

Laboratorio de Sistemas Basados en Microprocesadores

Práctica 0a: Tutorial del entorno de desarrollo/depuración del 80x86

En esta práctica se realizará un tutorial con objetivo de adquirir experiencia práctica con el entorno de desarrollo del 80x86. El alumno deberá seguir las instrucciones del presente guión y realizar los ejercicios prácticos propuestos en el mismo. La duración de la práctica es de 1 sesión.

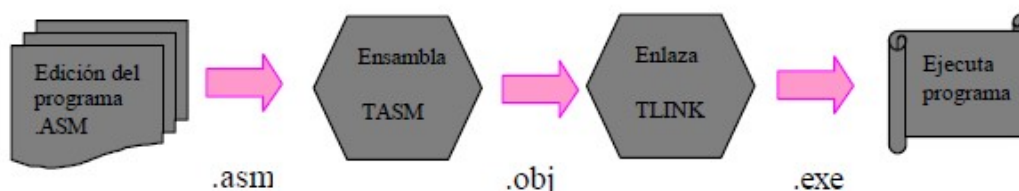
Desarrollo de Programas en Lenguaje Ensamblador

En general, el ciclo de vida de un programa se compone de las siguientes etapas:

1. Especificación del problema
2. Diseño de diagramas de flujo, etc., orientados al lenguaje objetivo
3. Edición del programa fuente en lenguaje ensamblador
4. Ensamblado y Enlazado
5. Ejecución
6. Depuración

Para poder crear un programa en lenguaje ensamblador que sea ejecutable en un PC (arquitectura 80x86, sistema operativo MS-DOS) se requieren varias herramientas:

1. Sistema operativo MS-DOS
2. Un editor para crear el programa fuente (.asm)
3. Un ensamblador (también llamado compilador) que "traduce" el programa fuente a un programa objeto en lenguaje máquina (.obj)
4. Un enlazador (*linker*), que genera el programa ejecutable (.exe) a partir del programa objeto creado en el paso anterior



Sistema Operativo MS-DOS

Dado que los equipos disponibles en el laboratorio no tienen MS-DOS como sistema operativo nativo, utilizaremos el programa **DOSBox**. DOSBox es un emulador que recrea un entorno

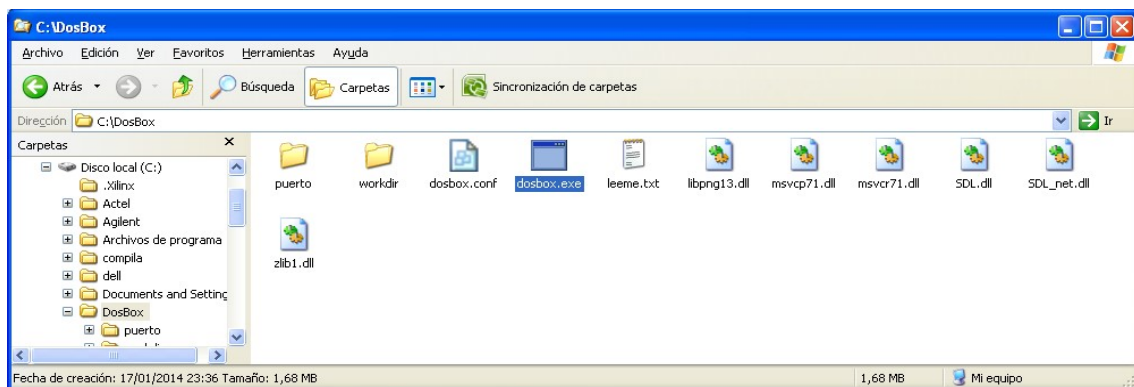
similar al sistema operativo MS-DOS con el objetivo de poder ejecutar programas originalmente escritos para MS-DOS en ordenadores más modernos o en diferentes arquitecturas.

Si utilizamos los ordenadores del laboratorio, como primer paso debemos abrir un Explorador de Windows e ir a la carpeta donde está instalado el DosBox en el disco C (C:\Program Files (x86)\DosBox), y debemos copiar la carpeta DosBox completa, en el raíz del disco C.

Si utilizamos el zip de Moodle, deberemos descomprimir el directorio en el raíz del disco C.

La carpeta Workdir será nuestro directorio de trabajo durante todo el curso, y contiene todos los programas necesarios para ensamblado, enlazado y depuración, así como los programas fuente que utilizaremos durante el desarrollo de este tutorial (contenidos dentro del subdirectorio pra0).

Una vez copiado el directorio Dosbox al raíz del disco C, ya podemos ejecutar el programa DosBox, que se encuentra en C:\DosBox\DosBox.exe. Podemos hacerlo directamente desde el explorador de Windows haciendo doble click en el icono DosBox.exe.



Tras ejecutar DosBox, se abre una ventana de comandos como la de la figura siguiente. Esta ventana será la que utilizaremos para ensamblar, enlazar y ejecutar los programas que desarrollaremos. Tecleando el comando “help”, aparecen todos los comandos soportados dentro de DosBox. Los que utilizaremos más a menudo son “cd” para cambiar de directorio y “dir” para listar los contenidos del directorio actual, así como “del” para borrar un fichero.

```
DOSBox 0.70, Cpu Cycles: 3000, Frameskip 0, Program: DOSBOX
For more information read the README file in the DOSBox directory.

HAVE FUN!
The DOSBox Team

Drive C is mounted as local directory c:\DosBox\workdir\
Keyboard layout sp loaded for codepage 858
C:\>help
If you want a list of all supported commands type help /all .
A short list of the most often used commands:
<CD      > Displays/changes the current directory.
<CLS     > Clear screen.
<COPY    > Copy files.
<DIR     > Directory View.
<DEL     > Removes one or more files.
<EXIT    > Exit from the shell.
<MD      > Make Directory.
<RD      > Remove Directory.
<TYPE    > Display the contents of a text-file.
<REN     > Renames one or more files.
<LOADHIGH> Loads a program into upper memory (requires xms=true,umb=true).
<CHOICE  > Waits for a keypress and sets ERRORLEVEL.
<VER     > View and set the reported DOS version.

C:\>_
```

El programa DosBox está configurado en los equipos del laboratorio para montar la carpeta C:\DosBox\Workdir como disco virtual C. Esto significa que una vez dentro de la ventana de DosBox, el disco C:\ será una imagen del disco C:\DosBox\Workdir del PC del laboratorio. De este modo, si tecleamos el comando “dir”, nos debe aparecer el contenido de nuestro directorio de trabajo C:\DosBox\Workdir, que veremos dentro de DosBox como disco C:\.

```
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

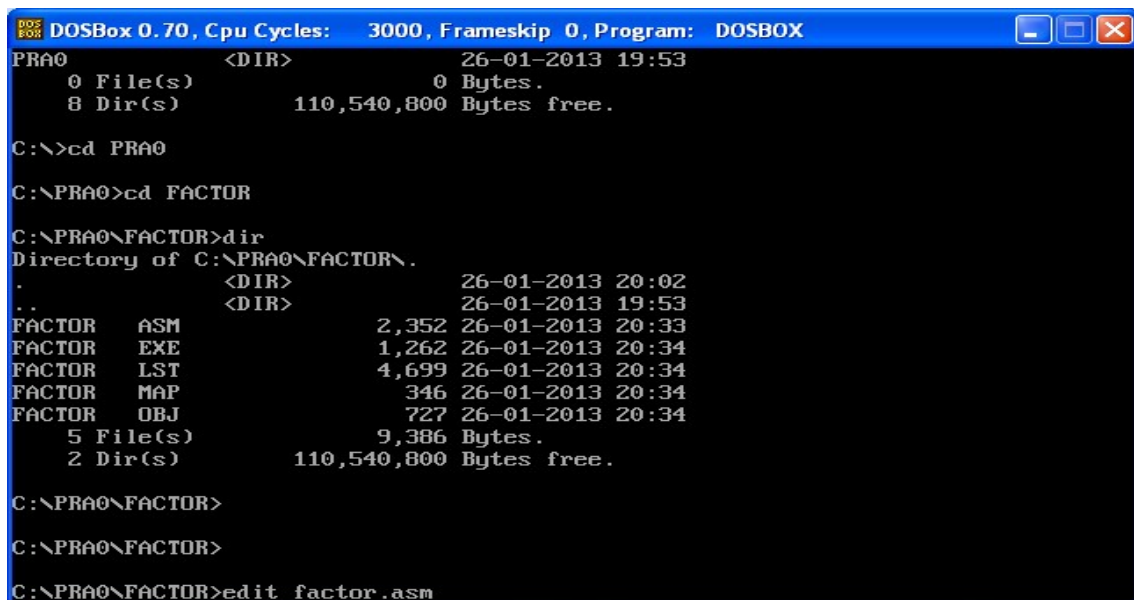
Drive C is mounted as local directory c:\workdir\
Keyboard layout sp loaded for codepage 858
C:\>dir
Directory of C:\.
.                <DIR>                17-01-2014 23:46
..               <DIR>                01-01-1980  0:00
BIN              <DIR>                17-01-2014 23:45
COMPILA         <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:\>_
```

Edición de Programas Fuente

Podemos utilizar cualquier editor de texto ASCII para editar el código del programa fuente en lenguaje ensamblador. Para que pueda ser procesado correctamente por el programa ensamblador, el fichero fuente del programa debe tener la extensión .ASM y el nombre del fichero debe tener ocho caracteres como máximo.

En el caso de querer utilizar un editor interno desde el DosBox, podemos utilizar EDIT como programa editor de texto. Para ver cómo funciona EDIT, nos cambiamos al directorio pra0 mediante el comando "cd pra0". Una vez dentro de pra0, entramos en el subdirectorio "factor" mediante el comando "cd factor" y a continuación tecleamos "edit factor.asm" para editar el fichero fuente factor.asm que se encuentra allí. Todo este proceso lo podemos ver en la siguiente figura.



```
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>

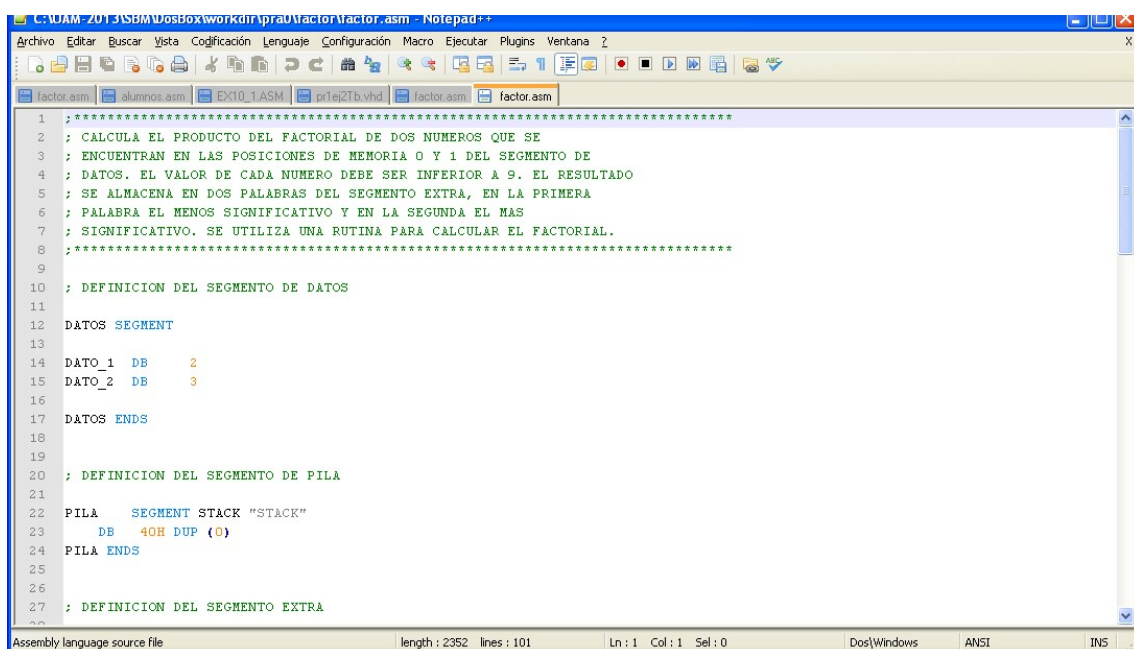
C:\PRA0\FACTOR>edit factor.asm
```

A continuación, aparecerá la ventana de edición del editor de texto EDIT. Pulsando ALT se accede al a la barra del menú superior. Para desplazarse por la barra se emplean las teclas de dirección.



```
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
```

Como alternativa al editor interno EDIT, podemos usar un editor externo a DosBox, por ejemplo, el bloc de notas o el editor Notepad++ (este último tiene la importante ventaja del resaltado de sintaxis). Para ello simplemente abrimos el fichero correspondiente con el editor elegido, que en nuestro caso sería el fichero C:\DosBox\Workdir\pra0\factor\factor.asm.

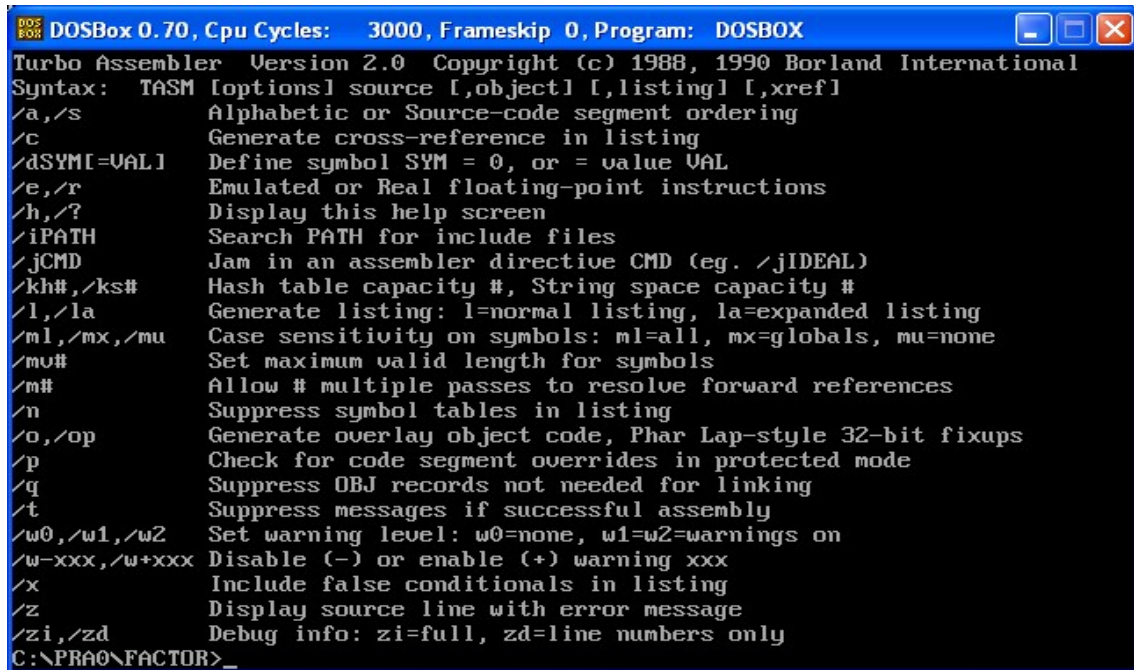


```
C:\WAM-2013\BMDosBox\workdir\pra0\factor\factor.asm - Notepad++
Archivo Editar Buscar Vista Codificación Lenguaje Configuración Macro Ejecutar Plugins Ventana ?
factor.asm alumnos.asm EX10_1.ASM pr1ej2Tb.vhd factor.asm factor.asm
1 *****
2 ; CALCULA EL PRODUCTO DEL FACTORIAL DE DOS NUMEROS QUE SE
3 ; ENCUENTRAN 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
Assembly language source file length: 2352 lines: 101 Ln: 1 Col: 1 Sel: 0 Dos\Windows ANSI INS
```

Ensamblado de Programas

Para el ensamblado de programas utilizaremos dentro del DosBox el programa Turbo Assembler (TASM), que generará un fichero objeto (extensión .OBJ) a partir de un fichero fuente (extensión .ASM). El fichero OBJ es un fichero intermedio, llamado así porque aún no es

un programa ejecutable pero tampoco es ya un programa en lenguaje fuente, sino en lenguaje máquina. Podemos ver las distintas opciones de que dispone el TASM tecleando “tasm” dentro de DosBox.

A screenshot of a DOSBox 0.70 window. The title bar reads "DOSBox 0.70, Cpu Cycles: 3000, Frameskip 0, Program: DOSBOX". The window contains the Turbo Assembler version 2.0 help text. The text is as follows:
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>_

En la siguiente figura, podemos ver el proceso de ensamblado del programa factor.asm. Para ello debemos teclear “tasm /zi factor.asm”. Hemos incluido la opción /zi para obtener información completa de depuración (ver figura anterior). Podemos comprobar que se ha generado el fichero objeto factor.obj si tecleamos el comando “dir”.

NOTA: Es posible que ya exista previamente el fichero factor.obj antes de ejecutar el tasm. En ese caso, podemos borrarlo tecleando “del factor.obj” antes de ejecutar el proceso de ensamblado, aunque tampoco es necesario ya que se sobrescribe, lo cual podemos comprobar observando la fecha del fichero mediante “dir”.


```
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>_
```

Enlazado de Programas

Para el enlazado de programas utilizaremos dentro del DosBox el programa Turbo Link (TLINK), que generará un fichero ejecutable (extensión .EXE) a partir del fichero objeto .OBJ generado en el paso anterior. En general, TLINK genera un fichero ejecutable partiendo de uno o varios ficheros objeto. Las distintas opciones disponibles en el TLINK aparecen tecleando "tlink" dentro de DosBox.

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

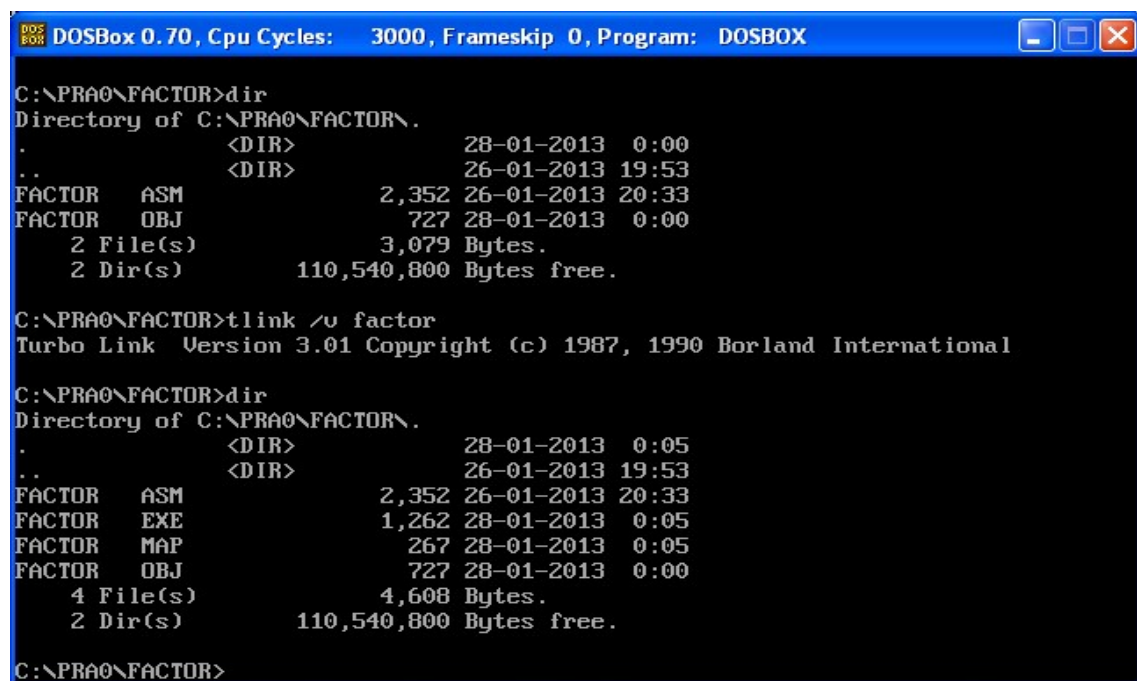
C:\PRA0\FACTOR>

C:\PRA0\FACTOR>tlink
Turbo Link Version 3.01 Copyright (c) 1987, 1990 Borland International
Syntax:  TLINK objfiles, exefile, mapfile, libfiles
@xxxxx indicates use response file xxxxx
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>
```

En la siguiente figura, podemos ver el proceso de generación del programa ejecutable a partir del fichero objeto. Para ello debemos teclear “tlink /v factor”. Hemos usado la opción /v para incluir información completa de depuración en el ejecutable (ver figura anterior). Podemos comprobar que se ha generado el fichero ejecutable factor.exe tecleando el comando “dir”.

NOTA: Es posible que ya exista previamente el fichero factor.exe antes de ejecutar el tlink. En ese caso, podemos borrarlo tecleando “del factor.exe” antes de ejecutar el proceso de enlazado, aunque tampoco es necesario ya que se sobrescribe, lo cual podemos comprobar observando la fecha del fichero mediante “dir”.



```
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>
```

Utilización de Ficheros Makefile

Los dos pasos anteriores, ensamblado y enlazado, podemos automatizarlos mediante el uso de ficheros “makefile”. Para ello, debemos crear un fichero dentro el directorio donde tengamos el fichero fuente, en nuestro caso el directorio “factor”. Ponemos el nombre “makefile” a este fichero y lo editamos con el bloc de notas o con el Notepad++. El contenido del fichero debe ser el siguiente (respetando los tabuladores):

```
all: factor.exe

factor.exe: factor.obj

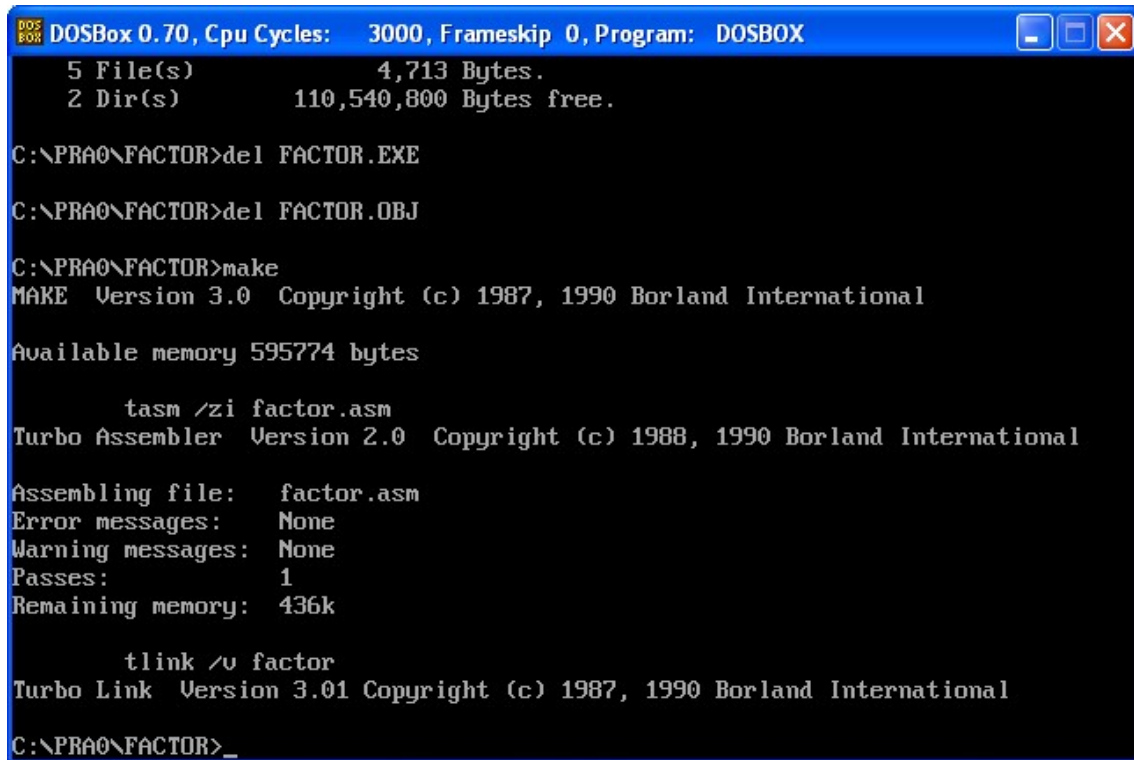
        tlink /v factor

factor.obj: factor.asm
```



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

Para comprobar el correcto funcionamiento del fichero makefile creado, borramos los ficheros factor.exe y factor.obj utilizando el comando “del”, y a continuación tecleamos “make”. Debe ejecutarse automáticamente el ensamblado y el enlazado, generando de nuevo los ficheros recién borrados.



```
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 /u factor
Turbo Link Version 3.01 Copyright (c) 1987, 1990 Borland International

C:\PRA0\FACTOR>_
```

Ejecución de Programas

Podemos ejecutar directamente escribiendo el nombre del programa correspondiente dentro de DosBox. En algunos casos, incluyendo el que nos ocupa, la ejecución del programa no es demasiado útil porque es posible que el programa en sí no saque información alguna por pantalla.

```
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

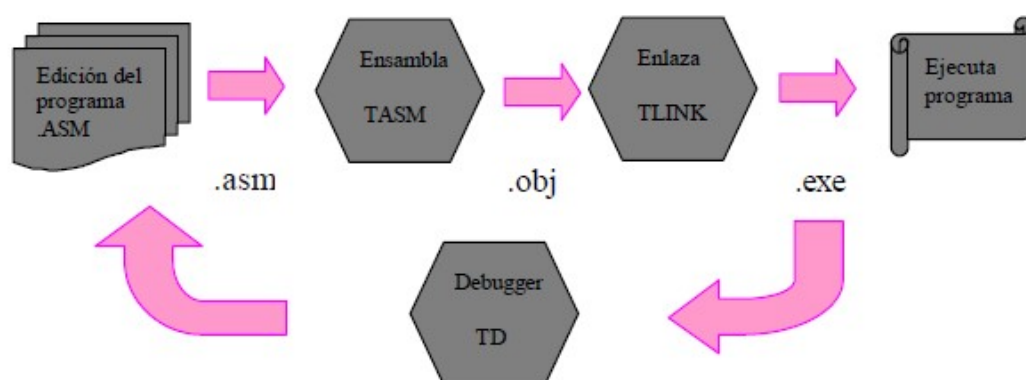
C:\PRA0\FACTOR>dir
Directory of C:\PRA0\FACTOR\
.                <DIR>                18-01-2014  9:24
..               <DIR>                17-01-2014 23:45
FACTOR  ASM             2,352 16-10-2007 14:53
FACTOR  EXE             1,262 18-01-2014  9:24
FACTOR  MAP              267 18-01-2014  9:24
FACTOR  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>_
```

Depuración de Programas

El depurador (debugger) es una herramienta que permite observar el funcionamiento de un programa ejecutable paso a paso. Usaremos el programa Turbo Debugger (en adelante TD) como depurador.



Con el objetivo de observar el funcionamiento paso a paso del programa factor, ejecutamos el programa TD tecleando "td factor.exe" dentro de 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 /u 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
```

Una vez iniciado el TD, se nos abrirá una ventana como la de la figura siguiente:

```
DOSBox 0.70, Cpu Cycles: 3000, Frameskip 0, Program: TD
- File View Run Breakpoints Data Options Window Help
[1]-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

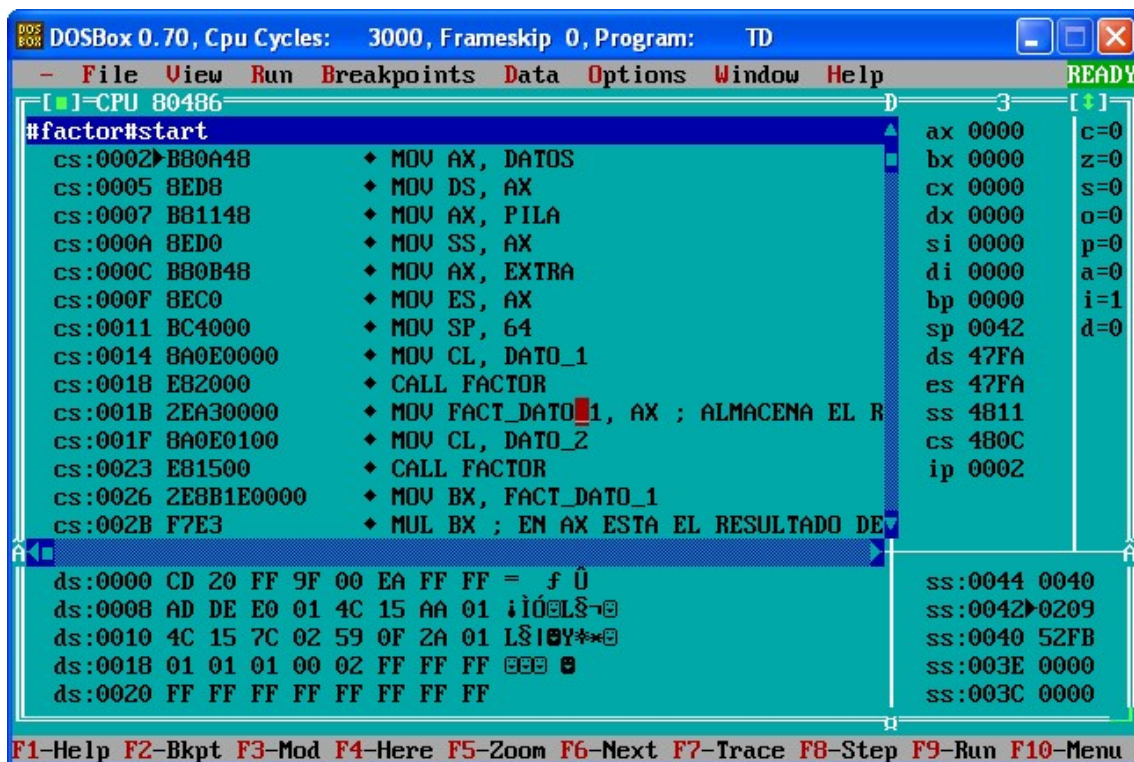
; CARGA EL PUNTERO DE PILA CON EL VALOR MAS ALTO
MOV SP, 64

; FIN DE LAS INICIALIZACIONES

Watches 2

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

Normalmente no trabajaremos con esta vista del TD, sino con otra vista llamada "CPU". Para cambiar la vista a "CPU", presionamos Alt-V y a continuación la tecla C. A continuación maximizamos la ventana que se ha abierto pulsando F5 y obtenemos lo siguiente:



Podemos observar dos barras de color gris, una en la parte superior y otra en la inferior, y una zona principal con 5 ventanas.

La barra superior es una barra que contiene diferentes menús de usuario. Podemos acceder a los diferentes menús pulsando la tecla Alt y la inicial del menú correspondiente, resaltada en color rojo. Una vez dentro de un menú, podemos usar los cursores para desplazarnos arriba y abajo por el mismo y ejecutar la opción deseada pulsando Enter, o alternatively pulsando la letra correspondiente resaltada en rojo. Para salir del menú pulsamos la tecla Esc.

La barra de la parte inferior incluye opciones de funciones rápidas a través de las teclas F1 a F10. Por ejemplo, si queremos ejecutar el programa completo pulsamos la tecla F9 (Run).

En la zona principal tenemos 5 ventanas. Siempre tendremos una sola ventana activa de las 5, que podemos seleccionar pulsando la tecla Tab. Pulsando Tab repetidas veces, la ventana activa va cambiando en el sentido de las agujas del reloj.

La mayor de las 5 ventanas, situada arriba a la izquierda, es la que está seleccionada por defecto. Esta ventana contiene las instrucciones desensambladas del programa.

Observemos detenidamente la primera línea:

cs:0002 ► B80A48 MOV AX, DATOS

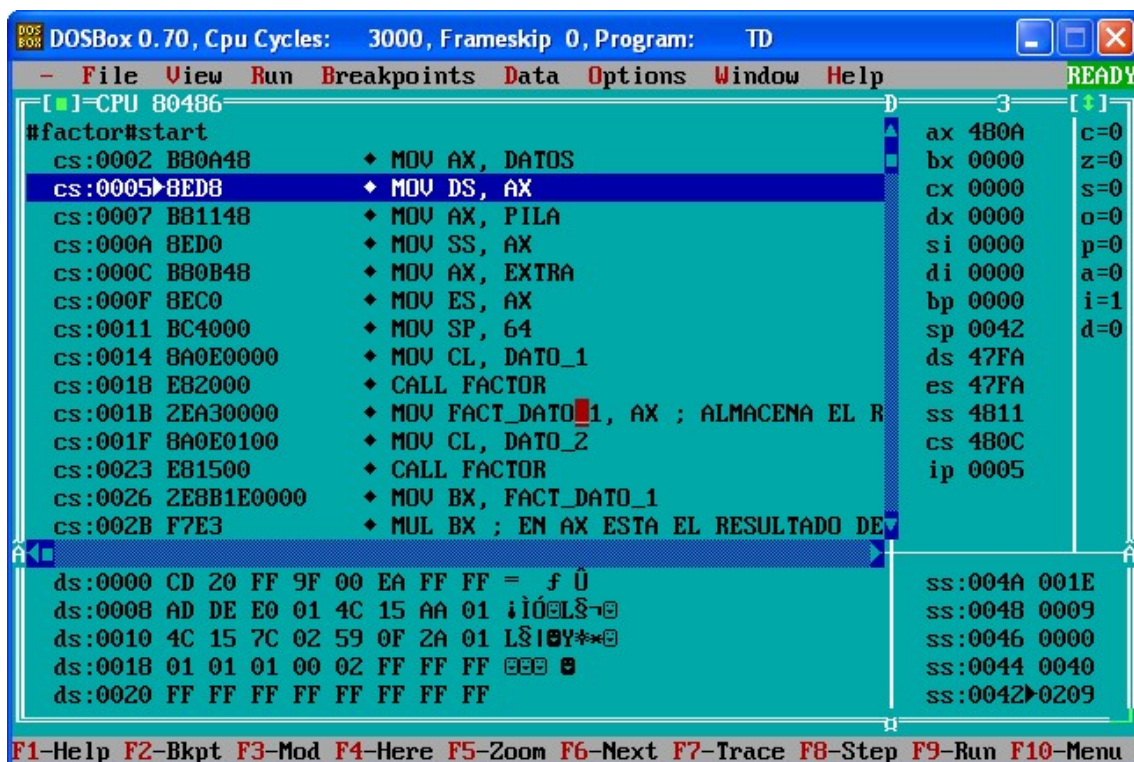
En general, cada línea contiene 3 campos diferenciados:

1. la dirección en memoria de la línea de código, en este caso cs:0002

2. el código de instrucción en hexadecimal, en este caso la instrucción ocupa 3 bytes
3. la instrucción correspondiente en código ensamblador

Por lo tanto, la posición 0002 del segmento de código cs contiene la instrucción “B80A48” de longitud 3 bytes y se corresponde con la instrucción “mov ax, datos”.

La flecha ► indica que es la siguiente instrucción a ejecutar. Si pulsamos la tecla F7 (Trace), ejecutamos la primera instrucción y vemos como la flecha pasa a la segunda instrucción. En general, iremos pulsando F7 varias veces y observando cómo se van ejecutando las distintas instrucciones del programa y los cambios que se van produciendo en las otras 4 ventanas debido a la ejecución de las mismas.



Presionando Alt+F10 (o el botón derecho del ratón) accedemos al menú propio de la ventana activa. En este caso, una vez dentro del menú propio de esta venta, pulsando M, podemos cambiar el modo “Mixed”. Si lo cambiamos a “Yes”, vemos las instrucciones tal como están en el fichero fuente y su traducción a código objeto. El modo que se suele utilizar es el “Both”. Para salir del menú propio de la ventana, pulsamos Esc.

La segunda ventana, a la derecha de la primera, muestra el contenido en hexadecimal de los registros del microprocesador 80x86. Podemos ver cómo van cambiando los diferentes registros cada vez que ejecutamos una instrucción. En particular, vemos como el registro ip contiene siempre la dirección de la siguiente instrucción a ejecutar.

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
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 ESTÁ EL RESULTADO DE
```

ax	480A	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	4811	
cs	480C	
ip	0005	

```
ds:0000 CD 20 FF 9F 00 EA FF FF = f 0
ds:0008 AD DE E0 01 4C 15 AA 01 ¡ÍÓLŠ-
ds:0010 4C 15 7C 02 59 0F 2A 01 LŠ!ØY*
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

La tercera ventana, a la derecha de la anterior, muestra el contenido del registro de estado (flags). Cada flag del registro se muestra con la inicial correspondiente (carry, zero, sign, overflow, parity, BCD carry, interrupt y direction) y su valor en binario. Al igual que los registros, los valores de estos flags van cambiando a medida que vamos ejecutando las diferentes instrucciones.

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 480A	c=0
cs:0005 B8ED8	♦ 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 ESTA 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 i0L\$-0

ds:0010 4C 15 7C 02 59 0F 2A 01 L\$10V*-0

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

La cuarta ventana muestra el contenido de la pila (*stack*) utilizada por el programa. La flecha indica la posición actual del puntero de pila (*stack pointer*).

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 4811	
cs:0023 E81500	♦ CALL FACTOR	cs 480C	
cs:0026 2E8B1E0000	♦ MOV BX, FACT_DATO_1	ip 0002	
cs:002B F7E3	♦ MUL BX ; EN AX ESTA 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 ¡ÍÓEL\$-

ds:0010 4C 15 7C 02 59 0F 2A 01 L\$ÍØY**

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

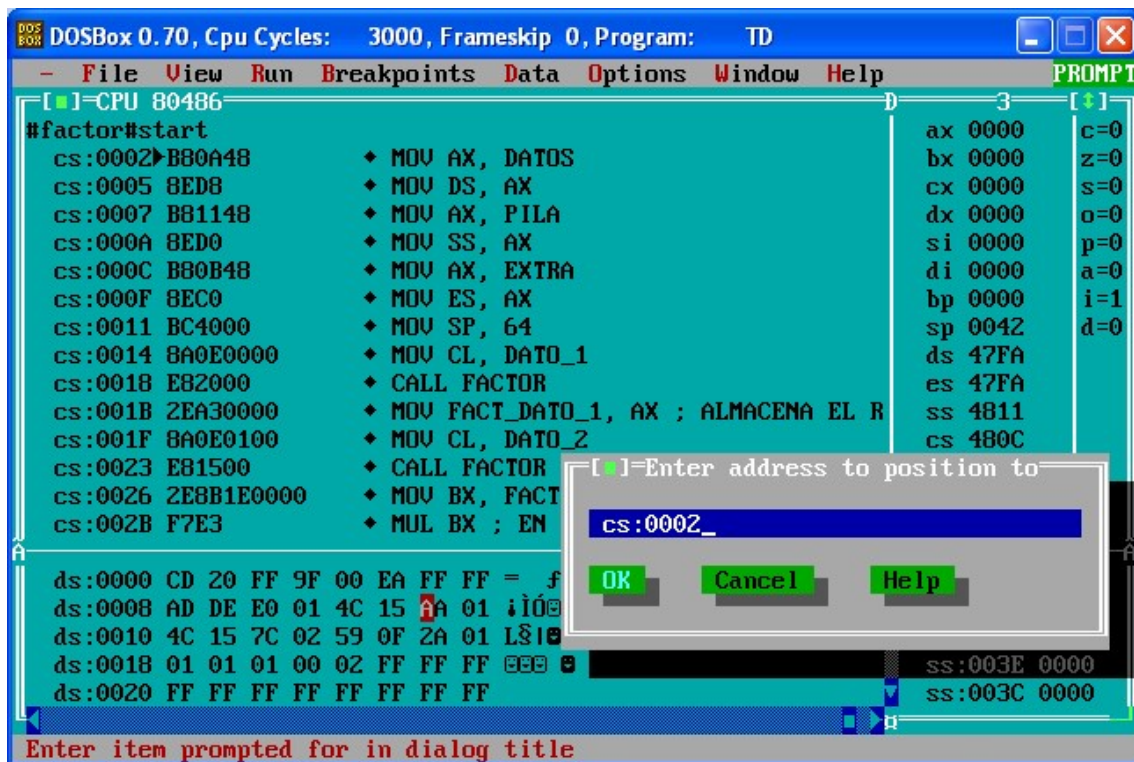
ss:0040 52FB

ss:003E 0000

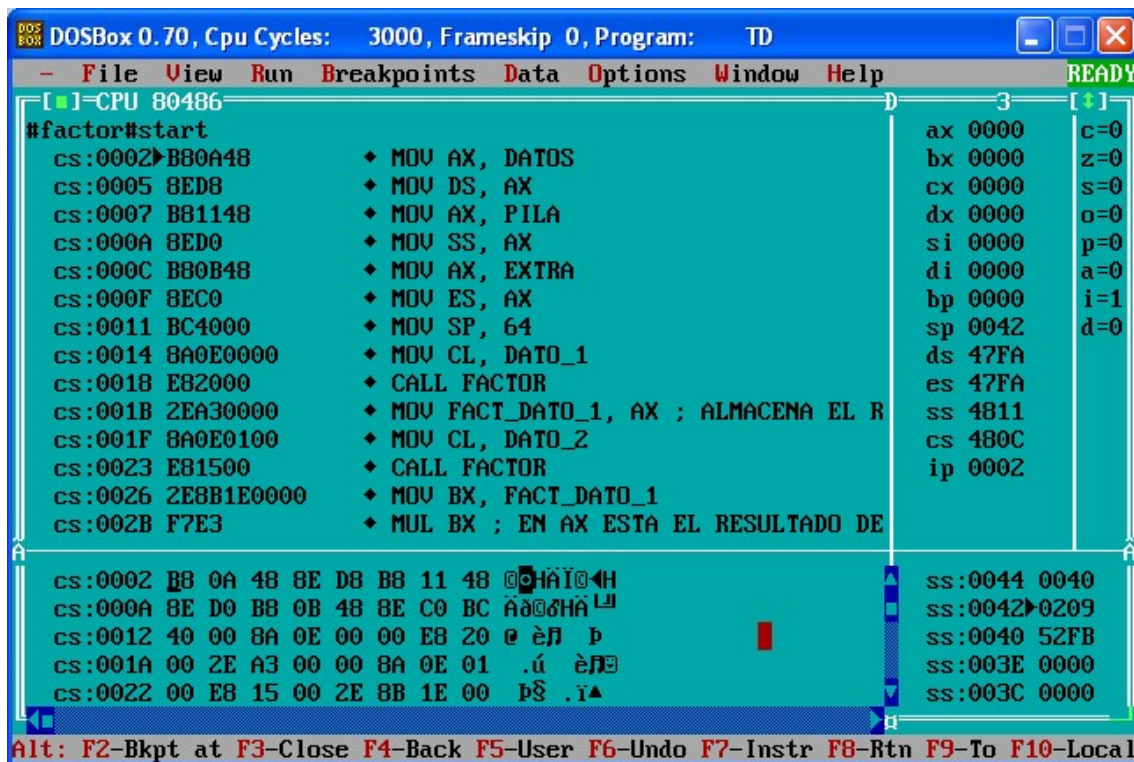
ss:003C 0000

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

Y por último, la quinta ventana, situada justo debajo de la primera, muestra el contenido de un área de memoria específica. Podemos mostrar la zona que queramos accediendo a su menú propio (botón derecho del ratón) y utilizando la opción Goto.



Si introducimos la dirección cs:0002, vemos las instrucciones del programa almacenadas en esa zona de la memoria. Obsérvese la correspondencia entre la primera ventana y la quinta.



Para salir del programa TD y volver a la línea de comandos de DosBox, basta con teclear Alt+X.

Estructura Básica de Programas en Lenguaje Ensamblador 80x86

Abrir el programa fuente “factor.asm” en un editor de texto. Podemos observar que básicamente contiene las definiciones de los diferentes segmentos del programa (datos, pila, extra y código). El orden en que aparecen las definiciones de los diferentes segmentos dentro del programa fuente es indiferente. No siempre estarán presentes los 4 segmentos en todos los programas. Por ejemplo, el segmento extra podría no estar definido.

Al comienzo del segmento de código hay que utilizar la directiva ASSUME para relacionar el nombre dado a los segmentos con el registro de segmento a través del cual van a ser accedidas las posiciones de memoria existentes en cada segmento. Sin embargo esta directiva no carga el valor de los registros de segmento (DS, SS, ES) con los valores asignados a los segmentos. Las instrucciones que cargan estos valores deben ser las primeras de cualquier programa.

```
MOV AX, DATOS  
  
MOV DS, AX  
  
MOV AX, PILA  
  
MOV SS, AX  
  
MOV AX, EXTRA  
  
MOV ES, AX
```

Las instrucciones sólo pueden existir en el segmento de código. Sin embargo, los datos pueden estar definidos en cualquier segmento. La situación más usual es que aparezcan todos en los segmentos de datos/extra, pero en este ejemplo vemos definida la variable FACT_DATO_1 dentro del segmento de código. Esta es la razón por la cual las instrucciones del programa comienzan en la dirección cs:0002 y no en la cs:0000 (que será la habitual durante el curso).

Para finalizar el programa es necesario utilizar la interrupción INT 21H con AX=4C00h, que devuelve el control al sistema operativo.

La última línea debe tener la directiva END seguida por el nombre del procedimiento donde empieza a ejecutarse el programa, en nuestro caso será el procedimiento START. Al arrancar el TD, la flecha apuntará justo al comienzo de dicho procedimiento, y el registro “ip” contendrá la dirección de memoria correspondiente.

Inspección de variables

En ocasiones puede sernos de gran utilidad ver el contenido de las variables de nuestro programa en memoria mientras estamos en el proceso de depuración con el “td”.

Para ello es necesario modificar ligeramente el makefile que creamos anteriormente. En concreto, vamos a modificar la última línea de la siguiente forma:

Antes:

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

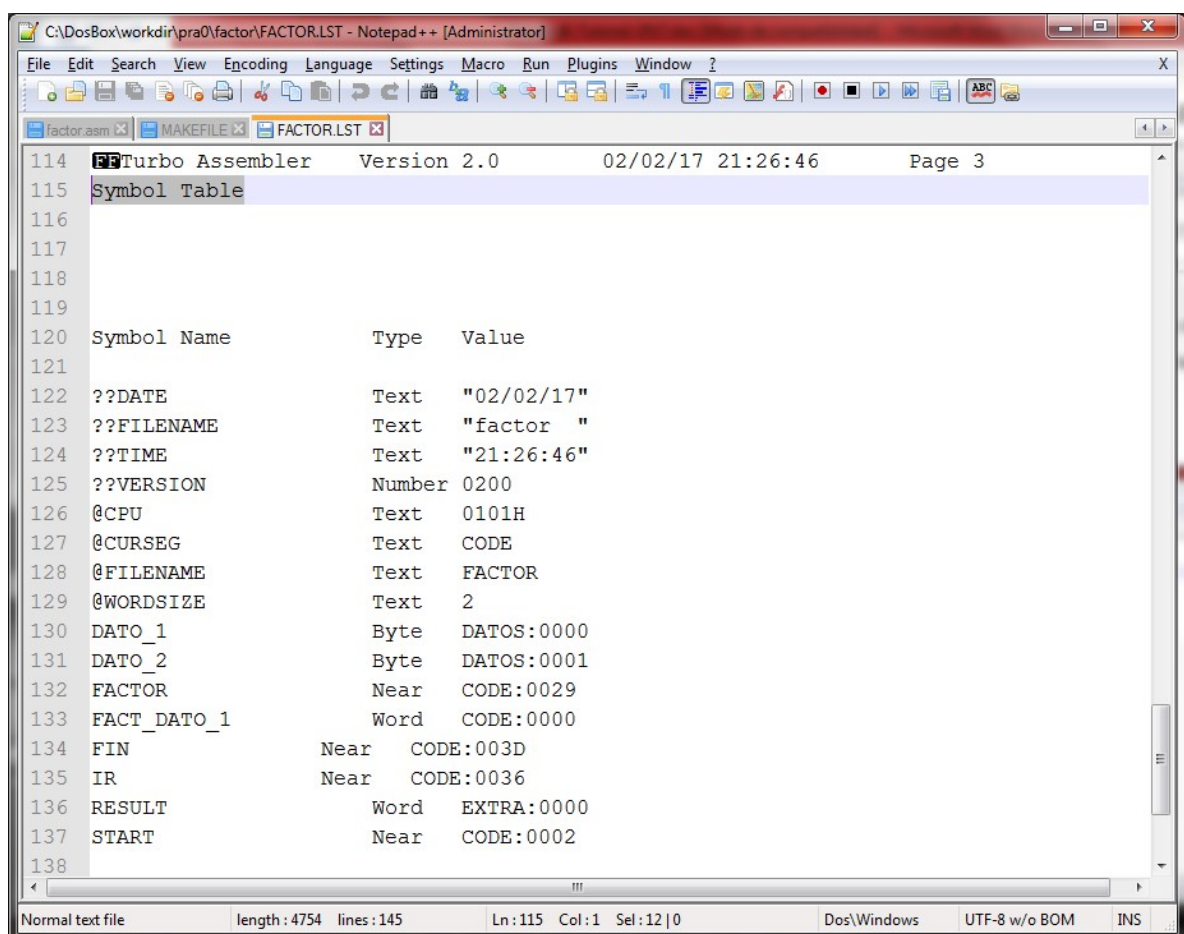
Ahora:

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

Si ejecutamos “make” tras estos cambios (borrando los ficheros factor.obj y factor.exe previamente), podemos observar tecleando “dir” que se ha creado un fichero adicional, llamado factor.lst.

En este nuevo fichero podemos encontrar en primer lugar la correspondencia entre las instrucciones en ensamblador de nuestro programa y las instrucciones en lenguaje máquina generadas (obsérvese que es una información similar a la que vemos en la ventana 1 del td).

Sin embargo, la información útil si queremos ver el contenido de las variables del programa se encuentra hacia el final del fichero factor.lst: la tabla de símbolos del programa.



A falta de adquirir los conocimientos teóricos que nos permitan entender completamente esta información, avanzaremos aquí que las direcciones lógicas de las variables se componen de dos campos: segmento y offset. Así, por ejemplo:

DATO_1: segmento DATOS, offset 0

DATO_2: segmento DATOS, offset 1

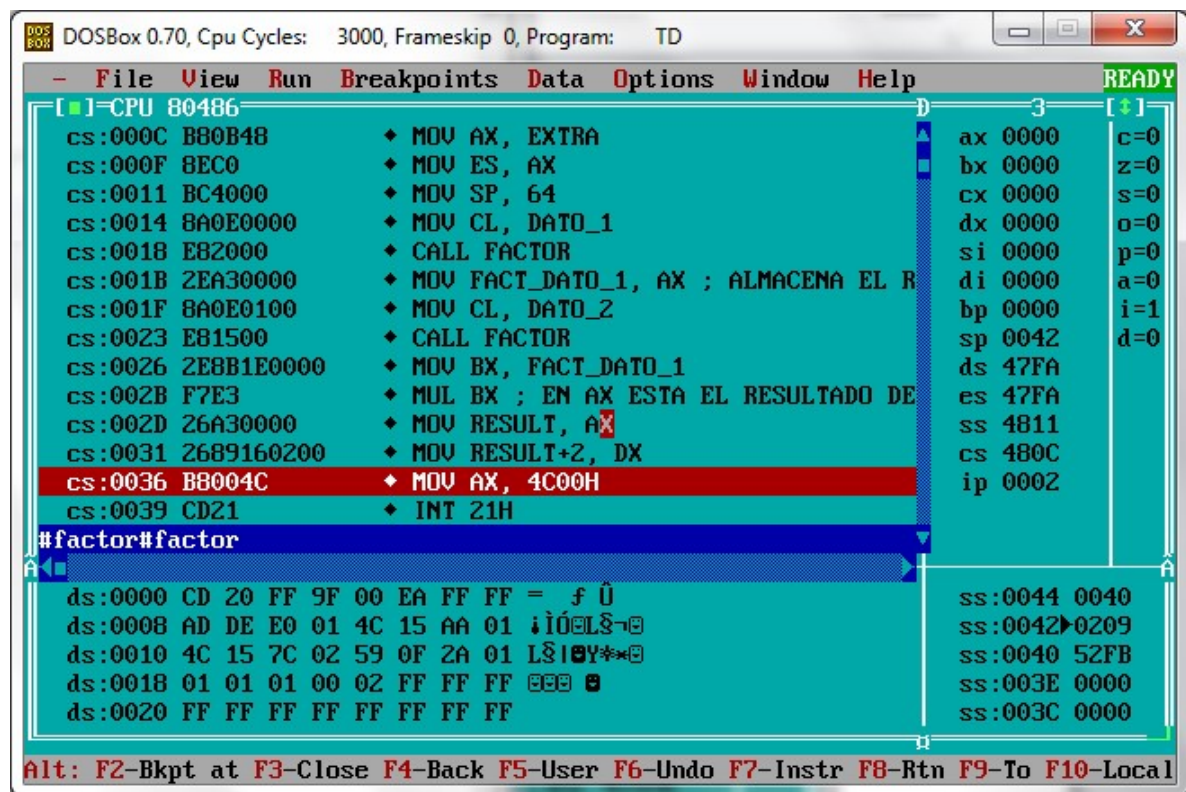
FACT_DATO_1: segmento CODE, offset 0

RESULT: segmento EXTRA, offset 0

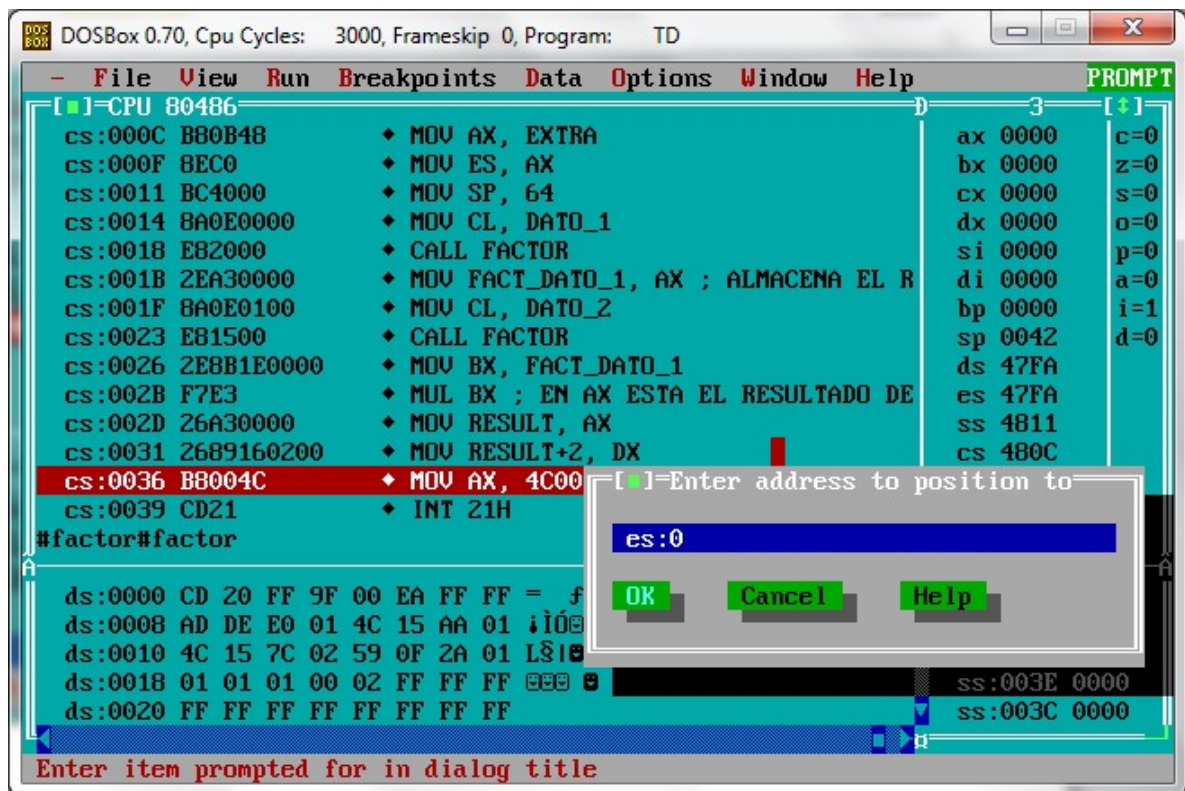
Con esta información en mente, vamos a ejecutar nuestro programa en el "td". Esta vez, en lugar de ejecutar paso a paso con F7, podemos colocar un *breakpoint* en la instrucción

MOV AX, 4C00H

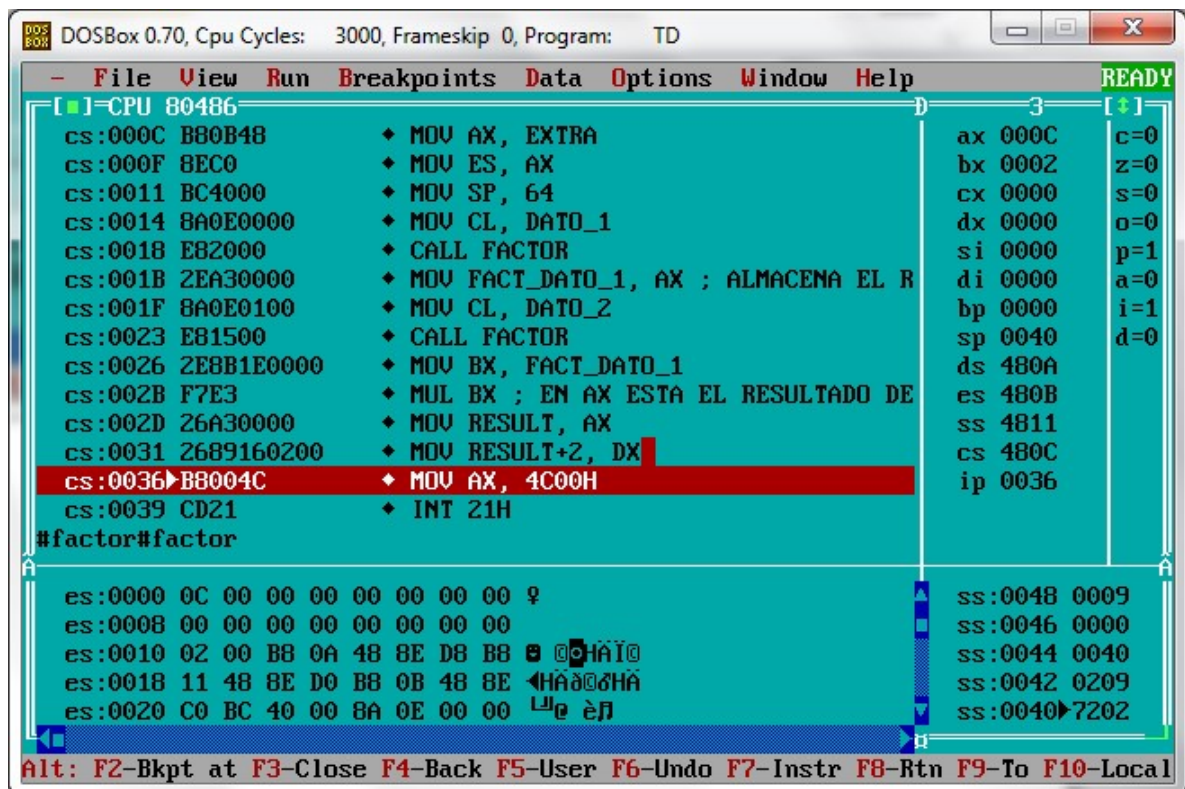
Para ello, nos situamos en la instrucción con el cursor y pulsamos F2. Si volvemos con el cursor hacia arriba comprobamos que el breakpoint se ha activado, lo cual se indica mediante una línea de color rojo.



Seguidamente ejecutamos el programa pulsando F9. El programa será ejecutado hasta detenerse en el breakpoint. En este momento la variable RESULT ya debe contener su valor final. Podemos inspeccionar su valor en la ventana número 5, mediante "Goto →ES:0". Recordemos que hemos asociado el segmento EXTRA al registro ES mediante la directiva ASSUME.



Podemos ver como la ventana muestra el valor **0C**, es decir, el valor 12 decimal (= 2! x 3!).



De manera análoga podríamos inspeccionar el resto de variables del programa (`DATO_1`, `DATO_2` y `FACT_DATO_1`), mediante "Goto →DS:0", "Goto →DS:1" y "Goto →CS:0" respectivamente.