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

Report

Laboratory Work Nr.1
on SOMIPP

Performed by Metei Vasile
Supervised by Rostislav Calin

October 19, 2018

Goals:

- Get more familiar with BIOS interrupts;
- Get more familiar with interrupts that take care of printing;
- Direct printing to the Video Memory;
- Understand how to work with the registers;

Tasks:

- Implement 8 ways of printing using BIOS interrupts;
- Print the ASCII characters;
- Print the content of the registers;

Laboratory Work Implementation

Task 1:

In order to print some characters, I used some interrupts. Because this code should work on an empty machine only BIOS interrupts were used. I find out 7 methods which allow us to print characters or strings using BIOS interrupts. They are:

- 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

The last method of printing is not using any interruption, it is printing directly in video memory.

First 3 interrupts are printing just one character at a time that's why I used a loop in order to iterate through my strings and finally print them.

Next 4 interrupts allows us to print strings, the only thing you should keep in mind is that the number of characters in the string should be kept in CX register.

Some of those interrupts beside printing are doing some other staff too, for example: updating the cursor position or using some attributes that allow us to display characters/strings using different colors or backgrounds.

Last 2 interrupts and printing directly in video memory allow us to specify for every character its own color. First you have to specify the character that should be printed and next to it the color: 'V', '0Bh'.

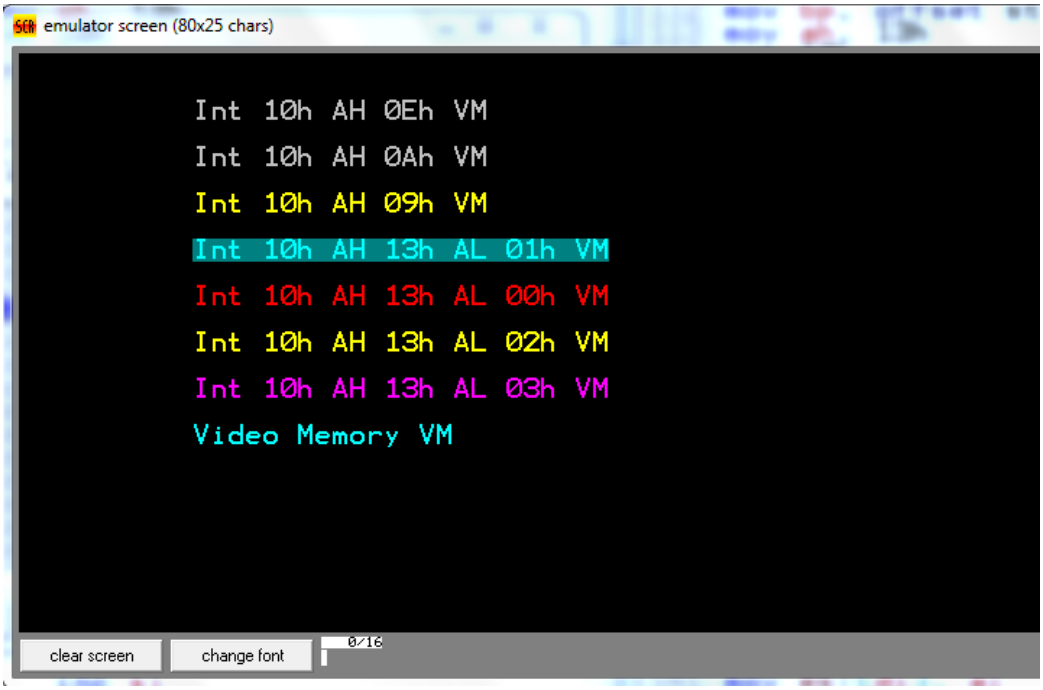

Figure 1: Task 1

Task 2:

This task is similar to the first one. It is also about the printing. The difference is that here I printed the ASCII characters. I was given the possibility to choose how I want to print them. So I selected Int 10h AH 0eh because it prints one by one character. And because I planned to print them in the columns it was important for me.

We all know that there are 256 ASCII characters and 8 of them are not printable. I remarked this point because it is an important feature of my program.

Because the first ASCII code start with 00h and goes through till FFh, the last character. I didn't even store them.

The interrupt I use is printing the character stored in AL register, that's why I wrote 00h in the AL. And then I kept incrementing AL till it reached the last character. It was performed in a loop.

As I mentioned above there are 8 characters which are not printable. Their codes are:00h, 07h, 08h, 09h, 0Ah, 0Dh, 20h, FFh. That's why before printing I checked what I have in AL. If there is a code that belongs to one of those 8 characters, it will not print it and will go to the next character.

Next functionality implemented by me for this task was printing them in the columns. This will help me to display them in a nicer way on the screen.

In order to do this I check what is the position of the cursor. If it reaches a certain value on the y-axis I move it on the next column.

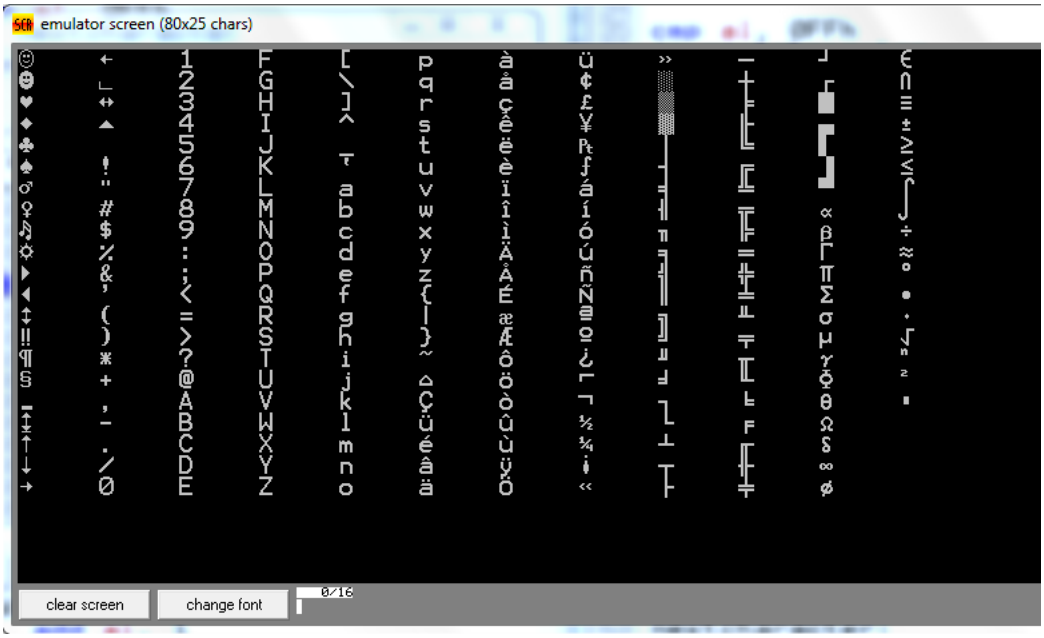

Figure 2: Task 2

Task 3:

This task beside the printing was aimed to introduce us in working with registers. Is not that simple to print register's content as it can appear at the first look. Just copying its content to variable an then printing it will not work. In such a way it will print the characters that belong to those codes, but not the codes themselves.

First of all I stored all the contents of the registers in a variable called str of DB type. Before storing I divided the value of the register in 2 parts: high and low and then wrote them in the variable mentioned above. The division was simple for registers AX, BX, CX, DX. The rest of the registers were divided by using byte ptr. It helps us to divide a word in 2 bytes. Also is important to mention that in order to store the content of IP register you have to call a procedure, otherwise you will not be able to store its value. While calling the procedure the value of IP is pushed in stack and after you just pop it.

After I stored every register as 2 parts. I started to divide each part in 2 parts again. It was performed by a shift right operation and a logical "and" operation. Now I have access to every single part of the value stored

in register. Next I add 30h that corresponds to '0' to every character. If the sum was larger than 39h that correspond to '9' it performs a correction which adds 7 to that value. Why 7? Because there are 7 characters between '9' and 'A'. Now I am ready to print it. After finishing to print an entire register it goes to the next line and move the cursor at the start of that line. So I print all the registers in different rows.

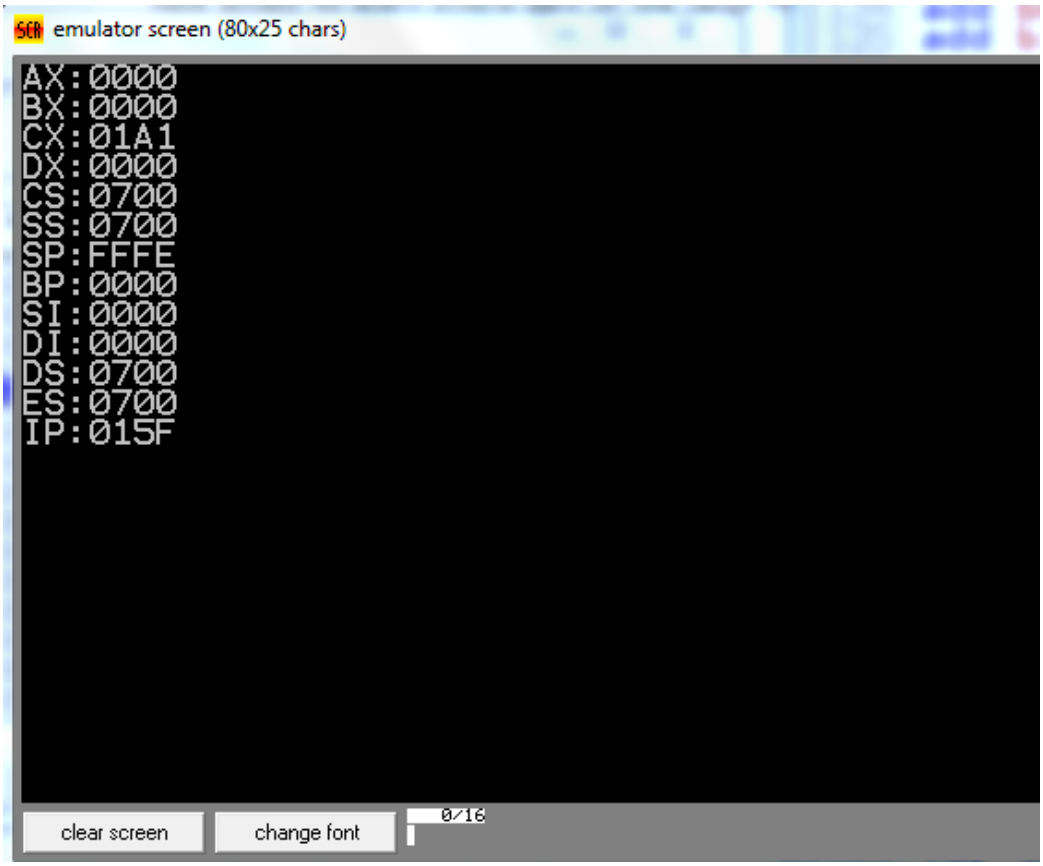

Figure 3: Task 3

Conclusion:

In this laboratory work I discovered what are the BIOS interrupts. I find out which of those interrupts can be used while printing or doing staff related to printing, moving the cursor for example.

Because this code should run on an empty machine I indicated the specific address at the beginning of the each file. So it will now the address from were to load the variables. It is important. Otherwise it will print wrong characters.

I learned how to work with registers, how to store their values and print them. It was way easier to work AX, BX, CX and DX registers because they are automatically divided in high and low parts. The rest of them should be divided using "byte ptr", that help to divide a word in 2 bytes. IP register behaves differently from others because it contains the pointer to the next instruction to be performed. Its values cannot be just stored. In order to store it I call a procedure. This will push its value to the stack automatically and then I pop it.