**Ministerul Educaţiei și Cercetării al Republicii Moldova Universitatea Tehnică a Moldovei**

**Facultatea Calculatoare, Informatică și Microelectronică**

**Operating Systems**

**Laboratory Work 1:**

INT 10H Video Services

Elaborated:

st. gr. FAF-212 Grama Alexandru

Verified:

asist. univ. Rostislav Calin

Chişinău - 2023

**Objective**

Implement these methods and make a bootable image:

- Method 1: Write character as TTY

- Method 2: Write character

- Method 3: Write character/attribute

- Method 4: Display character + attribute

- Method 5: Display character + attribute & update cursor

- Method 6: Display string

- Method 7: Display string & update cursor

**Implementation**

**Method 1**

To write a character as TTY, we use AH=0Eh. Here is how we print the “h” character:

Method\_a:

mov AH, 0Eh

mov AL, 'h'

int 10h

ret

**Method 2**

To write a character we use AH=0aH. Here is how we print the “e” character:

Method\_b:

mov AH, 0aH

mov AL, 'e'

int 10h

ret

**Method 3**

To write a character we use AH=09h. Here is how we print the “l” character:

Method\_c:

mov AH, 09h

mov AL, 'l'

mov BH, 0

mov BL, 0x07

mov CX, 1

int 10h

ret

**Method 4**

To display a character and an attribute we use AH=13h. Here is how we print the “l” character:

Method\_d:

mov AH, 13h

mov AL, 2

mov BH, 0

mov BL, 0x1E

mov DH, 4

mov DL, 0

mov CX, 1

lea BP, [Msg\_d]

int 10h

ret

**Method 5**

To display a character and an attribute and to update cursor we use AH=13h. Here is how we print the “o” character:

Method\_e:

mov AH, 13h

mov AL, 3

mov BH, 0

mov BL, 0x2E

mov DH, 5

mov DL, 0

mov CX, 1

lea BP, [Msg\_e]

int 10h

ret

**Method 6**

To display a string we use AH=13h. Here is how we print the “nice” string:

**Method\_str1:**

**mov AH, 13h**

**mov AL, 1**

**mov BH, 0**

**mov BL, 0x07**

**mov DH, 8**

**mov DL, 0**

**lea BP, Msg\_str1**

**mov CX, 8**

**int 10h**

**Method 7**

To display a string and update cursor we use AH=13h. Here is how we print the “race” string:

**Method\_str2:**

**mov AH, 13h**

**mov AL, 1**

**mov BH, 0**

**mov BL, 0x07**

**mov DH, 10**

**mov DL, 0**

**lea BP, Msg\_str2**

**mov CX, 8**

**int 10h**

**ret**

**Method 8**

To display a string directly to video memory. Here is how we print the “hello” string:

M8:

; Set ES to video segment

mov ax, 0xB800

mov es, ax

mov di, 1920

; Print 'h'

mov ax, 0x0720 | 'h' ; AH = attribute (grey on black), AL = 'h'

mov es:[di], ax

add di, 2

; Print 'e'

mov ax, 0x0720 | 'e'

mov es:[di], ax

add di, 2

; Print 'l'

mov ax, 0x0720 | 'l'

mov es:[di], ax

add di, 2

; Print 'l'

mov ax, 0x0720 | 'l'

mov es:[di], ax

add di, 2

; Print 'o'

mov ax, 0x0720 | 'o'

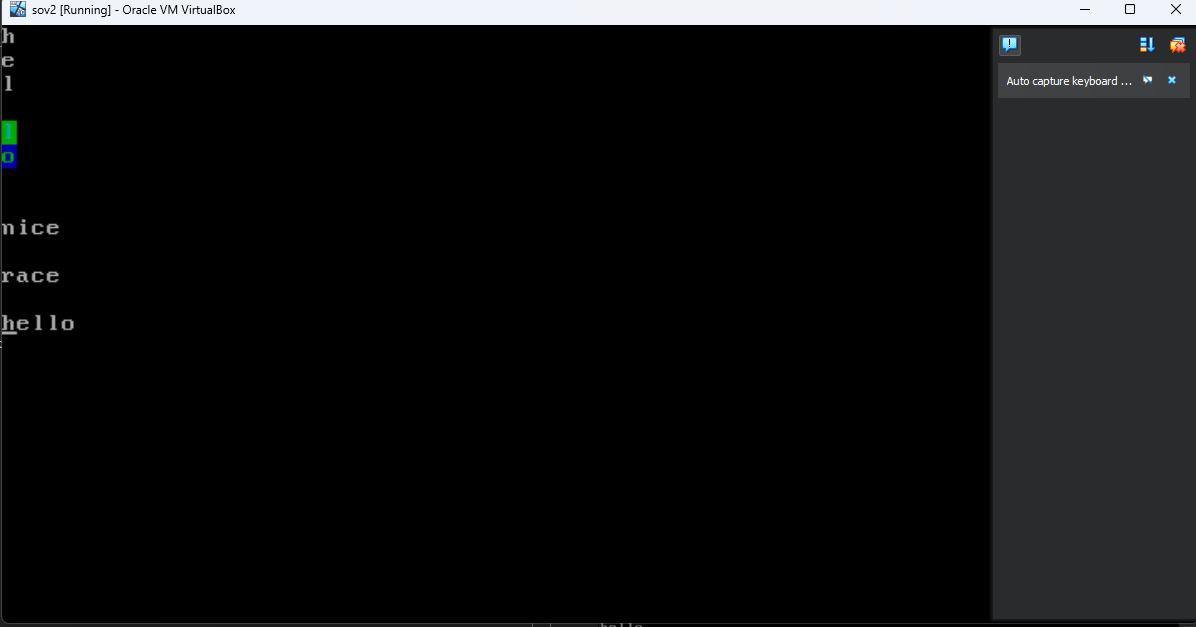
mov es:[di], ax

ret

**Results**

When booting, we get the following output:

Figure 1 – *Booted image*

****

**Conclusions**

The assembly code in discussion offers a fascinating glimpse into the world of low-level programming, particularly in how characters and strings are displayed on a screen. At the very beginning, the code sets up the necessary environment and delves straight into calling a series of methods to display characters. These methods predominantly leverage BIOS interrupts, which are essentially standard routines housed within a computer's firmware, tailored for tasks like displaying characters on a monitor.

However, the true essence of low-level control manifests in the M8 method. Instead of relying on the conventional BIOS routines, this method writes the word "hello" directly to the screen's memory. This approach, termed direct memory access, is heralded for its efficiency and speed. It provides programmers a more intimate control over the hardware, allowing for nuanced manipulations that might be cumbersome or impossible with higher-level methods.

The precision doesn't end at just writing to the memory. The position where "hello" appears on the screen is not left to chance. It's meticulously calculated using the desired row and column, translating those into a memory address. This calculation showcases the meticulous attention to detail required in assembly programming, where every byte and address counts.

Concluding, this assembly program is more than just a set of instructions. It's a tutorial, a journey that transitions from using conventional BIOS routines to the raw power of direct memory writing. Through this progression, one can appreciate the layers of abstraction that exist in modern computing and the sheer control and potential that assembly language provides to those who dare to delve deep.