Ministerul Educatiei, Culturii şi Cercetarii al Republicii
Moldova Universitatea Tehnică a Moldovei
Facultatea Calculatoare, Informatică şi
Microelectronică Departamentul Ingineria Software şi
Automatica

REPORT

Laboratory work No. 3
Course: Operating Systems
Theme: Floppy Disk I/O operations

Authors: Catlabuga Corneliu,
Golban Beatricia,
Gîtlan Gabriel,
FAF-213

Checked by: univ. lect.
Călin Rostislav

Chişinău 2023

Objective:

Acquire working skills with floppy disk access methods, especially reading and writing data, but not limited to them.

Task:

The given procedures also extend to other permanent data storage media such as HDD disks. The total disk volume of 1474560 bytes will be structured in 96 logical blocks of 30 sectors each. Each student will be allocated an individual block of 15360 bytes, for recording his data as follows. The block distribution is represented in the included file ("Floppy space distribution.xlsx").

1. In the first and last sector of each student's block (on diskette), textual information must be entered in the following format (without quotes): "@@@FAF-21* First name NAME###". This text string must be duplicated 10 times without additional delimiters.

2. Create an assembly language program that will have the following functions:

- (KEYBOARD ==> FLOPPY) : Reading from the keyboard a string with a maximum length of 256 characters (backspace correction should work) and writing this string to the floppy "N" times, starting at address {Head, Track, Sector }. Where "N" can take values in the range (1-30000). After the ENTER key is detected, if the length of the string is greater than 0 (zero), a blank line and then the newly entered string should be displayed. The variables "N", "Head", "Track" and "Sector" must be read visibly from the keyboard. After the disk write operation is finished, the error code should be displayed on the screen.
- (FLOPPY ==> RAM) : Reading from floppy disk "N" sectors starting at address {Head, Track, Sector} and transferring this data to RAM memory starting at address {XXXX:YYYY}. After the read operation from the diskette is finished, the error code should be displayed on the screen. After the error code, the entire volume of data at address {XXXX:YYYY} that was read from the disk should be displayed on the screen. If the displayed data volume is larger than a video page, then it is necessary to implement pagination by pressing the "SPACE" key. The variables "N", "Head", "Track" and "Sector" as well as the address {XXXX:YYYY} must also be read from the keyboard.
- (RAM ==> FLOPPY) : Writing to the floppy disk starting from the address {Head, Track, Sector} a volume of "Q" bytes, from the RAM memory starting from the address {XXXX:YYYY}. The data block of "Q" bytes must be displayed on the screen, and after the disk write operation is finished, the error code must be displayed on the screen.

3. After executing a function above, the program must be ready to execute the next function (any of the 3 functions described above).

4. The compiled code should preferably not exceed 512 bytes. Otherwise, it is necessary to implement the bypass of this restriction and finally to create the bootable disk image that works in VirtualBox.

Theoretical considerations:

This project is a comprehensive exploration of low-level programming using assembly language. It demonstrates a deep understanding of how computer systems operate at the hardware level.

Firstly, the bootloader, an essential component of any system, is responsible for loading the operating system into memory during startup. This process, known as bootstrapping, is crucial for the system to become operational.

Secondly, the project employs direct hardware manipulation, particularly with the floppy disk drive and RAM. This requires a thorough understanding of input/output operations and memory management, as well as the specifics of the hardware being used. The project also demonstrates the use of modular programming, with different .asm files handling different aspects of the program. This enhances the readability and maintainability of the code, making it easier to understand and modify.

Furthermore, the project involves the conversion of int and hex ASCII input to binary, indicating a manipulation of data types. This is a common operation in low-level programming and cryptography, and it's often used in processes like encryption.

Lastly, the project includes a user interaction aspect, with a menu system that allows users to select various operations. This demonstrates an understanding of user interface design, even at the low level of assembly language programming.

Compilation commands:
`nasm -f bin -o boot.img boot.asm &&` `truncate -s 1474560 boot.img`

Implementation and practical results:

boot.asm: *This is the bootloader. It sets up the initial environment and reads the sectors from the disk into memory.*

```
[bits 16]
org 0x7C00

mov bx, 0x0
mov es, bx
mov bx, 0x7E00
```

```
mov cl, 0x2
mov dh, 0x0
mov ch, 0x0

boot_read_loop:
    mov ah, 0x2
    mov al, 0x1
    int 0x13

    mov al, [boot_sectors_to_read]
    mov ah, [boot_sectors_read]
    cmp al, ah
    je boot_finish_read

    cmp cl, 0x12
    jl boot_read_continue
    mov cl, 0x0
    add dh, 0x1

    cmp dh, 0x2
    jl boot_read_continue
    mov dh, 0x0
    add ch, 0x1

    boot_read_continue:
        add cl, 0x1
        mov ax, 0x0
        mov ah, [boot_sectors_read]
        add ah, 0x1
        mov [boot_sectors_read], ah
        add bx, 0x200

    jmp boot_read_loop

boot_finish_read:
    jmp 0x7E00

; TODO: add disk read error handling

boot_sectors_to_read db 0x20
boot_sectors_read db 0x0

times 510-($-$$) db 0
dw 0xaa55
```

```
%include "main.asm"
```

main.asm:*This is the main program file. It calls various routines from other files and includes all the necessary files.*

```
main:

; Call "wirte_names" from "write_names.asm"
call write_names

; Go to "menu" from "menu.asm"
jmp menu

; Include all the files
%include "src/write_names.asm"

%include "src/menu.asm"
%include "src/prompts.asm"
%include "src/screen_routines.asm"
%include "src/string_convertors.asm"

%include "src/keyboard_to_floppy.asm"
%include "src/floppy_to_ram.asm"
%include "src/ram_to_floppy.asm"

%include "src/floppy_params.asm"
```

write_names.asm: *This file contains routines to write strings to the floppy disk.*

```
write_names:
    ; Write sector 1
    ; to 2041
    mov ah, 03h
    mov dl, 0x0         ; Drive 0
    mov al, 0x1         ; Sector count 1
    mov ch, 0x38        ; Track 56
    mov dh, 0x1         ; Head 1
    mov cl, 0x8         ; Sector 8
    mov bx, sector_1    ; Buffer
    int 13h

    ; to 2070
    mov ah, 03h
    mov dl, 0x0         ; Drive 0
    mov al, 0x1         ; Sector count 1
    mov ch, 0x39        ; Track 57
    mov dh, 0x1         ; Head 1
    mov cl, 0x1         ; Sector 1
    mov bx, sector_1    ; Buffer
    int 13h

    ; Write sector 2
    ; to 2281
    mov ah, 03h
```

```
    mov dl, 0x0         ; Drive 0
    mov al, 0x1         ; Sector count 1
    mov ch, 0x3F        ; Track 63
    mov dh, 0x0         ; Head 0
    mov cl, 0xE         ; Sector 14
    mov bx, sector_2    ; Buffer
    int 13h

    ; to 2310
    mov ah, 03h
    mov dl, 0x0         ; Drive 0
    mov al, 0x1         ; Sector count 1
    mov ch, 0x40        ; Track 64
    mov dh, 0x0         ; Head 0
    mov cl, 0x7         ; Sector 7
    mov bx, sector_2    ; Buffer
    int 13h

    ; Write sector 3
    ; to 2311
    mov ah, 03h
    mov dl, 0x0         ; Drive 0
    mov al, 0x1         ; Sector count 1
    mov ch, 0x40        ; Track 64
    mov dh, 0x0         ; Head 0
    mov cl, 0x8         ; Sector 8
    mov bx, sector_3    ; Buffer
    int 13h

    ; to 2340
    mov ah, 03h
    mov dl, 0x0         ; Drive 0
    mov al, 0x1         ; Sector count 1
    mov ch, 0x41        ; Track 65
    mov dh, 0x0         ; Head 0
    mov cl, 0x1         ; Sector 1
    mov bx, sector_3    ; Buffer
    int 13h

    ret

section .data
    sector_1 db "@@@FAF-213 Corneliu Catlabuga###@@@FAF-213 Corneliu
Catlabuga###@@@FAF-213 Corneliu Catlabuga###@@@FAF-213 Corneliu
Catlabuga###@@@FAF-213 Corneliu Catlabuga###@@@FAF-213 Corneliu
Catlabuga###@@@FAF-213 Corneliu Catlabuga###@@@FAF-213 Corneliu
Catlabuga###@@@FAF-213 Corneliu Catlabuga###@@@FAF-213 Corneliu
Catlabuga###"
    times 0x200 - ($ - sector_1) db 0x0

    sector_2 db "@@@FAF-213 Beatricia Golban###@@@FAF-213 Beatricia
Golban###@@@FAF-213 Beatricia Golban###@@@FAF-213 Beatricia
Golban###@@@FAF-213 Beatricia Golban###@@@FAF-213 Beatricia
Golban###@@@FAF-213 Beatricia Golban###@@@FAF-213 Beatricia
```

```
Golban###@@@FAF-213 Beatricia Golban###@@@FAF-213 Beatricia Golban###"
    times 0x200 - ($ - sector_2) db 0x0

    sector_3 db "@@@FAF-213 Gabriel Gitlan###@@@FAF-213 Gabriel
Gitlan###@@@FAF-213 Gabriel Gitlan###@@@FAF-213 Gabriel Gitlan###@@@FAF-
213 Gabriel Gitlan###@@@FAF-213 Gabriel Gitlan###@@@FAF-213 Gabriel
Gitlan###@@@FAF-213 Gabriel Gitlan###@@@FAF-213 Gabriel Gitlan###@@@FAF-
213 Gabriel Gitlan###"
    times 0x200 - ($ - sector_3) db 0x0
```

screen_routines.asm: *This file contains routines to handle screen operations such as clearing the screen and printing messages.*

```
; Clear the screen
clear_screen:
    pusha
    ; Clear the screen with INT 10h 07h
    mov ah, 07h
    mov al, 0x0
    mov bh, 0x7
    mov cx, 0x0
    mov dx, 0x184F
    int 10h

    ; Move the cursor to the top left corner
    mov ah, 02h
    mov dx, 0x0
    int 10h
    popa

    ret

; Remove the last character from the screen
remove_last_char:
    ; Move the cursor back
    mov ah, 02h
    sub dl, 0x1
    int 10h

    ; Remove the character from the screen
    mov ah, 0Ah
    mov al, 0x0
    int 10h

    ret

; Remove the last char on previous line
remove_last_char_line:
    ; Move the cursor to the end of the previous line
    mov ah, 02h
    mov dl, 0x4F
    sub dh, 0x1
    int 10h

    ; Remove the character from the screen
```

```
    mov ah, 0Ah
    mov al, 0x0
    int 10h

    ret

; Clear the current line
clear_row:
    mov ah, 03h
    int 10h

    mov ax, 1301h
    mov bx, 0x7
    mov dl, 0x0
    mov bp, clean_row
    mov cx, 0x50
    int 10h

    ret

; Print the floppy error
print_error:
    cmp ah, 00h
    je floppy_error_00

    cmp ah, 01h
    je floppy_error_01

    cmp ah, 02h
    je floppy_error_02

    cmp ah, 03h
    je floppy_error_03

    cmp ah, 04h
    je floppy_error_04

    cmp ah, 06h
    je floppy_error_06

    cmp ah, 08h
    je floppy_error_08

    cmp ah, 09h
    je floppy_error_09

    cmp ah, 0Ch
    je floppy_error_0C

    cmp ah, 10h
    je floppy_error_10

    cmp ah, 20h
    je floppy_error_20
```

```
cmp ah, 40h
je floppy_error_40

cmp ah, 80h
je floppy_error_80

floppy_error_00:
    mov bp, flp_err_00
    mov cx, flp_err_00_len
    jmp floppy_error_return

floppy_error_01:
    mov bp, flp_err_01
    mov cx, flp_err_01_len
    jmp floppy_error_return

floppy_error_02:
    mov bp, flp_err_02
    mov cx, flp_err_02_len
    jmp floppy_error_return

floppy_error_03:
    mov bp, flp_err_03
    mov cx, flp_err_03_len
    jmp floppy_error_return

floppy_error_04:
    mov bp, flp_err_04
    mov cx, flp_err_04_len
    jmp floppy_error_return

floppy_error_06:
    mov bp, flp_err_06
    mov cx, flp_err_06_len
    jmp floppy_error_return

floppy_error_08:
    mov bp, flp_err_08
    mov cx, flp_err_08_len
    jmp floppy_error_return

floppy_error_09:
    mov bp, flp_err_09
    mov cx, flp_err_09_len
    jmp floppy_error_return

floppy_error_0C:
    mov bp, flp_err_0C
    mov cx, flp_err_0C_len
    jmp floppy_error_return

floppy_error_10:
    mov bp, flp_err_10
```

```
        mov cx, flp_err_10_len
        jmp floppy_error_return

    floppy_error_20:
        mov bp, flp_err_20
        mov cx, flp_err_20_len
        jmp floppy_error_return

    floppy_error_40:
        mov bp, flp_err_40
        mov cx, flp_err_40_len
        jmp floppy_error_return

    floppy_error_80:
        mov bp, flp_err_80
        mov cx, flp_err_80_len
        jmp floppy_error_return

    floppy_error_return:
        mov ax, 1301h
        mov bx, 0x7
        mov dh, 0x0
        mov dl, 0x0
        int 10h

        clc

        mov ah, 00h
        int 16h

        ret

section .data
    flp_err_00 db "No error"
    flp_err_00_len equ $ - flp_err_00

    flp_err_01 db "Bad command"
    flp_err_01_len equ $ - flp_err_01

    flp_err_02 db "Bad address mark"
    flp_err_02_len equ $ - flp_err_02

    flp_err_03 db "Write protected"
    flp_err_03_len equ $ - flp_err_03

    flp_err_04 db "Bad sector ID"
    flp_err_04_len equ $ - flp_err_04

    flp_err_06 db "Disk change line active"
    flp_err_06_len equ $ - flp_err_06

    flp_err_08 db "DMA failure"
    flp_err_08_len equ $ - flp_err_08
```

```
    flp_err_09 db "DMA overrun"
    flp_err_09_len equ $ - flp_err_09

    flp_err_0C db "Media type not available"
    flp_err_0C_len equ $ - flp_err_0C

    flp_err_10 db "Bad CRC"
    flp_err_10_len equ $ - flp_err_10

    flp_err_20 db "Diskete controller failure"
    flp_err_20_len equ $ - flp_err_20

    flp_err_40 db "Bad seek"
    flp_err_40_len equ $ - flp_err_40

    flp_err_80 db "Time-out"
    flp_err_80_len equ $ - flp_err_80
```

menu.asm: *This file handles the main menu display and user input for selecting options.*

```
menu:
    ; Call "clear_screen" from "screen_routines.asm"
    call clear_screen

    ; Print the main message
    mov ax, 1301h
    mov bx, 0x7
    mov bp, main_message
    mov cx, main_message_size
    mov dl, 0x0
    mov dh, 0x0
    int 10h

    ; Read the option
    mov ah, 00h
    int 16h

    ; Check if option "1"
    cmp al, '1'
    je keyboard_to_floppy

    ; Check if option "2"
    cmp al, '2'
    je floppy_to_ram

    ; Check if option "3"
    cmp al, '3'
    je ram_to_floppy

    ; If the option is not valid display an unexpected option message
    ; Call "clear_screen" from "screen_routines.asm"
    call clear_screen

    ; Print the unexpected option message
    mov ax, 1301h
```

```
    mov bx, 0x7
    mov bp, unexpected_option_message
    mov cx, unexpected_option_message_size
    mov dl, 0x0
    mov dh, 0x0
    int 10h

    ; Wait for a key press
    mov ah, 00h
    int 16h

    ; Return to the menu
    jmp menu

section .data
    ; Main message buffer and size
    main_message db "Please select an option: ", 0xA, 0xD, "1. Keyboard to
Floppy", 0xA, 0xD, "2. Floppy to RAM", 0xA, 0xD, "3. RAM to Floppy", 0xA,
0xD, 0xA, 0xD, "Your option: "
    main_message_size equ $ - main_message

    ; Unexpected option message buffer and size
    unexpected_option_message db "Unexpected option!", 0xA, 0xD, 0xA, 0xD,
"Press any key to continue..."
    unexpected_option_message_size equ $ - unexpected_option_message
```

string_convertors.asm: *This file contains routines to convert string input into other formats.*

```
; Convert ASCII string to integer
read_num:
    ; Read a character
    mov ah, 00h
    int 16h

    ; Check if the key is escape
    ; (if yes: go to "read_num_return")
    cmp al, 0x1B
    je read_num_return

    ; (if no)
    ; Check if the key is enter
    ; (if yes: go to "read_num_return")
    cmp al, 0x0D
    je read_num_return

    ; (if no)
    ; Check if the key is backspace
    ; (if yes: go to "read_num_backspace")
    cmp al, 0x08
    je read_num_backspace

    ; Check if the key is in range 0-9
    ; (if no: loop back to "read_num")
    cmp al, 0x30
```

```
jl read_num
cmp al, 0x39
jg read_num

; (if yes)
; Subtract 0x30 from the ASCII value
sub al, 0x30

; Place number in CX
mov cl, al

; Move the number in AX
mov ax, [num_buffer]

; Check if the number is 3276
; (if yes go to "limit_num")
cmp ax, 0xCCC
je limit_num ; (accept between 0 and 7)

; (if no)
; Check if the number is less than 3276
; (if yes: go to "accept_num")
cmp ax, 0xCCC
jl accept_num

; (if no)
; Loop back to "read_num"
jmp read_num

accept_num:
    ; Place the number in AX
    ; Multiply the number by 10
    mov ax, [num_buffer]
    mov dx, 0xA
    mul dx

    ; Add to ax the number in CX
    add ax, cx

    ; Store to num_buffer the number in AX
    mov [num_buffer], ax

    ; Print the number
    mov ah, 0Eh
    mov al, cl
    add al, 0x30
    int 10h

    ; Loop back to "read_num"
    jmp read_num

; Return from the function
read_num_return:
    ret
```

```
    ; Handle backspace
    read_num_backspace:
        ; Check if the buffer is empty
        ; (if yes: loop back to "read_num")
        mov dx, 0x0
        mov ax, [num_buffer]
        cmp ax, 0x0
        je read_num

        ; (if no)
        ; Divide the number by 10
        mov cx, 0xA
        div cx

        ; Store the number in AX to num_buffer
        mov [num_buffer], ax

        ; Remove the last character from the screen
        pusha
        mov ah, 03h
        mov bh, 0x0
        int 10h
        mov ah, 02h
        sub dl, 0x1
        int 10h
        mov ah, 0Ah
        mov al, 0x0
        int 10h
        popa

        ; Loop back to "read_num"
        jmp read_num

    ; Limit the number to 3276
    limit_num:
        ; Check if the number is greater than 7
        ; (if yes: loop back to "read_num")
        cmp cl, 0x7
        jg read_num

        ; (if no)
        ; Go to "accept_num"
        jmp accept_num

; Convert ASCII string to hexadecimal
read_address:
    ; Read a character
    mov ah, 00h
    int 16h

    ; Check if the key is escape
    ; (if yes: go to "read_address_return")
    cmp al, 0x1B
```

```
je read_address_return

; (if no)
; Check if the key is enter
; (if yes: go to "read_address_return")
cmp al, 0x0D
je read_address_return

; (if no)
; Check if the key is backspace
; (if yes: go to "read_address_backspace")
cmp al, 0x08
je read_address_backspace

; (if no)
; Check if the key lower than 0x30
; (if yes: loop back to "read_address")
cmp al, 0x30
jl read_address

; (if no)
; Check if the less than 0x3A
; (if yes: go to "handle_number")
cmp al, 0x3A
jl handle_number

; (if no)
; Check if the key is less than 0x41
; (if yes: loop back to "read_address")
cmp al, 0x41
jl read_address

; (if no)
; Check if the key is less than 0x47
; (if yes: go to "handle_uppercase")
cmp al, 0x47
jl handle_uppercase

; (if no)
; Check if the key is less than 0x61
; (if yes: loop back to "read_address")
cmp al, 0x61
jl read_address

; (if no)
; Check if the key is less than 0x67
; (if yes: go to "handle_lowercase")
cmp al, 0x67
jl handle_lowercase

; (if no)
; Loop back to "read_address"
jmp read_address
```

```
; Handle the character as a number
handle_number:
    ; Subtract 0x30 from the ASCII value
    sub al, 0x30

    ; Got to "handle_value"
    jmp handle_value

; Handle the character as an uppercase letter
handle_uppercase:
    ; Subtract 0x37 from the ASCII value
    sub al, 0x37

    ; Got to "handle_value"
    jmp handle_value

; Handle the character as a lowercase letter
handle_lowercase:
    ; Subtract 0x57 from the ASCII value
    sub al, 0x57

    ; Got to "handle_value"
    jmp handle_value

; Handle the character as a value
handle_value:
    ; Place the value in CL
    mov cl, al

    ; Move the buffer in AX
    mov ax, [num_buffer]

    ; Check if the buffer is above 0xFFF
    ; (if yes: loop back to "read_address")
    cmp ax, 0xFFF
    ja read_address

    ; (if no)
    ; Multiply the buffer by 0x10
    mov dx, 0x10
    mul dx

    ; Add the value to the buffer
    add ax, cx
    mov [num_buffer], ax

    ; Print the value
    mov ah, 0Eh
    mov al, cl
    cmp cx, 0x9
    jg print_letter
        ; Print as number
        add al, 0x30
        int 10h
```

```
            ; Loop back to "read_address"
            jmp read_address

        print_letter:
            ; Print as letter
            add al, 0x37
            int 10h

            ; Loop back to "read_address"
            jmp read_address

    ; Return from the function
    read_address_return:
        ret

    ; Handle backspace
    read_address_backspace:
        ; Check if the buffer is empty
        ; (if yes: loop back to "read_address")
        mov dx, 0x0
        mov ax, [num_buffer]
        cmp ax, 0x0
        je read_address

        ; (if no)
        ; Divide the num_buffer by 0x10
        mov cx, 0x10
        div cx
        mov [num_buffer], ax

        ; Remove the last character from the screen
        pusha
        mov ah, 03h
        mov bh, 0x0
        int 10h
        mov ah, 02h
        sub dl, 0x1
        int 10h
        mov ah, 0Ah
        mov al, 0x0
        int 10h
        popa

        ; Loop back to "read_address"
        jmp read_address
```

prompts.asm: *This file contains routines to print various prompts on the screen.*

```
; Print the sectors prompt
print_sectors_prompt:
    mov ax, 1301h
    mov bx, 0x7
    mov dl, 0x0
    mov dh, 0xB
```

```
    mov bp, sectors_prompt
    mov cx, sectors_prompt_size
    int 10h

    ret

; Print the volume prompt
print_volume_prompt:
    mov ax, 1301h
    mov bx, 0x7
    mov dl, 0x0
    mov dh, 0xB
    mov bp, volume_prompt
    mov cx, volume_prompt_size
    int 10h

    ret

; Print the text prompt
print_text_prompt:
    mov ax, 1301h
    mov bx, 0x7
    mov dl, 0x0
    mov dh, 0x2
    mov bp, text_prompt
    mov cx, text_prompt_size
    int 10h

    ret

; Print the buffer
print_buffer:
    mov ax, 1301h
    mov bx, 0x7
    mov dl, 0x0
    mov dh, 0x6
    mov bp, buffer
    mov cx, si
    int 10h

    ret

; Print the side prompt
print_side_prompt:
    mov ax, 1301h
    mov bx, 0x7
    mov dl, 0x0
    mov dh, 0xC
    mov bp, side_prompt
    mov cx, side_prompt_size
    int 10h

    ret
```

```
; Print the track prompt
print_track_prompt:
    mov ax, 1301h
    mov bx, 0x7
    mov dl, 0x0
    mov dh, 0xD
    mov bp, track_prompt
    mov cx, track_prompt_size
    int 10h

    ret

; Print the sector prompt
print_sector_prompt:
    mov ax, 1301h
    mov bx, 0x7
    mov dl, 0x0
    mov dh, 0xE
    mov bp, sector_prompt
    mov cx, sector_prompt_size
    int 10h

    ret

; Print sectors amount warning
print_sectors_warning:
    mov ah, 03h
    int 10h

    mov ax, 1301h
    mov bx, 0x7
    mov dl, 0x0
    add dh, 0x1
    mov bp, sectors_warning
    mov cx, sectors_warning_size
    int 10h

    mov ah, 02h
    sub dh, 0x1
    int 10h

    ret

; Print the times prompt
print_times_prompt:
    mov ax, 1301h
    mov bx, 0x7
    mov dl, 0x0
    mov dh, 0xF
    mov bp, times_prompt
    mov cx, times_prompt_size
    int 10h

    ret
```

```
; Print the times amount warning
print_times_warning:
    mov ah, 03h
    int 10h

    mov ax, 1301h
    mov bx, 0x7
    mov dl, 0x0
    add dh, 0x1
    mov bp, times_warning
    mov cx, times_warning_size
    int 10h

    mov ah, 02h
    sub dh, 0x1
    int 10h

    ret

; Print the side warning
print_side_warning:
    mov ah, 03h
    int 10h

    mov ax, 1301h
    mov bx, 0x7
    mov dl, 0x0
    add dh, 0x1
    mov bp, side_warning
    mov cx, side_warning_size
    int 10h

    mov ah, 02h
    sub dh, 0x1
    int 10h

    ret

; Print the track warning
print_track_warning:
    mov ah, 03h
    int 10h

    mov ax, 1301h
    mov bx, 0x7
    mov dl, 0x0
    add dh, 0x1
    mov bp, track_warning
    mov cx, track_warning_size
    int 10h

    mov ah, 02h
    sub dh, 0x1
```

```
    int 10h

    ret

; Print the sector warning
print_sector_warning:
    mov ah, 03h
    int 10h

    mov ax, 1301h
    mov bx, 0x7
    mov dl, 0x0
    add dh, 0x1
    mov bp, sector_warning
    mov cx, sector_warning_size
    int 10h

    mov ah, 02h
    sub dh, 0x1
    int 10h

    ret

; Print the address prompt
print_address_prompt:
    mov ax, 1301h
    mov bx, 0x7
    mov dl, 0x0
    mov dh, 0xF
    mov bp, address_prompt
    mov cx, address_prompt_size
    int 10h

    ret

section .data
    override_disk_prompt db "Found diskette end. Aborting."
    override_disk_prompt_size equ $ - override_disk_prompt

    text_prompt db "Text: "
    text_prompt_size equ $ - text_prompt

    sectors_prompt db "Sectors: "
    sectors_prompt_size equ $ - sectors_prompt

    sectors_warning db "Sectors must be in range 1-2880"
    sectors_warning_size equ $ - sectors_warning

    volume_prompt db "Volume(bytes): "
    volume_prompt_size equ $ - volume_prompt

    side_prompt db "Side(0-1): "
    side_prompt_size equ $ - side_prompt
```

```
side_warning db "Side must be 0 or 1"
side_warning_size equ $ - side_warning

track_prompt db "Track(1-18): "
track_prompt_size equ $ - track_prompt

track_warning db "Track must be in range 1-18"
track_warning_size equ $ - track_warning

sector_prompt db "Sector(0-79): "
sector_prompt_size equ $ - sector_prompt

sector_warning db "Sector must be in range 0-79"
sector_warning_size equ $ - sector_warning

times_prompt db "Times(1-30000): "
times_prompt_size equ $ - times_prompt

times_warning db "Times must be in range 1-30000"
times_warning_size equ $ - times_warning

address_prompt db "Address(0-FFFF:0-FFFF): "
address_prompt_size equ $ - address_prompt
```

keyboard_to_floppy.asm: *This file contains routines to handle keyboard input and write it to the floppy disk.*

```
; Call "clear_screen" from "screen_routines.asm"
call clear_screen

; Move si to the beginning of the buffer
mov si, buffer

; Call "print_text_prompt" from "prompts.asm"
call print_text_prompt

ktf_input:
    ; Read a character from the keyboard
    mov ah, 00h
    int 16h

    ; Check if enter was pressed and handle it
    ; (if yes: go to "ktf_input_done")
    cmp al, 0x0D
    je ktf_input_done

    ; (if no)
    ; Check if backspace was pressed and handle it
    cmp al, 0x08
    je ktf_bakcspace

    ; Check if escape was pressed and handle it
    ; (if yes: go to "escape")
    cmp al, 0x1B
    je escape
```

```
        ; (if no)
        ; Check if the buffer is full
        ; (if yes: loop back to "ktf_input")
        cmp si, buffer + 0x100
        je ktf_input

        ; (if no)
        ; Check if the character is in printable ASCII limit
        ; (if no: loop back to "ktf_input")
        cmp al, 0x20
        jl ktf_input
        cmp al, 0x7E
        jg ktf_input

        ; (if yes)
        ; Store the character in the buffer
        mov [si], al
        add si, 0x1

        ; Print the character
        mov ah, 0Eh
        int 10h

        ; Loop back to "ktf_input"
        jmp ktf_input

; Handle enter
ktf_input_done:
    ; Check if the buffer is empty
    ; (if yes: go to "menu" from "menu.asm")
    sub si, buffer
    jz menu

    ; (if no)
    ; Get the current cursor position
    mov ah, 03h
    int 10h

    ; call "print_buffer" from "prompts.asm"
    call print_buffer

    ; Get side loop
    get_side:
        ; Clear num_buffer
        mov ax, 0x0
        mov [num_buffer], ax

        ; Move cursor to line 12, column 0
        mov ah, 02h
        mov dl, 0x0
        mov dh, 0xC
        int 10h
```

```
    ; Call "clear_row" from "screen_routines.asm"
    call clear_row

    ; Call "print_side_prompt" from "prompts.asm"
    call print_side_prompt

    ; Call "read_num" from "string_convertors.asm"
    call read_num

    ; Check if read_num returned 0x1B (escape)
    ; (if yes: go to "escape")
    cmp al, 0x1B
    je escape

    ; (if no)
    ; Load num_buffer into ax and store in side buffer
    mov ax, [num_buffer]
    mov [side], ax

    ; Check if side is in range [0, 1]
    ; (if yes: go to "get_track")
    mov ax, [side]
    cmp ax, 0x2
    jl get_track

    ; (if no)
    ; Call "print_side_warning" from "prompts.asm"
    call print_side_warning

    ; Clear the buffers
    mov ax, 0x0
    mov [num_buffer], ax
    mov [side], ax

    ; Loop back to "get_side"
    jmp get_side

; Get track loop
get_track:
    ; Clear num_buffer
    mov ax, 0x0
    mov [num_buffer], ax

    ; Move cursor to line 13, column 0
    mov ah, 02h
    mov dl, 0x0
    mov dh, 0xD
    int 10h

    ; Call "clear_row" from "screen_routines.asm"
    call clear_row

    ; Call "print_track_prompt" from "prompts.asm"
    call print_track_prompt
```

```
    ; Call "read_num" from "string_convertors.asm"
    call read_num

    ; Check if read_num returned 0x1B (escape)
    ; (if yes: go to "escape")
    cmp al, 0x1B
    je escape

    ; (if no)
    ; Load num_buffer into ax and store in track buffer
    mov ax, [num_buffer]
    mov [track], ax

    ; Check greater than 18
    ; (if yes: go to "track_fault")
    mov ax, [track]
    cmp ax, 0x12
    jg track_fault

    ; (if no)
    ; Check if track is 0
    ; (if yes: go to "track_fault")
    cmp ax, 0x0
    je track_fault

    ; (if no)
    ; Go to "get_sector"
    jmp get_sector

    ; Track fault
    track_fault:
        ; Call "print_track_warning" from "prompts.asm"
        call print_track_warning

        ; Clear the buffers
        mov ax, 0x0
        mov [num_buffer], ax
        mov [track], ax

        ; Loop back to "get_track"
        jmp get_track

; Get sector loop
get_sector:
    ; Clear num_buffer
    mov ax, 0x0
    mov [num_buffer], ax

    ; Move cursor to line 14, column 0
    mov ah, 02h
    mov dl, 0x0
    mov dh, 0xE
    int 10h
```

```
    ; Call "clear_row" from "screen_routines.asm"
    call clear_row

    ; Call "print_sector_prompt" from "prompts.asm"
    call print_sector_prompt

    ; Call "read_num" from "string_convertors.asm"
    call read_num

    ; Check if read_num returned 0x1B (escape)
    ; (if yes: go to "escape")
    cmp al, 0x1B
    je escape

    ; (if no)
    ; Load num_buffer into ax and store in sector buffer
    mov ax, [num_buffer]
    mov [sector], ax

    ; Check if sector is less than 80
    ; (if yes: go to "get_times")
    mov ax, [sector]
    cmp ax, 0x50
    jl get_times

    ; (if no)
    ; Call "print_sector_warning" from "prompts.asm"
    call print_sector_warning

    ; Clear the buffers
    mov ax, 0x0
    mov [num_buffer], ax
    mov [sector], ax

    ; Loop back to "get_sector"
    jmp get_sector

; Get times loop
get_times:
    ; Clear num_buffer
    mov ax, 0x0
    mov [num_buffer], ax

    ; Move cursor to line 15, column 0
    mov ah, 02h
    mov dl, 0x0
    mov dh, 0xF
    int 10h

    ; Call "clear_row" from "screen_routines.asm"
    call clear_row

    ; Call "print_times_prompt" from "prompts.asm"
```

```
        call print_times_prompt

        ; Call "read_num" from "string_convertors.asm"
        call read_num

        ; Place num_buffer in write_times
        mov ax, [num_buffer]
        mov [write_times], ax

        ; Check if write_times is less than 30001
        ; (if yes: go to "write_to_floppy")
        mov ax, [write_times]
        cmp ax, 0x7531
        jl write_to_floppy

        ; (if no)
        ; Call "print_times_warning" from "prompts.asm"
        call print_times_warning

        ; Clear the buffers
        mov ax, 0x0
        mov [num_buffer], ax
        mov [write_times], ax

        ; Loop back to "get_times"
        jmp get_times

; Write to floppy
write_to_floppy:
    ; Place di at the beginning of floppy_buffer
    mov di, floppy_buffer

    ; Reset ax (used to count len of floppy_buffer)
    mov ax, 0x0

    ; Write to floppy_buffer loop
    repeated_write:
        ; Place si at the beginning of buffer
        mov si, buffer

        ; Get the number times to copy
        mov cx, [write_times]

        ; Check if write_times is 0
        ; (if yes: go to "to_floppy")
        cmp cx, 0x0
        je to_floppy

        ; (if no)
        ; Decrement write_times by 1
        sub cx, 0x1
        mov [write_times], cx

        ; Read char from buffer loop
```

```
    char_write:
        ; Place the char from si to di
        mov cx, 0x0
        mov cl, byte [si]
        mov byte [di], cl

        ; Advance si, di
        add si, 0x1
        add di, 0x1

        ; Increment ax
        add ax, 0x1

        ; Check reached end of buffer
        ; (if yes: go to "repeated_write")
        cmp byte [si], 0x0
        je repeated_write

        ; (if no)
        ; Read next char from buffer
        jmp char_write

; Write floppy_buffer to floppy
to_floppy:
    ; Get the number of sectors to write by dividing ax by 512
    ; (ax is storing the length of floppy_buffer)
    mov dx, 0x0
    mov cx, 0x200
    div cx

    ; Check if dx is 0 (dx is remainder of division)
    ; (if yes: go to "dx_zero")
    cmp dx, 0x0
    je dx_zero

    ; (if no)
    ; Increment ax by 1 (ax is number of sectors to write)
    add ax, 0x1

    dx_zero:
        ; Place ax in cx
        mov cx, ax
        mov ax, 0x0

        ; Move bx to the beginning of floppy_buffer
        mov bx, floppy_buffer

        ; Write to floppy loop for each sector
        write_loop:
            ; Write to floppy at track, side, sector
            push cx
            mov ah, 03h
            mov al, 0x1
            mov dl, 0x0
```

```
            mov cl, [track]
            mov dh, [side]
            mov ch, [sector]
            int 13h

            ; Increment track
            add cl, 0x1

            ; Check if track is less than 19
            ; (if yes: go to "write_continue")
            cmp cl, 0x13
            jl write_continue

            ; (if no)
            ; Reset track, increment side
            mov cl, 0x1
            add dh, 0x1

            ; Check if side is less than 2
            ; (if yes: go to "write_continue")
            cmp dh, 0x2
            jl write_continue

            ; (if no)
            ; Reset side, increment sector
            mov dh, 0x0
            add ch, 0x1

            ; Save track, side, sector
            write_continue:
                mov [track], cl
                mov [side], dh
                mov [sector], ch

            pop cx

            ; Decrement cx (number of sectors to write) by 1
            sub cx, 0x1

            ; Advance bx by 512
            add bx, 0x200

            ; Check if cx is 0
            ; (if no: go to "write_loop")
            cmp cx, 0x0
            jne write_loop

; (if yes)
; Call "print_write_success" from "prompts.asm"
call clear_screen

; Check if cf is set to cy
; (if no: go to "ktf_write_success")
jnc ktf_write_success
```

```
    ; (if yes)
    ; Print the error
    call print_error

    ; Go to "ktf_write_success"
    jmp ktf_write_success

ktf_write_success:
    ; Move si to the beginning of buffer
    mov si, buffer

; Clear buffer loop
clear_buffer:
    ; Place 0x0 at si
    mov byte [si], 0x0

    ; Advance si by 1
    add si, 0x1

    ; Check if si is at the end of buffer
    ; (if no: loop to "clear_buffer")
    cmp si, buffer + 0x100
    jne clear_buffer

    ; (if yes)
    ; Move di to the beginning of floppy_buffer
    mov di, floppy_buffer

; Clear floppy_buffer loop
clear_floppy_buffer:
    ; Place 0x0 at di
    mov byte [di], 0x0

    ; Advance di by 1
    add di, 0x1

    ; Check if di is pointing to a null
    ; (if no: loop to "clear_floppy_buffer")
    cmp byte [di], 0x0
    jne clear_floppy_buffer

    ; (if yes)
    ; Go to "menu" from "menu.asm"
    jmp menu

; Handle backspace
ktf_bakcspace:
    ; Get the current cursor position
    mov ah, 03h
    int 10h

    ; Check if the cursor is at the start of buffer
    ; (if yes: loop back to "ktf_input")
```

```
    cmp si, buffer
    je ktf_input

    ; (if no)
    ; Decrement si by 1
    sub si, 0x1

    ; Replace the last char with a null
    mov byte [si], 0x0

    ; Check if the cursor is at the start of line
    ; (if yes: go to "ktf_bakcspace_no_newline")
    cmp dl, 0x0
    jz ktf_bakcspace_no_newline

    ; (if no)
    ; Call "remove_last_char" from "screen_routines.asm"
    call remove_last_char

    ; Loop back to "ktf_input"
    jmp ktf_input

; Handle backspace with to previous line
ktf_bakcspace_no_newline:
    ; Call "remove_last_char_line" from "screen_routines.asm"
    call remove_last_char_line

    ; Loop back to "ktf_input"
    jmp ktf_input

; Handle escape key
escape:
    ; Move si to the beginning of buffer
    mov si, buffer

    ; Go to "clear_buffer"
    jmp clear_buffer
```

floppy_to_ram.asm: *This file contains routines to read data from the floppy disk and write it to RAM.*

```
floppy_to_ram:
    ; Call "clear_screen" from "screen_routines.asm"
    call clear_screen

    ; Get the number of sectors to read
    ftr_get_sectors:
        ; Clear the buffer
        mov ax, 0x0
        mov [num_buffer], ax

        ; Move the cursor to the top left corner
        mov ah, 02h
        mov dl, 0x0
        mov dh, 0xB
```

```
    int 10h

    ; Call "clear_row" from "screen_routines.asm"
    call clear_row

    ; Call "print_sectors_prompt" from "promts.asm"
    call print_sectors_prompt

    ; Call "read_num" from "string_convertors.asm"
    call read_num

    ; Check if escape was pressed
    ; (if yes: go to "menu")
    cmp al, 0x1B
    je menu

    ; (if no)
    ; Move the number from the buffer to "sectors"
    mov ax, [num_buffer]
    mov [sectors], ax

    mov ax, [sectors]
    cmp ax, 0x0
    je sectors_fault

    ; Check if sectors is less than 2881
    ; (if yes: go to "ftr_get_side")
    cmp ax, 0xB40
    jg sectors_fault

    ; (if no)
    ; Go to "ftr_get_side"
    jmp ftr_get_side

    ; Print a warning message
    sectors_fault:
        ; Call "print_sectors_warning" from "propts.asm"
        call print_sectors_warning

        ; Reset the buffers
        mov ax, 0x0
        mov [num_buffer], ax
        mov [sectors], ax

        ; Go to "ftr_get_sectors"
        jmp ftr_get_sectors

; Prompt the user for side
ftr_get_side:
    ; Clear the buffer
    mov ax, 0x0
    mov [num_buffer], ax

    ; Move the cursor to row 12, column 0
```

```
    mov ah, 02h
    mov dl, 0x0
    mov dh, 0xC
    int 10h

    ; Call "clear_row" from "screen_routines.asm"
    call clear_row

    ; Call "print_side_prompt" from "promts.asm"
    call print_side_prompt

    ; Call "read_num" from "string_convertors.asm"
    call read_num

    ; Check if read_num returned escape
    ; (if yes: go to "menu")
    cmp al, 0x1B
    je menu

    ; (if no)
    ; Move the number from the buffer to "side"
    mov ax, [num_buffer]
    mov [side], ax

    ; Check if side is less than 2
    ; (if yes: go to "ftr_get_track")
    mov ax, [side]
    cmp ax, 0x2
    jl ftr_get_track

    ; (if no)
    ; Call "print_side_warning" from "screen_routines.asm"
    call print_side_warning

    ; Reset the buffers
    mov ax, 0x0
    mov [num_buffer], ax
    mov [side], ax

    ; Go to "ftr_get_side"
    jmp ftr_get_side

; Prompt the user for track
ftr_get_track:
    ; Clear the buffer
    mov ax, 0x0
    mov [num_buffer], ax

    ; Move the cursor to row 13, column 0
    mov ah, 02h
    mov dl, 0x0
    mov dh, 0xD
    int 10h
```

```
    ; Call "clear_row" from "screen_routines.asm"
    call clear_row

    ; Call "print_track_prompt" from "promts.asm"
    call print_track_prompt

    ; Call "read_num" from "string_convertors.asm"
    call read_num

    ; Check if read_num returned escape
    ; (if yes: go to "menu")
    cmp al, 0x1B
    je menu

    ; (if no)
    ; Move the number from the buffer to "track"
    mov ax, [num_buffer]
    mov [track], ax

    ; Check if track is greater than 19
    ; (if yes: go to "ftr_track_fault")
    mov ax, [track]
    cmp ax, 0x12
    jg ftr_track_fault

    ; (if no)
    ; Check if track is 0
    ; (if yes: go to "ftr_track_fault")
    cmp ax, 0x0
    je ftr_track_fault

    ; (if no)
    ; Go to "ftr_get_sector"
    jmp ftr_get_sector

    ; Print a warning message
    ftr_track_fault:
        ; Call "print_track_warning" from "screen_routines.asm"
        call print_track_warning

        ; Reset the buffers
        mov ax, 0x0
        mov [num_buffer], ax
        mov [track], ax

        ; Go to "ftr_get_track"
        jmp ftr_get_track

; Prompt the user for sector
ftr_get_sector:
    ; Clear the buffer
    mov ax, 0x0
    mov [num_buffer], ax
```

```
    ; Move the cursor to row 14, column 0
    mov ah, 02h
    mov dl, 0x0
    mov dh, 0xE
    int 10h

    ; Call "clear_row" from "screen_routines.asm"
    call clear_row

    ; Call "print_sector_prompt" from "promts.asm"
    call print_sector_prompt

    ; Call "read_num" from "string_convertors.asm"
    call read_num

    ; Check if read_num returned escape
    ; (if yes: go to "menu")
    cmp al, 0x1B
    je menu

    ; (if no)
    ; Move the number from the buffer to "sector"
    mov ax, [num_buffer]
    mov [sector], ax

    ; Check if sector is less than 80
    ; (if yes: go to "ftr_get_address_1")
    mov ax, [sector]
    cmp ax, 0x50
    jl ftr_get_address_1

    ; (if no)
    ; Call "print_sector_warning" from "screen_routines.asm"
    call print_sector_warning

    ; Reset the buffers
    mov ax, 0x0
    mov [num_buffer], ax
    mov [sector], ax

    ; Go to "ftr_get_sector"
    jmp ftr_get_sector

; Read the address
ftr_get_address_1:
    ; Clear the buffer
    mov ax, 0x0
    mov [num_buffer], ax

    ; Move the cursor to row 15, column 0
    mov ah, 02h
    mov dl, 0x0
    mov dh, 0xF
    int 10h
```

```
        ; Call "clear_row" from "screen_routines.asm"
        call clear_row

        ; Call "print_address_prompt" from "promts.asm"
        call print_address_prompt

        ; Call "read_address" from "string_convertors.asm"
        call read_address

        ; Check if read_address returned escape
        ; (if yes: go to "menu")
        cmp al, 0x1B
        je menu

        ; (if no)
        ; Move the number from the buffer to "xxxx"
        mov ax, [num_buffer]
        mov [xxxx], ax

    ; Read the second part of the address
    ftr_get_address_2:
        ; Clear the buffer
        mov ax, 0x0
        mov [num_buffer], ax

        ; Print to screen ":"
        mov ah, 0Eh
        mov al, ':'
        int 10h

        ; Call "read_address" from "string_convertors.asm"
        call read_address

        ; Check if read_address returned escape
        ; (if yes: go to "menu")
        cmp al, 0x1B
        je menu

        ; (if no)
        ; Move the number from the buffer to "yyyy"
        mov ax, [num_buffer]
        mov [yyyy], ax

; Read the floppy
ftr_read_floppy:
    ; Read the floppy at the specified address
    mov ah, 02h
    mov dl, 0x0
    mov al, [sectors]
    mov cl, [track]
    mov dh, [side]
    mov ch, [sector]
```

```
    mov bx, [xxxx]
    mov es, bx
    mov bx, [yyyy]

    int 13h

    ; Call "clear_screen" from "screen_routines.asm"
    call clear_screen

    ; Check if the read was successful
    ; (if yes: go to "ftr_read_success")
    jnc ftr_read_success

    ; (if no)
    ; Call "print_error" from "screen_routines.asm"
    call print_error

; Print the floppy to the screen
ftr_read_success:
    mov dx, [sectors]

    ; Move pointer to the start of the buffer
    mov bp, [xxxx]
    mov es, bp
    mov bp, [yyyy]

    ; Print the floppy to the screen
    ftr_print_loop:
        push dx
        call clear_screen
        mov ax, 1301h
        mov bx, 0x7
        mov cx, 0x200
        mov dh, 0x0
        mov dl, 0x0
        int 10h
        pop dx

        sub dx, 0x1
        add bp, 0x200

        ; Wait for a key to be pressed
        mov ah, 00h
        int 16h

        ; Check if all sectors have been printed
        ; (if no: go to "ftr_print_loop")
        cmp dx, 0x0
        jne ftr_print_loop

    ; (if yes)
    ; Go to "menu" in "menu.asm"
    jmp menu
```

ram_to_floppy.asm: *This file contains routines to read data from RAM and write it to the floppy disk.*

```
ram_to_floppy:
    ; Call "clear_screen" from "screen_routines.asm"
    call clear_screen

    ; Prompt user to enter volume number
    rtf_get_volume:
        ; Clear "num_buffer"
        mov ax, 0x0
        mov [num_buffer], ax

        ; Set cursor position
        mov ah, 02h
        mov dl, 0x0
        mov dh, 0x0
        int 10h

        ; Call "clear_row" from "screen_routines.asm"
        call clear_row

        ; Call "print_volume_prompt" from "prompts.asm"
        call print_volume_prompt

        ; Call "read_num" from "string_convertors.asm"
        call read_num

        ; Check if read_num returned escape
        ; (if yes: go to "menu")
        cmp al, 0x1B
        je menu

        ; (if no)
        ; Move "num_buffer" to "volume"
        mov ax, [num_buffer]
        mov [volume], ax

    ; Prompt user to enter side number
    rtf_get_side:
        ; Clear "num_buffer"
        mov ax, 0x0
        mov [num_buffer], ax

        ; Set cursor position
        mov ah, 02h
        mov dl, 0x0
        mov dh, 0xC
        int 10h

        ; Call "clear_row" from "screen_routines.asm"
        call clear_row

        ; Call "print_side_prompt" from "prompts.asm"
        call print_side_prompt
```

```
    ; Call "read_num" from "string_convertors.asm"
    call read_num

    ; Check if read_num returned escape
    ; (if yes: go to "menu")
    cmp al, 0x1B
    je menu

    ; (if no)
    ; Move "num_buffer" to "side"
    mov ax, [num_buffer]
    mov [side], ax

    ; Check if "side" is less than 2
    ; (if yes: go to "rtf_get_side")
    mov ax, [side]
    cmp ax, 0x2
    jl rtf_get_track

    ; (if no)
    ; Call "print_side_warning" from "prompts.asm"
    call print_side_warning

    ; Clear the buffers
    mov ax, 0x0
    mov [num_buffer], ax
    mov [side], ax

    ; Go to "rtf_get_side"
    jmp rtf_get_side

; Prompt user to enter track number
rtf_get_track:
    ; Clear "num_buffer"
    mov ax, 0x0
    mov [num_buffer], ax

    ; Set cursor position
    mov ah, 02h
    mov dl, 0x0
    mov dh, 0xD
    int 10h

    ; Call "clear_row" from "screen_routines.asm"
    call clear_row

    ; Call "print_track_prompt" from "prompts.asm"
    call print_track_prompt

    ; Call "read_num" from "string_convertors.asm"
    call read_num

    ; Check if read_num returned escape
```

```
    cmp al, 0x1B
    je menu

    ; (if no)
    ; Move "num_buffer" to "track"
    mov ax, [num_buffer]
    mov [track], ax

    ; Check if "track" is greater than 18
    ; (if yes: go to "rtf_track_fault")
    mov ax, [track]
    cmp ax, 0x12
    jg rtf_track_fault

    ; (if no)
    ; Check if "track" is 0
    ; (if yes: go to "rtf_track_fault")
    cmp ax, 0x0
    je rtf_track_fault

    ; (if no)
    ; Go to "rtf_get_sector"
    jmp rtf_get_sector

    ; Print warning message
    rtf_track_fault:
        ; Call "print_track_warning" from "prompts.asm"
        call print_track_warning

        ; Clear the buffers
        mov ax, 0x0
        mov [num_buffer], ax
        mov [track], ax

        ; Go to "rtf_get_track"
        jmp rtf_get_track

; Prompt user to enter sector number
rtf_get_sector:
    mov ax, 0x0
    mov [num_buffer], ax

    ; Set cursor position
    mov ah, 02h
    mov dl, 0x0
    mov dh, 0xE
    int 10h

    ; Call "clear_row" from "screen_routines.asm"
    call clear_row

    ; Call "print_sector_prompt" from "prompts.asm"
    call print_sector_prompt
```

```
    ; Call "read_num" from "string_convertors.asm"
    call read_num

    ; Check if read_num returned escape
    cmp al, 0x1B
    je menu

    ; (if no)
    ; Move "num_buffer" to "sector"
    mov ax, [num_buffer]
    mov [sector], ax

    ; Check if "sector" is less than 80
    ; (if yes: go to "rtf_get_address_1")
    mov ax, [sector]
    cmp ax, 0x50
    jl rft_get_address_1

    ; (if no)
    ; Call "print_sector_warning" from "prompts.asm"
    call print_sector_warning

    ; Clear the buffers
    mov ax, 0x0
    mov [num_buffer], ax
    mov [sector], ax

    ; Go to "rtf_get_sector"
    jmp rtf_get_sector

; Prompt user to enter address
rft_get_address_1:
    ; Clear "num_buffer"
    mov ax, 0x0
    mov [num_buffer], ax

    ; Set cursor position
    mov ah, 02h
    mov dl, 0x0
    mov dh, 0xF
    int 10h

    ; Call "clear_row" from "screen_routines.asm"
    call clear_row

    ; Call "print_address_prompt" from "prompts.asm"
    call print_address_prompt

    ; Call "read_address" from "string_convertors.asm"
    call read_address

    ; Check if read_address returned escape
    ; (if yes: go to "menu")
    cmp al, 0x1B
```

```
        je menu

        ; (if no)
        ; Move "num_buffer" to "xxxx"
        mov ax, [num_buffer]
        mov [xxxx], ax

    ; Get the second part of the address
    rft_get_address_2:
        ; Clear "num_buffer"
        mov ax, 0x0
        mov [num_buffer], ax

        ; Print ":"
        mov ah, 0Eh
        mov al, ':'
        int 10h

        ; Call "read_address" from "string_convertors.asm"
        call read_address

        ; Check if read_address returned escape
        ; (if yes: go to "menu")
        cmp al, 0x1B
        je menu

        ; (if no)
        ; Move "num_buffer" to "yyyy"
        mov ax, [num_buffer]
        mov [yyyy], ax

; Read form RAM and write to floppy
rtf_read_ram:
    ; Call "clear_screen" from "screen_routines.asm"
    call clear_screen

    ; Print to screen
    mov ax, 1301h
    mov bl, 0x7
    mov cx, [volume]
    mov bp, [xxxx]
    mov es, bp
    mov bp, [yyyy]
    int 10h

    ; Move the data to the floppy
    mov ah, 03h
    mov cl, [track]
    mov dh, [side]
    mov ch, [sector]
    mov bx, [xxxx]
    mov es, bx
    mov bx, [yyyy]
    int 13h
```

```
    ; Wait for key press
    push ax
    mov ah, 00h
    int 16h
    pop ax

    ; Call "clear_screen" from "screen_routines.asm"
    call clear_screen

    ; Check if the read was successful
    ; (if yes: go to "rtf_read_success")
    jnc rtf_write_success

    ; (if no)
    ; Call "print_error" from "screen_routines.asm"
    call print_error

rtf_write_success:
    ; Go to "menu" in "menu.asm"
    jmp menu
```

floppy_params.asm: *This file reserves memory for the floppy and RAM parameters.*

```
section .data
    ; Buffer for read_num and read_address
    num_buffer dw 0x0

    ; Buffers for floppy parameters
    sectors dw 0x0
    write_times dw 0x0
    volume dw 0x0
    side dw 0x0
    track dw 0x0
    sector dw 0x0

    ; Buffers for address
    xxxx dw 0x0
    yyyy dw 0x0

    ; Empty clean row
    clean_row times 0x50 db 0x0

    ; Buffer for text input
    buffer times 0x100 db 0x0

    ; Buffer for floppy write
    floppy_buffer db 0x0
```

Screenshots:

Figure 1 *– Main menu*

Figure 2 *– Keyboard to floppy menu*

Figure 3 *– Floppy to RAM menu*

Figure 4 – *Floppy to RAM output no.1*

Figure 5 *– Floppy to RAM output no.2*

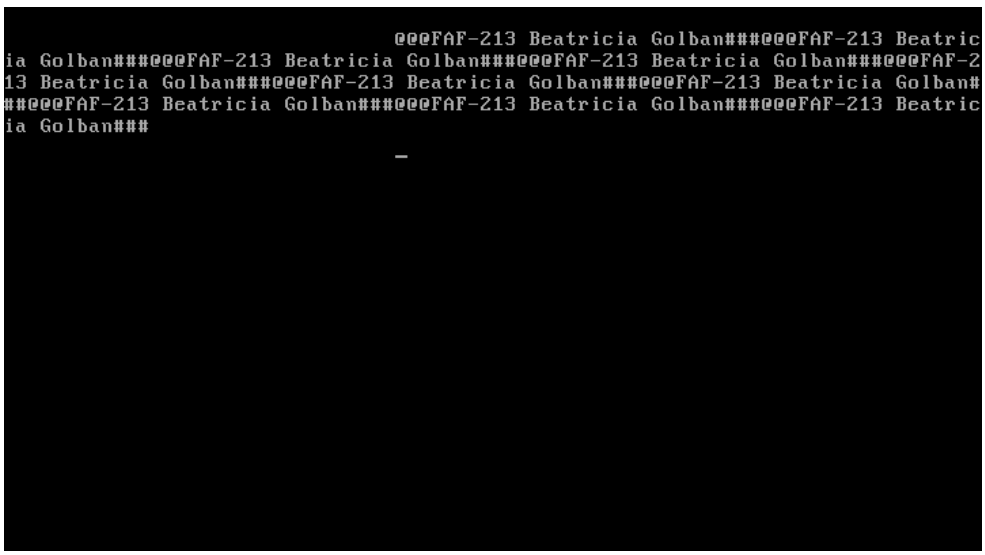

Figure 6 *– Floppy to RAM output no.3*

Figure 7 *– RAM to floppy menu*

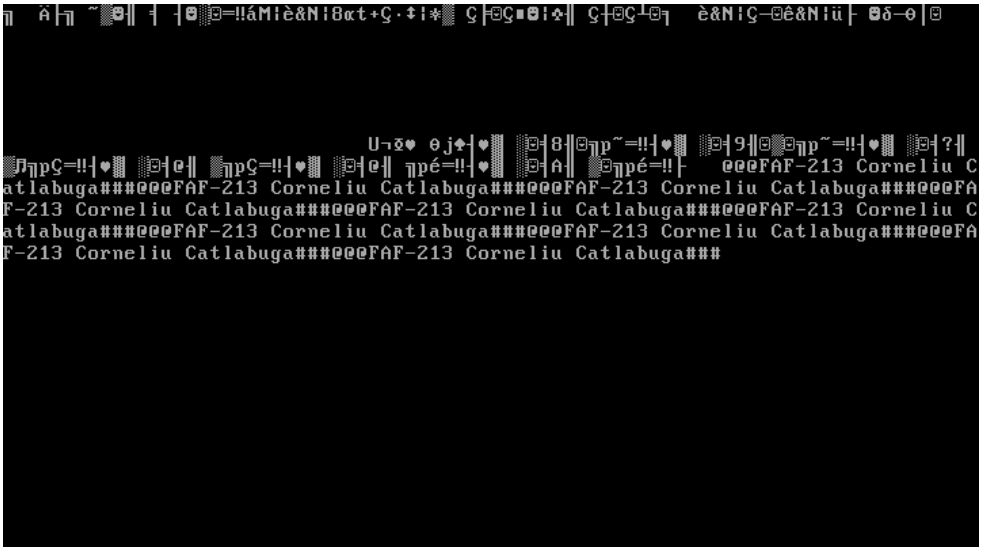

Figure 8 *– RAM to floppy output*

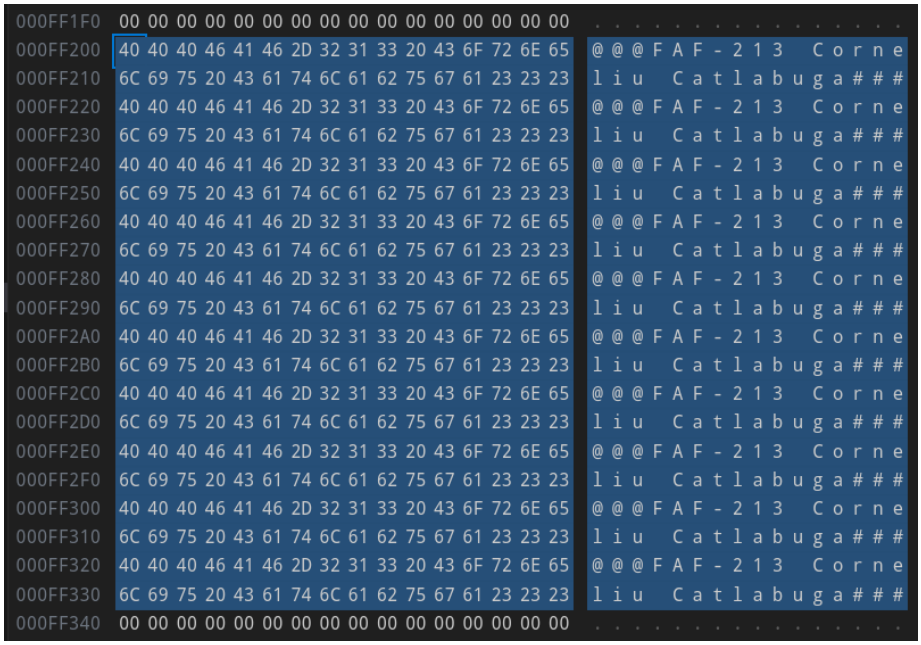

Figure 9 *– Task 1.1.1: Sector 2041 Name String*

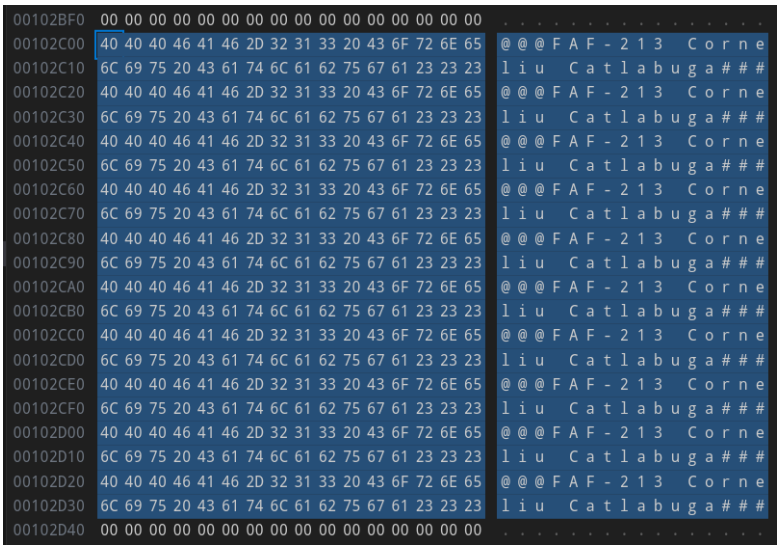

Figure 10 – *Task 1.1.2: Sector 2070 Name String*

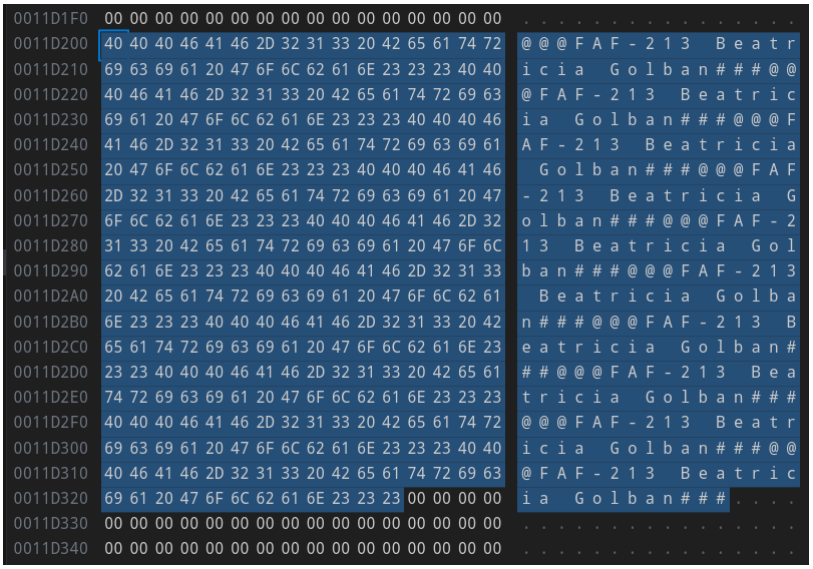

Figure 11 – *Task 1.2.1: Sector 2281 Name String*

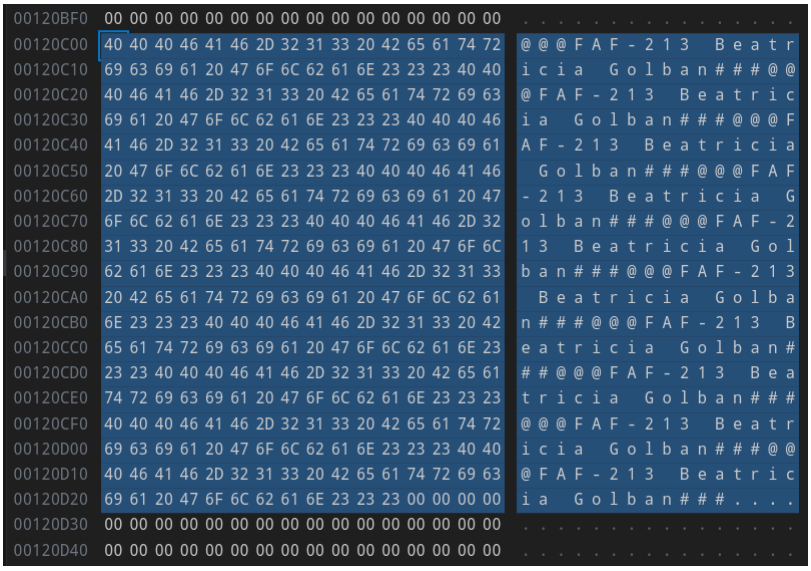

Figure 12 – *Task 1.2.2: Sector 2310 Name String*

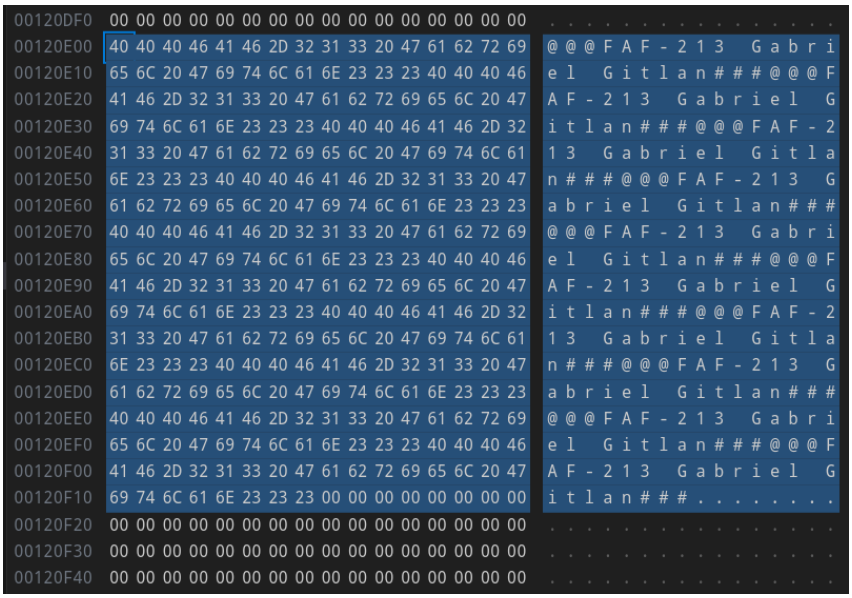

Figure 13 – *Task 1.3.1: Sector 2311 Name String*

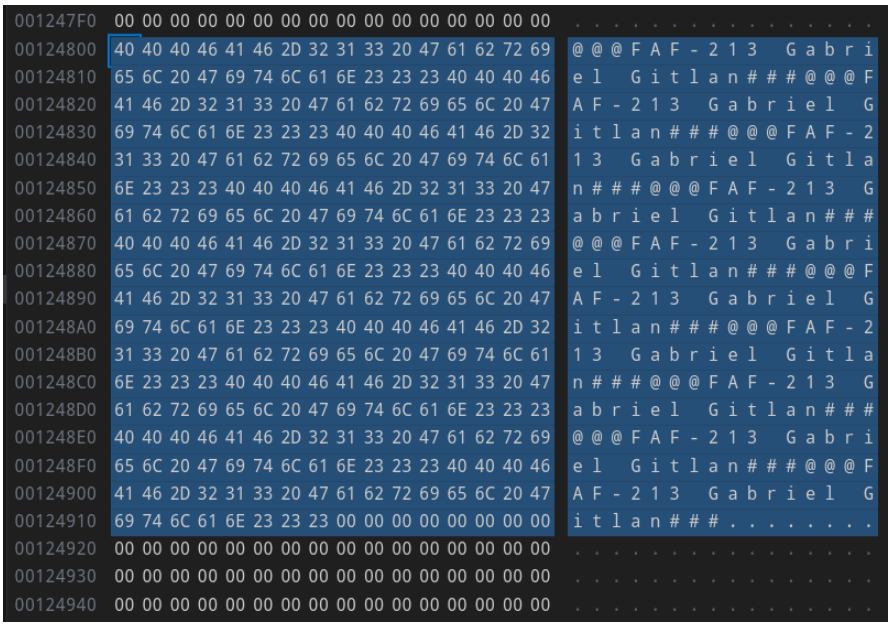

Figure 14 – *Task 1.3.2: Sector 2340 Name String*

Conclusions:

This project in assembly language serves as an example of the breadth and depth of knowledge needed to operate computer systems at a low level. The system adeptly incorporates several intricate features, such as direct hardware manipulation and bootloading, showcasing a strong understanding of system-level programming. Despite the inherent difficulty of assembly language, the construction of a user-friendly menu system demonstrates the potential to build intuitive interfaces even in low-level contexts. The project's modular structure, which divides its many operations into separate files, exemplifies a best practice in software development and improves the code's readability and maintainability.

The project's emphasis on effective memory management, as seen by the temporary data storage of buffers, shows a deep comprehension of resource optimization — a crucial component of low-level programming. Moreover, the addition of error handling procedures highlights the program design's resilience by demonstrating a careful foresight of possible problems. A high degree of expertise in assembly language programming is demonstrated by the ability to convert between multiple data kinds, manage complicated tasks like reading and writing to a floppy drive, and handle keyboard inputs.

To sum up, this project provides an outstanding example of the versatility and strength of assembly language in system-level programming. It is a thorough examination of low-level programming and how it may be used to develop reliable, effective, and user-friendly software. It not only showcases the user's proficiency in assembly language but also their understanding of computer systems and hardware manipulation.

References:

1. Laboratory work GitHub repository: https://github.com/muffindud/SO_Team_2
2. Image Creator Script: https://github.com/muffindud/ImgCreator
3. INT 10H Video Services: http://www.techhelpmanual.com/113-int_10h__video_services.html
4. INT 13H BIOS Disk I/O: http://www.techhelpmanual.com/185-int_13h__bios_disk_i_o.html
5. INT 16 Keyboard Services: http://www.techhelpmanual.com/228-int_16h__keyboard_services.html
6. NASM Quick Reference: https://www.cs.uaf.edu/2017/fall/cs301/reference/x86_64.html