

CENG 3420

Computer Organization & Design



Lecture 14: Cache Discussions

Bei Yu

CSE Department, CUHK

byu@cse.cuhk.edu.hk

(Textbook: Chapters 5.3–5.4)

2024 Spring



- ① Example 1
- ② Example 2
- ③ Example 3
- ④ Performance Issues



Example 1



```
short  A[10][4];
int    sum = 0;
int    j, i;
double mean;

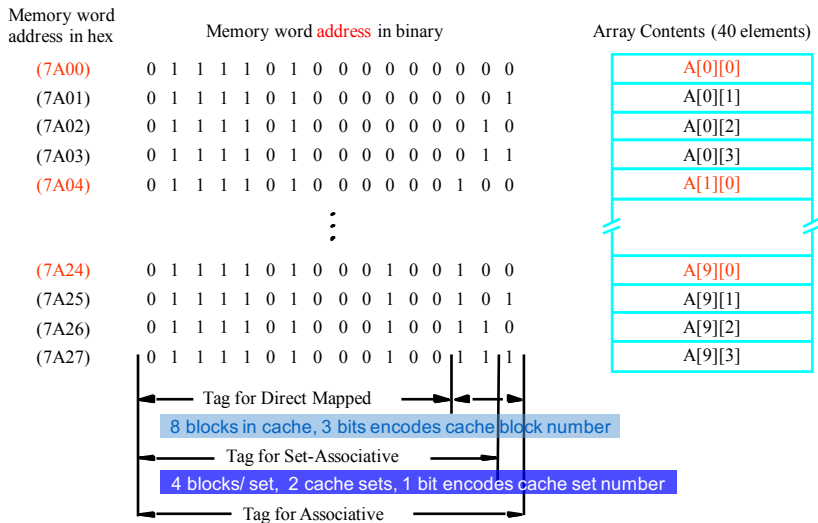
// forward loop
for (j = 0; j <= 9; j++)
    sum += A[j][0];

mean = sum / 10.0;

// backward loop
for (i = 9; i >= 0; i--)
    A[i][0] = A[i][0]/mean;
```

- Assume separate instruction and data caches
- So we consider only the data
- Cache has space for 8 blocks
- A block contains one word (byte)
- `A[10][4]` is an array of words located at 7A00-7A27 in row-major order

Cache Example



To simplify discussion: 16-bit word (byte) address; i.e. 1 word = 1 byte.

- Least significant 3-bits of address determine location
- No replacement algorithm is needed in Direct Mapping
- When $i == 9$ and $i == 8$, get a cache hit (2 hits in total)
- Only 2 out of the 8 cache positions used
- Very inefficient cache utilization

		Content of data cache after loop pass: (time line)																			
		j=0	j=1	j=2	j=3	j=4	j=5	j=6	j=7	j=8	j=9	i=9	i=8	i=7	i=6	i=5	i=4	i=3	i=2	i=1	i=0
Cache Block number	0	A[0][0]	A[0][0]	A[2][0]	A[2][0]	A[4][0]	A[4][0]	A[6][0]	A[6][0]	A[8][0]	A[8][0]	A[8][0]	A[8][0]	A[8][0]	A[6][0]	A[6][0]	A[4][0]	A[4][0]	A[2][0]	A[2][0]	A[0][0]
	1																				
	2																				
	3																				
	4		A[1][0]	A[1][0]	A[3][0]	A[3][0]	A[5][0]	A[5][0]	A[7][0]	A[7][0]	A[9][0]	A[9][0]	A[9][0]	A[9][0]	A[7][0]	A[7][0]	A[5][0]	A[5][0]	A[3][0]	A[3][0]	A[1][0]
	5																				
	6																				
	7																				

Tags not shown but are needed.

- LRU replacement policy: get cache hits for $i = 9, 8, \dots, 2$
- If i loop was a forward one, we would get **no** hits!

		Content of data cache after loop pass: (time line)																			
		j=0	j=1	j=2	j=3	j=4	j=5	j=6	j=7	j=8	j=9	i=9	i=8	i=7	i=6	i=5	i=4	i=3	i=2	i=1	i=0
Cache Block number	0	A[0][0]	A[0][0]	A[0][0]	A[0][0]	A[0][0]	A[0][0]	A[0][0]	A[0][0]	A[8][0]	A[8][0]	A[8][0]	A[8][0]	A[8][0]	A[8][0]	A[8][0]	A[8][0]	A[8][0]	A[8][0]	A[8][0]	A[0][0]
	1		A[1][0]	A[1][0]	A[1][0]	A[1][0]	A[1][0]	A[1][0]	A[1][0]	A[1][0]	A[9][0]	A[9][0]	A[9][0]	A[9][0]	A[9][0]	A[9][0]	A[9][0]	A[9][0]	A[9][0]	A[1][0]	A[1][0]
	2			A[2][0]	A[2][0]	A[2][0]	A[2][0]	A[2][0]	A[2][0]	A[2][0]	A[2][0]	A[2][0]	A[2][0]	A[2][0]	A[2][0]	A[2][0]	A[2][0]	A[2][0]	A[2][0]	A[2][0]	A[2][0]
	3				A[3][0]	A[3][0]	A[3][0]	A[3][0]	A[3][0]	A[3][0]	A[3][0]	A[3][0]	A[3][0]	A[3][0]	A[3][0]	A[3][0]	A[3][0]	A[3][0]	A[3][0]	A[3][0]	A[3][0]
	4					A[4][0]	A[4][0]	A[4][0]	A[4][0]	A[4][0]	A[4][0]	A[4][0]	A[4][0]	A[4][0]	A[4][0]	A[4][0]	A[4][0]	A[4][0]	A[4][0]	A[4][0]	A[4][0]
	5						A[5][0]	A[5][0]	A[5][0]	A[5][0]	A[5][0]	A[5][0]	A[5][0]	A[5][0]	A[5][0]	A[5][0]	A[5][0]	A[5][0]	A[5][0]	A[5][0]	A[5][0]
	6							A[6][0]	A[6][0]	A[6][0]	A[6][0]	A[6][0]	A[6][0]	A[6][0]	A[6][0]	A[6][0]	A[6][0]	A[6][0]	A[6][0]	A[6][0]	A[6][0]
	7								A[7][0]	A[7][0]	A[7][0]	A[7][0]	A[7][0]	A[7][0]	A[7][0]	A[7][0]	A[7][0]	A[7][0]	A[7][0]	A[7][0]	A[7][0]

Tags not shown but are needed; LRU Counters not shown but are needed.

Set Associative Mapping



- Since all accessed blocks have even addresses (7A00, 7A04, 7A08, ...), only half of the cache is used, i.e. they all map to set 0
- LRU replacement policy: get hits for i = 9, 8, 7 and 6
- Random replacement would have better average performance
- If i loop was a forward one, we would get **no** hits!

		Content of data cache after loop pass: (time line)																			
		j=0	j=1	j=2	j=3	j=4	j=5	j=6	j=7	j=8	j=9	i=9	i=8	i=7	i=6	i=5	i=4	i=3	i=2	i=1	i=0
Set 0	0	A[0][0]	A[0][0]	A[0][0]	A[0][0]	A[4][0]	A[4][0]	A[4][0]	A[4][0]	A[8][0]	A[8][0]	A[8][0]	A[8][0]	A[8][0]	A[8][0]	A[8][0]	A[4][0]	A[4][0]	A[4][0]	A[4][0]	A[0][0]
	1		A[1][0]	A[1][0]	A[1][0]	A[1][0]	A[5][0]	A[5][0]	A[5][0]	A[5][0]	A[9][0]	A[9][0]	A[9][0]	A[9][0]	A[9][0]	A[5][0]	A[5][0]	A[5][0]	A[5][0]	A[1][0]	A[1][0]
	2			A[2][0]	A[2][0]	A[2][0]	A[2][0]	A[6][0]	A[6][0]	A[6][0]	A[6][0]	A[6][0]	A[6][0]	A[6][0]	A[6][0]	A[6][0]	A[6][0]	A[6][0]	A[2][0]	A[2][0]	A[2][0]
	3			A[3][0]	A[3][0]	A[3][0]	A[3][0]	A[7][0]	A[7][0]	A[7][0]	A[7][0]	A[7][0]	A[7][0]	A[7][0]	A[7][0]	A[7][0]	A[3][0]	A[3][0]	A[3][0]	A[3][0]	A[3][0]
Set 1	4																				
	5																				
	6																				
	7																				

Tags not shown but are needed; LRU Counters not shown but are needed.



- In this example, Associative is best, then Set-Associative, lastly Direct Mapping.
- What are the advantages and disadvantages of each scheme?
- In practice,
 - Low hit rates like in the example is very rare.
 - Usually **Set-Associative with LRU replacement** scheme is used.
- Larger blocks and more blocks greatly improve cache hit rate, i.e. more cache memory



Example 2



Question:

How many total bits are required for a **direct**-mapped cache with 16 KiB of data and 4-word blocks, assuming a 32-bit address?



Question:

How many total bits are required for a **direct**-mapped cache with 16 KiB of data and 4-word blocks, assuming a 32-bit address?

Answer:

- In a 32-bit address CPU, 16 KiB is 4096 words.
- With a block size of 4 words, there are 1024 blocks.
- Each block has 4×32 or 128 bits of data plus a tag, which is $(32 - 10 - 2 - 2) = 18$ bits, plus a valid bit.
- Thus, the total cache size is $2^{10} \times (4 \times 32 + 18 + 1) = 2^{10} \times 147$ bits.
- the total number of bits in the cache is about $1.15 = \frac{147}{32 \times 4}$ times as many as needed just for the storage of the data.



Question:

How many total bits are required for an **associated**-mapped cache with 16 KiB of data and 4-word blocks, assuming a 32-bit address?



Question:

How many total bits are required for an **associated**-mapped cache with 16 KiB of data and 4-word blocks, assuming a 32-bit address?

Answer:

- In a 32-bit address CPU, 16 KiB is 4096 words.
- With a block size of 4 words, there are 1024 blocks.
- Each block has 4×32 or 128 bits of data plus a tag, which is $(32 - 2 - 2) = 28$ bits, plus a valid bit.
- Thus, the total cache size is $2^{10} \times (4 \times 32 + 28 + 1) = 2^{10} \times 157$ bits.
- the total number of bits in the cache is about $1.27 = \frac{157}{32 \times 4}$ times as many as needed just for the storage of the data.



Example 3

Question



We have designed a 64-bit address direct-mapped cache, and the bits of address used to access the cache are as shown below:

Table: Bits of the address to use in accessing the cache

Tag	Index	Offset
63-10	9-5	4-0

- 1 What is the block size of the cache in words?
- 2 Find the ratio between total bits required for such a cache design implementation over the data storage bits.
- 3 Beginning from power on, the following byte-addressed cache references are recorded as shown below.

Table: Recored byte-addressed cache references

Hex	00	04	10	84	E8	A0	400	1E	8C	C1C	B4	884
Dec	0	4	16	132	232	160	1024	30	140	3100	180	2180

Find the hit ratio.



- ① Each cache block consists of four 8-byte words. The total offset is 5 bits. Three of those 5 bits is the word offset (the offset into an 8-byte word). The remaining two bits are the block offset. Two bits allows us to enumerate $2^2 = 4$ words.
- ② The ratio is 1.21. The cache stores a total of
 $32\text{lines} \times 4\text{words/block} \times 8\text{bytes/word} = 1024\text{bytes} = 8192\text{bits}$. In addition to the data, each line contains 54 tag bits and 1 valid bit. Thus, the total bits required is $8192 + 54 \times 32 + 1 \times 32 = 9952$ bits.
- ③ The hit ratio is $\frac{4}{12} = 33\%$

Byte Address	Binary Address	Tag	Index	Offset	Hit/Miss	Bytes Replaced
0x00	0000 0000 0000	0x0	0x00	0x00	M	
0x04	0000 0000 0100	0x0	0x00	0x04	H	
0x10	0000 0001 0000	0x0	0x00	0x10	H	
0x84	0000 1000 0100	0x0	0x04	0x04	M	
0xe8	0000 1110 1000	0x0	0x07	0x08	M	
0xa0	0000 1010 0000	0x0	0x05	0x00	M	
0x400	0100 0000 0000	0x1	0x00	0x00	M	0x00-0x1F
0x1e	0000 0001 1110	0x0	0x00	0x1e	M	0x400-0x41F
0x8c	0000 1000 1100	0x0	0x04	0x0c	H	
0xc1c	1100 0001 1100	0x3	0x00	0x1c	M	0x00-0x1F
0xb4	0000 1011 0100	0x0	0x05	0x14	H	
0x884	1000 1000 0100	0x2	0x04	0x04	M	0x80-0x9f

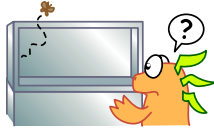


www.onlinedatingmagazine.com

"You say we went out
and I never called?
I can't remember.
My virtual memory
must be low!"



Sometimes, the cache doesn't have the memory block the computer's looking for. When this happens, it's called a cache miss. There are three causes of cache misses. Just remember the three C's:



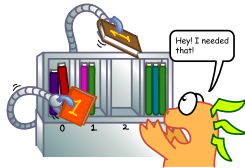
Compulsory

Compulsory misses happen when a block is referenced for the first time. The computer can't get a block that doesn't exist yet!



Capacity

The block is not in the cache because there is no space in the cache for it. Caches are of finite size, after all.



Conflict

These types of misses happen only in direct-mapped and set-associative caches. Multiple blocks can be mapped to a set, forcing evictions when the set is full.



Performance Issues



Q1: Where A Block Be Placed in Upper Level?

Scheme name	# of sets	Blocks per set
Direct mapped	# of blocks	1
Set associative	$\frac{\text{\# of blocks}}{\text{Associativity}}$	Associativity
Fully associative	1	# of blocks



Q1: Where A Block Be Placed in Upper Level?

Scheme name	# of sets	Blocks per set
Direct mapped	# of blocks	1
Set associative	$\frac{\text{\# of blocks}}{\text{Associativity}}$	Associativity
Fully associative	1	# of blocks

Q2: How Is Entry Be Found?

Scheme name	Location method	# of comparisons
Direct mapped	Index	1
Set associative	Index the set; compare set's tags	Degree of associativity
Fully associative	Compare all tags	# of blocks



Q3: Which Entry Should Be Replaced on a Miss?

- **Direct mapped**: only one choice
- **Set associative** or **fully associative**:
 - Random
 - LRU (Least Recently Used)

Note that:

- For a 2-way set associative, random replacement has a miss rate $1.1\times$ than LRU
- For high level associativity (4-way), LRU is too **costly**



Q4: What Happen On A Write?

- Write-Through:

- The information is written in both the block in cache & the block in lower level of memory
- Combined with write buffer, so write waits can be eliminated
- \oplus :
- \oplus :

- Write-Back:

- The information is written only to the block in cache
- The modification is written to lower level, only when the block is replaced
- Need dirty bit: tracks whether the block is clean or not
- Virtual memory always use write-back
- \oplus :
- \oplus :



Q4: What Happen On A Write?

- Write-Through:

- The information is written in both the block in cache & the block in lower level of memory
- Combined with write buffer, so write waits can be eliminated
- \oplus : read misses don't result in writes
- \oplus : easier to implement

- Write-Back:

- The information is written only to the block in cache
- The modification is written to lower level, only when the block is replaced
- Need dirty bit: tracks whether the block is clean or not
- Virtual memory always use write-back
- \oplus :
- \oplus :



Q4: What Happen On A Write?

- Write-Through:

- The information is written in both the block in cache & the block in lower level of memory
- Combined with write buffer, so write waits can be eliminated
- \oplus : read misses don't result in writes
- \oplus : easier to implement

- Write-Back:

- The information is written only to the block in cache
- The modification is written to lower level, only when the block is replaced
- Need dirty bit: tracks whether the block is clean or not
- Virtual memory always use write-back
- \oplus : write with speed of cache
- \oplus : repeated writes require only one write to lower level



Performance

How **fast** machine instructions can be brought into the processor and how **fast** they can be executed.

- Two key factors are **performance** and **cost**, i.e., **price/performance ratio**.
- For a hierarchical memory system with cache, the processor is able to access instructions and data more quickly when the data wanted are in the cache.
- Therefore, the impact of a cache on performance is dependent on the **hit and miss rates**.



- High hit rates over 0.9 are essential for **high-performance** computers.
- A penalty is incurred because extra time is needed to bring a block of data from a slower unit to a faster one in the hierarchy.
- During that time, the processor is **stalled**.
- The waiting time depends on the details of the cache operation.

Miss Penalty

Total access time seen by the processor when a **miss** occurs.



Example: Consider a computer with the following parameters:

Access times to the cache and the main memory are t and $10t$ respectively. When a cache miss occurs, a block of 8 words will be transferred from the MM to the cache. It takes $10t$ to transfer the first word of the block and the remaining 7 words are transferred at a rate of one word per t seconds.

- Miss penalty = $t + 10t + 7 \times t + t$
- First t : Initial cache access that results in a miss.
- Last t : Move data from the cache to the processor.



Average Memory Access Time

$$h \times C + (1 - h) \times M$$

- h : hit rate
 - M : miss penalty
 - C : cache access time
-
- High cache hit rates ($> 90\%$) are essential
 - Miss penalty must also be reduced



Question: Memory Access Time Example

- Assume 8 cycles to read a single memory word;
- 15 cycles to load a 8-word block from main memory (previous example);
- cache access time = 1 cycle
- For every 100 instructions, statistically 30 instructions are data read/ write
- Instruction fetch: 100 memory access: assume hit rate = 0.95
- Data read/ write: 30 memory access: assume hit rate = 0.90

Calculate: (1) Execution cycles without cache; (2) Execution cycles with cache.



Question: Memory Access Time Example

- Assume 8 cycles to read a single memory word;
- 15 cycles to load a 8-word block from main memory (previous example);
- cache access time = 1 cycle
- For every 100 instructions, statistically 30 instructions are data read/ write
- Instruction fetch: 100 memory access: assume hit rate = 0.95
- Data read/ write: 30 memory access: assume hit rate = 0.90

Calculate: (1) Execution cycles without cache; (2) Execution cycles with cache.

- Cycle# w/o. cache: $(100 + 30) \times 8$



Question: Memory Access Time Example

- Assume 8 cycles to read a single memory word;
- 15 cycles to load a 8-word block from main memory (previous example);
- cache access time = 1 cycle
- For every 100 instructions, statistically 30 instructions are data read/ write
- Instruction fetch: 100 memory access: assume hit rate = 0.95
- Data read/ write: 30 memory access: assume hit rate = 0.90

Calculate: (1) Execution cycles without cache; (2) Execution cycles with cache.

- Cycle# w/o. cache: $(100 + 30) \times 8$
- Cycle# w. cache: $100 \cdot [0.95 \times 1 + 0.05 \times (1 + 15 + 1)] + 30 \cdot [0.9 \times 1 + 0.1 \times (1 + 15 + 1)]$



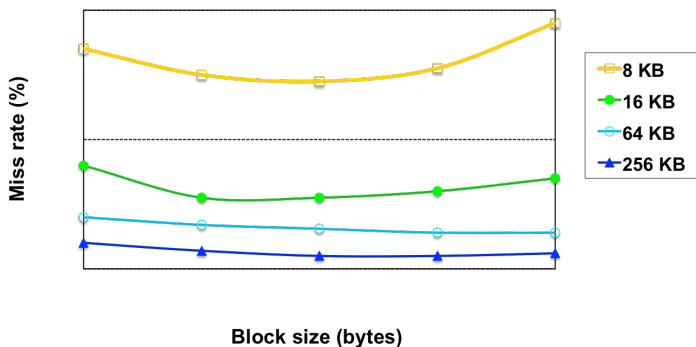
- In high-performance processors, two levels of caches are normally used, L1 and L2.
- L1 must be very fast as they determine the memory access time seen by the processor.
- L2 cache can be slower, but it should be much larger than the L1 cache to ensure a high hit rate. Its speed is less critical because it only affects the miss penalty of the L1 cache.
- Average access time on such a system:

$$h_1 \cdot C_1 + (1 - h_1) \cdot [h_2 \cdot C_2 + (1 - h_2) \cdot M]$$

- h_1 (h_2): the L1 (L2) hit rate
- C_1 the access time of L1 cache,
- C_2 the miss penalty to transfer data from L2 cache to L1
- M : the miss penalty to transfer data from MM to L2 and then to L1.



- Take advantage of spatial locality.
- 😊 If all items in a larger block are needed in a computation, it is better to load these items into the cache in a single miss.
- 😞 Larger blocks are effective only up to a certain size, beyond which too many items will remain unused before the block is replaced.
- 😞 Larger blocks take longer time to transfer and thus increase the miss penalty.
- Block sizes of 16 to 128 bytes are most popular.



Miss rate goes up if the block size becomes a significant fraction of the cache size because the number of blocks that can be held in the same size cache is smaller (increasing **capacity** misses)



Write buffer:

- Read request is served first.
- Write request stored in write buffer first and sent to memory whenever there is no read request.
- The addresses of a read request should be compared with the addresses of the write buffer.

Prefetch:

- Prefetch data into the cache before they are needed, while the processor is busy executing instructions that do not result in a read miss.
- Prefetch instructions can be inserted by the programmer or the compiler.

Load-through Approach

- Instead of waiting the whole block to be transferred, the processor resumes execution as soon as the required word is loaded in the cache.