8086 INTERRUPTS

- An interrupt is a special condition that arises during the working of a µP.
- The µP services it by executing a subroutine called Interrupt Service Routine (ISR).
- There are 3 sources of interrupts for 8086:

External Signal (Hardware Interrupts):

These interrupts occur as signals on the external pins of the µP.
8086 has two pins to accept hardware interrupts, NMI and INTR.

Special instructions (Software Interrupts):

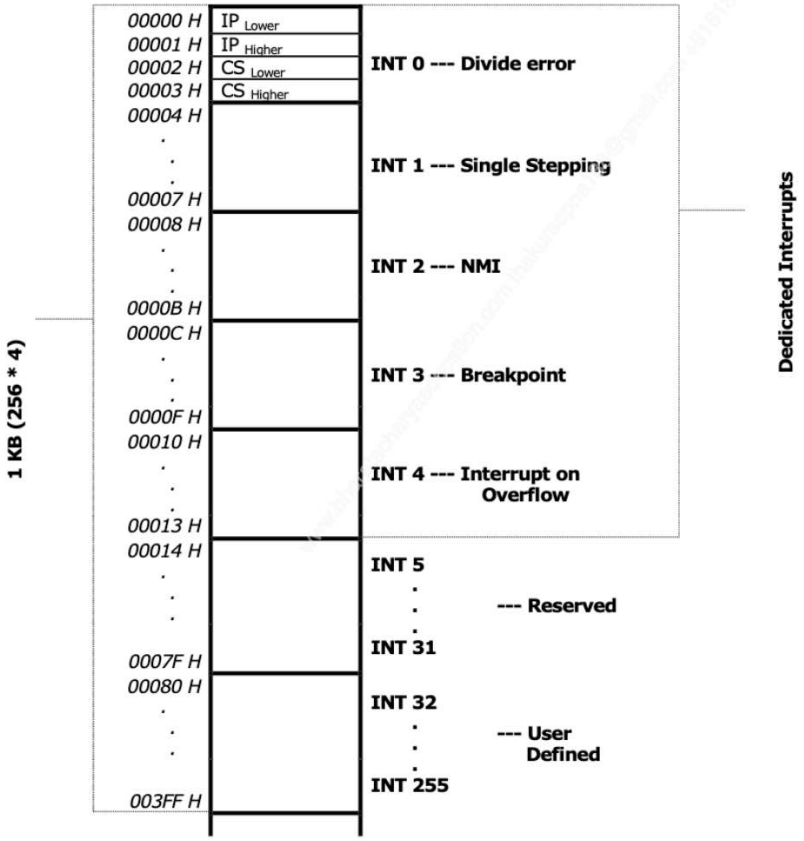
These interrupts are caused by writing the software interrupt instruction INTn where "n" can be any value from 0 to 255 (00H to FFH).
Hence all 256 interrupts can be invoked by software.

Condition Produced by the Program (Internally Generated Interrupts):

8086 is interrupted when some special conditions occur while executing certain instructions in the program.
Eg: **An error in division** automatically causes the INT 0 interrupt.

INTERRUPT VECTOR TABLE (IVT) {10M --- IMPORTANT }

The **IVT contains ISR address** for the 256 interrupts.
Each ISR **address** is stored as **CS and IP**.
As each ISR address is of 4 bytes (2–CS and 2-IP), each ISR address requires 4 locations to be stored.
There are **256 interrupts**: INT 0 … INT 255 ∴ the **total size of** the **IVT** is 256 x 4 **= 1KB**.
The first 1KB of memory, address 00000 H … 003FF H, are reserved for the IVT.
Whenever an interrupt **INT N occurs**, **µP does N x 4 to get** values of **IP and CS from the IVT and hence perform the ISR**.



DEDICATED INTERRUPTS (INT 0 ... INT 4)

1) **INT O** (Divide Error)
   This interrupt occurs whenever there is **division error**
   i.e. when the result of a division is too large to be stored.
   This condition normally occurs when the divisor is very small as compared to the dividend or the divisor is zero. #Refer example from Bharat Sir's lecture notes...
   Its ISR address is stored at location 0 x 4 = 00000H in the IVT.

2) **INT 1** (Single Step)
   The µP executes this interrupt **after every instruction if the TF is set**.
   It puts µP in **Single Stepping** Mode i.e. the µP pauses after executing every instruction.
   This is very useful during **debugging**. #Refer example from Bharat Sir's lecture notes...
   Its ISR generally displays contents of all registers.
   Its ISR address is stored at location 1 x 4 = 00004H in the IVT.

3) **INT 2** (Non Maskable Interrupt)
   The µP executes this ISR in **response to** an interrupt on the **NMI** line.
   Its ISR address is stored at location 2 x 4 = 00008H in the IVT.

4) **INT 3** (Breakpoint Interrupt)
   This interrupt is used to cause **Breakpoints** in the program.
   It is caused by writing the instruction INT 03H or simply INT.
   It is useful in **debugging large programs** where Single Stepping is inefficient.
   Its ISR is used to **display** the **contents of all registers** on the screen.
   Its ISR address is stored at location 3 x 4 = 0000CH in the IVT.

5) **INT 4** (Overflow Interrupt)
   This interrupt occurs if the **Overflow Flag is set AND** the µP executes the **INTO** instruction (Interrupt on overflow). #Show example from Bharat Sir's lecture notes...
   It is used to detect overflow error in **signed arithmetic** operations.
   Its ISR address is stored at location 4 x 4 = 00010H in the IVT.

   Please Note: INT 0 ... INT 4 are called as dedicated interrupts as these interrupts are dedicated for the above-mentioned special conditions.

RESERVED INTERRUPTS

INT 5 ... INT 31
These levels are **reserved** by INTEL to be used in higher processors like 80386, Pentium etc. They are **not available** to the user.

User defined Interrupts

INT 32 ... INT 255
These are **user defined**, **software** interrupts.
ISRs for these interrupts are written by the users to service various user defined conditions.
These interrupts are invoked by writing the instruction INT n.
Its ISR address is obtained by the µP from location n x 4 in the IVT.

HARDWARE INTERRUPTS

1) **NMI** (Non Maskable Interrupt)
   This is a **non-maskable**, **edge** triggered, **high priority** interrupt.
   On receiving an interrupt on NMI line, the µP executes **INT 2.**
   µP obtains the ISR address from location 2 x 4 = 00008H from the IVT.
   It reads 4 locations starting from this address to get the values for IP and CS, to execute the ISR.
   ☺ **For doubts contact Bharat Sir on 98204 08217**

2) **INTR**
   This is a **maskable**, **level** triggered, **low priority** interrupt.
   On receiving an interrupt on INTR line, the µP executes **2** $\overline{\textbf{INTA}}$ pulses.
   1st $\overline{\textbf{INTA}}$ pulse --- the interrupting device **calculates** (prepares to send) the **vector number**.
   2nd $\overline{\textbf{INTA}}$ pulse --- the interrupting device **sends** the **vector number "N"** to the µP.
   Now µP multiplies N x 4 and goes to the corresponding location in the IVT to obtain the ISR address.
   INTR is a maskable interrupt.
   It is masked by making IF = 0 by software through **CLI** instruction.
   It is unmasked by making IF = 1 by software through **STI** instruction.

Response to any interrupt --- INT N

i) The µP will **PUSH Flag** register into the Stack.
SS:[SP–1], SS:[SP-2] ← Flag
SP ← SP – 2

ii) **Clear IF and TF** in the Flag register and thus disables INTR interrupt.
IF ← 0, TF ← 0

iii) **PUSH CS** into the Stack.
SS:[SP–1], SS:[SP-2] ← CS
SP ← SP – 2

iv) **PUSH IP** into the Stack.
SS:[SP–1], SS:[SP-2] ← IP
SP ← SP – 2

v) **Load new IP** from the IVT
IP ← [N x 4], [N x 4 + 1]

vi) **Load new CS** from the IVT
IP ← [N x 4 + 2], [N x 4 + 3]

Since CS and IP get new values, control shifts to the address of the ISR and the ISR thus begins. At the end of the ISR the µP encounters the IRET instruction and returns to the main program in the following steps.

Response to IRET instruction

i) The µP will **restore IP from the stack**
IP ← SS:[SP], SS:[SP+1]
SP ← SP + 2

ii) The µP will **restore CS from the stack**
CS ← SS:[SP], SS:[SP+1]
SP ← SP + 2

iii) The µP will **restore FLAG register from the stack**
Flag ← SS:[SP], SS:[SP+1]
SP ← SP + 2

Interrupt Priorities

Interrupt	Priority	
	(Simultaneous occurrence)	**(To interrupt another ISR)**
Divide Error, INT n, INTO	1 (**Highest**)	Can interrupt any ISR
NMI	2	
INTR	3	Cannot interrupt an ISR (IF, TF ← 0)
Single Stepping	4(**Lowest**)	

Priority in 8086 interrupts is of two types:

1. Simultaneous Occurrence:

When more than one interrupts occur simultaneously then, **all s/w interrupts except single stepping**, get the **highest priority**.
This is followed by **NMI**. Next is **INTR**. Finally, the **lowest priority** is of the **single stepping** interrupt.

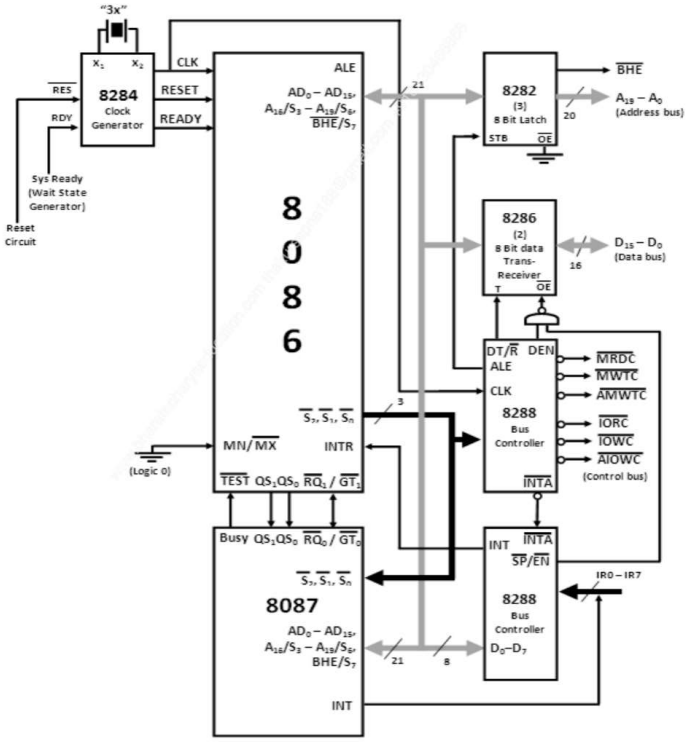
Eg: Assume the µP is executing a **DIV** instruction that causes a **division error** and **simultaneously INTR** occurs.
Here **INT 0** (Division error) will be **serviced first** i.e. its ISR will be executed, as it has higher priority, and **then INTR** will be **serviced**. #Please refer Bharat Sir's Lecture Notes for this ...

2. Ability to interrupt another ISR:

Since software interrupts (**INT N**) are **non-maskable**, they **can interrupt** any ISR.
NMI is also **non-maskable** hence it **can** also **interrupt** any ISR.
But **INTR** and **Single stepping cannot interrupt** another ISR **as** both are **disabled** before µP enters an ISR by **IF ← 0 and TF ← 0**.

Eg: Assume the µP executes DIV instruction that causes a **division error**. So µP gets the **INT 0** interrupt and now **µP enters the ISR for INT 0**. **During** the execution of **this ISR, NMI and INTR occur**.
Here **µP will branch out** from the ISR of INT 0 and **service NMI** (as **NMI is non-maskable**).
After completing the ISR of NMI **µP** will **return to** the **ISR for INT 0**.
INTR is **still pending but** the **µP will not service INTR** during the ISR of INT 0 (as **IF ← 0**).
µP will first **finish** the **INT 0 ISR** and **only then service INTR**.
Thus INTR and Single stepping cannot interrupt an existing ISR.

CO-PROCESSOR CONFIGURATION --- 8086 WITH 8087



1) As a co-processor (8087) is connected to 8086, 8086 operates in **Maximum Mode**.
   Hence **MN/ $\overline{MX}$ is grounded**.

2) **8284** provides the common **CLK, RESET, READY** signals,
   8282 are used to **latch** the **address**, **8286** are used as **data transreceivers**,
   8288 generates **control signals** using S_2, S_1, S_0 as input from the currently active processor,
   8259 PIC is used to accept the **interrupt from 8087** and send it **to** the **μP**.

3) This interface is also called a **Closely Coupled Co-Processor configuration**.
   Here **8086** is called as the **Host** and 8087 as **Co-Processor**, as it cannot operate all by itself.

4) We write a **homogeneous program** which contains both 8086 as well as 8087 instructions.

5) **Only 8086 can fetch instructions**, but these **instructions also enter 8087**.
   8087 treats 8086 instructions as NOP.

6) **ESC** is used as a **prefix for 8087 instructions**.
   When an instruction with ESC prefix (5 MSB bits as **11011**) is encountered, 8087 is activated.

7) The **ESC instruction** is **decoded by both** 8086 and 8087.

8) If **ESC is present 8087 executes** the instruction and 8086 discards it.
   If **ESC is not preset then 8086 executes the instruction** and 8087 discards it.

9) Once 8087 **begins execution** it makes the **BUSY o/p high** which is connected to the **$\overline{TEST}$** of μP.
   Now **8087** is **executing its instruction** and **8086** moves ahead with **its next instruction**.
   Hence ***MULTIPROCESSING*** *takes place*.

10) **During execution**, **if 8087 needs** to read/write more data (operands) from **the memory**, then it does so by **stealing bus cycles** from the μP in the following manner:
    The **$\overline{RQ}$ / $\overline{GT}$** of 8087 is connected to **$\overline{RQ}$ / $\overline{GT}$** of the μP.
    8087 gives an active low **Request pulse**.
    8086 completes the current bus cycle and **gives** the **grant pulse** and **enters** the **Hold state**.
    8087 uses the **shared system bus** to **perform** the **data transfer with** the **memory**.
    8087 gives the **release pulse** and **returns** the **system bus** back **to** the **μP**.

11) If **8086** requires the result of the 8087 operation, it first **executes** the **WAIT** instruction.
    WAIT makes the **μP check** the **$\overline{TEST}$** pin.
    If the **TEST** pin is **high** (**8087** is **BUSY**), then the **μP enters WAIT state**
    It **comes out** of it only **when** **$\overline{TEST}$** is low (**8087** has **finished its execution**).
    Thus 8086 gets the correct result of an 8087 operation.

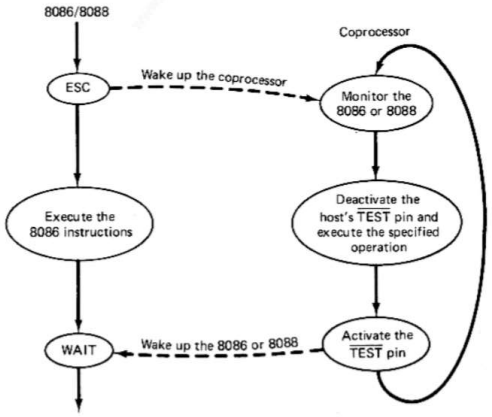
12) During the execution **if** an **exception occurs**, which is **unmasked**, **8087 interrupts μP** using the **INT** o/p **pin through** the **PIC 8259**.

13) The QS_0 and QS_1 lines are used by 8087 to monitor the queue of 8086.
8087 needs to know when 8086 will decode the ESC instruction so it synchronizes its queue with 8086 using QS_0 and QS_1 as follows: #Please refer Bharat Sir's Lecture Notes for this ...

QS_1	QS_0	8087 Operation
0	0	NOP
0	1	8087 removes Opcode from Queue and compares 5 MSB bits with 11011.
1	0	8087 clears its queue.
1	1	8087 removes operand if earlier comparison with Opcode is successful.

This is the **complete inter-processor communication** between 8086 and 8087 to form a Homogeneous System.

FLOWCHART (Optional – only for understanding)

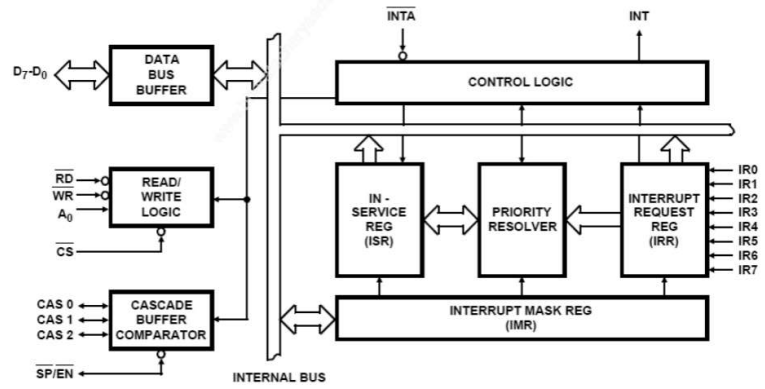


8259 - PIC

Salient Features

1) PIC 8259 is a Programmable Interrupt Controller that can work with 8085, 8086 etc.
2) **It is used to increase the number of interrupts.**
3) A single 8259 provides 8 interrupts while a cascaded configuration of 1 master 8259 and 8 slave 8259s can provide up to 64 interrupts.
4) 8259 can handle **edge as well as level triggered** interrupts.
5) 8259 has a **flexible priority** structure. ☺ For doubts contact Bharat Sir on 98204 08217
6) In 8259 interrupts can be **masked** individually.
7) The **Vector address** of the interrupts is **programmable**.
8) 8259 has to be **compulsorily initialized** by giving commands, to decide several properties such as Vector Numbers, Priority, Masking, Triggering etc.
9) In a **cascaded configuration**, **each 8259 has to be individually initialized**, master as well as each slave.

ARCHITECTURE OF 8259



1) Interrupt Request Register (IRR)
8259 has **8 interrupt** input lines **IR_7 ... IR_0**.
The IRR is an **8-bit register** having **one bit** for **each** of the **interrupt** lines.
When an **interrupt request** occurs on any of these lines, the **corresponding bit** is **set** in the Interrupt Request Register (**IRR**).

2) In-Service Register (InSR)
It is an **8-bit** register, which **stores** the **level** of the Interrupt Request, which is **currently** being **serviced**.

3) Interrupt Mask Register (IMR)
It is an **8-bit** register, which stores the **masking pattern** for the interrupts of 8259. It stores **one bit per interrupt level.**

4) Priority Resolver
It **examines** the **IRR**, **InSR**, and **IMR** and determines which interrupt is of **highest priority** and should be sent to the µP.

5) Control Logic
It has **INT output connected to** the **INTR** of the µP, to **send** the **Interrupt** to µP.
It also has the **$\overline{\text{INTA}}$ input** signal **connected to** the **$\overline{\text{INTA}}$** of the µP, to **receive** the interrupt **acknowledge**. It is also used to control the remaining blocks.

6) Data Bus Buffer
It is a bi-directional buffer used to **interface** the internal **data bus** of 8259 with the external (system) data bus.

7) Read/Write Logic
It is used to accept the **$\overline{\text{RD}}$** , **$\overline{\text{WR}}$** , A_0 and CS signal.
It also holds the Initialization Command Words (ICW's) and the Operational Command Words (OCW's).

8) Cascade Buffer / Comparator
It is used in **cascaded mode** of operation.
It has two components:

i. CAS_2, CAS_1, CAS_0 lines:
These lines are **output for the master**, **input for the slave**.
The **Master sends** the **address of** the **slave** on these lines (hence output).
The **Salves read** the **address** on these lines (hence input).
As there are 8 interrupt levels for the Master, there are **3 CAS lines ($2^3 = 8$).**

ii. $\overline{\text{SP}}$ / $\overline{\text{EN}}$ (Slave Program/Master Enable):
In **Buffered Mode**, it **functions** as the **EN line** and is used to **enable** the **buffer**.
In **Non buffered mode**, it **functions** as the **SP output line**.
For Master 8259 **SP** should be **high**, and **for** the **Slave SP** should be **low**.

INTERFACING AND WORKING OF A "SINGLE" 8259

A single 8259 can accept 8 interrupts.
Whenever a device interrupts 8259, 8259 will interrupt the **µP on INTR pin.**
Hence, first the **INTR** signal of the µP **should be enabled** using the **STI** instruction**.**
8259 is **initialized** by giving **ICW1** and **ICW2** (compulsory) and **ICW4** (optional).
Note that **ICW3** is **not given** as Single 8259 is used. OCWs are given if required.

Once 8259 is **initialized**, the **following sequence** of events takes place when one or more **interrupts occur** on the IR lines of the 8259.

1) The **corresponding bit** for an interrupt is **set in IRR**.

2) The **Priority Resolver checks** the 3 registers:
IRR (for highest interrupt request)
IMR (for the masking Status)
InSR (for the current level serviced)
and **determines** the **highest priority** interrupt.
It **sends** the **INT** signal **to** the **µP**.

3) The **µP finishes** the **current instruction** and **acknowledges** the interrupt by **sending** the **first $\overline{\text{INTA}}$ pulse**.

4) On receiving the first **$\overline{\text{INTA}}$** signal, the **corresponding bit** in the **InSR** is **set** (indicating that now this interrupt is in service) and the **bit** in the **IRR** is **reset** (to indicate that the request is accepted).

8259 now **prepares** to send the Vector number **N** to the µP on the data bus.

5) The **µP sends the second $\overline{\text{INTA}}$ pulse** to 8259.

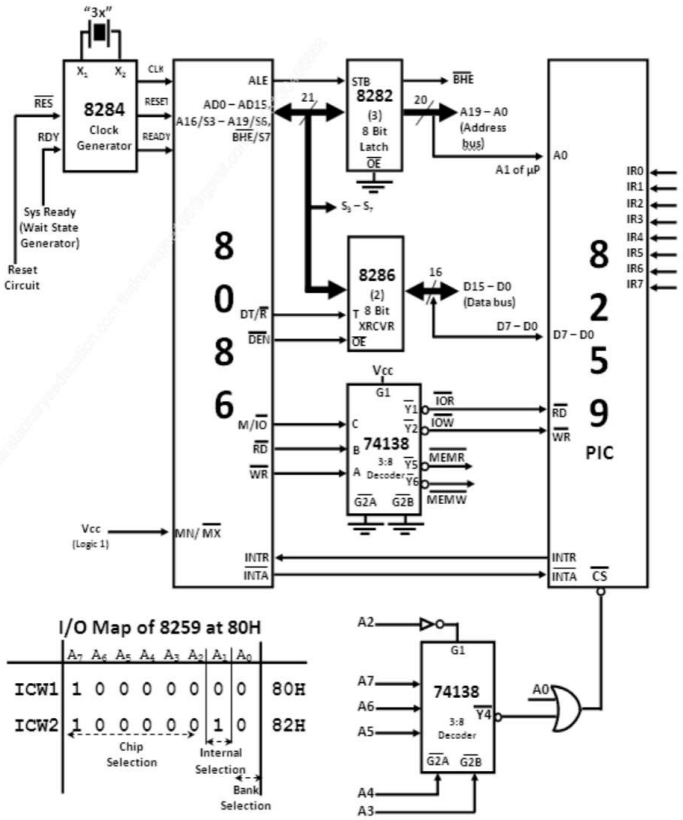
6) In response to the 2nd **$\overline{\text{INTA}}$** pulse, **8259 sends the one byte Vector Number N to µP**.

7) Now the **µP multiplies N x 4**, to get the values of CS and IP **from the IVT**.

8) **In** the **AEOI Mode** the **InSR bit is reset** at this point, **otherwise** it **remains set until** an **appropriate EOI** command is given at the End of the ISR.

9) The **µP pushes** the contents of **Flag** Register, **CS**, **IP**, into the **Stack, Clears IF and TF** and **transfers program to** the **address** of the **ISR**.. *#Please refer Bharat Sir's Lecture Notes for this ...*

10) **The ISR thus begins**.



Interfacing and Working of "CASCADED" 8259

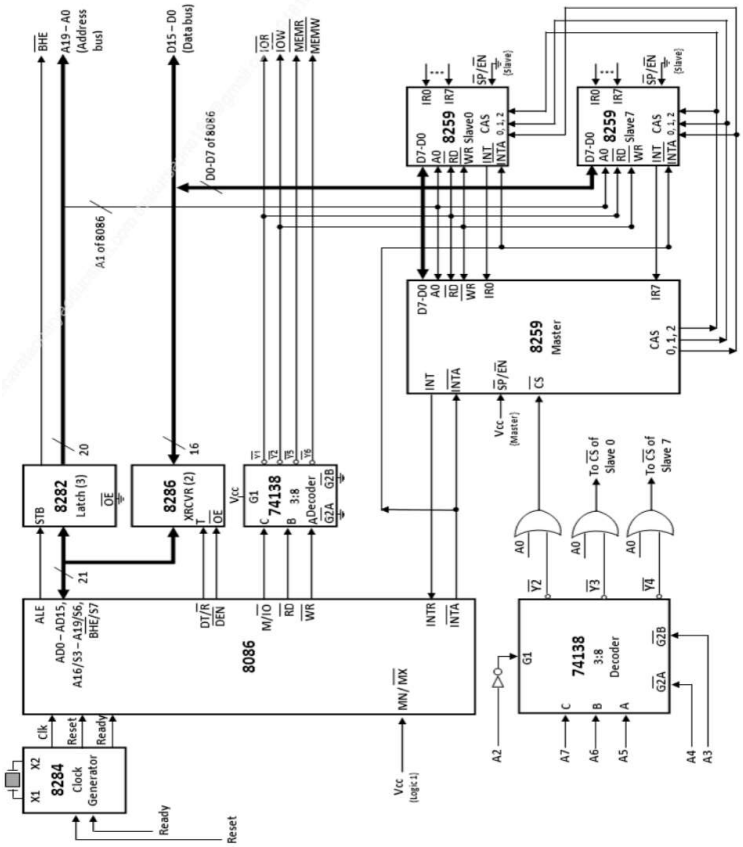
When **more than one 8259s** are connected to the µP, it is called as a **Cascaded configuration**.
A Cascaded configuration **increases** the **number of interrupts** handled by the system.
As the **maximum** number of **8259s** interfaced can be **9** (1 Master and 8 Slaves) the **Maximum** number of **interrupts** handled can be **64**.

The **master 8259** has $\overline{SP}$ / $\overline{EN}$ = **+5V** and the **slave** has $\overline{SP}$ / $\overline{EN}$ = **0V**.

Each slave's INT output is **connected** to the **IR input** of the **Master**.
The **INT** output **of** the **Master** is **connected** to the **INTR** input of the **µP**.
The **master addresses** the individual **slaves through** the **CAS_2, CAS_1, CAS_0 lines** connected from the master to each of the slaves.
First the **INTR** signal of the µP **should be enabled** using the **STI** instruction**.**
Each 8259 (Master or Slave) **has** its **own address** and **has to be initialized separately** by giving ICWs as per requirement.

When an **interrupt request** occurs **on** a **SLAVE**, the events are performed:

1) The **slave 8259 resolves** the **priority** of the interrupt and **sends** the **interrupt to** the **master** 8259.
2) The **master resolves** the **priority** among its slaves and **sends** the **interrupt to** the **µP**.
3) The **µP finishes the current instruction** and **responds** to the interrupt **by sending 2 $\overline{INTA}$ pulses**.
4) **In response to the first $\overline{INTA}$ pulse** the **following events occur**:
   i. The master **sends** the **3-bit salve identification number** on the **CAS lines**.
   ii. The **Master sets** the **corresponding bit** in **its InSR**.
   iii. The **Slave identifies** its number on the **CAS lines** and **sets** the **corresponding bit in its InSR**.
5) In **response to** the **second $\overline{INTA}$** pulse the **slave** places **Vector Number N** on the data bus.
6) **During** the **2nd $\overline{INTA}$** pulse the **InSR bit** of the **slave** is **cleared** in **AEOI mode**, **otherwise** it is **cleared by** the **EOI command** at the end of the ISR.
7) The **µP pushes** the contents of **Flag** Register, **CS**, **IP**, into the **Stack, Clears IF and TF** and **transfers program to** the **address** of the **ISR**.. *#Please refer Bharat Sir's Lecture Notes for this ...*
8) **The ISR thus begins**.



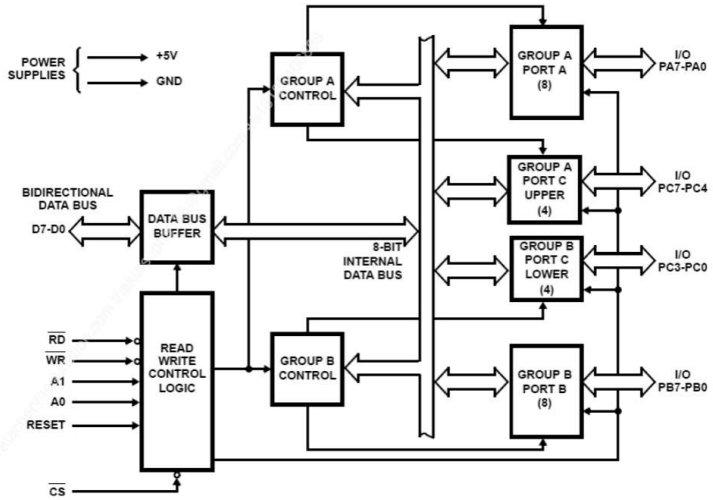
I/O map		A7	A6	A5	A4	A3	A2	A1	A0	I/O
8259	ICW1	**0**	**1**	**0**	0	0	0	**0**	0	**40**
Master	ICW2	**0**	**1**	**0**	0	0	0	**1**	0	**42**
8259	ICW1	**0**	**1**	**1**	0	0	0	**0**	0	**60**
Slave 0	ICW2	**0**	**1**	**1**	0	0	0	**1**	0	**62**
8259	ICW1	**1**	**0**	**0**	0	0	0	**0**	0	**80**
Slave 7	ICW2	**1**	**0**	**0**	0	0	0	**1**	0	**82**

8255 PPI

Salient Features

1) It is a **programmable** general-purpose **I/O** device.
2) It has 3 8-bit bi-directional I/O ports: Port A, Port B, and Port C.
3) It provides 3 modes of data transfer: Simple I/O, Handshake I/O and Bi-directional Handshake.
4) Additionally it also provides a Bit Set Reset Modes to alter individual bits of Port C.

ARCHITECTURE OF 8255



The architecture of 8255 can be divided into the following parts:

1) Data Bus Buffer
This is a 8-bit bi-directional buffer used to interface the internal data bus of 8255 with the external (system) data bus.
The CPU transfers data to and from the 8255 through this buffer.

2) Read/Write Control Logic
It accepts address and control signals from the µP.
The Control signals determine whether it is a read or a write operation and also select or reset the 8255 chip. **For doubts contact Bharat Sir on 98204 08217**
The Address bits (A_1, A_0) are used to select the Ports or the Control Word Register as shown:

For 8255 A_1 A_0	For 8086 A_2 A_1	Selection	Sample address
0 0	0 0	Port A	80 H (i.e. 1000 0**00**0)
0 1	0 1	Port B	82 H (i.e. 1000 0**01**0)
1 0	1 0	Port C	84 H (i.e. 1000 0**10**0)
1 1	1 1	Control Word	86 H (i.e. 1000 0**11**0)

The Ports are controlled by their respective Group Control Registers.

3) Group A Control
This Control block controls Port A and Port C_{Upper} i.e. PC_7-PC_4.
It accepts Control signals from the Control Word and forwards them to the respective Ports.

4) Group B Control
This Control block controls Port B and Port C_{Lower} i.e. PC_3-PC_0.
It accepts Control signals from the Control Word and forwards them to the respective Ports.

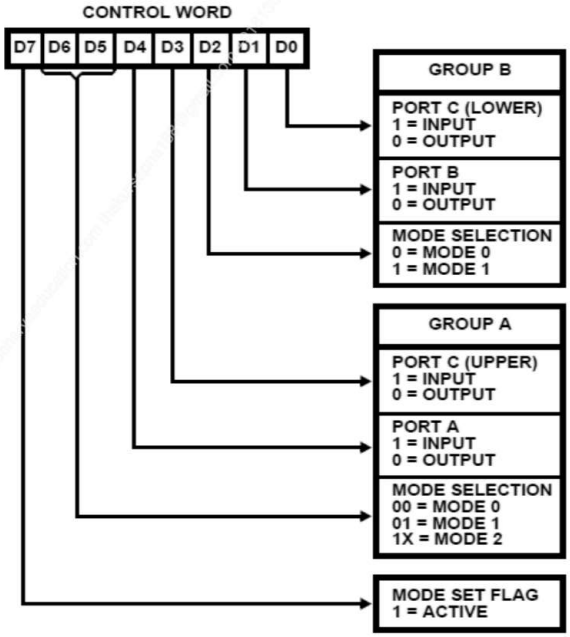
5) Port A, Port B, Port C
These are 8-bit Bi-directional Ports.
They can be programmed to wok in the various modes as follows:

Port	Mode 0	Mode 1	Mode 2
Port A	Yes	Yes	Yes
Port B	Yes	Yes	**No** *(Mode 0 or Mode 1)*
Port C	Yes	**No** *(Handshake signals)*	**No** *(Handshake signals)*

ONLY Port C can also be programmed to work in Bit Set reset Mode to manipulate its individual bits.

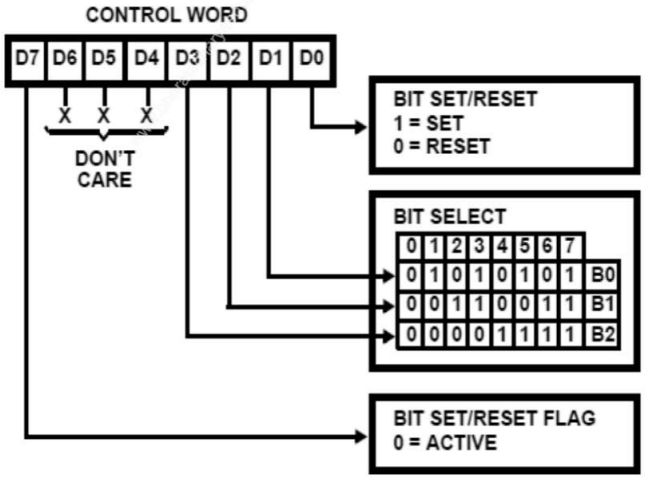
1)Control Word of 8255 - I/O Mode (I/O Command)

To do 8-bit data transfer using the Ports A, B or C, 8255 needs to be in the IO mode.
The bit pattern for the control word in the IO mode is as follows:



2)Control Word of 8255 - BSR Mode (BSR Command) { ONLY for Port C}

- The BSR Mode is used **ONLY for Port C.**
- In this Mode the **individual bits** of Port C can be **set or reset**.
- This is very useful as it provides **8 individually controllable lines** which can be used while interfacing with devices like an **A to D Converter** or a 7-segment display etc.
- The individual bit is **selected** and Set/reset through the **control word**.
- Since the D7 bit of the Control Word is 0, the BSR operation **will not affect the I/O operations** of 8255. **For doubts contact Bharat Sir on 98204 08217**



DATA TRANSFER MODES OF 8255

❖ Mode 0 (Simple Bi-directional I/O)

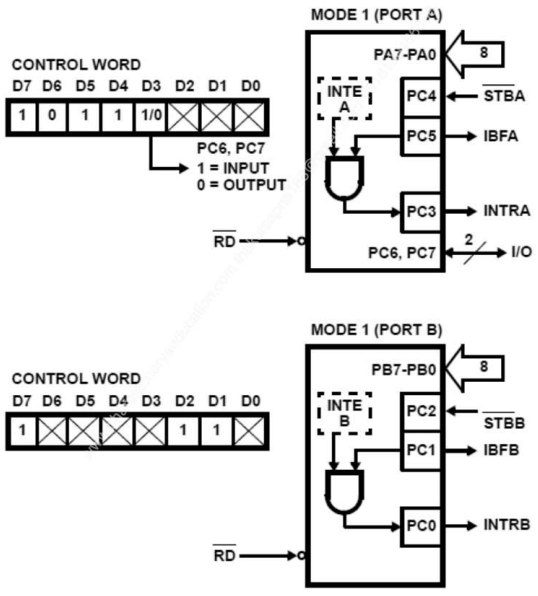
Port A and Port B used as 2 Simple 8-bit I/O Ports.
Port C is used as 2 simple 4-bit I/O Ports.
Each port can be programmed as input or output individually.
Ports do not have handshake or interrupting capability.
Hence, **slower** devices cannot be interfaced.

❖ Mode 1 (Handshake I/O)

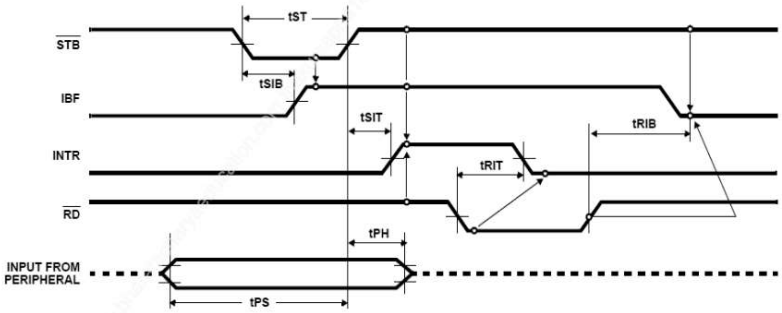
In Mode 1, handshake signals are exchanged between the devices before the data transfer takes place.
Port A and Port B used as 2 8-bit I/O Ports that can programmed in Input OR in output mode.
Each Port uses 3 lines from Port C for handshake. The remaining lines of Port C can be used for simple IO.
Interrupt driven data transfer and **Status driven** data transfer possible.
Hence, **slower** devices can be interfaced.

The handshake signals are different for input and output modes.
#Please refer Bharat Sir's Lecture Notes for this ...

♦ Mode 1 (Input Handshaking)



Timing Diagram for Mode 1 Input Transfer



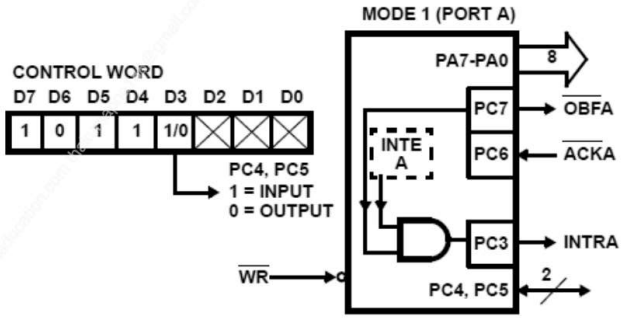
Working:

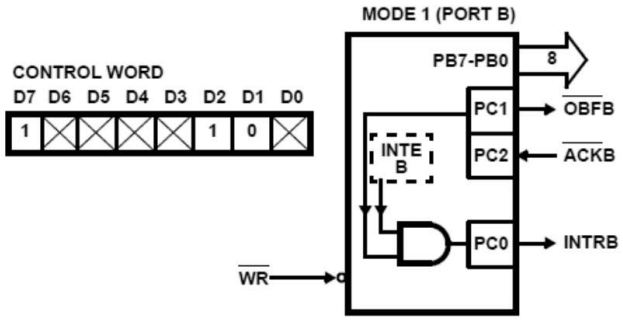
Each port uses **3 lines** of **Port C** for the following signals:
STB (Strobe), **IBF** (Input Buffer Full) ➔ Handhsake signals
INTR (interrupt) ➔ Interrupt signal
Additionally the $\overline{RD}$ signal of 8255 is also used.

Handshaking takes place in the following manner:

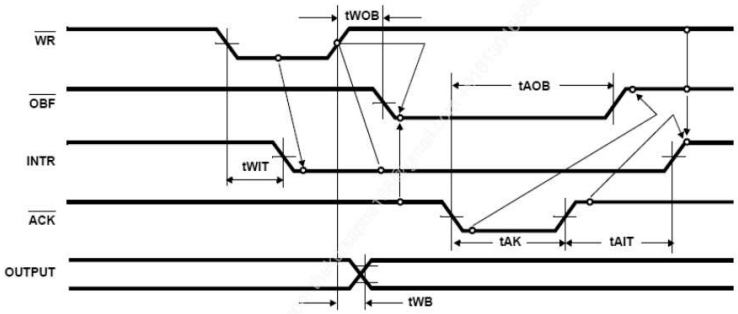
1) The **peripheral** device **places data** on the Port **bus** and informs the Port by **making STB low**.
2) The **input Port accepts** the **data** and informs the peripheral to wait by making **IBF high**. This **prevents** the peripheral from **sending more data** to the 8255 and **hence data loss** is prevented. ☺ In case of doubts, contact Bharat Sir: - 98204 08217.
3) **8255 interrupts** the **µP** through the **INTR** line provided the INTE flip-flop is set.
4) **In response** to the Interrupt, the **µP issues** the $\overline{RD}$ signal and **reads** the **data**. The **data byte** is **thus transferred** to the **µP**.
5) Now, the **IBF** signal **goes low** and the peripheral can **send more data** in the above sequence.

♦ Mode 1 (Output Handshaking)





Timing Diagram for Mode 1 Output Transfer



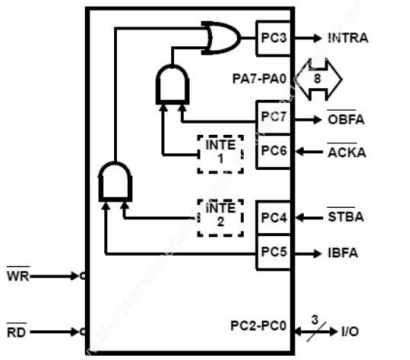
Working

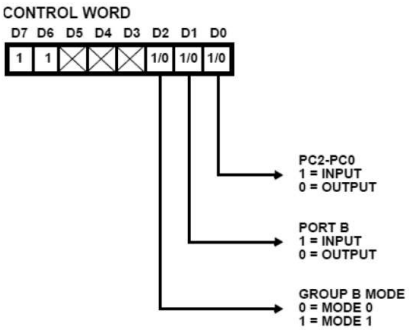
Each port uses **3 lines** of **Port C** for the following signals:
OBF (Output Buffer Full), **ACK** (Acknowledgement) ➔ Handhsake signals

INTR (interrupt) ➔ Interrupt signal. Additionally the **WR** signal of 8255 is also used. **Handshaking** takes place in the following manner:

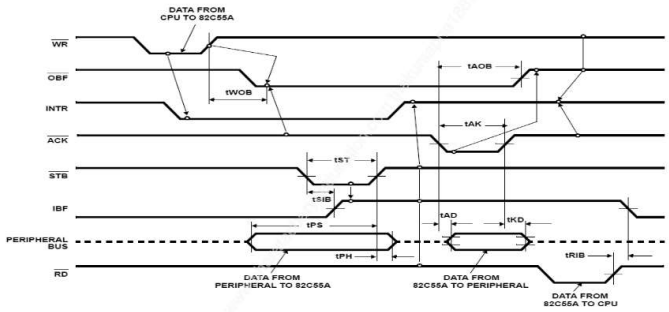
1) When the output port is empty (indicated by a high on the INTR line), the **µP writes data** on the output port by giving the **WR** signal.
2) As soon as the **WR** operation is complete, the **8255 makes the INTR low**, indicating that the µP should **wait**. This **prevents** the µP from **sending more data** to the 8255 and **hence data loss** is prevented.
3) **8255** also **makes** the **OBF low** to indicate to the output peripheral that **data** is **available** on the data bus.
4) The **peripheral accepts** the **data** and sends an acknowledgement by making the **ACK low**. The **data byte** is **thus transferred** to the peripheral.
5) Now, the **OBF** and **ACK** lines **go high**.
6) The **INTR** line **becomes high** to **inform** the **µP** that **another byte** can be **sent.** i.e. the output port is empty.
   This process is repeated for further bytes.

❖ Mode 2 (Bi-directional Handshake I/O)





Timing Diagram for Mode 2 Bi-Directional Transfer



Working:

In this mode, **Port A** is **used** as an **8-bit bi-directional Handshake I/O Port**.
Port A requires **5 signals from Port C** for doing Bi-directional handshake.
Port B has the following **options**:

1) **Use** the **remaining 3 lines** of **Port C** for handshaking so that **Port B is in Mode 1**. Here **Port C** lines will be **completely used for handshaking** (5 by Port A and 3 by Port B).
   OR
2) **Port B** works in **Mode 0** as simple I/O.
   In this case the **remaining 3 lines** of **Port C** can be used for **data transfer**.

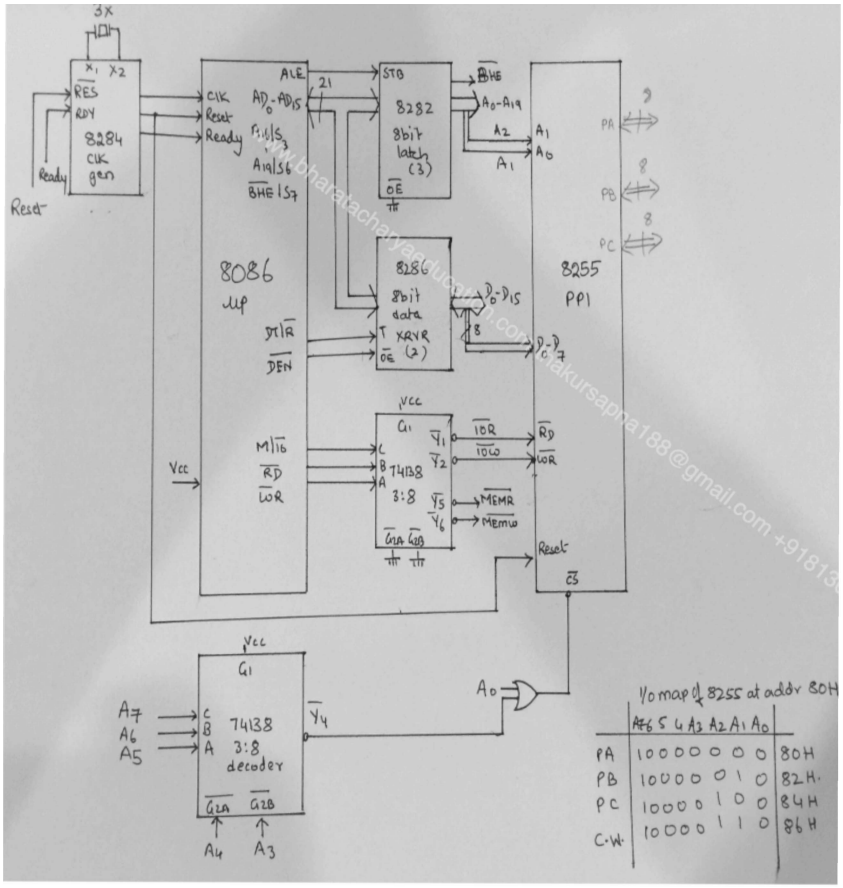
Port A can be used for data transfer between two computers as shown.
The high-speed computer is known as the master and the dedicated computer is known as the slave.
Handshaking process is similar to Mode 1.

For **Input**:
STB and **IBF** → handshaking signals, **INTR** → Interrupt signal.
For **Output**:
$\overline{\textbf{OBF}}$ and $\overline{\textbf{ACK}}$ → handshaking signals, **INTR** → Interrupt signal.

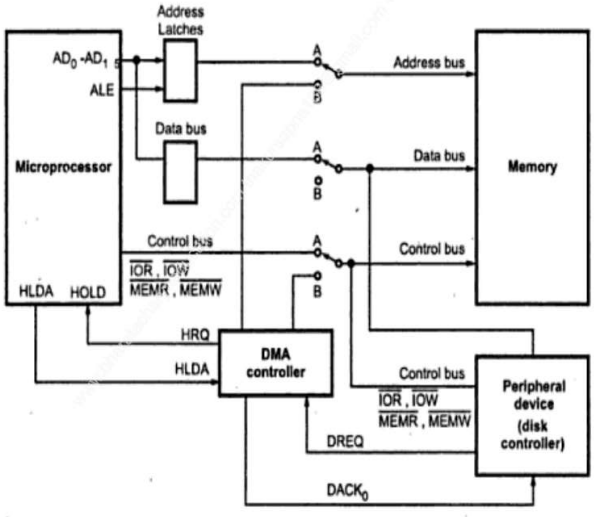
Thus the 5 signals used from Port C are:
STB, IBF, INTR, $\overline{\textbf{OBF}}$ and $\overline{\textbf{ACK}}$.

INTERFACING OF 8255 WITH 8086

1) 8255 is a **programmable peripheral interface**.
   It is used to interface microprocessor with I/O devices via three ports: PA, PB, PC.
   All ports are 8-bit and bidirectional.
2) 8255 transfers data with the microprocessor through its **8-bit data bus**.
3) The **two address lines** A1 and A0 are used to make **internal selection** in 8255.
   They can have 4 options, selecting PA, PB, PC or the control word.
   The ports are selected to transfer data.
   The Control word is selected to send commands.
4) **Two commands** can be sent to 8255, called the I/O command and the BSR command.
   I/O command is used to initialize the **mode and direction** of the ports.
   BSR command is used to **set or reset a single line** of Port C.
5) 8255 has **three operational modes** of data transfer.
6) **Mode 0** is a **simple data transfer** mode.
   It does not perform handshaking but all three ports are available for data transfer.
7) **Mode 1** performs **unidirectional handshaking**.
   That makes transfers more reliable.
   Port C lines are used by Port A and Port B to perform Handshaking.
8) **Mode 2** performs **bidirectional handshaking**.
   Only Port A can operate in Mode 2.
   At that time Port B can operate in Mode 1 or Mode 0.
   Port C lines are again used up for performing Handshaking for Port A and Port B.



CONCEPT OF DMA



DMA Transfer is a **hardware controlled I/O** Transfer technique.
It is mainly used for **high-speed data transfer between I/O and Memory** where the speed of the peripheral is generally faster than the µP. For doubts contact Bharat Sir on 98204 08217
In Program Controlled I/O, Status or interrupt driven I/O the speed of transfer is **slow** mainly **because instructions** need to be **decoded** and **then executed** for the transfer.
DMA transfer is **software independent** and **hence much faster**.
A device known as the DMA Controller (DMAC) is responsible for the DMA transfer.

The **sequence of DMA transfer** is as follows:

1) µP initializes the DMAC by giving the starting address and the number of bytes to be transferred.
2) An **I/O device requests** the **DMAC**, to perform DMA transfer, **through the DREQ line**.
3) The **DMAC** in turn sends a **request signal to** the **µP**, through the **HOLD line**.
4) The **µP finishes the current machine cycle** and **releases** the **system bus** (gets disconnected from it).
   It also **acknowledges** receiving the HOLD signal through the **HLDA line**.
5) The **DMAC acquires control of the system bus.**
   The **DMAC sends** the **DACK signal** to the I/O peripheral and the **DMA transfer begins**.
6) After every byte is transferred, the Address Reg is incremented (or decremented) and the Count Reg is decremented.
7) This continues till the Count reaches zero (Terminal Count). Now the DMA transfer is completed.
8) **At the end** of the transfer, **the system bus is released by the DMAC** by making HOLD = 0.
   Thus **µP takes control of the system bus** and continues its operation.

The DMA Controller (DMAC) does DMA transfer through its channels.
The minimum requirements of each channel are:

i. **Address Register** (to store the starting address for the transfer).
ii. **Count Register** (to store the number of bytes to be transferred).
iii. **A DREQ signal** from the IO device.
iv. **A DACK signal** to the IO device.

IMPORTANT POINTS TO REMEMBER FOR I/O DESIGNING

- **Normally** I/O devices are mapped using **I/O mapped I/O** which means I/O devices are given I/O addresses
- Here **I/O addresses** can be either **8-bit or 16 bit**.
- If the question says **direct addressing mode** or **fixed port addressing**,
  Then use an **8-bit address like 80H (A7-A0)**.
- If the question says **indirect addressing** or **variable port addressing**,
  Then use **16-bit address like 8000H (A15-A0)**.
- If nothing is mentioned, use any of the above techniques.
- If **memory mapped I/O** is asked (Very rare), then remember the **following changes**
  Give the I/O device a **20-bit unused memory address like 80000H (A19-A0)**
  Connect **MEMR# and MEMW#** signals to the I/O device instead of the usual IOR# and IOW# signals

Differentiate between

	I/O MAPPED I/O	MEMORY MAPPED I/O
1	I/O device is **treated as an I/O device** and hence **given an I/O address**.	I/O device is **treated like a memory device** and hence **given a memory address**.
2	I/O device has an **8 or 16 bit I/O address**.	I/O device has a **20 bit Memory address.**
3	I/O device is given **IOR# and IOW#** control signals	I/O device is given **MEMR# and MEMW#** control signals
4	**Decoding is easier** due to lesser address lines	**Decoding is more complex** due to more address lines
5	Decoding is **cheaper**	Decoding is more **expensive**
6	Works **faster due to less delays**	More gates add more delays hence **slower**
7	Allows **max 2^{16} = 65536 I/O devices**	Allows **many more I/O devices** as I/O addresses are now 20 bits.
8	I/O devices can **only** be accessed by **IN and OUT** instructions.	I/O devices can now be accessed using **any memory instruction**.
9	ONLY **AL/ AH/ AX registers** can be used to transfer data with the I/O device.	**Any register** can be used to transfer data with the I/O device.
10	**Popular** technique in **Microprocessors**.	**Popular** technique in **Microcontrollers**.

8257 DMAC | PIN DIAGRAM

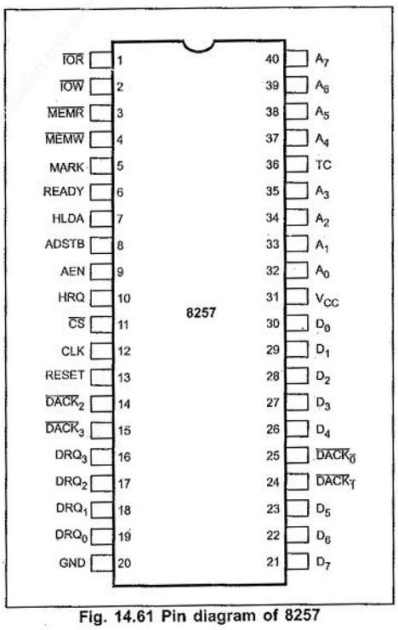


Fig. 14.61 Pin diagram of 8257

1) DREQ3 ... DREQ0 (Data Request lines)

These input pins are used by the I/O device to request the DMAC for DMA Transfer.
These pins can be made active high / active low through program.
By default, they are active high.
There are 4 DREQ pins as there are 4 channels (one per channel).
At a time only one channel can perform DMA transfer.
Thus, for multiple simultaneous requests, priorities are used.
DREQ0 has the highest priority while DREQ3 has the lowest priority. (Fixed priority mode)

2) $\overline{DACK_3}$ ... $\overline{DACK_0}$ (Data Acknowledgement)

These output pins are used by the DMAC to inform the I/O device that it is performing a data transfer.
This pin becomes low just before a DMA data transfer.
There are 4 DACK pins as there are 4 channels (one per channel).

3) HRQ (Hold Request)

This output pin is used by the DMAC to request the µP to release the system bus.
It is connected to the HOLD pin of the µP.

4) HLDA (Hold Acknowledge)

This input pin is used by the µP to inform the DMAC that it as released to system bus.
It is connected to the HLDA pin of the µP.

5) AEN (Address Enable)

This is an output pin from the DMAC.

When AEN = 0 (default), μP is the Bus master.
At that time the latches of the μP are enabled and the DMAC latch is disabled.

When AEN = 1, DMAC is the Bus master.
At that time the latches of the μP are disabled and the DMAC latch enabled.
This connects the DMAC's address-data lines to the system bus.

6) ADSTB (Address Strobe)

It is an output signal.
To put it simply this is the ALE issued by the DMAC.

When ADSTB = 1, the multiplexed bus (DB0-DB7) carries address (A8-A15).
When ADSTB = 0, the multiplexed bus (DB0-DB7) carries data (D0-D7).

7) DB7 – DB0 (Data Bus)

These are 8 bi-directional data lines used to connect the internal data bus of 8257 with the external (system) data bus.
When μP is the bus master, this bus is used by the μP to read/write from the DMAC.
In active cycle (which means DMAC is the bus master), this bus carries the 8 higher order bits of the 16 bit address (the other 8 bits being in the A7–A0).
During memory-to-memory transfer, this bus carries the data byte to be transferred.

8) A7 – A4 (Address bits)

These are 4 output address lines.
In active cycle, these lines carry the A7–A4 bits of the address at which the transfer is to be done. As this address is generated by the 8257, these are output lines.

9) A3 – A0 (Address bits)

These are 4 bi-directional address lines.
In idle cycle, µP sends the address A3–A0, to select one of its registers.
Since µP sends the address to 8257, these are input lines.
In active cycle, these lines carry the A3–A0 bits of the address at which the transfer is to be done. As this address is generated by the 8257, these are output lines.

10) $\overline{\text{IOR}}$, $\overline{\text{IOW}}$

These are bi-directional control lines.
During idle state, the µP issues these signals to read from or write into the 8257 (as the DMAC itself is an I/O device w.r.t. the µP).
During active state, the DMAC issues these signals to read from or write into an I/O Device.

11) $\overline{\text{MEMR}}$, $\overline{\text{MEMW}}$

These are output control lines.
During active state, the DMAC issues these signals to read from or write into the Memory.

12) Ready

Similar function as Ready pin of the μP.
It is used to synch DMAC with slower devices when DMAC is the Bus Master.
DMAC checks this signal during every DMA transfer cycle.

If Ready = 1, it means I/O device is ready.
Hence DMAC will continue with the transfer.

If Ready = 0, it means I/O device is not ready.
Hence DMAC will insert Wait states during the transfer to allow the device to become ready.

13)CLK

This is a clock-input signal for the DMAC.
It is usually connected to the system clock.

14)RESET

This is a reset-input signal for the DMAC.
This signal clears the internal registers of the DMAC and causes it to enter Idle State.

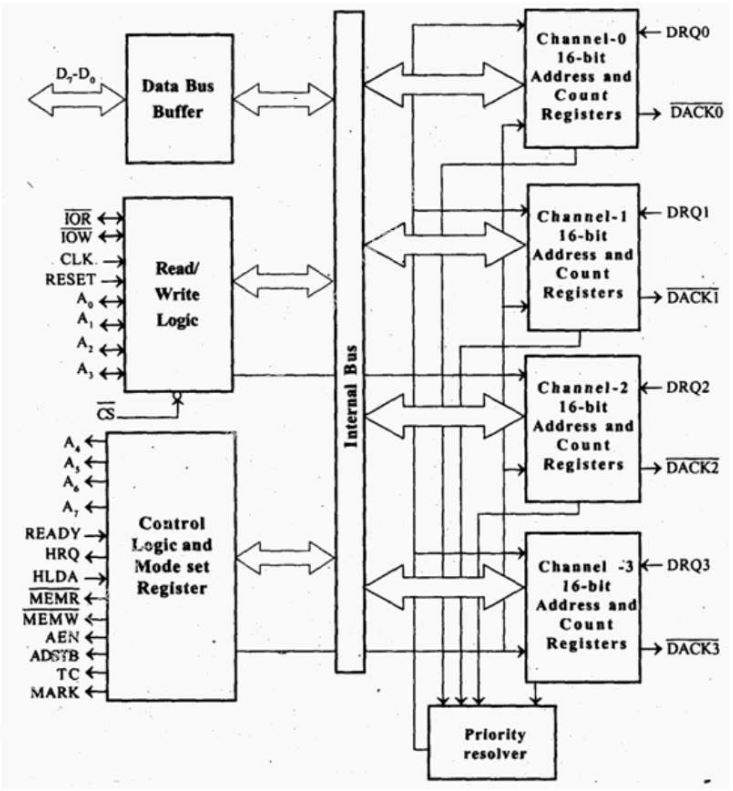
15) TC

This is an output signal from the DMAC.
It becomes 1 to indicate that terminal count is reached, that means the count has become 0 and hence the transfer has been completed.

16) Mark

This is an output signal from the DMAC.
It is a modulo 128 mark output. This means it becomes 1 on every 128th cycle of the data transfer as a marker to indicate that a batch of 128 cycles has been completed.

8257 DMAC | ARCHITECTURE



The internal architecture of 8257 has the following components

1) DMA Channels (0…3)
2) Priority Resolver
3) Data Bus Buffer
4) Read Write Logic
5) Control Logic and Mode Set Register

1) DMA Channels (0…3)

A single 8257 DMAC has 4 DMA Channels (0…3)
Four I/O devices are connected on these DMA Channels, one on each.
In the default priority mode (Fixed priority), Channel 0 is the highest and 3 is the lowest priority.

Each Channel has 4 components:
Address Register, **Terminal Count Register**, **DREQ** and **$\overline{\text{DACK}}$** .

a) Address Register (16 bit)

It is used to store the 16 bit memory address for the DMA Transfer.
μP initializes this register with the starting address of the DMA Transfer.
Thereafter, as each byte is transferred the address gets incremented
(or decremented, depending upon the mode selected by the programmer)

b) Terminal Count Register (16-bit)

It is used to store the 14bit count of the DMA Transfer.
The remaining higher two bits are used to decide the mode of DMA operation.
μP initializes this register with the 14 bit count (N-1) of the DMA Transfer.
Thereafter, as each byte is transferred the count gets decremented.
This repeats till the count becomes 0 also called Terminal Count (TC).

The higher two bits are used to select the mode.
They can be used to give 3 different modes: DMA Verify, DMA Read, DMA Write

DMA Read

In this mode, when DMAC becomes the bus master, it transfers data from
Memory to I/O. Hence in every transfer, the signals produced are **$\overline{\text{MEMR}}$ and $\overline{\text{IOW}}$**

Count Register: 16 bit (D15...D0)

D7	D6	D5	D4	D3	D2	D1	D0

← Lower 8 bits of the 14 bit count →

D15	D14	D13	D12	D11	D10	D9	D8

← Mode bits (2) → ← Higher 6 bits of the 14 bit count →

Mode Bits		Mode Selected
0	0	DMA Verify Cycle
0	1	DMA Write Cycle (I/O to Memory)
1	0	DMA Read Cycle (Memory to I/O)
1	1	No action

DMA Write

In this mode, when DMAC becomes the bus master, it transfers data from I/O to Memory. Hence in every transfer, the signals produced are **$\overline{\text{IOR}}$ and $\overline{\text{MEMW}}$**

DMA Verify

In this mode, when DMAC becomes the bus master, it doesn't really transfer any data. This mode is just used to verify the DMA process.
The DMAC will issue a HOLD, will become bus master, issue the acknowledgement to the I/O device and so on. It just will not produce any read or write signals to perform any data transfer. If you are learning by Piracy, you are a THIEF!

c) DREQ

I/O device gives this signal to the DMAC to request a DMA transfer

d) $\overline{\text{DACK}}$

It is given by DMAC to the I/O device, indicating that a DMA transfer is being performed.

2) Priority Resolver

Priority is needed when several DMA channels get request (DREQ) from I/O devices "simultaneously" for data transfer. Priority resolver decides which channel will be "serviced" first, and which one will become "pending". There are two priority schemes: Fixed Priority and Rotating Priority.

Fixed Priority

This is the Default Mode.
Channel 0 is the highest priority and Channel 3 is the lowest.
Fixed priority causes Domination.
If Channel 0 and 1 keep requesting all the time the Channel 2 and 3 will starve and never get a chance. This is called Domination.
To avoid this, we can use Rotating Priority.

Rotating Priority

Here, once a channel is serviced it becomes the lowest priority.
All channels below it rise up by one position in the priority order.
As priorities move in a circular manner, it is called Rotating Priority.
It gives every channel a fair chance of being high priority and hence prevents Domination.

Before CH.0 is serviced

Highest	CH.0
	CH.1
	CH.2
Lowest	CH.3

← Active DMA request

After CH.0 is serviced

CH.1	Highest
CH.2	
CH.3	
CH.0	Lowest

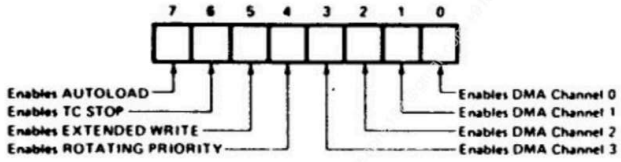
3) Read Write Logic

It mainly provides the read and write signals as well as the chip select signal.
The read and write signals are connected to **$\overline{\text{IOR}}$, $\overline{\text{IOW}}$** of the μP.
It also has address lines A3…A0 used for internal selection of components as explained earlier in the pin descriptions.

4) Control Logic and Mode Set Register

It generates the internal control signals for the DMAC.
It also provides external signals such as HRQ (Hold), HLDA, AEN, ADSTB, TC, MARK etc.
as explained earlier. If you are learning by Piracy, you are a THIEF!
Additionally, it has the Mode Set register.

Mode Set Register



Bit 0…3: Channel Enables

1: Enable the respective DMA channel
0: Disable

Bit 4: Rotating Priority

1: Rotating Priority
0: Fixed Priority

Bit 5: Extended Write

1: Extended Write
0: Normal Write

Extended Write Mode

Here the Write control signal gets activated one T-State in advance.
This is similar to the Advanced write signals of 8086 Maximum Mode.
Once the Write signal gets activated (goes low), the I/O device has to respond by Making Ready Signal=1 to indicate that it is ready for the transfer. A slow device may require additional time to get ready and hence this will force the DMAC to insert Wait states in between every cycle, thus making the whole operation slow.
To avoid this, we use the extended write mode.
Here, the Write signal goes low one T-state in advance hence the I/O gets a little extra time to become ready. This reduces the chances of inserting Wait states and hence prevents a slightly slow I/O device from making the whole DMA operation slow.

Bit 6: TC Stop

1: Enable TC Stop mode
0: Disable

TC Stop Mode

In this mode, the DMA operation stops once terminal count is reached.
The Respective DMA channel enable bit automatically becomes 0.

Bit 7: Auto load

1: Enable Auto Load Mode
0: Disable

Auto Load Mode

This is a continuous self-reloading mode.
It is applicable only for Channel 2.
When this mode is selected the original count and address register values of Channel 2 are stored as a back up in Channel 3 registers.
After every byte is transferred, Channel 2 registers keep changing but Channel 3 registers maintain the original values.
When Channel 2 reaches TC, There is an Automatic reload of address and count information from Channel 3 registers to Channel 2 registers and the DMA transfer restarts. This mode is basically used to perform repetitive DMA transfers.

5) Data Bus Buffer

It connects the external data bus of the system with the internal data bus of the DMAC, when the DMAC Chip is selected. If you are learning by Piracy, you are a THIEF!