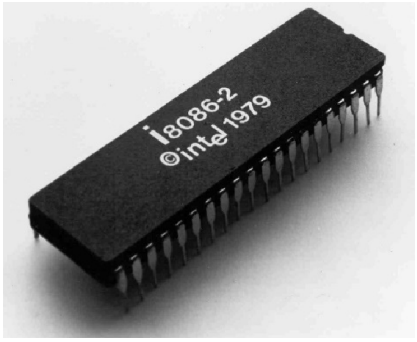
TUTORIAL EMU8086
EMULADOR

Introducción

EMU8086 es un emulador de la arquitectura de procesadores 8086. Posee una interfaz fácil de utilizar que permite familiarizarse con los fundamentos de la programación en lenguaje ensamblador de forma muy intuitiva, aparte de eso brinda una serie de recursos para ejecutar y depurar los programas. Tiene algunas desventajas como el de no soportar algunas interrupciones que posee el sistema operativo y tampoco puede acceder a los puertos físicos (reales), solo los emula usando otros programas.



El 8086, es un procesador creado por Intel y lanzado al mercado en 1978. Su interfaz de programación es ampliamente conocida y los procesadores actuales la soportan por compatibilidad.

Comenzaremos explicando la arquitectura del procesador 8086, de tal manera de conocer sus registros y conjunto de instrucciones.

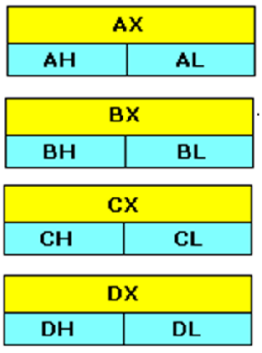
Arquitectura Intel 8086

El procesador 8086 posee un bus de direcciones es de 16 bits. El bus de direcciones de 20 bits que permiten acceder a 1MB de memoria RAM como máximo (2^{20} = 1MB). Posee varios registros de 16 bits organizados de la siguiente manera:

- **Registros de datos:**
  Posee 4 registros que tienen una función por defecto, pero que también pueden ser utilizados para lo que el programador necesite.

```
AX = Acumulador. Es el registro principal utilizado en las instrucciones
     aritméticas.
BX = Base. Utilizado como base, al cual se le suma el desplazamiento u offset.
CX = Contador. Se utiliza como contador en los bucles y operaciones iterativas.
DX = Datos. Utilizado también para operaciones aritméticas.
```

  Los registros también pueden ser manipulador por mitades (8 bits cada una) haciendo mención a la parte alta o High (más significativa) y parte baja o Low (menos significativa.)



  Ejemplo:

  AH: 8 bits más significativa de AX.
  AL: 8 bits menos significativos de AX.

- **Registros de segmentos:**

  CS = Registro de segmento de código. Contiene la dirección del segmento de código, lo que son las instrucciones del programa.

  DS = Registro de segmento de datos. Contiene la dirección del segmento de datos, es decir, el área de memoria donde se encuentran los datos del programa.

  SS = Registro de segmento de pila. Contiene la dirección del segmento de pila.

  ES = Registro de segmento extra. El segmento extra es un segmento adicional que se utiliza para superar la limitación de los 64 Kb del segmento de datos y para hacer la transferencia de datos entre segmentos.

- **Registros punteros de pila:**

  SP = Puntero de pila. Contiene la dirección relativa del segmento de pila.
  BP = Puntero base. Utilizado para fijar el puntero de la pila y poder acceder a los elementos de esta.

- **Registros índices:**

  SI = Índice fuente (Source Index)
  DI = Índice destino (Destination Index)

- **Registro puntero de instrucciones:**

  IP = Puntero de instrucción (Instruction Pointer). Es equivalente a PC de M32.

- **Registro de banderas:**
  Se usa para registrar la información de estado y de control de las operaciones del microprocesador, normalmente asociado a una comparación o una instrucción aritmética. Es equivalente al SR de M32.

  Hay 9 banderas:

  CF = Bandera de acarreo. Indica acarreo en las instrucciones aritméticas (Carry Flag)
  OF = Bandera de desbordamiento aritmético (Overflow Flag)
  ZF = Bandera de resultado cero o comparación igual (Zero Flag)
  SF = Bandera de resultado o comparación negativa (Sign Flag)
  PF = Bandera de paridad (Parity Flag)
  AF = Bandera auxiliar. Indica si hay necesidad de ajuste en las operaciones aritméticas con números BCD (Auxiliar Flag)

  * También hay banderas de control: Registran el funcionamiento del procesador:
  DF = Bandera de dirección. Controla la dirección (hacia adelante o hacia atrás) en las operaciones con cadenas de caracteres incrementando o decrementando automáticamente los registros índices (SI y DI) (Direction Flag)

  IF = Bandera de interrupciones. Indica si están permitidas o no las interrupciones de los dispositivos externos (Interrupt Flag)

  TF = Bandera de traza. Controla la operación modo paso a paso (Trap Flag)

- **Instrucciones básicas:**

MOV

Esta instrucción copia el segundo operando (origen) en el primer operando (destino). Se puede realizar copia directa de los números, o puede ser el registro en el que se encuentra guardado. Ambos operandos deben tener el mismo tamaño, es decir, si el registro de origen es de 8 bits, el registro destino debe ser de 8 bits también y viceversa.

Por ejemplo:

```
mov al, 5; ejemplo sumando un numero
mov bl, al; ejemplo registros 8bits
mov ax, bx; ejemplo registros 16bits
```

ADD

Esta instrucción realiza la suma de los dos operandos, almacenando el resultado en el primero de ellos, con la siguiente sintaxis:

```
add operando1, operando2
```

Debe haber correspondencia entre los tamaños de los registros, es decir, no se puede sumar a un registro de 16, un registro de 8 y viceversa

Ejemplo:

```
add ax, bx ; suma a ax lo que hay en bx y lo almacena en ax
add ax, 8 ; suma a ax el valor de 8 y lo almacena en ax
```

SUB

Esta instrucción realiza una resta entre los operandos. Aparte de eso, sus características y restricciones son las mismas del ADD:

```
sub operando1, operando2
```

- Debe haber correspondencia entre los tamaños de los registros, es decir: no se puede sumar a un registro de 16, un registro de 8 y viceversa
- el resultado se guarda con signo, en caso que el resultado sea un número negativo

Ejemplo:

```
sub ax, 3; Resta 3 a AX y lo guarda en AX
```

MUL

La instrucción MUL realiza multiplicación (sin signo) entre AL y otro registro o un número. El resultado se almacena en AX.

El algoritmo es:

```
AX = AL * registro
```

La sintaxis es:

```
MUL registro
```

Ejemplo:

```
mov dl, 5
mov al, 10
mul dl
-----
```

en ax quedaría por resultado 50

DIV

La instrucción **DIV** realiza la multiplicación (sin signo) entre AX y otro registro, quedando el resultado en AL y el módulo en AH.

El algoritmo es:

```
AL = AX/registro, AH = (modulo).
```

La sintaxis es:

```
div registro
```

Ejemplo:

```
mov ax, 203   ;
mov bl, 4
div bl        ; al = 50 , ah = 3
ret
```

CMP

Para realizar comparaciones lógicas condicionales existe la instrucción CMP:

```
cmp arg1, arg2
```

realiza una substracción arg1 – arg2 la que enciende algunas banderas del registro de estado según el resultado producido. Para cambiar el control del flujo se necesita agregar una de las siguientes instrucciones de salto condicional después de cmp, las siguientes son las instrucciones para saltos condicionales en un intel 8086.

Condición	Sin signo	Con signo
=	JE/JZ	JE/JZ
>	JA/JNBE	JG
<	JB/JNAE	JL
≥	JAE/JNB	JGE
≤	JBE/JNA	JLE
≠	JNE/JNZ	JNE/JNZ

A continuación se presenta un ejemplo que verifica si dos valores son iguales:

```
mov ax, [valor1] ;ax = valor1
mov bx, [valor2] ;bx = valoe2
cmp ax, bx ;ax - bx
je igual ;if (ax==bx) goto igual
```

Si se desea realizar un salto incondicional se debe utilizar la instrucción JMP:

```
jmp salir
```

PUSH y POP

Realizan las operaciones de apilado y desapilado en la pila del procesador respectivamente, admiten todos los tipos de direccionamiento (excepto inmediato). Los operandos deben ser siempre de 16 bits.

Ejemplos:

```
push ax  ; envía a la pila AX
pop ds  ; carga el primer elemento de la pila en DS
```

El Emulador EMU8086

El emulador en su versión 4.08 posee variadas herramientas para el desarrollador. Al iniciar el programa, lo primero que observaras será la pantalla de bienvenida (welcome), similar a la que se muestra en la Figura 1.

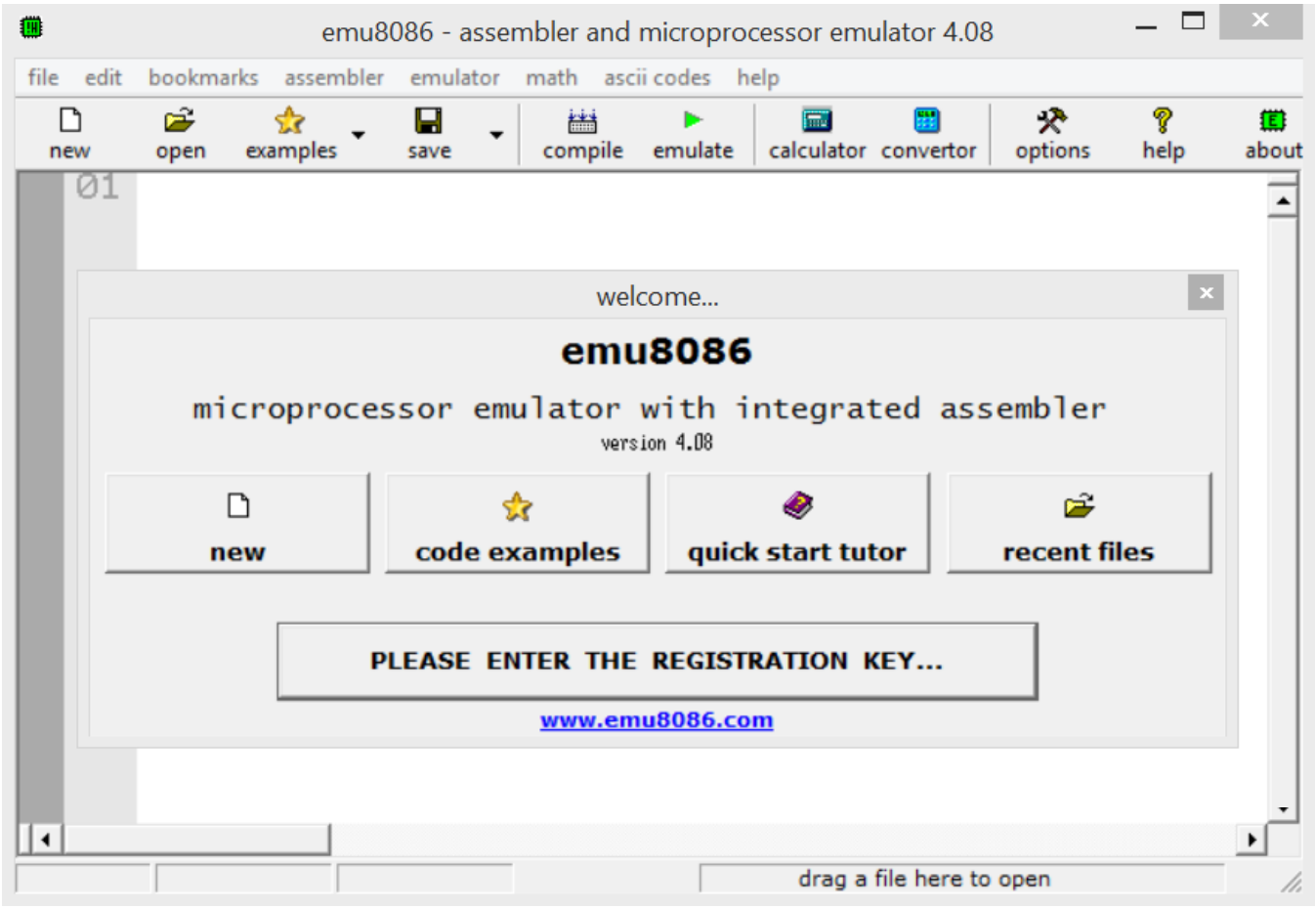


Figura 1. Pantalla de bienvenida del emulador EMU8086.

Se presentan cuatro diferentes opciones para elegir:

- **New:** permite escribir un nuevo código en lenguaje ensamblador (al que llamaremos “Código Fuente” y tendrá extensión .ASM)
- **Code examples:** permite acceder a una serie de programas ejemplos que pueden ayudarle a comprender funciones más complejas.
- **Quick star tutor:** activa un conjunto de documentos de ayuda, se recomienda revisarlos frecuentemente en caso de dudas.
- **Recent file:** muestra los últimos archivos que se trabajaron en la máquina.

Para continuar este primer contacto con el emulador, seleccione New. Observará una nueva caja de dialogo “choose code template”, como se muestra en la Figura 2.

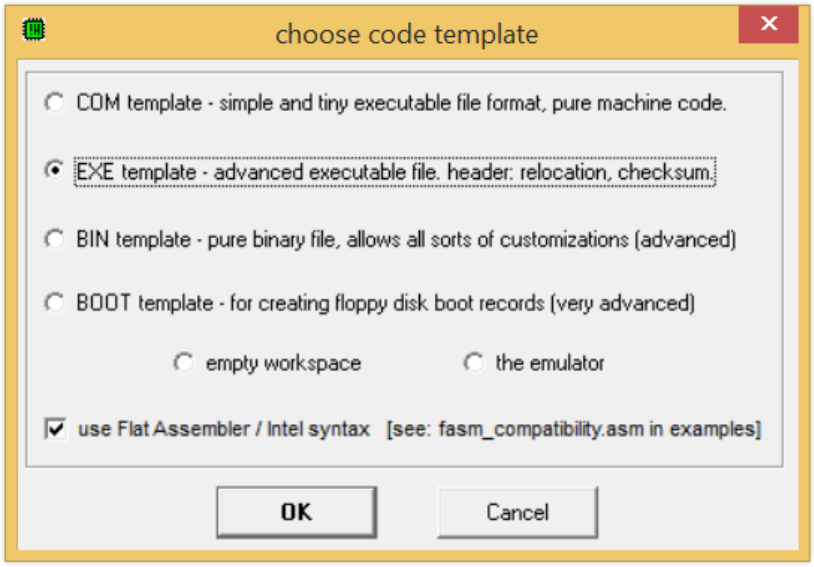


Figura 2. Caja de diálogo para seleccionar el tipo plantilla (template).

En ella se presentan seis opciones, cuatro que permiten usar plantillas predefinidas con algunos de los tipos de archivo que se pueden crear en lenguaje ensamblador: COM template, EXE template, BIN template y BOOT template (cada uno permite diferentes características). Dos que permiten usar un espacio vacío “empty workspace” (sin una plantilla) o activar el emulador mismo.

Selecciona la opción “empty workspace”. Observarás la ventana de edición o mejor dicho el Entorno de Desarrollo Integrado (Integrated Development Environmet IDE), como se muestra en la Figura 3, donde escribirá sus archivos fuentes en lenguaje ensamblador.

Podrá ver una barra de menú de Windows con sus opciones file, edit, etc. pero también verá unas opciones poco usuales como assembler, emulator, etc. propias del emulador.

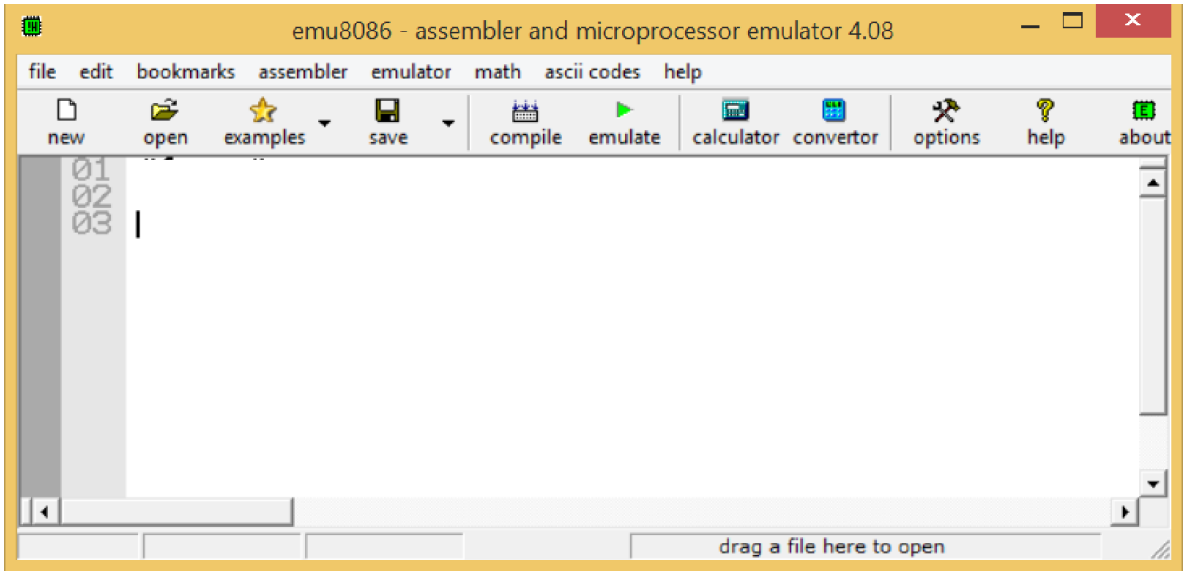


Figura 3. Ventana de edición o Entorno de Desarrollo Integrado IDE del EMU8086.

Bueno, es el momento de estudiar nuestro primer programa en lenguaje ensamblador, el cual imprime en pantalla Hola Mundo!!.

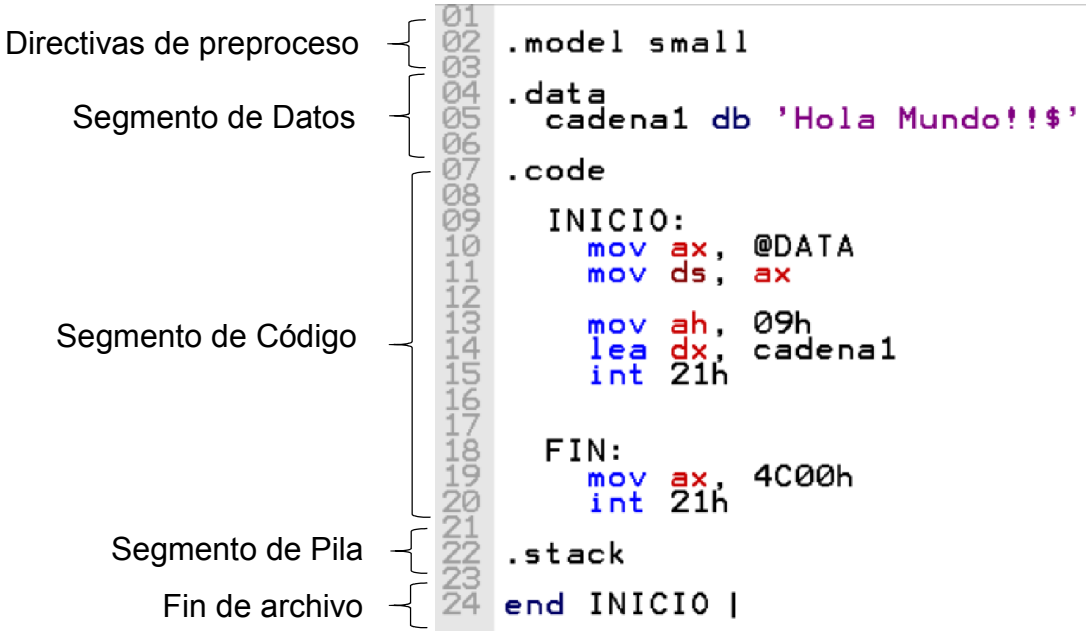


Figura 4. Código del programa holamundo.asm

Por favor note que el programa está compuesto de diferentes bloques:

a) Directivas de preproceso: Le indican al compilador que debe realizar una serie de acciones particulares en el momento de convertir el archivo fuente (ASM) en archivo objeto (OBJ).

b) Segmento de Datos: Donde se declaran las variables y constantes que el programa va a utilizar.

c) Segmento de Código: Donde especifica lo que deseamos que el programa haga. Para especificarlo se pueden usar: instrucciones (propias del microprocesador), Macro-instrucciones (similar a los comandos de los lenguajes de alto nivel) y procedimientos (similar a las funciones definidas por el usuario de los lenguajes de alto nivel).

d) Segmento de PILA o STACK: bloque de memoria donde almacenan datos intermedios que se generan durante la ejecución de un programa. En este no se declaran variables o constantes como en el segmento de datos, sino que se administra como una memoria LIFO, el último en entrar es el primero en salir.

e) Directiva que indica el fin del archivo, cualquier instrucción posterior a esta línea será ignorada.

Digite el código en el IDE, note como se le asignan diferentes colores a las líneas, dependiendo si son instrucciones, macro-instrucciones, comentarios, cadenas de texto (formato ASCII), directivas, etc.

Al terminar de digitar el código salve el archivo usando como nombre su número de carnet (8 caracteres) y con extensión ASM (ensamblador).

Emule el archivo EXE presionando el botón “emulate”. Notará que se abren dosventanas: Una llamada “emulator” en la que podrá monitorear la mayoría de procesos al ejecutar el programa y otra llamada “original sourse code” que muestra el código fuente del archivo, esta ventana es diferente de la que observa en el IDE porque en ésta podrá observar cual instrucción está a punto de ejecutarse, es ideal al correr programas pasos a paso.

Ahora observe con más detenimiento la ventana llamada “emulator” Figura 5, ésta será la que más utilice a la hora de ejecutar las prácticas de laboratorio, por lo que es importante que la conozca y maneje de forma efectiva.

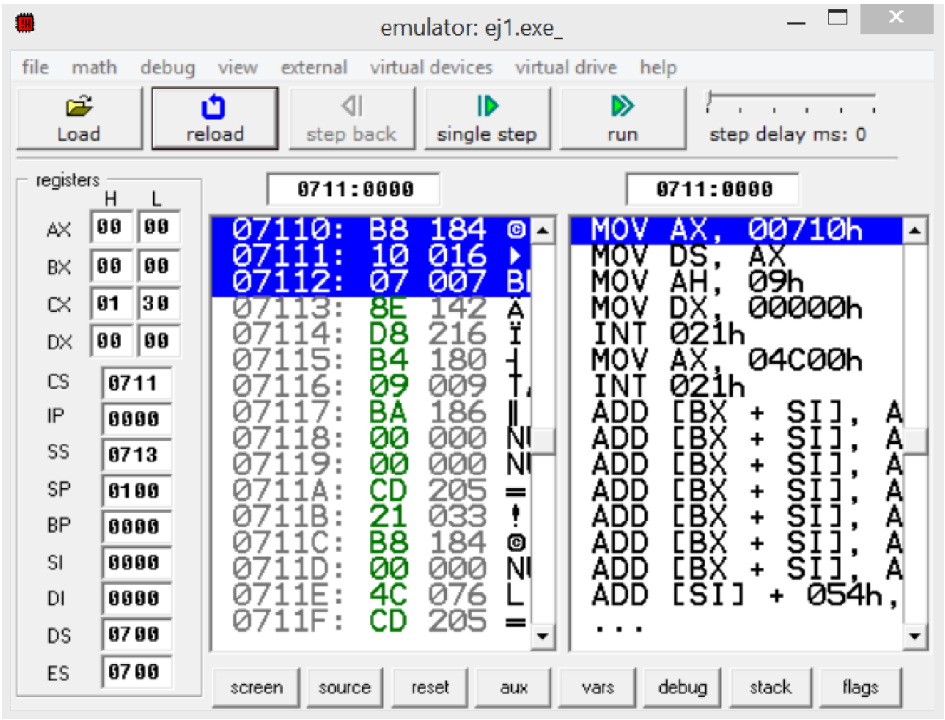


Figura 5. Ventana del Emulador.

En la parte superior tiene una barra de herramientas con las siguientes opciones:

- File, permite administrar (cargar o salvar) los archivos que va creando oejecutando
- Math, da acceso a una calculadora y un convertidor de basas de numeración.
- Debug, provee herramientas para depurar programas.
- View, permite abrir otras ventanas que pueden ser de mucha ayuda al ejecutar o depurar programas.
- External, permite ejecutar el programa con otras herramientas diferentes delEMU8086.

- Virtual devices, activa los dispositivos virtuales con que cuenta el programa, dado que se tata de un emulador no se tiene acceso a los puertos físicos de la computadora, por lo que estos son simulados.

- Virtual drive, da opciones para administrar las unidades virtuales dealmacenamiento (HDD y FDD virtuales).

- Help, activa la herramienta de ayuda.

Bajo la barra de herramientas hay cinco botones que le permiten:

- Load, carga un archivo ejecutable EXE, COM, etc. que ya existe.

- Reload, recarga (inicializa los registros) para ejecutar nuevamente un programa que acaba de ser corrido.

- Single step, ejecuta solamente una instrucción o macroinstrucción (paso apaso).

- Step back, retrocede una instrucción que ya fue ejecutada (función muy útil al depurar un programa)

- Run, ejecuta un programa en su totalidad o hasta presionar el botón “STOP”.Vale la pena hacer notar que también es posible, en la opción DEBUG, insertar un “break point” cuando se está depurando programas.

La parte media está dividida en tres partes:

- Parte izquierda denominada “registers”, donde se puede monitorear el contenido de los registros del microprocesador.

- Parte central, donde se puede observar el contenido de la memoria desde donde se está ejecutando el programa. Primero se notan las direcciones del bloque de memoria que se visualiza, estas direcciones de dan en un formato llamado físico o absoluto (existe otro formato para las direcciones) dado por cinco cifras hexadecimales (20 bits) lo que indica que en este bus de direcciones se puede direccionar desde la dirección 00000h (dirección 0) hasta la dirección FFFFFh (dirección 148575). Luego se indica el contenido de cada palabra (cada una de1 byte), por facilidad el contenido se presenta en hexadecimal, decimal e interpretado como un carácter ASCII.

- Parte derecha, donde puede observar el contenido de la memoria, pero esta vez no se detalla con direcciones específicas, sino que cada bloque de datos es interpretado como un conjunto de instrucciones (lo que llamaremos programa DESENSAMBLADO) que el microprocesador deberá ejecutar. Es importante mencionar que algunas instrucciones se expresan solo con un byte, pero otras necesitan varios bytes para ser expresadas.

Parte inferior, contiene una serie de botones que permiten un acceso rápido a una serie de ventanas auxiliares, algunas de las cuales se puede activar también en la barra de herramientas con la opción “view”.

Regresando a la estructura del programa estudiemos el Segmento de Datos: Puede observar que se ha declarado una cadena de datos llamada cadena1; note que luego del nombre de la cadena se ha incluido la directiva db (definir byte) que indica al compilador que ese es el tipo de datos que contendrá la cadena. A parte de eso se ha usado comillas simples para definir el contenido, eso indica al compilador que el texto debe ser interpretado como caracteres ASCII.

Para ejecutar el programa presione el botón “run”: note que automáticamente se activa la ventana “emulator screen” en la que se puede observar el texto impreso, que corresponde a cadena1. Vamos a estudiar más detenidamente la ejecución del programa.

Cierre la ventana “emulator screen”

Prepare el programa para ser ejecutado nuevamente presionando el botón “reload”. Observe la ventana “original source code”, parece que es una copia fiel del código fuente que digitó en el IDE, pero la primera línea de código está marcada con color de fondo diferente, esto indica que es la primera que se ejecutará al correr el programa.

```
01
02 .model small
03
04 .data
05 cadena1 db 'Hola Mundo!!$'
06
07 .code
08
09 INICIO:
10 mov ax, @DATA
11 mov ds, ax
12
13 mov ah, 09h
14 lea dx, cadena1
15 int 21h
16
17
18 FIN:
19 mov ax, 4C00h
20 int 21h
21
22 .stack
23
24 end INICIO
```

Figura 6. Ventana de código fuente.

Si observa la ventana “emulator” (Figura 5), se ha marcado un bloque de bytes almacenados en memoria, precisamente los de las direcciones físicas 07110h, 07111hy 07112h que contiene los datos B8h (que equivale a 184d y al carácter ASCII ‘@’), los otros datos son 10h y 07h. Los tres juntos equivalen, en lenguaje de máquina, a la MOV AX, @DATA, que ya desensamblada se transforma en mov AX, 0710h, Aparece marcada porque será la primera en ejecutarse. Un detalle importante es la sustitución de la expresión @DATA por un valor numérico, esta sustitución es producto de la compilación del código fuente que sustituye las etiquetas y nombres de variables por direcciones físicas de memoria. Presione el botón “single step” para continuar ejecutando instrucción por instrucción.