

PRÁCTICA 3

INTRODUCCION A LOS COMPUTADORES

PRÁCTICA 3:

MANEJO DEL SOFTWARE DE SIMULACIÓN DEL 8086

**ESTRUCTURA DE UN PROGRAMA EN
ENSAMBLADOR**

TEORÍA PRÁCTICA 3

OBJETIVOS:

- ✿ Conocer el esqueleto básico de un programa escrito en ensamblador 8086
- ✿ Conocer el editor como herramienta para la escritura de **código fuente**
- ✿ Conocer en qué consiste el proceso de compilación, y el código resultante: **código objeto**
- ✿ Conocer el proceso de **ejecución** de un programa sobre el emulador
- ✿ Manejar la ejecución paso a paso o **modo traza** sobre el emulador
- ✿ Manejar las interrupciones software para la E/S de información
- ✿ Conocer las normas para la entrega de trabajos

CONTENIDOS:

1. Estructura de un programa en ensamblador
2. Compilar, emular y ejecutar sobre el emulador
3. Manejo de trazas sobre el emulador
4. Manejo de interrupciones para la E/S de información
5. Documentación

TEORÍA PRÁCTICA 3

1.- Estructura de un programa en ensamblador 8086

Tipo de ficheros

- .ASM fichero fuente en lenguaje ensamblador
- .INC fichero fuente con programas librerías
- .OBJ fichero objeto obtenido después de compilar
- .EXE fichero ejecutable

Segmentos (definición):

```
{DATA|STAK|CODE} SEGMENT  
.....  
ENDS
```

O bien

```
<nombresegmento> SEGMENT {DATA|STAK|CODE}  
.....  
<nombresegmento> ENDS
```

TEORÍA PRÁCTICA 3

1.- Estructura de un programa en ensamblador 8086

INTRODUCCION A LOS COMPUTADORES

E
s
t
r
u
c
t
u
r
a

d
e

p
r
o
g
r
a
m
a

Directivas
Directivas

Datos
Datos

Pila
Pila

Código
Código

TITLE <nombre>

#MAKE_EXE#

; Ejemplo de estructura de programa

DATA SEGMENT dseg SEGMENT 'DATA'

.....
ENDS dseg ENDS

STACK SEGMENT sseg SEGMENT 'STACK'

.....
ENDS sseg ENDS

CODE SEGMENT cseg SEGMENT 'CODE'
principal: principal PROC NEAR

.....
RET
.....
ENDS principal ENDP
 cseg ENDS

END principal

Comentario

TEORÍA PRÁCTICA 3

1.- Estructura de un programa en ensamblador 8086

Directivas más comunes:

DEFINICIÓN DE PROCEDIMIENTOS

nombre_procedimiento **PROC** (NEAR O FAR)

nombre_procedimiento **ENDP**

INCLUSIÓN DE FICHEROS FUENTE .INC

INCLUDE 'nombre_fichero.inc'

DEFINICIÓN DE VARIABLES EN EL SEGMENTO DE DATOS

nombre_variable_tipo_byte **DB** valor_inicial

nombre_variable_tipo_word **DW** valor_inicial

nombre_variable **DB** 100 **DUP(?)**

TEORÍA PRÁCTICA 3

1.- Estructura de un programa en ensamblador 8086

Estructura de programa

TITLE Plantilla Codigo 8086 (fich. EXE)

```
; AUTOR      emu8086
; FECHA      ?
; VERSION    1.00
; FICHERO    ?.ASM
; 8086 Code Template
; Directiva para salida EXE :
  #MAKE_EXE#
```

DSEG **SEGMENT** 'DATA'

; Aquí van las variables

DSEG **ENDS**

SSEG **SEGMENT** STACK 'STACK'

DW 100h DUP(?)

SSEG **ENDS**

CSEG **SEGMENT** 'CODE'

;*****

START **PROC** **FAR**

; Se guardan direcciones para la vuelta al SO:

PUSH DS

MOV AX, 0

PUSH AX

; set segment registers:

MOV AX, DSEG

MOV DS, AX

MOV ES, AX

; TODO: add your code here!!!!

; vuelta al SO:

RET

START **ENDP**

;*****

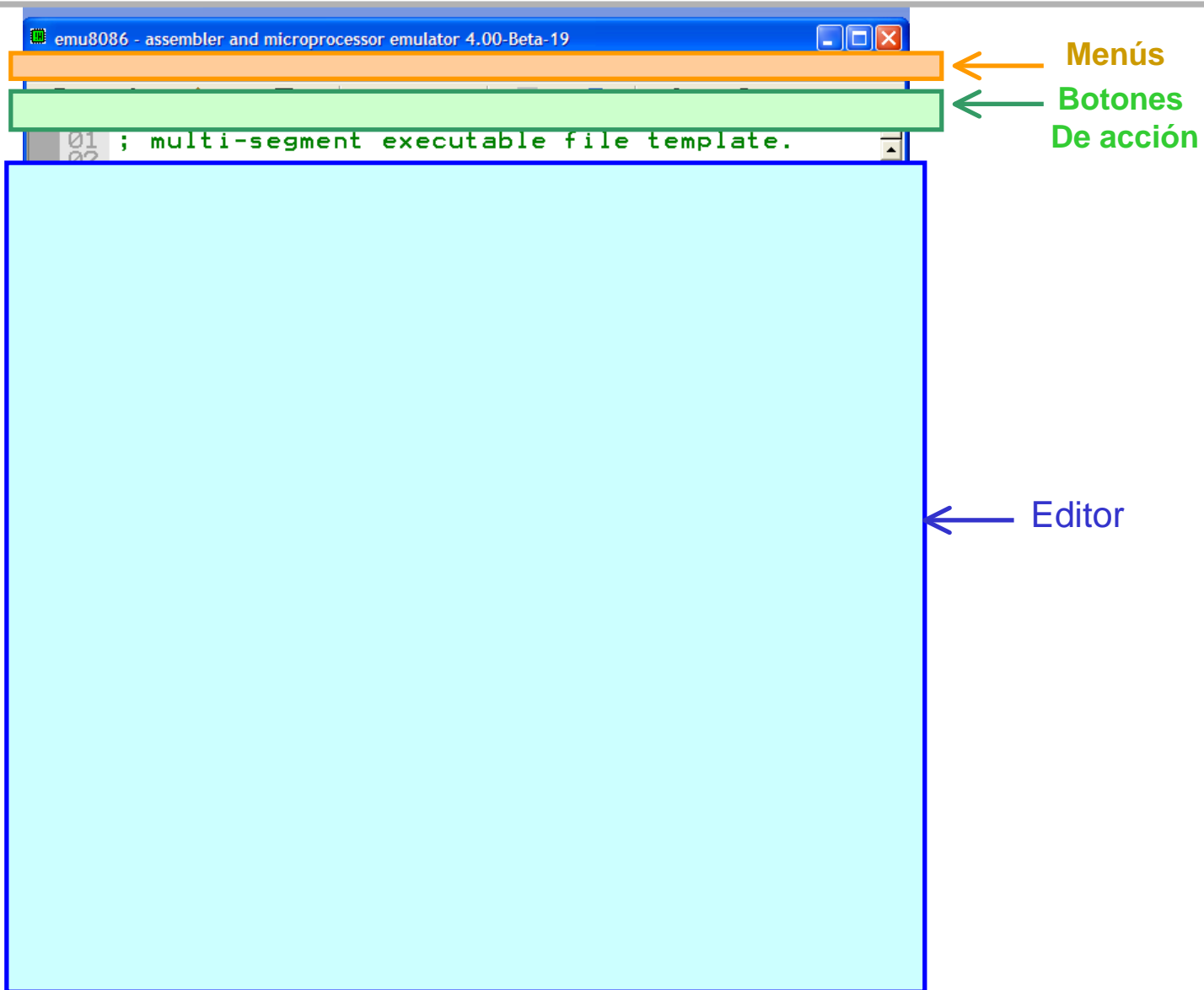
CSEG **ENDS**

END **START** ; Punto de comienzo.

TEORÍA PRÁCTICA 3

2. Compilar, emular y ejecutar sobre el emulador

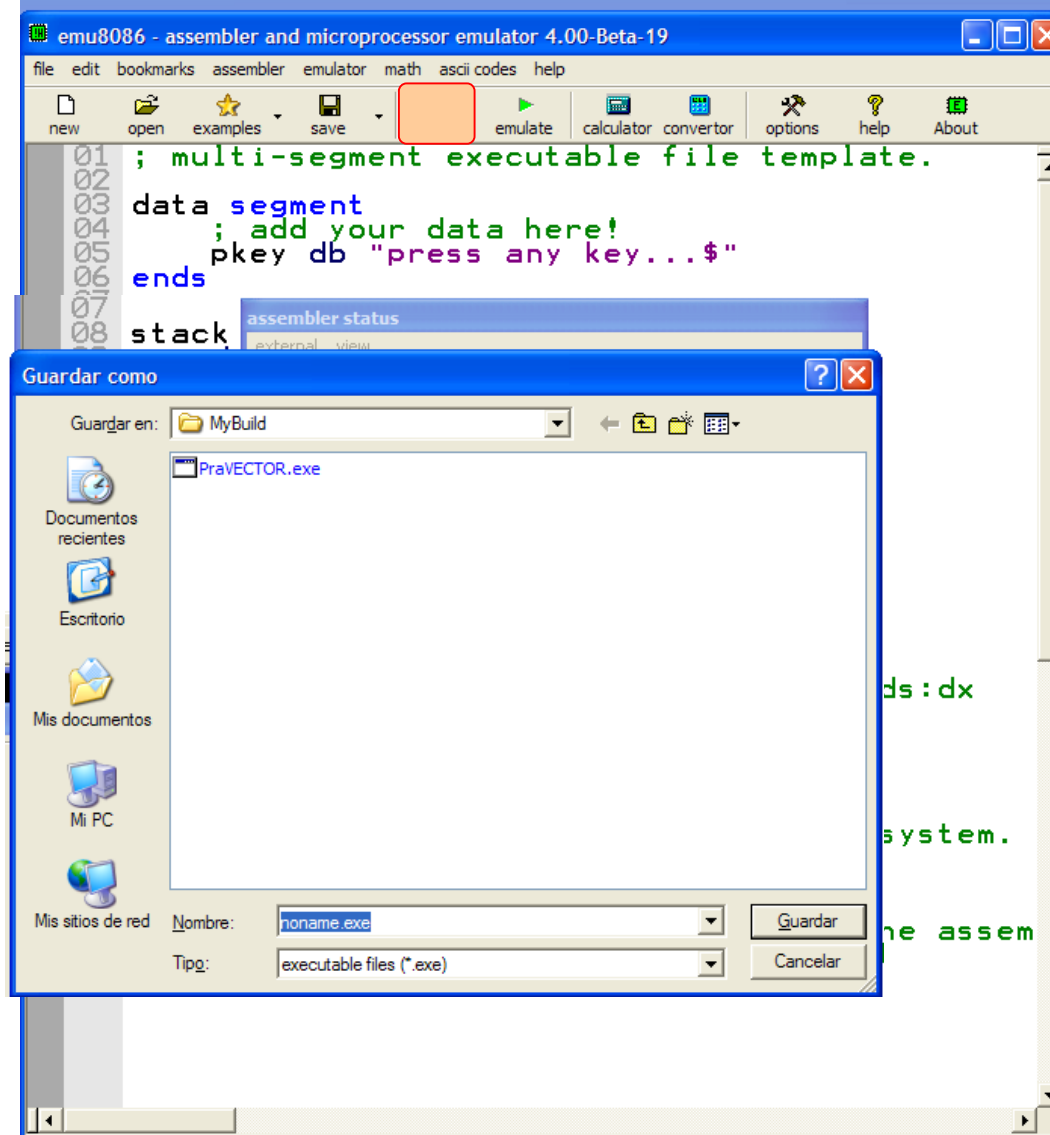
INTRODUCCION A LOS COMPUTADORES



TEORÍA PRÁCTICA 3

2. Compilar, emular y ejecutar sobre el emulador

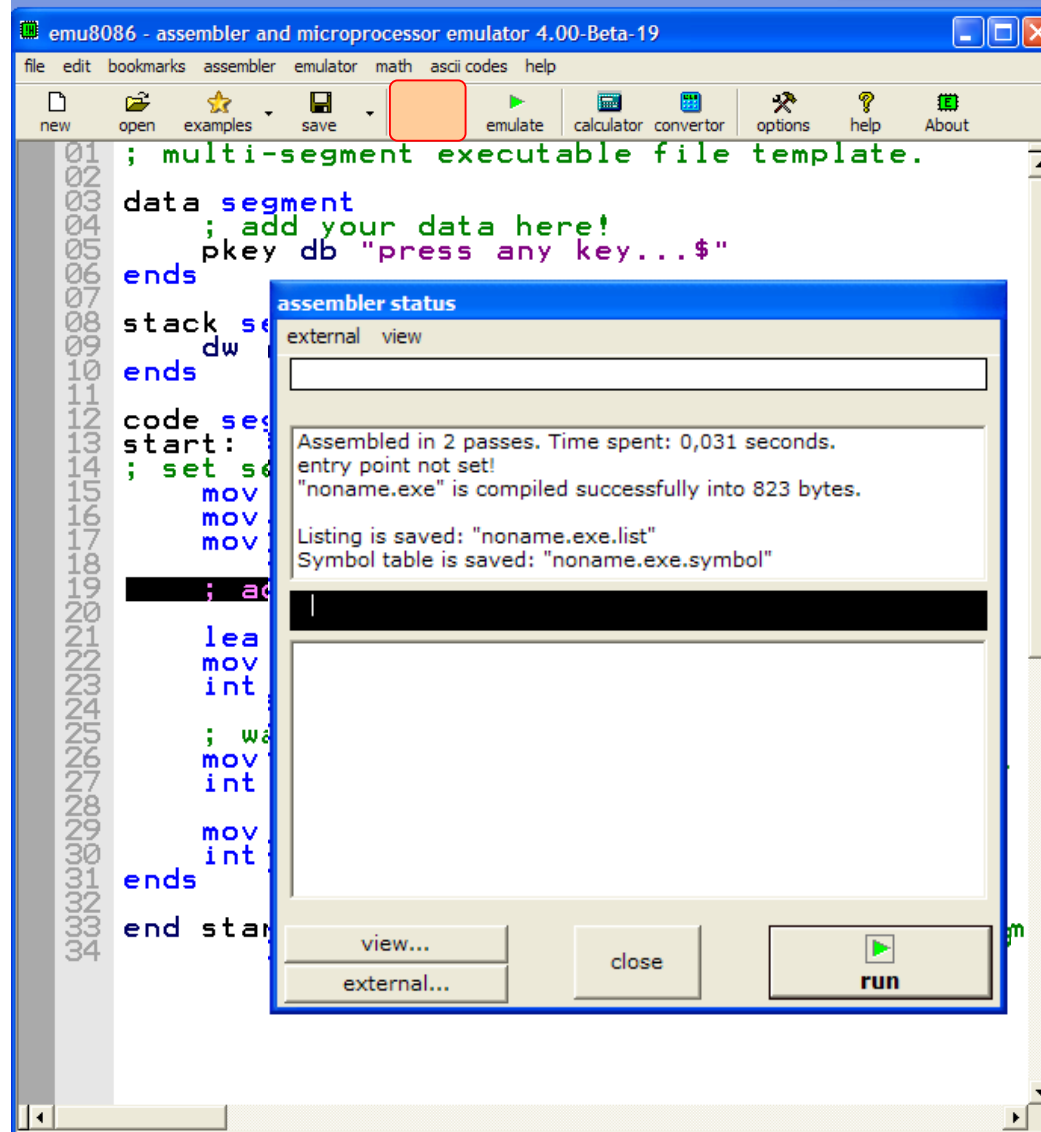
INTRODUCCION A LOS COMPUTADORES



TEORÍA PRÁCTICA 3

2. Compilar, emular y ejecutar sobre el emulador

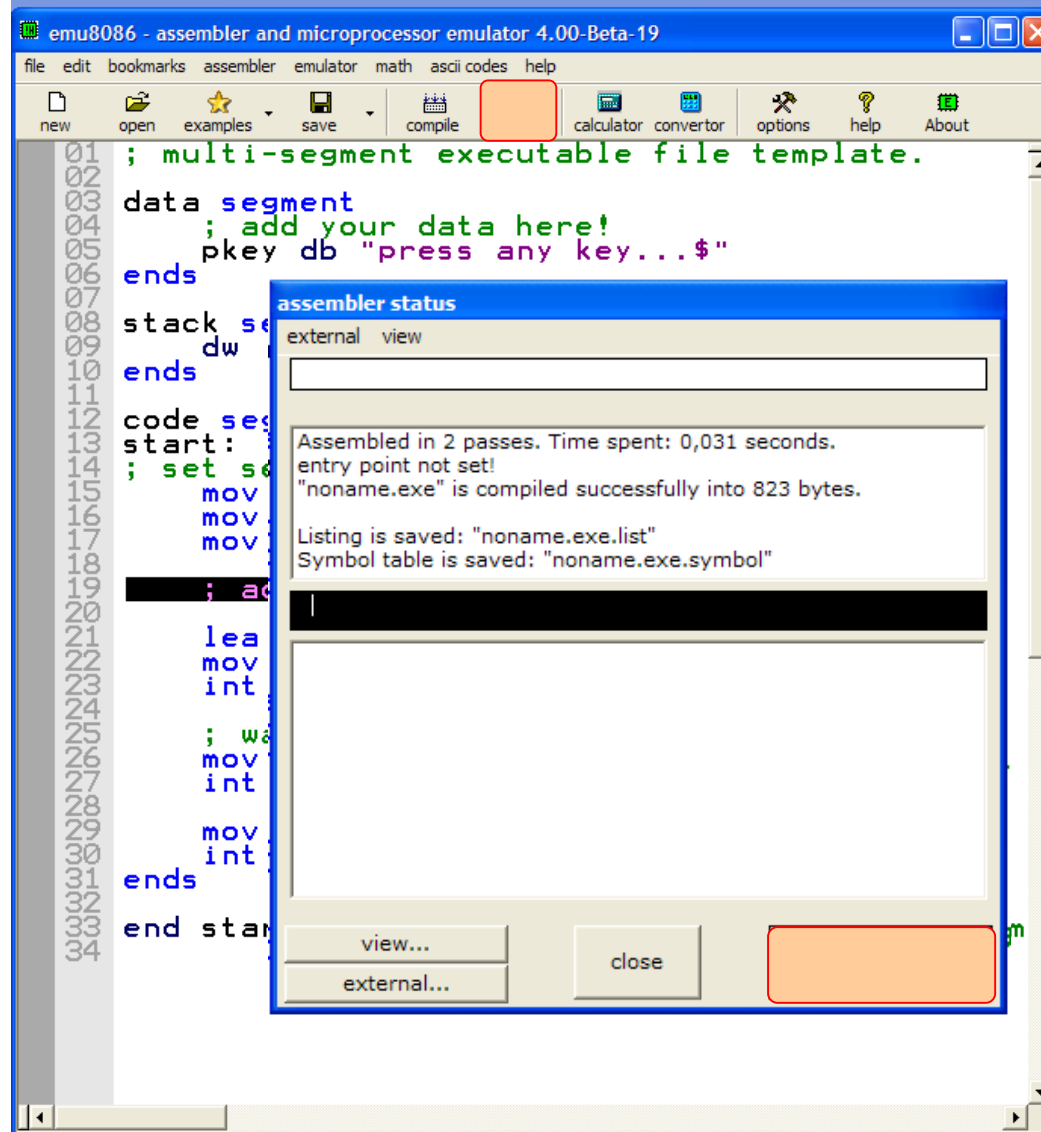
INTRODUCCION A LOS COMPUTADORES



TEORÍA PRÁCTICA 3

2. Compilar, emular y ejecutar sobre el emulador

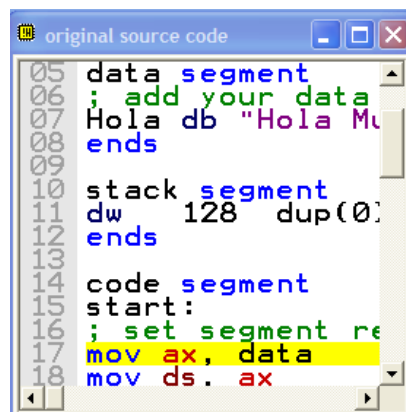
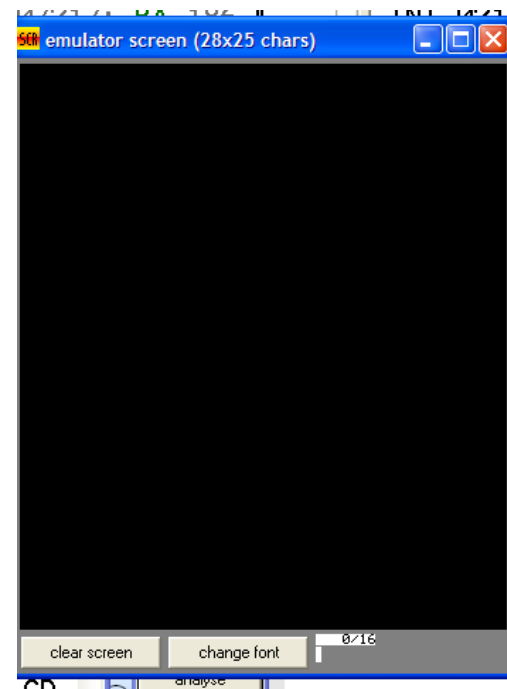
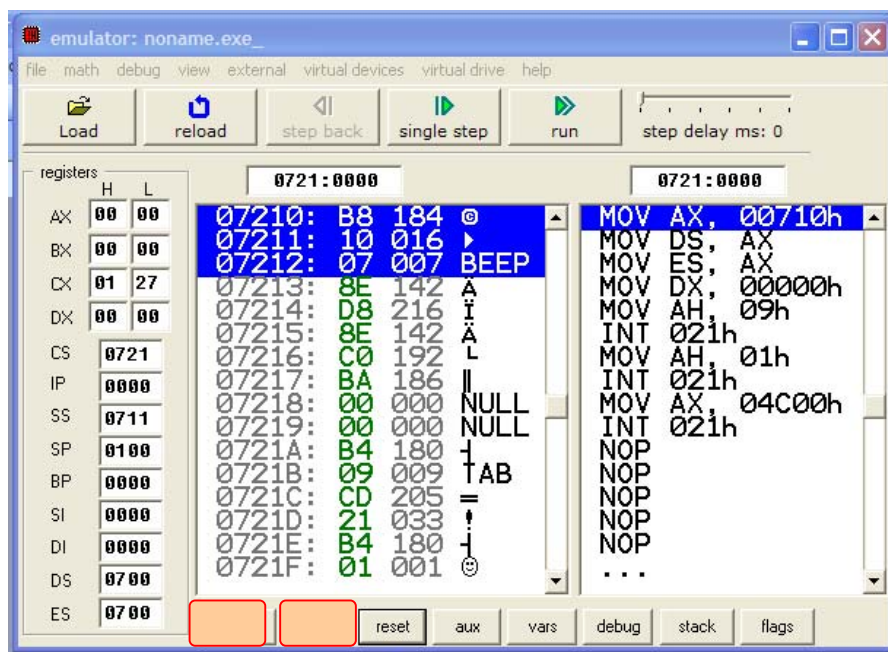
INTRODUCCION A LOS COMPUTADORES



TEORÍA PRÁCTICA 3

2. Compilar, emular y ejecutar sobre el emulador

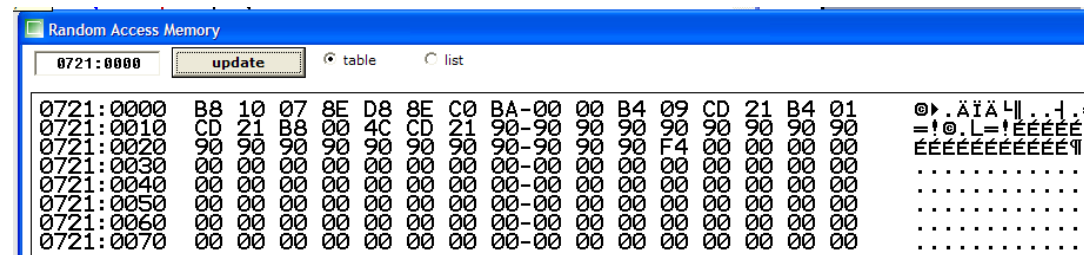
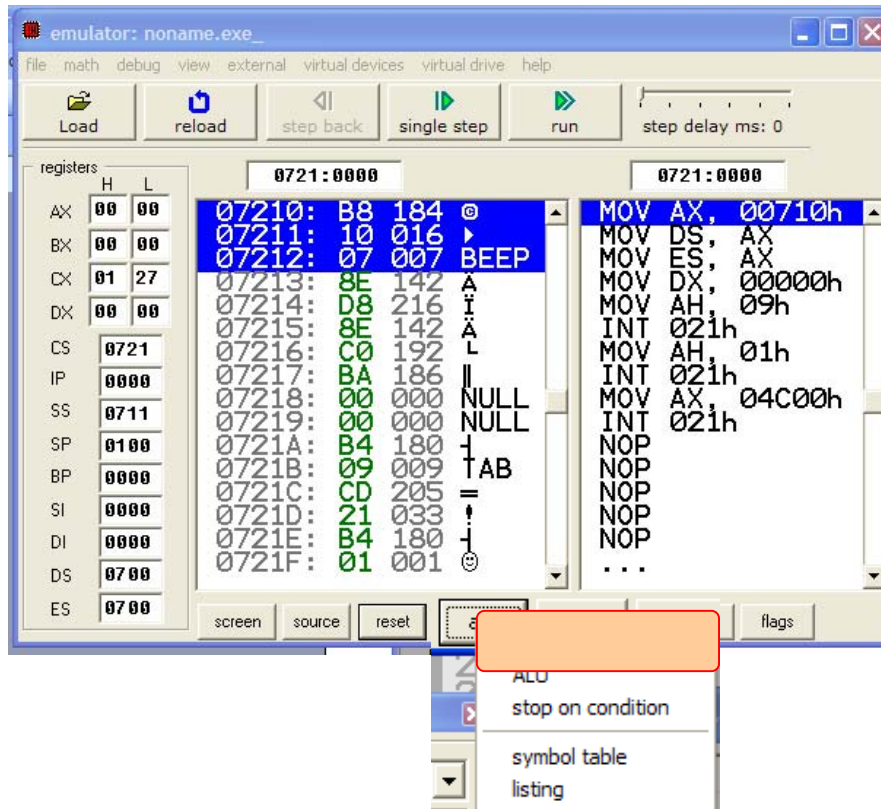
INTRODUCCION A LOS COMPUTADORES



TEORÍA PRÁCTICA 3

2. Compilar, emular y ejecutar sobre el emulador

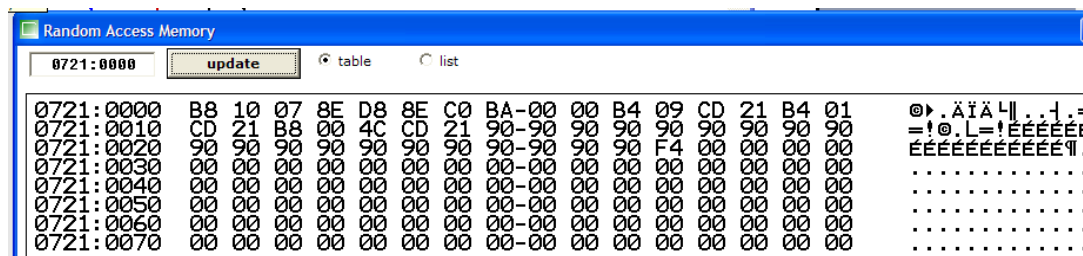
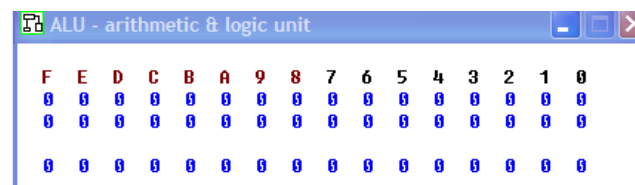
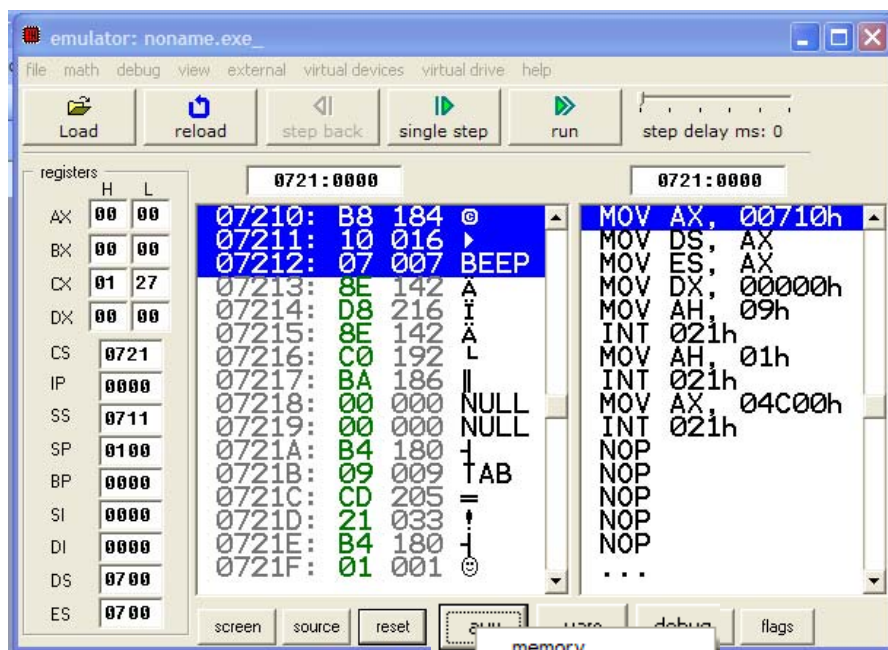
INTRODUCCION A LOS COMPUTADORES



TEORÍA PRÁCTICA 3

2. Compilar, emular y ejecutar sobre el emulador

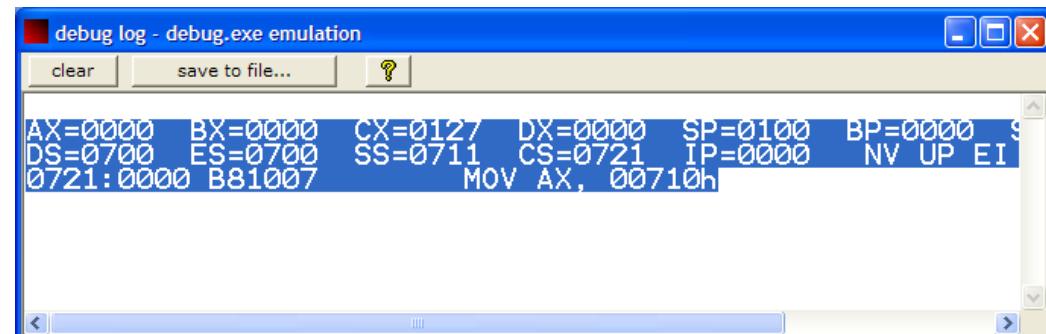
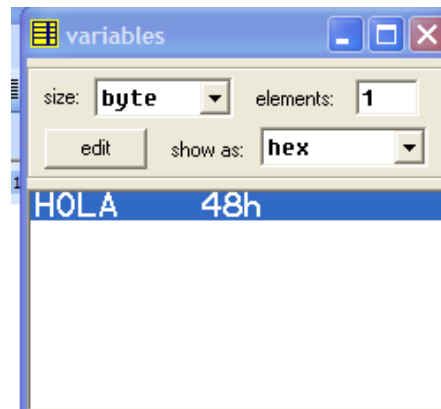
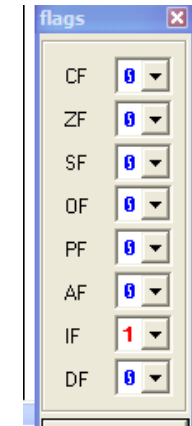
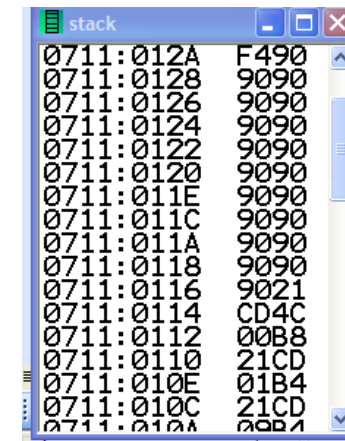
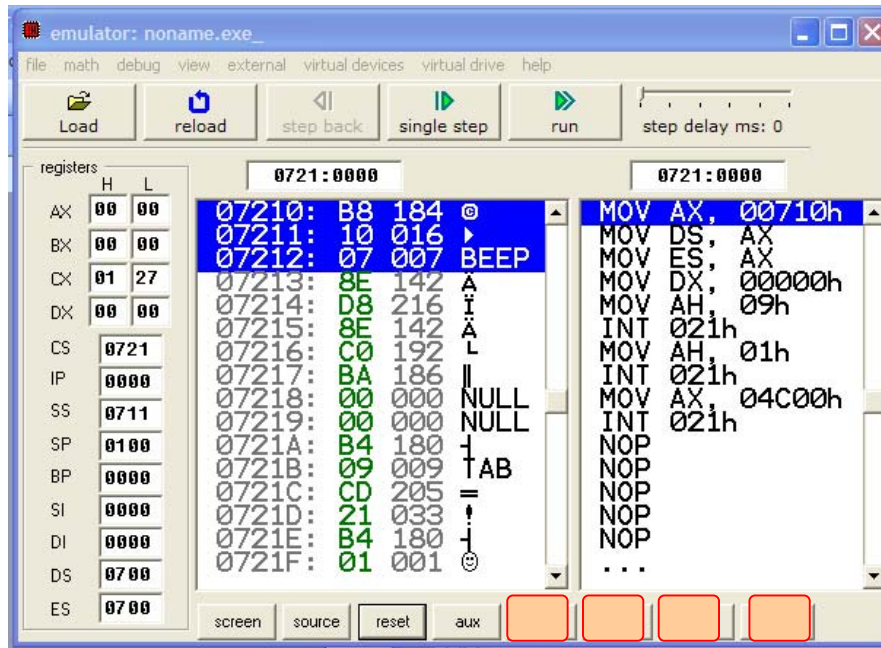
INTRODUCCION A LOS COMPUTADORES



TEORÍA PRÁCTICA 3

2. Compilar, emular y ejecutar sobre el emulador

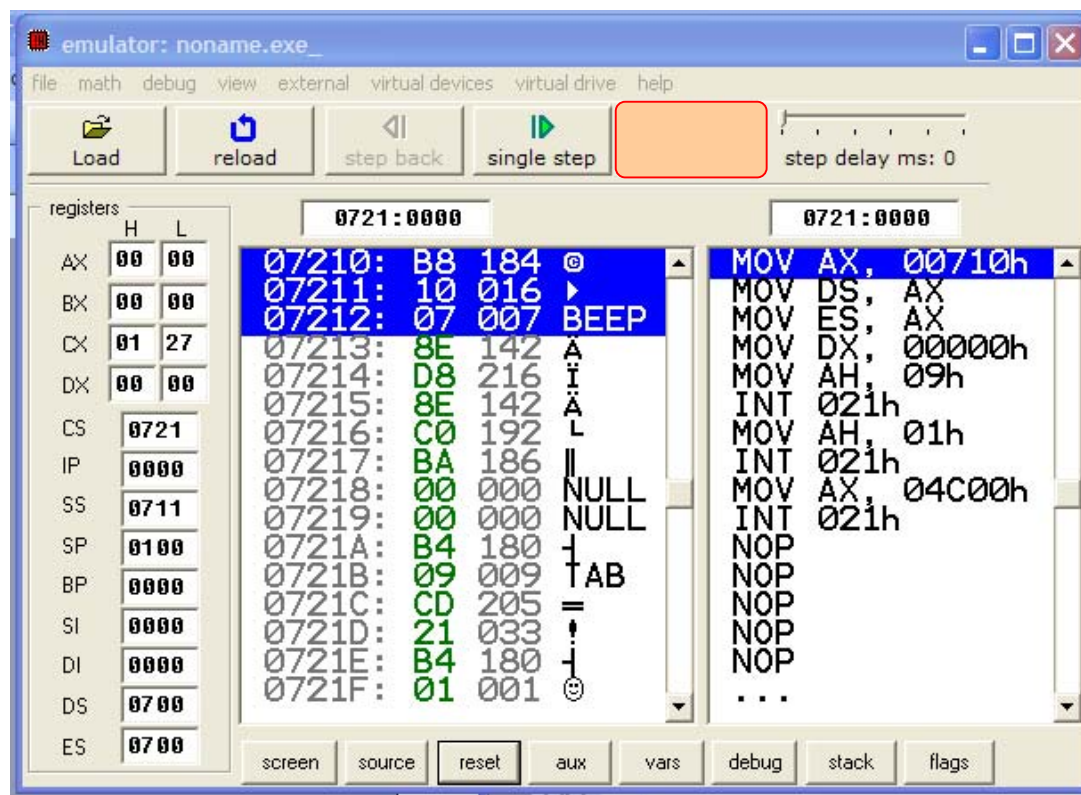
INTRODUCCION A LOS COMPUTADORES



TEORÍA PRÁCTICA 3

3. Trazas en el emulador

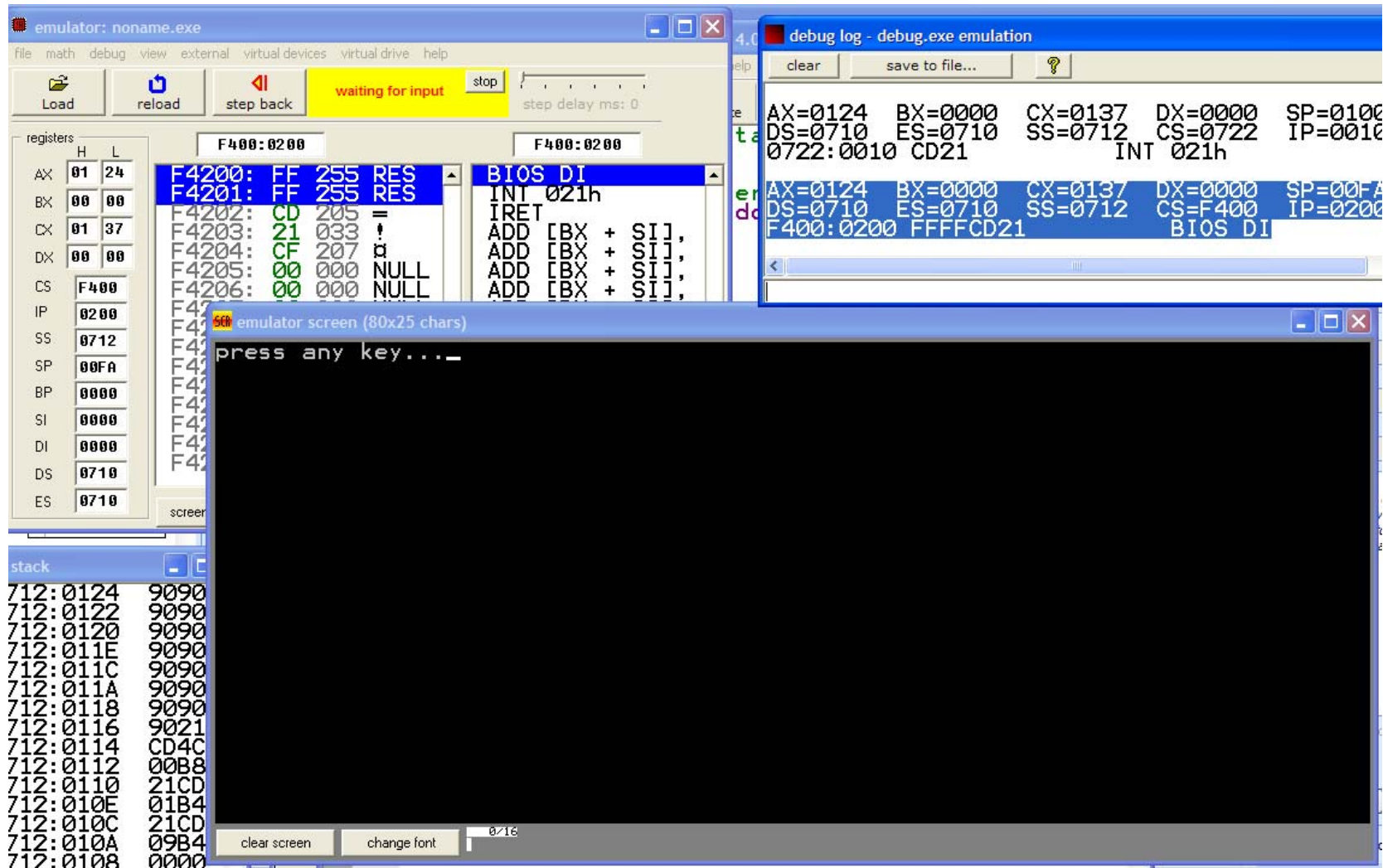
INTRODUCCION A LOS COMPUTADORES



TEORÍA PRÁCTICA 3

3. Trazas en el emulador

INTRODUCCION A LOS COMPUTADORES



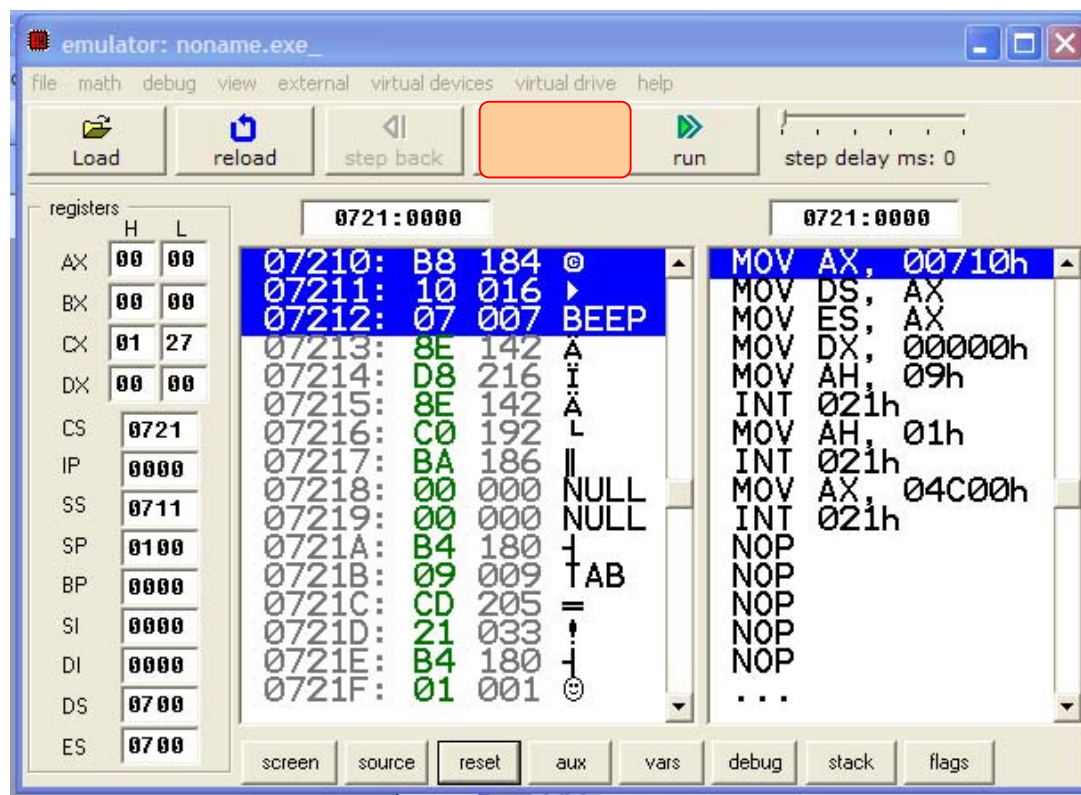
The screenshot displays an x86 emulator interface with several windows:

- emulator: noname.exe**: The main emulator window. It includes a menu bar (file, math, debug, view, external, virtual devices, virtual drive, help) and a toolbar with buttons for Load, reload, step back, waiting for input, and stop. Below the toolbar is a registers window showing the state of various registers (AX, BX, CX, DX, CS, IP, SS, SP, BP, SI, DI, DS, ES) with their high (H) and low (L) bytes. The stack window is also visible, showing memory addresses and values.
- debug log - debug.exe emulation**: A window showing the debug log. It contains a table of register values and instructions. The log shows the state of registers (AX=0124, BX=0000, CX=0137, DX=0000, SP=0100, DS=0710, ES=0710, SS=0712, CS=0722, IP=0010) and the instruction being executed (INT 021h). The log also shows the state of the stack (F400:0200 FFFCD21) and the instruction being executed (BIOS DI).
- emulator screen (80x25 chars)**: A window showing the output of the emulator. It displays the text "press any key..." on a black background.

TEORÍA PRÁCTICA 3

3. Trazas en el emulador

INTRODUCCION A LOS COMPUTADORES

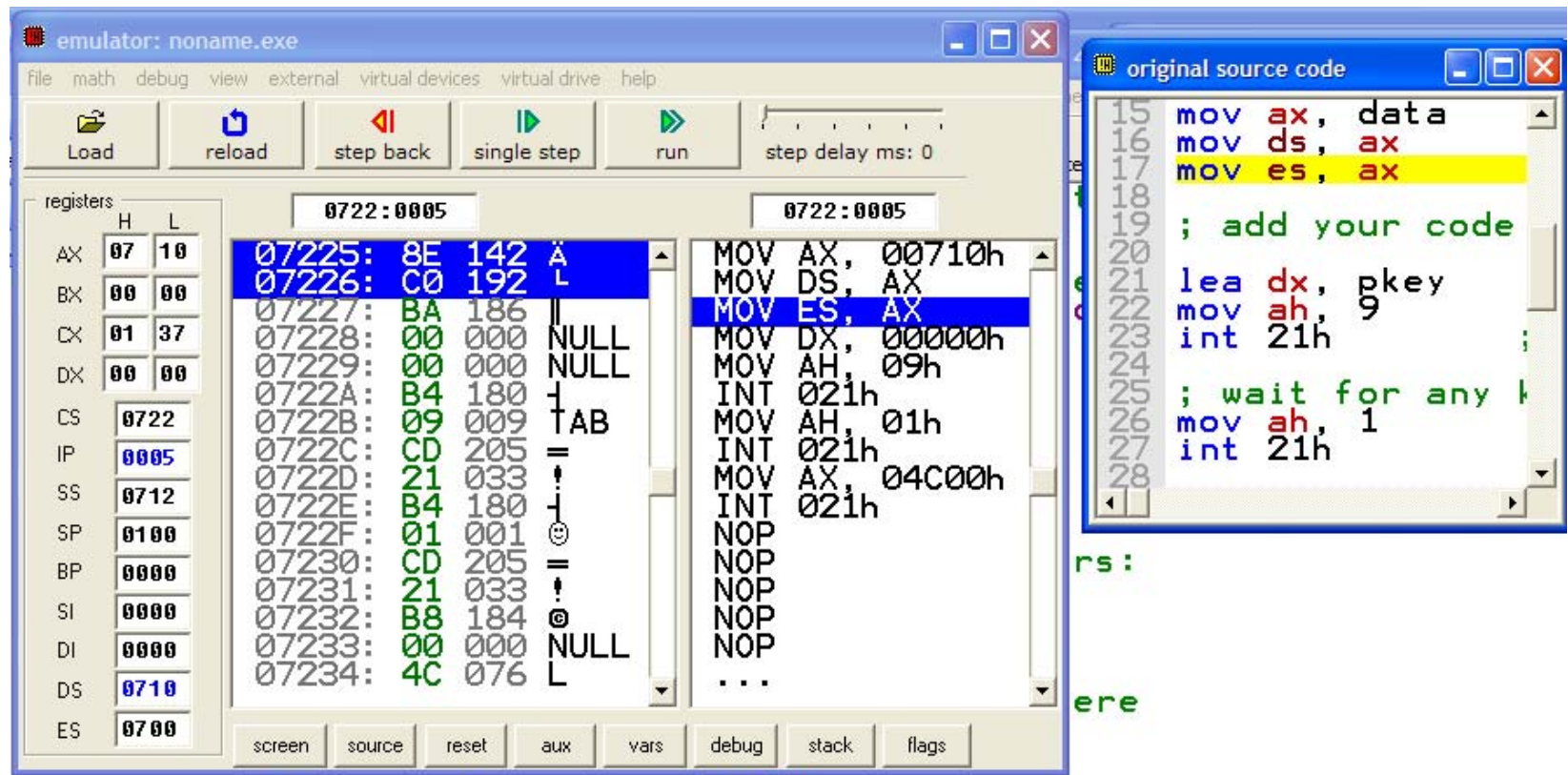




TEORÍA PRÁCTICA 3

3. Trazas en el emulador

INTRODUCCION A LOS COMPUTADORES



TEORÍA PRÁCTICA 3

3._Uso de interrupciones software para la E/S de información a través de teclado y pantalla.

Para realizar operaciones de E/S de información utilizaremos una interrupción software. Es una llamada a una rutina que gestiona la operación específica de E/S.

La interrupción software es: **INT 21h**

Esta interrupción sirve para hacer múltiples operaciones de E/S y para especificar la función exacta que desees realizar se utiliza el registro AH.

Por ejemplo si deseamos sacar un mensaje por pantalla, esto son los pasos:

1. Definir en el segmento de datos la variable de memoria que guarda el mensaje:

Mensaje DB 'ESTO ES UN EJEMPLO \$' (la cadena de caracteres debe acabar en \$)

2. Dentro del segmento de código pondremos las siguientes instrucciones:

```
MOV DX, OFFSET MENSAJE (DX debe guarda la dirección de memoria donde  
esta el mensaje guardado  
MOV AH, 9 (función para enviar una cadena de caracteres a pantalla)  
INT 21H
```

TEORÍA PRÁCTICA 3

3. Uso de interrupciones software para la E/S de información a través de teclado y pantalla.

Para introducir un valor numérico a través de teclado haremos lo siguiente:

1. Definir en el segmento de datos la variable de memoria que contendrá el número leído de teclado:

numero DB 0 (podemos ponerlo a valor inicial cero o cualquier otro valor)

2. Dentro del segmento de código:

```
MOV AH, 1      (función de lectura de teclado)
INT 21H        (espera a que se introduzca el número que se guarda en AL)
MOV numero,AL  (guardo en la variable numero el valor leído)
```

5. Documentación

```

TITLE 8086 Code Template (for EXE file)
;   AUTHOR      NOMBRE DEL ALUMNO/A
;   DATE        FECHA
;   VERSION     1.00
;   FILE        NOMBRE DEL FICHERO
; AQUÍ SE EXPLICA QUE HACE EL PROGRAMA, DONDE ESTAN LOS DATOS Y
LOS RESULTADOS.
DSEG SEGMENT 'DATA'
;Definición de variables y constantes; comentadas todo lo que no sea obvio
DSEG ENDS
SSEG SEGMENT STACK 'STACK'
      DW 100h DUP(?)
SSEG ENDS
CSEG SEGMENT 'CODE'
START PROC FAR
      PUSH DS                ; TODO DEBE ESTAR COMENTADO
      MOV AX, 0              ; cada procedimiento
      PUSH AX                ; cada bucle e instrucción
      .....
START ENDP

PROCEDI1 PROC NEAR ; que hace este procedimiento ...
      ....
      RET
PROCEDI1 ENDP
      .....
CSEG ENDS

```