MODULE 3 ACA

Hardware Technologies 2: Bus Systems, Cache Memory Organizations, Shared Memory Organizations, Sequential and Weak Consistency Models, Pipelining and Superscalar Techniques, Linear Pipeline Processors, Nonlinear Pipeline Processors. For all Algorithms or mechanisms any one example is sufficient.

1. Discuss bus arbitration and its types in multiprocessor systems.
2. Illustrate sequential and weak consistency models.
3. Discuss any two mapping techniques.
4. Explain with diagram Backplane bus Specification
5. Explain the following terms associated with cache and memory architecture
    I. Low order memory interleaving
    II. Atomic v/s non atomic memory
    III. Physical address cache vs virtual address cache
    IV. Memory bandwidth and fault tolerance

Bus Systems

Bus Systems

- The system bus operates on a contention basis, meaning multiple devices can request bus access simultaneously.
- Only one device is granted access to the bus at any given time.
- The effective bandwidth for each processor is inversely proportional to the number of processors contending for the bus.
- Due to this limitation, bus-based commercial multiprocessors are typically small, ranging from 4 to 16 processors, due to their simplicity and low cost.

5.1.1 Backplane Bus Specification

- A backplane bus connects processors, data storage, and peripherals in tightly coupled hardware.
- It allows communication between devices without disrupting their internal activities.
- Timing protocols and operational rules ensure orderly data transfers and arbitration among multiple requests.

Data Transfer Bus (DTB):

- Comprises data, address, and control lines.
- Address lines broadcast data and device addresses proportional to the log of the address space size.
- Data lines are proportional to memory word length.
- Control lines manage read/write operations, timing, and bus error conditions.

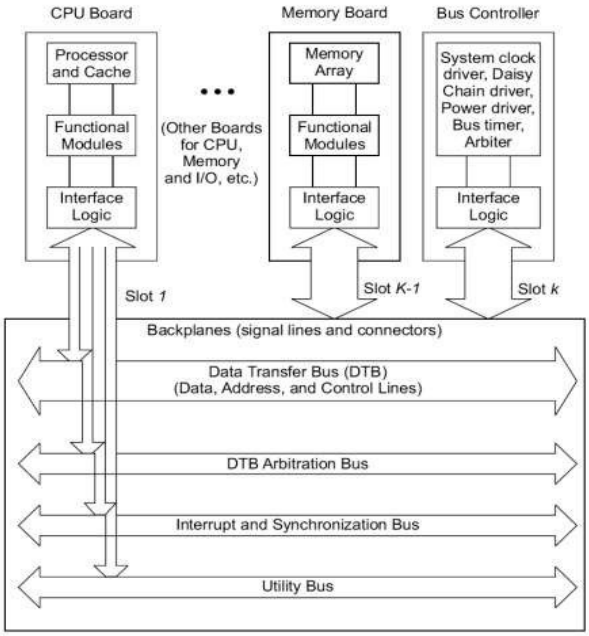


Fig. 5.1 Backplane buses, system interfaces, and slot connections to various functional boards in a multiprocessor system

Bus Arbitration and Control:

- Arbitration assigns control of the DTB to a requester, called a master, while the receiver is called a slave.
- Interrupt lines handle prioritized interrupts.
- Utility lines manage periodic timing, system clocking, and power sequences.
- The backplane includes signal lines and connectors, with a bus controller board for control logic.

Functional Modules:

- **Arbiter:** Grants DTB control to one requester.
- **Bus Timer:** Measures and terminates DTB cycle times.
- **Interrupter Module:** Manages interrupt requests and provides status/ID information.
- **Location Monitor:** Monitors DTB data transfers.
- **System Clock Driver:** Provides clock timing signals.

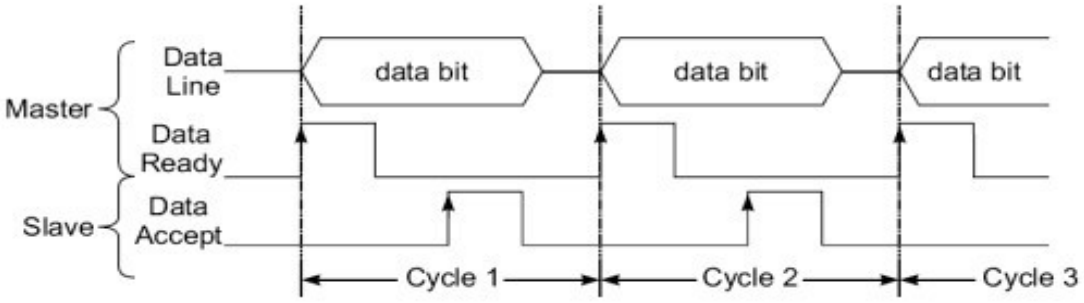
Physical Limitations:

- Limited number of boards can be plugged into a single backplane.
- Multiple backplane buses can be mounted on the same chassis.
- Scaling is limited by packaging constraints.

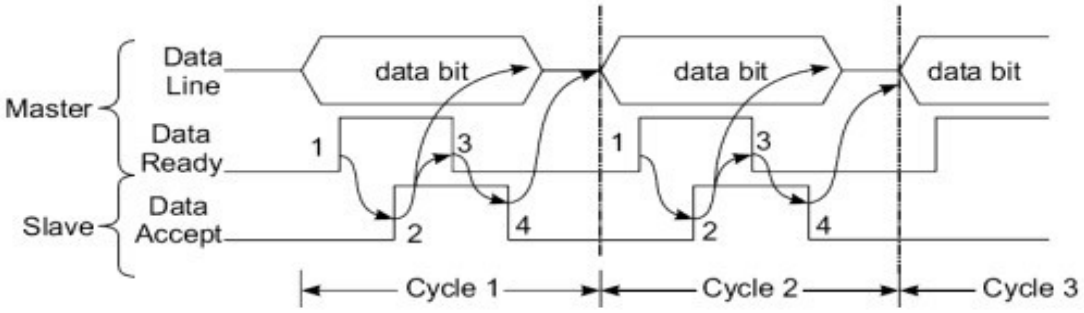
Timing Protocols

Synchronous Timing:

- Bus transactions occur at fixed clock edges.
- Clock cycle time is determined by the slowest device on the bus.
- D



(a) Synchronous bus timing with fixed-length clock signals for all devices



(b) Asynchronous bus timing using a four-edge handshaking (interlocking with variable length signals for different speed devices.

Fig. 5.3 Synchronous versus asynchronous bus timing protocols

Asynchronous Timing:

- Based on handshaking mechanisms without fixed clock cycles.
- A four-edge handshaking process ensures variable length clock signals for different speed devices.
- More flexible but complex and costly.

Arbitration, Transaction and Interrupt

Arbitration

- Process of selecting next bus master

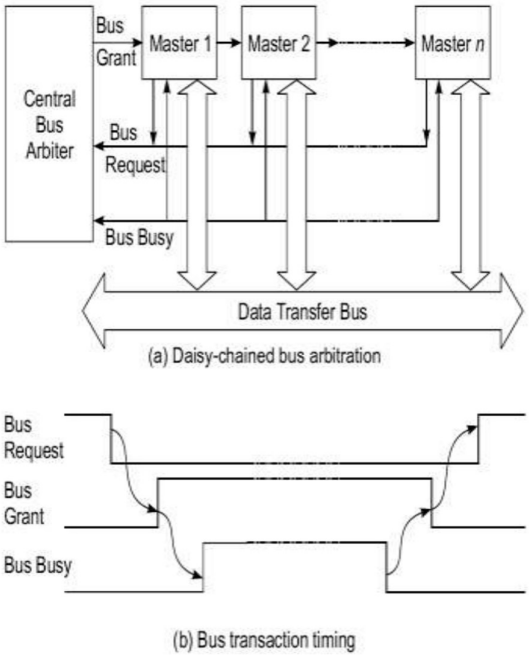
- Bus tenure is duration of master's control
- It restricts the tenure of the bus to one master at a time.
- Competing requests must be arbitrated on a fairness or priority basis
- Arbitration competition and bus transactions take place concurrently on a parallel bus over separate lines

Central Arbitration

- Uses a central arbiter as shown in Fig 5.4a

- Potential masters are daisy chained in a cascade
- A special signal line propagates *bus-gra`nt* from first master (at slot 1) to the last master (at slot n).

- All requests share the same *bus-request* line
- The *bus-request* signals the rise of the *bus-grant* level, which in turn raises the *bus-busy* level as shown in Fig. 5.4b.



g. 5.4 Central bus arbitration using shared requests and daisy-chained bus grants with a fixed priorit

- Simple scheme
- Easy to add devices
- Fixed-priority sequence – not fair
- Propagation of bus-grant signal is slow
- Not fault tolerant

Independent Requests and Grants

- Provide independent bus-request and grant signals for each master as shown in Fig5.5a.
- No daisy chaining is used in this scheme.
- Require a central arbiter, but can use a priority or fairness based policy
- More flexible and faster than a daisy-chained policy
- Larger number of lines – costly

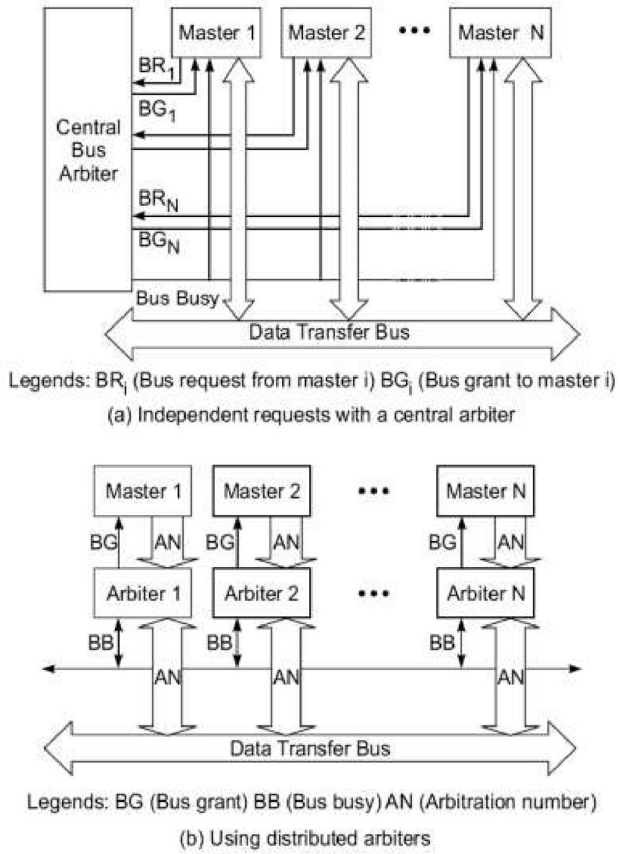


Fig. 5.5 Two bus arbitration schemes using independent requests and distributed arbiters, respectively

Distributed Arbitration

- Each master has its own arbiter and unique arbitration number as shown in Fig. 5.5b.
- Uses arbitration number to resolve arbitration competition
- When two or more devices compete for the bus, the winner is the one whose arbitration number is the largest determined by Parallel Contention Arbitration..
- All potential masters can send their arbitration number to shared-bus request/grant (SBRG) lines and compare its own number with SBRG number.
- If the SBRG number is greater, the requester is dismissed. At the end, the winner's arbitration number remains on the arbitration bus. After the current bus transaction is completed, the winner seizes control of the bus.
- Priority based scheme

Transfer Modes

- ***Address-only* transfer**: no data
- ***Compelled-data* transfer**: Address transfer followed by a block of one or more data transfers to one or more contiguous address.
- ***Packet-data* transfer**: Address transfer followed by a fixed-length block of data transfers from set of continuous address.
- ***Connected***: carry out master's request and a slave's response in a single bus transaction
- ***Split***: splits request and response into separate transactions
  - Allow devices with long latency or access time to use bus resources more efficiently
  - May require two or more connected bus transactions

Interrupt Mechanisms

- ***Interrupt****:* is a request from I/O or other devices to a processor for service or attention
- A priority interrupt bus is used to pass the interrupt signals
- Interrupter must provide status and identification information
- Have an interrupt handler for each request line
- Interrupts can be handled by message passing on data lines on a time-sharing basis.
  - Save lines, but use cycles

Use of time-shared data bus lines is a *virtual-interrupt*

Cache Memory Organizations

Cache memory lies between registers and RAM, holding recently used data and/or instructions.

Cache Addressing Models

- **Private Caches:** Used by most multiprocessor systems with an interconnection network between caches and main memory.
- **Addressing Models:**
  - **Physical Address Cache:** Cache is indexed and tagged with physical address. No aliasing problems but slower due to address translation.

- **Virtual Address Cache:** Cache is indexed or tagged with virtual address. Faster for hits but has aliasing problems.

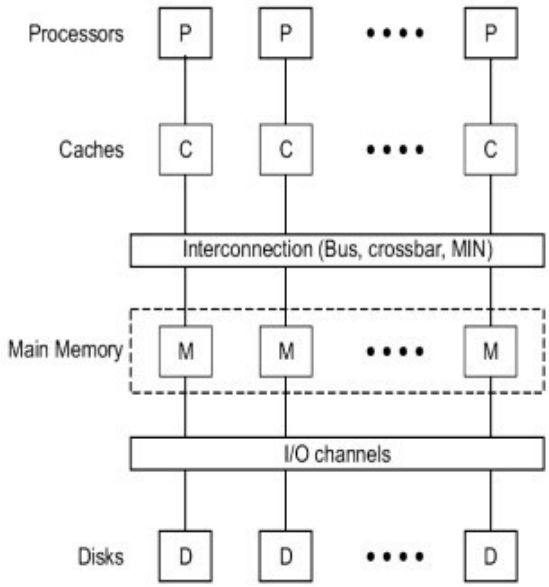
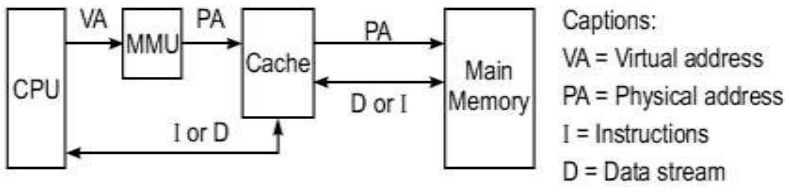


Fig. 5.6 A memory hierarchy for a shared-memory multiprocessor

Physical Address Cache

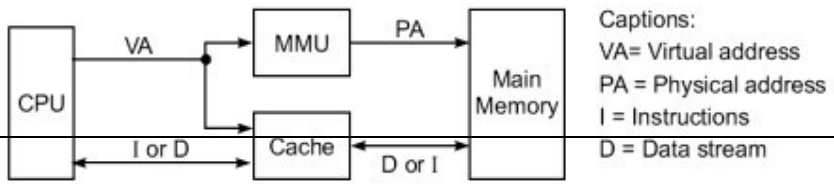
- Addressed by physical address.
- Cache lookup occurs after address translation.
- Advantages: No cache flushing on context switch, no aliasing issues.



(a) A unified cache accessed by physical address

Virtual Address Cache

- Addressed by virtual address.
- Parallel cache lookup and address translation.
- Advantages: Faster hits, efficient access.
- Disadvantages: Requires cache flushing on context switch, aliasing problems.



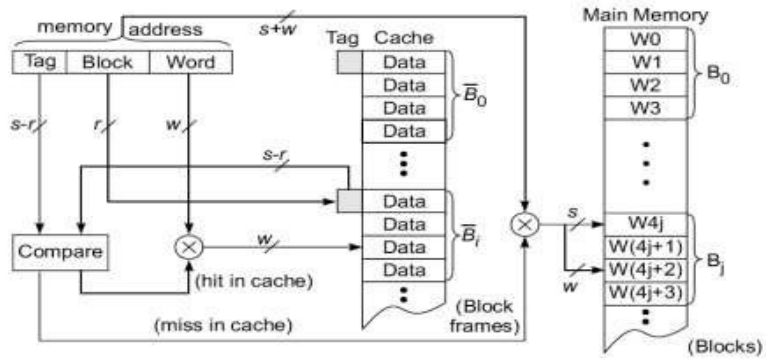
Direct Mapping Cache and Associative Cache

Direct Mapping Cache

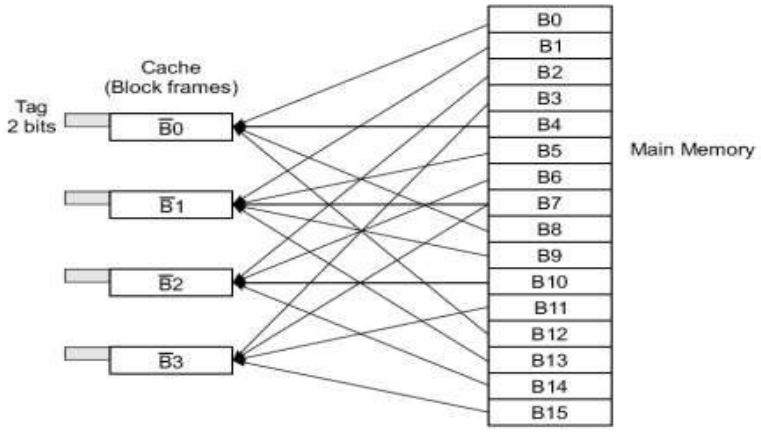
Overview: Direct Mapping Cache is a simple and efficient caching technique where each block of main memory is mapped to a specific cache line. This one-to-one mapping simplifies the hardware design but comes with some limitations in terms of flexibility and hit ratio.

Mechanism:

- **Memory Block Mapping:** Each memory block is mapped to a unique cache line using a modulo operation. For example, if there are N cache lines, a memory block with address A is mapped to cache line A mod N.
- **Cache Structure:** The cache is divided into several lines, each capable of holding one block of data.
- **Address Breakdown:**
  - **Tag:** Part of the address used to identify if the correct block is stored in the cache line.
  - **Index:** Part of the address used to determine which cache line the block is mapped to.
  - **Offset:** Used to locate the specific byte within the block.



(a) The cache/memory addressing



(b) Block B_j can be mapped to block frame $\overline{B}_i$ if $i = j$ (modulo 4)

Fig. 5.9 Direct-mapping cache organization and a mapping example

Advantages:

- **Simplicity:** The hardware required for direct mapping is straightforward and easy to implement.
- **Speed:** The simplicity of the mapping allows for fast cache access since the line is directly calculated using the index.

Disadvantages:

- **Rigid Mapping:** Each memory block can only be placed in one specific cache line. This can lead to frequent conflicts if multiple blocks map to the same line.
- **Poorer Hit Ratio:** Due to the rigid mapping, there are higher chances of cache misses, especially if programs frequently access multiple blocks that map to the same line.
- **No Page Replacement Policy:** The absence of a replacement policy can lead to inefficiencies, as the system cannot dynamically manage the contents of the cache to optimize hit rates.

Associative Cache

Overview: Associative Cache, also known as fully associative cache, allows any block of memory to be loaded into any cache line. This flexibility improves the hit ratio but requires more complex hardware to manage.

Mechanism:

- **Block Placement:** A memory block can be placed in any cache line. This eliminates the rigid mapping constraint of direct-mapped caches.
- **Address Breakdown:**
  - **Tag:** The entire address minus the block offset is used as the tag. This tag is compared against all tags in the cache to find the required block.
  - **Offset:** Used to locate the specific byte within the block.

Advantages:

- **Flexibility:** Any memory block can be placed in any cache line, significantly reducing the chances of conflicts.
- **Improved Hit Ratio:** The ability to place blocks anywhere in the cache leads to a higher hit ratio, as the cache can better adapt to the access patterns of the program.

Disadvantages:

- **Complexity:** The hardware required to search through all tags in parallel (associative search) is complex and costly.
- **Speed:** Associative search can slow down the cache access time due to the need to check all lines for a match.

Sector Mapping Cache

Overview: Sector Mapping Cache combines features of direct mapping and associative caches. It organizes the cache into sectors, each containing multiple blocks, and allows fully associative searches within these sectors.

Mechanism:

- **Sector Organization:** The cache is divided into sectors, each capable of holding several blocks.

- **Block Placement:** Within each sector, blocks can be placed in any line. However, sectors themselves are accessed using a direct mapping approach.
- **Associative Search:** When accessing a block, the system performs a fully associative search within the sector to find the required block.
- **Address Breakdown:**
  - **Sector Index:** Part of the address used to determine which sector the block is mapped to.
  - **Tag:** Used to identify the specific block within the sector.
  - **Offset:** Used to locate the specific byte within the block.

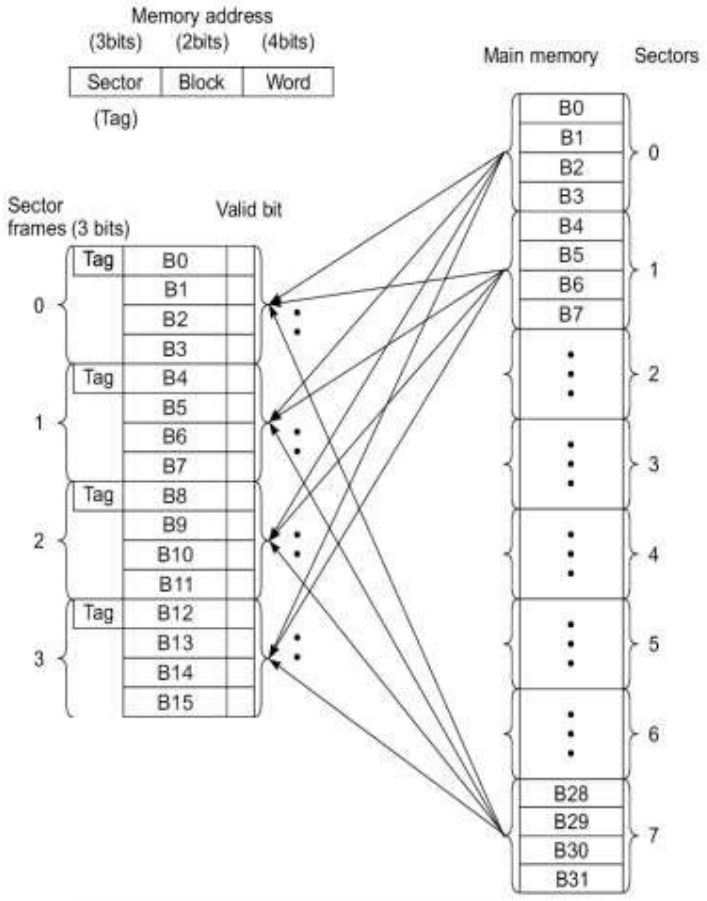


Fig. 5.12 A four-way sector mapping cache organization

Advantages:

- **Flexibility:** Allows flexible placement of blocks within sectors, reducing conflicts compared to direct mapping.
- **Block Replacement Algorithms:** Can implement various block replacement algorithms (e.g., LRU, FIFO) within each sector to optimize performance.

Disadvantages:

- **Complexity:** Requires more complex hardware to manage associative searches within sectors and maintain sector tags.
- **Cost:** The additional hardware for associative searches and block replacement algorithms increases the cost and power consumption of the cache system.

Shared Memory Organization

Interleaved memory organization involves dividing memory into multiple modules that can be accessed simultaneously. This technique is used to improve memory bandwidth and reduce access latency. The two main types of interleaving are low-order interleaving and high-order interleaving.

Low-Order Interleaving

Overview: Low-order interleaving spreads contiguous memory locations across different memory modules. This approach supports block access, making it suitable for applications requiring high data throughput.

Mechanism:

- **Address Mapping:** In low-order interleaving, the lower bits of the memory address determine the memory module, while the higher bits determine the location within the module. For instance, if there are NNN memory modules, an address A is divided such that:

$$\text{Module} = A \mod N$$

$$\text{Location within module} = \frac{A}{N}$$

- **Parallel Access:** Contiguous memory addresses are distributed across different modules, allowing multiple addresses to be accessed in parallel. For example, addresses 0, 1, 2, 3, ... would be distributed across modules 0, 1, 2, 3, ... in a round-robin fashion.
- **Block Access:** Since contiguous blocks of data are spread across modules, it is possible to fetch large blocks of data in parallel, which is beneficial for operations like vector processing or matrix multiplication.

Advantages:

- **Improved Bandwidth:** Parallel access to multiple modules increases memory bandwidth.
- **Efficient Block Access:** Suitable for applications that require frequent access to large blocks of data.

Disadvantages:

- **Complex Address Calculation:** The need to calculate the module and location within the module adds complexity to the address decoding logic.
- **Potential for Idle Modules:** If the access pattern does not utilize the parallelism, some modules may remain idle, reducing the effectiveness of the interleaving.

High-Order Interleaving

Overview: High-order interleaving assigns contiguous memory locations to the same memory module. This method does not support block access and is typically used in systems where access patterns are more random or localized.

Mechanism:

- **Address Mapping:** In high-order interleaving, the higher bits of the memory address determine the memory module, while the lower bits determine the location within the module. For example, if there are NNN memory modules, an address Ais divided such that:

$$\text{Module} = \frac{A}{M} \quad \text{(higher-order bits)}$$

$$\text{Location within module} = A \mod M \quad \text{(lower-order bits)}$$

- **Localized Access:** Contiguous memory addresses are located within the same module, which simplifies the address calculation and can be beneficial for localized data access patterns.

Advantages:

- **Simpler Address Calculation:** The mapping of addresses to modules is straightforward.
- **Suitable for Localized Access:** Efficient for applications where memory access patterns are localized, such as certain types of numerical simulations.

Disadvantages:

- **Limited Parallelism:** Does not support block access or parallel access to contiguous addresses.
- **Potential Bottlenecks:** Contiguous data access can create bottlenecks, as all accesses go to the same module.

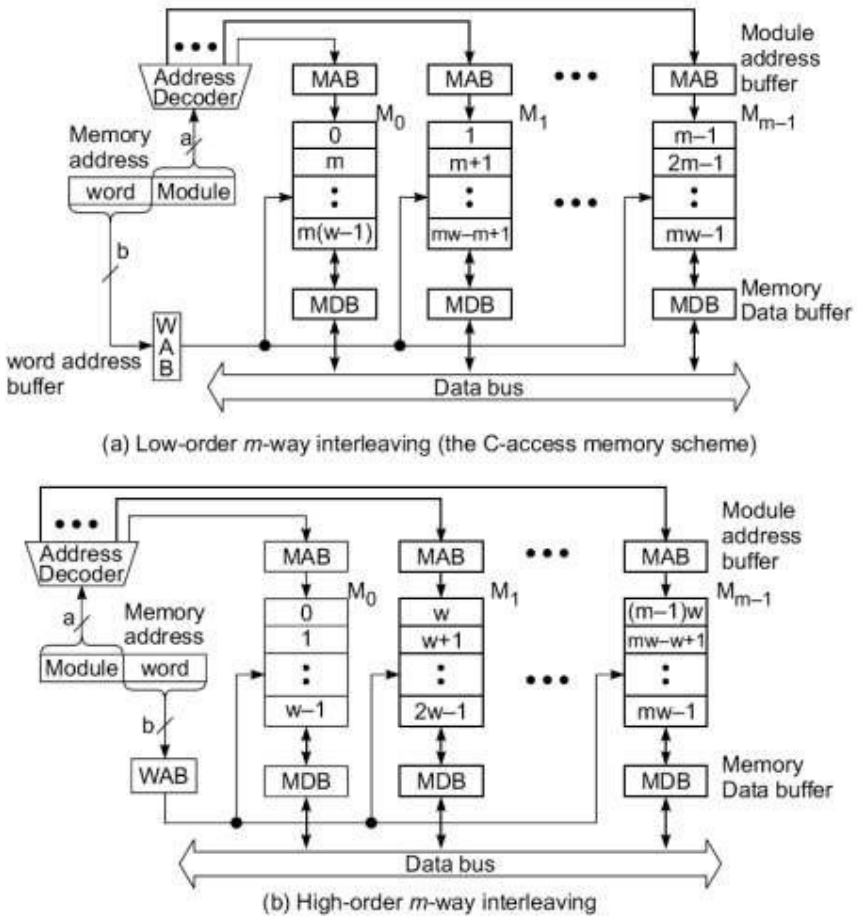


Fig. 5.15 Two interleaved memory organizations with $m = 2^a$ modules and $w = 2^b$ words per module (word addresses shown in boxes)

Pipelined Memory Access

Overview: Pipelined memory access is a technique used to improve the effective access time to memory by organizing memory operations in a pipelined manner. This approach involves breaking down memory access operations into multiple stages, allowing different stages to be executed concurrently.

Mechanism:

- **Memory Cycle Division:** The memory access cycle is divided into major and minor cycles. Major cycles handle overall memory access, while minor cycles handle sub-tasks within the access operation.
- **Pipelining Stages:** Memory access is broken down into stages such as address calculation, address decoding, data fetching, and data return. Each stage can operate in parallel with other stages for different memory accesses.
- **Overlapping Accesses:** By overlapping different stages of multiple memory accesses, the pipeline ensures that the memory modules are utilized more efficiently.

Example: Consider a memory system with a four-stage pipeline:

1. **Stage 1:** Address calculation.
2. **Stage 2:** Address decoding.
3. **Stage 3:** Data fetching.
4. **Stage 4:** Data return.

In each clock cycle, a new memory access request enters Stage 1, while previous requests move to the next stage. This overlapping allows for multiple memory operations to be in progress simultaneously.

Memory Allocation Schemes

Virtual Memory:

- Allows multiple processes to use main memory simultaneously.
- **Allocation Policies:**
  - **Memory Swapping:** Moves data blocks between memory levels.
  - **Nonpreemptive Allocation:** Swaps out allocated processes when memory is full.
  - **Preemptive Allocation:** Can interrupt and swap out executing processes.
  - **Local Allocation:** Focuses on the fault-causing process's working set.
  - **Global Allocation:** Considers all processes' working sets.

Swapping Systems:

- **Process-Level Swapping:** Swaps entire processes.
- **Swap Device and Space:** Disk portion used for temporary storage of swapped data.

Swapping in UNIX:

- Swapping triggered by creating child processes, increasing address space, stack space demand, or returning swapped-out processes.

Demand Paging Systems:

- Transfers only needed pages between main memory and swap device.
- Allows process address space to exceed physical memory.

Working Sets:

- Set of pages used by a process in the last 'n' references, keeping active process sets in memory.

Sequential and Weak Consistency Models

Sequential Consistency Model

- **Sufficient conditions:**

1. Before a *load* is allowed to perform wrt any other processor, all previous *loads* must be globally performed and all previous *stores* must be performed wrt all processors
2. Before a *store* is allowed to perform wrt any other processor, all previous *loads* must be globally performed and all previous *stores* must be performed wrt to all processors

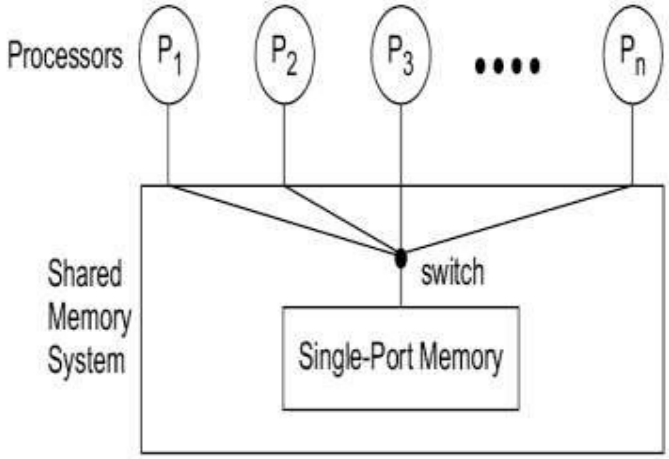


Fig. 5.20 Sequential consistency memory model (Courtesy of Sindhu, Frailong, and Cekleov; reprinted with permission from *Scalable Shared-Memory Multiprocessors*, Kluwer Academic Publishers, 1992)

Sequential Consistency Axioms

1. A *load* always returns the value written by the latest *store* to the same location time over all *loads*/*stores*
2. The memory order conforms to a total binary order in which shared memory is accessed in real
3. If two operations appear in particular program order, same memory order
4. *Swap* op is atomic with respect to *stores*. No other *store* can intervene between *load* and *store* parts of *swap*
5. All *stores* and *swaps* must eventually terminate

Implementation Considerations

- A single port software services one op at a time
- Order in which software is thrown determines global order of memory access ops
- *Strong ordering* preserves the program order in all processors

- Sequential consistency model leads to poor memory performance due to the imposed strong ordering of memory events

5.4.3 Weak Consistency Models

- Multiprocessor model may range from strong (sequential) consistency to various degrees of weak consistency

- Two models considered
  - DSB (Dubois, Scheurich and Briggs) model
  - TSO (Total Store Order) model

DSB Model

Dubois, Scheurich and Briggs have derived a weak consistency model by relating memory request ordering to synchronization points in the program. We call this the DSB model specified by the following 3 conditions:

1. All previous *synchronization* accesses must be performed, before a *load* or a *store* access is allowed to perform wrt any other processor.
2. All previous *load and store* accesses must be performed, before a *synchronization* access is allowed to perform wrt any other processor.

3. *Synchronization* accesses sequentially consistent with respect to one another

TSO Model

Sindhu, Frailong and Cekleov have specified the TSO weak consistency model with 6 behavioral axioms.

1. *Load* returns latest *store* result
2. Memory order is a total binary relation over all pairs of *store* operations
3. If two *stores* appear in a particular program order, then they must also appear in the same memory order
4. If a memory operation follows a *load* in program order, then it must also follow *load* in memory order

5. *A swap* operation is atomic with respect to other *stores* – no other *store* can interleave between *load/store* parts of *swap*
6. All *stores and swaps* must eventually terminate.

Chapter-6 Pipelining and Superscalar Techniques

Linear Pipeline Processors

A linear pipeline processor is a cascade of processing stages which are linearly connected to perform a fixed function over a stream of data flowing from one end to the other.

In modern computers, linear pipelines are applied for instruction execution, arithmetic computation, and memory-access operations.

6.1.1 Asynchronous & Synchronous models

- A linear pipeline processor is constructed with k processing stages. External inputs(operands) are fed into the pipeline at the first stage $\mathbf{S_1}$.
- The processed results are passed from stage $\mathbf{S_i}$ to stage $\mathbf{S_{i+1}}$, for all i=1,2,….,k-1. The final result emerges from the pipeline at the last stage $\mathbf{S_n}$.
- Depending on the control of data flow along the pipeline, we model linear pipelines in two categories: **Asynchronous** and **Synchronous**.

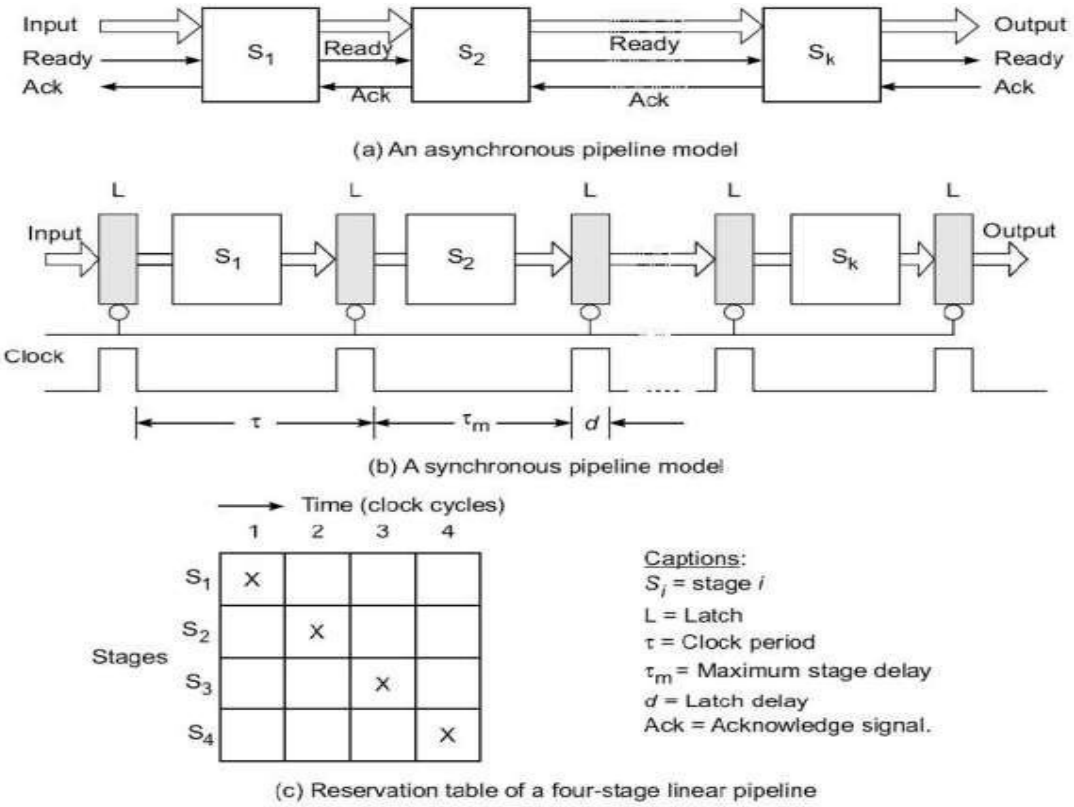


Fig. 6.1 Two models of linear pipeline units and the corresponding reservation table

Asynchronous Model

- As shown in the figure data flow between adjacent stages in an asynchronous pipeline is controlled by a handshaking protocol.
- When stage S_i is ready to transmit, it sends a ready signal to stage S_{i+1}. After stage receives the incoming data, it returns an acknowledge signal to S_i.
- Asynchronous pipelines are useful in designing communication channels in message- passing multicomputers where pipelined wormhole routing is practiced Asynchronous pipelines may have a variable throughput rate.
- Different amounts of delay may be experienced in different stages.

Synchronous Model:

- Synchronous pipelines are illustrated in Fig. Clocked latches are used to interface between stages.
- The latches are made with master-slave flip-flops, which can isolate inputs from outputs.
- Upon the arrival of a clock pulse All latches transfer data to the next stage simultaneously.
- The pipeline stages are combinational logic circuits. It is desired to have approximately equal delays in all stages.

- These delays determine the clock period and thus the speed of the pipeline. Unless otherwise specified, only synchronous pipelines are studied.
- The utilization pattern of successive stages in a synchronous pipeline is specified by a reservation table.

- For a linear pipeline, the utilization follows the diagonal streamline pattern shown in Fig. 6.1c.

- This table is essentially a space-time diagram depicting the precedence relationship in using the pipeline stages.
- Successive tasks or operations are initiated one per cycle to enter the pipeline. Once the pipeline is filled up, one result emerges from the pipeline for each additional cycle.
- This throughput is sustained only if the successive tasks are independent of each other.

I. Low Order Memory Interleaving

- **Definition**: A memory interleaving technique where consecutive memory addresses are distributed across multiple memory modules in a round-robin fashion.
- **Example**: If we have four memory modules (M0, M1, M2, M3), addresses would be assigned as follows:
  - Address 0 -> M0
  - Address 1 -> M1
  - Address 2 -> M2
  - Address 3 -> M3
  - Address 4 -> M0, and so on.
- **Benefits**:
  - Increases parallelism by allowing multiple memory accesses simultaneously.
  - Reduces contention and improves memory throughput.
- **Drawbacks**:
  - Less fault-tolerant; if one module fails, it can disrupt the whole system as consecutive data is spread across modules.

II. Atomic vs Non-Atomic Memory

- **Atomic Memory**:
  - Operations (like read-modify-write) appear to be indivisible.

  - Ensures that no other operation can access the memory location during its operation.
  - Essential for synchronization mechanisms in multiprocessor systems to avoid race conditions.
- **Non-Atomic Memory**:
  - Operations can be interrupted, and other processors can access the memory location mid-operation.
  - More prone to inconsistencies and race conditions without proper synchronization mechanisms.
- **Example**:
  - **Atomic**: Incrementing a counter in shared memory (the read-modify-write sequence is uninterrupted).
  - **Non-Atomic**: If two processors increment the counter simultaneously without atomicity, the final result may be incorrect due to interleaving.

III. Physical Address Cache vs Virtual Address Cache

- **Physical Address Cache**:
  - Caches data using physical addresses.
  - Requires address translation from virtual to physical before accessing the cache.
  - Benefits: Avoids synonym problems (where different virtual addresses map to the same physical address).
  - Drawbacks: Can introduce translation delay, slightly slower than virtual caches.
- **Virtual Address Cache**:
  - Caches data using virtual addresses.
  - Accesses the cache before address translation.
  - Benefits: Faster access as it bypasses the translation step initially.
  - Drawbacks: Can have synonym problems, where multiple virtual addresses point to the same physical address, leading to potential cache coherence issues.

IV. Memory Bandwidth and Fault Tolerance

- **Memory Bandwidth**:
  - The rate at which data can be read from or written to memory by the processor.
  - Measured in bytes/second or bits/second.
  - Higher memory bandwidth means faster data access and improved system performance.
  - Techniques to improve bandwidth include memory interleaving, wider data buses, and faster memory technologies.

- **Fault Tolerance**:
  - The ability of a system to continue functioning correctly even when some components fail.
  - **In Memory Systems**:
    - High-order interleaving provides better fault isolation as entire blocks of data are stored contiguously within each module.
    - Low-order interleaving spreads data across modules, making it less fault-tolerant because a single module failure can disrupt data across the whole memory space.
  - Techniques for fault tolerance include error-correcting codes (ECC), redundant memory modules, and robust memory management techniques that can detect and recover from failures.