Segmentación de cauce

* ¿Qué es la segmentación de cauce? Explique los atascos producidos por saltos.
* En un cauce segmentado, con secuencia de instrucciones independientes ¿Qué consecuencias trae el paso de una instrucción de salto? Analice los casos de salto incondicional y condicional. Mencione que posibles soluciones se pueden aplicar para evitar o disminuir las consecuencias.
* ¿Qué entiende por segmentación de cauce? ¿Qué ventajas proporciona su implementación?
* Segmentación de cauce. Describir atascos por dependencia de datos y su solución.
* ¿Qué es la segmentación de cauce? Describir técnicas para el tratamiento de saltos condicionales
* ¿Qué ventajas nos brinda un cauce segmentado? Describa las diferentes formas que pueden mejorar el funcionamiento de un cauce cuando ejecuta instrucciones de transferencia de control.
* ¿Que es la segmentación del cauce de instrucciones? Describa los métodos y técnicas para disminuir o evitar las paradas (stalls) que afectan el funcionamiento de los cauces
* Describa tres (3) diferentes causas o motivos que pueden retardar un cauce de instrucciones segmentado
* Qué es la segmentación de cauce. Tres motivos de retardo de cauce.
* ¿Qué es la segmentación del cauce de instrucción? ¿Cuánto mejora el rendimiento? Describa las dependencias de los datos que puedan afectar un cauce segmentado
* ¿Qué es la segmentación del cauce? Describa tipos de dependencias que afectan el funcionamiento de los cauces y las posibles soluciones para evitarlos.

# ¿Qué es la segmentación de cauce?

La segmentación de cauce (pipelining) es una forma de organizar el hardware de la CPU para realizar más operaciones al mismo tiempo: Supone dividir el ciclo de instrucción en varias etapas/fases separadas que operan secuencialmente, de modo que las instrucciones se mueven a través de estas etapas como en una cadena de montaje. Cada etapa puede estar trabajando en una instrucción diferente al mismo tiempo (ejecución simultánea). Explota el paralelismo entre las instrucciones de un flujo secuencial.

Se llama así porque en un extremo se aceptan nuevas entradas antes de que algunas entradas aceptadas con anterioridad aparezcan como salidas en el otro extremo (como en un pipe o tubería).

Si consideramos el procesamiento de una instrucción como dos etapas independientes, captación y ejecución, habrá períodos en la ejecución de una instrucción en los que no se accede a la memoria principal. Este tiempo podría usarse en captar la siguiente instrucción en paralelo con la ejecución de la actual. Mientras que la segunda etapa ejecuta la instrucción, la primera etapa utiliza algún ciclo de memoria no usado para captar y almacenar la siguiente instrucción (lo llamamos prebúsqueda, precaptación de instrucción o solapamiento de la captación).

# ¿Qué ventajas proporciona su implementación?

El pipelining es una técnica que mejora las prestaciones a nivel de diseño del hardware, es invisible al programador.

Este proceso acelerará la ejecución de instrucciones. Si las etapas de captación y ejecución fueran de igual duración, el tiempo del ciclo de instrucción se reduciría a la mitad. Pero el tiempo de ejecución suele ser más largo que el de captación (implica lectura y almacenamiento de operandos además de alguna operación) y además una instrucción de bifurcación condicional hace que la dirección de la siguiente instrucción a captar sea desconocida, por lo que la etapa de captación deba esperar hasta recibir la dirección de la misma desde la etapa de ejecución. Para conseguir mayor aceleración, el cauce debe partir las instrucciones en más etapas de duración similar (uniformizar las etapas al tiempo de la más lenta).

El incremento potencial de la segmentación del cauce es proporcional al número de etapas del cauce. Se incrementa la productividad pero no reduce el tiempo de ejecución de la instrucción.

El diseño de procesadores segmentados tiene gran dependencia del repertorio de instrucciones.

# Atascos de un cauce (stall)

Problema: No todas las instrucciones necesitan todas las etapas, no todas las etapas pueden ser manejadas en paralelo y no se tienen en cuenta los saltos de control.

Los atascos son situaciones que impiden a la siguiente instrucción que se ejecute en el ciclo que le corresponde. Pueden ser:

* **Estructurales**: Provocados por conflictos por los recursos (dos o más instrucciones necesitan usar el mismo recurso de HW en el mismo ciclo)
* **Por dependencia de datos**: Dos instrucciones se comunican por medio de un dato que no está disponible cuando se necesita en una etapa determinada del cauce.
* **Por dependencia de control (o de instrucciones)**: La ejecución de una instrucción depende de cómo se ejecute otra. Una instrucción que modifica el valor del PC no lo ha hecho cuando se tiene que comenzar la siguiente. Existe una penalización por salto.

Los atascos se pueden solucionar con paradas del cauce, pero disminuye el rendimiento teórico ya que el CPI (Ciclos Por Instrucción)>1, mientras que el máximo rendimiento es completar una instrucción con cada ciclo de reloj.

## Atascos estructurales: soluciones

La solución es simple: Replicar, segmentar o realizar turnos para el acceso a las unidades funcionales en conflicto.

* **Duplicación de recursos 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 1ra mitad de los ciclos de reloj y lecturas en la 2da mitad de los ciclos de reloj

## Atascos por dependencia de datos

Tipos de dependencias de datos:

* Lectura después de Escritura (RAW, dependencia verdadera): Una instrucción genera un dato que lee otra posterior
* Escritura después de Escritura (WAW, dependencia en salida): Una instrucción escribe un dato después que otra posterior. Sólo se da si se deja que las instrucciones se adelanten unas a otras
* Escritura después de Lectura (WAR, antidependencia): Una instrucción modifica un valor antes de que otra anterior que lo tiene que leer, lo lea. No se puede dar en nuestro cauce simple.

**Soluciones 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.

Hay dos soluciones/técnicas:

* **Técnica Hardware (adelantamiento, forwarding o cortocircuito)**: Adelantamiento de operandos. 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).
  + Es fácil de implementar si se identifican todos los adelantamientos y se comunican a los registros de segmentación correspondientes.
* **Técnica Software**: Evita los riesgos *reordenando las instrucciones* del código sin afectar los resultados. Es realizada por el compilador:
  + Introducción de instrucciones NOP: Se genera retardo

*Reordenación de instrucciones*: Máxima separación de instrucciones con dependencia RAW. Cuidado con la ejecución “fuera de orden”.

## Atascos por dependencia de control (producidos por saltos).

Las instrucciones de salto pueden ser:

* **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

**Tratamiento**: 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 dirección de salto (con sumador)

### Salto condicional

Cuando hay una bifurcación condicional o salto, hasta que no se ejecuta la instrucción no hay forma de saber qué instrucción vendrá a continuación. Por lo general se carga la siguiente instrucción secuencialmente y continúa. Si el salto no se produce se obtiene el máximo provecho en rendimiento. Si el salto se produce, el cauce debe limpiarse de instrucciones inútiles y se presenta una penalización en las prestaciones sufrida por no haber podido predecir el salto.

Técnicas para el tratamiento de saltos condicionales:

* **Flujos múltiples**: Duplicar las partes iniciales del cauce y dejar que éste capte las dos instrucciones utilizando los dos caminos. EL problema de esto es que con cauces múltiples hay retardos por la competencia por el acceso a registros y memoria. Además, pueden entrar en el cauce (cualquiera de los dos flujos) instrucciones de bifurcación adicionales antes de que se resuelva la bifurcación original
* **Precaptar el destino del salto**: Se precapta la instrucción del salto además de la siguiente a la bifurcación. Se guarda esta instrucción hasta que se ejecute la instrucción de bifurcación. Si se produce el salto, el destino ya habrá sido precaptado.
* **Buffer de bucles**: Un buffer de bucles es una memoria pequeña de gran velocidad, gestionada por la etapa de captación de instrucción del cauce, que contiene, secuencialmente, las n instrucciones captadas más recientemente. Si se va a producir un salto, el hardware comprueba en primer lugar si el destino del salto está en el buffer. Tiene tres utilidades:
  + Con la precaptación, se anticipa almacenando algunas instrucciones que secuencialmente están después de la dirección de donde se capta la instrucción actual, y las instrucciones que se capten secuencialmente estarán disponibles sin el tiempo de acceso a memoria habitual
  + Si ocurre un salto a un destino a solo unas pocas posiciones más allá de la dirección de la instrucción de bifurcación, el destino ya estará en el buffer. Es útil para secuencias IF-THEN e IF-THEN-ELSE
  + Si hay un bucle y el buffer de bucles es lo suficientemente grande como para contener todas las instrucciones de un bucle, entonces esas instrucciones solo necesitan ser captadas de la memoria una vez, durante la primera iteración. En las siguientes, ya estará en el buffer
* Predicción de saltos (para evitar la parada): técnica hardware
  + Técnicas estáticas: no dependen de la historia de la ejecución
    - Predecir que nunca se salta: Asume que el salto no se producirá y siempre capta la siguiente instrucción
    - Predecir que siempre se salta: Asume que el salto se producirá y siempre capta la instrucción destino del salto
    - Predecir según el código de operación: El procesador asume que el salto se producirá para ciertos códigos de operación de bifurcación y no para otros
  + Técnicas dinámicas: dependen de la historia de la ejecución, intentan mejorar la exactitud de predicción
    - Conmutador saltar/no saltar: Basado en la historia de las instrucciones, eficaz para los bucles
    - Tabla de historia de saltos (branch-target buffer): Pequeña caché asociada a la etapa de captación de instrucción del cauce. Cada elemento de la tabla consta de tres campos:
      * Dirección de una instrucción de bifurcación
      * Información de la instrucción destino (dirección del destino o instrucción destino)
      * Número de bits de estado (historia de uso de la instrucción)
* Salto retardado (o de relleno de ranura de retardo): técnica software
  + Se pueden mejorar las prestaciones de un cauce reordenando automáticamente las instrucciones de un programa, de forma que las instrucciones de salto tengan lugar después de lo realmente deseado (el compilador introduce instrucciones que se ejecutarán en cualquier caso después de la instrucción de salto). (VER DIAPO 21 DE CLASE 5)

**Fuente**: “Clase 05” y “Capítulo 11.4 - Segmentación de instrucciones - Tratamiento de saltos ” (Stalling 5ta ed. Pág 411)

**Fuente**: “Clase 04”, “Clase 05” y “Capítulo 11.4 - Segmentación de instrucciones” (Stalling 5ta ed. Pág 405)