Digital Design and Computer Organization – BC302
Module 4

35
LAKSHMI R
B.E., M.Tech (CNE)
Assistant Professor
Department of CSE

INPUT/OUTPUT ORGANIZATION

ACCESSING I/O DEVICES

Single bus arrangement – to connect I/o devices to a computer

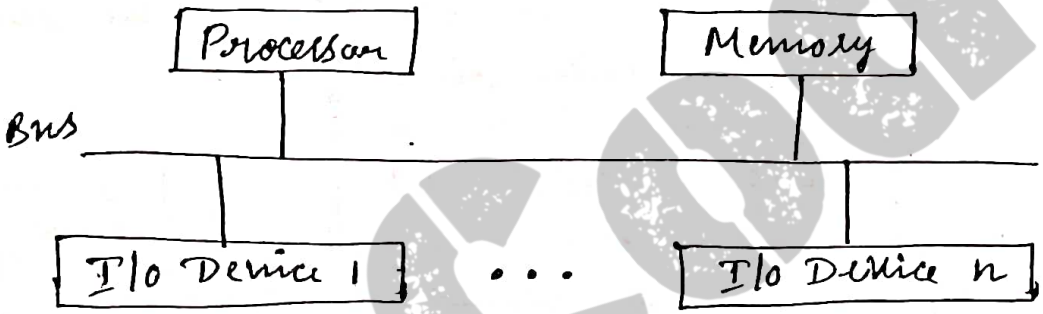


Fig 2.1 : Single bus structure

→ The bus enables all devices connected to it to exchange information.
→ Three sets of lines to carry – address
– data
– control signals
→ Each I/o device is assigned a unique set of addresses.
→ When the processor places a particular address on the address lines, the devices that recognizes that address responds.
→ The processor requests either read or write operation and this command is sent through control lines.
– Data to be read or written are sent through data lines

LAKSHMI R
B.E., M.Tech (CNE)
Assistant Professor
Department of CSE

Memory-mapped I/O

→ I/O devices and the memory share the same address space.

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

→ Eg:- DATAIN is the address of the input buffer associated with keyboard
DATAOUT is the address of the output buffer associated with display unit

```
MOVE   DATAIN, R0
```

{ Reads data from DATAIN buffer & moves the read data to register R0

```
MOVE   R0, DATAOUT
```

{ Writes the contents of register R0 to O/P buffer DATAOUT

I/O-mapped I/O

→ I/O devices have special I/O address space or incorporate them as part of the memory address space

→ Advantage: I/O devices deal with fewer address lines

→ Separate I/O address space does not necessarily mean that I/O address lines are physically separate from the memory address lines.

→ Some processors have special IN and OUT instructions to perform I/O transfers.

→ Fig 2.2 illustrates the hardware required to connect an I/O device to the bus.

LAKSHMI R
B.E., M.Tech (CNE)
Assistant Professor
Department of CSE

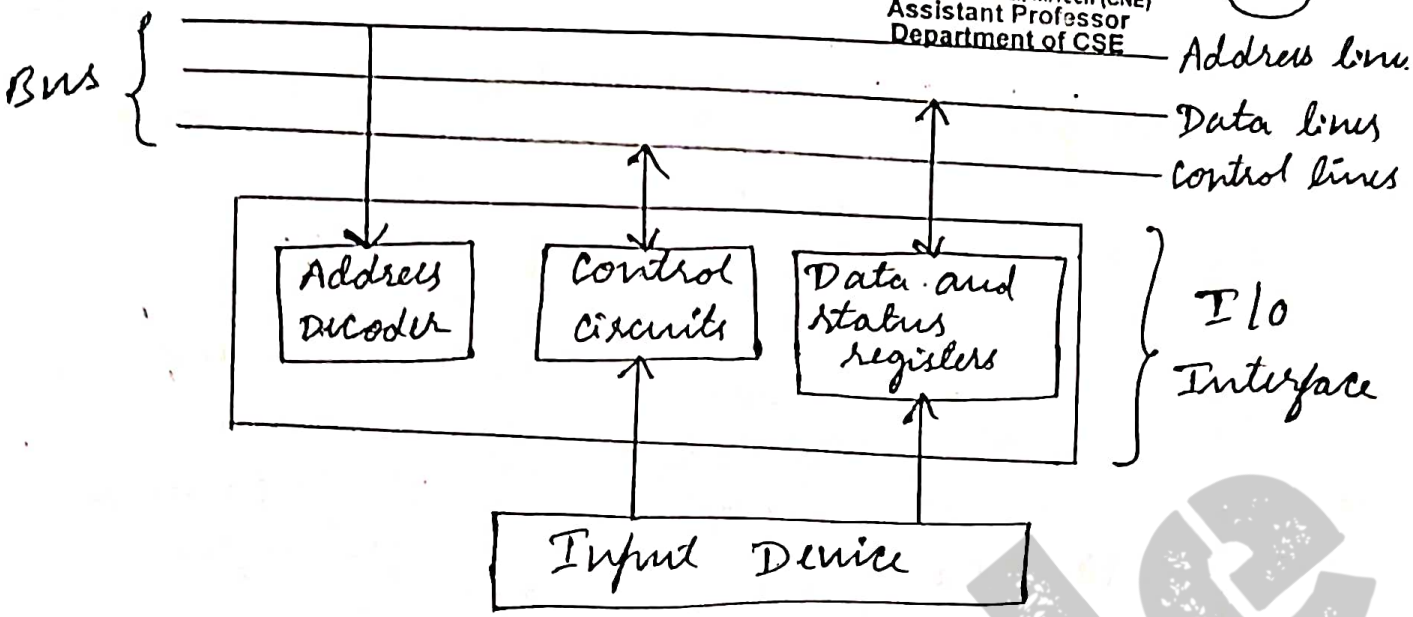


Fig 2-2 : I/O Interface for an Input Device

→ The address decoder enables the device to recognize its address when this address appears on the address lines.
→ The data register holds the data being transferred to or from the processor.
→ The status register contains information relevant to the operation of the I/O device.
→ The control circuitry coordinate I/O transfers.
→ All the above components constitute the device's interface circuitry.

→ The operating speeds of I/O devices and processor differ vastly.
→ I/O devices – slower : When a human operator is entering characters at a keyboard, the processor is capable of executing millions of instructions.

LAKSHMI R
B.E., M.Tech (CNE)
Assistant Professor
Department of CSE

EXAMPLE

I/o operations involved in a keyboard and a display device.

→ Registers involved are:

1) STATUS : Control flags used are:
   a) SIN → Status information for keyboard
   b) SOUT → " " display.
   c) KIRQ } used in conjunction
   d) DIRa } with interrupts.
2) CONTROL : Control flags.
   a) DEN → Display enable interrupt
   b) KEN → keyboard enable interrupt.
3) DATAIN : Data from the keyboard are made available in this register.
4) DATAOUT : Data sent to the display are stored in this register.

Program: To read a line of characters from keyboard and echo it back to display.

```
        MOVE  #LINE, R0          -> Initialize memory pointer
WAITK   TestBit #0, STATUS       -> Test SIN
        Branch=0 WAITK           -> WAIT for character to be entered
        MOVE  DATAIN, R1         -> Read character
WAITD   TestBit #1, STATUS       -> Test SOUT
        Branch=0 WAITD           -> Wait for display to become ready
        MOVE  R1, DATAOUT        -> Send character to display
        MOVE  R1, (R0)+          -> Store character & increment pointer
        Compare #$0D, R1         -> Check if carriage return
        Branch≠0 WAITK           -> If not, get another character
        MOVE #$0A, DATAOUT       -> otherwise, send line feed
        Call  PROCESS            -> Call subroutine to process the
                                    input line.
```

LAKSHMI R
B.E., M.Tech (CNE)
Assistant Professor
Department of CSE

Mechanisms to implement I/O operations

i) **Polling**: The example stated above illustrates program-controlled I/O
- Here, the processor checks a status flag to achieve the required synchronization b/w the processor and I/O device.
- We say that the processor polls the device.

ii) **Interrupts**: Synchronization is achieved by having I/O device send a special signal over the bus whenever it is ready for a data transfer.

iii) **DMA**: Direct Memory Access
- For high speed devices.
- Device interface transfer data directly to or from memory, without continuous involvement by the processor.

INTERRUPTS

- There are many situations where other tasks can be performed while the processor is waiting for an I/O device to become ready.

→ So, we can arrange for the I/O device to alert the processor when it becomes ready.

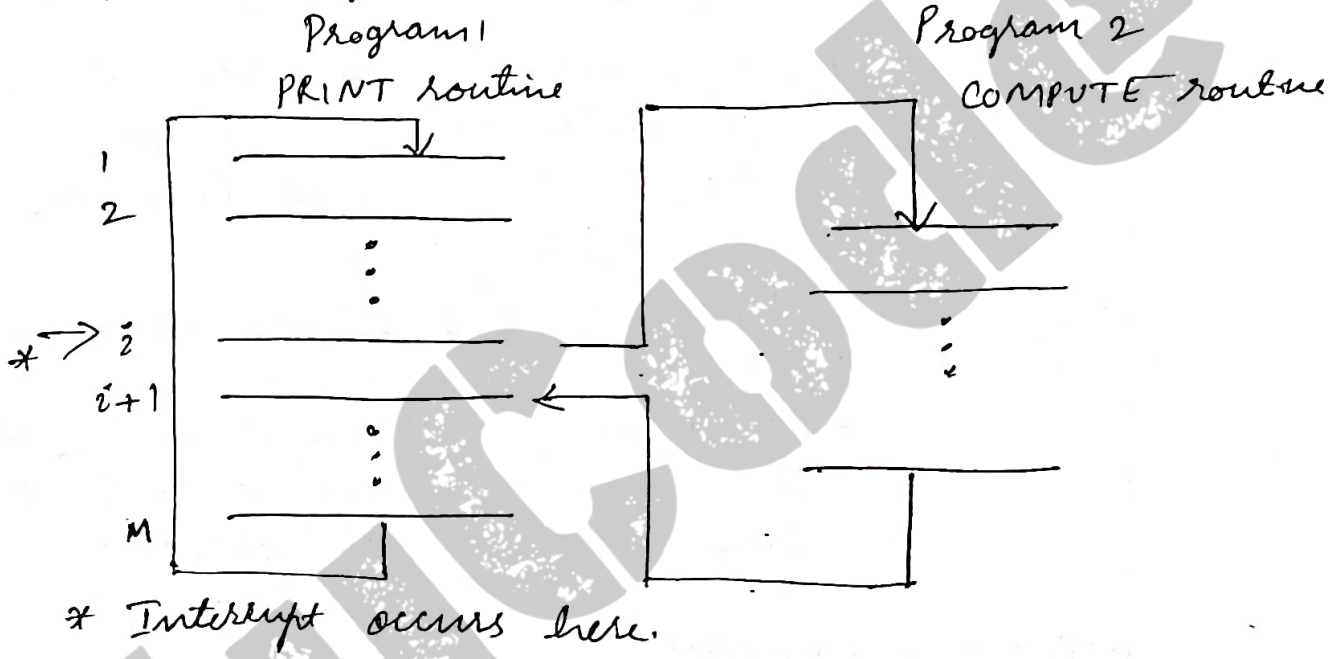
→ This can be done by sending a hardware signal called the interrupt to the processor.

- This signal is sent through the bus control lines called an interrupt-request line.

LAKSHMI R
B.E., M.Tech (CNE)
Assistant Professor
Department of CSE

Example: Consider a task that requires some computation to be performed and results to be printed on a line printer.
Program consists of 2 routines: COMPUTE & PRINT

COMPUTE produces a set of 'n' lines of output to be printed by the PRINT routine.



* Interrupt occurs here.

Fig 2.3: Transfer of control through the use of Interrupts.

→ COMPUTE is executed to produce 'n' lines of output
→ PRINT routine is executed to send the first line of text to the printer.
→ Instead of waiting for ~~print~~ the line to be printed, PRINT routine may be temporarily suspended and execution of COMPUTE routine executed.
→ When the printer becomes ready, it alerts the processor by sending an interrupt-signal.
→ In response, the processor interrupts the execution of the COMPUTE routine and transfers control to the PRINT routine.

LAKSHMI R. T
B.E., M Tech (CNE)
Assistant Professor
Department of CSE

→ The PRINT routine sends second line to the printer.
→ This process continues until all the lines have been printed.

~~~

→ The routine executed in response to an interrupt request is called the interrupt-service routine (ISR)
→ ISR is similar to subroutines.

→ Consider Fig 2.3. Assume that an interrupt request arrives during execution of instruction i.
→ Processor first completes execution of i, then it loads the PC with the address of the first instruction of the ISR.
→ After the execution of the ISR, the processor has to come back to instruction $i+1$
→ ∴ When an interrupt occurs, the current contents of PC, which points to instr $i+1$, must be stored in known location.
→ This return address is stored in processor stack
→ The return from ISR reloads the PC from stack and resumes execution at $i+1$.

→ An interrupt-acknowledge signal is send by processor to inform the device that its request has been recognized so that it may remove its interrupt-request signal.

→ Interrupt-latency is the delay that is caused between the time an interrupt request is received
~~~

and the start of the ISR. The delay is caused due to the saving of information and register contents before the processor starts executing ISR.

→ Interrupts enable transfer of control from one program to another to be initiated by an event external to the computer.

→ Execution of interrupted program resumes after the complete execution of ISR.

INTERRUPT HARDWARE

→ I/o devices request interrupt by activating a bus line called interrupt-request.

→ A single interrupt-request line may be used to serve 'n' devices as shown in figure.

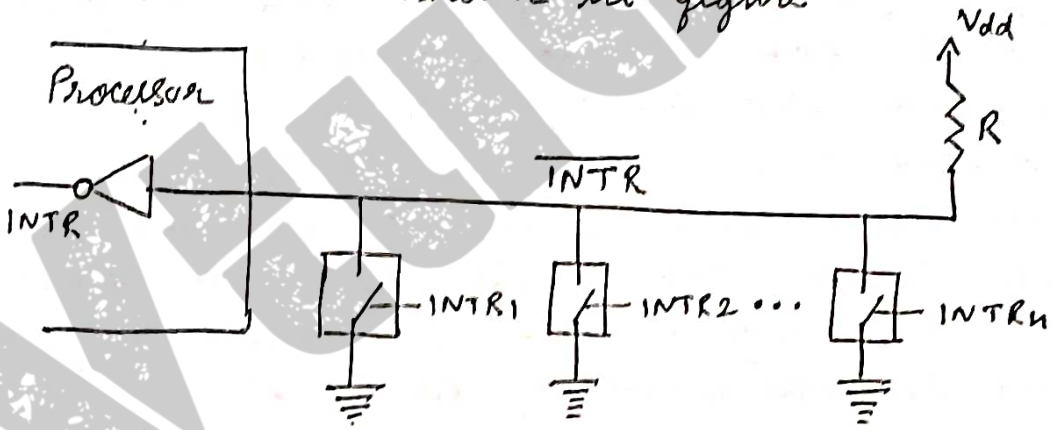


Fig 2.4: An equivalent circuit for an open-drain bus used to implement a common interrupt-request line

→ To request an interrupt, a device closes its associated switch.

→ All devices are connected to $\overline{INTR}$ line via switches to ground.

LAKSHMI R
B.E, M.Tech (CNE)
Assistant Professor
Department of CSE

→ If all the signals $INTR_1$ to $INTR_n$ are inactive, that is, if all the switches are open, the voltage on the interrupt-request line (INTR) will be equal to Vdd.

→ When a device requests an interrupt by closing its switch, the voltage on the line drops to 0.

→ This causes the interrupt-request signal, INTR, received by the processor to go to 1.

∴ Voltage on INTR line = Vdd — Inactive state
Voltage on INTR line = 0 — active state.

→ Closing of one or more switches will cause the line voltage to drop to zero.

→ ∴ the value of INTR is the logical OR of the requests from individual devices.

$$INTR = INTR_1 + INTR_2 + \cdots + INTR_n$$

→ ∴ $\overline{INTR}$ is active-low signal. (active when in the low voltage state)

→ Open-drain gate – used to drive INTR line.

- The o/p of open-drain gate is equivalent to a switch to ground that is open when the gate's input is in the '0' state and closed when it is in the '1' state.

→ Pull-up resistor – pulls the line voltage to the high-voltage state when the switches are open.

LAKSHMI
B.E., M.Tech (CNE
Assistant Professor
Department of CSE

ENABLING AND DISABLING INTERRUPTS

Why the ability to enable and disable interrupts are desired?

1) The arrival of interrupt request may occur at any time and they may alter the sequence of events that are given in the user program. Therefore, interruption of the program must be carefully controlled.
2) It may be necessary to guarantee that a particular sequence of instructions is executed to the end w/o interruption because the ISR may change some of the data used by the instructions.

→ When a device activates the interrupt-request signal, it keeps the signal active until the execution of the ISR or at least until an instruction is reached that accesses the device in question.

→ It is essential to ensure that this active request signal does not lead to successive interruptions. This may lead to system enter an infinite loop from which it cannot recover.

What are the different ways to address this problem?

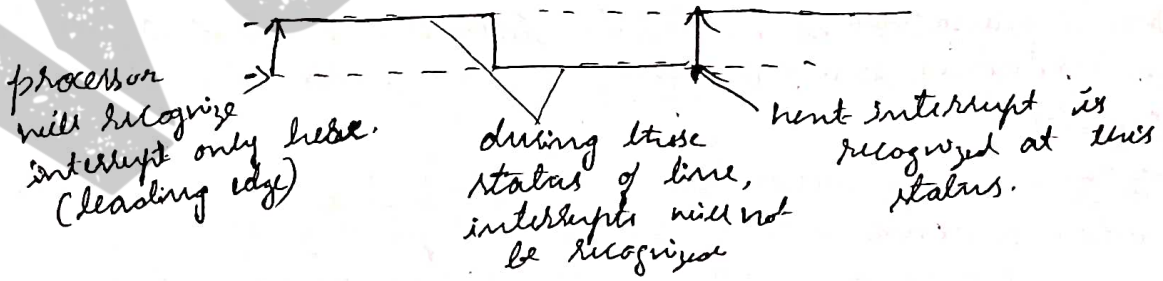
1) ISR to have the initial interrupt disable instr.
   - Processor hardware ignores the INTR line until the first instruction of ISR has been completed.
   - Interrupt-disable instruction is the first instruction in the ISR.
   - No further interruptions will occur until an interrupt-enable instruction is executed. (This will be the last instruction in ISR before return-from-interrupt instr.)

ii) Using Interrupt-enable bit in Processor status (PS) register.

- Processors with only one interrupt line.
- Automatically disable interrupts before starting the execution of ISR.
- Interrupt-enable bit in PS register
  If this bit = 0 – interrupts are not accepted
  = 1 – interrupts are accepted.
- Before starting ISR, interrupt-enable bit will be 1 in PS. (Auto enable = 1)
- The contents of PS and PC will be moved to stack before starting the ISR and then interrupt-enable bit is cleared by the processor, thus disabling further interrupts.
- When a return-from-instr is executed, the contents of PC and PS are restored from the stack, setting the interrupt-enable bit back to 1.

iii) Edge-triggered

→ Processor has a special interrupt line.
→ The interrupt-handling circuit responds only to the leading edge of this signal line.
→ Such a line is said to be edge-triggered.



→ Processor will receive only one request, regardless of how long the line is activated.
→ No danger of multiple interruptions and no need to explicitly disable-interrupt requests from this line.

LAKSHMI K
B.E., M.Tech (CN)
Assistant Professor
Department of CSE

The sequence of events involved in handling an interrupt requests from a single device.

1) The device raises an interrupt request.
2) The processor interrupts the program currently being executing.
3) Interrupts are disabled by changing the control bits in the PS (not in the case of edge-triggered)
4) The device is informed that its request has been recognized, and in response, it deactivates the interrupt-request signal.
5) The action requested by the interrupt is performed by the ISR.
6) Interrupts are enabled and execution of the interrupted program is resumed.

HANDLING MULTIPLE DEVICES

-> Situation where a number of devices capable of initiating interrupts are connected to the processor.
-> If two devices have activated the line at the same time, it must be possible to break the tie and select one of the two requests for service.
-> When one is completed, the second request can be serviced.
-> The information is needed to determine whether a device is requesting an interrupt is available in <u>status register</u>.
-> <u>IRQ</u> - one of the bits in <u>status register</u> is set to '1' when a device raises an interrupt signal.
-> Eg:- KIRQ and DIRQ are interrupt request bits for keyboard and display respectively.

-> The simplest way to identify the interrupting device is to have the ISR 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.
→ The main disadvantage of polling is the time spent interrogating the IRQ bits of all the devices that may not be requesting any device.
→ Alternative approach – Vectored interrupts.

Vectored Interrupts

→ To reduce the time involved in the polling process, a device requesting an interrupt may identify itself directly to the processor.
→ Identification – by sending a special code to the processor over the bus.
→ This is how the processor identify individual devices.
→ The code supplied by the device may represent the starting address of the ISR for that device.
→ The code length is typically in the range of 4 to 8 bits
→ The remainder of the address is supplied by processor.
→ ISR for a given device must always start at the same location.
→ This address is called the interrupt vector and the processor loads this address into the PC.

→ INTA line : When the processor is ready to receive the interrupt-vector code, it activates the interrupt-acknowledge line – INTA
→ After receiving INTA, device turns off the INTR signal

Interrupt Nesting

→ Interrupt should be disabled during the execution of an ISR.
→ Execution of ISR once started, always to continues to completion before accepting any other requests from other devices.

LAKSHMI K
B.E, M.Tech (VTU)
Assistant Professor
Department of CSE

→ For some devices, a long delay in responding to an interrupt request may lead to erroneous operation. Eg:- Real-time clock in computers.

→ It may be necessary to accept an interrupt request from the clock during the execution of an ISR for another device.

→ I/o devices should be organized in a priority structure.

→ An interrupt request from a high-priority device should be accepted while the processor is serving another request from a lower-priority device.

→ At the time the execution of an ISR for some device is started, the priority of the processor is raised to that of the device.

→ This action disables interrupts from devices at the same level or lower. Interrupts from higher-priority devices will be accepted.

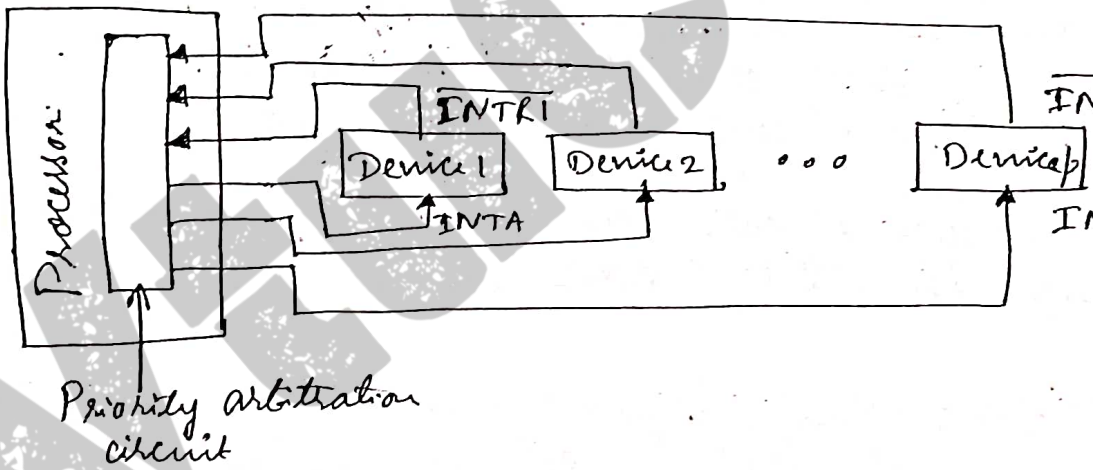


Fig 2.5: Implementation of interrupt priority using individual interrupt-request and acknowledge lines.

→ Each of the INTR lines is assigned a different priority level.

→ Requests received over these lines are sent to a priority arbitration circuit in the processor.

LAKSHMI K
B.E., M.Tech (CNE)
Assistant Professor
Department of CSE

Simultaneous Requests

→ Problem when two or more devices send the interrupt request signals. Which one to service first?
→ Highest priority device will be serviced first.
→ If several devices share one interrupt-request line, connect the devices to form a daisy chain
→ $\overline{INTR}$ line is common to all devices.
→ INTA line is connected in a daisy chain fashion such that the INTA signal propagates serially through the devices.
→ When several devices raised an $\overline{INTR}$ signal, signal line is activated, the processor responds by setting the INTA signal to '1'.
→ This signal is received by device 1. Device 1 passes the signal on to device 2 if device 1 does not require any service.
→ If device 1 has any pending request, it blocks INTA signal and puts its identity code on to data lines.
→ ∴ in daisy-chain arrangement, the device that is electrically closest to the processor has the highest priority.

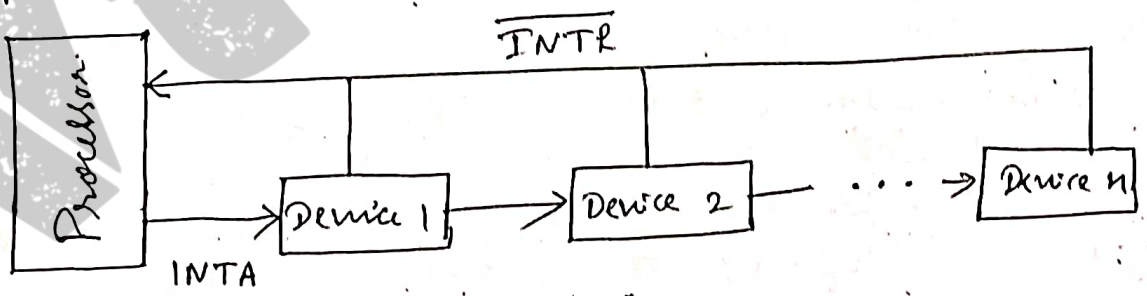


Fig 2-6: Daisy chain

→ The disadvantage of scheme dipicted in Fig 2.5 is that it allows the processor to accept request from some devices but not from others depending upon their priorities.

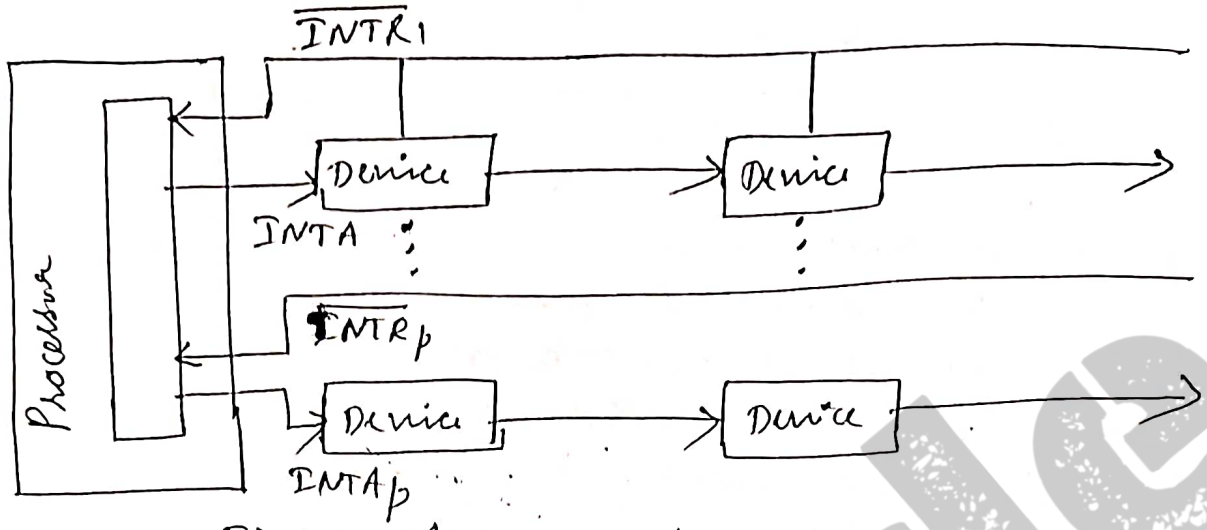


Fig 2.7: Arrangement of priority groups

→ The two schemes (Fig 2.5 and Fig 2.6) may be combined to produce the more general structure as shown in Fig. 2.7.

→ Devices are organized in groups, and each group is connected at a different priority level.

→ Within a group, devices are connected in a daisy chain.

EXCEPTIONS

→ The term exception is often used to refer to any event that causes an interruption.

Recovery from errors:

→ Error checking code to detect errors 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 instructions.

→ Eg:- i) OP-code field does not correspond to legal instruction
ii) Arithmetic instruction may attempt a division by zero

→ When exception processing is initiated as a result of such errors, the processor proceeds in exactly the same manner as in the case of an I/O interrupt request.

→ It suspends the program being executed and starts an Exception-Service Routine.

→ This routine takes appropriate action to recover from the error (if possible), or to inform the user about it.

Debugging

→ Debugger program - helps the programmer find errors in the program.

→ The debugger uses exceptions to provide two important facilities called trace and breakpoints.

Trace mode: When processor is operating in the trace mode, an execution exception occurs after execution of every instruction.

→ Followed by this, exception-service routine is run by debugger program.

→ The debugging prgm enables the user to examine the contents of registers, memory locations, and so on.

→ On return from the debugger program, the next instruction in the program being debugged is executed.

Breakpoints: Provide similar facility as that of trace, except that the program is interrupted only at specific points selected by the user. - breakpoint

LAKSHMI R
B.E., M.Tech (CNE)
Assistant Professor
Department of CSE

→ user may wish to interrupt program execution after instruction i.
→ The debugging routine saves instr^n '$i+1$' and replaces it with a s/w interrupt instr^n, and debugging routine is activated.
→ This gives the user a chance to examine memory and register contents.

Privilege Exception

→ Supervisor mode: Mode in which the processor is executing operating system program.
→ These instructions are called privileged instructions.
→ User mode: Mode in which the processor is executing user programs.
→ When ~~ps~~ processor is running in user mode and tries to execute privileged instructions, then privilege exception is produced.

Situations where privilege exception is produced:
i) In user mode, processor trying to execute instr^n that changes priority level of the processor.
ii) User program attempting to access areas in the computer memory that have been allocated to other users.

LAKSHMI R
B.E., M.Tech (CNE)
Assistant Professor
Department of CSE

DIRECT MEMORY ACCESS (DMA)

→ To transfer large blocks of data at high speed.

→ A special control unit may be provided to allow transfer of a block of data directly between an external device and the main memory, without continuous intervention by the processor.

→ This approach is called DMA.

→ DMA controller: Control circuit that is part of the I/O device that performs DMA transfers.

→ DMA controller performs the functions that would normally be carried out by the processor when accessing the main memory.

→ DMA controller must increment the memory address for successive words and keep track of the number of transfers.

→ Data transfer is done without the intervention of by processor, but under the control of the processor.

→ To initiate the block of data transfer, the processor sends:
- Starting address
- no. of words in the block
- direction of the transfer

→ When the entire block of data is transferred, the controller informs the processor by raising an interrupt signal.

→ While DMA transfer is taking place, the program that requested the transfer cannot continue. and the processor can be used to execute another program.

→ DMA control registers – accessed by the processor to initiate transfer operations.

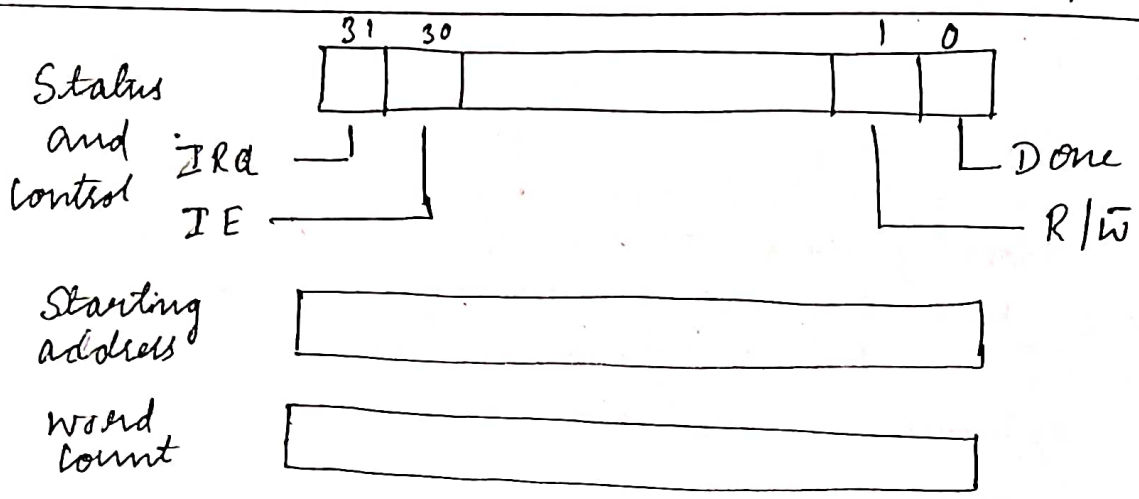


Fig 2.8: Registers in DMA controller

Status and control:

R/W: Determines the direction of the transfer
0 → Write operation (from I/O device to memory)
1 → Read operation (from memory to I/O device)

Done: 0 → transfer is going on
1 → Controller has completed transfer and ready to receive another command.

IE: Interrupt Enable flag
When IE=1, it causes the controller to raise an interrupt after the completion of the transfer.

IRQ: Controller sets IRQ to 1 when it has requested an interrupt.

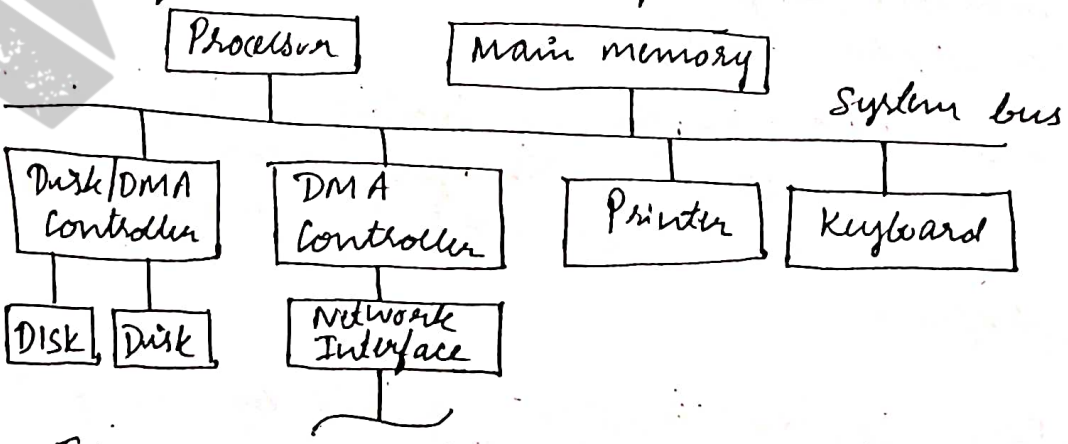


Fig 2.9: Use of DMA controllers in a computer system.

→ DMA controller connects a high-speed network to the computer bus.
→ Disk controller has DMA capability and controls two disks.
→ It provides two DMA channels.
→ It can perform two independent DMA operations.
→ All DMA interface registers are duplicated.

How the block of data transfer carried out?

Block of data transfer from main memory to a disk.

→ Program writes the address and word count information into the registers of the corresponding channel of the disk controller.
→ The DMA controller proceeds independently to implement the specified operation.
→ When the transfer is completed, this fact is recorded in the status and control registers of DMA channel by setting the 'Done' bit
→ If IE bit is set, the controller sends an interrupt request to the processor and sets the IRQ bit.
→ The status register can also be used to record other information - whether the transfer took place correctly or error occurred.

Cycle stealing

→ Memory accesses by the processor and DMA controller are interwoven.
→ Processor originates most of the memory access cycles.
→ The DMA controller is said to 'steal' memory access cycles from the processor.
→ Hence, this interweaving technique is called 'cycle stealing'.

Assistant Professor
Department of CSE

→ Alternatively, the DMA controller may be given exclusive access to the main memory to transfer a block of data without interruption. This is known as Block or burst mode.

BUS ARBITRATION

→ A conflict may arise if both the processor and a DMA controller or two DMA controllers try to use the bus at the same time to access main memory.

→ To resolve these conflicts, arbitration procedure is implemented.

→ The device that is allowed to initiate data transfers on the bus at any given time is called the bus master

→ When the current master relinquishes (releases) control of the bus, another device can acquire status.

→ Bus arbitration is the process by which the next device to become master is selected and bus mastership is transferred to it.

→ Two approaches → Centralized
→ Distributed

Centralized arbitration – a single bus arbiter performs the required arbitration.

Distributed arbitration – all devices participate in the selection of the next bus master.

B.E., M.Tech (CNE)
Assistant Professor
Department of CSE

CENTRALIZED ARBITRATION

→ Bus arbiter may be the processor or a separate unit connected to the bus.

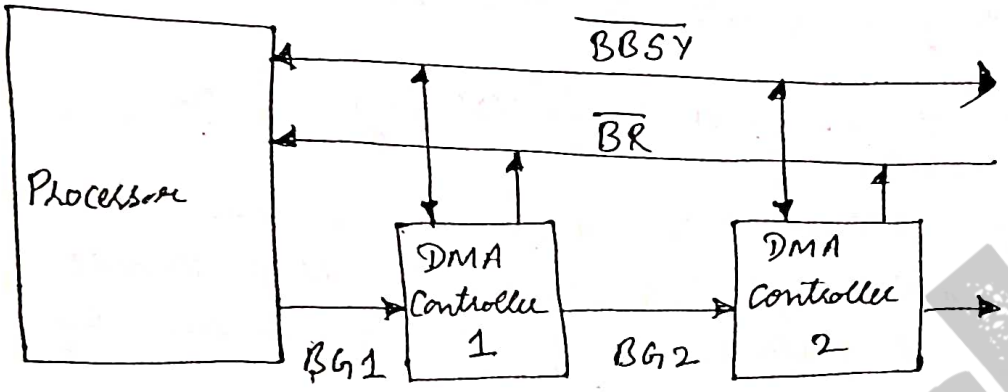


Fig 2.10: Bus Arbitration using a daisy chain

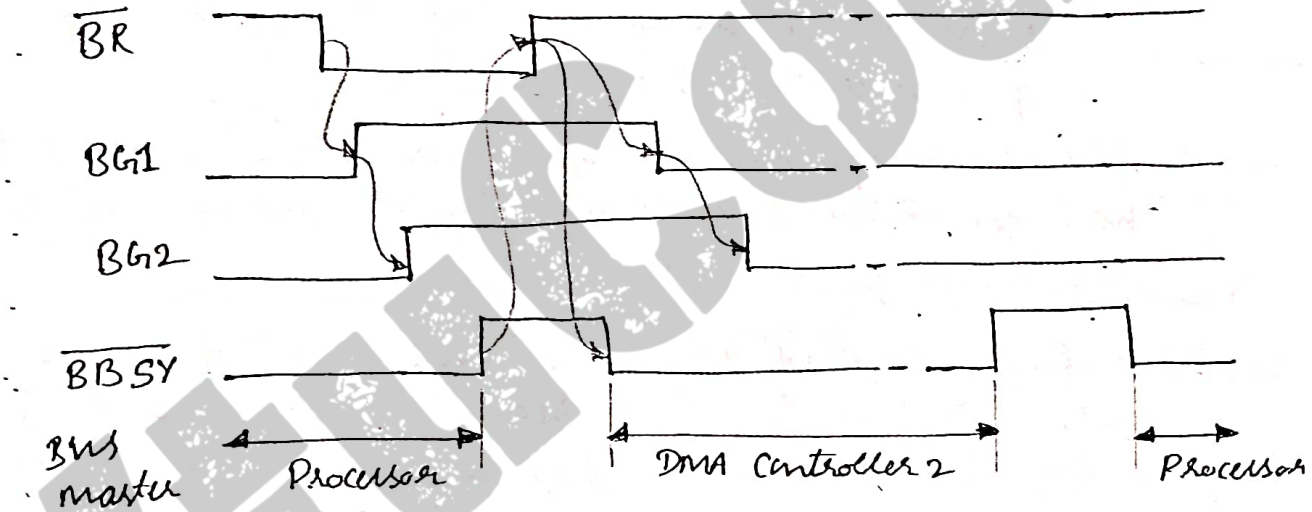


Fig 2.11: Sequence of signals during transfer of bus mastership for the devices in Fig 2.10

→ The processor is normally the bus master unless it grants bus mastership to one of the DMA controllers.

→ $\overline{BR}$: Bus-Request line: DMA controller indicates that it needs to become the bus master by activating this line.
- Active low signal

LAKSHMI ..
B.E., M.Tech (CNE)
Assistant Professor
Department of CSE

BG: Bus Grant Signal: When $\overline{BR}$ signal is activated the processor activates this line BG1.

→ This signal indicates to all DMA controllers that they may use the bus when it becomes free.

→ Connected to all DMA controllers using daisy-chain arrangement.

→ If DMA controller 1 is requesting the bus, it blocks the propagation of BG1 signal to other devices.

→ otherwise, it passes the grant downstream by asserting BG2.

$\overline{BBSY}$: The current bus master indicates to all devices that it is using the bus by activating another open-collector line Bus-Busy.

→ After receiving the Bus-Grant signal, a DMA controller waits for $\overline{BBSY}$ to become inactive, then gets the masterships of the bus.

→ In Fig 2.10 and 2.11, it is assumed that DMA Controller 2 requests and acquires bus mastership.

→ During its tenure as bus master, it may perform one or more data transfer operations.

→ After it releases the bus, the processor resumes bus mastership.

DISTRIBUTED ARBITRATION

→ All devices waiting to use the bus have equal responsibility in carrying out the arbitration process, without using a central arbiter.

→ Each device on bus is assigned a 4-bit ID number.

→ Devices requesting for the bus send the Start-arbitration signal and place their 4-bit ID on four open-collector lines $\overline{ARB0}$ - $\overline{ARB3}$.

→ A winner is selected as a result of the interaction among the signals transmitted over those lines by all the contenders

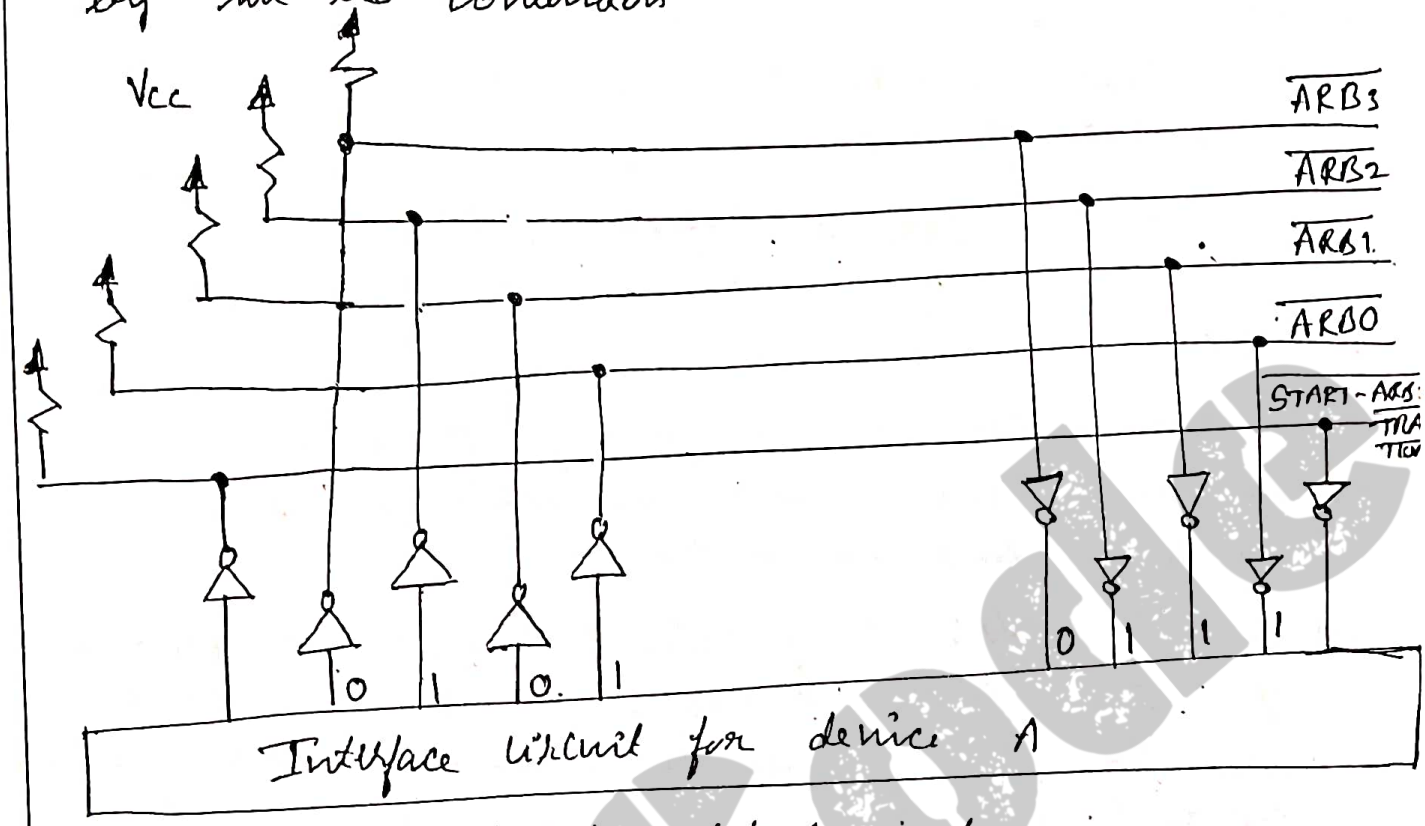


Fig 2.12: A distributed arbitration scheme.

→ The final code on the four lines is the request that has the highest ID number.

→ Assume that two devices A and B with ID numbers 5 and 6 are requesting the use of bus.

→ A → 5 → 0101
B → 6 → 0110
logical OR → 0111
} It performs logical OR operation. ∴ the code seen by devices A & B is 0111

→ Device A & B Compares its own ID with 0111 starting from most significant bit.

At A's end
0 1 1 1
0 1 0 1
0 1 0 0
↑ it detects bit change & hence disables all drivers hereafter.

At B's end
0 1 1 1
0 1 1 0
0 1 1 0
↑ detects bit change

If a device detects a difference at any bit pos^n, it disables its drivers at that bit pos^n and for all lower-order bits.
- This is done by placing 0
- For A, difference on line $\overline{ARB1}$.

-> [Note: Device B will enable ARB1 again once it sees a '0' on line ARB1 resulting from the action by 'A']

-> When A disables ARB1 and ARB0, the code seen on lines will be 0110, which means that B has won the contention.

-> Higher reliability arbitration approach.

BUSES

-> Provides a communication path for the transfer of data.

-> The bus includes the lines needed to support interrupts and arbitration.

-> Bus protocols: Set of rules that govern the behaviour of various devices connected to the bus as to when to place information on the bus, assert control signals, and so on.

-> Three types of bus lines: data, address, and control lines.

-> Control signals: specify whether read or write operation.

- line - R/$\overline{W}$: 0 -> write operation, 1 -> read operation } operands can be of size byte or word or longword.

- Timing information:
  They specify the times at which the processor and the I/O devices may place data on the bus or receive data from the bus.

- Different schemes for timing of data transfers
  - Synchronous
  - Asynchronous

- Master/Initiator: Device that initiates data transfers by issuing read or write commands on the bus.

- Slave/Target: The device addressed by the master.

SYNCHRONOUS BUS

→ All devices derive timing information from a common clock line.

→ Each of the time intervals constitutes a bus cycle during which one data transfer can take place.

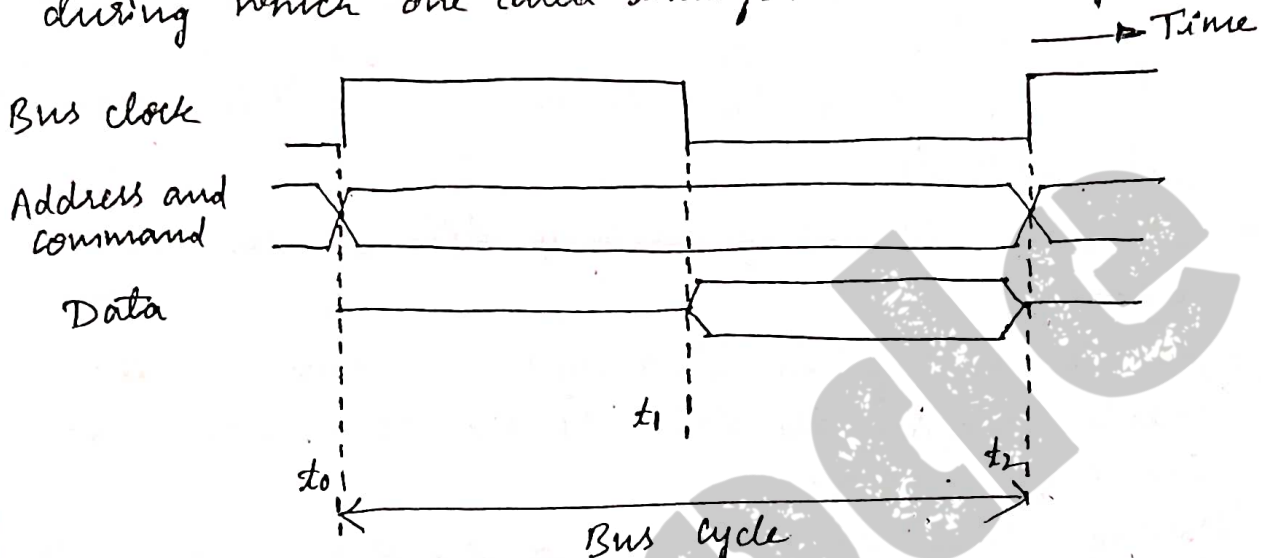
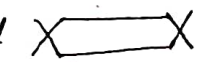
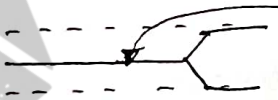


Fig 2.13 : Timing of an input transfer on a synchronous bus

→ Synchronous scheme is illustrated in Fig 2.13

→ convention used for address, data, & command  This indicates that some lines are high and some low, depending on the particular address or data pattern being transmitted.

→ The crossing points indicate the time at which these patterns change.

→  This signal line represents indeterminate or high impedance state (Neither high nor low)

Sequence of events during an input/read operation

→ At time t0, the master places the device address on the address lines and sends control command on control lines.

→ Command will indicate an input operation and specify the length of the operand to be read.

→ Information travels over the bus at a speed determined by its physical and electrical characteristics.

→ The clock pulse width $t_1 - t_0$ must be long enough to allow all devices to decode the address and control signals so that the addressed device can respond at time t_1.

→ Slave should not place any data on the bus before t_1.

→ At the end of the clock cycle, at time t_2, the master strobes the data on the data lines into its input buffer.

[Strobing: Capturing the values of the data and storing them into a buffer]

→ The period $t_2 - t_1$ must be greater than the maximum propagation time on the bus plus the setup time of the input buffer registers of the master.

→ The timing diagram in Fig 2.13 is an idealized representation.

→ Fig 2.14 shows the exact times at which signals actually change state are different from those shown because of propagation delays on bus wires and in the circuits of the devices.

→ Fig 2.14 shows two views of each signal

→ A given signal transition is seen by different devices at different times as signals take time to travel from one device to another.

→ one view shows the signal as seen by the master and the other as seen by the slave.

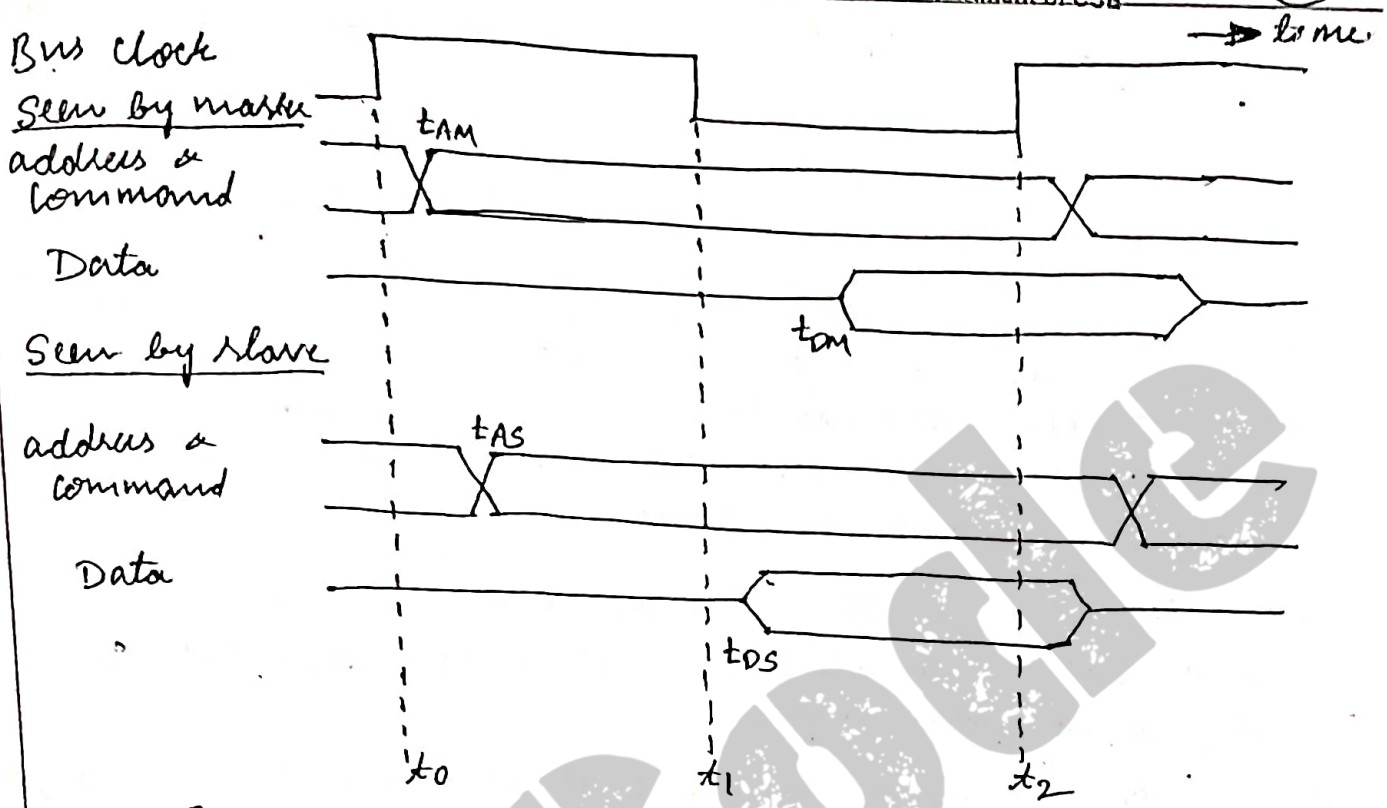


Fig 2.14: Detailed timing diagram for the input transfer of Fig 2.13

(1) Master sends the address and command signals on the rising edge at the beginning of clock period 1 → t_0

(2) Due to delay in bus driver circuit, these signals appear on the bus at t_{AM}.

(3) A while later, at t_{AS}, the signals reach slave.

(4) Slave decodes the address at t_1 and sends the requested data. The data appears on bus at t_{DS}.

(5) Data travel toward master and arrive at t_{DM}. At t_{DM}, master sets up its input buffer

(6) At t_2, master loads the data into its input buffer. $(t_2 - t_{DM})$ = set up time.

(7) The data must continue to be valid after t_2 for a period equal to the hold time of the buffer.

B.E., M.Tech (CNE)
Assistant Professor
Department of CSE

Multiple-Cycle Transfers: Limitations of previous design

→ The clock period $t_2 - t_0$, must be chosen to accommodate the longest delays on the bus and the slowest device interface.

→ This forces all devices to operate at the speed of the slowest device.

→ Processor has no way of determining whether addressed device has actually responded.

→ It simply assumes that, at t_2, the output data is received by I/O or input data is available on bus. Error will not be detected.

How to overcome these limitations?

→ Most buses incorporate control signals that represents a response from device.

→ These signal inform master that the slave has recognized its address and that it is ready to participate in data-transfer operation.

→ High-frequency clock signal is used such that a complete data transfer cycle would span several clock cycles.

→ The no. of clock cycles involved can vary from one device to another. (Slow devices - more cycles; fast devices - less cycles)

→ This approach is shown in Fig 2.15.

→ During cycle 1, the master sends address and command information on the bus, requesting read operation (eg).
- The slave receives this information and decodes it.

→ At clock cycle 2 beginning, it makes a decision to respond and begins to access the requested data (eg: DMA controller may initiate access from memory)

→ The data become ready and are placed on the bus in clock cycle 3
- At the same time, the slave asserts a control signal called Slave-ready.

→ At the end of cycle 3, master strobes the data into its input buffer.

→ The bus transfer operation is now complete and master may send a new address to start a new transfer in clock cycle 4.

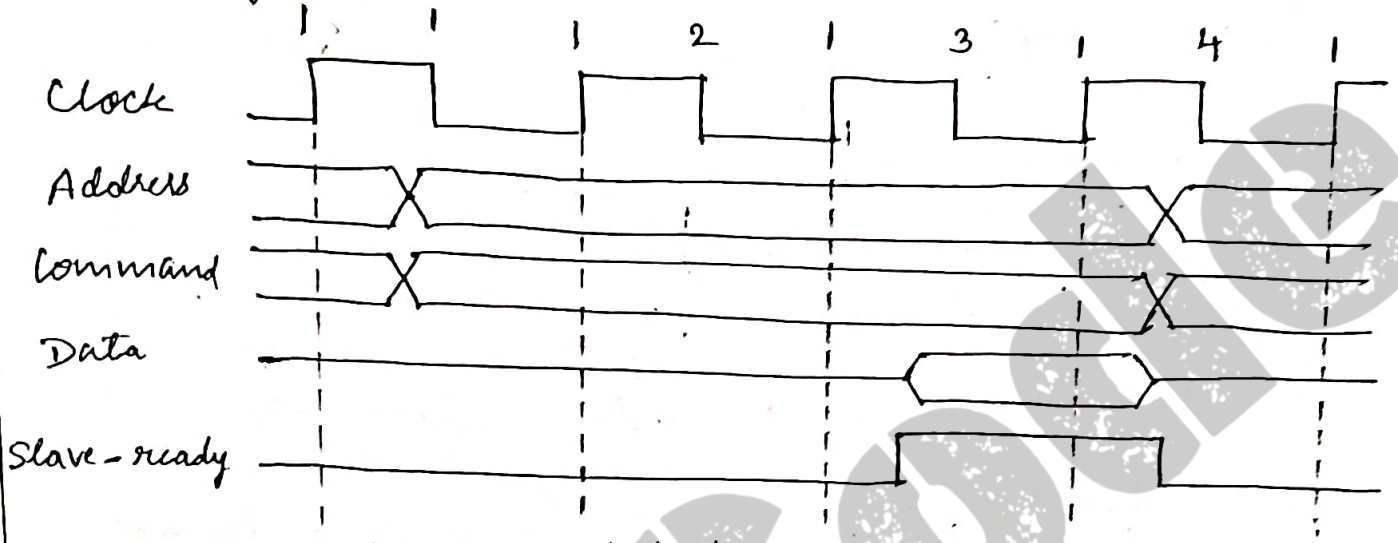


Fig 2.15: An input transfer using multiple clock cycles.

→ The Slave-ready signal is an acknowledgement from the slave to the master, confirming that valid data have been sent.

→ The slave responds in cycle 3.

ASYNCHRONOUS BUS

→ Based on the use of a handshake between master and the slave.

→ The common-clock is replaced by two timing control lines.
  → master-ready
  → slave-ready

→ The master places the address and command information on the bus.

→ Then it indicates to all devices that it has done so by activating the master-ready line.

→ The selected slave performs the required operation and informs the processor it has done so by activating slave-ready line.

LAKSHMI N
B.E., M.Tech (CNE)
Assistant Professor
Department of CSE

→ Master removes its signals from bus after it receives Slave-ready signal.

→ In case of read operation, it also strobes the data into its input buffer.

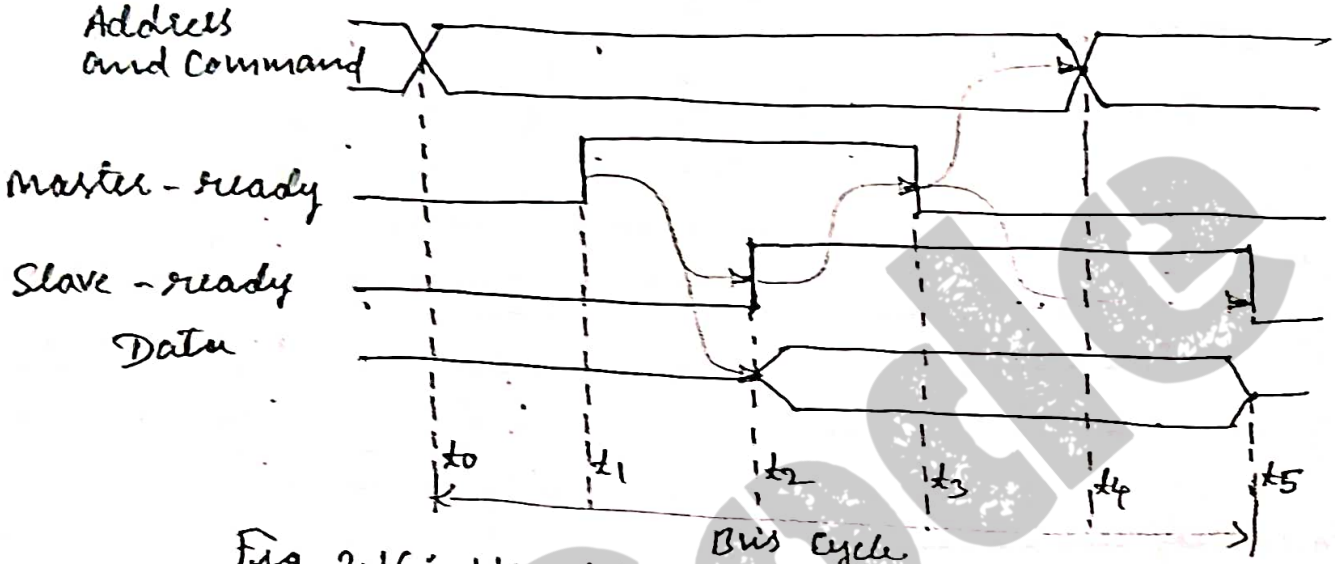


Fig 2.16: Handshake control of data transfer during an input operation.

Sequence of events

t_0 — The master places address and command on the bus. All devices on the bus begin to decode this info.

t_1 — Master sets master-ready to 1.

- Delay $(t_1 - t_0)$ is called bus skew.
- Bus skew: It occurs when two signals simultaneously transmitted from one source arrive at the destination at different times. (Because lines of the bus may have different propagation speeds)
- The delay $(t_1 - t_0)$ should be larger than the maximum possible bus skew so that the master-ready signal does not arrive at any device ahead of the address and command information.
- Sufficient delay should be included in period $(t_1 - t_0)$ to allow for the address circuitary to decode the address.

2: The selected device performs the required input operation.
- It places data from its data register on the data lines.
- At the same time, it sets the Slave-ready signal to 1.

3: The Slave-ready arrives at the master indicating that the input data is available on bus.
- The delay $(t_3 - t_2)$ allows the bus skew for master.
- It must also for allow for the set up time needed by its input buffer.
- Master strobes the data into its input buffer.

t_4: Master removes the address and command information from the bus.
- The delay $(t_4 - t_3)$ is to allow for bus skew.

t_5: Slave device removes the data and the Slave-ready signal from the bus. This completes the input transfer.

Output operation

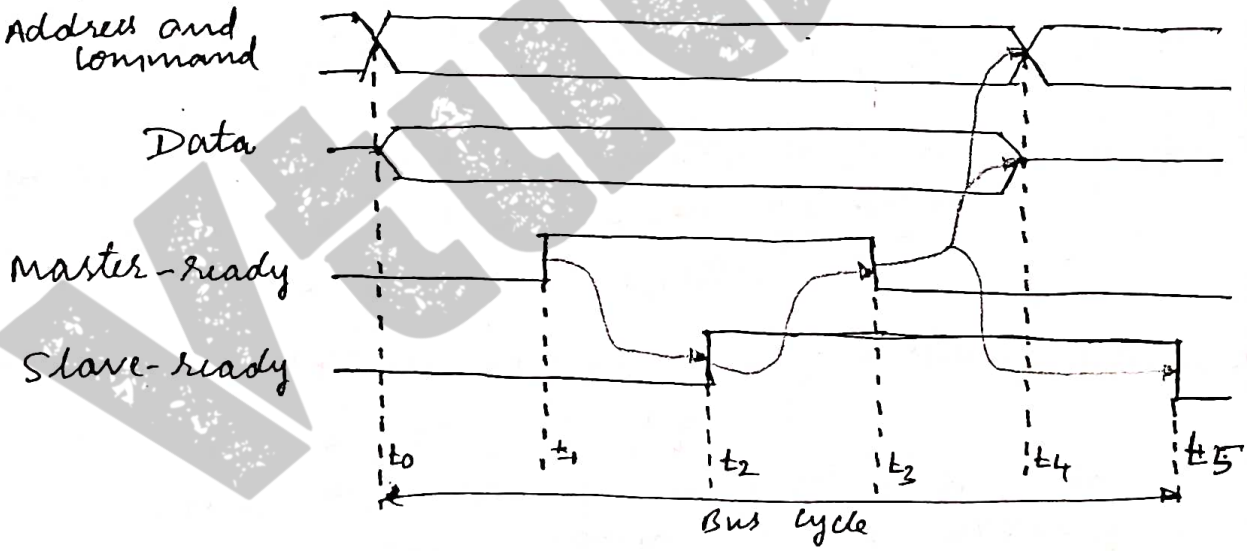


Fig 2.17: Handshake control of data transfer during an output operation.

-> Master places the output data on the data lines at the same time that it transmits address and command

LAKSHMI ...
B.E., M.Tech (CNE)
Assistant Professor
Department of CSE

→ The selected slave strobes the data into its output buffer when it receives the master-ready signal.

→ Slave indicates it has done so by ~~sending~~ setting the slave-ready signal to 1.

Full Handshake:

- The handshake signals dipicted in Fig 2.16 and Fig 2.17 are fully interlocked.
- A change of state in one signal is followed by a change in the other signal.

— To be continued...

LAKSHMI R
B.E., M.Tech (CNE)
Assistant Professor
Department of CSE

- Shorter seek and access times

- Differences

Hard Disk Drives	Flash Drives
→ Longer seek and access times	→ Shorter seek and access times
→ Higher power consumption	→ Lower power consumption.
→ Larger capacity and low cost per bit.	→ Smaller capacity and higher cost per bit.

SPEED, SIZE, AND COST

→ SRAM chips: Expensive (6 MOSFETs per bit)
: Fastest

→ DRAM Chips: Simpler cells, less expensive
: Significantly slower.

→ A huge amount of cost-effective storage can be provided by magnetic disks.

→ A large, yet affordable, main memory can be built with dynamic RAM technology.

→ SRAMs will be used in smaller units where speed is the criteria, such as in cache memories

→ Fastest access - processor registers.

→ Processor cache: Small amount of memory implemented directly on the processor chip.
- It holds the copies of instructions and data stored in a much larger memory (externally provided)
- This cache is referred to as primary cache.

→ Level 1 (L1) cache is primary cache

→ Level 2 (L2) cache is larger secondary cache placed between the primary cache and the rest of the memory. Implemented using SRAM chip

LAKSHMI R
B.E., M.Tech (CNE)
Assistant Professor
Department of CSE

⇒ Next in the hierarchy - main memory
⇒ Largest memory implemented using DRAM components.

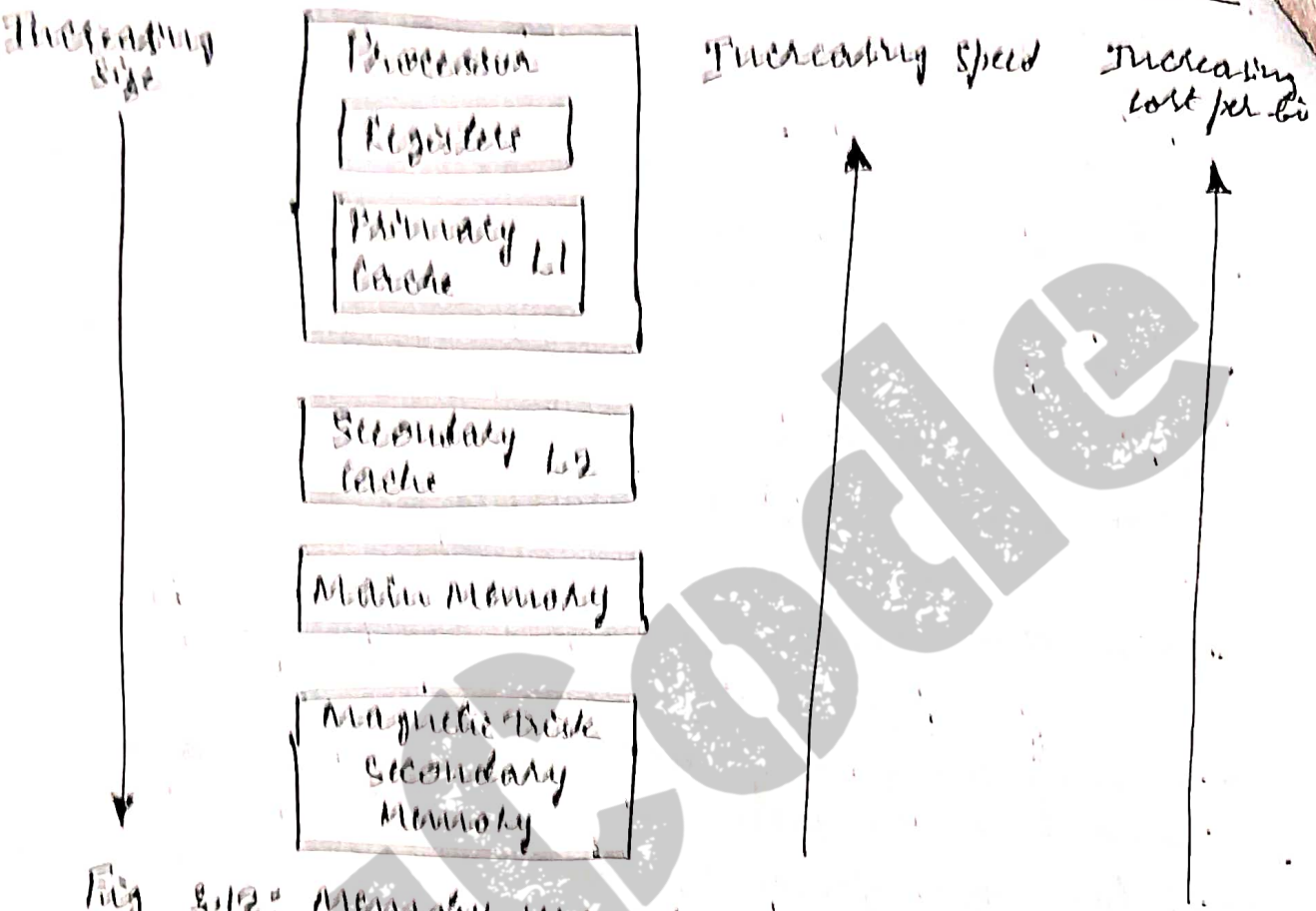


Fig 8.12: Memory Hierarchy

CACHE MEMORIES

⇒ Speed of main memory is very low when compared with the speed of processors.
⇒ For good performance, the processor cannot spend much of its time waiting to access instruction and data in memory.
⇒ So it is the use a cache memory which is faster and makes the main memory appear to the processor to be faster.
⇒ Effectiveness is based on locality of reference.
⇒ Many instructions in localized areas of the program are executed repeatedly during some time period, and

LAKSHMI R
B.E., M.Tech (CNE)
Assistant Professor
Department of CSE

the remainder of the program is accessed relatively infrequently. This is referred to as locality of reference
-> It manifests itself in 2 ways: temporal and spatial
-> Temporal: Recently executed instructions are likely to be executed again very soon.
-> Spatial: Instructions in close proximity to a recently executed instruction (w.r.t. instr addresses) are likely to be executed soon.
-> Temporal: Instr or data is brought to cache where it will hopefully retain until it is needed again.
-> Spatial: Instead of fetching just one item from main memory to cache, it is useful to fetch several items that reside at adjacent addresses as well.
-> Cache block/cache line: Refer to a set of contiguous address locations of some size.

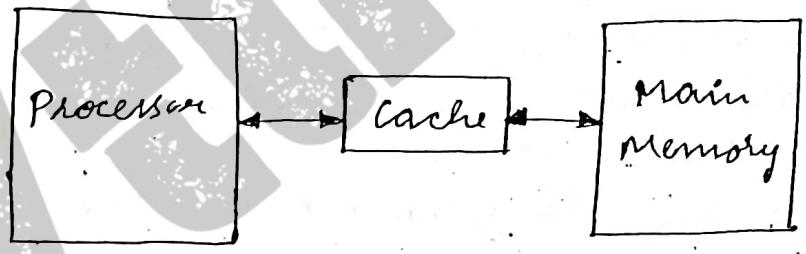


Fig 3.13: use of a cache memory

-> Cache memory can store a reasonable number of blocks at any given time. This number is small compared to the total number of blocks in main memory.
-> Working:
-> When a read request is received from the processor, the contents of a block of memory words from the specified location is transferred into

LAKSHMI R
B.E., M.Tech (CNE)
Assistant Professor
Department of CSE

the cache one word at a time.
-> When the program references any of the locations in this block, these contents are read directly from the cache.

-> Mapping Function: The correspondence between the main memory blocks and those in cache is specified by a mapping function.
-> When the cache is full and a memory word that is not in cache is referenced, the cache hardware must decide which block should be removed to create space for the new block.
-> The collection of rules for making this decision constitutes the replacing algorithm.

Read or Write hit:
-> Processor issues Read or write request along with the address of the location.
-> The cache control circuitry determines whether requested word currently in cache.
-> If it exists, the Read or write operation is performed on the appropriate cache location.
-> In this case, a 'read or write hit' is said to have occurred. Considering hit case,

Read operation: The main memory is not involved

Write operation: System can proceed in two ways:
① Write through protocol: The cache location and and the main memory location are updated simultaneously.
② Write back or copy back protocol
-> Update only the cache location and mark it as updated with an associated flag bit, called the dirty or modified bit.

LAKSHMI R
B.E., M.Tech (CNE)
Assistant Professor
Department of CSE

-> The main memory location of the word is updated later.
But, when?

-> When the block containing this marked word is to be removed from the cache to make room for new block (using some replacement algm)

Read & Write miss

Read miss:

-> When the addressed word in read operation is not in cache, a read miss occurs.

Handled in 2 ways

(1) The block of requested word is copied from the main memory into the cache.
- After the entire block is loaded to cache, the word is forwarded to the processor.

(2) The requested word may be sent to the processor as soon it is read from the main memory.
- This approach is called load-through or early restart.

Write miss:

-> When the addressed word in write operation is not in cache, a write miss occurs.

Handled in 2 ways

(1) In write through protocol, the information is directly written into the main memory.

(2) In write back protocol, the desired word in the cache is overwritten with the new information.

LAKSHMI R
B.E., M.Tech (CNE)
Assistant Professor
Department of CSE

Mapping Functions:

→ Possible methods for specifying where memory blocks are placed in the cache.

Eg:- Cache: 128 blocks of 16 words each
- total of 2048 (2k) words.

Memory: 64 k words - viewed as 4k blocks of 16 words
- Addressable by a 16-bit address.

Different mapping functions
- Direct Mapping
- Associative Mapping
- Set-Associative Mapping

Direct Mapping

→ Simplest way
→ Block j of the main memory maps onto block j modulo 128 of the cache.
→ Main memory blocks 0, 128, 256, ... is loaded in cache block 0.
→ Main memory blocks 1, 129, 257, ... is loaded in cache block 1 and so on.
→ Since more than one memory block is mapped onto a given cache block position, contention (dispute) may arise for that position even when the cache is not full.
→ Contention is resolved by allowing the new block to overwrite the currently resident block.
→ Memory address is divided into three fields:
→ Word: Low-order 4 bits select one of 16 words in a block
Block: 7-bit cache block field determines the cache position where block must be placed.
Tag: Higher 5-bits of the address

- Tag identifies which of the 32-bit blocks that are mapped into this cache.
- As execution proceeds, the high-order 5 bits of the address are compared with the tag bits associated with that cache location.

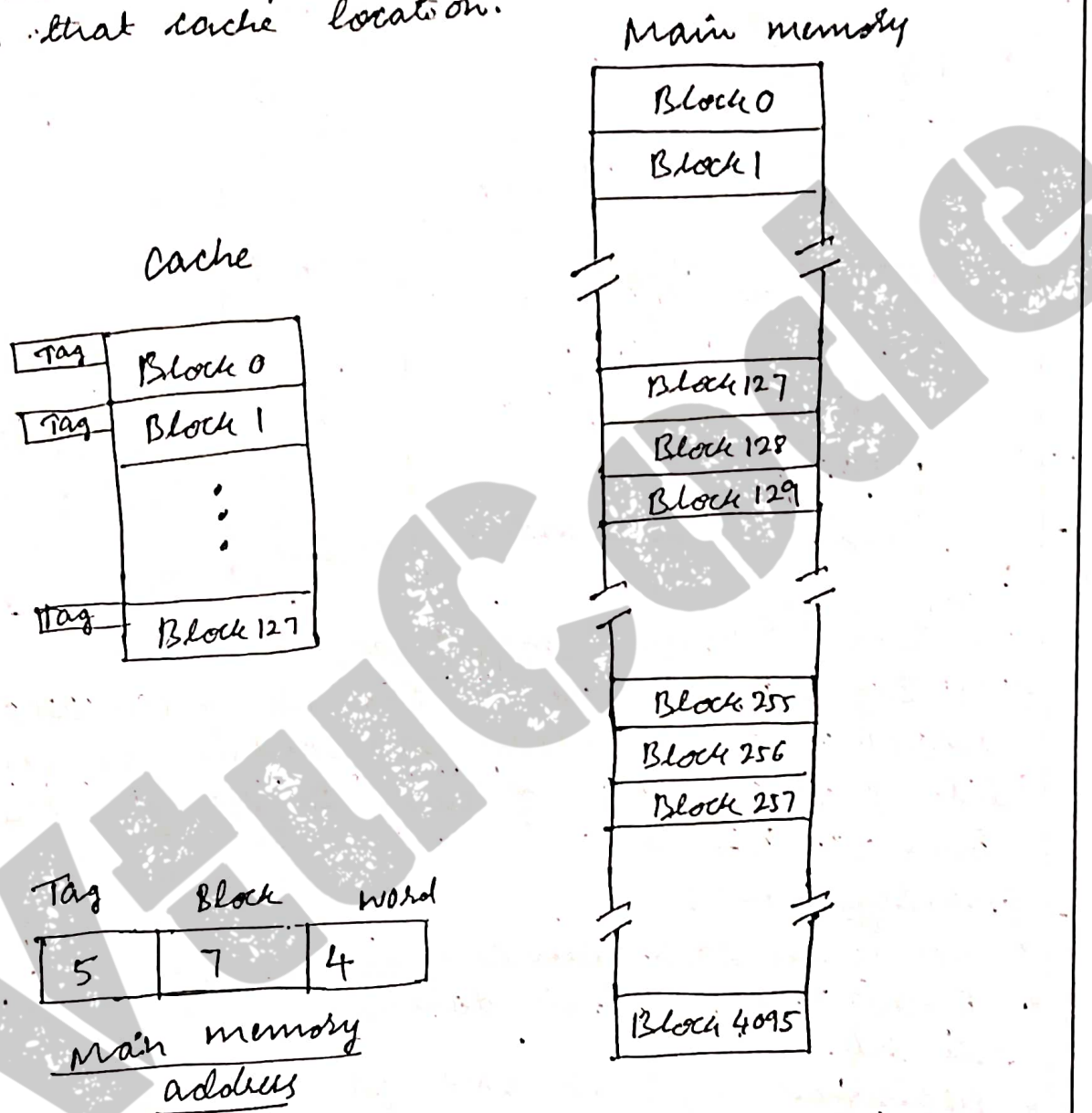


Fig 3.14 : Direct-mapped cache

Block 0 - 127 → 1st 32 blocks
Block 128 - 255 → 2nd 32 blocks
Block 256 - 383 → 3rd 32 blocks
Block 384 - 511 → 4th 32 blocks
...
Block 3968 - 4096 → 32nd 32 blocks

} 5-bit tag represents which 32 block

LAKSHMI R
B.E., M.Tech (CNE)
Assistant Professor
Department of CSE

Associative mapping:

→ Much more flexible.

→ Main memory block can be placed into any cache block position.

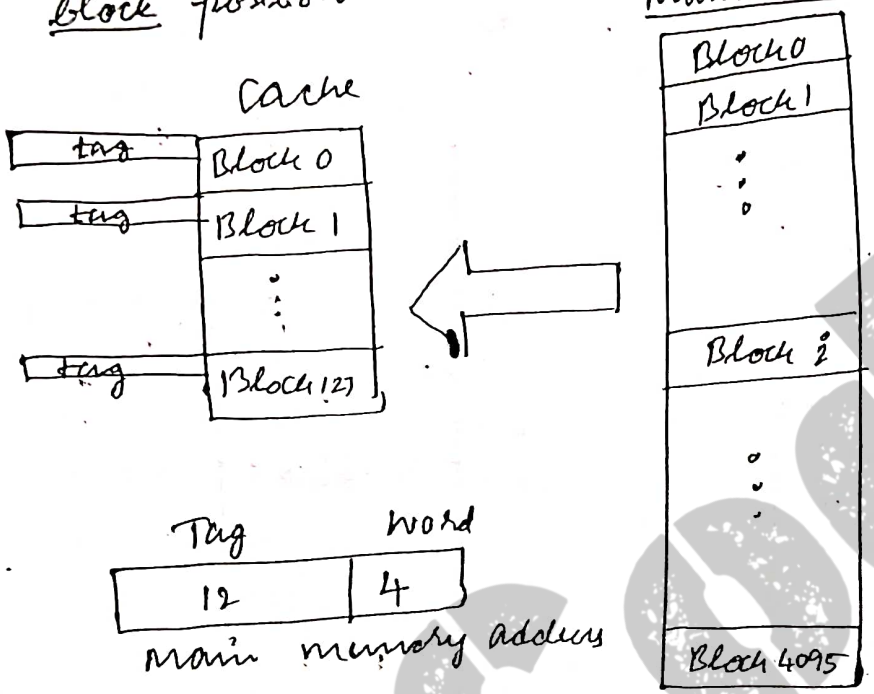


Fig 3.15: Associative-mapped cache

→ 12-tag bits are used. The tag bits of an address received from the processor are compared to the tag bits of each block of cache to see if desired block is present.

Advantages

- Space in the cache can be efficiently used.
- Block replacement is done only when the cache is full.
- Replacement algm is used for this purpose.

Disadvantages

- Cost of associative mapping is higher than direct map
- This is because, during search process, all 128 tag patterns have to be searched to see whether a given block is present in the cache or not (in the worst case)
- A search of this kind is called associative search.

LAKSHMI R
B.E., M Tech (CNE)
Assistant Professor
Department of CSE

Set-Associative Mapping

→ Combination of direct and associative mapping.
→ Blocks of the cache are grouped into sets.
→ Mapping allows a block of the main memory to reside in any block of a specific set.
→ Contention problem is resolved by having a few choices for block placement.
→ At the same time, the hardware cost is reduced by decreasing the size of associative search.

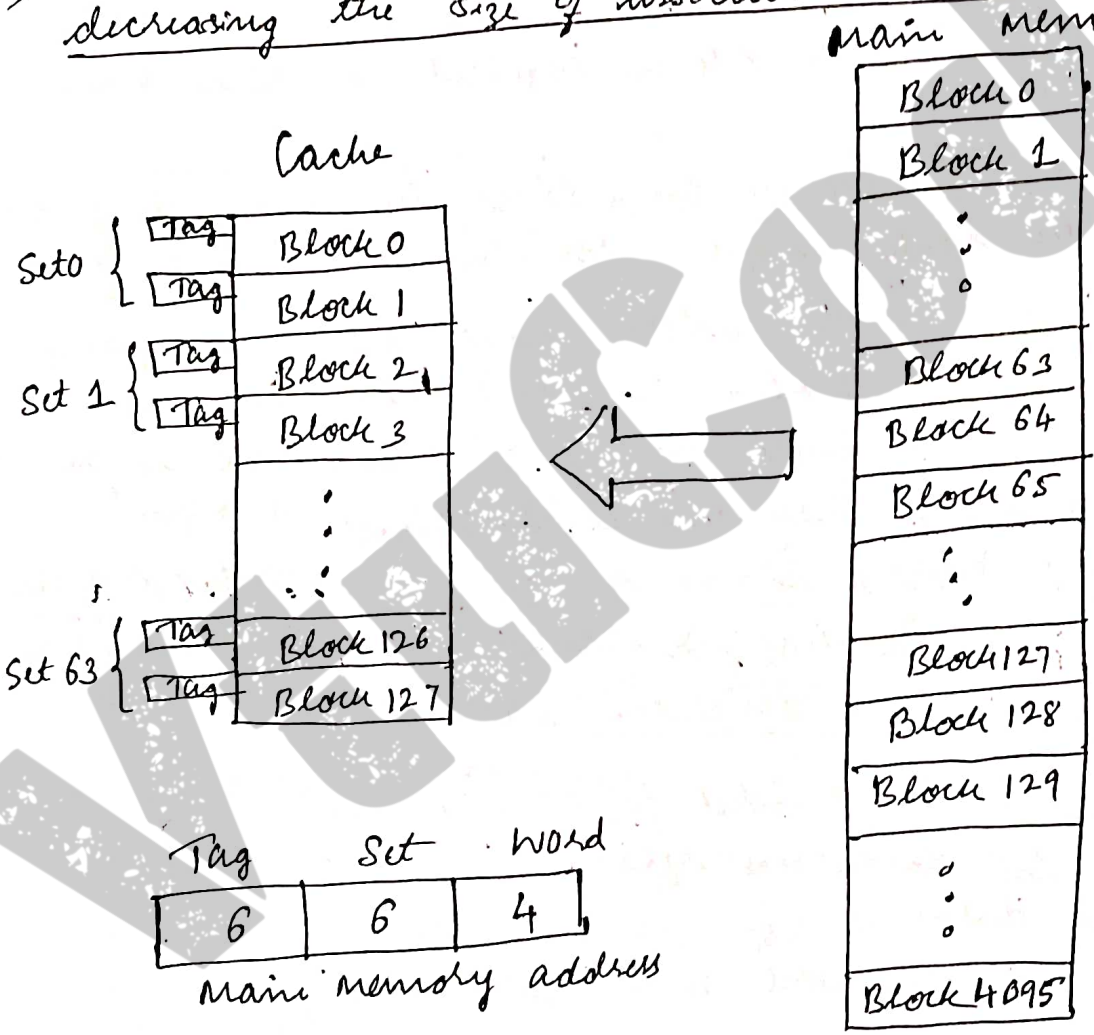


Fig 3.16: Set-associative-mapped cache with two blocks per set.

LAKSHMI R
B.E., M.Tech (CNE)
Assistant Professor
Department of CSE

→ Fig 3.16 shows cache with 2 blocks per set
→ Memory blocks 0, 64, 128, ..., 4032 map into cache set 0
- 1, 65, 129, ..., 4033 into cache set 1
- 2, 66, 130, ..., 4034 into cache set 2
- ...
- 63, 127, 191, ..., 4095 into cache set 63

} This is because there are 64 cache sets

→ 64 cache set → 6-bits are required to select a particular set
→ ∴ 6-bit set field of the address determines which set of the cache might contain the desired block.
→ Tag field of the address must then be compared (associatively) to the tags ~~field~~ of the two blocks of the set to check if the desired block is present. (reduced to only two comparisons)
→ This is referred to as two-way associative search.
→ A cache that has k blocks per set is referred to as a k-way set-associative cache.
→ Valid bit is provided for each block.
→ This bit indicates whether the block contains valid data.
→ This is associated with: DMA transfer

Working

→ The valid bits are all set to 0 when power is applied initially
or
when the new programs and data from disk is loaded to main memory.

-> Transfers from disk to the main memory are carried out by a DMA mechanism.

-> Normally, they bypass the cache.

-> The valid bit of a particular cache block is set to 1 the first time this block is loaded from the main memory.

-> Whenever a main memory block is updated by a source that bypass the cache, a check is made to determine whether the block being loaded is currently in the cache.

-> If it is, its valid bit is cleared to 0.

-> This ensures that stale data will not exist in the cache.

-> Similar difficulty arises when a DMA transfer is made from main memory to the disk, and the cache uses the write-back protocol.

-> Solution is to flush the cache by forcing the dirty data to be written back to the memory before DMA transfer takes place.

Cache-coherence problem

-> Ensuring that two different entities (the processor and DMA subsystems for eg) use the same copies of data is referred to as cache coherence problem.

Example:

A block set associative cache consists of a total of 64 blocks divided into 4 blocks sets. The main memory contains 4096 blocks, each consisting of 128 words.

a) How many bits are there in the main memory address?

b) How many bits are there in each of the TAG, SET and WORD field?

Sol^n: a) main memory = 4096 blocks of 128 words

$= 2^{12} \times 2^7 = 2^{19}$

$\therefore$ 19 bits are there in the main memory address

b) Cache has 64 blocks. This is divided into 4 blocks set.

$\frac{64}{4} = 16$ $\therefore$ there are totally 16 sets in cache.

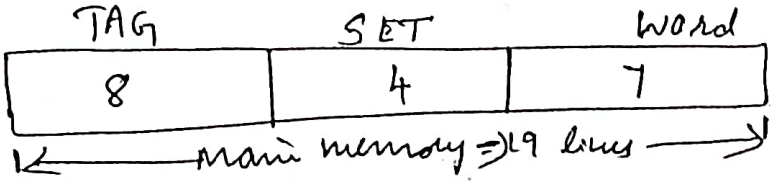
$\Rightarrow$ This requires **4 bits in SET field** ($\because 2^4 = 16$)

There are 128 words in each block.

$2^7 = 128$ **$\therefore$ WORD field has 7 bits**

From (a), we got to know that there are 19 address lines (bits). out of 19, 4 bits for SET and 7 for WORD.

$\therefore$ Remaining **8 bits represent TAG field.**



only for reference

Direct Mapping