Taller de Microarquitectura de procesadores

Organización del Computador II

9 de Diciembre de 2014

Menú del día

- **Repaso de Memoria Caché**
  - Clasificación de caché
  - Direccionamiento
  - Niveles de caché
  - Arquitectura de un procesador

- **Repaso de Pipelines**
  - Etapas de un pipeline
  - Trabas en el pipeline
  - Especulación

Repaso de Memoria Caché

Recordatorio inicial:

- **Memoria Caché**: Dicese de las memorias que están entre la memoria principal y el procesador.
- **Principio de localidad espacial**: Si se accede a un dato, es muy probable que se acceda a los bytes vecinos.
- **Principio de localidad temporal**: Si se accede a un dato, es muy probable que se acceda a el nuevamente.

Las memorias caché son las memorias a las que normalmente accede un procesador, salvo que explicitamente se le indique lo contrario. Son claves para tapar el hecho de que a los procesadores cada vez les cuesta mas ciclos de clock acceder a la memoria principal.

Caché: Clasificación

Las memorias caché se clasifican principalmente por su mecanismo de acceso para lectura y escritura de los datos.

- **Correspondencia directa**: Cada dirección de la memoria principal puede corresponder a un único lugar en la memoria caché.
- **Asociativa por conjuntos**: Cada dirección de la memoria principal puede corresponder a una reducida cantidad de lugares en la memoria caché.
- **Asociativa completa**: Cada dirección de la memoria principal puede puede corresponder a cualquier dirección en la memoria caché.

Caché: Clasificación

Podemos a clasificar las memorias caché por su uso también.

- **Caché de instrucciones**: Los bytes que el procesador cree que son las próximas instrucciones se van a traer aquí.
- **Caché de datos**: Los bytes que están siendo operados por el procesador corresponderán a datos y no a instrucciones.

Caché: Clasificación

Podemos a clasificar las memorias caché por su uso también.

- **Caché de instrucciones**: Los bytes que el procesador cree que son las próximas instrucciones se van a traer aquí.
- **Caché de datos**: Los bytes que están siendo operados por el procesador corresponderán a datos y no a instrucciones.

Y también por los datos que contienen.

- **Inclusivas**: Los niveles superiores de caché contienen todos los datos que tienen los niveles inferiores.
- **Exclusivas**: Cada nivel de caché contiene solamente sus datos y no lo de niveles inferiores.

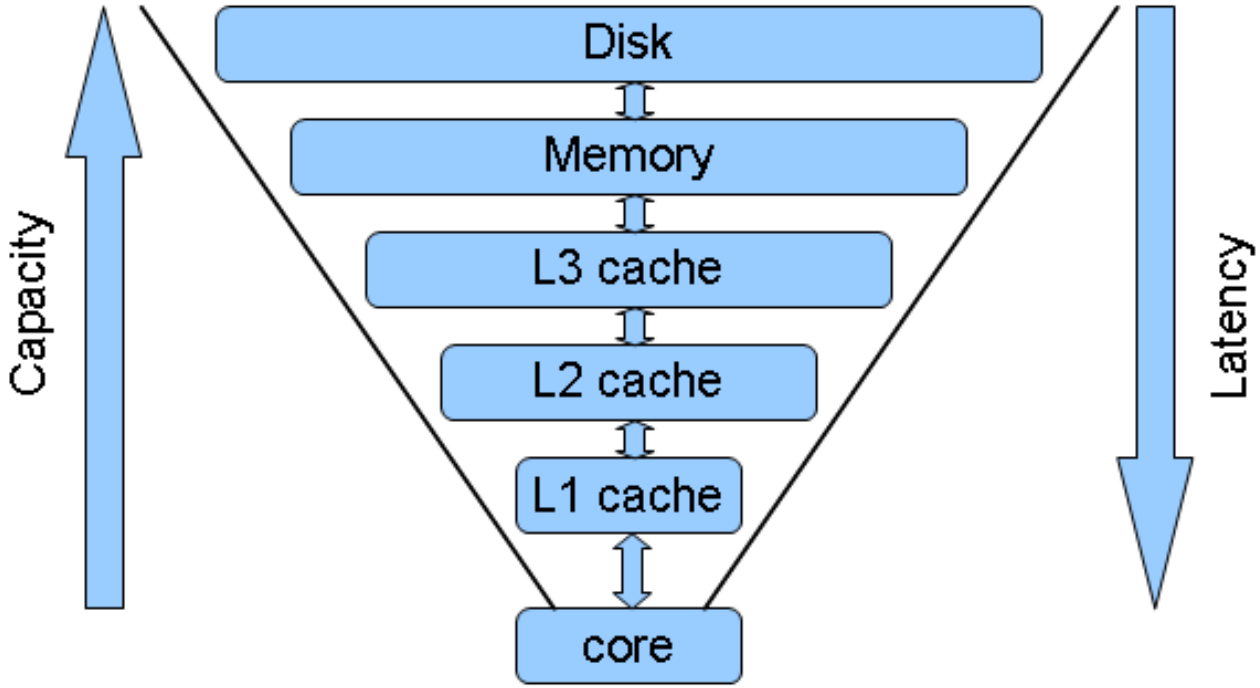
Caché: Direccionamiento

Las direcciones en la memoria caché se separan en índices, lineas y tags.

- **Tag**: Campo lógico de una dirección de memoria, que representa un identificador único de la dirección de memoria a la que corresponde.
- **Line**: Campo lógico de una dirección de memoria, que indica en que posición de la caché se va a guardar el dato que traigamos.
- **Index**: Campo lógico de una dirección de memoria, que corresponde al dato dentro de la línea que queremos recuperar o asignar.

Esto nos va a permitir separar las direcciones en distintos segmentos que vamos a poder usar para indexar dentro de las memorias caché.

Caché: Jerarquía

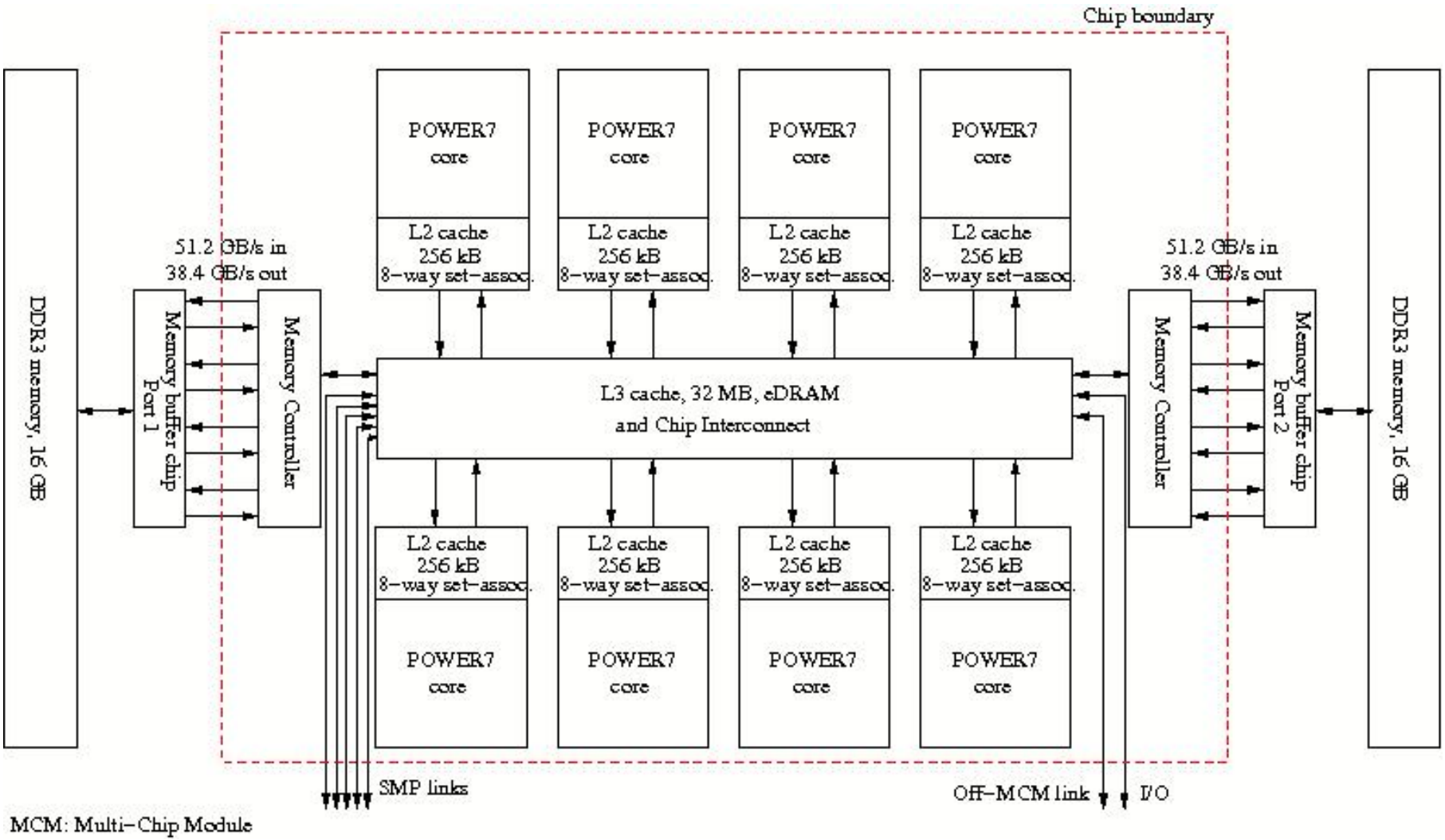


La idea de la jerarquía es usar tecnologías más eficientes solo las memorias de más intensidad de uso y usar tecnologías más rentables para las memorias de mayor capacidad.

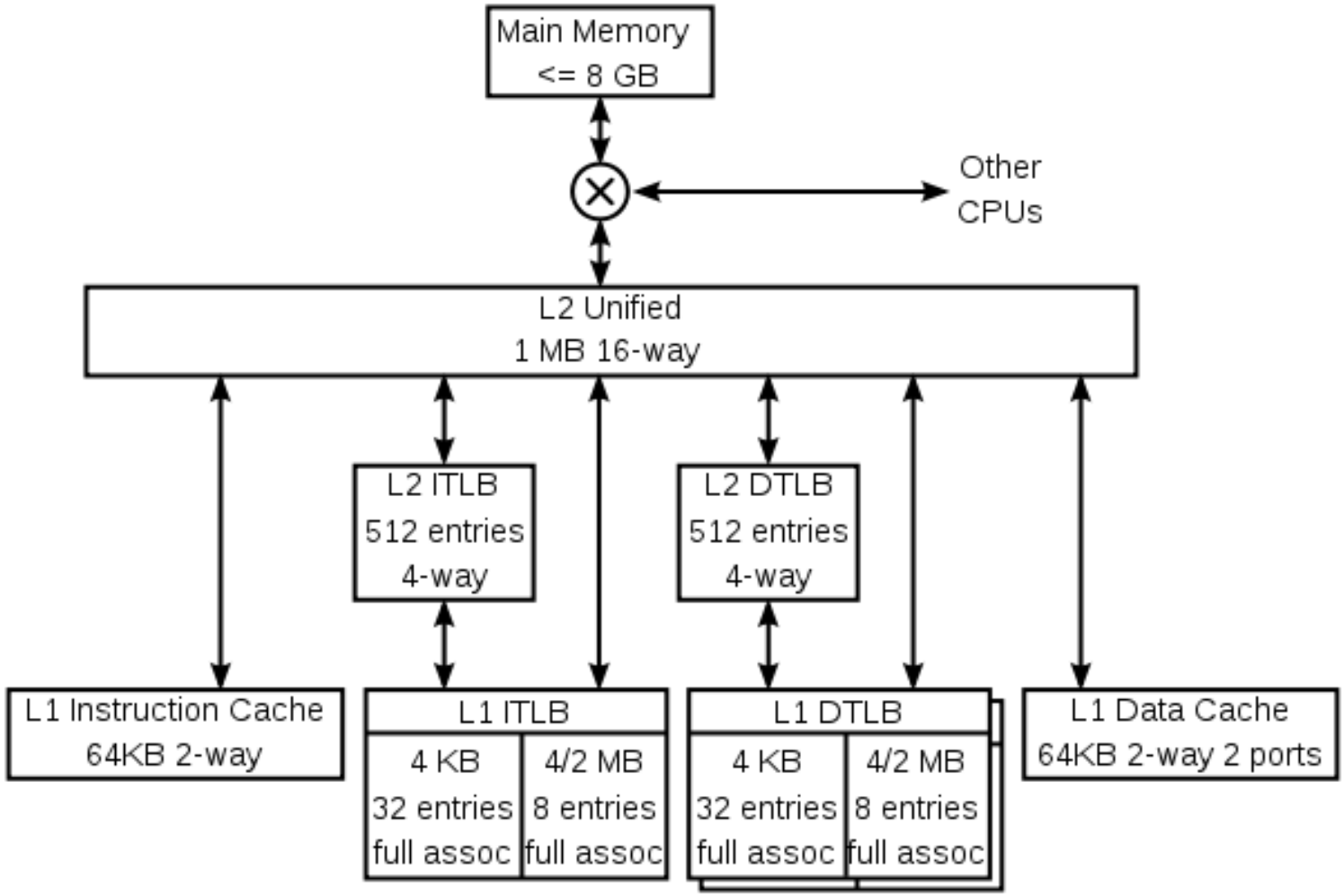
Caché: Arquitectura de un procesador

El acceso a las memorias de cada modelo de procesador se diseña tomando en cuenta las características que nos gustaría que tuviera y considerando las restricciones de costo y superficie.
Cada procesador soluciona el enlace a las memorias de maneras muy distintas. Algunos ejemplos:

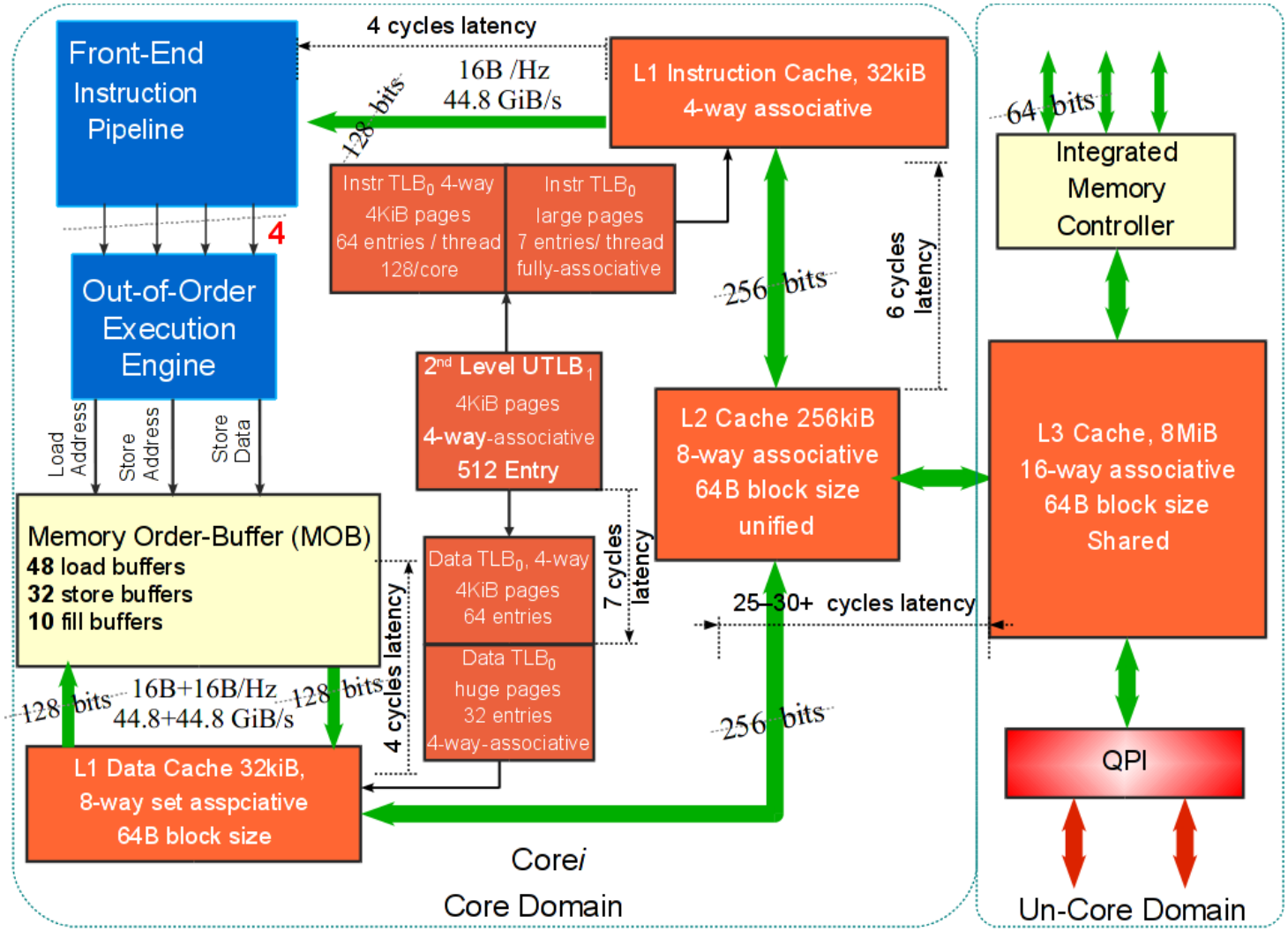
Caché: IBM - POWER7



Caché: AMD K8 - Athlon 64



Caché: Intel Nehalem - Core i7



Repaso de Pipelines

Recordatorio inicial:

Pipeline: Dicese de todo mecanismo que involucre realizar una tarea en pasos pequeños y bien definidos. En nuestro contexto los vamos a pensar como una herramienta que incluye un procesador para poder solucionar el problema de ejecutar una instrucción (y todo lo que ello conlleva) de forma paralela.

El pipeline es la principal herramienta para combatir la latencia de la memoria. Pero usar un pipeline puede traer conflictos nuevos.

Pipeline de instrucciones: Etapas

Los pasos que componen un pipeline los vamos a llamar *Etapas*.

Estas etapas engloban las 5 grandes tareas que componen una instrucción:

1. **Fetch**: Se trae de memoria la instrucción a ejecutar.
2. **Decode**: Se decodifica que instrucción va a ser ejecutada.
3. **Execute**: Se hace efectivamente la cuenta.
4. **Memory**: Se escribe el resultado en memoria, si es necesario.
5. **Writeback**: Se guardan en los registros del procesador los resultados.

A esto se lo conoce como el *clásico pipeline RISC*.

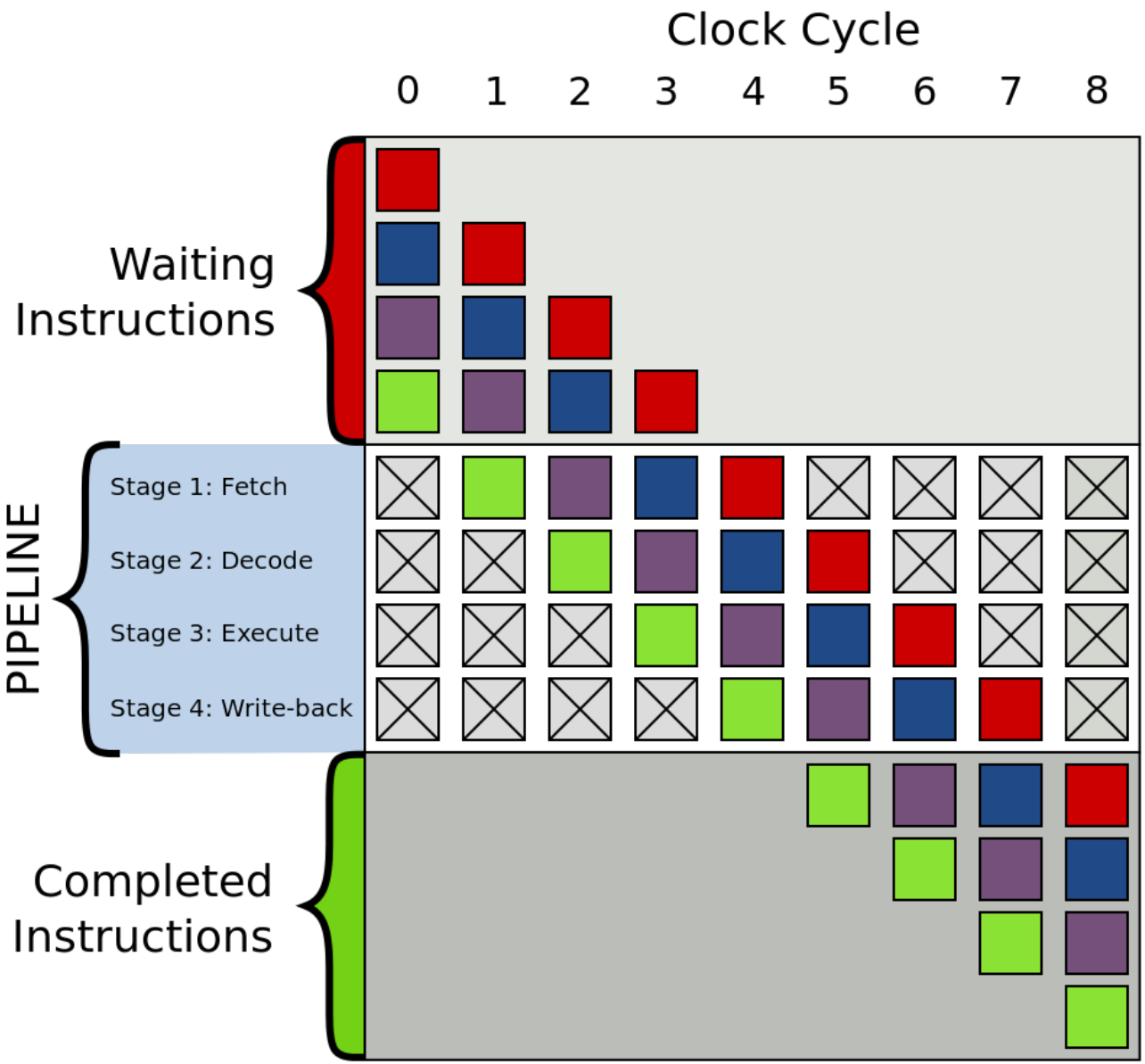
Pipeline de instrucciones: Etapas

Los pipeline varian de acuerdo al procesador. Algunos son cortos y simples (4 etapas), otros son profundos y sumamente complejos (20 etapas).

Las maquinas RISC suelen implementar pipelines sencillos, ya que las instrucciones se pueden resolver rápidamente. Un problema en el pipeline no es tan costoso.

Las maquinas CISC implementan instrucciones mucho mas complejas y que no pueden resolver inmediatamente y tienen que desarmarlas. Pagan muy caro los problemas que pueden surgir en el pipe.

Pipeline: Ejemplo 4 etapas



Pipeline: Problemas?

Con un pipeline tenemos varias instrucciones en distintas etapas de ejecución al mismo tiempo. Esto es bueno porque podemos tener un mecanismo tan granular que nos permita tener listas aproximadamente una instrucción por ciclo.

¿Qué pasa si tenemos un salto (jmp)?

Pipeline: Problemas?

Los saltos son instrucciones especiales. Cuando se ejecutan **puede ser** que no sigan el curso normal de las instrucciones (ip, ip+1, ip+2...)
¿Como impacta en el pipeline esto?

Pipeline: Problemas?

Los saltos son instrucciones especiales. Cuando se ejecutan **puede ser** que no sigan el curso normal de las instrucciones (ip, ip+1, ip+2...)
¿Como impacta en el pipeline esto?

Cuando se ejecuta el salto condicional, podemos saber si tenemos que mover el puntero de instrucción a la dirección del salto o no.

Si no se cumple el salto condicional, es como no haber hecho nada. Esto hace que siga valiendo las instrucciones que ya nos habíamos traído antes y las habíamos metido parcialmente en el pipeline.

Pipeline: Problemas?

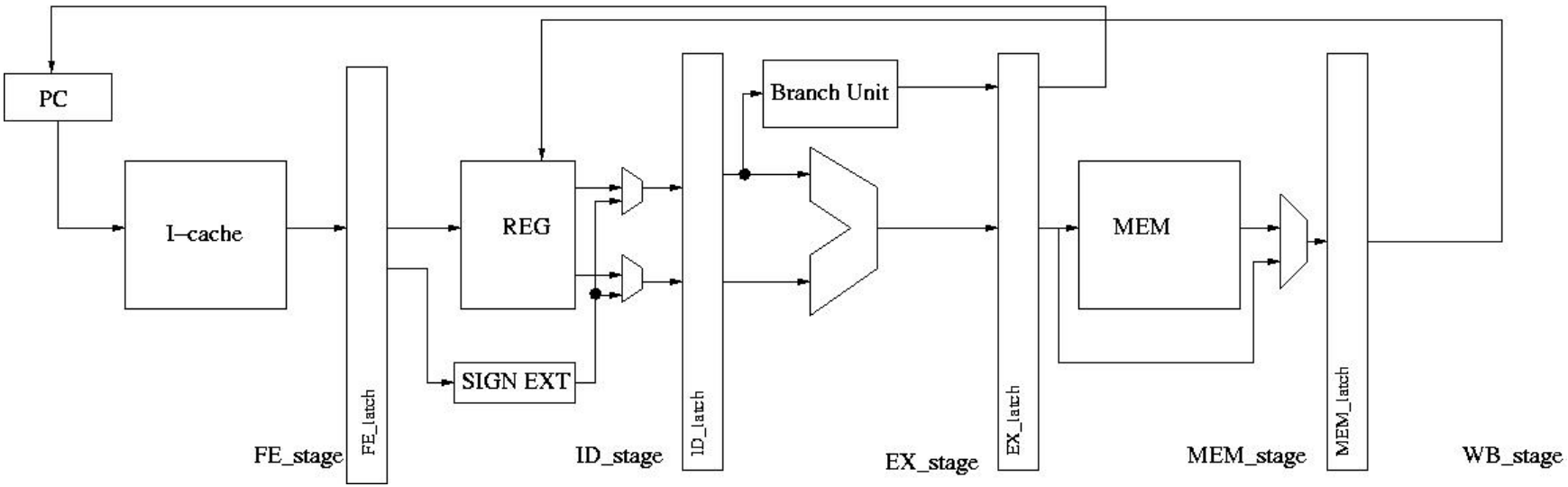
Los saltos son instrucciones especiales. Cuando se ejecutan **puede ser** que no sigan el curso normal de las instrucciones (ip, ip+1, ip+2...)
¿Como impacta en el pipeline esto?

Cuando se ejecuta el salto condicional, podemos saber si tenemos que mover el puntero de instrucción a la dirección del salto o no.

Si no se cumple el salto condicional, es como no haber hecho nada. Esto hace que siga valiendo las instrucciones que ya nos habíamos traído antes y las habíamos metido parcialmente en el pipeline.

Si se cumple el salto condicional, tenemos que hacer algo con el pipeline. No van a valer las instrucciones que antes pensamos que eran las siguientes. Vamos a tener que vaciar el pipeline, para que sean las nuevas instrucciones la que lo llenen.

Pipeline: Diagrama



Esto tenemos hasta ahora.

Pipeline: ¿Entonces?

- **¿Que queremos?** Queremos al oráculo de ~~Orga2~~ CPU, que nos diga que camino va a tomar cada salto, siempre.

- **¿Que tenemos?** Tenemos la posibilidad de agregar estructuras y mecanismos al procesador que especulen por que rama del salto se va seguir.

Pipeline: Algunos mecanismos de predicción estáticos

- **Siempre/nunca salto**: Los saltos condicionales no existen, especulo que nunca voy a saltar y a veces voy a tener razón. (usado en las primeras SPARC/MIPS)

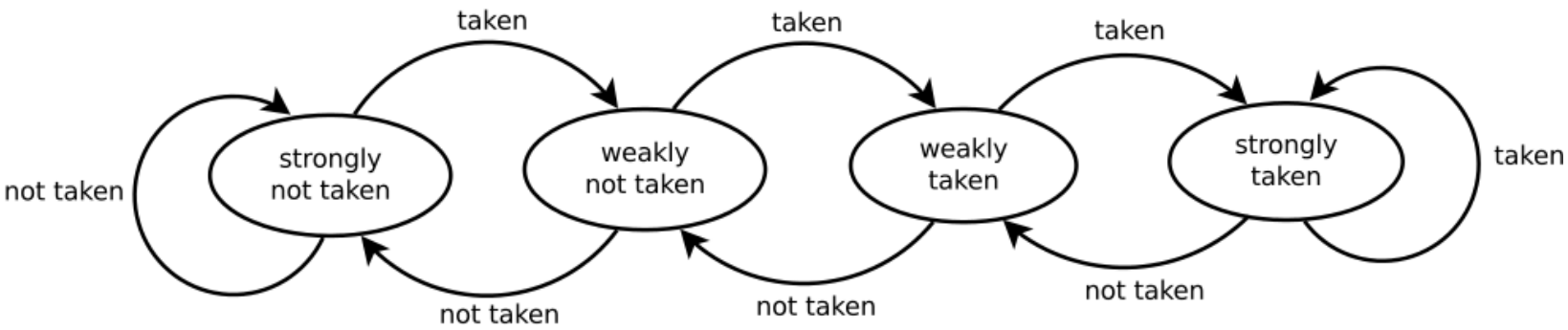
- **Salto siempre si voy para atrás**: La mayoría de los loops comprueban al final si vale la condición. Sino, vuelvo para atrás.

Pipeline: Predicción Dinámica

Si nos guardamos algo de información de lo que hicimos en el pasado, tal vez podamos mejorar nuestras predicciones a futuro.

- **Repetir el ultimo salto**: Si la última vez salté, entonces esta vez saltaré. Si no salte, esta vez no voy a saltar.

- **Contador de saturación**: Maquina de estados que predice mirando las ultimas dos veces que saltamos y se retroalimenta de acuerdo a si la decisión tomada fue correcta. Indica dado los últimos intentos de saltos si es posible que este sea un salto de verdad o que no se ejecute.



Pipeline: Agregando granularidad

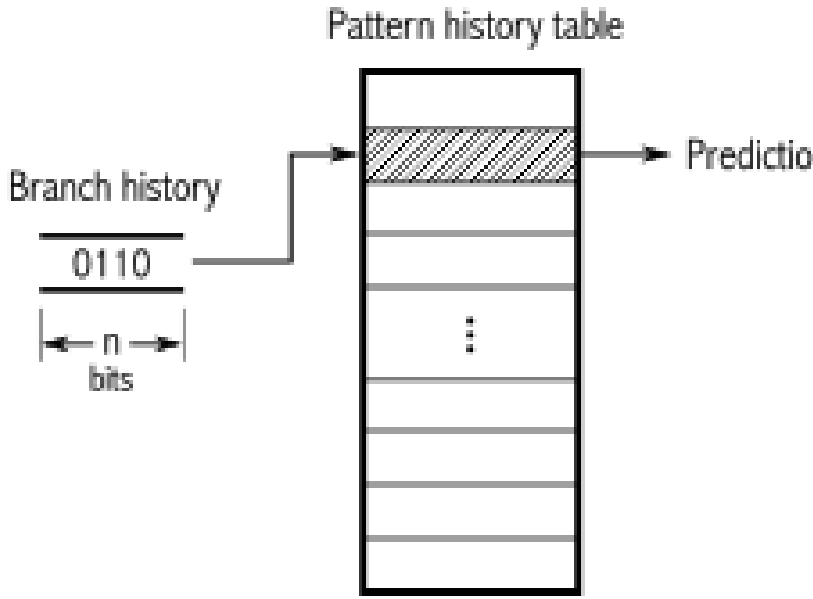
Para no tener una única maquina de estados para todo el sistema, podemos disponer de una tabla que tenga múltiples máquinas de estado. Podemos tener un diccionario que tome un instrucción pointer y devuelva la maquina de estado. Esto permite entender distintos loops y poder modelar apropiadamente cada uno, independientemente de los otros.

IP	Maquina de estados actual
...	
0xAAA0	NT
0xAAB4	ST
0xAADB	T
0xAAF0	ST
...	

Esto se conoce como Branch History Buffer. Se puede definir la cantidad de entradas como máximo que puede tener.

Pipeline: Predicción dinámica

- **Predictor adaptativo de 2 niveles**:
  Permite intentar predecir loops que tengan patrones más largos que 2. Se guarda en un registro de n bits, los últimos saltos tomados o no (1 o 0). Se genera una tabla con 2^n entradas, que en cada entrada tiene un contador de saturación. En cada salto, se indexa la tabla usando el registro y se mira el contador de ahi para predecir si va a ser tomado o no, y cuando se ejecuta, se actualiza el contador y el registro.
  Es la base de los métodos más usados actualmente.



El taller

El taller consta de dos ejercicios. Para resolverlos, vamos a explorar a través de una librería de Intel como podemos correr código ajeno y podemos sacar estadísticas de uso y ver que podrían pasar si la arquitectura interna se comportara de manera distinta.

Los ejercicios consisten:

1. **Simulador de memoria caché**: Vamos a simular memorias caché de distintos procesadores (con sus parámetros correspondientes) y ver cual de ellas presenta mejor desempeño para las distintas aplicaciones.
2. **Simulador de predictor de saltos**: Vamos a escribir uno de los predictores que vimos hoy, el contador por saturación, y ver como se comporta ese y los otros implementados. Veremos que pasa con distintos tamaños de tablas y como podrían impactar en la performance.

Libreria PIN

`http://software.intel.com/en-us/articles/pin-a-dynamic-binary-instrumentation-tool`

> *Pin is a tool for the instrumentation of programs. It supports the Android*, Linux*, OSX* and Windows* operating systems and executables for the IA-32, Intel(R) 64 and Intel(R) Many Integrated Core architectures.*
>
> *Pin allows a tool to insert arbitrary code (written in C or C++) in arbitrary places in the executable. The code is added dynamically while the executable is running. This also makes it possible to attach Pin to an already running process.*

Libreria PIN

Pin provides a rich API that abstracts away the underlying instruction set idiosyncracies and allows context information such as register contents to be passed to the injected code as parameters. Pin automatically saves and restores the registers that are overwritten by the injected code so the application continues to work. Limited access to symbol and debug information is available as well.