

Lecture #20: Caches 2

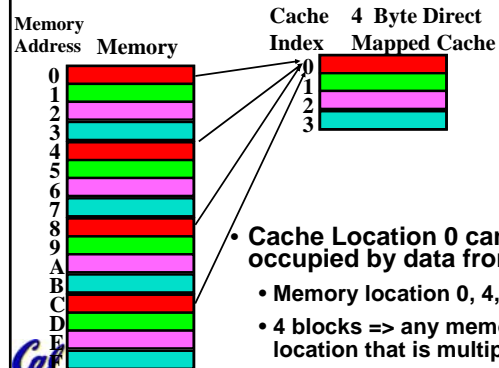


2005-07-26

Andy Carle



Review: Direct-Mapped Cache



• Cache Location 0 can be occupied by data from:

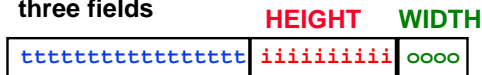
- Memory location 0, 4, 8, ...
- 4 blocks => any memory location that is multiple of 4



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



tag to check if have correct block

index to select block

byte offset within block



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)



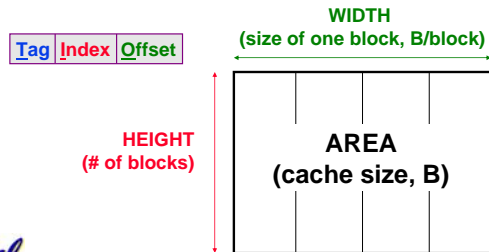
CS61C L32 Caches II (7)

A. Carls, Summer 2005 © UC Berkeley

TIO

$$\text{AREA (cache size, B)} = \text{HEIGHT (\# of blocks)} * \text{WIDTH (size of one block, B/block)}$$

$$2^{(H+W)} = 2^H * 2^W$$



CS61C L32 Caches II (8)

A. Carls, Summer 2005 © UC Berkeley

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)

A. Carls, Summer 2005 © UC Berkeley

Accessing data in a direct mapped cache

- Ex.: 16KB of data, direct-mapped, 4 word blocks
 - Read 4 addresses
- | 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 |
| ... | ... |
- Memory values on right:
 - only cache/memory level of hierarchy



CS61C L32 Caches II (10)

A. Carls, Summer 2005 © UC Berkeley

Accessing data in a direct mapped cache

- 4 Addresses:
 - 0x00000014, 0x0000001C, 0x00000034, 0x00008014
 - 4 Addresses divided (for convenience) into Tag, Index, Byte Offset fields
- | Tag | Index | Offset |
|--------------------|------------|--------|
| 000000000000000000 | 0000000001 | 0100 |
| 000000000000000000 | 0000000001 | 1100 |
| 000000000000000000 | 0000000011 | 0100 |
| 000000000000000010 | 0000000001 | 0100 |



CS61C L32 Caches II (11)

A. Carls, Summer 2005 © UC Berkeley

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					
3	0				
4	0				
5	0				
6	0				
7	0				
...					
1022	0				
1023	0				



CS61C L32 Caches II (12)

A. Carls, Summer 2005 © UC Berkeley

1. Read 0x00000014

- 00000000000000000000 0000000001 0100

Valid Tag field Index field Offset

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				

Cal

CS61C L32 Caches II (13) A. Carls, Summer 2005 © UC Berkeley

So we read block 1 (0000000001)

- 00000000000000000000 0000000001 0100

Valid Tag field Index field Offset

Index	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				

Cal

CS61C L32 Caches II (14) A. Carls, Summer 2005 © UC Berkeley

No valid data

- 00000000000000000000 0000000001 0100

Valid Tag field Index field Offset

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				

Cal

CS61C L32 Caches II (15) A. Carls, Summer 2005 © UC Berkeley

So load that data into cache, setting tag, valid

- 00000000000000000000 0000000001 0100

Valid Tag field Index field Offset

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

Cal

CS61C L32 Caches II (16) A. Carls, Summer 2005 © UC Berkeley

Read from cache at offset, return word b

- 00000000000000000000 0000000001 0100

Valid Tag field Index field Offset

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

Cal

CS61C L32 Caches II (17) A. Carls, Summer 2005 © UC Berkeley

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

- 00000000000000000000 0000000001 1100

Valid Tag field Index field Offset

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

Cal

CS61C L32 Caches II (18) A. Carls, Summer 2005 © UC Berkeley

Index is Valid

- 00000000000000000000 0000000001 1100

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

Cal

CS61C L32 Caches II (19) A. Carls, Summer 2005 © UCB

Index valid, Tag Matches

- 00000000000000000000 0000000001 1100

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

Cal

CS61C L32 Caches II (20) A. Carls, Summer 2005 © UCB

Index Valid, Tag Matches, return d

- 00000000000000000000 0000000001 1100

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

Cal

CS61C L32 Caches II (21) A. Carls, Summer 2005 © UCB

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

- 00000000000000000000 0000000011 0100

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

Cal

CS61C L32 Caches II (22) A. Carls, Summer 2005 © UCB

So read block 3

- 00000000000000000000 0000000011 0100

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

Cal

CS61C L32 Caches II (23) A. Carls, Summer 2005 © UCB

No valid data

- 00000000000000000000 0000000011 0100

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

Cal

CS61C L32 Caches II (24) A. Carls, Summer 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					

Cal

CS61C L32 Caches II (25) A. Carle, Summer 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					

Cal

CS61C L32 Caches II (26) A. Carle, Summer 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					

Cal

CS61C L32 Caches II (27) A. Carle, Summer 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					

Cal

CS61C L32 Caches II (28) A. Carle, Summer 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					

Cal

CS61C L32 Caches II (29) A. Carle, Summer 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					

Cal

CS61C L32 Caches II (30) A. Carle, Summer 2005 © UCB

Administrivia

- HW6 Due Tonight
- HW7 Out Today
- Midterm 2:
 - Friday, 11:00am – 2:00pm
 - 306 Soda (HP Auditorium)
 - Conflicts, DSP, &&|| terrified about the drop deadline: Contact Andy ASAP



CS61C L32 Caches II (34) A. Carle, Summer 2005 © UCB

Big Endian vs. Little Endian

Big-endian and little-endian derive from Jonathan Swift's *Gulliver's Travels* in which the Big Endians were a political faction that broke their eggs at the large end ("the primitive way") and rebelled against the Lilliputian King who required his subjects (the Little Endians) to break their eggs at the small end.

- The order in which **BYTES** are stored in memory
- Bits always stored as usual. (E.g., 0xC2=0b 1100 0010)

Consider the number 1025 as we normally write it:

BYTE3 BYTE2 BYTE1 BYTE0
00000000 00000000 00000100 00000001

Big Endian				Little Endian					
•	ADDR3	ADDR2	ADDR1	ADDR0	•	ADDR3	ADDR2	ADDR1	ADDR0
	BYTE0	BYTE1	BYTE2	BYTE3		BYTE3	BYTE2	BYTE1	BYTE0
	00000001	00000100	00000000	00000000		00000000	00000000	00000100	00000001
•	ADDR0	ADDR1	ADDR2	ADDR3	•	ADDR0	ADDR1	ADDR2	ADDR3
	BYTE3	BYTE2	BYTE1	BYTE0		BYTE0	BYTE1	BYTE2	BYTE3
	00000000	00000000	00000100	00000001		00000001	00000100	00000000	00000000


www.webopedia.com/TERM/b/big_endian.html
 searchnetworking.techtarget.com/sDefinition/0,,sid7_gci211659,00.html
 www.novelltheory.com/TechPapers/Endian.asp
 en.wikipedia.org/wiki/Big_endian

Memorized this table yet?

Blah blah Cache size 16KB blah blah 2²³ blocks blah blah how many bits?

Answer! 2^{XY} means...

X=0 ⇒ no suffix	Y=0 ⇒ 1
X=1 ⇒ kibi ~ Kilo 10 ³	Y=1 ⇒ 2
X=2 ⇒ mebi ~ Mega 10 ⁶	Y=2 ⇒ 4
X=3 ⇒ gibi ~ Giga 10 ⁹	Y=3 ⇒ 8
X=4 ⇒ tebi ~ Tera 10 ¹²	Y=4 ⇒ 16
X=5 ⇒ pebi ~ Peta 10 ¹⁵	Y=5 ⇒ 32
X=6 ⇒ exbi ~ Exa 10 ¹⁸	Y=6 ⇒ 64
X=7 ⇒ zebi ~ Zetta 10 ²¹	Y=7 ⇒ 128
X=8 ⇒ yobi ~ Yotta 10 ²⁴	Y=8 ⇒ 256
	Y=9 ⇒ 512




CS61C L32 Caches II (36) A. Carle, Summer 2005 © UCB

How Much Information IS that?

www.sims.berkeley.edu/research/projects/how-much-info-2003/


- Print, film, magnetic, and optical storage media produced about **5 exabytes** of new information in 2002. 92% of the new information stored on magnetic media, mostly in hard disks.
- Amt of new information stored on paper, film, magnetic, & optical media **~doubled in last 3 yrs**
- Information flows through electronic channels -- telephone, radio, TV, and the Internet -- contained **~18 exabytes** of new information in 2002, 3.5x more than is recorded in storage media. **98% of this total is the information sent & received in telephone calls** - incl. voice & data on fixed lines & wireless.
 - WWW ⇒ 170 Tb of information on its surface; in volume 17x the size of the Lib. of Congress print collections.
 - Instant messaging ⇒ 5x10⁹ msgs/day (750GB), 274 TB/yr.
 - Email ⇒ ~400 PB of new information/year worldwide.



CS61C L32 Caches II (37) A. Carle, Summer 2005 © UCB

Block Size Tradeoff (1/3)


- Benefits of Larger Block Size
 - **Spatial Locality**: if we access a given word, we're likely to access other nearby words soon
 - Very applicable with Stored-Program Concept: if we execute a given instruction, it's likely that we'll execute the next few as well
 - Works nicely in sequential array accesses too



CS61C L32 Caches II (38) A. Carle, Summer 2005 © UCB

Block Size Tradeoff (2/3)

- Drawbacks of Larger Block Size
 - Larger block size means **larger miss penalty**
 - on a miss, takes longer time to load a new block from next level
 - If block size is too big relative to cache size, then there are too few blocks
 - Result: miss rate goes up
- In general, minimize **Average Memory Access Time (AMAT)**
 - = Hit Time
 - + Miss Penalty x Miss Rate



CS61C L32 Caches II (39) A. Carle, Summer 2005 © UCB

Block Size Tradeoff (3/3)

- **Hit Time** = time to find and retrieve data from current level cache
- **Miss Penalty** = average time to retrieve data on a current level miss (includes the possibility of misses on successive levels of memory hierarchy)
- **Hit Rate** = % of requests that are found in current level cache
- **Miss Rate** = $1 - \text{Hit Rate}$



CS61C L32 Caches II (40)

A. Carle, Summer 2005 © UCB

Extreme Example: One Big Block

Valid Bit Tag Cache Data
 B3|B2|B1|B0

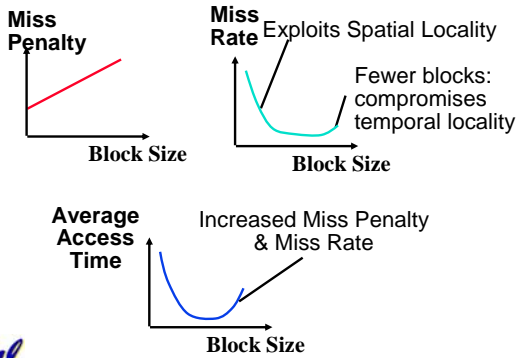
- Cache Size = 4 bytes Block Size = 4 bytes
 - Only **ONE** entry in the cache!
- If item accessed, likely accessed again soon
 - But unlikely will be accessed again immediately!
- The next access will likely to be a miss again
 - Continually loading data into the cache but discard data (force out) before use it again
 - Nightmare for cache designer: **Ping Pong Effect**



CS61C L32 Caches II (41)

A. Carle, Summer 2005 © UCB

Block Size Tradeoff Conclusions



CS61C L32 Caches II (42)

A. Carle, Summer 2005 © UCB

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

address: tag index offset
 00000000000000000000 0000000001 1100

Valid

Valid	Tag	0x0-3	0x4-7	0x8-b	0xc-f
1	0	a	b	c	d



CS61C L32 Caches II (45)

A. Carle, Summer 2005 © UCB