Academic Year: 2022-23 **Semester: IV**
Class/Branch: SE **Subject: MP**

Pentium Cache Organization and MESI protocol

→ Pentium processor has a separate code and data cache each of 8K bytes.

→ The cache line size is 32-bytes.

Note :- Blocks are called cache lines also and locations within blocks are called words also.

→ Since the Pentium processor has data bus of 8 bytes (64 bits), it requires a burst of 4 consecutive transfers to fill the cache line of 32 bytes.

→ In Pentium each cache is organized as 2-way set-associative.

→ The data cache can be configured as a write-through or a write back cache on a line-by line basis and it follows the MESI protocol.

→ The code cache does not require a write policy, as it is a read only cache.

→ Each cache has a dedicated translation look aside buffer (TLB) to translate linear addresses to physical addresses.

→ The cache can be enabled or disabled by sw or h/w

2-way associativity:

2-way associativity is a cache mapping technique.

What is cache mapping?

Cache mly lies blw the processor and the main mly (logically). Physically it might be an external cache or internal cache.

When processor wants any data, it first checks in cache. If it is available in the cache, it is a cache hit and if the data is not available it is a cache miss.

In case of cache miss, data is taken from main mly and copied in cache also.

We always want a high hit ratio. For which the principle of spatial locality is followed, which says if you are accessing a location, then you are more likely to access the locations around it.

In case of cache miss, it does not copy only that particular data, but it copies the entire block. So when a block is copied from main mly, where the block will get placed in cache mly - is what cache mapping is about.

Academic Year: 2022-23 **Semester: IV**
Class/Branch: SE **Subject: MP**

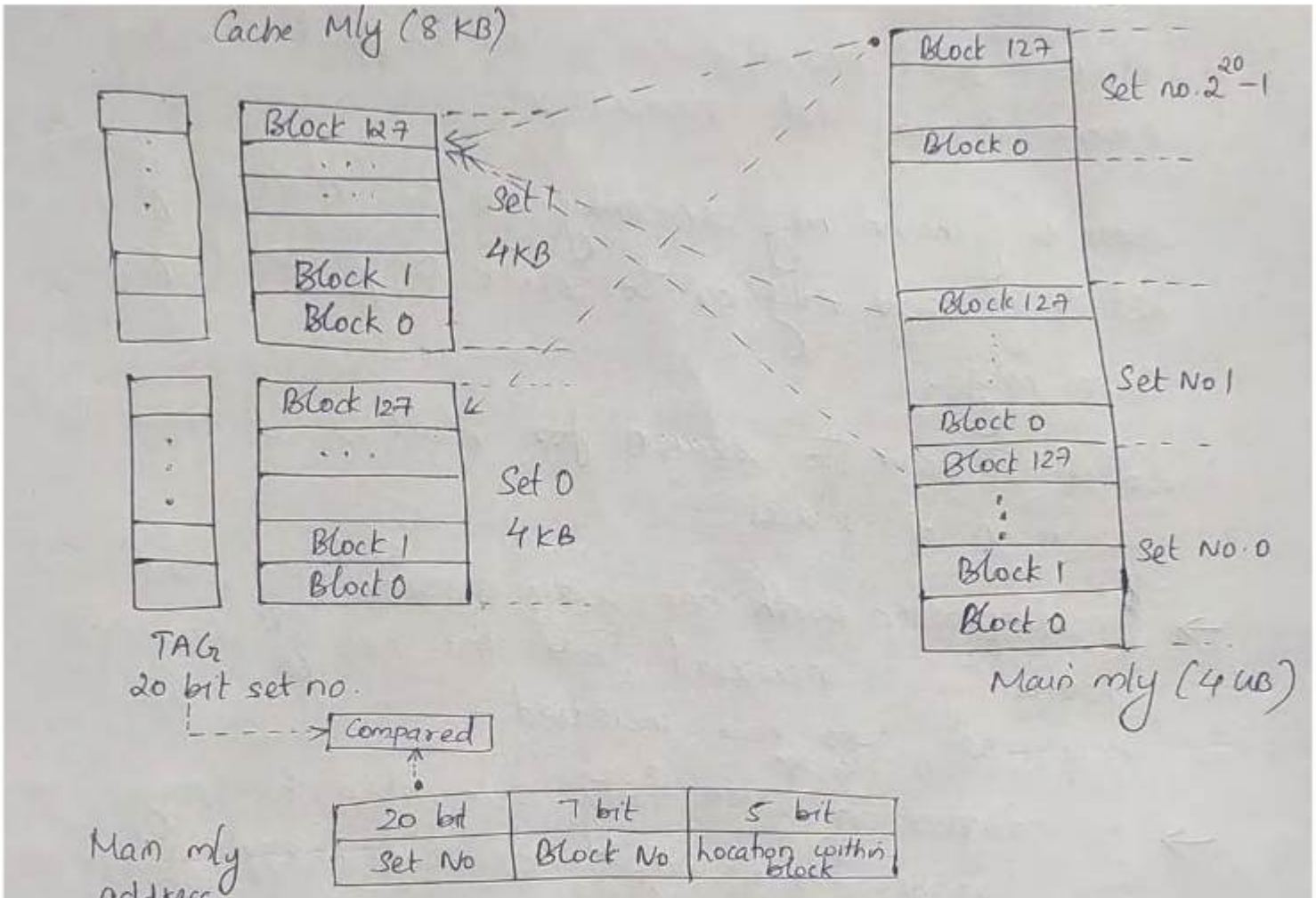


Figure: Code and Data Cache Organization.

With 2-way associative mapping, the cache is divided into 2 sets, each set having 128 blocks.

CM = 8 KB. So each set = 4 KB.

The main mly is divided into 2^{20} sets, each having 128 blocks and each block has 32 locations.

Here a block has 2 places to be mapped, so the name 2 way set associative.

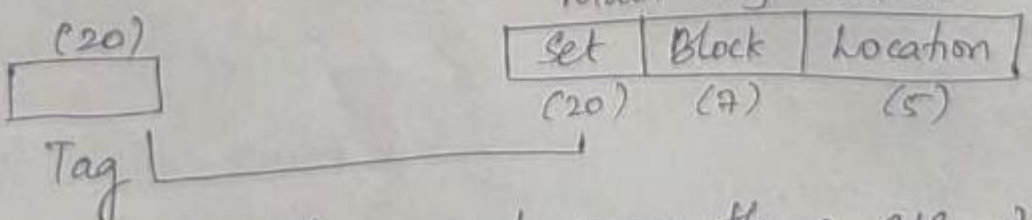
Block 0 of main mdy can only go to block 0 of set 0 of cache mdy or to block 0 of set 1 of cache mdy.

So if I have to search for block 0, I have to search in 2 places.

→ In comparison with one-way associativity, the flexibility is increased, but the no. of searches required also has increased.

→ In comparison with fully associative mapping (256 searches), 2-way associative mapping is still better.

When a block is copied from main mdy into cache mdy, the location and block no. will not change. Only the set no. changes.



Tag gives the set no and since there are 2^{20} sets tag will be of 20 bits.

tag (20 bits) in case of 2-way associative mapping is bigger than tag size in 1 way associative mapping (19 bits) but it is much smaller than the tag size in case of fully associative mapping (27 bits)

→ The hit ratio is much better than 1 way associative mapping. Why?

One way associative mapping has worst hit ratio.
For eg. consider a situation where an application needs only 2 blocks (Block 0 of set 1 and Block 0 of set 2).
With one way mapping, only one set can be mapped there in the cache only (with 1-way associative mapping, the cache only has only one set with 256 blocks). So when 2 blocks are required in alternate ways, every time there will be a miss.
This scenario does not arise with 2-way associative mapping as both the blocks can be mapped as there are 2 sets in cache only.

Write Policies / Cache consistency / Cache Coherency

Though cache mly has massive speed advantage, it comes with an inherent drawback. Due to cache, there are 2 copies of the same data; one in cache mly and other in main mly.

As long as both are same the cache is said to be consistent (coherent)

Inconsistency arises the moment a processor performs a write operation on the cache.

This means the same data in the main mly now has an old (stale / invalid) value

If another bus master (co-processor) accesses the same data from the main mly, it will get a stale value. Hence the cache becomes inconsistent.

This can be avoided by using good write policies such as:

- Write through
- Write back
- Snooping protocol
- MESI protocol.

Write through

- When a processor writes into Cache mly, it must also write into Main mly.
- This completely solves the problem of inconsistency but makes writes slow as writing into main mly is much slower as compared to reads. It is not all that bad because on an average majority operations are reads compared to writes.

Write Back (Delayed write / Posted write / Buffered write)

- Processor will only write into Cache mly. A Cache Controller (Eg. Intel 80386) keeps track of all blocks that have been modified in the Cache mly.
- When the processor is idle or at the time when this block has to be replaced, it will copy all modified information from cache mly to main mly.
- Thus makes processors writes very fast but keeps the main mly inconsistent for some time

Snooping

- Processor will write only into Cache mly. A Cache Controller keeps a track of all blocks that have been modified in the Cache Mly.

- Additionally the Cache Controller will 'snoop' activities of other bus masters.
- If it notices that another bus master is trying to access a location from main mly which has been modified in the Cache mly, the Cache Controller will stop activities of all bus masters, copy updated data from Cache mly to Main mly and allow the system to resume. This gives total consistency and also prevents unnecessary updates to main mly and hence gives the best performance.

Mesi Protocol

This is an advanced version of snooping for multiprocessor systems.

Note :- The Mesi protocol is only for the data cache and the SI protocol is used for code cache.

Each line in the data cache can be in any one of the 4 MESI states (Modified - Enclusive - Shared - Invalid) as indicated by the 2 bits stored along with the tag address.

Academic Year: 2022-23 **Semester: IV**
Class/Branch: SE **Subject: MP**

How does MESI work?

Consider 2 processors (A and B) each having its own local cache Mly and both sharing the Main memory. If the cache block is empty, its status is "invalid". Once a block gets occupied in only one cache mly, its status becomes "exclusive".

If the other processor also reads the same block into its own cache then both blocks have the status "shared". Such blocks are closely monitored by the cache controller.

If processor A writes into such a block, its status becomes "modified" and the corresponding block's status in processor B's cache becomes 'invalid'. If processor B wants this data it will have to access the Main mly

Now by the principle of snooping, the cache controller will update the modified data from the main mly and only then allow the other processor to access the data. Hence both processors will get the same consistent value of data.

How does MESI work?

Consider 2 processors (A and B) each having its own local cache M/y and both sharing the Main memory. If the cache block is empty, its status is "invalid". Once a block gets occupied in only one cache m/y, its status becomes "exclusive".

If the other processor also reads the same block into its own cache then both blocks have the status "shared". Such blocks are closely monitored by the cache controller.

If processor A writes into such a block, its status becomes "modified" and the corresponding blocks status in processor B's cache becomes "invalid". If processor B wants this data it will have to access the Main m/y.

Now by the principle of snooping, the cache controller will update the modified data from the main m/y and only then allow the other processor to access the data. Hence both processors will get the same consistent value of data.

Modified

- Indicates that this line in cache has been updated or modified due to a write hit in the cache.

In this case, when the cache subsystem snoops the s/m bus and finds a snoop hit, it writes the modified line back to mly (update the mly)

Exclusive

- It is the intermediate state b/w Shared and Modified.

Shared

- It indicates that this line may be resent in several caches and an exact duplicate of the information exists in each source (caches and main mly)

Invalid

- It is the initial state after reset and indicates that the line is not present in the cache.

Code Cache Directory Entry

Parity	Tag	S/I
(1 bit)	(20 bit)	(1 bit)

Data Cache Directory Entry

Parity	Tag	MESI
(1 bit)	(20 bit)	(2 bits)