DEPARTMENT OF
NETWORKED SYSTEMS
AND SERVICES

COMPUTER ARCHITECTURES

Practical Tasks in:

Cache Memory

Gábor Lencse

BUTE Department of Networked Systems and Services
lencse@hit.bme.hu

Budapest,
2024. 04. 25.

MŰEGYETEM 1782

TASK 1

- Let us consider a cache memory of 256 bytes. The block size is 64 bytes. The cache content is assumed to be invalid initially.
- A program reads from the following memory blocks (in the given order):
  - 1, 3, 8, 4, 3, 6, 8, 1
- Compute the number of cache misses and provide the final content of the cache
  a) in case of direct mapped organization,
  b) in case of fully associative organization with LRU block replacement strategy,
  c) in case of 2-way set associative organization with LRU block replacement strategy.

TASK 1 – SOLUTION

- Let us consider a cache memory of 256 bytes. The block size is 64 bytes. The cache content is assumed to be invalid initially.
- How many cache blocks do we have?
  - 256 / 64 = 4
- A program reads from the following memory blocks (in the given order):
  - 1, 3, 8, 4, 3, 6, 8, 1
- Compute the number of cache misses and provide the final content of the cache

a) in case of direct mapped organization,

→ The place of the blocks is determined by the last two bits of their block number

→ Let us see them in binary format!

TASK 1 – SOLUTION

- Reminder: the powers of 2
  $2^8=256$, $2^7=128$, $2^6=64$, $2^5=32$, $2^4=16$, $2^3=8$, $2^2=4$, $2^1=2$, $2^0=1$
- The block numbers (four bits are enough):
  - 1 → 00**01**, 3 → 00**11**, 8 → 10**00**, 4 → 01**00**, 3→ 00**11**,
    6 → 01**10**, 8 → 10**00**, 1→ 00**01**
- We draw the cache and follow the placement of the blocks

Valid Block number cache hits: 0, cache misses: 0

	Valid	Block number
00	0	?
01	0	?
10	0	?
11	0	?

TASK 1 – SOLUTION

- Reminder: the powers of 2
  2^8=256, 2^7=128, 2^6=64, 2^5=32, 2^4=16, 2^3=8, 2^2=4, 2^1=2, 2^0=1
- The block numbers (four bits are enough):
  - 1 → 00**01**, 3 → 00**11**, 8 → 10**00**, 4 → 01**00**, 3→ 00**11**,
    6 → 01**10**, 8 → 10**00**, 1→ 00**01**
- We draw the cache and follow the placement of the blocks

Valid Block number cache hits: 0, cache misses: 1

00	0	?
01	1	1
10	0	?
11	0	?

TASK 1 – SOLUTION

- Reminder: the powers of 2
  2^8=256, 2^7=128, 2^6=64, 2^5=32, 2^4=16, 2^3=8, 2^2=4, 2^1=2, 2^0=1
- The block numbers (four bits are enough):
  - 1 → 00**01**, 3 → 00**11**, 8 → 10**00**, 4 → 01**00**, 3→ 00**11**,
    6 → 01**10**, 8 → 10**00**, 1→ 00**01**
- We draw the cache and follow the placement of the blocks

cache hits: 0, cache misses: 2

	Valid	Block number
00	0	?
01	1	1
10	0	?
11	1	3

TASK 1 – SOLUTION

- Reminder: the powers of 2
  $2^8=256$, $2^7=128$, $2^6=64$, $2^5=32$, $2^4=16$, $2^3=8$, $2^2=4$, $2^1=2$, $2^0=1$
- The block numbers (four bits are enough):
  - 1 → 00**01**, 3 → 00**11**, 8 → 10**00**, 4 → 01**00**, 3→ 00**11**,
    6 → 01**10**, 8 → 10**00**, 1→ 00**01**
- We draw the cache and follow the placement of the blocks

cache hits: 0, cache misses: 3

	Valid	Block number
00	1	8
01	1	1
10	0	?
11	1	3

TASK 1 – SOLUTION

- Reminder: the powers of 2
  2^8=256, 2^7=128, 2^6=64, 2^5=32, 2^4=16, 2^3=8, 2^2=4, 2^1=2, 2^0=1
- The block numbers (four bits are enough):
  - 1 → 00**01**, 3 → 00**11**, 8 → 10**00**, 4 → 01**00**, 3→ 00**11**,
    6 → 01**10**, 8 → 10**00**, 1→ 00**01**
- We draw the cache and follow the placement of the blocks

cache hits: 0, cache misses: 4

	Valid	Block number
00	1	~~8~~ 4
01	1	1
10	0	?
11	1	3

TASK 1 – SOLUTION

- Reminder: the powers of 2
  2^8=256, 2^7=128, 2^6=64, 2^5=32, 2^4=16, 2^3=8, 2^2=4, 2^1=2, 2^0=1
- The block numbers (four bits are enough):
  - 1 → 00**01**, 3 → 00**11**, 8 → 10**00**, 4 → 01**00**, 3→ 00**11**,
    6 → 01**10**, 8 → 10**00**, 1→ 00**01**
- We draw the cache and follow the placement of the blocks

cache hits: 1, cache misses: 4

	Valid	Block number
00	1	~~8~~ 4
01	1	1
10	0	?
11	1	3

TASK 1 – SOLUTION

- Reminder: the powers of 2
  2^8=256, 2^7=128, 2^6=64, 2^5=32, 2^4=16, 2^3=8, 2^2=4, 2^1=2, 2^0=1
- The block numbers (four bits are enough):
  - 1 → 00**01**, 3 → 00**11**, 8 → 10**00**, 4 → 01**00**, 3→ 00**11**,
    6 → 01**10**, 8 → 10**00**, 1→ 00**01**
- We draw the cache and follow the placement of the blocks

cache hits: 1, cache misses: 5

	Valid	Block number
00	1	~~8~~ 4
01	1	1
10	1	6
11	1	3

TASK 1 – SOLUTION

- Reminder: the powers of 2
  2^8=256, 2^7=128, 2^6=64, 2^5=32, 2^4=16, 2^3=8, 2^2=4, 2^1=2, 2^0=1
- The block numbers (four bits are enough):
  - 1 → 00**01**, 3 → 00**11**, 8 → 10**00**, 4 → 01**00**, 3→ 00**11**,
    6 → 01**10**, 8 → 10**00**, 1→ 00**01**
- We draw the cache and follow the placement of the blocks

cache hits: 1, cache misses: 6

	Valid	Block number
00	1	~~8~~ ~~1~~ 8
01	1	1
10	1	6
11	1	3

TASK 1 – SOLUTION

- Reminder: the powers of 2
  2^8=256, 2^7=128, 2^6=64, 2^5=32, 2^4=16, 2^3=8, 2^2=4, 2^1=2, 2^0=1
- The block numbers (four bits are enough):
  - 1 → 00**01**, 3 → 00**11**, 8 → 10**00**, 4 → 01**00**, 3→ 00**11**,
    6 → 01**10**, 8 → 10**00**, 1→ 00**01**
- We draw the cache and follow the placement of the blocks

cache hits: 2, cache misses: 6

	Valid	Block number
00	1	~~8~~ ~~4~~ 8
01	1	1
10	1	6
11	1	3

TASK 1 – SOLUTION

- A program reads from the following memory blocks (in the given order):
  - 1, 3, 8, 4, 3, 6, 8, 1
- Compute the number of cache misses and provide the final content of the cache

a) in case of direct mapped organization, ← READY! :-)

b) in case of fully associative organization with LRU block replacement strategy,

→ Let us draw the cache and follow its operation!

cache hits: 0, cache misses: 0

Valid	Block	Age	Valid	Block	Age	Valid	Block	Age	Valid	Block	Age
0	?	1	0	?	2	0	?	3	0	?	4

TASK 1 – SOLUTION

- A program reads from the following memory blocks (in the given order):
  - 1, 3, 8, 4, 3, 6, 8, 1
- Compute the number of cache misses and provide the final content of the cache

a) in case of direct mapped organization, ← READY! :-)

b) in case of fully associative organization with LRU block replacement strategy,

→ Let us draw the cache and follow its operation!

cache hits: 0, cache misses: 1

Valid	Block	Age	Valid	Block	Age	Valid	Block	Age	Valid	Block	Age
1	1	1	0	?	2	0	?	3	0	?	4

TASK 1 – SOLUTION

- A program reads from the following memory blocks (in the given order):
  - 1, 3, 8, 4, 3, 6, 8, 1
- Compute the number of cache misses and provide the final content of the cache

a) in case of direct mapped organization, ← READY! :-)

b) in case of fully associative organization with LRU block replacement strategy,

→ Let us draw the cache and follow its operation!

cache hits: 0, cache misses: 2

Valid	Block	Age	Valid	Block	Age	Valid	Block	Age	Valid	Block	Age
1	1	2	1	3	1	0	?	3	0	?	4

TASK 1 – SOLUTION

- A program reads from the following memory blocks (in the given order):
  - 1, 3, 8, 4, 3, 6, 8, 1
- Compute the number of cache misses and provide the final content of the cache

a) in case of direct mapped organization, ← READY! :-)

b) in case of fully associative organization with LRU block replacement strategy,

→ Let us draw the cache and follow its operation!

cache hits: 0, cache misses: 3

Valid	Block	Age	Valid	Block	Age	Valid	Block	Age	Valid	Block	Age
1	1	3	1	3	2	1	8	1	0	?	4

TASK 1 – SOLUTION

- A program reads from the following memory blocks (in the given order):
  - 1, 3, 8, 4, 3, 6, 8, 1
- Compute the number of cache misses and provide the final content of the cache

a) in case of direct mapped organization, ← READY! :-)

b) in case of fully associative organization with LRU block replacement strategy,

→ Let us draw the cache and follow its operation!

cache hits: 0, cache misses: 4

Valid	Block	Age	Valid	Block	Age	Valid	Block	Age	Valid	Block	Age
1	1	4	1	3	3	1	8	2	1	4	1

TASK 1 – SOLUTION

- A program reads from the following memory blocks (in the given order):
  - 1, 3, 8, 4, 3, 6, 8, 1
- Compute the number of cache misses and provide the final content of the cache

a) in case of direct mapped organization, ← READY! :-)

b) in case of fully associative organization with LRU block replacement strategy,

→ Let us draw the cache and follow its operation!

cache hits: 1, cache misses: 4

Valid	Block	Age	Valid	Block	Age	Valid	Block	Age	Valid	Block	Age
1	1	4	1	3	1	1	8	3	1	4	2

TASK 1 – SOLUTION

- A program reads from the following memory blocks (in the given order):
  - 1, 3, 8, 4, 3, 6, 8, 1
- Compute the number of cache misses and provide the final content of the cache

a) in case of direct mapped organization, ← READY! :-)

b) in case of fully associative organization with LRU block replacement strategy,

→ Let us draw the cache and follow its operation!

cache hits: 1, cache misses: 5

Valid	Block	Age	Valid	Block	Age	Valid	Block	Age	Valid	Block	Age
1	6	1	1	3	2	1	8	4	1	4	3

TASK 1 – SOLUTION

- A program reads from the following memory blocks (in the given order):
  - 1, 3, 8, 4, 3, 6, 8, 1
- Compute the number of cache misses and provide the final content of the cache

a) in case of direct mapped organization, ← READY! :-)

b) in case of fully associative organization with LRU block replacement strategy,

→ Let us draw the cache and follow its operation!

cache hits: 2, cache misses: 5

Valid	Block	Age	Valid	Block	Age	Valid	Block	Age	Valid	Block	Age
1	6	2	1	3	3	1	8	1	1	4	4

TASK 1 – SOLUTION

- A program reads from the following memory blocks (in the given order):
  - 1, 3, 8, 4, 3, 6, 8, 1
- Compute the number of cache misses and provide the final content of the cache

a) in case of direct mapped organization, ← READY! :-)

b) in case of fully associative organization with LRU block replacement strategy, ← Now also READY! :-)

→ Let us draw the cache and follow its operation!

cache hits: 2, cache misses: 6

Valid	Block	Age	Valid	Block	Age	Valid	Block	Age	Valid	Block	Age
1	6	3	1	3	4	1	8	2	1	1	1

TASK 1 – SOLUTION

- The block numbers (four bits are enough):
  - 1 → 000**1**, 3 → 001**1**, 8 → 100**0**, 4 → 010**0**, 3→ 001**1**, 6 → 011**0**, 8 → 100**0**, 1→ 000**1**
- Compute the number of cache misses and provide the final content of the cache

  c) in case of 2-way set associative organization with LRU block replacement strategy.
  → Let us draw the cache and follow its operation!

cache hits: 0, cache misses: 0

	Valid	Block	Age	Valid	Block	Age
0	0	?	1	0	?	2
1	0	?	1	0	?	2

TASK 1 – SOLUTION

- The block numbers (four bits are enough):
  - 1 → 000**1**, 3 → 001**1**, 8 → 100**0**, 4 → 010**0**, 3→ 001**1**, 6 → 011**0**, 8 → 100**0**, 1→ 000**1**
- Compute the number of cache misses and provide the final content of the cache

  c) in case of 2-way set associative organization with LRU block replacement strategy.
  → Let us draw the cache and follow its operation!

cache hits: 0, cache misses: 1

	Valid	Block	Age	Valid	Block	Age
0	0	?	1	0	?	2
1	1	1	1	0	?	2

TASK 1 – SOLUTION

- The block numbers (four bits are enough):
  - 1 → 000**1**, 3 → 001**1**, 8 → 100**0**, 4 → 010**0**, 3→ 001**1**, 6 → 011**0**, 8 → 100**0**, 1→ 000**1**
- Compute the number of cache misses and provide the final content of the cache

  c) in case of 2-way set associative organization with LRU block replacement strategy.
  → Let us draw the cache and follow its operation!

cache hits: 0, cache misses: 2

	Valid	Block	Age	Valid	Block	Age
0	0	?	1	0	?	2
1	1	1	2	1	3	1

TASK 1 – SOLUTION

- The block numbers (four bits are enough):
  - 1 → 000**1**, 3 → 001**1**, 8 → 100**0**, 4 → 010**0**, 3→ 001**1**, 6 → 011**0**, 8 → 100**0**, 1→ 000**1**
- Compute the number of cache misses and provide the final content of the cache

  c) in case of 2-way set associative organization with LRU block replacement strategy.
  → Let us draw the cache and follow its operation!

cache hits: 0, cache misses: 3

	Valid	Block	Age	Valid	Block	Age
0	1	8	1	0	?	2
1	1	1	2	1	3	1

TASK 1 – SOLUTION

- The block numbers (four bits are enough):
  - 1 → 000**1**, 3 → 001**1**, 8 → 100**0**, 4 → 010**0**, 3→ 001**1**, 6 → 011**0**, 8 → 100**0**, 1→ 000**1**
- Compute the number of cache misses and provide the final content of the cache

  c) in case of 2-way set associative organization with LRU block replacement strategy.
  → Let us draw the cache and follow its operation!

cache hits: 0, cache misses: 4

	Valid	Block	Age	Valid	Block	Age
0	1	8	2	1	4	1
1	1	1	2	1	3	1

TASK 1 – SOLUTION

- The block numbers (four bits are enough):
  - 1 → 000**1**, 3 → 001**1**, 8 → 100**0**, 4 → 010**0**, 3→ 001**1**, 6 → 011**0**, 8 → 100**0**, 1→ 000**1**
- Compute the number of cache misses and provide the final content of the cache

  c) in case of 2-way set associative organization with LRU block replacement strategy.
  → Let us draw the cache and follow its operation!

cache hits: 1, cache misses: 4

	Valid	Block	Age	Valid	Block	Age
0	1	8	2	1	4	1
1	1	1	2	1	3	1

TASK 1 – SOLUTION

- The block numbers (four bits are enough):
  - 1 → 000**1**, 3 → 001**1**, 8 → 100**0**, 4 → 010**0**, 3→ 001**1**, 6 → 011**0**, 8 → 100**0**, 1→ 000**1**
- Compute the number of cache misses and provide the final content of the cache

  c) in case of 2-way set associative organization with LRU block replacement strategy.
  → Let us draw the cache and follow its operation!

cache hits: 1, cache misses: 5

	Valid	Block	Age	Valid	Block	Age
0	1	6	1	1	4	2
1	1	1	2	1	3	1

TASK 1 – SOLUTION

- The block numbers (four bits are enough):
  - 1 → 000**1**, 3 → 001**1**, 8 → 100**0**, 4 → 010**0**, 3→ 001**1**, 6 → 011**0**, 8 → 100**0**, 1→ 000**1**
- Compute the number of cache misses and provide the final content of the cache

  c) in case of 2-way set associative organization with LRU block replacement strategy.
  → Let us draw the cache and follow its operation!

cache hits: 1, cache misses: 6

	Valid	Block	Age	Valid	Block	Age
0	1	6	2	1	8	1
1	1	1	2	1	3	1

TASK 1 – SOLUTION

- The block numbers (four bits are enough):
  - 1 → 000**1**, 3 → 001**1**, 8 → 100**0**, 4 → 010**0**, 3→ 001**1**, 6 → 011**0**, 8 → 100**0**, 1→ 000**1**
- Compute the number of cache misses and provide the final content of the cache

  c) in case of 2-way set associative organization with LRU block replacement strategy.
  → Let us draw the cache and follow its operation!

cache hits: 2, cache misses: 6

	Valid	Block	Age	Valid	Block	Age
0	1	6	2	1	8	1
1	1	1	1	1	3	2

TASK 2

- Assume the total size of the cache memory is 512 bytes and the block size is 64 bytes. The addresses of the CPU are 16 bit wide.
- A program reads from the following memory addresses (in the given order):
  - 13, 136, 490, 541, 670, 74, 581, 980

  a) What are the ”tag”, ”index” and ”offset” values of the given addresses
    - in case of fully associative organization,
    - in case of direct mapped organization,
    - in case of 2-way set associative organization.
  - What is the final content of the cache (in all three cases)? The cache content is assumed to be invalid initially.

TASK 2 – SOLUTION

- Assume the total size of the cache memory is 512 bytes and the block size is 64 bytes. The addresses of the CPU are 16 bit wide.
  - How many cache blocks do we have?
    - 512/64=8 blocks
  - What are the sizes of the given fields?
    - Offset: $\log_2 64=6$ bits
    - Remains: 16-6=10 higher bits
    - Fully associative: tag: 10 bits
    - Direct mapping:
      - index: $\log_2 8=3$ bits, tag: 10-3=7 bits
    - 2-way set associative
      - There are two columns, and the number of rows is: 8/2=4
      - index: $\log_2 4=2$ bits, tag: 10-2=8 bits

|← 10 →|← 6 →|
[tag | offset]

|← 7 →| 3 |← 6 →|
[tag |index| offset]

|← 8 → | 2 |← 6 →|
[tag | idx | offset]

TASK 2 – SOLUTION

Fully associative

Address	← tag	→←offset→	tag	offset
13 =	0000 0000 0000	1101	0	13
136 =	0000 0000 1000	1000	2	8
490 =	0000 0001 1110	1010	7	42
541 =	0000 0010 0001	1101	8	29
670 =	0000 0010 1001	1110	10	30
74 =	0000 0000 0100	1010	1	10
581 =	0000 0010 0100	0101	9	5
980 =	0000 0011 1101	0100	15	20

TASK 2 – SOLUTION

Direct mapped

Address	← tag →←ix→←offset→	tag	index	offset
13 =	0000 0000 0000 1101	0	0	13
136 =	0000 0000 1000 1000	0	2	8
490 =	0000 0001 1110 1010	0	7	42
541 =	0000 0010 0001 1101	1	0	29
670 =	0000 0010 1001 1110	1	2	30
74 =	0000 0000 0100 1010	0	1	10
581 =	0000 0010 0100 0101	1	1	5
980 =	0000 0011 1101 0100	1	7	20

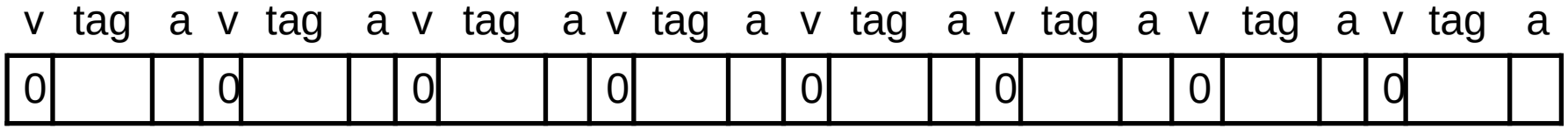
TASK 2 – SOLUTION

2-way set associative

Address	← tag →←i→←offset→	tag	index	offset
13 =	0000 0000 0000 1101	0	0	13
136 =	0000 0000 1000 1000	0	2	8
490 =	0000 0001 1110 1010	1	3	42
541 =	0000 0010 0001 1101	2	0	29
670 =	0000 0010 1001 1110	2	2	30
74 =	0000 0000 0100 1010	0	1	10
581 =	0000 0010 0100 0101	2	1	5
980 =	0000 0011 1101 0100	3	3	20

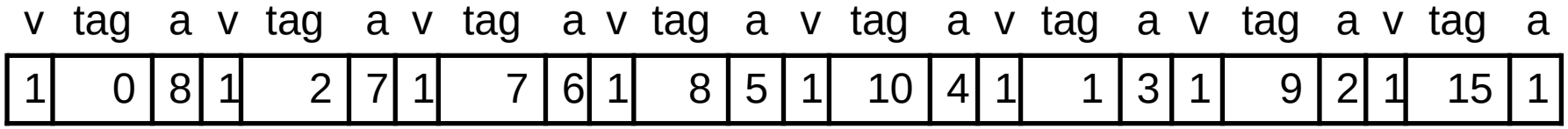
TASK 2 – SOLUTION

- What is the final content of the cache (in all three cases)? The cache content is assumed to be invalid initially.
  - Fully associative case: smart solution
    - There are 8 cache blocks
    - There are 8 memory addresses having all different tags
      → They all fit into the cache!
      → They can be filled into the cache from left to right.



TASK 2 – SOLUTION

- What is the final content of the cache (in all three cases)? The cache content is assumed to be invalid initially.
  - Fully associative case: smart solution
    - There are 8 cache blocks
    - There are 8 memory addresses having all different tags
      → They all fit into the cache!
      → They can be filled into the cache from left to right.



TASK 2 – SOLUTION

Direct mapped case: we do it step by step

Address	tag	index
13	0	0
136	0	2
490	0	7
541	1	0
670	1	2
74	0	1
581	1	1
980	1	7

	v	tag
0	0	
1	0	
2	0	
3	0	
4	0	
5	0	
6	0	
7	0	

TASK 2 – SOLUTION

Direct mapped case: we do it step by step

Address	tag	index
13	0	0
136	0	2
490	0	7
541	1	0
670	1	2
74	0	1
581	1	1
980	1	7

	v	tag
0	1	0
1	0	
2	0	
3	0	
4	0	
5	0	
6	0	
7	0	

TASK 2 – SOLUTION

Direct mapped case: we do it step by step

Address	tag	index
13	0	0
136	0	2
490	0	7
541	1	0
670	1	2
74	0	1
581	1	1
980	1	7

	v	tag
0	1	0
1	0	
2	1	0
3	0	
4	0	
5	0	
6	0	
7	0	

TASK 2 – SOLUTION

Direct mapped case: we do it step by step

Address	tag	index
13	0	0
136	0	2
490	0	7
541	1	0
670	1	2
74	0	1
581	1	1
980	1	7

	v	tag
0	1	0
1	0	
2	1	0
3	0	
4	0	
5	0	
6	0	
7	1	0

TASK 2 – SOLUTION

Direct mapped case: we do it step by step

Address	tag	index
13	0	0
136	0	2
490	0	7
541	1	0
670	1	2
74	0	1
581	1	1
980	1	7

	v	tag
0	1	~~0~~ 1
1	0	
2	1	0
3	0	
4	0	
5	0	
6	0	
7	1	0

TASK 2 – SOLUTION

Direct mapped case: we do it step by step

Address	tag	index
13	0	0
136	0	2
490	0	7
541	1	0
670	1	2
74	0	1
581	1	1
980	1	7

	v	tag
0	1	~~0~~ 1
1	0	
2	1	~~0~~ 1
3	0	
4	0	
5	0	
6	0	
7	1	0

TASK 2 – SOLUTION

Direct mapped case: we do it step by step

Address	tag	index
13	0	0
136	0	2
490	0	7
541	1	0
670	1	2
74	0	1
581	1	1
980	1	7

	v	tag
0	1	~~0~~ 1
1	1	0
2	1	~~0~~ 1
3	0	
4	0	
5	0	
6	0	
7	1	0

TASK 2 – SOLUTION

Direct mapped case: we do it step by step

Address	tag	index
13	0	0
136	0	2
490	0	7
541	1	0
670	1	2
74	0	1
581	1	1
980	1	7

	v	tag
0	1	~~0~~ 1
1	1	~~0~~ 1
2	1	~~0~~ 1
3	0	
4	0	
5	0	
6	0	
7	1	0

TASK 2 – SOLUTION

Direct mapped case: we do it step by step

Address	tag	index
13	0	0
136	0	2
490	0	7
541	1	0
670	1	2
74	0	1
581	1	1
980	1	7

	v	tag
0	1	~~0~~ 1
1	1	~~0~~ 1
2	1	~~0~~ 1
3	0	
4	0	
5	0	
6	0	
7	1	~~0~~ 1

TASK 2 – SOLUTION

2-way set associative: we do it step by step

Address	tag	index
13	0	0
136	0	2
490	1	3
541	2	0
670	2	2
74	0	1
581	2	1
980	3	3

	v	tag	a	v	tag	a
0	0			0		
1	0			0		
2	0			0		
3	0			0		

TASK 2 – SOLUTION

2-way set associative: we do it step by step

Address	tag	index
13	0	0
136	0	2
490	1	3
541	2	0
670	2	2
74	0	1
581	2	1
980	3	3

	v	tag	a	v	tag	a
0	1	0	1	0		
1	0			0		
2	0			0		
3	0			0		

TASK 2 – SOLUTION

2-way set associative: we do it step by step

Address	tag	index
13	0	0
136	0	2
490	1	3
541	2	0
670	2	2
74	0	1
581	2	1
980	3	3

	v	tag	a	v	tag	a
0	1	0		1	0	
1	0			0		
2	1	0		1	0	
3	0			0		

TASK 2 – SOLUTION

2-way set associative: we do it step by step

Address	tag	index
13	0	0
136	0	2
490	1	3
541	2	0
670	2	2
74	0	1
581	2	1
980	3	3

	v	tag	a	v	tag	a
0	1	0	1	0		
1	0			0		
2	1	0	1	0		
3	1	1	1	0		

TASK 2 – SOLUTION

2-way set associative: we do it step by step

Address	tag	index
13	0	0
136	0	2
490	1	3
541	2	0
670	2	2
74	0	1
581	2	1
980	3	3

	v	tag	a	v	tag	a
0	1	0	2	1	2	1
1	0			0		
2	1	0	1	0		
3	1	1	1	0		

TASK 2 – SOLUTION

2-way set associative: we do it step by step

Address	tag	index
13	0	0
136	0	2
490	1	3
541	2	0
670	2	2
74	0	1
581	2	1
980	3	3

	v	tag	a	v	tag	a
0	1	0	2	1	2	1
1	0			0		
2	1	0	2	1	2	1
3	1	1	1	0		

TASK 2 – SOLUTION

2-way set associative: we do it step by step

Address	tag	index
13	0	0
136	0	2
490	1	3
541	2	0
670	2	2
74	0	1
581	2	1
980	3	3

	v	tag	a	v	tag	a
0	1	0	2	1	2	1
1	1	0	1	0		
2	1	0	2	1	2	1
3	1	1	1	0		

TASK 2 – SOLUTION

2-way set associative: we do it step by step

Address	tag	index
13	0	0
136	0	2
490	1	3
541	2	0
670	2	2
74	0	1
581	2	1
980	3	3

	v	tag	a	v	tag	a
0	1	0	2	1	2	1
1	1	0	2	1	2	1
2	1	0	2	1	2	1
3	1	1	1	0		

TASK 2 – SOLUTION

2-way set associative: we do it step by step

Address	tag	index
13	0	0
136	0	2
490	1	3
541	2	0
670	2	2
74	0	1
581	2	1
980	3	3

	v	tag	a	v	tag	a
0	1	0	2	1	2	1
1	1	0	2	1	2	1
2	1	0	2	1	2	1
3	1	1	2	1	3	1

TASK 3

- The size of the cache memory of a CPU is 1kB, the block size is 64 bytes. The CPU executes the following program:

```
short int t[32][32];
int sum = 0;

for (int i=0; i<32; i++)
    for (int j=0; j<32; j++)
        sum += t[i][j];
```

TASK 3

Assumptions: the size of the `short int` type is 2 byte, array `t` starts at a block boundary in the memory, the two-dimensional array is arranged in a row-continuous way in the memory, the cache uses a direct mapped organization. Variables `i`,`j` are stored in registers, using them does not involve the cache memory.

a) How many cache misses occur during the execution of the given algorithm? Compute the cache miss ratio!

b) How many cache misses occur if the two for loops are swapped? Compute the cache miss ratio!

c) How large cache memory is needed to achieve the same cache miss ratio with the swapped variant as with the original variant?

TASK 3 – SOLUTION

- The size of the cache memory of a CPU is 1kB, the block size is 64 bytes.
  - How many block does the cache memory have?
    - 1024/64=16
  - How does the cache look like? See here →
  - How much memory is used by array `t[32][32]`?
    - 32*32*2byte=2kB
  - How much memory is used by a single row of array `t[32][32]`?
    - 32*2byte=64byte
      → it exactly fits into a cache block
      Recall that array `t` starts at a block boundary!

	v	tag
0	0	
1	0	
2	0	
3	0	
4	0	
5	0	
6	0	
7	0	
8	0	
9	0	
10	0	
11	0	
12	0	
13	0	
14	0	
15	0	

TASK 3 – SOLUTION

- How does array `t[32][32]` look like?

```
t[0][0],  t[0][1],  …, t[0][j],  …, t[0][30],  t[0][31],
t[1][0],  t[1][1],  …, t[1][j],  …, t[1][30],  t[1][31],
…
t[i][0],  t[i][1],  …, t[i][j],  …, t[i][30],  t[i][31],
…
t[30][0], t[30][1], …, t[30][j], …, t[30][30], t[30][31],
t[31][0], t[31][1], …, t[31][j], …, t[31][30], t[31][31],
```

```
for (int i=0; i<32; i++)
    for (int j=0; j<32; j++)
        sum += t[i][j];
```

The array is stored and it is also read in a row-continuous way!

1st row: 1 cache miss, 31 cache hits. 2nd row, 3rd row, etc.: the same!

TASK 3 – SOLUTION

Let us answer the questions!

a) How many cache misses occur during the execution of the given algorithm? Compute the cache miss ratio!
   - All in all: 32 cache misses
   - Cache miss ratio 32/(32*32)=1/32=3.125%

b) How many cache misses occur if the two for loops are swapped? Compute the cache miss ratio!

```
for (int j=0; i<32; i++)
    for (int i=0; j<32; j++)
        sum += t[i][j];
```

   - Column continuous traversing of the 2-dimensional array!
     - The inside loop loads the first 16 rows into the cache
     - And then overwrites them by the 2nd 16 rows! :-(
     - The outside loop repeats it 32 times
     - Cache miss ratio is 100%

TASK 3 – SOLUTION

c) How large cache memory is needed to achieve the same cache miss ratio with the swapped variant as with the original variant?

→ Let us double the size of the cache memory!

- What happens now?

```
for (int j=0; i<32; i++)
    for (int i=0; j<32; j++)
        sum += t[i][j];
```

- The first execution of the internal loop produces 32 cache misses, and it loads all the rows of the array into the cache
- All further executions of the internal loop will produce cache hits and no cache misses!
- Thus, we achieved the same cache miss ratio as before! :-)

  Answer: 2kB.