NAME: MUHAMMAD HAMZA
ROLL NO: BSSE23001-A
COAL ASSIGNMENT 04

QUESTION NO. 1:

DIRECT MAPPED:

- Each memory block maps to EXACTLY 1 cache block.
- Hardware is simplest with only 1 comparator per line.
- Access is fastest because of parallel tag comparisons
- Mapping is somewhat rigid which eventually causes most misses (as compared to other designs).

SET ASSOCIATIVES:

- Each memory block maps to a set means groups of blocks. It can be 2-way associative (means the group is of 2 blocks) or upto n-way (group is of n blocks)
- Hardware is of moderate level with n comparators per block (in case of n-way associative)
- A mechanism or policy is needed to replace any existing block (from

2 blocks in case of 2 way) to accomodate a new block. Most common mechanism used is Least resently used "LRU" policy.

- Misses are usually less than direct mapped.

FULLY ASSOCIATIVE:

- ANY memory block can go or placed in any cache block.
- Hardware is very complex because of a comparator for each block. (fully parallel)
- Hit rate is very high. Miss only occurs when data is not present in cache, or when the cache is full.
- Access is slowest became it has to do many comparisons.

Design	Hardware (complexity)	Hit Rate	Access Time
Direct-Map	Low	Low	Fastest
Set Associative	Moderate	Medium	Medium
Fully Associative	High	High	Slowest

PREFERRED SCENARIOS:

- Direct Mapped must be used when the need is for simple hardware

and speed and the working data is limited.

- Set Associative can be a general purpose design because of its good balancing. 2-way and 8-way are common.
- Fully Associative should be used when we need a small cache with high hit rate and have no worry for access time.

QUESTION NO. 2:

Given: Cache Size = 16KB = 2^{14} Bytes
Block Size = 32 bytes = 2^5 Bytes
Address Width = 32 bits
We know that

a) No. of Blocks in Cache = $\frac{\text{Cache Size}}{\text{Block Size}}$

=> No. of Blocks = $\frac{2^{14}}{2^5} = 2^9 = 512$

Block Offset INDEX = $\log_2$ (Block Size)

=> offset = $\log_2(2^5) = 5$

INDEX BITS = $\log_2$ (No. of Blocks)

=> I-B = $\log_2(2^9) = 9$

Tag Bits = Total Bits (Address width) − Offset Bits − Index Bits

=> Tag bits = 32 - 9 - 5 = 18 bits

b) ADDRESSES:-

0x00000000

Binary: 0000 0000 0000 0000 0000 0000 0000 0000 (tag | Index | offset)

Initial State (tag) at 0x0 = Empty (Now will change to 0x00000)

Hit / Miss => MISS

0x00000008

Binary: 0000 0000 0000 0000 0000 0000 0000 1000 (tag | Index | offset)

Prev State (tag) at Index 0x0 = 0x00000 ≈ 0x0

HIT / MIS => HIT (Tag Matched)

0x00000010

Binary: 0000 0000 0000 0000 0000 0000 0001 0000 (tag | Index | offset)

Prev State (tag) at Index 0x0 = 0x00000 ≈ 0x0

HIT / MIS => HIT (TAG Matched)

0x00000018

B: 0000 0000 0000 0000 0000 0000 0001 1000 (tag | Index | offset)

Prev State (tag) At Index 0x0 = 0x00000 ≈ 0x0

HIT / MIS => HIT (Tag Matched)

0x00000020

B: 0000 0000 0000 0000 0000 0000 0010 0000 (tag | Index | offset)

Prev tag at index 0x1 = Empty [Now will change to 0x00000 ≈ 0x0]

HIT / MIS => MISS

0x00000028

B: 0000 0000 0000 0000 0000 0000 0010 1000 (tag | Index | offset)

Prev tag at Index 0x1 = 0x00000 ≈ 0x0

HIT / MISS => HIT (Tag Matched)

0x00000030

B: 0000 0000 0000 0000 0000 0000 0011 0000 (tag | Index | offset)

Prev tag at Index 0x1 = 0x00000 ≈ 0x0

HIT / MISS => **HIT (tag Matched)**

0x00000038

B: 0000 0000 0000 0000 0000 0000 0011 1000 (tag | Index | offset)

Prev tag at Index 0x1 = 0x00000 ≈ 0x0

HIT / MISS => **HIT (tag Matched)**

0x00000040

B: 0000 0000 0000 0000 0000 0000 0100 0000 (tag | Index | offset)

Prev tag at Index 0x2 = ~~0x00000~~ EMPTY

(This will Now change to 0x00000 ≈ 0x0 after Miss Penality)

HIT / MISS => **MISS**

0x00000048

B: 0000 0000 0000 0000 0000 0000 0100 1000 (tag | Index | offset)

Prev tag at index 0x2 = 0x00000 ≈ 0x0

HIT / MISS => **HIT (Tag Matched)**

c) Total Accesses = 10

Total Hits = 7 ; Total Misses = 3

HIT RATE = $\frac{\text{HITS}}{\text{Accesses}} = \frac{7}{10} = .7 = 70\%$

QUESTION NO. 3:

Given: Total Size = 8KB = 2^3 Bytes

Block Size = 16 Bytes = 2^4 Bytes

Associativity = 2-way ; Address = 32 bits

Replacemed Policy = LRU

a) No. of Blocks = $\frac{2^{13}}{2^4} = 2^9 = 512$

No. of sets = $\frac{2^9}{2} = 2^8 = 256$

b) Block offset Bits = $\log_2(2^4) = 4$ bits

Set index bits = $\log_2(2^8) = 8$ bits

Tag bits = 32 - 8 - 4 = 20 bits

0x00001000 => Tag = 0x00001 ; I = 0x00 ; B-offset = 0x0

Previous tags in Set index 0x00

[Empty (one way) | Empty (2nd way)] (because 2-way).

MISS

Now Set Index 0x00 (after Miss) contains

[0x00001 (and other bits) | Empty]

0x00001004 => Tag = 0x00001 ; I = 0x00 ; B-off = 0x4

Prev tags in SET index 0x00

[0x00001 | Empty]

HIT - Tags Matched

0x00002000 => tag = 0x00002 ; I = 0x00 ; B-off = 0x0

Prev tags in SET index 0x00

[0x00001 | Empty]

No 0x00002 tag exists so

MISS

Now set Index 0x00 (after Miss) containing tags

[0x00001 | 0x00002]

0x00002004 => tag = 0x00002; I = 0x00; B-off = 0x4

Prev tags in SET Index 0x00

[0x00001 | 0x00002]

HIT - Tags Matched

0x00001000 => tag = 0x00001; I = 0x00; B-off = 0x0

Prev tags in SET Index 0x00

[0x00001 | 0x00002]

HIT - Tags Matched

0x00001004 => tag = 0x00001; I = 0x00; B-off = 0x4

Prev tags in SET index 0x00

[0x00001 | 0x00002]

HIT - Tags Matched

0x00002000 => tag = 0x00002; I = 0x00; B-off = 0x0

Prev tags in SET index 0x00

[0x00001 | 0x00002]

HIT - Tags Matched

0x00002004 => tag = 0x00002; I = 0x00; B-Off = 0x4

Prev tags in SET index 0x00

[0x00001 | 0x00002]

HIT - Tags Matched

Total Hits = 6 ; Total Miss = 2

c) Only SET at index 0x00 (means SET 0) is used which have the following 2 tags;

1st Way: 0x00001 & (other blocks bits)

2nd way: 0x00002 & (other blocks bits)

QUESTION NO. 4:

Given: Main Memory = 64M words = 2^{26} words

Cache size = 128K words = 2^{17} words

Block size = 8 words = 2^3 words

Word size = 4 bytes but we have both memories as word addressable so word size is irrelevent

Block Offset bits = $\log_2(2^3)$ = 3 bits

No. of Blocks = $2^{17}/2^3 = 2^{14}$ blocks

a) For direct Mapped

Index bits = $\log_2(2^{14})$ = 14 bits

Tag bits = 26 - 14 - 3 = 9 bits

Format: [9 bits (Tag) | 14 bits (Index) | 3 bits (offset)]

b) For fully associative (no index is req.)
Tag bits = 26 - 3 = 23 bits
Format: [23 bits (Tag) | 3 bits (offset)]

c) For 2-way associative
No. of sets = $2^{14}/2 = 2^{13}$
SET INDEX BITS = $\log_2(2^{13}) = 13$ bits
Tag bits = 26 - 13 - 3 = 10 bits
Format: [10 bits (Tag) | 13 bits (Index) | 3 bits (offset)]

d) For 8-way set associative
No. of sets = $2^{14}/8 = 2^{14}/2^3 = 2^{11}$
Tag bits = 26 - 11 - 3 = 12 bits
Format: [12 bits (Tag) | 11 bits (Index) | 3 bits (offset)]

QUESTION No. 5:
Given Direct Mapped cache with a 32-bit address, Bit division is as follows:-
Tag bits = 22 bits ; Index bits = 5 bits
offset bits = 5 bits.
(Assuming Memory is Byte Addressable)
a) From offset bits,
BLOCK SIZE = 2^5 = 32 bytes
As each word is 4 bytes then

Block Size = 32/4 = 8 words/block

b) From Index bits, we have
No. of Entries = 2^5 = 32 entries

c) Data Storage per block = 32×8 = 256 bits
Total storage per block = Data + Tag + Valid
=> 256 + 22 + 1 = 279 bits
Total Cache Storage = 32 × 279 = 8928 bits
Total Data Storage = 32 × 256 = 8192 bits
Ratio = 8928/8192 = 1.09

QUESTION NO. 6:

Given From previous question (Q5)
- Address is 32 bits with following divisions:- Tag = 22 bits ; Index = 5 bits
offset = 5 bits ; Block Size = 32 bytes
(means 1 block contains 0-31 byte range of data of respective to the index and particular tag)
No. of entries = 32 entries

=> Now analyzing each address

0 => 0000 0000 0000 0000 0000 0000 0000 0000 (Decimal to 32 bit Binary)

Tag = 0x000000 = 0x0 ; Index = 0x00 ; offset = 0x00
Initially the cache is empty so
MISS, After miss data placed at 0x0 index is 32 bytes referencing to Memory

Mem[0x00000000] – Mem[0x0000001F]

[0x00 → 0x00+(31)$_{10}$ = 0x1F]. Now the block at index 0x00 contains tag (0x0)

4 => 0000 0000 0000 0000 0000 0000 0000 0100

Tag = 0x0, Index = 0x00; offset = 0x00; HIT

Tag at Index (0x00) is 0x0 which matches

16 => 0000 0000 0000 0000 0000 0000 0001 0000

Tag = 0x0; Index = 0x00; offset = 0x10; HIT

Tag at Index 0x00 is 0x0 which matches

132 => 0000 0000 0000 0000 0000 0000 1000 0100

Tag = 0x0; Index = 0x04; offset = 0x04; MISS

After miss data placed at 0x04 index is 32 bytes referencing to Memory, Mem[0x00000080] to Mem[0x0000009F] [0x80 → 0x80+(31)$_{10}$ = 0x9F]

Now block at index 0x04 contains tag (0x0)

232 => 0000 0000 0000 0000 0000 0000 1110 1000

Tag = 0x0; Index = 0x07; offset = 0x08; MISS

After miss data placed at 0x07 index is 32 bytes referencing to Memory, Mem[0x000000E0] to Mem[0x000000FF] [0xE0 → 0xE0+(31)$_{10}$ = 0xFF]

Now block at index 0x07 contain tag (0x0)

160 => 0000 0000 0000 0000 0000 0000 1010 0000

Tag = 0x0; Index = 0x05; offset = 0x00; **MISS**

After miss data placed at index 0x07.

is 32 bytes referencing to Memory, Mem[0x000000A0]

to Mem[0x000000BF]. [0xA0 → 0xA0+(31)$_{10}$ = 0xBF]

Now index (0x05) contains tag (0x0)

1024 => 0000 0000 0000 0000 0000 0100 0000 0000

Tag = 0x1; Index = 0x00; offset = 0x00; **MISS**

Tags dont Match so the blocks data will be **REPLACED** from (Mem[0x00000000] to Mem[0x0000001F]) to (Mem[0x00000400] to Mem[0x0000041F])

30 => 0000 0000 0000 0000 0000 0000 0001 1110

Tag = 0x0, Index = 0x00; offset = 0x1E; **MISS**

Tags dont Match so the blocks data will be **REPLACED** from (Mem[0x00000400] - Mem[0x0000041F]) to Mem[0x00000000] to Mem[0x0000001F]

140 => 0000 0000 0000 0000 0000 0000 1000 1100

Tag = 0x0; Index = 0x04; offset = 0x0C; **HIT**

Tag at index 0x04 is 0x0 which matches

3100 => 0000 0000 0000 0000 0000 1100 0001 1100

Tag = 0x3; Index = 0x00; offset = 0x1C; **MISS**

Tags don't match so the data blocks will be REPLACED from (Mem[0x00000000] to Mem[0x0000001F]) to (Mem[0x00000C00] to Mem[0x00000C1F])

180 => 0000 0000 0000 0000 0000 0000 1011 0100

Tag = 0x0; Index = 0x05; offset = 0x14; HIT

tag at index (0x05) is 0x0 which matches

2180 => 0000 0000 0000 0000 0000 1000 1000 0100

Tag = 0x2; Index = 0x04; offset = 0x04; MISS

Tags don't Match so the data blocks will be REPLACED from (Mem[0x00000080] to Mem[0x0000009F]) to (Mem[0x00000880] to Mem[0x0000089F])

a) No. of blocks Replaced = 3 4 missed

b) Hit ratio = 4/12 = 33%

c) <0, 3, Mem[0x00000C00] - Mem[0x00000C1F]>

<2, Empty, Empty>; <3, Empty, Empty>

<4, 2, Mem[0x00000880] - Mem[0x0000089F]>

<5, 0, Mem[0x000000A0] - Mem[0x000000BF]>

<6, Empty, Empty>

<7, 0, Mem[0x000000E0] - Mem[0x000000FF]>

All others indexes are EMPTY.

QUESTION No. 7:

Given Physical address = 13 bits

4-way set associative cache

Block Size = 4 bytes ; Total Blocks = 32

a) => Offset bits = $\log_2(4)$ = 2 bits

=> No. of sets = $32/4$ = 8 sets

=> SET INDEX BITS = $\log_2(8)$ = 3 bits

=> Tag bits = 13 - 3 - 2 = 8 bits

b) 0x0D74 = 0000 1101 0111 0100

Block offset = $(00)_2$ = 0x0

Cache Index = $(101)_2$ = 0x5

Cache Tag = $(01101011)_2$ = 0x6B

From table, at Index (0x5), in all the ways the entries are either invalid or tags don't match so it will cause a **MISS.**

c) For way 0, the address must contain

tag: 0x06 => 00000110 ; Index: 7 => 111

Possible offsets: 00, 01, 10, 11

Thus 00000110 111 00 => 0x0DC

00000110 111 01 => 0x0DD

00000110 111 10 => 0x0DE

00000110 111 11 => 0x0DF

are the addresses for way 0

For way 1, Addresses must contain
Tag: 0x1D => 00011101 ; Index: 7 => 111
Thus 000111011100 => 0x03BC
000111011101 => 0x03BD
000111011110 => 0x03BE
000111011111 => 0x03BF
can be the possible addresses for way 1
For way 2, Addresses are not invalid
For way 3, Address must contain
Tag: 0x0D => 00001101 ; Index: 7 => 111
Thus 000011011100 => 0x01BC
000011011101 => 0x01BD
000011011110 => 0x01BE
000011011111 => 0x01BF
can be possible valid addresses for way 3

QUESTION No. 8:

1- Write-Through

- In this method, every write operation is simultaneously written to both cache and main memory.

Advantage:

- Simple structure.
- Main memory is always up-to-date which leaves no conflict of coherence.
- Useful in multi-core or multiprocessor systems.

Disadvantages:

- Writing is slow as both cache and main memory gets written simultaneously.

Example: A CPU writes the value 50 to address (0x1000). Both the cache and main memory immediately reflect this change.

2- Write-Back

- Data is written in cache only at initial phase. Main memory is only updated when cache block is replaced.

Advantage:

- Memory accesses is not frequent so writing is fast because usualy only cache is being written.
- Reduces constantly usage of Memory.

Disadvantage:

- More complex design because it needed an extra "dirty bit" to show chaged cache block.
- Data in Memory may not be coherent and become outdated.

Example: A CPU writes 60 to address: 0x1000. Cache stores 60 and on the dirty bit. Data is stored back to memory only at replacement.

3. Write Buffer

- It is a queue like buffer that holds data waiting to be written to memory, allowing the CPU to continue while the write buffer completely fills.

Advantages:

- It reduces CPU stalling because now it only stalls when buffer is full.
- It improves throughput.

Disadvantage:

- Hardware becomes complex.

Example: Write through cache uses a write buffer so CPU isn't blocked while data is beign written to main memory.

4- Write Allocate

- On a write miss, the block is first loaded into the cache and then the write is performed.

DisAdvantage:

- Causes unnecessary reads on write miss.

Advantage:

- Temporal locality becomes ensured.

=> It is often used with write back caches.

5. No-Write-Allocate

- On write miss, data is written directly to the memory, without loading to the cache.

Advantage:

- Avoids overhead of pulling data into the cache.

Disadvantage:

- Temporal locality is not ensured so effect performance.

=> It is often used with Write-through caches.

QUESTION No. 9

In specific cases direct-mapped caches can OUTPERFORM fully associative caches.(with LRU)

It's because in fully associative caches data can be placed anywhere but according to policy. while in direct mapped data knows where to go (modulo functionality)

Example:

- Assume 2 block cache, it is fully associative and have A, B initially.

ACCESS Pattern: C, A, B, C, A, B, C, ...

→ C comes in → A is replaced by C
→ A comes in → B is replaced by A
→ B comes in → C is replaced by B
So a loop of Misses gets continued
In contrast if it was Direct Mapped cache, and A can go to Index 0 ; B to Index 1 ; C to index 0. then let A & B in the cache
→ C comes in → A is replaced by C
→ A comes in → C is replaced by A
→ B comes in → HIT
And this system continues. It is better than fully associative because it gives at least one Hit / loop.