inst.eecs.berkeley.edu/~cs61c

CS61C : Machine Structures

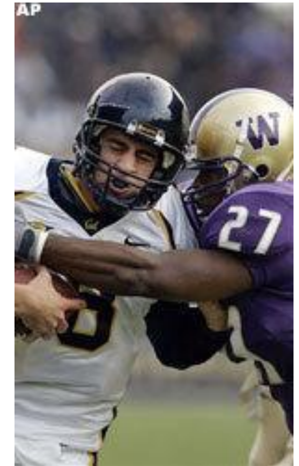
Lecture 33
Caches II

2004-11-15

Lecturer PSOE Dan Garcia

www.cs.berkeley.edu/~ddgarcia

The Incredibles! ⇒



Cal wins AGAIN, upping them to 8-1 with a 42-12 victory over UW. JJ Arrington rushed for 121 yards, and has 100+ yard efforts in all 9 games this year. (The Pac-10 record is 11 by Marcus Allen when he won the Heisman.) The Big Game is next Sat; we're 4th in ALL polls!

calbears.collegesports.com/sports/m-footbl/recaps/111304aaa.html

Review: Direct-Mapped Cache

Memory Address | Memory

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

Cache Index | 4 Byte Direct Mapped Cache

0, 1, 2, 3

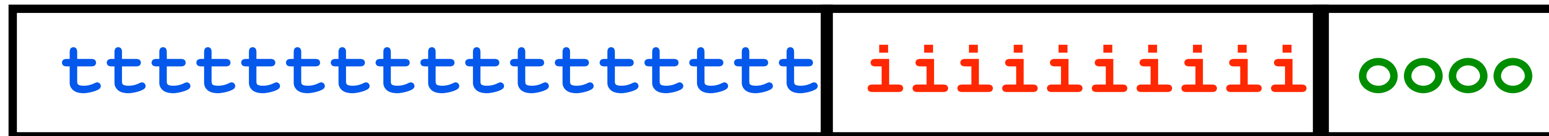
- Cache Location 0 can be occupied by data from:
  - Memory location 0, 4, 8, ...
  - 4 blocks => any memory location that is multiple of 4

Garcia, Fall 2004 © UCB

Issues with Direct-Mapped

- 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

HEIGHT WIDTH



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

    = 2^4 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 = 2^{14} bytes
  - block contains 2^4 bytes (4 words)
  - # blocks/cache

$$= \frac{\text{bytes/cache}}{\text{bytes/block}}$$

$$= \frac{2^{14}\ \text{bytes/cache}}{2^{4}\ \text{bytes/block}}$$

$$= 2^{10}\ \text{blocks/cache}$$

  - need 10 bits to specify this many rows

 Garcia, Fall 2004 © UCB

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)

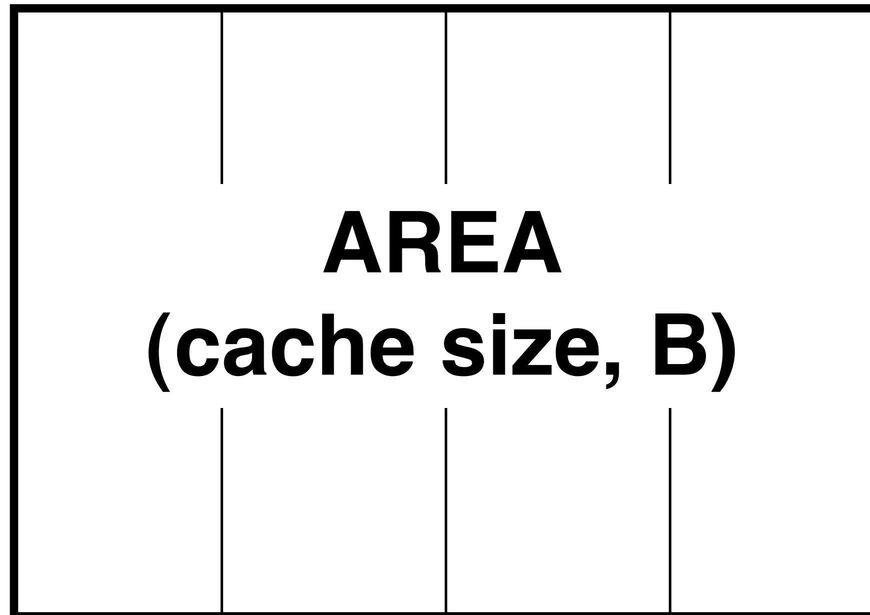
Garcia, Fall 2004 © UCB

TIO Dan's great cache mnemonic

AREA (cache size, B)
= HEIGHT (# of blocks)
* WIDTH (size of one block, B/block)

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

Tag	Index	Offset



Garcia, Fall 2004 © 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)

Garcia, Fall 2004 © 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

Memory

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
...	...

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**

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	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					

1. Read 0x00000014

- 000000000000000000 0000000001 0100

Tag field	Index field	Offset
000000000000000000	0000000001	0100

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					

So we read block 1 (0000000001)

- 000000000000000000 0000000001 0100

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					

No valid data

- 0000000000000000000 0000000001 0100

Tag field	Index field	Offset
0000000000000000000	0000000001	0100

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					

Garcia, Fall 2004 © UCB

So load that data into cache, setting tag, valid

- 000000000000000000 0000000001 0100
  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					

Read from cache at offset, return word b

- 000000000000000000 0000000001 0100

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					

2. Read 0x0000001C = 0…00 0..001 1100

- 000000000000000000 0000000001 1100

  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					

Index is Valid

- 00000000000000000 0000000001 1100

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					

Garcia, Fall 2004 © UCB

Index valid, Tag Matches

- 000000000000000000 0000000001 1100

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					

Index Valid, Tag Matches, return d

- 000000000000000000 0000000001 1100

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					

3. Read 0x00000034 = 0…00 0..011 0100

- 000000000000000000 0000000011 0100

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					

Garcia, Fall 2004 © UCB

So read block 3

- 0000000000000000000 0000000011 0100

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					

No valid data

- 00000000000000000 0000000011 0100

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					

Garcia, Fall 2004 © UCB

Load that cache block, return word f

- 0000000000000000000 0000000011 0100

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					

Garcia, Fall 2004 © UCB

4. Read 0x00008014 = 0…10 0..001 0100

- 000000000000000010 0000000001 0100

	Tag field	Index field	Offset
000000000000000010		0000000001	0100

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					

So read Cache Block 1, Data is Valid

- 00000000000000000010 0000000001 0100

Tag field	Index field	Offset
00000000000000000010	0000000001	0100

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					

Garcia, Fall 2004 © UCB

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

- 00000000000000000010 0000000001 0100

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					

Miss, so replace block 1 with new data & tag

- 00000000000000000010 0000000001 0100

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					

Garcia, Fall 2004 © UCB

And return word j

- `000000000000000000010 0000000001 0100`

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					

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`

Cache

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					
...			...			

Administrivia

Garcia, Fall 2004 © UCB

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

A. Mem hierarchies were invented before 1950. (UNIVAC I wasn't delivered 'til 1951)

B. If you know your computer's cache size, you can often make your code run faster.

C. 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

Garcia, Fall 2004 © UCB

Peer Instructions

1. All caches take advantage of spatial locality.
2. All caches take advantage of temporal locality.
3. 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

Garcia, Fall 2004 © UCB

And in Conclusion…

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