INDIAN INSTITUTE OF ENGINEERING SCIENCE AND TECHNOLOGY, SHIBPUR

B. TECH 5TH SEMESTER EXAMINATION, DECEMBER 2021

Computer ~~Arch~~ Architecture and Organization – II [CS 3103]

Name: Abhiroop Mukherjee
Examination Roll No.: 51051910 9
G-Suite ID: 51051910 9.abhiroop@students.iiests.ac.in

~~1, 2, 3, 4, 5, 6~~

Q.) (I) Write – Invalidate

→ In write-invalidate scheme, a write operation for some data (say X) in a block (say B) by a processing unit P_i does the following:

a) Invalidate all other cached copies of B [but not MM version]

→ in case ~~MM~~ cache is using write-through, update to X will be immediately ~~to~~ reflected in MM also

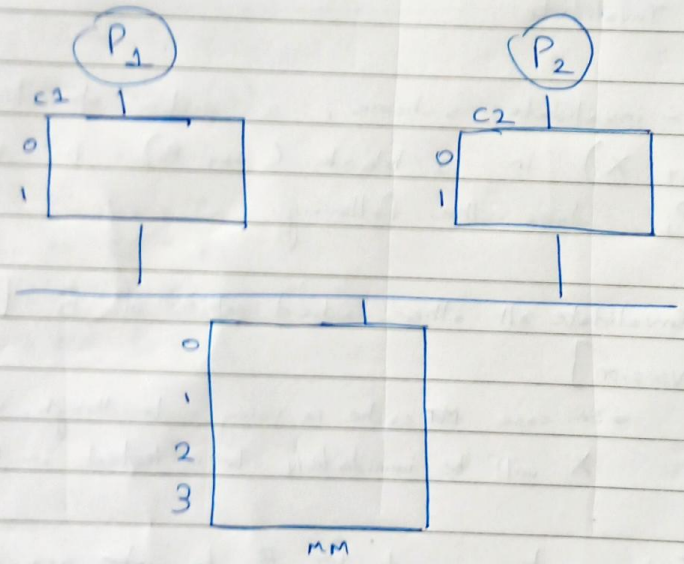
b) Subsequent request for B, by ~~this~~ any other processor other than P_i, will be treated ~~as~~ cache miss.

(II) Write update

→ An update of block B in P_i's cache, updates all cached copies of B & MM

→ Any subsequent request for B from any processor which held B before will be treated as hit.

Q2) ;;)



→ initial X = 20 (let us assume X is in loc 0 of MM)

→ caches initially empty & direct mapping implementation

I) write ~~update~~.

- before going into write update & write invalidate, some things are required to be addressed.

- (a) If X is in memory loc α (α can be 0, 1, 2, 3) in both C_1 and C_2 using direct mapping, X will be stored in α % 2 loc in cache.

→ ~~So same can~~

→ here we have ~~take~~ assumed that X will be in memory location 0, and hence X will be present in loc 0 in C1 & C2 also.

(b) Assuming write ~~back~~ through policy in cache

I) write ~~update~~. invalidate.

Activity	X at cache1	X at cache2	X at MM
→ initial			* 20
(1) P1 read X, read miss C1	20	–	20
(2) P2 read X, read miss C2	20	20	20
(3) P2 update X = 5, X at P1 invalidated, MM updated also due to write through policy	–	5	5
(4) P2 read X, read hit	20	–	20
(5) P1 update X = 20, X at C2 invalidated, write through to MM			
(6) P2 read X, read miss	20	20	20

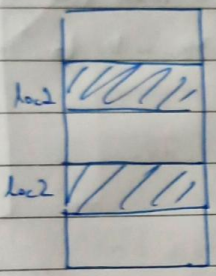
II write update

	Activity	X at C1	X at C2	X at MM
①	initial	–	–	20
①	P1 read X, read miss	20	–	20
②	P2 read X, read miss	20	20	–
③	P2 update X=5, broadcasted to everywhere	5	5	5
④	P2 read X, read hit	5	5	5
⑤	P1 update X=20, broadcast everywhere	20	20	20
⑥	P2 read X, read hit	20	20	20

Q3) (I) EREW SM SIMD computer

→ Exclusive Read Exclusive write memory
→ No two Processing element's are allowed to simultaneously read / write to same memory location.

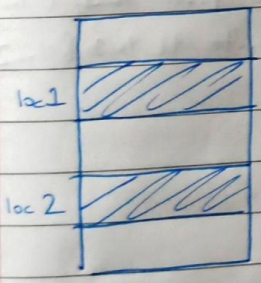
Eg



→ Simultaneous read from loc 1 & loc 2 ✓
→ Simultaneous write to loc1 & loc 2 ✓
→ simultaneous multiple read from loc 1 & 2 X
→ Simultaneous multiple write to loc 1 & 2 X

(II) CREW SM SIMD computer.

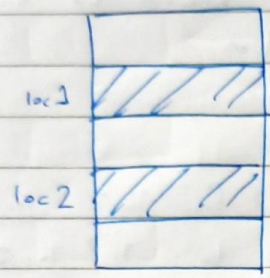
→ Concurrent read Exclusive write



simultaneous read from loc 1 & loc 2 ✓
simultaneous write to loc 1 & loc 2 ✓
simultaneous multiple read from loc 1 & loc 2 ✓
simultaneous multiple write from loc 1 & 2 X

III) ERCW SM SIMD computer

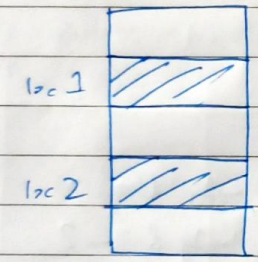
→ Exclusive read concurrent write



→ simultaneous read from loc 1 & loc 2 ✓
→ simultaneous write to loc 1 & loc 2 ✓
→ simultaneous multiple read from loc 1 & 2 ✗
→ simultaneous multiple write to loc 1 & 2 ✓

II) CRCW SM SIMD computer.

→ concurrent read concurrent write.



→ simultaneous read from loc 1 & loc 2 ✓
→ simultaneous write to loc 1 & loc 2 ✓
→ simultaneous multiple read from loc 1 & loc 2 ✓
→ simultaneous multiple write to loc 1 & loc 2 ✓

classmate
Date
Page

Time to replace elements n in table of n elements by 100 in an SIMD computer with N PE's

→ basic idea → search entire array ~~if there n~~ in

↓

if there is n in array

↓

~~retp~~ replace that n by 100

Eg: if table/array = {1, 2, 3, 4, n, 7, 8, n, n}

end result = {1, 2, 3, 4, 100, 7, 8, 100, 100}

→ Basic steps we will have to do in SIMD computer to achieve this

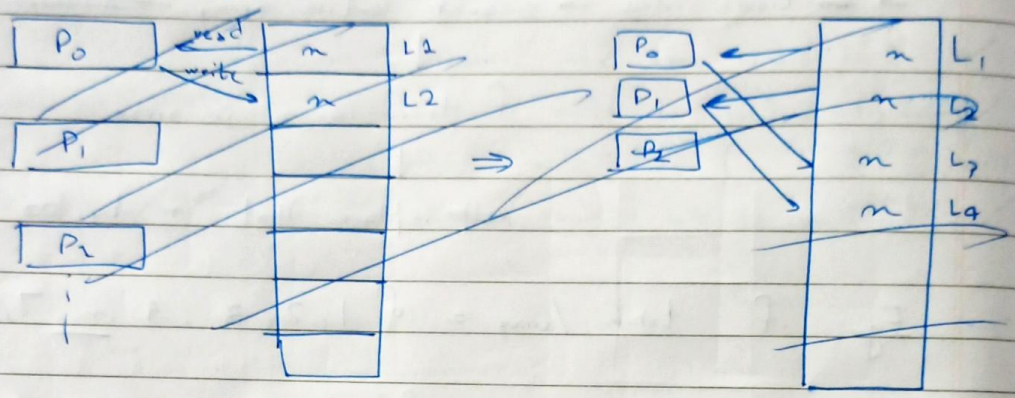
(a) Broadcast element to be searched (n) to all PE's

(b) all PE compare elements with n

(c) if n is found ~~the~~ by PE_i, it will write 100 to it's position., ie storing of result in memory

(I) ~~EREW~~ updating table in EREW

(a) Broadcasting



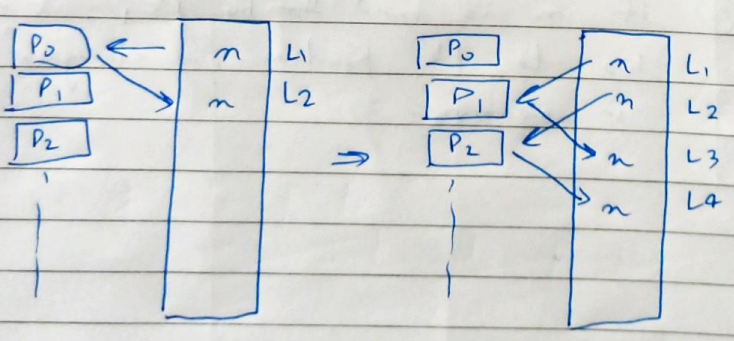
Procedure → Let n be in memory L_1

→ time step 1 → n read by P_0 from L_1 & then written to L_2

→ time step 2 → ~~n read by~~ P_1 & P_2 read n from L_1 & L_2 and write to L_3 & L_4

→ time step 3 → now n is in four loc

→ another four PE (3-6) read from $L_1 - L_4$ and writes in $L_5 - L_8$



→ using this method, we can broadcast n to all PE in $\log(N)$ time

(II) comparision of n with elements of table

→ if $N > n$ → will happen in $O(1)$

→ in case $N < n$, i.e no. of elem > no. of processing unit, each PE will compare $\frac{n}{N}$ elements with n.

→ ~~comparis~~ $\frac{n}{N}$ comparision inside a single PE happens sequentially & all PE executes parallely

so Time Complexity = $O\left(\frac{n}{N}\right)$

(III) Storing result in memory

→ if any element found n, PE will write 1 in it's location

→ in the worst case, all elements can be n, so ~~Time complexity = O(n)~~

→ here every PE can write simultaneous (different) location → TC = $O\left(\frac{n}{N}\right)$

∴ Total TC = $O(\lg N) + O\left(\frac{n}{N}\right) + O\left(\frac{n}{N}\right)$

if $n >> N$ → TC = $O(n)$

(II) CREW update.

(a) Broadcasting.

→ concurrent read → ~~one~~ broadcasting in $O(1)$

(b) comparision.

→ same as before $O(\frac{n}{N})$

(c) write update.

→ same as before $O(\frac{n}{N})$

$\therefore TC = O(1) + O(\frac{n}{N}) + O(\frac{n}{N})$

n ~~>>>~~ $n >>> N \rightarrow TC = O(n)$

(III) ERCW

(a) Broadcasting

→ same as ~~(II)~~ EREW → $O(\lg N)$

(b) comparision

→ same as before → $O(\frac{n}{N})$

(c) ~~Op~~ write update.

→ ~~concurrent write possible. , every PE can write simultaneously to d~~

→ same as before → $O(\frac{n}{N})$

$\therefore TC = O(\frac{n}{N})$ ~~N~~ $n >>> N \rightarrow TC = O(n)$

(IV) CRCW

(a) broadcast

→ same as CREW → $O(1)$

(b) comparision

→ same as before $O(\frac{n}{N})$

(c) write update.

→ same as before $O(\frac{n}{N})$

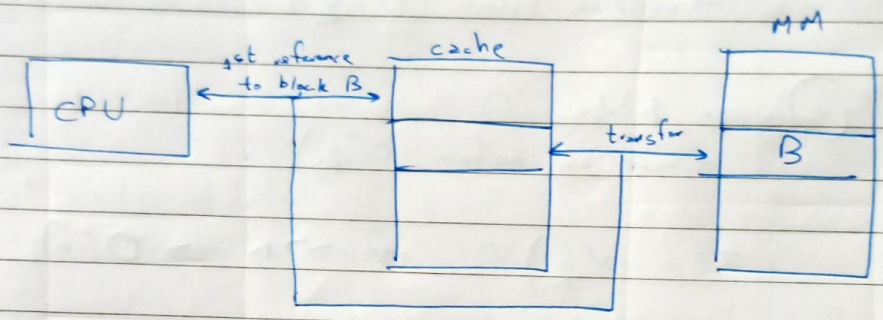
$\therefore T(=O(\frac{n}{N})$ ~~$n >> N$~~ $\Rightarrow O(n)$

Q5) (I) compulsory miss

→ The very first access to some block B will always be a miss

→ This miss is called compulsory miss.

→ This miss cannot be avoided.



(II) Conflict miss

→ when multiple memory blocks map to same cache frame, conflict happens and ~~exist~~ one frame is discarded

→ when access to discarded frame is made, it will result in a miss

→ that miss is called conflict miss.

capacity miss.

→ when cache capacity becomes full, some ~~blocks~~ caches will be ~~discarded~~ discarded back to MM for some other cache.

→ after discard, accessed to elements inside the discarded cache will be a miss

→ This miss is called capacity miss.

→ If we increase cache size [keeping block size constant]

→ no. of blocks inside cache increases, reducing chance of capacity miss.

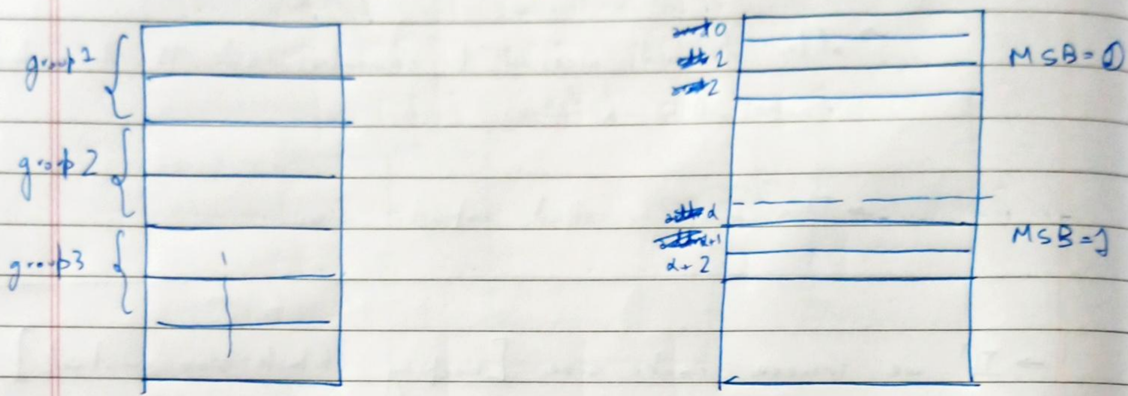
→ no. of compulsary miss will be constant

→ conflict miss ~~defend~~ depends on associativity

~~Eg for~~

Pseudo-Associative cache.

→ works very similar to 2-way set associative mapping, ~~the~~ only difference being as follows.



2 way set associative.

pseudo-associative.

→ in pseudo-associative cache ~~addr 0 and addr α~~ loc α & ~~loc forms as a group~~ loc β form a group

→ where α & β have ~~b~~ are equal, but MSB of α & β are different

→ this works faster than 2-way as bit addition is slower than inverting ~~a~~ only MSB

~~in 2 way → check if data present in cache α~~
~~→ then check if data pre~~

→ hit time of pseudo-associative almost same as direct mapped cache.

→ Pseudo-Associative cache reduces the time required for checking presence of frame, thereby reducing hit time.

Loop Fusion

(VI) given code.

```
for int i = 1 to 1000 with increment 1 :
    C[i] = B[i] + A[i]

for int i = 1 to 1000 with increment 1 :
    D[i] = B[i] * 2 * A[i] ;
```

let us assume two elements of each array A/B/C/D forms a block

let us see no. of misses for A → A has 1000 elements so 500 blocks.

let cache accomodate less than 500 blocks of A at a time. & assume cache replacement policy is first in first out

→ There will be 500 misses for A in first loop & then there will be 500 more misses for A in second loop

→ So total 1000 misses.

~~using loop fusion~~

(II) using loop fusion.

for ~~init~~ int i=1 to 1000 with increment 1 :

```
C[i] = B[i] + A[i];
D[i] = B[i] * 2 * A[i];
```

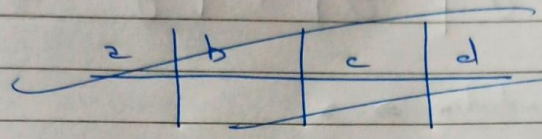
→ here only 500 misses will take place ~~as A[i] is~~ as ~~A[i]~~ in second line of loop A[i] will always be hit ~~.~~.

∴ Loop Fusion reduces miss rate by improving temporal locality

Q6) given sequence.

a a a b b c d c a a a b b c a a a

(I) 0th order.



Q6) 2) Context - based value predictor.

Context → sequence of n most recent values produced for n

→ classified according to order.

→ nth order model takes last n values & uses this for prediction.

(II) . given sequence

a. a a b b c d c a a a b b c a a a.

(I) 0th order.

a	b	c	d
9	4	3	1

prediction = a.

(II) 1st order.

	a	b	c	d
a	6	2	0	0
b	0	2	2	0
c	2	0	0	1
d	0	0	1	0

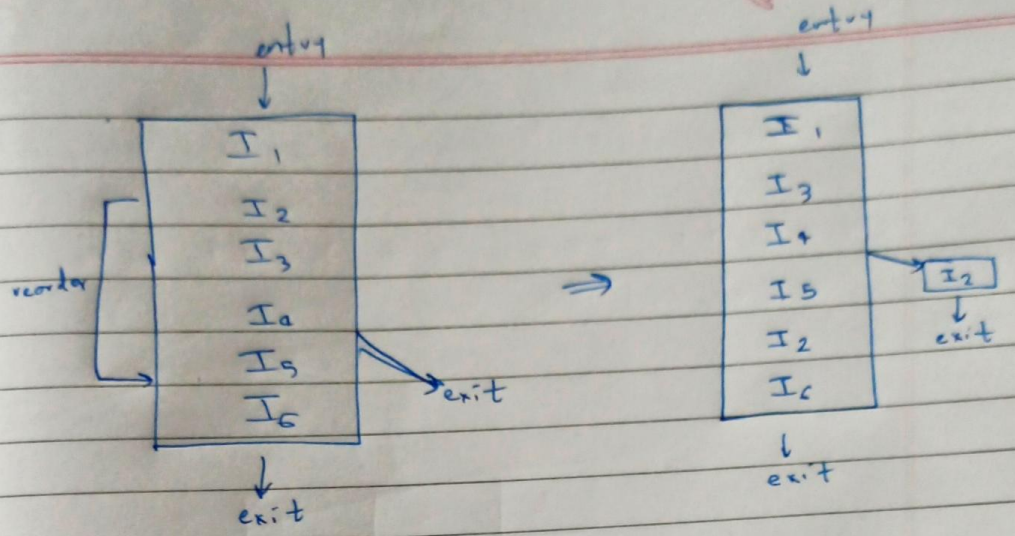
prediction ⇒ a

(III) 2nd order.

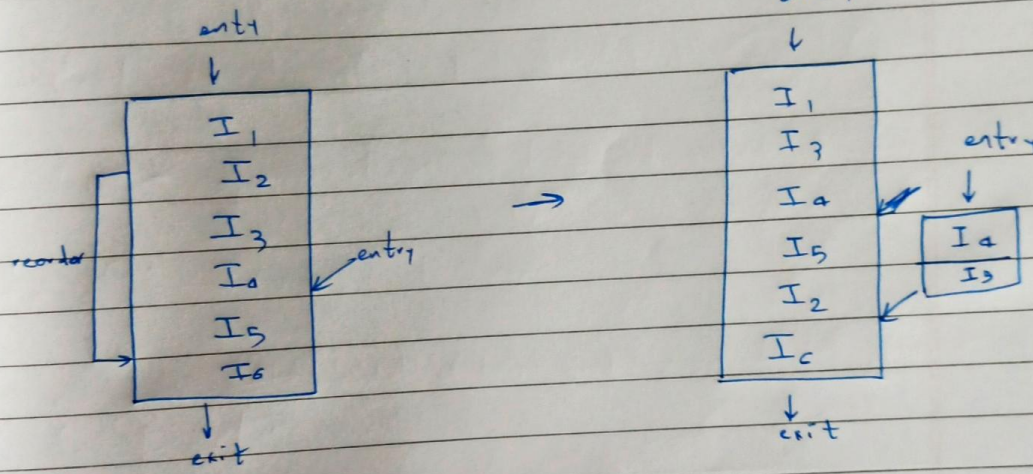
	a	b	c	d
aa	3	2	0	0
ab	0	2	0	0
ac	0	0	0	0
ad	0	0	0	0
ba	0	0	0	0
bb	0	0	2	0
bc	1	0	0	1
bd	0	0	0	0
ca	2	0	0	0
cb	0	0	0	0
cc	0	0	0	0
cd	0	0	1	0
da	0	0	0	0
db	0	0	0	0
dc	1	0	0	0
dd	0	0	0	0

prediction = a

Q6 b) A)



B)



Q6 c)