Ministerul Educației, Culturii și Cercetării al Republicii Moldova
Universitatea Tehnică a Moldovei
Facultatea Calculatoare, Informatică și Microelectronică
Departamentul Ingineria Software și Automatică

Raport
pentru lucrare de laborator Nr. 3
la cursul Sisteme de Operare
“Floppy Disk I/O operations”

A efectuat: Arteom KALAMAGHIN, FAF-211
A verificat: Rostislav CĂLIN

Chișinău - 2023

Subject: Floppy disk I/O operations

Tasks:

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

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 the floppy disk "N" sectors starting at the address {Head, Track, Sector} and transferring this data to the RAM memory starting at the 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.*

Implementation:

```
option1:
; display the key read

mov ah, 0eh                  ; Set up AH register for BIOS teletype output
int 10h                      ; Print the character in AL (previously read)

mov al, 2eh                  ; Set AL register to ASCII character '.'
int 10h                      ; Print the character

; print "STRING = "

call get_cursor_pos          ; Call function to get cursor position

inc dh                       ; Increment DH (row position)
mov dl, 0                    ; Set DL (column position) to 0

mov ax, 0                    ; Set AX register to 0
mov es, ax                   ; Set ES (Extra Segment) register to 0
mov bp, in_awaits_str1       ; Set BP register to point to in_awaits_str1

mov bl, 07h                  ; Set BL register for display attribute (color)
mov cx, str1_awaits_len1     ;  Set  CX  register  to  str1_awaits_len1  (length  of  the
string)

mov ax, 1301h                ; Set up AH for BIOS print string and advance cursor
int 10h                      ; Call BIOS interrupt 10h to display "STRING = "

; read user input (str)

call read_input              ; Call function to read user input

; save the string to its own buffer

mov si, storage_buffer       ; Set SI to point to storage_buffer
mov di, string               ; Set DI to point to string buffer

char_copy_loop:
mov al, [si]                 ; Move the byte from address pointed by SI to AL
mov [di], al                 ; Move the byte in AL to the address pointed by DI
inc si                       ; Increment SI
inc di                       ; Increment DI

cmp byte [si], 0             ; Compare the byte at SI with 0
                             ; (end of string marker)
jne char_copy_loop           ; Jump back to char_copy_loop if not end of string

; print "N = "

call get_cursor_pos          ; Get cursor position

inc dh                       ; Increment DH (row position)
mov dl, 0                    ; Set DL (column position) to 0

mov ax, 0                    ; Set AX register to 0
mov es, ax                   ; Set ES (Extra Segment) register to 0
mov si, in_awaits_str1       ; Set SI to point to in_awaits_str1
add si, str1_awaits_len1     ; Point SI to the second part in_awaits_str1
mov bp, si                   ; Set BP to point to the end of in_awaits_str1

mov bl, 07h                  ; Set BL register for display attribute (color)
mov cx, str1_awaits_len2     ;  Set  CX  register  to  str1_awaits_len2  (length  of  the
string)

mov ax, 1301h                ; Set up AH for BIOS print string and advance cursor
int 10h                      ; Call BIOS interrupt 10h to print "N = "

; read user input (n)

call read_input              ; Call function to read user input

; convert ascii read to an integer

mov di, nhts                 ; Set DI to point to nhts
mov si, storage_buffer       ; Set SI to point to storage_buffer
call atoi                    ; Convert ASCII characters to integer

; read HTS

call read_hts_address        ; Call the procedure to read HTS address

; prepare writing buffer

mov si, string               ; Set SI to point to string
call fill_storage_buffer     ; Call function to fill storage buffer

; calculate the number of sectors to write

xor dx, dx                   ; Clear DX register
mov ax, [storage_curr_size] ; Move value from storage_curr_size to AX
mov bx, 512                  ; Set BX to 512 (sector size)
div bx                       ; Divide AX by BX, quotient in AX, remainder in DX

; write to the floppy

push ax                      ; Save AX on the stack

mov ax, 0                    ; Clear AX register
mov es, ax                   ; Set ES register to 0 (clears ES)
mov bx, storage_buffer       ; Set BX to point to storage_buffer

pop ax                       ; Restore AX from the stack

mov ah, 03h                  ; Set up AH register for BIOS Write Sector
inc al                       ; Increment AL to fit the characters in “remainder”
mov ch, [nhts + 4]           ; Set CH to nhts + 4 (head number storage)
mov cl, [nhts + 6]           ; Set CL to nhts + 6 (track number storage)
mov dh, [nhts + 2]           ; Set DH to nhts + 2 (sector number storage)
mov dl, 0                    ; Set DL to 0 (drive number)

int 13h                      ; Call BIOS interrupt 13h to write sectors

; print error code

call display_error_code      ; Display disk I/O operation error code

; print string read

mov si, string               ; Set SI to point to string
call print_buff              ; Output the last string read

jmp _terminate               ; End the execution cycle
```

The first option takes the user input and writes it to the floppy. It first prints all the information a user has to enter and then saves the string input, converts the integer input from ASCII to its numeric representation. After that it reads an address associated with "HTS", which is Head, Track and Sector, prepares a buffer for data writing, calculates the number of sectors to write based on the buffer size, and finally writes data to a floppy disk.

```
option2:
; display the key read

mov ah, 0eh                 ; Set up AH register for BIOS teletype output
int 10h                     ; Print the character in AL (previously read)

mov al, 2eh                 ; Set AL register to ASCII character '.'
int 10h                     ; Print the character

; read RAM address XXXX:YYYY "

call read_ram_address       ; Call the procedure to read RAM address

; read HTS

call read_hts_address       ; Call the procedure to read HTS address

; print "N = "

call get_cursor_pos         ; Get cursor position

inc dh                      ; Increment DH (row position)
mov dl, 0                   ; Set DL (column position) to 0

mov ax, 0                   ; Set AX register to 0
mov es, ax                  ; Set ES (Extra Segment) register to 0
mov si, in_awaits_str1      ; Set SI to point to in_awaits_str1
add si, str1_awaits_len1    ; Point SI to the second part in_awaits_str1
mov bp, si                  ; Set BP to point to the end of in_awaits_str1

mov bl, 07h                 ; Set BL register for display attribute (color)
mov cx, str1_awaits_len2    ; Set CX register to str1_awaits_len2
                            ; (length of the string)

mov ax, 1301h               ; Set up AH for BIOS print string and advance cursor
int 10h                     ; Call BIOS interrupt 10h to print "N = "

; read user input (n)

call read_input             ; Call function to read user input

; convert ascii read to an integer

mov di, nhts                ; Set DI to point to nhts
mov si, storage_buffer      ; Set SI to point to storage_buffer
call atoi                   ; Convert ASCII characters to integer

; read data from floppy

mov es, [address]           ; Set ES to the value stored at [address]
mov bx, [address + 2]       ; Set BX to the value stored at [address + 2]

mov ah, 02h                 ; Set up AH for BIOS Read Sector
mov al, [nhts]              ; Set AL to value at nhts
mov ch, [nhts + 4]          ; Set CH to nhts + 4 (head number storage)
mov cl, [nhts + 6]          ; Set CL to nhts + 6 (track number storage)
mov dh, [nhts + 2]          ; Set DH to nhts + 2 (sector number storage)
mov dl, 0                   ; Set DL to 0 (drive number)

int 13h                     ; Call BIOS interrupt 13h to read sectors

; print error code

call display_error_code     ; Display disk I/O operation error code

; print the data read

call get_cursor_pos         ; Get cursor position

inc dh                      ; Increment DH (row position)
mov dl, 0                   ; Set DL (column position) to 0

mov es, [address]           ; Set SEGMENT part of the memory address
mov bp, [address + 2]       ; Set OFFSET part of the memory address

mov bl, 07h                 ; Output in light gray
mov cx, 512                 ; The first 512 characters read

mov ax, 1301h               ; Set up AH for BIOS print string and advance cursor
int 10h                     ; Call BIOS interrupt 10h to print "N = "

; call paginated_output     ; Not implemented yet ;)

jmp _terminate              ; End the execution cycle
```

The second option writes from floppy to RAM. First of all, it displays the information the user has to enter. It takes the RAM address (Segment and Offset), where the user wants to read. After that it scans an address associated with "HTS", which is Head, Track and Sector. It reads the data from floppy and writes it to RAM.

```
option3:
; display the key read

mov ah, 0eh                 ; Set up AH register for BIOS teletype output
int 10h                     ; Print the character in AL (previously read)

mov al, 2eh                 ; Set AL register to ASCII character '.'
int 10h                     ; Print the character

; read RAM address XXXX:YYYY "

call read_ram_address       ; Call the procedure to read RAM address

; read HTS

call read_hts_address       ; Call the procedure to read HTS address

; print "N = "

call get_cursor_pos         ; Get cursor position

inc dh                      ; Increment DH (row position)
mov dl, 0                   ; Set DL (column position) to 0

mov ax, 0                   ; Set AX register to 0
mov es, ax                  ; Set ES (Extra Segment) register to 0
mov si, in_awaits_str1      ; Set SI to point to in_awaits_str1
add si, str1_awaits_len1    ; Point SI to the second part in_awaits_str1
```

```
mov bp, si                  ; Set BP to point to the end of in_awaits_str1

mov bl, 07h                 ; Set BL register for display attribute (color)
mov cx, str1_awaits_len2    ; Set CX register to str1_awaits_len2 (length of the
string)

mov ax, 1301h               ; Set up AH for BIOS print string and advance cursor
int 10h                     ; Call BIOS interrupt 10h to print "N = "

; read user input (n)

call read_input             ; Call function to read user input

; convert ascii read to an integer

mov di, nhts                ; Set DI to point to nhts (n storage)
mov si, storage_buffer      ; Set SI to point to storage_buffer
call atoi                   ; Convert ASCII characters to integer

; print the data to write

call get_cursor_pos         ; Call a routine to get the current cursor position

inc dh                      ; Increment the value in DH (row position) to move
                            ; the cursor down by one row
mov dl, 0                   ; Move the column position (DL) to the beginning
                            ; (column 0)
mov es, [address]           ; Load the SEGMENT part of the memory address into ES
mov bp, [address + 2]       ; Load the OFFSET part of the memory address into BP

mov bl, 07h                 ; Set the display attribute for the text to white on
                            ; black background
mov cx, [nhts]              ; Load the number of sectors to display from memory
                            ; into CX
mov ax, 1301h               ; Set up AH for BIOS print string and advance cursor
int 10h                     ; Call the BIOS video interrupt to execute the
                            ; function specified in AX

; calculate the number of sectors to write

xor dx, dx                  ; Clear DX register
mov ax, [nhts]              ; Move value from nhts to AX
mov bx, 512                 ; Set BX to 512 (sector size)
div bx                      ; Divide AX by BX, quotient in AX, remainder in DX

; write data to floppy

mov es, [address]           ; Set ES to the value stored at [address] - SEGMENT
mov bx, [address + 2]       ; Set BX to the value stored at [address + 2] -
                            ; OFFSET
mov ah, 03h                 ; Set up AH for BIOS Write Sector
inc al                      ; Increment AL
mov al, [nhts]              ; Set AL to value at nhts
mov ch, [nhts + 4]          ; Set CH to nhts + 4 (head number storage)
mov cl, [nhts + 6]          ; Set CL to nhts + 6 (track number storage)
mov dh, [nhts + 2]          ; Set DH to nhts + 2 (sector number storage)
mov dl, 0                   ; Set DL to 0 (drive number)
int 13h                     ; Call BIOS disk operation

call display_error_code     ; Display disk I/O operation error code

jmp _terminate              ; End the execution cycle
```

The third option gives us the possibility to write from RAM back to floppy. In this way we can write the information from one sector to another directly from RAM. Like in the second option, the program reads RAM address the user entered, reads HTS, calculates the number of sectors to write and then writes to floppy.

```
read_hts_address:

; print "{H, T, S} (one value per line):"
call get_cursor_pos          ; Call subroutine to get cursor position

inc dh                       ; Increment DH (move to the next line)
mov dl, 0                    ; Move cursor to the start of the line

mov ax, 0                    ; Clear AX register
mov es, ax                   ; Set ES to 0 (video memory segment)
mov si, in_awaits_str1       ; Set SI to in_awaits_str1 address (string)
add si, str1_awaits_len1     ; Add length of first prompt
add si, str1_awaits_len2     ; Add length of second prompt
                             ; (end up on the third part)
mov bp, si                   ; Set BP to the updated SI position (string)

mov bl, 07h                  ; Set BL for display attribute (color)
mov cx, str1_awaits_len3     ; Set CX to str1_awaits_len3 (length of the string)

mov ax, 1301h                ; Set up AH for BIOS video scroll function
int 10h                      ; Call BIOS interrupt 10h to print the prompt

; read user input (h)

call break_line_with_prompt ; Call subroutine to move cursor and print prompt
call read_input              ; Call subroutine to read user input

; convert ascii read to an integer

mov di, nhts + 2             ; Set DI to nhts + 2 (for 'H')
mov si, storage_buffer       ; Set SI to storage_buffer (user input)
call atoi                    ; Convert ASCII input to an integer

; read user input (t)

call break_line_with_prompt ; Move cursor and print the prompt
call read_input              ; Read user input

; convert ascii read to an integer

mov di, nhts + 4             ; Set DI to nhts + 4 (for 'T')
mov si, storage_buffer       ; Set SI to storage_buffer (user input)
call atoi                    ; Convert ASCII input to an integer

; read user input (s)

call break_line_with_prompt ; Move cursor and print the prompt
call read_input              ; Read user input

; convert ascii read to an integer

mov di, nhts + 6             ; Set DI to nhts + 6 (for 'S')
mov si, storage_buffer       ; Set SI to storage_buffer (user input)
call atoi                    ; Convert ASCII input to an integer

ret                          ; Return from the subroutine
```

This piece of code guides the user to input values for “H”, “T”, and “S”. It reads each input and converts the ASCII characters to integers, storing them in memory for later use. The subroutine then returns, having collected and stored the values for “H”, “T”, and “S” in specific memory locations.

```
atoi:
atoi_conv_loop:
       cmp byte [si], 0      ; Compare the byte at SI with 0
       je atoi_conv_done     ; If it's null, jump to atoi_conv_done

       xor ax, ax            ; Clear AX register
       mov al, [si]          ; Move byte at SI to AL
       sub al, '0'           ; Convert ASCII to integer by subtracting '0'

       mov bx, [di]          ; Move the value accumulated to BX
       imul bx, 10           ; And multiply by 10 to shift digits to the left
       add bx, ax            ; Add the value in AX to BX
       mov [di], bx          ; Store the result back at [DI]


       inc si                ; Increment SI to point to the next digit-char

       jmp atoi_conv_loop    ; Jump to the beginning of the loop

atoi_conv_done:
       ret                   ; Return from the subroutine

atoh:
atoh_conv_loop:
       cmp byte [si], 0      ; Compare the byte at SI with 0
       je atoh_conv_done     ; If it's null, jump to atoh_conv_done

       xor ax, ax            ; Clear AX register
       mov al, [si]          ; Move byte at SI to AL
       cmp al, 65            ; Compare AL with ASCII 'A' (65)
       jl conv_digit         ; If less than 'A', jump to conv_digit

       conv_letter:
       sub al, 55                   ; Convert ASCII letter to hexadecimal value
       jmp atoh_finish_iteration    ; Jump to atoh_finish_iteration

       conv_digit:
       sub al, 48            ; Convert ASCII digit to integer

       atoh_finish_iteration:
       mov bx, [di]          ; Move the value accumulated to BX
       imul bx, 16           ; And multiply by 16 to shift digits to the left
       add bx, ax            ; Add the value in AX to BX
       mov [di], bx          ; Store the result back at [DI]

       inc si                ; Increment SI to point to the next character

       jmp atoh_conv_loop    ; Jump to the beginning of the loop

atoh_conv_done:
       ret                   ; Return from the subroutine
```

These two functions, **atoi** and **atoh**, are responsible for converting ASCII characters to integer:

- **atoi** ~ converts ASCII characters representing digits to their numeric integer values. It iterates through the string until it encounters a null terminator. For each character, it subtracts the ASCII value of '0' to obtain the actual digit value and constructs an integer from the individual digits.
- **atoh** ~ converts ASCII characters to their hexadecimal numeric values. Similar to **atoi**, it iterates through the string until it reaches the null terminator. For letters, it adjusts the ASCII values to convert them to their hexadecimal equivalents and constructs the hexadecimal value.

Both functions use a destination index (DI) to store the resulting converted values and utilize iterative loops to process the entire string before terminating and returning the converted values.

```
read_ram_address:

; print "SEGMENT (XXXX) = "

call get_cursor_pos          ; Call subroutine to get cursor position

inc dh                       ; Increment DH (move to the next line)
mov dl, 0                    ; Move cursor to the start of the line

mov ax, 0                    ; Clear AX register
mov es, ax                   ; Set ES to 0 (video memory segment)
mov bp, in_awaits_str2       ; Set BP to the in_awaits_str2 address (string)

mov bl, 07h                  ; Set BL for display attribute (color)
mov cx, str2_awaits_len1     ; Set CX to str2_awaits_len1 (length of the string)

mov ax, 1301h                ; Set up AH for BIOS print string and advance cursor
int 10h                      ; Call BIOS interrupt 10h to print the prompt

; read user input (segment)

call read_input              ; Call subroutine to read user input

; convert ascii read to a hex

mov di, address              ; Set DI to address (for SEGMENT)
mov si, storage_buffer       ; Set SI to storage_buffer (user input)
call atoh                    ; Convert ASCII input to hexadecimal

; print "OFFSET (YYYY) = "

call get_cursor_pos          ; Call subroutine to get cursor position

inc dh                       ; Increment DH (move to the next line)
mov dl, 0                    ; Move cursor to the start of the line

mov ax, 0                    ; Clear AX register
mov es, ax                   ; Set ES to 0 (video memory segment)
mov si, in_awaits_str2       ; Set SI to the in_awaits_str2 address (string)
add si, str2_awaits_len1     ; Add the length of the first prompt to SI
mov bp, si                   ; Set BP to the updated SI position

mov bl, 07h                  ; Set BL for display attribute (color)
mov cx, str2_awaits_len2     ; Set CX to str2_awaits_len2 (length of the string)

mov ax, 1301h                ; Set up AH for BIOS print string and advance cursor
int 10h                      ; Call BIOS interrupt 10h to print the prompt
```

```
; read user input (offset)

call read_input             ; Call subroutine to read user input

; convert ascii read to a hex

mov di, address + 2         ; Set DI to address + 2 (for OFFSET)
mov si, storage_buffer      ; Set SI to storage_buffer (user input)
call atoh                   ; Convert ASCII input to hexadecimal

ret                         ; Return from the subroutine
```

This piece of code defines a function that guides the user to input a segment and an offset in hexadecimal format. It prints prompts for the segment and offset, reads user input for each, converts the ASCII characters to hexadecimal, and stores these values in memory for later use. Finally, it returns after saving the segment and offset in specific memory locations.

```
fill_storage_buffer:
push si                     ; Preserve SI register
mov cx, 0                   ; Initialize CX register to 0

; Find the end of the string in SI

find_end:
       cmp byte [si], 0     ; Compare the byte at SI with 0
       je end_found         ; If it's null, jump to end_found

       inc si               ; Move to the next character
       inc cx               ; Increment CX (string length)

       jmp find_end         ; Continue searching for the end of the string

       end_found:
       pop si                      ; Restore SI register
       mov di, storage_buffer      ; Set DI to point to storage_buffer

; Copy string from SI to DI (storage_buffer) N times

copy_string_to_buffer_loop:
       push cx              ; Preserve CX
       push si              ; Preserve SI

       rep movsb            ; Move buffer od data from SI to DI

       pop si               ; Restore SI
       pop cx               ; Restore CX

       dec word [nhts]                    ; Decrement word at nhts
                                          ;(number of characters)
       add word [storage_curr_size], cx   ; Add CX to storage_curr_size

       cmp word [nhts], 0                 ; Compare word at nhts with 0
       jg copy_string_to_buffer_loop      ; If greater, continue copying

; Calculate padding with null character to get to sector size and how many sectors
on the floppy it is going to occupy

push di                     ; Preserve DI
sub di, storage_buffer      ; Calculate the offset between DI and storage_buffer
mov ax, di                  ; Move DI offset to AX
```

```
pop di                      ; Restore DI

xor dx, dx                  ; Clear DX register
mov bx, 512                 ; Set BX to 512 (sector size)
div bx                      ; Divide AX by BX (calculate number of sectors)
mov cx, 0                   ; Initialize CX to 0

; Fill the remaining space with null characters to align to sector size

nulls:
       mov byte [edi], 0     ; Store null character at DI

       inc di                ; Move to the next position
       inc cx                ; Increment CX

       cmp cx, dx            ; Compare CX with DX (number of sectors)
       jl nulls              ; If less, continue filling with nulls

return:
       ret                   ; Return from the subroutine
```

This function handles the storage buffer. It finds the end of the string to copy and copies it to the storage buffer N times or truncates the data when required. Then it calculates the remaining space in the buffer, ensures it's null-terminated, and pads the remaining space with null bytes if needed. It is important to mention that since the reservation of this buffer is located at the very end of the executable it allows us to save any amount of data (up to RAM size) in the buffer whose start is labeled as the storage_buffer.

Compiling and running the code

First of all we have to compile our program. It is done with a bash file with the command:

```
sh ./build_task2
```

```
# build_task2

if [ ! -d backup/ ]; then
mkdir backup/
fi
cp task2.asm backup/

rm -f floppy.img

nasm -f bin -o task2_boot.com task2_boot.asm
truncate -s 1474560 task2_boot.com
mv task2_boot.com floppy.img

nasm -f bin -o task2.com task2.asm
dd if=task2.com of=floppy.img bs=512 seek=1 conv=notrunc
rm -f task2.com

rm -f io_floppy.img
truncate -s 1474560 io_floppy.img
```

The script creates the backup of the program, the bootloader executable it truncates up to 1474560 bytes to create a bootable floppy image with the bootloader a the first sector and then writes the executable code of the program itself starting from the next sector after bootable one. It also creates an empty floppy image for the testing purposes.

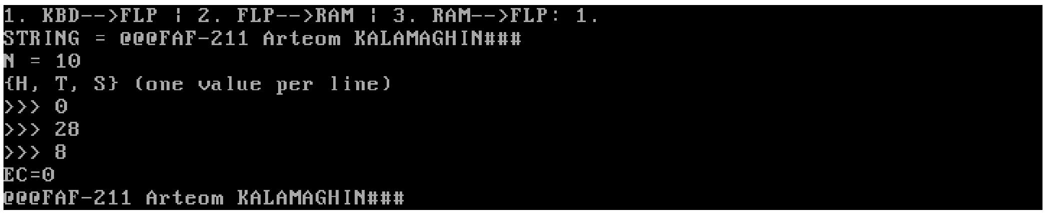
After running the floppy image in the Virtual Box, writing to the floppy, from floppy to RAM and from RAM to floppy, we can run the following command to see what's inside, if operations performed were indeed successful:

```
hexdump -C io_floppy.img
```

```
org 7c00h                    ; Set the origin of the code to memory address 7c00h
                             ; conventional RAM address for booting executable

mov ah, 00                   ; Set up AH register for BIOS Reset Disk System
int 13h                      ; Call BIOS interrupt 13h to reset the disk system

mov ax, 0000h                ; Set AX register to 0000h
mov es, ax                   ; Set ES (Extra Segment) register to the value in AX
mov bx, 7e00h                ; Set BX register to memory address 7e00h

mov ah, 02h                  ; Set up AH register for BIOS Read Sector
mov al, 4                    ; Set AL register to 4 sectors to read
mov ch, 0                    ; Set CH register to 0 (cylinder number)
mov cl, 2                    ; Set CL register to 2 (starting sector number)
mov dh, 0                    ; Set DH register to 0 (head number)
mov dl, 0                    ; Set DL register to 0 (drive number)
int 13h                      ; Call BIOS interrupt 13h to read sectors

jmp 0000h:7e00h              ; Jump to the memory address specified
                             ; (for program execution)

times 510-($-$$) db 0        ; Fill the remaining space up to 510 bytes with zeros
dw 0AA55h                    ; Add the boot signature at the end of
                             ; the boot sector
```

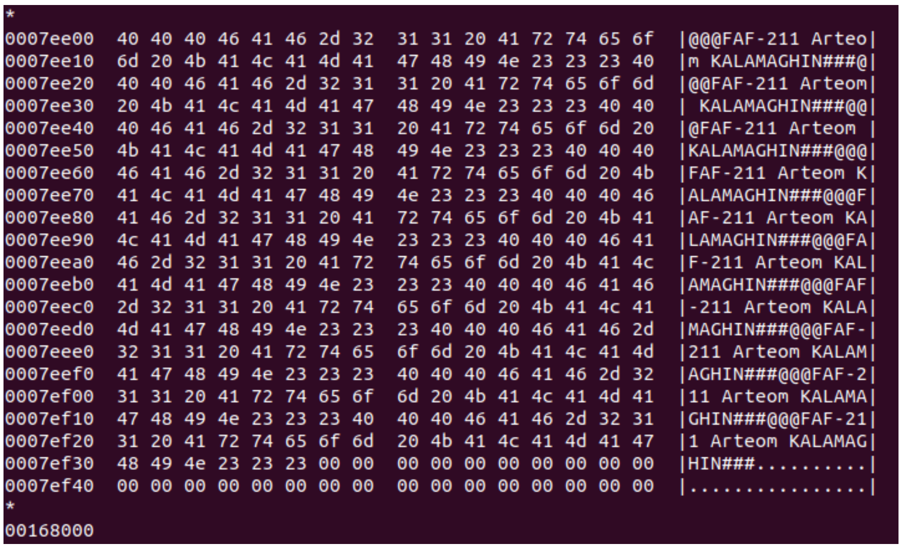
This is a bootloader, which initializes our program by reading the four sectors with the executable code from the disk into memory, and then jumping to the loaded code, assuming the read was successful. The padding ensures that nothing important was overridden, and the boot signature marks it as bootable for the BIOS.

The results:



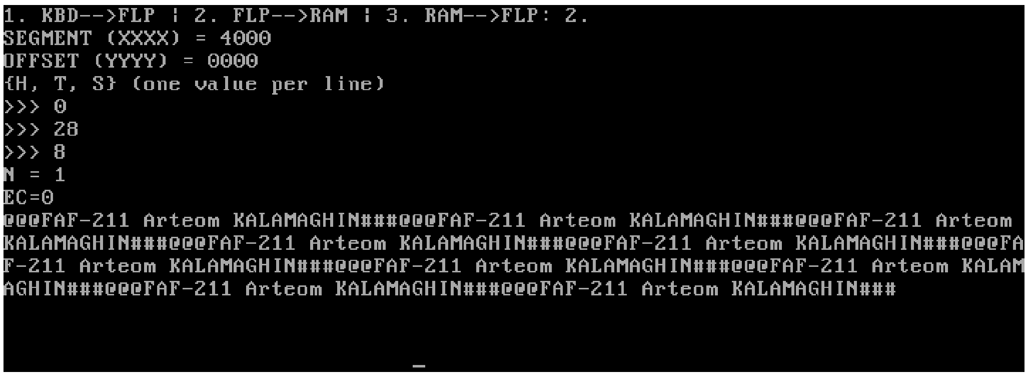
```
1. KBD-->FLP | 2. FLP-->RAM | 3. RAM-->FLP: 1.
STRING = @@@FAF-211 Arteom KALAMAGHIN###
N = 10
{H, T, S} (one value per line)
>>> 0
>>> 28
>>> 8
EC=0
@@@FAF-211 Arteom KALAMAGHIN###
```

Figure 1 – Keyboard to Floppy



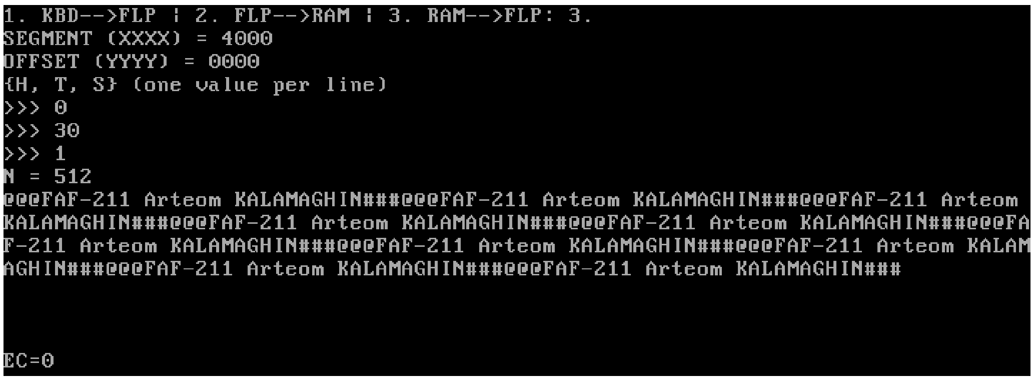
```
*
0007ee00  40 40 40 46 41 46 2d 32  31 31 20 41 72 74 65 6f  |@@@FAF-211 Arteo|
0007ee10  6d 20 4b 41 4c 41 4d 41  47 48 49 4e 23 23 23 40  |m KALAMAGHIN###@|
0007ee20  40 40 46 41 46 2d 32 31  31 20 41 72 74 65 6f 6d  |@@FAF-211 Arteom|
0007ee30  20 4b 41 4c 41 4d 41 47  48 49 4e 23 23 23 40 40  | KALAMAGHIN###@@|
0007ee40  40 46 41 46 2d 32 31 31  20 41 72 74 65 6f 6d 20  |@FAF-211 Arteom |
0007ee50  4b 41 4c 41 4d 41 47 48  49 4e 23 23 23 40 40 40  |KALAMAGHIN###@@@|
0007ee60  46 41 46 2d 32 31 31 20  41 72 74 65 6f 6d 20 4b  |FAF-211 Arteom K|
0007ee70  41 4c 41 4d 41 47 48 49  4e 23 23 23 40 40 40 46  |ALAMAGHIN###@@@F|
0007ee80  41 46 2d 32 31 31 20 41  72 74 65 6f 6d 20 4b 41  |AF-211 Arteom KA|
0007ee90  4c 41 4d 41 47 48 49 4e  23 23 23 40 40 40 46 41  |LAMAGHIN###@@@FA|
0007eea0  46 2d 32 31 31 20 41 72  74 65 6f 6d 20 4b 41 4c  |F-211 Arteom KAL|
0007eeb0  41 4d 41 47 48 49 4e 23  23 23 40 40 40 46 41 46  |AMAGHIN###@@@FAF|
0007eec0  2d 32 31 31 20 41 72 74  65 6f 6d 20 4b 41 4c 41  |-211 Arteom KALA|
0007eed0  4d 41 47 48 49 4e 23 23  23 40 40 40 46 41 46 2d  |MAGHIN###@@@FAF-|
0007eee0  32 31 31 20 41 72 74 65  6f 6d 20 4b 41 4c 41 4d  |211 Arteom KALAM|
0007eef0  41 47 48 49 4e 23 23 23  40 40 40 46 41 46 2d 32  |AGHIN###@@@FAF-2|
0007ef00  31 31 20 41 72 74 65 6f  6d 20 4b 41 4c 41 4d 41  |11 Arteom KALAMA|
0007ef10  47 48 49 4e 23 23 23 40  40 40 46 41 46 2d 32 31  |GHIN###@@@FAF-21|
0007ef20  31 20 41 72 74 65 6f 6d  20 4b 41 4c 41 4d 41 47  |1 Arteom KALAMAG|
0007ef30  48 49 4e 23 23 23 00 00  00 00 00 00 00 00 00 00  |HIN###..........|
0007ef40  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00168000
```

Figure 2 – The result of writing from keyboard to floppy



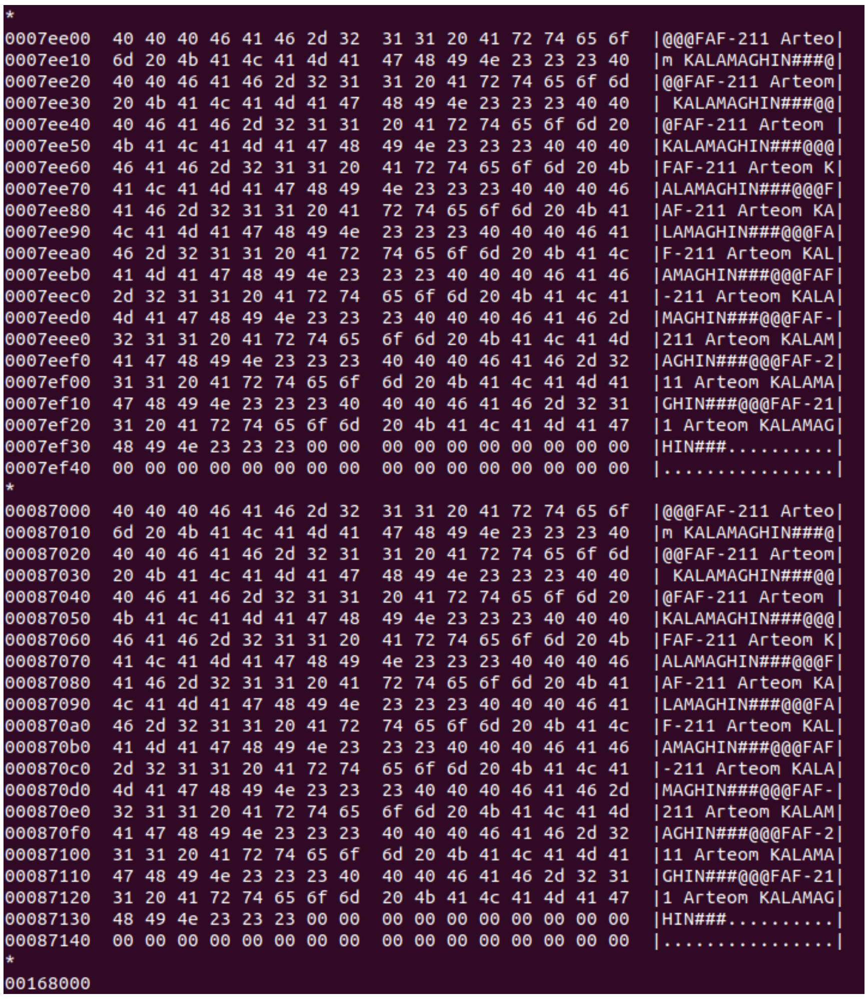
```
1. KBD-->FLP ¦ 2. FLP-->RAM ¦ 3. RAM-->FLP: 2.
SEGMENT (XXXX) = 4000
OFFSET (YYYY) = 0000
{H, T, S} (one value per line)
>>> 0
>>> 28
>>> 8
N = 1
EC=0
@@@FAF-211 Arteom KALAMAGHIN###@@@FAF-211 Arteom KALAMAGHIN###@@@FAF-211 Arteom
KALAMAGHIN###@@@FAF-211 Arteom KALAMAGHIN###@@@FAF-211 Arteom KALAMAGHIN###@@@FA
F-211 Arteom KALAMAGHIN###@@@FAF-211 Arteom KALAMAGHIN###@@@FAF-211 Arteom KALAM
AGHIN###@@@FAF-211 Arteom KALAMAGHIN###@@@FAF-211 Arteom KALAMAGHIN###
```

Figure 3 – Floppy to RAM



```
1. KBD-->FLP ¦ 2. FLP-->RAM ¦ 3. RAM-->FLP: 3.
SEGMENT (XXXX) = 4000
OFFSET (YYYY) = 0000
{H, T, S} (one value per line)
>>> 0
>>> 30
>>> 1
N = 512
@@@FAF-211 Arteom KALAMAGHIN###@@@FAF-211 Arteom KALAMAGHIN###@@@FAF-211 Arteom
KALAMAGHIN###@@@FAF-211 Arteom KALAMAGHIN###@@@FAF-211 Arteom KALAMAGHIN###@@@FA
F-211 Arteom KALAMAGHIN###@@@FAF-211 Arteom KALAMAGHIN###@@@FAF-211 Arteom KALAM
AGHIN###@@@FAF-211 Arteom KALAMAGHIN###@@@FAF-211 Arteom KALAMAGHIN###



EC=0
```

Figure 4 – RAM to Floppy



```
*
0007ee00  40 40 40 46 41 46 2d 32  31 31 20 41 72 74 65 6f  |@@@FAF-211 Arteo|
0007ee10  6d 20 4b 41 4c 41 4d 41  47 48 49 4e 23 23 23 40  |m KALAMAGHIN###@|
0007ee20  40 40 46 41 46 2d 32 31  31 20 41 72 74 65 6f 6d  |@@FAF-211 Arteom|
0007ee30  20 4b 41 4c 41 4d 41 47  48 49 4e 23 23 23 40 40  | KALAMAGHIN###@@|
0007ee40  40 46 41 46 2d 32 31 31  20 41 72 74 65 6f 6d 20  |@FAF-211 Arteom |
0007ee50  4b 41 4c 41 4d 41 47 48  49 4e 23 23 23 40 40 40  |KALAMAGHIN###@@@|
0007ee60  46 41 46 2d 32 31 31 20  41 72 74 65 6f 6d 20 4b  |FAF-211 Arteom K|
0007ee70  41 4c 41 4d 41 47 48 49  4e 23 23 23 40 40 40 46  |ALAMAGHIN###@@@F|
0007ee80  41 46 2d 32 31 31 20 41  72 74 65 6f 6d 20 4b 41  |AF-211 Arteom KA|
0007ee90  4c 41 4d 41 47 48 49 4e  23 23 23 40 40 40 46 41  |LAMAGHIN###@@@FA|
0007eea0  46 2d 32 31 31 20 41 72  74 65 6f 6d 20 4b 41 4c  |F-211 Arteom KAL|
0007eeb0  41 4d 41 47 48 49 4e 23  23 23 40 40 40 46 41 46  |AMAGHIN###@@@FAF|
0007eec0  2d 32 31 31 20 41 72 74  65 6f 6d 20 4b 41 4c 41  |-211 Arteom KALA|
0007eed0  4d 41 47 48 49 4e 23 23  23 40 40 40 46 41 46 2d  |MAGHIN###@@@FAF-|
0007eee0  32 31 31 20 41 72 74 65  6f 6d 20 4b 41 4c 41 4d  |211 Arteom KALAM|
0007eef0  41 47 48 49 4e 23 23 23  40 40 40 46 41 46 2d 32  |AGHIN###@@@FAF-2|
0007ef00  31 31 20 41 72 74 65 6f  6d 20 4b 41 4c 41 4d 41  |11 Arteom KALAMA|
0007ef10  47 48 49 4e 23 23 23 40  40 40 46 41 46 2d 32 31  |GHIN###@@@FAF-21|
0007ef20  31 20 41 72 74 65 6f 6d  20 4b 41 4c 41 4d 41 47  |1 Arteom KALAMAG|
0007ef30  48 49 4e 23 23 23 00 00  00 00 00 00 00 00 00 00  |HIN###..........|
0007ef40  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00087000  40 40 40 46 41 46 2d 32  31 31 20 41 72 74 65 6f  |@@@FAF-211 Arteo|
00087010  6d 20 4b 41 4c 41 4d 41  47 48 49 4e 23 23 23 40  |m KALAMAGHIN###@|
00087020  40 40 46 41 46 2d 32 31  31 20 41 72 74 65 6f 6d  |@@FAF-211 Arteom|
00087030  20 4b 41 4c 41 4d 41 47  48 49 4e 23 23 23 40 40  | KALAMAGHIN###@@|
00087040  40 46 41 46 2d 32 31 31  20 41 72 74 65 6f 6d 20  |@FAF-211 Arteom |
00087050  4b 41 4c 41 4d 41 47 48  49 4e 23 23 23 40 40 40  |KALAMAGHIN###@@@|
00087060  46 41 46 2d 32 31 31 20  41 72 74 65 6f 6d 20 4b  |FAF-211 Arteom K|
00087070  41 4c 41 4d 41 47 48 49  4e 23 23 23 40 40 40 46  |ALAMAGHIN###@@@F|
00087080  41 46 2d 32 31 31 20 41  72 74 65 6f 6d 20 4b 41  |AF-211 Arteom KA|
00087090  4c 41 4d 41 47 48 49 4e  23 23 23 40 40 40 46 41  |LAMAGHIN###@@@FA|
000870a0  46 2d 32 31 31 20 41 72  74 65 6f 6d 20 4b 41 4c  |F-211 Arteom KAL|
000870b0  41 4d 41 47 48 49 4e 23  23 23 40 40 40 46 41 46  |AMAGHIN###@@@FAF|
000870c0  2d 32 31 31 20 41 72 74  65 6f 6d 20 4b 41 4c 41  |-211 Arteom KALA|
000870d0  4d 41 47 48 49 4e 23 23  23 40 40 40 46 41 46 2d  |MAGHIN###@@@FAF-|
000870e0  32 31 31 20 41 72 74 65  6f 6d 20 4b 41 4c 41 4d  |211 Arteom KALAM|
000870f0  41 47 48 49 4e 23 23 23  40 40 40 46 41 46 2d 32  |AGHIN###@@@FAF-2|
00087100  31 31 20 41 72 74 65 6f  6d 20 4b 41 4c 41 4d 41  |11 Arteom KALAMA|
00087110  47 48 49 4e 23 23 23 40  40 40 46 41 46 2d 32 31  |GHIN###@@@FAF-21|
00087120  31 20 41 72 74 65 6f 6d  20 4b 41 4c 41 4d 41 47  |1 Arteom KALAMAG|
00087130  48 49 4e 23 23 23 00 00  00 00 00 00 00 00 00 00  |HIN###..........|
00087140  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00168000
```

Figure 5 – The result of writing from RAM to floppy

Conclusion:

In this laboratory work we had the possibility to work with the floppy disks and RAM on the computer and understand their working process. The project required different functions to handle different inputs, proper data handling and data transmission. Also the laboratory work involves the data conversion, from ASCII to integer and to hex, reading HTS address, reading RAM address, writing to the buffer, clearing the buffer and error detection and displaying error code. The laboratory work also gave the possibility to understand the working principle of a bootloader. For the program that we have it was necessary to make a bootloader which will boot the program properly by loading all the executable code that exceeds 512 bytes loaded automatically by default.