

Ministry of Education of Republic of Moldova
Technical University of Moldova
Department of Software engineering and Automatics

Report

Laboratory Work Nr.1

on SOMIPP

Performed by

Zaharia Gabriel

Checked by

Rostislav Calin

Chisinau,2018

Objectives:

- BIOS interrupts.
- How to use interrupts for printing some characters , string.
- Printing to the Video Memory.
- Purpose and working of registers.
- Understand the notion of loader and kernel.
- How memory cells are distributed and how to use them for printing.

Tasks:

- Using BIOS interrupts implement 8 ways of printing.
- ASCII characters.
- Printing the content of the registers.

Implementation of tasks

Task 1:

The main goal of this task is to print some characters using different BIOS interrupts.

What we are doing first? First we should inform about interrupts and how it works. Interrupts are the signal to the processor emitted by the hardware or software indicating an event that need a immediate attention.

From experience in assembly and operating with Virtual Box, we noticed that using the int 21h interrupt is restricted , because our implementation should be functional on another machine and here we can use for this task only the following interrupts:

- Int 10h AH 09h
- Int 10h AH 0Ah
- Int 10h AH 0Eh
- Int 10h AH 13h AL 00h
- Int 10h AH 13h AL 01h
- Int 10h AH 13h AL 02h
- Int 10h AH 13h AL 03h
- Video memory printing

So , let's analyze this all methods and see how they work in process of printing something.

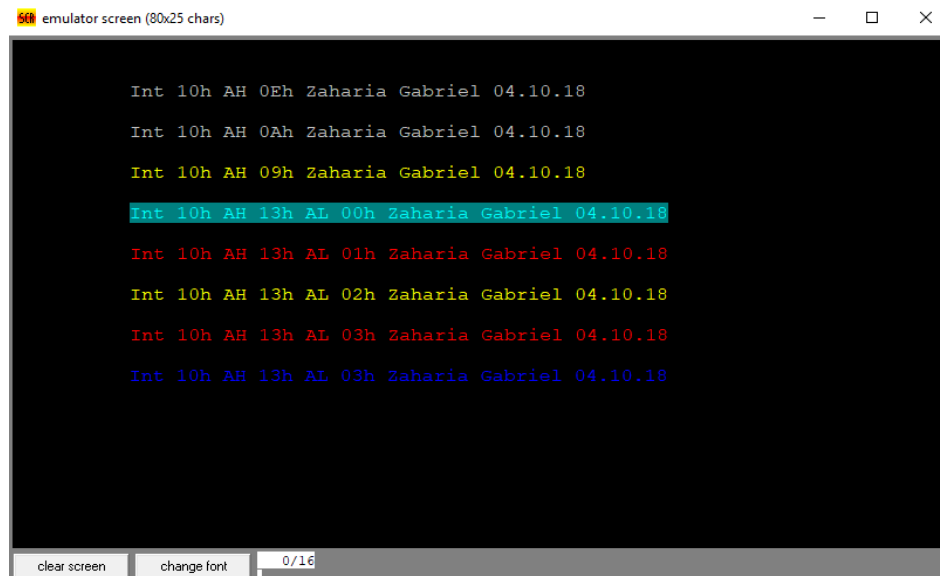
For methods with int 10h(09,0A,0E) for printing something on the screen I should to print character by character according to the documentation. So , for performing this way of printing I have used SI register for setting the start index of string, CX for store the total number of characters which should be displayed and a loop which I iterate through my characters that are stored in my str1 variable.

For next 4 methods of printing I also used the CX register for store the how many times it should print the character. Also , on this method we can use offset of string which is very useful. Another features of this is to using the attributes for characters , like: colors, backgrounds for characters which are set in the BL register.

Printing directly to video memory doesn't need some interrupts because here we work direct with another place to printing(0b800h) , this means the next cell that is free , without something inside.

The result of whole implementation we can see in the Fig.1.

Implementation of Task 1



```
sf emulator screen (80x25 chars)

Int 10h AH 0Eh Zaharia Gabriel 04.10.18
Int 10h AH 0Ah Zaharia Gabriel 04.10.18
Int 10h AH 09h Zaharia Gabriel 04.10.18
Int 10h AH 13h AL 00h Zaharia Gabriel 04.10.18
Int 10h AH 13h AL 01h Zaharia Gabriel 04.10.18
Int 10h AH 13h AL 02h Zaharia Gabriel 04.10.18
Int 10h AH 13h AL 03h Zaharia Gabriel 04.10.18
Int 10h AH 13h AL 03h Zaharia Gabriel 04.10.18

clear screen  change font  0/16
```

Fig.1: Result of implementation interrupts .

Task 2:

In this task we have to print the ASCII table. So for performing this task we also need some interrupts to print the characters on the screen. The method of printing that I chose to use allow me to print the characters without storing them somewhere , for this I used the AL register and to pass the characters I performed the increment of this till the last characters from ASCII code. Also , I had some comparison operation in my code , this I have done for not printable characters which are 8. Hexadecimal code for this characters are : 00h, 07h, 08h, 09h, 0Ah, 0Dh, 20h, FFh. To likes more structured I chose to print the table in columns.

Another possibility to print the table is in string from , one by one character, where it is enough only one small loop.

ASCII Table

emulator screen (80x25 chars)

clear screen change font 0/16

Fig.2: ASCII Table

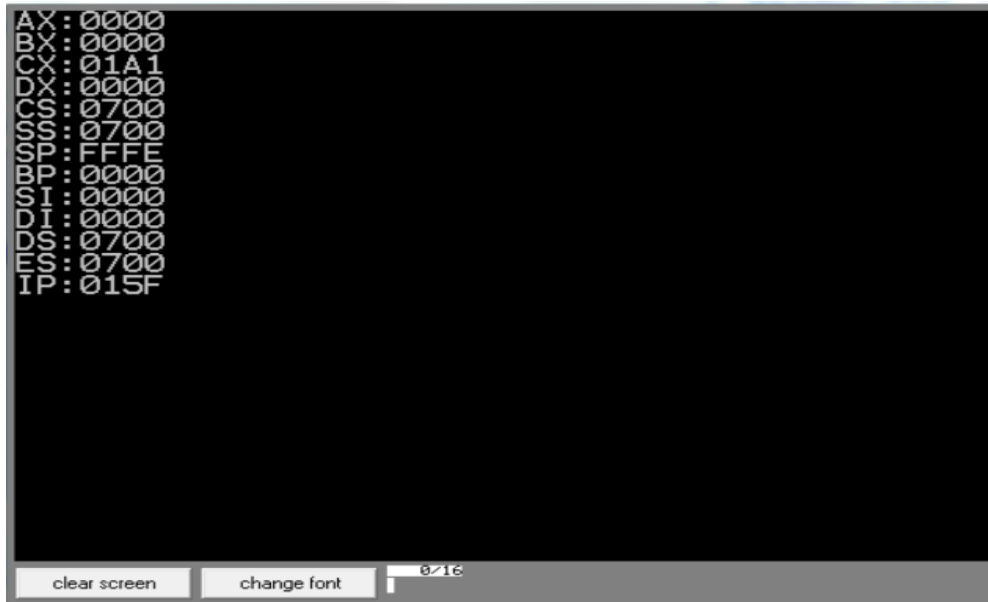
Task 3:

What is meaning of printing the content of the registers ? This action means to display on the screen values that are initially introduced or exists in the registers.

So , to implement this actions in assembly languages you should be informed very good about interrupts , procedures , ptr, shifting and correction. According to the knowledge's from the previous class and ability from current this task should be implemented in the following way:

- Dividing the content of registers in two parts and here I want to mention that this is very easy to do for the AX,BX,CX,DX registers , for next is not so easy.
- store the content in the variable of DB type.
- To divide the next registers should be used the byte ptr, it divides the word in 2 bytes , which automatically doing in the AX register.
- Operating with procedure for storing the content of IP register. I tried to get the value of IP in C , but it is not possible because it has not the connection of CPU registers.
- Procedure is working with push and pop options , here push and pop works based on LIFO method. This here is doing for storing in the stack and getting from this the value of IP register.
- Having the all of registers divided by parts and stored in the their own way we can print them.

Printing of registers content



```
AX: 0000
BX: 0000
CX: 01A1
DX: 0000
CS: 0700
SS: 0700
SP: FFFE
BP: 0000
SI: 0000
DI: 0000
DS: 0700
ES: 0700
IP: 015F
```

Fig.3: Registers content

Observations:

- This laboratory work was done in small students group, for increasing the productivity and speed of solving the tasks.
- Using of different cell of memory can crush our program.
- DOSBOX is a great tool for informing about assembly language.
- Explanation of paper how work the loader really help to understand and how it load another free place to execute our program.

Conclusion:

During this laboratory I obtained skills operating with BIOS interrupts and what is the difference between them for printing something in our case. Some of them can do not only printing , but getting some additional attributes , like: color, background, set the cursor position and update.

Another great experience in this laboratory work was working with registers and try to print their content. This took hours of understanding and implementing. Here , I also obtain skills of dividing the rest of registers in parts and store their content. The same as in C language , working with ptr, which calling the procedure will push and pop its value. AX,BX,CX,DX registers according to the memory working language have some automatically process , dividing in two parts , for example for AX register : AH and AL.

Working with Virtual Box helped me in better understanding about file size and how bigger should be the compiled file from assembly to be executable on some machine from VM. To optimize this operation of creating another file of specific size till the 1474560bytes, I have created a script which do that things. This is the bash script and to execute this you should type : ./script.sh filename sizefile.