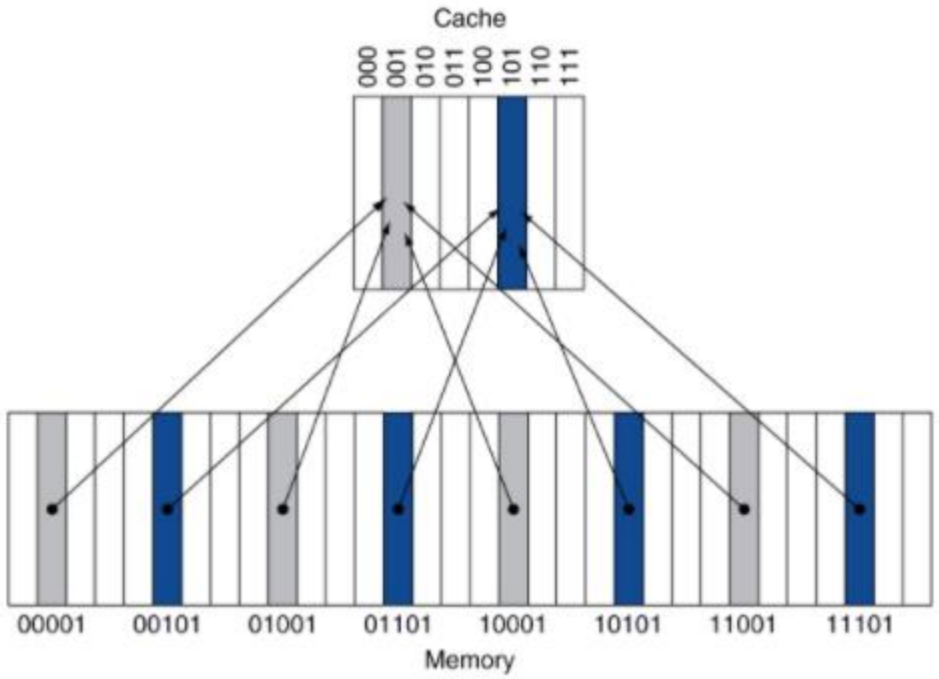
ECE M116C / CS 151B: Week 7 Section

Justin Feng

Cache and Memory Design

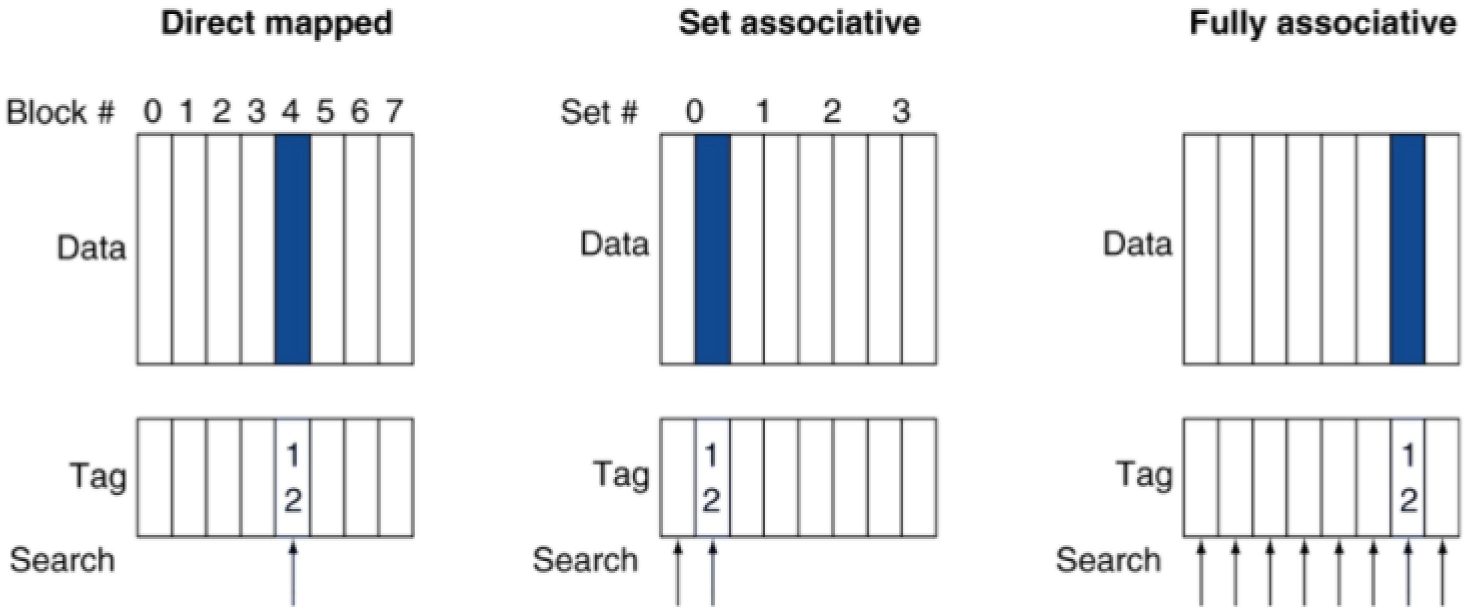
UCLA

Direct Mapped Cache



UCLA

Full Comparison



UCLA

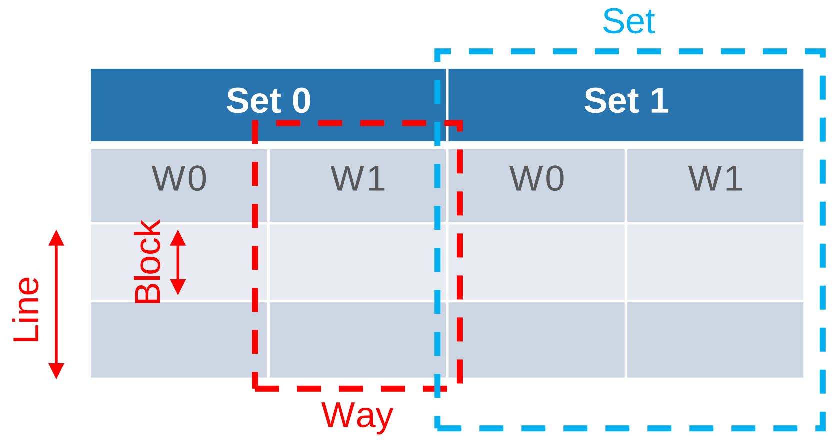
Definitions

Cache Block: minimum granularity brought up from cache (usually 4B or 8B)

Cache Line: consists of multiple blocks, that forms a "line"

Ways: same as cache line

Set: consist of multiple ways/lines



UCLA

Definitions

What are C, B, and S?

C = log(blocks per cache)

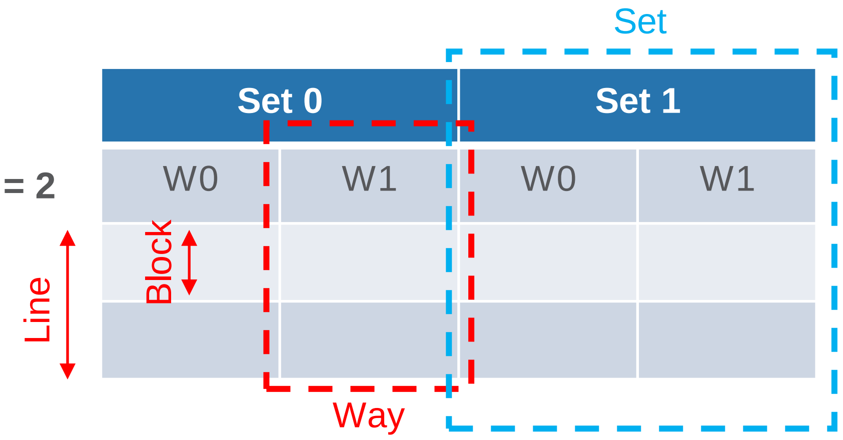
Ex) 8 blocks in this cache -> log(8) = 3

B = log(blocks per line)

Ex) if this cache has 8 blocks/line -> log(4) = 2

S = log(lines per set)

Ex) 2 lines per set -> log(2) = 1



How to determine Tag, Index, and Offset?

32 kB Cache, 8B cache line (assumes 1B cache block), 32 bit address

Direct Mapped?

Fully Associative?

2-Way Set Associative?

4-Way Set Associative?

How to determine Tag, Index, and Offset?

32 kB Cache, 8B cache line (assumes 1B cache block), 32 bit address

Direct Mapped? 17 bit Tag, 12 bit Index, 3 bit Offset

Fully Associative? 29 bit Tag, 3 bit Offset

2-Way Set Associative? 18 bit Tag, 11 bit Index, 3 bit Offset,

4-Way Set Associative? 19 bit Tag, 10 bit Index, 3 bit Offset

What if…

32 kB Cache, **16B cache line, 8B cache block**, 32 bit address

Direct Mapped?

Fully Associative?

2-Way Set Associative?

4-Way Set Associative?

What if…

32 kB Cache, **16B cache line, 8B cache block**, 32 bit address

Direct Mapped? 20 bit Tag, 11 bit Index, 1 bit Offset

Fully Associative? 31 bit Tag, 1 bit Offset

2-Way Set Associative? 21 bit Tag, 10 bit Index, 1 bit Offset

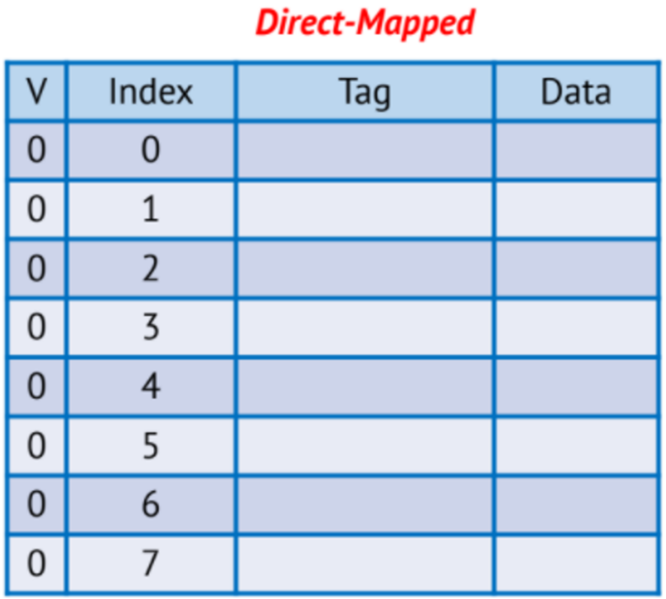
4-Way Set Associative? 22 bit Tag, 9 bit Index, 1 bit Offset

UCLA

Direct Mapped Example

Sequence of Memory References (Note: addresses already shifted from CPU to cache, so identify idx and tag!):

1) 22 (10110)
2) 26 (11010)
3) 22 (10110)
4) 26 (11010)
5) 16 (10000)
6) 3 (00011)
7) 16 (10000)
8) 18 (10010)
9) 0 (00000)



Direct-Mapped

V	Index	Tag	Data
0	0		
0	1		
0	2		
0	3		
0	4		
0	5		
0	6		
0	7		

UCLA

Direct Mapped Example

Sequence of Memory References:

1) **22 (10110)**
2) 26 (11010)
3) 22 (10110)
4) 26 (11010)
5) 16 (10000)
6) 3 (00011)
7) 16 (10000)
8) 18 (10010)
9) 0 (00000)

V	Index	Tag	Data
0	0		
0	1		
0	2		
0	3		
0	4		
0	5		
1	**6**	**10**	**M[10110]**
0	7		

UCLA

Direct Mapped Example

Sequence of Memory References:

1) 22 (10110)
2) **26 (11010)**
3) 22 (10110)
4) 26 (11010)
5) 16 (10000)
6) 3 (00011)
7) 16 (10000)
8) 18 (10010)
9) 0 (00000)

V	Index	Tag	Data
0	0		
0	1		
1	**2**	**11**	**M[11010]**
0	3		
0	4		
0	5		
1	6	10	M[10110]
0	7		

Direct Mapped Example

Sequence of Memory References:

1) 22 (10110)
2) 26 (11010)
3) **22 (10110)**
4) 26 (11010)
5) 16 (10000)
6) 3 (00011)
7) 16 (10000)
8) 18 (10010)
9) 0 (00000)

V	Index	Tag	Data
0	0		
0	1		
1	2	11	M[11010]
0	3		
0	4		
0	5		
1	**6**	**10**	**M[10110]**
0	7		

UCLA

Direct Mapped Example

Sequence of Memory References:

1) 22 (10110)
2) 26 (11010)
3) 22 (10110)
4) **26 (11010)**
5) 16 (10000)
6) 3 (00011)
7) 16 (10000)
8) 18 (10010)
9) 0 (00000)

V	Index	Tag	Data
0	0		
0	1		
1	**2**	**11**	**M[11010]**
0	3		
0	4		
0	5		
1	6	10	M[10110]
0	7		

UCLA

Direct Mapped Example

Sequence of Memory References:

1) 22 (10110)
2) 26 (11010)
3) 22 (10110)
4) 26 (11010)
5) **16 (10000)**
6) 3 (00011)
7) 16 (10000)
8) 18 (10010)
9) 0 (00000)

V	Index	Tag	Data
1	**0**	**10**	**M[10000]**
0	1		
1	2	11	M[11010]
0	3		
0	4		
0	5		
1	6	10	M[10110]
0	7		

Direct Mapped Example

Sequence of Memory References:

1) 22 (10110)
2) 26 (11010)
3) 22 (10110)
4) 26 (11010)
5) 16 (10000)
6) **3 (00011)**
7) 16 (10000)
8) 18 (10010)
9) 0 (00000)

V	Index	Tag	Data
1	0	10	M[10000]
0	1		
1	2	11	M[11010]
1	**3**	**00**	**M[00011]**
0	4		
0	5		
1	6	10	M[10110]
0	7		

Direct Mapped Example

Sequence of Memory References:

1) 22 (10110)
2) 26 (11010)
3) 22 (10110)
4) 26 (11010)
5) 16 (10000)
6) 3 (00011)
7) **16 (10000)**
8) 18 (10010)
9) 0 (00000)

V	Index	Tag	Data
1	**0**	**10**	**M[10000]**
0	1		
1	2	11	M[11010]
1	3	00	M[00011]
0	4		
0	5		
1	6	10	M[10110]
0	7		

UCLA

Direct Mapped Example

Sequence of Memory References:

1) 22 (10110)
2) 26 (11010)
3) 22 (10110)
4) 26 (11010)
5) 16 (10000)
6) 3 (00011)
7) 16 (10000)
8) **18 (10010)**
9) 0 (00000)

V	Index	Tag	Data
1	0	10	M[10000]
0	1		
1	**2**	**10**	**M[10010]**
1	3	00	M[00011]
0	4		
0	5		
1	6	10	M[10110]
0	7		

Direct Mapped Example

Sequence of Memory References:

1) 22 (10110)
2) 26 (11010)
3) 22 (10110)
4) 26 (11010)
5) 16 (10000)
6) 3 (00011)
7) 16 (10000)
8) 18 (10010)
9) **0 (00000)**

V	Index	Tag	Data
1	**0**	**00**	**M[00000]**
0	1		
1	2	10	M[10010]
1	3	00	M[00011]
0	4		
0	5		
1	6	10	M[10110]
0	7		

Direct Mapped Example

Miss Rate:

Total Misses: 6

Total Memory Accesses: 9

Miss Rate: 6/9 * 100 = 66%

V	Index	Tag	Data
1	0	00	M[00000]
0	1		
1	2	10	M[10010]
1	3	00	M[00011]
0	4		
0	5		
1	6	10	M[10110]
0	7		

Fully Associative Example (LRU)

Same Cache size as before.

Sequence of Memory References:

1) 0-7 (00000-00111) (filled in order)

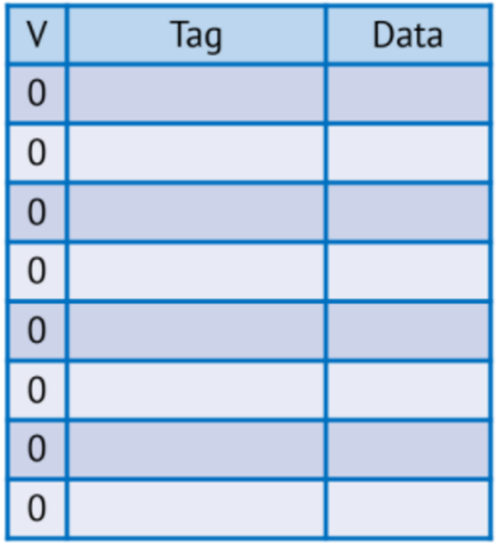
2) 1 (00001)

3) 8 (01000)

4) 11 (01011)

5) 9 (01001)

Fully Assoc.



V	Tag	Data
0		
0		
0		
0		
0		
0		
0		
0		

Fully Associative Example (LRU)

Sequence of Memory References:

1) 0-7 (00000-00111) (filled in order)

2) 1 (00001)

3) 8 (01000)

4) 11 (01011)

5) 9 (01001)

V	Tag	Data	LRU
1	"00000"	M[00000]	0
1	"00001"	M[00001]	1
1	"00010"	M[00010]	2
1	"00011"	M[00011]	3
1	"00100"	M[00100]	4
1	"00101"	M[00101]	5
1	"00110"	M[00110]	6
1	"00111"	M[00111]	7

UCLA

Fully Associative Example (LRU)

Sequence of Memory References:

1) 0-7 (00000-00111) (filled in order)

2) 1 (00001)

3) 8 (01000)

4) 11 (01011)

5) 9 (01001)

V	Tag	Data	LRU
1	"00000"	M[00000]	**0**
1	**"00001"**	**M[00001]**	**7**
1	"00010"	M[00010]	**1**
1	"00011"	M[00011]	**2**
1	"00100"	M[00100]	**3**
1	"00101"	M[00101]	**4**
1	"00110"	M[00110]	**5**
1	"00111"	M[00111]	**6**

Fully Associative Example (LRU)

Sequence of Memory References:

1) 0-7 (00000-00111) (filled in order)

2) 1 (00001)

3) 8 (01000)

4) 11 (01011)

5) 9 (01001)

V	Tag	Data	LRU
1	**"01000"**	**M[01000]**	**7**
1	"00001"	M[00001]	**6**
1	"00010"	M[00010]	**0**
1	"00011"	M[00011]	**1**
1	"00100"	M[00100]	**2**
1	"00101"	M[00101]	**3**
1	"00110"	M[00110]	**4**
1	"00111"	M[00111]	**5**

Fully Associative Example (LRU)

Sequence of Memory References:

1) 0-7 (00000-00111) (filled in order)

2) 1 (00001)

3) 8 (01000)

4) 11 (01011)

5) 9 (01001)

V	Tag	Data	LRU
1	"01000"	M[01000]	**6**
1	"00001"	M[00001]	**5**
1	**"01011"**	**M[01011]**	**7**
1	"00011"	M[00011]	**0**
1	"00100"	M[00100]	**1**
1	"00101"	M[00101]	**2**
1	"00110"	M[00110]	**3**
1	"00111"	M[00111]	**4**

Fully Associative Example (LRU)

Sequence of Memory References:

1) 0-7 (00000-00111) (filled in order)

2) 1 (00001)

3) 8 (01000)

4) 11 (01011)

5) 9 (01001)

V	Tag	Data	LRU
1	"01000"	M[01000]	**5**
1	"00001"	M[00001]	**4**
1	"01011"	M[01011]	**6**
1	**"01001"**	**M[01001]**	**7**
1	"00100"	M[00100]	**0**
1	"00101"	M[00101]	**1**
1	"00110"	M[00110]	**2**
1	"00111"	M[00111]	**3**

2-way Set Associative Example (LRU)

Same Cache size as before.

Sequence of Memory references:

1) 4 (00100)

2) 8 (01000)

3) 12 (01100)

4) 8 (01000)

Set 0		Set 1		Set 2		Set 3	
W0	W1	W0	W1	W0	W1	W0	W1
000(0)	000(1)	00(0)	00(1)	00(0)	00(1)	00(0)	00(1)

UCLA

2-way Set Associative Example (LRU)

Sequence of Memory references:

1) 4 (00100)

2) 8 (01000)

3) 12 (01100)

4) 8 (01000)

Set 0		Set 1		Set 2		Set 3	
W0	W1	W0	W1	W0	W1	W0	W1
001(1)	000(0)	00(0)	00(1)	00(0)	00(1)	00(0)	00(1)

2-way Set Associative Example (LRU)

Sequence of Memory references:

1) 4 (00100)

2) 8 (01000)

3) 12 (01100)

4) 8 (01000)

Set 0		Set 1		Set 2		Set 3	
W0	W1	W0	W1	W0	W1	W0	W1
001**(0)**	**010(1)**	00(1)	00(0)	00(1)	00(0)	00(1)	00(0)

2-way Set Associative Example (LRU)

Sequence of Memory references:

1) 4 (00100)

2) 8 (01000)

3) 12 (01100)

4) 8 (01000)

Set 0		Set 1		Set 2		Set 3	
W0	W1	W0	W1	W0	W1	W0	W1
011(1)	010**(0)**	00(1)	00(0)	00(1)	00(0)	00(1)	00(0)

UCLA

2-way Set Associative Example (LRU)

Sequence of Memory references:

1) 4 (00100)

2) 8 (01000)

3) 12 (01100)

4) 8 (01000)

Set 0		Set 1		Set 2		Set 3	
W0	W1	W0	W1	W0	W1	W0	W1
011**(0)**	010**(1)**	00(1)	00(0)	00(1)	00(0)	00(1)	00(0)

Pseudo LRU

-Trick: approximate LRU – store 1 bit per row

Algorithm:

- -every row is zero
- -if a hit, set flag to 1.
- -on a miss, overwrite data of first row with flag = 0
- -if all bits = 1: reset the bits (because no candidate to evict).

-Benefit: low overhead, efficient

Types of Cache Misses

-**Compulsory Miss**: any initial fetch of a cache line is a miss

-**Capacity Miss**: when we miss because before we could rereference a fetched line it got replaced due to limited capacity (most likely to occur in FA)

-**Conflict Miss**: when there is a cache line in an index that is not what we request (SA, DM)

How to Store?

What to do on a store?

<u>**Data Exists in Cache**</u>

-**Write back** (only write to cache. Write to main memory after an eviction. Need dirty bit) vs **Write through** (always write to both cache and main memory)

<u>**Data Not in Cache**</u>

-**Write allocate** (if store address not in cache, we copy stored value to the cache) vs **Write No Allocate** (bypass cache and just store in main memory)

-<u>Write back paired with write allocate</u> (value expected to be used a lot, so minimize activity between main memory and cache), <u>write through paired with write no allocate</u> (expect cache to not use this value, store in memory)

Reduce Miss Rate

-**Increase block size** (more spatial locality)

  -to accomplish this, reduce the number of lines

  -Downside: more useless data, increased miss penalty

-**Increase associativity**

  -Downside: more ways = slower hit time (diminishing returns)

-**Increase cache size**

  -Downside: slower hit time

Reduce Miss Rate

-**Prefetch** (guess access pattern before it happens)

-reduce compulsory misses, but if prefetcher is inaccurate, could result in cache pollution

-**Victim cache** (add small buffer for addresses with lots of conflicts)

-small overhead for high performance increase

-**Compiler/software level** (reorder accesses to increase locality, combining loops with similar behavior, use tiling in arrays, compiler profiling, etc.)

UCLA

Reduce Miss Penalty

-**Write buffer** (a load-store queue).

-No need to wait for stores (write from CPU to buffer, then buffer writes to memory when CPU has moved on)

-Lower miss penalty for loads (data might be in the buffer)

-Downside: more overhead

-Early restart (don't wait for all blocks to arrive. Forward the requested block as soon as it arrives).

-**Add more levels** (less access to main memory).

-**Sub-blocking**: use a larger block size with sub-blocks. Lower miss rate and penalty (on miss, only need to bring sub block)

Reduce Hit Time

-**Reduce associativity and size**

-Opposite of reducing miss rate

-Parallel lookup

-Access tag and data in parallel (don’t have to wait for data to arrive)

-Speculative load

-If a load after a store, speculate that load doesn’t depend on the store. Fix if a mistake was made.