(23CS304) COMPUTER ORGANIZATION & ARCHITECTURE

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, Cashe Memory.

Input-Output organization

1. Peripheral Devices

The input-output subsystem of a computer, referred to as I/O, provides an efficient mode of communication between the central system and the outside environment. Programs and data must be entered into computer memory for processing and results obtained from computations must be recorded or displayed for the user.

The most familiar means of entering information into a computer is through a typewriter-like keyboard that allows a person to enter alphanumeric information directly. Every time a key is depressed, the terminal sends a binary coded character to the computer. The fastest possible speed for entering information this way depends on the person's typing speed. On the other hand, the central processing unit is an extremely fast device capable of performing operations at very high speed.

When input information is transferred to the processor via a slow keyboard, the processor will be idle most of the time while waiting for the information to arrive. To use a computer efficiently, a large amount of programs and data must be prepared in advance and transmitted into a storage medium such as magnetic tapes or disks. The information in the disk is then transferred into computer memory at a rapid rate. Results of programs are also transferred into a high-speed storage, such as disks, from which they can be transferred later into a printer to provide a printed output of results.

Devices that are under the direct control of the computer are said to be connected on-line. These devices are designed to read information into or out of the memory unit upon command from the CPU and are considered to be part of the total computer system.

Input or output devices attached to the computer are also called peripherals. Among the most common peripherals are keyboards, display units, and printers. Peripherals that provide auxiliary storage for the system are magnetic disks and tapes. Peripherals are electro mechanical and electromagnetic devices of some complexity.

There are different types of video monitors, but the most popular use a cathode ray tube (CRT). It contains an electronic gun that sends an electronic beam to a phosphorescent screen in front of the tube. The beam can be reflected horizontally and vertically. To produce a pattern on the screen, a grid inside the CRT receives a variable voltage that causes the beam to hit the screen and make it glow at selected spots.

A characteristic feature of display devices is a cursor that marks the position in the screen where the next character will be inserted. The cursor can be moved to any position in the screen, to a single character, the beginning of a word, or to any line. Edit keys add or delete information based on the cursor position. The display terminal can operate in a Single-character mode where all characters entered on the screen through the keyboard are transmitted to the computer simultaneously.

Printers provide a permanent record on paper of computer output data or text. There are three basic types of character printers: *daisywheel, dot matrix,* and *laser printers*.

The daisywheel printer contains a wheel with the characters placed along the circumference. To print a character, the wheel rotates to the proper position and an energized magnet then presses

the letter against the ribbon.

The dot matrix printer contains a set of dots along the printing mechanism. For example, a 5 x 7 dot matrix printer that prints 80characters per line has seven horizontal lines, each consisting of 5 x 80 = 400dots. Each dot can be printed or not, depending on the specific characters that are printed on the line.

The laser printer uses a rotating photographic drum that is used to imprint the character images. The pattern is then transferred onto paper in the same manner as a copying machine.

Magnetic disks have high-speed rotational surfaces coated with magnetic material. Access is achieved by moving a read-write mechanism to a track in the magnetized surface. Disks are used mostly for bulk storage of programs and data.

Other input and output devices encountered in computer systems are digital incremental plotters, optical and magnetic character readers, analog-to-digital converters, and various data acquisition equipment.

The input-output organization of a computer is a function of the size of the computer and the devices connected to it. The difference between a small and a large system is mostly dependent on the amount of hardware the computer has available for communicating with peripheral units and the number of peripherals connected to the system. Since each peripheral behaves differently from any other, it would be prohibitive to dwell on the detailed interconnections needed between the computer and each peripheral.

ASCII Alphanumeric Characters

Input and output devices that communicate with people and the computer are usually involved in the transfer of alphanumeric information to and from the device and the computer. The standard binary code for the alphanumeric characters is ASCII (American Standard Code for Information Interchange). It uses seven bits to code 128 characters as shown in below Table.

The seven bits of the code are designated by b, through b-, with b, being the most significant bit. The letter A, for example, is represented in ASCII as 1000001(column 100, row 0001).

	$b_7b_6b_5$							
$b_4b_3b_2b_1$	000	001	010	011	100	101	110	111
0000	NUL	DLE	SP	0	@	P	‘	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	"	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	'	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	\	l	|
1101	CR	GS	−	=	M	]	m	}
1110	SO	RS	.	>	N	^	n	~
1111	SI	US	/	?	O	—	o	DEL

Control characters

NUL	Null	DLE	Data link escape
SOH	Start of heading	DC1	Device control 1
STX	Start of text	DC2	Device control 2
ETX	End of text	DC3	Device control 3
EOT	End of transmission	DC4	Device control 4
ENQ	Enquiry	NAK	Negative acknowledge
ACK	Acknowledge	SYN	Synchronous idle
BEL	Bell	ETB	End of transmission block
BS	Backspace	CAN	Cancel
HT	Horizontal tab	EM	End of medium
LF	Line feed	SUB	Substitute
VT	Vertical tab	ESC	Escape
FF	Form feed	FS	File separator
CR	Carriage return	GS	Group separator
SO	Shift out	RS	Record separator
SI	Shift in	US	Unit separator
SP	Space	DEL	Delete

Table: ASCII codes

ASCII is a 7-bit code, but most computers manipulate an 8-bit quantity as a single unit called a byte. Therefore, ASCII characters most often are store done per byte. The extra bit is sometimes used for other purposes, depending on the application. Additional 128 8-bit characters

with the most significant bit set to 1 are used for other symbols, such as the Greek alphabet or italic type font.

When used in data communication, the eighth bit may be employed to indicate the parity of the binary-coded character.

2. Input-Output interface

Input-output interface provides a method for transferring information between internal storage and external I/O devices. Peripherals connected to 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 maybe needed.
3. Data codes and formats in peripherals differ from the word format in the CPU and memory. .
4. The operating modes of peripherals are different from each other and each must be controlled so as not to disturb the operation of other peripherals connected to the CPU.

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

I/O Bus and Interface Modules:

A typical communication link between the processor and several peripherals is shown in below figure. The 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. It also synchronizes the data flow and supervises the transfer between peripheral and processor. Each peripheral has its own controller that operates the particular electromechanical device. A controller may be housed separately or may be physically integrated with the peripheral.

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. When the interface detects its own address, it activates the path between the bus lines and the device that it controls. All peripherals whose address does not correspond to the address in the bus are disabled by their interface.

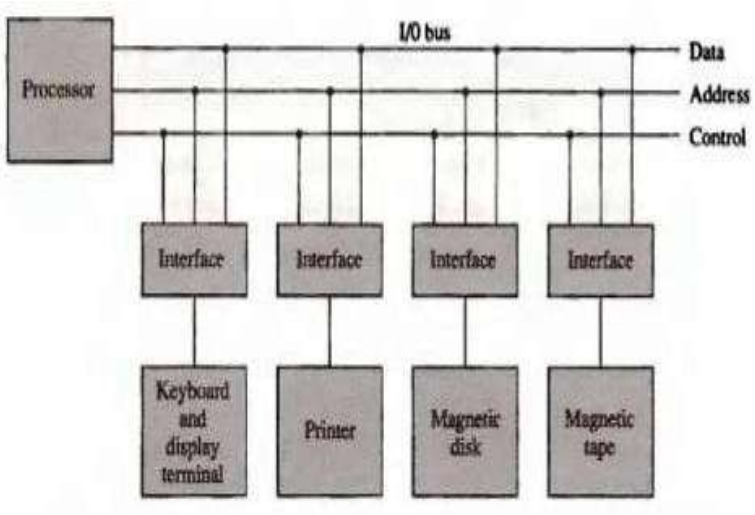


Fig: Connection of I/O bus to Input-Output devices.

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. The interpretation of the command depends on the peripheral that the processor is addressing.

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

C*ontrol 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. For example, the computer may wish to check the status of the peripheral before a transfer is initiated. During the transfer, one or more errors may occur which are detected by the interface. These errors are designated by setting bits in a status register that the processor can read at certain intervals.

A ***data output*** command causes the interface to respond by transferring data from the bus into one of its registers. Consider an example with a tape unit. The computer starts the tape moving by issuing a control command. The processor then monitors the status of the tape by means of a status command. When the tape is in the correct position, the processor issues a data output command. The interface responds to the address and command and transfers the information from the data lines in the bus to its buffer register. The interface then communicates with the tape controller and sends the data to be stored on tape. .

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. The processor checks if data are available by means of a status command and then issues a data input command. The interface places the data on the data lines, where they are accepted by the processor.

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 IOP in computers that provide a

separate I/O processor (IOP) in addition to the central processing unit (CPU). The memory communicates with both the CPU and the IOP through a memory bus. The IOP communicates also with the input and output devices through a separate I/O bus with its own address, data and control lines. The purpose of the IOP is to provide an independent pathway for the transfer of information between external devices and internal memory.
The I/O processor is sometimes called a data channel. In Sec. 11-7 we discuss the function of the IOP in more detail.

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. The memory read and memory write control lines are enabled during a memory transfer. This configuration isolates all I/O interface addresses from the addresses assigned to memory and is referred to as the isolated I/O method for assigning addresses in a common bus.

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. This informs the external components that are attached to the common bus that the address in the address lines is for an interface register and not for a memory word. On the other hand, when the CPU is fetching an instruction or an operand from memory, it places the memory address on the address lines and enables the memory read or memory write control line. This informs the external components that the address is for a memory word and not for an I/O interface.

The isolated I/O method isolates memory and 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. The computer treats an interface register as being part of the memory system. The assigned addresses for interface registers cannot be used for memory a word, which reduces the memory address range available.

In a memory-mapped I/O organization there is 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. Typically, a segment of the total address space is reserved for interface registers, but in general, they can be located at any address as long as there is not also a memory word that responds to the same address.

Computers with memory-mapped I/O can use memory-type instructions to access I/O data. It allows the computer to use the same instructions for either input-output transfers or for memory transfers. The advantage is that the load and store instructions used for reading and writing from memory can be used to input and output data from I/O registers. In a typical computer, there are more memory-reference instructions than I/O instructions. With memory mapped I/O all instructions that refer to memory are also available for I/O.

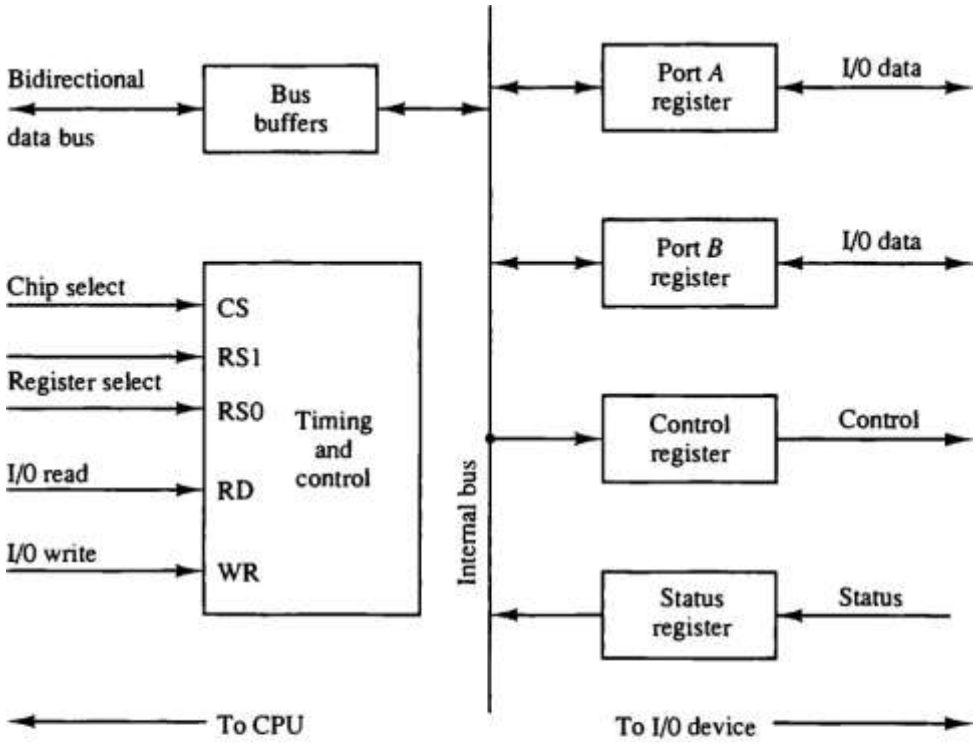
Example of I/O Interface:

An example of an I/O interface unit is shown in below as block diagram. 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 interface register with which the CPU communicates.

The control register receives control information from the CPU. By loading appropriate bits into the control register, the interface and the I/O device attached to it can be placed in a variety of operating modes. For example, port A may be defined as an input port and port B as an output port. A magnetic tape unit may be instructed to rewind the tape or to start the tape moving in the forward direction. The bits in the status register are used for status conditions and for recording errors that may occur during the data transfer. For example, a status bit may indicate that port A has received a

new data item from the I/O device. Another bit in the status register may indicate that a parity error has occurred during the 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

Fig: Example of I/O Interface Unit.

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 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.

3. Asynchronous Data Transfer

The internal operations in a digital system are synchronized by means of clock pulses supplied by a common pulse generator. Clock pulses are applied to all registers within a unit and all data transfers among internal registers occur simultaneously during the occurrence of a clock pulse. Two units, such as a CPU and an I/O interface, are designed independently of each other. If the registers in the interface share a common clock with the CPU registers, the transfer between the two units is said to be synchronous. In most cases, the internal timing in each unit is independent from the other in that each uses its own private clock for internal registers. In that case, the two units are said to be asynchronous to each other. This approach is widely used in most computer systems.

Asynchronous data transfer between two independent units requires that control signals be transmitted between the communicating units to indicate the time at which data is being transmitted.

One way of achieving this is by means of a strobe pulse supplied by one of the units to indicate to the other unit when the transfer has to occur.

Another method commonly used is to accompany each data item being transferred with a control signal that indicates the presence of data in the bus. The unit receiving the data item responds with another control signal to acknowledge receipt of the data. This type of agreement between two independent units is referred to as handshaking.

Strobe Control:

The strobe control method of asynchronous data transfer employs a single control line to time each transfer. The strobe may be activated by either the source or the destination unit. The below figure shown a source-initiated transfer.

The data bus carries the binary information from source unit to the destination unit. Typically, the bus has multiple lines to transfer an entire byte or word.

The strobe is a single line that informs the destination unit when a valid data word is available in the bus.

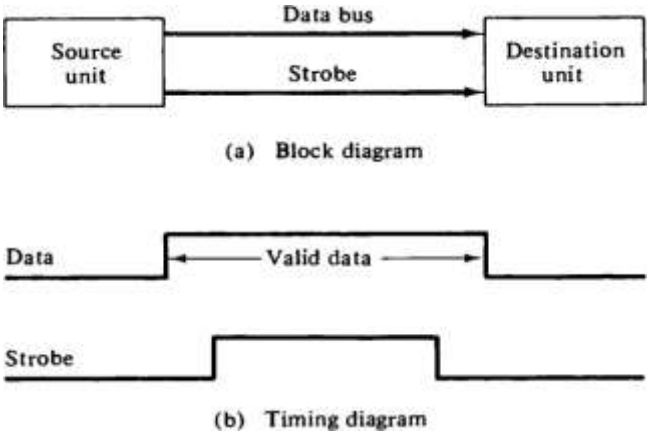


Fig: Source – initiated strobe for data transfer

The below figure shown a data transfer initiated by the destination unit. In this case the

destination unit activates the strobe pulse, informing the source to provide the data. The source unit responds by placing the requested binary information on the data bus. The data must be valid and remain in the bus long enough for the destination unit to accept it. The falling edge of the strobe pulse can be used again to trigger a destination register. The destination unit then disables the strobe. The source removes the data from the bus after a predetermined time interval.

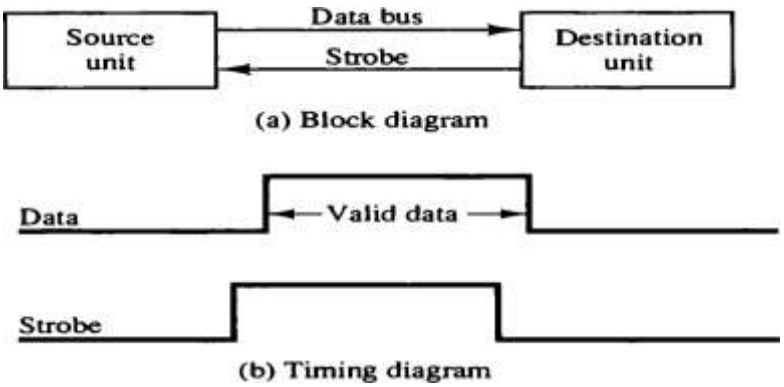


Fig: Destination – initiated strobe for data transfer

Dis-advantage:

In the strobe method is that the source unit that initiates the transfer has no way of knowing whether the destination unit has actually received the data item that was placed in the bus.

Similarly, a destination unit that initiates the transfer has no way of knowing whether the source unit has actually placed the data on the bus.

Handshaking:

The handshake method solves this problem by introducing a second control signal that provides a reply to the unit that initiates the transfer. The basic principle of the two-wire handshaking method of data transfer is as follows. One control line is in the same direction as the data flow in the bus from the source to the destination. It is used by the source unit to inform the destination unit whether there are valid data in the bus. The other control line is in the other direction from the destination to the source.

It is used by the destination unit to inform the source whether it can accept data. The sequence of control during the transfer depends on the unit that initiates the transfer.

The below figure shown the data transfer procedure when initiated by the source. The two and shaking lines are data valid, which is generated by the source unit, and data accepted, generated by the destination unit. The timing diagram shows the exchange of signals between the two units. The sequence of events listed in part (c) shows the four possible states that the system can be at any given time.

This scheme allows arbitrary delays from one state to the next and permits each unit to respond at its own data transfer rate. The rate of transfer is determined by the slowest unit.

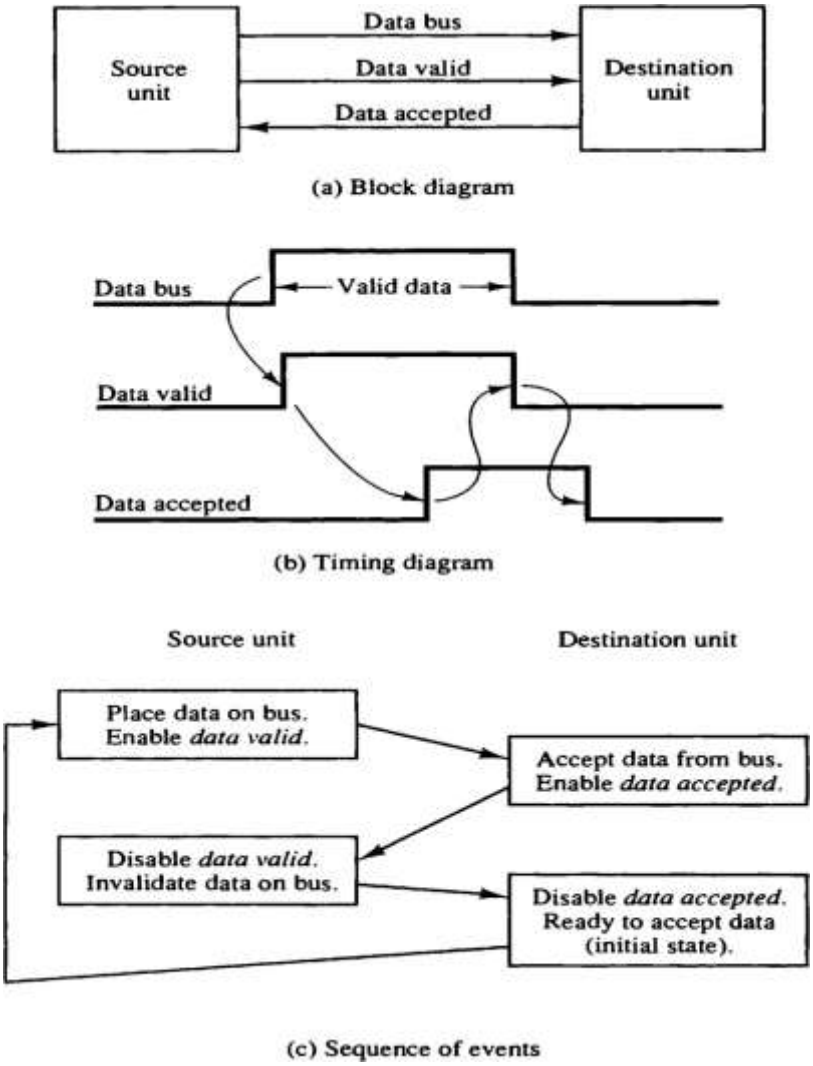


Fig: Source – initiated data transfer by using handshaking

The destination-initiated transfer using handshaking lines is shown in below figure. Note that the name of the signal generated by the destination unit has been changed to ready for data to reflect its new meaning. The source unit in this case does not place data on the bus until after it receives the ready for data signal from the destination unit. From there on, the handshaking

procedure follows the same pattern as in the source-initiated case. Note that the sequence of events in both cases would be identical if we consider the ready for data signal as the complement of data accepted. In fact, the only difference between the source-initiated and the destination-initiated transfer is in their choice of initial state.

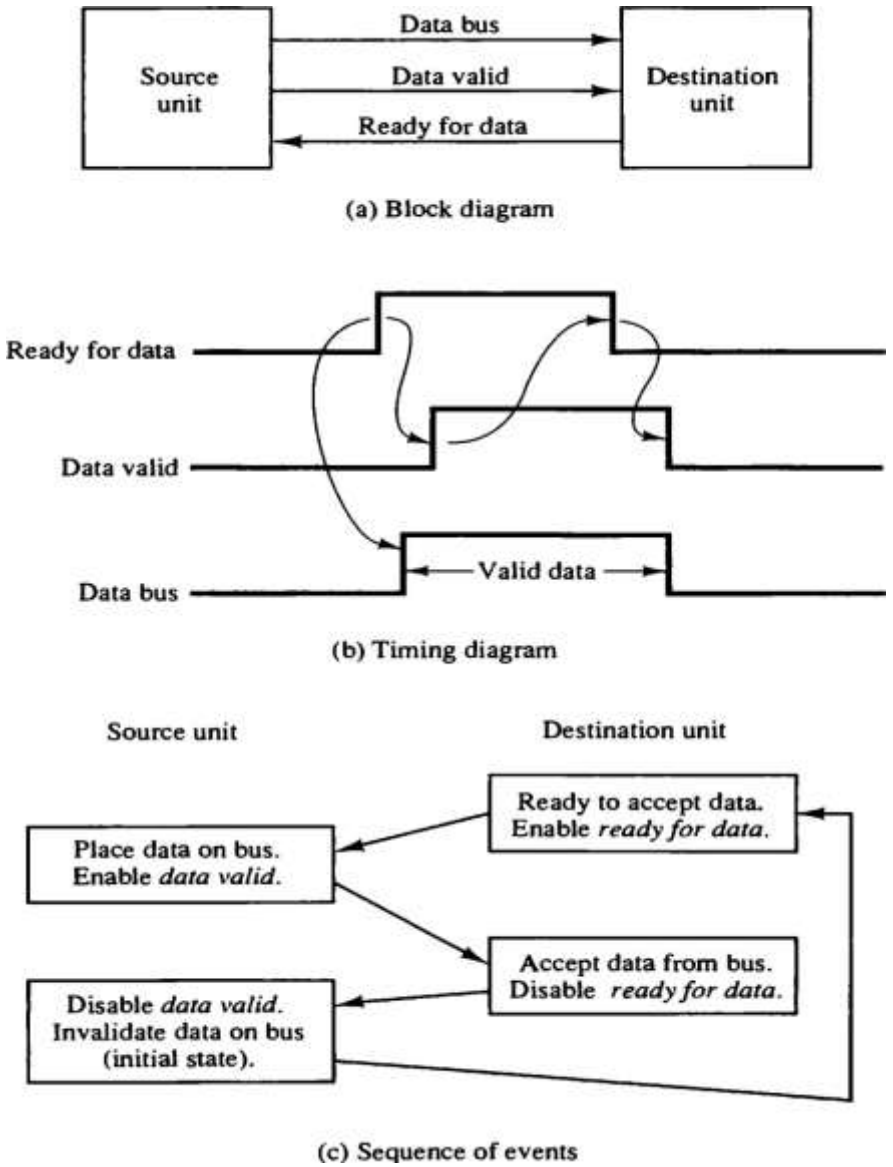


Fig: Destination – initiated data transfer by using handshaking

The handshaking scheme provides a high degree of flexibility and reliability because the successful completion of a data transfer relies on active participation by both units. If one unit is faulty, the data transfer will not be timeout completed. Such an error can be detected by means of a timeout mechanism, which produces an alarm if the data transfer is not completed within a predetermined time. The timeout is implemented by means of an internal clock that starts counting time when the unit enables one of its handshaking control signals. If the return handshake signal does not respond within a given time period, the unit assumes that an error has occurred. The timeout Signal can be used to interrupt the processor and hence execute a service routine that takes appropriate error recovery action.

4. Modes of Transfer

Binary information received from an external device is usually stored in memory for later processing. Information transferred from the central computer into an external device originates in the memory unit. The CPU merely executes the I/O instructions and may accept the data temporarily, but the ultimate source or destination is the memory unit. Data transfer between the central computer and I/O devices may be handled in a variety of modes.

Some modes use the CPU as an intermediate path; others transfer the data directly to and from the memory unit. Data transfer to and from peripherals may be handled in one of three possible modes:

1. Programmed I/O
2. Interrupt-initiated I/O
3. Direct memory access (DMA)

Programmed I/O:

In the programmed I/O method, the I/O device does 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. Other instructions may be needed to verify that the data are available from the device and to count the numbers of words transferred.

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

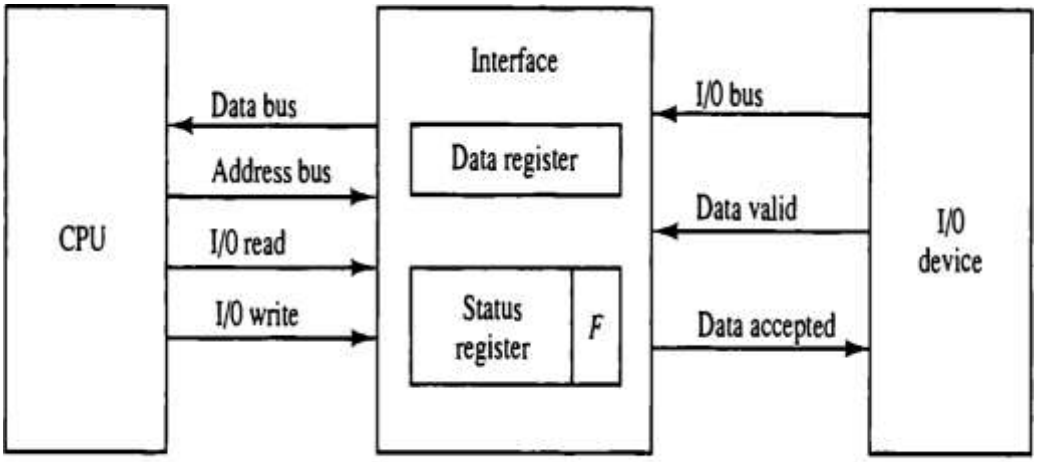


Fig: Data transfer from I/O device to CPU

The interface accepts the byte into its data register and enables the data accepted line. The interface sets a bit 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. This is according to the handshaking procedure established in below figure.

A flowchart of the program that must be written for the CPU is shown in below figure. 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 step3 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 from an I/O device and store them in a memory buffer.

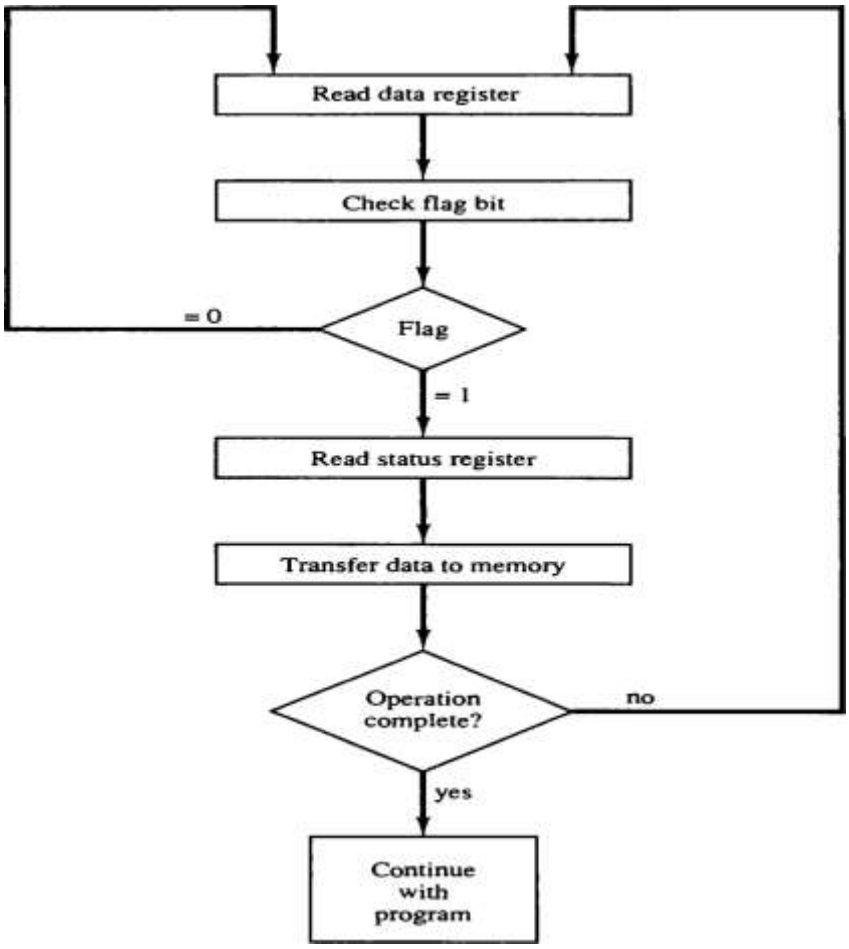


Fig: Flowchart for CPU program to input data

The programmed I/O method is particularly useful in small low-speed computers or in systems that are dedicated to monitor a device continuously. The difference in information transfer rate between the CPU and the I/O device makes this type of transfer inefficient.

Interrupt-Initiated I/O:

An alternative to the CPU constantly monitoring the flag is to let the interface inform the computer when it is ready to transfer data. This mode of transfer uses the interrupt facility. While the CPU is running a program, it does not check the flag. However, when the flag is set, the computer is momentarily interrupted from proceeding with the current program and is informed of the fact that the flag has been set. The CPU deviates from what it is doing to take care of the input or output transfer. After the transfer is completed, the computer returns to the previous program to continue what it was doing before the interrupt.

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 one unit to another. In principle, there are two methods for accomplishing this.

One is called vectored interrupt and the other, non vectored interrupt. Ina 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. In some computers the interrupt vector is the first address of the I/O service routine.

5. 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. The readiness of the device can be determined from an interrupt signal. The CPU responds to the interrupt request by storing the return address from PC into a memory stack and then the program branches to a service routine that processes the required transfer. Some processors also push the current PSW (program status word) onto the stack and load a new PSW for the service routine. We neglect the PSW here in order not to complicate the discussion of I/O interrupts.

In a typical application a number of I/O devices are attached to the computer, with each device being able to originate an interrupt request. The first task of the interrupt system is to identify the source of the interrupt. There is also the possibility that several sources will request service simultaneously. In this case the system must also decide which device to service first.

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

Establishing the priority of simultaneous interrupts can be done by software or hardware. A *polling procedure* is used to identify the highest-priority source by software means. In this method there is one common branch address for all interrupts. The program that takes care of interrupts begins at the branch address and polls the interrupt sources in sequence. The order in which they are tested determines the priority of each interrupt. The highest-priority source is tested first, and if its interrupt signal is on, control branches to a service routine for this source. Otherwise, the next-lower-priority source is tested, and priority interrupt so on. Thus the initial service routine for all interrupts consists of a program that tests the interrupt sources in sequence and branches to one of many possible service routines. The particular service routine reached belongs to the highest-priority device among all devices that interrupted the computer. The *disadvantage of the software 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 below figure. 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 is equivalent to a negative logic OR operation. The CPU responds to an interrupt request by enabling the interrupt acknowledge line. 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 1has 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) 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.

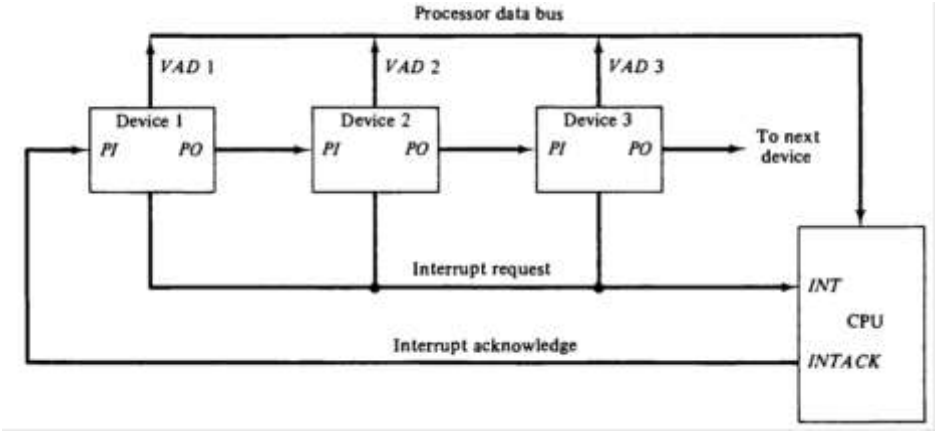


Fig: Daisy-chain priority interrupts

The below figure shown the internal logic that must be included within 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.

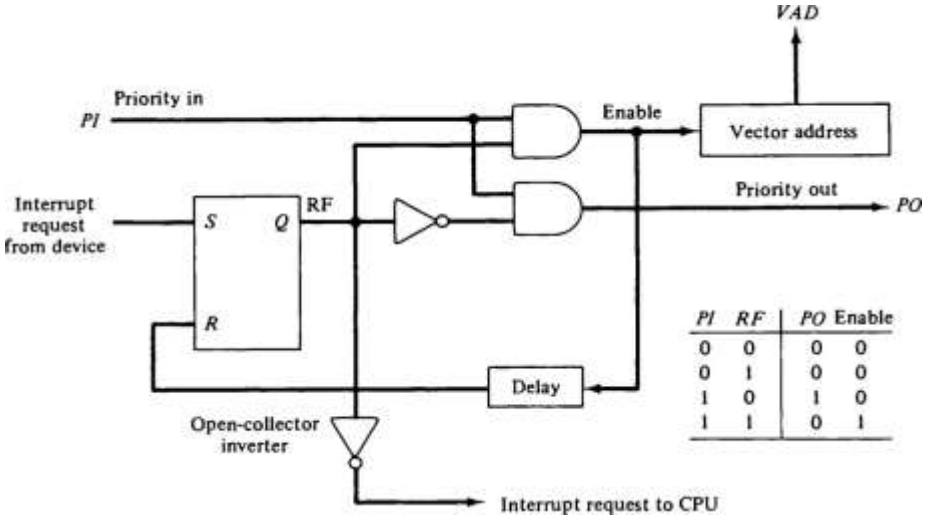


Fig: One stage of the daisy-chain priority arrangement

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 below figure. 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. Each interrupt bit and its corresponding mask bit are applied to an AND gate to produce the four inputs to a priority encoder. In this way an interrupt is recognized only if its corresponding mask bit is set to 1 by the program. The priority encoder generates two bits of the vector address, which is transferred to the CPU.

Another output from the encoder sets an interrupt status flip-flop IST when an interrupt that is not masked occurs. The interrupt enable flip-flop IEN can be set or cleared by the program to provide an overall control over the interrupt system. The outputs of IST ANDed with IEN provide a common interrupt signal for the CPU. The interrupt acknowledge INIACK signal from the CPU enables the bus buffers in the output register and a vector address VAD is placed into the data bus. We will now explain the priority encoder circuit and then discuss the interaction between the priority interrupt controller and the CPU.

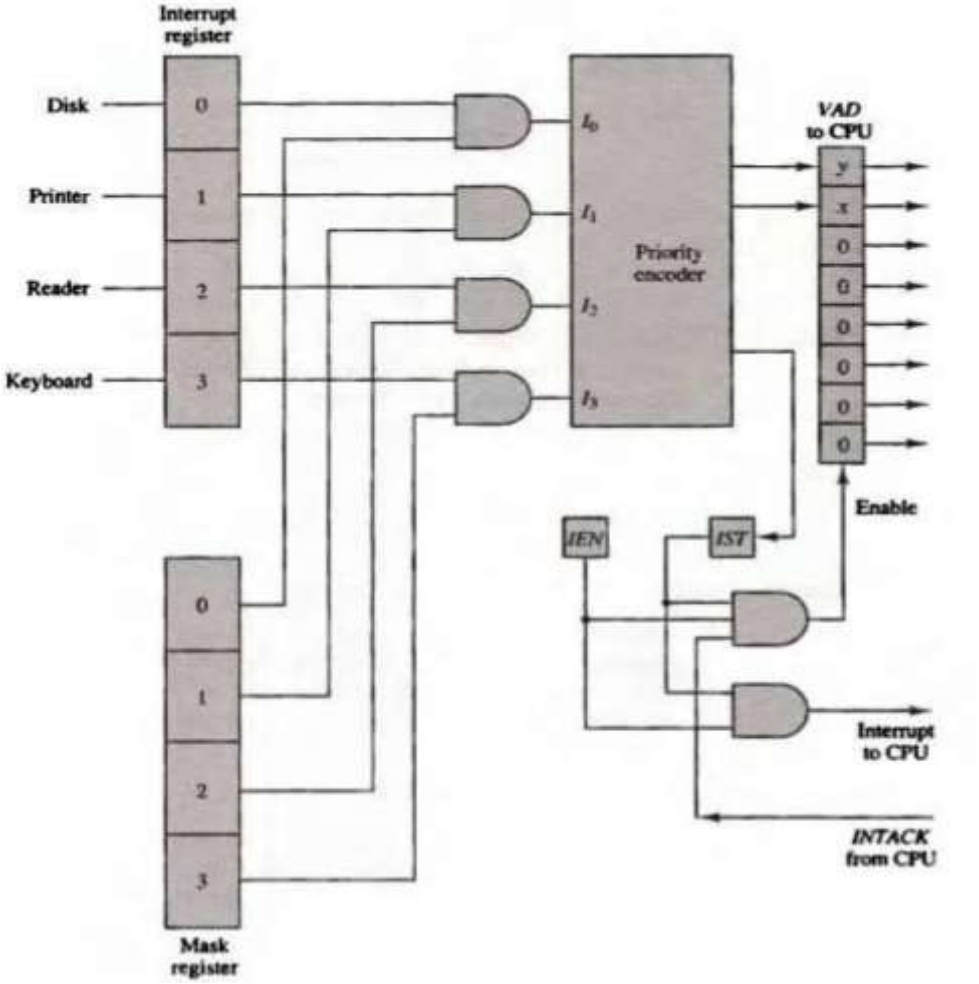


Fig: Priority interrupts Hardware

Priority Encoder:

The priority encoder is a circuit that implements the priority function. The logic of the priority encoder is such that if two or more inputs arrive at the same time, the input having the highest priority will take precedence. The truth table of a four-input priority encoder is given in below table. The x's in the table designate don't-care conditions. Input 10 has the highest priority; so regardless of the values of other inputs, when this input is I, the output generates an output xy = 00. II has the next priority level. The output is 01 if $I_1 = 1$ provided that $1_0 = 0$, regardless of the values of the other two lower-priority inputs. The output for 12 is generated only if higher-priority inputs are 0, and so on down the priority level. The interrupt status IST is set only when one or more inputs are equal to 1. If all inputs are 0, IST is cleared to 0 and the other outputs of the encoder are not used, so they are marked with don't-care conditions. This is because the vector address is not transferred to the CPU when IST = 0. The Boolean functions listed in the table specify the internal logic of the encoder. Usually, a computer will have more than four interrupt sources. A priority encoder with eight inputs, for example, will generate an output of three bits.

Inputs				Outputs			
I_0	I_1	I_2	I_3	x	y	IST	Boolean functions
1	×	×	×	0	0	1	
0	1	×	×	0	1	1	$x = I_0' I_1'$
0	0	1	×	1	0	1	$y = I_0' I_1 + I_0' I_2'$
0	0	0	1	1	1	1	$(IST) = I_0 + I_1 + I_2 + I_3$
0	0	0	0	×	×	0	

Table: Priority encoder truth table

The output of the priority encoder is used to form part of the vector address for each interrupt source. The other bits of the vector address can be assigned any value. For example, the vector address can be formed by appending six zeros to the x and y outputs of the encoder. With this choice the interrupt vectors for the four I/O devices are assigned binary numbers 0, 1, 2, and 3.

Interrupt Cycle:

The interrupt enable flip-flop IEN shown in above figure can be set or cleared by program instructions. When IEN is cleared, the interrupt request coming from IST is neglected by the CPU. The program-controlled IEN bit allows the programmer to choose whether to use the interrupt facility. If an instruction to clear IEN has been inserted in the program, it means that the user does not want his program to be interrupted. An instruction to set IEN indicates that the interrupt facility will be used while the current program is running. Most computers include internal hardware that clears IEN to 0 every time an interrupt is acknowledged by the processor.

At the end of each instruction cycle the CPU checks IEN and the interrupt signal from IST. If either is equal to 0, control continues with the next instruction. If both IEN and IST are equal to 1, the CPU goes to an interrupt cycle. During the interrupt cycle the CPU performs the following sequence of micro operations:

$SP \leftarrow SP - 1$	Decrement stack pointer
$M[SP] \leftarrow PC$	Push PC into stack
$INTACK \leftarrow 1$	Enable interrupt acknowledge
$PC \leftarrow VAD$	Transfer vector address to PC
$IEN \leftarrow 0$	Disable further interrupts
Go to fetch next instruction	

The CPU pushes the return address from PC into the stack. It then acknowledges the interrupt by enabling the INTACK line. The priority interrupt unit responds by placing a unique interrupt vector into the CPU data bus. The CPU transfers the vector address into PC and clears IEN prior to going to the next fetch phase. The instruction read from memory during the next fetch phase will be the one located at the vector address.

Software Routines:

A priority interrupt system is a combination of hardware and software techniques. So far we have discussed the hardware aspects of a priority interrupt system. The computer must also have software routines for servicing the interrupt requests and for controlling the interrupt hardware registers. The figure shows the programs that must reside in memory for handling the interrupt system. Each device has its own service program that can be reached through a jump (JMP) instruction stored at the assigned vector address. The symbolic name of each routine represents the starting address of the service program. The stack shown in the diagram is used for storing the return address after each interrupt.

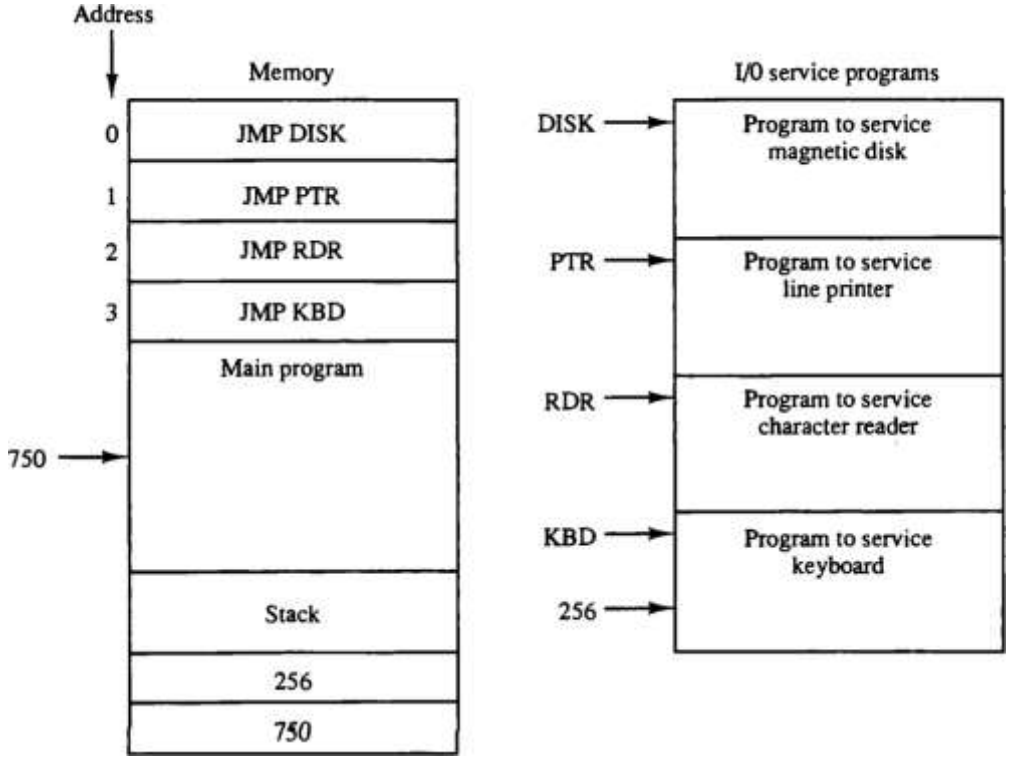


Fig: Program stored in memory for servicing interrupts

To illustrate with a specific example assume that the keyboard sets its interrupt bit while the CPU is executing the instruction in location 749 of the main program. At the end of the instruction cycle, the computer goes to an interrupt cycle. It stores the return address 750 in the stack and then accepts the vector address 00000011from the bus and transfers it to PC. The instruction in location 3 is executed next, resulting in transfer of control to the KBD routine. Now suppose that the disk sets its interrupt bit when the CPU is executing the instruction at address 255 in the KBD program. Address 256 is pushed into the stack and control is transferred to the DISK service program. The last instruction in each routine is a return from interrupt instruction. When the disk service program is completed, the return instruction pops the stack and places 256 into PC. This returns control to the KBD routine to continue servicing the keyboard. At the end of the KBD program T the last instruction pops the stack and returns control to the main program at address 750. Thus, a higher-priority device can interrupt a lower-priority device. It is assumed that the time spent in servicing the high-priority interrupt is short compared to the transfer rate of the low-priority device so that no loss of information takes place.

Initial and Final Operations:

Each interrupt service routine must have an initial and final set of operations for controlling the registers in the hardware interrupt system. Remember that the interrupt enable IEN is cleared at the end of an interrupt cycle. This flip-flop must be set again to enable higher-priority interrupt requests, but not before lower-priority interrupts are disabled. The initial sequence of each interrupt service routine must have instructions to control the interrupt hardware in the following manner:

1. Clear lower-level mask register bits.
2. Clear interrupt status bit IST.

3. Save contents of processor registers.
4. Set interrupt enable bit IEN.
5. Proceed with service routine.

The lower-level mask register bits (including the bit of the source that interrupted) are cleared to prevent these conditions from enabling the interrupt. Although lower-priority interrupt sources are assigned to higher-numbered bits in the mask register, priority can be changed if desired since the programmer can use any bit configuration for the mask register. The interrupt status bit must be cleared so it can be set again when a higher-priority interrupt occurs. The contents of processor registers are saved because they may be needed by the program that has been interrupted after control returns to it. The interrupt enable IEN is then set to allow other (higher-priority) interrupts and the computer proceeds to service the interrupt request.

The final sequence in each interrupt service routine must have instructions to control the interrupt hardware in the following manner:

1. Clear interrupt enable bit IEN.
2. Restore contents of processor registers.
3. Clear the bit in the interrupt register belonging to the source that has been serviced.
4. Set lower-level priority bits in the mask register.
5. Restore return address into PC and set IEN.

The bit in the interrupt register belonging to the source of the interrupt must be cleared so that it will be available again for the source to interrupt. The lower-priority bits in the mask register (including the bit of the source being interrupted) are set so they can enable the interrupt. The return to the interrupted program is accomplished by restoring the return address to PC. Note that the hardware must be designed so that no interrupts occur while executing steps 2 through 5; otherwise, the return address may be lost and the information in the mask and processor registers may be ambiguous if an interrupt is acknowledged while executing the operations in these steps. For this reason IEN is initially cleared and then set after the return address is transferred into PC.

6. Direct Memory Access (DMA)

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.

Large blocks of data transferred at a high speed to or from high speed devices, magnetic drums, disks, tapes, etc.

DMA controller Interface that provides I/O transfer of data directly to and from the memory and the I/O device

CPU initializes the DMA controller by sending a memory address and the number of words to be transferred

Actual transfer of data is done directly between the device and memory through DMA controller -> Freeing CPU for other tasks

The transfer of data between the peripheral and memory without the interaction of CPU and letting the peripheral device manage the memory bus directly is termed as Direct Memory Access (DMA).

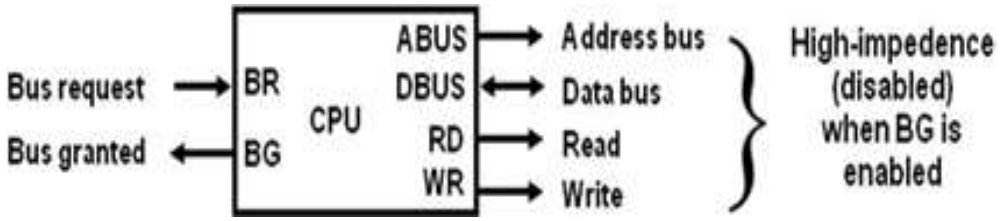


Fig: CPU bus signal for DMA transfer

The two control signals Bus Request and Bus Grant are used to fascinate the DMA transfer. The bus request input is used by the DMA controller to request the CPU for the control of the buses. When BR signal is high, the CPU terminates the execution of the current instructions and then places the address, data, read and write lines to the high impedance state and sends the bus

grant signal. The DMA controller now takes the control of the buses and transfers the data directly between memory and I/O without processor interaction. When the transfer is completed, the bus request signal is made low by DMA. In response to which CPU disables the bus grant and again CPU takes the control of address, data, read and write lines.

The transfer of data between the memory and I/O of course facilitates in two ways which are *DMA Burst* and *Cycle Stealing*.

DMA Burst: The block of data consisting a number of memory words is transferred at a time.

Cycle Stealing: DMA transfers one data word at a time after which it must return control of the buses to the CPU.

- CPU is usually much faster than I/O (DMA), thus CPU uses the most of the memory cycles
- DMA Controller steals the memory cycles from CPU
- For those stolen cycles, CPU remains idle
- For those slow CPU, DMA Controller may steal most of the memory cycles which may cause CPU remain idle long time

DMA Controller:

The DMA controller communicates with the CPU through the data bus and control lines. DMA select signal is used for selecting the controller, the register select is for selecting the register. When the bus grant signal is zero, the CPU communicates through the data bus to read or write into the DMA register. When bus grant is one, the DMA controller takes the control of buses and transfers the data between the memory and I/O.

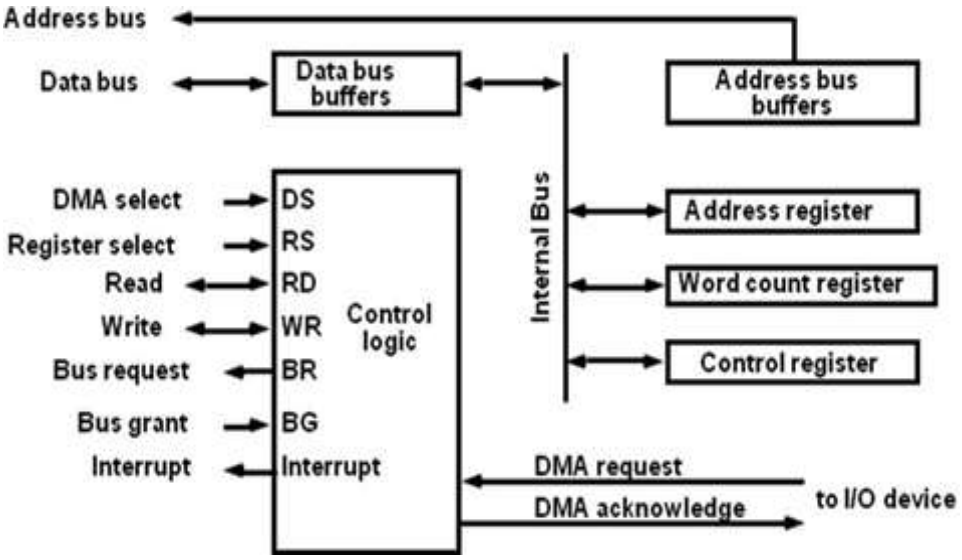


Fig: Block diagram of a DMA Controller

The address register specifies the desired location of the memory which is incremented after each word is transferred to the memory. The word count register holds the number of words to be transferred which is decremented after each transfer until it is zero. When it is zero, it indicates the end of transfer. After which the bus grant signal from CPU is made low and CPU returns to its normal operation. The control register specifies the mode of transfer which is Read or Write.

DMA Transfer:

The data transfer can be showed as below sequences:

1. The DMA request line is used to request a DMA transfer.
2. The bus request (BR) signal is used by the DMA controller to request the CPU to relinquish control of the buses.

3. The CPU activates the bus grant (BG) output to inform the external DMA that its buses are in a high-impedance state (so that they can be used in the DMA transfer.)
4. The address bus is used to address the DMA controller and memory at given location
5. The Device select (DS) and register select (RS) lines are activated by addressing the DMA controller.
6. The RD and WR lines are used to specify either a read (RD) or write (WR) operation on the given memory location.
7. The DMA acknowledge line is set when the system is ready to initiate data transfer.
8. The data bus is used to transfer data between the I/O device and memory.
9. When the last word of data in the DMA transfer is transferred, the DMA controller informs the termination of the transfer to the CPU by means of the interrupt line.

The diagrammatic representation can be showed as below:

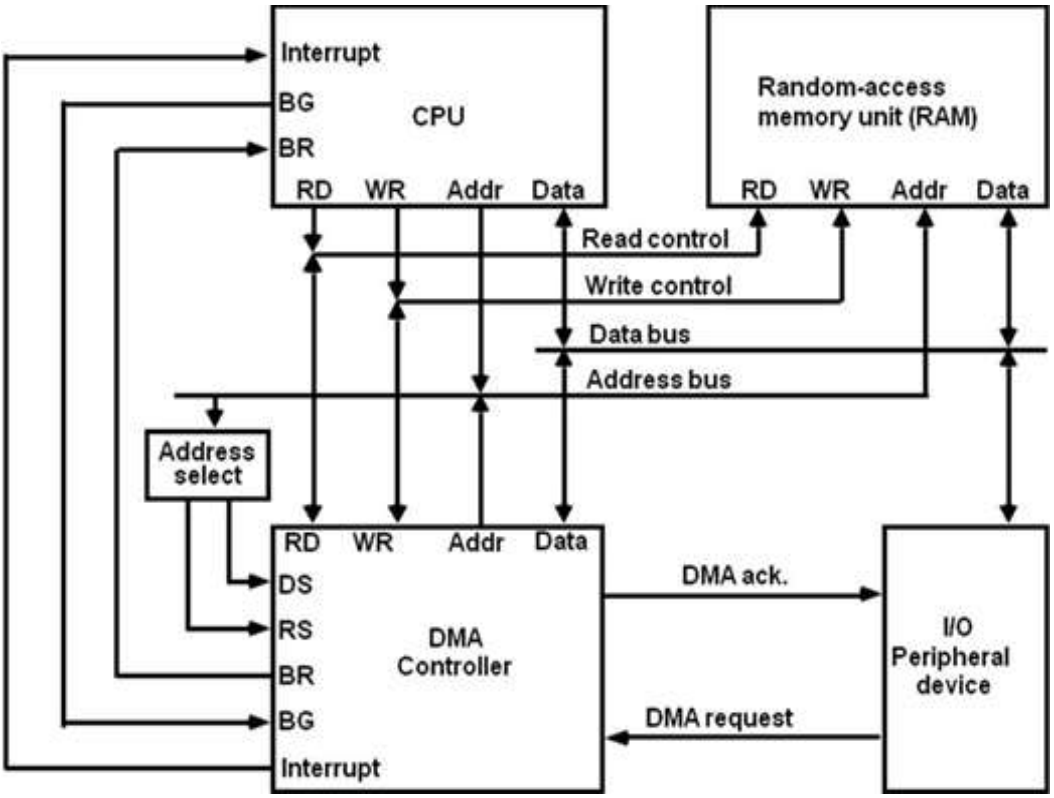


Fig: DMA transfer in a Computer System

Memory Organization

1. Microcomputer Memory

Memory is an essential component of the microcomputer system. It stores binary instructions and datum for the microcomputer. The memory is the place where the computer holds current programs and data that are in use.

Computer memory exhibits perhaps the widest range of type, technology, organization, performance and cost of any feature of a computer system. The memory unit that communicates directly with the CPU is called main memory. Devices that provide backup storage are called auxiliary memory or secondary memory.

Characteristics of memory systems:

The memory system can be characterised with their Location, Capacity, Unit of transfer, Access method, Performance, Physical type, Physical characteristics, Organisation.

Location:

Processor memory: The memory like registers is included within the processor and termed as processor memory.

Internal memory: It is often termed as main memory and resides within the CPU.

External memory: It consists of peripheral storage devices such as disk and magnetic tape that are accessible to processor via i/o controllers.

Capacity:

Word size: Capacity is expressed in terms of words or bytes. The natural unit of organisation
Number of words: Common word lengths are 8, 16, 32 bits etc.

Unit of Transfer:

Internal: For internal memory, the unit of transfer is equal to the number of data lines into and out of the memory module.

External: For external memory, they are transferred in block which is larger than a word.

Addressable unit

- Smallest location which can be uniquely addressed
- Word internally
- Cluster on Magnetic disks

Access Method:

Sequential access: In this access, it must start with beginning and read through a specific linear sequence. This means access time of data unit depends on position of records (unit of data) and previous location. e.g. tape

Direct Access: Individual blocks of records have unique address based on location. Access is accomplished by jumping (direct access) to general vicinity plus a sequential search to reach the final location. e.g. disk

Random access: The time to access a given location is independent of the sequence of prior accesses and is constant. Thus any location can be selected out randomly and directly addressed and accessed. e.g. RAM

Associative access: This is random access type of memory that enables one to make a comparison of desired bit locations within a word for a specified match, and to do this for all words simultaneously. e.g. cache

Performance:

Access time: For random access memory, access time is the time it takes to perform a read or write operation i.e. time taken to address a memory plus to read / write from addressed memory location. Whereas for non-random access, it is the time needed to position read / write mechanism at desired location. i.e. Time between presenting the address and getting the valid data

Memory Cycle time: It is the total time that is required to store next memory access operation from the previous memory access operation.

Memory cycle time = access time plus transient time (any additional time required before a second access can commence). i.e. Time may be required for the memory to "recover" before next access. *Note:*Cycle time is access + recovery.

Transfer Rate: This is the rate at which data can be transferred in and out of a memory unit.i.e. Rate at which data can be moved

- For random access, R = 1 / cycle time
- For non-random access, Tn = Ta + N / R; where Tn – average time to read or write N bits, Ta – average access time, N – number of bits, R – Transfer rate in bits per second (bps).

Physical Types:

- Semiconductor
  RAM
- Magnetic
  Disk & Tape
- Optical
  CD & DVD
- Others
  Bubble
  Hologram

Physical Characteristics:

- Decay: Information decays mean data loss.
- Volatility: Information decays when electrical power is switched off.
- Erasable: Erasable means permission to erase.
- Power consumption: how much power consumes?

Organization:

- Physical arrangement of bits into words
- Not always obvious
  - e.g. interleaved

2. The Memory Hierarchy

Capacity, cost and speed of different types of memory play a vital role while designing a memory system for computers.

If the memory has larger capacity, more application will get space to run smoothly.

It's better to have fastest memory as far as possible to achieve a greater performance. Moreover for the practical system, the cost should be reasonable.

There is a trade-off between these three characteristics cost, capacity and access time. One cannot achieve all these quantities in same memory module because

If capacity increases then access time increases (slower) and due to which cost per bit decreases.

If access time decreases (faster), capacity decreases and due to which cost per bit increases.

The designer tries to increase capacity because cost per bit decreases and the more application program can be accommodated. But at the same time, access time increases and hence decreases the performance. ***So the best idea will be to use memory hierarchy.***

Memory Hierarchy is to obtain the highest possible access speed while minimizing the total cost of the memory system.

Not all accumulated information is needed by the CPU at the same time.

Therefore, it is more economical to use low-cost storage devices to serve as a backup for storing the information that is not currently used by CPU

The memory unit that directly communicate with CPU is called the *main memory*

Devices that provide backup storage are called *auxiliary memory*

The memory hierarchy system consists of all storage devices employed in a computer system from the slow by high-capacity auxiliary memory to a relatively faster main memory, to an even smaller and faster cache memory

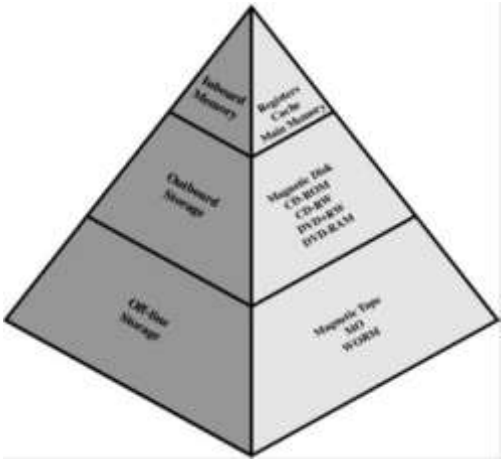
The main memory occupies a central position by being able to communicate directly with the CPU and with auxiliary memory devices through an I/O processor

A special very-high-speed memory called **cache** is used to increase the speed of processing by making current programs and data available to the CPU at a rapid rate.

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

The cache is used for storing segments of programs currently being executed in the CPU and temporary data frequently needed in the present calculations

The memory hierarchy system consists of all storage devices employed in a computer system from slow but high capacity auxiliary memory to a relatively faster cache memory accessible to high speed processing logic. The figure below illustrates memory hierarchy.



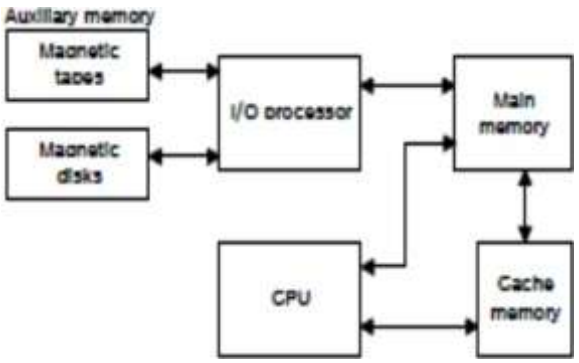


Fig: Memory Hierarchy

As we go down in the hierarchy

- Cost per bit decreases
- Capacity of memory increases
- Access time increases
- Frequency of access of memory by processor also decreases.

Hierarchy List

- Registers
- L1 Cache
- L2 Cache
- Main memory
- Disk cache
- Disk
- Optical
- Tape

3. Internal (or) Main Memory

- The main memory is the central unit of the computer system. It is relatively large and fast memory to store programs and data during the computer operation. These memories employ semiconductor integrated circuits. The basic element of the semiconductor memory is the memory cell.
- The memory cell has three functional terminals which carries the electrical signal.
    - The select terminal: It selects the cell.
    - The data in terminal: It is used to input data as 0 or 1 and data out or sense terminal is used for the output of the cell's state.
    - The control terminal: It controls the function i.e. it indicates read and write.

- Most of the main memory in a general purpose computer is made up of RAM integrated circuits chips, but a portion of the memory may be constructed with ROM chips

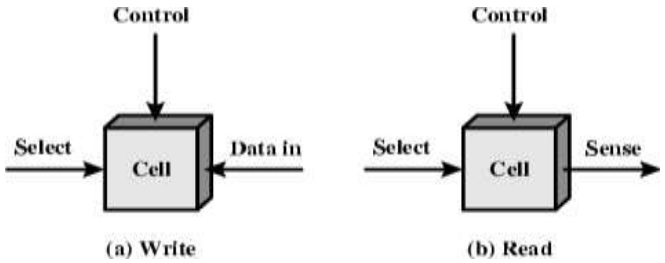


Fig: Memory Cell

RAM– Random Access memory:

- Memory cells can be accessed for information transfer from any desired random location.
- The process of locating a word in memory is the same and requires of locating a word in memory is the same and requires an equal amount of time no matter where the cells are located physically in memory thus named 'Random access'.
- Integrated RAM are available in two possible operating modes, S*tatic and Dynamic.*

SRAM versus DRAM

- Both volatile
  - Power needed to preserve data
- Static RAM
  - Uses flip flop to store information
  - Needs more space
  - Faster, digital device
  - Expensive, big in size
  - Don't require refreshing circuit
  - Used in cache memory
- Dynamic RAM
  - Uses capacitor to store information
  - More dense i.e. more cells can be accommodated per unit area
  - Slower, analog device
  - Less expensive, small in size
  - Needs refreshing circuit
  - Used in main memory, larger memory units

ROM– Read Only memory:

- Read only memory (ROM) contains a permanent pattern of data that cannot be changed.
- A ROM is non-volatile that is no power source is required to maintain the bit values in memory.
- While it is possible to read a ROM, it is not possible to write new data into it.
- The data or program is permanently presented in main memory and never be loaded from a secondary storage device with the advantage of ROM.
- A ROM is created like any other integrated circuit chip, with the data actually wired into the chip as part of the fabrication process.
- It presents two problems
  - The data insertion step includes a relatively large fixed cost, whether one or thousands of copies of a particular ROM are fabricated.
  - There is no room for error. If one bit is wrong, the whole batch of ROM must be thrown out.

Types of ROM

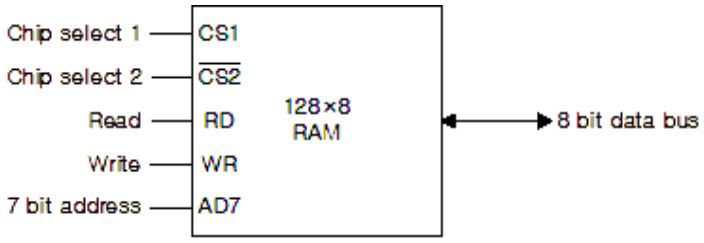
- Programmable ROM (PROM)

- It is non-volatile and may be written into only once. The writing process is performed electrically and may be performed by a supplier or customer at a time later than the original chip fabrication.

- Erasable Programmable ROM (EPROM)
  - It is read and written electrically. However, before a write operation, all the storage cells must be erased to the same initial state by exposure of the packaged chip to ultraviolet radiation (UV ray). Erasure is performed by shining an intense ultraviolet light through a window that is designed into the memory chip. EPROM is optically managed and more expensive than PROM, but it has the advantage of the multiple update capability.
- Electrically Erasable programmable ROM (EEPROM)
  - This is a read mostly memory that can be written into at any time without erasing prior contents, only the byte or byte addresses are updated. The write operation takes considerably longer than the read operation, on the order of several hundred microseconds per byte. The EEPROM combines the advantage of non-volatility with the flexibility of being updatable in place, using ordinary bus control, addresses and data lines. EEPROM is more expensive than EPROM and also is less dense, supporting fewer bits per chip.
- Flash Memory
  - Flash memory is also the semiconductor memory and because of the speed with which it can be reprogrammed, it is termed as flash. It is interpreted between EPROM and EEPROM in both cost and functionality. Like EEPROM, flash memory uses an electrical erasing technology. An entire flash memory can be erased in one or a few seconds, which is much faster than EPROM. In addition, it is possible to erase just blocks of memory rather than an entire chip. However, flash memory doesn't provide byte level erasure, a section of memory cells are erased in an action or 'flash'.

Now, we can assume RAM and ROM Chips sizes are

- RAM and ROM Chips
- Typical RAM chip, as shown in below figure.
- 128 X 8 RAM : $2^7 = 128$ (7 bit address lines)
- Typical ROM chip, as shown in below figure.
- 512 X 8 ROM : $2^9 = 512$ (9 bit address lines)



(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

Fig: RAM chip

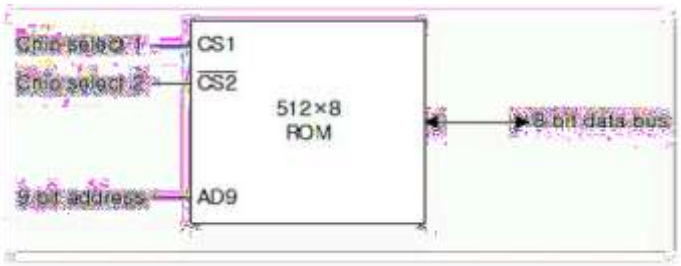


Fig: ROM chip

If we want to construct the Main memory size as 1024 X 8, then the Memory Configuration:

512 bytes **RAM** + **512** bytes **ROM**

1 x **512** byte **ROM** + **4** x **128** bytes RAM

Memory Address Map can be showed as below:

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

Memory Connection to CPU uses 2 x 4 Decoder: RAM select (**CS1**)

The circuit as shown below:

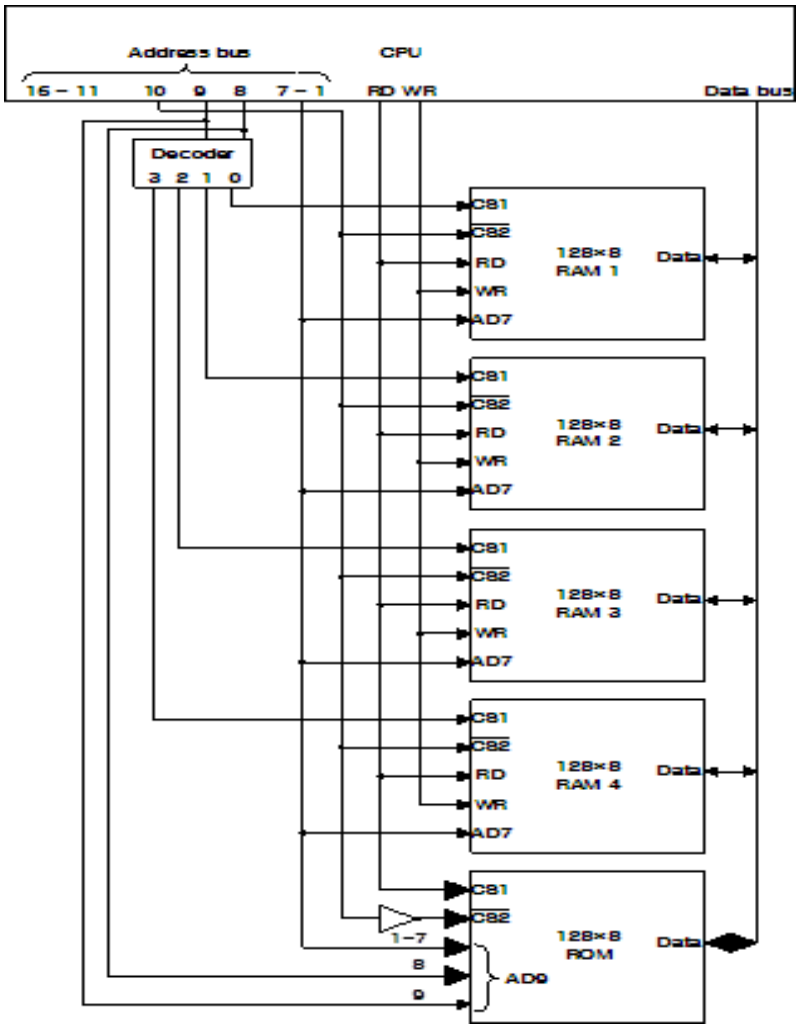


Fig: Main memory

4. 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. Many search algorithms have been developed to minimize the number of accesses while searching for an item in a random or sequential access memory.

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 content addressable 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 locates 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. Moreover, searches can be done on an entire word or on a specific field within a word. An associative memory is more expensive than a random access memory because each cell must have storage capability as well as logic circuits for matching its content with an external argument.

Note: 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 is shown in below figure. It consists of a memory array andlogic for m-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. The matchregister Mhas m-bits, one for each memory word. The words thatmatch 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.

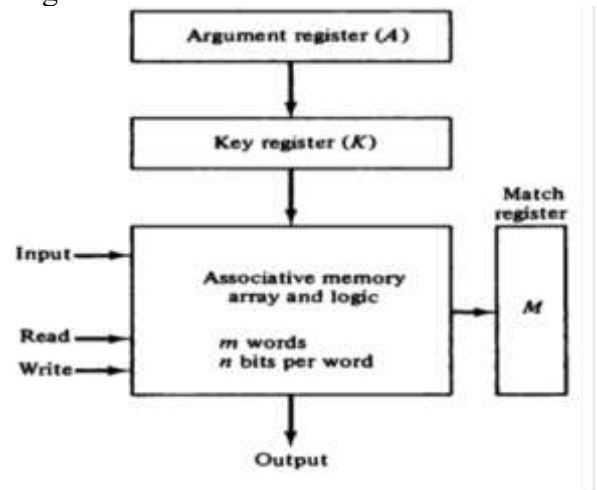


Fig: Block diagram of Associative Memory

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 l'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 Cwith 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 bitj in word i. A bit Aj in the argument register is compared with all the bits in column j of the array provided that Kj = 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.

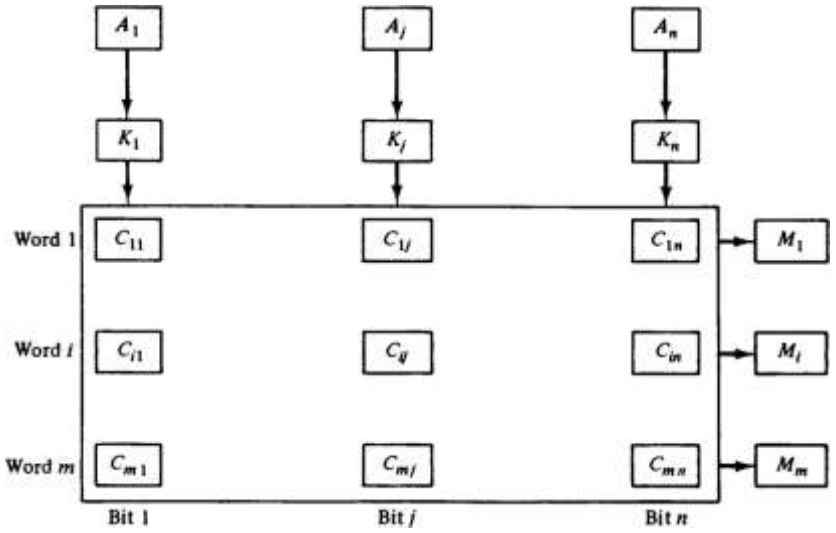


Fig: Associative Memory for m-words, n-cells for word

5. External Memory (or) Auxiliary memory

- The devices that provide backup storage are called external memory or auxiliary memory. It includes serial access type such as magnetic tapes and random access type such as magnetic disks.

Magnetic Tape:

- A magnetic tape is the strip of plastic coated with a magnetic recording medium. Data can be recorded and read as a sequence of character through read / write head. It can be stopped, started to move forward or in reverse or can be rewound.
- Data on tapes are structured as number of parallel tracks running length wise. Earlier tape system typically used nine tracks. This made it possible to store data one byte at a time with additional parity bit as 9th track. The recording of data in this form is referred to as parallel recording.

Magnetic Disk:

- A magnetic disk is a circular plate constructed with metal or plastic coated with magnetic material often both side of disk are used and several disk stacked on one spindle which Read/write head available on each surface. All disks rotate together at high speed. Bits are stored in magnetize surface in spots along concentric circles called tracks. The tracks are commonly divided into sections called sectors. After the read/write head are positioned in

specified track the system has to wait until the rotating disk reaches the specified sector under read/write head.

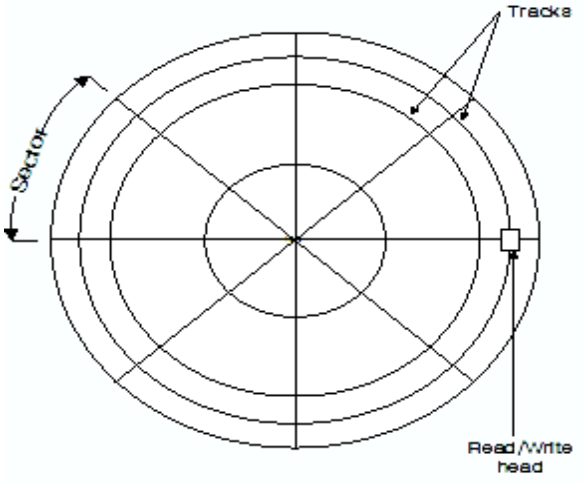


Fig: Magnetic Disk

- Information transfer is very fast once the beginning of sector has been reached. Disk that are permanently attached to the unit assembly and cannot be used by occasional user are called hard disk drive with removal disk is called floppy disk.

6. Cache memory

Principles:

- Intended to give memory speed approaching that of fastest memories available but with large size, at close to price of slower memories
- Cache is checked first for all memory references.
- If not found, the entire block in which that reference resides in main memory is stored in a cache slot, called a line
- Each line includes a tag (usually a portion of the main memory address) which identifies which particular block is being stored
- Locality of reference implies that future references will likely come from this block of memory, so that cache line will probably be utilized repeatedly.
- The proportion of memory references, which are found already stored in cache, is called the hit ratio.
- Cache memory is intended to give memory speed approaching that of the fastest memories available, and at the same time provide a large memory size at the price of less expensive types of semiconductor memories. There is a relatively large and slow main memory together with a smaller, faster cache memory contains a copy of portions of main memory.
- When the processor attempts to read a word of memory, a check is made to determine if the word is in the cache. If so, the word is delivered to the processor. If not, a block of main memory, consisting of fixed number of words is read into the cache and then the word is delivered to the processor.
- The locality of reference property states that over a short interval of time, address generated by a typical program refers to a few localized area of memory repeatedly. So if programs and data which are accessed frequently are placed in a fast memory, the average access time can be reduced. This type of small, fast memory is called cache memory which is placed in between the CPU and the main memory.

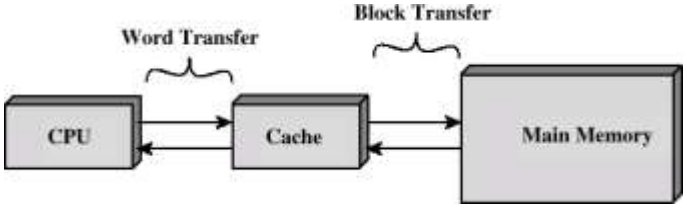


Fig: Positions of the Cache Memory

- When the CPU needs to access memory, cache is examined. If the word is found in cache, it is read from the cache and if the word is not found in cache, main memory is accessed to read word. A block of word containing the one just accessed is then transferred from main memory to cache memory.
- Cache connects to the processor via data control and address line. The data and address lines also attached to data and address buffer which attached to a system bus from which main memory is reached.
- When a cache hit occurs, the data and address buffers are disabled and the communication is only between processor and cache with no system bus traffic. When a cache miss occurs, the desired word is first read into the cache and then transferred from cache to processor. For later case, the cache is physically interposed between the processor and main memory for all data, address and control lines.
    - CPU generates the receive address (RA) of a word to be moved (read).
    - Check a block containing RA is in cache.
    - If present, get from cache (fast) and return.
    - If not present, access and read required block from main memory to cache.
    - Allocate cache line for this new found block.
    - Load bock for cache and deliver word to CPU
    - Cache includes tags to identify which block of main memory is in each cache slot

Locality of Reference:

- The reference to memory at any given interval of time tends to be confined within a few localized area of memory. This property is called locality of reference. This is possible because the program loops and subroutine calls are encountered frequently. When program loop is executed, the CPU will execute same portion of program repeatedly. Similarly, when a subroutine is called, the CPU fetched starting address of subroutine and executes the subroutine program. Thus loops and subroutine localize reference to memory.
- This principle states that memory references tend to cluster over a long period of time, the clusters in use changes but over a short period of time, the processor is primarily working with fixed clusters of memory references.

Spatial Locality:

- It refers to the tendency of execution to involve a number of memory locations that are clustered.
- It reflects tendency of a program to access data locations sequentially, such as when processing a table of data.

Temporal Locality:

- It refers to the tendency for a processor to access memory locations that have been used frequently. For e.g. Iteration loops executes same set of instructions repeatedly.
-

Cache Memory Mapping Functions:

- The transformation of data from main memory to cache memory is referred to as memory mapping process.
- Because there are fewer cache lines than main memory blocks, an algorithm is needed for mapping main memory blocks into cache lines.
- There are three different types of mapping functions in common use and are direct, associative and set associative.

Example of Cache and Main memory sizes are:

Main memory: 32 K x 12 bit word (15 bit address lines)

Cache memory: 512 x 12 bit word

CPU sends a 15-bit address to cache

Hit : CPU accepts the 12-bit data from cache

Miss : CPU reads the data from main memory (then data is written to cache).

1) *Direct Mapping:*

It is the simplex technique, maps each block of main memory into only one possible cache line i.e. a given main memory block can be placed in one and only one place on cache.

$i = j \ modulo \ m$,Where i = cache line number; j = main memory block number; m = number of lines in the cache

The mapping function is easily implemented using the address. For purposes of cache access, each main memory address can be viewed as consisting of three fields.

The least significant w bits identify a unique word or byte within a block of main memory. The remaining s bits specify one of the 2s blocks of main memory.

The cache logic interprets these s bits as a tag of (s-r) bits most significant position and a line field of r bits. The latter field identifies one of the m = 2r lines of the cache.

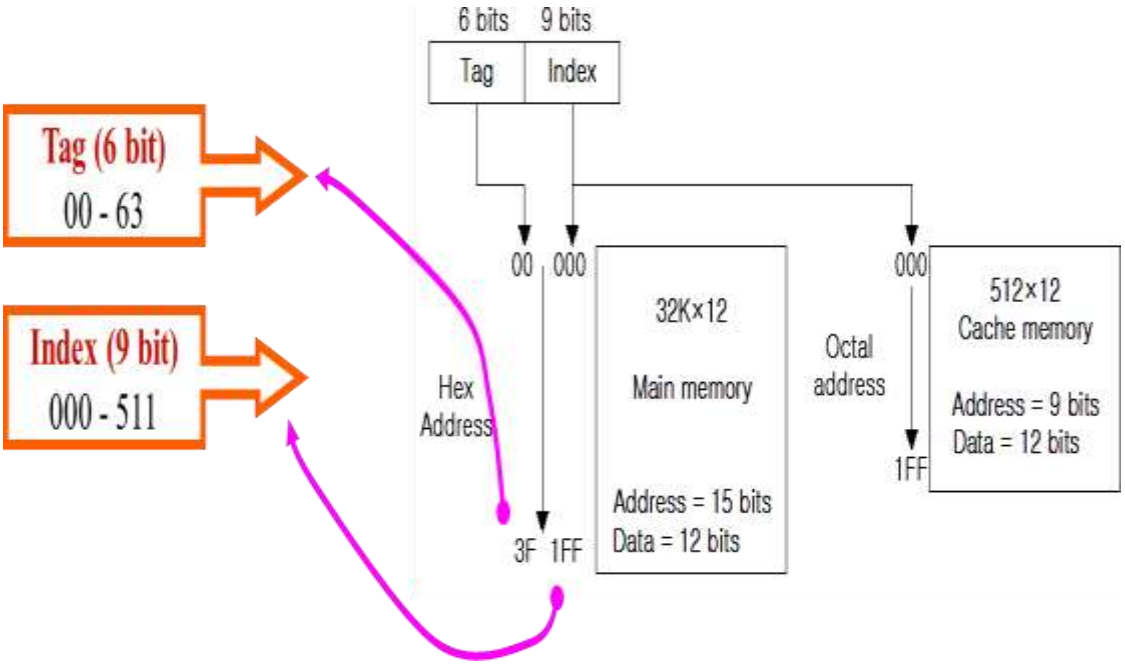


Fig: Addressing Relations b/w Main and Cache memory

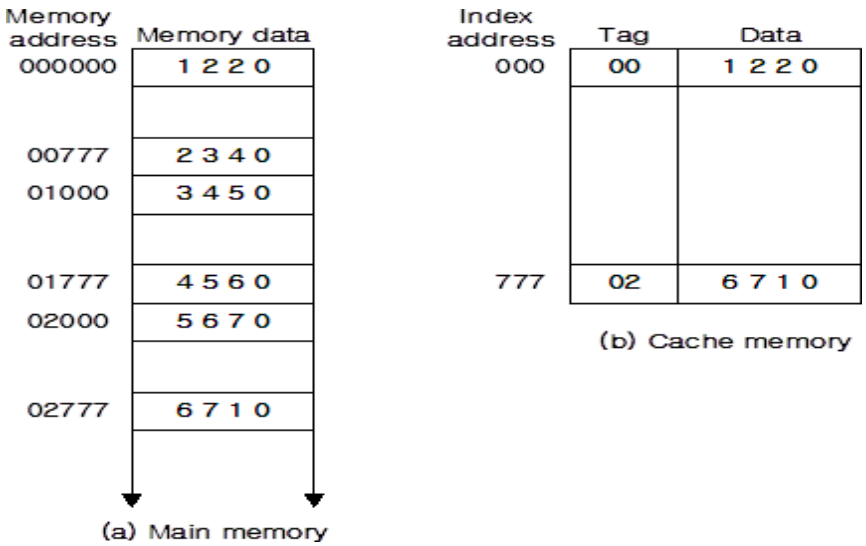


Fig: Direct Mapping

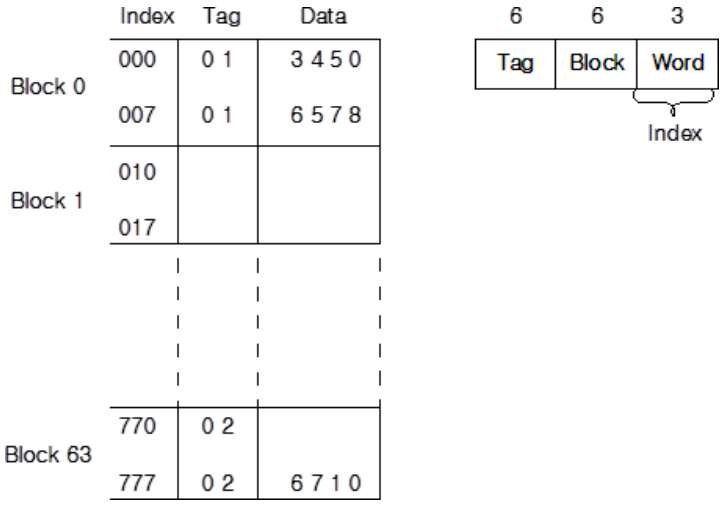


Fig: Direct Mapping with Block size of 8 words

2) Associative mapping:

It overcomes the disadvantage of direct mapping by permitting each main memory block to be loaded into any line of cache.

Cache control logic interprets a memory address simply as a tag and a word field Tag uniquely identifies block of memory

Cache control logic must simultaneously examine every line's tag for a match which requires fully associative memory very complex circuitry, complexity increases exponentially with size cache searching gets expensive

Address DataCache memory

Tag field (n - k) and Index field (k)

2^k words cache memory + 2^n words main memory

Tag = 6 bit (15 - 9), Index = 9 bit

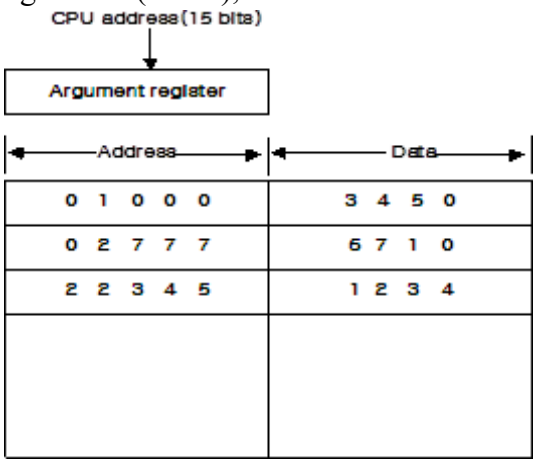


Fig: Associative Memory Mapping

3) Set-Associative Mapping:

It is a compromise between direct and associative mappings that exhibits the strength and reduces the disadvantages

Cache is divided into v sets, each of which has k lines; number of cache lines = vk

$M = v X k$

$I = j \ modulo \ v$

Where, i = cache set number; j = main memory block number; m = number of lines in the cache. So a given block will map directly to a particular set, but can occupy any line in that set (associative mapping is used within the set)

Cache control logic interprets a memory address simply as three fields tag, set and word. The d set bits specify one of v = 2d sets. Thus s bits of tag and set fields specify one of the 2s block of main memory.

The most common set associative mapping is 2 lines per set, and is called two-way set associative. It significantly improves hit ratio over direct mapping, and the associative hardware is not too expensive.

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

Fig: Set-Associative Memory Mapping

Fig: Memory Table in Page System

Important Questions:

1. Define Memory and analyze the memory hierarchy in terms of speed, size and Cost.
2. Explain internal organization of memory chips, Design 64k X 16 memory chip using 16k X 8 memory chips
3. Define Auxiliary memory? Discuss with neat diagrams
4. What is associate memory? Explain with block diagram
5. Discuss about different mapping procedures of cache memory
6. Discuss about the virtual memory? Discuss about the mapping of virtual address to memory table.