ENS2001

MANUAL DE USUARIO

versión 1.1 – Febrero 2003

0. ÍNDICE

1. INTRODUCCIÓN

ENS2001 es una aplicación que integra la función de **Ensamblador** de un subconjunto de instrucciones del estándar *IEEE 694* [De Miguel 1996] y la función de **Simulador**, ya que es capaz de ejecutar programas ensamblados para dicha implementación particular del estándar. Se trata de una mejora de las herramientas disponibles hasta ahora, como son *ASS* y *ENS96*, a las que añade una arquitectura de máquina virtual mejorada, una implementación del lenguaje más homogénea y, principalmente, una nueva y cómoda interfaz gráfica.

El **Ensamblador** permite cargar desde un fichero el código fuente que será ejecutado, informando de la corrección del mismo o de los errores que pudiera contener, según el caso, y como se puede ver en la Figura 1.1.

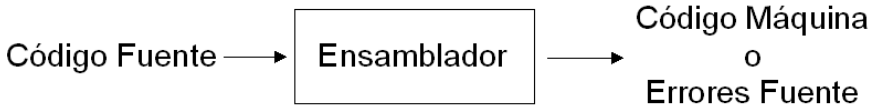

Figura 1.1. El Ensamblador de *ENS2001*

El **Simulador**, aparte de las diversas opciones de configuración y ejecución, proporciona las típicas funciones de acceso a memoria, pila, banco de registros, código fuente y consola, tanto para consultar datos como para alterarlos manualmente. En la Figura 1.2 se muestra el diagrama de bloques simplificado para esta parte de la herramienta.

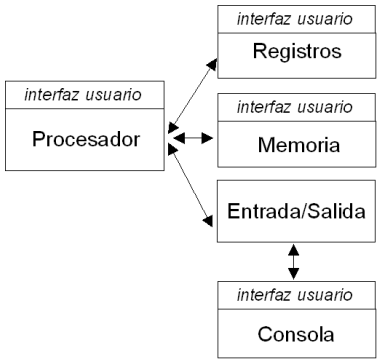

Figura 1.2. El Simulador de *ENS2001*

En principio, la aplicación está dirigida a los alumnos de Compiladores, que deberán probar el código generado por sus respectivas prácticas, si bien también puede ser de utilidad, sobre todo didáctica, para todo aquél que esté interesado en iniciarse en el mundo de la programación en ensamblador, sin profundizar en ningún lenguaje para ningún microprocesador en concreto. En esta herramienta hallará el complemento ideal para comprobar en un entorno simulado la ejecución de los programas que vaya creando durante las etapas de aprendizaje.

A continuación se describe el sistema que la herramienta simula y el lenguaje ensamblador ideado para dicho sistema. Posteriormente se ahondará en los detalles acerca de las características de las dos versiones disponibles de la herramienta, entorno gráfico y consola. En cada apartado se describe completamente cada versión. De esta manera se evita al usuario tener que leer ambos. Si está interesado únicamente en la versión gráfica,

puede ignorar el Apartado 9, mientras que si está interesado solamente en la versión textual, puede ignorar el Apartado 8.

La herramienta se distribuye en tres ficheros comprimidos que contienen cada una de las tres distintas versiones disponibles de manera independiente. Los ficheros están comprimidos en formato zip para las versiones de Windows y en un tar.gz para la de Linux.

No es necesario ningún tipo de instalación. Basta con descomprimir el fichero de la versión que se desee emplear en un directorio cualquiera del disco duro. También se adjuntan el manual de usuario y algunos programas de ejemplo. Además, la versión para consola Linux se distribuye con el código fuente.

2. MÁQUINA VIRTUAL

La Máquina Virtual que simula la aplicación posee las siguientes características:

- **Procesador** con ancho de palabra de 16 bits.
- **Memoria** de 64 Kpalabras (de 16 bits cada una). Por tanto, el direccionamiento es de 16 bits, coincidiendo con el ancho de palabra, desde la dirección 0 a la 65535 (FFFFh).
- **Banco de Registros**. Todos ellos son de 16 bits.
  - **PC** (Contador de Programa): Indica la posición en memoria de la siguiente instrucción que se va a ejecutar.
  - **SP** (Puntero de Pila): Indica la posición de memoria donde se encuentra la cima libre de la pila.
  - **SR** (Registro de Estado): Almacena el conjunto de los biestables de estado, según la Tabla 2.1.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	H	S	P	V	C	Z

Tabla 2.1. Biestables de Estado

  - **IX**, **IY** (Registros Índices): Se emplean para efectuar direccionamientos relativos.
  - **A** (Acumulador): Almacena el resultado de las operaciones aritméticas y lógicas de dos operandos.
  - **R0**..**R9** (Registros de Propósito General): Son registros cuyo uso decidirá el programador en cada momento. Los registros se codifican internamente con 4 bits, valores de 0 a 15, según se muestra en la Tabla 2.2.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PC	SP	IY	IX	SR	A	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0

Tabla 2.2. Registros del Simulador

- **Biestables de estado:** Se actualizan después de que el procesador ha ejecutado una operación aritmética o de comparación. Su significado y condiciones de activación son los siguientes:

  - **Z (cero).** Si el resultado de la operación es 0, se activa. En caso contrario, se desactiva.
  - **C (acarreo).** Si el resultado excede los 16 bits de longitud, se activa. En caso contrario, se desactiva.
  - **V (desbordamiento)**. Si el resultado de la operación excede el rango de representación de los enteros de 16 bits en complemento a 2 (desde –32768 hasta 32767), se activa. En caso contrario, se desactiva.
  - **P (paridad)**. Es el resultado de realizar un O Lógico Exclusivo con todos los bits del resultado. Así, se activa si el número de bits que valen 1 es impar y se desactiva si el número de bits que valen 1 es par.
  - **S (signo)**. Se activa cuando el resultado de la operación es negativo y se desactiva si es positivo. En otras palabras, indica el valor del bit más significativo del resultado de la operación (como la ALU opera en complemento a 2, será 0 para valores positivos y 1 para valores negativos).
  - **H (fin de ejecucion)**. Se activa únicamente después de que el procesador haya ejecutado una instrucción de parada (`HALT`).

- **Entrada/Salida** por consola: Se proveen instrucciones para leer y escribir enteros, caracteres y cadenas de caracteres acabadas en el carácter nulo (‘\0’), según el estilo convencional del lenguaje *C*.

- **Unidad Aritmético-Lógica** sobre enteros en complemento a 2 de 16 bits: El juego de instrucciones proporciona operaciones de suma, resta, multiplicación, división, módulo, cambio de signo, incremento, decremento, *or* lógico, *and* lógico, *or* exclusivo lógico y negación lógica.

  Al operar en complemento a 2, los enteros negativos (en el rango –32768..–1) se transforman en los correspondientes positivos (32768..65535 respectivamente). Por lo tanto, al introducirlos, bien en el código fuente, bien durante la ejecución, serán tratados indistintamente por la herramienta, tanto en el ensamblador como durante la simulación.

- Generación de **Excepciones**: Durante la simulación se pueden producir las siguientes condiciones de excepción, que detendrán la ejecución:

  - *Instrucción no implementada*. Ocurre cuando el procesador lee un código de operación que no corresponde con ninguna instrucción de su juego de instrucciones. Esto puede ocurrir si el usuario introduce código directamente editando la memoria, o no se posiciona el contador de programa adecuadamente antes de lanzar la ejecución, por ejemplo, en zonas de datos o en una dirección de memoria intermedia dentro de una instrucción que ocupe varias posiciones.
  - *División por cero*. Ocurre cuando el segundo operando de una operación de división (el divisor) toma el valor cero.
  - *Sobrepasado el límite de la memoria*. Ocurre cuando, tras ejecutar una instrucción, el Contador de Programa toma un valor más alto que el límite superior de memoria. O bien cuando se está ejecutando una instrucción `INSTR` o `WRSTR` y la cadena sobrepasa dicho límite.

- *Contador de programa invade la zona de pila*. Esta comprobación es opcional, se puede activar o desactivar a voluntad del usuario. Cuando está activada, se genera una excepción si PC va a tomar un valor comprendido dentro de los límites de la pila del sistema.
- *Puntero de pila invade la zona de código*. Esta comprobación es opcional, se puede activar o desactivar a voluntad del usuario. Cuando está activada, se genera una excepción si SP va a tomar un valor comprendido dentro de los límites del código almacenado en memoria.
- *Ejecución detenida por el usuario*. El usuario puede detener la simulación en cualquier momento, generándose esta excepción.

3. ESTRUCTURA DEL CÓDIGO FUENTE

El código fuente se lee desde un **fichero de texto plano**. Esta es la forma común de alimentar a la aplicación de programas para ejecutar, si bien es posible introducir el código ensamblado a mano modificando directamente las posiciones de memoria, lo cual resulta bastante más tedioso y peligroso ya que no se comprobará si se está cometiendo algún error (además no se actualizarán los límites de la zona de código). En la Figura 3.1 se muestra la arquitectura del módulo ensamblador. Se parte de un fichero donde está almacenado el código fuente y se genera en dos pasadas el código máquina, o bien una lista de errores en el programa introducido.

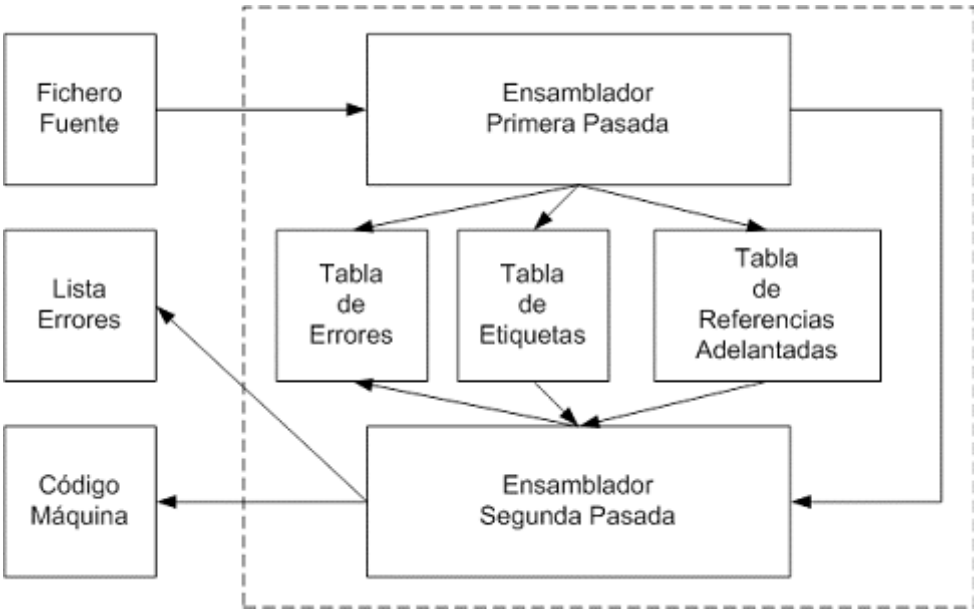

Figura 3.1. Arquitectura del Ensamblador

El código fuente se compone de una sucesión de líneas que obedecen al siguiente formato:

```
[[Etiqueta ':'] Instrucción] [';' Comentario]
```

No existe límite para el número de caracteres que pueda ocupar una línea (salvo las restricciones propias del sistema operativo o la memoria disponible). Son perfectamente posibles combinaciones tales como introducir líneas en blanco, comentarios tras las instrucciones, líneas únicamente con comentarios, etc.

Además, excepcionalmente, se permite introducir saltos de línea y comentarios entre etiquetas e instrucciones, en aras de aumentar la legibilidad del código. La única restricción "real" es que cada instrucción estará contenida en una única línea, ni más ni menos (es decir, una instrucción sólo ocupa una línea y en cada línea sólo hay una instrucción como mucho). Por ejemplo:

```
BZ .R3 ; saltar si el contador es cero          Correcto
; esta línea sólo tiene comentarios             Correcto
ADD #300 ; primer op , [.R0] ; segundo op       Erróneo
```

El ensamblador distingue entre mayúsculas y minúsculas (lo cual tiene especial importancia a la hora de definir etiquetas). En los casos especiales de las instrucciones y

los identificadores de registros, serán válidos en uno u otro formato (pero sin mezclar ambos). Por ejemplo:

`NOP`	Correcto
`nop`	Correcto
`Nop`	Erróneo (se tomará como una etiqueta)

El formato de las instrucciones es el siguiente:

```
Mnemónico [Operando1 [Operando2]]
```

Un mnemónico es una **palabra** o una **abreviatura** en inglés que hace referencia a la operación.

Los operandos pueden ser **enteros**, **nombres de registros** o **etiquetas**. En cualquier caso, se deben ajustar a alguno de los modos de direccionamiento que se analizan en el siguiente apartado. No todas las instrucciones admiten todos los modos de direccionamiento posibles. Las combinaciones permitidas se verán al estudiar con detalle el juego de instrucciones y pueden consultarse en la Tabla 10.5.

4. MODOS DE DIRECCIONAMIENTO

El estándar *IEEE 694* define un conjunto de modos de direccionamiento para determinar los distintos operandos o su ubicación. El modelo simulado por *ENS2001* permite **siete** de ellos (aunque en realidad son seis, pero uno de ellos se desdobla por comodidad). Dependiendo de cada instrucción, se pueden usar unos u otros en los operandos. Los modos de direccionamiento son los siguientes:

4.1. DIRECCIONAMIENTO INMEDIATO

El direccionamiento es inmediato cuando el operando se encuentra **contenido** en la propia instrucción. Por tanto, dadas las características de la máquina virtual, internamente serán necesarios 16 bits para este tipo de direccionamiento.

Para indicar este modo de direccionamiento, se empleará el carácter **'#'** (almohadilla) precediendo al operando, que en este caso indicará un valor entero de 16 bits en complemento a 2.

Se pueden introducir valores decimales con signo, en el intervalo comprendido entre –32768 y 32767, o bien sin signo en el rango 0..65535. También se permite introducir valores hexadecimales, precedidos por el prefijo '0x', en el rango 0..FFFFh.

Ejemplos:

- La instrucción de carga `MOVE #1000,.R1`, transfiere el valor 1000d al registro R1, como se muestra en la Figura 4.1.1.

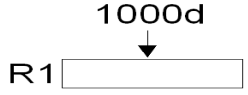

Figura 4.1.1. Direccionamiento Inmediato (ejemplo 1)

- La instrucción de suma `ADD .R3,#0x00FF` suma el contenido de R3 con el valor FFh y lo almacena en el acumulador, tal y como se aprecia en la Figura 4.1.2.

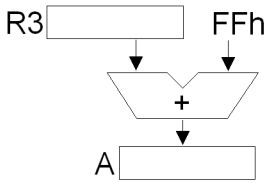

Figura 4.1.2. Direccionamiento Inmediato (ejemplo 2)

4.2. Direccionamiento Directo a Registro

El direccionamiento es directo a registro cuando expresa la **dirección** (en este caso el **nombre del registro**) donde se encuentra almacenado el objeto. Dado el banco de registros definido para la máquina virtual, internemente serán necesarios 4 bits para este tipo de direccionamiento (ya que el banco de registros contiene 16 registros en total).

Para indicar este modo de direccionamiento, se empleará el carácter **'.'** (punto) precediendo al operando, que en este caso será el nombre de un registro del banco de registros.

Por ejemplo, en la instrucción de resta `SUB .R3, #0xFF`, el primer operando emplea un direccionamiento directo a registro, ya que se está indicando el nombre del registro donde se encuentra almacenado. El segundo operando emplea direccionamiento inmediato, de la misma forma que el ejemplo anterior (Figura 4.1.2). Dicha instrucción resta al contenido del registro R3 el valor inmediato FFh, y lo almacena en el acumulador.

4.3. Direccionamiento Directo a Memoria

El direccionamiento es directo a memoria cuando se expresa la **dirección absoluta de memoria** donde se encuentra almacenado el operando. Como la memoria consta de 64 KPalabras, internamente se necesitan 16 bits para este tipo de direccionamiento.

Para indicar este modo de direccionamiento, se empleará el carácter **'/'** (barra) precediendo al operando, que en este caso indicará una dirección de memoria. Como en el caso del direccionamiento inmediato, se permiten valores decimales con signo, en el intervalo –32768..32767 o bien sin signo en el rango 0..65535. También se permite introducir valores hexadecimales, precedidos por el prefijo '0x', en el rango 0..FFFFh. En el caso de que se introduzcan decimales con signo, se traducirán a la dirección que se correspondería con su representación en complemento a 2 (en 16 bits); por ejemplo, -1 se corresponde con 65535, -2 con 65534 y así sucesivamente.

Por ejemplo, para la instrucción de incremento `INC /0x1000`, el resultado será que el valor contenido en la posición de memoria 1000h se habrá incrementado en una unidad, tal y como se muestra en la Figura 4.3.1.

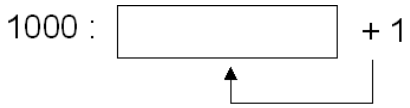

Figura 4.3.1. Direccionamiento Directo a Memoria

En el caso de haber introducido `INC /-30`, por ejemplo, el ensamblador haría la conversión pertinente, y el operando se correspondería con la celda de memoria ubicada en la dirección 65506 (que es la representación en complemento a 2 con 16 bits del valor –30). Por tanto, dicha instrucción es la misma que `INC /65506`.

4.4. Direccionamiento Indirecto

Se trata de un direccionamiento directo, pero del que no se obtiene el operando, sino la **dirección donde se encuentra el operando**. Dicha dirección se encuentra almacenada en un registro. Por tanto, se requerirá un acceso a memoria adicional (el primero al banco de registros y el segundo a memoria principal) para recuperar el objeto. Así las cosas, internamente se requieren 4 bits para indicar cuál es el registro a partir del que se recupera la dirección de memoria destino.

Para indicar este modo de direccionamiento, se encierra entre corchetes **'[ ]'** el identificador del registro sobre el que se efectúa la indirección en notación de registro, esto es, precedido por el carácter '**.**' (punto).

Por ejemplo, la instrucción de decremento `DEC [.R3]` decrementará en una unidad el contenido de la posición de memoria contenida en el registro R3. Se puede ver con claridad en la Figura 4.4.1.

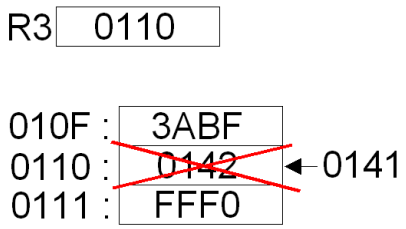

Figura 4.4.1. Direccionamiento Indirecto

4.5. Direccionamiento Relativo a Registro Índice

El direccionamiento es relativo a un registro índice cuando la instrucción contiene un **desplazamiento** sobre una **dirección** marcada por un **puntero**. La dirección del operando, por tanto, se calcula sumando el desplazamiento al puntero de referencia.

Para indicar este modo de direccionamiento, se empleará el carácter **'#'** (almohadilla) precediendo al operando, que en este caso indicará un desplazamiento (entero de 8 bits en complemento a 2, esto es, valores comprendidos entre –128 y 127), y detrás, en **notación de indirección '[ ]'** (entre corchetes), el nombre del registro índice. El valor del desplazamiento se puede introducir, como cualquier otro entero, bien en base decimal, bien en base hexadecimal, siguiendo las mismas consideraciones que en los casos anteriores de direccionamiento inmediato y directo a memoria, atendiendo al detalle adicional de que en este caso sólo se permiten valores de 8 bits.

ENS2001 incluye dos registros IX e IY que actúan como índices, y se van a permitir desplazamientos restringidos a enteros de 8 bits en complemento a 2. Por tanto, internamente, este tipo de direccionamiento requiere de 9 bits, 8 para el desplazamiento y uno adicional para indicar el registro índice. A efectos prácticos, se considerarán el relativo a IX y el relativo a IY como dos modos de direccionamiento distintos.

Ejemplos:

- En la instrucción de carga `MOVE #6[.IX],.R1`, se carga en el registro R1 el contenido de la posición de memoria a la que hace referencia el índice IX más 6 posiciones (Figura 4.5.1).

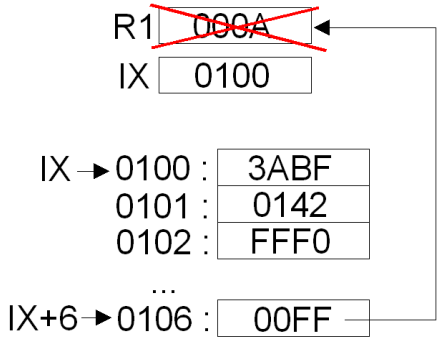

Figura 4.5.1. Direccionamiento Relativo a Índice

- Las siguientes instrucciones son análogas: `DEC #-10[.IY]`; `DEC #0xF6[.IY]`; `DEC #246[.IY]`, debido a la equivalencia entre las representaciones decimal y hexadecimal en complemento a 2.

4.6. Direccionamiento Relativo a Contador de Programa

Se trata de un caso particular de direccionamiento directo relativo. El registro **puntero** es el **contador de programa (PC)**, y se indica el desplazamiento, que como en el caso general se trata de un valor entero de 8 bits en complemento a 2. Por tanto, este modo de direccionamiento requiere 8 bits para su representación interna.

Así, la dirección del objeto, en este caso, la próxima instrucción que se va a ejecutar, se obtiene sumando el desplazamiento al registro puntero. En la práctica, el desplazamiento indica una cantidad entera que se va a sumar al contador de programa para reubicarlo antes de la ejecución de la siguiente instrucción.

Para indicar este modo de direccionamiento, se empleará el carácter **'$'** (símbolo de dólar) precediendo al operando, que se trata del desplazamiento. El formato del desplazamiento es idéntico al caso general (enteros comprendidos entre –128 y 127).

Por ejemplo, en la instrucción de salto incondicional `BR $3` de la Figura 4.6.1, la siguiente instrucción que ejecutará el procesador se encuentra en la posición de memoria a la que apunte el contador de programa más 3 posiciones. Hay que recordar que el contador de programa apunta siempre a la siguiente instrucción de la que se está ejecutando.

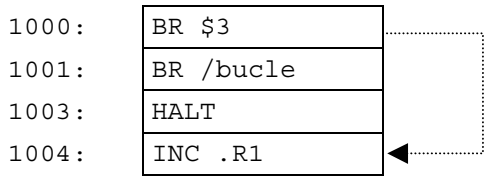

Figura 4.6.1. Direccionamiento Relativo a PC

La siguiente instrucción que se ejecutaría sería `INC .R1` (y no `HALT` como cabría esperar, ya que en el momento de ejecutar `BR $3`, PC vale 1001, así que PC = 1001+3).

En las instrucciones de salto es muy frecuente el empleo de etiquetas, tanto si el direccionamiento es directo a memoria como relativo a contador de programa. No obstante, en el caso de direccionamiento relativo, la etiqueta no se traduce por su valor, si no que se calcula el desplazamiento entre la posición a la que apuntaría PC en el caso de seguir el flujo de ejecución y la posición que ocupa la etiqueta. Por ello, si se introduce en el código fuente un direccionamiento relativo a contador de programa hacia una etiqueta que está alejada más de 128 posiciones de memoria (127 hacia adelante, 128 hacia atrás, cantidades representables con 8 bits), el ensamblador devolverá un error en tiempo de compilación, ya que no puede calcular un desplazamiento válido.

El caso anterior, representado en la Figura 4.6.2, pero empleando etiquetas en vez de números enteros, sería el siguiente. La etiqueta `SEGUIR` tomaría el valor 1004, y el valor del desplazamiento seguiría siendo 3.

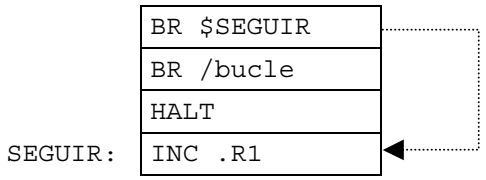

Figura 4.6.2. Direccionamiento Relativo a PC con etiquetas

5. JUEGO DE INSTRUCCIONES

A continuación se enumeran todas las instrucciones que componen el juego de instrucciones, ordenadas por código de operación. En la Tabla 10.5 se puede consultar la relación completa de instrucciones y los modos de direccionamiento que soportan.

Instrucción	NOP
Descripción	Instrucción de no operación.
Formato	NOP
Código de Operación	0
Número de Operandos	0
Modos de Direccionamiento	N/A
Comportamiento	Ninguno No modifica los biestables de estado.

Instrucción	HALT
Descripción	Detener la ejecución de la máquina.
Formato	HALT
Código de Operación	1
Número de Operandos	0
Modos de Direccionamiento	N/A
Comportamiento	Activa el biestable H de fin de programa y detiene el procesador virtual

TRANSFERENCIA DE DATOS.

Instrucción	MOVE
Descripción	Copiar operando origen en operando destino.
Formato	MOVE op1, op2
Código de Operación	2
Número de Operandos	2
Modos de Direccionamiento	op1: inmediato, registro, memoria, indirecto, relativo op2: registro, memoria, indirecto, relativo Para el operando 2 no se permite direccionamiento inmediato, ya que no tiene sentido como destino de un movimiento de datos.
Comportamiento	Lee el contenido del operando 1 (origen) y lo escribe en el operando 2 (destino). No modifica los biestables de estado.

Instrucción	PUSH
Descripción	Poner en la pila.
Formato	PUSH op1
Código de Operación	3
Número de Operandos	1
Modos de Direccionamiento	op1: inmediato, registro, memoria, indirecto, relativo
Comportamiento	Depende del modo de funcionamiento de la pila: • Para crecimiento hacia direcciones descendentes de memoria, almacena el contenido del operando 1 en la dirección apuntada por SP y decrementa el valor del puntero de pila. • Para crecimiento hacia direcciones ascendentes de memoria, incrementa el valor del puntero de pila y almacena el contenido del operando 1 en la dirección apuntada por SP. No modifica los biestables de estado.

Instrucción	POP
Descripción	Sacar de la pila.
Formato	POP op1
Código de Operación	4
Número de Operandos	1
Modos de Direccionamiento	op1: registro, memoria, indirecto, relativo Para el operando no se permite direccionamiento inmediato, ya que no tiene sentido como destino de un movimiento de datos.
Comportamiento	Depende del modo de funcionamiento de la pila: • Para crecimiento hacia direcciones descendentes de memoria, incrementa el valor del puntero de pila y almacena en el operando 1 el valor contenido en la dirección de memoria apuntada por SP. • Para crecimiento hacia direcciones crecientes de memoria, almacena en el operando 1 el valor contenido en la dirección de memoria apuntada por SP y decrementa el valor del puntero de pila. No modifica los biestables de estado.

ARITMÉTICAS.

De dos operandos.

Instrucción	ADD
Descripción	Suma de números enteros.
Formato	ADD op1, op2
Código de Operación	5
Número de Operandos	2
Modos de Direccionamiento	op1: inmediato, registro, memoria, indirecto, relativo op2: inmediato, registro, memoria, indirecto, relativo
Comportamiento	Suma el contenido del operando 1 y el operando 2, y almacena el resultado en el registro acumulador. Modifica los biestables de estado.

Instrucción	SUB
Descripción	Resta de números enteros.
Formato	SUB op1, op2
Código de Operación	6
Número de Operandos	2
Modos de Direccionamiento	op1: inmediato, registro, memoria, indirecto, relativo op2: inmediato, registro, memoria, indirecto, relativo
Comportamiento	Resta el contenido del operando 1 y el operando 2, y almacena el resultado en el registro acumulador. Modifica los biestables de estado.

Instrucción	MUL
Descripción	Producto de números enteros.
Formato	MUL op1, op2
Código de Operación	7
Número de Operandos	2
Modos de Direccionamiento	op1: inmediato, registro, memoria, indirecto, relativo op2: inmediato, registro, memoria, indirecto, relativo
Comportamiento	Multiplica el contenido del operando 1 y el operando 2, y almacena el resultado en el registro acumulador. Modifica los biestables de estado.

Instrucción	DIV
Descripción	Cociente de la división de números enteros.
Formato	DIV op1, op2
Código de Operación	8
Número de Operandos	2
Modos de Direccionamiento	op1: inmediato, registro, memoria, indirecto, relativo op2: inmediato, registro, memoria, indirecto, relativo
Comportamiento	Divide el contenido del operando 1 y el operando 2, y almacena el cociente de la operación en el registro acumulador. Si el operando 2 contiene el valor 0, se genera una excepción de tipo “división por cero”. Modifica los biestables de estado.

Instrucción	MOD
Descripción	Resto de la división de números enteros.
Formato	DIV op1, op2
Código de Operación	9
Número de Operandos	2
Modos de Direccionamiento	op1: inmediato, registro, memoria, indirecto, relativo op2: inmediato, registro, memoria, indirecto, relativo
Comportamiento	Divide el contenido del operando 1 y el operando 2, y almacena el resto de la operación en el registro acumulador. Si el operando 2 contiene el valor 0, se genera una excepción de tipo "división por cero". Modifica los biestables de estado.

De un operando.

Instrucción	INC
Descripción	Incremento unitario.
Formato	INC op1
Código de Operación	10
Número de Operandos	1
Modos de Direccionamiento	op1: registro, memoria, indirecto, relativo
Comportamiento	Incrementa el contenido del operando en una unidad. Modifica los biestables de estado.

Instrucción	DEC
Descripción	Decremento unitario.
Formato	DEC op1
Código de Operación	11
Número de Operandos	1
Modos de Direccionamiento	op1: registro, memoria, indirecto, relativo
Comportamiento	Decrementa el contenido del operando en una unidad. Modifica los biestables de estado.

Instrucción	NEG
Descripción	Cambio de signo.
Formato	NEG op1
Código de Operación	12
Número de Operandos	1
Modos de Direccionamiento	op1: registro, memoria, indirecto, relativo
Comportamiento	Cambia de signo el operando. Modifica los biestables de estado.

Comparaciones.

Instrucción	CMP
Descripción	Comparación.
Formato	CMP op1, op2
Código de Operación	13
Número de Operandos	2
Modos de Direccionamiento	op1: inmediato, registro, memoria, indirecto, relativo op2: inmediato, registro, memoria, indirecto, relativo
Comportamiento	Resta el contenido del operando 1 y el operando 2 (pero no almacena el resultado de la operación en ningún sitio). Modifica los biestables de estado.

LÓGICAS.

De dos operandos.

Instrucción	AND
Descripción	Y Lógico.
Formato	AND op1, op2
Código de Operación	14
Número de Operandos	2
Modos de Direccionamiento	op1: inmediato, registro, memoria, indirecto, relativo op2: inmediato, registro, memoria, indirecto, relativo
Comportamiento	Efectúa la operación 'y lógico' bit a bit entre el contenido del operando 1 y el operando 2, y almacena el resultado en el registro acumulador. La tabla de verdad de la operación es: op1 AND op2 | 0 | 1 0 | 0 | 0 1 | 0 | 1 No modifica los biestables de estado.

Instrucción	OR
Descripción	O Lógico.
Formato	OR op1, op2
Código de Operación	15
Número de Operandos	2
Modos de Direccionamiento	op1: inmediato, registro, memoria, indirecto, relativo op2: inmediato, registro, memoria, indirecto, relativo
Comportamiento	Efectúa la operación 'o lógico' bit a bit entre el contenido del operando 1 y el operando 2, y almacena el resultado en el registro acumulador. La tabla de verdad de la operación es: op1 OR op2 | 0 | 1 0 | 0 | 1 1 | 1 | 1 No modifica los biestables de estado.

Instrucción	XOR
Descripción	O Exclusivo Lógico.
Formato	XOR op1, op2
Código de Operación	16
Número de Operandos	2
Modos de Direccionamiento	op1: inmediato, registro, memoria, indirecto, relativo op2: inmediato, registro, memoria, indirecto, relativo
Comportamiento	Efectúa la operación 'o exclusivo lógico' bit a bit entre el contenido del operando 1 y el operando 2, y almacena el resultado en el registro acumulador. La tabla de verdad de la operación es: op1 XOR op2 | 0 | 1 0 | 0 | 1 1 | 1 | 0 No modifica los biestables de estado.

De un operando.

Instrucción	NOT
Descripción	Negación Lógica.
Formato	NOT op1
Código de Operación	17
Número de Operandos	1
Modos de Direccionamiento	op1: registro, memoria, indirecto, relativo
Comportamiento	Efectúa la operación 'negación lógica' bit a bit en el operando 1. La tabla de verdad de la operación es: op1 | NOT op1 0 | 1 1 | 0 No modifica los biestables de estado.

BIFURCACIONES.

Todas ellas producen un salto a la posición de memoria indicada por op1, dependiendo de la instrucción y el contenido de los biestables de estado.

Instrucción	BR
Descripción	Bifurcación incondicional.
Formato	BR op1
Código de Operación	18
Número de Operandos	1
Modos de Direccionamiento	op1: memoria, relativo a contador de programa, indirecto
Comportamiento	Carga el PC con el valor contenido del operando 1.

Instrucción	BZ
Descripción	Bifuración si resultado igual cero.
Formato	BZ op1
Código de Operación	19
Número de Operandos	1
Modos de Direccionamiento	op1: memoria, relativo a contador de programa, indirecto
Comportamiento	Si el biestable Z está activo (Z=1), carga el PC con el valor contenido del operando 1.

Instrucción	BNZ
Descripción	Bifurcación si resultado distinto de cero.
Formato	BNZ op1
Código de Operación	20
Número de Operandos	1
Modos de Direccionamiento	op1: memoria, relativo a contador de programa, indirecto
Comportamiento	Si el biestable Z está inactivo (Z=0), carga el PC con el valor contenido en el operando 1.

Instrucción	BP
Descripción	Bifurcación si resultado positivo.
Formato	BP op1
Código de Operación	21
Número de Operandos	1
Modos de Direccionamiento	op1: memoria, relativo a contador de programa, indirecto
Comportamiento	Si el biestable S está inactivo (S=0), carga el PC con el valor contenido en el operando 1.

Instrucción	BN
Descripción	Bifurcación si resultado negativo.
Formato	BN op1
Código de Operación	22
Número de Operandos	1
Modos de Direccionamiento	op1: memoria, relativo a contador de programa, indirecto
Comportamiento	Si el biestable S está activo (S=1), carga el PC con el valor contenido en el operando 1.

Instrucción	BV
Descripción	Bifuración si hay desbordamiento.
Formato	BV op1
Código de Operación	23
Número de Operandos	1
Modos de Direccionamiento	op1: memoria, relativo a contador de programa, indirecto
Comportamiento	Si el biestable V está activo (V=1), carga el PC con el valor contenido en el operando 1.

Instrucción	BNV
Descripción	Bifurcación si no hay desbordamiento.
Formato	BNV op1
Código de Operación	24
Número de Operandos	1
Modos de Direccionamiento	op1: memoria, relativo a contador de programa, indirecto
Comportamiento	Si el biestable V está inactivo (V=0), carga el PC con el valor contenido en el operando 1.

Instrucción	BC
Descripción	Bifurcación si hay acarreo.
Formato	BC op1
Código de Operación	25
Número de Operandos	1
Modos de Direccionamiento	op1: memoria, relativo a contador de programa, indirecto
Comportamiento	Si el biestable C está activo (C=1), carga el PC con el valor contenido en el operando 1.

Instrucción	BNC
Descripción	Bifurcación si no hay acarreo.
Formato	BNC op1
Código de Operación	26
Número de Operandos	1
Modos de Direccionamiento	op1: memoria, relativo a contador de programa, indirecto
Comportamiento	Si el biestable C está inactivo (C=0), carga el PC con el valor contenido en el operando 1.

Instrucción	BE
Descripción	Bifurcación si el resultado tiene paridad par.
Formato	BE op1
Código de Operación	27
Número de Operandos	1
Modos de Direccionamiento	op1: memoria, relativo a contador de programa, indirecto
Comportamiento	Si el biestable P está inactivo (P=0), carga el PC con el valor contenido en el operando 1.

Instrucción	BO
Descripción	Bifurcación si el resultado tiene paridad impar.
Formato	BO op1
Código de Operación	28
Número de Operandos	1
Modos de Direccionamiento	op1: memoria, relativo a contador de programa, indirecto
Comportamiento	Si el biestable P está activo (P=1), carga el PC con el valor contenido en el operando 1.

CONTROL DE SUBRUTINAS.

Instrucción	CALL
Descripción	Llamada a subrutina.
Formato	CALL op1
Código de Operación	29
Número de Operandos	1
Modos de Direccionamiento	op1: memoria, relativo a contador de programa, indirecto
Comportamiento	Almacena en la pila el valor del contador de programa y salta a la dirección de destino indicada por el operando 1.

Instrucción	RET
Descripción	Retorno de subrutina.
Formato	RET
Código de Operación	30
Número de Operandos	0
Modos de Direccionamiento	N/A
Comportamiento	Rescata de la pila el contenido del contador de programa.

ENTRADA/SALIDA.

Instrucción	INCHAR
Descripción	Leer un carácter.
Formato	INCHAR op1
Código de Operación	31
Número de Operandos	1
Modos de Direccionamiento	op1: registro, memoria, indirecto, relativo
Comportamiento	Lee un carácter de la consola y lo codifica en *ASCII*, dejando a cero los 8 bits superiores de la palabra de 16 bits. Almacena el carácter leído en el operando 1.

Instrucción	ININT
Descripción	Leer un entero.
Formato	ININT op1
Código de Operación	32
Número de Operandos	1
Modos de Direccionamiento	op1: registro, memoria, indirecto, relativo
Comportamiento	Lee un entero de la consola. Si la entrada no puede ser convertida en un número entero (porque se sale de rango o no es un número), la instrucción supondrá que se ha leído un cero. Los enteros se pueden introducir en formato decimal o hexadecimal (precedidos por '0x').

Instrucción	INSTR
Descripción	Leer una cadena.
Formato	INSTR op1
Código de Operación	33
Número de Operandos	1
Modos de Direccionamiento	op1: memoria, indirecto, relativo
Comportamiento	Lee una cadena de caracteres de la consola y la almacena en posiciones consecutivas de memoria a partir de la dirección indicada por el operando 1, acabada con el carácter '\0'. No se comprueba previamente si hay espacio para almacenar la cadena, por lo que se puede producir una excepción si se sobrepasa el límite superior de la memoria.

Instrucción	WRCHAR
Descripción	Escribir un carácter.
Formato	WRCHAR op1
Código de Operación	34
Número de Operandos	1
Modos de Direccionamiento	op1: inmediato, registro, memoria, indirecto, relativo
Comportamiento	Escribe en la consola el carácter cuyo valor ASCII se corresponde con los 8 bits inferiores del valor del operando 1.

Instrucción	WRINT
Descripción	Escribir un entero.
Formato	WRINT op1
Código de Operación	35
Número de Operandos	1
Modos de Direccionamiento	op1: inmediato, registro, memoria, indirecto, relativo
Comportamiento	Escribe en la consola el valor del operando 1. El formato de escritura (decimal o hexadecimal), dependerá de la configuración de visualización de números enteros. En cualquier caso, el formato decimal siempre se escribe con signo (-32768 a 32767).

Instrucción	WRSTR
Descripción	Escribir una cadena.
Formato	WRSTR op1
Código de Operación	36
Número de Operandos	1
Modos de Direccionamiento	op1: memoria, indirecto, relativo
Comportamiento	Escribe en la consola una cadena de caracteres, que estará almacenada en memoria a partir de la dirección indicada por el operando 1. Se escribirán caracteres hasta que se llegue al carácter '\0'. Por tanto, puede ocurrir que durante este proceso se genere una excepción si no se encuentra el carácter de fin de cadena antes de llegar al límite superior de la memoria.

6. MACROINSTRUCCIONES DEL ENSAMBLADOR

Las macroinstrucciones (también conocidas como pseudoinstrucciones) son órdenes dirigidas al programa ensamblador. Sirven para indicarle cómo se quiere que realice la traducción del programa, pero no forman parte de él. Sin embargo, algunas pseudoinstrucciones sirven para reservar posiciones de memoria, destinadas a los datos.

Algunas de ellas van seguidas de una *expresión*. En este contexto, se entiende que una expresión consta de enteros (en base decimal o hexadecimal) combinados con los operadores clásicos: suma, resta, producto, división, módulo y paréntesis, y con las reglas de precedencia de operadores habituales. En el caso de la división, el resultado se trunca (3/2=1). Por ejemplo:

```
(3+7*(16-1)/(4-1))%27
```
(el resultado es 11)

1. ORG expresión
Ensambla el código a partir de la posición de memoria resultado de evaluar la expresión. Se pueden usar macroinstrucciones `ORG` a lo largo del código fuente todas las veces que se precise. Sin embargo, si no se emplean con orden, se pueden obtener resultados inesperados en el código máquina. Por ejemplo, en este fragmento de código:

```
            ORG 0
bloque_1:   MOVE .R2,.R1
            ...
            ...
            ORG 0
bloque_2:   PUSH #0xFF
            ...
```

Se lee la pseudoinstrucción `ORG 0`, por tanto, la etiqueta `bloque_1` tomará el valor 0 y se ensamblarán las siguientes instrucciones a partir de dicha dirección. Luego se vuelve a leer otra pseudoinstrucción `ORG 0`, y nuevamente, la etiqueta `bloque_2` tomará el valor 0 y se ensamblarán las siguientes instrucciones a partir de dicha dirección, sobrescribiendo a las del primer bloque que se ensambló.

Si no se incluye ninguna pseudoinstrucción `ORG` en el código fuente, se comenzará a ensamblar a partir de la posición de memoria 0.

2. EQU expresión
Equivalencia. Sirve para dar valor numérico a una etiqueta, empleándola de la siguiente forma:

```
etiqueta: EQU expresión
```
(etiqueta=expresión)

Por tanto, cada vez que aparezca la etiqueta como operando en una instrucción, será sustituida por el valor que ha tomado en la pseudoinstrucción `EQU`. Si no se indica la etiqueta, la expresión se calcula, pero su valor se pierde.

3. END
Marca el final del código. Después de la macroinstrucción `END`, el ensamblador da por finalizado el proceso de traducción, ignorando cualquier instrucción posterior.

4. RES expresión
Reserva tantas posiciones de memoria como indica el resultado de evaluar la expresión. Para marcar el inicio de la zona de memoria, se emplea de la siguiente forma:

```
etiqueta: RES expresión
```
(etiqueta toma el valor de la primera posición reservada)

Por tanto, cada vez que aparezca la etiqueta como operando en una instrucción, será sustituida por el valor que ha tomado en la pseudoinstrucción `RES`. Si no se indica ninguna etiqueta, la zona de memoria se reservará igualmente, pero se pierde la referencia a su dirección de comienzo.

5. DATA a, b, c...
Define un conjunto de datos en memoria. Los datos a, b, c... pueden ser enteros (en decimal o hexadecimal) o cadenas de caracteres delimitadas entre comillas dobles. Para marcar el inicio de la zona de datos se emplea de la siguiente forma:

```
etiqueta: DATA a, b, c...
```
(etiqueta toma el valor de la primera posición de la zona de datos)

Por tanto, cada vez que aparezca la etiqueta como operando en una instrucción, será sustituida por el valor que ha tomado en la pseudoinstrucción `DATA`. Si no se indica ninguna etiqueta, los datos se almacenarán en memoria igualmente, pero se pierde la referencia al primero de ellos.

Las cadenas de caracteres definidas en esta pseudoinstrucción pueden contener los caracteres especiales '\0' (carácter nulo), '\t' (tabulador) y '\n' (fin de línea), y cualquiera de los caracteres imprimibles.

Un posible ejemplo de uso de esta pseudoinstrucción sería:

```
ORG 0
DATA "texto\n", 33
END
```

Este listado fuente produce el resultado que se observa en la Figura 6.1.

0000 :	't'
0001 :	'e'
0002 :	'x'
0003 :	't'
0004 :	'o'
0005 :	'\n'
0006 :	'\0'
0007 :	33d

Figura 6.1. Resultado de la pseudoinstrucción DATA

7. DETECCIÓN DE ERRORES EN EL CÓDIGO FUENTE

Tras el proceso de ensamblado, si todo ha ido bien, la herramienta habrá ubicado en memoria el código máquina correspondiente al fuente introducido. Sin embargo, si había

errores en el listado fuente, es el momento en que se comunicarán al usuario. *ENS2001* presenta una lista de errores con este formato:

```
Línea m: ERROR[n] Descripción del error – Token que causó el error
```

Donde `m` es el número de línea donde se encontró el error y `n` es el código del error. A continuación se muestra una somera descripción y, si procede, la entrada que produjo el error. Sólo se muestra un error por línea (aunque existan varios). Obviamente, si hay errores no se genera código máquina (y no se altera el contenido de la memoria que existiera antes del ensamblado).

Estos son los errores detectados por *ENS2001*:

- **Error 01**. *Modo de direccionamiento del Operando 1 erróneo*.
  El modo de direccionamiento del primer operando no se encuentra dentro de los permitidos para la instrucción. Ejemplo:

  ```
  CALL #10[.IX]
  ```

- **Error 02**. *Modo de direccionamiento del Operando 2 erróneo*.
  El modo de direccionamiento del segundo operando no se encuentra dentro de los permitidos para la instrucción. Ejemplo:

  ```
  MOVE .R2,#1000
  ```

- **Error 03**. *Instrucción no reconocida*.
  La instrucción introducida no pertenece al juego de instrucciones de la máquina virtual. Ejemplo:

  ```
  JP $bucle
  ```

- **Error 04**. *Operando 1 incorrecto*.
  El primer operando introducido no es un operando válido, pero sí que es un *token* del lenguaje. Ejemplo:

  ```
  WRINT 33
  ```

- **Error 05**. *Operando 2 incorrecto*.
  El segundo operando introducido no es un operando válido, pero sí que es un *token* del lenguaje. Ejemplo:

  ```
  MOVE .r4,"cadena"
  ```

- **Error 06**. *Etiqueta duplicada*.
  La etiqueta ya se había definido previamente en el código. Ejemplo:

  ```
  datos: DATA 1,2,3,4,5
  ...
  datos: MOVE #0,.r5
  ```

- **Error 07**. *Etiqueta no definida*.
  La etiqueta usada como operando no se ha definido en ningún lugar del código. Ejemplo:

```
CALL /rutina
```

- **Error 08**. *Entrada incorrecta*.
  Se ha encontrado un *token* no válido en el código fuente. Ejemplo:

```
símbolo: EQU 10
```

- **Error 09**. *Expresión errónea*.
  La expresión no está bien formada, contiene operadores u operandos no válidos, o los paréntesis no están balanceados. Ejemplo:

```
datos: RES 10*(2-(256/4)
```

- **Error 10**. *Pseudoinstrucción ORG define el comienzo del código fuera de la memoria*.
  El comienzo del código viene definido por una expresión cuyo valor excede el límite superior de la memoria (65535d). Ejemplo:

```
ORG 65535+1
```

- **Error 11**. *Pseudoinstrucción RES reserva espacio fuera de la memoria*.
  La cantidad de espacio reservado por la pseudoinstrucción viene definida por una expresión cuyo valor excede el límite superior de la memoria a partir de la posición actual de ensamblado. Ejemplo:

```
RES 32768*2
```

- **Error 12**. *Pseudoinstrucción DATA ubica datos fuera de la memoria*.
  La ubicación de la lista de datos ha excedido el límite superior de la memoria. Ejemplo:

```
ORG 65535
DATA "p"
```

- **Error 13**. *Expresión fuera de rango*.
  El valor de la expresión excede la representación de 16 bits. Ejemplo:

```
etiqueta: EQU 32768*2
```

- **Error 14**. *Nombre de etiqueta reservado*.
  Se ha utilizado como etiqueta el mnemónico de una instrucción del procesador. Ejemplo:

```
BNZ /MOVE
```

- **Error 15**. *Entero fuera de rango*.
  Se ha introducido como operando un entero que excede la representación en 16 bits o bien en 8 bits para los modos de direccionamiento relativos a índices y a PC. Ejemplo:

```
WRINT 65536
```

- **Error 16**. *Se esperaba el Operando 1*.
  No se encontró el primer operando. Ejemplo:

```
CALL
```

- **Error 17**. *Se esperaba el Operando 2*.
  No se encontró el segundo operando. Ejemplo:

```
ADD #-10[.IY],
```

- **Error 18**. *Se esperaba carácter de fin de línea*.
  No se ha encontrado el carácter fin de línea donde debería estar. Ejemplo:

```
MUL .R1,.R2,.R3
```

- **Error 19**. *Se esperaba carácter separador*.
  No se ha encontrado el separador (coma) entre los dos operandos de una instrucción. Ejemplo:

```
DIV #0x7000 [.R0]
```

- **Error 20**. *Lista de datos errónea*.
  La lista de datos que acompaña a la pseudoinstrucción *equ* no se compone de enteros o cadenas encerradas entre comillas dobles, separados por comas. Ejemplo:

```
datos: DATA 'cadena',1,2,3
```

- **Error 21**. *Desplazamiento fuera de rango*.
  El desplazamiento viene indicado por el valor de una etiqueta que excede los 8 bits permitidos. Ejemplo:

```
bucle: RES 256
...
BR $bucle
```

- **Error 22**. *Error en asignación de memoria*.
  Ha habido un error de asignación de espacio en memoria para las estructuras internas del ensamblador. El usuario debería salir de la aplicación.

8. INTERFAZ GRÁFICA

En la Figura 8.1 se muestra la ventana principal de la aplicación *ENS2001* en su versión *Windows*. A partir de aquí, se efectuará un repaso exhaustivo por todas las funcionalidades que ofrece.

Para ejecutar la aplicación, se hará a través del archivo `wens2001.exe`.

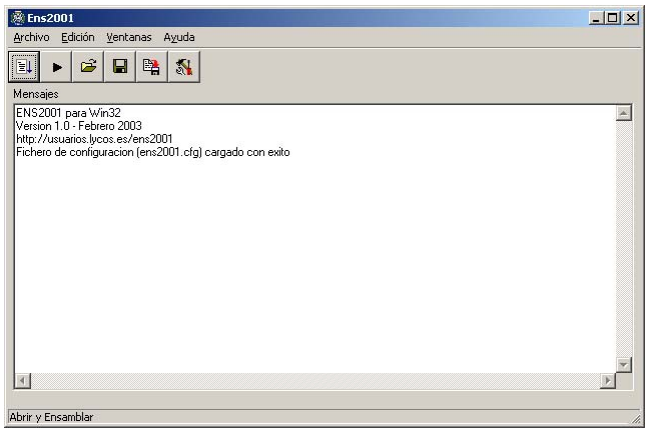

Figura 8.1. Ventana Principal *ENS2001-Win32*

Se observan varias partes bien diferenciadas.

- **Menú de la aplicación**. A través de él se puede acceder a la totalidad de opciones que presenta la herramienta.
- **Barra de botones** con los comandos más comunes.
- **Cuadro de mensajes**. En él aparecerán los mensajes de error o confirmación de operaciones que la herramienta vaya generando a lo largo de la sesión.
- **Barra de estado**. Ofrece una descripción del botón activo, cuando se está a la espera de una orden del usuario, o bien indica qué operación se está procesando.

Además de esta ventana principal, la aplicación dispone de otras cinco ventanas flotantes, accesibles desde el menú o con las combinaciones de acceso rápido (Ctrl+F1 a Ctrl+F5) y una sexta ventana donde se muestran las opciones de configuración.

8.1.MENÚ DE LA APLICACIÓN

El menú principal de la aplicación (Figura 8.1.1) consta de cuatro opciones, que conforman cuatro submenús:

Figura 8.1.1. Menú Principal

- Archivo.
- Edición.
- Ventanas.
- Ayuda.

Submenú Archivo.

El submenú Archivo (Figura 8.1.2) agrupa las opciones relativas al manejo principal de la herramienta. Son las siguientes:

Figura 8.1.2. Menú Archivo

Abrir y Ensamblar (atajo Ctrl+E).

Con este comando, la herramienta abre un cuadro de diálogo para que el usuario seleccione un fichero.

Una vez seleccionado, lee el código fuente contenido en él, lo ensambla y, si el proceso de ensamblado fue correcto, coloca el código en memoria. Si no, muestra en el cuadro de mensajes un listado con los errores producidos.

Al ensamblar un fichero, se crean los archivos temporales `memoria.tmp` (contenido de la memoria tras ensamblar el código fuente) y `errores.tmp` (listado de los errores producidos al ensamblar, si los hubiera), que se eliminarán tras el proceso. No obstante, si se produce algún error en la creación o acceso a estos archivos, no se podrá completar la operación. Posiblemente habrá que salir de la aplicación y solucionar el error (liberando espacio en disco, porque se haya ejecutado *ENS2001* desde una unidad de red con acceso exclusivo de escritura o un *CD-ROM*, etc.).

Ejecutar (atajo Ctrl+J).

Invoca al procesador virtual para lanzar la simulación. El procesador ejecutará instrucciones hasta que se cumpla una de las siguientes condiciones:

- Ejecución Paso a Paso activa (se detiene nada más ejecutar una instrucción).
- Punto de Ruptura Activado (se detiene antes de ejecutar la instrucción).
- Excepción durante la ejecución.
- Instrucción `HALT`

Al finalizar la simulación, la herramienta mostrará en el cuadro de mensajes cuál de estas causas fue la que detuvo al procesador.

Si durante la ejecución el procesador encuentra una instrucción de entrada o salida por consola, se mostrará la ventana de la consola y esperará a la introducción del dato por parte del usuario para continuar. En el caso de que el procesador se encuentre esperando por un dato de entrada, el usuario podrá detener la ejecución pulsando la tecla ESCAPE.

Cargar Imagen de Memoria (atajo Ctrl+L).

Este comando permite recuperar el contenido íntegro de la memoria desde un fichero en el que se habrá guardado previamente. La herramienta mostrará un cuadro de diálogo donde el usuario seleccionará el fichero que desee cargar.

Como la lectura se hace en formato binario, si se intenta leer un fichero cualquiera que no haya sido generado previamente mediante el comando *Guardar Imagen de Memoria*, los resultados son inesperados.

Guardar Imagen de Memoria (atajo Ctrl+S).

Este comando permite guardar el contenido íntegro de la memoria en un fichero. La herramienta mostrará un cuadro de diálogo para que el usuario escoja el nombre del fichero.

Si no existe, se creará un nuevo fichero con una longitud de 128 Kbytes, y formato *big-endian*. En caso de que existiera, la aplicación preguntará si se desea sobrescribir o bien da la opción de introducir otro nombre.

NOTA: *big-endian* es una forma de almacenar los números de 16 bits en palabras de 8 bits, de forma que primero se almacena la palabra más significativa, y después la menos significativa. La forma contraria de hacerlo se denomina *little-endian*.

Guardar Sesión (atajo Ctrl+G).

Este comando, que es el único que sólo está disponible para la versión gráfica de la aplicación, permite guardar en un fichero el contenido del cuadro de mensajes y la consola. Para ello, como es habitual, presentará un cuadro de diálogo donde el usuario puede escoger en qué fichero desea guardar la sesión.

Igual que en la operación anterior, si el fichero ya existe, la herramienta dará a elegir entre sobrescribirlo o introducir un nombre nuevo.

Configuración (atajo Ctrl+F).

Este comando se estudiará en profundidad al hablar de la ventana de configuración. Muestra dicha ventana, en la que se presentan las diferentes opciones de configuración de la herramienta.

Salir (atajo Ctrl+X).

Este comando sirve para salir de la aplicación. Tiene el mismo efecto que pulsar el icono de cerrar (X) en la esquina superior derecha de la ventana principal. Al salir, tras pedir confirmación al usuario, se guardará la configuración actual y la posición y estado de las ventanas flotantes.

Submenú Edición.

Este submenú (Figura 8.1.3), que también es activable haciendo clic con el botón derecho del ratón sobre el cuadro de mensajes, ofrece cuatro sencillas opciones de edición para dicho cuadro:

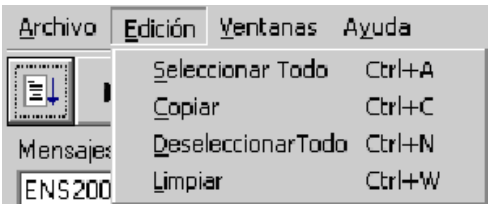

Figura 8.1.3. Menú Edición

Seleccionar Todo (atajo Ctrl+A).

Selecciona todo el texto contenido en el cuadro de mensajes.

Copiar (atajo Ctrl+C).

Copia en el portapapeles el texto del cuadro de mensajes que se encuentre seleccionado en ese momento.

Deseleccionar Todo (atajo Ctrl+N).

Descarta la selección actual del cuadro de mensajes.

Limpiar (atajo Ctrl+W).

Borra el contenido del cuadro de mensajes.

Submenú Ventanas.

En este submenú (Figura 8.1.4) el usuario puede mostrar u ocultar las cinco ventanas flotantes de las que dispone la herramienta. También puede conseguirse el mismo efecto mediante los atajos de teclado. En el menú se muestra un *tick* a la izquierda de aquellas ventanas que estén visibles (en la figura sería la ventana de consola).

Figura 8.1.4. Menú Ventanas

- **Código Fuente** (atajo Ctrl+F1).
- **Consola** (atajo Ctrl+F2).
- **Memoria** (atajo Ctrl+F3).
- **Registros** (atajo Ctrl+F4).
- **Pila** (atajo Ctrl+F5).

Haciendo clic en cada opción se muestra la ventana si estaba oculta, o se oculta si era visible. También es posible ocultar las ventanas haciendo clic en su icono de cerrar respectivo, en la esquina superior derecha de todas ellas.

Más adelante se describe con detenimiento cada una de las ventanas. Tanto la posición como el tamaño y el estado de las ventanas se guarda en el fichero `wens2001.cfg` cada vez que se finalice el uso de la herramienta, y se intentará recuperar cuando se inicie la siguiente sesión. Si no se puede recuperar el fichero, se creará de nuevo con los valores predeterminados.

Submenú Ayuda.

Figura 8.1.5. Menú Ayuda

Como se puede ver en la Figura 8.1.5, sólo contiene una opción, **Acerca de** (atajo Alt+D), que muestra información acerca de la versión de *ENS2001* que se está ejecutando, la fecha de compilación, y la *URL* del proyecto (Figura 8.1.6).

Figura 8.1.6. Cuadro de Diálogo Acerca de

8.2. Barra de Botones

La barra de botones sirve de acceso rápido a las acciones más frecuentes de la herramienta. Su comportamiento es análogo a seleccionar la misma acción desde el menú de la aplicación. Son los siguientes (de izquierda a derecha):

Abrir y Ensamblar

Ejecutar

Cargar Imagen de Memoria

Guardar Imagen de Memoria

Guardar Sesión

Configuración

El botón Ejecutar cambiará de icono y de función, dependiendo de las siguientes condiciones:

- Cuando haya un programa ejecutándose, su función será **Detener** la ejecución, y se mostrará de esta forma:

- Cuando en la configuración esté seleccionado el Modo de Ejecución **Paso a Paso**, tendrá esta forma:

8.3. CUADRO DE MENSAJES

En esta ventana (Figura 8.3.1) se irán mostrando los distintos mensajes generados por la herramienta, como consecuencia del proceso de ensamblado o ejecución. Ejemplos de estos mensajes pueden ser el número de líneas procesadas por el ensamblador, los errores producidos (caso de que existan), o la condición de detención del simulador tras la ejecución.

Figura 8.3.1. Cuadro de Mensajes

Además, como se comentó anteriormente, haciendo clic con el botón derecho del ratón, se puede acceder al menú de edición, pero esta vez de forma flotante.

8.4. VENTANA DE CÓDIGO FUENTE

Se trata de una ventana en la que el usuario tiene acceso al contenido de la memoria, pero una vez desensamblado, con lo que se puede leer fácilmente el código contenido en ella, como se aprecia en la Figura 8.4.1. El contenido de la ventana se actualiza automáticamente tras ensamblar con éxito un programa fuente y tras concluir la ejecución de un programa contenido en memoria. En este último caso, además, la ventana mostrará el código ensamblado a partir de la posición a la que apunte el contador de programa en ese instante.

El tamaño de la ventana es variable, por lo que el número de instrucciones mostradas dependerá del mismo.

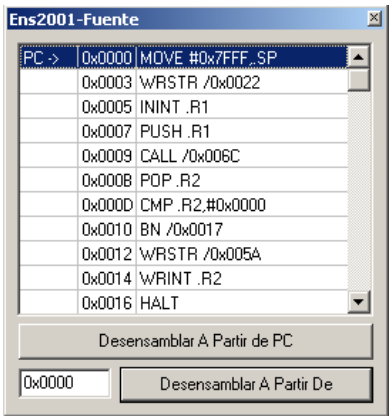

Figura 8.4.1. Ventana de Código Fuente

Hay que reseñar que, debido a que todas las instrucciones no ocupan el mismo tamaño en memoria (las hay de 1, 2 y 3 palabras de longitud), se pueden obtener resultados erróneos al desensamblar a partir de una dirección arbitraria. También puede leerse código extraño al desensamblar las zonas de datos de los programas, por ejemplo. Para evitarlo, se debería desensamblar siempre a partir del comienzo del código o bien a partir de la posición ocupada por el contador de programa o una instrucción desensamblada anteriormente.

Si se desplaza la ventana hacia abajo una posición, se saltará una instrucción, mientras que si se desplaza hacia arriba, se saltará una posición de memoria. Esto es así porque, partiendo de una posición cualquiera, no es posible saber en qué dirección comienza la instrucción anterior. Por ejemplo, si en la ventana de la Figura 8.4.1, el usuario presiona la flecha hacia abajo, la primera posición de memoria pasará a ser la 3, pero si a continuación presiona la flecha hacia arriba, no volverá a la 0, sino que la primera será la 2.

Para desensamblar código a partir de una dirección arbitraria, se puede introducir dicha dirección de comienzo en el cuadro de texto situado al lado del botón **Desensamblar A Partir De**, y a continuación pulsar dicho botón. Para desensamblar a partir de la dirección a la que apunta el contador de programa (PC), se pulsará el botón **Desensamblar A Partir De PC**.

En esta ventana también se pueden activar y desactivar los puntos de ruptura incondicionales, que servirán de ayuda al usuario a la hora de depurar sus programas. Para ello, basta con hacer doble clic en la posición de memoria que se quiera editar, y se irá alternando entre punto de ruptura activado y desactivado. Los puntos de ruptura activos se marcan con un asterisco "*" en la columna de la izquierda, como se observa en la Figura 8.4.2, en la que hay un punto de ruptura activo en la dirección 0009h.

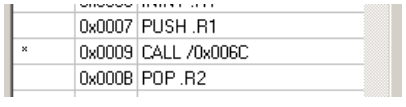

Figura 8.4.2. Marca de Punto de Ruptura Activado

Como información adicional, se puede consultar la posición a la que apunta el contador de programa. Se trata del indicador "PC →" que aparecerá en la columna de la izquierda. En la Figura 8.4.3, al desensamblar esta zona de la memoria, el contador de programa apuntaba a la dirección 0007h.

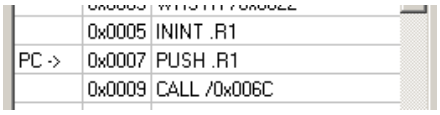

Figura 8.4.3. Indicador de contador de programa

8.5. VENTANA DE CONSOLA

A través de esta ventana (Figura 8.5.1) se efectuarán las operaciones de entrada/salida durante la simulación. En ella se visualizará el resultado de las instrucciones de salida (`WRCHAR`, `WRINT` y `WRSTR`) y en ella el usuario deberá introducir los datos de entrada para las instrucciones correspondientes (`INCHAR`, `ININT` e `INSTR`) cuando se ejecute alguna de ellas.

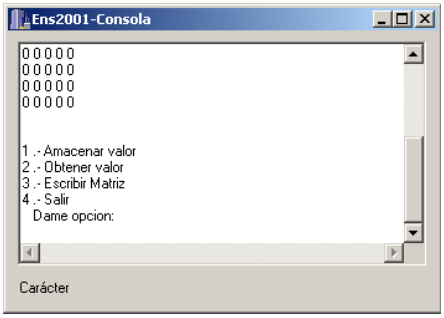

Figura 8.5.1. Consola

Aunque esté oculta, en el momento en que se ejecute cualquiera de las seis instrucciones citadas, la ventana de consola pasará a un primer plano. Además, si se trata de una instrucción de entrada, la ejecución se detendrá hasta que el usuario introduzca el dato solicitado, tras lo que continuará la ejecución, o bien pulse la tecla ESCAPE, con lo que se generará la excepción *Ejecución Detenida por el Usuario*.

Cuando se esté ejecutando una instrucción de entrada de datos, en la parte inferior izquierda de la ventana aparecerá una indicación sobre el tipo de datos que se está leyendo (carácter, entero o cadena). En la Figura 8.5.1 se está ejecutando una instrucción `INCHAR`.

La consola consta de un menú de edición análogo al del cuadro de mensajes, accesible haciendo clic con el botón derecho del ratón encima de la misma, tal y como se observa en la Figura 8.5.2. Ofrece las siguientes opciones:

Seleccionar Todo (atajo Ctrl+A).

Selecciona todo el texto contenido en la consola.

Copiar (atajo Ctrl+C).

Copia en el portapapeles el texto de la consola que se encuentre seleccionado en ese momento.

Deseleccionar Todo (atajo Ctrl+N).

Descarta la selección actual de la consola.

Limpiar (atajo Ctrl+W).

Borra el contenido de la consola.

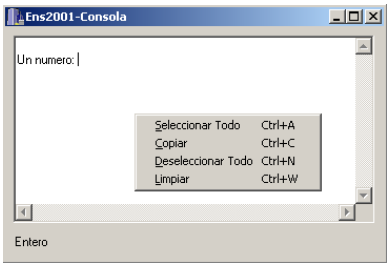

Figura 8.5.2. Menú Edición de la consola

8.6. VENTANA DE MEMORIA

En esta ventana (Figura 8.6.1) es posible consultar el valor de tantas celdas de memoria como quepan en ella, dependiendo de su tamaño. También es posible editar el contenido de las celdas, haciendo doble clic en cualquiera de ellas.

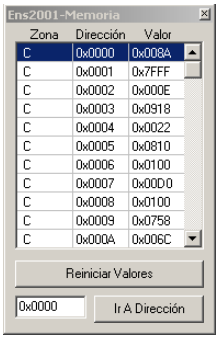

Figura 8.6.1. Ventana de Memoria de *ENS2001*

Además, aparecerán marcadas con una 'C' en la columna de la izquierda aquellas posiciones de memoria que pertenezcan a la zona de código, mientras que aparecerán marcadas con una 'P' aquellas posiciones que ocupe la pila, tal y como se aprecia en la Figura 8.6.2, en la que las posiciones 017Ch a 017Eh están ocupadas por código y las posiciones 017Fh a 0182h están ocupadas por la pila del sistema.

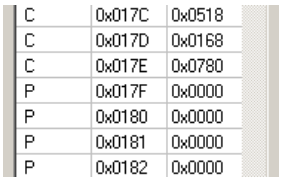

Figura 8.6.2. Indicadores de Zona Código y Pila

Se dispone de dos botones (Figura 8.6.1). El primero de ellos, **Reiniciar Valores**, sirve para poner a cero el contenido de todas las posiciones de memoria. El segundo, **Ir A Dirección**, permite visualizar en la ventana la memoria a partir de la dirección introducida en el cuadro de texto anexo.

8.7. VENTANA DE REGISTROS

En esta ventana se obtiene una visión global del banco de registros de la máquina virtual, con los valores que toma cada uno de ellos y la posibilidad de modificarlos haciendo doble clic encima de aquél cuyo valor se desee editar. Puede observarse su aspecto en la Figura 8.7.1.

También muestra información sobre los biestables de estado (éstos no se pueden editar directamente, pero se modificarán al editar el registro `SR`), la siguiente instrucción que se va a ejecutar (a la que apunta `PC`), y el rango de direcciones ocupado por la pila del sistema y el código ensamblado.

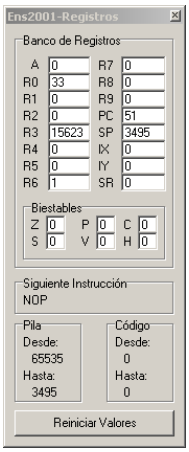

Figura 8.7.1. Ventana de Banco de Registros

Por último, también se dispone de un botón para **Reiniciar Valores**, que inicializará el contenido del banco de registros. Al pulsarlo, se pedirá confirmación, y si el usuario pulsa el botón **Aceptar**, todos los registros tomarán el valor 0, excepto el puntero de pila, que tiene un tratamiento especial: La zona de código puede dejar un hueco anterior y otro posterior a ella. Se calcula cuál es el mayor hueco. Entonces, si la pila crece en sentido ascendente, el puntero de pila ocupará la primera posición del hueco mayor, mientras que si crece en sentido descendente, ocupará la última posición de dicho hueco.

8.8. VENTANA DE PILA

En esta ventana (Figura 8.8.1) se visualizará la zona de memoria correspondiente a la pila. El manejo de la misma, en cuanto a la edición de posiciones de memoria, es idéntico al de la ventana de memoria, por lo que no se abundará más en el tema. En este caso no aparece información acerca de las zonas de código y pila, sólo un indicador de la posición de memoria a la que apunta el puntero de pila (`SP`).

Igual que en la ventana de memoria, la cantidad de posiciones de memoria visibles simultáneamente dependerá del tamaño de la ventana. Adicionalmente, posee dos botones. El primero, **Ir A Dirección**, que centra la visualización en la dirección de memoria introducida en el cuadro de texto a la izquierda del botón, y el segundo, **Ir A Puntero de Pila**, que muestra la Pila hasta la posición de memoria a la que apunta `SP`. Se trata del indicador "SP →" que aparecerá en la columna de la izquierda. Si la pila crece ascendentemente, `SP` aparecerá en la parte inferior de la ventana. Si crece descendentemente, `SP` aparecerá en la parte superior. De esta forma, es posible visualizar las posiciones cercanas a la cima.

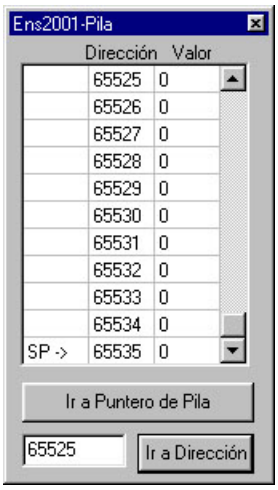

Figura 8.8.1. Ventana de Pila

8.9. VENTANA DE CONFIGURACIÓN

En esta ventana se encuentran accesibles todas las opciones de configuración, como se muestra en la Figura 8.9.1. Se trata de siete grupos de opciones, cada una de las cuales puede tomar dos valores:

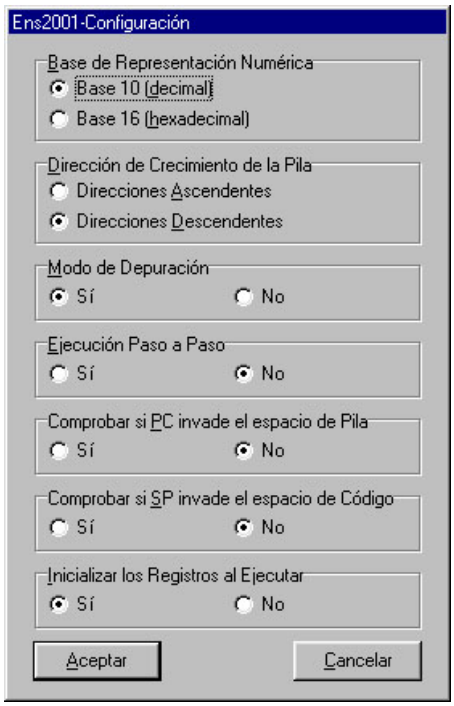

Figura 8.9.1. Ventana de Configuración

Base de Representación Numérica:

Se trata de un selector que permite indicarle al simulador la base de representación numérica que empleará al visualizar los números enteros, bien decimal, bien hexadecimal.

- Base 10 (decimal).
- Base 16 (hexadecimal).

Dirección de Crecimiento de la Pila:

Se trata de un selector que permite indicarle al simulador cuál será el comportamiento de la pila. Hay dos modos predefinidos:

- Direcciones crecientes. La pila crece ascendentemente. Las instrucciones `PUSH` y `POP` se comportan de la siguiente manera:

`PUSH`	`.sp ++` `M(.sp) ← op1`
`POP`	`op1 ← M(.SP)` `.sp --`

- Direcciones decrecientes. La pila crece descendentemente. Las instrucciones `PUSH` y `POP` se comportan de la siguiente manera:

`PUSH`	`M(.sp) ← op1` `.sp --`
`POP`	`.sp ++` `op1 ← M(.SP)`

Modo de Depuración:

Se trata de un selector que permite indicarle al simulador que se detenga cuando encuentre un punto de ruptura, o bien que los ignore por completo.

- Sí.
- No.

Ejecución Paso a Paso:

Se trata de un selector que permite indicarle al simulador que ejecute las instrucciones una a una, deteniéndose después de cada una de ellas, o bien que comience a ejecutar hasta que encuentre un punto de ruptura (si están activos), una instrucción `HALT` en el código, o bien se produzca una excepción.

- Sí.
- No.

Comprobar si PC invade el espacio de Pila:

Se trata de un selector que permite indicarle al simulador que lance una excepción si el contador de programa se sitúa entre los límites superior e inferior de la pila, o bien que ignore dicha situación.

- Sí.
- No.

Comprobar si SP invade el espacio de Código:

Se trata de un selector que permite indicarle al simulador que lance una excepción si el puntero de pila se sitúa entre los límites superior e inferior de la zona perteneciente al código, o bien que ignore dicha situación.

- Sí.
- No.

Inicializar los registros al ejecutar:

Se trata de un selector que permite indicarle al simulador si debe reiniciar el banco de registros al comenzar la ejecución de un programa. No obstante, sólo se reiniciará tras finalizar una ejecución anterior, esto es, cuando el flag `H` esté activo (en otras palabras, si la última instrucción que se ejecutó fue `HALT`).

- Sí.
- No.

Pulsando el botón **Aceptar** se validarán las opciones marcadas, mientras que al pulsar el botón **Cancelar**, los cambios que se hayan introducido no tendrán efecto.

Al salir de la herramienta se guardarán las opciones de configuración en el fichero `ens2001.cfg`, para que sean restauradas en el inicio de una nueva sesión.

9. INTERFAZ CONSOLA

Salvo la posibilidad de guardar la sesión en un fichero, el resto de funcionalidades son las mismas que para la versión gráfica. Sin embargo, por motivos obvios el manejo es totalmente distinto.

La invocación de la herramienta desde la consola se hace introduciendo en la línea de comandos:

```
ens2001 [nombre_fichero]
```

Donde `nombre_fichero` es el fichero fuente que se ensamblará y colocará en memoria nada más arrancar. Este parámetro es opcional. Si no se indica, se entrará en la herramienta sin más.

En el proceso de arranque, la herramienta intentará cargar la configuración de la sesión anterior, almacenada en el fichero `ens2001.cfg`. En este caso, se mostrará el mensaje:

```
Fichero de configuración (ens2001.cfg) cargado con éxito
```

Si el fichero no existe, o es inválido, se creará uno nuevo y válido con las opciones por defecto, y aparecerá este otro mensaje:

```
Fichero de configuración (ens2001.cfg) no encontrado o erróneo
Se usarán las opciones por defecto
```

Tras esto, se muestra la configuración actual, cualquiera que sea. La configuración por defecto que aplicará la herramienta, si no se encontró el fichero de configuración, es la siguiente:

- Representación en base decimal.
- La pila crece en sentido descendente.
- La ejecución se detiene al encontrar un punto de ruptura.

- La ejecución se efectúa hasta que se dé una condición de parada (no paso a paso).
- No se comprobará si PC invade el espacio de Pila.
- No se comprobará si SP invade el espacio de Código.
- Se inicializará el Banco de Registros al comenzar la ejecución (después de la finalización de una ejecución anterior).

Si se indicó que se ensamblara directamente un fichero (mediante el parámetro opcional de la línea de comandos), a continuación aparecerá el resultado de dicha operación (ver comando *'c' Abrir y Ensamblar*).

Para finalizar el proceso de arranque, se muestra el indicador de la herramienta:

```
ENS2001>
```

Esto quiere decir que está preparada para recibir órdenes.

Todos los comandos de *ENS2001* constan de una letra, seguida en su caso de ninguno, uno o dos argumentos. Los comandos se aceptan tanto en mayúsculas como en minúsculas. A continuación se enumeran todos los comandos disponibles, agrupados por funcionalidades.

'h' (Ayuda).

Muestra una lista de los comandos permitidos, igual a la que se muestra a continuación.

```
Referencia Rapida de Comandos Modo Consola
--------------------------------------------------------------------------
a               Activa/Desactiva Modo Paso A Paso
b               Activa/Desactiva Puntos de Ruptura
b dir           Pone/Quita un Punto de Ruptura en la dirección dir
c fichero       Carga y Ensambla el fichero
d dir num       Desensambla num instrucciones a partir de dir
e               Ejecuta el código
g               Activa/Desactiva comprobación si PC invade la zona de pila
h               Visualiza la ayuda
i               Activa/Desactiva comprobación si SP invade la zona de código
k               Modifica el Funcionamiento de la pila (Creciente/Decreciente)
l fichero       Carga una Imagen de Memoria desde fichero
m dir [valor]   Visualiza/Modifica el contenido de la Dirección dir
n base          Base de Representación Numerica (10 o 16)
o               Configuración del Simulador
p num           Vuelca num posiciones de la pila
q               Salir del Programa
r               Visualiza el Banco de Registros
r reg [valor]   Visualiza/Modifica el Contenido del Registro reg
s fichero       Guarda una Imagen de Memoria desde fichero
t               Resetea Registros/No Resetea Registros al ejecutar programa
v dir num       Vuelca num posiciones de memoria a partir de la dirección dir
--------------------------------------------------------------------------
```

Configuración del simulador.

'a' (Activar/Desactivar modo Paso a Paso).

Se trata de un selector que permite indicarle al simulador que ejecute las instrucciones una a una, deteniéndose después de cada una de ellas, o bien que comience a ejecutar hasta que encuentre un punto de ruptura (si están activos), una instrucción `HALT` en el código, o bien se produzca una excepción.

'b' (Activar/Desactivar Puntos de Ruptura).

Se trata de un selector que permite indicarle al simulador que se detenga cuando encuentre un punto de ruptura, o bien que los ignore por completo.

'g' (Activar/Desactivar comprobación si PC invade la zona de Pila).

Se trata de un selector que permite indicarle al simulador que lance una excepción si el contador de programa se sitúa entre los límites superior e inferior de la pila, o bien que ignore dicha situación.

'i' (Activar/Desactivar comprobación si SP invade la zona de Código).

Se trata de un selector que permite indicarle al simulador que lance una excepción si el puntero de pila se sitúa entre los límites superior e inferior de la zona perteneciente al código, o bien que ignore dicha situación.

Los límites del código los define el ensamblador, lo que quiere decir que si se modifica el código por otros medios (carga de imagen de memoria, edición manual, etc.), dichos límites no se corresponderán con la realidad y los resultados pueden ser inesperados.

'k' (Seleccionar Modo de Funcionamiento de la Pila).

Se trata de un selector que permite indicarle al simulador cuál será el comportamiento de la pila. Hay dos modos predefinidos:

- Direcciones crecientes. La pila crece ascendentemente. Las instrucciones `PUSH` y `POP` se comportan de la siguiente manera:

PUSH	.sp ++ M(.sp) ← op1
POP	op1 ← M(.SP) .sp --

- Direcciones decrecientes. La pila crece descendentemente. Las instrucciones `PUSH` y `POP` se comportan de la siguiente manera:

PUSH	M(.sp) ← op1 .sp--
POP	.sp ++ op1 ← M(.SP)

'n' base (Seleccionar la Base de Representación Numérica).

El parámetro base indica la base de representación con la que la herramienta presentará todos los números enteros, tanto de direcciones, como de datos, salida por consola, etc. Sólo se permiten los valores 10 (base decimal) y 16 (base hexadecimal).

't' (Activar/Desactivar Reiniciar Registros al comenzar la ejecución).

Se trata de un selector que permite indicarle al simulador si debe reiniciar el banco de registros al comenzar la ejecución de un programa. No obstante, sólo se reiniciará tras

finalizar una ejecución anterior, esto es, cuando el biestable `H` esté activo (en otras palabras, si la última instrucción que se ejecutó fue `HALT`).

'o' (Mostrar la Configuración Actual).

Muestra una lista con las opciones de configuración seleccionadas actualmente.

A continuación se enumeran los comandos que permiten consultar y modificar individualmente los valores contenidos tanto en la memoria como en los registros del simulador.

'm' (Acceso a memoria).

Este comando tiene tres posibles usos. Si se emplea la sintaxis:

```
m dir
```

presenta en pantalla el contenido de la posición de memoria `dir`. En cambio, si se emplea este otro formato:

```
m dir valor
```

modifica el contenido de la posición de memoria dir, actualizándolo con el entero indicado por `valor`. Se puede producir un error si `dir` o `valor` no son valores válidos para una dirección de memoria o un entero, respectivamente. Por último, introduciendo el comando:

```
m reset
```

se reinician a cero todas las posiciones de memoria.

'r' (Acceso al Banco de Registros).

Este comando también tiene varios usos, en concreto cuatro, según la sintaxis empleada. Si se introduce únicamente:

```
r
```

muestra en pantalla el contenido de todos los registros del banco de registros, así como los biestables de estado y los límites definidos para las zonas de código y pila. Si se quiere consultar únicamente el contenido de un registro en concreto, se usará la sintaxis:

```
r reg
```

donde `reg` es el nombre del registro (sin el punto delante) cuyo valor se va a consultar. Si se indica un nombre que no se corresponde con ninguno de los registros de la máquina, devolverá un error.

Para modificar el contenido de un registro se empleará el siguiente formato:

```
r reg valor
```

con lo que se actualizará el contenido del registro `reg` con el entero indicado por `valor`. Tanto si `reg` no se corresponde con ninguno de los registros de la máquina, como si `valor` no entra dentro del rango permitido de los enteros, devolverá un error.

Por último, existe la posibilidad de reiniciar el contenido de todos los registros, introduciendo el comando:

```
r reset
```

Todos los registros tomarán el valor 0, excepto el puntero de pila, que tiene un tratamiento especial. La zona de código puede dejar un hueco anterior y otro posterior a ella. Se calcula cuál es el mayor hueco. Entonces, si la pila crece en sentido ascendente, el puntero de pila ocupará la primera posición del hueco mayor, mientras que si crece en sentido descendente, ocupará la última posición de dicho hueco.

La herramienta pone a disposición del usuario algunos comandos para visualizar la información contenida en bloques de memoria, incluso para desensamblar código.

'v' dir num (Acceso a Bloques de Memoria).

Este comando permite visualizar un bloque de memoria a partir de la dirección `dir` y de tantas posiciones de memoria como las indicadas por el parámetro `num`. Dado el caso, informará si hay algún error en los parámetros.

'p' num (Acceso a la Pila).

El comportamiento de este comando es análogo al anterior, sólo que como inicio del bloque tomará la dirección a la que apunte el puntero de pila, y centrará la visualización hacia posiciones ascendentes o descendentes, según esté configurado el funcionamiento de la pila, de manera que permita visualizar el contenido de las posiciones cercanas a la cima.

'd' dir num (Desensamblar).

Este comando permite desensamblar `num` instrucciones a partir de la dirección de memoria `dir`. Hay que reseñar que, debido a que todas las instrucciones no ocupan el mismo tamaño en memoria (las hay de 1, 2 y 3 palabras de longitud), se pueden obtener resultados erróneos al desensamblar a partir de una dirección arbitraria. También puede leerse código extraño al desensamblar las zonas de datos de los programas, por ejemplo. Para evitarlo, siempre se debería desensamblar a partir del comienzo del código o bien a partir de la posición ocupada por el contador de programa o una instrucción desensamblada anteriormente.

A continuación se muestran los comandos que permiten cargar y grabar información:

'c' fichero (Cargar y Ensamblar).

Con este comando, la herramienta lee el código fuente contenido en `fichero`, lo ensambla y, si el proceso de ensamblado fue correcto, coloca el código en memoria. Si no, muestra un listado con los errores producidos.

Al ensamblar un fichero, se crean los archivos temporales `memoria.tmp` (contenido de la memoria tras ensamblar el código fuente) y `errores.tmp` (listado de los errores producidos al ensamblar, si los hubiera), que se eliminarán una vez finalizado el proceso. Si no pudiera crear alguno de ellos, se mostrará el siguiente error y no se completará la operación:

```
Error generando fichero nombre_fichero
El directorio de trabajo está lleno o protegido contra escritura
```

Esto podría ocurrir, por ejemplo, si se intenta ejecutar la aplicación desde un *CD-ROM* o una unidad de red con acceso exclusivo de lectura.

's' (Guardar una Imagen de Memoria).

Este comando permite guardar el contenido íntegro de la memoria en un fichero. Se creará un nuevo fichero con una longitud de 128 Kbytes, en formato *big-endian*. Si el fichero ya existía, se sobrescribirá.

'l' (Cargar una Imagen de Memoria).

Este comando permite recuperar el contenido íntegro de la memoria desde un fichero en el que se habrá guardado previamente mediante el comando *'s' Guardar una Imagen de Memoria*. Como la lectura se hace en formato binario, si se intenta leer un fichero cualquiera que no haya sido generado por la aplicación los resultados son inesperados.

Y, por último, pero no ello menos importantes, se muestran los comandos que lanzan la simulación y dan control sobre las partes que se van a ejecutar.

'b' dir (Activar/Desactivar un Punto de Ruptura).

Activa un punto de ruptura en la dirección `dir`, o bien, si ya estaba activo, lo desactiva.

'e' (Ejecutar).

Invoca al procesador virtual para lanzar la simulación. El procesador ejecutará instrucciones hasta que se cumpla una de las siguientes condiciones:

- Ejecución Paso a Paso activa (se detiene nada más ejecutar una instrucción).
- Punto de Ruptura Activado (se detiene antes de ejecutar la instrucción).
- Excepción durante la ejecución.
- Instrucción `HALT`

Al finalizar la simulación, la herramienta mostrará en pantalla cuál de estas causas fue la que detuvo al procesador.

Si durante la ejecución el procesador encuentra una instrucción de entrada por consola, esperará a la introducción del dato por parte del usuario para continuar. Si encuentra una instrucción de salida, el resultado aparecerá en la consola, intercalado con la salida de la propia interfaz.

Para detener la ejecución en cualquier momento, incluso cuando se están ejecutando instrucciones de entrada/salida, bastará con que el usuario pulse la tecla ESCAPE. Se generará una excepción que detendrá al procesador. Esta última característica no está disponible en la versión consola para *Linux*.

10. TABLAS RESUMEN

TABLA 10.1. RESUMEN DE FORMATOS DE INSTRUCCIÓN

Instrucción sin operandos:

15 6	5 3	2 0
Código de operación	000	000

Instrucción con un operando inmediato o directo a memoria:

15 6	5 3	2 0
Código de operación	Modo Dir op1	000

15 0
Operando 1

Instrucción con un operando ni inmediato ni directo a memoria:

15 6	5 3	2 0
Código de operación	Modo Dir op1	000

15 8	7 0
Operando 1	00000000

Instrucción con dos operandos, ambos o bien inmediatos o bien directos a memoria:

15 6	5 3	2 0
Código de operación	Modo Dir op1	Modo Dir op2

15 0	15 0
Operando 1	Operando 2

Instrucción con dos operandos, el primero inmediato o directo a memoria y el segundo no.

15 6	5 3	2 0
Código de operación	Modo Dir op1	Modo Dir op2

15 0
Operando 1

15 8	7 0
00000000	Operando 2

Instrucción con dos operandos, el segundo inmediato o directo a memoria y el primero no.

15 6	5 3	2 0
Código de operación	Modo Dir op1	Modo Dir op2

15 8	7 0
Operando 1	00000000

15 0
Operando 2

Instrucción con dos operandos, ninguno de los cuales es inmediato ni directo a memoria.

15 6	5 3	2 0
Código de operación	Modo Dir op1	Modo Dir op2

15 8	7 0
Operando 1	Operando 2

Nota: los operandos que usen sólo 4 bits rellenarán los 4 bits superiores con ceros.

TABLA 10.2. MODOS DE DIRECCIONAMIENTO, ANCHO Y CODIFICACIÓN

Modo	Ancho	Codificación
Sin Operando	N/A	000 (0)
Inmediato	16 bits	001 (1)
Registro	4 bits	010 (2)
Memoria	16 bits	011 (3)
Indirecto	4 bits	100 (4)
Relativo a IX	8 bits	101 (5)
Relativo a IY	8 bits	110 (6)
Relativo a PC	8 bits	111 (7)

TABLA 10.3. BANCO DE REGISTROS Y CODIFICACIÓN

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PC	SP	IY	IX	SR	A	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0

TABLA 10.4. POSICIÓN DE LOS BIESTABLES DE ESTADO DENTRO DEL REGISTRO DE ESTADO

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	H	S	P	V	C	Z

TABLA 10.5. INSTRUCCIONES Y DIRECCIONAMIENTOS PERMITIDOS

Mnemónico	Cód. Oper.	Modo Dir. Op1							Modo Dir. Op2						
		#	.	/	[ ]	#[.ix]	#[.iy]	$	#	.	/	[ ]	#[.ix]	#[.iy]	$
NOP	0														
HALT	1														
MOVE	2	✓	✓	✓	✓	✓	✓			✓	✓	✓	✓	✓	
PUSH	3	✓	✓	✓	✓	✓	✓								
POP	4		✓	✓	✓	✓	✓								
ADD	5	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	
SUB	6	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	
MUL	7	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	
DIV	8	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	
MOD	9	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	
INC	10		✓	✓	✓	✓	✓								
DEC	11		✓	✓	✓	✓	✓								
NEG	12		✓	✓	✓	✓	✓								
CMP	13	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	
AND	14	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	
OR	15	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	
XOR	16	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	
NOT	17		✓	✓	✓	✓	✓								
BR	18			✓	✓			✓							
BZ	19			✓	✓			✓							
BNZ	20			✓	✓			✓							
BP	21			✓	✓			✓							
BN	22			✓	✓			✓							
BV	23			✓	✓			✓							
BNV	24			✓	✓			✓							
BC	25			✓	✓			✓							
BNC	26			✓	✓			✓							
BE	27			✓	✓			✓							
BO	28			✓	✓			✓							
CALL	29			✓	✓			✓							
RET	30														
INCHAR	31		✓	✓	✓	✓	✓								
ININT	32		✓	✓	✓	✓	✓								
INSTR	33			✓	✓	✓	✓								
WRCHAR	34	✓	✓	✓	✓	✓	✓								
WRINT	35	✓	✓	✓	✓	✓	✓								
WRSTR	36			✓	✓	✓	✓								

11. BIBLIOGRAFÍA

[De Miguel 1996] Miguel, P.: *Fundamentos de los Computadores*. Quinta Edición Revisada. Ed. Paraninfo, 1996.

12. LISTADO DE ILUSTRACIONES