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. 1
la cursul Sisteme de Operare - “Text print”

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

Chișinău - 2023

Subject: *NASM text print options*

Tasks: *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***:**
    - a. *M1: Write character as TTY;*
    - b. *M2: Write character;*
    - c. *M3: Write character/attribute;*
    - d. *M4: Display character + attribute;*
    - e. *M5: Display character + attribute & update cursor;*
    - f. *M6: Display string;*
    - g. *M7: Display string & update cursor;*
    - h. *M8: 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 a new version of the program.*
5. *In order to use documentation from TechHelp/XView DOS application, students can install DosBox.*

Implementation:

* In order to compile the assembly code and create an image that is possible to run in a VM from floppy I use the following bash script:

```
#!/bin/bash

if [ $# -ne 1 ]; then
   echo "Usage: $0 <filename>"
   exit 1
fi

filename="$1"
ext1=".asm"
ext2=".com"

cp "$filename$ext1" backup/
rm -f "floppy.img"

nasm -f bin -o "$filename$ext2"
"$filename$ext1"
truncate -s 1474560 "$filename$ext2"
mv "$filename$ext2" floppy.img
```

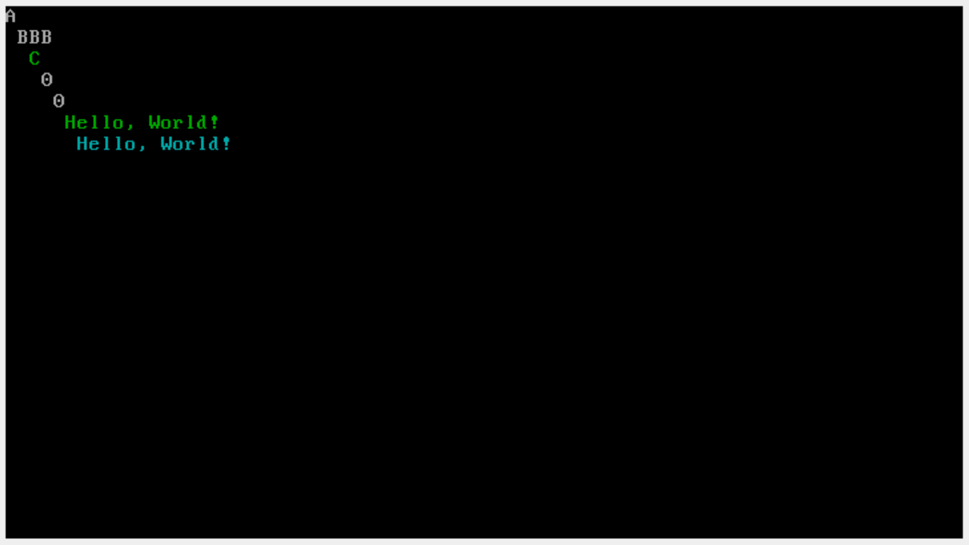
On this page you may find the explicitly commented source code of the assembly program implementing methods from 1 to 3:

```
mov     AH, 0xE         ; 0xE - write a char as TTY (M1)
mov     AL, 'A'         ; Char to display
int     0x10            ; Call the Video ServicES BIOS Interrupt

mov     DH, 0x1         ; 2nd row
mov     AH, 0x2         ; Move cursor
mov     DL, 0x1         ; 2nd column
int     0x10            ; Call the Video ServicES BIOS Interrupt

; ================

mov     AH, 0xA         ; 0xA - write character (M2)
mov     AL, 'B'         ; Char to display
mov     CX, 0x3         ; 3 timES
int     0x10            ; Call the Video ServicES BIOS Interrupt

mov     AH, 0x2         ; Move cursor
mov     DH, 0x2         ; 3rd row
mov     DL, 0x2         ; 3rd column
int     0x10            ; Call the Video ServicES BIOS Interrupt

; ================

mov     AH, 0x9         ; 0xA - write character/attribute (M3)
mov     AL, 'C'         ; Char to display
mov     BL, 0x2         ; Text color (green)
mov     CX, 0x1         ; 1 time
int     0x10            ; Call the Video ServicES BIOS Interrupt
```

On the next page the code for the next 4 methods (1300h - 1303h) is listed…

```
mov    AX, 0x0       ; ?
mov    ES, AX        ; ?
mov    CX, 0x1       ; 1 character to display
mov    DH, 0x3       ; On the 4th row
mov    DL, DH        ; In the 4th column
mov    BP, char      ; The character to display
mov    AX, 1302h     ; 1302h - display character/attribute cells
int    0x10          ; Call the Video ServicES BIOS Interrupt

; ===============

mov    AX, 0x0       ; ?
mov    ES, AX        ; ?
mov    CX, 0x1       ; 1 character to display
mov    DH, 0x4       ; On the 5th row
mov    DL, DH        ; In the 5th column
mov    BP, char      ; The character to display
mov    AX, 1303h     ; 1302h - display character/attribute cells
int    10h           ; Call the Video ServicES BIOS Interrupt

; ================

mov    AX, 0x0       ; Prepare memory
mov    ES, AX        ; Prepare memory
mov    BL, 0x2       ; Text color (green)
mov    CX, 0xF       ; 15 characters to display
mov    DH, 0x5       ; On the 6th row
mov    DL, DH        ; In the 6th column
mov    BP, string    ; The string to display
mov    AX, 1300h     ; 1300h - display string
int    0x10          ; Call the Video ServicES BIOS Interrupt

; ================

mov    AX, 0x0       ; Prepare memory
mov    ES, AX        ; Prepare memory
mov    BL, 0x3       ; Text color (cyan)
mov    CX, 0xF       ; 15 characters to display
mov    DH, 0x6       ; On the 7th row
mov    DL, DH        ; In the 7th column
mov    BP, string    ; The string to display
mov    AX, 1301h     ; 1301h - display string and update cursor
int    0x10          ; Call the Video ServicES BIOS Interrupt
```

And here is the result of running this from a floppy:

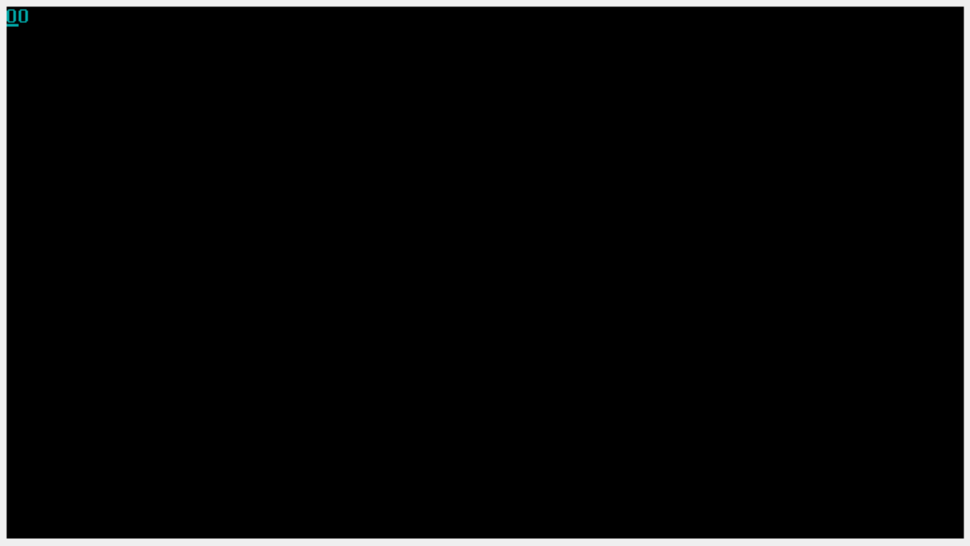


```
A
 BBB
  C
   0
    0
     Hello, World!
      Hello, World!
```

The following assembly code doesn’t use INT 10 but writes the characters directly to the video memory:

```
org 7c00h

section .text
   global _start

_start:
   mov     AX, 0xB800        ; Pointer to Video memory
   mov     ES, AX            ; Equal es to ax to video memory
   xor     DI, DI            ; Offset (B800:0000) - offset to write
                             ; characters to video memory pointer

   mov     AX, 'O'           ; Character to print
   stosb                     ; Write the character to the memory
   mov     AX, 0x3           ; Text color (cyan)
   stosb                     ; Write the attribute to the memory

   mov     AX, 'O'           ; ...
   stosb                     ; ...
   mov     AX, 0x3           ; ...
   stosb                     ; ...
```

And we get:



```
OO
```

Conclusion:

The lab work report concluded by examining the flexible BIOS text print options via direct video memory writing and INT 10 Interruption. The study illustrated the importance of these methods in low-level programming, offering insightful information about effective text rendering on the screen and deepening my comprehension of operating system internals. This information is crucial for programmers and developers who want to maximize text display in a variety of applications, demonstrating the continuing importance of these fundamental ideas in computer science. I came to the following conclusions for myself: The first approach, known as TTY, is the easiest to use and allows for cursor advancement, so I suppose it's ideal for producing characters directly from the keyboard; the second is a little slower, but it supports multiple similar characters output, which makes it appropriate for line drawing, I suppose; the third is a bit more advanced in terms of text attributes; the fifth and fourth have greater position and attribute capabilities than the previous and support multiple different characters display, but they are much more complex than the previous methods; the last two can handle strings, which makes them great for long text display; and the last one is the fastest since it involves access of the video graphic array directly used by the video services… right? :)