cmp cx, 1              ; Compare the result of string comparison
    je equal_option_3

    cmp cx, 0              ; Compare the result of string comparison
    je equal_random_string

equal_help:              ; Label for equal help strings
    mov si, help_desc      ; Load the address of help_desc into SI
    call print_string_si

    jmp done

equal_option_1:          ; Label for equal option_1 strings
    mov si, variables_1    ; Load the address of variables_1 into SI
    call print_string_si
    mov si, n_prompt       ; Load the address of n_prompt into SI
    call print_string_si

    inc byte [var_flag]    ; Increment var_flag
    jmp done

n_processing:            ; Label for processing n input
    call convert_input_int
    mov al, [result]      ; Load the result into AL register
    mov [n], al           ; Store the result in the variable n

    mov si, head_prompt    ; Load the address of head_prompt into SI
    call print_string_si

    inc byte [var_flag]    ; Increment var_flag
    jmp done

head_processing:         ; Label for processing head input
    call convert_input_int ; Call subroutine to convert input to integer
    mov al, [result]      ; Load the result into AL register
    mov [head], al        ; Store the result in the variable head

    mov si, track_prompt   ; Load the address of track_prompt into SI
    call print_string_si

    inc byte [var_flag]    ; Increment var_flag
    jmp done

track_processing:        ; Label for processing track input
    call convert_input_int
    mov al, [result]      ; Load the result into AL register
    mov [track], al       ; Store the result in the variable track

    mov si, sector_prompt  ; Load the address of sector_prompt into SI
    call print_string_si

    inc byte [var_flag]    ; Increment var_flag
    jmp done

sector_processing:       ; Label for processing sector input
    call convert_input_int
    mov al, [result]      ; Load the result into AL register
    mov [sector], al      ; Store the result in the variable sector

    cmp byte [ram_flag], 1 ; Check if RAM flag is set to 1
    je ram_processing

    mov si, string_prompt  ; Load the address of string_prompt into SI
    call print_string_si

    inc byte [var_flag]    ; Increment var_flag
    jmp done

ram_processing:          ; Label for processing RAM input
    mov si, ram_start_prompt
    call print_string_si

    inc byte [ram_flag]    ; Increment ram_flag
    jmp done

segment_processing:     ; Label for processing segment input
    mov si, ram_start      ; Load the address of ram_start into SI
    call read_address_process_input

    mov si, ram_end_prompt ; Load the address of ram_end_prompt into SI
    call print_string_si

    inc byte [ram_flag]    ; Increment ram_flag
    jmp done

address_processing:     ; Label for processing address input
    mov si, ram_end        ; Load the address of ram_end into SI
    call read_address_process_input

    mov si, new_line       ; Load the address of new_line into SI
    call print_string_si
    cmp byte [q_flag], 2  ; Compare q_flag with 2
    je ram_to_floppy

    jmp read_floppy

read_address_process_input: ; Subroutine for processing address input
   mov di, input           ; Load the address of input into DI

address_processing_input: ; Label for processing address input loop
   cmp di, input + 4       ; Compare DI with the end of the address input
   je address_processing_input_done

   mov al, [di + 2]        ; Load the high byte of the address
   shl al, 4               ; Shift it left by 4 bits
   or al, [di + 3]         ; OR it with the low nibble of the high byte
   mov ah, [di]            ; Load the low byte of the address
   shl ah, 4               ; Shift it left by 4 bits
   or ah, [di + 1]         ; OR it with the low nibble of the low byte
   mov word [si], ax   ; Store the 16-bit result in the destination address
   add di, 4               ; Move to the next 4 bytes
   add si, 2               ; Move to the next 2 bytes
   inc bl                  ; Increment a counter (not used)

   jmp address_processing_input

address_processing_input_done:
    ret

string_processing:        ; Label for processing string input
    jmp fill_write_buffer

equal_option_2:           ; Label for equal option_2 strings
    mov si, variables_2    ; Load the address of variables_2 into SI
    call print_string_si
    mov si, n_prompt       ; Load the address of n_prompt into SI
    call print_string_si

    inc byte [ram_flag]    ; Increment ram_flag
    inc byte [var_flag]    ; Increment var_flag

    jmp done

equal_option_3:
    mov si, variables_3       ; Set SI to point to variables description
    call print_string_si
    mov si, q_prompt          ; Set SI to point to "q = "
    call print_string_si

    inc byte [q_flag]

    jmp done

q_processing:
    call convert_input_int
    mov al, [result]           ; Move the result to AL
    mov [q], al                ; Store the result in q

    mov si, head_prompt        ; Set SI to point to "head = "
    call print_string_si

    inc byte [var_flag]         ; Increment var_flag
    inc byte [var_flag]         ; Increment var_flag again
    inc byte [ram_flag]         ; Increment ram_flag
    inc byte [q_flag]           ; Increment q_flag

    jmp done

equal_random_string:
    mov si, new_line            ; Set SI to point to a new line
    call print_string_si       ; Print a new line

    mov si, input               ; Set SI to point to the input buffer
    call print_string_si       ; Print the contents of the input buffer

    mov si, new_line            ; Set SI to point to a new line
    call print_string_si       ; Print a new line

    jmp done

done:
    cmp bx, 0                   ; Compare buffer index with 0
    je exit


    dec bx                      ; Decrement buffer index
    mov byte [input+bx], 0      ; Null-terminate the input string

    jmp done

exit:
    ret

convert_input_int:
    mov si, input               ; Set SI to point to the input buffer
    mov byte [result], 0        ; Clear the result variable
    xor ax, ax                  ; Clear AX register
    xor cx, cx                  ; Clear CX register

    next_digit:
        lodsb             ; Load byte at address SI into AL, increment SI
        cmp al, 0               ; Check for end of string
        je stop
        sub al, '0'             ; Convert from ASCII to number
        movzx ax, al            ; Zero-extend AL into AX
        imul cx, 10             ; Multiply CX by 10
        add cx, ax              ; Add AX to CX
        add [result], cx        ; Add CX to result
        jmp next_digit
    stop:
        ret

fill_write_buffer:
    mov si, input    ; Move address of 'input' to source index register
    mov di, floppy_buffer ; Move address to destination index register
    xor ax, ax                 ; Clear AX register
    xor bx, bx                 ; Clear BX register

    loop_buffer:
        cmp ax, 512             ; Compare the value in AX with 512
        je write_to_floppy
        cmp byte [n], 0      ; Compare the value at memory location 'n'
        je write_to_floppy

        mov bl, byte [si]
        mov byte [di], bl

        inc ax                  ; Increment the value in AX
        inc si                  ; Increment the value in SI
        inc di                  ; Increment the value in DI

        cmp byte [si], 0
        jne loop_buffer
        mov si, input           ; Move the address of 'input' to SI
        dec byte [n]           ; Decrement the byte at the address in 'n'

        jmp loop_buffer

clear_buffer:
    cmp byte [di], 0
    je done

    mov byte [di], 0            ; Move 0 to the byte at the address in DI
    inc di                      ; Increment the value in DI
    cmp di, floppy_buffer + 512
    je done

    jmp clear_buffer

write_to_floppy:
    ; set the address of the first sector to write
    mov ah, 03h                 ; Set AH register to 3 (disk write)
    mov al, 1                   ; (number of sectors to write)
    mov ch, [track]             ; track
    mov cl, [sector]            ; sector
    mov dl, 0                  ; Set DL register to 0 (floppy disk drive)
    mov dh, [head]              ; head
    mov bx, floppy_buffer
    int 13h                     ; Call BIOS interrupt 13h

    mov si, error_message
    call print_string_si
    ; print error code
    mov al, '0'                 ; Move the ASCII value of '0' to AL
    add al, ah                  ; Add the value in AH to AL
    mov ah, 0eh                ; Set AH register to 0eh (teletype output)
    int 10h                     ; Call BIOS interrupt 10h

    mov si, new_line            ; Move the address of 'new_line' to SI
    call print_string_si

    mov si, new_line            ; Move the address of 'new_line' to SI
    call print_string_si

    mov byte [var_flag], 0

    jmp clear_buffer

read_floppy:
    mov ah, 02h                 ; Set AH register to 2 (disk read)
    mov al, [n]
    mov ch, [track]            ; track
    mov cl, [sector]           ; sector
    mov dl, 0                  ; Set DL register to 0 (floppy disk drive)
    mov dh, [head]             ; head
    mov bx, [ram_start]
    mov es, bx
    mov bx, [ram_end]
    int 13h                     ; Call BIOS interrupt 13h


    mov si, new_line            ; Move the address of 'new_line' to SI
    call print_string_si

    mov si, error_message
    call print_string_si

    ; print error code
    mov al, '0'                 ; Move the ASCII value of '0' to AL
    add al, ah                  ; Add the value in AH to AL
    mov [ram_success], al
    mov ah, 0eh                ; Set AH register to 0eh (teletype output)
    int 10h                     ; Call BIOS interrupt 10h

    mov byte [ram_flag], 0
    mov byte [var_flag], 0

    cmp byte [ram_success], 0
    jne print_ram

    cmp byte [ram_success], 0
    je print_fail_statement

print_ram:
    call clear_screen
    mov si, success_ram         ; Move the address of 'success_ram' to SI
    call print_string_si

    call print_ram_volume

    mov si, new_line            ; Move the address of 'new_line' to SI
    call print_string_si

    jmp done

print_fail_statement:
    call clear_screen
    mov si, fail_ram            ; Move the address of 'fail_ram' to SI
    call print_string_si

    jmp done

clear_screen:
    mov ax, 0x0003
    int 0x10                    ; Call BIOS interrupt 0x10
    ret

print_ram_volume:
    mov ax, 0x1301          ; (BIOS function to write text to the screen)
    mov bx, [ram_start]
    mov es, bx                ; Move the value in BX to ES (Extra Segment)
    mov bx, 0x0007        ; Set BX register to 0x0007 (attribute for text)
    mov cx, 512             ; (number of characters to print)
    mov bp, [ram_end]
    int 0x10                    ; Call BIOS interrupt 0x10

    ret

ram_to_floppy:
    xor dx, dx                  ; Clear DX register
    mov ax, [q]
    mov cx, 512                 ; Set CX register to 512
    div cx               ; Divide AX by CX, result in AX, remainder in DX

    cmp dx, 0                    ; Compare the value in DX with 0
    jne ram_copy_interrupt
    dec ax                       ; Decrement the value in AX

ram_copy_interrupt:
    mov ah, 03h                 ; Set AH register to 3 (disk write)
    mov al, 1                   ; (number of sectors to write)
    mov ch, [track]             ; track
    mov cl, [sector]            ; sector
    mov dl, 0                  ; Set DL register to 0 (floppy disk drive)
    mov dh, [head]
   mov es, [ram_start]         ; (Extra Segment)
    mov bx, [ram_end]
    int 13h                     ; Call BIOS interrupt 13h
    mov si, new_line            ; Move the address of 'new_line' to SI
    call print_string_si

    mov si, error_message
    call print_string_si

    ; print error code
    mov al, '0'                 ; Move the ASCII value of '0' to AL
    add al, ah                  ; Add the value in AH to AL
    mov ah, 0eh                ; Set AH register to 0eh (teletype output)
    int 10h                     ; Call BIOS interrupt 10h

    mov byte [ram_flag], 0
    mov byte [var_flag], 0
    mov byte [q_flag], 0   ; Move 0 to the byte at the address in 'q_flag'

    mov si, new_line            ; Move the address of 'new_line' to SI
    call print_string_si

    jmp clear_buffer

; Data section
help_desc: db "1 - keyboard to flp, 2 - floppy to ram, 3 - ram to floppy",
0x0d, 0xa, 0
variables_1: db "n, head, track, sector, string", 0x0d, 0xa, 0
variables_2: db "n, head, track, sector, start, end", 0x0d, 0xa, 0
variables_3: db "q, head, track, sector, start, end", 0x0d, 0xa, 0
q_prompt: db "q = ", 0
n_prompt: db "n = ", 0
head_prompt: db "head = ", 0
track_prompt: db "track = ", 0
sector_prompt: db "sector = ", 0
string_prompt: db "string = ", 0
ram_start_prompt: db "start addr = ", 0
ram_end_prompt: db "end addr = ", 0

goodbye: db 0x0d, 0xa, "Exiting...", 0x0d, 0xa, 0
help_command: db "help", 0
option_1: db "1", 0
option_2: db "2", 0
option_3: db "3", 0
success_ram: db "Successfully wrote to RAM", 0
fail_ram: db "Failed to write to RAM", 0
error_message: db "Error message: ", 0

new_line: db 0x0d, 0xa, 0

q: db 0
n: db 0
head: db 0
track: db 0
sector: db 0
ram_start: dw 0
```

```
ram_end: dw 0
var_flag: db 0
ram_flag: db 0
q_flag: db 0
result: db 0
ram_success: db 0

floppy_buffer: times 512 db 0
input: times 256 db 0
```

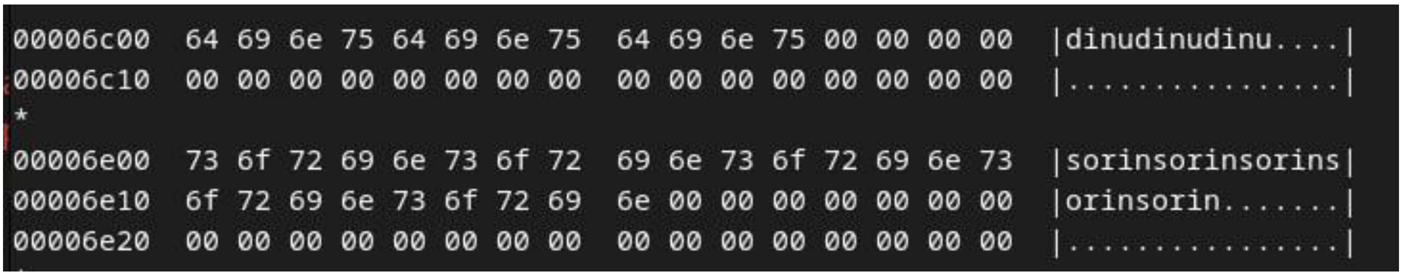
Results

Here are some examples for each function from task 2.


BestOS [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Right Ctrl

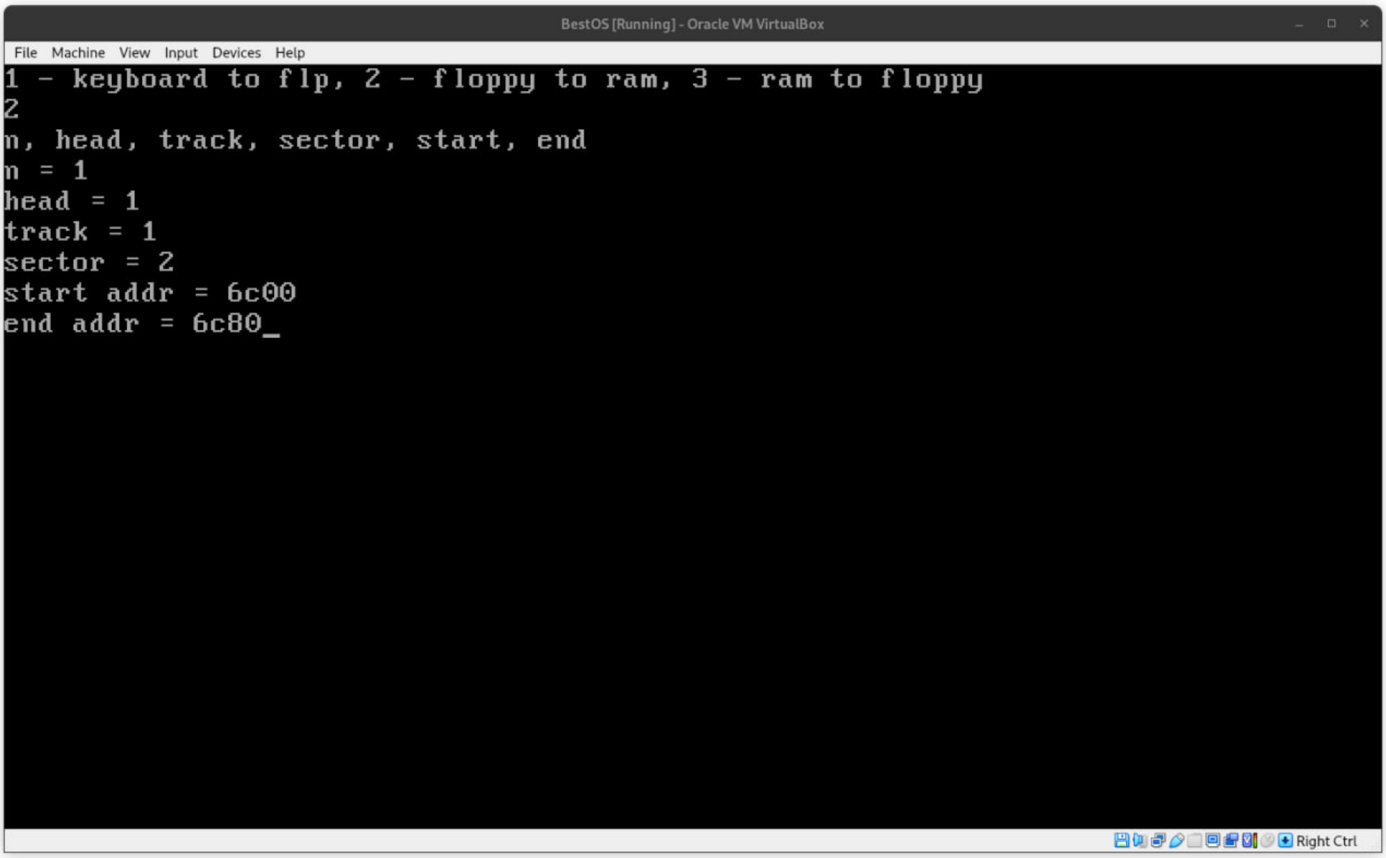
```
1 - keyboard to flp, 2 - floppy to ram, 3 - ram to floppy
1
n, head, track, sector, string
n = 3
head = 1
track = 1
sector = 1
string = dinu
Error message: 0

1
n, head, track, sector, string
n = 5
head = 1
track = 1
sector = 2
string = sorin
Error message: 0


_
```

Figure 1. Keyboard to floppy



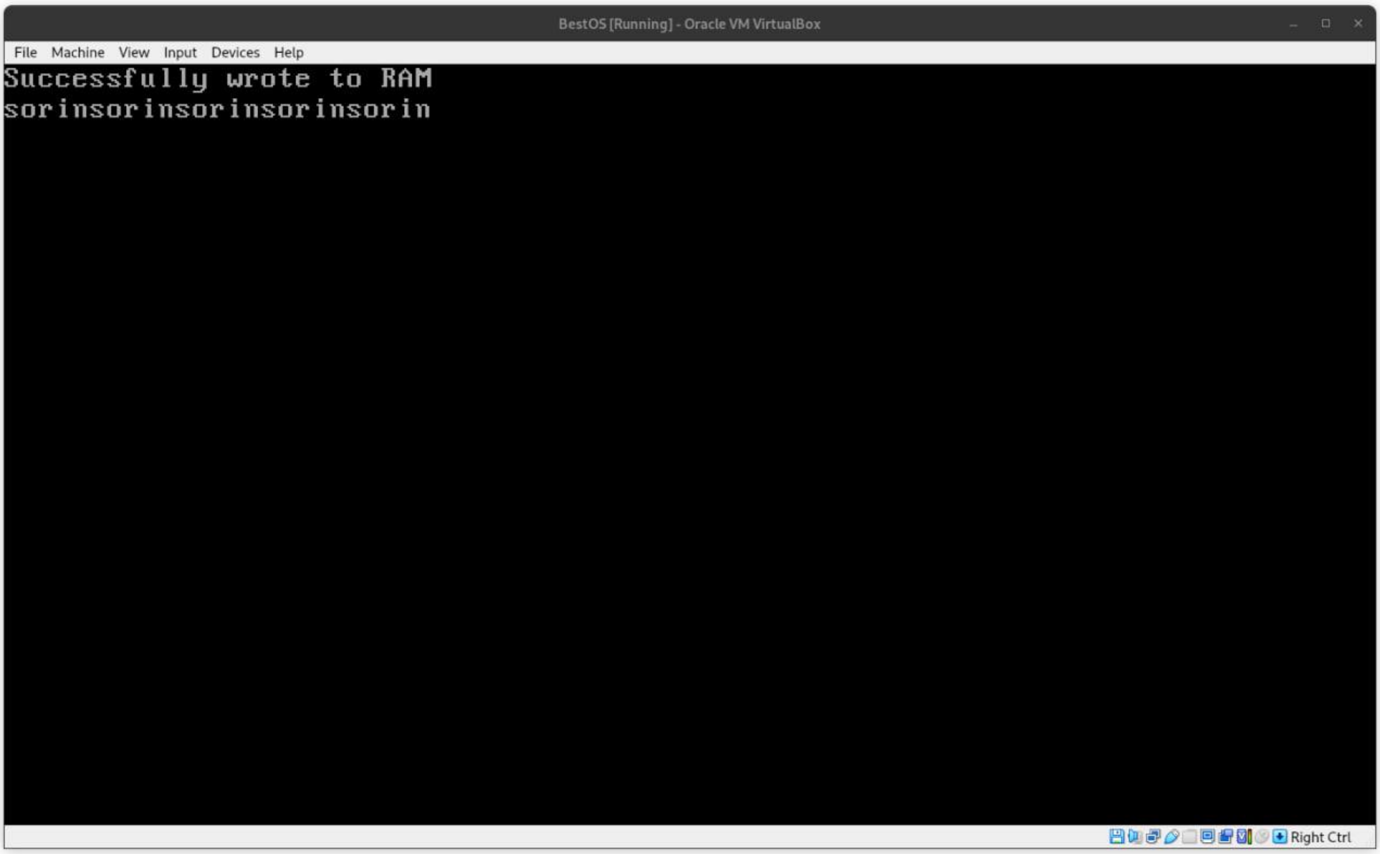
```
00006c00  64 69 6e 75 64 69 6e 75  64 69 6e 75 00 00 00 00  |dinudinudinu....|
00006c10  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00006e00  73 6f 72 69 6e 73 6f 72  69 6e 73 6f 72 69 6e 73  |sorinsorinsorins|
00006e10  6f 72 69 6e 73 6f 72 69  6e 00 00 00 00 00 00 00  |orinsorin.......|
00006e20  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
```

Figure 1.1. Hex dump of the bootable image


BestOS [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Right Ctrl

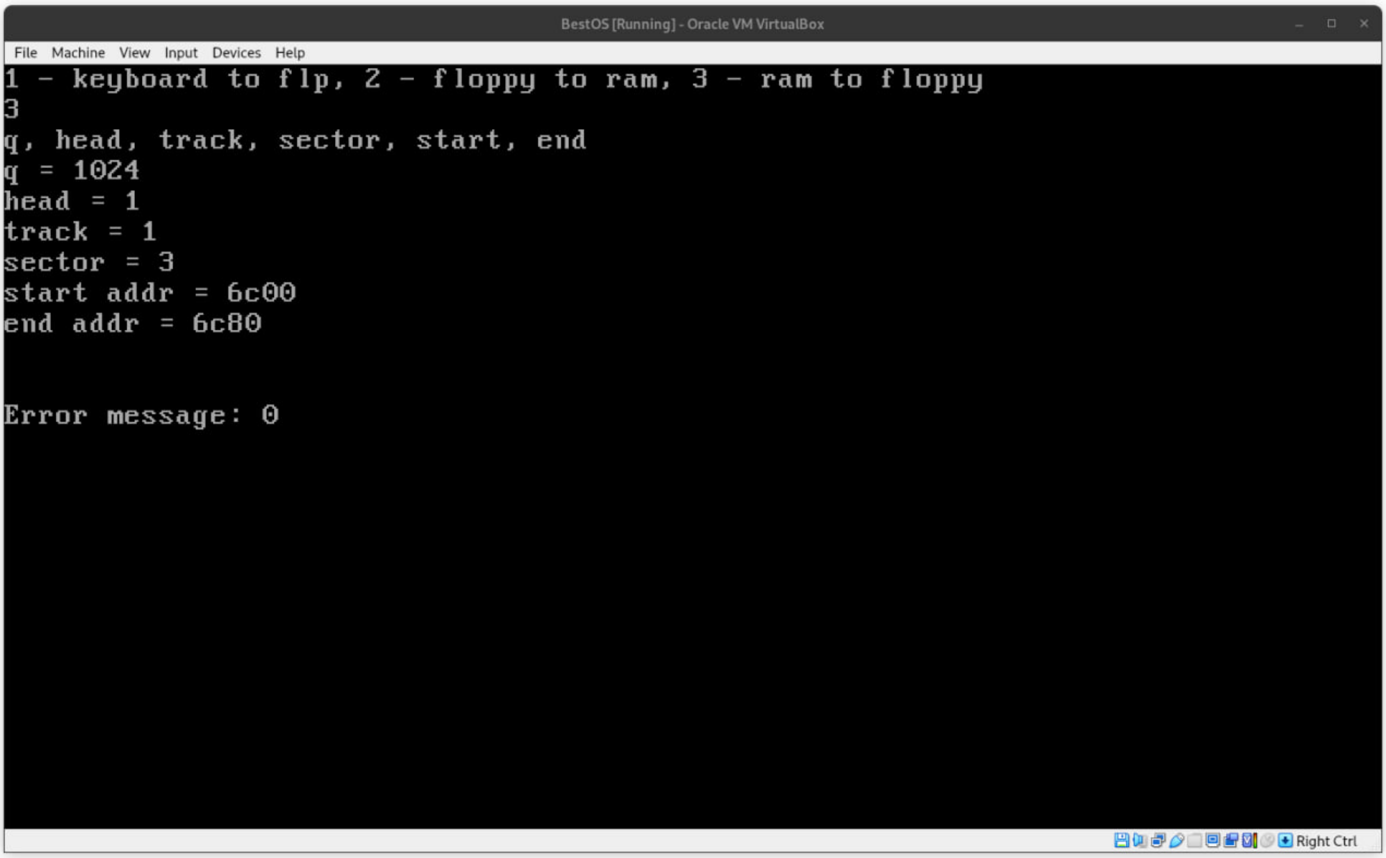
```
1 - keyboard to flp, 2 - floppy to ram, 3 - ram to floppy
2
n, head, track, sector, start, end
n = 1
head = 1
track = 1
sector = 2
start addr = 6c00
end addr = 6c80_
```

Figure 2.1. Floppy to RAM


BestOS [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Right Ctrl

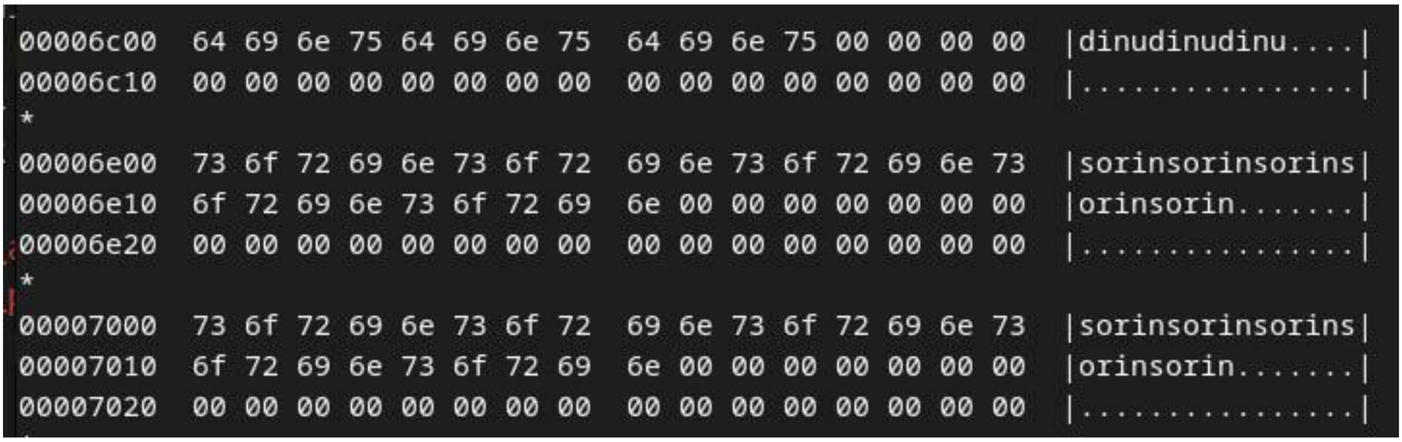
```
Successfully wrote to RAM
sorinsorinsorinsorinsorin
```

Figure 2.2. RAM Contents


BestOS [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Right Ctrl

```
1 - keyboard to flp, 2 - floppy to ram, 3 - ram to floppy
3
q, head, track, sector, start, end
q = 1024
head = 1
track = 1
sector = 3
start addr = 6c00
end addr = 6c80


Error message: 0
```

Figure 3.1. RAM to Floppy



```
00006c00  64 69 6e 75 64 69 6e 75  64 69 6e 75 00 00 00 00  |dinudinudinu....|
00006c10  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00006e00  73 6f 72 69 6e 73 6f 72  69 6e 73 6f 72 69 6e 73  |sorinsorinsorins|
00006e10  6f 72 69 6e 73 6f 72 69  6e 00 00 00 00 00 00 00  |orinsorin.......|
00006e20  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00007000  73 6f 72 69 6e 73 6f 72  69 6e 73 6f 72 69 6e 73  |sorinsorinsorins|
00007010  6f 72 69 6e 73 6f 72 69  6e 00 00 00 00 00 00 00  |orinsorin.......|
00007020  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
```

Figure 3.2. Hex dump of the bootable image

Conclusion

To wrap up, this laboratory work requires deep knowledge of Assembly language. As a team, we faced many challenges to complete the tasks such as difficult bugs, wrong order of operations, lack of resources and tutorials and so on. However, we managed to plow through all of this and achieved our desired result, even though at first this laboratory worked seemed impossible. We adapted the build script to our requirements, implemented helper functions such as print_string and str_compare, created the bootloader and finally the main program itself. We tested thoroughly the functions, although there is still room for improvement.

Github: Syn4z/SO-Team (github.com)