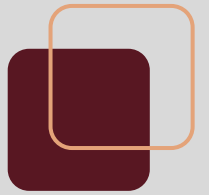
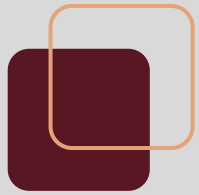


Aspectos avanzados de arquitectura de computadoras
Predicción de Saltos

Facultad de Ingeniería - Universidad de la República

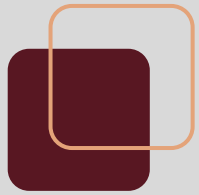
Curso 2017

Introducción



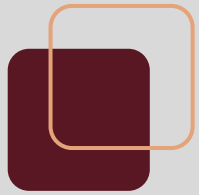
- La ejecución contínua de instrucciones en un pipeline exige la mitigación de *hazards*, en particular los de control, generados por dependencias de control.
- La predicción de saltos es la técnica por excelencia utilizada hoy en día para mitigar los hazards de control.

Técnicas ya estudiadas



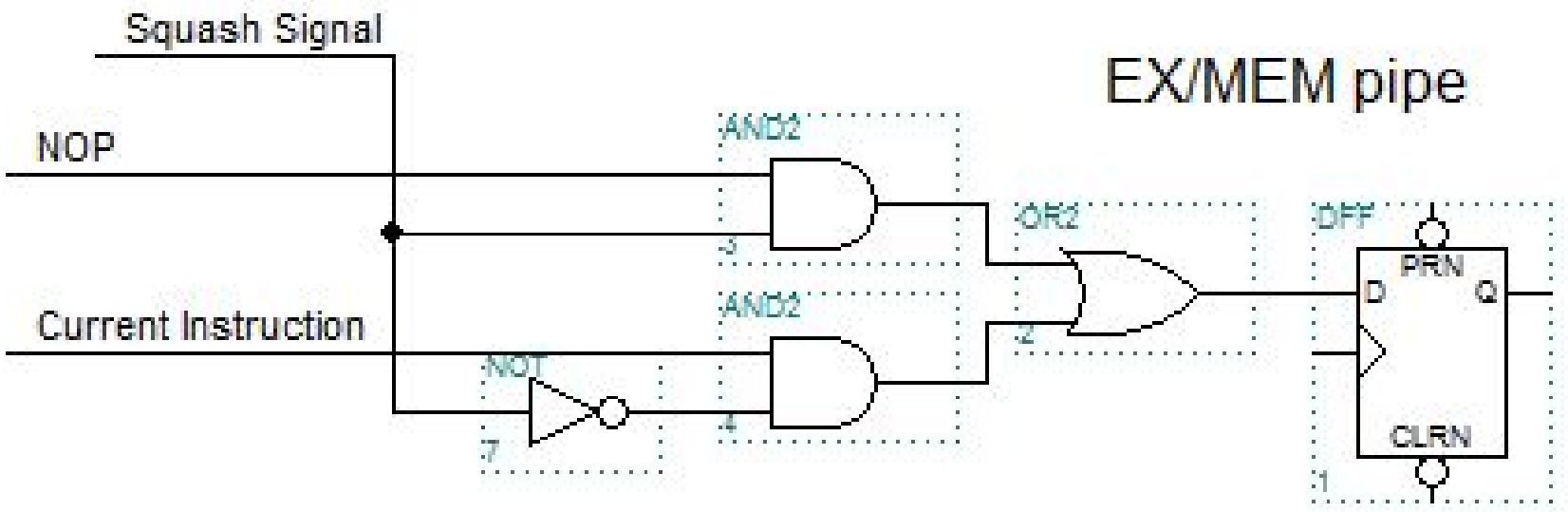
- Predictores:
  - Predecir que nunca se toma el salto (always not taken)
  - Predecir que siempre se toma el salto (always taken)
  - Predicción basada en el código de operación
  - Conmutar taken/not taken (predictor de dos bits)

Predecir Always Not Taken



- Se asume que el salto nunca se toma, por lo cual siempre se comienza a ejecutar la siguiente instrucción al salto.
- Hardware adicional para el caso en que sí se tome el salto:
  - Se deben convertir las instrucciones cargadas en NOPs!

Squash Signal

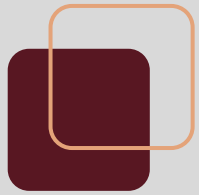


- Squash Signal =

  (OPCode_EX == JMP) ||

  (OPCode_EX == Branch && Branch_Taken)

Predecir Always Taken

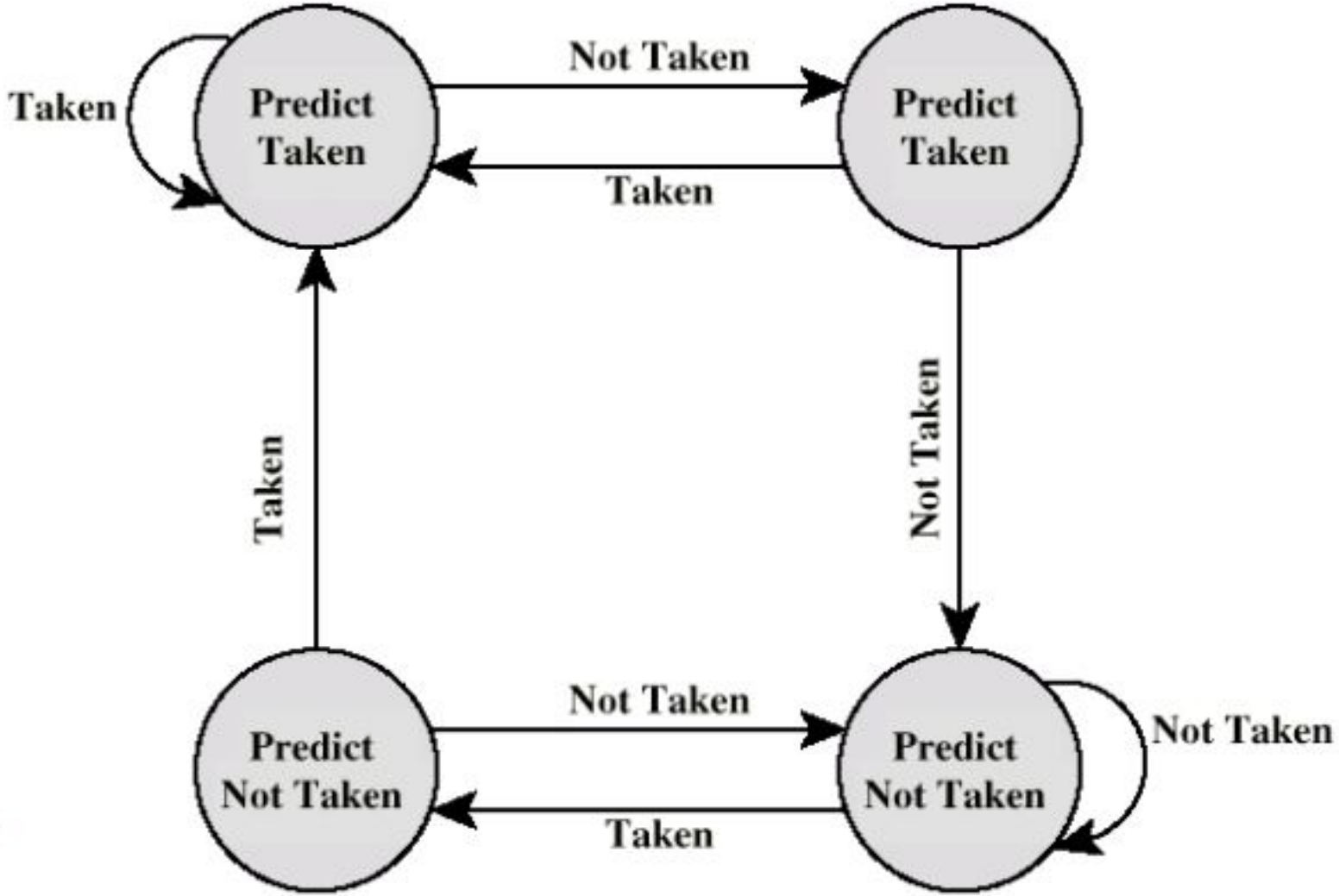


- Se asume que el salto siempre se toma, por lo cual siempre se comienza a ejecutar.
  - A menos que la decodificación se realice en la etapa IF, se pierden ciclos aún con 100% de accuracy.
- Squash Signal =

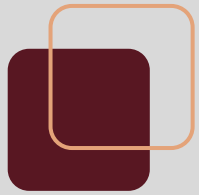
  (OPCode_EX == JMP) ||

  (OPCode_EX == Branch && Branch_Taken)

Conmutar Taken / Not Taken

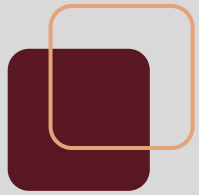


Predicción Basada en OpCode



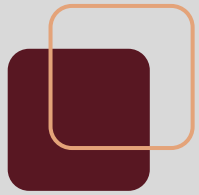
- Consiste en mantener diferentes tipos de predicción dado el código de operación de la instrucción que se está ejecutando.
- EJ:
  - BEQ -> Always Taken
  - BNZ -> Predictor de dos bits

Branch History Table (BHT)



- Consiste en mantener una pequeña memoria en el CPU, la cual se indiza con la dirección de la instrucción. Cada entrada mantiene un predictor en particular.
- Mejora las predicciones ya que disminuye la probabilidad de que dos saltos interfieran en las predicciones.

Predictores correlativos (1/3)

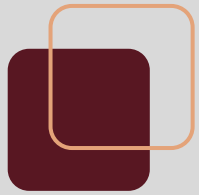


- Las técnicas utilizadas hasta el momento solo utilizan la información del propio salto para decidir la predicción.

```
if (aa==2)
    aa=0;
if (bb==2)
    bb=0;
if (aa!=bb) { … }
```

- En el código anterior, el tercer salto depende de los dos anteriores!

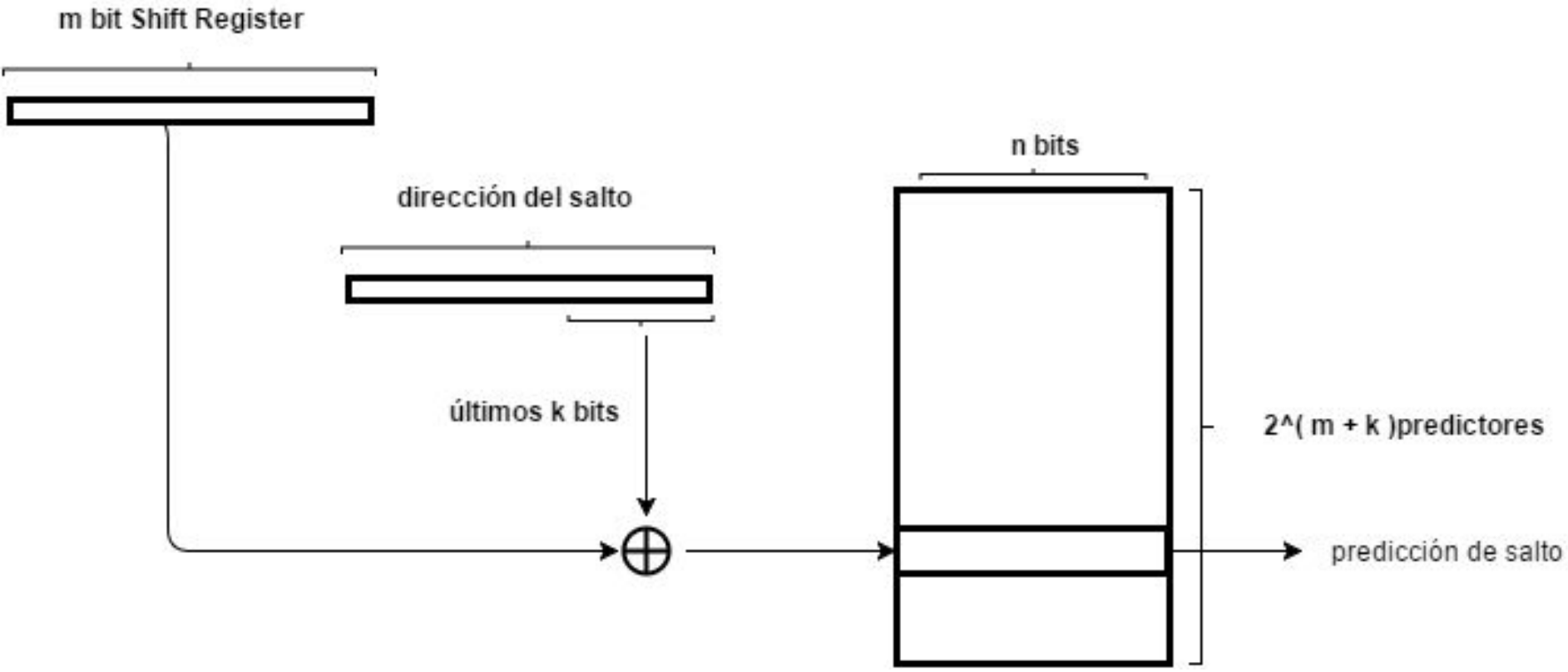
Predictores correlativos (2/3)



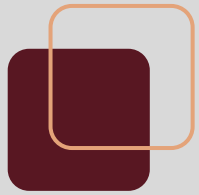
- Los predictores que utilizan la información de saltos anteriores se llaman predictores correlativos (o predictores de dos niveles).
- Un predictor de dos niveles *(m,n)*, toma la información de los m saltos anteriores para decidir entre 2^m predictores de n bits.

Predictores correlativos (3/3)

- Implementación

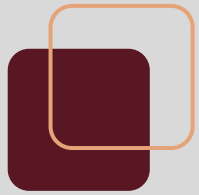


Predictores de Torneo



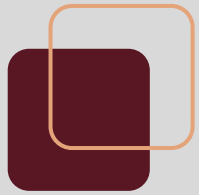
- Combinan predictores globals (2-level) con locales (BHT).
- Se mantienen ambos predictores y se mantiene un contador de dos bits para indicar cuál predictor se utilizará, según cuál haya sido el más efectivo en las últimas predicciones.

Branch-Target Buffer (1/2)



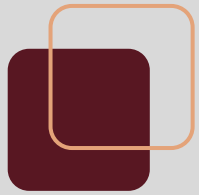
- Las técnicas de predicción de saltos utilizadas hasta el momento no logran eliminar por completo las demoras aún con una predicción 100% correcta, porque antes de aplicarlas se debe decodificar la instrucción, perdiendo un ciclo en el proceso para el caso del pipeline MIPS de 5 etapas.
- Más ciclos perdidos si la dirección de salto es calculada!

Branch-Target Buffer (2/2)



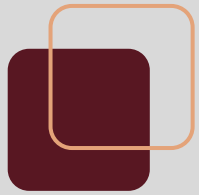
- Un *Branch-Target Buffer* es una memoria *tipo* caché utilizada en predicción de saltos.
- Guarda la dirección destino del los saltos ejecutados.
- Al cargar una instrucción, se busca la misma en el BTB, si está, es un salto y ya se tiene la dirección del salto en caso de que se prediga como tomado.

Branch Folding



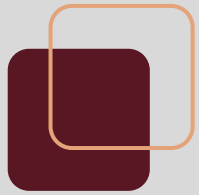
- Una modificación muy interesante al BTB es guardar la *instrucción destino* del salto en lugar de su dirección.
  - Al tomar encontrarse una instrucción en el BTB, se sustituye la instrucción del salto con la instrucción destino.
- Esta técnica permite ejecutar saltos (generalmente incondicionales) en 0 ciclos!

Return Address Stack (1/2)



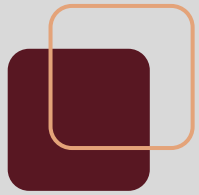
- Además de saltos, las instrucciones RET generan dependencias de control particularmente interesantes, ya que la dirección destino varía en tiempo de ejecución.
- Un *return address stack (RAS)*, es un stack de hardware que guarda las direcciones de retorno de las llamadas a procedimientos ejecutadas.

Return Address Stack (2/2)



- ¿Cómo funciona?
  - Al ejecutar un CALL, se realiza un push de la dirección de retorno.
  - Al decodificar un RET, se obtiene la siguiente dirección a buscar haciendo un POP del RAS.
  - Se ganan ciclos de reloj al no tener que realizar el acceso a memoria.

Fin



¿Preguntas?