CS2100 COMPUTER ORGANISATION

http://www.comp.nus.edu.sg/~cs2100/

Recitation-12

Cache

14 April 2025

Aaron Tan

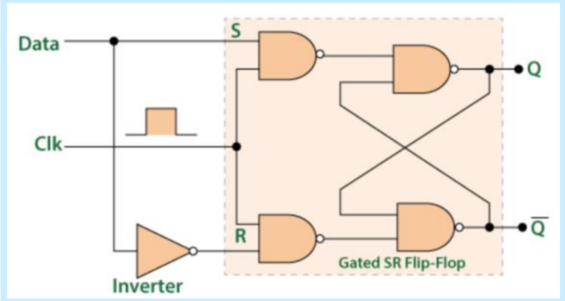
NUS National University of Singapore | School of Computing

Logic Gate:

Contents

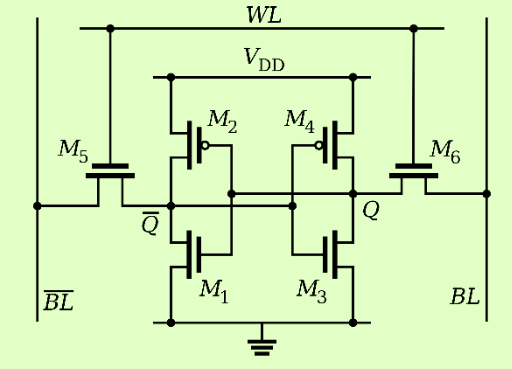
Memory Hierarchy

Cache: 10 to 100 times faster than RAM.



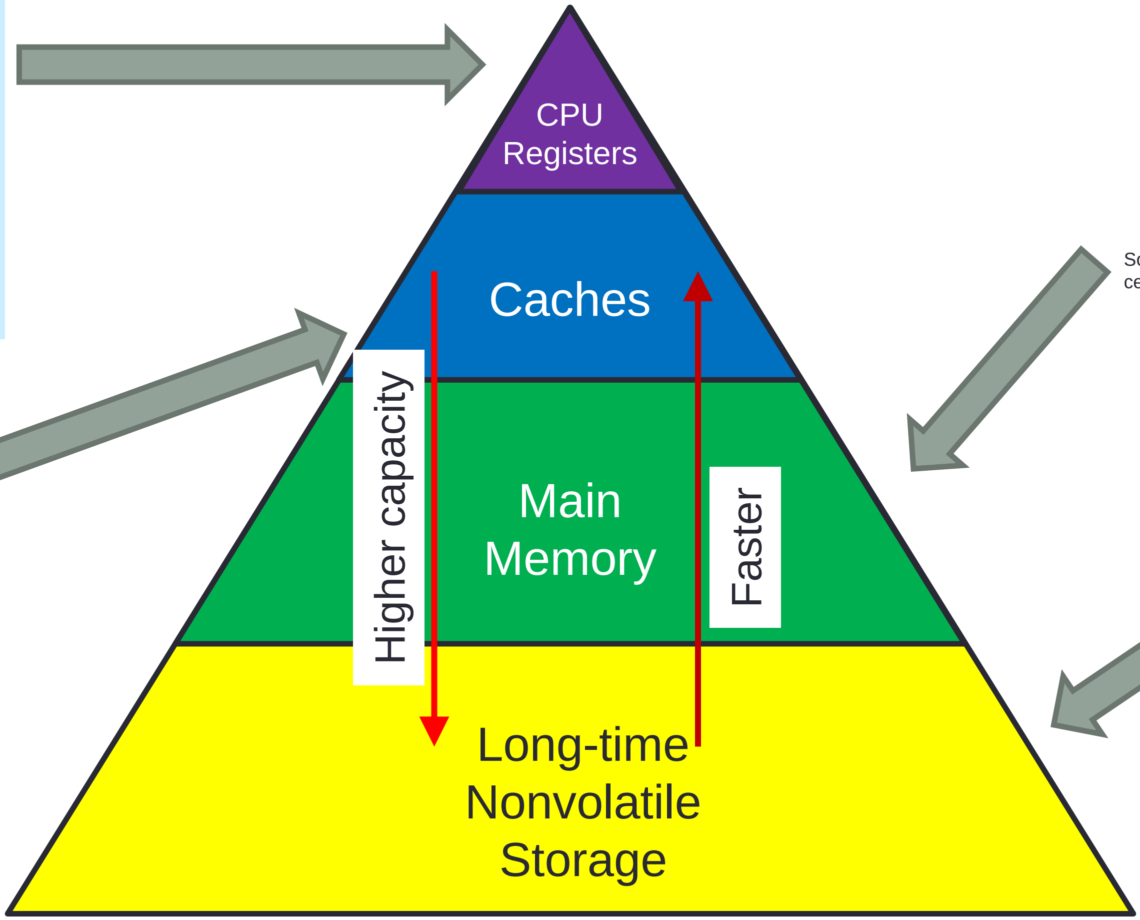
D flip-flops

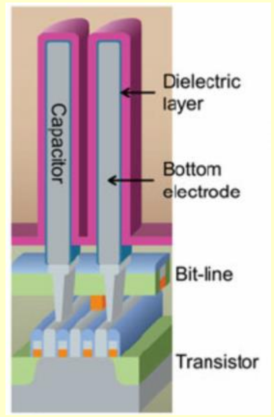
Source: Wikipedia



Static Random Access Memory

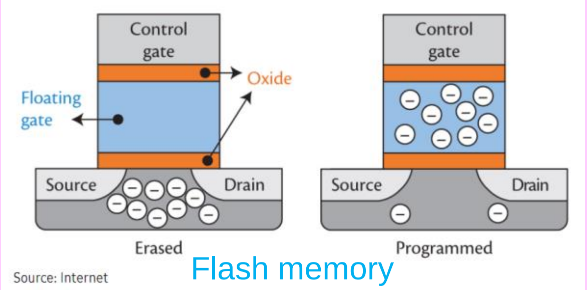
Source: Wikipedia



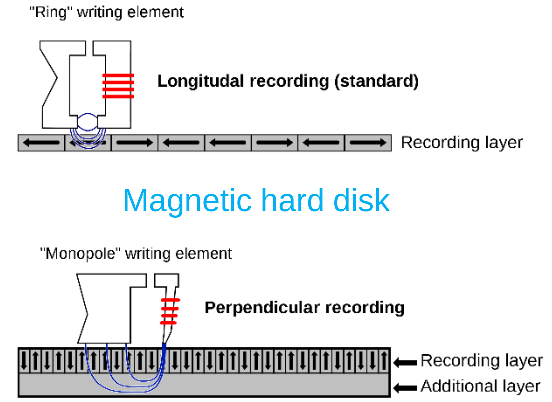


Dynamic Random Access Memory

Source: https://www.researchgate.net/figure/Schematic-diagrams-of-a-DRAM-cells-which-consist-of-a-cell-transistor-and-capacitor_fig1_258797946



Flash memory

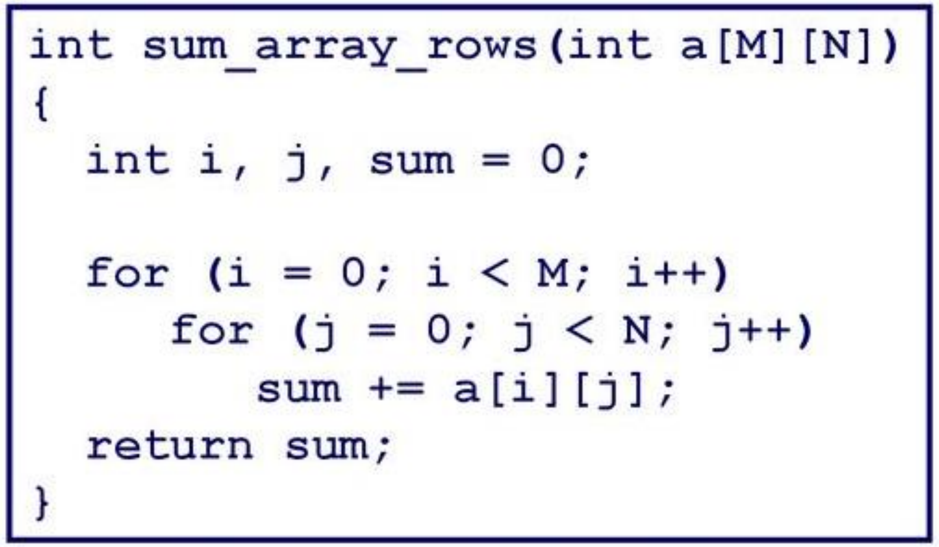


Magnetic hard disk

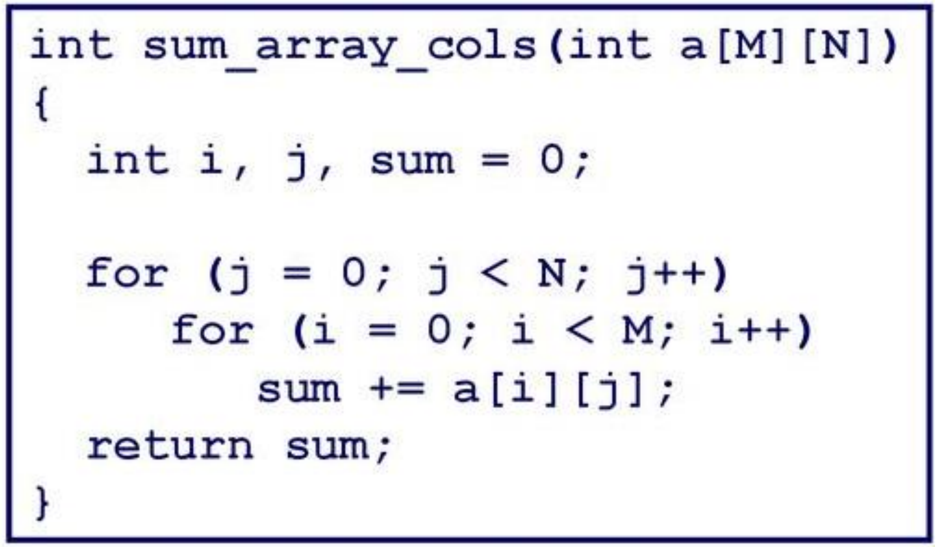
Source: Wikipedia

Temporal & Spatial Locality

Example: 4-byte words, 4-word cache blocks.



```
int sum_array_rows(int a[M][N])
{
  int i, j, sum = 0;

  for (i = 0; i < M; i++)
     for (j = 0; j < N; j++)
        sum += a[i][j];
  return sum;
}
```



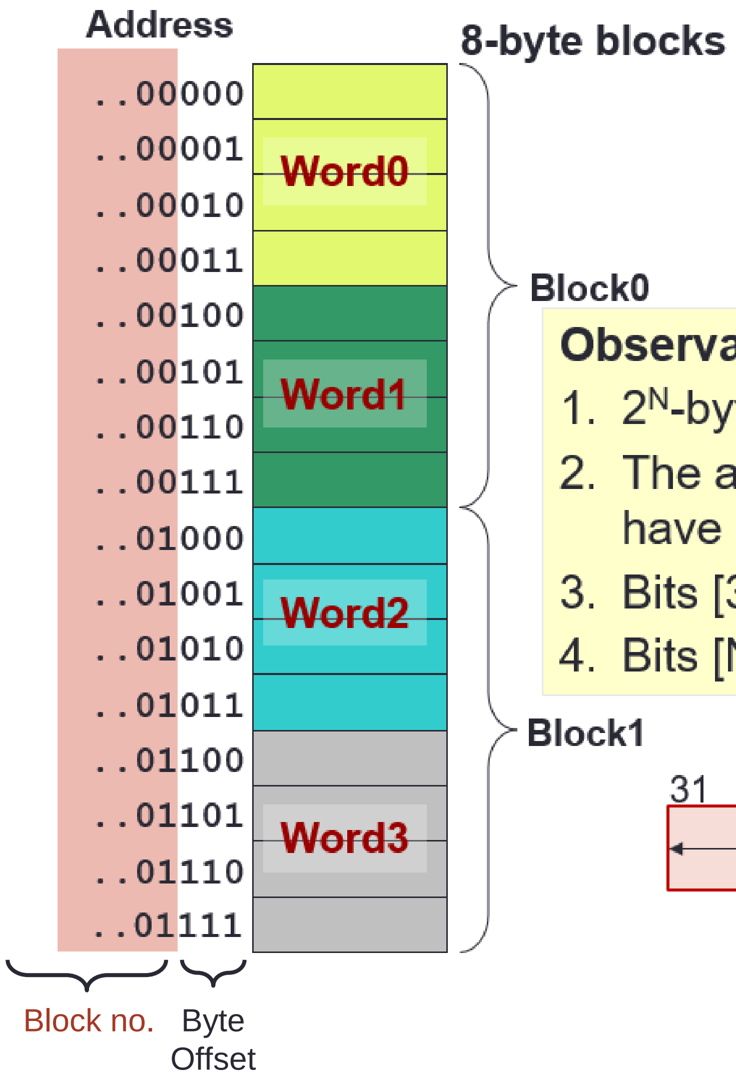
```
int sum_array_cols(int a[M][N])
{
  int i, j, sum = 0;

  for (j = 0; j < N; j++)
     for (i = 0; i < M; i++)
        sum += a[i][j];
  return sum;
}
```

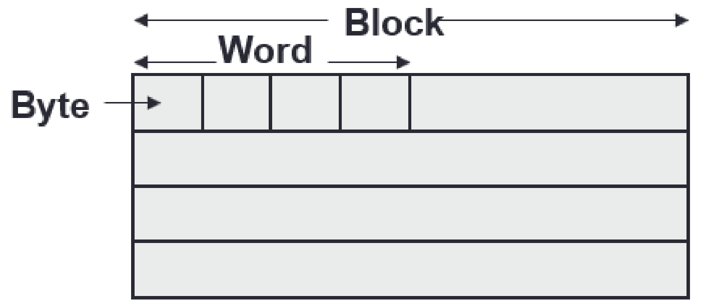
3×4 integer array *A*

A[0][0]	A[0][1]	A[0][2]	A[0][3]
A[1][0]	A[1][1]	A[1][2]	A[1][3]
A[2][0]	A[2][1]	A[2][2]	A[2][3]

Direct Mapped Cache - Basics

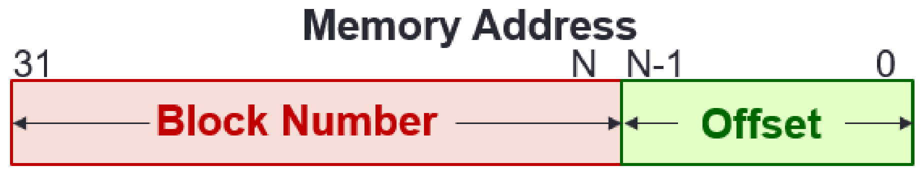
- Cache blocks/Line
- Block Size
- Block Number
- Offset
- Cache Index
- Tag





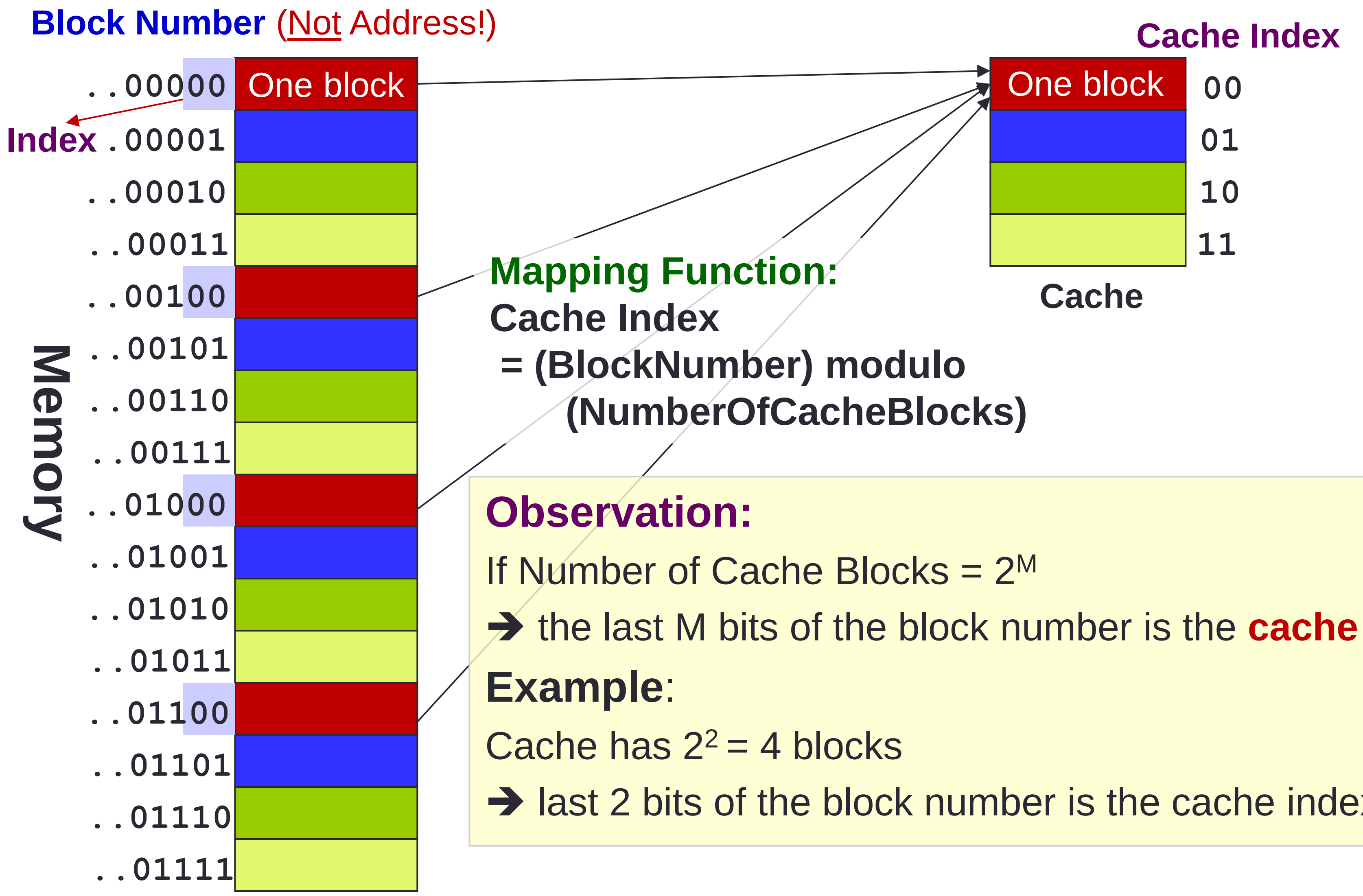
Observations:

1. 2^N-byte blocks are aligned at 2^N-byte boundaries
2. The addresses of words within a 2^N-byte block have identical (32-N) most significant bits (MSB).
3. Bits [31:N] ➔ the **block number**
4. Bits [N-1:0] ➔ the **byte offset** within a block



Direct Mapped Cache - Basics

- Cache blocks/Line
- Block Size
- Block Number
- Offset
- Cache Index
- Tag



Mapping Function:
Cache Index
= (BlockNumber) modulo (NumberOfCacheBlocks)

Observation:

If Number of Cache Blocks = 2^M

➔ the last M bits of the block number is the **cache index**

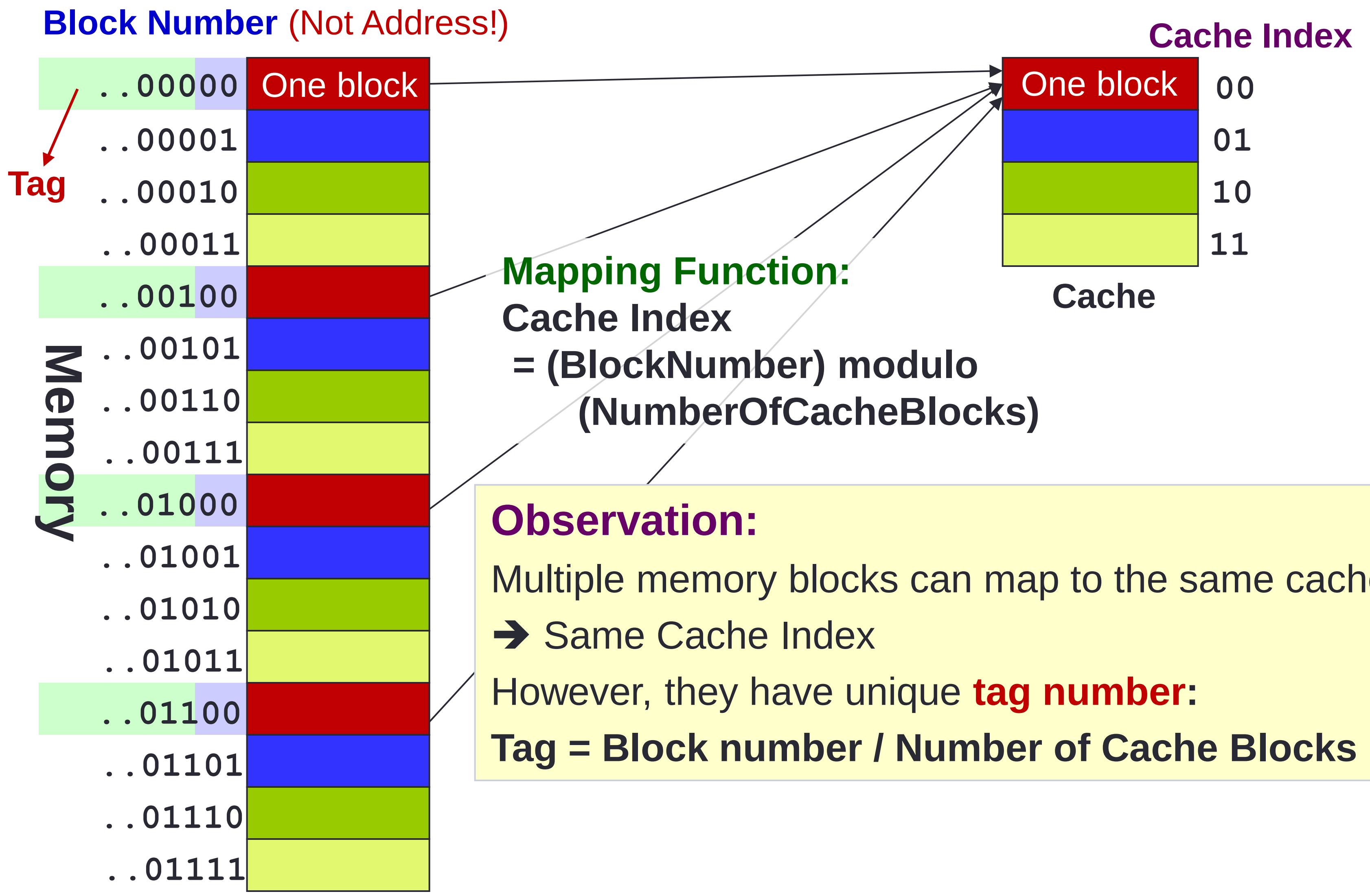
Example:

Cache has $2^2 = 4$ blocks

➔ last 2 bits of the block number is the cache index.

Direct Mapped Cache - Basics

- Cache blocks/Line
- Block Size
- Block Number
- Offset
- Cache Index
- Tag



Mapping Function:
Cache Index = (BlockNumber) modulo (NumberOfCacheBlocks)

Observation:

Multiple memory blocks can map to the same cache block

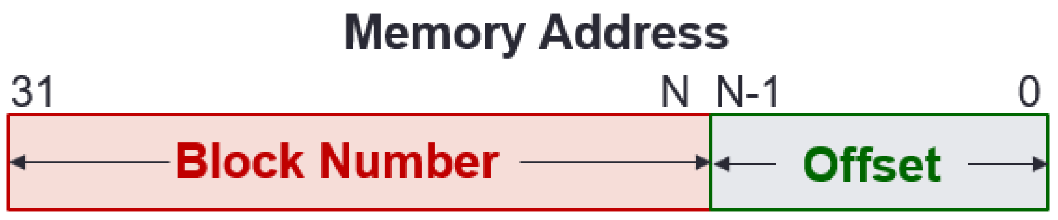
➔ Same Cache Index

However, they have unique **tag number**:

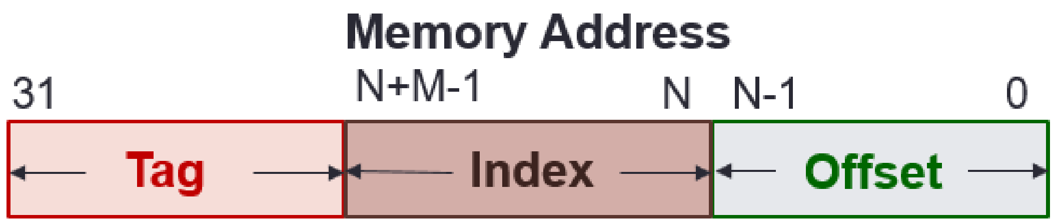
Tag = Block number / Number of Cache Blocks

Direct Mapped Cache - Basics

- Memory address
  - Tag
  - Index
  - Byte Offset



Cache Block size = 2^N bytes



Cache Block size = 2^N bytes

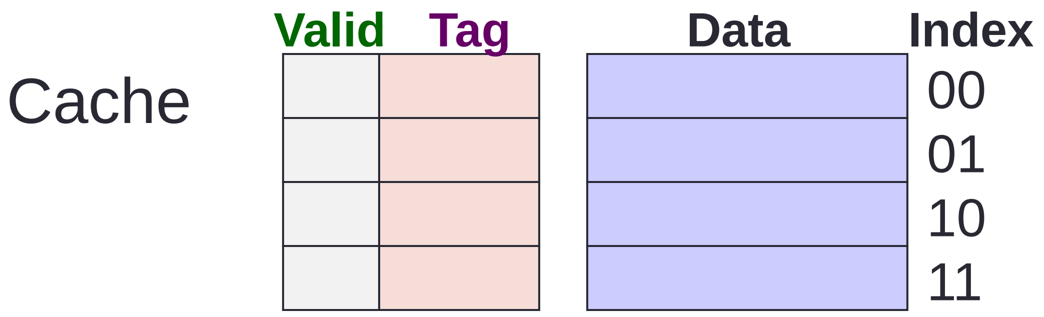
Number of cache blocks = 2^M

Offset = N bits

Index = M bits

Tag = 32 – (N + M) bits

Direct Mapped Cache - Basics



Along with a data block (line), cache also contains the following administrative information (overheads):

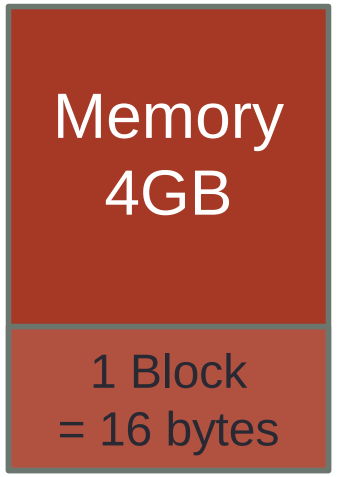
1. **Tag** of the memory block
2. **Valid bit** indicating whether the cache line contains valid data

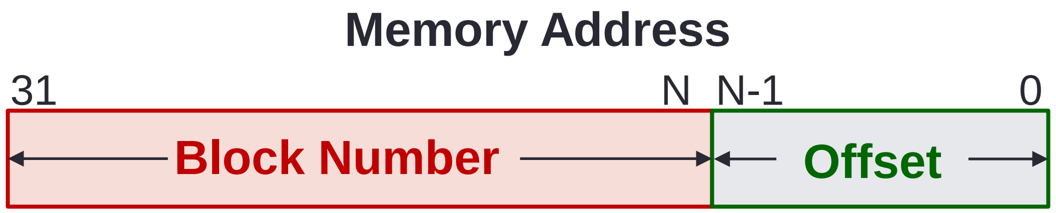
When is there a cache hit?

(Valid[index] = TRUE) **AND**

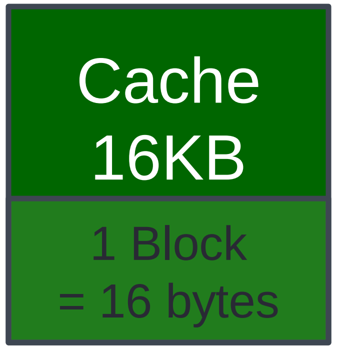
(Tag[index] = Tag[memory address])

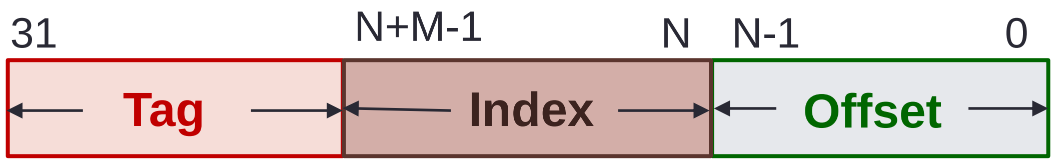
Direct Mapped Cache - Example





Offset, N = 4 bits
Block Number = 32 – 4 = **28 bits**
Check: Number of Blocks = **2^{28}**

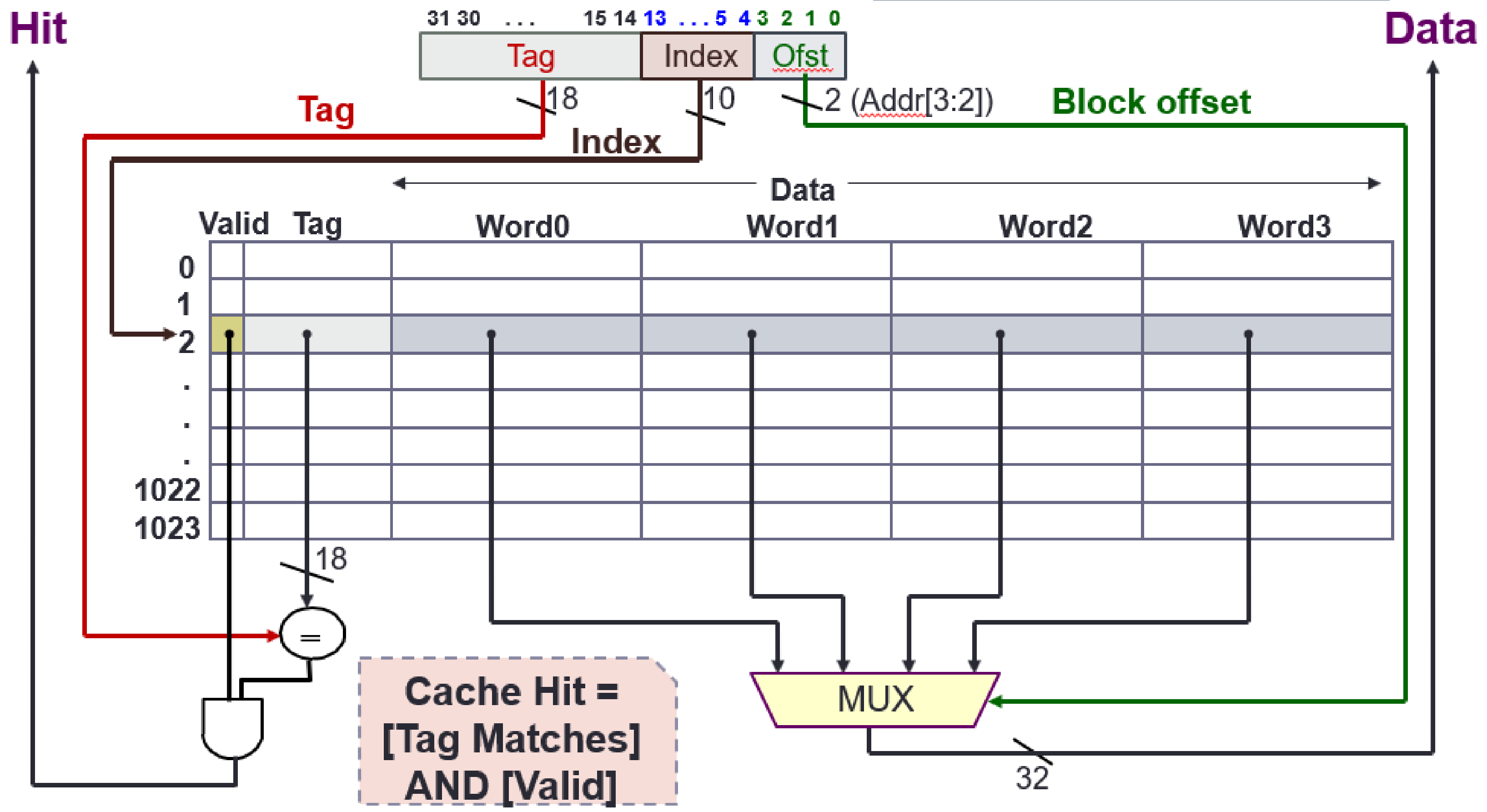




Number of Cache Blocks
= 16KB / 16bytes = 1024 = **2^{10}**
Cache Index, M = **10bits**
Cache Tag = 32 – 10 – 4 = **18 bits**

Direct Mapped Cache - Example

16-KB cache:
4-word (16-byte) blocks



Cache Hit = [Tag Matches] AND [Valid]

5. Reading Data: **Setup**

- Given a direct mapped 16KB cache:
  - 16-byte blocks x 1024 cache blocks
- Trace the following memory accesses:

Tag	Index	Offset
31 14	13 4	3 0
000000000000000000	0000000001	0100
000000000000000000	0000000001	1100
000000000000000000	0000000011	0100
000000000000000010	0000000001	1000
000000000000000000	0000000001	0000

Direct Mapped Cache

5. Reading Data: **Initial State**

- Intially cache is empty
  - ➔ All ***valid*** bits are zeroes (false)

			Data			
Index	**Valid**	**Tag**	**Word0** Bytes 0-3	**Word1** Bytes 4-7	**Word2** Bytes 8-11	**Word3** Bytes 12-15
0	**0**					
1	**0**					
2	**0**					
3	**0**					
4	**0**					
5	**0**					
		... ...	... ...	... ...	... ...	
1022	**0**					
1023	**0**					

5. Reading Data: **Load #1-1**

- Load from

Tag	Index	Offset
000000000000000000	0000000001	0100

Step 1. Check Cache Block at index **1**

Index	**Valid**	**Tag**	Data: **Word0** Bytes 0-3	**Word1** Bytes 4-7	**Word2** Bytes 8-11	**Word3** Bytes 12-15
0	**0**					
1	**0**					
2	**0**					
3	**0**					
4	**0**					
5	**0**					
	...	...	...	...	...	...
1022	**0**					
1023	**0**					

Direct Mapped Cache

5. Reading Data: **Load #1-2**

- Load from

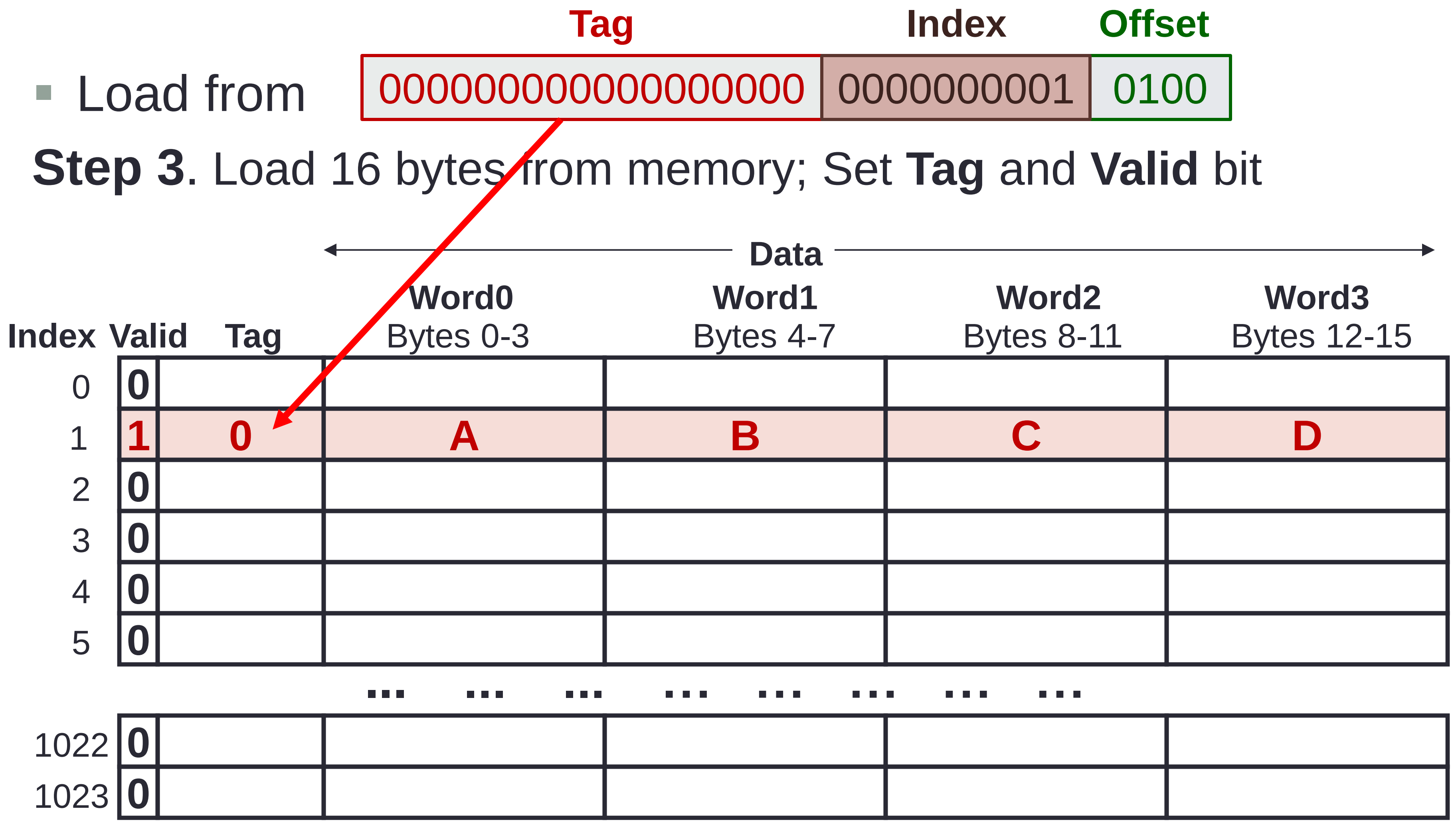
Tag	Index	Offset
000000000000000000	0000000001	0100

Step 2. Data in block 1 is **invalid** **[Cold/Compulsory Miss]**

			Data			
			Word0	**Word1**	**Word2**	**Word3**
Index	**Valid**	**Tag**	Bytes 0-3	Bytes 4-7	Bytes 8-11	Bytes 12-15
0	**0**					
1	**0**					
2	**0**					
3	**0**					
4	**0**					
5	**0**					
		…	…	…	…	…
1022	**0**					
1023	**0**					

Direct Mapped Cache

5. Reading Data: **Load #1-3**



- Load from 000000000000000000 0000000001 0100

Step 3. Load 16 bytes from memory; Set **Tag** and **Valid** bit

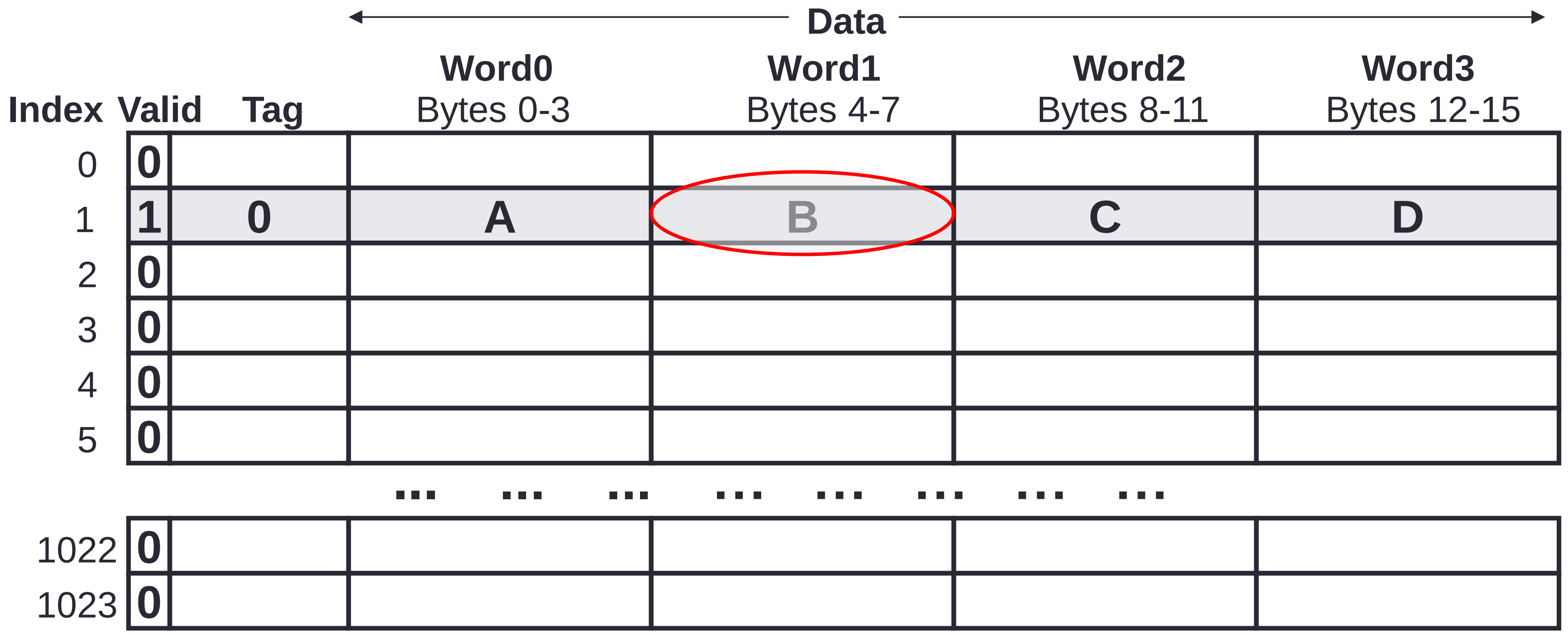
Index	Valid	Tag	Word0 Bytes 0-3	Word1 Bytes 4-7	Word2 Bytes 8-11	Word3 Bytes 12-15
0	0					
1	1	0	A	B	C	D
2	0					
3	0					
4	0					
5	0					
	...	...	...	...	...	...
1022	0					
1023	0					

Direct Mapped Cache

5. Reading Data: **Load #1-4**

	Tag	Index	Offset
▪ Load from	0000000000000000000	0000000001	0100

Step 4. Return **Word1** (byte offset = 4) to Register



Index	Valid	Tag	Word0 Bytes 0-3	Word1 Bytes 4-7	Word2 Bytes 8-11	Word3 Bytes 12-15
0	0					
1	1	0	A	B	C	D
2	0					
3	0					
4	0					
5	0					
	...	...	...	...	...	...
1022	0					
1023	0					

Direct Mapped Cache

5. Reading Data: **Load #2-1**

- Load from

Tag	Index	Offset
000000000000000000	0000000001	1100

Step 1. Check Cache Block at index **1**

Index	Valid	Tag	Data: Word0 Bytes 0-3	Word1 Bytes 4-7	Word2 Bytes 8-11	Word3 Bytes 12-15
0	0					
1	1	0	A	B	C	D
2	0					
3	0					
4	0					
5	0					
		... ...	... ...	... ...	... ...	
1022	0					
1023	0					

Direct Mapped Cache

5. Reading Data: **Load #2-2**

	Tag	Index	Offset
Load from	000000000000000000	0000000001	1100

Step 2. [Cache Block is Valid] AND [Tags match] ➔ Cache hit!

			Data			
			Word0	**Word1**	**Word2**	**Word3**
Index	**Valid**	**Tag**	Bytes 0-3	Bytes 4-7	Bytes 8-11	Bytes 12-15
0	0					
1	1	0	A	B	C	D
2	0					
3	0					
4	0					
5	0					
		...	...	...	...	...
1022	0					
1023	0					

5. Reading Data: **Load #2-3**

	Tag	Index	Offset
▪ Load from	000000000000000000	0000000001	1100

Step 3. Return **Word3** (byte offset = 12) to Register **[Spatial Locality]**

			Data			
			Word0	**Word1**	**Word2**	**Word3**
Index	**Valid**	**Tag**	Bytes 0-3	Bytes 4-7	Bytes 8-11	Bytes 12-15
0	**0**					
1	**1**	**0**	**A**	**B**	**C**	**D**
2	**0**					
3	**0**					
4	**0**					
5	**0**					
		... ...	... ...	... ...	... ...	
1022	**0**					
1023	**0**					

Direct Mapped Cache

5. Reading Data: **Load #3-1**

	Tag	Index	Offset
▪ Load from	000000000000000000	0000000011	0100

Step 1. Check Cache Block at index **3**

			Data			
			Word0	**Word1**	**Word2**	**Word3**
Index	**Valid**	**Tag**	Bytes 0-3	Bytes 4-7	Bytes 8-11	Bytes 12-15
0	**0**					
1	**1**	**0**	**A**	**B**	**C**	**D**
2	**0**					
3	**0**					
4	**0**					
5	**0**					
		…	…	… …	… …	… …
1022	**0**					
1023	**0**					

Direct Mapped Cache

5. Reading Data: **Load #3-2**

	Tag	Index	Offset
▪ Load from	000000000000000000	0000000011	0100

Step 2. Data in block 3 is **invalid** **[Cold/Compulsory Miss]**

Index	Valid	Tag	Data: Word0 Bytes 0-3	Data: Word1 Bytes 4-7	Data: Word2 Bytes 8-11	Data: Word3 Bytes 12-15
0	0					
1	1	0	A	B	C	D
2	0					
3	0					
4	0					
5	0					
		...	...	...	...	...
1022	0					
1023	0					

Direct Mapped Cache

5. Reading Data: **Load #3-3**

- Load from

Tag	Index	Offset
000000000000000000	0000000011	0100

Step 3. Load 16 bytes from memory; Set **Tag** and **Valid** bit

Index	**Valid**	**Tag**	**Data** **Word0** Bytes 0-3	**Data** **Word1** Bytes 4-7	**Data** **Word2** Bytes 8-11	**Data** **Word3** Bytes 12-15
0	**0**					
1	**1**	**0**	**A**	**B**	**C**	**D**
2	**0**					
3	**1**	**0**	**I**	**J**	**K**	**L**
4	**0**					
5	**0**					
		…	**…**	**…**	**…**	**…**
1022	**0**					
1023	**0**					

Direct Mapped Cache

5. Reading Data: **Load #3-4**

	Tag	Index	Offset
▪ Load from	000000000000000000	0000000011	0100

Step 4. Return **Word1** (byte offset = 4) to Register

			Data			
Index	**Valid**	**Tag**	**Word0** Bytes 0-3	**Word1** Bytes 4-7	**Word2** Bytes 8-11	**Word3** Bytes 12-15
0	**0**					
1	**1**	**0**	**A**	**B**	**C**	**D**
2	**0**					
3	**1**	**0**	**I**	**J**	**K**	**L**
4	**0**					
5	**0**					
		...	... ...	... ...	... ...	...
1022	**0**					
1023	**0**					

Direct Mapped Cache

5. Reading Data: **Load #4-1**

Tag	Index	Offset
0000000000000000010	0000000001	1000

- Load from 0000000000000000010 0000000001 1000

Step 1. Check Cache Block at index **1**

			Data			
Index	**Valid**	**Tag**	**Word0** Bytes 0-3	**Word1** Bytes 4-7	**Word2** Bytes 8-11	**Word3** Bytes 12-15
0	**0**					
1	**1**	**0**	**A**	**B**	**C**	**D**
2	**0**					
3	**1**	**0**	**I**	**J**	**K**	**L**
4	**0**					
5	**0**					
	...	...	...	...	...	...
1022	**0**					
1023	**0**					

Direct Mapped Cache

5. Reading Data: **Load #4-2**

- Load from

Tag	**Index**	**Offset**
000000000000000010	0000000001	1000

Step 2. Cache block is **Valid** but **Tags mismatch** **[Cold miss]**

Index	**Valid**	**Tag**	**Word0** Bytes 0-3	**Word1** Bytes 4-7	**Word2** Bytes 8-11	**Word3** Bytes 12-15
0	**0**					
1	**1**	**0**	**A**	**B**	**C**	**D**
2	**0**					
3	**1**	**0**	**I**	**J**	**K**	**L**
4	**0**					
5	**0**					
		...	...	...	...	...
1022	**0**					
1023	**0**					

Data spans Word0–Word3.

Direct Mapped Cache

5. Reading Data: **Load #4-3**

Tag	Index	Offset
000000000000000010	0000000001	1000

- Load from 000000000000000010 0000000001 1000

Step 3. Replace block **1** with new data; Set **Tag**

Index	Valid	Tag	Word0 Bytes 0-3	Word1 Bytes 4-7	Word2 Bytes 8-11	Word3 Bytes 12-15
0	0					
1	1	2	E	F	G	H
2	0					
3	1	0	I	J	K	L
4	0					
5	0					
…	…	…	…	…	…	…
1022	0					
1023	0					

Data

Direct Mapped Cache

5. Reading Data: **Load #4-4**

- Load from

Tag	Index	Offset
000000000000000010	0000000001	1000

Step 4. Return **Word2** (byte offset = 8) to Register

Index	**Valid**	**Tag**	**Word0** Bytes 0-3	**Word1** Bytes 4-7	**Word2** Bytes 8-11	**Word3** Bytes 12-15
0	0					
1	1	2	E	F	G	H
2	0					
3	1	0	I	J	K	L
4	0					
5	0					
...	...	...	...	...	...	...
1022	0					
1023	0					

Direct Mapped Cache

5. Reading Data: **Load #5-1**

- Load from

Tag	Index	Offset
000000000000000000	0000000001	0000

Step 1. Check Cache Block at index **1**

			Data			
			Word0	**Word1**	**Word2**	**Word3**
Index	**Valid**	**Tag**	Bytes 0-3	Bytes 4-7	Bytes 8-11	Bytes 12-15
0	**0**					
1	**1**	**2**	**E**	**F**	**G**	**H**
2	**0**					
3	**1**	**0**	**I**	**J**	**K**	**L**
4	**0**					
5	**0**					
		…	…	…	…	…
1022	**0**					
1023	**0**					

Direct Mapped Cache

5. Reading Data: **Load #5-2**

	Tag	Index	Offset
▪ Load from	000000000000000000	0000000001	0000

Step 2. Cache block is **Valid** but **Tags mismatch** **[Cold miss]**

			Data			
			Word0	**Word1**	**Word2**	**Word3**
Index	**Valid**	**Tag**	Bytes 0-3	Bytes 4-7	Bytes 8-11	Bytes 12-15
0	0					
1	1	2	E	F	G	H
2	0					
3	1	0	I	J	K	L
4	0					
5	0					
		…	…	…	…	…
1022	0					
1023	0					

Direct Mapped Cache

5. Reading Data: **Load #5-3**

- Load from

Tag	Index	Offset
000000000000000000	0000000001	0000

Step 3. Replace block **1** with new data; Set **Tag**

			Data			
			Word0	**Word1**	**Word2**	**Word3**
Index	**Valid**	**Tag**	Bytes 0-3	Bytes 4-7	Bytes 8-11	Bytes 12-15
0	**0**					
1	**1**	**0**	**A**	**B**	**C**	**D**
2	**0**					
3	**1**	**0**	**I**	**J**	**K**	**L**
4	**0**					
5	**0**					
		...	...	...	... ...	... ...	...
1022	**0**					
1023	**0**					

Direct Mapped Cache

5. Reading Data: **Load #5-4**

- Load from

Tag	Index	Offset
000000000000000000	0000000001	0000

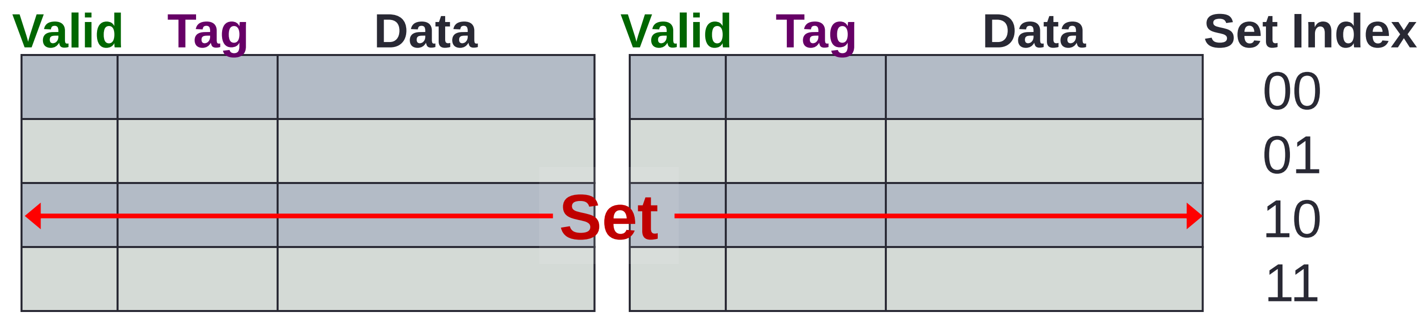
Step 4. Return **Word0** (byte offset = 0) to Register

Index	**Valid**	**Tag**	**Word0** Bytes 0-3	**Word1** Bytes 4-7	**Word2** Bytes 8-11	**Word3** Bytes 12-15
0	**0**					
1	**1**	**0**	**A**	**B**	**C**	**D**
2	**0**					
3	**1**	**0**	**I**	**J**	**K**	**L**
4	**0**					
5	**0**					
		…	…	…	…	…
1022	**0**					
1023	**0**					

3. Set Associative (SA) Cache

- **N-way Set Associative Cache**
  - A memory block can be placed in a fixed number (**N**) of locations in the cache, where **N** > 1
- **Key Idea:**
  - Cache consists of a number of sets:
    - Each set contains **N** cache blocks
  - Each memory block maps to a unique cache set
  - Within the set, a memory block can be placed in **any** of the **N** cache blocks in the set

3. Set Associative Cache: **Structure**

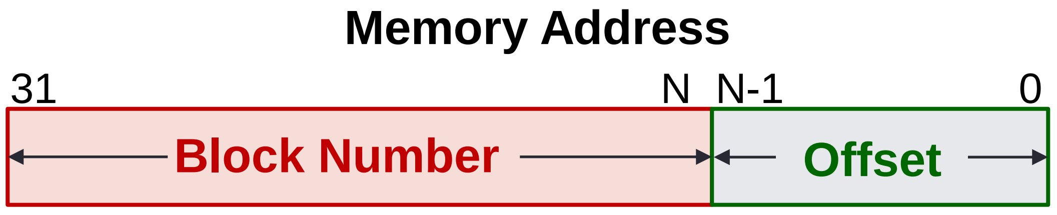


2-way Set Associative Cache

- An example of 2-way set associative cache
  - Each set has two cache blocks
- A memory block maps to a **unique set**
  - In the set, the memory block can be placed in **either of the cache blocks**

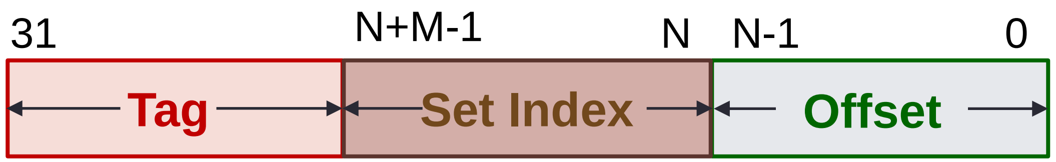
➔ Need to search both blocks in the set to look for the memory block

3. Set Associative Cache: **Mapping**



Cache Block size = 2^N bytes

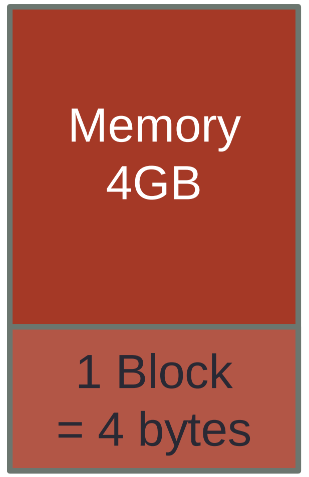
Cache Set Index
= (BlockNumber) modulo (NumberOfCacheSets)

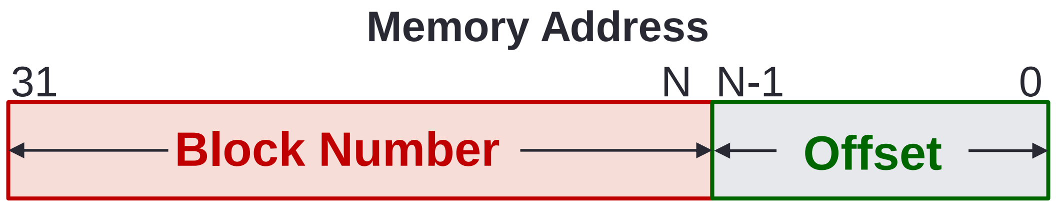


Cache Block size = 2^N bytes
Number of cache sets = 2^M
Offset = **N bits**
Set Index = **M bits**
Tag = **32 – (N + M) bits**

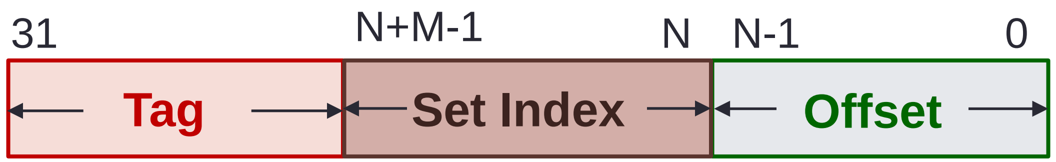
Observation:
It is essentially unchanged from the direct-mapping formula

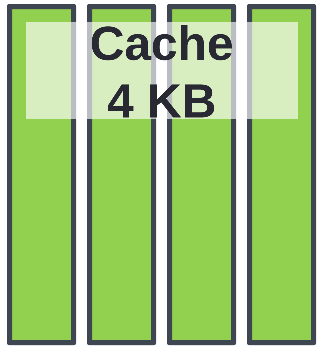
3. Set Associative Cache: **Example**





Offset, N = 2 bits
Block Number = 32 – 2 = **30 bits**
Check: Number of Blocks = $\mathbf{2^{30}}$



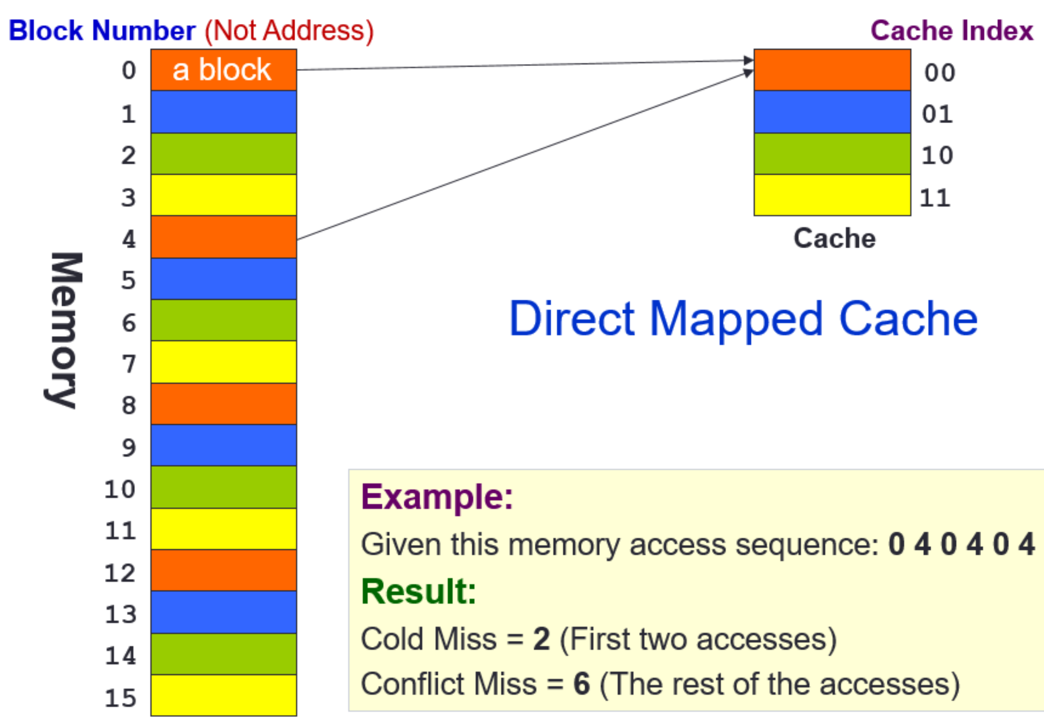


Number of Cache Blocks
= 4KB / 4bytes = 1024 **=** $\mathbf{2^{10}}$

4-way associative, number of sets
= 1024 / 4 = 256 **=** $\mathbf{2^{8}}$
Set Index, M = 8 bits

Cache Tag = 32 – 8 – 2 = **22 bits**

Direct Mapped Cache vs 2-way Set Assoc Cache



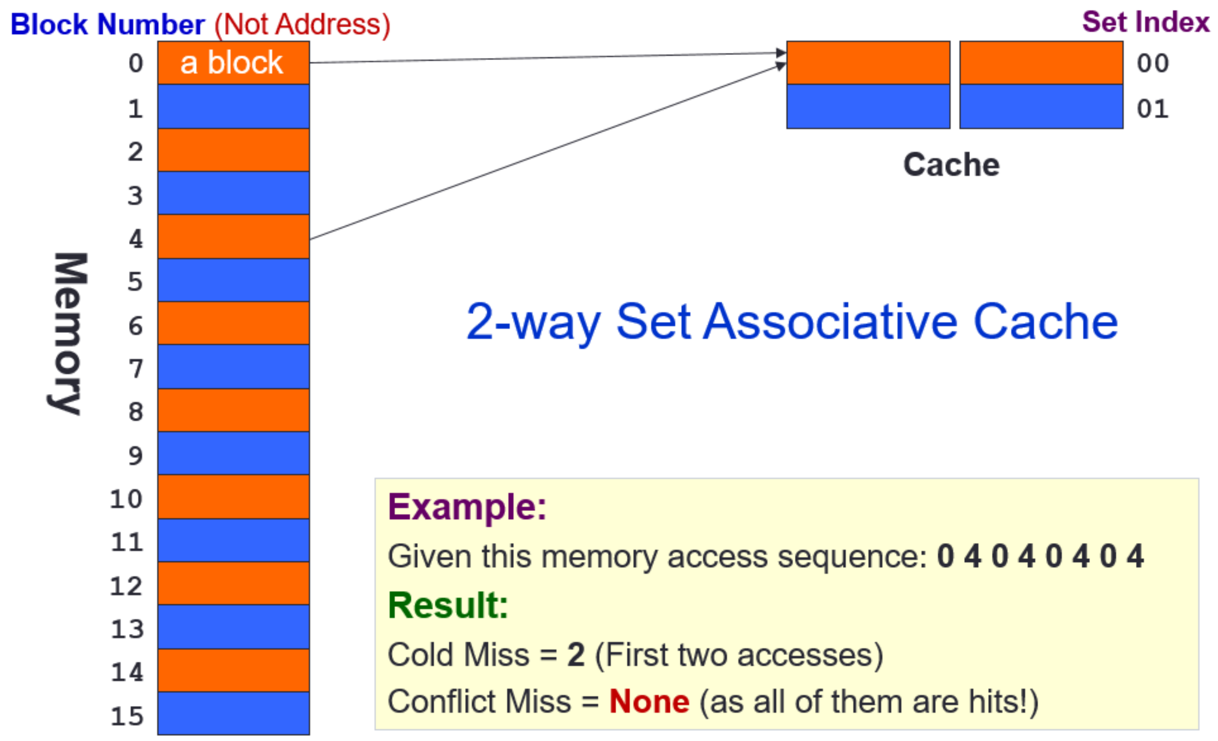
Example:

Given this memory access sequence: **0 4 0 4 0 4 0 4**

Result:

Cold Miss = **2** (First two accesses)

Conflict Miss = **6** (The rest of the accesses)



Example:

Given this memory access sequence: **0 4 0 4 0 4 0 4**

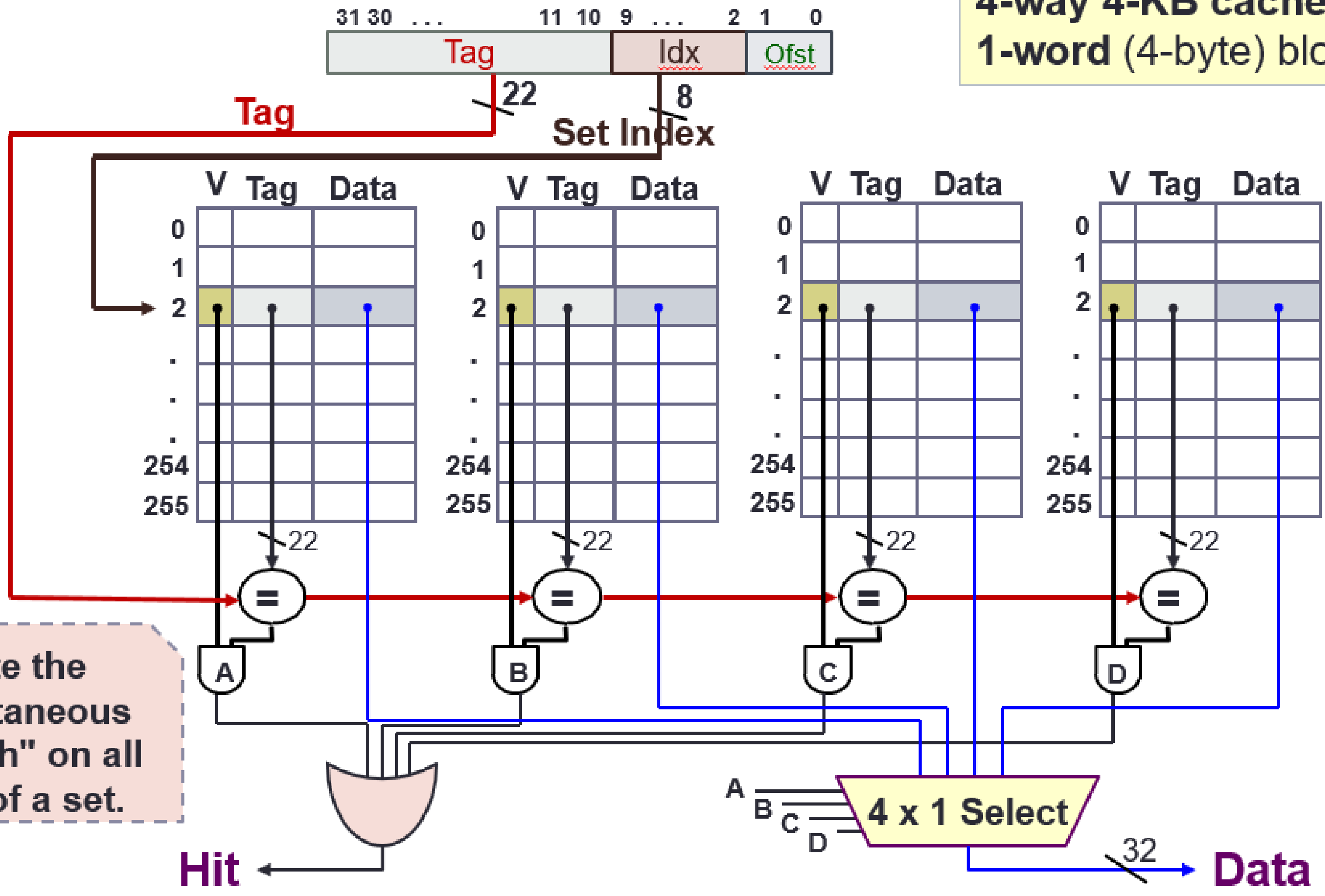
Result:

Cold Miss = **2** (First two accesses)

Conflict Miss = **None** (as all of them are hits!)

4-way Set Associative Cache

4-way 4-KB cache:
1-word (4-byte) blocks



Note the simultaneous "search" on all tags of a set.

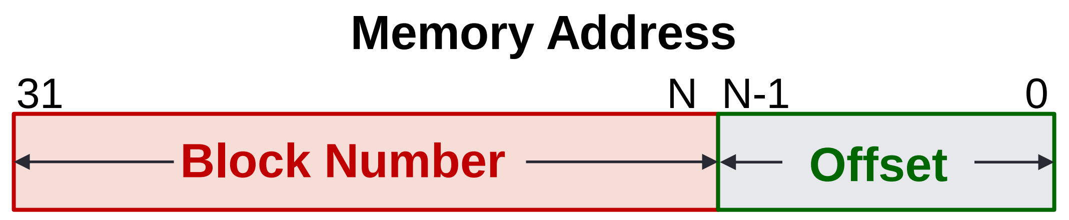
4. Fully Associative (FA) Cache

- **Fully Associative Cache**
  - A memory block can be placed in any location in the cache
- **Key Idea:**
  - Memory block placement is no longer restricted by cache index or cache set index

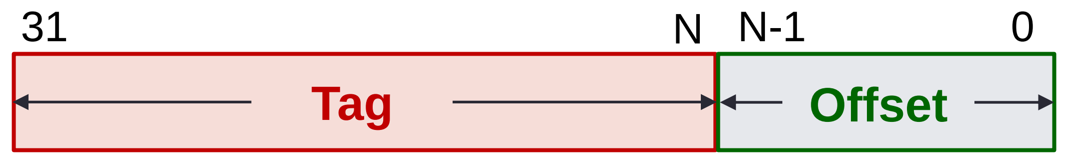
++ Can be placed in any location, **BUT**

--- Need to search all cache blocks for memory access

4. Fully Associative Cache: **Mapping**



Cache Block size = 2^N bytes



Cache Block size = 2^N bytes

Number of cache blocks = 2^M

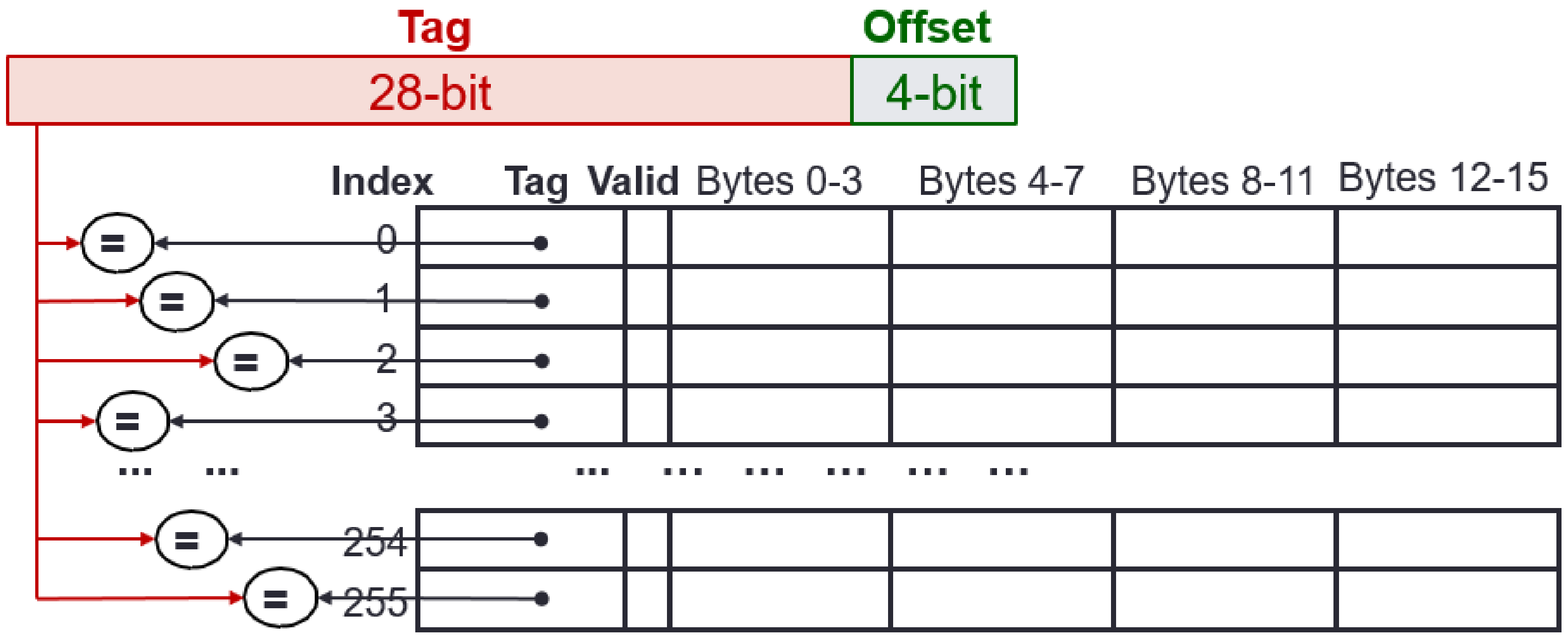
Offset = **N bits**

Tag = **32 – N bits**

Observation:
The block number serves as the tag in FA cache.

Fully Associative Cache

- Example:
  - 4KB cache size and 16-Byte block size
  - Compare tags and valid bit in parallel



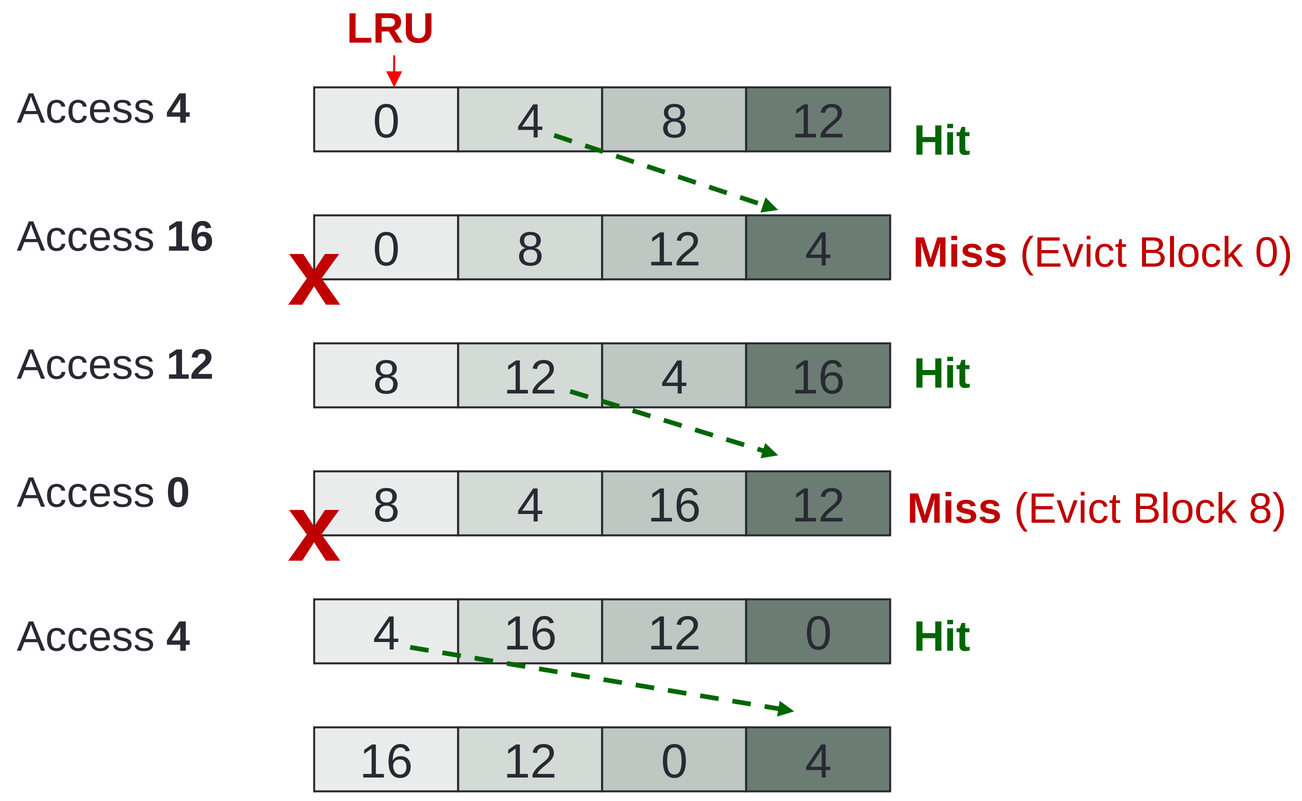
No Conflict Miss (since data can go anywhere)

5. Block Replacement Policy (1/3)

- Set Associative or Fully Associative Cache:
  - Can choose where to place a memory block
  - Potentially replacing another cache block if full
  - Need **block replacement policy**
- Least **R**ecently Used (**LRU**)
  - **How:** For cache hit, record the cache block that was accessed
    - When replacing a block, choose one which has not been accessed for the longest time
  - **Why:** Temporal locality

5. Block Replacement Policy (2/3)

- Least Recently Used policy in action:
  - 4-way SA cache
  - Memory accesses: **0 4 8 12 4 16 12 0 4**



5. Block Replacement Policy (3/3)

- Drawback for LRU
  - Hard to keep track if there are many choices
- Other replacement policies:
  - First in first out (FIFO)
  - Random replacement (RR)
  - Least frequently used (LFU)

Types of Cache Misses

- Compulsory misses
  - On the first access to a block; the block must be brought into the cache
  - Also called cold start misses or first reference misses
- Conflict misses
  - Occur in the case of direct mapped cache or set associative cache, when several blocks are mapped to the same block/set
  - Also called collision misses or interference misses
- Capacity misses
  - Occur when blocks are discarded from cache as cache cannot contain all blocks needed

Cache Summary

One-way set associative (direct mapped)

Block	Tag	Data
0		
1		
2		
3		
4		
5		
6		
7		

Two-way set associative

Set	Tag	Data	Tag	Data
0				
1				
2				
3				

Four-way set associative

Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0								
1								

Eight-way set associative (fully associative)

Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data

Cache Summary

Block Placement: Where can a block be placed in cache?

Direct Mapped:	**N-way Set-Associative:**	**Fully Associative:**
• Only one block defined by index	• Any one of the **N** blocks within the set defined by index	• Any cache block

Block Identification: How is a block found if it is in the cache?

Direct Mapped:	**N-way Set Associative:**	**Fully Associative:**
• Tag match with only one block	• Tag match for all the blocks within the set	• Tag match for all the blocks within the cache

Cache Summary

Block Replacement: Which block should be replaced on a cache miss?

Direct Mapped:	**N-way Set-Associative:**	**Fully Associative:**
• No Choice	• Based on replacement policy	• Based on replacement policy

Write Strategy: What happens on a write?

Write Policy: Write-through vs write-back

Write Miss Policy: Write allocate vs write no allocate

AY2021/22 Semester 1 Exam Q16

AY2021/22 Sem1 Q16

Q16. Cache [18 marks]

Refer to the following MIPS code which is the same as the one in question 14. Here, we look only at instructions I1 to I18. The data segment in the MIPS code in question 14 no longer applies here as the arrays contain a lot more elements in this question.

```
       add  $t0, $0, $0    # I1
       add  $t9, $0, $0    # I2
       addi $t1, $a1, 0    # I3
       addi $t2, $a2, 0    # I4
       sll  $t8, $a0, 2    # I5
loop:  slt  $t3, $t0, $t8  # I6
       beq  $t3, $0, end   # I7
       lw   $s1, 0($t1)    # I8
       lw   $s2, 0($t2)    # I9
       slt  $t3, $s1, $s2  # I10
       bne  $t3, $0, else  # I11
       add  $t9, $t9, $s1  # I12
       j    cont           # I13
else:  sub  $t9, $t9, $s2  # I14
cont:  addi $t0, $t0, 4    # I15
       addi $t1, $t1, 4    # I16
       addi $t2, $t2, 4    # I17
       j    loop           # I18
end:
```

For parts (a) to (d): You are given a **direct-mapped data cache** with 512 words in total and each block contains 8 words. You may assume the following:

- Array *A* starts at address **0xCDEF 0400**.
- Array *B* follows immediately after array *A* in the memory. That is, if the last element of array *A* is at address *x*, then the first element of array *B* is at address (*x* + 4).

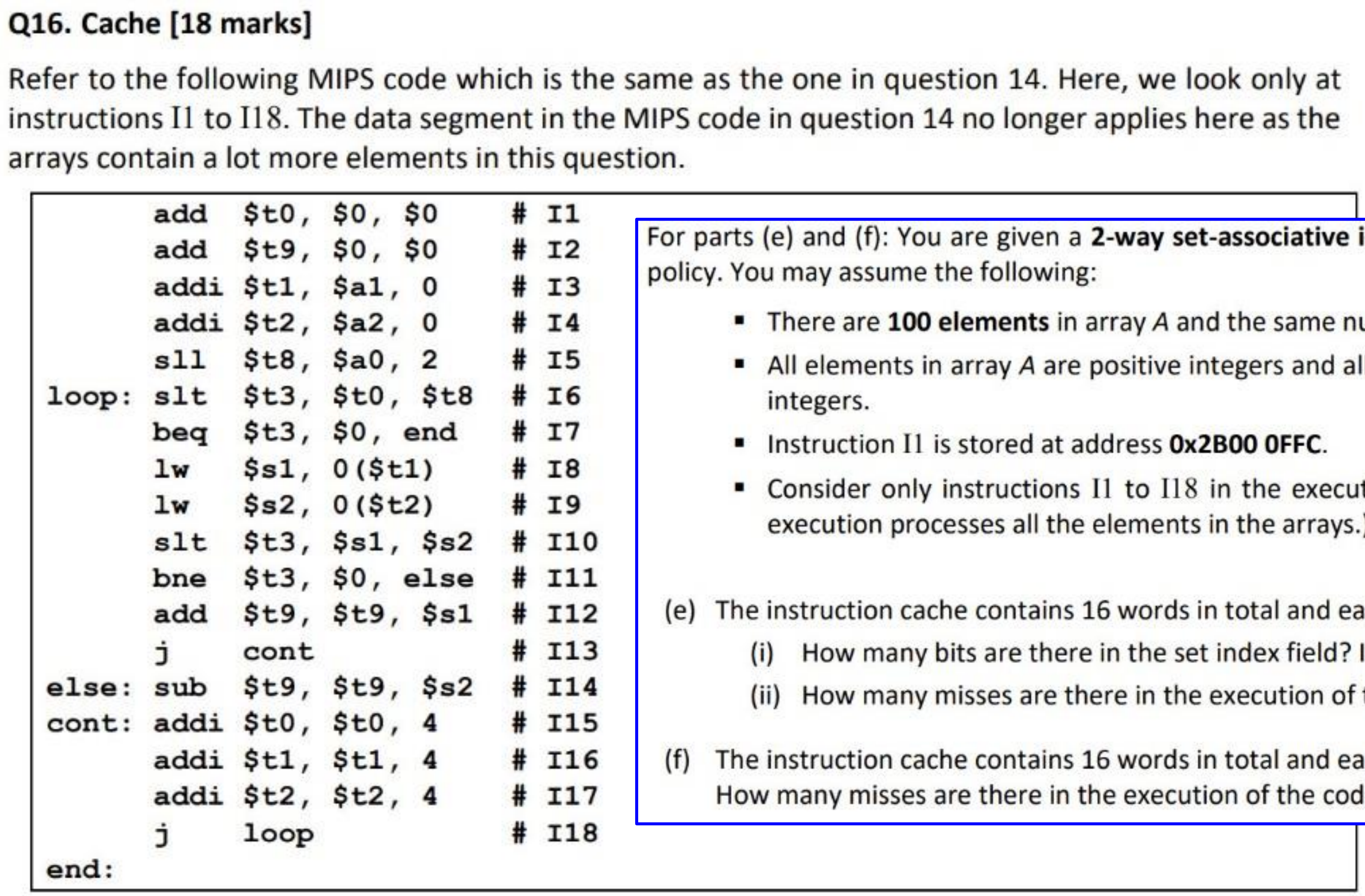
(a) How many bits are there in the index field? In the byte offset field? [2]

(b) Assuming that array *A* contains **1032** elements and array *B* contains the same number of elements, what is the hit rate of the data cache (i) for array *A* and (ii) for array *B*? Write your answers in fractions if they are not equal to zero. [2]

(c) Assuming that array *A* contains **1028** elements and array *B* contains the same number of elements, what is the hit rate of the data cache (i) for array *A* and (ii) for array *B*? Write your answers in fractions if they are not equal to zero. [4]

(d) Assuming that arrays *A* and *B* have at least 100 elements each, (i) what is the lowest hit rate possible for array *A*? Write your answer in fraction if it is not zero; (ii) how many elements in array *A* would result in this lowest hit rate? [2]

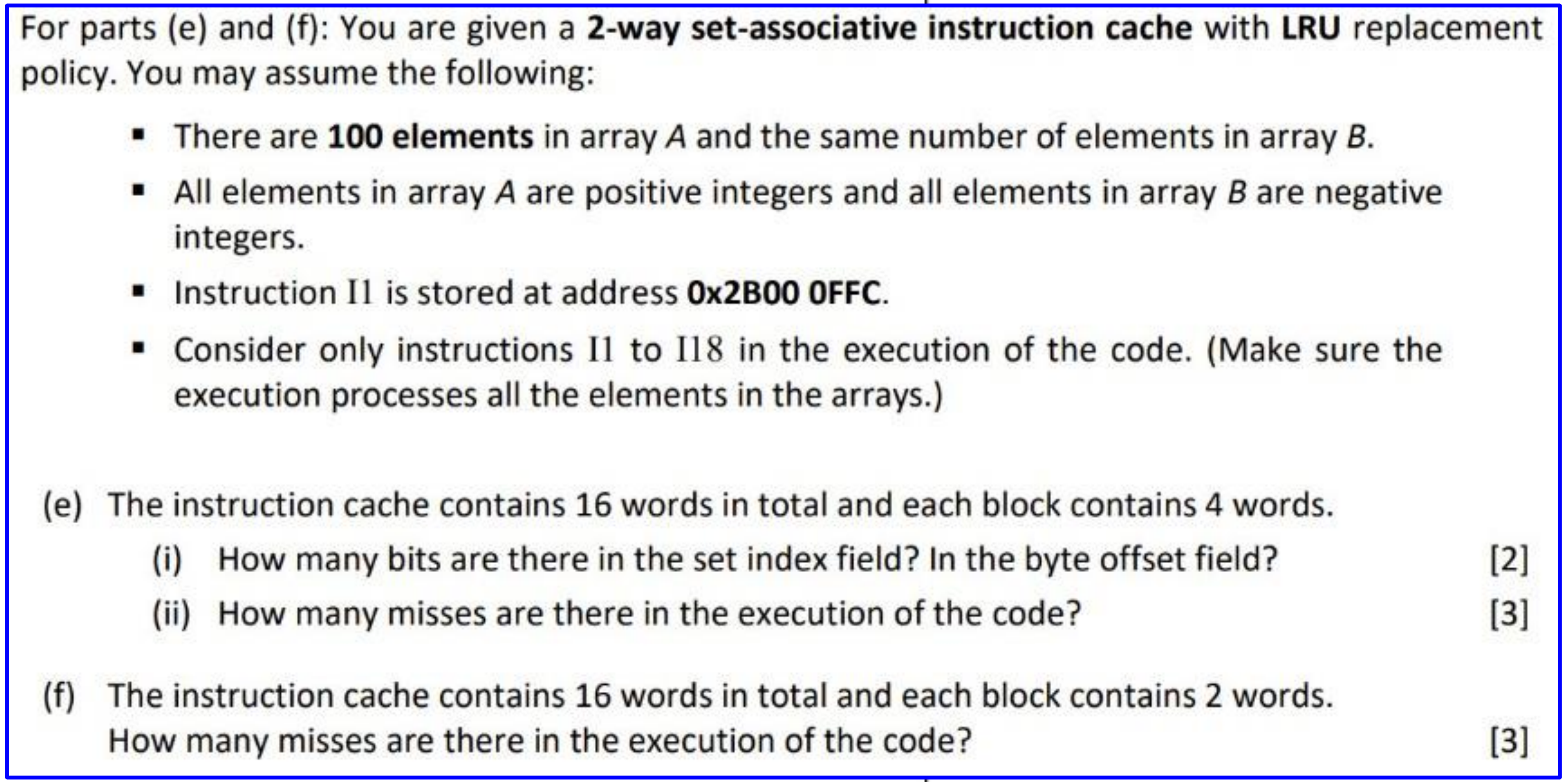
AY2021/22 Sem1 Q16



Q16. Cache [18 marks]

Refer to the following MIPS code which is the same as the one in question 14. Here, we look only at instructions I1 to I18. The data segment in the MIPS code in question 14 no longer applies here as the arrays contain a lot more elements in this question.

```
        add  $t0, $0, $0    # I1
        add  $t9, $0, $0    # I2
        addi $t1, $a1, 0    # I3
        addi $t2, $a2, 0    # I4
        sll  $t8, $a0, 2    # I5
loop:   slt  $t3, $t0, $t8  # I6
        beq  $t3, $0, end   # I7
        lw   $s1, 0($t1)    # I8
        lw   $s2, 0($t2)    # I9
        slt  $t3, $s1, $s2  # I10
        bne  $t3, $0, else  # I11
        add  $t9, $t9, $s1  # I12
        j    cont           # I13
else:   sub  $t9, $t9, $s2  # I14
cont:   addi $t0, $t0, 4    # I15
        addi $t1, $t1, 4    # I16
        addi $t2, $t2, 4    # I17
        j    loop           # I18
end:
```



For parts (e) and (f): You are given a **2-way set-associative instruction cache** with **LRU** replacement policy. You may assume the following:

- There are **100 elements** in array *A* and the same number of elements in array *B*.
- All elements in array *A* are positive integers and all elements in array *B* are negative integers.
- Instruction I1 is stored at address **0x2B00 0FFC**.
- Consider only instructions I1 to I18 in the execution of the code. (Make sure the execution processes all the elements in the arrays.)

(e) The instruction cache contains 16 words in total and each block contains 4 words.

(i) How many bits are there in the set index field? In the byte offset field? [2]

(ii) How many misses are there in the execution of the code? [3]

(f) The instruction cache contains 16 words in total and each block contains 2 words. How many misses are there in the execution of the code? [3]

```
        add   $t0, $0, $0      # I1
        add   $t9, $0, $0      # I2
        addi  $t1, $a1, 0      # I3
        addi  $t2, $a2, 0      # I4
        sll   $t8, $a0, 2      # I5
loop:   slt   $t3, $t0, $t8    # I6
        beq   $t3, $0, end     # I7
        lw    $s1, 0($t1)      # I8
        lw    $s2, 0($t2)      # I9
        slt   $t3, $s1, $s2    # I10
        bne   $t3, $0, else    # I11
        add   $t9, $t9, $s1    # I12
        j     cont             # I13
else:   sub   $t9, $t9, $s2    # I14
cont:   addi  $t0, $t0, 4      # I15
        addi  $t1, $t1, 4      # I16
        addi  $t2, $t2, 4      # I17
        j     loop             # I18
end:
```

```
for (answer=0, k=0; k<size; k++) {
  if (A[k]>=B[k]) answer += A[k];
             else answer -= B[k];
}
```

Direct-mapped data cache with 512 words in total and each block contains 8 words.
You may assume:

- Array *A* starts at address 0xCDEF 0400.
- Array *B* follows immediately after array *A* in the memory.

(a) How many bits are there in the index field? Byte offset field?

Number of blocks = 512/8 = 64 = 2^6.
Therefore, index field: 6 bits.

Number of bytes in a block = 8×4 = 32 = 2^5.
Therefore, byte offset field: 5 bits.

Direct-mapped data cache with 512 words in total and each block contains 8 words.
You may assume:

- Array *A* starts at address 0xCDEF 0400.
- Array *B* follows immediately after array *A* in the memory.

Index: 6 bits
Byte offset: 5 bits

(b) Arrays *A* and *B* each has 1032 elements. What is the hit rate for arrays *A* and *B*?

Block								
32	A0	A1	A2	A3	A4	A5	A6	A7
33	B0	B1	B2	B3	B4	B5	B6	B7

Array *A* start addr = 0xCDEF 0400
= 0b … 0000 0**100 0000 0000**.
Therefore, *A*[0] resides in word 0 of block 32.

A[1024] to *A*[1031] will occupy block 32.
Therefore, *B*[0] will occupy word 0 of block 33.

```
for (answer=0, k=0; k<size; k++) {
  if (A[k]>=B[k]) answer += A[k];
              else answer -= B[k];
}
```

No cache trashing. For every 8 elements, the first is a miss and the rest are hits.
Therefore, hit rate is **7/8** for both arrays.

Direct-mapped data cache with 512 words in total and each block contains 8 words.

You may assume:

- Array *A* starts at address 0xCDEF 0400.
- Array *B* follows immediately after array *A* in the memory.

Index: 6 bits
Byte offset: 5 bits

(c) Arrays *A* and *B* each has 1028 elements. What is the hit rate for arrays *A* and *B*?

Block

32	A1024	A1025	A1026	A1027	B0	B1	B2	B3

Array *A* start addr = 0xCDEF 0400
= 0b … 0000 0**100 0000 0000**.
Therefore, *A*[0] resides in word 0 of block 32.

A[1024] to *A*[1027] will occupy the first 4 words of block 32, and *B*[0] to *B*[3] will occupy the next 4 words in the same block.

Therefore, for every 8 elements, the first 5 will be misses and the next 3 are hits, and this applies to the first 1024 elements. For the last 4 elements, they are all misses due to thrashing.
Hence, there are 3/8 × 1024 = 384 hits altogether for each array.
Therefore, hit rate = **384/1028**.

Snapshot of first 16 elements:

Block 32	**A0 (M)**	**A1 (M)**	**A2 (M)**	**A3 (M)**	**A4 (M) B0 (M)**	**A5 (H) B1 (M)**	**A6 (H) B2 (M)**	**A7 (H) B3 (M)**
Block 33	**B4 (M) A8 (M)**	**B5 (H) A9 (M)**	**B6 (H) A10 (M)**	**B7 (H) A11 (M)**	**B8 (M) A12 (M)**	**B9 (M) A13 (H)**	**B10 (M) A14 (H)**	**B11 (M) A15 (H)**
Block 34	**B12 (M)**	**B13 (H)**	**B14 (H)**	**B15 (H)**				

Direct-mapped data cache with 512 words in total and each block contains 8 words.
You may assume:

- Array *A* starts at address 0xCDEF 0400.
- Array *B* follows immediately after array *A* in the memory.

Index: 6 bits
Byte offset: 5 bits

(d) Assuming that arrays *A* and *B* have at least 100 elements each, (i) what is the lowest hit rate possible for array *A*? (ii) how many elements in array *A* would result in this lowest hit rate?

(i) Lowest hit rate is **0**. This occurs when there is cache trashing, i.e. the first elements of array *A* and array *B* are mapped to the same cache location.

(ii) When array *A* has a multiple of 512 elements (this is the simplest answer).

```
      add  $t0, $0, $0    # I1
      add  $t9, $0, $0    # I2
      addi $t1, $a1, 0    # I3
      addi $t2, $a2, 0    # I4
      sll  $t8, $a0, 2    # I5
loop: slt  $t3, $t0, $t8  # I6
      beq  $t3, $0, end   # I7
      lw   $s1, 0($t1)    # I8
      lw   $s2, 0($t2)    # I9
      slt  $t3, $s1, $s2  # I10
      bne  $t3, $0, else  # I11
      add  $t9, $t9, $s1  # I12
      j    cont           # I13
~~else: sub  $t9, $t9, $s2  # I14~~
cont: addi $t0, $t0, 4    # I15
      addi $t1, $t1, 4    # I16
      addi $t2, $t2, 4    # I17
      j    loop           # I18
end:
```

```
for (answer=0, k=0; k<size; k++) {
  if (A[k]>=B[k]) answer += A[k];
              else answer -= B[k];
}
```

2-way set associative instruction cache with **LRU** replacement policy. You may assume:

- Array *A* and array *B* each contains **100 elements.**
- Array *A* contains +ve values, array *B* contains -ve values.
- Instruction I1 is stored at address **0x2B00 0FFC**.

(e) The instruction cache contains **16** words in total and each block contains **4** words.

(i) How many bits in the set index field? In the byte offset field?

Number of sets = 16/4/2 = 2 = 2^1.
Therefore, set index field: 1 bit.

Number of bytes in a block = 4×4 = 16 = 2^4.
Therefore, byte offset field: 4 bits.

```
       add  $t0, $0, $0     # I1
       add  $t9, $0, $0     # I2
       addi $t1, $a1, 0     # I3
       addi $t2, $a2, 0     # I4
       sll  $t8, $a0, 2     # I5
loop:  slt  $t3, $t0, $t8   # I6
       beq  $t3, $0, end    # I7
       lw   $s1, 0($t1)     # I8
       lw   $s2, 0($t2)     # I9
       slt  $t3, $s1, $s2   # I10
       bne  $t3, $0, else   # I11
       add  $t9, $t9, $s1   # I12
       j    cont            # I13
~~else: sub  $t9, $t9, $s2   # I14~~
cont:  addi $t0, $t0, 4     # I15
       addi $t1, $t1, 4     # I16
       addi $t2, $t2, 4     # I17
       j    loop            # I18
end:
```

2-way set associative instruction cache with **LRU** replacement policy. You may assume:

- Array *A* and array *B* each contains **100 elements.**
- Array *A* contains +ve values, array *B* contains -ve values.
- Instruction I1 is stored at address **0x2B00 0FFC**.

(e) The instruction cache contains **16** words in total and each block contains **4** words.

(ii) How many misses are there?

Addr of I1 = 0x2B00 0FFC = 0b… 1111 **1100**.
So I1 is stored in set **1**, word **3**.

1st iteration: I1 – I13, I15 – I18;
2nd – 100th: I6 – I13, I15 – I18; winding up: I6 and I7.

Set 0	**I2 (M)** **I18 (M)**	**I3 (H)**	**I4 (H)**	**I5 (H)**	**I10 (M)**	**I11 (H)**	**I12 (H)**	**I13 (H)**
Set 1		**I15 (M)**	**I16 (H)**	**I1 (M)** **I17 (H)**	**I6 (M)**	**I7 (H)**	**I8 (H)**	**I9 (H)**

6 misses in 1st iteration, thereafter all hits.

2-way set associative instruction cache with **LRU** replacement policy. You may assume:

- Array *A* and array *B* each contains **100 elements.**
- Array *A* contains +ve values, array *B* contains -ve values.
- Instruction I1 is stored at address **0x2B00 0FFC**.

(f) The instruction cache contains **16** words in total and each block contains **2** words. How many misses are there?

Set index: **2** bits
Byte offset: **3** bits

Set 0	**I2 (M)** **I18 (M)**	**I3 (H)**	**I10 (M)**	**I11 (H)**
Set 1	**I4 (M)**	**I5 (H)**	**I12 (M)**	**I13 (H)**
Set 2	**I6 (M)**	**I7 (H)**		**I15 (M)**
Set 3	**I16 (M)**	**I1 (M)** **I17 (H)**	**I8 (M)**	**I9 (H)**

Addr of I1 = 0x2B00 0FFC = 0b… 1111 1100.
So I1 is stored in set **3**, word **1**.

1st iteration: I1 – I13, I15 – I18;
2nd – 100th: I6 – I13, I15 – I18; winding up: I6 and I7.

10 misses in 1st iteration, thereafter all hits.

CACHE QUIZ

Cache Quiz Question 1

For a memory with size 4GB, each block is 32 bytes. The size of the cache is 32KB with the same block size. Fill in the following with respect to Direct Mapping Cache:

1. #bits for Offset: $\log_2(32) = \mathbf{5}$

   How many bytes per block? $32 = 2^{\mathbf{5}}$

2. #bits for Block number: $32 - 5 = \mathbf{27}$

   Memory size = 4GB = 2^{32} bytes. Hence address has 32 bits, out of which 5 bits are used for offset.

3. #bits for index: **10**

   How many blocks in cache? $32KB/32 = (32\times 2^{10})/32 = 2^{\mathbf{10}}$

4. #bits for Tag: $32 - 10 - 5 = \mathbf{17}$

   Tag size = address size – index size – byte offset size

5. #cache blocks: **1024**

   $2^{10} = 1024$

6. If the address is 0x12345678, then answer the following in binary E.g. 0010:

   1. Tag: **0b00010010001101000**
   2. Index: **0b1010110011**
   3. Offset: **0b11000**

0x12345678
= 0b 0001 0010 0011 0100 0|101 0110 011|1 1000

Cache Quiz Question 2

Consider a 4-way set associative mapped cache with block size 4KB. The size of the main memory is 16GB and there are 10 bits in the tag. Find the number of sets in the cache and the size of the cache.

Number of sets in cache (answer in 2^x format):	2^{12}
Cache size in MB:	64

Memory size = 16GB = 2^{34} bytes. Hence address has 34 bits.

Size of one block = 4KB = 2^{12} bytes. Therefore, byte offsets = 12 bits.

No. of set index bits = address size – tag size – byte offset size
= 34 – 10 – 12 = 12.

Therefore, number of sets = 2^{12}.

Size of each cache set = 4 $\times$ block size = 4 $\times$ 4KB = 16KB = 2^{14} bytes.
Therefore, cache size = $2^{12} \times 2^{14}$ bytes = 2^{26} bytes = 2^{6} MB = 64MB.

Cache Quiz Question 3

Given a fully set associative cache. The size of the main memory is 256MB, size of cache memory is 16KB and the block size is 256bytes. Find the number of bit for byte offset and tag.

Number of bits for offset: **8**

Number of bits for tag: 28 – 8 = **20**

Memory size = 256MB = 2^{28} bytes. Hence address has 28 bits.

Block size = 256 bytes = 2^{8} bytes.

FINAL EXAM

Final Exam (AY2024/25 semester 2)

3 May 2025, Saturday, 9 – 11am

MPSH6

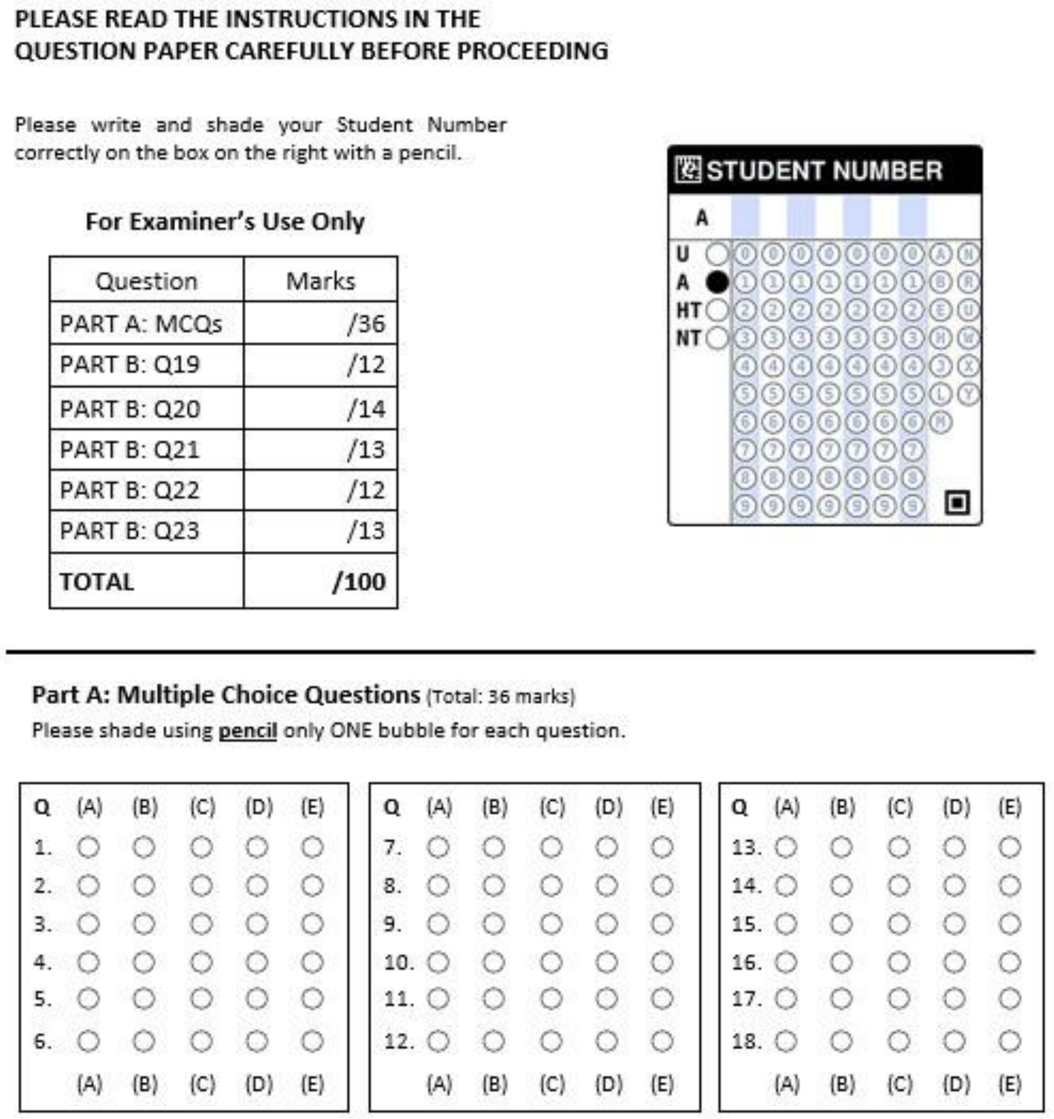
- Earlier topics (≈30%)
- Later topics (≈70%)

Format:

- 18 MCQs (36 marks)
- 5 multi-parts questions (64 marks)

Final Exam (AY2024/25 semester 2)

- Bring your pencils (2B or above), eraser, writing papers.
- Shade your Student Number correctly (and check!) using pencil.
- Shade the MCQ answers using pencil.
- Do NOT write your answers outside the boxes provided.



PLEASE READ THE INSTRUCTIONS IN THE QUESTION PAPER CAREFULLY BEFORE PROCEEDING

Please write and shade your Student Number correctly on the box on the right with a pencil.

For Examiner's Use Only

Question	Marks
PART A: MCQs	/36
PART B: Q19	/12
PART B: Q20	/14
PART B: Q21	/13
PART B: Q22	/12
PART B: Q23	/13
TOTAL	**/100**

STUDENT NUMBER

A

U	○	0	0	0	0	0	0	0	A	N
A	●	1	1	1	1	1	1	1	B	R
HT	○	2	2	2	2	2	2	2	E	U
NT	○	3	3	3	3	3	3	3	H	W
		4	4	4	4	4	4	4	J	X
		5	5	5	5	5	5	5	L	Y
		6	6	6	6	6	6	6	M	
		7	7	7	7	7	7	7		
		8	8	8	8	8	8	8		
		9	9	9	9	9	9	9		

Part A: Multiple Choice Questions (Total: 36 marks)

Please shade using pencil only ONE bubble for each question.

Q	(A)	(B)	(C)	(D)	(E)
1.	○	○	○	○	○
2.	○	○	○	○	○
3.	○	○	○	○	○
4.	○	○	○	○	○
5.	○	○	○	○	○
6.	○	○	○	○	○
	(A)	(B)	(C)	(D)	(E)

Q	(A)	(B)	(C)	(D)	(E)
7.	○	○	○	○	○
8.	○	○	○	○	○
9.	○	○	○	○	○
10.	○	○	○	○	○
11.	○	○	○	○	○
12.	○	○	○	○	○
	(A)	(B)	(C)	(D)	(E)

Q	(A)	(B)	(C)	(D)	(E)
13.	○	○	○	○	○
14.	○	○	○	○	○
15.	○	○	○	○	○
16.	○	○	○	○	○
17.	○	○	○	○	○
18.	○	○	○	○	○
	(A)	(B)	(C)	(D)	(E)

End of File