# HW-2

① @x[256] = AAAA0000
@y[256] = AABB0000
@a[512] = AAAA8000
@b[512] = AABB8000
|element| = 8.
b = 8, s = 8, E = 4.

```
for i=0 to 255
    res1 = x(i) * y(i)
    for j = 0 to 1
        res2 = a(2i+j) * b(2i+j)
```

(a) m = 32

$C = B \times E \times S$
$= 2^8 \times 2^2 \times 2^8$
$= 2^{18}$
$= 256$ KBytes

$m = t + b + s$
$32 = t + 8 + 8$
$t = 16$

Thus, cache parameters are: (m, C, B, E, S, t, b, s)
= (32, 256kBytes, 256, 4, 256, 16, 8, 8) Ans

(b) Hit rate for x, y, a, b

Access pattern in the given code is:
x(0) * y(0)
x(1) * y(1) ⇒ this exhibits spatial locality
⋮

Each element is 8 bytes & each cache line can hold 256 bytes ⇒ $\frac{256}{8}$ = 32 elements

Given that each set has 4 lines

⇒ x(0) = 1010 1010 1010 1010 | 0000 0000 | 0000 0000

set(0)

say, Line 1 cache

⇒ y(0) = 1010 1010 1010 1010 | 0000 0000 | 0000 0000

set(0) (Different tag bits)

say, Line 2 cache.

⇒ a(0) = 1010 1010 1010 1010 | 1000 0000 | 0000 0000

set(128)

say cache line 1

⇒ b(0) = 1010 1010 1011 1011 | 1000 0000 | 0000 0000

set(128)

say cache line 2.

Now that each cache line can hold 32 elements

⇒ x(0..31) → will be in set 0 || line 1

y(0..31) → will be in set 0 || line 2

x(32..63) → set 1

y(32..63) → set 1

x :

y :

x :

x(224-255) → set 7

y(224-255) → set 7.

2

So, hit rate for x & y each will be

$\frac{31}{32} \Rightarrow 97\%$ Ans

Similarly, the access pattern for a[] & b[] is

a(0) b(0)
a(1) b(1)
⋮ ⋮
a(511) b(511)

⇒ This also exhibits good spatial locality

a(0..31) } set 128
b(0..31) }

a(32..63) } set 129
b(32..63) }

⋮

a(480-511) } set 143.
b(480-511) }

Thus hit rate = $\frac{31}{32} \Rightarrow 97\%$ Ans

Hit rate x ⇒ 97%

y ⇒ 97%

a ⇒ 97% Ans

b ⇒ 97%

(c) Total # of Miss =

$x_0, x_{32} \ldots x_{224} = 8$

$y_0, y_{32} \ldots y_{224} = 8$

$a_0, a_{32} \ldots a_{480} = 16$

$b_0, b_{32} \ldots b_{480} = 16$

$\Rightarrow 8 + 8 + 16 + 16$

$= 48$

Total # of access = $256 + 256 + 512 + 512$

$= 1536$

Hit Rate = $1 - \frac{48}{1536} \approx 97\%$

(d) after $i = 0$ is executed:

Set 0

| | valid | |
|---|---|---|
| line1 | 1 | x(0..31) |
| line2 | 1 | y(0..31) |
| line 3 | 0 | |
| line 4 | 0 | |

Set 128

| | | |
|---|---|---|
| line1 | 1 | a[0..31) |
| line2 | 1 | b(0..31) |
| line 3 | 0 | |
| line 4 | 0 | |

(e) After completion of outter loop, Set 0-7, each will have 2 cache lines populated. Each cache line will hold 32 elements of x or y. Set 128-143, each will have 2 cache lines populated. One cache line will hold 32 elements of a & other will hold 32 elements of b

~~set 0~~ <showing only valid cache lines>

<showing sets that have atleast one valid cache line>

$S_0 \rightarrow$ x (0..31)
y (0..31)

$S_1 \rightarrow$ x (32..63)
y (32..63)

$S_2 \rightarrow$ x (64..95)
y (64..95)

$S_3 \rightarrow$ x (96..127)
y (96..127)

$S_4 \rightarrow$ x (128..159)
y (128..159)

$S_5 \rightarrow$ x (160..191)
y (160..191)

$S_6 \rightarrow$ x (192..223)
y (~~223~~ 192..223)

$S_7 \rightarrow$ x (224..255)
y (224..255)

$S_{1,28}$ → a (0..31)
b (0..31)

$S_{1,29}$ → a (32..63)
b (32..63)

$S_{1,30}$ → a (64..95)
b (64..95)

$S_{1,31}$ → a (96..127)
b (96..127)

$S_{1,32}$ → a (128..159)
b (128..159)

$S_{1,33}$ → a (160..191)
b (160..191)

$S_{1,34}$ → a (192..223)
b (192..223)

$S_{1,35}$ → a (224..255)
b (224,-255)

$S_{1,36}$ → a (256..287)
b (256..287)

$S_{1,37}$ → a (288..319)
b (288..319)

$S_{1,38}$ → a (320..351)
b (320..351)

$S_{139} \rightarrow$ a (352 .. 383)
b (352 .. 383)

$S_{140} \rightarrow$ a (384 .. 415)
b (384 .. 415)

$S_{141} \rightarrow$ a (416 .. 447)
b (416 .. 447)

$S_{142} \rightarrow$ a (448 .. 479)
b (448 .. 479)

$S_{143} \rightarrow$ a (480 .. 511)
b (480 .. 511)

(2) for 8 way set associative cache, $E = 8$
Lets say we have $s = 1 \Rightarrow S = 2$
Lets say block size is 2 & element size is 1
$\Rightarrow b = 1$ & $B = 2^1 = 2$.

If we use the three eviction bits as a counter that points to the next cache line to use/evict, that is, first in first out policy, our strategy will work well incase of spatial locality in data access pattern. Where as the downside is, this will ignore the temporal locality. That is, in case some element is accessed frequently, it will still be evicted on FIFO basis

@ x[256] = AA00

b = 1, s = 1, E = 8

Eviction bits
000

| | | | | | |
|---|---|---|---|---|---|
| x(0) = | 1010 | 1010 | 0000 | 0000 | set 0 |
| x(1) = | 1010 | 1010 | 0000 | 0001 | |
| x(2) = | 1010 | 1010 | 0000 | 0010 | set 1 |
| x(3) = | 1010 | 1010 | 0000 | 0011 | |
| x(4) = | 1010 | 1010 | 0000 | 0100 | set 0 |
| x(5) = | 1010 | 1010 | 0000 | 0101 | |
| x(6) = | 1010 | 1010 | 0000 | 0110 | set 1 |
| x(7) = | 1010 | 1010 | 0000 | 0111 | |
| x(8) = | 1010 | 1010 | 0000 | 1000 | set 0 |
| x(9) = | 1010 | 1010 | 0000 | 1001 | |
| x(10) = | 1010 | 1010 | 0000 | 1010 | |
| x(11) = | 1010 | 1010 | 0000 | 1011 | |
| x(12) = | 1010 | 1010 | 0000 | 1100 | set 0 |
| x(13) = | 1010 | 1010 | 0000 | 1101 | |
| x(14) = | 1010 | 1010 | 0000 | 1110 | |
| x(15) = | 1010 | 1010 | 0000 | 1111 | |

→ Consider only set 0

→ to begin with Eviction bits = 000

⇒ cache line 0 should be used to fill the new block

⇒ x(0..1) will go in line 0

Increment Eviction Bits ⇒ Eviction bits = 001

So, set 0

| | Data | Eviction bits |
|---|---|---|
| | | 000 |
| cache line 0 | x(0..1) | → 001 |
| cache line 1 | x(4..5) | → 010 |
| cache line 2 | x(8..9) | → 011 |
| cache line 3 | x(12..13) | → 100 |
| cache line 4 | x(16..17) | → 101 |
| cache line 5 | x(20..21) | → 110 |
| cache line 6 | x(24..25) | → 111 |

Now x(28..29) will go in cache line 0

Cache line → $x(28..29) \rightarrow \cdots \rightarrow 000$

Works well if Access pattern is:

AA00
AA01
AA02
⋮
AA1A
AA1B
AA10

Works poorly when:

AA00 → $x_0$
AA04
⋮
AA0F → $x_{15}$
⋮
AA1D → $x_{29}$
* AA00 (Hit)
AA0E (Hit)
AA21 (Miss) (Evict AA00 - AA01)
* AA00 (Miss) (Evict AA03 - AA04)

Had we taken into account that AA00 was recently used, we could have evicted AA03-AA04 (cache line 1) But with current policy, Eviction bits were 000 => Next line to be evicted was Line 0.

3. LRV - cache Miss

once we have determined the sets to scan:

(1) Scan 1 : Determine if the data already exists in the cache

Match the tag bits & valid bits for the requested address in each scanned cache line

(2) Scan 2 : if the requested address doesnot exist, => it is a Miss.

Scan to check if a cache line exists where valid bit is set to false.

If no such cache line is found, scan again to determine which cache line to evict

(3) Scan 3 : Now one cache line has to be evicted

Scan & look at eviction bits of each line & determine the oldest cache line in the set.

Once the LRU cache line is determined, the newly fetched data block is placed in this line.

If the memory controller has direct access/pointer to the cache line determined to be evicted, the new data line will be written there directly. Else an additional Scan will be required to look up the cache line with eviction bits (determined by the processing logic after Scan 3)

4. Let S = 1, E = 4, |B| = |Element|

case 1 : LRU

→ AA00 (Cold Miss), AA01 (Cold Miss), AA02 (Cold Miss), AA03 (Cold Miss)

→ AA02 (Hit), AA02 (Hit), ~~AA03 (Hit)~~, AA04 (Miss ⇒ Evicts AA00), AA03 (Hit), AA05 (Miss ⇒ Evict AA01)

→ AA06 (Miss ⇒ Evict AA02)

Although AA02 was accessed thrice & AA03 was accessed twice, still AA02 was evicted because AA03 was accessed more recently

case 2 : LFU

Had we applied LFU in the above sequence, AA06 would have evicted AA04

AA07 ⇒ Miss (Evict AA05)

AA08 ⇒ Miss (Evict AA06)

Even though we haven't accessed AA02 & AA03 recently, they haven't been evicted because their access freq count was 3 & 2 respectively.

Hence, when addresses are accessed frequently, same LFU works well.

When addresses are accessed sequentially, LRU works well.

5.

b = 8, s = 0, E = 256

@x[8192] = AAAA0000

@y[8192] = AAAB0000

|Element| = 8

```
for (i=0 to 8191)
    res1 = x[i] + y[i]
```

(a) Cache parameters (m, C, E, B, S, b, s, t)

Fully - Associative Cache.

m = 32

E = 256

B = $2^8$ = 256

S = 1

C = B x E x S = $2^8 \times 2^8 \times 1 = 2^{16}$ = 64K Bytes

t = m - s - b ⇒ 32 - 0 - 8 = 24

(m, C, B, E, S, b, s, t) = (32, 64K Bytes, 256, 256, 1, 8, 0, 24)

(b) Element size = $2^3$

Each cache line has $\frac{2^8}{2^3}$ Elements ⇒ 32 elements

Hit rate = $\frac{31}{32}$ = 97%

(c) Overall Hit rate = 97%

(d) After i = 8191

Assuming data goes in cache lines sequentially

There is only 1 set, 256 cache lines

32 Elements in each cache line

| | |
|---|---|
| $E_0$ : x (0..31) | x (4096-4127) |
| $E_1$ : y (0..31) | y (4096-4127) |
| | |
| $E_{254}$ : x (4064-4095) | x (8160-8191) |
| $E_{255}$ : y (4064-4095) | y (8160-8191) |

All cache lines will be valid.

Out of 256 cache lines, 128 lines will hold second half of array x ⇒ $x_{4096-8191}$

Other 128 lines will hold second half of array y ⇒ $y_{4096-8191}$. x & y will be in alternate cache lines

6.

Case 1

| | Hit cycles | Hit Rate |
|---|---|---|
| L1 | 2 | 25% |
| L2 | 10 | 80% |
| L3 | 80 | 95% |
| Main | 500 | |

$$AMAT_1 = 2 + \left(\frac{3}{4} \times AMAT_2\right) = 25.25$$

Rough: $2 + \frac{3}{4} \times 31$

$$AMAT_2 = 10 + \left(\frac{1}{5} \times AMAT_3\right) = 31$$

Rough: $10 + \frac{1}{5} \times 105$

$$AMAT_3 = 80 + \left(\frac{1}{20} \times 500\right) = 105$$

13

Case 2

| | Hit time | Hit rate |
|---|---|---|
| L1 | 2 | 50% |
| L2 | 5 | 65% |
| L3 | 20 | 80% |
| Main | 500 | |

$$AMAT = 2 + \left(\frac{1}{2} \times AMAT_2\right)$$

$$= 2 + \left(\frac{1}{2} \times \left\{5 + \frac{7}{20} \times AMAT_3\right\}\right)$$

$$= 2 + \left(\frac{1}{2} \times \left\{5 + \frac{7}{20}\left[20 + \frac{1}{5} \times 500\right]\right\}\right)$$

$$= 2 + \left(\frac{1}{2} \times \left\{5 + \frac{7}{20}\left[120\right]\right\}\right)$$

$$= 2 + \left(\frac{1}{2} \times \left\{5 + 42\right\}\right)$$

$$= 2 + \left(\frac{1}{2} \times 47\right)$$

$$= 25.5$$

Clearly, cache Hierarchy 1 is giving a better AMAT.