INGENIERIA DE COMPUTADORES II

TEMA I
PROCESADORES SEGMENTADOS

UNED
Manuel Fernandez Barcell
Http://www.mfbarcell.es

CONTENIDO TEMA I

- Capítulo 1. Procesadores segmentados
- 1.1. Guión-esquema
- 1.2. Introducción
- 1.3. Procesadores RISC frente a procesadores CISC
- 1.4. Clasificación de las arquitecturas paralelas
- 1.5. Evaluación y mejora del rendimiento de un computador
- 1.6. Características de los procesadores segmentados
- 1.7. Arquitectura segmentada genérica
- 1.7.1. Repertorio de instrucciones de la ASG
- 1.7.2. Implementación de la segmentación de instrucciones en la ASG
- 1.8. Riesgos en la segmentación
- 1.8.1. Riesgos estructurales
- 1.8.2. Riesgos por dependencias de datos
- 1.8.2.1. La reorganización de código
- 1.8.2.2. El interbloqueo entre etapas
- 1.8.2.3. El adelantamiento (caminos de bypass o forwarding)
- 1.8.3. Riesgos de control
- 1.9. Planificación dinámica: Algoritmo de Tomasulo
- 1.10. Resumen
- 1.11. Referencias
- 1.12. Preguntas de autoevaluación
- 1.13. Actividades

1.2 Introducción

- Un procesador segmentado es capaz de procesar varias instrucciones simultáneamente, aunque cada una de ellas puede encontrarse en una etapa distinta de su procesamiento.
  - Aprovecha el paralelismo intrínseco existente entre las instrucciones
    - Dos etapas del RISC I (1982),
    - Las 10 etapas del Pentium III.
    - Las 20 ó 31 etapas del Pentium 4 (2000-2005) según modelo.
    - Hasta las 14 etapas de los procesadores basados en la micro arquitectura Intel Core (2006-2010).

1.3. Procesadores RISC frente a procesadores CISC

- CICS (COMPLEX INSTRUCTION SET COMPUTER)
  - Inicialmente los procesadores fueron dotados de conjunto de instrucciones muy potentes, que realizaban gran cantidad de operaciones internas
    - Memoria escasa:
      - Programas con pocas instrucciones que hacían muchas operaciones en el micro código
      - Un gran numero de modos de direccionamiento
    - Dificultad para el paralelismos
      - Instrucciones de distintos tamaños, distinto número de argumentos…
  - Estudios demostraban que el 80 % del tiempo ejecutaban un 20% de Instrucciones

RISC (REDUCED INSTRUCTION SET COMPUTER)

- Conjunto de instrucciones sencillas con formatos fijos
  - Hardware de control mas sencillo
- Ventajas
  - Aumenta la velocidad de ejecución
  - Reduce el tamaño de la CPU, con menor consumo y más fácil diseño al tener lógica de control más simple
    - Aumenta el número de registros
    - Reduce los acceso a memoria cargando los datos en registros
  - Posibilita la segmentación y el paralelismo

1.4. Clasificación de las arquitecturas paralelas

- Clasificacion de Flynn (años 60)
  - Computadores SISD (SI, Single Instruction, SD, Single data)
  - Computadores SIMD (SI, Single Instruction, MD, Multiple Data)
  - Computadores MIMD (MI, Multiple Instruction MD, Multiple Data)
  - Computadores MISD (MI, Multiple Instruction SD, Single data)

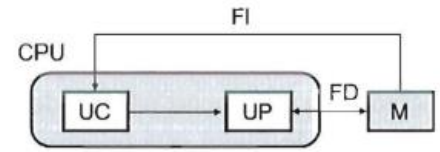
UC_i: Unidad de control *i*-ésima

UP_i: Unidad de procesamiento *i*-ésima

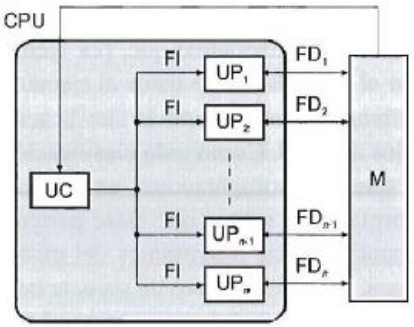
M: Memoria

FI_i: Flujo de instrucciones *i*-ésimo

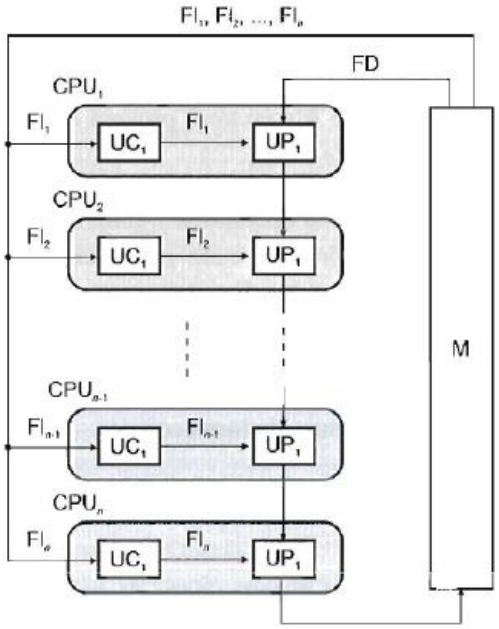
FD_i: Flujo de datos *i*-ésimo



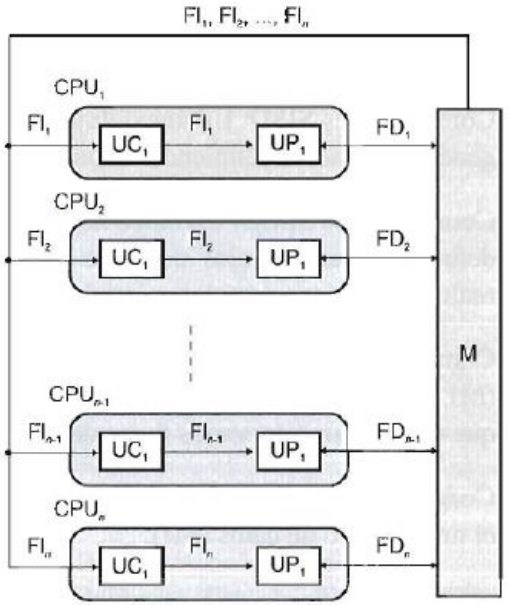
(a) SISD



(b) SIMD



(c) MISD



(d) MIMD

SISD

- **EJEMPLO**

```
for (i=1; i<=4; i++)
    C[i] := A[i] + B[i];
    F[i] := D[i] - E[i];
    G[i] := K[i] * H[i];
end for;
```

- Un SISD necesita 12 Intervalos.

SIMD

```
for all UPi ( i=1; i<=4; i++)
    C[i] := A[i] + B[i];
    F[i] := D[i] - E[i];
    G[i] := K[i] * H[i];
end for;
```

Figura 1.3: Fragmento de pseudocódigo del ejemplo correspondiente a un computador SIMD con cuatro unidades de procesamiento.

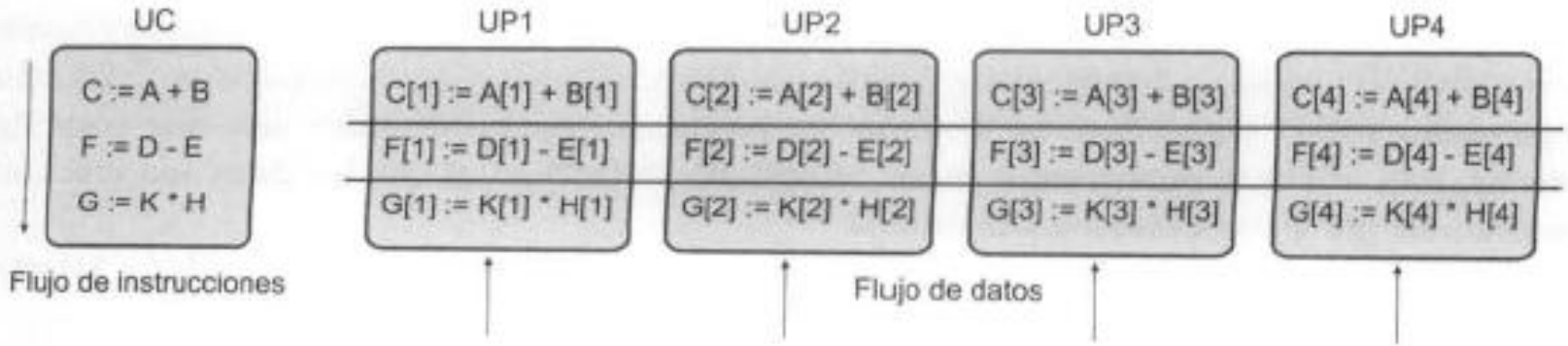


Figura 1.4: Esquema de procesamiento de tres instrucciones en un computador SIMD con cuatro unidades de procesamiento.

Tarda 3 intervalos de tiempo
Con SISD serían 12

MIMD

Procesador 1	Procesador 2	Procesador 3

for (i=1; i<=4; i++)
C[i] := A[i] + B[i];
end for;
	for (i=1; i<=4; i++)
	F[i] := D[i] - E[i];
	end for;
		for (i=1; i<=4; i++)
		G[i] := K[i] * H[i];
		end for;

Figura 1.5: Fragmento de código del ejemplo cuando se paraleliza en tres procesadores.

- Tarda 4 intervalos de tiempo

Tipos de Paralelismo

- Paralelismo de Datos
  - Cuando una misma función, instrucción, se ejecuta repetidas veces en paralelo **sobre datos diferentes**
- Paralelismo Funcional
  - Cuando las funciones (iguales o distintas) se ejecutan en paralelo

Niveles de paralelismo

- Granularidad
  - Cantidad de trabajo asociado a cada tipo de tarea candidata a la paralelización
- Nivel de instrucciones (ILP, *Instruction Level Parallelism*)
  - En paralelo instrucciones.
  - Granularidad fina
- Nivel de Bucle
  - En paralelo distintas iteraciones de un bucle o secuencias de instrucciones de programa.
  - Granularidad fina-media.
- Nivel de función
  - Los distintos procedimientos se ejecutan simultáneamente.
  - Granularidad media.
- Nivel de Programa
  - Programas ejecutados en paralelo.
  - Granularidad gruesa.

Paralelismo entre instrucciones ILP

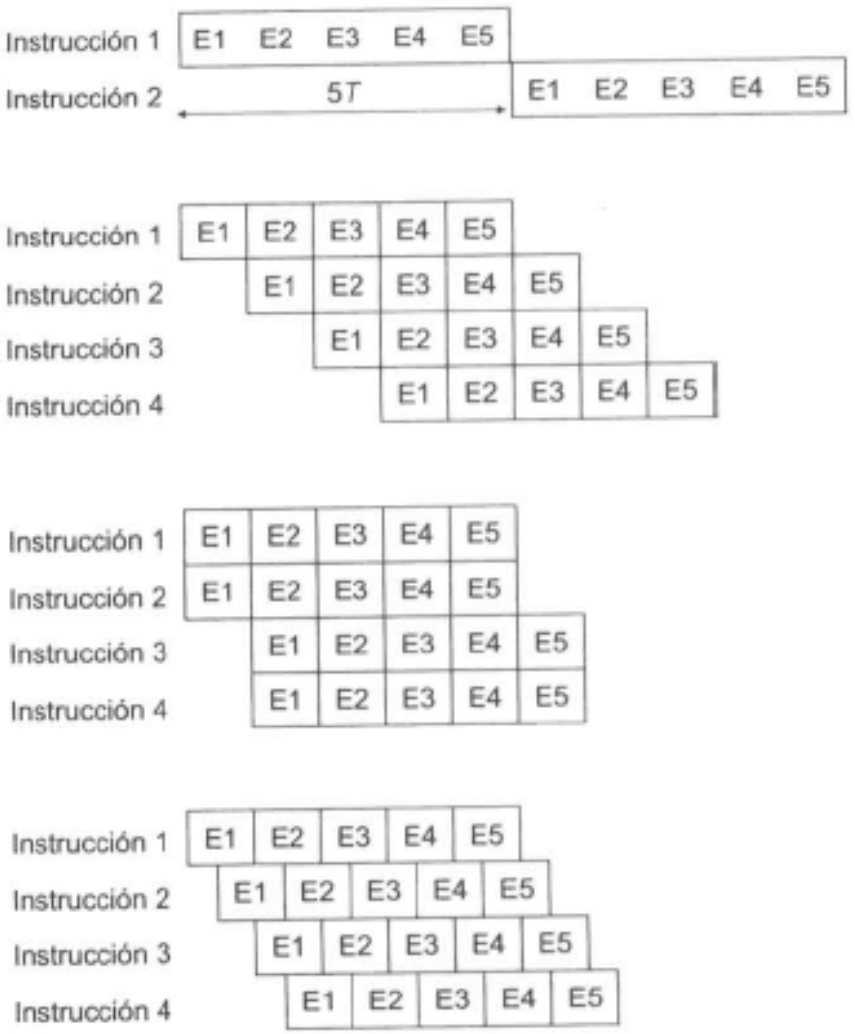


Figura 1.6: Alternativas para el aprovechamiento del paralelismo entre instrucciones (ILP).

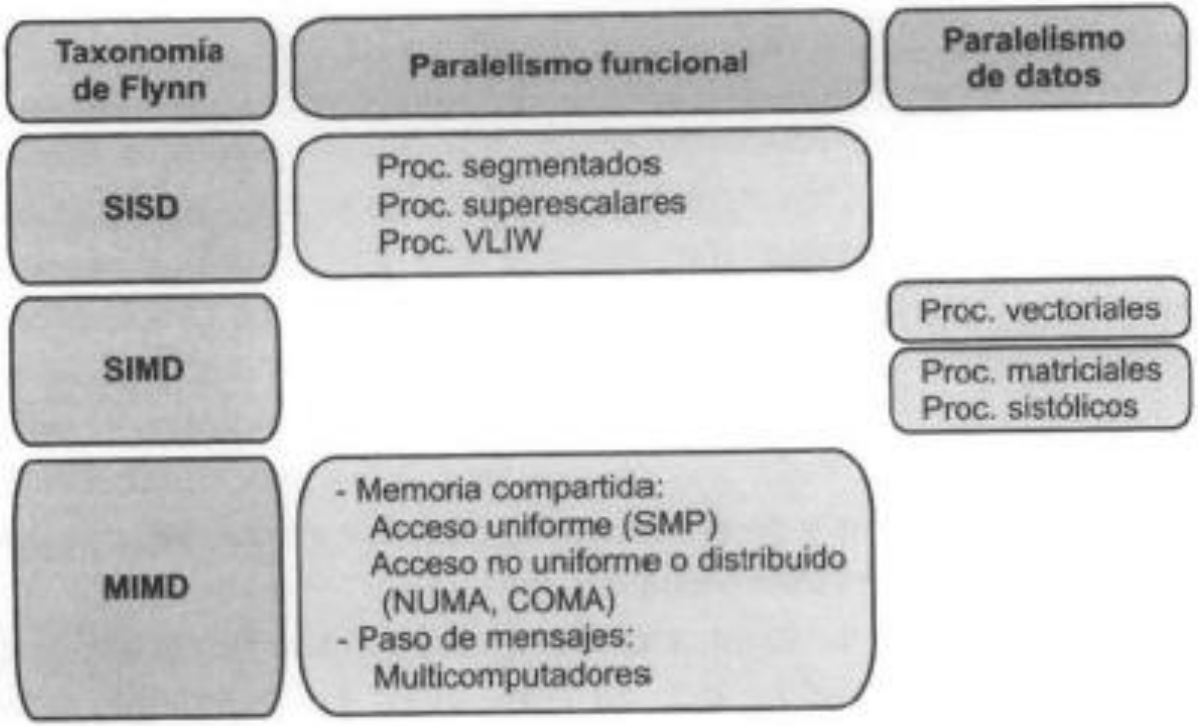


Figura 1.7: Arquitecturas según el tipo de paralelismo que aprovechan y su ubicación en la taxonomía de Flynn.

VLIW: Very Long Instruction Word
CPI: Número de Ciclos por Instrucción
SPMD: Simple Program Multiple Data

Tabla 3.1: Diferencias principales entre las arquitecturas superescalar, VLIW y EPIC.

	Agrupamiento de operaciones	Asignación de unidad funcional	Secuencia de emisión a las unidades funcionales
Superescalar	Hardware	Hardware	Hardware
EPIC	Compilador	Hardware	Hardware
VLIW	Compilador	Compilador	Compilador

1.5. Evaluación y mejora del rendimiento de un computador

- Tiempo de Respuesta:
  - Tiempo que tarda el computador en procesar una entrada
- Productividad (*Throughput*)
  - Número de entradas por Unidad de tiempo
- Funcionalidad
  - Tipos de entradas diferentes que es capaz de procesar
- Expansibilidad
  - Posibilidad de ampliar la capacidad de procesamiento añadiendo bloques
- Escalabilidad
  - Posibilidad de ampliar el sistema sin devaluar sus prestaciones
- Eficacia
  - Relación entre el rendimiento obtenido y el coste que ha supuesto conseguirlo (eficiencia = rendimiento/ coste).

Denominación

- La denominación de estos parámetros y su importancia depende del elemento que se estudia

Ejemplos

MEMORIA	Entradas	Accesos de memoria
	Rendimiento	Latencia de memoria. Ancho de banda. Escalabilidad.

COMPUTADOR	Entradas	Programas
	Rendimiento	Tiempo de respuesta. Productividad.

PROCESADOR	Entradas	Instrucciones
	Rendimiento	CPI*i* CPI *(medio)* Frecuencia de reloj

$$T_{CPU} = NI \cdot CPI \cdot T_{ciclo} = NI \cdot \left(\frac{CPI}{f} \right)$$

T_{CPU} : tiempo de CPU de un programa o tarea.
NI : número de instrucciones máquina del programa que se ejecutan.
CPI : número medio de ciclos por instrucción.
T_{ciclo} : periodo del reloj del procesador.

$$CPI = \frac{\text{Ciclos de reloj del programa}}{NI} = \frac{\sum_{i=1}^{n} NI_i \cdot CPI_i}{NI}$$

CPI_i .número medio de ciclos de las instrucciones de tipo i.
NI_i : número de instrucciones de ese tipo.

$$T_{CPU} = NI \cdot \left(\frac{CPE}{IPE} \right) \cdot T_{ciclo}$$

CPE : número medio de ciclos entre inicios de ejecución de instrucciones. (Ciclos por emisión)
IPE : número medio de instrucciones que se emiten. (Instrucciones por emisión)

$$T_{CPU} = \left(\frac{N_{operaciones}}{Op_{instrucción}} \right) \cdot CPI \cdot T_{ciclo}$$

$$MIPS = \frac{NI}{T_{CPU} \cdot 10^6} = \frac{f}{CPI \cdot 10^6}$$

$MIPS$: Millones de instrucciones por segundo.

$$MFLOPS = \frac{\text{Operaciones en coma flotante}}{T_{CPU}}$$

Factor de mejora o ganancia

$$S_p = \frac{rendimiento(p)}{rendimiento_original} = \frac{T_{CPU_original}}{T_{CPU_mejorada}}$$

$$S_p \leq \frac{p}{1 + f\,(p-1)} \quad \text{ley de Amdahl}$$

P= factor de mejora
F= fracción de tiempo que no se aplica la mejora

$$T_p \geq f \cdot T_1 + (1-f)\,\frac{T_1}{p}$$

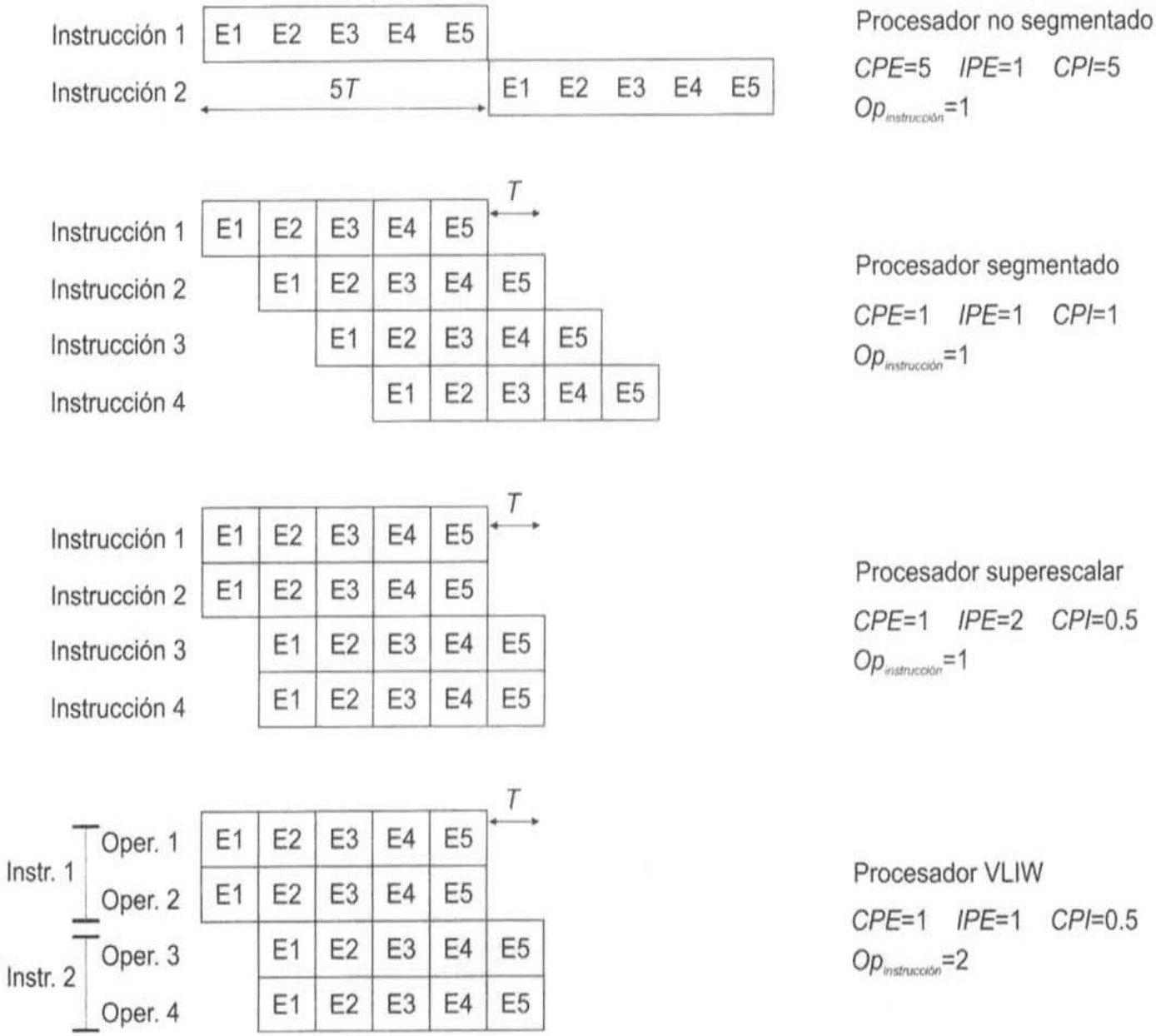


Figura 1.8: Ejemplos de valores característicos de *CPE*, *IPE* y *CPI* según las microarquitecturas segmentadas, superescalares y VLIW.

Ejemplo de Ley de AMDAHL

- Si una máquina pasa un **25%** de su tiempo procesando **instrucciones de coma flotante** y se mejora la máquina haciendo que esas **instrucciones** se ejecuten en la **mitad de tiempo**, esto es ***p* = 2**, entonces la ganancia que se puede obtener es

$$S_p \leq 2/(1 + 0,75 \cdot (2 - 1)) = 1,14.$$

- Es decir, la **máquina mejorada** solo es un **14%** mejor.
- Por mucho que se mejore el recurso, la ganancia será siempre limitada por 1/*f*

Ejemplo

- Se desea mejorar el rendimiento de un computador introduciendo un coprocesador matemático que realice las operaciones aritméticas en la mitad de tiempo. ¿Cuál sería la ganancia en velocidad del sistema para la ejecución de un programa si el 60% del mismo se dedicase a operaciones aritméticas? Si el programa tarda 12 segundos en ejecutarse sin la mejora, ¿Cuánto tardará con la mejora?
- Ganancia: p=2
- f= fracción de tiempo sin mejora. Mejora el 60% del tiempo · 1-0.6= 0.4 sin mejora

$$S_p = \frac{p}{1 + f \cdot (p-1)} = \frac{2}{1 + 0,4 \cdot (2-1)} = 1,42$$

Tiempo con la mejora:

$$S_p = \frac{T_{CPU_original}}{T_{CPU_mejorada}} = \frac{12}{T_{CPU_mejorada}} = 1,42$$

$$TCPU\ mejorada = \frac{12}{1,42} = 8,45\ seg$$

1.6. Características de los procesadores segmentados

- Procesamiento segmentado = fabricación en cadena
- Al número de etapas:
  - Profundidad de segmentación
- Para que el tiempo de latencia del procesador segmentado sea el mínimo posible, es necesario que **el procesador esté equilibrado**
  - Que todas las subtareas en que se haya dividido la tarea total tarden en procesarse el mismo tiempo.
- **La relación de precedencia** de un conjunto de subtareas T1, …. , T17 que componen cierta tarea T, específica para cada subtarea Tj que no puede comenzarse hasta que hayan terminado ciertas subtareas Ti.
- Las relaciones de precedencia para todas las subtareas de T forman su grafo de precedencia.
  - En el ejemplo de la Figura se ha supuesto que las tareas que se procesan en el cauce tienen un grafo de precedencia lineal.
    - Esto significa que una subtarea Tj no puede comenzar hasta que todas las subtareas previas, es decir Ti, ∀i < j , hayan finalizado.
    - A los procesadores segmentados que solo pueden procesar tareas con grafo de precedencia de este tipo se les denomina de **cauce lineal.**

Figura 1.9: Tarea procesada de forma totalmente secuencial.

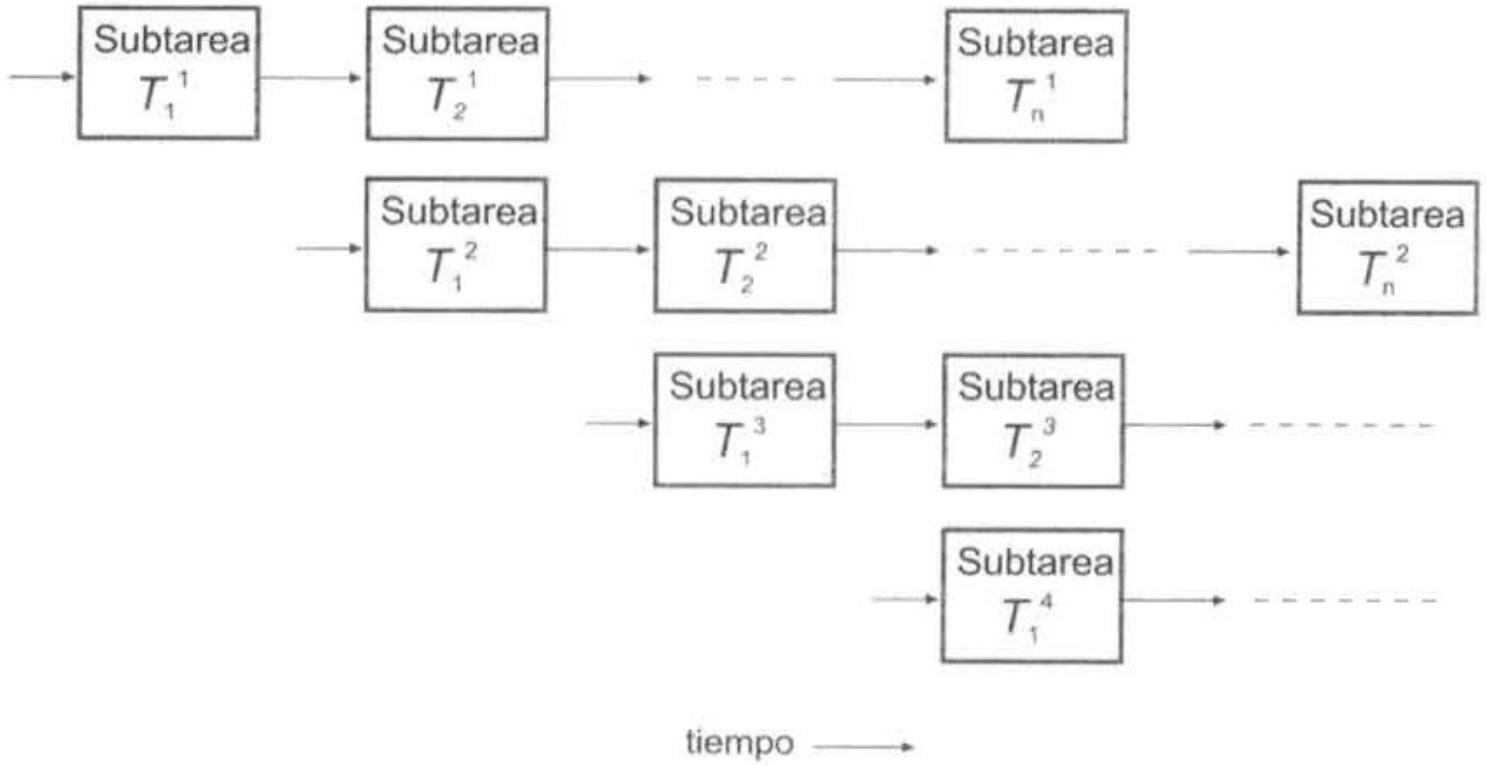


Figura 1.10: Conjunto de tareas ejecutadas en un procesador segmentado.

1.7. Arquitectura segmentada genérica (ASG)

- Existen dos características importantes de los repertorios de instrucciones que permiten clasificar las arquitecturas de propósito general:
  - El número de operandos que pueden tener las instrucciones aritmético-lógicas.
  - El número de operandos que se pueden direccionar en memoria en las instrucciones aritmético-lógicas.
- Las instrucciones aritmético-lógicas de la ASG utilizan en total tres operandos y ninguno de ellos se referencia en memoria.
  - A las máquinas en las que los operandos no se referencian en memoria se les denomina máquinas registro-registro o máquinas de carga/almacenamiento.

DE = <Registro Base> + OP; OP = Desplazamiento

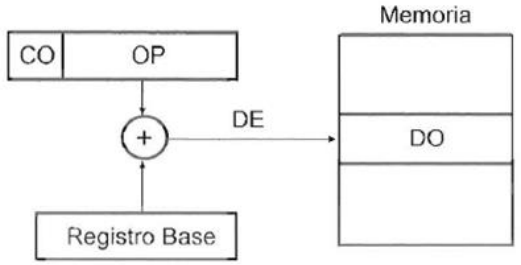


Figura 1.11: Esquema de direccionamiento con desplazamiento.

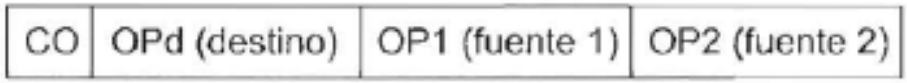
1.7.1.Repertorio de instrucciones de la ASG

- Descripción de la ASG (propósitos educativos)
  - 32 registros de 32 bits
  - Registros de coma flotante de 64 bits
- Máquina registro-registro
  - Los operandos no hacen referencia a memoria – (carga/almacenamiento)

Registros de la ASG	Registros genéricos	32 de 32 bits	R0 .. R31	R0 = 0
	Reg. Coma flotante	32 de simple precisión, 32 bits	F0 .. F31	
		16 parejas de 64 bits	F0, F2, F4, .. F30	

Tipos básicos de instrucciones	Aritméticas y lógicas
	Transferencia de datos
	Bifurcaciones y saltos incondicionales
	Saltos condicionales

1.7.1.1. Aritméticas y lógicas



Instrucción aritmético-lógica genérica de 3 operandos.

Tabla 1.1: Ejemplos de instrucciones aritméticas.

Instrucción ejemplo		Nombre de la instrucción	Significado
ADD	R3,R1,R2	Suma	R3 ← R1 + R2
ADDI	R2,R1,#4	Suma inmediata	R2 ← R1 + 4
ADDD	F2,F4,F6	Suma flotante	F2 ← F4 + F6
ADDDI	F8,F2,#4	Suma inmediata flotante	F8 ← F2 + 4
SUB	R5,R2,R3	Resta	R5 ← R2 - R3
SUBI	R4,R1,#8	Resta inmediata	R4 ← R1 – 8
SUBD	F4,F6,F8	Resta flotante	F4 ← F6 – F8
SUBDI	F2,F4,#8	Resta inmediata flotante	F2 ← F4 – 8
MULT	R3,R6,R5	Multiplicación	R3 ← R6 * R5
MULTI	R6,R1,#5	Multiplicación inmediata	R6 ← R1 * 5

MULTD	F4,F6,F8	Multiplicación flotante	F4 ← F6 * F8
MULTDI	F6,F2,#5	Multiplicación inmediata flotante	F6 ← F2 * 5
DIV	R7,R3,R5	División	R7 ← R3/R5
DIVI	R8,R1,#3	División inmediata	R8 ← R1/3
DIVD	F2,F6,F8	División flotante	F2 ← F6/F8
DIVDI	F8,F4,#3	División inmediata flotante	F8 ← F4/3
SGT	R3,R6,R5	Comparación mayor que	if R6 > R5 R3 ← 1 else R3 ← 0
SGTI	R6,R1,#5	Comparación inmediata mayor que	if R1 > 5 R6 ← 1 else R6 ← 0
SGE	R4,R6,R8	Comparación mayor o igual que	if R6 ≥ R8 R4 ← 1 else R4 ← 0
SGEI	R6,R2,#5	Comparación inmediata mayor o igual que	if R2 ≥ 5 R6 ← 1 else R6 ← 0
SLT	R3,R6,R5	Comparación menor que	if R6 < R5 R3 ← 1 else R3 ← 0
SLTI	R6,R1,#5	Comparación inmediata menor que	if R1 < 5 R6 ← 1 else R6 ← 0
SLE	R4,R6,R8	Comparación menor o igual que	if R6 ≤ R8 R4 ← 1 else R4 ← 0
SLEI	R6,R2,#5	Comparación inmediata menor o igual que	if R2 ≤ 5 R6 ← 1 else R6 ← 0

1.7.1.2. Transferencia de datos

```
LD   Fd,desplazamiento(Rf)
SD   desplazamiento(Rd),Ff
```

Ejemplos de instrucciones de carga y almacenamiento.

Instrucción ejemplo	Nombre de la instrucción	Significado
LD R1,4(R2)	Carga	R1 ← M[R2 + 4]
SD 0(R3),R5	Almacenamiento	M[R3 + 0] ← R5
LD F6,5(R3)	Carga flotante	F6 ← M[R3 + 5]
SD 6(R1),F4	Almacenamiento flotante	M[R1 + 6] ← F4

Ejemplos de instrucciones de bifurcación y salto.

Instrucción ejemplo	Nombre de la instrucción	Significado
JUMP etiqueta	Bifurcación	PC ← etiqueta
JUMP R4	Bifurcación a registro	PC ← R4
BEQZ R3,etiqueta	Salto igual a cero	if (R3 = 0) PC ← PC + etiqueta
BNEZ R2,etiqueta	Salto distinto de cero	if (R2 ≠ 0) PC ← PC + etiqueta
BGT R5,etiqueta	Salto mayor que cero	if (R5 > 0) PC ← PC + etiqueta
BLT R7,etiqueta	Salto menor que cero	if (R7 < 0) PC ← PC + etiqueta

1.7. Arquitectura segmentada genérica (ASG)

- Etapas
  - IF (*Instruction Fetch*)
    - Lectura de la instrucción de la caché de instrucciones
  - ID (*Instruction Decoding*)
    - Decodificación y lectura de los operandos del fichero de registros
  - EX (*Execution*)
    - Ejecución de las operaciones si se trata de una instrucción aritmético-lógica y del cálculo de la condición y de la dirección de salto si se trata de una bifurcación o salto condicional.
  - MEM (*Memory Access*) Acceso a la cache de datos
    - Acceso a la caché de datos para lecturas (cargas) o escrituras (almacenamientos).
  - WB (*Write-Back results*)
    - Escritura del resultado en el fichero de registros

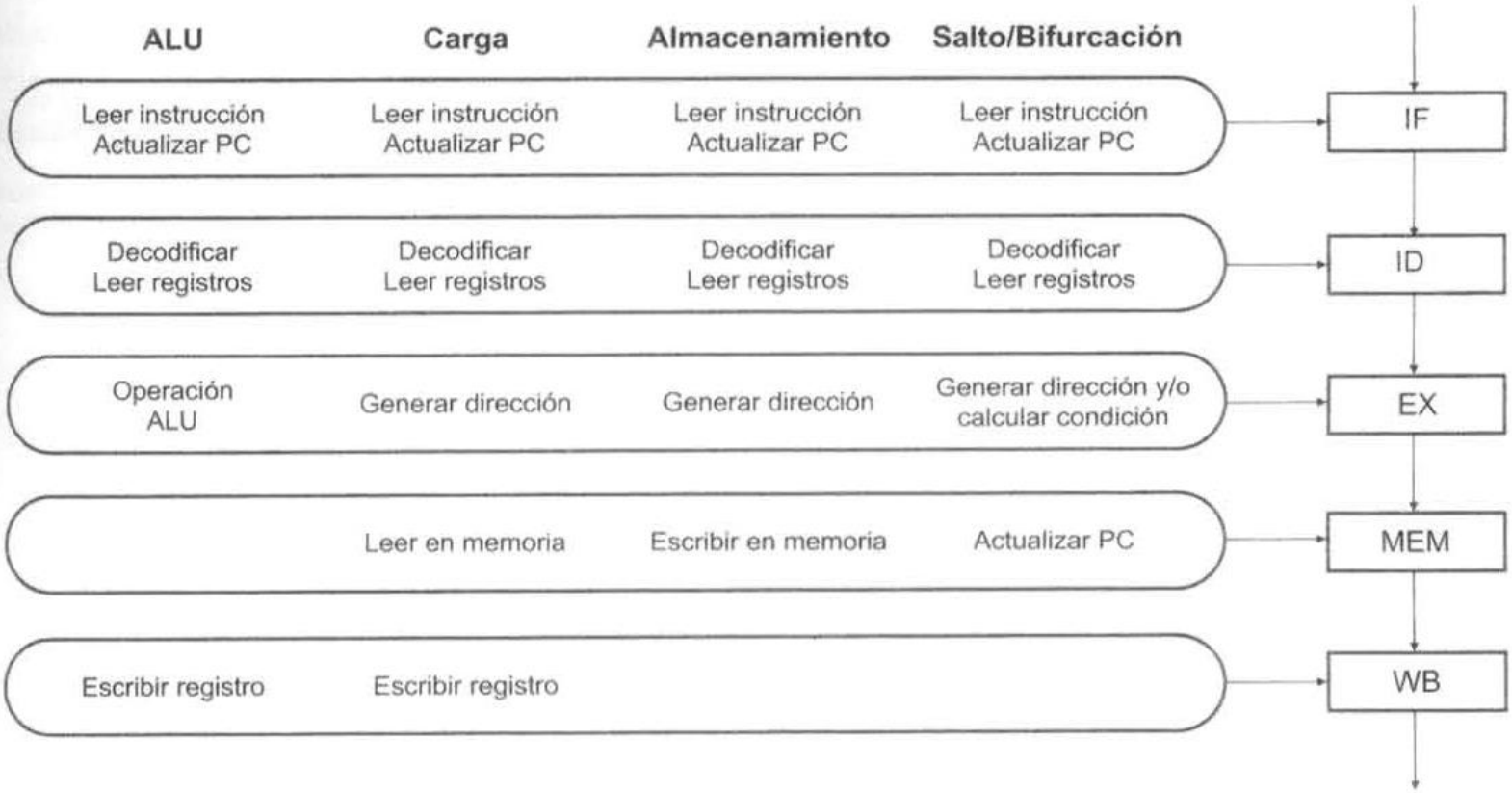
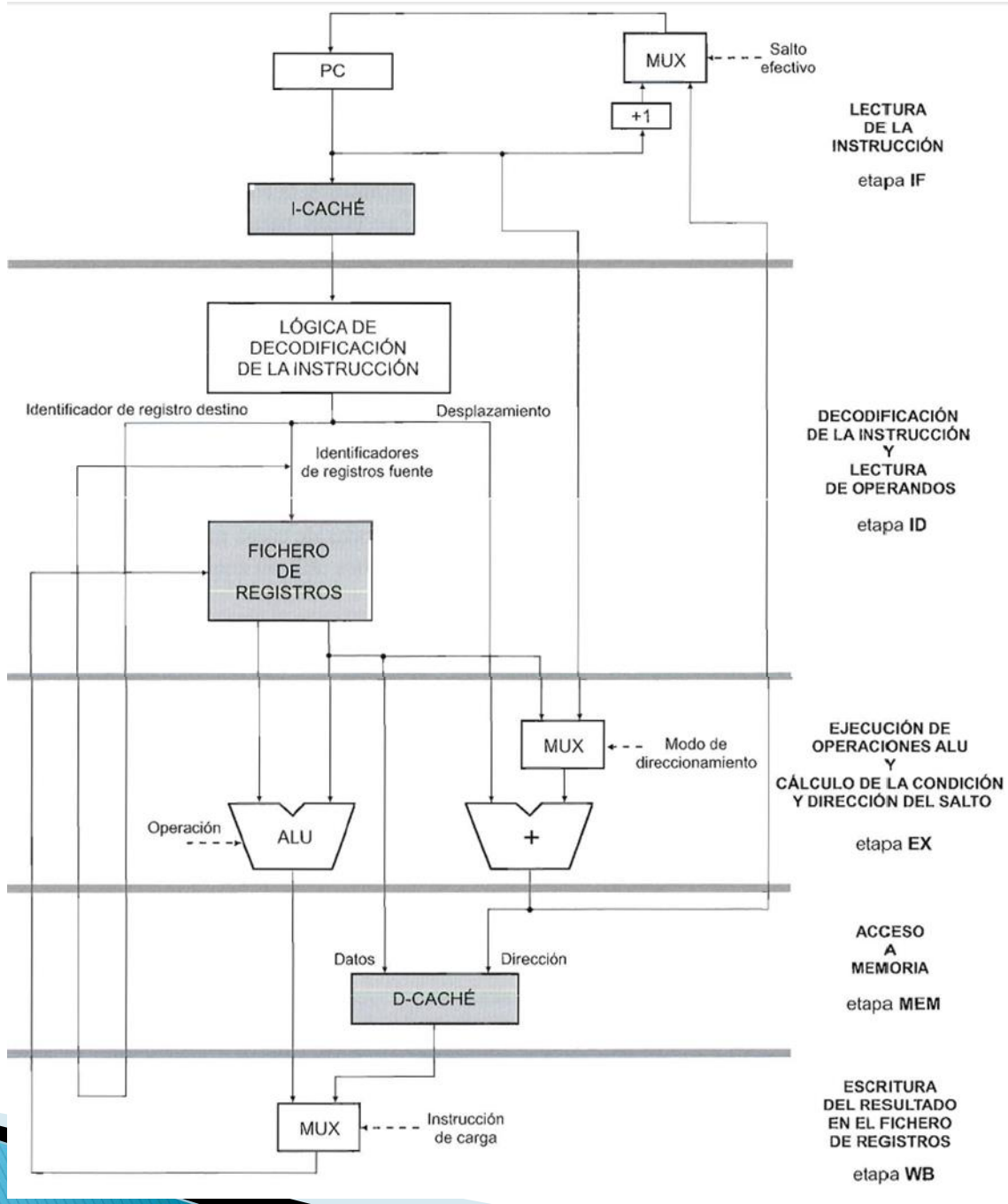


Figura 1.13: Operaciones que cada tipo de instrucción (ALU, carga, almacenamiento y salto) realiza en las etapas de que consta la segmentación ASG.


PC
MUX
Salto efectivo
+1
I-CACHÉ
LECTURA DE LA INSTRUCCIÓN
etapa IF
LÓGICA DE DECODIFICACIÓN DE LA INSTRUCCIÓN
Identificador de registro destino
Desplazamiento
Identificadores de registros fuente
FICHERO DE REGISTROS
DECODIFICACIÓN DE LA INSTRUCCIÓN Y LECTURA DE OPERANDOS
etapa ID
MUX
Modo de direccionamiento
Operación
ALU
+
EJECUCIÓN DE OPERACIONES ALU Y CÁLCULO DE LA CONDICIÓN Y DIRECCIÓN DEL SALTO
etapa EX
Datos
Dirección
D-CACHÉ
ACCESO A MEMORIA
etapa MEM
MUX
Instrucción de carga
ESCRITURA DEL RESULTADO EN EL FICHERO DE REGISTROS
etapa WB

Patrón de ejecución segmentación ASG

Número de instrucción	Ciclo de reloj 1	2	3	4	5	6	7	8	9
Instrucción *i*	IF	ID	EX	MEM	WB				
Instrucción *i*+1		IF	ID	EX	MEM	WB			
Instrucción *i*+2			IF	ID	EX	MEM	WB		
Instrucción *i*+3				IF	ID	EX	MEM	WB	
Instrucción *i*+4					IF	ID	EX	MEM	WB

Patrón de ejecución obtenido en la segmentación de la ASG al procesar cinco instrucciones.

Factores que determinan el tiempo de ejecución de las instrucciones

- Los cerrojos o buffers de contención
- La duración de todas las etapas de la segmentación es similar
  - Viene determinada por la duración de la etapa más lenta
- Los riesgos que se producen en la segmentación y que introducen detenciones en el cauce.
  - La segmentación aumenta el número de instrucciones completadas por unidad de tiempo,
  - Pero no reduce (incrementa) el tiempo de ejecución de una instrucción
    - El tiempo total de ejecución de la instrucción segmentada es ligeramente superior al de su equivalente no segmentada debido al tiempo que se consume en el control de la segmentación.

Ejecución no segmentada y segmentada

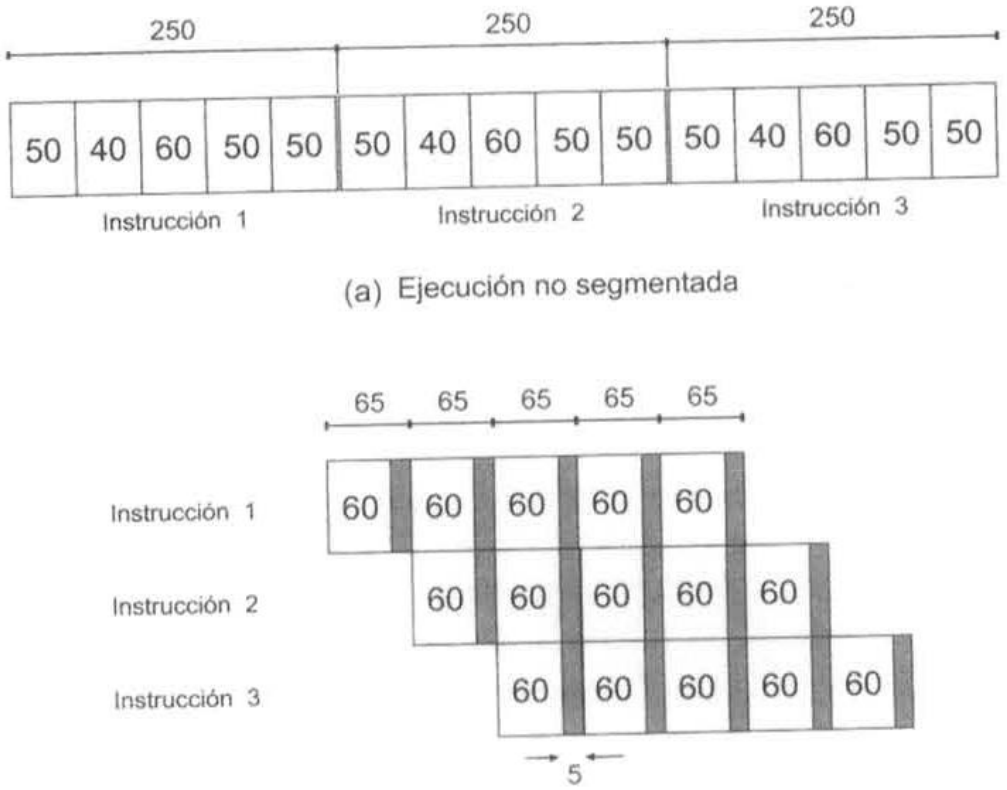


Figura 1.16: Patrón de ejecución segmentada y no segmentada de tres instrucciones.

1.8. Riesgos en la segmentación

- Riesgo
  - A la situación que impide a una instrucción acceder a la ejecución de sus etapas al depender de otra anterior.
- Riesgos estructurales
  - Insuficiencia de hardware
    - El hardware que necesita está ocupado
- Riesgos de dependencia de datos
  - Una instrucción necesita los datos de otra anterior
- Riesgos de control
  - Las instrucciones de control de flujo (saltos y bifurcaciones)
    - No pueden leer la instrucción siguiente hasta que no se conoce su dirección, que se calcula en la instrucción de control de flujo

1.8.1. Riesgos estructurales

Dos instrucciones necesitan el mismo recurso hardware

Acceso a memoria para buscar un dato

Conflicto al acceder a memoria

Instrucción	Ciclo de reloj 1	2	3	4	5	6	7	8	9	10
Instrucción de carga	IF	ID	EX	MEM	WB					
Instrucción *i*+1		IF	ID	EX	MEM	WB				
Instrucción *i*+2			IF	ID	EX	MEM	WB			
Instrucción *i*+3				IF	ID	EX	MEM	WB		
Instrucción *i*+4					IF	ID	EX	MEM	WB	

Acceso a memoria para buscar una instrucción

Solución al conflicto al acceder a memoria

Instrucción	Ciclo de reloj 1	2	3	4	5	6	7	8	9	10
Instrucción de carga	IF	ID	EX	MEM	WB					
Instrucción *i*+1		IF	ID	EX	MEM	WB				
Instrucción *i*+2			IF	ID	EX	MEM	WB			
Instrucción *i*+3				detención	IF	ID	EX	MEM	WB	
Instrucción *i*+4						IF	ID	EX	MEM	WB

Figura 1.17: Detención durante un riesgo estructural en una máquina con un único puerto de memoria cuando se ejecuta una instrucción de carga seguida de cuatro instrucciones aritmético-lógicas.

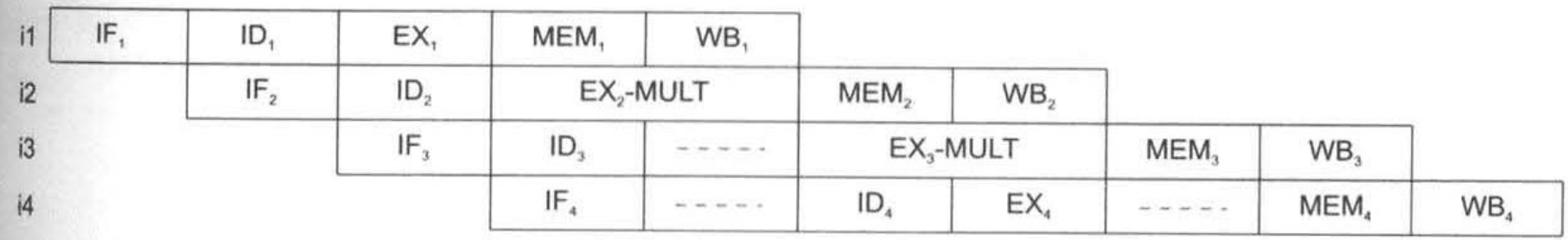
- La ASG no se presenta este inconveniente porque se dispone de dos memorias caché, una para instrucciones (I-caché) y otra para datos (D-caché).

Instrucciones más complejas que otras

Mayor duración de la etapa de ejecución

```
i1: ADD    R3, R6, R7
i2: MULT   R4, R1, R2
i3: MULT   R5, R8, R9
i4: SD     7(R11), R10
```

(a)



(b)

Figura 1.18: Ejemplo de riesgo estructural cuando hay instrucciones más complejas que otras.

Solución 1:Duplicar la unidad funcional

```
i1: ADD   R3, R6, R7
i2: MULT  R4, R1, R2
i3: MULT  R5, R8, R9
i4: SD    7(R11), R10
```

(a)

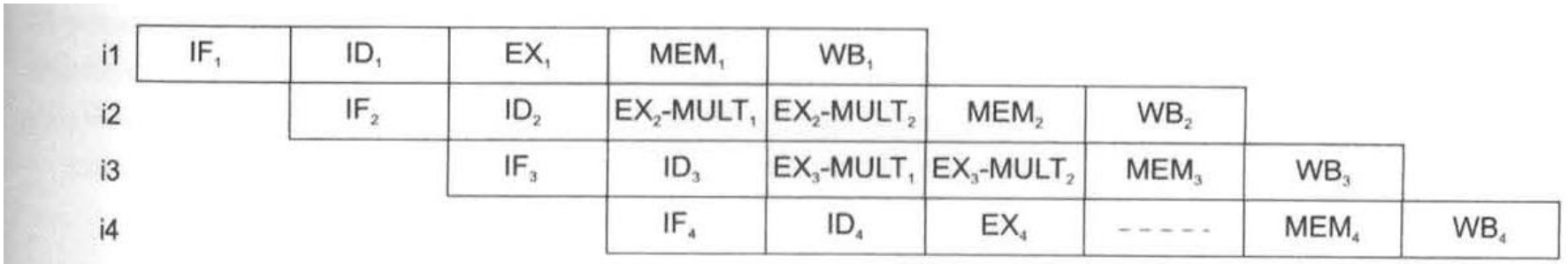


Figura 1.19: Duplicación de la unidad funcional aritmética. Se tiene un ciclo de espera en la ejecución de estas 4 instrucciones.

Solución 2:Planificación de código

Cambiar el orden de ejecución de las instrucciones)

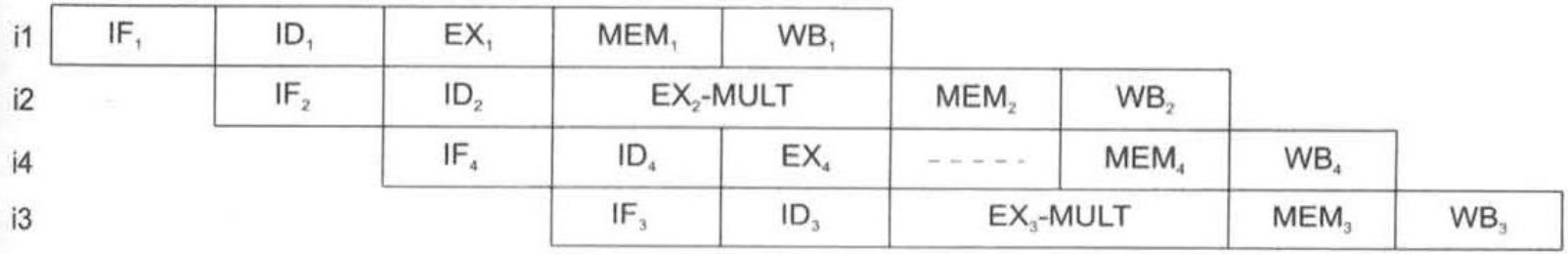
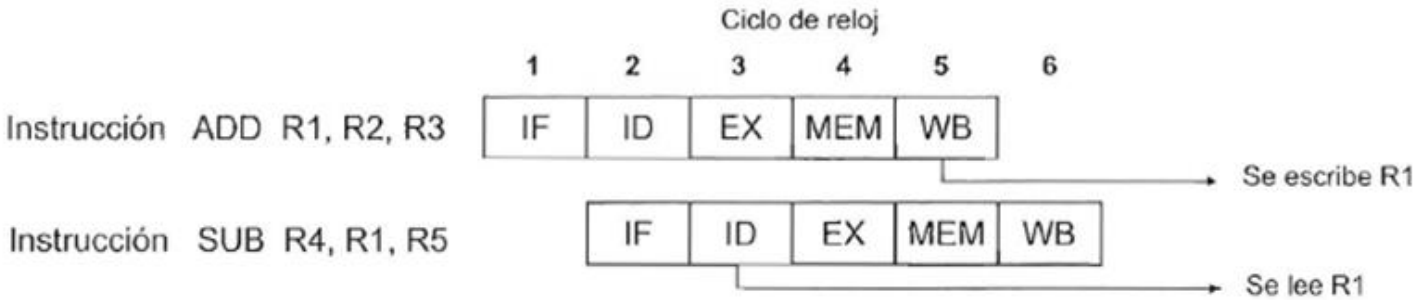


Figura 1.20: Planificación de código del tal manera que no se solapen dos etapas EX de dos instrucciones en el mismo ciclo de reloj. Se tiene un ciclo de espera en la ejecución de estas 4 instrucciones.

1.8.2. Riesgos por dependencias de datos

- Cuando dos instrucciones comparten algún dato
- Tipos de dependencias de datos
  - **Riesgo de tipo WAR** (Write After Read – Escritura tras lectura)
    - Instrucción *j* trata de escribir en su destino antes que éste sea leído por la instrucción
  - **Riesgo del tipo WAW** (Write After Write)
    - Instrucción *j* intenta escribir un operando antes de que éste sea escrito por la instrucción *i.*
  - **Riesgos de tipo RAW** (Read After Write- Lectura Tras escritura)

Ejemplo de RAW



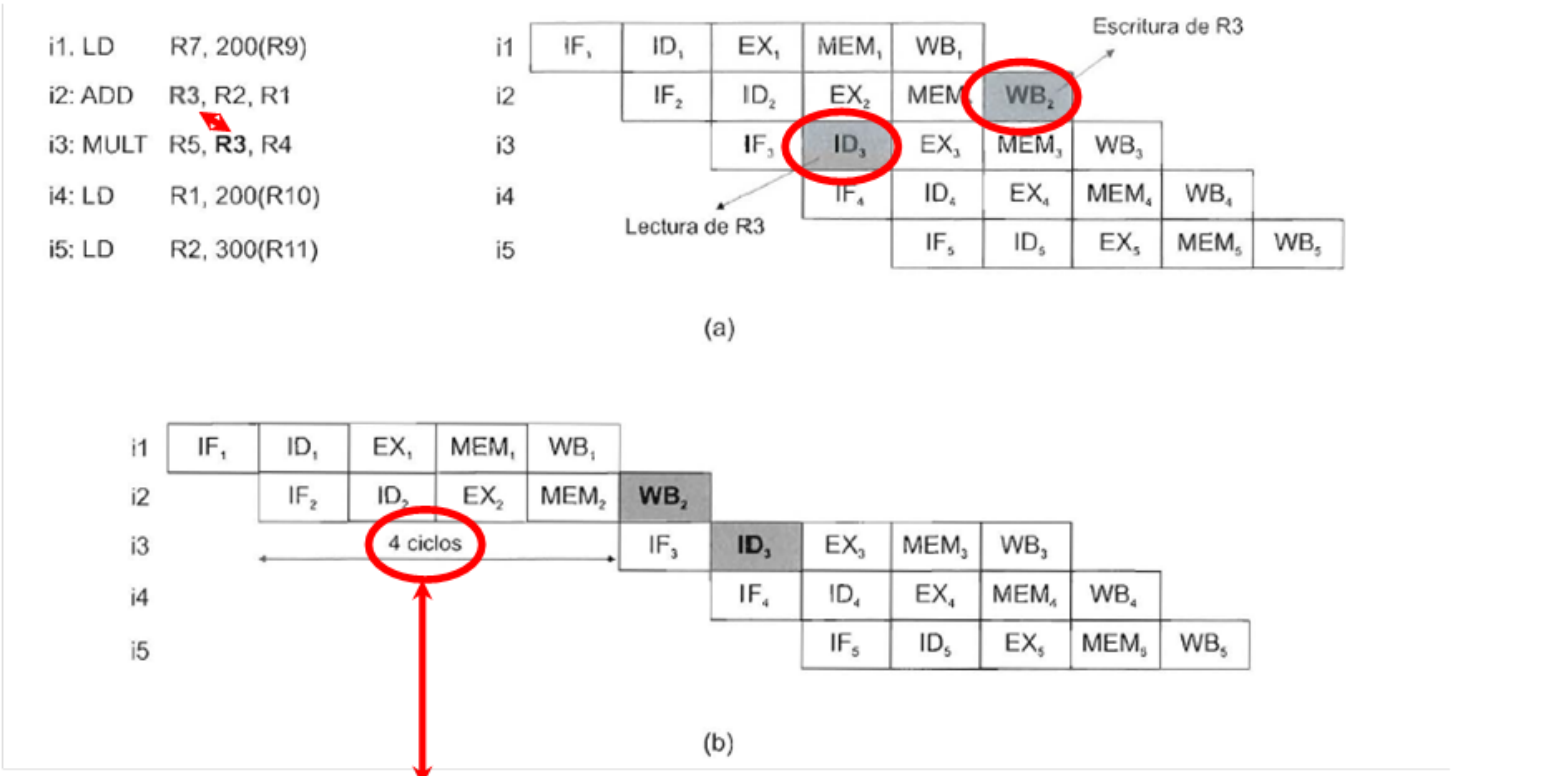
: Ejemplo de riesgo por dependencia de datos.

- Soluciones
  - Reorganización del código
  - Interbloqueo entre etapas
  - Adelantamiento

1.8.2.1. (RAW) La reorganización de código

- Detención de ciclos

CUANDO EL COMPILADOR RETRASA LA 2ª INST. RESPECTO DE LA 1ª



1. AL NO REORGANIZAR CÓDIGO → UNA DETENCIÓN (BURBUJA) DE 4 CICLOS
2. EL COMPILADOR INSERTA INSTRUCCIONES NOP

Inserción de NOP

Si no se pude reordenar el código, sin alterar la lógica del programa

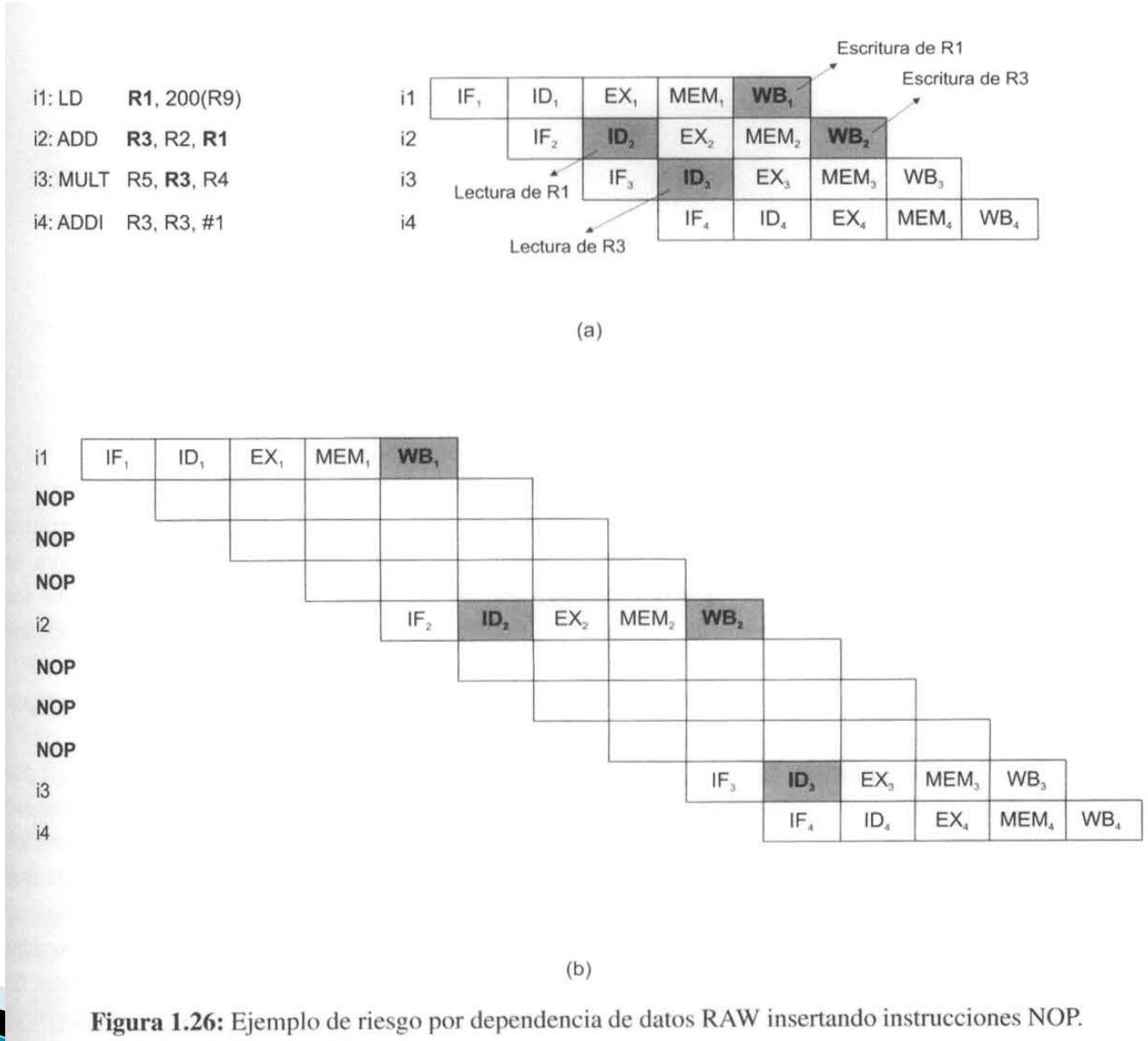
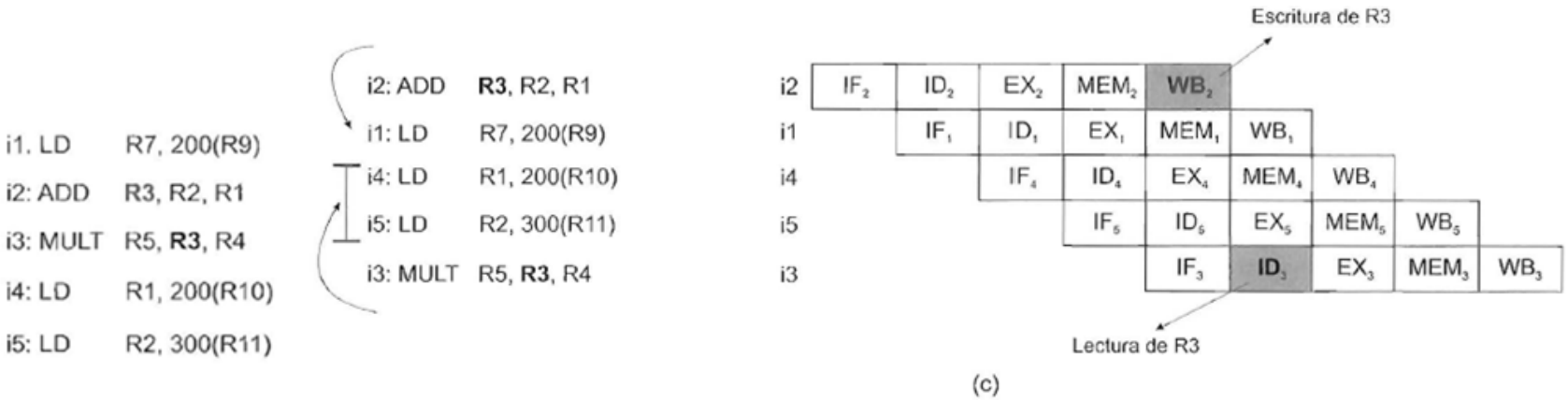


Figura 1.26: Ejemplo de riesgo por dependencia de datos RAW insertando instrucciones NOP.

Reorganización de código



(c)

Ejemplo de riesgo por dependencia de datos RAW con reorganización de código.

Flancos de subida y bajada

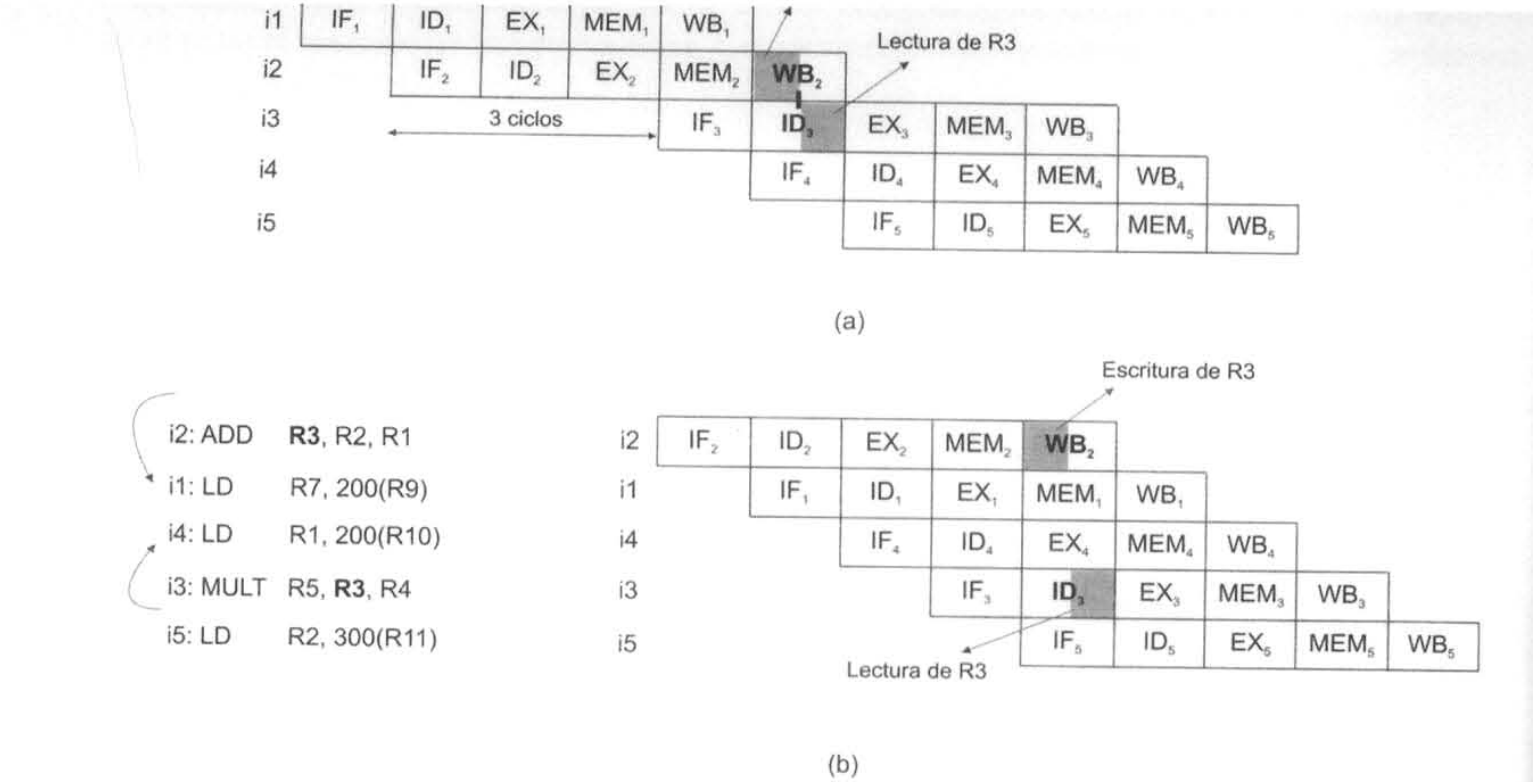


Figura 1.25: Ejemplo de riesgo por dependencia de datos RAW sin y con reorganización de código cuando la escritura se realiza en el flanco de subida y la lectura en el flanco de bajada.

1.8.2.2. (RAW) Interbloqueo de etapas

Introducir elementos hardware en el cauce para detectar la existencia de dependencias (no el compilador)

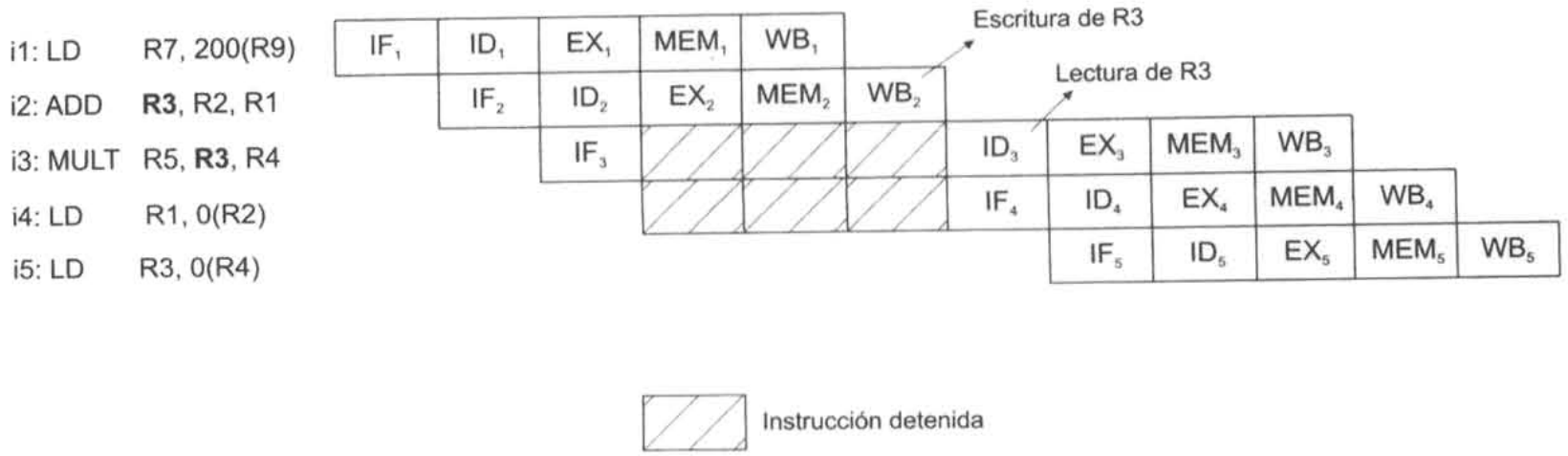


Figura 1.27: Ejemplo de riesgo por dependencia de datos RAW con interbloqueo entre etapas.

1.8.2.3. (RAW) El adelantamiento (caminos de bypass o forwarding)

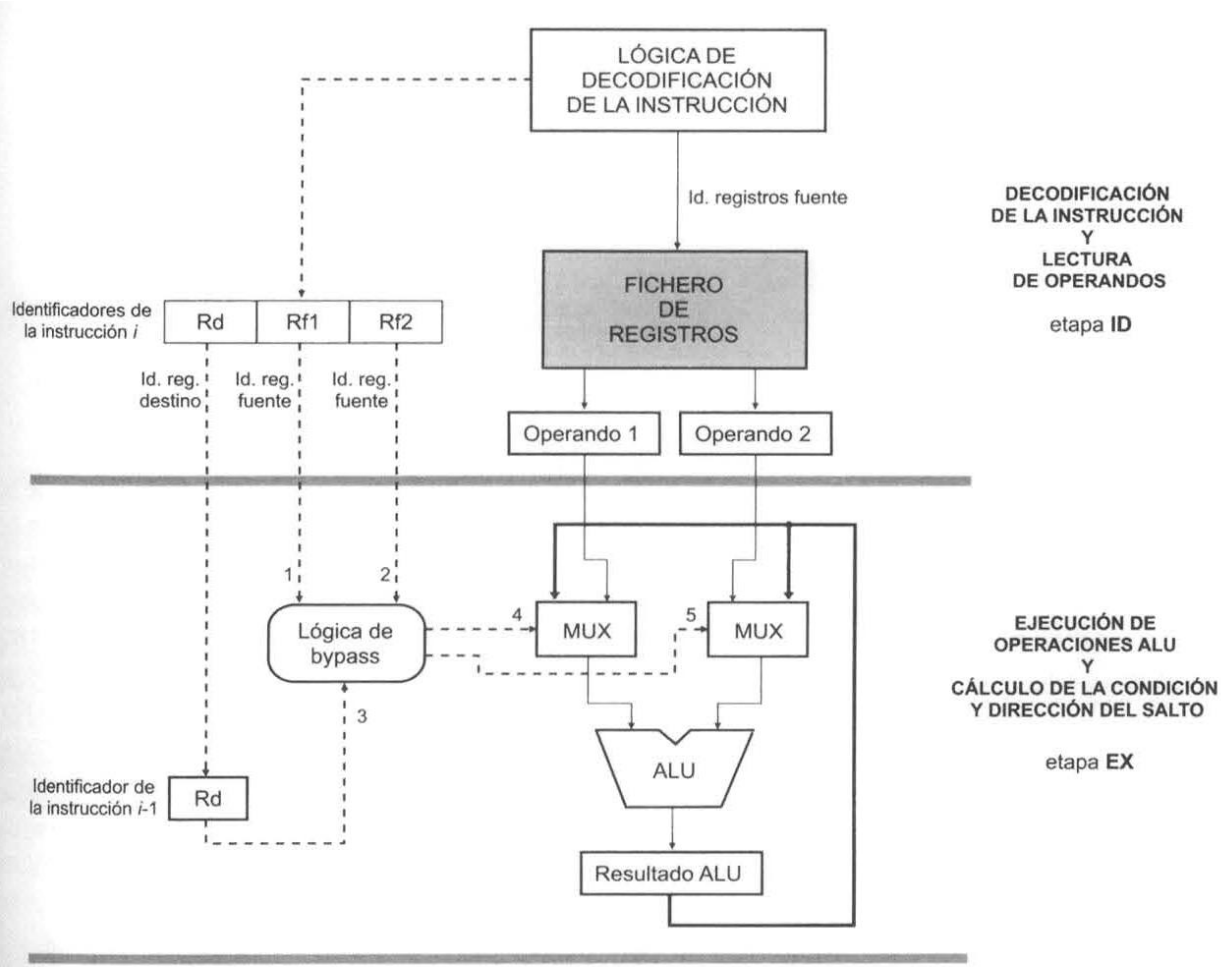


Figura 1.28: Solución con bypass o atajo para riesgos por dependencia de datos RAW.

1.8.2.3. (RAW) El adelantamiento (caminos de bypass o forwarding)

- Uso de los elementos que en la técnica de interbloqueo (hardware) permiten detectar la existencia de dependencias entre instrucciones.
- Esta información ahora se aprovecha para habilitar una serie de caminos (buses) que se añaden al cauce para permitir que los resultados de una etapa pasen como entradas a la etapa donde son necesarios en caso de dependencias RAW, al mismo tiempo que siguen su camino para almacenarse en el fichero de registros.

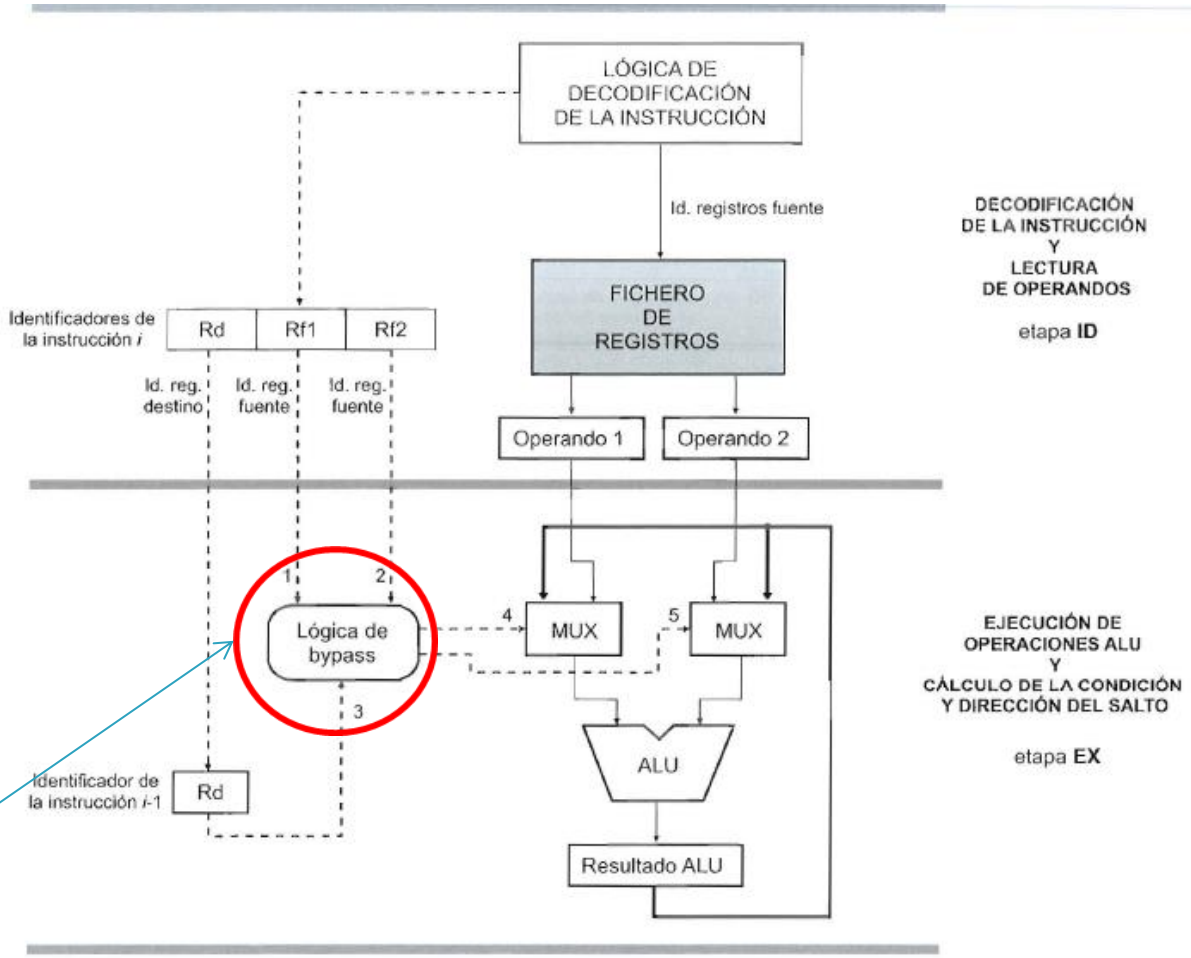


Figura 1.28: Solución con bypass o atajo para riesgos por dependencia de datos RAW.

Detecta si hace falta algún resultado que se ha de dejar en un registro para la siguiente instrucción.
En caso afirmativo, lo guarda en el registro, pero se lo suministra directamente a la ALU

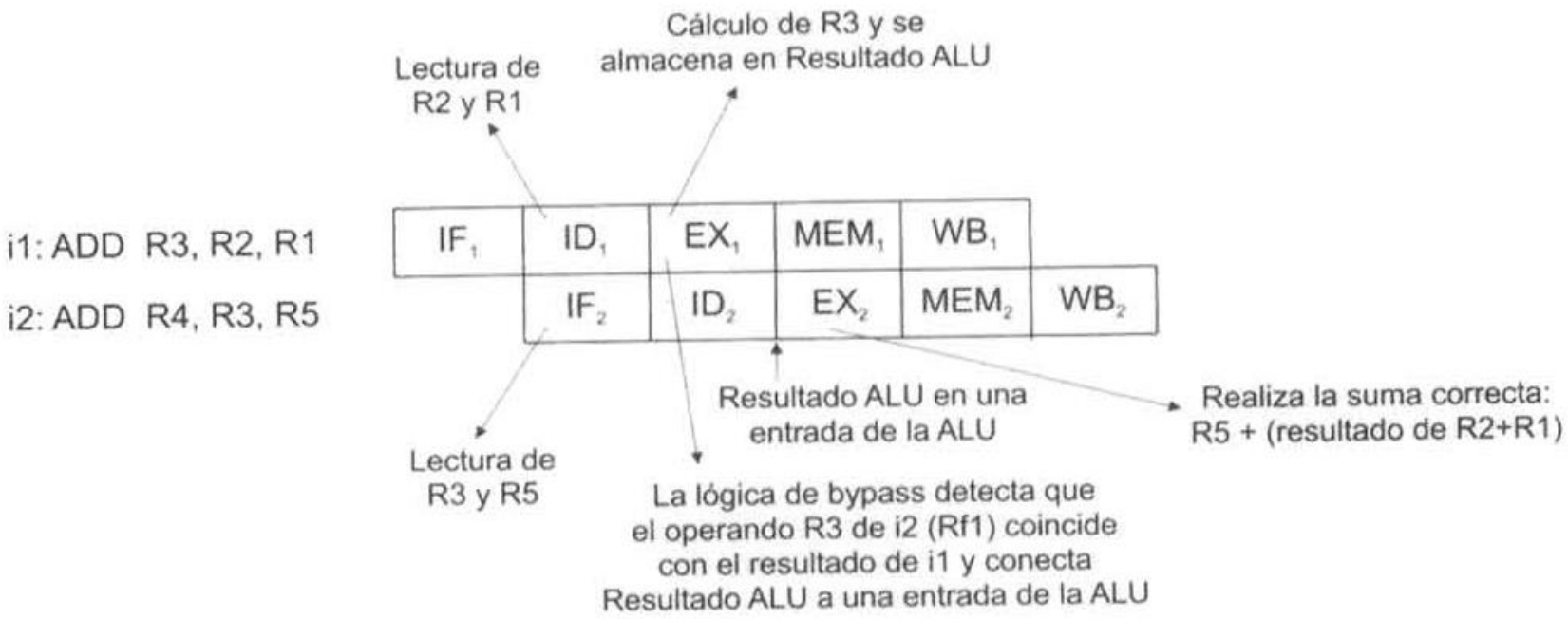
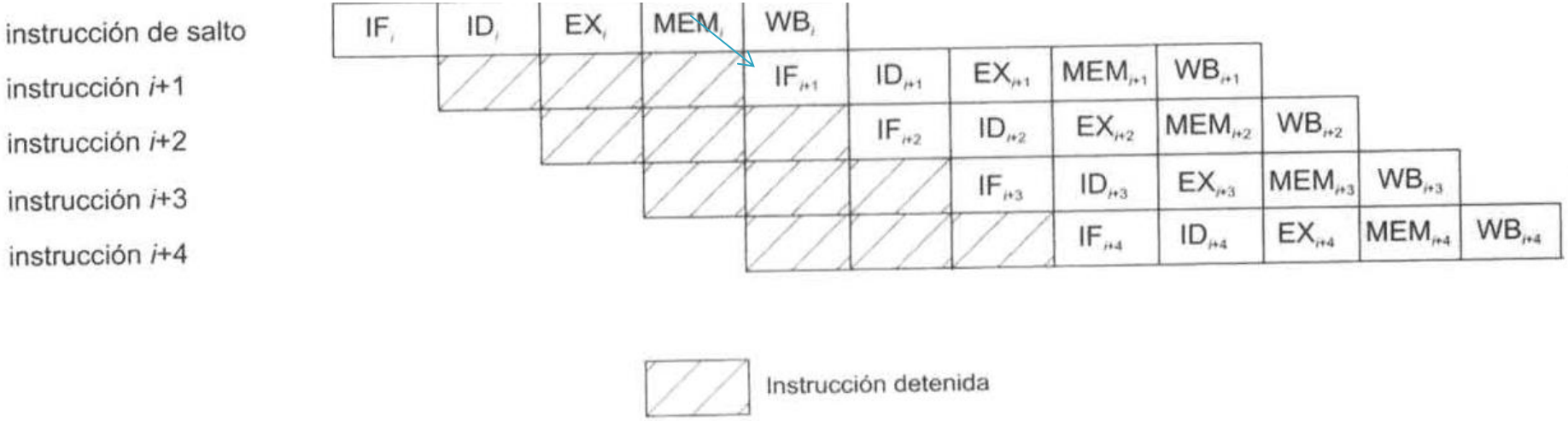


Figura 1.29: Ejemplo del funcionamiento de la técnica de atajo o bypass.

1.8.3. Riesgos de control

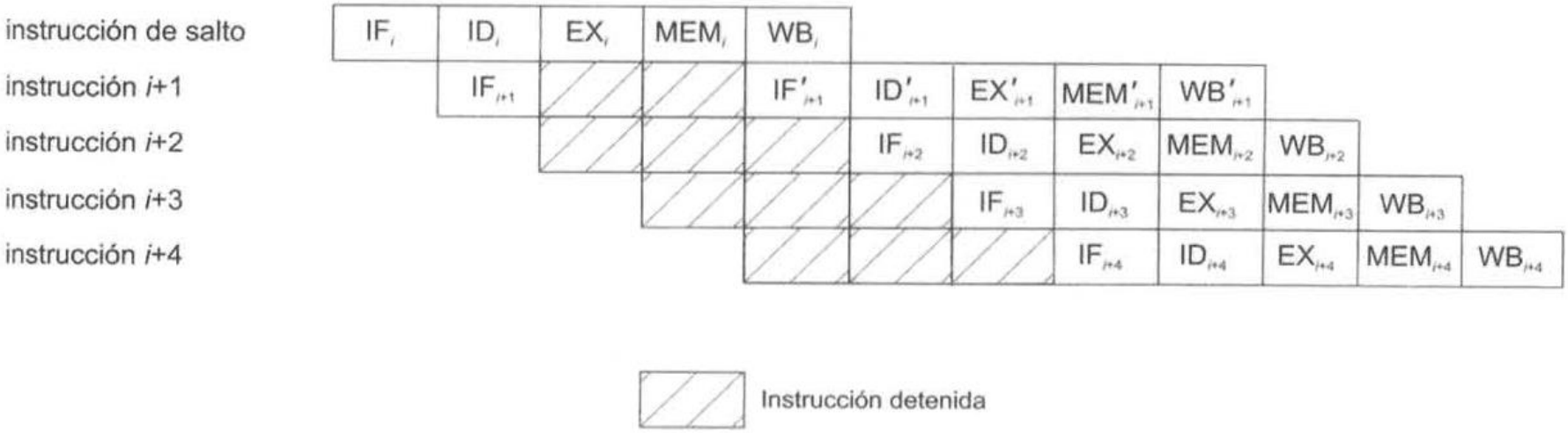
- Un salto condicional, el valor del contador del programa puede incrementarse automáticamente o cambiar su valor en función de que el salto sea efectivo o no efectivo.
- En la ASG la instrucción i es un salto efectivo entonces el PC no se actualiza hasta el final de la etapa MEM de la segmentación



La instrucción *i*+1 puede corresponderse con la siguiente en la secuencia del programa o estar alejada cuando el salto sea efectivo.

Figura 1.30: Detención ideal de la ASG debido a un riesgo de control.

Detección real después de riesgo de control



La instrucción *i*+1 puede corresponderse con la siguiente en la secuencia del programa o estar alejada cuando el salto sea efectivo. Si el salto no es efectivo, la etapa IF'$_{i+1}$ es redundante para la instrucción *i*+1.

Figura 1.31: Detención real de la ASG después de un riesgo de control.

- Busca la siguiente instrucción en la secuencia, aunque se ignora y se vuelve a comenzar la búsqueda de la instrucción correcta una vez que se conoce el destino del salto

- Modificaciones en las etapas de segmentación para procesar instrucciones de salto condicional
- La señal que controla el camino habilitado por el multiplexor lo genera el módulo “logica de condición”

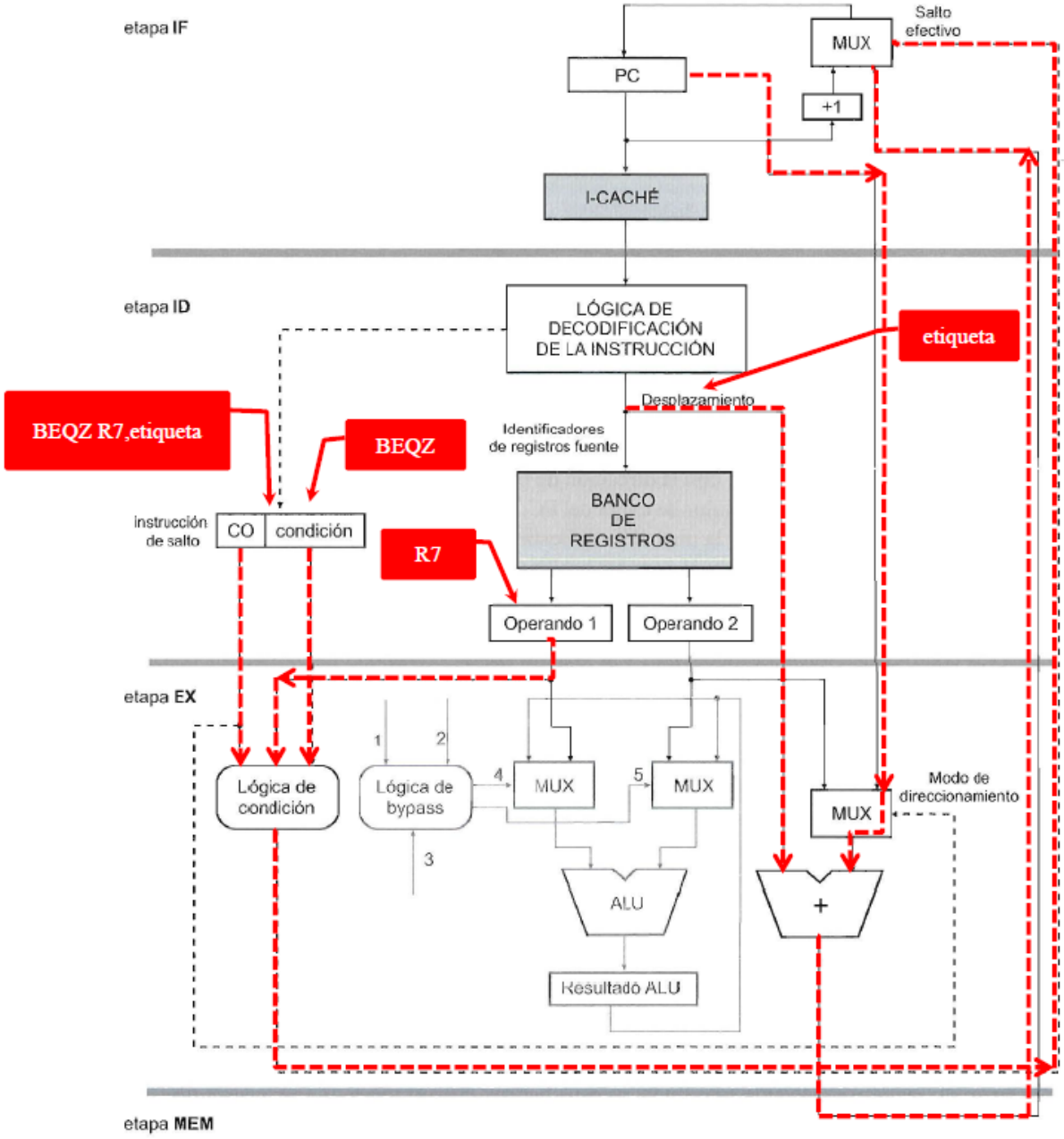


Figura 1.32: Implementación de los saltos condicionales en la ASG.

Soluciones estáticas (compilador)

Detener la segmentación hasta que no se conoce el resultado del salto, introduciendo tres operaciones NOP

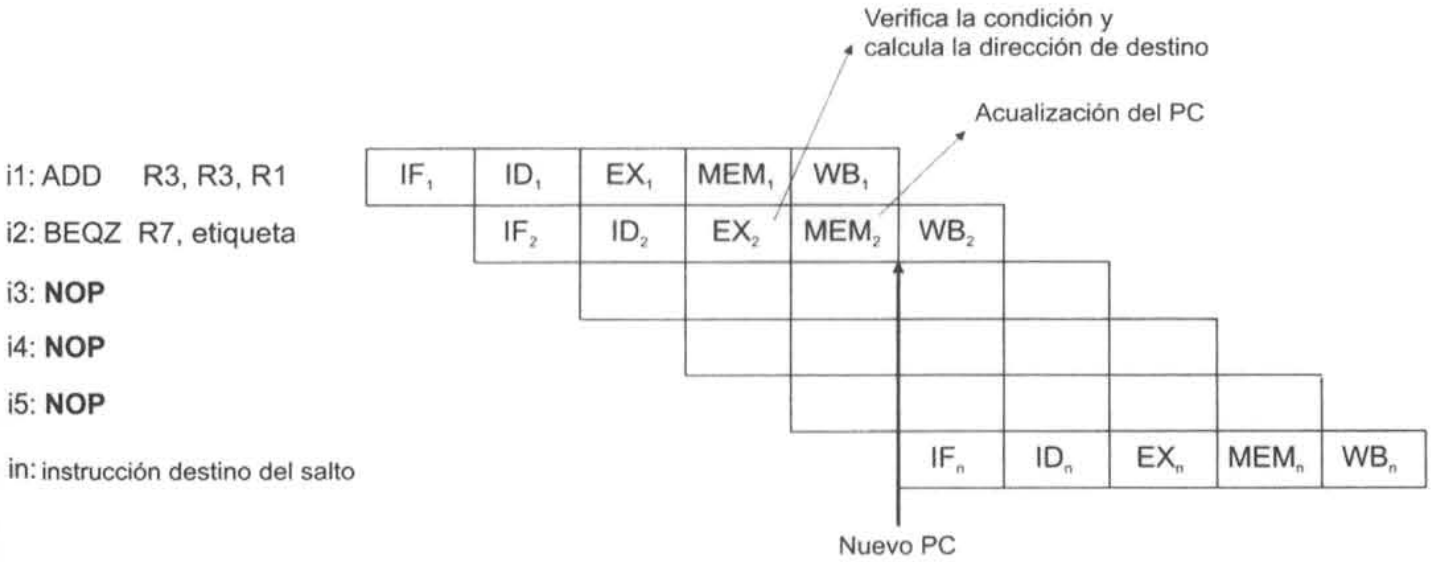
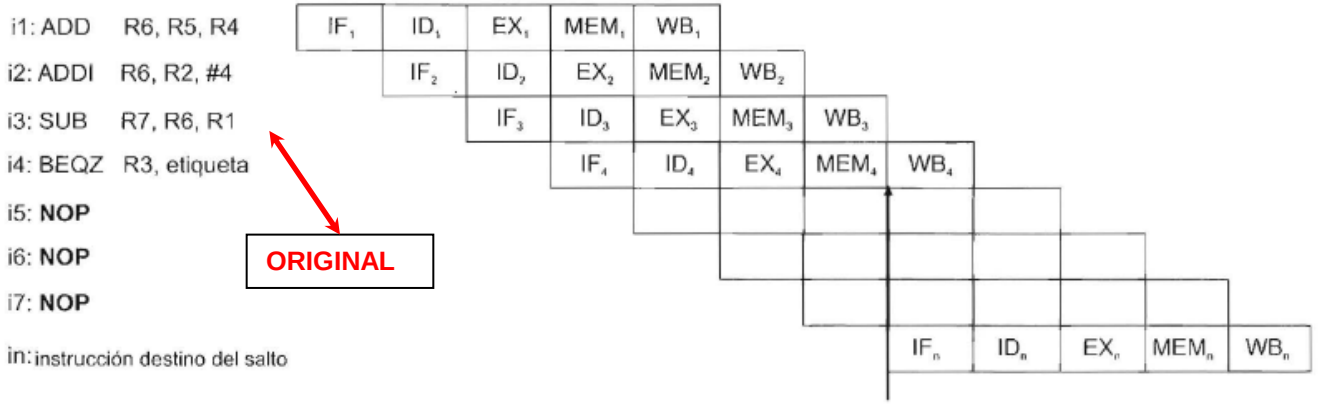


Figura 1.33: Uso de instrucciones NOP en la ejecución de una instrucción de salto condicional para no abortar instrucciones captadas erróneamente.

Técnica de salto retardado (Delay branch)

- Otra alternativa sería dejar que las instrucciones que se han captado prosigan su ejecución en el cauce.
- En este caso, el compilador debería **introducir en esos huecos instrucciones** que se **tengan** que **ejecutar antes** de la **instrucción destino** de salto de forma que **su efecto** sea **independiente** de que el **salto** sea **efectivo o no**.



i1, i2 e i3 no influyen para el salto, por lo tanto se pueden cambiar de posición. Mientras se codifica y ejecuta el salto (i4) se van ejecutando las i1, i2 e i3

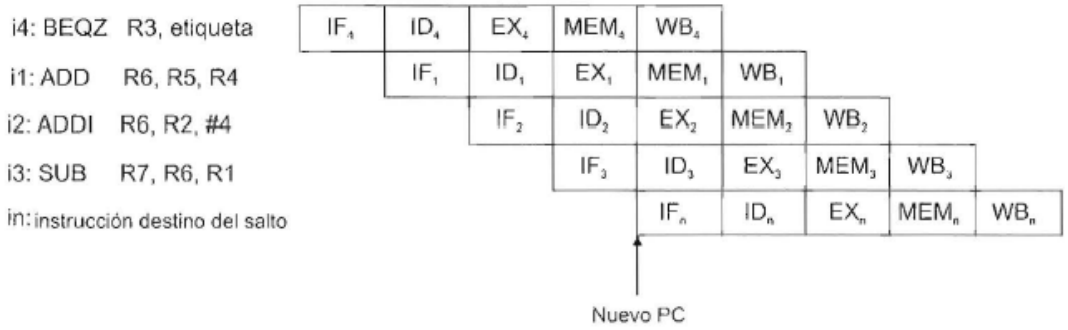
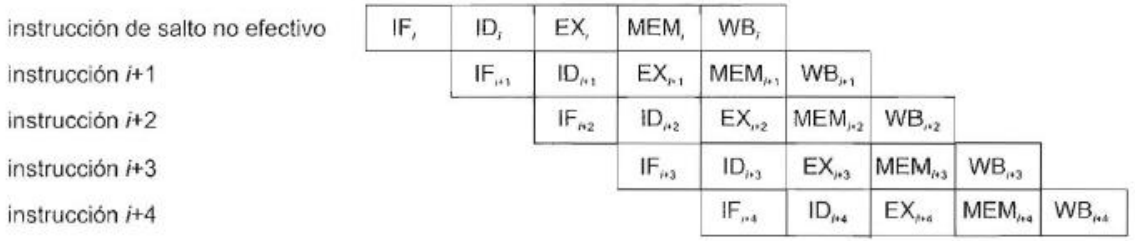


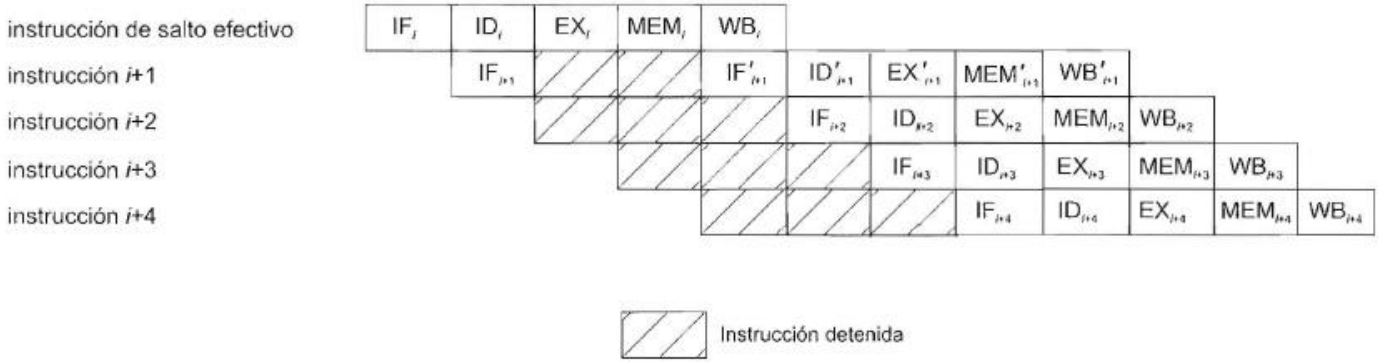
Figura 1.34: Ejemplo de aplicación de la técnica de salto retardado.

Ejecución especulativa

- Predecir el salto como no efectivo.
- En caso que sea efectivo , se detiene la segmentación y hay que recomenzar la búsqueda de la instrucción destino del salto



(a)



La etapa IF'$_{i+1}$ se corresponde con la instrucción destino del salto (i+1).

(b)

Figura 1.35: Esquema de predecir un salto como no efectivo y la secuencia de la segmentación cuando no es efectivo (a) y cuando es efectivo (b).

Técnica de adelanto del cálculo de la dirección (por predicción)

- Con Hardware adicional en la etapa ID con un sumador adicional

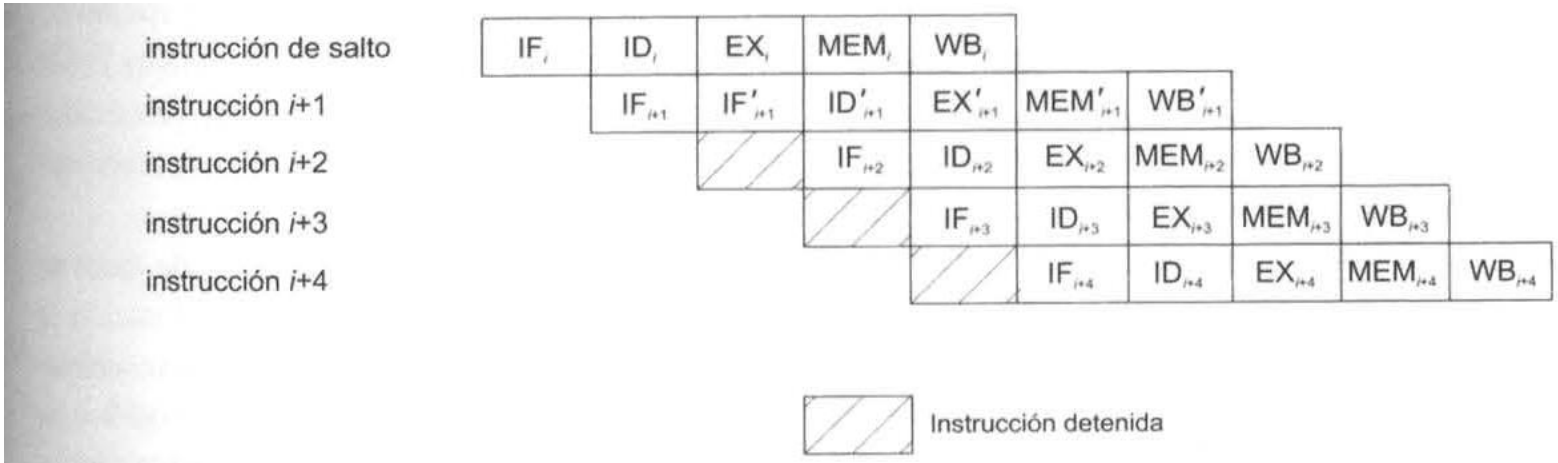


Figura 1.36: Detención real de la ASG después de un riesgo de control cuando se tiene hardware adicional para tratar el salto en la etapa ID.

1.9. Planificación dinámica: Algoritmo de Tomasulo

- Técnica de segmentación estática:
  - No busca o emite otra instrucción hasta que se soluciona la dependencia
  - Emite las instrucciones en orden,
    - Si se detiene una instrucción, las posteriores no pueden avanzar aunque no tengan dependencia de esta

Técnica de segmentación dinámica

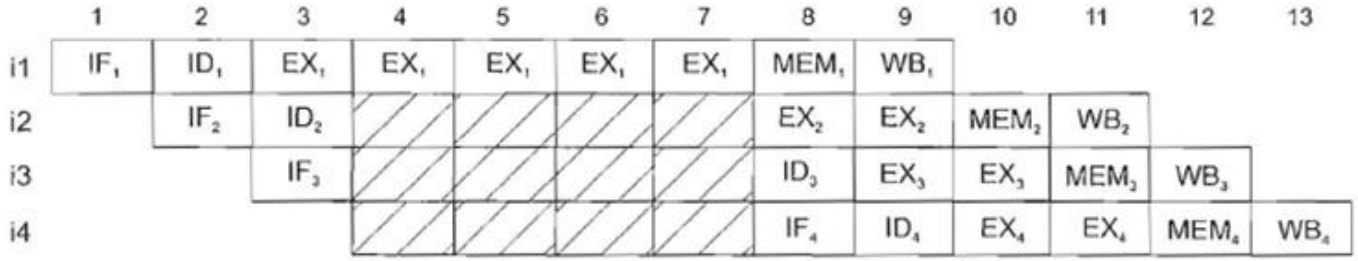
- El hardware reorganiza la ejecución de la instrucción, para reducir las detenciones manteniendo el flujo de datos y la consistencia de procesador y memoria
- Mediante la planificación dinámica se comprueban los **riesgos estructurales cuando** se **decodifica** la **instrucción.**
  - Este hecho conlleva que todavía se mantiene el orden del programa, pero las instrucciones comienzan a ejecutarse tan pronto como todos sus operandos están disponibles.
    - De esta manera, el procesador segmentado realizaría ejecución de instrucciones fuera de orden, lo que implica terminación fuera de orden
  - Se necesitan disponer de múltiples unidades funcionales
    - Hay que diferenciar entre distribuir las instrucciones hacia las unidades funcionales y comenzar su ejecución.
- Para permitir le ejecución fuera de orden hay que desdoblar la etapa ID en dos fases
  - ***Decodificación*** **(ID,** ***Instruction Decoding):***
    - Decodificación de instrucciones y comprobación de los riesgos estructurales.
  - ***Emisión*** **(II,** ***Instruction Issue):***
    - La instrucción espera hasta que no haya riesgos de tipo RAW y cuando estén listos todos los operandos fuente, se leen y se emite la instrucción hacia la unidad funcional.

Ejemplo con y sin planificación

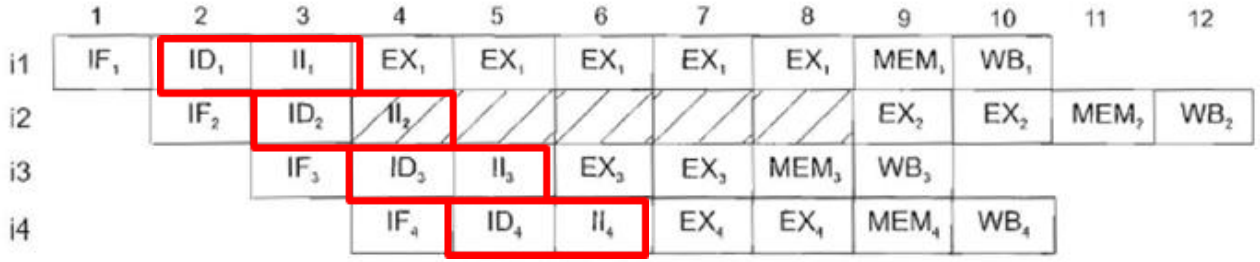
Latencia de unidad de multiplicación, 5 ciclos.
Latencia de unidad de suma, 2 ciclos.

i1: MULTD F0, F2, F6
i2: ADDD F4, F0, F2
i3: ADDD F8, F2, F6
i4: SUBD F10, F6, F2

(a)



(b) Sin planificación



(c) Planificación dinámica

Instrucción detenida

Unidad de coma flotante IBM 360/91: algoritmo de Tomasulo

Dos unidades funcionales:
- Suma de coma flotante.
- Multiplicación/división de coma flotante.

Los buffers de coma flotante (FB).
- Los datos de memoria se cargan en uno de los 6 registros FB.

Los buffers de almacenamiento de datos (SDB).
- El resultado a almacenar en memoria se coloca en uno de los 3 registros SDB.
- Una unidad independiente accede a los SDB para terminar el almacenamiento del resultado en la ubicación de memoria correspondiente.

Mediante estos dos ficheros de registros adicionales FB y SDB, la FPU admite instrucciones de almacenamiento-registro y registro-almacenamiento, funcionando como una máquina registro a registro.

Cola de operaciones de coma flotante FLOS (Floating Point Operation Stack).

Estaciones de reserva (RS, Reservation Stations).
- La unidad de instrucción decodifica todas las instrucciones y pasa en orden a la cola de operaciones de coma flotante FLOS (Floating Point Operation Stack).
- A continuación, ya en la FPU, las instrucciones se emiten en orden desde la FLOS a las dos unidades funcionales.

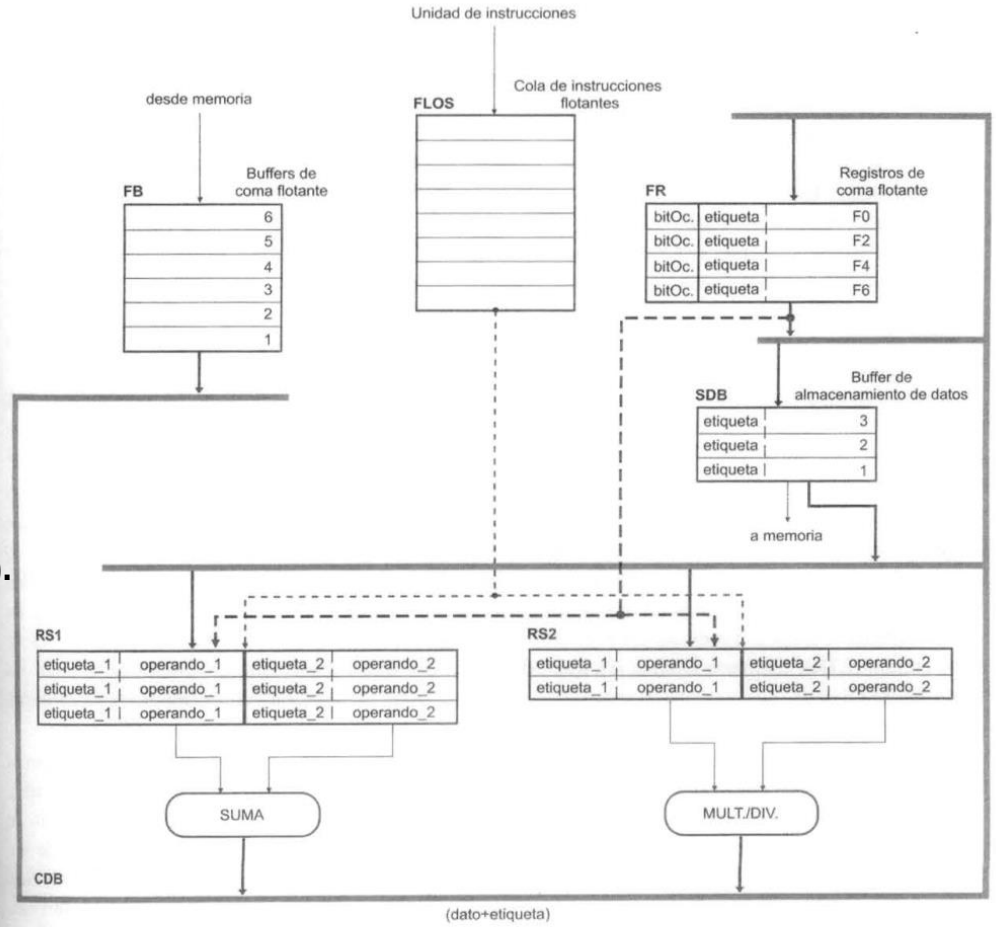


Figura 1.38: Diseño modificado de la unidad de coma flotante del IBM 360/91 con el algoritmo de Tomasulo.

IBM/360

- El uso de estaciones de reserva y del fichero de registros centralizado FR da lugar a dos propiedades:
  - La detección de riesgos y el control de la ejecución están distribuidos ya que la información mantenida en las estaciones de reserva de cada unidad funcional determina si una instrucción puede empezar su ejecución en esa unidad.
  - Los resultados pueden pasar directamente a la unidad funcional desde las estaciones de reserva donde estaban almacenados, en vez de acceder a ellos a través de los registros.
    - Para realizar este adelantamiento se utiliza un bus de datos común (CDB, Common Data Bus).
- Cuando la FLOS envía una instrucción a una unidad funcional, la asigna una estación de reserva y comprueba si están disponibles los operandos necesarios.
  - Si un operando se encuentra disponible en el FR, el contenido de ese registro del FR se copia a la estación de reserva;
    - En caso contrario, se copia una etiqueta para indicar que esa instrucción está a la espera de un operando pendiente de ser generado.
    - La etiqueta indica de dónde procederá el operando pendiente, pudiendo ser de una instrucción que está actualmente en una de las cinco estaciones de reserva o de uno de los seis registros de FLB.
- Cada estación de reserva contiene dos campos por operando, donde uno corresponde a la etiqueta y otro al valor del operando.
  - Los cuatro FR y los tres registros del SDB también llevan un campo etiqueta asociado.
  - En las estaciones de reserva si un campo de operando contiene datos reales, entonces su campo de etiqueta se establece a cero. En caso contrario, su campo de etiqueta identifica el origen del que procede el operando pendiente.
- Al mismo tiempo, se establece a 1 el bit de ocupado asociado con el registro destino del resultado en el FR, lo que indica que existe una actualización pendiente de ese registro, y el valor de la etiqueta que identifica la estación de reserva a la que se distribuye la instrucción se escribe en el campo de etiqueta del FR correspondiente.

Ejemplo de ejecución

- Para reducir el número de ciclos máquina se permite que la FLOS distribuya hasta dos instrucciones en cada ciclo según el orden del programa.
- Una instrucción puede comenzar su ejecución en el mismo ciclo en que se distribuye a una estación de reserva.
- La operación suma tiene una latencia de dos ciclos y la de multiplicación de tres ciclos.
- Se permite que una instrucción reenvíe su resultado a instrucciones dependientes durante su último ciclo de ejecución.
  - De esta forma, una instrucción a la espera de un resultado puede comenzar su ejecución en el siguiente ciclo si detecta una coincidencia.
- Los valores de etiqueta 01 , 02 , 03 se utilizan para identificar las tres estaciones de reserva de la unidad funcional de suma, mientras que 04 y 05 se utilizan para identificar las dos estaciones de reserva de la unidad funcional de multiplicación/división.
  - Estos valores de etiqueta son los ID de las estaciones de reserva.
- Inicialmente, el valor de los registros es F0=6.0, F2=3.5, F4=10.0 y F6=7.8.

Ciclo 1

```
i1: ADDD   F4, F0, F6
i2: MULTD  F2, F0, F4
i3: ADDD   F4, F4, F6
i4: MULTD  F6, F4, F2
```

(a) Secuencia de instrucciones

Ciclo 1: Distribución de i1 e i2 (en orden)

FR

	bitOc.	etiqueta	dato
F0			6,0
F2	Sí	04	3,5
F4	Sí	01	10,0
F6			7,8

RS

	ID	etiqueta_1	operando_1	etiqueta_2	operando_2
i1	01	00	6,0	00	7,8
	02				
	03				

i1 → SUMA

RS

	ID	etiqueta_1	operando_1	etiqueta_2	operando_2
i2	04	00	6,0	01	.....
	05				

MULT./DIV.

Ciclo 1:

- 1. Se distribuyen i1 e i2 a RS01 (suma) y RS04 (mul/div).
- 2. Los registros destino son F4 y F2.
  - Los bits de ocupado FR se activan.
- 3. Como i1 a RS01 (1º libre de RS suma) · etiqueta de FR =01 a F4. (Indica de qué RS se obtendrá)
- 4. Como i2 a RS02 (1º libre de RS mul/div) · etiqueta de FR =04 a F2. (Indica de qué RS se obtendrá)
- 5. i1: ADDD F4, F0, F6 ·
  - Como están disponibles F0 y F6 sus etiquetas de los operandos de RS01 se ponen a 0.
- 6. Comienza la ejecución de i1.
- 7. I2: MULTD F2, F0, F4 ·
  - i2 necesita F4 que viene de i1. Por lo tanto la etiqueta de RS04 = 01 (De la RS01 donde está i1)

Ciclo 2

i1: ADDD F4, F0, F6

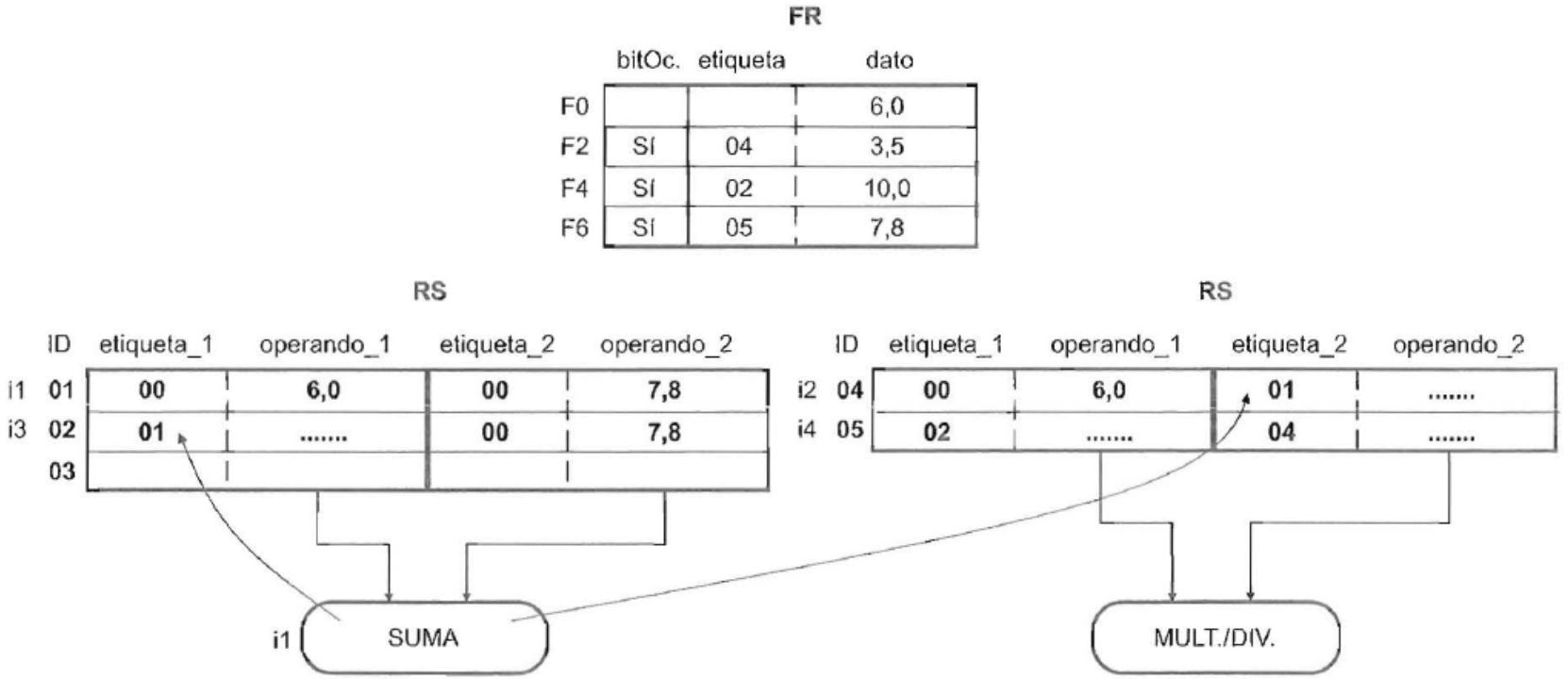
i2: MULTD F2, F0, F4

i3: ADDD F4, F4, F6

i4: MULTD F6, F4, F2

(a) Secuencia de instrucciones

Ciclo 2: Distribución de i3 e i4 (en orden)



Ciclo 2

- 1. i3: ADDD F4, F4, F6
- 2. i4: MULTD F6, F4, F2
- 3. Se distribuyen i3 e i4 a RS02 (suma) y RS05 (mul/div).
- 4. Para i3 se necesita el resultado de i1 (F4). En RS02 se pone etiqueta a 01 (Indica que depende de RS01).
- 5. Para i4 se necesita el resultado de i2 (F2) e i3 (F4). En RS05 pone etiquetas 02 y 04 (Indica que depende de RS02 y RS04).
- 6. Como el destino de i3 es F4, se actualiza la etiqueta de FR de 01 a 02. Indica que RS02 deberá actualizar el valor de F4. Bit de ocupado sigue activo.
- 7. Cuando i4 se distribuye a RS05, el bit de ocupado de FR se activa y su etiqueta (FR) se pone a 05 (Indica que es RS05 el que debe actualizar le valor de F6.
- 8. Termina i1, emite su ID (RS01) y su resultado a CDB. Todos los campos con etiqueta = 01 se actualizan al valor de F4.
  - Actualizando RS02 (i3) con valor de operando a 13,8 y etiqueta a 00.
  - Actualizando RS04 (i2) con valor de operando a 13,8 y etiqueta a 00.

Ciclo 3

Ciclo 3:

i1: ADDD F4, F0, F6
i2: MULTD F2, F0, F4
i3: ADDD F4, F4, F6
i4: MULTD F6, F4, F2

(a) Secuencia de instrucciones

FR

	bitOc.	etiqueta	dato
F0			6,0
F2	Si	04	3,5
F4	Si	02	10,0
F6	Si	05	7,8

RS

	ID	etiqueta_1	operando_1	etiqueta_2	operando_2
	01				
i3	02	00	13,8	00	7,8
	03				

i3 SUMA

RS

	ID	etiqueta_1	operando_1	etiqueta_2	operando_2
i2	04	00	6,0	00	13,8
i4	05	02	.....	04	.....

i2 MULT./DIV.

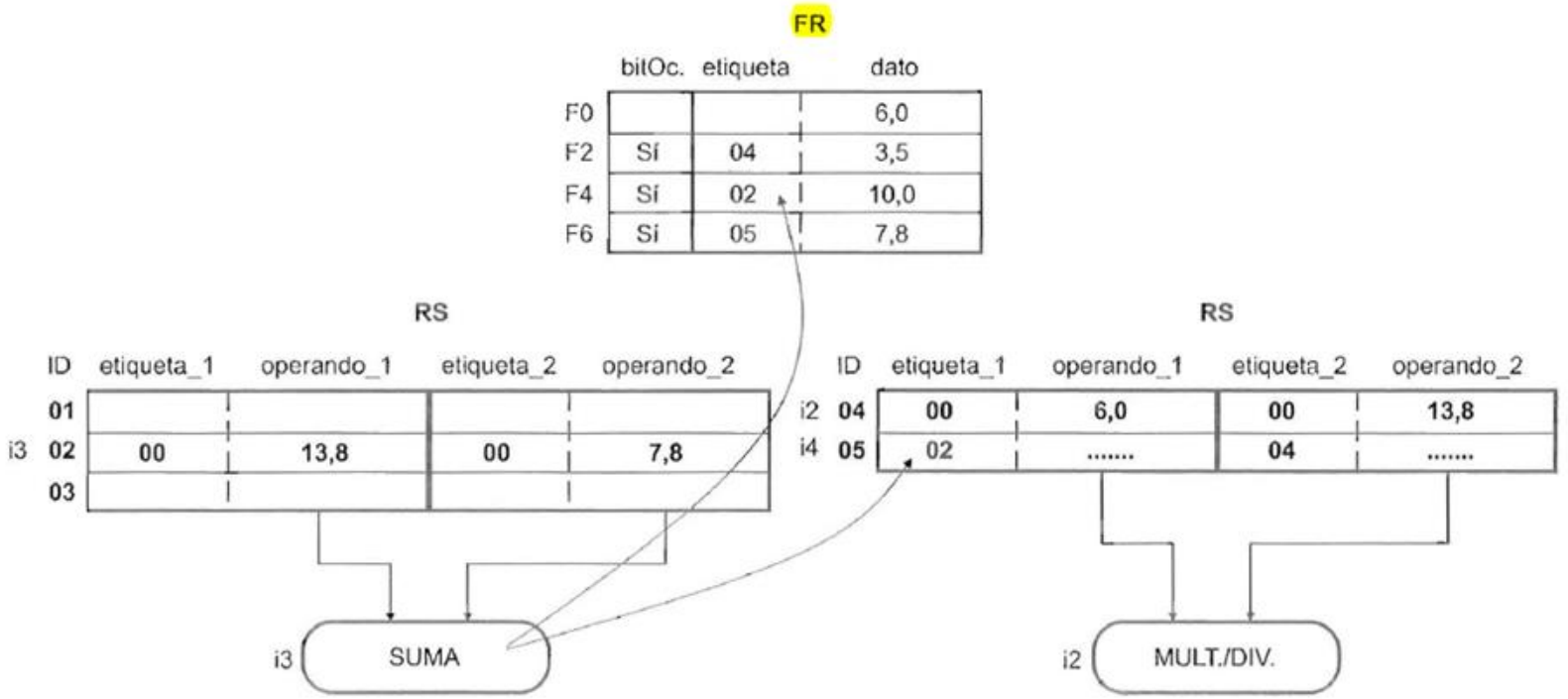
- Comienza a ejecutarse i3 e i2 en unidad de suma y unidad de mul/div.
- Se libera RS01.

Ciclo 4

i1: ADDD F4, F0, F6
i2: MULTD F2, F0, F4
i3: ADDD F4, F4, F6
i4: MULTD F6, F4, F2

(a) Secuencia de instrucciones

Ciclo 4:



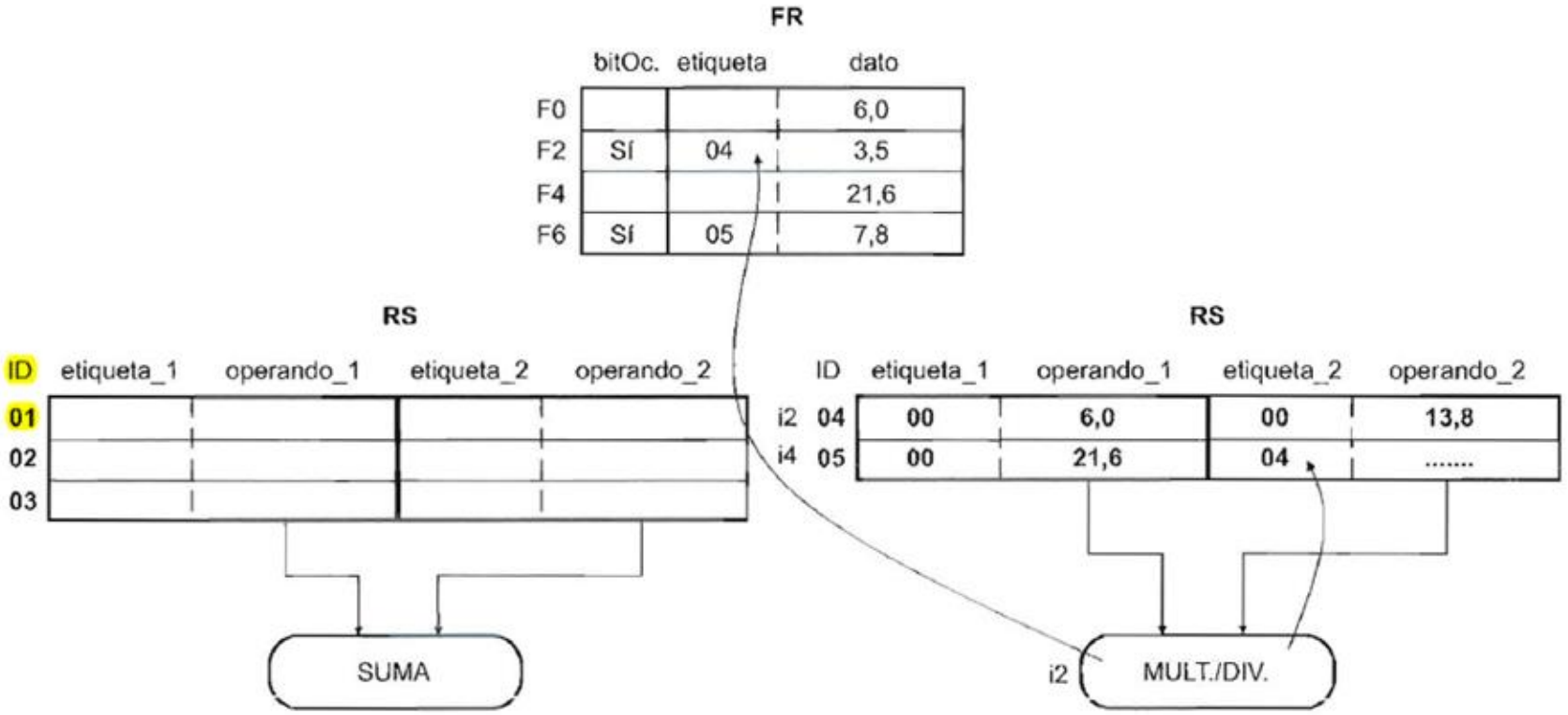
- Finaliza i3 y emite resultado a CDB y la etiqueta 02.
- Como RS05 tiene etiqueta 02 → se pone el resultado de la suma (21,6) y el valor de su etiqueta (RS05) se pone a 00.
- Se libera RS02.

Ciclo 5

i1: ADDD F4, F0, F6
i2: MULTD F2, F0, F4
i3: ADDD F4, F4, F6
i4: MULTD F6, F4, F2

(a) Secuencia de instrucciones

Ciclo 5:



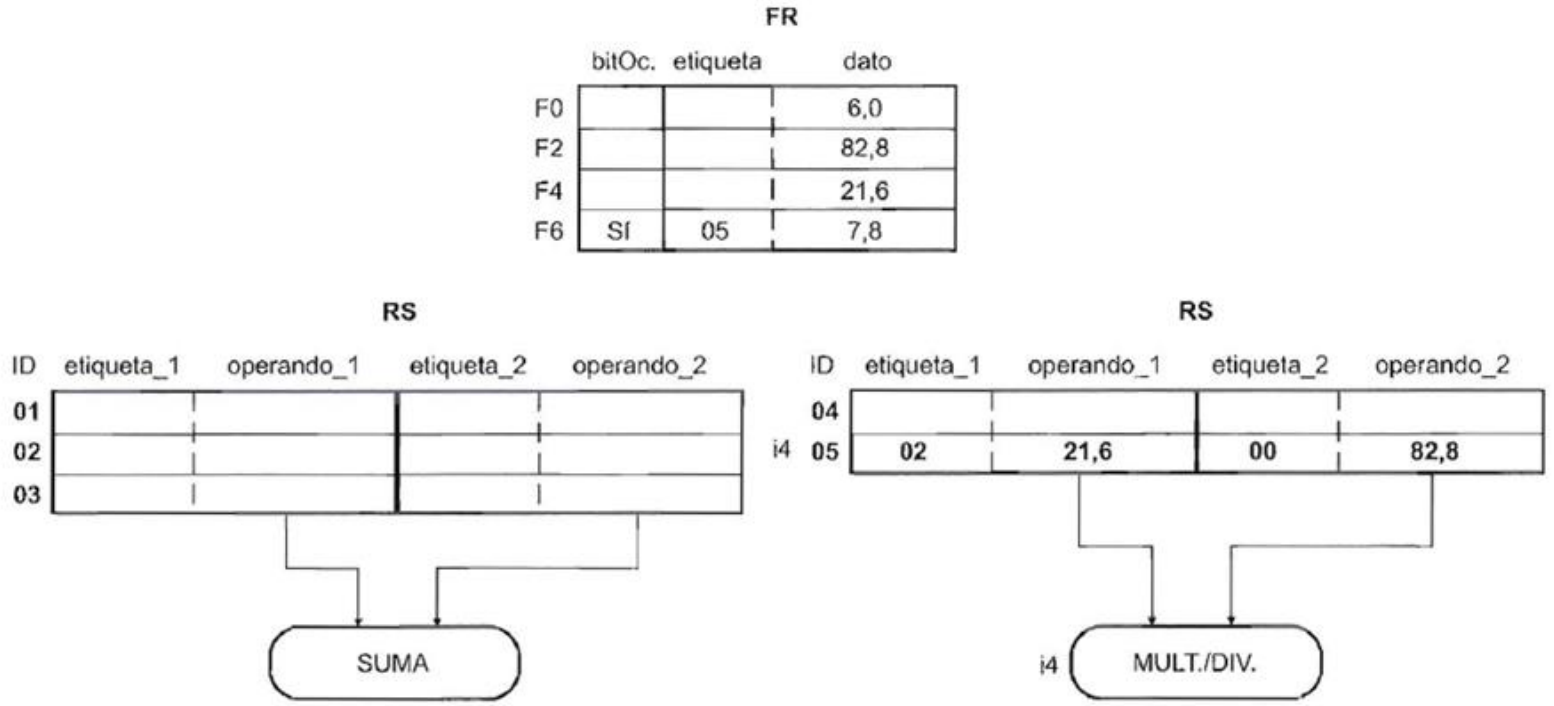
- Finaliza i2 y emite resultado a CDB y la etiqueta 04.
- Como RS05 tiene etiqueta 04 → se pone el resultado de la multiplicación (82,8) y el valor de su etiqueta (RS05) se pone a 00.
- Se libera RS04.

Ciclo 6

i1: ADDD F4, F0, F6
i2: MULTD F2, F0, F4
i3: ADDD F4, F4, F6
i4: MULTD F6, F4, F2

(a) Secuencia de instrucciones

Ciclo 6:



- Comienza ejecución de i4 y acaba en el ciclo 8.