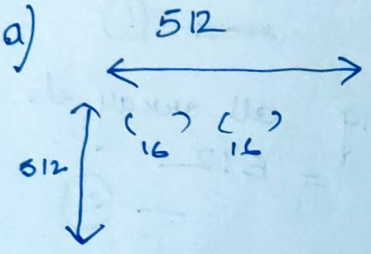
A1

32 KB Direct Cache $= 2^{15}$ Bytes.

Array element size $= 8$ Bytes

No of array element that could be in one block $= \frac{128}{8} = 16$

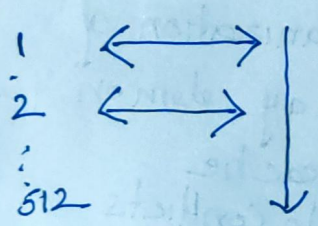
a)



When an array is accessed. 16 element of row caches

Thus for the first row of array no of times we have to fetch data to cache $= \frac{512}{16} = 32$

Thus



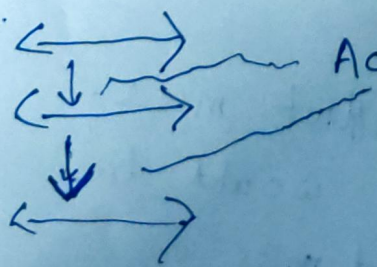
Miss

32
32
⋮

$32 \times 512 = M1$

(b)

Data is getting accessed column wise



Access pattern of P2

Thus every data access in P2 would result in miss

$$\frac{M1}{M2} = \frac{32 \times 512}{512 \times 512} = \frac{1}{16}$$

Answer no 2

Direct Cache Size = 8KB = 2^{13} Bytes

Block Size = 16 Bytes

No of array elements in one block = $\frac{16}{4} = 4$

No of lines in Cache = $\frac{2^{13}}{2^4} = 2^9 = 512$ — (1)

No of lines needed for caching all array ele = $\frac{2048}{4} = 512$ — (2)

Since (2) is not greater than (1) there would be no conflicts

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

:

A[2024] A[2025] A[2026] A[2027]

512

organization of array elemen in cache No conflicts

(B) No of Cache misses while accessing data in reverse order = forward order = no of lines = 512 *

* Since there is no conflict miss. every one would be cold miss

Answer no 3

2 Way set associative cache, hence 2 blocks per set

No of sets in cache = $\frac{256}{2} = 128$

Block indexing to cache is determined by.

$c = j \bmod m$

$j \rightarrow$ Memory block

$m \rightarrow$ No of sets.

(For First pass)

BNO mod 128

SNO	BNo	Indexing	Miss	SNo of Block evicted
1	0	0	Compulsory	N/A
2	128	0	Compulsory	N/A
3	256	0	Compulsory	(1)
4	128	0	No Miss	N/A
5	0	0	Conflict	(3)
6	128	0	No Miss	N/A
7	256	0	Conflict	(5)
8	128	0	No Miss	N/A
9.	1	1	Compulsory	N/A
10	129	1	Compulsory	N/A
11	257	1	Compulsory	(9)
12	129	1	No Miss	N/A
13	1	1	Conflict	(11)
14	129	1	No Miss	N/A
15	257	1	Conflict	(13)
16	129	1	No Miss	N/A

4 conflict Miss

According to definition

Compulsory miss and no conflict miss

After first pass set 0 has 128, 256 and set 1 has 129, 257 Blocks

(For second pass)

SNo	BNo	Indexing	Miss.	S/No of block evicted
1	0	0	Conflict	(7) from last table
2	128	0	No Miss	N/A
3	256	0	Conflict	(1)
4	128	0	Hit	N/A
5	0	0	Conflict	(3)
6	128	0	Hit	N/A
7	256	0	Conflict	(5)
8	128	0	Hit	N/A
9	1	1	Conflict	(15) from last table
10	129	1	Hit	N/A
11	257	1	Conflict	(9)
12	129	1	Hit	N/A
13	1	1	Conflict	(11)
14	129	1	Hit	N/A
15	257	1	Conflict	(13)
16	129	1	Hit	N/A

8 Conflict Miss

After second pass set 0 has 128, 256 and set 1 has 129, 257 blocks

Since the contents of cache and access pattern are same same 8 conflict in next iterations

Total no of Conflict Miss = (4) First + (9×8) second to tenth.

= 4 + 72

= 76

Answer no 4

i)

IF → ID → OF → EX → WB

5 → 4 → 20 → 10 → 3.

(2 under each arrow)

Clock speed of pipeline

= max(7, 6, 22, 12, 3)

= 22

Time taken to execute 20 instructions.

= (K + (n-1)) × Tp → clock speed

↓ stages

= (5 + 19) × 22 = 24 × 22

ii) IF → ID → OF → EX → WB.

5 → 4 → 12 → 8 → 10 → 3

(2 under each arrow)

Clock speed of pipeline = max(7, 6, 14, 10, 12, 3) = 1

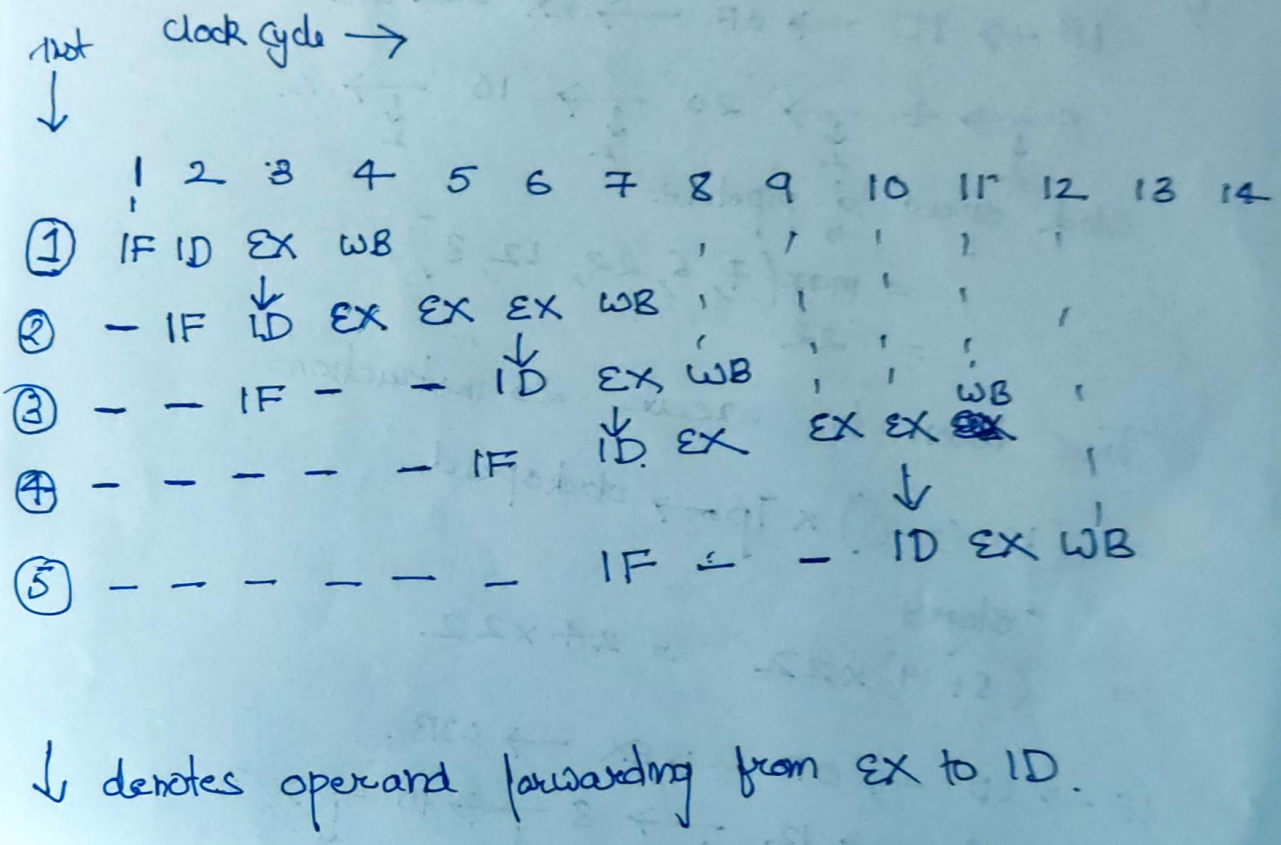
Time taken thus = (K + (n-1)) Tp.

= (6 + 19) × 14 = 25 × 14.

$$\text{speedup} = \frac{\text{Naive Implimen}}{\text{Efficent}} = \frac{24 \times 22}{25 \times 14} = \frac{528}{350}$$

$$= 1.508$$

Answer no 5.



↓ denotes operand forwarding from EX to ID.

Answer no 6 ①

int a[10] b[10], i;

↳ Each integer is 32 bits / 4 bytes
Hence consecutive block of int array is 4 bytes apart

L01: jeq r1, r2, end
→ if (r1 == r2) go to end.

L02: lw, r5, 0(r4)
→ r5 ← Memory [r4 + 0].
** r4 has base address of array b. Hence r5 has element of array b.

L03: shl r5, r5, U1 → r5 ← r5 << U1
✓ U1 = 3
** Multiplying array element by 8 hence left shift 3 (a[i] = b[i] x 8).

L04: sw, r5 0(r3) → Memory. [r3+0] ← r5
* Storing content of b to array a for specific index

L05: add r3, r3, U2

L06: add r4, r4, U3

→ r3 = r3 + U2
r4 = r4 + U3.

✓ U2 = U3 = 4

* Incrementing the address to point to next memory loc in a and b which is 4 bytes long.

14

L07: add r1, r1, 1 → r1 = r1 + 1

* Incrementing i index

L08: jmp U4 → go to U4

✓ U4 = L01

* After incrementing i index we have to check for loop cond^n which is L01