

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.

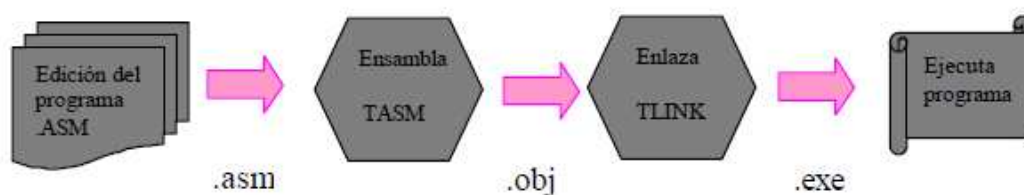
Assembler Program Develop Walkthrough

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.



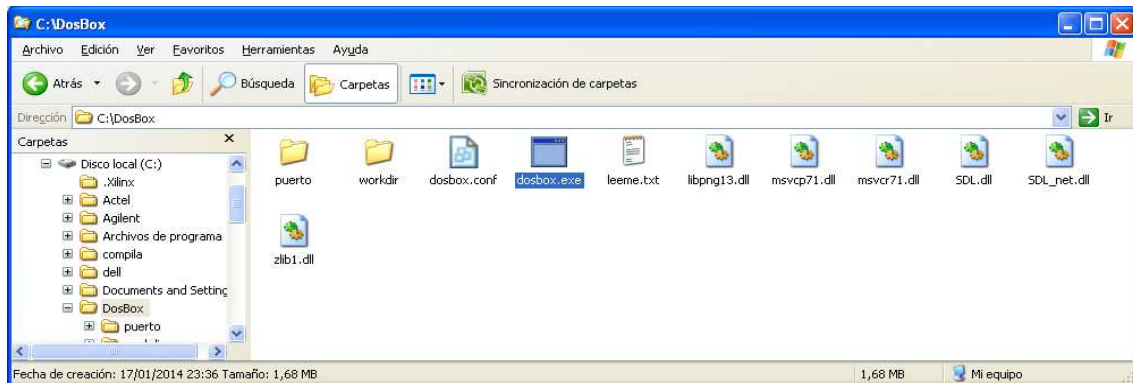
MS-DOS Operating System

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.

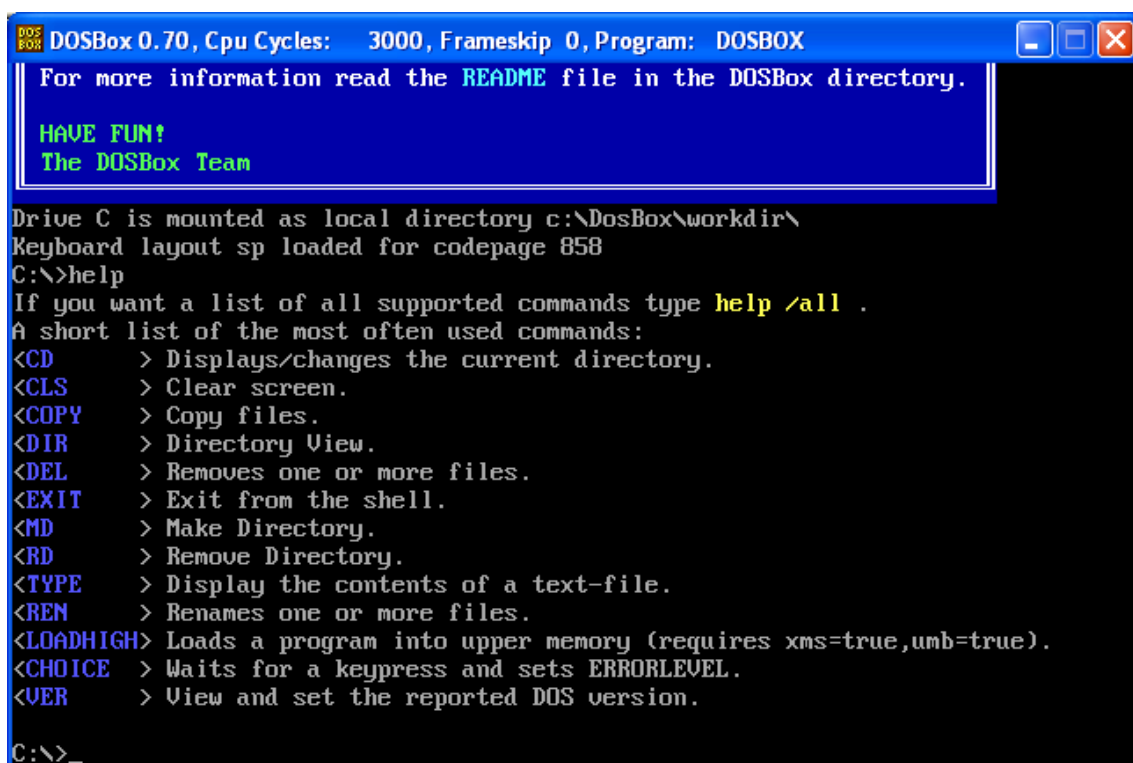
Open Windows Explorer and select the folder where DosBox is installed (C:\Program Files (x86)\DosBox), and copy the complete folder \DosBox\ in the root directory of drive C (C:\) . The subfolder Workdir (DosBox/Workdir) will be our default working directory. Workdir has all programs and tools needed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

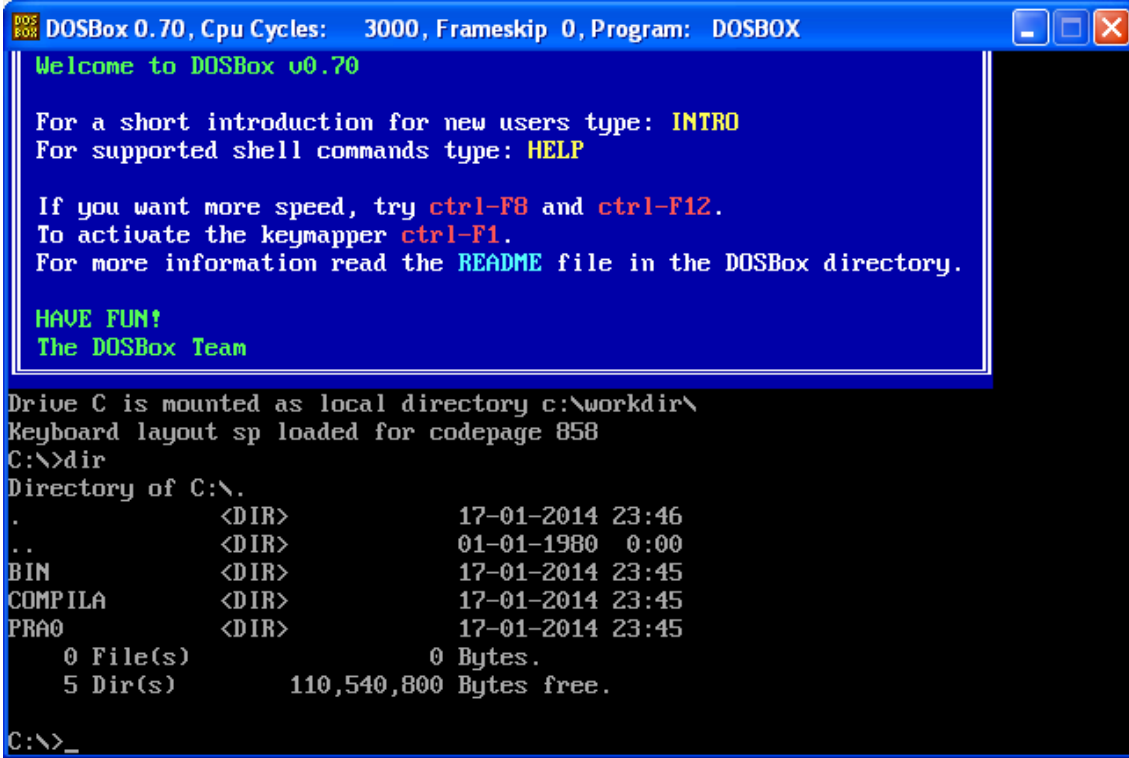
After copying Workdir we can execute the DosBox program that is found drive C (C:\DosBox\DosBox.exe), by double clicking on it.



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.



```
DOSBox 0.70, Cpu Cycles: 3000, Frameskip 0, Program: DOSBOX
Welcome to DOSBox v0.70

For a short introduction for new users type: INTRO
For supported shell commands type: HELP

If you want more speed, try ctrl-F8 and ctrl-F12.
To activate the keymapper ctrl-F1.
For more information read the README file in the DOSBox directory.

HAVE FUN!
The DOSBox Team

Drive C is mounted as local directory c:\workdir\
Keyboard layout sp loaded for codepage 858
C:\>dir
Directory of C:\.
.                <DIR>                17-01-2014 23:46
..               <DIR>                01-01-1980  0:00
BIN              <DIR>                17-01-2014 23:45
COMPILA         <DIR>                17-01-2014 23:45
PRA0            <DIR>                17-01-2014 23:45
    0 File(s)                0 Bytes.
    5 Dir(s)                110,540,800 Bytes free.

C:\>_
```

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.

```
DOSBox 0.70, Cpu Cycles: 3000, Frameskip 0, Program: DOSBOX
PRA0 <DIR> 26-01-2013 19:53
 0 File(s) 0 Bytes.
 8 Dir(s) 110,540,800 Bytes free.

C:\>cd PRA0

C:\PRA0>cd FACTOR

C:\PRA0\FACTOR>dir
Directory of C:\PRA0\FACTOR\
. <DIR> 26-01-2013 20:02
.. <DIR> 26-01-2013 19:53
FACTOR ASM 2,352 26-01-2013 20:33
FACTOR EXE 1,262 26-01-2013 20:34
FACTOR LST 4,699 26-01-2013 20:34
FACTOR MAP 346 26-01-2013 20:34
FACTOR OBJ 727 26-01-2013 20:34
 5 File(s) 9,386 Bytes.
 2 Dir(s) 110,540,800 Bytes free.

C:\PRA0\FACTOR>

C:\PRA0\FACTOR>

C:\PRA0\FACTOR>edit factor.asm
```

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.

```
DOSBox 0.70, Cpu Cycles: 3000, Frameskip 0, Program: EDIT
FreeDos Edit Beta Version 0.4
File Edit Search Utilities Options Window Help
FACTOR.ASM
*****
; CALCULA EL PRODUCTO DEL FACTORIAL DE DOS NUMEROS QUE SE
; ENCUENTRAN EN LAS POSICIONES DE MEMORIA 0 Y 1 DEL SEGMENTO DE
; DATOS. EL VALOR DE CADA NUMERO DEBE SER INFERIOR A 9. EL RESULTADO
; SE ALMACENA EN DOS PALABRAS DEL SEGMENTO EXTRA, EN LA PRIMERA
; PALABRA EL MENOS SIGNIFICATIVO Y EN LA SEGUNDA EL MAS
; SIGNIFICATIVO. SE UTILIZA UNA RUTINA PARA CALCULAR EL FACTORIAL.
*****

; DEFINICION DEL SEGMENTO DE DATOS
DATOS SEGMENT
DATO_1 DB 4
DATO_2 DB 3
DATOS ENDS

; DEFINICION DEL SEGMENTO DE PILA
F1=Help ! FreeDos EditLine: 0 Column: 0 Mode:INS 11:49pm
```

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\

```

1  ; *****
2  ; CALCULA EL PRODUCTO DEL FACTORIAL DE DOS NUMEROS QUE SE
3  ; ENCUESTRAN EN LAS POSICIONES DE MEMORIA 0 Y 1 DEL SEGMENTO DE
4  ; DATOS. EL VALOR DE CADA NUMERO DEBE SER INFERIOR A 9. EL RESULTADO
5  ; SE ALMACENA EN DOS PALABRAS DEL SEGMENTO EXTRA, EN LA PRIMERA
6  ; PALABRA EL MENOS SIGNIFICATIVO Y EN LA SEGUNDA EL MAS
7  ; SIGNIFICATIVO. SE UTILIZA UNA RUTINA PARA CALCULAR EL FACTORIAL.
8  ; *****
9
10 ; DEFINICION DEL SEGMENTO DE DATOS
11
12 DATOS SEGMENT
13
14 DATO_1 DB 2
15 DATO_2 DB 3
16
17 DATOS ENDS
18
19
20 ; DEFINICION DEL SEGMENTO DE PILA
21
22 PILA SEGMENT STACK "STACK"
23 DB 40H DUP (0)
24 PILA ENDS
25
26
27 ; DEFINICION DEL SEGMENTO EXTRA

```

Assembling Programs

To assemble 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.

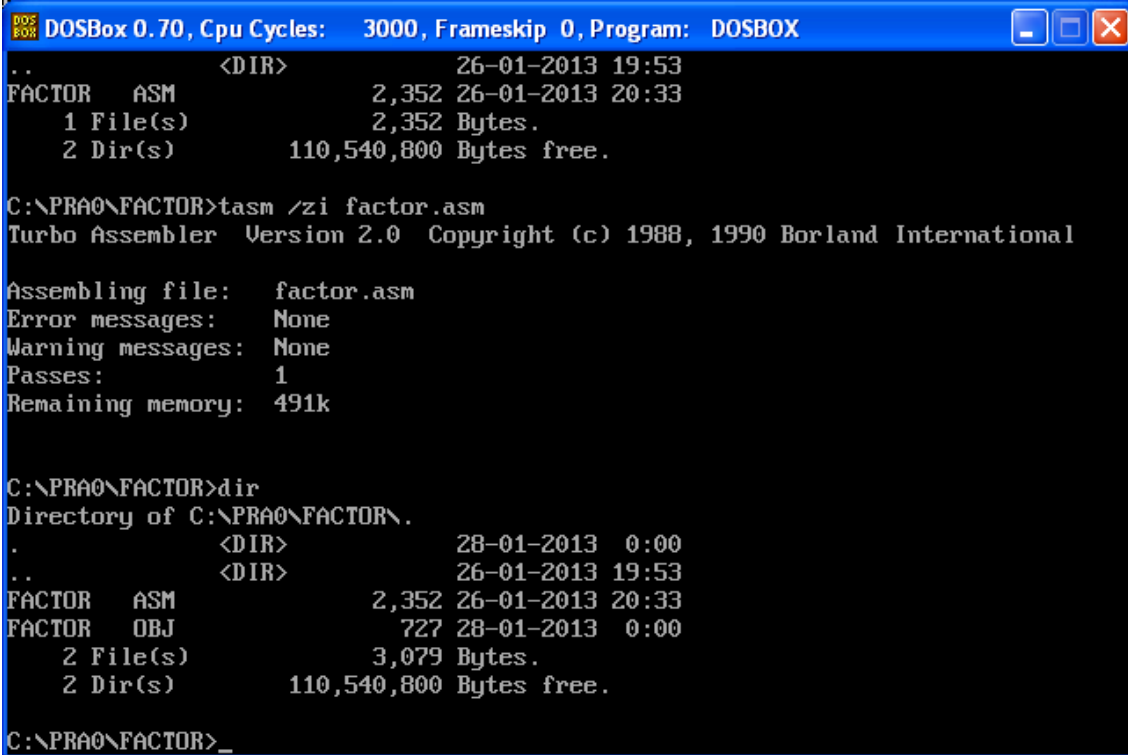
```

DOSBox 0.70, Cpu Cycles: 3000, Frameskip 0, Program: DOSBOX
Turbo Assembler Version 2.0 Copyright (c) 1988, 1990 Borland International
Syntax: TASM [options] source [,object] [,listing] [,xref]
/a,/s      Alphabetic or Source-code segment ordering
/c         Generate cross-reference in listing
/dSYM[=VAL] Define symbol SYM = 0, or = value VAL
/e,/r      Emulated or Real floating-point instructions
/h,/?      Display this help screen
/iPATH     Search PATH for include files
/jCMD      Jam in an assembler directive CMD (eg. /jIDEAL)
/kh#,/ks#  Hash table capacity #, String space capacity #
/l,/la     Generate listing: l=normal listing, la=expanded listing
/ml,/mx,/mu Case sensitivity on symbols: ml=all, mx=globals, mu=none
/mv#       Set maximum valid length for symbols
/m#        Allow # multiple passes to resolve forward references
/n         Suppress symbol tables in listing
/o,/op     Generate overlay object code, Phar Lap-style 32-bit fixups
/p         Check for code segment overrides in protected mode
/q         Suppress OBJ records not needed for linking
/t         Suppress messages if successful assembly
/w0,/w1,/w2 Set warning level: w0=none, w1=w2=warnings on
/w-xxx,/w+xxx Disable (-) or enable (+) warning xxx
/x         Include false conditionals in listing
/z         Display source line with error message
/zi,/zd    Debug info: zi=full, zd=line numbers only
C:\PRA0\FACTOR>_

```

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.



```
DOSBox 0.70, Cpu Cycles: 3000, Frameskip 0, Program: DOSBOX
..          <DIR>                26-01-2013 19:53
FACTOR  ASM                2,352 26-01-2013 20:33
  1 File(s)                2,352 Bytes.
  2 Dir(s)                110,540,800 Bytes free.

C:\PRA0\FACTOR>tasm /zi factor.asm
Turbo Assembler Version 2.0 Copyright (c) 1988, 1990 Borland International

Assembling file:  factor.asm
Error messages:   None
Warning messages: None
Passes:          1
Remaining memory: 491k

C:\PRA0\FACTOR>dir
Directory of C:\PRA0\FACTOR\..
.          <DIR>                28-01-2013  0:00
..         <DIR>                26-01-2013 19:53
FACTOR  ASM                2,352 26-01-2013 20:33
FACTOR  OBJ                 727 28-01-2013  0:00
  2 File(s)                3,079 Bytes.
  2 Dir(s)                110,540,800 Bytes free.

C:\PRA0\FACTOR>_
```

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.

```
DOSBox 0.70, Cpu Cycles: 3000, Frameskip 0, Program: DOSBOX

C:\PRA0\FACTOR>

C:\PRA0\FACTOR>tlink
Turbo Link Version 3.01 Copyright (c) 1987, 1990 Borland International
Syntax: TLINK objfiles, exefile, mapfile, libfiles
@xxxx indicates use response file xxxx
Options: /m = map file with publics
         /x = no map file at all
         /i = initialize all segments
         /l = include source line numbers
         /s = detailed map of segments
         /n = no default libraries
         /d = warn if duplicate symbols in libraries
         /c = lower case significant in symbols
         /3 = enable 32-bit processing
         /v = include full symbolic debug information
         /e = ignore Extended Dictionary
         /t = create COM file
         /o = overlay switch
         /ye = expanded memory swapping
         /yx = extended memory swapping

C:\PRA0\FACTOR>
```

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.

```
DOSBox 0.70, Cpu Cycles: 3000, Frameskip 0, Program: DOSBOX

C:\PRA0\FACTOR>dir
Directory of C:\PRA0\FACTOR\.
.                <DIR>                28-01-2013  0:00
..               <DIR>                26-01-2013 19:53
FACTOR  ASM                2,352 26-01-2013 20:33
FACTOR  OBJ                 727 28-01-2013  0:00
      2 File(s)              3,079 Bytes.
      2 Dir(s)             110,540,800 Bytes free.

C:\PRA0\FACTOR>tlink /v factor
Turbo Link Version 3.01 Copyright (c) 1987, 1990 Borland International

C:\PRA0\FACTOR>dir
Directory of C:\PRA0\FACTOR\.
.                <DIR>                28-01-2013  0:05
..               <DIR>                26-01-2013 19:53
FACTOR  ASM                2,352 26-01-2013 20:33
FACTOR  EXE                 1,262 28-01-2013  0:05
FACTOR  MAP                 267 28-01-2013  0:05
FACTOR  OBJ                 727 28-01-2013  0:00
      4 File(s)              4,608 Bytes.
      2 Dir(s)             110,540,800 Bytes free.

C:\PRA0\FACTOR>
```

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

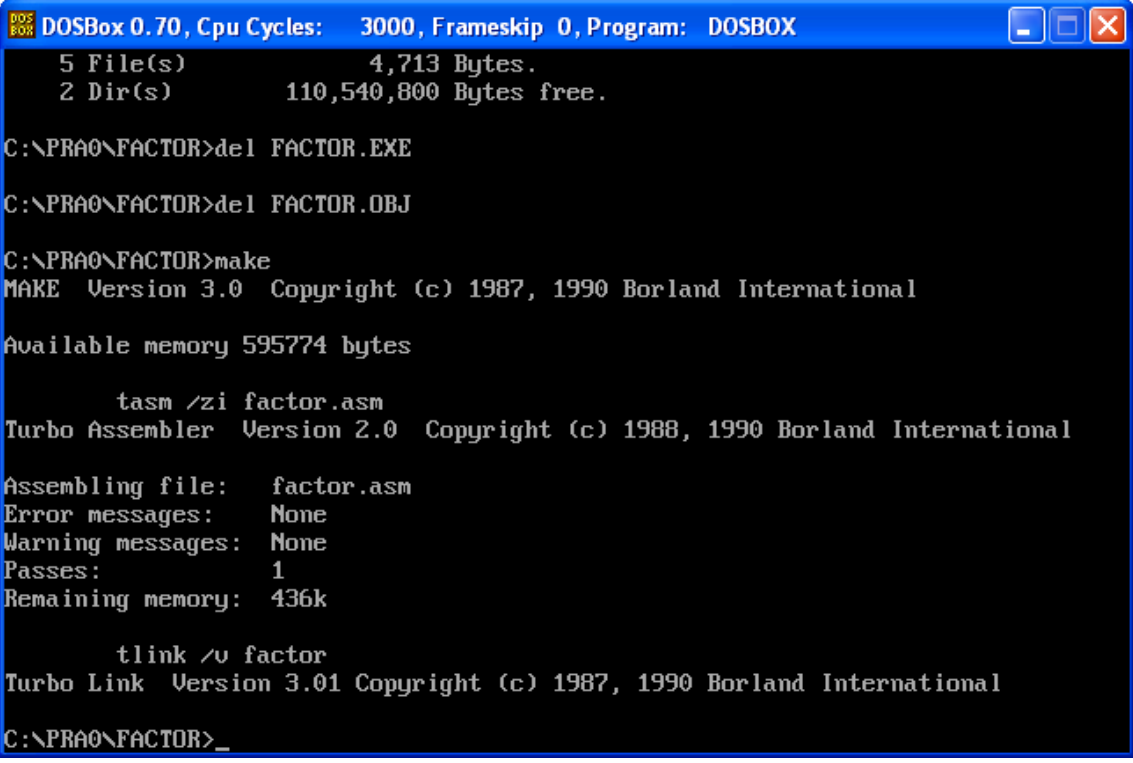
factor.exe: factor.obj

    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.



```
DOSBox 0.70, Cpu Cycles: 3000, Frameskip 0, Program: DOSBOX
5 File(s)          4,713 Bytes.
2 Dir(s)           110,540,800 Bytes free.

C:\PRA0\FACTOR>del FACTOR.EXE

C:\PRA0\FACTOR>del FACTOR.OBJ

C:\PRA0\FACTOR>make
MAKE Version 3.0 Copyright (c) 1987, 1990 Borland International

Available memory 595774 bytes

    tasm /zi factor.asm
Turbo Assembler Version 2.0 Copyright (c) 1988, 1990 Borland International

Assembling file:   factor.asm
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  436k

    tlink /v factor
Turbo Link Version 3.01 Copyright (c) 1987, 1990 Borland International

C:\PRA0\FACTOR>_
```

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.

```
DOSBox 0.70, Cpu Cycles: 3000, Frameskip 0, Program: DOSBOX

Assembling file: factor.asm
Error messages: None
Warning messages: None
Passes: 1
Remaining memory: 491k

C:\PRA0\FACTOR>tlink /v factor
Turbo Link Version 3.01 Copyright (c) 1987, 1990 Borland International

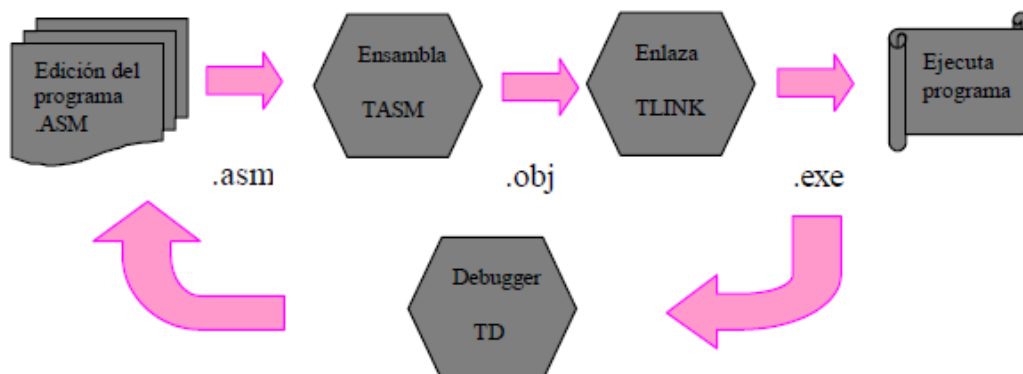
C:\PRA0\FACTOR>dir
Directory of C:\PRA0\FACTOR\..
.                <DIR>                18-01-2014  9:24
..               <DIR>                17-01-2014 23:45
FACTOR  ASM      2,352 16-10-2007 14:53
FACTOR  EXE      1,262 18-01-2014  9:24
FACTOR  MAP      267  18-01-2014  9:24
FACTOR  OBJ      727  18-01-2014  9:23
  4 File(s)            4,608 Bytes.
  2 Dir(s)           110,540,800 Bytes free.

C:\PRA0\FACTOR>factor.exe

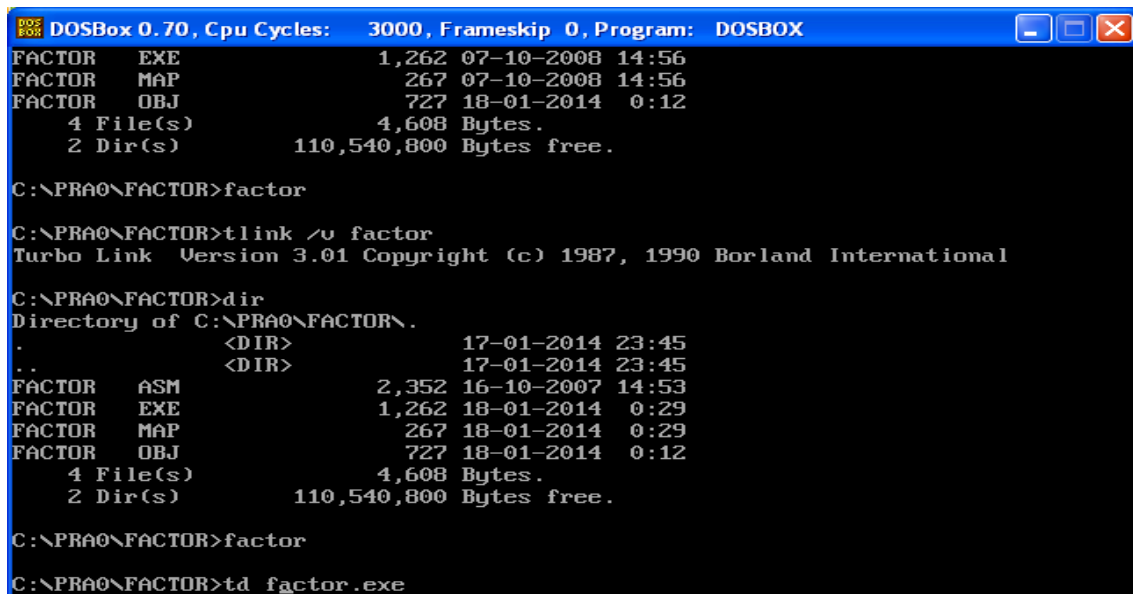
C:\PRA0\FACTOR>_
```

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.



```
DOSBox 0.70, Cpu Cycles: 3000, Frameskip 0, Program: DOSBOX
FACTOR   EXE           1,262 07-10-2008 14:56
FACTOR   MAP           267 07-10-2008 14:56
FACTOR   OBJ           727 18-01-2014 0:12
 4 File(s)              4,608 Bytes.
 2 Dir(s)             110,540,800 Bytes free.

C:\PRA0\FACTOR>factor

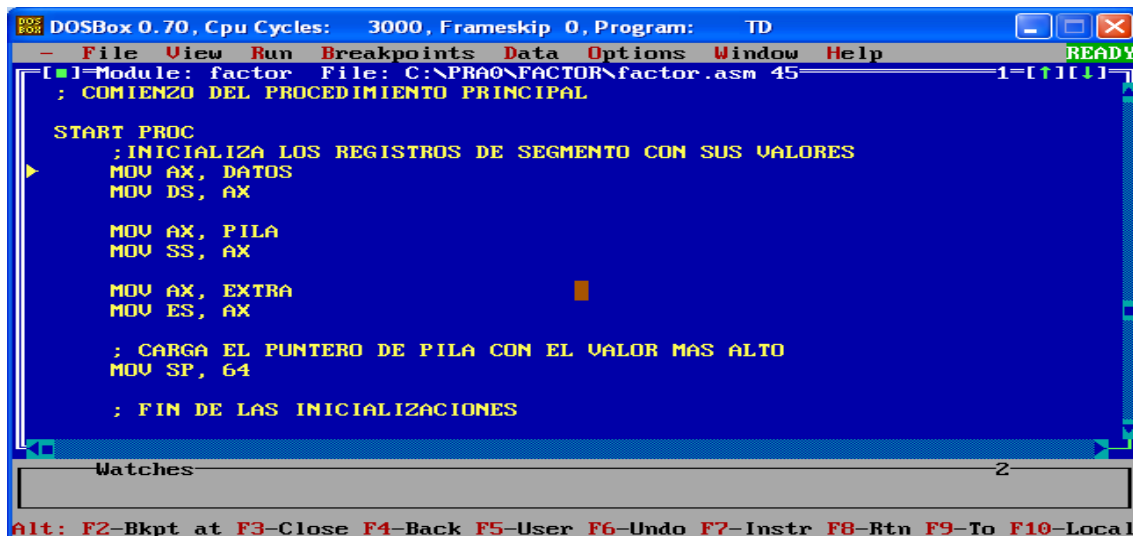
C:\PRA0\FACTOR>tlink /o factor
Turbo Link Version 3.01 Copyright (c) 1987, 1990 Borland International

C:\PRA0\FACTOR>dir
Directory of C:\PRA0\FACTOR\
.                <DIR>              17-01-2014 23:45
..               <DIR>              17-01-2014 23:45
FACTOR   ASM           2,352 16-10-2007 14:53
FACTOR   EXE           1,262 18-01-2014 0:29
FACTOR   MAP           267 18-01-2014 0:29
FACTOR   OBJ           727 18-01-2014 0:12
 4 File(s)              4,608 Bytes.
 2 Dir(s)             110,540,800 Bytes free.

C:\PRA0\FACTOR>factor

C:\PRA0\FACTOR>td factor.exe
```

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



```
DOSBox 0.70, Cpu Cycles: 3000, Frameskip 0, Program: TD
- File View Run Breakpoints Data Options Window Help
[ ] Module: factor File: C:\PRA0\FACTOR\factor.asm 45
; COMIENZO DEL PROCEDIMIENTO PRINCIPAL

START PROC
; INICIALIZA LOS REGISTROS DE SEGMENTO CON SUS VALORES
MOV AX, DATOS
MOV DS, AX

MOV AX, PILA
MOV SS, AX

MOV AX, EXTRA
MOV ES, AX

; CARGA EL PUNTERO DE PILA CON EL VALOR MAS ALTO
MOV SP, 64

; FIN DE LAS INICIALIZACIONES

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

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.

DOSBox 0.70, Cpu Cycles: 3000, Frameskip 0, Program: TD

File View Run Breakpoints Data Options Window Help

[CPU 80486]

#factor#start

cs:0002 B80A48	♦ MOV AX, DATOS	ax 0000	c=0
cs:0005 8ED8	♦ MOV DS, AX	bx 0000	z=0
cs:0007 B81148	♦ MOV AX, PILA	cx 0000	s=0
cs:000A 8ED0	♦ MOV SS, AX	dx 0000	o=0
cs:000C B80B48	♦ MOV AX, EXTRA	si 0000	p=0
cs:000F 8EC0	♦ MOV ES, AX	di 0000	a=0
cs:0011 BC4000	♦ MOV SP, 64	bp 0000	i=1
cs:0014 8A0E0000	♦ MOV CL, DATO_1	sp 0042	d=0
cs:0018 E82000	♦ CALL FACTOR	ds 47FA	
cs:001B 2EA30000	♦ MOV FACT_DATO_1, AX ; ALMACENA EL R	es 47FA	
cs:001F 8A0E0100	♦ MOV CL, DATO_2	ss 4B11	
cs:0023 E81500	♦ CALL FACTOR	cs 4B0C	
cs:0026 2E8B1E0000	♦ MOV BX, FACT_DATO_1	ip 0002	
cs:002B F7E3	♦ MUL BX ; EN AX ESTÁ EL RESULTADO DE		

ds:0000 CD 20 FF 9F 00 EA FF FF = f 0
ds:0008 AD DE E0 01 4C 15 AA 01 i i O L S -
ds:0010 4C 15 7C 02 59 0F 2A 01 L S i O Y * *
ds:0018 01 01 01 00 02 FF FF FF 000 0
ds:0020 FF FF FF FF FF FF FF FF

ss:0044 0040
ss:0042 0209
ss:0040 52FB
ss:003E 0000
ss:003C 0000

F1-Help F2-Bkpt F3-Mod F4-Here F5-Zoom F6-Next F7-Trace F8-Step F9-Run F10-Menu

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.

The screenshot shows the DOSBox 0.70 interface with the following components:

- Top Bar:** DOSBox 0.70, Cpu Cycles: 3000, Frameskip 0, Program: TD
- Menu Bar:** File, View, Run, Breakpoints, Data, Options, Window, Help
- Assembly Window:**
 - Address: CPU 80486
 - Code: #factor#start
 - Instructions:
 - cs:0002 B80A48 ♦ MOV AX, DATOS
 - cs:0005 B8ED8 ♦ MOV DS, AX
 - cs:0007 B81148 ♦ MOV AX, PILA
 - cs:000A 8ED0 ♦ MOV SS, AX
 - cs:000C B80B48 ♦ MOV AX, EXTRA
 - cs:000F 8EC0 ♦ MOV ES, AX
 - cs:0011 BC4000 ♦ MOV SP, 64
 - cs:0014 8A0E0000 ♦ MOV CL, DATO_1
 - cs:0018 E82000 ♦ CALL FACTOR
 - cs:001B 2EA30000 ♦ MOV FACT_DATO_1, AX ; ALMACENA EL R
 - cs:001F 8A0E0100 ♦ MOV CL, DATO_2
 - cs:0023 E81500 ♦ CALL FACTOR
 - cs:0026 2E8B1E0000 ♦ MOV BX, FACT_DATO_1
 - cs:002B F7E3 ♦ MUL BX ; EN AX ESTA EL RESULTADO DE
- Registers Window:**
 - ax 480A, bx 0000, cx 0000, dx 0000, si 0000, di 0000, bp 0000, sp 0042, ds 47FA, es 47FA, ss 4811, cs 480C, ip 0005
 - Other registers: c=0, z=0, s=0, o=0, p=0, a=0, i=1, d=0
- Memory Window:**
 - ds:0000 CD 20 FF 9F 00 EA FF FF = f 0
 - ds:0008 AD DE E0 01 4C 15 AA 01 iÓLŠ-@
 - ds:0010 4C 15 7C 02 59 0F 2A 01 LŠiY*~@
 - ds:0018 01 01 01 00 02 FF FF FF @@@ @
 - ds:0020 FF FF FF FF FF FF FF FF
 - ss:004A 001E, ss:0048 0009, ss:0046 0000, ss:0044 0040, ss:0042 0209
- Footer:** F1-Help F2-Bkpt F3-Mod F4-Here F5-Zoom F6-Next F7-Trace F8-Step F9-Run F10-Menu

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.

DOSBox 0.70, Cpu Cycles: 3000, Frameskip 0, Program: TD

File View Run Breakpoints Data Options Window Help

[CPU 80486]

Address	Instruction	Comment	Register/Value	Flag
cs:0002	B80A48	MOV AX, DATOS	ax 480A	c=0
cs:0005	8ED8	MOV DS, AX	bx 0000	z=0
cs:0007	B81148	MOV AX, PILA	cx 0000	s=0
cs:000A	8ED0	MOV SS, AX	dx 0000	o=0
cs:000C	B80B48	MOV AX, EXTRA	si 0000	p=0
cs:000F	8EC0	MOV ES, AX	di 0000	a=0
cs:0011	BC4000	MOV SP, 64	bp 0000	i=1
cs:0014	8A0E0000	MOV CL, DATO_1	sp 0042	d=0
cs:0018	E82000	CALL FACTOR	ds 47FA	
cs:001B	2EA30000	MOV FACT_DATO_1, AX ; ALMACENA EL R	es 47FA	
cs:001F	8A0E0100	MOV CL, DATO_2	ss 4811	
cs:0023	E81500	CALL FACTOR	cs 480C	
cs:0026	2E8B1E0000	MOV BX, FACT_DATO_1	ip 0005	
cs:002B	F7E3	MUL BX ; EN AX ESTÁ EL RESULTADO DE		

ds:0000 CD 20 FF 9F 00 EA FF FF = f 0
ds:0008 AD DE E0 01 4C 15 AA 01 iúLŠ-
ds:0010 4C 15 7C 02 59 0F 2A 01 LŠY*
ds:0018 01 01 01 00 02 FF FF FF 000 0
ds:0020 FF FF FF FF FF FF FF FF

ss:004A 001E
ss:0048 0009
ss:0046 0000
ss:0044 0040
ss:0042 0209

F1-Help F2-Bkpt F3-Mod F4-Here F5-Zoom F6-Next F7-Trace F8-Step F9-Run F10-Menu

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.

DOSBox 0.70, Cpu Cycles: 3000, Frameskip 0, Program: TD

File View Run Breakpoints Data Options Window Help

[CPU 80486]

Address	Instruction	Comment	Register/Value	Flag
cs:0002	B80A48	MOV AX, DATOS	ax 480A	c=0
cs:0005	8ED8	MOV DS, AX	bx 0000	z=0
cs:0007	B81148	MOV AX, PILA	cx 0000	s=0
cs:000A	8ED0	MOV SS, AX	dx 0000	o=0
cs:000C	B80B48	MOV AX, EXTRA	si 0000	p=0
cs:000F	8EC0	MOV ES, AX	di 0000	a=0
cs:0011	BC4000	MOV SP, 64	bp 0000	i=1
cs:0014	8A0E0000	MOV CL, DATO_1	sp 0042	d=0
cs:0018	E82000	CALL FACTOR	ds 47FA	
cs:001B	2EA30000	MOV FACT_DATO_1, AX ; ALMACENA EL R	es 47FA	
cs:001F	8A0E0100	MOV CL, DATO_2	ss 4811	
cs:0023	E81500	CALL FACTOR	cs 480C	
cs:0026	2E8B1E0000	MOV BX, FACT_DATO_1	ip 0005	
cs:002B	F7E3	MUL BX ; EN AX ESTÁ EL RESULTADO DE		

ds:0000 CD 20 FF 9F 00 EA FF FF = f 0
ds:0008 AD DE E0 01 4C 15 AA 01 iúLŠ-
ds:0010 4C 15 7C 02 59 0F 2A 01 LŠY*
ds:0018 01 01 01 00 02 FF FF FF 000 0
ds:0020 FF FF FF FF FF FF FF FF

ss:004A 001E
ss:0048 0009
ss:0046 0000
ss:0044 0040
ss:0042 0209

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

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.

The screenshot shows the DOSBox 0.70 interface with the following components:

- Top Bar:** DOSBox 0.70, Cpu Cycles: 3000, Frameskip 0, Program: TD. Buttons: File, View, Run, Breakpoints, Data, Options, Window, Help. Status: READY.
- Assembly Window (Left):**

```

[ ] CPU 80486
#factor#start
cs:0002 B80A4B  ♦ MOV AX, DATOS
cs:0005 8ED8     ♦ MOV DS, AX
cs:0007 B8114B   ♦ MOV AX, PILA
cs:000A 8ED0     ♦ MOV SS, AX
cs:000C B80B4B   ♦ MOV AX, EXTRA
cs:000F 8EC0     ♦ MOV ES, AX
cs:0011 BC4000   ♦ MOV SP, 64
cs:0014 BA0E0000 ♦ MOV CL, DATO_1
cs:0018 E82000   ♦ CALL FACTOR
cs:001B 2EA30000 ♦ MOV FACT_DATO_1, AX ; ALMACENA EL R
cs:001F BA0E0100 ♦ MOV CL, DATO_2
cs:0023 E81500   ♦ CALL FACTOR
cs:0026 2E8B1E0000 ♦ MOV BX, FACT_DATO_1
cs:002B F7E3     ♦ MUL BX ; EN AX ESTÁ EL RESULTADO DE

```
- Registers Window (Right):**

```

ax 0000  c=0
bx 0000  z=0
cx 0000  s=0
dx 0000  o=0
si 0000  p=0
di 0000  a=0
bp 0000  i=1
sp 0042  d=0
ds 47FA
es 47FA
ss 4B11
cs 4B0C
ip 0002

```
- Memory Window (Bottom):**

```

ds:0000 CD 20 FF 9F 00 EA FF FF = f 0
ds:0008 AD DE E0 01 4C 15 AA 01 ¡ÏÛLŠ-
ds:0010 4C 15 7C 02 59 0F 2A 01 LŠÏY*-
ds:0018 01 01 01 00 02 FF FF FF 000 0
ds:0020 FF FF FF FF FF FF FF FF

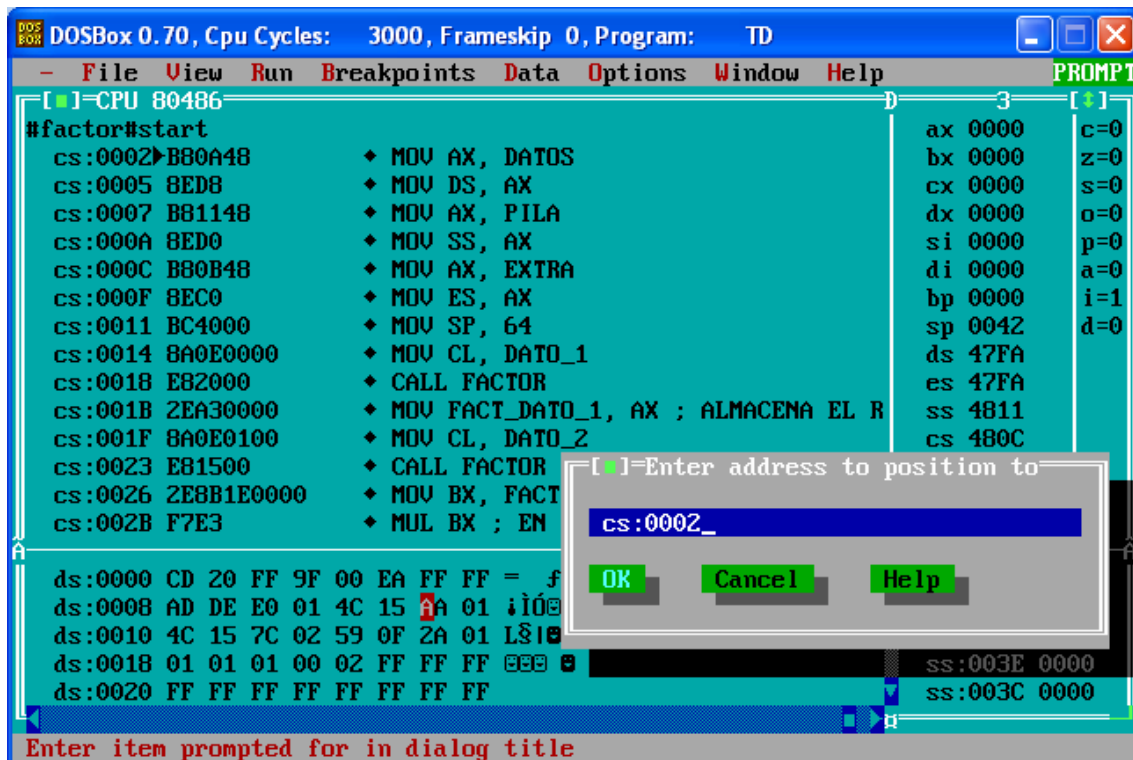
```
- Stack Window (Bottom Right):**

```

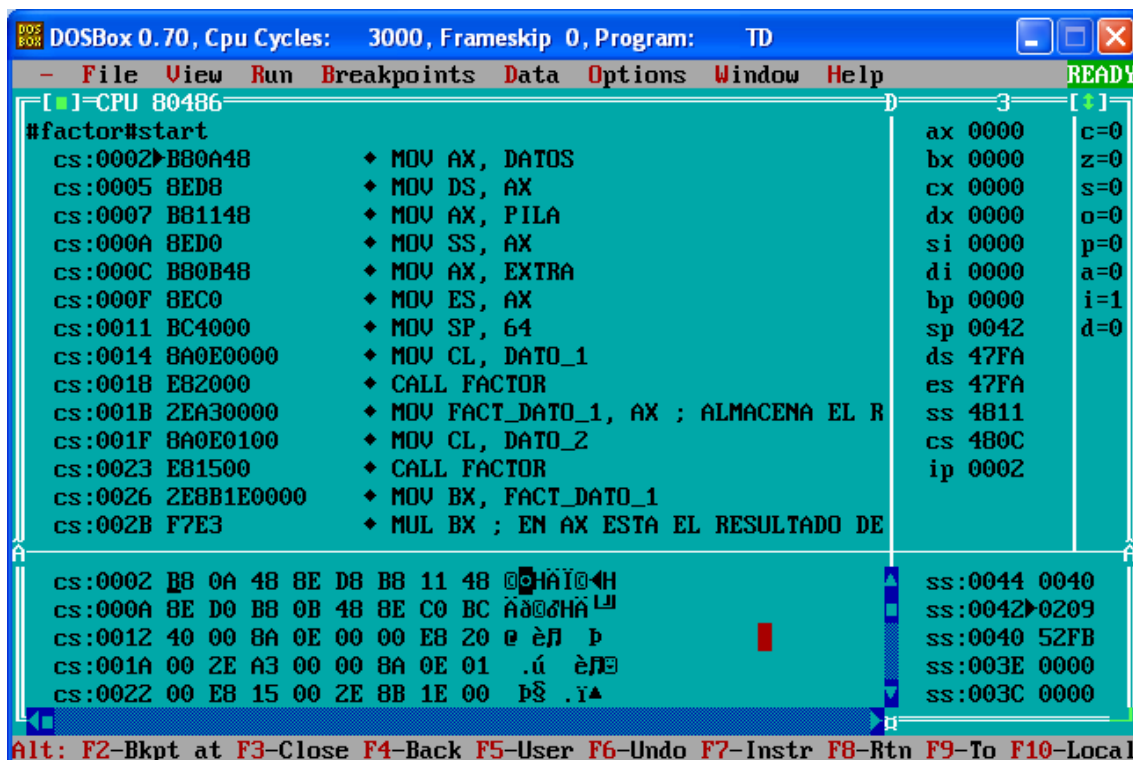
ss:0044 0040
ss:0042 0209
ss:0040 52FB
ss:003E 0000
ss:003C 0000

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

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
```

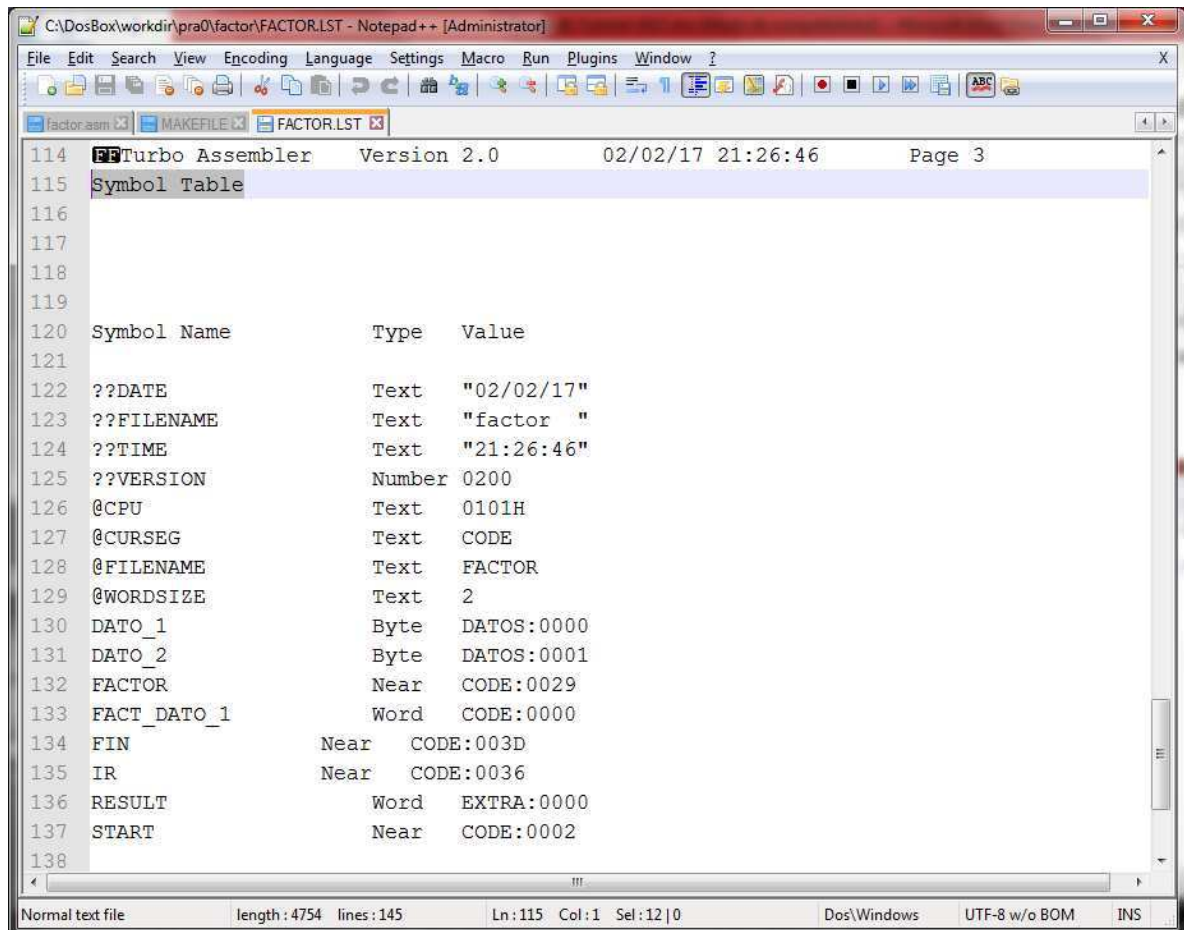
by

```
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.



```
114 FTurbo Assembler Version 2.0 02/02/17 21:26:46 Page 3
115 Symbol Table
116
117
118
119
120 Symbol Name      Type  Value
121
122 ??DATE            Text  "02/02/17"
123 ??FILENAME        Text  "factor "
124 ??TIME            Text  "21:26:46"
125 ??VERSION         Number 0200
126 @CPU              Text  0101H
127 @CURSEG           Text  CODE
128 @FILENAME          Text  FACTOR
129 @WORDSIZE          Text  2
130 DATO_1             Byte  DATOS:0000
131 DATO_2             Byte  DATOS:0001
132 FACTOR             Near  CODE:0029
133 FACT_DATO_1        Word  CODE:0000
134 FIN                Near  CODE:003D
135 IR                 Near  CODE:0036
136 RESULT             Word  EXTRA:0000
137 START              Near  CODE:0002
138
```

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

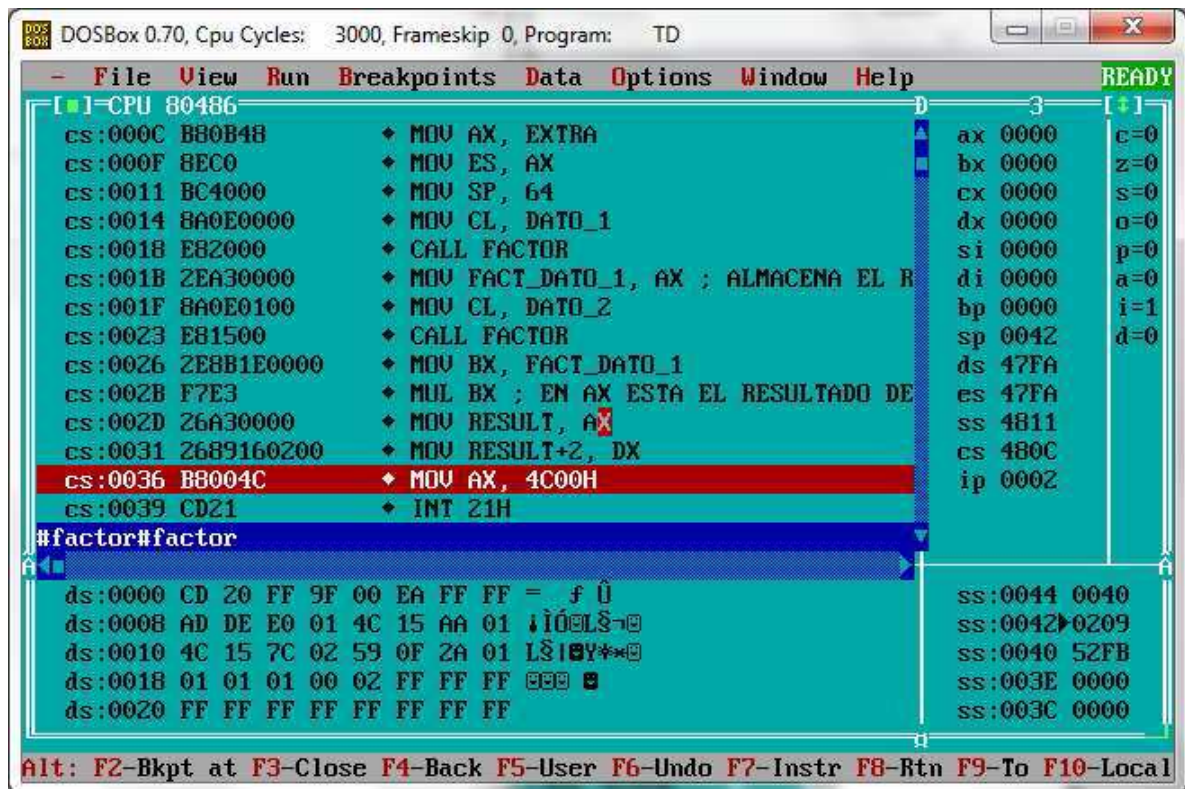
FACT_DATO_1: CODE segment, offset 0

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

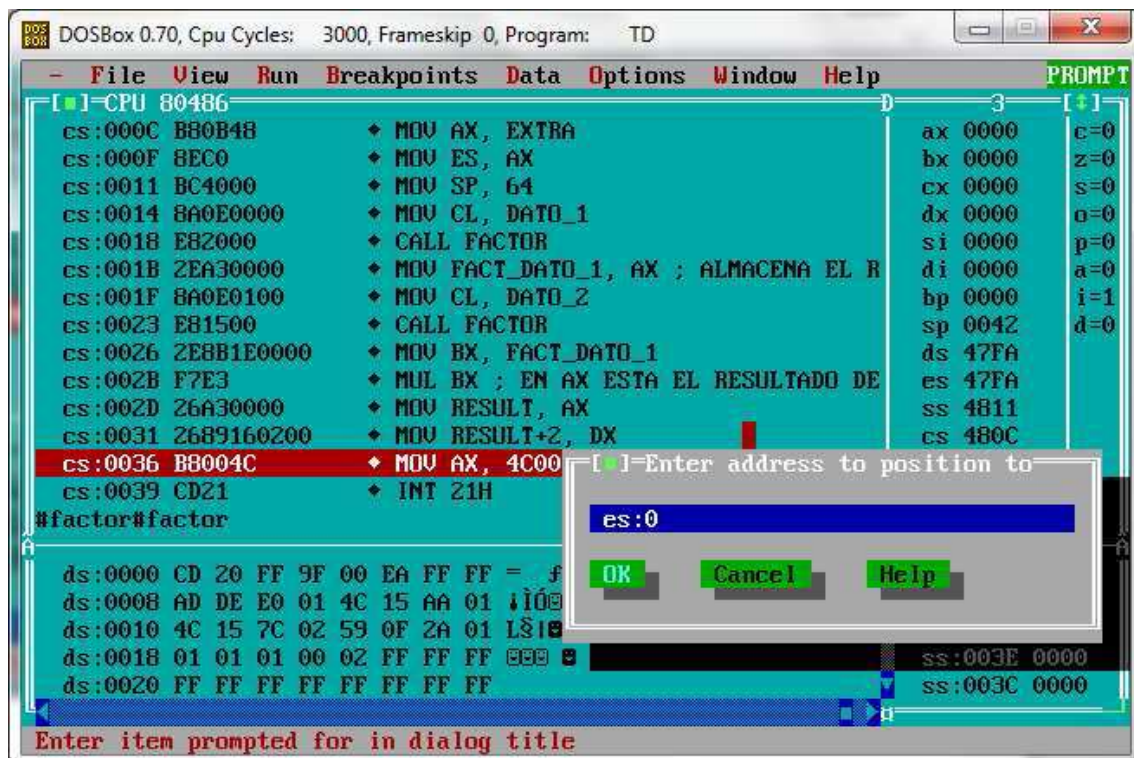
To do this, select the instruction with the cursor and the press F2. Move the cursor upwards to check that the breakpoint has been activated, (indicated with a red line)



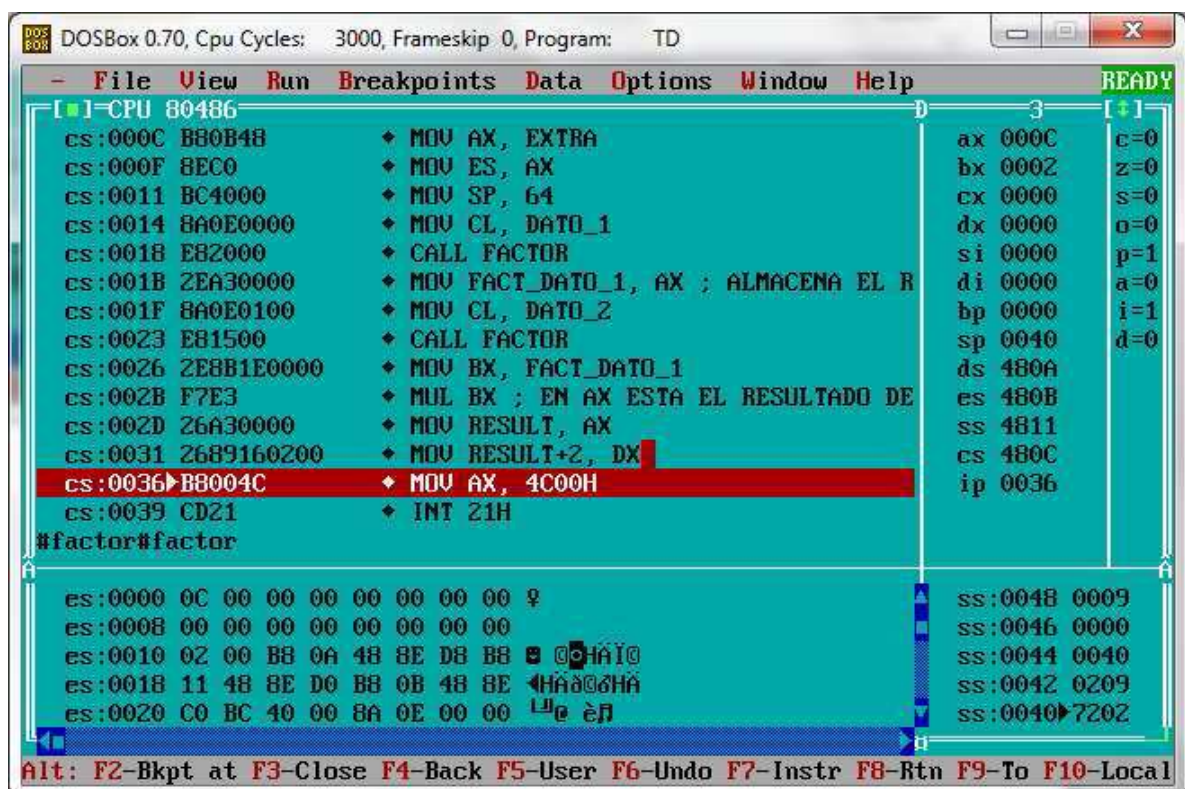
Then execute the program pressing F9 key. The program will then be executed and stopped at the breakpoint. In this moment, the RESULT variable should contain its final value.

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

Remember that EXTRA segment is associated to 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. $4! \times 5! =$
2. $8! =$
3. $9! =$
4. $8! \times 7! =$

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. $(2 \times 3)! =$
2. $(2 \times 4)! =$
3. $(3 \times 3)! =$
4. $(2 \times 7)! =$

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 .