Microprocessor-Based Systems - Lab Sessions.

P0: 80x86 Development and debugging environment tutorial

The goal of this lab session is to become familiar with the 80x86 development environment. The student shall follow the instructions in a step-by-step basis and complete the proposed exercises.

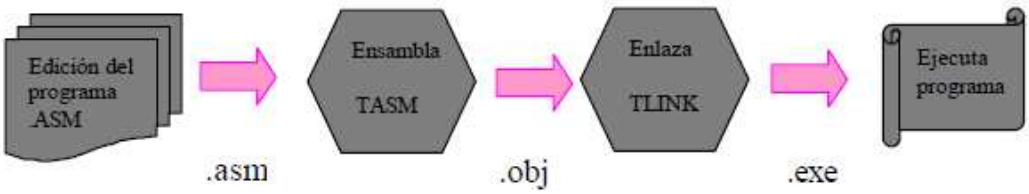
Assembler Program Develop Walkthrough

The common life cycle of an assembler program has the following steps:

1. Problem specification
2. Specific language flowchart development
3. Program codification
4. Assembly and linking
5. Execution
6. Debugging

To code ASM executable programs in a PC (80x86 architecture, MS-DOS operating system) you will need some Tools:

1. MS-DOS Operating System
2. A text editor to write your source code (.asm)
3. An “assembler” (also called “compiler”) that “translates” your source code to an object machine code (.obj)
4. A linker that generate the executable program from the object previously generated.



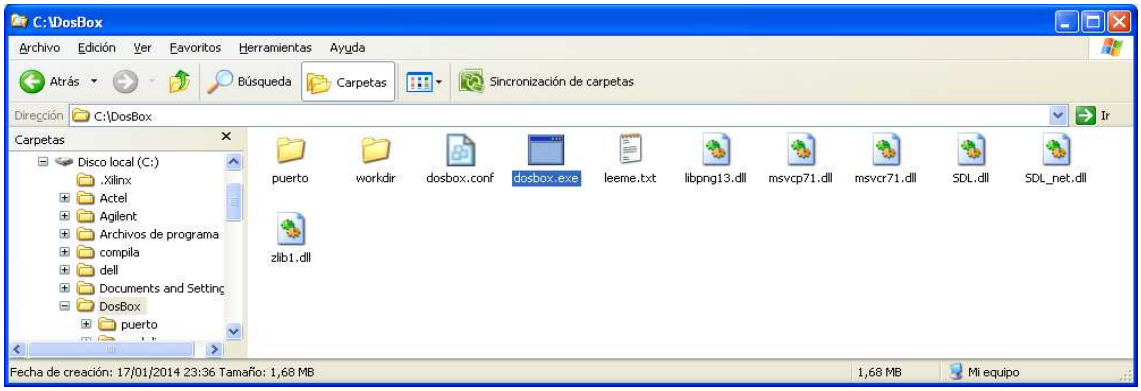
MS-DOS Operating System

PCs in the labs doesn’t have MS-DOS as native operating system, because of this we will use a program called **DOSBox**. DOSBox is an emulator that creates a very similar environment to MS-DOS operating system that will allow us to execute programs originally written for MS-DOS in modern computers with newer operating systems.

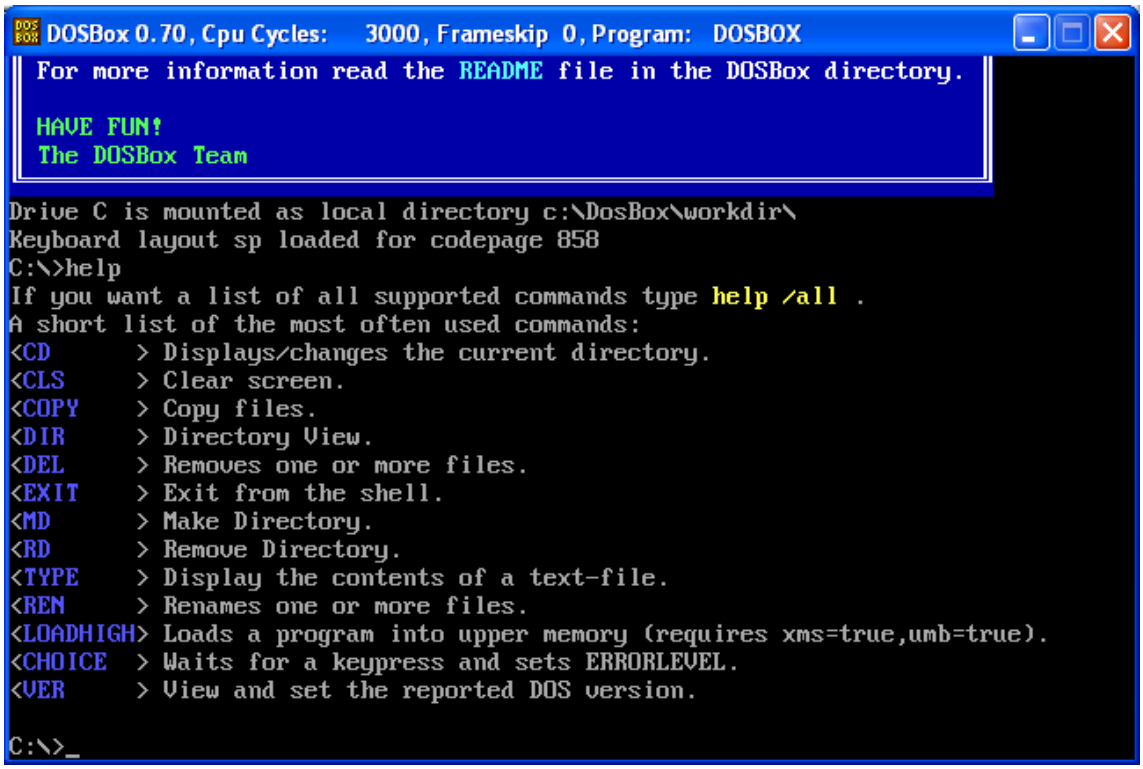
Open Windows Explorer and select the folder where DosBox is installed (C:\Program Files (x86)\DosBox), and copy the complete folder \DosBox\ in the root directory of drive C (C:\) . The subfolder Workdir (DosBox/Workdir) will be our default working directory. Workdir has all programs and tools needed to compile, link and debug the programs that we will code.

You may also find on that folder all the files that we will use in this tutorial (under pra0 subfolder)

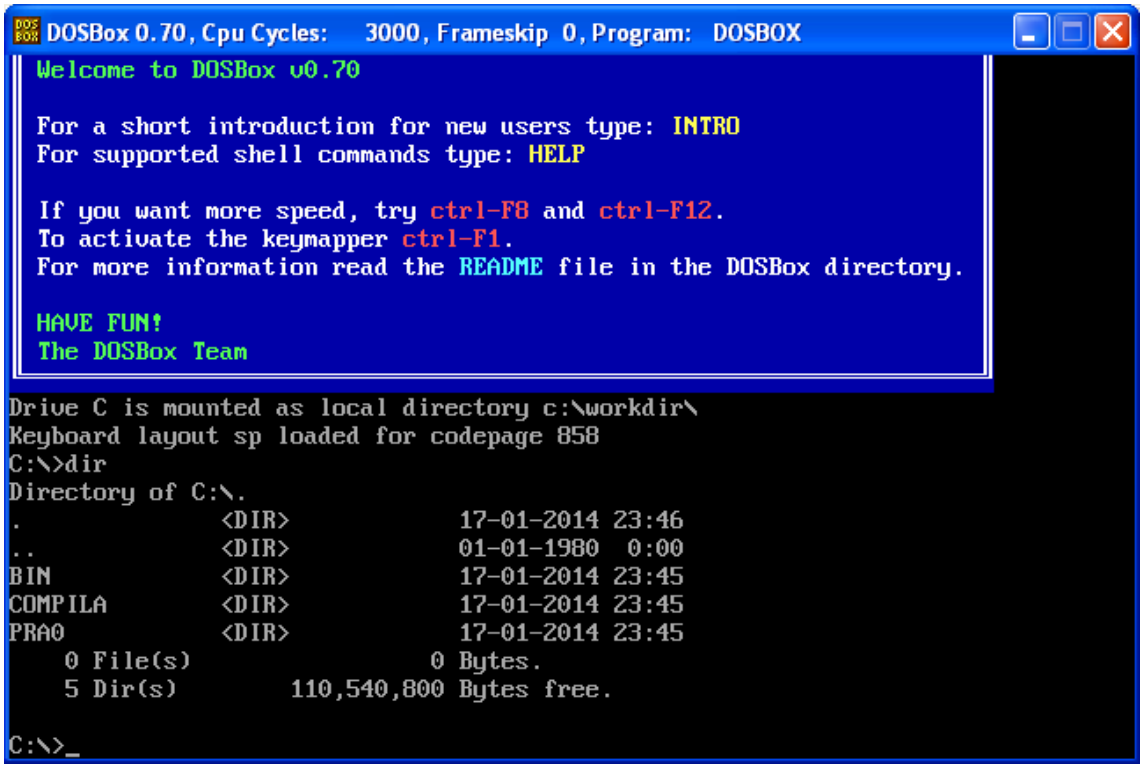
After copying Workdir we can execute the DosBox program that is found drive C (C:\DosBox\DosBox.exe), by double clicking on it.



It will open a new command window (like the one at next capture). We will compile, link and execute our programs using that window. Typing “help”, DosBox will show us all the supported commands. The ones we are going to use most are: “cd” to change directory, “dir” to list the files at the current directory, and “del” to delete files.



DosBox is configured to mount virtual drive C: at C:\DosBox\Workdir, i.e. when typing “dir” at C: inside DosBox we will see the contents of C:\DosBox\Workdir.

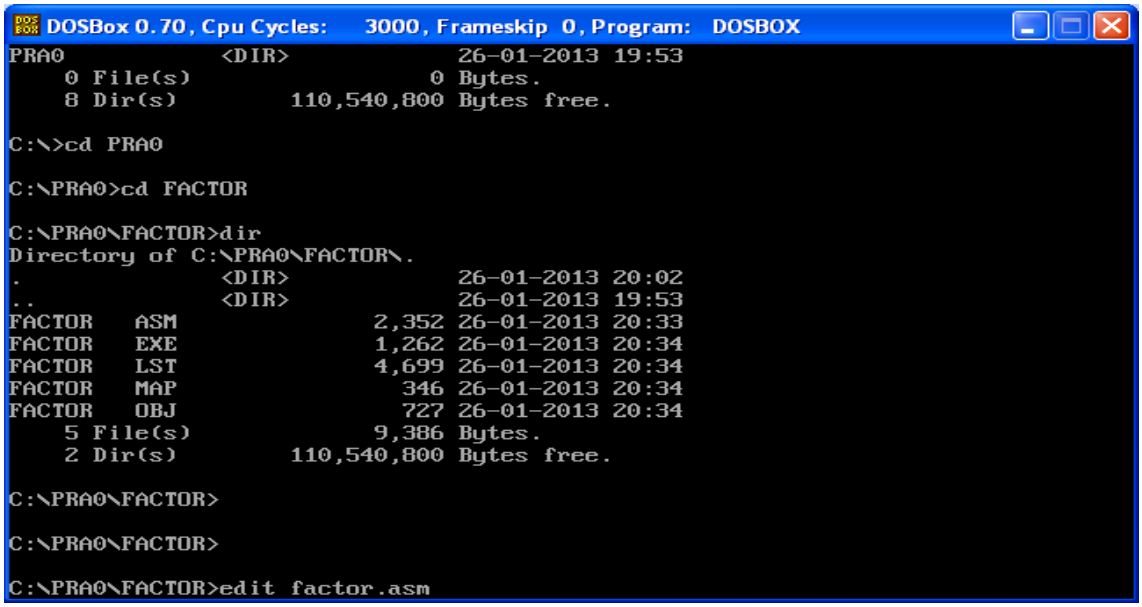


Source Code editing

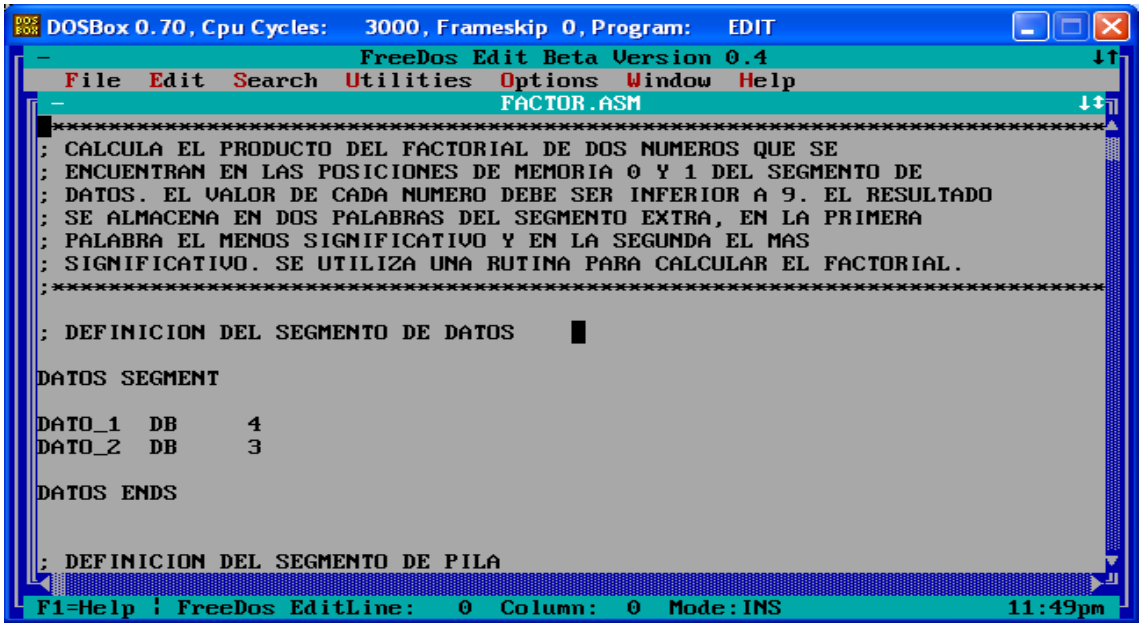
We can use any ASCII text editor to write our source code but to assemble it correctly the name of the file must have no more than 8 characters and .ASM extension.

DosBox has an internal text editor that we can launch typing EDIT inside DosBox window. To learn how it works we change to the “pra0” directory (command “cd pra0”) and then to the “factor” subfolder (command “cd factor”), We then type “edit factor.asm” to start editing the file factor.asm using EDIT.

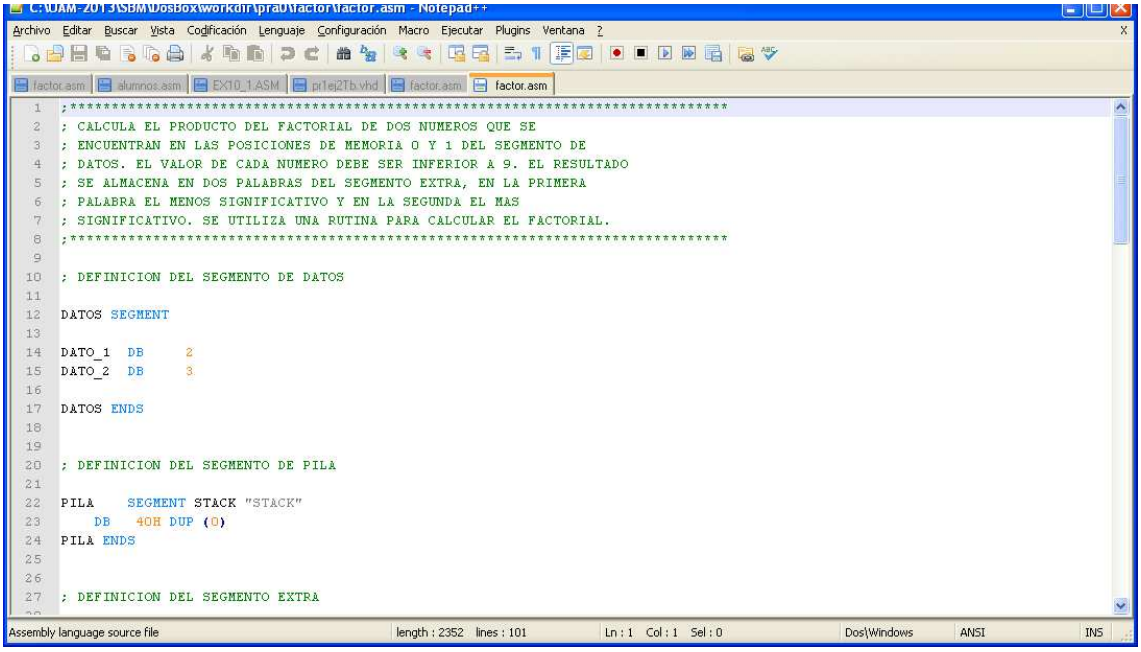
All this process can be seen in the picture bellow.



EDIT editor will appear inside the DosBox window. To access the top menu you should press the ALT key. Inside the top menu you can change the active option using the arrow keys, and then press the enter key to confirm the selected one.



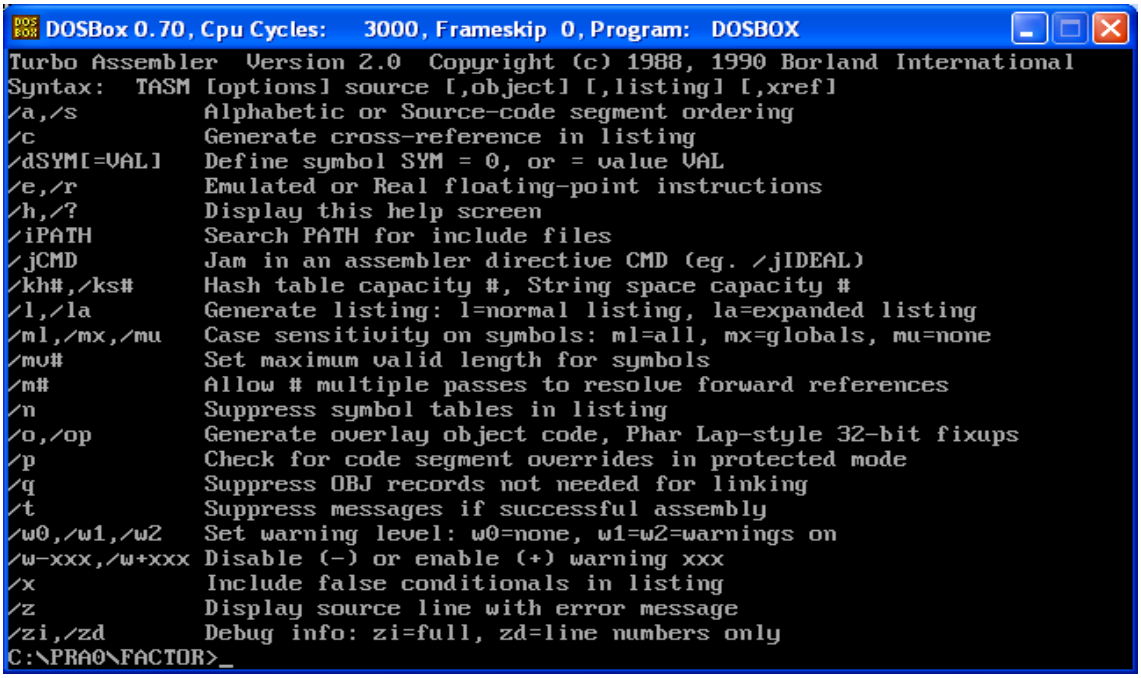
Alternatively to EDIT, we can use Windows Notepad or Notepad ++ (the latter has the advantage of syntax highlighting). To do that, just open the factor.asm file located in C:\DosBox\Workdir\pra0\factor\



Assembling Programs

To assembly a program we will use Turbo Assembler inside DosBox (TASM). It will generate the object file (.OBJ extension) from a source file (.ASM extension). The OBJ file is an intermediate file between a source file and an executable file.

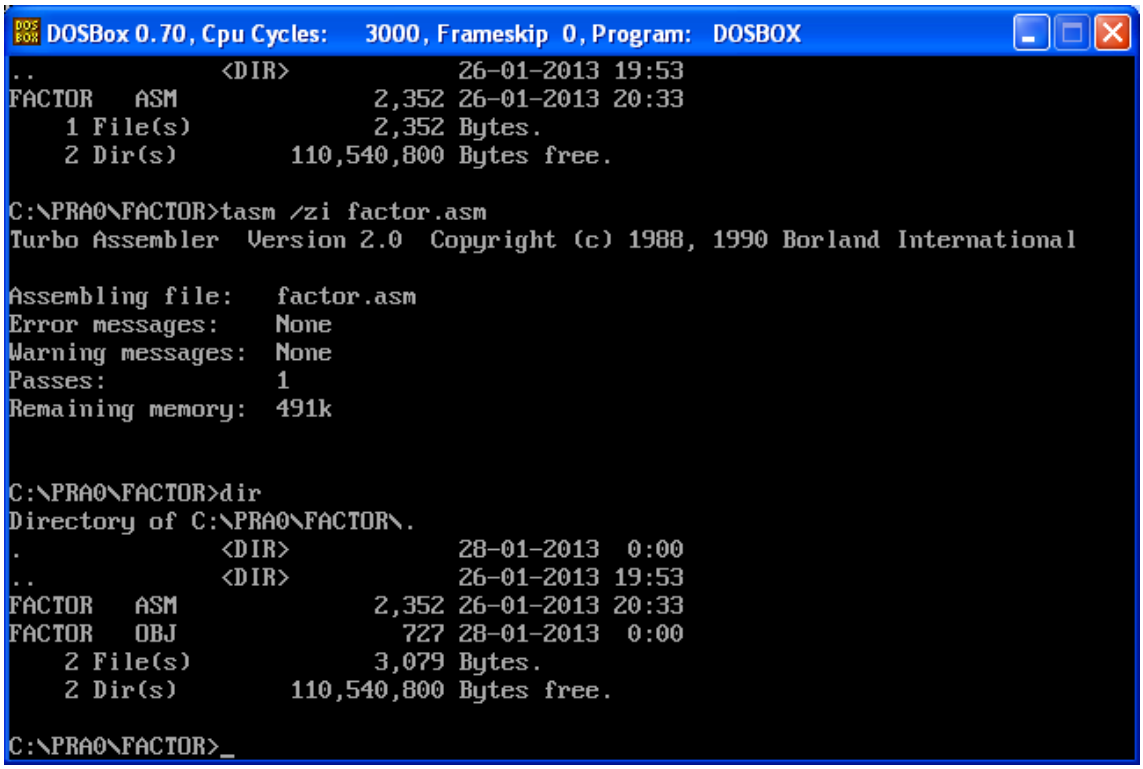
We may observe all TASM options typing “tasm” inside DosBox window.



In this picture we can see the assembly process of factor.asm file. To do this we type “tasm /zi factor.asm”. The “/zi” option is needed to have complete debug information (see previous picture). We can find factor.obj if we execute “dir” command.

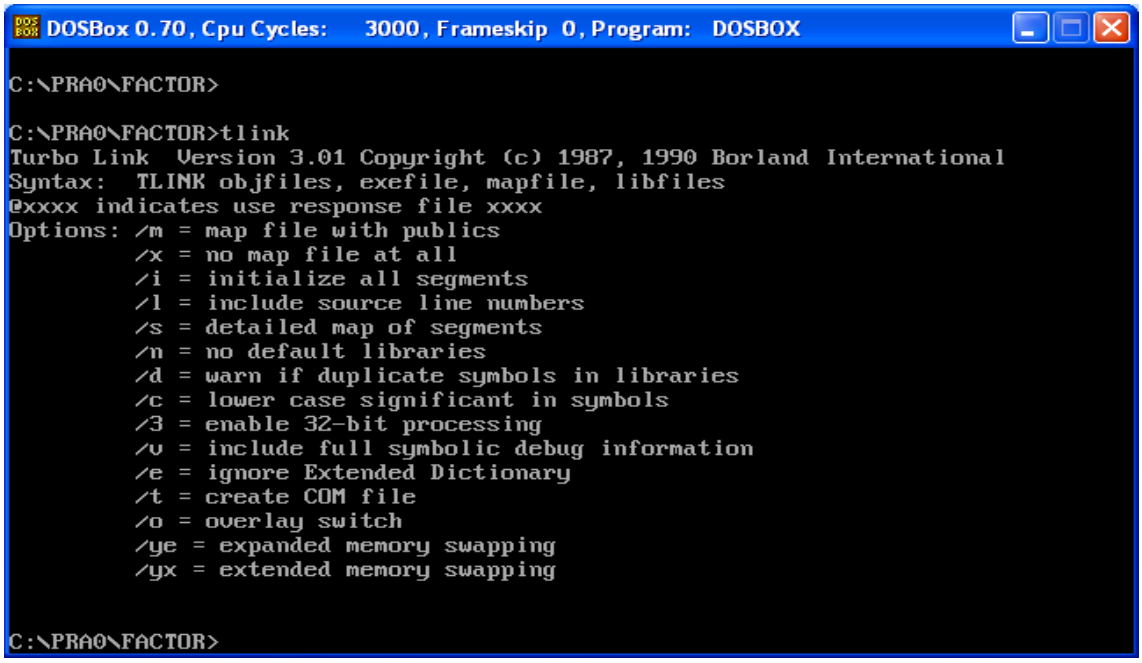
NOTE: It is possible that the factor.obj file already exists before executing tasm. In that case, we may delete it by typing " factor.obj" before executing Tasm.

.

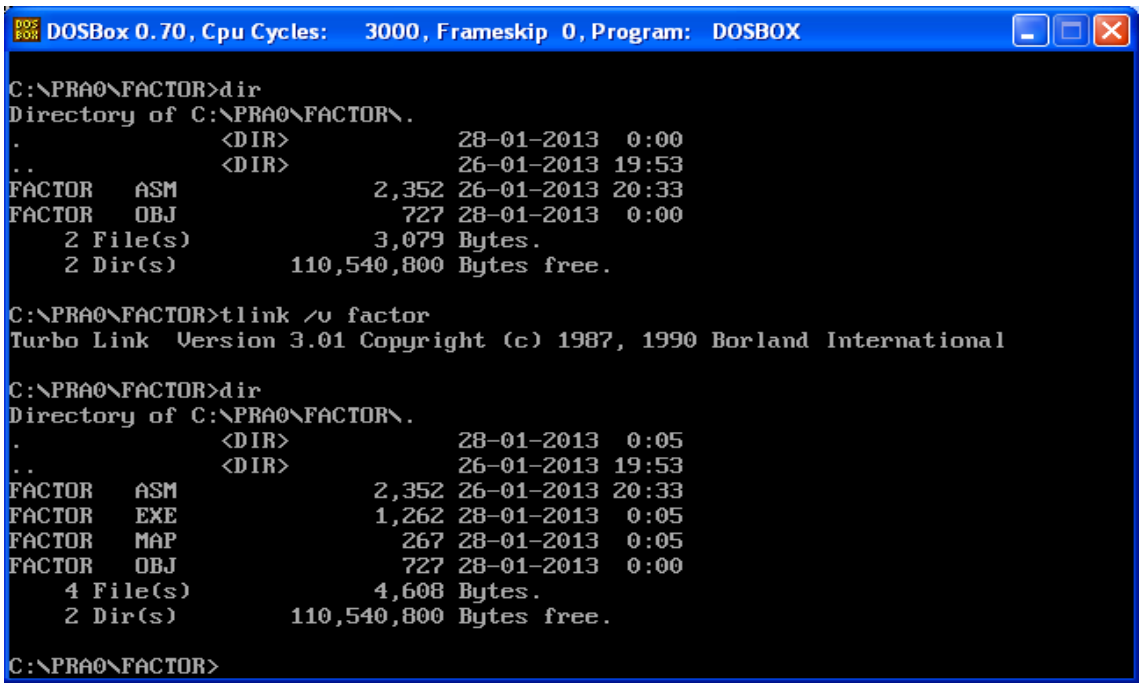


Linking Programs

To link programs in DosBox we will use Turbo Link (TLINK). It will generate an executable file from one or more .OBJ files. The different options of TLINK are shown by typing “tlink” in the command line.



In the following picture we can observe the process of generating the executable file from the object file. The sequence to be typed is “tlink /v factor”. Notice that we used the /v option to copy debugging information in the executable file (see previous picture). Finally, we may find the .exe file by typing “dir” command in DosBox window.

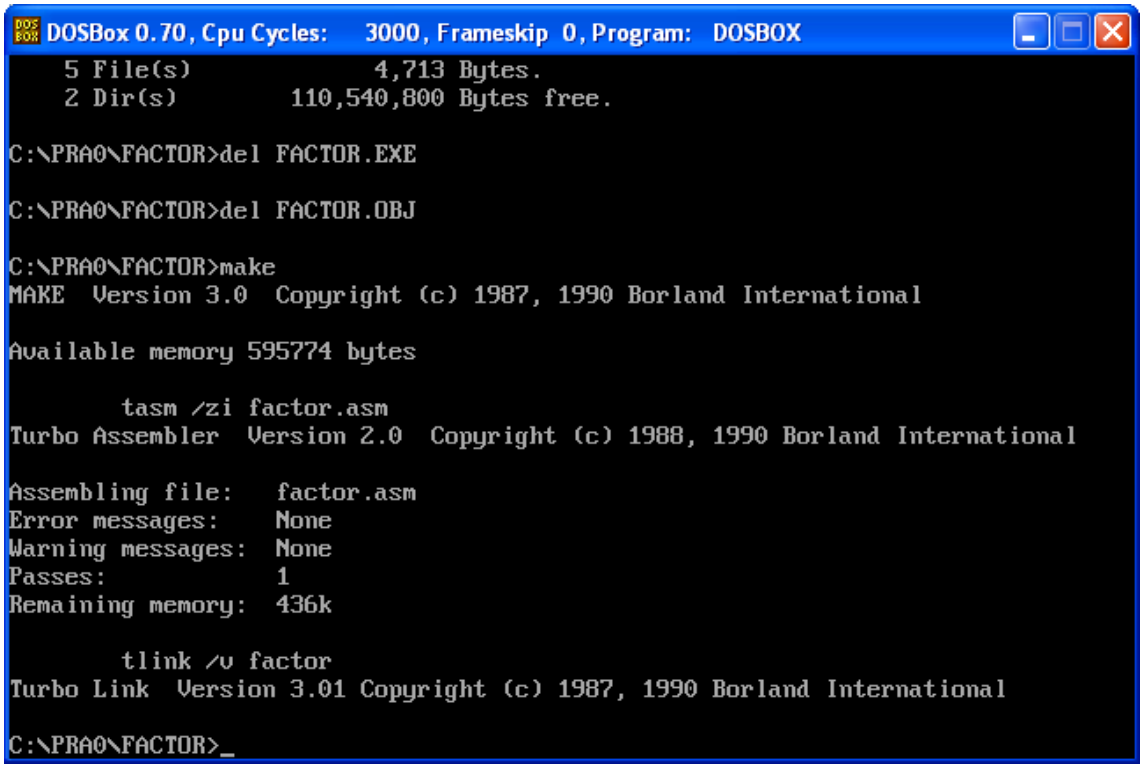


Using Makefiles

All previous steps (assembling and linking) can be created automatically by using a “makefile”. The makefile is a new file that shall be created inside the source files folder. (“factor” in this example). We may code this file with the help of notepad or NotePad++, and it must be saved with the name “makefile”. File contents should be as follow (respecting tabs):

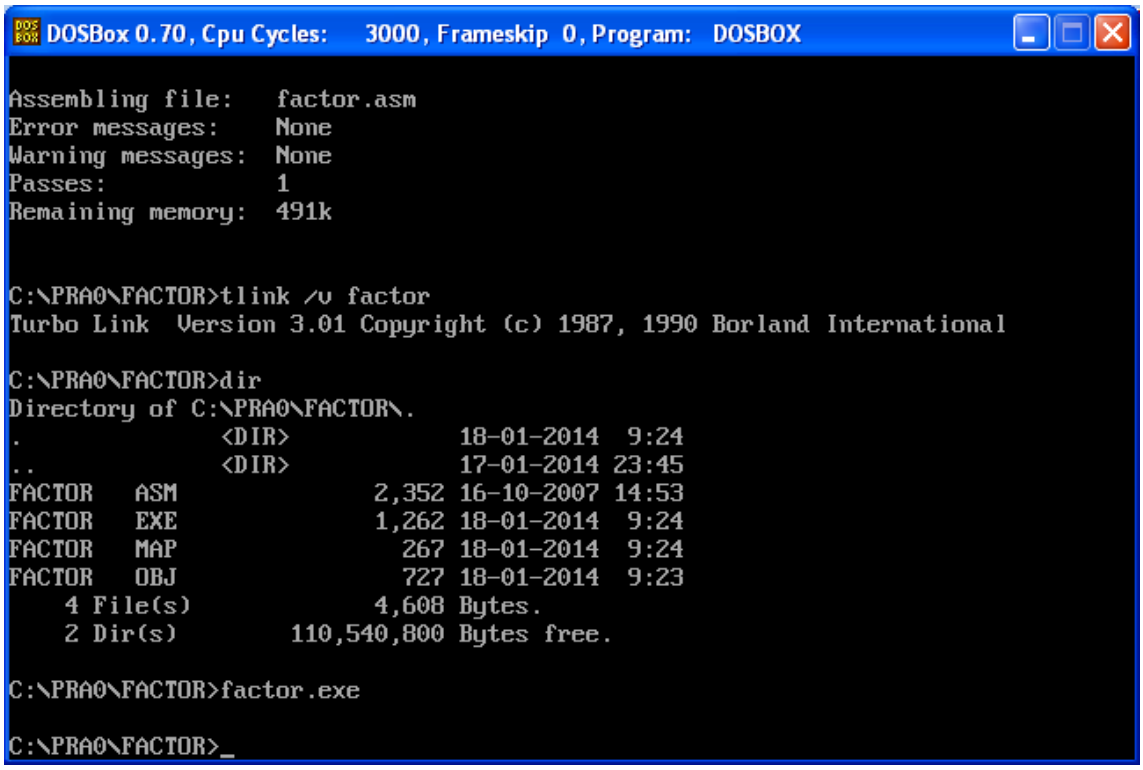
```
all: factor.exe

factor.exe: factor.obj

	tlink /v factor

factor.obj: factor.asm

	tasm /zi factor.asm
```

To verify that the makefile is working properly, we should delete factor.exe and factor.obj by using the “del” command. Just type “make”, and the processes for assembly and linking should be executed, while generating the .OBJ and .EXE files.



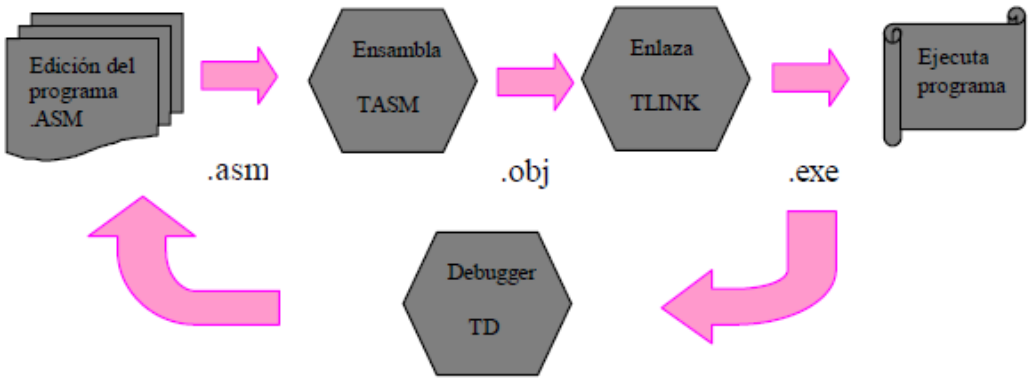
Program Execution

We can execute it directly from DosBox command line typing the program name. Be aware, as in this case, the execution will not be very useful since the program won't print any information on screen.

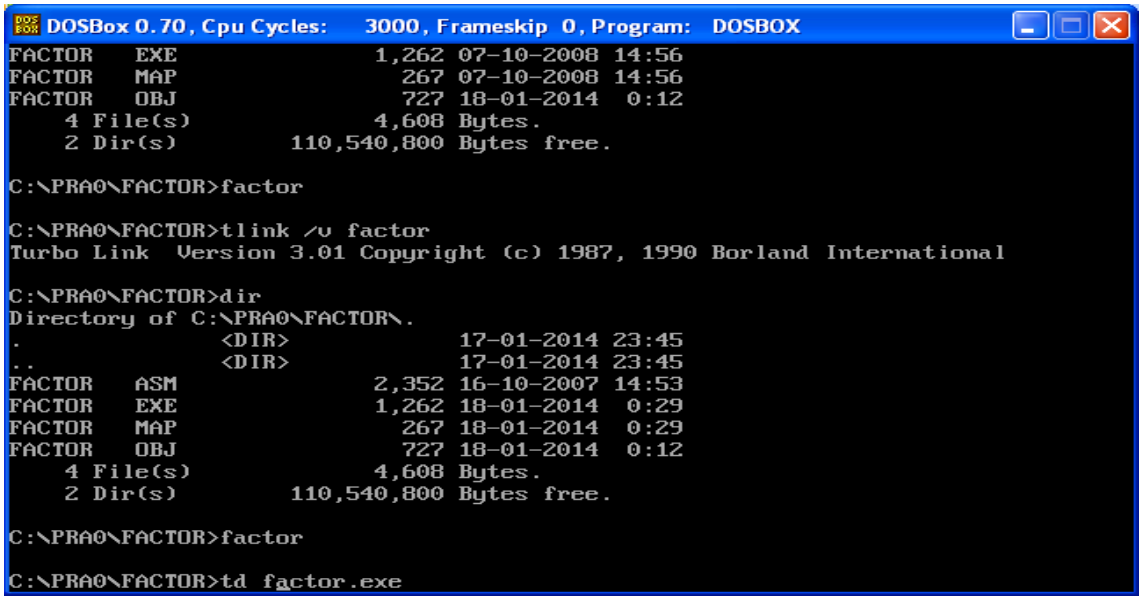


Debugging Programs

The debugger is the tool we use to follow the execution of a program step by step. We will use Turbo Debugger (TD) from Borland..



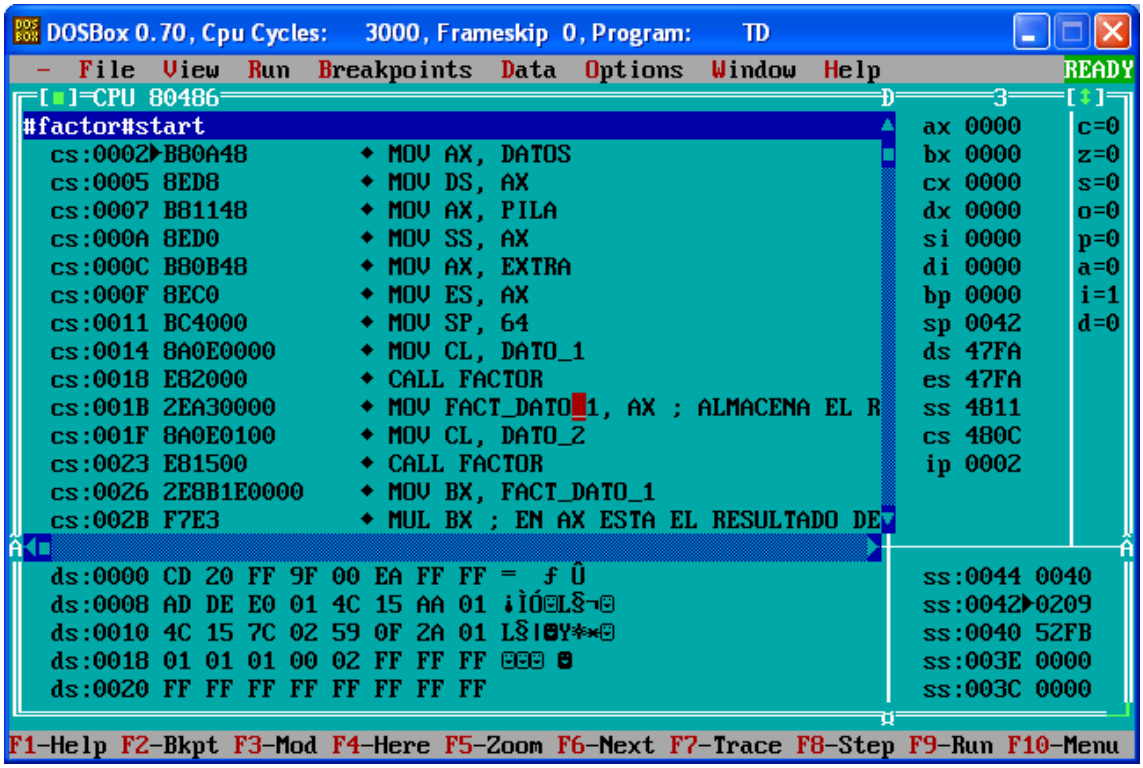
In order to see the step-by-step operation of the program factor, we run the program by typing “td factor.exe” in DosBox.



Once TD is launched, it will open a window similar to this one:

Normally, we will not use this view of TD, but the view called "CPU". To change to the “CPU” view, we may press ‘Alt+V’ and then ‘C’.

To maximize the screen, please press the ‘F5’ key.



This screen has two command bars (one at the top of the window and one at the bottom) and 5 windows in the main area.

The top bar contains all user menus. We can enter that bar pressing the 'Alt' key and then move through it using the arrow keys, and then press the 'enter' key to confirm the selected one. Press the 'ESC' key to exit the menu.

The bottom bar includes shortcuts to options using 'F1' to 'F10' keys. For instance, we can run our program by pressing 'F9'.

We have 5 windows in the main area. We can only have one of them active. The active window can be changed by pressing the 'Tab' key. The top left window is selected by default, and contains the program disassembled.

Let's look carefully at the first line:

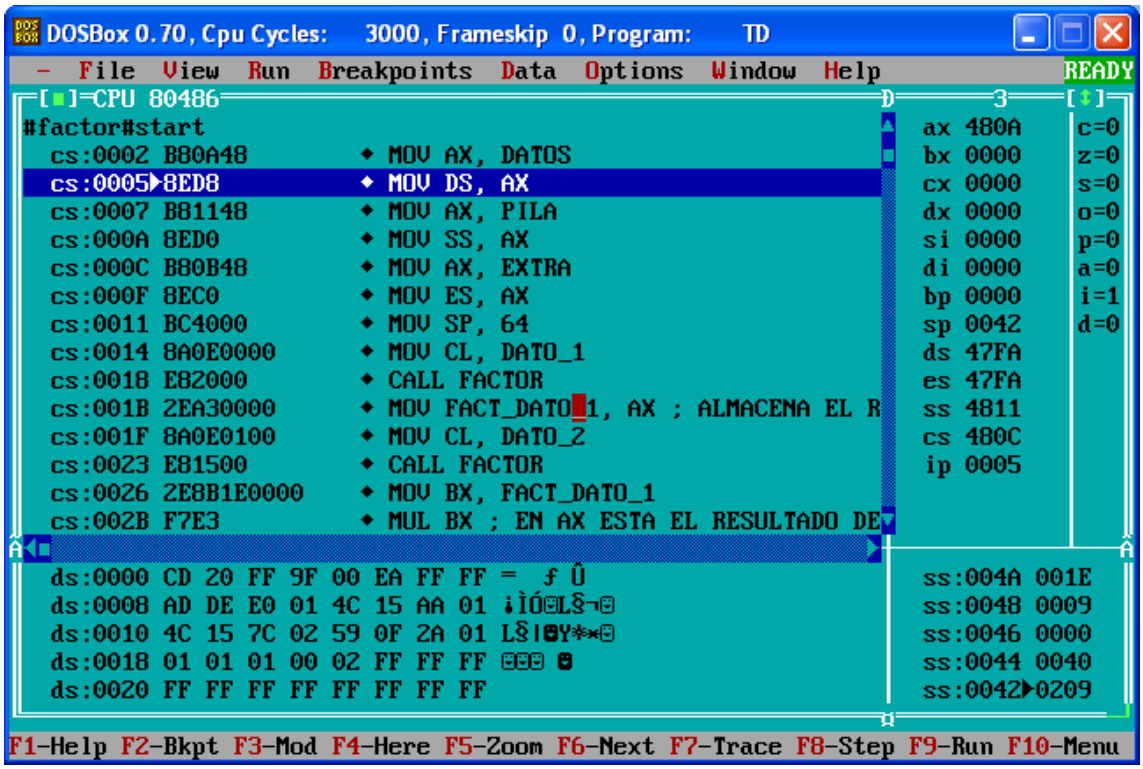
cs:0002 ►B80A48 MOV AX, DATOS

In general, each line contains 3 different fields:

1. the memory address of the code line, in this example cs:0002
2. the execution code in hexadecimal, in this case the instruction has 3 bytes
3. the corresponding code in assembly language

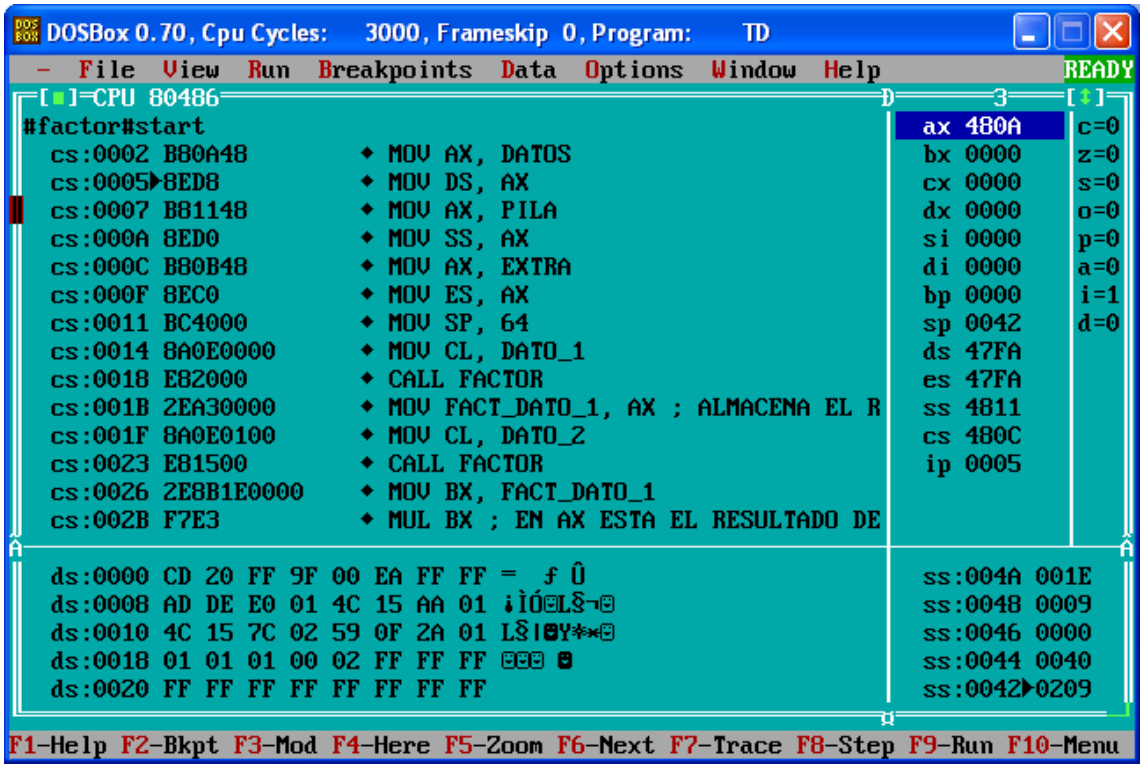
Therefore, the position 0002 at code segment (cs) contains the hexadecimal code "B80A48" that corresponds with the "mov ax, datos" instruction.

The marker indicates the next instruction to be executed. By pressing the ‘F7’ key (Trace), we execute the first instruction and we can observe how the marker moves to the second instruction. In general, we may press repeatedly the ‘F7’ key to observe how the instructions of the program are run, and the changes produced in the other 4 windows.

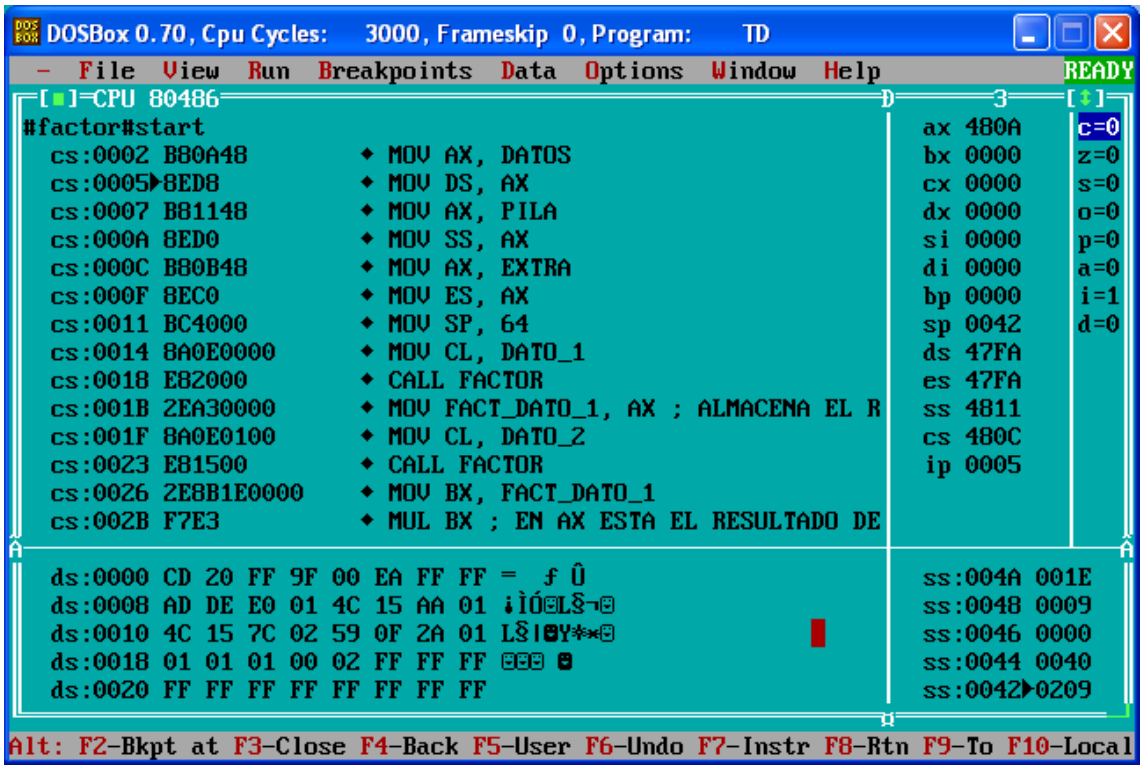


Pressing ‘Alt+F10’ (or the right mouse button) we get access to the context menu of the active window. In this case, once in the menu, we can change the “Mixed” mode state by pressing ‘M’. If we change it to “Yes”, we will see the instructions as they are written in the source file, along with its object code translation. The most common mode to use is “Both”. To exit from the window menu, press ‘ESC’.

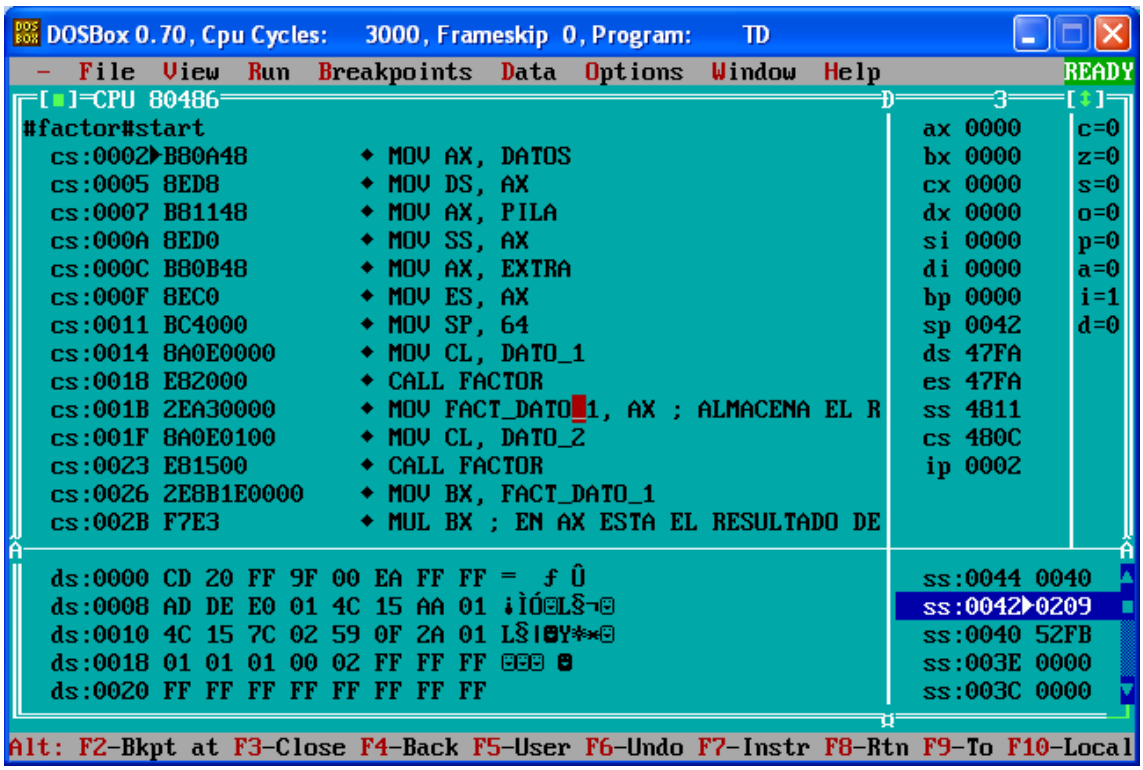
The second window on top shows the hexadecimal content of the 80x86 registers. We may see the how the register’s values are changing each time an instruction is executed. In particular, we may observe the changes in the ‘IP’ register every time an instruction is executed.



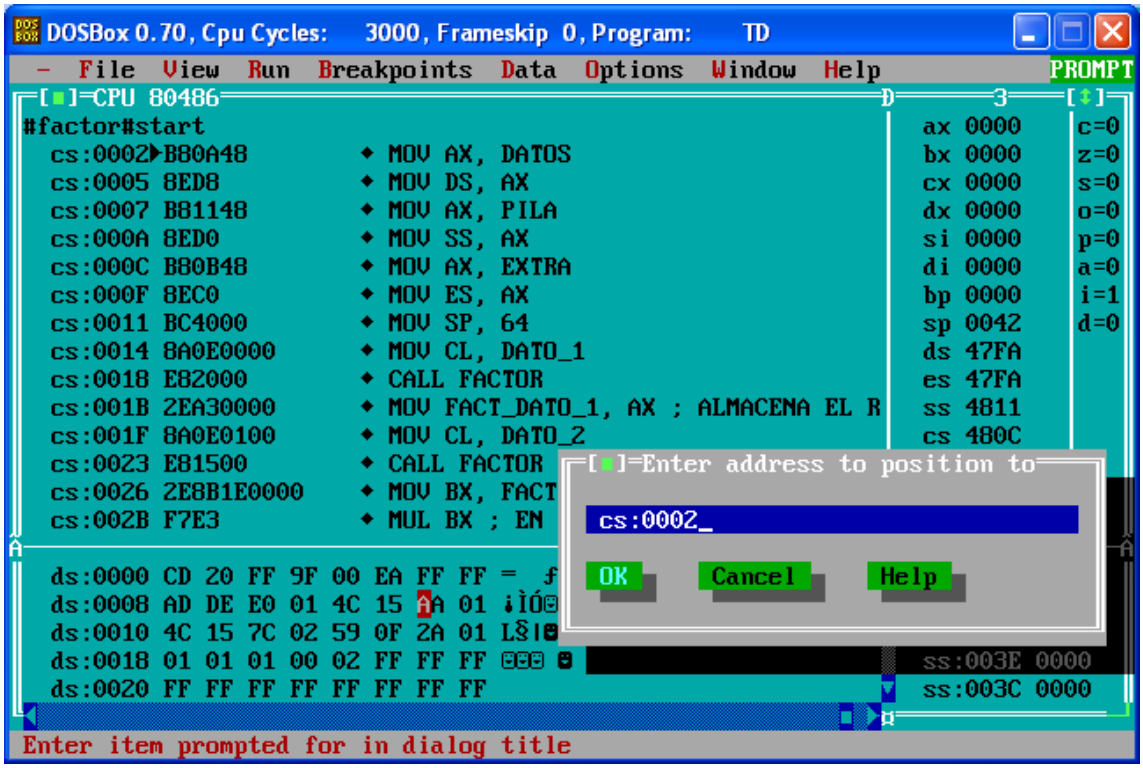
The third window, at the right of the previous one, shows the flag registers. Each flag is named according its initial (C-Carry, C-Zero, S-Sign,O- Overflow,P- Parity, A-Above (BCD carry), I-Interrupt and D-Direction), along with its binary value.



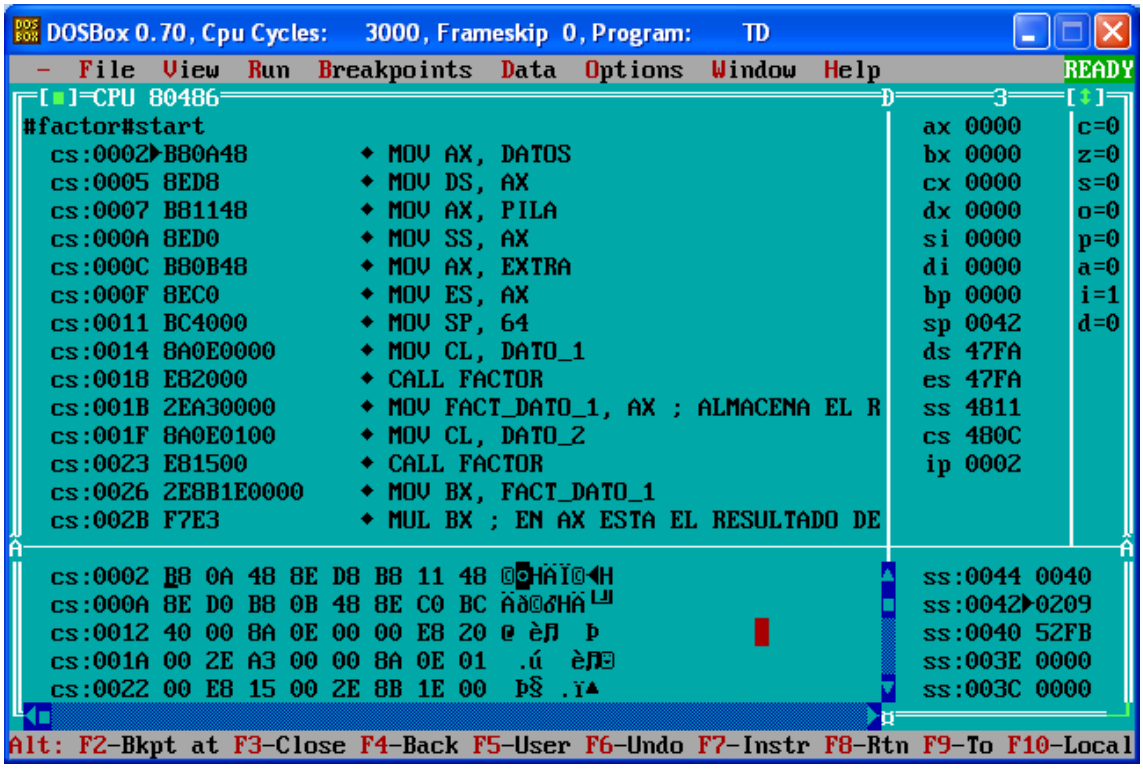
The fourth window, on the bottom right corner, displays the contents of the stack currently in use by the program. The arrow marker indicates the current position of the stack pointer.



Finally, the fifth window, that is located in the bottom left, shows the contents of a specific memory area. It can display any desired area by clicking in the context menu of this window and using the ‘Goto’ option.



Notice that if we type the address cs:0002, we see the program instructions stored in that area of memory. Note the correspondence between the code and the data window.



To exit the TD program and return to the DosBox command line, simply use ‘ALT+x’.

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

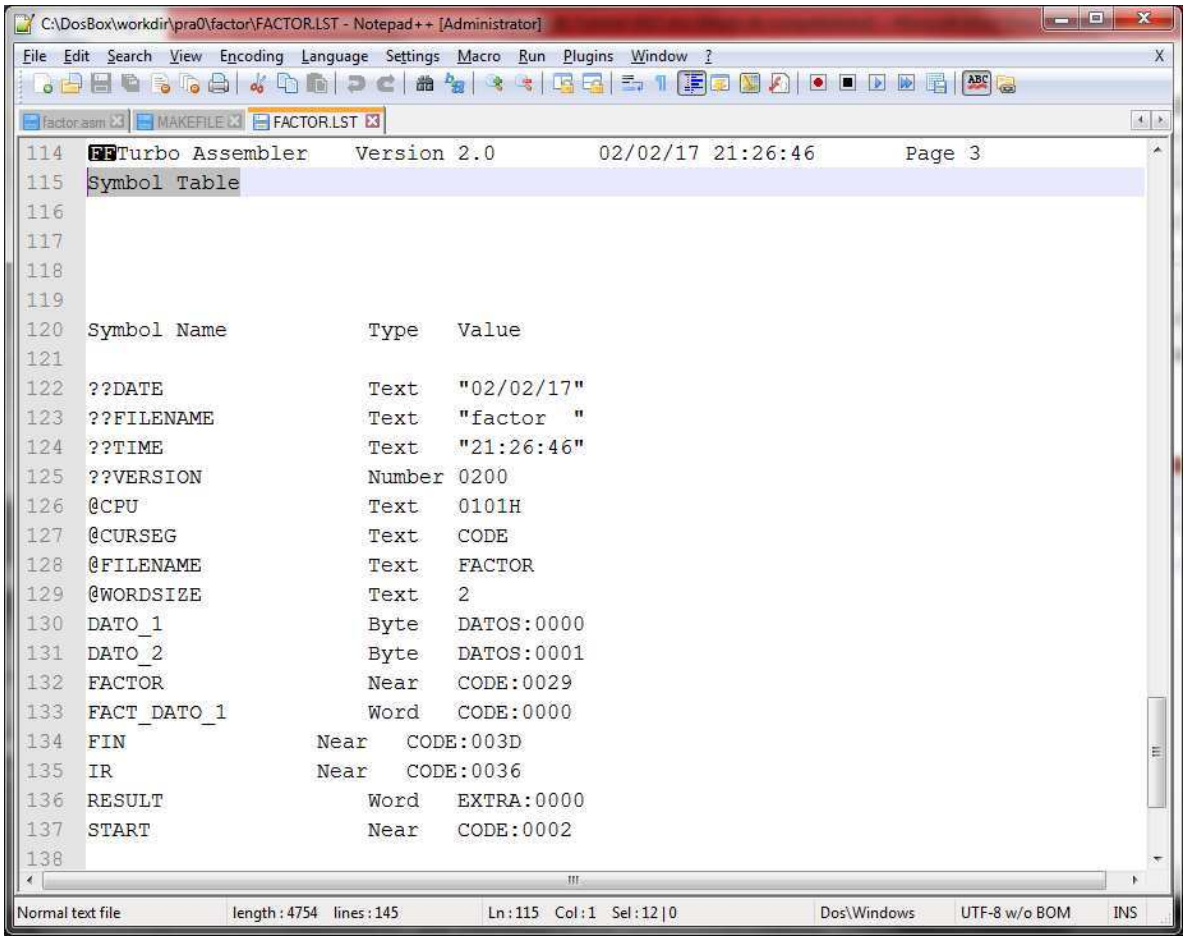
by

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.



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

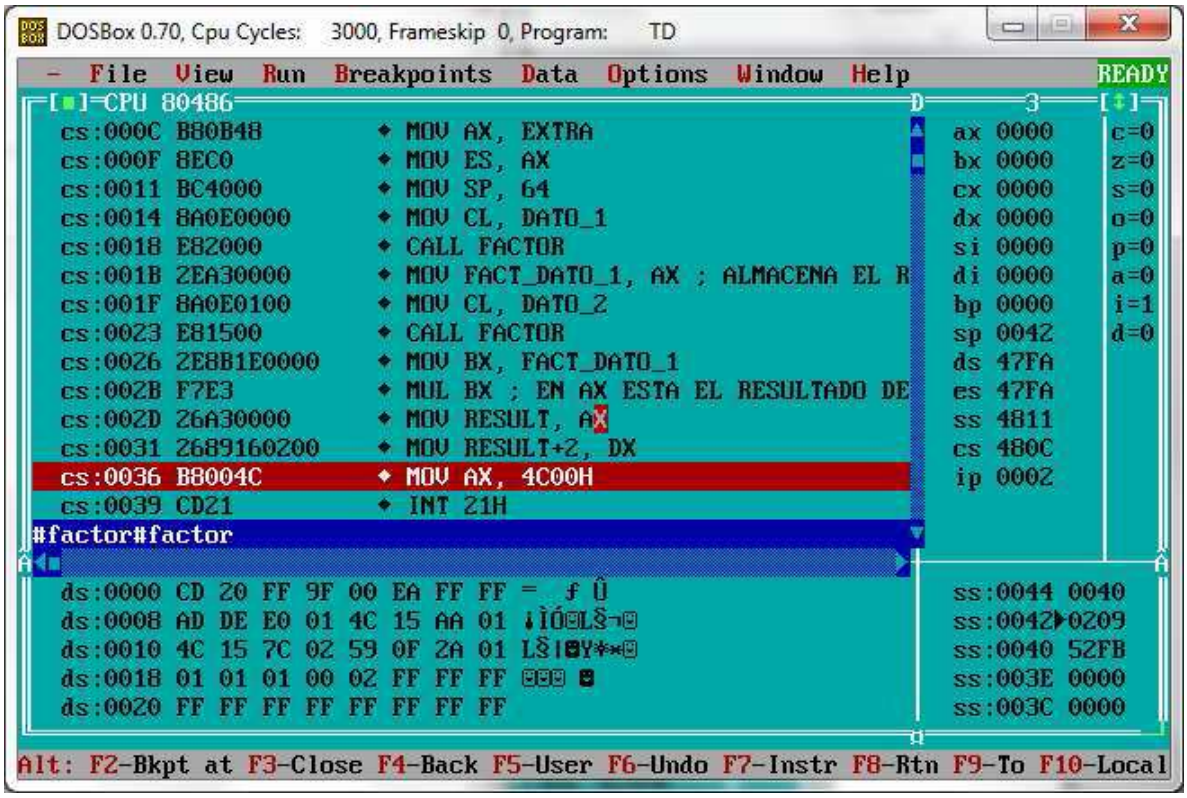
FACT_DATO_1: CODE segment, offset 0

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

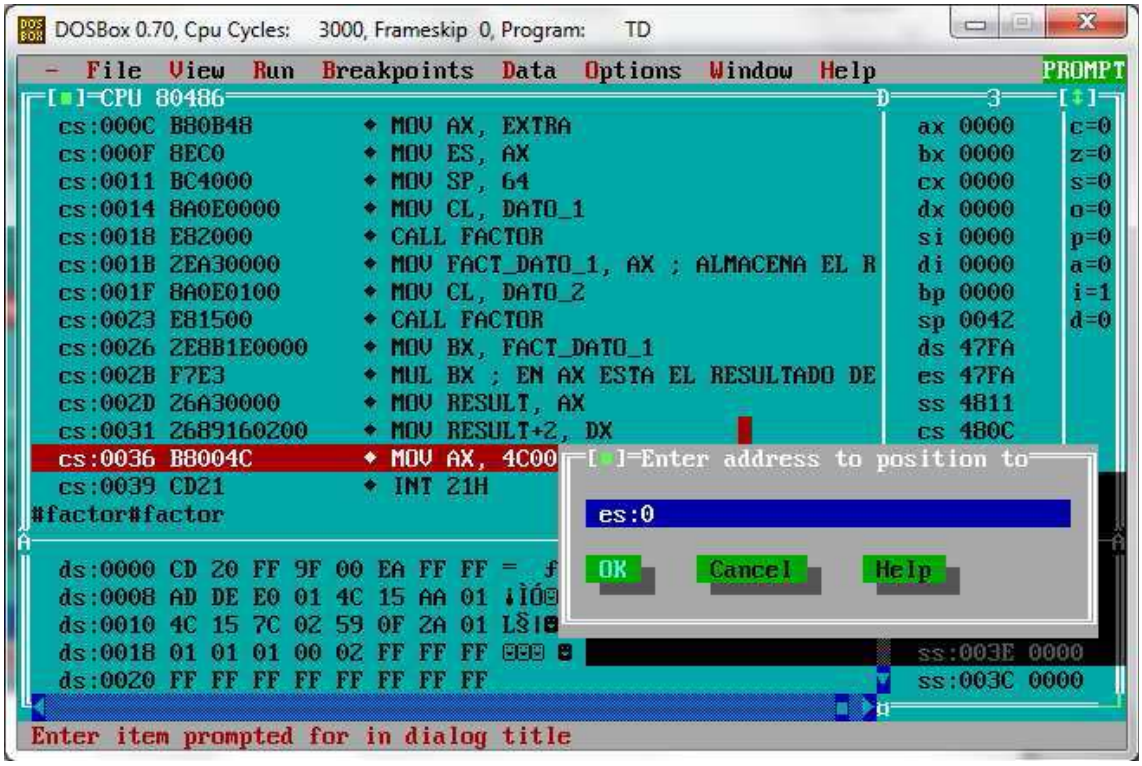
To do this, select the instruction with the cursor and the press F2. Move the cursor upwards to check that the breakpoint has been activated, (indicated with a red line)



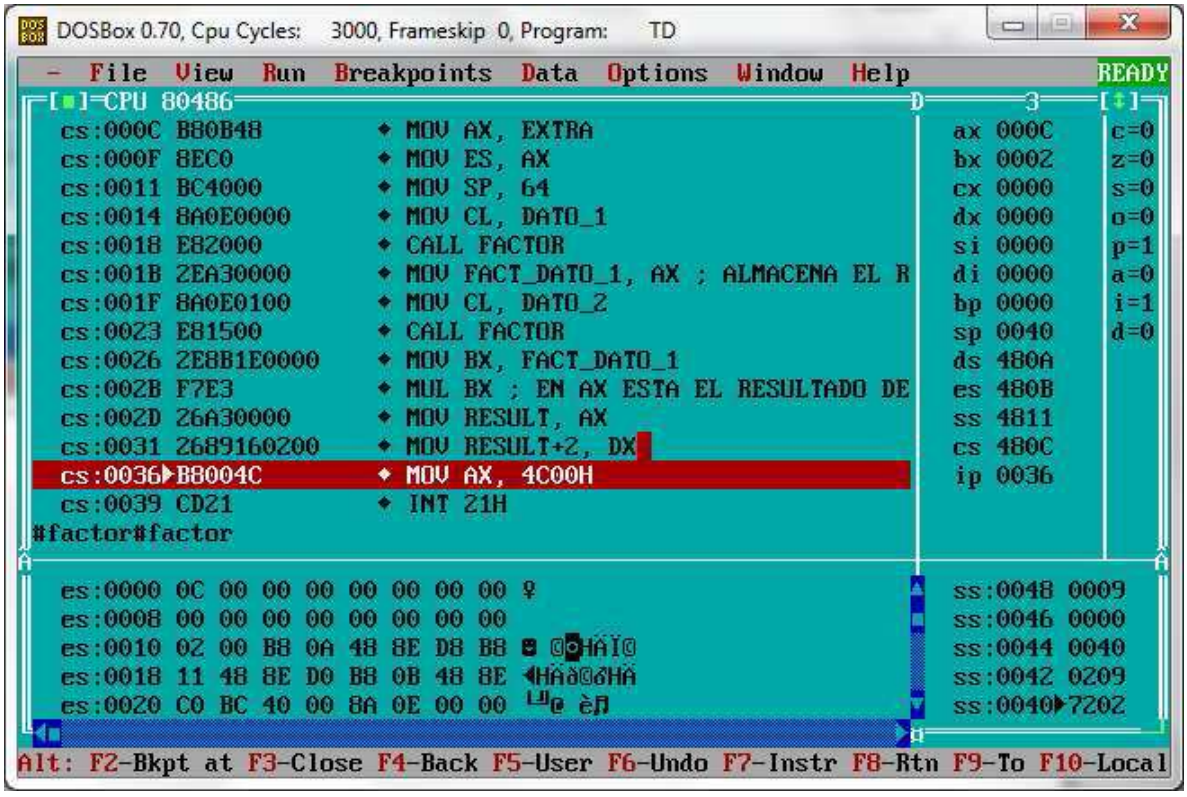
Then execute the program pressing F9 key. The program will then be executed and stopped at the breakpoint. In this moment, the RESULT variable should contain its final value.

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.



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



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, and show them to the teacher, including a brief explanation of the results.

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

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, and show them to the teacher, including a brief explanation of the results.

1. (2 x 3) ! =
2. (2 x 4) ! =
3. (3 x 3) ! =
4. (2 x 7) ! =

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 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 show the source code and its execution to the teacher .