Ministry of Education and Investigation Republic of Moldova

Technical University of Moldova

Faculty of Computers, Informatics and Microelectronics

REPORT

Laboratory work nr.3

on the course “Operating Systems”

Executed by:

st. gr. FAF-212 Vladimir Luchianov

Anatolie Telug

Alexei Ciumac

Verified by:

prof. univ. Rostislav Calin

Chişinău - 2023

Topic: Floppy disk input/output operations

Tasks:

1. **Description of Data Format in Student Blocks:**
    - In the first and last sector of each student's block (on a floppy disk), the following textual information should be written in the specified format (without quotes): "@@@FAF-21* Firstname LASTNAME###".
    - This text string must be duplicated 10 times without additional delimiter characters.
    - Examples are provided for illustration.
2. **Assembly Program Functions:**
    - **(KEYBOARD ==> FLOPPY):**
        - Read a string from the keyboard with a maximum length of 256 characters (backspace correction should work).
        - Write this string to the floppy disk "N" times, starting at the address {Head, Track, Sector}, where "N" can take values in the range (1-30000).
        - After detecting the ENTER key, if the length of the string is greater than 0, display an empty line and then the recently entered string.
        - Variables "N," "Head," "Track," and "Sector" must be visibly read from the keyboard.
        - After completing the write operation to the floppy disk, display the error code.
    - **(FLOPPY ==> RAM):**
        - Read "N" sectors from the floppy disk starting at the address {Head, Track, Sector}.
        - Transfer this data to RAM starting at the address {XXXX:YYYY}.

        - After completing the read operation from the floppy disk, display the error code.
        - After the error code, display the entire volume of data at the address {XXXX:YYYY} that was read from the floppy disk.
        - If the displayed data volume is larger than a video page, implement paging by pressing the "SPACE" key.
        - Variables "N," "Head," "Track," "Sector," and the address {XXXX:YYYY} must be read from the keyboard.
    - **(RAM ==> FLOPPY):**
        - Write to the floppy disk starting at the address {Head, Track, Sector} a volume of "Q" bytes from RAM starting at the address {XXXX:YYYY}.
        - Display the data block of "Q" bytes on the screen.
        - After completing the write operation to the floppy disk, display the error code.
3. **Post-Function Execution:**
    - After executing any of the above functions, the program should be ready for the execution of the next function (any of the three functions described above).

Implementation and results:

1.

```
org 7c00h   ; Origin point in memory where the bootloader typically starts executing



; Disk read operation using BIOS interrupt 13h to load data from the disk to memory

mov ah, 02h     ; Function 02h of interrupt 13h (Read Sectors Into Memory)
```

```
mov al, 6       ; Number of sectors to read

mov cx, 2       ; Cylinder number

mov dh, 0       ; Head number

mov dl, 0        ; Drive number (in this case, the boot
drive)

mov bx, 0       ; Segment address to which the data will
be loaded

mov es, bx      ; Set ES (Extra Segment) register to the
segment address

mov bx, 7e00h  ; Offset where the data will be loaded in
the segment (destination address)

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


call write_Name_Group_1 ; 1st student

call write_Name_Group_2 ; 2nd student

call write_Name_Group_3 ; 3nd student



go:

    call clean_screen           ; Clear the screen


    mov word [Address], Options ; Set the memory address
for the Options text

    call write_chr                ; Display the Options
text on the screen
```

```
select_option:
    call clean_line             ; Clear the input line
    mov byte [N_of_Chars], 1    ; Set the maximum number
of characters for input to 1
    call int_input              ; Accept user input for
selection

    cmp byte [Error_Val], 0     ; Check for input error
(if any)
    jne go                      ; If an error is detected,
jump back to display the Options text again

    cmp word [Digit_Buffer], 1  ; Check the user input
to determine the operation
    je Keyboard_to_Floppy       ; If the user selected
'1', jump to the Keyboard_to_Floppy section
    cmp word [Digit_Buffer], 2  ; Check for '2'
    je Floppy_To_RAM            ; Jump to the Floppy_To_RAM
section if the user selected '2'
    cmp word [Digit_Buffer], 3  ; Check for '3'
    je RAM_To_Floppy            ; Jump to the RAM_To_Floppy
section if the user selected '3'

    jmp select_option           ; If none of the valid
options were selected, repeat the selection process

Keyboard_to_Floppy:
```

```
    call newline                ; Move to a new line for
better UI separation

    mov word [Address], Text_Prompt  ; Set the address
for the Text_Prompt message

    call write_chr              ; Display the prompt for
text input


    mov si, Text_Buffer         ; Set the source index
to Text_Buffer for string input

    call str_input              ; Get a string input from
the user and store it in Text_Buffer


    call newline                ; Move to a new line

    mov word [Address], N_Prompt  ; Set the address for
the N_Prompt message

    call write_chr              ; Display the prompt for
the number of characters to save


K_t_F_loop:

    call clean_line             ; Clear the input line

    mov byte [N_of_Chars], 5    ; Limit input to 5
characters for the number of bytes


    call int_input              ; Accept user input for
the number of bytes to save


    cmp word [Digit_Buffer], 1  ; Compare user input with
1
```

```
    jl K_t_F_loop                ; If input is less than
1, loop back to re-enter


    cmp word [Digit_Buffer], 30000  ; Compare user input
with 30000

    jg K_t_F_loop                ; If input is greater
than 30000, loop back to re-enter


    cmp byte [Error_Val], 2      ; Check for an input
error (Error_Val = 2 for invalid input)

    je K_t_F_loop                ; If error detected, loop
back to re-enter


    cmp byte [Error_Val], 1      ; Check for an input
error (Error_Val = 1 for escape)

    je go                        ; If escape detected,
jump to 'go' label (exit point)


    mov ax, word [Digit_Buffer] ; Move the user-input
bytes count to 'N' variable

    mov word [N], ax             ; Store the count in 'N'
variable for further operations


    call newline                 ; Move to a new line

    call prompt_inputs           ; Display prompt messages
for head, track, sector, and ES:BP input


    call save_in_floppy          ; Save the data in the
floppy
```

```
    push ax                         ; Preserve AX register
value

    call newline                    ; Move to a new line

    pop ax                          ; Restore AX register
value

    cmp ah, 0                       ; Check if the operation
was successful (AH = 0 for success)

    je K_t_F_success                ; Jump to success message
if AH = 0


K_t_F_fail:

    mov word [Address], Keyboard_to_Floppy_fail   ; Set
address for failure message

    push ax                         ; Preserve AX register
value

    call write_chr                  ; Display the failure
message

    pop ax                          ; Restore AX register
value

    mov bl, ah                      ; Move error code to BL
register for display

    call print_to_hex_byte          ; Print error code in
hex format

    jmp K_t_F_done                  ; Jump to the end of this
section


K_t_F_success:

    mov word [Address], Keyboard_to_Floppy_success   ;
Set address for success message
```

```
    call write_chr                ; Display the success
message


K_t_F_done:

    mov ah, 00h                   ; Reset AH register to
prepare for interrupt

    int 16h                       ; Wait for a key press
before continuing

    jmp go                        ; Jump back to the main
loop ('go') for further operations


Floppy_To_RAM:

    call newline                  ; Move to a new line for
better UI separation

    mov word [Address], N_Prompt  ; Set the address for
the N_Prompt message

    call write_chr                ; Display the prompt for
the number of characters to read from floppy


F_t_RAM_loop:

    call clean_line               ; Clear the input line

    mov byte [N_of_Chars], 5      ; Limit input to 5
characters for the number of bytes to read


    call int_input                ; Accept user input for
the number of bytes to read

    mov al, [Digit_Buffer]        ; Move the input to AL
register
```

```
    mov [N], al                    ; Store the number of
bytes to read in variable 'N'


    cmp word [Digit_Buffer], 1  ; Compare user input with
1

    jl F_t_RAM_loop             ; If input is less than
1, loop back to re-enter


    cmp word [Digit_Buffer], 30000  ; Compare user input
with 30000

    jg F_t_RAM_loop             ; If input is greater
than 30000, loop back to re-enter


    cmp byte [Error_Val], 2      ; Check for an input
error (Error_Val = 2 for invalid input)

    je F_t_RAM_loop             ; If error detected, loop
back to re-enter


    cmp byte [Error_Val], 1      ; Check for an input
error (Error_Val = 1 for escape)

    je go                        ; If escape detected,
jump to 'go' label (exit point)


    call newline                ; Move to a new line

    call prompt_inputs          ; Display prompt messages
for head, track, sector, and ES:BP input


    call newline                ; Move to a new line
```

```
    call address_input          ; Accept user input for
ES:BP address, checking for errors


    cmp byte [Error_Val], 1     ; Check for an address
input error (Error_Val = 1 for escape)

    je go                       ; If escape detected,
jump to 'go' label (exit point)


    mov ah, 02h                 ; Prepare AH register
for BIOS function 02h (read sectors)

    mov al, [N]                 ; Number of sectors to
read from floppy

    mov ch, [Track]             ; Cylinder (track) number

    mov cl, [Sector]            ; Sector number

    mov dl, 0                   ; Floppy drive number

    mov dh, [Head]              ; Head number

    mov bx, [ES_Hex]            ; Segment of buffer
address

    mov es, bx                  ; Set ES to segment of
buffer address

    mov bx, [BP_Hex]            ; Offset of buffer address

    int 13h                     ; BIOS interrupt for
reading sectors from floppy


    push ax                     ; Preserve AX register
valueə

    call newline                ; Move to a new line
```

```
    pop ax                          ; Restore AX register
value


    cmp ah, 0                       ; Check if the operation
was successful (AH = 0 for success)

    jnz F_t_RAM_fail                ; If not successful, jump
to failure label


    mov word [Address], Floppy_To_RAM_success   ; Set
address for success message

    call write_chr                  ; Display the success
message


    mov bx, [ES_Hex]                ; Move segment of buffer
address to BX register

    mov es, bx                      ; Set ES to the segment
of the buffer address

    mov bp, [BP_Hex]                ; Move offset of buffer
address to BP register

    call write_RAM                  ; Display the content
stored in the buffer


    jmp go                          ; Jump back to the main
loop ('go') for further operations


F_t_RAM_fail:

    mov word [Address], Floppy_To_RAM_fail  ; Set address
for failure message
```

```
    push ax                         ; Preserve AX register
value

    call write_chr                  ; Display the failure
message

    pop ax                          ; Restore AX register
value

    mov bl, ah                     ; Move error code to BL
register for display

    call print_to_hex_byte          ; Print error code in
hex format

    mov ah, 00h                     ; Reset AH register to
prepare for interrupt

    int 16h                         ; Wait for a key press
before continuing

    jmp go                         ; Jump back to the main
loop ('go') for further operations


RAM_To_Floppy:

    call newline                   ; Move to a new line for
UI separation

    call address_input              ; Accept user input for
ES:BP address

    cmp byte [Error_Val], 1         ; Check for an address
input error (Error_Val = 1 for escape)

    je go                           ; If escape detected,
jump to 'go' label (exit point)


    call newline                   ; Move to a new line
```

```
    mov word [Address], Q_Prompt  ; Set address for the
prompt asking for the quantity of bytes to write

    call write_chr                ; Display the prompt
message

    call newline                  ; Move to a new line


byte_write:

    call clean_line               ; Clear the input line

    mov byte [N_of_Chars], 5      ; Set the maximum number
of characters for input to 5


    call int_input                ; Accept user input for
the quantity of bytes to write

    cmp word [Digit_Buffer], 1    ; Compare the input with
1

    jl byte_write                 ; If less than 1, loop
back to re-enter


    cmp byte [Error_Val], 2       ; Check for an input error
(Error_Val = 2 for invalid input)

    je byte_write                 ; If error detected, loop
back to re-enter


    cmp byte [Error_Val], 1       ; Check for an input error
(Error_Val = 1 for escape)

    je go                         ; If escape detected, jump
to 'go' label (exit point)
```

```
    mov ax, word [Digit_Buffer]  ; Move the input to AX register

    mov word [Q], ax       ; Store the quantity of bytes to write in variable 'Q'


    call newline            ; Move to a new line

    call prompt_inputs        ; Display prompt messages for head, track, sector, and ES:BP input


    call from_RAM           ; Read from RAM and write to floppy

    push ax                 ; Preserve AX register value

    call newline            ; Move to a new line

    pop ax                  ; Restore AX register value

    cmp ah, 0               ; Check if the operation was successful (AH = 0 for success)

    je RAM_t_F_success        ; If successful, jump to success label


    mov word [Address], RAM_To_Floppy_fail  ; Set address for failure message

    push ax                 ; Preserve AX register value

    call write_chr          ; Display the failure message

    pop ax                  ; Restore AX register value

    mov bl, ah                ; Move error code to BL register for display

    call print_to_hex_byte   ; Print error code in hex format
```

```
    jmp K_t_F_done          ; Jump to exit point


RAM_t_F_success:

    mov word [Address], RAM_To_Floppy_success   ; Set address for success message

    call write_chr  ; Display the success message

    mov ah, 00h     ; Reset AH register to prepare for interrupt

    int 16h             ; Wait for a key press before continuing

    jmp go          ; Jump back to the main loop ('go') for further operations


times 510 - ($ - $$) db 0    ; Fill the remaining space in the boot sector with zeros

dw 0aa55h                    ; Boot sector signature


save_in_floppy:

    mov si, Text_Buffer      ; Move Text_Buffer address to SI (source index)

    mov di, Memory_Buffer    ; Move Memory_Buffer address to DI (destination index)

    xor ax, ax               ; Clear AX register


full_floppy_buffer:

    cmp ax, 512              ; Compare AX with 512
```

```
    je buffer_to_floppy      ; Jump to buffer_to_floppy if AX equals 512

    cmp word [N], 0          ; Compare the value in variable N with 0

    je buffer_to_floppy      ; Jump to buffer_to_floppy if N equals 0


    mov bl, byte [si]        ; Move the byte at SI into BL

    mov byte [di], bl        ; Move the byte in BL to the address in DI

    inc ax                   ; Increment AX (counter for buffer size)

    inc si                   ; Increment SI (source pointer)

    inc di                   ; Increment DI (destination pointer)


    cmp byte [si], 0         ; Check if the byte at SI is 0 (end of string)

    jne full_floppy_buffer   ; If not zero, continue to full_floppy_buffer

    mov si, Text_Buffer      ; Reset SI to Text_Buffer

    dec word [N]             ; Decrement the value in variable N


    jmp full_floppy_buffer   ; Jump back to full_floppy_buffer
```

```
buffer_to_floppy:
    mov ch, [Track]           ; Move Track to CH
    mov cl, [Sector]          ; Move Sector to CL
    mov dh, [Head]            ; Move Head to DH
    xor dl, dl                ; Clear DL (drive number)
    xor ax, ax                ; Clear AX
    mov es, ax                 ; Set ES to 0 (segment of
Memory_Buffer)
    mov bx, Memory_Buffer    ; Move Memory_Buffer address
to BX
    mov ax, 0301h               ; Set AH to 03 (write
sectors) and AL to 01 (sector count)
    int 13h                    ; Call BIOS interrupt for
disk I/O


    cmp ah, 0                 ; Check if AH (error code)
is zero
    jne save_in_floppy_done   ; If not zero, jump to
save_in_floppy_done


    mov di, Memory_Buffer    ; Move Memory_Buffer address
to DI (resetting DI)


memory_buffer_clean:
    cmp byte [di], 0          ; Compare the byte at DI
with 0
    je next_floppy_memory_part  ; If it's zero, jump to
next_floppy_memory_part
```

```
    mov byte [di], 0          ; Set the byte at DI to 0
    inc di                    ; Increment DI
    cmp di, Memory_Buffer + 512   ; Compare DI with Memory_Buffer + 512
    je next_floppy_memory_part   ; If DI reached the end, jump to next_floppy_memory_part
    jmp memory_buffer_clean   ; Continue cleaning the memory buffer

next_floppy_memory_part:
    cmp word [N], 0             ; Compare the value in variable N with 0
    je save_in_floppy_done    ; If N is zero, jump to save_in_floppy_done

    inc byte [Sector]         ; Increment Sector
    cmp byte [Sector], 19     ; Compare Sector with 19
    jl save_loop             ; If less, jump to save_loop

    mov byte [Sector], 1      ; Reset Sector to 1
    inc byte [Head]           ; Increment Head

    cmp byte [Head], 2        ; Compare Head with 2
    jl save_loop             ; If less, jump to save_loop

    mov byte [Head], 0        ; Reset Head to 0
```

```
    inc byte [Track]          ; Increment Track


    cmp byte [Track], 80      ; Compare Track with 80

    je  save_in_floppy_done       ;  If  equal,  jump  to
save_in_floppy_done


save_loop:

    mov di, Memory_Buffer    ; Move Memory_Buffer address
to DI

    xor ax, ax                ; Clear AX

    cmp byte [si], 0            ; Compare the byte at SI
with 0

    jne full_floppy_buffer     ;  If  not  zero,  jump  to
full_floppy_buffer

    mov si, Text_Buffer       ; Reset SI to Text_Buffer

    jmp   full_floppy_buffer          ;   Jump   back   to
full_floppy_buffer


save_in_floppy_done:

    ret                      ; Return from the subroutine


from_RAM:

    xor dx, dx              ; Clear DX

    mov ax, [Q]             ; Move Q value to AX

    mov cx, 512             ; Set CX to 512

    div cx                  ; AX = AX/CX, DX = Remainder
```

```
    cmp dx, 0               ; Compare remainder DX with 0
    jne RAM_t_F_stop        ; Jump to RAM_t_F_stop if not
equal to 0
    dec ax                  ; Decrement AX

RAM_t_F_stop:
    inc ax                  ; Increment AX
    mov ch, [Track]         ; Move Track to CH
    mov cl, [Sector]        ; Move Sector to CL
    mov dh, [Head]          ; Move Head to DH
    xor dl, dl              ; Clear DL (drive number)
    mov es, [ES_Hex]        ; Move ES_Hex to ES
    mov bx, [BP_Hex]        ; Move BP_Hex to BX
    mov ah, 03h             ; Set AH to 03 (read sectors)
    int 13h                 ; BIOS disk I/O interrupt

    cmp ah, 0               ; Compare AH (error code) with
0
    jne RAM_t_F_done        ; Jump to RAM_t_F_done if AH
is not zero

RAM_t_F_done:
    ret                     ; Return from the subroutine
```

```
write_chr:
    call get_cursor_pos   ; Get cursor position
    mov si, [Address]     ; Move Address to SI

strlen:
    cmp byte [si], 0      ; Compare byte at SI with 0
    je print_word          ; Jump to print_word if it's
zero
    inc si                ; Increment SI
    jmp strlen            ; Jump back to strlen

print_word:
    sub si, [Address]     ; Calculate string length
    mov bx, 0007h         ; Text attribute
    mov cx, si            ; Move string length to CX
    mov ax, 0             ; Clear AX
    mov es, ax            ; Set ES to 0
    mov bp, [Address]     ; Move Address to BP
    mov ax, 1301h          ; Function to write string at
cursor position
    int 10h               ; Video BIOS interrupt

    ret                   ; Return from the subroutine

write_RAM:
```

```
    mov di, 0              ; Clear DI (destination index)

write_RAM_sectors:
    xor ax, ax             ; Clear AX
    mov al, byte [N]       ; Move N value to AL
    cmp di, ax             ; Compare DI with AL
    jae write_RAM_done      ; Jump if DI is greater or
equal to AL

    ; Display input prompts and values on screen
    call clean_screen
    mov word [Address], Head_Input
    call write_chr
    xor ax, ax
    mov al, [Head]
    push ax
    call print_number
    mov word [Address], Track_Input
    call write_chr
    xor ax, ax
    mov al, [Track]
    push ax
    call print_number
    mov word [Address], Sector_Input
```

```
call write_chr
xor ax, ax
mov al, [Track]
push ax
call print_number
mov word [Address], ESBP_Input
call write_chr
push word [ES_Hex]
call print_to_hex_word
mov ah, 0x0e
mov al, ':'
int 0x10
push word [BP_Hex]
call print_to_hex_word
call newline

; Write content to the RAM sectors
mov word [Address], Space_Key
call write_chr
mov ax, 1301h
mov bx, [ES_Hex]
mov es, bx
mov bx, 0007h
mov cx, 512
```

```
    mov dx, 0200h
    mov bp, [BP_Hex]
    int 0x10
    inc di                    ; Increment DI
    inc byte [Sector]         ; Increment Sector
    add word [BP_Hex], 512    ; Add 512 to BP_Hex

write_RAM_press:
    mov ah, 00h
    int 16h
    cmp ah, 01h               ; Check for keyboard input
    je write_RAM_done         ; Jump to write_RAM_done if
'Esc' is pressed
    cmp al, 20h               ; Check if 'Space' is pressed
    jne write_RAM_press       ; If not 'Space', continue
to write_RAM_press
    jmp   write_RAM_sectors          ;   Jump   back   to
write_RAM_sectors

write_RAM_done:
    ret                       ; Return from the subroutine

str_input:
    mov ah, 0                 ; Set AH to 0 (interrupt code
for keyboard input)
```

```
    int 16h                     ; BIOS interrupt: Wait for
key press

    cmp ah, 0eh                 ; Check for backspace key
    je str_backspace            ; Jump to str_backspace if
backspace is pressed

    cmp ah, 1ch                 ; Check for enter key
    je str_enter                ; Jump to str_enter if enter
is pressed

    cmp al, 20h                 ; Check for printable ASCII
characters (above space)
    jl str_input                ; Jump back if below the
printable ASCII range
    cmp al, 7fh                 ; Check for printable ASCII
characters (above DEL)
    je str_input                ; Jump back if above the
printable ASCII range

    cmp si, Text_Buffer + 256   ; Check if the buffer is
full
    je str_input                ; Jump back if the buffer is
full
    mov [si], al                ; Store the entered character
in the buffer
    inc si                      ; Increment buffer pointer
```

```
    mov ah, 0eh                 ; BIOS interrupt: Teletype
output function
    int 10h                     ; Display the character

    jmp str_input               ; Repeat the input process

str_backspace:
    cmp si, Text_Buffer         ; Check if the buffer is
empty
    je str_input                ; Jump back if the buffer is
empty
    dec si                      ; Decrement buffer pointer
    mov byte [si], 0            ; Clear the last entered
character

    call get_cursor_pos         ; Get the cursor position

    cmp dl, 0                   ; Check if at the beginning
of a line
    jz prev_line                ; Jump to prev_line if at
the start of the line
    jmp write_space             ; Jump to write_space

prev_line:
    mov dl, 79                  ; Move to the previous column
    dec dh                      ; Move to the previous row
```

```
write_space:
    mov ah, 02h             ; BIOS interrupt: Set cursor
position
    dec dl                  ; Move cursor one space back
    int 10h                 ; Update cursor position

    mov ah, 0ah               ; BIOS interrupt: Write
character and attribute
    mov al, 20h             ; Write a space
    mov cx, 1               ; Write one character
    int 10h                 ; Write the space

    jmp str_input           ; Repeat the input process

str_enter:
    ret                     ; Return from the subroutine

int_input:
    mov byte [Error_Val], 0    ; Clear error value
    mov word [Digit_Buffer], 0 ; Clear the buffer for
digits
    mov byte [Digits], 0         ; Clear the count of
digits entered
    xor cx, cx                ; Clear CX register
```

```
int_press:
    xor ah, ah                  ; Clear AH register
    int 16h                     ; BIOS interrupt: Wait
for key press

    cmp ah, 01h                 ; Check for ESC key
    je int_escape               ; Jump to int_escape if
ESC is pressed

    cmp ah, 0eh                 ; Check for backspace key
    je int_backspace            ; Jump to int_backspace
if backspace is pressed

    cmp ah, 1ch                 ; Check for enter key
    je int_enter                ; Jump to int_enter if
enter is pressed

    mov cl, byte [Digits]       ; Load the count of entered
digits
    cmp cl, byte [N_of_Chars]   ; Compare with the allowed
number of digits
    je int_press                ; Jump back if the limit
is reached

    cmp al, 30h                 ; Check if entered
character is less than '0'
    jl int_press                ; Jump back if less than
'0'
```

```
    cmp al, 39h                     ; Check if entered
character is greater than '9'
    jg int_press                  ; Jump back if greater
than '9'


    mov ah, 0eh                ; BIOS interrupt: Teletype
output function
    mov bl, 0                       ; Display the entered
digit
    int 10h


    sub al, 30h                     ; Convert ASCII to
numerical value
    mov cl, al                    ; Store the numerical
value


    mov ax, word [Digit_Buffer] ; Load the current buffer
content
    mov dx, 10                      ; Set up divisor for
decimal placement
    mul dx                      ; Multiply buffer content
by 10
    cmp dx, 0                   ; Check if multiplication
resulted in overflow
    jg int_error                ; Jump to error handling
if overflow occurred


    add ax, cx                      ; Add the new digit to
the buffer
```

```
    mov word [Digit_Buffer], ax ; Store the updated buffer content

    inc byte [Digits]           ; Increment the count of entered digits

    jmp int_press              ; Repeat the input process


int_backspace:

    cmp byte [Digits], 0         ; Check if there are digits to erase

    je int_press                ; Jump back if no digits to erase

    dec byte [Digits]           ; Decrement the count of entered digits


    mov ax, word [Digit_Buffer] ; Load the current buffer content

    mov cx, 0ah                  ; Set up divisor for decimal placement

    mov dx, 0                   ; Clear DX for division

    div cx                       ; Divide buffer content by 10

    mov word [Digit_Buffer], ax ; Store the updated buffer content


    call get_cursor_pos           ; Get current cursor position


    mov ah, 02h                  ; BIOS interrupt: Set cursor position
```

```
    dec dl                          ; Move cursor one space
back
    int 10h                         ; Update cursor position

    mov ah, 0ah                     ; BIOS interrupt: Write
character and attribute
    mov al, 20h                     ; Write a space to erase
the character
    mov cx, 1                       ; Write one character
    int 10h                         ; Write the space
    jmp int_press                   ; Repeat the input process

int_enter:
    cmp byte [Digits], 0            ; Check if no digits were
entered
    jg int_press_done               ; Jump to done if digits
were entered
    jmp int_input                   ; If no digits entered,
restart the input process

int_error:
    inc byte [Error_Val]            ; Set an error flag for
overflow

int_escape:
    inc byte [Error_Val]            ; Set an error flag for
ESC key pressed
```

```
int_press_done:
    ret                                 ; Return from the
subroutine


newline:
    call get_cursor_pos             ; Get current cursor
position


    cmp dh, 24                      ; Compare current row
with the maximum row number
    jl newline_scroll_skip      ; Jump if the row is less
than the maximum allowed


    mov ax, 0601h                   ; Scroll screen up by
one line
    mov cx, 0                     ; Top left corner (row
0, column 0)
    mov dx, 184fh                   ; Bottom right corner
(row 24, column 79)
    int 10h                           ; BIOS interrupt to
scroll the screen
    mov dh, 17h                  ; Set cursor at the last
visible row


newline_scroll_skip:
    mov ah, 02h                     ; BIOS interrupt: Set
cursor position
```

```
    inc dh                      ; Move to the next line
    mov dl, 0                   ; Set column to 0
    int 10h                     ; Update cursor position
    ret                             ; Return from the
subroutine

clean_line:
    call get_cursor_pos          ; Get current cursor
position

    mov ax, 0600h               ; BIOS interrupt: Scroll
window up
    mov bh, 07h                 ; Text attribute (white
on black)
    mov ch, dh                  ; Current row
    xor cl, cl                  ; Column 0
    mov dl, 4fh                 ; End column (79)
    int 10h                     ; Scroll the current line
up

    mov ah, 02h                   ; BIOS interrupt: Set
cursor position
    mov bh, 0                   ; Page number
    mov dl, 0                   ; Column 0
    int 10h                     ; Update cursor position
    ret                     ; Return from the subroutine
```

```
get_cursor_pos:
    mov ah, 03h                     ; BIOS interrupt: Get
cursor position
    mov bh, 0                       ; Page number (0 for text
mode)
    int 10h                        ; Retrieve cursor position
    ret                           ; Return from the subroutine


address_input:
    mov ax, 1300h                        ; BIOS interrupt:
Display Address Prompt
    mov bl, 07h                           ; Text attribute
(white on black)
    mov cx, Address_Prompt_length     ; Length of the
prompt
    mov bp, Address_Prompt             ; Pointer to the
prompt message
    int 10h                           ; Display the prompt
    call newline                        ; Move to the next
line


    mov ax, 1300h                        ; BIOS interrupt:
Display Address Input
    mov bl, 07h                           ; Text attribute
(white on black)
    mov cx, Address_Input_length    ; Length of the input
```

```
    mov bp, Address_Input              ; Pointer to the
input area
    int 10h                           ; Display the input
area

    mov di, ES_Buffer                  ; Set destination
index to ES_Buffer (destination for user input)

address_press:                         ; Label for input
handling
    mov ah, 00h                        ; BIOS interrupt:
Wait for keypress
    int 0x16                          ; Wait for keyboard
input

    cmp ah, 0eh                       ; Check for backspace
    je address_backspace               ; Handle backspace

    cmp ah, 1ch                       ; Check for Enter key
    je address_enter                   ; Handle Enter

    cmp ah, 01h                        ; Check for Esc key
    je address_escape                  ; Handle Esc

    cmp al, 20h                       ; Check for printable
characters
```

```
    jae address_default                ; Jump to default
input handling

    jmp address_press                  ; Jump back for non-
printable characters

address_backspace:                     ; Label for handling
backspace
    cmp di, ES_Buffer                  ; Check if at the
start of input buffer
    je address_press                   ; Jump back if at the
start
    mov ah, 03h                        ; BIOS interrupt:
Move cursor left
    int 10h                            ; Move cursor left
    dec dl                             ; Decrement column
position
    cmp di, BP_Buffer                  ; Check if at BP_Buffer
    jne backspace_hop                  ; Jump to backspace_hop
if not
    dec dl                             ; Decrement column
position again

backspace_hop:                         ; Label to handle
backspace column position
    mov ah, 02h                        ; BIOS interrupt:
Move cursor left
    int 10h                            ; Move cursor left
```

```
    mov ah, 0ah                         ; BIOS interrupt:
Write character function

    mov al, 'X'                          ; ASCII code for
space ('X' used for illustration)

    mov bh, 0                           ; Page number

    mov cx, 1                           ; Repeat 1 time

    int 10h                             ; Display a space by
moving the cursor back


    mov [di], byte 0                    ; Clear the character
in the buffer

    dec di                             ; Move the destination
index back

    jmp address_press                   ; Jump back to input
handling


address_enter:                          ; Label for handling
Enter key

    cmp di, BP_Buffer + 4               ; Check if the input
is complete

    jne address_press                    ; Jump back if not
complete


    mov di, ES_Buffer                   ; Reset destination
index to ES_Buffer

    mov si, ES_Hex                       ; Set source index
to ES_Hex
```

```
address_input_loop:                ; Loop to process the
address input

    cmp di, ES_Buffer + 8          ; Check if the end
of input buffer is reached

    je  address_input_done                  ;  Jump  to
address_input_done if yes


    mov al, [di + 2]               ; Load high nibble
of the byte

    shl al, 4                      ; Shift to the left
to make room for the low nibble

    or al, [di + 3]                ; OR operation to
combine with the low nibble

    mov ah, [di]                   ; Load high nibble
of the next byte

    shl ah, 4                      ; Shift to the left
to make room for the low nibble

    or ah, [di + 1]                ; OR operation to
combine with the low nibble

    mov word [si], ax              ; Store the combined
word into ES_Hex

    add di, 4                      ; Move to the next
set of hex digits

    add si, 2                      ; Move to the next
word in ES_Hex

    inc bl                         ; Increment bl (used
for tracking the loop)


    jmp address_input_loop             ;  Jump  back  to
continue the loop
```

```
address_input_done:                    ; Label indicating
the completion of address input

    mov byte [Error_Val], 0          ; Set Error_Val to 0
(no error)

    ret                                 ; Return from the
subroutine


address_escape:                        ; Label for handling
Esc key

    mov byte [Error_Val], 1          ; Set Error_Val to 1
(indicating an escape)

    ret                                 ; Return from the
subroutine


address_default:

    cmp di, BP_Buffer+4                ; Check if reached
the end of input buffer

    je address_press                      ; Jump if input
complete

    cmp al, '0'-1                      ; Check if character
is less than '0'

    jbe check_letters                  ; If less than '0',
check if it's a letter

    cmp al, '9'                        ; Check if character
is a digit

    mov bl, '0'                         ; Set bl to ASCII
value of '0'

    jbe check_successful               ; Jump if successful
```

```
check_letters:
    cmp al, 'a'-1                   ; Check if character
is less than 'a'
    jbe check_fail                  ; If less than 'a',
it's not a valid hex digit
    cmp al, 'f'                     ; Check if character
is in the range 'a' to 'f'
    ja check_fail                   ; If not in range,
it's not a valid hex digit
    mov bl, 'a'-10                  ; Set bl to adjust
ASCII value for 'a'-'f'


check_successful:
    mov ah, 0x0e                    ; BIOS interrupt:
Write character
    int 0x10                        ; Write the character
    sub al, bl                      ; Convert ASCII
character to hex value
    stosb                           ; Store the byte
value in ES:DI
    cmp di, BP_Buffer               ; Check if at the end
of the buffer
    jne address_press               ; Jump if not at the
end
    mov ah, 03h                     ; BIOS interrupt:
Move cursor position
    mov bh, 0                       ; Page number
    int 10h                         ; Move cursor position
```

```
    inc dl                              ; Increment column
position

    mov ah, 02h                        ; BIOS interrupt: Set
cursor position

    int 10h                            ; Set cursor position


check_fail:

    jmp address_press                   ; Continue with the
input


prompt_inputs:

    mov word [Address], Head_Prompt ; Display prompt for
head input

    call write_chr



head_input:

    call clean_line                 ; Clear the input line

    mov byte [N_of_Chars], 1       ; Set maximum character
count to 1

    call int_input                   ; Accept and process
user input


    cmp byte [Error_Val], 0        ; Check for input error

    jne go                             ; Jump if there's an
error
```

```
    mov al, [Digit_Buffer]        ; Get the digit entered
by the user

    cmp al, 2                      ; Compare the digit
with 2 (limit)

    jae head_input                 ; If greater than or
equal to 2, retry input


    mov [Head], al                 ; Store the valid input
in the Head variable


    call newline                   ; Move to a new line

    mov word [Address], Track_Prompt  ; Display prompt
for track input

    call write_chr


track_input:

    call clean_line                ; Clear the input line

    mov byte [N_of_Chars], 2       ; Set maximum character
count to 2

    call int_input                 ; Accept and process
user input


    cmp byte [Error_Val], 0        ; Check for input error

    jne go                         ; Jump if there's an
error


    mov al, [Digit_Buffer]         ; Get the digit entered
by the user
```

```
    cmp al, 80                          ; Compare the digit
with 80 (limit)

    jae track_input                     ; If greater than or
equal to 80, retry input


    mov [Track], al                     ; Store the valid input
in the Track variable


    call newline                        ; Move to a new line

    mov word [Address], Sector_Prompt   ; Display prompt
for sector input

    call write_chr


sector_input:

    call clean_line                     ; Clear the input line

    mov byte [N_of_Chars], 2            ; Set maximum character
count to 2

    call int_input                      ; Accept and process
user input


    cmp byte [Error_Val], 0             ; Check for input error

    jne go                              ; Jump if there's an
error


    mov al, [Digit_Buffer]              ; Get the digit entered
by the user

    cmp al, 0                           ; Compare the digit
with 0 (lower limit)
```

```
    je sector_input                ; If equal to 0, retry
input

    cmp al, 18                       ; Compare the digit
with 18 (upper limit)

    ja sector_input                ; If greater than 18,
retry input


    mov [Sector], al               ; Store the valid input
in the Sector variable

    ret                                  ; Return from the
subroutine


clean_screen:

    ; Set video mode to clear the screen

    mov ax, 0600h      ; AH = 06h (Scroll up window), AL
= 00h (Clear entire window)

    mov bh, 07h        ; Page number (default)

    xor cx, cx         ; Upper-left corner X-coordinate
(0)

    mov dx, 184fh      ; Lower-right corner Y-coordinate
(184fh = 24 decimal)

    int 10h            ; BIOS video services interrupt

    mov ah, 02h        ; Set cursor position function

    mov bh, 0          ; Page number (default)

    xor dx, dx         ; DH = 0, row; DL = 0, column

    int 10h            ; BIOS video services interrupt

    ret                ; Return from subroutine
```

```
print_to_hex_word:
    ; Print a word (2 bytes) in hexadecimal format
    mov bp, sp        ; Set base pointer to stack pointer
    push ax            ; Preserve AX register
    mov ah, 0eh        ; Teletype output function

    mov al, byte [bp + 3]   ; Get high byte of the word
    shr al, 4               ; Shift right to isolate the
upper nibble
    call print_to_hex_word_next

    mov al, byte [bp + 3]   ; Get high byte of the word
    and al, 0fh             ; Mask to isolate the lower
nibble
    call print_to_hex_word_next

    mov al, byte [bp + 2]   ; Get low byte of the word
    shr al, 4               ; Shift right to isolate the
upper nibble
    call print_to_hex_word_next

    mov al, byte [bp + 2]   ; Get low byte of the word
    and al, 0x0f            ; Mask to isolate the lower
nibble
    call print_to_hex_word_next
```

```
    pop ax              ; Restore AX register
    ret 2               ; Return from subroutine, removing
2 bytes from the stack


print_to_hex_word_next:
    cmp al, 0ah                         ; Check if the
character is greater than or equal to 0Ah
    jae print_to_hex_word_letter        ; If greater or
equal, it's a letter
    or al, 30h                          ; Convert the number
to ASCII representation
    jmp print_to_hex_word_done          ; Jump to the end of
the routine


print_to_hex_word_letter:
    add al, 37h           ; Convert the number to ASCII
letter representation


print_to_hex_word_done:
    int 10h               ; BIOS video services interrupt
to display the character
    ret                 ; Return from subroutine


print_to_hex_byte:
    ; Print a byte (8 bits) in hexadecimal format
    push ax             ; Preserve AX register
```

```
    mov ah, 0eh          ; Teletype output function


    mov al, bl           ; BL contains the byte to print
    shr al, 4              ; Shift right to isolate the
upper nibble
    call print_to_hex_byte_next  ; Call the next routine
to handle the printing


    mov al, bl           ; BL contains the byte to print
    and al, 0fh         ; Mask to isolate the lower nibble
    call print_to_hex_byte_next  ; Call the next routine
to handle the printing


    mov al, ' '          ; Print a space between the two
hexadecimal digits
    int 10h              ; BIOS video services interrupt
    pop ax               ; Restore AX register
    ret 2               ; Return from subroutine, removing
2 bytes from the stack


print_to_hex_byte_next:
    cmp al, 0ah            ; Check if the character is
greater than or equal to 0Ah
    jae print_to_hex_byte_letter  ; If greater or equal,
it's a letter
    or al, 30h             ; Convert the number to ASCII
representation
```

```
    jmp print_to_hex_byte_done     ; Jump to the end of
the routine


print_to_hex_byte_letter:

    add al, 37h           ; Convert the number to ASCII
letter representation


print_to_hex_byte_done:

    int 10h               ; BIOS video services interrupt
to display the character

    ret                   ; Return from subroutine


print_number:

    mov bp, sp            ; BP points to the top of the
stack

    xor dx, dx           ; Clear DX register for division

    mov ax, word [bp + 2]   ; Load the number to print
from the stack

    mov bx, ax            ; Preserve a copy of the number
in BX for subtraction


    ; Check the magnitude of the number for different
digit places

    cmp ax, 10000

    jae print_10000th

    cmp ax, 1000

    jae print_1000th
```

```
    cmp ax, 100
    jae print_100th
    cmp ax, 10
    jae print_10th
    jmp print_units       ; If number is less than 10, print the units place

print_10000th:
    mov cx, 10000         ; Load divisor for 10000
    div cx                ; Divide AX by 10000
    xor ax, 0e30h         ; Convert the quotient to ASCII character
    int 10h               ; Display the character
    xor ax, 0e30h         ; Clear AX for multiplication
    mul cx                ; Multiply quotient by 10000
    sub bx, ax            ; Subtract the product from the number
    mov ax, bx            ; Restore BX to AX for next division

; (Similar logic for print_1000th, print_100th, print_10th)

print_1000th:
    mov cx, 1000
    div cx
```

```
    xor ax, 0e30h

    int 10h
    xor ax, 0e30h
    mul cx
    sub bx, ax
    mov ax, bx

print_100th:
    mov cx, 100
    div cx
    xor ax, 0e30h
    int 10h
    xor ax, 0e30h
    mul cx
    sub bx, ax
    mov ax, bx

print_10th:
    mov cx, 10
    div cx
    xor ax, 0e30h
    int 10h
    xor ax, 0e30h
```

```
    mul cx
    sub bx, ax
    mov ax, bx


print_units:
    xor ax, 0e30h          ; Convert the units place to
ASCII character
    int 10h               ; Display the character


    mov ax, 0e00h         ; Display a new line
    int 10h               ; Using BIOS video services
    ret 2                  ; Return from the subroutine,
removing 2 bytes from the stack



write_Name_Group_1:
    mov ch, 45              ;1621/36 = nr of track
    mov cl, 2                  ;(1621 mod 18)+1 = nr of
sector
    mov dl, 0
    mov dh, 0
    mov ax, 0               ; Clear AX
    mov es, ax                ; Set ES to 0 (segment of
Name_Group 1)
    mov bx, Name_Group_1     ; Move Name_Group 1 address
to BX
```

```
    mov ax, 0301h          ; Set AH to 03 (write sectors)
and AL to 01 (sector count)
    int 13h


    mov ch, 45             ;1650/36 = nr of track
    mov cl, 13               ;(1650 mod 18)+1 = nr of
sector
    mov dl, 0
    mov dh, 0
    mov ax, 0              ; Clear AX
    mov es, ax               ; Set ES to 0 (segment of
Name_Group 1)
    mov bx, Name_Group_1     ; Move Name_Group 1 address
to BX
    mov ax, 0301h          ; Set AH to 03 (write sectors)
and AL to 01 (sector count)
    int 13h
    ret


write_Name_Group_2:
    mov ch, 30             ;1111/36 = nr of track
    mov cl, 14             ;1111 mod 18 = nr of sector
    mov dl, 0
    mov dh, 0
    mov ax, 0              ; Clear AX
```

```
    mov es, ax                ; Set ES to 0 (segment of
Name_Group 2)
    mov bx, Name_Group_2      ; Move Name_Group 2 address
to BX
    mov ax, 0301h           ; Set AH to 03 (write sectors)
and AL to 01 (sector count)
    int 13h


    mov ch, 31              ;1140/36 = nr of track
    mov cl, 7               ;1140 mod 18 = nr of sector
    mov dl, 0
    mov dh, 0
    mov ax, 0               ; Clear AX
    mov es, ax                ; Set ES to 0 (segment of
Name_Group 2)
    mov bx, Name_Group_2      ; Move Name_Group 2 address
to BX
    mov ax, 0301h           ; Set AH to 03 (write sectors)
and AL to 01 (sector count)
    int 13h
    ret


write_Name_Group_3:
    mov ch, 35              ;1291/36 = nr of track
    mov cl, 14                ;(1291 mod 18)+1 = nr of
sector
    mov dl, 0
```

```
    mov dh, 0
    mov ax, 0                ; Clear AX
    mov es, ax                 ; Set ES to 0 (segment of
Name_Group 2)
    mov bx, Name_Group_3     ; Move Name_Group 3 address
to BX
    mov ax, 0301h          ; Set AH to 03 (write sectors)
and AL to 01 (sector count)
    int 13h


    mov ch, 36               ;1320/36 = nr of track
    mov cl, 7                  ;(1320 mod 18)+1 = nr of
sector
    mov dl, 0
    mov dh, 0
    mov ax, 0                ; Clear AX
    mov es, ax                 ; Set ES to 0 (segment of
Name_Group 2)
    mov bx, Name_Group_3     ; Move Name_Group 3 address
to BX
    mov ax, 0301h          ; Set AH to 03 (write sectors)
and AL to 01 (sector count)
    int 13h
    ret


Options db "1. Keyboard -> Floppy", 0dh, 0ah, "2. Floppy
-> RAM", 0dh, 0ah, "3. RAM -> Floppy", 0dh, 0ah
```

```
Choose db "Choose option:", 0dh, 0ah, 0,


Text_Prompt db "Input Text", 0dh, 0ah, 0

N_Prompt db "N - Number of N [1-30000]", 0dh, 0ah, 0

Head_Prompt db "Head [0-1]", 0dh, 0ah, 0

Track_Prompt db "Track [0-79]", 0dh, 0ah, 0

Sector_Prompt db "Sector [1-18]", 0dh, 0ah, 0

Q_Prompt db "Q - Number of Bytes [1-32767] (Rounded up
to multiples of 512)", 0

Head_Input db "Head: ", 0

Track_Input db "Track: ", 0

Sector_Input db "Sector: ", 0

ESBP_Input db "ES:BP: ", 0

Space_Key db "Press Space for to turn page", 0


Keyboard_to_Floppy_success  db  "Keyboard_to_Floppy  -
success, press Enter.", 0

Keyboard_to_Floppy_fail db "Keyboard_to_Floppy - fail,
error: ", 0

Floppy_To_RAM_success db "Floppy_To_RAM - success, press
Space.", 0

Floppy_To_RAM_fail db "Floppy_To_RAM - fail, error: ", 0

RAM_To_Floppy_success db "RAM_To_Floppy - success.", 0

RAM_To_Floppy_fail db "RAM_To_Floppy - fail, error: ", 0
```

```
N dw 0
Q dw 0
Head db 0
Track db 0
Sector db 0
Address dw 0
N_of_Chars db 0
Digits db 0
Digit_Buffer dw 0

ES_Hex dw 0
BP_Hex dw 0

Error_Val db 0

Address_Prompt db "ES:BP "
Address_Prompt_length equ $-Address_Prompt
Address_Input db "XXXX:XXXX"
Address_Input_length equ $-Address_Input

ES_Buffer times 4 db 0
BP_Buffer times 4 db 0

Text_Buffer times 256 db 0
```

```
Memory_Buffer times 512 db 0

Name_Group_1 times 10 db "@@@FAF-212 Anatolie TELUG###",
0

Name_Group_2 times 10 db "@@@FAF-212 Alexei CIUMAC###",
0

Name_Group_3   times   10   db   "@@@FAF-212   Vladimir
LUCHIANOV###", 0
```

Figure 1. Output of the program with first choice

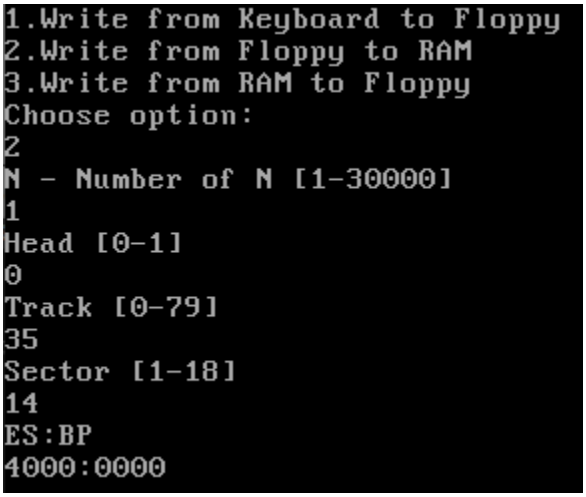

Figure 2. Input for the second choice

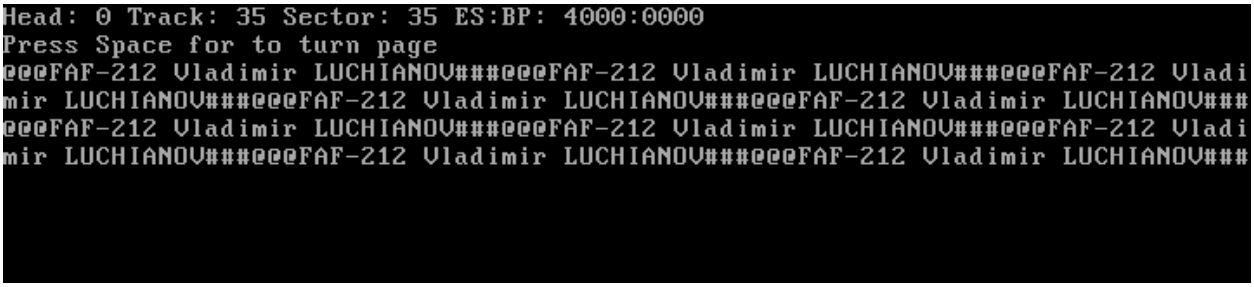

Figure 3. Output of the second choice

Figure 4. Output for the third choice

To build an image file from this .asm file we simply compile it with NASM program and truncate it to 1474560 bytes.

Conclusions:

In this laboratory work, we focused on tasks related to data transfer between different storage media and system memory using x86 assembly language. The three main functions included reading from the keyboard and writing to a floppy disk, reading from a floppy disk and transferring data to RAM, and writing data from RAM to a floppy disk. The implementation involved user input, error handling, and clear presentation of results on the screen, providing valuable insights into low-level programming and data management.

Additionally, the scope extended to building image files using Linux shell operations, broadening the understanding of file manipulation and system-level operations in a Unix-like environment.