Zunaira Abdul Aziz
BSSE2305 8.
section A.

Assignment #4
COAL

Q1

- Direct Mapped Cache maps each memory block to a specific cache block/line, based on the memory address in the main memory. It uses a simple mapping method where the index bits of the memory address determines the cache block. The tags are compared with the cache and memory addresses. This makes it very fast and easy to implement because only one comparison is needed to check if the data is in the cache. However the main drawback is that it has a higher conflict misses, meaning different memory addresses that map to the same cache block will replace each other leading to poor performance if there are many collisions. This design is ideal for systems that prioritize speed and simplicity such as embedded devices or small (simple) cache. Used in microcontrollers (like printers etc)

- Set Associative Cache ~~improvement~~ is better than the direct mapped cache. It divides the cache into multiple sets, with each set containing the same no of cache blocks. A memory block is mapped to a set, but can be placed in any block within that set. This reduces conflict misses since

multiple blocks within the set. However, it requires more comparators, one per cache block in the set making it more complex than direct mapped cache. The hit rate also improves as there is a higher chance that the memory block will be found in one of the available blocks in the set. These caches are usually used in general purpose processors where a balance b/w performance and hardware cost is needed. Access time is also moderate since the set may need to check multiple blocks before finding the data. Used in Intel cores etc.

- Fully Associative cache. don't have any restrictions. A memory block can be placed anywhere in the cache. any block. This design has the best hit rate because there are no fixed locations and the cache can store any block in any line, However it requires more hardware because it needs to compare the memory address with all the caches tags (one for each line) making it the slowest of the three design. It needs a more complex replacement policy eg LRU or Random, to decide which data to replace when the cache is full. (also used by set Associative Cache); These are used in small, performance critical components like TLBs, where avoiding cache misses is more important than the hardware cost or speed.

Q2 a. Cache size = 16 KB = 16 × 1024 = 16384 bytes

Block size = 32 bytes

Address width = 32 bits.

$$\text{no of blocks} = \frac{\text{Cache size}}{\text{Block size}} = \frac{16384}{32}$$

$$= 512 \text{ blocks.}$$

b. Tag | index | offset.

From Block size

$$\text{Offset bits} = \log_2(32) = 5 \text{ bits}$$

$$\text{Index bits} = \log_2(\text{no of blocks}) = \log_2(512) = 9 \text{ bits}$$

$$\text{Tag bits} = 32 - (5 + 9)$$

$$= 18 \text{ bits.}$$

(1) 0x00000000 $\Rightarrow$ $^{31}00\cdots00^{0}$

- Tag = (18 zeros) $^{31}0\cdots0^{14}$
- Index = $^{13}0\cdots0^{5}$
- Offset = $^{4}0\cdots0^{0}$
- Miss (As cache empty).

Miss.

(2) 0x00000008 $\Rightarrow$ $^{31}00\cdots00\cdots1000^{0}$

- Tag = $^{31}0\cdots0^{14}$
- Index = $^{13}0\cdots0^{5}$
- Offset = $^{4}01000^{0}$

Hit (As. Index has the same. Tag as 0x0...00).

③ 0x00000010 => 00 10000

Tag = 0 ... 0

Inden = 0 ... 0

Offset = 10000

Hit (As Inden has the same tag as 0x00...00)

(Data will be replaced)

④ 0x00000018 => 00 ... 11000

Tag = 0 ... 0

Inden = 0 ... 0

Offset = 11000

Hit (same Tag as 0x0...10)

⑤ 0x00000020 => 00 ... 100000

Tag = 0 ... 0

Inden = 0 ... 01

Offset = 00000

Miss. (Nothing at inden 0...01)

⑥ 0x00000028 => 00 ... 101000

Tag = 0 ... 0

Inden = 0 ... 1

Offset = 01000

Hit (As Inden has the same tag as 0x0...20)

(Data will be replaced.

⑦ 0x00000030 => 00 ... 110000

Tag = 0 ... 0

Inden = 0 ... 1

Offset = 10000

Hit (As Inden has the same tag as 0x00-30)

(Data will be replaced)

(8) 0x00000038 => 0^{31} ... 111000

Tag = 0^{31} ... 0^{14}

Index = 0^{13} ... 1^{5} Hit.

offset = 11000^{0}

(9) 0x00000040 => 0^{31} ... 01000000

Tag = 0^{31} ... 0^{14}

Index = 0^{13} ... 10^{5} Miss.

offset = 01000^{0}

(10) ~~0x000000~~ 0x00000048 => 0^{31} ... 1001000

Tag = 0^{31} ... 0^{14}

Index = 0^{13} ... 10^{5} Hit. (same as 0x...4 Data will be replaced.

offset = 0000^{0}

C. Hit Rate = ?

$$\text{Hit Rate} = \frac{\text{No of hits}}{\text{Total no. of accesses}} = \frac{7}{10} = 0.7$$

70% Hit Rate.

Q3 2 set Associative cache

Total size = 8KB = 8x1024 = 8192 bytes

Block size = 16 bytes.

32 bit memory address

a.

No. of blocks = $\frac{\text{Cache size}}{\text{Block size}}$

= $\frac{8192}{16}$ = 512 blocks.

No of sets = $\frac{\text{Total Blocks}}{\text{Blocks per set}}$ = $\frac{512}{2}$ = 256 sets.

b.

① 0x00001000 => 0 ---- 1000000000000

Offset bits = $\log_2(16)$ = 4 bits.

Set Index bits = $\log_2(256)$ = 8 bits.

Tag bits = 32 - (4 + 8) = 20 bits.

Tag bits = 31 0 ..... 12 1

Index bits = 00000000 : Set = 0. Miss.

offset bits = 3 0000 0

(2). 0x2000 1004 => 0.....10000000001004

Tag: 0 ---- 1

Index. 0 ---- 0 Set=0 Hit

offset. 0100.

(3) 0x00002000 => 0 ---- 10000000000000

Tag = 0 ---- 10

Index = 0 ---- 0 Set=0. Miss

offset = 0000 (Diffuent Tag)

(4) 0x00002004 => 0 ---- 10000000000100

Tag = 0...10

Index = 0 ---- 0 Set=0 Hit.

offset = 0100

(5) 0x00001000 => 0 ---- 10000000000000

Tag = 0 ---- 1

Index = 0 ---- 0 Set=0 ~~Miss~~ Hit.

offset = 0000. ~~(Diffuent Tag)~~

(6) 0x20001004 => 0 ---- 10000000000100.

Tag = 0 ---- 1

Index = 0 ---- 0 Set=0 Hit.

offset = 0100.

⑦ 0x00002000

Tag : 0 — 10

Index : 0 — 0 Set 0 Hit

offset : 0000

⑧ 0x00002004

Tag : 0 — 10

Index : 0 — 0 Set 0 Hit

offset : 0100.

C.

Block 1 & Set 0 & Block 2

① 0x00001000 Miss | Tag 1 | Empty.

② 0x00001004 Hit (Most recently used) | Tag 1 used again | Empty.

③ 0x00002000 Miss. | Tag 1 | Tag 2.

④ 0x00002004 ~~Miss~~ Hit (Most recently used) | Tag 2 used again | Tag 1. → LRU

⑤ 0x00001000 Hit. | Tag 1 | Tag 2. → LRU

⑥ 0x00001004 Hit | Tag 1 | Tag 2. → LRU

⑦ 0x00002000 Hit. | Tag 2 | Tag 1. → LRU

⑧ 0x00002004 Hit | Tag 2 | Tag 1

Final State

Block 1 : Tag 1 (Most recently used)

Block 2 : Tag 2. (Least Recently used)

2 blocks present.

$2^{(26)}$ → 26 memory address bits

Q4 Main Memory = 64M words $64 \times 2^{20} = 2^{26}$ words

Cache Size = 128k words = $128 \times 1024 \times 4$

Block Size = 8 words = 8×4

Word size = 4 bytes

a. Direct Mapped cache.

Tag | Index | Offset

No of blocks = $\frac{128 \times 1024 \times 4}{8 \times 4} = 16384$

Index = $\log_2(16384) = 14$ bits

Offset = $\log_2(8) = 3$ bits

Tag = $26 - (14 + 3) = 9$ bits

b. Fully Associative Cache.

Tag | Offset

Offset = $\log_2(8) = 3$ bits

Tag = $26 - 3 = 23$ bits.

c. 2 Way set Associative Cache

Tag | Set Index | Offset

Offset = 3 bits

No of sets = $\frac{\text{No of blocks}}{\text{Blocks per set}} = \frac{16384}{2} = 8192$

Index set = $\log_2(8192) = 13$ bits.

Tag = $26 - (13 + 3) = 10$ bits.

d) 8 way set Associative cache.

No of sets = $\frac{16384}{8} = 2048$

Index = $\log_2(2048) = 11$ bits

Offset = 3 bits

Tag = $26 - (3 + 11)$

= 12 bits.

Q5

Tag 31-10	Index (9-5)	offset (4-0)
22 bits.	5 bits.	5 bits

(a) Cache block size. depends on offsets.

Cache block size = 2^5 = 32 bytes / 4

= 8 words

b. Entries. = ?

= No of blocks. = ?

Entries = 2^5 = 32 blocks

c. Index | V | Tag | Data.

The cache has 32 blocks.

Each block stores 32 bytes.

Total bits : $2^n \times$ (Block size + tag size + valid field size)

$= 32 \times (32 \times 8 + 22 + 1)$

$= 8928$

Data Storage bits : $32 \times 32 \times 8$

$= 8192$.

Ratio : $\frac{\text{Total bits}}{\text{storage bits}} = \frac{8928}{8192}$

$= 1.09$

Q6 a.

① 0 => 31 ... 0

Tag	Index	Offset	
31 ... 10	9 ... 5	4 ... 0	
0 ... 0	0 ... 0	00000	Miss (Inserted At Index 0. Tag 0)

② 4 => 31 ... 0100

0 ... 0	0 ... 0	00100	Hit (found at index 0)

③ 16 => 0 ... 10000

Tag	Index	Offset	
0 ... 0	0 ... 0	10000	Hit. (found at index 0)

④ 32 => 0

Tag	Index	Offset	
0 ... 0	00100	00000	Miss (Inserted at Index 4 Tag 0

⑤ 232

0 – 0 | 00111 | 01000 Miss (Inserted at Index 7 Tag 0)

⑥ 160

0 – 0 | 00101 | 00000 Miss (Inserted at Index 5 Tag 0)

⑦ 1024.

R 0 – 1 | 00000 | 00000 Miss (Replace at Index 0 Tag 1)

⑧ 30

R 0 – 0 | 0 – 0 | 11110 Miss (Replace at Index 0 Tag 0)

⑨ 140

0 – 0 | 00100 | 01100 Hit ~~(Replace)~~

⑩ 3100

R 0 – 11 | 00000 | 11100 Miss (Replace at Index 0 Tag 3)

⑪ 180

0 – 0 | 00101 | 10000 Hit

⑫ 2180

R 0 – 10 | 00100 | 00100 Miss (Replace At Index 4 Tag 2)

4 blocks replaced.

b. Hit Ratio = ?

$= \frac{4 \text{ Hits}}{12} = 0.333$

33.3% Hit Ratio.

c.

	Index	Tag	Data
~~0~~	0	0	30
~~Index 1 empty.~~			
~~3~~ 4	2	2	180
5	0	1	60
7	0	2	32

Q2 a. 13 bit Physical address

Block offset = 1 word — 4 bytes = 2^2 = 2 bits

Cache Index = ?

No of blocks = 32

No of sets = $\frac{\text{No of blocks}}{\text{Blocks per set}}$

$= \frac{32}{4} = 8$ sets

Cache Index = $\log_2(8)$ = 3 bits

Tag bits = 13 − (2 + 3)

= 8 bits.

b. 0x0D74

0	D	7	4
0000	1101	0111	0100

Block Offset bits = 2 bits
Index = 3 bits.
Tag = 8 bits.

Tag	Index.	Offset
01101011.	101	00
107	5	0

Miss As. nothing in set 5.

c.

Set 7	Tag	
Way 0	0x06.	1
Way 1	0x1D	1
Way 2	Invalid	0.
Way 3	0x0D.	1.

0x06. Offset = 2 Tag = 8 Index 3.

~~00000000~~ | 001 | 10

Set 7 Index 111

Tag 0x06 = 00000110

0x06

Tag	Index	offset	
00000110	111	00	0x00DC
=	=	01	0x00DD
=	=	10	0x00DE
=	=	11	0x00DF

0x1D

Tag	Index	offset	
0001 1101	111	00	0x03BC
=	111	01	0x03BD
=	111	10	0x03BE
=	111	11	0x03BF

0x0D

Tag	Index	offset	
0000 1101	111	00	0x01BC
=	111	01	0x01BD
=	111	10	0x01BE
=	111	11	0x01BF

Q8 Write through. => whenever cache is written main memory is also written.

Advantage: - Cache and memory are always in sync.

- They is good & simple for multicore CPU's as they will not cause

Disadvantage: - It is slower without write buffer.
as every write must wait for memory.
- This causes stalls.

→ Write back - writes cache at first, if cache block is to be replaced then main memory is updated.

Adv: writes faster (no waiting) and less memory traffic (good performance)

Dis: Risk of memory loss if power ignored before writing back
Harder to manage for multi-core processors.

Q9

Yes in some cases only, depending on access pattern.
~~In Direct~~ Fully Associative caches reduce conflict misses while direct mapped cache are more prone to them.
but not all the time.

Cache - 2 blocks Memory

A - Block-Index 0
B - Block-Index 1
C - Block-Index 0

↳ In Direct mapped Cache

Access			
A	A		Miss
B	A B		Miss
A	A B		Hit
~~In Direct~~ B	A B		Hit
C	C B		Miss
A	A B		Miss
B	A B		Hit

Total Hits = 3 Miss = 4

In full associative cache.

Access	Cache	Replace	M/H
A	A	-	miss
B	AB	-	miss
A	AB	-	Hit
B	AB	-	Hit
C	BC	A	miss
A	AC	B	miss
B	BC	A	miss

Total Hit = 2 miss = 5

Direct Cache Hit > Fully Associative cache Hit

It all depends on the access pattern.