CC3301 Programación de software de sistemas

Memorias caché de varios grados de asociatividad, arquitecturas microprogramadas, en pipeline, superescalares, especulativas y con ejecución fuera de orden

Diseño de un caché de 8 KB de 2 grados de asociatividad con líneas de 16 bytes

- Se construye a partir de 2 bancos. Cada banco es un caché de acceso directo de 4 KB que almacena hasta 256 líneas de 16 bytes (C=256). La figura muestra un extracto del contenido:

línea cache	Banco 1		Banco 2	
	etiqueta	contenido	etiqueta	contenido
0a	b0a		80a	
3b	53b		13b	
81	481		681	

- Una línea de la DRAM puede estar solo en uno de los bancos. Por ejemplo la línea 0x3b almacena en el banco izquierdo la línea de la DRAM con etiqueta 0x53b (rango de direcciones 0x53b0 a 0x53bf) y el banco derecho la línea con etiqueta 0x13b.
- Un acceso a la línea L de la DRAM es un acierto si y solo si L es un acierto en alguno de los 2 bancos. La revisión de los 2 cachés se hace en paralelo. Típicamente el acceso toma un ciclo más que en un caché de acceso directo.
- En caso de desacierto, se elige pseudo aleatoriamente uno de los 2 bancos y se hace el *reemplazo* en el banco elegido. Toma unos 100 ciclos del reloj
- Ejercicio: Un programa accede a las direcciones de memoria b0a4, 13b0, 10a8, 4810, 6810, 4818, b0a0, 80a0. Indique qué accesos caen en la memoria caché y cuáles no. Además muestre un posible estado del caché después de aquellos accesos.

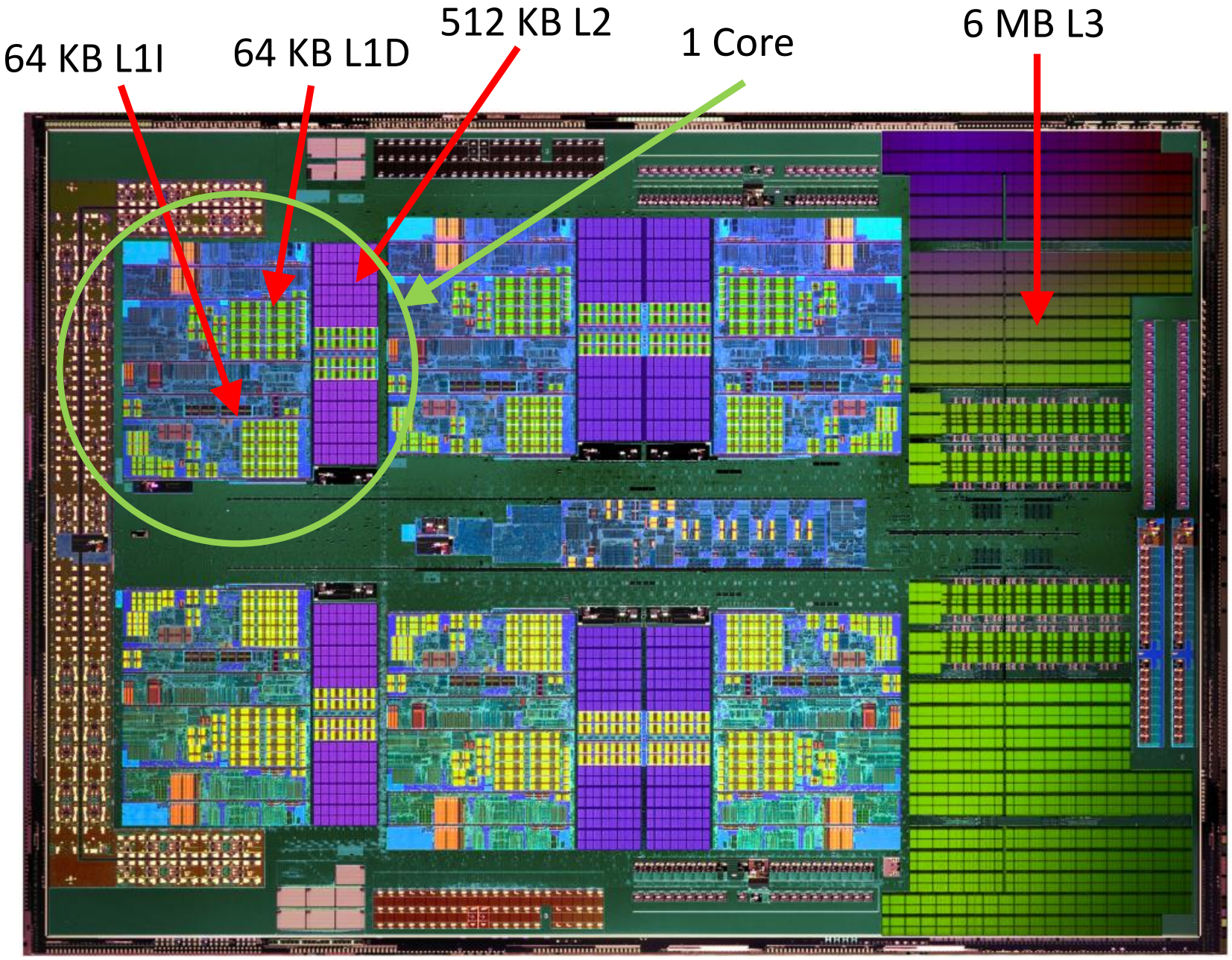
Discusión sobre la eficiencia de un caché

- Un caché de *n* grados de asociatividad se forma de *n* bancos, cada uno corresponde a un caché de acceso directo
- A igual tamaño total del caché, un incremento en los grados de asociatividad mejora la tasa de aciertos
- Para cachés de gran tamaño, 16 grados de asociatividad aumenta marginalmente el área de silicio con respecto a 1 solo grado de asociatividad, pero a partir de 16 grados de asociatividad, no se obtiene una mejora significativa en la tasa de aciertos
- *Tiempo de acceso promedio*: $\alpha \cdot T_C + (1-\alpha) \cdot T_M$
- α es la tasa de aciertos en el caché, T_C es el tiempo de acceso de un acierto y T_M es el tiempo de acceso de un desacierto
- Para cachés de tamaño pequeño, muchos grados de asociatividad empeoran el tiempo de acceso promedio
- Los estudios con programas reales muestran que cuadruplicar el tamaño del caché disminuye los desaciertos solo a la mitad, pero también hace empeorar T_C, lo que puede empeorar el tiempo de acceso promedio: *un cache de 64 KB es mejor que el de 256 KB*

Jerarquía de cachés

- Típicamente se usa una *jerarquía de cachés*: un caché de nivel 1 de 32 KB solo para las instrucciones (L1I) con 3 o 4 ciclos de latencia y otro caché de nivel 1 de 32 KB solo para los datos (L1D). *¡Se lee una instrucción de L1I en paralelo con un acceso a L1D!*
- **La tasa de aciertos en el cache L1 está entre 95 y 97%**
- En caso de desacierto en L1, se busca en un caché L2 de 512 KB con unos 15 ciclos de latencia
- En caso de desacierto en L2, se busca en un caché L3 de 8 MB con unos 25 ciclos de latencia.
- Para cachés L1 se usan 2 o 4 grados de asociatividad, 8 para L2 y 16 para L3, típicamente
- Cada core tiene su propio caché L1 y L2, pero el caché L3 es compartido por grupos de 4 u 8 cores
- *El conocimiento del funcionamiento del caché permite escribir programas que hagan un uso eficiente de la jerarquía de cachés y la memoria*
- *El diseño del cache en los procesadores modernos es crucial para su buen desempeño*
- *Una alta tasa de aciertos en el caché también contribuye a disminuir el consumo energético porque ir a buscar datos a la memoria consume considerablemente más energía que recuperarlos del caché*
- Pagar por una memoria DRAM 25% más rápida, se traduce en un pequeño aumento en la velocidad del computador, muy lejos del 25%

AMD Phenom X6 (2008)



Arquitecturas microprogramadas

- Una micro-CPU guía la ejecución de las instrucciones de la CPU real
- Genera las señales de control que instruyen a las diversas unidades qué operaciones deben realizar
- La unidades son: ALU, banco de registros, interfaz con el bus, saltos condicionales, Pc, Inst, multiplexores, etc.
- Cada instrucción requiere varios ciclos del reloj, por ejemplo *fetch*, *decode*, *execute*
- Ejemplo de ejecución:

```
A.  add   t1,a0,a1
B.  ori   t2,a0,15
C.  lw    t3,16(t1)
D.  bge   t3,t2, I
E.  addi  t4,t2,10
F.  …
G.  …
H.  …
I.  div   t2,t1,t2
J.  addi  t6,t1,1
```

Ciclo	Fetch	Decode	Execute
1	A		
2		A	
3			A
4	B		
5		B	
6			B
7	C		
8		C	
9			C
10	D		
Etc.		D	

- Ejemplo de CPU: Intel 386

Arquitecturas en pipeline

- Las instrucciones ingresan a un pipeline similar a una línea de ensamblaje de automóviles, por ejemplo con etapas *fetch*, *decode* y *execute*: Se ejecuta A, al mismo tiempo que se decodifica B y se lee C
- Toma varios ciclos ejecutar una instrucción
- Se completa la ejecución de una instrucción en cada ciclo
- Problema: un salto anula el trabajo previo de *fetch* y *decode*
- Ejemplo de ejecución:

```
A.  add   t1,a0,a1
B.  ori   t2,a0,15
C.  lw    t3,16(t1)
D.  bge   t3,t2, I
E.  addi  t4,t2,10
F.  …
G.  …
H.  …
I.  div   t6,t1,t2
J.  addi  t2,t1,1
K.  …
L.  …
```

Ciclo	Fetch	Decode	Execute
1	A		
2	B	A	
3	C	B	A
4	D	C	B
5	E	D	C
6	F	E	D
7	I		
8	J	I	
9	K	J	I
10	L	K	J
Etc.			

- Ejemplo de CPU: Intel 486

Arquitecturas superescalares

- Las instrucciones ingresan alternadamente a varios pipelines que avanzan de manera síncrona (Pentium, 1993 y Cortex A55, 2017)
- Con *n* pipelines se puede completar la ejecución de *n* instrucciones por cada ciclo del reloj, como máximo
- Ejemplo con 2 pipelines:

A. add t1,a0,a1
B. ori t2,a0,15
C. lw **t3**,16(t1)
D. bge **t3**,t2, **I**
E. addi t4,t2,10
F. …
G. …
H. …
I. div t6,t1,t2
J. addi t2,t1,1
K. …
L. …

Ciclo	Fetch	Decode	Execute
1	A B		
2	C D	A B	
3	E F	C D	A B
4		D	C
5	G H	E F	D
6	I J		
7	K L	I J	
8	…	K L	I J
Etc.			

- Problemas: La fase de ejecución de una multiplicación toma 3 ciclos, una división muchos más, las lecturas de memoria pueden tomar 3, 15, 25 o 100 ciclos del reloj dependiendo de si se encuentra en algún caché o no
- *Se denominan burbujas en el pipeline y bloquean todos los pipelines, porque las instrucciones necesitan ejecutarse en orden para cumplir la especificación de la arquitectura*

Arquitecturas con ejecución especulativa y fuera de orden (Out Of Order: **OOO**)

- Se dispone de múltiples unidades de ejecución (*ports*) que operan en paralelo: alus, coprocesadores de punto flotante, lectura y escritura de memoria, saltos condicionales
- Instrucciones se ejecutan cuando están listos sus operandos
- En una división o lectura de memoria se continúa ejecutando las instrucciones siguientes, a menos que usen los resultados de la división o lectura
- En un salto condicional se especula (se predice) si ocurre o no
- Ejemplos de CPU: Pentium Pro, 1995, Cortex A9, 2012
- Ejemplo de ejecución:

```
A.  add   t1,a0,a1
B.  ori   t2,a0,15
C.  lw    t3,16(t1)
D.  bge   t3,t2, I
E.  addi  t4,t2,10
F.  …
G.  …
H.  …
I.  div   t6,t1,t2
J.  addi  t2,t1,1
K.  …
L.  …
```

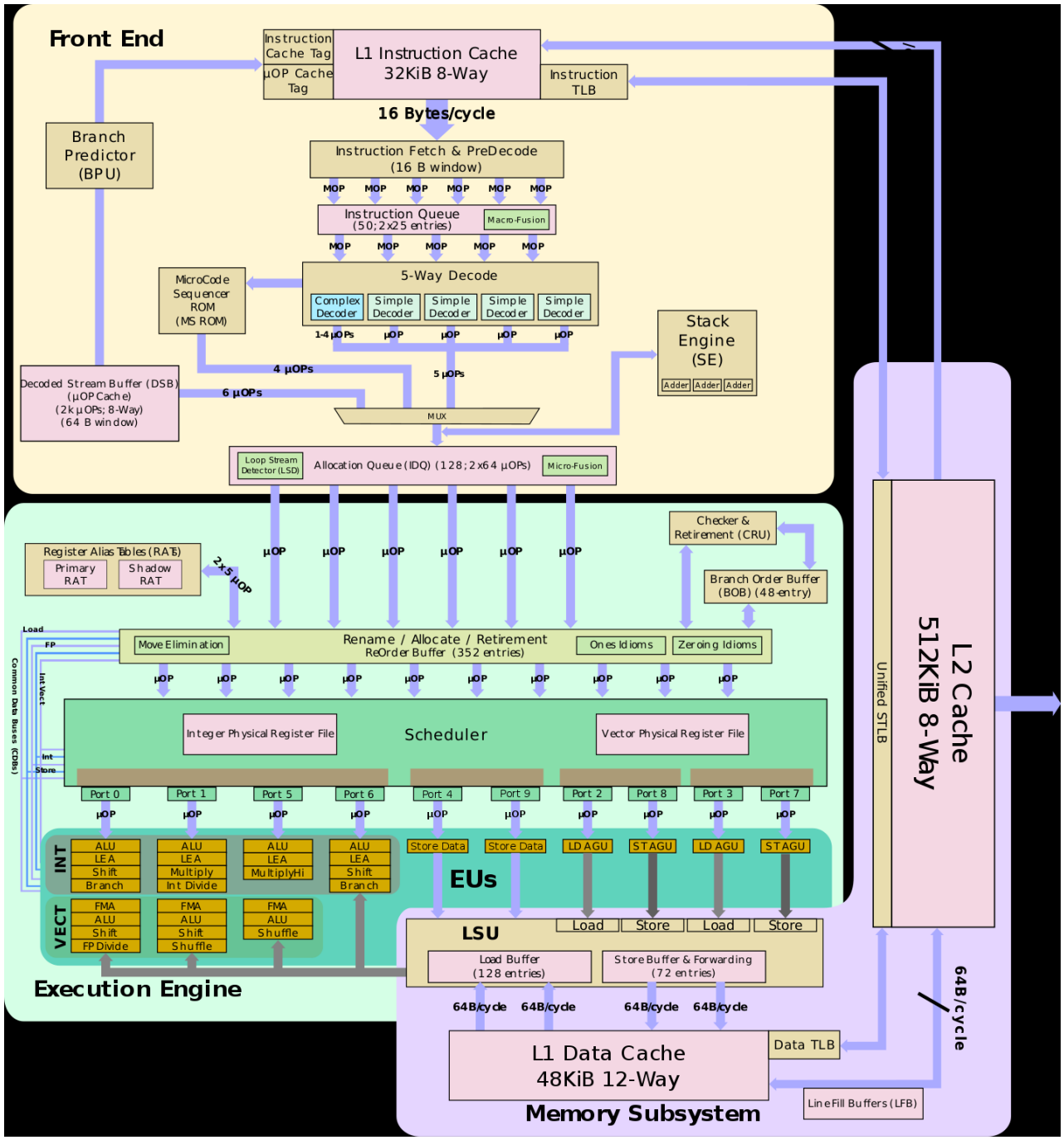
Ciclo	Fetch	Decode	Execute	Retire
1	A B			
2	C D	A B		
3	I J (1)	C D	A B	
4	K L	I J	C (2)	A B
5			C I J (3,4)	
6			C I …	
7			D I …	C (5)
8			I	D
Etc.				I J

Notas

1) La unidad de predicción de saltos especula que D sí salta y por eso se leen I y J, no E y F
2) Se ejecuta solo C porque D depende del resultado de C, además C toma 3 ciclos (está en el caché L1D)
3) Se ejecuta I y J antes que D, a pesar de que D podría no saltar
4) Se pueden ejecutar múltiples instrucciones en paralelo, si hay unidades disponibles y los operandos ya se calcularon
5) Si se determina que D no salta, hay que anular el trabajo hecho por I y J, y retroceder a la instrucción E. El problema es que J alcanzó a modificar t2.

- Los registros arquitecturales son 31, pero existen cientos de registros físicos. Hasta J el registro arquitectural t2 podría estar en el registro físico F125. El resultado de J se almacena en el registro F67 y se anota que t2 es F67 para los futuros usos, pero F125 no se modifica. Esto se llama *register renaming*.
- Cuando se determina que D no salta, se retrocede a la asignación de registros en ese instante, de manera que se deshace el cambio hecho en t2
- Si una instrucción supera la etapa *retire* (se jubila) quiere decir que todos los saltos condicionales en su camino ya fueron resueltos
- La instrucciones se ejecutan fuera de orden (**O**ut **O**f **O**rder), pero se deben jubilar en orden

Ejemplo: Intel sunny cove (2019)



https://en.wikichip.org/wiki/intel/microarchitectures/sunny_cove

Discusión

- Ventaja: a mayor número de unidades de ejecución, mayor velocidad de ejecución, pero depende de la calidad de los predictores
- Problemas:
  - Extremadamente complejo de diseñar
  - Baja eficiencia energética
  - Gran área de silicio
  - Extremadamente difícil de garantizar 100% de seguridad. En 2018 se descubrió que todos los procesadores de Intel con ejecución OOO (desde 1995) tenían un defecto de seguridad que se llamó *meltdown*: ¡Un proceso sin privilegio de administrador puede leer cualquier parte de la memoria!
  - Se debe a que aún cuando se anulen los cambios realizados después de un salto mal especulado, quedan huellas (*side channel*) de los accesos a memoria realizados en la memoria caché, de los cuales se puede deducir contenidos de memoria inaccesibles a los procesos.
  - Aún cuando fue fácil corregir el defecto en las nuevas CPUs de Intel, es imposible corregirlo en las CPUs ya vendidas
  - Se modificaron todos los núcleos de los sistemas operativos para evitar el defecto, pero haciéndolos menos eficientes
  - Aún así persiste otro defecto de seguridad, más difícil de explotar, pero que afecta a todos los cores con ejecución OOO: *spectre*
- Conocer la manera en que se ejecutan las instrucciones en una CPU permite escribir programas más eficientes y evitar fallas de seguridad