

Ministerul Educației și Cercetării al Republicii Moldova
Universitatea Tehnică a Moldovei
Facultatea Calculatoare, Informatică și Microelectronică

Laboratory work 1: Operational Systems

Elaborated:
st. gr. FAF-213

Afteni Maria

Verified:

Rastislav Călin

Chișinău - 2023

ALGORITHM ANALYSIS

Objective: Print Text

- Tasks:**
- 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
 - M7: Display string & update cursor
 - M8(optional): Print directly to video memory

IMPLEMENTATION

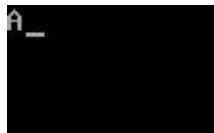
Method 1

Method1: Write character as TTY

```
go:
    mov ah, 0Eh
    mov al, 41h ; Load the ASCII code for 'A'
    int 10h
```

I set up the AH register to prepare for console output. Then, I loaded the value 'A' by placing its ASCII code, which is 41h, into the AL register. Finally, I triggered interrupt 10h to display the character represented by the value in the AL register on the screen.

Output:



Method 2

Method 2: Write character

```
go:
    mov ah, 0x0A ; Set AH to 0x0A to specify the "Write Character" function
    mov al, 'A'
    mov bh, 0x00
    mov cx, 0x01
    int 0x10
```

I first set the AH register to 0x0A, to use the "Write Character" function. Next, I loaded the character 'A' into the AL register. Additionally, I set BH to 0x00 to define the display page, and CX to 0x01 to specify the number of times the character 'A' should be displayed on the screen. Finally, I used the "int 0x10" instruction to display 'A' on the screen.

Output:



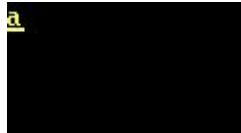
Method 3

M3: Write character/attribute

```
go:
    mov ah, 09h
    mov al, 'a'
    mov bh, 0
    mov bl, 0Eh
    mov cx, 01h
    int 10H
```

Firstly, I loaded 09h into the ah register to indicate a print character function. Then, I placed the character 'a' into the al register, which represents the data to be displayed. To control the display color, I assigned 0 to bh and 0Eh to bl in the bh and bl registers, respectively. After setting up the display parameters, I specified that I wanted to print 1 character by storing 01h in the cx register. Finally, I invoked the video services interrupt 10H to display the character 'a' with the specified attributes.

Output:



Method 4

```
go:
    mov ah, 09h
    mov al, 'M'
    mov bl, 13H    ; Attribute/Color
    mov cx, 8      ; Number of characters to write
    int 10h
```

First, I load 09h into the AH register, which indicates that I want to use the "Write Character" operation. Next, I assign the character 'M' to the AL register to display it later. Additionally, I set the BL register to 13H, determining the attribute of the text, and I specify that I intend to display 8 characters on the screen using the CX register. Finally, I execute the video operation by triggering interrupt 10h.

Output:



Method 5

```
go:
    mov ah, 09h    ; Function 09h = Write Character and Attribute at Cursor
Position
    mov al, 'M'
    mov bl, 13H    ; Attribute/Color
    mov cx, 8      ; Number of characters to write
    int 10h
```

```

mov ah, 02h    ; Function 02h = Set Cursor Position
mov bh, 0      ; Page number
mov dh, 0      ; Row
mov dl, 8      ; Column
int 10h

```

In the first register configuration, I'm preparing to write a character 'M' with a designated attribute or color (13H) at the current cursor position. Additionally, I'm specifying that I want to display a total of 8 characters using function 09h, which allows me to write text with specific attributes.

In the second register setup, I'm using a different function (02h) to precisely position the cursor on the screen. I'm setting the page number (0), row (0), and column (8) to determine where the cursor will be placed. This cursor positioning will influence the location where the next character or text will be displayed on the screen. These instructions collectively enable me to control what appears on the screen and precisely where it's displayed.

Output:



Method 6

M6: Display string

```

org 7c00H

mov bh, 0
mov ax, 0H
mov es, ax
mov bp, msg

mov bl, 0fH
mov al, 1
mov cx, 12
mov dh, 1
mov dl, dh
mov ax, 1300H

int 10H

jmp $

msg dd "Hello World!"

```

At the start of the program, I set the program's origin to memory address 7C00H using "org 7c00H". Next, I initialize several registers, including ES, BP, BL, AL, CX, DH, and DL, which are essential for various data manipulation and display operations. Towards the end of the code, I use "int 10H" to trigger an interrupt, followed by "jmp \$" to create an endless loop, effectively pausing program execution. The data "Hello World!" is stored in memory at the label "msg."

Output:

A screenshot of a terminal window with a black background. The text "Hello World!" is displayed in a light blue monospaced font. A small white cursor is positioned at the end of the line, after the exclamation mark.

Method 7

M7: Display string & update cursor

```
org 0x7c00

mov ax, 1300h
mov al, 1
mov bl, 13H
mov cx, msg_len
mov dh, 0x01
mov dl, 0x00
mov bp, msg

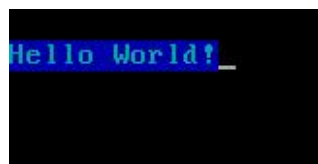
int 10H

jmp $

msg db "Hello World!"
msg_len equ $-msg
```

I set the initial program execution point at memory address 0x7c00. I load values into various registers for display and messaging purposes, including setting up the message "Hello World!" to be shown. Finally, I display the message on the screen and enter an infinite loop with the "jmp \$" instruction.

Output:

A screenshot of a terminal window with a black background. The text "Hello World!" is displayed in a light blue monospaced font. A small white cursor is positioned at the end of the line, after the exclamation mark.

CONCLUSIONS

In conclusion, this laboratory work has been an insightful exploration into the low-level programming of text and character display in assembly language. I successfully implemented various methods to interact with the display, ranging from basic character output to more advanced techniques that involve attributes and cursor manipulation.

Through this practical experience, I have not only deepened my understanding of assembly language but also learned the essential techniques required to interact with text and character-based displays, which can be invaluable for creating custom user interfaces, console applications, and even low-level graphics.