Manual de Usuario

Simulador de Símplez

Revisión 1.0

Autor: Iván Peña Rueda

Contenidos

Prefacio

Símplez es un ordenador didáctico diseñado y desarrollado por Gregorio Fernández en su libro de texto "Conceptos básicos de Arquitectura y Sistemas Operativos", [Fernández,98].

Gracias al simulador descrito a lo largo de este manual, se pueden escribir, ensamblar y ejecutar programas en Símplez. El simulador ofrece varios modos de ejecución y un conjunto completo de herramientas que permiten al usuario entender a fondo el funcionamiento de Símplez.

El simulador está desarrollado en Java. Lo que permite ejecutar el mismo binario en diferentes plataformas, y superar una de las limitaciones de los simuladores de Símplez existentes hasta la fecha.

Usuarios de este manual

Este manual está destinado a toda persona interesada en aprender conceptos básicos de arquitectura de computadores a través del ordenador ficticio Símplez. Aunque el manual trata de cubrir todos los aspectos teóricos necesarios para poder utilizar Símplez, es altamente recomendable el estudio de algunas de las lecciones del libro de referencia [Fernández,98].

El manual está disponible en varios formatos en la página de web de la asignatura de **Fundamentos de Ordenadores**, **Departamento de Ingeniería Telemática** de la **Escuela Superior de Ingenieros de Telecomunicación** de la **Universidad Politécnica de Madrid**. La URL de la citada escuela es:

`http://www.etsit.upm.es`

Requerimientos mínimos

Como requisito previo a la instalación y utilización del simulador, es necesario tener disponible en el sistema el conjunto de herramientas JDK, con una versión igual o superior a la 1.2. Éste se puede obtener de la URL:

```
http://java.sun.com
```

Una vez instalado el jdk se tendrá un directorio:

```
c:\jdk1.2\bin
```

en el caso de Windows 95/98/NT, y algo parecido a:

```
/usr/java1.2/bin
```

en el caso de Unix/Linux.

Para independizar las explicaciones respecto a la plataforma, se utilizará $JAVAHOME para referirse a estos directorios.

En el Manual, para hablar de directorios, se usará la terminología de Unix, por lo que la barra inclinada utilizada para separarlos será el carácter '/'. En el caso de Windows se debe utilizar '\'.

El simulador se ejecuta adecuadamente en un Pentium 133 Mhz con 32 Mb de memoria, pero es aconsejable utilizar uno de velocidad superior y 64 Mb de memoria.

Para un correcto funcionamiento de ventanas como la de Pantalla y Teclado, se recomienda tener instalados los diferentes tamaños de letra de la fuente **Courier**.

Instalación

El simulador se encuentra empaquetado en un único fichero. El compresor utilizado es el comando **jar**, incluido entre las herramientas de la máquina virtual de Java, y que utiliza el formato zip. Dicho fichero tendrá extensión jar:

```
simulador.jar
```

El primer paso para instalarlo consiste en desempaquetar el directorio de imágenes dentro del directorio de instalación, para lo cual se ejecutará desde línea de comandos:

```
$JAVAHOME/jar xvf simulador.jar Imagenes
```

Tras esto, en el directorio de instalación deben existir tanto el fichero ’simulador.jar’ como el directorio ’Imagenes’, ambos necesarios.

Para arrancar el simulador se ejecutará:

```
$JAVAHOME/java −jar simulador.jar
```

Organización del manual

El manual comienza describiendo el entorno de simulación y sus componentes. Esto permite al usuario tener una visión global del mismo.

En los siguientes capítulos se abordan las diferentes maneras en que se puede introducir y ejecutar un programa en el simulador.

Los siguientes cuatro capítulos están dedicados a describir en detalle la utilización de las diferentes herramientas que incluye el simulador: Ruta de Datos, Cronogramas, módulo de Pantalla y Teclado, editor de programas y editor de Memorias de Control o microprogramas.

Por último se introducen la programación y microprogramación de Símplez utilizando el simulador.

Convenciones tipográficas

En la Tabla 1 se describen las convenciones tipográficas utilizadas en este Manual.

Tipo de letra	**Significado**
`abdefg12345`	Comandos de sistema y secciones de código
"abcdefg12345"	Referencias

Tabla 1: Convenciones tipográficas del Manual.

Cuando haya que indicar algo importante se resaltará como una nota con el siguiente formato:

Nota− Esto es algo importante a destacar.

Bibliografía

[Fernández,98] Conceptos básicos de arquitectura y sistemas operativos. Curso de Ordenadores. (2da edición) Ed. Syserso. Madrid 1998.

[Patterson−Hennessy,95] Organización y diseño de computadores. La interfaz hardware/software. (2da edición) Ed. McGraw−Hill. España 1995.

Capítulo 1

El entorno de simulación

El simulador permite escribir y ejecutar programas de Símplez a nivel de máquina convencional. Para escribirlos se pueden introducir directamente en memoria, utilizando lenguaje máquina, o utilizar el editor desarrollado por Francisco Javier Rodríguez incluido en el simulador, utilizando lenguaje ensamblador.

Incluye un editor de Memorias de Control, de manera que es posible microprogramar el secuenciador de Símplez. Es decir, es posible programar a nivel de micromáquina y modificar el repertorio de instrucciones de Símplez. A este nivel controlamos el comportamiento de los elementos hardware de la máquina y decidimos qué queremos que hagan y cuándo. La microprogramación no siempre es sencilla y puede acarrear conflictos entre los diferentes elementos. Todos estos conceptos se explican en la segunda parte de [Fernández,98], y para el caso concreto de Símplez en la lección 11.

Para facilitar la comprensión de los conceptos de microprogramación la herramienta incluye un simulador gráfico, que permite seguir la ejecución a nivel de Ruta de Datos, y un generador de cronogramas, que permite ver cómo evolucionan las señales a lo largo del tiempo.

Para interactuar con el usuario el simulador proporciona la pantalla y el teclado de Símplez

En este capítulo se describe cada uno de estos módulos y sus diferentes componentes.

1.1 Ventanas del Simulador

El simulador se compone de siete ventanas principales, identificadas por los nombres que aparecen en el borde superior de las mismas. Además, ante determinadas operaciones o ante ciertas situaciones se abren otros dos tipos de ventanas, los llamados Diálogos modales y las ventanas de selección de ficheros.

Simulador de Símplez

Es la ventana principal del simulador y desde la cual se puede acceder a toda su funcionalidad se muestra en la Figura 1.

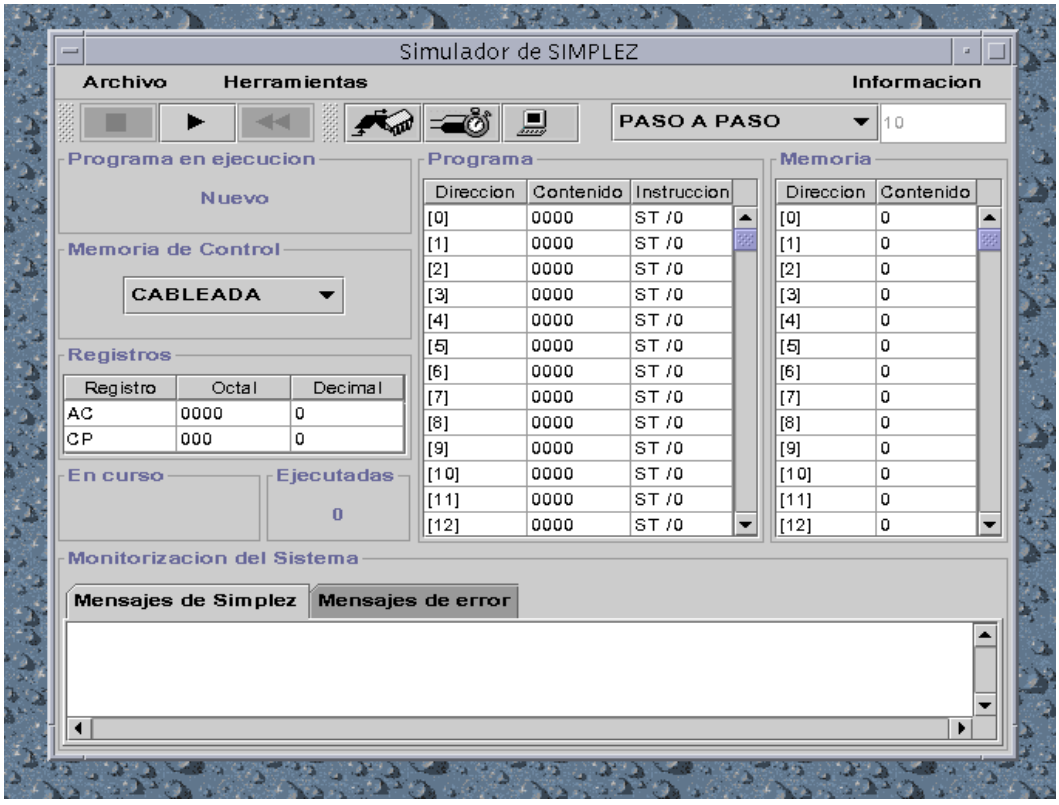


Figura 1: Ventana principal del Simulador.

La ventana se compone de una barra de menús y el cuerpo principal de la misma. Éste está a su vez dividido en 3 zonas diferentes; una barra de herramientas, un área que muestra el estado de ejecución de la máquina y un área de monitorización del Simulador.

Barra de Menús

La barra de menús se muestra en la Figura 2.

Figura 2: Barra de menús.

Desde ella se accede a los tres menús siguientes:

Menú de Archivo

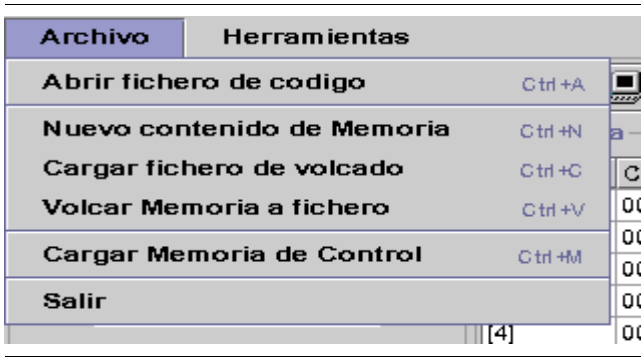


Figura 3: Menú de Archivo.

- **Abrir fichero de código.** Sustituye el contenido de la memoria por el código que contiene un fichero con extensión **.lns**, generado por el editor de programas.

- **Nuevo Contenido de memoria.** Borra el contenido de la memoria, introduciendo en todas sus posiciones el valor cero.

- **Volcar Memoria a Fichero.** Guarda el contenido de toda la memoria en un fichero de extensión **.vms**. En algunos casos puede ser interesante guardar el contenido actual de la memoria antes de realizar tales modificaciones en el programa fuente.

- **Cargar Fichero de volcado.** Sustituye el contenido de la memoria con el contenido de un fichero de volcado.

- **Cargar Memoria de Control.** Sustituye el contenido actual de la Memoria de Control por los microprogramas contenidos en un fichero de extensión **.mcs**.

- **Salir.** Permite salir del Simulador.

Menú de Herramientas

Figura 4: Menú de Herramientas.

- **Editor de programas.** Herramienta que permite escribir y ensamblar programas para Símplez.
- **Editor de Microprogramas.** Herramienta que permite escribir microprogramas para la Memoria de Control de Símplez.
- **Ocultar/Mostrar paneles de Monitorización.** Permite ocultar en cualquier momento los paneles de mensajes de Símplez, de manera que la ventana principal del simulador ocupe menos espacio en pantalla.

Ayuda

Figura 5: Menú de Ayuda.

- **Versión.** Informa sobre la versión del Simulador.
- **Sobre el Autor.** Informa sobre los autores del Simulador.

Barras de Herramientas

En primer término se encuentran las barras de herramientas, mostradas en la Figura 6, desde las que se puede controlar la ejecución de la simulación y abrir el resto de módulos que componen el Simulador.

Figura 6: Barras de herramientas.

Los diferentes controles que componen la primera barra de herramientas son:

Parada. Detener de la ejecución.

Arranque. Comenzar la ejecución.

Deshacer la ejecución de una instrucción.

Los controles que componen la segunda barra de herramientas son:

Abrir/Cerrar la ventana de la **Ruta de Datos**.

Abrir/Cerrar la ventana de los **Cronogramas**.

Abrir/Cerrar la ventana de **Pantalla y Teclado**.

Selector de **Modo de Ejecución**.

Número de instrucciones a ejecutar seguidas en el modo **Continuo Hasta**.

Área de estado de ejecución

A continuación se encuentra el área mostrada en la Figura 7 y que contiene información sobre el estado de ejecución de la máquina. Desde aquí se puede también modificar parte de ese estado.

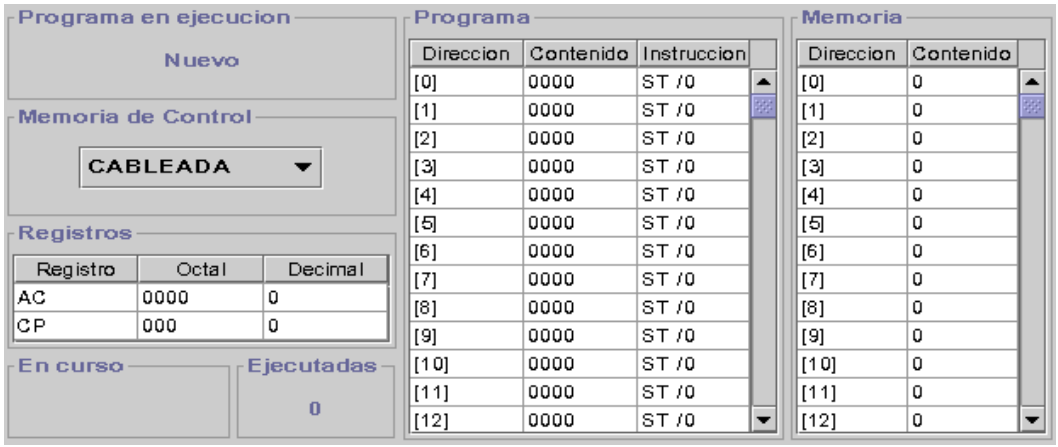


Figura 7: Área que muestra el estado de la ejecución.

Se compone de los siguientes elementos:

- **Programa en ejecución.** Nombre del programa cargado en ese momento en memoria. En el caso de no haber cargado ninguno aparecerá la etiqueta "Nuevo".

- **Registros.** Contenido de los registros de sistema. Los formatos de visualización son octal y decimal. Esta tabla es editable.

- **En curso.** Campo que muestra la instrucción actualmente en ejecución.

- **Ejecutadas.** En este campo se indica el número de instrucciones ejecutadas desde la última vez que se reseteó el estado de la máquina. El significado exacto del reseteo y las situaciones en las que se produce se explicarán en el Capítulo 3.

- **Programa.** Tabla que muestra el contenido de la memoria. Se compone de tres columnas. La primera muestra la dirección de memoria, la segunda su contenido en notación octal y la tercera muestra el contenido interpretado como una instrucción de Símplez. Esta tabla es editable.

- **Memoria.** Tabla que muestra el contenido de la memoria. Se compone de dos columnas. La primera es la dirección de memoria y la segunda el contenido en notación decimal. Esta tabla es editable.

- **Memoria de Control.** Selector que permite cambiar la Memoria de Control cargada en el Secuenciador. Cuando se carga una nueva Memoria de Control se añade a lista desplegable tras las dos Memorias que implementa el Simulador por defecto. Se pueden tener cargadas hasta diez memorias, contando las dos anteriores. Al cargar una más se elimina la última de la lista. La Figura 8 muestra el selector tras haber cargado desde fichero dos Memorias de Control, "Prueba 2" y "Prueba 1".

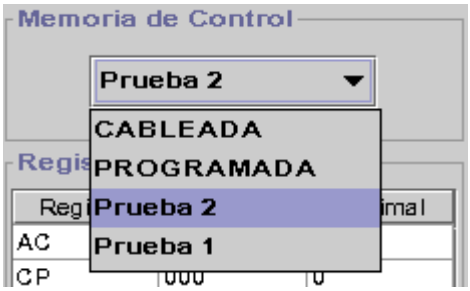


Figura 8: Selector de Memorias de Control.

Área de monitorización

Por último se encuentra el área de Monitorización del Sistema, compuesta por dos solapas y mostrada en la Figura 9:

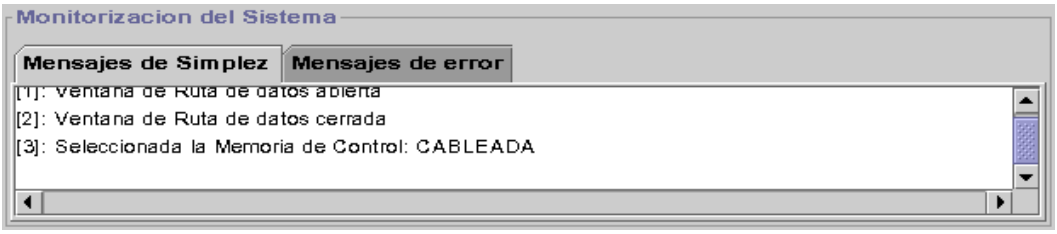


Figura 9: Área de monitorización del Simulador.

La solapa **Mensajes de Símplez** mostrará mensajes relacionados con el entorno gráfico y con la ejecución. Así por ejemplo, algunas situaciones en las que se mostrarán mensajes son la carga de un nuevo programa en la memoria, la selección de una determinada Memoria de Control, la interrupción de la ejecución de una instrucción y la apertura o cierre de ciertas ventanas.

Pero el verdadero valor de esta solapa aparece en ejecución. Cuando la Ruta de Datos o los Cronogramas están abiertos, se van mostrando mensajes que ayudan a entender qué ocurre en la máquina Símplez a medida que avanza la ejecución; indicando qué cambios se van produciendo en los diferentes elementos.

En la solapa de **Mensajes de error** se mostrarán los motivos de operaciones fallidas de edición, carga de ficheros, etcétera, como parte complementaria a los diálogos modales.

Una propiedad interesante de este área de monitorización es que se puede ocultar en cualquier momento que sea necesario, lo que permite disminuir el área que ocupa la ventana principal.

Ruta de Datos

Esta ventana muestra la Ruta de Datos de Símplez. Para abrirla basta con pulsar el botón indicado en el apartado que describe la barra de herramientas de la ventana principal del simulador. El botón cambia de aspecto permitiendo saber si la ventana está abierta o cerrada.

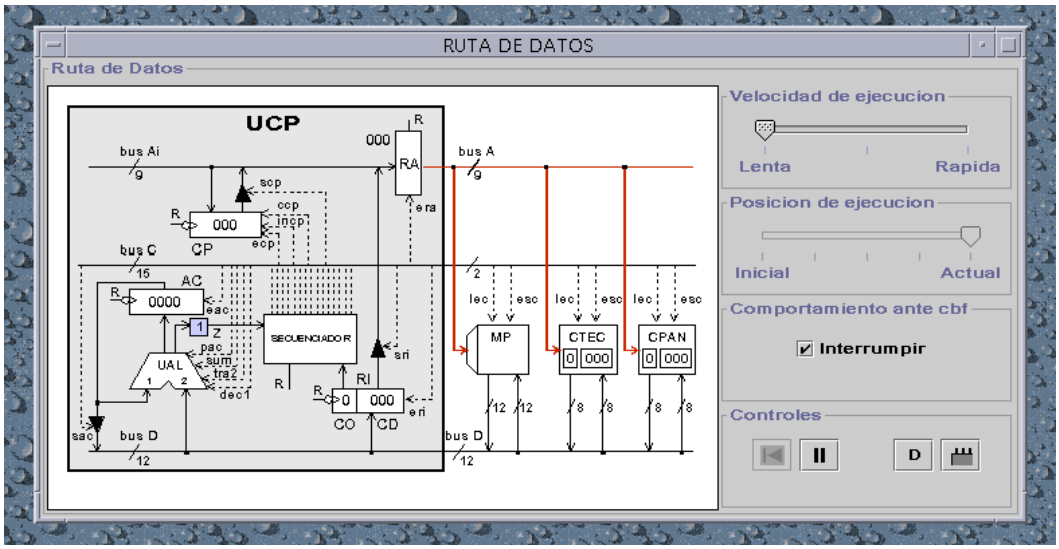


Figura 10: Ventana de Ruta de Datos.

La ventana se divide en dos partes. En la de la izquierda se representa la Ruta de Datos de Símplez. Ésta dependerá del tipo de secuenciador seleccionado. En la Figura 10 se representa la Ruta de Datos correspondiente al secuenciador cableado y en la Figura 11 el detalle de la diferencia correspondiente al secuenciador microprogramado.

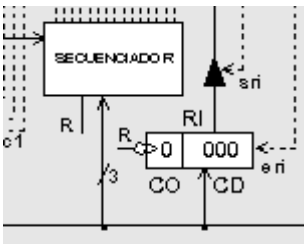


Figura 11: Detalle de la Ruta de Datos.

En el Capítulo 4 se describe en detalle el comportamiento de cada elemento de la Ruta de Datos y sus diferentes estados.

La parte derecha de la ventana es un área de control que permiten modificar el estado de ejecución de la instrucción en curso.

La primera barra de desplazamiento permite modificar la **Velocidad de ejecución**. Desplazando la misma hacia la derecha se acelera la acción de la Ruta de Datos.

La segunda barra de desplazamiento permite cambiar el instante actual de ejecución de la Ruta de Datos, llamado **Posición de ejecución**, a otro dentro del ciclo de instrucción ya ejecutado.

Debajo de las barras de desplazamiento encontramos un cuadro seleccionable llamado **Comportamiento ante cbf**. Este cuadro permite indicar qué se debe hacer al encontrar la señal **cbf** cuando trabajamos con secuenciadores microprogramados (para el secuenciador cableado no influye). Si está señalado, al segundo ciclo de reloj con **cbf** activa se interrumpe la ejecución. Si no está señalado, la ejecución continúa a pesar de **cbf**. Así por ejemplo, la instrucción HALT del secuenciador microprogramado por defecto se ejecutaría eternamente.

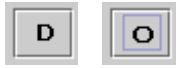
En la parte inferior de encuentra el área de control de la Figura 12.

Figura 12: Área de Control de la Ruta de Datos.

Los botones que la componen:

Retroceso.

Pausa y **continuación**.



Selección de **formato numérico**.

Abrir/cerrar la ventana de los **Secuenciadores.**

El **botón de retroceso** permite llevar la posición de ejecución al comienzo de la instrucción. Es equivalente a desplazar la barra de **Posición de ejecución** hasta el extremo de la izquierda.

El **botón de pausa** permite detener momentáneamente la simulación de la instrucción. Cuando se pulsa, el aspecto del botón cambia para mostrar el icono de continuación de ejecución.

Los botones de **selección de formato numérico** permiten elegir entre los formatos decimal y octal para mostrar los valores numéricos presentes en la Ruta de Datos. Cuando se pulsa el aspecto del botón cambia para indicar el formato alternativo.

El último botón permite abrir la **ventana de los secuenciadores**. Una vez pulsado su aspecto cambia indicando que dicha ventana está abierta.

Secuenciador

Esta ventana muestra los diferentes estados del secuenciador durante la ejecución de una instrucción. Es accesible a través del botón del área de control de la ventana de la Ruta de Datos.

El tipo de secuenciador mostrado depende de si la Memoria de Control escogida a través del selector de la ventana principal es **cableada** o **microprogramada**. De acuerdo a estos dos valores, los dos aspectos posibles de la ventana del Secuenciador se muestran en la Figura 13 y en la Figura 14.

Esta ventana depende de la de la Ruta de Datos. Si se cierra esta última, la ventana del Secuenciador también se oculta.

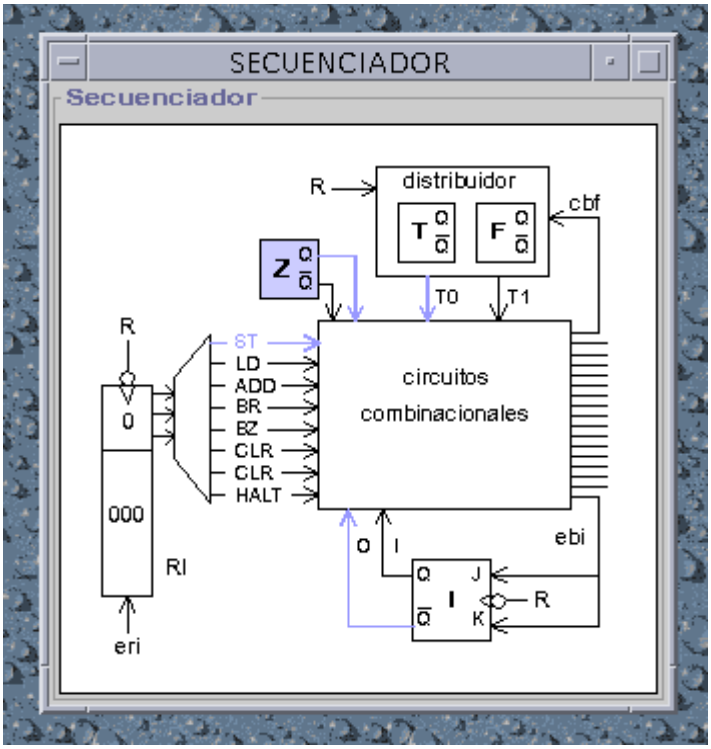


Figura 13: Secuenciador cableado.

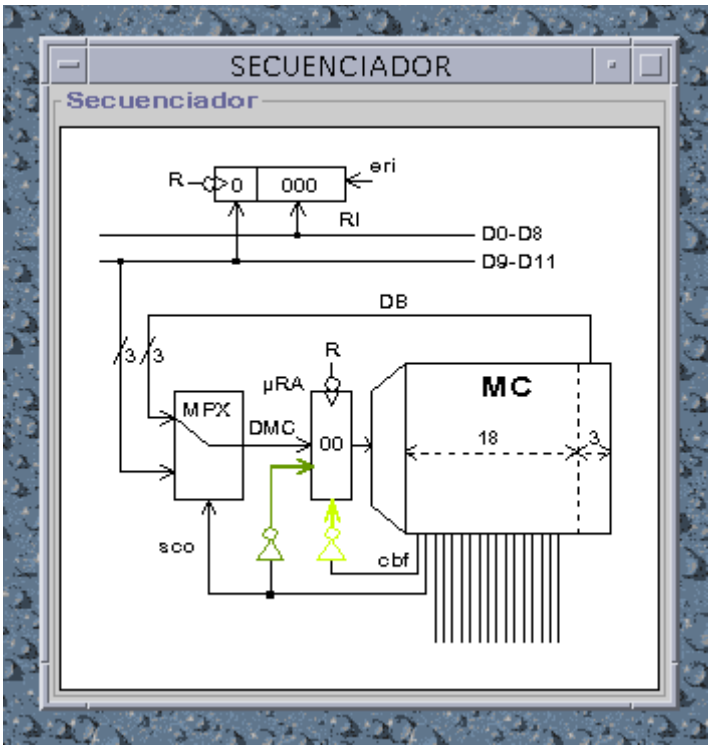


Figura 14: Secuenciador microprogramado.

Cronogramas

En esta ventana se muestran secciones de los cronogramas correspondientes a la ejecución de una instrucción. Para abrirla basta con pulsar el botón indicado en el apartado que describe la barra de herramientas de la ventana principal del simulador. El botón cambia de aspecto permitiendo saber si la ventana está abierta. Para entender a fondo el significado de los cronogramas consultar el Capítulo 5.

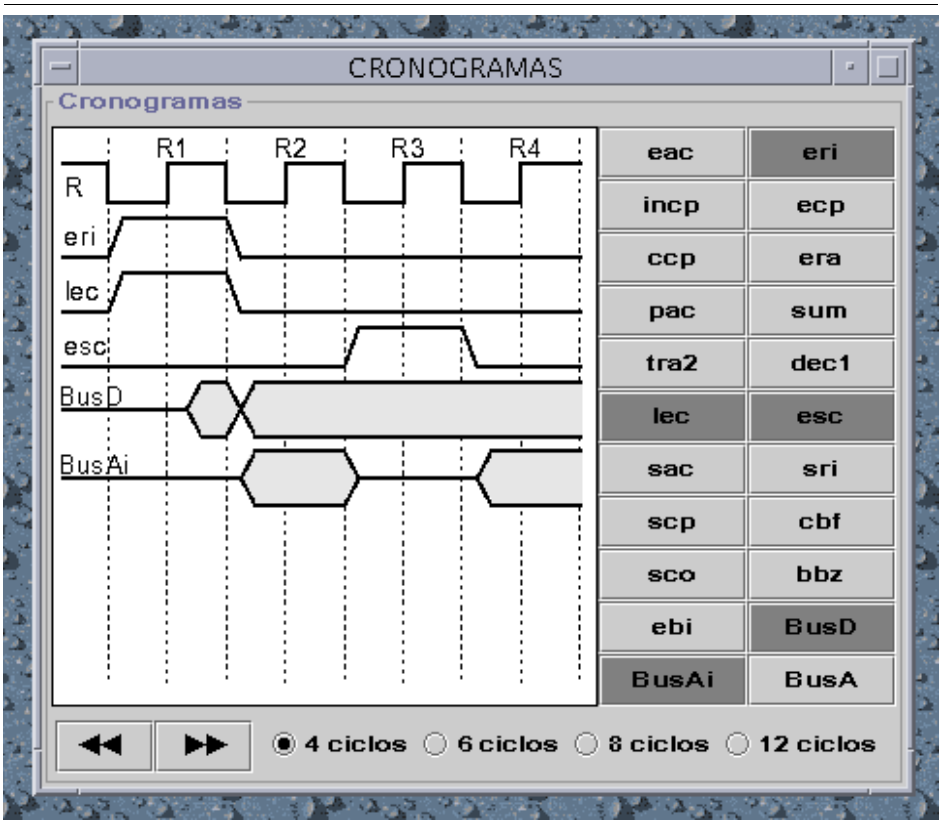


Figura 15: Ventana de Cronogramas.

La ventana se compone de tres partes. Un área de dibujo a la izquierda, un cuadro de botones a la derecha y una barra de control en la parte inferior.

El **área de dibujo** muestra el estado actual de los cronogramas de las microórdenes y buses seleccionados en ese momento. La señal de reloj aparece dibujada en la parte superior desde el comienzo de la instrucción, independientemente de la duración de ésta, de manera que sirva como referencia. Al redimensionar la ventana el área de dibujo se redibuja automáticamente al tamaño adecuado.

El cuadro de botones permite seleccionar las microórdenes y buses a dibujar. Cuando se pulsa el botón correspondiente a una señal, su nombre aparece dibujado en el área de dibujo y si ya tiene algún valor se dibuja también. El botón cambia de aspecto indicando que esa señal está siendo representada.

A medida que se seleccionan más señales para que sean representadas se va llenando el área de dibujo. Una vez que no queda espacio libre, con cada nueva señal el área se repinta acomodando todas las señales de manera proporcional. Cuando una señal representada hasta un determinado momento se deja de seleccionar, se produce el proceso inverso y se acomodan el resto de señales en el espacio disponible.

La **barra de control** se compone de un par de botones de **avance** y **retroceso**, y de un selector de número de ciclos. Los botones permiten avanzar o retroceder en los cronogramas cuando la instrucción ocupa más ciclos de reloj que los representados en ese momento. El selector de ciclos, compuesto por los cuatro botones circulares, permite decidir el número de ciclos de reloj de una instrucción que se representarán al mismo tiempo.

Cuando se modifica el número de ciclos de reloj representados el área de dibujo se repinta reflejando el cambio, tal y como se muestra en la Figura 16, en donde se han seleccionado seis ciclos de reloj.

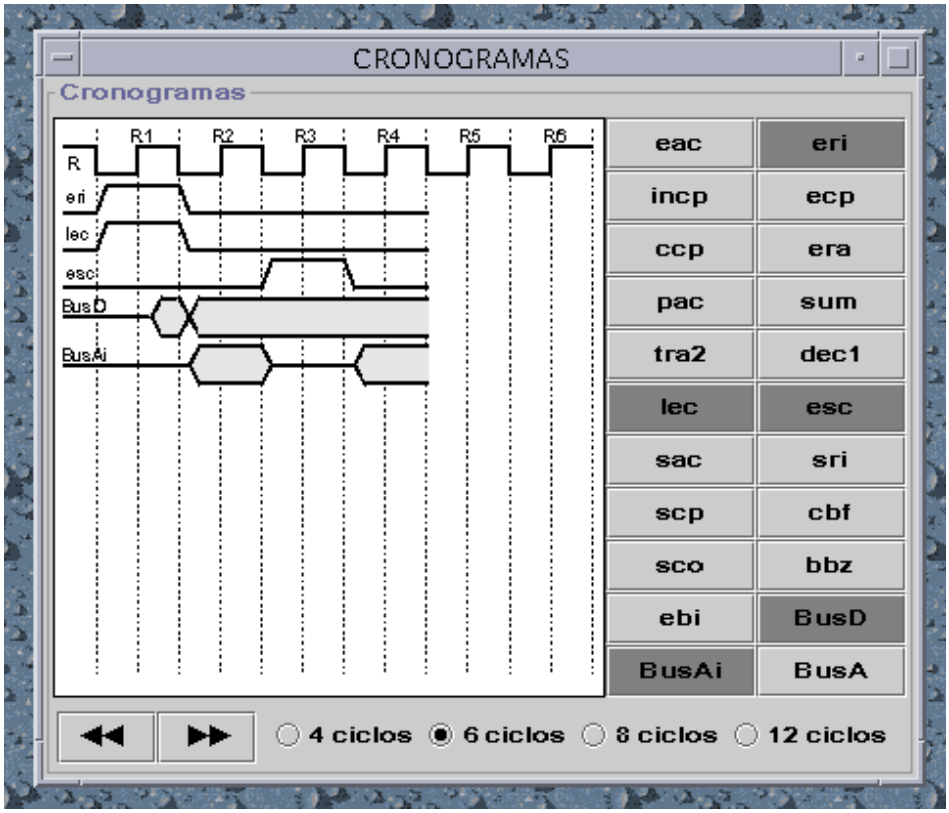


Figura 16: Cronogramas seleccionados 6 ciclos de reloj.

Nota− Los cronogramas sólo muestran la información de una instrucción en todo momento. Cuando ésta ocupe menos ciclos de reloj que los representados, el resto de pantalla quedará vacía.

Pantalla y Teclado

Pulsando en el botón correspondiente de la barra de herramientas de la ventana principal se abre la ventana mostrada en la Figura 17. El botón cambia de aspecto indicando que la ventana está abierta.

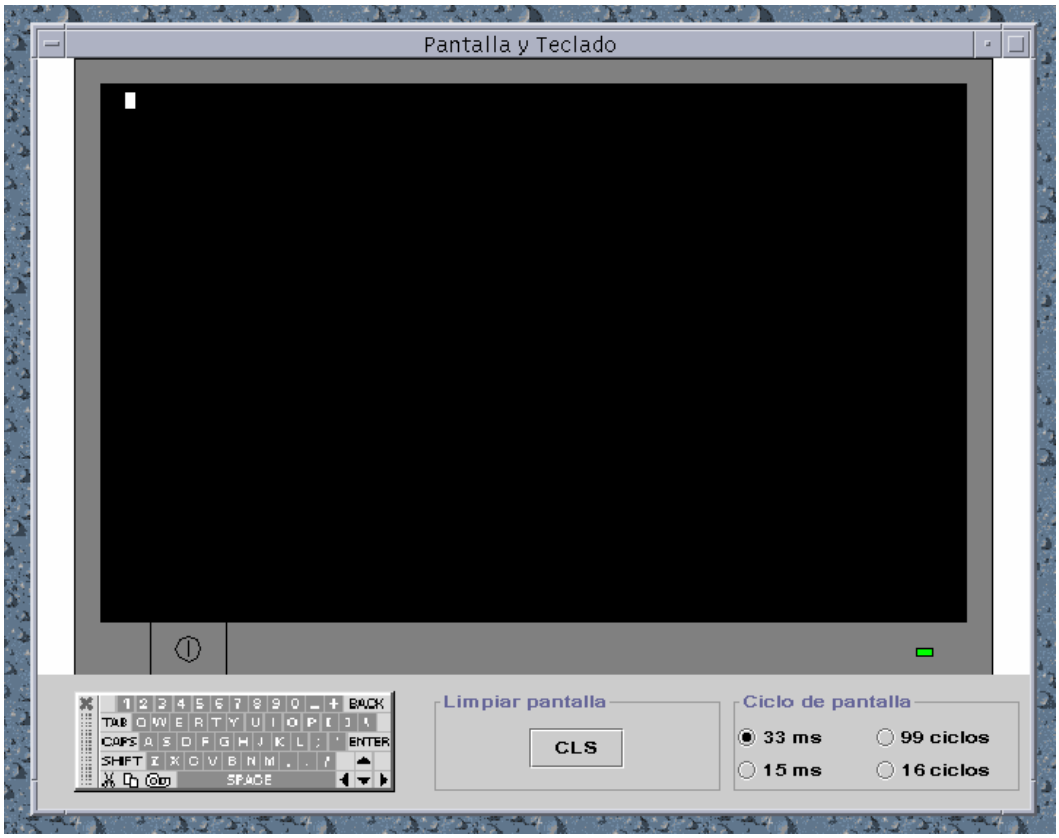


Figura 17: Ventana de Pantalla y Teclado.

La ventana se encuentra divida en dos partes. En la superior se representa la pantalla de Símplez y en la inferior hay una barra de control.

La **pantalla de Símplez** representa un monitor de texto de 20 filas por 80 columnas. El juego de caracteres que reconoce se describe en el Capítulo 6. Cuando se redimensiona la ventana, la pantalla se redibuja de manera proporcional al nuevo tamaño.

La **barra de control** se compone de los siguientes elementos:

- **Teclado**. La imagen representa el teclado de Símplez. El teclado está activo mientras el ratón permanece encima, y las teclas que se pulsen se pasan directamente al puerto de entrada de Símplez, tal y como se explica en el Capítulo 6.

- **Limpiar pantalla**. Este botón permite eliminar los caracteres actualmente en pantalla y situar el cursor en la parte superior izquierda.

- **Ciclo de pantalla**. Este grupo de botones permite modificar el tiempo que transcurre desde que un carácter se escribe en el puerto de salida hasta que por fin es mostrado por pantalla.

Editor de programas

Esta ventana contiene el editor de programas de Símplez. Para abrirla seleccionar la entrada **Editor de programas** del **menú de Herramientas**.

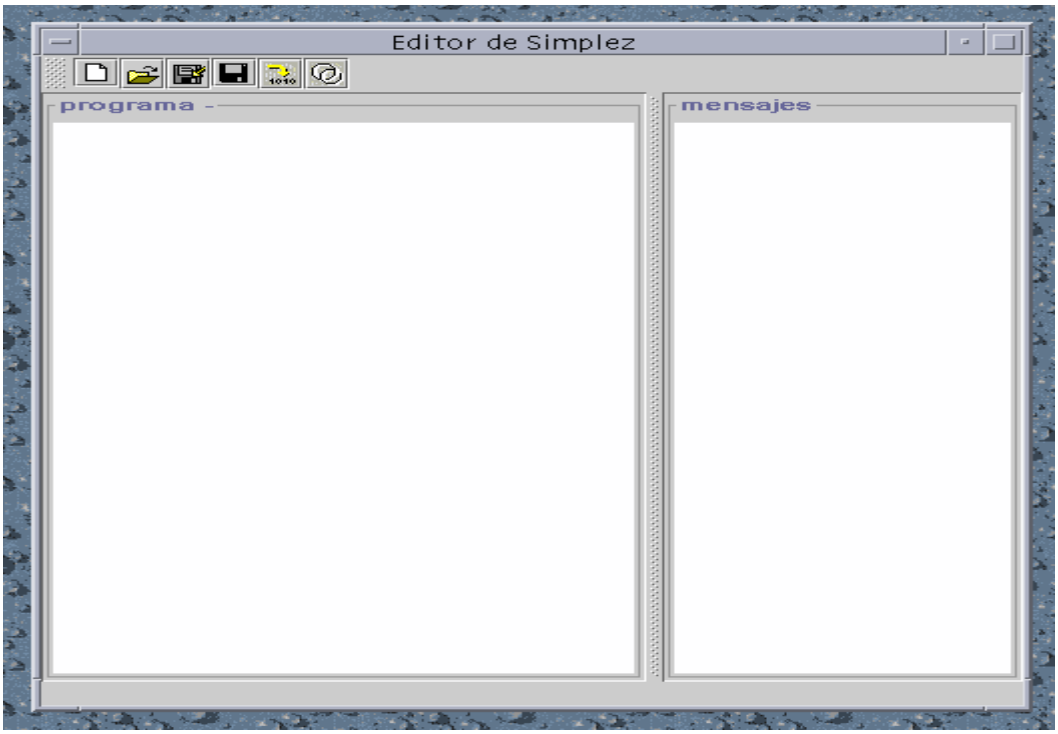


Figura 18: Ventana del Editor de programas.

La ventana se compone de una barra de control en la parte superior y del cuerpo principal de la misma debajo.

La barra de control se muestra en la Figura 19.

Figura 19: Barra de control.

Y se compone de los siguientes elementos:

 **Nuevo** programa ensamblador.

 **Abrir** programa ensamblador.

 **Guardar**.

 **Guardar como**.

 **Ensamblar**.

 **Montar**.

El área principal se divide en dos áreas de texto. Un área de edición a la izquierda, **programa**, donde se escriben los programas y otra a la derecha, **mensajes**, donde aparecerán mensajes o resultados de las acciones que se realicen.

Para una completa descripción del manejo de esta herramienta consultar el Capítulo 7.

Editor de Microprogramas

Esta ventana es accesible a través de la entrada **Editor de Microprogramas** del **menú de Herramientas** de la barra de menús.

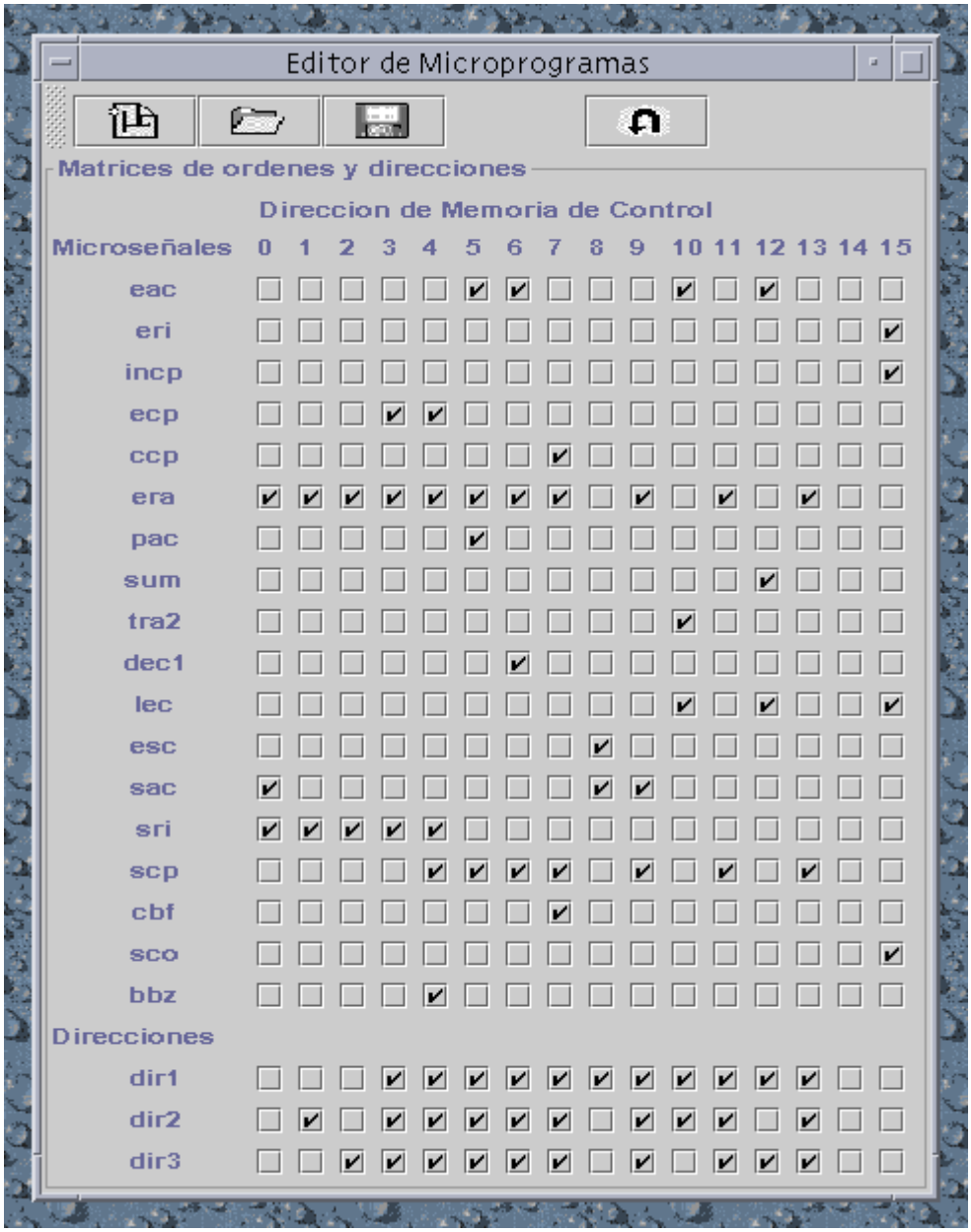


Figura 20: Editor de Microprogramas.

La ventana está formada por la barra de control mostrada en la Figura 21, situada en la parte superior, y el editor de la Memoria en la parte inferior.

Figura 21: Barra de control del Editor de Microprogramas.

Los elementos que forman la barra de control son:

Nueva memoria de Control.

Cargar una Memoria de Control desde fichero.

Salvar a fichero la Memoria de Control en edición.

Cargar la Memoria de Control incluida por defecto.

Esta herramienta permite escribir los microprogramas que forman la Memoria de Control del secuenciador de Símplez. Su utilización se describe en el Capítulo 7.

Diálogos Modales

Con determinadas operaciones o ante ciertas situaciones se abre otro tipo de ventana, los Diálogos Modales. Son pequeñas ventanas de información, llamadas modales porque bloquean la entrada de usuario al resto de ventanas del simulador. A continuación se muestran los tres tipos de diálogos modales.

- **Advertencias.** Permiten informar del resultado que provocará una determinada operación.

Figura 22: Diálogo de advertencia.

- **Errores.** Informan de una situación anómala provocada por una determinada acción. En general se acompañan de un mensaje en el panel de Mensajes de error.

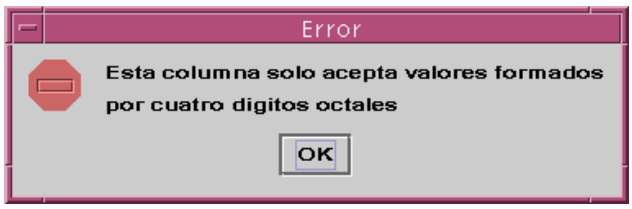


Figura 23: Diálogo de error.

- **Entrada de datos.** Permiten recoger datos del usuario.

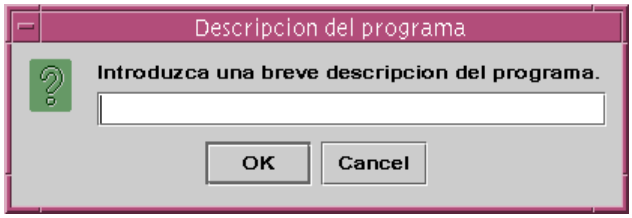


Figura 24: Diálogo de entrada de datos.

Ventanas de manejo de ficheros

Estas ventanas tienen el formato mostrado en la Figura 25. Hay dos tipos según la operación, abrir o salvar, que vayamos a realizar. Permiten navegar por los directorios del sistema.

Figura 25: Ventana de apertura de ficheros.

1.2 Flexibilidad

La interfaz gráfica del simulador se ha diseñado tratando de conseguir la mayor flexibilidad y sencillez de manejo posibles. Entre los objetivos de diseño estaban el que pudiera ser utilizada en monitores de cualquier tamaño, que el usuario pudiera acomodarla a su gusto y que fuera fácil e intuitiva de utilizar.

Teniendo en cuenta estas premisas, la aplicación se dividió, siguiendo un criterio funcional, en las ventanas descritas a lo largo de este capítulo, de manera que en todo momento se pudiera tener abierto sólo lo que se va a utilizar.

La ventana principal permite ocultar el área de monitorización, ocupando así menos espacio.

En algunos casos puede que esto no sea suficiente. Por ejemplo, si se quiere trabajar con la Ruta de Datos la ventana principal no es necesaria y sería preferible poder iconizarla. Pero en ese caso las barras de control de ejecución quedaría ocultas. Para solucionar esto, estas barras se pueden desglosar del resto de la ventana pinchando en la zona punteada de las mismas y arrastrándolas fuera.

Otras características de la interfaz gráfica son:

- Textos de ayuda al situar el cursor del ratón sobre los elementos que componen la interfaz.
- Organización de los controles en barras de herramientas y menús de fácil acceso.
- Manejo de todos los elementos con el ratón.
- Posibilidad de cambiar el tamaño de todos los elementos de la interfaz. En casos como el de las ventanas de la Ruta de Datos y los Cronogramas sus contenidos se redimensionan automáticamente.
- Iconos con apariencia diferente según el estado de la ventana a la que representan.
- Utilización de varios formatos numéricos en la ventana principal y en la ventana de la Ruta de Datos.

Capítulo 2

Introducción de un programa

En este capítulo se describe cómo introducir un programa en la memoria de Símplez. Se pueden seguir varios métodos que explicarán de manera detallada.

2.1 Editor de programas

El método más potente pasa por la utilización del *Editor de Programas* para generar un fichero de código ensamblado. El editor ha sido desarrollado por Francisco Javier Rodríguez y es accesible a través del menú de **Herramientas** seleccionando el campo "**Editor de programas**" tal y como se muestra en la Figura 26:

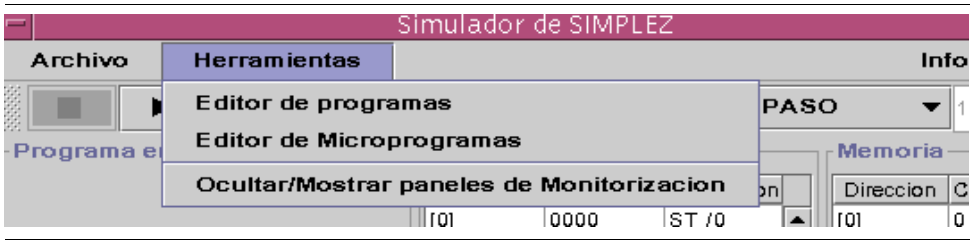


Figura 26: Menú de Herramientas.

Esta herramienta permite beneficiarse de la escritura de programas en lenguaje ensamblador. Su utilización detallada se explica en el Capítulo 7, y en el Capítulo 8 se desarrolla un ejemplo de utilización completo. Genera un fichero de extensión **.lns**, el cual se puede cargar en memoria con la opción "**Abrir fichero de código**" del menú **Archivo** o utilizando la secuencia **CTRL+A**. Se abrirá una ventana de selección de ficheros desde la cual se puede seleccionar.

2.2 Tablas de Memoria

Una segunda forma de introducir el programa en memoria es editando ésta directamente. El inconveniente de este método es que necesitamos conocer desde el principio la disposición que tendrá la memoria. Supóngase que se está introduciendo un programa de 100 instrucciones. Si por casualidad tras introducirlas todas se descubre que se necesita incluir una más en la posición 25, se tendrían que desplazar las 75 que están por debajo una posición. Por tanto es un método poco flexible, pero por contra es que se trata de una manera muy sencilla, rápida e intuitiva de introducir los datos en memoria. Es el método idóneo para programas pequeños y pruebas de microprogramación.

Es importante conocer los formatos que aceptan las tablas de memoria. Si el formato del número introducido es incorrecto, aparecerá un diálogo modal avisando de este hecho y se recuperará el valor que originariamente contenía dicha posición. El formato de las tres columnas que aceptan valores es.

- La columna **Contenido** de la tabla *Programa* acepta cuatro dígitos en formato octal.

- La columna **Instrucción** de la tabla *Programa* acepta el código nemónico de la instrucción. Este código nemónico tiene la estructura:

  <Código Operación> [/<Código Dirección>]

  Los *códigos de instrucción* se pueden introducir tanto en mayúsculas como en minúsculas. Estos códigos son:

ST	LD	ADD	BR
BZ	CLR	DEC	HALT

  El *código de dirección* es un número en formato decimal entre 0 y 511, precedido del carácter "/". No todas las instrucciones requieren el código de dirección. En concreto, si es introducido en el caso de CLR, DEC y HALT simplemente se descarta.

- La columna **Contenido** de la tabla *Memoria* admite números en formato decimal comprendidos entre 0 y 4095.

Como ejemplo se introducirá el Programa 1, tomado de la página 59 del libro del curso [Fernández,98], que suma el contenido de las posiciones de memoria 10 y 11 y lo almacena en la posición 12.

```
 Dir. MP   Cont.      Cont.    Comentarios
  (dec)   (octal)  (nemónico)
--------  -------  ----------- ----------------------
   [0]      1012   LD    /10   ;Carga el primer número
   [1]      2013   ADD   /11   ; le suma el segundo
   [2]      0014   ST    /12   ; almacena el resultado
   [3]      7000   HALT
```

Programa 1: Suma el contenido de dos posiciones de memoria y almacena el resultado en otra.

Para editar una determinada posición de memoria basta con posicionar el ratón sobre la misma y pinchar. Tal y como se muestra en la Figura 27, la celda cambia de aspecto y entra en modo de edición. Únicamente hay que tener en cuenta el formato de la columna que se ha escogido para introducir el valor. Una vez que se haya terminado de introducir el valor, si se pulsa *Enter* se pasa a editar la siguiente posición de memoria. Para salir del modo edición basta con pulsar el ratón fuera de la tabla.

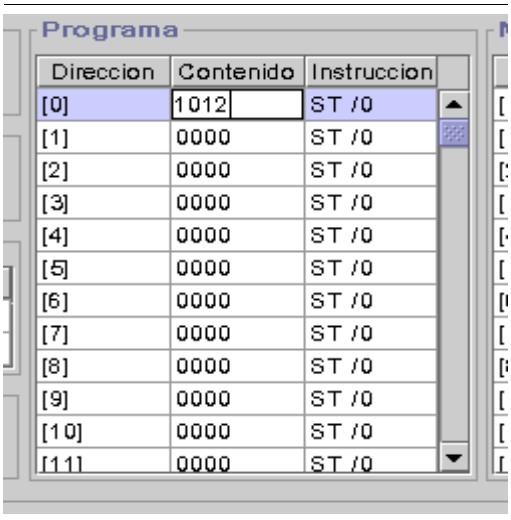


Figura 27: Edición de la Tabla de Programa

Las tablas *Programa* y *Memoria* no son independientes entre sí. Cuando se acaba de editar una celda se actualizan automáticamente las celdas homólogas de las columnas restantes de ambas tablas. Se puede estar en modo edición en ambas, editando la misma o diferentes celdas.

Nota− Si en ambas tablas se está editando la misma posición de memoria, el valor final será el de la última celda que salga del modo de edición. Si en ésta se comete un error permanecerá el valor introducido con la otra.

Supóngase que para introducir la primera instrucción se utiliza el formato octal. Para ello se pulsaría en la primera celda de la columna **Contenido** de la tabla de *Programa*, como se mostraba en la Figura 27. Una vez introducido el valor 1012 se puede pulsar con el ratón fuera para desactivar la edición, o pulsar Enter con lo que se pasaría a editar la siguiente posición de memoria. Para introducir el resto de instrucciones se puede utilizar el código nemónico. Para lo cual se pulsa con el ratón en la segunda celda de la columna **Instrucción** de la tabla de *Programa*, se introduce la cadena "add /11" y se pulsa *Enter*.

Nota− Las instrucciones se pueden introducir tanto en mayúsculas como en minúsculas.

Por último, para introducir los datos a sumar en las posiciones [10] y [11] se puede hacer a través de la tabla de Memoria utilizando el formato decimal. Suponiendo que los valores a sumar son el 3 y el 8, se pulsarían respectivamente las posiciones de memoria [10] y [11] de la columna "Contenido" de la Tabla de Memoria tal y como muestra la Figura 28.

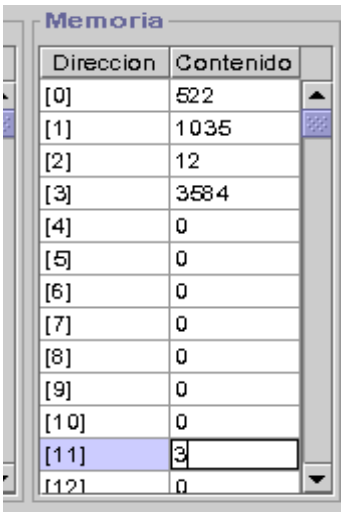


Figura 28: Edición de la Tabla de Memoria.

Como se habrá podido observar al introducir valores en una celda determinada las celdas homólogas de las columnas de ambas tablas cambian también su contenido adecuándolo al nuevo valor introducido, tal y como se indicó con anterioridad.

El contenido final de la memoria será el siguiente:

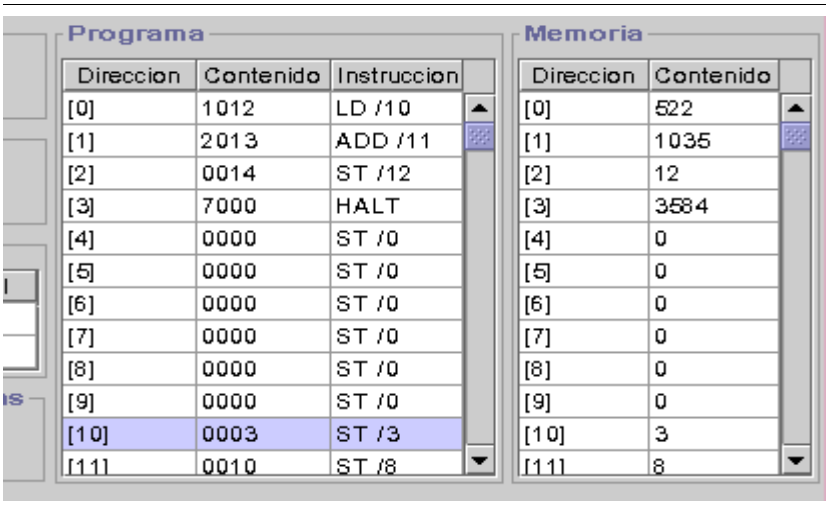


Figura 29: Programa cargado en la Memoria de Símplez.

Las posiciones [10] y [11] de la columna **Instrucción** de la tabla de Programa contienen los valores "ST /3" y "ST/8", que son los códigos nemónicos que se corresponderían con los valores que contienen dichas posiciones. Esto es debido a que esta columna siempre interpreta el contenido de la memoria como instrucciones de Símplez. Por tanto hay que tener claro qué zonas de memoria contienen datos y cuáles código. De hecho una posición de memoria puede ser tanto uno como otro según se acceda a ella. Si el sistema accede a ella buscando la siguiente instrucción a ejecutar no hay duda que se trata de código, y si en cambio lo hace porque su contenido se va a sumar con otro operando se está hablando de datos.

Nota− La edición sólo es posible si no se está en ejecución.

2.3 Volcado de Memoria

Otra posibilidad para cargar un programa es cargar un volcado de memoria desde un fichero con extensión **.vms** utilizando la entrada "**Cargar Memoria desde Fichero**" del menú de **Archivo**, o pulsar la secuencia **CONTROL C**.

Un volcado de memoria es el resultado de salvar a fichero el contenido de la memoria byte a byte. Permite guardar el estado de la memoria en cualquier momento y recuperarlo posteriormente.

Para crear un volcado de memoria se puede utilizar la entrada "**Volcar Memoria a fichero**" del menú de Archivo, o pulsar la secuencia de teclas **CONTROL V**. Se abrirá una ventana de manejo de ficheros.

2.4 Nuevo programa

En cualquier momento se puede reiniciar la memoria y hacer que todas sus palabras contengan un cero, seleccionando la entrada "**Nuevo contenido de Memoria**" del menú de **Archivo**, o pulsar la secuencia de teclas **CONTROL N**.

Capítulo 3

Ejecución de un programa

Los programas escritos para Símplez no pueden ejecutarse directamente sobre ninguna computadora, dado que el repertorio y formato de sus instrucciones, totalmente dependientes del modelo estructural y funcional subyacente, y el tipo de datos manejados seguramente no coincida con el de la máquina en cuestión. Y dado que aún no se ha implementado la versión hardware de Símplez, es necesaria la utilización de un simulador para su ejecución. La gran ventaja de un simulador es el control que permite tener sobre la ejecución de un programa, pudiendo llegar a hacerse cosas imposibles con la máquina real o sólo posibles con un debugger de código máquina. Como por ejemplo la modificación de los registros del sistema a mano, cambiar la posición de ejecución o poder deshacer los efectos de una instrucción ya ejecutada.

Este simulador en concreto nos permite además ejecutar programas escritos para Símplez sobre cualquier máquina para la que se haya desarrollado una máquina virtual Java.

Se pueden distinguir dos tipos de simulación; la simulación de la ejecución de las instrucciones y la simulación gráfica de la Ruta de Datos y los cronogramas.

Los principales elementos para controlar la ejecución de un programa en el simulador son las barras de herramientas y las áreas de información y monitorización. Con las barras de herramientas se puede en todo momento decidir cuándo arrancar o parar la ejecución, qué debe estar visible en el simulador o qué modo de ejecución utilizar. Las áreas de información y monitorización permiten conocer cuál es el estado actual de la máquina y saber qué está pasando en todo momento.

El control de la simulación gráfica se realiza a través del panel de controles situado en la ventana de la Ruta de Datos.

3.1 Controles

El simulador tiene dos barras de herramientas en la ventana principal y un área de control en la ventana de la Ruta de Datos, que permiten controlar la ejecución de programas.

La primera barra de herramientas, Figura 30, se encuentra inicialmente a la izquierda de la ventana principal, contiene los botones de **arranque**, **parada** y **deshacer**.

Figura 30: Barra de Herramientas uno.

Según los botones que estén habilitados en un momento dado se puede saber si la simulación está en marcha, botón de **parada** habilitado, o no, botón de **arranque** habilitado. Inicialmente el único botón habilitado es el de **arranque.**

Para comenzar la ejecución de un programa basta con pulsar el botón de **arranque**. Las tablas de memoria y de registros no pueden estar en edición en ese instante. Si una de ellas se encuentra en dicho estado el simulador mostrará un mensaje indicando que se termine la edición antes de comenzar a ejecutar.

Se puede interrumpir la ejecución de un programa en cualquier momento pulsando el botón de parada. La instrucción en curso se interrumpe y la máquina pasa al estado que tenía antes del anterior al comienzo de la misma.

Se puede deshacer la ejecución de hasta un total de 10 instrucciones, independientemente del modo de ejecución utilizado. Sin embargo cuando se cargue un nuevo programa en memoria, la información sobre las instrucciones ya ejecutadas pertenecientes al programa actualmente en memoria se perderá y no será posible deshacerlas. Ya que se ha producido un reseteo del hardware de la máquina. En la página 45 se explica qué significa el reseteo. Cuando se edita el contenido del Contador de Programas, también se resetea y no es posible deshacer la ejecución de las últimas instrucciones. En ambas situaciones el botón de deshacer pasa a estar deshabilitado.

Con la posibilidad de deshacer la ejecución de instrucción se pueden comparar los resultados de una instrucción con diferentes microprogramas y la implementación cableada. Esto permite entender perfectamente los efectos de una decisión de diseño.

Nota− Los cambios realizados tras la ejecución de instrucciones en los registros y la memoria mediante edición de las tablas se perderán cuando se deshaga la ejecución de las mismas, debido a que los estados de la máquina originales no los incluían.

La segunda barra de herramientas, Figura 31, permite decidir qué ventanas tener abiertas en todo momento y el modo de ejecución a utilizar.

Figura 31: Barra herramientas dos.

Cuando la ventana de la Ruta de Datos o la de Cronogramas están abiertas, hay simulación gráfica y la velocidad de ésta se controla desde el área de control de la Ruta de Datos.

Desde el panel de control de la ventana de la Ruta de Datos, Figura 32, podemos controlar la simulación gráfica tanto de la Ruta de Datos como de los cronogramas.

Figura 32: Área de control de la Ruta de Datos.

Tal y como se explico en el Capítulo 1, la primera barra de desplazamiento nos permite modificar la **Velocidad de ejecución**. No se trata de modificar la frecuencia de reloj de Símplez, ni la velocidad de ejecución de un programa en el simulador, sino la velocidad a la que se muestra gráficamente la ejecución de una instrucción en la Ruta de Datos y en los Cronogramas. De hecho, si ambas ventanas están cerradas la posición de esta barra de desplazamiento no tiene ningún efecto sobre la simulación.

La barra de **Posición de ejecución** y el **botón de retroceso** situado en la parte inferior permiten desplazar el instante actual dentro de la simulación gráfica de una instrucción a uno ya ejecutado. Por tanto, el rango de tiempo que cubre va desde el primer flanco de bajada de reloj correspondiente a la instrucción hasta el instante de tiempo en que se mueve la posición del indicador. Cuando el secuenciador seleccionado es el microprogramado puede ocurrir que la ejecución de una instrucción implique muchos ciclos de reloj. En estos casos llegado a cierto número de ciclos, por motivos de consumo de memoria, a medida que avanza la simulación de la instrucción se van eliminando ciclos de reloj del comienzo de la misma. El botón de **retroceso** desplaza el instante de ejecución hasta el primero almacenado, que normalmente coincide con el comienzo de la instrucción excepto en el caso indicado.

Al mover el indicador a una posición determinada, éste se va desplazando desde ella hasta que alcanza el final de la barra y en ese instante se continua con la simulación de la instrucción en el punto en el que se dejó.

Una vez terminada la ejecución de una instrucción es posible mover el indicador hacia atrás y revisar lo qué ha ido pasando, evitando tener que deshacer la ejecución de la instrucción y tener que volver a ejecutarla. Esto es de gran interés en situaciones en que por una microprogramación incorrecta diferentes ejecuciones de la misma instrucción puedan producir resultados diferentes y se quiera revisar cada uno de ellos en detalle.

La barra de Posición de ejecución y el botón de retroceso están activos cuando hay información sobre la instrucción ejecutada. Cuando se pulsa el botón de **Detener la ejecución** durante la simulación de una instrucción, la información gráfica de la misma se pierde y tanto la barra como el botón se deshabilitan. Cuando se deshace la ejecución de una instrucción su información gráfica también se pierde y se deshabilitan. Cuando se cambie el tipo de secuenciador se deshabilitarán, pero el motivo es prevenir confusiones tal y como se explica un poco más adelante.

En ocasiones puede que se perciba un parpadeo en la Ruta de Datos o en los Cronogramas al modificar la posición de ejecución. Esto es debido a que a medida que se mueve el indicador de posición se van generando eventos de entorno gráfico y algunos de ellos son atendidos antes de terminar de moverlo.

El botón de **pausa** detiene temporalmente la simulación gráfica. Este botón es muy útil cuando se quiere modificar la **Posición de ejecución** con la barra de desplazamiento, ya que permite detener la misma hasta que se haya situado en el lugar deseado, o incluso ir repasando detalles moviendo el indicador de la barra. Si antes de

pulsar el botón de comienzo de ejecución se pulsa el de pausa, la simulación en la Ruta de Datos queda congelada justo antes del primer flanco de bajada del reloj.

Nota− El botón de pausa tiene efecto aunque la ventana de la Ruta de Datos no esté abierta. Una instrucción puede parecer que se ha colgado y que no termina nunca de ejecutarse debido a que el botón esté pulsado. Para solucionarlo basta con detener la ejecución de la misma y volver a pulsar el de comienzo. El botón de parada deshabilita la pausa.

Por último, el botón de **Comportamiento ante cbf** le indica al simulador qué debe hacer cuando el secuenciador microprogramado genera la microorden **cbf**. Si no está pulsado, el simulador no interrumpirá la ejecución cuando **cbf** esté activa. Este botón no tiene efecto con el secuenciador cableado.

Nota− Este cuadro marca el comportamiento ante cbf incluso cuando la Ventana de Ruta de Datos está cerrada. Una instrucción puede parecer que se ha colgado y que no termina nunca de ejecutarse debido a que el cuadro no está marcado.

Los dos botones restantes no influyen en la simulación.

Modos de ejecución

Un *modo de ejecución* es la manera que tiene el simulador de estructurar la ejecución de las instrucciones que componen un programa. El simulador utiliza tres modos de ejecución, seleccionables mediante el menú desplegable, mostrado en la Figura 33, situado en una de las barras de herramientas.

Figura 33: Selector de Modo de Ejecución.

El modo de ejecución únicamente se puede modificar con la simulación parada, por lo que una vez comienza la ejecución el selector se deshabilita.

Paso a paso

Este modo de ejecución permite ir ejecutando una a una las instrucciones del programa. Cuando la instrucción en curso finaliza, el simulador detiene la ejecución. Esta manera de simular se suele denominar depuración.

Continuo

Éste es el modo de ejecución clásico. El simulador ejecuta una instrucción tras otra hasta que encuentra una instrucción HALT, tras la cual se detiene.

Continuo hasta

En este modo de ejecución el simulador ejecuta tantas instrucciones seguidas como indique el campo numérico situado a la izquierda del selector, hecho lo cual se detiene. En el caso de encontrar una instrucción HALT la ejecución de detiene igualmente antes de completar dicho número.

El campo numérico sólo está habilitado cuando el modo de ejecución seleccionado sea Continuo hasta. El número máximo de instrucciones ejecutables en este modo son 99.

3.2 Área de información

Desde este área de la ventana principal del simulador, mostrada en la Figura 34, se puede visualizar tanto información de la ejecución como del estado de la máquina. Desde aquí también es posible modificar parte de ese estado.

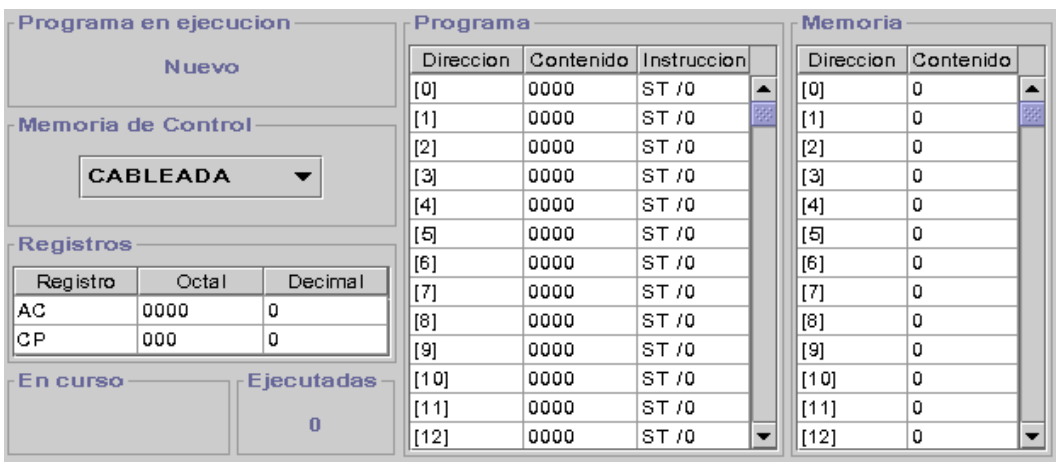


Figura 34: Área de información.

El área **Mensajes de Símplez** del área de **Monitorización del Sistema** también es de gran utilidad cuando está abierta la ventana de la Ruta de Datos o la de Cronogramas para entender como evoluciona la máquina. En ella se van mostrando mensajes que indican cada cambio que se produce en la máquina.

Información de ejecución

El campo **Programa en ejecución** indica el nombre del programa cargado en memoria. Su valor se actualiza al salvar a disco, momento en que se solicita un nombre para el programa que se va a guardar, y cuando se carga un programa desde un fichero.

El campo **En curso** indica la instrucción en ejecución. Cuando finaliza la ejecución mantiene la información de la última ejecutada. Cuando se interrumpe o se deshace la ejecución de una instrucción el campo pasa a estar vacío.

El campo **Ejecutadas** muestra el número de instrucciones ejecutadas desde el último reseteo de la máquina. Al deshacer la ejecución de una instrucción su valor se decrementa.

Información de estado

El campo Memoria de Control permite seleccionar la memoria de control que utilizará el secuenciador. Cuando se seleccione la **cableada** realmente se estará refiriendo a los circuitos combinaciones que forman el secuenciador cableado. La Memoria de Control **microprogramada** es la implementada por defecto en el sistema, cuyo contenido se muestra en el Capítulo 9. A medida que se carguen Memorias de Control desde fichero, utilizando la entrada del menú de **Archivo** o pulsando **CONTROL+M**, se irán añadiendo al desplegable. Cuando se carga una memoria de control con el nombre CABLEADA o PROGRAMADA al nombre se le añade un sufijo para distinguirlas de las memorias por defecto. Cuando se carga una con el mismo nombre que una ya cargada se sustituye esta última.

Al seleccionar un secuenciador aparece un mensaje en el área de Monitorización. Esto permite comprobar con qué Memoria de Control se han estado ejecutando las instrucciones.

Cuando se cambie el tipo de secuenciador, los controles de la Ruta de Datos que permiten modificar la posición de la simulación gráfica se deshabilitan. De lo contrario, cambiando la posición de ejecución no sólo se vería la ejecución generada por otro secuenciador sino que incluso se podría estar mostrando la información con una Ruta de Datos incorrecta. La manera de volver a visualizar la instrucción con un determinado secuenciador es volver a seleccionarlo, deshacer la ejecución de la instrucción y volver a ejecutarla.

La tabla de **Registros** muestra el contenido del Acumulador y del Contador de Programa. Sus valores son editables pinchando sobre la columna correspondiente al formato en que se quiere introducir el nuevo valor. La edición del Contador de Programa hace que se resetee el hardware de la máquina.

El valor de estos registros está gobernado por flanco de bajada de reloj, por lo que en ciertas instrucciones, como DEC para AC y BZ para CP, su valor no se modifica hasta el flanco que marca el final de la instrucción y dado que éste se considera incluido en la siguiente instrucción, será durante la ejecución de esta última que su valor cambiará. Esto puede producir alguna situación confusa, dado que durante la ejecución de la siguiente instrucción el valor de dicho registro puede cambiar también, lo que se podría interpretarse como que el simulador se ha saltado un valor. La forma de estar seguros es abrir la Ruta de Datos y comprobar la evolución del contenido. Ésta es una limitación del simulador que no se puede evitar debido a la microprogramación. Las instrucciones no pueden incluir el último flanco debido que en él ya se están leyendo las microórdenes de la siguiente instrucción, pero a priori no se sabe con qué secuenciador se va a ejecutar.

Hay dos tablas que muestran el contenido de la **memoria**. La tabla de *Programa*, como su nombre indica, está asociada a la ejecución del programa y mostrará en todo momento la zona de la memoria que contiene la instrucción en curso. Por otro lado la tabla de *Memoria* permite situarse en una zona de la memoria determinada y tenerla siempre a la vista, de manera que se pueda observar qué cambios se producen en ella. Está más relacionada con los datos que maneja el programa. El formato numérico de ambas responde también a esos criterios.

Tal y como se ha visto en el capítulo anterior se puede editar el contenido de la memoria con solo pinchar en la posición de memoria deseada.

Nota− Las tablas muestran el resultado de la ejecución, por lo que no van síncronas con la Ruta de Datos ni con los cronogramas. La simulación de la ejecución siempre va adelantada cuatro de ciclos de reloj respecto a la simulación gráfica. Los datos de esta última se muestran con un cierto retraso respecto al valor que realmente tienen en un momento dado.

Con instrucciones de hasta cuatro ciclos de reloj se observará que las tablas muestran el valor final y en cambio en la Ruta de Datos se están mostrando valores temporales. Con instrucciones de más de cuatro ciclos los dos mostrarán valores temporales. Algo similar pasa con los valores mostrados con la pantalla y el teclado y los valores de los puertos mostrados en la Ruta de Datos. La pantalla y el teclado muestran los valores de manera síncrona a la ejecución.

Todos los elementos del área de información de estado se deshabilitan durante la ejecución y su contenido no es editable.

3.3 Reseteo del Hardware

En todo momento el hardware de Símplez tiene un estado. Cuando se cargue un nuevo programa en memoria o se cambie el contenido del Contador de Programa, el estado actual de los buses, de los puertos de entrada y salida, de la memoria, y en general de toda la máquina hasta ese momento deja de tener valor, por lo que se resetea. Así mismo, se inhabilita el botón que permite deshacer la ejecución de la última instrucción y se vacían los campos de instrucción en curso y número de instrucciones ejecutadas.

Cuando se modifique el valor del Contador de Programa, se guardará el contenido actual de la Memoria Principal y del Acumulador.

Tras la ejecución de una instrucción HALT es necesario resetear el hardware de la máquina si se quiere volver a ejecutar el programa, ya que aunque el Contador de Programa está a cero, las microórdenes de inhibición de la ejecución siguen activas. Para resetear basta con pulsar el ratón encima del Contador de Programa.

Capítulo 4

Ruta de Datos

Según la definición dada en [Fernández,98], *"Una **ruta de datos** es una estructura constituida por un conjunto de elementos (buses, registros y circuitos combinacionales) interconectados, cuyas funciones son transferir, memorizar y procesar las informaciones (instrucciones, direcciones y operandos) procedentes de (o con destino a) la MP y los dispositivos periféricos."*

La Ruta de Datos de Símplez se muestra en la Figura 35.

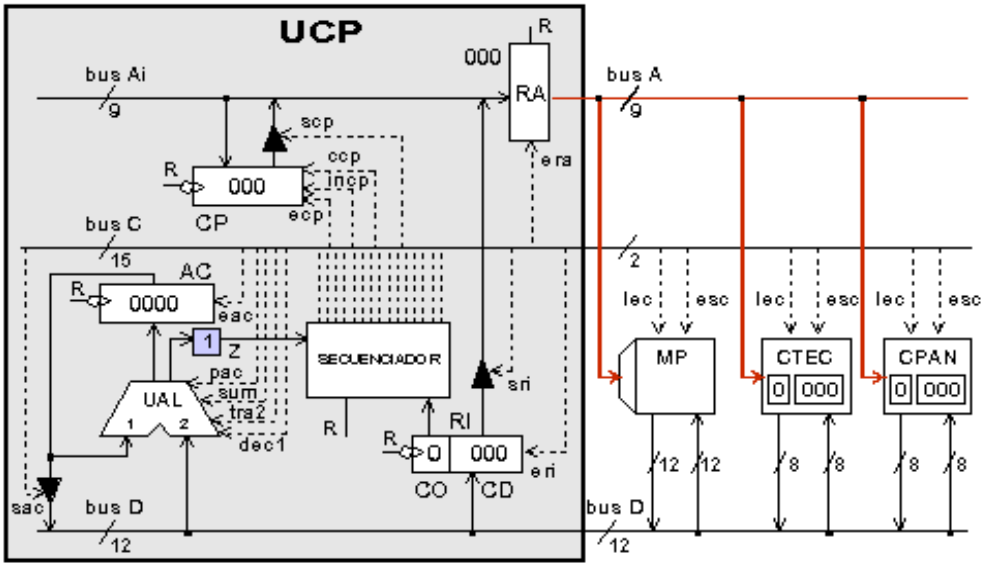


Figura 35: Ruta de Datos de Símplez.

La Ruta de Datos es un reflejo del modelo estructural de la máquina. En el caso de Símplez queda patente que se trata de una máquina basada en el esquema de von Neumann.

Von Neumman hablaba de una serie de elementos, órganos, fundamentales, que debían estar presentes en una máquina computadora de propósito general, y que se encuentran en la Ruta de Datos de Símplez. Estos elementos son la "**UAL**" (unidad aritmético−lógica), la "**MP**" (memoria principal), la "**UC**" (unidad de control) y las "**UE/S**" (unidades de entrada y salida). En la Ruta de Datos de Símplez, el Secuenciador se identifica con la UC, y el CTEC y el CPAN con las UE/S. La UC y la UAL componen la **Unidad Central de Proceso**, que en la Ruta de Datos se corresponde con el cuadrado de color con la etiqueta **UCP**.

A continuación se analizan uno a uno los diferentes elementos que componen la Ruta de Datos de Símplez y los posibles aspectos que pueden tener en el simulador según el estado en que se encuentren.

4.1 Secuenciador

"Un sistema que genera en cada momento las microórdenes oportunas (señales binarias) para que lleven a cabo los procesos en la ruta de datos". El secuenciador de Símplez se muestra en la Figura 36.

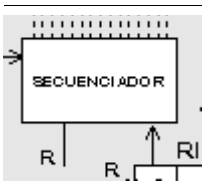


Figura 36:
Secuenciador.

Tal y como se describe en [Fernández,98], para que el secuenciador pueda gobernar al resto de elementos de la Ruta de Datos y que las instrucciones se ejecuten de acuerdo al modelo funcional de la máquina, ha de repetir cíclicamente los siguientes pasos:

1. Extrae una instrucción almacenada en una palabra de la Memoria Principal.

2. Interpreta esa instrucción: la descodifica y genera las microórdenes para poner en funcionamiento al resto de las unidades.

3. Genera la dirección de la Memoria Principal en donde se encuentra la siguiente instrucción a ejecutar. Esta etapa, tal y como se explicará en detalle en el Capítulo 5, realmente se encuentra integrada en las dos etapas anteriores.

El secuenciador es el único elemento que no cambia de aspecto en la ventana de la Ruta de Datos, dado que tiene su propia ventana asociada donde podemos ver en qué estado está.

4.2 Microórdenes

Las microórdenes son señales binarias que conectan al secuenciador con los diferentes elementos que componen la Ruta de Datos. Las lista de microórdenes se muestra en la Tabla 2.

Nombre	Efecto
eac	Entrada en el acumulador desde la salida de la UAL
eri	Entrada en el registro de instrucciones desde el bus D
incp	Incremento del contador de programa
ecp	Entrada en el contador de programa desde el bus Ai
ccp	Puesta a cero del contador de programa
era	Entrada en el registro de dirección desde el bus Ai
pac	Puesta a cero de la salida de la UAL
sum	Suma en la UAL de las entradas 1 y 2, dejando el resultado en la salida
tra2	Transferencia de la entrada 2 de la UAL a la salida
dec1	Decremento de la entrada 1 de la UAL, dejando el resultado en la salida
lec	Lectura en la MP o en el puerto de entrada
esc	Escritura en la MP o en el puerto de salida
sac	Salida del acumulador al bus D
sri	Salida de los nueve bits de menor peso del registro de instrucciones al bus Ai
scp	Salida del contador de programa al bus Ai
cbf	Inhibición del funcionamiento
sco	Salida del código de operación (UC microprogramada)
bbz	Para la instrucción BZ (UC microprogramada)
ebi	Entrada en el biestable I (UC cableada)

Tabla 2: Microórdenes de Símplez.

Cuando una microorden está activa aparece resaltada en color, tal y como se muestra en la Figura 37, el resto del tiempo se pintarán en negro.

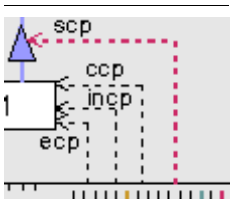


Figura 37: Microorden activa.

Se considera que una microorden está activa cuando su valor es “1”, y las transiciones de un valor a otro no se consideran dentro de este estado. Cada microorden tiene asignado un color diferente.

4.3 Unidad Aritmético−Lógica

“... la unidad aritmético y lógica, o UAL, es un subsistema que puede tomar dos operandos (o sólo uno, como en el caso de “NOT") y generar el resultado correspondiente a la operación que se le indique, de entre un conjunto de operaciones previstas en su diseño.” La UAL de Símplez se muestra en la Figura 38.

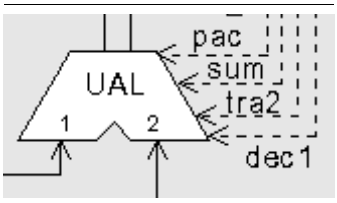


Figura 38: UAL de Símplez.

La UAL tiene dos entradas, cada una de ellas de 12 bits, y genera dos salidas, una de 12 bits al acumulador, que dependerá del tipo de operación seleccionada, y una salida binaria al biestable Z, que depende de si el resultado de la última operación fue cero o no. Las operaciones posibles son:

- **Poner a cero** los doce bits de la salida.
- **Sumar** los dos operandos de las entradas.
- **Transferir** a la salida el operando presente en la entrada 2.
- **Decrementar** en una unidad el operando presente en la entrada 1.

Cada una de estas operaciones se selecciona activando las microórdenes **pac**, **sum**, **tra2** y **dec1** respectivamente. Cuando más de una microorden esté activa sólo se ejecutará la operación correspondiente a la microorden con más prioridad entre las activas. El orden de prioridad de las microórdenes es el siguiente: **pac**, **sum**, **tra2**, **dec1.**

La UAL se considerará activa cuando alguna de sus microórdenes de control esté activa. El aspecto de la UAL cuando esto ocurra es el de la Figura 39.

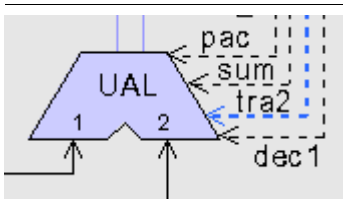


Figura 39: UAL activada por tra2.

4.4 Biestables

El biestable es un circuito con “memoria”, de dos estados posibles según el valor que contenga. En la Ruta de Datos los biestables se representan como en la Figura 40 cuando contengan un “0”.

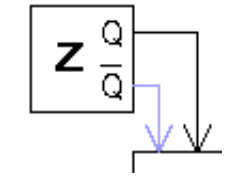


Figura 40: Biestable con "0".

Y cuando contengan un “1” se pintarán como en la Figura 41.

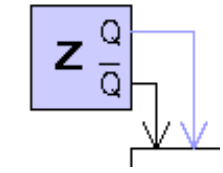


Figura 41: Biestable con "1".

El biestable Z es asíncrono, es decir no tiene entrada de reloj. Por tanto en todo momento seguirá la evolución de sus entradas.

En el caso del secuenciador cableado hay tres biestables más. Dichos biestables son síncronos, es decir sus variaciones de estado están marcadas por la señal de reloj. En concreto los biestables F y T están sincronizados por flanco de subida y el biestable I por flanco de bajada.

Registros

"Un registro no es más que un conjunto de biestables con una entrada de control común." " Al utilizar un registro haremos abstracción del tipo de sus biestables componentes y de los registros lógicos asociados. Sólo nos interesa saber que es un registro de "carga en paralelo", que mantiene un estado de n bits a su salida mientras no se den ciertas condiciones en sus entradas, y que cuando estas condiciones se dan su estado queda sustituido por los valores de las n entradas. Sí será importante indicar si es de tipo cerrojo, o sincronizado por flancos de subida, o sincronizado por flancos de bajada."

En la Ruta de Datos de Símplez hay dos tipos de registros, de cerrojo positivo y sincronizados por flanco de bajada. En la Figura 42 se muestra un registro sincronizado por flanco de bajada.

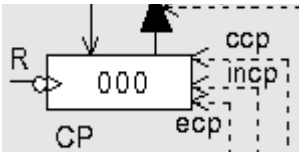


Figura 42: Registro sincronizado por flanco de bajada.

En este tipo de registro los cambios de su contenido tienen lugar únicamente en el instante de transición de "1" a "0" de la señal de reloj y solamente si en ese instante la microorden correspondiente está activa.

Es interesante observar como en el caso de estos registros, sus valores permanecen constantes durante todo un ciclo de reloj a pesar de estar activa una microorden. Esto se debe a que las microórdenes son generadas por el secuenciador, y éste también está sincronizado por el reloj. De manera que cuando una microorden pasa a estar activa, lo hace inmediatamente después de un flanco de bajada pero con cierto retraso, por lo que no tomará efecto hasta el siguiente flanco de reloj. De igual manera la microorden y el dato de entrada se quitan en el mismo flanco de bajada, pero como ambas acciones son dependientes del secuenciador irán con cierto retraso respecto al flanco y al registro le habrá dado tiempo a actualizar su contenido con el nuevo valor antes de que desaparezca del bus.

En la Figura 43 se muestra un registro de tipo cerrojo positivo.

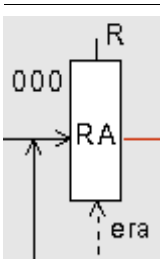


Figura 43: Registro de tipo cerrojo.

En un registro de este tipo la salida toma el valor que tenga la entrada, y seguirá cualquier cambio que se produzca en ésta durante todo el tiempo en que la microorden esté activa y la señal de reloj valga "1". Se utiliza un registro de este tipo para RA, de manera que la dirección de acceso a memoria o puerto de entrada/salida se estabilice en el bus Ai antes de pedir un ciclo de memoria.

Los registros se considerarán activos cuando alguna de sus microórdenes de control esté activa y cambiarán de aspecto, tal y como se muestra en la Figura 44.

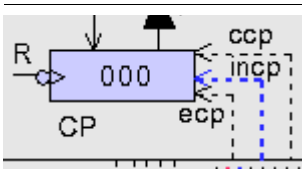


Figura 44: Registro activo.

En concreto el caso del Contador de Programa es un poco especial, ya que, tal y como muestra la figura, tiene tres microórdenes de control que permiten poner a cero su contenido (**ccp**), incrementarlo (**incp**) o sustituirlo por el valor presente a su entrada (**ecp**). Cuando hay más de una activa sólo una de ellas tendrá efecto. El orden de prioridad que indica qué microorden es la que prevalece es el siguiente: **ccp, incp, ecp.**

El registro de instrucciones se divide en dos partes, CO y CD, las cuales pueden estar iluminadas independientemente de que el registro esté activo en ese momento o no, tal y como muestra la Figura 45. De esta manera se indica que dicha parte de RI es de interés en ese momento.

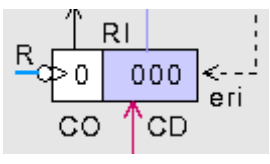


Figura 45: CD activo.

4.5 Buses

"La interconexión se hace a través de unas 'vías' o 'canales', que en adelante llamaremos buses, a los que pueden tener acceso varias unidades." "Físicamente un bus es un conjunto de líneas 'binarias'."

La Ruta de Datos de Símplez tiene cuatro buses:

- **Bus C.** 15 bits. Contiene todas las microórdenes del sistema.
- **Bus D**. 12 bits. Transporta la información entre la Memoria Principal, o los periféricos, y la UAL y RI.
- **Bus Ai**. 9 bits. Bus interno de direcciones.
- **Bus A**. 9 bits. Transporta direcciones de la UCP a la Memoria Principal y periféricos.

En lugar de pintar en paralelo las n líneas que lo componen, los buses se representan con una línea únicamente, tal y como muestra la Figura 46. Para indicar cuántas líneas lo forman se pinta el número de ellas, acompañado de una raya que cruza el bus, debajo del nombre.

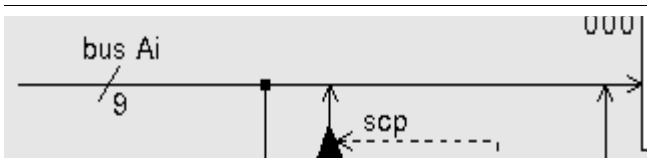


Figura 46: Bus Ai.

Cuando el bus contiene un valor estable, tanto si es valido como si se ha producido una colisión, cambia de color. Las transiciones por tanto no se resaltan, y el bus vuelve a pintarse en negro durante el tiempo que duren. Cada bus tiene un color asignado diferente. La Figura 47 muestra el bus Ai con un valor ya estabilizado.

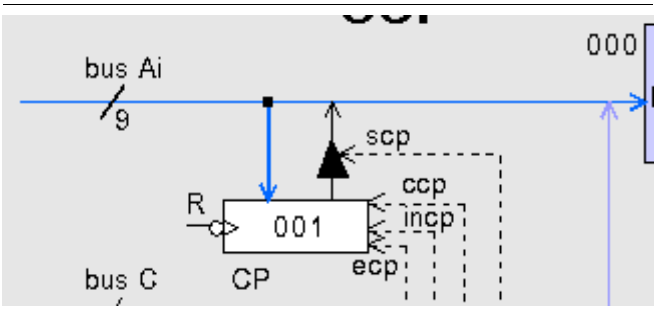


Figura 47: Bus Ai conteniendo un valor.

El caso del **bus A** es un poco especial. Dado que toma su valor del registro **RA**, y éste nunca tiene sus salidas en alta impedancia, siempre se estará volcando el contenido del registro en el bus. Esto hace que los únicos momentos en que el bus no aparezca iluminado sean durante las basculaciones de valores debido a un cambio de contenido del registro RA.

Las líneas de interconexión entre los buses y los elementos de la Ruta de Datos siguen el siguiente criterio. Las que son de entrada a algún elemento se iluminan con el estado del bus. Las que son de salida desde un elemento hacia el bus sólo se iluminan cuando se está volcando un valor sobre el mismo. Cuando se trata de una puerta lógica controlada por una microorden se iluminará cuando esta última esté activa, tal y como muestra la Figura 48.

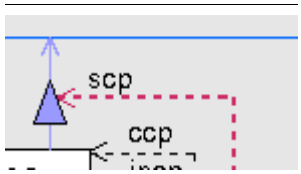


Figura 48: Puerta lógica activa.

Los buses no tienen "memoria", de manera que cuando se deja de volcar un valor sobre un bus, y pasa a alta impedancia, una operación de lectura sobre él genera un valor imprevisible.

En un momento dado solamente un componente debería estar depositando su información en el bus, de los contrario se produciría una colisión en el mismo y el valor contenido por el bus no tendría ninguna utilidad. Por el contrario varios elementos pueden leer al mismo tiempo la información contenida en él.

4.6 Memoria Principal

"... basta definir la MP como un componente en el que se pueden registrar (almacenar, o escribir) o recuperar (extraer, o leer) informaciones binarias." "La longitud de la palabra de la MP de Símplez es doce bits y su capacidad es 512 palabras."

La Memoria Principal de Símplez se muestra en la Figura 49.

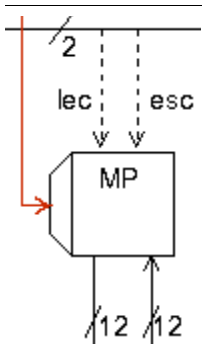


Figura 49: Memoria Principal.

La Memoria Principal se gobierna con dos microórdenes que permiten volcar sobre el bus D el contenido de una de sus posiciones (**lec**) o sustituir el contenido de una de ellas por el valor presente a su entrada (**esc**). La precedencia de las microórdenes que gobiernan la MP es la siguiente: **lec, esc.**

Cuando se comienza una operación sobre la memoria, ésta cambia de apariencia cambiando de color y mostrando la dirección a la que se accede y debajo el contenido de dicha dirección, tal y como se muestra en la Figura 50.

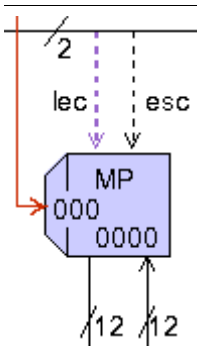


Figura 50: Operación sobre la MP.

Nota− La dirección a la que se accede se muestra siempre en decimal.

Cuando se trata de un ciclo de lectura y la memoria ya ha extraído la información de la posición correspondiente, se encenderá la línea de salida al bus D.

Una vez que se ha completado la operación a realizar desaparece la posición accedida y el aspecto de la memoria es el mostrado en la Figura 51.

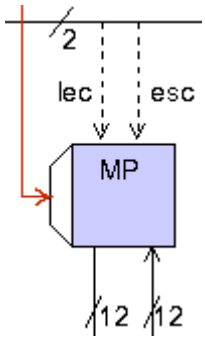


Figura 51: Final del ciclo de memoria.

La memoria permanece en este estado hasta que termina el ciclo de acceso a memoria.

Las cuatro últimas palabras de la memoria no se pueden utilizar como las demás, pues están reservadas para las comunicaciones con los periféricos. Se trata de direcciones robadas al mapa de memoria. La manera de manejarlas se explica en detalle en el Capítulo 6.

Restricciones temporales

Es importante entender las restricciones temporales a las que está sujeta la memoria, pues el no respetarlas significa un comportamiento incorrecto de ésta. Esto es especialmente importante cuando se está microprogramando el secuenciador. Hay dos parámetros temporales básicos: el tiempo de acceso y el ciclo de memoria.

El **tiempo de acceso para escritura** es el que transcurre desde que se le pide a la memoria con **esc** que escriba el contenido del bus D en una palabra hasta que este contenido queda definitivamente escrito.

El **tiempo de acceso para lectura** es el tiempo que transcurre desde que se le da la dirección y la orden de lectura **lec** a la memoria, hasta que se puede disponer del contenido de la palabra de dicha dirección en la salida.

Tras una operación sobre la memoria no siempre se puede iniciar inmediatamente otra. El **ciclo de memoria** es el tiempo que debe transcurrir para poder iniciar una nueva operación desde que se inició la anterior.

En una memoria de lectura y escritura, como es el caso de la de Símplez, ambos tiempos se pueden suponer iguales. En Símplez el tiempo de acceso se supone de 100 ns y el ciclo de memoria de 200 ns.

Además hay que tener en cuenta un aspecto más. Es necesario mantener estable la dirección de la posición de memoria, de donde se lee o a donde se escribe, desde un poco antes de que empiece el ciclo de reloj donde se realiza la operación y hasta un poco después de terminar el acceso. Esto se debe a que los circuitos descodificadores necesitan un tiempo para direccionar la poción en cuestión. Este tiempo es menor a un semiciclo de reloj.

Por último, en la memoria de Símplez no es necesario esperar cada cierto tiempo un tiempo de refresco entre operaciones. Lo que permite comenzar una nueva operación en cuanto termina el ciclo de memoria de la anterior.

Teniendo en cuenta estos aspectos temporales, las diferentes situaciones que se pueden dar en el simulador si no se respetan son:

- **Qué ocurre cuando no se respeta el tiempo que debe permanecer estable la dirección de acceso.** La operación se realiza, pero la posición apuntada por los circuitos descodificadores es imprevisible y no es posible saber en qué posición terminará por realizarse.

- **Qué ocurre si la memoria está siendo accedida y la microorden de la operación se desactiva.** La operación no se realiza, pero el ciclo de memoria se completará con 100 ns más.

- **Qué ocurre si el dato a almacenar durante un acceso de escritura no permanece constante.** La operación se realiza, pero el dato almacenado es imprevisible.

- **Qué ocurre si la dirección de acceso cambia y pasa a ser la de un puerto de entrada y salida.** La operación no se realiza, pero el ciclo de memoria se completará con 100 ns más.

- **Qué ocurre si lec y esc permanecen activas durante todo el ciclo de memoria.** Nada. La operación se realiza sin problemas. En cuanto la memoria termina de ejecutarla deja de atender a las microórdenes de control hasta que se cumple el ciclo de memoria.

Puede parecer que el segundo caso es imposible debido a que **lec** y **esc** permanecen constantes entre flancos de reloj. Hay que tener en cuenta que la memoria es un elemento asíncrono y que en cuanto detecta una dirección de su rango y una microorden activa se pone a trabajar, y en casos de microprogramaciones incorrectas se podría producir una de las situaciones descritas. Así para el segundo caso podría pasar que con **lec** o **esc** activas y direccionando los puertos de entrada y salida, en mitad del ciclo de reloj cambia la dirección en el bus Ai debido a que **era** se activa, y la nueva dirección cargada en RA pasa a pertenecer al rango de memoria. En ese momento la MP lo detecta y comienza a trabajar. Llegados al final del ciclo de reloj sus circuitos descodificadores ya apuntan a la posición en cuestión, pero en ese momento la microorden activa hasta ese momento pasa a valer “0”.

Como se puede observar se ha pretendido tener en cuenta todos los posibles casos que se pueden dar en la Ruta de Datos. El simulador permite comprobar los efectos catastróficos sobre el sistema de una mala microprogramación.

4.7 Dispositivos periféricos

Símplez tiene dos periféricos; un teclado y una pantalla de texto. La Figura 52 muestra su representación en la Ruta de Datos.

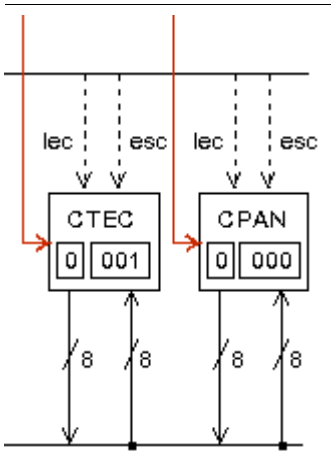


Figura 52: Periféricos de Símplez

Dentro de cada controlador se localizan dos registros. El de una cifra es el registro de control y el de tres es el registro de datos.

La sincronización es similar que en los casos de acceso a la Memoria Principal. A los 100 ns de generar **lec** o **esc** la operación de entrada o salida ha terminado. A continuación se verá que en el caso del controlador de pantalla habrá terminado en lo que respecta a la UCP, pero no necesariamente para el periférico.

A continuación se revisan los diferentes aspectos de los controladores según el acceso realizado y el estado en el que se encuentran. Para entender a fondo este apartado en conveniente leer el Capítulo 6, donde se explica el comportamiento de los puertos.

Controlador de Teclado

Los puertos de control y datos del controlador de teclado tienen las direcciones 510 y 511 del mapa de memoria respectivamente.

Cuando se accede a ellos para lectura se iluminan mientras esté activa la microorden **lec**. La Figura 53 muestra el aspecto del puerto de control cuando se ejecuta la instrucción “LD /510”.

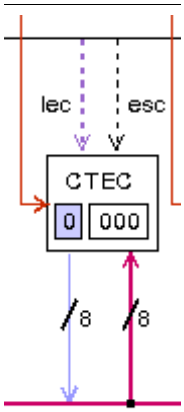


Figura 53: Puerto de control activo.

Los accesos en escritura a estos puertos se ignoran y el aspecto de CTEC no cambia, dado que los circuitos descodificadores de dirección no llegan a direccionar al registro de control al detectar que la operación es una escritura.

Cuando se pulsa una tecla del teclado de Símplez, el puerto de datos pasa a contener el código ISO Latin1 correspondiente y el puerto de datos contendrá un “1”, tal y como muestra la Figura 54.

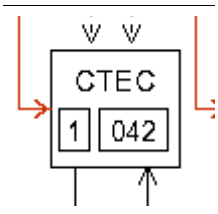


Figura 54: Puertos tras pulsar una tecla.

Controlador de Pantalla

Los puertos de control y datos del controlador de pantalla se corresponden con las direcciones 510 y 511 respectivamente.

El aspecto que tienen los puertos cuando se realiza sobre ellos una operación es semejante al de los puertos del controlador de Teclado. Sobre los puertos de este controlador se pueden realizar lecturas o escrituras, excepto en el caso de realizar una lectura sobre el puerto de datos. En este caso simplemente se descarta la operación por los circuitos descodificadores y el controlador no cambia de aspecto.

La escritura sobre el puerto de datos lleva asociados varios estados más. Las situaciones posibles se explican a continuación.

La primera situación se da cuando no se ha inicializado el puerto de control, la escritura sobre el puerto de datos ilumina el puerto en la Ruta de Datos mientras esté activa la microorden **esc**, pero su valor no se actualiza.

La segunda situación posible se produce en el caso de haber inicializado el puerto de control, veremos que los estados por lo que pasa el CPAN son los siguientes:

1. Durante el primer semiciclo de reloj de O0 el puerto pasa a estar direccionado y se ilumina, tal y como muestra la Figura 55.

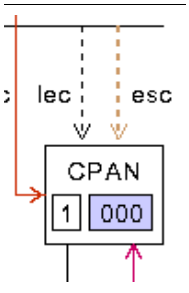


Figura 55: CPAN durante O0.

2. Durante el segundo semiciclo de reloj de O0 el puerto de datos actualiza su valor, el puerto de control pasa a contener un “0” y el aspecto de CPAN cambia indicando que comienza un ciclo de pantalla (es a partir de este instante que se miden los 33 ms).

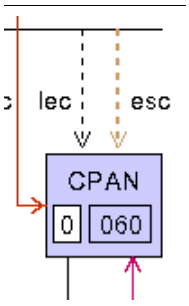


Figura 56: Comienza ciclo de pantalla.

3. Cuando la microorden **esc** deja de estar activa el registro de datos vuelve a su aspecto original, quedando el CPAN como en la Figura 57. Permanecerá así hasta que pase el ciclo de pantalla y el carácter se muestre, momento en el cual volverá al estado original y el registro de datos pasará a contener un “1”.

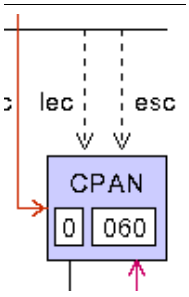


Figura 57: Aspecto durante el ciclo de Pantalla.

Por tanto, que esté iluminado CPAN significa que hay una operación de escritura en pantalla en curso. Que esté iluminado alguno de los registros significa que se está realizando una operación sobre ellos.

Las últimas situaciones posibles se producen cuando comienza una operación sobre uno de los puertos de CPAN durante el ciclo de pantalla. La lectura sobre el puerto de datos se descarta, y sobre el puerto de control se lleva a cabo (es de hecho la manera de comprobar que la operación de escritura en pantalla ha terminado).

Si se escribe sobre el puerto de control, éste actualizará su valor tal y como muestra la Figura 58.

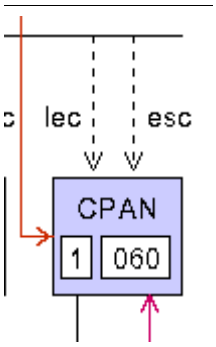


Figura 58: Modificación del puerto de control.

Si se escribe sobre el puerto de datos durante el ciclo de pantalla, el registro se ilumina pero la operación se descarta tanto si el registro de control está a “1” como si está a “0”.

4.8 Circuitos Combinacionales

El corazón del secuenciador cableado es el circuito combinacional mostrado en la Figura 59.

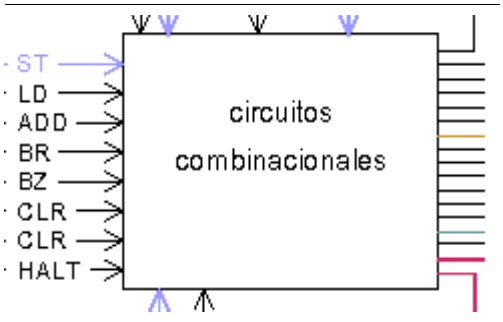


Figura 59: Circuitos Combinacionales del secuenciador cableado.

Son los encargados de ir generando las diferentes microórdenes de acuerdo al tipo de instrucción y al estado actual de la ejecución, determinado por los valores de los biestables T e I.

Los estados posibles son los indicados en la Tabla 3.

I	T	Estado
0	0	O0 (fase 0 del ciclo de operando)
0	1	O1 (fase 1 del ciclo de operando)
1	0	I0 (fase 0 del ciclo de instrucción)
1	1	I1 (fase 1 del ciclo de instrucción)

Tabla 3: Estados del secuenciador de Símplez.

Gracias a estos estados los Circuitos Combinacionales saben qué grupo de microórdenes deben generar en cada momento para una determinada instrucción.

En la Ruta de Datos el aspecto de los circuitos combinacionales cambia justo con el flanco de bajada de reloj y permanece como el mostrado en la Figura 60 hasta que las nuevas microórdenes generadas se estabilizan.

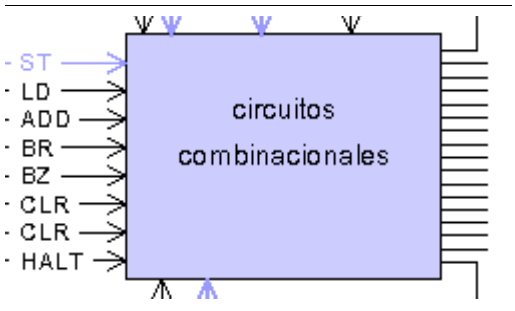


Figura 60: Circuitos combinacionales activos.

4.9 Memoria de Control

De manera análoga al caso del secuenciador cableado, la Memoria de Control es el corazón del secuenciador microprogramado. En este caso las instrucciones implican la ejecución sucesiva de una serie de microinstrucciones, que componen un microprograma, y que se encuentran almacenadas en la Memoria de Control.

Como se puede apreciar en la Figura 61, la Memoria de Control está dividida en dos partes. Una matriz de órdenes y una matriz de direcciones. La primera tiene un tamaño de 18 bits y son las microórdenes que gobernarán la Ruta de Datos. La segunda, de 3 bits, es la dirección dentro de la Memoria de Control de la siguiente microinstrucción a ejecutar.

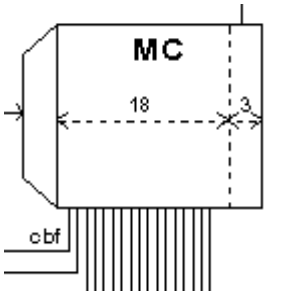


Figura 61: MC del secuenciador microprogramado.

En el caso de Símplez la MC tiene 16 palabras, por lo que son necesarios 4 bits para direccionar todas las microórdenes que la componen. La matriz de direcciones sólo tiene 3 bits, al igual que el código de operación de las instrucciones (lo que supone un repertorio de 8 instrucciones tal y como se explica en el Capítulo 8). Con el código de operación, se direccionan las ocho primeras direcciones de la tabla, una por cada una de las instrucciones del repertorio, y gracias a la microorden **sco**, que no existía en el secuenciador cableado, se direccionan las ocho direcciones restantes. Por tanto la microorden **sco** se comporta como el cuarto bit de dirección; pero no sólo uniéndose a los demás, sino decidiendo además si los tres restantes provendrán del código de operación (cargándolo de los tres bits de mayor peso del bus D) o de la matriz de direcciones de la MC. Tal y como muestran la Figura 62 y la Figura 63, cuando **sco** es "1" en el registro **μRA** se carga el CO, y cuando es "0" se carga DB.

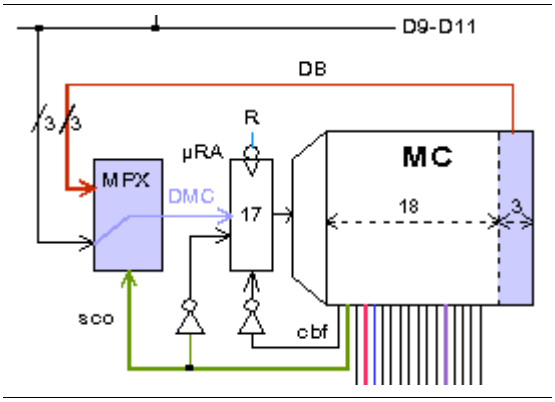


Figura 62: sco con valor 1.

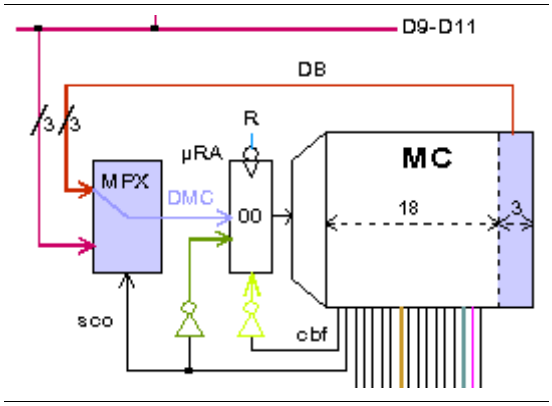


Figura 63: sco con valor 0.

En el caso del secuenciador cableado eran los biestables I y T, junto con el tipo de instrucción en curso, los que indicaban a los circuitos combinacionales la siguiente microinstrucción. Por cuestiones de diseño una instrucción se podía componer como máximo de cuatro microinstrucciones. En el caso del secuenciador microprogramado son las propias microinstrucciones las que indican cuál es la siguiente microinstrucción. Ahora podemos programar una instrucción compuesta por más de cuatro microinstrucciones.

La activación de **sco** marca el comienzo de una nueva instrucción, dado que cargará un nuevo Código de Operación. Éste es precisamente el criterio utilizado por el simulador para marcar el final de una instrucción y el comienzo de una nueva.

Hay otra microorden nueva respecto al secuenciador cableado, **cbf**, que hace la función que hacía el biestable F en aquel. Cuando se activa, se inhibe la carga en el registro **µRA**, y el secuenciador genera indefinidamente las últimas microórdenes .

La Memoria de Control puede presentar uno de los dos aspectos siguientes. Tras el flanco de bajada de reloj el aspecto es el de la Figura 64.

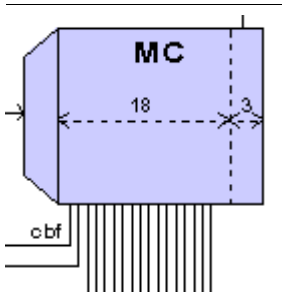


Figura 64: MC activa.

Cuando las nuevas microórdenes se estabilizan el aspecto de la MC pasa a ser el de la Figura 65.

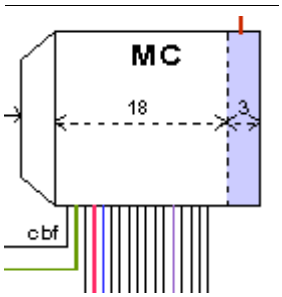


Figura 65: MC entre flancos.

4.10 Señal de Reloj

Símplez funciona de manera síncrona, gobernado por una señal de reloj periódica. El período de la señal de reloj es 100 ns (10 Mhz). Esta frecuencia viene marcada por la Memoria Principal. Al ser el elemento de la Ruta de Datos con mayores tiempos de respuesta es el que marcará cuándo termina un estado y empieza el siguiente. Dadas las restricciones temporales de la memoria de Símplez, en que una operación dura 100 ns y el ciclo de memoria 200 ns, es la frecuencia adecuada para tener a la memoria constantemente activa.

En la Ruta de Datos el reloj se representa con una línea de entrada a los elementos que necesitan sincronización, acompañada por la letra R.

4.11 Controles

En la ventana de la Ruta de Datos se encuentra los controles descritos en el Capítulo 1. Con ellos se puede cambiar el formato numérico de los datos mostrados en la Ruta de Datos, eligiendo entre octal o decimal.

Se puede detener la ejecución de la Ruta de Datos en cualquier momento, lo que a su vez permite cambiar la posición de la ejecución, situándola en un momento determinado que se quiera analizar en detalle. También se puede desplazar la ejecución hasta el comienzo de la instrucción. O en el caso de instrucciones que duren muchos ciclos de reloj, retrasarla hasta el primero almacenado.

Podemos modificar la velocidad de la ejecución desde la barra de desplazamiento, acelerándola.

En el caso de estar utilizando el secuenciador microprogramado se puede hacer que la ejecución continúe a pesar de estar activa la microorden **cbf**, que de lo contrario pararía la ejecución.

Por último, desde aquí se puede abrir la parte de la Ruta de Datos correspondiente al interior del secuenciador.

4.12 Funcionamiento conjunto

Por último, para entender cómo evoluciona todo en conjunto se analizará la ejecución de una instrucción. Se utilizará como ejemplo la instrucción "**LD /11**", almacenada en la posición 0 de memoria, la cual llevará el contenido de la posición 11 de memoria, que en el ejemplo será "15", al acumulador.

El estado inicial de la Ruta de datos será el mostrado en la Figura 66.

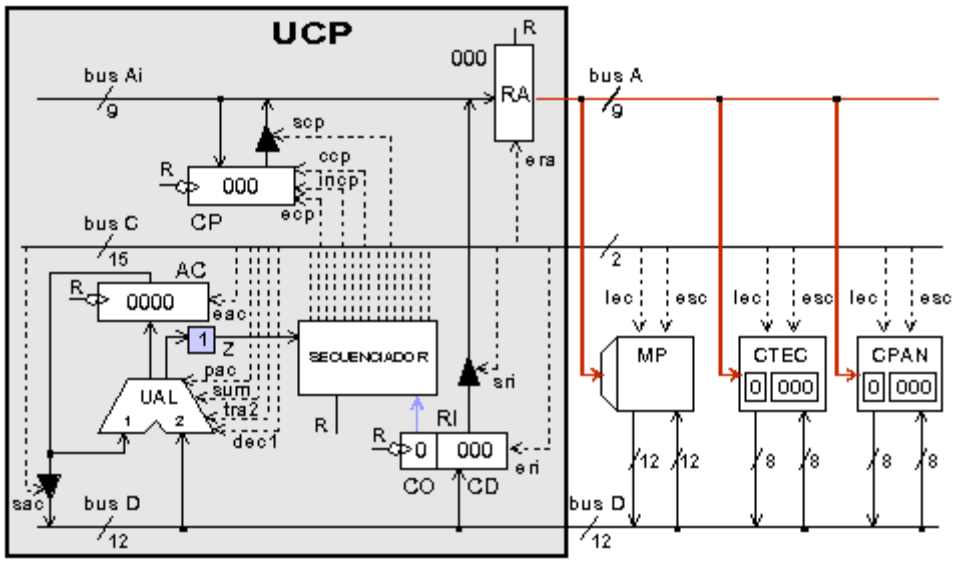


Figura 66: Estado inicial de la Ruta de Datos.

Tras el primer flanco de bajada de reloj las microórdenes comienzan a cambiar (en la imagen no se observa ninguna activa debido a que se trata de la Ruta de Datos tras inicializar la máquina) y pasados unos instantes se observará que se han activado **incp**, **eri** y **lec**. Con lo que en la Ruta de Datos se activarán **CP**, **RI** y la **MP**, tal y como muestra la Figura 67.

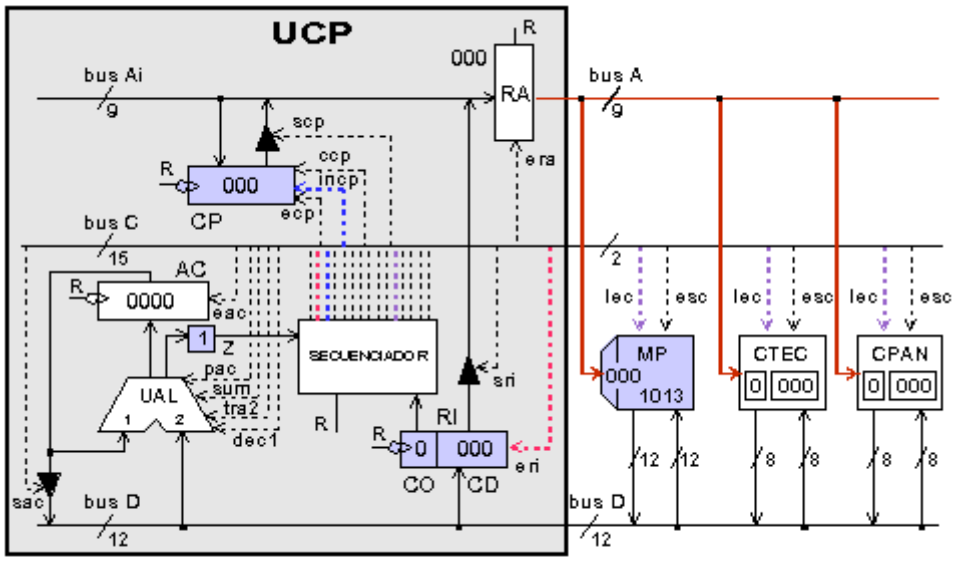


Figura 67: Ruta de Datos tras estabilizarse las microórdenes de I0.

En la MP es posible observar que se está direccionando la posición 0 y también el contenido de ésta, que se corresponde con el Código de Operación 1 y el Código de Operando en octal 13, que se corresponde con 11 en decimal. Pasados unos instantes la Memoria depositará en el bus D el contenido de la posición 0, tal y como muestra la Figura 68.

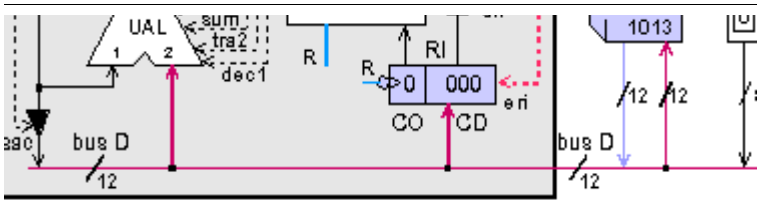


Figura 68: Bus D conteniendo los datos leídos de la memoria.

Esta situación se mantendrá hasta el siguiente flanco de bajada de reloj, momento en el cual se cargará en RI la instrucción a ejecutar, CP habrá incrementado su contenido y ambos registros se desactivarán. Las microórdenes se desactivarán y al hacerlo, la MP deja de volcar en el bus D y cambia de aspecto indicando que se está completando el ciclo de memoria. El contenido de CO se utiliza para obtener de los Circuitos Combinacionales la siguientes microórdenes, y pasado el período de transición **sri** y **era** se activan, tal y como muestra la Figura 69.

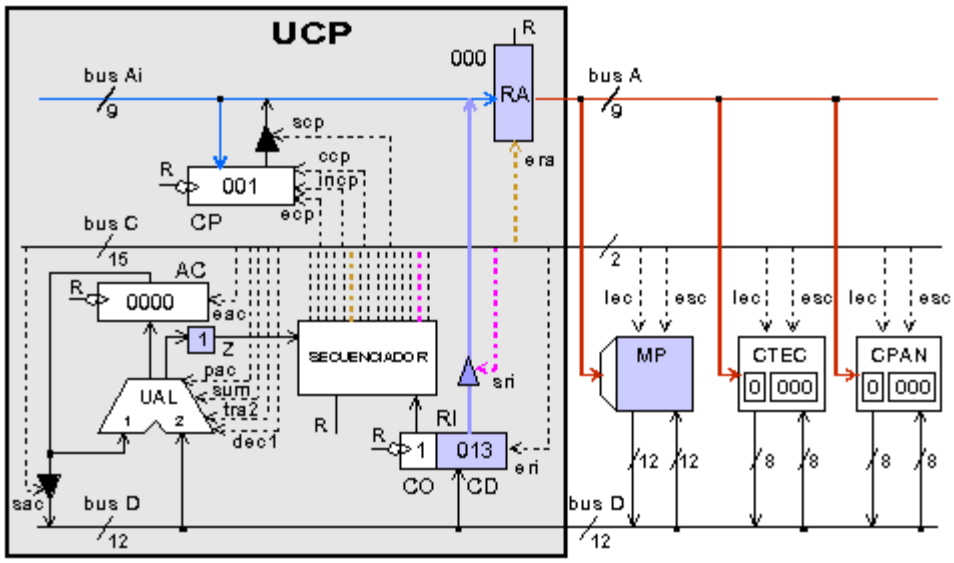


Figura 69: Ruta de Datos al comienzo de I1.

Al estar activas **sri** y **era** al llegar el flanco de subida de reloj, se cargará el contenido de CD, que es la dirección en memoria del operando, en el registro RA como se muestra en la Figura 70, donde se observa que la señal de reloj está activa.

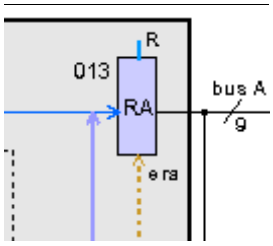


Figura 70: RA tras cargar la dirección.

Pasados unos instantes la dirección se habrá estabilizado en el bus A y estará preparada para la lectura del operando desde la MP.

Con el siguiente flanco de reloj se desactivan las microórdenes y comienzan a activarse las correspondientes a O0. Coincidiendo con el flanco, pero no debido a éste, termina también el ciclo de Memoria y ésta vuelve a estar prepara para una nueva operación.

Cuando se estabilizan las señales se observa que **eac**, **tra2** y **lec** están activas. Al detectar la memoria la activación de **lec** y que el bus A contiene una dirección de su rango se activa y comienza una nueva operación tal y como muestra la Figura 71.

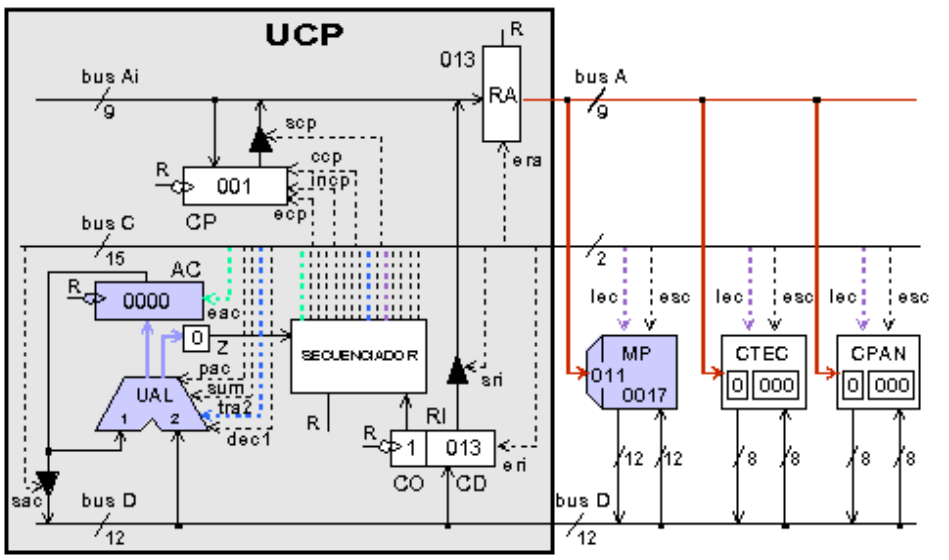


Figura 71: Ruta de Datos al comienzo de O0.

Ahora la posición direccionada es la 11, y es posible observar que su contenido en octal es “17”, en formato decimal “15”.

En la ruta de datos también están activos la **UAL** y el registro **AC** , al estar activadas la microórdenes **tra2** y **eac** respectivamente. También se observa cómo el biestable **Z** ha pasado a contener “0”. Esto se debe a que ahora la **UAL** está depositando en su salida el valor disponible en su entrada 2 y éste es diferente de “0”. A pesar de que nadie está volcando en el bus D en ese momento, eso no quiere decir que una lectura sobre él deba ser “0”. De hecho en el caso del ejemplo no lo es y el valor residual que contiene hace que el biestable **Z** cambie de valor.

Pasados unos instantes la Memoria deposita en el bus D el contenido de su posición 11, tal y como muestra la Figura 72.

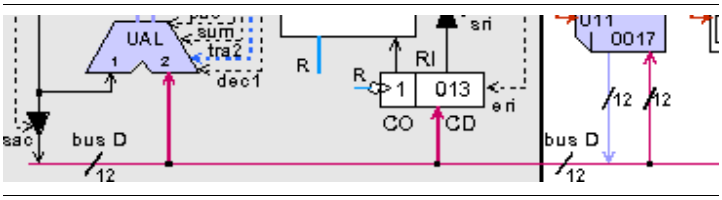


Figura 72: La memoria deposita en el bus D el operando.

Con la llegado del flanco de bajada de reloj el operando se carga en el registro AC y las microórdenes se desactivan, como muestra la Figura 73.

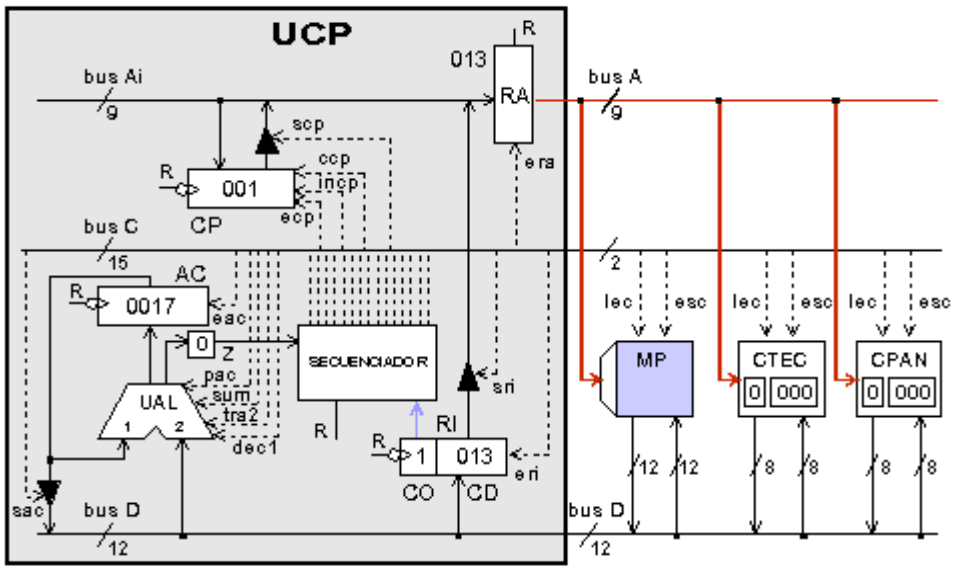


Figura 73: El operando se ha cargado en el registro AC.

Coincidiendo con el flanco, la Memoria deja de volcar en el bus D. Puede parecer que se debe a que la microorden **lec** se ha desactivado, pero no es así. La Memoria se rige por tiempos, y una vez que se han completado los 100 ns que le ocupa llevar a cabo la operación de lectura pasa a otro estado independientemente del estado de **lec**. De hecho ésta podría permanecer activa todo el ciclo restante sin que tuviera efecto alguno.

Pasado el período de transición las nuevas microórdenes, **scp** y **era** se activan . Esto hace que la puerta lógica controlada por **scp** se abra y el contenido del registro CP se deposite en el bus Ai, Figura 74.

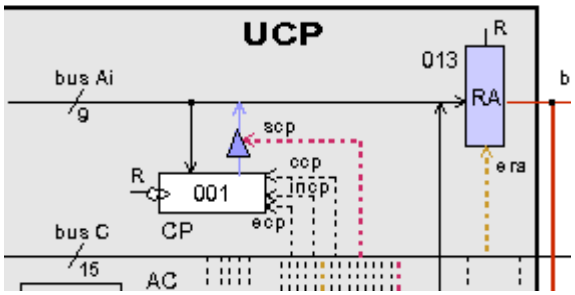


Figura 74: El contenido de CP se vuelca en el bus Ai.

Pasados unos instantes la dirección se habrá estabilizado en el bus Ai, con lo que éste se iluminará y con la llegada del flanco de subida de reloj la dirección se cargará en el registro RA, tal y como muestra la Figura 75. Es la dirección de la siguiente instrucción a ejecutar.

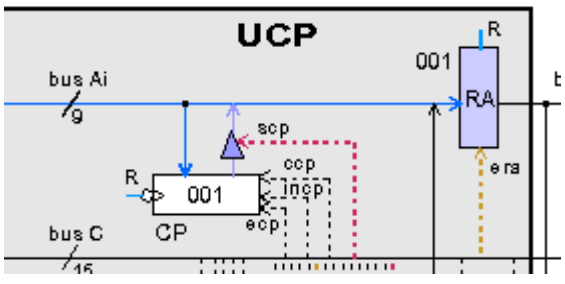


Figura 75: RA carga la dirección de la siguiente instrucción.

Acabada la ejecución de la instrucción el aspecto de la Ruta de datos es el mostrado en la Figura 76. Las microórdenes seguirán activas hasta el flanco de bajada de reloj, que en el simulador se considera perteneciente a la siguiente instrucción. La Memoria acabará también el ciclo de Memoria, permitiendo comenzar un nuevo ciclo de instrucción con I0.

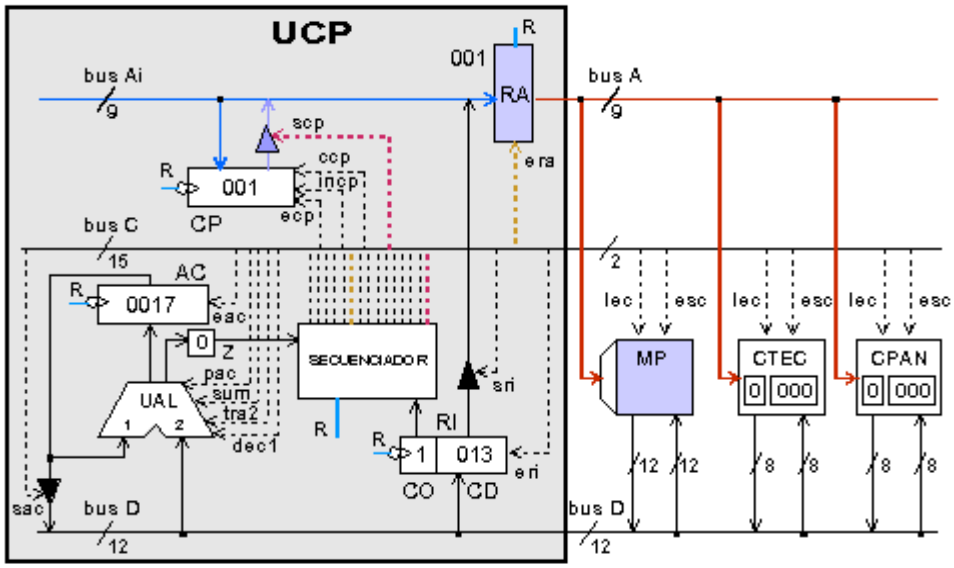


Figura 76: Ruta de Datos tras la ejecución de la instrucción.

Capítulo 5

Cronogramas

Tal y como se explica en [Fernández,98] *"Todas las microórdenes están sincronizadas con el reloj y permanecen constantes entre dos flancos de bajada. Cada vez que llega uno el secuenciador cambia de estado, y , con un pequeño retardo (despreciable con relación al período de reloj), algunas de las microórdenes que estaban "activas" desaparecen, otras que no lo estaban aparecen, y otras mantienen su valor."*

Los cronogramas son la representación temporal de la evolución de las microórdenes y buses que forman Símplez. Por tanto permiten seguir esos cambios indicados en el párrafo anterior y entender cómo influyen unas señales sobre otras. Son la herramienta complementaria de la Ruta de Datos, ya que permiten entender a fondo qué está ocurriendo en ésta.

En el simulador el cronograma de una instrucción se considera que comienza con el primer flanco de bajada de reloj de los ciclos que forman la misma, que en el caso del secuenciador cableado se corresponde con el ciclo común a todas la instrucciones, I0. Pero para disponer de una referencia se muestra también el valor que tenía la señal justo un instante antes del flanco. Si se seleccionan las señales que se quieren estudiar, se pulsa el botón de pausa y luego el de ejecución, se observarán estos valores anteriores al flanco de bajada, tal y como se muestra en Figura 77.

De manera análoga, el cronograma de una instrucción termina justo antes del flanco de bajada de reloj del ciclo que se corresponde con el I0 de la siguiente instrucción. Es importante resaltar por tanto, que el último flanco de bajada no está incluido, tal y como muestra la Figura 78, y como ya se comentó en el Capítulo 3. Estará incluido en el cronograma de la siguiente instrucción.

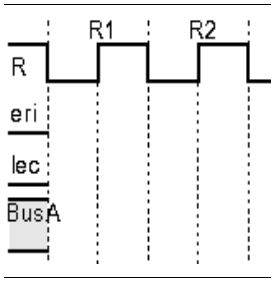


Figura 77: Comienzo de un cronograma.

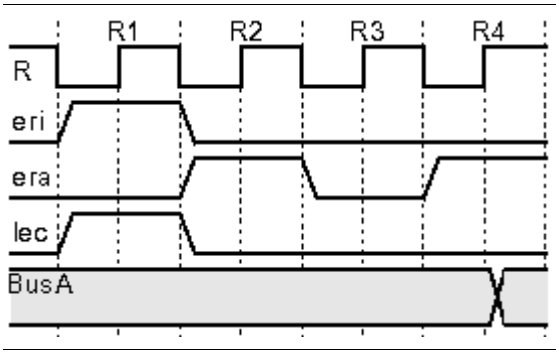


Figura 78: Final de un cronograma.

Nota− Los cronogramas no incluyen el flanco de reloj que marca el comienzo de la siguiente instrucción.

5.1 Representación de las señales

La única señal representada siempre es la señal de reloj. Esta señal en todas las computadoras es producida por un chip de reloj, el cual genera una señal cuadrada con una frecuencia marcada por la vibración de un cristal de cuarzo. La señal de reloj nunca es del todo cuadrada, pero dado que en el caso de Símplez su tiempo de transición es mucho más pequeño que el del resto de elementos de la máquina, se puede considerar cuadrada sin miedo a cometer error.

Como referencia se han dibujado una serie de líneas verticales discontinuas en cada flanco de bajada y subida del reloj. Esto facilita la identificación de los instantes de cambio de las señales.

Microórdenes

Las microórdenes tienen dos valores posibles, uno y cero, en terminología binaria, o verdadero y falso en booleana. Estos valores se representan en los cronogramas tal y como se muestra en las figuras siguientes.

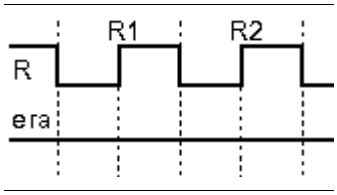


Figura 79: Valor cero.

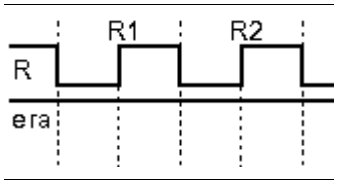


Figura 80: Valor uno.

Como se puede observar la manera de distinguirlos es tomar como referencia su posición respecto al nombre de la microorden en cuestión.

El paso de un valor a otro no es instantáneo y se produce tras un tiempo de transición, que en el caso de Símplez es algo menor de 15 ns. Dado que las microórdenes determinan lo que hará una instrucción, a este periodo de tiempo también se le llama descodificación de la instrucción. Las Figuras 81 y 82 muestran las dos posibles transiciones de una microorden: de 0 a 1, y de 1 a 0.

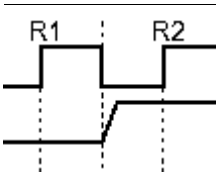


Figura 81: 0 a 1.

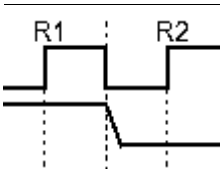


Figura 82: 1 a 0.

Hay un caso especial que se puede producir durante una transición en determinadas circunstancias con el secuenciador microprogramado. Las microórdenes **sri**, **ecp** y **scp** dependen del valor del biestable Z y del valor de la microorden **bbz** tal y como se explica en el Capítulo 9. El resultado en los cronogramas es el pico mostrado en la Figura 83.

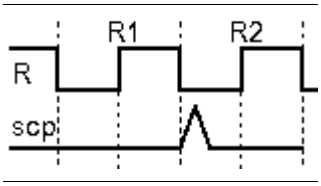


Figura 83: Pico durante transición.

Buses

En el caso de los buses, para ser precisos, no se habla de valor sino de estado. El bus contiene un valor, que es el conjunto de valores de los hilos que lo forman, y se encuentra en un estado. Es importante resaltar este punto porque uno de los estados en cuestión es la ausencia de valores.

Los buses se pueden encontrar en uno de los siguientes estados:

- **Alta impedancia.** Ningún elemento de la Ruta de Datos está volcando su contenido en el bus . Éste se encuentra en alta impedancia, es decir con ausencia de valor. Si en este estado se realiza una lectura sobre él, el valor resultante es imprevisible. En los cronogramas este estado se representa tal y como muestra la Figura 84.

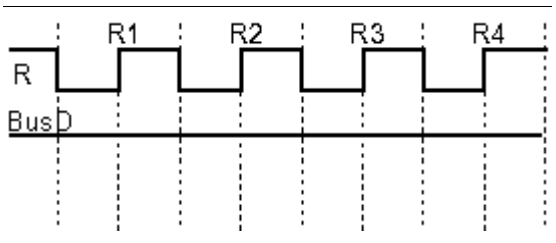


Figura 84: Bus en alta impedancia.

- **Con valor.** Un elemento, y sólo uno, está volcando su valor sobre el bus. Una lectura del bus devuelve dicho valor.

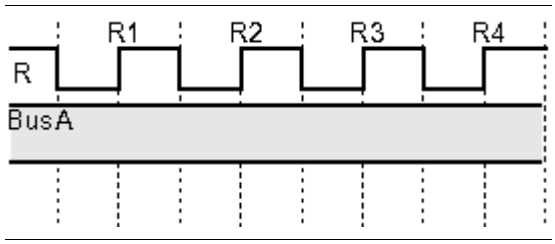


Figura 85: Bus con Valor.

- **Colisión**. Más de un elemento está volcando su valor sobre el bus. El resultado es una colisión en el mismo. Se puede considerar que el bus contiene un valor, al no estar en alta impedancia, pero una lectura del mismo produce un valor imprevisible. En lo cronogramas se representa en negro tal y como muestra la Figura 86.

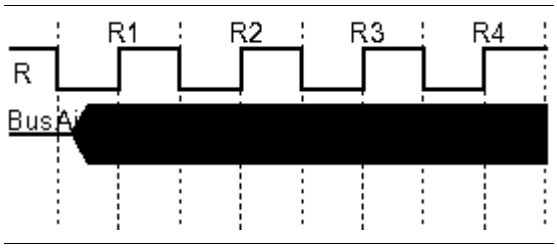


Figura 86: Bus con colisión.

Las transiciones entre estos estados toman, al igual que en el caso de las microórdenes, un tiempo algo inferior a 15 ns. Las tres posibles transiciones son:

- Transición de alta impedancia a contener un valor.

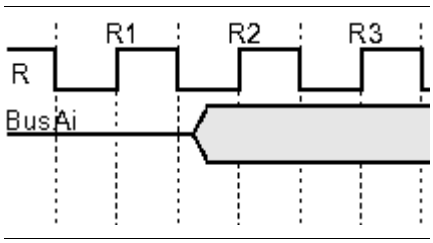


Figura 87: Transición de alta impedancia a contener valor.

- Transición de contener un valor a alta impedancia.

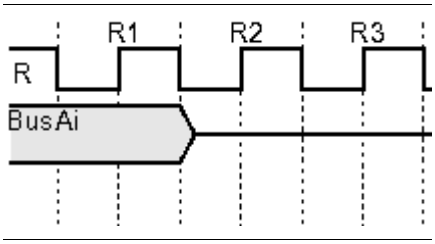


Figura 88: Transición de contener valor a alta impedancia.

- Transición de un valor a otro valor.

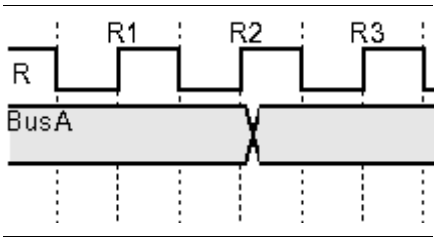


Figura 89: Transición entre valores.

5.2 Cronogramas de las instrucciones

Los cronogramas de las instrucciones de Símplez dependerán del tipo de secuenciador seleccionado.

A continuación se explican paso a paso los cronogramas para el caso del secuenciador cableado, para luego hacer los comentarios pertinentes en el caso del secuenciador microprogramado.

Tal y como se explica en [Fernández,98] las instrucciones pueden tener dos o cuatro estados, que se corresponden con ciclos de reloj. A estos se les denomina:

- **I0 e I1.** Estados del ciclo de instrucción. Se corresponden con los primeros dos ciclos de todas las instrucciones. Durante I0 se lee la instrucción a ejecutar desde la memoria y durante I1 se interpreta.

- **O0 y O1.** Estados del ciclo de operando. Únicamente las instrucciones que manejan datos pasan por estos estados. Durante O0 se lee o escribe, dependiendo de la instrucción de la que se trate, el operando de la memoria. Durante O1 continua la operación, la cual termina al final del mismo.

Se utilizarán esta terminología para explicar cada uno de los cronogramas. Hay que hacer resaltar que cuando se microprograme el secuenciador estos estados pueden no existir, tal y como se han definido, y los estados de las instrucciones dependerán totalmente del diseño de la Memoria de Control.

Estado I0

El estado I0, mostrado en la Figura 90, es común a todas las instrucciones.

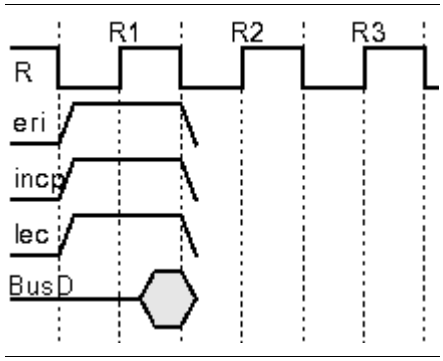


Figura 90: Ciclo correspondiente al estado I0.

Durante el mismo se lee de memoria la instrucción a ejecutar, para lo cual previamente se ha cargado en el registro RA la dirección de memoria donde se encuentra. Para que esto sea posible se activan las microórdenes **lec**, **eri** e **incp**. Con **lec** se indica a la Memoria que vuelque sobre el bus D el contenido de la dirección indicada por el bus A. Así, al final de I0, se observa que el bus D cambia de alta impedancia a contener un valor. Este valor será el de la instrucción a ejecutar. En el flanco de bajada que marca el final de I0, la instrucción disponible en el bus D se introduce en RI gracias a **eri**. En este instante también se incrementa CP, con **incp**, de manera que apunte a la siguiente instrucción a ejecutar.

Instrucción ST

El cronograma de la instrucción **ST** con el secuenciador cableado es el mostrado en la Figura 91.

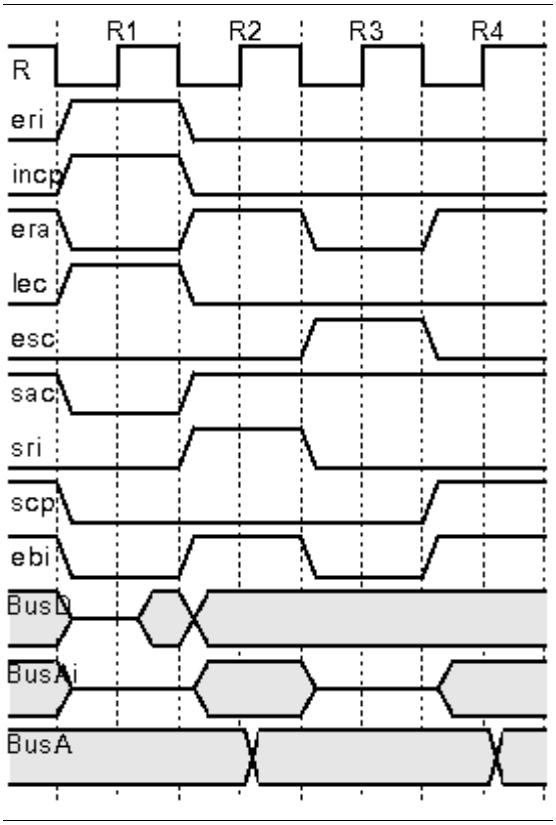


Figura 91: Cronograma de ST.

Esta instrucción lleva el contenido del acumulador a la posición de memoria deseada. Para ello son necesarias varias operaciones que a continuación se analizarán detalladamente.

Durante I1 (R2) se transfiere la dirección destino desde **CD** a **RA** de la siguiente manera: mediante **sri** se transfiere el contenido de CD al bus Ai, en el cronograma se observa como a mitad del primer semiciclo de R2 el bus Ai deja de estar en alta impedancia y contiene la dirección, y con **era** se le indica al registro RA que en el siguiente semiciclo de reloj con valor “1” cargue lo que tiene a la entrada (este registro no está regido por flanco así que durante todo el semiciclo de reloj reflejará cualquier cambio que se produzca en su entrada). Así se puede observar como tras el flanco de subida de reloj de R2 el contenido del Bus A, que contiene la dirección de acceso a memoria, cambia para reflejar el nuevo contenido del registro RA.

También durante I1 (R2) se activa **sac** de manera que ya esté estable cuando se active **esc** y se cumplan las restricciones temporales de escritura en memoria ya explicadas en el capítulo de la Ruta de Datos. Por la misma razón **sac** se mantiene activa durante todo el ciclo de escritura, que ocupa toda la duración de la instrucción, de manera que el contenido del acumulador permanece estable en el Bus D.

Al final de I1 (R2) se puede desactivar **sri** y **sac** al estar la dirección ya cargada en el registro RA. El Bus Ai pasa entonces a alta impedancia.

Durante O0 (R3) únicamente se activa **esc** para indicar a la memoria la operación a realizar.

Durante O1 (R4) se activan **scp** y **era** para ir preparando la lectura de la instrucción siguiente. Con **scp** el contenido del registro CP, previamente incrementado con **incp** durante I0, pasa al Bus Ai y con **era** este valor pasa al registro RA cuando la señal de reloj pasa a valer “1”.

En el cronograma se puede observar como la microorden **ebi** va cambiando de valor, de manera que permite ir pasando por los diferentes estados I0, I1, O0 y O1. En el caso del secuenciador microprogramado esta señal no existe y la única diferencia es la aparición de la microorden **sco**, mostrada en la Figura 92, la cual permite leer desde el Bus D el código de instrucción y así poder descodificar de qué instrucción se trata.

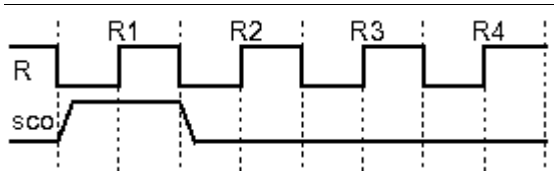


Figura 92: Microorden **sco** en el secuenciador microprogramado.

Instrucción LD

La instrucción **LD** permite leer un valor desde la memoria y transferirlo al acumulador. La dirección de memoria de dicho valor está contenida en CD. El cronograma de esta instrucción es el mostrado en la Figura 93.

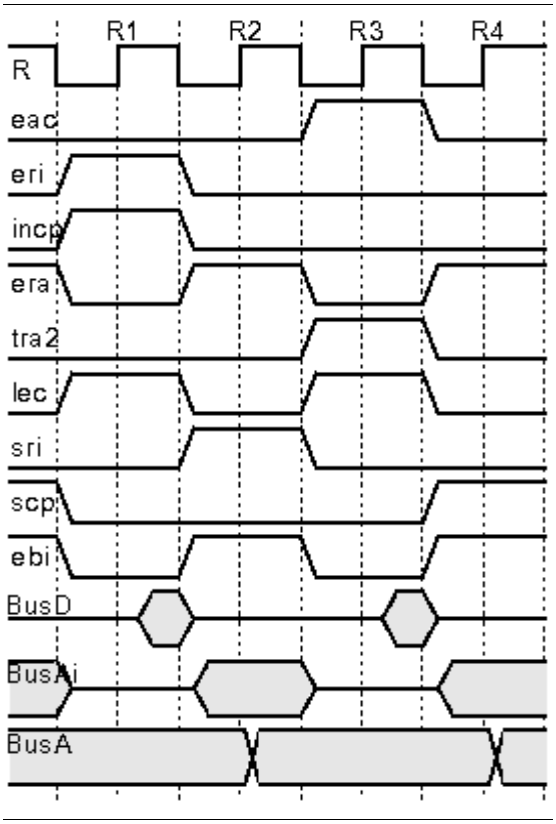


Figura 93: Cronograma de LD.

Esta instrucción también implica un ciclo de operando, por lo que al igual que en el caso de **ST**, durante I1 se transfiere el contenido de CD al registro RA mediante las microórdenes **sri** y **era**, de manera que al comienzo de O0 (R3) la dirección de memoria del operando ya está en el Bus A y puede comenzar su lectura.

Durante O0 (R3) se generan las microórdenes **lec**, **tra2** y **eac**. La microorden **lec** le indica a la memoria que la operación a realizar es una lectura. Con **tra2** le indica a la UAL que transfiera lo que hay en su entrada 2 a la salida y con **eac** le indica al registro AC que actualice su contenido en el siguiente flanco de bajada de reloj con ese valor. Es decir se ha creado un acceso directo entre el Bus D y el registro AC, de manera que cuando la memoria deposite el operando en el bus casi al final de O0 (R3) éste se pueda pasar directamente a AC.

Durante O1 (R4), como en el caso anterior, se prepara la lectura de la instrucción siguiente mediante las microórdenes **era** y **scp**.

Instrucción ADD

Esta instrucción permite sumar el contenido del acumulador con el contenido de la palabra de memoria de dirección indicada por CD, para luego dejarlo nuevamente en el acumulador. Su cronograma, mostrado en la Figura 94, es muy similar al de la instrucción **LD**.

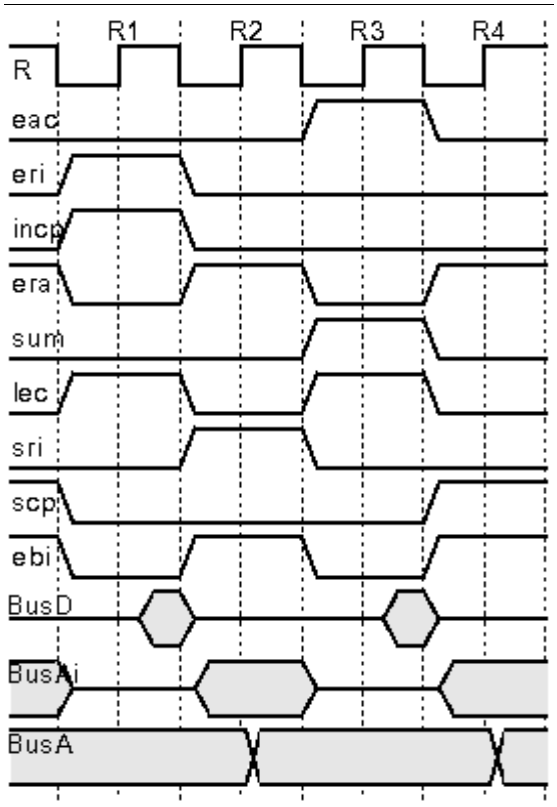


Figura 94: Cronograma de ADD.

Los estados I0 e I1 son iguales en ambas instrucciones. En el caso de **ADD** en O0 (R3) se generan **lec**, **sum** y **eac**. Cuando al final de O0 el contenido de la posición de memoria indicada por CD esté en el bus D, **sum** le indica a la UAL que en su salida deposite la suma de este valor y del contenido actual del acumulador. Dicha suma estará estable en la entrada del acumulador cuando llegue el flanco de bajada de reloj y actualizará el contenido del mismo gracias a **eac**.

Instrucción BR

La instrucción **BR** permite bifurcar incondicionalmente el flujo de ejecución a la dirección de indicada en CD. Su cronograma se muestra en la Figura 95.

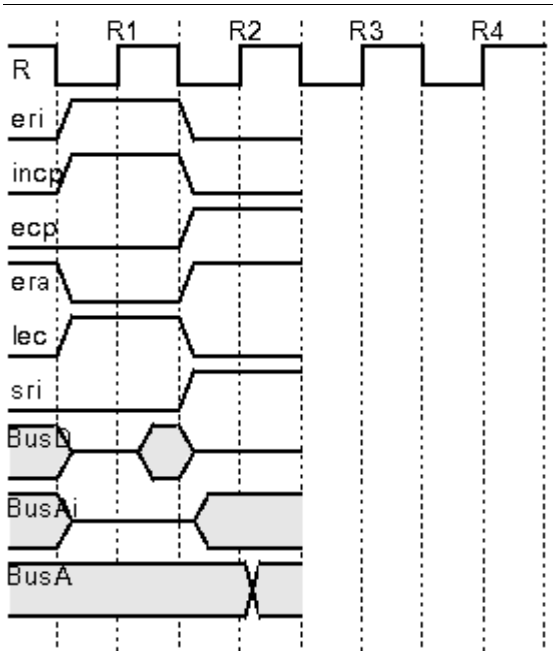


Figura 95: Cronograma de BR.

Durante I1 (R2) se debe transferir el contenido de **CD** al registro **RA**, para que cuando comience la ejecución de la siguiente instrucción la dirección ya se haya estabilizado en el bus A, por lo que, como en el caso de las instrucciones anteriores, se activan **sri** y **era**. Pero dicha dirección se debe introducir en el contador de programa también. Así que durante I1 también se activa **ecp**, de manera que el contenido del CP se actualice al final de I1.

Cuando termina la ejecución de **BR** la dirección de memoria de la siguiente instrucción ya está estable en el bus A y con el flanco de bajada de reloj que marca el comienzo de la siguiente instrucción se actualizará el contador de programa.

Instrucción BZ

Esta instrucción permite bifurcar el flujo de ejecución a la dirección indicada en CD si el resultado de la última operación realizada por la UAL ha sido cero, y por tanto **Z** contiene un "1". De no haberlo sido no se hace nada y el flujo de ejecución continúa en este punto.

Por tanto, en el caso de que **Z** contenga un "1" hay que bifurcar y el cronograma es igual al de **BR**. Por el contrario si el resultado de la operación de la UAL no fue cero y el contenido de **Z** es "0", el cronograma de **BZ** es el mostrado en la Figura 96.

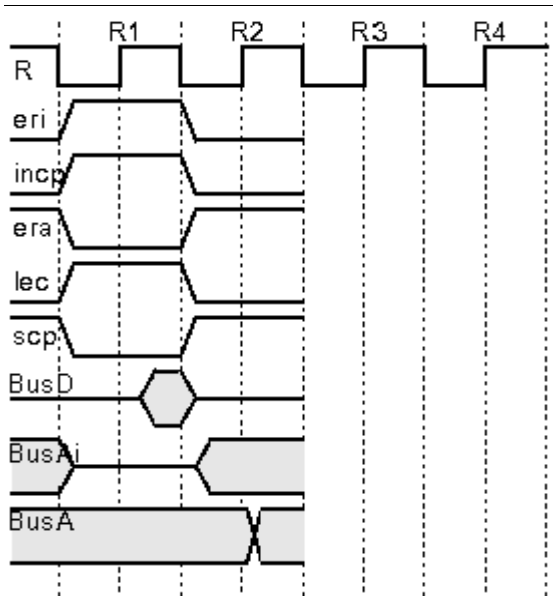


Figura 96: Cronograma de BZ cuando Z=0.

Lo único que se hace en este caso es generar durante I1 (R2) las microórdenes **scp** y **era**, de manera que el contenido del contador de programa, incrementado durante I1 (R1), se transfiere al bus Ai y de ahí al registro **RA** cuando la señal de reloj valga "1".

El cronograma de **BZ** cuando se utiliza el secuenciador microprogramado es un poco diferente. En la Figura 97 se muestra el cronograma cuando **Z** contiene un “1”, y se bifurca, y en la Figura 98 se muestra el cronograma con **Z** conteniendo un “0”, en que no se bifurca.

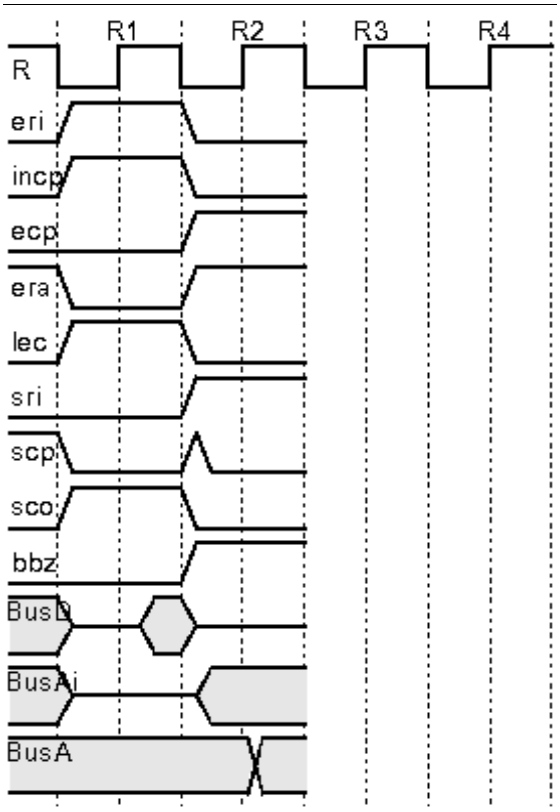


Figura 97: Cronograma de BZ con secuenciador microprogramado y (Z)=1.

Como se puede observar, en ambos casos la Memoria de Control genera durante I1 (R2) las microórdenes **ecp**, **sri**, **scp** y **bbz.** Al estar activa esta última microorden, la lógica de control del secuenciador inhibe unas microórdenes u otras dependiendo del valor del biestable Z. Pero dado que **bbz** se genera al mismo tiempo que ellas, hasta que no alcanza el valor "1" no empieza la inhibición y como resultado aparecen los picos mostrados en los cronogramas.

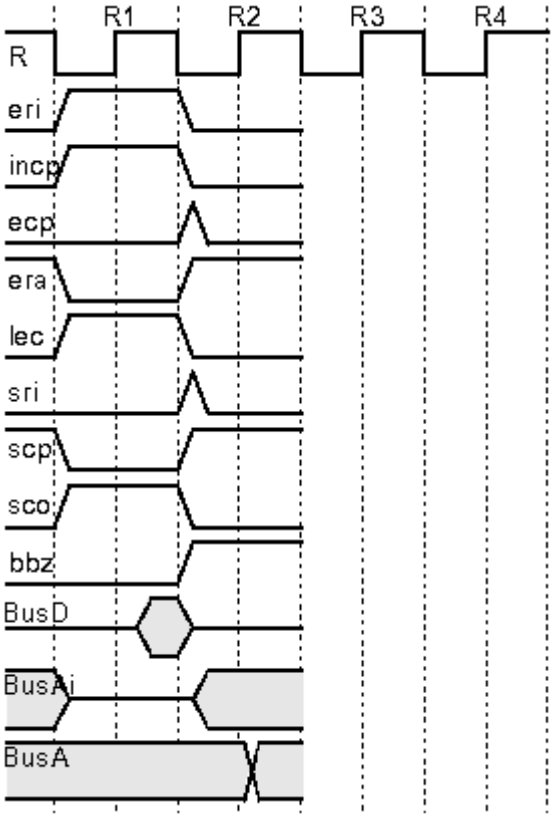


Figura 98: Cronograma de BZ con secuenciador microprogramado y (Z)=0.

Instrucción CLR

Esta instrucción permite poner a cero el contenido del acumulador. Su cronograma es el mostrado en la Figura 99.

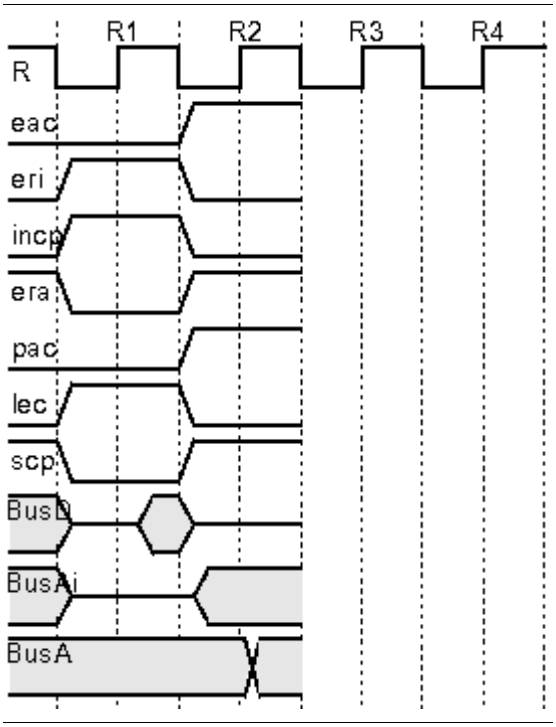


Figura 99: Cronograma de CLR.

Durante I1 (R2) el secuenciador genera la microorden **pac**, que pone a cero la salida de la UAL, y la microorden **eac**. De esta manera se carga este valor en el acumulador al final del ciclo de reloj. Al igual que en casos anteriores el nuevo valor se carga en el acumulador con el flanco de bajada de reloj que marca el comienzo de la siguiente instrucción. Además se activa **era** para cargar el contenido de CP, previamente incrementado, en el registro RA.

Instrucción DEC

Esta instrucción permite decrementar en una unidad el contenido del acumulador y volver a almacenar el valor resultante en él. Su cronograma se muestra en la Figura 100.

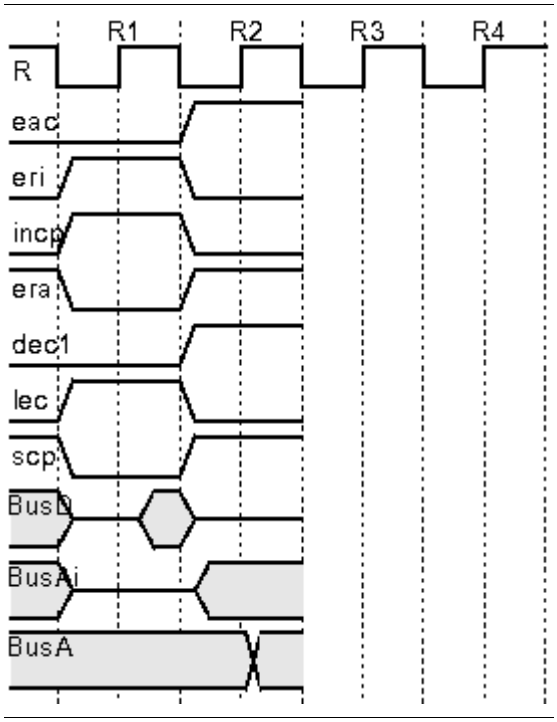


Figura 100: Cronograma de DEC.

Durante I0 (R2) se genera la microorden **dec1**, la cual hace que la UAL decremente en una unidad el valor presente en su entrada 1, que es la que viene de la salida del acumulador, y lo ponga en la salida. También se activa **eac**, por lo que con el flanco de bajada de reloj que marca el final de I1 el acumulador actualizará su contenido con el nuevo valor.

Instrucción HALT

Esta instrucción permite parar la ejecución de la máquina. Su cronograma se muestra en la Figura 101.

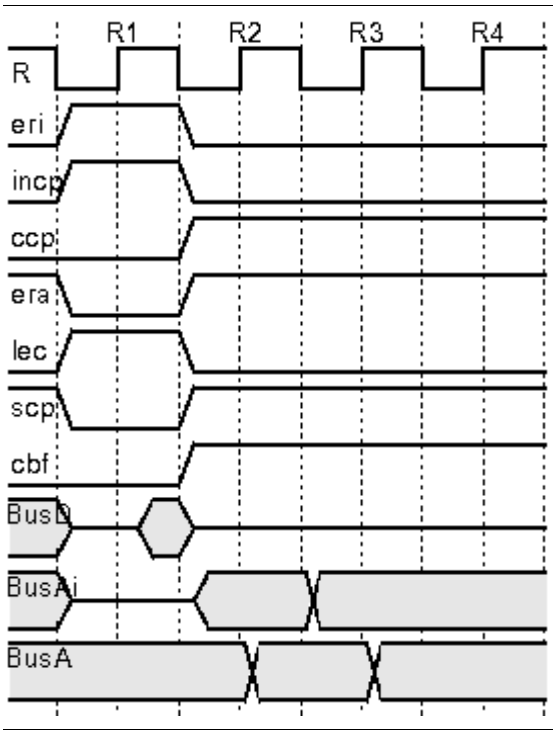


Figura 101: Cronograma de HALT.

Para inhibir la ejecución lo que hace el secuenciador es generar la microorden **cbf** durante I1 (R2). Además, para que cuando se resetee la máquina la ejecución comience por la instrucción de posición cero, el secuenciador genera las microórdenes **ccp**, **scp** y **era**, lo que introduce un cero tanto en el contador de programa con el flanco de bajada de reloj, como en el registro RA cuando la señal de reloj pase a valer “1”.

Es interesante observar como a partir de I1 estas microórdenes se quedan activas para siempre, hasta que se resetee. Los únicos cambios que se observan a partir de ese momento son los siguientes. El bus Ai deja de estar en alta impedancia al comienzo de R2 y sobre él se vuelca el contenido actual del contador de programa, aunque su valor en este momento carece de utilidad. Luego, al final de R2, el bus Ai pasa a contener un nuevo valor debido a que el registro de programa ha sido puesto a cero. Dado que **era** está activa, los cambios en el bus Ai se trasladan al registro RA cuando el reloj pasa a valer “1”, con lo que en el bus A se observa una transición de un valor a otro en la mitad de R2 y otra transición a la mitad de R3, cuando en el registro RA se ha cargado el valor final cero. A partir de este momento nada más cambia en los cronogramas.

Capítulo 6

Entrada y salida

Símplez tiene dos periféricos, un teclado y una pantalla de texto. A través del botón correspondiente de la barra de herramientas de la ventana principal se puede abrir la ventana que los muestra.

Para activar el teclado de Símplez basta con situar el puntero del ratón encima. Cuando esté activo el teclado de Símplez, si el puerto de datos del controlador del teclado está preparado para recibir un nuevo carácter, cualquier tecla que se pulse se transferirá directamente al mismo. Para introducir valores en el puerto de datos del teclado también se puede utilizar otro método. Basta con pinchar con el ratón en la casilla correspondiente a la dirección 511 de las tablas de Memoria de la ventana principal. Pero este último método sólo se puede utilizar si no se está en ejecución y el contenido del puerto de control es 0.

Figura 102: Teclado de Símplez.

Es importante resaltar que la introducción de datos a través del teclado no es síncrona con la representación en la Ruta de Datos. Cuando se esté representando la ejecución de una instrucción en la Ruta de Datos y se introduzca un carácter a través del teclado, el estado de los puertos asociados al teclado no cambia inmediatamente dado que la representación en la Ruta de Datos lleva un retraso de varios ciclos de reloj respecto a la ejecución. El retraso máximo es de 4 ciclos. En el caso del secuenciador cableado el cambio se verá reflejado al terminar la ejecución de la instrucción representada en la Ruta de Datos.

La pantalla de Símplez tiene 20 líneas de 80 caracteres cada una. El tiempo que tarda la pantalla en mostrar un carácter se denomina ciclo de pantalla. En Símplez este ciclo dura 33 ms. Dado que 33 ms es mucho tiempo y exige la ejecución de miles de instrucciones antes de que se imprima el carácter, desde el selector de ciclo de pantalla que incluye la ventana de Pantalla y Teclado se podrá reducir ese tiempo hasta 16 ciclos de reloj. De esta manera es posible simular la entrada y salida sin necesidad de esperar tanto tiempo por carácter.

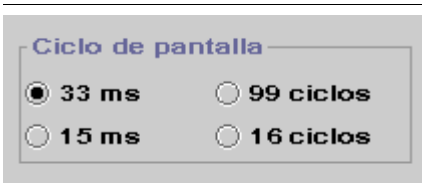


Figura 103: Selector de ciclo de pantalla.

Para comunicarse con los puertos desde un programa se utiliza el acumulador. En cada operación se transfiere un carácter codificado en el código ISO Latin1. Dependiendo del tipo de operación se tendrá una transferencia en un sentido o en otro, desde el acumulador a la pantalla o desde el teclado al acumulador.

6.1 Puertos

En Símplez los puertos de entrada y salida comparten el espacio de direccionamiento de la Memoria Principal. Para comunicarse con ellos se seguirá el mismo procedimiento utilizado para acceder a una posición de la MP, se pondrá en el bus de direcciones la dirección del puerto y se generará **lec** o **esc**, dependiendo del tipo de operación a realizar. El descodificador de direcciones detectará que no se trata de una dirección de la MP y activará el puerto correspondiente.

La dirección 509 corresponde al puerto de la pantalla de texto y la 511 al puerto del teclado. Por tanto, el resultado de ejecutar la instrucción "ST /509" es la transferencia de los bits 0 al 7 desde el acumulador al puerto de la pantalla, de manera que se interpretan como la codificación de un carácter. De manera similar la ejecución de la instrucción "LD /511" provoca la transferencia de la codificación de un carácter desde el puerto de teclado a los bits 0 al 7 del acumulador.

La sincronización con los puertos de entrada y salida se considera semejante a la de las operaciones con la Memoria Principal explicada en el Capítulo 4, y a los 100 ns de generar **lec** o **esc** la operación de entrada o salida ha terminado.

Que la operación de entrada o salida haya terminado para la UCP no quiere decir que el periférico correspondiente esté preparado para iniciar una nueva operación. A los 100 ns el contenido del acumulador se habrá transferido a un registro interno, o puerto del controlador del periférico. La UCP podrá continuar con la ejecución de otra instrucción, pero el periférico aún tardará algún tiempo en completar la operación. En el caso concreto de la pantalla de Símplez han de pasar 33 ms para que el carácter aparezca en la misma. Por tanto, dado que algunos periféricos son mucho más lentos que la UCP de la máquina, antes de comenzar una operación hay que asegurarse de que el periférico está preparado. De lo contrario se podría estar enviando caracteres consecutivos a la pantalla y estar perdiéndolos debido a que esta última no ha terminado de escribir el primero de ellos. De igual manera, lecturas sucesivas sobre el teclado devolverían el mismo carácter si el usuario no ha introducido nuevos caracteres y el programa no lo ha comprobado.

La forma de permitir que un programa pueda comprobar el estado de un puerto antes de realizar una operación sobre él, es acompañar a cada puerto de datos de un puerto de control por el que el controlador de periférico puede comunicar el estado de éste a la UCP. El estado puede ser simplemente una información binaria que indique si está o no preparado.

En Símplez los puertos de control se encuentran en las direcciones del mapa de memoria 508, para la pantalla, y 510, para el teclado. Cuando contengan un "1" significará que el periférico en cuestión estará preparado para una nueva operación, y "0" si no lo está. El mapeo de puertos de entrada y salida en Símplez y su comportamiento es el resumido en la Tabla 4, obtenida de [Fernández,98].

dir	Función
508	Puerto de estado de la pantalla. Debe tener el valor inicial "1". El controlador pone "0" cuando recibe la orden de escribir un carácter (provocada por la ejecución de ST /509), y pone "1" cuando ha terminado y está preparado para escribir otro.
509	Puerto de datos de la pantalla.
510	Puerto de estado del teclado. Debe tener el valor inicial "0". El controlador pone "1" cuando se pulsa una tecla, y "0" cuando recibe la orden de leer el carácter (provocada por la ejecución de LD /511).
511	Puerto de datos del teclado.

Tabla 4: Puertos de datos y control de Símplez.

Antes de ejecutar una instrucción "ST /509" para sacar un nuevo carácter por pantalla, se debe ejecutar la instrucción "LD /508" para llevar el contenido del puerto de control, es decir el estado del periférico, al acumulador y poder comprobar si está preparado para recibir el nuevo carácter. De manera análoga, antes de ejecutar una instrucción "LD /511" se debe ejecutar la instrucción "LD /510" para comprobar si hay un nuevo carácter en el puertos de datos del teclado.

La operación adecuada a realizar depende del puerto al que se acceda, pero un programa puede no respetarlo y realizar otra operación. El comportamiento de los puertos de los periféricos respecto a las lecturas y escrituras es el siguiente. Los puertos de datos y de control del teclado ignoran las escrituras. El puerto de control de pantalla atiende lecturas y escrituras en cualquier momento. El puerto de datos de pantalla ignora las lecturas, y para las escrituras es necesario que el puerto de control esté a uno y que no haya un carácter pendiente de ser mostrado por pantalla para poder transferir el contenido del bus D al registro.

Hay que observar que cuando se realice una lectura sobre el puerto de pantalla, lo que realmente se carga en el acumulador es el valor del bus D, que en ese momento seguramente se encuentre en alta impedancia.

Nota− Para poder mandar un carácter al puerto de datos de pantalla es necesario inicializar por programa el puerto de control a "1".

Los registros de control únicamente contienen un "1" o un "0", por tanto para las escrituras sobre el puerto de control del teclado sólo se considera el último bit del valor transferido.

6.2 Representación de los caracteres

La comunicación con el usuario a través de los periféricos se realiza mediante cadenas de caracteres. Para representar cada carácter de la cadena se utiliza el código ISO Latin1. Los caracteres que forman parte de una cadena se almacenan en posiciones consecutivas de memoria. El código ISO Latin1, como se verá a continuación, utiliza 8 bits para codificar un carácter y la longitud de palabra de la Memoria Principal de Símplez es de 12 bits, por lo que se almacenará un carácter por cada palabra de memoria y ocupará los ocho bits menos significativos de la misma.

Es importante indicar que cuando el usuario quiera introducir un número en Símplez, o cuando se quiera sacar un número por pantalla será necesario representar éste como una cadena de caracteres. Supóngase que se quiere sacar por pantalla el número 218. Si se transfiere desde el acumulador al puerto de datos de la pantalla el valor 218, tras cumplirse el ciclo de pantalla aparecerá en ésta el carácter "Ú" en lugar del número 218. Para lograr que aparezca el número, se debe transferir en operaciones consecutivas los caracteres "2", "1"y "8" al puerto de datos de la pantalla. Es decir, se deben introducir consecutivamente en el acumulador los números "50", "49" y "56", que son los que se corresponden en ISO Latin1 con "2", "1" y "8" respectivamente.

Por tanto es labor del programa el traducir de cadenas de caracteres a números, o viceversa, cuando quiera sacarlos por pantalla, o introducirlos por teclado.

Del juego de caracteres de ISO Latin1, los que entiende Símplez son lo mostrados en las tablas siguientes.

Dec	Hex	Oct	Secuencia escape	
7	07	007	BEL	(bell)
8	08	010	BS	(backspace)
9	09	011	TAB	(horizontal tab)
10	0A	012	LF	(NL line feed, new line)
12	0C	014	FF	(NP form feed, new page)
13	0D	015	CR	(carriage return)

Tabla 5: Tabla de secuencias ASCII implementadas.

Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char
32	20	040	SPACE	64	40	100	@	96	60	140	‘
33	21	041	!	65	41	101	A	97	61	141	a
34	22	042	"	66	42	102	B	98	62	142	b
35	23	043	#	67	43	103	C	99	63	143	c
36	24	044	$	68	44	104	D	100	64	144	d
37	25	045	%	69	45	105	E	101	65	145	e
38	26	046	&	70	46	106	F	102	66	146	f
39	27	047	’	71	47	107	G	103	67	147	g
40	28	050	(	72	48	110	H	104	68	150	h
41	29	051	)	73	49	111	I	105	69	151	i
42	2A	052	*	74	4A	112	J	106	6A	152	j
43	2B	053	+	75	4B	113	K	107	6B	153	k
44	2C	054	,	76	4C	114	L	108	6C	154	l
45	2D	055	−	77	4D	115	M	109	6D	155	m
46	2E	056	.	78	4E	116	N	110	6E	156	n
47	2F	057	/	79	4F	117	O	111	6F	157	o
48	30	060	0	80	50	120	P	112	70	160	p
49	31	061	1	81	51	121	Q	113	71	161	q
50	32	062	2	82	52	122	R	114	72	162	r
51	33	063	3	83	53	123	S	115	73	163	s
52	34	064	4	84	54	124	T	116	74	164	t
53	35	065	5	85	55	125	U	117	75	165	u
54	36	066	6	86	56	126	V	118	76	166	v
55	37	067	7	87	57	127	W	119	77	167	w
56	38	070	8	88	58	130	X	120	78	170	x
57	39	071	9	89	59	131	Y	121	79	171	y
58	3A	072	:	90	5A	132	Z	122	7A	172	z
59	3B	073	;	91	5B	133	[	123	7B	173	{
60	3C	074	<	92	5C	134	\	124	7C	174	|
61	3D	075	=	93	5D	135	]	125	7D	175	}
62	3E	076	>	94	5E	136	^	126	7E	176	~
63	3F	077	?	95	5F	137	_	127	7F	177	DEL

Tabla 6: Tabla 1 de caracteres ISO Latin 1.

Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char
160	A0	240	SPACE	192	C0	300	À	224	E0	340	à
161	A1	241	¡	193	C1	301	Á	225	E1	341	á
162	A2	242	¢	194	C2	302	Â	226	E2	342	â
163	A3	243	£	195	C3	303	Ã	227	E3	343	ã
164	A4	244	¤	196	C4	304	Ä	228	E4	344	ä
165	A5	245	¥	197	C5	305	Å	229	E5	345	å
166	A6	246	¦	198	C6	306	Æ	230	E6	346	æ
167	A7	247	§	199	C7	307	Ç	231	E7	347	ç
168	A8	250	¨	200	C8	31	È	232	E8	350	è
169	A9	251	©	201	C9	31	É	233	E9	351	é
170	AA	252	ª	202	CA	312	Ê	234	EA	352	ê
171	AB	253	«	203	CB	313	Ë	235	EB	353	ë
172	AC	254	¬	204	CC	314	Ì	236	EC	354	ì
173	AD	255	-	205	CD	315	Í	237	ED	355	í
174	AE	256	®	206	CE	316	Î	238	EE	356	î
175	AF	257	¯	207	CF	317	Ï	239	EF	357	ï
176	B0	260	°	208	D0	320	Đ	240	F0	360	ð
177	B1	261	±	209	D1	321	Ñ	241	F1	361	ñ
178	B2	262	²	210	D2	322	Ò	242	F2	362	ò
179	B3	263	³	211	D3	323	Ó	243	F3	363	ó
180	B4	264	´	212	D4	324	Ô	244	F4	364	ô
181	B5	265	µ	213	D5	325	Õ	245	F5	365	õ
182	B6	266	¶	214	D6	326	Ö	246	F6	366	ö
183	B7	267	·	215	D7	327	×	247	F7	367	÷
184	B8	270	¸	216	D8	330	Ø	248	F8	370	ø
185	B9	271	¹	217	D9	331	Ù	249	F9	371	ù
186	BA	272	º	218	DA	332	Ú	250	FA	372	ú
187	BB	273	»	219	DB	333	Û	251	FB	373	û
188	BC	274	¼	220	DC	334	Ü	252	FC	374	ü
189	BD	275	½	221	DD	335	Ý	253	FD	375	ý
190	BE	276	¾	222	DE	336	Þ	254	FE	376	þ
191	BF	277	¿	223	DF	337	ß	255	FF	377	ÿ

Tabla 7: Tabla 2 de caracteres de ISO Latin 1.

Capítulo 7

Herramientas

El simulador incluye un par de herramientas muy útiles para el desarrollo de programas y de secuenciadores microprogramados. En este capítulo se explica en detalle su utilización.

7.1 Editor de programas

El editor de programas ha sido desarrollado por Francisco Javier Rodríguez. Esta herramienta incluye un editor y un ensamblador−montador de programas. Un ensamblador es un programa que permite traducir programas escritos en un lenguaje simbólico a las correspondientes instrucciones en binario que entiende una máquina. Partiendo de un programa fuente, se verá a lo largo de este apartado cómo introducirlo en el editor de programas para posteriormente ensamblarlo y montarlo, de manera que se haya traducido a código binario y sea ejecutable por Símplez.

En la página 108 se incluye la especificación completa del lenguaje ensamblador necesaria para utilizar el editor. Y al final del apartado, página 117, a modo de resumen se explican las diferentes operaciones que implica el proceso de traducción desde un lenguaje simbólico a lenguaje máquina.

El editor es accesible a través del menú **Herramientas**, seleccionando el campo "**Editor de programas**". Se abrirá la ventana mostrada en la Figura 104 y ya introducida en el Capítulo 1.

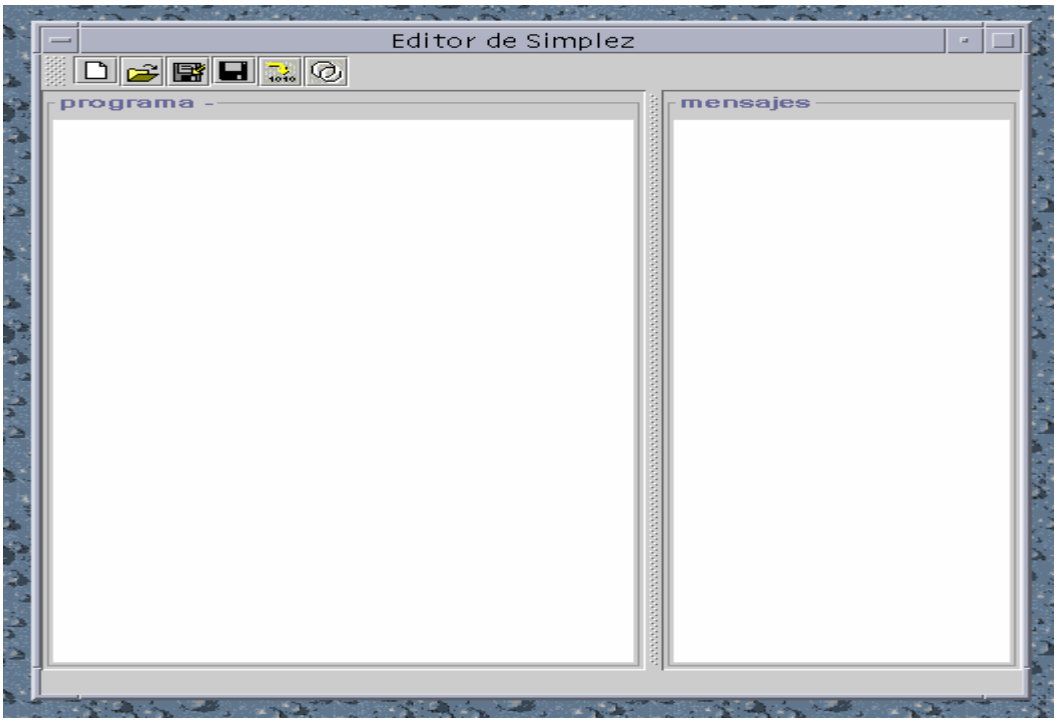


Figura 104: Editor de programas de Símplez.

Debajo de la barra de herramientas hay dos áreas de texto. El tamaño de una respecto a la otra se puede cambiar desplazando la columna central que las separa. Para ello basta con pinchar el ratón encima de ella, y manteniéndolo pulsado, arrastrarla a la posición deseada.

El área con etiqueta **programa** es el área de edición. En ella se pueden escribir los programas fuente utilizando el lenguaje ensamblador descrito más adelante en la página 108. Como en cualquier editor existen una serie de operaciones de manejo de texto asociadas a diferentes teclas o combinaciones de ellas, descritas en el punto siguiente, y de manejo de ficheros, pudiéndose en cualquier momento salvar a disco el contenido del área de edición, cargar un programa fuente desde un fichero o comenzar a trabajar con un programa desde cero. Estas operaciones con ficheros son realizables a través de los botones correspondientes de la barra de herramientas.

El área con etiqueta **mensajes** mostrará el resultado de las operaciones realizadas con el editor. En el proceso de ensamblado se analiza el programa fuente línea a línea y se va traduciendo a código máquina. Si se detecta un error en el código, el análisis se detiene y en el área de **mensajes** aparece un mensaje de error indicando la línea donde se ha producido y la naturaleza del mismo. La Figura 105 muestra un mensaje de error producido durante la fase de ensamblado.

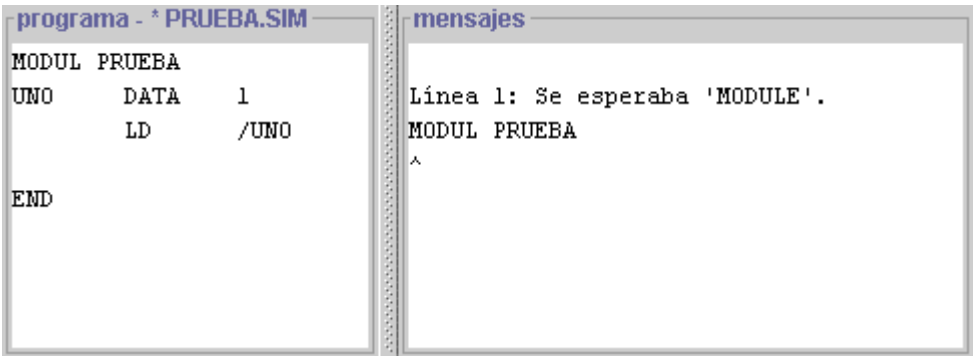


Figura 105: Error durante el ensamblado.

Una vez ensamblado el programa fuente, en el área de mensajes aparece el código máquina del mismo en notación octal, acompañado de las direcciones de memoria que ocupará, tal y como muestra la Figura 106.

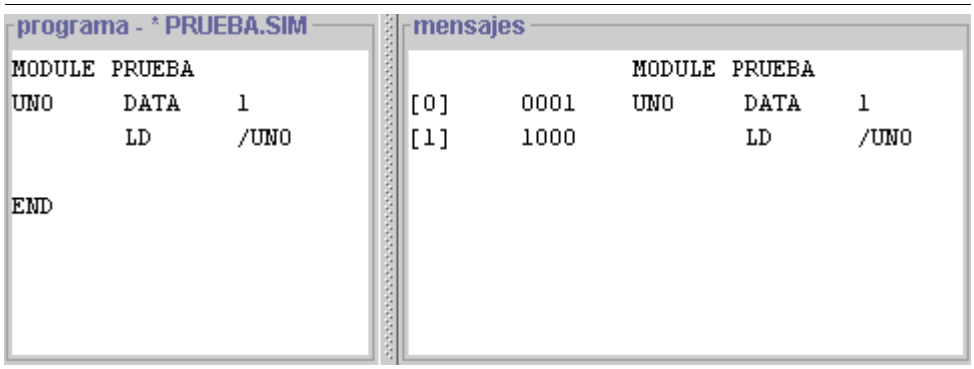


Figura 106: Programa fuente una vez ensamblado.

Como ejemplo se utilizará el editor para introducir el Programa 2, sacado de [Fernández,98], y que suma los diez primeros términos de la sucesión de Fibonacci: t(0)=0; t(1)=1; t(n)=t(n−1)+t(n−2),n>1

Para introducirlo en el editor se puede utilizar el área de edición, o utilizar cualquier otro editor de texto, salvarlo a disco y luego cargarlo desde el editor de programas. El fichero que almacene el programa fuente debe tener extensión **.sim**.

```
; Programa que suma los diez primeros términos de la
; sucesión de Fibonacci.
MODULE FIBONACC
          BR        /INICIO
UNO       DATA      1           ; Valor 1
OCHO      DATA      8           ; Valor 8
CONT      RES       1           ; Contador

PEN       RES       1           ; Diferentes
ULT       RES       1           ; términos de
SIG       RES       1           ; la sucesión
SUM       RES       1           ; Suma de ellos
INICIO    CLR                   ; AC a 0
          ST        /PEN        ; a PEN
          LD        /UNO        ; AC a 1
          ST        /ULT        ; a ULT
          ST        /SUM        ; a SUM
          LD        /OCHO       ; AC a 8
          ST        /CONT       ; a CONT
BUCLE     LD        /PEN        ; PEN a AC
          ADD       /ULT        ; PEN+ULT
          ST        /SIG        ; a SIG
          LD        /SUM        ; SUM a AC
          ADD       /SIG        ; SIG+SUM
          ST        /SUM        ; a SUM
          LD        /ULT        ; ULT a AC
          ST        /PEN        ; a PEN
          LD        /SIG        ; SIG a AC
          ST        /ULT        ; a ULT
          LD        /CONT       ; CONT a AC
          DEC                   ; lo decrementa
          BZ        /FIN        ; si cero, sale
                                ; del bucle
          ST        /CONT       ; si no,guardarlo
          BR        /BUCLE      ; vuelve a bucle
FIN       HALT
          END
```

Programa 2: Suma de los 10 primeros términos de la sucesión de Fibonacci.

Nota− Si se utiliza el área de edición, una vez escrito el programa fuente se debe salvar a un fichero antes de ensamblarlo.

Esto es necesario debido a que el Editor de programas necesita un nombre de referencia para utilizar en los ficheros que genera. Si tras introducir el programa fuente en el área de edición no se salva a disco y se pulsa el botón de ensamblado, no se producirá ninguna acción y no se mostrará nada en el área de mensajes.

Una vez cargado el programa y salvado a un fichero el aspecto del editor será el de la Figura 107.

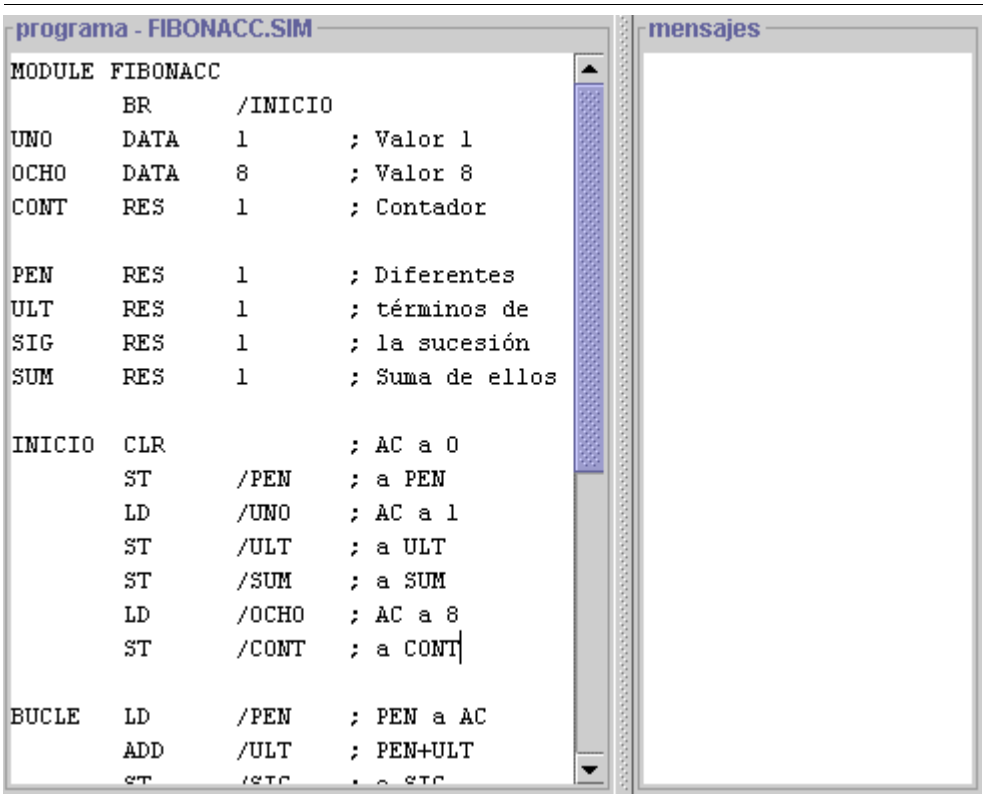


Figura 107: Programa FIBONACCI cargado en el editor de programas.

El siguiente paso es **ensamblarlo** pulsando el botón correspondiente de la barra de herramientas. Esta operación genera dos ficheros con el mismo nombre que el fichero que contiene el código fuente, pero con distinta extensión. El primer fichero, de extensión **.ens**, es un fichero de texto que añade al código fuente la traducción octal del programa y las posiciones que van a ocupar en memoria las instrucciones y seudoinstrucciones. El segundo fichero, de extensión **.obs**, contiene el código objeto. Si el programa contiene algún error, el ensamblado se interrumpe y no se generan los ficheros.

Además de generarse dichos ficheros, en el área de mensajes aparece el código máquina del programa en notación octal acompañado de las direcciones de memoria que ocupará cada instrucción y el código ensamblador correspondiente.

En la Figura 108 se muestra el resultado de la ensamblación del programa que suma los diez primeros términos de la serie de Fibonacci.

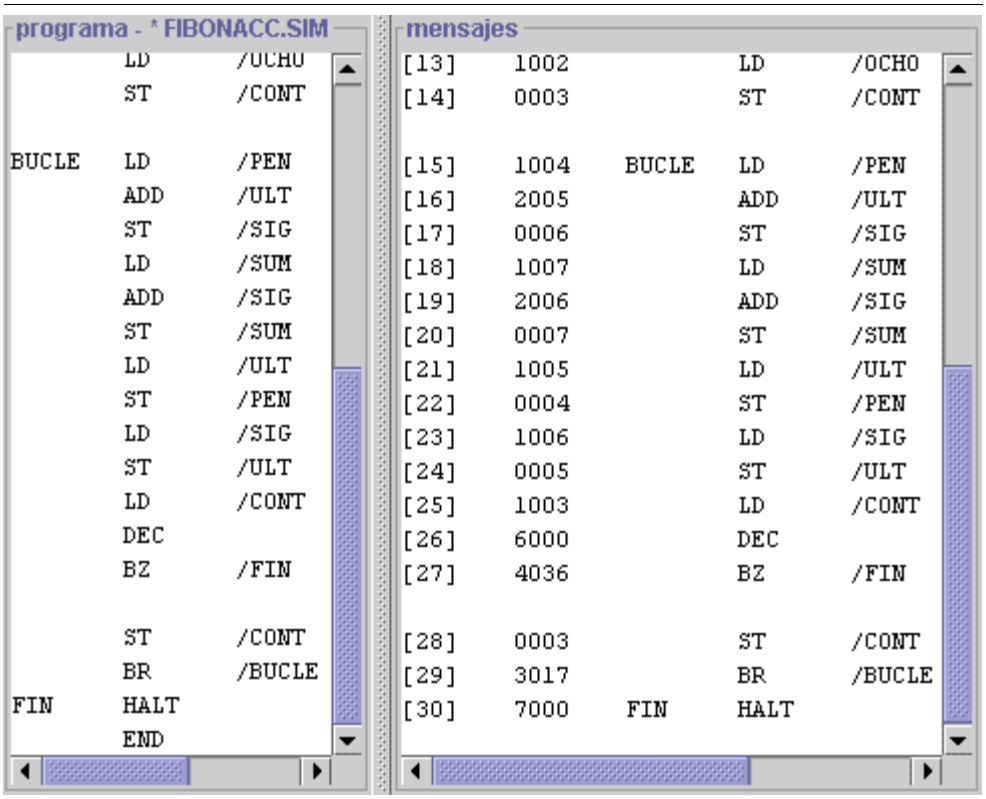


Figura 108: Programa FIBONACCI ensamblado.

Una vez ensamblado, el último paso es **montar** el programa pulsando el botón correspondiente de la barra de herramientas. Dicha operación genera un fichero de extensión **.lns**, que se puede cargar en la memoria de Símplez, tal y como se explica en el Capítulo 2.

Operaciones de manejo de texto

Las operaciones asociadas a teclas, o combinaciones de ellas, disponibles en el área de edición son las mostradas en la siguiente tabla.

Tecla	**Operación**
Tabulador	Avanza el cursor ocho posiciones respecto a la actual.
Retorno Carro	Termina la línea actual, crea una nueva y sitúa el cursor al comienzo de la misma.
Inicio	Sitúa el cursor al comienzo de la línea actual.
Fin	Sitúa el cursor al final de la línea actual.
Flecha arriba	Sitúa el cursor en la línea precedente, si existe.
Flecha abajo	Sitúa el cursor en la línea siguiente, si existe.
Flecha derecha	Sitúa el cursor una posición a la derecha de la actual, si existe.
Flecha izquierda	Sitúa el cursor una posición a la izquierda de la actual, si existe.
Supr	Borra el carácter situado a la derecha del cursor.
Retroceso	Borra el carácter situado a la izquierda del cursor.
CTRL + A	Selecciona todo el texto presente en el área de edición.
CRTL + C	Copia el texto seleccionado al portapapeles.
CRTL + V	Pega en el área de edición el texto presente en el portapapeles.
CRTL + X	Corta el texto seleccionado al portapapeles.

Tabla 8: Operaciones de manejo de texto del editor y teclas asociadas.

Lenguaje ensamblador

El lenguaje ensamblador es un lenguaje simbólico que permite escribir programas evitando en la medida de lo posible el manejo de números, tanto datos o códigos de instrucción como posiciones de memoria. Lo que da como resultado un lenguaje más cercano al humano y no tan críptico como el lenguaje máquina, y que evita así parte de los errores producidos durante la creación de un programa.

El lenguaje ensamblador se compone de instrucciones, seudoinstrucciones, directivas y etiquetas. Pero además, para que un programa ensamblador pueda interpretar el lenguaje simbólico es necesario que éste cumpla una serie de reglas. Este conjunto de reglas y la especificación completa de los elementos anteriores, definen la sintaxis del lenguaje ensamblador.

El modelo funcional de Símplez, descrito en la página 126, junto con la sintaxis del lenguaje ensamblador permiten escribir cualquier programa usando el editor.

Sintaxis del lenguaje ensamblador

La sintaxis del lenguaje ensamblador manejado por el editor de Símplez es la siguiente:

- El código fuente ha de comenzar por la directiva **MODULE**.
- Así mismo ha de terminar por la directiva **END**.
- Cada instrucción, seudoinstrucción o directiva definen una sentencia del lenguaje ensamblador.
- Cada sentencia se escribe en una línea, terminada por un retorno de carro.
- Cada sentencia se compone de varias partes, separadas por al menos un espacio en blanco, formando una estructura fija.
- En las palabras reservadas sólo se admiten las letras mayúsculas. En las etiquetas sólo se admiten mayúsculas y determinados caracteres.

Etiquetas

Una etiqueta es un nombre, cadena de caracteres, que sustituye a una dirección de memoria. Esta dirección será la de la posición que ocupe en memoria la instrucción a la que acompaña. La etiqueta puede ser utilizada en las instrucciones del programa para hacer referencia a dicha dirección. Dado que la memoria de Símplez tiene 512 palabras, el rango de una etiqueta va de 0 a 511.

Una etiqueta ha de cumplir las siguientes normas:

- Su primer carácter ha de ser una letra mayúscula.
- El resto de caracteres pueden ser letras mayúsculas, números o el carácter "_".
- La longitud máxima de una etiqueta es de 12 caracteres.
- Delante de una etiqueta no puede haber espacios en blanco.

La Figura 109 muestra un ejemplo de utilización de etiquetas.

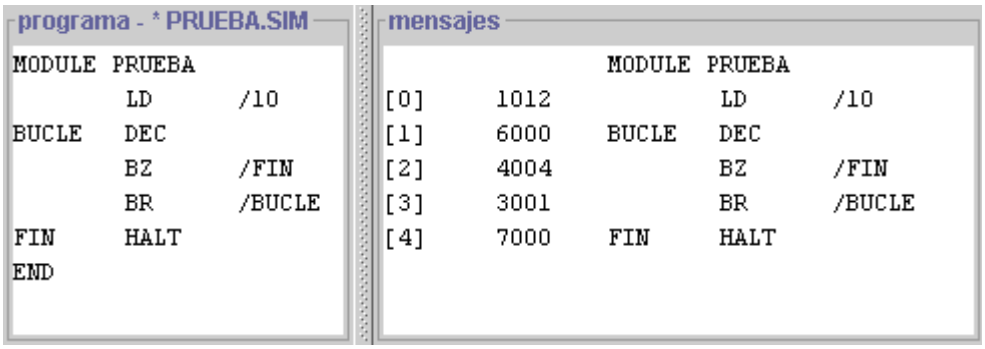


Figura 109: Utilización de las etiquetas.

En el ejemplo se han definidos dos etiquetas, "FIN" y "BUCLE". Una vez ensamblado el programa se puede observar que "FIN" tiene el valor "4" y "BUCLE" el valor "1". Las instrucciones, tal y como se verá a continuación, pueden hacer uso de las etiquetas en el campo de dirección. En el ejemplo las instrucciones '`BZ /FIN`' y '`BR /BUCLE`' son equivalentes a '`BZ /4`' y '`BR /1`'. Si se realiza algún cambio en el código fuente y se vuelve a ensamblar, se observaría que las etiquetas pueden haber cambiado. Una de las ventajas más interesantes del uso de etiquetas es que libera de la obligación de conocer cuál será la posición final de una instrucción al hacer referencia a ella.

Instrucciones

Las estructuras posibles de una instrucción en el lenguaje ensamblador son las mostradas en la Figura 110.

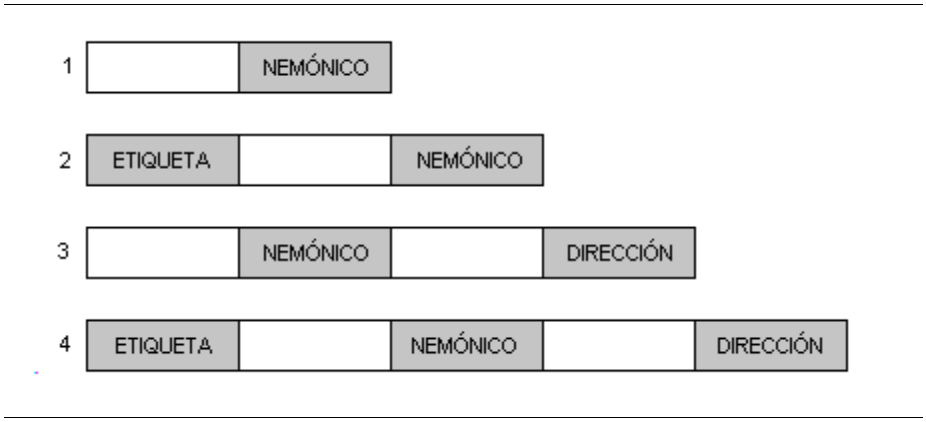


Figura 110: Estructuras de las instrucciones en lenguaje ensamblador.

Los **nemónicos** son secuencias de letras mayúsculas que representan a las instrucciones del repertorio. Las secuencias válidas son "ST", "LD", "ADD", "BR", "BZ", "CLR", "DEC" y "HALT".

Las instrucciones deben ir siempre precedidas de al menos un espacio en blanco, lleven o no una etiqueta delante. Si no es así, se producirá un error en ensamblación. Las etiquetas no pueden ir precedidas de ningún espacio en blanco.

Las líneas uno y dos se corresponden con instrucciones que no hacen referencia a memoria: CLR, DEC y HALT.

Las líneas tres y cuatro representan al resto de instrucciones: ST, LD, ADD, BR y BZ. El campo **dirección** especifica la dirección del operando en memoria precedida del carácter "/". La dirección puede estar representada por una etiqueta o ir indicada explícitamente por un número entero entre 0 y 511. La única condición para el uso de una etiqueta en el campo de dirección, es que ésta esté definida en una línea anterior o posterior. En la Figura 109 se podían observar los ejemplos "`LD /10`" y "`BZ /FIN`".

Seudoinstrucciones

Una seudoinstrucción es una información dirigida al programa ensamblador indicándole que haga algo en una dirección de memoria, pero que no se traducirá por una instrucción de lenguaje máquina, al no pertenecer la seudoinstrucción al repertorio de instrucciones. El ensamblador de Símplez dispone de dos seudoinstrucciones, **DATA** y **RES**, cuyas estructuras posibles se muestran en la Figura 111.

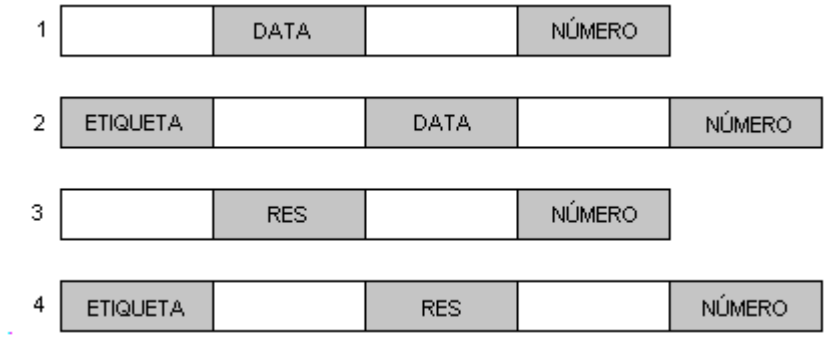


Figura 111: Seudoinstrucciones del lenguaje ensamblador de Símplez.

La seudoinstrucción **RES** permite indicarle al programa ensamblador que reserve tantas posiciones de memoria como indica el número entero que le sigue. Dicho número puede estar comprendido entre 1 y 512. Se suele utilizar para reservar zonas de memoria que en ejecución albergarán datos. Debe ir precedida de al menos un espacio en blanco, y se le puede asociar una etiqueta. Esta etiqueta permite hacer referencia desde las instrucciones del programa a las posiciones reservadas.

Analizando el ejemplo de la Figura 112, se puede observar que "RES 3" reservará tres posiciones de memoria desde el punto del programa donde se encuentra.

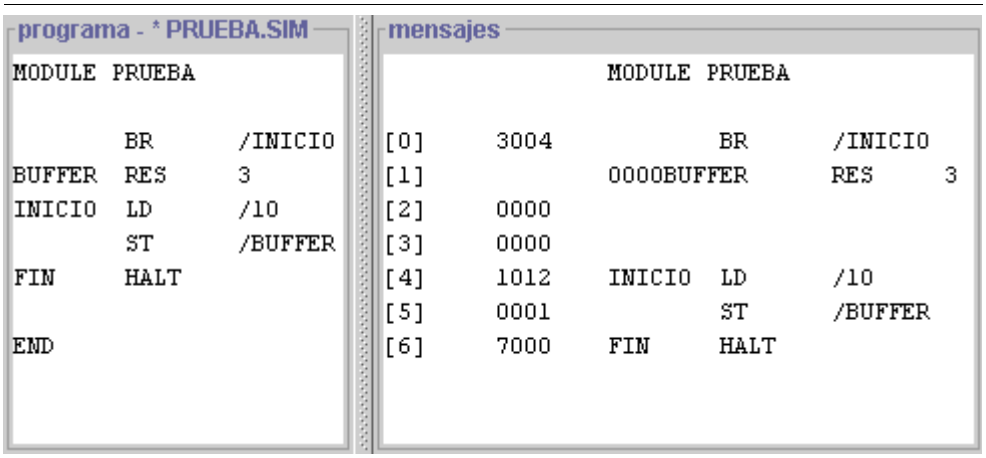


Figura 112: Utilización de la seudoinstrucción RES.

Al haber una única instrucción delante, dichas posiciones serán la 1, la 2 y la 3. Posteriormente se utiliza la etiqueta que tienen asociada, "BUFFER", para almacenar el contenido de la posición 10 de memoria en la primera de ellas.

La seudoinstrucción **DATA** le indica al programa ensamblador que, en la dirección correspondiente a la posición que ocupa, almacene el valor indicado por el número entero que le sigue. El rango de valores de este número va de 0 a 4095. Al igual que en el caso de RES, DATA siempre ha de ir precedida de al menos un espacio en blanco y puede ir acompañada de una etiqueta.

Esta seudoinstrucción suele utilizarse para guardar constantes en memoria, para inicializar una variable, como puede ser el contador de un bucle, a un valor determinado, etc. Si se ha definido una etiqueta, se puede hacer referencia a dicho valor a través de ella. Un ejemplo es el programa de la Figura 113, y la etiqueta "UNO".

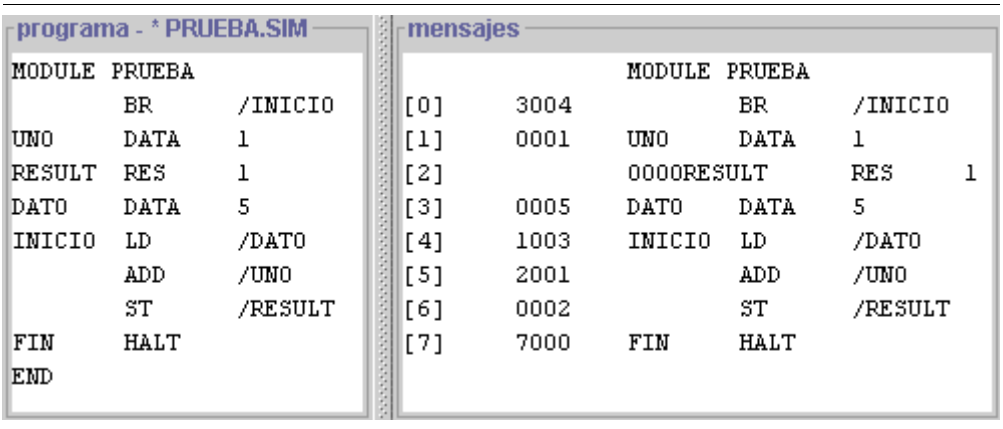


Figura 113: Utilización de la seudoinstrucción DATA.

Si se quieren salvar varios valores en posiciones consecutivas de memoria, basta con utilizar una única seudoinstrucción DATA y poner a continuación los valores separados por comas, tal y como muestra la Figura 114.

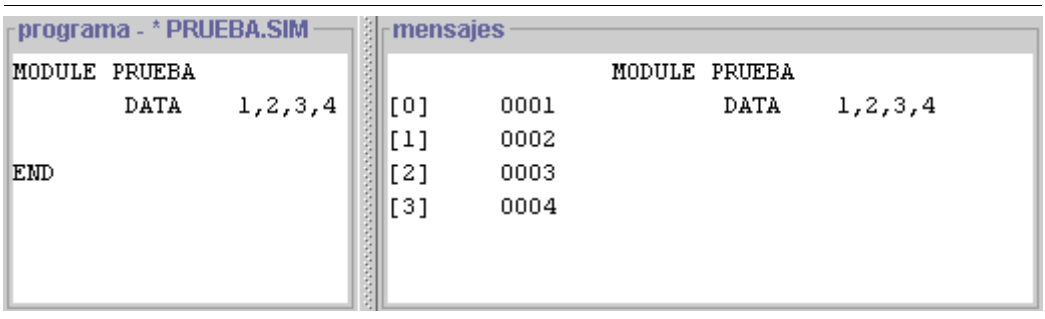


Figura 114: Definición con DATA de valores en posiciones consecutivas.

El valor numérico que la seudoinstrucción DATA debe almacenar se puede introducir en los formatos numéricos decimal, octal, hexadecimal y binario. Para ello basta con utilizar la notación:

DATA X'VALOR

Siendo X' la inicial en mayúsculas del formato en cuestión, seguida del carácter "'". Los valores que por tanto puede adoptar son: "D'" decimal, "O'" octal, "H'" hexadecimal y "B'" binario. Por defecto la ausencia de este modificador implica valor decimal. La Figura 115 muestra algunos ejemplos de números introducidos utilizando diferentes formatos.

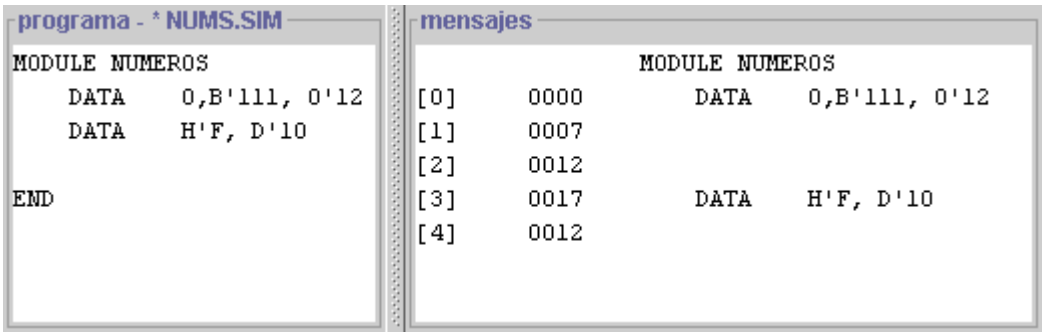


Figura 115: Formatos numéricos manejados por DATA.

Directivas

Las directivas son informaciones para el programa ensamblador, indicándole cómo organizar el código. A diferencia de las seudoinstrucciones no implican ninguna operación sobre una dirección de memoria. El ensamblador de Símplez maneja tres directivas; **MODULE**, **EQU** y **END**, mostradas en la Figura 116.

Figura 116: Directivas del lenguaje ensamblador.

La directiva **MODULE** es obligatoria e indica dónde comienza el código fuente. Puede ir precedida de espacios en blanco y siempre debe ir acompañada de un nombre separado por al menos un espacio en blanco de ella. La cadena de caracteres que forme el nombre debe cumplir las siguientes reglas:

- Su primer carácter ha de ser una letra mayúscula.
- El resto de caracteres pueden ser letras mayúsculas, números o el carácter "_".
- La longitud máxima de una etiqueta es de 12 caracteres.

La directiva **END** es obligatoria e indica dónde termina el código fuente. Le indica al programa ensamblador que deje de procesar en esa línea, por lo que cualquier cosa escrita a continuación de ella se descarta. Puede ir precedida de espacios en blanco.

La directiva **EQU** sirve para definir valores de etiquetas, asociando a la etiqueta que la precede el valor que la sucede. Por tanto siempre ha de ir acompañada de una etiqueta. Es equivalente a definir una constante que será utilizada en otras partes del programa. La diferencia con una pareja etiqueta y seudoinstrucción DATA, es que en el caso de estas últimas su utilización se traduce en el almacenamiento en una posición de memoria del valor indicado por la seudoinstrucción y la asignación a la etiqueta de dicha dirección. En cambio el uso de EQU no implica almacenamiento alguno y únicamente se trata de una constante manejada durante el periodo de ensamblación.

La Figura 117 muestra un ejemplo de uso de la directiva EQU.

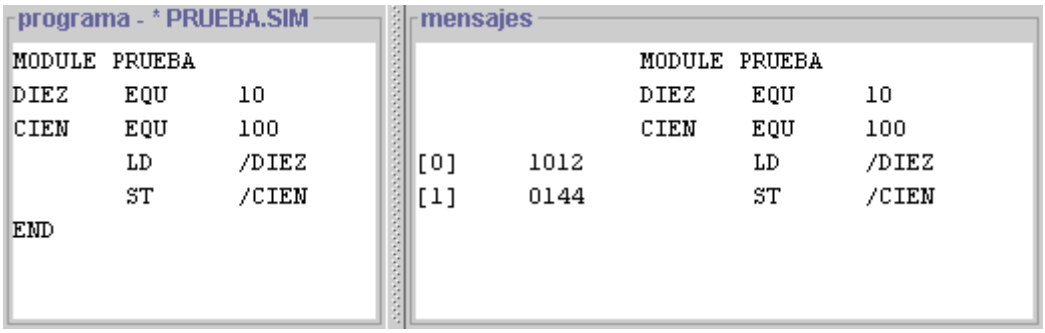


Figura 117: Utilización de la directiva EQU.

La instrucción "`LD /DIEZ`" implica la carga en el acumulador del contenido de la posición de memoria 10, y "`ST /CIEN`" almacena en la posición 100 de memoria dicho valor.

Comentarios

El programa ensamblador permite también poner comentarios. Estos ayudan a organizar un programa y explicar qué se está haciendo en determinado sitio. Un punto y coma identifica el comienzo de un comentario, y el resto de la línea donde aparece se interpreta como tal.

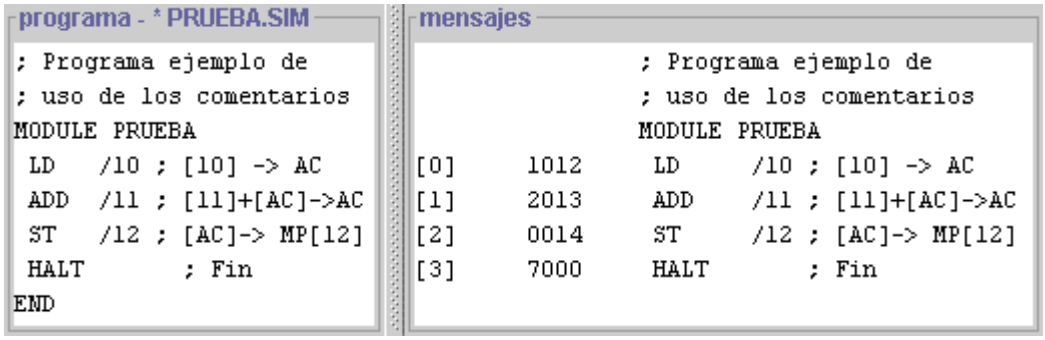


Figura 118: Uso de los comentarios en un programa fuente.

Aritmética de etiquetas y direcciones

Se puede utilizar cierta aritmética a la hora de manejar y definir etiquetas. Las operaciones permitidas están definidas por las siguientes reglas:

1. Las operaciones permitidas son la suma y la resta.

2. Se pueden utilizar los paréntesis para agrupar operandos.

3. Los operandos pueden ser números enteros y etiquetas.

4. Los operandos numéricos pueden expresarse en cualquiera de los formatos decimal, octal, hexadecimal o binario.

5. El resultado final de las operaciones ha de estar comprendido entre 0 y 512.

6. Las etiquetas definidas con EQU se denominan **absolutas**. El resto de etiquetas se denominan **reubicables**, dado que hasta que no se evalúen no se conoce su valor.

7. La únicas operaciones no permitidas entre etiquetas son la suma de dos etiquetas reubicables y la resta de una etiqueta reubicable a una absoluta.

8. Las seudoinstrucciones no admiten operaciones en su campo numérico. Sí admiten en cambio etiquetas resultado de una operación. La única condición es que la etiqueta esté definida antes de su uso.

9. La directiva EQU admite operaciones en su campo numérico.

10. Las instrucciones admiten operaciones en su campo de dirección.

La Figura 119 muestra ejemplos de operaciones realizadas con etiquetas.

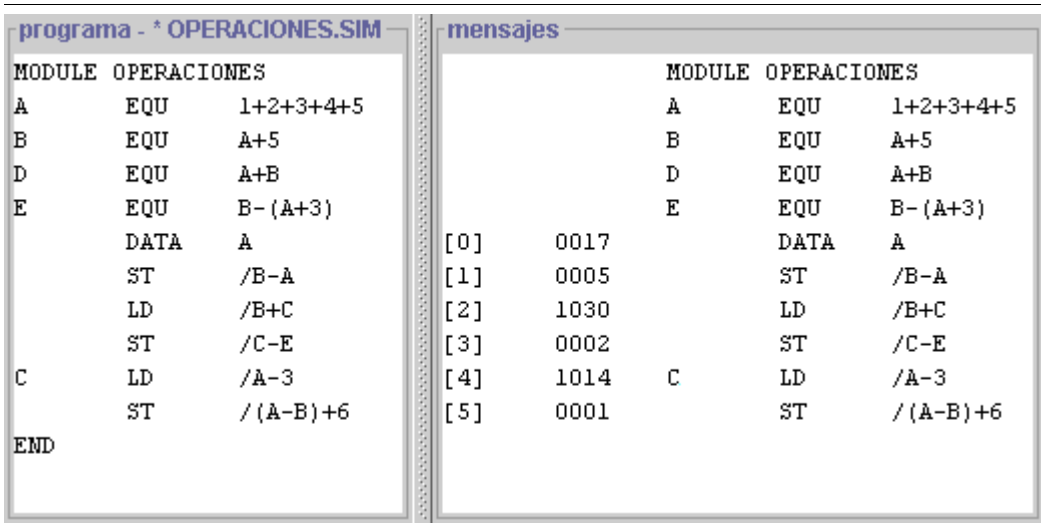


Figura 119: Ejemplos de operaciones realizadas con etiquetas.

Las posibilidades de utilización de estas operaciones son múltiples. Un ejemplo es el mostrado en la Figura 120.

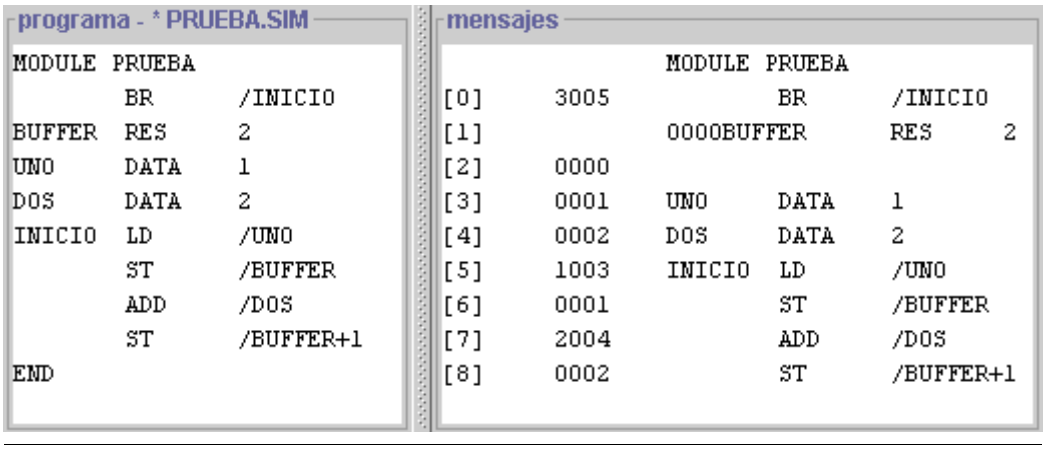


Figura 120: Ejemplo de acceso a memoria utilizando una operación con etiquetas.

En él se han reservado un par de direcciones bajo la etiqueta "BUFFER". Para acceder a ambas basta con utilizar en los campos de dirección los valores "BUFFER" y "BUFFER+1". Por tanto, definiendo una etiqueta al principio de una zona reservada, luego se puede hacer referencia a las diferentes posiciones con solo ir incrementando la etiqueta.

Por último hacer mención de la utilización de etiquetas en lugar del número entero en la seudoinstrucción RES, Figura 121.

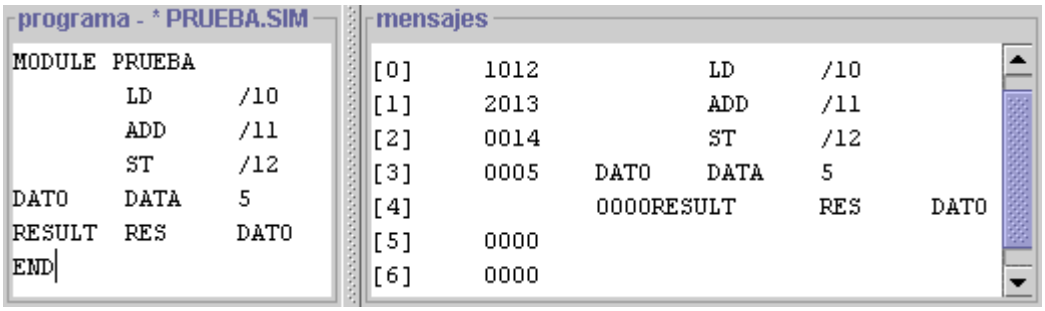


Figura 121: Utilización de RES con una etiqueta.

En este caso se reservarán tantas posiciones de memoria como el valor de dirección indicado por la etiqueta. En la figura vemos que la etiqueta "DATO" hace referencia a la dirección de memoria 3, que contiene la constante 5. Al ensamblar se puede comprobar que se han reservado 3 posiciones, valor de la etiqueta "DATO", y no 5, que es el valor que contiene la posición de memoria apuntada por la etiqueta.

Ensamblado y ejecución de programas

A continuación se revisa cuál es la secuencia de operaciones necesaria para ejecutar un programa escrito en un lenguaje fuente, para luego concretarlo en el caso del simulador de Símplez.

El objetivo es siempre el mismo; partiendo de un **programa fuente**, la idea es traducirlo a su equivalente en lenguaje máquina. Es decir al conjunto de palabras binarias que la máquina es capaz de interpretar y ejecutar. Al programa en lenguaje máquina resultante se le suele denominar **código objeto**.

Para realizar dicha traducción es necesario un **programa traductor** que procese el programa fuente y lo vaya convirtiendo a lenguaje máquina. Según sea el código fuente empleado se necesitará un traductor u otro. Situándose al nivel de lenguajes de alto nivel, como C o C++, a este traductor se le llama **compilador**. En el simulador se manejan programas escritos en lenguaje ensamblador, a cuyo traductor se le llama programa **ensamblador**.

Los pasos necesarios para poder ejecutar un programa fuente, teniendo en cuenta que para poder ejecutar un programa éste debe cargarse previamente en memoria, son los siguientes:

1. El sistema operativo carga en memoria el programa ensamblador.

2. Se ejecuta el programa ensamblador. El cual toma como datos el programa fuente y genera como salida el código objeto.

3. El sistema operativo carga en memoria el código objeto generado.

4. Se ejecuta el código objeto.

En el proceso descrito, el ensamblador genera un código objeto interpretable por la máquina y el sistema operativo en cuestión. Si se quisiera ejecutar el programa fuente en otra máquina con otra arquitectura, o en la misma máquina pero con otro sistema operativo, probablemente no bastaría con mover el código objeto de una a otra. Sería necesario repetir todo el proceso, pero utilizando esta vez un nuevo programa ensamblador adaptado a la nueva arquitectura que genere las instrucciones binarias que ésta interpreta.

En el caso del programa ensamblador del simulador ocurre algo parecido. Tras ensamblar el programa fuente se obtiene un código objeto ejecutable por Símplez, pero no por la máquina en que se está ejecutando el editor de programas, probablemente un PC con Windows. Por ello, o se utiliza un ensamblador que lo traduzca a dicha arquitectura, o se utiliza un **simulador** que permita ejecutar el código objeto como si se estuviera ejecutando sobre Símplez. Es decir, el simulador se comporta como una **máquina virtual** de Símplez.

Por tanto, para el caso de Símplez, el proceso para lograr ejecutar el programa fuente escrito en ensamblador es el mostrado en la Figura 122, donde hay dos pasos adicionales.

A) Con un editor se genera el programa fuente, fichero con extensión **.SIM**, y carga en memoria el programa ensamblador.

B) Se ejecuta el ensamblador, el cual genera el código objeto y lo almacena en disco en un fichero de extensión **.LNS**.

C) Una vez ensamblado el programa, el sistema operativo carga en memoria el programa simulador, y lo ejecuta. El programa ensamblador ya no es necesario y se puede cerrar.

D) El simulador carga en memoria el código objeto y lo ejecuta.

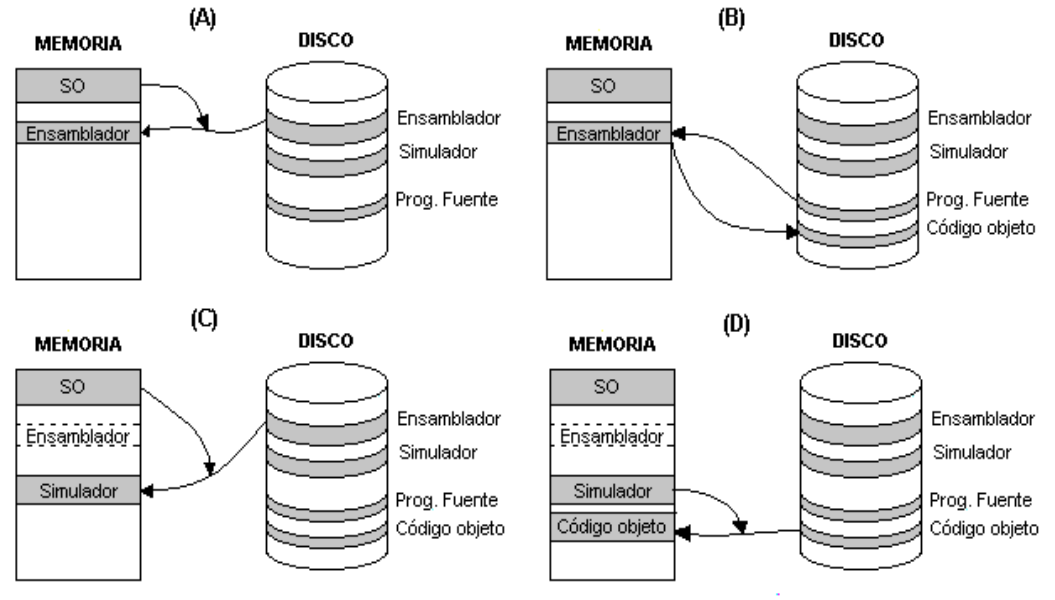


Figura 122: Proceso de ejecución del programa fuente.

En el caso particular del simulador descrito en este manual hay que hacer dos puntualizaciones. El entorno de edición, ensamblación y simulación están integrados, por lo que una vez que se arranque el simulador, tal y como se explica en el Prefacio, todas estas herramientas estarán cargadas ya en memoria y el paso (C) no es necesario.

En segundo lugar el simulador está escrito en Java, cuyo código compilado no es ejecutable directamente en ningún sistema operativo. La compilación de un programa escrito en Java genera un código denominado **código de bytes**. Este código necesita un interprete, denominado **máquina virtual de Java**, que permita ejecutarlo.

De esta manera el simulador escrito en Java, es compilado a este código de bytes. El resultado es un simulador independiente de la plataforma donde se vaya a ejecutar, y es el interprete o máquina virtual el que es dependiente de la misma. Esto permite llevarse el simulador de una máquina a otra, donde haya instalada una máquina virtual de Java, y ejecutarlo independientemente de la plataforma de la máquina destino sin necesidad de recompilarlo.

Haciendo un analogía, se puede aplicar la Figura 122 para ilustrar este proceso. Sólo hay que sustituir la palabra ensamblador por compilador de Java, y simulador por máquina virtual de Java. El código objeto de la figura sería el simulador de Símplez. Cuando se quiera ejecutarlo, paso (D), se necesita arrancar la máquina virtual indicándole que ejecute el simulador, tal y como se explicó en el Prefacio. Una vez arrancado el simulador se completaría la figura con los pasos (B), para el caso de ensamblación de un programa de Símplez, y (D), para el caso de ejecución de un programa de Símplez.

7.2 Editor de Microprogramas

Esta herramienta permite crear Memorias de Control para el secuenciador microprogramado de una manera sencilla.

Para arrancar el editor de Microprogramas basta con seleccionar la entrada "**Editor de microprogramas**" del menú **Herramientas**, y aparecerá la ventana mostrada en la Figura 123.

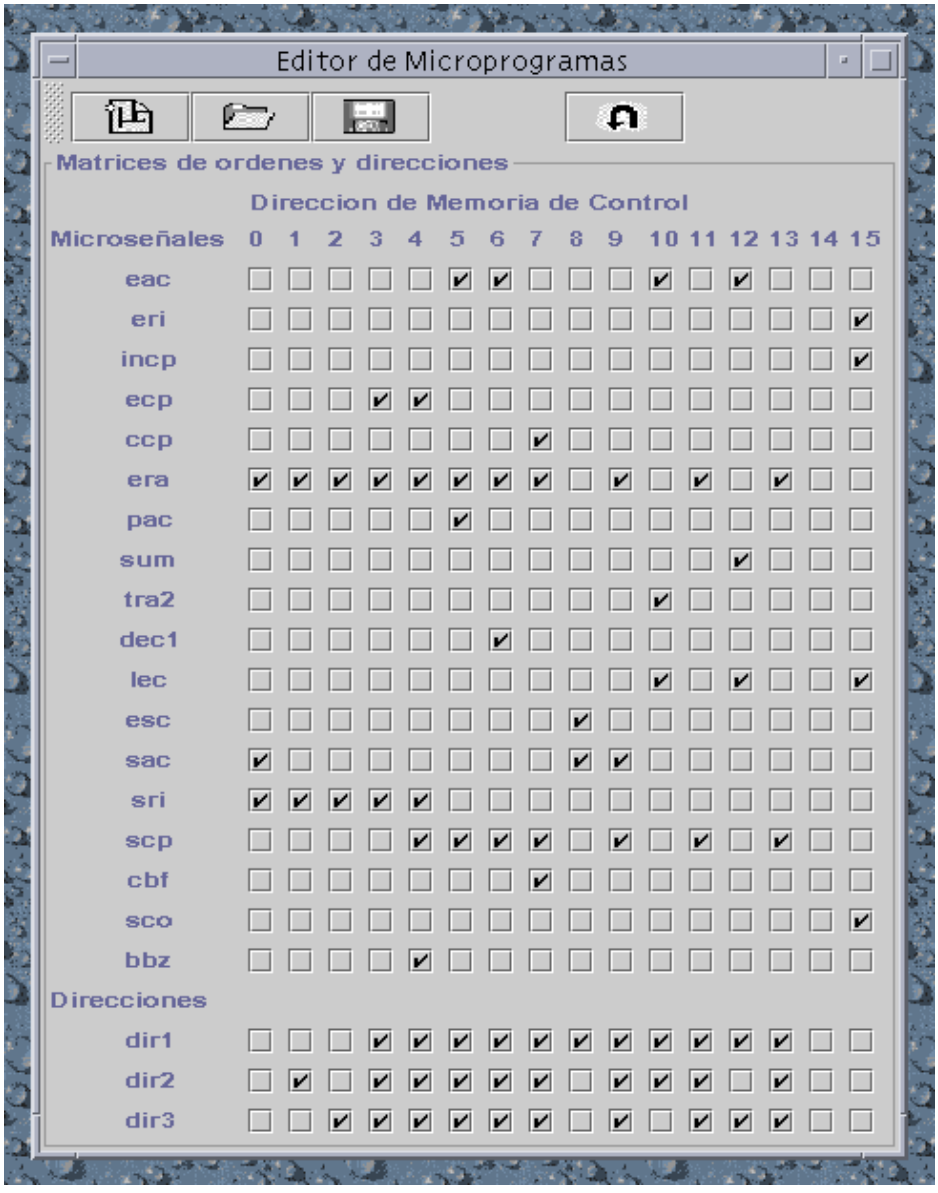


Figura 123: Editor de Microprogramas.

Como se puede apreciar, debajo de la barra de herramientas hay dos matrices de botones: la **matriz de ordenes** y la **matriz de direcciones**. Tal y como se vio en el Capítulo 4, la estructura de la Memoria de Control es la mostrada en la Figura 124

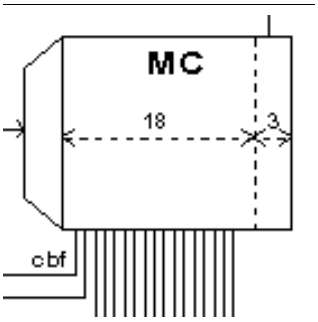


Figura 124: Estructura de la Memoria de Control.

La parte de la Memoria de Control de la que salen las líneas paralelas, y que tiene un tamaño señalado de 18 bits, es la que se corresponde con la **matriz de ordenes** del editor de microprogramas. Los 18 bits se corresponden con cada una de las microórdenes de control de la Ruta de Datos.

La parte de la derecha, de 3 bits de tamaño y de la que sale el bus DB que conecta con el multiplexor MPX, es la **matriz de direcciones**.

La Figura 125 muestra la Memoria de Control interpretada desde el punto de vista del editor de microprogramas.

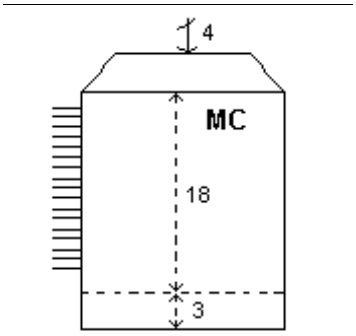


Figura 125: Memoria de Control vista desde el editor.

Como se puede observar, a los circuitos de direccionamiento llegan 4 bits, lo que permite direccionar 16 posiciones o, lo que es lo mismo, 16 microinstrucciones. Por tanto en el editor tendremos que cada columna de botones es una microinstrucción y el número que tiene encima es la dirección que ocupa dentro de la Memoria de Control.

Para hacer que una microorden esté activa dentro de una microinstrucción determinada, únicamente hay que pulsar el botón de la fila y columna correspondientes, haciendo que éste tenga el aspecto de la Figura 126. En caso contrario tendrá valor "0".

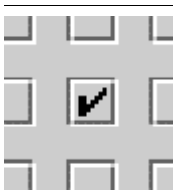


Figura 126: Microorden activa.

El comportamiento de los bits de la **matriz de dirección** es el mismo, y que uno de los bits esté marcado significará que está a uno. Los bits de la matriz de dirección dir1, dir2 y dir3 se corresponden respectivamente con los bits cero, uno y dos de los circuitos de direccionamiento de la Memoria de Control. Ésta se compone de 16 posiciones direccionables mediante 4 bits. La matriz de direcciones únicamente tiene 3 bits, los de menor peso, por lo que con ella sólo se direccionan 8 posiciones. Tal y como se explicaba en la página 65 es a través de la microorden **sco** que se puede direccionar toda la memoria, direccionando alternativamente las ocho posiciones más bajas o las ocho más altas según esté activa o no **sco**. El proceso es el siguiente; cuando **sco** esté activa, el primer bit de dirección que llega a los circuitos de direccionamiento estará a cero gracias a una puerta lógica NOT, y el resto de la dirección se toma del **bus D** gracias a un multiplexor controlado por **sco**. Cuando **sco** esté a cero, la puerta lógica NOT entrega a los circuitos de direccionamiento un uno, por lo que esta vez se direccionarán las ocho posiciones más altas de la Memoria de Control y gracias al multiplexor el resto de bits de dirección son los pertenecientes a la matriz de dirección de la Memoria de Control.

Resumiendo, con la matriz de dirección se direccionan las ocho posiciones más altas de la Memoria de Control siempre y cuando **sco** no esté activa. La Figura 127 muestra un ejemplo de direccionamiento.

Figura 127: Ejemplo de direccionamiento en MC.

La primera columna de botones se corresponde con la microinstrucción de dirección 0, la segunda con la de dirección 1 y así sucesivamente. Tal y como se puede apreciar la microinstrucción de dirección 0 tiene todos sus bits de dirección y **sco** a 0, por lo que la siguiente microinstrucción a ejecutar será la de posición 8. La microinstrucción de dirección 1 tiene su bit dir2 a uno, por lo que la siguiente microinstrucción a ejecutar será la de dirección 10. Por último la microinstrucción de dirección 2 apuntaría como siguiente microinstrucción a la de dirección 12, pero al estar activa **sco** lo único que se sabe de la siguiente microinstrucción es que estará entre las ocho primeras de la Memoria de Control. Como se puede apreciar, cuando **sco** está activa el contenido de los bits de la matriz de direcciones es irrelevante.

Cuando se arranca la herramienta todas las microórdenes están a "0". Para crear una Memoria de Control se pueden meter uno a uno todos los valores de las microórdenes para las diferentes microinstrucciones o partir de una Memoria de Control ya existente. Si se quiere partir de la Memoria de Control que incluye por defecto el simulador para el secuenciador microprogramado, basta con pulsar el botón de la derecha de la barra de herramientas y los contenidos de ésta se cargarán en el editor.

También se puede cargar una Memoria de Control salvada a disco con anterioridad. Para ello se ha de pulsar el botón correspondiente de la barra de herramientas y luego seleccionar el fichero, con extensión **.MCS**, que la contiene.

Una vez se haya terminado la edición de una Memoria de Control, es necesario salvarla a disco para poder cargarla desde el simulador. Para ello basta con pulsar el tercer botón de la barra de herramientas. Una vez que está salvada, para cargar la Memoria de Control, desde el menú de **Archivo** de la ventana principal del simulador seleccionar la entrada **Cargar Memoria de Control**, o pulsar directamente sobre la ventana las teclas **CONTROL M**, y se abrirá la ventana de selección de ficheros, desde donde se puede seleccionar el fichero de extensión **.MCS**. Tras cargar la Memoria de Control su nombre aparecerá en el selector de la ventana principal.

Nota− Cada vez que se edite la Memoria de Control y se salve a disco hay que cargarla de nuevo desde el menú.

Una vez se haya terminado de editar una Memoria de Control, se pueden limpiar las matrices pulsando el botón de la izquierda de la barra de herramientas.

Como se puede observar el manejo del editor es muy sencillo. Supóngase por ejemplo que se quiere que la microorden **eac** esté activa en todos los ciclos de reloj, independientemente de la utilidad que esto pueda tener. Cada ciclo de reloj se corresponde con una microinstrucción de la Memoria de Control, por lo que se debería marcar **eac** en todas las columnas del editor tal y como muestra la Figura 128.

Figura 128: MC con **eac** activa en todas la microinstrucciones.

Capítulo 8

Programación de Símplez

En los capítulos anteriores se ha explicado en detalle la utilización del simulador y se han introducido como ejemplo un par de programas. La finalidad última del simulador es la ejecución de programas escritos para Símplez. Quedaría por tanto explicar cómo programar el ordenador Símplez.

Dado que la programación de Símplez se cubre ampliamente en [Fernández, 98] y que la finalidad del manual de usuario no es explicar metodología de programación, este capítulo sólo se centra en cómo programar Símplez desde el punto de vista del simulador. Para ello primero se explicarán los conceptos de Símplez necesarios y luego se desarrollarán dos ejemplos, con los que se explicarán las dos formas de programar Símplez utilizando el simulador. Esto también valdrá para revisar conceptos vistos a lo largo de los capítulos anteriores.

8.1 Niveles de abstracción

La programación se realiza a nivel de máquina convencional, es decir para un sistema basado en una **máquina programable binaria que interpreta las instrucciones** que componen los programas. Se habla de **máquina binaria** porque en ella todo se codifica en código binario. La **interpretación** que de ese código binario hace la máquina está definida por el **modelo funcional**, el cual es necesario entender para poder programarla adecuadamente y que define el **lenguaje máquina** de la misma. Con este lenguaje se pueden escribir los programas a ejecutar por la máquina.

El programador podría, entendiendo el modelo funcional, programar Símplez escribiendo los códigos binarios del lenguaje máquina a introducir en memoria. Pero dado la dificultad que esto plantea, se define un código nemónico, que es una representación simbólica de los código binarios de las instrucciones que la máquina

maneja, de manera que es más fácil leer e interpretar los programas. Para facilitar aún más la tarea se puede definir un **lenguaje ensamblador** de manera que una vez escrito el programa se puede utilizar un programa ensamblador que interprete dicho código nemónico y que genere los códigos binarios.

Figura 129: Niveles de abstracción.

8.2 Código nemónico

El código nemónico representa instrucciones. Para ello se compondrá del código de operación seguido, si se trata de una instrucción con acceso a memoria, de la dirección absoluta (es decir dada respecto a la posición cero de la memoria) en decimal y precedida del símbolo "/".

<Código Operación> [/<Código Dirección>]

8.3 Modelo Funcional de Símplez

Para poder programar Símplez a nivel de máquina convencional es fundamental conocer su modelo funcional. El modelo funcional a este nivel se compone de:

- Convenios para la representación de la información dentro de la máquina, tanto datos como instrucciones.

- Repertorio de instrucciones con una explicación detallada de lo que hace cada una.

Convenios para la representación de la información

Los tipos de información manejados por Símplez son números, caracteres y las instrucciones.

Números

En Símplez únicamente se consideran los números enteros no negativos. Estos se almacenarán en una única palabra de la Memoria Principal. Dado que la longitud de la palabra de la MP es de 12 bits el rango de números representables en Símplez va de 0 a 4095.

Dentro de la memoria se usa el convenio de interpretar el bit más a la izquierda de la palabra como el más significativo, y el de más a la derecha como el menos significativo.

Dado que casi todos los elementos que componen la Ruta de Datos de Símplez, excepto los puertos de entrada y salida, utilizan tamaños de palabra múltiplos de tres, la representación octal es ideal para representar los números y hacerlos más legibles que en el código binario. El Simulador incluye este formato en todos los sitios donde se manejan números.

Teniendo en cuenta todos estos aspectos, la Tabla muestra algunas equivalencias entre binario, decimal y octal para el tamaño de palabra manejado en Símplez.

Binario	**Octal**	**Decimal**
000000000000	0000	0
0000000000001	0001	1
000000000010	0002	2
000000000011	0003	3
...	...	...
000000000111	0007	7
000000001000	0010	8
000000001001	0011	9
...	...	...
111111111110	7776	4094
111111111111	7777	4095

Tabla 9: Correspondencia entre formatos para el tamaño de palabra de Símplez.

Caracteres

Tal y como se explicó en el Capítulo 6, la comunicación con el usuario a través de los dispositivos periféricos se realiza mediante cadenas de caracteres. Cada carácter se representa por su codificación en ISO Latin1. Este código utiliza ocho bits para representar cada carácter. En el Capítulo 6 se incluye una descripción del código ISO Latin1.

Instrucciones

Las instrucciones. de Símplez tienen dos campos: el código de operación (CO) y la dirección del operando (CD). Cada instrucción ocupa una palabra de MP. Las palabras de MP tienen 12 bits, de los cuales CD ocupa los bits 0 a 8, y CO ocupa los bits 9, 10 y 11.

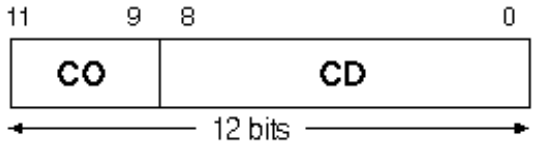


Figura 130: Formato de instrucción de Símplez.

Repertorio de Instrucciones

El repertorio de Símplez se compone de las ocho instrucciones, identificadas por el Código de Operación (CO), mostradas en la Tabla 10.

La primera columna muestra el Código de operación en binario. Éste es el formato manejado por la máquina. La segunda columna muestra su codificación en octal, más natural para nosotros, y directa al tratarse de 3 bits. Este formato es, además, uno de los manejados en la Ruta de Datos del simulador. En la tercera columna se muestra el código nemónico asociado. Este código es el utilizado para representar a las instrucciones en el lenguaje ensamblador y en la tabla de Programa de la ventana principal.

CO (bin.)	CO (oct.)	CO (nem.)	Significado
000	0	ST	Almacena ("STORE") el contenido del acumulador en la palabra de memoria cuya dirección se indica en el campo CD. Es decir, lleva los 12 bits del acumulador a esa palabra, con lo que desaparece de ella lo que tuviera anteriormente (pero permaneciendo en el acumulador).
001	1	LD	Carga ("LOAD") en el acumulador el contenido de la palabra de memoria cuya dirección se indica en el campo CD. Es decir, es la inversa de la anterior: lleva los 12 bits de la palabra al acumulador, borrando lo que éste contuviese previamente.
010	2	ADD	Suma ("ADD") al acumulador el contenido de la palabra de memoria especificada en CD, dejando el resultado en el propio acumulador (y borrando así su contenido previo).
011	3	BR	Bifurca ("BRANCH") incondicionalmente a la dirección indicada en CD: le dice a la unidad de control que la siguiente instrucción a ejecutar es la almacenada en la dirección dada por CD.
100	4	BZ	Bifurca si el resultado de la última operación realizada por la UAL ha sido cero; en caso contrario sigue con la secuencia normal, y la instrucción no hace nada.
101	5	CLR	Pone a cero ("CLEAR") el acumulador (12 ceros binarios).
110	6	DEC	Decrementa en una unidad el contenido del acumulador (y deja el resultado en él).
111	7	HALT	Hace pararse a la máquina, es decir, hace que no se vuelva al "paso 1" en el ciclo que realiza la UC.

Tabla 10: Repertorio de instrucciones de Símplez.

8.4 Almacenado de Programas

Los computadores se construyen sobre dos principios:

1. Las instrucciones se representan como números.

2. Los programas pueden almacenarse en memoria para ser leídos o escritos igual que números.

El primero se ha desarrollado en los apartados anteriores. El segundo es el concepto de programa almacenado desarrollado por von Neumann. Tal y como se indica en [Fernández, 98] *"Supondremos que las instrucciones y los demás contenidos (constantes, palabras reservadas para variables, etc.) de un programa se almacenan siempre, de manera consecutiva, a partir de la palabra de dirección 0, y que existe algún procedimiento para almacenar, o cargar, los programas en la MP previamente a su ejecución."*

8.5 Programación utilizando las tablas de memoria

El objetivo es desarrollar un programa que muestre por pantalla los caracteres de la codificación ISO Latin1. Para programarlo se utilizarán las tablas de memoria directamente, tal y como se explicó en el Capítulo 2, y se irán modificando a medida que se vaya probando. Es decir se programará directamente en código máquina de Símplez. Al haber bajado un nivel de abstracción se observará algo más de complejidad e inconvenientes que si se utilizase el método explicado en el apartado siguiente.

Se podría almacenar los códigos de los casi 200 caracteres que reconoce Símplez en diferentes posiciones de memoria, y utilizar un programa similar al utilizado en [Fernández,98], que sumaba cien números, para ir mostrándolos en pantalla. La idea sería ir modificando la instrucción que carga el carácter desde la posición de memoria correspondiente al acumulador, para que fuera recorriendo todas las posiciones de memoria donde hay un carácter almacenado.

Pero dado que la codificación de los caracteres ISO Latin1 es consecutiva, sería poco eficiente tener que introducir antes los 200 caracteres en memoria, o incluso el crearse una pequeña rutina que lo hiciera.

La solución más eficiente pasa por almacenar en una posición de memoria el código del primer carácter y luego ir incrementándolo. Para esto se utilizará la posición 1 de memoria según se muestra en las tablas de caracteres del código ISO Latin1, incluidas en el Capítulo 6, el valor inicial de dicha posición será "32". También se necesitará otra posición de memoria, que almacene el valor “1”, para luego ir sumándoselo al código de carácter. Esto no sería necesario si Símplez incluyera una operación que permitiera incrementar el contenido del acumulador.

Para ir modificando el código del carácter e ir imprimiéndolo en pantalla se utilizará el bucle:

```
LD      /1
ST      /509
ADD     /2
ST      /1
BR      /5
```

Dado que a priori no se sabe cuántas variables se necesitarán, se ha situado el comienzo del bucle en la posición 5. Para pasar a ejecutar a esta posición, en la posición 0 de memoria se almacenará la instrucción BR /5.

Introduciendo el programa a través de las tablas de memoria, el aspecto de esta primera versión es el mostrado en la Figura 131.

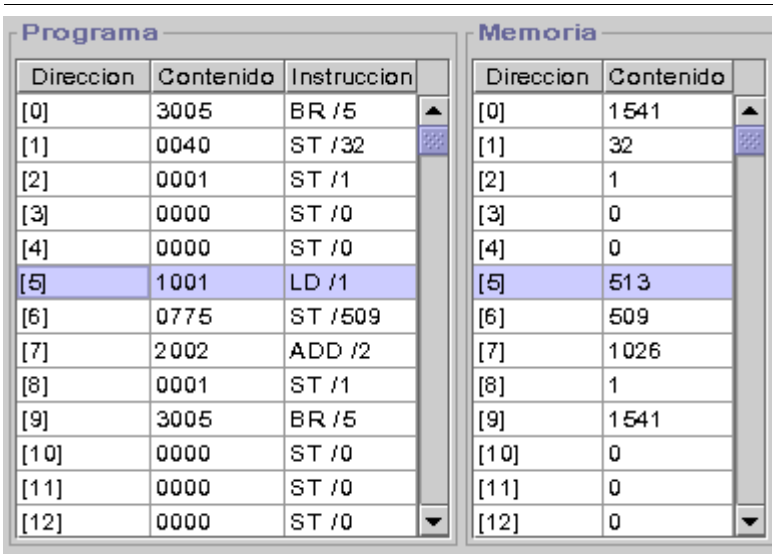


Figura 131: Aspecto del programa en las tablas de memoria.

Y el aspecto inicial de la pantalla es el mostrado en la Figura 132. Para hacer las pruebas se seleccionará el modo de ejecución CONTINUO y se reducirá el ciclo de pantalla, de manera que no se tenga que esperar mucho para ver imprimirse los caracteres.

Figura 132: Aspecto inicial de la pantalla.

Si se pulsa el botón de ejecución se verá que no aparece ningún carácter por pantalla. Esto se debe a que no se ha inicializado el puerto de control de la pantalla a “1 ”. Dado que esto hay que hacerlo antes de empezar a mandar caracteres a la pantalla, se debería situar en la posición 5, por lo que se deben desplazar todas las instrucciones dos posiciones hacia abajo y modificar la instrucción BR para que ahora el bucle comience en la posición 7. Por tanto, la utilización de este método hace que un pequeño descuido obligue a modificar gran parte de la tabla de memoria.

Una vez modificado el programa, el bucle quedaría:

```
[5]     LD     /2
[6]     ST     /508
[7]     LD     /1
[8]     ST     /509
[9]     ADD    /2
[10]    ST     /1
[11]    BR     /7
```

Si ahora se vuelve a pulsar el comienzo de ejecución se observará al cabo de un rato una salida como la siguiente:

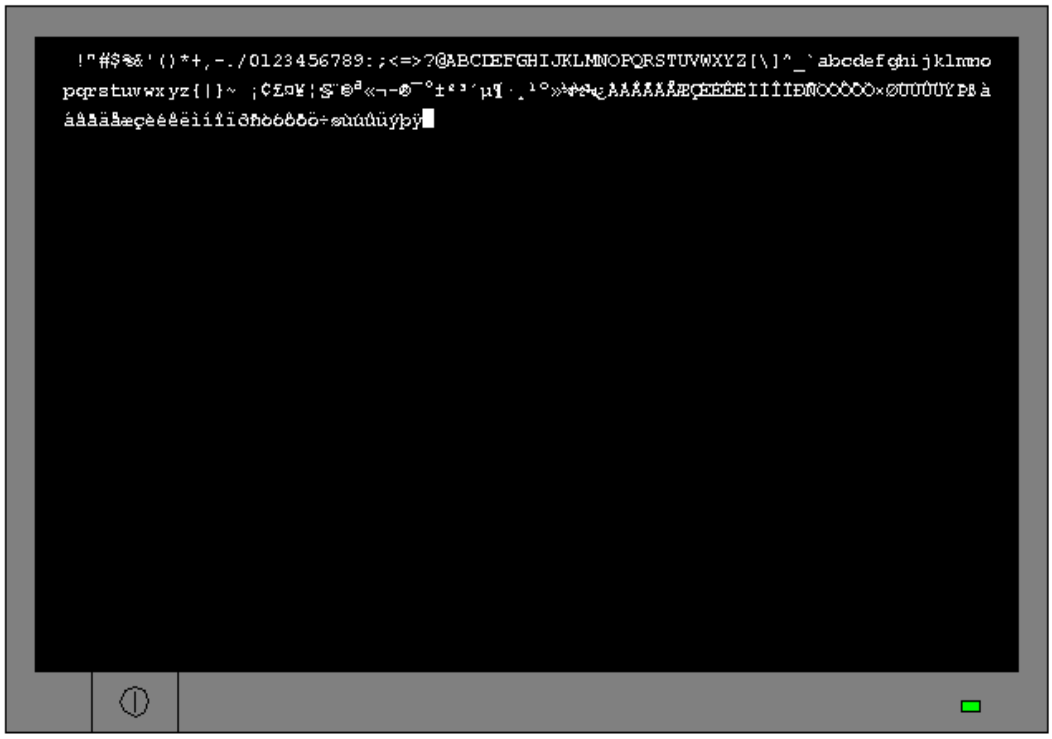


Figura 133: Resultado de la ejecución con la segunda versión.

Dado que el objetivo era pintar los caracteres, podría parecer que esto ya está hecho y que el programa está funcionando correctamente, ya que salen todos y en secuencia. Sin embargo este programa contiene un error muy grave. Manda los caracteres al controlador de pantalla sin comprobar si éste está preparado. Al haber reducido la duración del ciclo de pantalla ha coincidido que la duración de las instrucciones del bucle es mayor de 16 ciclos y da tiempo a terminar el ciclo de pantalla. Si se utilizara un ciclo de pantalla mayor o incluso el siguiente bucle:

```
[5]     LD     /2
[6]     ST     /508
[7]     LD     /1
[8]     ST     /509
[9]     ADD    /2
[10]    BR     /8
```

que en principio debería producir el mismo resultado pero sin ir almacenando y leyendo el valor del carácter en curso, se observará que el resultado sacado por pantalla es notablemente diferente tal y como muestra la Figura 134.

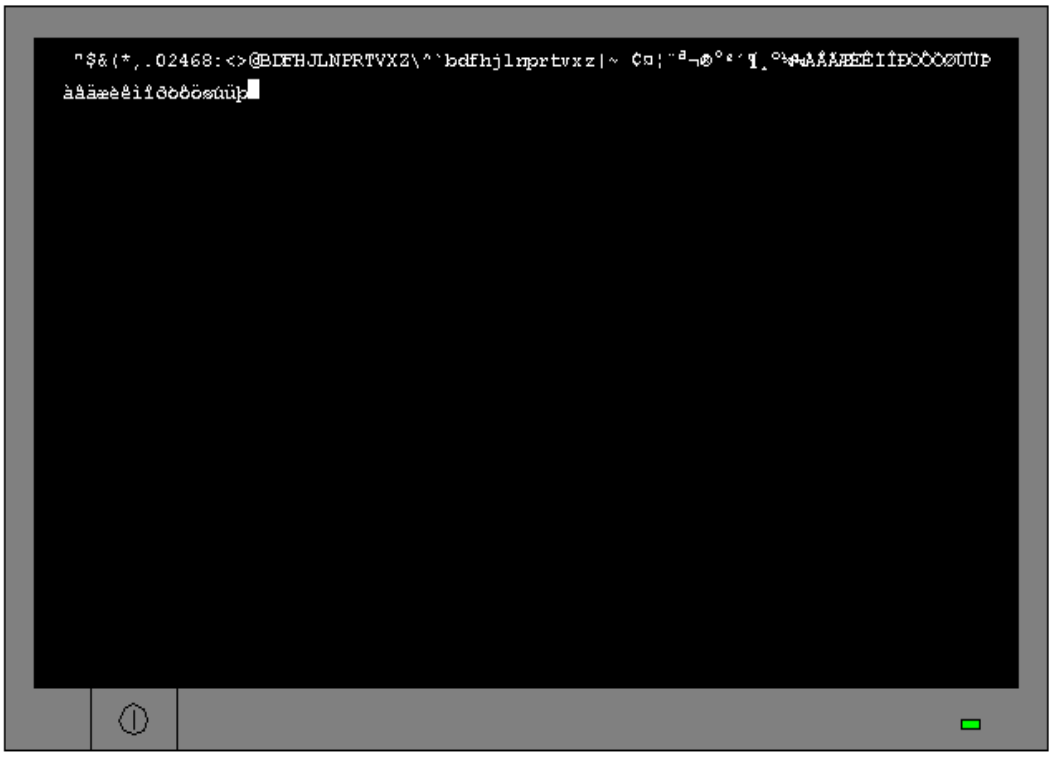


Figura 134: Resultado por pantalla utilizando el bucle optimizado.

En el caso del nuevo bucle se pierde un carácter de cada dos. Si se utilizara la versión original pero aumentando el ciclo de pantalla a 99 ciclos se observaría que se pierden 1 de cada 6 caracteres. Si se utiliza el ciclo de pantalla por defecto se pierden todos los caracteres excepto el primero, un espacio en blanco.

Para evitar que esto ocurra hay que utilizar la espera activa, tal y como se indica en [Fernández,98], que se basa en leer el puerto de control repetidamente hasta que se compruebe que la pantalla está preparada para recibir otro carácter.

Lo único que se debe hacer por tanto, es añadir una instrucción que lea el puerto de control y otra que compruebe si contiene “1”, con lo que se continuaría con la ejecución, o “0”, en cuyo caso volvería a leerlo. El programa modificado quedaría:

```
[5]     LD     /2
[6]     ST     /508
[7]     LD     /1
[8]     ST     /509
[9]     ADD    /2
[10]    ST     /1
[11]    LD     /508
[12]    BZ     /11
[13]    BR     /7
```

La instrucción "BZ /11" deja continuar con la ejecución de la siguiente instrucción únicamente en el caso de que el puerto de control de la pantalla contenga un “1”. Ahora el programa mostrará todos los caracteres independientemente del ciclo de pantalla.

Con esta última versión aún se pueden apreciar un par de cosas; en primer lugar el programa no se detiene nunca y repite indefinidamente las secuencias por pantalla. Pasado un rato se observará que el cursor se retrasa una posición, luego se adelanta cinco posiciones, da un retorno de carro y por último se borra la pantalla. Pasado un tiempo los caracteres se vuelven a dibujar. Esto se debe a que el puerto de datos de la pantalla ya ha pasado por el valor 255 y comienza de nuevo por cero, imprimiendo las secuencias de control y a continuación los caracteres, Lo más elegante sería que una vez se hayan impreso todos los caracteres se detuviera la ejecución.

En segundo lugar se observa que entre el carácter “~” y el carácter <p> pasa un tiempo sin dibujarse nada en pantalla. Esto se debe a que los caracteres están divididos en dos bloques cuya codificación no es consecutiva y se están enviando al puerto de datos 32 valores que no se corresponden con ningún carácter. Lo más adecuado sería pasar a imprimir por pantalla el carácter <p> una vez se haya impreso el carácter <d> que, aunque no se perciba en pantalla, es el carácter siguiente a "~".

Para solucionar lo primero basta con comprobar que el último carácter impreso por pantalla ha sido "ÿ", tras lo cual bifurcar a una posición que contenga la instrucción HALT. Para realizar la comprobación se puede almacenar el valor complementario a 4095 del código correspondiente a "ÿ", de manera que la suma de ambos valores sea 4096, que dentro del acumulador pasaría a ser un cero. Dado que el código de "ÿ" es 255, y que dicho valor ya se ha incrementado con la instrucción de posición [9], el valor que se debe sumar para obtener un 0 en el acumulador es 3840. Por tanto quedaría algo semejante a:

```
[a]   LD   /código_carácter
[b]   ADD  /complementario
[c]   BZ   /e
[d]   BR   /7
[e]   HALT
```

Para evitar el segundo inconveniente del programa, se puede adoptar varias soluciones. Una de ellas consiste en bifurcar la ejecución a otra zona de memoria cuando el último carácter impreso sea <d>. Esta zona de memoria repetiría esencialmente el mismo código almacenado desde la posición [7], pero cargando la instrucción LD, almacenada en esa posición, el valor 160, correspondiente al carácter <p>, en lugar de cargar el valor 32, correspondiente al carácter <s>.

Otra posibilidad consiste en bifurcar la ejecución, una vez se haya impreso en pantalla el primer bloque de caracteres, a una pequeña rutina que sustituya el contenido de la posición 1 de memoria por el valor correspondiente al carácter <p>, y luego devolver la ejecución de nuevo a la posición 5 de memoria.

En ambos casos para comprobar que el último carácter impreso es <d>, se utilizará el mismo método indicado antes para bifurcar a la instrucción HALT. Teniendo en cuenta que el código del carácter <d> es 127 y que el contenido de la posición de memoria [1], que almacena el carácter en curso, ya se ha incrementado, el valor a utilizar es 3968.

Aún hay que hacer una modificación más. Con el espacio actual reservado para variables no se podría almacenar todas las necesarias. Comenzando el programa en la posición [5] deja espacio para cuatro variables y el programa necesita cinco: carácter inicial del primer bloque, carácter inicial del segundo bloque, valor complementario de <d>, valor complementario de "ÿ", y valor "1". Se podría almacenar una de ellas fuera del espacio de caracteres o desplazar todo el programa una posición. Utilizando esta segunda posibilidad el aspecto final del programa será el del Programa 3.

```
Dir.   Cont.    Contenido  Comentarios
Mem.  (octal)  (nemónico) ____________________________
----  -------  ----------
 [0]    3006   BR  /6      ; Salta a Inicio bloque1
 [1]    0040       32      ; Código carácter <s>
 [2]    0001       1       ; Unidad
 [3]    0240       160     ; Código carácter <p>
 [4]    7600       3968    ; Complementario de <d>
 [5]    7400       3840    ; Complemento de ÿ
 [6]    1002   LD  /2      ; Inicio bloque1
 [7]    0774   ST  /508    ; Inicializa puerto control
 [8]    1001   LD  /1      ; Carga el carácter en AC
 [9]    0775   ST  /509    ; Imprimirlo por pantalla
[10]    2002   ADD /2      ; Incrementamos código car.
[11]    0001   ST  /1      ; Guardamos el nuevo código
[12]    1774   LD  /508    ; Comprueba si ha terminado
[13]    4014   BZ  /12     ;  de imprimirlo
[14]    1001   LD  /1      ; Comprueba si es el último
[15]    2004   ADD /4      ;  carácter de este bloque y
[16]    4022   BZ  /18     ;  si lo es salta al bloque2
[17]    3010   BR  /8      ; Vuelta a imprimir otro
[18]    1003   LD  /3      ; Inicio bloque2. Carga car.
[19]    0775   ST  /509    ; Imprimirlo por pantalla
[20]    2002   ADD /2      ; Incrementamos código car.
[21]    0003   ST  /3      ; Guardamos el nuevo código
[22]    1774   LD  /508    ; Comprueba si ha terminado
[23]    4026   BZ  /2      ;  de imprimirlo
[24]    1003   LD  /3      ; Comprueba si es el último
[25]    2005   ADD /5      ;  carácter de este bloque y
[26]    4034   BZ  /28     ; si lo es salta a terminar
[27]    3022   BR  /18     ; Vuelta a imprimir otro
[28]    7000   HALT        ; Terminar
```

Programa 3: Muestra todos los caracteres por pantalla con dos bucles.

La solución utilizando una subrutina supone bifurcar el flujo de ejecución a una pequeña sección de código que realice las modificaciones necesarias y que luego devuelva la ejecución al comienzo del bucle que muestra los caracteres. Se utilizará, por tanto, el mismo bucle para imprimir por pantalla los dos bloques de caracteres.

Las modificaciones a realizar sobre el programa por la subrutina son:

- Código correspondiente al primer carácter del bloque de caracteres.
- Complemento a 4095 del último carácter del bloque.
- Dirección de bifurcación tras la impresión del último carácter.

En el caso de las dos primeras se puede, o modificar las instrucciones que manejan dichos datos para que los tomen de otra dirección, o modificar el dato en sí, dejando las instrucciones intactas. Se utilizará esta última opción. Para el caso de la dirección de bifurcación es inevitable tener que modificar la instrucción BZ.

La subrutina sustituirá el contenido de las posiciones de memoria correspondientes a cada uno de los valores anteriores por valores leídos de otras tantas direcciones de memoria. Por último tendrá que retornar el flujo de ejecución al comienzo del bucle, pasada la inicialización del puerto de control. El aspecto del bucle será por tanto:

```
[a]   LD   /NUEVO_CARÁCTER
[b]   ST   /1
[c]   LD   /NUEVO_COMPLEMENTO
[d]   ST   /4
[e]   LD   /h
[f]   ST   /16
[g]   BR   /INICIO BUCLE
[h]   BZ   /18
```

Antes de la ejecución de la subrutina, el aspecto de la sección de código que hace las comprobaciones es:

```
[14]     LD      /1
[15]     ADD     /4
[16]     BZ      /a
[17]     BR      /8
[18]     HALT
```

Tras la ejecución de la subrutina dicho aspecto sería:

```
[14]    LD      /1
[15]    ADD     /4
[16]    BZ      /18
[17]    BR      /8
[18]    HALT
```

Como se puede comprobar ahora tras imprimirse el último carácter se bifurca a la instrucción HALT, en lugar de bifurcar a la subrutina, lo que permite que una vez impresos todos los caracteres por pantalla se detenga la ejecución.

El aspecto final del programa situando la rutina en la posición 50 de memoria es el de Programa 4.

En ambas soluciones se puede observar que una vez mandado a imprimir por pantalla el último carácter, el programa espera que se cumpla el ciclo de pantalla antes de bifurcar a la instrucción HALT. Si no se hiciera así y, por ejemplo, la comprobación de que la pantalla ha terminado de imprimir el carácter estuviera al comienzo del bucle, el simulador interrumpiría la ejecución y el último carácter no se imprimiría. La máquina real, en cambio, sí continuaría ejecutando infinitamente las microórdenes generadas por HALT y al final el último carácter se imprimiría. Este comportamiento se puede simular seleccionando el secuenciador microprogramado y deshabilitando el botón de interrupción con **cbf** de la ventana de la Ruta de Datos. Con el secuenciador cableado la única manera de lograr que terminara por imprimirse sería pulsar repetidamente el botón de ejecución. Si el ciclo de pantalla es de 33 ms esto último supondría pulsarlo miles de veces.

Como conclusión, indicar que en el caso de utilizar este método para desarrollar programas es aconsejable, a ser posible, que el espacio reservado para constantes se encuentre en una zona alta de memoria en lugar de hacerlo al principio del programa. De esta manera se pueden añadir tantas como se necesiten sin tener que modificar el código, lo que implicaría desplazar el programa y buscar las instrucciones de bifurcación o con referencias a posiciones de éste y cambiarlas.

```
Dir.   Cont.    Contenido  Comentarios
Mem.  (octal)  (nemónico) ____________________________
----  -------  ---------- 
 [0]    3006   BR  /6      ; Salta a Inicio bloque1
 [1]    0040       32      ; Código carácter <s>
 [2]    0001       1       ; Unidad
 [3]    0240       160     ; Código carácter <p>
 [4]    7600       3968    ; Complementario de <d>
 [5]    7400       3840    ; Complemento de ÿ
 [6]    1002   LD  /2      ; Inicio
 [7]    0774   ST  /508     ; Inicializa puerto control
 [8]    1001   LD  /1      ; Carga el carácter en AC
 [9]    0775   ST  /509     ; Imprimirlo por pantalla
[10]    2002   ADD /2      ; Incrementamos código car.
[11]    0001   ST  /1      ; Guardamos el nuevo código
[12]    1774   LD  /508     ; Comprueba si ha terminado
[13]    4014   BZ  /12      ;  de imprimirlo
[14]    1001   LD  /1      ; Comprueba si es el último
[15]    2004   ADD /4      ;  carácter de este bloque y
[16]    4022   BZ  /50      ;  si es, salta a subrutina
[17]    3010   BR  /8      ; Vuelta a imprimir otro
[18]    7000   HALT        ; Terminar
 ...     ...   ... ...     ...
[50]    1003   LD  /3      ; Carga primer carácter
[51]    0001   ST  /1      ; Modifica la variable car.
[52]    1005   LD  /5      ; Carga nuevo complementario
[53]    0004   ST  /4      ; Modifica el complementario
[54]    1071   LD  /57     ; Carga nueva bifurcación
[55]    0020   ST  /16     ; Modifica la bifurcación
[56]    3010   BR  /8      ; Vuelta al bucle
[57]    4022   BZ  /18     ; Nueva inst. de bifurcación
```

Programa 4: Muestra todos los caracteres utilizando un bucle y una subrutina.

8.6 Programación a través del editor−ensamblador

A continuación se revisa un ejemplo de programación utilizando el editor de programas, que suma diez números almacenados en memoria, y que es una modificación del programa explicado en [Fernández,98], que sumaba cien números.

Este editor permite programar utilizando el lenguaje ensamblador explicado en el Capítulo 7. La gran ventaja de utilizar un lenguaje ensamblador son las etiquetas que sustituyen a las direcciones de memoria y la posibilidad de definir variables y constantes según se vayan a utilizar posteriormente.

En el ejemplo a programar se necesita una variable que indique cuántos números se han de sumar y una constante que marque la posición inicial de los datos en memoria.

```
DIEZ      DATA     10
DIR       EQU      109
```

Si se decidiera cambiar la cantidad de datos a sumar bastaría con cambiar el valor de esta variable y ensamblar de nuevo.

La posición del primer dato a sumar es la del número almacenado en la posición de dirección de memoria más alta; ya que a diferencia del programa del apartado anterior, en que se necesitaba incrementar la variable que indicaba el carácter a imprimir, ahora el resultado es independiente del orden en que se manejan los datos. Esto permite poder ir decrementando simplemente la dirección donde se encuentran estos.

Se necesita una variable que lleve la cuenta de las vueltas que ha dado el bucle. Para ello se reserva una posición de memoria:

```
CONT      RES      1
```

En este caso tampoco se tiene la necesidad de ir incrementando esta variable. Inicializándola con el número de datos a sumar, se puede ir decrementándola hasta llegar a cero, momento en que simplemente se debe bifurcar al final del programa. Esto permite no tener que almacenar un número complementario. En los programas del apartado anterior también se podría haber utilizado esta técnica para saber si ya se habían sacado por pantalla todos los caracteres. En lugar de comprobar si el código del carácter en curso coincidía con el último del bloque correspondiente, bastaba con comprobar que se habían imprimido 96 caracteres de ese bloque.

Para ir recorriendo las diferentes posiciones de memoria donde están guardados los datos basta con ir decrementando en cada vuelta del bucle la instrucción ADD, que lee el dato de memoria y se lo suma al contenido actual del acumulador, donde previamente se habrá cargado el valor parcial de la suma. Si dicha instrucción se sitúa al comienzo del bucle, bastará con una única etiqueta tanto para las bifurcaciones como para hacer referencia a ella desde las instrucciones de modificación.

El aspecto final del programa será el del Programa 5.

```
; Este programa suma diez números almacenados
; en diez direcciones de memoria.

MODULE SUMADIEZ
         BR    /INICIO  ; Bifurca al comienzo
DIEZ     DATA  10       ; Cantidad de números
CONT     RES   1        ; Vueltas del bucle
         EQU   109      ; Dirección del último
                        ; número a sumar
SUMA     RES   1        ; Resultado de la suma

INICIO   LD    /DIEZ    ; Inicializa el contador
         ST    /CONT    ; de vueltas del bucle
         CLR
BUCLE    ADD   /DIR     ; Suma un número
         ST    /SUMA    ; Guarda suma parcial
         LD    /CONT    ; Decrementa el
         DEC            ; contador
         BZ    /FIN     ; Si es cero terminamos
         ST    /CONT    ; Si no, actualiza el contador
         LD    /BUCLE   ; Modificamos la instrucción
         DEC            ; de suma para que tome el
         ST    /BUCLE   ; siguiente número
         LD    /SUMA    ; Cargamos en AC la suma parcial
         BR    /BUCLE   ; Volvemos al comienzo del bucle
FIN      HALT
         END
```

Programa 5: Suma de diez números almacenados en memoria.

Las ventajas de utilizar etiquetas son evidentes. En ningún momento se ha tenido que hacer referencia a una dirección de memoria. Si se necesita introducir una instrucción en cualquier lugar del programa, no es necesario hacer ningún cambio.

Para introducir el programa en el ensamblador se puede utilizar un editor de texto para escribirlo o utilizar directamente el área de edición del editor de programas. Si se utiliza un editor de texto, una vez escrito el programa, se debe salvarlo como texto plano en un fichero con extensión **.sim**. Desde el editor de programas se puede cargar pulsando el botón **Abrir** de la barra de herramientas, y seleccionándolo desde la ventana de manejo de ficheros.

La otra posibilidad es introducirlo directamente utilizando el área de edición del editor de programas. Este método permite ir escribiendo y chequeando la corrección del programa simultáneamente.

Una vez introducido el programa en lenguaje ensamblador en el editor de programas, el aspecto del mismo es el mostrado en la Figura 135.

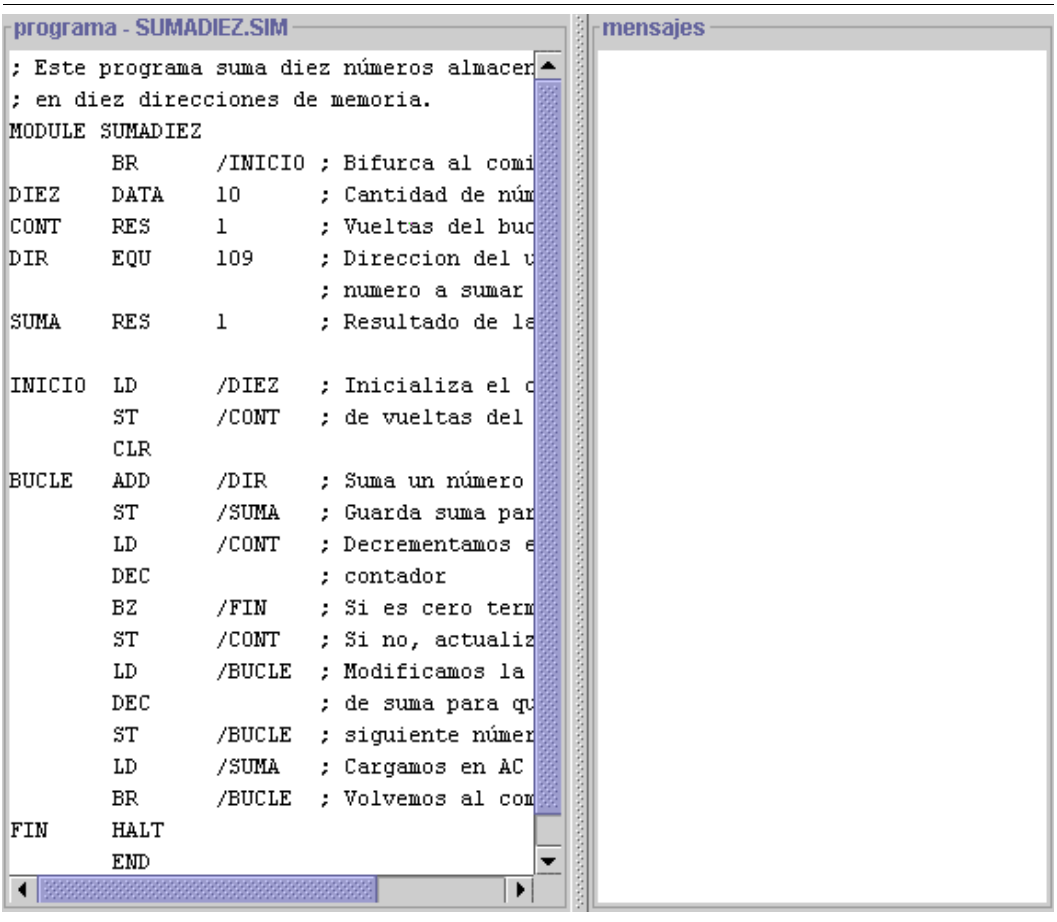


Figura 135: Programa SUMADIEZ cargado en el editor de programas.

Si el programa es correcto, pulsando el botón de **Ensamblado** de la barra de herramientas aparecerá en el área de mensajes de la izquierda el resultado del ensamblado, tal y como muestra la Figura 136.

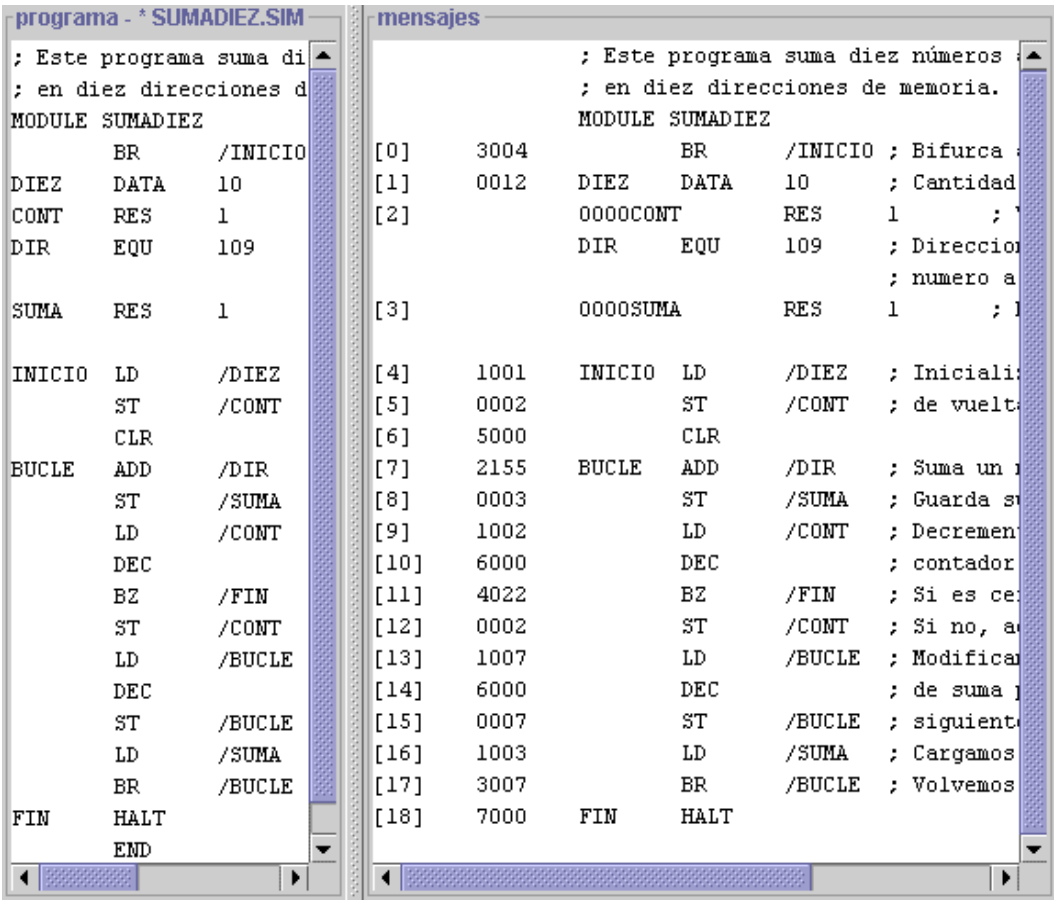


Figura 136: Resultado de ensamblar el programa SUMADIEZ.

Desde esta ventana es importante identificar la posición de memoria que se le ha asignado a la etiqueta SUMA, y que será donde se encuentre el resultado final. En la figura se observa que se trata de la posición 3.

Una vez comprobado que no se producen errores al ensamblar se puede **Montar** el programa pulsando el botón correspondiente de la barra de herramientas. Hecho esto se habrá generado un fichero de extensión **.lns**, el cual se puede cargar en el simulador desde la entrada **Abrir fichero de código** del menú **Archivo** de la ventana principal.

Una vez cargado el programa, el aspecto del simulador es el mostrado en la Figura 137.

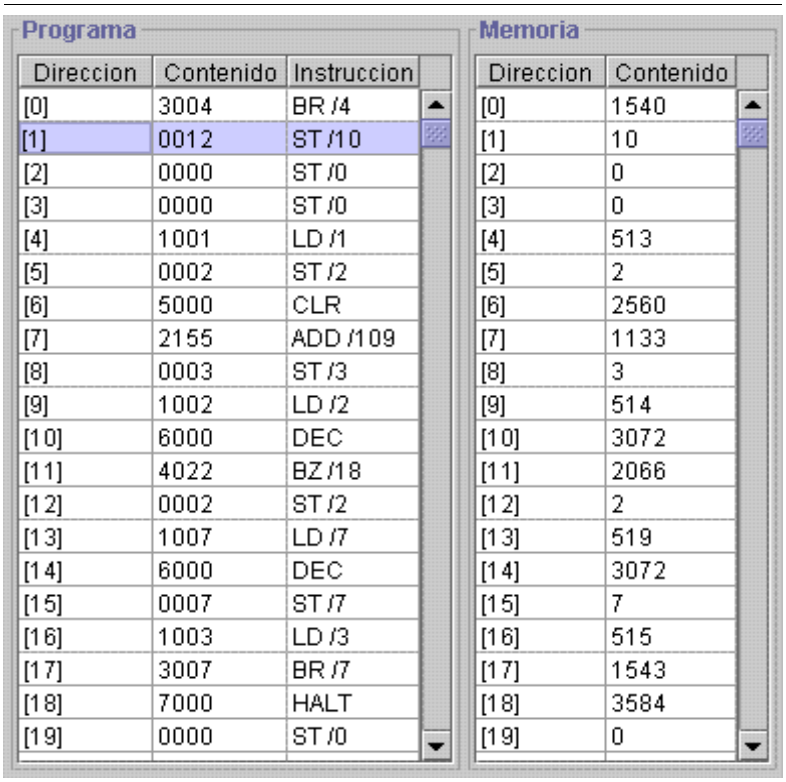


Figura 137: Programa SUMADIEZ cargado en el simulador.

Sólo quedan por introducir los datos a sumar en las posiciones que van desde la 100 a la 109. Para ello se puede utilizar directamente la tabla de Memoria de la ventana principal. Supóngase que son los mostrados en la Figura 138.

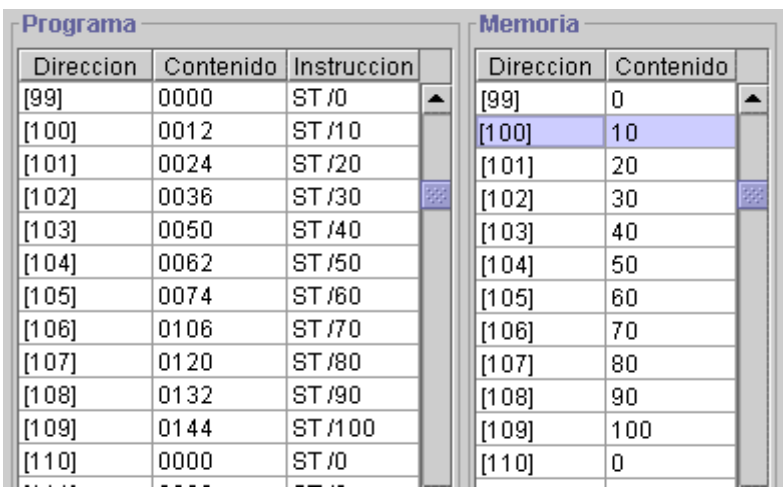


Figura 138: Datos a sumar por el programa SUMADIEZ.

El resultado de la suma de estos números es 550. Si se ejecuta el programa se comprobará que éste es precisamente el resultado obtenido, tal y como muestra la Figura 139.

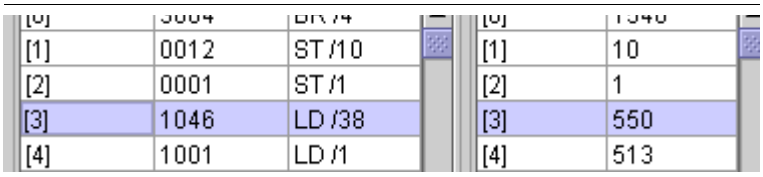


Figura 139: Resultado de la ejecución del programa SUMADIEZ.

El resultado de la suma se encuentra almacenado en la posición 3 de la memoria.

Como se ha podido observar este método es mucho más flexible para desarrollar un programa. Alguno de los errores cometidos al programar son muy difíciles de detectar y corregir. Gracias a la representación simbólica que admite la tabla de Programa alguno de estos errores se elimina, sobre todo los debidos a codificación. Pero aún así, se seguirá cometiendo gran número de errores. Uno de ellos muy común es el debido a desplazamientos de secciones de código a otra posición de memora y olvidar modificar las referencias a estas secciones desde otras instrucciones del programa. Esto se evita con la utilización de etiquetas en lenguaje ensamblador, que permite no tener que manejar direcciones de memoria.

Capítulo 9

Microprogramación de Símplez

Los simuladores de computadores implementan normalmente todo lo visto hasta ahora. Permiten escribir, ensamblar y simular la ejecución de programas escritos en su lenguaje máquina, ensamblador y, en algún caso, en lenguajes de alto nivel.

El simulador de Símplez permite dar un paso más y microprogramar la máquina. Es decir, permite modificar el diseño del secuenciador de la UCP. Lo que supone la modificación del hardware de la máquina.

Gracias a la microprogramación se pueden crear nuevas instrucciones que realicen tareas diferentes a las implementadas por el repertorio de instrucciones del secuenciador cableado. Estas tareas están muy limitadas debido al sencillo diseño de Símplez, pero permiten entender en profundidad el diseño y funcionamiento de la Ruta de Datos.

Primero se revisan brevemente los conceptos teóricos necesarios para microprogramar. En [Fernández,98] se puede encontrar un explicación completa de los mismos.

9.1 Conceptos teóricos

Como ya se sabe una **microorden** es una señal binaria que permite gobernar el comportamiento de un elemento de la Ruta de Datos. Hay elementos gobernados por más de una microorden.

Para poder gobernar adecuadamente toda la Ruta de datos en cualquier instante es necesario generar las microórdenes correspondientes a todos y cada uno de los elementos que la forman. Al conjunto de todas las microórdenes generadas en un momento dado se le denomina **microinstrucción**. El repertorio de microinstrucciones se obtiene combinando las microórdenes que las forman. Dado que en el modelo microprogramado de Símplez hay 18 microórdenes y 3 bits de direccionamiento de la Memoria de Control, cada microinstrucción estará formada por 21 bits, por lo tanto hay 2.097.152 microinstrucciones posibles.

Tal y como se explicó en el Capítulo 4, los elementos de la Ruta de Datos a los que llegan más de una microorden establecen un orden de precedencia entre ellas en el caso en que más de una esté activa, y sólo la microorden de mayor precedencia tendrá efecto. Esto significa que al número total de microinstrucciones hay que restarle el conjunto resultante de combinar las microórdenes de menor precedencia de un mismo elemento cuando una de más precedencia está activa.

Microprogramar Símplez implica seleccionar las microinstrucciones adecuadas para lograr llevar a cabo la operación que debe realizar una instrucción. Esta operación normalmente no se podrá implementar con una única microinstrucción y será necesario una secuencia de ellas. A esta secuencia de microinstrucciones se le denomina **microprograma**. El tamaño de un microprograma será variable de una instrucción a otra, y dependerá de la operación a realizar por la misma.

Por ejemplo el microprograma correspondiente a la instrucción ST, utilizando la notación de estados vista para la implementación cableada, es el siguiente:

Estado	microórdenes
I0	lec, eri, incp, sco
I1	sri, era, sac
O0	esc, sac
O1	scp, era, sac

Tabla 11: Microprograma de ST.

El conjunto de microprogramas que implementan el repertorio de instrucciones de Símplez forman la **Memoria de Control**.

Cada microinstrucción de un microprograma incorpora un campo de dirección que permite direccionar la siguiente microinstrucción a ejecutar dentro de la Memoria de Control. Por tanto microprogramar implica una labor adicional a la mera selección de microinstrucciones. Esta labor consiste en la disposición de las microinstrucciones dentro de la Memoria de Control, y en ir dando valores a los campos de dirección de las mismas según las posiciones que ocupen. El direccionamiento de la Memoria de Control se explicó en detalle en la página 65.

Para la implementación microprogramada por defecto, la disposición de los diferentes microprogramas es la siguiente:

Dirección MC	Microinstrucción correspondiente al estado
0	I1 de ST
1	I1 de LD
2	I1 de ADD
3	I1 de BR
4	I1 de BZ
5	I1 de CLR
6	I1 de DEC
7	I1 de HALT
8	O0 de ST
9	O1 de ST
10	O0 de LD
11	O1 de LD
12	O0 de ADD
13	O1 de ADD
14	Libre
15	I0 común

Tabla 12: Disposición de la Memoria de Control.

Según esta disposición y teniendo en cuenta el formato del editor de microprogramas, el contenido final de la Memoria de Control por defecto incluida en el simulador es el mostrado en la siguiente tabla.

Dir.	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
μO.																
eac	0	0	0	0	0	1	1	0	0	0	1	0	1	0	0	0
eri	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
incp	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
ecp	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0
ccp	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
era	1	1	1	1	1	1	1	1	0	1	0	1	0	1	0	0
pac	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
sum	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
tra2	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
dec1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
lec	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	1
esc	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
sac	1	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0
sri	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
scp	0	0	0	0	1	1	1	1	0	1	0	1	0	1	0	0
cbf	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
sco	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
bbz	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
dir1	0	0	0	1	1	1	1	1	1	1	1	1	1	1	0	0
dir2	0	1	0	1	1	1	1	1	0	1	1	1	0	1	0	0
dir3	0	0	1	1	1	1	1	1	0	1	0	1	1	1	0	0

Tabla 13: Contenido de la Memoria de Control.

9.2 Implementación

Hay una serie de cuestiones relativas a la implementación del simulador que es necesario tener en cuenta a la hora de microprogramar.

Final de instrucción

El criterio utilizado por el simulador para saber que comienza una nueva instrucción es que la microinstrucción siguiente contenga la microorden **sco** activa. La razón es que esta microorden hace que la dirección de la siguiente microinstrucción de la Memoria de Control a ejecutar se lea desde el bus D, en lugar de utilizar el campo de dirección de la microinstrucción en curso.

En la implementación cableada el criterio seguido es que el biestable I tenga el valor "1" y el biestable T el valor "0".

Instrucción en curso

El campo "**En curso**" de la ventana principal indica en todo momento qué instrucción se está ejecutando. El campo muestra además la dirección de memoria dónde se encuentra.

El simulador obtiene la instrucción leyendo el contenido del Registro de Instrucción. La dirección en memoria de la instrucción la obtiene leyendo del bus directamente. Esto hace que únicamente la parte correspondiente a la instrucción del campo "**En curso**" sea cierta con seguridad. La dirección puede no serla. Por ejemplo, en el caso en que no se genere la microorden **eri** durante I0 para actualizar el Registro de Instrucción con el valor presente en el bus D, en el siguiente ciclo de reloj se ejecutará la instrucción ya presente en el mismo al comienzo de I0. El campo "**En curso**" mostrará en cambio la dirección de la instrucción que debería estar ejecutándose.

Puede parecer que esta situación no es posible debido a que siempre se ejecutaría la instrucción "ST /0", que es la que se corresponde con el contenido inicial del Registro de Instrucción, y nunca cambiaría su valor. Pero no hay que olvidar que el cualquier momento se puede cambiar el secuenciador a utilizar y el valor contenido por el Registro de Instrucción correspondería a la última instrucción ejecutada con otro secuenciador.

9.3 Microprogramación

En este apartado se verá un ejemplo de utilización del editor para microprogramar Símplez. Supóngase que se quiere crear una instrucción compuesta por tantas microinstrucciones como sea posible. Se podría pensar en crear una instrucción compuesta por 16 microinstrucciones, marcando **sco** o no según fuera necesario en cada microinstrucción para direccionar toda la Memoria de Control. Pero esto no es posible dado que, como ya se explicó en capítulos anteriores, esta microorden también marca el comienzo de una nueva instrucción. De manera que su aparición supone el final de la instrucción en curso.

Por tanto la instrucción podrá estar compuesta por un máximo de 9 microinstrucciones. La primera la de dirección 15, que tal y como se explica en el Capítulo 9, es común a todas las instrucciones del repertorio y se encarga, gracias a **sco**, de comprobar de qué instrucción se trata y así poder decidir cuál es el punto de entrada en la Memoria de Control. La segunda microinstrucción será una de las 8 primeras, dependiendo de qué instrucción del repertorio se vaya a sobreescribir. El resto de microinstrucciones se corresponden con las que van de la posición 8 a la 14 inclusive.

En realidad cuando se habla de sobreescribir una instrucción se refiere a utilizar su código de operación, ya que al utilizar toda la parte superior de la Memoria de Control se está sobreescribiendo ST, LD y ADD que son las que necesitan 4 microinstrucciones para llevar a cabo su cometido. La instrucción escogida para utilizar su código de operación es ST. Cuando en el registro de instrucción se cargue la instrucción a ejecutar por obra de la microinstrucción de dirección 15 y ésta resulte ser ST, se ejecutarán las ocho microinstrucciones restantes.

El contenido de la matriz de direcciones, habiendo puesto a cero las que no son relevantes e incluyendo la línea que indica a que microinstrucción pertenece cada una, es el de la Figura 140.

Figura 140: Matriz de direcciones para una instrucción de nueve microinstrucciones.

Tal y como se puede observar, la microinstrucción de dirección 1, la primera de ST, indica que la siguiente microinstrucción es la de dirección 8. A partir de ésta se irán recorriendo secuencialmente las direcciones hasta la 14. La microinstrucción de dirección 14 indica que la siguiente a ejecutar es la de dirección 15, que es la primera microinstrucción común a todas las instrucciones y que por contener **sco** activa (que no se ha incluido en la imagen) pertenecerá a la siguiente instrucción. Todas las instrucciones deben acabar con una microinstrucción que apunte a la microinstrucción de dirección 15.