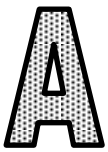
ANEXO



DISEÑO DE PROCESADORES RISC

1. LA ARQUITECTURA "RISC"

1.1. Procesadores RISC y CISC

Hasta finales de la década de los 70 se pretendía reducir el coste del "hardware", mediante el uso de complejos juegos de instrucciones basados en la microprogramación. En esta época el tiempo de acceso a Memoria Principal era muy superior al de decodificación y procesamiento de las instrucciones, debido al uso de los circuitos integrados. Para paliar ese desequilibrio se disminuyó el empleo de la Memoria Principal y se aumentó el del procesador, creando así los juegos de instrucciones complejos o **CISC**.

Otros aspectos ventajosos son: el abaratamiento del "hardware", la facilitación del diseño de los compiladores, y la disminución del tamaño de los programas.

Los avances tecnológicos han permitido disminuir el desequilibrio antes mencionado con nuevas memorias más rápidas y el uso de las memorias caché. Esto ha supuesto la pérdida de interés por la microprogramación y la aparición, por su parte, de las tecnologías **RISC** en la década de los 80. El principal objetivo de esta tecnología es disponer de instrucciones muy simples, con el mínimo número de microinstrucciones.

En la actualidad se admite la superioridad de la arquitectura **RISC**, respecto a la arquitectura **CISC**, si bien **CISC** todavía persiste.

1.2. Generaciones de los procesadores RISC

1.2.1. Primera generación

A ella pertenecen los tres primeros modelos de la primera mitad de la década de los 80:

- IBM 801.
- RISC (diseñado por Patterson).
- MIPS (diseñado por Hennessy).

1.2.2. Segunda generación

Nace en la segunda mitad de la década de los años 80, con los siguientes modelos representativos:

- SPARC.
- CLIPPER C100.
- HPPA.
- M88100.
- MIPS R2000.
- AMD 29000.

1.2.3. Tercera generación

Integrado por los siguientes modelos de la primera mitad de la década de los 90:

- i860.
- M88110.
- IBM RS6000.
- PA7100.
- C400.
- Super SPARC.
- ALPHA.

1.3. Metodología de diseño

El profesor Hennessy inició en 1981 el proyecto **MIPS**, que culminó en 1984 con el desarrollo de la primera arquitectura de la familia de procesadores **RISC**.

Basándonos en este modelo resumiremos los pasos a desarrollar como método de diseño de procesadores:

- Definición del Repertorio de **Instrucciones Máquina**. Modos de direccionamiento y formato.
- Especificaciones que debe cumplir la **ALU** y su diseño correspondiente.
- Diseño del **Camino de Datos** "**monociclo**". ALU, Banco de Registros, Memoria y componentes auxiliares.
- Diseño del **Camino de Datos** "**multiciclo**".
- Diseño de la **Unidad de Control**. Técnicas cableada y microprogramada.
- Diseño de un procesador "**segmentado**". La técnica de la segmentación.

1.4. Fórmulas para la evaluación del rendimiento de los procesadores

Las fórmulas y magnitudes más importantes que definen el rendimiento de un procesador se describen aquí:

- El **CPI** especifica los ciclos que tarda de promedio en ejecutarse cada instrucción del programa de prueba al que se hace referencia. Para calcular el **CPI** hay que tener en cuenta los ciclos de cada instrucción, así como el porcentaje de dicha instrucción que se ejecuta en el programa en cuestión.
- El **rendimiento** del procesador se evalúa como el inverso del tiempo que necesita para ejecutar el programa de referencia.
- Para calcular los **MIPS** nativos se hallan los millones de instrucciones que se pueden realizar en un segundo
- Los **MIPSVAX** se calculan teniendo en cuenta el tiempo que tarda en ejecutarse un programa en el **VAX 11/780** y en el procesador a evaluar.

2. REPERTORIO DE INSTRUCCIONES MÁQUINA

Nos referiremos a una versión reducida de los procesadores **MIPS** para el estudio del repertorio de instrucciones. La palabra de estado de este procesador es de 32 bits, como el código máquina de cada instrucción.

Frente a la arquitectura memoria-memoria, que necesita utilizar demasiado espacio de memoria, el **MIPS** responde a la arquitectura carga-almacenamiento, en los que únicamente los registros internos soportan los operandos de la instrucción y sus contenidos se cargan en memoria ("load") y se almacenan ("store").

A la hora de desarrollar las instrucciones únicamente hacemos referencia a los componentes destinados a números enteros.

El **MIPS** dispone de un Banco de 32 registros, cada uno de 32 bits de tamaño. La designación de los registros para números enteros es \$0, \$1,.....,\$31, tal como se representa.

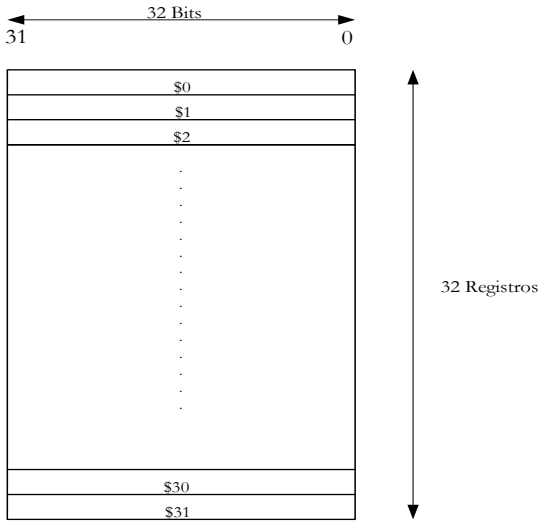


Figura 1—Representación del Banco de Registros para números enteros del MIPS.

2.1. Repertorio de instrucciones

Tenemos nueve instrucciones distribuidas de la siguiente forma:

2.1.1 Instrucciones aritméticas y lógicas

Tanto los dos operandos fuente, como destino de este grupo son registros. Explicamos la nomenclatura usada en Ensamblador y su funcionalidad.

- **add**

Realiza la suma binaria de dos registros dejando el resultado en el registro destino. En la nomenclatura se comienza designando el registro destino, seguido de los dos fuentes.

add $1, $2, $3 #$1 = $2 **+** $3

Una variante es el **addi**, que admite que uno de los opèrandos sea un valor inmediato.

addi $3, $5, 9 #$3 = $5 **+** 9

Las restantes instrucciones del grupo son similares a **add** y sólo varía la operación que realizan.

add $1, $2, $3 #$1 = $2 **+** $3
sub $1, $2, $3 #$1 = $2 **--** $3
and $1, $2, $3 #$1 = $2 **AND** $3
or $1, $2, $3 #$1 = $2 **OR** $3

- **slt**

Compara el contenido de los dos registros que actuan como operandos fuentes y pone un tercero a 1, si el primero es menor que el segundo, sino a 0.

slt $4, $6, $8 #$4 = 1 Si $6 < $8
#$4 = 0 Si $6 > $8

Esta tiene una variante **slti**, que compara el valor de un registro con un inmediato

slti $2, $5, 100 #$2 = 1 Si $5 < 100

2.1.2 Instrucciones de transferencia de datos con la memoria

Para realizar transferencias de datos entre la Memoria y los registros en ambas direcciones existen dos instrucciones.

- **lw** (load word)

Carga el contenido de una posición de Memoria, cuya dirección viene expresada en la instrucción por la suma del contenido de un registro y un valor inmediato de 16 bits, en un registro. Detrás del
nemónico se exprésale registro que se va a cargar y después la dirección de inicio en la memoria de un array y el índice del elemento del array en el que está depositado el dato a transferir.

lw $5, 5000($3) #$5 = M[5000 + ($3)]

Esta instrucción carga en el registro $5 el contenido de la posición de la Memoria expresada por la suma de una dirección que indica el comienzo de un array de datos y un índice, contenido en un registro, que se denomina "registro índice". En el ejemplo la dirección de inicio del array se expresa directamente mediante el valor 5000 y el registro índice $3.

- **sw** (store load)

Almacena la palabra contenida en un registro en una dirección de la Memoria expresada de la misma forma que la instrucción **lw**.

```
sw  $7, Array($1)                    #M[Array + ($1)] = $7
```

En este ejemplo la dirección de inicio del array de datos viene expresada mediante la etiqueta Array, que corresponde a un valor inmediato de 16 bits.

2.1.3 Instrucciones de salto condicional

- **beq**

Si el contenido de los dos registros fuente son iguales, se salta a la instrucción cuya dirección se halla sumando el valor del PC el inmediato de 16 bits que se incluye en la instrucción.

```
beq  $3, $5, 1000                    # Si $3 = 45, PC = PC+1
```

2.1.4 Instrucciones de bifurcación incondicional

En el núcleo básico de instrucciones del **MIPS** sólo se estudia en este grupo la instrucción **jump**, de mnemónico **j**. Esta instrucción bifurca incondicionalmente el programa a la dirección de destino que acompaña a la instrucción.

```
j  14600                             #PC = 14600
```

2.2 Modos de direccionamiento

El **MIPS** simplificado que se estudia dispone de cuatro modos de direccionamiento.

- **1º Direccionamiento por registro**

El operando es el contenido de un registro, como en la instrucción **add**.

- **2º Direccionamiento base o desplazamiento**

El operando reside en una posición de la memoria cuya dirección se obtiene sumando el contenido de un registro y un valor inmediato de 16 bits que incluye la instrucción **lw**

- **3º Direccionamiento inmediato**

El operando es una constante expresada en la propia instrucción mediante un valor de 16 bits, esto sucede con la instrucción **addi**

- **4º Direccionamiento relativo al PC**

El PC se carga con un valor resultante del valor que tenía con uno inmediato de 16 bits que acompaña a la instrucción. Este direccionamiento se usa en la instrucción **beq**.

2.3 Formato de las Instrucciones

Todas las instrucciones del **MIPS** tienen una longitud de 32 bits y su formato responde a tres modelos que reciben los nombres de **formato R**, **formato I**, y **formato J**.

2.3.1 Formato de las instrucciones lógicas y aritméticas

Este grupo de instrucciones responden al **formato R**, en el cual la longitud de 32 bits de la instrucción se descompone en 6 campos:

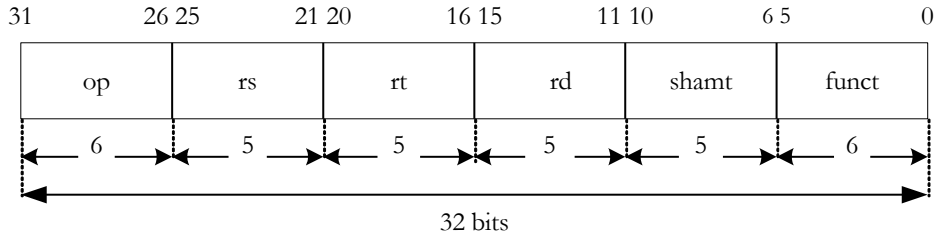


Figura 2 – Formato R, correspondiente a las instrucciones lógicas y aritméticas.

El campo **op** sirve para discriminar la instrucción. El campo **rs** expresa en binario el registro que contiene un operando fuente. El campo **rt** expresa el registro que contiene el segundo operando fuente. El campo **rd** indica el registro que actúa como destino. El campo **shamt** expresa un desplazamiento cuya misión se comentará más adelante. Y el campo **funct** sirve para discriminar las instrucciones que tienen el mismo campo **op**.

2.3.2 Formato de las instrucciones de transferencia

Responden al **formato I**, que descomponen los 32 bits de la instrucción en los campos que se muestran a continuación:

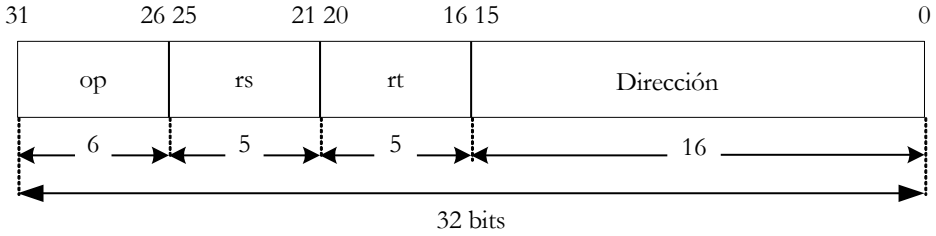


Figura 3—Campos en los que se descomponen las instrucciones de formato I

op: Código de la operación de la instrucción.

- **rs**: registro que contiene el Índice que apunta un elemento del "array" de datos de la Memoria.
- **rt**: registro que recibe el resultado de la operación de transferencia. En **lw** se le carga con el contenido de la dirección de la Memoria y en **sw** su valor se almacena en la dirección de la memoria.
- **dirección**: es un valor inmediato que representa la dirección de comienzo del "array" de datos.

Las instrucciones de direccionamiento inmediato, como **addi** y **slti**, también responden al **formato I**.

2.3.3 Formato de las instrucciones de salto condicional

La instrucción **beq** responde al **formato I**.

2.3.4 Formato de las instrucciones de bifurcación incondicional

La instrucción de bifurcación incondicional tiene el **formato J**, caracterizado por dividir la longitud de la instrucción en sólo dos campos, el del código **op** y el de **dirección**.

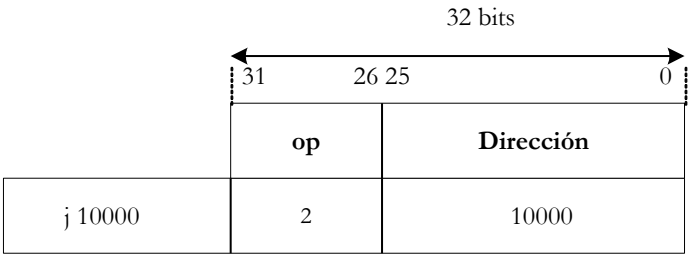


Figura 4—Formato J al que responde la instrucción de bifurcación incondicional.

La **Unidad de Control** cuando reciba el código **op** de cada instrucción determinará a qué formato corresponde y los campos en los que se descompone.

3. DISEÑO DE LA ALU

Las instrucciones lógicas y aritméticas son las principales responsables de las funciones que debe desarrollar la **ALU**, que son las siguientes:

- Suma binaria (**add**).
- Resta lógica (**sub**).
- AND lógica (**and**).
- OR lógica (**or**).
- Poner a 1 ó a 0 (**slt**).

Otras instrucciones también requieren realizar algunas operaciones lógicas o aritméticas. Así, por ejemplo, las instrucciones **lw** y **sw** precisan sumar el contenido de un registro a un inmediato de 16 bits extendido en signo y convertido en un valor de 32 bits.

En este apartado nos ocuparemos esencialmente de la **ALU** del procesador principal de **MIPS** que opera con números enteros.

3.1 Una ALU básica

La **ALU** del **MIPS** funciona con operandos de 32 bits, pero su implementación se consigue mediante el conectado en paralelo de 32 **ALU** elementales de 1 bit. En el siguiente dibujo se muestra una **ALU** básica de 1 bit que es capaz de efectuar las tres operaciones básicas: **suma binaria**, **and** y **or**.

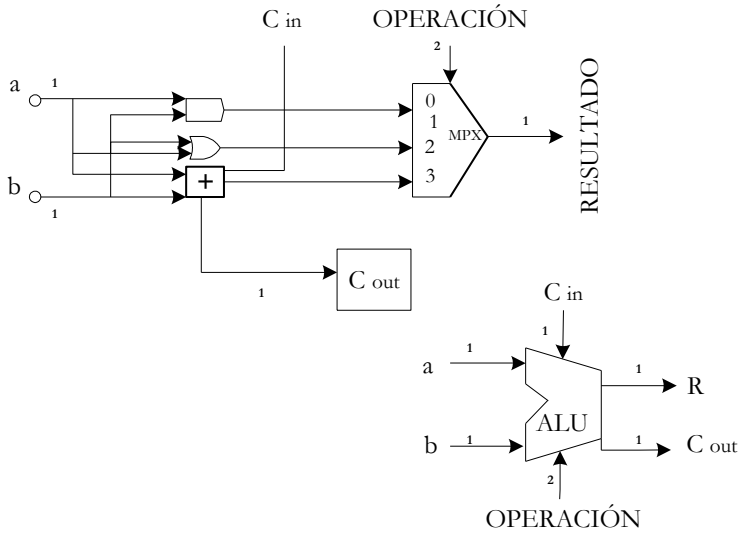


Figura 5—Esquema de la ALU básica de 1 bit, capaz de efectuar las 3 operaciones básicas. Incluyendo el esquema simplificado de este componente.

Por lo tanto, una **ALU** con operandos de 32 bits estará formada por 32 **ALU** básicas conectadas en paralelo.

3.2 Implementación de la resta en la ALU

Para posibilitar la resta binaria en la **ALU** básica sin introducir un restador, se utiliza la técnica del complemento a 2. La resta se realiza mediante una suma utilizando el sumador que ya se disponía en el circuito.

Funcionamiento del complemento a 2:

$$a + \overline{b} + 1 = a + (\overline{b}+1) = a + (-b) = a - b,$$

teniendo en cuenta que (b + 1) es el complemento a 2 de b, o sea, -b.

En la siguiente figura se muestra el esquema modificado de la **ALU** básica para permitir realizar la resta de los dos operandos de 1 bit. Apréciese que cuando se desea efectuar la resta de "a – b", las señales auxiliares **Binv** y **Cin** tienen que valer 1.

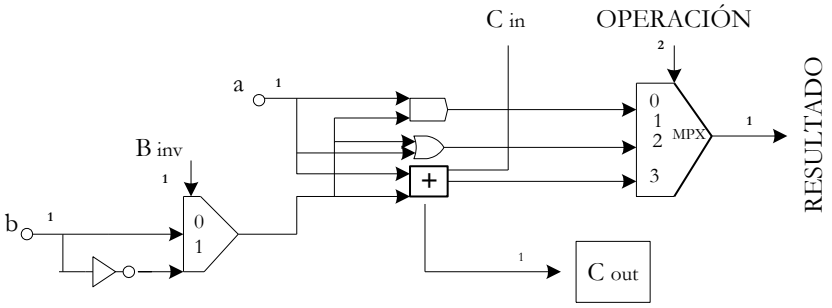


Figura 6—La posibilidad de poder introducir el sumador inverso de b posibilita la resta mediante dicho sumador.

3.3 La operación de detección de menor para inicializar a 1

La función de la instrucción **slt** es comprobar si **rs** es menor que **rt**, en cuyo caso pone a 1 el registro destino **rd**. En caso contrario **rd** se pone a 0. Al final de esta instrucción todos los bits de **rd** son 0 con excepción del bit de menos peso que valdrá 1, únicamente en el caso que **rs<rt** (que ocurrirá cuando el resultado de la resta sea negativo). Para soportar la instrucción **slt** se utiliza la entrada 3 libre del multiplexor de la **ALU** de la figura anterior. A dicha entrada se le aplica la señal **MENOR**. Dicha señal debe valer 0 en todas las **ALU s** elementales de bit, excepto en la que proporciona el bit de menos peso (**ALU0**), cuyo valor depende de la resta de **rs** y **rt**.

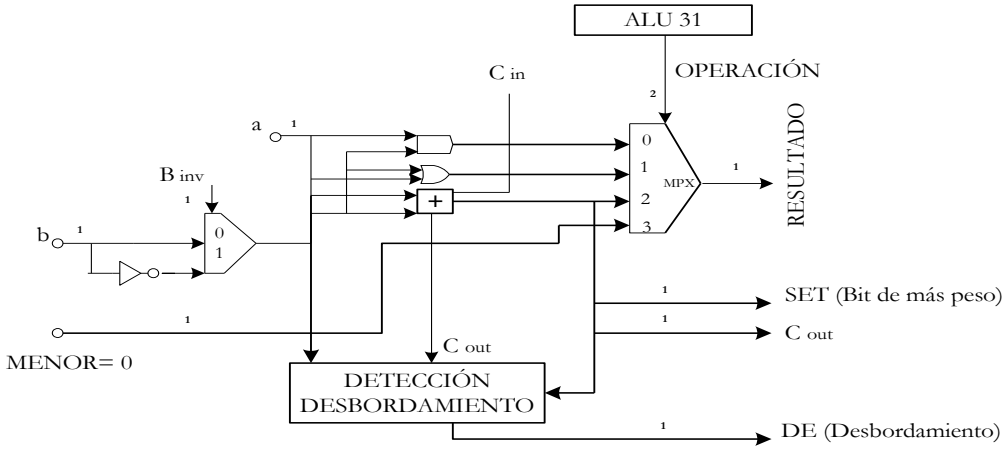


Figura 7—Esquema de la ALU correspondiente a los bits 31, de más peso, con las modificaciones precisas para poder realizar la instrucción slt. Las ALU de los restantes bits no tienen la lógica de "detección del desbordamiento". la entrada MENOR de la ALU0 está conectada a la salida SET de la ALU31, en lugar de ir a tierra.

3.4 La ALU completa del MIPS

En la figura de abajo, se muestra el esquema interno de la **ALU** del **MIPS**, compuesta por 32 **ALU** de 1 bit interconectadas entre sí. Las dos señales **Binv** y **Cin** se constituyen en una sola puesto que siempre tienen el mismo valor. Cuando se efectúa una suma ambas valen 0 y con una resta valen 1. A esta señal única se la denomina **BNEG**.

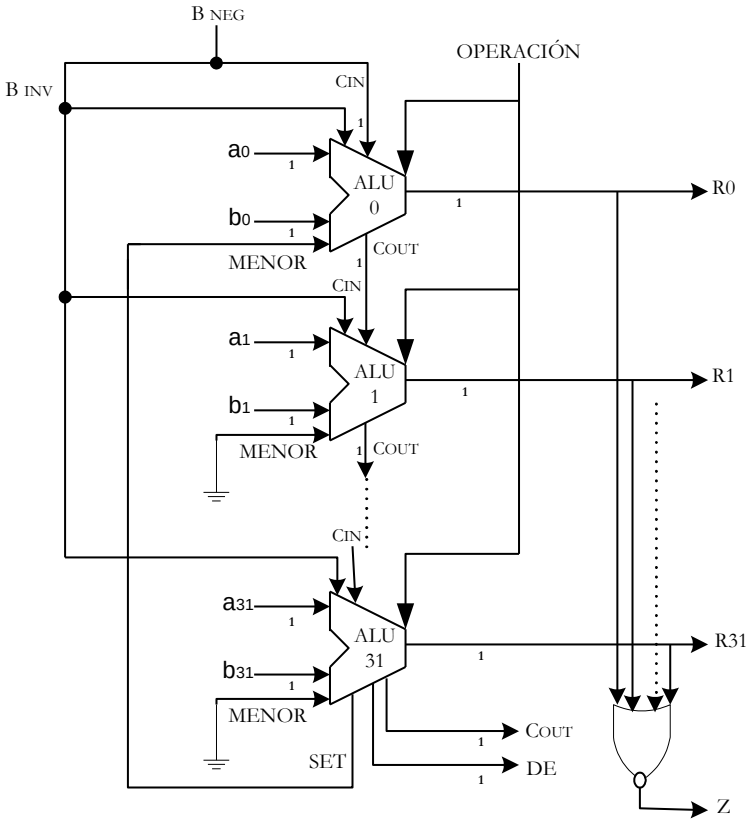
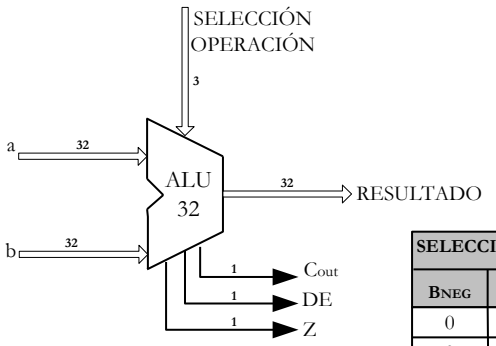


Figura 8—Esquema interno de la ALU del MIPS.

La siguiente figura representa el símbolo simplificado de la **ALU** de 32 bits, junto a la tabla de la verdad que selecciona las operaciones en función de las 3 señales de control.



SELECCIÓN OPERACIÓN			INSTRUCCIÓN
BNEG	OP1	OP0	
0	0	0	AND
0	0	1	OR
0	1	0	ADD
1	1	0	SUB
1	1	1	SLT

Figura 9—Símbolo de la ALU del MIPS y la tabla de la verdad para seleccionar la operación que realiza

3.5 Importancia de la coma flotante

Los modelos comerciales de procesadores **MIPS** disponen de un coprocesador matemático que les permite efectuar numerosas operaciones sobre datos expresados en coma flotante. La principal aportación que supone utilizar números en coma flotante (nota científica) es la de poder manejar con pocos dígitos tanto números muy grandes como pequeños.

MIPS también incorpora en su interior un coprocesador matemático que sigue el formato propuesto por la **norma IEEE 754**, la cual expresa a los números (en binario).

La **norma IEEE 754** dispone de 2 opciones según el tamaño en bits de los operandos. Simple precisión cuando el tamaño es de 32 bits y doble precisión cuando el tamaño alcanza los 64 bits.

4 "DISEÑO DEL CAMINO DE DATOS DEL MIPS MONOCICLO"

4.1.Fundamentos y fases operativas

En este apartado describiremos el funcionamiento del Camino de Datos básico de un procesador **MIPS**, también llamado "**monociclo**".

Se entiende por Camino de Datos la parte del procesador que se encarga de ejecutar las instrucciones del repertorio. Sus principales componentes son los siguientes:

- La **ALU**.
- El Banco de Datos.
- La Memoria de instrucciones y la Memoria de datos. Separadas en el caso del **MIPS monociclo**.

Además, el Camino de Datos dispone de un conjunto de componentes auxiliares (registros, sumadores, etc…), que dirigen la información.

En el Camino de Datos **monociclo** cada instrucción se ejecuta en un solo ciclo de reloj, por lo tanto la duración del ciclo de reloj viene determinada por la duración de la instrucción más larga. Esto provoca que haya una gran perdida de rendimiento en las instrucciones de menor duración ya que consume un ciclo completo sin ser necesario.

Por otra parte, no se puede utilizar más de una vez un recurso con cada instrucción. Por este motivo es necesario disponer de una memoria de instrucciones independiente de la de datos, puesto que en muchas instrucciones hay que acceder a buscar instrucciones y a leer o escribir datos al mismo tiempo.

Las fases en las que se desarrollan las instrucciones pueden agruparse de la siguiente forma:

- 1ª FASE : BUSQUEDA DE LA INSTRUCCIÖN

  Consiste en leer de la Memoria de instrucciones el código de 32 bits correspondiente a la instrucción en curso. La dirección donde se encuentra la instrucción la proporciona el registro

llamado Contador de Programa (PC). Posteriormente el PC se debe incrementar en 4 unidades para que pueda apuntar a la siguiente instrucción de la secuencia.

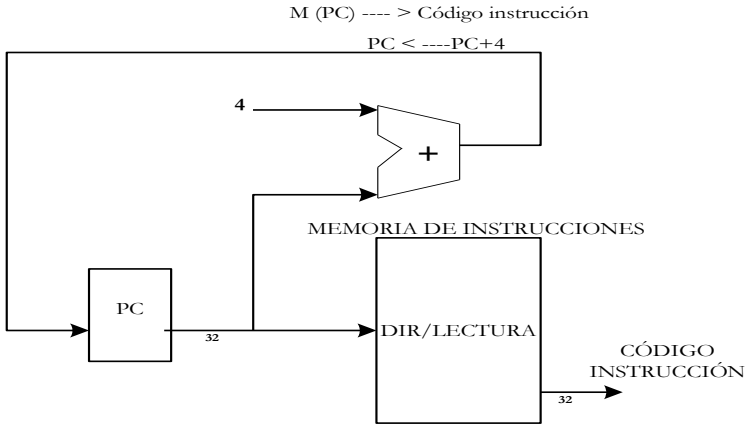


Figura 10—En la fase de búsqueda de la instrucción se lee la posición de la memoria de instrucciones direccionada por el PC y se incrementa en 4 unidades el contenido del PC.

- 2ª FASE : A/ EJECUCIÓN DE LA INSTRUCCIÓN LOGICO_ARITMETICAS

Las operaciones que se realizan en la segunda fase de cada instrucción dependen del tipo que se trate. Las instrucciones lógico-aritméticas las desarrolla la **ALU** utilizando como operandos fuente y destino elementos del Banco de Registros.

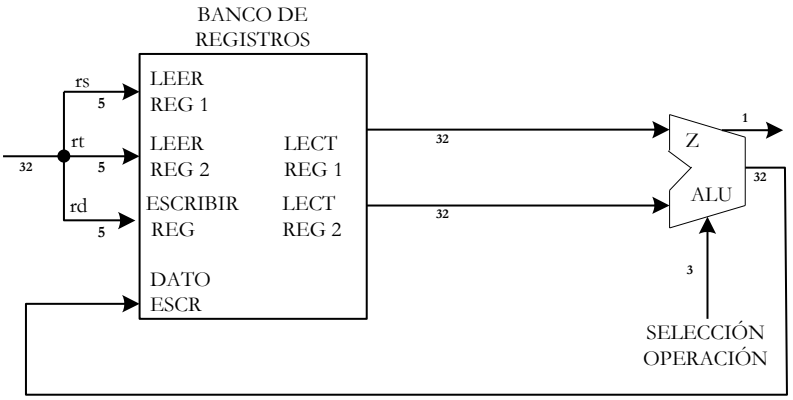


Fig. 11—La fase de ejecución de las instrucciones lógicas y aritméticas consiste en leer los registros rs y rt, y efectuar por medio de la ALU, la operación correspondiente con sus contenidos y el resultado escribirlo en el registro destino rd.

- 2ª FASE : B/ REALIZACIÓN DE LAS INSTRUCCIONES DE TRANSFERENCIA

En este grupo existen dos instrucciones (**lw** y **sw**), ambas responden al formato I.

Por ejemplo la instrucción **lw** $1, DESPL16($3) carga en el registro $1 el contenido de la posición de la Memoria de datos cuya dirección se obtiene mediante la suma del contenido del registro·$3 y el valor inmediato de 16 bits DESPL16 extendido el signo hasta alcanzar el tamaño de 32 bits. Para llevar a cabo esta instrucción la **ALU** debe obtener la dirección a acceder en la memoria de Datos mediante la suma de **rs** y el valor inmediato extendido el

signo. El dato obtenido de la Memoria se aplica por la entrada DATO ESCR y se graba en el registro **rd**.

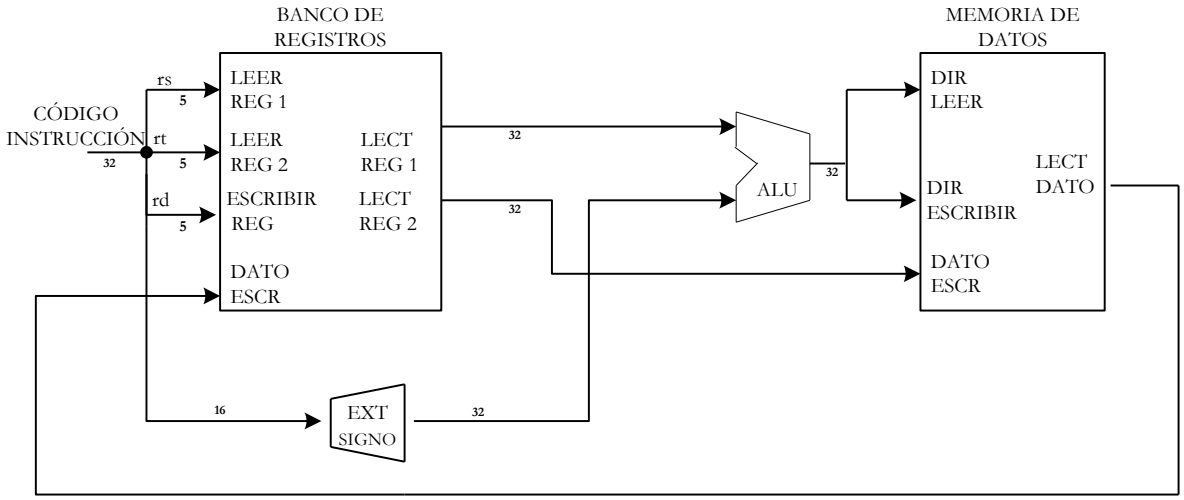


Figura 12—Interconexión precisa entre la ALU, el Banco de Registros y la Memoria de Datos para implementar la ejecución de las instrucciones de transferencia sw y lw.

- 2ª FASE : C/ REALIZACIÓN DE LAS INSTRUCCIONES DE SALTO CONDICIONAL

La instrucción **beq** responde al formato I y en su expresión se utilizan dos registros y un valor inmediato de 16 bits **beq** $3, $4, DESPL16. A este valor inmediato se le extiende el signo hasta completar los 32 bits y se le realiza un desplazamiento de 2 posiciones a la izquierda para hacerlo múltiplo de 4, sumándolo al PC+4 para obtener la dirección de la siguiente instrucción. Teniendo en cuenta que es necesario que el contenido de los dos registros sea igual o lo que es lo mismo que el flag Z=1, al efectuar su resta. Como la **ALU** se emplea para restar los registros operandos (**rs** y **rt**) se necesita un sumador complementario que se encargue de sumar al valor del PC el valor del salto.

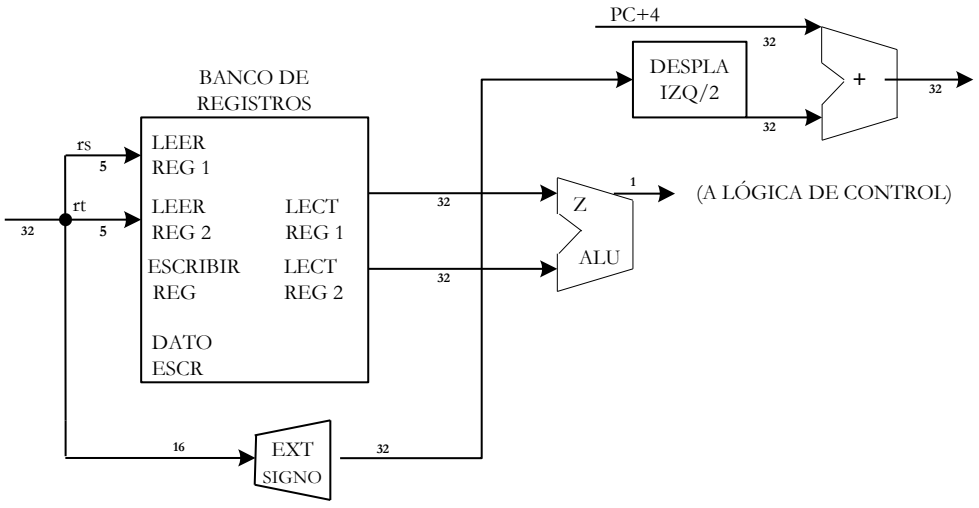


Figura 13—En la instrucción de salto condicional beq la ALU resta el contenido de los registros rs y rt y si Z=1, el sumador deposita su resultado en el PC. El DESPL IZQ/2 desprecia los bits procedentes de la extensión de signo e introduce dos ceros por la derecha para múltiplo de 4.

4.2. Esquema general del camino de datos del monociclo

Solapando los esquemas anteriores correspondientes a las distintas fases se obtiene el esquema general del camino de datos **monociclo**.

Como el CPI en los procesadores es igual a 1, cada vez que se precise utilizar un recurso más de una vez en una instrucción hay que duplicarlo. De aquí surge la necesidad de disponer de una Memoria de Instrucciones y otra de Datos.

Lo que sí es posible es compartir un mismo recurso por diferentes instrucciones. Para ello se requiere seleccionar entre diversas entradas al recurso, con este fin se utilizan frecuentemente los multiplexores.

Combinamos los dos esquemas de los caminos de datos para las instrucciones lógico-aritméticas y las de transferencia, añadiendo los recursos precisos para soportar, la fase de búsqueda de las instrucciones.

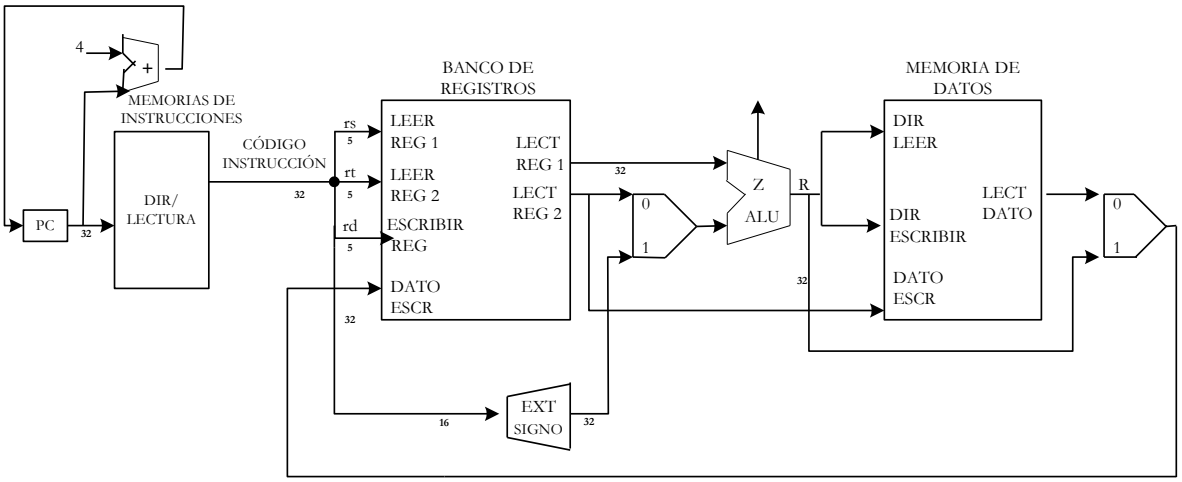


Figura 14—Esquema completo del Camino de Datos monociclo capaz de ejecutar íntegramente las instrucciones lógico-aritméticas y las transferencias.

Añadiendo a la figura anterior los recursos necesarios para resolver la ejecución de la instrucción de salto condicional se llega al siguiente Camino de Datos del **monociclo** del **MIPS**.

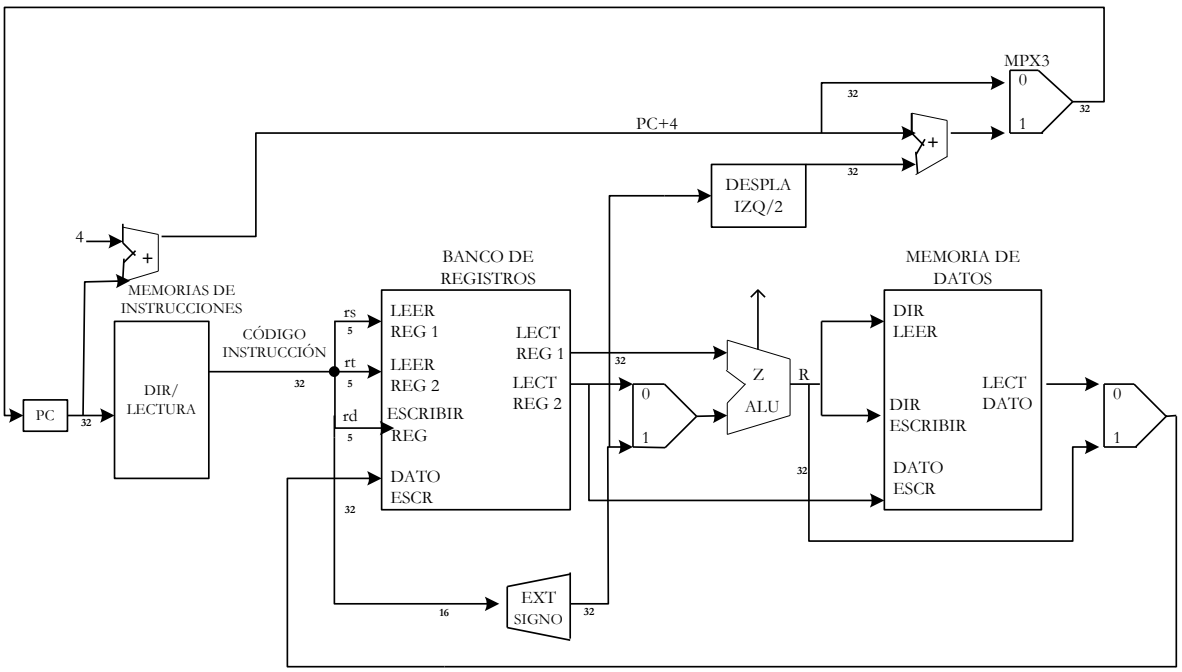


Figura 15—Esquema general del Camino de Datos del MIPS monociclo, que soporta la ejecución de instrucciones lógico-aritméticas, transferencias y salto condicional.

5. "LA UNIDAD DE CONTROL PARA EL PROCESADOR MONOCICLO"

5.1. El papel de la Unidad de Control

La Unidad de Control es el bloque más complejo del procesador. Su misión es la de gobernar las señales de control que regulan la actuación del Camino de Datos en la ejecución de las instrucciones. Tiene que interpretar o decodificar el código máquina de la instrucción en curso para activar aquellas señales que sean precisas para el desarrollo de la misma mediante los recursos existentes en el Camino de Datos del procesador.

En la siguiente figura se muestra el diagrama de bloque de la Unidad de Control del procesador **MIPS monociclo**

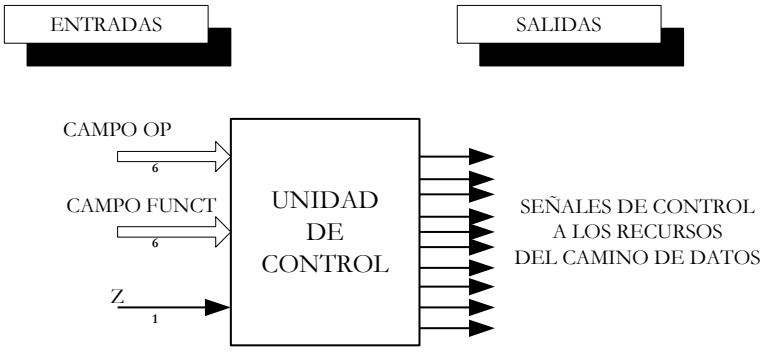


Figura 16—5ª unidad de control del MIPS monociclo.

En cada ciclo se ejecuta una instrucción (**MIPS monociclo**) y en dicho tiempo la Unidad de Control debe activar con el estado adecuado las señales de control que aplicará al Camino de Datos. En el procesador **monociclo** la Unidad de Control responde a un circuito combinacional que sigue una tabla de verdad que tiene como entradas las señales que determinan el tipo de instrucción que se trata y cuyas salidas son los estados correspondientes que deben soportar las señales de control aplicadas a los componentes del Camino de Datos.

Uno de los recursos principales del Camino de Datos a la hora de realizar una instrucción es la **ALU**.

El Banco de Registros siempre dispone de dos entradas que indican los registros que se leen y cuyos contenidos aparecen por las salidas y una tercera entrada que indica el registro que se desea escribir con la información que se aplica por DATO ESCR. Este registro solo se carga cuando se activa una señal de control llamada "Reg Write".

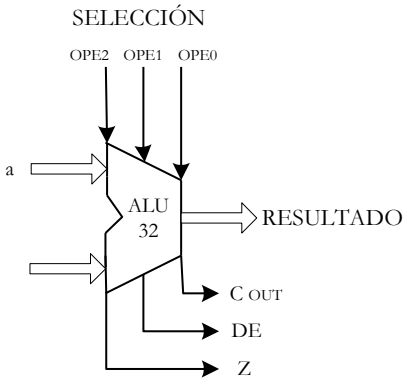
La Memoria dispone de dos secciones independientes, una encargada de contener las instrucciones y otra los datos. La memoria de instrucciones sólo se puede leer, mientras que la de datos se puede leer y escribir. Para controlar la Memora de datos existen dos señales de control denominadas "Mem Read" y "Mem Write".

El Camino de Datos dispone de unos componentes auxiliares que determinan la información que se aplica a los recursos principales. Se trata de los multiplexores

Para poder diseñar la Unidad de Control de un procesador hay que conocer el contenido de las diversas señales de control que gobiernan los recursos del Camino de Datos y el estado que deben tomar en cada instrucción.

5.2. Señales de control de la ALU

La **ALU** del **MIPS** soporta la realización de cinco operaciones diferente de tipo lógico y aritmético, que se seleccionan mediante tres señales de control que llamaremos OPERACIÓN1, OPERACIÓN2, y OPERACIÓN0. Mostramos el signo de la **ALU** y la tabla de verdad que selecciona la operación que realiza.



OPE2	OPE1	OPE0	OPERACIÓN
0	0	0	AND
0	0	1	OR
0	1	0	ADD
1	1	0	SUB
1	1	1	MENOR

Figura 17—Símbolo de la ALU y la tabla de la verdad que muestra la operación que realiza en la función del valor lógico que adoptan sus tres señales de control.

De acuerdo con el valor de 6 bits de más peso del código de la instrucción, que corresponde al campo del código OP, la Unidad de Control genera dos bits que constituyen las señales de control

designadas como **ALU op** y que discriminan entre los tres tipos principales de instrucciones: lógico-aritméticas, de transferencia y el salto condicional.

Para cualquier instrucción de tipo lógico-aritmética la Unidad de Control genera el mismo código del campo **ALU op** (10). Por lo tanto, para identificar la operación que debe realizar la **ALU** y obtener el valor de las tres señales que la controlan, hay que tener en cuenta el campo auxiliar func.

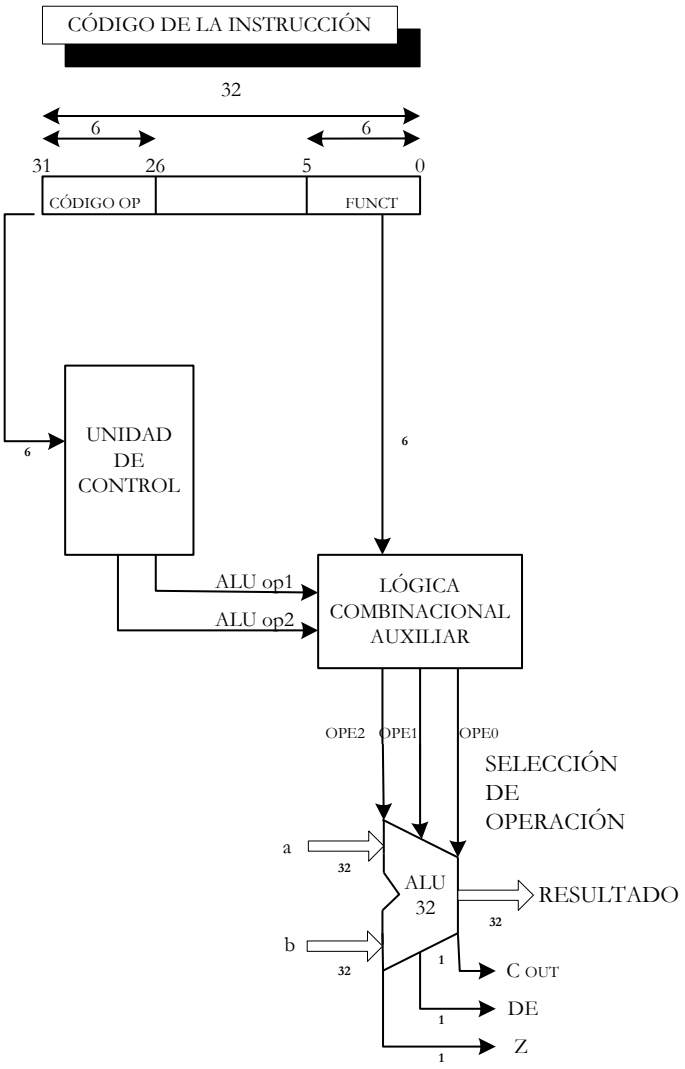


Figura 18—Una lógica combinacional complementaria se encarga de generar las tres señales de control que seleccionan la operación ALU y discriminan las diversas alternativas de las instrucciones de tipo R, mediante el análisis de los bits del campo funct.

En la figura siguiente se muestran los valores de las diversas operaciones que se manejan en los elementos mostrados en el esquema de la figura 18.

Código OP	ALU op	Instrucción	Funct F5-F0	Operación de la ALU	Ope2	Ope1	Ope0
lw	0 0	Cargar	xxxxxx	Sumar	0	1	0
sw	0 0	Almacenar	xxxxxx	Sumar	0	1	0
beq	0 1	Salto condi.	xxxxxx	Restar	1	1	0
tipo R	1 0	Sumar	100 000	Sumar	0	1	0
tipo R	1 0	Restar	100 010	Restar	1	1	0
tipo R	1 0	And	100 100	And	0	0	0
tipo R	1 0	Or	100 101	Or	0	0	1
tipo R	1 0	Menor	101 010	Menor	1	1	1

Tabla 1—Tabla en la que se indican los valores que toman las diversas señales que participan en el funcionamiento de los elementos mostrados en la figura anterior.

5.3. Diseño de la Unidad de Control cableada

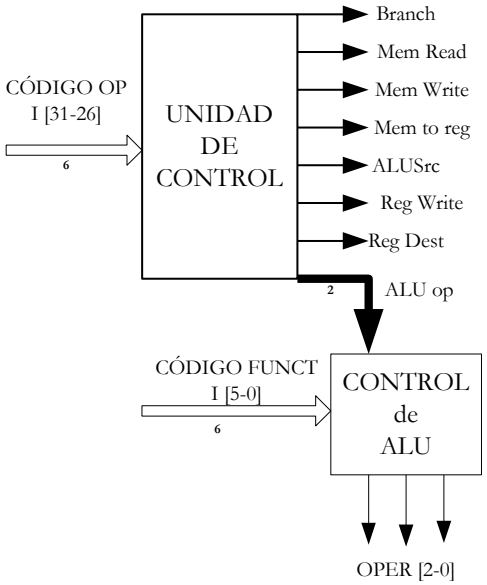


Figura 19—Señales de entrada y salida de la Unidad de Control

Como se conoce el código **op** correspondiente a cada tipo de instrucción se puede generar la tabla de verdad, en la cual las entradas serán los 6 bits del código y las salidas las 9 señales que debe generar la Unidad de Control.

Instrucción	op5	op4	op3	op2	op1	op0	Reg Dest	ALU Src	Mem ro Reg	Reg Write	Mem read	Mem Write	Branch	ALU op1	ALU op0
Formato R	0	0	0	0	0	0	1	0	0	1	0	0	0	1	0
lw	1	0	0	0	1	1	0	1	1	1	1	0	0	0	0
sw	1	0	1	0	1	1	x	1	x	0	0	1	0	0	0
beq	0	0	0	1	0	0	x	0	x	0	0	0	1	0	1

Tabla 2—Tabla de la verdad que establece las salidas que debe generar la Unidad de Control en función del estado que se recibe por sus entradas y que corresponde al código OP de la instrucción a ejecutar.

Implementación física de la Unidad de Control y tabla de control del **MIPS monociclo.**

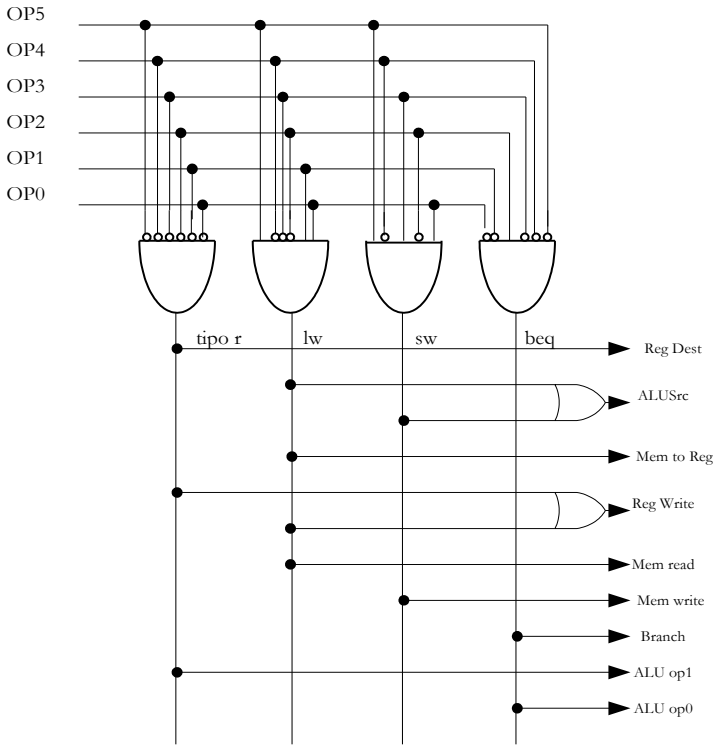


Figura 20—Implementación física de la Unidad de Control, mediante una PLA que resuelve las ecuaciones lógicas de las señales de control en función de los estados del código OP de cada instrucción.

5.4. Implementación de la instrucción de salto incondicional

La instrucción **j A** ocasiona un salto incondicional a una dirección de la Memoria de Instrucciones que se obtiene con el dato inmediato **A**. Su formato:

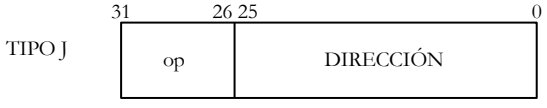


Figura 21—Formato J correspondiente a la instrucción j Dirección.

La dirección a la que se salta con esta instrucción y que se carga en el PC, se calcula de la siguiente forma:

- Se mantiene el valor de los 4 bits del PC.
- Detrás de los 4 bits de más peso del PC que no varían se colocan los 26 bits del valor inmediato que viene en la instrucción.
- Se añaden dos bits cero para ocupar las dos posiciones de menos peso haciendo que el valor sea múltiplo de 4.

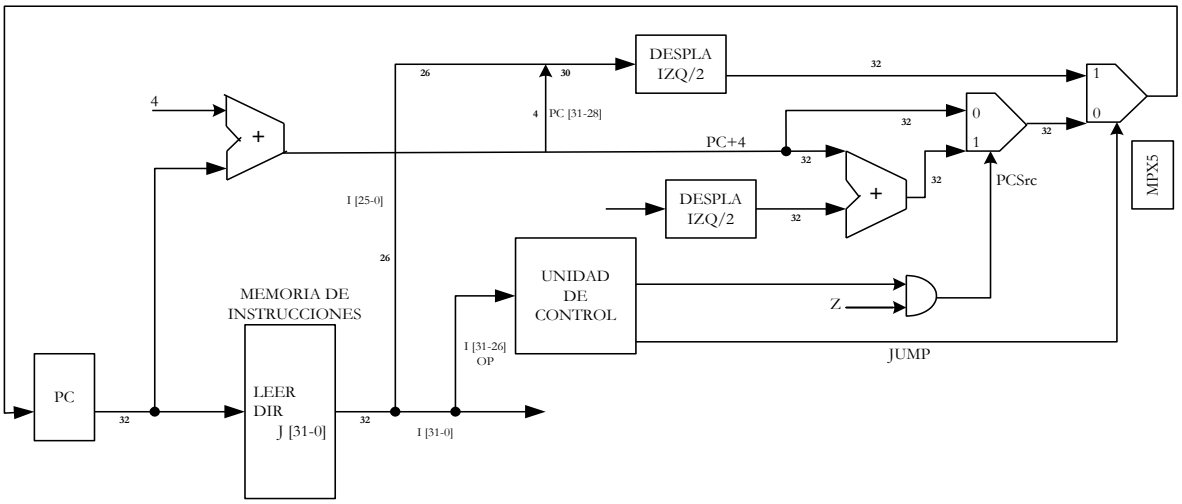


Figura 22—Elementos que hay que añadir al esquema del procesador monociclo para implementar la instrucción de bifurcación incondicional

5.5. Esquema general del procesador monociclo

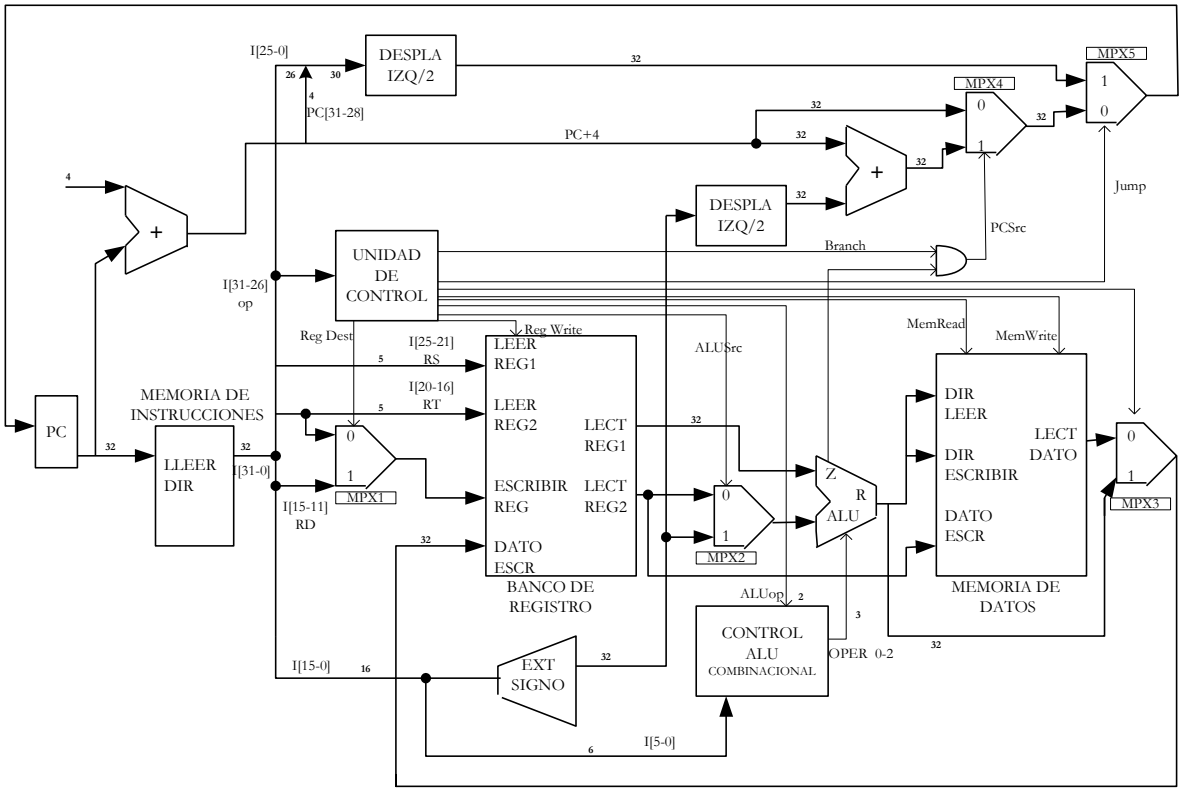


Figura 23—Esquema general del procesador MIPS

5.6. Problemática del procesador monociclo

La característica principal del procesador **monociclo** es su capacidad de realizar todas las instrucciones en un solo ciclo de reloj. Esto conlleva que la duración del ciclo del reloj sea la correspondiente a la instrucción más larga, lo que supone una gran pérdida del rendimiento del computador, pues muchas instrucciones simples tardarán en ejecutarse el mismo tiempo que la instrucción más compleja del repertorio. La instrucción más larga del **MIPS monociclo** es la **lw**, pues lee la Memoria de Instrucciones, lee el Banco de Registros, realiza una suma la **ALU**, se lee la Memoria de Datos y finalmente se escribe en el Banco de Registros. Para mejorar el Camino de Datos **monociclo** pasamos al **multiciclo**.

6- CAMINO DE DATOS DEL MIPS MULTICICLO

La aparición del MIPS multiciclo viene como solución a la gran pérdida de rendimiento del MIPS monociclo, ya que en este todas las instrucciones se ejecutaban en un mismo ciclo. En una implementación multiciclo, cada paso de la ejecución, empleará un ciclo de reloj.

La implementación multiciclo permite que una unidad funcional sea utilizada más de una vez por instrucción, mientras se utilice en diferentes ciclos de reloj. La posibilidad de que las instrucciones

empleen diferentes números de ciclos de reloj y la posibilidad de compartir unidades funcionales en la ejecución de una única instrucción son las mayores ventajas del diseño multiciclo. Esto supone una gran ventaja a nivel hardware, reflejándose en su arquitectura en dos aspectos:

1º. Utilización de una única memoria para instrucciones y datos.

2º. Se utiliza un registro para guardar la instrucción una vez que se lee. Este **Registro de Instrucciones (RI)** se necesita porque la memoria puede ser reutilizada para acceder más tarde al dato en la ejecución de la instrucción.

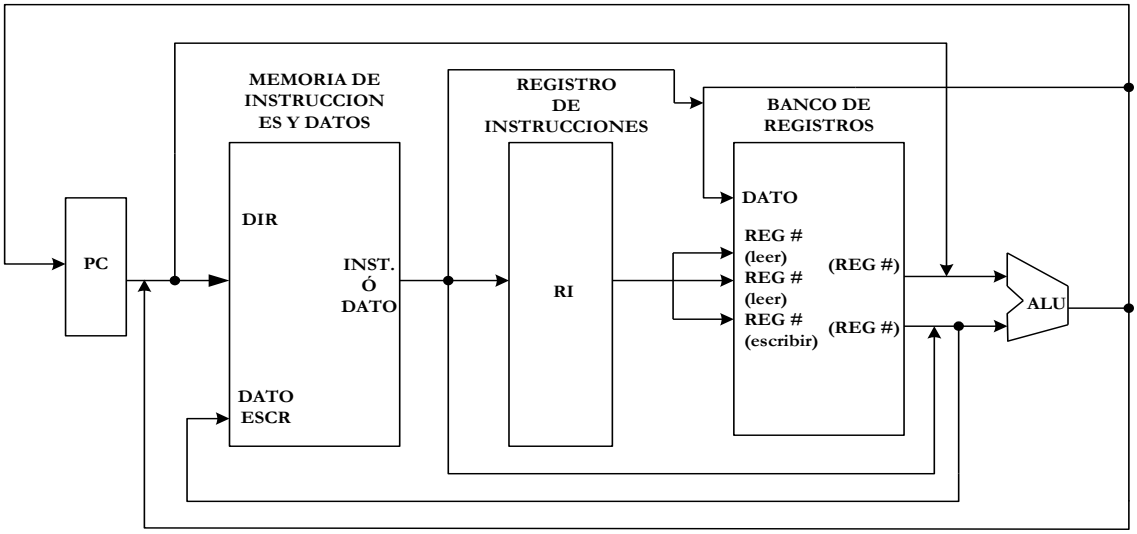


Figura 24.-Diagrama general del camino de datos multiciclo del MIPS.

Hay una sola ALU, en lugar de una ALU y dos sumadores.

Como en una instrucción varios recursos pueden ser utilizados para diversos propósitos, es necesario añadir algunos multiplexores:

1º. Como utilizamos una memoria tanto para datos como para instrucciones necesita un multiplexor de dos entradas para seleccionar entre las dos posibles fuentes de una dirección de memoria, a saber, el PC (para acceder a la instrucción) y el resultado de la ALU (para acceder a los datos).

2º. El multiplexor de la segunda entrada de la ALU hay que ampliarlo con 4 entradas ya que vamos a eliminar todos los sumadores auxiliares (la ALU realizará todas las operaciones).

3º. Se añade otro de 2 entradas a la primera entrada de la ALU, para seleccionar entre introducir a la ALU el contenido del registro **rs** o el del **PC**.

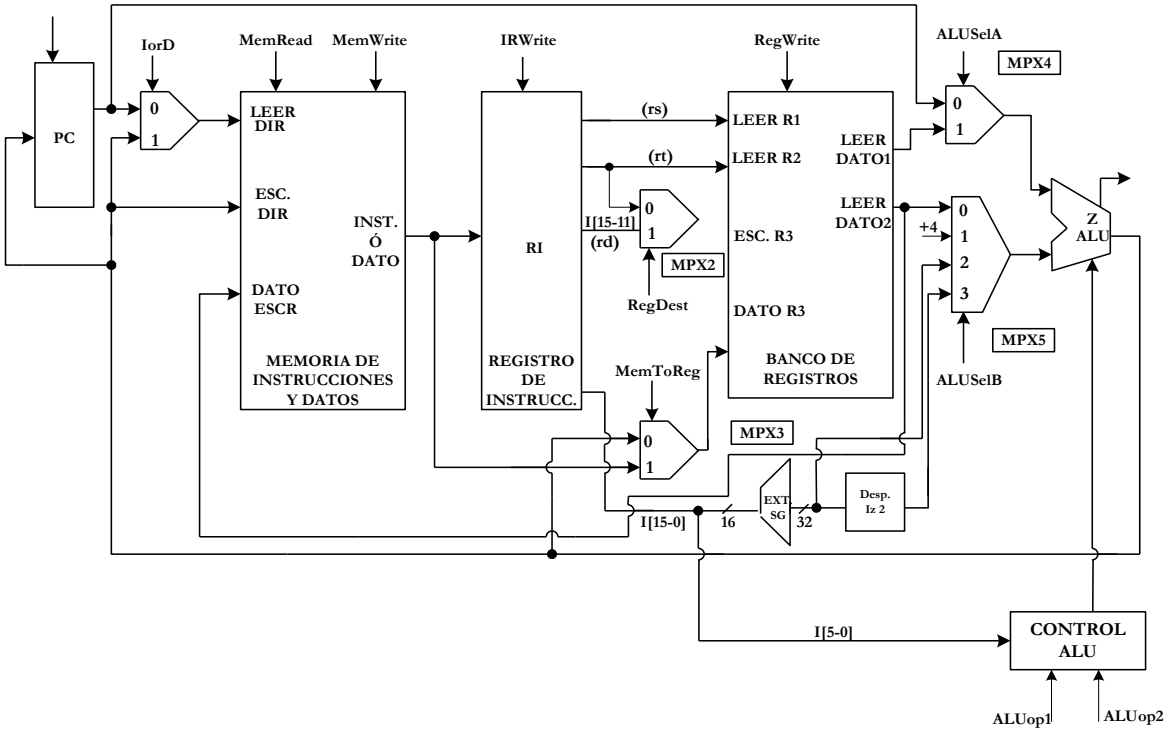


Figura 25.- Esquema del camino de datos del multiciclo de MIPS.

6.1- Descomposición de las instrucciones en operaciones elementales

Al realizarse en cada ciclo una instrucción elemental de las que configuran la instrucción en ejecución , se debe conseguir equilibrar los tiempos de las operaciones elementales, es decir, que duren más o menos lo mismo. Para ello, solo se puede dejar que en un mismo ciclo se utilice en secuencia un único recurso de los más lentos: ALU, Memoria y Banco de Registros. Puesto que la instrucción que más tarde en ejecutarse será la que determine el ciclo de trabajo del procesador.

Se precisa el uso de registros en los siguientes dos casos:

1º. La señal se calcula en un ciclo de reloj y se utiliza en otro.

2º. Las entradas al bloque funcional de donde sale esta señal pueden cambiar antes que la señal se escriba en un elemento de estado.

Necesitamos almacenar el valor en RI por que la memoria, que produce el valor cambia su salida antes que completemos todos los usos de los campos de la instrucción.

No se precisa un registro en la salida de la ALU, en instrucciones de formato R, porque el código de registros a leer (rs, y rt) no cambia en toda la instrucción, sino que provienen del registro RI.

Al descomponer una instrucción en operaciones más elementales, algunos recursos pueden operar en **paralelo** solapándose sus tiempos, mientras que otros trabajan en **serie** acumulando tiempos.

La operación elemental más larga determina la duración del ciclo.

Las instrucciones se dividen en cinco operaciones elementales o pasos:

1º. Paso de búsqueda de instrucción:

Busca la instrucción en memoria e incrementa el contador de programa. También se le

denomina fase de búsqueda.

IR = Memoria[PC];

PC = PC + 4;

En esta operación se realizan dos instrucciones en paralelo. Por un lado se lee la memoria para recoger el código de la instrucción y almacenarlo en el registro **RI**. Por otro lado y puesto que está inactiva la ALU se procederá a sumar 4 unidades al PC, para que apunte a la dirección de la siguiente instrucción a ejecutar.

Las señales de control toman los siguientes valores:

MemRead = 1; IRWrite = 1; IorD = 0 [M(PC) = RI]

ALUSelA = 0; ALUSelB = 01; ALUop = 00 (suma) [PC = PC + 4]

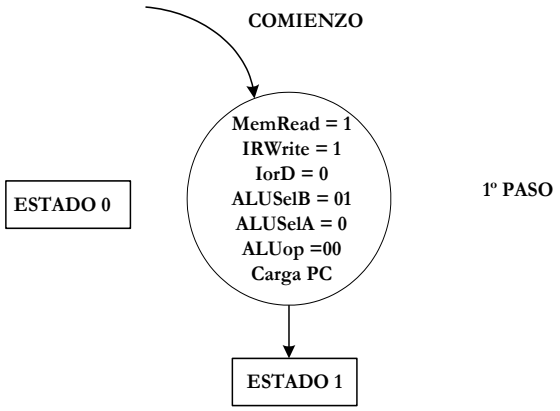


Figura 26.- Representación de la operación elemental de búsqueda de la instrucción. Se denomina estado 0.

2º. Paso de decodificación de la instrucción y búsqueda de registros :

El segundo paso sería la decodificación que es común a todas las instrucciones y se encuentra en el registro RI. Se denomina **estado 1**.

Durante la decodificación se lleva a cabo la lectura de los registros especificados por rs y rt, obteniéndose los valores de A y de B.

A = [rs]; rs =I[25-21]

B = [rt]; rt = I[20-16]

También calcularemos la dirección de salto, porque no sabemos si esta instrucción va a realizar el salto. La almacenaremos en un registro auxiliar (Target), por si la debemos utilizar más adelante.

Target (destino) = PC + IR[15-0]EXT-SIGNO

Para calcular la dirección Destino de salto condicional, se accede al banco de registros, el cual proporcionará por sus dos salidas de lectura el contenido de los registros especificados por sus entradas, sin necesidad de activar señales de control. Se calcula la dirección destino y se almacena en Target. Para ello se debe inicializar **ALUSelB** al **valor 11** (para que el campo de desplazamiento, que tiene los 16 bits de menos peso, esté con el signo extendido y desplazado), **ALUSelA a 0** y **ALUOp a 00**. Para almacenar en Target la señal de control **Target Write tiene que valer 1**.

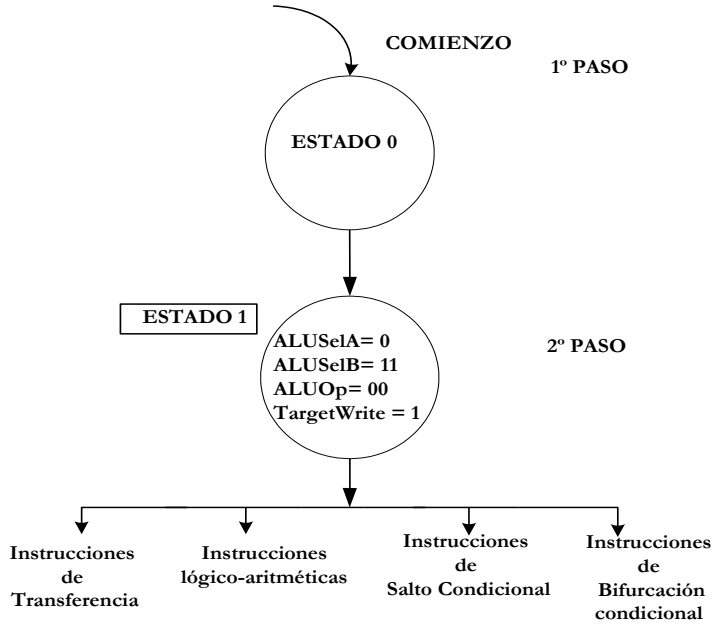


Figura 27.- El estado 1 es común para todas las instrucciones y requiere la activación de 4 grupos de señales diferentes.

Después de este ciclo de reloj, el siguiente estado depende de la siguiente instrucción a ejecutar.

3º. Paso: Selección de estado según el tipo de instrucción :

La salida de la ALU no es necesario guardarla en un registro temporal, porque las entradas van a ser estables e invariables durante el transcurso de la instrucción, al seleccionarse con los campos rs y rt provenientes de RI.

A) Ejecución de la operación en las instrucciones lógico-aritméticas (ESTADO 6).

Se obtiene el resultado por la ALU de la operación que hay que hacer con el contenido de los dos registros fuentes:

ALU = ALU = A opB

Las señales de control a activar son:

ALUSelA=1; ALUSelB=00; ALUOp=10

B) Cálculo de la dirección de memoria en las instrucciones de transferencia (ESTADO 2).

El tercer estado en las instrucciones de transferencia de información con la Memoria de Datos, la ALU calcula la dirección de acceso.

ALU_{out}= A + I[15-0]EXT-SIGNO

Para realizar esta suma hay que activar la siguientes señales de control:

ALUSelA=1; ALUSelB=10; ALUOp=00

C) Terminación del salto condicional BEQ (ESTADO 2).

Su tercer y último estado los dedica a comparar el contenido de los dos registros, mediante su resta por parte de la ALU (el resultado no se guarda) y la activación consecuente del flan Z. Si **Z=1** el contenido de Target hay que llevarlo a PC para que se pueda producir el salto:

ALU_{out}=A – B; si Z=1, PC = Target

Las señales de control son:

ALUSelA=1; ALUSelB=00; ALUOp= 01(resta)

Si **Z=0**, el PC apunta a la siguiente instrucción en secuencia.

6.1.1- Control de escritura del PC

El PC se puede cargar desde tres puntos diferentes:

1º. Puede cargarse con **PC + 4** cuando se sigue la secuencia normal del programa.

2º.Si se trata de un salto condicional y **Z=1**, se carga desde el registro auxiliar **Target**.

3º.Si se trata de una bifurcación incondicional se carga con los 4 bits de más peso del PC, seguido por los 26 bits de menos peso del código de la instrucción I[25-0] y finalmente, por 2 ceros(la extensión de signo).

Para seleccionar este punto se usa un multiplexor de 4 entradas, aunque una se invalida. Se controla con dos bits del campo **PCSorce**. Si vale **00** selecciona PC + 4, si vale **01** el Target y si es **10** la dirección de la bifurcación incondicional.

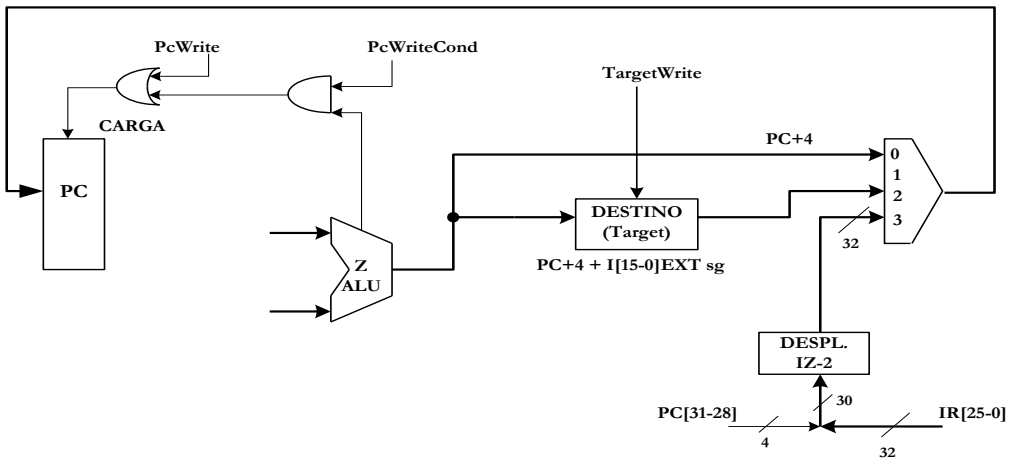


Figura 28.-Control de la carga del PC.

Si no es una instrucción beq se activa **PCWrite** y el PC se carga con lo que procede del multiplexor. Si se trata de una beq, **PCWrite =0** y **PCWrteCond=1**, pero el PC no se carga con lo que le llega del multiplexor, a menos que Z=1, sino permanece con el valor que se había cargado.

D) Terminación de las instrucciones de bifurcación incondicional (ESTADO 9).

La instrucción de bifurcación condicional, j, se completa en el tercer estado, cargando al PC con la dirección del salto.

Las señales de control que se deben activar son:

PCWrite = 1 y PCSrc = 10

Tras esta operación se inicia la siguiente instrucción, que será la que ha determinado la bifurcación.

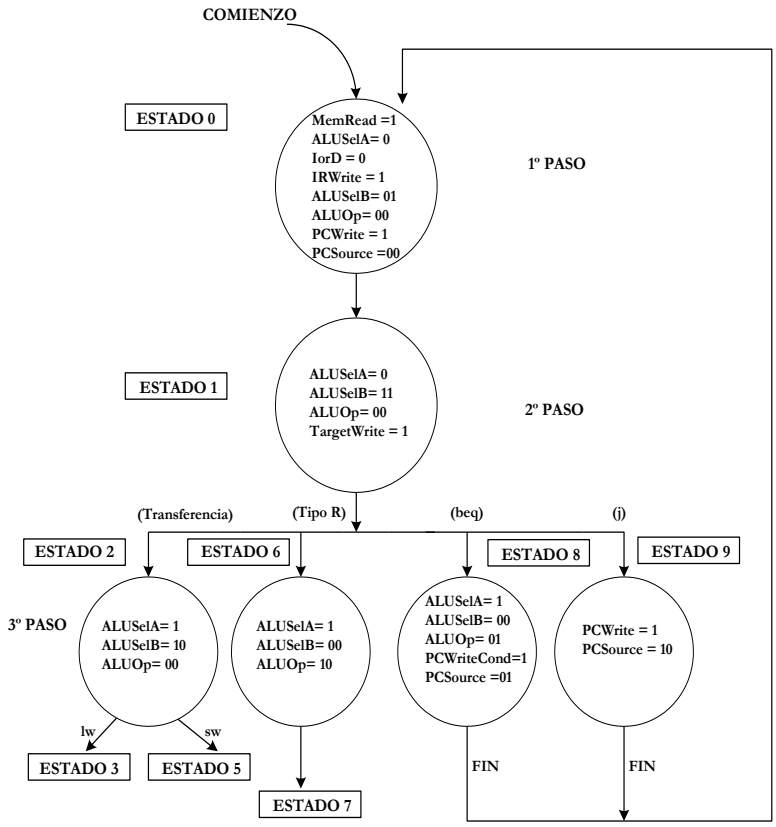


Figura 29.-Diagrama con los tres primeros estados de los posibles tipos de instrucciones del MIPS.

Las instrucciones de bifurcación y salto terminan en el tercer estado.

4º. Paso: Instrucciones tipo r y de transferencia :

Sólo lo realizan las instrucciones lógico-artméticas, que finalizan en este estado, y las de transferencia.

- **Instrucciones lógico-artiméticas(Estado7)**

  Se escribe el resultado de la ALU en el registro destino(rd).

  ALU_{out}=rd(I[15-11])

  Se activan las siguientes señales: ResDest= 1; Reg Write=1; MentoReg=0

  Las señales que gobiernan la ALU no se modifican.

- **Instrucciones de transferencia**

  Son dos las instrucciones de este grupo:

  1.- La instrucción de carga (lw) que lee una dirección de la Memoria de Datos y su contenido lo carga en el registro principal rt.

  2.- La instrucción de almacenamiento (sw) el contenido del registro rt lo escribe en la dirección de acceso a la Memoria de Datos. El cuarto paso es diferente según se trate de carga o almacenamiento.

- **Instrucción de carga (ESTADO 3)**

  En el cuarto paso, se lee la dirección de la Memoria de Daros que se ha calculado en el paso anterior, obteniéndose un paso para el quinto.

M(ALU$_{out}$)= Dato

Se lee la Memoria de Datos y se activan **MemRead = 1** y **IorD = 1**. Las señales de la ALU permanecen invariables.

- **Instrucciones de almacenamiento (ESTADO 5)**

Se escribe el contenido del registro rt en la posición direccionada por la salida de la ALU.

(rt) = M(ALU$_{out}$)

Se activan **Mem Write=1** y **IorD = 1**. Las señales de la ALU no varían.

5º. Paso: Terminación de la instrucción (ESTADO4):

La instrucción de carga (LW) es la unica que tiene cinco pasos. Es la más larga, y se encarga de escribir el dato leído de la Memoria de Datos en el registro rt.

Dato = (rt); rt = I[20-16]

Se activan **MemToReg = 1**, **Reg Write= 1**, **Reg Dest = 0**. Las señales de la ALU no varían.

La figura 1B.30 muestra el diagrama de estados completo al que responde el repertorio básico de instrucciones del MIPS.

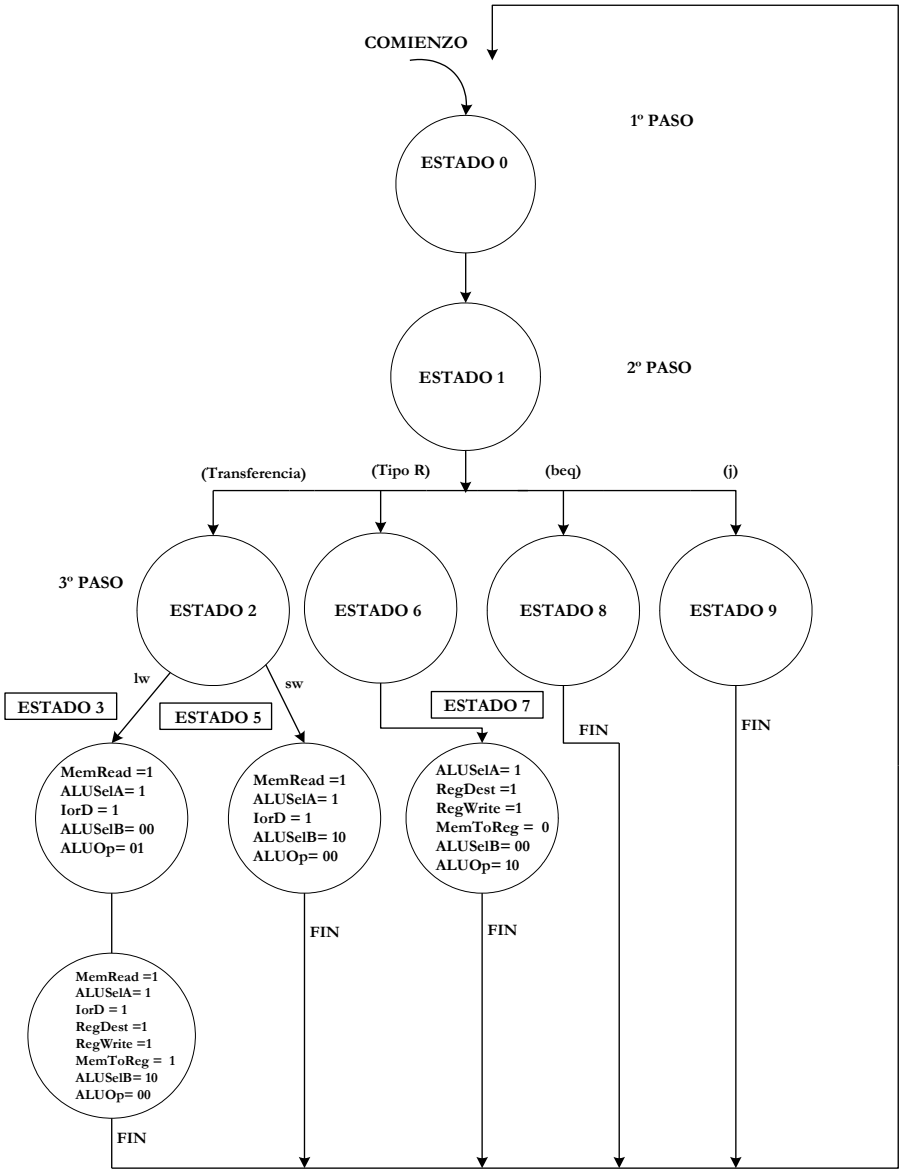


Figura 30.-Diagrama con los tres primeros estados de los posibles tipos de instrucciones del MIPS.

7- LA UNIDAD DE CONTROL PARA EL MIPS MULTICICLO

Se distinguen dos modelos de diseño o construcción de Unidades de Control:

1º. La Unidad de Control Cableada

2º. La Unidad de Control microprogramaza

7.1- Unidad de Control Cableada

Es un circuito, basado en hardware, encargado de generar las señales de control de cada estado. Presenta como una ventaja la rapidez de respuesta, pero su gran inconveniente consiste en su elevado coste. Por ello este tipo de Unidad de Control solo se emplea para procesadores de elevadas prestaciones.

Para su diseño se parte de la tabla de la verdad a la que responde el sistema. Las entradas de dicha tabla están formadas por los 6 bits del código Op (OP5-0) de la instrucción y los 4 bits (S·-0) que definen el estado actual dentro de los 10 posibles que existen en el grafo de estados del repertorio de instrucciones. Las salidas están formadas por los valores lógicos que toman las señales de control y por los 4 bits (S´3-0) que indican el estado siguiente que corresponde.

	ENTRADAS		**SALIDAS**				
	CÓDIGO OP	**ESTADO ACTUAL**				**ESTADO SIGTE.**	
ESTADO ACTUAL	**OP5 OP4 OP3 OP2 OP1 OP0**	**S3 S2 S1 S0**	**ALUop**	**ALUSelA**	**PCSource**	**S3´ S2' S1' S0'**	**ESTADO FIGURADO**
0	X X X X X X	0 0 0 0	0 0	0	0 0	0 0 0 1	1
1	CÓDIGO OP = lw y sw	0 0 0	0 0	0	- -	0 0 1 0	2
1	CÓDIGO OP = beq	0 0 0	0 0	0	- -	1 0 0 0	8
1	CÓDIGO OP = tipo R	0 0 0	0 0	0	- -	0 1 1 0	6
1	CÓDIGO OP = j	0 0 0	0 0	0	- -	1 0 0 1	9

Tabla 3.- Representación de una parte de la tabla de la verdad a que corresponde el grafo de estados del repertorio de instrucciones del MIPS multiciclo.

Si el número de instrucciones del repertorio y los modos de diseccionado son elevados, el grafo de estados se hace muy complejo y la tabla de la verdad admite miles de combinaciones. Debido a su elevado coste, en estos casos no es posible realizar un diseño cableado de la Unidad de Control.

Una vez obtenida la tabla de la verdad se pueden deducir las ecuaciones lógicas a las que responden las salidas en función de las entradas.

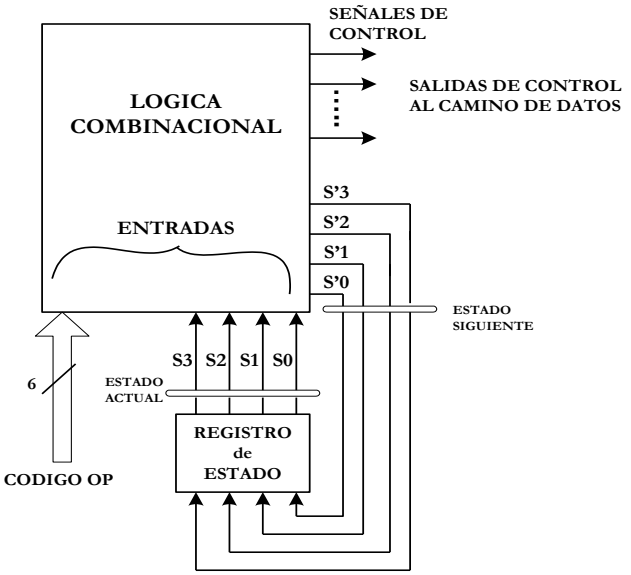


Figura 31.- Esquema general de la Unidad de Control cableada del MIPS multiciclo.

7.2- Unidad de Control Microprogramada

Cuando el número de instrucciones de repertorio y los modos de direccionado son elevados, se debe diseñar mediante **microprogramación**. Esta técnica es mucho más lenta que la Unidad de Control cableada pero tiene una gran ventaja ya que es barata y sencilla de implementar montones de instrucciones.

Está basada en la grabación de una memoria especial, denominada **memoria de control**. Cada posición de dicha memoria guarda el valor lógico de las señales de control que debe generarse para un estado determinado de una instrucción. Por lo tanto una instrucción está implementada en la memoria de control por un conjunto deposiciones que representan las diversas microinstrucciones o estados de la máquina. A este pequeño programa se le llama **microprograma**.

Según el secuenciamiento de las microinstrucciones existen dos tipos de Unidades de Control microprogramadas. Las de **Secuenciamiento Implícito**, en que todas las microinstrucciones de cada instrucción están grabadas en la memoria de control de forma secuencial y las de **Secuenciamiento Explícito**, en la que la dirección de la siguiente microinstrucción se especifica en uno de los campos de la microinstrucción anterior.

7.3- Formato de las microinstrucciones

En el formato de las microinstrucciones del MIPS multiciclo se pueden distinguir los siguientes campos:

CONTROL ALU: Especifica la operación de la ALU.
SCR1: Especifica la fuente para el primer operando de la ALU.
SCR2: Especifica la fuente para el segundo operando de la ALU.
DESTINO ALU: Indica el registro donde se guardará el resultado.
MEMORIA: Especifica lectura o escritura y la fuente de dirección.

REGISTRO MEMORIA: Especifica el registro que se carga con el dato leído en la memoria o que se escribe en ella.
CONTROL PCWRITE: Especifica como coger la siguiente instrucción que se va a ejecutar.

Existe otro campo llamado **Secuenciamiento,** que especifica la dirección donde se encuentra la siguiente microinstrucción a ejecutar. La Unidad de Control con Secuenciamiento Explícito tiene 3 posibilidades de determinar la siguiente microinstrucción a ejecutar:

1º. El campo secuenciamiento , toma el valor **seq** cuando la microinstrucción siguiente ocupa la dirección consecutiva actual.

2º. Cuando en el microprograma correspondiente a una instrucción se llega a la última microinstrucción, la siguiente es la primera de la siguiente instrucción, que siempre es una de tipo **fetch**. En el campo Secuenciamiento se indica con la opción **fetch**.

3º. Cuando la siguiente instrucción no es fija, sino que hay varias alternativas, según alguna entrada a la Unidad de Control, que generalmente suele ser el código OP. Para estos casos se hace una tabla para cada uno de los estados con esta posibilidad. En el caso de los MIPS esto pasa en el estado 1 y en el 2. Indicamos que la microinstrucción siguiente debería escogerse por una operación de distribución colocando **Dispatch i**, donde **i** es el número de la tabla de distribución en el campo Secuenciamiento ("Sequencing").

En la siguiente tabla se muestran los campos de las microinstrucciones del MIPS multiciclo, con los valores que admiten y la función que realizan.

NOMBRE DEL CAMPO	VALORES QUE ADMITE	FUNCIÓN QUE REALIZA
CONTROL ALU	Add Function case Sub	Suma la ALU (Transferencia) Depende del código de función (logic-arit) Resta (beq)
SRC1	PC Rs	1ª Entrada ALU PC 1ª Entrada ALU el rs
SRC2	4 Extend Extshift Rt	2ª Entrada ALU 4 I[15-0] 2ª Entrada ALU ext signo I[15-0] 2ª Entrada ALU ext signo y des.2 izq 2ª Entrada ALU rd
DESTINO ALU	Target (Destino) Rd	La salida de ALU se escribe en TARGET La salida de ALU se escribe en rd
MEMORIA	Read PC Read ALU Write ALU	Lee memoria con dirección PC Lee memoria con dirección ALU_{out} Escribe memoria con dirección ALU_{out}
REGISTRO MEMORIA	IR Write rt (lw) Read rt (sw)	El dato leido en mem se escribe en IR El dato leido en mem se escribe en rt El dato escrito en mem proviene de rt
CONTROL PCWRITE	ALU Target-Cond Jump Address	Escribe en PC Si Z= 1 PC TARGET Escribe PC con dirección bifurcación
SECUENCIAMIENTO	Seq Fetch Dispatch i	Elige secuencialmente la siguiente μinstrc. Va a 1ª μinstrc. De una nueva instrucción Distribuye utilizando la tabla i Dispatch

Tabla 4.- Tabla con los campos que componen las microinstrucciones del MIPS multiciclo y sus principales características.

En la tabla 5 se muestran los valores que toman los diversos campos correspondientes a las dos primeras microinstrucciones de cada instrucción. La primera microinstrucción para las 9 instrucciones del repertorio básico, que siempre son las mismas, es la **fetch**, que reailza la fase de búsqueda de instrucción, M(PC) = IR, y PC = PC + 4. La segunda, es la que realiza la fase de decodificación, por lo

que se denomina **decodif,** y además busca los dos operandos rs yrt y calcula la dirección de salto para las beq.

µInstrcción rotulo o etq.	Contol ALU.	SRC1	SRC2	Destino ALU	Memoria	Registro Memoria	Control PCWrite	Secuenciamiento
Fetch (1ª)	Add	PC	4		ReadPC	IR	ALU	Seq
Decodif.(2ª)	Add	PC	Extshift	Target				Dispatch 1

Tabla 5.- Tabla que muestra el valor que toman los campos de las dos primeras microinstrucciones , que son comunes a todas las instrucciones del MIPS multiciclo.

Para la primera microinstrucción los campos CONTROL ALU, SCR1 y SCR2 determinan la operación PC = PC + 4. Los campos relativos a memoria y registro-memoria, determinan la lectura de la memoria y la carga en IR el código de la instrucción M(PC) = IR, mientras que el campo PCWRITE permite que el PC se cargue con la salida de la ALU y el secuenciamiento será de tipo seq, ya que de la primera microinstrucción siempre se pasa a la segunda.

En cuanto a la segunda microinstrucción los campos que controlan la actuación de la ALU consiguen que se realice la operación PC + (EXTSG IR[15-0]x 4) = Target. Para el campo de Secuenciamiento hay que acudir a la Tabla 1 que determina según el código OP de la instrucción el siguiente estado o microinstrucción que se debe ejecutar.

En la tabla 6 se muestran las microinstrucciones correspondientes a las instrucciones de transferencia. La primera microinstrucción de la tabla, LWSW1, es igual para las dos instrucciones de transferencia lw y sw. Corresponde al estado 2 del grafo de estados del MIPS multiciclo. EN cuanto a LW2 y LW3 son las dos microinstrucciones que completan la instrucción lw, mientras que la microinstrucción SW" es la que completa la instrucción sw. La microinstrucción LWSW1 es la encargada de calcular la dirección de la memoria que hay que acceder. LW2 lee dicha posición y obtiene un dato para que en lw3 dicho dato se escribe en rt. En SW se escribe el valor de rt en la dirección calculada anteriormente.

µInstrcción rotulo o etq.	Contol ALU.	SRC1	SRC2	Destino ALU	Memoria	Registro Memoria	Control PCWrite	Secuencia-miento
LWSW	Add	rs	Extend					Dispatch 2
LW2	Add	rs	Extend		ReadALU			Seq
LW3	Add	rs	Extend		ReadALU	Write rt		Fetch
SW	Add	rs	Extend		WriteALU	Read rt		Fetch

Figura 6.- Tabla que recoge las microinstrucciones correspondientes a las instrucciones de transferencia lw y sw.

En la tabla 7, se proponen las dos microinstrucciones que finalizan la ejecución de las instrucciones lógico-aritméticas, de tipo R.

µInstrcción rotulo o etq.	Contol ALU	SRC1	SRC2	Destino ALU	Memoria	Registro Memoria	Control PCWrite	Secuencia-miento
Rformat 1	Fun code	rs	rt					Seq
	Fun code	rs	rt	Rd				Fetch

Tabla 7.- Tabla con los valores que toman los campos de las microinstrucciones que terminan la ejecución de las instrucciones lógico-aritméticas o de tipo R.

En la siguiente tabla se presentan los campos de todas las microinstrucciones que sirven para implementar el repertorio básico, que son 10 microinstrucciones, tantas como estados (0-9).

µInstrcción rotulo o etq.	Contol ALU	SRC1	SRC2	Destino ALU	Memoria	Registro Memoria	Control PCWrite	Secuencia-miento
Fetch	Add	PC	4		ReadPC	IR	ALU	Seq
	Add	PC	Extshf	Target				Dispatch 1
LSW1	Add	rs	Extend					Dispatch 2
LW2	Add	rs	Extend		ReadALU			Seq
	Add	rs	Extend		ReadALU	Write rt		Fetch
SW2	Add	rs	Extend		WriteALU	Read rt		Fetch
Rformat 1	Codefunc	rs	rt					Seq
	Codefunc	rs	rt	Rd				Fetch
BEQ 1	Sub	rs	rt				Target-cond	Fetch
JUMP 1							Jump adder.	Fetch

Tabla 8 - Representación de los valores que toman los diferentes campos de las 10 microinstrucciones básicas.

La siguiente figura muestra el esquema general de la Unidad de Control Microprogramaza del MIPS multiciclo. EL bloque principal es la Memoria de Control que estará formada por 10 posiciones con los bits precisos para contener todos los campos de las instrucciones. También existe una lógica auxiliar para configurar el Secuenciador , disponiendo como entradas los dos bits que eligen una de las tres alternativas del campo Secuenciamiento, y los 6 bits del código OP.

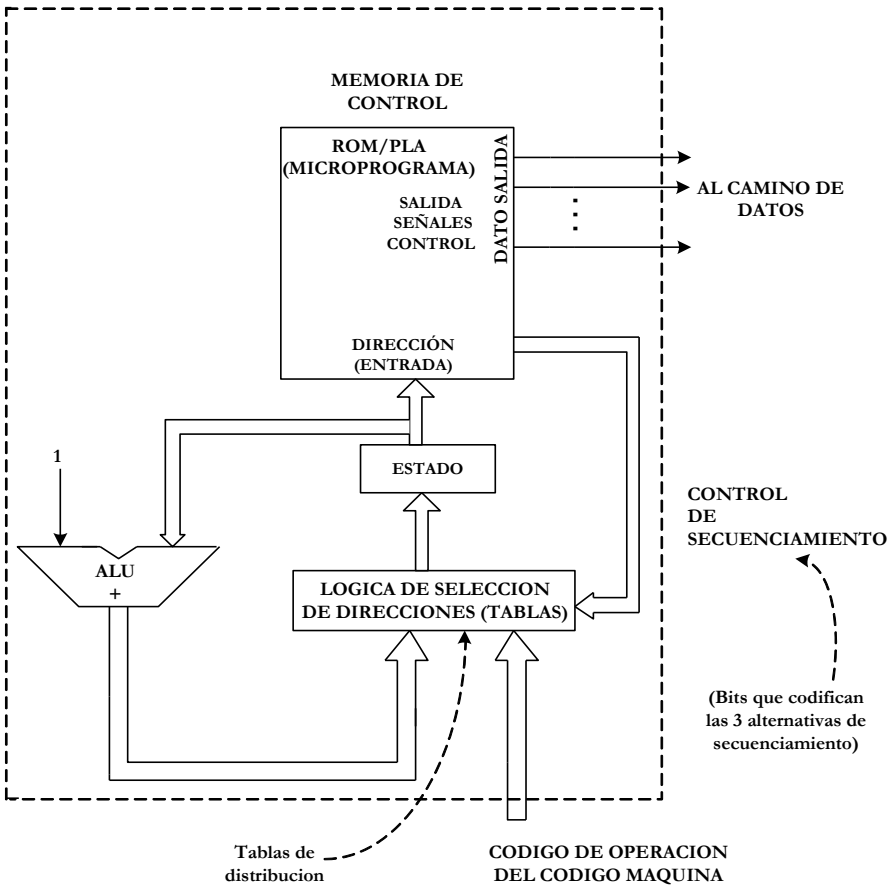


Figura 32.- Esquema general de la Unidad de Control microprogramada del MIPS multiciclo.

7.4- Excepciones e interrupciones

Se llama **excepción** a un acontecimiento inesperado o anómalo del procesador (por ejemplo: Desbordamiento, Uso del código no definido de instrucción, etc), que origina una ruptura automática del flujo de control.

Una interrupción es una excepción provocada desde el exterior al procesador. Para soportar las interrupciones el procesador dispone de patitas que detectan las peticiones externas cuando son activadas por el hardware externo.

Por lo tanto la Unidad de Control debe ser capaz de detectar las situaciones especiales que causan las excepciones y producir una bifurcación del control a la rutina que resuelve la anomalía. Después de resolverla, hay que regresar al punto de partida del programa principal.

La unidad de control del MIPS es capaz de detectar dos excepciones:

1ª : Código OP de instrucción no definido.

2ª : Desbordamiento de una operación de la ALU.

Cuando hay varias causas que provocan excepciones, existen también diferentes maneras de atenderlas. A dichos procedimientos se accede de dos maneras:

a) Mediante interrupciones vectorizadas (80x86)

Se trata de hacer corresponder a cada excepción una dirección concreta de la memoria donde debe comenzar la rutina que la atienda.

b) Mediante un registro de estado cuyos bits especifican el tipo de excepción (MIPS)

En el MIPS hay un registro, llamado **CAUSE**, cuyo último bit determina una de las dos posibles causas de excepción. Si vale 1, significa que se trata de un código OP no definido y si vale 0, de un desbordamiento. Figura 33.

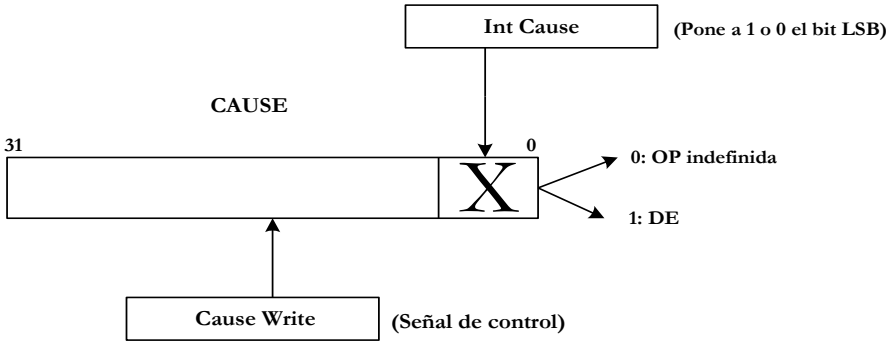


Figura 33.- El último bit del registro CAUSE determina el tipo de excepción que ha sucedido en el MIPS.

Una vez atendida la excepción se debe retornar al programa principal, para ello se necesita un registro auxiliar para guardar el valor de PC, este registro es el EPC. Figura 34.

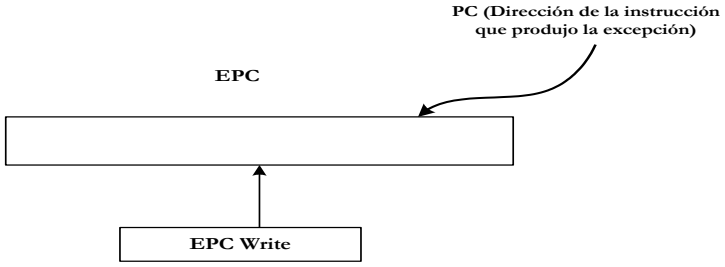


Figura 34.- Registro EPC del MIPS.

En el MIPS, cada vez que se produce una excepción, se salta a una rutina común que comienza en la dirección que se carga en el PC cuyo valor **es 0100 0000 0000 0000 0000 0000 0000 0000**. Luego la propia rutina leyendo el bit de menos peso de CAUSE determina cual de las dos posibles excepciones ha sucedido y atiende en consecuencia.

El multiplexor de 4 entradas que cargaba el PC, tenía una entrada libre que se emplea para dejar pasar con la señal de control PCSource = 11, el valor del PC indicado en el párrafo anterior que indica la rutina de atención a la excepción.

Además el registro EPC se debe cargar con el valor del PC donde ha sucedido la excepción y no con el del PC + 4 obtenido del PC desde la fase de búsqueda de la instrucción. Para conseguir dicho valor la ALU debe restar 4 al valor de PC + 4. El resultado obtenido se carga en el registro EPC.

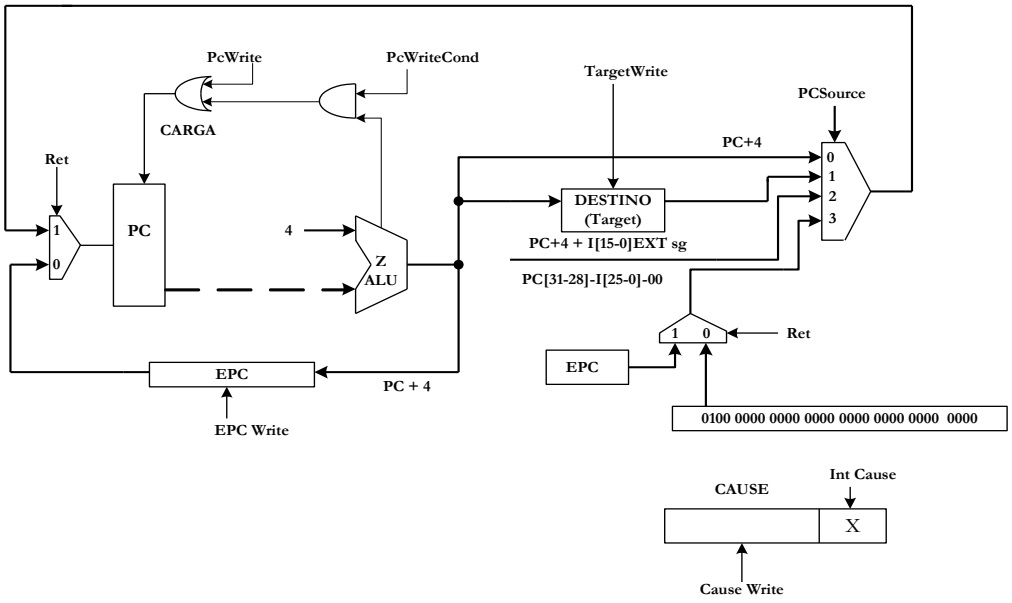


Figura 35.- Esquema de los elementos complementarios que hay que añadir al MIPS multiciclo para que sea capaz de soportar el tratamiento de las excepciones.

En la figura 36 se muestra el grafo de estados del MIPS multiciclo, con capacidad para el tratamiento de las dos excepciones, la del código OP no definido y la del desbordamiento. La primera se detecta después del estado 1, cuando se comprueba que el código OP no es ninguno de los admitidos. En cuanto a la segunda, ésta se detecta una vez acabado el estado 7, ultimo de las instrucciones lógicas y aritméticas, que es cuando se reconoce si se ha producido desbordamiento.

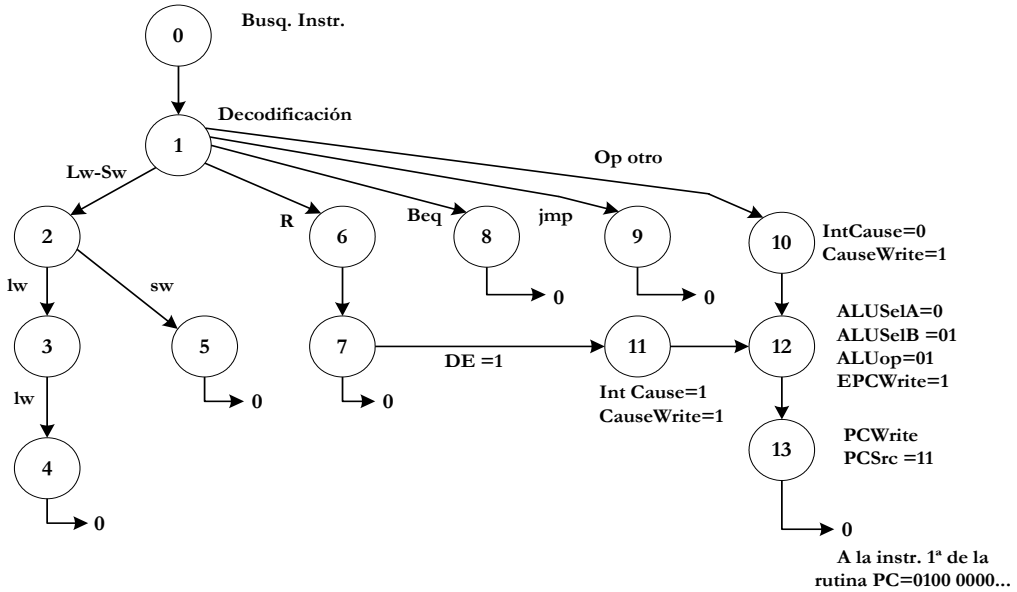


Figura 36.-Grafo de estados del MIPS multiciclo con posibilidad de tratar tanto la excepción de código OP no definido y la de desbordamiento de la ALU.

8- EL MIPS SEGMENTADO

En la actualidad la mayoría de los procesadores alcanzan rendimientos espectaculares debido a tres aspectos fundamentales:

1º. Funcionan a una frecuencia superior a los 100 Mhz.

2º. Disponen millones de transistores, que les permiten implementar:

a) Una potente y sofisticada arquitectura dotada de todos los recursos de los grandes procesadores.

b) Una memoria caché de gran capacidad y varios niveles jerarquicos.

3º. Se integran en el mismo procesador las dos técnicas más representativas y eficaces para elevar la potencia de procesamiento:

a)SUPERSEGMENTACIÓN

Se consigue un número muy elevado en las etapas de segmentación, así por ejemplo el Pentium Pro alcanza un nivel 16.

b)SUPERESCALAR

Disponen de más de un cauce segmentado y son capaces de iniciar más de una instrucción en cada ciclo de reloj.

La segmentación fue planteada por primera vez al comienzo de la década de los años 60 en el procesador IBM 703. Hitos importantes de esta técnica fueron los procesadores CDC 6600 en 1964 y el IBM 360/91, en 1966.

Los compiladores deben ser capaces de evitar introducir al mismo tiempo a los cauces de los procesadores superescalares pares o conjuntos de instrucciones que sean dependientes entre sí.

Los compiladores deben deshacer las dependencias que surgen entre instrucciones consecutivas de un programa, cuando se introducen a un cauce segmentado. Si se trata de una instrucción de bifurcación condicional (beq), la siguiente instrucción deberá introducirse al cauce cuando se haya resuelto la condición de bifurcación y la instrucción de salto se haya completado.

8.1- La técnica de la segmentación

Mediante la segmentación se pueden ejecutar varias instrucciones a la vez en un mismo procesador multiciclo, con lo que se pretende que cada instrucción utilice en cada ciclo un recurso diferente, consiguiendo un cierto paralelismo, que recibe el nombre de **paralelismo implícito**.

Todos los procesadores modernos utilizan la segmentación de forma exclusiva.

Una línea o cauce bien equilibrado es el que tiene sus etapas de la misma duración, por lo que cada ciclo o tiempo empleado en la operación de cada célula, una instrucción pasa a la célula siguiente y de la última sale una acabada. Estas células reciben el nombre de **etapas de segmentación** o **etapas** en el procesador.

Con la segmentación no se reduce el tiempo, pero se mejora la productividad.

Si las etapas están perfectamente equilibradas, es decir, duran lo mismo:

$$\text{Tiempo tarda en salir Instrucción Segmentada (una vez este el cauce lleno)} = \frac{\text{Tiempo instrucción sin segmentar}}{\text{NUMERO DE ETAPAS}}$$

Por lo tanto la mejora de velocidad en la terminación de instrucciones es igual al número de etapas, si están bien equilibradas. En caso contrario, existe una pérdida de rendimiento al existir perdida de rendimiento en algunas.

En la práctica, las etapas nunca llegan a estar bien equilibradas.

EJEMPLO:

Analicemos el siguiente supuesto en el que la tecnología de fabricación utilizada en la construcción de MIPS proporciona los siguientes retardos significativos:

T Acceso a memoria = 80 ns

T Acceso a banco = 50 ns

T Retardo ALU = 60 ns

Comparemos el aumento de la velocidad en la ejecución de instrucciones del MIPS monociclo y del segmentado.

A continuación calculamos los tiempos que dura cada instrucción en el monociclo:

INSTRUCCIÓN	**T_{acc} M_{ins}**	**T_{acc} Banco Lectura**	**T ALU**	**T_{acc} M_{datos}**	**T_{acc} Banco Escritura**	**TOTAL**
Lw	80	50	60	80	50	320 ns
Sw	80	50	60	80		270 ns
R	80	50	60		50	240 ns
Beq	80	50	60			190 ns

Tabla 9.- Cálculo de la duración de las instrucciones en el MIPS monociclo.

El ciclo que se utiliza en este tipo de procesador el da la instrucción más lenta. Por lo tanto, el ciclo será de 320 ns.

En el MIPS segmentado, a cada etapa se le da la duración de la que más dura, que en nuestro caso será la de acceso a la memoria que son 80 ns. De este modo, las cinco etapas en las que se descompone la instrucción duran 80 ns cada una.

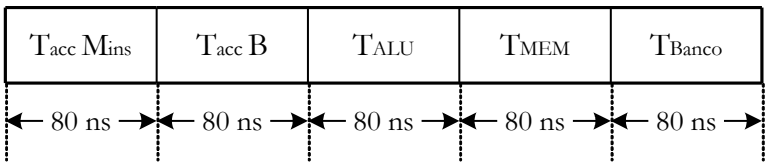


Figura 37.- Duración de los ciclos de una instrucción.

Una vez que se llena el cauce de las 5 etapas, transcurridos 80 ns sale de él una instrucción terminada. En algunas etapas, existe una pérdida de rendimiento, debido a que algunas se podían haber realizado en menos tiempo.

Por lo tanto, el aumento de velocidad entre un MIPS monociclo y uno segmentado es de: 320 / 80 = 4. Es decir el resultado sería 4, aunque el teórico sería 5, debido a que hay 5 cauces, esto es debido al desequilibrio de tiempos entre las 5 etapas.

8.2- MIPS con camino de datos segmentado

Para determinar las etapas del cauce segmentado se parte del MIPS monociclo, y de los pasos en los que se descomponen las instrucciones. Obteniéndose de este modo 5 etapas:

1ª: IF : Búsqueda de la instrucción.

2ª: ID : Decodificación y Búsqueda de operandos.

3ª: EX : Ejecución y Cálculo Dirección Efectiva.

4ª: MEM : Acceso a Memoria de Datos.

5ª: WB : Postescritura.

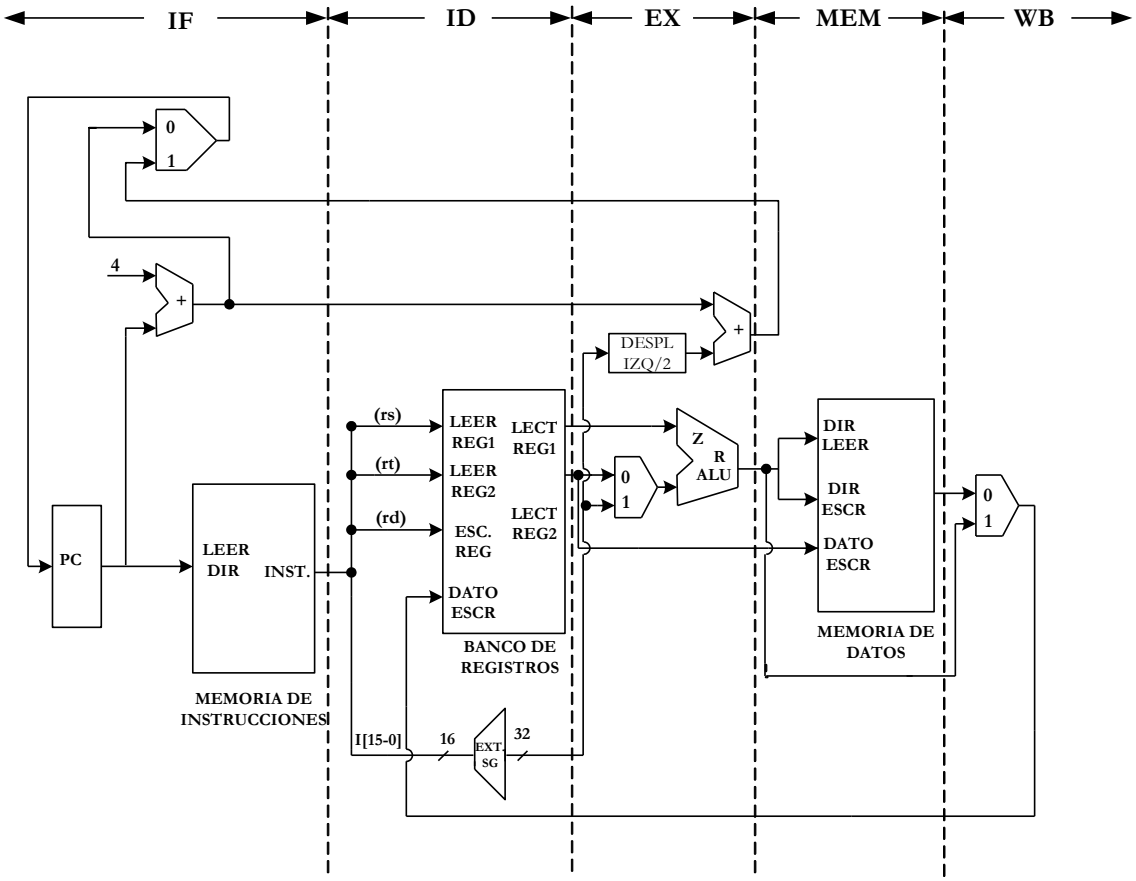


Figura 38.- Cauce sementado del Camino de Datos del MIPS

El flujo de datos e instrucciones va recorriendo las 5 etapas de izquierda a derecha, aunque hay dos excepciones:

- En la etapa MEM existe una realimentación a la etapa anterior IF con el valor de salto que se carga en PC para la instrucción beq.
- En la etapa de Postescritura, el resultado que genera se escribe en la etapa anterior ID, en el Banco de Registros.

Para poder compartir un mismo recurso por varias instrucciones se colocan registros intermedios entre las etapas, que almacenan el resultado que produce cada una.

8.3- Representación gráfica de la segmentación

Mediante este gráfico se pretende mostrar la ejecución de varias instrucciones, como si cada una tuviese su propio camino.

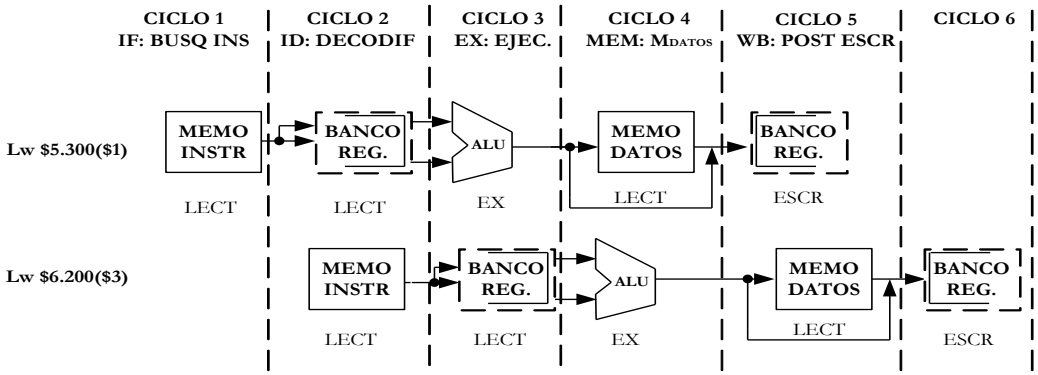


Figura 39.- Representación gráfica de la segmentación que muestra la ejecución de varias instrucciones.

En cada ciclo de reloj sólo se utiliza un recurso. El **Banco de Registros** se descompone en dos módulos independientes, uno se encargará de la lectura de una pareja de registros y el otro de la escritura del que se especifica por la entrada correspondiente. Para su correcta representación, se marca con línea gruesa la mitad derecha del banco de registros cuando se lee (la izquierda a trazos), y cuando se escribe se intercambia la representación de las dos mitades.

La memoria de instrucciones solo se usa en la fase IF después queda libre pudiendo ser accedida por otras instrucciones, pero para ello es necesario almacenar en in registro (RI) el código leído.

Las instrucciones avanzan en su ejecución de una etapa a la siguiente en cada ciclo, pasando la información que les corresponde desde un registro de segmentación al siguiente.

Los registros deben de ser lo suficientemente grandes para almacenar toda la información que genera cada etapa y que hay que pasar a la siguiente.

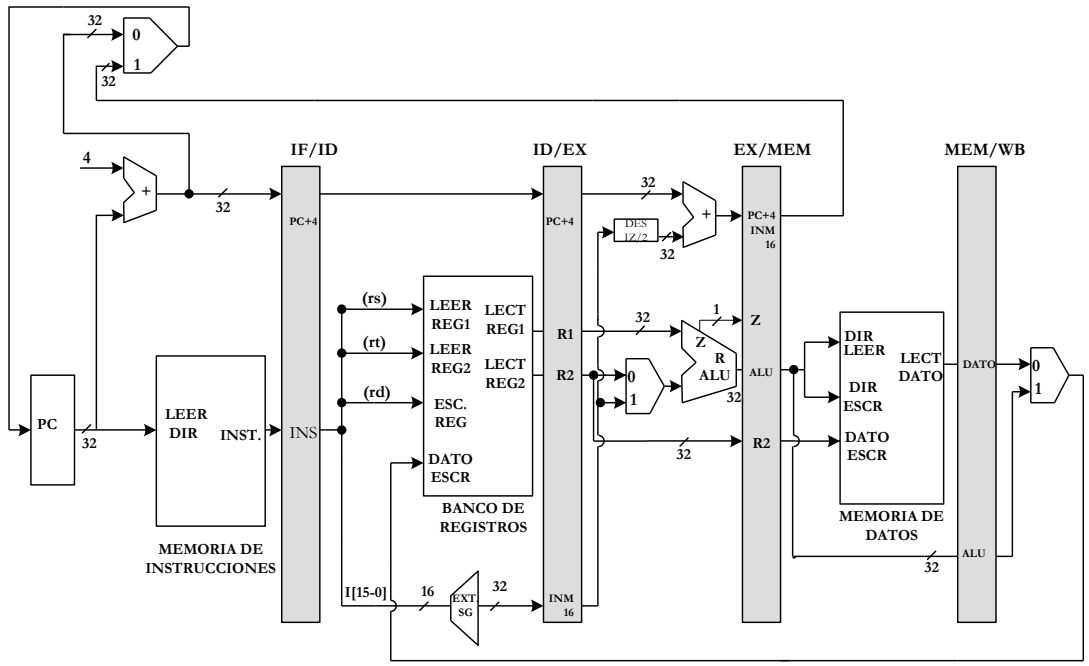


Figura 40.- Los registros de segmentación existentes entre etapas, sirven para almacenar información correspondiente a la instrucción que utiliza esa etapa, proporcionándosela a la siguiente.

Para saber que información debe almacenarse, inicialmente se hace el estudio de la segmentación referida a la arquitectura del procesador monociclo

- Primera etapa **IF**

  Operaciones elementales de esta etapa:

  M(PC)= INS ; PC + 4= PC

  En el registro IF/ID se guardan 64 bits, 32 del PC y los restantes 32 del código INS.

- Segunda etapa **ID**

  En el MIPS monociclo, se lleva a cabo la **decodificación**, la lectura automática de los registros R1 y R2 y la extensión de signo de valor INMEDIATO.

  En ID/EX se guardan 128 bits : (PC +4), (R1), (R2), y (INM32).

- Tercera etapa **EX**

  La ALU realiza la suma de R1 (rs) con INM32 para calcular la dirección de acceso a la **memoria de datos**.

  EN el registro EX/MEM se guardan 97 bits: ALU_{out}, Z, [PC + 4 + INM32] y R2.

- Cuarta etapa **MEM**

  Se lee la memoria de datos con la dirección que sale por la ALU.

  En el registro MEM/WB se guardan 64 bits: DATO32 y ALU_{out}.

- Quinta etapa **WB**

  El dato leído en la memoria de datos se escribe en el registro R3 (rt). En esta última etapa no hay registro de segmentación, puesto que en esta etapa se escribe en el Banco, bien el dato de la memoria o bien el ALU_{out}.

En esta última etapa surge un problema: hay que escribir el dato en R3 y en consecuencia , se debería haber guardado el valor de dicho registro que venía en el código de INS y se obtenía desde la etapa ID, a través de las distintas etapas hasta la WB.

Para resolver este problema, el valor de R3 (rt), que se obtiene a la salida de ID, no se aplica al Banco de Registros, sino que se guarda en el registro ID/EX y se va pasando hasta la etapa WB, donde se usa para especificar donde hay que escribir. Este procedimiento también se puede realizar en todas las instrucciones aritmético-lógicas.

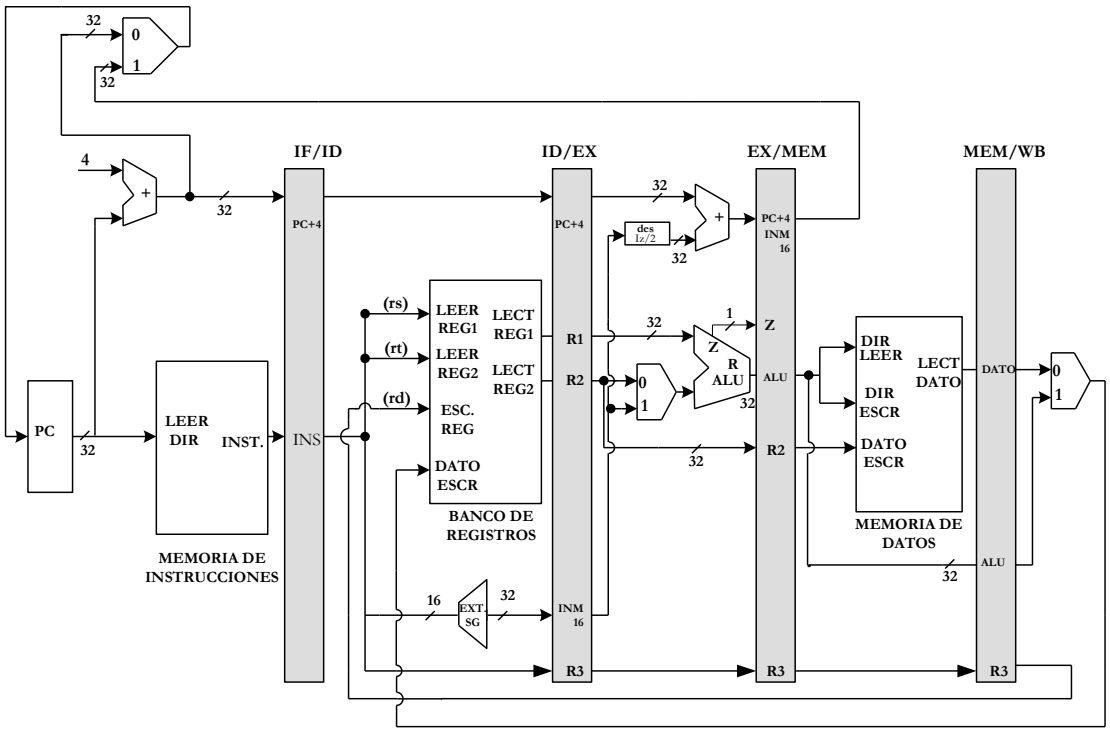


Figura 41.- El valor del registro donde hay que escribir, se va traspasando registro a registro, hasta la etapa WB.

En la instrucción de almacenamiento (sw), en la etapa EX se usa el registro R2 para pasarlo a la etapa MEM para ser escrito en la dirección que se obtiene por ALU_{out}. Después de escribir la memoria de datos no pasa nada más.

8.4- Unidad de control segmentada

Se trata de definir el valor lógico de las líneas que controlan los recursos del Camino de Datos segmentado y calcular sus valores para cada ciclo de reloj de cada instrucción.

Las señales de control de cada etapa son:

- Etapa **IF :** Las señales de control para leer la memoria de instrucciones y para escribir el PC están activadas siempre, así que no hay control especial en esta etapa.
- Etapa **ID :** Ocurre lo mismo que en la etapa anterior, por lo que no hay líneas de control opcionales que inicializar.

- Etapa **EX :** Las señales que se van a inicializar son **RegDst**, **ALUop** y **ALUSrc**. Las señales seleccionan el registro Result, la operación de la ALU, y/o un registro o un inmediato con signo extendido para la ALU.
- Etapa **MEM :** Las líneas de control que se inicializan son **Branch**, **MemRead** y **MemWrite**. Estas señales las inicializan las instrucciones saltar sobre igual, cargar y almacenar, respectivamente.
- Etapa **WB :** Las dos líneas de control son **MemtoReg**, que decide entre enviar el resultado de la ALU o el valor de memoria a los registros, y **RegWrite**, que escribe el valor escgido.

En la siguiente tabla se muestran las diversas señales de control que regulan el comportamiento del Camino de Datos.

Instrucción	ETAPA EX				ETAPA MEM			ETAPA WB	
	Reg Dest	ALU op1	op op0	ALU Src	Branch	Mem Read	Mem Write	Reg Write	Mem to Reg
TIPO R	1	1	0	0	0	0	0	1	0
lw	0	0	0	1	0	1	0	1	1
sw	X	0	0	1	0	0	1	0	X
beq	X	0	1	0	1	0	0	0	X

Tabla 10.- Tabla que muestra las señales de control que regulan el funcionamiento del Camino de Datos.

A continuación se muestra el Camino de Datos segmentado con sus correspondientes señales de control.

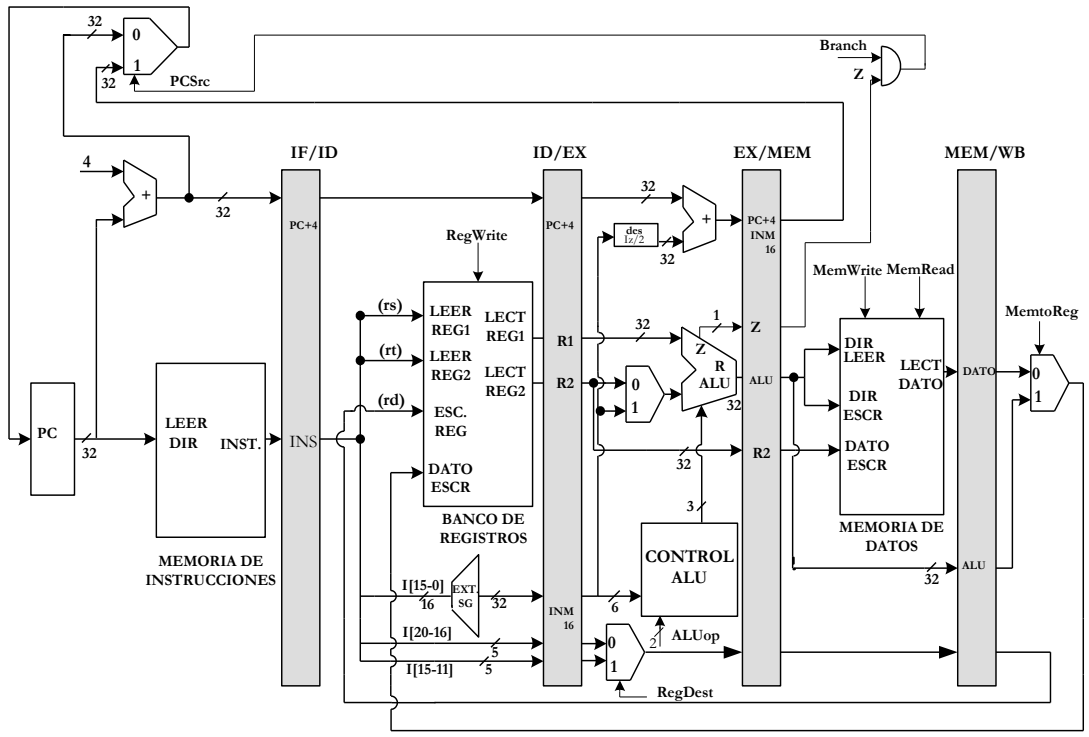


Figura 42.- Esquema del Camino de Datos segmentado con sus señales de control.

Existen 9 señales de control, los cuales sólo se emplean en las tres últimas etapas de segmentación. Tras la segunda etapa ID se calculan para cada instrucción el valor de las 9 señales de control y se van traspasando por los registros interetapas , hasta alcanzar cada señal de control la etapa donde debe actuar.

8.5- Los conflictos en la segmentación y su incidencia en el rendimiento del procesador

Se denominan **conflictos** y también **riesgos** ("hazards") a las causas o situaciones que impiden introducir instrucciones al cauce segmentado. Se puede interrumpir la entrada de interrupciones al cauce durante uno o varios ciclos de reloj, ciclos vacíos en contenido que reciben el nombre de **burbujas** o **ciclos de detección**.

Por ello, al detenerse la entrada de instrucciones en el cauce se disminuye notoriamente el rendimiento del procesador.

Debido a los conflictos, el CPI$_{IDEAL}$ con el que se parte teóricamente se convierten el CPI$_{IRREAL}$, de mayor valor que el primero al incrementarse a causa de los ciclos de detención del cauce. Este CPI$_{IDEAL}$ se obtiene al dividir el CPI sin segmentación por el número de etapas de segmentación, que se denomina **profundidad**.

$$\mathbf{CPI}_{IDEAL} = \frac{\text{CPI SIN SEGMENTACIÓN}}{\text{PROFUNDIDAD}}$$

CPI$_{SS}$ = CPI$_{IDEAL}$ x PROFUNDIDAD

De donde la aceleración con segmentación,

ACELERACIÓN$_{segm}$= Tiempo de una instrucción sin segm / Tiempo de una instrucción con segm

CIP$_{SS}$ x Ciclo reloj$_{ss}$ = CPI$_{IDEAL}$ x PROFUNDIDAD x CICLOS$_{SS}$
CIP$_{CS}$ x Ciclo reloj$_{cs}$ = CPICS x CICLOS$_{CS}$

Al tenerse en cuenta los ciclos de reloj de detención que se producen por promedio por cada instrucción a causa de los conflictos, la aceleración con segmentación queda como:

CPI$_{CS}$ = CPI$_{IDEAL}$ + Ciclos$_{Detención\ instrucción}$

ACELERACIÓN$_{segm}$= Ciclo reloj$_{ss}$ x CPI$_{IDEAL}$ x PROFUNDIDAD / Ciclo reloj$_{cs}$ x CPI$_{IDEAL}$ + Ciclos Detención

Debido a los gastos de la segmentación el ciclo de reloj con segmentación llega a ser muy parecido al ciclo sin segmentación, por lo que el primer término de la última fórmula se supone igual a la unidad, quedando, la expresión simplificada como:

ACELERACIÓN$_{segm}$= CPI$_{IDEAL}$ x PROFUNDIDAD / CPI$_{IDEAL}$ + Ciclos Detención

8.5.1- Tipos de conflictos

Se pueden producir cuatro tipos de conflictos según las causas que los provocan:

1º.Conflictos estructurales: Se producen cuando el cauce segmentado soporta varias instrucciones a la vez y algunas de ellas necesitan utilizar a la vez un mismo recurso del Camino de Datos. También se les suele denominar **colisiones**.

Una colisión se produce, por ejemplo, cuando en el mismo ciclo de reloj una etapa de una instrucción tiene que acceder a la memoria para buscar la instrucción y otra al mismo tiempo debe acceder a la memoria para leer o escribir un dato.

Este conflicto se resuelve dotando al procesador de dos memorias tipo caché, independientes, una para datos y otra para instrucciones.

2º.Conflictos por dependencia de datos: Se producen cuando una instrucción utiliza como operando el resultado de una instrucción previa que se está ejecutando en el cauce y todavía no ha producido el valor del resultado:

```
add  $7, $1, $2
or   $3, $7, $4
```

En este caso cuando la instrucción se encuentra en la segunda etapa del cauce (ID), debe proceder a leer el valor de uno de los operandos que es el registro $7, pero como la instrucción anterior se encuentra en la etapa de ejecución todavía no se actualizado su valor, por ello la instrucción or deberá esperar tres ciclos, que den tiempo a la instrucción add a escribir el resultado en el registro $7.

3º.Conflictos por saltos condicionales: Hasta que no se complete la instrucción condicional (beq), no puede introducirse otra instrucción en el cauce, ya que no se sabe cual será la siguiente.

Para solucionar este tipo de conflictos, se emplean métodos de predicción de salto.

4º.Conflictos por excepciones e interrupciones: Cuando se produce una interrupción o una excepción durante la ejecución de una instrucción de un programa se genera un salto a una rutina de atención, debiéndose eliminar las instrucciones del programa que se hayan introducido al cauce después de la que origina la excepción o interrupción.

8.6- Conflictos estructurales

Para poder introducir al cauce cualquier combinación posible de instrucciones hay que duplicar los recursos, ya que sino dos instrucciones pueden requerir un mismo recurso, produciéndose así un conflicto.

Una solución para los conflictos estructurales sería duplicar la memoria para poder realizar así accesos simultáneos a datos e instrucciones.

En nuestro caso si solo existiese una memoria para datos e instrucciones, cada vez que se introdujese en el cauce una instrucción de transferencia, habría que impedir la entrada de una nueva instrucción en el cauce en el ciclo en el que las dos accediesen simultáneamente a memoria a leer instrucciones o a leer o escribir datos. Por lo que habría que añadir un ciclo de reloj para que se pudiesen realizar correctamente estas instrucciones.

Otro recurso a tener en cuenta sería el Banco de Registros , que solo tiene un puerto de escritura y que podría dar lugar a un conflicto si sucediese que dos instrucciones del cauce tendrían que escribir en el mismo ciclo en el banco. Para evitar esta posibilidad es necesario seleccionar cuidadosamente el repertorio de instrucciones y su división en etapas.

Como la duplicación aumenta el coste, se buscan otras soluciones, como la segmentación o división en etapas de recursos, como es el caso de la ALU. Se pueden tomar dos medidas, una añadir dos sumadores, lo que aumentaría el coste u otra el segmentar o dividir en etapas la ALU con lo que se evitaría el aumento de coste, pero aumentaría la profundidad, lo que puede provocar una disminución del rendimiento.

8.7- Conflictos por dependencia de datos

Se produce cuando la segmentación cambia el orden de acceso a los operandos, con relación al que sigue a la secuencia normal de instrucciones. Ejemplo:

```
add $7, $1, $2
or $3, $7, $4
and $5, $6, $7
add $8, $7, $7
lw $9, 8($7)
```

Todas las instrucciones que siguen a la primera del programa utilizan como operando el contenido del registro $7, que es el resultado de la primera add. Por lo tanto hasta que no se termine la primera instrucción, el valor de dicho registro no se actualizará y las instrucciones que lean este valor y que se realicen antes de que se actualice dicho registro leerán un valor erróneo.

En la siguiente figura mostramos un ejemplo:

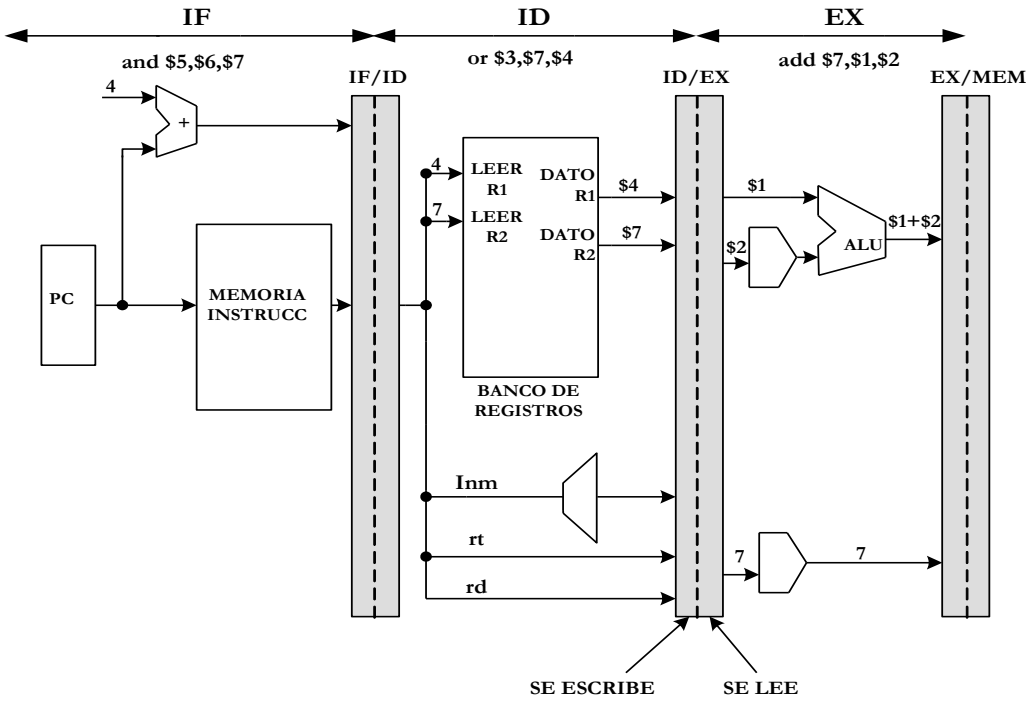


Figura 43.- La instrucción or se halla en la etapa ID y procede a leer sus dos operandos que son el registro $7 y $4. La instrucción anterior se halla en la etapa EX y debe actualizar el valor de $7, pero aún debe pasar por las etapas MEM y WB para hacerlo.

Del estudio del esquema de la figura 43, se deduce que va a producirse un conflicto por dependencia de datos en el cauce, ya que mientras se está generando la suma $1+$2, a la vez que se está determinando que el registro destino rd, será el $7, que se halla aún sin actualizar.

Este conflicto se representa de la siguiente manera:

ID/EX.Escribir Registro = IF/ID.Leer Registro1=$7

A continuación se analizan los distintos conflictos que surgen con las instrucciones del ejemplo:

1.-Conflicto add-or

Se detecta cuando add está en la etapa EX y la or en la ID donde el registro causante del conflicto es el $7, que tiene que ser escrito por add y leido por or.

ID/EX.Escribir Registro=IF/ID.Leer Registro1 = $7

2.-Conflicto add-and

Se detecta cuando add está en la etapa MEM y la and en ID.

EX/MEM.Escribir Registro = IF/ID.Leer Registro2 = $7

3.-Conflicto add-add (primero)

La primera add está en la etapa WB, y la segunda se encuentra en la etapa ID.

MEM/WB.Escribir Registro = IF/ID.Leer Registro1 = $7

4.-Conflicto add-add (segundo)

El conflicto se debe a que la segunda utiliza el registro $7 para sus dos operandos.

MEM/WB.escribir Registro = IF/ID.Leer Registro2 = $7

No existe conflicto entre la primera instrucción y la última, ya que para cuando entra en el cauce la última instrucción la primera ya ha acabado.

8.7.1- Detección y eliminación de las dependencias de datos

Para eliminar estos conflictos distinguimos dos alternativas. Una emplea un hardware auxiliar del procesador, mientras que la otra lo realiza mediante software, concretamente mediante el compilador.

La solución hardware consiste en diseñar una lógica combinacional capaz de detectar los conflictos de dependencias de datos, a esta lógica se le llama Unidad de detección de conflictos.

Un conflicto de dependencia de datos ocurre cuando se cumplen estas condiciones:

1. La señal Reg. Write = 1, en las etapas EX, MEM ó WB. Esto significa que una instrucción previa del cauce tiene que escribir un registro y aún no lo ha hecho.
2. Que el registro o registros que hay que leer en la etapa ID coinciden con el que hay que escribir en cualquiera de las etapas siguientes.

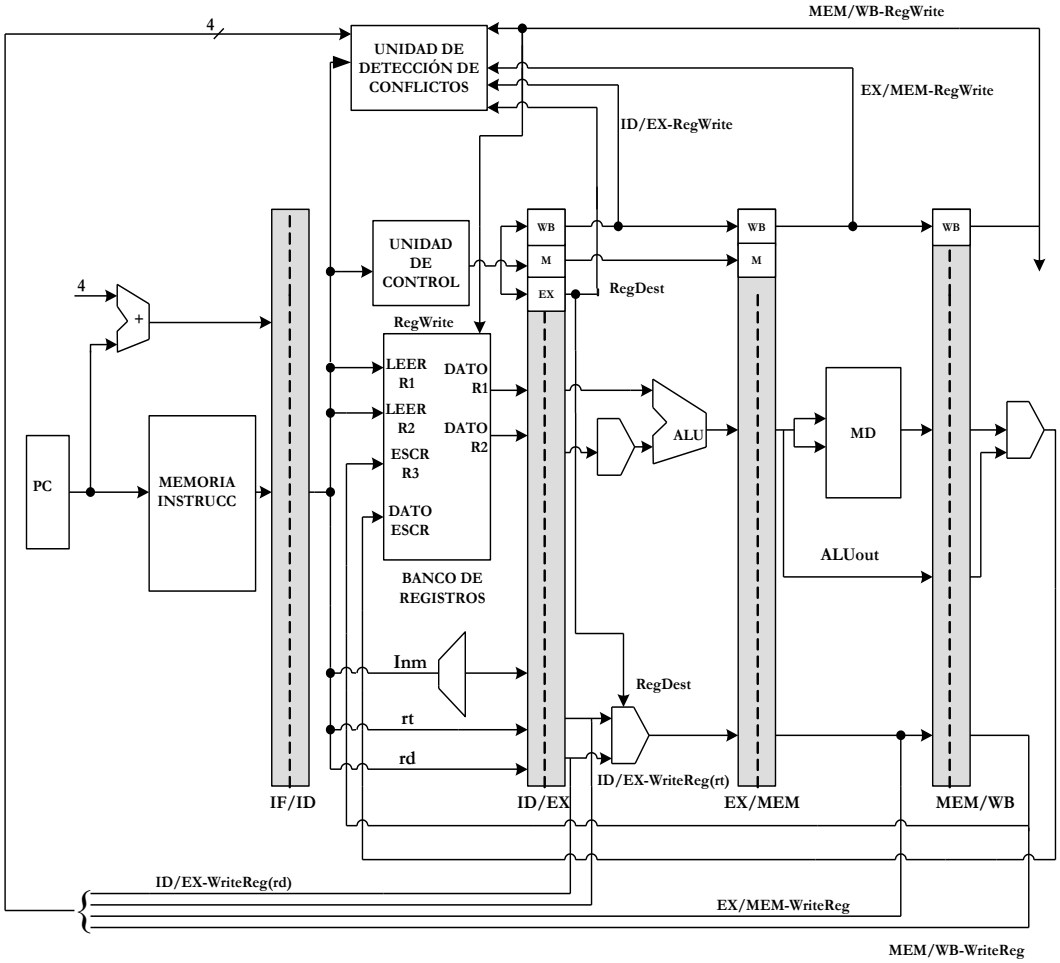


Figura 44.- Representación de las líneas principales que determinan cuando se debe escribir un registro y su selección mediante la señal de control Reg.Dst.

La **Unidad de detección de conflictos** de la figura 44, considera las señales de entrada:

- Señales Reg.Write de las etapas EX, MEM y WB.
- Señal Reg.Dst de la etapa EX.
- Señal Esch.Reg de las etapas EX, MEM y WB.
- Señales de lectura de registros R1 y R2 etapa ID.

La Unidad de Detección de Conflictos produce una salida **S = 1**, si es que se ha producido un conflicto por dependencia de datos, si no es así, tomara el valor **S = 0**.

Si se produce conflicto, se debe detener el traspaso de instrucciones de las etapas IF e ID, hasta que se solucione el conflicto. Esto se puede realizar mediante un compilador introduciendo instrucciones NOP en el cauce.

Por hardware también se pude conseguir, si la Unidad de Control genera como valor de sus nueve señales de control un "cero"(desconexión). De este modo no se modifica el contenido de PC ni de IF/ID.

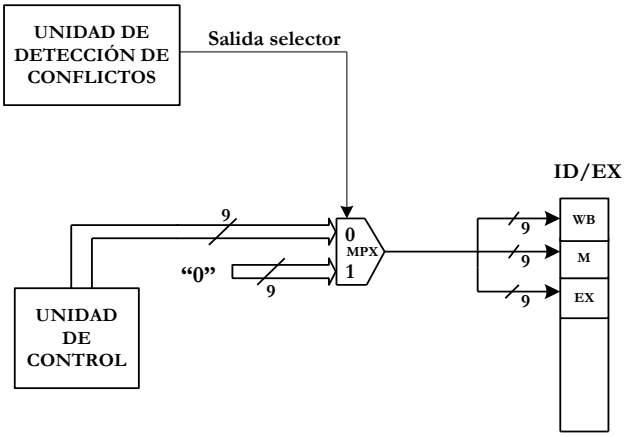


Figura 45.- Cuando la Unidad de Detección de Conflictos detecta uno y genera una salida de nivel alto, las nueve señales de control se transforman en ceros mediante el multiplexor.

Tanto por software como por hardware se consigue evitar el conflicto introduciendo burbujas en el cauce, lo que degrada el rendimiento del procesador.

8.7.2- Técnica de anticipación para reducir los conflictos por dependencias de datos en registros

También se le denomina **desvío** o "bypassing". Consiste en almacenar temporalmente los resultados de la ALU en un par de registros e implementar una lógica de desvío que examine si el resultado producido por la ALU y que más tarde se escribirá en un registro, se usa como operando en alguna de las instrucciones posteriores, si es así se introduce el resultado de la ALU, mediante un multiplexor a una de las entradas de la ALU, evitando la introducción de burbujas. Figura 46.

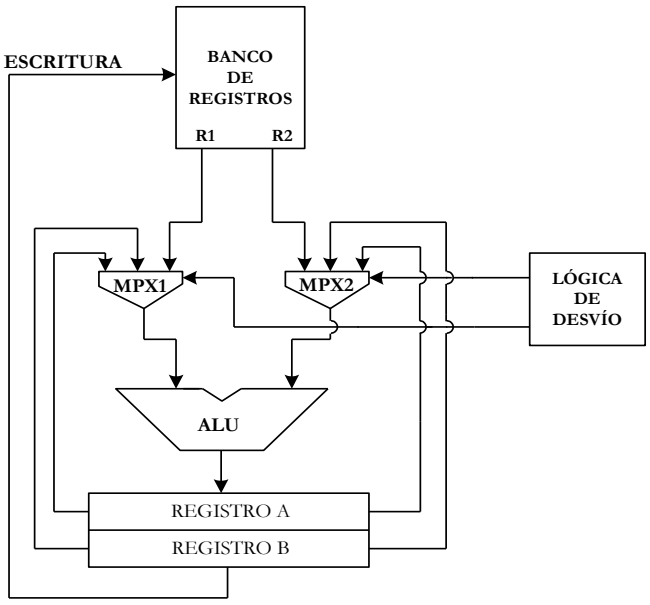


Figura 46.- Funcionamiento de la técnica de anticipación.

Para implementar la técnica de la anticipación en el MIPS se realizan las siguientes hipótesis:

Los resultados que hay que cargar en los registros afectados, se encuentran en los registros interetapa EX/MEM y MEM/WB.

- Para poder introducir en la ALU los resultados obtenidos en ella, se utilizan multiplexores que realimentan la salida con las entradas.
- Al control de estos multiplexores lo efectúa una **lógica auxiliar de desvío,** que analiza si los resultados van a ser leídos por instrucciones posteriores en el cauce.

La lógica de desvío compara los registros que actúan como operandos en la etapa ID, con los registros que actúan como destino. Si hay coincidencia, se introduce como entrada de la ALU, mediante multiplexores de realimentación, el valor de la salida de la ALU procedente del registro EX/MEM o del MEM/WB. Figura 47.

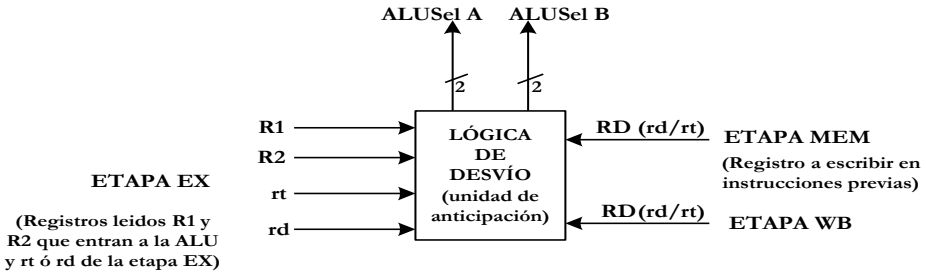


Figura 47.- La lógica de desvío analiza los registros R1 y R2 que se introducen como operandos a la ALU, con los valores de los posibles registros donde hay que escribir en las etapas MEM y WB.

Se pueden producir dos combinaciones de conflicto:

Si los registros que se introducen como operandos a la ALU en la etapa EX coinciden con alguno que hay que escribir en la etapa MEM

Conflicto EX

- Si EX/MEM.Reg Write

  y EX/MEM.Escribir Registro = ID/EX.Leer Registro1 ALUSelA=01
- Si EX/MEM.Reg Write

  y EX/MEM.Escribir Registro = ID/EX.Leer Registro2 ALUSelB=01

Conflicto MEM

- Si MEM/WB.Reg Write

  y MEM/WB.Escribir Registro = ID/EX.Leer Registro1 ALUSelA=01
- Si MEM/WB.Reg Write

  y MEM/WB.Escribir Registro = ID/EX.Leer Registro2 ALUSelB=01

No se considera el tercer conflicto WB, porque se supone que en la etapa ID se lee el resultado correcto si en dicho ciclo la instrucción en la etapa WB escribe alguno de los registros.

8.7.3- Técnica de anticipación para evitar conflictos por dependencias de datos en accesos a memoria

Estudiemos la siguiente secuencia de instrucciones:

lw $7, 50($2)

add $3, $7, $1

Durante el mismo ciclo de reloj add lee el valor del registro $7 que actúa como operando de la ALU, y la instrucción lw está accediendo a Memoria de Datos para leer el valor que tiene que cargar en el registro $7.

Como el dato que se va a cargar mediante la instrucción lw no va a estar disponible cuando lo necesita la add, la única solución para este conflicto es detener la segmentación. Por lo tanto habrá que introducir una burbuja para dar tiempo a disponer del dato que sale de la memoria.

Siempre que exista una instrucción lw seguida de otra que utilice como operando el registro que carga la lw, será preciso introducir una burbuja en el cauce.

Para detectar este tipo de conflicto se usa la Unidad de Detección de Conflictos, que deberá introducir nueve ceros como señales de control cuando detecte este conflicto, que se expresa de la siguiente manera:

Si ID/EX.Reg Write e ID/EX.RegDst = 0 e

ID/EX.Escribir Reg rt = IF/ID.Leer Registro1 ó

ID/EX.Escribir Reg rt = IF/ID.Leer Registro2

detener la segmentación

La figura 48 muestra la actuación de la Unidad de Detección de Conflictos.

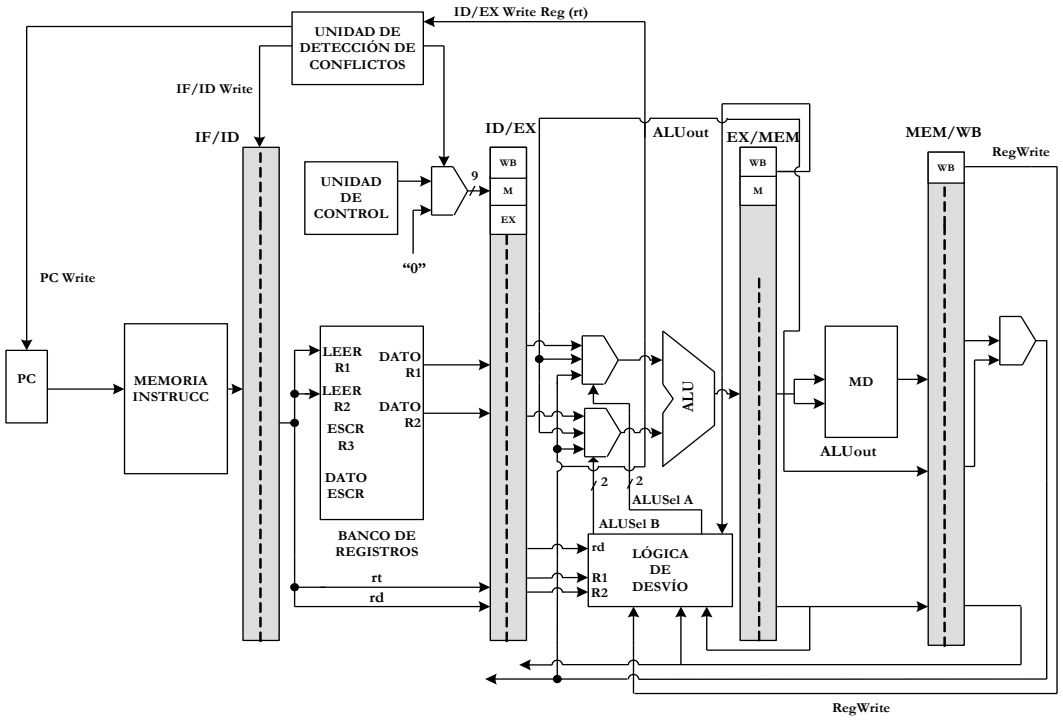


Figura 48.- Actuación de la Unidad de Detección de conflictos cuando trata los procedentes de las dependencias de datos con instrucciones de transferencia con memoria.

8.8- Conflictos de control por saltos condicionales

Los conflictos de control se producen cuando entran al cauce instrucciones de salto condicional (beq). Hasta que no sean completadas dichas instrucciones en la etapa MEM no se va a poder conocer la siguiente instrucción que ha de introducirse al cauce.

La técnica más sencilla y eficaz para solucionar este conflicto es introducir tres burbujas detrás de cada instrucción condicional. Aunque este procedimiento degrada mucho el rendimiento, debido a la gran cantidad de instrucciones condicionales que existen en los programas.

8.8.1- Técnicas para reducir los conflictos de control

Las técnicas suelen emplear conjuntamente un hardware específico y la actuación de un compilador, que deberá ser específico para el procesador segmentado que se utilice.

- Diseñar una lógica auxiliar específica que cuando detecta en la etapa ID una instrucción beq, calcule el valor que tomará el flan Z y halle el valor del PC correspondiente. Este hardware resulta muy caro.
- Diseñar un compilador específico para el procesador que analice las instrucciones y actúe en los saltos condicionales realizando alguna de las siguientes funciones:
    1. Introduciendo instrucciones NOP o burbujas al detectar las instrucciones condicionales. Esta solución disminuye el rendimiento.
    2. Eligiendo las instrucciones alternativas del programa, previas a la condicional, tales que no influyan en el desarrollo del salto. El rendimiento dependerá de las instrucciones que se introduzcan.
    3. Suponer que todos los saltos van a ser **no efectivos.** Se realiza un estudio de probabilidades. Cuando se falla en la predicción hay que deshacer todos los cambios que se han realizado. Para esta técnica se necesita un hardware adicional que lleve a cabo la **limpieza del cauce** ("flushing"). El MIPS emplea esta técnica.
    4. Suponer que todos los saltos van a ser **efectivos.** Esta solución es interesante en programas con instrucciones complejas.

8.8.2- Predicción de bifurcaciones

Consiste en suponer que no se va a realizar el salto y proseguir introduciendo instrucciones en el cauce.

Si se falla en la predicción habrá que deshacer todos los cambios que ha ocasionado las instrucciones posteriores a beq.

En el MIPS se añade un hardware auxiliar que cuando detecta que el salto es efectivo, limpia el cauce. La Unidad de Control comprueba si FZ = 1 para cada instrucción beq, en caso afirmativo genera una señal de **flush** a las etapas que se citan:

- **Etapa IF**: Envía una señal **IF. Flush** que pone a cero el campo de la instrucción del registro IF/ID.
- **Etapa ID**: La Unidad de Control genera una señal **ID. Flush** que realiza una operación OR con la señal de detección de conflictos. Ver figura 39.
- **Etapa EX**: Se genera una señal **EX. Flush** que a través de dos multiplexores pone a cero las señales de control que pasan a la siguiente etapa.

Cuando la Unidad de Control detecta que FZ = 0 para una instrucción beq que está en la etapa MEM, activa las señales **IF. Flush, ID. Flush, EX. Flush** que deshacen todo lo originado por las tres instrucciones introducidas en el cauce detrás de beq. Además carga en PC la dirección del salto para que la siguiente instrucción que se meta en el cauce sea correcta.

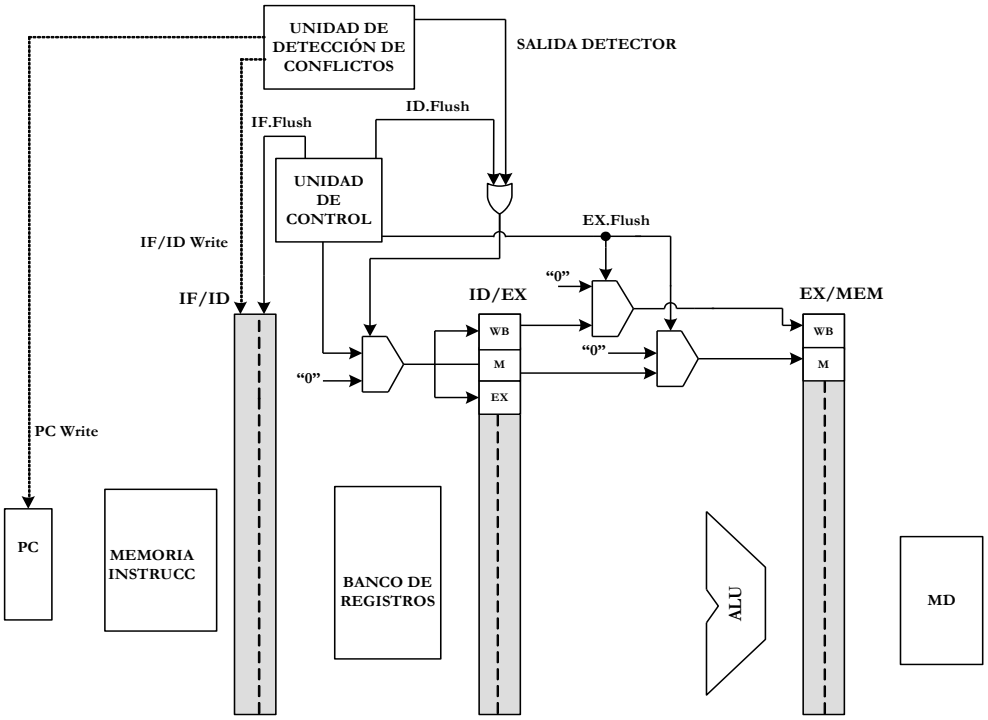


Figura 49.- Esquema del hardware necesario para la implementación de la técnica de predicción de bifurcaciones.

8.9- Conflictos por excepciones e interrupciones

Cuando se produce una excepción o interrupción sucede algo parecido a una instrucción de salto condicional. En el caso del MIPS se aceptan dos tipos de excepciones: Código OP no válido y desbordamiento en una instrucción aritmética. Las fases para el tratamiento de estas excepciones:

- Se detiene la instrucción en curso y se invalidan los resultados de las siguientes instrucciones introducidas al cauce.
- Se guarda la dirección del PC de la instrucción que ha provocado la excepción en EPC.
- Se salta q la rutina de tratamiento de la excepción, que comienza en la dirección 4000 0004 H. Aquí se analiza la causa (último bit del registro cause).

- Al finalizar la rutina de atención a la excepción se retorna a la instrucción que la produjo.

Para la desactivación de las señales de control en el tratamiento de las excepciones se utiliza la misma técnica que en las de salto condicional. Además se introduce desde un multiplexor al PC el valor 4000 0004 H. Figura 50.

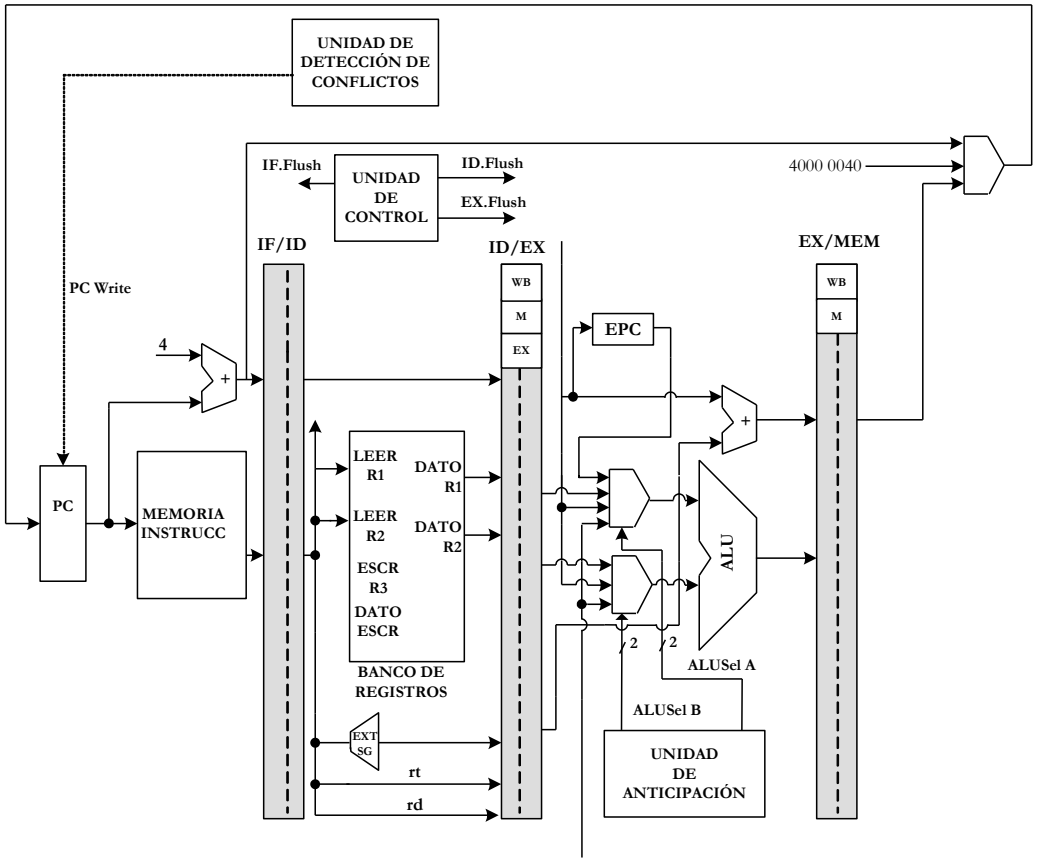


Figura 50.-Esquema de los elementos que actúan en el tratamiento de los conflictos por excepciones.

Si se trata de una excepción por desbordamiento se impide que se escriba el resultado de la ALU en la etapa WB para evitar que si un registro actúa como operando y destino, se borre su valor inicial y no se pueda conocer lo que ha ocurrido.

8.10- Incidencia de la profundidad de la segmentación en el rendimiento

El aumento de la profundidad en la segmentación no siempre incrementa el rendimiento del procesador.

Al aumentar la profundidad se incrementan las detenciones ocasionadas por los conflictos por dependencias de datos. También se elevan las burbujas necesarias para resolver conflictos de control. Por último, al elevar el número de etapas tiene mayor incidencia los tiempos de carga de los registros interetapas y el de los flancos de la señal de reloj.

En la siguiente figura se muestra un gráfico que relaciona la profundidad con el rendimiento relativo. Si el nivel de segmentación supera a 8, el rendimiento disminuye.

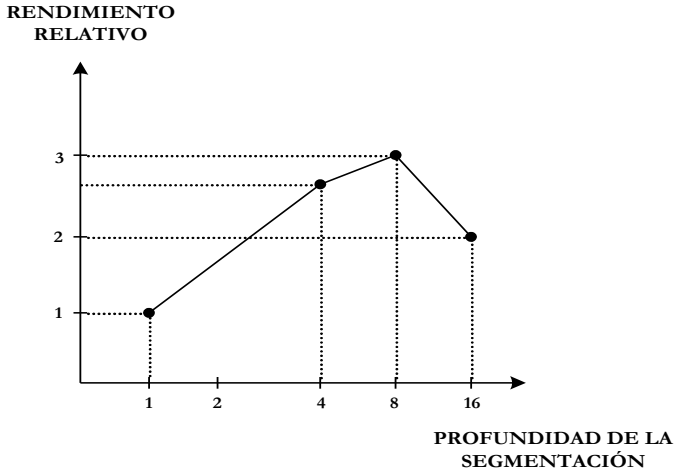


Figura 51.- Al aumentar la profundidad de la segmentación se eleva el rendimiento del procesador. Sin embargo si supera a 8 el rendimiento disminuye como consecuencia de los conflictos por dependencia de datos y de control, así como por la incidencia del tiempo de carga de los registros interetapas y del flanco de la señal de reloj.