Chapter 6 Memory System Design Principles & Modern Memory Technology PART 2

Contents of this chapter (Part 2)

- Cache Coherence in Multi-core/Multi-CPUs & Protocols to resolve

ANNEXURE – Self-Reading on basic memory operations

Cache Coherence Problem in Multi-core/CPU systems

The problem of data inconsistency between the MM and cache is referred to as the cache coherence problem

Three possible sources:

- Sharing of data that can be modified
- Process migration
- I/O activity

Refer to figures in the PDF

Two protocols are suggested and followed widely in practice

- *Snoopy Bus Protocols*
- *Directory Based Protocols*

Snoopy Bus protocol:

We have a bus watcher unit with each processor that observes the transactions on the bus.

In particular, the watcher monitors all the mem write operations.

Two different approaches are possible. These are

(a). Write-invalidate protocol
(b). Write-update protocol

Further, the basic protocol in achieving the consistency can be through:

(i). Write-through caches
(ii). Write-back caches

Write-through caches:

We will consider only the write-invalidate protocol.

For write-through caches, we have two possible states that a cache block can assume whenever a read, write, and a replacement operations are effected : valid and invalid states

In valid state - all processors can read safely
In invalid state - block is invalidated or being replaced

Whenever a processor writes into its local cache, all other cache copies become invalidated.

Write-back caches: Here,

valid state -> RW (read-write) state
+ RO (read only) state

invalid state - same as before

This scheme corresponds to an ownership protocol.

Ownership Protocol

- In this scheme, whenever a SM (shared memory) owns a block, the remote caches can only contain RO copies of the block.
- This means that multiple copies can exist, but all the processors are allowed to only read.
- The RW state corresponds to only one copy existing in the entire system owned by the local processor, say i. Then, read and write can be performed safely.

Modifying a block using Ownership Protocol

Suppose a block needs to be modified by a processor, then:

- *First it has to obtain a permission to do so. This is done by broadcasting a request to all caches and the SM.*
- *If a modified block exists in a remote cache, then the SM is first updated and all other caches will be invalidated.*
- *After updating, the ownership is transferred to the requesting cache.*

Remarks: Write invalidate vs Write Update

- A write invalidate protocol may lead to a heavy bus traffic caused by read-misses, resulting from the processor updating of a variable and other processors trying to read the same variable.

- Also, the write update protocol may update data items in remote caches which will never be used by other processors.

These problems pose additional limitations in using buses to build large multiprocessors.

Consequently, we have….

Directory based protocols :

Snoopy bus methods are not suitable when the number of processors is large, as broadcasting becomes more expensive.

This is an alternative protocol which recommends every node to maintain a directory of the blocks that are currently cached.

Two flavours :

Full Map directory method and Limited Directories

If interested, read pages 358 and 359 in your recommended text (before Full-Map directories)

Workings of Directory-based protocol

System: Shared memory; N Processors with each with its local cache; higher bus bandwidth (scalability w.r.t N is possible);

A directory is used to ensure cache coherence.

Main purpose of the directory residing in the main memory: To know which blocks are in caches and their status.

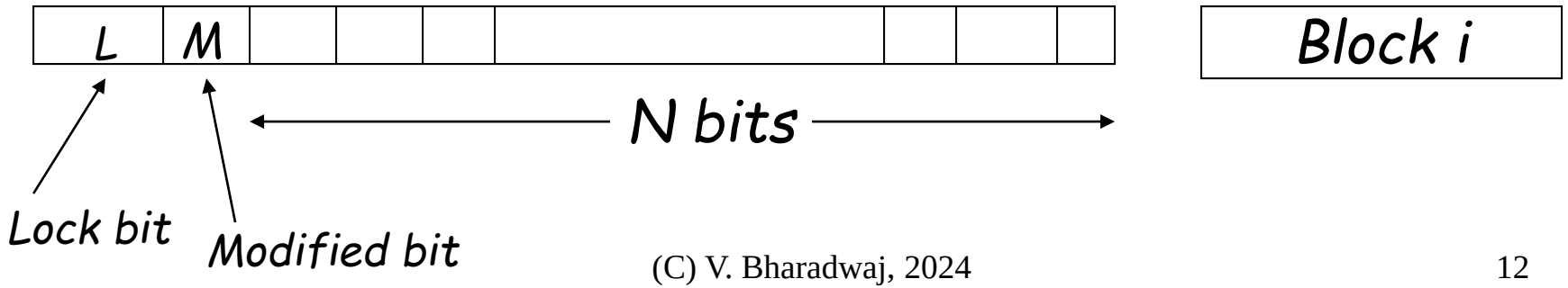
Suppose a Multi-Pro has M blocks in main memory and there are N processors and each processor has a cache.

Each memory block has an N bit directory.

Consider the following example which demonstrates the working of directory-based protocol.

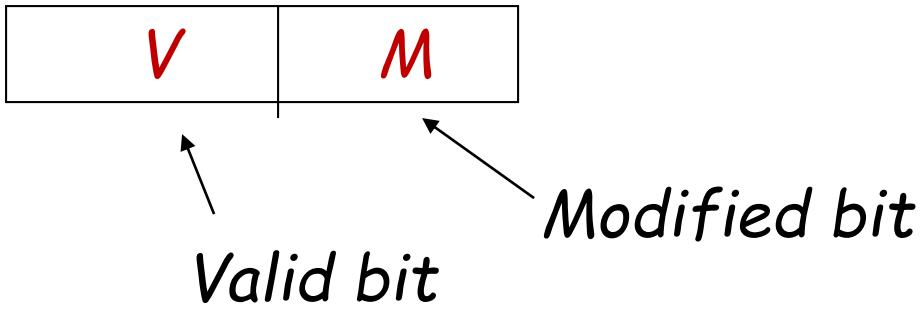
Number of Main Memory blocks: M
Number of Processors: N

Each MM block has N bit directory entry as follows.
Space complexity: O(MN) -> expensive!



If k^{th} processor's cache has this block i, then the k^{th} bit is set to 1

Similarly for each block in cache memory we maintain a two-bit information as follows.



Following table identifies different states that can exist in the system for a block

For a MM block

State	L	M	Remarks
Absent (A)	0	0	No cache holds a copy of this block
Present (P)	1	0	One or more caches has this block
Present Ex (PM)	0	1	Exactly one cache has this copy
Locked (L)	1	1	Operation on this block is going on and it is locked

For a Cache Memory block

State	V	M	Remarks
Invalid (I)	0	X	Contents of this block on cache are invalid
Valid (V)	1	0	Valid and unmodified
Valid M (VM)	1	1	Valid, modified, and it is the only copy.

Note: Cache consistency is maintained by actions taken by mem controllers. The actions depend on read/write requests, state of the block in cache and main memory.

Now consider how we use these tables while performing the following operation.

Scenario: "Read from Memory to Processor Register"

Assumptions:

- Write-invalidate protocol
- Write-through cache

Description: k^{th} processor executes "load" instruction

Case I: Read hit

This means that a block containing a word is in the k^{th} processor's cache

a) if status of the block in cache is V or VM, then read the word; no other action.

b) Suppose if the status is I, then read the valid copy from the cache of a processor holding a valid copy

Case II: Read Miss

This means that the block is not in the local cache. We have to check the status of this block in the main memory

a) If the status of the block is A, then we do the following.

(i) Replacement algorithm to be invoked and bring the block from MM to the k-th processor's cache

(ii) Full-fill the read request from cache (write-through cache)

(iii) Enter a '1' in the k-th bit position of the directory entry of this block

*(iv) Change the status bits of the directory entry of this block to **PM***

*(v) Change the status bits of the k-th processor cache block to **V***

(b) If the status of the block in MM is P, then carry out the actions mentioned above - (i), (ii), (iii), and (v)

(c) If the status of the block in MM is PM, then carry out the actions - (i), (ii), and (iii) and the following.

*(vi) Change the status bits of the directory entry of this block to **P***

*(v) Change the status bits of k^{th} processor's cache block to **V***

Virtual Memory Technology

Though the modern day computers have enough MM, still the amount of applications that run demand a large working space.

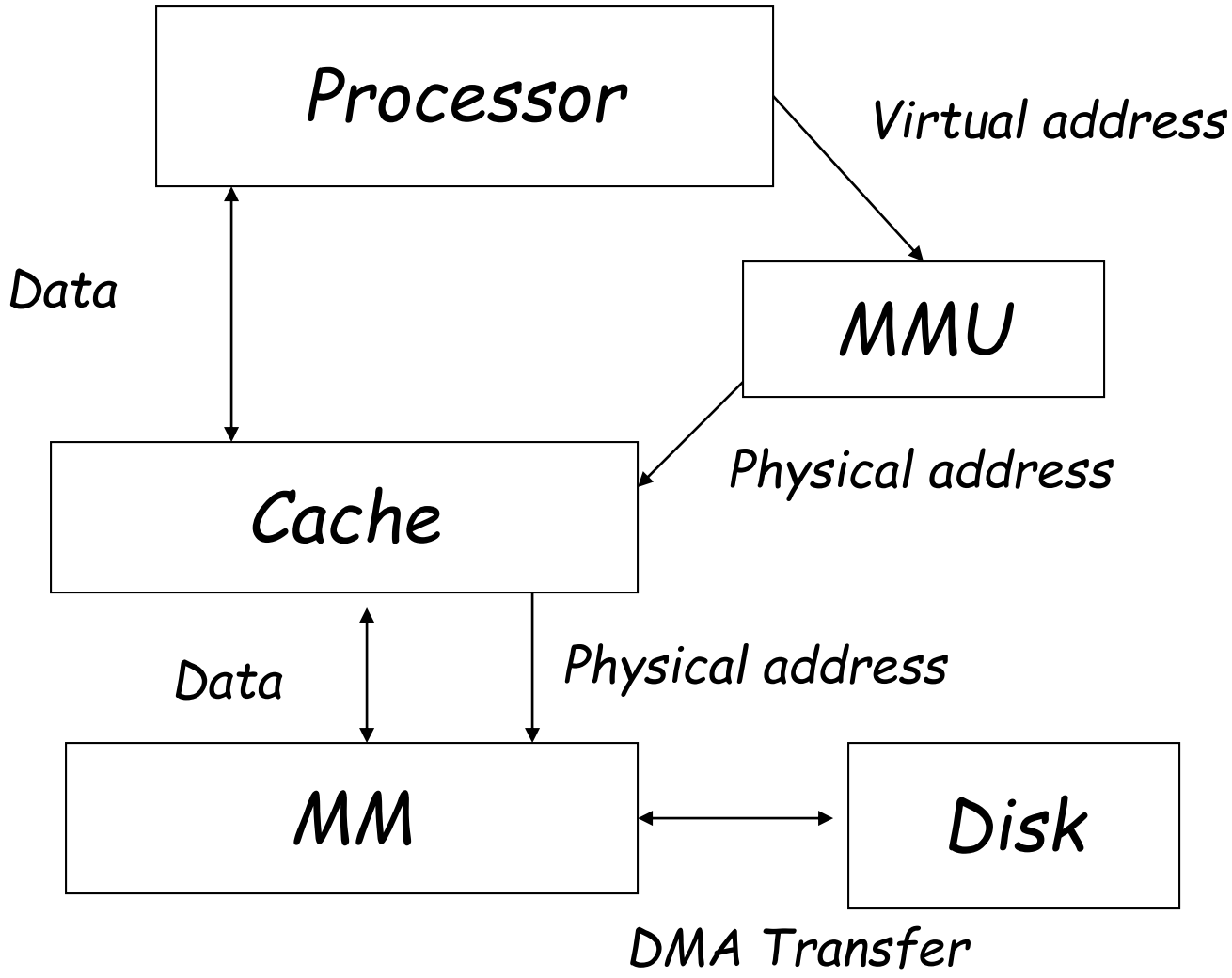
Also, usually, the physical MM is not as large as the address space spanned by an address issued by the processor.

When a program is to be executed, it has to be brought into the MM . The OS moves the data/pgm between the MM and the secondary storage devices

Techniques that move the pgm or the data automatically to the MM when they are required by the CPU are called virtual memory techniques.

The binary addresses that the processor issues for either instructions or data are called virtual or logical addresses.

These virtual addresses are translated into physical addresses by a combination of h/w and s/w components.


Processor
Virtual address
Data
MMU
Physical address
Cache
Data
Physical address
MM
Disk
DMA Transfer

Note:

- *A cache bridges the speed gap between CPU and MM and is implemented in h/w.*

- *The VM mechanism bridges the speed and size gap between MM and secondary storage and is implemented using s/w.*

Effect of page size

Page size is often a parameter that can be chosen by the OS designers.

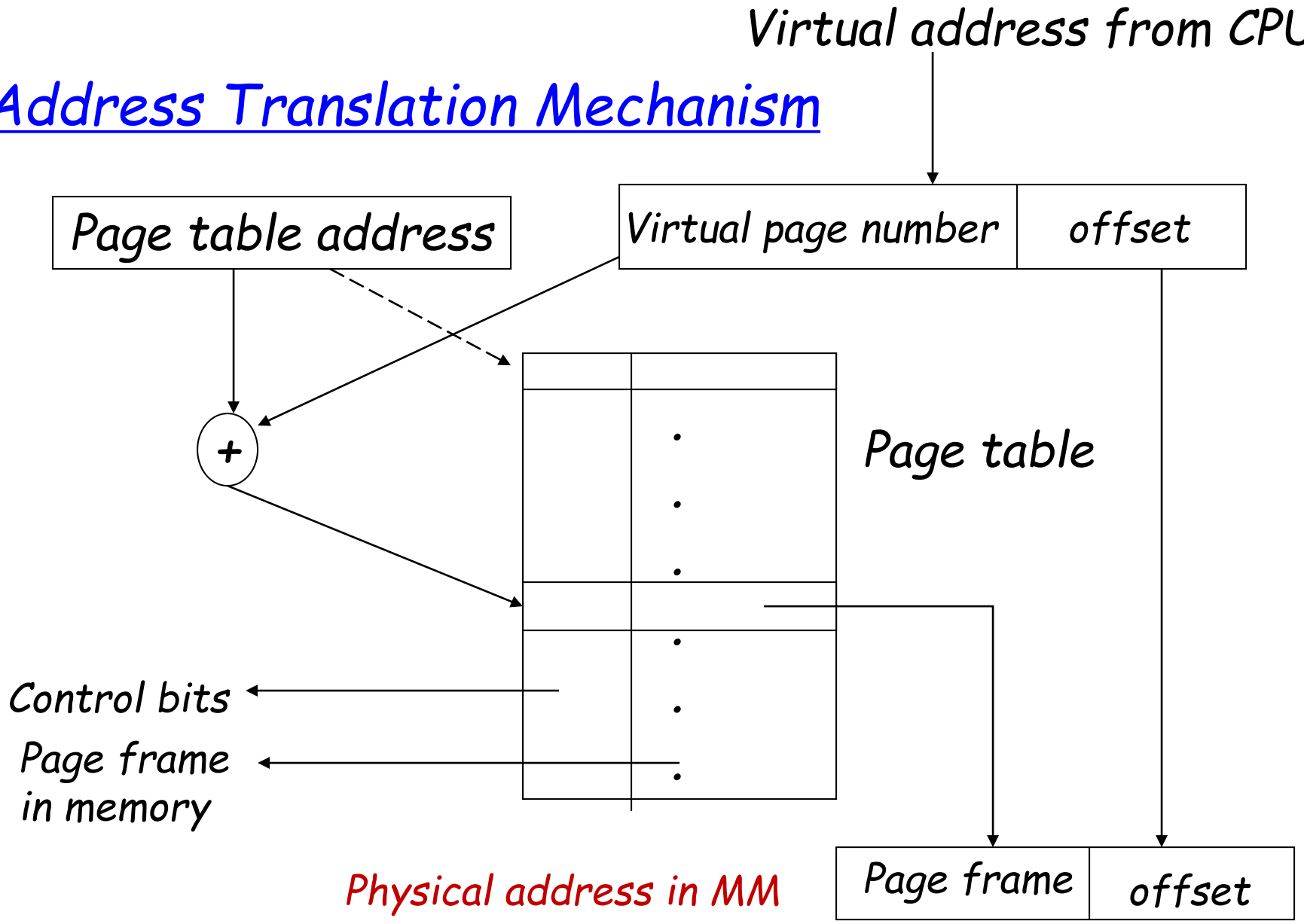
Even if the hardware has been designed with, for example, 512-byte pages, the OS can easily consider pages 0 and 1, 2 and 3, 4 and 5, and so on, as 1K pages by always allocating two consecutive 512-byte page frames for them.

Determining the optimum page size requires a careful balance between several competing factors.

For instance, a randomly chosen text, data, will not fill an integral number of pages. On an average, half of the final page will be empty. The extra space in that page is usually wasted (internal fragmentation).

Thus, with n segments in memory and a page size of p bytes, np/2 bytes will be wasted on internal fragmentation.

Address Translation Mechanism



Estimating the size of the Page Table

Each process uses different number of pages.

This means, memory space demands of each process is different.

Question: For every page that belongs to a process we need to have an entry in the PT. When each process uses different number of pages, how do I estimate the size of a page table?

P: Size of a page in bytes;
S: size of a process in bytes;
e: #of information bits needed per page in the PT (in bytes);

S/P - # of pages used by the process;
Se/P: # of bits;
P/2: Overhead due to internal fragmentation

Note: On an average P/2 amount of memory is wasted in the last page in a process and this is referred to as an internal fragmentation;

Estimating the size of the Page Table

Thus, the total overhead is: $Se/P + P/2$

1st term has an inverse relationship in P;
2nd term is directly proportional to P;

This prompts us that an optimal value must be somewhere in between!

Determine an optimal P*:

This yields:

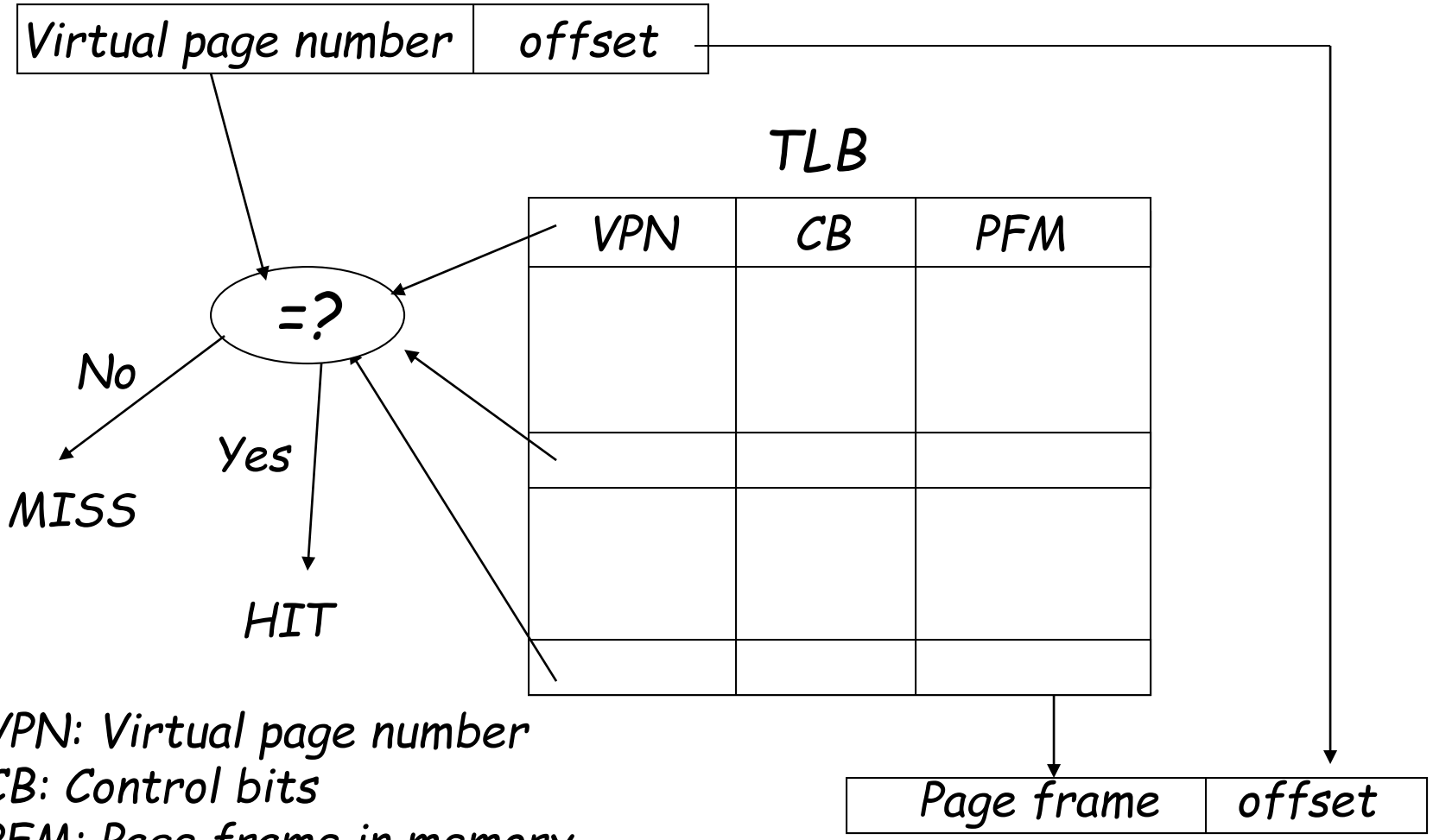
$$-Se/P^2 + \frac{1}{2} = 0;$$

implies

$$P^* = (2.S.e)^{1/2}$$

As an example, S = 128K; e = 8 bits, implies: $P = (2.S.e)^{1/2}$ = 1448 bytes! As a thumb rule, in practice, a 1K or 2K size will be used depending on other factors such as disk speeds & other overheads;

Use of Translation Look-aside Buffer (TLB)



VPN: Virtual page number
CB: Control bits
PFM: Page frame in memory

ANNEXURE

- *Basic Concepts in Memory System*

Memory Systems Design Principles

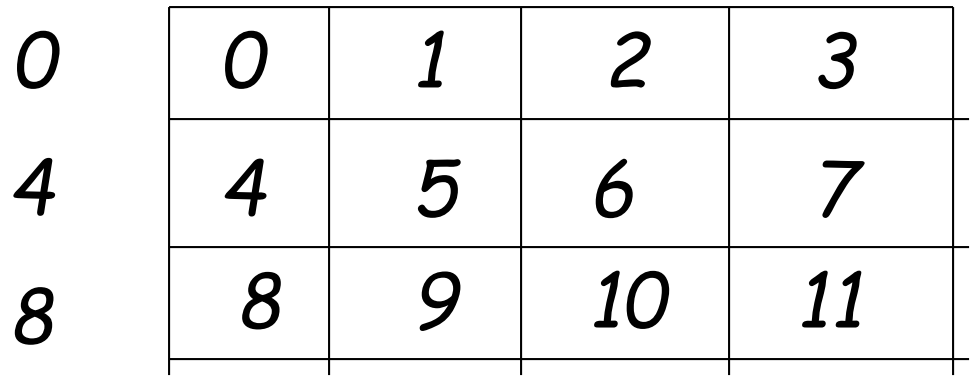
Note: Slides 90 to 101 – For you to read on some very basic stuff…

Some basic concepts and facts !

- *Max size of the memory that can be used in any computer is determined by the addressing scheme.*

  A 16 bit computer generates 16-bit addresses and is capable of spanning up to 2^{16} = 64K memory locations.

- *Modern computers are byte addressable.*



Organization of the main memory in a 32-bit byte-addressable computer

Big-endian arrangement

- *The main memory (MM) is usually designed to store and retrieve data in word-lengths*

Note: *The number of bits actually stored and retrieved in one MM access is the most common definition of the word length of a computer.*

Example

Consider a byte-addressable computer with the structure shown in the above figure. When a 32-bit address is generated from the CPU and sent to MMU, the higher order 30 bits determine which word to access. If a byte quantity is specified, the low order 2 bits of the address specify which byte location to access.

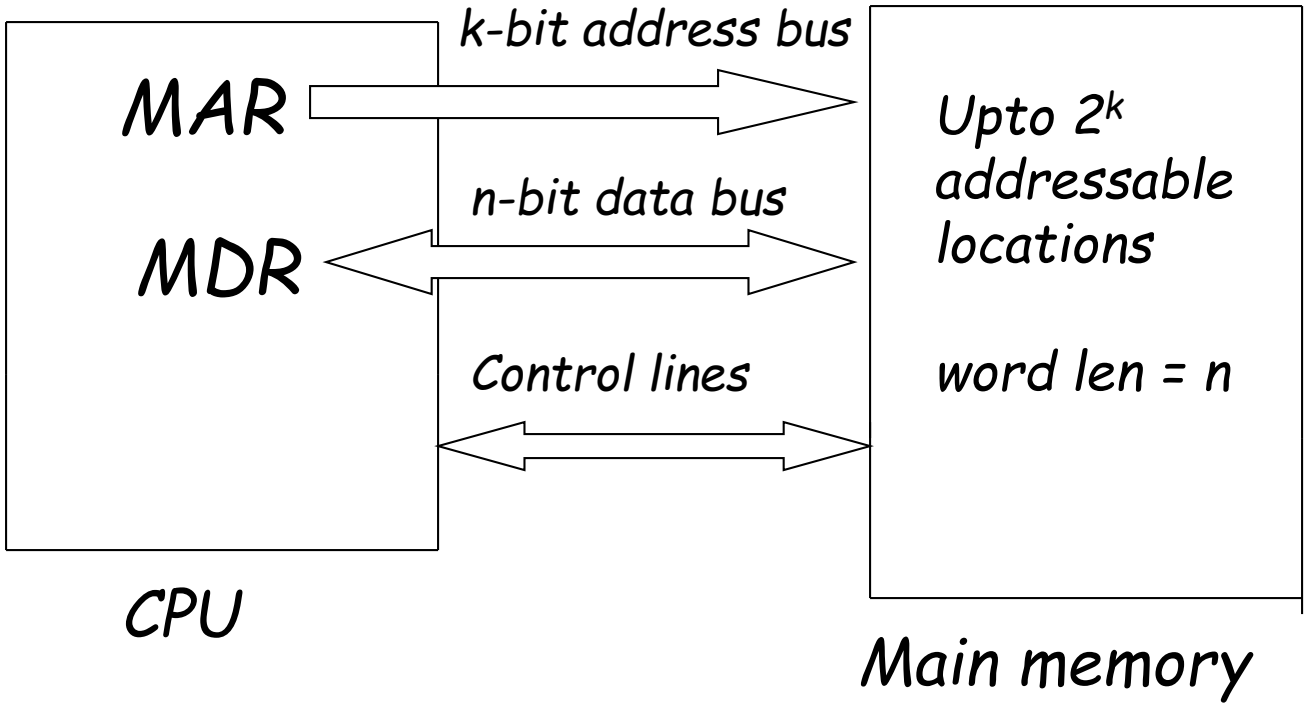
- *Data transfer between CPU and MM takes place through the use of two CPU registers, usually called MAR (memory address register) and MDR (memory data register).*

- *If MAR is k bits long and MDR is n bits long, then the memory unit may contain up to 2^k addressable locations.*

- *During a memory cycle, n bits are transferred between memory and CPU. This transfer takes place over processor bus which has k address lines and n data lines.*

- *The processor bus also includes control lines Read, Write, and Memory function complete(MFC) for co-ordinating data transfers.*

- *In a byte addressable computers, another line may be present to indicate that the transferred word is just a byte rather than a full word of n bits.*



CPU-Memory interaction:

1. CPU initiates a mem operation by loading the appropriate data into registers MDR and MAR.

2. It then sets either read/write memory control line to 1.

3. When the required mem operation is completed the mem control circuitry indicates this to the CPU by setting MFC to 1.

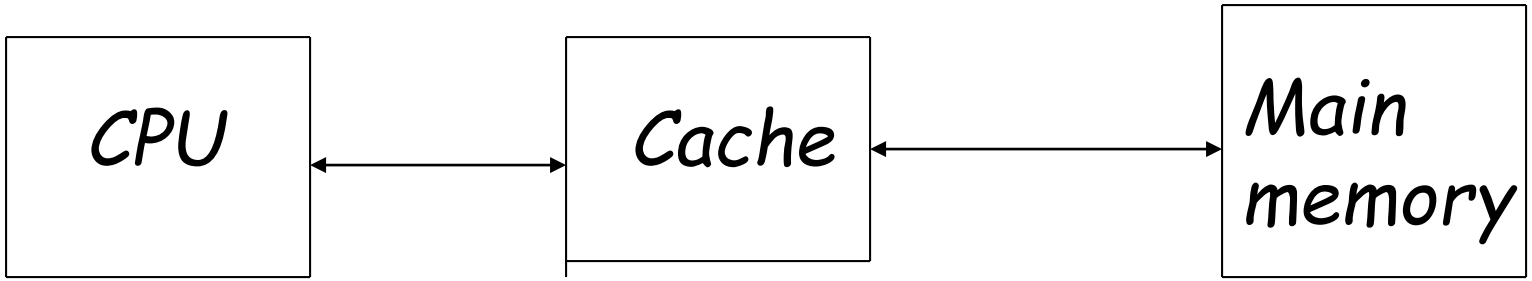
Some common terminology

A useful measure of the speed of the memory units is the time that elapses between the initiation of an operation and the completion of that operation, for example, time between the Read and MFC signals. This is referred to as memory access time.

Another useful measure is the memory cycle time, which is the minimum delay required between the initiation of two successive memory operations, for example, between two successive Read operations.

Cycle time is usually > access time, depending on the implementation details of the memory units.

Cache memories



Why cache?

- *CPU processes the instr. and data at a much faster rate than they can be fetched from mem unit.*

- *Then, memory cycle time is the bottleneck of the system.*

Note that even if we prefetch and keep some of the instructions and data into a location, unless the memory cycle time is reduced, the problem cannot be solved.

Cache memory helps to solve this problem. This is a small piece(!!) of memory placed between CPU and the MM unit so that CPU always checks for the required data/instr. in cache first and then in the MM. This is a very fast memory unit,

specially designed to match the speed of the CPU.

- *Effectiveness of the cache mechanism is based on a property of computer programs called the locality of reference.*

Typical analysis of programs revealed a nice property that maximum time is spent on instructions that are repeated frequently .

This property manifests in two ways - spatial and temporal.

The spatial property means that a recently executed instruction is likely to be executed again very soon, and instructions in "close" proximity are also likely to be executed soon.

The temporal property suggests that whenever an information item is first needed, this item should be brought into the cache where it will hopefully remain until it is needed again.

Spatial aspect suggests that instead of bringing one instruction, bring a set of instructions "close" to the currently needed one.

We will use the term block to refer to a set of contiguous addresses of some size.

Note : Some authors use cache line to refer to a cache block, in the literature.

General working principle of cache based systems

When a read req is received from CPU, the contents of a block of memory words containing the location specified are transferred to the cache one word at a time. Subsequently, when the program asks for any of the words from this block, the desired word is fetched directly from the cache.