UNIT IV

1	Explain in detail about memory technologies. MEMORY TECHNOLOGIES There are four primary technologies used. They are • Static Random Access Memory (SRAM). • Dynamic Random Access Memory (DRAM). • ROM and Flash Memory. • Magnetic Disk. **4.3.1. SRAM** □ It is very part of the RAM. It is main memory and located very close to the processor. □ SRAM's are simply integrated circuits that are memory array with a single access port that can provide either a read or write. □ SRAM have a fixed access time to any data, but the read and write access time may differ. □ SRAM don't need to refresh and so the access time is very close to the cycle time. □ SRAM use 6 to 8 transistors per bit. □ SRAM needs only minimal power to retain the charge in standby mode. **4.3.2. DRAM** □ In a SRAM, as long as power is applied the value can be kept indefinitely. □ In DRAM, the value is kept in a cell and is stored as a charge in a capacitor, □ DRAM has single transistor used to access this stored charge, either to read the value or to overcome the charge stored here. □ Asynchronous DRAM. □ Synchronous DRAM. □ Rambus Memory. 4.3.3. **ASYNCHRONOUS DRAM**

- DRAM's stores all the charge on a capacitor so cannot be kept indefinitely and must periodically be refreshed. That is why this memory is called dynamic.

- In DRAM to refresh the cell, we **read its contents** and **write it back.** The charge can be kept for several milliseconds.

SYNCHRONOUS DRAM

- Improves the access time significantly, to improve the interface to **processor's added clocks** with its and it is called as synchronous DRAM or SDRAM.

- The advantage of SDRAM is that the use of clock eliminates the time for the memory and processor to synchronize.

- SDRAM's transfer the bits in the burst. The fastest version is called **Double Data Rate(DDR).**

- **DDR-** It means data transfers on both the **rising and failing** edge of the clock, thereby getting twice as much bandwidth as we expect based on the clock rate and data width.

SRAM	**DRAM**
Information is stored in 1 bit cell called Flip Flop	Information is represented as charge across a capacitor
Information will be stored as long as the power is ON	Information may be lost, if power loss
No refreshing is needed	Refreshing is needed
Less packaging density	High packaging density
More complex hardware	Less complex hardware
More expensive	Less expensive

RAMBUS MEMORY

- Rambus Dynamic Random Access memory(RDRAM) in short Rambus memory is the fastest type of computer memory available. Typical SDRAM can be

transfer data at speed upto 133 MHZ, but standard RDRAM can transfer data over 1 GHZ.

- RDRAM is used for video memory on graphics accelerator cards, for cache memory

and for system only in high performance workstations and servers.

- An improvement to RDRAM called Direct Rambus(DRDRAM)allows for even faster data transfer rates.

4.3.6 ROM and FLASH MEMORY

- The read operation is same but write operation is different in non-volatile memory.
- A special writing process is needed to store information into this memory.
- There are different variations in non-volatile memory. They are

a) Read only memory (ROM).

b) Programmable ROM (PROM). c) ErasablePROM (EPROM).

d) Electrically EPROM (EEPROM). e) Flash memory.

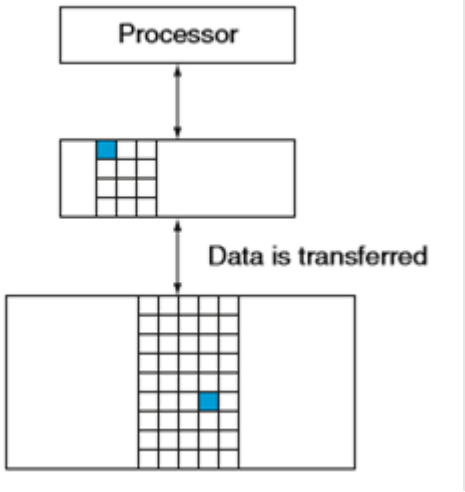
ROM (READ ONLY MEMORY)

- In ROM, permanent data and programs are stored. It is permanently in-built in the computer at the time of its production.
- Since, it is non-volatile memory it holds data even if the power is turned off.

PROM (PROGRAMMABLE READ ONLY MEMORY)

- Permanent data and programs are stored in ROM. But ROM designs allow the user to load programs and data. Such ROM designs are called Programmable ROM (PROM).
- PROM is once programmable and it is more flexiable and convenient than ROM.
- PROM are faster and less expensive.

	EPROM (ERASABLE PROGRAMMABLE READ ONLY MEMORY) ROM designs allow the stored data to be erased and new data to be loaded. Such an Erasable Re-Programmable ROM (EPROM). **EEPROM (ELECTRICALLY ERASABLE PROGRAMMABLE READ ONLY MEMORY)** It can be both programmed and erased electrically such chips are called as Electrically EPROM. They need not to be removed for erasing. It is possible to erase the chip contents selectively. **FLASH MEMORY** Flash memory is a type of EEPROM. But still there are some major difference between EEPROM and flash memory. **In EEPROM,** it is possible to read and write the contents of a single cell. **In FLASH MEMORY,** it is possible to read the contents of a single cell, but during writing operations the entire block of cells must be written. The previous contents of the clock of cells must be erased before writing.
2	Explain the basic operations of cache memory in detail with diagram. When a read request is received from a processor, a block of memory words containing the location are transformed to cache. Cache memory can store reasonable number of blocks, but this number is small compared to the total number of blocks in the main memory. The correspondence between the main memory blocks those in the cache are specified by mapping function.



Cache was the name chosen to represent the level of memory hierarchy between the processor and main memory.

- Cache is a **safe place for hiding or storing things** that we need to examine.

- When the CPU finds a requested data item in the cache it is called as **cache hit.**

- When the CPU does not find a requested data item in the cache it is called as **cache miss.**

- A fixed size collection of data containing the requested word called a **block.**

4.4.4 BLOCK REPLACEMENT

* When the cache is full and a memory word is not in the cache is referenced, the cache control hardware replaces a block from the cache to store the new block.
* The algorithm used for this process is called **Replacement algorithm.**
* The fraction of memory accesses found in a level of the memory hierarchy is called as **hit rate** or **hit ratio.**

WRITE OPERATION

* There are two ways for write operation
  - Write-through.
  - Write-back.
  - Write-buffer.
* **WRITE-THROUGH**
  - The cache location and the main memory location are updated simultaneously.
* **WRITE-BACK**
  - Only the cache location is updated a flag bit is used to specify the updation. It is called **dirty bit** or **modified bit.**
* **WRITE-BUFFER**
  - Stores the data while it is waiting to be written into the memory.
  - After writing data, processor can continue execution.
  - Write completes - write buffer is free.

Table 4.3 Both write back and write through have their advantages

WRITE-BACK	WRITE-THROUGH
1) Individually words can be written by the processor at the rate that the cache rather than the memory can accept them.	1) Misses are simpler and cheaper because they never require a block to be written back to the lower level.
2) Multiple words within a block require only one write to the lower level in the hierarchy.	**2) Write-through is easier** to implement than **write-back.**

3) When blocks are written back, the can make effective use of high bandwidth transfer since entire block is written.	

READ MISS

* When the addressed word in a read operation is **not in the cache,** it is called as **Read miss.**
* Alternatively, this word may be sent to the processor as soon as it is read from the memory. This approach is called **Load-through** or **early restart.**

WRITE MISS

* During write operation, word is not in the cache, it is called as **write miss.**

MISS RATE = 1 - HIT RATE

MISS RATE

* The fraction of memory accesses not found in the level of the memory hierarchy is called **miss rate.**

MISS PENALTY

* It is the **time to replace a block** in the upper level with the corresponding block from the lower level, plus the **time to deliver** this block to processor.

AMAT (AVERAGE MEMORY ACCESS TIME)

* It is the average time to access memory consider both **hits and misses** and the frequent of different accesses.

AMAT = TIME FOR A BIT + MISS RATE * MISS PENALTY

3	Explain the virtual memory address translation and TLB with necessary diagram.

4.6.1 ADDRESS TRANSLATION

- A virtual memory address translation method based on the concept of fixed-length.
- Each virtual address generated by the processor consists of two fields:
  i. Virtual page number (Higher order bits)
  ii. Offset (Lower order bit).

i. Virtual page number (Higher order bits)

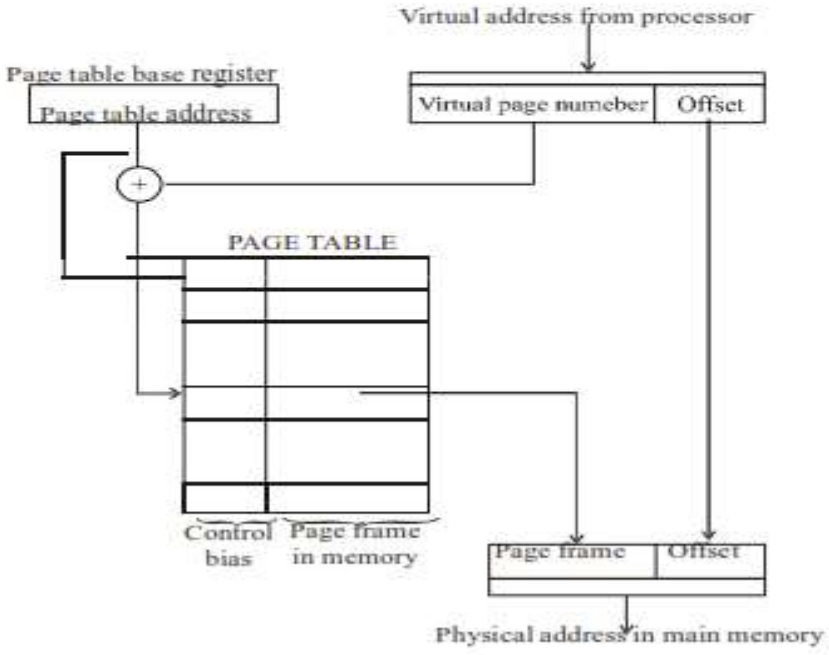


Fig 4.12 Virtual Memory Address translation

ii. Offset (Lower order bit).

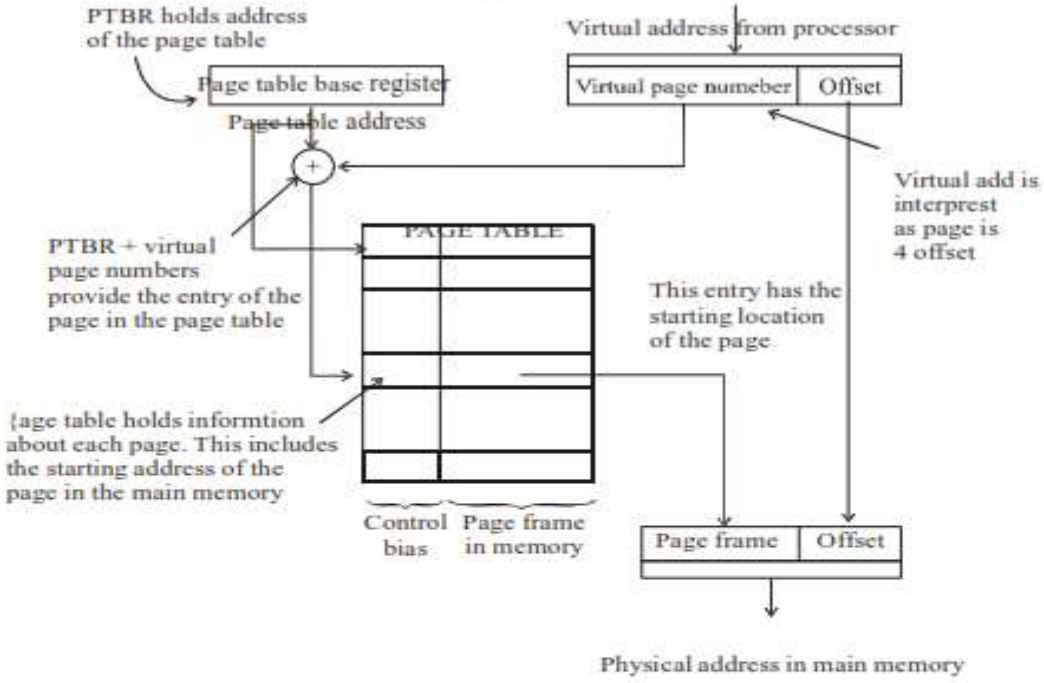


Fig 4.13. Virtual Memory Address translation

4.6.2 PAGE TABLE

Page table contains **information about the main memory location** of each page.

- This information includes the **main memory address** where the page is stored and **current status** of the page.
- An area in the main memory that holds one page is called **page frame.**
- Starting address of the page table is kept in a page table base register.
- The content of **page table base register is added with the virtual page number** to get the corresponding entry in the page table.
- Each entry in the page table also includes some control bits.
- Two important control bits are

1) **Valid bit** – indicates validity of the page.

2) **Modify bit** – indicates whether the page has been modified during residency in the memory.

The following steps are used in address translation:

- Virtual address generated and divided into two parts
  - **Virtual page number.**
  - **Offset.**
- Virtual page number in the address is added with the contents of page table base register to the entry in the page table.

4.6.3. TRANSLATION LOOKASIDE BUFFER (TLB)

* Virtual memory processor has to access page table is kept in the memory. To reduce the access time and degradation of performance, a small portion of page table is accommodated in MMU. This small cache (portion) is called as **Translation lookaside buffer (TLB).**

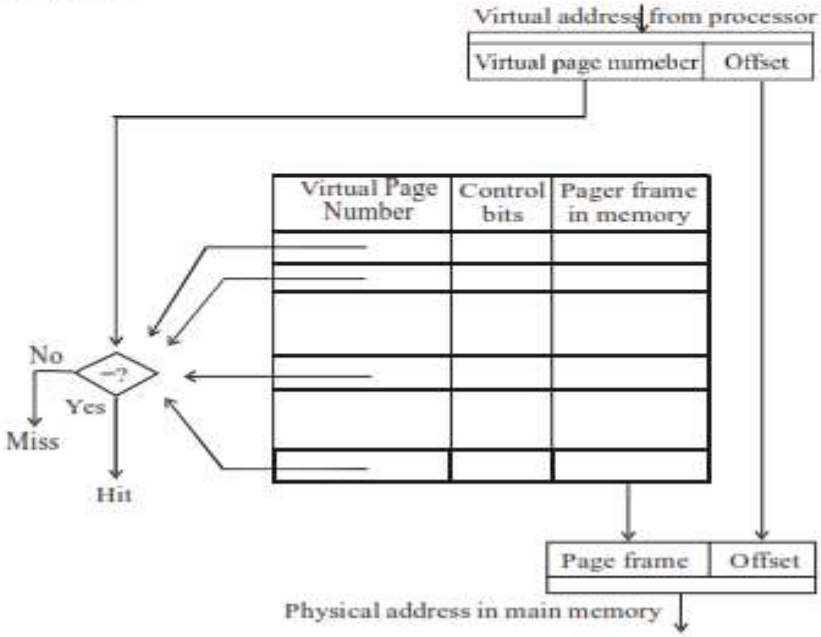


Fig 4.14. Translation lookaside buffer

* TLB contains virtual address of entry, in addition to the page table entry.

Following steps are used in address translation process with TLB:

- Processor generates the virtual addresses.
- MMU looks in the TLB for the referenced page.
- If the page table entry for this page **is found** in the TLB, the physical address is obtained immediately.
- If the page table entry is **is not found,** then the corresponding page entry is obtained from the page table in the main memory.
- Then the TLB is updated accordingly.
  - **Page fault**
  - **Page replacement.**
  - **Write operation.**

4	Explain in detail about parallel and serial bus architectures.

4.4 PARALLEL BUS ARCHITECTURES

Single bus architectures connect multiple processors with their own cache memory using shared bus. This is a simple architecture but it suffers from latency and bandwidth issues. This naturally led to deploying parallel or multiple bus architectures. Multiple bus multiprocessor systems use several parallel buses to interconnect multiple processors with multiple memory modules. The following are the connection schemes in multi bus architectures:

1. **Multiple-bus with full bus–memory connection (MBFBMC)**

This has all memory modules connected to all buses. The multiple-bus with single bus memory connection has each memory module connected to a specific bus. For N processors with M memory modules and B buses, the number of connections requires are: B(N+M) and the load on each bus will ne N+M.

2. **Multiple bus with partial bus–memory connection (MBPBMC)**

The multiple-bus with partial bus–memory connection, has each memory module connected to a subset of buses.

3. **Multiple bus with class-based memory connection (MBCBMC)**

The multiple-bus with class-based memory connection (MBCBMC), has memory modules grouped into classes whereby each class is connected to a specific subset of buses. A class is just an arbitrary collection of memory modules.

4. **Multiple bus with single bus memory connection (MBSBMC)**

Here, only single bus will be connected to single memory, but the processor can access all the buses. The numbers of connections:

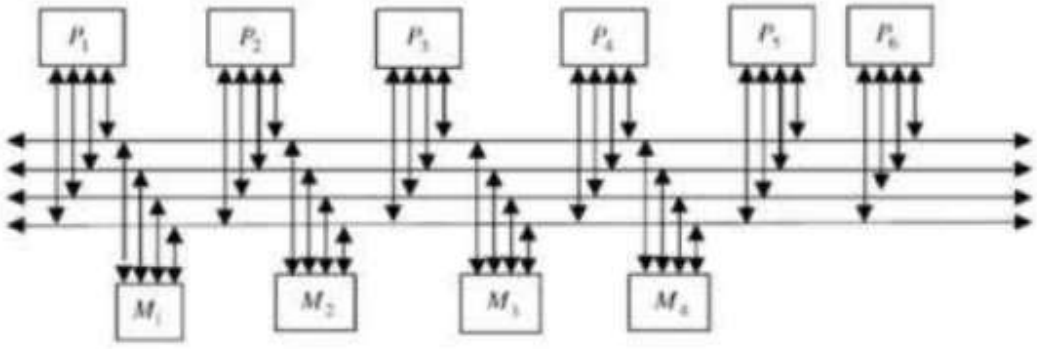


Fig 4.16 a) Multiple-bus with full bus–memory connection (MBFBMC)

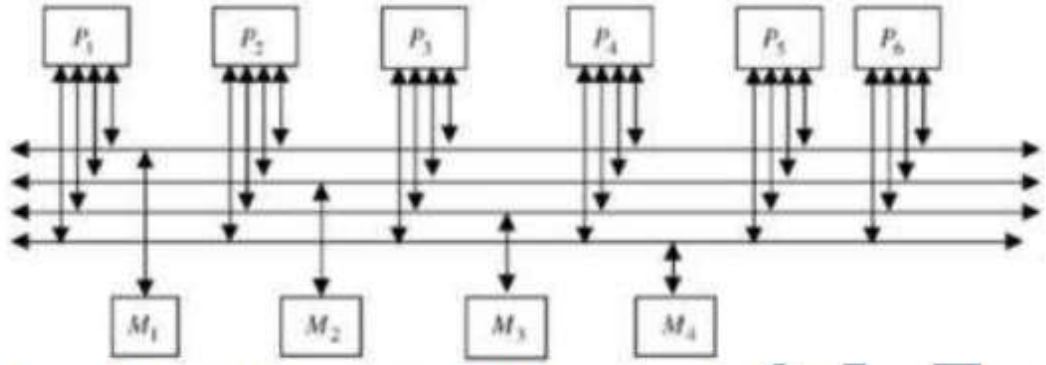


Fig 4.16 b) Multiple bus with single bus memory connection (MBSBMC)

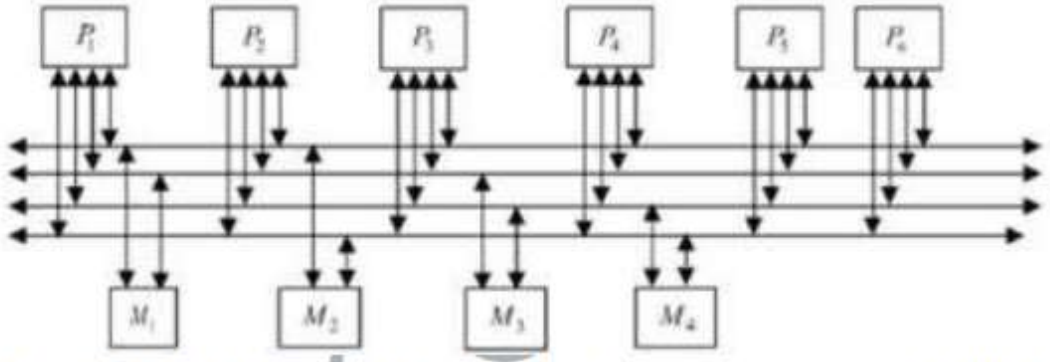


Fig 4.16 c) Multiple bus with partial bus–memory connection (MBPBMC)

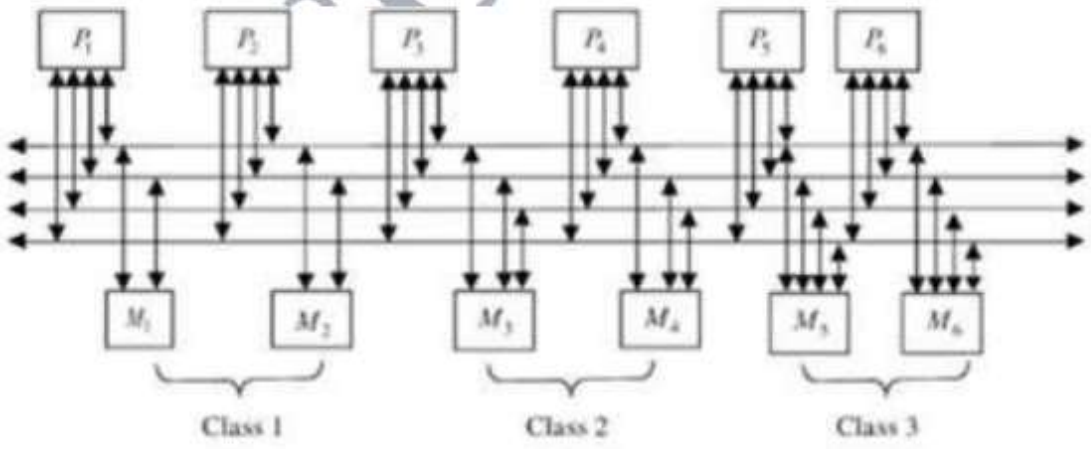


Fig 4.16 d) Multiple bus with class-based memory connection (MBCBMC)

4.6 SERIAL BUS ARCHITECTURES

The peripheral devices and external buffer that operate at relatively low frequencies communicate with the processor using serial bus. There are two popular serial buses: Serial Peripheral Interface (SPI) and Inter-Integrated Circuit (I^2C).

4.6.1 Serial Peripheral Interface (SPI)

Serial Peripheral Interface (SPI) is an interface bus designed by Motorola to send data between microcontrollers and small peripherals such as shift registers, sensors, and SD cards. It uses separate clock and data lines, along with a select line to choose the device.

- A standard SPI connection involves a master connected to slaves using the serial clock (SCK), Master Out Slave In (MOSI), Master In Slave Out (MISO), and Slave Select (SS) lines.
- The SCK, MOSI, and MISO signals can be shared by slaves while each slave has a unique SS line.
- The SPI interface defines no protocol for data exchange, limiting overhead and allowing for high speed data streaming.

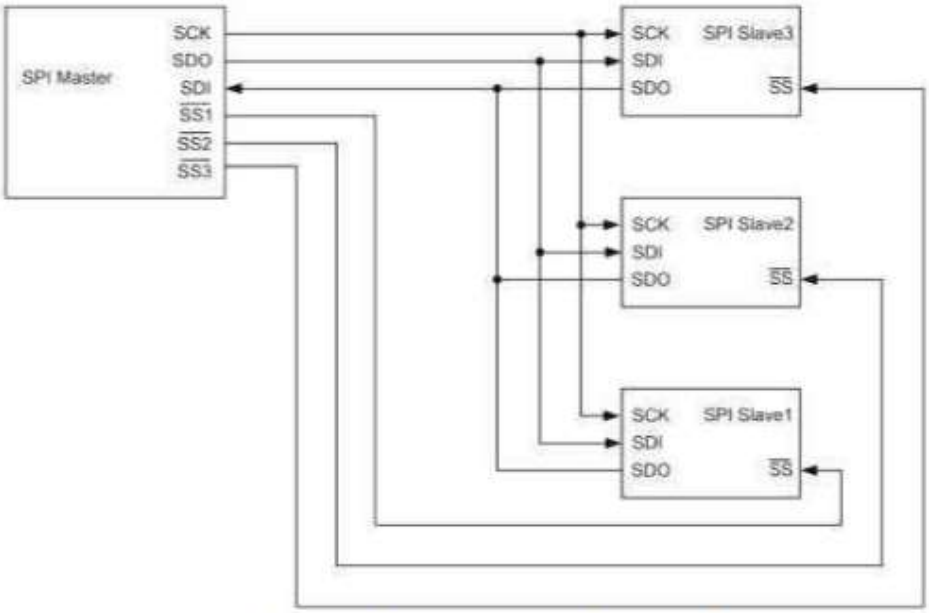


Fig 4.21: SPI master with three slaves

Mode	CPOL	CPHA
0	0	0
1	0	1
2	1	0
3	1	1

Fig 4.22: Modes in SPI

- In addition to the standard 4-wire configuration, the SPI interface has been extended to include a variety of IO standards including 3-wire for reduced pin count and dual or quad I/O for higher throughput.
- In 3-wire mode, MOSI and MISO lines are combined to a single bidirectional data line.
- Transactions are half-duplex to allow for bidirectional communication. Reducing the number of data lines and operating in half-duplex mode also decreases maximum possible throughput; many 3-wire devices have low performance requirements and are instead designed with low pin count in mind.

- Multi I/O variants such as dual I/O and quad I/O add additional data lines to the standard for increased throughput.
- Components that utilize multi I/O modes can rival the read speed of parallel devices while still offering reduced pin counts. This performance increase enables random access and direct program execution from flash memory (execute-in-place).

5	Explain in detail about internal communication methodologies

CPU of the computer system communicates with the memory and the I/O devices in order to transfer data between them. The method of communication of the CPU with memory and I/O devices is different. The CPU may communicate with the memory either directly or through the Cache memory. However, the communication between the CPU and I/O devices is usually implemented with the help of interface. There are three types of internal communications:

- Programmed I/O
- Interrupt driven I/O
- Direct Memory Access (DMA)

4.5.1 Programmed I/O

- Programmed I/O is implicated to data transfers that are initiated by a CPU, under driver software control to access Registers or Memory on a device.
- With programmed I/O, data are exchanged between the processor and the I/O module.
- The processor executes a program that gives it direct control of the I/O operation, including sensing device status, sending a read or write command, and transferring the data.
- When the processor issues a command to the I/O module, it must wait until the I/O operation is complete.
- If the processor is faster than the I/O module, this is wasteful of processor time. With interrupt-driven I/O, the processor issues I/O command, continues to execute other instructions, and is interrupted by the I/O module when the latter has completed its work.
- With both programmed and interrupt I/O, the processor is responsible for extracting data from main memory for output and storing data in main memory for input.
- The alternative is known as direct memory access. In this mode, the I/O module and main memory exchange data directly, without processor involvement.
- With programmed I/O, the I/O module will perform the requested action and then set the appropriate bits in the I/O status register.
- The I/O module takes no further action to alert the processor.
- When the processor is executing a program and encounters an instruction relating to I/O, it executes that instruction by issuing a command to the appropriate I/O module. In particular, it does not interrupt the processor.
- It is the responsibility of the processor periodically to check the status of the I/O module. Then if the device is ready for the transfer (read/write).
- The processor transfers the data to or from the I/O device as required. As the CPU is faster than the I/O module, the problem with programmed I/O is that the CPU has to wait a long time for the I/O module of concern to be ready for either reception or transmission of data.
- The CPU, while waiting, must repeatedly check the status of the I/O module, and this process is known as **Polling.**
- The level of the performance of the entire system is severely degraded.

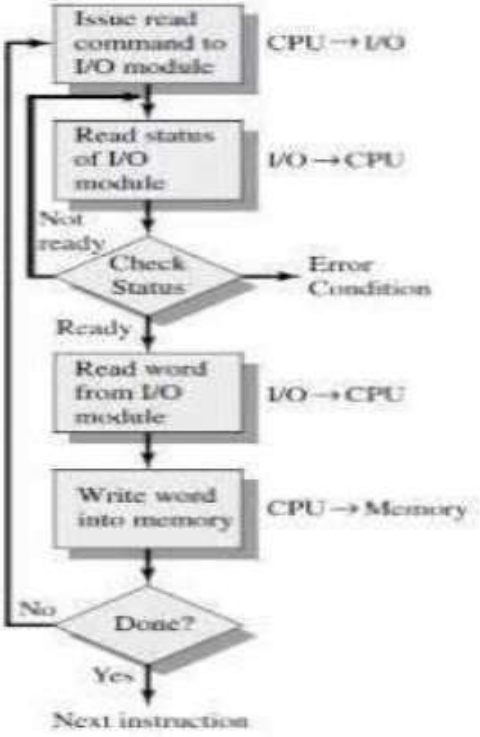


Fig 4.18: Workflow in programmed I/O

4.5.2 Interrupt Driven I/O

- The CPU issues commands to the I/O module then proceeds with its normal work until interrupted by I/O device on completion of its work.
- For input, the device interrupts the CPU when new data has arrived and is ready to be retrieved by the system processor. The actual actions to perform depend on whether the device uses I/O ports, memory mapping.
- For output, the device delivers an interrupt either when it is ready to accept new data or to acknowledge a successful data transfer. Memory-mapped and DMA-capable devices usually generate interrupts to tell the system they are done with the buffer.
- Although Interrupt relieves the CPU of having to wait for the devices, but it is still inefficient in data transfer of large amount because the CPU has to transfer the data word by word between I/O module and memory.
- Below are the basic operations of Interrupt:

1. CPU issues read command
2. I/O module gets data from peripheral whilst CPU does other work
3. I/O module interrupts CPU
4. CPU requests data
5. I/O module transfers data

4.5.3 Direct Memory Access (DMA)

- Direct Memory Access (DMA) means CPU grants I/O module authority to read from or write to memory without involvement.
- DMA module controls exchange of data between main memory and the I/O device.
- Because of DMA device can transfer data directly to and from memory, rather than using the CPU as an intermediary, and can thus relieve congestion on the bus.
- CPU is only involved at the beginning and end of the transfer and interrupted only after entire block has been transferred.

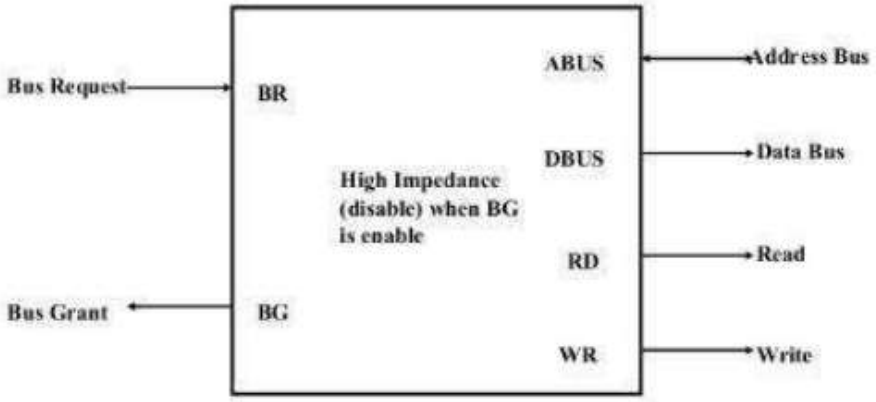


Fig 4.19: CPU bus signals for DMA transfer

- The CPU programs the DMA controller by setting its registers so it knows what to transfer where.
- It also issues a command to the disk controller telling it to read data from the disk into its internal buffer and verify the checksum.

	When valid data are in the disk controller's buffer, DMA can begin. The DMA controller initiates the transfer by issuing a read request over the bus to the disk controller. This read request looks like any other read request, and the disk controller does not know whether it came from the CPU or from a DMA controller. The memory address to write to is on the bus address lines, so when the disk controller fetches the next word from its internal buffer, it knows where to write it. The write to memory is another standard bus cycle. When the write is complete, the disk controller sends an acknowledgement signal to the DMA controller, also over the bus. The DMA controller then increments the memory address to use and decrements the byte count. If the byte count is still greater than 0, steps 2 through 4 are repeated until the count reaches 0. At that time, the DMA controller interrupts the CPU to let it know that the transfer is now complete. When the operating system starts up, it does not have to copy the disk block to memory; it is already there. The DMA controller requests the disk controller to transfer data from the disk controller's buffer to the main memory. In the first step, the CPU issues a command to the disk controller telling it to read data from the disk into its internal buffer.
6	Explain the design of a typical input and output interface. **Input Interface** Figure 7.10 shows a circuit that can be used to connect a keyboard to a processor. The registers in this circuit correspond to those given in Figure 3.3. Assume that interrupts are not used, so there is no need for a control register. There are only two registers: a data register, KBD_DATA, and a status register, KBD_STATUS. The latter contains the keyboard status flag, KIN. A typical keyboard consists of mechanical switches that are normally open. When a key is pressed, its switch closes and establishes a path for an electrical signal. This signal is detected by an encoder circuit that generates the ASCII code for the corresponding character. A difficulty with such mechanical pushbutton switches is that the contacts *bounce* when a key is pressed, resulting in the electrical connection being made then broken several times before the switch settles in the closed position. Although bouncing may last only one or two milliseconds, this is long enough for the computer to erroneously interpret a single pressing of a key as the key being pressed and released several times. The effect of bouncing can be eliminated using a simple debouncing circuit, which could be part of the keyboard hardware

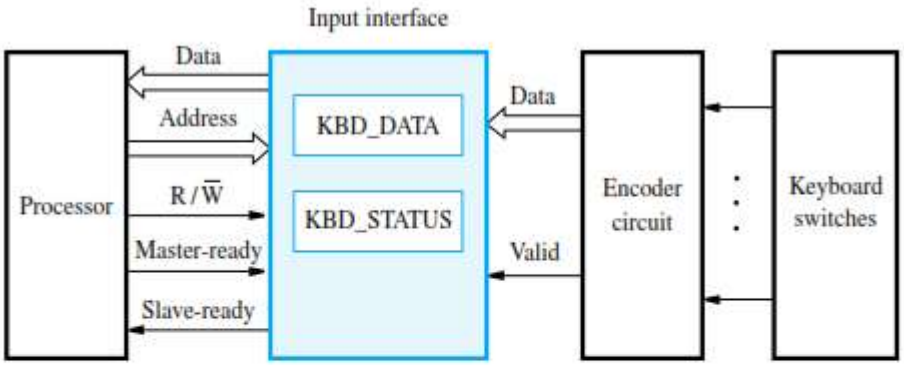


Figure 7.10 Keyboard to processor connection.

or may be incorporated in the encoder circuit. Alternatively, switch bouncing can be dealt with in software. The software detects that a key has been pressed when it observes that the keyboard status flag, KIN, has been set to 1. The I/O routine can then introduce sufficient delay before reading the contents of the input buffer, KBD_DATA, to ensure that bouncing has subsided. When debouncing is implemented in hardware, the I/O routine can read the input character as soon as it detects that KIN is equal to 1.

The output of the encoder in Figure 7.10 consists of one byte of data representing the encoded character and one control signal called Valid. When a key is pressed, the Valid signal changes from 0 to 1, causing the ASCII code of the corresponding character to be loaded into the KBD_DATA register and the status flag KIN to be set to 1. The status flag is cleared to 0 when the processor reads the contents of the KBD_DATA register. The interface circuit is shown connected to an asynchronous bus on which transfers are controlled by the handshake signals Master-ready and Slave-ready, as in Figure 7.6. The bus has one other control line, $R/\overline{W}$, which indicates a Read operation when equal to 1.

Output Interface

Let us now consider the output interface shown in Figure 7.13, which can be used to connect an output device such as a display. We have assumed that the display uses two handshake signals, New-data and Ready, in a manner similar to the handshake between the bus signals Master-ready and Slave-ready. When the display is ready to accept a character, it asserts its Ready signal, which causes the DOUT flag in the DISP_STATUS register to be set to 1. When the I/O routine checks DOUT and finds it equal to 1, it sends a character to DISP_DATA. This clears the DOUT flag to 0 and sets the New-data signal to 1. In response, the display returns Ready to 0 and accepts and displays the character in DISP_DATA. When it is ready to receive another character, it asserts Ready again, and the cycle repeats.

Figure 7.14 shows an implementation of this interface. Its operation is similar to that of the input interface of Figure 7.11, except that it responds to both Read and Write operations. A Write operation in which $A_2 = 0$ loads a byte of data into register DISP_DATA. A Read operation in which $A_2 = 1$ reads the contents of the status register DISP_STATUS. In this case, only the DOUT flag, which is bit b_2 of the status register, is sent by the interface. The remaining bits of DISP_STATUS are not used. The state of the status flag is determined

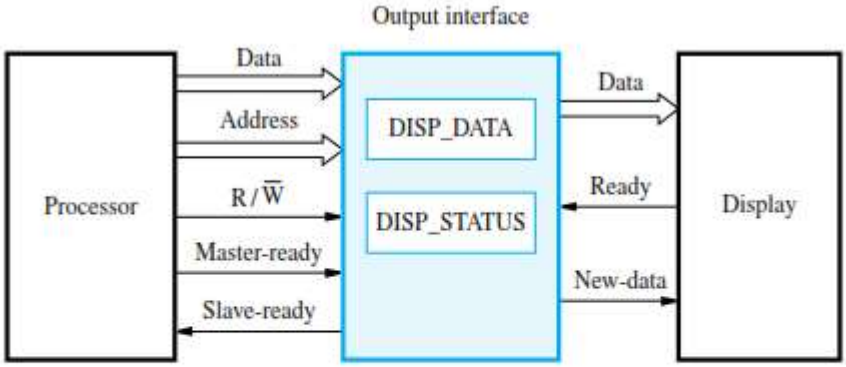


Figure 7.13 Display to processor connection.

by the handshake control circuit. A state diagram describing the behavior of this circuit is given as Example 7.4 at the end of the chapter.

7

Consider a system, which transfers 2MB file from memory to pen drive.
i) If memory is using handshaking protocol to send the file, depict clearly how the data transfer takes place in case of source initiated and destination-initiated data transfer. (7)

1. Source-Initiated Data Transfer (Push Model)

In this approach, the **memory (source)** initiates the transfer of data to the **pen drive (destination)** using a handshaking mechanism.

Steps:

1. **Memory (Source) Requests to Send:**
   - The memory places data on the bus.
   - It asserts a **Request (REQ)** signal to inform the pen drive that data is available.
2. **Pen Drive (Destination) Acknowledges:**
   - If the pen drive is ready to receive, it responds by asserting an **Acknowledge (ACK)** signal.
   - If the pen drive is busy, it waits until it is ready before sending the acknowledgment.
3. **Data Transfer Occurs:**
   - Once the acknowledgment is received, the memory transfers a portion of the file (e.g., one byte or a block of data).
   - The memory waits for another acknowledgment before sending the next portion.
4. **Repeat Until Transfer is Complete:**
   - Steps 1-3 are repeated until the entire **2MB file** is transferred.
   - The memory stops sending data after the last portion of the file is transmitted.

2. Destination-Initiated Data Transfer (Pull Model)

In this approach, the **pen drive (destination)** initiates the data transfer by requesting

data from the **memory (source)**.

Steps:

1. **Pen Drive Requests Data:**
   - The pen drive asserts a **Request (REQ)** signal to the memory, asking for data.
2. **Memory Acknowledges:**
   - If data is ready, the memory responds with an **Acknowledge (ACK)** signal and places data on the bus.
3. **Pen Drive Reads Data:**
   - The pen drive reads the data once the acknowledgment is received.
4. **Repeat Until Transfer is Complete:**
   - The pen drive continues requesting more data until the **entire 2MB file** is transferred.
   - Once all data is received, the request signal stops.

ii) When the file is being transferred there should be minimal intervention of the processor. Suggest a suitable technique for the above operation and explain it with proper justification and diagrams. (8)

When transferring a **2MB file** from memory to a pen drive, the steps are:

Step 1: Processor Initiates DMA Transfer

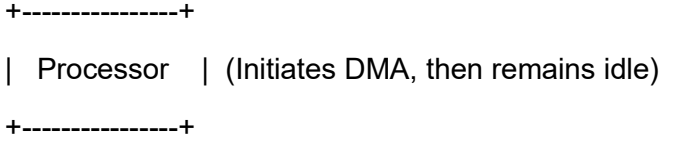
- The CPU configures the **DMA controller** by providing:
  - **Source Address** (Memory location of the file)
  - **Destination Address** (Pen drive address)
  - **Block Size** (2MB)
  - **Transfer Mode** (Burst, Block, or Cycle Stealing mode)
- The CPU then **issues a DMA request** and goes into a low-power or other processing state.

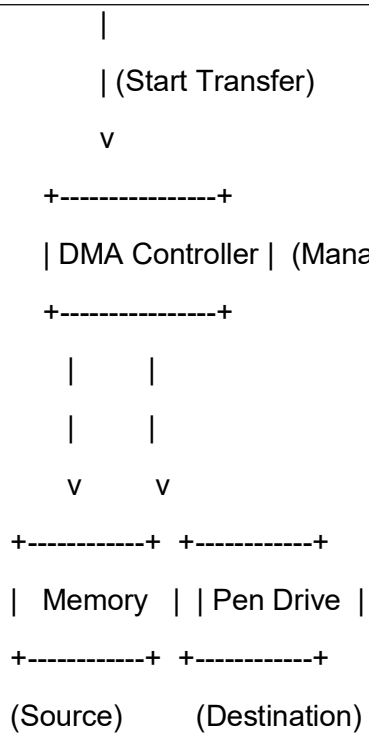
Step 2: DMA Handles Data Transfer

- The **DMA controller** takes control of the system bus and initiates the data transfer.
- The memory places data on the bus, and the pen drive receives it via the **USB interface**.

Step 3: Completion and Interrupt

- Once the **entire 2MB file** is transferred, the **DMA controller generates an interrupt** to notify the CPU that the transfer is complete.
- The CPU resumes control and continues normal operation.





Justification for Using DMA

1. **Reduces CPU Load** – The CPU is not involved in data movement, allowing it to perform other tasks or remain idle.
2. **Faster Data Transfer** – DMA can transfer large chunks of data efficiently compared to CPU-based transfers.
3. **Efficient Bus Utilization** – Since the CPU does not need to mediate each byte, the system bus is used more effectively.
4. **Supports Large File Transfers** – The DMA controller is well-suited for moving **2MB files** without continuous CPU intervention.

8	Consider a cache of 256 blocks in size, each block has 2^4 words. The main memory size is 2^{12} blocks, each block has 2^4 words. How many bits are required for each of the TAG, SET/BLOCK and WORD FIELDS for different mapping techniques? Wherever needed assume that there are 8 ways in each set. **1. Direct Mapped Cache** **Address Breakdown** A memory address consists of three parts: 1. **Word Offset**: Identifies the word within a block. 2. **Index (Set/Block Field)**: Identifies the cache block where a memory block will be placed. 3. **Tag**: Uniquely identifies a memory block in case of conflicts. **Bit Calculation** 1. **Word Field (Offset):** • Each block contains **24 words**, so we need $\log_2 24 = 5$ **bits** to select a word. 2. **Block Field (Index):** • The cache has **256 blocks**, so we need $\log_2 256 = 8$ **bits** to index them. 3. **Tag Field:** • Main memory has 2^{12} **blocks**, so **total address bits required for block identification** = $\log_2(2^{12}) = 12$ bits. • Since **8 bits** are used for the block field, the remaining **(12 - 8) = 4** bits are used for the **Tag**. 2. **Fully Associative Cache** In a **fully associative** cache, a memory block can be placed in **any** cache block. **Bit Calculation** 1. **Word Field (Offset):** ○ Same as before, **5 bits**. 2. **Block Field (Index):** ○ Since any memory block can be placed anywhere, there is **no set index**. ○ This means the block field is not needed. 3. **Tag Field:** ○ The entire remaining address is used as the tag. ○ Since the total number of memory blocks is **2^12**, we need **12 bits** for the tag.

3. 8-Way Set Associative Cache

In an **8-way set associative** cache, the **256 blocks** are divided into **sets**, each containing **8 blocks**.

Bit Calculation

1. **Word Field (Offset):**
   - **5 bits** (same as before).
2. **Set Field (Index):**
   - Since there are **256 blocks** and **each set has 8 blocks**, the number of sets is:

$$\frac{256}{8} = 32 \text{ sets}$$

   - We need $\log_2 32 = 5$ **bits** for the set index.
3. **Tag Field:**
   - The total number of memory blocks is 2^{12} = **4096 blocks**.
   - Since we already use **5 bits** for the set index, the remaining **(12 - 5) = 7 bits** are used for the tag.