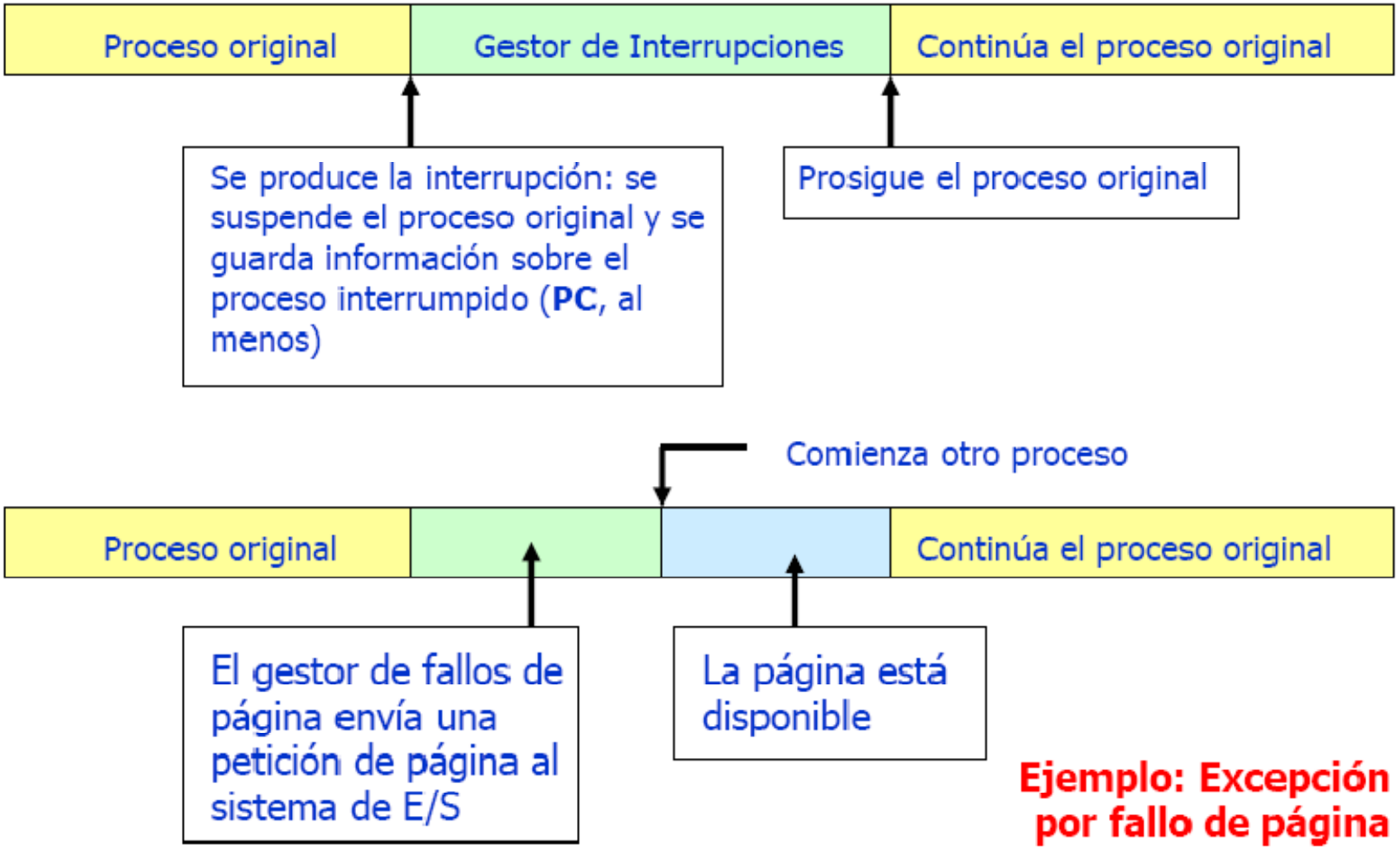
Arquitectura e Ingeniería de Computadores

Tema 2 – Segmentación y superescalares
clase 2: Interrupciones y optimización

Ingeniería en Informática

Departamento de Tecnología Informática y Computación

- Interrupciones: las interrupciones y las excepciones afectan negativamente al rendimiento del procesador al romper el flujo de instrucciones (igual que en los riesgos de control o datos)



- Taxonomía

	Indican una condición de error y dan paso a una rutina de recuperación	Interrupciones críticas para comunicarse con el S.O.
Fuente Interna a la CPU	Excepciones (A)	Memoria Virtual Instrucciones no implementadas (B)
Fuente Externa	Fallo hardware (C)	Temporización E/S (D)

- Ejemplos

Interrupciones (incluyen las excepciones y los traps):

- Petición de dispositivo de E/S (D)
- Llamada a un servicio del OS desde un programa de usuario (B)
- Trazas de ejecución de instrucciones (B)
- Interrupción solicitada por el programador (breakpoint) (B)
- Overflow o underflow en aritmética entera (A)
- Anomalía en aritmética de Coma Flotante (A)
- Fallo de página (B)
- Acceso a memoria no alineado (si se precisa que sea un acceso alineado) (A)
- Violación de la protección de memoria (A)
- Uso de instrucción no definida (B)
- Error en el hardware, fallo de potencia,.... (C)

	Indican una condición de error y dan paso a una rutina de recuperación	Interrupciones críticas para comunicarse con el S.O.
Fuente Interna a la CPU	Excepciones (A)	Memoria Virtual Instrucciones no implementadas (B)
Fuente Externa	Fallo hardware (C)	Temporización E/S (D)

Las interrupciones son difíciles de manejar en un procesador segmentado (superescalar o no) dado que el solapamiento entre instrucciones hace más difícil determinar el momento en que una instrucción puede cambiar de forma segura el estado de la máquina.

Posibilidades

Excepciones Síncronas/Asíncronas: Si el evento se produce en el mismo lugar cada vez que se ejecuta el programa es síncrono. Los eventos asíncronos se originan en los dispositivos externos al procesador y la memoria. Los eventos asíncronos son más fáciles de manejar puesto que se pueden atender después de que se complete la instrucción en curso.

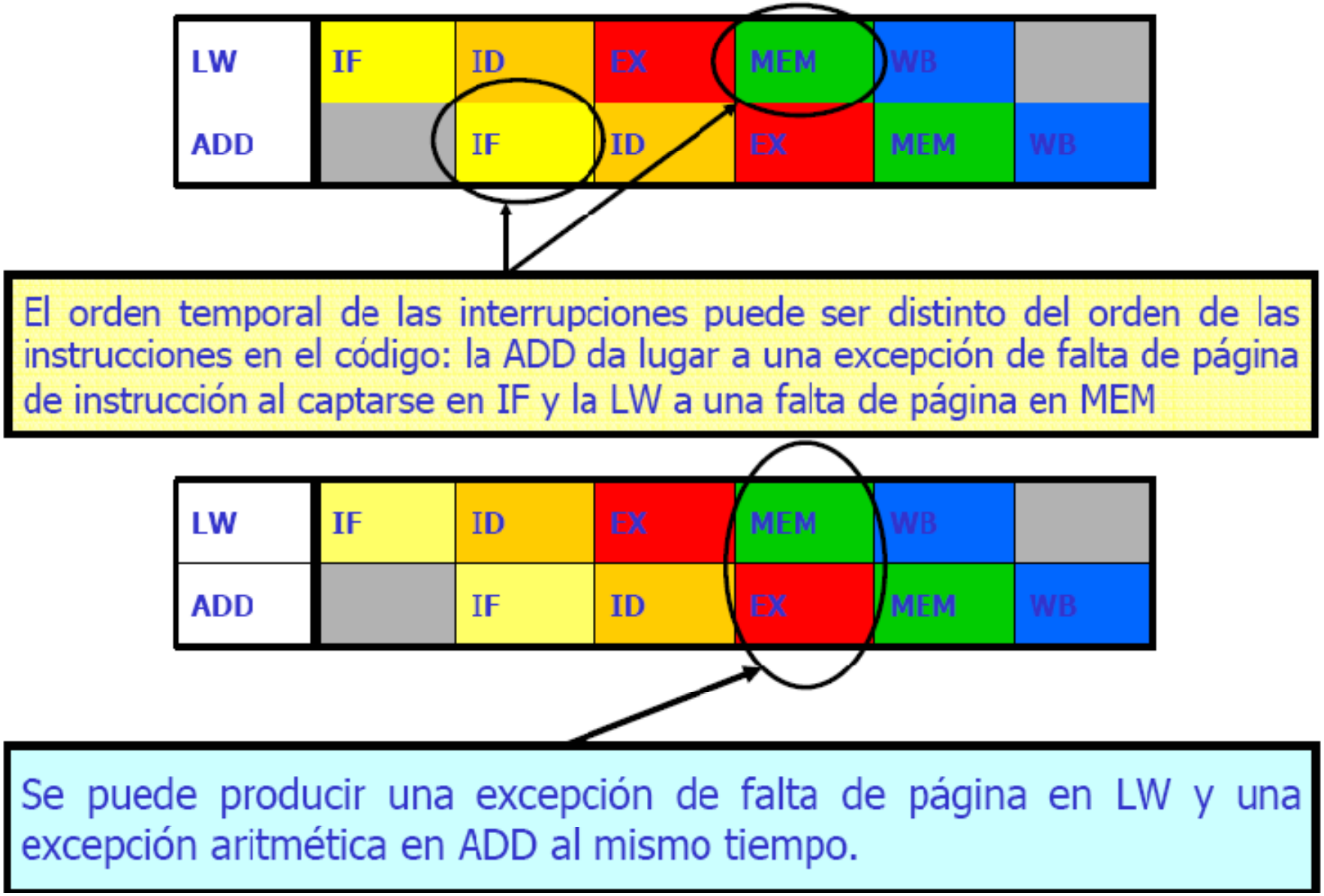
Solicitadas por el usuario o sobrevenidas: Si las solicita el usuario no son realmente excepciones puesto que son predecibles (se tratan como excepciones porque utilizan el mismo mecanismo para almacenar y restaurar el estado). Las sobrevenidas se deben a un evento que no controla el programa.

Enmascarables por el usuario o no enmascarables

Dentro de una instrucción o entre instrucciones: Según el evento impide que se termine la ejecución de la instrucción porque ocurre en mitad de la misma (son siempre excepciones síncronas y son difíciles de procesar porque la instrucción debe pararse y empezar de nuevo) o bien ocurren entre instrucciones

Excepciones para terminar o con continuación ('terminate/resume') que terminan el programa (***catastróficas***) o que permiten continuar el programa tras ser procesadas.

- Problemas en el procesamiento de las interrupciones



- Las interrupciones pueden ser *precisas o imprecisas*, según se atiendan respetando el orden de ejecución de las instrucciones o no

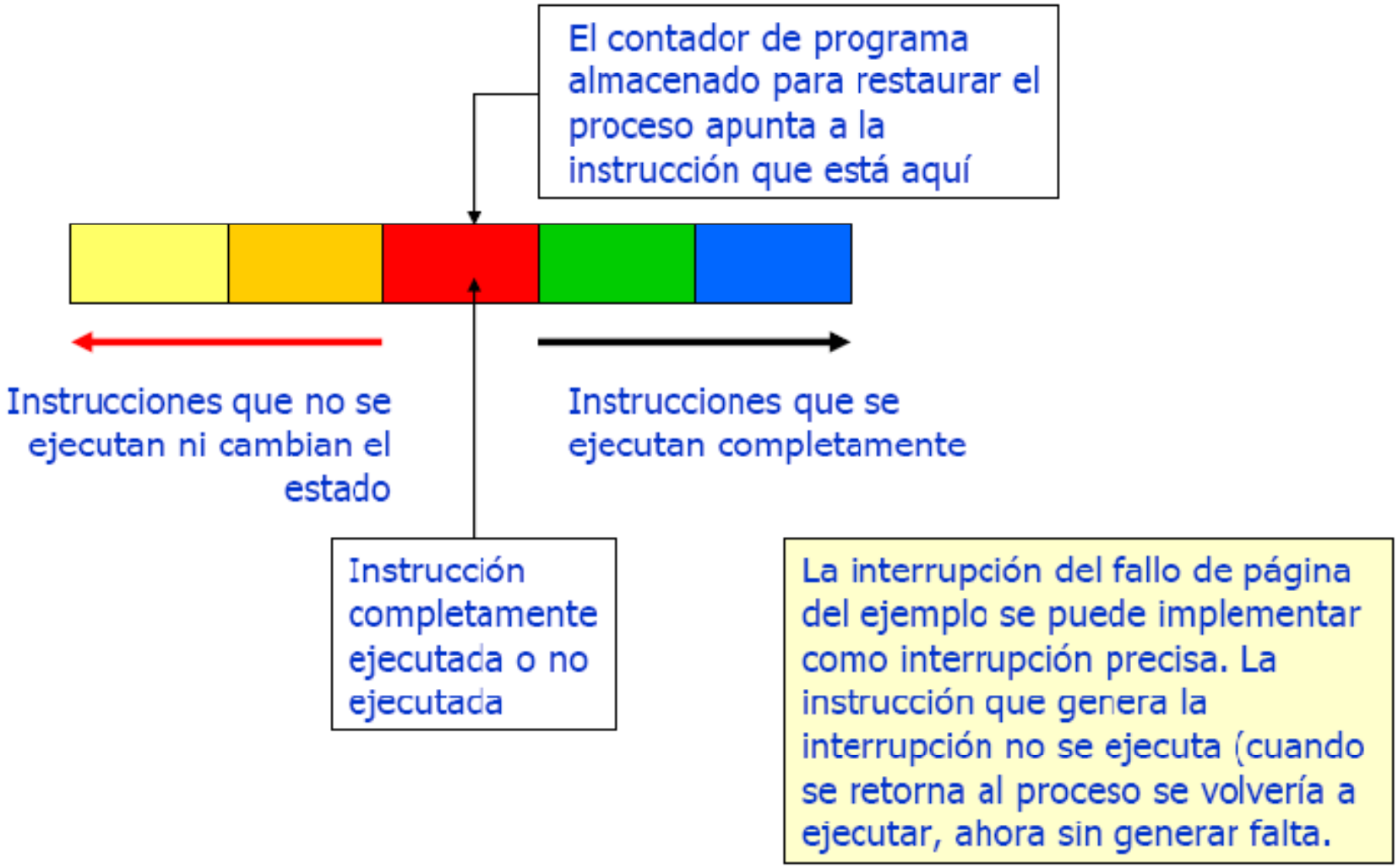
Interrupciones Precisa:

Permiten garantizar que, después de una interrupción no catastrófica, el proceso interrumpido continúe correctamente.

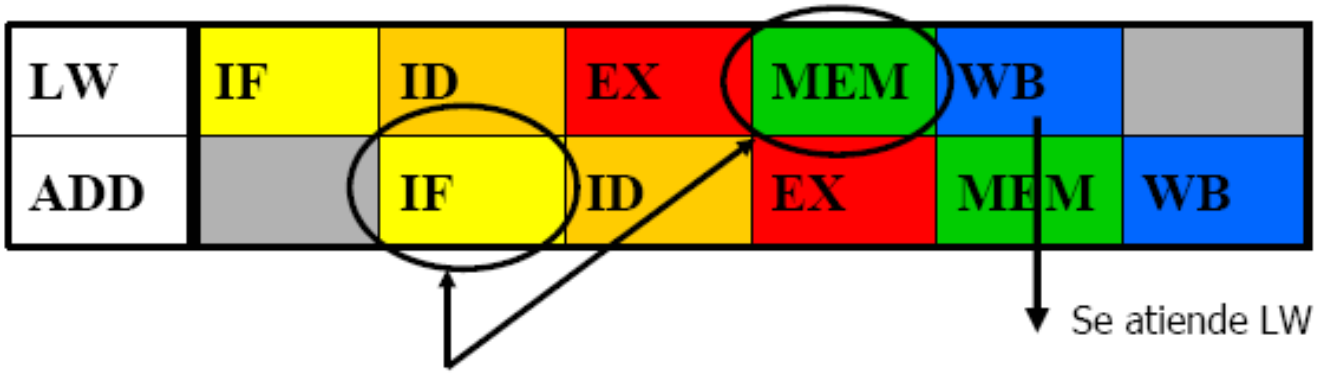
El estado de la máquina en el momento de la interrupción es idéntico al que existiría si la ejecución fuese secuencial. Ese estado se llama estado preciso y cumple las siguientes condiciones (Smith y Pleszkun, 85):

- Todas las instrucciones que se emitieron antes de la instrucción indicada por el PC almacenado (para continuar el proceso interrumpido) se han completado.
- Las instrucciones posteriores al valor indicado por el PC almacenado no se ejecutan y no han cambiado el estado del procesador..
- Si la interrupción fue ocasionada por una instrucción, el contador de programa apunta a esa instrucción, que puede ejecutarse completamente o no.

Implementación interrupción precisa



Ejemplo tratamiento preciso



Interrupciones Precisas: Debe atenderse **primero a la excepción causada por LW (instrucción i) y luego a la producida por ADD (instrucción i+1).**

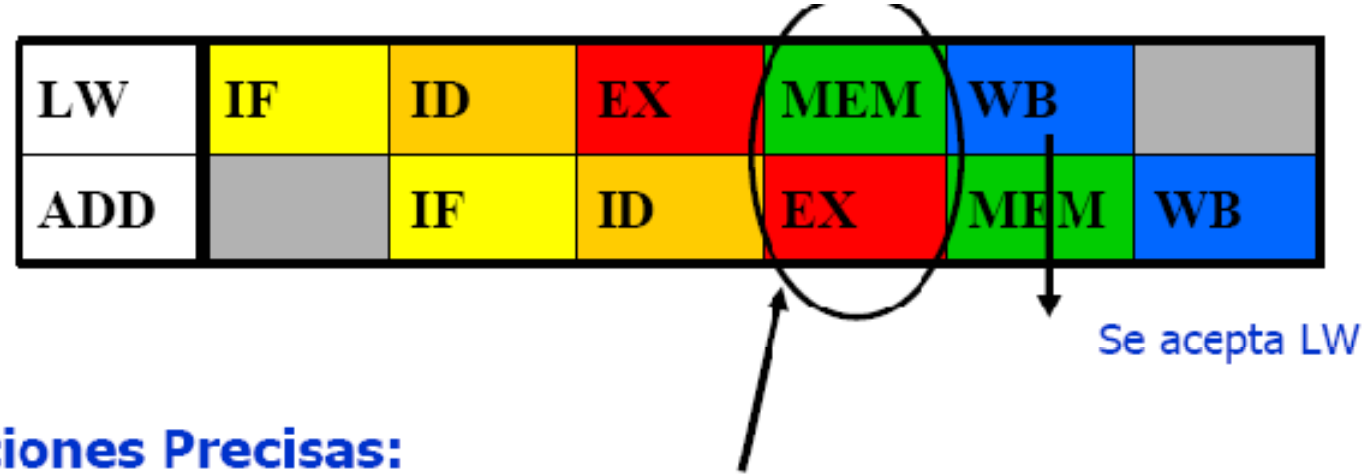
Las excepciones no se atienden cuando se producen sino **cuando la instrucción llega a una etapa determinada (usualmente la última etapa (WB)):**

De esta forma se consigue una *serialización* del estado de las instrucciones

Ejemplo de Procesamiento de Interrupciones precisas

- El hardware indica las excepciones que se van produciendo en un vector asociado a cada instrucción y que va pasando con ella de una etapa a otra.
- Si una instrucción tiene activo su vector de estado de excepción, se inhabilita toda señal de escritura (se impiden las escrituras en memoria y en registros: el cauce debería hacerlas siempre en las últimas etapas).

- Ejemplo de tratamiento preciso



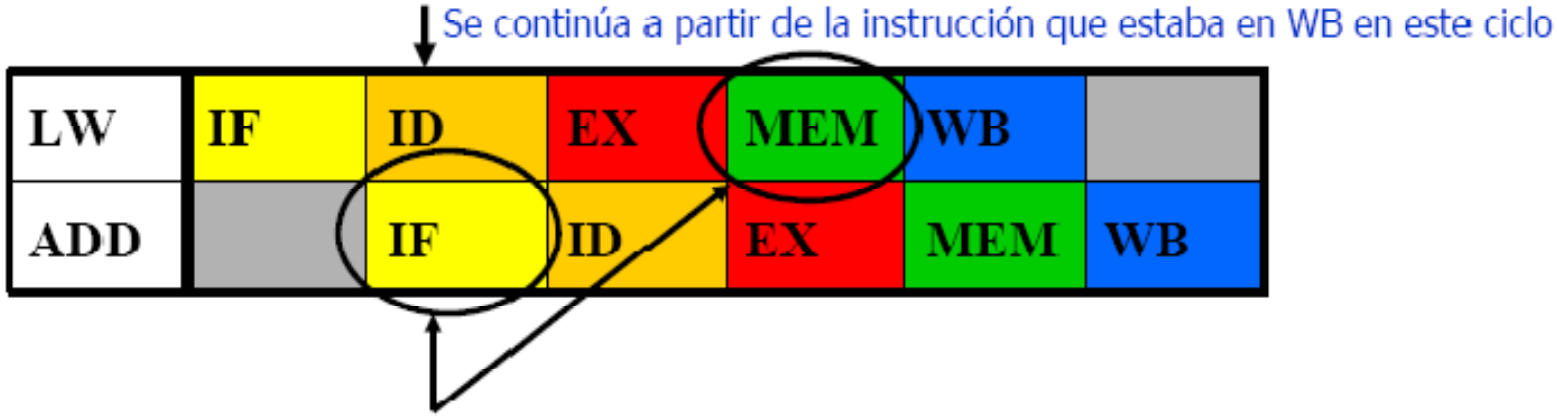
Interrupciones Precisas:

Se acepta la correspondiente a la instrucción LW y se reinicia la ejecución de instrucciones (por LW). La segunda excepción se volverá a producir pero no la primera, ya que se ha traído la página correspondiente.

Una organización adecuada de las etapas del cauce es esencial para facilitar el procesamiento de la instrucción:

Que las escrituras se hagan en (y sólo en) la última etapa facilita la implementación

Tratamiento impreciso



Interrupciones Imprecisas: Debe atenderse **primero a la excepción causada por ADD (instrucción i+1)**, si en el momento en que se atiende la interrupción no se haya producido la interrupción producida por LW.

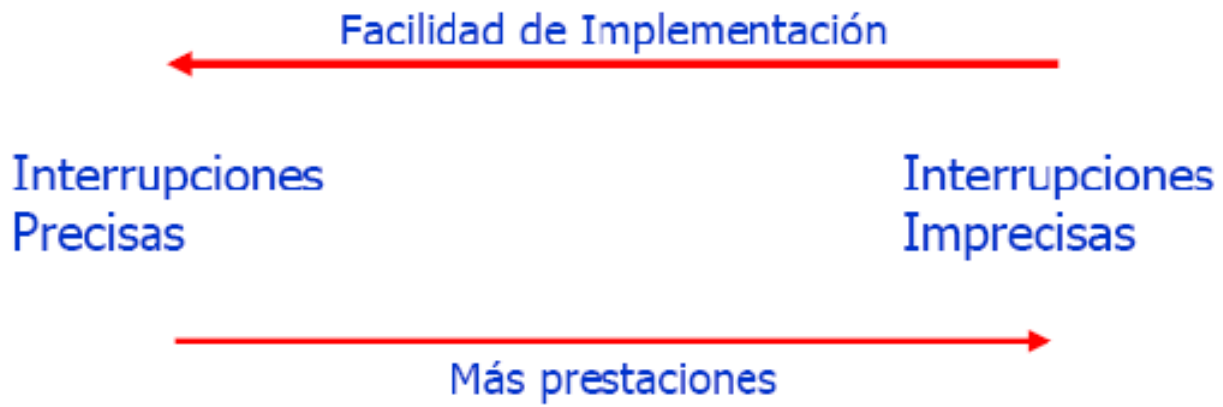
Las excepciones se atienden cuando se producen (lo más pronto posible)

Ejemplo de Procesamiento de Interrupciones Imprecisas

- Se guarda el estado que ha modificado la última instrucción que estaba en la etapa WB (que ha podido escribir en el banco de registros o en memoria) y el PC apunta a la siguiente instrucción (por ella se reanuda el programa interrumpido).
- Se vuelve a llenar el cauce desde el valor de PC que se había guardado (vuelven a pasar por etapas que ya habían visitado. En el ejemplo, se atiende (y se soluciona) el problema de ADD, y luego (seguramente) se producirá el problema con LW.

La forma de implementar las instrucciones que se ha descrito sólo se puede utilizar en el caso de cauces donde se produce finalización ordenada.

Al estudiar los Superescalares se estudiarán los aspectos relativos a las instrucciones con finalización desordenada



Optimización en la implementación de las interrupciones

No todos los tipos de interrupciones (A, B, C, D) requieren el mismo tipo de implementación (precisa o imprecisa)

Externas-Críticas (D): Una vez detectada la interrupción, se deja de captar instrucciones, se terminan las instrucciones del cauce y se inicia la ejecución del gestor de interrupción. Luego se continúa a partir de la instrucción a la que apuntaba PC y que no se captó.

Externas-Error (C): Se pueden implementar de forma sencilla como en el caso anterior (o congelar el estado del procesador en el momento de la interrupción – del error – para que la rutina de gestión pueda analizar las causas).

Internas-Error (A): Se pueden implementar de forma imprecisa en la mayoría de los casos, ya que el programa interrumpido no suele poder continuar. En el caso del uso de un *debugger*, sí deben implementarse como interrupciones precisas (no importa mucho si se degradan las prestaciones).

Internas-Críticas (B): Deben implementarse como interrupciones precisas, y de forma eficaz puesto que son necesarias para el funcionamiento correcto de todos los programas.

En el caso de ejecución fuera de orden y fallos de acceso a memoria (memoria virtual), se puede hacer que las instrucciones que acceden a memoria deben ejecutarse manteniendo el orden.

Implementación de los cauces

- Cauce lineal: Cada etapa está conectada sólo con una etapa anterior y otra posterior, y las entradas a procesar (instrucciones, operandos,…) se pueden introducir en el cauce al ritmo de una cada ciclo de reloj
- Cauce no lineal:
  - Hay etapas que necesitan varios ciclos de reloj.
  - Algunas etapas se vuelven a reutilizar por una misma operación.
  - Una misma operación (instrucción,..) puede utilizar más de una etapa al mismo tiempo.
  - El orden en que se visitan las etapas (y las etapas que se visitan) puede cambiar de una operación a otra (**cauces multifuncionales**).
  - Pueden existir dependencias entre las operaciones que se introducen en el cauce, de forma que el orden en que una operación visite las etapas cambie dinámicamente (**cauces dinámicos multifuncionales**)

La productividad de un cauce depende de **(1) que exista una fuente continua de operaciones** (instrucciones,..) a realizar, y **(2) de que exista un procedimiento eficaz de planificación del cauce.**

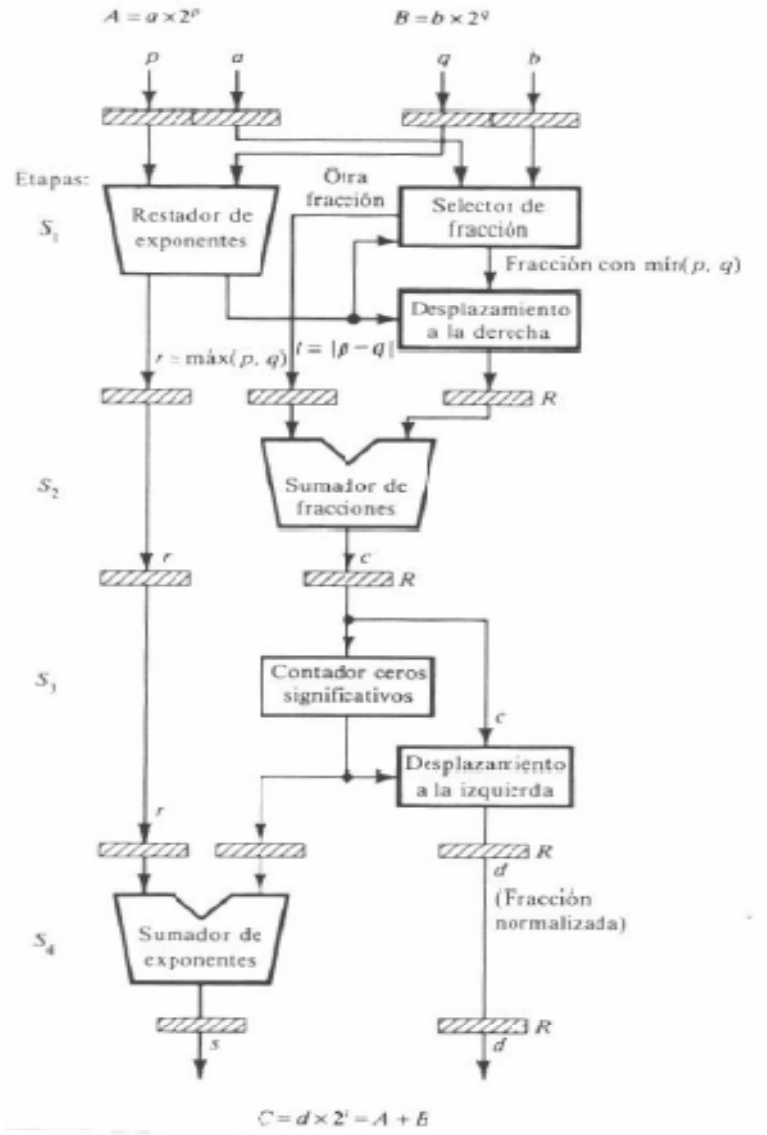
- Ejemplo de cauce lineal: sumador FP

S1: Determina el exponente mayor (r) y la diferencia de exponentes (t). Desplaza t bits a la derecha la fracción del exponente más pequeño.

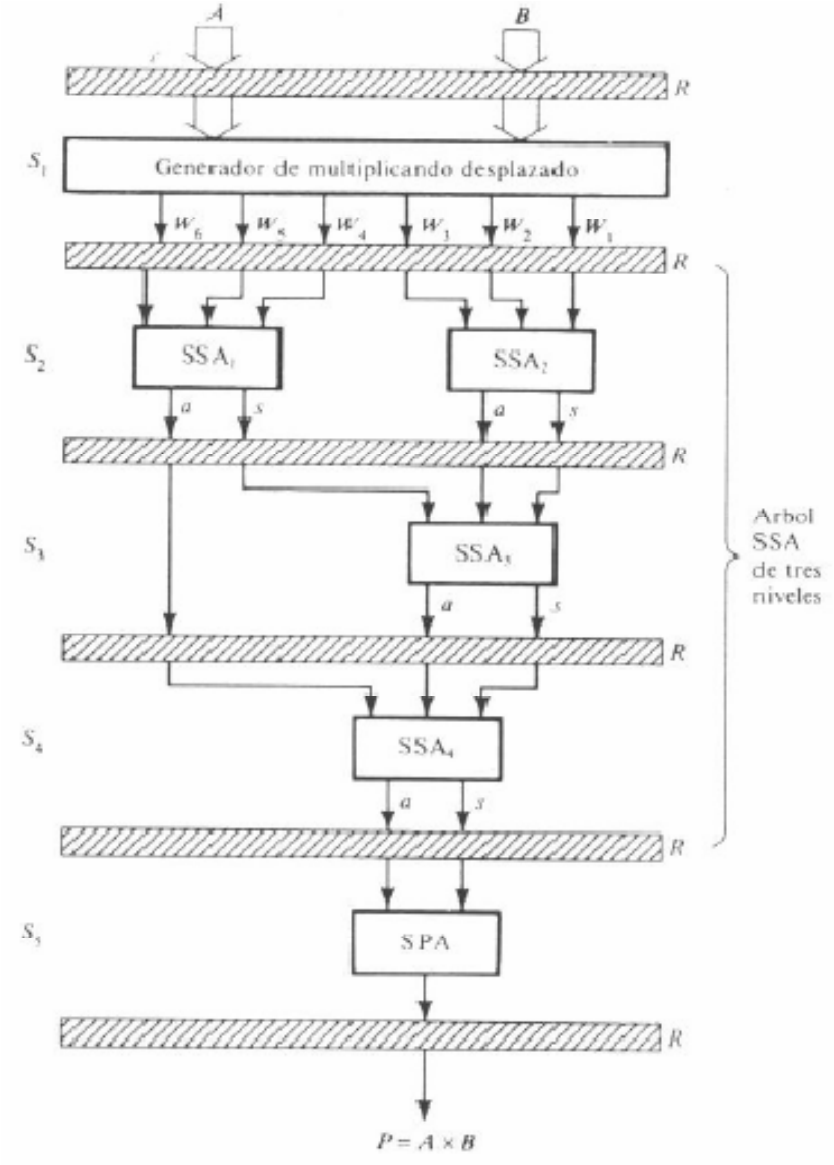
S2: Suma las dos fracciones (c)

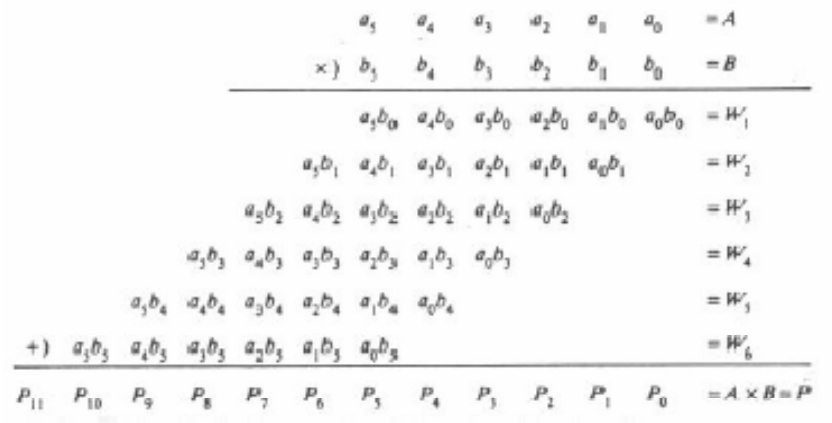
S3: Recuento del número de ceros significativos (u) de la fracción obtenida (c), que se desplaza ese número de bits hacia la izquierda, obteniéndose la fracción normalizada (d).

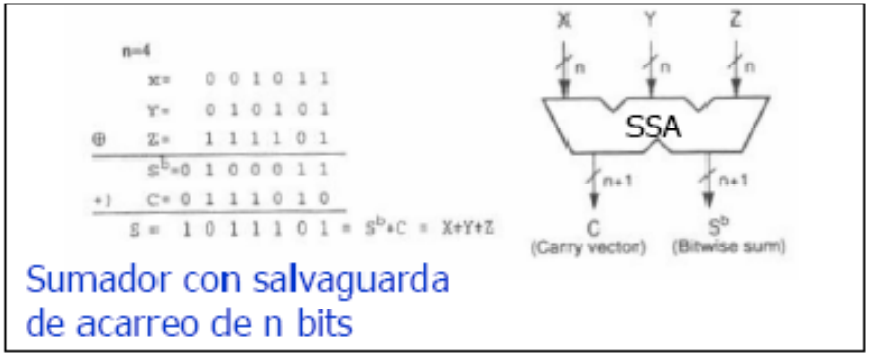
S4: Actualización del exponente mayor (s): s = r – u



Ejemplo cauce lineal: multiplicador 6bits





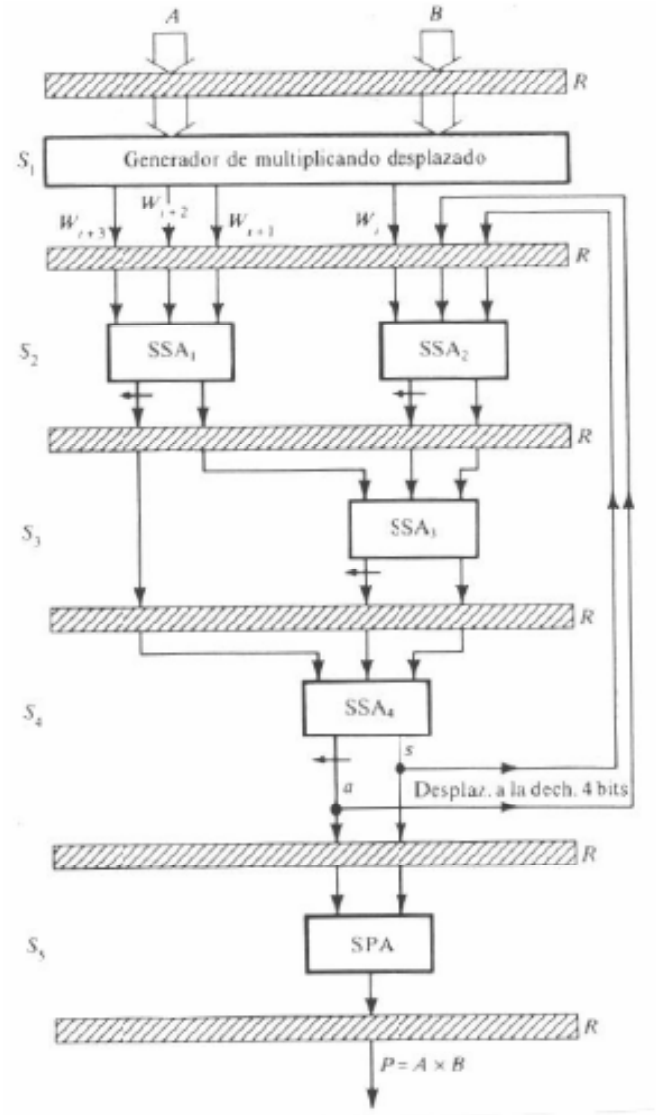


- Multiplicadores en árbol SSA: con un árbol de l niveles se pueden sumar N(l) números

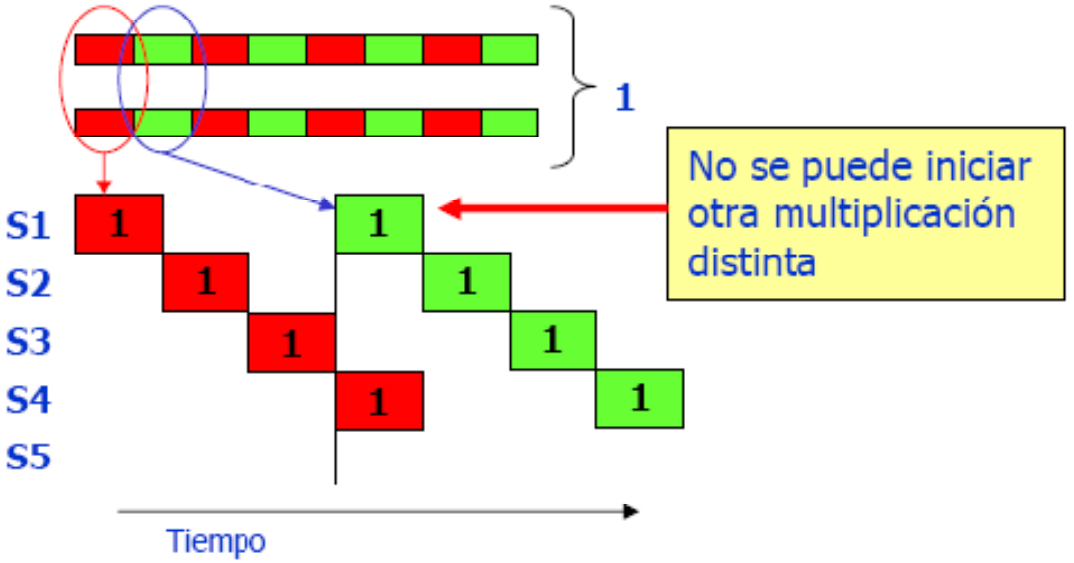
$$N(l) = \begin{cases} \left\lceil \frac{N(l-1)}{2} \right\rceil \cdot 3 + N(l-1) \bmod 2, & l > 1 \\ 3 & l = 1 \end{cases}$$

l	1	2	3	4	5	6	7	8	9	10
N(l)	3	4	6	9	13	19	28	42	63	94

Ejemplo de cauce NO lineal: multiplicador de 32 bits



- Multiplicador de 32 bits, a partir de la estructura de multiplicador de 6 bits anterior.
- Un multiplicador lineal de 32 bits necesitaría 30 SSAs.
- Los operandos a multiplicar se introducen en trozos de 4 bits (de menos a más significativos) y reutilizan etapas del cauce.



Comparación multiplicador 32bits

- Lineal => 8 niveles (30SSA) + 1 nivel (1SPA)

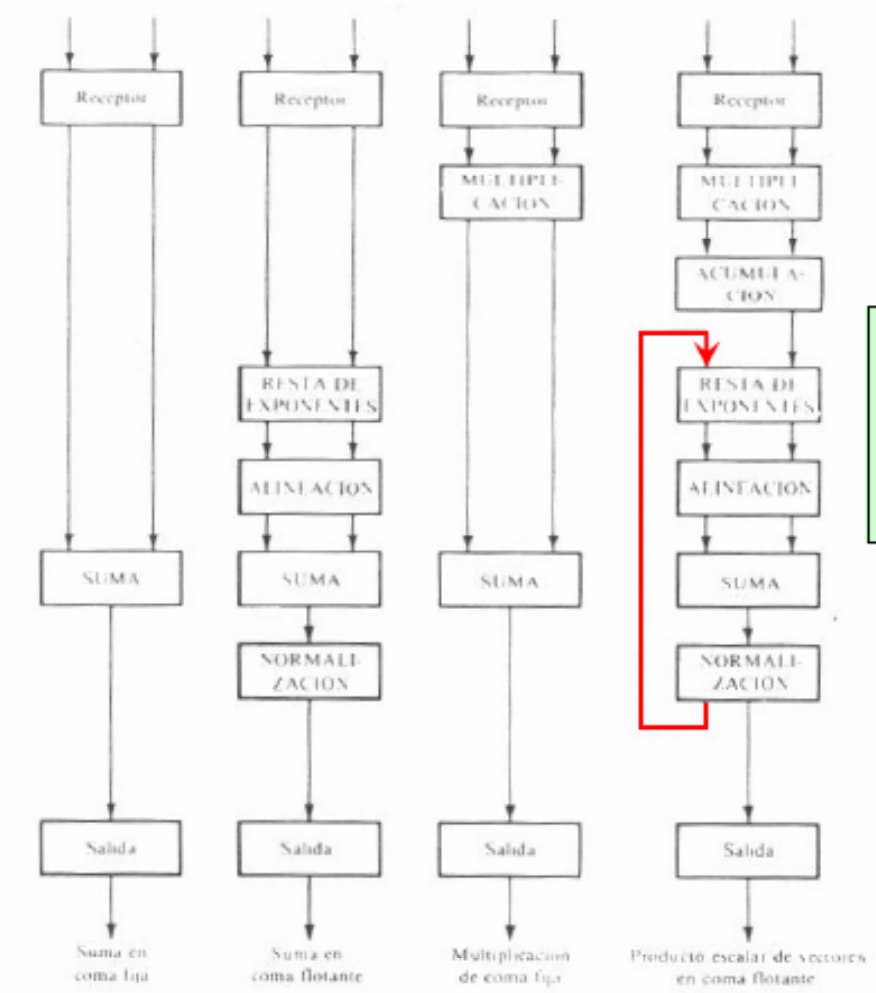
  Teje=1+8+2=11clk

- No lineal => 3 niveles (4SSA) + 1 nivel (SPA)

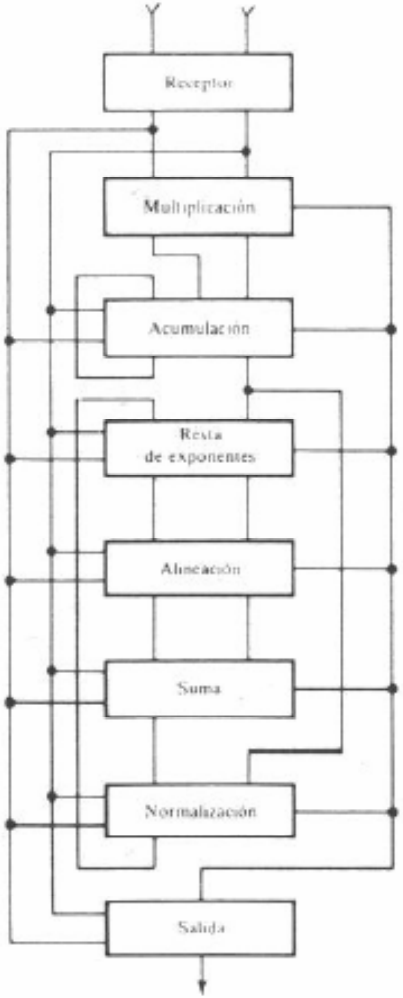
  Teje=1+3x8+2=27clk

La solución no lineal es aprox. 7 veces menos costosa en hw que lineal y solamente 2,5 veces más lenta

Ejemplo cauce aritmético segmentado: TI-ASC



Reutilización de algunas etapas para implementar un producto escalar

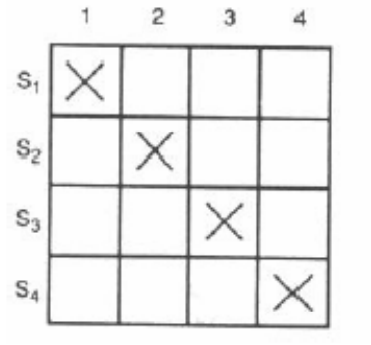


Diseño de los diferentes cauces lógicos del procesador

Implementación de los cauces lógicos en un cauce físico

Consecuecias de la no linealidad:

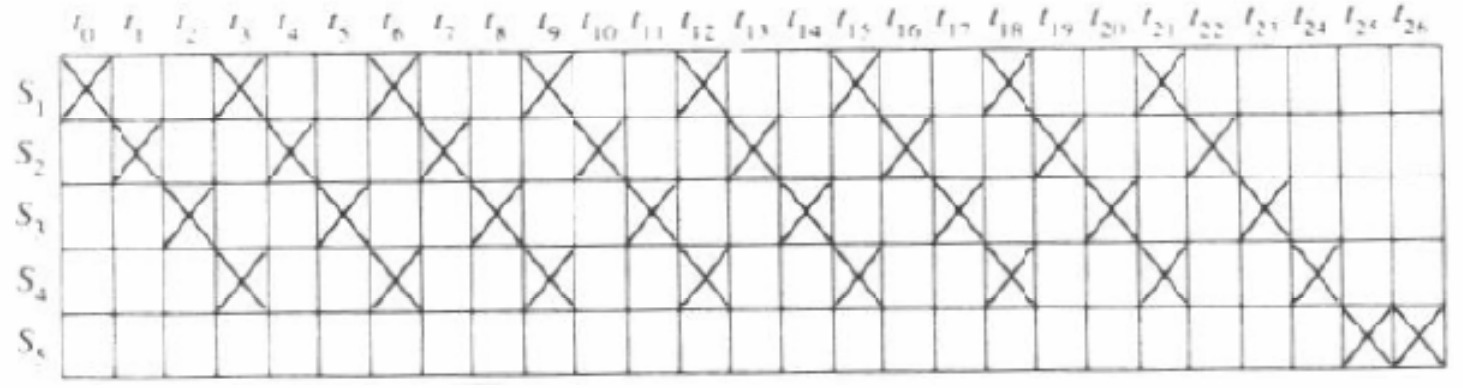
- No podemos introducir datos cada ciclo de reloj; es preciso sincronizar la entrada de datos con el flujo interno para evitar colisiones



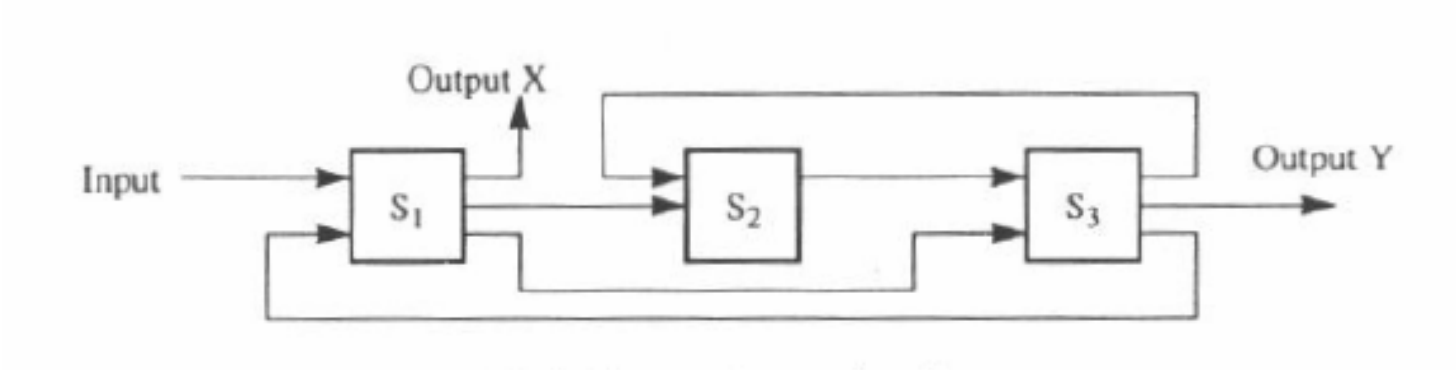
Cauce Lineal (Unifuncional)

La Tabla de Reservas describe el uso que una operación, instrucción,... hace de las distintas etapas del cauce en su paso por el mismo

Permite determinar el tiempo de latencia de la operación, instrucción,...



Cauce No-Lineal (Unifuncional)



Cauce Multifuncional

	1	2	3	4	5	6	7	8
S_1	X					X		X
S_2		X		X				
S_3			X		X		X	

	1	2	3	4	5	6
S_1	Y				Y	
S_2			Y			
S_3		Y		Y		Y

Tablas de Reservas para las dos formas distintas (X, Y) de poder utilizar el cauce

Optimización cauce unifuncional

Definiciones

- **Tabla de reservas**: representación espacio-tiempo de la ocupación de etapas
- **Latencia**: tiempo entre dos entradas de datos consecutivas
- **Latencias prohibidas**: provocan colisiones (coincide con las distancias entre dos X de una misma fase de segmentación)
- **Vector de colisión**: vector binario que representa las latencias prohibidas

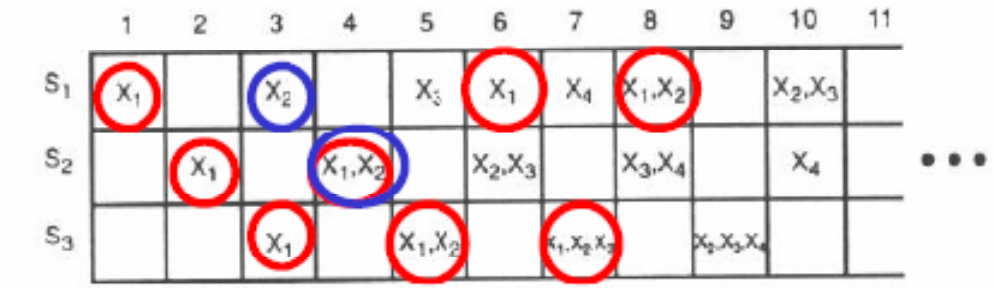
$$VC = (C_n, C_{n-1}, \ldots, C_1) \qquad C_i = \begin{cases} 1 & \text{si } i \in CLP \\ 0 & \text{si } i \notin CLP \end{cases}$$

- **Vector de estado**: VC para cada estado

$$VE(0) = VC$$

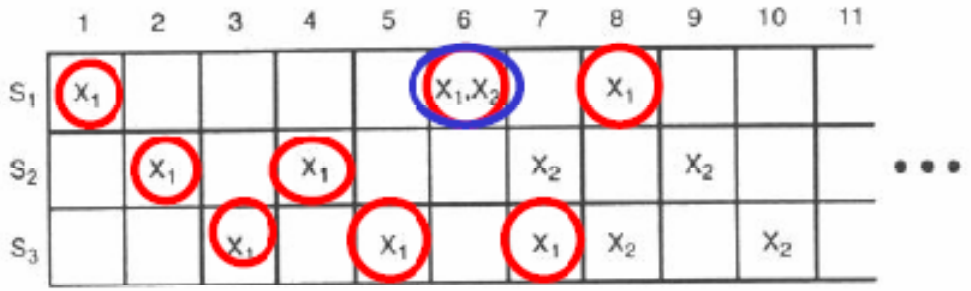
$$VE(t+1) = \begin{cases} \text{Despl. dcha. } (VE(t)) & \text{si no hay entrada de datos en t} \\ \text{Despl. dcha. } (VE(t)) + VC & \text{si hay entrada de datos en t} \end{cases}$$

- **Diagrama de estados**: evolución del sistema según la introducción de datos



Latencia Prohibida 2:

Se produce colisión si se introduce X_1 y 2 ciclos después X_2.



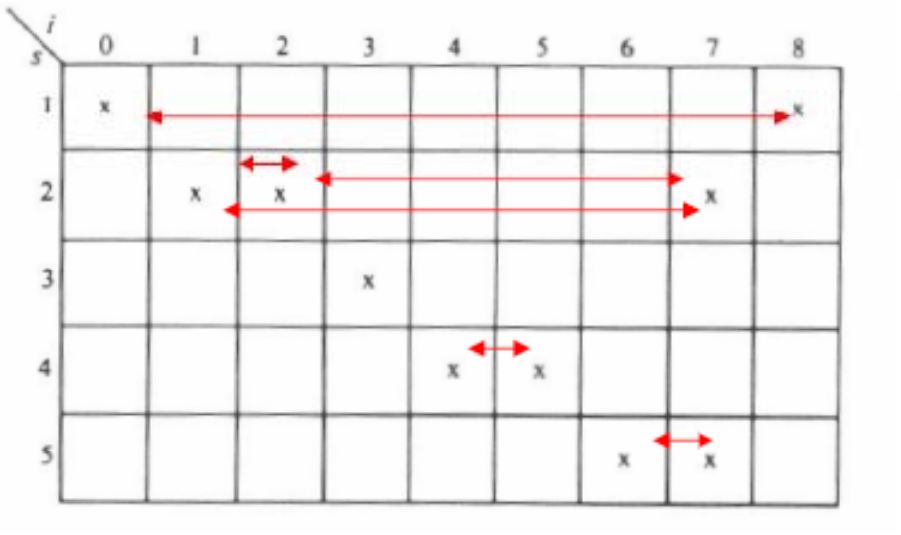
Latencia Prohibida 5:

Se produce colisión si se introduce X_1 y 5 ciclos después X_2.

X_i: instrucción i-ésima del tipo X

En un cauce no se pueden introducir operaciones (instrucciones,...) con retardos iguales a las **latencias prohibidas** del cauce: **se producirían colisiones en el cauce**

- Análisis cauce segmentado: permite obtener una productividad máxima



F={8,1,5,6} (Latencias Prohibidas)

C=(10110001) (Vector de Colisiones)

Unos en las componentes 1,5,6,8 (las componentes se cuentan de derecha a izquierda empezando desde 1)

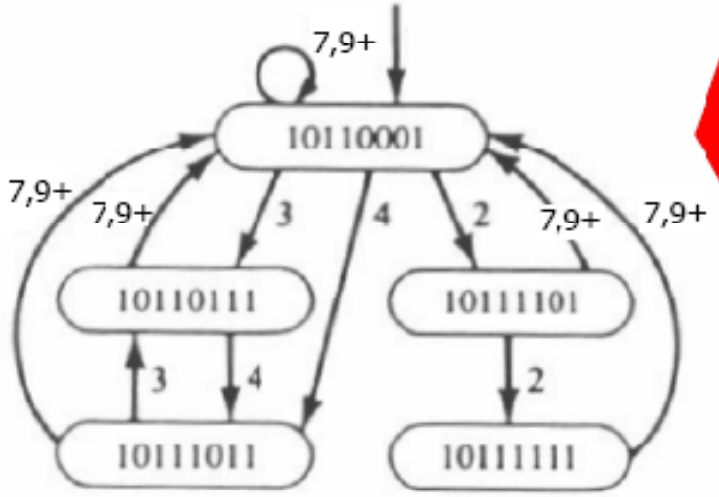
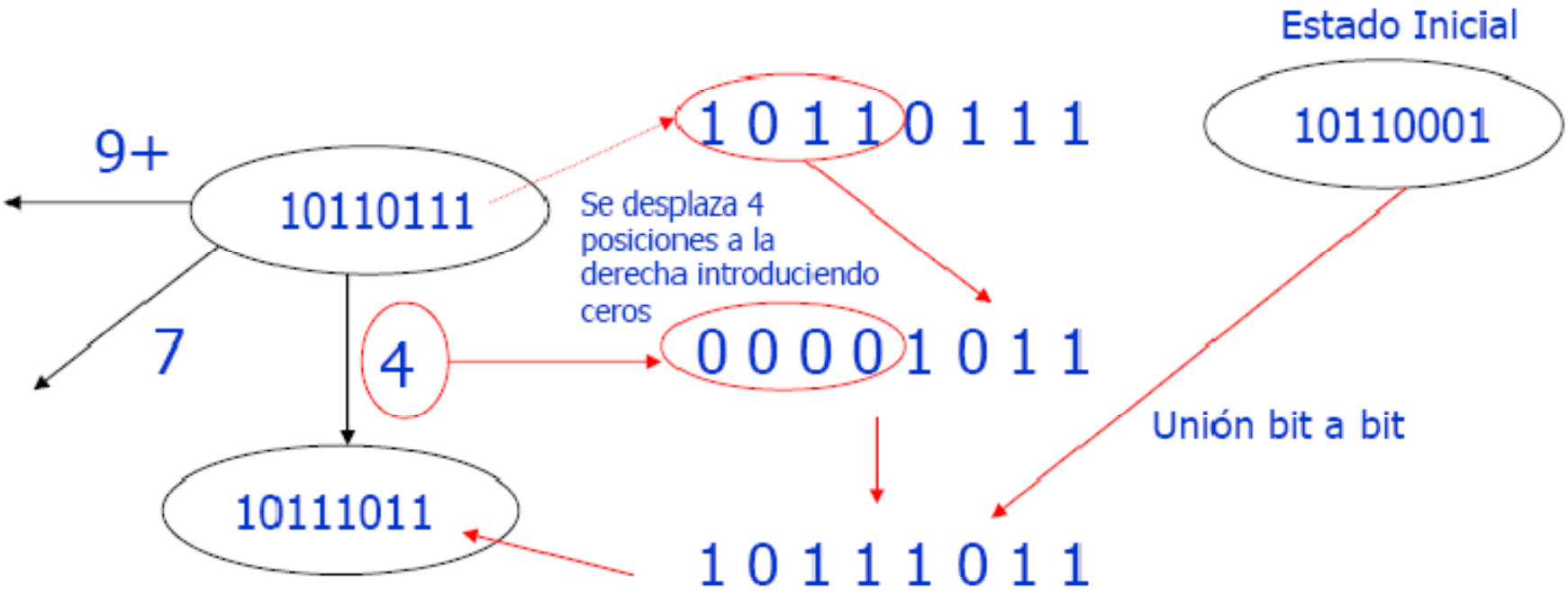


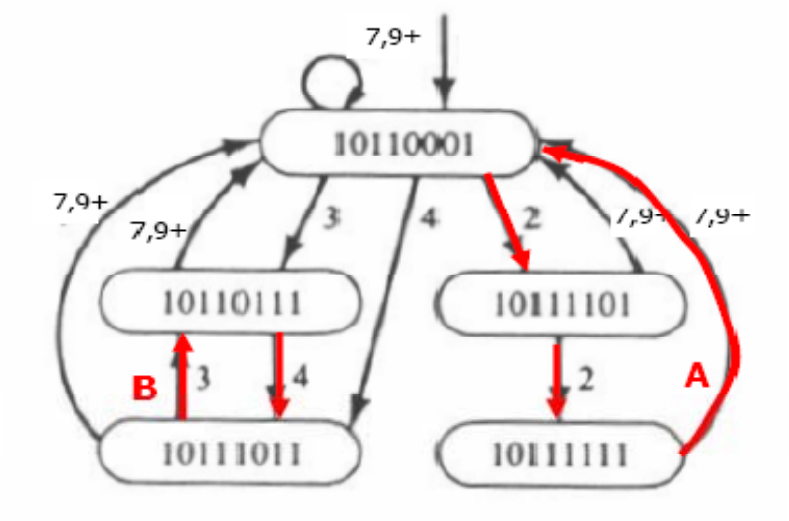
Diagrama de Estados

(1) Construir la Tabla de Reservas
(2) Determinar la Lista de Latencias Prohibidas
(3) Construir el vector de Colisiones
(4) Construir el Diagrama de Estados

- Construcción del diagrama de estados



- Aprovechamiento óptimo de la unidad a través del análisis del diagrama de estados: obtención de la secuencia óptima de introducción de datos (**ciclo**= conjunto de latencias que nos permiten partiendo de un estado volver a él). 2 estrategias:
  - Buscar el ciclo con menor LM => rendimiento óptimo
  - Ciclo avaricioso: se genera tomando en cada transición la primera latencia libre=> simplifica el diseño de la unidad de control



Latencia Media de un Ciclo (LM)

$$LM = \frac{Suma\_de\_Latencias\_del\_Ciclo}{Numero\_de\_Estados\_del\_Ciclo}$$

LM(Ciclo A) = 11/3 = 3.67

LM(Ciclo B) = 7/2 = 3.5

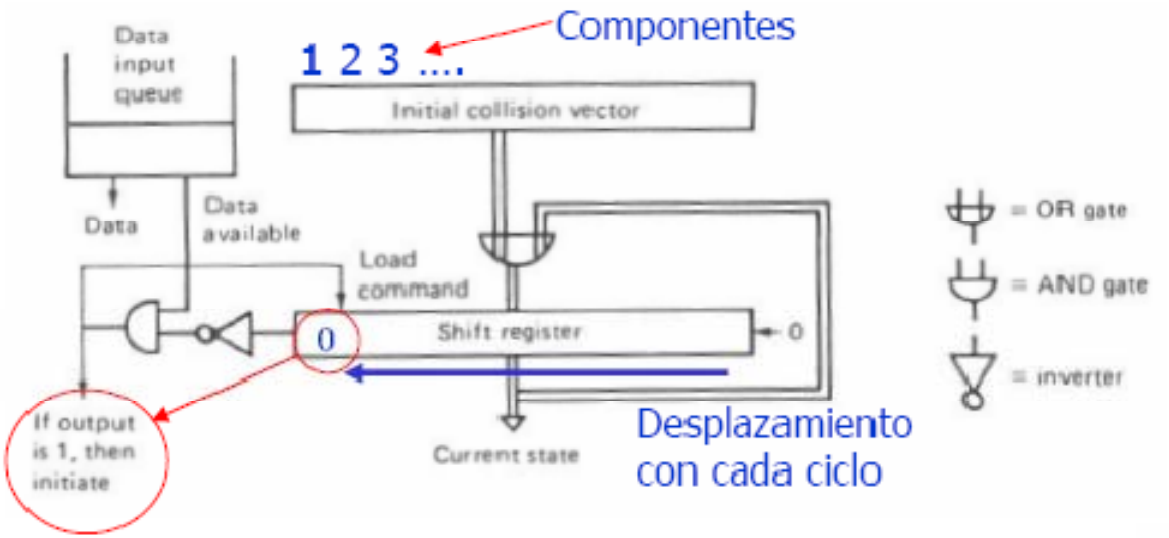
El ciclo para el que se obtiene la menor latencia media (MLM) es el que permite conseguir el mejor rendimiento del cauce:

Ciclos avariciosos: Se obtienen a partir de cada estado, seleccionando la opción correspondiente al mínimo tiempo de espera. Los ciclos A y B de la figura son ciclos avariciosos

$$W_{max} = \frac{1}{MLM \times t}$$

t = periodo del cauce

- Implementación unidad de control



Para Ciclo Avaricioso

Para Ciclo de Mínima Latencia Mínima

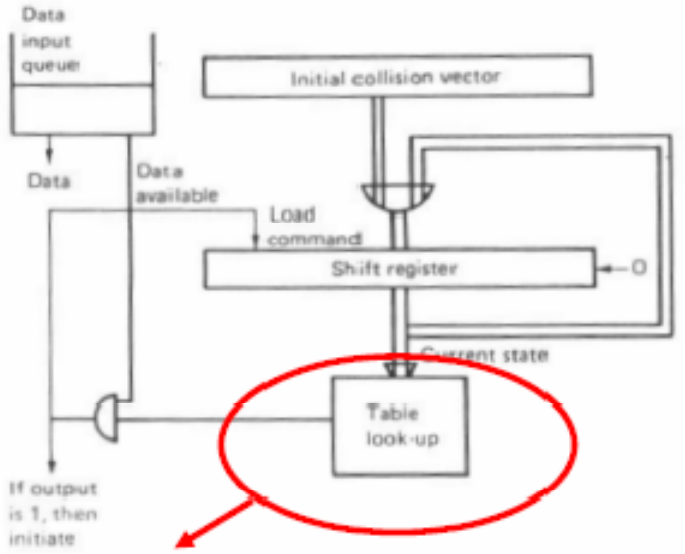


Tabla definida según el ciclo de MLM del diagrama

Optimización cauce multifuncional

s \ ℓ	0	1	2	3	4
1	A	B		A	B
2		A		B	
3	B		AB		A

F_{AB} : Latencias prohibidas que se generan al introducir una **operación B** en el cauce **para las operaciones A que puedan venir después**

FAA ={2,3} **FAB={1,2,4}**

FBA ={2,4} **FBB={2,3}**

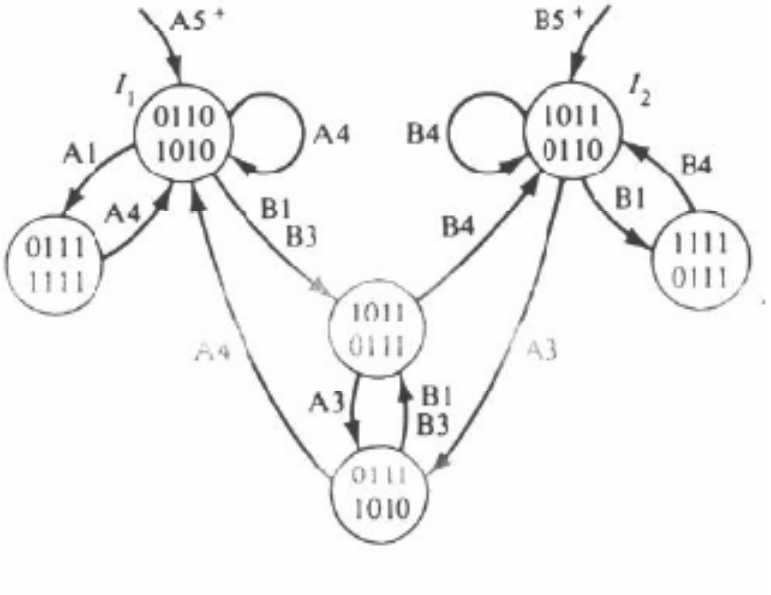
Vectores de Colisiones Cruzadas:

V_{AA}=(0110) V_{AB}=(1011)

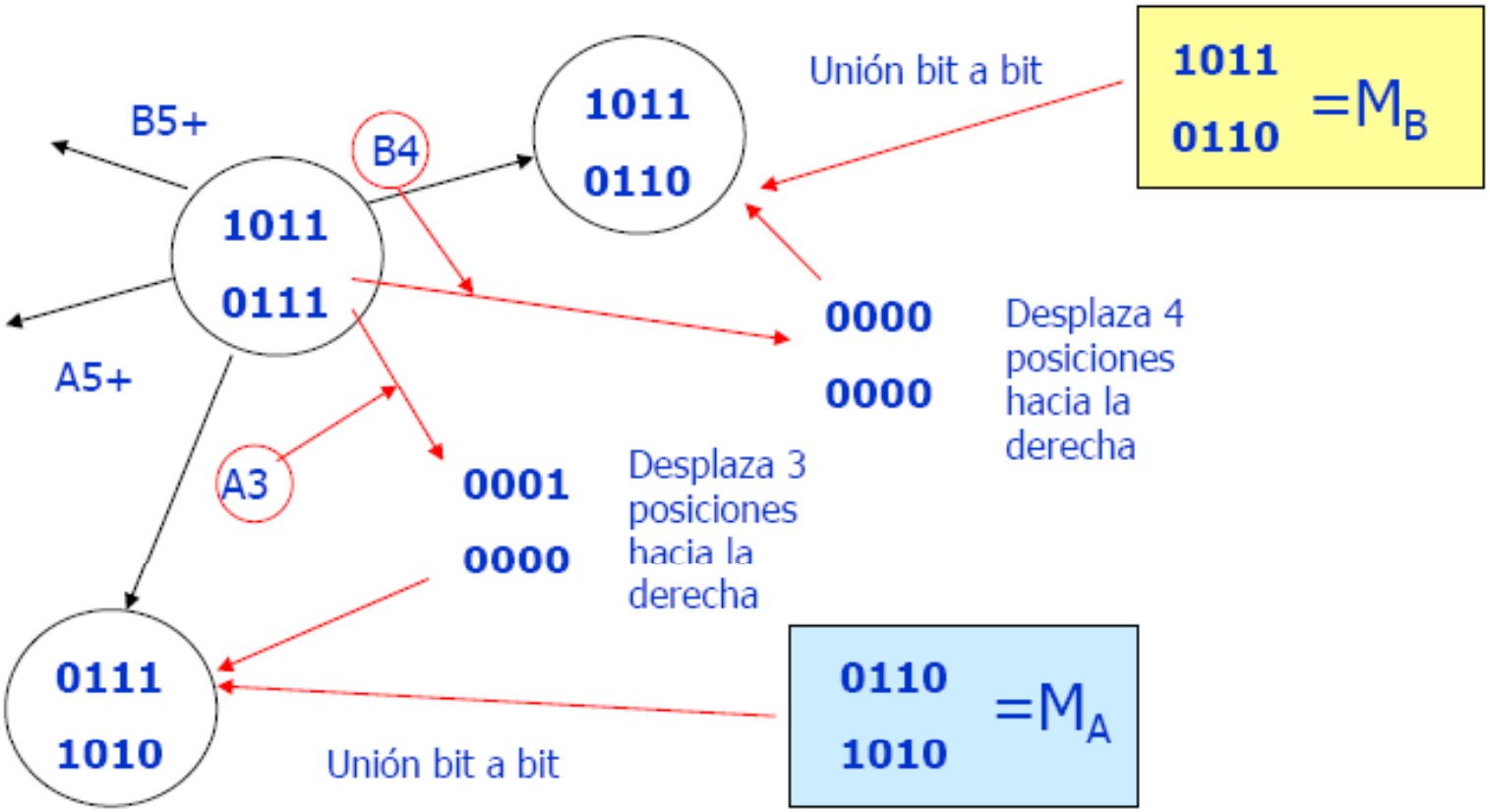
V_{BA}=(1010) V_{BB}=(0110)

Matrices de Colisión:

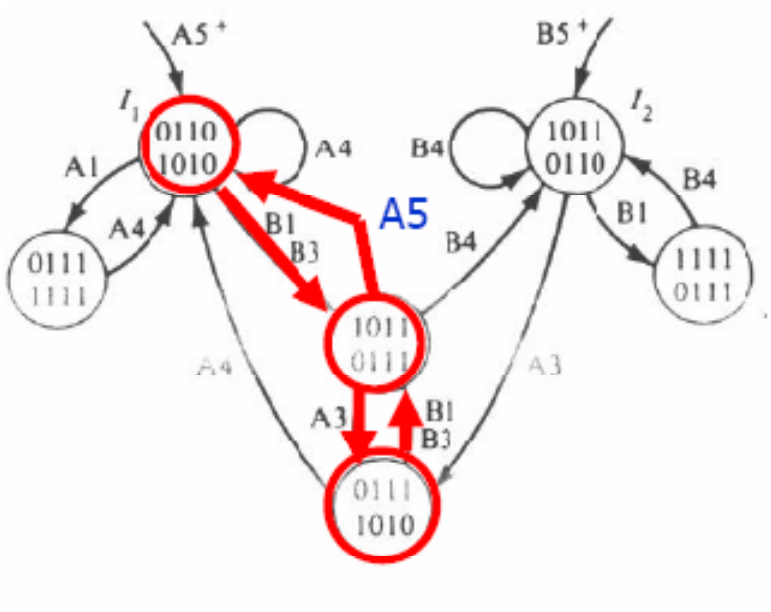
$$M_A = \begin{bmatrix} 0110 & AA \\ 1010 & BA \end{bmatrix} \qquad M_B = \begin{bmatrix} 1011 & AB \\ 0110 & BB \end{bmatrix}$$



- Construcción diagrama de estados



- Construcción diagrama de estados

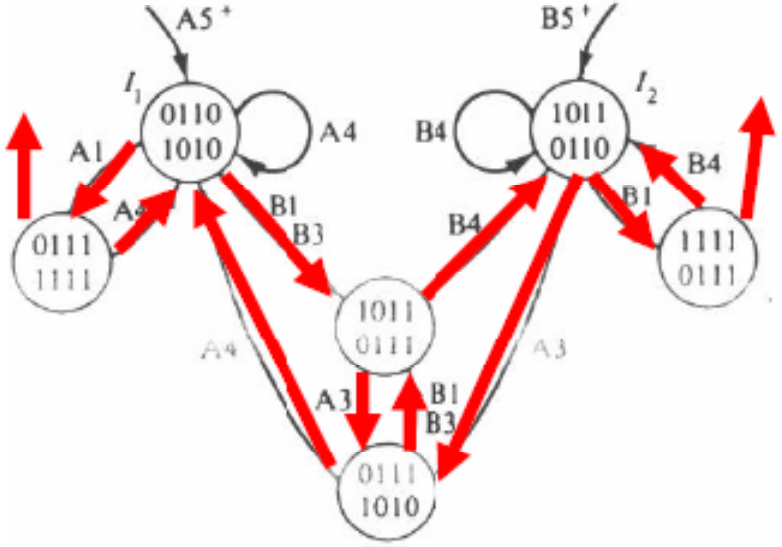


Si se sabe la secuencia de Instrucciones se puede establecer el diagrama de estados (es un subgrafo del obtenido).

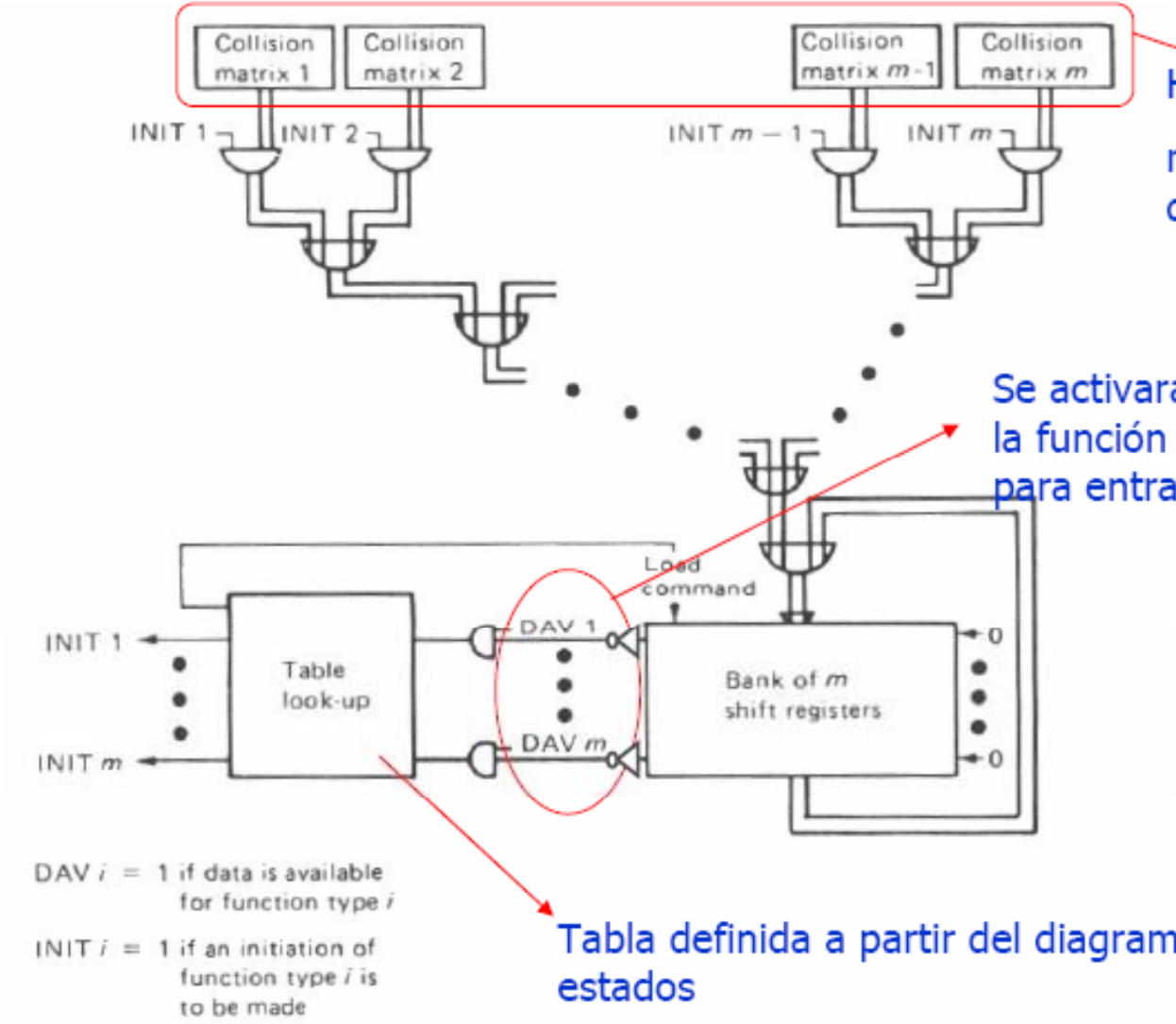
Ejemplo: **ABABAB....**

MLM=(3+1)/2=2

Diagrama de estados a partir de ciclos avariciosos (se sale de cada estado esperando la mínima latencia)



- Implementación unidad de control



Hay m funciones distintas:

m matrices de colisión que definen m estados iniciales

Se activará el que corresponda a la función que está preparada para entrar DAVi=1

Tabla definida a partir del diagrama de estados

Optimizaciones

- Optimización I: Duplicación de etapas

  El nº máximo de X en una etapa coincide con la MLMteórica

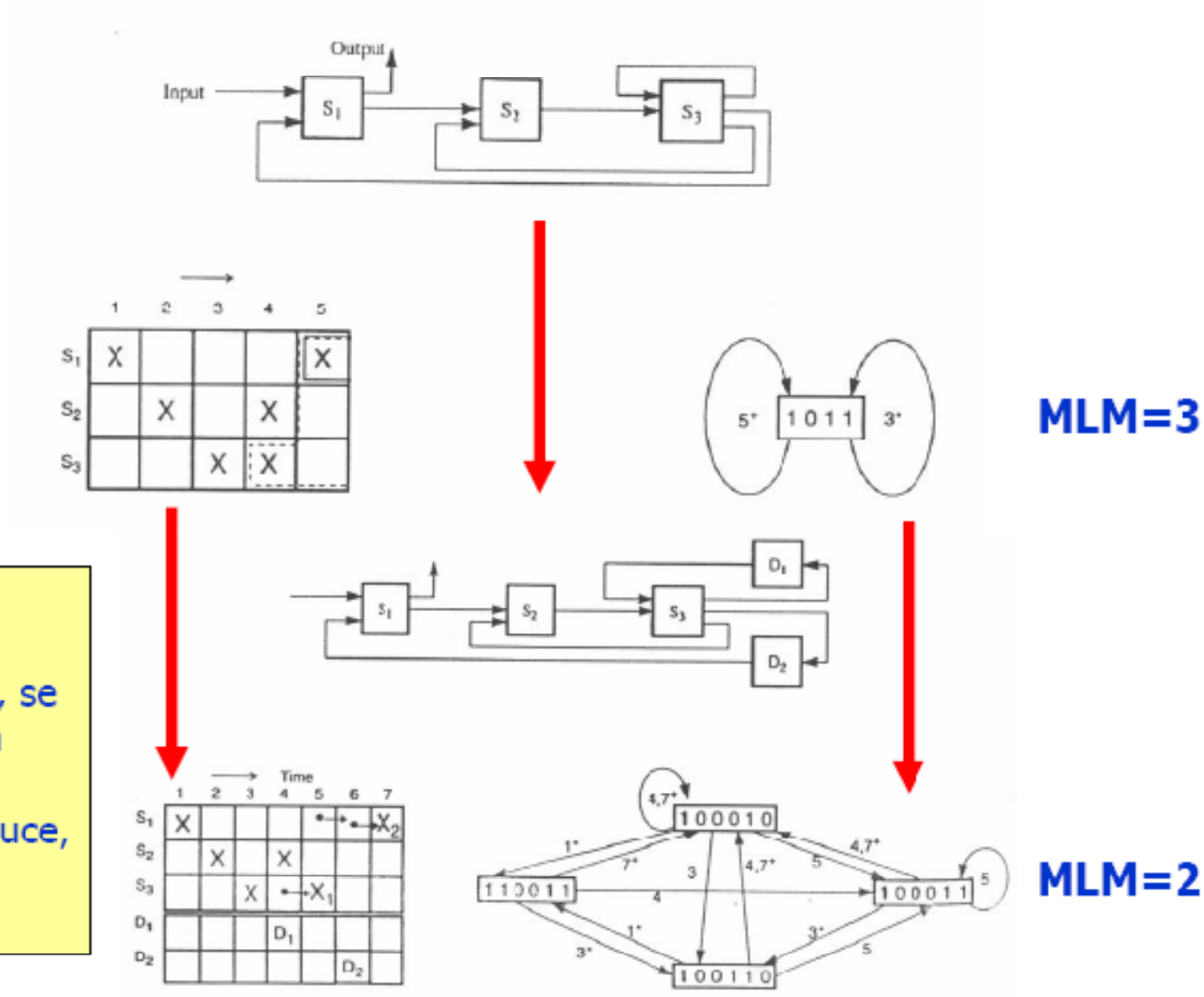
  Si $MLM_{teórica}=MLM_{real}$ podemos disminiur la MLMt eliminando X en la tabla de reserva => duplicación de etapas

- Optimización II: se puede demostrar que si en el CLP existen latencias que son múltiplo de la MLMteórica => MLMreal>MLMteórica

  Para romper esta multiplicidad:

  a) Insertar etapas sin cálculo (almacenamiento temporal) entre etapas
  b) Disminuir la latencia que es múltiplo

- Ejemplo 1



Introduciendo retardos convenientemente, se consigue reducir la MLM (mejorar el rendimiento del cauce, aunque la latencia aumenta)

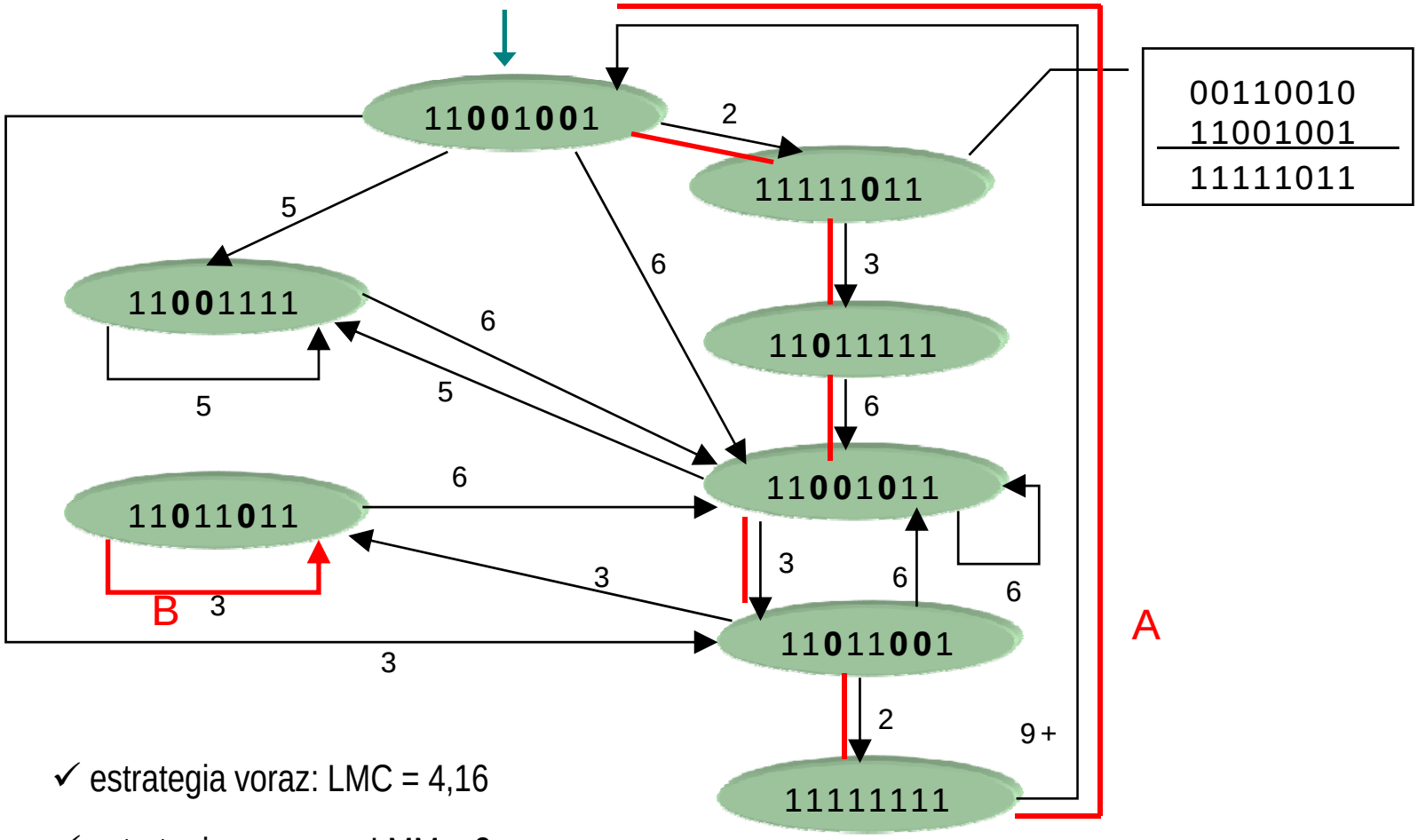
- Ejemplo 2:

S_0	X							X	X			
S_1		X	X									
S_2			X	X								
S_3					X	X						
S_4							X				X	
S_5									X	X		
S_6										X	X	
S_7								X				X

CLP = {1, 4, 7, 8}

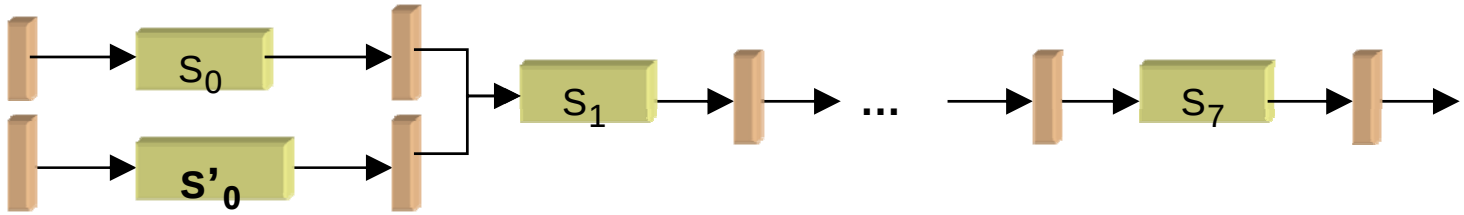
VC = (11001001)

Diagrama de estados

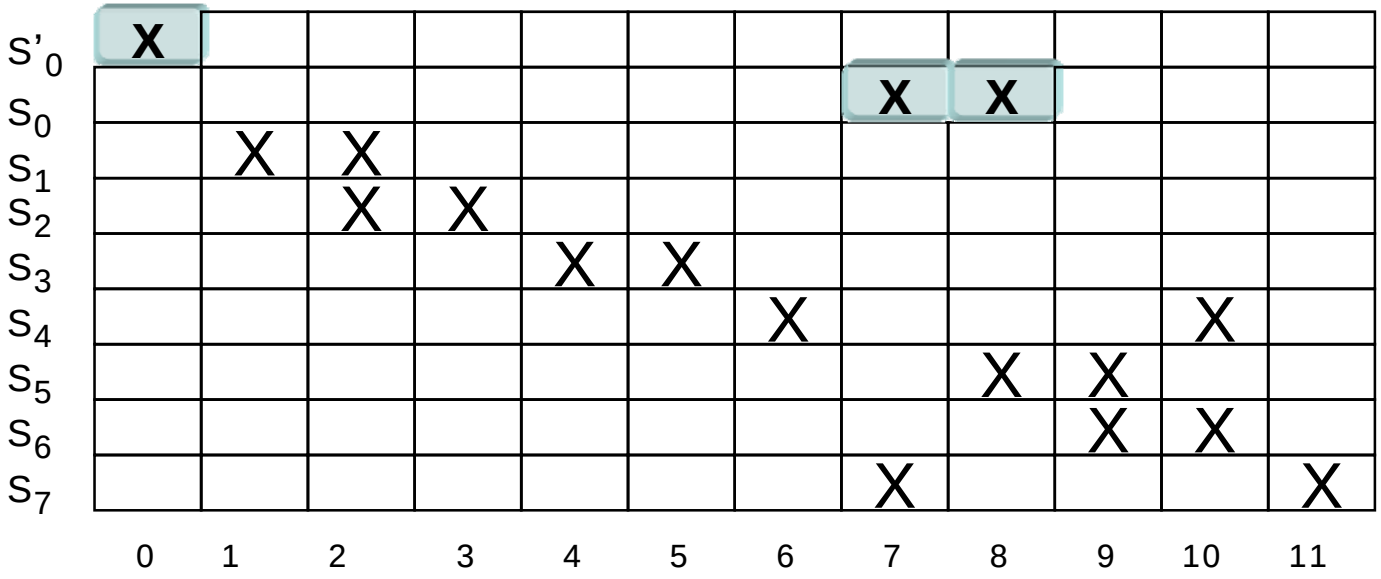


- ✓ estrategia voraz: LMC = 4,16
- ✓ estrategia no voraz: LMM = 3

- Optimización:
  1. Estrategia óptima: evolucionar hacia ciclo con MLM
  2. Duplicación de etapas: $MLM_t = MLM_r$



✓ La LMM_t coincide con el nº máximo de reservas para cualquier etapa



CLP = {1, 4}

VC = (1001)

MLM_t=2;

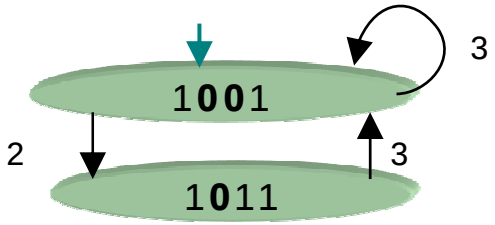
2.Duplicación de etapas

Ciclos:

(2,3); LMC=5/2=2,5

(3) ; LMC=3

(3,2); LMC=2,5

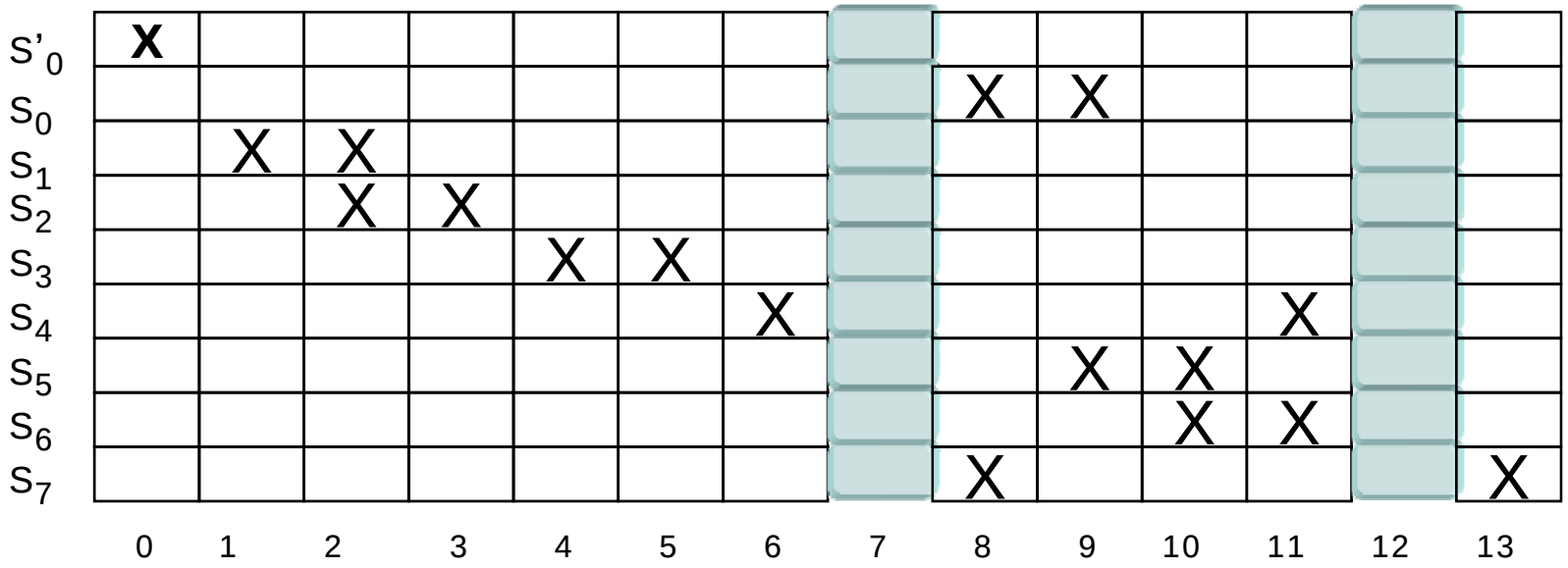


$LMM_r = 2,5$

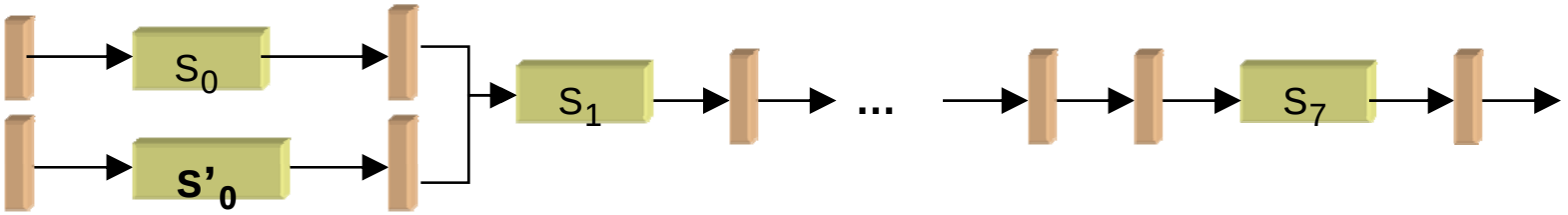
3.Inserción de etapas sin cálculo

✓ Si en el CLP no existe latencia múltiplo de LMM_t , se puede obtener ciclo simple de LMC = LMM_t

CLP = {1, 4} => CLP = {1, 5} , MLMt =2



3. Inserción de etapas sin cálculo



Ciclos:

(2,4); MLM=3

(4); MLM=4

(2); MLM=2

(4,2); MLM=2

(3); MLM=3

(4,3); MLM=

MLMr=2