

● ● ● ● ● ●

•  
•  
•  
•  
•  
•

Apéndice4.3: Especificación semántica de las instrucciones	187
Apéndice4.4: Control del encaminamiento por instrucción	191
Apéndice4.5: Señales de error debido al formato de la instrucción	193
Apéndice4.6: Señales de control de la ALU	195
Apéndice4.7: Señales de control de la memoria de datos	197
Apéndice4.8: Señales de control del secuenciamiento condicional	199
Apéndice4.9: Organización de las librerías y ficheros utilizados en el diseño del procesador	201
Memoria de instrucciones: libMI	202
Unidad aritmético-lógica: libalu	203
Memoria de datos: libMD	206
Decodificador de instrucciones: libdeco	209
Elementos: libelem	224
Banco de registros: libBR	226
Secuenciamiento de instrucciones: libsecu	227
Contador de programa: libCP	228
Apéndice4.10: Retardos	231
Apéndice4.11: Impresión de la información del camino de datos	233
Formato de la impresión	233
Apéndice4.12: Programas de prueba	235
Simpletest	235
Test_alu	240
Test_branch	241
Test_mem	243
Suma de los elementos de un vector	244
Ordenación de los elementos de un vector	245
Multiplicación en base 2	246
Multiplicación en base 4	248
Factorial	249



## Práctica 4

### Procesador: arquitectura, camino de datos y control

.....

En esta sesión se trata de consolidar los conocimientos adquiridos sobre el camino de datos de un procesador y el control del mismo.

Para ello utilizaremos parte del repertorio de instrucciones MIPS32. Este repertorio de instrucciones, el cual se describe, tiene como peculiaridad la utilización de retardo en la modificación del secuenciamiento al interpretar una instrucción de secuenciamiento.

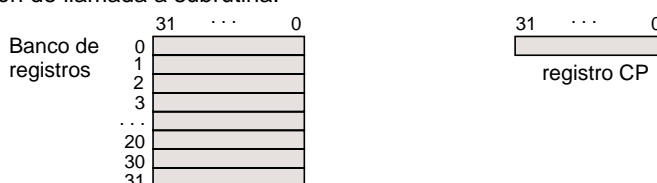
El objetivo de la práctica es identificar los componentes del camino de datos que utiliza cada tipo de instrucción y el control del mismo. Así mismo, teniendo en cuenta el retardo de cada componente del camino de datos, se analiza el camino crítico para cada instrucción. El objetivo es que cualquier instrucción se pueda interpretar en un ciclo de reloj. Finalmente se determina el valor mínimo del periodo de la señal de reloj y la duración de la misma en cada uno de los dos niveles lógicos.

### Descripción de la arquitectura del procesador

Todas las instrucciones del repertorio de instrucciones MIPS32 están codificadas utilizando 4 bytes y el camino de datos del procesador es de 32 bits.

La arquitectura define 32 registros (R0, . . ., R31) de 32 bits, que son accesibles mediante campos especificados en una instrucción (banco de registros, Figura 4.1). El registro R0 es especial y siempre tiene el valor cero. Cualquier actualización del registro R0 se ignora. El registro R31 puede ser utilizado explícitamente por cualquier instrucción, pero la mayoría de las instrucciones de llamada a subrutina lo utilizan de forma implícita para almacenar la dirección de retorno.

El registro contador de programa (CP) no pertenece al banco de registros (Figura 4.1). Los dos bits menos significativos de este registro son siempre cero, ya que las instrucciones deben estar alineadas a cuatro bytes en el espacio lógico. Este registro lo puede actualizar una instrucción de secuenciamiento condicional o incondicional y lo puede leer una instrucción de llamada a subrutina.



**Figura 4.1** Banco de registros y registro CP.

El lenguaje máquina MIPS32 tiene instrucciones para leer o escribir datos en memoria (MEM), instrucciones para efectuar cálculos aritmético-lógicos (CAL) e instrucciones que permiten modificar el secuenciamiento implícito (MS). En la Figura 4.2 se muestran los formato base de las instrucciones.

		Campos de la instrucción																						
		31	...	26	25	...	21	20	...	16	15	...	11	10	...	6	5	...	0	identificador de bit				
Formato		6				5				5				5				6				Ejemplos de utilización		
R		Co				rs				rt				rd				shamt				func	CAL, MS	
I		Co				rs				rt				offset										CAL, MEM, MS
J		Co				address														MS				

**Figura 4.2** Formato de las instrucciones de lenguaje máquina. Todas las instrucciones se codifican utilizando 4 bytes, con un campo de código de operación de 6 bits, ubicado en los bits <31:26> de la instrucción.

Seguidamente se describen para cada uno de los formatos base ejemplos de instrucciones que los utilizan. Posteriormente se describen conjuntos de instrucciones por su funcionalidad. La descripción de todas las instrucciones consideradas en esta práctica está en el "Especificación semántica de las instrucciones". El superíndice v en un identificador de registro indica contenido del registro.

**Instrucciones de tipo R.** La operación de la instrucción se codifica en los campos Co y funct. El valor del campo Co en todas las instrucciones de tipo R es cero (Figura 4.3). Entonces, la operación está codificada en el campo funct. Los operandos están codificados en los campos rs, rt y rd que se interpretan como identificadores de registro. Los campos rs y rt son identificadores de registro fuente y el campo rd es identificador de registro destino. El campo shamt sólo se utiliza en instrucciones de desplazamiento. En las otras instrucciones de tipo R este campo es igual a cero. En una instrucción de llamada a procedimiento, la dirección de retorno que se almacena es la dirección de la instrucción almacenada a una distancia de dos instrucciones de la instrucción de secuenciamiento en el espacio lógico. Esto es, se almacena como dirección de retorno la direc-

ción de la instrucción de secuenciamiento más 8 bytes. Esta característica está definida en la arquitectura como retardo en el establecimiento del secuenciamiento, ya que la instrucción que sigue a una instrucción de secuenciamiento siempre se interpreta.

Descripción	Operandos	Especificación semántica
suma algebraica (add)	rs rt rd 0	$rd^V = rs^V + rt^V$
desplazamiento lógico a la izquierda (sll)	rt rd shamt	$rd^V = rt^V \ll shamt$ ; shamt es un natural
operación or bit a bit (or)	rs rt rd 0	$rd^V = rs^V \text{ or } rt^V$
llamada a subrutina de forma indexada (jal)	rs 0 rd 0	$rd^V = CP^V + 8$ ; $CP^V = rs^V$

**Figura 4.3** Ejemplos de instrucciones de tipo R. La dirección de una instrucción está almacenada en el registro CP. Entre paréntesis se muestra el nemotécnico de la instrucción.

**Instrucciones de tipo I.** La operación está codificada en el campo Co. Estas instrucciones utilizan como operandos fuente un registro (campo rs) y el campo offset especificado en la instrucción (Figura 4.4). El campo rt especifica el identificador del registro destino. Sin embargo, en función del campo Co, el campo rt, puede ser el identificador de un registro fuente (por ejemplo, en la instrucción beq en la Figura 4.4). El campo offset se interpreta como un número natural en instrucciones que especifican una operación lógica y en complemento a dos en los otros casos. Podemos considerar incluidas en este formato a las instrucciones de secuenciamiento que tienen como código de operación '000001' y utilizan el campo rt para una posterior decodificación (bltzal). Respecto a las instrucciones de secuenciamiento, recordemos que el efecto de una instrucción de secuenciamiento tiene lugar después de ejecutar la siguiente instrucción en el espacio lógico. Es por ello que el desplazamiento (offset) especificado en la instrucción es relativo a la dirección en la cual se almacena la siguiente instrucción en el espacio lógico.

Descripción	Operandos	Especificación semántica
suma con literal (addi)	rs rt offset	$rt^V = rs^V + \text{ExtSig}[\text{offset}]$ ; offset es un entero
operación and bit a bit con literal (andi)	rs rt offset	$rt^V = rs^V \text{ and } \text{ExtCero}[\text{offset}]$ ; offset es un natural
lectura de memoria (lw)	rs rt offset	$rt^V = M[rs^V + \text{ExtSig}(\text{offset})]$ ; offset es un entero
escritura en memoria (sw)	rs rt offset	$M[rs^V + \text{ExtSig}(\text{offset})] = rt^V$ ; offset es un entero
secuenciamiento condicional (beq)	rs rt offset	if $(rs^V = rt^V)$ then $CP^V = CP^V + 4 + \text{ExtSig}(\text{offset})$ ; else $CP^V = CP^V + 8$
secuenciamiento condicional (bgtz)	rs 0 offset	if $(rs^V > 0)$ then $CP^V = CP^V + 4 + \text{ExtSig}(\text{offset})$ ; else $CP^V = CP^V + 8$
llamada a subrutina condicional (bltzal)	rs rt* offset	$r31^V = CP^V + 8$ , if $(rs^V < 0)$ then $CP^V = CP^V + 4 + \text{ExtSig}(\text{offset})$ ; else $CP^V = CP^V + 8$

**Figura 4.4** Ejemplos de instrucciones de tipo I. (\*) En la instrucción bltzal el campo rt tiene el valor '000001'. Una extensión de signo hasta completar un operando de 32 bits se identifica como ExtSig. Añadir cero en los bits más significativos hasta completar un operando de 32 bits se identifica como ExtCero.  $M[. . .]$  indica acceso a memoria. CP almacena la dirección de la instrucción.

**Instrucciones de tipo J.** Este tipo de instrucciones sólo se utiliza para modificar el secuenciamiento (Figura 4.5). El campo offset, junto con los cuatro bits más significativos de la dirección de la siguiente instrucción, se utiliza para construir una dirección efectiva para buscar una instrucción. Recordemos que el efecto de una instrucción de secuenciamiento tiene lugar después de ejecutar la siguiente instrucción en el espacio lógico.

Descripción	Operandos	Especificación semántica
Secuenciamiento incondicional (j)	offset	$CP^V = (CP^V + 4)_{ 31...28} \& (\text{offset} \ll 2) \& '00'$
Llamada a subrutina (jal)	offset	$r31^V = CP^V + 8; CP^V = (CP^V + 4)_{ 31...28} \& (\text{offset} \ll 2) \& '00'$

**Figura 4.5** Ejemplos de instrucciones de tipo J. El símbolo & indica concatenar. Los dos bits menos significativos de la dirección de una instrucción deben ser ceros al tener que estar alineada a cuatro bytes. La especificación  $|_{31...28}$  indica los cuatro bits más significativos.

## Instrucciones de cálculo

Las instrucciones de cálculo especifican como operandos fuente dos registros del banco de registros o un registro del banco de registros y un literal (campos offset o shamt). El resultado del cálculo se almacena en un registro del banco. Ejemplos de ellas son las tres primeras instrucciones de la Figura 4.3 y las dos primeras instrucciones de la Figura 4.4.

## Instrucciones de secuenciamiento

Las instrucciones de secuenciamiento condicional especifican dos registros o un registro del banco de registros y utilizan el formato R o I. En el primer caso, la condición que se evalúa es el resultado de comparar el contenido de los dos registros (rs y rt, Figura 4.4). En el segundo caso se evalúa si el contenido del registro es mayor que cero o el complementario. Para establecer el valor del registro, en este segundo caso, se dispone de instrucciones de comparación, que se han ejecutado previamente. Ejemplos de instrucciones de secuenciamiento son la última instrucción de la Figura 4.3 y las tres últimas instrucciones de la Figura 4.4. En la Figura 4.5 también se muestran dos instrucciones de secuenciamiento incondicional.

**Instrucciones de comparación.** En el lenguaje máquina se dispone de instrucciones que permiten comparar el contenido de dos registros interpretando el contenido en complemento a dos o como números naturales (Figura 4.6).

Descripción	Operandos	Especificación semántica
Comparación de menor. Interpretando el contenido de los registros en complemento a dos (slt) o como naturales (sltu)	rs   rt   rd	if ( $rs^V < rt^V$ ) then $rd^V = 1$ ; else $rd^V = 0$

**Figura 4.6** Instrucciones de comparación

Una instrucción de secuenciamiento incondicional se implementa mediante una instrucción de secuenciamiento condicional comparando el registro R0 con el registro R0.

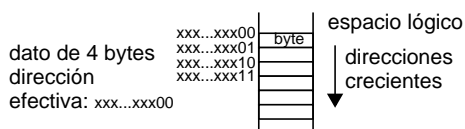
También se dispone de instrucciones que, además de modificar el secuenciamiento, permiten almacenar la dirección de una instrucción, denominada dirección de retorno. El objetivo es modificar el flujo de instrucciones que se está interpretando en secuencia y volver a él, una vez se ha ejecutado un conjunto de instrucciones almacenada en otro trozo del espacio lógico (subrutinas). Ejemplos son las últimas instrucciones de la Figura 4.3, de la Figura 4.4 y de la Figura 4.5.

## Instrucciones de acceso a memoria

Las instrucciones de acceso a memoria utilizan el formato I. El contenido del registro fuente rs se suma al campo offset formateado (se interpreta en complemento a dos) para calcular una dirección efectiva que se utiliza para acceder a memoria. El segundo identificador de registro rt se utiliza como registro destino en una instrucción load y como registro fuente en una instrucción store. Ejemplos son la tercera y cuarta instrucción de la Figura 4.4.

Un dirección efectiva de memoria especifica la dirección de una posición de almacenamiento que contiene un byte. Las instrucciones load y store que leen o escriben 32 bits deben tener alineada la dirección a 4 bytes y las que leen o escriben 16 bits deben tener alineada la dirección a 2 bytes.

El formato en el cual se almacenan datos de tamaño mayor que un byte se denomina “little-endian” (Figura 4.7). El byte de menor ponderación de un dato se almacena en la dirección menor de las direcciones que permiten acceder a los bytes individuales del dato. El resto de bytes se almacena de forma consecutiva siguiendo la ponderación. En estas condiciones, en un acceso a memoria la dirección efectiva es la del byte de menor peso.



**Figura 4.7** Formato de almacenamiento “little-endian”.

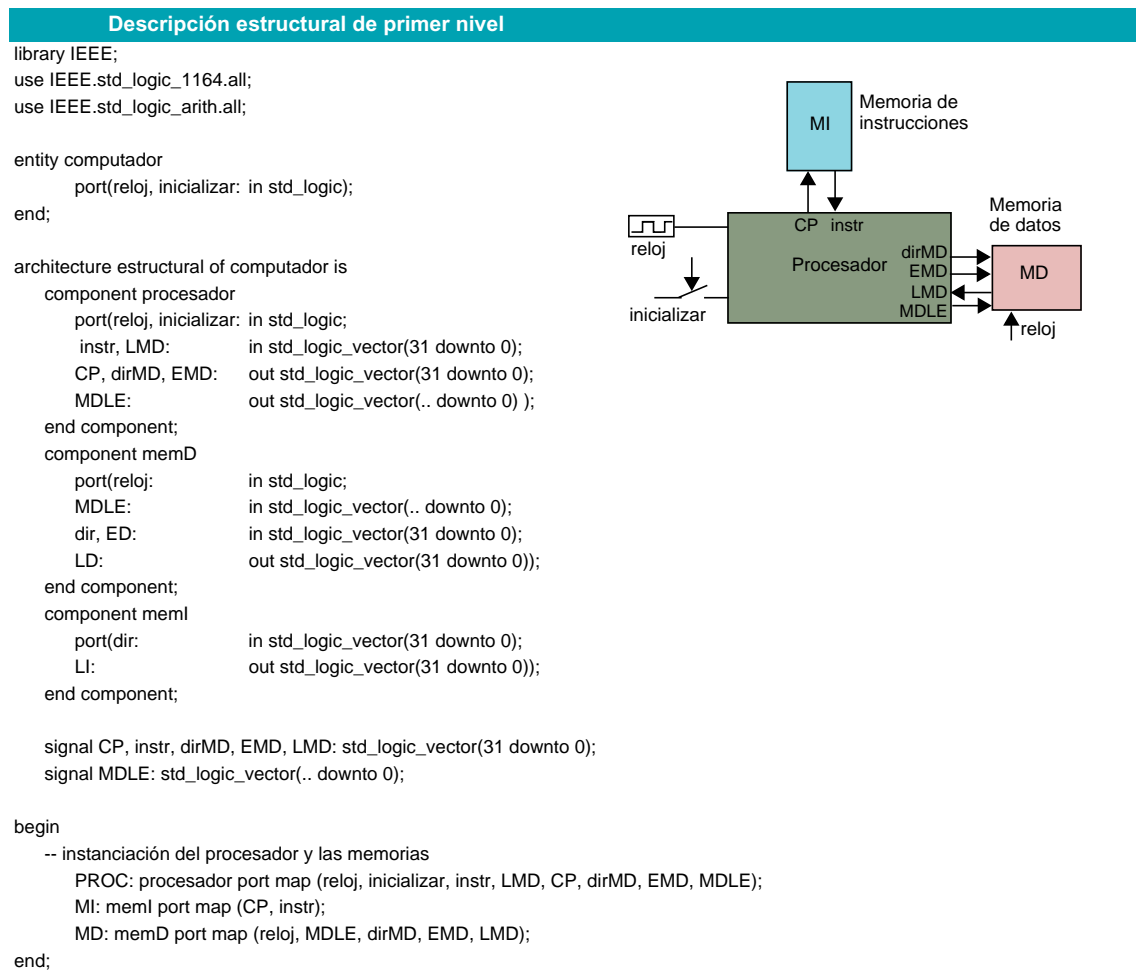
## Microarquitectura

En esta práctica nos centraremos en la microarquitectura de un procesador. La microarquitectura es el nivel existente entre la arquitectura o lenguaje máquina de un procesador y los elementos lógicos que se utilizan para construirla.

Para efectuar la descripción utilizaremos niveles de abstracción. Empezaremos desde el más externo para llegar al de componentes como el banco de registros y la ALU.

## Primer nivel

En el nivel de abstracción más externo distinguimos la unidad procesador y los elementos de memorización de instrucciones y datos (Figura 4.8). La unidad procesador utiliza un registro denominado contador de programa (CP) para leer una instrucción almacenada en la memoria de instrucciones (MI) y la procesa. El procesamiento de la instrucción actualiza el estado del procesador<sup>1</sup> y en ocasiones la memoria de datos (MD). Algunas instrucciones requieren al ser procesadas leer información almacenada en la memoria de datos.



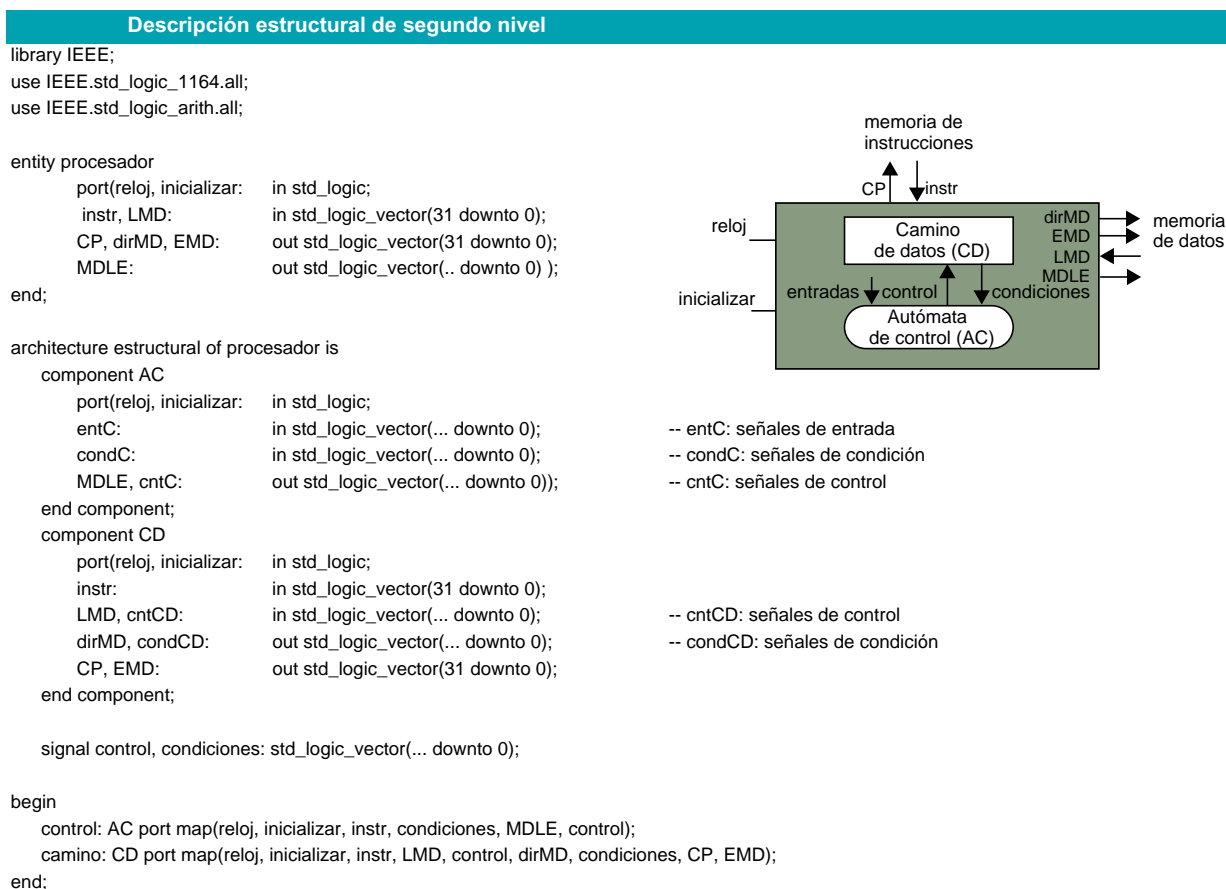
**Figura 4.8** Nivel de descripción más externo de un computador.

1. El estado de un procesador está determinado por el contenido de los registros del banco de registros y el contador de programa. El estado de la arquitectura incluye la memoria de datos.



## Segundo nivel

En el siguiente nivel de abstracción más interno, centrándonos en la unidad procesador, se distingue el camino de datos y la unidad de control del mismo (Figura 4.9). En el camino de datos se dispone de elementos lógicos combinacionales y secuenciales para almacenar el estado del procesador y para procesar las instrucciones. La unidad de control determina los elementos que se utilizan del camino de datos y la parte que se actualiza del estado del procesador y de la memoria de datos.



**Figura 4.9** Descripción del camino de datos (CD) y el autómata de control (AC) del procesador.

El camino de datos contiene el registro CP y una instrucción leída de MI es una entrada en el camino de datos. Entre el camino de datos y la unidad de control existe comunicación de información en los dos sentidos. La información denominada de entrada es utilizada por la unidad de control para determinar los elementos que deben utilizarse del camino de datos. La unidad de control transmite esta información al camino de datos

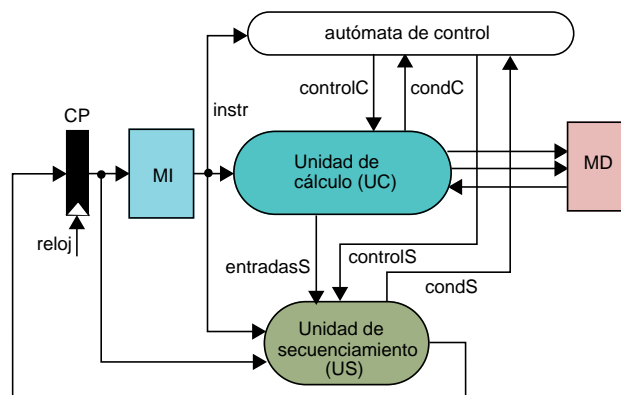
mediante las señales denominadas señales de control. Durante el procesado, el camino de datos también puede transmitir información a la unidad de control para que finalice la acción de control en el procesado de una instrucción (señales denominadas condiciones).

Observemos que el esquema descrito es un esquema genérico de un autómata que dispone de unidad de control y camino de datos. La unidad de control es un autómata de estados finitos cuyas entradas son las señales de entrada y condiciones. Las salidas del autómata son las denominadas señales de control.

## Tercer nivel

En un nivel más interno de abstracción nos centramos en el camino de datos, en el cual distinguimos los siguientes elementos: a) unidad de cálculo y b) unidad de secuenciamiento (Figura 4.10). Por claridad del esquema no se muestra la distribución de las señales de reloj y de inicialización.

La unidad de secuenciamiento (US) es la encargada de determinar la dirección que se utiliza para buscar la siguiente instrucción en la memoria de instrucciones. El autómata de control se lo indica a la US mediante las señales ControlS. La US evalúa condiciones (entradasS) que se transmiten al autómata de control para determinar el secuenciamiento (condS). La salida de esta unidad actualiza el registro contador de programa (CP), el cual pertenece al estado del procesador.



**Figura 4.10** Esquema del camino de datos donde se identifican la unidad de cálculo y la unidad de secuenciamiento. También se muestra la comunicación de información con el autómata de control.

La unidad de cálculo se utiliza para determinar los valores con los que se actualiza: a) el estado almacenado en el banco de registros y en la memoria de datos, o b) se calcula una condición (*condC*), que utiliza el autómata de control para controlar la actualización del banco de registros y la generación de señales de excepción.

Seguidamente, en un nivel más interno de abstracción de la jerarquía de niveles, nos centramos en primer lugar en la unidad de cálculo y la parte de la unidad de control o autómata de control asociada. Posteriormente se analiza el acceso a la memoria de datos, la unidad de secuenciamiento y las organizaciones de la memoria de instrucciones y la memoria de datos.

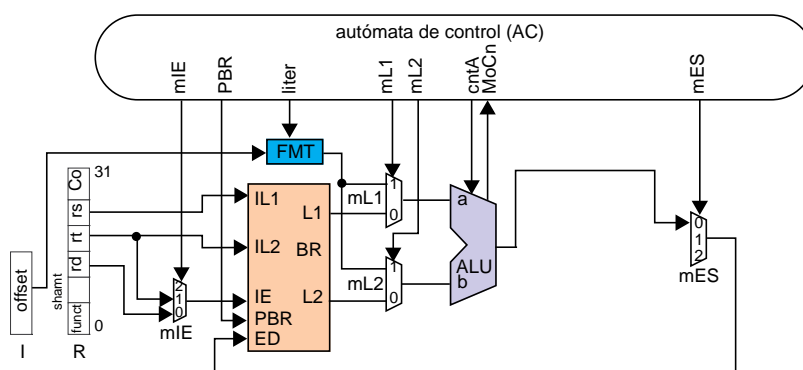
En los esquemas de los siguientes apartados no se muestran las señales de reloj y de inicialización. Tampoco se muestra, en la mayoría de ellos, la entrada de la instrucción en el autómata de control.

## Cuarto nivel: unidad de cálculo

En este apartado se describe la unidad de cálculo y la parte asociada del autómata de control (Figura 4.11). En la unidad de cálculo distinguimos los siguientes elementos: a) banco de registros (BR), b) unidad aritmético lógica (ALU), c) formateador (FMT) y d) elementos de encaminamiento (multiplexores).

El autómata de control (AC) determina los operandos que se utilizan en el cálculo (a, b) y para ello controla los elementos de encaminamiento (mL1, mL2). Además indica a la ALU la operación que debe realizar con los operandos (cntA). Finalmente activa las señales necesarias para actualizar el banco de registros (mIE, PBR). La señal PBR en ocasiones es dependiente de una indicación de la ALU (MoCn).

En el autómata de control distinguiremos los siguientes componentes: a) control de la ALU (cntA) y b) control del resto de elementos.



**Figura 4.11** Unidad de cálculo. Banco de registros (BR), unidad aritmético lógica (ALU) y elementos de encaminamiento. En la parte izquierda se muestran los formatos R e I de posibles instrucciones. Los campos Co y funct de la instrucción son entradas del autómata de control (AC) que no se muestran.

La unidad aritmético lógica, como su nombre indica, es la encargada de efectuar operaciones aritméticas y lógicas. Ejemplos de las primeras son la suma algebraica y comparaciones. Un ejemplo de la segunda es una operación lógica and bit a bit.

En primer lugar se describe el acceso al banco de registros y el encaminamiento de la información. Posteriormente se describen las unidades funcionales incluidas en la ALU.

## Acceso al banco de registros y encaminamiento

El banco de registros, mostrado en la Figura 4.11, tiene dos caminos de lectura (L1 y L2) y un camino de escritura (ED). Las entradas de identificadores de registro asociadas son respectivamente IL1, IL2 e IE. Asociado al camino de escritura existe una señal de permiso de escritura (PBR) y la señal de reloj que no se muestra.

Seguidamente nos centraremos en la obtención de los operandos que serán procesados en la ALU y en el identificador de registro utilizado para actualizar el banco de registros. Para ello utilizaremos un subconjunto de las instrucciones que interpreta el procesador. En la tabla de la Figura 4.12 se especifica, para el subconjunto de instrucciones, el nemotécnico utilizado en ensamblador, una descripción textual de la funcionalidad y una descripción de la operación a nivel de transferencia entre registros. Instrucciones como add y sllv y otras requieren que algún campo de la instrucción tenga un valor concreto. Si este no es el caso hay que generar una condición de error.

Tipo	Nemotécnico	Descripción	Especificación semántica	Condición de error
R	add	suma algebraica	$rd^V = rs^V + rt^V$	shamt $\neq 0$
	sll	desplazamiento lógico a la izquierda	$rd^V = rt^V \ll \text{shamt}$ ; shamt es un natural	
	sllv	desplazamiento lógico a la izquierda variable	$rd^V = rt^V \ll rs^V[4..0]$	shamt $\neq 0$
I	addi	suma con inmediato	$rt^V = rs^V + \text{ExtSig}[\text{offset}]$ ; offset es un entero	
	andi	operación and bit a bit con literal (offset)	$rt^V = rs^V \& \text{ExtCero}[\text{offset}]$ ; offset es un natural	

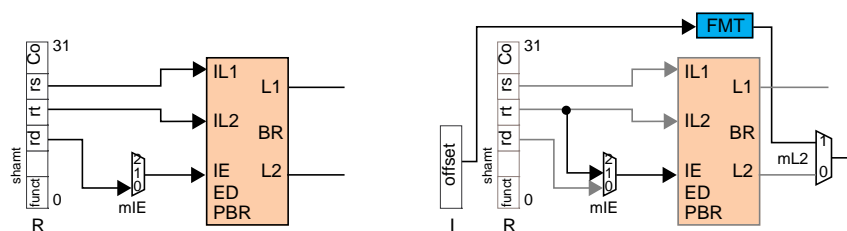
**Figura 4.12** Subconjunto de instrucciones que se ejecutan en la unidad de cálculo. Una extensión de signo hasta completar un operando de 32 bits se identifica como ExtSig. Añadir cero en los bits más significativos hasta completar un operando de 32 bits se identifica como ExtCero. La especificación /4...0 indica los cinco bits menos significativos.

En el acceso al banco de registros un objetivo, si no tenemos en cuenta el consumo energético, es utilizar directamente los campos de la instrucción para efectuar la fase de lectura del banco de registros. Esto es, que no sea necesario esperar alguna señal del decodificador (autómata de control). En este sentido distinguimos dos grupos de instrucciones que realizan operaciones aritméticas o lógicas. Las instrucciones cuyos dos

operandos se encuentran almacenados en registros del banco de registros (tipo R) y las instrucciones donde uno de los operandos es un literal (offset) especificado en la instrucción (tipo I).

En las figuras que se muestran seguidamente se detalla el encaminamiento de las señales. Inicialmente no se muestran todas las señales de entrada de un elemento. Estas señales se añadirán a medida que se describe el subconjunto de instrucciones. Además, en las figuras se utilizan dos tonos de trazado (difuso y nítido) para distinguir el encaminamiento añadido (trazo nítido) del encaminamiento descrito previamente (trazo difuso).

La instrucción “add” utiliza como identificadores de registros fuente rs y rt. La instrucción “addi” utiliza como identificador de registro fuente rs y el otro operando es el campo offset especificado en la instrucción. El campo rs es entrada en el puerto IL1 y el campo rt es entrada en el puerto IL2 (parte izquierda de la Figura 4.13). El campo offset debe interpretarse en complemento a dos. Por ello, antes de utilizarlo como operando hay que formatearlo (FMT). El valor debe convertirse a una representación en 32 bits, que es el número de bits de los operandos. La salida en la fase de lectura del banco de registros es L1 y L2 o el literal formateado. Entonces, después del puerto L2 es necesario un multiplexor (mL2) para poder efectuar una selección en función de la instrucción (parte derecha de la Figura 4.13).



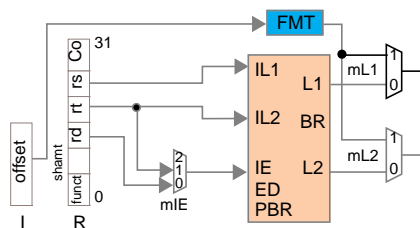
**Figura 4.13** Encaminamiento de los identificadores de registro y salida de la fase de lectura del banco de registros. En la izquierda de cada esquema se muestra el formato de las instrucciones analizadas. En la parte izquierda de la figura se muestra la instrucción “add” y en la parte derecha de la figura se muestran las instrucciones “add” y “addi”.

El identificador de registro destino, el cual se utiliza en la fase de actualización del banco de registros, es el campo rd en la instrucción “add” y el campo rt en la instrucción “addi”. En consecuencia es necesario un multiplexor para poder efectuar una selección en función del tipo de instrucción (Figura 4.13).

La instrucción “sll” utiliza como identificador de un registro fuente el campo rt y el otro operando es el campo shamt especificado en la instrucción (Figura 4.12). Para leer el operando del banco de registros utilizaremos el puerto IL2. Por otro lado, el campo shamt es un subconjunto de los bits utilizados para especificar el campo offset en la instrucción “addi” (Figura 4.13 derecha) y hay que alinearlos al bit menos significativo. En conse-

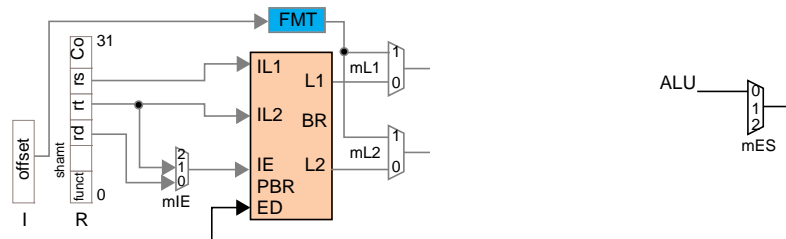
cuencia, el formateador requiere una señal de control para distinguir entre las dos instrucciones (“addi y sll”). Para suministrar este operando a la ALU utilizaremos un multiplexor (mL1) cuyas entradas son el puerto L1 y la salida de FMT (Figura 4.14).

La instrucción “sllv” utiliza los campos rt y rs de la instrucción como identificadores de registro fuente. El campo rs es entrada en el puerto IL1 y el campo rt es entrada en el puerto IL2 (Figura 4.14). Del valor leído en este último puerto sólo se utilizan en la ALU los cinco bits menos significativos. Este filtrado se efectúa en la ALU.



**Figura 4.14** Encaminamiento de los identificadores de registro y salida de la fase de lectura del banco de registros de las instrucciones previas y las instrucciones “sll” y “sllv”. En la izquierda del esquema se muestra el formato de las instrucciones analizadas.

**Actualización del banco de registros.** El resultado de las instrucciones de la Figura 4.12 se utiliza para actualizar el banco de registros (Figura 4.15).



**Figura 4.15** Encaminamiento del resultado producido por la ALU para actualizar el banco de registros.

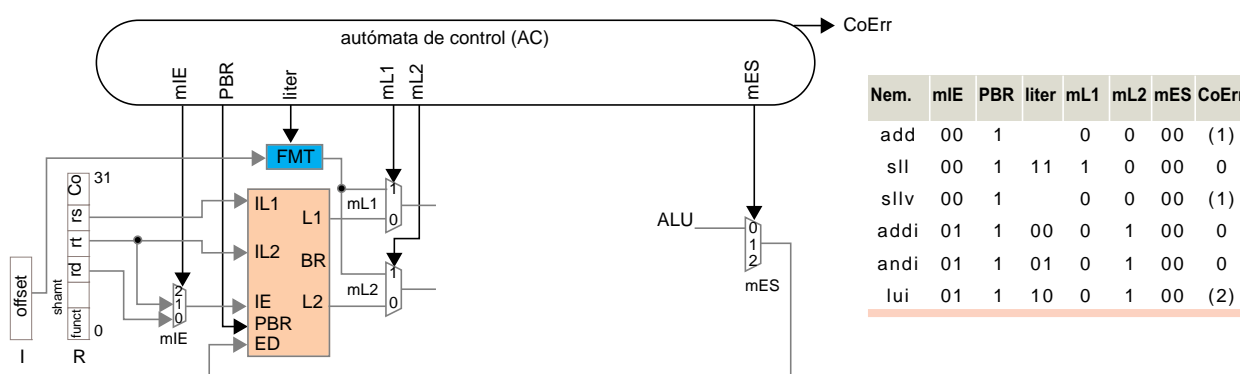
**Unidad de formateo (FMT).** La unidad FMT prepara el literal (offset ó shamt) especificado en una instrucción para ser utilizado como un operando en la ALU. El rango de los valores de entrada en la ALU es mayor que el disponible en el campo literal. En operaciones aritméticas el campo offset debe interpretarse en complemento a dos. Por tanto, se replica el signo del campo literal (offset) para disponer de un operando de 32 bits (“addi”, Figura 4.12). En operaciones lógicas se añaden ceros por la izquierda hasta completar 32 bits (“andi”, Figura 4.12). En operaciones de desplazamiento hay que alinear el campo shamt y se añaden ceros por la izquierda. En una instrucción “lui”, que es de tipo I, y se describe en la Figura 4.16, se añaden ceros por la derecha hasta completar 32 bits. Esto

es, el offset se desplaza 16 posiciones a la izquierda. Por tanto, para distinguir entre las instrucciones “add”, “andi”, “sll” y “lui” son necesarias señales de control en la lógica FMT. Por otro lado, la instrucción “lui” es un ejemplo donde algún campo debe tener un valor concreto. En caso contrario hay que generar una condición de error.

Tipo	Nemotécnico	Descripción	Especificación semántica	Condición de error
I	lui	almacenar el literal en los 16 bits más significativos	$rt^V = [\text{offset}] \& 16'b0;$	si $rs^V \neq 0$

**Figura 4.16** Instrucción lui. El símbolo & indica concatenación y el símbolo 16'b0 indica 16 ceros.

**Señales de encaminamiento y control.** En la tabla de la Figura 4.17 se muestran las señales de control de los multiplexores, las señales de control de la lógica FMT y las señales de error (CoErr) que genera el autómata de control. El número de bits de control de los multiplexores se corresponde con las necesidades cuando se tienen en cuenta todas las instrucciones implementadas. Una entrada vacía en las tablas indica que el valor puede ser uno o cero indistintamente.



**Figura 4.17** Señales de encaminamiento y señales de control y error. Señal de error: (1) la señal CoErr se activa si el campo shamt es distinto de cero, (2) la señal CoErr se activa si el campo rs es distinto de cero. La señal PBR es el permiso de escritura en el banco de registros. Una casilla sin valor indica que no importa el valor de la señal.

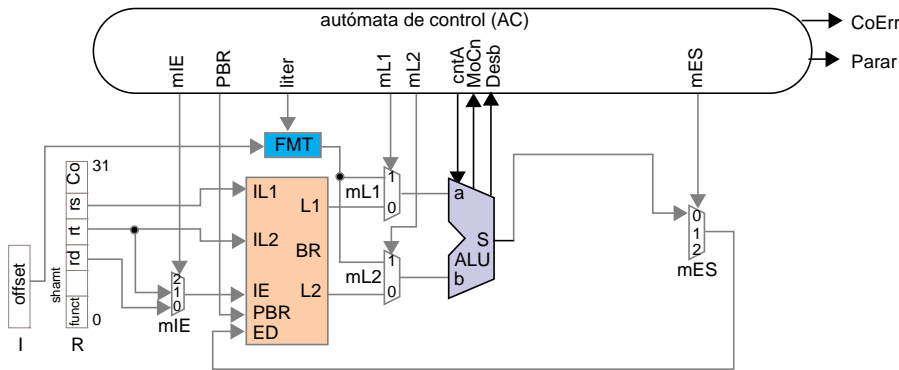
## Unidad aritmético-lógica (ALU)

La ALU implementa, mediante varias unidades lógicas, un conjunto de operaciones aritméticas y lógicas (Figura 4.18).

La operación aritmética básica que realiza la ALU es la suma algebraica con detección de desbordamiento. Como característica relevante indicar que, al ejecutar ciertas instrucciones, el registro destino no se actualiza si se detecta desbordamiento o si no se cumple

una condición. Esta característica requiere que la ALU se comunique con el autómata de control (AC) para anular el permiso de escritura (MoCn y Desb) y si es el caso, generar una excepción (parar).

Otras operaciones que realiza la ALU son: a) resultado de una comparación, b) operación lógica, c) desplazamiento lógico o aritmético a la derecha o a la izquierda (como máximo 32 posiciones) y d) evaluación de una condición para determinar si el contenido de un registro del banco de registros se almacena en otro registro (MoCn).



**Figura 4.18** Unidad funcional ALU y señales de control.

En la tabla de la Figura 4.20 se muestran dos instrucciones de suma (“add” y “addu”). En la instrucción “add” los vectores de bits se interpretan como números enteros representados en complemento a dos y existe detección de desbordamiento. En la instrucción “addu” los vectores de bits se interpretan como números naturales y no existe detección de desbordamiento.

**Instrucciones predicadas (movimiento condicional).** Además de las instrucciones descritas, el lenguaje máquina dispone de instrucciones de movimiento condicional entre registros. El contenido de un registro se copia en otro registro en función de una condición evaluada utilizando el contenido de un tercer registro (Figura 4.19). Esta evaluación se efectúa en la ALU. El resultado de la evaluación debe transmitirse al autómata de control para que anule el permiso de escritura cuando no se cumple la condición (MoCn, Figura 4.20).

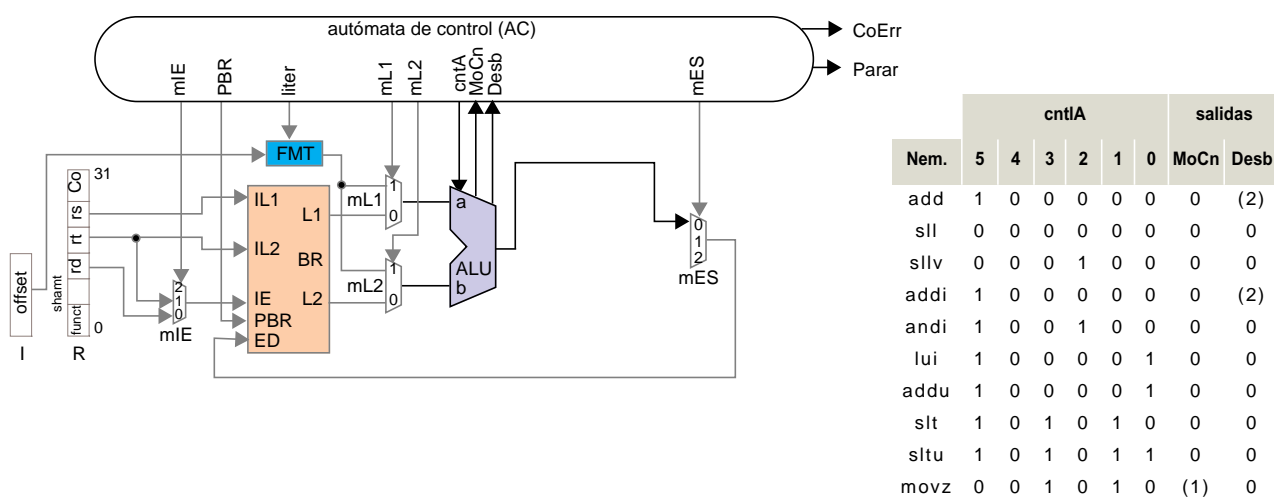
Tipo	Nemotécnico	Descripción	Especificación semántica
R	movn	movimiento condicional si distinto de cero	if (rt <sup>v</sup> ≠ 0) then rd <sup>v</sup> = rs <sup>v</sup> else rd <sup>v</sup> = rd <sup>v</sup>
	movz	movimiento condicional si cero	if (rt <sup>v</sup> = 0) then rd <sup>v</sup> = rs <sup>v</sup> else rd <sup>v</sup> = rd <sup>v</sup>

**Figura 4.19** Instrucciones de movimiento condicional.



**Instrucciones de comparación.** El lenguaje máquina dispone de instrucciones (slt, sltu) para comparar el contenido de dos registros interpretando el contenido en complementos a dos o como números naturales (Figura 4.6). El resultado de la evaluación se almacena en un registro. El contenido de este registro es utilizado por instrucciones de secuenciamiento para determinar el flujo de intrucciones que se interpreta.

**Señales de control de la ALU.** La ALU requiere de señales de control para determinar en cada unidad funcional la operación que debe efectuarse y señales para encaminar las salidas de la unidades funcionales a la salida de la ALU (Figura 4.20). El bit menos significativo de la señal cntA distingue entre operaciones donde se interpretan los vectores de bits en complemento a dos o como números naturales ("add" y "addu", y "slt" y "sltu"). La explicación detallada de los valores de la tabla de la Figura 4.20 se efectúa en Quinto nivel: control de la ALU en la página 169.



**Figura 4.20** Señales de control de la ALU. Señal de condición: (1) el valor de la señal MoCn determina si se actualiza o no el banco de registros. Señal de error: (2) la ALU activa la señal Desb cuando en una operación en complemento a dos detecta desbordamiento. La activación de la señal Desb determina que el autómata de control también active la señal parar.

## Cuarto nivel: acceso a la memoria de datos

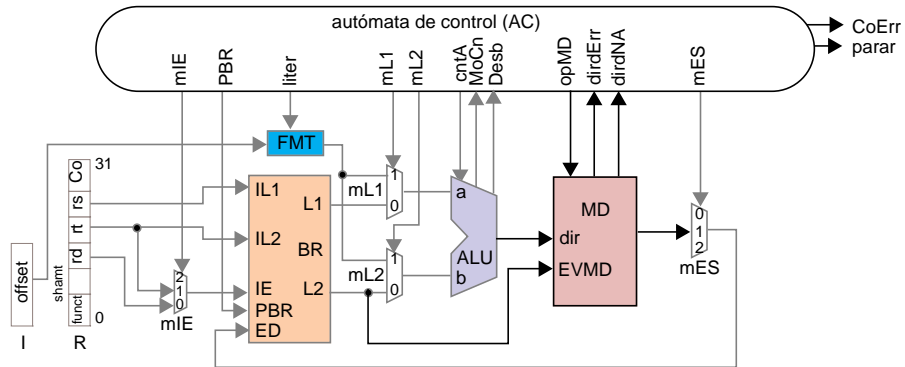
La memoria de datos es un elemento donde se almacenan datos que se pueden leer y actualizar. En este apartado se describe el encaminamiento de un subconjunto de las instrucciones que acceden a la memoria de datos (Figura 4.21).

Tipo	Nemotécnico	Descripción	Especificación semántica
I	lw	lectura de palabra	$rt^V = M[rs^V + ExtSig[offset]]$ ; offset es un entero
	sw	almacenamiento de palabra	$M[rs^V + ExtSig[offset]] = rt^V$ ; offset es un entero

**Figura 4.21** Subconjunto de instrucciones de acceso a la memoria de datos.

$M[. . .]$  indica acceso a memoria. La expresión que sustituye a los puntos calcula la dirección efectiva.  $ExtSig$  indica extensión de signo.

Las instrucciones de acceso a memoria utilizan el banco de registros para obtener los operandos, con los cuales se calcula la dirección efectiva y se obtiene el dato que se almacena en memoria, si es el caso. Posteriormente utilizan el sumador de la ALU para calcular la dirección efectiva, mediante la cual se accede a la memoria de datos (Figura 4.22).



**Figura 4.22** Acceso a la memoria de datos. En la memoria no se muestra la entrada de la señal de reloj.

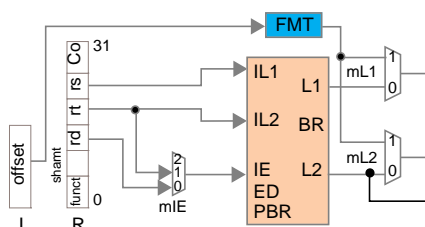
El autómata de control (AC) genera las señales  $opMD$  y procesa las señales  $dirErr$  (dirección fuera de rango) y  $dirDNA$  (dirección no alineada). La activación de estas últimas señales determina que el AC active la señal  $parar$ .

## Acceso al banco de registros y encaminamiento

Seguidamente se describe la fase de lectura del banco de registros para un subconjunto de las instrucciones de acceso a memoria (Figura 4.21).

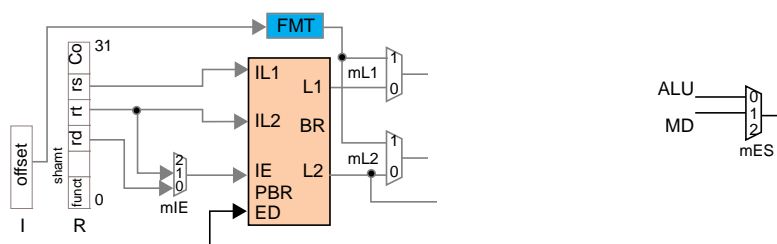
En una instrucción “sw” son necesarios tres operandos. Dos operandos para calcular la dirección efectiva y un tercer operando cuyo valor se almacena en la memoria de datos. Los operandos para calcular la dirección efectiva son el contenido del registro identificado por el campo rs y el campo literal (offset) de la instrucción. El campo rs es entrada en el puerto IL1. El operando literal se suministra a la ALU utilizando el multiplexor mL2 como en una instrucción “addi” (Figura 4.13). El operando que se almacena en memoria es el contenido del registro cuyo identificador se almacena en el campo rt de la instrucción. El campo rt es entrada en el puerto IL2 (Figura 4.23).

En una instrucción “lw” son necesarios dos operandos para calcular la dirección efectiva. Estos operandos son el contenido del registro identificado por el campo rs y el campo literal de la instrucción. El campo rs es entrada en el puerto IL1. El campo rt de la instrucción es el identificador del registro destino y es entrada del puerto IE.



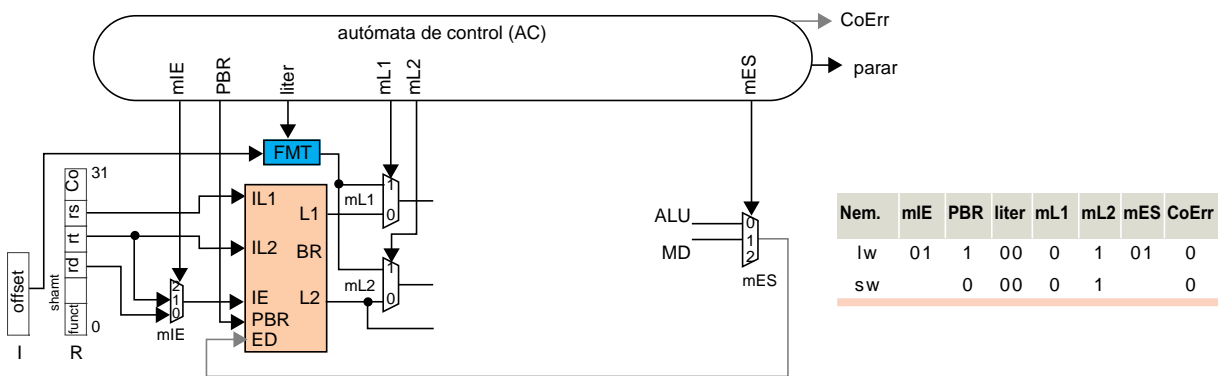
**Figura 4.23** Encaminamiento de los identificadores de registro y salida de la fase de lectura del banco de registros de las instrucciones previas y las instrucciones “sw” y “lw”. En la izquierda del esquema se muestra el formato de las instrucciones analizadas.

**Actualización del banco de registros.** El resultado de las instrucciones mostradas en la Figura 4.12 y en la Figura 4.21 puede producirse en la ALU o al leer de la memoria de datos. En consecuencia, es necesario un multiplexor que permita seleccionar entre estas dos fuentes de datos para actualizar el banco de registros (multiplexor mES en la Figura 4.24).



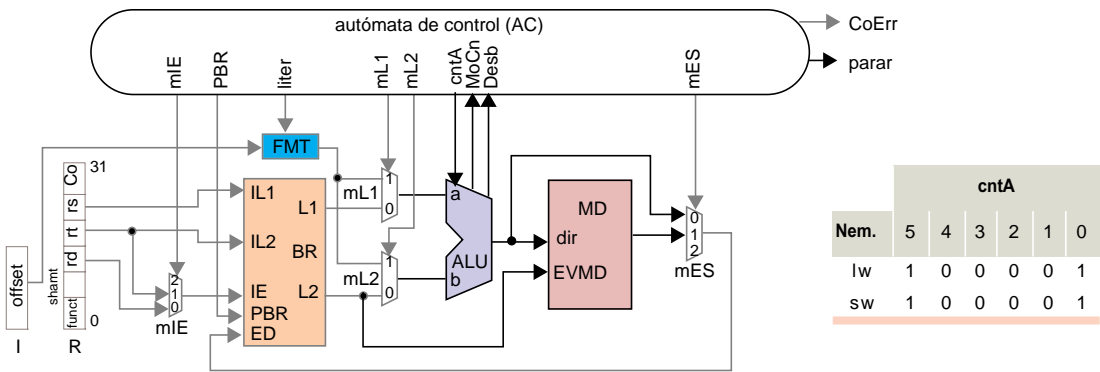
**Figura 4.24** Encaminamiento del resultado producido por la ALU y MD para actualizar el banco de registros.

**Señales de encaminamiento y control.** En la tabla de la Figura 4.25 se muestran las señales de control de los multiplexores, las señales de control de la lógica FMT y las señales de error (CoErr) que genera el autómata de control al decodificar las instrucciones de acceso a memoria. El número de bits de control de los multiplexores se corresponde con las necesidades cuando se tienen en cuenta todas las instrucciones implementadas.



**Figura 4.25** Señales de encaminamiento y señales de control para las instrucciones de acceso a memoria. La señal PBR es el permiso de escritura en el banco de registros.

**Señales de control de la ALU.** Una instrucción de acceso a memoria requiere que la ALU efectúe la suma de las entradas sin detección de desbordamiento. Por ello, las señales de control de la ALU son las mismas que en una instrucción addu. En la Figura 4.26 se muestran las señales de control.



**Figura 4.26** Control de la ALU en instrucciones de acceso a memoria.

## Tipos de instrucciones de acceso a la memoria de datos.

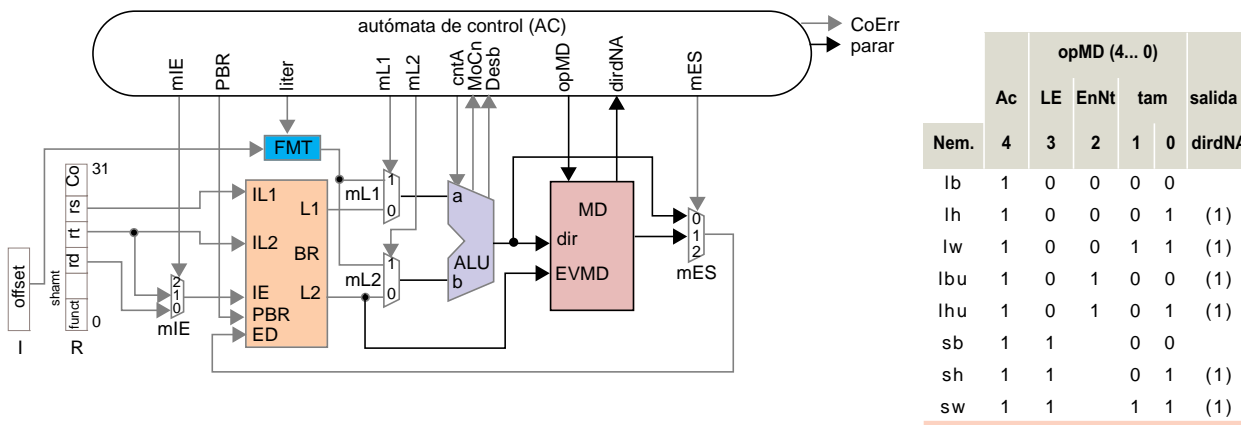
En la tabla de la Figura 4.27 se muestran las instrucciones de acceso a memoria. Se distinguen varios tipos de instrucciones en función del tamaño del dato que se lee o escribe. También se distinguen varios tipos en función de cómo se interpreta el valor leído de la memoria de datos (entero o natural).

Tipo	Nemotécnico	Descripción	Especificación semántica
I	lb	lectura de un byte	$rt^V = \text{ExtSig} ( (M [rs^V + \text{ExtSig} [\text{offset}]] ) _{7:0} )$
	lh	lectura de dos bytes (media palabra)	$rt^V = \text{ExtSig} ( (M [rs^V + \text{ExtSig} [\text{offset}]] ) _{15:0} )$
	lw	lectura de una palabra	$rt^V = M [rs^V + \text{ExtSig} [\text{offset}]]$
I	lbu	lectura de un byte como natural	$rt^V = \text{ExtCero} ( (M [rs^V + \text{ExtSig} [\text{offset}]] ) _{7:0} )$
	lhu	lectura de media palabra como natural	$rt^V = \text{ExtCero} ( (M [rs^V + \text{ExtSig} [\text{offset}]] ) _{15:0} )$
I	sb	almacenamiento de un byte	$M [rs^V + \text{ExtSig} [\text{offset}]] = rt^V _{7:0}$
	sh	almacenamiento de media palabra	$M [rs^V + \text{ExtSig} [\text{offset}]] = rt^V _{15:0}$
	sw	almacenamiento de una palabra	$M [rs^V + \text{ExtSig} [\text{offset}]] = rt^V$

**Figura 4.27** Instrucciones de acceso a memoria. Los subíndices indican los bits de un vector de bits de 32 bits que se leen o escriben. Una extensión de signo hasta completar un operando de 32 bits se identifica como *ExtSig*. Añadir cero en los bits más significativos hasta completar un operando de 32 bits se identifica como *ExtCero*. La especificación  $|_{b...a}$  indica una secuencia contigua de bits identificada por los ordinales *b* y *a*.

La dirección efectiva debe estar alineada a 4 bytes cuando se leen 32 bits y a 2 bytes cuando se leen 16 bits. Si este no es el caso hay que generar una condición de error (dirdNA).

**Señales de control de la memoria de datos.** En la Figura 4.28 se muestran las señales necesarias para controlar el acceso a la memoria de datos. Las señales deben especificar el tipo de acceso (lectura o escritura, LE), la granularidad o tamaño del acceso (tam) y la forma de extender, a 32 bits, el dato leído cuando sea necesario (EnNt). El bit más significativo de la señal opMD indica que es un acceso a memoria.



**Figura 4.28** Señales de control de la memoria de datos. Señal de error: (1) la MD activa la señal dirdNA cuando, en un acceso a memoria, se detecta que el acceso no está alineado. La activación de esta señal determina que el autómata de control también active la señal parar.

## Cuarto nivel: unidad de secuenciamiento

La unidad de secuenciamiento es la encargada de determinar la dirección de la siguiente instrucción que debe interpretarse.

## Descripción funcional

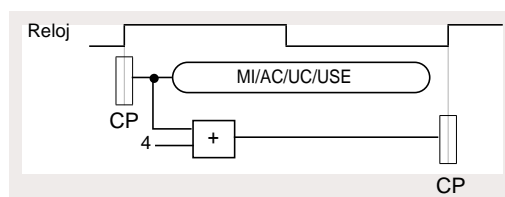
El procesador dispone de un secuenciamiento por defecto o implícito y de un secuenciamiento explícito que se efectúa al interpretar una instrucción de secuenciamiento.

El secuenciamiento implícito calcula direcciones de instrucciones almacenadas en posiciones consecutivas de memoria. Como el tamaño de una instrucción es fijo e igual a cuatro bytes, es suficiente sumar cuatro a la dirección de una instrucción para calcular la dirección de la siguiente instrucción.

El valor de la dirección de una instrucción se almacena en un registro denominado CP. En la Figura 4.29 se muestra el registro CP, la lógica para la actualización explícita del mismo (USE) y la circuitería para el secuenciamiento implícito (sumador). Además se muestra la relación del retardo de la lógica con la señal de reloj. En la parte superior de la figura se muestra la señal de reloj.

Debajo de la señal de reloj se observan registros tanto en el primer flanco ascendente como en el segundo flanco ascendente. Notemos también que el nombre de los registros es el mismo. Un registro que se muestra en el primer flanco se analiza desde el punto de vista del valor de la señal que almacena y que se observa en su salida. Un registro que se muestra en el segundo flanco se analiza desde el punto de vista del valor de la señal en la entrada del registro. Esta entrada se almacenará en el registro en el flanco ascendente.

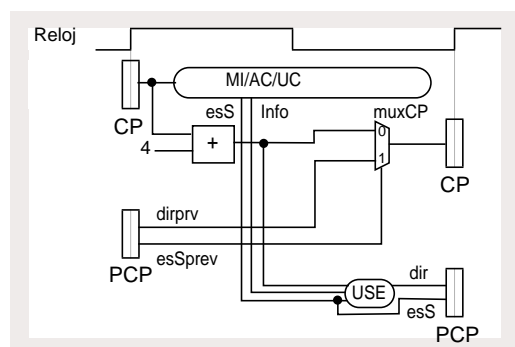
Debajo de la señal de reloj y en la parte izquierda de la Figura 4.29 se muestra el registro CP. La salida de este registro se utiliza para acceder a la memoria de instrucciones y posteriormente interpretar la instrucción en la UC y la USE. En este apartado nos centramos en la USE que gestiona el secuenciamiento explícito. Debajo de la lógica etiquetada como MI/AC/UC/USE se muestra la lógica utilizada para efectuar el secuenciamiento implícito. El resultado de sumar cuatro a la dirección de la instrucción es entrada del registro CP. El valor se transfiere a la salida del registro CP en el siguiente flanco ascendente.



**Figura 4.29** Secuenciamiento implícito. AC: autómata de control. UC: unidad de cálculo. MI: memoria de instrucciones. USE: unidad de secuenciamiento explícito. La unidad US incluye la unidad USE y el sumador (secuenciamiento implícito).

La semántica de todas las instrucciones de secuenciamiento indica que, antes de establecer el secuenciamiento, debe interpretarse la siguiente instrucción en secuencia. En consecuencia, la dirección que determina una instrucción de secuenciamiento hay que almacenarla durante un ciclo. Para ello utilizaremos un registro que denominaremos próximo contador de programa (PCP). En la parte izquierda de la Figura 4.30 se muestra un esquema del camino de datos teniendo en cuenta el comportamiento descrito.

Al interpretar una instrucción de secuenciamiento se utiliza información de la USE y la salida del sumador para determinar la dirección. El autómata de control (AC) indica que es una instrucción de secuenciamiento (esS) y suministra información para el cálculo de la dirección al módulo USE (Info). La dirección calculada y la indicación de que debe utilizarse esta dirección son entradas del registro PCP antes de que finalice el ciclo de reloj.



**Figura 4.30** *Secuenciamiento explícito.*

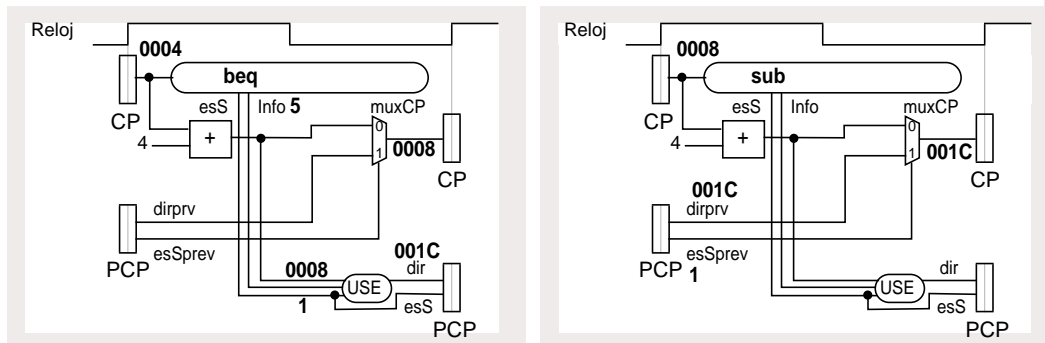
La dirección de la próxima instrucción (entrada del registro CP, salida de muxCP) es la dirección de la instrucción actual más cuatro o la dirección que ha determinado la instrucción de secuenciamiento que se ha interpretado en el ciclo previo (salida del registro PCP).

En la parte izquierda de la Figura 4.31 se muestra una secuencia de instrucciones que incluye una instrucción de secuenciamiento. Después de esta instrucción (dirección 0x0004<sup>1</sup>) siempre se interpreta la siguiente instrucción (dirección 0x0008). En la parte central de la Figura 4.31 se observa que se está interpretando la instrucción de secuenciamiento. La entrada del registro CP es 0x0008 y la entrada del registro PCP es 0x001C. La señal esS está activada y la señal Info transfiere los siguientes datos: a) literal (0x5) y contenido de los registros r3 y r0 (no mostrados). Notemos también, que el valor de la salida del sumador es 0x0008. Como suponemos que se cumple la condición, la salida del módulo USE tiene el valor 0x001C (0x0008 + 0x5<<2).

1. 0x indica representación en hexadecimal.



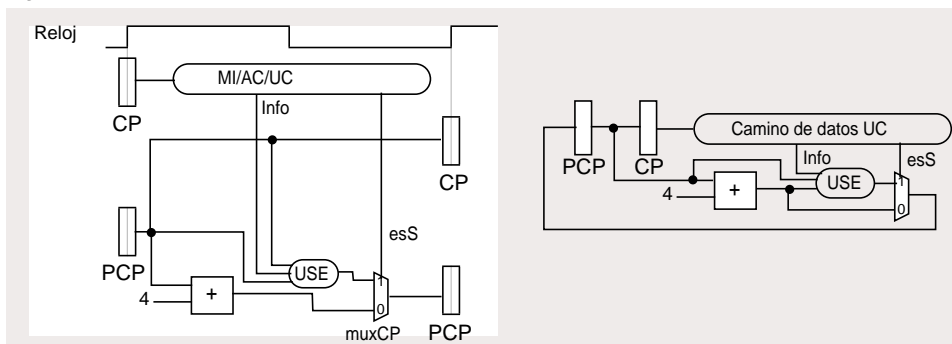
secuencia de instrucciones  
 0000 add r3, r2, r1  
 0004 beq r3, r0, 0x5  
 0008 sub r7, r6, r2  
 ....  
 ....  
 001C xor r2, r5, r9



**Figura 4.31** Ejemplo de secuenciamiento explícito. Los valores numéricos se interpretan en hexadecimal.

En la parte derecha de la Figura 4.31 se muestra el siguiente ciclo. La salida del registro CP tiene el valor 0x0008. En consecuencia, se está interpretando la instrucción que sigue en secuencia a la instrucción de secuenciamiento (sub). Los valores en la salida del registro PCP son: a) dirprv = 0x001C y b) esSprev = 1. Entonces, el valor de la salida del multiplexor muxCP es 0x001C. Por tanto, en el siguiente ciclo se interpreta la instrucción almacenada en la dirección 0x001C.

En la parte izquierda de la Figura 4.32 se muestra una reorganización de la lógica que realiza las mismas funciones que la lógica mostrada en la Figura 4.30<sup>1</sup>. En la parte derecha de la Figura 4.32 se muestra el esquema, de la parte izquierda de la misma figura, de la forma en que usualmente se muestra en el camino de datos del procesador.

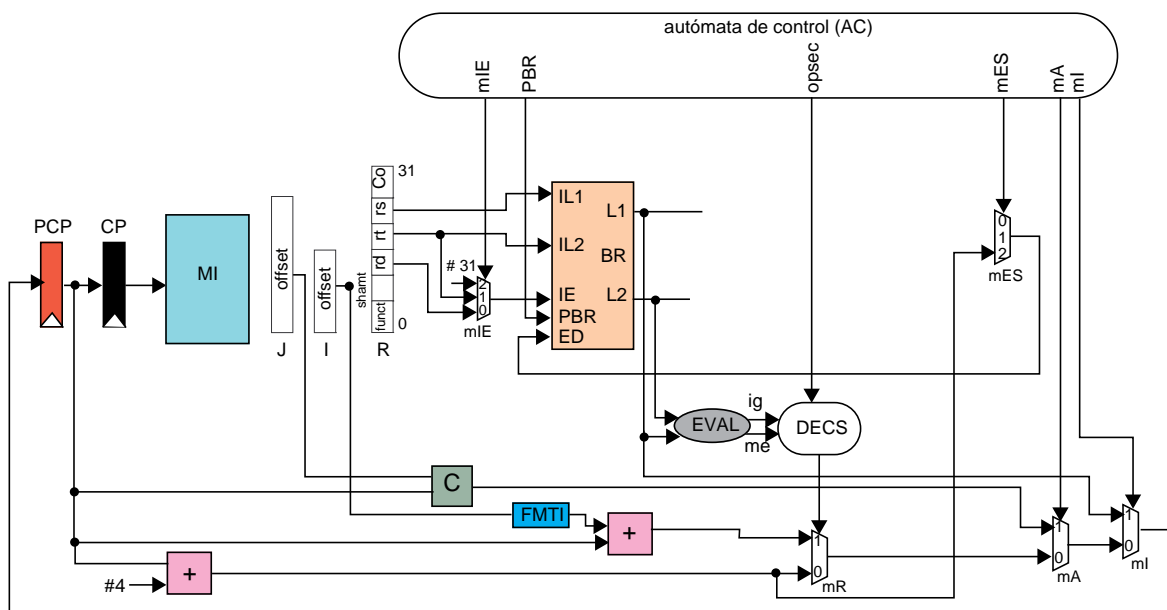


**Figura 4.32** Esquema simplificado del secuenciamiento.

1. Ambos circuitos son equivalentes si el registro PCP se inicializa convenientemente (CP+4).

En la Figura 4.33 se muestra de forma más detallada la unidad de secuenciamiento. En la unidad de secuenciamiento podemos distinguir: a) el registro CP, la lógica para su actualización y c) el autómata de control.

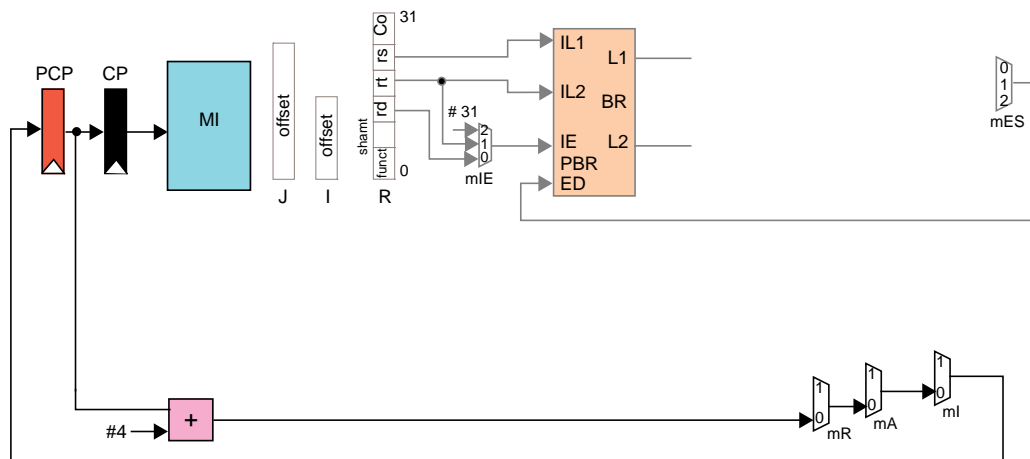
En la lógica de actualización se distinguen: a) sumadores, b) un circuito para concatenar vectores de bits (C), c) un circuito para formatear el campo offset (FMTI) y d) un circuito para evaluar condiciones (EVAL). El circuito DECS es parte del autómata de control (AC) y se utiliza para determinar el sentido del secuenciamiento en instrucciones de secuenciamiento condicional. Las señales *ig* y *me* indican el resultado de la condición evaluada en el módulo EVAL de la unidad de secuenciamiento y transmitida al autómata de control (DECS).



**Figura 4.33** Unidad de secuenciamiento.

## Secuenciamiento implícito

La circuitería básica de secuenciamiento incluye el registro CP, un sumador y el registro próximo contador de programa (PCP, Figura 4.34). Cuando se inicializa el procesador la diferencia entre los registros PCP y CP es el valor cuatro que se corresponde con el tamaño en bytes de una instrucción.



**Figura 4.34** Encaminamiento para el secuenciamiento implícito.

## Secuenciamiento explícito

Seguidamente analizaremos un subconjunto de instrucciones de secuenciamiento con el objetivo de diseñar la lógica que se utiliza para actualizar el registro CP (tabla de la Figura 4.35).

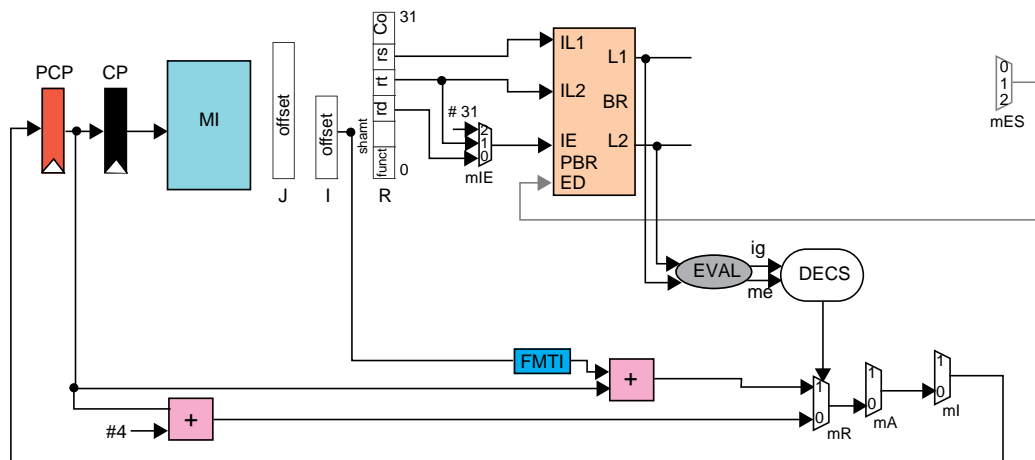
Tipo	Nemotécnico	Descripción	Especificación semántica	condiciones de error
I	beq	secuenciamiento condicional relativo	$\text{if } (rs^v = rt^v) \text{ then } CP^v = CP^v + 4 + \text{ExtSig}(\text{offset}); \text{ else } CP^v = CP^v + 8$	
I	bgtz	secuenciamiento condicional relativo	$\text{if } (rs^v > 0) \text{ then } CP^v = CP^v + 4 + \text{ExtSig}(\text{offset}); \text{ else } CP^v = CP^v + 8$	si $rt \neq 0$
I*	bltzal	secuenciamiento condicional relativo y almacenamiento de la dirección de retorno	$R31 = CP + 8$ $\text{if } (rs^v < 0) \text{ then } CP^v = CP^v + 4 + \text{ExtSig}(\text{offset}); \text{ else } CP^v = CP^v + 8$	
J	j	secuenciamiento incondicional	$CP^v = (CP^v + 4)_{ 31...28} \& \text{offset} << 2 \& '00'$	
I	jr	secuenciamiento indexado	$CP^v = rs^v$	

**Figura 4.35** Subconjunto de instrucciones de secuenciamiento. El símbolo  $I^*$  indica una modificación del formato I. Una extensión de signo hasta completar un operando de 32 bits se identifica como ExtSig. La especificación  $|_{b...a}$  indica una secuencia contigua de bits identificada por los ordinales b y a. El símbolo & indica concatenación. El símbolo “<< 2” indica desplazar dos posiciones a la izquierda y el símbolo ‘00’ indica dos ceros en binario.

La instrucción “beq” es una instrucción de secuenciamiento condicional. La condición que se utiliza para determinar el secuenciamiento es el resultado de comparar si el contenido de dos registros es igual (EVAL). Los identificadores de los registros se obtienen de los campos rs y rt de la instrucción. El campo rs es una entrada del puerto IL1

y el campo *rt* es entrada del puerto IL2 (Figura 4.36). Si no se cumple la condición se sigue en secuencia. Si se cumple la condición, la dirección destino es la suma de la dirección de la instrucción de secuenciamento más cuatro bytes (siguiente instrucción), más el offset especificado en la instrucción desplazado dos posiciones a la izquierda. Previamente, el valor numérico que representa el campo *offset*, interpretándolo en complemento a dos, se representa con 32 bits (FMTI).

En la instrucción “bgtz” se evalúa si el contenido del registro, cuyo identificador es *rs*, es mayor que cero.



**Figura 4.36** Instrucción de secuenciamento condicional *beq*.

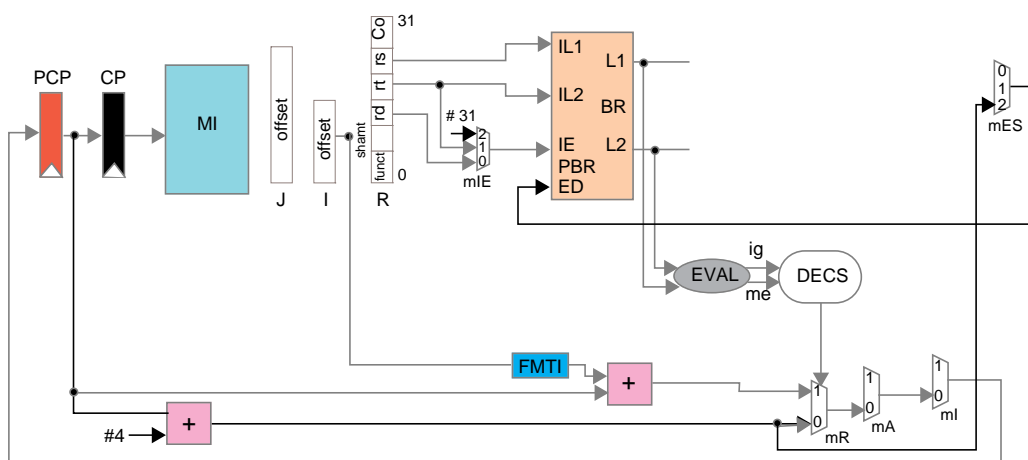
Un secuenciamento incondicional relativo a la dirección de la instrucción de secuenciamento se implementa mediante una condición de igualdad y el registro *R0* en los dos operandos, para que siempre se cumpla la condición.

**Circuito de evaluación de la condición (EVAL).** Este circuito realiza la comparación del contenido de dos registros y genera la señal *ig* (igualdad). También analiza el dato leído del banco de registro por el puerto *L1* y establece el valor de la señal *me* (menor que cero). En la Figura 4.37 se muestra, para un subconjunto de instrucciones de secuenciamento condicional, el valor de las señales *ig* y *me* para que se cumpla la condición evaluada.

Nemotécnico	Descripción	Condición evaluada	ig	me	Campos predeterminados en la instrucción
beq	modificación del secuenciamiento si el contenido es el mismo	$rs^v = rt^v$	1		
bgtz	modificación del secuenciamiento si el contenido es mayor que cero	$rs^v > 0$	0	0	$rt = 0$
bltzal	modificación del secuenciamiento si el contenido es menor que cero	$rs^v < 0$		1	

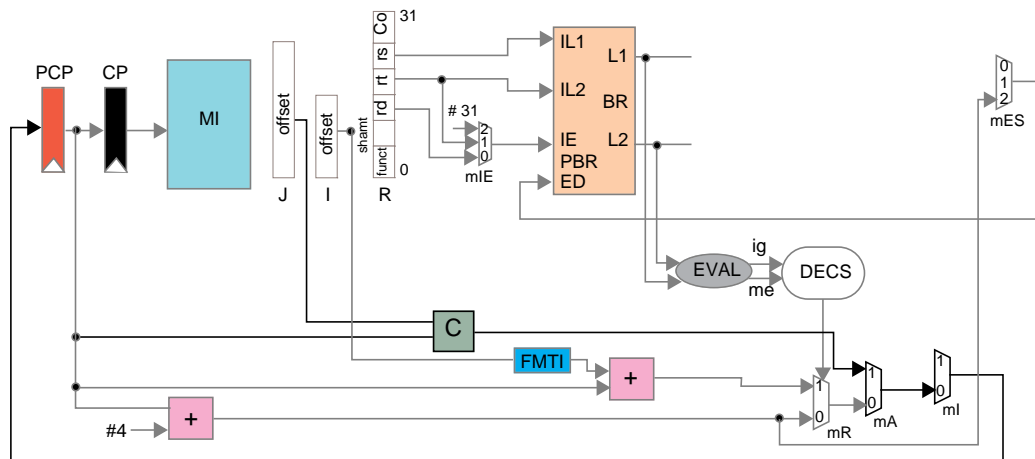
**Figura 4.37** Condiciones evaluadas en instrucciones de secuenciamiento condicional y valores de las señales de salida del circuito EVAL. Una entrada sin valor indica que puede ser indistintamente cualquier valor. En la primera instrucción se compara el contenido de dos registros. En las siguientes instrucciones se analiza el contenido de un registro.

La instrucción “bltzal” es similar a la instrucción “bgtz”. La diferencia reside en la condición que se evalúa para determinar el secuenciamiento. En la instrucción “bltzal” se evalúa si el contenido de un registro es menor que cero. El identificador de registro es el campo rs de la instrucción que es entrada del puerto IL1. Adicionalmente esta instrucción almacena en un registro la dirección de la instrucción que está a una distancia de 8 bytes de la instrucción de secuenciamiento (Figura 4.38). El registro donde se almacena la dirección está implícito en la codificación de la instrucción, siendo el registro cuyo identificador es 31 en decimal (R31).



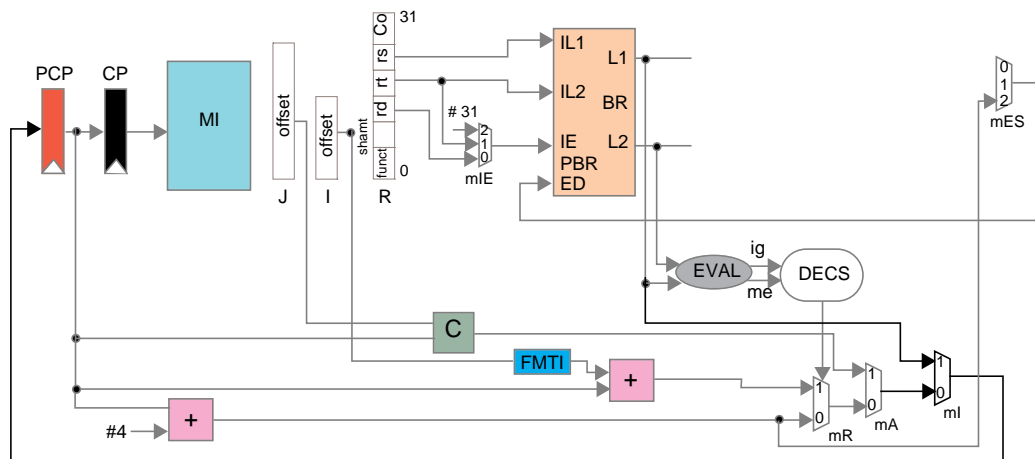
**Figura 4.38** Instrucción de secuenciamiento condicional bltzal.

La instrucción “j” determina un secuenciamiento incondicional. La dirección destino se obtiene concatenando (módulo C) los cuatro bits más significativos de la dirección de la instrucción, que sigue a la instrucción de secuenciamiento en el espacio lógico, con los bits del campo offset de la instrucción. Los dos bits menos significativos son cero para que la dirección destino esté alineada a cuatro bytes (Figura 4.39).



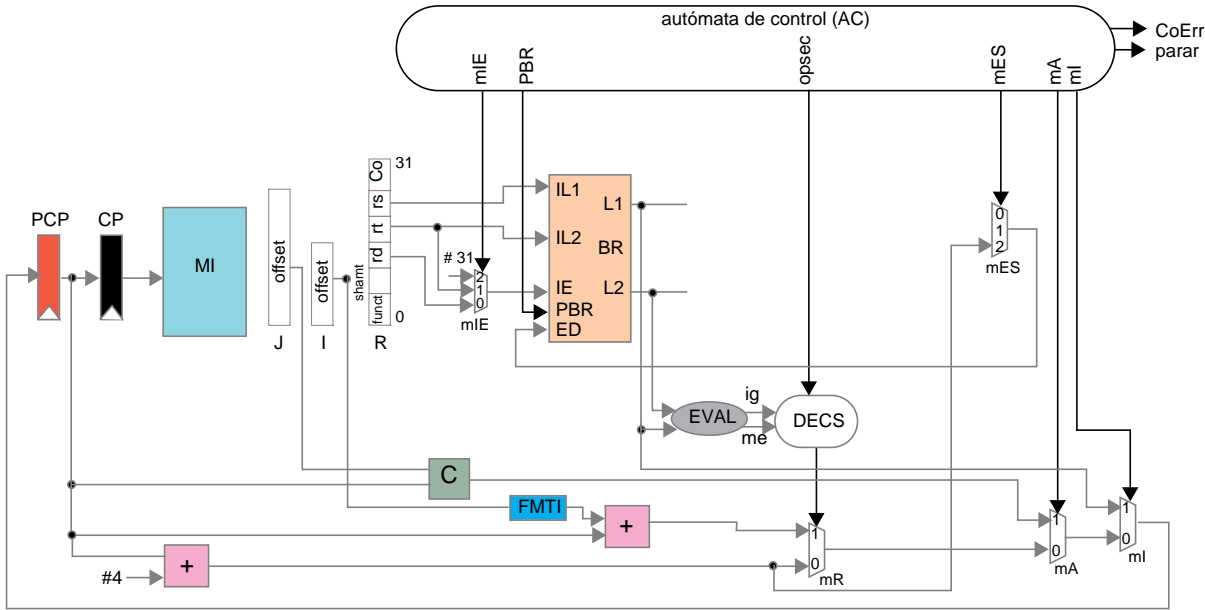
**Figura 4.39** Instrucción de secuenciamiento incondicional j.

La instrucción “jr” utiliza el campo rs como identificador de registro fuente, el cual es entrada del puerto IL1. El contenido del registro es la dirección efectiva para acceder a la memoria de instrucciones (Figura 4.40).



**Figura 4.40** Instrucción de secuenciamiento incondicional jr.

**Señales de encaminamiento y actualización del banco de registros.** En la Figura 4.41 se muestran las señales de control de los multiplexores que genera el autómata de control. El circuito DECS está incluido en el autómata de control y las señales de entrada se muestran en la tabla de la Figura 4.42. El número de bits de control de los multiplexores se corresponde con las necesidades cuando se tienen en cuenta todas las instrucciones implementadas.



**Figura 4.41** Señales de encaminamiento de la unidad de secuenciamiento.

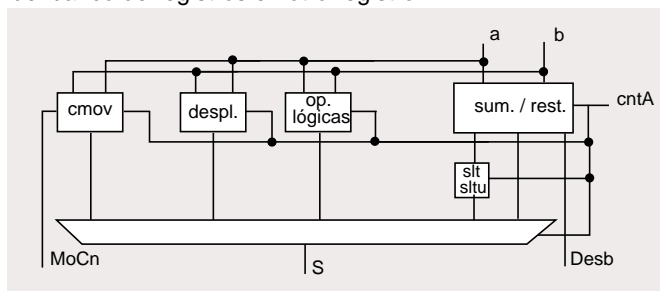
En la tabla de la Figura 4.42 se muestra el valor de la señales para controlar el encaminamiento en la unidad de secuenciamiento. Para las instrucciones de secuenciamiento condicional se muestran dos filas en función del resultado al evaluar la condición.

Nemotécnico	CE	mIE	PBR	mR	mA	ml	mES
beq	0		0	0	0	0	
	1		0	1	0	0	
bgtz	0		0	0	0	0	
	1		0	1	0	0	
bltzal	0	10	1	0	0	0	10
	1	10	1	1	0	0	10
j			0		1	0	
jr			0			1	

**Figura 4.42** En la columna CE se indica si la condición evaluada es cierta (1) o falsa (0).

## Quinto nivel: unidad aritmético lógico (ALU)

En la Figura 4.43 se muestra un esquema de la ALU. De derecha a izquierda se identifican las siguientes unidades funcionales: a) sumador algebraico, b) resultado de una comparación, c) operación lógica, d) desplazamiento lógico o aritmético a la derecha o a la izquierda (como máximo 32 posiciones) y e) almacenamiento condicional del contenido de un registro del banco de registros en otro registro.



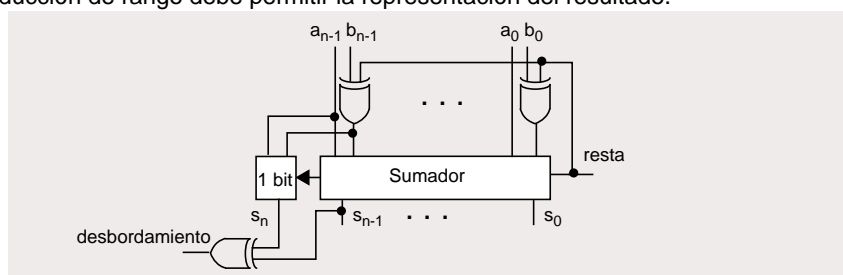
**Figura 4.43** Unidades funcionales disponibles en la ALU.

En la tabla de la Figura 4.20 se ha mostrado el valor de las señales de control de la ALU, las cuales se utilizan para seleccionar la entrada del multiplexor de salida de la unidad mostrado en la Figura 4.43.

## Sexto nivel: unidades funcionales de la ALU

En este apartado se describen parcialmente dos unidades funcionales incluidas en la ALU (Figura 4.43): el módulo sumador y el módulo desplazador (lógico y aritmético).

**Suma algebraica.** En la Figura 4.44 se muestra un circuito que permite realizar sumas y restas y que detecta la condición de desbordamiento cuando se interpretan los operandos en complemento a dos. Se utiliza un sumador binario de  $n$  bits y un sumador adicional de un bit. El desbordamiento se detecta comparando la representación del valor numérico en la salida utilizando  $n$  y  $n+1$  bit. Para que no se produzca una situación de desbordamiento, la reducción de rango debe permitir la representación del resultado.



**Figura 4.44** Sumador y restador en complemento a dos con detección de desbordamiento.



Otras operaciones básicas que se realizan en la ALU son las denominadas desplazamientos. Un desplazamiento de un vector de bits puede ser fijo o variable. Efectuar un desplazamiento fijo en hardware es trivial ya que sólo es necesario encaminar directamente los cables que transportan los bits. Por tanto, nos centraremos en el caso de un desplazamiento variable.

Distinguiremos tres tipos de desplazamiento y cada uno de ellos puede ser a la izquierda o a la derecha.

**Rotación.** Rotación de los bits en círculo. Las posiciones que quedan vacías se rellenan con los bits que se descartan por el otro lado.

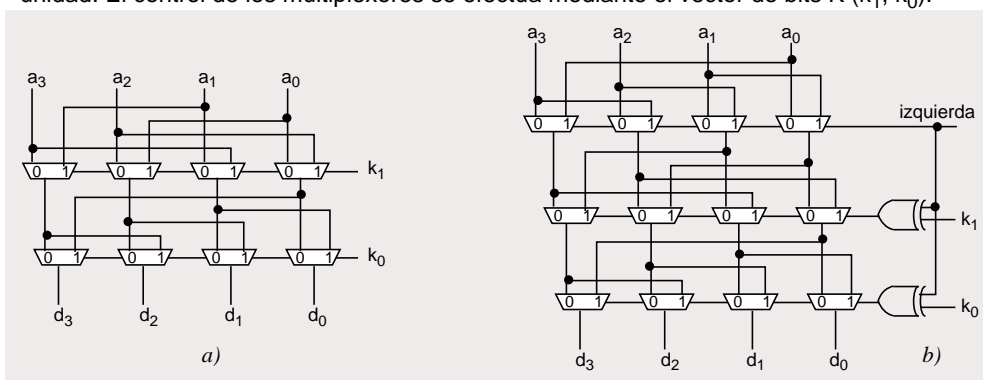
**Desplazamiento lógico.** Desplazamiento de los bits a la derecha o a la izquierda y las posiciones que quedan vacías se rellenan de ceros.

**Desplazamiento aritmético.** Igual que el desplazamiento lógico, pero cuando el desplazamiento es a la derecha, las posiciones más significativas del vector de bits que quedan vacías se rellenan replicando el bit de signo, ya que el vector de bits se interpreta en complemento a dos.

Un desplazamiento aritmético es una operación de multiplicación o división por una potencia de dos en función de si el desplazamiento es a la derecha o a la izquierda.

Dado un vector de  $n$  bits, conceptualmente una rotación requiere un conjunto de  $n$  multiplexores de  $n$  entradas, para seleccionar cada una de las salidas a partir de cada una de las entradas. Los desplazamientos lógicos y aritméticos son similares a rotaciones pero rellenan las posiciones descartadas que se crean con ceros o con unos.

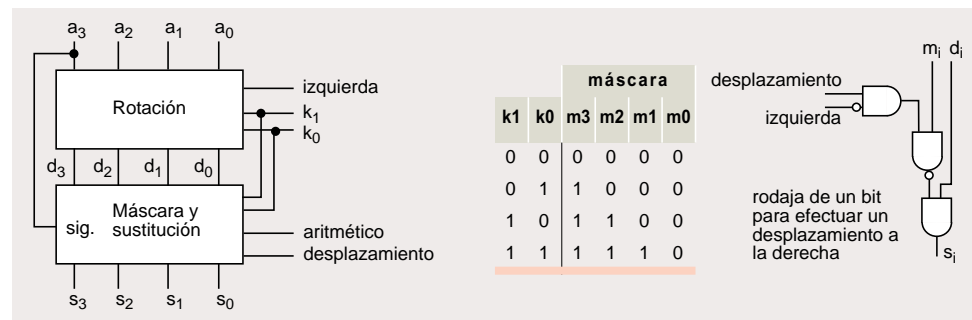
En la parte izquierda de la Figura 4.45, para un vector de cuatro bits, se muestra un circuito que efectúa rotaciones a la derecha y ha sido implementado mediante multiplexores de dos entradas. Con la primera fila de multiplexores se puede efectuar una rotación de dos unidades, mientras que con la segunda fila la posible rotación es de una unidad. El control de los multiplexores se efectúa mediante el vector de bits  $K$  ( $k_1, k_0$ ).



**Figura 4.45** a) Circuito que efectúa rotaciones a la derecha (entre 0 y 3 posiciones), b) circuito que efectúa rotaciones a la derecha e izquierda.

En la parte derecha de la Figura 4.45 se muestra una modificación del circuito que permite realizar también rotaciones a la izquierda. Se añade una fila de multiplexores que permite realizar una rotación a la derecha de una posición y se modifica el control de las siguientes dos filas respecto de una rotación a la derecha. Esto es, cuando la rotación es a la izquierda se utiliza la primera fila de multiplexores y el vector K, de control de los multiplexores, se complementa.

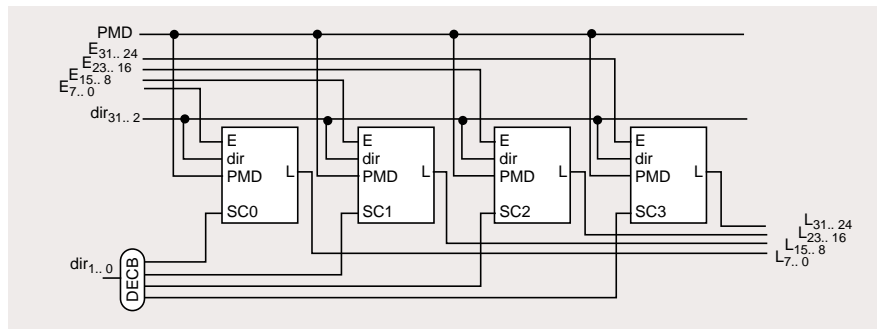
El circuito de la Figura 4.45 se puede utilizar para construir un módulo que permita realizar desplazamientos lógicos y aritméticos. Para ello se añade en su salida un circuito que se denomina “Máscara y sustitución”. Su función es introducir ceros o unos en las posiciones que se rellenarían con los bits descartados al efectuar una rotación. Para ello se construye una máscara a partir del vector K. La tabla de verdad de la máscara se muestra en una tabla en el centro de la Figura 4.46. Posteriormente se utiliza la máscara y las restantes señales de entrada del circuito denominado “Máscara y sustitución” para determinar las posiciones donde se establecen ceros o unos. En la parte derecha de la Figura 4.46 se muestra parte del circuito. En concreto se muestra, para un bit, el circuito que permite obtener un desplazamiento a la derecha a partir de una rotación a la derecha. El bit que ha sido rotado se sustituye por un valor obtenido a partir de la máscara.



**Figura 4.46** Módulo que realiza rotaciones y desplazamiento lógicos y aritméticos.

## Segundo nivel: organización de la memoria de datos

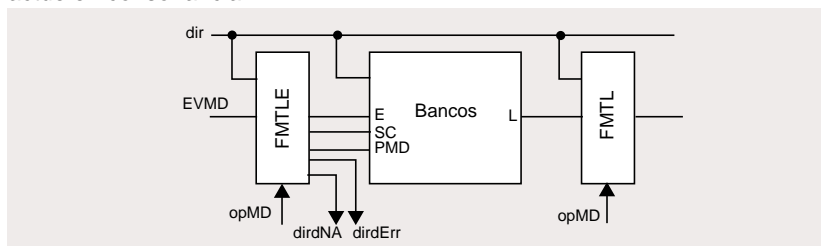
La memoria de datos está organizada en cuatro bancos (Figura 4.47). Cada posición de un banco almacena un byte y las posiciones de memoria están entrelazadas entre los bancos utilizando los dos bits menos significativos de una dirección. Las entradas SCx en los bancos se utilizan para seleccionar los bancos accedidos en una operación de acceso a memoria.



**Figura 4.47** Organización de la memoria de datos. PMD es la señal de permiso de escritura. El símbolo  $/_{a..b}$  indica los bits que se consideran en un vector de 32 bits.

En la especificación de lenguaje máquina, en un acceso a memoria se pueden leer o escribir tres tamaños de datos: a) byte, b) dos bytes, c) cuatro bytes o palabra. Para soportar los distintos tamaños de acceso se utilizan dos circuitos formateadores (Figura 4.48). Uno de ellos se ubica antes de los bancos de memoria (FMTLE) y es utilizado al interpretar las instrucciones de almacenamiento en la memoria de datos. El otro circuito formateador (FMTL) se ubica después de los bancos de memoria y es utilizado al interpretar las instrucciones de lectura de la memoria de datos.

El circuito FMTLE también se utiliza para comprobar si el acceso es alineado y si está dentro del rango permitido de direcciones (dirdNA, dirErr). En cualquiera de los dos casos la activación de la señal correspondiente es utilizada por el autómata de control para que actúe en consonancia.



**Figura 4.48** Esquema de la memoria de datos con los circuitos formateadores. La activación de la señal dirdNA indica dirección no alineada y la activación de la señal dirdErr indica que la dirección está fuera del rango permitido.

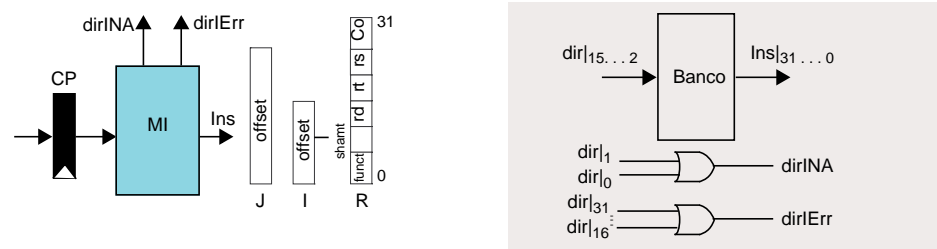
El circuito FMTLE alinea el dato proveniente del banco de registros para actualizar los bancos de memoria correspondientes en una operación de escritura. Para ello se utilizan las señales de control opMD.

La función del circuito FMTL es posicionar los datos leídos en los bits menos significativos de la salida y rellenar el resto de bits más significativos, hasta 32 bits, con ceros o unos en función del código de operación y del dato leído de la memoria de datos. En la Figura 4.28 se ha mostrado la codificación de la señal de control de la memoria de datos.

## Segundo nivel: memoria de instrucciones

Las instrucciones se almacenan en posiciones de memoria alineadas a cuatro bytes. En consecuencia, los dos bits menos significativos de una dirección, utilizada para leer una instrucción, son siempre cero. Si este no es el caso se genera una señal de excepción (dirINA, parte derecha de la Figura 4.49).

La memoria de instrucciones está organizada de forma simple utilizando un único banco. Cada entrada de la memoria almacena 4 bytes que se corresponde con el tamaño fijo de una instrucción. En consecuencia la distancia entre las direcciones de dos entradas de la memoria de instrucciones es cuatro. Esto es, el tamaño de una instrucción. La dirección que se utiliza para acceder a la memoria de instrucción es el valor almacenado en el registro CP (parte izquierda de la Figura 4.49).



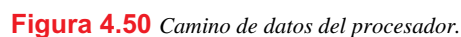
**Figura 4.49** Memoria de instrucciones.

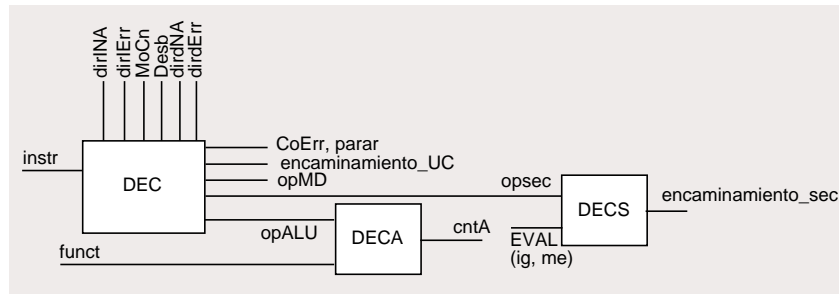
El tamaño disponible en la memoria de instrucciones es  $2^{16}$  bytes y el número de instrucciones que se puede almacenar es  $2^{14}$ . Esto es, los 16 bits más significativos de la dirección almacenada en el registro CP no se utilizan. Si estos bits son distintos de cero se genera una señal de excepción (dirIErr, Figura 4.49). La activación de las señales dirINA y dirIErr determina que el autómata de control también active la señal parar.

## Cuarto nivel: autómeta de control

El autómata de control se encarga de decodificar las instrucciones y generar las señales de control del camino de datos y de las unidades funcionales. Las entradas del autómata son una instrucción y señales de condición que se evalúan en el camino de datos.

Procesador: arquitectura, camino de datos y control



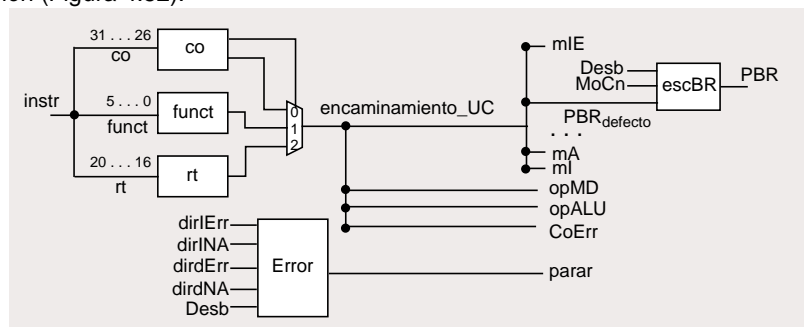


**Figura 4.51** Módulos en el autómata de control. DEC: decodificación y gestión de las señales de condición. DECA: segundo nivel de decodificación para el control de la ALU. DECS: segundo nivel de decodificación para las instrucciones de secuenciamiento condicional.

## Quinto nivel: organización del decodificador

El decodificador (DEC) se divide en varios decodificadores o partes para considerar los casos en los cuales es necesario decodificar campos adicionales al campo código de operación. En concreto, los valores cero (SPECIAL) y uno (REGIMM) del código de operación requieren decodificar el campo funct y el campo rt respectivamente (“Codificación de las instrucciones”).

Una implementación es utilizar tres decodificadores y posteriormente efectuar una selección (Figura 4.52).



**Figura 4.52** Organización del decodificador DEC. El módulo co utiliza los bits del código de operación de una instrucción para generar las señales de salida y la señal de control del multiplexor. Los otros módulos utilizan los bits mostrados para generar las señales de salida.

Todos los decodificadores establecen valores para las señales de encaminamiento en el camino de datos y para las señales de control de las unidades funcionales (ALU y MD). El decodificador denominado co determina el valor de la señal que controla el multiplexor de salida.

En cada decodificador la decodificación se efectúa en dos niveles. En primer lugar se utilizan los dos o tres bits más significativos y posteriormente los tres bits menos significativos.

La salida del decodificador (DEC) para controlar la memoria de datos es opMD. El bit más significativo (quinto) indica que es una operación de acceso a memoria (Figura 4.53). El cuarto bit indica si es una operación de lectura o escritura y se utiliza para establecer el valor de la señal permiso de escritura. El tercer bit se utiliza en instrucciones load para indicar si el dato que se lee de memoria se interpreta como número entero o como natural. Los dos bits menos significativos indican el tamaño de la operación de acceso a memoria: palabra (11), 16 bytes (01), un byte (00).

	Ac	LE	EnNt	tam	
	4	3	2	1	0
opMD (4... 0)	1	Co(3)	Co(2 . . . 0)		

**Figura 4.53** Señal de control de la memoria de datos: opMD.

Las señales de error, la señal de desbordamiento o la señal de movimiento condicional se validan en los módulos que las generan. Por ejemplo, la señal dirdNA no se activa si la instrucción no es de acceso a memoria.

El decodificador al observar la activación de una de las señales de error o desbordamiento activa la señal parar (Figura 4.51). Esta última señal también se activa al decodificar las instrucciones syscall y break.

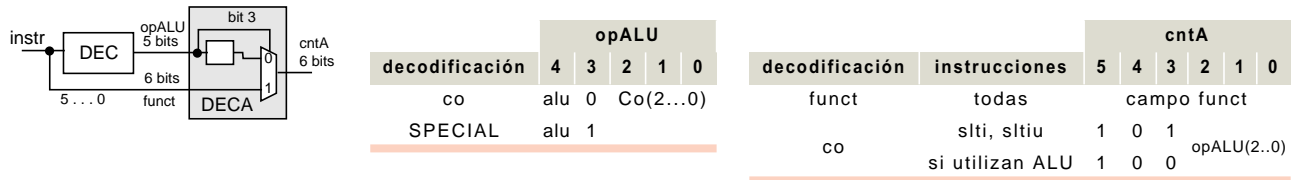
La señal de permiso de escritura en el banco de registro está incluida en el conjunto de señales denominadas encaminamiento. Las señales de condición desbordamiento (Desb) y movimiento condicional (MoCn) modifican el valor por defecto de la señal de permiso de escritura en el banco de registros (módulo EscBr, Figura 4.52).

## Quinto nivel: control de la ALU

La salida del decodificador para gestionar el control de la ALU es opALU (Figura 4.51). El quinto bit de opALU indica si hay que utilizar la ALU (Figura 4.54). El cuarto bit indica si hay que utilizar el campo funct de la instrucción para determinar la operación que se realiza en la ALU. Si este último es el caso, los valores del campo funct se utilizan para codificar la operación en la ALU. Los tres bits menos significativos de opALU sólo son de interés cuando es suficiente el campo Co para decodificar la instrucción.

Las señales de control de la ALU (módulo DECA) se generan a partir de la salida del módulo DEC y del campo funct de la instrucción (Figura 4.54). En el caso de instrucciones con código de operación SPECIAL los bits de control de la ALU (cntA) son directamente los bits del campo funct. En instrucciones en las que sólo se necesita decodificar el campo

Co los tres bits menos significativos de opALU son los tres bits menos significativos del campo Co. El resto de bits de opALU está determinado por el diseño del encaminamiento de la salida de cada unidad funcional de la ALU hacia la salida de la alu.

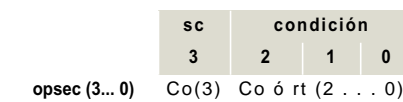


**Figura 4.54** Generación de las señales de control de la ALU: cntA. El acrónimo co indica instrucciones en las que sólo es necesario decodificar el campo código de operación. El acrónimo funct indica que hay que decodificar el campo funct después del código de operación (SPECIAL).

## Quinto nivel: control de secuenciamiento condicional

En la Figura 4.50 y en la Figura 4.51 se muestra el módulo DECS, que gestiona el secuenciamiento condicional, desagregado del módulo de decodificación (DEC). La comunicación entre los dos módulos se efectúa mediante la señal opsec.

Los tres bits menos significativos de opsec son iguales a los tres bits menos significativos del código de operación (Co) o del campo rt si el código de operación es RGIMM (Figura 4.55). El bit más significativo indica que se está interpretando una instrucción de secuenciamiento condicional y hay que tener en cuenta las salidas del módulo EVAL, el cual evalúa condiciones a partir de los datos leídos del banco de registros.



**Figura 4.55** Señal de control del secuenciamiento condicional: opsec.

## Espacio lógico

El código de un programa empieza en la dirección x"00000040" y las instrucciones se almacenan en posiciones contiguas de memoria a partir de dicha dirección.

En las direcciones previas del espacio lógico se almacenan instrucciones que: a) inician el registro que contiene la dirección que indica la cabeza de la pila, y b) instrucciones que establecen como valor del registro CP la primera dirección del programa. A esta secuencia de instrucciones la denominamos código de inicio (Figura 4.56).



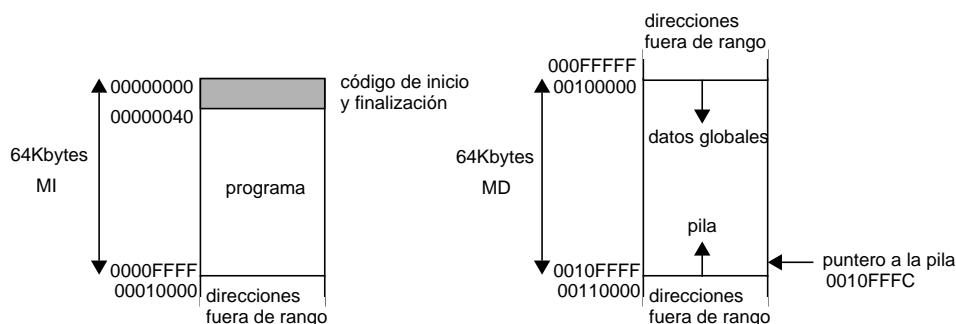
Después de código de inicio hay una secuencia de instrucciones, denominada código de finalización, que suspenden la interpretación de instrucciones. La última instrucción de un programa establece como valor del registro CP la primera dirección del código de finalización.

direc.	leng. maq.	Código de inicialización y finalización	Comentarios
		.text	
		.set noat	
		.set noreorder	
		.set mips32	
		stack_pointer = 0x0010FFFC	indicación de la dirección de la cabeza de la pila
		# código de inicialización	
		# sp: se utiliza el registro r29	
0:	3c1d0001	lui \$sp,0x10	Código de inicio
4:	37bdfffc	ori \$29,\$29,0xFFFC	
8:	0c000010	jal main	
c:	00000000	nop	Código de finalización
10:	00000000	nop	
14:	0000000d	nop	
18:	00000000	nop	
1c:	0000000d	break	
20:	00000000	nop	

**Figura 4.56** Códigos de inicio y finalización. La primera columna de la izquierda indica los bits significativos de la dirección en hexadecimal. La segunda columna muestra la instrucción en lenguaje máquina. La siguiente columna muestra la instrucción en ensamblador.

Los datos globales se almacenan a partir de la dirección x"00100000". La primera posición de la pila es la dirección x"0010FFFC" y su tamaño se incrementa hacia direcciones decrecientes.

El código de un programa y los códigos de inicio y finalización se almacenan en la memoria de instrucciones (MI, Figura 4.57). Los datos globales y la pila se almacenan en la memoria de datos (MD).



**Figura 4.57** Espacio lógico y asociación a las memorias del procesador.

En el Programas de prueba en la página 235 se muestra un programa que incluye la mayoría de las instrucciones que interpreta el procesador. Este programa es el que está almacenado por defecto en MI del camino de datos que se suministra en la práctica y es de utilidad para comprobar el funcionamiento del procesador.

## Señal de reloj, señal de inicio y elementos de almacenamiento

Los registros PCP y CP se actualizan en el flanco ascendente de la señal de reloj (Figura 4.50 y Figura 4.58). Al activar la señal inicializar (Figura 4.8, señal PCERO en la Figura 4.72) el registro PCP toma el valor cuatro y el registro CP toma el valor cero. En consecuencia, se lee de la memoria de instrucciones (MI) la instrucción que está en la dirección cero.

El banco de registros se actualiza en el flanco ascendente de la señal de reloj si el permiso de escritura (PBR) está activado (Figura 4.58).

Un banco de la memoria de datos se actualiza cuando el permiso de escritura está activado (PMD, Figura 4.47), el banco ha sido seleccionado (SCx, Figura 4.47) y el nivel de la señal de reloj es cero (Figura 4.58).



**Figura 4.58** Señal de reloj y actualización de los elementos de memorización. El retardo de actualización no está a escala.

## Simulación


El circuito con el procesador descrito se encuentra en el fichero "Camino\_datos.cct". En este fichero se distingue el camino de datos y la señal PCERO (inicializar).

## Preparación de la simulación

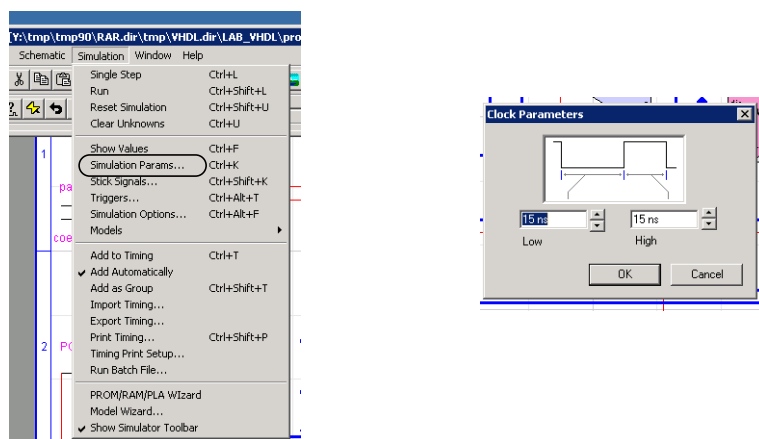
Utilizando la paleta de control debe pararse la simulación (Figura 4.59).



**Figura 4.59** Parar la simulación.

La señal rljP es la salida del elemento reloj  que controla la simulación. La señal reloj, que controla el procesador serie, es exactamente idéntica a la señal rljP una vez se ha iniciado la simulación.

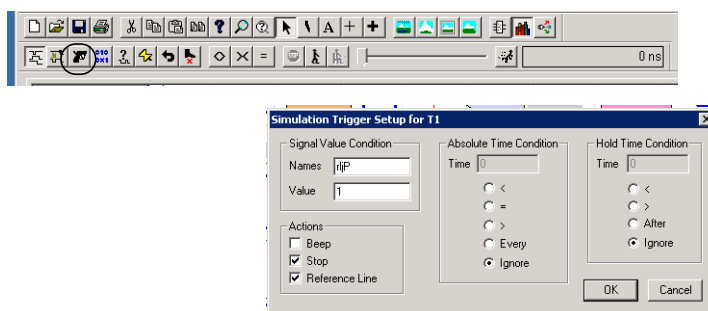
**Periodo de la señal de reloj (rljP).** El periodo de la señal de reloj se puede observar seleccionando el elemento reloj y siguiendo posteriormente los pasos que se muestran en la parte izquierda de la Figura 4.60. Para modificar el periodo de la señal de reloj se modifican los valores que se muestran en la ventana derecha de la Figura 4.60.



**Figura 4.60** Visualización del periodo de la señal de reloj y modificación del mismo.

#### Marcas en la ventana de tiempo y suspensión automática de la simulación.

Efectúe las operaciones que se indican en la Figura 4.61.

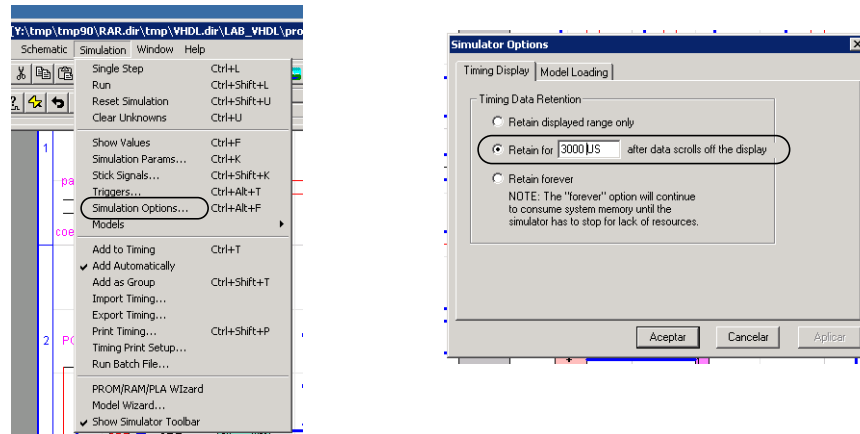


**Figura 4.61** Marcas en la ventana de tiempo y suspensión de la simulación.

Con la acción de indicar la señal rljP, el valor 1 y marcar la etiqueta "Reference Line", podremos observar, en la ventana Timing, una línea vertical cada vez que la señal rljP toma el valor uno.

Igualmente es de utilidad que la simulación se suspenda en cada ciclo de reloj. Para ello se marca en la ventana de la Figura 4.61 la etiqueta Stop.

**Tiempo de simulación que se almacena.** También interesa guardar más unidades de tiempo simuladas que las que se visualizan en la ventana de tiempo. En la Figura 4.62 se muestran los pasos que hay que efectuar para indicar un intervalo de tiempo.



**Figura 4.62** Modificación del tiempo de simulación que se almacena.

**Número de ciclos observados en la ventana de tiempo.** Para incrementar o disminuir el número de ciclos de simulación que se observan en la ventana tiempo se utiliza la paleta de control de la simulación (Figura 4.63). Pulsando el icono X un ciclo se representa en menos espacio en la escala de tiempo. En función de la escala de tiempos actual, al pulsar el icono X, se modifican las unidades de la escala de tiempo, las cuales se muestran en la izquierda de la ventana de tiempo o se modifican los valores numéricos en la escala de tiempo. Si se pulsa el icono <> se incrementa el espacio utilizado para representar un ciclo en la escala de tiempo.



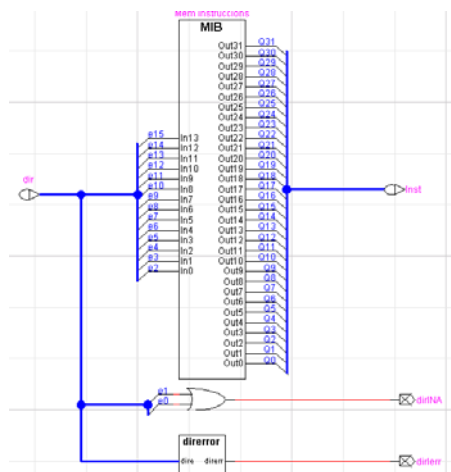
**Figura 4.63** Paleta de control. Iconos para modificar la escala en la ventana de tiempo.

## Almacenamiento de un programa

Un programa consta de cinco ficheros. Uno de ellos contiene las instrucciones en lenguaje máquina y el resto de ficheros almacena los datos.

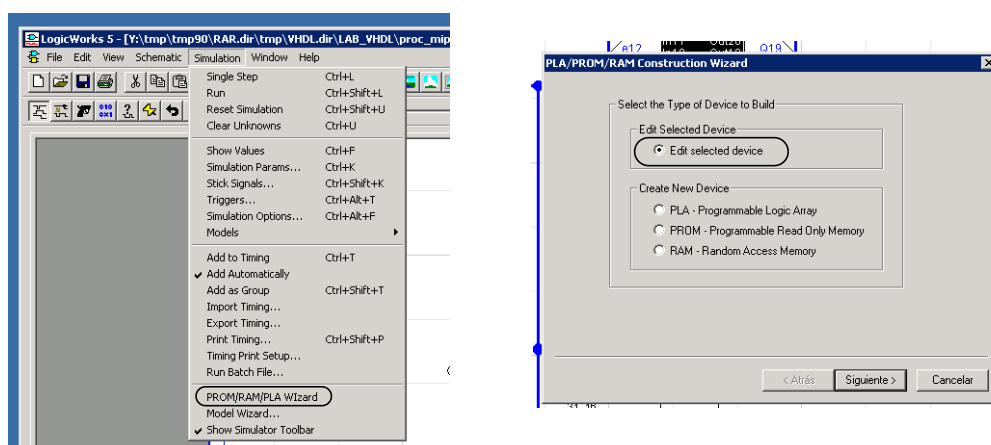
En primer lugar la simulación debe estar parada y la señal PCERO debe tener el valor cero. Para almacenar el código en la memoria de instrucciones (MI) deben seguirse los siguientes pasos:

- Seleccione en el fichero “Camino\_datos.cct” el módulo MI. Pulse dos veces el botón izquierdo del ratón y se muestran los elementos que constituyen el módulo (Figura 4.64).



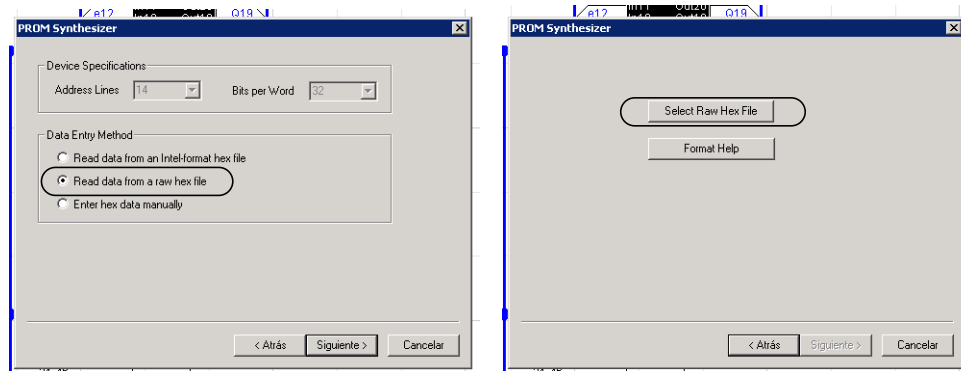
**Figura 4.64** Detalle del módulo MI.

- Seleccione el módulo MIB. Siga los pasos que se muestran en la Figura 4.65. En la pantalla emergente (parte derecha de la figura), seleccione “Edit selected device” y posteriormente “Siguiente”.



**Figura 4.65** Pasos para almacenar el código de un programa.

- En la nueva pantalla, mostrada en la parte izquierda de la Figura 4.66, seleccione “Read data from a raw file”. En la siguiente pantalla (parte derecha de la Figura 4.66), seleccione “Select Raw Hex File”.



**Figura 4.66** Selección del fichero que contiene el código de un programa.

- Seguidamente, emerge una ventana que permite recorrer el árbol de directorios. Una vez en el directorio, seleccione el fichero de código (nombre\_codigo.hex). Al volver a la pantalla previa, seleccione “Siguiente”.

Antes de almacenar los datos del programa debe de haber establecido que la simulación se suspenda de forma automática.

El siguiente paso es hacer que la señal PCERO tome el valor uno con el objetivo de inicializar los registros PCP y CP. Para ello, además de actuar sobre el conmutador hay que activar la simulación (Figura 4.67). La simulación se suspende en el primer flanco ascendente de la señal de reloj si la parada automática, descrita previamente, está activada (Figura 4.68), lo cual es necesario en esta fase.



**Figura 4.67** Activación de la simulación.

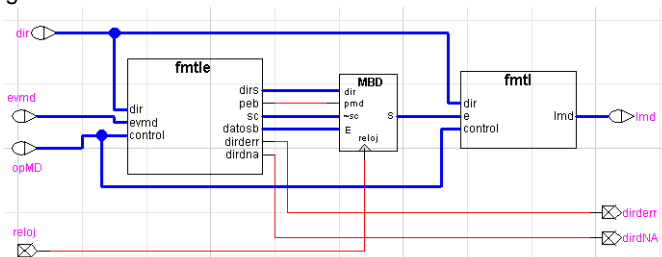
Una vez se han inicializado los registros PCP y CP y se ha suspendido la simulación, se procede a almacenar en la memoria de datos (MD) la información de datos, correspondiente al programa que se ha almacenado en la memoria de instrucciones (MI).



**Figura 4.68** Simulación suspendida.

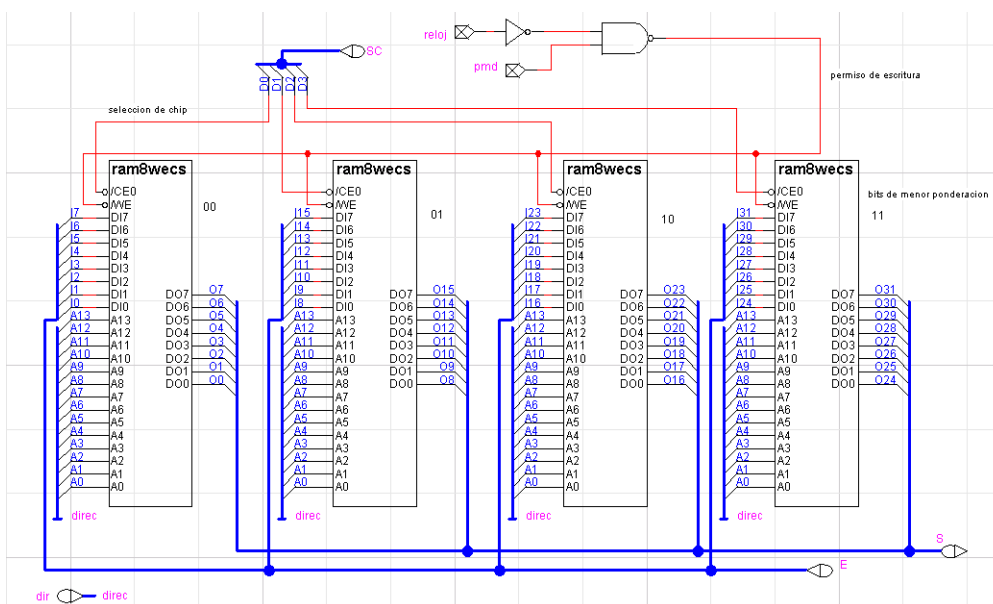
Para almacenar un programa en la memoria de datos (MD) deben seguirse los siguientes pasos:

- Seleccione en el fichero “Camino\_datos.cct” el módulo MD. Pulse dos veces el botón izquierdo del ratón y se muestran los elementos que constituyen el módulo. En la Figura 4.69 se muestra el contenido del módulo.



**Figura 4.69** *Detalle del módulo MD.*

- Ahora seleccione el módulo MBD. En la Figura 4.69 se muestra el contenido del módulo. Cada banco “ram8wecc” almacena un byte y los bancos están entrelazados a nivel de byte. Esto es, bytes consecutivos se almacenan en banco consecutivos. El banco que está espacialmente más a la izquierda almacena los bytes cuya dirección tienen ceros en los dos bits menos significativos. El siguiente banco hacia la derecha almacena los bytes cuya dirección tiene un uno en el bit menos significativo y un cero en el siguiente bit significativo. El resto de bancos sigue el mismo patrón en cuanto a las direcciones de los bytes que almacenan.



**Figura 4.70** *Detalle del módulo MBD.*

- Seguidamente debe almacenar los ficheros de datos en cada uno de los bancos “ram8wecs”. El fichero que debe almacenarse en cada elemento está identificado.

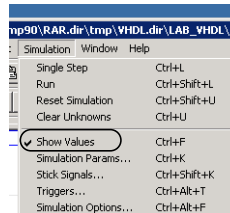
además de por caracteres alfabéticos, por dos dígitos que identifican el valor de los dos bits de menor ponderación de la dirección de los bytes que contiene (nombre\_datosxx.hex, tomando xx los valores 00, 01, 10 y 11).

- Para almacenar la información en cada banco siga los mismos pasos que se han indicado para almacenar el código.

Después de almacenar los datos establezca el valor cero en la señal PCERO y active la simulación.

## Inicio de la simulación

Antes de empezar a interpretar instrucciones es de interés activar la visualización de los valores que transportan los buses del circuito. Para ello se efectúa la acción que se muestra en la Figura 4.71.



**Figura 4.71** Activación de la observación de los valores que transportan los buses y cables en un circuito.

En la Figura 4.72 se muestran los valores que se observan en un ciclo determinado. La información que se muestra en un bus representa todos los bits que transporta el bus y está codificada en hexadecimal. Entonces, cuando sólo se conectan algunos bits de un bus a un conector hay que extraer la información del valor a partir de la información que representa el valor en todo el bus. Por ejemplo, el bus que sale del módulo MI transporta la instrucción "20010FFF". El bit menos significativo está a la derecha. Entonces, el campo denominado literal (shamt) es "0FFF".

Por otro lado, tenga en cuenta que el valor que transporta un bus sólo se muestra una vez. Por ejemplo, en la Figura 4.72, la salida del puerto de lectura L2 del banco de registros se muestra cerca de la entrada "evmd" del módulo MD. Este valor también es entrada en la entrada cero del multiplexor ubicado en la entrada inferior de la ALU (b) y en la entrada b del módulo eval.

Finalmente, tenga en cuenta que LogicWork puede modificar, cuando se para la simulación y se vuelve a activar, la ubicación en el circuito de la ventana de visualización que muestra los valores que transportan los buses.





