Apéndice A

Manual de usuario

En este manual se explica como utilizar *Simula3MS*. En la sección A.1 se enumeran los pasos básicos para usar la herramienta. A continuación se describen de forma más detallada la edición de programas en lenguaje ensamblador para este procesador, los distintos elementos del simulador: el segmento de texto, el segmento de datos, los registros, etc, y las distintas configuraciones: monociclo, multiciclo y segmentado.

A.1. Guía rápida

A continuación se indican los pasos básicos para empezar a trabajar con *Simula3MS*:

1. Una vez abierta la ventana de *Simula3MS*, existen dos opciones:
   - Cargar un fichero que ha sido editado con anterioridad.
   - Editar un nuevo código en lenguaje ensamblador.
2. Una vez editado o cargado el fichero, el siguiente paso es ensamblarlo, para ello hay que pulsar el botón *Ensamblar*. A partir de aquí hay dos posibles resultados:
   - Si el código que queremos ejecutar no tiene errores sintácticos se activará el botón *Ejecutar* que permite acceder a la ventana de la simulación de la ejecución del código analizado.
   - En caso de que el código no sea correcto, en la parte inferior de la ventana aparecerá un listado con todos los errores y el primero de

ellos aparecerá remarcado. Se puede acceder a los siguientes, en caso de que los hubiera, por medio del botón *Error siguiente*. Una vez corregidos estos fallos se vuelve a pulsar botón *Ensamblar* y se repite este paso.

3. El paso siguiente, previo a ejecutar, será escoger la configuración del camino de datos sobre el cual queremos que se ejecute el código. Para ello, en el menú *Configuración* tenemos tres posibles opciones: *Monociclo, Multiciclo* y *Segmentado*. Por defecto la opción activada es la *Monociclo*. Si escogemos la opción *Multiciclo*, podremos elegir el número de ciclos que queremos que ocupen las operaciones de Suma/Resta, Multiplicación y División en punto flotante.

4. Una vez obtenido, el código correcto y configurado el simulador, se pulsa *Ejecutar* y tenemos acceso a la ventana en la cual se simula la ejecución del código escogido. En esta ventana se puede observar la ejecución del programa completo de modo que se mostrarán sólo los valores finales, o bien ciclo a ciclo, teniendo también la posibilidad de retroceder a ciclos anteriores, pudiendo ver así las modificaciones que cada instrucción realiza en cada ciclo.

A.2. Manual extendido

A.2.1. Edición del código

El primer paso al iniciar *Simula3MS* es editar un código en lenguaje ensamblador, para ello se puede cargar el código de un fichero o bien editarlo. La sintaxis básica utilizada por *Simula3MS* tiene las siguientes características:

- Los comentarios empiezan por el símbolo #, todo lo que aparezca en la misma línea a continuación de este símbolo es ignorado.

- Los programas se dividen en dos partes:

  - *.text:* sección obligatoria en todos los programas, contiene el conjunto de las instrucciones del programa.
  - *.data:* sección opcional, aunque normalmente necesaria. Es la sección de declaración de las variables del programa.

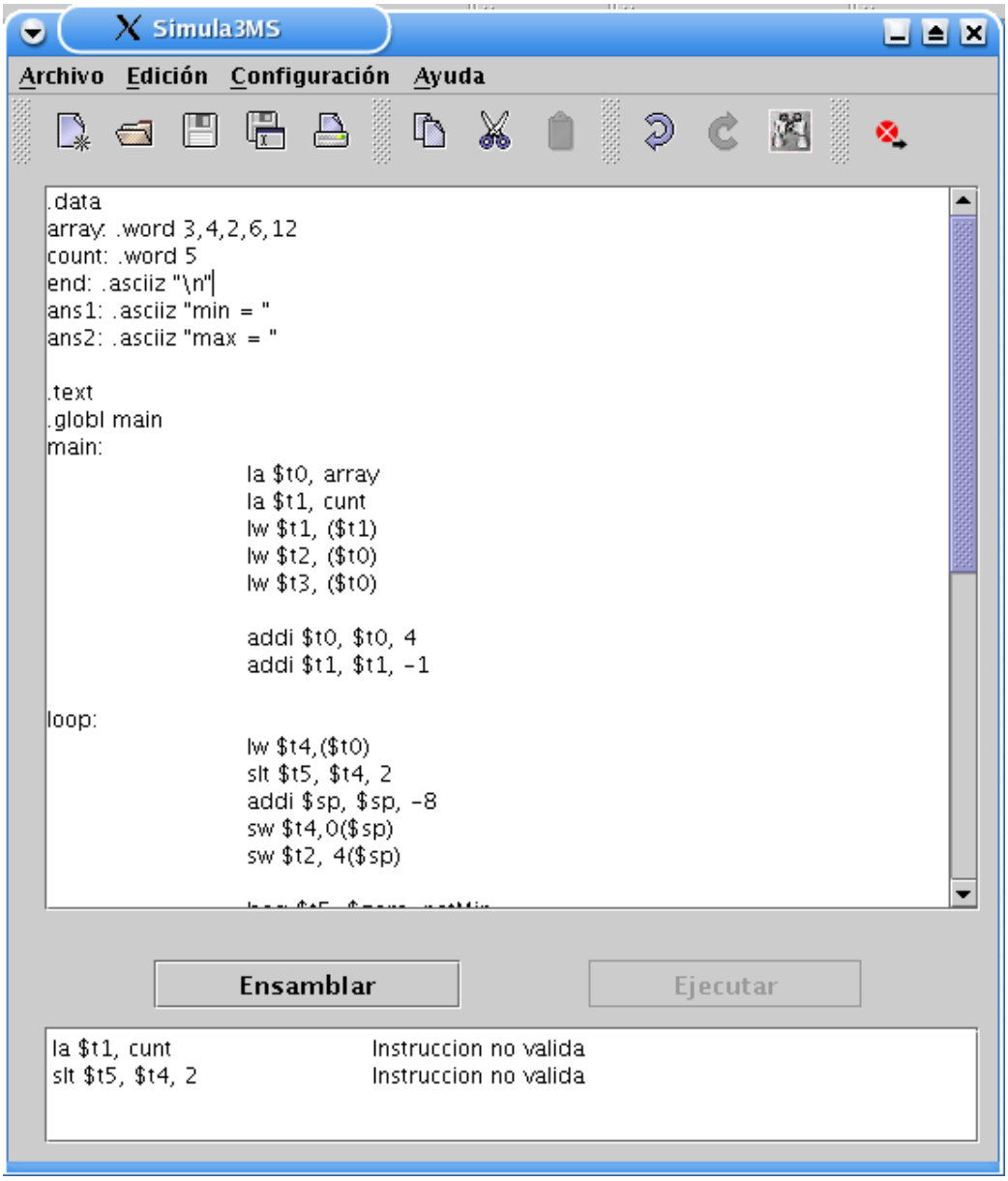

Figura A.1: Ventana del editor

Características de .text

La sección *.text* es obligatoria en todos los programas, contiene el conjunto de instrucciones del programa. Los elementos siguientes se guardan en el segmento de texto.

Las primeras líneas de la sección .text, obligatorias ya que indican al simulador donde debe empezar la ejecución del programa, son:

.globl main

.main:

A continuación se escribe el código del programa, siguiendo estas reglas:

- Las etiquetas van seguidas por dos puntos (:). Una etiqueta válida es una secuencia de caracteres alfanuméricos, subrayados (_) y puntos (.).
- En la línea de la etiqueta no puede haber una instrucción, las instrucciones se situarán a partir de la línea siguiente a la etiqueta.
- Las instrucciones válidas implementadas en este simulador se muestran en tablas según su tipo. Las enteras tipo R en el cuadro A.1, las tipo I en el A.2 y las tipo J en la tabla A.3. En cuanto a las de punto flotante, las de tipo R se muestran en los cuadros A.4, A.5 y A.6, mientras que las de tipo I están en el cuadro A.7.
- Los registros enteros pueden indicarse por su número de orden o por su nombre, por ejemplo $t1=$9. Las equivalencias se muestran en el cuadro A.9. En el caso de los registros de punto flotante, también se permiten ambas nomenclaturas pero la equivalencia se limita a suprimir la f. Ejemplo: $f6=$6.
- Por defecto los números se representan en base 10.
- Se pueden incluir llamadas al sistema operativo (*syscall*), para solicitar algún servicio como impresión por pantalla, ver sección A.5.
- Con el objetivo de ayudar al usuario a cargar variables, *Simula3MS* tiene también implementada la pseudoinstrucción `la`. Por ejemplo:

  ```
  la $1, variable
  ```

  carga en el registro $1 la dirección de memoria donde está almacenada la variable. La ejecución de está pseudoinstrucción se divide en dos instrucciones:

`lui $1, 0x1001` (representa los 16 bits de mayor peso de la dirección de memoria)

`ori $ra, 0x0004` (16 bits de menor peso de la dirección de memoria)

Inst.	**Definición**	**Formato**
add	Almacena en el registro $1 el valor de la suma de los otros dos	add $1, $2, $3
sub	Almacena en el $1 el valor de la resta de $2 (minuendo), menos el $3 (sustraendo)	sub $1, $2, $3
and	Almacena en el $1 el resultado de hacer una operación AND entre los otros dos registros	and $1, $2, $3
or	Almacena en el $1 el resultado de hacer una operación OR entre los registros $2 y $3	or $1, $2, $3
slt	Coloca un 1 en el registro $1 si el valor almacenado en $2 es menor que el de $3. Si no lo es se almacena un 0 en $1.	slt $1, $2, $3
jr	Modifica la dirección del PC, por aquel valor almacenado en el registro $1	jr $1
div	Se almacena en el HI el resto y en LO el cociente de dividir $1 (dividendo) entre $2(divisor).	div $1, $2.
mult	Se almacena en el registro LO los 32 bits de menor peso y en HI los 32 de mayor peso de la multiplicar $1 y $2.	mult $1, $1.
mfhi	Copia el valor almacenado en el registro HI en $1	mfhi $1.
mflo	Copia el valor almacenado en LO en el registro $1	mflo $1.

Cuadro A.1: Instrucciones tipo R

Características de .data

La sección *.data* contiene la declaración de las variables del programa. Los elementos siguientes se guardan en el segmento de datos, en concreto en la memoria. Esta sección es opcional.

Inst.	**Definición**	**Formato**
addi	Se almacena en el registro $1 la suma del registro $2 y el valor de la constante	addi $1, $2, constante
andi	El registro $1 tiene el resultado de una operación AND entre $2 y la constante	andi $1, $2, constante
ori	El registro $1 tiene el resultado de una operación OR entre $2 y la constante	ori $1, $2, constante
slti	Almacena en $1 un 1 si el valor registro $2 es menor que el de la constante. En caso contrario almacena un 0	slti $1, $1, constante
lui	Carga constante en los 16 bits superiores del registro $1	lui $1, direcConst
lw	Carga en el registro $1 la palabra almacenada en la dirección de memoria que contiene el registro $2 más el desplazamiento. La nueva dirección calculada debe ser múltiplo de cuatro	lw $1, desplazamiento($2)
lb	Carga en el registro $1 el byte de memoria apuntado por la dirección almacenada en el registro $2 más el desplazamiento	lb $1, desplazamiento($2)
sw	Almacena en memoria en la posición obtenida de sumarle el desplazamiento a la dirección del registro $2, la palabra del registro $1. La dirección debe ser múltiplo de 4.	sw $1, desplazamiento($1)
sb	Almacena en la posición de memoria correspondiente al valor de $2 más el desplazamiento, el primer byte de la palabra almacenada en $1.	sb $1, desplazamiento($2).
beq	Si el valor de $1 y $2 es igual se modifica el valor del PC para pasar a ejecutar el trozo de código apuntado por la etiqueta.	beq $1, $1, etiqueta.
bne	Si el valor de $1 y $2 no es igual se modifica el valor del PC para pasar a ejecutar el trozo de código apuntado por la etiqueta.	bne $1, $1, etiqueta.

Cuadro A.2: Instrucciones tipo I

Inst.	**Definición**	**Formato**
j	Modifica el valor del PC para ejecutar las instrucciones siguientes a la etiqueta	j etiqueta
jal	Modifica el valor del PC por aquel al que apunta la etiqueta y almacena la dirección actual del PC en $ra	jal etiqueta

Cuadro A.3: Instrucciones tipo J

La declaración de las variables del programa se ajusta a las siguientes reglas:

- En cada línea no puede haber más de una etiqueta.
- La declaración de una variable sigue este formato:
  - En primer lugar debe ir un identificador de variable válido (etiqueta). Se considera válida cualquier secuencia de caracteres alfanuméricos, subrayados y puntos.
  - A continuación se indica el tipo de variable. La tabla A.8 muestra una relación de los tipos implementados en *Simula3MS*, sus características y la estructura de la definición.
  - Finalmente se inicializa la variable.
- Las cadenas de caracteres se encierran entre comillas dobles.
- Los números se consideran en base 10 por defecto. Si van precedidos del prefijo 0x se interpretan en hexadecimal.

La Figura A.1 muestra un ejemplo de código.

Ensamblar

Una vez editado o cargado el código del programa es necesario ensamblarlo, para ello se utiliza el botón *Ensamblar*. Si el código es sintácticamente correcto se activará el botón *Ejecutar* y se puede avanzar a la simulación del programa, si no es necesario corregir el código teniendo en cuenta los errores.

Inst.	**Definición**	**Formato**
add.s	Se almacena en el registro $f1 el valor de la suma de los otros dos (precisión simple)	add.s $f1, $f2, $f3
add.d	Se almacena en el registro $f0 el valor de la suma de los otros dos (precisión doble)	add.d $f0, $f2, $f4
sub.s	Se almacena en el registro $f1 el valor de la resta de los otros dos (precisión simple)	sub.s $f1, $f2, $f3
sub.d	Se almacena en el registro $f0 el valor de la resta de los otros dos (precisión doble)	sub.d $f0, $f2, $f4
mul.s	Se almacena en el registro $f1 el valor de la multiplicación de los otros dos (precisión simple)	mul.s $f1, $f2, $f3
mul.d	Se almacena en el registro $f0 el valor de la multiplicación de los otros dos (precisión doble)	mul.d $f0, $f2, $f4
div.s	Se almacena en el registro $f1 el valor de la división de los otros dos (precisión simple)	div.s $f1, $f2, $f3
div.d	Se almacena en el registro $f0 el valor de la división de los otros dos (precisión doble)	div.d $f0, $f2, $f4

Cuadro A.4: Instrucciones tipo R con 3 argumentos

Inst.	**Definición**	**Formato**
mov.s	Transfiere el número en punto flotante de precisión simple del registro $f2 al $f1	mov.s $f1, $f2
mov.d	Transfiere el número en punto flotante de precisión doble del registro $f2 al $f0	mov.s $f0, $f2
abs.s	Calcula el valor absoluto del número en punto flotante de precisión simple del registro $f2 y lo almacena en $f1	abs.s $f1, $f2
abs.d	Calcula el valor absoluto del número en punto flotante de precisión doble del registro $f2 y lo almacena en $f0	abs.d $f0, $f2
neg.s	Niega el número en punto flotante de precisión simple del registro $f2 y lo almacena en $f1	neg.s $f1, $f2
neg.d	Niega el número en punto flotante de precisión doble del registro $f2 y lo almacena en $f0	neg.d $f0, $f2
c.eq.s	Si los valores de los registros de precisión simple $f1 y $f2 son iguales se escribe un 1 en el registro status	c.eq.s $f1, $f2
c.eq.d	Si los valores de los registros de precisión doble $f0 y $f2 son iguales se escribe un 1 en el registro status	c.eq.d $f0, $f2
c.le.s	Si el valor del registro $f1 es menor o igual que el del registro $f2, ambos de precisión simple, se escribe un 1 en el registro status	c.le.s $f1, $f2
c.le.d	Si el valor del registro $f0 es menor o igual que el del registro $f2, ambos de precisión doble, se escribe un 1 en el registro status	c.le.d $f0, $f2
c.lt.s	Si el valor del registro $f1 es menor que el del registro $f2, ambos de precisión simple, se escribe un 1 en el registro status	c.lt.s $f1, $f2
c.lt.d	Si el valor del registro $f0 es menor que el del registro $f2, ambos de precisión doble, se escribe un 1 en el registro status	c.lt.d $f0, $f2
cvt.d.s	Convierte el número en punto flotante de precisión simple del registro $f1 a un número de precisión doble y lo guarda en $f2	cvt.d.s $f2, $f1
cvt.s.d	Convierte el número en punto flotante de precisión doble del registro $f2 a un número de precisión simple y lo guarda en $f1	cvt.s.d $f1, $f2

Cuadro A.5: Instrucciones tipo R con 2 argumentos

Inst.	**Definición**	**Formato**
mtc1	Transfieren el registro $t1 de la CPU al registro $f1 del coprocesador de punto flotante	mtc1 $f1, $t1
mfc1	Transfieren el registro $f1 del coprocesador de punto flotante al registro $t1 de la CPU	mfc1 $t1, $f1
cvt.d.w	Convierte el número entero del registro $t1 a un número de precisión doble y lo guarda en $f2	cvt.d.w $f2, $t1
cvt.s.w	Convierte el número entero del registro $t1 a un número de precisión simple y lo guarda en $f1	cvt.s.w $f1, $t1
cvt.w.d	Convierte el número en punto flotante de precisión doble del registro $f2 a un número entero y lo guarda en $t1	cvt.w.d $t1, $f2
cvt.w.s	Convierte el número en punto flotante de precisión simple del registro $f1 a un número entero y lo guarda en $t1	cvt.w.s $t1, $f1

Cuadro A.6: Instrucciones que usan registros enteros

Para facilitar esta tarea podemos ayudarnos del botón *ErrorSiguiente* que, como su nombre indica, avanza al siguiente error de forma cómoda.

A.2.2. Configuración del simulador

Por defecto al inicializar *Simula3MS* la configuración del procesador es monociclo, pero puede cambiarse en cualquier momento, bien antes de editar el código o después de terminar de escribir el programa en ensamblador, o bien después de realizar cualquier simulación previa.

Para cambiar la configuración acceder al menú *Configurar* en la barra de herramientas y seleccionar la opción adecuada. En caso de que la configuración escogida sea la multiciclo, se mostrará una nueva ventana en la que además, podremos configurar la latencia de las unidades funcionales de punto flotante.

Inst.	**Definición**	**Formato**
lwc1	Carga en el registro $f1 la palabra almacenada en la dirección de memoria que contiene el registro $s2 más el desplazamiento.La nueva dirección debe ser múltiplo de 4	lwc1 $f1, desplazamiento($s2)
swc1	Se almacena la palabra del registro $f1 en la posición de memoria obtenida al sumar la dirección que contiene el registro $s2 más el desplazamiento.La dirección debe ser múltiplo de 4	swc1 $f1, desplazamiento($s2)
bc1t	Si status=1 se modifica el valor del PC para ejecutar el trozo de código apuntado por la etiqueta	bc1t etiqueta
bc1f	Si status=0 se modifica el valor del PC para ejecutar el trozo de código apuntado por la etiqueta	bc1f etiqueta

Cuadro A.7: Instrucciones tipo I

Tipo	**Definición**	**Estructura de la definición**
.ascii	Almacena en memoria el string como una lista de caracteres	variable: .ascii "string a almacenar"
.asciiz	Almacena en memoria el string como una lista de caracteres y lo termina con 0	variable: .asciiz "string a almacenar"
.word	Almacena la lista de palabra en posiciones secuenciales de memoria	variable: .word palabra1, palabra2, ...
.space	Reserva n bytes de espacio en la memoria.	variable: .space n
.float	Almacena la lista de números en punto flotante de simple precisión en posiciones sucesivas de memoria.	variable: .float f1, f2, ...
.double	Almacena la lista de números en punto flotante de doble precisión en posiciones sucesivas de memoria.	variable: .double d1, d2, ...

Cuadro A.8: Tipos de datos en *Simula3MS*

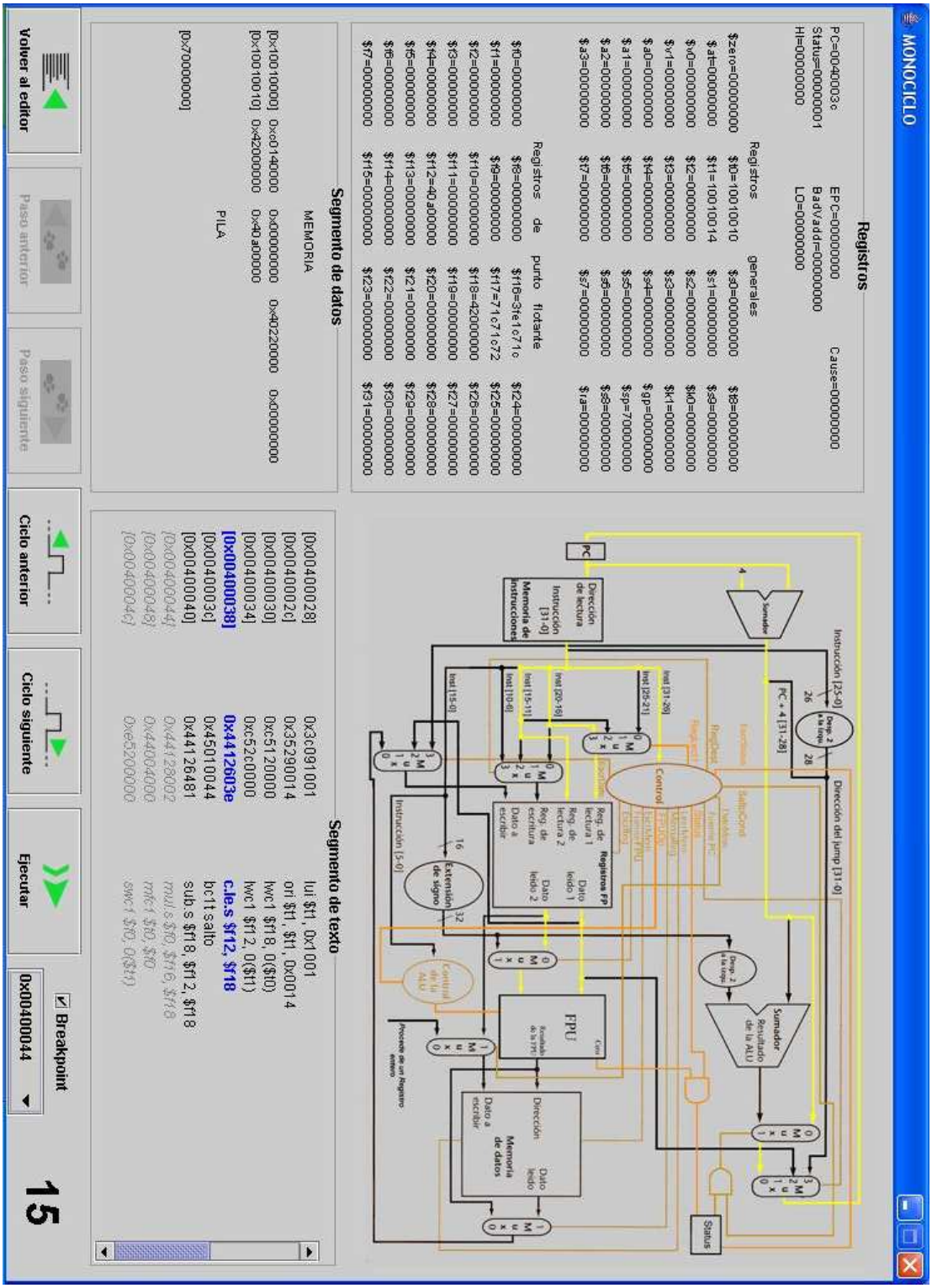

Figura A.2: Ventana de la configuración monociclo

A.3. Ventana de registros

Los registros se agrupan en tres clases:

- **Registros especiales:**
  - *PC* es el contador del programa (*program couter*). Este registro se inicializa por el sistema operativo apuntando a la dirección de la primera instrucción del programa en memoria. Al cargar cualquier instrucción de memoria el PC se incrementa de forma que la CPU tendrá el valor de la dirección de la siguiente instrucción que va a ser cargada.
  - *EPC, Cause, Status* y *BadVaddr* son registros utilizados para el manejo de instrucciones e interrupciones.
    - El registro *EPC* contiene la dirección de la instrucción que ha causado la excepción.
    - El registro *Cause* almacena el tipo de excepción y bits de interrupción pendiente.
    - El registro *Status* se usa como condición de salto en las instrucciones *bc1t* y *bc1f*.
    - El registro *BadVaddr* contiene la dirección de memoria en la que ha ocurrido la referencia de memoria.

    Los registros *EPC*, *Cause* y *BadVaddr* no están implementados actualmente en *Simula3MS*.
  - *Hi* y *Lo* son dos registros para poder operar con operandos de 64 bits, como sucede en el caso de la multiplicación y la división.
- **Registros generales:** La arquitectura MIPS posee 32 registros genéricos de 32 bits ($0-$31) para uso del programador (ver tabla A.9). Existen reglas de *uso de los registros*, también llamadas *convención de llamada a procedimiento*, que indican cual debe ser el uso de los registros, especialmente en las llamadas a procedimientos, que hace el programa. Tal como indica su nombre, estas reglas son, en su mayor parte, convenciones seguidas por los programas más que reglas forzadas por la circuitería. De todos modos la mayoría de los programadores se esfuerzan por seguir estas convenciones porque violarlas sería fuente de un mayor número de errores.

Nombre	**Número**	**Uso**
$zero	0	Valor constante 0. Sólo es de lectura.
$at	1	Reservado por el ensamblador.
$v0-$v1	2-3	Valores para resultados y evaluación de expresiones.
$a0-$a3	4-7	Argumentos.
$t0-$t7	8-15	Registros temporales.
$s0-$s7	16-23	Registros salvados.
$t8-$t9	24-25	Registros temporales.
$k0-$k1	26-27	Reservados por el Sistema Operativo.
$gp	28	Puntero global. No implementado en *Simula3MS*.
$sp	29	Puntero de pila.
$fp	30	Puntero de bloque de activación. No implementado en *Simula3MS*.
$ra	31	Dirección de retorno de las subrutinas.

Cuadro A.9: Convención de los registros MIPS

- **Registros de punto flotante:** La arquitectura MIPS R2000 (que simula esta herramienta) no dispone de unidad de punto flotante incluida en el microprocesador si no que implementa estas funciones en coprocesadores separados. La arquitectura MIPS tiene en cada coprocesador 32 registros de 32 bits para punto flotante ($f0-$f31), que pueden ser organizados en 16 registros de doble precisión, con 64 bits (para ello se toman las designaciones par de los registros).

A.4. Memoria

Los sistemas basados en procesadores MIPS habitualmente dividen la memoria en tres partes.

- La primera parte, junto al extremo inferior del espacio de direcciones, es el *segmento de texto*, que contiene las direcciones del programa y empieza en la dirección 0x00400000.
- La segunda parte, sobre el segmento de texto, es el *segmento de datos*, que se divide en dos partes:

  - Los *datos estáticos*, a partir de la dirección 0x10010000. Estos datos contienen objetos cuyo tamaño es conocido por el compilador y su tiempo de vida es la ejecución entera del programa.
  - Inmediatamente después de los datos estáticos están los *datos dinámicos*. Como su nombre indica, estos datos son reservados por el programa durante su ejecución. Como el compilador no puede predecir cuánta memoria va a reservar un programa el sistema operativo extiende el área de datos dinámicos para satisfacer la demanda.

- La tercera parte, la *pila* del programa reside en el extremo superior del espacio de direcciones virtual, situado en la dirección 0x70000000. El tamaño máximo de la pila que usará un programa no se puede conocer previamente, por lo tanto es necesario que el usuario desplace el puntero de pila hacia abajo antes de insertar nuevos elementos.

Esta división de la memoria en tres partes no es la única posible. De todos modos, tiene como característica más importante que la pila y la memoria están lo más alejadas posibles, de forma que la pila puede crecer hasta ocupar el espacio de direcciones del programa por entero.

A.4.1. Segmento de datos

En la representación de *Simula3MS*, la pila y el segmento de datos explicado anteriormente se agrupan en *segmento de datos* ya que ambos almacenan los datos necesarios durante la ejecución del programa.

En *Simula3MS* la memoria está dividida en palabras de 32 bits, por lo que hay 4 bytes almacenado en cada palabra. Debido a esta organización de la memoria nos encontramos con un requisito denominado *restricción de alineación* para el acceso a las palabras de memoria, ya que las direcciones deben ser siempre múltiplo de 4 y la dirección de una palabra es la de su primer byte. Esta restricción de alineación facilita la transferencia de datos más rápidamente.

Otro aspecto curioso está relacionado con la forma en que se representan y referencian los bytes dentro de una palabra. En *Simula3MS* se sigue el convenio *Little Endian* ya que se usa la dirección del byte de más a la derecha o de menor peso como dirección de palabra.

Cada línea del segmento de texto de *Simula3MS* (ver figura A.2) sigue el formato:

El primer valor, entre corchetes, indica la dirección en memoria a partir de la cual se almacena la primera palabra, las otras palabras están situadas en posiciones sucesivas de memoria.

Las restantes cuatro palabras representan los valores, en hexadecimal, de los datos almacenados en esas direcciones.

En la pila el formato es aproximadamente el mismo, pero sólo se indica la dirección a la que apunta el puntero de pila. Hay que recordar que para que la pila crezca es necesario desplazar el puntero de pila hacia posiciones inferiores.

A.4.2. Segmento de texto

El *segmento de texto* muestra las instrucciones del programa que se cargan automáticamente cuando se empieza la simulación. Cada instrucción se muestra en una línea (ver figura A.2) de la siguiente forma:

- El primer número de la línea, entre corchetes, es la dirección de memoria hexadecimal de la instrucción.
- El segundo número es la codificación numérica de la instrucción, es decir, el valor que almacenaría el procesador para la instrucción en lenguaje máquina (también se muestra en hexadecimal).
- En último lugar se muestra la descripción de la instrucción. En todos los casos la descripción de la instrucción coincide con la instrucción del código del programa, excepto en el caso de la pseudoinstrucción *la*, la única aceptada por *Simula3MS*.

A.5. Llamadas al sistema

Simuala3MS ofrece un pequeño conjunto de servicios de llamada al sistema operativo a través de la instrucción de llamada al sistema (*syscall*). Para pedir un servicio el programa carga el código de llamada al sistema (tabla A.10) en el registro $v0 y los argumentos en los registros $a0 y $a1

(o $f12 para valores de punto flotante). Las llamadas al sistema que devuelven valores ponen sus resultados en el registro $v0 (o $f0 para resultados de punto flotante).

Existen llamadas al sistema para salida de datos:

- A la llamada al sistema *print_ int* se le pasa un valor y lo muestra como un entero en un diálogo de información.
- *print_ float* muestra en el diálogo el valor en punto flotante que se le pasa.
- *print_ double* hace lo mismo que *print_ float* pero permite un rango más amplio de números en punto flotante.
- *print_ string* muestra en el diálogo la cadena almacenada a partir de la dirección que indica $a0 hasta que encuentra el carácter de finalización de cadena.

También hay llamadas al sistema para entrada de datos, estos datos se solicitan al usuario a través de diálogos. Son:

- *read_ int* almacena el entero que introduce el usuario en $v0.
- *read_ float* almacena el número de punto flotante que le introduce el usuario en el registro $f0 con formato de simple precisión.
- *read_ double* almacena el número de punto flotante que le introduce el usuario en los registros $f0 y $f1, puesto que el formato de doble precisión necesita dos registros para almacenar un valor.
- *read_ string* almacena la dirección de comienzo de la cadena que introduce el usuario en el registro $a0 y la longitud de dicha cadena en $a1. Estos datos son reservados por el programa durante su ejecución y por tanto se consideran datos dinámicos.

Otra llamada al sistema muy utilizada es *exit* utilizada para indicar al procesador que se ha terminado la ejecución del programa.

Código	**Nombre**	**Operación**
1	*print_int*	Imprime como un entero aquello que se encuentra en $a0.
2	*print_float*	Imprime como un número en punto flotante de aquello que se encuentra en $f12.
3	*print_double*	Imprime como un número en punto flotante aquello que se encuentra en $f12 y $f13, considerando que forman un sólo registro de doble precisión.
4	*print_string*	Imprime como un string aquello que se encuentra en la posición indicada por $a0.
5	*read_int*	Solicita un entero que se almacenará en el registro $v0.
6	*read_float*	Solicita un número en punto flotante que se almacenará en el registro $f0 con formato de simple precisión.
7	*read_double*	Solicita un número en punto flotante que se almacenará en el registro $f0 (y $f1) con formato de doble precisión.
8	*read_string*	Solicita un string que se almacena en $a0 y cuya longitud se guarda en $a1.
10	*exit*	Finaliza la ejecución.

Cuadro A.10: Códigos asociados a syscall

A.6. Punto de ruptura

Los puntos de ruptura se utilizan para parar la ejecución del simulador en una instrucción determinada.

En el caso de *Simula3MS* pueden insertarse y desactivarse en cualquier momento de la ejecución mediante un menú desplegable que contiene una lista de las direcciones de memoria donde están almacenadas las instrucciones. Esta lista de direcciones comprende desde la de la instrucción siguiente a la que está en ejecución hasta la última del código.

A.7. Simulación con el procesador monociclo

Un procesador monociclo se caracteriza por la ejecución de cada instrucción en un ciclo de reloj.

La simulación de este procesador en *Simula3MS* se representa en la figura A.2 cuyas características han sido detalladas en los apartados anteriores.

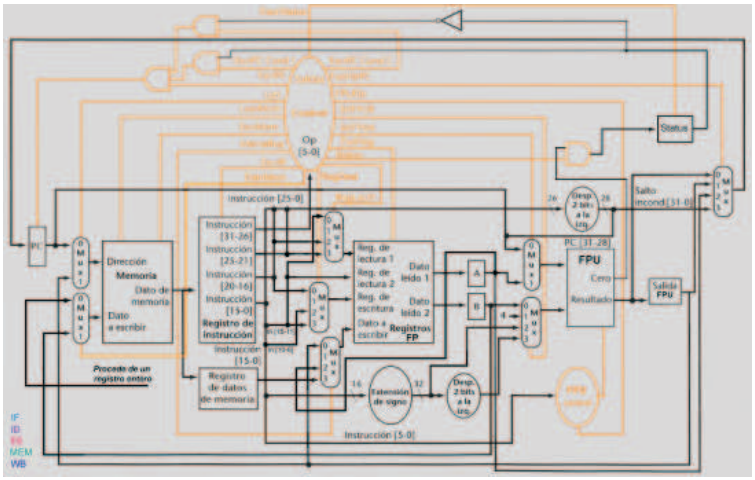

Figura A.3: Primer modelo de camino de datos

Tanto en el procesador monociclo como en el muticiclo, podemos distinguir 3 tipos de caminos de datos. En primer lugar están los correspondientes a instrucciones enteras (ver figura A.3) que constan de una ALU y un banco de registros enteros.

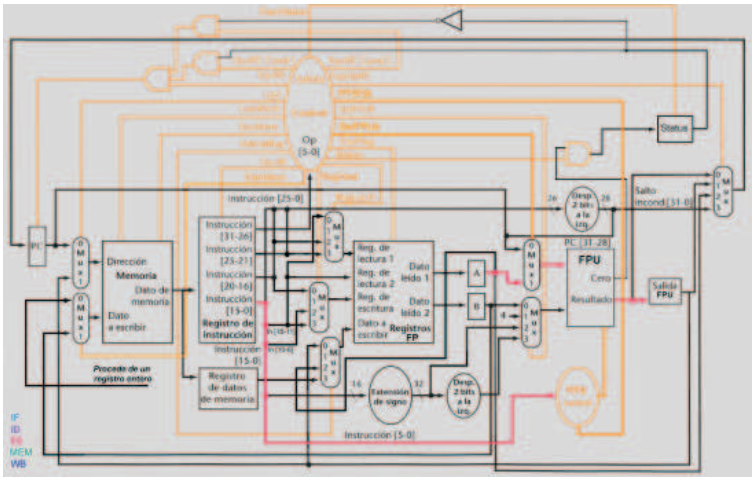

Figura A.4: Segundo modelo de camino de datos

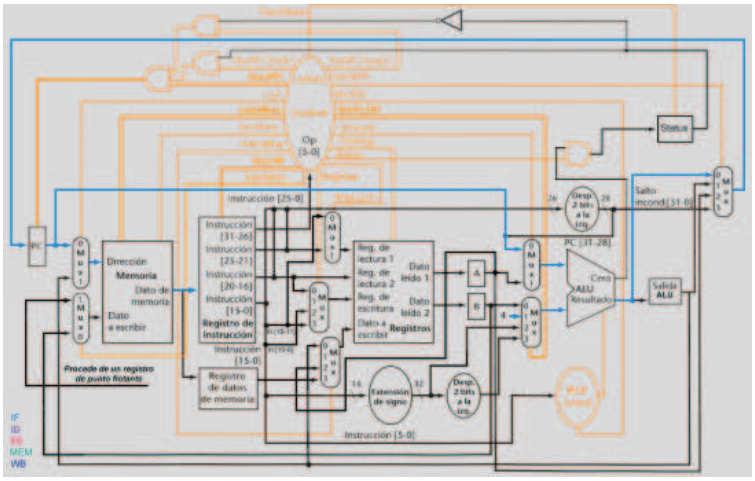

Figura A.5: Tercer modelo de camino de datos

La mayoría de las instrucciones de punto flotante usan un camino de datos que substituye la ALU por una FPU (Floating Point Unit) y los registros enteros por los de punto flotante. Este modelo incluye además la representación del registro especial *Status* (ver figura A.4).

Por último, existe un modelo híbrido de los dos. Se trata de un camino de datos con una ALU y un banco de registros enteros, pero incluyendo el registro Status (con sus líneas de control correspondientes) y unas líneas que nos indican cuando leer o escribir en un registro de punto flotante. Este modelo (figura A.5) es necesario en instrucciones de punto flotante que operen con ambos tipos de registros, como pueden ser las instrucciones de carga o almacenamiento de punto flotante.

El funcionamiento de los botones de esta herramienta es el siguiente:

Volver al editor. En el caso de que se produzca algún error en ejecución será el único botón que permanecerá activado.

Paso anterior. Sólo aparece habilitado en la configuración multiciclo.

Paso siguiente. Sólo aparece habilitado en la configuración multiciclo.

Ciclo anterior. Retrocede un ciclo en la ejecución. Si se trata de una operación aritmética en punto flotante y se encuentra en la etapa de ejecución del simulador multiciclo, retrocederá el número de ciclos que se haya especificado en la configuración.

Ciclo siguiente. Avanza un ciclo en la ejecución. Si se trata de una operación aritmética en punto flotante y se encuentra en la etapa 2 del simulador multiciclo, avanzará el número de ciclos que se haya especificado en la configuración.

Ejecutar. Ejecuta la totalidad del código, a menos que se haya insertado un punto de ruptura en cuyo caso se ejecutará hasta ese punto.

Breakpoint. Si el botón de selección correspondiente está activado se despliega una lista con los PCs comprendidos entre el de la instrucción que en ese momento se está ejecutando y el último. La identificación de las instrucciones por medio de su PC correspondiente no resulta complicado, ya que en el segmento de texto ambos aparecen relacionados.

Ciclos Contador del número de ciclos que ha consumido la ejecución de las instrucciones hasta ese momento.

A.8. Simulación con el procesador multiciclo

Un procesador multiciclo se caracteriza por subdividir las instrucciones en diferentes etapas, cada una de las cuales se ejecuta en un ciclo de reloj. Si la instrucción que se está ejecutando es una suma, resta, multiplicación o división en punto flotante, la etapa de ejecución ocupará tantos ciclos de reloj como se le especifique en la configuración.

El entorno gráfico asociado a esta simulación (ver figura A.6) es análogo al de una simulación monociclo, la diferencia más significativa es la activación de los botones *Paso siguiente* y *Paso anterior*.

Paso anterior. Sitúa la ejecución en el primer ciclo de la instrucción anterior.

Paso siguiente. Sitúa la ejecución en el último ciclo de la instrucción actual. Si ésta instrucción está en el último ciclo avanza hasta el último de la instrucción siguiente.

A.9. Simulación con el procesador segmentado

Un procesador segmentado se caracteriza por la ejecución solapada de diferentes instrucciones.

Es necesario destacar que:

- Los saltos se deciden en la segunda etapa, y se ha implementado la estrategia de salto retardado, por tanto la instrucción que va a continuación del salto se ejecutará siempre, tanto si se realiza el salto como si no se realiza. En ocasiones será útil utilizar una instrucción *nop* después de instrucciones de salto para evitar que el programa funcione erróneamente.
- Para evitar riesgos el procesador inserta burbujas.
- Antes de realizar una llamada al sistema el procesador espera a que el pipeline se vacíe.
- Hemos considerado que la ejecución de una llamada al sistema consume un ciclo de reloj y no contabilizamos los ciclos que consume el SO al realizar el servicio requerido.

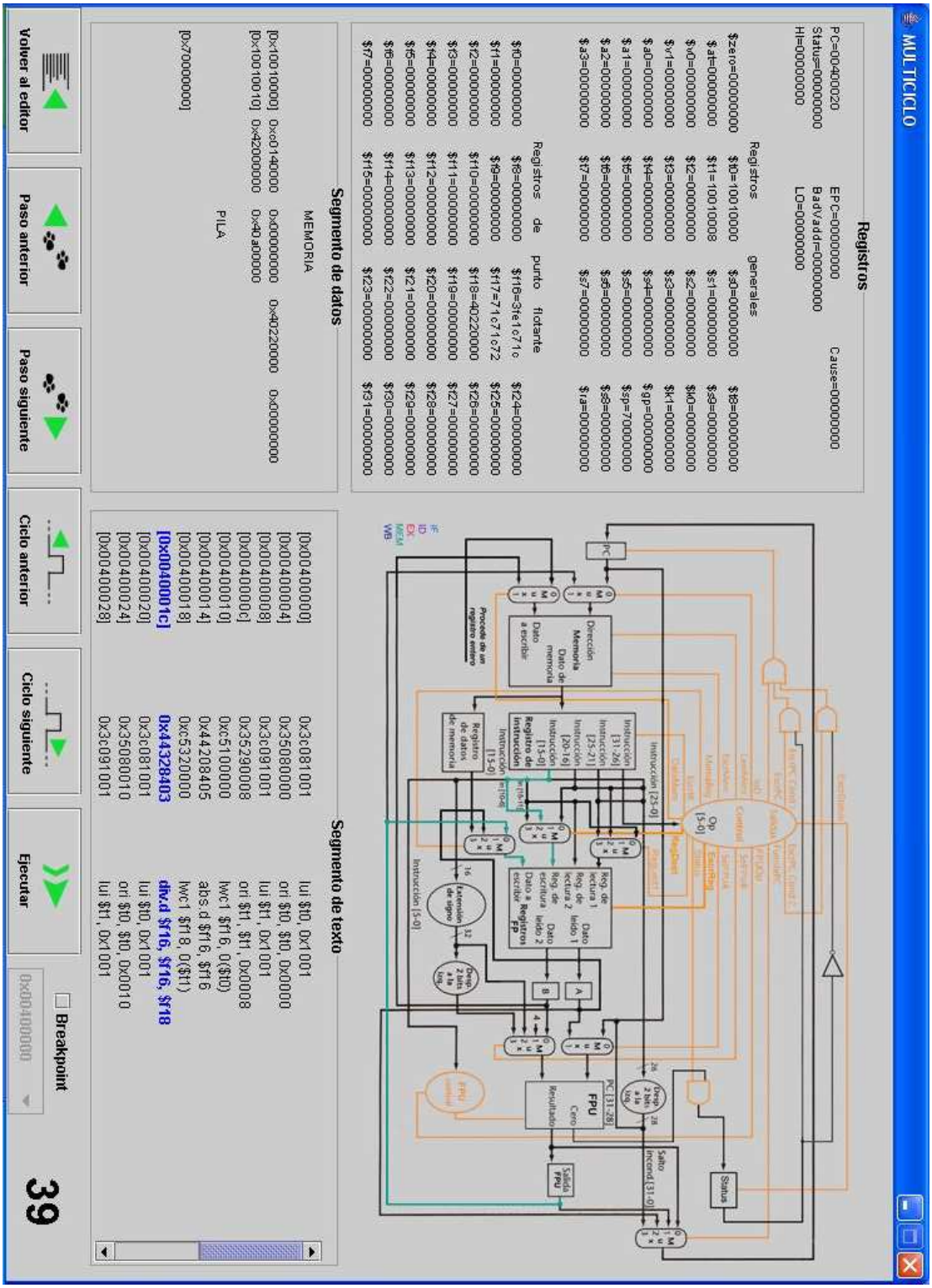

Figura A.6: Configuración multiciclo

- Después de la llamada al sistema se sigue con la ejecución del resto del código.
- Si la ejecución del programa en ensamblador no terminase con la llamada al sistema de finalización, sino porque en el segmento de texto no hay más instrucciones o porque se ha insertado un punto de ruptura, el pipeline parará antes de terminar la ejecución de todas las instrucciones, por lo que no se ejecutará el programa completo.
- La segmentación se puede representar gráficamente mediante dos tipos de diagramas.
  - Los **diagramas uniciclo** muestran el estado del camino de datos durante un ciclo de reloj, y las cinco instrucciones del pipeline se identifican con etiquetas encima de sus correspondientes etapas. Se usa este tipo de diagrama para mostrar con más detalle que está ocurriendo dentro del pipeline durante cada ciclo del reloj, ver figura A.7.
  - Los **diagramas multiciclo** se utilizan para dar una perspectiva general de diferentes situaciones dentro de la segmentación. Se considera que el tiempo avanza de izquierda a derecha y las instrucciones avanzan de la parte superior a la inferior del panel, ver figura A.8.
- El simulador dispone de botones que permiten ejecutar el programa escrito en ensamblador completo, o bien ciclo a ciclo. También se tiene la posibilidad de retroceder a ciclos anteriores. En cualquier momento de la ejecución se podrá poner un punto de ruptura.

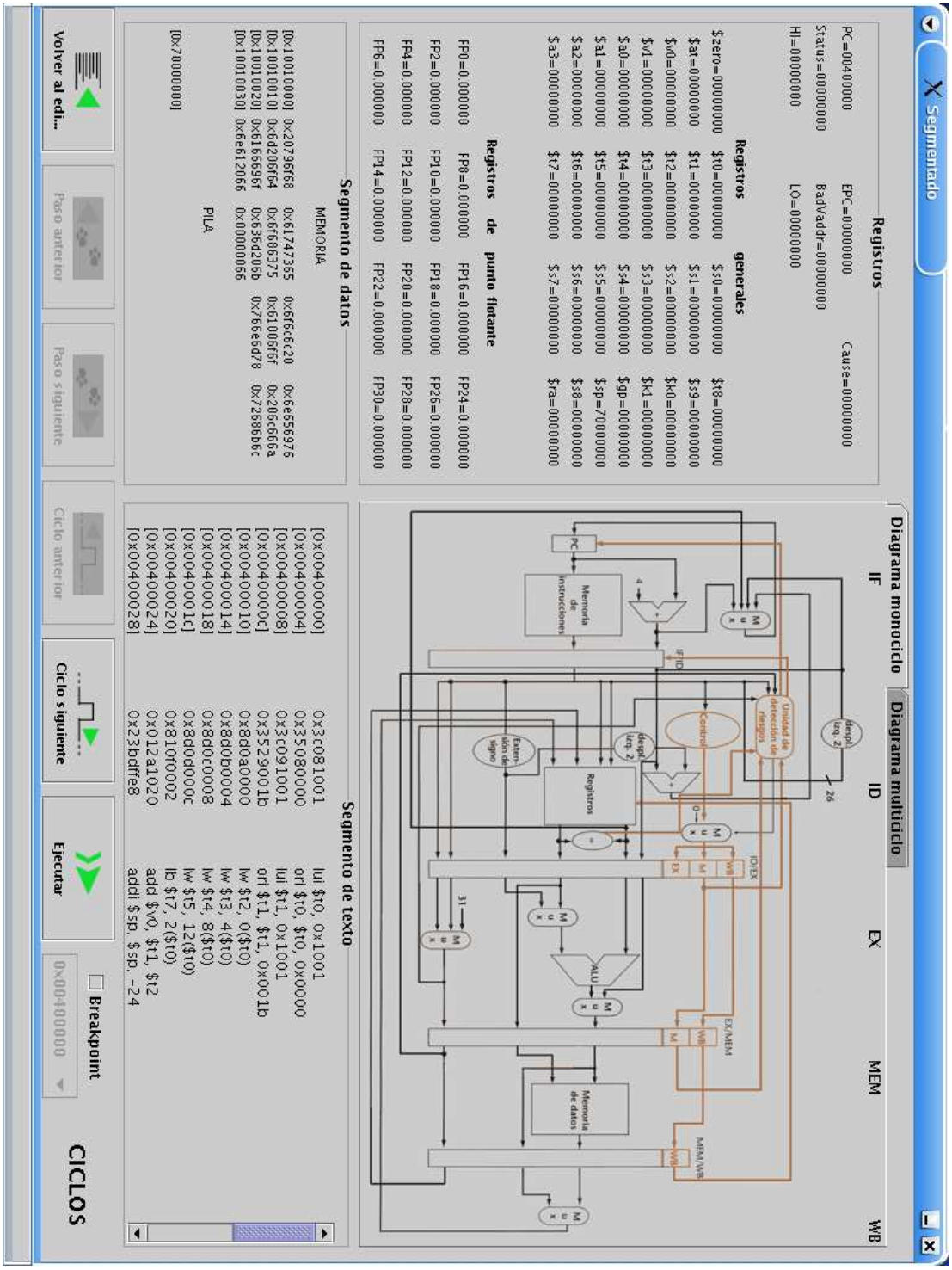

Figura A.7: Diagrama para un solo ciclo de ejecución en el camino segmentado

Figura A.8: Diagrama que muestra multiples ciclos en el camino segmentado