

**Ministry of Education, Culture and Research of the Republic of Moldova**

**Technical University of Moldova**

**Department of Software and Automation Engineering**

**REPORT**

Laboratory work no. 4

*Bootloader*

Elaborated:

st. gr. FAF-213 Botnari Ciprian

Verified:

lect. univ. Rostislav Călin

Chişinău – 2023

Table of Contents

[Topic: Bootloader 3](#_Toc153201250)

[Tasks 3](#_Toc153201251)

[Code 3](#_Toc153201252)

[Build script 3](#_Toc153201253)

[Code 4](#_Toc153201254)

[**print\_string.asm** 5](#_Toc153201255)

[**init-bootloader.asm** 5](#_Toc153201256)

[**bootloader.asm** 5](#_Toc153201257)

[**main.asm** 12](#_Toc153201258)

[Conclusion 18](#_Toc153201259)

[Github 18](#_Toc153201260)

# Topic: Bootloader

# Tasks

Create in assembler language an application that will act as Boot Loader and will do the following:

1. It will display a hello message that will include the author's name and wait for the keyboard to enter the "source" address on the floppy, from which to read the kernel (or other compiled code that is intended to be loaded and executed).

* The address will be entered in SIDE, TRACK, SECTOR format and the address must be strictly in the student author's reserved slide as it was in Lab3.

2. Will wait for the keyboard to read the "destination" address of the RAM where the data block read from the floppy will be loaded.

* The RAM address must be in the format XXXXh:XXXXh, identical as it was for Lab3.

3. It will transfer the FLOPPY ==> RAM data and display the error code with which the given operation was completed.

4. Will display a message to press a key and start the kernel (or to execute the code to be executed).

5. When the execution of the kernel or the executed code is complete, it will display a message to press a key and execute the Boot Loader again!

# Code

**NASM Version**: 2.16.01

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

## Build script

#!/bin/bash

binary\_file="$1"

bootloader="$2"

init\_bootloader="$3"

# Check if the binary file exists

nasm -f bin $binary\_file -o main.bin

nasm -f bin $bootloader -o bootloader.bin

nasm -f bin $init\_bootloader -o init\_bootloader.bin

# Create an empty floppy disk image (1.44MB size)

floppy\_image="floppy.img"

truncate -s 1474560 init\_bootloader.bin

mv init\_bootloader.bin $floppy\_image

dd if="bootloader.bin" of="$floppy\_image" bs=512 seek=1 conv=notrunc

dd if="main.bin" of="$floppy\_image" bs=512 seek=3 conv=notrunc

echo "Binary file '$binary\_file' successfully added to floppy image '$floppy\_image'."

VM\_NAME="BestOS"

VBoxManage controlvm "$VM\_NAME" poweroff

echo "Virtual Machine $VM\_NAME closed."

sleep 3

# Step 5: Change the storage to $flp\_file in VirtualBox

VBoxManage storageattach "$VM\_NAME" --storagectl "Floppy" --port 0 --device 0 --type fdd --medium "$floppy\_image"

echo "Step 5: Storage in VirtualBox changed to $floppy\_image."

# Step 6: Start the Virtual Machine

VBoxManage startvm "$VM\_NAME"

echo "Step 6: Virtual Machine $VM\_NAME started."

echo "All steps completed successfully."

## Code

Here is the floppy space distribution from lab 3.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Nr | Group | Student | Block | Start | End | Bytes |
| 6 | FAF-213 | Botnari Ciprian | 66 | 1951 | 1980 | 15360 |

**Table 1**. Floppy space distribution

We applied the following formula to write to floppy:

Let’s apply the formula for

Thus we obtain the following results:

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

For the end = 1980, we obtain the following results:

* Track = 30
* Sector = 1
* Head = 1, since

### **print\_string.asm**

print\_string\_si:

    push ax

    mov ah, 0x0e

    call print\_next\_char

    pop ax

    ret

print\_next\_char:

    mov al, [si]

    cmp al, 0

    jz if\_zero

    int 0x10

    inc si

    jmp print\_next\_char

if\_zero:

    ret

### **init-bootloader.asm**

org 7c00h

mov ah, 00

int 13h

mov ax, 0000h

mov es, ax

mov bx, 7d00h

mov ah, 02h

mov al, 2

mov ch, 0

mov cl, 2

mov dh, 0

mov dl, 0

int 13h

jmp 0000h:7d00h

times 510-($-$$) db 0

dw 0AA55h

### **bootloader.asm**

org 7d00h

mov byte [marker], 0

; print initial prompt

mov si, prompt

call print

; read option

mov ah, 00h

int 16h

; display character as TTY

mov ah, 0eh

mov bl, 07h

int 10h

call newline

mov si, hts\_prompt

call print

call newline

; print sector count prompt

mov ah, 0eh

mov al, '#'

mov bl, 07h

int 10h

mov byte [result], 0

call clear

call read\_buffer

mov al, [result]

mov byte [sc], al

call newline

; print head prompt

mov ah, 0eh

mov al,'#'

mov bl, 07h

int 10h

mov byte [result], 0

call clear

call read\_buffer

mov al, [result]

mov byte [h], al

call newline

; print track prompt

mov ah, 0eh

mov al,'#'

mov bl, 07h

int 10h

mov byte [result], 0

call clear

call read\_buffer

mov al, [result]

mov byte [t], al

call newline

; print sector prompt

mov ah, 0eh

mov al,'#'

mov bl, 07h

int 10h

mov byte [result], 0

call clear

call read\_buffer

mov al, [result]

mov byte [s], al

call newline

call newline

inc byte [marker]

; print ram address prompt

mov si, so\_prompt

call print

call newline

; print segment prompt

mov ah, 0eh

mov al,'#'

mov bl, 07h

int 10h

call clear

call read\_buffer

mov ax, [hex\_result]

mov [add1], ax

call newline

; print offset prompt

mov ah, 0eh

mov al,'#'

mov bl, 07h

int 10h

call clear

call read\_buffer

mov ax, [hex\_result]

mov [add2], ax

call newline

call load\_kernel

; print a prompt to load the kernel

mov si, kernel\_start

call newline

call print

; read option

mov ah, 00h

int 16h

; display character as TTY

mov ah, 0eh

mov bl, 07h

int 10h

call newline

call newline

; remember segment and offset in ax:bx

mov ax, [add1]

mov bx, [add2]

; jump to the loaded NASM script

add ax, bx

jmp ax

load\_kernel:

mov ah, 0h

int 13h

mov ax, [add2]

mov es, ax

mov bx, [add1]

; load the NASM script into memory

mov ah, 02h

mov al, [sc]

mov ch, [t]

mov cl, [s]

mov dh, [h]

mov dl, 0

int 13h

; print error code

mov al, '0'

add al, ah

mov ah, 0eh

int 10h

call newline

ret

read\_buffer:

read\_char:

; read character

mov ah, 00h

int 16h

; check if the ENTER key was introduced

cmp al, 0dh

je hdl\_enter

; check if the BACKSPACE key was introduced

cmp al, 08h

je hdl\_backspace

; add character into the buffer and increment its pointer

mov [si], al

inc si

inc byte [c]

; display character as TTY

mov ah, 0eh

mov bl, 07h

int 10h

jmp read\_char

hdl\_enter:

mov byte [si], 0

mov si, buffer

cmp byte [marker], 0

je atoi\_jump

jmp atoh\_jump

hdl\_backspace:

call cursor

cmp byte [c], 0

je read\_char

; clear last buffer char

dec si

dec byte [c]

; move cursor to the left

mov ah, 02h

mov bh, 0

dec dl

int 10h

; print space instead of the cleared char

mov ah, 0ah

mov al, ' '

mov bh, 0

mov cx, 1

int 10h

jmp read\_char

atoi\_jump:

call atoi

jmp end\_read\_buffer

atoh\_jump:

call atoh

jmp end\_read\_buffer

end\_read\_buffer:

ret

atoi:

xor ax, ax

xor bx, bx

atoi\_d:

lodsb

sub al, '0'

xor bh, bh

imul bx, 10

add bl, al

mov [result], bl

dec byte [c]

cmp byte [c], 0

jne atoi\_d

ret

atoh:

xor bx, bx

mov di, hex\_result

atoh\_s:

xor ax, ax

mov al, [si]

cmp al, 65

jg atoh\_l

sub al, 48

jmp continue

atoh\_l:

sub al, 55

jmp continue

continue:

mov bx, [di]

imul bx, 16

add bx, ax

mov [di], bx

inc si

dec byte [c]

jnz atoh\_s

ret

print:

call cursor

print\_char:

mov al, [si]

cmp al, '$'

je end\_print

mov ah, 0eh

int 10h

inc si

jmp print\_char

end\_print:

ret

clear:

mov byte [c], 0

mov byte [si], 0

mov si, buffer

ret

cursor:

mov ah, 03h

mov bh, 0

int 10h

ret

newline:

call cursor

mov ah, 02h

mov bh, 0

inc dh

mov dl, 0

int 10h

ret

section .data:

prompt db 'Welcome, to Ciprian Bootloader. Press any key to continue: $'

hts\_prompt db "Enter N, Head, Track, Sector: $"

so\_prompt db "Enter RAM address in this format XXXX:YYYY $"

kernel\_start db "Press any key to load the kernel: $"

sc db 0

h db 0

t db 0

s db 0

c db 0

result db 0

marker db 0

section .bss:

hex\_result resb 2

add1 resb 2

add2 resb 2

buffer resb 2

dw 0AA55h

### **main.asm**

org 4000h

bits 16

jmp start

%include "print\_string.asm"

start:

mov al, 0x3

mov ah, 0

int 0x10

mov si, help

call print\_string\_si

mov bx, 0

mov byte [counter], 0

mov si, buffer

read\_key:

mov ah, 0

int 0x16 ; Read keypress

cmp ah, 0x0e ; Backspace

je input\_bksp

cmp ah, 0x1c ; Enter

je input\_enter

cmp al, 0x2f ; Slash '/'

je go\_back

cmp al, 0x20

jge echo\_char

jmp read\_key ; Always read for keyboard inputs

input\_bksp:

cmp si, buffer ; Check if buffer is empty

je read\_key

dec si

dec bx

dec byte [counter]

mov byte [si], 0 ; Delete last char in buffer

mov ah, 0x03

mov bh, 0

int 0x10

cmp dl, 0 ; Check if cursor is at the start of the line

jz prev\_line

jmp prev\_char

prev\_char:

mov ah, 0x02

dec dl

int 0x10

jmp overwrite\_char

prev\_line:

mov ah, 0x02

mov dl, 79

dec dh

int 0x10

overwrite\_char:

mov ah, 0xa

mov al, 0x20

mov cx, 1

int 0x10

jmp read\_key

reverse\_buffer:

push si

push di

push ax

push cx

dec bx

lea si, [buffer + bx]

mov di, buffer\_rev

mov cx, 0

rev\_loop:

mov al, [si]

mov [di], al

dec si

inc di

inc cx

cmp cx, 255

jne rev\_loop

pop ax

pop cx

pop di

pop si

jmp print\_echo

is\_palindrome:

push si

push di

push ax

push cx

push dx

inc byte [result]

mov si, buffer

mov di, buffer\_rev

mov cx, 0

comp:

mov al, [si]

mov dl, [di]

cmp al, dl

jne not\_equal

cmp cx, [counter]

je equal

inc si

inc di

inc cx

cmp cx, [counter]

jne comp

equal:

pop si

pop di

pop ax

pop cx

pop dx

jmp print\_result

not\_equal:

pop si

pop di

pop ax

pop cx

pop dx

dec byte [result]

jmp print\_result

input\_enter:

mov ah, 0x03

mov bh, 0

int 0x10 ; x = DL, y = DH

sub si, buffer ; Check if buffer is empty

jz write\_newline

mov ah, 0x03 ; DL, DH store cursor (x,y) positions

mov bh, 0

int 0x10

cmp dh, 24

jge start

cmp dh, 24

jl reverse\_buffer

mov ah, 0x06 ; Scroll down once to make space for the string

mov al, 1

mov bh, 0x07 ; Draw new line as White on Black

mov cx, 0 ; (0,0): Top-left corner of the screen

mov dx, 0x184f ; (79,24): Bottom-right corner of the screen

int 0x10

mov dh, 0x17 ; Move cursor 1 line above target

print\_echo:

mov bh, 0 ; Video page number.

mov ax, 0

mov es, ax ; ES:BP is the pointer to the buffer

mov bp, buffer\_rev

mov bl, 14 ; Attribute: Yellow on Black

mov cx, si ; String length

inc dh ; y coordinate

mov dl, 0 ; x coordinate

mov ax, 0x1301 ; Write mode: character only, cursor moved

int 0x10

write\_newline:

cmp dh, 24 ; Last line of the screen

je scroll\_down ; Scroll screen down 1 line

mov ah, 0x03 ; DL, DH store cursor (x,y) positions

mov bh, 0

int 0x10

jmp move\_down

scroll\_down:

mov ah, 0x06

mov al, 1

mov bh, 0x07 ; Draw new line as White on Black

mov cx, 0 ; (0,0): Top-left corner of the screen

mov dx, 0x184f ; (79,24): Bottom-right corner of the screen

int 0x10

mov dh, 0x17 ; Move cursor 1 line above target

move\_down:

mov ah, 0x02 ; Move the cursor at the start of the line below this one

mov bh, 0

inc dh

mov dl, 0

int 0x10

jmp is\_palindrome

clear\_buffer:

mov byte [si], 0 ; Replace every non 0 byte to 0 in the buffer

inc si

cmp si, 0

jne clear\_buffer

; Print new line

mov ah, 0x0e

mov al, 0x0a

int 0x10

mov al, 0x0a

int 0x10

mov al, 0x08

int 0x10

add si, buffer\_rev

jmp clean\_buffer\_rev

mov si, buffer

jmp read\_key

print\_result:

cmp byte [result], 1

je print\_true

; Not palindrome

mov ah, 0x0e

mov al, '0'

int 0x10

add si, buffer

jmp clear\_buffer

print\_true:

mov ah, 0x0e

mov al, '1'

int 0x10

add si, buffer

clean\_buffer\_rev:

mov byte [si], 0 ; Replace every non 0 byte to 0 in the buffer

inc si

cmp si, 0

jne clean\_buffer\_rev

mov bx, 0

mov byte [counter], 0

mov si, buffer

jmp read\_key

echo\_char:

cmp si, buffer + 255 ; If buffer is at max size (255), ignore further inputs

je read\_key

mov [si], al

inc si

inc bx

inc byte [counter]

mov ah, 0xe ; Echo any valid characters to screen

int 10h

jmp read\_key

go\_back:

mov al, 0x3

mov ah, 0

int 0x10

mov ah, 00

int 13h

mov ax, 0000h

mov es, ax

mov bx, 7d00h

mov ah, 02h

mov al, 5

mov ch, 0

mov cl, 2

mov dh, 0

mov dl, 0

int 13h

jmp 0000h:7d00h

help: db "Check if string is a palindrome (0 = false, 1 = true): ", 0xd, 0xa, 0

help\_len: equ $-help

buffer: times 256 db 0x0 ; Empty 256 char buffer for our code

buffer\_rev: times 256 db 0x0 ; Empty 256 char buffer for our code

result: db 0

counter: db 0

# Conclusion

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

Github: [Sufferal/os (github.com)](https://github.com/Sufferal/os)