UNIT IV

Input-Output Organization: Input-Output Interface, Asynchronous data transfer, Modes of Transfer, Priority Interrupt Direct memory Access.

Memory Organization: Memory Hierarchy, Main Memory, Auxiliary memory, Associate Memory, Cache Memory.

Input –output Organization

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 need.

3. Data codes and formats in peripherals differ form 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.

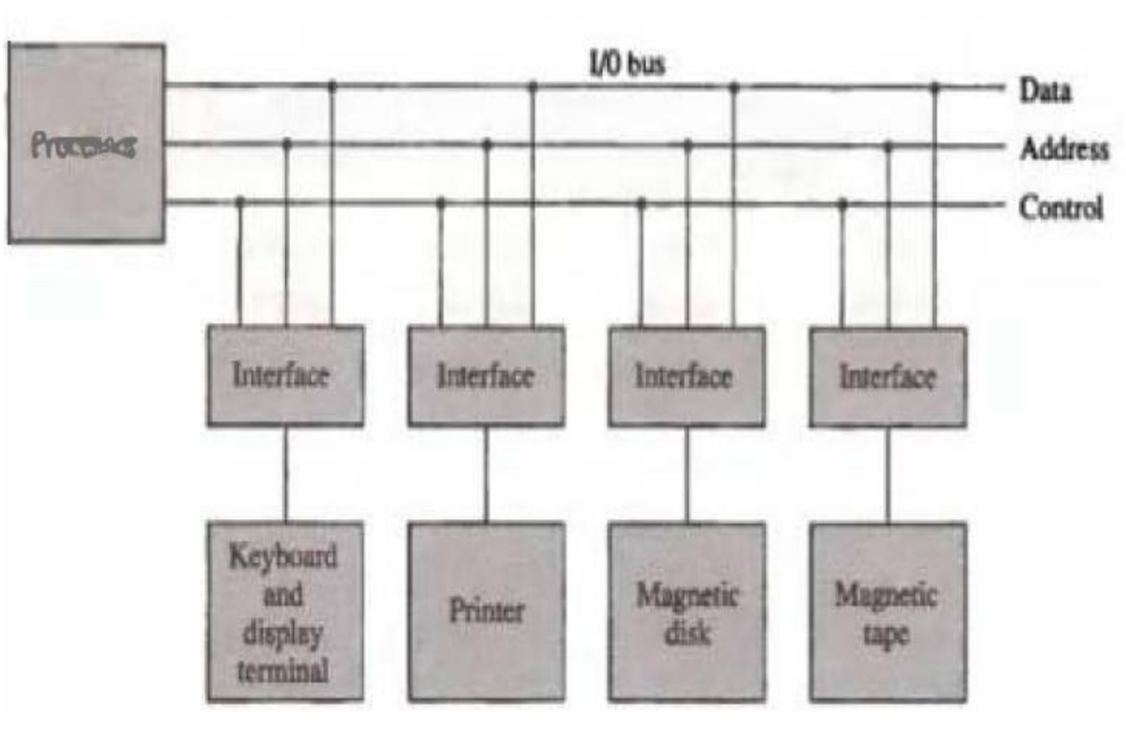
Interface units are used because they interface between the processor bus and the peripheral device. In addition, each device may have its own controller that supervises the operations of the particular mechanism in the peripheral.

Two main types of interface are CPU interface that corresponds to the system bus and input-output interface that depends on the nature of input- output device.

I/O Bus and Interface Modules

A typical communication link between the processor and several peripherals is shown in Fig below. I/O bus consists of data lines, address lines, and control lines. The magnetic disk, printer, and terminal are employed in practically any general-purpose computer. The magnetic tape is used in some computers for backup storage. Each peripheral device has associated with it an interface unit.

Each interface decodes the address and control received from the I/O bus, interprets them for the peripheral, and provides signals for the peripheral controller. Each peripheral has its own controller that operates the particular electromechanical device.



The I/O bus from the processor is attached to all peripheral interfaces. To communicate with a particular device, the processor places a device address on the address lines. Each interface attached to the I/O bus contains an address decoder that monitors the address lines.

All peripherals whose address does not correspond to the address in the bus are disabled their interface.

The interface selected responds to the function code and proceeds to execute it. The function code is referred to as an I/O command and is in essence an instruction that is executed in the interface and its attached peripheral unit.

There are four types of commands that an interface may receive. They are classified as control, status, status, data output, and data input.

A control command is issued to activate the peripheral and to inform it what to do. For example, a magnetic tape unit may be instructed to backspace the tape by one record, to rewind the tape, or to start the tape moving in the forward direction. The particular control command issued depends on the peripheral, and each peripheral receives its own distinguished sequence of control commands, depending on its mode of operation.

A status command is used to test various status conditions in the interface and the peripheral.

A data output command causes the interface to respond by transferring data from the bus into one of its registers.

The data input command is the opposite of the data output.
In this case the interface receives an item of data from the peripheral
and places it in its buffer register

I/O versus Memory Bus

In addition to communicating 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 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.

In the first method, the computer has independent sets of data, address, and control buses, one for accessing memory and the other for I/O. This is done in computers that provide a separate I/O processor (IOP) in addition to the central processing unit (CPU).

The purpose of the IOP is to provide an independent pathway for the transfer of information between external devices and internal memory.

Isolated Versus Memory-Mapped I/O

Many computers use one common bus to transfer information between memory or I/O and the CPU. The distinction between a memory transfer and I/O transfer is made through separate read and write lines. The CPU specifies whether the address on the address lines is for a memory word or for an interface register by enabling one of two possible read or write lines. The I/O read and I/O write control lines are enabled during an I/O transfer.

In the isolated I/O configuration, the CPU has distinct input and output instructions, and each of these instructions is associated with the address of an interface register. When the CPU fetches and decodes the operation code of an input or output instruction, it places the address associated with the instruction into the common address lines. At the same time, it enables the I/O read (for input) or I/O write (for output) control line.

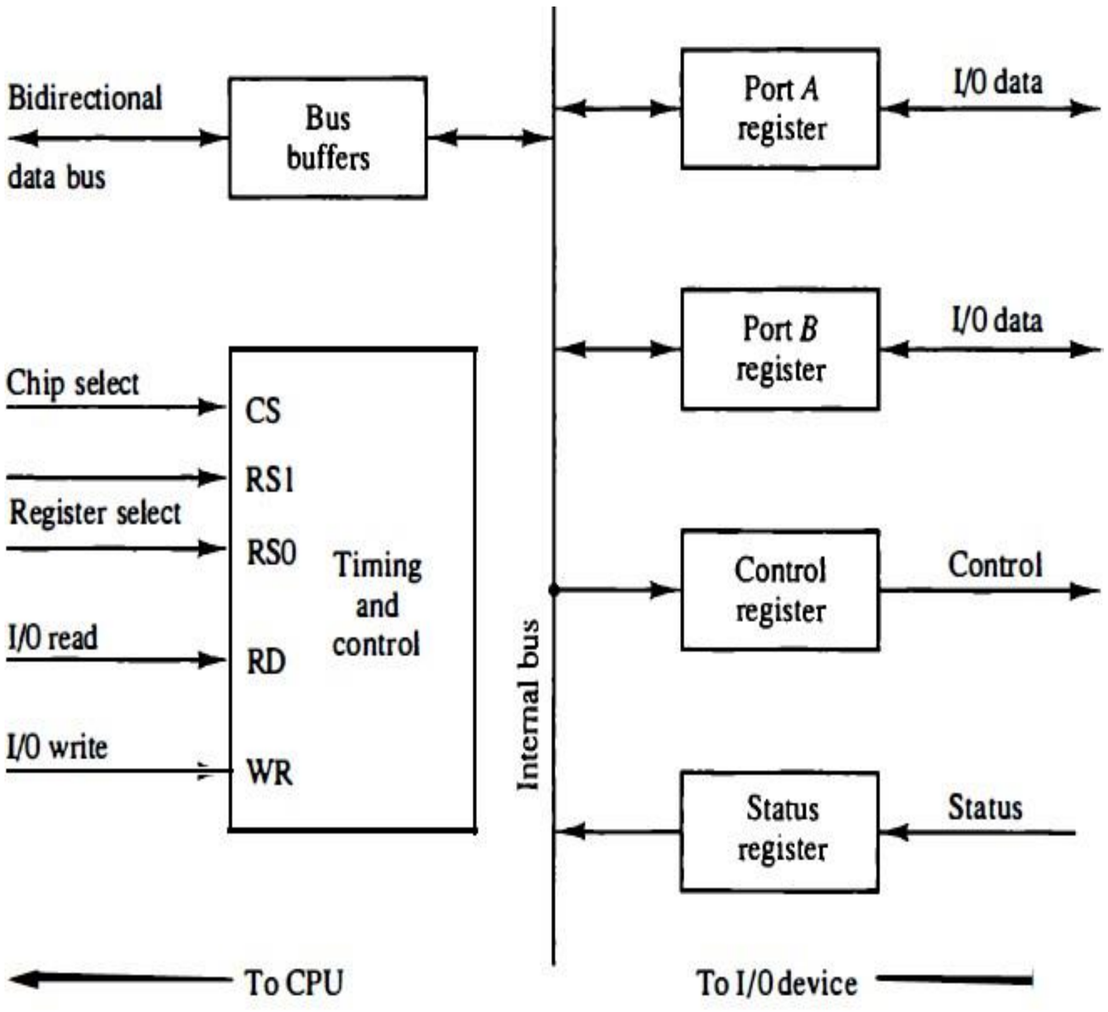
The isolated I/O method isolates memory word and not for an I/O addresses so that memory address values are not affected by interface address assignment since each has its own address space. The other alternative is to use the same address space for both memory and I/O. This is the case in computers that employ only one set of read and write signals and do not distinguish between memory and I/O addresses. This configuration is referred to as memory-mapped I/O.

In a memory-mapped I/O organization there are no specific input or output instructions. The CPU can manipulate I/O data residing in interface registers with the same instructions that are used to manipulate memory words. Each interface is organized as a set of registers that respond to read and write requests in the normal address space.

Example of I/O Interface

An example of an I/O interface unit is shown in block diagram form in Fig below.

Asynchronous data transfer



CS	RS1	RS0	Register selected
0	×	×	None: data bus in high-impedance
1	0	0	Port *A* register
1	0	1	Port *B* register
1	1	0	Control register
1	1	1	Status register

It consists of two data registers called ports, a control register, a status register, bus buffers, and timing and control circuits. The interface communicates with the CPU through the data bus. The chip select and register select inputs determine the address assigned to the interface. The I/O read and write are two control lines that specify an input or output, respectively. The four registers communicate directly with the I/O device attached to the interface.

The I/O data to and from the device can be transferred into either port A or Port B. The interface may operate with an output device or with an input device, or with a device that requires both input and output. If the interface is connected to a printer, it will only output data, and if it services a character reader, it will only input data. A magnetic disk unit transfers data in both directions but not at the same time, so the interface can use bidirectional lines. A command is passed to the I/O device by sending a word to the appropriate interface register. In a system like this, the function code in the I/O bus is not needed because control is sent to the control register, status information is received from the status register, and data are transferred to and from ports A and B registers. Thus the transfer of data, control, and status information is always via the common data bus. The distinction between data, control, or status information is determined from the particular register with which the CPU communicates.

The interface registers communicate with the CPU through the bidirectional data bus. The address bus selects the interface unit through the chip select and the two register select inputs. A circuit must be provided externally (usually, a decoder) to detect the address assigned to the interface registers. This circuit enables the chip select (CS) input when the interface is selected by the address bus. The two register select inputs RS1 and RS0 are usually connected to the two least significant lines of the lines address bus. These two inputs select one of the four registers in the interface as specified in the table accompanying the diagram. The content of the selected register is transfer into the CPU via the data bus when the I/O read signal is enabled. The CPU transfers binary information into the selected register via the data bus when the I/O write input is enabled.

Input –output Organization

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:

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.

The data transfer rate of peripherals is usually slower than the transfer rate of the CPU, and consequently, a synchronization mechanism may be need.

Data codes and formats in peripherals differ form the word format in the CPU and memory.

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.

Interface units are used because they interface between the processor bus and the peripheral device. In addition, each device may have its own controller that supervises the operations of the particular mechanism in the peripheral.

Two main types of interface are CPU interface that corresponds to the system bus and input-output interface that depends on the nature of input-output device.

Modes of Transfer

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)

Programmed I/O operations are the result of I/O instructions written in the computer program.

When the interface determines that the device is ready for data transfer, it generates an interrupt request to the computer. Upon detecting the external interrupt signal, the CPU momentarily stops the task it is processing, branches to a service program to process the I/O transfer, and then returns to the task it was originally performing

Transfer of data under programmed I/O is between CPU and peripheral. In direct memory access (DMA), the interface transfers data into and out of the memory unit through the memory bus. The CPU initiates the transfer by supplying the interface with the starting address and the number of words needed to be transferred and then proceeds to execute other tasks.

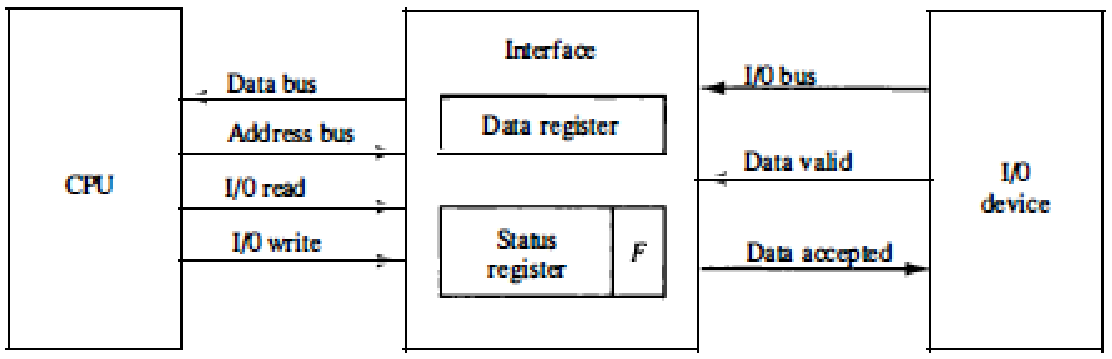
Many computers combine the interface logic with the requirements for direct memory access into one unit and call it an I/O processor (IOP). The IOP can handle many peripherals through a DMA and interrupt facility. In such a system, the computer is divided into three separate modules: the memory unit, the CPU, and the

IOP.

Example of Programmed I/O

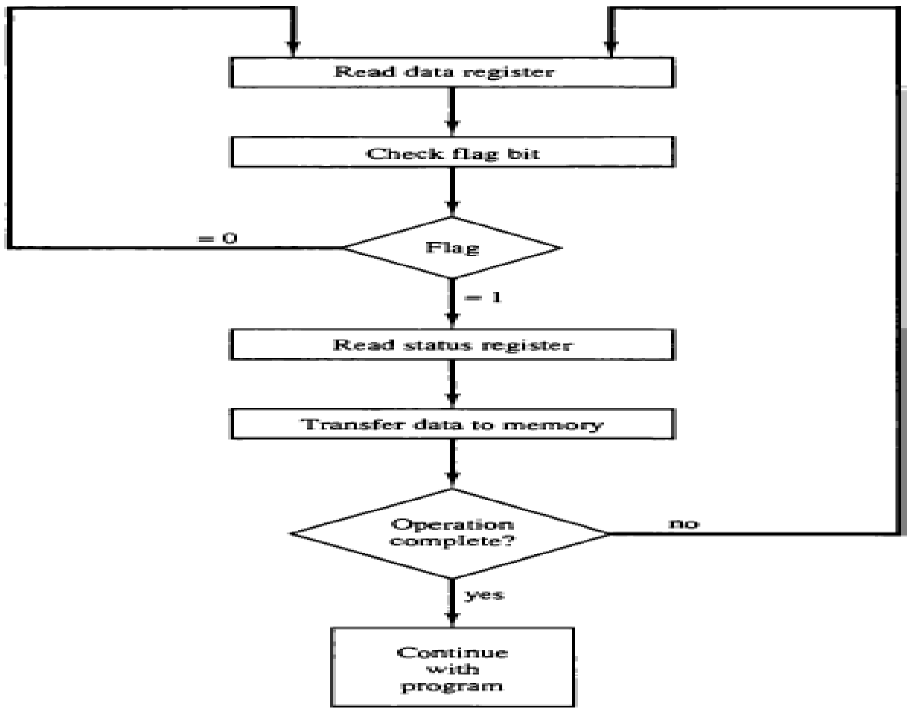
In the programmed I/O method, the I/O device dies not have direct access to memory. A transfer from an I/O device to memory requires the execution of several instructions by the CPU, including an input instruction to transfer the data from the device to the CPU, and a store instruction to transfer the data from the CPU to memory.

An example of data transfer from an I/O device through an interface into the CPU is shown in Fig.



F = Flag bit

The device transfers bytes of data one at a time as they are available.
When a byte of data is available, the device places it in the I/O bus and enables its data valid line. The interface accepts the byte into its data register and enables the data accepted line. The interface sets a it in the status register that we will refer to as an F or "flag" bit. The device can now disable the data valid line, but it will not transfer another byte until the data accepted line is disabled by the interface. A flowchart of the program that must be written for the CPU.



It is assumed that the device is sending a sequence of bytes that must be stored in memory. The transfer of each byte requires three instructions:

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

Each byte is read into a CPU register and then transferred to memory with a store instruction. A common I/O programming task is to transfer a block of words form an I/O device and store them in a memory buffer. A program that stores input characters in a memory buffer using the instructions mentioned in the earlier chapter.

_Interrupt-Initiated I/O

The CPU responds to the interrupt signal by storing the return address from the program counter into a memory stack and then control branches to a service routine that processes the required I/O transfer. The way that the processor chooses the branch address of the service routine varies from tone unit to another. In principle, there are two methods for accomplishing this. One is called vectored interrupt and the other, no vectored interrupt. In a non vectored interrupt, the branch address is assigned to a fixed location in memory. In a vectored interrupt, the source that interrupts supplies the branch information to the computer. This information is called the **interrupt vector.**

Software Considerations

A computer must also have software routines for controlling peripherals and for transfer of data between the processor and peripherals. I/O routines must issue control commands to activate the peripheral and to check the device status to determine when it is ready for data transfer.Error checking and other useful steps often accompany the transfers. In interrupt-controlled transfers, the I/O software must issue commands to the peripheral to interrupt when ready and to service the interrupt when it occurs. In DMA transfer, the I/O software must initiate
the DMA channel to start its operation.

Priority Interrupt

Data transfer between the CPU and an I/O device is initiated by the CPU. However, the CPU cannot start the transfer unless the device is ready to communicate with the CPU.

A priority interrupts is a system that establishes a priority over the various sources to determine which condition is to be serviced first when two or more request 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 request which, if delayed of interrupted, could have serious consequences. Devices with high- speed transfers such as keyboards receive low priority. When two devices interrupt the computer at the same time, the computer services the devices interrupt the computer at the same time, the computer services the device, with the higher priority first.

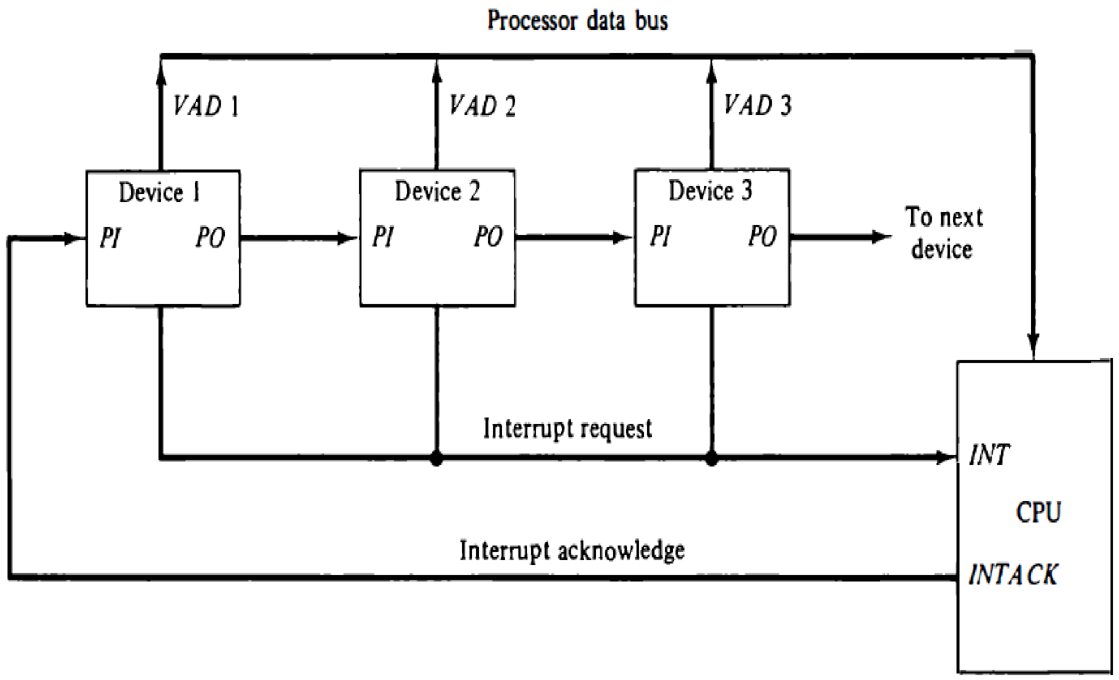
The disadvantage of the soft ware method is that if there are many interrupts, the time required to poll them can exceed the time available to service the I/O device. In this situation a hardware priority-interrupt unit can be used to speed up the operation.

A hardware priority-interrupt unit functions as an overall manager in an interrupt system environment. It accepts interrupt requests from many sources, determines which of the incoming requests has the highest priority, and issues an interrupt request to the computer based on this determination. To speed up the operation, each interrupt source has its own interrupt vector to access its own service routine directly. Thus no polling is required because all the decisions are established by the hardware priority-interrupt unit. The hardware priority function can be established by either a serial or a parallel connection of interrupt lines. The serial connection is also known as the **daisy chaining method.**

Daisy-Chaining Priority

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. This method of connection between three devices and the CPU is



shown in Fig.

The interrupt request line is common to all devices and forms a wired logic connection. If any device has its interrupt signal in the low-level state, the interrupt line goes to the low-level state and enables the interrupt input in the CPU.

When no interrupts are pending, the interrupt line stays in the high-level state and no interrupts are recognized by the CPU.

This signal is received by device 1 at its PI (priority in) input. The acknowledge signal passes on to the next device through the PO (priority out) output only if device 1 is not requesting an interrupt. If device 1 has a pending interrupt, it blocks the acknowledge signal from the next device by placing a 0 in the PO output. It then proceeds to insert its own interrupt vector address (VAD) into the data bus for the CPU to use during the interrupt cycle.

A device with a 0 in its PI input generates a 0 in its PO output to inform the next-lower-priority device that the acknowledge signal has been blocked. A device that is requesting an interrupt and has a 1 in its PI input will intercept the acknowledge signal by placing a 0 in its PO output. If the device does not have pending interrupts, it transmits the acknowledge signal to the next device by placing a 1 in its PO output. Thus the device with PI = 1 and PO = 0 is the one with the highest priority that is requesting an interrupt, and this device places its VAD on the data bus. The daisy chain arrangement gives the highest priority to the device that receives the interrupt acknowledge signal from the CPU. The farther the device is from the first position, the lower is its priority.

Below figure shows the internal logic that must be included with in each device when connected in the daisy-chaining scheme.

The device sets its RF flip-flop when it wants to interrupt the CPU. The output of the RF flip-flop goes through an open-collector inverter, a circuit that provides the wired logic for the common interrupt line. If PI = 0, both PO and the enable line to VAD are equal to 0, irrespective of the value of RF. If PI = 1 and RF = 0, then PO = 1 and the vector address is disabled. This condition passes the acknowledge signal to the next device through PO. The device is active when PI = 1 and RF = 1.

This condition places a 0 in PO and enables the vector address for the data bus. It is assumed that each device has its own distinct vector address. The RF flip-flop is reset after a sufficient delay to ensure that the CPU has received the vector address.

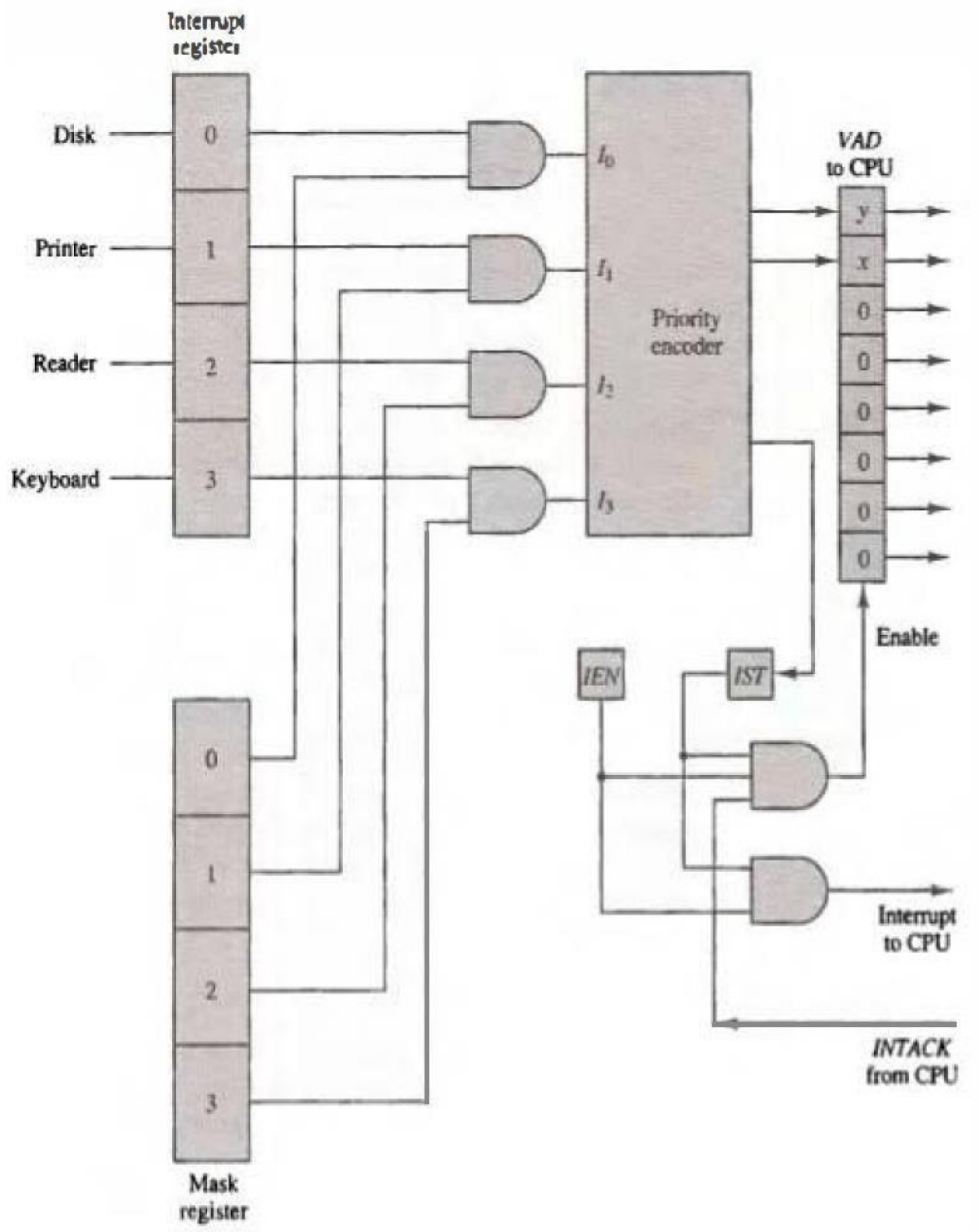
Parallel Priority Interrupt

The parallel priority interrupt method uses a register whose bits are set separately by the interrupt signal from each device. Priority is established according to the position of the bits in the register. In addition to the interrupt register the circuit may include a mask register whose purpose is to control the status of each interrupt request.

The mask register can be programmed to disable lower-priority interrupts while a higher-priority device is being serviced. It can also provide a facility that allows a high-priority device to interrupt the CPU while a lower-priority device is being serviced.

The priority logic for a system of four interrupt sources is shown in Fig.

It consists of an interrupt register whose individual bits are set by external conditions and cleared by program instructions. The magnetic disk, being a high-speed device, is given the highest priority. The printer has the next priority, followed by a character reader and a keyboard. The mask register has the same number of bits as the interrupt register. By means of program instructions, it is possible to set or reset any bit in the mask register.



MEMORY ORGANIZATION

MEMORY HIERARCHY

The memory unit is an essential component in any digital computer since it is needed for storing programs and data. A very small computer with a limited application may be able to fulfill its intended task without the need of additional storage capacity. Most general-purpose computers would run more efficiently if they were equipped with additional storage beyond the capacity of the main memory. There is just not enough space in one memory unit to accommodate all the programs used in a typical computer. Moreover, most computer users accumulate and continue to accumulate large amounts of data-processing software.

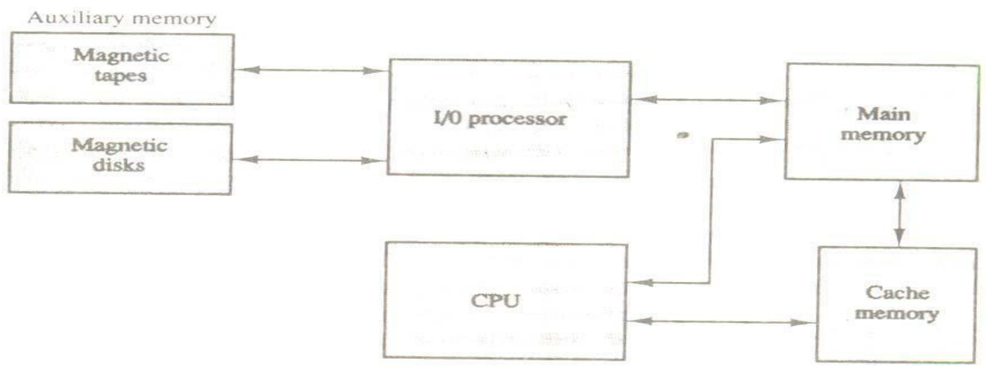
The memory unit that communicates directly with the CPU is called the main memory.

Devices that provide backup storage are called auxiliary memory. The most common auxiliary memory devices used in computer systems are magnetic disks and tapes. They are used for storing system programs, large data files, and other backup information.

Only programs and data currently needed by the processor reside in main memory. All other information is stored in auxiliary memory and transferred to main memory when needed.

The total memory capacity of a computer can be visualized as being a hierarchy of components.

A special very-high speed memory called a cache is sometimes used to increase the speed of processing by making current programs and data available to the CPU at a rapid rate. The cache memory is employed in computer systems to compensate for the speed differential between main memory access time and processor logic. CPU logic is usually faster than main memory access time, with the result that processing speed is limited primarily by the speed of main memory.



Memory hierarchy in a computer system.

Making programs and data available at a rapid rate, it is possible to increase the performance rate of the computer.

While the I/O processor manages data transfers between auxiliary memory and main memory, the cache organization is concerned with the transfer of information between main memory and CPU. Thus each is involved with a different level in the memory hierarchy system. The reason for having two or three levels of memory hierarchy is economics.

As the storage capacity of the memory increases, the cost per bit for storing binary information decreases and the access time of the memory becomes longer.

The auxiliary memory has a large storage capacity, is relatively inexpensive, but has low access speed compared to main memory. The cache memory is very small, relatively expensive, and has very high access speed.

MAIN MEMORY

The main memory is the central storage unit in a computer system. It is a relatively large and fast memory used to store programs and data during the computer operation.

The principal technology used for the main memory is based on semiconductor integrated circuits. Integrated circuit RAM chips are available in two possible operating modes, **static and dynamic**. The static RAM consists essentially of internal flip-flops that store the binary information.The stored information remains valid as long as power is applied to unit.

The dynamic RAM stores the binary information in the form of electric charges that are applied to capacitors. The stored charge on the capacitors tend to discharge with time and the capacitors must be periodically recharged by refreshing the dynamic memory.

Refreshing is done by cycling through the words every few milliseconds to restore the decaying charge. The dynamic RAM offers reduced power consumption and larger storage capacity in a single memory chip.

The static RAM is easier to use and has shorted read and write cycles.Originally, RAM was used to refer to a random-access memory, but now it is used to designate a read/write memory to distinguish it from a read-only memory, although ROM is also random access. RAM is used for storing the bulk of the programs and data that are subject to change. ROM is used for storing programs that are permanently resident in the computer and for tables of constants that do not change in value one the production of the computer is completed.

The ROM portion of main memory is needed for storing an initial program called a bootstrap loader. The bootstrap loader is a program whose function is to start the computer software operating when power is turned on.

Since RAM is volatile, its contents are destroyed when power is turned off. The contents of ROM remain unchanged after power is turned off and on again transferred to the operating system, which prepares the computer for general use.

RAM and ROM chips are available in a variety of sizes. If the memory needed for the computer is larger than the capacity of one chip, it is necessary to combine a number of chips to form the required memory size

RAM AND ROM CHIPS

A RAM chip is better suited for communication with the CPU if it has one or more control inputs that select the chip only when needed. Another common feature is a bidirectional data bus that allows the transfer of data either from memory to CPU during a read operation or from CPU to memory during a write operation.

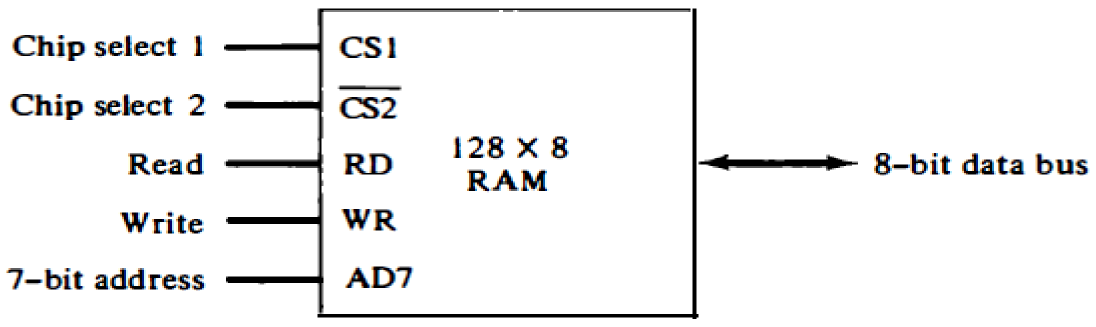
A bidirectional bus can be constructed with three-state buffers. A three- state buffer output can be placed in one of three possible states: a signal equivalent to logic 1, a signal equivalent to logic 0, or a high-impedance state. The logic 1 and 0 are normal digital signals. The high-impedance state behaves like an open circuit, which means that the output does not carry a signal and has no logic significance.

The block diagram of a RAM chip is shown in Fig .The capacity of the memory is 128 words of eight bits (one byte) per word. This requires a 7-bit address and an 8-bit bidirectional data bus.

The read and write inputs specify the memory operation and the two chips select (CS) control inputs are for enabling the chip only when it is selected by the microprocessor.

The availability of more than one control input to select the chip facilitates the decoding of the address lines when multiple chips are used in the microcomputer.

The read and write inputs are sometimes combined into one line labeled R/W. When the chip is selected, the two binary states in this line specify the two operations or read or write.



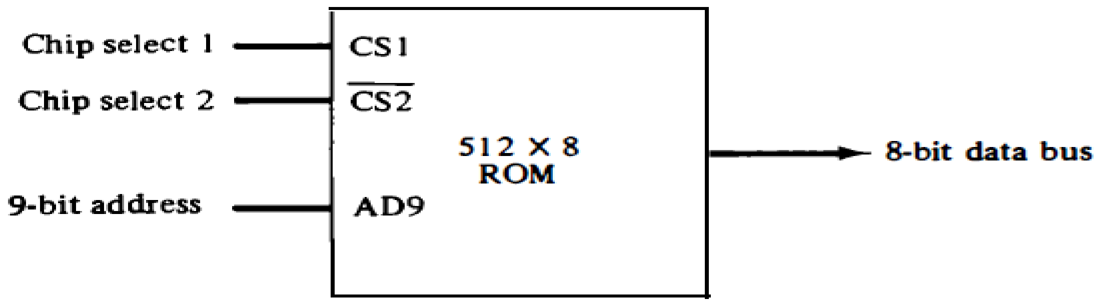
(a) Block diagram

CS1	$\overline{CS2}$	RD	WR	Memory function	State of data bus
0	0	×	×	Inhibit	High-impedance
0	1	×	×	Inhibit	High-impedance
1	0	0	0	Inhibit	High-impedance
1	0	0	1	Write	Input data to RAM
1	0	1	×	Read	Output data from RAM
1	1	×	×	Inhibit	High-impedance

(b) Function table

The function table listed in Fig(b) specifies the operation of the RAM chip. The unit is in operation only when CSI = 1 and CS2 = 0. The bar on top of the second select variable indicates that this input in enabled when it is equal to 0. If the chip select inputs are not enabled, or if they are enabled but the read but the read or write inputs are not enabled, the memory is inhibited and its data bus is in a high-impedance state. When SC1 = 1 and CS2 = 0, the memory can be placed in a write or read mode. When the WR input is enabled, the memory stores a byte from the data bus into a location specified by the address input lines. When the RD input is enabled, the content of the selected byte is placed into the data bus. The RD and WR signals control the memory operation as well as the bus buffers associated with the bidirectional data bus.

A ROM chip is organized externally in a similar manner. However, since a ROM can only read, the data bus can only be in an output mode. The block diagram of a ROM chip is shown in Fig. For the same-size chip, it is possible to have more bits of ROM occupy less space than in RAM. For this reason, the diagram specifies a 512-byte ROM, while the RAM has only 128 bytes.

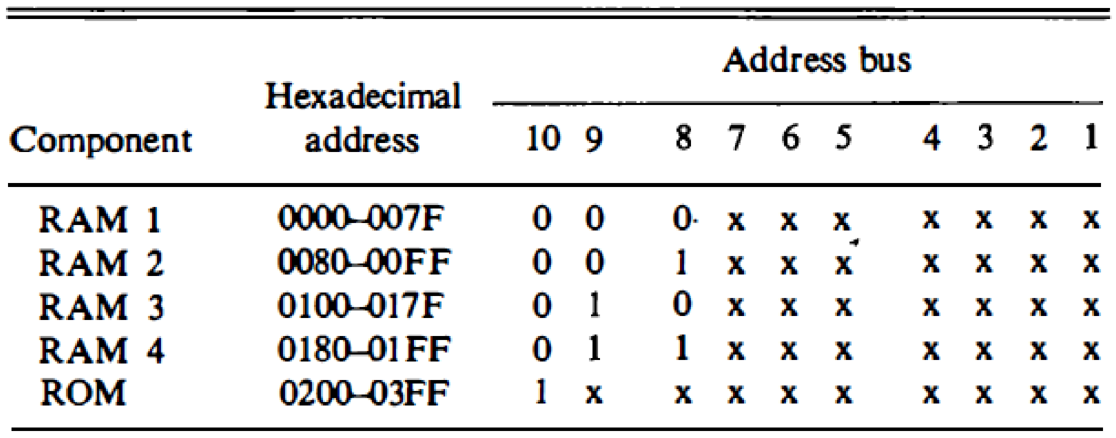


The nine address lines in the ROM chip specify any one of the 512 bytes stored in it. The two chip select inputs must be CS1 = 1 and CS2 = 0 for the unit to operate. Otherwise, the data bus is in a high-impedance state. There is no need for a read or write control because the unit can only read. Thus when the chip is enabled by the two select inputs, the byte selected by the address lines appears on the data bus.

MEMORY ADDRESS MAP

The designer of a computer system must calculate the amount of memory required for the particular application and assign it to either RAM or ROM. The interconnection between memory and processor is then established form knowledge of the size of memory needed and the type of RAM and ROM chips available. The addressing of memory can be established by means of a table that specifies the memory address assigned to each chip. The table, called a memory address map, is a pictorial representation of assigned address space for each chip in the system. To demonstrate with a particular example, assume that a computer system needs 512 bytes of RAM and 512 bytes of ROM. The RAM and ROM chips

Memory Address Map for Microprocomputer



Component	Hexadecimal address	Address bus									
		10	9	8	7	6	5	4	3	2	1
RAM 1	0000–007F	0	0	0	x	x	x	x	x	x	x
RAM 2	0080–00FF	0	0	1	x	x	x	x	x	x	x
RAM 3	0100–017F	0	1	0	x	x	x	x	x	x	x
RAM 4	0180–01FF	0	1	1	x	x	x	x	x	x	x
ROM	0200–03FF	1	x	x	x	x	x	x	x	x	x

The component column specifies whether a RAM or a ROM chip is used. The hexadecimal address column assigns a range of hexadecimal equivalent addresses for each chip. The address bus lines are listed in the third column. Although there are 16 lines in the address bus, the table shows only 10 lines because the other 6 are not used in this example and are assumed to be zero. The small x's under the address bus lines designate those lines that must be connected to the address inputs in each chip. The RAM chips have 128 bytes and need seven address lines. The ROM chip has 512 bytes and needs 9 address lines. The x's are always assigned to the low-order bus lines: lines 1 through 7 for the RAM and lines 1 through 9 for the ROM. It is now necessary to distinguish between four RAM chips by assigning to each a different address. For this particular example we choose bus lines 8 and 9 to represent four distinct binary combinations. Note that any other pair of unused bus lines can be chosen for this purpose. The table clearly shows that the nine low-order bus lines constitute a memory space fro RAM equal to 2^9 = 512 bytes. The distinction between a RAM and ROM address is done with another bus line. Here we choose line 10 for this purpose. When line 10 is 0, the CPU selects a RAM, and when this line is equal to 1, it selects the ROM.

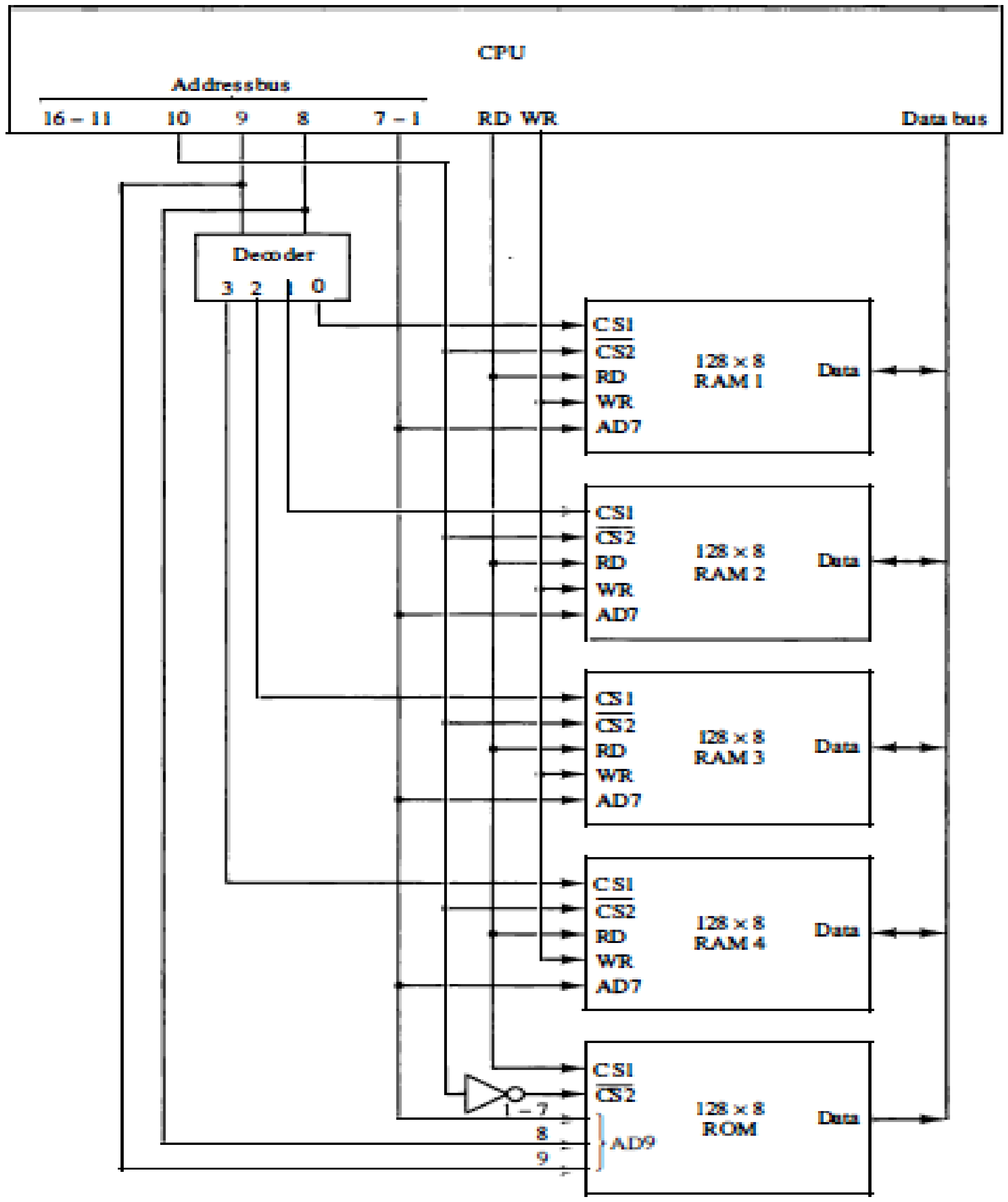
The equivalent hexadecimal address for each chip is obtained form the information under the address bus assignment. The address bus lines are subdivided into groups of four bits each so that each group can be represented with a hexadecimal digit. The first hexadecimal digit represents lines 13 to 16 and is always 0. The next hexadecimal digit represents lines 9 to 12, but lines 11 and 12 are always 0. The range of hexadecimal addresses for each component is determined from the x's associated with it. These x's represent a binary number that can range from an all-0's to an all-1's value.

MEMORY CONNECTION TO CPU

RAM and ROM chips are connected to a CPU through the data and address buses. The low-order lines in the address bus select the byte within the chips and other lines in the address bus select a particular chip through its chip select inputs. The connection of memory chips to the CPU is shown in Fig.

This configuration gives a memory capacity of 512 bytes of RAM and 512 bytes of ROM. Each RAM receives the seven low-order bits of the address bus to select one of 128 possible bytes. The particular RAM chip selected is determined from lines 8 and 9 in the address bus.

This is done through a 2X4 decoder whose outputs go to the SCI input in each RAM chip. Thus, when address lines 8 and 9 are equal to 00, the first RAM chip is selected. When 01, the second RAM chip is selected, and so on. The RD and WR outputs from the microprocessor are applied to the inputs of each RAM chip.



The selection between RAM and ROM is achieved through bus line 10. The RAMs are selected when the bit in this line is 0, and the ROM when the bit is 1. The other chip select input in the ROM is connected to the RD control line for the ROM chip to be enabled only during a read operation. Address bus lines 1 to 9 are applied to the input address of ROM without going through the decoder. This assigns addresses 0 to 511 to RAM and 512 to 1023 to ROM. The data bus of the ROM has only an output capability, whereas the data bus connected to the RAMs can transfer information in both directions.

AUXILIARY MEMORY

The most common auxiliary memory devices used in computer systems are magnetic disks and tapes. Other components used, but not as frequently, are magnetic drums, magnetic bubble memory, and optical disks. To understand fully the physical mechanism of auxiliary memory devices one must have a knowledge of magnetics, electronics, and electromechanical systems. Although the physical properties of these storage devices can be quite complex, their logical properties can be characterized and compared by a few parameters. The important characteristics of any device are its access mode, access time, transfer rate, capacity, and cost.

The average time required to reach a storage location in memory and obtain its contents is called the **access time.** In electromechanical devices with moving parts such as disks and tapes, the access time consists of a seek time required to position the read-write head to a location and a transfer time required to transfer data to or from the device. Because the seek time is usually much longer than the transfer time, auxiliary storage is organized in records or blocks.

A record is a specified number of characters or words. Reading or writing is always done on entire records. The transfer rate is the number of characters or words that the device can transfer per second, after it has been positioned at the beginning of the record.

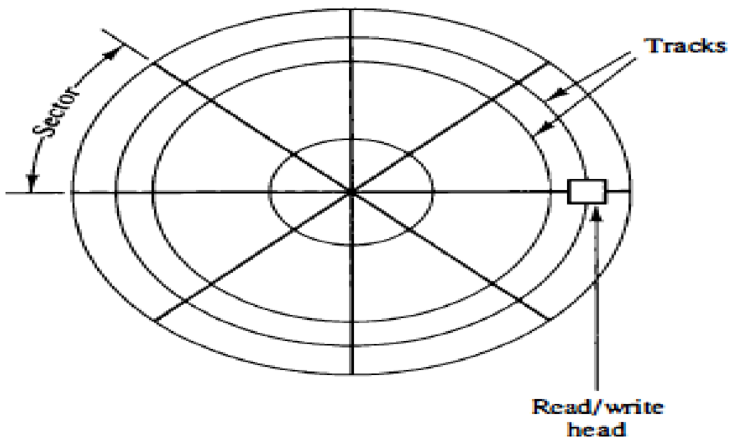
Magnetic drums and disks are quite similar in operation. Both consist of high- speed rotating surfaces coated with a magnetic recording medium. The rotating surface of the drum is a cylinder and that of the disk, a round flat plate. The amount of surface available for recording in a disk is greater than in a drum of equal physical size. Therefore, more information can be stored on a disk than on a drum of comparable size. For this reason, disks have replaced drums in more recent computers

MAGNETIC DISKS

A magnetic disk is a circular plate constructed of metal or plastic coated with magnetized material. Often both sides of the disk are used and several disks may be stacked on one spindle with read/write heads available on each surface. All disks rotate together at high speed and are not stopped or started from access purposes. Bits are stored in the magnetized surface in spots along concentric circles called tracks. The tracks are commonly divided into sections called **sectors.** In most systems, the minimum quantity of information which can be transferred is a sector. The sub division of tone disk surface into tracks and sectors.

Some units use a single read/write head from each disk surface and some different read/write head. The address can then select a particular track electronically through a decoder circuit. This type of unit is more expensive and is found only in very large computer systems.

Disks may have multiple heads and simultaneous transfer of bits from several tracks at the same time.



Disks that are permanently attached to the unit assembly and cannot be removed by the occasional user are called hard disks. A disk drive with removable disks is called a floppy disk. The disks used with a floppy disk drive are small removable disks made of plastic coated with magnetic recording material. There are two sizes commonly used, with diameters of 5.25 and 3.5 inches. The 3.5-inch disks are smaller and can store more data than can the 5.25-inch disks. Floppy disks are extensively used in personal computers as a medium for distributing software to computer users.

MAGNETIC TAPE

A magnetic tape transport consists of the electrical, mechanical, and electronic components to provide the parts and control mechanism for a magnetic-tape unit. The tape itself is a strip of plastic coated with a magnetic recording medium. Bits are recorded as magnetic spots on the tape along several tracks. Usually, seven or nine bits are recorded simultaneously to form a character together with a parity bit. Read/write heads are mounted one in each track so that data can be recorded and read as a sequence of characters.

Magnetic tape units can be stopped, started to move forward or in reverse, or can be rewound. However, they cannot be started or stopped fast enough between individual characters. For this reason, information is recorded in blocks referred to as records. Gaps of unrecorded tape are inserted between records where the tape can be stopped.

ASSOCIATIVE MEMORY

Many data-processing applications require the search of items in a table stored in memory. An assembler program searches the symbol address table in order to extract the symbol’s binary equivalent. An account number may be searched in a file to determine the holder’s name and account status.

The established way to search a table is to store all items where they can be addressed in sequence. The search procedure is a strategy for choosing a sequence of addresses, reading the content of memory at each address, and comparing the information read with the item being searched until a match occurs. The number of accesses to memory depends on the location of the item and the efficiency of the search algorithm.

The time required to find an item stored in memory can be reduced considerably if stored data can be identified for access by the content of the data itself rather than by an address. A memory unit accessed by content is called **an associative memory or content addressable memory (CAM).**

This type of memory is accessed simultaneously and in parallel on the basis of data content rather than by specific address or location. When a word is written in an associative memory, no address is given. The memory is capable of finding an empty unused location to store the word. When a word is to be read from an associative memory, the content of the word, or part of the word, is specified.

The memory locaters all words which match the specified content and marks them for reading. Because of its organization, the associative memory is uniquely suited to do parallel searches by data association.

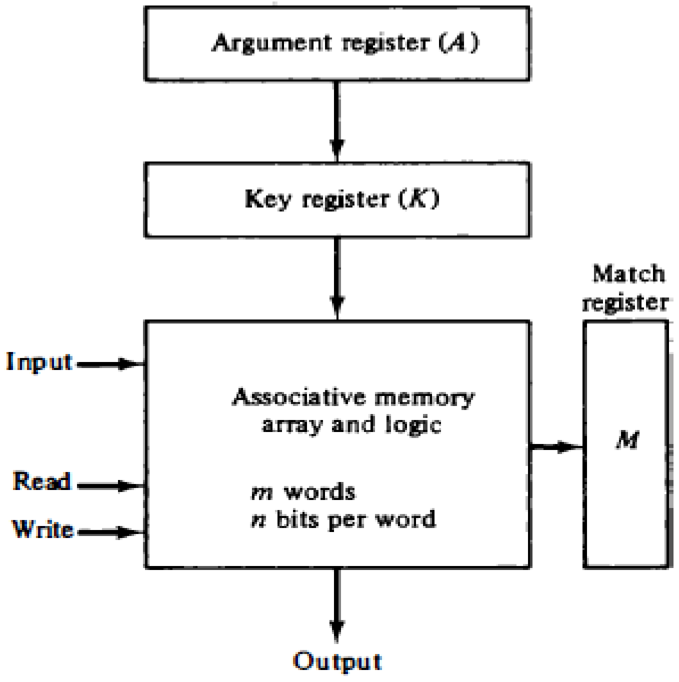
An associative memory is more expensive then a random access memory because each cell must have storage capability as well as logic circuits for matching its content with an external argument.

For this reason, associative memories are used in applications where the search time is very critical and must be very short.

HARDWARE ORGANIZATION

The block diagram of an associative memory consists of a memory array and logic from words with n bits per word. The argument register A and key register K each have n bits, one for each bit of a word.

Block diagram for Associate Memory



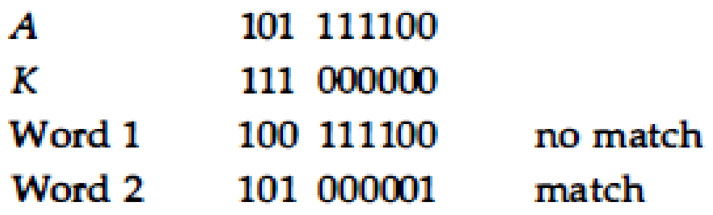
The match register M has m bits, one for each memory word. Each word in memory is compared in parallel with the content of the argument register.

The words that match the bits of the argument register set a corresponding bit in the match register. After the matching process, those bits in the match register that have been set indicate the fact that their corresponding words have been matched. Reading is accomplished by a sequential access to memory for those words whose corresponding bits in the match register have been set.

The key register provides a mask for choosing a particular field or key in the argument word. The entire argument is compared with each memory word if the key register contains all 1's. Otherwise, only those bits in the argument that have 1's in their corresponding position of the key register are compared. Thus the key provides a mask or identifying piece of information which specifies how the reference to memory is made.

To illustrate with a numerical example, suppose that the argument register A and the key register K have the bit configuration shown below. Only the three leftmost bits of

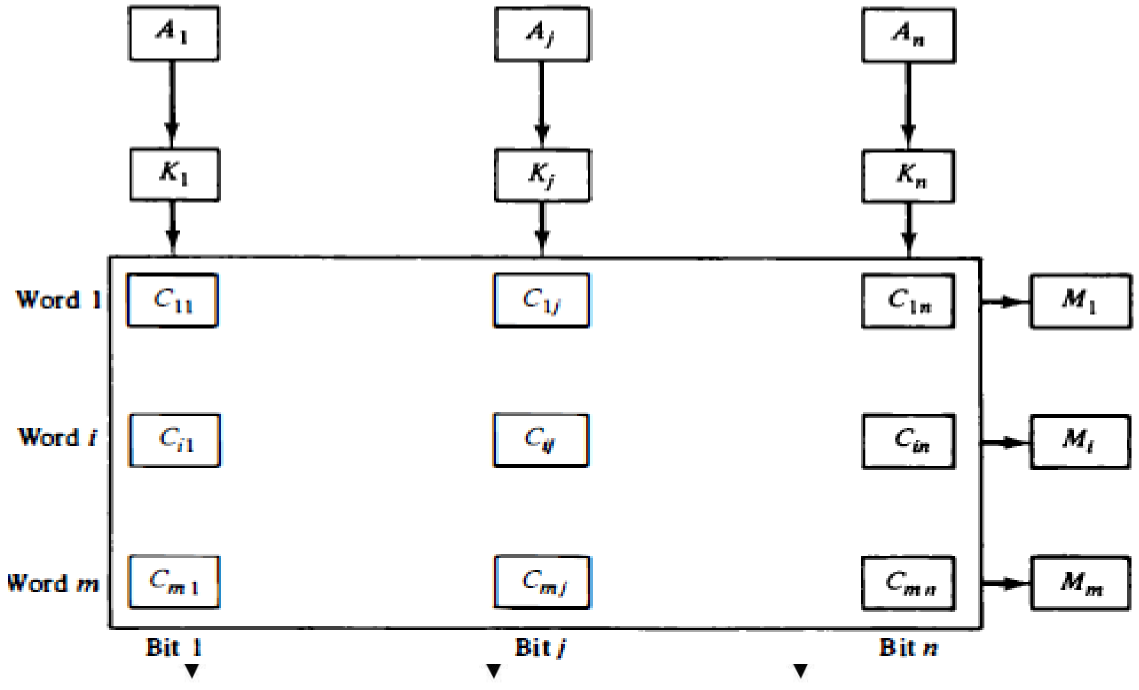
A are compared with memory words because K has 1's in these positions.



A	101 111100	
K	111 000000	
Word 1	100 111100	no match
Word 2	101 000001	match

Word 2 matches the unmasked argument field because the three leftmost bits of the argument and the word are equal.

The relation between the memory array and external registers in an associative memory is shown in below figure.



The cells in the array are marked by the letter C with two subscripts. The first subscript gives the word number and the second specifies the bit position in the word. Thus cell Cij is the cell for bit j in word i. A bit A j in the argument register is compared with all the bits in column j of the array provided that K j =
1. This is done for all columns j = 1, 2,…,n. If a match occurs between all the unmasked bits of the argument and the bits in word i, the corresponding bit Mi in the match register is set to 1.

If one or more unmasked bits of the argument and the word do not match, Mi is cleared to 0.

Flop storage element Fij and the circuits for reading, writing, and matching the cell. The input bit is transferred into the storage cell during a write operation. The bit stored is read out during a read operation. The match logic compares the content of the storage cell with the corresponding unmasked bit of the argument and provides an output for the decision logic that sets the bit in Mi.

_MATCH LOGIC

The match logic for each word can be derived from the comparison algorithm for two binary numbers. First, we neglect the key bits and compare the argument in A with the bits stored in the cells of the words. Word i is equal to the argument in A if Aj = Fij for j = 1, 2,…, n. Two bits are equal if they are both 1 or both 0. The equality of two bits can be expressed logically by the Boolean function

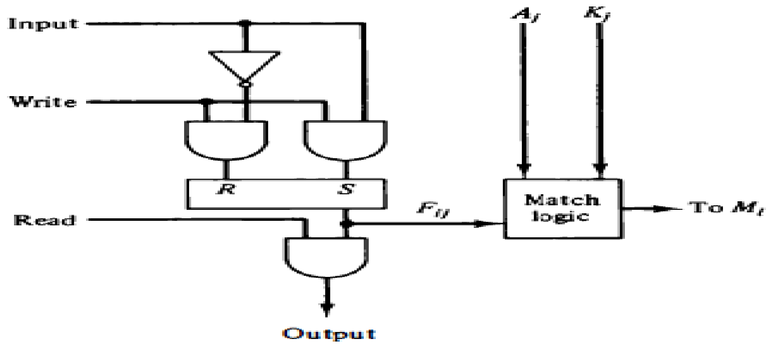
$x_j = A_j F_{ij} + A' F_{ij}'$

where xj = 1 if the pair of bits in position j are equal; otherwise, xj = 0.

For a word i to be equal to the argument in A we must have all xj variables equal to 1. This is the condition for setting the corresponding match bit Mi to 1. The Boolean function for this condition is

$M_i = x_1 \; x_2 \; x_3 \; \dots \; x_n$

and constitutes the AND operation of all pairs of matched bits in a word. One



cell of associative memory We now include the key bit Kj in the comparison logic. The requirement is that if Kj = 0, the corresponding bits of Aj and Fij need no comparison. Only when Kj = 1 must they be compared. This requirement is achieved by ORing each term with Kj' , thus:

$$x_j + K_j' = \begin{cases} x_j & \text{if } K_j = 1 \\ 1 & \text{if } K_j = 0 \end{cases}$$

When K j = 1, we have K j ' = 0 and xj + 0 = xj. When Kj = 0, then Kj' = 1 xj + 1 = 1. A term (xj + Kj') will be in the 1 state if its pair of bitsis not compared. This is necessary because each term is ANDed with all other terms so that an output of 1 will have no effect. The comparison of the bits has an effect only when K_j = 1. The match logic for word i in an associative memory can now be expressed by the following Boolean function:

$$\mathbf{M_i = (x_1 + K'_j)(x_2 + K'_j)(x_3 + K'_j) \ldots (x_n + K'_j)}$$

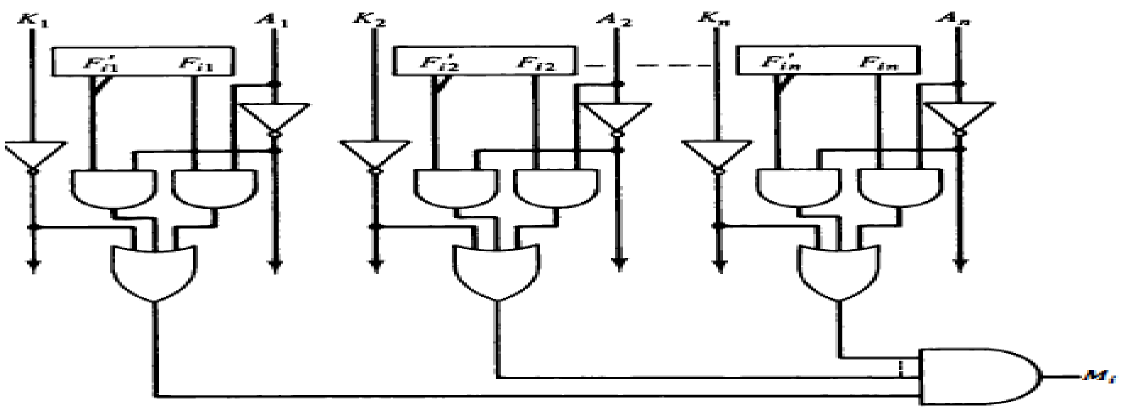
Each term in the expression will be equal to 1 if its corresponding $K'_j = 0$. if $K_j = 1$, the

$$M_i = \prod_{j=1}^{n} (A_j F_{ij} + A'_j F'_{ij} + K'_j)$$

term will be either 0 or 1 depending on the value of x_j. A match will occur and M_i will be equal to 1 if all terms are equal to 1. If we substitute the original definition of x_j. the Boolean function above can be expressed as follows:

Where ∏ is a product symbol designating the AND operation of all n terms. We need m such functions, one for each word i = 1, 2, 3, …., m.

The circuit for catching one word is shown in below figure.



Each cell requires two AND gates and one OR gate. The inverters for A_j and K_j are needed once for each column and are used for all bits in the column. The output of all OR gates in the cells of the same word go to the input of a common AND gate to generate the match signal for M_i. M_i will be logic 1 if a catch occurs and 0 if no match occurs. Note that if the key register contains all 0's, output M_i will be a 1 irrespective of the value of A or the word. This occurrence must be avoided during normal operation.

READ OPERATION

The matched words are read in sequence by applying a read signal to each word line whose corresponding Mi bit is a 1. In most applications, the associative memory stores a table with no two identical items under a given key. In this case, only one word may match the unmasked argument field. By connecting output Mi directly to the read line in the same word position (instead of the M register), the content of the matched word will be presented automatically at the output lines and no special read command signal is needed. Furthermore, if we exclude words having a zero content, an all-zero output will indicate that no match occurred and that the searched item is not available in memory.

WRITE OPERATION

If the entire memory is loaded with new information at once prior to a search operation then the writing can be done by addressing each location in sequence. This will make the device a random-access memory for writing and a content addressable memory for reading. The advantage here is that the address for input can be decoded as in a random-access memory. Thus instead of having m address lines, one for each word in memory, the number of address lines can be reduced by the decoder to d lines, where $m = 2^d$.

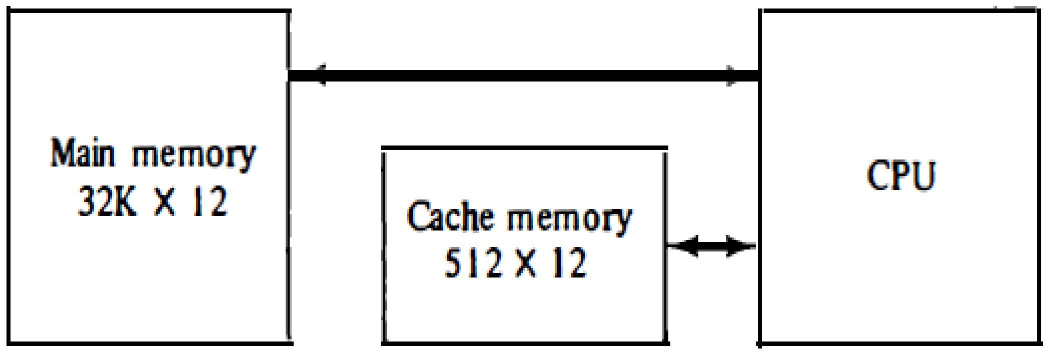
If unwanted words have to be deleted and new words inserted one at a time, there is a need for a special register to distinguish between active and inactive words. This register, sometimes called a tag register, would have as many bits as there are words in the memory. For every active word stored in memory, the corresponding bit in the tag register is set to 1. A word is deleted from memory by clearing its tag bit to 0. Words are stored in memory by scanning the tag register until the first 0 bit is encountered. This gives the first available inactive word and a position for writing a new word. After the new word is stored in memory it is made active by setting its tag bit to 1. An unwanted word when deleted from memory can be cleared to all 0's if this value is used to specify an empty location.

CACHE MEMORY

Analysis of a large number of typical programs has shown that the references, to memory at any given interval of time tend to be confined within a few localized areas in memory. The phenomenon is known as the **property of locality of reference.**

The locality of reference property, which states that over a short interval of time, the addresses generated by a typical program refer to a few localized areas of memory repeatedly, while the remainder of memory is accessed relatively frequently.

If the active portions of the program and data are placed in a fast small memory, the average memory access time can be reduced, thus reducing the total execution time



of the program. Such a fast small memory is referred to as **a cache memory.** It is placed between the CPU and main memory as illustrated in figure.

The cache memory access time is less than the access time of main memory by a factor of 5 to 10. The cache is the fastest component in the memory hierarchy and approaches the speed of CPU components.

The fundamental idea of cache organization is that by keeping the most frequently accessed instructions and data in the fast cache memory.

The basic operation of the cache is as follows. When the CPU needs to access memory, the cache is examined. If the word is found in the cache, it is read from the fast memory. If the word addressed by the CPU is not found in the cache, the main memory is accessed to read the word. A block of words containing the one just accessed is then transferred from main memory to cache memory.

The block size may vary from one word (the one just accessed) to about 16 words adjacent to the one just accessed. In this manner, some data are transferred to cache so that future references to memory find the required words in the fast cache memory.

The performance of cache memory is frequently measured in terms of a quantity **called hit ratio.** When the CPU refers to memory and finds the word in cache, it is said to produce **a hit.** If the word is not found in cache, it is in main memory and it counts as a **miss.** The ratio of the number of hits divided by the total CPU references to memory (hits plus misses) is the **hit ratio.**

For example, a computer with cache access time of 100 ns, a main memory access time of 1000 ns, and a hit ratio of 0.9 produces an average access time of 200 ns. This is a considerable improvement over a similar computer without a cache memory, whose access time is 1000 ns.

The basic characteristic of cache memory is its fast access time.

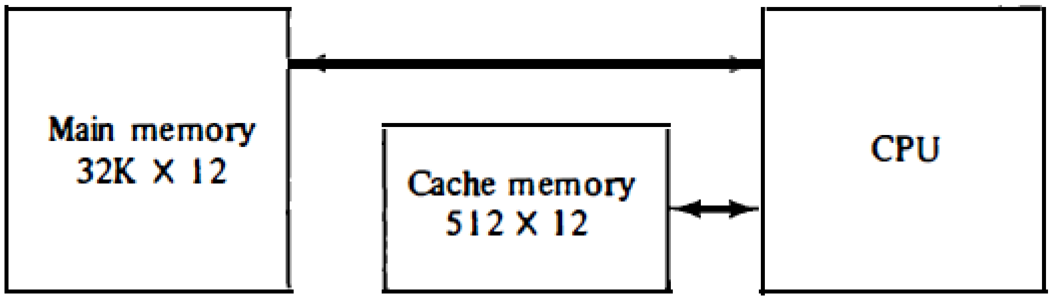
Therefore, very little or no time must be wasted when searching for words in the cache.

The transformation of data from main memory to cache memory is referred to as a **mapping process.**

Three types of mapping procedures are of practical interest when considering the organization of cache memory:

1. Associative mapping
2. Direct mapping
3. Set-associative mapping

To helping the discussion of these three mapping procedures we will use a specific



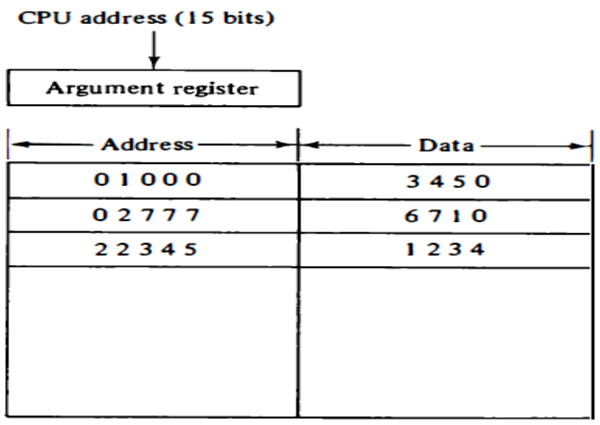
example of a memory organization as shown in figure.

The main memory can store 32K words of 12 bits each. The cache is capable of storing 512 of these words at any given time. For every word stored in cache, there is a duplicate copy in main memory.

The CPU communicates with both memories. It first sends a 15-bit address to cache. If there is a hit, the CPU accepts the 12 -bit data from cache. If there is a miss, the CPU reads the word from main memory and the word is then transferred to cache.

ASSOCIATIVE MAPPING

The fasters and most flexible cache organization uses an associative memory.



The associative memory stores both the address and content (data) of the memory word. This permits any location in cache to store any word from main memory. The diagram shows three words presently stored in the cache. The address value of 15 bits is shown as a five-digit octal number and its corresponding 12-bit word is shown as a four-digit octal number. A CPU address of 15 bits is placed in the argument register and the associative memory is searched for a matching address. If the address is found, the corresponding 12- bit data is read and sent to the CPU. If no match occurs, the main memory is accessed for the word. This constitutes a first-in first-out (FIFO) replacement policy.

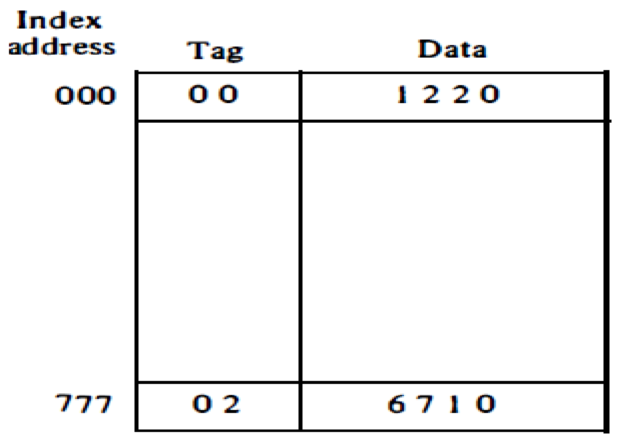
DIRECT MAPPING

Associative memories are expensive compared to random-access memories because of the added logic associated with each cell. The possibility of using a random-access memory for the cache is investigated in Fig.

The CPU address of 15 bits is divided into two fields. The nine least significant bits constitute the index field and the remaining six bits form the tag and the index bits. The number of bits in the index field is equal to the number of address bits required to access the cache memory.

In the general case, there are 2^k words in cache memory and 2^n words in main memory. The n-bit memory address is divided into two fields: k bits for the index field and n − k bits for the tag field. The direct mapping cache organization uses the n-bit address to access the main memory and the k-bit index to access the cache.

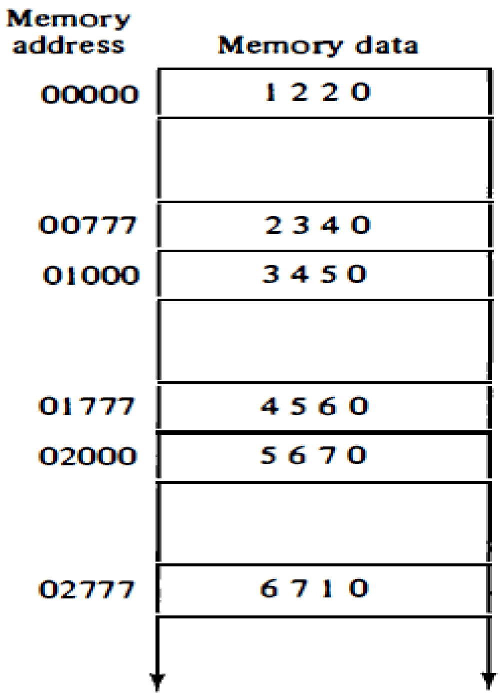
The internal organization of the words in the cache memory is as shown in Fig.



Each word in cache consists of the data word and its associated tag. When a new word is first brought into the cache, the tag bits are stored alongside the data bits. When the CPU generates a memory request, the index field is used for the address to access the cache.

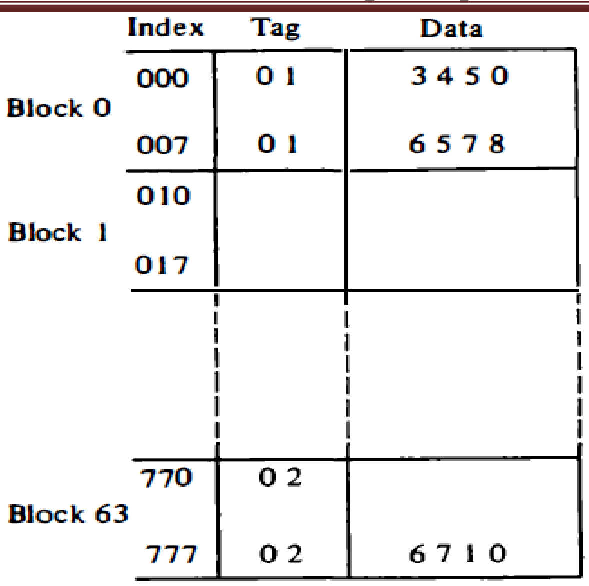
The tag field of the CPU address is compared with the tag in the word read from the cache. If the two tags match, there is a hit and the desired data word is in cache. If the two tags match, there is a hit and the desired data word is in cache. If there is no match, there is a miss and the required word is read from main memory.

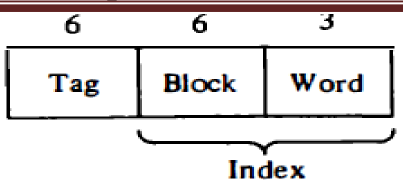
To see how the direct-mapping organization operates, consider the numerical example shown in Fig below.



The word at address zero is presently stored in the cache (index = 000, tag = 00, data = 1220). Suppose that the CPU now wants to access the word at address 02000. The index address is 000, so it is sued to access the cache. The two tags are then compared. The cache tag is 00 but the address tag is 02, which does not produce a match. Therefore, the main memory is accessed and the data word 5670 is transferred to the CPU. The cache word at index address 000 is then replaced with a tag of 02 and data of 5670.

The same organization but using a block size of 8 words is shown in Fig.





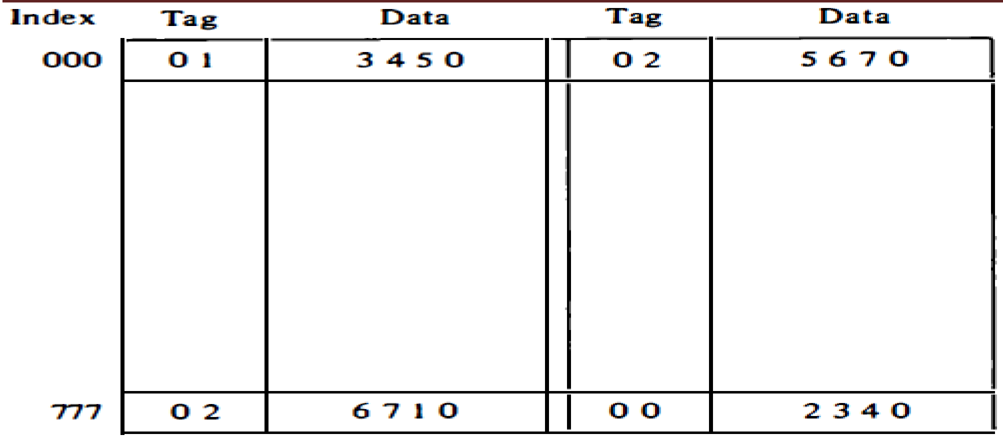
The index field is now divided into two parts: **the block field and the word field**. In a 512-word cache there are 64 block of 8 words each, since 64X8 = 512. The block number is specified with a 6-bit field and the word within the block is specified with a 3-bit field.

The tag field stored within the cache is common to all eight words of the same block. Every time a miss occurs, an entire block of eight words must be transferred from main memory to cache memory. Although this takes extra time, the hit ratio will most likely improve with a larger block size because of the sequential nature of computer programs.

SET-ASSOCIATIVE MAPPING

It was mentioned previously that the disadvantage of direct mapping is that two words with the same index in their address but with different tag values cannot reside in cache memory at the same time. A third type of cache organization, called **set-associative mapping**, is an improvement over the direct-mapping organization in that each word of cache can store two or more words of memory under the same index address. Each data word is stored together with its tag and the number of tag-data items in one word of cache is said to form a set.

An example of a set-associative cache organization for a set size of two is shown in Fig.



Index	Tag	Data	Tag	Data
000	0 1	3 4 5 0	0 2	5 6 7 0
777	0 2	6 7 1 0	0 0	2 3 4 0

Each index address refers to two data words and their associated tags. Each tag requires six bits and each data word has 12 bits, so the word length is 2(6 + 12) = 36 bits. An index address of nine bits can accommodate 512 words. Thus the size of cache memory is 512 X 36.

It can accommodate 1024 words of main memory since each word of cache contains two data words. In general, a set-associative cache of set size k will accommodate k words of main memory in each word of cache.

With reference to the main memory content the following is illustrated. The words stored at addresses 01000 and 02000 of main memory are stored in cache memory at index address 000. Similarly, the words at addresses 02777 and 00777 are stored in cache at index address 777.

When a miss occurs in a set-associative cache and the set is full, it is necessary to replace one of the tag-data items with a new value. The most common replacement algorithms used are: random replacement, first-in, first out (FIFO), and least recently used (LRU). With the random replacement policy the control chooses one tag-data item for replacement at random. The FIFO procedure selects for replacement the item that has been in the set the longest. The LRU algorithm selects for replacement the item that has been least recently used by the CPU. Both FIFO and LRU can be implemented by adding a few extra bits in each word of cache.

WRITING INTO CACHE

For write operation, there are two ways that the system can proceed. The simplest and most commonly used procedure is to up data main memory with every memory write operation, with cache memory being updated in parallel if it contains the word at the specified address. This is called the **write-through method.**

The second procedure is called the **write-back method.** In this method only the cache location is updated during a write operation. The location is then marked by a flag so that later when the words are removed from the cache it is copied into main memory.

CACHE INITIALIZATION

One more aspect of cache organization that must be taken into consideration is the problem of initialization. The cache is initialized when power is applied to the computer or when the main memory is loaded with a complete set of programs from auxiliary memory. After initialization the cache is considered to be empty, built in effect it contains some non-valid data. It is customary to include with each word in cache a valid bit to indicate whether or not the word contains valid data. The cache is initialized by clearing all the valid bits to 0. The valid bit of a particular cache word is set to 1 the first time this word is loaded from main memory and stays set unless the cache has to be initialized again.