

**Lecture 32
 Caches II**



Lecturer PSOE Dan Garcia

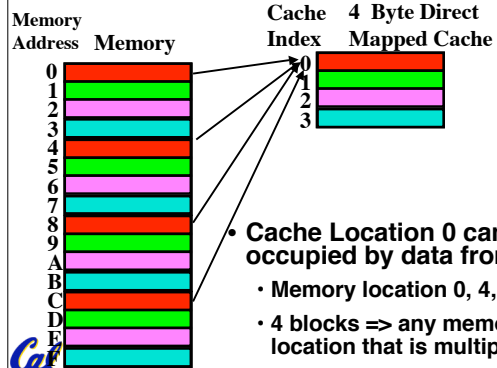
www.cs.berkeley.edu/~ddgarcia

ACM Programming contest =>

Shanghai Jiaotong Univ took top honors this year. The highest-ranked US team was U of Illinois at 17th place. Recall UC Berkeley won in 1996 (there's a plaque on the 3rd floor of Soda Hall). Will US/UCB rule again?



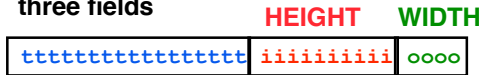
Review: Direct-Mapped Cache



Issues with Direct-Mapped

Tag Index Offset

- Since multiple memory addresses map to same cache index, how do we tell which one is in there?
- What if we have a block size > 1 byte?
- Answer: divide memory address into three fields



Direct-Mapped Cache Terminology

- All fields are read as unsigned integers.
- **Index**: specifies the cache index (which "row" of the cache we should look in)
- **Offset**: once we've found correct block, specifies which byte within the block we want -- i.e., which "column"
- **Tag**: the remaining bits after offset and index are determined; these are used to distinguish between all the memory addresses that map to the same location

Direct-Mapped Cache Example (1/3)

- Suppose we have a 16KB of data in a direct-mapped cache with 4 word blocks
- Determine the size of the tag, index and offset fields if we're using a 32-bit architecture
- **Offset**
 - need to specify correct byte within a block
 - block contains 4 words
 - = 16 bytes
 - = 2⁴ bytes
 - need **4 bits** to specify correct byte

Direct-Mapped Cache Example (2/3)

- **Index**: (~index into an "array of blocks")
 - need to specify correct row in cache
 - cache contains 16 KB = 2¹⁴ bytes
 - block contains 2⁴ bytes (4 words)
 - # blocks/cache
 - = $\frac{\text{bytes/cache}}{\text{bytes/block}}$
 - = $\frac{2^{14} \text{ bytes/cache}}{2^4 \text{ bytes/block}}$
 - = 2¹⁰ blocks/cache
 - need **10 bits** to specify this many rows

Direct-Mapped Cache Example (3/3)

- **Tag:** use remaining bits as tag
 - tag length = addr length - offset - index
= 32 - 4 - 10 bits
= 18 bits
 - so tag is leftmost **18 bits** of memory address
- Why not full 32 bit address as tag?
 - All bytes within block need same address (4b)
 - Index must be same for every address within a block, so its redundant in tag check, thus can leave off to save memory (10 bits in this example)

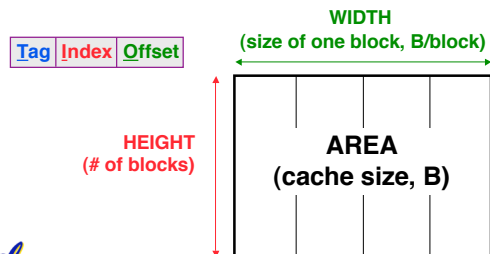


CS61C L32 Caches II (7)

Garcia, 2005 © UCB

TIO Dan's great cache mnemonic

AREA (cache size, B) = $2^{(H+W)} = 2^H * 2^W$
 = **HEIGHT (# of blocks)**
 * **WIDTH (size of one block, B/block)**



CS61C L32 Caches II (8)

Garcia, 2005 © UCB

Caching Terminology

- When we try to read memory, 3 things can happen:
 1. **cache hit:** cache block is valid and contains proper address, so read desired word
 2. **cache miss:** nothing in cache in appropriate block, so fetch from memory
 3. **cache miss, block replacement:** wrong data is in cache at appropriate block, so discard it and fetch desired data from memory (cache always copy)



CS61C L32 Caches II (9)

Garcia, 2005 © UCB

Accessing data in a direct mapped cache

- Ex.: 16KB of data, direct-mapped, 4 word blocks

- Read 4 addresses

1. 0x00000014
2. 0x0000001C
3. 0x00000034
4. 0x00008014

- Memory values on right:
 - only cache/memory level of hierarchy

Address (hex)	Value of Word
...	...
00000010	a
00000014	b
00000018	c
0000001C	d
...	...
00000030	e
00000034	f
00000038	g
0000003C	h
...	...
00008010	i
00008014	j
00008018	k
0000801C	l
...	...



CS61C L32 Caches II (10)

Garcia, 2005 © UCB

Accessing data in a direct mapped cache

- 4 Addresses:
 - 0x00000014, 0x0000001C, 0x00000034, 0x00008014
- 4 Addresses divided (for convenience) into Tag, Index, Byte Offset fields

000000000000000000	0000000001	0100
000000000000000000	0000000001	1100
000000000000000000	0000000011	0100
000000000000000010	0000000001	0100
Tag	Index	Offset



CS61C L32 Caches II (11)

Garcia, 2005 © UCB

16 KB Direct Mapped Cache, 16B blocks

- **Valid bit:** determines whether anything is stored in that row (when computer initially turned on, all entries invalid)

Index	Tag	0x0-3	0x4-7	0x8-b	0xc-f
0	0				
1	0				
2	0				
3	0				
4	0				
5	0				
6	0				
7	0				
...					
1022	0				
1023	0				



CS61C L32 Caches II (12)

Garcia, 2005 © UCB

1. Read 0x00000014

• 00000000000000000000 000000001 0100
 Valid Tag field Index field Offset

Index	Valid	Tag	0x0-3	0x4-7	0x8-b	0xc-f
0	0					
1	0					
2	0					
3	0					
4	0					
5	0					
6	0					
7	0					
...						
1022	0					
1023	0					



CS61C L3Z Caches II (13)

Garcia, 2005 © UCB

So we read block 1 (000000001)

• 00000000000000000000 000000001 0100
 Valid Tag field Index field Offset

Index	Valid	Tag	0x0-3	0x4-7	0x8-b	0xc-f
0	0					
1	1					
2	0					
3	0					
4	0					
5	0					
6	0					
7	0					
...						
1022	0					
1023	0					



CS61C L3Z Caches II (14)

Garcia, 2005 © UCB

No valid data

• 00000000000000000000 000000001 0100
 Valid Tag field Index field Offset

Index	Valid	Tag	0x0-3	0x4-7	0x8-b	0xc-f
0	0					
1	0					
2	0					
3	0					
4	0					
5	0					
6	0					
7	0					
...						
1022	0					
1023	0					



CS61C L3Z Caches II (15)

Garcia, 2005 © UCB

So load that data into cache, setting tag, valid

• 00000000000000000000 000000001 0100
 Valid Tag field Index field Offset

Index	Valid	Tag	0x0-3	0x4-7	0x8-b	0xc-f
0	0					
1	1	0	a	b	c	d
2	0					
3	0					
4	0					
5	0					
6	0					
7	0					
...						
1022	0					
1023	0					



CS61C L3Z Caches II (16)

Garcia, 2005 © UCB

Read from cache at offset, return word b

• 00000000000000000000 000000001 0100
 Valid Tag field Index field Offset

Index	Valid	Tag	0x0-3	0x4-7	0x8-b	0xc-f
0	0					
1	1	0	a	b	c	d
2	0					
3	0					
4	0					
5	0					
6	0					
7	0					
...						
1022	0					
1023	0					



CS61C L3Z Caches II (17)

Garcia, 2005 © UCB

2. Read 0x0000001C = 0...00 0..001 1100

• 00000000000000000000 000000001 1100
 Valid Tag field Index field Offset

Index	Valid	Tag	0x0-3	0x4-7	0x8-b	0xc-f
0	0					
1	1	0	a	b	c	d
2	0					
3	0					
4	0					
5	0					
6	0					
7	0					
...						
1022	0					
1023	0					



CS61C L3Z Caches II (18)

Garcia, 2005 © UCB

Index is Valid

- 00000000000000000000 000000001 1100

	Valid	Tag field	Index field	Offset		
Index	Tag	0x0-3	0x4-7	0x8-b	0xc-f	
0	0					
1	1	0	a	b	c	d
2	0					
3	0					
4	0					
5	0					
6	0					
7	0					
...						
1022	0					
1023	0					



CS61C L3Z Caches II (19)

Garcia, 2005 © UCB

Index valid, Tag Matches

- 00000000000000000000 000000001 1100

	Valid	Tag field	Index field	Offset		
Index	Tag	0x0-3	0x4-7	0x8-b	0xc-f	
0	0					
1	1	0	a	b	c	d
2	0					
3	0					
4	0					
5	0					
6	0					
7	0					
...						
1022	0					
1023	0					



CS61C L3Z Caches II (20)

Garcia, 2005 © UCB

Index Valid, Tag Matches, return d

- 00000000000000000000 000000001 1100

	Valid	Tag field	Index field	Offset		
Index	Tag	0x0-3	0x4-7	0x8-b	0xc-f	
0	0					
1	1	0	a	b	c	d
2	0					
3	0					
4	0					
5	0					
6	0					
7	0					
...						
1022	0					
1023	0					



CS61C L3Z Caches II (21)

Garcia, 2005 © UCB

3. Read 0x00000034 = 0...00 0.011 0100

- 00000000000000000000 000000011 0100

	Valid	Tag field	Index field	Offset		
Index	Tag	0x0-3	0x4-7	0x8-b	0xc-f	
0	0					
1	1	0	a	b	c	d
2	0					
3	0					
4	0					
5	0					
6	0					
7	0					
...						
1022	0					
1023	0					



CS61C L3Z Caches II (22)

Garcia, 2005 © UCB

So read block 3

- 00000000000000000000 000000011 0100

	Valid	Tag field	Index field	Offset		
Index	Tag	0x0-3	0x4-7	0x8-b	0xc-f	
0	0					
1	1	0	a	b	c	d
2	0					
3	0					
4	0					
5	0					
6	0					
7	0					
...						
1022	0					
1023	0					



CS61C L3Z Caches II (23)

Garcia, 2005 © UCB

No valid data

- 00000000000000000000 000000011 0100

	Valid	Tag field	Index field	Offset		
Index	Tag	0x0-3	0x4-7	0x8-b	0xc-f	
0	0					
1	1	0	a	b	c	d
2	0					
3	0					
4	0					
5	0					
6	0					
7	0					
...						
1022	0					
1023	0					



CS61C L3Z Caches II (24)

Garcia, 2005 © UCB

Load that cache block, return word f

- 000000000000000000000000 0000000011 0100

Valid Tag field Index field Offset

Index	Valid	Tag	0x0-3	0x4-7	0x8-b	0xc-f
0	0					
1	1	0	a	b	c	d
2	0					
3	1	0	e	f	g	h
4	0					
5	0					
6	0					
7	0					
...						
1022	0					
1023	0					

CS61C L3Z Caches II (25) Garcia, 2005 © UCB

4. Read 0x00008014 = 0...10 0..001 0100

- 000000000000000000000010 0000000001 0100

Valid Tag field Index field Offset

Index	Valid	Tag	0x0-3	0x4-7	0x8-b	0xc-f
0	0					
1	1	0	a	b	c	d
2	0					
3	1	0	e	f	g	h
4	0					
5	0					
6	0					
7	0					
...						
1022	0					
1023	0					

CS61C L3Z Caches II (26) Garcia, 2005 © UCB

So read Cache Block 1, Data is Valid

- 000000000000000000000010 0000000001 0100

Valid Tag field Index field Offset

Index	Valid	Tag	0x0-3	0x4-7	0x8-b	0xc-f
0	0					
1	1	0	a	b	c	d
2	0					
3	1	0	e	f	g	h
4	0					
5	0					
6	0					
7	0					
...						
1022	0					
1023	0					

CS61C L3Z Caches II (27) Garcia, 2005 © UCB

Cache Block 1 Tag does not match (0 != 2)

- 000000000000000000000010 0000000001 0100

Valid Tag field Index field Offset

Index	Valid	Tag	0x0-3	0x4-7	0x8-b	0xc-f
0	0					
1	1	0	a	b	c	d
2	0					
3	1	0	e	f	g	h
4	0					
5	0					
6	0					
7	0					
...						
1022	0					
1023	0					

CS61C L3Z Caches II (28) Garcia, 2005 © UCB

Miss, so replace block 1 with new data & tag

- 000000000000000000000010 0000000001 0100

Valid Tag field Index field Offset

Index	Valid	Tag	0x0-3	0x4-7	0x8-b	0xc-f
0	0					
1	1	2	i	j	k	l
2	0					
3	1	0	e	f	g	h
4	0					
5	0					
6	0					
7	0					
...						
1022	0					
1023	0					

CS61C L3Z Caches II (29) Garcia, 2005 © UCB

And return word j

- 000000000000000000000010 0000000001 0100

Valid Tag field Index field Offset

Index	Valid	Tag	0x0-3	0x4-7	0x8-b	0xc-f
0	0					
1	1	2	i	j	k	l
2	0					
3	1	0	e	f	g	h
4	0					
5	0					
6	0					
7	0					
...						
1022	0					
1023	0					

CS61C L3Z Caches II (30) Garcia, 2005 © UCB

Do an example yourself. What happens?

- Chose from: Cache: Hit, Miss, Miss w. replace
Values returned: a, b, c, d, e, ..., k, l
- Read address 0x00000030 ?
000000000000000000 0000000011 0000
- Read address 0x0000001c ?
000000000000000000 0000000001 1100

Valid	Tag	0x0-3	0x4-7	0x8-b	0xc-f
0	0				
1	2	i	i	k	l
2	0				
3	0	e	f	g	h
4	0				
5	0				
6	0				
7	0				

Administrivia

- Remember I'm moving my OH on Wed, Thu to Monday (2 days ago) and Friday 11am-noon

Answers

- 0x00000030 a **hit**
Index = 3, Tag matches, Offset = 0, value = e
- 0x0000001c a **miss**
Index = 1, Tag mismatch, so replace from memory, Offset = 0xc, value = d
- Since reads, values must = memory values whether or not cached:
 - 0x00000030 = e
 - 0x0000001c = d

Memory Address	Value of Word
00000010	a
00000014	b
00000018	c
0000001c	d
...	...
00000030	e
00000034	f
00000038	g
0000003c	h
...	...
00008010	i
00008014	j
00008018	k
0000801c	l

Peer Instruction

- Mem hierarchies were invented before 1950. (UNIVAC I wasn't delivered 'til 1951)
- If you know your computer's cache size, you can often make your code run faster.
- Memory hierarchies take advantage of spatial locality by keeping the most recent data items closer to the processor.

ABC
1: FFF
2: FFT
3: FTF
4: FTT
5: TFF
6: TFT
7: TTF
8: TTT

Peer Instructions

- All caches take advantage of spatial locality.
- All caches take advantage of temporal locality.
- On a read, the return value will depend on what is in the cache.

ABC
1: FFF
2: FFT
3: FTF
4: FTT
5: TFF
6: TFT
7: TTF
8: TTT

And in Conclusion...

- Mechanism for transparent movement of data among levels of a storage hierarchy
 - set of address/value bindings
 - address \Rightarrow index to set of candidates
 - compare desired address with tag
 - service hit or miss
 - load new block and binding on miss

