

**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 3](#_Toc151919771)

[Tasks 3](#_Toc151919772)

[Requirements 3](#_Toc151919773)

[Code 3](#_Toc151919774)

[Build script 3](#_Toc151919775)

[Task 1 4](#_Toc151919776)

[**ciprian.asm** 5](#_Toc151919777)

[**sorin.asm** 5](#_Toc151919778)

[**dinu.asm** 6](#_Toc151919779)

[Task 2 7](#_Toc151919780)

[**bootloader.asm** 7](#_Toc151919781)

[**print\_string.asm** 7](#_Toc151919782)

[**str\_compare.asm** 8](#_Toc151919783)

[**main.asm** 8](#_Toc151919784)

[Conclusion 21](#_Toc151919785)

[Github 21](#_Toc151919786)

# 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.

1. Write a program in Assembly with the following functions:
   1. **(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.
   2. **(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.
   3. **(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:

Let’s apply the formula for

Thus we obtain the following results:

* Track = 28
* Sector = 3
* Head = 1, since

### **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

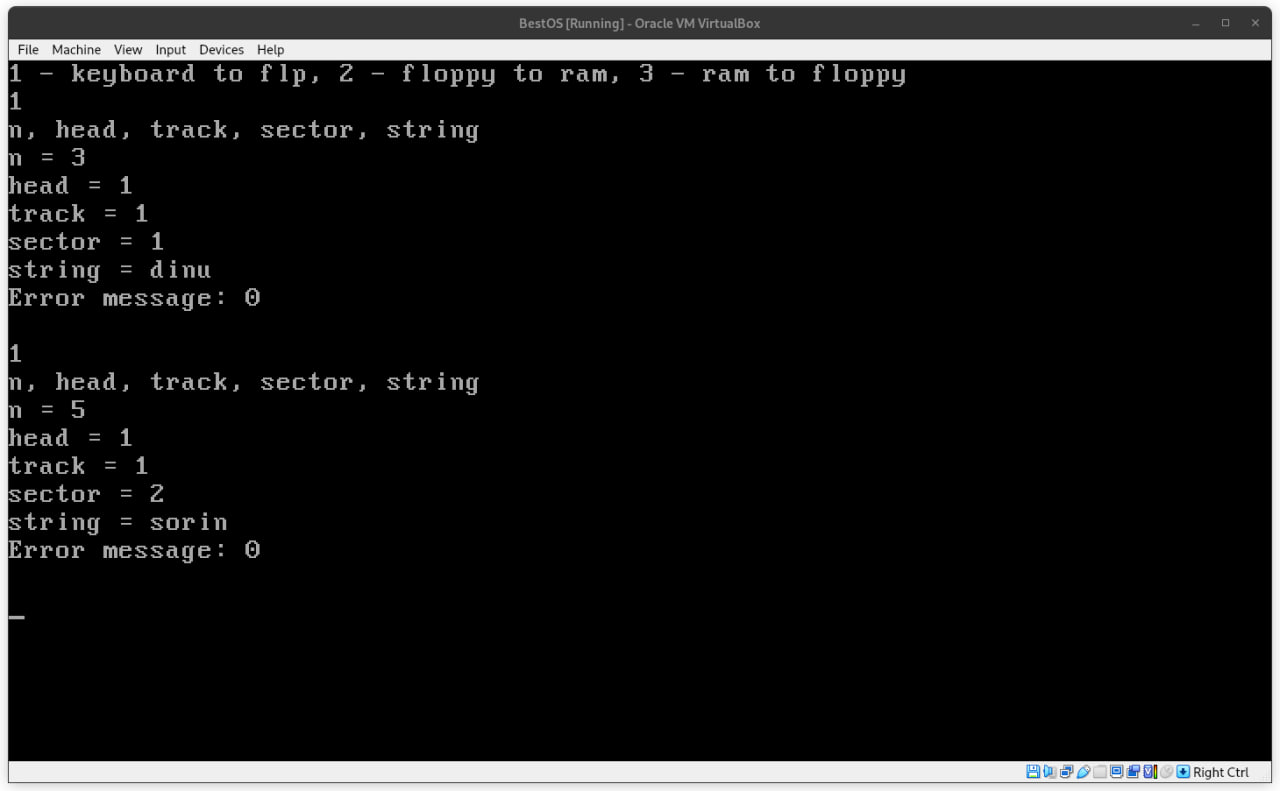
ram\_success: db 0

floppy\_buffer: times 512 db 0

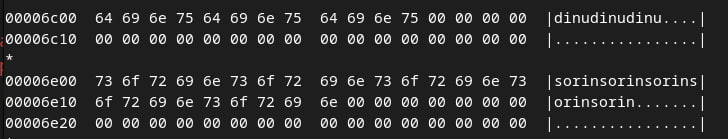
input: times 256 db 0

**Results**

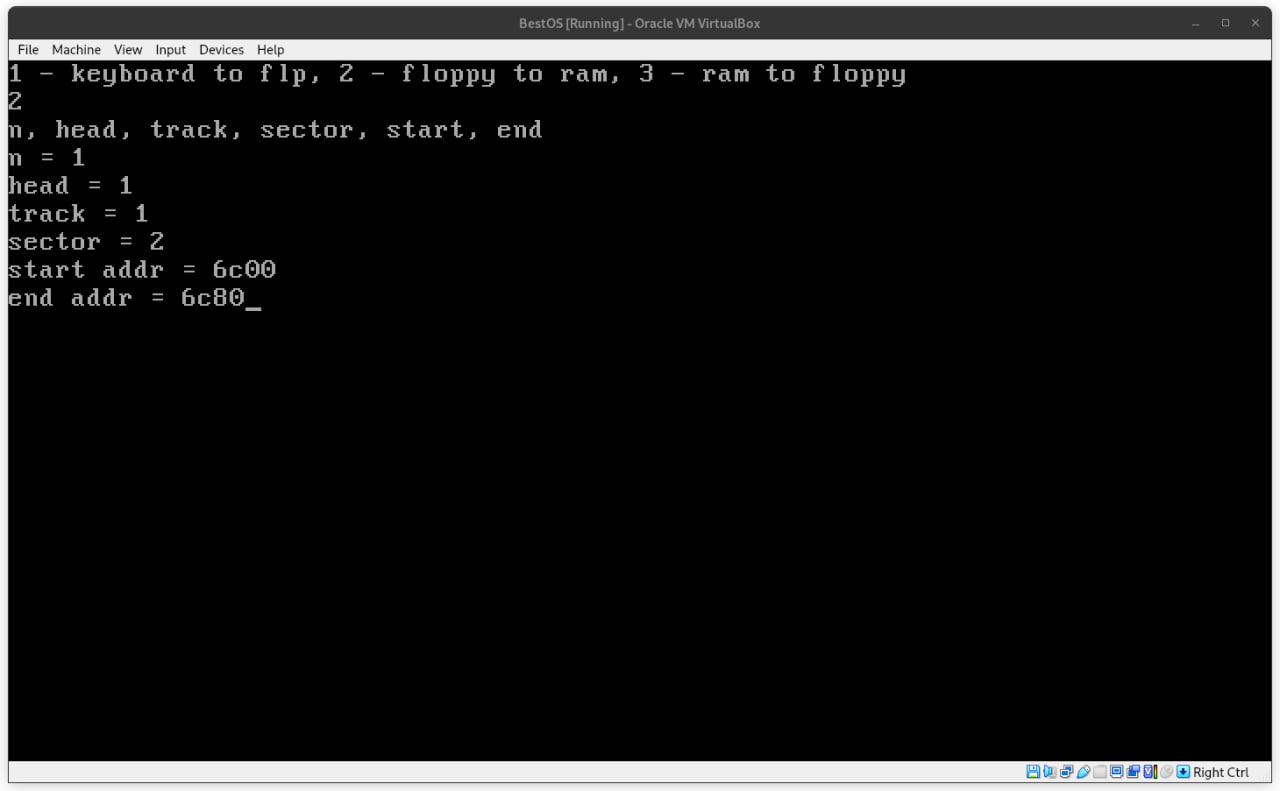
Here are some examples for each function from task 2.



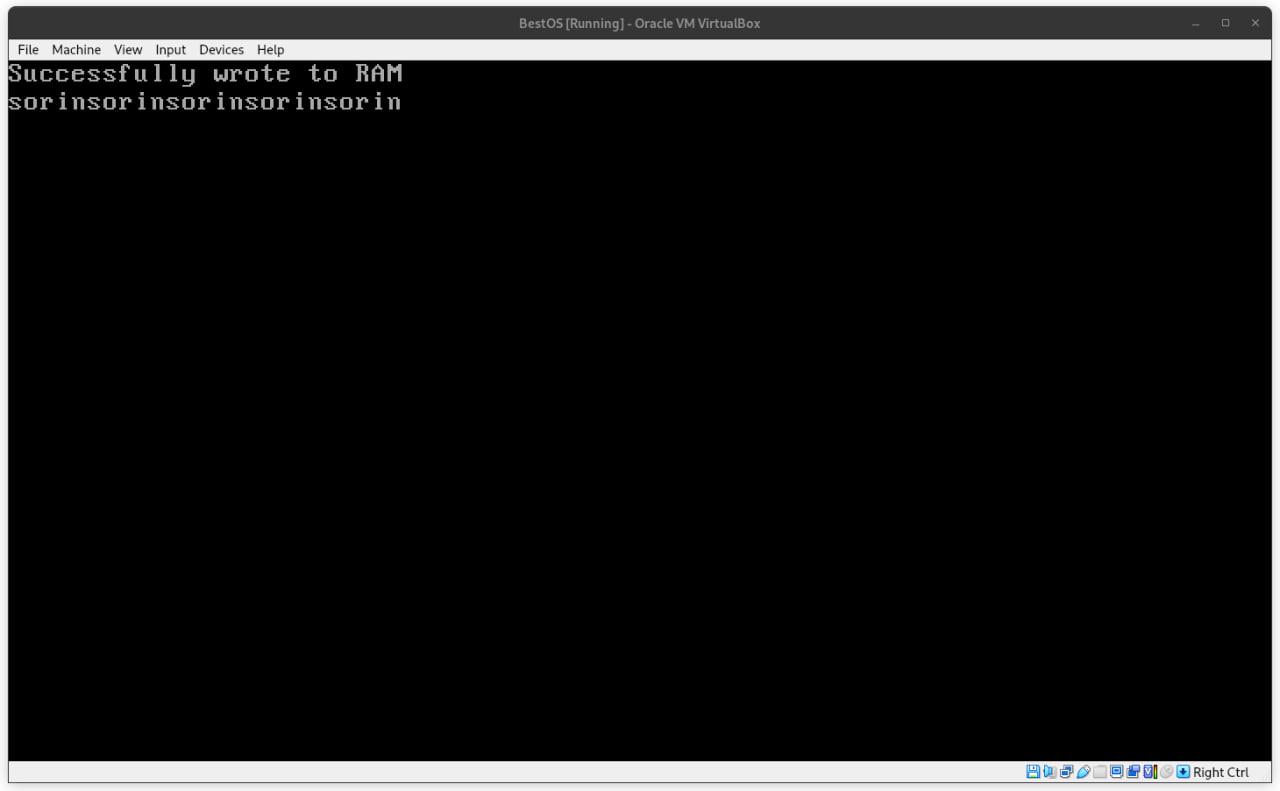
**Figure 1**. Keyboard to floppy



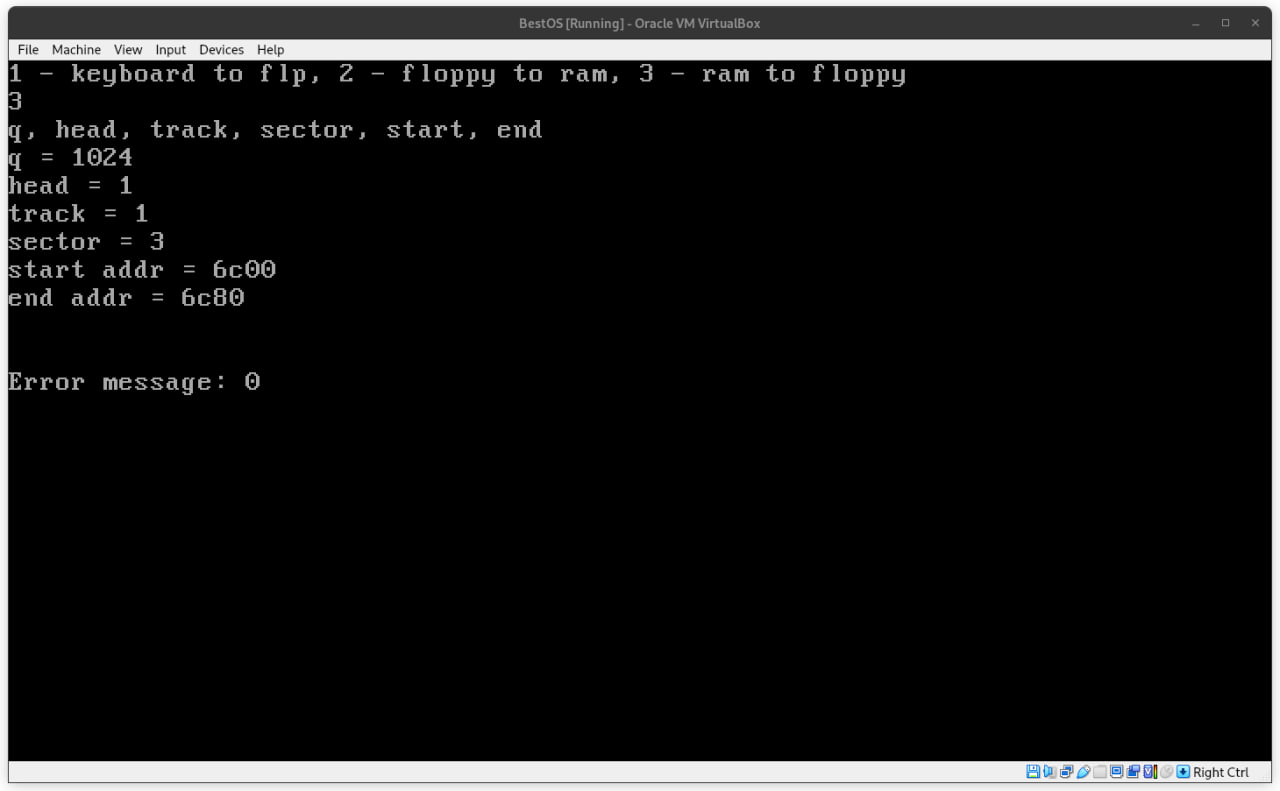
**Figure 1.1**. Hex dump of the bootable image



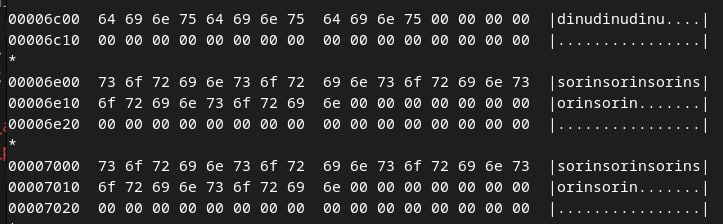
**Figure 2.1**. Floppy to RAM



**Figure 2.2**. RAM Contents



**Figure 3.1**. RAM to Floppy



**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)](https://github.com/syn4z/so-team)