

UNIT-V

Parallel Processors

Introduction to parallel processors

Parallel processing is a term used to denote a large class of techniques that are used to provide simultaneous data-processing tasks for the purpose of in a easing the computational speed of a computer system. Instead of processing each instruction sequentially as in a conventional computer, a parallel processing system is able to perform concurrent data processing to achieve faster execution time.

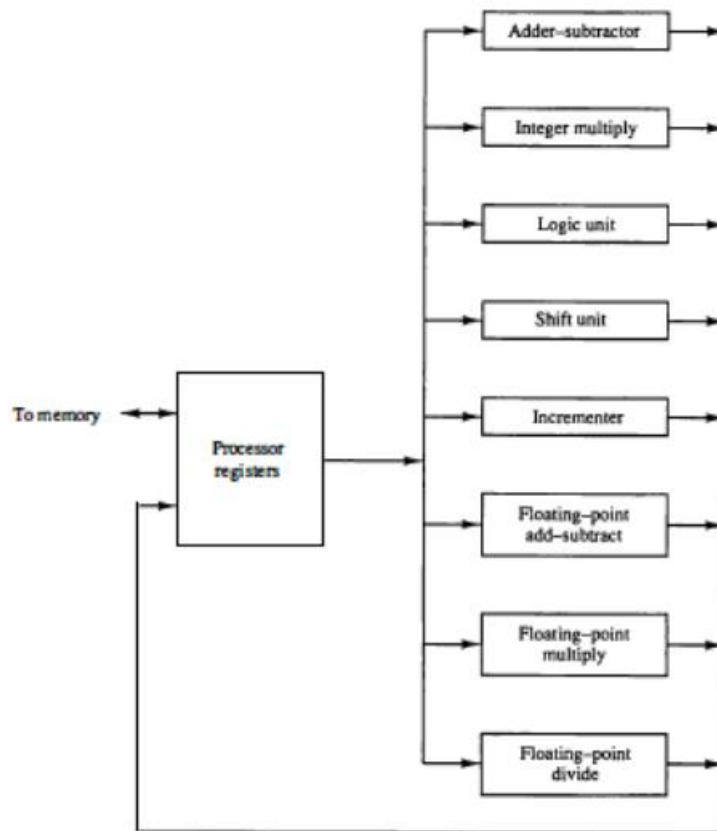
The purpose of parallel processing is to speed up the computer processing capability and increase its throughput, that is, the amount of processing that can be accomplished during a given interval of time. The amount of hardware increases with parallel processing and with it, the cost of the system increases. However, technological developments have reduced hardware costs to the point where parallel processing techniques a.re economically feasible.

Parallel processing can be viewed from various levels of complexity. At the lowest level, we distinguish between parallel and serial operations by the type of registers used. Shift registers operate in serial fashion one bit at a time, while registers with parallel load operate with all the bits of the word simultaneously.

Parallel processing at a higher level of complexity can be achieved by having a multiplicity of functional units that perform identical or different operations simultaneously. Parallel processing is established by distributing the data among the multiple functional units. For example, the arithmetic, logic, and shift operations can be separated into three units and the operands diverted to each unit under the supervision of a control unit.

Figure 9-1 shows one possible way of separating the execution unit into eight functional units operating in parallel. The operands in the registers are applied to one of the units depending on the operation specified by the instruction associated with the operands. The operation performed in each functional unit is indicated in each block of the diagram. The adder and integer multiplier perform the arithmetic operations with integer numbers.

Figure 9-1 Processor with multiple functional units.



There are a variety of ways that parallel processing can be classified. It can be considered from the internal organization of the processors, from the interconnection structure between processors, or from the flow of information through the system. One classification introduced by M. J. Flynn considers the organization of a computer system by the number of instructions and data items that are manipulated simultaneously. The normal operation of a computer is to fetch instructions from memory and execute them in the processor.

The sequence of instructions read from memory constitutes an instruction stream . The operations performed on the data in the processor constitutes a data stream . Parallel processing may occur in the instruction stream, in the data stream, or in both.

Flynn's classification divides computers into four major groups as follows:

- Single instruction stream, single data stream (SISD)
- Single instruction stream, multiple data stream (SIMD)
- Multiple instruction stream, single data stream (MISD)
- Multiple instruction stream, multiple data stream (MIMD)

SISD represents the organization of a single computer containing a control unit, a processor unit, and a memory unit. Instructions are executed sequentially and the system may or may not have internal parallel processing capabilities. Parallel processing in this case may be achieved by means of multiple functional units or by pipeline processing.

SIMD represents an organization that includes many processing units under the supervision of a common control unit. All processors receive the same instruction from the control unit but operate on different items of data. The shared memory unit must contain multiple modules so that it can communicate with all the processors simultaneously.

MISD structure is only of theoretical interest since no practical system has been constructed using this organization.

MIMD organization refers to a computer system capable of processing several programs at the same time. Most multiprocessor and multicomputer systems can be classified in this category.

Concurrent access to memory and cache coherency:

The primary advantage of cache is its ability to reduce the average access time in uniprocessors. When the processor finds a word in cache during a read operation, the main memory is not involved in the transfer. If the operation is to write, there are two commonly used procedures to update memory.

Write-through policy: In the write-through policy, both cache and main memory are updated with every write operation.

Write-back policy: In the write-back policy, only the cache is updated and the location is marked so that it can be copied later into main memory.

In a shared memory multiprocessor system, all the processors share a common memory. In addition, each processor may have a local memory, part or all of which may be a cache. The compelling reason for having separate caches for each processor is to reduce the average access time in each processor. The same information may reside in a number of copies in some caches and main memory. To ensure the ability of the system to execute memory operations correctly, the multiple copies must be kept identical.

This requirement imposes a cache coherence problem. A memory scheme is coherent if the value returned on a load instruction is always the value given by the latest store instruction with the same address. Without a proper solution to the cache coherence problem, caching cannot be used in bus-oriented multiprocessors with two or more processors.

Conditions for Incoherence

Cache coherence problems exist in multiprocessors with private caches because of the need to share writable data. Read-only data can safely be replicated without cache coherence enforcement mechanisms.

To illustrate the problem, consider the three-processor configuration with private caches shown in Fig. 13-12. Sometime during the operation an element X from main memory is loaded into the three processors, P1, P2, and P3. As a consequence, it is also copied into the private caches of the three processors. For simplicity, we assume that X contains the value of 52. The load on X to the three processors results in consistent copies in the caches and main memory. If one of the processors performs a store to X, the copies of X in the caches become inconsistent. A load by the other processors will not return the latest value. Depending on the memory update policy used in the cache, the main memory may also be inconsistent with respect to the cache.

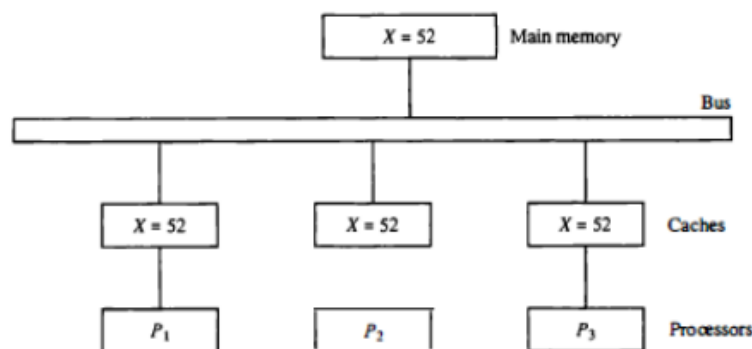
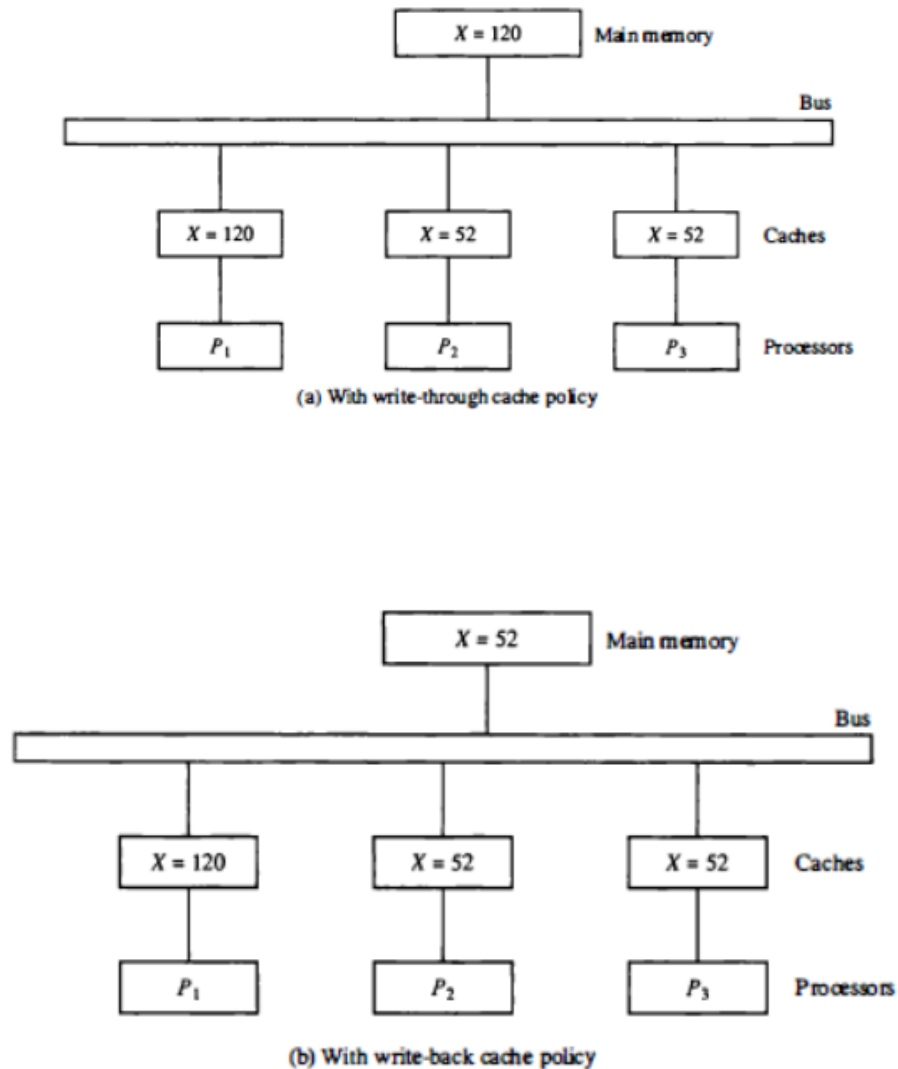


Figure 13-12 Cache configuration after a load on X.

This is shown in Fig. 13-13. A store to X (of the value of 120) into the cache of processor P1 updates memory to the new value in a write-through policy. A write-through policy maintains consistency between memory and the originating cache, but the other two caches are inconsistent since they still hold the old value. In a write-back policy, main memory is not updated at the time of the store. The copies in the other two caches and main memory are inconsistent. Memory is updated eventually when the modified data in the cache are copied back into memory.

Figure 13-13 Cache configuration after a store to X by processor P₁.



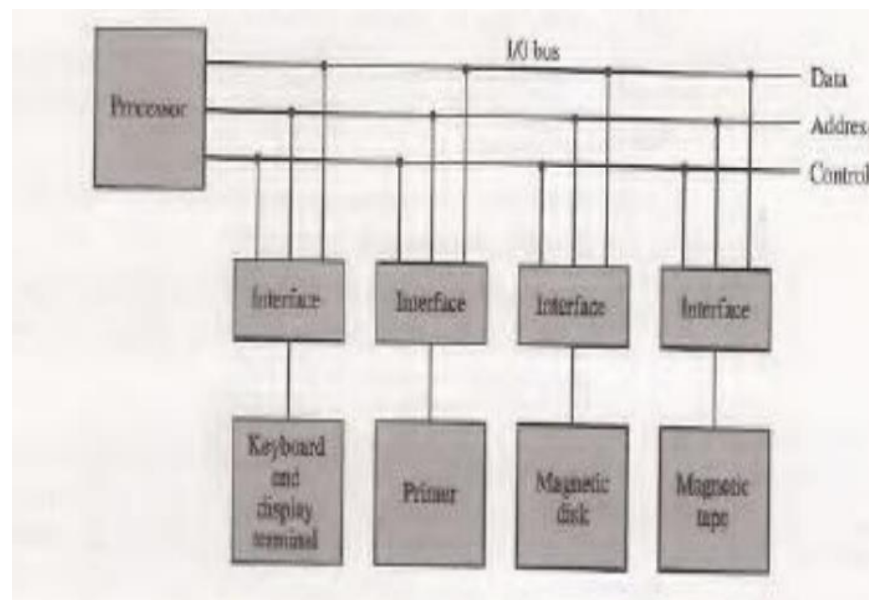
Another configuration that may cause consistency problems is a direct memory access (DMA) activity in conjunction with an IOP connected to the system bus. In the case of input, the DMA may modify locations in main memory that also reside in cache without updating the cache. During a DMA output, memory locations may be read before they are updated from the cache when using a write-back policy. VO-based memory incoherence can be overcome by making the IOP a participant in the cache coherent solution that is adopted in the system.

Input-Output Interface

Input-output interface provides a method for transferring information between internal storage and external I/O devices. Peripherals connected to a computer need special communication links for interfacing them with the central processing unit. The purpose of the communication link is to resolve the differences that exist between the central computer and each peripheral. The major differences are:

1. Peripherals are electromechanical and electromagnetic devices and their manner of operation is different from the operation of the CPU and memory, which are electronic devices. Therefore, a conversion of signal values may be required.
2. The data transfer rate of peripherals is usually slower than the transfer rate of the CPU, and consequently, a synchronization mechanism may be needed.
3. Data codes and formats in peripherals differ from the word format in the CPU and memory.
4. The operating modes of peripherals are different from each other and each must be controlled so as not to disturb the operation of other peripherals connected to the CPU.

To resolve these differences, computer systems include special hardware components between the CPU and peripherals to supervise and synchronize all input and output transfers. These components are called interface units because they interface between the processor bus and the peripheral device.

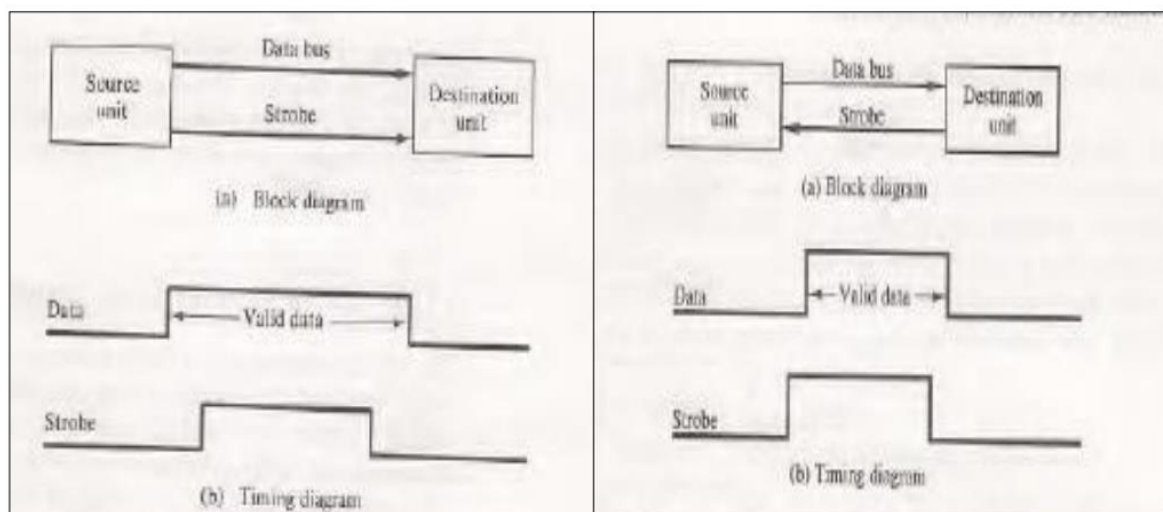


In addition to communicating with I/O, the processor must communicate with the memory unit. There are three ways that computer buses can be used to communicate with memory and I/O:

1. Use two separate buses, one for memory and the other for I/O.
2. Use one common bus for both memory and I/O but have separate control lines for each.
3. Use one common bus for memory and I/O with common control lines.

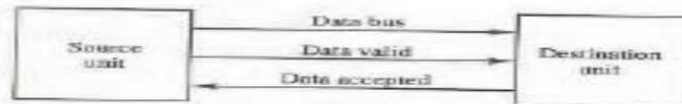
Asynchronous Data Transfer

- Asynchronous data transfer between two independent units requires that control signals be transmitted between the communicating units to indicate the time at which data is being transmitted.
- One way of achieving this is strobe by means of a strobe pulse supplied by one of the units to indicate to the other unit when the transfer has to occur.
- Another method commonly used is to accompany each data item being transferred with a control signal that indicates the presence of data in the bus.
- The unit receiving the data item responds with another control signal to acknowledge receipt of the data. This type of agreement between two independent units is referred to as handshaking.

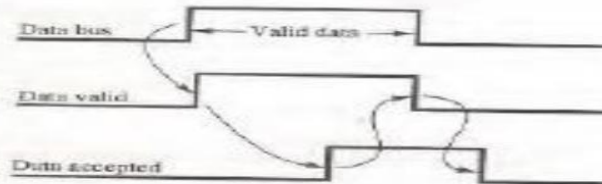


Source- initiated strobe for data transfer.

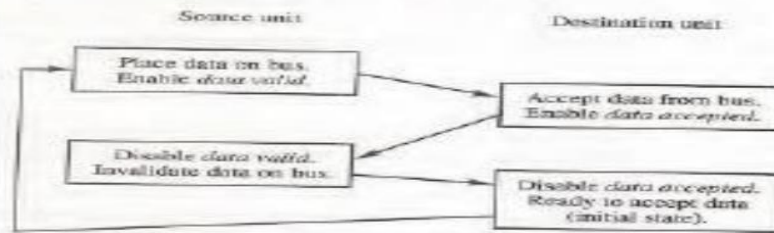
Destination- initiated strobe for data transfer.



(a) Block diagram

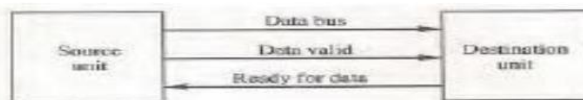


(b) Timing diagram

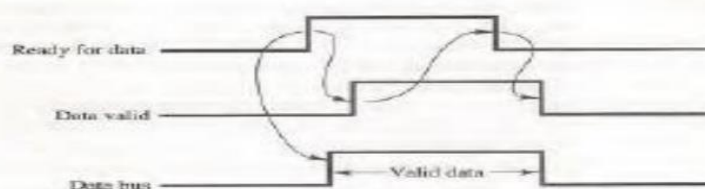


(c) Sequence of events

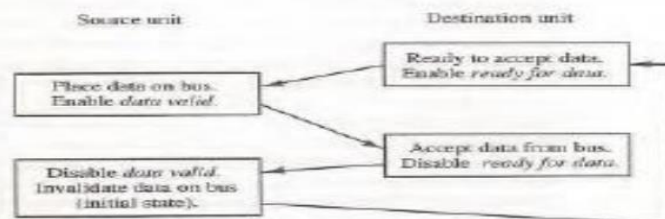
Source-initiated transfer using handshaking



(a) Block diagram



(b) Timing diagram



(c) Sequence of events

Destination-initiated transfer using handshaking

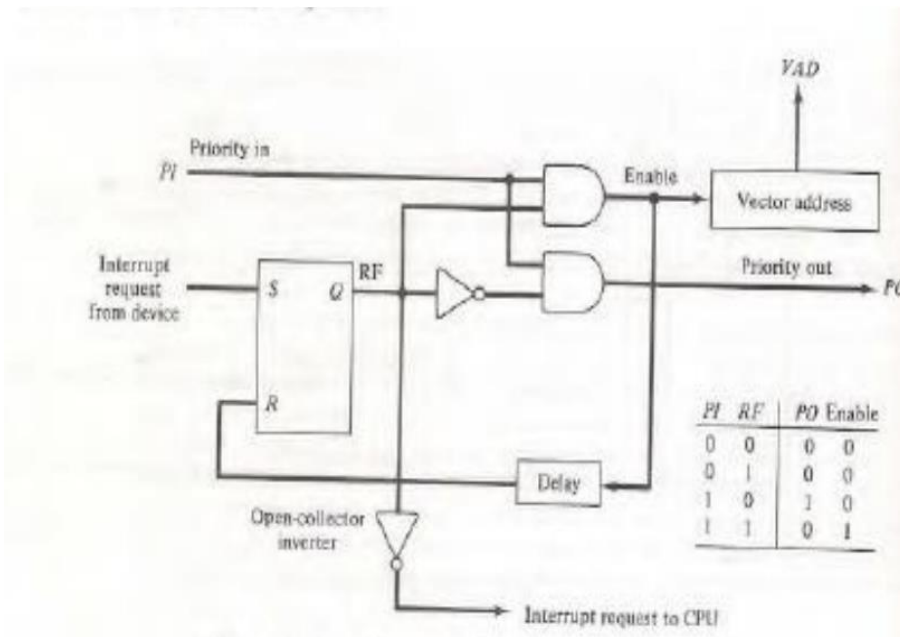
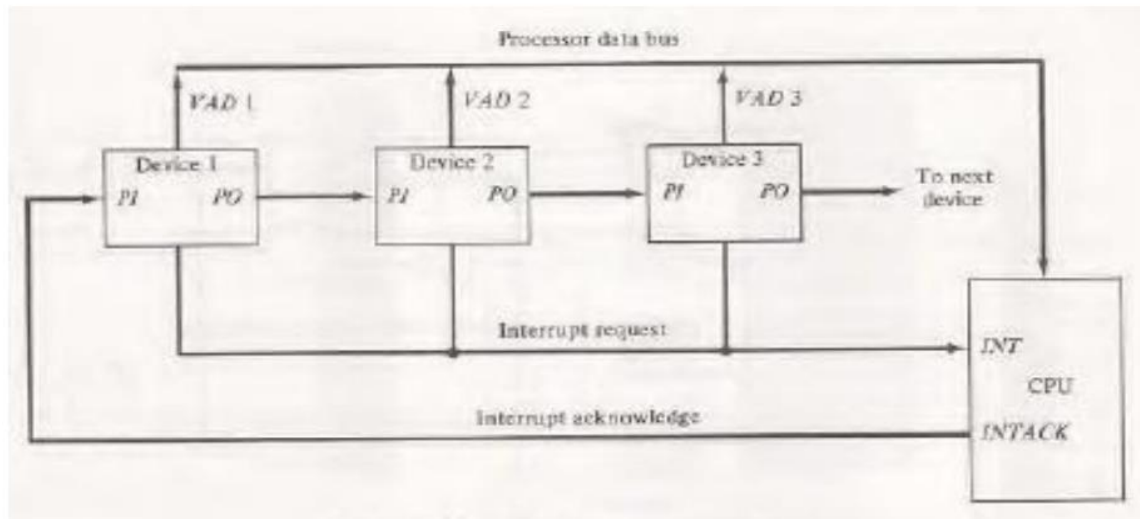
Modes of Transfer

- Binary information received from an external device is usually stored in memory for later processing. Information transferred from the central computer into an external device originates in the memory unit.
- The CPU merely executes the I/O instructions and may accept the data temporarily, but the ultimate source or destination is the memory unit. Data transfer between the central computer and I/O devices may be handled in a variety of modes. Some modes use the CPU as an intermediate path; others transfer the data directly to and from the memory unit.
- Data transfer to and from peripherals may be handled in one of three possible modes:
 1. Programmed I/O
 2. Interrupt-initiated I/O
 3. Direct memory access (DMA).

Priority Interrupt

- A priority interrupt is a system that establishes a priority over the various sources to determine which condition is to be serviced first when two or more requests arrive simultaneously.
- The system may also determine which conditions are permitted to interrupt the computer while another interrupt is being serviced.
- Higher-priority interrupt levels are assigned to requests which, if delayed or interrupted, could have serious consequences.
- Devices with high speed transfers such as magnetic disks are given high priority, and slow devices such as keyboards receive low priority.
- When two devices interrupt the computer at the same time, the computer services the device, with the higher priority first.

The daisy-chaining method of establishing priority consists of a serial connection of all devices that request an interrupt. The device with the highest priority is placed in the first position, followed by lower-priority devices up to the device with the lowest priority, which is placed last in the chain.



One stage of
the daisy-
chain
priority
arrangement

Direct Memory Access

The data transfer between a fast storage media such as magnetic disk and memory unit is limited by the speed of the CPU. Thus we can allow the peripherals directly communicate with each other using the memory buses, removing the intervention of the CPU. This type of data transfer technique is known as DMA or direct memory access. During DMA the CPU is idle and it has

no control over the memory buses. The DMA controller takes over the buses to manage the transfer directly between the I/O devices and the memory unit.

Bus Request: It is used by the DMA controller to request the CPU to relinquish the control of the buses.

Bus Grant: It is activated by the CPU to Inform the external DMA controller that the buses are in high impedance state and the requesting DMA can take control of the buses. Once the DMA has taken the control of the buses it transfers the data. This transfer can take place in many ways.

Types of DMA transfer using DMA controller:

Burst Transfer:

DMA returns the bus after complete data transfer. A register is used as a byte count, being decremented for each byte transfer, and upon the byte count reaching zero, the DMAC will release the bus. When the DMAC operates in burst mode, the CPU is halted for the duration of the data transfer.

Steps involved are:

1. Bus grant request time.
 2. Transfer the entire block of data at transfer rate of device because the device is usually slow than the speed at which the data can be transferred to CPU.
 3. Release the control of the bus back to CPU
- So, total time taken to transfer the N bytes = Bus grant request time + (N) * (memory transfer rate) + Bus release control time.

Where, $X \mu\text{sec}$ = data transfer time or preparation time (words/block) $Y \mu\text{sec}$ = memory cycle time or cycle time or transfer time (words/block) % CPU idle (Blocked) = $(Y/X+Y)*100$ % CPU Busy = $(X/X+Y)*100$

Cyclic Stealing:

An alternative method in which DMA controller transfers one word at a time after which it must return the control of the buses to the CPU. The CPU delays its operation only for one memory cycle to allow the direct memory I/O transfer to “steal” one memory cycle.

Steps Involved are:

1. Buffer the byte into the buffer
2. Inform the CPU that the device has 1 byte to transfer (i.e. bus grant request)
3. Transfer the byte (at system bus speed)
4. Release the control of the bus back to CPU.

In cycle stealing mode we always follow pipelining concept that when one byte is getting transferred then Device is parallel preparing the next byte. “The fraction of CPU time to the data transfer time” if asked then cycle stealing mode is used. Where, $X \mu\text{sec}$ =data transfer time or preparation time (words/block) $Y \mu\text{sec}$ =memory cycle time or cycle time or transfer time (words/block) % CPU idle (Blocked) $= (Y/X) * 100$ % CPU busy $= (X/Y) * 100$

Interleaved mode: In this technique, the DMA controller takes over the system bus when the microprocessor is not using it. An alternate half cycle i.e. half cycle DMA + half cycle processor.