Data Transfer Schemes

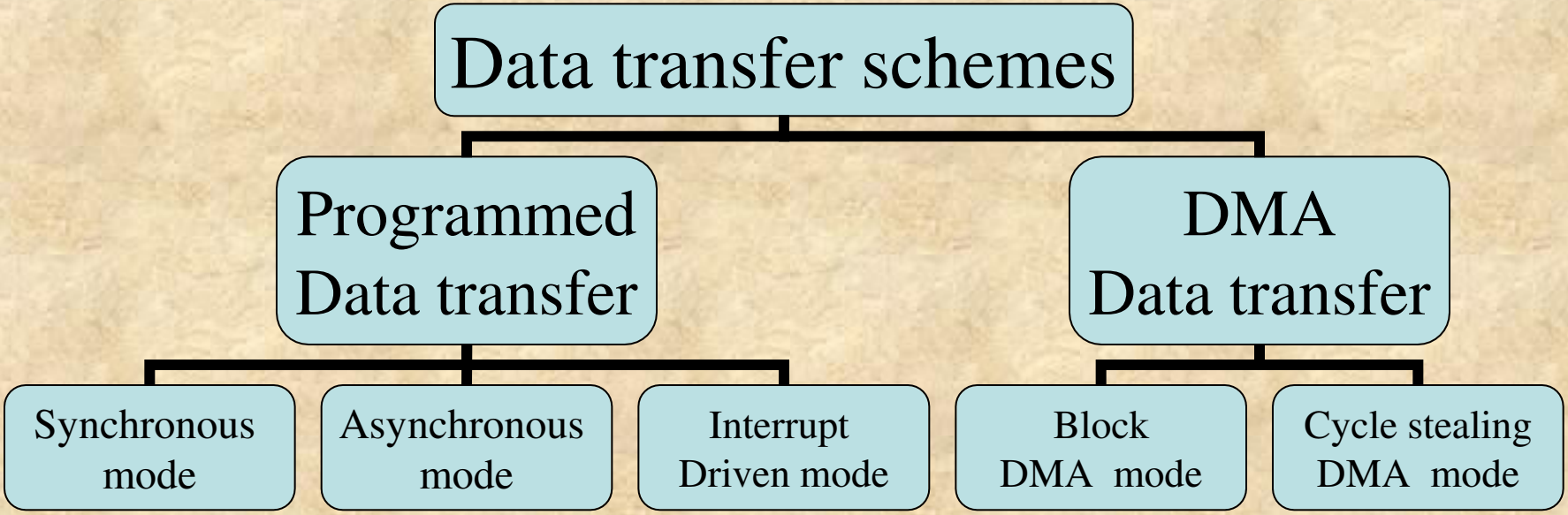
Why do we need data transfer schemes ?

- Availability of wide variety of I/O devices because of variations in manufacturing technologies e.g. electromechanical, electrical, mechanical, electronic etc.

- Enormous variation in the range of speed.

- Wide variation in the format of data.

Classification of Data Transfer Schemes



Programmed Data Transfer Scheme

- The data transfer takes place under the control of a program residing in the main memory.
- These programs are executed by the CPU when an I/O device is ready to transfer data.
- To transfer one byte of data, it needs to execute some instructions.
- This scheme is very slow and thus suitable when small amount of data is to be transferred.

Programmed data transfer

- Programmed data transfer is written and controlled by programmer and executed by the processor.

- The data transfer between processor and I/O devices or vice versa takes place by executing the corresponding instruction/program.

- Programmed I/O data transfers are identical to read and write operations for memories or devices.
- Example instructions – MOV M,A, IN 01, OUT 02 etc.

Programmed data transfer contd..

- The execution of programmed data transfer can take place at predefined period determined by the programmer.
- Based on the time of execution of the data transfer instruction, the programmed data transfer is divided into following three types namely:

a) **Synchronous mode of data transfer**

b) **Asynchronous mode of data transfer**

c) **Interrupt driven data transfer**

Synchronous Mode of Data Transfer

- Its used for I/O devices whose *timing characteristics* are fast enough to be compatible in speed with the communicating Processor.
- In this case the status of the I/O device is not checked before data transfer.
- The data transfer is executed using IN and OUT instructions.

- In simple or synchronous mode, the data is read, from the input device, by the processor irrespective of the status of the input device.

- It is assumed that the input device is ready with the data as and when the processor reads the data and the input device is in synchronism with the processor.

- Similarly, the data is written onto the output device irrespective of its status assuming that the output device is in synchronism with the processor.

- Memory compatible with Processors are available. Hence this method is invariably used with compatible memory devices.

- The I/O devices compatible in speed with Processor are usually not available. Hence this technique is **rarely** used for I/O.

Asynchronous Data Transfer

- This method of data transfer is also called ***Handshaking mode***.

- This scheme is used when speed of I/O device does not match with that of Processor and the timing characteristics are not predictable.

- The Processor first sends a request to the device and then keeps on checking its status.

- The data transfer instructions are executed only when the I/O device is **ready** to accept or supply data.

- MPU initiates the I/O devices and then continuously checks the status of I/O devices till the I/O device becomes **ready**. Each data transfer is preceded by a requesting signal sent by MPU and READY signal from the device.

- In this mode of data transfer, the data is read from an input device when the processor or CPU is ready and executes the data transfer instruction.

- If the input device is not ready the processor will wait until the device is ready with data.

- Similarly, the data is written onto an output device by the processor when it executes the data ‘write’ instruction to the corresponding output device.

- The program is written and the processor will wait in a loop until the output device is **ready** to receive data.

- As it can be seen clearly, the **processor’s time is wasted** in this mode of data transfer as it **waits for the device** to be ready.

Disadvantages

- A lot of Processor time is wasted during looping to check the device status which may be prohibitive in many time critical situations.

- Some simple devices may not have status signals. In such a case Processor goes on checking whether data is available on the port or not. e.g a keyboard interfaced to MPU.

Interrupt driven data transfer

Basic Interrupt Structure

- Interrupt is a mechanism by which the processor is made to transfer control from its current program execution to another program of more importance or higher priority.
- The interrupt signal may be given to the processor by any external peripheral device.
- Interrupts are, in general, generated by a variety of sources either internal or external to the CPU.
- Interrupts are the primary means by which Input and Output devices obtain the services of the CPU.

Types of Interrupts

- Interrupts can be classified in the following ways:

1. Vectored and Non-vectored Interrupts
2. Maskable and Non-maskable Interrupts
3. Software and Hardware Interrupt

  - Vectored (the **address** of the service routine is hard-wired)
  - Non-vectored (the address of the service routine needs to be supplied externally by the device)

Interrupts…

- An interrupt is considered to be an emergency signal that may be serviced.
  - The Microprocessor may respond to it as soon as possible.

- What happens when MP is interrupted ?
  - When the Microprocessor receives an interrupt signal, it suspends the currently executing program and jumps to an Interrupt Service Routine (ISR) to respond to the incoming interrupt.
  - Each interrupt will have its corresponding ISR.

Responding to Interrupts

- Responding to an interrupt may be immediate or delayed depending on whether the interrupt is maskable or non-maskable.
- There are two ways of redirecting the execution to the ISR depending on whether the interrupt is vectored or non-vectored.
  - Vectored: The address of the subroutine is already known to the Microprocessor
  - Non Vectored: The device will have to supply the address of the subroutine to the Microprocessor accordingly.

The 8085 Interrupts

- When a device interrupts, it actually wants the MP to give a service which is equivalent to asking the MP to call a subroutine. This service subroutine is called ISR (Interrupt Service Routine or Interrupt Service Subroutine).

- The **maskable** interrupt process in the 8085 is controlled by a single flip flop inside the microprocessor. This **Interrupt Enable** flip flop is controlled using the two instructions **"EI" and "DI"**.

- The 8085 has a single Non-Maskable interrupt i.e. TRAP.
  - The non-maskable interrupt is not affected by the value of the Interrupt Enable flip flop.

The 8085 Interrupts

- The 8085 has 5 interrupt input pins.
  - The INTR input.
    - The INTR input is the only non-vectored interrupt.
    - INTR is maskable using the EI/DI instructions.
  - RST 5.5, RST 6.5, RST 7.5 are all automatically vectored.
    - RST 5.5, RST 6.5, and RST 7.5 are all maskable.
  - TRAP is the only non-maskable interrupt in the 8085
    - TRAP is also automatically vectored.

The 8085 Interrupts

Interrupt name	Maskable	Vectored
INTR	Yes	No
RST 5.5	Yes	Yes
RST 6.5	Yes	Yes
RST 7.5	Yes	Yes
TRAP	No	Yes

The 8085 Maskable/Vectored Interrupts

- The 8085 has 4 Masked and 4 Vectored interrupt inputs.
  - RST 5.5, RST 6.5, RST 7.5
    - They are all maskable.
    - They are automatically vectored according to the following table:

Interrupt	Vector
RST 5.5	002CH
RST 6.5	0034H
RST 7.5	003CH
TRAP	0024H

  - The vectors for these interrupt fall in between the vectors for the RST instructions. That's why they have names like RST 5.5 (RST 5 and a half).

Interrupt Vectors and the Vector Table

- An interrupt vector is a pointer to where the ISR is stored in memory.
- All interrupts (vectored or otherwise) are mapped onto a memory area called the Interrupt Vector Table (IVT).
  - The IVT is usually located in memory page 00 (0000H - 00FFH).
  - The purpose of the IVT is to hold the vectors that **redirect** the microprocessor to the right place when an interrupt arrives.

The 8085 Non-Vectored Interrupt Process

- The 8085 recognizes 8 RESTART instructions: **RST0 - RST7.**
  - each of these would send the execution to a predetermined memory location on page 00:

Restart Instruction	Equivalent to
RST0	CALL 0000H
RST1	CALL 0008H
RST2	CALL 0010H
RST3	CALL 0018H
RST4	CALL 0020H
RST5	CALL 0028H
RST6	CALL 0030H
RST7	CALL 0038H

The 8085 **Non-Vectored** Interrupt Process

1. The interrupt process should be enabled using the EI instruction.
2. The 8085 checks for an interrupt during the execution of every instruction.
3. If INTR is high, MP completes current instruction, disables the interrupt and sends $\overline{\text{INTA}}$ (Interrupt acknowledge) signal to the device that interrupted.
4. $\overline{\text{INTA}}$ allows the I/O device to send a RST instruction through data bus.

Upon receiving the $\overline{\text{INTA}}$ signal, the interrupting device is expected to return the op-code of one of the 8 RST instructions.

The 8085 Non-Vectored Interrupt Process

5. When the microprocessor executes the RST instruction, received from the device, it saves the address of the next instruction on the stack and jumps to the appropriate entry in the IVT (which is on the page 00).

6. The IVT entry must redirect the microprocessor to the actual service routine.

7. Microprocessor then Performs the ISR.

8. The service routine must include the instruction EI to re-enable the interrupt process.

9. RET instruction at the end of the ISR allows the MP to retrieve the return address from the stack and the program is transferred back to where the program was interrupted.

- Example: Let , a device interrupts the Microprocessor using the RST 7.5 interrupt line.
  - Because the RST 7.5 interrupt is vectored, Microprocessor knows, in which memory location it has to go.
  - RST7.5 is equivalent to Call 003CH.
  - Microprocessor goes to 003C location and will get a JMP instruction there with the actual ISR address.
  - The Microprocessor will then, jump to the ISR location in the R/W memory and executes the ISR there and return back to the main program.

Interrupt Structure

- The important points in the interrupt structure of any microprocessor are:-
- The number and type of interrupt signals available.
- The address of the memory where **Interrupt Service Routine** (ISR) is located for a particular interrupt signal. This address is called as Interrupt Vector address.
- The masking and unmasking facility for the interrupt signals.
- This facility allows the programmer to execute the interrupt service routine only when required.

Interrupt Structure contd..

- The priority of interrupts when more than one interrupt signals are available.
- Handling and storing of information on the stack, about the interrupted program (status information).
- This information must be loaded in to CPU when interrupt service routine is executed and control RETURNS back from interrupt.
- The control should transfer back to the next location of the interrupted program.

Interrupt Driven Data Transfer

- In this scheme the Processor initiates an I/O device to get ready and then **it executes its main program** instead of remaining in the loop to check the status of the device.

- When the device gets ready, it sends a signal to the MPU through a special input line called an interrupt line.

- The Processor acknowledges the interrupt signal after completing/executing the current instruction.

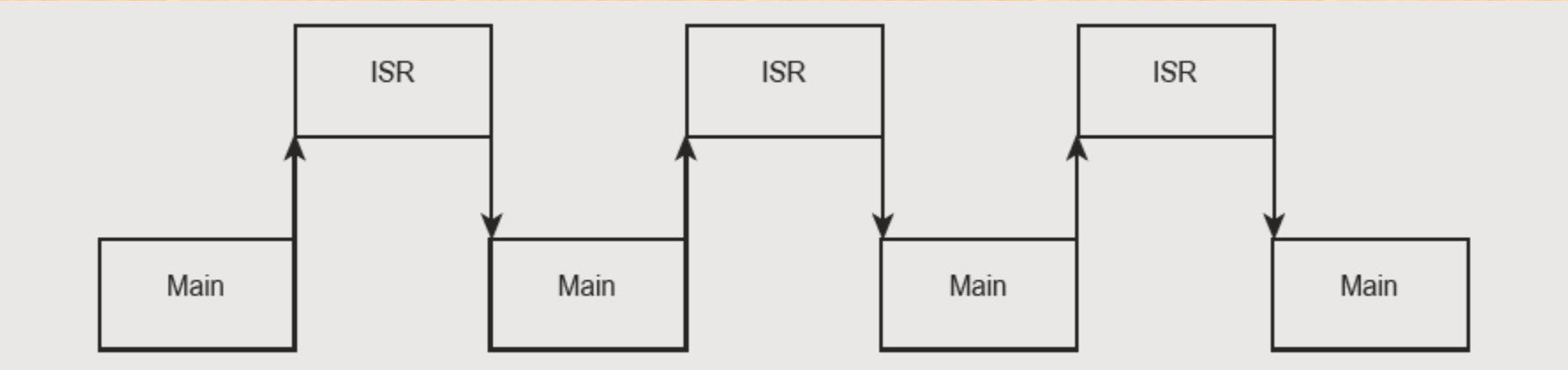
- The CPU responds to an interrupt request by a transfer of control to another program in a manner similar to a **subroutine call**.

- The Processor **saves** the contents of the PC (address of the next instruction) and supplementary information about current state (flags, registers, etc.) on the stack first.
- Load PC with the beginning address of an Interrupt Service Routine (ISR) in the IVT. Control then jump to the routine and then start executing it.

- Finish ISR/ISS when return instruction is executed.

- After returning from ISS, the Processor again loads the PC with the address that was earlier loaded in the stack and (other status information also) thus returns to the main program.

Transfer of control from Main program to ISR



- It is efficient because precious time of Processor is not wasted while the I/O device gets ready.

  In the earlier techniques, Processor has to either wait for the device to get ready or has to check the Ready status of device continuously.
- In this scheme the data transfer may also be initiated by the I/O device.

Multiple Interrupts

- 1. The Processor has one interrupt level and several I/O devices to be connected to it which are attended in the order of priority.

- 2. The MPU has several interrupt levels and one I/O device is to be connected to each interrupt level.

- 3. The MPU has several interrupt levels and more than one I/O devices are to be connected to each interrupt level.

- In this case the MPU executes multiple interrupts by using a device polling technique to know which device connected to which interrupt level has interrupted the processor.

Software Polling

- Each I/O device is equipped with a status FF. The processor reads this status FF of each device according to some pre-assigned priority and checks whether it is set or not. This process is continued until one device is found with set status or the status checking of all I/O devices is exhausted. Then the ISS of the device with set status is invoked.
- In this scheme the processor has an overhead for identifying the interrupting source, particularly, if the number of I/O devices is large.

- To overcome the problem of software polling, another approach is used, called **hardware polling**.

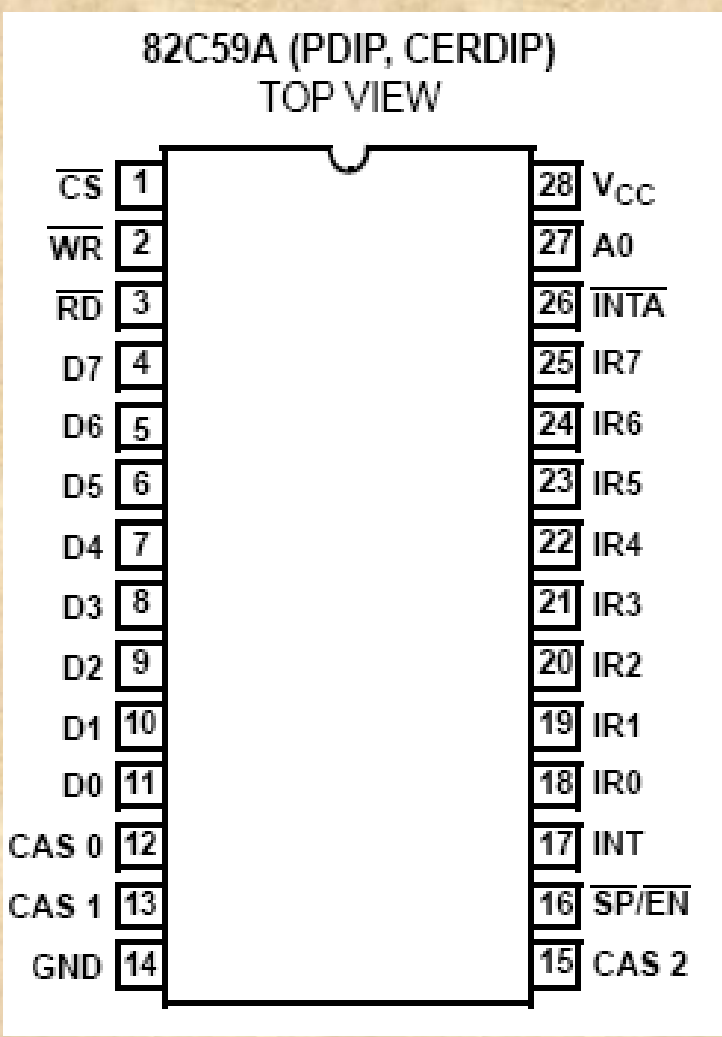
- **Hardware Polling:** External hardware circuit is used for this purpose. **Programmable Interrupt Controller (PIC)** chip is used with the processor to implement hardware polling.

Programmable Interrupt Controller (8259)

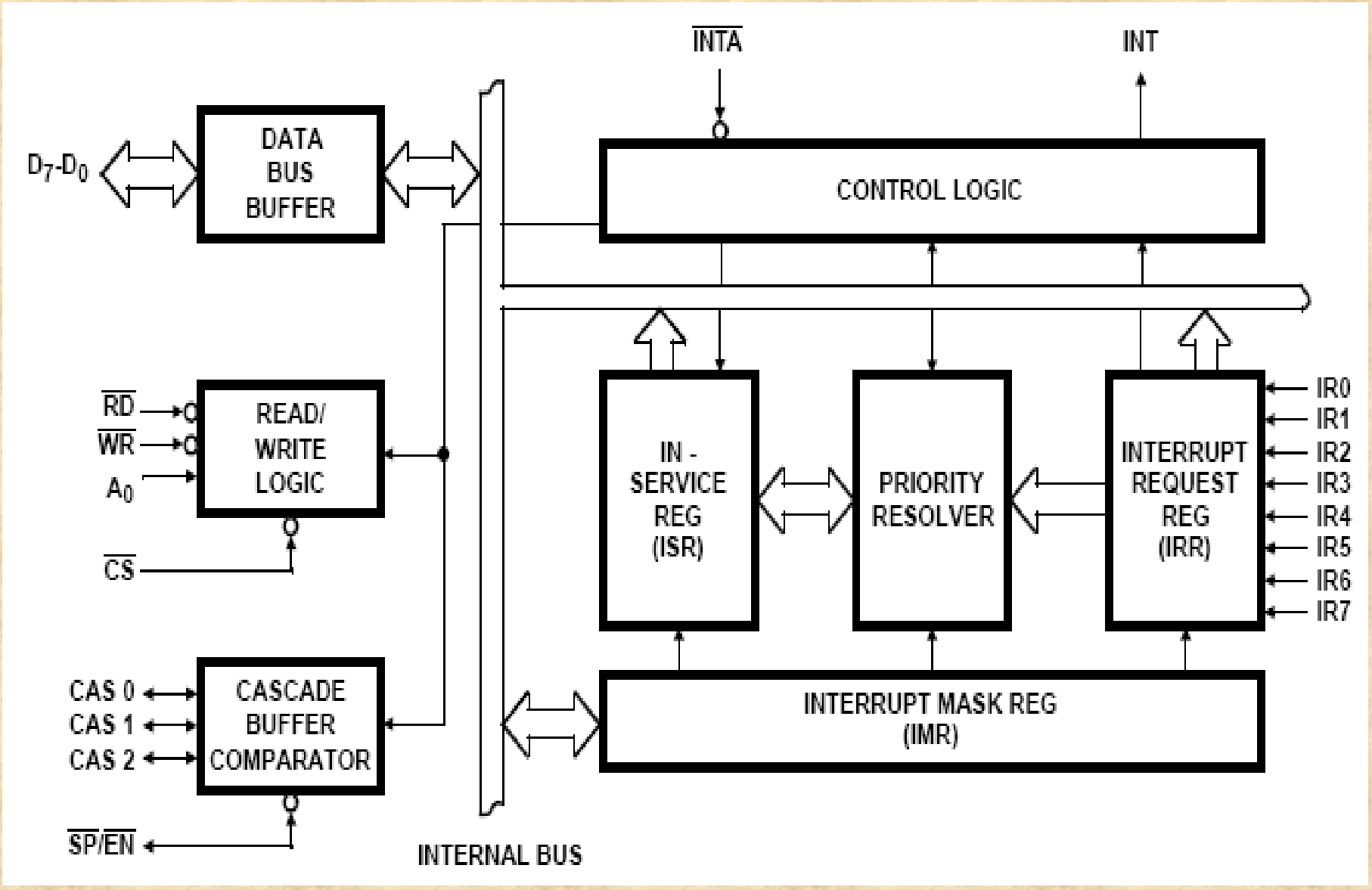
- It (8259) is the Programmable Interrupt Controller.
- It is a 28 pin IC
- The 8259 is designed to relieve the system CPU from the task of polling in a multi-level priority system.
- The high speed and industry standard configuration of the 8259 make it compatible with processors such as 8085, 8086, 80286 etc.
- The 8259 can handle up to eight vectored priority interrupting sources and is cascadable to 64, without additional circuitry.

8259A- PIN OUT

PIN	DESCRIPTION
D7 - D0	Data Bus (Bidirectional)
$\overline{RD}$	Read Input
$\overline{WR}$	Write Input
A0	Command Select Address
$\overline{CS}$	Chip Select
CAS 2 - CAS 0	Cascade Lines
$\overline{SP}/\overline{EN}$	Slave Program Input Enable
INT	Interrupt Output
$\overline{INTA}$	Interrupt Acknowledge Input
IR0 - IR7	Interrupt Request Inputs



8259A-FUNCTIONAL BLOCK

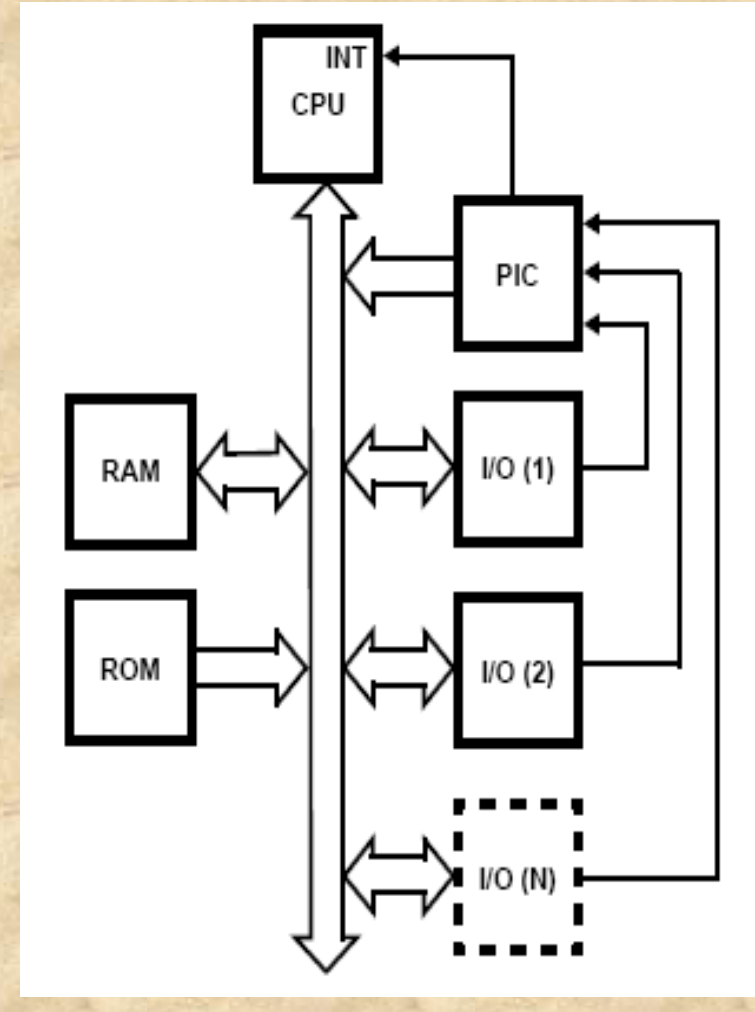


8259A-FUNCTIONAL DESCRIPTION

- The Programmable Interrupt Controller (PIC) functions as an overall manager in an Interrupt-Driven system.

- It accepts requests from the peripheral devices, determines which of the incoming requests is of the highest importance (priority), ascertains whether the incoming request has a higher priority value than the level currently being serviced, and issues an interrupt to the MPU based on this determination.

  The PIC, after issuing an interrupt to the Processor, inputs the information to the Processor that can “point” the Program Counter to the service routine associated with the requesting device.



8259A- INTERRUPT PROCEDURE

I/O Devices can request for interrupt through the IR0-IR7 lines. The input requests are received and recorded in the 8-bit IRR register. IMR register (8-bit) holds the information about the masking of interrupt requests and the ISR register holds the information about the interrupt request **currently under process**. Whenever a new interrupt request appears on the IRR, the priority resolver checks the status of the IMR and ISR and the highest priority device is selected accordingly .

8259A- INTERRUPT PROCEDURE

These events occur during this:

1. One or more of the INTERRUPT REQUEST lines (IR0 - IR7) are raised high, setting the corresponding IRR bit(s).
2. The 8259A evaluates those requests in the priority resolver and sends an interrupt (INT) to the processor accordingly.
3. The processor acknowledges the INT and responds with an $\overline{\text{INTA}}$ pulse.
4. Upon receiving an $\overline{\text{INTA}}$ from the processor, the highest priority ISR bit is set, and the corresponding IRR bit is reset.

8259A-CASCADING

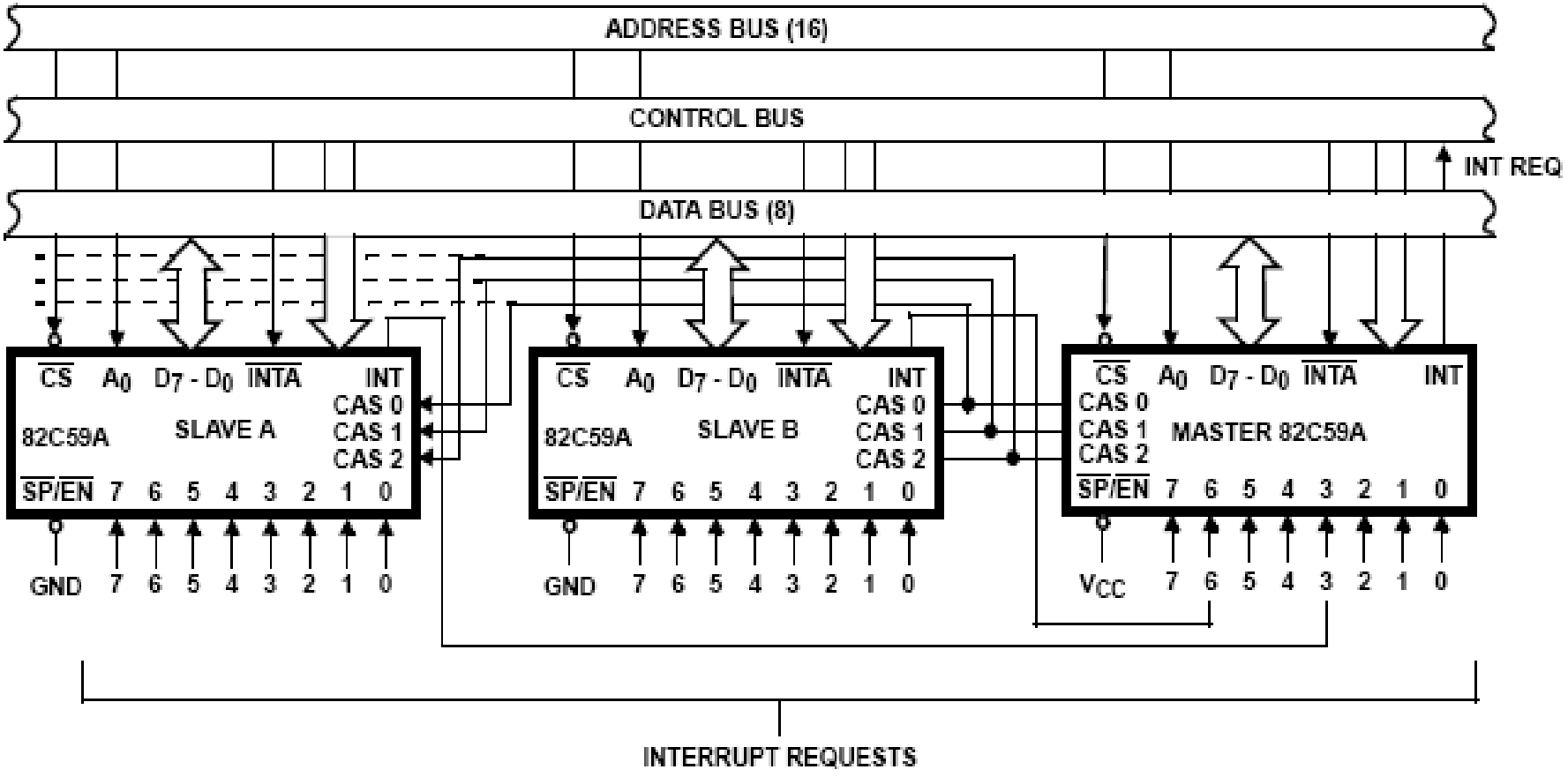


FIGURE 11. CASCADING THE 82C59A

CASCADING

Cascade Mode

- The 8259A can be easily interconnected in a system of one master with up to eight slaves to handle up to 64 priority levels.
- The master controls the slaves through the 3 line cascade bus (CAS0,1,2). The cascade bus acts like chip selects to the slaves during the $\overline{\text{INTA}}$ sequence.
- In a cascade configuration, the slave interrupt outputs (INT) are connected to the master interrupt request inputs (IR0…7). When a slave request line is activated and afterwards acknowledged, the master will enable the corresponding slave to release the device routine address accordingly.

Interrupts of 8085

On the basis of priority the interrupt signals are as follows

- TRAP
- RST 7.5
- RST6.5
- RST5.5
- INTR

These interrupts are implemented by the hardware

Interrupt Instructions

- EI (Enable Interrupt) This instruction sets the interrupt enable Flip Flop to activate the interrupts.

- DI (Disable Interrupt) This instruction resets the interrupt enable Flip Flop and deactivates all the interrupts except the non-maskable interrupt i.e. TRAP

- SIM (Set Interrupt Mask) This enables/disables interrupts according to the bit pattern in accumulator obtained through masking.

- RIM (Read Interrupt Mask): This instruction helps the programmer to know the current status of pending interrupt.

Accumulator bit pattern for SIM instruction

Bit position	D7	D6	D5	D4	D3	D2	D1	D0
Name	SOD	SDE	X	R7.5	MSE	M7.5	M6.5	M5.5
Explanation	Serial data to be sent	Serial data enable—set to 1 for sending	Not used	Reset RST 7.5 flip-flop	Mask set enable—Set to 1 to mask interrupts	Set to 1 to mask RST 7.5	Set to 1 to mask RST 6.5	Set to 1 to mask RST 5.5

RIM Instruction

Bit position	D7	D6	D5	D4	D3	D2	D1	D0
Name	SID	I7.5	I6.5	I5.5	IE	M7.5	M6.5	M5.5
Explanation	Serial input data in the SID pin	Set to 1 if RST 7.5 is pending	Set to 1 if RST 6.5 is pending	Set to 1 if RST 5.5 is pending	Set to 1 if interrupts are enabled	Set to 1 if RST 7.5 is masked	Set to 1 if RST 6.5 is masked	Set to 1 if RST 5.5 is masked

DMA

- **Programmed data transfers** move data from memory to the accumulator, and then from the accumulator to the output ports.
- A program has to be written to transfer data from a device to the memory in programmed data transfers. Thus the programmed data transfer is a slow process.
- This causes a problem only while transferring large amounts of data.

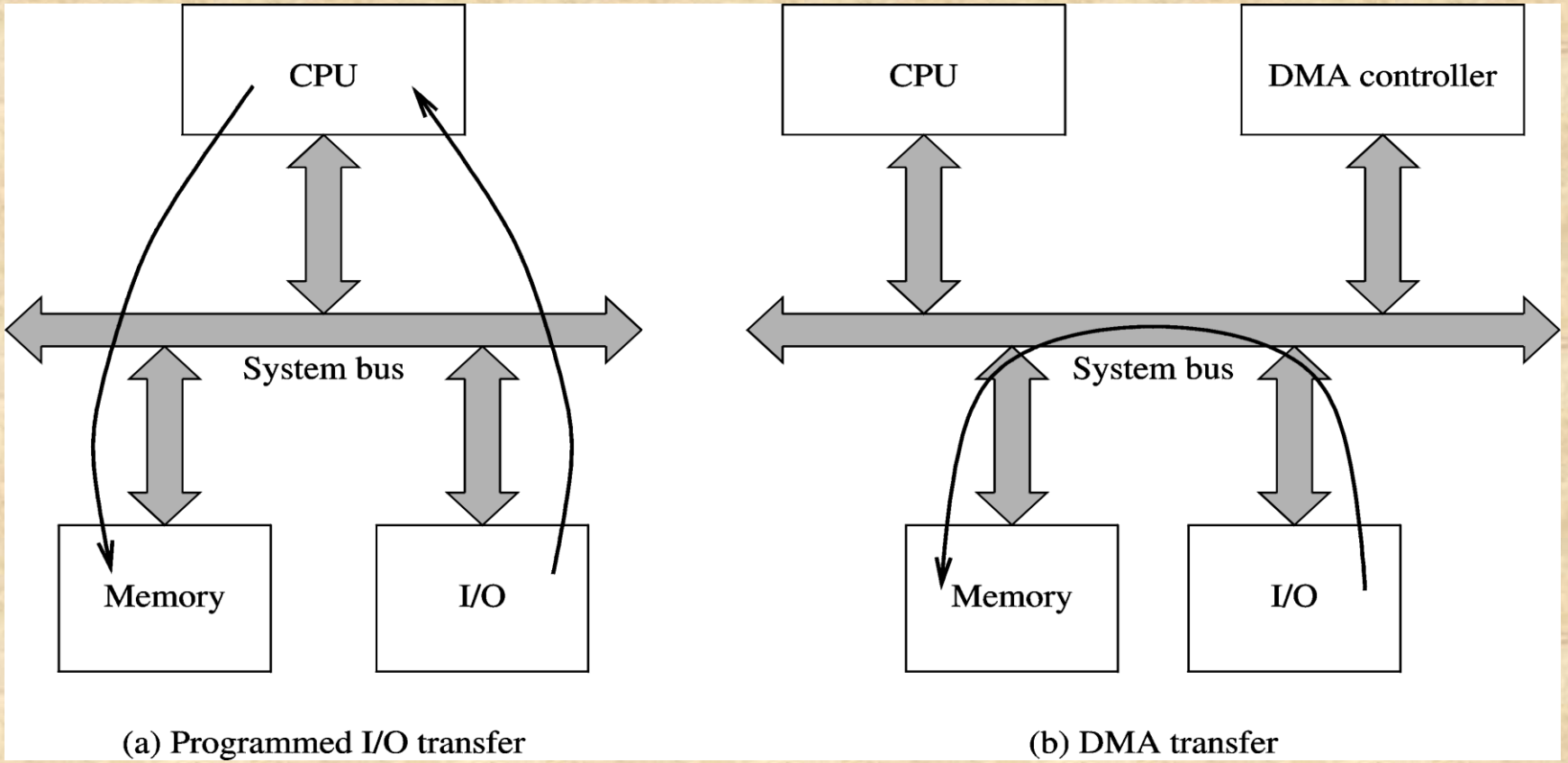
- DMA stands for **Direct Memory Access**. It is one of the ways to accomplish high-speed data transfers directly between memory and peripheral devices.
- DMA is a method of data transfer between Memory and I/O devices without the intervention of microprocessor. This method is often used when large block of data is to be transferred.

DMA Data Transfer scheme

- Data transfer from I/O device to memory or vice-versa is controlled by a DMA controller.
- This scheme is employed when **large amount** of data is to be transferred.
- The DMA requests the control of buses through the HOLD signal and the MPU acknowledges the request through HLDA signal and releases the control of buses to DMA.
- It's a faster scheme and hence may be used for high speed printers.

- Processor provides starting address of the memory location from which the data is started,
- The size (total number of bytes) of data to be transferred,
- Control signal is used to decide the type of operation e.g. read, write.

DMA Data Transfer (cont'd)



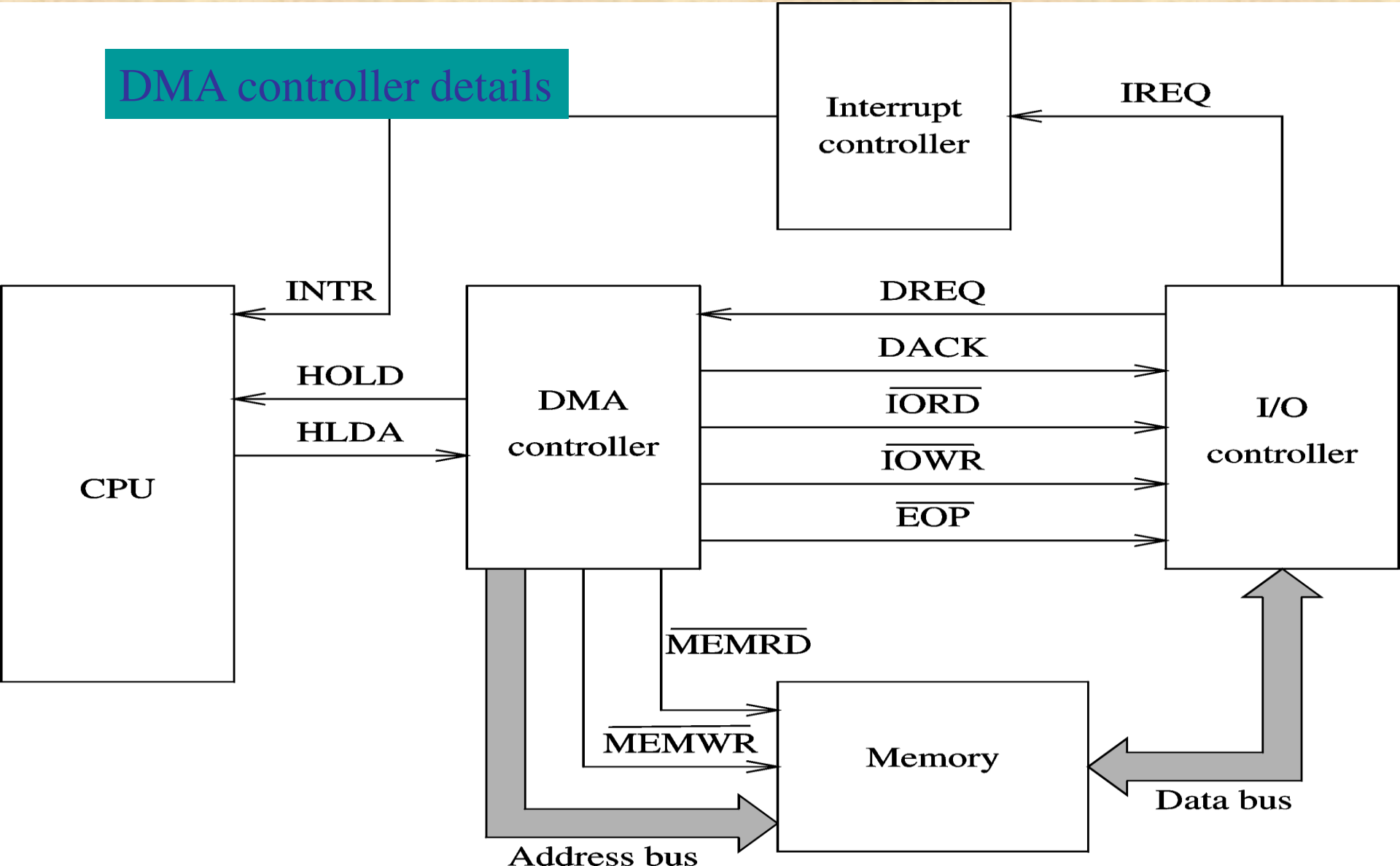
DMA Data Transfer (cont'd)

- DMA is implemented using a DMA controller
  - DMA controller
    - Acts as slave to the processor
    - Receives instructions from processor
    - Example: Reading from an I/O device
      - Processor gives details to the DMA controller
        - » I/O device number
        - » Main memory location (address)
        - » Number of bytes to transfer
        - » Direction of transfer (memory → I/O device, or vice versa)

DMA Data Transfer (cont'd)

- Steps in a DMA operation
  - Processor initiates the DMA controller
    - Gives device number, memory buffer pointer, …etc.
      - Called ***channel initialization***
    - Once initialized, it is ready for data transfer
  - When ready, I/O device informs the DMA controller
    - DMA controller starts the data transfer process
      - Obtains bus by going through bus arbitration (taking bus from processor)
      - Places memory address and appropriate control signals
      - Completes transfer and releases the bus
      - Updates memory address and count value
      - If more to read, loops back to repeat the process

DMA Data Transfer (cont'd)

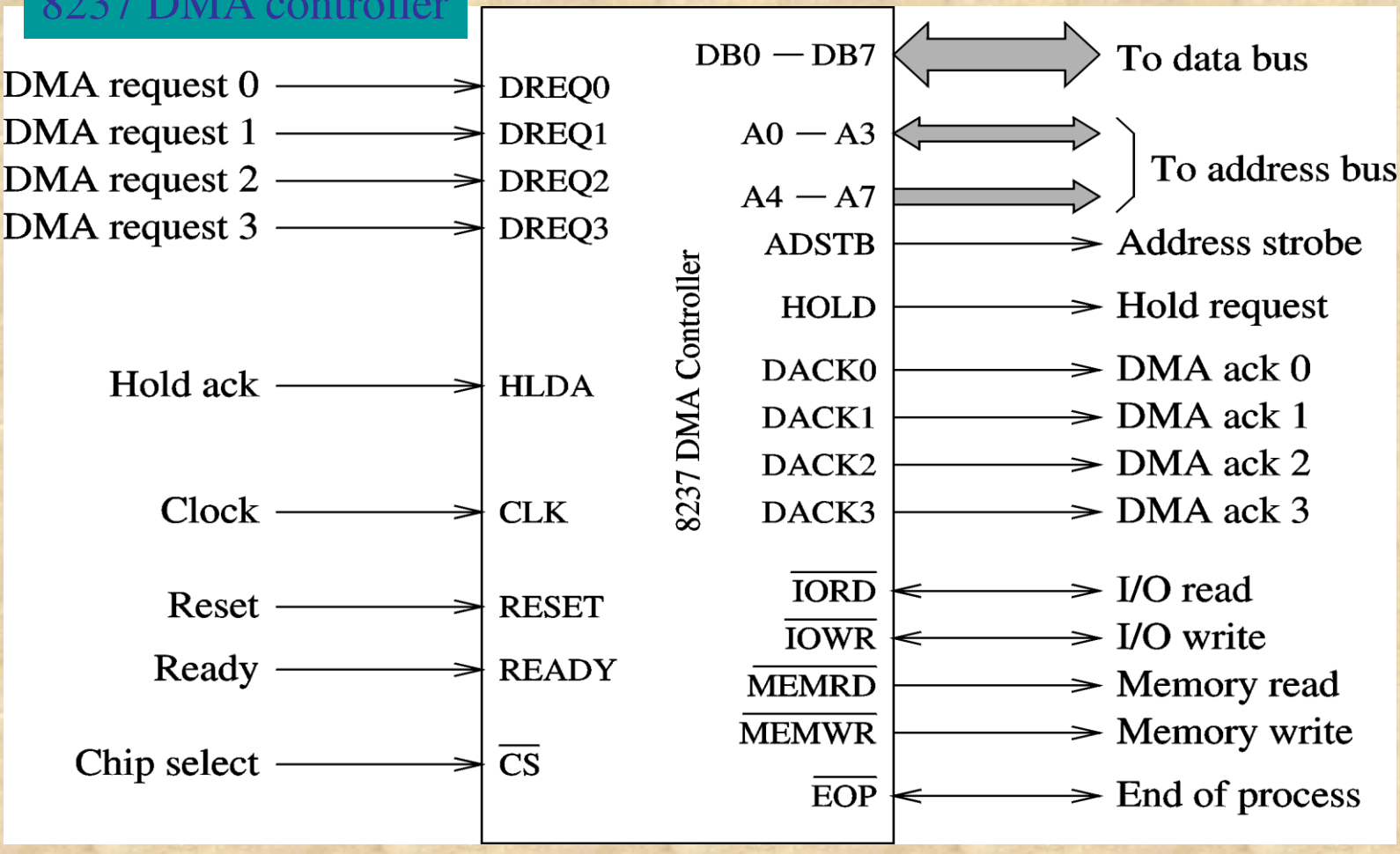


DMA Controller

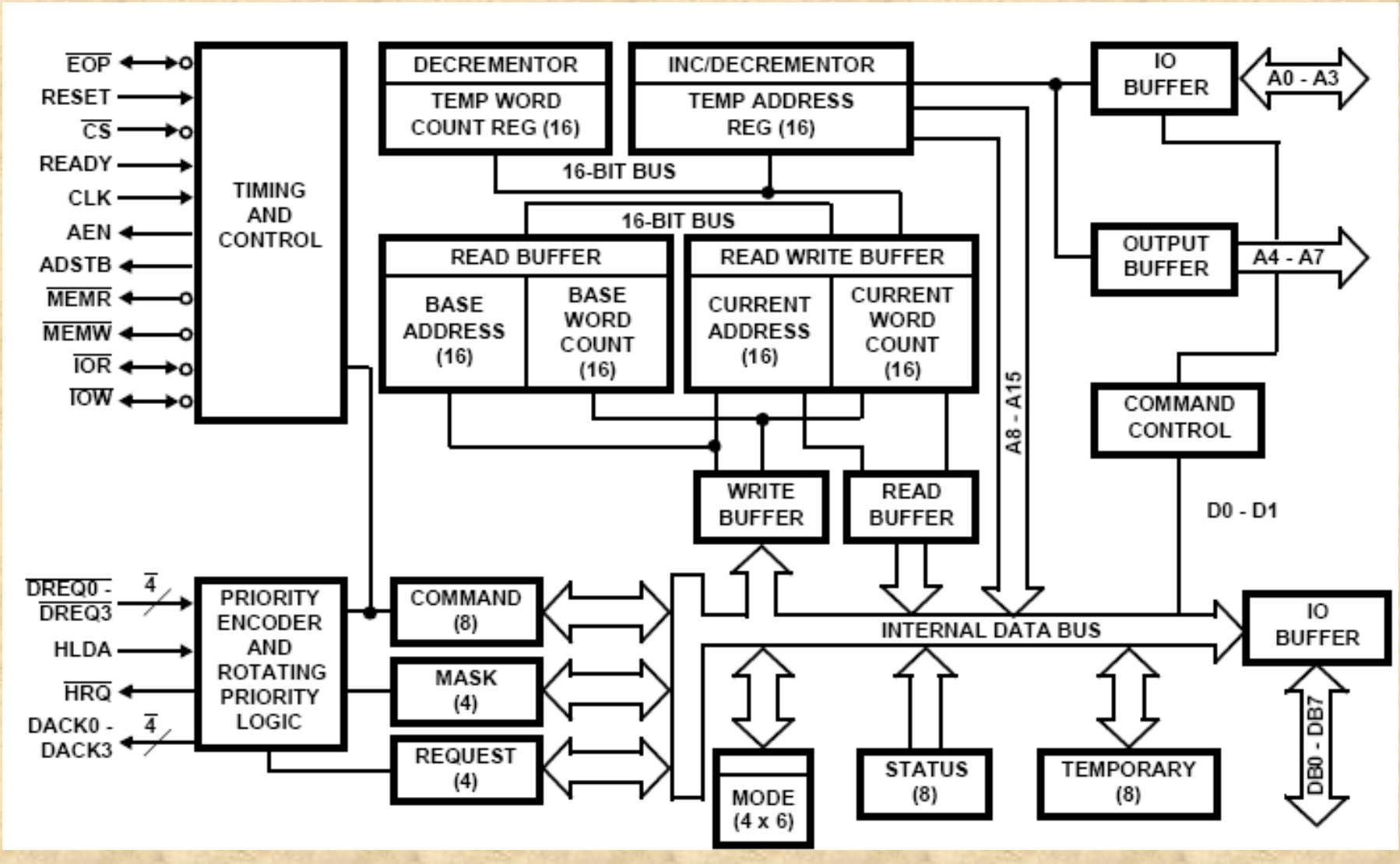
- DMA controller commonly used with 8085 and 8088 processors is the **8237** programmable DMA controller device.
- The 8237 is a **4-channel** device. Each channel is dedicated to a specific peripheral device and capable of addressing 64 K bytes section of memory.

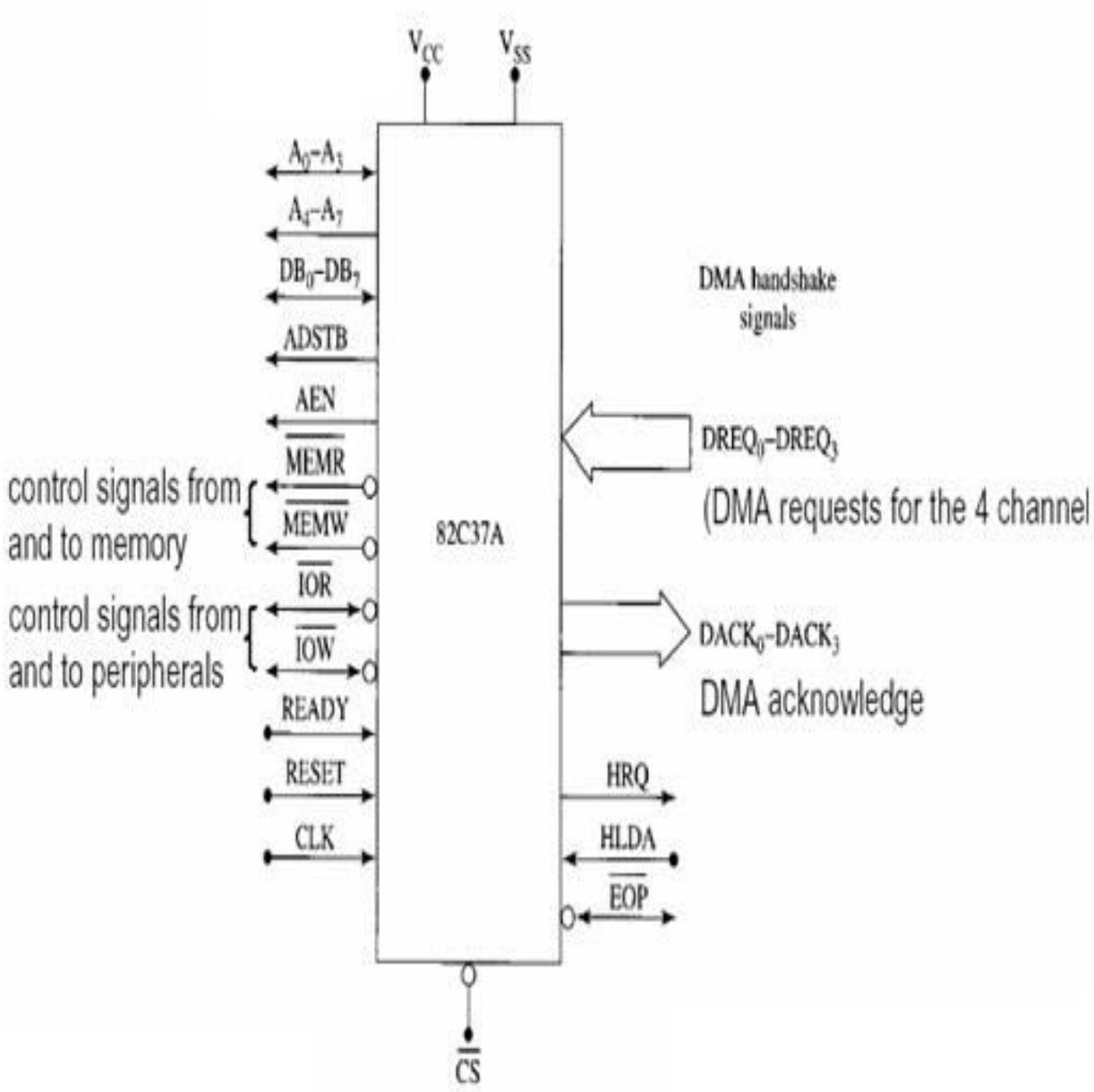
DMA Data Transfer (cont'd)

8237 DMA controller



Architecture and pin details of 8237





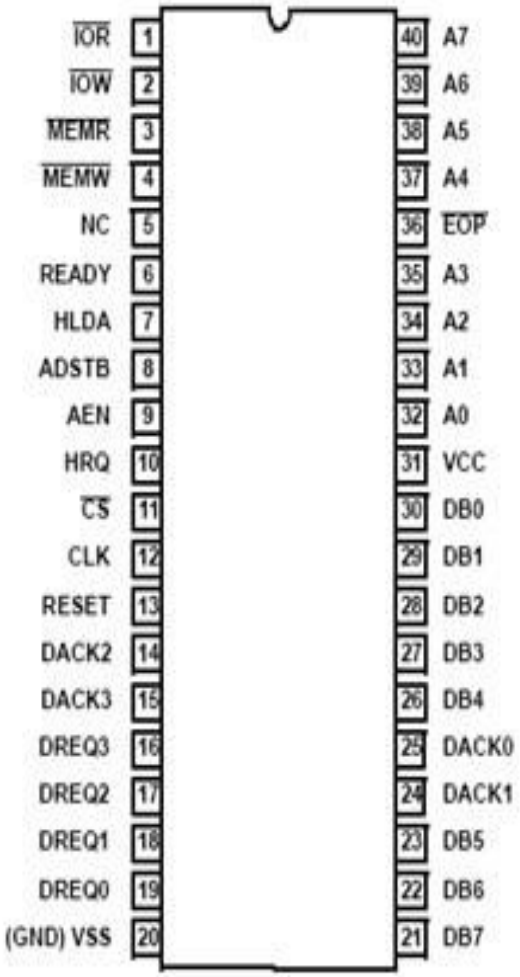


Figure 7.65 Pin details of 8237

DMA Data Transfer (cont'd)

- 8237 supports four DMA channels
- It has the following internal registers
  - Current address register
    - One 16-bit register for each channel
    - Holds address for the current DMA transfer
  - Current word register
    - Keeps the byte count
    - Generates terminal count (TC) signal when the count goes from zero to FFFFH
  - Command register
    - Used to program 8257 (type of priority, …)

DMA Data Transfer (cont'd)

- Mode register
  - Each channel can be programmed to
    - Read or write
    - Auto-increment or auto-decrement the address
    - Auto-initialize the channel
- Request register
  - For software-initiated DMA
- Mask register
  - Used to disable a specific channel
- Status register
- Temporary register
  - Used for memory-to-memory transfers

- The following sequences explain the DMA method of data transfer in detail.
- The starting address of the data is loaded into the 8237 Current and Base Address registers for a particular channel, and the length of the block is loaded into the channel's Word Count Register.
- The corresponding Mode register is programmed for a memory-to-I/O operation (read transfer), and various options are selected by the Command register and the other Mode register bits.
- The channel's mask bit is cleared to enable recognition of a DMA request (DREQ). The DREQ can either be a hardware signal or a software command.

- When the peripheral device has the first byte of data ready, it sends a DMA request - DREQ signal to the DMA controller.

- If the input (channel) of the DMA controller is unmasked, the DMA controller will send a hold-request - HRQ signal to the microprocessor HOLD input.
- The microprocessor will respond to this input by floating its buses and sends a hold-acknowledge signal, to the DMA controller.

- When the DMA controller receives the HLDA signal, it will send out a control signal disconnects the processor from the buses and connects the DMA controller to the buses.
- When the DMA controller gets control of the buses, it sends out the memory address where the first byte of data from the peripheral device is to be written.
- Then the DMA controller sends a DMA-acknowledge, DACKO, signal to the peripheral device to tell it to get ready to the byte.

- Finally, the DMA controller asserts both the and the lines on the control bus.
- Asserting the signal enables the addressed memory to accept data written into it.
- Asserting the signal enables the disk controller to output the byte of data from the disk on the data bus.
- The byte of data then is transferred directly from the peripheral device to the memory location without passing through the CPU or the DMA controller.

- When the data transfer is complete, the DMA controller unsets its hold-request signal to the processor and releases the buses. This lets the processor take over the buses again until another DMA transfer is needed.
- The processor continues executing from where it left off in the program.
- The entire process is explained with the flowchart given in next slide figure 7.74.

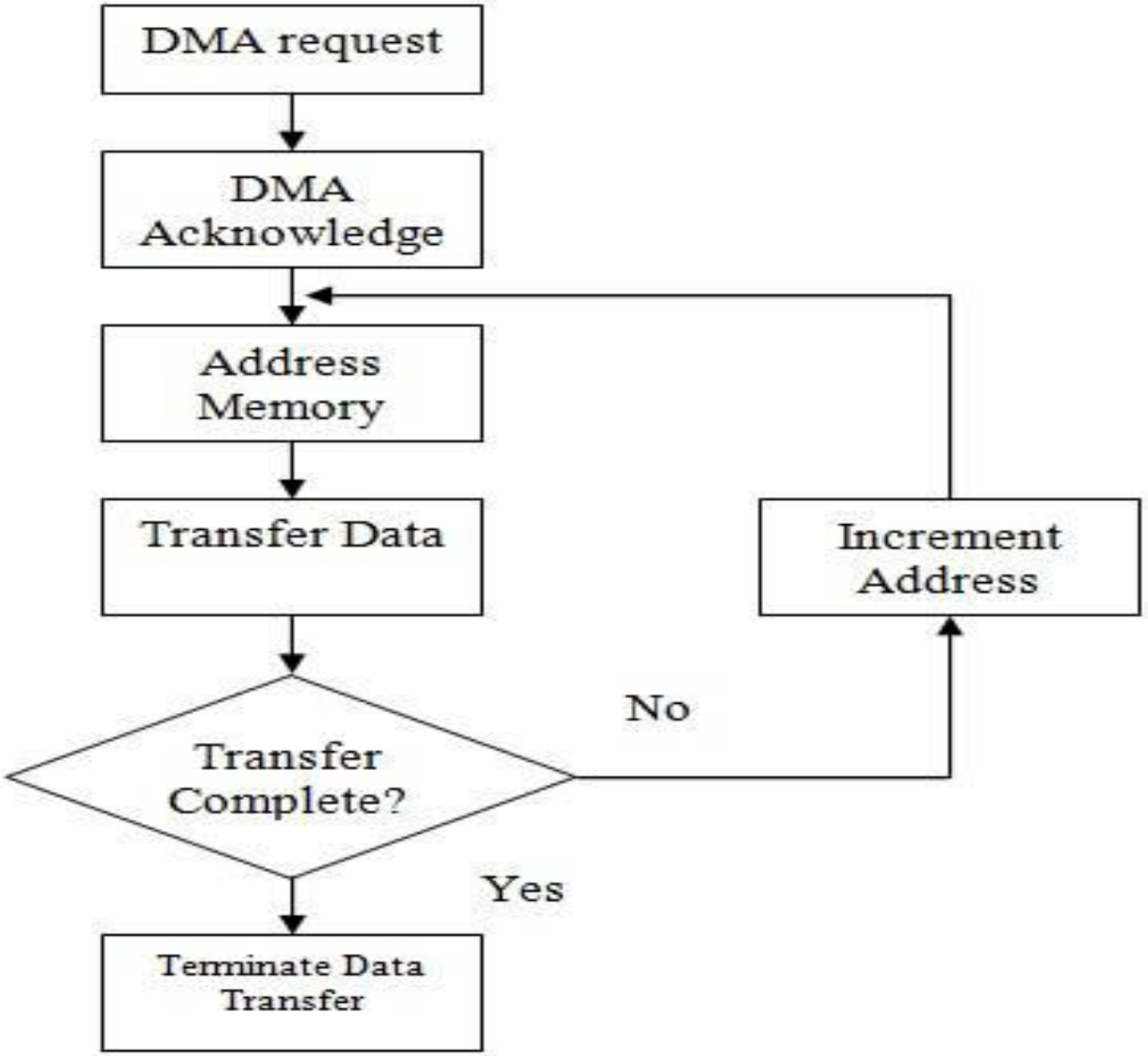


Fig 7.74 Sequence of Operation in Direct Memory Access

Block mode of data transfer

In this scheme the I/O device withdraws the DMA request only after all the data bytes have been transferred.

Cycle stealing technique

In this scheme the bytes are divided into several parts and after transferring every part the control of buses is given back to MPU and later stolen back when MPU does not need it.

SERIAL MODE OF DATA TRANSFER

There are typical situations where serial mode of data transfer is required like:

1. The serial mode is used for cost considerations as it requires only one pair of wires for data transfer.
2. Availability of suitable serial communication media for long distance transfer e.g. telephone network.
3. Inherent characteristics of many I/O devices forces the processor to store and retrieve data in serial mode e.g. printer, tape etc.

Serial data transfer contd..

- **Drawback of parallel data transfer**:

a) The parallel data transfer has the drawback of many wires needed to transfer all the bits of data.

b) So, the parallel data transfer can not be effectively used for long distance transfers.

c) As one wire is used for each bit, byte wise data transfers are eight times more expensive than a single bit transfer.

Serial data transfer is the solution for data transfers over long distances.

Also, serial communications are low cost way to send data over long distances.

MODES OF SERIAL DATA TRANSFER

(i) Simplex mode: Data transfer takes place only in one direction. e.g. from a computer to a remote printer.

(ii) **Full duplex**: There are two serial links one for each direction. Data transfer can take place in both directions independently and simultaneously.

(iii) **Half-Duplex**: Data transfer in both directions can be achieved by using a single link. However, it needs some protocol to be established to determine the direction of the link (typical wireless set of police deptt.).

IO Addressing

In programmed IO the CPU, main memory and IO devices communicate via the system bus.

The address bus lines of the system bus that are used to select memory locations are also used to select IO devices.

The IO devices are connected to the system bus via IO ports. IO ports are a little different than memory locations.

There are two different methods of IO addressing namely ‘Memory Mapped IO’ and ‘IO Mapped IO’.

Memory Mapped I/O

- In memory mapped I/O, each input device or output device is treated as if it is a memory location.
- The control signal for read and write operation of I/O device is same as that of memory chips.
- Each input or output device is identified by a unique address in the memory address range.

Memory Mapped IO

(i) It assigns a part of main memory address space to IO ports.

(ii) The system uses same address space for both memory and IO.

(iii) The CPU treats an IO port register as being part of the memory system .

(iv) A memory referencing instruction that loads or stores instruction at an address X can also be used to access an IO instruction if X is made the address of an IO port.

(v) The same memory load and stores instructions are used to transfer data words to or from IO ports.

(vi) No special IO instructions are needed.

(vii) The same read and write control lines are used to initiate IO transfer. All the memory related instructions used to read data from memory are used to access input and output device.

(viii) Since the I/O devices use some of the memory address space, the maximum memory addressing capacity will be reduced in a microprocessor based system.

I/O mapped I/O data transfer

- In ‘I/O mapped I/O’ device data transfer method, the I/O devices are treated separately from memory.
- Separate address range will be assigned for the input and output devices.
- The signals for read and write from I/O devices are completely separate from the signals used for memory access.

IO Mapped IO

(i) The memory and IO address spaces are separate.

(ii) Separate control lines for memory and IO reference.

(iii) A memory referencing instruction activates Read M and Write M control lines which does not affect the IO devices.

(iv) CPU must execute separate IO instructions to activate Read IO and Write IO lines.

(v) The IO device and memory location can have the same address.

(vi) When the CPU fetches and decodes IO instruction, it places the address associated with the instruction onto the common address lines.

IO Mapped IO

- (vii) Similarly for memory accesses the CPU enables the Read M and Write M control lines.
- (viii) The microprocessor will have separate instructions for Input and output device access such as IN instruction and OUT instruction of 8085.
- (ix) As the memory and I/O device access is completely different, a single address can be assigned to both an I/O device and a memory location.