

ELEC2041

Microprocessors and Interfacing

Lectures 32: Cache Memory - II

<http://webct.edtec.unsw.edu.au/>

May 2006

Saeid Nooshabadi

saeid@unsw.edu.au

ELEC2041 lec32-Cache-II.1 Some of the slides are adopted from David Patterson (UCB) Saeid Nooshabadi

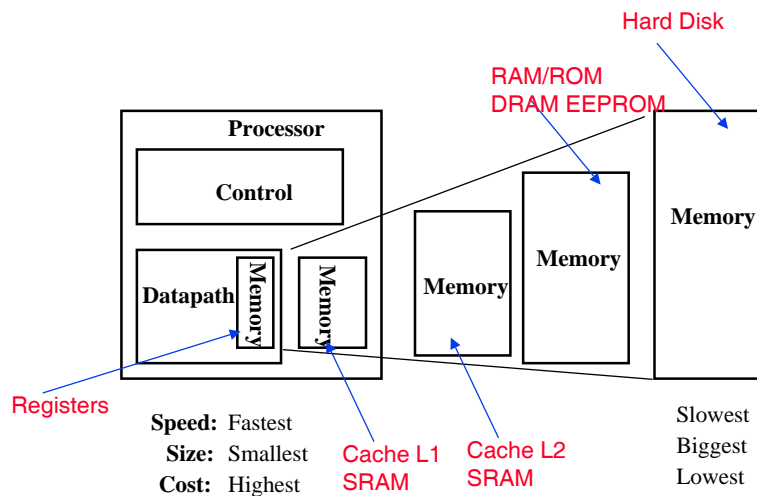
Outline

- Direct-Mapped Cache
- Types of Cache Misses
- A (long) detailed example
- Peer - to - peer education example
- Block Size Tradeoff
- Types of Cache Misses

ELEC2041 lec32-Cache-II.2

Saeid Nooshabadi

Review: Memory Hierarchy



ELEC2041 lec32-Cache-II.3

Saeid Nooshabadi

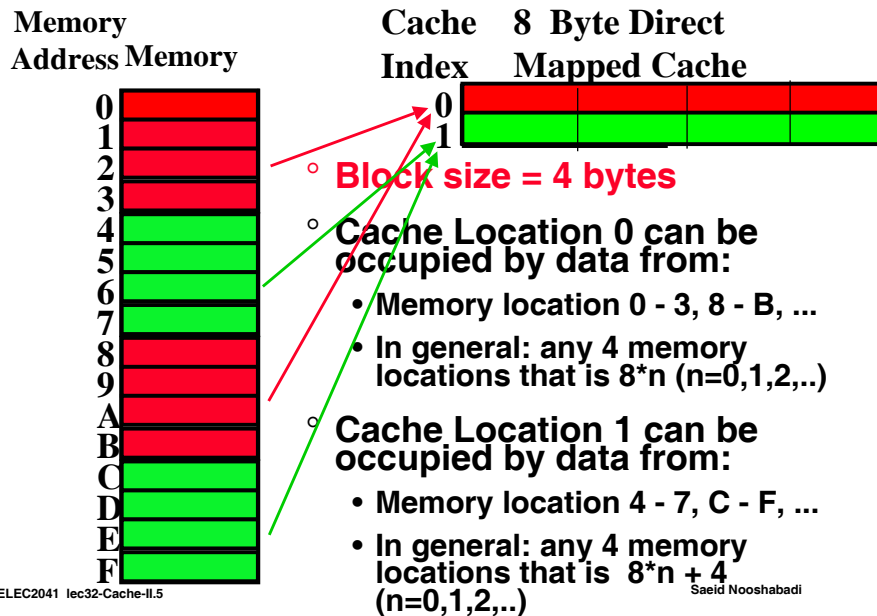
Review: Direct-Mapped Cache

- In a **direct-mapped cache**, each memory address is associated with one possible **block** within the cache
 - Therefore, we only need to look in a single location in the cache for the data if it exists in the cache
 - Block is the unit of transfer between cache and memory

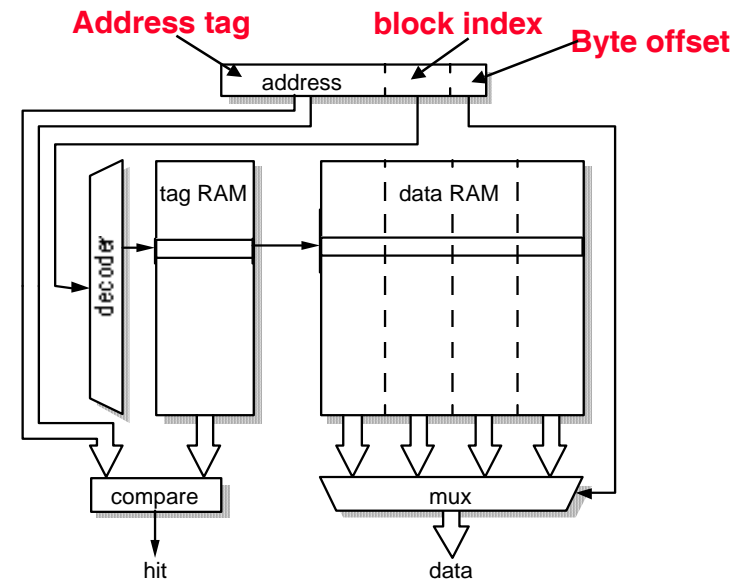
ELEC2041 lec32-Cache-II.4

Saeid Nooshabadi

Review: Direct-Mapped Cache 1 Word Block



Direct-Mapped with 1 word Blocks Example



Review: Direct-Mapped Cache Terminology

- All fields are read as unsigned integers.
- Index:** specifies the cache index (which “row” or “line” of the cache we should look in)
- Offset:** once we’ve found correct block, specifies which byte within the block we want
- 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

Reading Material

- Steve Furber: ARM System On-Chip; 2nd Ed, Addison-Wesley, 2000, ISBN: 0-201-67519-6. [Chapter 10.](#)

Accessing Data in a Direct Mapped Cache (#1/3)

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

- Read 4 addresses

0x00000014,
0x0000001C,
0x00000034,
0x00008014

- Only cache/memory level of hierarchy

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

ELEC2041 lec32-Cache-II.9

Saeid Nooshabadi

Accessing Data in a Direct Mapped Cache (#2/3)

- 4 Addresses:

• 0x00000014, 0x0000001C, 0x00000034, 0x00008014

- 4 Addresses divided (for convenience) into Tag, Index, Byte Offset fields

tttttttttttttttttttt	iiiiiiiiii	oooo
tag to check if have correct block	index to select block	byte offset within block
00000000000000000000	0000000001	0100
00000000000000000000	0000000001	1100
00000000000000000000	0000000011	0100
00000000000000000010	0000000001	0100
Tag	Index	Offset

ELEC2041 lec32-Cache-II.10

Saeid Nooshabadi

Accessing Data in a Direct Mapped Cache (#3/3)

- So lets go through accessing some data in this cache

- 16KB data, direct-mapped, 4 word blocks

- Will see 3 types of events:

- cache miss:** nothing in cache in appropriate block, so fetch from memory

- cache hit:** cache block is valid and contains proper address, so read desired word

- cache miss, block replacement:** wrong data is in cache at appropriate block, so discard it and fetch desired data from memory

ELEC2041 lec32-Cache-II.11

Saeid Nooshabadi

16 KB Direct Mapped Cache, 16B blocks

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

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

ELEC2041 lec32-Cache-II.12

Saeid Nooshabadi

Read 0x00000014 = 0...00 0..001 0100

◦ 00000000000000000000 0000000001 0100

Valid Tag field Index field Offset

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				
0					

So we read block 1 (0000000001)

◦ 00000000000000000000 0000000001 0100

Valid Tag field Index field Offset

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

◦ 00000000000000000000 0000000001 0100

Valid Tag field Index field Offset

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 load that data into cache, setting tag, valid

◦ 00000000000000000000 0000000001 0100

Valid Tag field Index field Offset

Valid	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				

Read from cache at offset, return word b

◦ 00000000000000000000 0000000001 0100
 Valid Tag field Index field Offset

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

...

1022	0				
1023	0				

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

◦ 00000000000000000000 0000000001 1100
 Valid Tag field Index field Offset

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

...

1022	0				
1023	0				

Data valid, tag OK, so read offset return word d

◦ 00000000000000000000 0000000001 1100

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

...

1022	0				
1023	0				

Read 0x00000034 = 0...00 0..011 0100

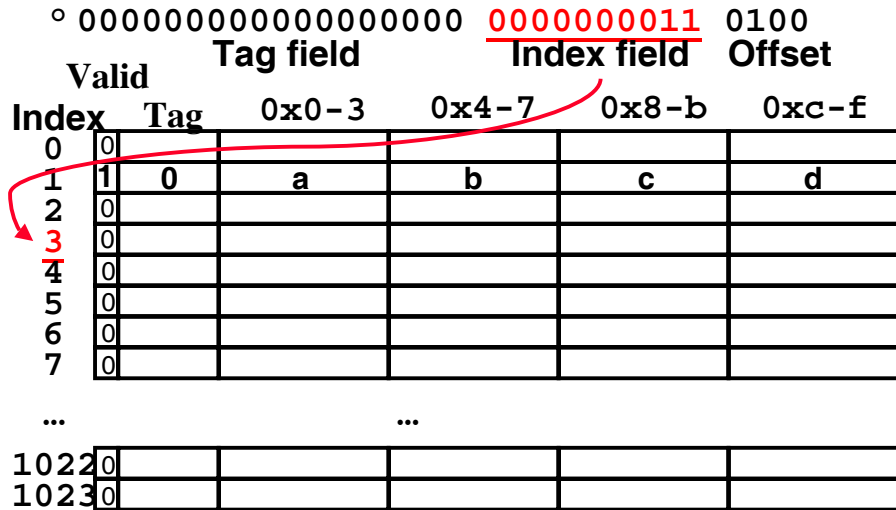
◦ 00000000000000000000 0000000011 0100
 Valid Tag field Index field Offset

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

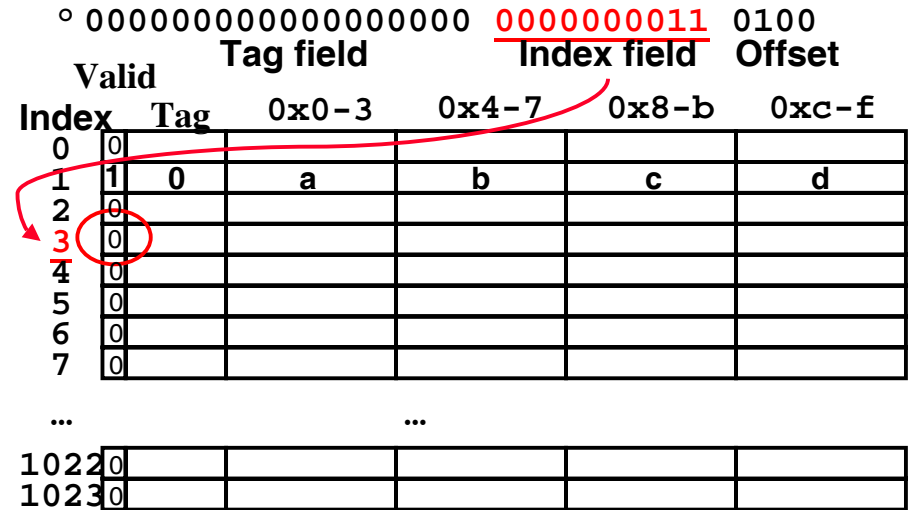
...

1022	0				
1023	0				

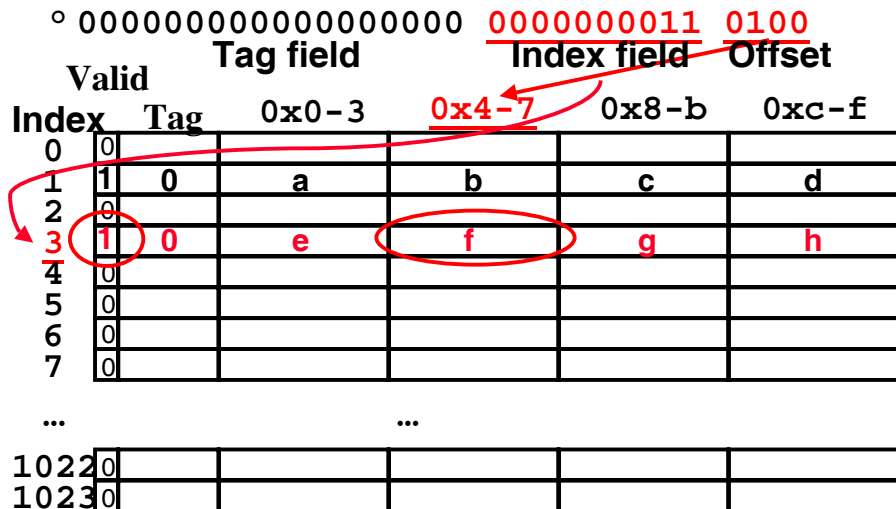
So read block 3



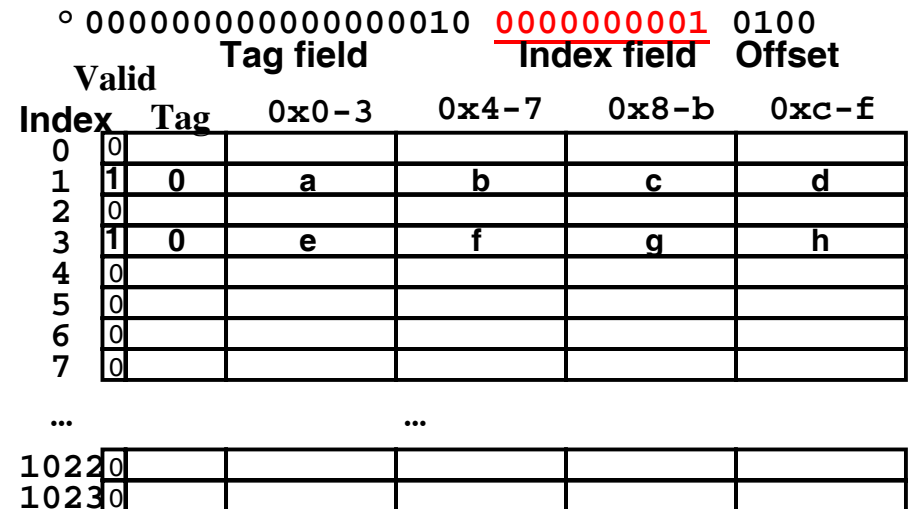
No valid data



Load that cache block, return word f



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



So read Cache Block 1, Data is Valid

◦ 0000000000000000010 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					

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

◦ 0000000000000000010 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					

Miss, so replace block 1 with new data & tag

◦ 0000000000000000010 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					

And return word j

◦ 0000000000000000010 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					

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	i	k	l
2	0					
3	1	0	e	f	g	h
4	0					
5	0					
6	0					
7	0					
...						

ELEC2041 lec32-Cache-II.29

Saeid Nooshabadi

Answers

◦ 0x00000030 a **hit**
Index = 3, Tag matches,
Offset = 0, value = e

◦ 0x0000001c a **miss with replacement**

Index = 1, Tag mismatch,
so replace from memory,
Offset = 0xc, value = d

◦ The Values read from
Cache
must equal memory values
whether or not cached:

• 0x00000030 = e

• 0x0000001c = d

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

ELEC2041 lec32-Cache-II.30

Saeid Nooshabadi

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 (Another Big Idea)
- 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

ELEC2041 lec32-Cache-II.31

Saeid Nooshabadi

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 Access Time

$$= \text{Hit Time} \times \text{Hit Rate} + \text{Miss Penalty} \times \text{Miss Rate}$$

ELEC2041 lec32-Cache-II.32

Saeid Nooshabadi

Block Size Tradeoff (#3/3)

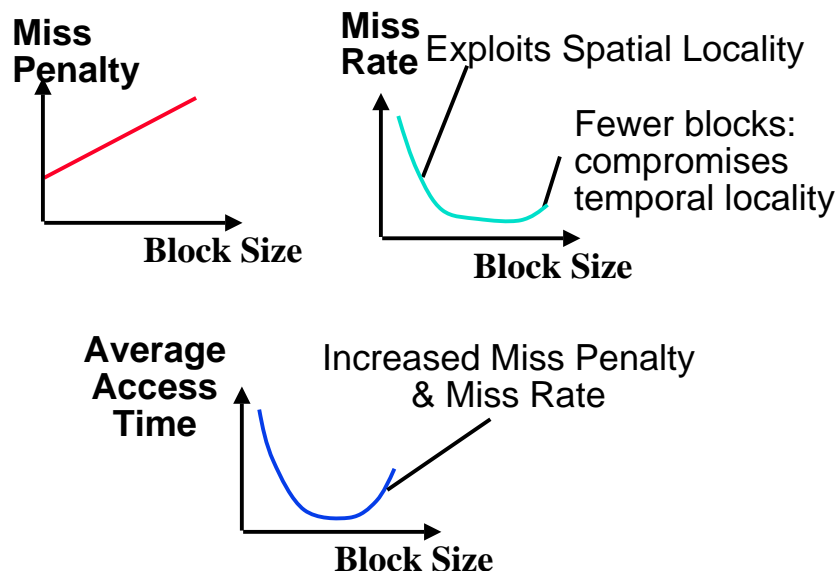
- **Hit Time** = time to find and retrieve data from current level cache
- **Miss Penalty** = average time to retrieve data on a current level miss (includes the possibility of misses on successive levels of memory hierarchy)
- **Hit Rate** = % of requests that are found in current level cache
- **Miss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

Extreme Example: One Big Block

Valid Bit	Tag	Cache Data
<input type="checkbox"/>		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**

Block Size Tradeoff Conclusions



Things to Remember

- Cache Access involves 3 types of events:
 - **cache miss**: nothing in cache in appropriate block, so fetch from memory
 - **cache hit**: cache block is valid and contains proper address, so read desired word
 - **cache miss, block replacement**: wrong data is in cache at appropriate block, so discard it and fetch desired data from memory