Microprocessor-Based Systems - Lab Sessions.

P0: 80x86 Development and debugging environment Exercises

Basic structure of programs in Assembly language 80x86

Open the “factor.asm” file in a text editor. The first remark is the definition of the different segments of the program (data, stack, extra and code). The order in which the definition of the different segments appear within the program source is indifferent. Not all segments are always present in all programs. For example, the extra segment may not be defined in some programs.

At the beginning of the code segment, you must use the ASSUME directive to associate the segments name with the segment to be used when accessing to the memory addresses in each segment. However, this is only a syntactic helper, and this directive does not load any value on the segment registers (DS, SS, ES), which should be assigned programmatically. The code to load these values should be at first in any program.

```
MOV AX, DATOS

MOV DS, AX

MOV AX, PILA

MOV SS, AX

MOV AX, EXTRA

MOV ES, AX
```

Instructions can only be present in the code segment. However, the data can be defined in any segment. The default state is that the data are defined in the data/extra segments, but in this example we may observe a variable defined in the code segment. This is the reason why the program starts at address cs:0002 and not at cs:0000 (that is the default start).

To finish the program execution, it is used the 21H Interruption in conjunction with AX=4C00h. This combination takes the control back to the operating system.

In the last line of the file the ‘END’ directive shall be included and followed by the name of the procedure where the program execution should be started (entry) (in our example, it is the START procedure). When starting TD, the arrow marker will point right at the beginning of this procedure, and the IP register will be loaded with the corresponding memory address.

Variable visualization

Quite often while debugging the code with the "td", it may be necessary to see the content of the variables used in our program that are stored in memory. For this it is necessary to modify slightly the makefile we created previously by replacing the last line

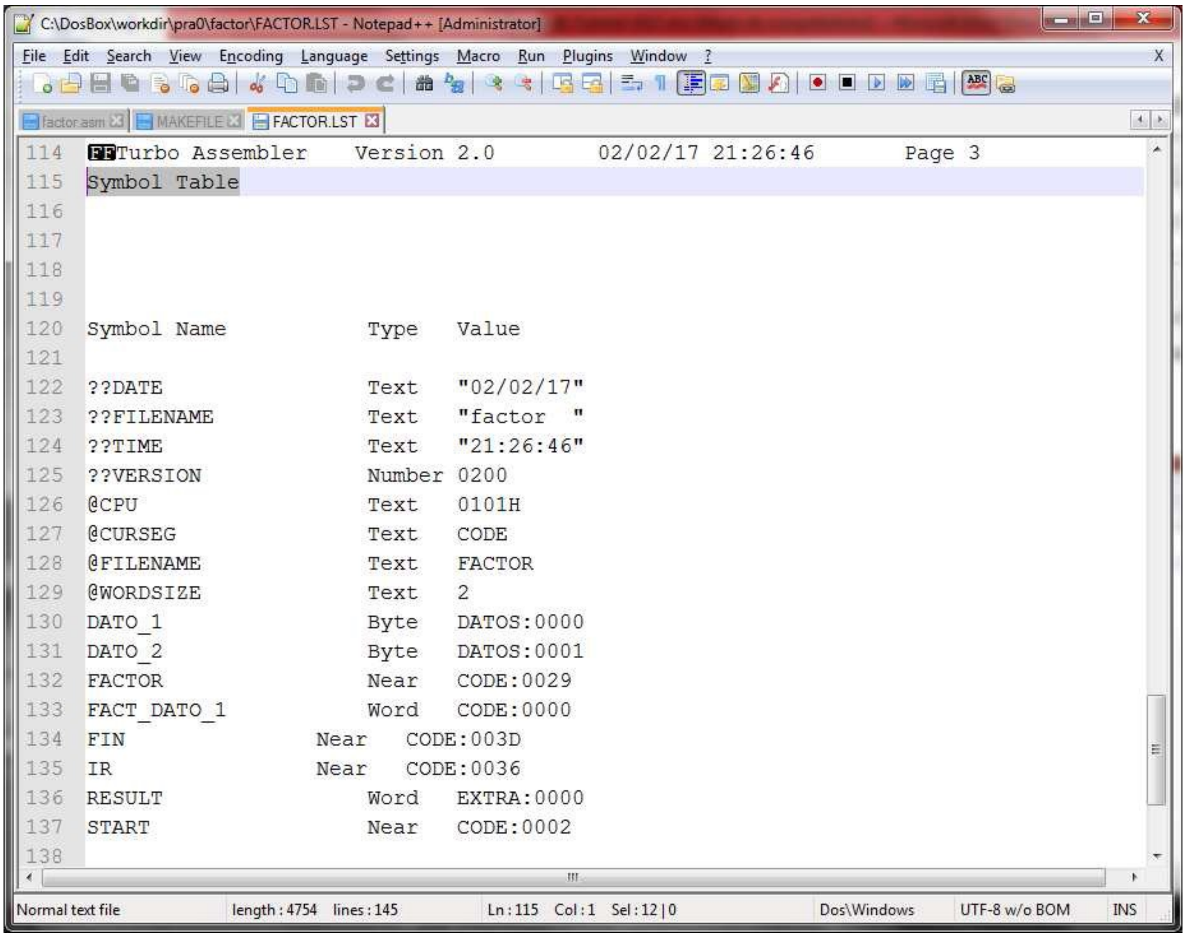
```
tasm /zi factor.asm
```

by

```
tasm / zi factor.asm ,, factor.lst
```

If we execute "make" after these changes (deleting the factor.obj and factor.exe files previously), we can observe by typing "dir" that an additional file has been created, called factor.lst. In this new file we can find first the correspondence between the Assembly instructions of our program and the instructions in machine code.

However, to see the content of the program variables we have to look towards the end of the file factor.lst since it contains the symbol table used in the program.



C:\DosBox\workdir\pra0\factor\FACTOR.LST - Notepad++ [Administrator]
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
factor.asm
MAKEFILE
FACTOR.LST
Normal text file
length : 4754 lines : 145
Ln : 115 Col : 1 Sel : 12 | 0
Dos\Windows
UTF-8 w/o BOM
INS

```
FFTurbo Assembler    Version 2.0        02/02/17 21:26:46        Page 3
Symbol Table




Symbol Name                 Type   Value

??DATE                      Text   "02/02/17"
??FILENAME                  Text   "factor  "
??TIME                      Text   "21:26:46"
??VERSION                   Number 0200
@CPU                        Text   0101H
@CURSEG                     Text   CODE
@FILENAME                   Text   FACTOR
@WORDSIZE                   Text   2
DATO_1                      Byte   DATOS:0000
DATO_2                      Byte   DATOS:0001
FACTOR                      Near   CODE:0029
FACT_DATO_1                 Word   CODE:0000
FIN                     Near   CODE:003D
IR                      Near   CODE:0036
RESULT                      Word   EXTRA:0000
START                       Near   CODE:0002

```

The logical direction of any variables is decomposed in two pieces of information: a segment and an offset.

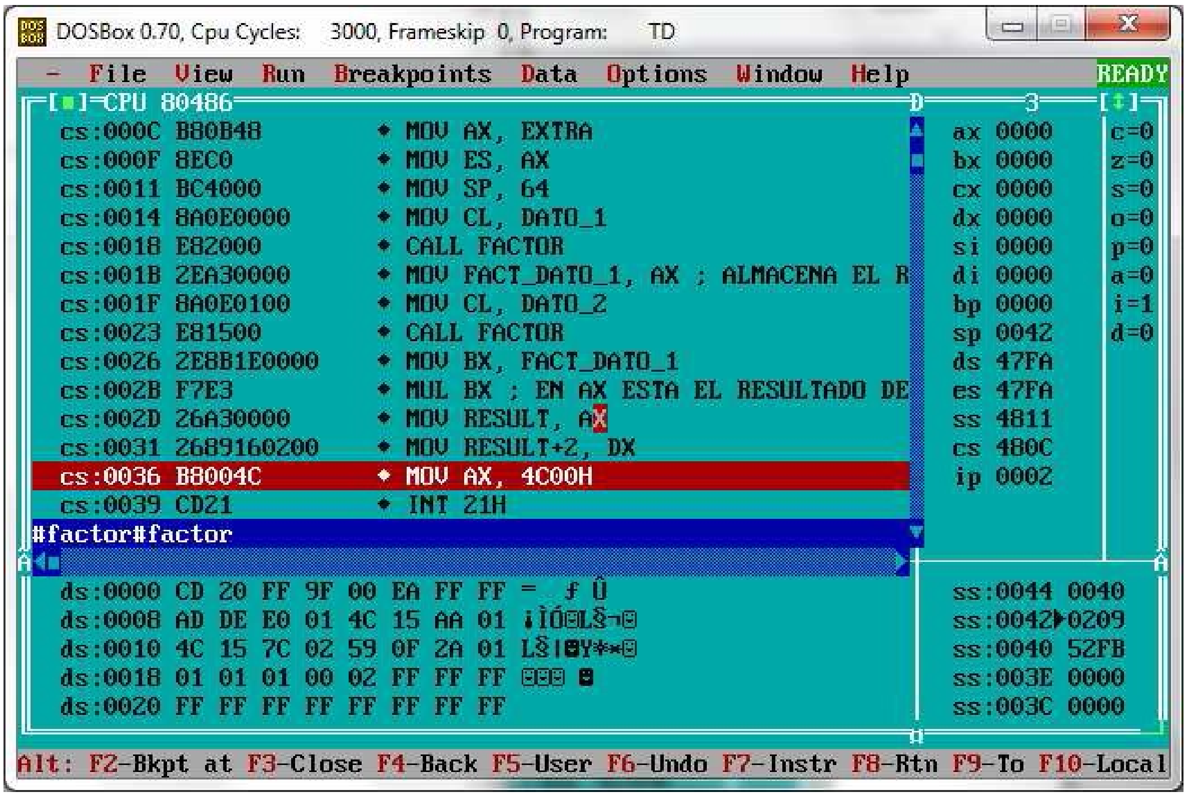
For example:

```
DATO_1: DATOS segment, offset 0

DATO_2: DATOS segment, offset 1

FACT_DATO_1: CODE segment, offset 0

RESULT: EXTRA segment, offset 0
```

let's run our program again on the "td". This time, instead of running step by step with F7 key, we will place a breakpoint at the instruction

```
MOV AX, 4C00H
```

To do this, select the instruction with the cursor and the press F2. Move the cursor upwards to check that the breakpoint has been activated, (indicated with a red line)

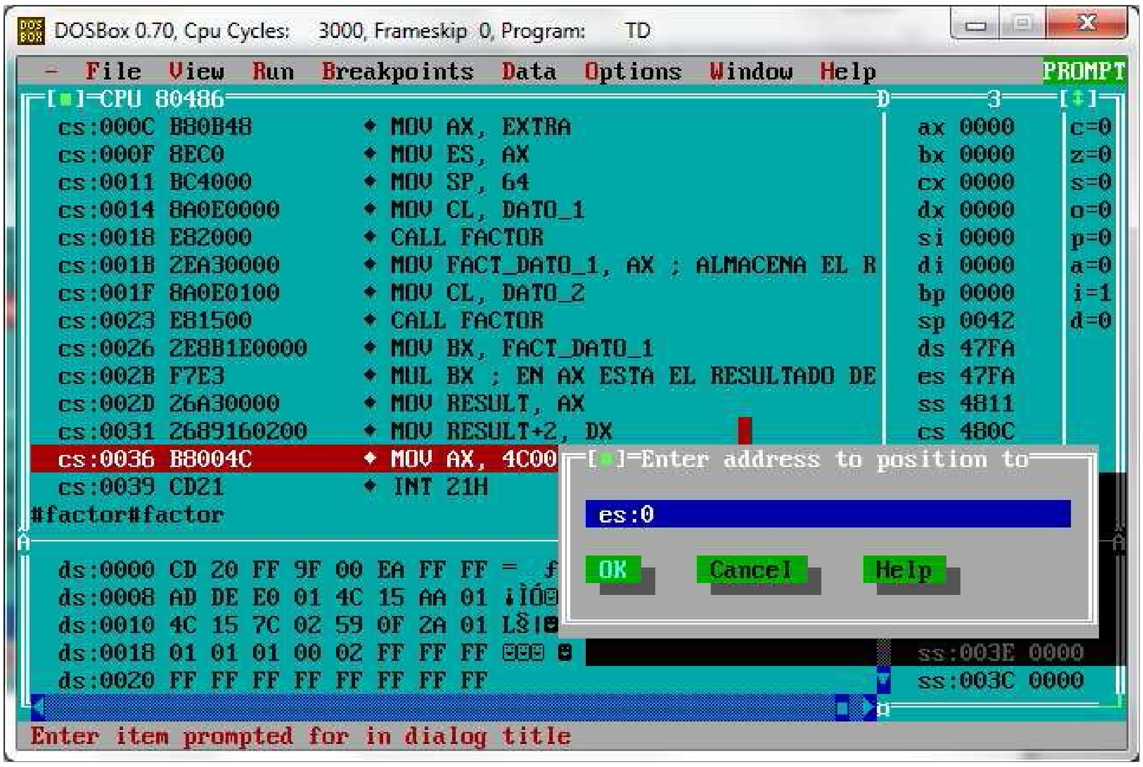


```
DOSBox 0.70, Cpu Cycles:    3000, Frameskip  0, Program:     TD
 ≡  File  View  Run  Breakpoints  Data  Options  Window  Help                 READY
[■]=CPU 80486
  cs:000C B80B48        ♦ MOV AX, EXTRA                              ax 0000   c=0
  cs:000F 8EC0          ♦ MOV ES, AX                                 bx 0000   z=0
  cs:0011 BC4000        ♦ MOV SP, 64                                 cx 0000   s=0
  cs:0014 8A0E0000      ♦ MOV CL, DATO_1                             dx 0000   o=0
  cs:0018 E82000        ♦ CALL FACTOR                                si 0000   p=0
  cs:001B 2EA30000      ♦ MOV FACT_DATO_1, AX ; ALMACENA EL R        di 0000   a=0
  cs:001F 8A0E0100      ♦ MOV CL, DATO_2                             bp 0000   i=1
  cs:0023 E81500        ♦ CALL FACTOR                                sp 0042   d=0
  cs:0026 2E8B1E0000    ♦ MOV BX, FACT_DATO_1                        ds 47FA
  cs:002B F7E3          ♦ MUL BX ; EN AX ESTA EL RESULTADO DE        es 47FA
  cs:002D 26A30000      ♦ MOV RESULT, AX                             ss 4811
  cs:0031 2689160200    ♦ MOV RESULT+2, DX                           cs 480C
  cs:0036 B8004C        ♦ MOV AX, 4C00H                              ip 0002
  cs:0039 CD21          ♦ INT 21H
#factor#factor

  ds:0000 CD 20 FF 9F 00 EA FF FF =  ƒ Û                             ss:0044  0040
  ds:0008 AD DE E0 01 4C 15 AA 01 ¡ÌÓ☺L§¬☺                           ss:0042►0209
  ds:0010 4C 15 7C 02 59 0F 2A 01 L§|☻Y☼*☺                           ss:0040  52FB
  ds:0018 01 01 01 00 02 FF FF FF ☺☺☺ ☻                              ss:003E  0000
  ds:0020 FF FF FF FF FF FF FF FF                                    ss:003C  0000
Alt: F2-Bkpt at F3-Close F4-Back F5-User F6-Undo F7-Instr F8-Rtn F9-To F10-Local
```

Then execute the program pressing F9 key. The program will then be executed and stopped at the breakpoint. In this moment, the RESULT variable should contain its final value.

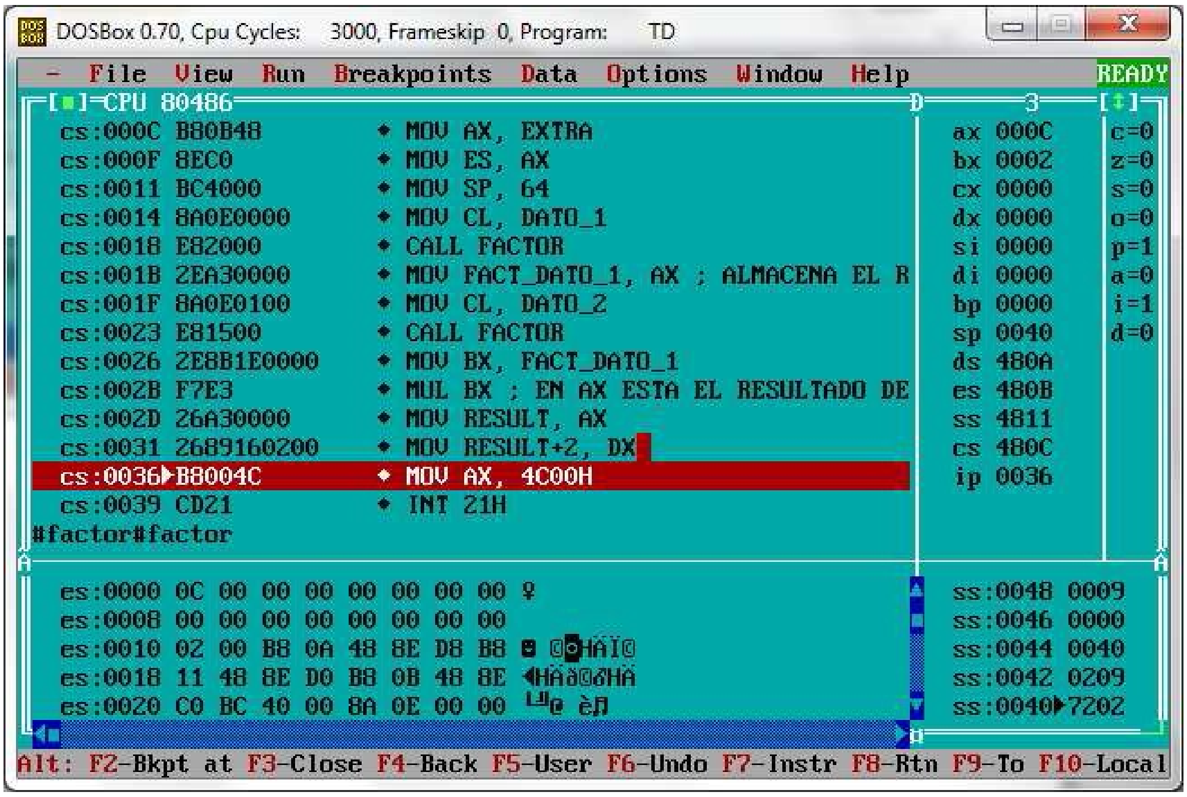
This value can be found in the data window, using "Goto ES: 0".

Remember that EXTRA segment is associated to to the ES register through the directive ASSUME.


DOSBox 0.70, Cpu Cycles: 3000, Frameskip 0, Program: TD
- File View Run Breakpoints Data Options Window Help PROMPT
Enter address to position to
OK
Cancel
Help

```
[■]=CPU 80486
  cs:000C B80B48        ♦ MOV AX, EXTRA                          ax 0000   c=0
  cs:000F 8EC0          ♦ MOV ES, AX                             bx 0000   z=0
  cs:0011 BC4000        ♦ MOV SP, 64                             cx 0000   s=0
  cs:0014 8A0E0000      ♦ MOV CL, DATO_1                         dx 0000   o=0
  cs:0018 E82000        ♦ CALL FACTOR                            si 0000   p=0
  cs:001B 2EA30000      ♦ MOV FACT_DATO_1, AX ; ALMACENA EL R    di 0000   a=0
  cs:001F 8A0E0100      ♦ MOV CL, DATO_2                         bp 0000   i=1
  cs:0023 E81500        ♦ CALL FACTOR                            sp 0042   d=0
  cs:0026 2E8B1E0000    ♦ MOV BX, FACT_DATO_1                    ds 47FA
  cs:002B F7E3          ♦ MUL BX ; EN AX ESTA EL RESULTADO DE    es 47FA
  cs:002D 26A30000      ♦ MOV RESULT, AX                         ss 4811
  cs:0031 2689160200    ♦ MOV RESULT+2, DX                       cs 480C
  cs:0036 B8004C        ♦ MOV AX, 4C00
  cs:0039 CD21          ♦ INT 21H
#factor#factor

  ds:0000 CD 20 FF 9F 00 EA FF FF = f
  ds:0008 AD DE E0 01 4C 15 AA 01 ¡ÌÓ
  ds:0010 4C 15 7C 02 59 0F 2A 01 L§|
  ds:0018 01 01 01 00 02 FF FF FF                        ss:003E 0000
  ds:0020 FF FF FF FF FF FF FF FF                        ss:003C 0000

es:0

Enter item prompted for in dialog title
```

Next images show the window data with the value 0C, that is, the value 12 in decimal (= 2! X 3!).


DOSBox 0.70, Cpu Cycles: 3000, Frameskip 0, Program: TD
- File View Run Breakpoints Data Options Window Help READY

```
[■]=CPU 80486
  cs:000C B80B48        ♦ MOV AX, EXTRA                          ax 000C   c=0
  cs:000F 8EC0          ♦ MOV ES, AX                             bx 0002   z=0
  cs:0011 BC4000        ♦ MOV SP, 64                             cx 0000   s=0
  cs:0014 8A0E0000      ♦ MOV CL, DATO_1                         dx 0000   o=0
  cs:0018 E82000        ♦ CALL FACTOR                            si 0000   p=1
  cs:001B 2EA30000      ♦ MOV FACT_DATO_1, AX ; ALMACENA EL R    di 0000   a=0
  cs:001F 8A0E0100      ♦ MOV CL, DATO_2                         bp 0000   i=1
  cs:0023 E81500        ♦ CALL FACTOR                            sp 0040   d=0
  cs:0026 2E8B1E0000    ♦ MOV BX, FACT_DATO_1                    ds 480A
  cs:002B F7E3          ♦ MUL BX ; EN AX ESTA EL RESULTADO DE    es 480B
  cs:002D 26A30000      ♦ MOV RESULT, AX                         ss 4811
  cs:0031 2689160200    ♦ MOV RESULT+2, DX                       cs 480C
  cs:0036▶B8004C        ♦ MOV AX, 4C00H                          ip 0036
  cs:0039 CD21          ♦ INT 21H
#factor#factor

  es:0000 0C 00 00 00 00 00 00 00 ♀                      ss:0048 0009
  es:0008 00 00 00 00 00 00 00 00                        ss:0046 0000
  es:0010 02 00 B8 0A 48 8E D8 B8  ©◘HÄÏ©                ss:0044 0040
  es:0018 11 48 8E D0 B8 0B 48 8E ◄HÄð©♂HÄ               ss:0042 0209
  es:0020 C0 BC 40 00 8A 0E 00 00 └╝@ è♫                 ss:0040▶7202

Alt: F2-Bkpt at F3-Close F4-Back F5-User F6-Undo F7-Instr F8-Rtn F9-To F10-Local
```

In the same way, we could inspect the rest of the variables of the program (DATO_1, DATA_2 and DATA_FACT_1), using "Goto DS: 0", "Goto DS: 1" and "Goto CS: 0" respectively.

Exercise 1: Factorial calculation

The program “factor.asm” has been designed to calculate the product of the factorial of a number by the factorial of another number. Make the appropriate modifications in the source code to calculate the following products with the help of TD. Record the results on a sheet in hexadecimal and decimal format, including a screenshot and a brief explanation of the changes and results obtained.

1. 4! x 5! =
2. 8! =
3. 9!=
4. 8! x 7! =

What is the value of FACT_DATO_1 on each case?

Exercise 2: “Factor” Program Modification

Modify the program “factor.asm” to calculate factorial of products instead of products of factorials. When you are done, calculate the expressions shown below with the help of TD.

Record the results on a sheet in hexadecimal and decimal format, including a screenshot and a brief explanation of the changes and results obtained.

1. (2 x 3) ! =
2. (2 x 4) ! =
3. (3 x 3) ! =
4. (5 x 2) ! =

Exercise 3: “Alumno” Program Modification

The program “alumno.asm”, has been designed to ask the user for the introduction of a name using the keyboard and print a line of text on the screen, using the input name.

You can find the program in the subfolder “alumnos” inside the “pra0” directory.

In this exercise you must repeat the process made with the “factor” program (i.e. assemble, link and run using the TD). This program can be also executed from the command line, since it includes input/output data by keyboard/screen.

Make the appropriate source code modifications to ask separately for the name, surname and country of the user, and then print a single line of text including the 3 previously entered fields. For instance: “John Doe (FROM United States) IS COURSING SECOND COURSE OF COMPUTER SCIENCE”.

During the debugging process, we can see what the program "alumno" is writing on screen by using the key combination ‘Alt+F5’.

Once the exercise is done you must add comments to the source code explaining all the changes performed.

<u>ASSIGMENT SUBMISSION: CONTENTS</u>

<u>Exercises 1 and 2:</u> A report must be upload to Moodle. It must contain the results, in both decimal and hexadecimal format, including screenshots of DosBox and a brief explanation of the modifications performed, as well as the results obtained.

<u>Exercise 3:</u> In this case you must upload a compress zip file containing both the source code and a makefile. The source code must include the comments with the modifications carried out and the student’s name in its header.

<u>ASSIGMENT SUBMISSION: DEADLINES</u>

<u>Exercises 1 and 2:</u>

Thursday’s groups: March 4th at 20:15

Friday’s groups: March 5th at 19:15

<u>Exercises 3:</u>

Thursday’s groups: March 10th at 23:55

Friday’s groups: March 11th at 23:55