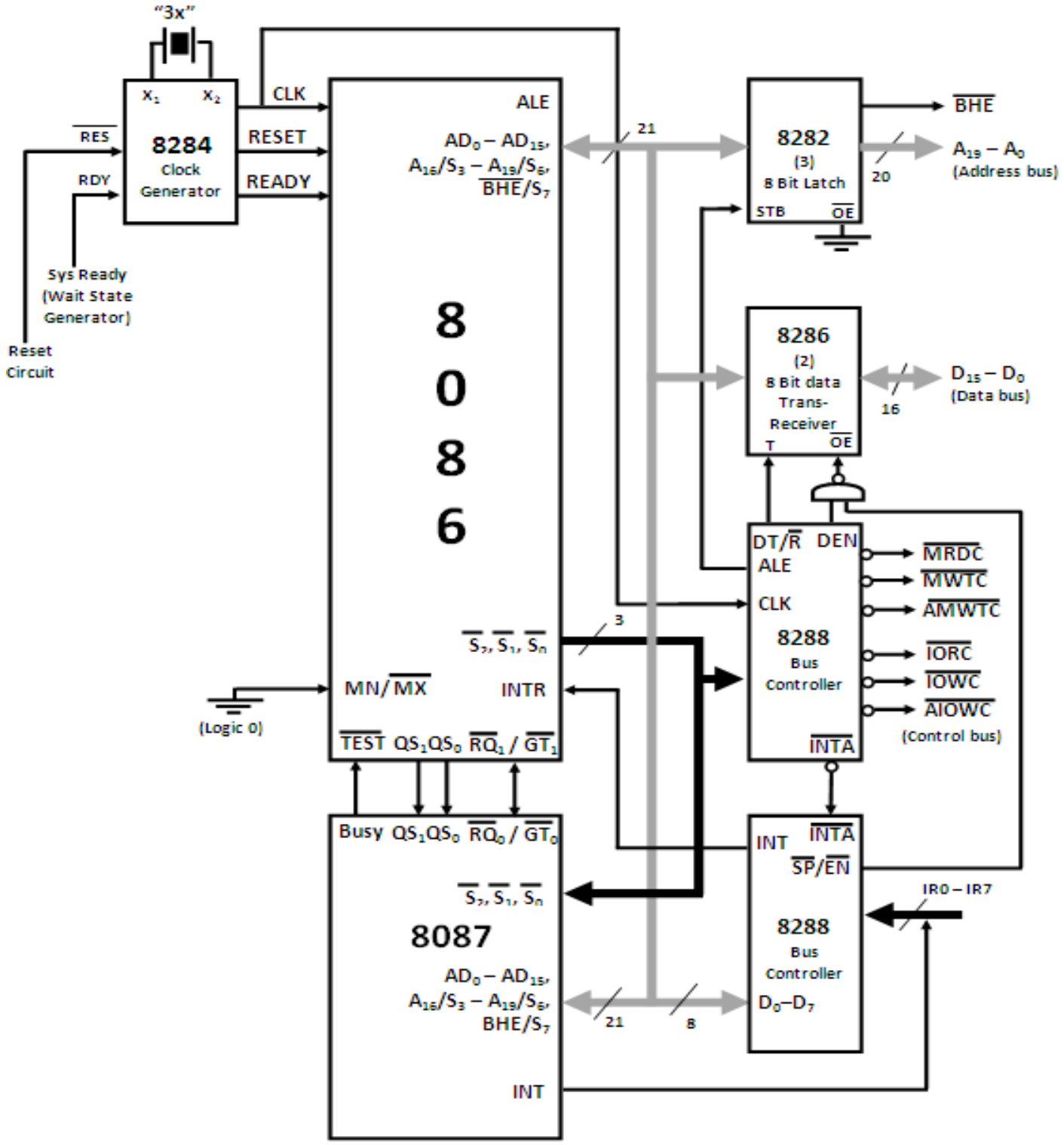
CO-PROCESSOR CONFIGURATION --- 8086 WITH 8087



1) As a co-processor (8087) is connected to 8086, 8086 operates in **Maximum Mode**.
   Hence **MN/ $\overline{\text{MX}}$ 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 **$\overline{\text{TEST}}$** of μP.
   Now **8087** is **executing its instruction** and **8086** moves ahead with **its next instruction**.
   Hence ***MULTIPROCESSING*** *takes place.* **For doubts contact #BharatSir @9820408217**

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 **$\overline{\text{RQ}}$ / $\overline{\text{GT}}$** of 8087 is connected to **$\overline{\text{RQ}}$ / $\overline{\text{GT}}$** 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 **$\overline{\text{TEST}}$** pin.
    If the **TEST** pin is **high** (**8087** is **BUSY**), then the **μP enters WAIT state**
    It **comes out** of it only **when** **$\overline{\text{TEST}}$** 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)