Simulador WepSIM

Versión 1.0

Félix García Carballeira
Alejandro Calderón Mateos
Javier Prieto Cepeda
Saul Alonso Monsalve

Grupo de Arquitectura de Computadores
Universidad Carlos III de Madrid
noviembre de 2016

Índice general

Índice alfabético **39**

Capítulo 1

Simulador WepSIM

WepSIM [1], es una plataforma, que integra el uso de la microprogramación con la programación en ensamblador. Permite la definición de diferentes juegos y formatos de instrucciones, la definición del microcódigo asociado a dicho juego de instrucciones y la ejecución de programas en ensamblador que utilizan las instrucciones máquina definidas previamente. Es posible definir instrucciones de distintos tipos de procesador (MIPS, ARM, Intel, Z80, etc.) lo que facilita posteriormente cargar un programa que use el conjunto de instrucciones definido. El simulador incluye como ejemplo un subconjunto de las instrucciones del MIPS, pero es posible definir instrucciones de otros conjuntos de forma similar.

WepSIM se basa un procesador elemental de 32 bits, que direcciona la memoria por bytes, con una unidad de control microprogramada, que utiliza secuenciamiento implícito. Utiliza un bus interno y dispone de un banco con 32 registros. Para simplificarlo, se ha decidido que únicamente conste de una Unidad Aritmético Lógica de números enteros. El mapa de memoria y de entrada/salida se encuentran separados. La descripción detallada del procesador se encuentra en el capítulo 2.

En WepSIM, una vez ensamblado un programa en ensamblador, permite realizar simulaciones del código visualizando la activación de las señales del procesador y el tránsito de datos a través de los buses y registros. De esta forma, WepSIM, ofrece 3 funcionalidades diferentes: la definición de un juego de instrucciones y su microcódigo, la creación de programas en ensamblador que utilizan dicho juego de instrucciones y la depuración y simulación del microcódigo diseñado y de los programas en ensamblador creados.

El simulador incluye tres funcionalidades:

1. La definición de un determinado juego de instrucciones y su microcódigo.
2. La definición y carga de programas en ensamblador.
3. La visualización de las señales de control y caminos de datos necesarios para la ejecución de un determinado programa. El simulador incluye los principales aspectos relacionados con el uso del simulador (ejecución paso a paso de instrucciones o microinstrucciones, puntos de ruptura, señales interactivas, etc.)

El resto del documento se estructura de la siguiente forma: el capítulo 2 realiza una descripción detallada del procesador WepSIM, y el capítulo 3 describe los principales aspectos relacionados con el uso del simulador.

[1] El simulador se encuentra disponible en http://www.arcos.inf.uc3m.es/ ec-2ed

Capítulo 2

El procesador WepSIM

El procesador WepSIM es un procesador de 32 bits que presenta las siguientes características:

- Direcciona la memoria por bytes. El funcionamiento de la memoria puede ser síncrono o asíncrono (en la actual versión del simulador, la memoria es síncrona. En versiones futuras se incluirá su comportamiento asíncrono).
- El campo de operación de todas las instrucciones es fijo y su tamaño es de 6 bits.
- Consta de un banco de 32 registros visibles al programador.
- El mapa de memoria y de E/S están separados.
- Utiliza una unidad de control microprogramada con secuenciamiento implícito.

Sobre este procesador se puede definir diferentes conjuntos de instrucciones (MIPS, ARM, etc.), se pueden renombrar los registros y se pueden definir de igual forma pseudoinstrucciones. La Figura 2.1 muestra la estructura general del procesador WepSIM.

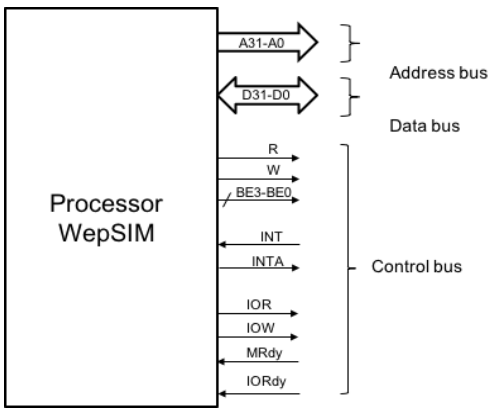


Figura 2.1: Estructura general del procesador WepSIM

2.1. Organización y elementos del procesador

La Figura 2.2 muestra la estructura de un procesador elemental con sus principales componentes y su conexión con la memoria principal. En las siguientes secciones se detallan cada uno de estos componentes.

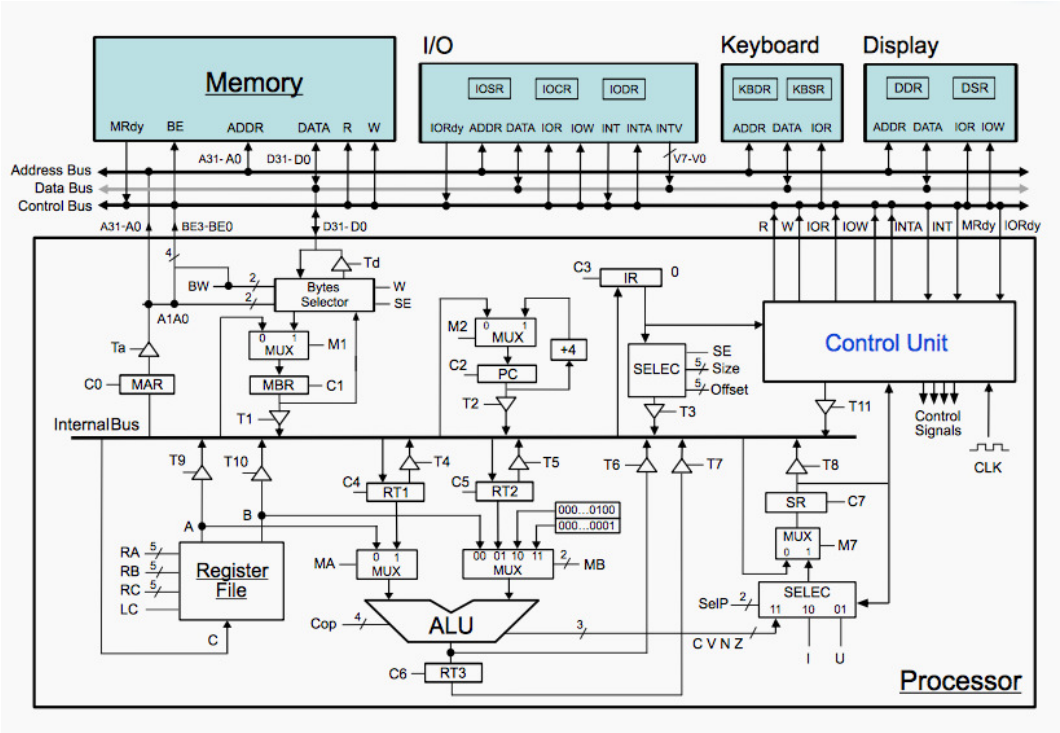


Figura 2.2: Estructura de un procesador elemental

2.1.1. Registros

Los registros están formados por un conjunto de biestables, cada uno de los cuales puede almacenar un bit. Los registros actúan como memoria interna dentro del procesador y su tamaño es de 32 bits. La Figura 2.3 ilustra la estructura de un registro para un ancho de palabra de n bits. La señal *carga* (*Load*) se utiliza para realizar la carga en el registro del dato situado a la entrada. Típicamente, la carga en los registros se realiza en los flancos de la señal, bien en el flanco de subida de la señal de carga o en el flanco de bajada de dicha señal (*falling edge*). Adicionalmente pueden existir otras señales, como por ejemplo, la señal *clear* utilizada para poner a cero el contenido del registro. En la figura se puede ver el funcionamiento de la señal de carga. La entrada debe mantenerse establece un cierto tiempo, denominado *tiempo de estabilización* (*Setup Time*) antes de que se produzca la señal de carga (por flanco de bajada de la señal en el caso de la figura). Además, el dato de entrada debe mantenerse estable un cierto tiempo, denominado *tiempo de mantenimiento* (*Hold Time*), una vez realizada la señal de carga y después de la señal de flanco.

Dentro del procesador los registros se organizan en un *banco de registros*. El banco de registros del procesador es un elemento fundamental dentro de la organización del mismo. El banco de registros de WepSIM consta de un conjunto de 32 registros de 32 bits que pueden leerse y escribirse indicando el número del registro que se desea leer o escribir. En la figura 2.2 se ilustra un ejemplo de banco de registros. Éste consta de dos puertas de lectura y una de escritura. Las señales RA y RB se utilizan para

indicar el número del registro cuyo contenido se quiere volcar por las salidas A y B, respectivamente. La señal RC se utiliza para indicar el registro sobre el que se quiere escribir el dato que se sitúa en la puerta de escritura E. La señal LC se utiliza para indicar la carga del dato situado en la puerta C en el registro indicado por RC. Las señales de selección de los registros (RA, RB y RC) tienen 5 líneas de un bit, lo que permite seleccionar un total de 32 registros. En este banco de registros, el registro 0 tiene su valor cableado a 0.

2.1.2. Tipos de registros

Los registros dentro del procesador WepSIM se pueden clasificar atendiendo a varias categorías.

Registros visibles al programador. Son aquellos que se pueden utilizar desde el juego de instrucciones del procesador. Este tipo de registros se emplea para reducir el número de accesos a memoria, almacenando en los mismos datos que son frecuentemente accedidos por el procesador. Así por ejemplo, el índice utilizado para recorrer un array conviene almacenarlo en un registro, puesto que se accede a él repetidas veces. Para el procesador WepSIM los registros se pueden etiquetar de distintas formas, por ejemplo: Ri ($R0 \ldots R31$). Se puede indicar, asimismo el valor cableado que tiene un determinado registro, e indicar qué registro actúa como puntero de pila.

Registros de control y estado. Son aquellos registros que se emplean para controlar el funcionamiento del procesador y en la mayor parte de los sistemas actuales no son visibles al programador. Entre ellos se encuentran los siguientes:

- Contador de programa (PC, *program counter*): registro que contiene la dirección de memoria donde se almacena la siguiente instrucción a ejecutar. La carga en el PC del procesador de la Figura 2.2 se realiza con la señal de control $C2$. El multiplexor situado a la entrada del PC permite seleccionar qué se carga en el PC, o bien el contenido que viene por el bus interno (entrada 0 del multiplexor controlado por la señal $M2$) o bien el contenido procedente del sumador (entrada 1 del multiplexor) y que se emplea para que el contador de programa apunte a la siguiente dirección de memoria. Nótese que este sumador incrementa la dirección almacenada en el contador de programa en 4 unidades, dado que el procesador direcciona la memoria por bytes y las palabras en este procesador son de 4 bytes.

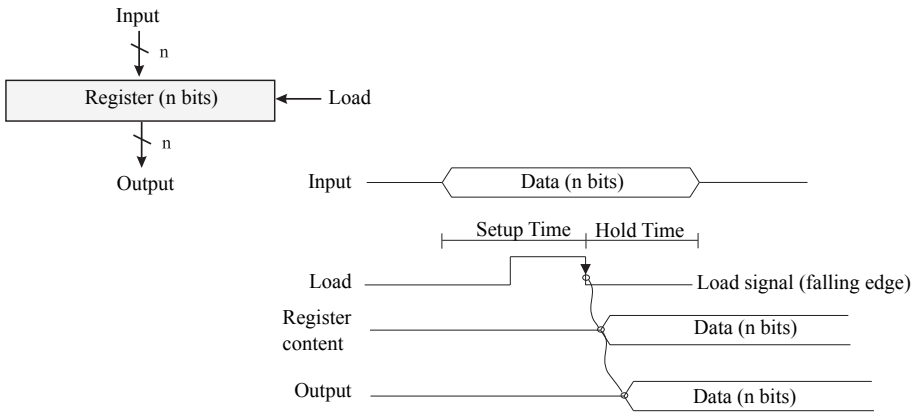


Figura 2.3: Esquema de un registro y señales necesarias para su utilización

- Registro de instrucción (IR, *instruction register*): registro que almacena la instrucción máquina que se está ejecutando en un momento determinado. La carga en este registro se realiza utilizando la señal $C3$. Este registro lo utiliza la unidad de control para poder decodificar la instrucción.

- Puntero de pila (SP, *stack pointer*): contiene la dirección de la cima de la pila. Como se verá en el Capítulo 3, se puede definir como puntero de pila cualquier registro del banco de registros.

- Registro de dirección de memoria (MAR, *memory address register*): almacena la dirección de la posición de memoria a leer o escribir. La carga se realiza en este registro activando la señal $C0$. En este procesador se puede especificar el tamaño de acceso a memoria (byte, dos bytes o una palabra de cuatro bytes), mediante las señales de control BE de la memoria. Mediante esta señal se especifica a la memoria qué bytes del bus de datos son los activos en un acceso a memoria. Cuando se especifica que todos los bytes son activos el acceso se realiza a una palabra, cuando se especifica que solo un byte es el activo, el acceso se realiza a un byte individual. En la sección 2.1.5 se describirá con más detalle el acceso a la memoria en este procesador.

- Registro de datos de memoria (MBR, *memory buffer register*): en él se almacena la palabra leída de memoria o bien se coloca la palabra a escribir en memoria. La señal $C1$ se utiliza para realizar la carga en este registro. El multiplexor (controlado por la señal $M1$) situado a su entrada permite seleccionar si el dato a cargar en este registro proviene de la memoria a través del selector de bytes, que se describirá en la sección 2.1.5 o del bus interno del procesador.

- Otros registros no visibles al programador. Estos registros no se pueden utilizar desde las instrucciones máquina del computador y se utilizan como almacenamiento temporal de datos. En el procesador de la Figura 2.2 los registros $RT1$, $RT2$ y $RT3$ son de este tipo. Se utilizan las señales $C4$, $C5$ y $C6$ respectivamente para realizar la carga en ellos.

- Registro de estado (SR, *status register*): este registro incluye información sobre el estado de la ejecución de procesador y del programa en curso. Se utilizan la señal de control $C7$ para realizar la carga del dato en este registro. Este dato proviene del multiplexor que hay a su entrada. El registro de estado dispone de una serie de bits que definen, entre otras, la siguiente información:

  - Información sobre el estado de la última operación realizada en la ALU: *signo*, que indica el signo de la última operación; *cero*, si el resultado de la última operación fue 0; *acarreo*, para indicar si la última operación dio lugar a un acarreo; *igual*, para señalar si el resultado de una operación lógica fue la igualdad; *desbordamiento*, que indica si el resultado de la última operación dio lugar a un desbordamiento. Para el procesador WepWIM, son cuatro los bits de información que se almacenan sobre el estado de la última operación: si el resultado en la ALU fue cero (Z), si fue negativo (N), si se ha producido un desbordamiento (0, *overflow*) o si se ha producido un acarreo en el bit 32 (C). Esta información puede ser útil en las instrucciones de salto condicional. Por ejemplo, una instrucción que bifurca a una dirección de memoria cuando el contenido de dos registros es el mismo, se puede realizar, restando el contenido de los dos registros y analizando posteriormente si el bit Z se ha activado o no. Si se ha activado, indica que los dos registros son iguales y en este caso se procede a bifurcar, en caso contrario no. Para WepSIM los bits C V, N y Z, se corresponden con los bits 31, 30, 29 y 28 de este registro (SR_{31}, SR_{30}, SR_{29} y SR_{28})).
  - Modo de ejecución. Este bit se utiliza para indicar si el procesador está ejecutando en modo núcleo o modo usuario. En la Sección 2.4 se tratan los modos de ejecución de WepSIM. Para el procesador WepSIM, el bit correspondiente a este modo (U) se va a considerar que es el bit 0 (SR_0).

- Interrupciones habilitadas/inhabilitadas. Se utiliza para indicar si las interrupciones se encuentran o no habilitadas (véase la Sección 2.5). Se va a considerar que este bit (I) es el bit 1 (SR_1).

Circuitos selectores para el registro de instrucción y estado

Como se aprecia en la Figura 2.2 existen dos circuitos selectores asociados a los registros de instrucción y estado. En el caso del registro de instrucción (IR), el circuito selector permite seleccionar qué bits del registro de instrucción pueden volcarse al bus interno del procesador. Este circuito tiene como entrada el contenido almacenado en IR y como señales de control (generadas por la unidad de control) SE, $Size$ y $Offset$. La señal $Size$ codifica el tamaño del contenido que se quiere volcar al bus interno. La señal $Offset$ codifica el bit inicial (se indica el bit menos significativo) del dato que se quiere volcar al bus interno. Por último, la señal SE se utiliza para indicar si se quiere hacer extensión de signo sobre el dato que se quiere volcar (un valor igual a cero indicará que no se realiza extensión de signo y un valor igual a uno que sí).

A modo de ejemplo, considere que el registro de estado almacena la instrucción máquina del MIPS que realiza la suma de un registro con un valor inmediato (para este computador `addi Rd, Rf, inmediato`). Se va a considerar que el formato de esta instrucción es el siguiente:

- Los primeros 6 bits ($IR_{31} \dots IR_{26}$) codifican el código de operación.
- El registro destino (Rd) se codifica en los siguientes 5 bits ($IR_{25} \dots IR_{21}$).
- El registro fuente (Rf) se codifica en los siguientes 5 bits ($IR_{20} \dots IR_{16}$).
- El valor inmediato a sumar al registro anterior se almacena en los últimos 16 bits ($IR_{15} \dots IR_0$).

Para este ejemplo, cuando se quiere volcar el valor inmediato almacenado en el registro de instrucción al bus interno, de forma que pueda llegar a la ALU, habría que activar las señales SE, $Size$ y $Offset$ con los siguientes valores:

- $Size = 10000$, puesto que se quieren volcar 16 bits al bus interno ($16_{(10} = 10000_{(2}$).
- $Offset = 00000$, puesto que se quiere volcar al bus interno el dato almacenado a partir del bit 0 del registro de instrucción (IR_0).
- $SE = 1$, puesto que como se quiere realizar una suma, el dato a volcar al bus interno que es de 32 bits debe ir con el signo adecuado y, por tanto, debe ir con su extensión de signo correspondiente.

En cuanto al registro de estado (SR) el circuito selector gobernado por la señal de control $SelP$ se utiliza para realizar la actualización selectiva de los bits de este registro. Este circuito selector recibe cuatro entradas: el contenido actual del registro de estado, los cuatro bits provenientes de la ALU (C, V, N, y Z), y las señales de control I y U que permite cambiar el contenido de los bits que indican el modo de ejecución e interrupciones habilitadas/inhabilitadas en el registro de estado. Dependiendo del valor de la señal $SelP$, de dos bits, se actualiza una determinada información. Para WepSIM, si $SelP$ es 11 se actualiza los bits C, V, Ny Z; si su valor es 10 se actualiza el bit I, y si es 01 se actualiza el bit U, de acuerdo a la Tabla 2.1.

Cuadro 2.1: Funcionamiento del circuito selector asociado al registro de estado del procesador de la Figura 2.2

Entradas			Salidas
Bits de entrada	SR	$SelP$	Bits de salida
C, V, N, Z, I, U	$SR_{31}, SR_{30}, SR_{29}, SR_{28} \ldots SR_1, SR_0$	11	$C, V, N, Z, SR_{27}, \ldots SR_1, SR_0$
C, V, N, Z, I, U	$SR_{31}, SR_{30}, SR_{29}, SR_{28} \ldots SR_1, SR_0$	10	$SR_{31}, SR_{30}, SR_{29}, SR_{28} \ldots I, SR_0$
C, V, N, Z, I, U	$SR_{31}, SR_{30}, SR_{29}, SR_{28} \ldots SR_1, SR_0$	01	$SR_{31}, SR_{30}, SR_{29} \ldots SR_1, U$

2.1.3. Unidad aritmético lógica

La unidad aritmético lógica (ALU) es la encargada de realizar todas las operaciones dentro del procesador. Estas operaciones son las aritméticas, las lógicas y las operaciones de desplazamiento y rotación. Existen procesadores que disponen además de unidades aritméticas para realizar operaciones en coma flotante, en WepSIM no se va a contemplar operaciones en coma flotante. En la Figura 2.2 se muestra la estructura de la ALU de WepSIM interconectada al banco de registros. La ALU utiliza 4 bits de control para indicar la operación a realizar dentro de la ALU. Además está dotada de una serie de biestables, denominados *biestables de estado*, que almacenan ciertas condiciones relativas a las operaciones aritméticas que se efectúan en ella. Los valores de estos biestables se almacenan en el registro de estado e incluyen,la siguiente información:

- Acarreo (C), cuando se produce un acarreo en el bit 32. Este bit se activa en una suma de dos valores o en un desplazamiento a la izquierda cuando se genera un 1 en el bit 32.
- Desbordamiento (V), que se activa cuando se produce un desbordamiento en la operación realizada. Se produce un desbordamiento cuando al sumar dos cantidades positivas se obtiene un valor negativo, o cuando se suman dos cantidades negativas y se obtiene un valor positivo.
- Cero (Z), que se activa cuando el resultado de la última operación realizada fue cero.
- Negativo (N), para indicar que el resultado de la última operación fue negativo.

Además en el caso de una division por cero se activan los bits V y Z.

La ALU del procesador de la Figura 2.2 dispone de dos entradas procedentes de los multiplexores gobernados por las señales MA y MB. El multiplexor gobernado por la señal A permite elegir como entrada el dato almacenado en el registro $RT1$ o el procedente de la puerta de salida A del banco de registros. El multiplexor gobernado por la señal B permite seleccionar como entrada el dato almacenado en el registro $RT2$, el dato procedente de la puerta de salida B del banco de registros, el valor 4 ($000 \ldots 0100$) o el valor 1 ($000 \ldots 001$). La salida de la ALU se puede almacenar en el registro $RT3$ o llevar directamente al bus interno del procesador. La operación a realizar se especifica con las señales de control Cop e incluye las operaciones que se muestran en la Tabla 2.2.

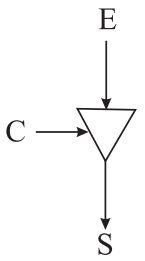
2.1.4. Interconexión entre los elementos del procesador

La interconexión entre los elementos del procesador de la Figura 2.2, al igual que en otros muchos procesadores, se realiza a través de un bus interno que permite conectar todos los elementos entre sí. Este bus interno establece un camino por el que se transmiten datos y direcciones de memoria. La conexión de un elemento al bus se realiza a través de unas puertas lógicas denominadas puertas triestado. Una puerta o *buffer* triestado es un elemento que conecta su entrada con la salida cuando se encuentra activada la señal de control que la gobierna. En la Figura 2.4 se muestra su estructura y la

Cop (Cop_3 - Cop_0)	Operación
0000	*nop*
0001	A **and** B
0010	A **or** B
0011	**not** A
0100	A **xor** B
0101	**Shift Right Logical** (A)
0110	B = number of bits to shift **Shift Right Arithgmetic** (A)
0111	B = number of bits to shift **Shift Left** (A)
1000	B = number of bits to shift **Rotate Right** (A)
1001	B = number of bits to shift **Rotate Left** (A)
1010	B = number of bits to shift $A + B$
1011	A - B
1100	A * B (with overflow)
1101	A / B (integer division)
1110	A % B (integer division)
1111	**LUI** (A)

Cuadro 2.2: Códigos de operación de la ALU

tabla de funcionamiento del mismo. Cuando la señal de control que lo gobierna (C) no está activada, el buffer triestado se encuentra en un estado de alta impedancia (Z) y en su salida no circula corriente. Cuando la señal de control se activa, la salida toma el valor que se encuentra en la entrada. Este tipo de elementos permiten seleccionar el componente del procesador que sitúa un dato en el bus interno. Solo una de estas puertas puede tener activada su señal de control a la vez. Si hubiera varias, el dato situado en el bus se corrompería. La corrupción de datos significa una alteración incontrolada de su contenido. Este caso se produciría cuando distintos componentes vuelcan de forma simultánea señales eléctricas en un mismo bus, lo que origina un nivel eléctrico incoherente. Durante el diseño de la unidad de control, hay que encargarse de que ésta no genere más de una señal de este tipo a la vez.



E	C	S
0	0	Z
1	0	Z
0	1	0
1	1	1

Figura 2.4: Buffer triestado y tabla de funcionamiento

2.1.5. Conexión con la memoria

En esta sección se describe la conexión con la memoria que utiliza el procesador WepSIM. En primer lugar se describirá el funcionamiento de la memoria y, a continuación, cómo realiza el procesador los accesos a memoria.

Funcionamiento de la memoria

La idea del simulador WepSIM es permitir configurar la memoria para que tenga un funcionamiento síncrono o asíncrono. En esta primera versión del simulador, la memoria tiene un funcionamiento síncrono.En un funcionamiento síncrono se especifica el número de ciclos necesarios para realizar una operación de lectura y escritura. En un funcionamiento asíncrono, la memoria activa la señal $MRdy$ para informar al procesador de que la operación ha finalizado. La Figura 2.5 muestra las señales de control necesarias para realizar una operación de lectura síncrona, que requiere 3 ciclos de reloj. La Figura 2.6 muestra el funcionamiento de la memoria en un acceso asíncrono.

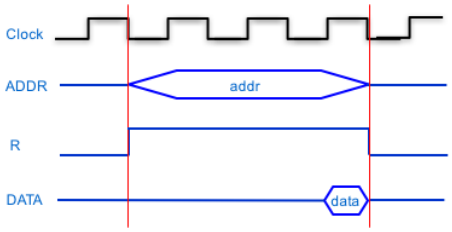


Figura 2.5: Acceso a memoria con funcionamiento síncrono

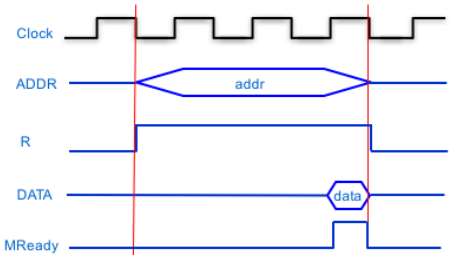


Figura 2.6: Acceso a memoria con funcionamiento asíncrono

Desde el punto de vista de la memoria, el procesador WepSIM es un procesador Little-endian, es decir, el byte menos significativo de un dato que ocupa varios bytes se almacena en la dirección de memoria menos significativa de una palabra (véase la Figura 2.7).

Las señales de acceso a memoria son las siguientes:

- $MRdy$, señal que se activa en un funcionamiento asíncrono de la memoria cuando la operación de lectura o escritura ha finalizado.
- $ADDR$, dirección de memoria sobre la que se realiza el acceso.
- $DATA$, en una operación de lectura la memoria vuelca el dato leído en estos 32 bits. En una operación de escritura, el dato a escribir se encuentra a la entrada, en el bus de datos.
- R, señal que indica una operación de lectura.
- W, señal que indica una operación de escritura.

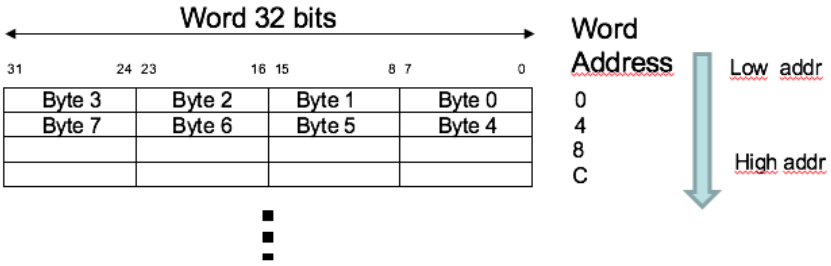


Figura 2.7: Ordenación de los bytes en memoria

- *BE*, señal de 4 bits, que indica el tamaño de acceso. En la Figura 2.8 se indica el comportamiento de estos bits en una operación de lectura y en la Figura 2.9 su funcionamiento en una operación de escritura.

Bytes en la palabra de la memoria				Selección de bytes				Salidas al bus			
D31-D24	D23-D16	D15-D8	D7-D0	BE3	BE2	BE1	BE0	D31-D24	D23-D16	D15-D8	D7-D0
Byte 3	Byte 2	Byte 1	Byte 0	0	0	0	0	---	---	---	Byte 0
Byte 3	Byte 2	Byte 1	Byte 0	0	0	0	1	---	---	Byte 1	---
Byte 3	Byte 2	Byte 1	Byte 0	0	0	1	0	---	Byte 2	---	---
Byte 3	Byte 2	Byte 1	Byte 0	0	0	1	1	Byte 3	---	---	---
Byte 3	Byte 2	Byte 1	Byte 0	0	1	0	X	---	---	Byte 1	Byte 2
Byte 3	Byte 2	Byte 1	Byte 0	0	1	1	X	Byte 3	Byte 2	---	---
Byte 3	Byte 2	Byte 1	Byte 0	1	1	X	X	Byte 3	Byte 2	Byte 1	Byte 0

Figura 2.8: Funcionamiento de la señal BE en una operación de lectura

Dato en el bus				Selección de bytes				Bytes escritos en memoria			
D31-D24	D23-D16	D15-D8	D7-D0	BE3	BE2	BE1	BE0	D31-D24	D23-D16	D15-D8	D7-D0
Byte 3	Byte 2	Byte 1	Byte 0	0	0	0	0	---	---	---	Byte 0
Byte 3	Byte 2	Byte 1	Byte 0	0	0	0	1	---	---	Byte 1	---
Byte 3	Byte 2	Byte 1	Byte 0	0	0	1	0	---	Byte 2	---	---
Byte 3	Byte 2	Byte 1	Byte 0	0	0	1	1	Byte 3	---	---	---
Byte 3	Byte 2	Byte 1	Byte 0	0	1	0	X	---	---	Byte 1	Byte 2
Byte 3	Byte 2	Byte 1	Byte 0	0	1	1	X	Byte 3	Byte 2	---	---
Byte 3	Byte 2	Byte 1	Byte 0	1	1	X	X	Byte 3	Byte 2	Byte 1	Byte 0

Figura 2.9: Funcionamiento de la señal BE en una operación de escritura

Acceso a la memoria por parte del procesador

La conexión con la memoria se realiza a través de una serie de registros y señales de control (véase la Figura 2.2). Los registros implicados son los siguientes:

- **Registro de direcciones de memoria** (MAR, *memory address register*). En este registro ha de cargarse la dirección correspondiente a una operación de lectura o escritura.
- **Registro de datos de memoria** (MBR, *memory buffer register*). En este registro se almacena el dato a escribir en memoria en una operación de escritura. En el caso de una operación de lectura, la memoria almacena en él el dato leído.

Aparte de estos dos registros, la unidad de control tiene que general las señales de control necesarias para realizar los accesos a memoria. Aparte de la señales, R, W y BE descritas en la sección anterior. la unidad de control, como se verá más adelante debe realizar las siguientes acciones para una lectura o escritura:

- Operación de lectura: debe almacenarse en MAR la dirección de memoria a leer y a continuación activar la señal de lectura, R, y la señal Ta, que permite que la dirección almacenada en el registro MAR pase al bus de direcciones. Cuando la memoria finaliza la operación, vuelca el dato leído al bus de datos. Este dato se puede cargar en el registro MBR activando la señal de carga en este registro ($C1$) y la señal $M1$.
- Operación de escritura: debe almacenarse en MAR la dirección de memoria en la cual se desea escribir y en MBR el dato a escribir. A continuación se activará la señal de escritura, W, para indicar a la memoria que realice la operación, así como las señales Ta y Td que permite volcar la dirección y el dato a escribir en el bus de direcciones y de datos respectivamente.

En WepWIM cada palabra de 32 bits consta de 4 bytes, y cada uno de estos bytes se puede direccionar de forma independiente. De este modo, las direcciones de palabras consecutivas difieren en 4. El procesador puede acceder, por tanto, a datos de 1 byte, 2 bytes o una palabra completa de 4 bytes. El tipo de acceso se realiza mediante la señal BE. Los valores para estas señales se obtienen a partir de las señales de control $BW1$ y $BW0$ y los dos bits inferiores de una dirección de memoria A_1A_0. De forma que $BE3 = BW1$, $BE2 = BW0$, $BE1 = A_1$ y $BE0 = A_0$. Esto se consigue mediante el circuito *Bytes Selector*, que permite controlar que parte de una palabra se lee o escribe en la memoria. La señal SE se utiliza para indicar si se realiza extensión de signo en la lectura de un dato de 1 o 2 bytes.

En una operación de lectura, por tanto, basta con indicar el tamaño del acceso mediante los valores de las señales $BW1$ y $BW0$, tal y como se muestra en la Figura 2.10, y Figura 2.11, para una operación de escritura. Cuando se quiere realizar una operación de lectura de 1 o 2 bytes con extensión de signo, es necesario activa la señal SE como se muestra en la Tabla 2.12

2.2. La unidad de control de WepSIM

El objetivo básico de la unidad de control es gobernar el funcionamiento del procesador permitiendo la ejecución de las instrucciones máquina que componen los programas. La unidad de control se encuentra, por tanto, realizando de forma repetida las tres siguientes acciones:

1. Lectura de la instrucción, también denominada fase de captación de la instrucción o *fetch*.
2. Decodificación de la instrucción.

Bytes en memoria				Selección de bytes				Bytes en la MBR			
D31-D24	D23-D16	D15-D8	D7-D0	BW1	BW0	A1	A0	D31-D24	D23-D16	D15-D8	D7-D0
Byte 3	Byte 2	Byte 1	Byte 0	0	0	0	0	0	0	0	Byte 0
Byte 3	Byte 2	Byte 1	Byte 0	0	0	0	1	0	0	0	Byte 1
Byte 3	Byte 2	Byte 1	Byte 0	0	0	1	0	0	0	0	Byte 2
Byte 3	Byte 2	Byte 1	Byte 0	0	0	1	1	0	0	0	Byte 3
Byte 3	Byte 2	Byte 1	Byte 0	0	1	0	X	0	0	Byte 1	Byte 0
Byte 3	Byte 2	Byte 1	Byte 0	0	1	1	X	0	0	Byte 3	Byte 2
Byte 3	Byte 2	Byte 1	Byte 0	1	1	X	X	Byte 3	Byte 2	Byte 1	Byte 0

Figura 2.10: Señales para el acceso a un dato en una operación de lectura, sin extensión de signo

Bytes en MBR				Selección de bytes				Bytes escritos en memoria			
D31-D24	D23-D16	D15-D8	D7-D0	BW1	BW0	A1	A0	D31-D24	D23-D16	D15-D8	D7-D0
---	---	---	Byte 0	0	0	0	0	---	---	---	Byte 0
---	---	---	Byte 0	0	0	0	1	---	---	Byte 0	---
---	---	---	Byte 0	0	0	1	0	---	Byte 0	---	---
---	---	---	Byte 0	0	0	1	1	Byte 0	---	---	---
---	---	Byte 1	Byte 0	0	1	0	X	---	---	Byte 1	Byte 0
---	---	Byte 1	Byte 0	0	1	1	X	Byte 1	Byte 0	---	---
Byte 3	Byte 2	Byte 1	Byte 0	1	1	X	X	Byte 3	Byte 2	Byte 1	Byte 0

Figura 2.11: Señales para el acceso a un dato en una operación de escritura

Bytes en memoria				Selección de bytes				Bytes en MBR			
D31-D24	D23-D16	D15-D8	D7-D0	BW1	BW0	A1	A0	D31-D24	D23-D16	D15-D8	D7-D0
Byte 3	Byte 2	Byte 1	Byte 0	0	0	0	0	S	S	S	Byte 0
Byte 3	Byte 2	Byte 1	Byte 0	0	0	0	1	S	S	S	Byte 1
Byte 3	Byte 2	Byte 1	Byte 0	0	0	1	0	S	S	S	Byte 2
Byte 3	Byte 2	Byte 1	Byte 0	0	0	1	1	S	S	S	Byte 3
Byte 3	Byte 2	Byte 1	Byte 0	0	1	0	X	S	S	Byte 1	Byte 0
Byte 3	Byte 2	Byte 1	Byte 0	0	1	1	X	S	S	Byte 3	Byte 2
Byte 3	Byte 2	Byte 1	Byte 0	1	1	X	X	Byte 3	Byte 2	Byte 1	Byte 0

Figura 2.12: Señales para el acceso a un dato en una operación de lectura, con extensión de signo (S es el valor del bit más significativo del byte almacenado en $D_7 - D_0$ o el bit más significativo almacenado en $D_{15} - D_8$ cuando se accede a dos bytes)

3. Ejecución de la instrucción.

Estas fases configuran el ciclo de ejecución de instrucciones. Para realizar estas acciones, la unidad de control realiza en cada una de las fases anteriores una serie de operaciones muy básicas, denominadas *operaciones elementales* o *microoperaciones*. Existen dos tipos de microoperaciones: microoperaciones de *transferencia* y de *proceso*. Estos dos tipos de operaciones se realizan cada una en un ciclo de reloj.

Las operaciones elementales de transferencia son las que involucran el movimiento de un dato de un registro a otro. Se especifican como $R_{destino} \leftarrow R_{origen}$. Un ejemplo de este tipo de operaciones elementales es la operación $MAR \leftarrow PC$, que ocurre en el ciclo de *fetch* como paso previo a la lectura de la siguiente instrucción a ejecutar. Para llevar a cabo este tipo de operaciones elementales, la unidad de control debe activar una serie de señales de control que permitan por un lado establecer un camino de datos entre los registros involucrados y por otro lado, una señal de control que permita realizar la carga en el registro destino.

La unidad de control debe activar durante un ciclo de reloj la señal $T2$, correspondiente a la puerta triestado que conecta el PC al bus interno. La activación de esta señal permite que el contenido del PC se vuelque al bus interno del procesador. En este mismo ciclo de reloj ha de activar la señal $C0$ que permite cargar en el registro MAR el dato situado en su entrada. Si se considera que la carga se realiza en el ciclo de bajada de esta señal, la carga en el registro MAR se efectuará al final del ciclo. Durante este ciclo, la unidad de control debe asegurarse de no activar ninguna señal que active cualquier otra puerta triestado. Si esto no fuera así, habría varios registros cuyas salidas coincidirían en el bus dando lugar a valores lógicos incoherentes, tal y como se comentó anteriormente.

Las operaciones elementales de proceso involucran una operación en la ALU sobre dos registros. Por ejemplo, la operación $RT3 \leftarrow RT1 + RT2$ es una operación elemental que realiza la suma de los datos almacenados en los registros $RT1$ y $RT2$. El resultado de esta suma se almacena en el registro $RT3$. Para realizar esta operación elemental, la unidad de control debe activar durante un ciclo de reloj, las siguientes señales:

- MA de forma que el multiplexor gobernado por esta señal deje pasar a la ALU el contenido del registro $RT1$. Asimismo debe activarse MB con el valor 01 para que el contenido del registro $RT2$ pase a la ALU.
- El código de operación, Cop, correspondiente a la operación de suma (1010
- La señal de carga en el registro de estado, para actualizar el valor de los biestables de estado correspondiente a la operación realizada. Esta carga se realiza activando $C7$, la señal $M7$ y la señal $SelP$ (con el valor 11) del selector situado a la entrada del registro de estado, para indicar que lo que se desea es actualizar en dicho registro los bits C, V, N y Z con los bits procedentes de la ALU.
- La señal de carga, $C6$, que permite cargar el resultado procedente de la ALU en el registro $RT3$. Al igual que antes, si se asume que la carga se realiza en el flanco de bajada de la señal $C6$, la carga en dicho registro se realizará al final de ciclo de reloj y el dato estaría disponible en dicho registro durante el ciclo siguiente.

Como se puede ver en la Figura 2.2 y 2.13, la unidad de control de WepSIM toma como entradas el código de operación (CO) que reside en el registro de instrucción, el contenido del registro de estado, información sobre las señales de interrupción y la señal de temporización del reloj. A partir de esta información se debe encargar de generar las señales de control necesarias para la correcta ejecución de

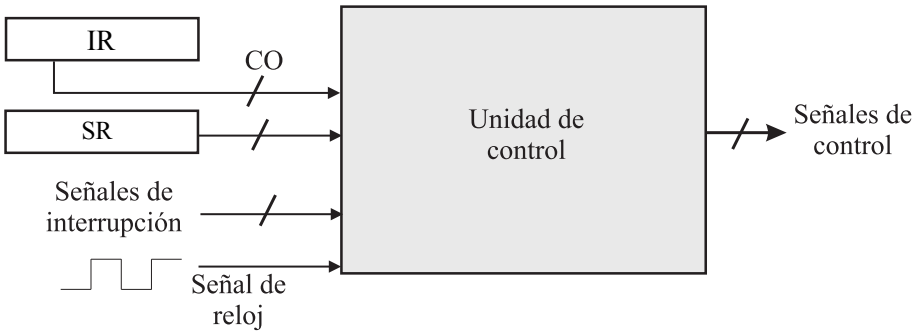


Figura 2.13: Entradas y salidas de la unidad de control

de cada instrucción. En WepSIM el código de operación se almacena en los 6 bits más significativos del registro de instrucción

WepSIM utiliza una unidad de control microprogramada, que emplea una memoria de control para almacenar el estado de las diferentes señales de control durante cada uno de los ciclos de cada instrucción. En este tipo de unidades de control, se denomina **microinstrucción** al conjunto de bits que identifican las señales de control que debe generar la unidad de control en cada ciclo de reloj. Un **microprograma** constituye la secuencia ordenada de microinstrucciones que debe generar la unidad de control para ejecutar cada instrucción máquina. Finalmente, se denomina **microcódigo** al conjunto de todas los microprogramas de un computador. En WepSIM se emplea una unidad de control con **secuenciamiento implícito**. En este tipo de unidades de control, la microinstrucción no incluye la dirección de la siguiente microinstrucción a ejecutar. La siguiente microinstrucción a ejecutar, siempre que sea del mismo microprograma, es la siguiente, es decir, la que ocupa la posición de memoria consecutiva en la memoria de control. En este tipo de unidades de control, la memoria de control debe almacenar de forma consecutiva todas las microinstrucciones asociadas a un mismo microprograma (véase la Figura 2.14).

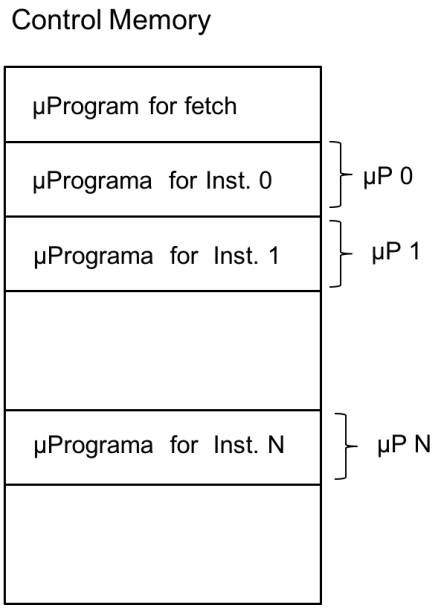


Figura 2.14: Memoria de control en una unidad de control microprogramada con secuenciamiento implícito

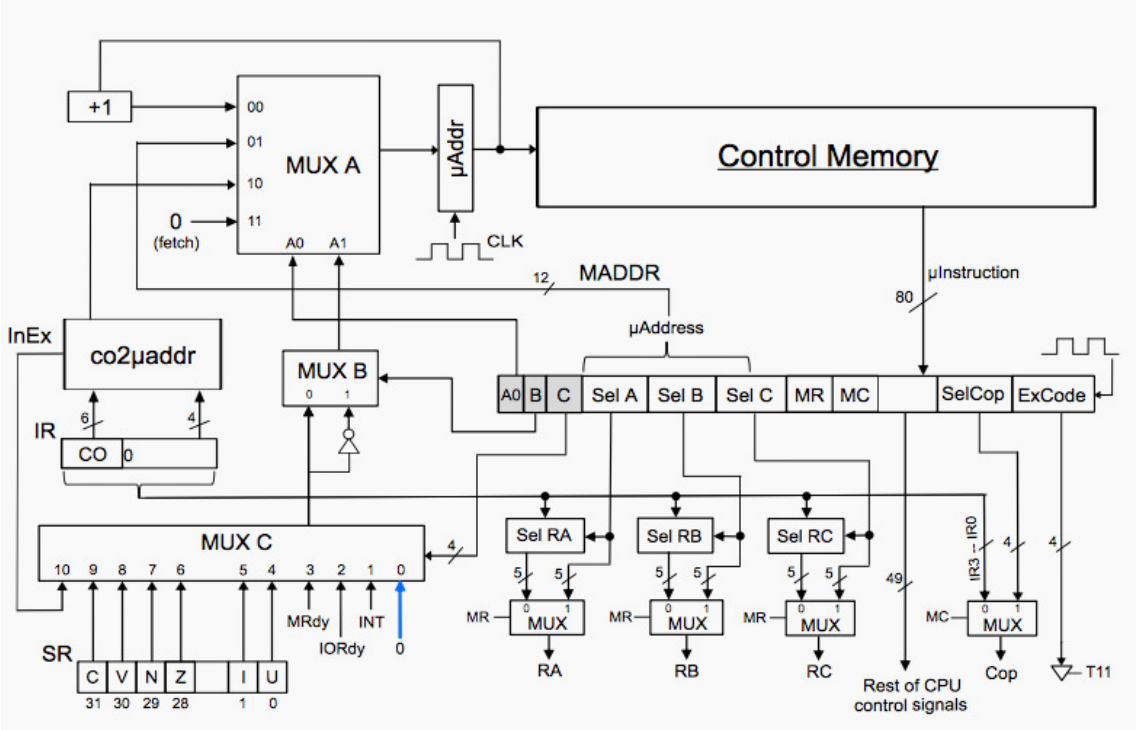


Figura 2.15: Esquema de una unidad de control microprogramada de WepSIM

La Figura 2.15 muestra la estructura de una unidad de control microprogramada de WepSIM. Incluye una memoria de control donde almacenar todas las microinstrucciones correspondientes al juego de instrucciones del computador y un secuenciador encargado de ir generando las direcciones de la memoria de control. En cada ciclo de reloj, el secuenciador debe generar la dirección de la memoria de control donde se encuentra almacenada la microinstrucción a ejecutar en ese ciclo de reloj. Esta microinstrucción contiene los valores de las señales de control que deben estar activos durante ese ciclo de reloj.

La memoria de control consta de 4096 micropalabras de 80 bits cada una (véase la Figura 2.16).

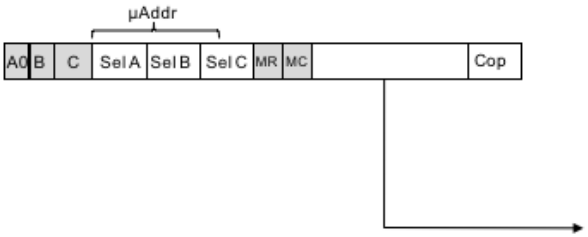
La nomenclatura general usada para el nombre las señales de control es la siguiente:

- Cx: señal de carga de registro x.
- Tx: señal de activación triestado x.
- Ry: identificador de registro para el punto de salida de valor y.
- Mx: selección en multiplexor x.

Adicionalmente la señal LC indica la escritura en el banco de registros, Cop indica la operación a realizar en la ALU, $SelP$ selecciona qué bits actualizar en el registro de estado (SR), y BW permite especificar el tamaño del dato al que se accede en memoria: una palabra completa, media palabra o un byte.

Para direccionar la memoria de control se precisan direcciones de 12 bits. Como se puede apreciar en la Figura 2.15, la unidad de control puede elegir como siguiente microdirección, utilizando para ello el multiplexor A:

- La que se encuentra en la dirección 0 ($A0A1 = 11$), es decir, donde se almacena la primera microinstrucción correspondiente al *fetch*.



C0...C7	Carga en registros
Ta, Td	Triestados a buses
T1...T10	Puertas triestado
M1, M2, M/, MA, MB	Multiplexores
SelP	Selector del registro de estado
SC	Carga en el banco de registros
SE	Extensión de signo
Size, Offset	Selector del registro IR
BW	Tamaño de la operación en memoria
R, W	Operación en memoria
IOR, IOW	Operación de E/S
INTA	Reconocimiento de interrupción
I	Habilitar/Deshabilitar interrupciones
U	Modo usuario/núcleo

Figura 2.16: Formato de la Microinstrucción en WepSIM

- La siguiente, que se obtiene a partir del sumador que se muestra en la Figura 2.15 ($A0A1 = 00$).
- La microdirección donde se almacena la primera microinstrucción correspondiente a una instrucción máquina ($A0A1 = 10$). Se obtiene a partir del elemento $CO2\mu Addr$ que se observa en la Figura 2.15. Esta memoria permite obtener la dirección de la primera microinstrucción de una instrucción máquina, y se obtiene a partir de los 6 bits del código de operación almacenados en el registro de instrucción (IR) y los cuatro bits inferiores almacenados en este registro. Posteriormente se describirán el uso de estos cuatro bits.
- Otra microdirección que se almacena en la propia microinstrucción (campo $\mu Address$ que se solapa con los campos $SelA$, $SelB$ y $SelC$ de la microinstrución), que se utiliza en las instrucciones de salto condicional ($A0A1 = 01$).

El elemento $CO2\mu Addr$ dispone de una señal InE que se activa cuando la instrucción almacena en el registro de instrucción no está definida.

Los campos de la microinstrucción que permiten controlar el direccionamiento anteriormente descrito son:

- A, que controla el bit $A0$ del multiplexor A.
- B, que controla el multiplexor B.
- C, que controla el multiplexor C.

El multiplexor C se utiliza para poder implementar microsaltos concionales. Así un valor $C = 0011$ permite obtener en la salida del multiplexor C el valor de la señal MRdy procedente de la memoria.

La generación de las señales RA, RB y RC del banco de resgistros se realiza a partir de los circuitos selectores $SelRA$, $SelRB$ y $SelRC$ controlados por las señales almacenadas en la microinstrucción

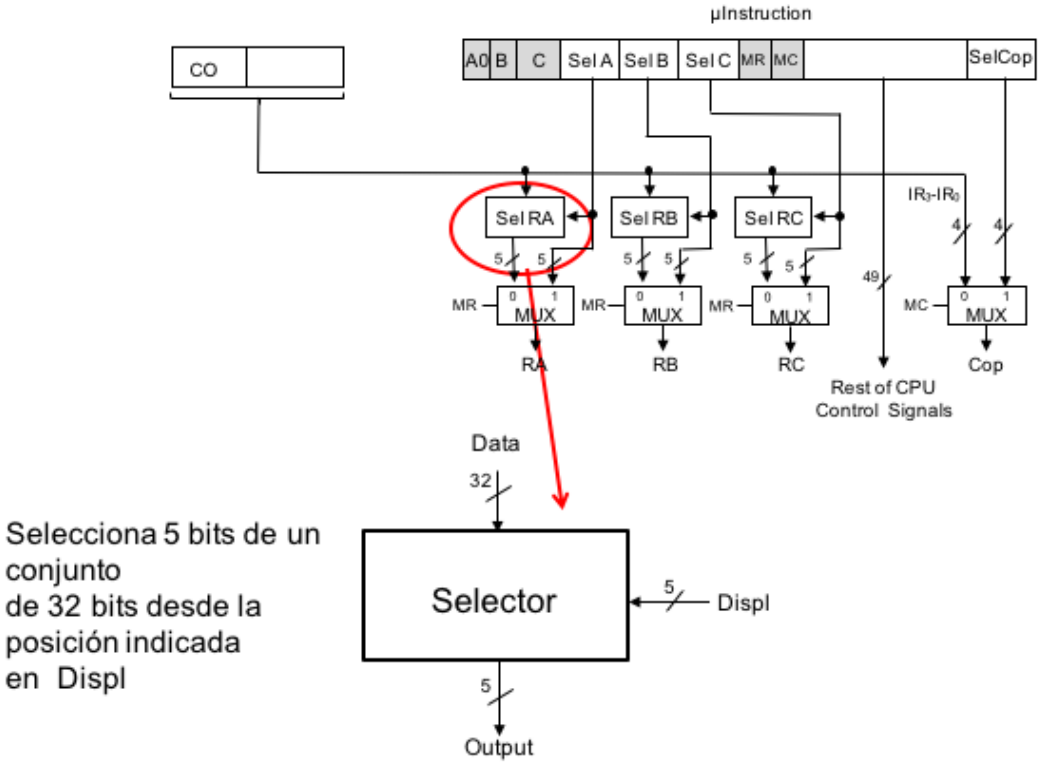


Figura 2.17: Selectores de las señales RA, RB y RC del banco de registros

$SelA$, $SelB$ y $SelC$ (véase la figura 2.17). Estos circuitos selectores toman como entrada el valor almacenado en el registro de instrucción y extraen de este valor un campo de 5 bits. El campo $SelA$ permite indicar, cuando $MR = 0$, qué parte del registro de instrucción incluye el identificador del registro a utilizar como señal RA. Así por ejemplo cuando $SelA = 00100$, el identificador del registro se obtendrá a partir de los bits $IR_4IR_3IR_2IR_1IR_0$ almacenados en el registro de instrucción. En caso de que la señal $MR = 1$ se considerará que el valor para la señal RA, que indica qué registro del banco de registros se quiere direccionar por la puerta A (véase la Figura 2.2), se almacena directamente en el campo $SelA$. Este mecanismo es similar para las señales RB y RC. Así por ejemplo, para una instrucción con un formato como el mostrado en la Figura 2.18, los valores que se deben almacenar en $SelA$, $SelB$ y $SelC$ para poder direccionar el registro $Reg1$ con la señal RC, el registro $Reg2$ con la señal RB y el registro $Reg1$ con la señal RA son:

- $SelB = 10100$, RB se obtiene de los bits 20...16 de IR.
- $SelA = 01111$, RA se obtiene de los bits 15 ... 11 de IR.
- $SelC = 11001$, RC se obtiene de los bits 25 ... 21 de IR.

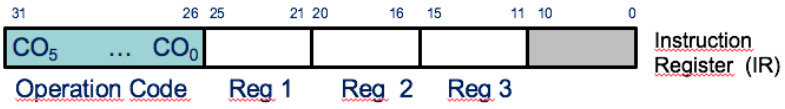


Figura 2.18: Ejemplo de formato de instrucción

Los cuatro bits inferiores almacenados en el registro de instrucción se pueden utilizan para seleccionar la operación a realizar en la ALU. Esto permite poder diseñar instrucciones aritméticas con código de operación igual a 0, por ejemplo, y diferenciar la operación a realizar con estos últimos 4 bits. Como se puede observar en la Figura 2.15 la señal MC permite indicar si el código de operación de la ALU (Cop), se obtiene de los últimos 4 bits almacenados en el registro de instrucción ($MC = 0$) o del campos $SelCop$ almacenado directamente en la microinstrucción ($MC = 1$).

Por último, los cuatro bits inferiores almacenados en la microinstrucción se pueden utilizar para almacenar el vector de interrupción (vease la Sección 2.5) correspondiente a una excepción generada como parte de la ejecución de la instrucción.

2.3. Ejecución de instrucciones

En esta sección se va a describir las operaciones elementales y señales de control necesarias para ejecutar una serie de instrucciones máquina en WepSIM.

En primer lugar se van a describir las operaciones elementales y señales de control para el ciclo de *fetch*

El ciclo de *fetch* involucra la lectura de la instrucción apuntada por el PC y su almacenamiento en el registro de instrucción IR, para su posterior decodificación. Se indican los ciclos de reloj, las operaciones elementales y las señales de control que deben activarse en cada ciclo. En este ejemplo se considera que la memoria es síncrona y necesita un ciclo de reloj para realizar un acceso a memoria.

Microinstrucción	Operaciones elementales	Señales de control
$\mu 1$	$MAR \leftarrow PC$	$T2$, $C0$
$\mu 2$	$PC \leftarrow PC + 4$ $MBR \leftarrow Memory$	$M2$, $C2$ Ta, R, $BW = 11$, $C1$, $M1$
$\mu 3$	$IR \leftarrow MBR$	$T1$, $C3$, $A0$

Como se puede observar, el ciclo de fetch comprende tres microinstrucciones. En la microinstrucción 1, durante el primer ciclo de fetch, se lleva la dirección almacenada en el PC al registro MAR. Este es el paso previo para realizar la lectura de la instrucción. En la segunda microinstrucción se realizan dos operaciones elementales en paralelo. Por un lado, se incrementa el contenido del contador de programa, de forma que apunte a la siguiente instrucción a ejecutar [1]. Para ello se utiliza el sumador asociado al PC. La salida del sumador se vuelve a cargar en el PC mediante la activación de las señales $C2$ y $M2$. Observe que la carga se realiza al final del ciclo. En esta microinstrucción también se realiza la operación de lectura de la instrucción. Para ello se activa la señal de lectura R y la señal Ta, que vuelca el dato del registro MAR al bus de direcciones. Esto permite a la memoria leer la dirección almacenada en MAR. El dato leído, correspondiente a la instrucción almacenada en PC, se vuelca al bus de datos y de él se carga en el registro MBR, para lo cual hay que activar la señal $C1$ y $M1$. También se activa la señal $BW = 11$ puesto que se va a leer de memoria una palabra completa. En las dos microinstrucciones anteriores, los valores correspondientes a las señales $A0$, B y C de la unidad de control (véase la Figura 2.15), no están activados, de forma que se selecciona en el multiplexor A de la unidad de control la entrada 00, para que se seleccione de la memoria de control la siguiente microinstrucción.

La última microinstrucción de la fase de *fetch*, se corresponde con la transferencia de la instrucción leída al registro de instrucción para su posterior decodificación. Esto se realiza mediante la operación elemental $IR \leftarrow MBR$ y la activación de las señales $T1$ y $C3$. En esta microinstrucción también se activa la señal $A0$, lo que permite seleccionar en el multiplexor A de la unidad de control la entrada 10.

[1] El PC se incrementa en 4, puesto que la memoria se direcciona por bytes y cada palabra consta de 4 bytes.

Esta entrada procedente de $Co2\mu Addr$ permite obtener la dirección de la memoria de control donde se almacena la primera microinstrucción del microprograma asociado a la instrucción máquina, que se ha leído, y que se encuentra almacenada en el registro de instrucción.

Ejecución de instrucciones de transferencia de datos. A continuación se describen las operaciones elementales necesarias para la ejecución de la instrucción del MIPS `lw reg, addr` que carga en el registro *reg* el contenido de la dirección de memoria *addr*. También se muestran las microinstrucciones que comprenden el microprograma asociado a esta instrucción máquin. Se va a considerar que el campo con la dirección ocupa los últimos 16 bits del registro de instrucción. Como se puede observar durante la primera microinstrucción se transfiere al registro MAR los 16 bits inferiores del registro de instrucción. Para ello se activa $size$ con valor 16 ($10000_{(2}$) y $Offset$ con valor 0 ($00000_{(2}$), lo que permite seleccionar los 16 bits inferiores de IR comenzando en el 15: $R_{15} \ldots R_0$. Esta dirección se vuelca sin extensión de signo (no se activa la señal SE). En la segunda microinstrucción se realiza el acceso a memoria. En la tercera microinstrucción se carga el dato leído de memoria en el registro reg indicado en la instrucción. Durante esta microinstrucción $SelC = 10101$, puesto que el bit menos significativo de la instrucción donde se almacena el campo reg es el $21 = 10101_{(2}$. Durante esta microinstrucción se activan las señales $A0$ y B para seleccionar la entrada 11 en el multiplexor A de la unidad de control. De esta forma, la siguiente microinstrucción a ejecutar será la que se almacena en la microdirección 0 de la memoria de control y que se corresponde con la primera microinstrucción del ciclo de *fetch*.

Microinstrucción	Operación elemental	Señales de control
$\mu 1$	$MAR \leftarrow IR(dir)$	$C0$, $T3$, $Size = 10000$, $Offset = 00000$
$\mu 2$	$MBR \leftarrow Memory$	Ta, R, $C1$, $M1$, $BW = 11$
$\mu 3$	$reg \leftarrow MBR$	$T1$, SC, $SelC = 10101$, $A0$, B

Ejecución de instrucciones aritmético-lógicas. En este apartado se va a considerar la ejecución de la instrucción del MIPS `add Rd, Rf1, Rf2` que suma el contenido de $Rf1$ y $Rf2$ y deja el resultado en Rd. Se va a considerar que esta microinstrucción se codifica según el formato que se muestra en la Figura 2.19. El código de operación es 000000, para indicar que se trata de una instrucción aritmético-lógica. En este caso el código de operación de la ALU se extrae de los últimos cuatro bits.

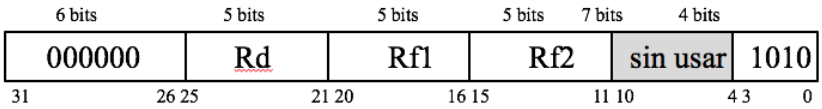


Figura 2.19: Formato de la instrucción `add Rd, Rf1, Rf2`

Las operaciones elementales y microinstrucciones necesarias son las siguientes:

Microinstrucción	Operación elemental	Señales de control
$\mu 1$	$Rd \leftarrow Rf1 + Rf2$	$SelP = 11$, $SelA = 10100$, $SelB = 01111$, $SelC = 10111$, $M7$, $C7$, $T6$, SC, $A0$, B

Hay que indicar que esta operación es una operación que puede modificar algunos de los biestables del registro de estado, como pueden ser el que indica acarreo, desbordamiento, cero o negativo. Por tanto, es necesario activar la señal $C7$ para realizar la carga en el registro de estado de los biestables correspondientes, así como la señal $SelP$ con valor 11. En la única microinstrucción de esta instrucción

también se activan las señales $A0$ y B para seleccionar la entrada 11 del multiplexor A de la unidad de control. De esta forma la siguientes microinstrucción a ejecutar será la correspondiente al *fetch*. Durante esta microinstrucción no se activa la señal MC de la unidad de control. De esta forma el código de operación de la ALU se extrae de los últimos cuatro bits de la instrucción.

Si se desea que la instrucción anterior no incluya en la propia instrucción el código de operación, se puede codificar de acuerdo al formato de la Figura 2.20. En este caso la instrucción tendría como código de operación un valor distinto de 0 y el código de operación de la ALU no estaría en la propia instrucción, sino que tendría que almacenarse en la microinstrucción asociada a esta instrucción máquina, tal y como se muestra a continuación. En este caso el código de operación de la ALU se almacena en el campo $SelCop$ de la microinstrucción y se activa la señal MC para generar la señal Cop de la ALU a partir del valor codificado en $SelCop$.

Microinstrucción	Operación elemental	Señales de control
$\mu 1$	$Rd \leftarrow Rf1 + Rf2$	$SelCop = 1010$, $SelP = 11$, MC $SelA = 10100$, $SelB = 01111$, $SelC = 10111$, $M7$, $C7$, $T6$, SC, $A0$, B

Figura 2.20: Otro formato para la instrucción `add Rd, Rf1, Rf2`

Ejecución de instrucciones de salto. La instrucción del MIPS `j addr` modifica el contenido del contador de programa para pasar a ejecutar la instrucción almacenada en la posición indicada en *addr*. Las operaciones elementales necesarias son las siguientes:

Microinsrucción	Operación elemental	Señales de control
$\mu 1$	$PC \leftarrow IR(direccion)$	$Size = 11010$, $T3$, $C2$, $A0$, B

En esta microinstrucción se modifica el contenido del contador de programa con la dirección almacenada en el registro de instrucción. De esta forma, la siguiente instrucción a ejecutar será la almacenada en *addr*. En la operación elemental $PC \leftarrow IR(direccion)$ es necesario volcar al bus interno el contenido del registro de instrucción que incluye la dirección de salto. Para ello se utiliza el registro selector que se encuentra a la salida de dicho registro. Como la instrucción *j addr* emplea 6 bits para el código de operación, se va a considerar que en esta instrucción los 26 bits para el campo de la dirección se encuentran a continuación del código de operación (los bits $IR_{25} \ldots IR_0$). Las señales de control del circuito selector deberían tomar los siguiente valores: $SE = 0$ (no se activa), puesto que no se desea extender el signo de la dirección, $Size = 11010$ (el campo de la dirección ocupa 26 bits, es decir, $11010_{(2)}$). Por último la señal $Offset$ debería tomar valor 00000 (no se activa), para indicar que el campo a volcar en el bus interno es el que se encuentra a partir del bit IR_0.

A continuación se va a considerar la instrucción de salto condicional `beq R1, R2, desplazamiento`. Esta instrucción, al igual que su equivalente MIPS, utiliza direccionamiento relativo a contador de programa para bifurcar a la dirección $PC + desplazamiento$ en el caso de que el contenido del de los registros $R1$ y $R2$ sean iguales. El formato de esta instrucción se muestra en la Figura 2.21. Las operaciones elementales necesarias para la ejecución de esta instrucción son las siguientes:

Microinstrucción	Operación elemental	Señales de control
$\mu 1$	$RT2 \leftarrow SR$	$T8$, $C5$
$\mu 2$	$R1 - R2$	$SelA = 10101$, $SelB = 10000$, MC,
		$SelCop = 1011$, $SelP = 11$, $M7$, $C7$
$\mu 3$	$if(Z == 0) salto a \mu 9$	B, $C = 0110$, $\mu Addr = \mu 8$
$\mu 4$	$SR \leftarrow RT2$	$T5$, $C7$
$\mu 5$	$RT2 \leftarrow IR(desp)$	SE, $Size = 10000$, $T3$, $C5$
$\mu 6$	$RT1 \leftarrow PC$	$T2$, $C4$
$\mu 7$	$PC \leftarrow RT1 + RT2$	MA, MB, MC, $SelCop = 1010$, $T6$, $C2$, $A0$, B
$\mu 9$	$SR \leftarrow RT2$	$T5$, $C7$, $A0$, B

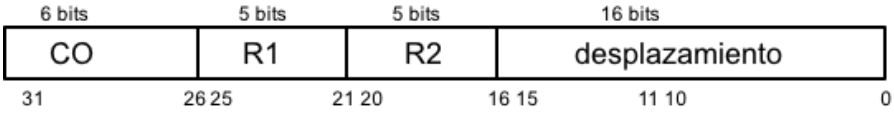


Figura 2.21: Formato de la instrucción `beq R1, R2, desplazamiento`

La ejecución de esta instrucción se basa en restar el contenido de los registros $R1$ y $R2$ y comprobar si el resultado es 0. Esto se puede conseguir actualizando el registro de estado y comprobando posteriormente el valor del biestable Z. Como esta instrucción no debe, sin embargo, modificar el registro de estado, lo primero que hace esta instrucción es guardar el valor que tiene el registro SR en el registro temporal $RT2$, para posteriormente restaurarlo. Esto se hace en la microinstrucción 1. En la segunda microinstrucción realiza la operación $R1 - R2$, y se debe activar durante este ciclo la señal $M7$ y $C7$ para realizar la carga en el registro de estado del resultado correspondiente al biestable Z, que indica si el resultado de la última operación fue 0. El resultado de esta operación solo es 0 cuando el contenido del registro $R1$ y $R2$ es el mismo. En la microinstrucción 3 se comprueba el valor del biestable Z, para ello se activa $C = 0110$ que permite sacar por el multiplexor C de la unidad de control el valor de este bit. En esta microinstrucción se activa B para que la entrada al multiplexor B sea el valor negado del bit Z. En caso de que Z esté activado, la entrada que se seleccionará en el multiplexor A será la 00, es decir, la siguiente. En caso contrario se seleccionará en el multiplexor A la entrada 01 y se bifurcará a la microinstrucción $\mu 8$. En este caso la microinstrucción 3 tiene que tener codificado la microdirección correspondiente a la microinstrucción 8, que se solapa con los campos $SelA$, $SelB$ y $SelC$. Como se verá en el siguiente capítulo, esta microdirección se obtiene directamente a partir del microcódigo. Cuando los dos registros son distintos se salta de la microinstrucción 3 a la 8. En esta microinstrucción se restaura el contenido del registro SR y se salta a *fetch* sin haber modificado el contador de programa.

En caso de que los dos registros sean iguales, en las siguientes microinstrucciones se realiza la operación $PC \leftarrow PC + desplazamiento$.

2.4. Modos de ejecución

La mayoría de los computadores presentan normalmente dos modos de ejecución: modo usuario y modo núcleo. En **modo usuario**, el procesador sólo puede ejecutar un subconjunto de todas las instrucciones máquina, estando prohibidas para su ejecución el resto, que se denominan *instrucciones privilegiadas*. Entre estas instrucciones privilegiadas se encuentran las instrucciones de E/S que permiten acceder a los periféricos, las instrucciones de habilitación e inhabilitación de interrupciones. Los

programas de usuario se ejecutan en este modo. El **modo núcleo**, que está reservado a la ejecución del sistema operativo, permite la ejecución de todas las instrucciones del computador.

El modo de ejecución en el que se encuentra el procesador WepSIM se indica mediante un bit situado en el registro de estado (este bit se corresponde con el bit SR_0 del registro de estado). Cuando el sistema operativo cede el control a un programa, se debe encargar de que este bit indique modo usuario ($U = 0$). Siempre que ejecuta el sistema operativo, cosa que ocurre, como se verá en la siguiente sección, cuando se genera una interrupción, el procesador eleva su modo de ejecución a modo núcleo ($U = 1$) permitiendo, de esta forma, que el sistema operativo pueda ejecutar cualquier instrucción máquina. El objetivo de disponer de al menos dos modos de ejecución en el procesador es la protección en el acceso a ciertos recursos del sistema.

2.5. Excepciones e interrupciones

La unidad de control se encarga de ejecutar instrucciones de forma secuencial. Esta secuencia solo se rompe con las instrucciones de bifurcación. Sin embargo, la unidad de control necesita otro mecanismo que permita romper este ciclo secuencial de ejecución de instrucciones. A este tipo de ruptura se le denomina interrupción. En el caso de WepSIM se distinguen:

- **Excepciones**. Ocurren como consecuencia de un error detectado en la ejecución de la instrucción en curso. Por ejemplo:
  - Ejecución de una instrucción no definida.
  - División por cero.
  - Desbordamiento aritmético.
  - Ejecución de una instrucción privilegiada en modo usuario.
  - Acceso ilegal a memoria.
- **Llamadas al sistema**. Ejecución de una instrucción que provoca la ejecución de una rutina en modo núcleo. Mecanismo típico utilizado para solicitar servicios al sistema operativo.
- **Interrupciones**, que ocurren por eventos externos típicamente producidos por dispositivos de E/S. En el caso de WepSIM existe un módulo de E/S genérico, que se puede configurar (véase el Capítulo 3) para generar interrupciones.

Una interrupción generada por el dispositivo de E/S se solicita activando un señal INT que llega a la unidad de control. El tratamiento de las interrupciones por parte de la unidad de control obliga a modificar el ciclo de ejecución de instrucciones añadiendo una fase más denominada **ciclo de reconocimiento de la interrupción**. La Figura 2.22 muestra el ciclo completo de ejecución de instrucciones.

El ciclo de reconocimiento de la interrupción se ejecuta siempre que las interrupciones estén habilitadas (lo que ocurre cuando en el registro de estado el bit I está activado) cuando termina la ejecución de la instrucción máquina en curso. El objetivo de este ciclo es comprobar si se ha solicitado una interrupción y en caso afirmativo suspender la ejecución del programa en curso para pasar a ejecutar otro programa diferente que se encarga de atender y tratar la interrupción. Este programa se denomina **rutina de tratamiento de la interrupción** y forma parte del código del sistema operativo de WepSIM. Cuando la rutina de tratamiento de la interrupción finaliza se reanuda la ejecución del programa suspendido.

Para obtener la dirección de la rutina de tratamiento de la interrupción, WepSIM utiliza **interrupciones vectorizadas** (véase la Figura 2.23). Con este mecanismo existe una tabla de interrupciones

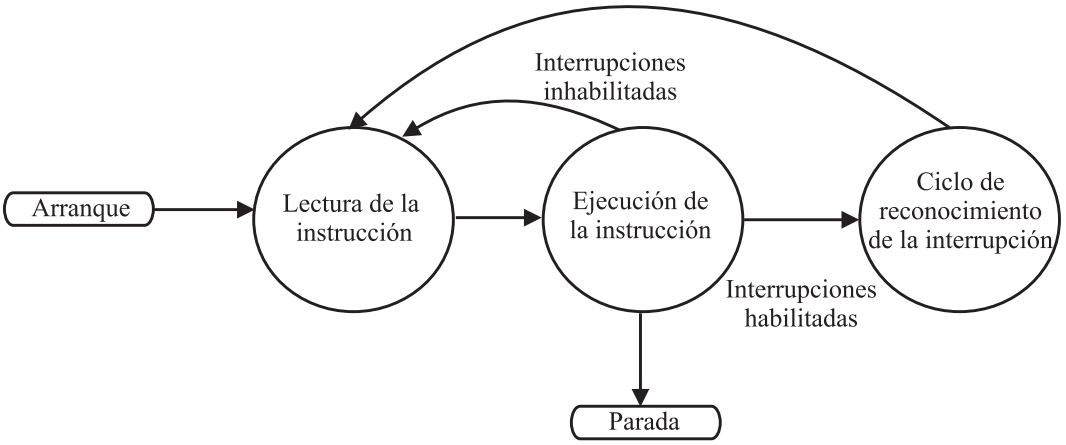


Figura 2.22: Ciclo de instrucción con tratamiento de interrupciones

almacenada en una zona de la memoria (en una dirección prefijada) asignada al sistema operativo. En el caso de WepSIM es la dirección 0 y su tamaño es de 256 palabras (de la 0 a la 255). Cada índice de la tabla representa un tipo de interrupción y el contenido de cada elemento de la tabla almacena la dirección de la rutina de tratamiento de la interrupción. El elemento que interrumpe, en este caso el módulo de E/S, suministra el **vector de interrupción**, el índice de la tabla que contiene la dirección de la rutina. La unidad de control lee el contenido almacenado en la entrada correspondiente de la tabla de interrupciones y carga ese valor en el contador de programa. Como resultado de esta carga, la siguiente instrucción máquina a ejecutar será la primera instrucción de la rutina de tratamiento. Es responsabilidad de cada sistema operativo rellenar esta tabla con las direcciones de cada una de las rutinas de tratamiento. El contenido, por tanto, de esta tabla es dependiente de cada sistema operativo.

A continuación se van a mostrar las operaciones elementales que tiene que llevar a cabo la unidad de control durante el ciclo de reconocimiento de la interrupción cuando se ha producido una interrupción. Para ello:

1. La unidad de control salva en la pila el contenido del PC y del registro SR.

2. Se eleva el modo de ejecución a modo núcleo, para lo cual la unidad de control debe activar la señal de control U, $C7$, $M7$ y $SelP$ con el valor 01.

3. Se obtiene el vector de interrupción. Durante el ciclo de reconocimiento de la interrupción, el módulo de E/S coloca en el bus de datos dicho vector. Se va a asumir que este vector se obtiene a través del registro MBR. Como este vector es un índice en la tabla de vectores de interrupción y cada entrada es de 32 bits, para obtener la dirección de la entrada adecuada es necesario sumar a la dirección de inicio de la tabla el desplazamiento que se obtiene de multiplicar este vector por 4.

En el caso de la excepciones, es la propia instrucción, en la que se detecta la excepción, la que se encarga de salvar el contenido del contador del contador de programa y del registro de estado, cambiar a modo núcleo, obtener la dirección de la rutina de tratamiento de la excepción y almacenar en el PC dicho valor. El vector de interrupción en el caso de las excepciones de obtiene del campo `ExCode` de la microinstrucción y que tiene, como se puede ver en la Figura 2.15, salida al bus interno del procesador. Al igual que en el caso de las interrupciones, para obtener la dirección de la rutina de tratamiento de la excepción es necesario sumar a la dirección de inicio de la tabla el desplazamiento que se obtiene de multiplicar el código almacenado en `ExCode` por 4. Se puede ver este código, por

tanto, como el vector de interrupción asociado a la excepción. Por ejemplo, si se considera que el vector de interrupción asociado a un desbordamiento aritmético es el 1, toda instrucción máquina que pueda provocar una excepción de este tipo, deberá almacenar el PC y el SR en la pila, cambiar a modo núcleo, leer la dirección de la rutina de tratamiento de la excepción que se encuentra en la posición de memoria 1×4 y cargar en el PC dicho valor. El código 1 habrá de almacenarse en el campo `ExCode` de la microinstrucción adecuada. En el caso de las llamadas al sistema, es la propia instrucción máquina que genera la llamada al sistema la que tiene que hacer los mismos pasos.

La tabla de vectores de interrupción de WepSIM se encuentra en la dirección 0 y dispone de 256 entradas:

- Los vectores 0 a 8 se corresponden con excepciones hardware generadas por la unidad de control (división por cero, ejecución de instrucción ilegal, etc) y por llamadas al sistema. El vector de interrupción lo vuelca al bus interno la unidad de control a través del triestado T11 (véase las figuras 2.1 y 2.15).
- Las entradas 9 a 255 se corresponden con las interrupciones externas generadas por el módulo de E/S de la Figura 2.2. El vector de interrupción se obtiene del propio dispositivo de E/S.

Los vectores de interrupción asignados a las excepciones y llamadas al sistema pueden ajustarse a una especificación concreta que se haga de WepSIM. Es decir, a una llamada al sistema se le podría asignar el vector 0 o el 4, dependerá del microcódigo cargado.

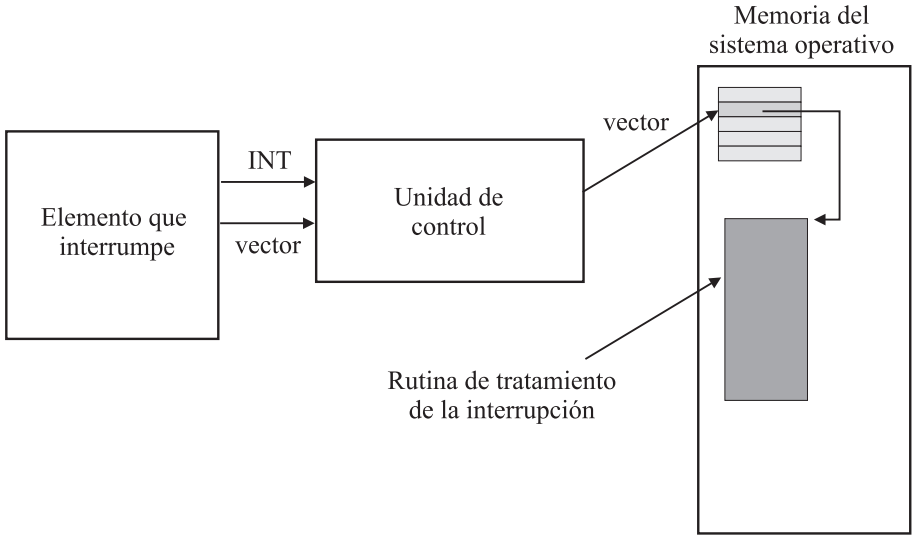


Figura 2.23: Interrupciones vectorizadas

La instrucción máquina de retorno de interrupción $RETI$ es la última instrucción que debe ejecutar la rutina de tratamiento de la interrupción y su objetivo es reanudar la ejecución del programa suspendido. Esta instrucción máquina realiza la función inversa a la del ciclo de aceptación de interrupción. Restituye de la pila, el contenido del registro de estado y del contador de programa. De esta forma se consigue que la siguiente instrucción a ejecutar sea la del programa suspendido con anterioridad. La restauración del registro de estado implica además que el procesador pase a ejecutar en modo usuario y con el registro SR en el mismo estado en el que se encontraba justo antes de tratar la interrupción.

El módulo de E/S de WepSIM dispone además de tres registros:

- El registro de estado $IOSR$ (dirección $0x1100$).

- El registro de control $IOCR$ (dirección $0x1104$).
- El registro de datos $IODR$ (dirección $0x1108$).

2.6. Teclado y pantalla en WepSIM

El teclado y la pantalla en WepSIM están controlados por módulos de E/S que utilizan E/S programada. El teclado dispone de dos registros:

- El registro $KDBR$ (dirección $0x0100$), es el registro de datos, que almacena el código ASCII de la tecla pulsada.
- El registro $KBSR$ (dirección = $0x0104$), es el registro de estado. Se pone a 1 cuando se pulsa una tecla. Cuando se lee el código del registro de datos, $KBSR$ se pone automáticamente a 0.

La pantalla en WepSIM está controlada por un módulo de E/S que dispone de dos registros:

- El registro DDR (dirección $0x1000$), es el registro de datos, almacena el código ASCII a mostrar.
- El registro DSR (dirección = $0x1004$), es el registro de estado. Se pone a 1 cuando el módulo finaliza el volcado en la pantalla. Se pone a 0 cuando se almacena un valor en DDR.

WepSIM utiliza un mapa de E/S separado y los programas de usuario necesitan señales de E/S para poder acceder a estos dispositivos. A continuación se muestran las microinstrucciones asociadas a las instrucciones máquina `in reg, IOaddr` y `out reg, IOaddr`. Se asume que el código de operación se almacena en los primeros 6 bits, el identificador del registro en los 5 siguientes y la dirección de E/S en los 16 inferiores. La primera almacena en el registro *reg* del procesador el dato almacenado en el registro de E/S *IOaddr*. La segunda almacena en el registro de E/S *IOaddr* el valor del registro *reg*.

Las microinstrucciones necesarias para la ejecución de la instrucción `in` son las siguientes:

Microinstrucción	Operación elemental	Señales de control
$\mu 1$	$MAR \leftarrow IR(IOaddr)$	$Size = 10000$, $T3$, $C0$
$\mu 2$	$MBR \leftarrow IOaddr$	Ta, IOR, $M1$, $C1$
$\mu 2$	$reg \leftarrow MBR$	$T1$, LC, $SelC = 10101$, $A0$, B

Para poder leer de un registro de E/S es necesario colocar en el bus de direcciones la dirección del registro de E/S y activar la señal IOR. Para poder escribir en un registro de E/S es necesario colocar en el bus de direcciones la dirección del registro de E/S, en el bus de datos el dato a escribir y activar la señal IOW. Las microinstrucciones para la instrucción `out` son las siguientes:

Microinstrucción	Operación elemental	Señales de control
$\mu 1$	$MAR \leftarrow IR(IOaddr)$	$Size = 10000$, $T3$, $C0$
$\mu 2$	$MBR \leftarrow reg$	$SelA = 10101$, $T9$, $C1$
$\mu 2$	$IOAddr \leftarrow MBR$	Ta, Td, IOW, AO, B

Con estas dos instrucciones se pueden escribir rutinas que permiten acceder al teclado y al monitor. A continuación se muestra una rutina que se encarga de leer un código ASCII del teclado y escribirlo en pantalla:

```
lstart: IN    R1, KBSR
        beqz  R1, lstart
        IN    R1, KBDR
        OUT   R1, DDR
echo:   IN    R1, DSR
        beqz  R1, echo
```

2.6.1. Operaciones elementales de la instrucción de llamada al sistema

En esta sección se van a indicar todas las operaciones elementales necesarias para invocar una llamada al sistema. Se va a considerar que la instrucción máquina que permite invocar una llamada al sistema es `syscall`. Como se indicó anteriormente, esta llamada debe encargarse de:

1. Salva en la pila el contenido del PC y del registro SR.
2. Elevar el modo de ejecución a modo núcleo, para lo cual la unidad de control debe activar la señal de control U, $C7$, $M7$ y $SelP$ con el valor 01.
3. Obtener vector de interrupción. Se va a considerar que el vector de interrupción correspondiente a las llamadas al sistema es el 8. Esto quiere decir que la dirección de la rutina que trata las llamadas al sistema es la $8 \times 4 = 32$. El número 8 se almacenará en el campo `ExCode`.

A continuación se muestran las microinstrucciones necesarias para la ejecución de la instrucción `syscall`. Se va a considerar que el registro 29 ($11101_{(2)}$) es el puntero de pila (SP) y que este registro almacena la dirección de memoria donde se encuentra la cima de la pila.

Microinstrucción	Operación elemental	Señales de control
$\mu 1$	$SP \leftarrow SP - 4$	$SelA = 11101$, MR, $MB = 10$, $Cop = 1011$, $T6$
		$SelC = 11101$, LC
$\mu 2$	$MAR \leftarrow SP$	$SelA = 11101$, MR, $T9$, $C0$
$\mu 3$	$MBR \leftarrow PC$	$T2$, $C1$
$\mu 4$	$Memory \leftarrow MBR$	Ta, Td, W, $BW = 11$
$\mu 5$	$SP \leftarrow SP - 4$	$SelA = 11101$, MR, $MB = 10$, $Cop = 1011$, $T6$
		$SelC = 11101$, LC
$\mu 6$	$MAR \leftarrow SP$	$SelA = 11101$, MR, $T9$, $C0$
$\mu 7$	$MBR \leftarrow SR$	$T8$, $C1$
$\mu 8$	$Memory \leftarrow MBR$	Ta, Td, W, $BW = 11$
$\mu 9$	$ModoNcleo$	I, $SelP = 10$, $M7$, $C7$
	$RT1 \leftarrow ExCode$	$T11$, $C4$
$\mu 10$	$MAR \leftarrow RT1 \times 4$	$Cop = 1100$, MA, $MB = 10$, $T6$, $C0$
$\mu 11$	$MBR \leftarrow Memory$	Ta, Td, $BW = 11$, R
$\mu 12$	$PC \leftarrow MBR$	$T1$, $C2$

Durante la microinstrucción 9 se vuelca el código de excepción al bus interno del procesador. A continuación se obtiene la dirección de donde se almacena la dirección de la rutina de tratamiento de la llamada al sistema, que se obtiene en la microinstrucción 10, se lee la dirección de la rutina de tratamiento de la excepción y se almacena en el PC en la última microinstrucción. La siguiente instrucción a ejecutar se corresponderá con la primera instrucción máquina de la rutina de tratamiento de la llamada al sistema, que estará ejecutando durante su ejecución en modo núcleo. Para retornar al programa interrumpido, la rutina de tratamiento debe finalizar su ejecución con una llamada máquina

especial, que permite retornar del tratamiento de interrupción o excepción. Esta instrucción máquina especial, debe encargarse de restaurar el contenido del contador del programa y del registro de estado del programa interrumpido y que se encuentran en la cima de la pila. Con esto se consigue pasar de nuevo a modo usuario y continuar la ejecución en la siguiente instrucción máquina del programa interrumpido. Esta instrucción en el MIPS es `eret`

2.7. Arranque de WepSIM

Cuando se inicia el simulador WepSIM, este toma como datos de entrada el micrócodigo que define las instrucciones máquina que va a ejecutar el procesador WepSIM y el programa en ensamblador a ejecutar. Cuando se pulsa en el simulador el botón de reset, se realizan las siguientes acciones:

- Se ponen a cero todos los registros del banco de registros y los registros $RT1$, $RT2$ y $RT3$.
- Se almacena en el contador de programa el valor $0x0100$ correspondiente a la dirección donde se almacena la primera instrucción del programa ensamblador cargado (en la dirección `kmain`) y que debe corresponder al segmento del núcleo (véase la sección 2.8.2). En caso de no existir segmento para el núcleo se almacena en el contador de programa la dirección $0x8000$ correspondiente a la función `main` del espacio de usuario.
- Se fija el puntero de pila, que haya establecido el usuario, al valor $0xFFFF$. En la versión actual solo hay un segmento de pila, a compartir entre el el modo kernel y usuario. En futuras versiones, cada modo tendrá su segmento de pila.

En versiones futuras del simulador, el procesador arrancará en modo kernel ($U = 1$) y con las interrupciones inhabilitadas ($I = 0$). Es decir, se almacena en el registro SR el valor 1.

Una vez comenzada la ejecución del código inicial del sistema operativo, si existe, éste se debe encargar, al menos, de:

- Almacenar en la tabla de vectores de interrupción las direcciones de las rutinas de tratamiento de excepción, interrupción y llamadas al sistema.
- Fijar el puntero de pila a la dirección correspondiente.
- Habilitar las interrupciones.
- Ceder el control al programa almacenado en la dirección `main`.

2.8. Programación en ensamblador en WepSIM

El procesador WepSIM permite ejecutar un amplio conjunto de instrucciones en ensamblador, basta con incluir el microcódigo en la memoria de control. En el próximo capítulo se darán algunos detalles sobre la forma de hacerlo. Tres son los aspectos que se mantienen constantes con independencia del juego de instrucciones a ejecutar: el espacio de memoria de los programas, las directivas del ensamblador y las llamadas al sistema. Ambas se inspiran en el ensamblador del MIPS.

WepSIM permite asignar códigos simbólicos a los 32 registros del procesador y definir qué registro se utilizará como puntero de pila. En futuras versiones, se permitirá fijar el valor por defecto inicial e indicar si este valor es de solo lectura o no.

2.8.1. Sintaxis

Un programa en ensamblador estará formado por una secuencia de instrucciones y directivas del ensamblador que puede contener comentarios. A continuación se describen las principales reglas léxicas.

Comentarios. El carácter `#` indica el inicio de un comentario, que acaba con el final de la línea.

Identificadores. Un identificador es una secuencia de caracteres alfanuméricos (incluyendo los caracteres `.`, `_` y `$`). Se consideran distintas las letras mayúsculas de las minúsculas. Los identificadores se utilizan para dar nombre a las etiquetas.

Constantes decimales. Una constante decimal es una secuencia de dígitos decimales que no comienza por cero. Pueden incluir signo.

Constantes hexadecimales. Una constante hexadecimal es una secuencia de dígitos hexadecimales (donde las letras pueden ir en mayúsculas o minúsculas) precedidos de los caracteres `0x` o `0X`. Pueden incluir signo.

Constantes octales. Una constante octal es una secuencia de dígitos octales, precedidas del dígito `0`. Pueden incluir signo.

Constantes de carácter. Una constante de tipo carácter es cualquier carácter delimitado por comillas simples (por ejemplo `'x'`. Se pueden utilizar las secuencias de escape especificadas en la Tabla 2.3.

Constantes de cadena. Es una cadena de caracteres delimitada por comillas dobles (por ejemplo `"hola"`). Se puede utilizar las secuencias de escape especificadas en la Tabla 2.3.

Cuadro 2.3: Secuencias de escape para caracteres y cadenas de caracteres

Secuencia	Código ASCII	Significado
`\a`	`0x07`	Alerta (*alert*).
`\b`	`0x08`	Retroceso (*backspace*).
`\f`	`0x0c`	Avance de página (*form feed*).
`\n`	`0x0a`	Salto de línea.
`\r`	`0x0d`	Retorno de carro.
`\t`	`0x09`	Tabulación horizontal.
`\v`	`0x0b`	Tabulación vertical.
`\\`	`0x5c`	Carácter \.
`\"`	`0x22`	Comilla doble.
`\'`	`0x27`	Comilla simple.

2.8.2. Directivas del ensamblador

Las directivas del ensamblador son indicaciones al lenguaje ensamblador que permiten, entre otras cosas, establecer la estructura de las instrucciones y datos de un programa.

Un primer grupo de las directivas del ensamblador lo constituyen, aquellas que permiten indicar el comienzo de una sección de código (`.text` y `.ktext`) o de datos (`.data`, `.kdata`). Cada sección comienza con una de las mencionadas directivas y acaba con el comienzo de otra directiva.

A continuación se muestra una estructura típica de un programa en ensamblador.

```
.data
        #declaraciones de datos globales
.text
        #instrucciones de programa de usuario
```

Dentro de cualquier sección de datos se pueden asociar etiquetas a directivas de definición de datos (`.ascii`, `.asciiz`, `.byte`, `.half`, `.space` y `.word`). En el siguiente fragmento se muestran algunas declaraciones de variables globales en una sección `.data`.

```
.data:
    cadena: .asciiz "Hola mundo\n"
    i1:      .word 10   # int i1=10
    i2:      .word -5   # int i2=-5
    c1:      .byte 100  # char c1=100
    c2:      .byte 'a'  # char c2='a'
    v:      .word 0, -1, 0xffffffff      # int v[3] = { 0, -1, 0xffffffff}
.text
 #instrucciones de programa de usuario
```

La constante `cadena` representa la dirección de memoria donde se almacena una cadena de caracteres. Una cadena de caracteres se representa como una secuencia de caracteres (códigos ASCII) que finaliza con el valor 0 (directiva `.asciiz`).

El simulador WepSIM soporta una buena parte de las directivas de ensamblador MIPS32. A continuación se describen las directivas soportadas por WepSIM:

`.align n` Avanza el contador de posición, que se usa para asignar la dirección del siguiente dato, de forma que los `n` bits menos significativos del contador sean cero. Esto permite forzar un alineamiento a una dirección múltiplo de 2^n. Las directivas `.half`, `.word` alinean sus datos de forma automática, por ejemplo `.word` hace un `.align 2` implícito). Tenga en cuenta que poner una etiqueta delante de una directiva `.align` equivale a ponerla después de la misma.

`.ascii ''cadena''` Almacena la cadena en posiciones de memoria sucesivas. Esta directiva no añade ningún *byte* de terminación al final de la cadena. Se pueden indicar caracteres especiales mediante el carácter \. Se pueden indicar varias cadenas separadas por comas. En este caso se asignan posiciones de memoria consecutivas a las distintas cadenas. Es decir, el primer carácter de la segunda cadena se almacena en la posición de memoria siguiente al último carácter de la primera cadena.

`.asciiz ''cadena''` Se comporta igual que `.ascii`, pero coloca un carácter terminador de cadena (carácter con código ASCII 0) al final de cada cadena.

`.byte:` define un byte, permitiendo el formato en octal, hexadecimal, decimal y carácter. Admite uno o varios valores separados por coma.

`.half:` define media palabra (2 bytes), permitiendo el formato octal, hexadecimal y decimal. Admite uno o varios valores separados por coma.

`.word:` define una palabra (4 bytes), permitiendo el formato octal, hexadecimal y decimal. Admite uno o varios valores separados por coma.

`.space:` define una reserva de espacio en memoria del número de bytes indicado en formato decimal.

`.data` Marca el comienzo del segmento de datos del programa.

`.kdata` Indica que los datos declarados a continuación deben almacenarse en el segmento de datos del núcleo (*kernel*).

`.ktext` Marca el comienzo del segmento de código del núcleo.

`.text` Marca el comienzo del segmento de código del programa.

2.8.3. Espacio de memoria de un programa

WepSIM permite cargar y ejecutar un programa escrito en el ensamblador definido en el microcódigo. Un programa en ensamblador debe incluir:

- **Segmento de texto del núcleo**. Comienza en la etiqueta `.kmain` y se corresponde con la dirección de memoria $0x1000$ que se almacena en el PC en el reset del procesador WepSIM. Este segmento se debe encargar de rellenar la tabla de vectores de interrupción con las direcciones de las rutinas de tratamiento de excepciones, interrupciones y llamadas al sistemas. Debe habilitar las interrupciones, y ceder el control al programa de usuario que reside en la etiqueta `.main`. Este segmento debe incluir las rutinas de tratamiento de excepciones, interrupciones y llamadas al sistema de las que se quiera disponer. Puede o no definirse, es opcional.
- **Segmento de datos del núcleo** Se corresponde la con la etiqueta `.kdata` y en él se almacenan cualquier dato que desee el sistema operativo que se quiera cargar en WepSIM.
- **Segmento de texto del programa de usuario**. Comienza en la etiqueta `.main`.
- **Segmento de datos del programa de usuario**. Comienza en la etiqueta `.data`.

En ejecución un programa incluye además un **segmento de pila**. El valor de inicio para este segmento se fija en $0xFFFF$. Aunque el código definido en el núcleo podría definir el uso de una pila para el núcleo y otra para el programa de usuario, el esquema más sencillo es disponer de una única pila compartida por el núcleo y el programa de usuario.

En la figura 2.24 se muestra el mapa de memoria por defecto usado en WepSIM.

2.8.4. Llamadas al sistema

Las llamadas al sistema disponibles para los programas escritos en ensamblador pueden configurarse libremente. Una posibilidad, por ejemplo, es utilizar el convenio de llamadas al sistema disponibles en el simulador SPIM. Lo único que debe hacerse es fijar un determinada convenio y programar en el segmento de texto del núcleo el código de estas rutinas. Su invocación debe hacerse con una instrucción máquina especial (trap, syscall, etc.) que debe estar previamente definida en el microcódigo.

2.8.5. Ejemplo de programa

A continuación se va a mostrar un posible ejemplo de programa en ensamblador que incluye la definición del segmento de texto y datos del kernel y del segmento de texto y datos del programa de usuario. También se incluye, como ejemplo, una única rutina de tratamiento de interrupciones y excepciones, en este caso la llamada al sistema y se asume que el vector de interrupción asociado a la rutina de tratamiento de la llamada al sistema es el 0. Se considera que el el juego de instrucciones incluye las principales instrucciones del MIPS, el nombrado de registros del MIPS y además las siguientes instrucciones máquina, cuya definición debería incluirse en el microcódigo como parte de su definición:

- `EI`, instrucción máquina que habilita las interrupciones.

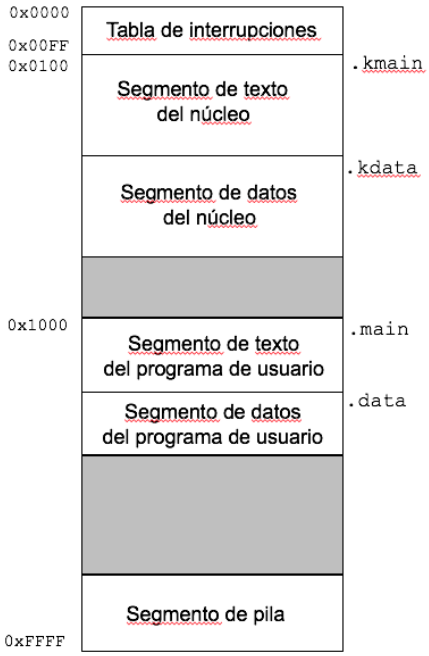


Figura 2.24: Mapa de memoria por defecto usado en WepSIM

- `DI`, instrucción máquina que inhibe las interrupciones.
- `UM`, instrucción máquina que pasa a modo usuario.
- `IN R, IOR`, instrucción de E/S que lee el contenido de un registro de E/S en un registro (R) del procesador.
- `OUT R, IOR`, instrucción de E/S que escribe el contenido de un registro del procesador en un registro de E/S (IOR).
- `HALT`, instrucción máquina que para la ejecución del procesador.

Todas estas instrucciones son instrucciones que deberían ejecutarse solo en modo núcleo, por tanto, en un escenario más real, en caso de ejecutarse una instrucción de las anteriores en modo usuario se debería generar una excepción e invocar a la rutina correspondiente, que se encargaría de abortar la ejecución del programa.

Las llamadas al sistema se van a invocar con `syscall` y se van a considerar solo tres posibilidades:

- Lectura de un carácter. Código de operación $v0 = 0$. El carácter leído se almacena en $v0$.
- Escritura de un carácter en el display. Código de operación $v0 = 1$. El carácter a escribir se almacena en $a0$.
- Finalización de la ejecución (`exit`). Código de operación $v0 = 2$.

Se utiliza la instrucción máquina `eret` para retornar de una rutina de tratamiento de interrupción, llamada al sistema o excepción (también debería incluirse su definición en el microcódigo). Este programa comienza su ejecución en la instrucción que se encuentra en la etiqueta `.kmain`, puesto que está definida.

```
.kdata:


.ktext:

    kmain:
                # el procesador arranca en modo núcleo
                # se almacena en la entrada 0 de la tabla de interrupciones la
                # rutina de tratamiento de la llamada al sistema
                la      $t1, syscallRoutine
                sw      $t1, 0

                # se habilitan las interrupciones
                EI
                # se pasa a modo usuario
                UM

                # se salta a ejecutar el programa de usuario
                jal main

                HALT


    syscallRoutine:

                # se salva el contenido de $s1
                addi    $sp, $sp, -4
                sw      $s1, $sp
                beq     $v0, $0, read
                li      $s1, 1
                beq     $v0, $s1, write
                li      $s1, 2
                beq     $v0, $s1, exit
                b       end
    read:       in      $s1, KBSR
                beqz    $s1, read
                in      $v0, KBDR
                b       end
    write:      out     $a0, DDR
    noready:    in      $s1, DSR
                beqz    $s1, noready
                b       end
    end:        lw      $s1, ($sp)
                addi    $sp, $sp, 4
                eret                #retorno de llamada al sistema
    exit:       jr      $ra         #retorna a kmain


# segmento correspondiente al programa de usuario

.data
    c:  .char 'a'

text:
```

```
main:   #salva en la pila la dirección de retorno
        addi    $sp, $sp, -4
        sw      $ra, $sp

        li      $v0, 0      # lee un carácter
        syscall

        move    $a0, $v0
        li      $v0, 1      # escribe el carácter leído
        syscall


        #restaura de la pila la dirección de retorno
        lw      $ra, $sp
        addi    $sp, $sp, -4

        li      $v0, 2      #exit
        syscall
```

Capítulo 3

Manual de usuario

En esta sección se describe el manual de usuario del simulador WepSIM.

En WepSIM, una vez ensamblado un programa en ensamblador, se pueden realizar simulaciones del código visualizando la activación de las señales del procesador y el tránsito de datos a través de los buses y registros. De esta forma, WepSIM, ofrece 3 funcionalidades diferentes: la definición de un juego de instrucciones y su microcódigo, la creación de programas en ensamblador que utilizan dicho juego de instrucciones, y la depuración y simulación del microcódigo diseñado y de los programas en ensamblador creados.

La explicación de estas tres funcionalidades va a contemplar:

1. La definición de un determinado juego de instrucciones y su microcódigo.
2. La definición y carga de programas en ensamblador.
3. Los principales aspectos relacionados con el uso del simulador (ejecución paso a paso de instrucciones o microinstrucciones, puntos de ruptura, señales interactivas, etc.)

En esta sección no se incluye en la versión inicial de manual el manual de usuario, puesto que el propio simulador incluye un manual detallado en línea.

Bibliografía

Índice alfabético