[Resumen clase 5](https://drive.google.com/file/d/1-U2Sw3BmQppBUVj00MFV73XvtLt7dGIl/view?usp=sharing)

# Posibles soluciones a atascos

Si resolvemos con paradas del cauce, disminuye el rendimiento teórico y CPI>1.

## Soluciones a riesgos estructurales

Simple: replicar, segmentar o realizar turnos para el acceso a las unidades funcionales en conflicto.

* Duplicación de recursos de hardware: sumadores o restadores además de la ALU
* Separación en memorias de instrucciones y datos
* Turnar el acceso al banco de registros
  + Escrituras en la 1° mitad de los ciclos de reloj
  + Lecturas en la 2° mitad de los ciclos de reloj

## Soluciones a riesgos de datos

Para riesgos RAW: se debe determinar cómo y cuándo aparecen esos riesgos.

Será necesaria una unidad de detección de riesgos y/o compilador más complejo.

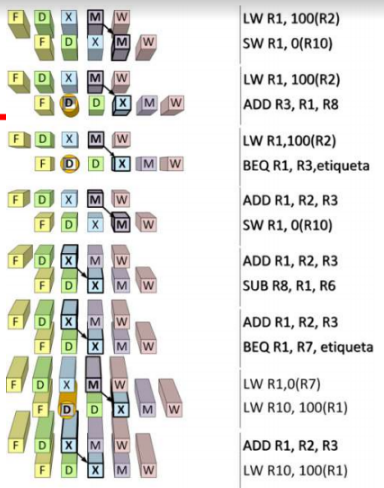
Dos soluciones:

* Hardware: adelantamiento de operandos (forwarding)
* Software: instrucciones NOP o reordenación del código

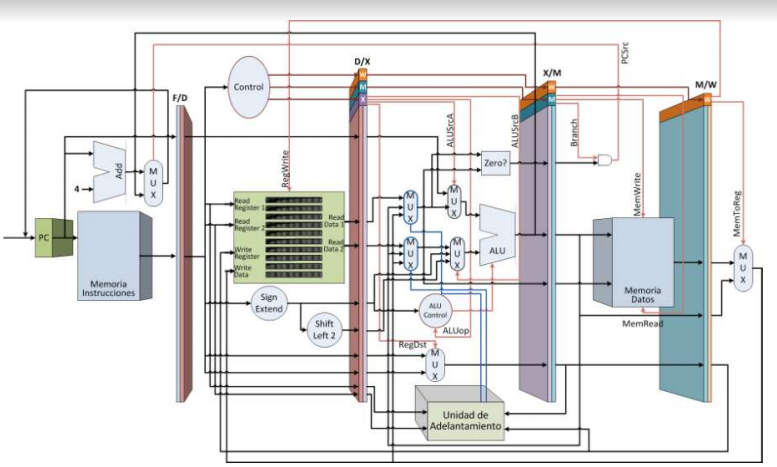
### Técnica de hardware (Adelantamiento, forwarding o cortocircuito)

* Consiste en pasar directamente el resultado obtenido con una instrucción a las instrucciones que lo necesitan como operando
  + Si el dato necesario está disponible a la salida de la ALU (Xi) se lleva a la entrada de la etapa correspondiente (Xi+1) sin esperar a la escritura (Mi o Wi)
* Fácil de implementar si se identifican todos los adelantamientos y se comunican a los registros de segmentación correspondientes

Riesgos RAW



Ruta de datos con adelantamiento



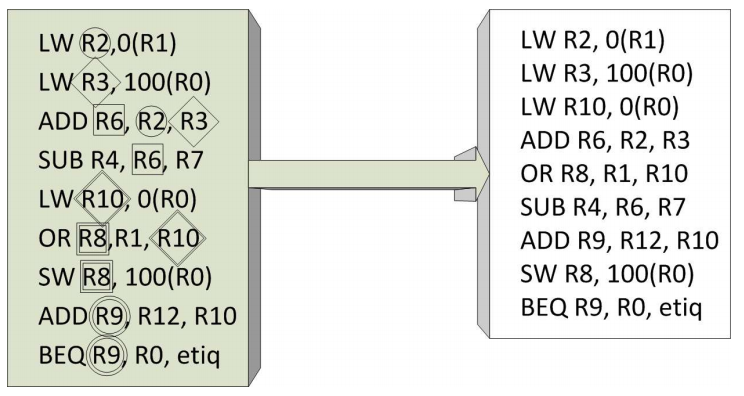
### Técnica de software

Evita los riesgos reordenando las instrucciones del código sin afectar los resultados.

Realizada por el compilador

* Introducción de instrucciones NOP (Se genera retardo)
* Reordenación de instrucciones
  + Máxima separación de instrucciones con dependencia RAW
  + Cuidado con ejecución “fuera de orden”

Reordenación por compilador:



## Soluciones a riesgos de control

Existe una penalización por salto.

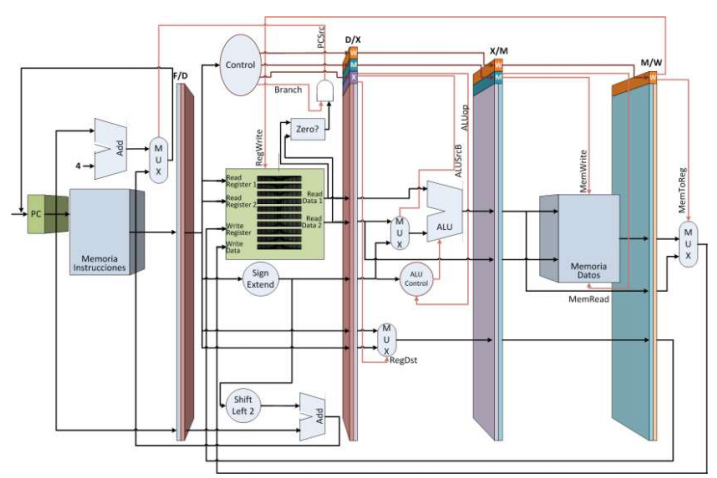
Instrucciones de salto:

* Incondicional: la dirección de destino se debe determinar lo más pronto posible, dentro del cauce, para reducir la penalización
* Condicional: Introduce riesgo adicional por la dependencia entre la condición de salto y el resultado de una instrucción previa

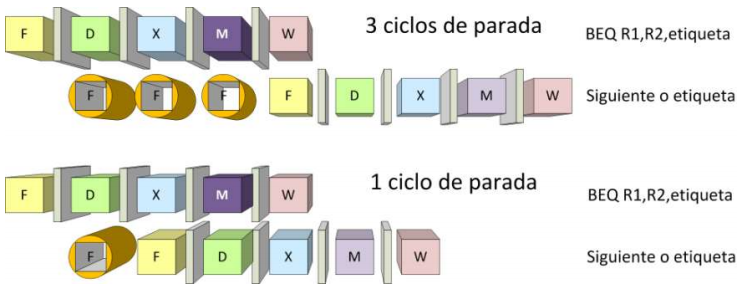
Modificación sencilla de la ruta de datos para reducir la cantidad de paradas a un solo ciclo

* Adelantar la resolución de los saltos a la etapa D
  + En ella se decodifica y se sabe que es un salto
  + Se puede evaluar la condición de salto (con restador)
  + Se puede calcular la condición de salto (con sumador)

Ruta de datos mejorada:



Reducción de paradas por mejora



Para tratamiento de saltos hay:

* Técnica hardware: predicción de saltos para evitar la parada
* Técnica software: salto retardado o de relleno de ranura de retardo (el compilador introduce instrucciones que se ejecutarán en cualquier caso después de la instrucción de salto)

### Predicción de saltos

* Técnicas estáticas
  + Predecir que nunca salta: asume que el salto no se producirá y siempre capta la próxima instrucción.



* + Predecir que siempre salta: asume que el salto se producirá y siempre capta la instrucción de destino del salto
* Técnicas dinámicas
  + Conmutador saltar/no saltar:
    - Basado en la historia de las instrucciones
    - Eficaz para los bucles
  + Tabla de historial de saltos (branch-target buffer)
    - Pequeña caché asociada a la etapa de búsqueda (F)
    - Tres campos:
      * Dirección de una instrucción de bifurcación
      * Información de la instrucción destino (dirección del destino o instrucción destino)
      * N bits de estado (historia de uso)

Diagrama de flujo de predicción saltar/no saltar:

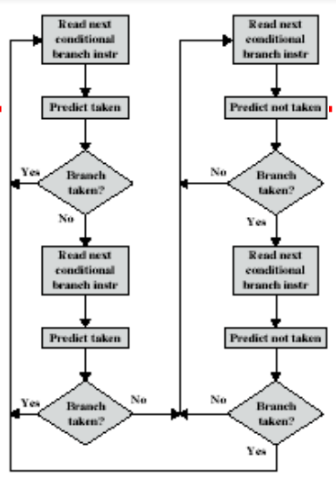
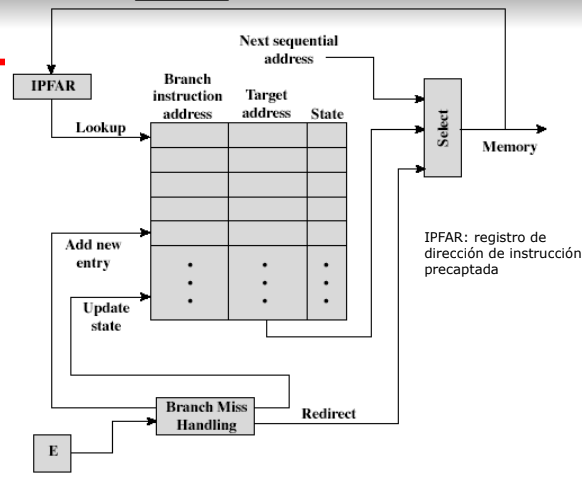


Tabla de historia de saltos (BTB)

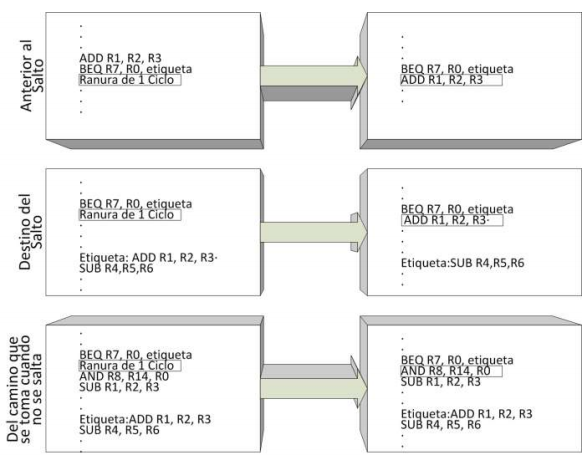


### Salto retardado

Idea: realizar trabajo útil mientras el salto se resuelve.

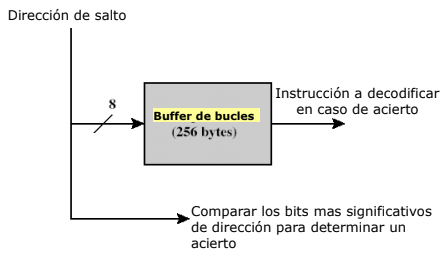
* Hueco o ranura de retardo de salto (delay-slot) es el período de penalización o parada luego de una instrucción de salto
* El compilador trata de situar instrucciones útiles (que no dependan del salto) en los huecos de retardo. Si no es posible, se utilizan instrucciones NOP{
* Las instrucciones en los huecos de retardo de salto se captan siempre
* Requiere reordenar las instrucciones

Alternativas para el salto retardado:



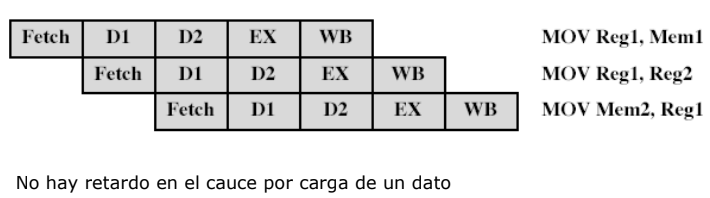
Otras soluciones hardware (a riesgos de control):

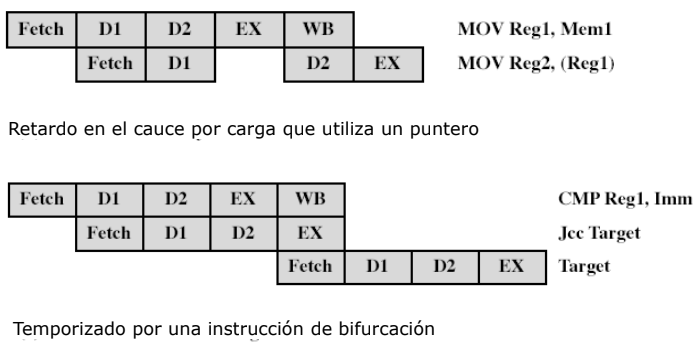
* Predecir segun el código de operación
  + Hay instrucciones con más probabilidad de saltar
  + La tasa de aciertos puede llegar a alcanzar un 75%
* Flujos múltiples
  + Varios cauces (uno por cada opción de salto)
  + Precaptan cada salto en diferentes cauces
  + Se debe utilizar el cauce correcto
  + Desventajas:
    - Provoca retardos en el acceso al bus y a los registros
    - Si hay múltiples saltos, se necesita un mayor número de cauces
* Precaptar el destino del salto
  + Se precapta la instrucción destino del salto, ademas de las instrucciones siguientes a la bifurcación
  + La instrucción se guarda hasta que se ejecute la instrucción de bifurcación
  + El IBM 360/91 usa ese método
* Buffer de bucles
  + Memoria muy rápida gestionada por la etapa de captación de instrucción del cauce
  + Comprueba el buffer antes de hacer la captación de memoria
  + Muy eficaz para pequeños bucles y saltos



## Segmentación en el 80486

* Tiene un cauce de 5 etapas: FI, D1, D2, EX y WB
* Ejemplos de funcionamiento del cauce:



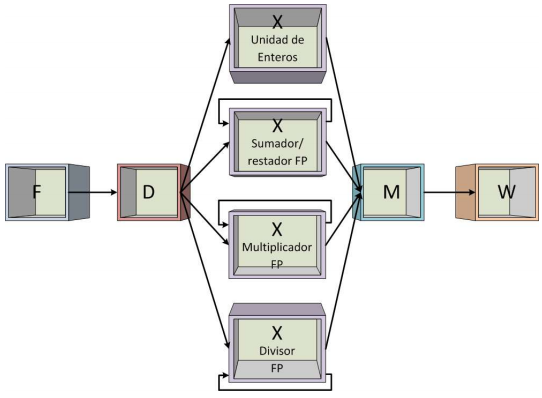


### Mejoras al repertorio

Quiero datos con representación en coma flotante. Necesito:

* Registros para su almacenamiento
* Hardware para las operaciones aritméticas: suma y resta, multiplicación, división

Etapa de ejecución multifuncional



Unidades de ejecución segmentadas:

