

# Arquitectura de Computadoras

CURSO 2022

Turno:

Clase 5

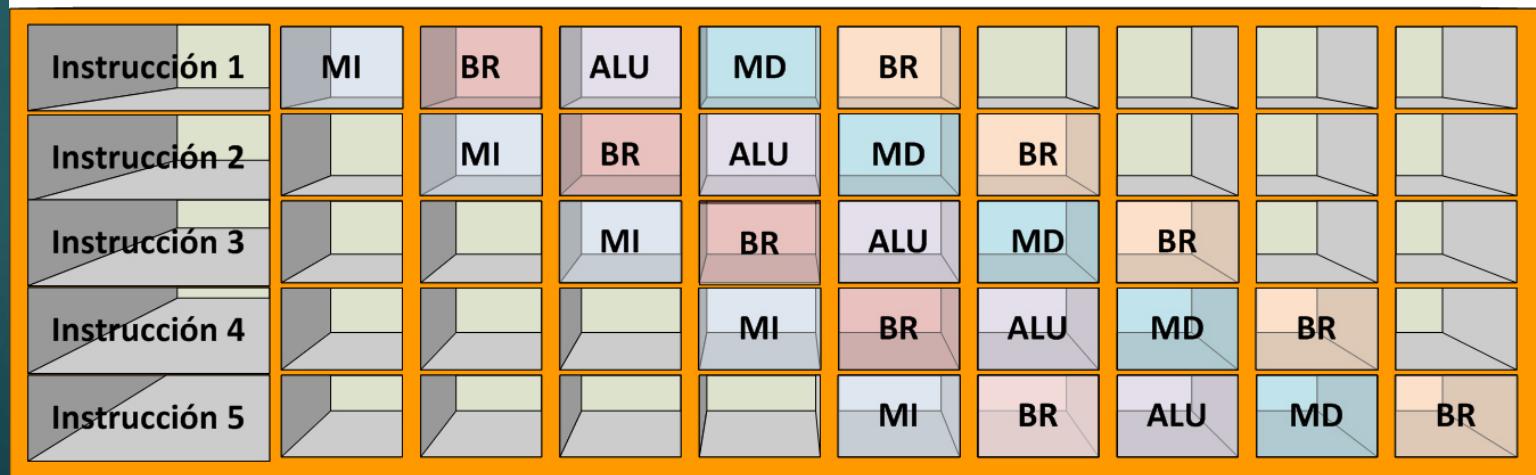
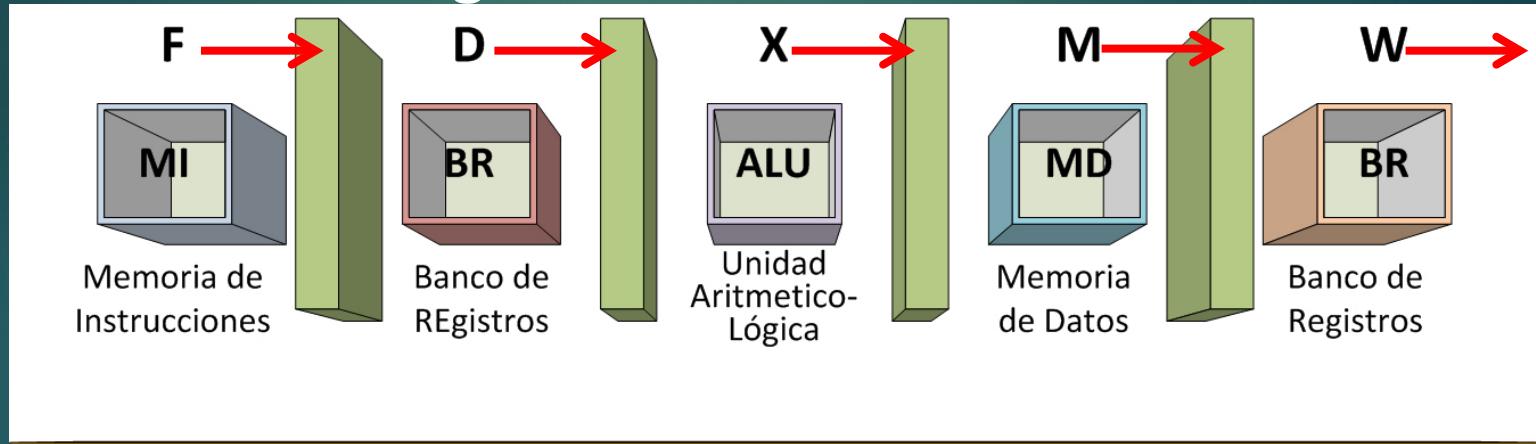
# Resumen clase 5

## Correcciones en procesadores segmentados

- Conflictos en la segmentación del cauce del nanoMips:  
estructurales, por dependencia de datos, por instrucciones de salto
- Soluciones a conflictos por dependencia de datos
  - Por hardware: ciclos de parada, adelantamiento
  - Por software: reordenamiento
- Soluciones a conflictos por instrucciones de salto
  - Por hardware: predictivas estáticas y dinámicas
  - Por software
- Segementación del 80486
- Ejecución de la segmentación en un nanoMIPS multifuncional

# Segmentación y solapamiento del cauce en el nanoMIPS

De acuerdo a lo presentado en la clase 4, la ejecución segmentada en 5 etapas en el nanoMIPS, de un secuencia de 5 instrucciones es la siguiente:



# Problemas de la Segmentación

La imagen anterior representa una situación ideal. En la clase anterior se identificaron los tipos de conflictos que aparecen en un procesador segmentado real. Estos riesgos pueden producir atascos en el flujo de las instrucciones.

- 1.- Estructurales: provocados por conflictos en el uso de los “recursos”. Los recursos típicamente son: memoria, ALU, registros.
- 2.- Por dependencia de datos: son conflictos originados entre 2 o más instrucciones que comparten un mismo dato. Por ejemplo, una instrucción produce un resultado que lo necesita otra, ambas instrucciones dentro del cauce de datos.
- 3.- Por dependencia de control: Ejecución de instrucciones que alteran la secuencia normal de ejecución, es decir, instrucciones de salto (condicional e incondicional).

# Soluciones a los problemas de la Segmentación

## Solución elemental

- La solución mas elemental para superar los atascos, producto de los 3 tipos de riesgos que se pueden presentar en un procesador segmentado, es agregar puntos de parada en el cauce hasta que el conflicto desaparezca.
- El gran problema que tiene este tipo de solución es que el rendimiento cae fuertemente por la cantidad de tiempos muertos que se insertan, inclusive puede llegar a ser hasta inútil aplicar la segmentación.
- Hay varias técnicas que permiten reducir y, en algunos casos, eliminar los efectos de los riesgos.
- A continuación se presentan algunas de las soluciones usadas en las máquinas con segmentación del cauce.

# Soluciones a los problemas de la Segmentación

## Solución a los conflictos estructurales por uso de los recursos

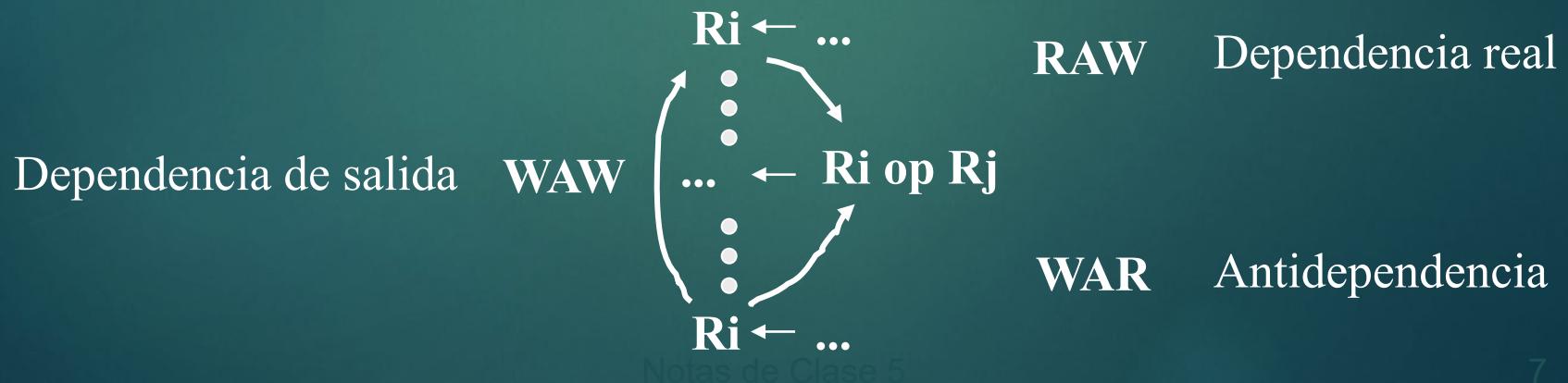
La solución a estos conflictos consiste básicamente en “agregar más hardware”. Algunas estrategias usadas son:

- 1.- Duplicación de recursos de hardware  
Ej: agregar sumadores o restadores además de la ALU.
- 2.- Separación del recurso en conflicto  
Ej: separar memorias de instrucciones y datos.
- 3.- Subdividir el acceso al recurso  
Ej: en el acceso a los registros (banco de registros) la escritura de un registro se puede hacer en el primer semiciclo de reloj, y la lectura en el segundo semiciclo, con lo que en un ciclo se pueden hacer las 2 operaciones, siempre y cuando la velocidad del recurso lo soporte.

# Soluciones a los problemas de la Segmentación

## Solución a los conflictos por dependencia de datos

- Recordemos que existen 3 tipos de dependencias de datos en un proceso de segmentación:
  - RAW: lectura después de escritura
  - WAR: escritura después de lectura
  - WAW: escritura después de escritura
- Esas situaciones se muestran en el siguiente esquema:



# Soluciones a los problemas de la Segmentación

## Solución a conflictos por dependencia de datos

Las soluciones se pueden implementar de 2 maneras:

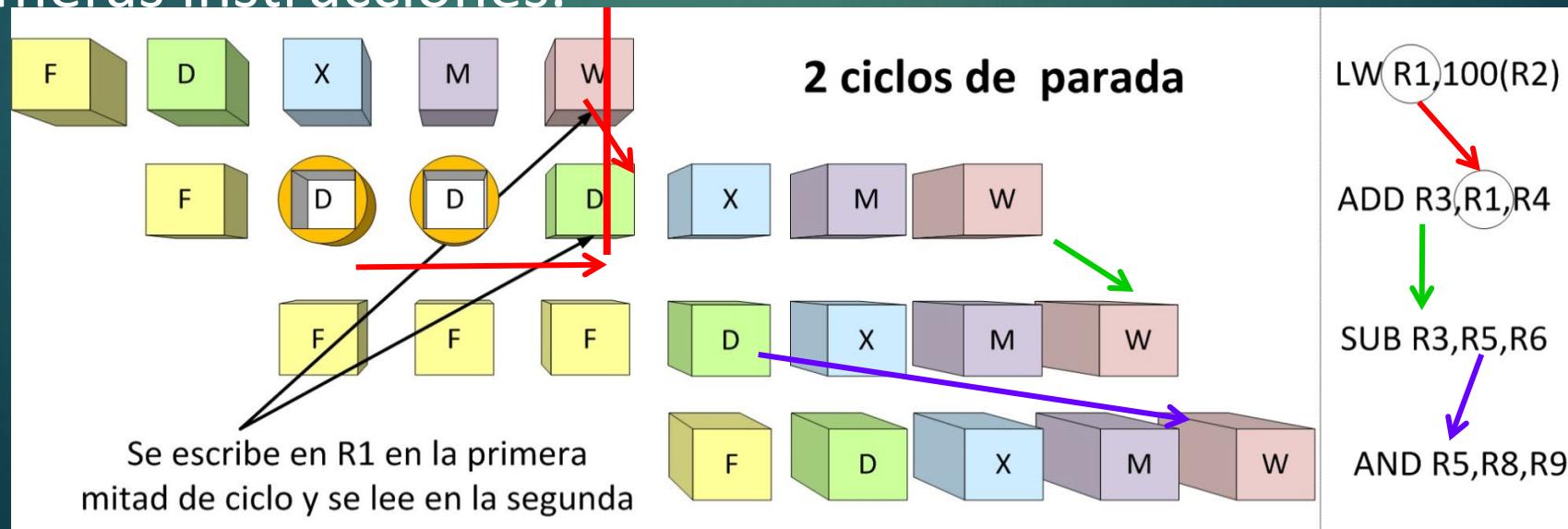
- 1.- por Hardware
  - Retardos en las etapas del cauce, es la solución ya vista.
  - Adelantamiento de operandos (forwarding) entre etapas del cauce.
- 2.- por Software
  - Introducción de NOP (equivalente a insertar retardos), equivalente a introducir puntos de parada.
  - Reordenación de código.

A continuación se van a analizar las diferentes técnicas.

# Soluciones a los problemas de la Segmentación

## Solución a conflictos por dependencia de datos con ciclos de parada

La solución más sencilla para eliminar conflictos por dependencia de datos es agregando retardos o ciclos de parada en la ruta de datos. En el siguiente ejemplo, hay una dependencia del tipo RAW con el registro R1 entre las 2 primeras instrucciones.



# Soluciones a los problemas de la Segmentación

## Solución a conflictos por dependencia de datos con ciclos de parada

- En el ejemplo anterior, el conflicto surge entre las instrucciones:  
 $LW R1,100(R2)$   
 $ADD R3,R1,R4$
- La instrucción ADD requiere, en la etapa D (decodificación), el valor de R1, que se carga en la instrucción previa LW en la etapa W (escritura en BR). La etapa D de ADD no se puede ejecutar hasta que se ejecute la etapa W de LW.
- Si se considera que R1 se escribe en el primer medio ciclo de la etapa W, y se lee en el segundo semiciclo de la etapa D, entonces agregando 2 ciclos de parada, como se muestra en la imagen anterior, se resuelve el conflicto.

# Soluciones a los problemas de la Segmentación

## Solución a conflictos por dependencia de datos con ciclos de parada

- Está claro que este tipo de solución permite resolver los conflictos por dependencia de datos.
- El gran problema de esta solución es que los ciclos de parada agregados reducen fuertemente la performance del procesador segmentado.

# Soluciones a los problemas de la Segmentación

Solución a conflictos por dependencia de datos con forwarding o adelantamiento

- Este método consiste en pasar directamente (“adelantar”), desde una unidad funcional a otra, el resultado obtenido en una instrucción a las instrucciones siguientes que lo necesitan como operando.
- Si el dato que necesita la instrucción  $i+1$  ya está calculado por la instrucción  $i$ , se puede llevar (adelantar) a la entrada de la etapa de la instrucción  $i+1$  que lo necesita, sin esperar la etapa final de escritura de la instrucción  $i$ .
- Esta solución puede ser relativamente sencilla de implementar si se identifican todos los adelantamientos y se pueden adelantar los datos a las unidades que lo necesitan.

# Soluciones a los problemas de la Segmentación

Solución a conflictos por dependencia de datos con forwarding o adelantamiento

Consideremos esta solución para el caso del nanoMIPS.

En la siguiente secuencia de instrucciones hay dependencia del tipo RAW entre la escritura del registro R1 en la instrucción LW y su lectura en la instrucción SW.

LW R1, 100(R2)

SW R1, 0(R10)

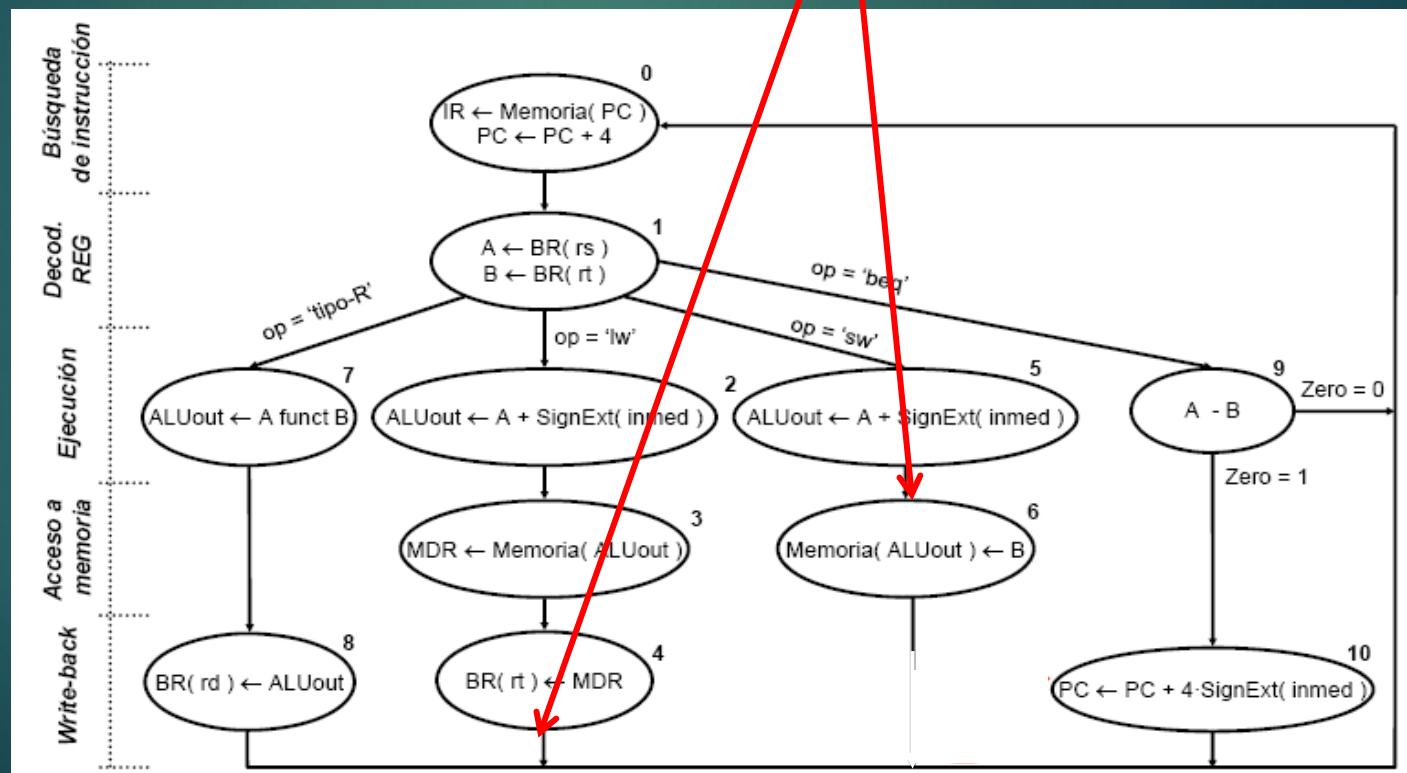
En la imagen siguiente se muestra la relación entre las etapas de la segmentación y los ciclos donde se ejecutan la escritura de R1 (en la etapa 4, fase W, de la instrucción de LOAD) y la lectura de R1 (en la etapa 6, fase M, de la instrucción STORE).

# Soluciones a los problemas de la Segmentación

LW R1, 100(R2)



SW R1, 0(R10)



# Soluciones a los problemas de la Segmentación

Solución a conflictos por dependencia de datos con forwarding o adelantamiento

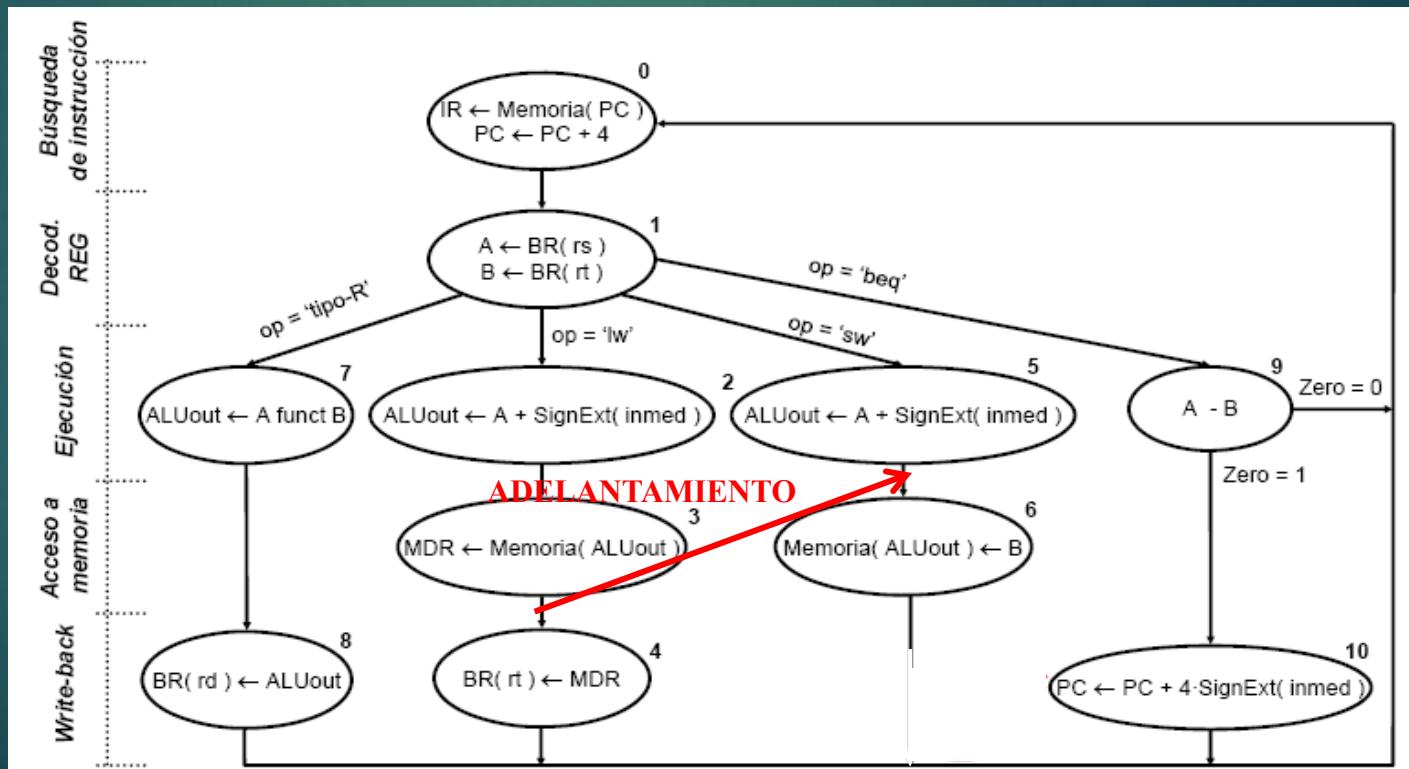
- En la imagen anterior se puede apreciar que no se puede ejecutar el ciclo M de la instrucción SW hasta que se resuelva el ciclo W de la instrucción LW.
- En este caso, el dato que se va a cargar en R1 ya está disponible en la etapa previa (M) de la instrucción LW, aunque aún no se ha cargado en el R1 (etapa W).
- Si hubiera un camino que adelantara el dato desde la salida de la unidad M hasta la entrada de la misma unidad, no habría que esperar el ciclo de W de la instrucción LW, para ejecutar el ciclo M de la instrucción SW.
- Esta situación se muestra en la figura siguiente.

# Soluciones a los problemas de la Segmentación

LW R1, 100(R2)



SW R1, 0(R10)



# Soluciones a los problemas de la Segmentación

Solución a conflictos por dependencia de datos con forwarding o adelantamiento

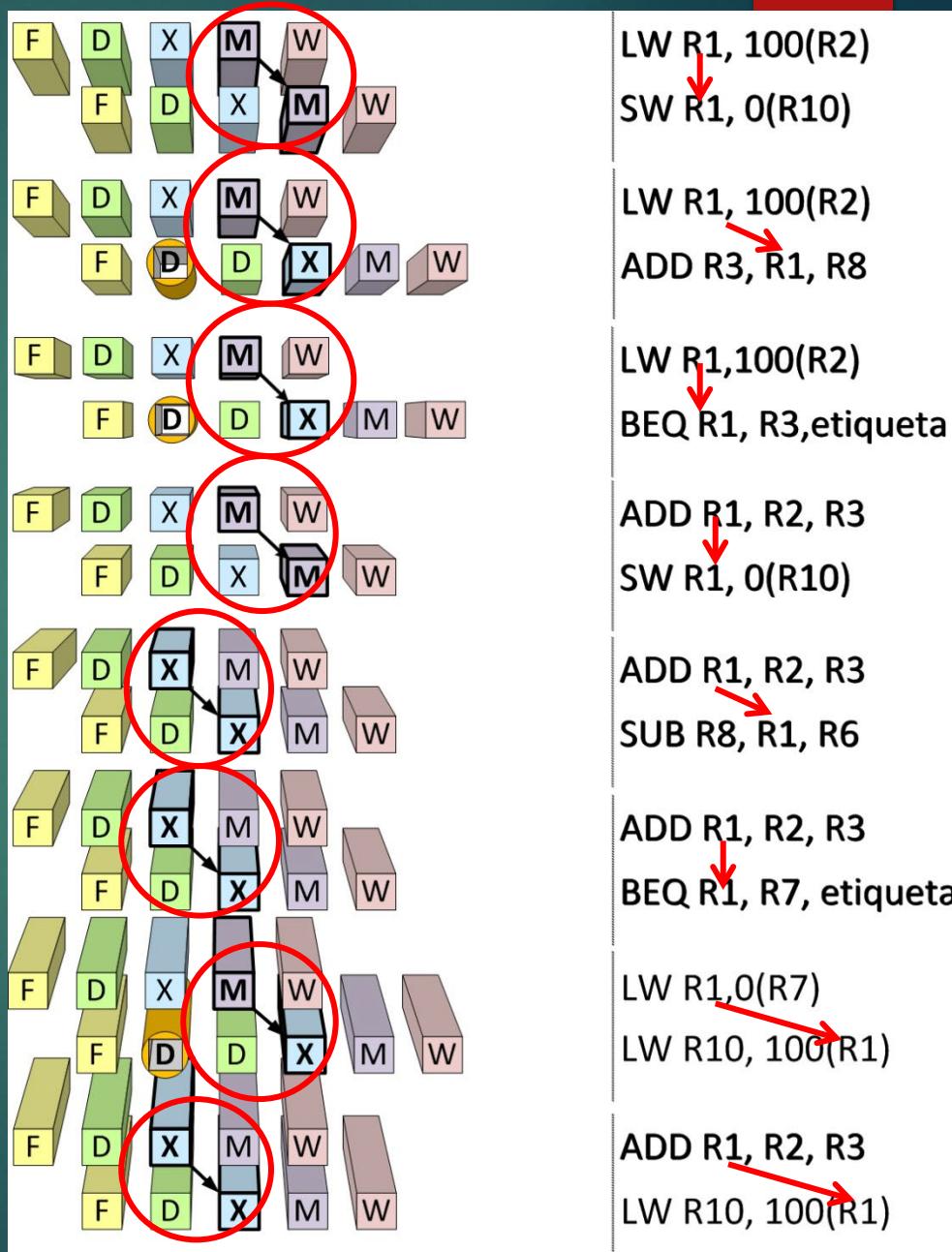
- Se puede hacer el mismo análisis para todas las secuencias de instrucciones, a fin de determinar los posibles casos de adelantamiento y su impacto en la mejora de la segmentación.
- En procesadores sencillos como el nanoMIPS este análisis es relativamente fácil. Si el procesador es más complejo el análisis puede resultar muy complicado.
- En el caso del nanoMIPS se pueden identificar 8 tipos de conflictos por dependencia de datos (del tipo RAW únicamente), y sus posibles soluciones con adelantamiento se muestran en la figura siguiente.

Clase 5 | Documento: Personal

# Soluciones a los problemas de la Segmentación

## CASOS a considerar:

- 1)LW seguido de SW
  - 2)LW seguido de Aritmética-Lógica
  - 3)LW seguido de BEQ
  - 4)Aritmético-Lógica seguida de SW
  - 5)Aritmético-Lógica seguida de  
Aritmético-Lógica.
  - 6)Aritmético-Lógica seguida de  
BEQ
  - 7)LW seguido de LW
  - 8)Aritmético-Lógica seguida de LW



# Soluciones a los problemas de la Segmentación

Solución a conflictos por dependencia de datos con forwarding o adelantamiento

En la figura anterior, se han identificado 8 casos de conflictos por dependencia del tipo RAW:

- 1)LW seguido de SW
- 2)LW seguido de Aritmética-Lógica
- 3)LW seguido de BEQ
- 4)Aritmético-Lógica seguida de SW
- 5)Aritmético-Lógica seguida de Aritmético-Lógica.
- 6)Aritmético-Lógica seguida de BEQ
- 7)LW seguido de LW
- 8)Aritmético-Lógica seguida de LW

# Soluciones a los problemas de la Segmentación

Solución a conflictos por dependencia de datos con forwarding o adelantamiento

Para cada caso se considera:

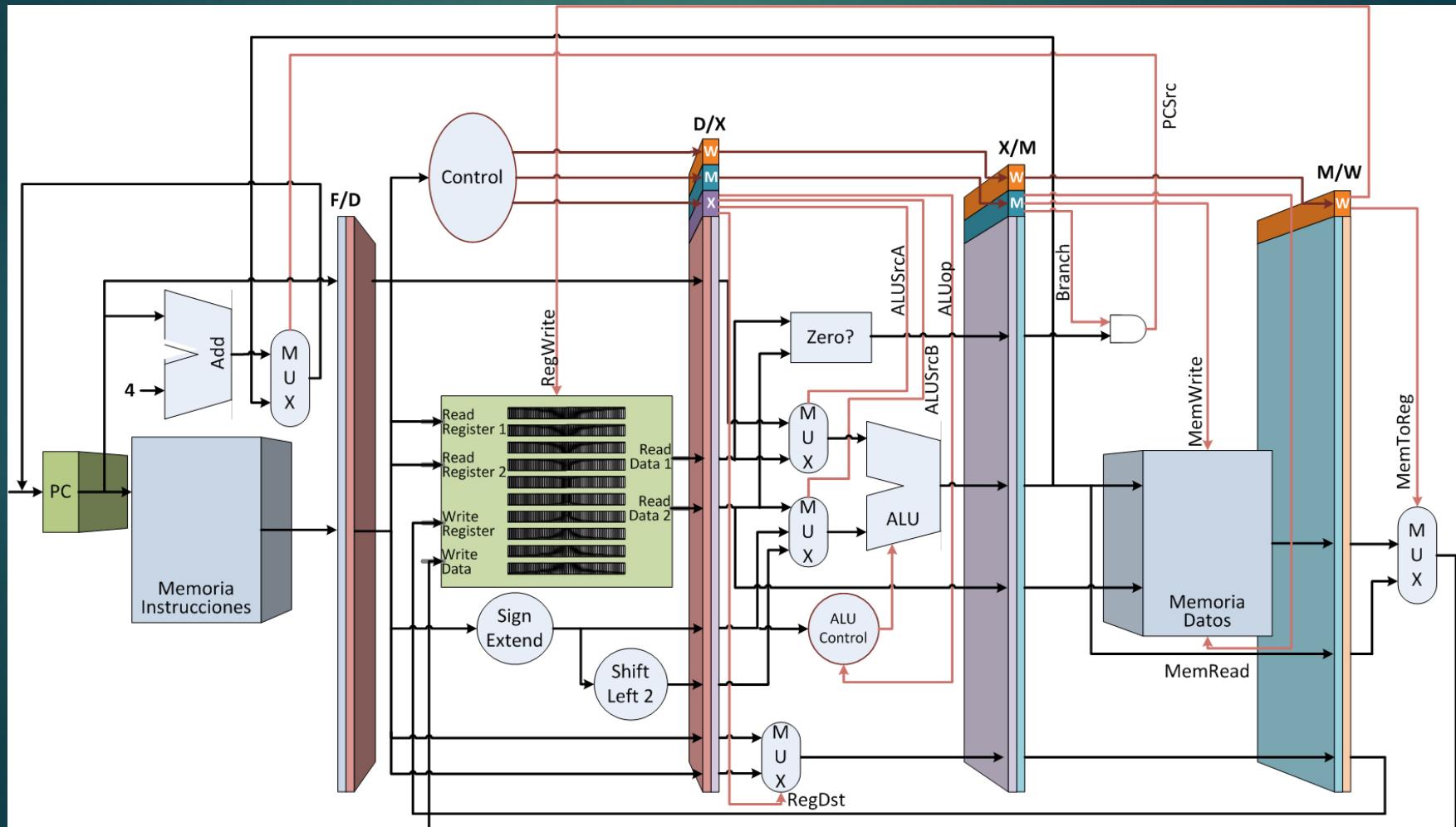
- A la derecha está remarcada la dependencia en la secuencia de instrucciones
- A la izquierda se indica donde se debe implementar el adelantamiento.

Por ejemplo, en el caso 1 de LW seguido de SW, el adelantamiento debería hacerse entre la salida de la M (memoria de datos) y la propia entrada de M.

A continuación se describe la forma de implementar estos adelantamientos, y su impacto en la organización interna de la máquina.

# Soluciones a los problemas de la Segmentación

El modelo del nanoMIPS segmentado era el siguiente.



# Soluciones a los problemas de la Segmentación

Solución a conflictos por dependencia de datos con forwarding o adelantamiento

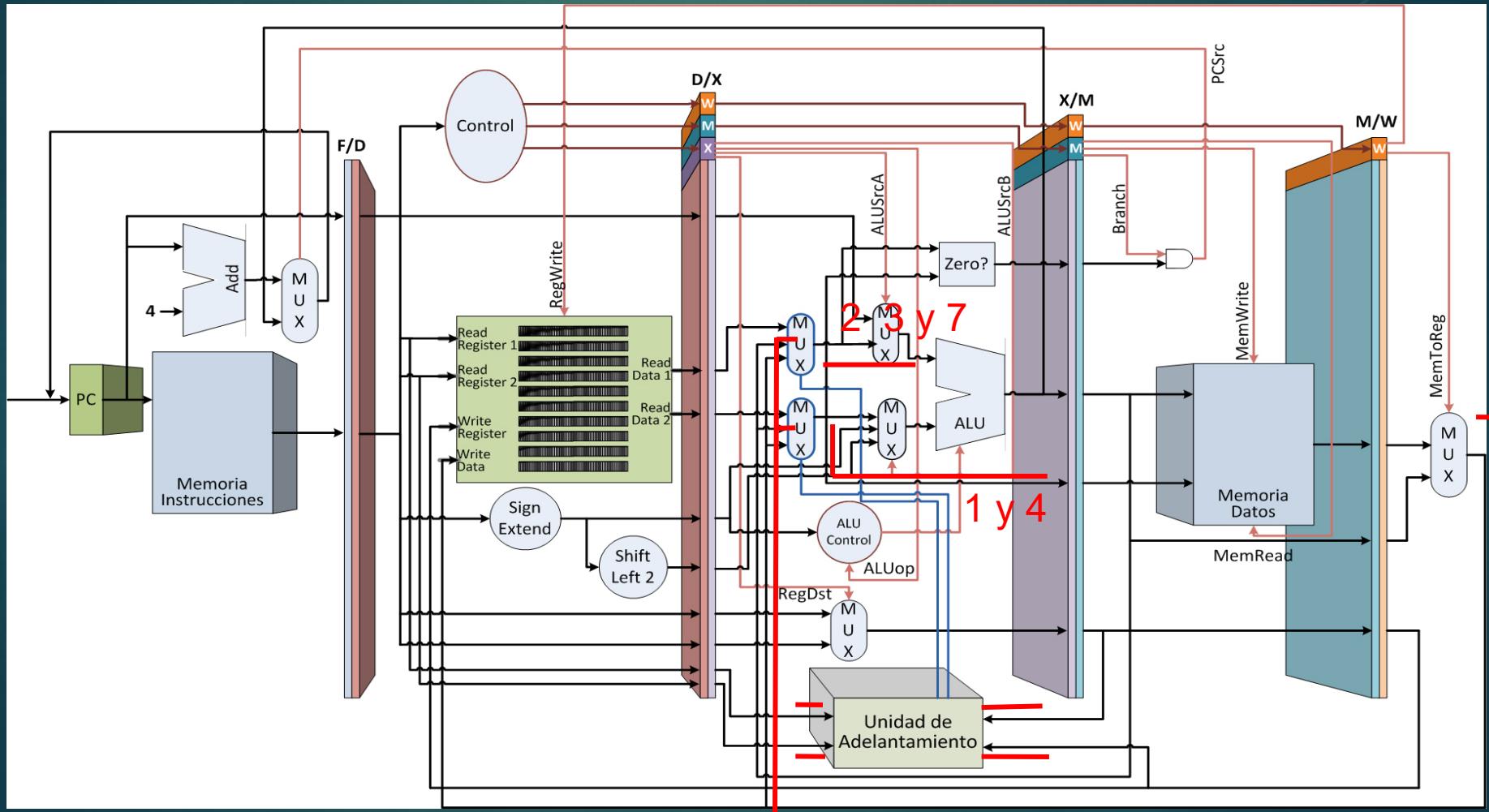
Los 8 casos encontrados, se pueden agrupar en 3 problemas de adelantamiento:

- 1) En los casos 1 y 4, adelantamiento desde la salida de M a la entrada de M.
- 2) En los casos 2, 3 y 7, adelantamiento desde la salida de M a la entrada de X.
- 3) En los casos 5, 6 y 8, adelantamiento desde la salida de X a la entrada de X.

Para solucionar el primer grupo (casos 1 y 4) se requiere prever un camino de datos directo desde la salida de M a la entrada de M, y para el segundo grupo (casos 2, 3 y 7) se requiere prever un camino de datos directo desde la salida de M a la entrada de X. Eso se puede resolver usando multiplexores (MUX) como se muestra a continuación.

# Soluciones a los problemas de la Segmentación

Solución de adelantamiento en casos 1, 2, 3, 4 y 7:



# Soluciones a los problemas de la Segmentación

Solución a conflictos por dependencia de datos con forwarding o adelantamiento

En la imagen anterior, la trayectoria de los datos desde la M hasta el BR (banco de registros) tiene una derivación que termina en multiplexores (MUX), que permiten seleccionar distintas alternativas de fuentes de datos a X (ALU) y a M (memoria de datos).

Cuando una unidad auxiliar, identificada como unidad de adelantamiento detecta en los campos de las instrucciones que se están ejecutando, las dependencias 1,2,3,4 y 7, selecciona en los MUX asociados a ella, las entradas necesarias para adelantar los datos requeridos (por X o M).

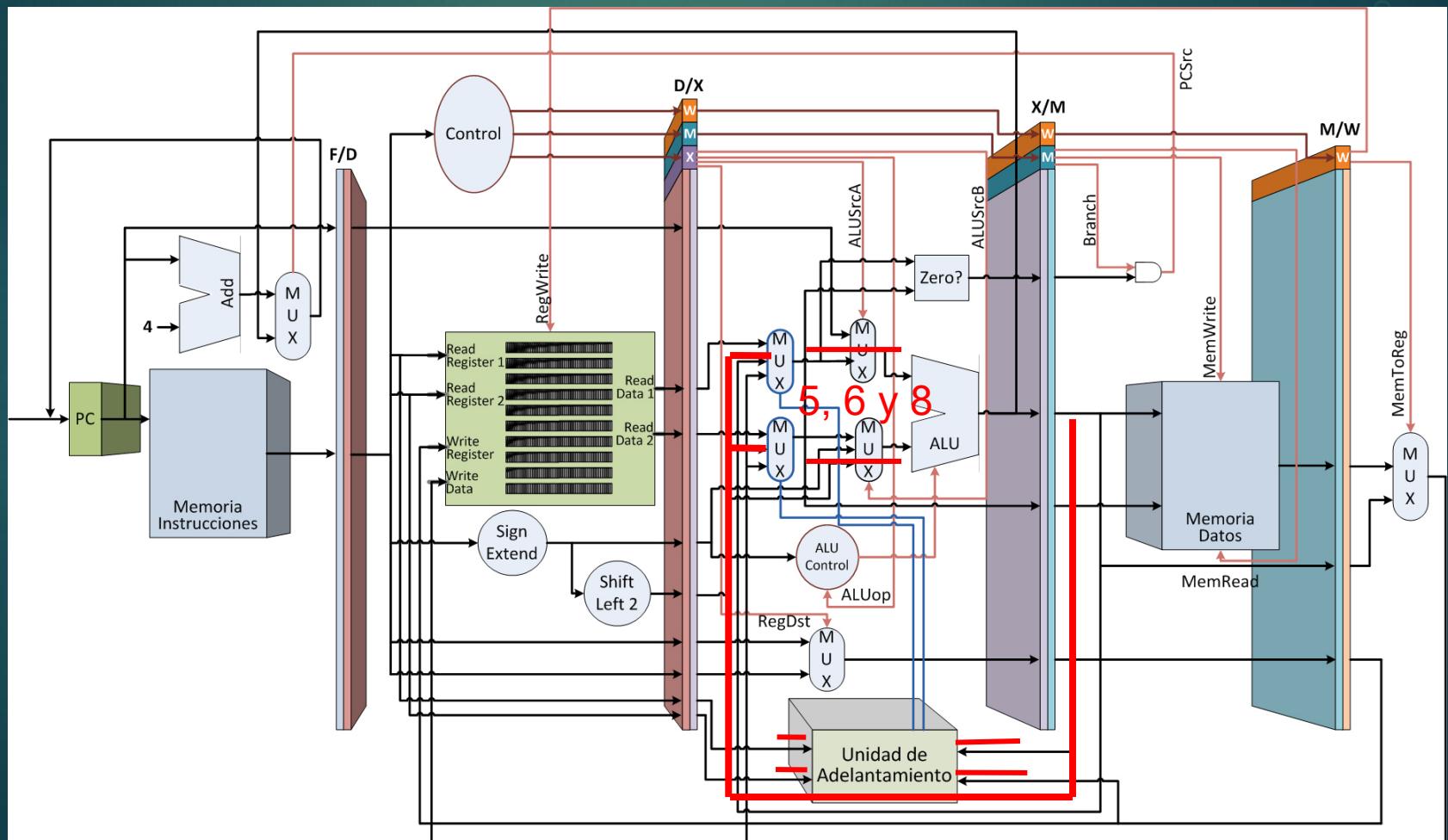
# Soluciones a los problemas de la Segmentación

Solución a conflictos por dependencia de datos con forwarding o adelantamiento

Para solucionar el tercer grupo (casos 5, 6 y 8) se requiere prever un camino de datos directo desde la salida de X a la entrada de X. Eso se puede resolver usando multiplexores (MUX) como se muestra a continuación.

# Soluciones a los problemas de la Segmentación

Solución de adelantamiento en casos 5, 6 y 8:



# Soluciones a los problemas de la Segmentación

Solución a conflictos por dependencia de datos con forwarding o adelantamiento

En la imagen anterior, la trayectoria de los datos desde la X hasta M o el BR (banco de registros) tiene una derivación que termina en los mismos multiplexores (MUX), que permiten seleccionar distintas alternativas de fuentes de datos a X (ALU) y a M (memoria de datos).

Cuando la unidad de adelantamiento detecta en los campos de las instrucciones que se están ejecutando, las dependencias 5, 6 y 8, selecciona en los MUX asociados a ella, las entradas necesarias para adelantar los datos requeridos (por X).

# Soluciones a los problemas de la Segmentación

Solución a conflictos por dependencia de datos con forwarding o adelantamiento

## Conclusión:

En los análisis anteriores se puede apreciar que existen soluciones a los conflictos por dependencia de datos que permiten reducir o eliminar el impacto de estos conflictos en la performance del procesador.

Los cambios introducidos no son complicados de implementar, y sus resultados justifican ampliamente su uso.

# Soluciones a los problemas de la Segmentación

## Solución a conflictos por dependencia de datos por software

- Otra técnica que se usa para evitar los atascos por dependencia de datos, es a través del software.
- Esta solución es realizada por el compilador, por lo que, al igual que las técnicas por hardware, es “transparente” al programador. Hay 2 posibles soluciones:
  - Introducción de instrucciones NOP (es decir, retardos) entre las instrucciones que tienen dependencia de datos, pero genera retardos que reducen la performance del procesador (equivalente al uso de puntos de parada).
  - Reordenamiento de instrucciones, para tratar de aumentar la separación entre las instrucciones con dependencia de datos, pero hay que tener cuidado con modificar el comportamiento del programa.

# Soluciones a los problemas de la Segmentación

Solución a conflictos por dependencia de datos mediante reordenamiento de instrucciones:

- 1 LW R2, 0(R1)
- 2 LW R3, 100(R0)
- 3 ADD R6, R2, R3
- 4 SUB R4, R6, R7
- 5 LW R10, 0(R0)
- 6 OR R8, R1, R10
- 7 SW R8, 100(R0)
- 8 ADD R9, R12, R10
- 9 BEQ R9, R0, etiq

- LW R2, 0(R1)
- LW R3, 100(R0)
- LW R10, 0(R0)
- ADD R6, R2, R3
- OR R8, R1, R10
- SUB R4, R6, R7
- ADD R9, R12, R10
- SW R8, 100(R0)
- BEQ R9, R0, etiq

# Soluciones a los problemas de la Segmentación

Solución a conflictos por dependencia de datos mediante reordenamiento de instrucciones:

En la imagen anterior se muestra a la izquierda el programa original (de 9 instrucciones), y a la derecha, el programa reordenado por el compilador al detectar conflictos de dependencia de datos.

Por ejemplo, hay dependencias entre las instrucciones 2 y 3 (registro R3), y entre la 3 y la 4 (registro R6). La instrucción 4 no se puede reubicar porque tiene dependencia con la 3, pero la instrucción 5 no depende de ninguna de las anteriores. Entonces, reubicando la instrucción 5 a continuación de la 2, se elimina la dependencia sin alterar la lógica del programa.

El compilador continúa con este proceso intentando reordenar las instrucciones para evitar los conflictos. En caso de no poder resolverlos, deberá insertar instrucciones de NO-OPERACIÓN (NOP) para eliminar los que persisten luego del reordenado.

# Soluciones a los problemas de la Segmentación

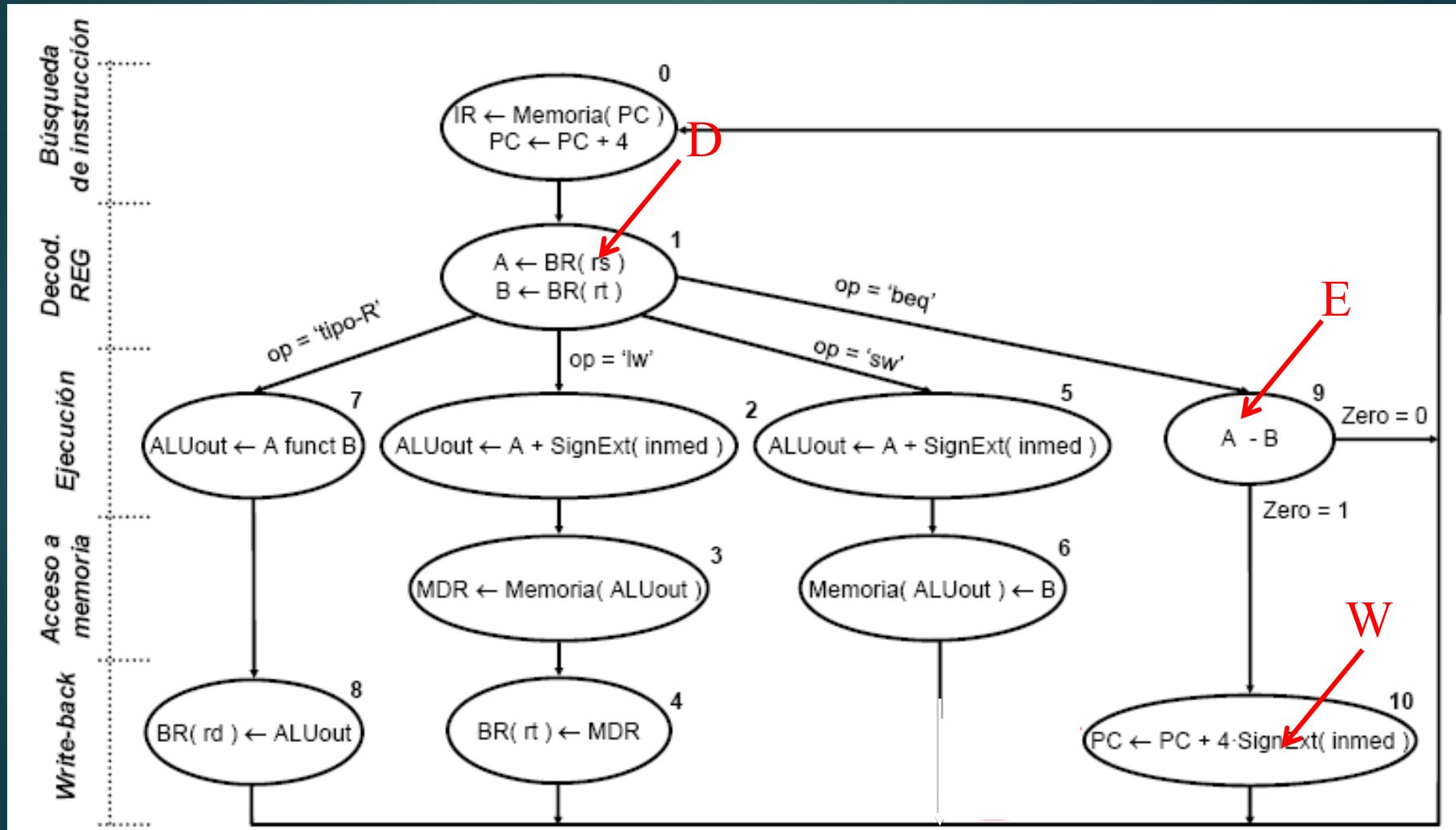
## Solución a los conflictos por dependencia de control

El tercer tipo de riesgo que puede haber en la segmentación es la presencia de instrucciones de salto, que pueden ser de 2 tipos:

- Incondicional: donde la dirección de destino se debe determinar lo más pronto posible, dentro del cauce, para reducir la penalización.
- Condicional: introduce el riesgo adicional por la dependencia entre la condición de salto y el resultado dependiente de una instrucción previa.
- Por ejemplo: en la instrucción de salto (condicional) BEQ en el nanoMIPS, recién se calcula la dirección de salto durante la fase W. Esta situación se muestra en la figura siguiente.

# Soluciones a los problemas de la Segmentación

## Instrucción de salto condicional BEQ en el nanoMIPS.

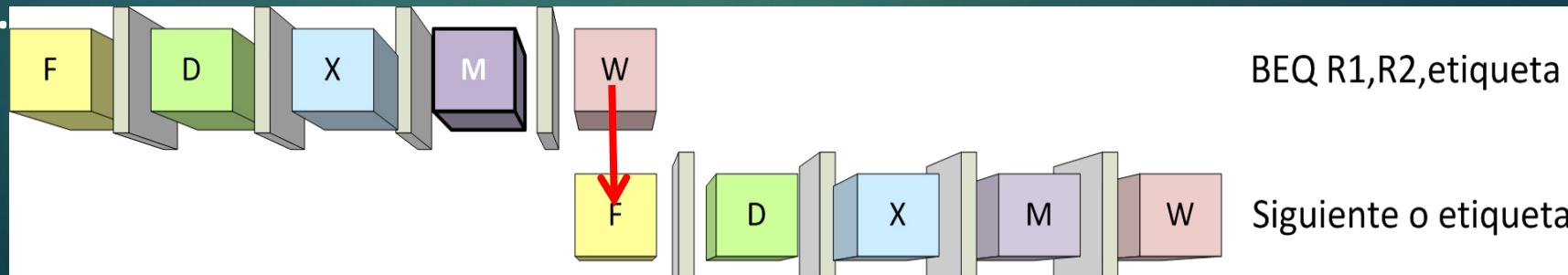


# Soluciones a los problemas de la Segmentación

## Solución a los conflictos por dependencia de control

Igual que antes, la forma más elemental de resolver el conflicto es agregando tiempos muertos (ciclos de parada) hasta que desaparece el problema.

En el caso del nanoMIPS, se requerirían 3 ciclos de reloj para empezar la instrucción siguiente a la de salto, considerando que se puede calcular el salto en el primer semiciclo del ciclo W de la instrucción BEQ, y se puede leer el resultado en el segundo semiciclo del ciclo F de la instrucción a donde se salta.



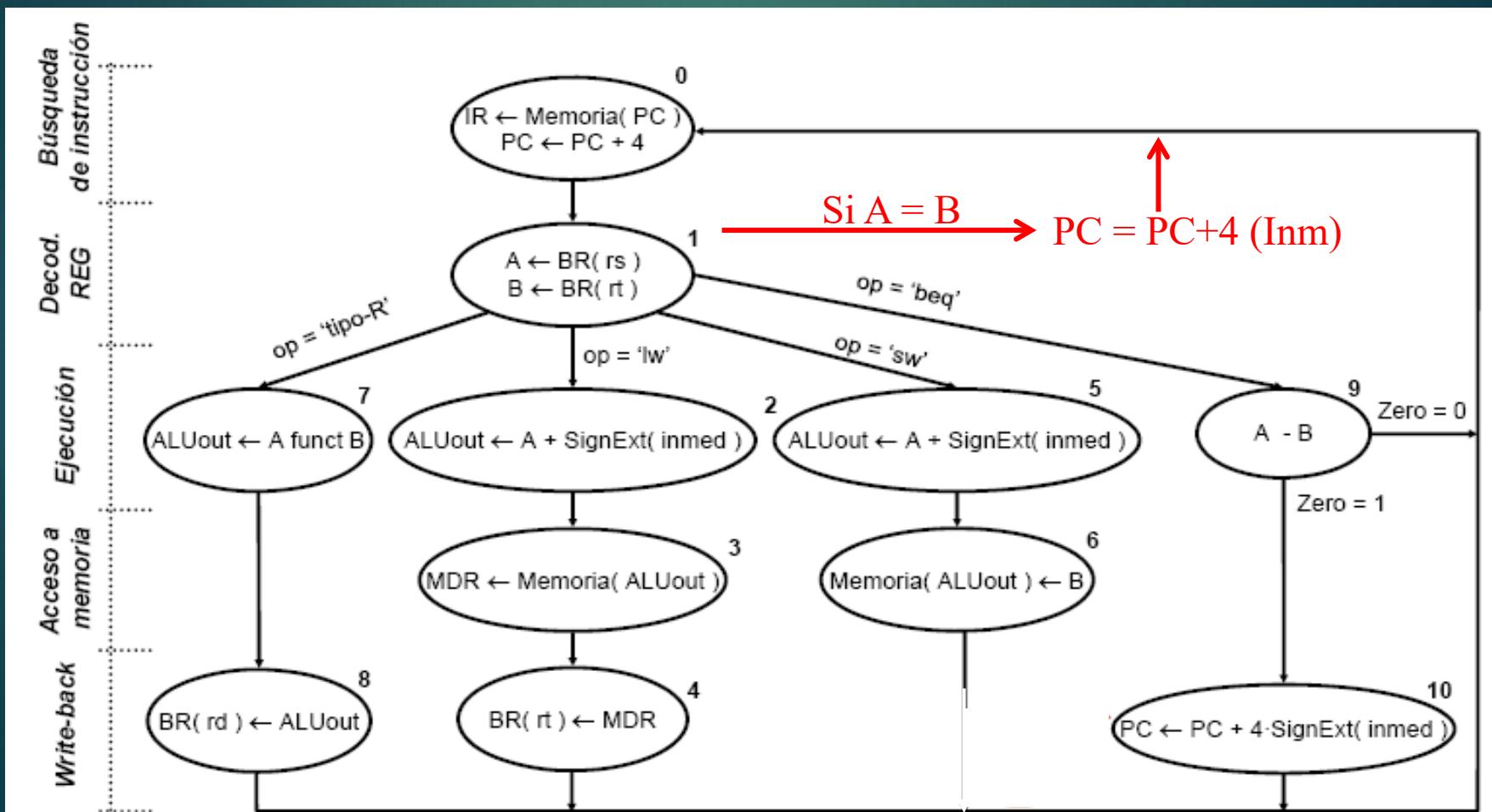
# Soluciones a los problemas de la Segmentación

Solución a los conflictos por dependencia de control – penalización con adelantamiento

- Se puede mejorar la situación anterior si se modifica la ruta de datos para reducir la cantidad de ciclos muertos, adelantando la resolución de los saltos a la etapa D, donde se decodifica y se accede a los registros.
- El objetivo del adelantamiento es evaluar tempranamente la condición de salto, usando, por ejemplo, un restador extra a la salida del banco de registros, en lugar de esperar a la etapa X asociada a la ALU, para determinar la condición de igualdad de la instrucción BEQ.
- Además se puede agregar un sumador adicional para calcular la dirección de salto, en lugar de esperar a la etapa X con la ALU, para adelantar la obtención del salto.

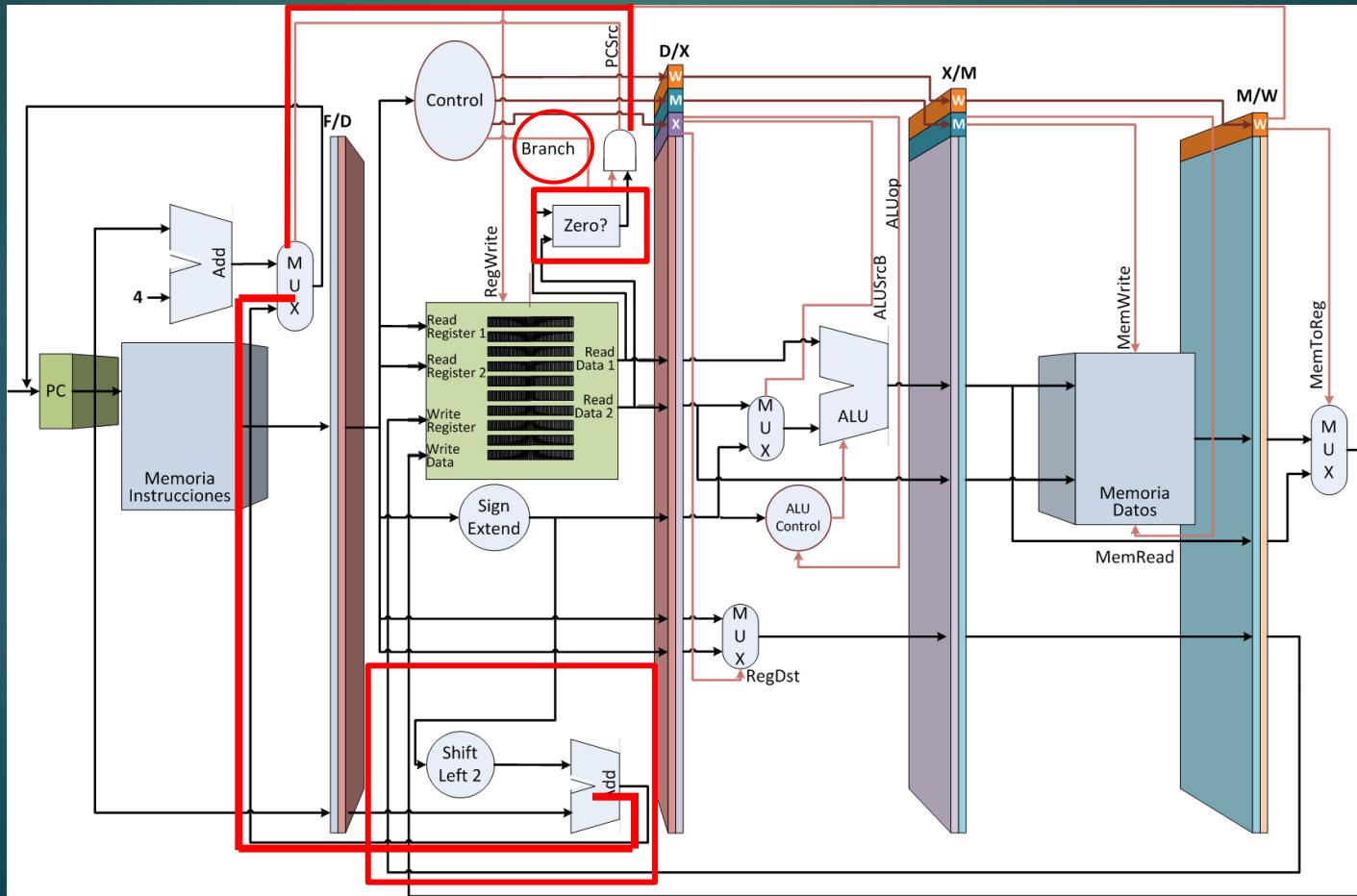
# Soluciones a los problemas de la Segmentación

Adelantamiento en el nanoMIPS a la resolución de la instrucción BEQ y cálculo de la próxima instrucción



# Soluciones a los problemas de la Segmentación

Adelantamiento en el nanoMIPS a la resolución de la instrucción BEQ y cálculo de la próxima instrucción



# Soluciones a los problemas de la Segmentación

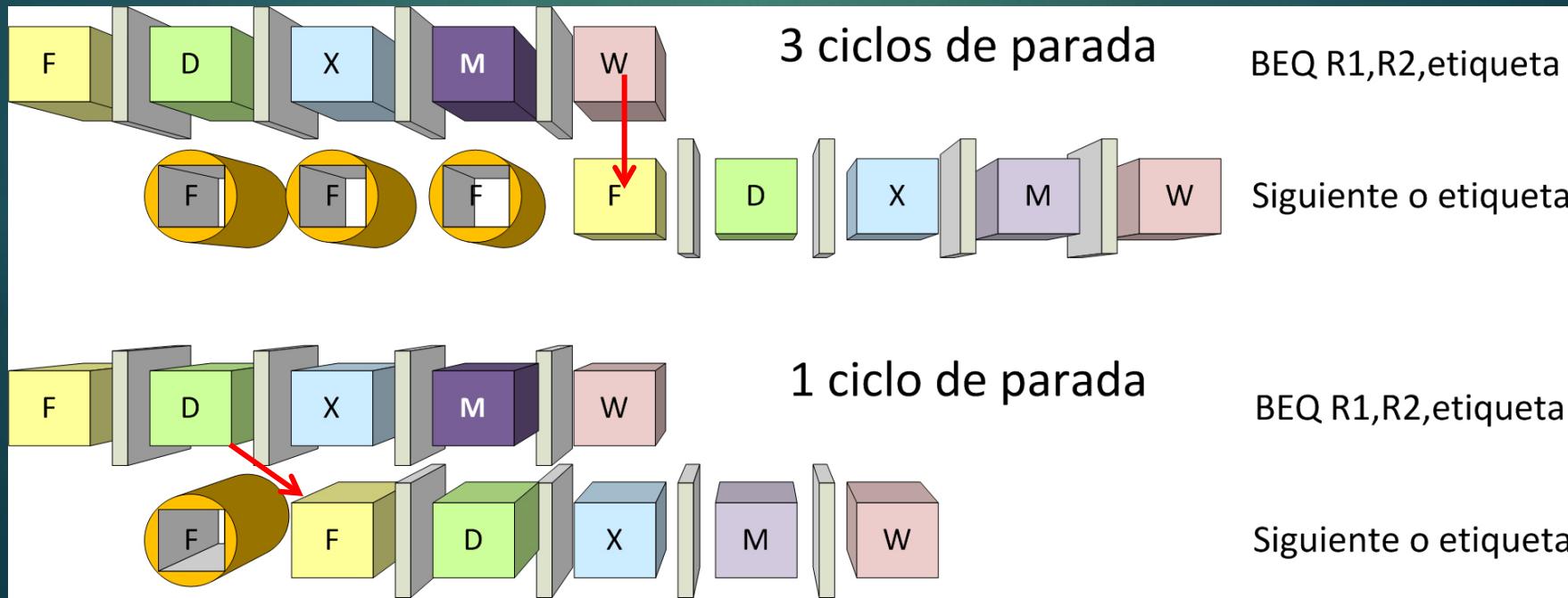
Adelantamiento en el nanoMIPS a la resolución de la instrucción BEQ y cálculo de la próxima instrucción

- Se puede ver en la imagen anterior que a la salida del BR se ha agregado una lógica para detectar en forma temprana la condición de igualdad en una instrucción de BEQ, que controla un MUX.
- El MUX selecciona el valor con el que se va a cargar el PC.
- Una de las entradas al MUX es el valor actual del PC con el desplazamiento incluido en la instrucción de BEQ.
- El hardware agregado determina:
  - si: la instrucción es BEQ, y A=B
  - entonces:  $PC = PC + \text{desplazamiento}$
  - donde desplazamiento se calcula con el modo de direccionamiento.

# Soluciones a los problemas de la Segmentación

Adelantamiento en el nanoMIPS a la resolución de la instrucción BEQ y cálculo de la próxima instrucción

Con la ruta de datos modificada, la respuesta del nanoMIPS a una instrucción de salto BEQ se pierde 1 solo ciclo de reloj, en lugar de los 3 que se perdían originalmente.



# Soluciones a los problemas de la Segmentación

## Técnicas de tratamiento de saltos

En máquinas más complejas, la solución antes presentada para el nanoMIPS puede no ser tan sencilla de implementar. En general las técnicas para mitigar el impacto, en la performance de procesadores segmentados, de las instrucciones de salto pueden ser de 2 tipos:

- Técnicas por hardware: que consiste en resolver los conflictos a nivel de hardware mediante predicción de saltos.
- Técnicas por software: que intentan resolver los conflictos por software a nivel del compilador.

# Soluciones a los problemas de la Segmentación

## Técnicas de resolución de instrucciones de salto por hardware

Estas técnicas se basan en usar estrategias para predecir el resultado de la instrucción de salto. Pueden ser de 2 tipos:

- Técnicas estáticas: no tienen en cuenta información previa de la ejecución del programa.
- Técnicas dinámicas: tienen en cuenta la historia previa del programa en ejecución.

# Soluciones a los problemas de la Segmentación

## Técnicas de predicción de saltos por hardware - estática

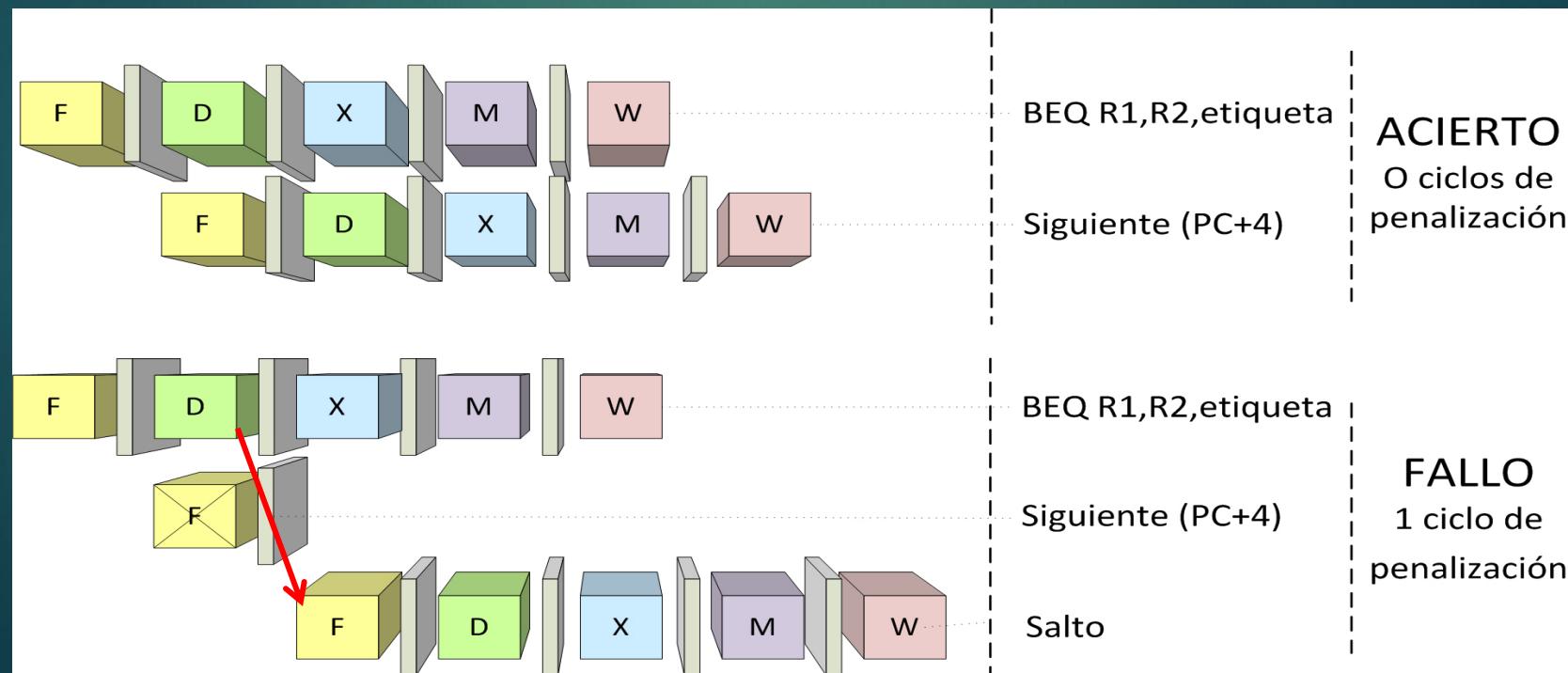
Básicamente se presume una condición (salta o no salta) y se ejecuta en base a esa predicción.

- Predecir que nunca se salta: asume que el salto no se producirá, y por lo tanto siempre capta la siguiente instrucción.
- Predecir que siempre se salta: asume que el salto se producirá, y por lo tanto siempre capta la instrucción destino del salto.

# Soluciones a los problemas de la Segmentación

## Técnicas de predicción de saltos por hardware - estática

En el nanoMIPS siempre capta la próxima instrucción, es decir predice que no va a saltar. Si no salta no se pierde ningún ciclo, si salta se pierde 1 ciclo.



# Soluciones a los problemas de la Segmentación

## Técnicas de predicción de saltos por hardware - estática

En el nanoMIPS siempre capta la próxima instrucción, es decir predice que no va a saltar. Si no salta no se pierde ningún ciclo, si salta se pierde 1 ciclo.

- Ventajas:
  - Sencilla de implementar
  - Pequeños cambios en la Unidad de control
  - Ruta de datos sin cambios
- Desventajas:
  - La instrucción a descartar puede afectar el estado del procesador (registros y/o memoria).
  - En el nanoMIPS el ciclo que se descarta es de búsqueda de la instrucción (ciclo F) que no modifica el procesador.

# Soluciones a los problemas de la Segmentación

## Técnicas de predicción de saltos por hardware - dinámica

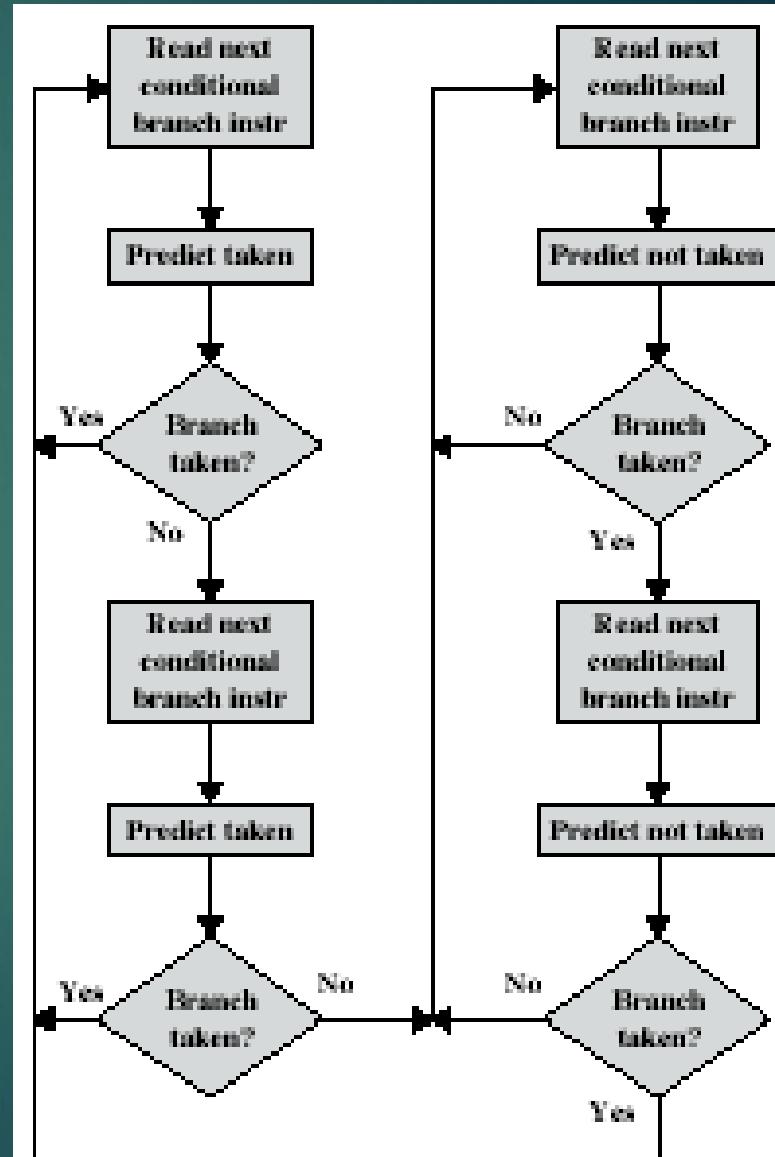
- Esta técnica se basa en el análisis de datos previos (la “historia”) sobre como se comporta el programa, para predecir la acción.
- Algunos ejemplos de técnicas de predicción por hardware son:
  1. Comutador Saltar/no saltar
  2. Tabla de historia de saltos (branch history table)
  3. Predicción según el código de operación
  4. Varios cauces (uno por cada opción de salto) (Multiple stream)
  5. Precaptación del destino del salto (Prefetch branch target)
  6. Buffer de bucles

# Soluciones a los problemas de la Segmentación

## Comutador Saltar/no saltar

Estando en uno de las 2 condiciones (predecir que salta o predecir que no salta), se requieren 2 predicciones fallidas consecutivas para conmutar al otro estado.

Esta predicción tiene un buen comportamiento en lazos donde el resultado de una consulta (por ejemplo un contador que tiene que llegar a 0) se puede repetir muchas veces antes de que cambie el resultado de la consulta.



# Soluciones a los problemas de la Segmentación

## Tabla de historia de saltos (BHT branch history table)

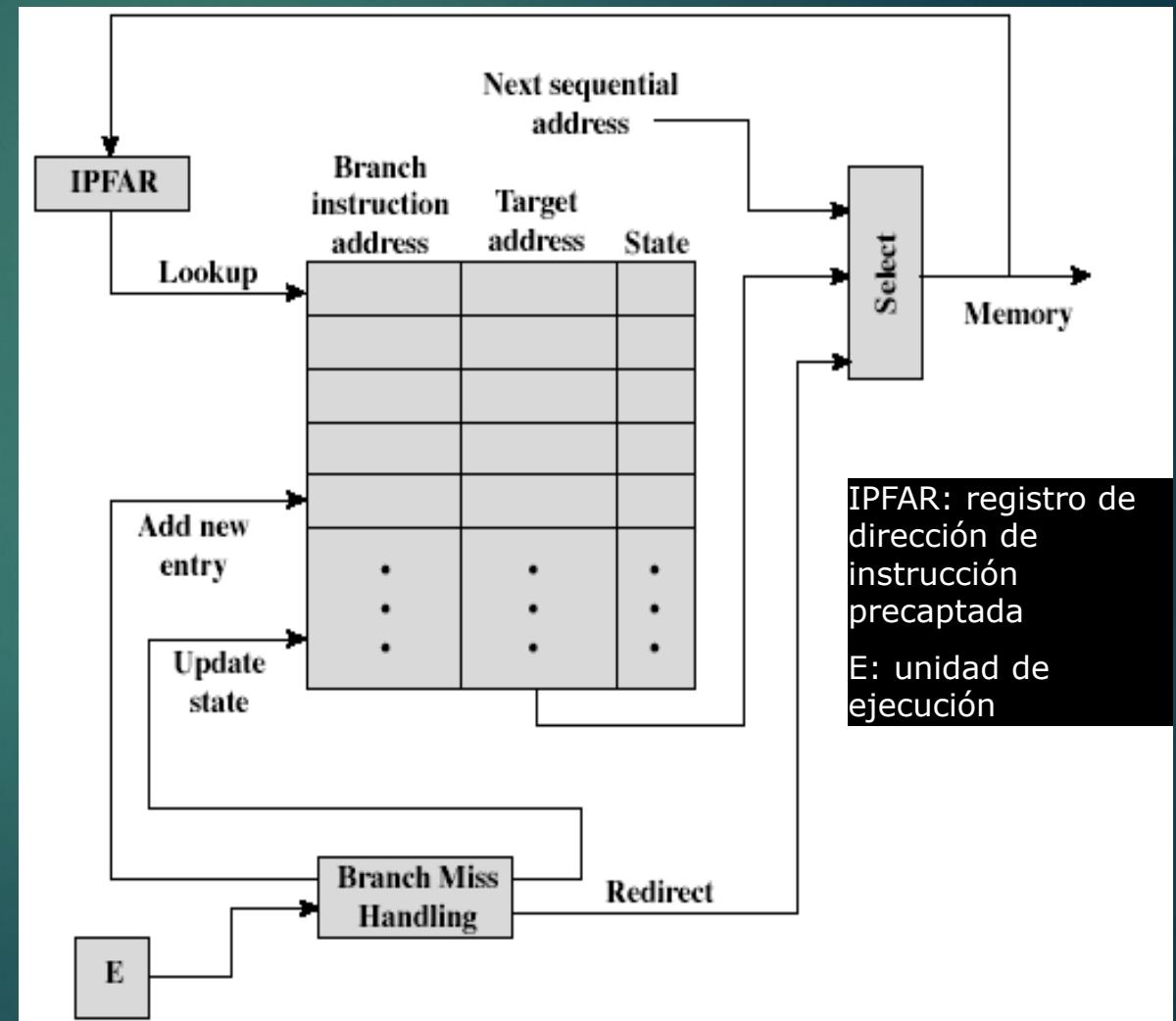
Se implementa usando una pequeña cache asociada a la etapa de búsqueda (F). La cache tiene 3 campos:

- Dirección de una instrucción que es del tipo bifurcación
  - Dirección del destino ó Instrucción destino
  - N bits de estado (historia de uso)
- Cada vez que se preculta una instrucción, se busca en la cache BHT si la instrucción es de ramificación.
- Se toma una decisión predictiva en función de los bits de la historia de uso.
- Si la predicción es saltar, se capta la dirección de salto.
- Si la predicción falla, se actualiza la tabla
- Si la instrucción de ramificación no estaba en la BHT, se la carga como nueva entrada.

# Soluciones a los problemas de la Segmentación

## BHT

Tabla caché para almacenar instrucciones de ramificación con las direcciones de destino para el caso de predecir el salto.



# Soluciones a los problemas de la Segmentación

## Predicción según el código de operación

- Se basa en la suposición de que algunas instrucciones de salto tienen mayor probabilidad de saltar o de no saltar.
- Por ejemplo, si la instrucción de salto involucra un lazo que se repite muchas veces, la condición que se usa para repetir el lazo ocurrirá muchas más veces que la que se usa para salir del lazo (que ocurrirá 1 sola vez).
- La tasa de acierto puede llegar a alcanzar un 75%

# Soluciones a los problemas de la Segmentación

## Varios cauces (uno por cada opción de salto- Multiple stream)

- En esta solución se usan cauces distintos (duplicando hardware) para ejecutar simultáneamente los cauces, el que corresponde a la opción de no saltar y el que corresponde a la instrucción de salto.
- Luego, cuando se determina el resultado de la ramificación, se debe utilizar el cauce correcto y descartar el incorrecto.
- Como se están captando simultáneamente 2 cauces distintos, aumentan las búsquedas, lo que puede provocar retardos en el acceso al bus y a los registros.
- Si hay nuevos saltos en los cauces en ejecución, se necesita disponer de más cauces para contemplar esta situación, lo que aumenta la complejidad del hardware.

# Soluciones a los problemas de la Segmentación

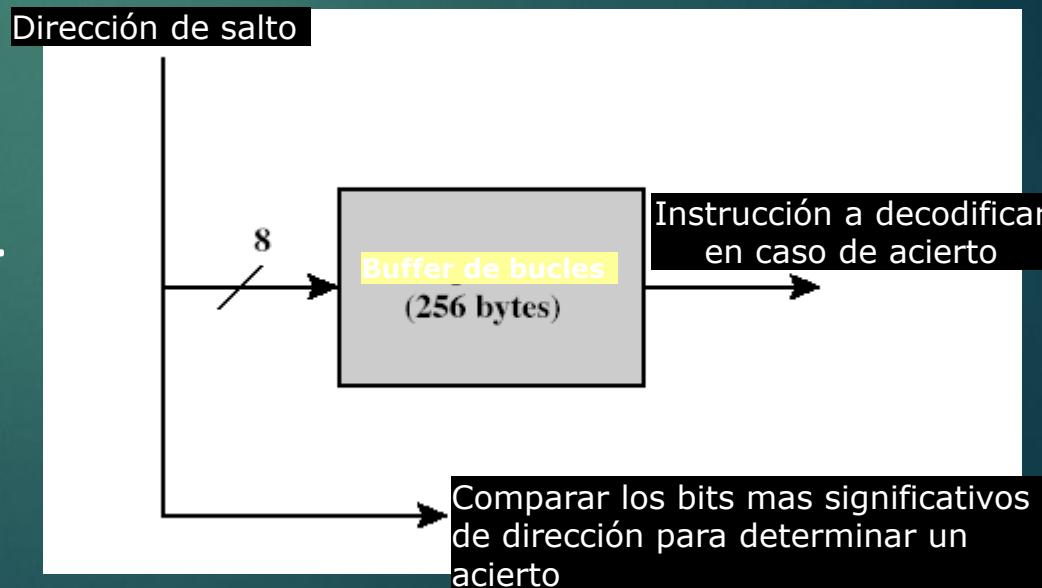
## Precaptación del destino del salto (Prefetch branch target)

- Se precapta la instrucción destino del salto, además de las instrucciones siguientes a la bifurcación.
- La instrucción se guarda hasta que se ejecute la instrucción de bifurcación.
- El IBM 360/91 usa este método.

# Soluciones a los problemas de la Segmentación

## Buffer de bucles

- Se usa una memoria muy rápida, gestionada por la etapa de captación de instrucciones, que contiene las últimas instrucciones recientemente buscadas.
- Si hay una instrucción de salto, el hardware comprueba si está en el Buffer de bucles. Si es así, la próxima instrucción se busca desde el buffer.
- Muy eficaz para pequeños bucles y saltos, si el buffer es capaz de contener todo el bucle. Funciona parecido a la cache, solo que contiene instrucciones consecutivas, únicamente.



# Soluciones a los problemas de la Segmentación

## Técnicas de resolución de instrucciones de salto por software

- Las técnicas de resolución por software de conflictos debido a las instrucciones de salto se basan en tratar de realizar trabajo útil mientras el salto se resuelve.
- Cuando hay una instrucción de salto, se necesita 1 ciclo (o más) de tiempo para determinar si se va a ejecutar el salto o no. Ese tiempo se llama hueco o ranura de retardo de salto (Branch delay slot).
- Hay máquinas que captan y ejecutan siempre la instrucción siguiente a una instrucción de ramificación (en lugar de descartarla).
- El compilador puede tratar de insertar, en dichos huecos, instrucciones útiles que en lo posible no dependan del salto. De esta manera se elimina el conflicto y se hace trabajo útil.
- Si no es posible insertar instrucciones, se necesita agregar instrucciones del tipo NO-OPERACION (NOP) para evitar el conflicto.

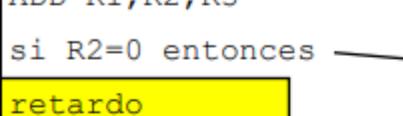
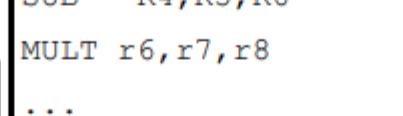
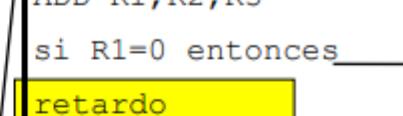
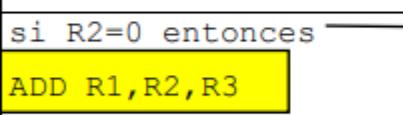
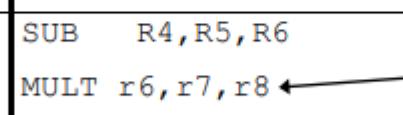
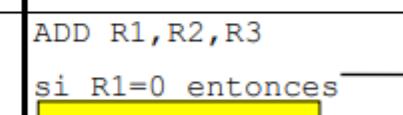
# Soluciones a los problemas de la Segmentación

## Técnicas de resolucion de instrucciones de salto por software

- La forma de llenar los huecos que quedan luego de las instrucciones de salto es mediante el reordenamiento de las instrucciones. Ese trabajo lo hace el compilador.
- Las instrucciones a reordenar pueden provenir de:
  - Caso 1: Instrucciones antes del salto
  - Caso 2: Instrucciones del destino del salto
  - Caso 3: Instrucciones a continuación del salto
- La mejor solución sería reordenar instrucciones anteriores a la de salto, porque siempre deben ejecutarse y van a ser útiles.
- El reordenamiento con instrucciones del destino del salto o a continuación del salto deben ser tal que no modifiquen el estado del proceso aun cuando falle la predicción.

# Soluciones a los problemas de la Segmentación

Proceso de reordenamiento de instrucciones en máquinas con delay-slot (casos 1, 2 y 3):

| Caso 1   | Caso 2   | Caso 3  |
|--|--|---|
| <b>Antes:</b><br>ADD R1,R2,R3<br>si R2=0 entonces<br><br>...<br>MULT r6,r7,r8 | <b>Antes:</b><br>SUB R4,R5,R6<br>MULT r6,r7,r8<br>...<br>ADD R1,R2,R3<br>si R1=0 entonces<br><br>... | <b>Antes:</b><br>ADD R1,R2,R3<br>si R1=0 entonces<br><br>... |
| <b>Despues:</b><br><br>ADD R1,R2,R3<br>...<br>MULT r6,r7,r8                 | <b>Despues:</b><br>SUB R4,R5,R6<br><br>...   | <b>Despues:</b><br>ADD R1,R2,R3<br><br>...                 |

# Soluciones a los problemas de la Segmentación

## Técnicas de resolucion de instrucciones de salto por software

- En el ejemplo anterior:
  - Caso 1: el hueco se rellena con una instrucción anterior al salto. Sirve siempre, es la mejor opción (no siempre es posible).
  - Caso 2: Como la instrucción ADD R1,R2,R3 se requiere ejecutar antes de la instrucción de salto, no se puede reubicar después del salto. El compilador busca una instrucción a reubicar, en el lugar a donde puede llegar a saltar, por ejemplo SUB R4,R5,R6. La copia en el hueco, y corrige el lugar a donde salta (MULT r6,r7,r8). Si la instrucción de salto efectivamente salta, entonces la instrucción insertada en el hueco (SUB R4,R5,R6) sirve y se adelantó trabajo. Si la instrucción no salta, la instrucción insertada (SUB R4,R5,R6) se ejecuta pero no sirve y el resultado no debe alterar la lógica. En otras palabras, el valor que se cargue en R4 (en la instrucción SUB R4,R5,R6) no sirve y no debe alterar la lógica del programa.

# Soluciones a los problemas de la Segmentación

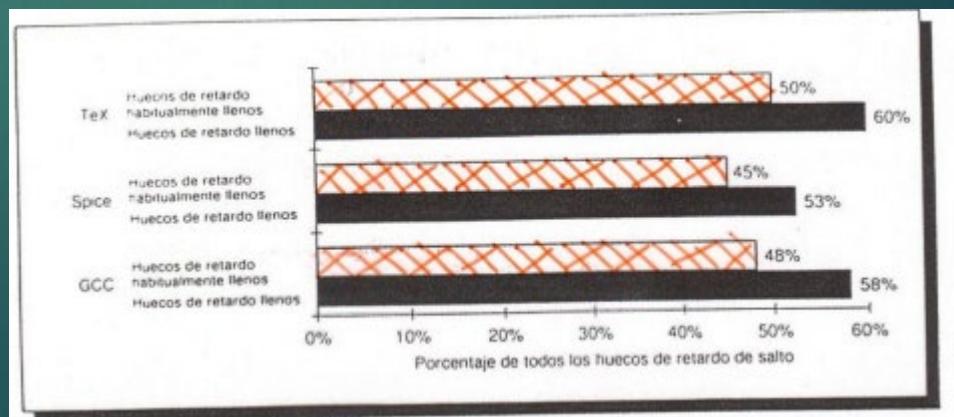
## Técnicas de resolucion de instrucciones de salto por software

- Caso 3: Como la instrucción ADD R1,R2,R3 se requiere ejecutar antes de la instrucción de salto, no se puede reubicar después del salto. El compilador busca una instrucción a reubicar, a continuación de la de salto, por ejemplo SUB R4,R5,R6. Si la instrucción no salta, entonces la instrucción insertada en el hueco (SUB R4,R5,R6) sirve y se adelantó trabajo. Si la instrucción salta, la instrucción insertada (SUB R4,R5,R6) se ejecuta pero no sirve y el resultado no debe alterar la lógica. En otras palabras, el valor que se cargue en R4 (en la instrucción SUB R4,R5,R6) no sirve y no debe alterar la lógica del programa.

# Soluciones a los problemas de la Segmentación

## Técnicas de resolucion de instrucciones de salto por software

- En la imagen inferior se muestran estudios realizados en 3 máquinas con delay-slot. Se observa que:
  - Aproximadamente el 60% de los huecos se llenan con instrucciones diferentes a NOP (barra oscura en la imagen inferior)
  - Del 60% de huecos llenados con instrucciones (los 3 casos vistos antes), entre el 80 y 85% corresponde a instrucciones útiles (barras rayadas).
  - En definitiva, el 50% de los huecos (retardos) se aprovechan con instrucciones útiles.



# Segmentación en el i80486

Como ejemplo de procesador convencional (tipo CISC) se puede analizar el procesador de Intel 80486, que tiene un cauce compuesto por 5 etapas, 2 de ellas de decodificación:

- Fetch (F)
- Decode 1 (D1)
- Decode 2 (D2)
- Execute (EX)
- Write back (WB)
- En las figuras siguientes se muestran 3 casos de secuencias de instrucciones y el comportamiento del cauce para cada secuencia.

# Segmentación en el i80486

## Secuencia de instrucciones sin penalización

3 instrucciones de movimiento entre memoria y registro sin retrasos.

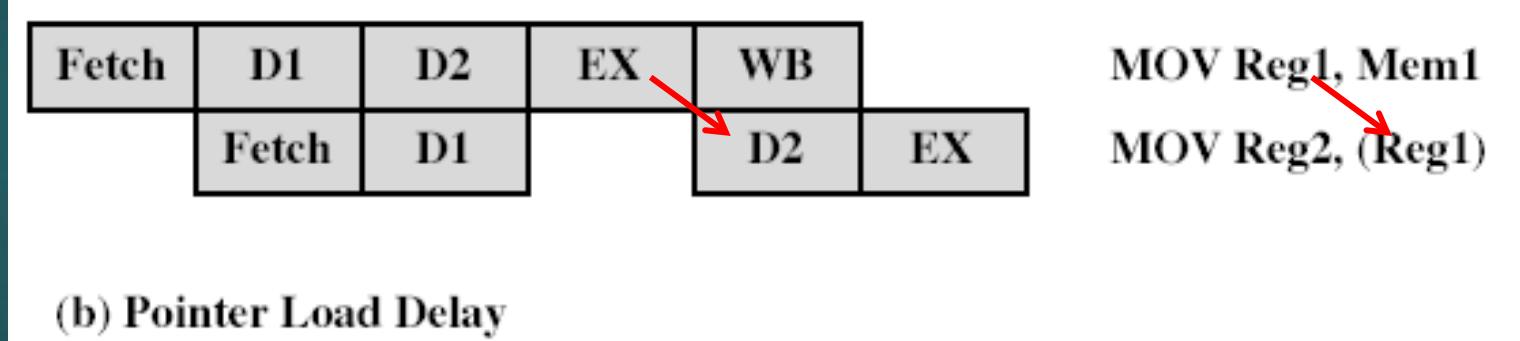
|       |       |       |    |    |    |                |                |
|-------|-------|-------|----|----|----|----------------|----------------|
| Fetch | D1    | D2    | EX | WB |    | MOV Reg1, Mem1 |                |
|       | Fetch | D1    | D2 | EX | WB | MOV Reg1, Reg2 |                |
|       |       | Fetch | D1 | D2 | EX | WB             | MOV Mem2, Reg1 |

No hay retardo en el cauce por carga de un dato desde memoria.

# Segmentación en el i80486

## Secuencia de instrucciones con penalización de 1 ciclo

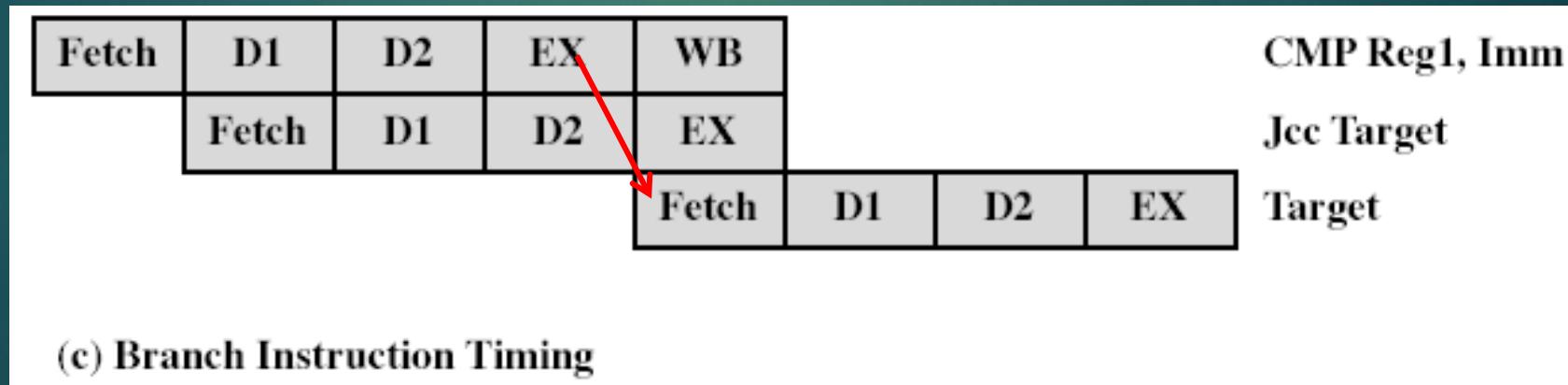
Retardo de 1 ciclo en el cauce por carga que utiliza un puntero (con adelantamiento desde la etapa EX a la D2).



# Segmentación en el i80486

## Secuencia de instrucciones con penalización de 2 ciclos

Retardo de 2 ciclos en el cauce por temporizado en una instrucción de bifurcación

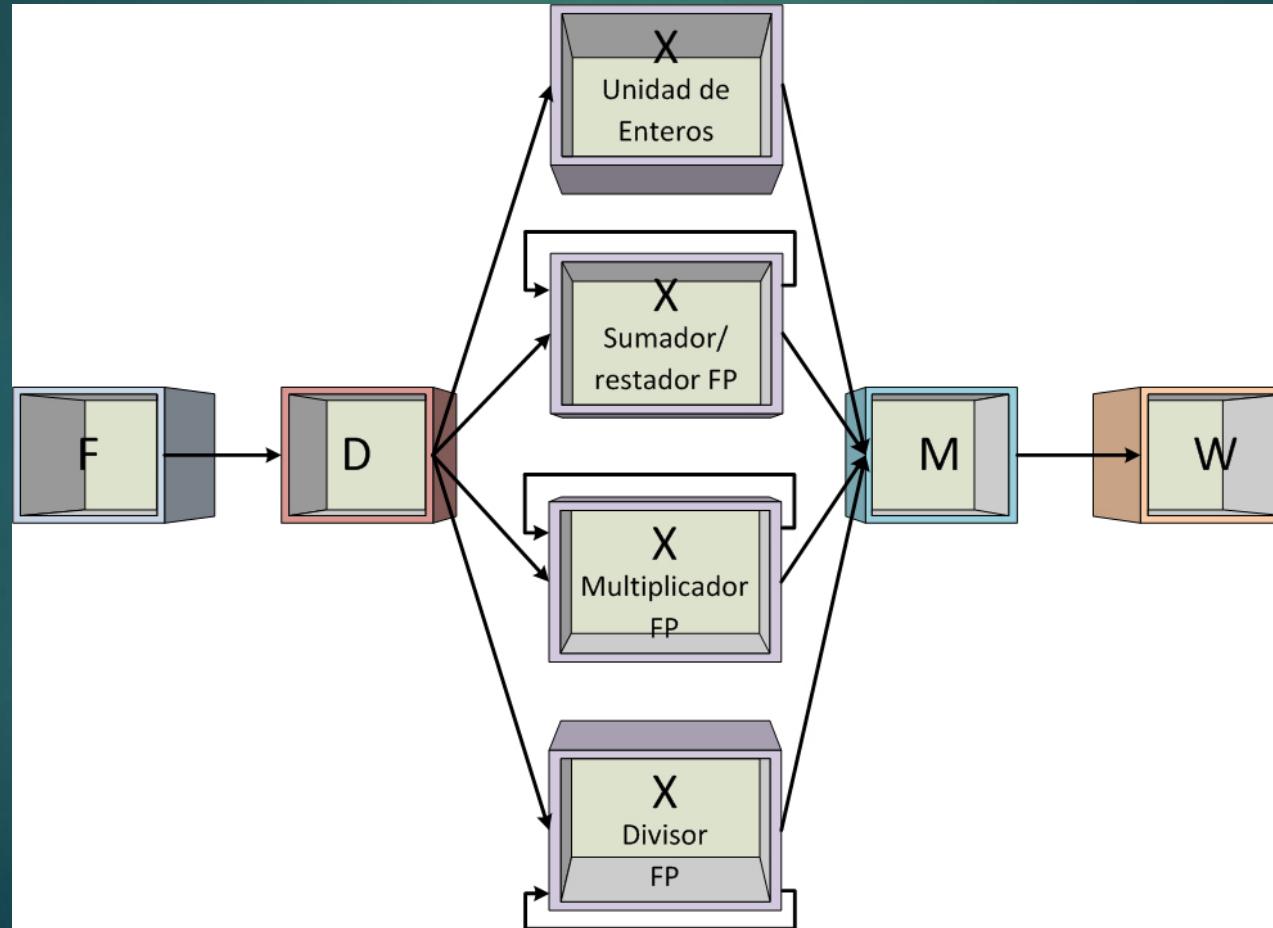


# Procesador segmentado nanoMIPS multifuncional

- En general es imprescindible que los procesadores dispongan de operaciones en punto flotante. El modelo visto del nanoMIPS no las tiene.
- Si se quiere agregar operaciones en punto flotante en el nanoMIPS, se requiere realizar las siguientes modificaciones:
  - Agregar instrucciones en PF (Ej: ADD.D, LD.D)
  - Agregar registros de PF (Ej: F1, F2, etc.)
  - Agregar hardware para las operaciones aritméticas de: Suma, Resta, Multiplicación y División
- Es posible compatibilizar la segmentación con las operaciones en punto flotante. La primera solución es la que se muestra en la figura siguiente.

# Procesador segmentado nanoMIPS multifuncional

Esquema del nanoMIPS segmentado con Unidades de punto flotante



# Procesador segmentado

## nanoMIPS multifuncional

- El problema que tiene el esquema anterior es que los tiempos de ejecución de las unidades de punto flotante son muy distintos de los de la unidad de enteros.
- Hasta ahora habíamos considerado que todas las etapas del nanoMIPS tenían la misma duración, pero ahora las unidades funcionales de PF pueden requerir varios ciclos para completarse.
  - Por ejemplo, la división en punto flotante lleva muchos mas ciclos que una suma de números enteros
- Conclusión: cuando se incorpora punto flotante, no todas las etapas del cauce van a durar lo mismo.

# Procesador segmentado

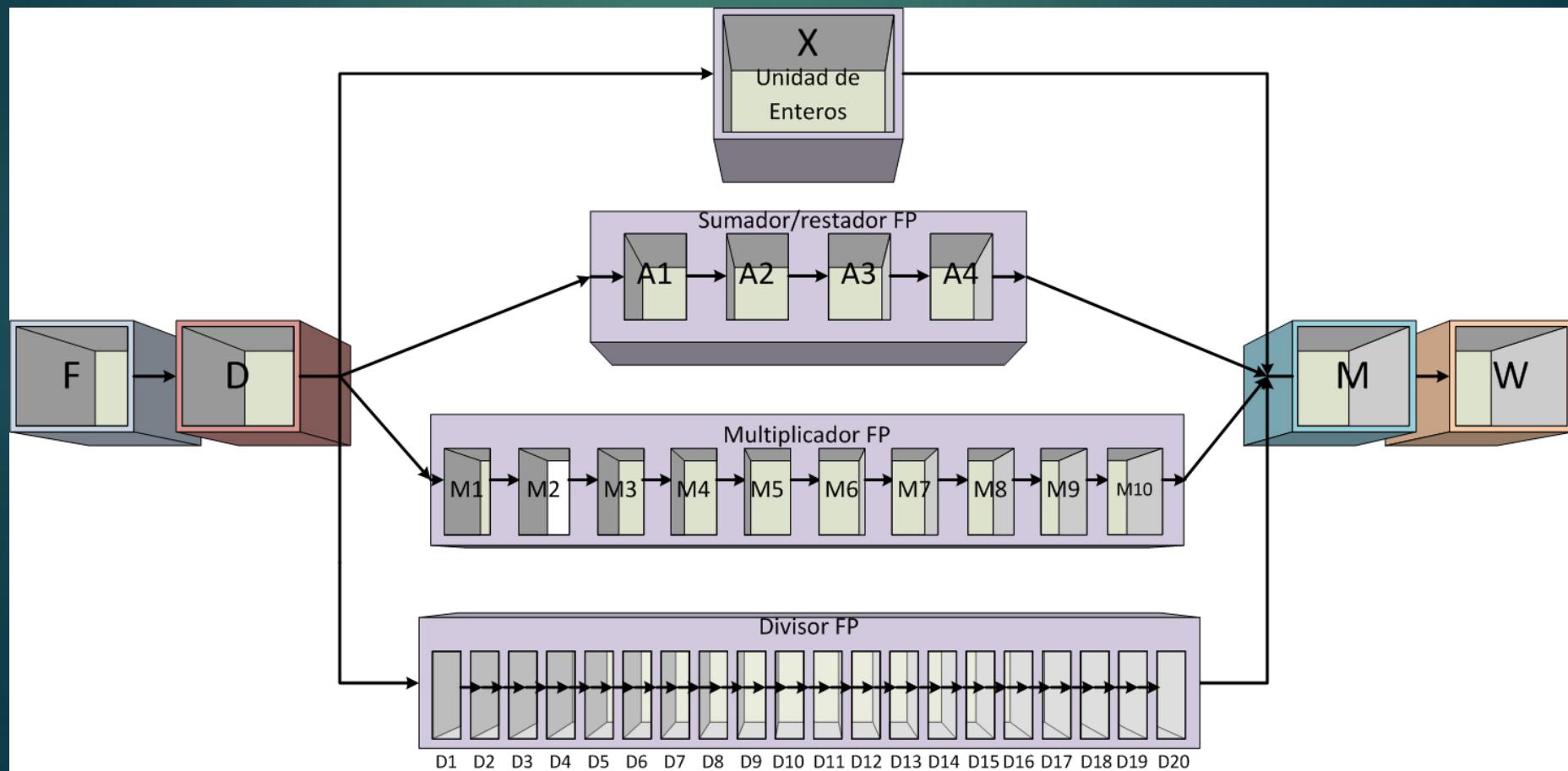
## nanoMIPS multifuncional

- Si las unidades funcionales en punto flotante tardan varios ciclos, entonces una secuencia de 2 instrucciones consecutivas en punto flotante requeriría retardar la segunda instrucción hasta que la primera se complete.
- En el siguiente ejemplo se muestra cual es la situación:
  - ADD.D F1, F2, F3
  - ADD.D F4, F5, F6
- No se puede empezar la segunda instrucción ADD.D F4,F5,F6 hasta que no se termine la primera.
- Hay una solución a este problema: segmentar las unidades funcionales de punto flotante.
- Esta solución se muestra en la figura siguiente.

# Procesador segmentado

## nanoMIPS multifuncional

Esquema del nanoMIPS segmentado con Unidades de punto flotante segmentadas



# Procesador segmentado

## nanoMIPS multifuncional

- Segmentando las unidades de punto flotante se pueden iniciar nuevas instrucciones en PF sin haber completado a actual.
- En la figura anterior se ha segmentado:
  - La unidad de suma en PF en 4 etapas
  - La Unidad de multiplicación en PF en 10 etapas
  - La unidad de división en PF en 20 etapas.

# Referencias

- W. Stallings, 5º Ed - Capítulo 11.
- Hwang & Briggs – Computer Architecture & Parallel Processing – Chapter 10.