MINISTRY OF EDUCATION, CULTURE AND RESEARCH OF REPUBLIC OF MOLDOVA

TECHNICAL UNIVERSITY OF MOLDOVA

FACULTY OF COMPUTERS, INFORMATICS AND MICROELECTRONICS

DEPARTMENT OF SOFTWARE ENGINEERING AND AUTOMATICS

Operating Systems

Laboratory work 1: Video Output

Elaborated:	
st.gr. FAF-212	Cristian Brinza
Verified:	Rostislav Calin

Chișinău, 2023

Task:

Create a program in assembler which will print text to the screen.

Students should respect the following conditions:
1. ALL possible methods should be used in order to print text.
(update):
- Method1: Write character as TTY
- M2: Write character
- M3: Write character/attribute
- M4: Display character + attribute
- M5: Display character + attribute & update cursor- M6: Display string
- M6: Display string
- M7: Display string & update cursor
- M8(optional): Print directly to video memory

2. Compiled program should be used in order to create a floppy image and it should be bootable. Use this image to boot the OS in a VirtualBox VM and the text which you intended to print should appear on the screen.
3. You can use any assembly compiler.
4. Students should be able to modify the code, to recompile it and to boot the VM with new version of program.
5. In order to use documentation from TechHelp/XView DOS application, students can install DosBox.

*** This instructions can update with some new details if it will be required by the teaching process.

Execution:

1. **Method1 (TTY mode function):** Uses the **10h** interrupt with **AH** set to **0aH**.
2. **Method2 (Write character):** Uses the **10h** interrupt with **AH** set to **09h**.
3. **Method3 (Write character/attribute):** Uses the **10h** interrupt with **AH** set to **09h** and an attribute set in **BL**.
4. **Method4 (Display character + attribute):** Displays a character with a given attribute without updating the cursor.
5. **Method5 (Display character + attribute & update cursor):** Displays a character with a given attribute and updates the cursor position.
6. **Method6 (Display string):** Displays a string of characters.
7. **Method7 (Display string & update cursor):** Displays a string of characters and updates the cursor position.
8. **Method8 (Print directly to video memory - optional):** Directly writes the character and attribute to video memory.

```
ORG 100h              ; origin, standard for .COM files

; Method 1: Write character as TTY
mov ah, 09h
mov dx, msg
int 21h
call newline

; Method 2: Write character
mov ah, 0Eh
mov al, 'a'
int 10h
call newline

; Method 3: Write character/attribute
mov ah, 09h
mov al, 'b'
mov bl, 07h          ; attribute (gray on black)
mov cx, 1
int 10h
call newline

; Method 4: Display character + attribute
mov ah, 09h
mov al, 'c'
mov bl, 0Fh          ; attribute (bright white on black)
mov cx, 1
int 10h
call newline

; Method 5: Display character + attribute & update cursor
mov ah, 09h
mov al, 'd'
mov bl, 02h          ; attribute (green on black)
mov cx, 1
int 10h
call newline

; Method 6: Display string
mov ah, 09h
mov dx, msg
int 21h
call newline

; Method 7: Display string & update cursor
mov ah, 09h
mov dx, msg
int 21h
call newline

; Method 8: Print directly to video memory
mov ax, 0B800h
mov es, ax
mov di, 24*160       ; 14th line
mov ax, 0700h | 'a'  ; attribute and character
mov [es:di], ax
call newline

; Exit
mov ah, 4Ch
int 21h

newline:
    mov ah, 02h
    mov dl, 0Ah
    int 21h
    mov dl, 0Dh
    int 21h
    ret

msg: db 'a$', 0Ah, 0Dh, '$'
```

a. Writing the Assembly Code:

The first step is to write the assembly code for the bootloader. The code is written in a human-readable format, using mnemonics to represent machine instructions.

For example, the instruction **mov ax, 07h** moves the hexadecimal value **07** into the **AX** register.

b. Assembling the Code:

Once the assembly code is written, it needs to be converted into machine code. This is done using an assembler like NASM.

Command used:

```
nasm -f bin -o bootloader.bin bootloader.asm
```

c. Creating a Blank Floppy Image:

A blank floppy image is created to simulate a floppy disk. This image will be used to boot the system in VirtualBox.

Command used:

```
dd if=/dev/zero of=floppy.img bs=512 count=2880
```

d. Copying the Binary Code to the Floppy Image:

The machine code (binary code) generated from the assembly code is then copied to the floppy image.

Command used:

```
dd if=bootloader.bin of=floppy.img conv=notrunc
```

e. Booting the Image in VirtualBox:

The final step is to boot the floppy image in VirtualBox. This is done by creating a new Virtual Machine, adding a floppy controller, and attaching the **floppy.img** file to it.

Commented functionality:

```
BITS 16                 ; Specify the target mode of the processor. 16-bit mode for real mode.
ORG 0x7C00              ; Set the origin. This is where the bootloader is loaded in memory by the
BIOS.

start:
    cli                 ; Disable interrupts to ensure no interruptions during the bootloader
execution.
    xor ax, ax          ; Set AX register to 0. XORing a register with itself is a common way to
clear it.
    mov ds, ax          ; Set DS (Data Segment) register to 0.
    mov es, ax          ; Set ES (Extra Segment) register to 0.
    mov ss, ax          ; Set SS (Stack Segment) register to 0.
    mov sp, 0x7C00      ; Set SP (Stack Pointer) to 0x7C00, initializing the stack.

    call display_methods ; Call the display_methods subroutine to display the character using
various methods.

    ; Infinite loop to halt the CPU after execution
    hang:
        hlt            ; Halt instruction stops the CPU execution.
        jmp hang       ; Jump to the hang label, creating an infinite loop.

display_methods:
    ; Method 1: Write character as TTY using DOS interrupt
    mov ah, 09h        ; AH=09h specifies the function to display a string.
    mov dx, msg        ; Point DX to the message string.
    int 21h            ; Call DOS interrupt 21h to display the string.
    call newline       ; Call the newline subroutine to move to the next line.

    ; Method 2: Teletype character using BIOS interrupt
    mov ah, 0Eh        ; AH=0Eh specifies the teletype function.
    mov al, 'a'        ; Load the character 'a' into AL.
    int 10h            ; Call BIOS interrupt 10h to display the character.
    call newline       ; Move to the next line.

    ; Method 3: Display character with attribute using BIOS interrupt
    mov ah, 09h         ; AH=09h specifies the function to display a character with an attribute.
    mov al, 'a'         ; Load the character 'a' into AL.
    mov bl, 07h         ; BL=07h sets the attribute (gray on black).
    mov cx, 1           ; CX specifies how many times to print the character.
    int 10h             ; Call BIOS interrupt 10h.
    call newline        ; Move to the next line.

 ; Method 4: Display character with a different attribute
    mov ah, 09h         ; AH=09h: Set the BIOS function to display a character with an attribute.
    mov al, 'a'         ; AL='a': Load the character 'a' into the AL register for display.
    mov bl, 0Fh         ; BL=0Fh: Set the attribute to bright white text on a black background.
    mov cx, 1           ; CX=1: Specify that the character should be printed only once.
    int 10h             ; Call BIOS interrupt 10h to execute the display function.
    call newline        ; Call the newline subroutine to move the cursor to the next line.

    ; Method 5: Display character, update attribute & cursor
    mov ah, 09h         ; AH=09h: Set the BIOS function to display a character with an attribute.
    mov al, 'a'         ; AL='a': Load the character 'a' into the AL register for display.
    mov bl, 02h         ; BL=02h: Set the attribute to green text on a black background.
    mov cx, 1           ; CX=1: Specify that the character should be printed only once.
    int 10h             ; Call BIOS interrupt 10h to execute the display function.
    call newline        ; Call the newline subroutine to move the cursor to the next line.

    ; Method 6: Display string using DOS interrupt
    mov ah, 09h         ; AH=09h: Set the DOS function to display a string.
    mov dx, msg         ; DX=msg: Point the DX register to the address of the msg string.
    int 21h             ; Call DOS interrupt 21h to execute the display function.
    call newline        ; Call the newline subroutine to move the cursor to the next line.

    ; Method 7: Display string and update cursor
    mov ah, 09h         ; AH=09h: Set the DOS function to display a string.
    mov dx, msg         ; DX=msg: Point the DX register to the address of the msg string.
    int 21h             ; Call DOS interrupt 21h to execute the display function.
    call newline        ; Call the newline subroutine to move the cursor to the next line.


    ; Method 8: Print directly to video memory
    mov ax, 0B800h      ; Video memory segment for color displays.
    mov es, ax          ; Set ES to point to video memory.
    mov di, 14*160      ; DI points to the 14th line on the screen.
    mov ax, 0700h | 'a'; Combine attribute (gray on black) with character 'a'.
    mov [es:di], ax     ; Write to video memory.
    call newline

    ret                 ; Return from the display_methods subroutine.

newline:
    ; Print a newline character using DOS interrupt
    mov ah, 02h         ; AH=02h specifies the function to display a character.
    mov dl, 0Ah         ; ASCII for newline.
    int 21h
    mov dl, 0Dh         ; ASCII for carriage return.
    int 21h
    ret                 ; Return from the newline subroutine.
```

```
msg: db 'a$', 0Ah, 0Dh, '$' ; Define the message string. '$' is a string terminator for DOS
functions.

; Bootloader padding and signature
times 510-($-$$) db 0  ; Fill the remaining bytes with 0 to make the bootloader 512 bytes.
dw 0xAA55              ; Boot signature. This tells the BIOS that it's a valid bootloader.
```

A step-by-step breakdown of the code:

1. **Method 1 (TTY Mode)**: We'll use the DOS interrupt **int 21h** with function **09h** to display a string. This function expects the address of the string in the **DX** register and the string should be terminated by a **$** character.
2. **Method 2 (Write Character)**: We'll use the BIOS interrupt **int 10h** with function **0Eh** to teletype a character. This function expects the character in the **AL** register and the page number in **BH**.
3. **Method 3 (Write Character/Attribute)**: We'll use the BIOS interrupt **int 10h** with function **09h** to display a character and attribute. This function expects the character in **AL**, the attribute in **BL**, the page number in **BH**, and the number of times to write the character in **CX**.
4. **Method 4 (Display Character + Attribute)**: Similar to Method 3 but with a different attribute.
5. **Method 5 (Display Character + Attribute & Update Cursor)**: We'll first display the character with an attribute and then move the cursor to the next line.
6. **Method 6 (Display String)**: Similar to Method 1 but with a different string.
7. **Method 7 (Display String & Update Cursor)**: We'll display a string and then move the cursor to the next line.
8. **Method 8 (Print Directly to Video Memory)**: We'll write directly to the video memory segment **B800:0000** for color displays.

Key components:

- **BITS 16**: This tells the assembler that we're working in 16-bit mode, which is the mode used by the BIOS when the computer starts.
- **ORG 0x7C00**: The BIOS loads the bootloader at memory address **0x7C00**. This directive tells the assembler to start the program at this address.
- **cli**: This instruction disables interrupts. Interrupts are signals that can pause the CPU to handle external events. By disabling them, we ensure our code runs without interruption.

- **xor, mov**: These are data movement and manipulation instructions. They set up the initial environment for our bootloader.
- **call**: This instruction calls a subroutine. In this case, it's calling the **display_methods** subroutine.
- **hlt**: This instruction halts the CPU. It's used here to stop the CPU after our bootloader has finished running.
- **jmp hang**: This creates an infinite loop, ensuring the CPU doesn't try to execute random data after the bootloader.
- **times 510-($-$$) db 0**: This is a NASM directive. It fills the bootloader with zeros until it's 510 bytes long. This is because the bootloader must be exactly 512 bytes, with the last two bytes being the signature **0xAA55**.
- **dw 0xAA55**: This is the bootloader signature. The BIOS looks for this signature to identify a bootable disk.

Conclusion:

Through this laboratory work, we have gained a basic understanding of assembly language programming and the process of creating a bootable floppy image. We have successfully created a simple bootloader that displays text on the screen and tested it in a virtual environment using VirtualBox.

References:

1. "Assembly Language for x86 Processors" by Kip R. Irvine
2. VirtualBox User Manual
3. NASM Documentation