

# **MANUAL SIMULADOR MÁQUINA RUDIMENTARIA (SIMR 3.0)**

LABORATORI DE FONAMENTS DE COMPUTADORS  
2N SEMESTRE ETIS

# 1. Máquina Rudimentaria

## 1.1 INTRODUCCIÓN

---

La Máquina Rudimentaria (MR) es un procesador pedagógico. Su principal objetivo es servir como herramienta para enseñar los conceptos básicos sobre estructura y arquitectura de computadores. El diseño lógico de la MR se ha realizado utilizando conocimientos básicos sobre sistemas digitales, por lo cual puede utilizarse como ejemplo de un sistema complejo en un curso de lógica digital, permitiendo realizar una transición natural entre el estudio de los circuitos digitales y el del lenguaje máquina. La MR ha sido diseñada en el Departament d' Arquitectura de Computadores (DAC) de la Universitat Politècnica de Catalunya (Barcelona, España).

### **Contenidos:**

- lenguaje máquina
- lenguaje ensamblador
- implementación
- ejemplo de programa

## 1.2 LENGUAJE MÁQUINA DE LA MR

---

### **INTRODUCCIÓN**

La MR es un procesador RISC de propósito general. Tiene un banco de registros de 8 registros de 16 bits, numerados desde R0 a R7. El R0 es un registro especial que no se puede escribir y que siempre contiene el valor 0. Dispone de tres indicadores de condición: V (overflow), Z (cero) y N (negativo) que las instrucciones de salto usan para decidir si el salto se produce o no.

La memoria está compuesta por 256 palabras de 16 bits, y es direccionable a nivel de palabra. Los operandos de la MR son números enteros codificados en complemento a dos con 16 bits.

Las instrucciones de la MR son de longitud fija de 16 bits. El código de operación se encuentra en los dos bits de mayor peso. Existen tres tipos de instrucciones:

- Instrucciones aritmético-lógicas.
- Instrucciones de acceso a memoria.
- Instrucciones de salto.

Las instrucciones usan cuatro modos de direccionamiento:

- Modo registro.
- Modo inmediato.
- Modo relativo (o base + desplazamiento).
- Modo absoluto.

El repertorio de instrucciones de la MR se describe a continuación.

- Instrucciones aritmético-lógicas
- Instrucciones de acceso a memoria
- Instrucciones de salto

## **INSTRUCCIONES ARITMÉTICO-LÓGICAS**

Las instrucciones aritmético-lógicas usan el modo registro y el modo inmediato.

- En el modo registro se accede a un registro del banco de registros.
- El operando inmediato (en aquellas instrucciones que lo utilizan) es un número de 5 bits codificado en complemento a dos y contenido dentro de la propia instrucción.

Todas las instrucciones aritmético-lógicas escriben su resultado en un registro (Rd) y activan convenientemente los indicadores de condición V, N y Z. Si el registro indicado es el R0 no se escribe el resultado, y solamente se activan los indicadores V, N y Z.

Se distinguen dos tipos de instrucciones aritmético-lógicas, en función de donde están sus operandos fuente: las que leen los operandos de registros y las que tienen un operando en un registro y el otro operando es inmediato. La MR dispone de seis instrucciones aritmético-lógicas:

- ADD-Suma con registros: suma (el contenido de) dos registros (Rf1 + Rf2).
- SUB-Resta con registros: resta dos registros (Rf1 - Rf2).
- ASR-Desplazamiento aritmético: desplaza 1 bit a la derecha con extensión de signo un registro (Rf >> 1). Esta instrucción tiene un solo operando fuente.
- AND- and lógica bit a bit de dos registros (Rf1 and Rf2).
- ADDI-Suma con inmediato: suma un registro y el operando inmediato (n) extendido a 16 bits (Rf + n).
- SUBI-Resta con inmediato: resta el operando inmediato (n) extendido a 16 bits de un registro (Rf - n).

Todas las instrucciones aritmético-lógicas usan el mismo código de operación (11). A continuación se codifican en binario los registros destino, fuente 1 y fuente 2. En las instrucciones registro registro no se usan los dos bits que vienen a continuación. En las instrucciones con un operando inmediato se codifica este operando usando los tres bits de Rf2 y los dos bits no usados. En el caso particular de la instrucción ASR, de sólo un operando, éste se codifica en el campo Rf2. Los tres bits más bajos de la instrucción constituyen el campo denominado OP, y codifican el tipo de operación a realizar con los datos. El campo OP (bits 2-0) permite identificar a cada instrucción según la siguiente tabla:

- ADD: 100
- SUB: 101
- ASR: 110
- AND: 111
- ADDI: 000
- SUBI: 001

Como puede observarse en la tabla anterior, el bit OP2 distingue los dos tipos de instrucciones aritmético-lógicas: registro-registro (OP2=1) y registro-inmediato (OP2=0).

## **INSTRUCCIONES DE SALTO**

Las instrucciones de salto usan el modo absoluto para indicar la dirección de salto. Se requieren 8 bits para identificar una dirección absoluta. Ninguna instrucción de salto

altera el valor de los indicadores de condición. Existen seis instrucciones de salto condicional y una de salto incondicional:

- BG-Saltar si mayor: se produce el salto si  $N=0$  y  $Z=0$ .
- BGE-Saltar si mayor o igual: se produce el salto si  $N=0$ .
- BL-Saltar si menor: se produce el salto si  $N!=V$ .
- BLE-Saltar si menor o igual: se produce el salto si  $N!=V$  ó  $Z=1$ .
- BEQ-Saltar si igual: se produce el salto si  $Z=1$ .
- BNE-Saltar si distinto: se produce el salto si  $Z=0$ .
- BR-Saltar incondicionalmente: se produce el salto siempre.

Todas las instrucciones de salto usan el código de operación (10) codificado en los bits de mayor peso. El campo COND (bits 13-11), codificado a continuación, permite identificar a cada instrucción según la siguiente tabla:

- BR: 000
- BEQ: 001
- BL: 010
- BLE: 011
- BNE: 101
- BGE: 110
- BG: 111

A continuación del campo COND se codifican tres bits a 0. Finalmente, los 8 bits más bajos de la instrucción contienen la dirección de salto.

### ***INSTRUCCIONES DE ACCESO A MEMORIA***

Las instrucciones de acceso a memoria usan el modo registro y el modo desplazamiento. La dirección de memoria a la que acceden se calcula sumando los 8 bits de menor peso de un registro del banco de registros, denominado registro índice ( $R_i$ ), con una dirección codificada en la propia instrucción (dirección base).

Existen dos instrucciones de acceso a memoria

- Load: almacena en un registro ( $R_d$ ) el contenido de una posición de memoria y activa los indicadores V, N y Z. Si el registro destino es  $R_0$ , no se escribe. Su código de operación es 00.
- Store: almacena en una posición de memoria el contenido de un registro ( $R_f$ ). Esta instrucción no altera el valor de los indicadores de condición. Su código de operación es 01.

El campo CO (código de operación) ocupa los dos bits de mayor peso. A continuación se codifica el registro destino (LOAD) o el registro fuente (STORE). Después se codifican los tres bits del registro índice y, finalmente, en los 8 bits de menor peso se codifica la dirección base.

### 1.3 ARQUITECTURA DE LA MÁQUINA RUDIMENTARIA: NIVEL DE LENGUAJE ENSAMBLADOR

---

El lenguaje ensamblador (LE) se introduce para simplificar la programación en lenguaje máquina. El LE de la MR está constituido por instrucciones, etiquetas, directivas, expresiones y macros.

#### **INSTRUCCIONES**

El LE codifica, mediante la utilización de un código mnemotécnico, cada una de las distintas instrucciones del lenguaje máquina y la forma de acceder a los operandos.

Instrucciones aritmético-lógicas

- ADD Rf1, Rf2, Rd
- SUB Rf1, Rf2, Rd
- AND Rf1, Rf2, Rd
- ASR Rf, Rd
- ADDI Rf1, #num, Rd
- SUBI Rf1, #num, Rd

Instrucciones de acceso a memoria

- LOAD dir\_base(Ri), Rd
- STORE Rf, dir:base(Ri)

Instrucciones de salto

- BEQ dir\_destino
- BNE dir\_destino
- BG dir\_destino
- BGE dir\_destino
- BL dir\_destino
- BLE dir\_destino
- BR dir\_destino

Descripción de los mnemotécnicos y de la especificación de los operandos.

- Los registros fuente (Rf, Rf1 y Rf2) contienen los operandos de la instrucción.
- El registro destino (Rd) almacena el resultado de la operación.
- La constante #num es un operando inmediato codificado en complemento a dos con cinco bits.
- La dirección dir\_base es la dirección base de memoria para realizar el acceso a un dato.
- El registro índice (Ri) contiene el desplazamiento a sumar a la dirección base para obtener la dirección efectiva del operando.
- La dirección de memoria dir\_destino es la dirección donde se debe saltar en caso de cumplirse la condición de un salto.

## ETIQUETAS

Las etiquetas son un mecanismo para establecer referencias a instrucciones y datos sin necesidad de conocer su dirección exacta en memoria. Una etiqueta se define al principio de una línea mediante un identificador seguido por el símbolo ":". Los símbolos definidos como etiquetas pueden ser utilizados para especificar direcciones en instrucciones de acceso a memoria o de salto. El proceso de ensamblaje encargado de la traducción a lenguaje máquina calcula la dirección de memoria exacta correspondiente a cada etiqueta, y la almacena en una tabla de símbolos que es utilizada para la traducción de cada instrucción.

## DIRECTIVAS

Las directivas son indicaciones que se introducen en un programa para controlar el proceso de ensamblaje. Suelen empezar con un punto. Existen dos tipos básicos de directivas en el LE de la MR:

**Directivas de definición de variables y constantes:** se utilizan para reservar posiciones de memoria o inicializarlas con valores concretos. También permiten la asignación de valores a símbolos.

**.dw  $n_1$ ,  $n_2$ , ...,  $n_N$ :** inicializa posiciones de memoria consecutivas con los valores indicados ( $n_1$ ,  $n_2$ , etc).

**.rw  $n$**  reserva  $n$  palabras consecutivas de memoria sin asignarles valor inicial (la implementación del ensamblador incluido en este entorno de simulación asigna un 0 a estas posiciones).

**Identificador =  $n$**  establece una equivalencia entre un identificador y un valor numérico. La declaración de esta directiva sólo afecta a la tabla de símbolos, y no ocupa por tanto memoria del programa.

**Directivas de control de ejecución:** Permiten conocer la dirección de la primera y la última instrucción a ejecutar.

**.begin identificador:** indica la dirección de la primera instrucción a ejecutar en un programa.

**.end:** indica la finalización de un programa.

Cualquier programa en lenguaje ensamblador debe tener una directiva **.begin** y al menos una directiva **.end**.

## EXPRESIONES ARITMÉTICAS

El lenguaje ensamblador permite el uso de expresiones aritméticas en lugar de valores numéricos. Las expresiones aritméticas son evaluadas durante el proceso de ensamblaje. Una expresión es:

- un valor numérico,
- una etiqueta,
- la suma (+), la resta (-), el producto (\*) o la división (/) de dos expresiones.

Para realizar la evaluación de las expresiones se utiliza el orden de precedencia clásico de los operadores. No se permiten paréntesis.

## MACROS

Una *macro* o *pseudoinstrucción* es un módulo de instrucciones parametrizable que puede ser referenciado desde cualquier punto del programa. Para definir una macro se dispone de dos directivas especiales, entre las cuales debe escribirse el código (cuerpo) de la macro:

- `.def nombre {lista de argumentos} cuerpo de la macro`
- `.enddef`

La directiva `.def` señala el principio de la definición de la macro y le da nombre. La directiva `.enddef` indica el final de la macro. Los argumentos pueden ser de tres tipos, y van precedidos de los símbolos `$d`, `$i` o `$`, según el espacio de direcciones en el que está almacenado el dato al que hacen referencia:

- `$dn`: el operando `n` debe ser interpretado como una dirección de memoria,
- `$in`: el operando `n` debe ser interpretado como una constante inmediata, y
- `$n`: el operando `n` está en el registro `Rn` de la UP.

## 1.4 IMPLEMENTACIÓN DE LA MR

---

La Máquina Rudimentaria (MR) posee una arquitectura de tipo Von Neumann. Por tanto, la MR ejecuta programas guardados en la memoria que también contiene los datos requeridos por los programas. La forma normal de introducir datos y programas en la memoria en un computador von Neumann es haciendo uso de la unidad de Entrada/Salida (E/S). Dado que la MR no dispone de esta unidad, se asumirá que los datos y las instrucciones se encuentran en una dirección de memoria determinada, sin preocuparse de como han sido almacenados en ella.

Describiremos a continuación las características generales de cada una de las unidades de la MR, según el modelo de arquitectura von Neumann:

- la memoria,
- la Unidad Central de Proceso o procesador y
- los Buses de interconexión.

### **LA MEMORIA**

La memoria de la MR está organizada en 256 posiciones de 16 bits cada una. Esta memoria recibe los datos a través del bus de entrada Min y los envía al procesador a través del bus Mout. Ambos buses son de 16 bits. Para acceder a una posición de memoria es preciso poner su dirección en el bus de direcciones M@. Este bus es de 8 bits. La señal L/E indica a la memoria si debe escribir en (1) o leer de (0) la dirección presente en el bus de direcciones.

La MR trabaja con datos numéricos enteros de 16 bits, codificados en complemento a 2. Las instrucciones de la MR se codifican en 16 bits, de modo que cada posición de memoria puede contener indistintamente una instrucción o un dato.

### **LA UNIDAD CENTRAL DE PROCESO**

La Unidad Central de Proceso (CPU) de la MR esta constituida por dos elementos:

- Unidad de Proceso: es la encargada de realizar operaciones sobre los datos almacenados en la memoria.
- Unidad de Control: es la encargada de gobernar el correcto funcionamiento de los circuitos lógicos que componen la Unidad de Proceso.

### **UNIDA DE PROCESO (UP)**

La UP es capaz de ejecutar las instrucciones descritas en el nivel de lenguaje máquina. Ha sido diseñada con el doble objetivo de ser eficiente y de fácil comprensión.

La UP tiene los siguientes elementos, gestionados por la UC:

- Un *Registro de Instrucciones* (IR) de 16 bits, encargado de almacenar la instrucción de lenguaje máquina que se esta ejecutando en cada momento.
- Un *Contador de Programa* (PC) de 8 bits, encargado de almacenar la dirección en memoria de la siguiente instrucción a ejecutar (implementando así el secuenciamiento implícito).
- Un *Banco de Registros* de 8 registros de 16 bits numerados de R0 a R7, encargado de almacenar datos. Todos los registros son visibles para el programador de lenguaje máquina. El registro R0 es un registro especial, que contiene siempre la constante 0 (que no puede ser moficada).



Una *Unidad Aritmético-Lógica* para realizar las operaciones numéricas requeridas por las instrucciones. Además, esta unidad se encarga de evaluar si el resultado de la última operación realizada ha sido cero, negativo o se ha producido overflow. El resultado de esta evaluación se guarda en 3 registros de un bit, denominados *flags de condición* N (negativo), Z (cero) y V (overflow).

En la UP se distinguen dos circuitos sencillos que realizan funciones específicas y tres grandes bloques:

Evaluación de la condición: es un circuito combinacional que recibe como entrada el valor actual de los indicadores de condición V, N y Z y los tres bits que codifican el campo COND de una instrucción de salto. Envía un bit a la UC que indica si el salto debe o no producirse.

Extensión de signo (EXT): realiza la extensión de signo a 16 bits del operando inmediato de 5 bits.

Bloque de gestión del banco de registros: El banco de registros contiene 8 registros de 16 bits (R0 siempre vale cero). Tiene un puerto de salida y un puerto de entrada, y permite lectura y escritura simultánea. La señal de escritura (ERd) está gobernada por la UC. La codificación del registro a escribir se lee directamente de los bits 13-11 de la instrucción (registro destino de las instrucciones aritmético-lógicas y *Load*). El registro a leer se selecciona entre los campos que codifican el registro fuente en las instrucciones aritmético-lógicas (bits 10-8 y 7-5), el registro índice en las instrucciones de acceso a memoria (bits 10-8) o el registro fuente en las instrucciones *Store* (bits 13-11). La selección se realiza mediante el multiplexor SELREG, cuyos bits de control (CRf) son gestionados por la UC.

Bloque de cálculo aritmético-lógico: Este bloque incluye la Unidad Aritmético-Lógica (UAL), en la que se realizan las operaciones requeridas por las instrucciones y se calcula el valor de los indicadores de condición V, N y Z. La UAL también puede dejar pasar el operando B para activar los indicadores de condición durante la ejecución de las instrucciones *Load*. La señal OPERAR de la UAL es gestionada por la UC, e indica si debe realizarse una operación (OPERAR=1) o bien dejar pasar el operando B (OPERAR=0). La UAL ha sido diseñada de tal modo que los bits 1-0 que codifican el campo OP en las instrucciones aritmético-lógicas coincidan con la codificación de la operación a realizar, por lo que estos bits del IR pueden conectarse directamente a los dos bits de control de menor peso de la UAL. El operando A de la UAL siempre proviene del banco de registros. El multiplexor SELDAT permite seleccionar el operando B entre el banco de registros y el operando inmediato para las instrucciones aritméticas, o de la memoria para las instrucciones *Load*.

Bloque de cálculo de direcciones: En este bloque se realiza el cálculo de todas las direcciones de memoria a las que se accede durante la ejecución de un programa. Las direcciones se almacenan en el PC o en el R@, y la UC selecciona el registro adecuado mediante el multiplexor SELDIR. Los accesos a memoria pueden realizarse por tres causas distintas:

Secuenciamiento implícito: La dirección de la siguiente instrucción a ejecutar se lee del PC. Incluye el autoincremento del PC.

Ejecución de una instrucción de acceso a memoria: La dirección de acceso está en el registro R@, y se calcula sumando la dirección base (bits 7-0 de la instrucción) con los 8 bits de menor peso del registro índice. Cuando se realiza el cálculo de la dirección, la UC selecciona la entrada 1 del multiplexor SELREG para leer el registro índice.

Ejecución de una instrucción de salto: La dirección de acceso está en el registro R@. Para almacenar en R@ los bits 7-0 de la instrucción (dirección de salto) se suman éstos con 0. Para ello, la UC selecciona en el multiplexor SELREG la entrada 1. Esta entrada selecciona los bits 10-8 de la instrucción, que en las instrucciones de salto valen "000", y por tanto se selecciona el registro R0 del banco de registros, poniendo un 0 en la entrada del sumador.

### **UNIDAD DE CONTROL (UC)**

El funcionamiento de la UP está gobernado por la UC. La UC es un sistema secuencial que determina el orden en que deben actuar los distintos elementos en la UP para completar la ejecución de cada una de las instrucciones. La UC gestiona la carga de los distintos registros (Ld\_RA, Ld\_IR, Ld\_PC, Ld\_R@, Ld\_RV, Ld\_RZ y Ld\_RN) y del banco de registros (ERd), la memoria (L/E) la UAL (OPERAR) y los distintos multiplexores (PC'/@ y CRf). El control de estos bloques se realiza según el código de operación de la instrucción en ejecución (CO) y la evaluación de la condición de salto (Cond).

La UC se comunica con la Unidad de Proceso mediante un conjunto de señales de control. Su función principal es controlar el secuenciamiento de las operaciones realizadas en la Unidad de Proceso para que las instrucciones del programa se ejecuten correctamente. Este circuito recibe como entradas 3 bits de la Unidad de Proceso y envía 12 bits de salida para controlar los módulos que la forman (registros, ALU, multiplexores, etc.).

La UC se especifica como una máquina de estados finita usando el modelo de *Moore*. En este modelo, las salidas están asociadas a los estados y las transiciones entre estados dependen de las entradas del sistema. Las operaciones a realizar en cada estado se especifican en una tabla de salidas.

### **LOS BUSES**

Los buses de la MR establecen la comunicación entre la CPU y memoria. En la MR, los buses tienen las siguientes características.

- El bus de datos está dividido en dos:
  - Un bus de 16 bits para transportar los datos de la CPU a la memoria (Min).
  - Un bus de 16 bits para transportar los datos de la memoria (Mout) a la CPU.
- El bus de direcciones es de 8 bits y va sólo de la CPU a la memoria (la MR no dispone de unidad de E/S).
- El bus de control tiene únicamente la señal que indica a la memoria si la operación que se realiza es de lectura o de escritura. En los ciclos de escritura no está definido el contenido de la salida Mout.

## 1.5 ¿CÓMO ES UN PROGRAMA ESCRITO EN LENGUAJE ENSAMBLADOR DE LA MR?

---

Cada línea del programa debe contener una sola instrucción

Todos los programas deben tener una línea con la directiva `'.BEGIN '`; la etiqueta corresponde a la dirección de la primera instrucción que comenzará a ejecutar el simulador.

Todo programa debe finalizar con una directiva `'.END'` (puede haber más de una de estas directivas). Esta directiva es en realidad una instrucción `HALT` camuflada, que le indica al simulador que el programa ha terminado. Como es una instrucción, ocupa memoria, y por eso debe ser la última instrucción del programa. Si no fuese así, es preciso considerar que ocupa 16 bits en la memoria de cara a tener en cuenta posibles traducciones a lenguaje máquina del programa y a conocer la ubicación física de cada instrucción.

Las etiquetas se definen como , pudiendo estar la instrucción a la que referencian en la misma línea o en una nueva (a continuación, con tantos espacios en blanco o saltos de línea como sean necesarios). Una etiqueta no puede comenzar por el carácter `'_'`.

Una etiqueta en el código se puede usar para referenciar la posición de memoria en la que está una instrucción o un dato (la escrita inmediatamente a continuación).

Se pueden definir constantes que no ocupan memoria, pero que sí aparecen en la tabla de símbolos: por ejemplo, `B=13`.

Se pueden poner comentarios. Un comentario es cualquier cosa que va después de un `';` y llega hasta el final de la línea.

Existen las directivas `'.RW'` y `'.DW'`, que permiten reservar espacio de memoria para variables o reservar espacio asignando valores respectivamente. En la presente versión del compilador se soporta que los números incluidos en estas directivas puedan ser definidos, usando la notación explicada entre paréntesis para cada ejemplo, como decimales (242), binarios (1010b) o hexadecimales (0xA001).

Existe la directiva `'.ORG dir'`. Esta directiva hace que la instrucción que se encuentra inmediatamente a continuación se traduzca como si se almacenase en la dirección de memoria *dir*. El compilador traducirá las siguientes instrucciones a continuación de ésta. Esta directiva es útil si quiere dividirse la memoria en varias zonas (por ejemplo, una zona de datos y una zona de instrucciones).

Una macro es un grupo de instrucciones asociado con un nombre y con algunos posibles parámetros. Su homónimo en un lenguaje de alto nivel sería una función declarada como *inline*. Por tanto, para ejecutar una macro no se inserta una instrucción de llamada (como se haría con una subrutina), si no que se ejecuta el código de la macro allí donde se requiere, substituyendo adecuadamente los parámetros.

Las macros se pueden definir en un fichero aparte, que se selecciona en el menú *Compilador -> Fichero de Macros por defecto*, y que por defecto es el fichero `"macros.mr"`. El fichero de macros contiene una especificación de cada una de las macros. No obstante, las macros pueden ser también definidas al comienzo del fichero `'.mr'` que contiene el programa.

Es conveniente tener todos los ficheros que intervendrán en la compilación en el mismo directorio.

Para especificar una macro en el fichero, el formato es el siguiente:

.DEF

(lista instrucciones de la macro, usando si es necesario los parámetros)

.ENDDEF

Para llamar a una macro desde un programa sólo hay que escribir su nombre y los parámetros que pueda necesitar.

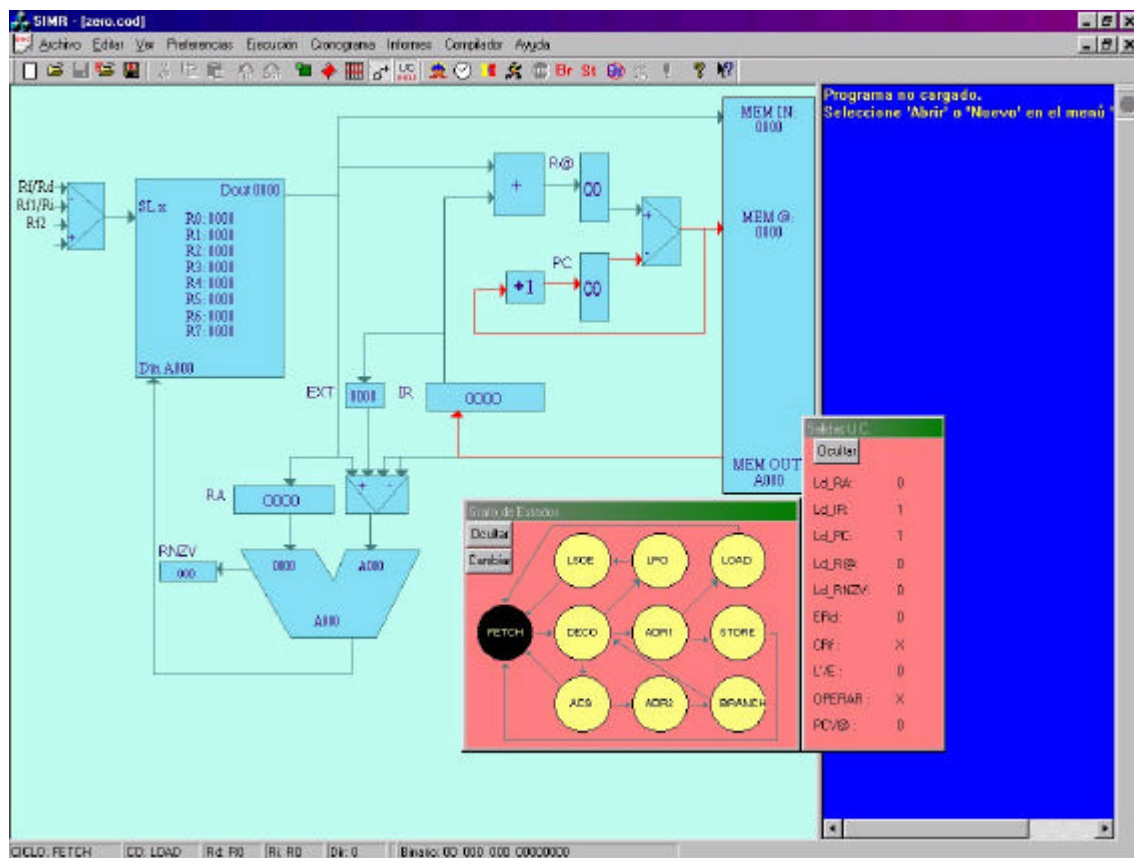
## 2. SiMR

### 2.1 DESCRIPCIÓN GENERAL DEL ENTORNO DE SIMULACIÓN

El entorno de simulación de la Máquina Rudimentaria, SiMR ha sido diseñado para correr sobre plataformas Windows 95 y posteriores. Ofrece al usuario un sencillo editor de programas, un simulador y un compilador de la MR, todo integrado en la misma herramienta.

Para comenzar a trabajar con el entorno es recomendable identificarse. Los datos suministrados en esta identificación figurarán en todos los informes generados. Se puede comenzar a trabajar en el entorno sin identificarse si no se pretende elaborar ningún informe del trabajo realizado.

La figura siguiente presenta la interfaz principal del entorno de trabajo:



En ella se pueden ver dos ventanas azules diferentes:

- la de la izquierda representa el *datapath* (camino de datos) de la máquina rudimentaria (MR),
- la de la derecha contiene el código del programa que se encuentra cargado en la memoria.

Ambas ventanas son escalables: con el ratón se puede desplazar la línea divisoria entre ambas, ajustando así el tamaño necesario para cada usuario.

Finalmente, el cuadro de diálogo rojo-anaranjado que hay flotando en el centro muestra el grafo de estados de la MR, en el cual está resaltado (en negro) el estado

actual de la Unidad de Control. A la derecha se muestran las salidas de la UC en el estado actual.

El programa se ha diseñado pensando en una resolución entre 1024x768 o una 1280x1024, pero su escalabilidad le permiten trabajar en altas resoluciones o en resoluciones menores. En este caso no se podrán ver todos los datos simultáneamente, pero se dispone de todas las funcionalidades de SiMR.


Cada nuevo fichero que se abra se hará en una instancia diferente de SiMR. Como el programa guarda cuantiosa información sobre el estado de la máquina, cronogramas, y datos para deshacer acciones, no se aconseja tener mas de 5 instancias abiertas simultáneamente. Este dato puede variar en función de las prestaciones del equipo (y de la versión de windows) donde se ejecute.


## 2.2 BARRA DE BOTONES


---

La barra de botones permite controlar las acciones más comunes que suelen utilizarse en SIMR. Comenzaremos la descripción de la funcionalidad de cada botón empezando por los botones de más a la izquierda.




Los tres primeros botones son viejos conocidos de cualquier usuario de Windows. Las opciones “abrir archivo”, “nuevo archivo” o “guardar archivo” son comunes a todas las aplicaciones Windows actuales y funcionan siguiendo la misma filosofía.



El botón “abrir archivo”  permite seleccionar un programa de entre los almacenados en el directorio en curso. Se permite abrir archivos de código (.COD) y archivos de compilación (.MR, .ASM). Ver Archivos MR (2.9). El directorio actual se puede cambiar con la interfaz común de Windows.

El botón “nuevo archivo”  permite abrir un archivo vacío para poner código. Su uso más común es el de empezar a escribir un programa.


El botón “guardar archivo”  permite guardar el archivo actual. El programa ya sugiere automáticamente una extensión; en el caso del compilador es .MR, pero si se selecciona la extensión .ASM se omite el paso del precompilador en el fichero (el precompilador traduce las macros, generando un fichero .ASM en el que sólo hay instrucciones del repertorio de la MR).


SIMR trata los botones anteriores como es usual, con algunas excepciones. No esperamos que nadie vaya a programar o retocar un archivo de código máquina, así que los botones “nuevo” y “guardar” hacen referencia únicamente a los archivos del compilador, mientras que el botón “abrir” permite abrir ficheros de todos tipos (.COD, .ASM, .MR). La diferencia entre ASM y MR es la inclusión del precompilador en el proceso de generar un ejecutable, pues en el fichero .ASM no hay llamadas a macros mientras que en un fichero .MR sí puede haberlas.


Los botones que vienen a continuación, “copiar” , “cortar”  y “pegar”  sólo son válidos cuando se trabaja en el editor de texto del compilador. Este editor sirve para la creación y modificación de programas escritos en lenguaje ensamblador. Estos programas están almacenados en ficheros con la extensión '.MR' o bien con la extensión '.ASM'.


El siguiente grupo de botones hace referencia únicamente al simulador. Los dos primeros permiten deshacer (undo) las últimas acciones realizadas por el simulador: el primero de los botones  permite retroceder un ciclo de reloj en el estado del simulador, mientras que el segundo  permite deshacer la ejecución de una instrucción completa. Estos botones pueden ser usados una sola vez antes de continuar avanzando en la ejecución del programa, y no estarán activos hasta que se

haya completado un paso (ciclo o estado de la Unidad de Control) o la ejecución de una instrucción en el simulador. Por tanto, no es posible hacer más de un undo consecutivo.


El siguiente botón  permite mostrar todas las señales de la Unidad de Control sobre la Unidad de Proceso. Con un clic las muestra o las oculta. Este tema se explica en profundidad en la [Ventana UC](#)


El botón  permite ver el contenido de la memoria con detalle, además de modificarla y presentar gráficamente el tipo de accesos. Ver Memoria MR (2.5).



El botón  tiene por función mostrar la ventana del cronograma, que se define a través del [menú de cronograma](#)

botón  permite hacer un *reset* del simulador de la MR. Después de pulsarlo, el PC apunta a la dirección de inicio del programa (la indicada en la directiva '.BEGIN'), la memoria queda tal y como estaba especificada en el fichero '.COD' que contenía el programa y todo queda listo para una nueva ejecución desde las condiciones iniciales. Este paso es irreversible (no acepta *undo*), y por tanto se debe estar seguro de hacerlo antes de usarlo. Por prudencia, el programa pedirá una confirmación antes de realizarlo.



Los botones descritos a continuación indican al simulador la manera en la que debe ejecutar las instrucciones del programa que está cargado en la memoria.



El botón  da la orden de avanzar un ciclo de reloj (*ciclo*) la ejecución. Este avance corresponde a la realización de un estado de la Unidad de Control que esté activa. Al pulsarlo se actualizan las ventanas de código y datapath, así como el grafo de estados y cualquier otro cuadro de diálogo que pueda estar usándose, como por ejemplo el de modificar registros o memoria. Esta opción es útil cuando se quiere seguir en detalle la ejecución de una determinada instrucción del programa, ya que permite seguir paso a paso los cambios que se van produciendo en el datapath a medida que se avanza a través de los estados de la Unidad de Control.

El botón  da la orden al simulador de acabar la instrucción en curso, o lo que es lo mismo, de ejecutar los ciclos que sean necesarios hasta que la unidad de control se encuentre en estado de FETCH. Esta opción (paso o *step*) permite seguir con menos detalle que la anterior el funcionamiento de la MR, y es útil cuando se quiere seguir instrucción a instrucción la ejecución de un programa.

Los siguientes botones,  , son complementarios. El primer botón (*run*) permite ejecutar un programa sin interrupción desde la instrucción en curso hasta el final (a menos que durante la ejecución se encuentre un [breakpoint](#) o un [stop](#); el segundo botón (*stop*) permite detener en cualquier momento esta ejecución.

En el menú Opciones à Preferencias se puede elegir el tiempo de ciclo (número de milisegundos transcurridos entre dos flancos de reloj consecutivos) de cada ciclo en el modo *Run*. Esta opción permite adaptar el simulador tanto a máquinas rápidas como a lentas, y a usuarios pausados o estresados. El número de milisegundos es una mera indicación relativa de tiempo, y en ningún caso puede considerarse como el tiempo real de ejecución en el simulador, aunque sí se verá reflejado en los cronogramas que puedan realizarse de la ejecución del programa.

Los dos siguientes botones   hacen referencia al compilador: el primero de ellos (compilar) da la orden de [compilar](#) el programa que se encuentre en este momento en la ventana de código, mientras que el segundo (*compilar y ejecutar*) da la orden de ejecutar en el simulador el programa actual, compilándolo si no ha sido compilado previamente. La ejecución no tendrá lugar cuando la compilación haya sido errónea.

Los dos últimos botones   permiten el acceso a la ayuda. El primero de ellos muestra la ventana "Acerca de", y el segundo muestra la ayuda contextual.

## 2.3 MENUS

---

### **MENU ARCHIVO**

Este menú nos proporciona la posibilidad de tratar con los fichero que soporta SIMR. Dispone de las siguientes opciones:

*Nuevo.* Tecla de Acceso directo: *Ctrl + N*. Abre un nuevo archivo de compilación (un ejecutable no se puede editar). La extensión del nuevo archivo no esta definida, cuando sea necesario salvarlo se definirá. Se puede salvar con extensión '.mr' o '.asm'.

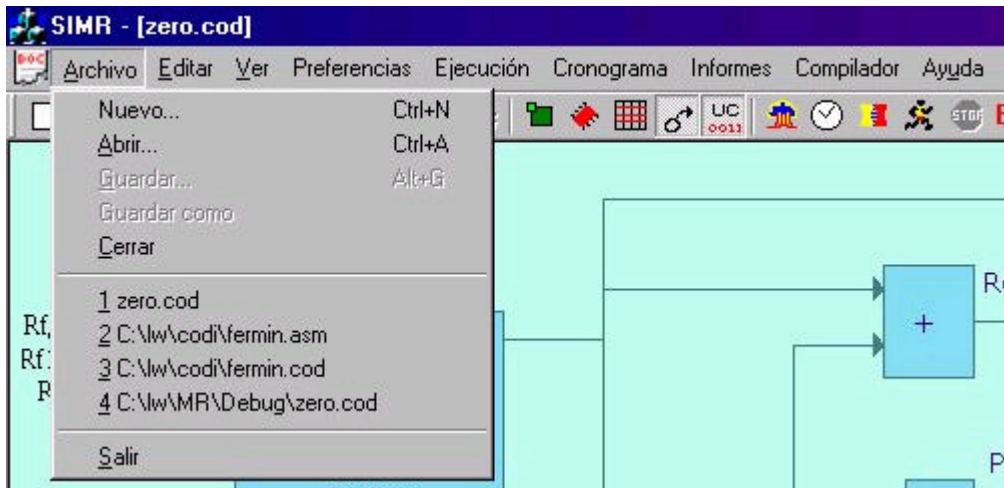
*Abrir.* Tecla de Acceso directo: *Ctrl + A*. Abre cualquier tipo de archivo soportado por SIMR, en especial las extensiones .COD, .MR y .ASM. Los otros tipos de archivo de SIMR deben ser abiertos mas tarde, pues su información no permite arrancar un programa completo. (p.e .STT). Se permite abrir archivos de cualquier extensión, pero si su formato no se reconoce el resultado no está definido. Ficheros con otras extensiones deben ser abiertos con las opciones apropiadas.

*Guardar.* Tecla de Acceso directo: *Alt + G*. Guarda el programa actual en caso que este sea de compilación (los ejecutables no se pueden modificar).

*Guardar como.* Permite guardar cambiando el nombre del fichero.

*Cerrar.* Cierra el archivo actual.

*Salir.* Cierra SIMR.



La lista seguida de nombres que se ve en la imagen corresponde a la lista de ficheros recientes. Aquí se ven los 4 últimos ficheros abiertos; con solo elegirlos se volverán a abrir.

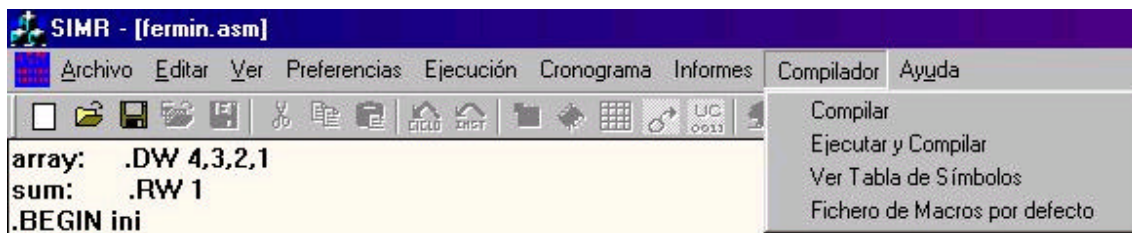
### **COMANDOS DEL MENÚ AYUDA**

El menú Ayuda contiene los siguientes comandos, que le brindan ayuda sobre esta aplicación:



Temas de ayuda	Proporciona un índice con los temas sobre los cuales se puede obtener ayuda.
Acerca de	Muestra el número de versión de la aplicación.

## MENU COMPILADOR



En este menu se accede a las ordenes para el compilador:

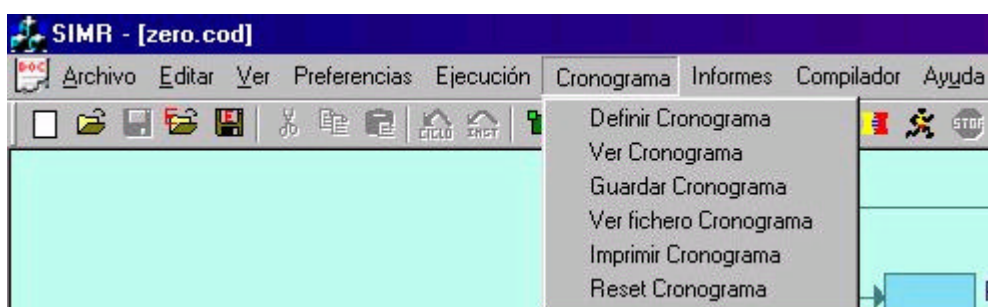
*Compilar:* Genera un fichero '.COD' con el código actual. Si hay errores los muestra en un cuadro de diálogo. Clicando sobre el error se accede directamente al error en la ventana de código, lo que facilita su identificación.

*Ejecutar y Compilar:* Realiza una compilación como en el menu anterior, pero si ésta tiene éxito, se abre una nueva instancia del programa con el programa cargado en el simulador.

*Ver Tabla de Símbolos:* Si existe una tabla de símbolos asociada con el programa actual, esta opción la muestra. (Para que exista una tabla de símbolos, el programa debe haberse compilado con éxito por lo menos una vez).

*Fichero de Macros por defecto:* Permite especificar qué fichero de macros usará el compilador. Por defecto es 'macros.mr'.

## MENU CRONOGRAMA



Este menú controla todo lo relacionado con la generación, almacenamiento y visualización de cronogramas. Dispone de las siguientes opciones

Definir Cronograma: Permite seleccionar qué señales aparecerán en el cronograma.

Ver Cronograma: Muestra la ventana del Cronograma con la ejecución hasta el momento actual.

*Guardar Cronograma:* Guarda el cronograma actual en un archivo '.CM'.

*Ver fichero Cronograma:* Carga un cronograma de un fichero en disco y lo muestra.

*Imprimir Cronograma:* Imprime el cronograma actual.

*Reset Cronograma:* Vacía toda la memoria del cronograma.

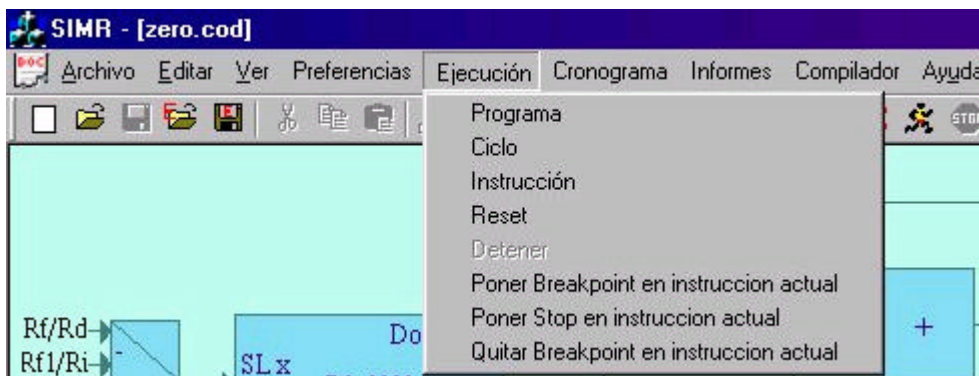
## MENU EDITAR



Este sencillo menú es muy común en Windows, pero SIMR limita su funcionamiento al momento de tratar ficheros de compilación. Dispone de las siguientes opciones:

- *Deshacer.* Tecla de Acceso directo: *Ctrl + Z*. Deshace la última acción, un ciclo en la ejecución o una acción en el editor de texto.
- *Cortar.* Tecla de Acceso directo: *Ctrl + X*. Mueve el texto seleccionado al portapapeles.
- *Copiar.* Tecla de Acceso directo: *Ctrl + C*. Copia el texto seleccionado al portapapeles.
- *Pegar.* Tecla de Acceso directo: *Ctrl + V*. Inserta el texto del portapapeles en la posición actual del cursor.

## MENU EJECUCIÓN



Desde este menú se puede controlar la forma de ejecución de un programa:

*Programa:* Esta opción (modo *run*) hará que SiMR empiece la ejecución de un programa y sólo se detenga cuando se lo indique el usuario (haciendo click sobre el botón stop, por ejemplo) o el programa haya acabado su ejecución. En la barra de herramientas, el botón [RUN](#) tiene la misma funcionalidad. La ejecución también se detendrá al encontrar un *breakpoint* o un *stop*.

*Ciclo:* Ejecuta un ciclo el programa actual. En la barra de herramientas, el botón [STEP](#) tiene la misma funcionalidad. Falta dibujar el botón

*Instrucción:* Avanza una instrucción el programa actual. En la barra de herramientas, el botón [INST](#) tiene la misma funcionalidad. Idem

*Reset:* Inicia la máquina como si el programa acabara de cargarse en memoria.

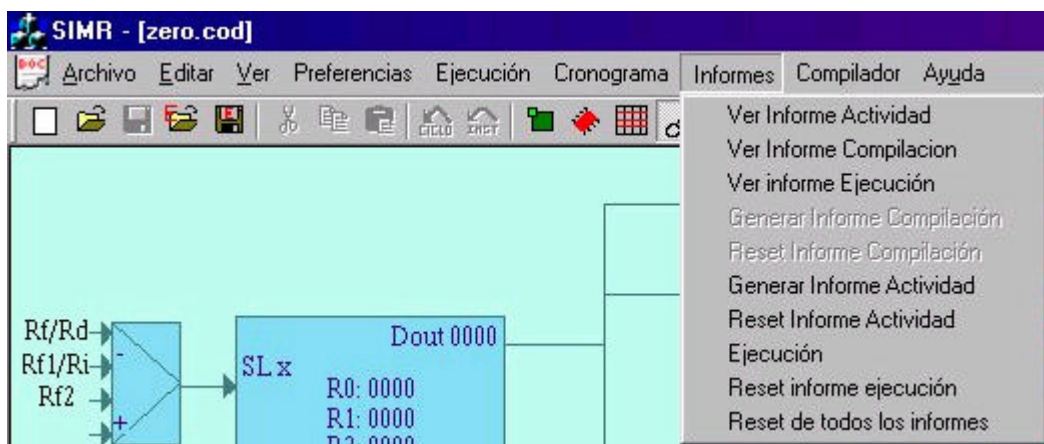
*Detener:* Detiene la ejecución del programa si se está ejecutando en modo run.

*Poner breakpoint en instrucción actual:* Inserta un punto de ruptura en la instrucción que se va a ejecutar.

*Poner stop en instrucción actual:* Inserta un punto de parada (*stop*) en la instrucción que se va a ejecutar. La diferencia entre un *stop* y un *breakpoint* es que el *stop* desaparece una vez la ejecución del programa se ha detenido en la instrucción marcada, mientras que el *breakpoint* permanece hasta que el propio usuario lo borra o se hace un *reset*.

*Quitar breakpoint en instrucción actual:* Quita el breakpoint de la instrucción que va a ejecutarse.

## MENU INFORMES



Desde este menú se controlan los informes que SiMR puede realizar. Dispone de las siguientes opciones:

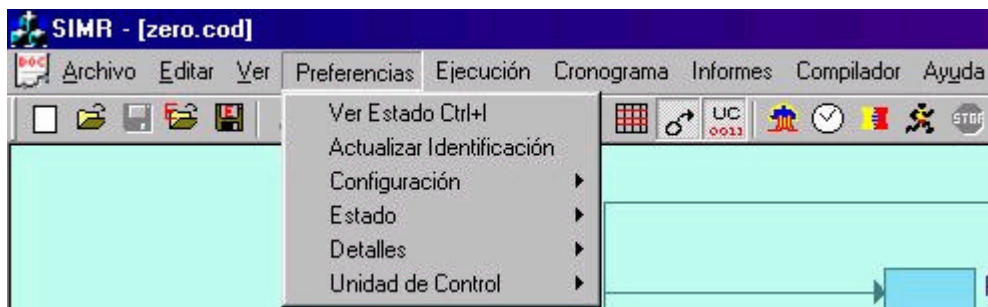
*Ver Informe Actividad, Compilación, Ejecución:* Estas opciones permiten mostrar el informe actual sobre cada tema. El informe actividad guarda todas las acciones realizadas en el simulador. El informe de compilación guarda todo lo sucedido al compilar un programa. El informe de ejecución guarda la información de como empezó la máquina y como ha acabado, su estado.

*Reset Informe Actividad, Compilación, Ejecución:* Borra toda la información almacenada hasta el momento actual relativa a los informes correspondientes.

*Compilación:* Generea el informe de compilación, que da información precisa de lo que ha sucedido mientras se compilaba: cuántas veces, qué errores, tabla de símbolos, etc.

*Ejecución:* El informe de ejecución cuenta con información sobre el estado de la máquina al empezar, al acabar, y enumera sus variaciones.

## MENU PREFERENCIAS

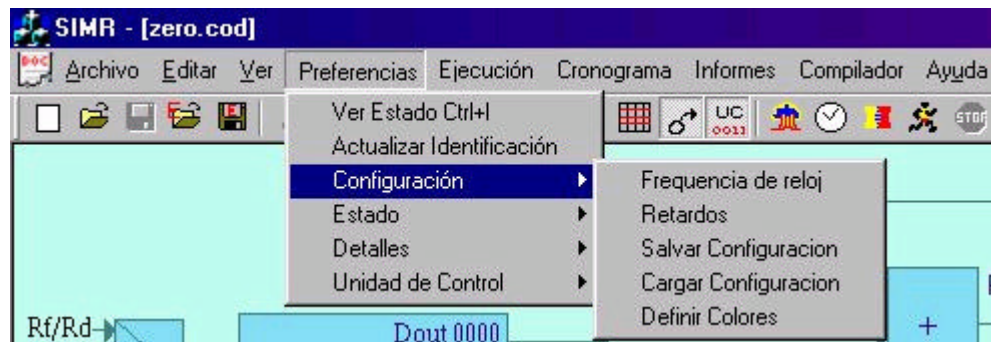


Este menú nos permite definir las principales opciones de SIMR. Dispone de las siguientes opciones:

*Ver Estado.* Tecla de Acceso directo: *ctrl. + I* . Permite modificar el estado de la MR en simulación, así como guardar este estado y cargar uno nuevo.

*Actualizar Identificación:* El usuario puede volverse a identificar si desea cambiar el nombre o algún componente del grupo.

*Configuración:*



Desde este submenú se controlan los parámetros generales de SiMR:

*Frecuencia de reloj:* Se permite cambiar el tiempo en milisegundos que hay entre ciclo y ciclo de SiMR en modo run (ejecución sin detenerse), y el tiempo total de la máquina en cada ciclo, por si se quieren variar los retardos.

*Retardos:* Todo componente de la MR tiene un retardo asociado. Desde esta opción se puede modificar cada uno de estos retardos para probar el funcionamiento de la máquina.

*Salvar Configuración:* Guarda la configuración actual de SiMR, que incluye los retardos, el tipo de UC. que se esta usando, las frecuencias de reloj y los colores.

*Cargar Configuración:* Carga una configuración de disco, con extensión '.CMR'.

*Definir Colores:* Permite especificar los colores de SiMR, para adaptar el programa a pantallas con mayor contraste o con menos tensión a la vista.

*Estado:*





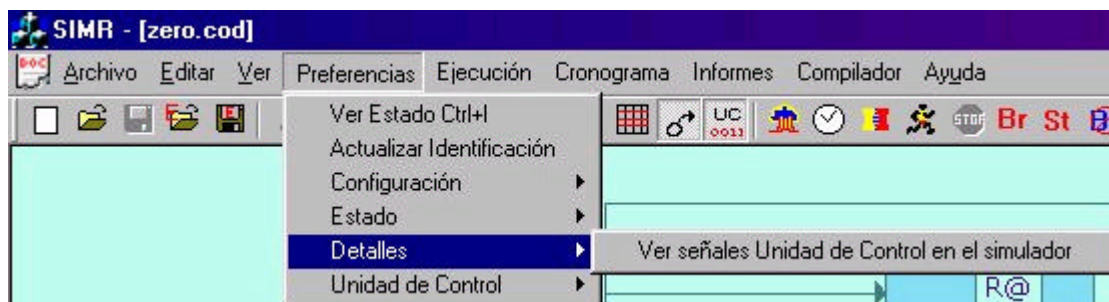
Esta opción permite tratar el estado de la máquina. Dispone de las siguientes opciones.

*Salvar Estado:* Guarda el estado actual de la máquina (valores en los registros, el estado de la UC, la memoria) en un fichero de extensión '.STT'.

*Restaurar Estado:* Carga un fichero de estado y actualiza la máquina.

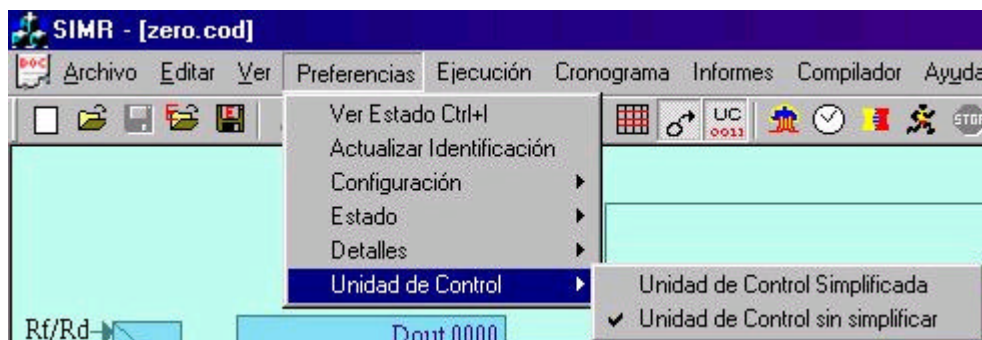
*Modificar parámetros de Estado:* Permite definir el estado de la máquina desde un solo diálogo.

*Detalles:*



En esta opción se puede activar la visualización de las señales de control en la Unidad de Proceso, lo cual aumentará la información sobre lo que está sucediendo en la máquina, pero se perderá claridad y simpleza en ventana.

*Unidad de Control:*



Se puede seleccionar qué unidad de control se quiere usar en el simulador, siendo posible cambiar entre una y otra en medio de una ejecución. El cambio también se puede realizar desde la ventana que muestra el grafo de estados de la UC.

Sada

## 2.4 VISUALIZACIÓN DEL CONTENIDO DE LA MEMORIA Y REGISTROS

El valor de cualquier posición de memoria y de cualquier registro del banco de registros (excepto el R0) puede alterarse de forma manual. Para ello, el simulador dispone de un cuadro de diálogo específico al que puede accederse desde el menú: *Preferencias -> Ver Estado*:

**Inicialización**

**Registros**

R0:	0000
R1:	0001
R2:	0004
R3:	0004
R4:	0000
R5:	0000
R6:	0000
R7:	0000

Registro PC: 0009 h

Registro IR: 880D h

Registro A: 0001 h

Registro R@: 0007 h

**Memoria**

00:	0004
01:	0003
02:	0002
03:	0001
04:	0000
05:	C925
06:	D004
07:	C121
08:	880D
09:	1900
0A:	D344
0B:	C908
0C:	8007
0D:	5004
0E:	A000
0F:	0000
10:	0000
11:	0000
12:	0000
13:	0000
14:	0000
15:	0000
16:	0000
17:	0000
18:	0000
19:	0000
1A:	0000

**Unidad de Control sin Simplificar**

☒ FETCH ☐ DECO

☐ LPO ☐ LSOE

☐ ADR1 ☐ ACS

☐ LOAD ☐ STORE

☐ ADR2 ☐ BRANCH

**Unidad de Control Simplificada**

☐ FETCH ☐ DECO

☐ LOAD ☐ STORE

☐ ARIT ☐ BRANCH

**Registros:**

RZ 0 RN 1 RV 0

OK Cancel

El cuadro de diálogo especifica qué valores hay almacenados en cada registro, *flag* o posición de memoria; los *flags* (RZ, RN y RV) tienen contenidos binarios, mientras que los valores de los registros y la memoria se presentan en hexadecimal. Cualquiera de los valores presentados puede cambiarse en cualquier momento de la ejecución de un programa, aunque no es recomendable hacerlo porque puede alterarse el resultado de la ejecución de un programa.

Este cuadro de diálogo permite alterar el estado de la máquina en cualquier momento. Ha sido diseñado exclusivamente para facilitar al alumno el poder ejecutar una instrucción específica a partir de un estado determinado de la MR. La instrucción se introduce en formato hexadecimal directamente en la memoria y se inicializan los registros, flags y demás posiciones de memoria con los valores predeterminados deseados. Finalmente, se sale de este cuadro de diálogo y se ejecuta la instrucción o grupo de instrucciones que desean analizarse.

Descripción de los Botones

OK: Una vez se han puesto los valores adecuados, el botón *OK* permite pasar estos valores al estado actual del simulador, teniendo en cuenta que los cambios en la memoria que afectan a zonas del código no se reflejarán en la ventana del código. Esto implica que se puede dejar al simulador en un estado en el cual la información aportada por la ventana de código no será fiable, por lo cual esta opción debe ser usada con extrema prudencia.

CANCEL: Si pulsamos en el botón cancel no se realizará ningún cambio sobre el estado actual del simulador.

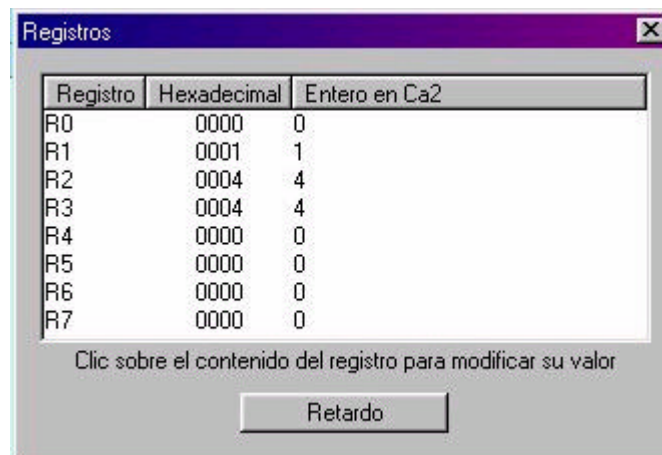
GUARDAR: El botón guardar permite guardar el estado actual de la máquina (los registros y resto de información mostrada en el cuadro de diálogo) en un fichero con extensión '.STT'. El nombre del fichero es por defecto *MiEstado.stt*, pero puede ser cambiado con facilidad desde el propio menú *GUARDAR*.

Este estado puede ser recuperado después con el botón *ABRIR* seleccionando el fichero correspondiente; una vez la información esté cargada en el cuadro de diálogo, con el botón *OK* la introduciremos en el simulador como estado actual.

## 2.5 VENTANA DE LA UNIDAD DE PROCESO

En la parte superior izquierda del datapath se muestra el contenido del banco de registros. Haciendo doble clic sobre él se abre un cuadro de diálogo que muestra los valores de los ocho registros, el valor de salida del banco de registros, el valor de entrada y de SL, o el valor que viene del multiplexor SELDAT. Los valores de los registros pueden modificarse en tiempo de ejecución en este diálogo:

### Registros del Banco de registros



Este diálogo nos permite ver el contenido de los diferentes registros que tiene el Banco de registros, modificar su valor con sólo pulsar dos veces encima de uno, y ver su valor en hexadecimal y en Ca2


El botón retardo que tiene en su parte inferior permite especificar el tiempo de retardo que el banco de registros tiene asociado

La memoria está representada por un rectángulo a la derecha del *datapath*. En él puede consultarse el valor de los buses de datos y del bus de direcciones. Para consultar el valor de todas las posiciones de memoria, así como variar su contenido en tiempo de ejecución, basta con hacer un simple doble clic encima del recuadro. Se abre entonces un cuadro de diálogo: Mirar Control de los accesos a memoria.

Las líneas que hay dibujadas en el datapath, generalmente de color gris oscuro, representan los buses por donde pasan los datos. Sobre ellas se indica el número de bits de cada bus. Si una de estas líneas está pintada en rojo significa que este bus está siendo usado en el estado actual de la Unidad de Control; el dato que esta presente en el momento actual será recogido al final del ciclo de reloj por el bloque secuencial destinatario de la flecha (la memoria, un registro o el banco de registros).

En caso de que la flecha desemboque en un bloque combinacional (ALU, multiplexor o sumador), el dato estará presente a la salida del bloque, dentro del mismo ciclo de reloj (estado), un cierto tiempo después de estar presente en su entrada. Este tiempo depende del retardo introducido por cada bloque. En esta versión del simulador se permite cambiar los retardos de cada módulo.

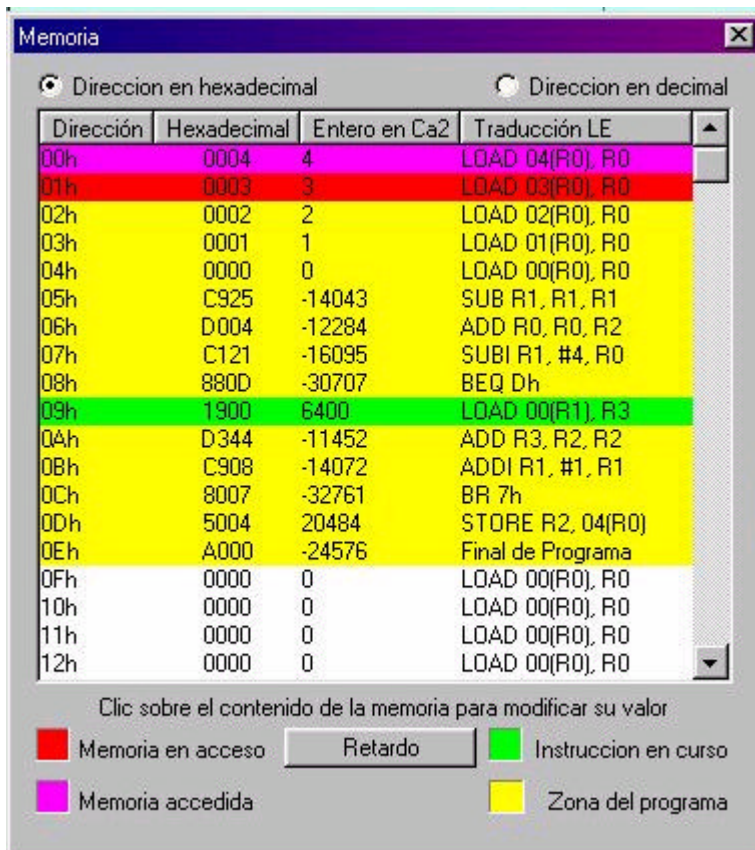
### Visualización de señales de control de la Unidad de Proceso

El funcionamiento del *datapath* se muestra con más detalle al pulsar, en la barra de botones, el que tiene el icono . Este botón permite ver las señales de control de la Unidad de Proceso. Esta opción no esta activada por defecto porque, en general, resulta molesto tener demasiados datos cambiando constantemente. No obstante, puede ser interesante activarla para observar en detalle el funcionamiento de una zona particular o de toda la Unidad de Proceso. Se desactiva volviendo a pulsar sobre ella. El menu *Preferencias -> Detalles -> Ver Señales de la Unidad de Control en el simulador* también permite ocultar o mostrar estos datos.

### [Visualización del contenido de la Memoria y Registros](#)

### CONTROL DE LOS ACCESOS A MEMORIA

Una de las nuevas funcionalidades de SIMR es la posibilidad de controlar los accesos a memoria realizados por el programa mediante la siguiente ventana de memoria:





Esta ventana permite realizar un seguimiento preciso del progreso del programa. En esta ventana podemos observar dos *Radio-button* en la parte superior. Con ellos se puede elegir si la dirección se representa en formato decimal o hexadecimal. En la parte inferior puede verse una leyenda que describe el significado de cada color de fondo:

el amarillo corresponde a la zona de instrucciones del programa. Corresponde a las direcciones de memoria que van desde la dirección especificada por la directiva '.BEGIN' hasta la posición en la que se encuentra la directiva '.END'

la línea verde marca la instrucción en curso

la línea roja señala la posición de memoria que se está accediendo con la instrucción en curso, en el supuesto de que se esté ejecutando una instrucción LOAD o STORE.

las marcas lilas corresponden a las posiciones que, habiendo sido definidas como zona de datos (con directivas '.DW' y '.RW') han sido accedidas en el transcurso del programa. La utilidad de estas marcas es la de comprobar que todos los accesos se hayan hecho en zona de datos y no se ha sobrescrito ninguna posición de código o alguna posición errática.

Esta ventana permite además al usuario cambiar los valores de las posiciones de memoria, ya sea de datos o código. Esta opción debe ser usada con extremo cuidado, ya que el programa resultante podría ser incorrecto, no tener fin o no hacer lo que debiera.

Para facilitar la comprensión de las líneas de código en memoria, se ha incluido una columna en la ventana que muestra la instrucción que sería interpretada por el simulador con el contenido de esa posición de memoria. Las traducciones no incluyen variables ni etiquetas, ya que esta información no es deducible del valor almacenado en memoria; por eso, el código no tiene por qué coincidir con el que puede observarse en la ventana de código.

Cada posición de la memoria indica su dirección, su contenido en Hexadecimal y en Ca2.

El botón que hay abajo permite especificar el retardo de la memoria.

## 2.6 VENTANA DE CÓDIGO

---

La ventana de código contiene el código ensamblador del programa que está cargado en la memoria de la MR. En esta ventana pueden seguirse los avances de la ejecución del programa, ya que la instrucción en curso se resalta pintando el fondo de negro. En condiciones normales, la ventana tiene un fondo azul y las letras se escriben en amarillo.

- Breakpoints
- Macros: expansión y compresión

### **BREAKPOINTS**

Un breakpoint (punto de ruptura) es una marca que se pone en una instrucción para que la ejecución del programa se detenga justo en el instante anterior a la ejecución de dicha instrucción.

Se pueden poner dos tipos de *BreakPoints* en SIMR:

Permanentes: Un *breakpoint* permanente provoca la detención de la simulación cada vez que se ejecute la instrucción marcada.

De paso: Este *breakpoint* provoca la detención de la simulación en la instrucción marcada únicamente cuando el programa la ejecute la primera vez, pero no las sucesivas. Una vez ejecutada la instrucción, el *breakpoint* desaparece.

#### Breakpoints permanentes

Se pueden poner *breakpoints* permanentes en cualquier línea del código del programa mediante el siguiente proceso:

1. hacer clic con el botón derecho del ratón sobre la línea donde queremos poner el *breakpoint*
2. seleccionar la opción de “Poner *Breakpoint*”

Las líneas en las que hay una marca de *breakpoint* quedan resaltadas con el fondo de color rojo.

Para quitar un *breakpoint* permanente hay que seguir el mismo proceso, pero esta vez hay que seleccionar en el menú la opción de “Quitar *Breakpoint*”.

1. hacer clic con el botón derecho del ratón sobre la línea donde queremos poner el *breakpoint*
2. seleccionar la opción de “Quitar *Breakpoint*”

#### Breakpoints de paso

Se pueden poner *breakpoints* de paso en cualquier línea del código del programa mediante el siguiente proceso:

1. hacer clic con el botón derecho del ratón sobre la línea donde queremos poner el *breakpoint*
2. seleccionar la opción de “Poner *Stop*”

Las líneas en las que hay una marca de *breakpoint* quedan resaltadas con el fondo de color rosa.

### **MACROS: EXPANSIÓN Y COMPRESIÓN**


La ventana de código también gestiona las macros incluidas en el código. Para diferenciarlas de las instrucciones elementales de la MR, las macros se presentan pintadas con azul claro.

Es posible expandir y comprimir una macro sobre la propia ventana de código, de forma que podamos ver la secuencia de instrucciones que la forman o sólo su nombre.

- Haciendo doble clic encima de la macro se produce la expansión del código de la macro, es decir, la macro muestra todas las instrucciones elementales y/o macros que la forman. Esta expansión se presenta con una tabulación en las líneas expandidas para indicar que se trata del código de la macro, lo cual facilita su lectura.
- Con otro doble clic en la cabecera de una macro expandida ésta se contrae de nuevo en una sola línea, provocando además la contracción de todas las macros que pudiera contener.

Mientras está ejecutándose cualquiera de las instrucciones de una macro, ésta permanece resaltada con fondo negro y letras blancas.

Si el código interno de las macros resulta molesto y tan solo se quiere observar el código ensamblador y las llamadas a las macros, a la derecha de la ventana de código


hay un botón  que permite ocultar todas las macros con un clic. Este proceso es reversible haciendo clic de nuevo sobre el mismo botón.

Si hay breakpoints dentro de una macro, toda ella quedará resaltada en fondo rojo, pero el breakpoint se ejecutará en la instrucción definida, no al principio de la macro. Si se define un breakpoint en una macro sin expandir, ésta pondrá el breakpoint en la primera instrucción que contenga. No es posible definir un breakpoint en el nombre de una macro expandida, pues la cabecera solo es una ayuda gráfica, no representa ninguna instrucción.

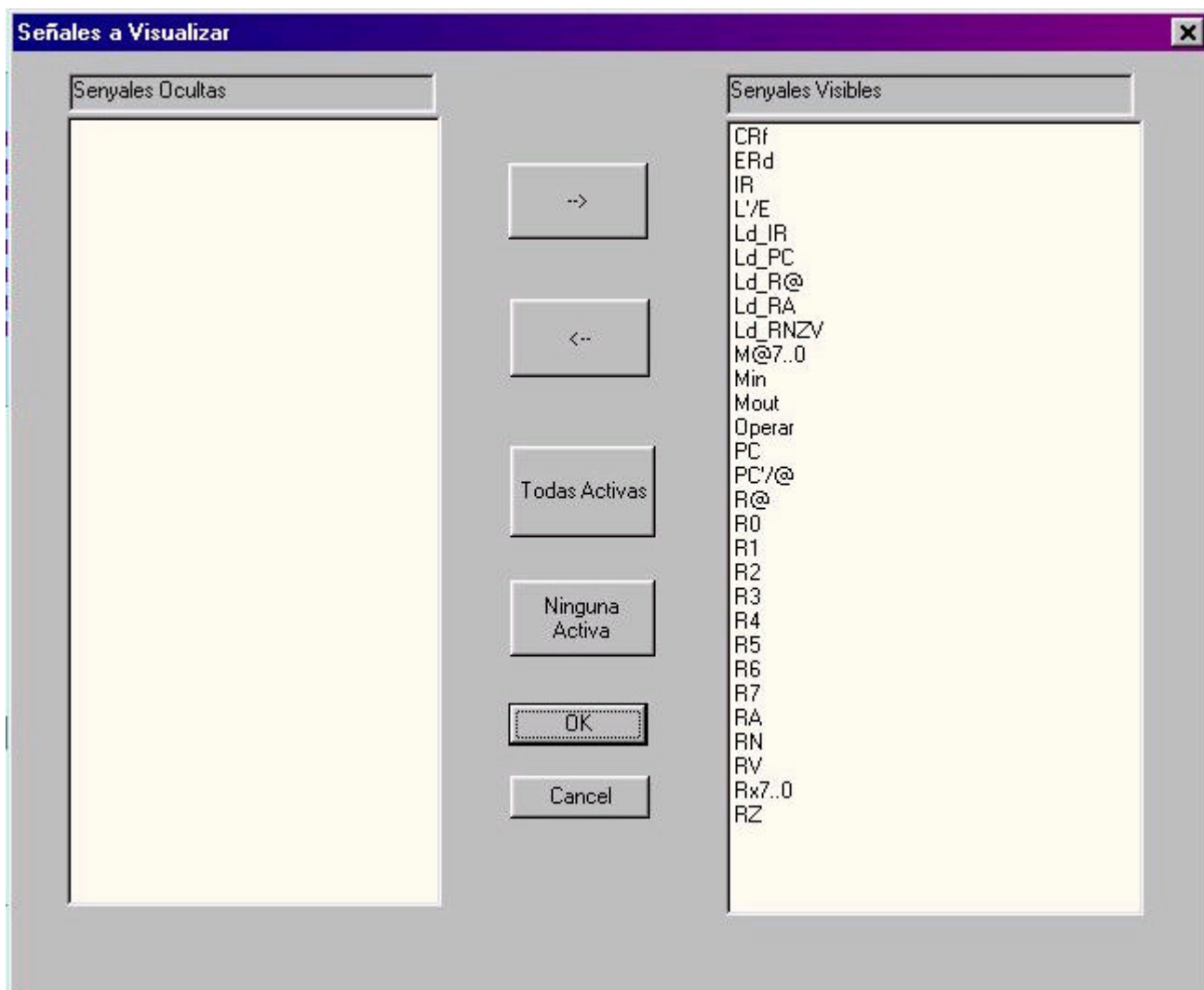
## 2.7 VENTANA CRONOGRAMA

---

SIMR permite ver la evolución temporal tanto del valor de las señales de la Unidad de Control como el valor de los Registros de la Unidad de Proceso y de algunos buses de datos. A esta ventana se le denomina *cronograma*.

Existe una ventana de cronograma por cada fichero '.COD' que esté abierto (varias instancias del simulador pueden estar siendo ejecutadas simultáneamente con diferentes programas cargados). Puede accederse a la ventana de cronograma desde el menú *Cronograma à Ver Cronograma*, o pulsando en  de la barra de botones.

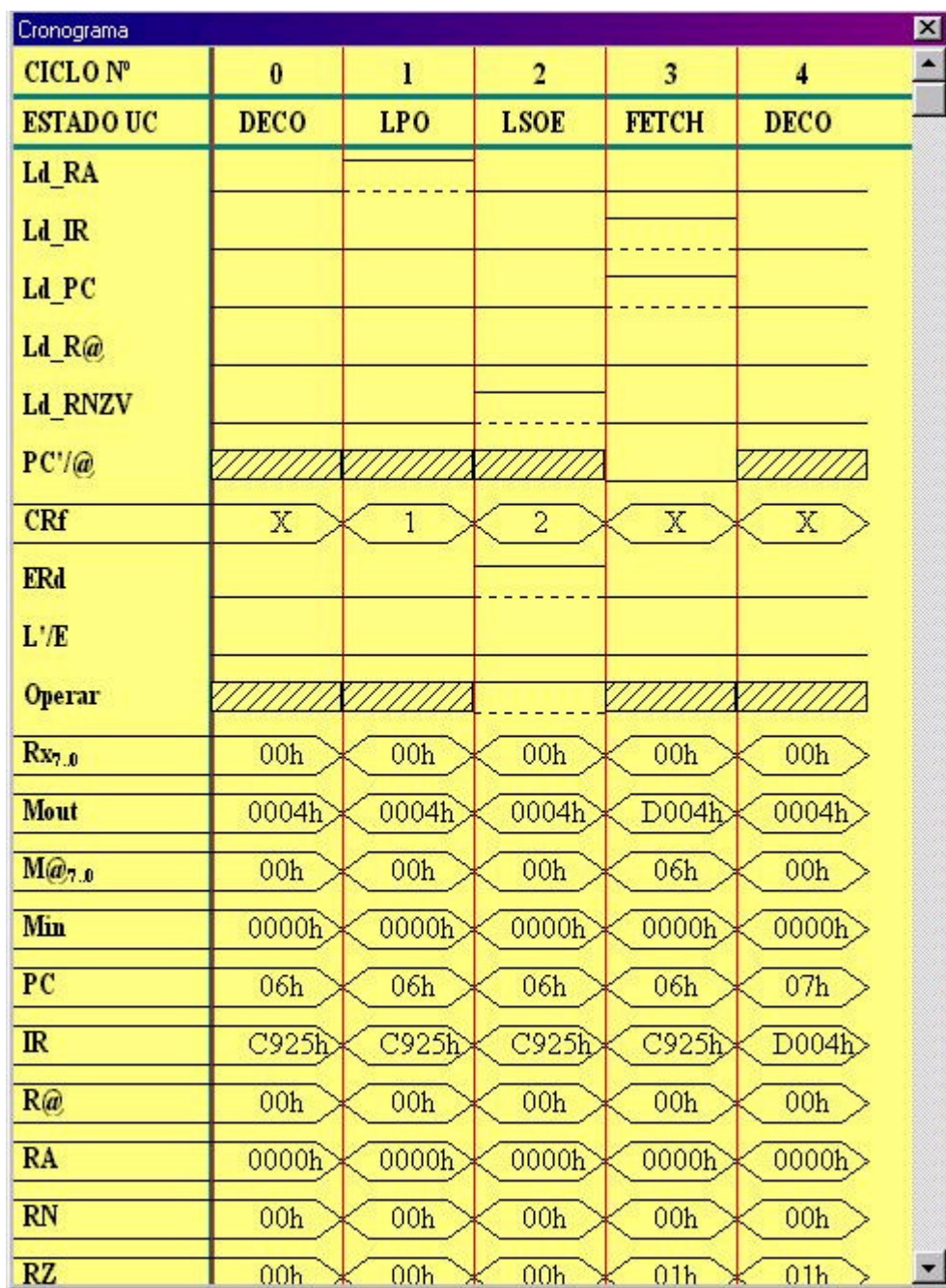
Para decidir las señales y registros que aparecen en el cronograma se dispone de un cuadro de diálogo al que se accede desde la opción *Cronograma -> Definir Cronograma*.



En este intuitivo cuadro de diálogo se muestran dos listas, la de señales *fuera* (no incluidas en el cronograma) y la de señales *dentro* (señales que aparecerán en el cronograma). Es posible seleccionar una o más señales de cualquier grupo y pasarlas al otro grupo. Las señales seleccionadas son una opción del programa, no del fichero '.COD', por lo que un cambio de definición de los grupos de señales afectará a todos los cronogramas futuros.

Los botones de --> y <-- sirven para mover las señales seleccionadas a una lista o otra, respectivamente. El botón *Todas Activas* mueve todas las señales a la derecha, y el botón *Ninguna Activa* las mueve todas a la izquierda.

A continuación se muestra un ejemplo de cronograma:



En el cronograma anterior se puede observar la evolución a través del tiempo de todas las señales seleccionadas. En general, las señales binarias de control se presentan en forma de onda cuadrada, que refleja los cambios de 0 a 1 y viceversa. Una señal binaria no definida en un ciclo se presenta con una zona sombreada en todo el ciclo. Hay señales binarias -como los *flags*- que se representan como un 0 ó un 1, o numéricas -como los registros- en los que se utiliza la representación hexadecimal para indicar el valor de sus 16 bits.

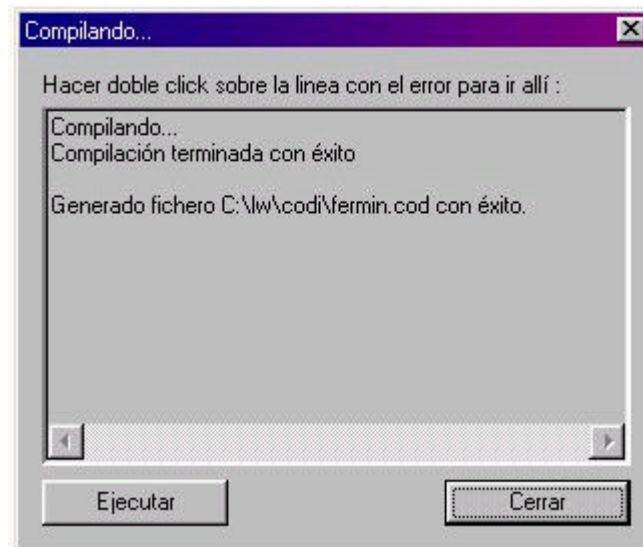
## 2.8 COMPILADOR

El compilador, que en esta nueva versión se ha integrado con el simulador, es una fusión de los dos pasos que usaban las versiones anteriores: el precompilador -que transformaba el código de las macros de un fichero '.MR' a código ensamblador en un fichero '.ASM'- y el postcompilador -que transformaba el código ensamblador de un

fichero '.ASM' a código máquina en un fichero '.cod'-; mantiene, por tanto, compatibilidad con programas escritos para versiones anteriores.

En la ventana de código se puede editar un fichero del mismo modo que se haría si se usase el *notepad* de Windows.

Los ficheros con extensión '.asm' no realizarán el paso del precompilador.



Cuando se compila, aparece un cuadro de diálogo que contiene los mensajes producidos durante el proceso de compilación. Si la compilación se ha realizado con éxito, se puede pulsar el botón “Ejecutar” en el diálogo para arrancar directamente el simulador ejecutando el fichero '.cod' generado en la compilación. En caso de que el programa contenga errores sintácticos, en el cuadro de diálogo se mostrarán los mensajes de error, indicando la línea que los ha provocado y el tipo de error de que se trata. Haciendo doble clic encima de la línea que muestra el error, se resaltará en la ventana de edición del código la línea conflictiva.

## 2.9 TIPOS DE ARCHIVO

---

### **.COD**

Esta es la extensión de ficheros compilados (código máquina). Se pueden abrir con SIMR y realizar su ejecución paso a paso, ver cronogramas y las funcionalidades de SIMR. La información contenida en este fichero es el código fuente con el que se compiló, el código objeto, un estado inicial de la memoria y el valor inicial del PC.

### **.MR, .ASM**

Con estas extensiones trabaja el compilador, siendo la extensión MR la mas usada pues permite el uso de las macros. Cuando se compila con esta extensión se genera un archivo intermedio '.ASM' que el segundo paso del compilador transformará en un archivo '.COD'. Si el fichero tiene ya inicialmente la extensión .ASM, el primer paso de compilación se omitirá. Los dos ficheros son de tipo texto, editables desde MS-DOS.

### **.STT**

Este fichero define un estado de la MR, es decir, guarda el contenido de la memoria, del banco de registros y de todos los flags y registros de la Unidad de Proceso. No guarda el código fuente del programa.

### **.CMR**

Esta extensión identifica a los ficheros de configuración que guardan los retardos, las frecuencias y los colores de SiMR.

### **.CM**

Esta extensión sirve para guardar un cronograma.

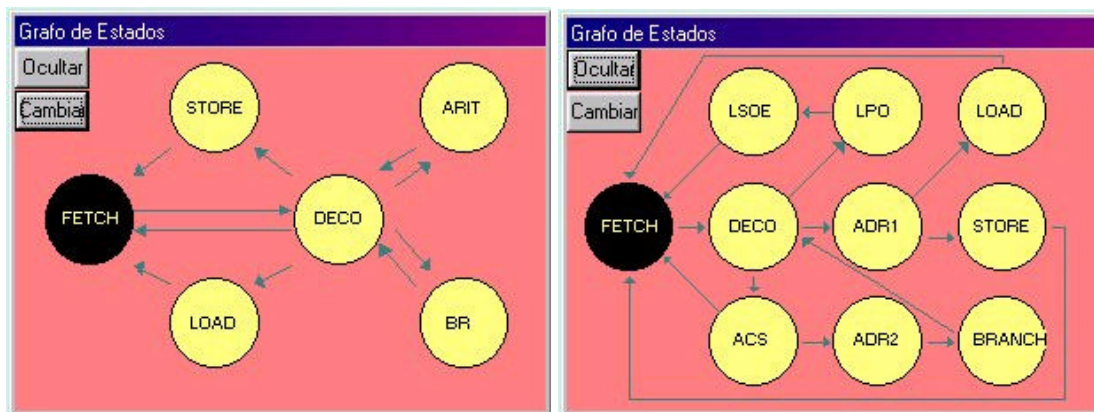
### **.CMP**

Esta es la extensión de los ficheros de informe de compilación.

### **.RUN**

Esta es la extensión de los ficheros de informe de ejecución.

## 2.10 CUADRO DE DIÁLOGO DE LA UNIDAD DE CONTROL



Este diálogo muestra (resaltado con fondo negro) en todo momento el estado actual de la UC, y se puede alternar entre ellos en cualquier momento con solo pulsar el botón *cambiar*. La opción del menú *Preferencias -> Unidad de Control -> \** también permite escoger la UC. Este cambio solo afecta a la UC, el simulador funciona igual con una UC que con otra. Para ocultar el diálogo se puede usar el botón *ocultar*, lo mismo que en el menú *Ver -> Grafo de Estados*.