ECE 571 – Advanced Microprocessor-Based Design Lecture 14

Vince Weaver

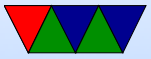
https://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

4 October 2024

Announcements

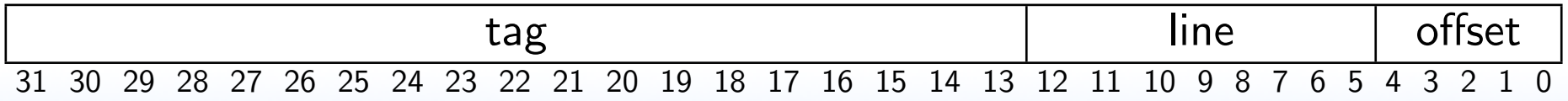
- HW#5 will be posted, caches

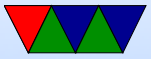


Cache Parameters Example 2

32kB cache (2^{15}), 4-way (2^2)
32 Byte linesize (2^5), 32-bit address size (2^{32})

offset = $log_2(linesize)$ = 5 bits
lines = $log_2((cachesize/\#ways)/linesize)$ = 256 lines (8 bits)
tag = addresssize - (offset bits + line bits) = 19 bits



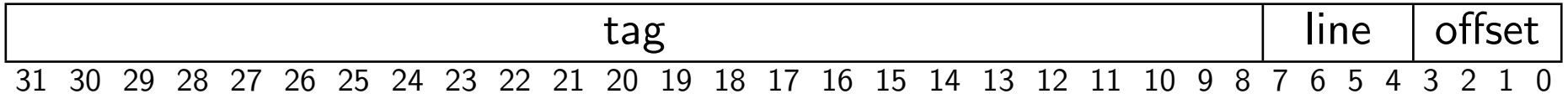


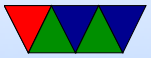
Cache Example

512 Byte cache, 2-Way Set Associative, with 16 byte lines, LRU replacement.

LRU=1 (least recently used is true) means it is oldest and should be replaced.

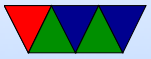
24-bit tag, 16 lines (4 bits), 4-bit offset.





Cache Example 1

- We're just simulating the bits and tags
- Do we need to simulate the actual data?

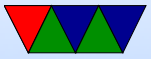


Cache Example – Instruction 1

```
ldrb r1, 0x00000000
```

	Way 0				Way 1			
line	V	D	LRU	Tag	V	D	LRU	Tag
0	1	0	0	0000 00	0			
1	0				0			
2	0				0			
3	0				0			
4	0				0			
5	0				0			
...								
b	0				0			
c	0				0			
d	0				0			
e	0				0			
f	0				0			

Miss, Cold

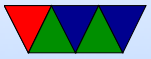


Cache Example – Instruction 2

```
ldrb r1, 0x00000001
```

	Way 0				Way 1			
line	V	D	LRU	Tag	V	D	LRU	Tag
0	1	0	0	0000 00	0			
1	0				0			
2	0				0			
3	0				0			
4	0				0			
5	0				0			
...								
b	0				0			
c	0				0			
d	0				0			
e	0				0			
f	0				0			

Hit

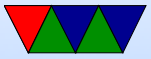


Cache Example – Instruction 3

```
ldrb r1, 0x00000010
```

	Way 0				Way 1			
line	V	D	LRU	Tag	V	D	LRU	Tag
0	1	0	0	0000 00	0			
1	1	0	0	0000 00	0			
2	0				0			
3	0				0			
4	0				0			
5	0				0			
...								
b	0				0			
c	0				0			
d	0				0			
e	0				0			
f	0				0			

Miss, Cold

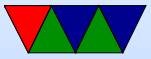


Cache Example – Instruction 4

```
ldrb r1, 0x80000010
```

	Way 0				Way 1			
line	V	D	LRU	Tag	V	D	LRU	Tag
0	1	0	0	0000 00	0			
1	1	0	1	0000 00	1	0	0	8000 00
2	0				0			
3	0				0			
4	0				0			
5	0				0			
...								
b	0				0			
c	0				0			
d	0				0			
e	0				0			
f	0				0			

Miss, Cold

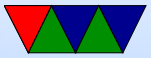


Cache Example – Instruction 5

```
ldrb r1, 0xC0000010
```

	Way 0				Way 1			
line	V	D	LRU	Tag	V	D	LRU	Tag
0	1	0	0	0000 00	0			
1	1	0	0	C000 00	1	0	1	8000 00
2	0				0			
3	0				0			
4	0				0			
5	0				0			
...								
b	0				0			
c	0				0			
d	0				0			
e	0				0			
f	0				0			

Miss, Cold (never in cache previously)

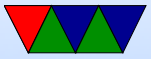


Cache Example – Instruction 6

```
ldrb r1, 0xC0000002
```

	Way 0				Way 1			
line	V	D	LRU	Tag	V	D	LRU	Tag
0	1	0	1	0000 00	1	0	0	c000 00
1	1	0	0	C000 00	1	0	1	8000 00
2	0				0			
3	0				0			
4	0				0			
5	0				0			
. . .								
b	0				0			
c	0				0			
d	0				0			
e	0				0			
f	0				0			

Miss, Cold



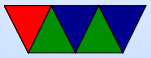
Cache Example – Instruction 7

```
ldrb r1, 0x00000010
```

	Way 0				Way 1			
line	V	D	LRU	Tag	V	D	LRU	Tag
0	1	0	1	0000 00	1	0	0	c000 00
1	1	0	1	C000 00	1	0	0	0000 00
2	0				0			
3	0				0			
4	0				0			
5	0				0			
. . .								
b	0				0			
c	0				0			
d	0				0			
e	0				0			
f	0				0			

Miss, Conflict

Would not have been a miss if fully associative cache

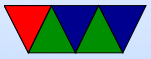


Cache Example – Instruction 8

```
strb r1, 0x00000005
```

	Way 0				Way 1			
line	V	D	LRU	Tag	V	D	LRU	Tag
0	1	1	0	0000 00	1	0	1	c000 00
1	1	0	1	C000 00	1	0	0	0000 00
2	0				0			
3	0				0			
4	0				0			
5	0				0			
. . .								
b	0				0			
c	0				0			
d	0				0			
e	0				0			
f	0				0			

Hit

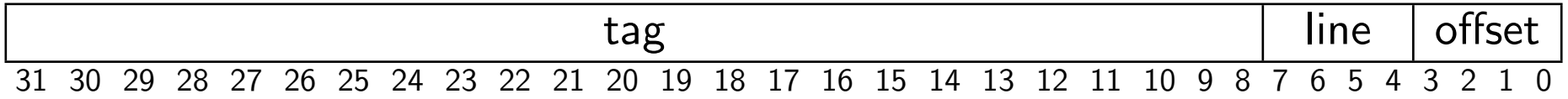


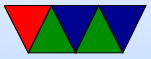
Cache Example Two

512 Byte cache, 2-Way Set Associative, with 16 byte lines, LRU replacement.

write-back, write-allocate

24-bit tag, 16 lines (4 bits), 4-bit offset.

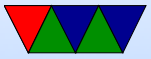




Cache Example 2

Note for LRU: least-recently used

- LRU = 0, least recently used is false, so is most recent
- LRU = 1, least recently used i true, so is oldest

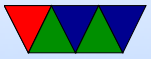


Cache Example – Instruction 1

```
ldrb r1, 0x00000000
```

	Way 0				Way 1			
line	V	D	LRU	Tag	V	D	LRU	Tag
0	1	0	0	0000 00	0			
1	0				0			
2	0				0			
3	0				0			
4	0				0			
5	0				0			
...								
b	0				0			
c	0				0			
d	0				0			
e	0				0			
f	0				0			

Miss, Cold

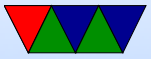


Cache Example – Instruction 2

```
strb r1, 0x00000001
```

	Way 0				Way 1			
line	V	D	LRU	Tag	V	D	LRU	Tag
0	1	1	0	0000 00	0			
1	0				0			
2	0				0			
3	0				0			
4	0				0			
5	0				0			
...								
b	0				0			
c	0				0			
d	0				0			
e	0				0			
f	0				0			

Hit (updates the dirty bit)

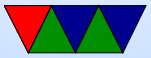


Cache Example – Instruction 3

```
strb r1, 0x00000105
```

	Way 0				Way 1			
line	V	D	LRU	Tag	V	D	LRU	Tag
0	1	1	1	0000 00	1	1	0	0000 01
1	0				0			
2	0				0			
3	0				0			
4	0				0			
5	0				0			
...								
b	0				0			
c	0				0			
d	0				0			
e	0				0			
f	0				0			

Miss, Cold (bring in as it's write allocate)

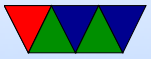


Cache Example – Instruction 4

```
ldrb r1, 0x00000206
```

	Way 0				Way 1			
line	V	D	LRU	Tag	V	D	LRU	Tag
0	1	0	0	0000 02	1	1	1	0000 01
1	0				0			
2	0				0			
3	0				0			
4	0				0			
5	0				0			
...								
b	0				0			
c	0				0			
d	0				0			
e	0				0			
f	0				0			

Miss, Cold

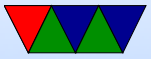


Cache Example – Instruction 5

```
ldrb r1, 0x00000000
```

	Way 0				Way 1			
line	V	D	LRU	Tag	V	D	LRU	Tag
0	1	0	1	0000 02	1	0	0	0000 00
1	0				0			
2	0				0			
3	0				0			
4	0				0			
5	0				0			
...								
b	0				0			
c	0				0			
d	0				0			
e	0				0			
f	0				0			

Miss, Conflict

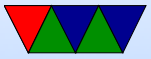


Cache Example – Instruction 6

```
ldrb r1, 0x00000030
```

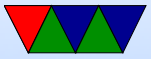
	Way 0				Way 1			
line	V	D	LRU	Tag	V	D	LRU	Tag
0	1	0	1	0000 02	1	0	0	0000 00
1	0				0			
2	0				0			
3	1	0	0	0000 00	0			
4	0				0			
5	0				0			
...								
b	0				0			
c	0				0			
d	0				0			
e	0				0			
f	0				0			

Miss, Cold

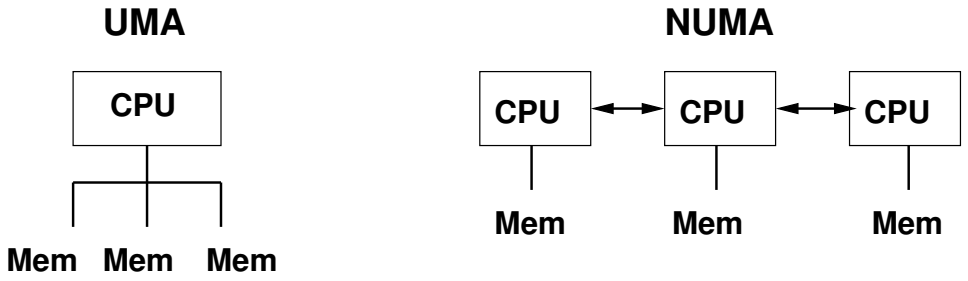


CMP Issues

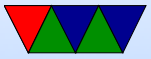
Things were complex enough, what happens when we allow multiple cores to share memory but also have caches.



UMA and NUMA

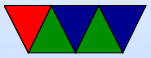


- UMA – Uniform Memory Access
  same speed to access all of memory
- NUMA – Non-Uniform Memory Access
  accesses to memory connected to other CPU can take longer



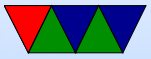
Cache Coherency

- Only gets complicated when you start writing.
- Need some way to know if other core's caches have the address you are accessing
  - Snoopy – "snoop" the bus
  - Directory – have central logic (doesn't scale as well)
- Cache coherency protocols
  - MESI
  - MOESI



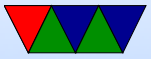
MESI

- State machine: modified, exclusive, shared, invalid
- Invalid – starts out here
- Shared – only has been read, same as memory, can be in multiple caches
- Exclusive – a core is requesting to write, so it gets exclusive access, invalidates all other copies
- Modified – dirty, has been written to. Write back and then can change to S

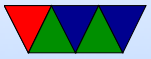


Other Concerns

- What about load/store values in the pipeline that haven't been retired yet?
- What about Memory Model? Can loads pass stores? x86 has strict model, others no so much
  Apple M2 has special "act like x86 mode" to make emulation easier
- Special assembly instructions to flush cache, memory barriers. etc
- Locking in multi-threaded apps

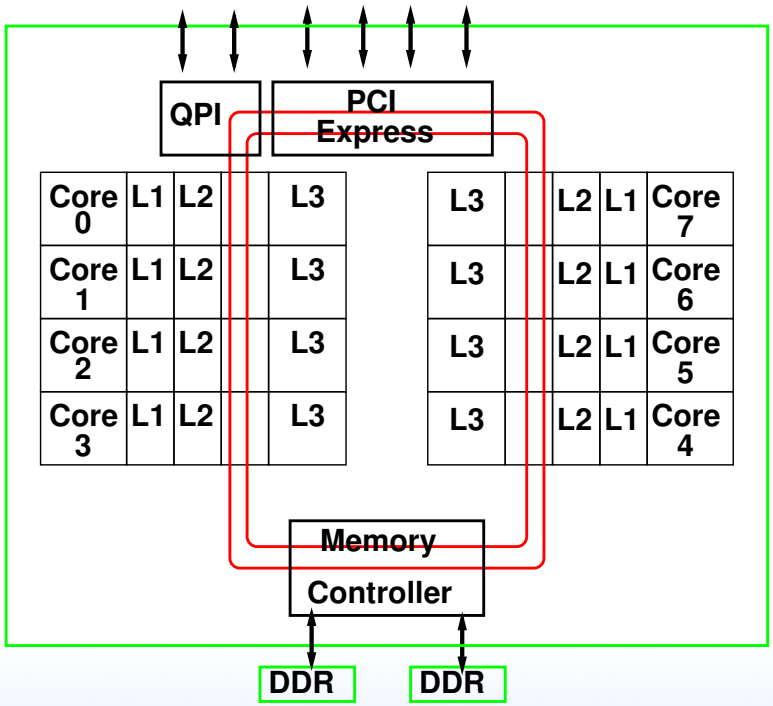


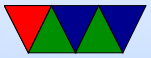
Example Real-World Cache layouts



Haswell-EP Cache Layout

Note: see "Cache Coherence Protocol and Memory Performance of the Intel Haswell-EP Architecture" by Molka, Hackenberg, Schöne, Nagel.



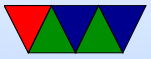


Haswell-EP Cache Parameters

- per core 32kB L1 I/D – 4 clocks
  64B/line, 8-Way
  (shared if hyper-threaded)
  writeback

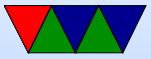
- mOp cache? 1.5K instructions, 8-way, 6Mop/line
  Loop stream detector, can execute w/o accessing icache

- per core 256kB L2 unified – 11 clocks

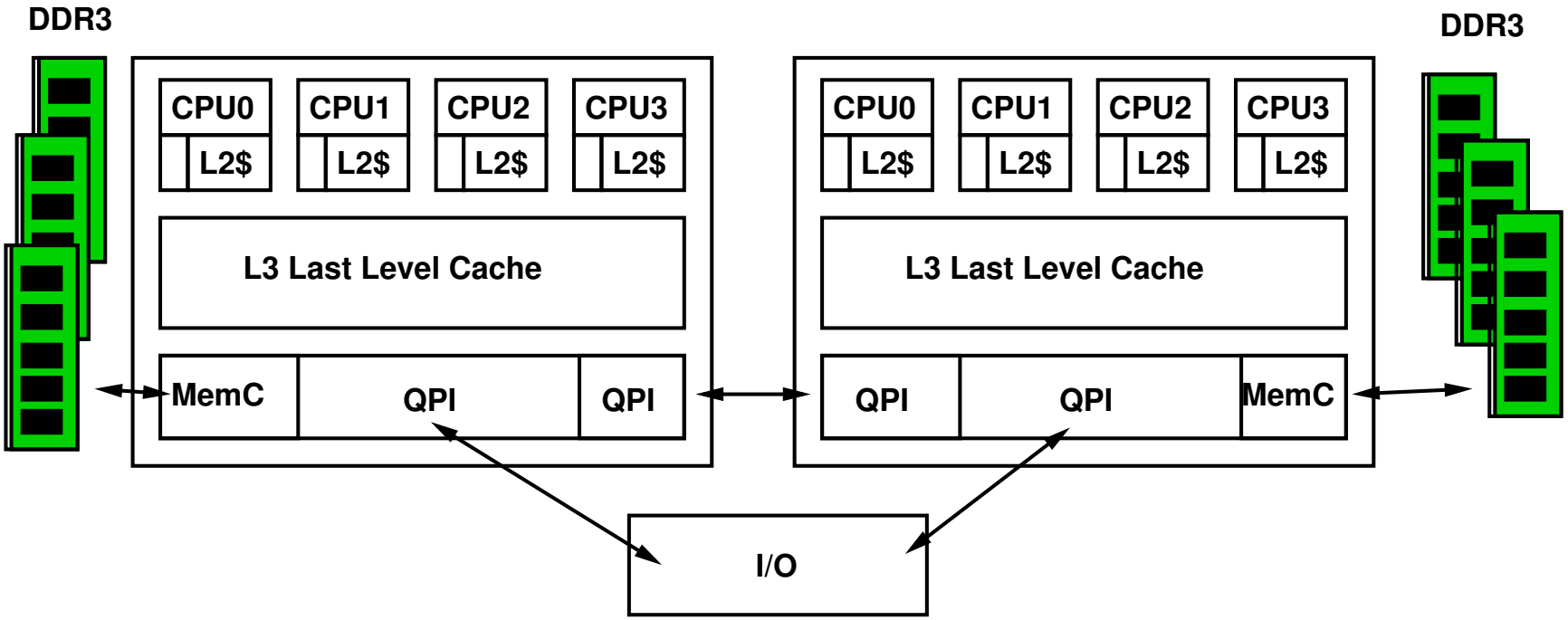


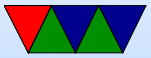
64B/line, 8-way
writeback. non-inclusive

- shared L3 20MB, writeback
- various hw prefetchers operating



SandyBridge Cache Layout



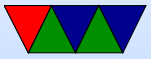


SandyBridge Cache Parameters

- per core 32kB L1 I/D – 4 clocks
  64B/line, 8-Way
  (shared if hyper-threaded)
  writeback

- mOp cache? 1.5K instructions, 8-way, 6Mop/line
  Loop stream detector, can execute w/o accessing icache

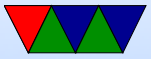
- per core 256kB L2 unified – 12 clocks



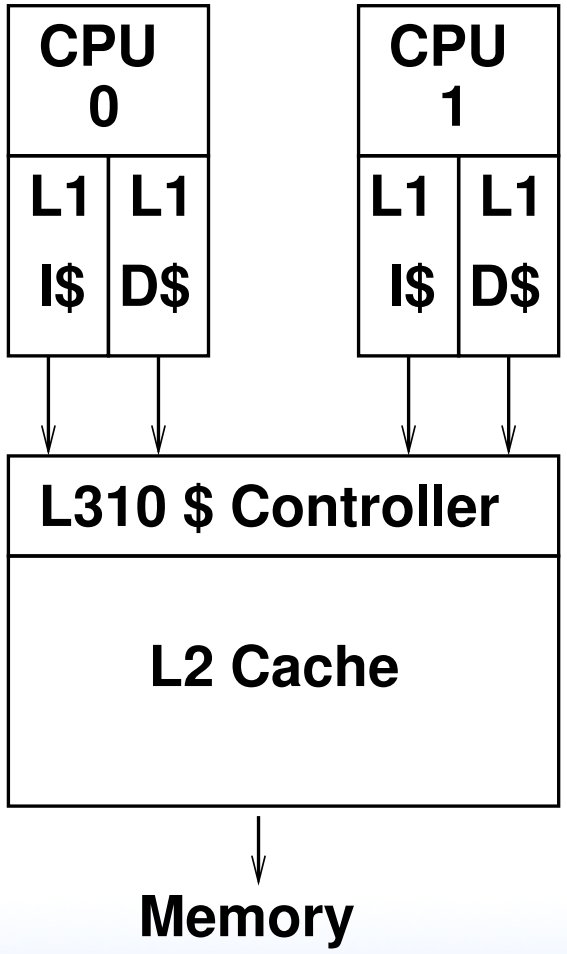
64B/line, 8-way
writeback. non-inclusive

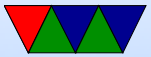
- shared L3 1MB-20MB – 26-31 clocks
  64B/line. 12-way (varies)
  writeback, inclusive

- various hw prefetchers operating



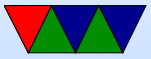
Cortex A9 Cache Layout





Cortex A9 Cache Layout

- OMAP4430 processor
- 32kB 4-way associative, separate L1-I and L1-D
  pseudo-round-robin or pseudo random replacement
  8-word line size (32B)
  critical-word first filling
  instruction: VIPT, data: PIPT
- Optional L2 cache controller
  Pandaboard has L310 L2 cache controller, 1MB 16-way



Optional prefetcher

- data cache reads/writes non-blocking, 4 outstanding misses
  write buffer: 4 64-bit, allowing write combining