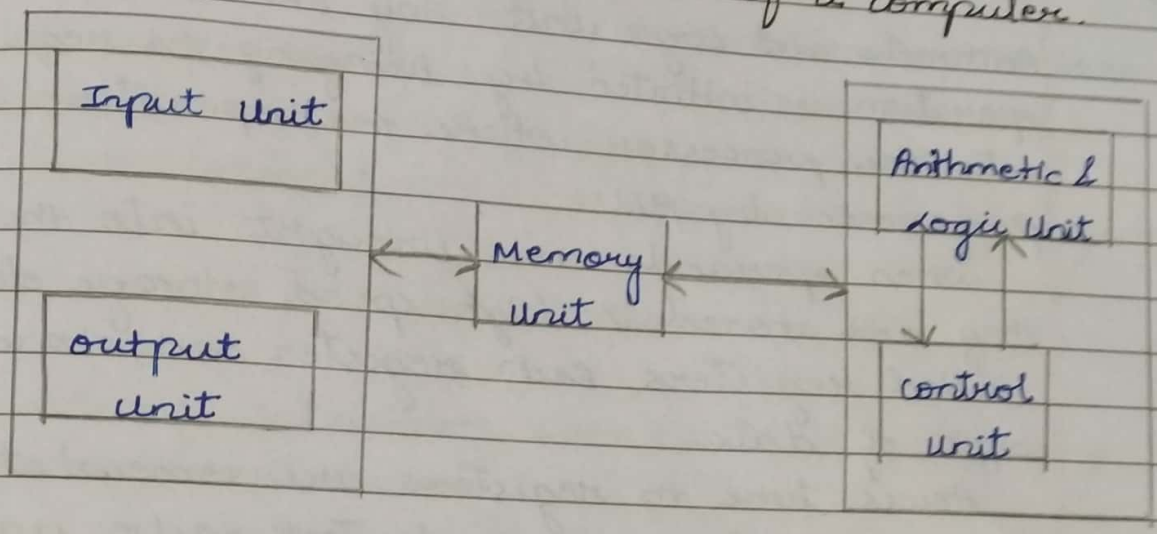
* Unit - 1

① Explain different functional units of a computer.

Sol^n



- Input Unit : Computer accepts encoded information through input unit. The standard input device is a keyboard. whenever a key is pressed, keyboard controller sends the code to CPU/ Memory.
  Examples include Mouse, Joystick, Scanner etc.

- Memory Unit : It stores the program instructions, data and results of computation etc. It is classified as:

a) Primary / Main memory
b) Secondary / Auxillary memory

Here primary memory is a fast memory that operates at electronic speeds. It contains a large number of semiconductor, storage cells, each capable of storing one bit of information. These cells are rarely read or written as individual cells but instead are processed in groups of fixed sizes called words. The memory is organized so that the contents of one word, containing n bits can be

stored or retrieved in one basic operation. A distinct address is associated with each word location.

- Arithmetic and Logic unit: Any arithmetic or logic operation is initiated by bringing the required operand into the processor, where the operation is performed by ALU.
  When operands are brought into the processor, they are stored in high speed storage elements called registers. Each register can store one word of data.
  Access time to registers are somewhat faster than access time to the fastest cache unit in memory hierarchy.

- control unit: The operations of all the functional units are controlled by this unit. Control unit is a well defined, physically separate unit that interacts with other parts of the machine. A large set of control lines (wires) carries the signals used for timing and synchronization of events in all units.

② Basic operational concepts

An instruction consists of two parts, an operation code and operand/s as shown below.

OPCODE	OPERAND/s

Let us see a typical instruction

ADD LOCA, R0

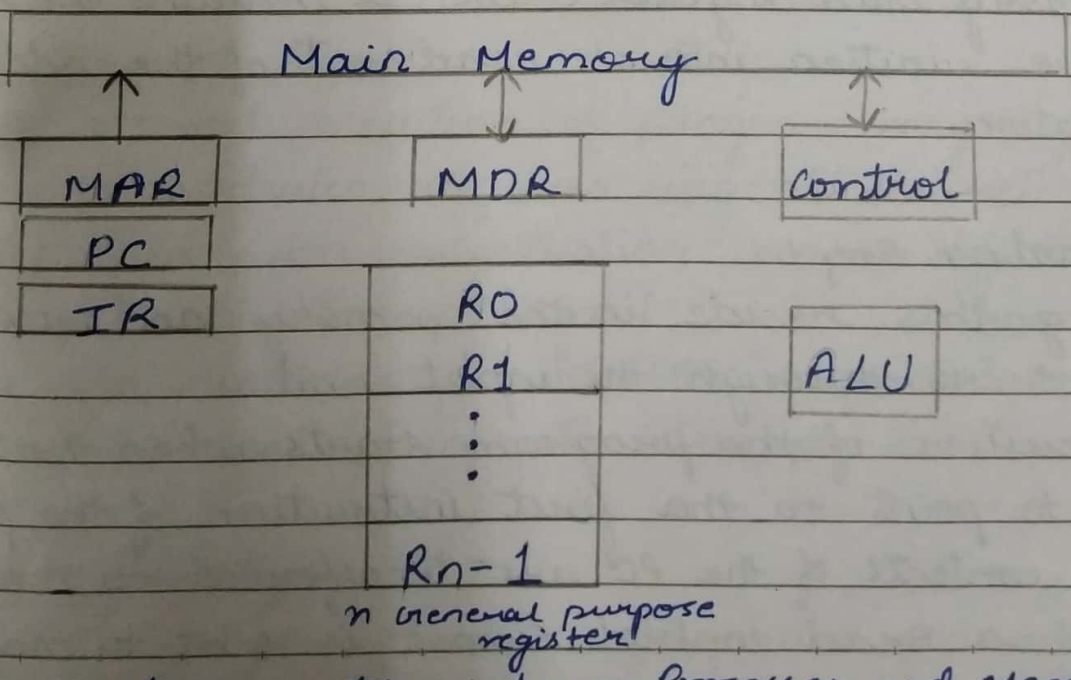
This instruction is an addition operation. The

following are the steps to execute the instruction:
Step 1: Fetch the instruction from main memory into the processor.
Step 2: Fetch the operand at location LOCA from main memory into the processor.
Step 3: Add the memory operand (i.e fetched contents of LOCA) to the contents of register R0.
Step 4: store the result (sum) in R0.

→ The same instruction can be realized using two instructions as:

Load LOCA, R1
Add R1, R0

Step 1: Fetch the instruction from main memory into the processor.
Step 2: Fetch the operand at location LOCA from main memory into the processor Register R1.
Step 3: Add the content of Register R1 and the contents of register R0.
Step 4: store the result (sum) in R0.



(Interconnection between Processor and Memory)

Registers: They are fast stand-alone storage locations that hold data temporarily. Multiple registers are needed to facilitate the operation of the CPU.

Some of these Registers are:

1. Instruction register (IR): It holds the instruction that is currently being executed. Its output is available to the control circuits, which generate the timing signals that controls the various processing elements.

2. Program counter (PC): It keeps track of the execution of the program. It contains the memory address of the next instruction to be fetched and executed.

3. General purpose register (R0 - Rn-1): There are n general purpose registers R0 to Rn-1 which can be used by the programmers during writing programs.

4. Memory address register (MAR): It holds the address of the location to be accessed.

5. Memory Data Register (MDR): It contains the data to be written into or read out of the addressed location.

→ Operating Steps:

- Programs reside in the memory and usually get there through the input unit.
- Execution of the program starts when the PC is set to point to the first instruction of the program.
- The contents of the PC are transferred to the MAR and a Read control signal is sent to memory.

- After the time required to access the memory elapses, the addressed word (in this case, the first instruction of the program) is read out of the memory and loaded into the MDR.
- Next the contents of the MDR are transferred to the IR. At this point, the instruction is ready to be decoded and executed.
- If the instruction involves an operation to be performed by the ALU, it is necessary to obtain the required operands. If an operand resides in the memory, it has to be fetched by sending the its address to the MAR and initiating a Read cycle. When the operand has been read from the memory into the MDR, it is transferred to the ALU. If the result is to be stored in the memory then the result is sent to the MDR. The address of the location where the result is to be stored is sent to MAR.
- As soon as the execution of the current instruction is completed, a new instruction fetch maybe started.

③ Interrupt signal

Normal execution of program maybe precepted if some device requires urgent servicing. In order to deal with the situation immediately, the device raises an interrupt signal.

The processor provides the requested service by executing an appropriate interrupt-service routine.

④ Bus Structures

To achieve good performance of a computer

system all the computer units can transfer one word of data at a time. All the bits of a word transfer in parallel, that is the bits are transferred simultaneously over many lines one bit per line. There are many ways to connect different parts inside a computer together.

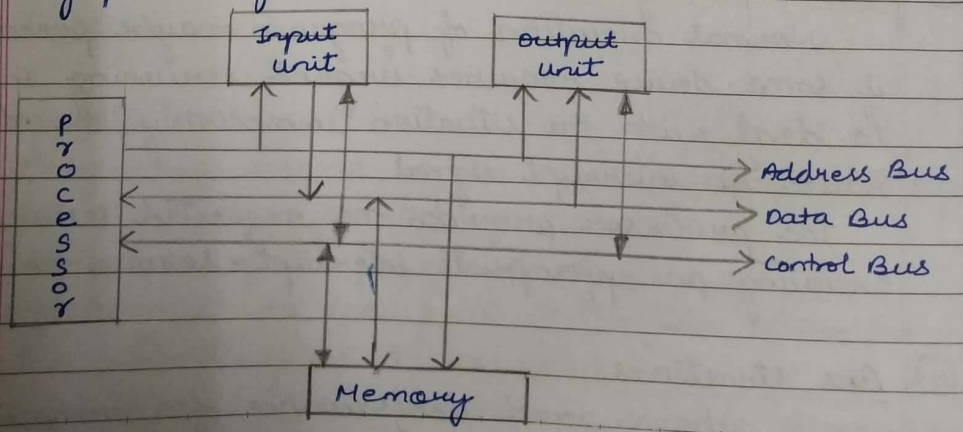
A group of lines that serves as a connecting path for several devices is called a bus. A bus that connects major components is called system bus. which is divided into three functional groups:

a) Address bus: It is unidirectional and consist of group of wires which carries address information bits from processor to peripherals (16, 20, 24 or more parallel signal lines).

b) Databus: It is bidirectional and consist of group of wires which carries data information bit form processor to peripherals and vice-versa.

c) control bus: It is bidirectional and consist of group of wires which carries control signals from processor to peripherals and vice-versa.

only two units can use bus structure at any point of time.



- Single Bus structure: It is a common bus used to communicate between peripherals and microprocessor.

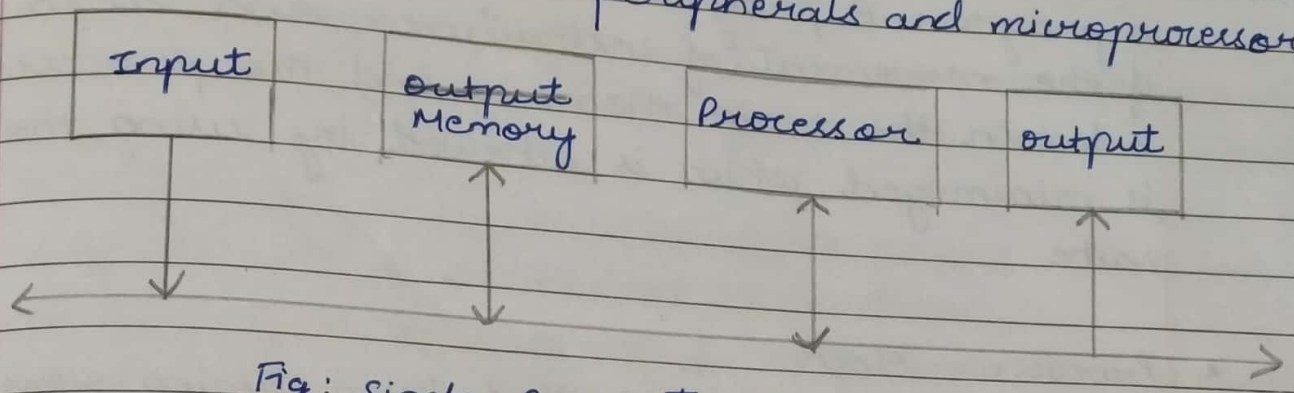


Fig: Single Bus structure

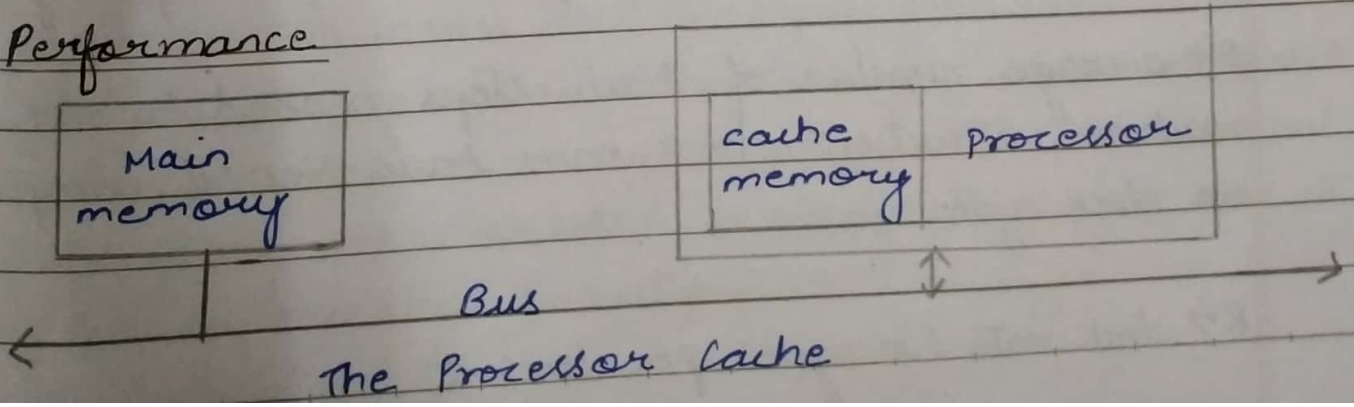
The main virtue of the single bus structure is its low cost and its flexibility for attaching peripheral devices.

- Multibus structure: To improve performance multibus structure can be used. In two bus structure, one bus can be used to fetch instruction other can be used to fetch data, required for execution. Thus, on improving the performance, cost increases.

(5) Buffer Registers

They are used with devices to hold the information during transfers. Buffer registers smooth out timing differences among processors, memories and I/O devices. It allows the processor to switch rapidly from one device to another.

(6) Performance



It is the time taken by the system to execute a program. A program will be executed faster if the movement of instructions and data between the main memory and the processor is minimized, which is achieved by using the cache.

- **Processor Clock**

Processor circuits are controlled by a timing signal called a clock. The clock defines regular time intervals called clock cycles.

To execute a machine instruction, the processor divides the action to be performed into a sequence of basic steps, such that each step can be completed in one clock cycle.

The length P of one clock cycle is an important parameter that affects processor performance. It's inverse is the clock rate R.

$$R = \frac{1}{P} \text{ Hz}$$

- **Basic Performance Equation**

T → processor time required to execute a program that has been prepared in some-high level language.

N → no. of machine level language instructions that are executed in the execution of one complete program.

S → average number of basic steps needed to execute one machine instruction where each basic step is completed in one clock cycle.

R → clock rate (in cycles per second)

$$\therefore T = \frac{N \times S}{R}$$

→ To achieve higher performance, T must be decreased. This is achieved by reducing N, and S and increasing R.

→ The value of N is reduced if the source code is compiled into fewer machine instructions.

→ The value of S is reduced if instructions have a smaller no. of basic steps performed if the execution of instruction is overlapped.

→ The value of R is increased by using higher frequency clock which reduces the execution time of a basic step.

- Pipelining and superscalar

Pipelining is the overlapping of execution of successive instructions.

Add R_1, R_2, R_3

The above command adds the contents of registers R_1 and R_2 and places the sum into R_3.

The contents of R_1 and R_2 are first transferred to the inputs of ALU. After the add operation is performed the sum is transferred to R_3. The processor can read the next instruction from the memory while the addition operation is being performed.

A higher degree of concurrency is achieved if multiple instruction pipelines are implemented in the processor. This means that multiple functional units are used, creating parallel paths through which different instructions can be executed in parallel. Thus several instructions are executed in every clock cycle. This is called superscalar Execution.

- Clock Rate

There are two ways to increase the clock rate (R):

a) improving the IC technology, making logic circuit faster
b) reducing the amount of processing done in one basic step.

- Instruction set: CISC and RISC

Reducing amount of processing done in one basic step also makes it possible to reduce the clock period, P. However, if the actions that have to be performed by an instruction remain the same, the number of basic steps needed may increase. It reduces the number of basic steps to execute.

RISC: Reduced instruction set computers

CISC: complex instruction set computers

- Compiler:

It translates a high level language program into a sequence of machine instructions. An optimizing compiler takes advantage of the various features of the target processor to reduce the product N×S, which is the total no. of clock cycles needed to execute a program.

The compiler may rearrange program instructions to achieve better performance.

(7) Performance Measurement

The performance measure is the time taken by the computer to execute a given benchmark. Initially some attempts were made to create artificial programs that could be used as benchmark programs. A non profit organization called SPEC-system performance evaluation

corporation selects and publishes benchmarks.
The program selected range from game-playing, compiler and database applications to numerically intensive programs in astrophysics and quantum chemistry. In each case, the program is compiled under test, and the running time on a real computer is measured. The same program is also compiled and run on one computer selected as reference.

The 'SPEC' rating is computed as follows:-

$$\text{SPEC rating} = \frac{\text{Running time on the reference computer}}{\text{Running time on the computer under test}}$$

$$\text{SPEC rating} = \left(\prod_{i=1}^{n} \text{SPEC}_i\right)^{1/n}$$

If the SPEC rating = 50 means that the computer under test is 50 times as fast as the ultra sparc 10. This is repeated for all the programs in the SPEC suit, and the geometric mean of the result is computed.

⑧ History of computers

Development of technologies used to fabricate the processors, memory and I/O units of the computers has been divided into various generations as given below:

- First generation (1946 to 1955): Computers of this generation used Vacuum tubes. The computers were built using stored program concept. Ex: ENIAC, EDSAC, IBM 701. Computers of this age typically used about ten thousand vacuum tubes. They were bulky in size had slow operating speed, short life time and limited programming faulities.

- Second generation (1955 to 1965): Computers of this generatio

used the germanium transistors as the active switching electronic device. Eg: IBM 7000, B5000, IBM 1400. Comparatively smaller in size and about ten times fast operating speed as compared to first generation vacuum tube based computers. Consumed less power, had fairly good reliability. Availability of large memory was an added advantage.

- Third generation (1965 to 1975): The computers of this generation used the integrated circuits as the active electronic components. Eg: IBM system 360, PDP minicomputer etc. They were still smaller in size. They had powerful CPUs with the capacity of executing 1 million instructions per second (MIPS). It used to consume very less power consumption.

- Fourth generation (1976 to 1990): The computers of this generation used the VLSI chips like microprocessor as their active electronic element. HCL horizon III and WIPRO Uniplus + HCL's Busybee PC etc. They used high speed microprocessor as CPU. They were more user friendly and highly reliable systems. They had large storage capacity disk memories.

- Beyond Fourth generation (1990 onwards): Specialized and dedicated VLSI chips are used to control specific functions of these computers. Modern Desktop PC's, Laptops or Notebook computers.

  Computer featuring artificial intelligence, massively parallel machines and extensively distributed systems are examples of current trends.

(9) Multiprocessors & Multicomputers

→ Large computer systems containing a large no of processor units are called multiprocessor systems. These systems either execute a number of different application tasks in parallel or they execute subtasks of a single large task in parallel.

→ Shared Memory Multiprocessor systems - It means that all the processors in a multiprocessor system have access to all of the memory in the system.

→ Multicomputer system is a group of interconnected, complete computers. When the task they are executing need to communicate data, they do so by exchanging messages over a communication network.

→ Message Passing Multicomputer Systems - The computer normally has access only to their own memory units, thus there is a need to exchange messages between computers.

(10) Basic Input / output operations

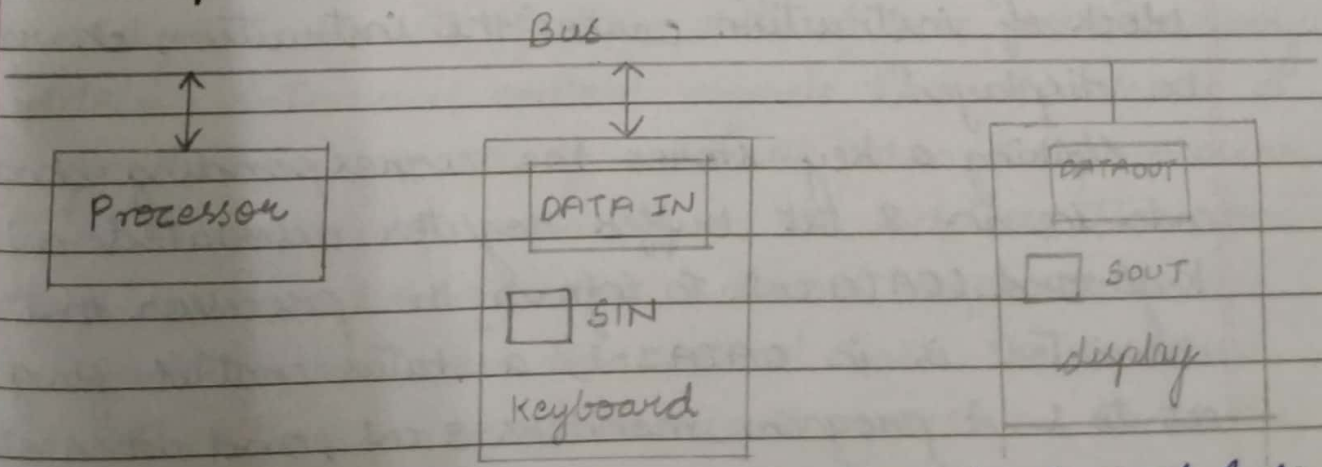


fig: Bus connection for processor, keyboard & display

A simple way of performing basic I/O tasks is to use a method known as program controlled I/O.

The rate of data transfer from the keyboard to a computer is limited by the typing speed of the user, which is unlikely to exceed a few characters per second. The rate of output transfers from the computer to the display is much higher. It is still slower than the speed of a processor that can execute millions of instructions per second. The difference in speeds between the processor & I/O devices creates the need for mechanisms to synchronise the transfer of data between them.

A solution to this problem is as follows: on output, the processor sends the first character and then waits for the signal from the display that the character has been received. It then sends the character and so on. Input from the keyboard is sent in a similar way. The processor waits for a signal indicating that a character has been struck and that its code is available in some buffer register.

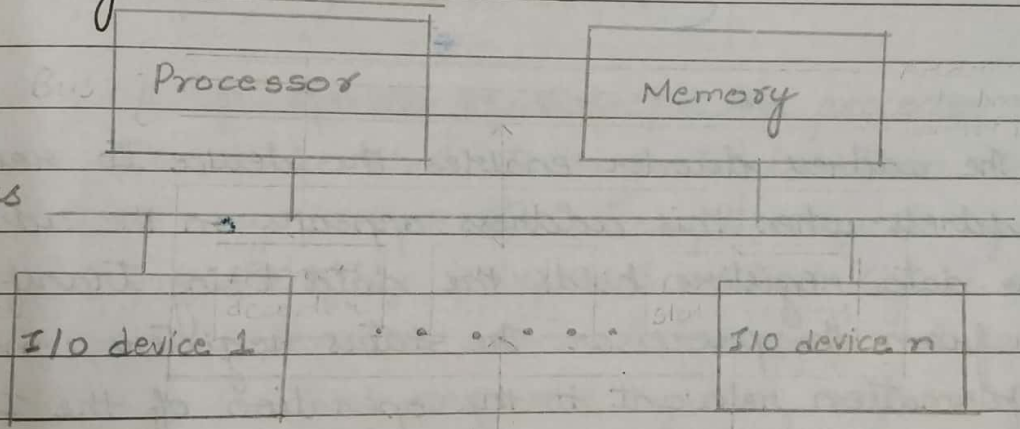
The action of striking a key doesnot automatically cause the corresponding character to be displayed. One block of instruction in the I/O program transfers the character into the processor, and another associated block of instruction causes the instruction character to be displayed.

Striking a key stores the corresponding character code in an 8 bit buffer register associated with the keyboard (DATAIN). To inform the processor that a valid character is in DATAIN, a status control flag, SIN is set to 1. A program monitors SIN, and when SIN is set to 1, the processor reads the character from DATAIN. When the character is transferred to the processor, SIN is automatically cleared to 0.

An analogous process takes place when characters are transferred from the processor to the display. When the status control flag SOUT equals to 1, the character is transferred to the buffer register DATAOUT. The transfer of data to DATAOUT clears SOUT to 0. When the display device is ready to receive another character, SOUT is again set to 1.

The buffer registers DATAIN & DATAOUT and the status flags SIN & SOUT are part of circuitry known as Device Interface.

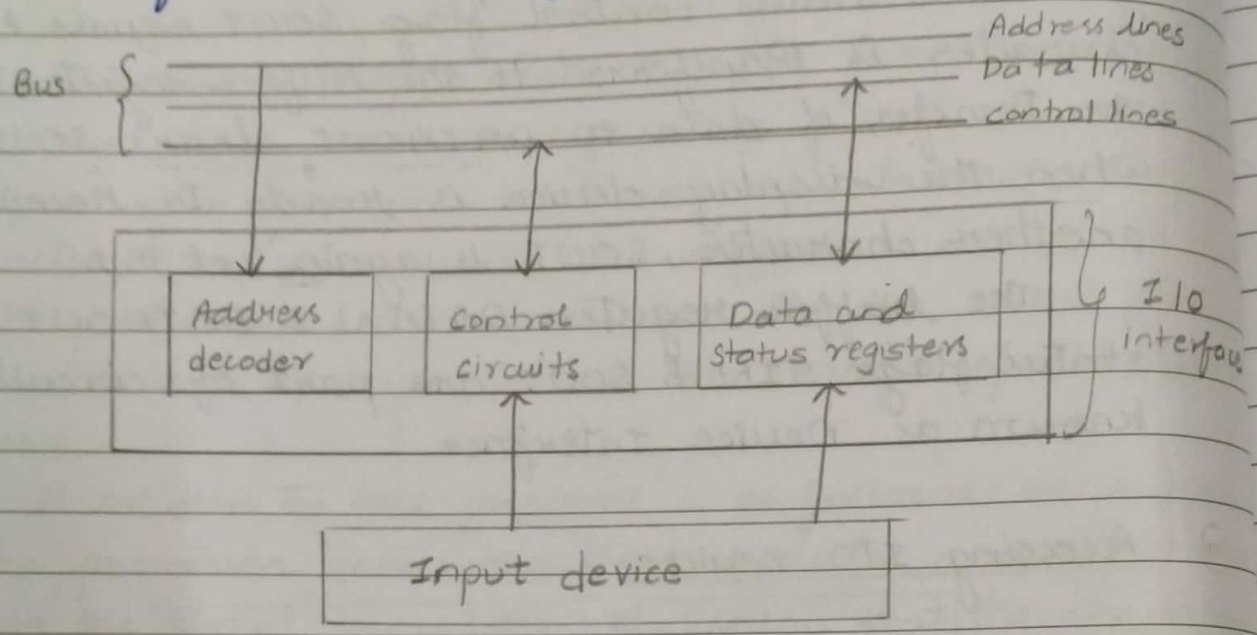
(11) Accessing I/O Devices



Multiple I/O devices may be connected to the processor and the memory via a bus. A bus enables all the devices connected to it to exchange information. Typically, it consists of three sets of lines used to carry address, data and control signals. Each I/O device is assigned a unique set of address. When the processor places a particular address on the address line, the device that recognizes this address responds to the commands issued on the control lines. The processor requests either a read or write operation, and the requested data is transferred over the data lines.

Memory Mapped I/O: When I/O devices and the memory share the same address space. In this, any

machine instruction that can access memory can be used to transfer data to or from an I/O device.



I/O interface for an input device

The address decoder enables the device to recognise its address when this address appears on the address line. The data registers holds the data being transferred to or from the processor. The status registers contains the information relevant to the operation of the I/O device. Both the data & status registers are connected to the data bus and are assigned unique addresses.

The address decoder, data & status registers and the control circuitry required to coordinate I/O transfers constitute the devices interface circuit.

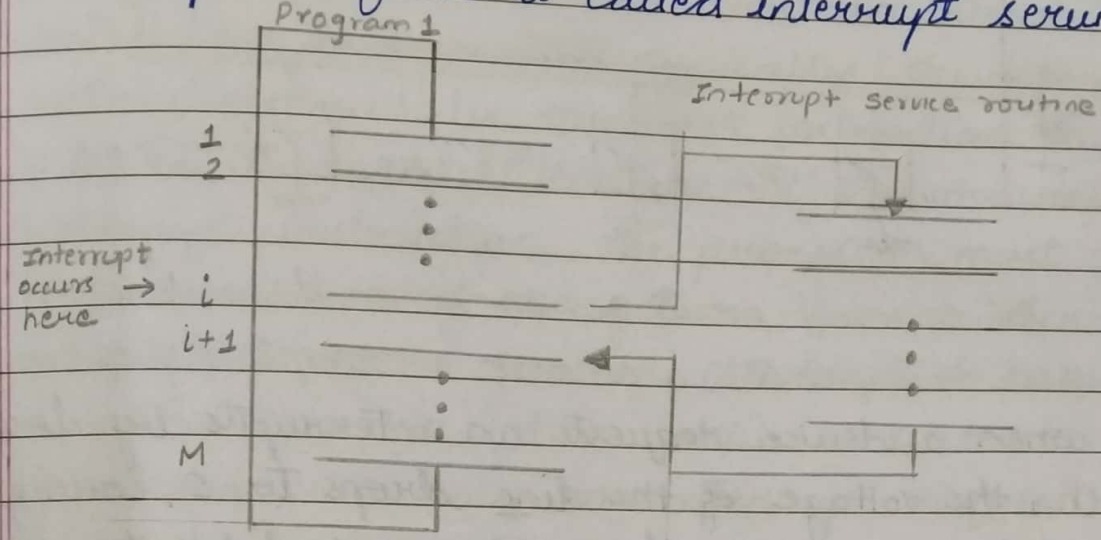
Two other mechanisms used for synchronizing data transfers between the processor and memory

a) Interrupts
b) Direct Memory Access

(12) Interrupts

There are many situations where other tasks can be performed while waiting for an I/O device to get ready. It can do so by sending a hardware signal called interrupt

to the processor. Atleast one of the bus control lines, called the interrupt request line is usually dedicated for this purpose. The routine executed in response to an interrupt request is called interrupt service routine.



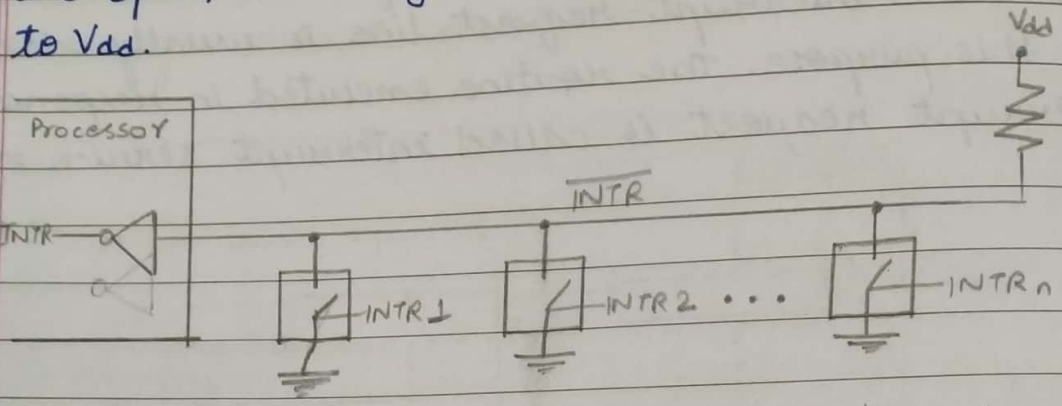
As part of handling interrupts, the processor must inform the device that its requests has been recognised so that it may remove its interrupt request signal. This may be acheived by a special control signal on the bus, called interrupt acknowledge signal. A subroutine performs a function required by the program from which it is called. Interrupt service routine may not have anything in common with the program being executed.

Interrupt latency - The process of saving and restoring registers involve memory transfers that increase the execution time. This delay is called interrupt latency.

(13) Interrupt hardware : An I/O device requests an interrupt by activating a bus line called interrupt request.

A single interrupt request line maybe used to serve n devices. All devices are connected to the line via switches to ground. To request an interrupt, a device closes its associated switch. Thus if all interrupt-request signals INTR to INTRn are inactive i.e if all switches

are open, the voltage on the request line will be equ to Vdd.



An equivalent circuit for an open drain bus used to implement a common interrupt request line

when a device requests an interrupts by closing a switch, the voltage of the line drops to 0, causing the interrupt request signal, INTR, received by the processor to go to 1. Since the closing of one or more switches will cause the line voltage to drop to 0, the value of INTR is the logical OR of the requests from individual devices i.e $INTR = INTR_1 + \cdots + INTR_n$

It is customary to use the complemented form, $\overline{INTR}$, to name the interrupt-request signal on the common line, because this signal is active when in the low voltage state. In the electronic implementation of the circuit, special gates called open-collector (for bipolar circuits) or open-drain (for MOS circuits) are used to drive the $\overline{INTR}$ line.

Resistor R is called a pull up resistor because it pulls the line voltage up to the high voltage state when the switches are open.

(14) Enabling and Disabling Interrupts

The first possibility is to have the processor hardware ignore the interrupt request line until the execution of the first instruction of the interrupt service routine has

been completed. Then by giving using an interrupt disable instruction as the first instruction in the interrupt-service routine, that no further interruptions will occur until an interrupt-enable instruction is executed. Typically, the interrupt enable instruction will be the last instruction in the interrupt service routine before the return-from interrupt instruction. The processor must guarantee that execution of the return-from-interrupt is completed before further interruption can occur.

The second option which is suitable for a simple processor with only one interrupt request line is to have the processor automatically disable interrupts before starting the execution of the interrupt service routine. After saving the contents of the PC and the processor status register (PS) on the stack, the processor performs the equivalent of executing an interrupt-disable instruction. It is often the case that one bit in the PS register, called Interrupt enable, indicates whether interrupts are enabled. An interrupt request received while this bit is equal to 1 will be accepted. After saving the contents of the PS on the stack, with the interrupt enable bit equal to 1, the processor clears the interrupt enable bit in the its PS register, thus disabling further interrupts. When a return-from interrupt instruction is executed, the contents of the PS are restored from the stack, setting the interrupt enable bit back to 1.

In the third option, the processor has a special interrupt request line for which the interrupt handling circuit responds only to the leading edge of the signal. Such a line is said to be edge triggered. In this case,

the processor will receive only one request regardless of how long the line is activated.

(15) Sequence of events involved in handling an interrupt (assuming interrupts are enabled)

- The device raises an interrupt request.
- The processor interrupts the program currently being executed.
- Interrupts are disabled by changing the control bits in the PS (except in edge triggered interrupts)
- The device is informed that its request has been recognized and in response, it deactivates the interrupt-request signal.
- The action requested by the interrupt is performed by the interrupt-service routine.
- Interrupts are enabled and the execution of the interrupted program is resumed.

(16) Handling Multiple Interrupts

Multiple I/O devices may be connected to the processor and the memory via a bus. Some or all of these devices may be capable of generating interrupt requests. Each device operates independently and hence no definite order can be imposed on how the devices generate interrupt requests.

The information needed to determine whether a device is requesting an interrupt is available in its status register. When a device raises an interrupt request, it sets to 1, one of the bits in the status register called the IRQ bit. The simplest way to identify the interrupting device is to have the interrupt-service routine poll all the I/O devices connected to the bus. The first

device encountered with its IRQ bit set is the device that should be serviced. An appropriate subroutine is called to provide the requested service.

The polling scheme is easy to implement. Its main disadvantage is the time spent interrogating the IRQ bits of all the devices that may not be requesting any service.

(17) Vectored Interrupts

To reduce the time involved in the polling process, a device requesting an interrupt may identify itself directly to the processor.

A device requesting an interrupt can identify itself by sending a special code to the processor over the bus. This enables the processor to identify individual devices even if they share a single interrupt-request line. The code supplied by the device may represent the starting address of the interrupt-service routine for that device. The code length is in the range of 4 to 8 bits.

This arrangement implies that the interrupt service routine for a given device must start at the same location. The location pointed to by the interrupting device is used to store the starting address of the interrupt service routine. The processor reads this address, called the interrupt vector, and loads it into the PC.

(18) Interrupt Nesting

When several devices are involved, in which case execution of a given interrupt-service routine, once started, always continues to completion before the

processor accepts an interrupt request from a servo device. In some cases, a long delay in responding to an interrupt request may lead to erroneous operation.

A multiple level priority organization means that during execution of an interrupt service routine, interrupt request will be accepted from some devices but not from others, depending upon the devices priority. To implement this scheme, we can assign a priority level to the processor that can be changed under program control. The priority level of the processor is the priority of the program that is currently being executed. The processor accepts interrupts only from devices that have priority higher than its own. At the time the execution of an interrupt-service routine for some device is started, the priority of the processor is raised to that of the device.

The processors priority is usually encoded in a few bits of the processor status word. It can be changed by program instructions that write into the PS. These are privileged instructions which can be executed only while the processor is running in supervisor mode. The processor is in the supervisor mode only when running operating system routines and switches to user mode to execute application programs.

An attempt to execute a privileged instruction while in the user mode leads to a special type of interrupt called a privilage exception.

P.T.O

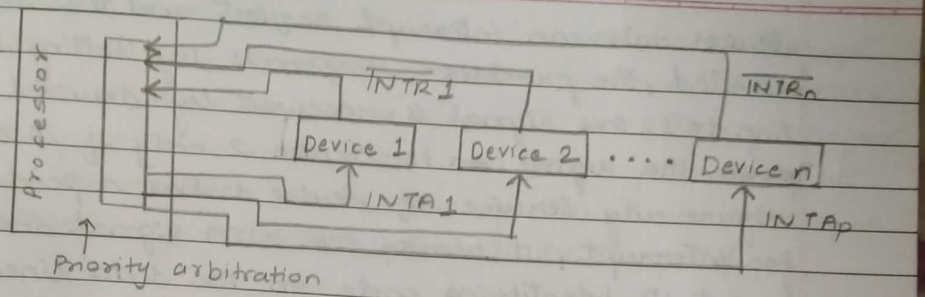


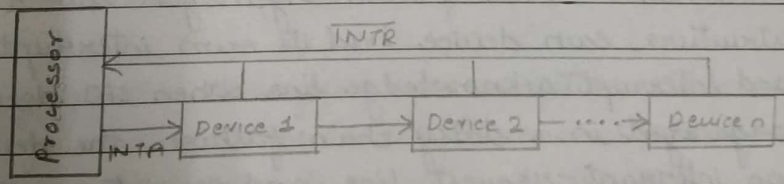
fig: Implementation of interrupt priority using individual interrupt request and acknowledge lines.

(19) Simultaneous Requests

- Polling scheme

If the processor uses a polling mechanism to poll the status registers of I/O devices to determine which device is requesting an interrupt. In this case the priority is determined by the order in which the devices are polled. The first device with status bit set to 1 is the device whose interrupt request is accepted.

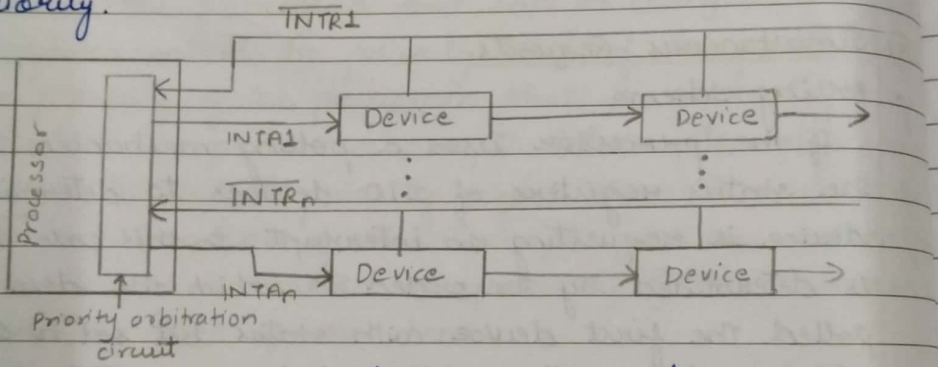
- Daisy chain scheme



To solve the problem of simultaneous arrivals of interrupts requests from two or more devices, the devices are connected to form a daisy chain.

The interrupt request line INTR is common to all devices. The interrupt-acknowledge line INTA is connected in a daisy-chain fashion, such that the INTA signal propagates serially through the devices. When several

devices raise an interrupt request and the INTR line is activated, the processor responds by setting the INTA line to 1. This signal is received by device 1. Device 1 passes the signal on to device 2 only if it does not require any service. If device 1 has a pending request for interrupt, it blocks the INTA signal and proceeds to put its identifying code on the data lines. Therefore in daisy chain arrangement, the device that is electrically closest to the processor has the highest priority.



Devices are organised in groups, and each group is connected at a different priority level. within a group devices are connected in a daisy chain.

When I/O devices were organized into a priority structure, each device had its own interrupt-request and interrupt acknowledge line. When I/O devices were organized in a daisy chain fashion, the devices shared an interrupt-request line, and the interrupt acknowledge propagated through the devices. A combination of priority structure and daisy chain scheme can also be used.

(20) **Controlling Device Request**

The control needed is usually provided in the form of an interrupt-enable bit in the devices interface circuit.

There are two independant mechanisms for controlling

(21)

interrupt requests. At the device end, an interrupt enable bit in a control register determines whether the device is allowed to generate an interrupt request. At the processor end, either an interrupt enable bit in a PS register or a priority structure determines whether a given interrupt will be accepted.

(21) Exceptions

Exception is used to refer to any event that causes an interruption.

- Recovery from Errors: Many computers contain an error checking code in their main memory, which allows detection of error in the stored data. If an error occurs, the control hardware detects it and informs the processor by raising an interrupt.

The processor may also interrupt a program if it detects an error or an unusual condition while executing the program (eg: division by zero). When exception processing is initiated as a result of such errors, the processor suspends the program being executed and starts an exception service routine.

- Debugging:- The debugger uses exceptions to provide two important facilities called trace and breakpoints.

When a processor is operating in the trace mode, an exception occurs after execution of every instruction, using the debugger program as exception service routine. The debugging program enables the user to examine the contents of registers, memory locations and so on. The trace exception is disabled during the execution of the debugging program.

Breakpoints provide a similar facility, except that the

program is interrupted at specific points selected by the user. An instruction called trap or software interrupt is usually provided for this purpose.

- Privilage Exception: To protect the operating system of a computer from being corrupted by user programs, certain instructions can be executed only when the processor is in supervisor mode. These instructions are called privilaged instructions (eg changing the priority level of the processor). An attempt to execute such an instruction will produce a privilage exception, causing the processor to switch to the supervisor mode and begin executing an appropriate routine.

* Unit - 5 [Embedded systems]

A physical system that employs computer control for a specific purpose rather than for general purpose computation, is referred to as an embedded system.

⇒ Microwave Oven

This application is based on a magnetron power unit that generates microwaves used to heat food in a confined space.

When turned on, the magnetron generates its maximum power output. Lower power levels are achieved by turning the magnetron on and off for controlled time intervals. Thus, by controlling the power level and the total heating time, it is possible to realize a variety of user-selected cooking options.

The specifications for a microwave oven may include the following cooking options:

- Manual selection of the power level and cooking time.
- A manually selected sequence of different cooking steps.
- Automatic selection where the user specifies the type of food and the weight of food. An appropriate power level and time are then calculated by the controller.

The oven includes an output display that can show:

- Decrementing clock timer while cooking
- Information messages to the user

An audio alert signal, in the form of a beep tone, is used to indicate the end of a cooking operati An exhaust fan and oven light are provided. Also, a door interlock must turn the magnetron off if the door of the oven is open.

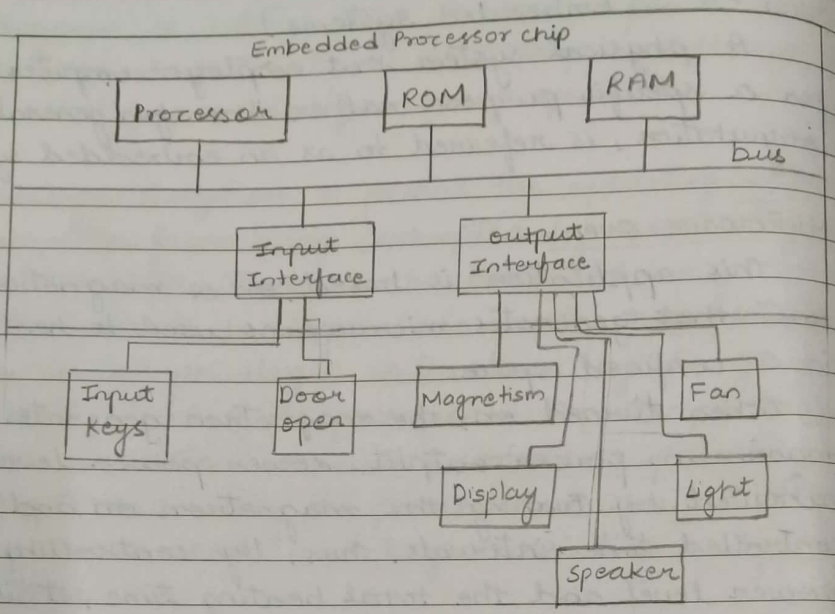


fig: Block diagram of microwave oven

The controller for a microwave oven can be implemented by a small microprocessor based computer unit. The program needed to implement the desired action is quite small. It must be stored in a non-volatile read only memory, so that it will not be lost when the power is turned off. It is also necessary to have a small RAM for use during computations and to hold the user entered data.

To find a cost effective solution to realize the desired controller, parallel I/O ports provide a convenient mechanism for dealing with the input and output signals. It is possible to realise most of this circuitry on a single VLSI chip.

⇒ Device Camera

In a digital camera, an array of optical sensors is used to capture images. These sensors are based on

photodiodes which convert light into electrical charge. The intensity of light determines the amount of charge that is generated.

Different types of sensors can be used, like charge coupled devices (CCDs) and complimentory Metal Oxide Semiconductors (CMOS). Each sensing element generates a charge that corresponds to one pixel, which is one point of a pictorial image. The number of pixels determines the quality of pictures that can be recorded and displayed.

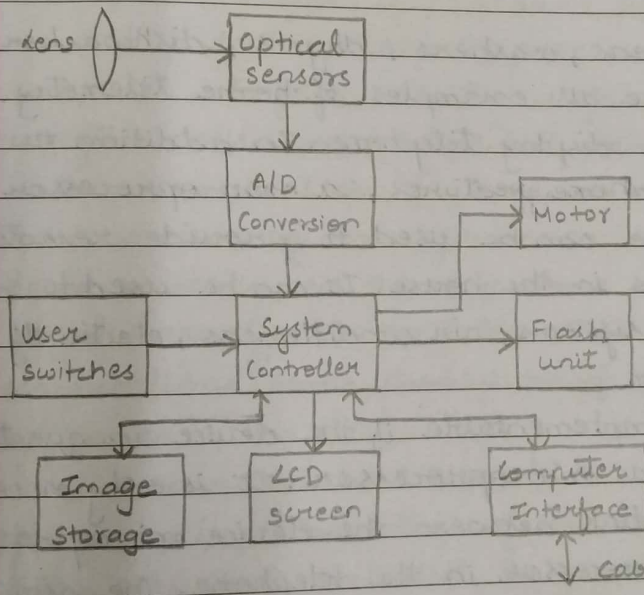


fig: Block diagram of a Digital Camera

The charge is an analog quantity, which is converted into digital representation using A/D conversion circuits. It produces a digital representation of the image in which the colour and intensity of each pixel is represented by a number of bits.

The system controller contains a processor memory (both RAM and EEPROM) and a variety of interface circuits. The processed images are stored in a larger

image storage device (flash memory, miniature hard disk drives etc).

A captured and processed image can be displayed on a LCD screen included in the camera. A standard interface provides a simple mechanism for transferring the images to a computer or a printer, via a PCI or USB. The main formats used to store processed images are TIFF for uncompressed images and JPEG for compressed images.

⇒ Home Telemetry

Microwave ovens, washers, dryers, dishwashers, air conditioners are all examples of home telemetry. Another example is the display telephone. In addition to the standard telephone features, a microprocessor controlled phone can be used to provide remote access to other devices in the house. It can be used to control home security systems, air conditioners, electricity/gas/water meters etc.

All this is implementable if the device in question is controlled by a microprocessor. It is only necessary to provide a link between the device microprocessor and the microprocessor in the telephone. The simplest way to establish such links is to use bit serial communication which is easily accomplished if the controller chips include UART interfaces.

Using signalling from a remote location to observe and control the state of equipment is referred to as telemetry.

⇒ Processor chips for Embedded Applications

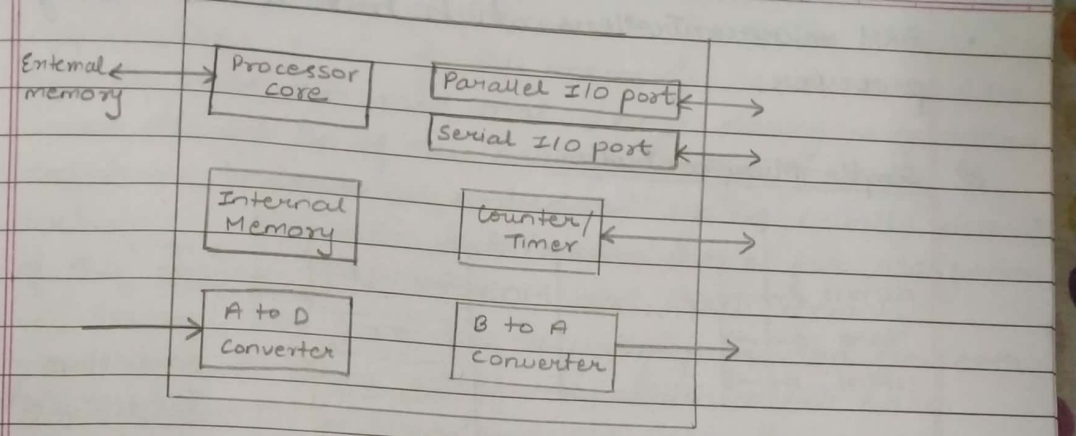


fig: Block diagram of embedded processor (Ans at back)

A chip that contains a processor, some memory and I/O interface circuitry useful in embedded applications is often called an Embedded processor/microcontroller.

The main part is a processor core, which may be a basic version of microprocessor. It is useful to include some memory on the chip. Some of this memory has to be of RAM type to hold the data that changes during computations. Some should be of ROM type to hold the software.

Both parallel and serial I/O ports are provided for easy implementation of standard I/O connections. A timer circuit is included to generate control signals at programmable time intervals. It can also be used to count the no. of pulses on a given input signal.

An embedded system may include some analog devices. To deal with them, we have a A↔D conversion circuitry. or D/A conversions circuits. (→ + back) Many embedded processor chips are available commercially. They are

- Freescale's 68HC 11 and 68K / Cold Fire Families
- Intel's 8051 and MCS-96 families, all of which have CISC style processor cores.)

- ARM microcontrollers which have a RISC - style processor.

⇒ Simple Microcontroller

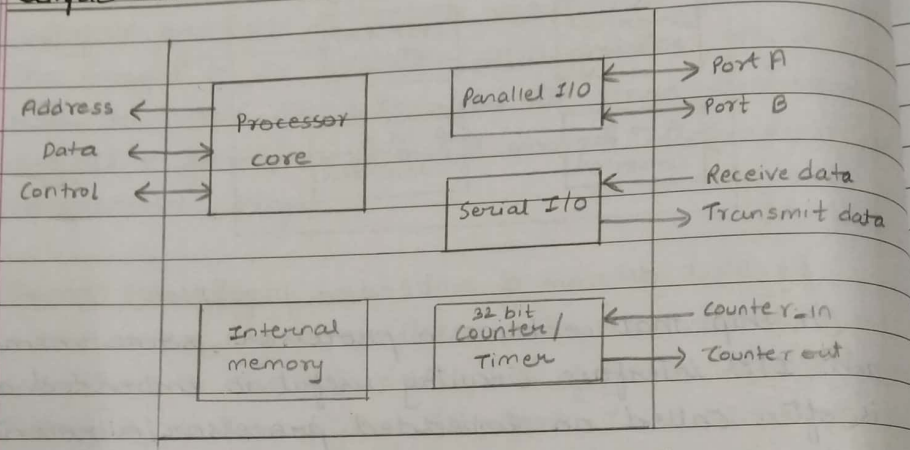


fig: an example of microcontroller

- There is a processor core and some on-chip memory. Since the on-chip memory may not be sufficient to support all potential applications, the processor bus connections are also provided on the pins of the chip so that external memory can be added.
- There are two 8-bit parallel interfaces, called port A and port B and one serial interface.
- The microcontroller also contains a 32-bit counter/timer circuit, which can be used to generate internal interrupts at programmed time intervals, to serve as a system timer, to count the pulses on an input line, to generate square wave output signals and so on.

⇒ Microcontroller chips for Embedded Applications

The main part is a processor core, which may be a basic version of a commercially available microprocessor. It includes some memory on the chip, sufficient to

satisfy the memory requirements found in small applications. Some of this memory has to be of RAM type to hold the data that change during computations. Some should be of the read-only type to hold the software, because an embedded system usually doesnot include a magnetic disk. Popular choices for realization of this storage are EEPROM and Flash memory.

Several I/O ports are usually provided for both parallel and serial interfaces, which allow easy implementation of standard I/O connections. In many applications, it is necessary to generate control signals at programmable time intervals. This task is achieved easily if a timer circuit is included in the microcontroller chip. Since the timer is a circuit that counts clockpulses it can also be used for event-counting purposes, for example to count the number of pulses generated by a moving mechanical arm or a rotating shaft.

An embedded system may include some analog devices. To deal with such devices, it is necessary to be able to convert analog signals into digital representati and vice versa. This is conveniently accomplished if the embedded controller includes A/D and D/A conversion circuits. (→ front pg)

⇒ Parallel I/O Interface

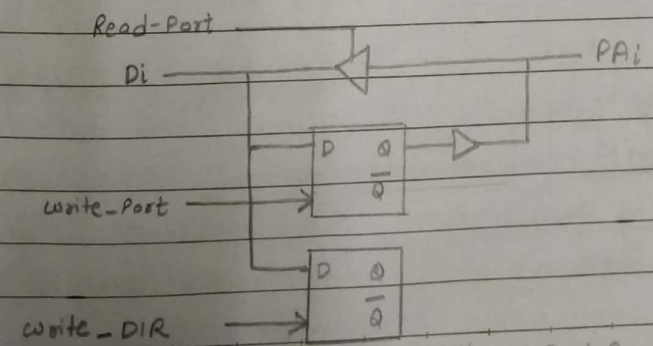


fig: Access to 1-bit in Port A

Each parallel port has an associated 8-bit DDR (Data Direction Register). DDR can be used to configure individual data lines as either input or output.

If the data direction flip flop contains a 0, then Port pin PA, is treated as an input. If the data direction flipflop contains a 1, then Port pin PA is treated as an output. Activation of control signal Read_Port, places the logic value on the port-pin onto the data line D. Activation of control-signal write_Port, places value loaded into output data flip-flop onto port-pin.

Addressable Registers are:

1) Input registers (PAIN for port A, PBIN for port B)
2) Output registers (PAOUT for port A, PBOUT for port B)
3) Direction registers (PADIR for port A, PBDIR for port B)
4) Status register (PSTAT) &
5) Control register (PCONT)

Address		
FFFFFFF0	PAIN	Port A input
FFFFFFF1	PAOUT	Port A output
FFFFFFF2	PADIR	Port A direction
FFFFFFF3	PBIN	Port B input
FFFFFFF4	PBOUT	Port B output
FFFFFFF5	PBDIR	Port B direction

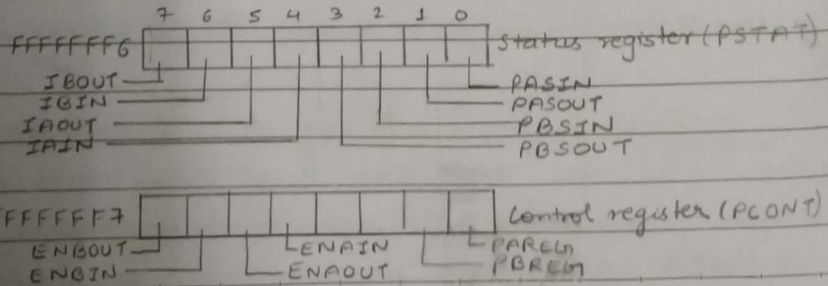


fig: Parallel interface registers

- Status Register: It provides information about the current status of input registers & output registers.
  - → PASIN = 1 → when there are new data on port A
  - → PASIN = 0 : when the processor accepts the data by reading the PAIN register.
  - → The interface uses a separate control line to indicate the availability of new data to the connected device.
  - → PASOUT = 1 : when the data in register PAOUT are accepted by the connected device.
  - → PASOUT = 0 : when the processor writes data into PAOUT.

  Like PASIN & PAOUT, the flags PBSIN and PBSOUT perform the same function on port B. The status register also contains four interrupt flags. They are IAIN, IAOUT, IBIN & IBOUT.
  - → IAIN = 1 : when interrupt is enabled and the corresponding I/O action occurs.
  - → ENAIN = 1 : when the corresponding interrupt is enabled. Ex: If ENAIN = 1 & PASIN = 1, then interrupt flag IAIN is set to 1 and an interrupt request is raised

- Control Register: It is used for controlling data transfers to/from the devices connected to ports A/B. Port A has two control lines CAIN and CAOUT. They are used to provide an automatic signaling mechanism between interface and attached device.

⇒ Serial I/O Interface

The serial interface provides the UART (Universal Asynchronous Receiver/Transmitter) capability to transfer data. Double buffering is used in both the transmit and receive paths, and is needed to handle bursts in I/O transfers correctly.

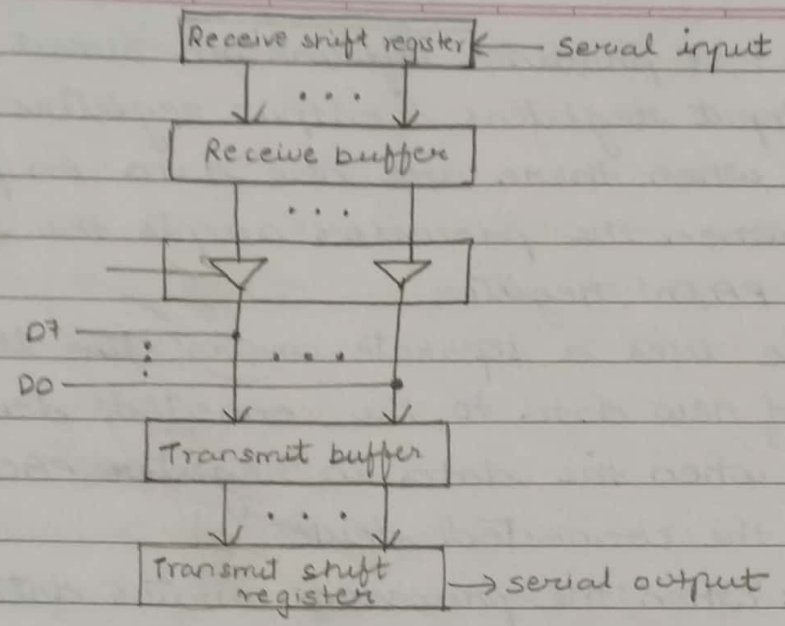


fig: Receive and transmit structure of the serial interface

Addresable Registers are

1) Receive Buffer
2) Transmit buffer
3) status-register (SSTAT)
4) control register (SCONT)
5) clock divisor register (DIV)

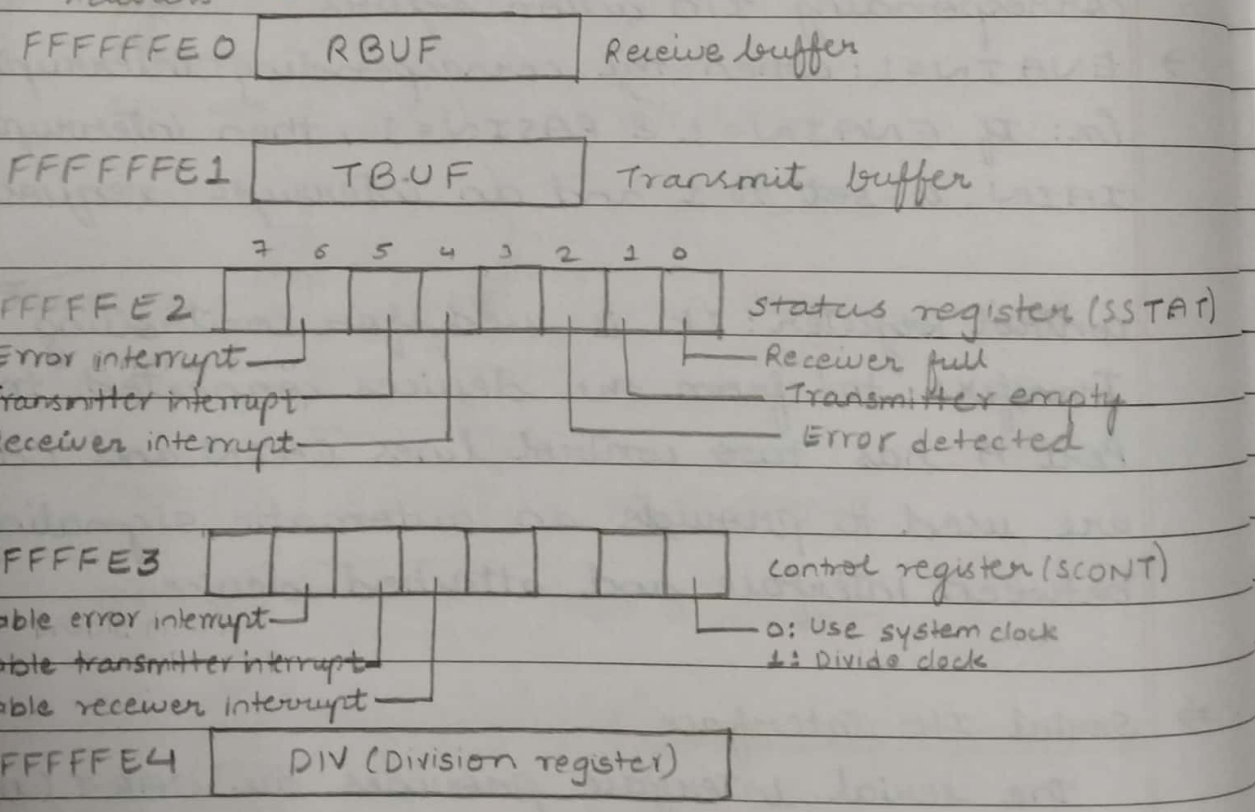


fig: Serial interface registers

Input data are read from the Receive buffer and output data are loaded into the transmit buffer.

- Status Register (SSTAT) provides information about the

current status of receive units and transmit units.

→ Bit SSTAT0 = 1 : when there are new data in the receive buffer.

→ Bit SSTAT0 = 0 : when the processor accepts the data by reading the receive buffer.

→ SSTAT1 = 1 : when the data in transmit buffer are accepted by the connected device.

→ SSTAT1 = 0 : when the processor writes data into transmit buffer.

→ SSTAT2 = 1 : If an error occurs during the receive process. The status register also contains the interrupt flags.

→ SSTAT4 = 1 : when the receive buffer becomes full and the receiver-interrupt is enabled.

→ SSTAT5 = 1 : when the transmit-buffer becomes empty & the Transmitter-interrupt is enabled.

- Control Register (SCONT) is used to hold the interrupt enable bits.

→ If SCONT6-4 = 1
Then the corresponding interrupts are enabled.
otherwise, the corresponding interrupts are disabled.

→ If SCONT0 = 0
Then, the transmit clock is same as the processor clock.

- Clock divisor register (DIV) divides system-clock signal to generate the serial transmission clock.

Programming considerations

The microcontroller under consideration is used to transfer 8-bit characters from a bit serial source to a bit-parallel destination. The source is connected to the serial interface and the destination is connected to parallel port.

The following two approaches can be used:

a) Polling Approach

It involves testing a status flag repeatedly until a character is received.

```
#define RBUF (volatile char *) 0XFFFFFFE0
#define SSTAT (volatile char *) 0XFFFFFFE2
#define PAOUT (char *) 0XFFFFFFF1
#define PADIR (char *) 0XFFFFFFF2
void main( )
{
  *PADIR = 0XFF;
  while (1)
{
  while ((*SSTAT & 0x1) == =0);
   *PAOUT = *RBUF;
}
 }
```

In a C language program, a memory mapped I/O location can be represented using a pointer variable, where the value of the pointer is the address of the location. If the contents of this location are to be treated as a character, the pointer should be declared to be of character type. This defines the contents as being one byte in length, which is the size of I/O registers.

⇒ Interrupt Approach

In this approach, instead of polling the bit SSTAT_0 to detect a new character, we can configure the I/O interface to raise an interrupt request when SSTAT_0=1. The corresponding interrupt enable bit in the SCONT register, SCONT_4, has to be set to 1. It is also necessary to enable IRQ interrupts in the processor

by setting PSR6 to 1.

```
# define RBUF (volatile char *) 0*FFFFFFE0
# define SCONT (char *) 0XFFFFFFE3
# define PAOUT (char *) 0XFFFFFFF1
# define PADIR (char *) 0XFFFFFFF2
#define int_addr (int *) (0X24)
void intserw();
void main()
{
 *PADIR = 0XFF;
 int_addr = &intserw;
 __asm__ ("Move #0X40, % PSR");
 *SCONT = 0X10;
 while(1);
 3
 void intserw()
 {
 *PAOUT = *RBUF;
 __asm__ ("Return I");
 3
```