papergri

Date: / /

MODULE - V

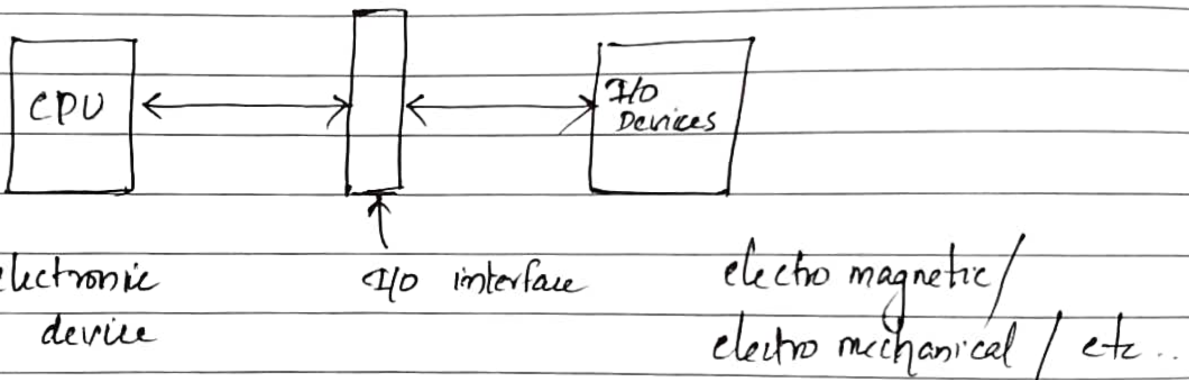
I/O Organization : Accessing of I/O devices - interrupts, interrupt hardware - Direct memory access.

Peripheral :-

Devices connected to processor externally except main m

- Input devices, o/p devices, storage devices.

Can CPU Access IO directly ?



Why these I/o interfaces.

- act as a transilator
- provides synchronization b/w High speed CPU and slow speed I/o devices.
- Data format conversion

Interface - It is a shared boundary between two seperate components of the computer system which can be used to attach two or more components of the system for communication purposes.

papergrid

Date: / /

Need for IO Interface

(1) Peripherals are electromechanical or electromagnetic devices and their manner of operation is different from the operation of the CPU and memory which are electronic devices. So conversion of signal is required.

(2) The data transfer rate of peripherals is synchronization is required.

(3) Data codes and format in peripherals differ from the word format of the CPU and memory. So conversion of formats is required.

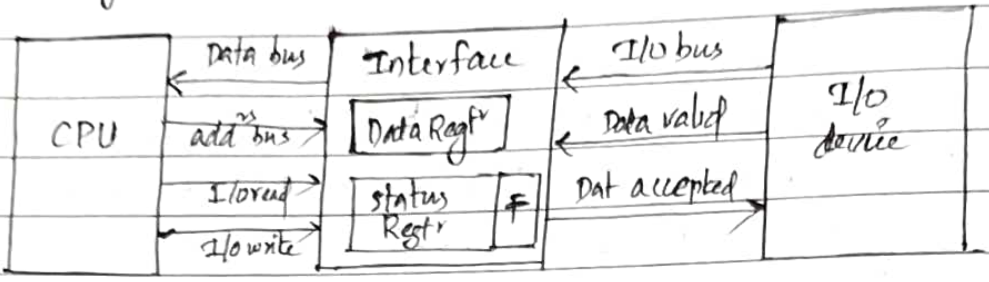
(4) The operating modes of peripherals are different from each other and each must be controlled so a peripheral does not disturb the operations of other peripherals.

Modes of I/O Data Transfer

Data transfer between the central unit and I/O devices can be handled in generally three types of modes.

(1) Programmed I/O
(2) Interrupt initiated I/O
(3) Direct Memory Access.

(1) Programmed I/O



→ When ever a key of a keyboard is pressed, the data (ASCII) value is transfferred through Data valid line and is received in interface (Data Regtr)

→ Simultaneously the I/o interface change the value of flag bit from '0' to '1' (I/O data is ready for transfer)

→

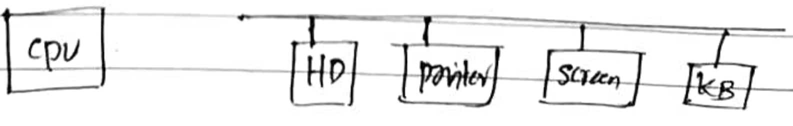
1) Read the status register
2) Check the status of the flag bit and branch to step 1 if not set or to step 3 if set
3) Read the data regtr

{ cpu continuously monitor the I/o devices }

☹ the efficiency of the ~~CP~~ system get reduced. ⇒ used in very slow concepts.

(2) Interrupt initiated I/O

Instead of continuously monitoring the I/o devices, whenever the I/o devices needs to transfer the data, an interrupt signal is generated.



papergrid

Date: / /

Whenever the CPU will stops its normal pgm execution and the ISR (interrupt service routine) related with that interrupt is executed. The ctrl will ~~contr~~ came back to the normal exec^n after the execution of ISR.

How the branching of ctrl to the ISR occurs, there are two methods.

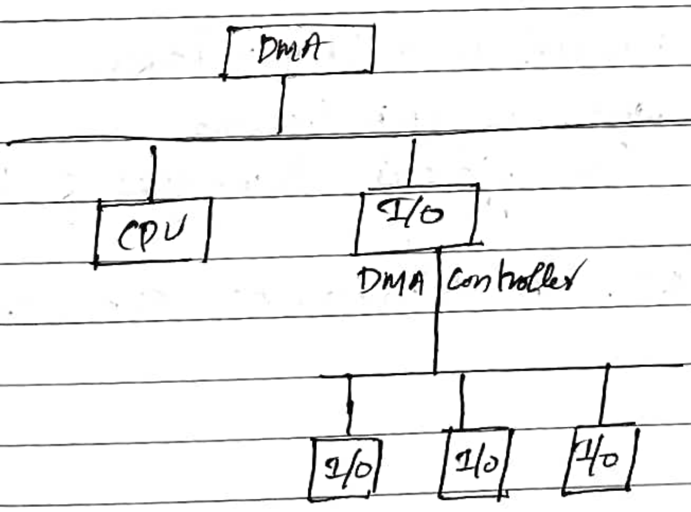
(1) Non vectored interrupt
(2) vectored interrupt.

(1) Non vectored - On initiating interrupt signal, the CPU by default move to the location where ISR is locked.

(2) Vectored interrupt - I/O interface transfer the add^n where the ISR is located.

the I/O device, on generating the interrupt signal itself generate the add^n of the ISR

(3) Direct Memory Access (DMA)



🙂 Programmed I/O
- Simple to implement
- requires very little h/w support
- CPU check the status bits periodically

☹ - the processor has to wait for a long time for the I/O module to be ready for either transmission or reception
- the performance of the s/m is severely degraded.

🙂 · Interrupt driven I/O
- faster and more efficient than programmed I/O

☹ tough to get various pieces of work well together
h/w manufacturer / OS maker usually implements

- What is the function of ISR
- How the time involved in polling process is
- reduced in interrupted I/O ?
- What are vectored interrupts ?
- List and describe the registers in a DMA interface
- Compare the two main modes of DMA transfer
- Describe the different bus arbitration techniques for DMA data transfer.
- What is interrupt ? Discuss the difference b/w subroutine and ISR

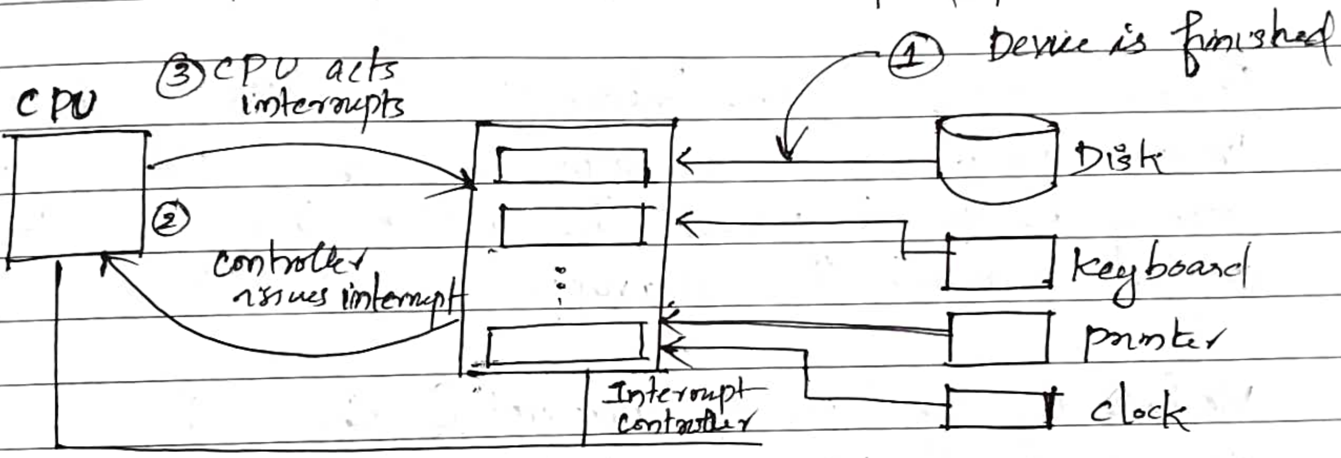
Date: / /

Interrupts

Each and every operation in a computer is guided by an interrupt.

time needed to execute an instn – instn cycle.

* CPU interrupt - request line triggered by I/O device
* Interrupt handler receives interrupts.
* – Determines the best course of action
  - Find out the source of interrupt generation
  - Analyze its status
  - Restarts it when appropriate with the next operation
  - Returns control to the interrupted process



papergrid
Date: / /

Interrupt

- Interrupt is a signal emitted by h/w or s/w when a process or an event needs immediate attention

- The process that runs when an interrupt is generated is the Interrupt Handler. (Interrupt Service Routine ISR)

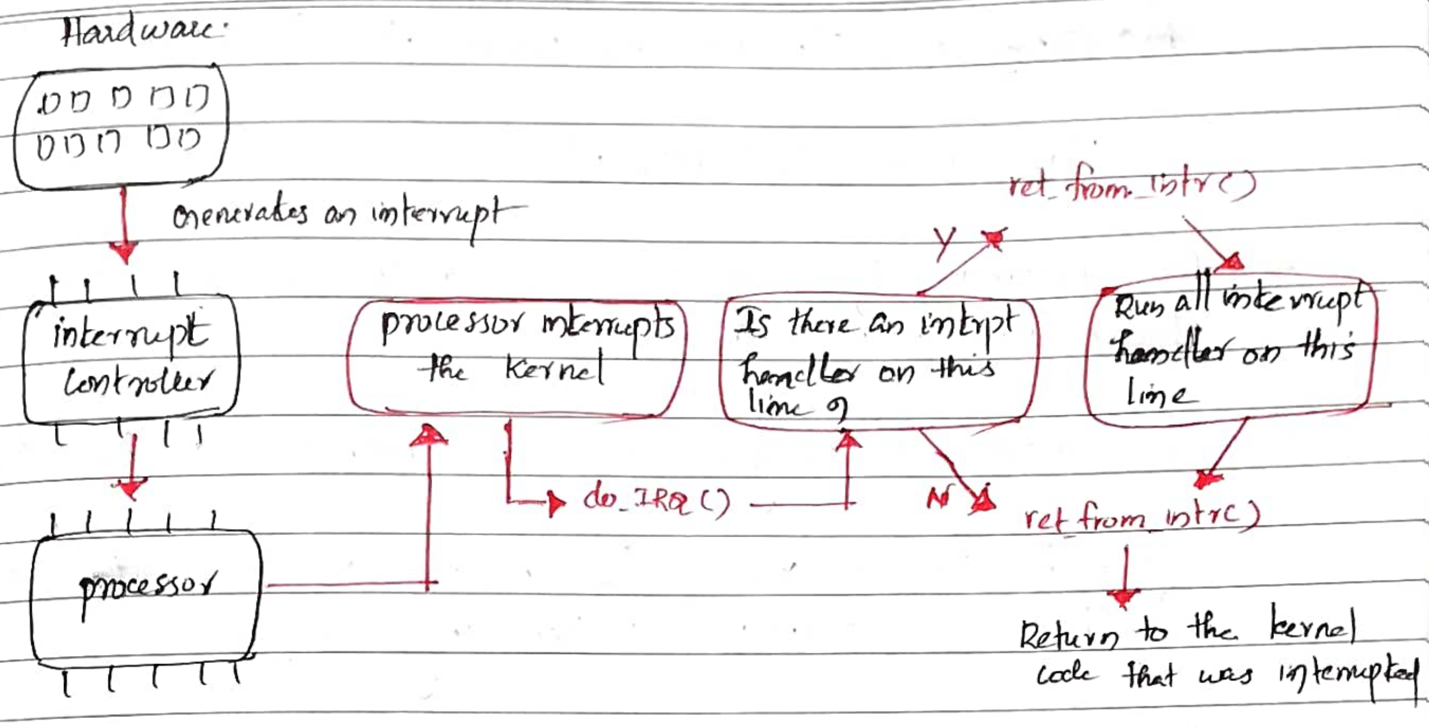
- The CPU saves the state of the ongoing process and shifts its attention to the interrupt generated by giving access to the interrupt handler.
  This entire process is called interrupt handling.

- ISR handles the request and sends it to the CPU. When the ISR is complete, the process gets resumed.

Before working on the interrupt, the state of the current program which was in execution is saved. After saving the state of the current process, the control is then given to a program to handle the interrupt.

We have several types of interrupt handlers. There is usually an interrupt handler associated with an interrupt.

For example, the keyboard has its interrupt handler; and the printer has its interrupt handler, and so on.


Hardware
Generates an interrupt
interrupt controller
processor
processor interrupts the kernel
do_IRQ()
Is there an intrpt handler on this line ?
Y
N
ret_from_intr()
Run all interrupt handler on this line
ret_from_intr()
Return to the kernel code that was interrupted

Types of Interrupt Handlers

~~(1)~~ We can broadly divide the interrupt handlers into two categories according to the interrupt handling time.

(1) First Level Interrupt Handler or FLIH

- quickly services an interrupt and schedules the second level interrupt handler if needed. FLIH mainly deals with maskable interrupts.
- It is also known as fast interrupt handler (hard interrupt handler)

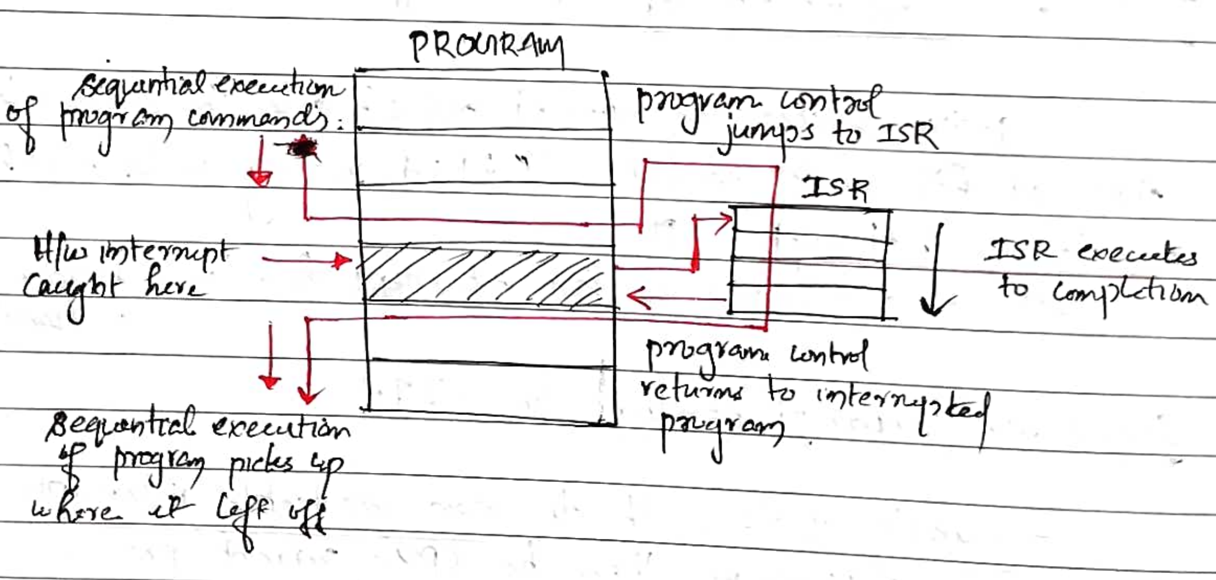
(2) Second Level Interrupt Handler or SLIH

- usually deals with the non-maskable interrupts that have higher priority than the CPU's current process.
- It is also known as second level interrupt handler or slow interrupt handler, or soft interrupt handler.

{Maskable and Non Maskable Interrupts. } Later

How to Handle Interrupt ?

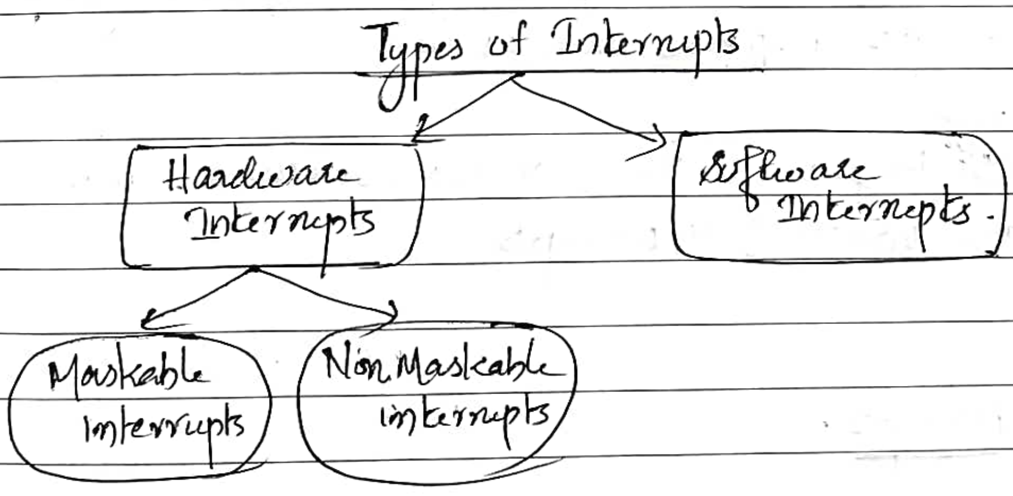
- An interrupt first ~~goto~~ goes to hardware known as Interrupt Controller.
- This controller will generate an interrupt to the CPU.
- After the completion of '1' instruction cycle, the CPU checks whether an interrupt is pending or not.
- If any interrupt is not there, then the CPU will continue with the next instn.
- But if there is some unserviced interrupt, then the CPU will pay attention to that interrupt.


PROGRAM
Sequential execution of program commands:
program control jumps to ISR
ISR
H/w interrupt caught here
ISR executes to completion
program control returns to interrupted program.
Sequential execution of program picks up where it left off

During interrupt handling, the state of the current program which was in execution is saved along with its corresponding register states.
After saving the state of the current process, the control is then given to a program to handle the interrupt. (ISR)

The CPU will now detect the kind of interrupt and its respective interrupt number. The interrupt numb' and its corresponding instⁿ set address are stored in a vector table known as IVT (Interrupt Vector Table).

Using the number and IVT, the CPU will get to know the base addⁿ of the process that is needed to handle the respective interrupt. (ISR). Now the control will be given to the ISR and after execution, the control will be coming back to the ~~[illegible]~~ suspended process and continues its execution.



Maskable interrupt - ~~The~~ An interrupt that can be disabled ~~by~~ or ignored by the instⁿˢ of CPU are called Maskable interrupts.

eg: RST 6.5, RST 7.5, RST 5.5 of 8085

papergrid
Date: / /

Non Maskable Interrupts. - An interrupt that cannot be disabled or ignored by the instn of CPU are called as Non maskable interrupts

eg: Trap of 8085

Differences.

When maskable interrupt occur, it can be handled after executing the current instn.

When non-maskable interrupt occur, the current instns and status are stored in stack for the CPU to handle the interrupt.

Maskable interrupts used to interface with peripheral devices. Non maskable interrupt used for emergency purpose

In maskable interrupts, response time is high. In non maskable interrupts response time is low.

Maskable interrupt may be vectored or non vectored. non maskable interrupts are vectored interrupts

Vectored Interrupts.

They are a type of interrupt mechanism where the interrupting device or program directly provides the processor with information about the specific interrupt source.

device sends two things to the processor,
1) Interrupt signal. 2) Vector addrs (Base addrs of ISR)

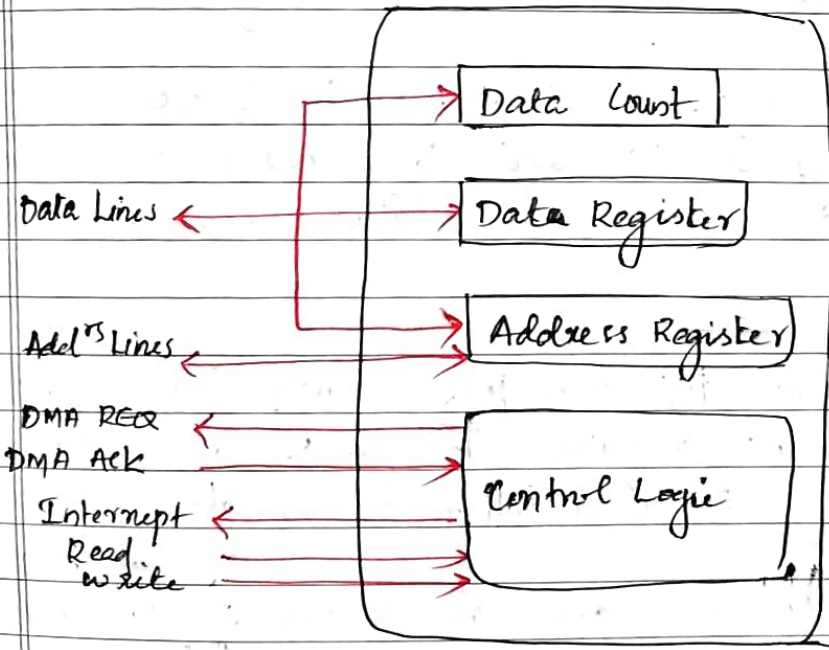
Non-vectored Interrupts.

They are also known as basic interrupts or scalar interrupts, do not provide direct information about the interrupt source. When a non-vectored interrupt occurs, the processor executes a default service routine (DSR) to determine which interrupt handler routine to execute. device only sends the interrupt signal to the CPU.

Ref: Difference b/w Interrupt service routine (ISR) and subroutine.

Direct Memory Access.

It is a process which enables data transfer between the memory and the I/O. device without the need of (without the involvement) of CPU during data transfer.



– Block diagram of DMA controller.

Working of DMA.

→ DMA need a h/w called Direct Memory Access controller (DMAC) which will help in the ~~throughput~~ throughout process of data transfer between the Memory and IO devices directly.

→ Firstly, IO devices sends the DMA request to DMA controller, then further DMAC device sends HOLD signal to CPU by which it asks CPU for several information which are needed while transferring data.

→ CPU then shares two basic info with DMAC before the data transfer which are:
(1) starting add^rs (memory add^rs starting from where data transfer should be performed).
(2) Data Count (no: of bytes or words to be transferred)

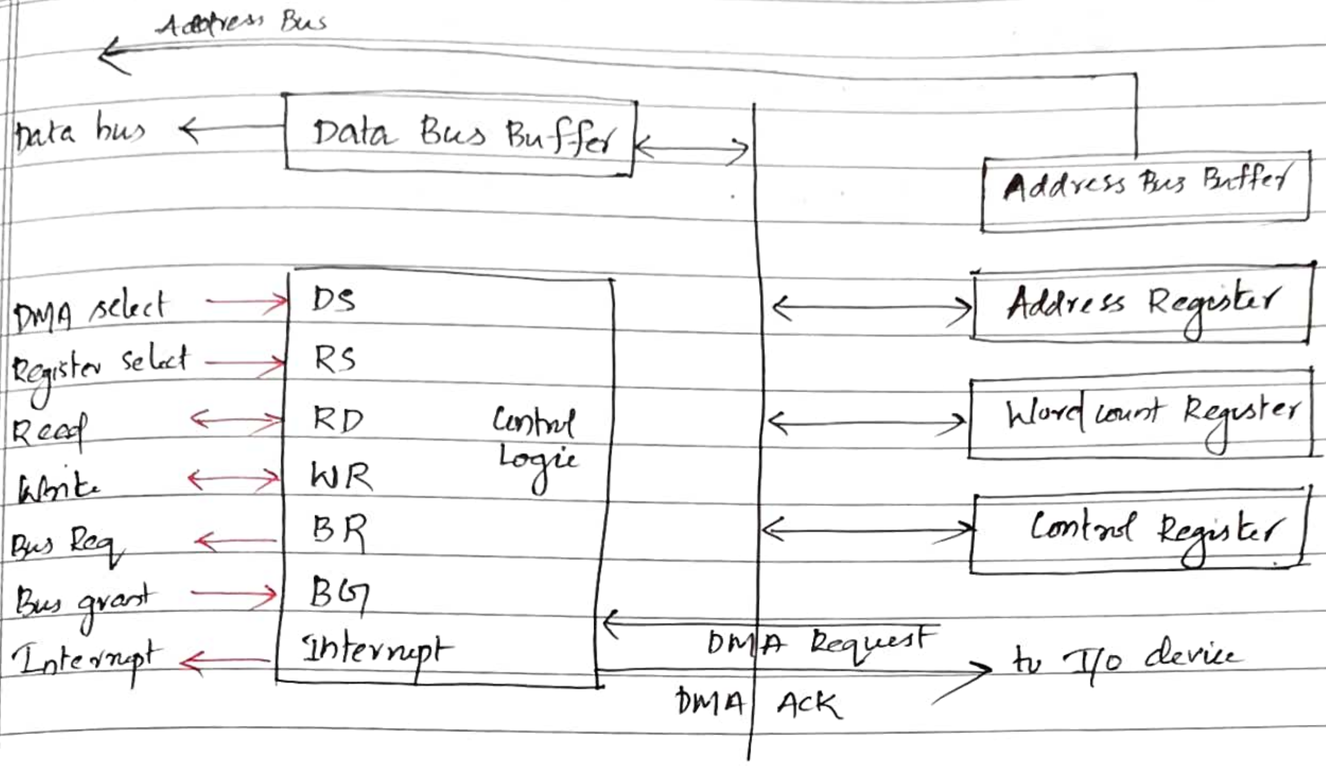
→ CPU then sends HLDACK back to DMAC illustrating that now DMAC can successfully pass on the information.

→ Then further DMAC shares the DMA Ack to the IO device which would eventually let IO device to access or transfer the data from memory in a direct and efficient manner.

During the DMA transfer CPU can perform only those operation in which it doesn't require the access of System Bus which means mostly CPU will be in blocked state.

How much time CPU will give the control ~~of~~ of sm bus to DMAC depends on the modes of DMA Transfer.

fig: WORKING DIAGRAM OF DMA CONTROLLER



Modes of DMA Transfer

(1) Burst Mode – Burst of data is transferred before CPU takes control of the buses back from DMAC.

- this is the quickest mode of DMA transfer since at once only the huge amount of data is being transferred so time will be saved in huge amount

☺ fastest mode

☹ Less user friendly (CPU will be in blocked state)

(2) Cycle Stealing Mode – Slow I/O device will take some time to prepare data and within that time CPU keeps the control of the bus.

- Once the data or word is ready CPU gives back control of system buses to DMAC for 1 cycle in which the prepared word is transferred to m/y

- Compared to burst mode, little bit slowest since it requires little bit of time which is actually consumed by IO device

🙂 Most efficient way of transfer
CPU won't be blocked entire time.

🙁 Rate of DMA transfer will be less.