UNIT – V

Memory system design: Semiconductor memory technologies, memory organization.
Memory organization: Memory interleaving, concept of hierarchical memory organization, Cache memory, cache size vs. block size, mapping functions, Replacement algorithms, write policies.

Semiconductor Memory Technologies:

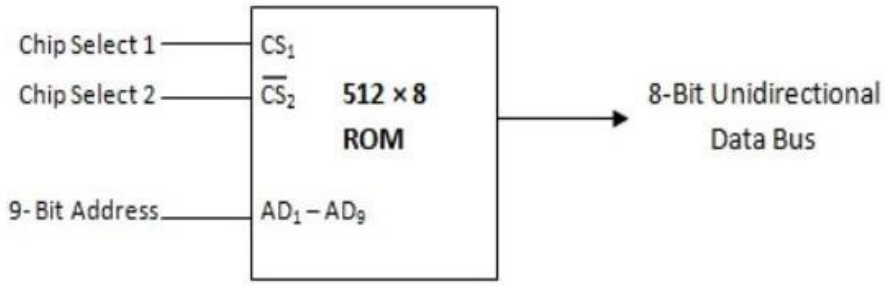
Semiconductor random-access memories (RAMs) are available in a wide range of speeds. Their cycle times range from 100 ns to less than 10 ns. Semiconductor memory is used in any electronics assembly that uses computer processing technology. The use of semiconductor memory has grown, and the size of these memory cards has increased as the need for larger and larger amounts of storage is needed.

There are two main types or categories that can be used for semiconductor technology.

RAM - Random Access Memory: As the names suggest, the RAM or random access memory is a form of semiconductor memory technology that is used for reading and writing data in any order - in other words as it is required by the processor. It is used for such applications as the computer or processor memory where variables and other stored and are required on a random basis. Data is stored and read many times to and from this type of memory.

Block Diagram Representing 128 x 8 RAM
(Random Access Memory)

ROM - Read Only Memory: A ROM is a form of semiconductor memory technology used where the data is written once and then not changed. In view of this it is used where data needs to be stored permanently, even when the power is removed - many memory technologies lose the data once the power is removed. As a result, this type of semiconductor memory technology is widely used for storing programs and data that must survive when a computer or processor is powered down. For example the BIOS of a computer will be stored in ROM. As the name implies, data cannot be easily written to ROM. Depending on the technology used in the ROM, writing the data into the ROM initially may require special hardware. Although it is often possible to change the data, this gain requires special hardware to erase the data ready for new data to be written in.


Chip Select 1
Chip Select 2
9- Bit Address
CS_1
$\overline{CS_2}$
$AD_1 - AD_9$
512 × 8
ROM
8-Bit Unidirectional
Data Bus

The different memory types or memory technologies are detailed below:

DRAM: Dynamic RAM is a form of random access memory. DRAM uses a capacitor to store each bit of data, and the level of charge on each capacitor determines whether that bit is a logical 1 or 0. However these capacitors do not hold their charge indefinitely, and therefore the data needs to be refreshed periodically. As a result of this dynamic refreshing it gains its name of being a dynamic RAM. DRAM is the form of semiconductor memory that is often used in equipment including personal computers and workstations where it forms the main RAM for the computer.

EEPROM: This is an Electrically Erasable Programmable Read Only Memory. Data can be written to it and it can be erased using an electrical voltage. This is typically applied to an erase pin on the chip. Like other types of PROM, EEPROM retains the contents of the memory even when the power is turned off. Also like other types of ROM, EEPROM is not as fast as RAM.

EPROM: This is an Erasable Programmable Read Only Memory. This form of semiconductor memory can be programmed and then erased at a later time. This is normally achieved by exposing the silicon to ultraviolet light. To enable this to happen there is a circular window in the package of the EPROM to enable the light to reach the silicon of the chip. When the PROM is in use, this window is normally covered by a label, especially when the data may need to be preserved for an extended period. The PROM stores its data as a charge on a capacitor. There is a charge storage capacitor for each cell and this can be read repeatedly as required. However it is found that after many years the charge may leak away and the data may be lost. Nevertheless, this type of semiconductor memory used to be widely used in applications where a form of ROM was required, but where the data needed to be changed periodically, as in a development environment, or where quantities were low.

FLASH MEMORY: Flash memory may be considered as a development of EEPROM technology. Data can be written to it and it can be erased, although only in blocks, but data can be read on an individual cell basis. To erase and re-programme areas of the chip, programming voltages at levels that are available within electronic equipment are used. It is also non-volatile, and this makes it particularly useful. As a result Flash memory is widely used in many applications including memory cards for digital cameras, mobile phones, computer memory sticks and many other applications.

F-RAM: Ferroelectric RAM is a random-access memory technology that has many similarities to the standard DRAM technology. The major difference is that it incorporates a ferroelectric layer instead of the more usual dielectric layer and this provides its non-volatile capability. As it offers a non-volatile capability, F-RAM is a direct competitor to Flash.

MRAM: This is Magneto-resistive RAM, or Magnetic RAM. It is a non-volatile RAM memory technology that uses magnetic charges to store data instead of electric charges. Unlike technologies including DRAM, which require a constant flow of electricity to maintain the integrity of the data, MRAM retains data even when the power is removed. An additional advantage is that it only requires low power for active operation. As a result this technology could become a major player in the electronics industry now that production processes have been developed to enable it to be produced.

P-RAM / PCM: This type of semiconductor memory is known as Phase change Random Access Memory, P-RAM or just Phase Change memory, PCM. It is based around a phenomenon where a form of chalcogenide glass changes is state or phase between an amorphous state (high resistance) and a polycrystalline state (low resistance). It is possible to detect the state of an individual cell and hence use this for data storage. Currently this type of memory has not been widely commercialized, but it is expected to be a competitor for flash memory.

PROM: This stands for Programmable Read Only Memory. It is a semiconductor memory which can only have data written to it once - the data written to it is permanent. These memories are bought in a blank format and they are programmed using a special PROM programmer. Typically a PROM will consist of an array of fuseable links some of which are "blown" during the programming process to provide the required data pattern.

SDRAM: Synchronous DRAM. This form of semiconductor memory can run at faster speeds than conventional DRAM. It is synchronised to the clock of the processor and is capable of keeping two sets of memory addresses open simultaneously. By transferring data alternately from one set of addresses, and then the other, SDRAM cuts down on the delays associated with non-synchronous RAM, which must close one address bank before opening the next.

SRAM: Static Random Access Memory. This form of semiconductor memory gains its name from the fact that, unlike DRAM, the data does not need to be refreshed dynamically. It is able to support faster read and write times than DRAM (typically 10 ns against 60 ns for DRAM), and in addition its cycle time is much shorter because it does not need to pause between accesses. However it consumes more power, is less dense and more expensive than DRAM. As a result of this it is normally used for caches, while DRAM is used as the main semiconductor memory technology.

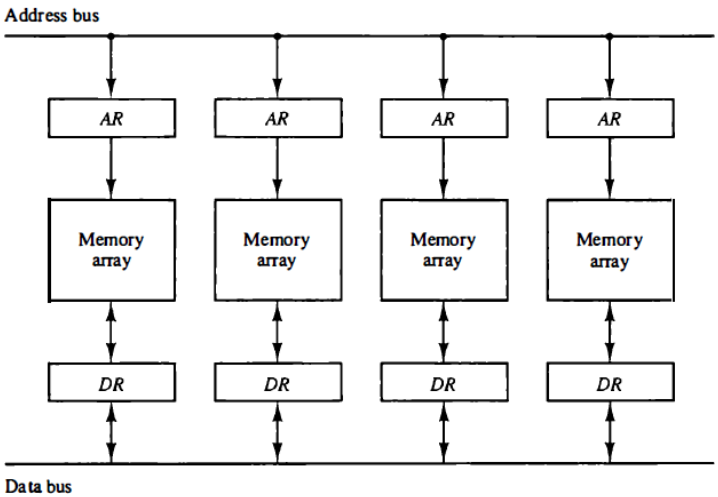
MEMORY ORGANIZATION

Memory Interleaving:

Pipeline and vector processors often require simultaneous access to memory from two or more sources. An instruction pipeline may require the fetching of an instruction and an operand at the same time from two different segments.

Similarly, an arithmetic pipeline usually requires two or more operands to enter the pipeline at the same time. Instead of using two memory buses for simultaneous access, the memory can be partitioned into a number of modules connected to a common memory address and data buses. A memory module is a memory array together with its own address and data registers. Figure 9-13 shows a memory unit with four modules. Each memory array has its own address register AR and data register DR.

Figure 9-13 Multiple module memory organization.



The address registers receive information from a common address bus and the data registers communicate with a bidirectional data bus. The two least significant bits of the address can be used to distinguish between the four modules. The modular system permits one module to initiate a memory access while other modules are in the process of reading or writing a word and each module can honor a memory request independent of the state of the other modules.

The advantage of a modular memory is that it allows the use of a technique called interleaving. In an interleaved memory, different sets of addresses are assigned to different memory modules. For example, in a two-module memory system, the even addresses may be in one module and the odd addresses in the other.

Concept of Hierarchical Memory Organization

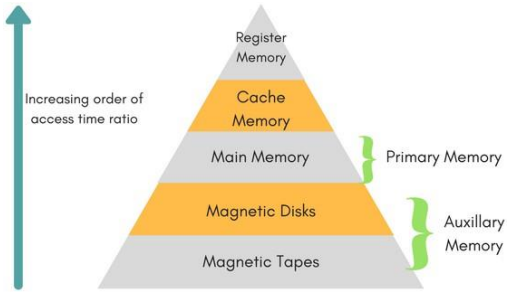
This Memory Hierarchy Design is divided into 2 main types:

External Memory or Secondary Memory

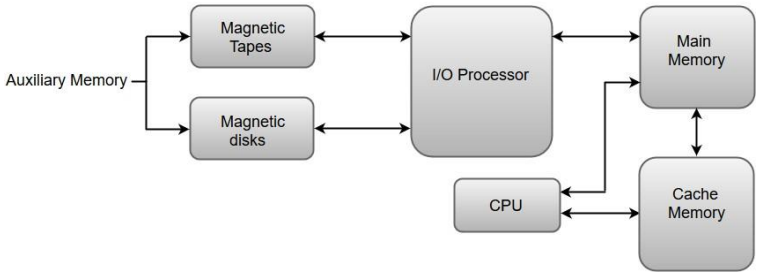
Comprising of Magnetic Disk, Optical Disk, Magnetic Tape i.e. peripheral storage devices which are accessible by the processor via I/O Module.

Internal Memory or Primary Memory

Comprising of Main Memory, Cache Memory & CPU registers. This is directly accessible by the processor.



Memory Hierarchy in a Computer System:



Characteristics of Memory Hierarchy

Capacity:

It is the global volume of information the memory can store. As we move from top to bottom in the Hierarchy, the capacity increases.

Access Time:

It is the time interval between the read/write request and the availability of the data. As we move from top to bottom in the Hierarchy, the access time increases.

Performance:

Earlier when the computer system was designed without Memory Hierarchy design, the speed gap increases between the CPU registers and Main Memory due to large difference in access time. This results in lower performance of the system and thus, enhancement was required. This enhancement was made in the form of Memory Hierarchy Design because of which the performance of the system increases. One of the most significant ways to increase system performance is minimizing how far down the memory hierarchy one has to go to manipulate data.

Cost per bit:

As we move from bottom to top in the Hierarchy, the cost per bit increases i.e. Internal Memory is costlier than External Memory.

Cache Memories:

The cache is a small and very fast memory, interposed between the processor and the main memory. Its purpose is to make the main memory appear to the processor to be much faster than it actually is. The effectiveness of this approach is based on a property of computer programs called locality of reference.

Analysis of programs shows that most of their execution time is spent in routines in which many instructions are executed repeatedly. These instructions may constitute a simple loop, nested loops, or a few procedures that repeatedly call each other.

The cache memory can store a reasonable number of blocks at any given time, but this number is small compared to the total number of blocks in the main memory. The correspondence between the main memory blocks and those in the cache is specified by a mapping function.

When the cache is full and a memory word (instruction or data) that is not in the cache is referenced, the cache control hardware must decide which block should be removed to create space for the new block that contains the referenced word. The collection of rules for making this decision constitutes the cache‟s *replacement algorithm*.

Cache Hits

The processor does not need to know explicitly about the existence of the cache. It simply issues Read andWrite requests using addresses that refer to locations in the memory. The cache control circuitry determines whether the requested word currently exists in the cache.
If it does, the Read orWrite operation is performed on the appropriate cache location. In this case, a *read* or *write hit* is said to have occurred.

Cache Misses

A Read operation for a word that is not in the cache constitutes a *Read miss*. It causes the block of words containing the requested word to be copied from the main memory into the cache.

Cache Mapping:

There are three different types of mapping used for the purpose of cache memory which are as follows: Direct mapping, Associative mapping, and Set-Associative mapping. These are explained as following below.

Direct mapping

The simplest way to determine cache locations in which to store memory blocks is the *direct-mapping* technique. In this technique, block *j* of the main memory maps onto block *j* modulo 128 of the cache, as depicted in Figure 8.16. Thus, whenever one of the main memory blocks 0, 128, 256, . . . is loaded into the cache, it is stored in cache block 0. Blocks 1, 129, 257, . . . are stored in cache block 1, and so on. Since more than one memory block is mapped onto a given cache block position, contention may arise for that position even when the cache is not full.

For example, instructions of a program may start in block 1 and continue in block 129, possibly after a branch. As this program is executed, both of these blocks must be transferred to the block-1 position in the cache. Contention is resolved by allowing the new block to overwrite the currently resident block.

With direct mapping, the replacement algorithm is trivial. Placement of a block in the cache is determined by its memory address. The memory address can be divided into three fields, as shown in Figure 8.16. The low-order 4 bits select one of 16 words in a block.

When a new block enters the cache, the 7-bit cache block field determines the cache position in which this block must be stored. If they match, then the desired word is in that block of the cache. If there is no match, then the block containing the required word must first be read from the main memory and loaded into the cache.

The direct-mapping technique is easy to implement, but it is not very flexible.

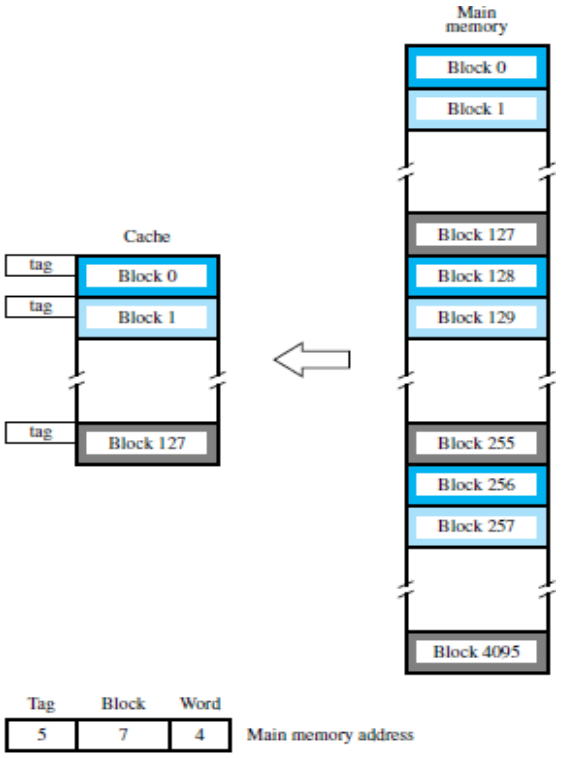


Figure 8.16 Direct-mapped cache.

Associative Mapping

In Associative mapping method, in which a main memory block can be placed into any cache block position. In this case, 12 tag bits are required to identify a memory block when it is resident in the cache. The tag bits of an address received from the processor are compared to the tag bits of each block of the cache to see if the desired block is present. This is called the *associative-mapping* technique.

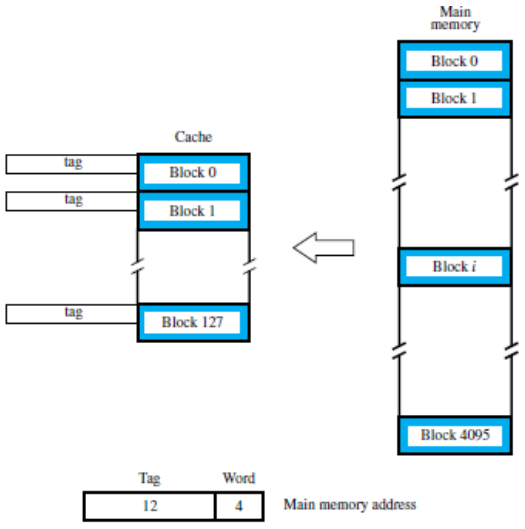


Figure 8.17 Associative-mapped cache.

It gives complete freedom in choosing the cache location in which to place the memory block, resulting in a more efficient use of the space in the cache. When a new block is brought into the cache, it replaces (ejects) an existing block only if the cache is full. In this case, we need an algorithm to select the block to be replaced.

To avoid a long delay, the tags must be searched in parallel. A search of this kind is called an *associative search*.

Set-Associative Mapping

Another approach is to use a combination of the direct- and associative-mapping techniques. The blocks of the cache are grouped into sets, and the mapping allows a block of the main memory to reside in any block of a specific set. Hence, the contention problem of the direct method is eased by having a few choices for block placement.

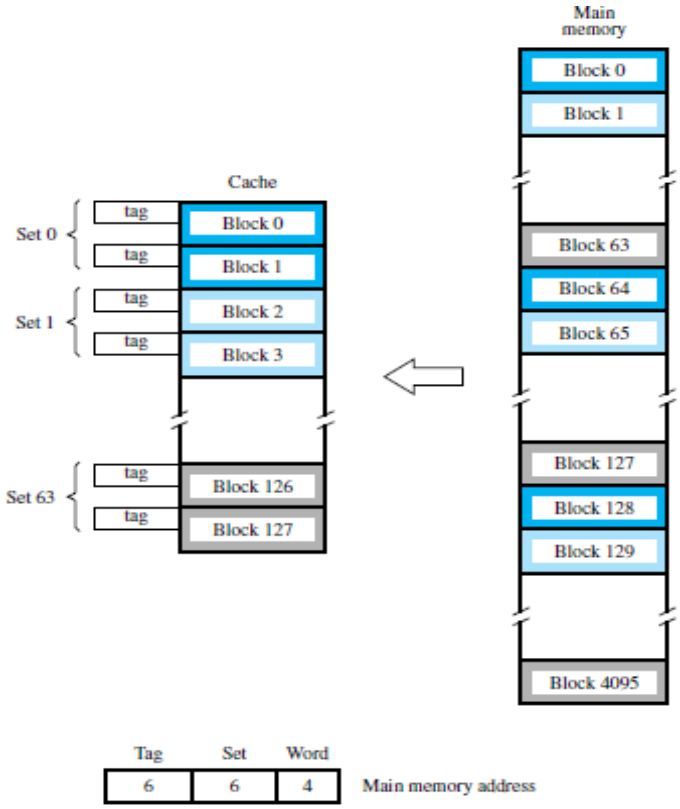


Figure 8.18 Set-associative-mapped cache with two blocks per set.

At the same time, the hardware cost is reduced by decreasing the size of the associative search. An example of this *set-associative-mapping* technique is shown in Figure 8.18 for a cache with two blocks per set. In this case, memory blocks 0, 64, 128, . . . , 4032 map into cache set 0, and they can occupy either of the two block positions within this set.

Having 64 sets means that the 6-bit set field of the address determines which set of the cache might contain the desired block. The tag field of the address must then be associatively compared to the tags of the two blocks of the set to check if the desired block is present. This two-way associative search is simple to implement.

The number of blocks per set is a parameter that can be selected to suit the requirements of a particular computer. For the main memory and cache sizes in Figure 8.18, four blocks per set can be accommodated by a 5-bit set field, eight blocks per set by a 4-bit set field, and so on. The extreme condition of 128 blocks per set requires no set bits and corresponds to the fully-associative technique, with 12 tag bits. The other extreme of one block per set is the direct-mapping.

Replacement Algorithms

In a direct-mapped cache, the position of each block is predetermined by its address; hence, the replacement strategy is trivial. In associative and set-associative caches there exists some flexibility. When a new block is to be brought into the cache and all the positions that it may occupy are full, the cache controller must decide which of the old blocks to overwrite.

This is an important issue, because the decision can be a strong determining factor in system performance. In general, the objective is to keep blocks in the cache that are likely to be referenced in the near future. But, it is not easy to determine which blocks are about to be referenced.

The property of locality of reference in programs gives a clue to a reasonable strategy. Because program execution usually stays in localized areas for reasonable periods of time, there is a high probability that the blocks that have been referenced recently will be referenced again soon. Therefore, when a block is to be overwritten, it is sensible to overwrite the one that has gone the longest time without being referenced. This block is called the *least recently used* (LRU) block, and the technique is called the *LRU replacement algorithm*.

The LRU algorithm has been used extensively. Although it performs well for many access patterns, it can lead to poor performance in some cases.

Write Policies

The write operation is proceeding in 2 ways.

- Write-through protocol
- Write-back protocol

Write-through protocol:

Here the cache location and the main memory locations are updated simultaneously.

Write-back protocol:

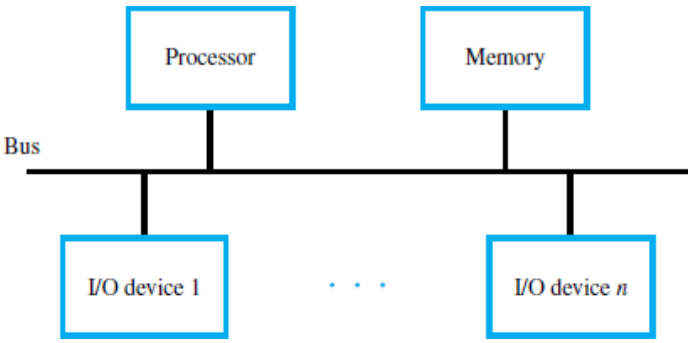
- This technique is to update only the cache location and to mark it as with associated flag bit called dirty/modified bit.
- The word in the main memory will be updated later, when the block containing this marked word is to be removed from the cache to make room for a new block.
- To overcome the read miss Load –through / Early restart protocol is used.

Peripheral devices and their characteristics: Input-output subsystems, I/O device interface, I/O transfers – program controlled, interrupt driven and DMA, privileged and non-privileged instructions, software interrupts and exceptions. Programs and processes – role of interrupts in process state transitions, I/O device interfaces – SCII, USB

Input-output subsystems

The Input/output organization of computer depends upon the size of computer and the peripherals connected to it. The I/O Subsystem of the computer provides an efficient mode of communication between the central system and the outside environment**.**

The most common input output devices are: Monitor, Keyboard, Mouse, Printer, Magnetic tapes 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 communication link is to resolve the differences that exist between the central computer and each peripheral.



The Major Differences are:-

- Peripherals are electromechnical and electromagnetic devices and CPU and memory are electronic devices. Therefore, a conversion of signal values may be needed.
- The data transfer rate of peripherals is usually slower than the transfer rate of CPU and consequently, a synchronization mechanism may be needed.
- Data codes and formats in the peripherals differ from the word format in the CPU and memory.
- The operating modes of peripherals are different from each other and 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 supervises and synchronizes all input and out transfers. These components are called Interface Units because they interface between the processor bus and the peripheral devices.

I/O device interface

The I/O Bus consists of data lines, address lines and control lines. The I/O bus from the processor is attached to all peripherals interface. To communicate with a particular device, the processor places a device address on address lines. Each Interface decodes the address and control received from the I/O bus, interprets them for peripherals and provides signals for the peripheral controller. It is also synchronizes the data flow and supervises the transfer between peripheral and processor. Each peripheral has its own controller.

For example, the printer controller controls the paper motion, the print timing. The control lines are referred as I/O command. The commands are as following:
Control command- A control command is issued to activate the peripheral and to inform it what to do.
Status command- A status command is used to test various status conditions in the interface and the peripheral.
Data Output command- A data output command causes the interface to respond by transferring data from the bus into one of its registers.
Data Input command- The data input command is the opposite of the data output.

In this case the interface receives on item of data from the peripheral and places it in its buffer register.
I/O Versus Memory Bus

To communicate with I/O, the processor must communicate with the memory unit. Like the I/O bus, the memory bus contains data, address and read/write control lines. There are 3 ways that computer buses can be used to communicate with memory and I/O:
1. Use two Separate buses, one for memory and other for I/O.
2. Use one common bus for both memory and I/O but separate control lines for each.
3. Use one common bus for memory and I/O with common control lines.

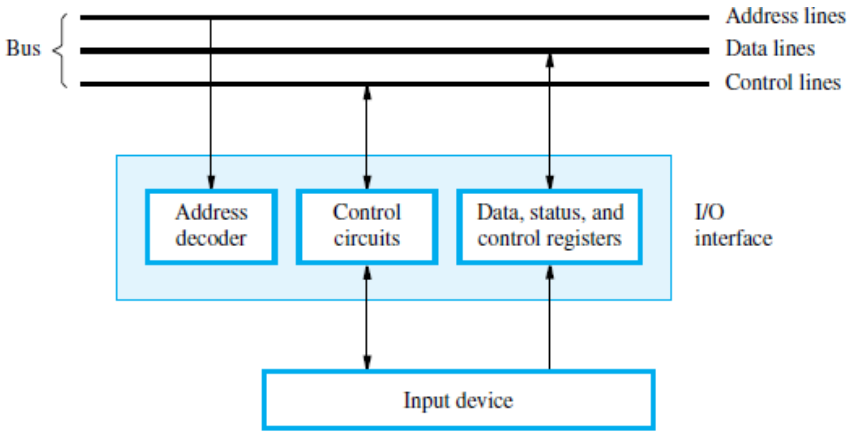


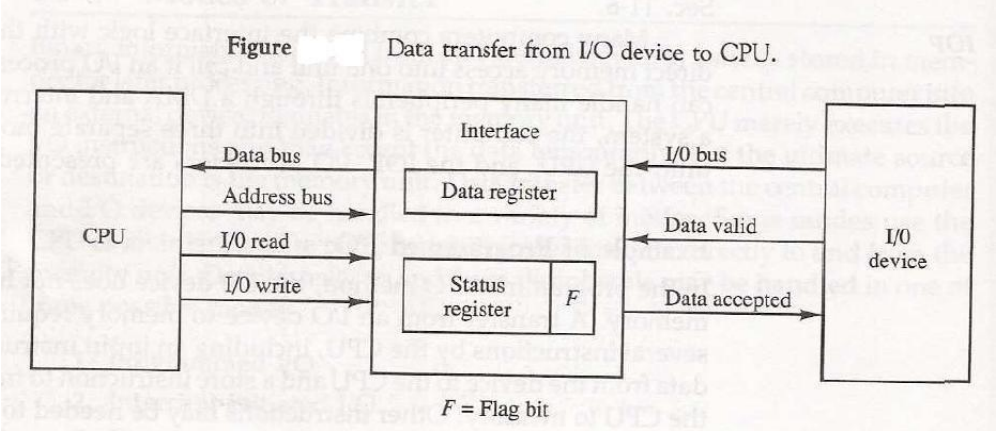
Figure 7.2 I/O interface for an input device.

Programmed I/O Mode:

In this mode of data transfer the operations are the results in I/O instructions which is a part of computer program. Each data transfer is initiated by a instruction in the program. Normally the transfer is from a CPU register to peripheral device or vice-versa. Once the data is initiated the CPU starts monitoring the interface to see when next transfer can made. The instructions of the program keep close tabs on everything that takes place in the interface unit and the I/O devices.

The transfer of data requires three instructions:

- Read the status register.
- Check the status of the flag bit and branch to step 1 if not set or to step 3 if set.
- Read the data register.



In this technique CPU is responsible for executing data from the memory for output and storing data in memory for executing of Programmed I/O as shown in Fig.

Drawback of the Programmed I/O:

The main drawback of the Program Initiated I/O was that the CPU has to monitor the units all the times when the program is executing. Thus the CPU stays in a program loop until the I/O unit indicates that it is ready for data transfer. This is a time consuming process and the CPU time is wasted a lot in keeping an eye to the executing of program.

Interrupt-Initiated I/O:

In this method an interrupt facility an interrupt command is used to inform the device about the start and end of transfer. In the meantime the CPU executes other program. When the interface determines that the device is ready for data transfer it generates an Interrupt Request and sends it to the computer.

When the CPU receives such an signal, it temporarily stops the execution of the program and branches to a service program to process the I/O transfer and after completing it returns back to task, what it was originally performing.

In this type of IO, computer does not check the flag. It continues to perform its task. Whenever any device wants the attention, it sends the interrupt signal to the CPU.CPU then deviates from what it was doing, store the return address from PC and branch to the address of the subroutine.

There are two ways of choosing the branch address:

Vectored Interrupt: In vectored interrupt the source that interrupts the CPU provides the branch information. This information is called interrupt vectored.

Non-vectored Interrupt: In non-vectored interrupt, the branch address is assigned to the fixed address in the memory.

Direct Memory Access (DMA):

In the Direct Memory Access (DMA) the interface transfer the data into and out of the memory unit through the memory bus. The transfer of data between a fast storage device such as magnetic disk and memory is often limited by the speed of the CPU. Removing the CPU from the path and letting the peripheral device manage the memory buses directly would improve the speed of transfer. This transfer technique is called Direct Memory Access (DMA).

During the DMA transfer, the CPU is idle and has no control of the memory buses. A DMA Controller takes over the buses to manage the transfer directly between the I/O device and memory.

The CPU may be placed in an idle state in a variety of ways. One common method extensively used in microprocessor is to disable the buses through special control signals such as:

- Bus Request (BR)
- Bus Grant (BG)

These two control signals in the CPU that facilitates the DMA transfer. The Bus Request (BR) input

is used by the DMA controller to request the CPU. When this input is active, the CPU terminates the execution of the current instruction and places the address bus, data bus and read write lines into a high Impedance state. High Impedance state means that the output is disconnected.

The CPU activates the Bus Grant (BG) output to inform the external DMA that the Bus Request (BR) can now take control of the buses to conduct memory transfer without processor.
When the DMA terminates the transfer, it disables the Bus Request (BR) line. The CPU disables the Bus Grant (BG), takes control of the buses and return to its normal operation.

The transfer can be made in several ways that are:

- DMA Burst
- Cycle Stealing

DMA Burst: In DMA Burst transfer, a block sequence consisting of a number of memory words is transferred in continuous burst while the DMA controller is master of the memory buses.
Cycle Stealing: Cycle stealing allows the DMA controller to transfer one data word at a time, after which it must returns control of the buses to the CPU.

DMA Controller:
The DMA controller needs the usual circuits of an interface to communicate with the CPU and I/O device. The DMA controller has three registers:

- Address Register
- Word Count Register
- Control Register

Address Register: Address Register contains an address to specify the desired location in memory.
Word Count Register: WC holds the number of words to be transferred. The register is incre/decre by one after each word transfer and internally tested for zero.
Control Register: Control Register specifies the mode of transfer

The unit communicates with the CPU via the data bus and control lines. The registers in the DMA are selected by the CPU through the address bus by enabling the DS (DMA select) and RS (Register select) inputs. The RD (read) and WR (write) inputs are bidirectional.
When the BG (Bus Grant) input is 0, the CPU can communicate with the DMA registers through the data bus to read from or write to the DMA registers. When BG =1, the DMA can communicate directly with the memory by specifying an address in the address bus and activating the RD or WR control.
DMA Transfer:

The CPU communicates with the DMA through the address and data buses as with any interface unit. The DMA has its own address, which activates the DS and RS lines. The CPU initializes the DMA through the data bus. Once the DMA receives the start control command, it can transfer between the peripheral and the memory.

When BG = 0 the RD and WR are input lines allowing the CPU to communicate with the internal DMA registers. When BG=1, the RD and WR are output lines from the DMA controller to the random access memory to specify the read or write operation of data.

Privileged Instructions and Non- Privileged Instructions:

Instructions are divided into two categories:

- non-privileged instructions
- privileged instructions.

A non-privileged instruction is an instruction that any application or user can execute.

Examples of non-privileged instructions:

```
1  movl
2  addl
3  call
4  ret
```

A privileged instruction, on the other hand, is an instruction that can only be executed in kernel mode. Instructions are divided in this manner because privileged instructions could harm the kernel.

Examples of privileged instructions:

```
1  insl
2  outb
3  inb
4  int
```

Exceptions and Software interrupts:

Exceptions and interrupts are unexpected events that disrupt the normal flow of instruction execution. An exception is an unexpected event from within the processor. An interrupt is an unexpected event from outside the processor. You are to implement exception and interrupt handling in your multicycle CPU design.

External interrupts come from input input (I/O) devices, from a timing device, from a circuit monitoring the power supply, or from any other external source. Examples that cause external interrupts are l/0 device requesting transfer of data, l/0 device finished transfer of data, elapsed time of an event, or power failure.

Internal interrupts arise from illegal or erroneous use of an instruction or data. Internal interrupts are also called traps. Examples of interrupts caused by internal error conditions are register overflow, attempt to divide by zero, an invalid operation code, stack overflow, and protection violation.

External and internal interrupts are initiated from signals that occur in the hardware of the CPU. A software interrupt is initiated by executing an instruction.

Software interrupt is a special call instruction that behaves like an interrupt rather than a subroutine call. It can be used by the programmer to initiate an interrupt procedure at any desired point in the program.

The most common use of software interrupt is associated with a supervisor call instruction. This instruction provides means for switching from a CPU user mode to the supervisor mode. Certain operations in the computer may be assigned to the supervisor mode only, as for example, a complex input or output transfer procedure.

A program written by a user must run in the user mode. When an input or output transfer is required, the supervisor mode is requested by means of a supervisor call instruction. This instruction causes a software interrupt that stores the old CPU state and brings in a new PSW that belongs to the supervisor mode. The calling program must pass information to the operating system in order to specify the particular task requested.

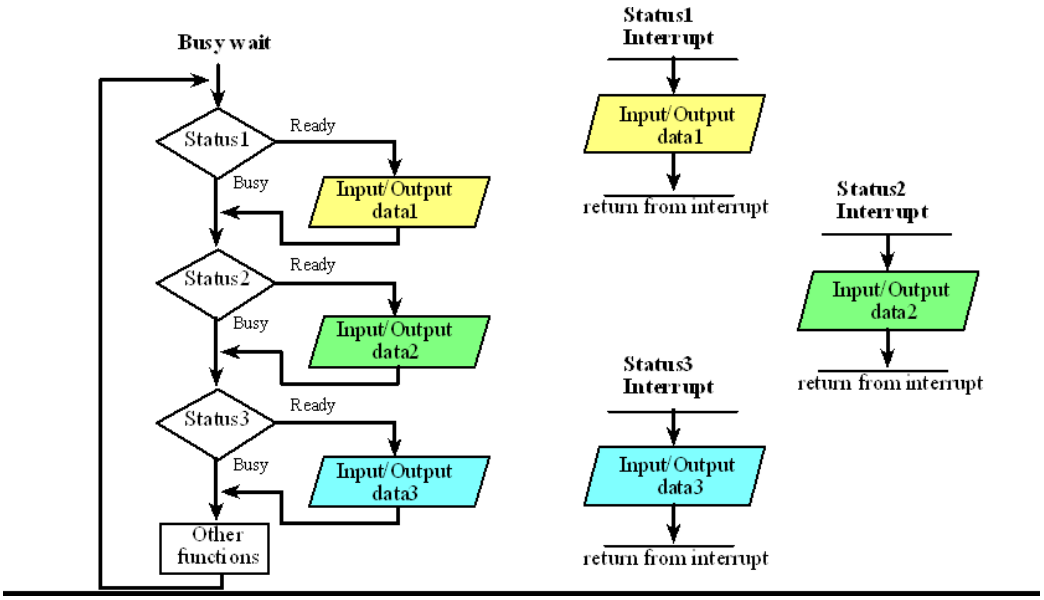
Programs and processes-Role of interrupts in process state transitions

An **interrupt** is the automatic transfer of software execution in response to a hardware event that is asynchronous with the current software execution. This hardware event is called a **trigger**. The hardware event can either be a busy to ready transition in an external I/O device (like the UART input/output) or an internal event (like bus fault, memory fault, or a periodic timer).

When the hardware needs service, signified by a busy to ready state transition, it will request an interrupt by setting its trigger flag. A **thread** is defined as the path of action of software as it executes. The execution of the interrupt service routine is called a background thread. This thread is created by the hardware interrupt request and is killed when the interrupt service routine returns from interrupt (e.g., by executing a **BX LR**). A new thread is created for each interrupt request.

It is important to consider each individual request as a separate thread because local variables and registers used in the interrupt service routine are unique and separate from one interrupt event to the next interrupt. In a **multi-threaded** system, we consider the threads as cooperating to perform an overall task. Consequently we will develop ways for the threads to communicate (e.g., FIFO) and to synchronize with each other. Most embedded systems have a single common overall goal.

On the other hand, general-purpose computers can have multiple unrelated functions to perform. A **process** is also defined as the action of software as it executes. Processes do not necessarily cooperate towards a common shared goal. Threads share access to I/O devices, system resources, and global variables, while processes have separate global variables and system resources. Processes do not share I/O devices.



I/O Device Interfaces

SCSI:

The acronym SCSI stands for Small Computer System Interface. It refers to a standard bus defined by the American National Standards Institute (ANSI) under the designation X3.131 . In the original specifications of the standard, devices such as disks are connected to a computer via a 50-wire cable, which can be up to 25 meters in length and can transfer data at rates up to 5 megabytes/s. The SCSI bus standard has undergone many revisions, and its data transfer capability has increased very rapidly, almost doubling every two years. SCSI-2 and SCSI-3 have been defined, and each has several options.

A SCSI bus may have eight data lines, in which case it is called a narrow bus and transfers data one byte at a time. Alternatively, a wide SCSI bus has 16 data lines and transfers data 16 bits at a time. There are also several options for the electrical signaling scheme used. Devices connected to the SCSI bus are not part of the address space of the processor in the same way as devices connected to the processor bus. The SCSI bus is connected to the processor bus through a SCSI controller. This controller uses DMA to transfer data packets from the main memory to the device, or vice versa. A packet may contain a block of data, commands from the processor to the device, or status information about the device.

To illustrate the operation of the SCSI bus, let us consider how it may be used with a disk drive. Communication with a disk drive differs substantially from communication with the main memory. A controller connected to a SCSI bus is one of two types – an initiator or a target. An initiator has the ability to select a particular target and to send commands specifying the operations to be performed. Clearly, the controller on the processor side, such as the SCSI controller, must be able to operate as an initiator. The disk controller operates as a target. It carries out the commands it receives from the initiator. The initiator establishes a logical connection with the intended target. Once this connection has been established, it can be suspended and restored as needed to transfer commands and bursts of data. While a particular connection is suspended, other device can use the bus to transfer information. This ability to overlap data transfer requests is one of the key features of the SCSI bus that leads to its high performance.

Data transfers on the SCSI bus are always controlled by the target controller. To send a command to a target, an initiator requests control of the bus and, after winning arbitration, selects the controller it wants to communicate with and hands control of the bus over to it.

Then the controller starts a data transfer operation to receive a command from the initiator.

The processor sends a command to the SCSI controller, which causes the following sequence of event to take place:

1. The SCSI controller, acting as an initiator, contends for control of the bus.
2. When the initiator wins the arbitration process, it selects the target controller and hands over control of the bus to it.
3. The target starts an output operation (from initiator to target); in response to this, the initiator sends a command specifying the required read operation.
4. The target, realizing that it first needs to perform a disk seek operation, sends a message to the initiator indicating that it will temporarily suspend the connection between them. Then it releases the bus.
5. The target controller sends a command to the disk drive to move the read head to the first sector involved in the requested read operation. Then, it reads the data stored in that sector and stores them in a data buffer. When it is ready to begin transferring data to the initiator, the target requests control of the bus. After it wins arbitration, it reselects the initiator controller, thus restoring the suspended connection.
6. The target transfers the contents of the data buffer to the initiator and then suspends the connection again. Data are transferred either 8 or 16 bits in parallel, depending on the width of the bus.
7. The target controller sends a command to the disk drive to perform another seek operation. Then, it transfers the contents of the second disk sector to the initiator as before. At the end of these transfers, the logical connection between the two controllers is terminated.
8. As the initiator controller receives the data, it stores them into the main memory using the DMA approach.
9. The SCSI controller sends as interrupt to the processor to inform it that the requested operation has been completed.

This scenario show that the messages exchanged over the SCSI bus are at a higher level than those exchanged over the processor bus. In this context, a “higher level” means that the messages refer to operations that may require several steps to complete, depending on the device. Neither the processor nor the SCSI controller need be aware of the details of operation of the particular device involved in a data transfer. In the preceding example, the processor need not be involved in the disk seek operation.

USB

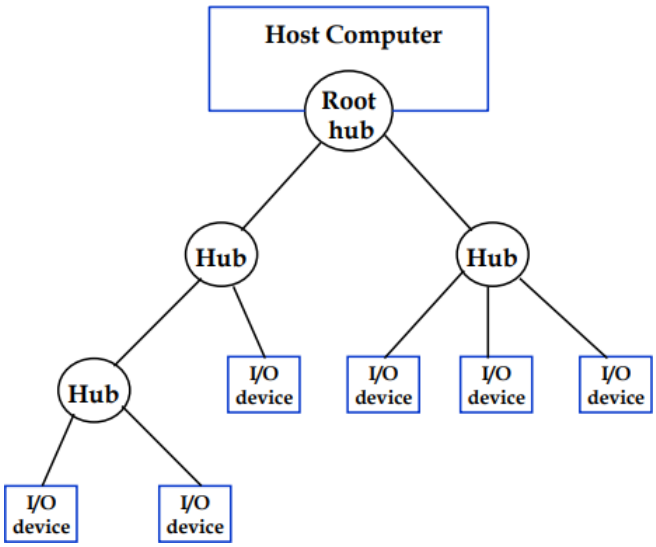
The USB has been designed to meet several key objectives:

- Provide a simple, low-cost, and easy to use interconnection system that overcomes the difficulties due to the limited number of I/O ports available on a computer. ‹
- Accommodate a wide range of data transfer characteristics for I/O devices, including telephone and Internet connections. ‹
- Enhance user convenience through a “plug-and-play” mode of operation.

USB Structure

- A serial transmission format has been chosen for the USB because a serial bus satisfies the low-cost and flexibility requirements.
- Clock and data information are encoded together and transmitted as a single signal.Hence, there are no limitations on clock frequency or distance arising from data skew.
- To accommodate a large number of devices that can be added or removed at any time, the USB has the tree structure. Each node of the tree has a device called a hub, which acts as an intermediate control point between the host and the I/O device. At the root of the tree, a root hub connects the entire tree to the host computer.
- The tree structure enables many devices to be connected while using only simple point-to-point serial links.
- Each hub has a number of ports where devices may be connected, including other hubs.

- In normal operation, a hub copies a message that it receives from its upstream connection to all its downstream ports. As a result, a message sent by the host computer is broadcast to all I/O devices, but only the addressed device will respond to that message.
- A message sent from an I/O device is sent only upstream towards the root of the tree and is not seen by other devices. Hence, USB enables the host to communicate with the I/O devices, but it does not enable these devices to communicate with each other.



USB Protocols:

- All information transferred over the USB is organized in packets, where a packet consists of one or more bytes of information.
- The information transferred on the USB can be divided into two broad categories: control and data. ‹ Control packets perform such tasks as addressing a device to initiate data transfer, acknowledging that data have been received correctly, or indicating an error. Data packets carry information that is delivered to a device. For example, input and output data are transferred inside data packets