Cache coherence

Cache does improve the performance of a system. It does, however, create a problem known as the cache coherence problem.

In a single CPU system, two copies of same data, one in cache memory and another in main memory may become different (inconsistent).

The contents of cache and main memory can be altered by more than one device e.g CPU can write to cache and I/O modules or DMA (Direct memory access) can directly write to main memory. This can result in inconsistencies in the values of cache and main memory.

Techniques for cache coherence in single CPU system

(1) Write Through: Write the data in cache as well as main memory. This will always keep the two copies of data (one in cache and other in main memory) consistent. The disadvantage of this technique is that a bottleneck is created due to large number of access of main memory.

② Write block: In this method updates are made only in the cache, setting a bit called update bit. Only those block whose update bit is set is replaced in the main memory. But here all the accesses to main memory whether from the CPU or input/output modules need to be from the cache resulting in complex circuitry.

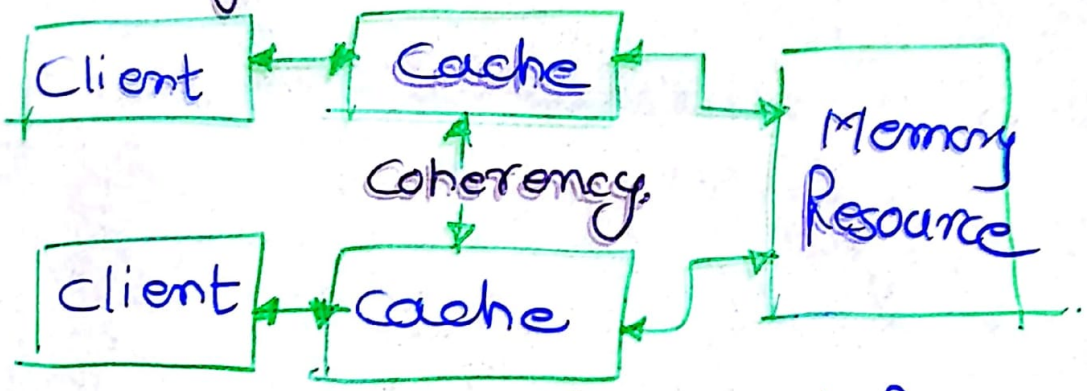
③ Instruction Cache: An Instruction cache is one, which is employed for accessing only the instructions and nothing else. Since the instructions do not change there will not be any inconsistency.

Cache Coherence in multiprocessor System

In a multiprocessor system, it is a general practice to have a cache associated with each processor. Multiple copies of same data can exist in different caches simultaneously, and if processors are allowed to freely update their own copies an inconsistency may arise. In general three sources of problems have been identified. ** (12A page)

① Sharing of ~~ve~~ writable data
② Process migration
③ I/O activity.

** (page 2)
When clients in the system maintains caches of a common memory resource, problems may arise with inconsistent data This is particularly true in a multiprocessing system. as shown.



Maltiple caches of shared Resource.

Top client has a copy of a memory block from a previous read and the bottom client changes that memory block without any notification of the change. Cache coherence is intended to manage such conflicts and maintain consistency between cache and memory.

* page 2
In case of power failure, the data in the cache memory is lost, so there is no way to tell which locations of the main memory contain old data. Therefore the main memory as well as cache must be considered volatile and provision must be made to save the data in cache. →

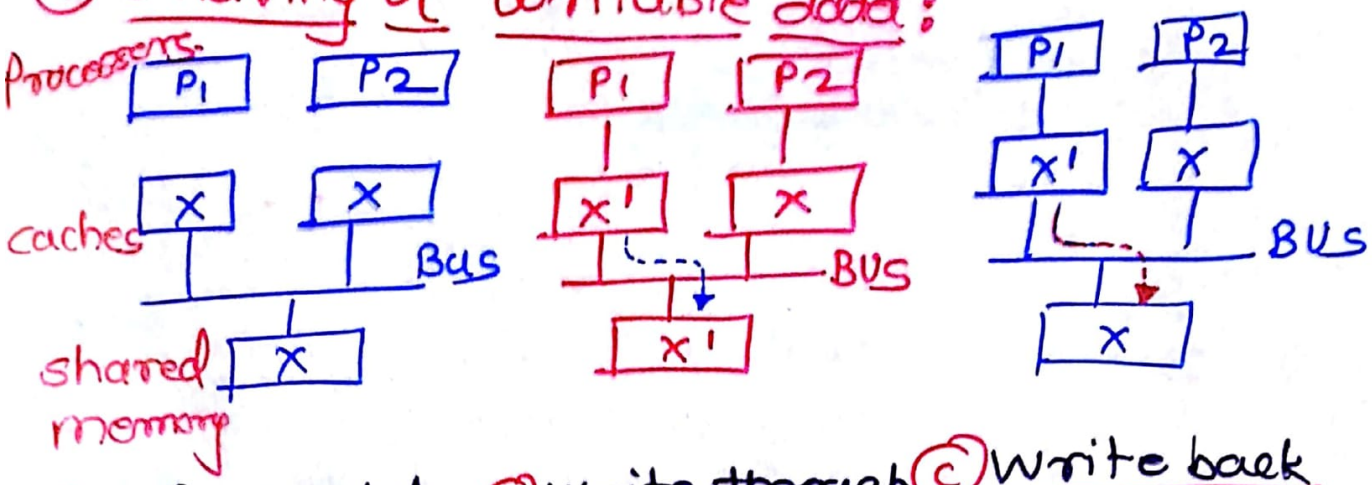
→ It is necessary that all the altered blocks must be written to the main memory before another device can access these blocks in main memory.

*** from page 1 write through

In Write through system main memory always contains a valid data and thus any block in the cache can be overwritten immediately without data loss.

- Simple approach
- This approach requires time to write data in main memory with increase in bus traffic.
- This in effect reduces the system performance.

(1) Sharing of writable data:



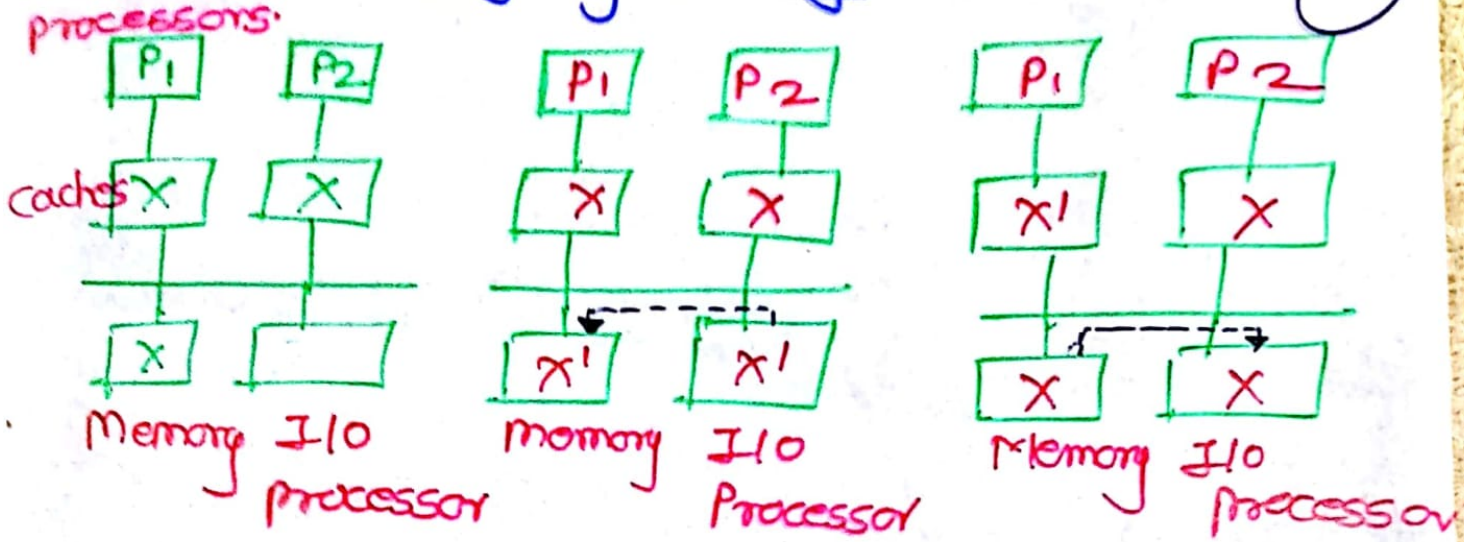
(a) Before update (b) Write through (c) Write back or Write block

Consider a multiprocessor with two processors P_1 and P_2. Each processor has a private cache and each one is sharing the main memory. Let X be a shared data element which has been referenced by both processors. Before update three copies of X are consistent. (as shown in a)

If processor P1 writes new data X' into the cache, the same copy will be written immediately into shared memory using a write through policy. In this case inconsistency occurs between two copies (X' and X) in two caches as shown (in b)

On the other hand, inconsistency may also occur when a write-block policy is used as shown. (in c)

I/O activity bypassing cache:



(i) Before (ii) Write through (iii) Write block

I/O operation bypassing the cache.

Inconsistency problems may occur during I/O operations that bypass the caches

- When the I/O processor load a new data x' into the main memory, by passing the write through cache. (middle (ii)) and inconsistency occurs between cache 1 and shared memory.
- When outputting a data directly from the shared memory (bypassing the cache 1), the write-block caches also creates inconsistency.

Cache coherence strategies (A)

These strategies are divided into

1. Software solutions.
2. Hardware solutions.
   - (i) Snoopy Protocols
     - (a) Write-invalidate protocols (MESI)
     - (b) Write update protocols.

Software solutions: the problem of cache coherence is managed entirely by compiler and operating system.

compiler mark the data which are likely to be changed during execution subsequently operating system prevents such noncacheable data from being cached. The simplest approach is to prevent any shared data variable being cached. Shared data structure may be exclusively used during some periods and may be effectively read-only during other periods.

It is the only during periods when at least one process may update the variable and at least one other process may access the variable then cache coherence is not the issue.

(2) Hardware Solutions

Hardware-based solutions are generally referred to as cache coherence protocols. These protocols dynamically handle inconsistency conditions during runtime. This leads to more effective utilization of cache and hence improved performance.

(i) Snoopy protocols:

When a number of processors are connected to a common bus and each processor has its own local cache then cache consistency can be maintained using:

(a) write invalidate policy
(b) write update policy

In Write-invalidate, whenever changes are made in the local cache of a processor, all remote copies (copies with other processors are invalidated (a processor does not use the invalidated data) Thus only one copy will remain after Write-invalidate. Valid

The Write-update will broadcast (send to all processors) the new data blocks to all caches containing a copy of the block. Thus all copies of data will become identical after updation in local cache.

Snoopy protocols are based on bus watching mechanism

- When using write-invalidate protocol, if a processor modifies a value in its local cache, all other copies of data are invalidated via bus.
- Invalidated blocks are called dirty blocks and they should not be used.
- The write-update protocol requires that updated block be broadcast to all caches copies via the bus.

The MESI Protocol

Write-invalidate approach is the most widely used in multiprocessor systems. Write invalidate protocol is also known as MESI protocol.

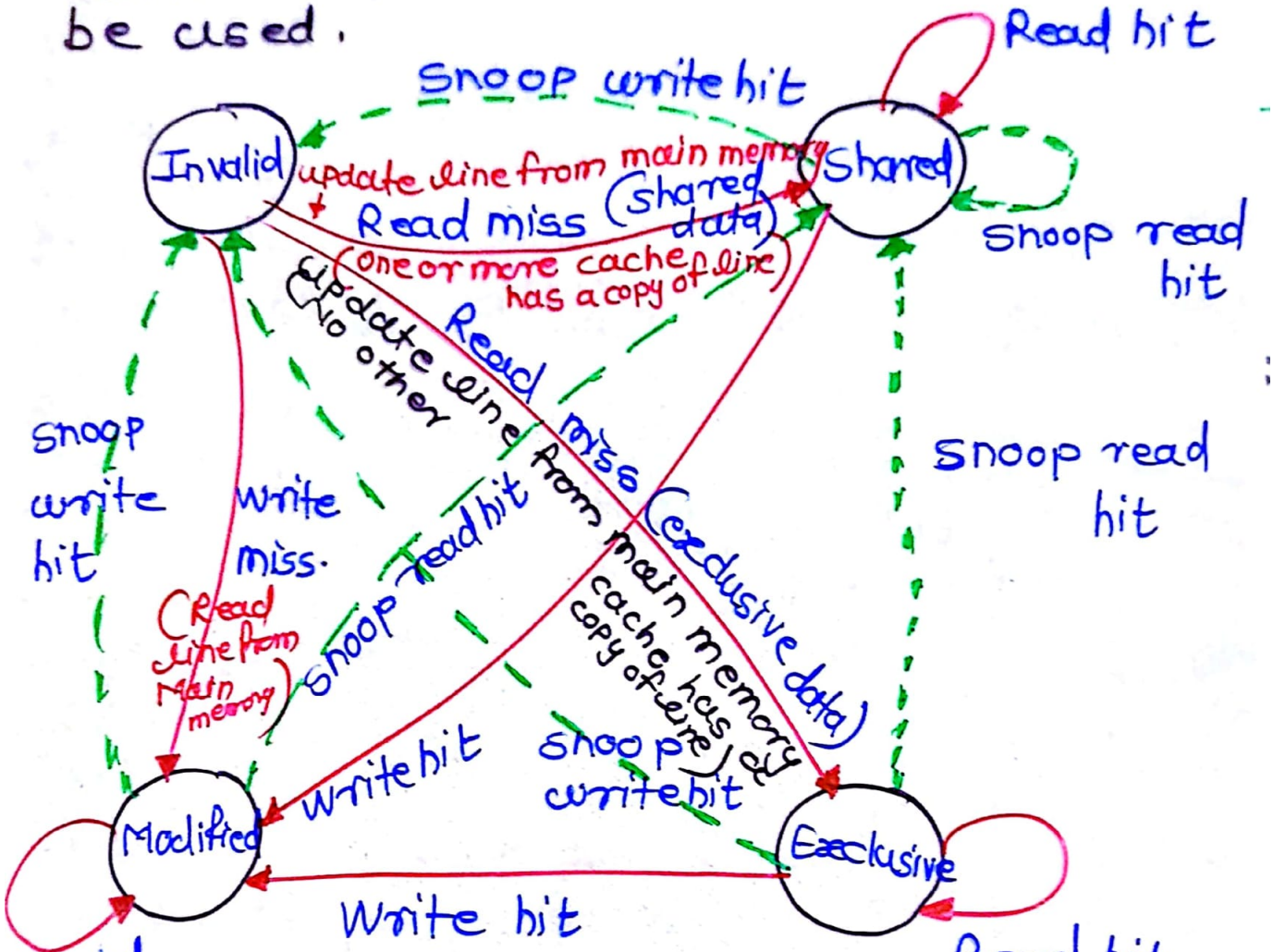
In MESI protocol, data cache has 2 state bits per tag with two bits (00, 01, 10, 11), each cache block will be in one of four states.

- **Modified:** The block in the cache has been modified (different from main memory) and is available in this cache. This is the only valid copy data.

Exclusive: The block in cache is same as that in main memory and is not present in any other cache.

Shared: The block in cache is the same as that in main memory and may be present in another cache.

Invalid: The block in cache does not contain valid data and it should not be used.



Solid lines: Command issued by local processor.

Dotted lines: Action taken by a remote process in response to command issued by a local processor.

State-transition graph (simplified) for a cache block using MESI coherence protocol.

E

The cache coherence mechanism receives requests from both the processor and the bus and responds to these based on the type of request such as read/write and hit/miss in the cache, and the state of the chache block specified in the request

(Snoop: Investigate or look around)

Read Miss: When a read miss occurs in the local cache, the processor reads the line of main memory which contains missing address. The snoopy cache controller sends read miss command on the bus. This alerts all other processors to snoop the transaction.

Read Hit: When a read hit occurs on a block which is currently in local cache, the processor reads the block in its cache. There is no change in its state. The state remains modified, shared or exclusive

Write Hit: When a write hit occurs on a block which is currently in the local cache, the action which is to be performed depends on the current state of that line in the local cache.

Write Miss: When a write miss occurs in the local cache, the processor gives a command on the bus to read the block of main memory containing missing

address. When the block is loaded, its status is marked as modified.

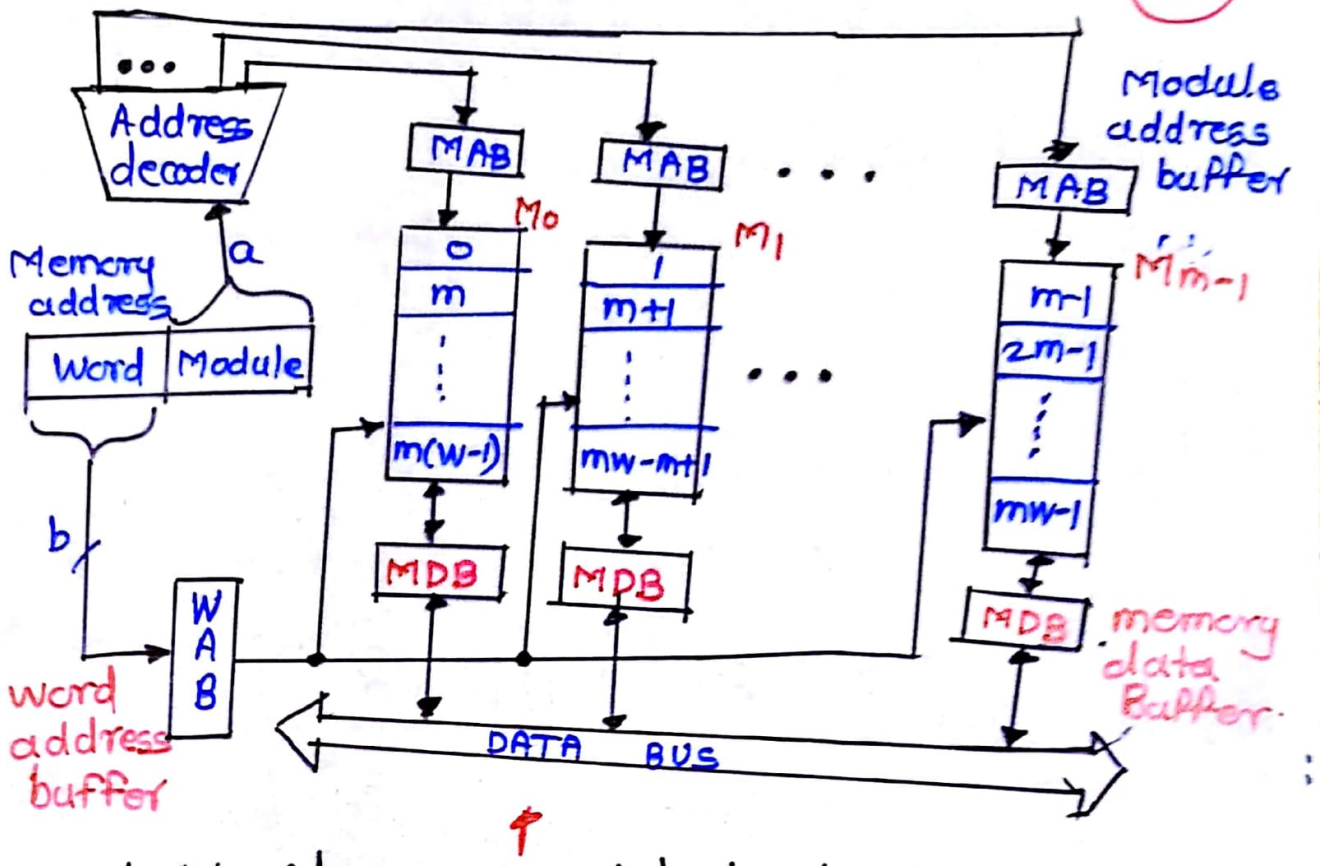
Interleaved Memory

- Interleaved memory implements the concept of accessing more words in single memory access cycle.
- Memory can be partitioned into N seperate memory modules. Thus N accesses can be carried out to the memory simultaneously.
- Once presented with a memory address, each memory module returns one word per cycle. It is possible to present different different addresses to different memory modules so that parallel access to multiple words can be done simultaneously or in a pipelined fashion.
- The maximum processor bandwidth in interleaved memory can be equal to the number of modules ie N words per cycle.

* To achieve the address interleaving consecutive addresses are distributed among N interleaved modules. For eg if we have consecutive addresses and 4 interleaved memory modules then $0^{th}, 4^{th}, 8^{th}$ addresses will be assigned to the first memory module and so on.

0, 4, 8, 12, 16 ---- Addresses to memory module 0
1, 5, 9, 13, 17 --- Addresses to memory module 1
2, 6, 10, 14, 18 --- —11— —11— 2
3, 7, 11, 15, 19 --- —11— —11— 3

* Consider a main memory formed with $m = 2^a$ memory modules, each containing $w = 2^b$ words of memory cells. The total memory capacity is $m \cdot w = 2^{a+b}$ words.



Low-order m-way interleaving (the c-access memory scheme)

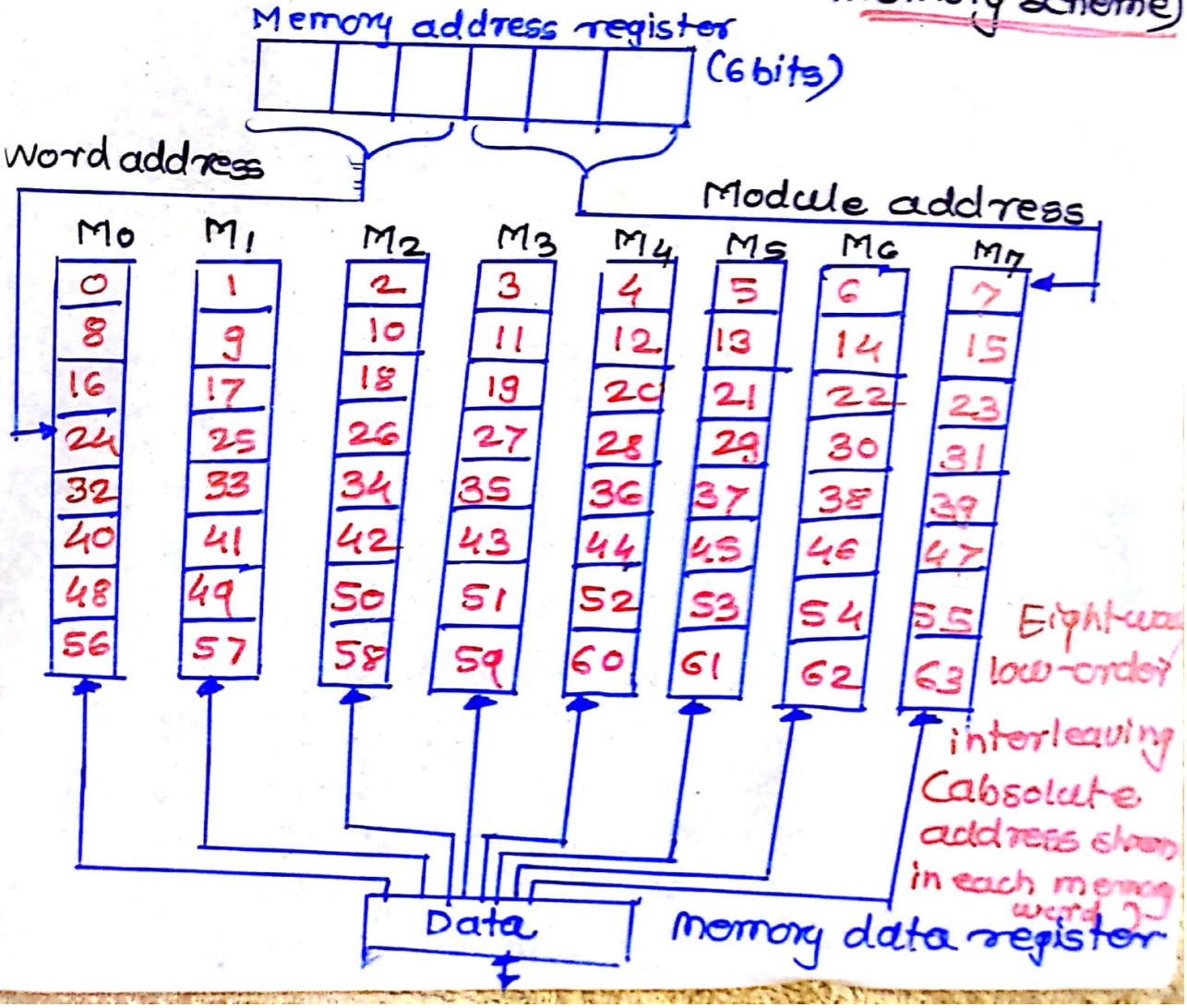


Fig shows memory format for memory interleaving. Interleaving spreads contiguous memory locations across m modules horizontally. This implies that the low-order a bits of the memory address are used to identify the memory module.

The high-order b bits are used to address a word inside a module, Same word address is applied to all memory modules simultaneously. A module address decoder is used to distribute module addreases.

Associative Memory

In associative memory any stored item can be accessed by using the contents of the item. The subfield chosen to address the memory is called the key. Items stored in an associative memory can be viewed as having the two field format:

KEY, DATA

where KEY is the stored address and DATA is the information to be accessed.

A key to be searched is matched simultaneously

Associative searching is based on simultaneous matching of the key to be searched with the stored key associated with each line of data. A word is retrieved based on a portion of its contents rather than its address.

Virtual memories

In most computer systems the physical main memory is not as large as the address space of processor. For example if the processor uses 32 bit address then its address space will be

$$2^{32} = 2^{30} \times 2^2 = 4\,GB$$

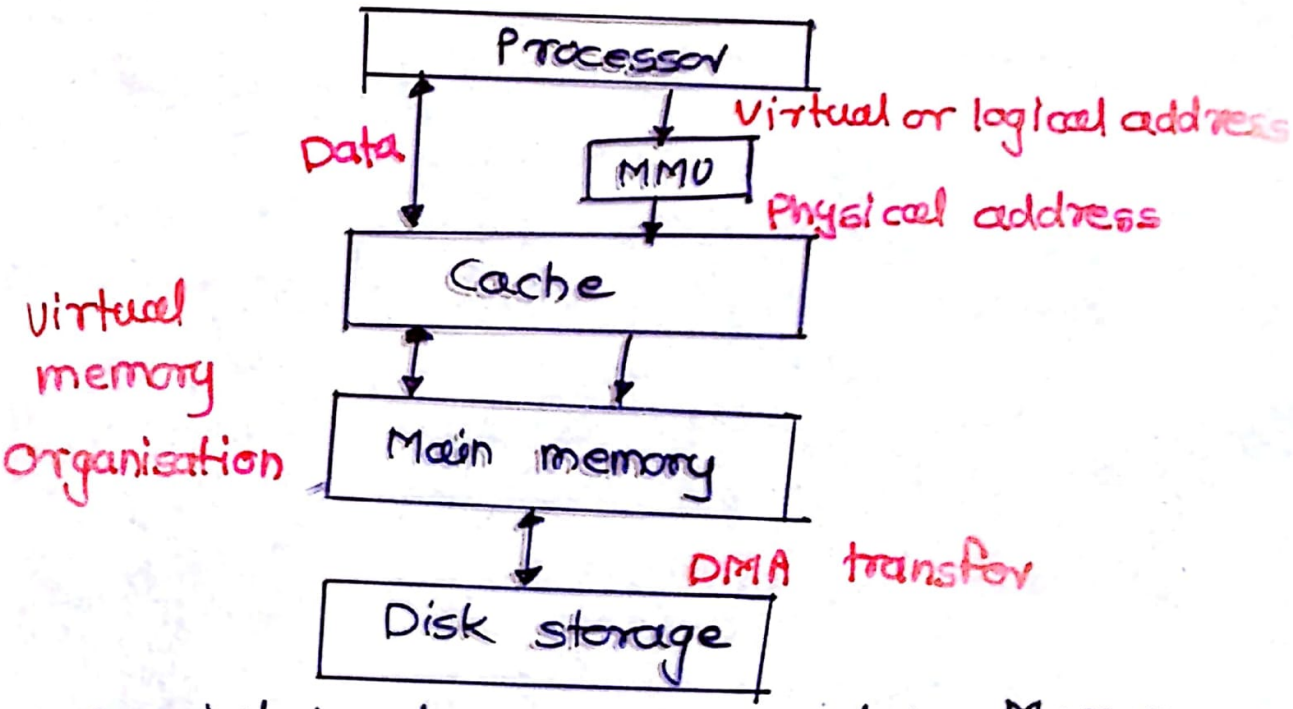
In reality the physical memory that we use in a computer system is in megabytes and not in gigabytes.

When we try to run a program that will not completely fit into the main memory the parts of it currently being executed are stored in main memory and the remaining portion is stored in secondary storage device such as hard disk.

Of course, all parts of a program which are needed for execution are first brought into the main memory. When a new segment of the program is to be brought and the memory is full, it must replace another segment already in the memory. Thus an application program can run without any limitation imposed by available main memory.

The techniques that automatically move program and data blocks into the physical main memory when they are required for execution are called virtual memory techniques.

- Program or processor references an instruction and data space that is independent of available physical memory space.
- The address issued by the processor (either instruction or data) is called virtual or logical address
- The virtual address is translated into physical address by a combination of hardware and software components.
- If a virtual address refers to a part of the program or data space that is currently in the physical memory, then it is accessed immediately
- If the referenced address is not in the main memory, its contents must be brought into main memory before it can be used



special hardware unit called the Memory Management Unit (MMU) translates virtual address into physical address.