

Generation

First

Generation

Year

b/w

1971 & 1973

Technology

PMOS

P-type metallic
oxide SemiconductorProcessor

4-bit

Second

1973 -

1978

NMOS

8-bit

N-type metallic
oxide Semiconductor

Third

1978 -

1980

HMOS

16-bit

Hybrid / high
density metallic
oxide Semiconductor

Fourth

1980 -

HCMOS

32-bit

High Speed
Complementary metal
oxide semiconductor

Fifth

1993

onwards

64-bit

Additional Points→ First Generation

- ✓ PMOS technology provided
 - * low cost
 - * slow speed
 - * low output current

✓ Require as high as 30 ICs to form a system.

✓ I generation processors → 4-bit ; 16 pins.

8-bit & 16-bit ; 40 pins.

eg: Intel Corporation introduced 4004, the first microprocessor in 1971.

i.e. INTEL 4004 } 4-bit processors
 INTEL 4040 etc...

INTEL 8008 } 8-bit processors
 MOSTEK 6065 etc...

NATIONAL IMP 16 } 16-bit processors
 NATIONAL PACE

SECOND GENERATION

- ✓ Marked the beginning of very efficient 8-bit microprocessors.
- ✓ NMOS technology offers faster speed and higher density than PMOS.
- ✓ 40 pins
- ✓ More no. of on-chip decoded timing sigs.
- ✓ Ability to address large memory spaces.
- ✓ Ability to address more I/O ports.
- ✓ Faster operation
- ✓ More powerful instruction set.
- ✓ Better interrupt handling capabilities.

eg: INTEL 8080 ZILOG Z80 } 8-bit
 INTEL 8085

Mar Baselios College of Engineering & Technology

0.5 MIPS (execute 0.5 million instructions per second)

INTERSIL 6100

TOSHIBA TLCS-12

TI TMS 9900

General Instrument CP 1600

} 12-bit processor

} 16-bit processor

* Third Generation (1978 - 1980)

- ✓ This age is dominated by 16-bits microprocessors
- ✓ Some of them were:

INTEL's 8086 / 80186 / 80286

MOTOROLA 68000 / 68010

- ✓ Provided with 40 / 48 / 64 pins.
- ✓ High Speed and very strong processing capability
- ✓ Easier to program.
- ✓ Size of internal registers are 8 / 16 / 32 bits.
- ✓ Physical memory space is from 1 to 16 Megabytes
- ✓ flexible I/O port addressing.
- ✓ Different modes of operations.

* Fourth Generation (1980 - 1995)

- ✓ This era marked the beginning of 32-bit microprocessors.
- ✓ Using low-power version of the NMOS technology called HCMOS.

- * (2^{24}) 16 Mb physical memory space.
- * (2^{40}) 1 Tb virtual memory space.
- * Floating point hardware is incorporated.
- * Supported increase no. of addressing modes.

eg: INTEL 80386 / 80486 → **54 MIPS**
 (execute 54 Million instructions per second)

Motorola MC88100 / M68020 / M68030

FIFTH Generation

→ This age the emphasis is on introducing chips that carry on-chip functionalities an improvement in the speed of memory and I/O devices, along with introduction of "64-bit microprocessors".

→ Intel leads the show here with Pentium, Celeron and very recently dual & quad core processors working with upto 3.5 GHz speed. A pentium processor can execute 2000 MIPS.

Question: Explain the evolution of microprocessors?

Reference

Microprocessors and microcontrollers, 2e.
 A. Nagoor Kani Page No: 103 to 1.5

Title: Comparison of Microprocessor Vs Microcontroller

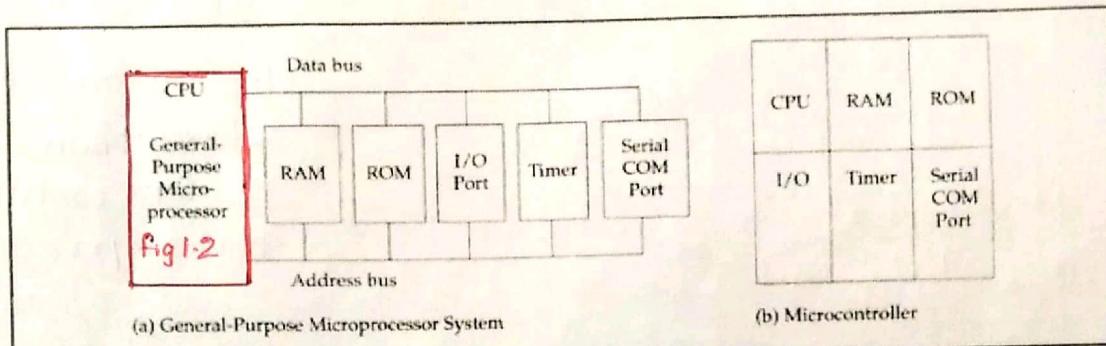


Figure 1-1. Microprocessor System Contrasted With Microcontroller System

Microprocessor

- * Functional blocks
→ ALU, Registers & timing & control unit.
- * Large no. of instructions for moving data b/w the external m/y and the microprocessor
- * Few no. of bit manipulating instructions.
- * Require interfacing of a large no. of additional ICs to form a microcomputer based system.
- * Used for designing general purpose digital computing system.

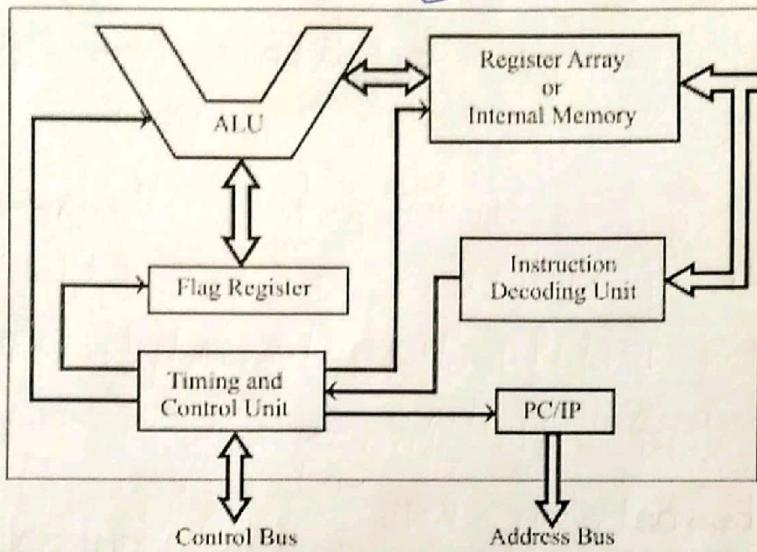
Microcontroller

- * Functional blocks
↳ microprocessor, timer, parallel & serial IO port, Internal RAM, EEPROM/EEPROM memory & ADC or DAC converter.
- * Few instructions for data transfer.
- * large no. of bit manipulating instructions.
- * Microcontroller can be used to form a single chip microcomputer based system without any additional ICs.
- * Used for designing application specific dedicated systems.

For any microprocessor, there will be a set of instructions given by the manufacturer of the processor. For doing any useful work with the microprocessor

we have to write a program using

these instructions and store them in a memory device external to the microprocessor.



J. 1.2: Block diagram showing basic functional blocks of a microprocessor.

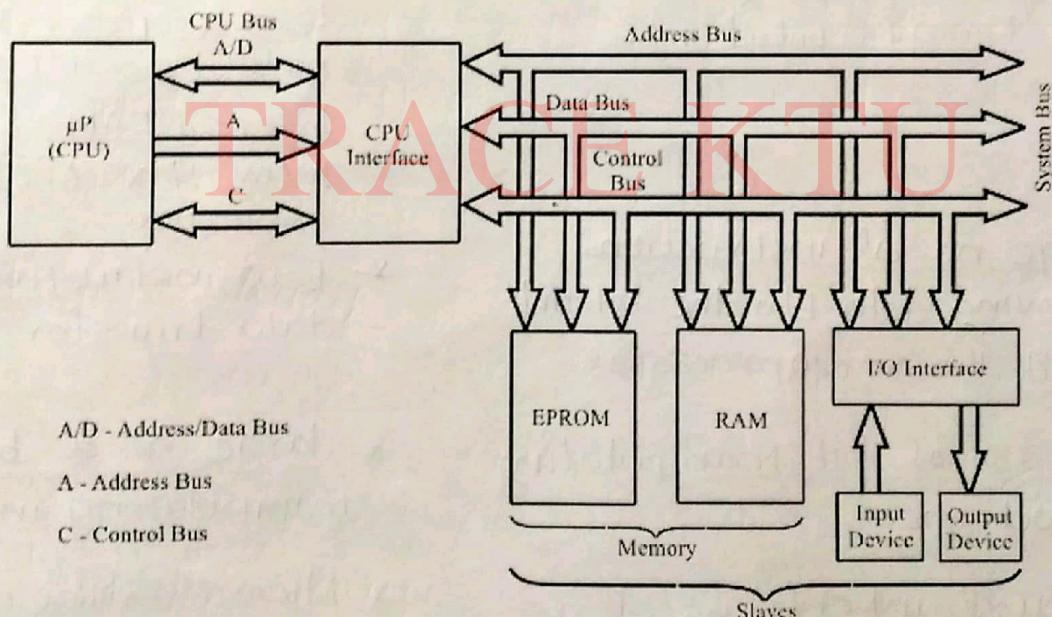


Fig. 1.3: Microprocessor-based system (organization of microcomputer).

EPROM: Store permanent pgms and data.

RAM : Store temporary pgms and data.

Input device: Used to enter the pgms, data and to operate the sm.

Output device: Used for examining the results.

Mar Baselios College of Engineering & Technology

CPU Interface: The speed of IO devices doesn't match with the speed of the microprocessor since this is provided b/w the sm bus of the IO devices.

Page #...6

Explanation of fig 1.2 Basic functional blocks of a Microprocessor

Programmable IC which is capable of performing arithmetic and logical operations.

Basic functional Units

1) ALU

Computational unit of the microprocessor, which performs arithmetic and logical operations on binary data.

2) Flag Register

Various conditions of the result, are stored as status bits called flags in the flag register. from ALU

eg: If the result is '-ve' then '1' is stored in the sign flag and if the result is '+ve', then 0 is stored in the sign flag.

3) Register Array (internal storage device)

- * Input data for ALU
 - * Output data of ALU
 - * Any other binary info needed for processing
- } are stored in the register array.

4) PC / IP (Program Counter / Instruction pointer)

Generates the "address of the instructions to be fetched from the memory" and send that address through the address bus to the memory.

5) Instruction Decoding Unit

The m/y will send the instruction codes and data through the data bus.

⇒ Instruction codes are decoded by the decoding unit and indicate that the ALU "what operation is to be performed".

6) Timing and Control Unit

Generate the necessary control signals for the internal and external operations of the microprocessor.

e.g: m/y read signal, m/y write signal
I/O read signal, I/O write signal

Explanation fig 1.3 (Microprocessor-based s/m)

In this s/m, the "microprocessor is the master" and all "other peripherals are slaves."

Mastee controls all the peripherals (memories, input device, o/p device and interfacing devices) and initiates all operations.

⇒ Buses are groups of lines that carry "data", "addresses" or "control signals".

i.e. Data Bus, Address Bus, Control Bus.

⇒ At any one time, communication takes place between the mastee and one of the slaves.

Questions

- ① Compare microprocessors and microcontrollers?
- ② Explain the basic functional blocks of a microprocessor with neat diagrams.
- ③ Explain the microprocessor based system or organization of microcomputer with neat diagrams

Reference

Microprocessors and microcontrollers, 2e.

A. Nagarkami Page No: 2.1, 1.5 to 1.7

TRACE KTU

Introduction to 8086 processor

- Developed by INTEL in 1978.
- 3rd generation of processor.
- 20-bit Address bus
- 2^{20} ie. 1 Mb memory space.
- 16-bit data bus
- Registers are 16 bit, but it can also access as 8 bit.
- 40 pin IC

Title: Architecture of 8086 Microprocessor. Memory Organisation

8086 has a pipelined architecture.

In pipelined architecture, the processor will have a no. of functional units. The execution time of the functional units are overlapped.

The architecture can be internally divided into 2 separate functional units:

(1) Bus Interface Unit (2) Execution Unit

(1) Bus INTERFACE UNIT

⇒ main blocks/units are Segment registers, IP, Instruction queue, Address Generation Unit, Bus control unit.

⇒ Basic functions ✓ fetches instructions
✓ read data from M/y & I/O ports.
✓ write data to M/y & I/O ports.

Memory Organisation

8086 processor has a memory space of (2^{20}) [i.e. 1MB. M/y space]. It can be divided into segments of 64KB each. The segments are Code Segment (CS), Data Segment (DS), Extra Segment (ES) & Stack Segment (SS). This allows the s/m designer to allocate separate areas for storing ~~codes and data~~.
Mar Baselios College of Engineering & Technology

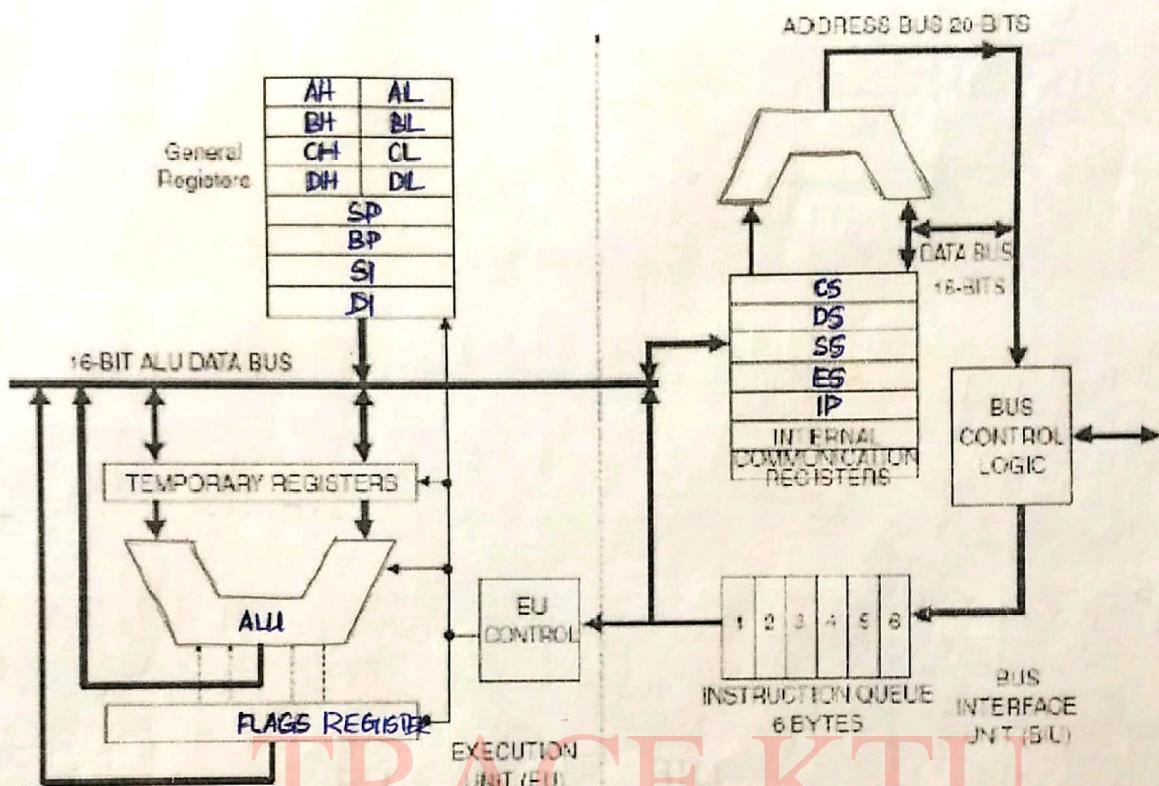
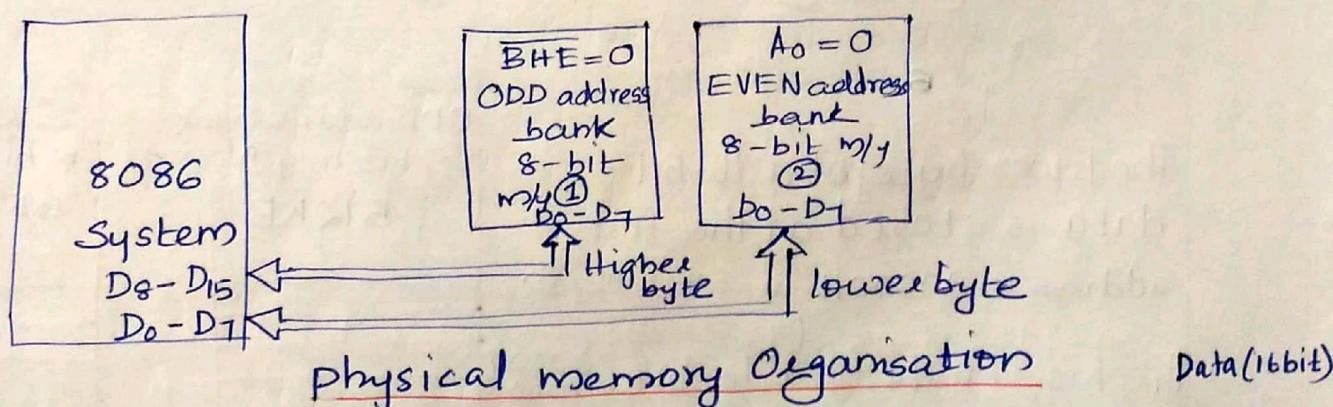


Fig 1.4 ARCHITECTURE OF 8086 MICROPROCESSOR



1 Mb (1024 kb) → 512 kb ODD ADDRESS BANK [higher byte]
 ↓ → 512 kb EVEN ADDRESS BANK [lower byte]

Address range of physical address : 00000 to FFFFF

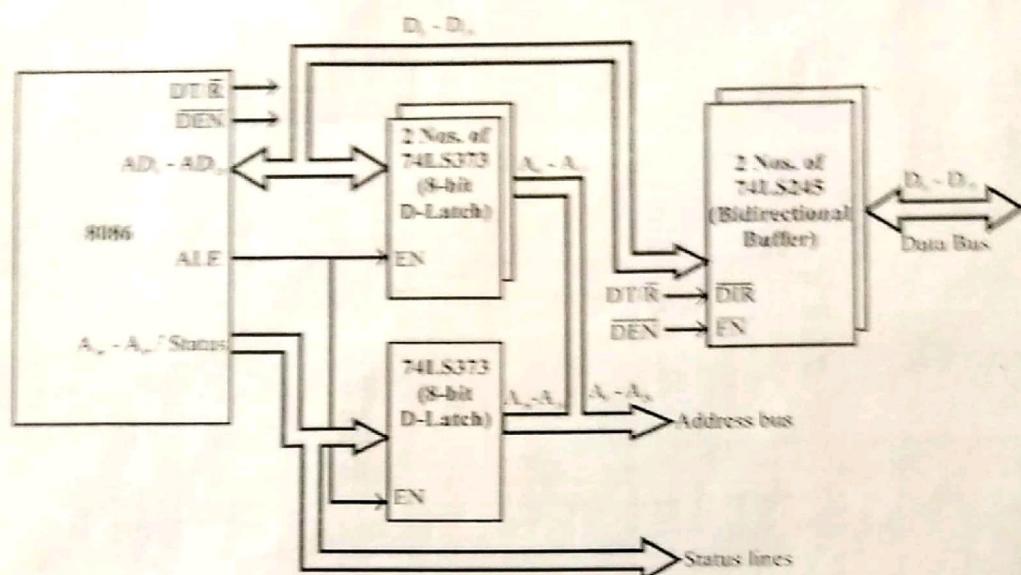


Fig. 1.5: Demultiplexing of address and data lines in an 8086 processor.

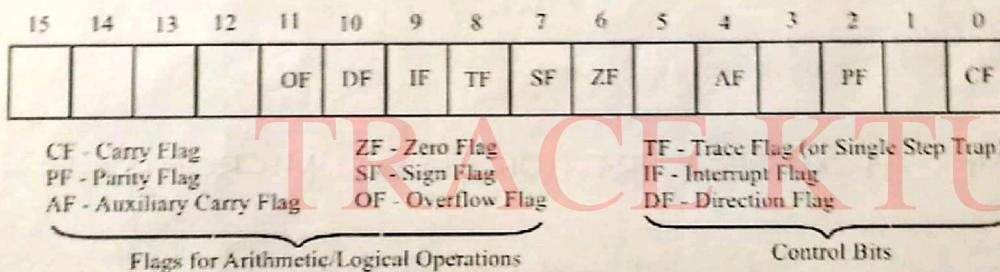
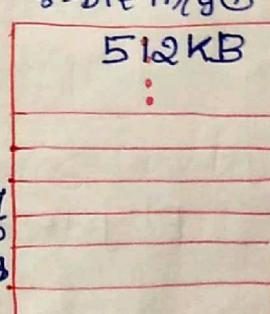


Fig. 1.16 : Bit positions of various flags in the flag register of 8086.

The lower byte of a 16-bit data is stored at the first address i.e. 000000H in 8-bit memory.

The higher byte of a 16-bit data is stored in the next address i.e. 00001 in 8-bit memory.



- ✓ When Addressline 0 (A0) become low, then the even my bank is enabled.
 - ✓ When Bus High Enable (BHE) become low, then the odd my bank is enabled.

Status of A₀ & BHE during memory access

Memory Bank	Operand type	Status of A ₀ BHE	Data lines used for my access	No. of Bus cycles
Even	Byte	0 1	D ₀ - D ₇	1
Odd	Byte	1 0	D ₈ - D ₁₅	1
Even/ Odd	Word	0 0	D ₀ - D ₁₅	1
Ddd/Even	Word	1 0 0 1	D ₈ - D ₁₅ D ₀ - D ₇	First cycle Second cycle

1. a) Segment Registers (16-bit)

4 segment registers (CS, DS, ES, SS) holds four segment base address respectively.

Contents of segment registers are programmable. Hence the programmer can access the code and data in any part of the my by changing the contents of the segment registers.

1. b) Instruction Pointer (IP)

IP holds the address of the next instruction to be fetch & gets incremented by two after every bus cycle.

1. c) Address Generation Unit (AGU)

Generates a - 20-bit physical address from the segment base address and an offset or effective address.

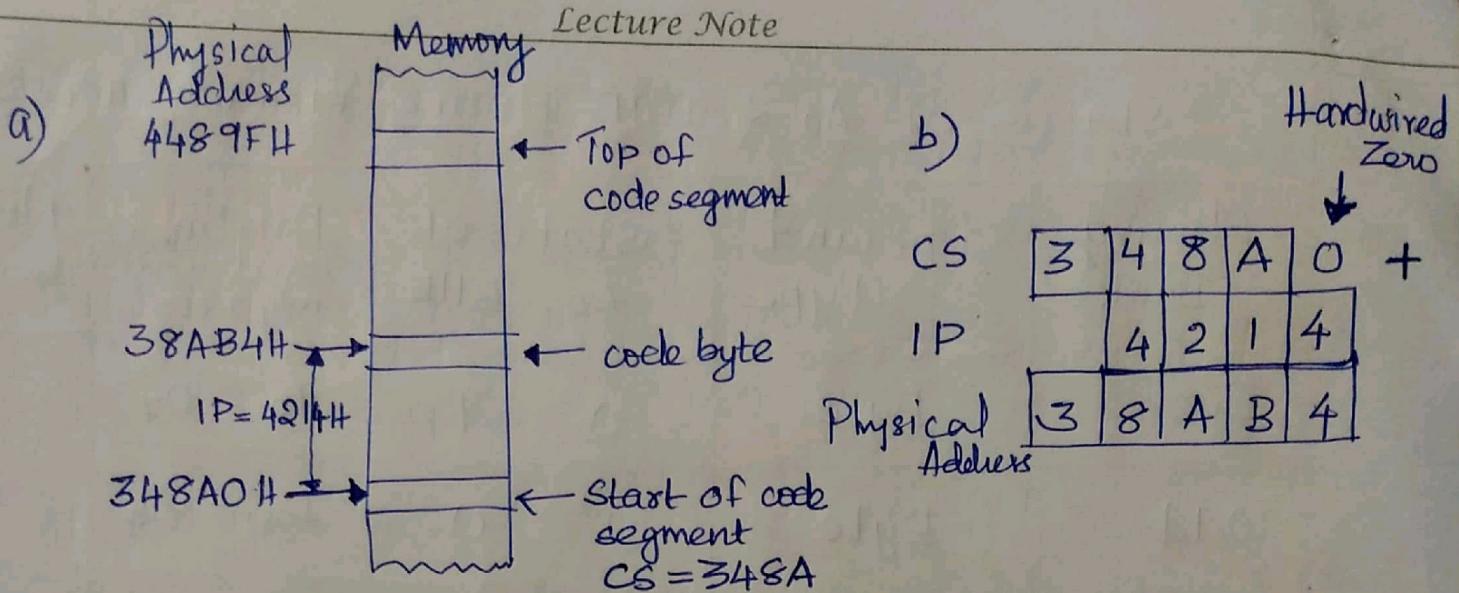


figure 1.7 Addition of IP to CS to produce the physical address of the code byte

a) Diagram b) Computation

Physical Memory Address = Base address \times 10H + Offset

Case-I	TRACE	(CS) \times 10H + IP
Case-II		(DS) \times 10H + Offset specified by the instruction.
Case-III		(ES) \times 10H + offset specified by the inst.
Case-IV		(SS) \times 10H + SP

1. d) Bus Control logic Unit

Generate all the bus control signals such as my read, my write, I/O read, I/O write etc....

1.e) Instruction Queue

FIFO group of registers where size = 6 bytes.

BIU fetches the instruction code from memory as the address generated from ALU and stores it in the queue. Then the Execution Unit (EU) fetches the instruction code from the queue.

2. Execution Unit (EU)

Basic Functions

- ✓ Decodes the fetched instruction from the queue
- ✓ then execute it.

EU contain * Internal Control System

TRACE KTU

- * ALU
- * General purpose register
- * Flag register.

2.a) Internal Control System

Decodes the instruction & translates the instruction.

2.b) ALU

16 bit ALU perform arithmetic & logical operations.

2.c) General Purpose Registers (16-bit)

AX, BX, CX, DX, SI, DI, BP, SP

- ✓ AX, BX, CX, DX registers can also access as 8-bit registers such that AH, AL, BH, BL, CH, CL, DH, DL.
- ✓ G1-P.R can be used for data storage. When they are not involved in any, special functions assigned to them.

RegistersSpecial function

AX

16-bit accumulator.

(stores the 16-bit results of certain arithmetic and logical operations.)

AL

8-bit accumulator

BX

Base Register: hold the base value in based addressing mode to access my data. [Data Segment]

CX

Counter Register: hold the count value in SHIFT, ROTATE and LOOP instructions.

DX

Data Register: hold data for multiplication and division operations.

SP

Stack Pointer: hold the offset address of Top of the stack my (TOS).

BP

Base Pointer: hold the base value in base addressing mode using the stack segment. (Offset)For
String
instruction
DISource Index: hold the index value of the source operand.Destination Index: hold the index value of the destination operand.

Mar Baselios College of Engineering & Technology

2-d) Flag Registers (fig: 1-6)

- ✓ 3 flags are known as control bits [DF, IF, TF]
 - used to control the processor operations.
- ✓ 6 flags for ALU operations. [CF, AF, PF, ZF, SF, OF]
 - used to indicate the status of the result of the ALU operations.

- ① CF (Carry Flag) $\begin{cases} 1 & \text{high;} \\ 0 & \text{no carry;} \end{cases}$ If there is a carry from the addition or borrow from the subtraction.
- ② AF (Auxiliary Carry Flag) $\begin{cases} 1 & \text{If there is a carry from low mibble to high mibble of a 8 bit no.} \\ 0 & \text{no carry.} \end{cases}$
- ③ OF (Overflow Flag) $\begin{cases} 1 & \text{If there is an arithmetic overflow ie. If the size of the result exceeds the capacity of the destination location.} \\ 0 & \text{no overflow.} \end{cases}$
- ④ PF (Parity Flag) $\begin{cases} 1 & \text{even parity.} \\ 0 & \text{odd parity.} \end{cases}$
- ⑤ ZF (Zero Flag) $\begin{cases} 1 & \text{result is zero.} \\ 0 & \text{result is not zero.} \end{cases}$
- ⑥ SF (Sign Flag) $\begin{cases} 1 & \text{result is negative.} \\ 0 & \text{result is positive.} \end{cases}$
- ⑦ DF (Direction Flag) $\begin{cases} 1 & \text{autodecrement SI \& DI my pointer} \\ 0 & \text{autoincrement SI \& DI my pointer} \end{cases}$
 {used for string data accessing}

⑧ IF (Interrupt Flag) \rightarrow 1; } to enable interrupt
 \rightarrow 0; to disable interrupt
(dueing external interrupt request)

⑨ TF (Trap Flag) \rightarrow 1; Step by step execution
 0; otherwise (8086 generates internal interrupt after execution of each instruction)

Questions

- i) Compute the physical address
 a) CS: F272 & IP: 0F2F b) 2F72 : 2971
- ii) Draw and Explain the flag register of 8086.
- iii) With **TRACE**, explain the architecture of 8086.
- iv) Explain the physical memory organisation of 8086.
- v) Explain the register organisation of 8086.

References

1. A. Nagarkatti, Microprocessors and Microcontrollers Second Edition Page No: 1.29 - 1.32
2. Douglas V. Hall, SSSP Rao, Microprocessors and Interfacing, 2e.

Title: 8086 Pinout and Signals

Vss (GND)	1	40	Vcc (+5V)
AD14	2	39	AD15
AD13	3	38	A16/S3
AD12	4	37	A17/S4
AD11	5	36	A18/S5
AD10	6	35	A19/S6
AD9	7	34	BHE/S7
AD8	8	33	MN/MX MIN
AD7	9	32	RD MODE
AD6	10	31	RQ/GTO HOLD
AD5	11	30	RQ/GT1 HLDA
AD4	12	29	LOCK WR
AD3	13	28	S2 M/I/O
AD2	14	27	S1 DT/R
AD1	15	26	S0 DEN
AD0	16	25	QS0 ALE
NMI	17	24	QS1 INTA
INTR	18	23	TEST
CLK	19	22	READY
Vss (GND)	20	21	RESET

MAX MODE

- ✓ First 16-bit processor released by INTEL in 1978
- ✓ Designed using the HMOS Technology.
- ✓ Contains
 - * 29,000 transistors
 - * 40 pin DIP
 - * Requires single 5 Volt Supply
 - * No internal clock circuit
 - * External clock \rightarrow 8284 clock generator generates max internal clock of 5MHz.
- ✓ In addition to 20 address lines, 8086 uses a separate 16 bit address; so it generates 64KB I/O addresses.
 - To differentiate
 - \rightarrow M/I/O addresses
 - \rightarrow I/O addresses
 - using $M/I\bar{O} \rightarrow$ high
 $M/I\bar{O} \rightarrow$ low

Common Signals Lecture Note

<u>Pin</u>	<u>Function</u>
① & ⑩	Ground
④	Vcc
⑯ to ②	A ₀ to A ₁₄
③	A ₁₅
⑧ to ⑤	A ₁₆ /S ₃ , A ₁₇ /S ₄ , A ₁₈ /S ₅ , A ₁₉ /S ₆
⑨	Clock } 8284 clock generator chip used to generate the required clock.

fig. 1.5 explain the demultiplexing of Address/Data and Address/Status lines in 8086 processors.

For Demultiplexing address/data using 3 Nos of 8-bit 74LS373 D-Latches. When ALE become high, two latches contain address A₀ to A₇ and A₈ to A₁₅ and third latch contains address A₁₆ to A₁₉. Then the address given out through the 20 bit address bus.

When ALE become low, two bidirectional buffer (8 bit) contain data D₀ to D₇ & D₈ to D₁₅ and data given out through the 16 bit data bus.

Third latch contains status signals S₃ to S₆. & given out through the status lines.

S_4	S_3	
0	0	ES
0	1	SS
1	0	CS
1	1	DS

Specifies the segment Register

$S_5 \rightarrow$ Indicates the status of an 8086 interrupt enable flag.

$S_6 \rightarrow$ low ; 8086 is the bus master

\searrow high ; local bus master gains the ^{bus} control from the processor.

(34) BHE / $S_7 \rightarrow$ 0 ; select odd address bank

Bus High Enable

(16) A_0 (address line 0) \rightarrow 0 ; select even address bank

(17) & (18) \rightarrow Interrupt pins

17 \rightarrow NMI (Non maskable Interrupt)

18 \rightarrow Interrupt Request

(23) MN/MX $\begin{cases} \rightarrow \text{high} ; 8086 \text{ in minimum mode} \\ \rightarrow \text{low} ; 8086 \text{ in maximum mode} \end{cases}$

(32) RD $\begin{cases} \rightarrow \text{low} ; \text{M/R or I/O read operation} \\ \rightarrow \text{high} ; \text{no read} \end{cases}$

<u>Pin</u>	<u>Lecture Note</u>	<u>Function</u>
(23) TEST ↓ low; } stop execution } of WAIT instruction.		Used to synchronize an external activity to the processor's internal operation. 8086 will enter a WAIT state after execution of the WAIT instruction. It will resume execution only when TEST is made low.
(22) READY	→ normally 1 (high) → low; Introduce WAIT states OR	Used by the memory or I/O device to get extra time for data transfer.
(21) RESET	→ high; system reset for atleast 4 clock cycles. CS points to FFFF and DS, SS, ES, IP & flag registers are cleared & queue is emptied.	

8086 operates in 2 Modes

Min mode	Max mode
* MN/MX → high	* MN/MX → low
* S/m → uniprocessor S/m	* multiprocessor S/m
* 8086 itself generates bus control signals.	* Use External bus controller 8288 to generate bus control signals.

→ Signals assigned to pin 24 to 31 will be different for minimum and maximum mode. All other pins are common for min and max mode of operations.

Min Mode

PIN	Function
(31) HOLD → high	; Request to processor from other bus master for getting bus control.
(30) HLDA → high	; Acknowledgement sig by the processor to the bus master after giving bus control.
(29) WR → low	; To write data to memory/I/O port.
(28) M/IO → high	; memory access. → low ; I/O address access.
(27) DT/R → high	; to transmit data. → low ; to receive data.
(26) DEN → low	; to enable the external bi-directional data buffers.
(25) ALE → high	; address lines → low ; data lines/status lines

Used to demultiplex address & datalines

(24) INTA → an acknowledgement signal produced by the processor, when the interrupt request is accepted.

Maximum Mode

31 $\overline{RQ}/\overline{GTO}$ } (Bus Request and Bus Grant)
 30 $\overline{RQ}/\overline{GITI}$ } → Request used by the local bus masters to force the processor in order to release the local bus at the end of the processor's current bus cycle.
 $\overline{GTO} > \overline{GITI}$
 priority

(29) LOCK ⇒ While executing an instruction prefixed by Lock, this pin is tied low in order to prevent other bus masters from gaining control of the system bus!

(28) (27) (26) $\overline{S2}, \overline{S1}, \overline{S0}$ ⇒ Status signals (Used by 8288 bus controller in order to generate timing & control signals.)

$\overline{S2}$	$\overline{S1}$	$\overline{S0}$	Machine cycle
0	0	0	Interrupt ack
0	0	1	Read I/O port
0	1	0	Write I/O port
0	1	1	Halt
1	0	0	Code access
1	0	1	Read mem
1	1	0	Write mem

②5 & ②4 \Rightarrow Instruction Queue Status

QSO , QSI Used by external device, to track the internal status of the queue in 8086.

QSI	QSO	Queue Operation
0	0	No operation
0	1	First byte of an opcode from queue.
1	0	Empty the queue
1	1	Subsequent byte from queue.

Questions

1. Explain the pins and signals of 8086.
2. What are the functions of the TEST and READY input pins of 8086?
3. Discuss the functions of the signals which are significant with respect to maximum mode in 8086.
4. Discuss the functions of the signals which are significant with respect to minimum mode in 8086.

Reference

1. A. Nagarkatti, Microprocessors and Microcontrollers
Second edition page No: 1.24 to 1.29
2. A. K Ray, Advanced Microprocessors and peripherals.
page No: 8-12

TRACE KTU

MINIMUM MODE 8086 SYSTEM AND TIMINGS

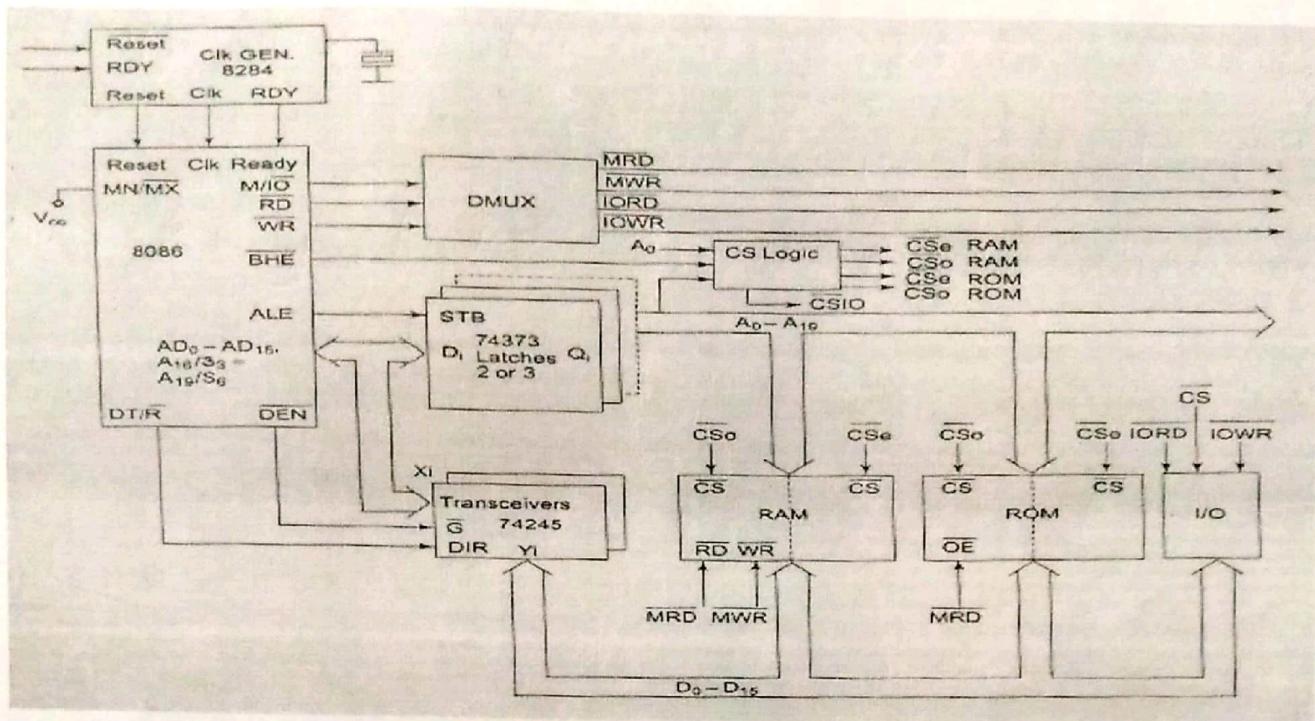


Fig 1.1. Minimum Mode 8086 System

In a minimum mode 8086 system, the microprocessor 8086 is operated in minimum mode by strapping its MN/MX* pin to logic1.

In this mode, all the control signals are given out by the microprocessor chip itself. There is a single microprocessor in the minimum mode system.

The remaining components in the system are latches, transreceivers, clock generator, memory and I/O devices. Some type of chip selection logic may be required for selecting memory or I/O devices, depending upon the address map of the system.

The latches are generally buffered output D-type flip-flops, like, 74LS373 or 8282. They are used for separating the valid address from the multiplexed address/data signals and are controlled by the ALE signal generated by 8086.

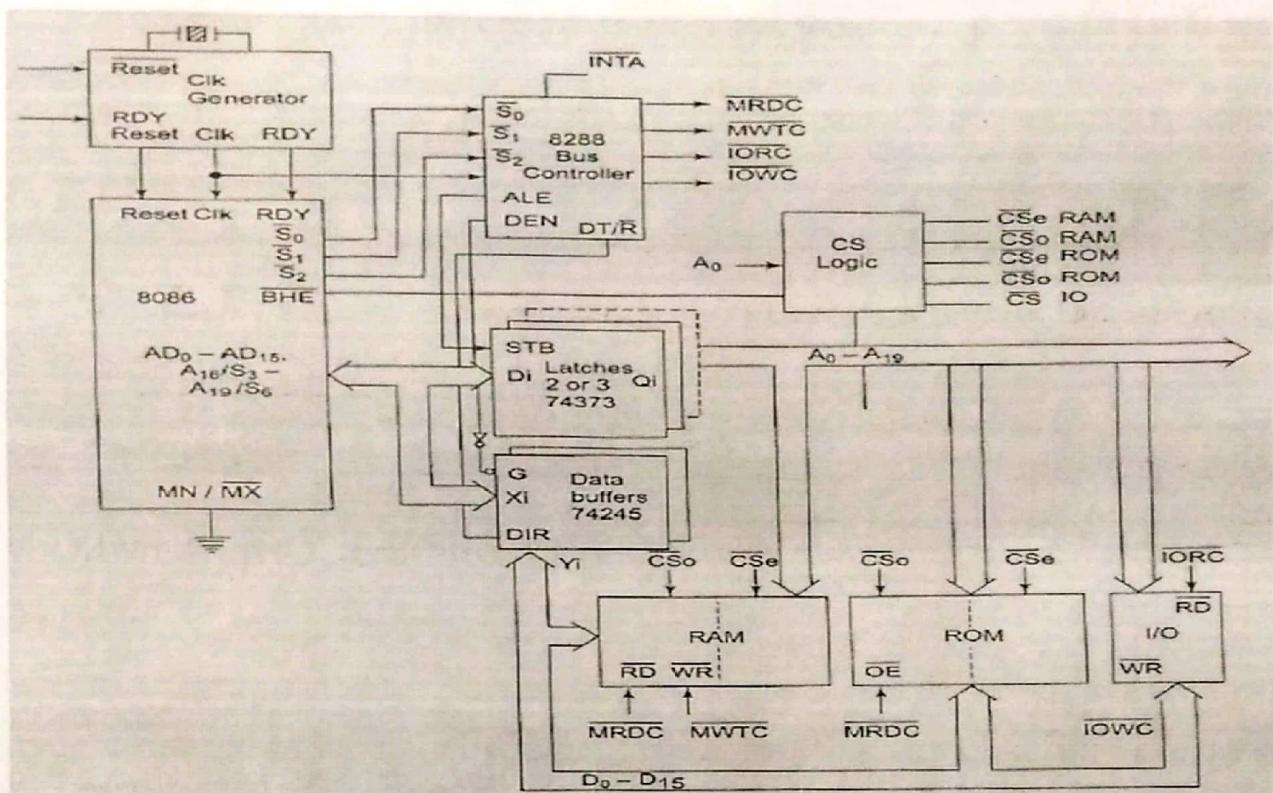
Transreceivers are the bidirectional buffers and sometimes they are called as data amplifiers. They are required to separate the valid data from the time multiplexed address/data signal. They are controlled by two signals, namely, DEN* and DT/R*. The DEN* signal indicates that the valid data is available on the data bus, while DT/R indicates the direction of data, i.e. from or to the processor.

The system contains memory for the monitor and users program storage. Usually, EPROMS are used for monitor storage, while RAMs for users program storage.

A system may contain I/O devices for communication with the processor as well as some special purpose I/O devices.

The clock generator generates the clock from the crystal oscillator and then shapes it and divides to make it more precise so that it can be used as an accurate timing reference for the system. The clock generator also synchronizes some external signals with the system clock.

MAXIMUM MODE 8086 SYSTEM



TRACE KTU

In the maximum mode, the 8086 is operated by strapping the MN/MX* pin to ground.

In this mode, the processor derives the status signals S₂*, S₁* and S₀*. Another chip called bus controller derives the control signals using this status information.

In the maximum mode, there may be more than one microprocessor in the system configuration. The other components in the system are the same as in the minimum mode system.

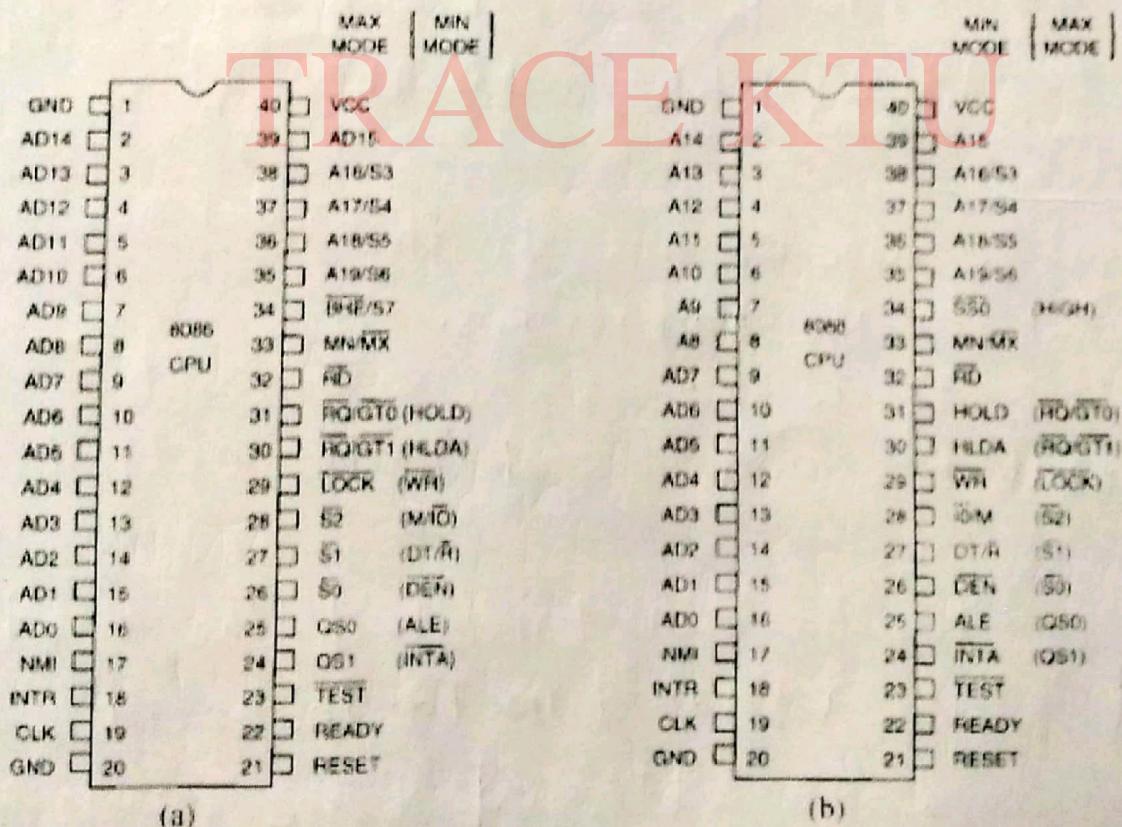
The basic functions of the bus controller chip IC8288, is to derive control signals like RD* and WR* (for memory and I/O devices), DEN*, DT/R*, ALE, etc. using the information made available by the processor on the status lines. The bus controller chip has input lines S₂*, S₁* and S₀* and CLK. These inputs to 8288 are driven by the CPU. It derives the outputs ALE, DEN*, DT/R*, MWTC*, AMWC*, IORC*, IOWC* and AIOWC*. The AEN*, IOB and CEN pins are specially useful for multiprocessor systems. AEN* and IOB are generally grounded. CEN pin is usually tied to +5V.

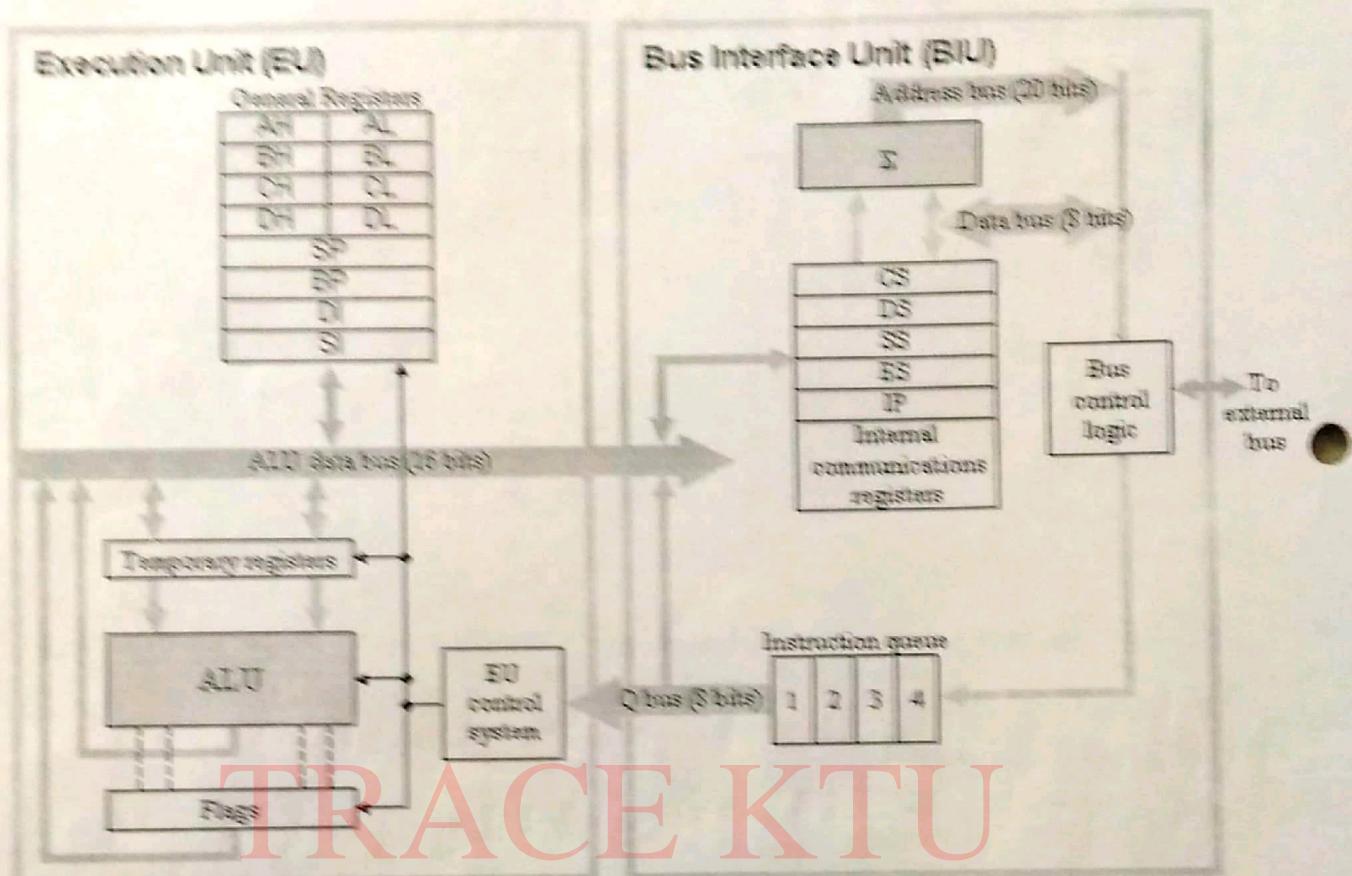
IORC*, IOWC* are I/O read command and I/O write command signals respectively. These signals enable an IO interface to read or write the data from or to the addressed port. The MRDC*, MWTC* are memory read command and memory write command signals respectively and may be used as memory read and write signals. All these command signals instruct the memory to accept or send data from or to the bus. For both of these write command signals, the advanced signals namely AIOWC* and AMWTC* are available. They also serve the same purpose, but are activated one clock cycle earlier than the IOWC* and MWTC* signals, respectively.

Comparison between 8086 and 8088

8086	8088
* Access memory in bytes and words	* Access memory in bytes only
* 16 bit data bus and 20 bit address bus	* 8 bit data bus and 20 bit address bus
* BHE signal to access higher byte	* Data bus is 8bit wide, it does not have BHE signal.
* 6 byte instruction queue.	* 4 byte instruction queue.
* (M/I) for memory or I/O access.	* RD for memory or I/O access.
* Pin no:34 is BHE/S7 . During T2, T3, T4 it carries S7. In maximum mode 8087 monitors this pin to identify the CPU as 8086 or 8088 and accordingly sets the queue length to 4 or 6 bytes.	* Pin no. 34 is SS0 . It acts as SD in the minimum mode of 8088. This signal is combined with RD and (DT/R) to decode the function of the current bus cycle.

8086/8088 Pin Configuration





TRACE KTU

Architecture of 20SS processor

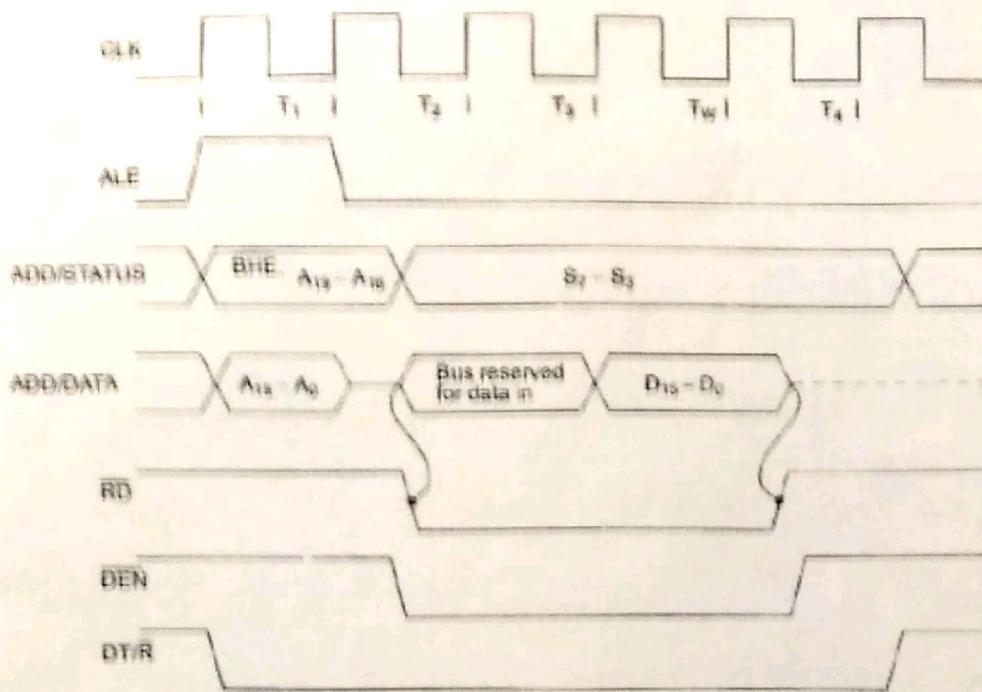
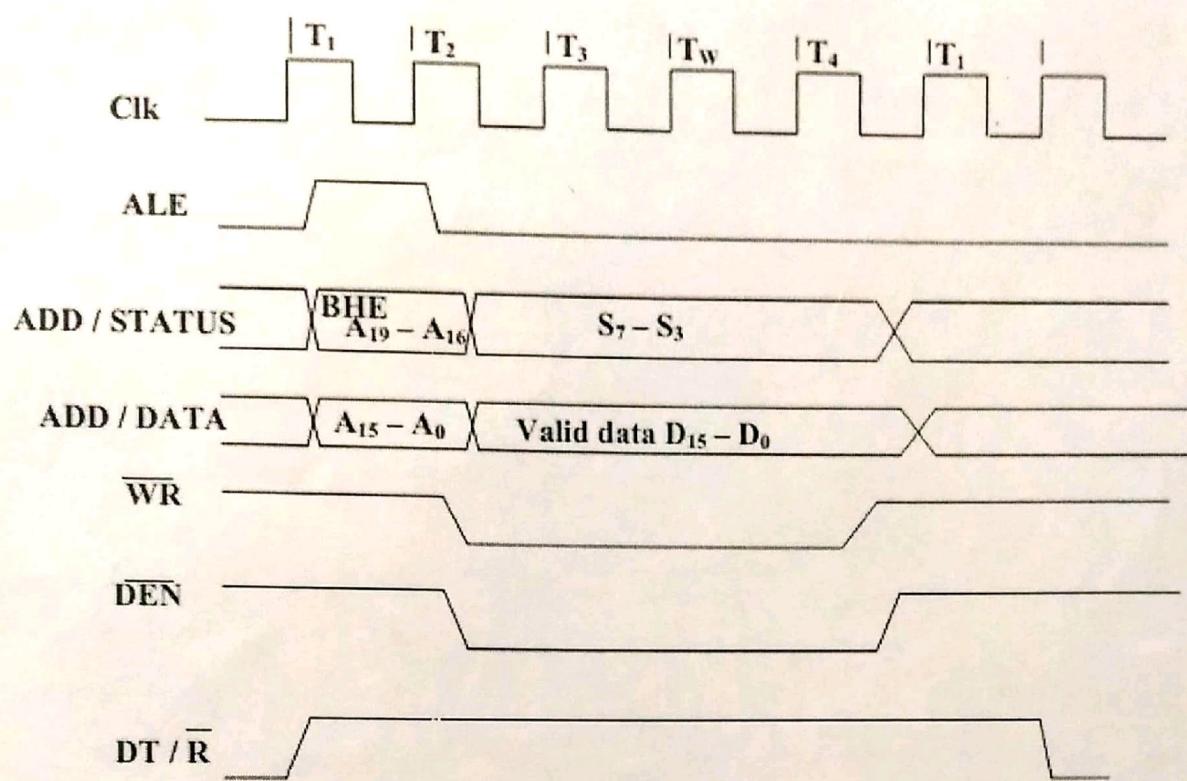


Fig. 2.4 Read cycle timing diagram for minimum mode

The read cycle begins in T₁ with the assertion of the Address latch Enable(ALE) high and M/I/O signal ($M/I/O \begin{cases} \rightarrow \text{high; memory address} \\ \rightarrow \text{low; I/O address} \end{cases}$). During the negative edge of this sig, the valid address is latched on the ~~local~~ bus.

At T₂ the address is removed from the ~~local~~ bus and is sent to the output.

Then the bus is tristated. The read(RD) signal is also activated in T₂. i.e. become low that causes the addressed device to enable its data bus driven. After RD goes low, the valid data is available on the data bus.



Write Cycle Timing Diagram for Minimum Mode

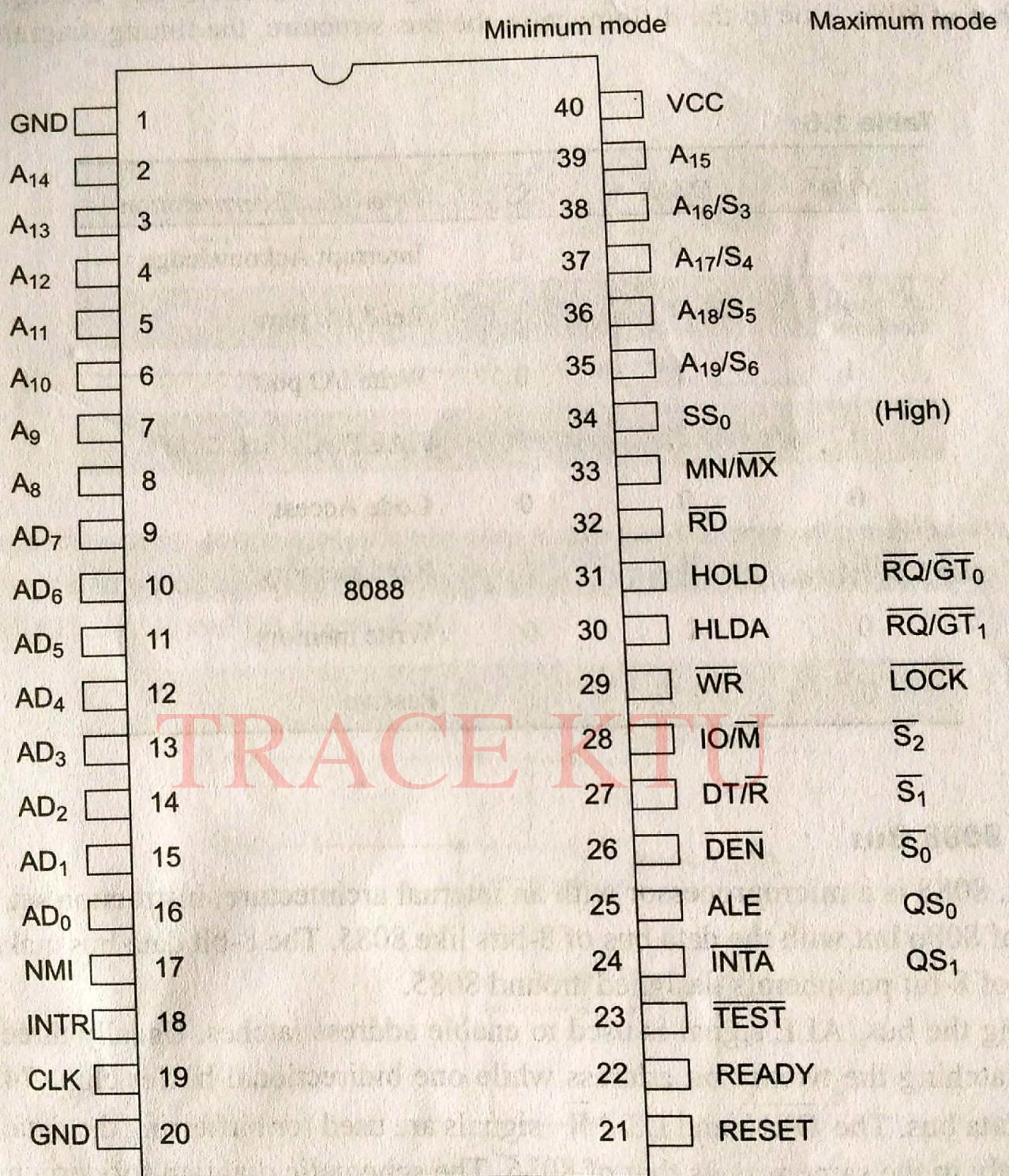


Fig. 1.18 Pin Diagram of 8088

Table 1.6

IO/M	DT/R	\overline{SS}_0	<i>Operation/Interpretation</i>
1	0	0	Interrupt Acknowledge
1	0	1	Read I/O port
1	1	0	Write I/O port
1	1	1	HALT
0	0	0	Code Access
0	0	1	Read memory
0	1	0	Write memory
0	1	1	Passive

Title: Addressing Modes of 8086

Addressing Mode

The method of specifying the data to be operated by the instruction.

⇒ 12 Addressing Modes and they can be classified into 5 groups.

Group 1

Addressing Modes for "register & Immediate data".

- ① Register Addressing.
- ② Immediate Addressing.

Group 2

Addressing Modes for "memory data".

- ③ Direct Addressing
- ④ Register Indirect Addressing.
- ⑤ Based Addressing.
- ⑥ Indexed Addressing
- ⑦ Based Index Addressing
- ⑧ String Addressing

Group 3

Addressing modes for I/O ports

- ⑨ Direct I/O port Addressing
- ⑩ Indirect I/O port Addressing

Group 4

- ⑪ Relative Addressing Mode

Group 5

- ⑫ Implied Addressing Mode

① Register A.M

The instruction will specify "the name of the register" which holds the data to be operated by the instruction.

eg: $MOV CL, DH ; (CL) \leftarrow (DH)$
 $MOV BX, DX ; (BX) \leftarrow (DX)$

② Immediate A.M

an 8-bit or 16-bit data is specified as part of the instruction.

eg: $MOV DL, 08H ; (DL) \leftarrow 08H$
~~TRACE KU~~
 $MOV AX, 0A9FH ; (AX) \leftarrow 0A9FH$

③ Direct Addressing

an unsigned 16-bit displacement or signed 8-bit displacement will be specified in the instruction

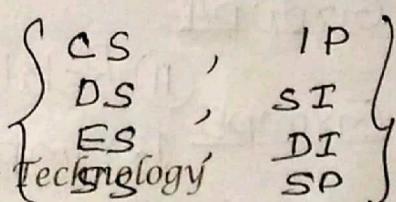
Displacement i.e. \Rightarrow effective address
 $(E.A)$

8-bit displacement

E.A is obtained by sign extending the 8-bit displacement to 16-bit.

$$\text{Memory Address} = \text{BA} + \text{EA}$$

(20-bit physical
addr. of m/y)



eg: MOV DX, [08H]

EA = 0008H (sign extended 8-bit displacement)

BA = (DS) \times 16₁₀ ; MA = BA + EA
(Base address) \times 10H (Memory address)

(DX) \leftarrow (MA)

eg: MOV AX, [089DH]

EA = 089DH ; BA = (DS) \times 10H

MA = BA + EA

(AX) \leftarrow (MA)

Register Indirect Addressing

The instruction will specify "the name of the register, which holds the effective Address

(BX, SI, DI)

eg: MOV CX, [BX]

EA = (BX)

BA = (DS) \times 10H

MA = BA + EA

(CX) \leftarrow (MA)

MOV AX, [SI]

EA = (SI)

BA = (DS) \times 10H

MA = BA + EA

(AX) \leftarrow (MA)

Based A.M

In this A.M, BX or BP register is used to hold a base value of E.A and

a signed 8-bit or unsigned 16-bit will be specified in the instruction.

eg: $MOV AX, [BX + 08H]$

eg: $MOV AX, [BX + 08H]$

$0008H \leftarrow \begin{matrix} \text{sign} \\ \text{extend} \end{matrix} 08H$

Segment

$BX \rightarrow DS$

$BP \rightarrow SS$

$$EA = (BX) + 0008H$$

$$BA = (DS) \times 10H$$

$$MA = BA + EA$$

$$(AX) \leftarrow (MA)$$

b) Indexed Addressing Mode

In this A.M, SI or DI register is used to hold an index value for my data and a signed 8-bit displacement or unsigned 16-bit displacement will be specified in the instruction.

eg: $MOV CX, [SI + 08H]$

$0008H \leftarrow \begin{matrix} \text{sign} \\ \text{extend} \end{matrix} 08H ; EA = (SI) + 0008H$

$$BA = (DS) \times 10H ; MA = BA + EA$$

$$(CX) \leftarrow (MA)$$

Based Indexed Addressing Mode

In this A.M, SI or DI register is used to hold an index value for my data and a signed 8-bit displacement or unsigned 16-bit displacement will be specified in the instruction.

eg: $MOV DX, [BX + SI + OAH]$

$$0000AH \xleftarrow[\text{extend}]{\text{sign}} OAH ; EA = (BX) + (SI) + 0000AH$$

$$BA = (DS) \times 16_{10} ; MA = BA + EA$$

$$(DX) \leftarrow (MA)$$

8) String A.M (operate on string data)

In this A.M, the E.A of Src data $\xrightarrow[\text{in}]{\text{stored}}$ \rightarrow SI reg & the E.A of Dest. Data $\xrightarrow[\text{in}]{\text{stored}}$ \rightarrow DI reg.

\Rightarrow Seg. register is used for calculating base address.

for src data \rightarrow DS

for dest. data \rightarrow ES.

eg: $MOVSB$ REI > $MOVSB$

Source; $EA = (SI) ; BA = (DS) \times 16_{10} ; MA = BA + EA$

Dest; $EA_E = (DI) ; BA_E = (ES) \times 16_{10} ; MA = BA_E + EA_E$

$$(MA_E) \leftarrow (MA)$$

If $DF=1$, $(SI) \leftarrow (SI)-1$ & $(DI) \leftarrow (DI)-1$

If $DF=0$, $(SI) \leftarrow (SI)+1$ & $(DI) \leftarrow (DI)+1$

⑨ Direct I/O port Addressing Mode

(used to access data from I/O devices or port)

a. 8-bit port Address is directly specified in the instruction.

e.g: IN AL, [094]

Port addre = 094

TRACE KTU

⑩ INDIRECT I/O port A.M

(used to access data from I/O devices or port)

The instru will specify "the name of the reg which holds the port address".

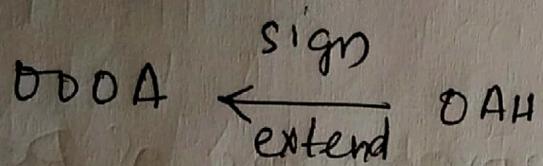
In 8086, 16 bit port address is stored in DX reg

e.g: Port addre = (DX); (PORT) \leftarrow (AX)

The content of AX is moved to port whose address is specified by DX register.

⑪ Relative A.M

The E.A of a Pgm instruction is specified relative to IP by an 8-bit signed displacement
eg: JZ 0A1H



If ZF=1, then $(IP) \leftarrow (IP) + 000AH$

$$(EA) = (IP) + 000AH$$

$$(BA) = (CS) * 16^{16}$$

MA = (BA) + (EA); then the pgm ctrl

TRACE KTU

jumps to new address as calculated.

If ZF=0; then next instr

of the Pgm is executed.

⑫ Implied Addressing

In this A.M, the instruction itself will specify the data to be operated by the instruction

eg: CLC

Clear carry
 $; CF \leftarrow 0$

is dropped by the requesting master, the HLDA is dropped by the processor at the trailing edge of the next clock, as shown in Fig. 1.14 (c). The other conditions have already been discussed in the signal description section for the HOLD and HLDA signals.

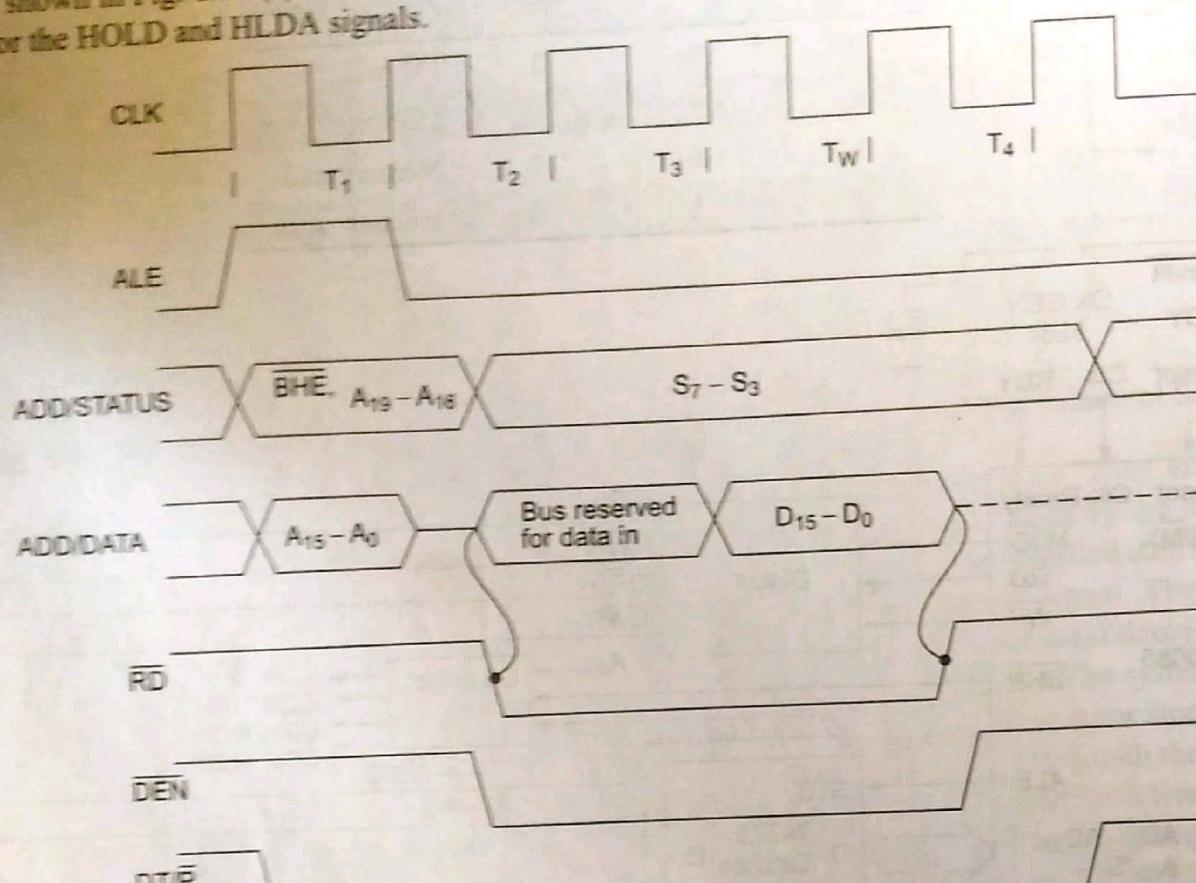


Fig. 1.14(a) Read Cycle Timing Diagram for Minimum Mode

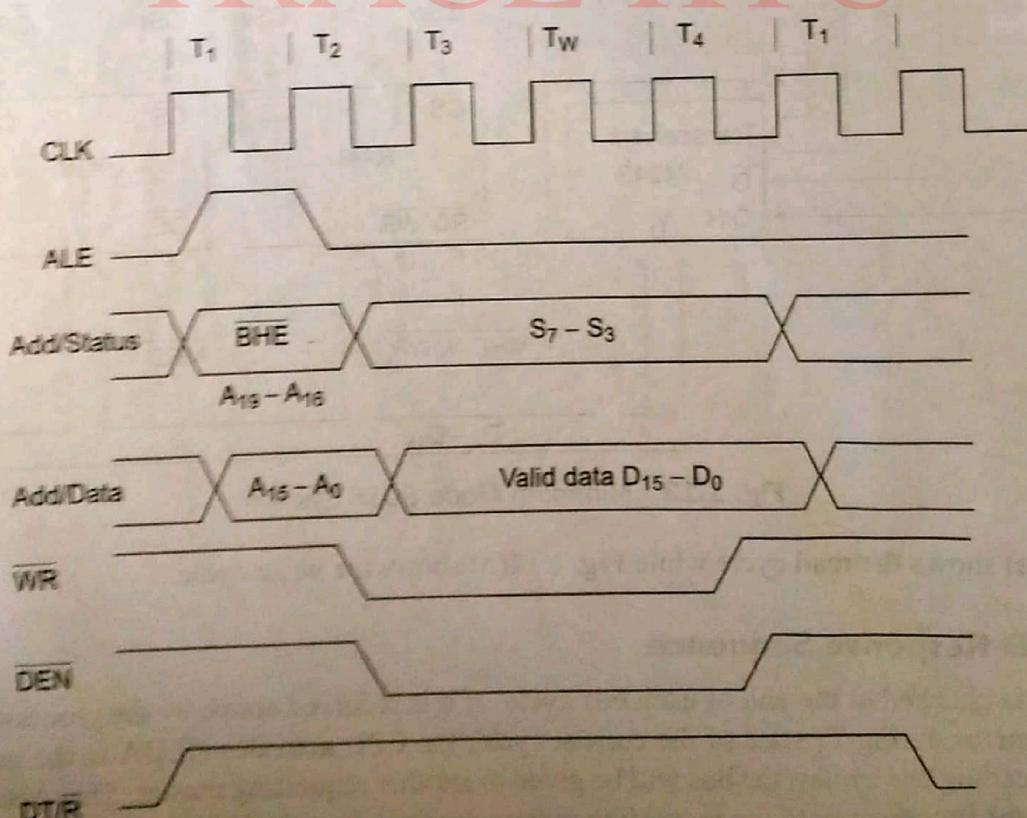


Fig. 1.14(b) Write Cycle Timing Diagram for Minimum Mode Operation

Title: ASSEMBLER DIRECTIVES

These are the instructions to the assembler regarding the pgm being assembled.

functions

- ✓ Used to specify start & end of a pgm
- ✓ Attach value to variable.
- ✓ Allocate storage locations to i/p/o/p data.
- ✓ To define start and end of segments, procedures, macros etc.

① ASSUME

Indicates the name of each segment to the assembler.

eg: ASSUME CS: CODE, DS: DATA

↳ Informs the assembler that the instructions of the programs are stored in the segment CODE & data are stored in the segment DATA.

② DB

Used to define byte type variable.

General form variablename DB value/ values

eg: AREA DB 45H

③ DW

Used to define word type variable
variablename DW value/values

eg: Weight DW 1250H

④ DD / DQ / DT

↳ Used to define double word (32 bit),
Quad word (64 bit), Ten bytes of a variable
respectively.

⑤ SEGMENT AND ENDS

Used to indicate
the beginning of
a code/data(stack
segment).

Used to indicate
the end of code/
data / stack segment.

general form Segname SEGMENT

- - - - } pgmcode
- - - - } or
- - - - } Data defn
- - - - } Stats

Segname ENDS

⑥ ORG → Used to assign the starting
address(EA) for a pgm / data segment

eg: ORG 1000H

⑦ EVEN

Inform the assembler to store pgm/data in segment starting from even address.

⑧ EQU

Used to attach a value to a variable.

eg: PORT1 EQU OF2H ; the value of variable PORT1 is F2H.

⑨ PROC , FAR, NEAR, ENDP

↓ ↓ ↓ → indicate end
beginning of inter intrasegment of a procedure.
procedure segment call

General form } PROC name PROC [NEAR] FAR]
 { } pgm stats
 { } in procedure
 RET
 PROC name ENDP

⑩ MACRO AND ENDM

Indicate the beginning of a macro → end of a macro

General Form } Macro name MACRO [Arg1, Arg2, ...]
 { } definitions
 { } declaration
Macro name ENDM Pgm statements

Question

Explain the assemble directives of 8085 with examples?

Reference

- (i) Advanced Microprocessors & peripherals -3e, A.K Ray
Page No: 68
- (ii) Microprocessors and Microcontrollers, 2e
A. Nagoor Kani
Page: 8.9

TRACE KTU

Title Sample Prgms (Hardware)

(1) Addition of Two - 16-bit numbers

(Hardware kit
Pgm)

```

MOV CX, 0000H
MOV AX, [1200]
ADD AX, [1202]
JNC L1
L1: MOV [1300], AX
    MOV [1302], CX
    HLT

```

Input

NUM1 : [1200, 1201] = 1234H

NUM2 : [1202, 1203] = 5613H

SUM : [1300, 1301, 1302, 1303] = 0000 6847H

(2) Subtraction of Two 16-bit numbers

(Hardware kit
Pgm)

```

MOV CX, 0000H
MOV AX, [1200]
SUB AX, [1202]
JNB L1
INC CX
NEG AX ; Taking 2's complement
L1: MOV [1300], AX if result is -ve.
    MOV [1302], CX
    HLT

```

NUM1 : [1200, 1201] = 1101 H
 NUM2 : [1202, 1203] = 5678 H
 Difference: [1300, 1301, 1302, 1303] = 0001 4577 H
 (0001 indicates -ve result)

(3) To multiply two 16-bit unsigned numbers stored in memory and to get the 32-bit product in memory (masm pgm)

DATA1 SEGMENT

MULTIPLICAND DW 204AH

MULTIPLIER DW 3B2AH

PRODUCT DW 2 DUP(0)

DATA ENDS

CODE1 SEGMENT

ASSUME CS:CODE1 DS:DATA1

START: MOV AX, DATA1

MOV DS, AX

MOV AX, MULTIPLICAND

MUL MULTIPLIER

MOV PRODUCT, AX

MOV PRODUCT+2, DX

MOV AH, 4CH

INT 21H

CODE1 ENDS

Mar Baselios College of Engineering & Technology

END START

(H) Double word by word division

(MASM pgm)

DATA SEGMENT

DIVIDEND DD 0CF7564CH

DIVISOR DW 3567H

QUOTIENT DW 1 DUP(0)

REMAINDER DW 1 DUP(0)

DATA ENDS

CODE SEGMENT

ASSUME CS: CODE, DS: DATA

START: MOV AX, DATA

MOV DS, AX

MOV AX, WORD PTR DIVIDEND

MOV DX, WORD PTR DIVIDEND+2

DIV DIVISOR

MOV QUOTIENT, AX

MOV REMAINDER, DX

MOV AH, 4CH

INT 21H

CODE ENDS

END START

(5) To display the message 'WELCOME
Thiruvananthapuram' in the display mon.

DATA1 SEGMENT

MSG DB 'Welcome to MBCET, Thiruvananthapuram'; here \$ functions as a string terminator

DATA1 END

CODE1 SEGMENT

ASSUME CS: CODE1, DS: DATA1

START: MOV AX, DATA1

MOV DS, AX

MOV AH, 09H

MOV DX, OFFSET MSG

INT 21H

} DOS function call for displaying a string on the screen.

MOV AH, 4CH

INT 21H

} DOS fn call to terminate the execution of the user program to exit to DOS command prompt.

CODE1 ENDS

END START

6) To move a string from a source location to a destination location, skipping over 100 bytes of memory location.

DATA1 SEGMENT

SRC-STR DB 'Mar Baselios Engineering College'

STR_LENGTH EQU (\$ - SRC-STR)

DB 100 DUP(0)

DST-STR DB STR_LENGTH DUP(0)

DATA1 ENDS

CODE1 SEGMENT

ASSUME CS: CODE1, DS: DATA1, ES: DATA1

START : MOV AX, DATA1

MOV DS, AX

MOV ES, AX

LEA SI, SRC-STR

LEA DI, DST-STR

MOV CX, STR_LENGTH

CLD

REP MOVSB

MOV AH, 4CH

INT 21H

CODE1 ENDS

Mar Baselios College of Engineering & Technology

END START

Page #..... 49

(7) Program for adding a series of 8-bit numbers.

DATA SEGMENT

LIST DB 25H, 23H, 12H, 0FFH, 0FFH, 86H

COUNT EQU 061H

SUM DB 02H DUP(0)

DATA ENDS

CODE SEGMENT

ASSUME CS: CODE, DS: DATA

START: MOV AX, DATA

MOV DS, AX

MOV CX, COUNT

MOV AX, 000DH

MOV SI, OFFSET LIST

L2: ADD AL, [SI]

INC LI

INC AH

L1: INC SI

MOV AH, 4CH

INT 21H

Loop L2

CODE ENDS

MOV SUM, AL

END START

MOV SUM+1, AH

(8) To find the largest signed number

Lecture Note

DATA SEGMENT

LIST DB 7FH, OF3H, 80H, 8FH

COUNT EQU 04H

LARGEST DB 01H DUP(0)

DATA ENDS

CODE SEGMENT

ASSUME CS:CODE, DS: DATA

START: MOV AX, DATA

MOV DS, AX

MOV SI, OFFSET LIST

MOV CX, COUNT - 1

MOV AL, [SI]

L2: CMP AL, [SI+1]

JAE LI ; for largest

JBE LI ; for smallest

MOV AL, [SI+1]

LI: INC SI

MOV AH, 4CH

INT 21H

LOOP L2

CODE ENDS

MOV LARGEST, AL

END START

(9) To sort 8-bit unsigned nos. in ascending order

DATA SEGMENT

LIST DB 7FH, 00H, FFH, 80H

COUNT EQU 04H

DATA ENDS

CODE SEGMENT

ASSUME CS:CODE, DS:DATA

START: MOV AX, DATA

MOV DS, AX

MOV BX, COUNT - 1

L3: MOV CX, BX

MOV SI, OFFSET LIST //LEA SI, LIST

L2: MOV AL, [SI]

CMP AL, [SI + 1]

JBE L1 //; for ascend ; JAE L1 for desc

XCHG AL, [SI + 1]

MOV [SI], AL

L1: INC SI

DEC BX

DEC CX

JNZ L3

JNZ L2

MOV AH, 4CH

INT 21H

Mar Baselios College of Engineering & Technology

CODE ENDS

Page #...52

(10) Addition of two 3x3 matrices

Lecture Note

DATA SEGMENT

```
MAT1    DB  0FFH, 02H, 03H, 04H, 05, 06H, 07H, 08H, 09H  
MAT2    DB  0FFH, 02H, 03H, 04H, 05H, 06H, 07H, 08H, 09H  
MAT3    DW  09H DUP(00)  
COUNT   EQU 09H
```

DATA ENDS

CODE SEGMENT

```
ASSUME CS:CODE, DS:DATA
```

START: MOV AX, DATA

MOV DS, AX

MOV CX, COUNT

MOV SI, OFFSET MAT1

MOV DI, OFFSET MAT2

MOV BX, OFFSET MAT3

NEXT: XOR AX, AX , AX is initialized
to zero & carry flag
is reset.

MOV AL, [SI]

ADD AL, [DI]

INC LI

INC AH

(I)

```

LI: MOV [BX], AX
    INC SI
    INC DI
    ADD BX, 02
    DEC CX
    JNZ NEXT } Loop NEXT
    MOV AH, 4CH
    INT 21H

```

CODE ENDS

END START

(II) ~~BCD to Binary Conversion~~

~~MOV BX, 1100~~ MSD x OA + LSD

~~MOV AL, [BX]~~

~~MOV DL, AL~~

~~AND DL, OF~~ ; Now DL contain LSD

~~AND AL, OFO~~

~~MOV CL, 04~~

~~ROR AL, CL~~ ; Now AL contain MSD

~~MOV DH, OA~~

~~MUL DH~~ ; $(AX) \leftarrow (AL) \times (DH)$

~~ADD AL, DL~~

Mar Baselios College of Engineering & Technology

MOV [BX+□], AL

Page #... 54

Title: MACRO & PROCEDURESMacro programming

- ① To display the following strings one below the other on the screen.

HELLO WORLD

MACRO PROGRAMMING,

IS A WONDERFUL PROGRAMMING!

DISP_STR MACRO MSG1

MOV AH, 09H

MOV DX, OFFSET MSGQ

INT 21H

ENDM

DATA SEGMENT

CR EQU 0DH ; CR moves the cursor to the first column

LF EQU 0AH ; LF moves the cursor to the next line
below

MSG1 DB 'HELLO WORLD', LF, CR, '\$'

MSG2 DB 'MACRO PROGRAMMING', LF, CR, '\$'

MSG3 DB 'IS A WONDERFUL PROGRAMMING\$', '\$'

DATA ENDS

CODE SEGMENT

ASSUME CS: CODE, DS: DATA

START: MOV AX, DATA

MOV DS, AX

DISP_STR MSG1

DISP_STR MSG2

DISP_STR MSG3

Mar Baselios College of Engineering & Technology

CODE ENDS

END START

② Passing parameters To & From Procedures
using Registers

Two digit BCD to Binary Conversion

DATA SEGMENT

BCD-INPUT DB 89H

BIN-VALUE DB DUP(0)

DATA ENDS

STACK SEGMENT STACK

DW 100 DUP(0)

TOP_STACK LABEL WORD

STACK SEG ENDS

CODE SEGMENT

ASSUME CS: CODE, DS: DATA, SS: STACK SEG

START: MOV AX, DATA

MOV DS, AX

MOV AX, STACK SEG

MOV SS, AX

MOV SP, OFFSET TOP_STACK

MOV AL, BCD-INPUT

CALL BCD-BIN ; procedure call

MOV BN-VALUE, AL

; Procedure starts

BCD-BIN PROC NEAR
PUSHF

; does the conversion now

MOV BL, AL
AND BL, 0FH
AND AL, 0FOH
MOV CL, 04H
ROR AL, CL
MOV BH, OAH
MUL BH
ADD AL, BL

POPF
RET

BCD-BIN ENDP

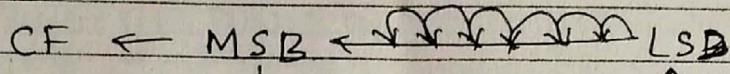
CODE ENDS
END START

C) Rotate Instruction

(C.1) ROL (Rotate All Bits of Operand Left, MSB to LSB)

ROL Destination, Count

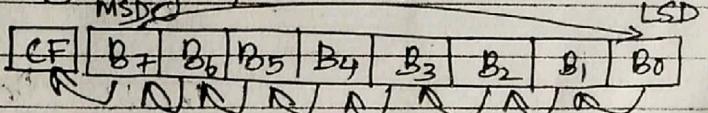
mem loc reg



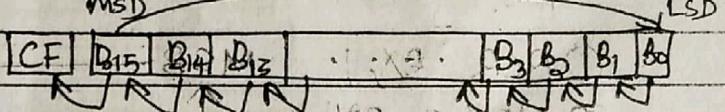
Note:

- ① ROL AX, 1
- ② MOV CL, 04H
ROL BL, CL

a) reg 8 / mem 8

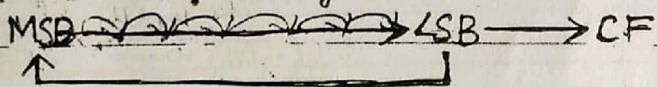


b) reg 16 / mem 16



(C.2) ROR (Rotate All Bits of Operand Right, LSB to MSB)

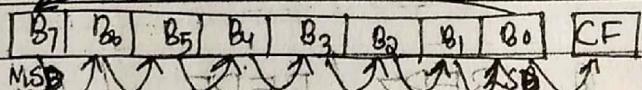
ROR Destination, Count
mem loc reg



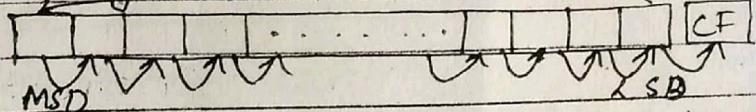
Note:

- ① ROR BH, 1
- ② MOV CL, 08H
ROR AL, CL

a) reg 8 / mem 8



b) reg 16 / mem 16



(C.3) RCL (Rotate Operand Around to the left through)

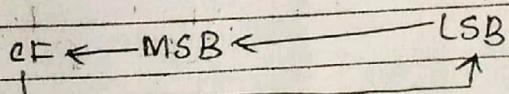
RCL Destination, Count

mem loc reg

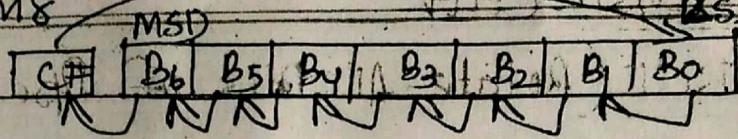
Note: ① RCL DX, 1

② MOV CL, 4

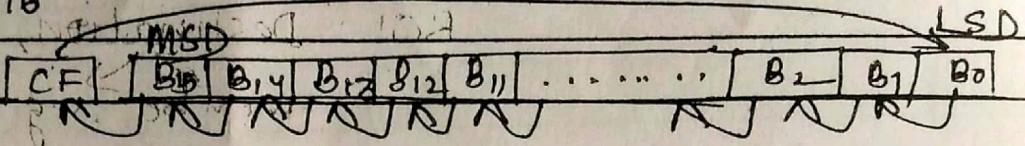
RCL AX, CL



② reg8/mem8



⑥ ~~reg16/mem16~~



(C4) RCR (Rotate Open and Around to the Right through CF

RCR Destination, Count
Memloc Reg

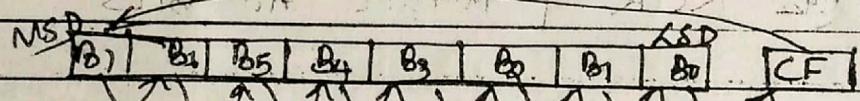
Note :-

① RCR BX,1 MSB → LSB → CF

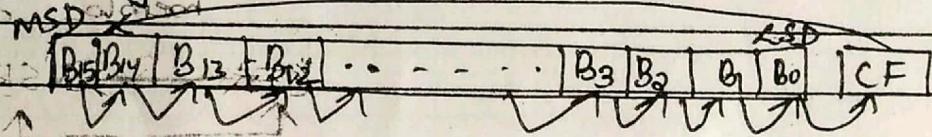
⑤ MOV CL,02H

RCR BX, CL

a) seg8 / mem8



⑥ segment members



STRING INSTRUCTION

a) REPZ / REPE (Repeat if Zero / Repeat if Equal)

(a)

Binary to BCD Conversion

```
MOV BX, 2000  
MOV AL, [BX]  
MOV DX, 0000
```

```
HUND: CMP AL, 64  
JC TEN  
SUB AL, 64  
INC DL  
JMP HUND
```

```
TEN: CMP AL, 0A  
JC UNIT  
SUB AL, 0A  
INC DH  
JMP TEN
```

```
UNIT: MOV CL, 04  
ROR DH, CL  
ADD AL, DH  
MOV [BX+1], AL  
MOV [BX+2], DL  
HLT
```

} UNIT: MOV [BX+1], AL
MOV [BX+2], DH
MOV [BX+3], DL

HLT

lecture No: 13, 14 INTERRUPT

INTERRUPT

The process of interrupting the normal program execution to carry out a specific task / work.

How it's works

Processor receives an interrupt signal



Stops execution of current normal pgs



Save the status of the IP, CS, flag reg. in stack



then Processor executes a subroutine / procedure in order to perform the specific task / work by the interrupt.



At the end of ISR, the stored status of Regs in stack are restored to respective Regs

execute
response
as interr

Interrupt
Service
Subroutine
(ISR)

Processor Restart the Normal pgs execution.

SOURCES OF INTERRUPT IN 8086

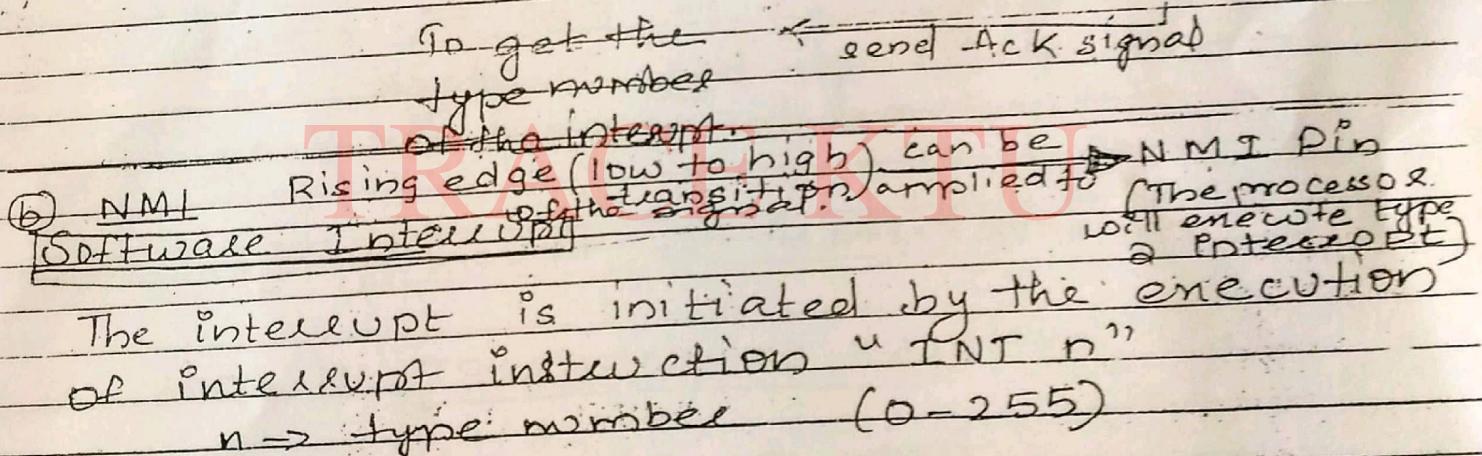
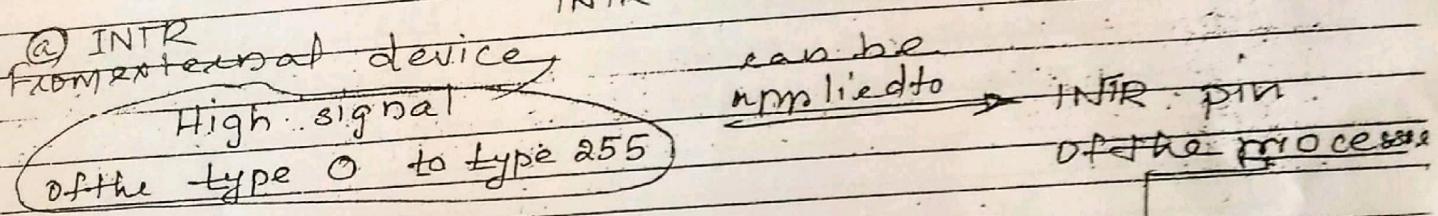
- ✓ From an external signal applied to NMI or INTR pins of the processor
- ✓ Execution of the interrupt instruction "INT n" n → type number (0 - 255)
- ✓ From conditional interrupts produced in the 8086 by the execution of an instruction:
eg: Type-0 (Divide by zero interrupt)

CLASSIFICATION OF INTERRUPTS

1. Hardware and software interrupts
2. Vectored and non-vectored interrupts
3. Maskable and non-maskable interrupts.

① Hardware Interrupt

The interrupt initiated by the external signal applied to the interrupt pins of the processor.



The interrupt is initiated by the execution of interrupt instruction "INT n"
n → type number (0-255)

(8086 processor has 256 types of s/w interrupts)

② Vectored and Non-Vectored Interrupts.

③ Vectored Interrupt

When an interrupt signal is accepted by the processor, if the program control automatically branches to a specific address (vector address), then the interrupt is called V.I.

⑥ Non-vectored Interrupts

In N-V Interrupts, the interrupting device should supply the address of the ISR to be executed in response to the interrupt.

③ Maskable & Non-Maskable Interrupts

① Maskable Interrupts

The Interrupts whose request can be either accepted or rejected by the processor.

e.g.: Interrupt initiated through INTR pin.

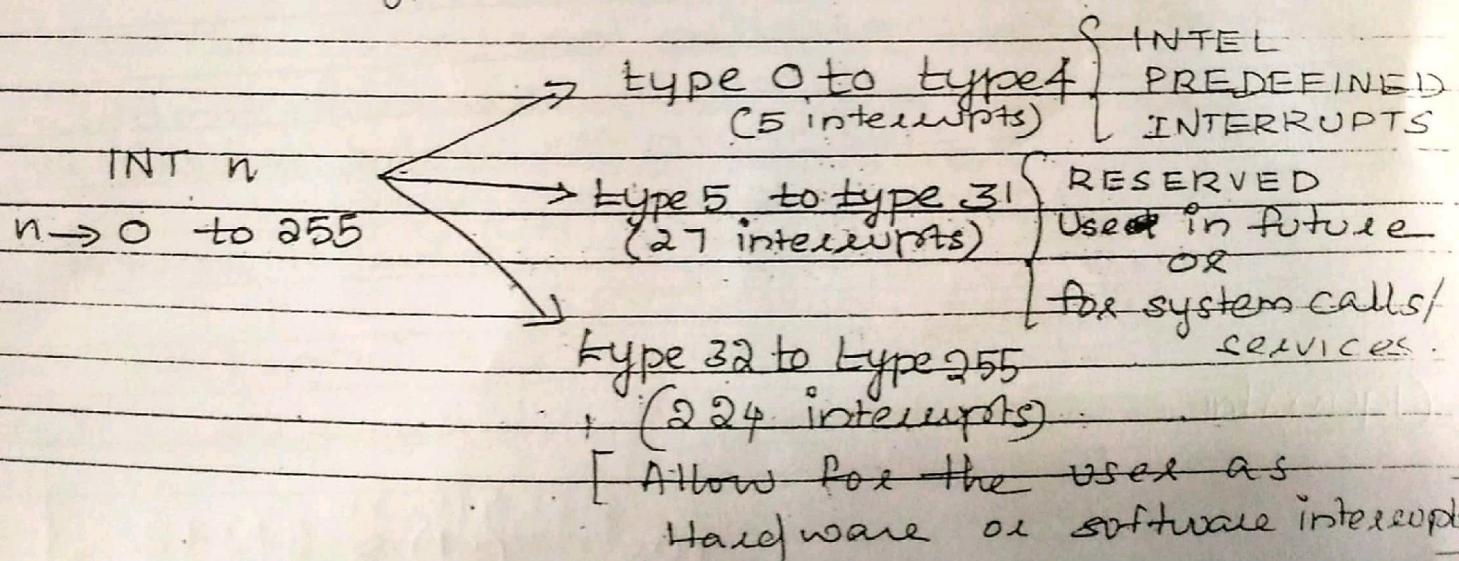
② Non-Maskable Interrupts

The Interrupt whose request has to be definitely accepted by the processor.

e.g.: Interrupt initiated through NMI pin and all software interrupts "INT n"
 $n \rightarrow 0$ to 255.

INTERRUPTS OF 8086

In 8086, 256 types of Interrupts



INTERRUPT VECTOR TABLE

s/m designer has to create an IVT in the fixed 1kb memory space [addr range $00000H$ to $0D3FFH$]
 vector address \downarrow

SERVICING AN INTERRUPT BY 8086

8086 Processor receives an interrupt signal

\downarrow
 Stop Execution of current normal program

\downarrow
 Save the status of the IP, CS, flags in stack

$(SP) \leftarrow (SP) + 2$; TOS \leftarrow (flag reg)
 $(SP) \leftarrow (SP) + 2$; TOS \leftarrow (CS)
 $(SP) \leftarrow (SP) + 2$; TOS \leftarrow (IP)

$IF \leftarrow 0$ $TF \leftarrow 0$

\downarrow
 Then processor execute a subroutine / procedure
 in order to perform the specific task by the interrupt.

\therefore M.A (ISR) = new IP value + new CS value $\times 10H$

New IP value (Offset add)

Multiplying the type number specified in the instruction by 4:

eg: INT 8

$$\Rightarrow 4 \times 8 = 32_{10} = 20H$$

\therefore New IP value will be read from add $00020H$

(vector address)

New CS value (Base add)

Multiplying the type number by 4 and plus 2.

eg: INT 8

$$\Rightarrow 4 \times 8 = 32 + 2 = 34_{10} = 22H$$

\therefore New CS value will be read from add $00022H$

(vector address)

At the end of ISR, the stored status of Regs in stack are restored to respective Regs.

Pop operations

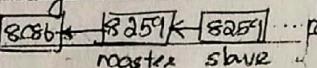
$(IP) \leftarrow TOS$	$(SP) \leftarrow (SP) + 2$
$(CS) \leftarrow TOS$	$(SP) \leftarrow (SP) + 2$
$(Flag \ reg) \leftarrow TOS$	$(SP) \leftarrow (SP) + 2$

Processor Restart
Normal
Program execution

PROGRAMMABLE INTERRUPT CONTROLLER

INTEL 8259

Features

- 8259 is a programmable interrupt controller, used to expand the interrupts of 8085 or 8086 processor.
- One 8259 accept 8 interrupt request and allow one by one to the processor INTR pin.
- 8259 used in cascade mode in a system to expand the interrupts upto 64. 
- 8086 processor based system, it supply the type number of the interrupt. The type number is programmable.
- The priorities of the interrupts are programmable.

Operating Modes decides the priorities① Automatic RotationIR_x → lowest priority
(Int request being serviced)IR_{x+1} → highest priorityIR₃ - being serviced
(lowest priority)eg: IR₀ IR₁ IR₂ IR₃ IR₄ IR₅ IR₆ IR₇
4 5 6 7 0 1 2 3② Fully Nested ModeIR_x → IR_{*}IR_x → highest priority(Any IR can be assigned
the highest priority)eg:
IR₀ IR₁ IR₂ IR₃ IR₄ IR₅ IR₆ IR₇
5 6 7 0 1 2 3 4IR_{x-1} → lowest priority③ Specific Rotation Mode

Similar to the automatic rotation mode

except the user can select any IR for lowest priority, thus fixing all other priorities.

IR₀ - IR₇ (INTERRUPT REQUEST)

→ The interrupts can be masked or unmasked individually.

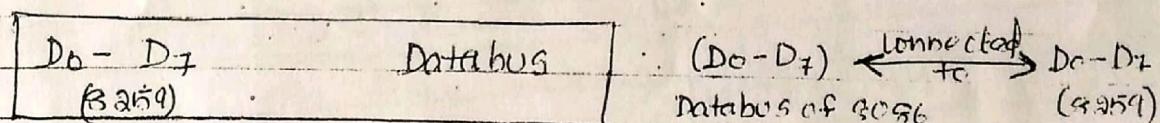
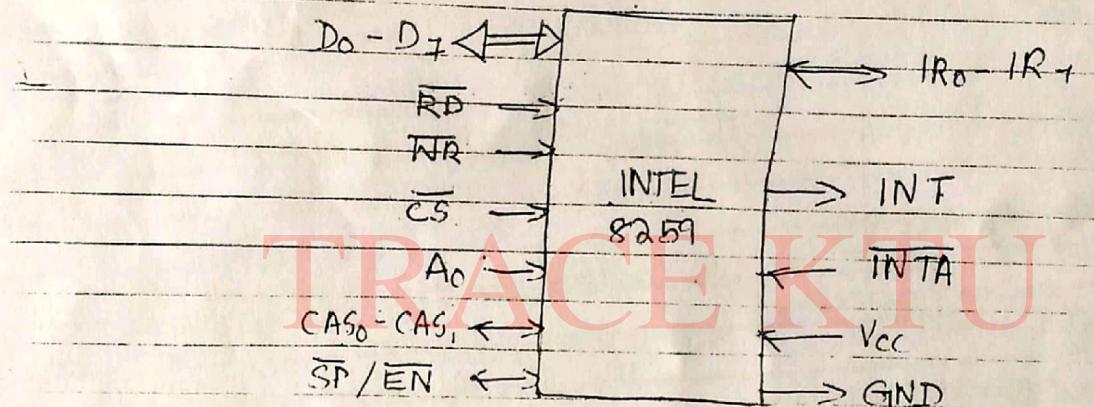
(disabled)

(enabled)

→ 8259 is programmed to accept either level triggered or edge triggered interrupt signal.

→ 8259 provides the status of pending interrupt, marked interrupts and interrupt being serviced.

→ 8259 should be programmed by sending Initialization Command Word (ICW) and Operational Command Word (OCW).



RD

Read control signal

WR

Write control signal

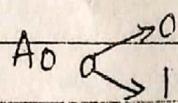
CS

Chip select signal (When CS goes low, 8259 is enabled)
(8259 requires 1 chip select signal and generated by using 2-to-8 decoder)

3-to-8 decoder

i/p address lines → A₀, A₁, A₂

Decoder enable signals → A₀, M/IO



Internal Address lines.
2_N Address lines.

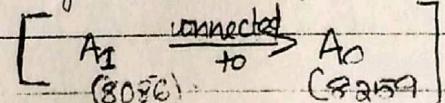
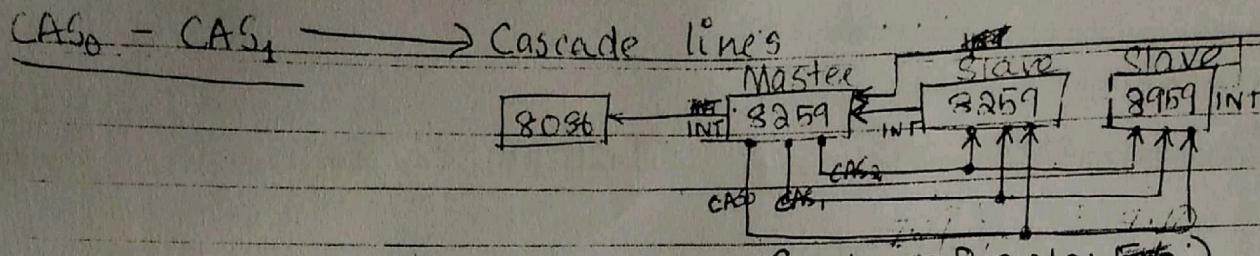


fig No: 54 page No: 182



ST/EN → Slave Programs / Enable buffer
(i/p or o/p signal)

(Page No: 185, fig No: 55)

Access
In next buffered Mode Buffered Mode
i/p signal o/p signal

ST/EN ↑ (Master) Used to disable the
 ↓ (Slave) data buffer while data
 is transferred from
 8259 to the CPU.

~~Page No. 5.4
I/O EXplanation~~
Working of 8259 with 8086 Processor
(Processing of interrupt by 8259)
First 8259 programmed by sending ICW and OCW

Ref 1.
page No. 249 Check its priority & check whether it
is masked or not

High priority & unmasked, then it is serviced

8259 → Send INT signal → INTR pin of (8086)

Indicate Acceptance INTA

To get type number of interrupt → INTA (Do-DP datibus)
Type number getting type number in 8086

Using type number, calculate vector add of Memory add of the ISR and execute ISR

8086 → command word → 8259
AECI → OCW2 (to inform 8259 abt the end of Interrupt)
IR₀ - IR₇ (INTERRUPT REQUEST)

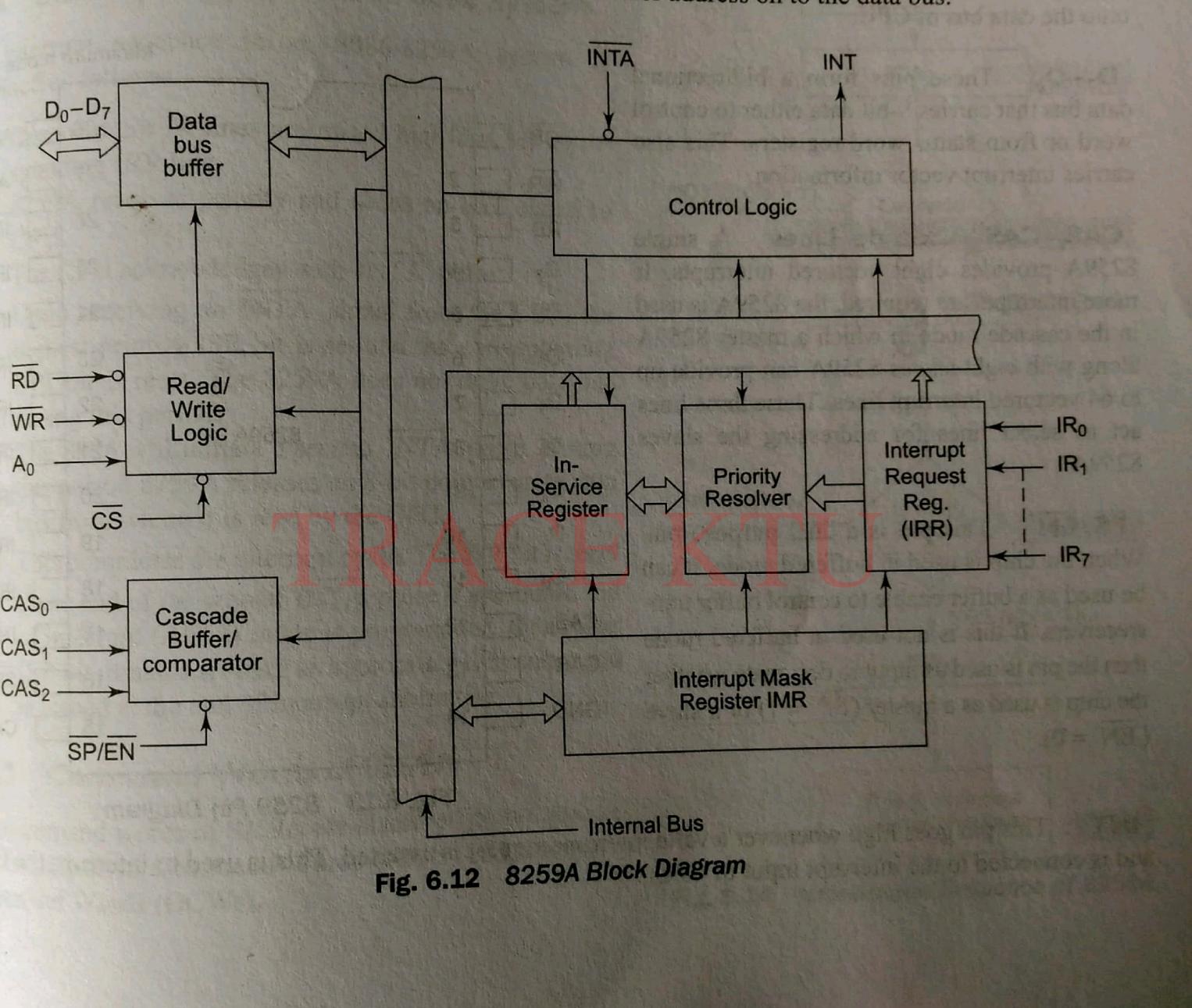


Fig. 6.12 8259A Block Diagram

Functional Block Diagram of 8259

{Page No. 184}
fig: 8259

8 Functional blocks.

(1) DATA BUS BUFFER

- Processor send ^(command) control word to data bus buffer through D₀-D₇.
- Processor read status word from data bus buffer through D₀-D₇.
- From the data bus buffer, the 8259 send type number of the interrupt to the processor through D₀-D₇.

(2) Read / Write logic

(internal addline)

- The processor uses RD, WR and A₀=0, to write a command word or read a status word.
→ 8259 is selected by CS.

(3) Cascade Buffer / Comparator

- Used to expand the interrupts of 8259.

- In Cascade connection,
~~one~~ 8259 directly interrupting 8086

master 8259 SP/EN → high.

8259s interrupting the master 8259

slave 8259s. SP/EN → low.

- Cascade pins (CAS₀, CAS₁, CAS₂) from the master are connected to the corresponding pins of the slave. (^{input} pins, ^{output} pins)

(4) Interrupt Request Register (IRR)

IRR is used to store information about all the interrupt requests, which are requesting for interrupt service.

If an interrupt input is requesting an interrupt service, then the corresponding bit in the IRR is set.

⑤ In service Register (ISR)

Used to store information about the interrupt currently being serviced and the voices pending bit in ISR is set.

⑥ Interrupt Mask Register

(1)

→ Used to disable (mask) or enable(unmask) individual interrupt request.

→ This register has a specific bit for each interrupt request.

⑦ Priority Resolver

It checks 3 Register.

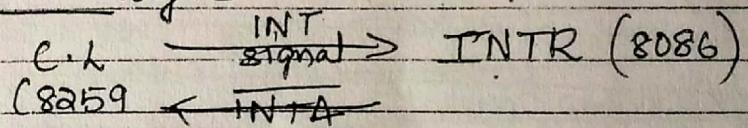
IRR checked for Interrupt Request

IMR checked for Masking Bits

ISR checked for interrupt being served.

and also determines the priority of the bit set in the IRR.

⑧ Control Logic



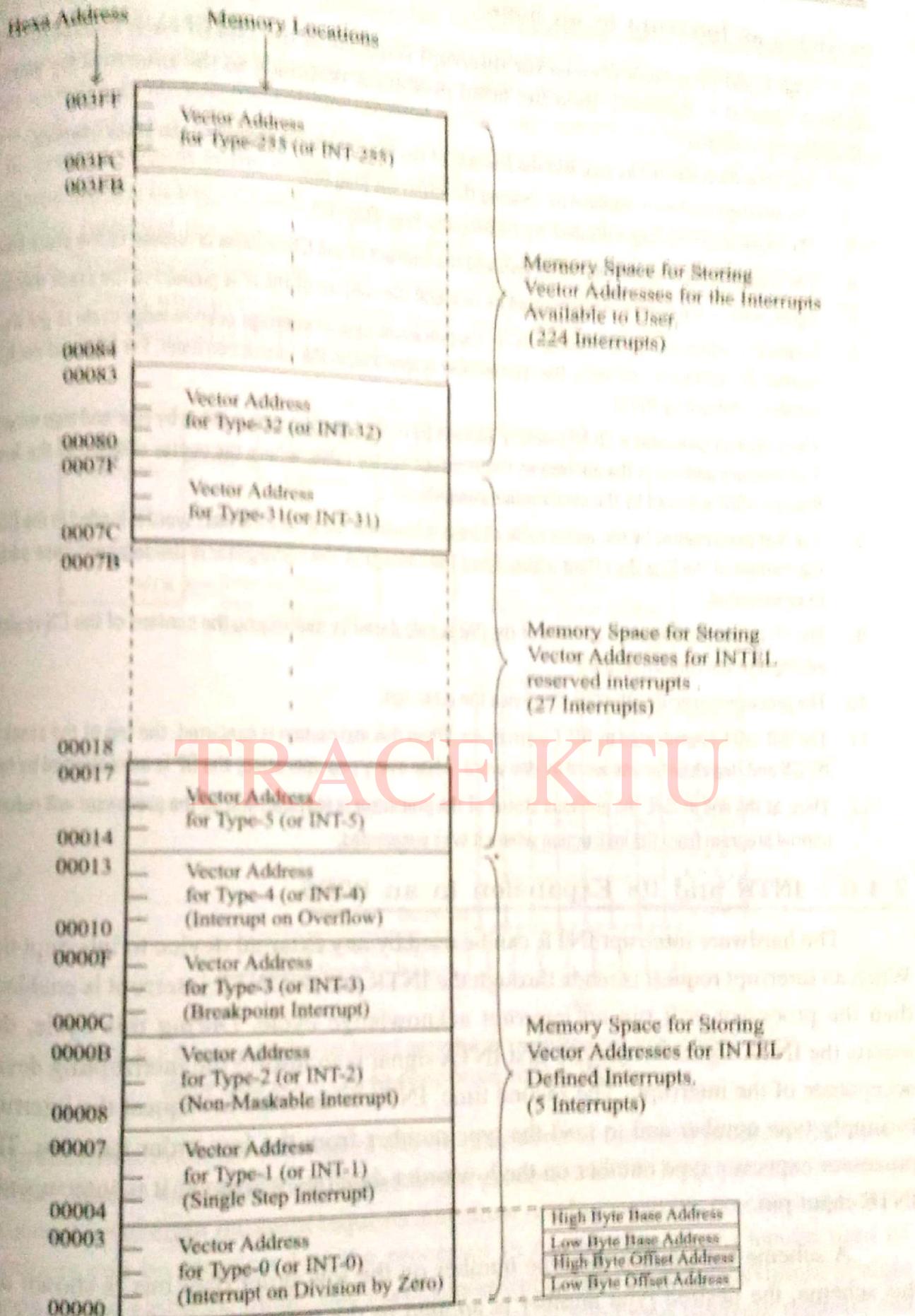


Fig. 7.4 : Organization of an interrupt vector table in 8086.

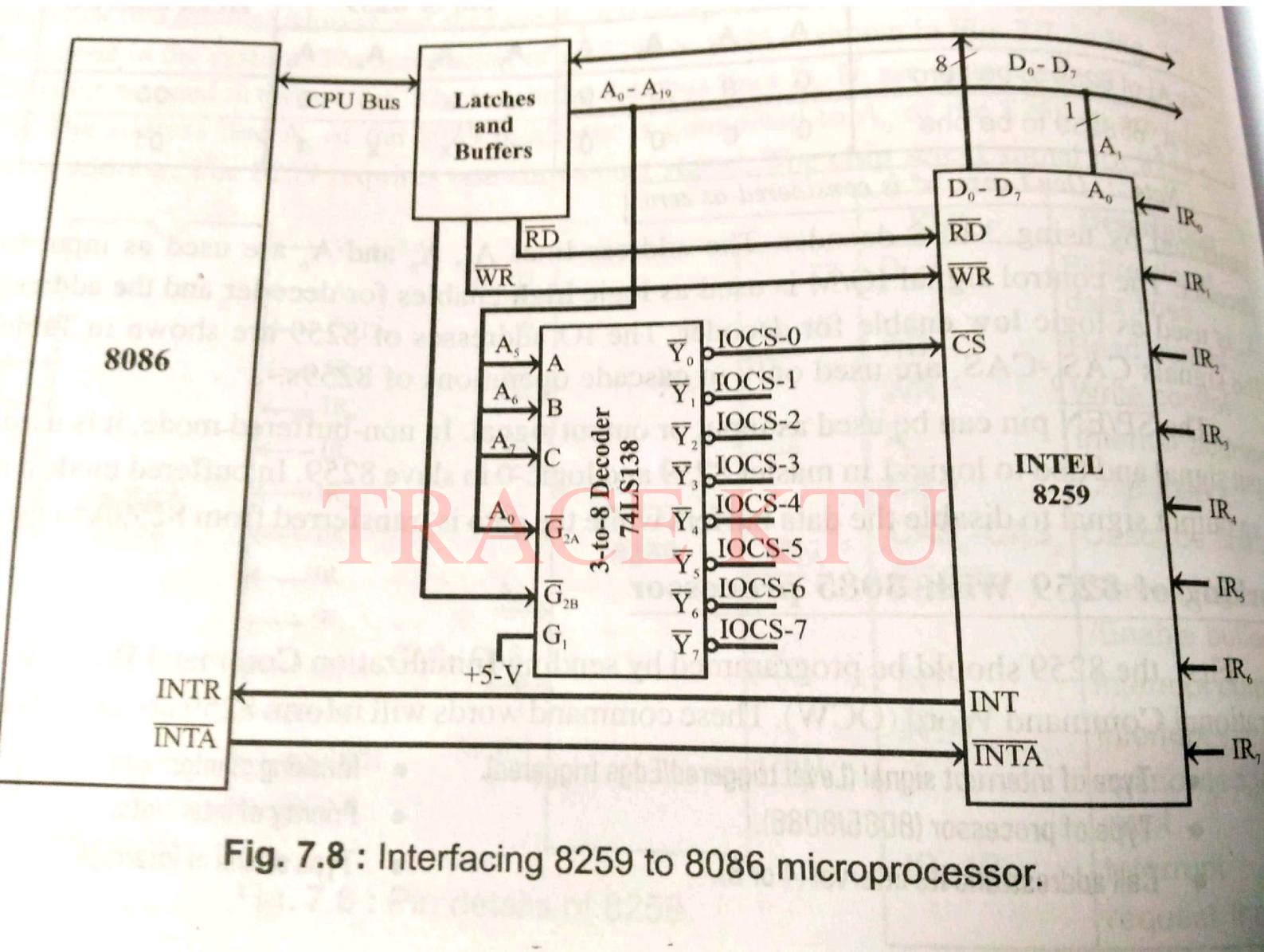


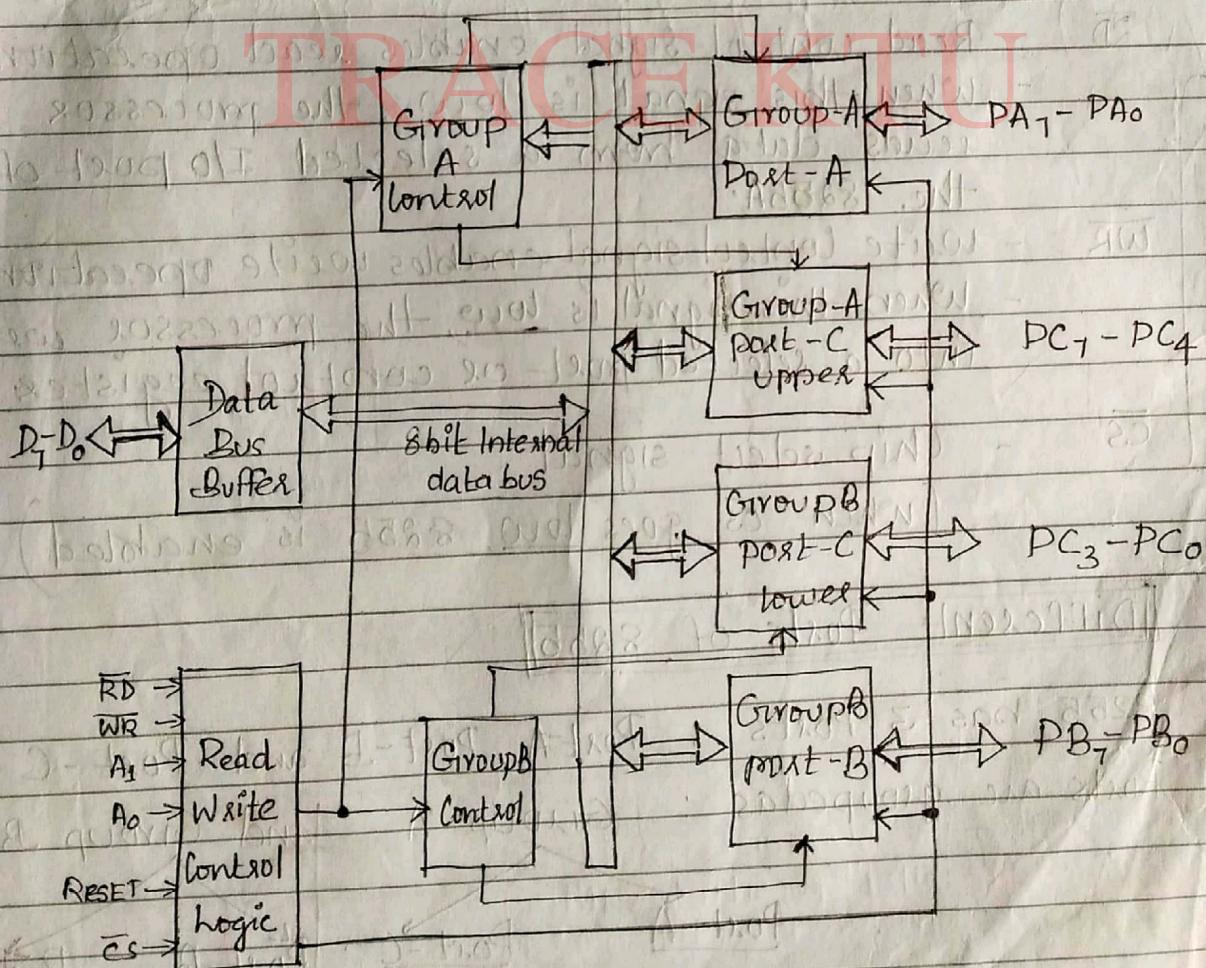
Fig. 7.8 : Interfacing 8259 to 8086 microprocessor.

PROGRAMMABLE PERIPHERAL INTERFACE - INTEL 8255

INTEL 8255 is used to interface a slow I/O device to the fast processor and to implement parallel data transfer between processor and slow peripheral devices like ADC, DAC, keyboard, 7-segment display, LCD etc...

	Pin	Description
D ₀ -D ₇	D ₀ -D ₇	Data lines
RD	RESET	Reset input
WR	CS	Chip select
A ₁	RD	Read control
A ₀	WR	Write control
RESET	A ₀ , A ₁	Internal address
CS	PA ₇ -PA ₀	Port-A pins
(+5V) V _{CC}	PB ₇ -PB ₀	Port-B pins
(0V) GND	PC ₇ -PC ₀	Port-C pins

Pin Description



Internal block diagram of 8255

* Data Bus Buffer

→ 8 data lines can be used for communication with processor.

→ Control words, status words are written into 8255 registers through data bus buffer.

→ Also write data to port, through data bus buffer via $D_0 - D_7$.

* Read/Write logic

The processor uses RD, WR and internal address lines $A_0 \& A_1$, to "write control words into control registers" and data to ports. & also "read status words from status register and data from port."

Internal address lines $A_0 \& A_1 \Rightarrow$ select any one of the

A_1	A_0	Device Selected	4 internal devices
0	0	Port A	
0	1	Port B	
1	0	Port C	
1	1	Control Registers	

RD - Read control signal enables read operation.
- When this signal is low, the processor reads data from a selected I/O port of the 8255A.

WR - Write control signal enables write operation.
- When this signal is low, the processor writes into a selected port or control register.

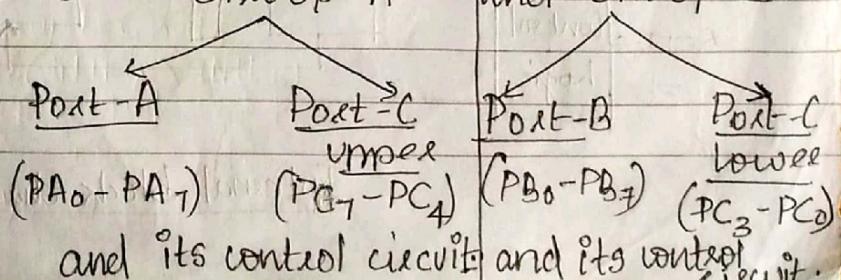
CS - Chip select signal

(When CS goes low, 8255 is enabled)

* Different Ports of 8255

8255 has 3 Ports : Port-A, Port-B and Port-C.

Ports are grouped as : Group A and Group B



✓ 3 Operating Modes: Mode-0 → Simple I/O port.

Mode-1 → Handshake I/O port.

Mode-2 → Bidirectional I/O port.

	<u>Port A</u>	<u>Port B</u>	<u>Port C</u>
<u>Act as</u>	8-bit input or output parallel port	8bit i/p or o/p parallel port	a) 8 bit i/p or o/p parallel port b) 2 nos. of 4 bit i/p & o/p parallel port c) individual pins can be set or reset for various applications
<u>Programmed to work in</u>	Any one of the 3 operating modes.	either in mode-0 or mode-1	with respect to the operating modes of Port-A and Port-B.

- ✓ If Port-A and B are programmed in Mode-0, then the Port-C can perform any 1 of the following function
- As 8 bit i/p or o/p parallel port in Mode-0.
 - As two nos. of 4-bit i/p or o/p parallel port in Mode-0.
 - Individual pins of port-C can be set or reset for various control applications.

✓ If Port-A is programmed in Mode-1/mode-2 as Port-B in Mode-1,

then ✓ some of the pins of port-C $\xrightarrow{\text{used for}}$ handshake signals

✓ Remaining pins $\xrightarrow{\text{used for}}$ individually set/reset for control applications.

	Mode-0	Mode-1	Mode-2
Programmed as	Input or Output port (all 3 ports)	Input or Output port (Only Port-A and Port-B)	Bidirectional* port (Port-A Only)

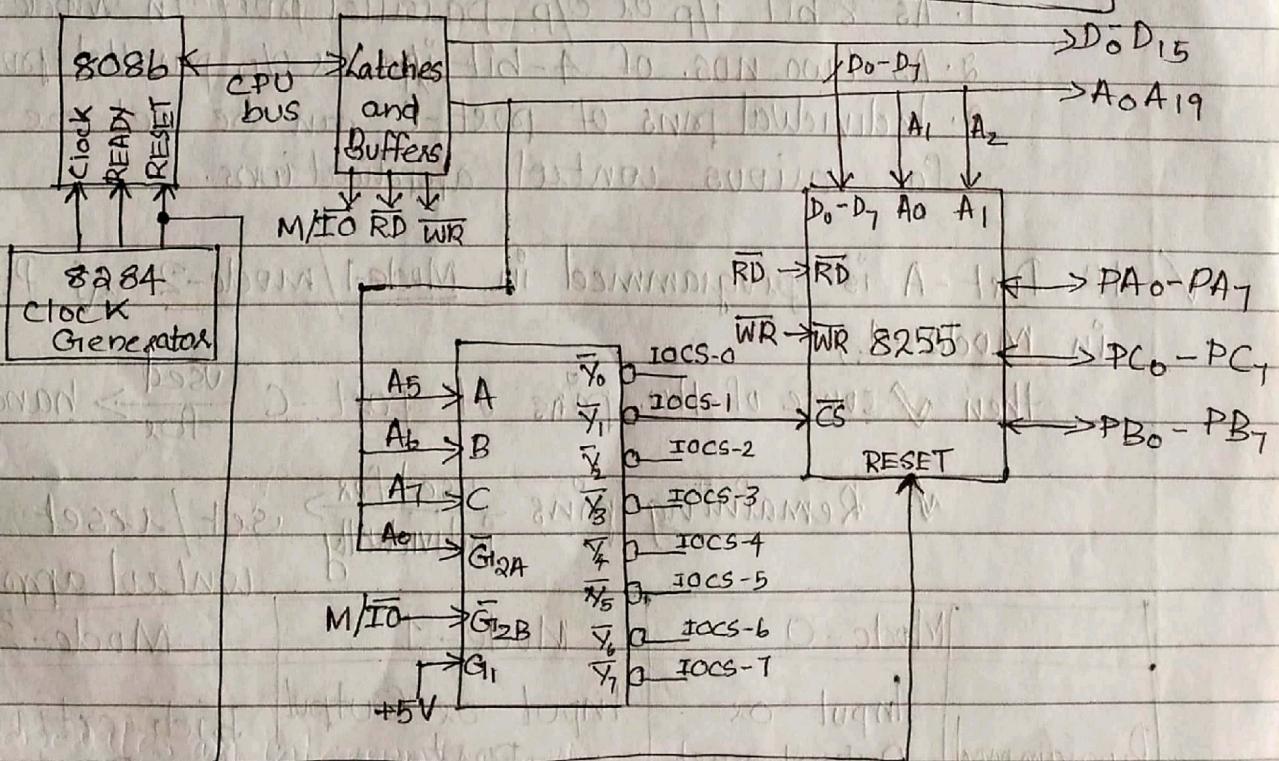
* Bidirectional port → The processor can perform both read & write operations with an I/O device connected to a port.

	Mode - 0	Mode - 1	Mode - 2
O/p data → latched I/p data → Not latched		Both I/p and O/p data are latched	
Uses / Application.	→ No Handshake & Interrupt Capability. → Used to Interface DIP switches, Hexa keypad, LEDs and 7-segment LEDs to the Processor.	Handshake* (Port-C) Interrupt** Capability.	→ Handshake capability (5 pins of Port C) → Data transfer between two computers or floppy disk controller interface

* Handshake signals are exchanged between the processor and peripheral devices to data transfer.

** Interrupt driven data transfer scheme.

Interfacing of 8255 with 8086 processor



→ 8255 can be either memory mapped or I/O mapped in the system.

→ 3-to-8 decoder

3-Decodes

Input lines → A₅ A₆ A₇ address lines of 8086.

A₀ and M/I_O → Used to enable 3-to-8 decodes.

Decoder Output lines → Chip Select Signal (CS)
(IOCS-1, ..., IOCS-7) (IOCS-1) used to select 8255.
O/P line

Internal Device	Binary Address							
	A ₁	A ₆	A ₅	A ₄	A ₃	A ₂	A ₁	A ₀
Port - A	0	0	1	x	x	0	0	0
Port - B	0	0	1	x	x	0	1	0
Port - C	0	0	1	x	x	1	0	0
Control Register	0	0	1	x	x	1	1	0

I/O Addresses of 8255

→ A₁ and A₂ of 8086 are connected to A₀ and A₁ of 8255 to provide Internal Address.

→ Data lines D₀-D₇ of 8086 are connected to D₀-D₇ of 8255.

RD and WR signals of 8255 are connected to RD and WR of the processor 8086 to achieve parallel data transfer.

→ Clock signals, supplied by the clock generator 8284.

→ Data transfer can be performed between processor and slow peripheral devices using

Interrupt driven data transfer scheme Checking the status (Status word) of 8255

Interrupt Controller 8259 has to be interfaced to 8086 & the interrupts of Port-A and Port-B should be connected to 2 IR inputs of 8259.

Programming 8255

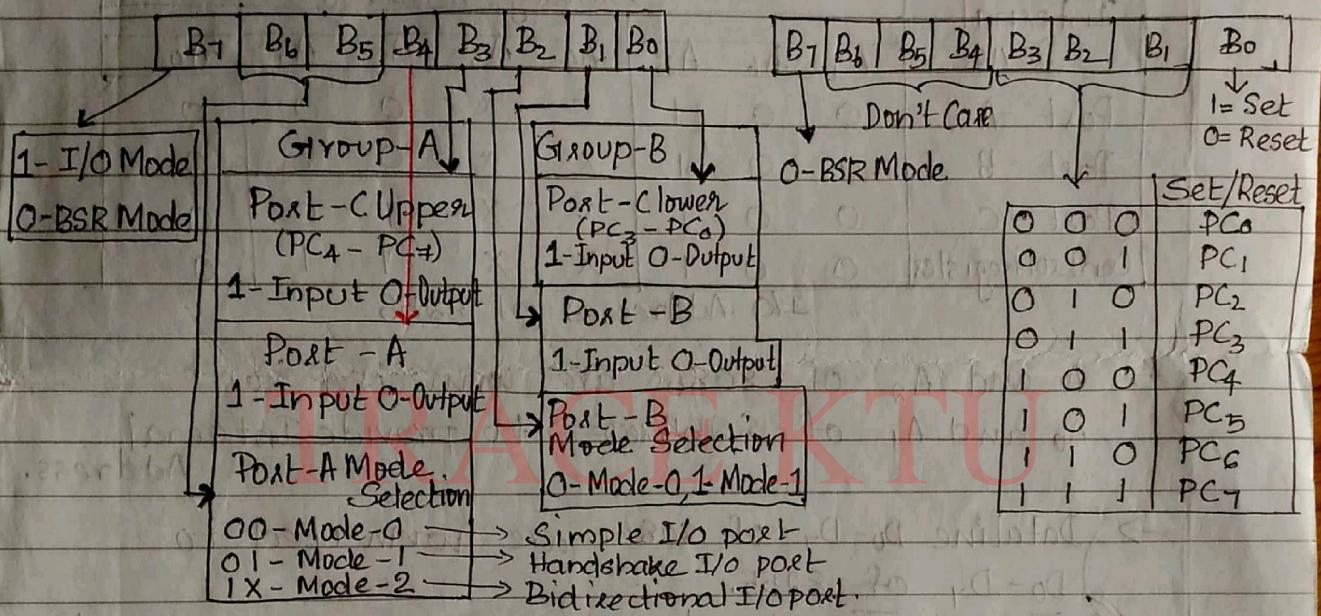
⇒ 8255 ports are programmed by writing control word in the control register.

[I/O Mode Set control Word]
(MSW)

For setting I/O functions &
Mode of operation

[Bit Set/Reset Mode
control Word]
(BSR)

For setting / resetting
individual pins of Port-C.



⇒ Handshake Mode (ie. Mode-1 & Mode-2)

Data transfer b/w the processor and the port.

Implemented via...

Interrupt Method

Checking the status of 8255

(8086 processor can check
the status of Port-A and
Port-B in the status
word)

INTEA
↓
PC₄
to high

INTEB
↓
PC₂
to low

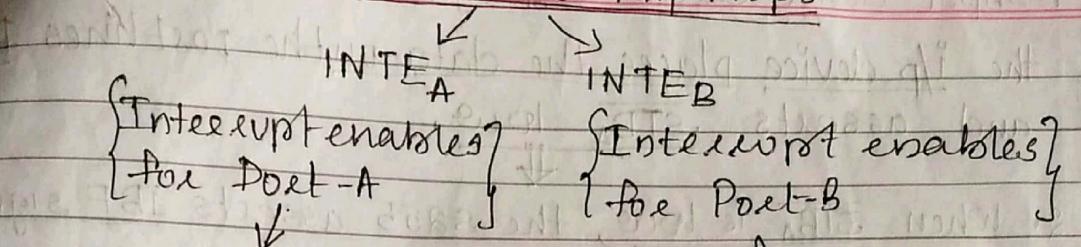
When the port is ready for
data transfer

8255 generates an "interrupt signal"
if the internal flip flops (INTEA & INTEB)
are enabled using BSR control word.

the processor
executes a read/
write cycle.

it interrupt the 8086 through
NMI/INTR pin for read/write operation.

\Rightarrow 8255 has 2 internal flip flops



* Enabled by

Setting PC₄ to high
using BSR control word.

* Enabled by

Setting PC₂ to high
using BSR control word.

* Disabled by

resetting to zero.

* Disabled by

resetting to zero.

Format of Status word

Port-C BITS

B ₇	B ₆	B ₅	B ₄	B ₃	B ₂	B ₁	B ₀
----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------

Status of Port-A
(input port)

B ₁	B ₀	B ₅	B ₄	B ₃
I/O	I/O	IBFA	INTR _A	INTR _A

Port A status

Port-B status

B ₂	B ₁	B ₀
INTEB	IBFB	INTRB

Status of Port-B
(input port)

Status of Port-A
(output port)

OBFA	INTEA	I/O	I/O	INTR _A
------	-------	-----	-----	-------------------

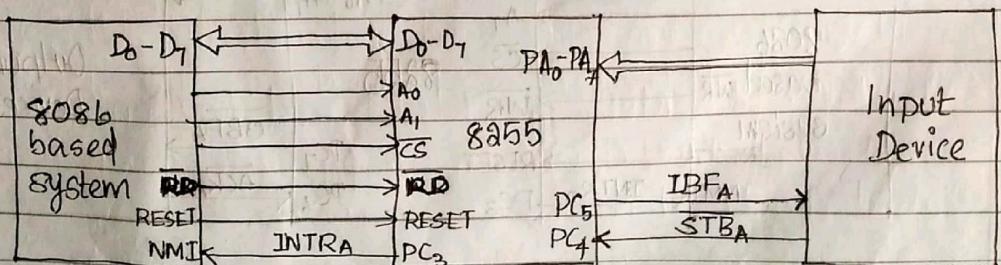
Status of Port-B
(output port)

INTED OBFB INTRB

* INTERRUPT METHOD

\Rightarrow ① 8255 Handshake Input Port (Mode-1)

Handshake signals used for "data transfer between input device and 8086 processor" using Port-A of 8255 as "handshake input port".



STB - Strobe

IBF - Input Buffer Full

OBF - Output Buffer Full

INTR - Interrupt Request

Working

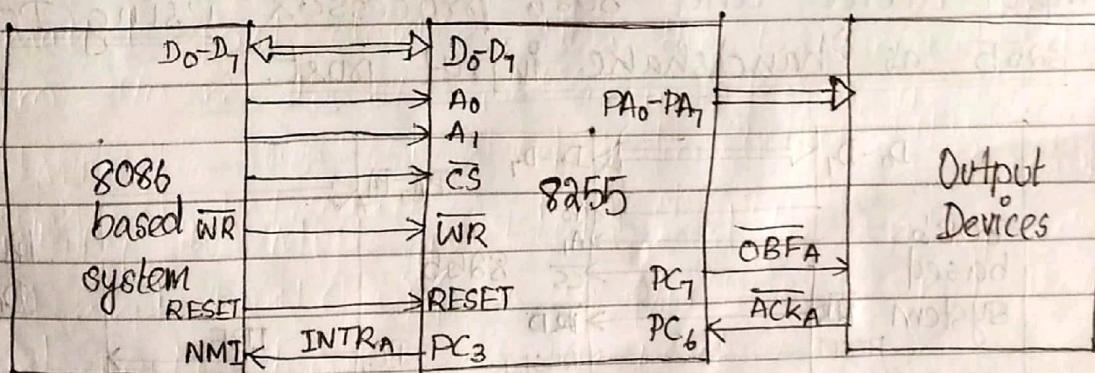
1. Input device "checks $\overline{IBF_A}$ signal," if it is low then the i/p device places the data on the portlines $\overline{PA_0}$ - $\overline{PA_7}$ and asserts $\overline{STB_A}$ low.
2. When $\overline{STB_A}$ is low, the 8255 asserts IBF signal high. After a delay time $\overline{STB_A}$ is asserted high, then the data is latched to the port and $\overline{INTR_A}$ is set high.
3. When $\overline{INTR_A}$ goes high, the processor is interrupted through NMI pin to execute a subroutine; for reading the data from the port.

4. For a read operation, the processor asserts RD low and the $\overline{INTR_A}$ is reset to low.

After read operation, the RD become high. Then the IBF is asserted low and the input device can send the next data.

⑥ 8255 Handshake Output Port (Mode-1)

Handshake signals used for "data transfer between output device and microprocessor" using "Port - A" of 8255 as handshake output port.



Working

① When the port is empty, the processor write a byte in the port.

② For write operation, the processor asserts WR low. After write operation to the port, at the rising edge of WR both INTR_A and OBFA are asserted low by the 8255.

③ When the OBFA become low, it inform that the output device that the data is ready.

④ If the o/p device accepts the data then it send an acknowledge signal by asserting ACK_A low.

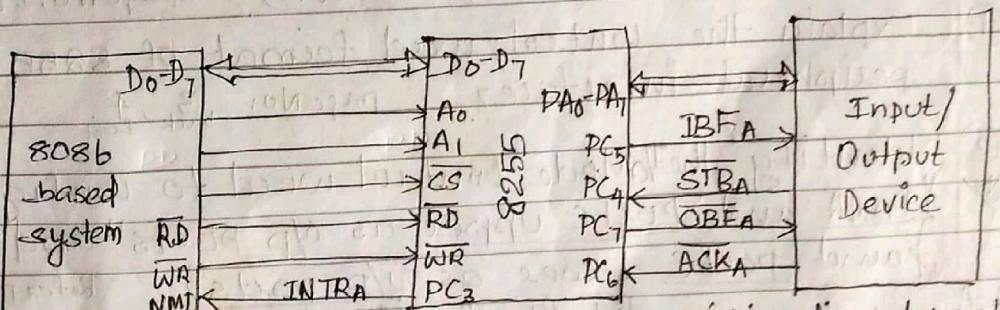
⑤ When ACK_A is low, OBFA is asserted high by the 8255.

After reached the ACK_A in the 8255, it asserted ACK_A high, then the INTR_A is set (asserted high).

⑥ When INTR_A goes high, the processor is interrupted through NMI input pin "to execute a interrupt service routine" in order to load next data in the output port.

⑦ 8255 Bidirectional port (Mode-2)

Handshake signals used for data transfer between Input/Output device and 8086 processor using Port A of 8255 as bidirectional port.



Port-A of 8255 as bidirectional port
(Mode-2)

- In Mode-2, the port A can be used either as an input port or as an output port.
- At any one time, the processor will perform either read or write operation.
- ✓ In mode-2, the read operation can be followed by write or write operation can be followed by read.
- The signals involved and the operations performed for read operation are similar to mode-1 Input port.
- The signals involved and the operations performed for write operation are similar to mode-1 Output port.

UNIVERSITY QUESTIONS

- ① What are the different modes of operations of programmable peripheral interface 8255? Page No: 2, 4, 5
 - ② Explain handshake signals used by 8255 in input and output operations? Page No: 4, 5
 - ③ With a neat diagram, explain the architecture and control word format of 8255 programmable parallel port device. How the BSR mode differs from I/O mode of operations? Page No: 1, 2, 3 (back)
 - ④ Explain the control word format of 8255 programmable peripheral interface? Page No: 3 back, 4 front
 - ⑤ Identify the Mode control word to configure Port A and Port C upper as "O/P ports" and Port B and Port C lower as "I/P ports"? Refer Page: 3 back for explanation
- Amal*
- Port-A
- Simple I/O input
- Port-B
- Handshake input
- | | | | | | | | |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
| B ₇ | B ₆ | B ₅ | B ₄ | B ₃ | B ₂ | B ₁ | B ₀ |
| 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |