Tema 07: Temas avanzados de pipelining

Arquitectura de Computadoras

Ing. Nicolás Majorel Padilla (npadilla@herrera.unt.edu.ar)

http://microprocesadores.unt.edu.ar/arqcom/

Temas que veremos

- Manejo de Excepciones e Interrupciones.
- Superpipelining.
  - Impacto en la performance.
- Predicción de saltos.
  - Estática y Dinámica.

Lectura recomendada

- Computer Organization and Design, RISC-V Edition (2da ed, 2021)
  - Sección 4.9: *Control Hazards*
  - Sección 4.10: *Exceptions*
- Para los más curiosos:
  - Sección 4.12: *The Intel Core i7 6700 and ARM Cortex-A53.*

Recapitulación

- Diseñamos un procesador en pipeline.
  - CPI = 1, T pequeño. Genial.
- Pero tiene riesgos, que aumentan el CPI.
  - Los de control los minimizamos, pero no los evitamos.
    - No se pueden evitar, porque son inherentes a los saltos, que están siempre presentes en todos los ISA.
      - Un benchmark promedio posee un 16-20% de saltos.
    - CPI pasa a ser como mínimo de 1,2.
      - Considerando la penalidad de solamente un ciclo.

Repaso – Clasificación de Saltos

- Absolutos vs relativos.
  - La dirección destino se pasa completa o como un desplazamiento relativo a PC.
- Directos vs indirectos.
  - La dirección destino viene dada en la instrucción o está contenida en un registro.
- Condicionales vs incondicionales.
  - El salto depende de la evaluación de una condición.
- Tomados vs no tomados.
  - El salto condicional realmente se efectúa o no.

Repaso – Clasificación de Saltos

- En RV32I:
  - Todos los saltos condicionales (*branches*) son relativos.
  - Hay un solo salto absoluto, que también es indirecto.
    - Mediante la pseudoinstrucción **jr rs1** PC ← R[rs1]
    - Realmente es la instrucción jalr zero, rs, 0
  - Los llamados a subrutinas son saltos directos relativos.
    - Mediante la instrucción **jal rd, destino**
    - O mediante un par de instrucciones si el destino está más lejos.
  - Los retornos de subrutinas son saltos indirectos.
    - Mediante la pseudoinstrucción **ret** PC ← R[ra]

- Pero hay otros saltos, muy comunes, que no son necesariamente instrucciones…

Manejo de Excepciones e Interrupciones

- Los procesadores deben proveer soporte para manejo de excepciones e interrupciones.
  - **Preservar el comportamiento de las excepciones es una propiedad esencial para asegurar la exactitud de un programa.**
- Para un pipeline, las excepciones e interrupciones son riesgos de control.

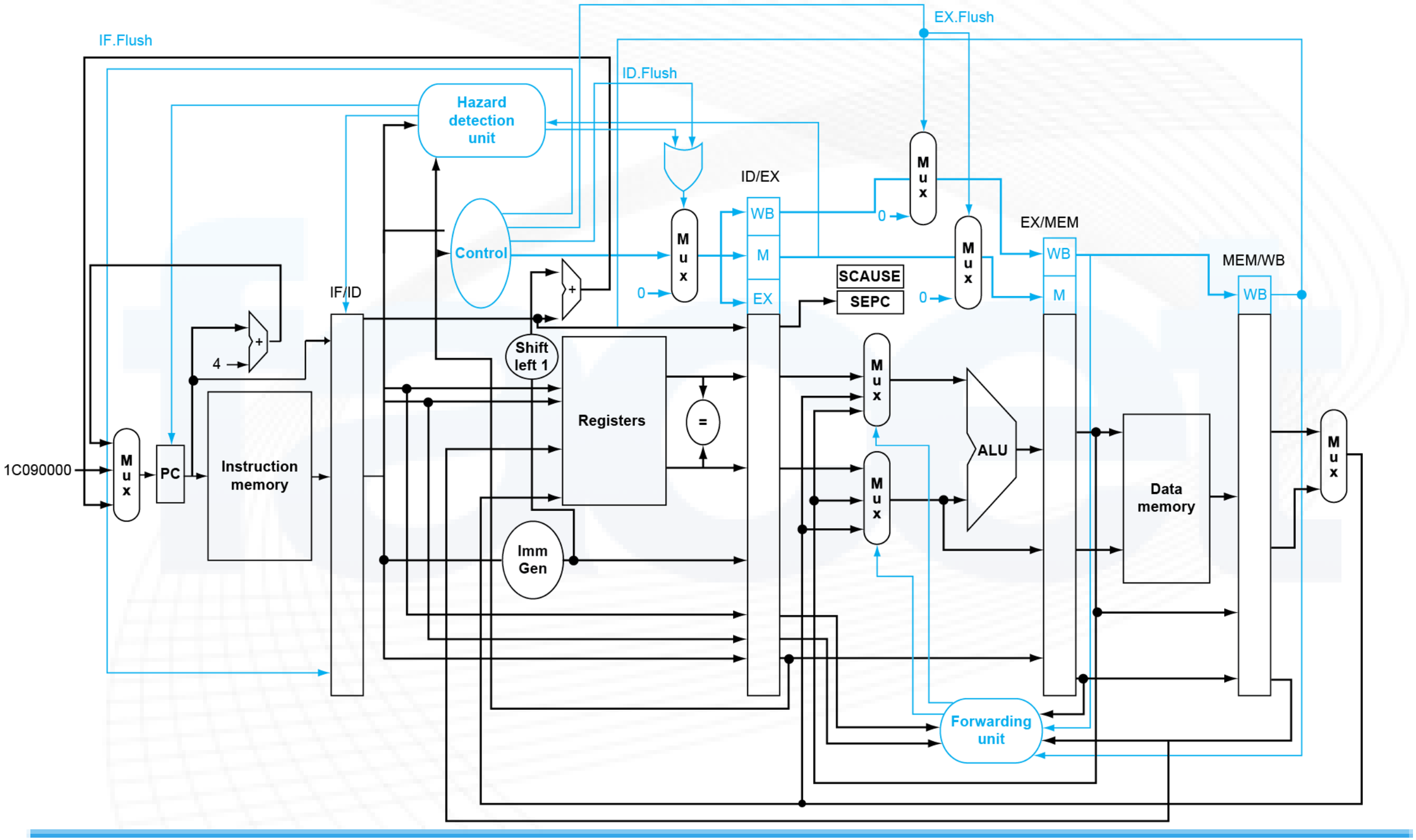
Manejo de Excepciones e Interrupciones

- Supongamos que una instrucción add que está en la etapa EX genera una excepción.
  - Se debe evitar que esa instrucción escriba en el banco de registros.
  - Se deben completar las instrucciones previas, que están en MEM y WB.
  - Se quita del pipeline la instrucción que provocó la excepción, y las siguientes (*flush*).
  - Se guarda el PC de la instrucción que causó la excepción, y la causa de la misma, y se transfiere el control al manejador de excepciones.
- Cuando se retorna del manejador de excepciones, la instrucción problemática vuelve a ser buscada en IF y ejecutada.

Manejo de Excepciones e Interrupciones

- Modificaciones al camino de datos:
  - Se debe guardar el PC de la instrucción interrumpida (o que causó la excepción) **en un registro adicional**.
  - Se debe guardar la causa de la interrupción (o excepción) **en otro registro adicional**.
  - Se debe cargar el PC con la dirección donde se ubica el manejador de interrupciones/excepciones.
    - Agregando una entrada adicional al multiplexor FuentePC.
  - Se debe agregar otro multiplexor para borrar las señales de control de la etapa EX.

Camino de datos para Interrupciones



Manejo de Excepciones e Interrupciones

- Como en un pipeline se ejecutan varias instrucciones en simultáneo, pueden ocurrir múltiples excepciones simultáneas.
- El enfoque más simple es manejar primero la excepción de la instrucción más antigua y vaciar las siguientes.
  - Eventualmente, las demás excepciones volverán a ocurrir y serán manejadas en su momento.
  - Esto se denomina **excepciones precisas**.

Superpipelining – Idea general

- Nuestro diseño tiene 5 etapas, y produce una aceleración igual a 4 (sin riesgos).
  - *¿Por qué la aceleración no es 5?*
- Sin embargo, recordando el Tema 3, podríamos dividir las etapas (la solución serial).
  - Dividiendo las etapas, se puede disminuir la duración del ciclo T.
    - P.ej: hagamos 8 etapas de 100 ps.
  - La teoría dice que aumentaríamos la performance.
    - Siempre y cuando el pipeline sea lo más cercano posible al ideal.

Longitudes típicas de pipelines modernos

- Los pipelines de procesadores modernos de alta performance suelen tener más de 5 etapas.
- Dependen del tipo de procesador.
  - Procesadores pequeños, optimizados en área, suelen tener entre 1-3 etapas.
  - Procesadores medianos suelen tener entre 5-10 etapas.
  - Procesadores de alta performance suelen tener entre 10-20 etapas.
    - AMD Ryzen 9 7900X3D (2023): 19 etapas.
    - ARM Cortex X4 (2023): 13 etapas.
    - Apple M4: 15 etapas
    - Intel Core i7-13700K (2022): entre 16 y 19 etapas.
    - SiFive P870 (2024): entre 10 y 18 etapas.
- El procesador con más etapas que hubo fue un Pentium 4 (2004), con 31 etapas.

Superpipelining – Inconvenientes

- **Validez del modelo:** se debe poder dividir una etapa.
  - No podemos hacer media operación en la ALU, o media búsqueda en memoria.
  - Los retardos de los buffers intermedios pasan a ser significativos.
- La disminución de la duración del ciclo equivale a un aumento de la frecuencia de reloj, lo que implica un **aumento del consumo de energía**.
- **Los riesgos se incrementan:**
  - Mayor posibilidad de riesgos estructurales.
  - La unidad de adelantamiento se puede volver muy compleja.
  - El riesgo adicional del LW puede requerir más de una burbuja.
  - La penalidad por no acertar la predicción de saltos puede volverse muy grande.
- El último inconveniente es el más difícil de solucionar.
  - Porque los riesgos de control son inevitables, y los saltos son muchos.

Impacto en la Performance

- Supongamos un benchmark con un 20% de saltos, que se ejecuta en un procesador con CPI = 1.1 (considerando riesgos de datos), y que tiene una penalidad por fallo en la predicción de saltos de 8 ciclos.
- Si la tasa de acierto de la predicción de saltos es del 75%, ¿cuánto sería el CPI incluyendo riesgos de control?

- $CPI = CPI_{ideal}$ + paradas por fallos
- Paradas por fallos = %saltos * %fallos_pred * penalidad
- CPI = 1.1 + 0.2 * 0.25 * 8 = 1.5
- Se pierde casi un 40% de performance.

Predicción de Saltos Estática

- Primer intento: predecir que nunca saltan (o que siempre saltan).
  - Se busca (y ejecuta) siempre la instrucción siguiente al salto.
  - Si se acierta la predicción, la penalidad es cero.
  - Si no se acierta la predicción, la penalidad es la misma que antes (sin predicción).
- Ventajas:
  - Muy simple y sencilla.
- Tasa de acierto: aprox 70%.
- *¿Realmente será igual la tasa de acierto en caso de predecir en ambos casos?*

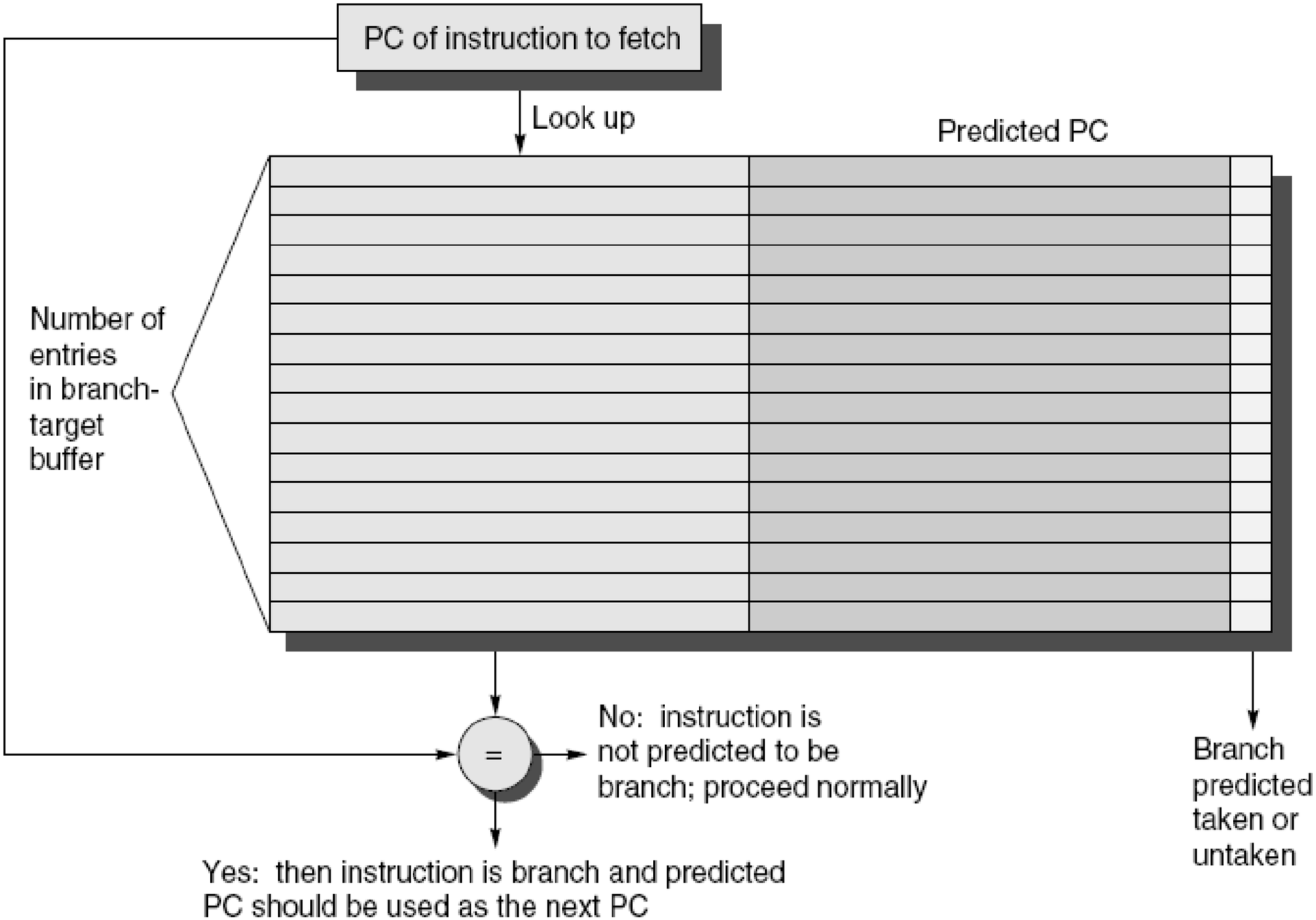
Predicción de Saltos Estática

- Segundo intento: BTFNT (***B**ackwards **T**aken, **F**orwards **N**ot **T**aken*).
  - Basada en el comportamiento típico de los saltos.
    - Los saltos hacia atrás son casi siempre tomados (lazos).
    - Los saltos hacia adelante son casi siempre no tomados (sentencias if).
- Tasa de acierto: aprox 80%.
- Prácticamente no se usa más ninguna.
  - Porque las tasas de acierto son inaceptablemente bajas actualmente.

Predicción Dinámica de saltos

- La predicción cambia en función del historial del salto.
- Se basa en mantener una historia de cada salto, y hacer la predicción en base a ella, usando dos estructuras.
- Se mantiene una **Tabla de Historia de Saltos (*Branch History Table*, BHT)**, que guarda:
  - La dirección origen del salto (para identificarlo).
  - Bits para saber su historial (para predecirlo).
- Se mantiene también una memoria que almacena **las direcciones destino de los saltos (*Branch Target Buffer*, BTB)**.
  - Para no calcularla nuevamente.

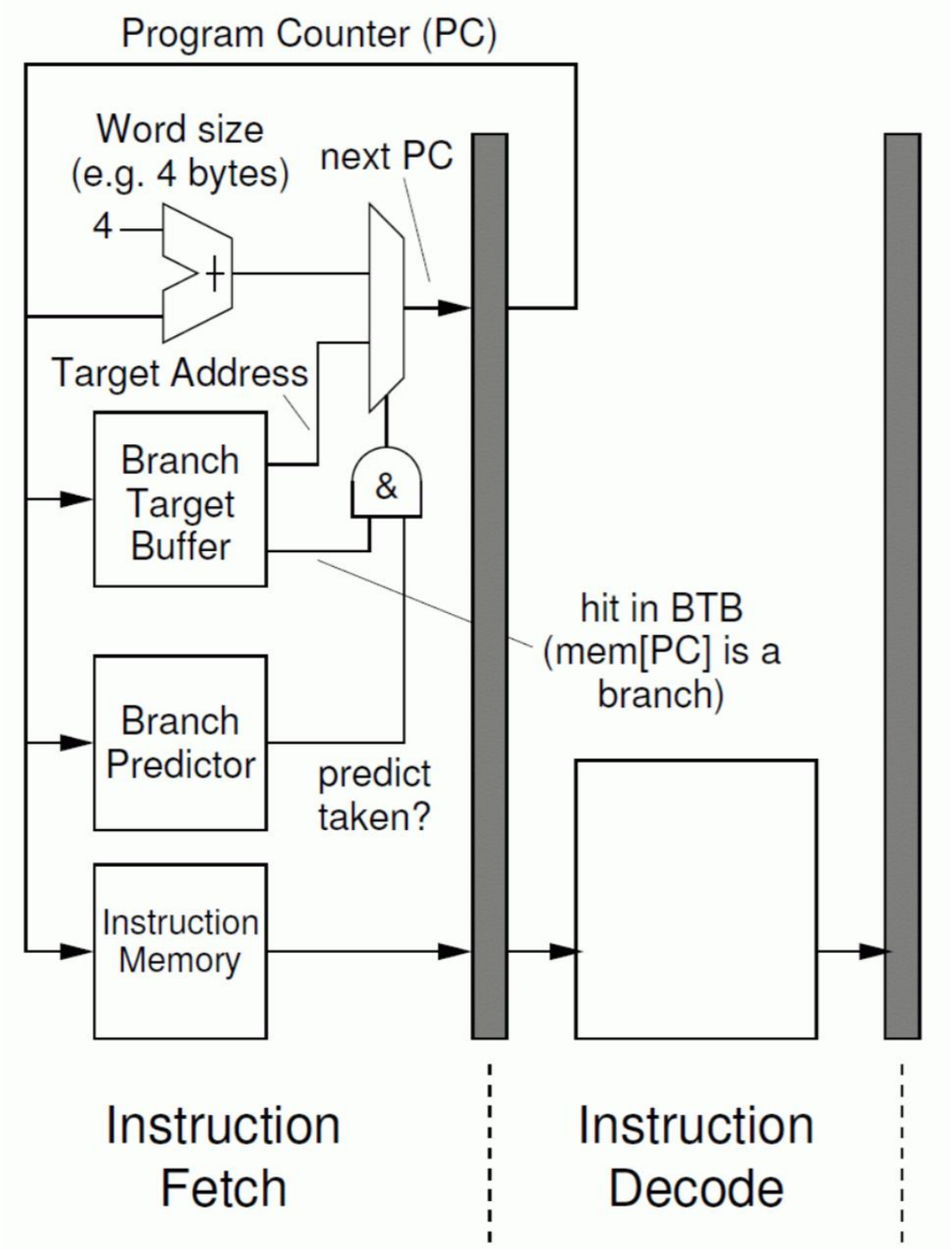
Predicción Dinámica de saltos



Predicción Dinámica de saltos

- Al buscar una instrucción, se revisa la BHT para saber si es un salto.
  - Si lo es, se espera la misma salida de la última vez, y se prepara la dirección destino (del BTB o PC+4).
  - Si no se acierta la predicción, se vacía el pipeline y se actualizan los bits del historial.

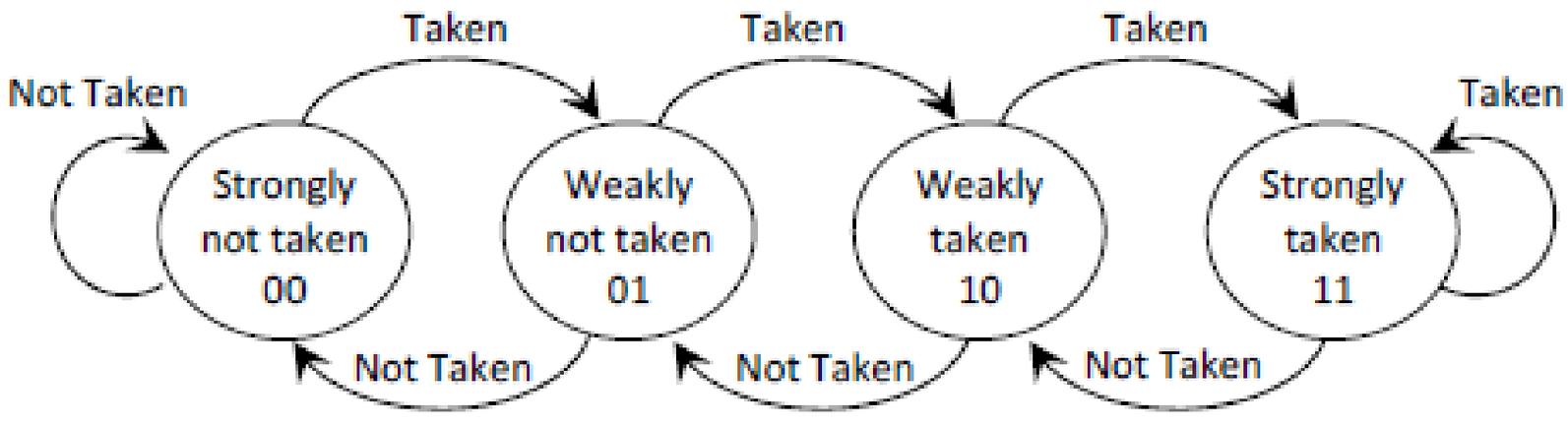
Predicción Dinámica de saltos



- Nótese que con BHT y BTB, **toda la resolución del salto se puede hacer en la etapa 1 (IF)**.
  - ¡En el tema anterior dijimos que era ilógico!
- Si la predicción es correcta, no se inserta ninguna burbuja en el pipeline.
  - Se denomina *Zero-bubble Branch Prediction*.

Predicción Dinámica de saltos

- *¿Cuántos bits se usan para mantener la historia?*
  - 1 bit: lo más simple (repite lo que pasó la última vez).
    - Tasa de acierto: aprox 85%.
  - 2 bits: sólo se cambia la predicción al no acertar dos veces consecutivas.
    - Usa una MEF simple de cuatro estados, que implemente un contador saturado.



  - Tasa de acierto: aprox 90%.

Predicción Dinámica de saltos

- Usar más bits para guardar la historia del salto no resulta conveniente:
  - Genera un mayor aumento de costo que el aumento en la mejora de la tasa de acierto.
  - Sólo algunos saltos necesitan mayor cantidad de bits.
    - Los que no se relacionan con eventos recientes.
- Una propuesta es usar no una, sino **múltiples tablas de historial de saltos**.
  - Cada una con diferente cantidad de bits de historial.
    - La cantidad de bits de cada tabla crece de manera exponencial (o geométrica).
  - Se busca en todas las tablas en paralelo, y gana aquella que resulte tener el historial más apropiado para ese salto.
- Algoritmo TAGE (*Tagged Geometric*).
  - Tasa de acierto: aprox 98%.

Predicción Dinámica de saltos

- Para mejorar aún más la tasa de acierto del algoritmo anterior, se le agrega un corrector estadístico:
  - Se mantiene una estructura basada en contadores de diversos patrones.
  - Si la suma de los valores de todos esos contadores supera un umbral, se cambia la predicción del TAGE.
- Para mejorar aún más la tasa de acierto, se agrega también un predictor de lazos:
  - Diseñado para lazos *for*, que son tomados n-1 veces y no tomados exactamente una vez.
  - Mantiene un contador de iteraciones actuales, y un registro de máximas iteraciones anteriores.
  - Cuando el valor actual iguala al máximo, lanza una predicción más prioritaria que la del historial.
- Esta es la idea del **algoritmo TAGE-SC-L**.
  - Tasa de acierto: aprox 99.2%.
  - Usado en las versiones más recientes de procesadores de Intel y AMD.

Predicción Dinámica de saltos

- Aún hoy, es un tópico muy investigado.
  - Se prueban múltiples algoritmos de predicción como variantes de TAGE.
    - ¡Algunos diseños usan redes neuronales!
  - Se prueban múltiple cantidad de bits para las longitudes de historial de TAGE.
- Se buscan tasas de predicción mayores que 99%, para todos los saltos.
- Se prefiere continuar invirtiendo en Hw, en vez de pagar la penalidad.
  - Impulsados por la Ley de Moore.

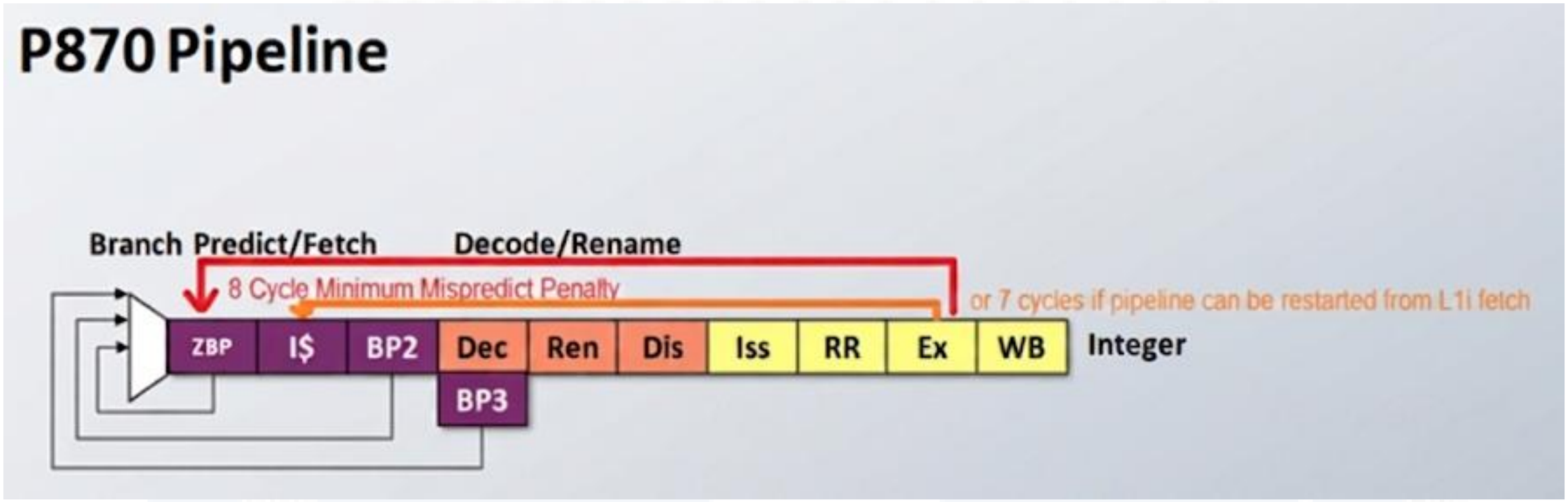
Return Address Stack

- Problema adicional: los saltos indirectos suelen ser muy difíciles de predecir.
  - Porque la dirección del salto siempre es la misma, pero el destino no.
- Lamentablemente, son muy utilizados.
  - Son los retornos de subrutinas.
- Como en general el comportamiento de las subrutinas es de tipo LIFO, los microprocesadores modernos incorporan un buffer especial para manejar estos casos
  - Denominada **Pila de Direcciones de retorno** (*Return Address Stack*, RAS).
  - Un buffer pequeño y circular, de 32 entradas aproximadamente.
    - Los M3 de Apple tienen 48, y la arquitectura Zen 5 de AMD tienen 52.
- Si se detecta que el salto es un llamado a subrutinas, se guarda automáticamente la dirección de retorno en el RAS.
- Al detectar el salto del retorno, la dirección del salto se obtiene del RAS y no del BTB.

Branch Target Instruction Cache

- El BTB almacena las direcciones destino de los saltos.
  - Para no calcularlas nuevamente.
  - Con esas direcciones se accede a la memoria para hacer el Fetch correspondiente.
- Algunos procesadores modernos incorporan una estructura adicional, denominada *Branch Target Instruction Cache (BTIC)*.
  - Almacena directamente el contenido de la memoria apuntado por la dirección del salto.
    - O sea, la instrucción que se buscaría en el Fetch.
  - Suele tener 16 entradas.
    - Al ser muy pequeño, es mucho más rápido que la memoria de instrucciones.
- Puede ayudar a mejorar la performance en algunos pipelines, con un costo casi insignificante.

Pipeline de un procesador moderno



- Este pipeline es solamente para instrucciones Tipo R y saltos.
  - No contempla lw y sw.
- Posee 10 etapas:
  - 3 para Fetch y predicción de saltos.
  - 3 para Decodificación y Renombramiento de registros.
  - 4 para la Ejecución y Write-Back.
- Tiene una penalidad por fallo en la predicción de saltos de 8 ciclos.

Resumen final

- Las excepciones e interrupciones son riesgos de control que deben ser manejados.
- Los procesadores deben soportar el manejo de excepciones precisas.
- Mejora de performance a través de superpipelining.
  - Aumentar la cantidad de etapas dividiéndolas, para disminuir T.
  - Problemas: validez, consumo de energía y penalidad para los riesgos de control.
- Para no pagar esa penalidad, se usa Predicción Dinámica de Saltos.
  - Basada en una BHT y un BTB.
  - Si se acierta la predicción, no se paga penalidad.
- Búsqueda constante de mejorar la tasa de acierto:
  - Uso de múltiples tablas, con distintos historiales (TAGE).
  - Estructuras adicionales para manejar casos específicos (lazos, retornos).