KTU NOTES

The learning companion.

KTU STUDY MATERIALS | SYLLABUS | LIVE NOTIFICATIONS | SOLVED QUESTION PAPERS

Lecture No: 1 Title: Evolution of Microprocessor

Generation	Year	Technology	Processor
First Generation	b/w 1971 & 1973	PMOS P-type metallic oxide Semiconductor	4-bit
Second	1973 - 1978	NMOS N-type metallic oxide Semiconductor	8-bit
Third	1978 - 1980	HMOS Hybrid / high density metallic oxide Semiconductor	16-bit
Fourth	1980 - 1993	HCMOS High Speed Complementary metal oxide semiconductor	32-bit
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.

ie. INTEL 4004
INTEL 4040 etc... } 4-bit processors

INTEL 8008
MOSTEK5065 etc... } 8-bit processors

NATIONAL IMP/16
NATIONAL PACE } 16-bit processors

SECOND GIENERATION

- ✓ 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 s/gs.
- ✓ Ability to address large m/y 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
} 0.5 MIPS (execute 0.5 million instrs per second)

INTERSIL 6100
TOSHIBA TLCS-12 } 12-bit processor

TI TMS 9900
General Instrument CP1600 } 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 m/y 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 HMOS technology called HCMOS.

* (2^{24}) 16 Mb physical m/y space.
* (2^{40}) 1 Tb virtual m/y 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 m/y 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. & pentium processor can execute 2000 MIPS.

Question: Explain the evolution of microprocessors?

Reference

Microprocessors and microcontrollers, 2e.
A. Nagoor kani page No: 1.3 to 1.5

Title: Comparison of Microprocessor Vs Microcontroller

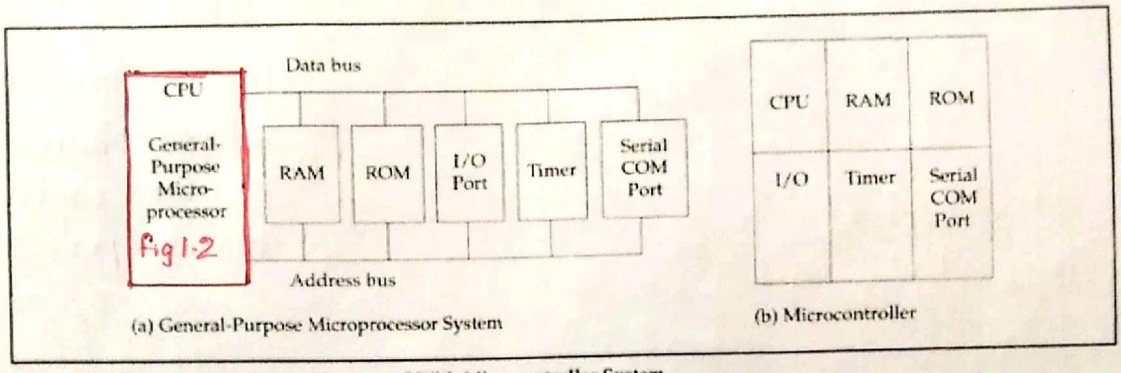


Fig 1-2

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, EPROM/EEPROM memory & ADC or DAC convertor.
* 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 m/y device external to the microprocessor.

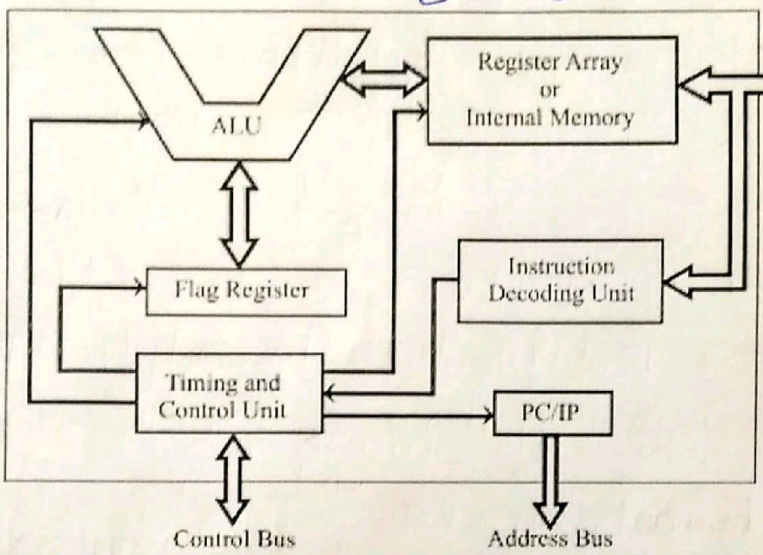


Fig. 1.2: Block diagram showing basic functional blocks of a microprocessor.

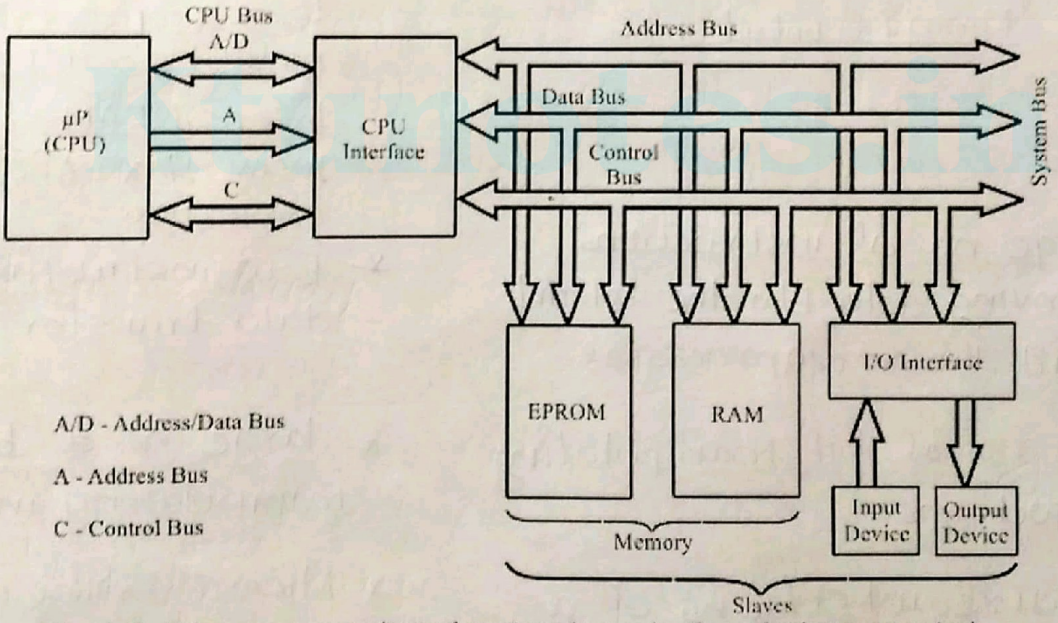


Fig. 1.3: Microprocessor-based system (organization of microcomputer).

EPROM: Store permanent pgms and data.

RAM: Store temporary pgms and data.

Input device: Used to enter the pgms, data and to operate the s/m.

Output device: Used for examing the results.

CPU Interface: The speed of IO devices doesnot match with the speed of the microprocessor. Since this is provided b/w the s/m bus & the IO devices.

Explanation of fig 1.2 Basic final blocks of a Microprocessor

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

Basic final Units

1) ALU

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

2) Flag Register

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

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 or internal m/y

- * Input data for ALU
- * Ouput 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 the ALU "what operation is to be performs".

6) Timing and Control Unit

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

eg: m/y read signal, m/y write signal
I/O read signal, I/O write signal

Explaination fig 1.3 (Microprocessor-based s/m)

In this s/m, the "microprocessor is the master" and all "other peripherals are slaves". Master 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".
ie. Data Bus, Address Bus, Control Bus.

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

Questions

1. Compare microprocessor and microcontroller?
2. Explain the basic functional blocks of a microprocessor with neat diagram.
3. Explain the microprocessor based system or organization of microcomputer with neat diagram

Reference

Microprocessors and microcontrollers, 2e.
A. Nagoor Kani Page No: 2.1, 1.5 to 1.7

Introduction to 8086 processor

- → Developed by INTEL in 1978.
- → 3rd generation of processor.
- → 20-bit Address bus
- → 2^{20} ie. 1Mb m/y 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 4 segment 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.

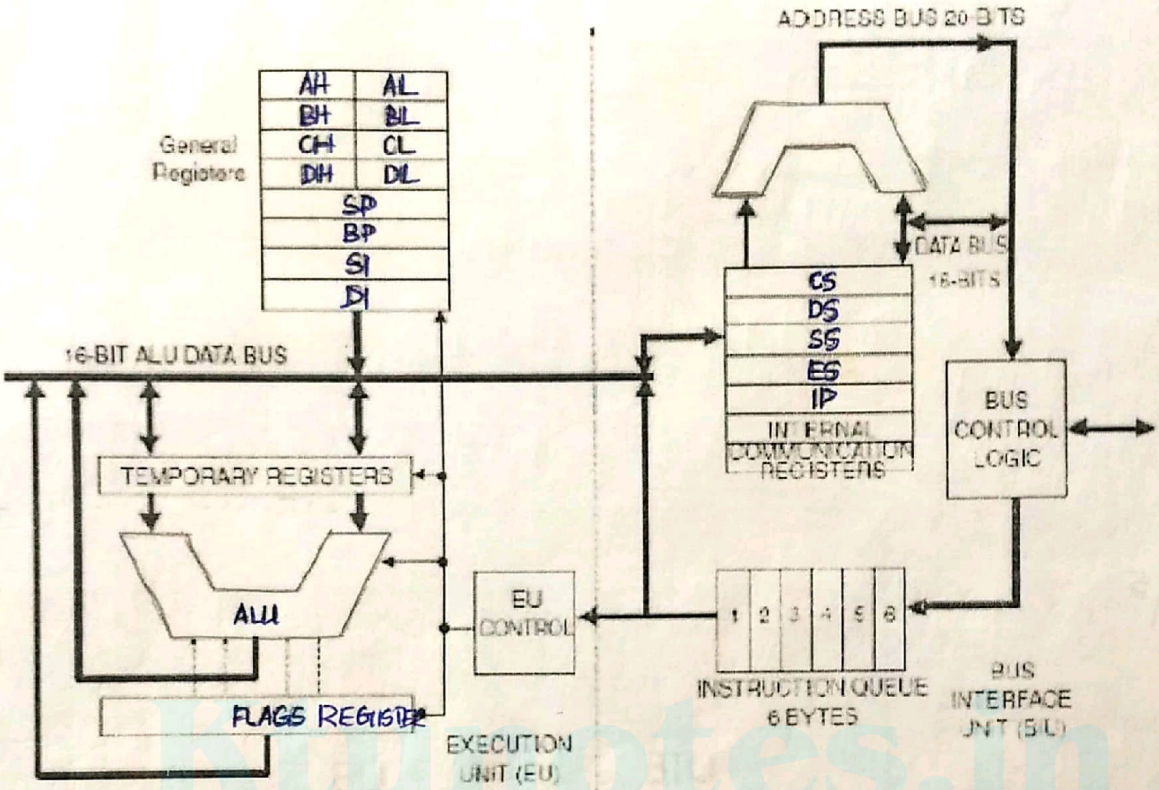
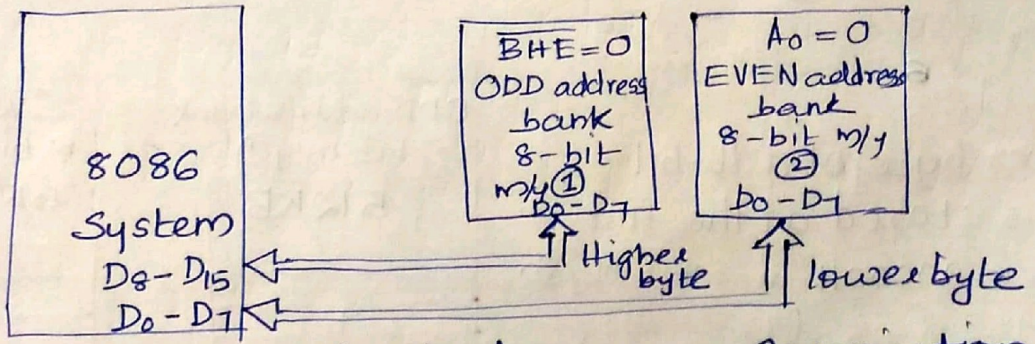


Fig 1.4 ARCHITECTURE OF 8086 MICROPROCESSOR



Physical memory Organisation

Data (16bit)

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

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

Address range of physical address : 00000 to FFFFF

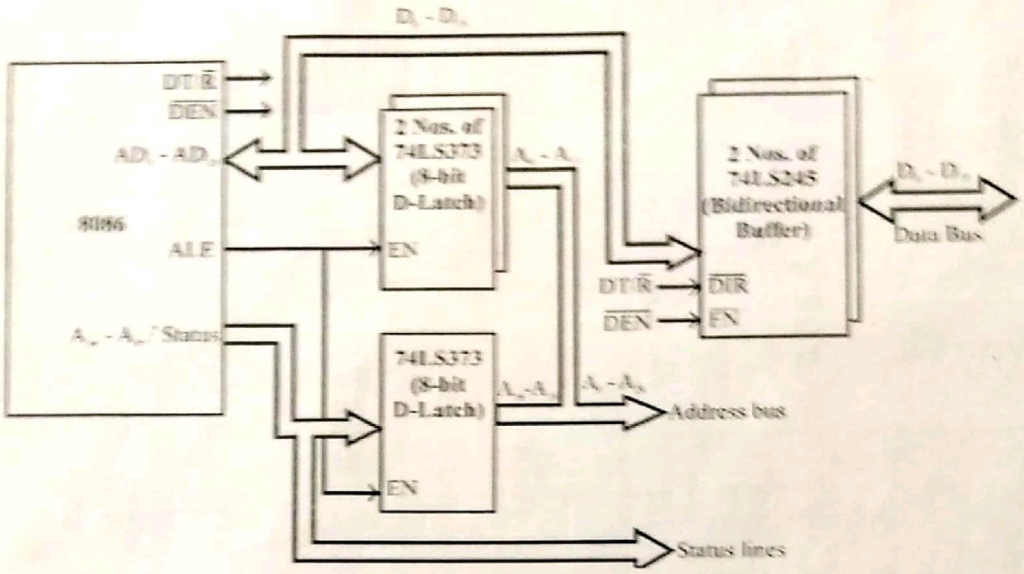


Fig. 1.5: Demultiplexing of address and data lines in an 8086 processor.

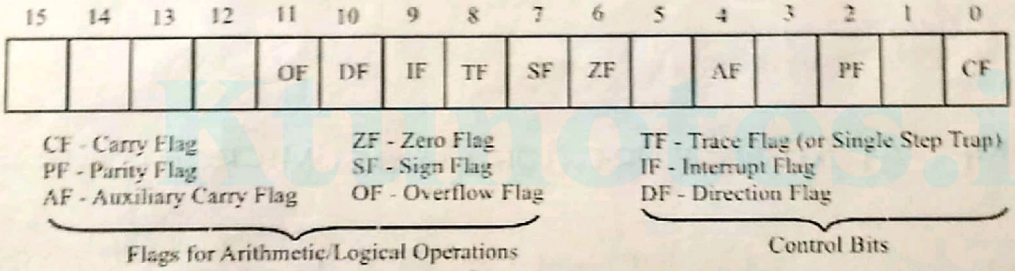
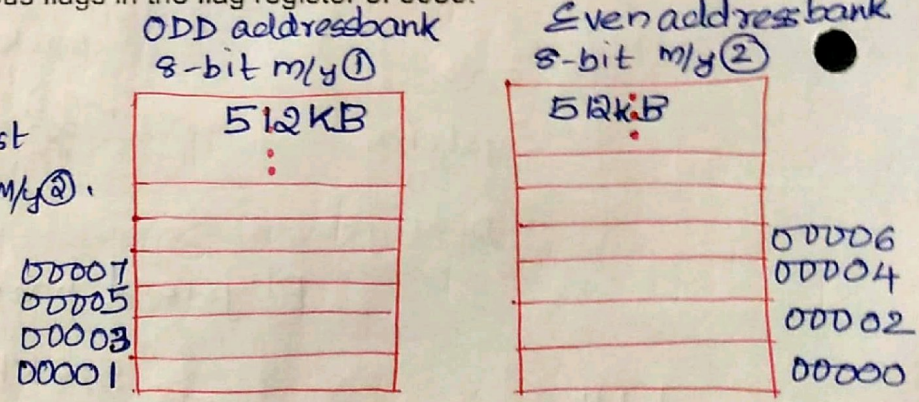


Fig. 1.6 : Bit positions of various flags in the flag register of 8086.

The lower byte of a 16-bit data is stored at the first address ie. 00000H in 8-bit m/y ②.

The higher byte of a 16-bit data is stored in the next address ie. 00001 in 8-bit m/y ①.



- ✓ When Address line 0 (A0) become low, then the even m/y bank is enabled.
- ✓ When Bus High Enable (BHE) become low, then the odd m/y bank is enabled.

Status of A0 & $\overline{BHE}$ during memory access

Memory Bank	Operand type	Status of A0	Status of BHE	Datalines used for m/y 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
Odd/Even	Word	1	0	$D_8 - D_{15}$	First cycle
		0	1	$D_0 - D_7$	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 m/y 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.

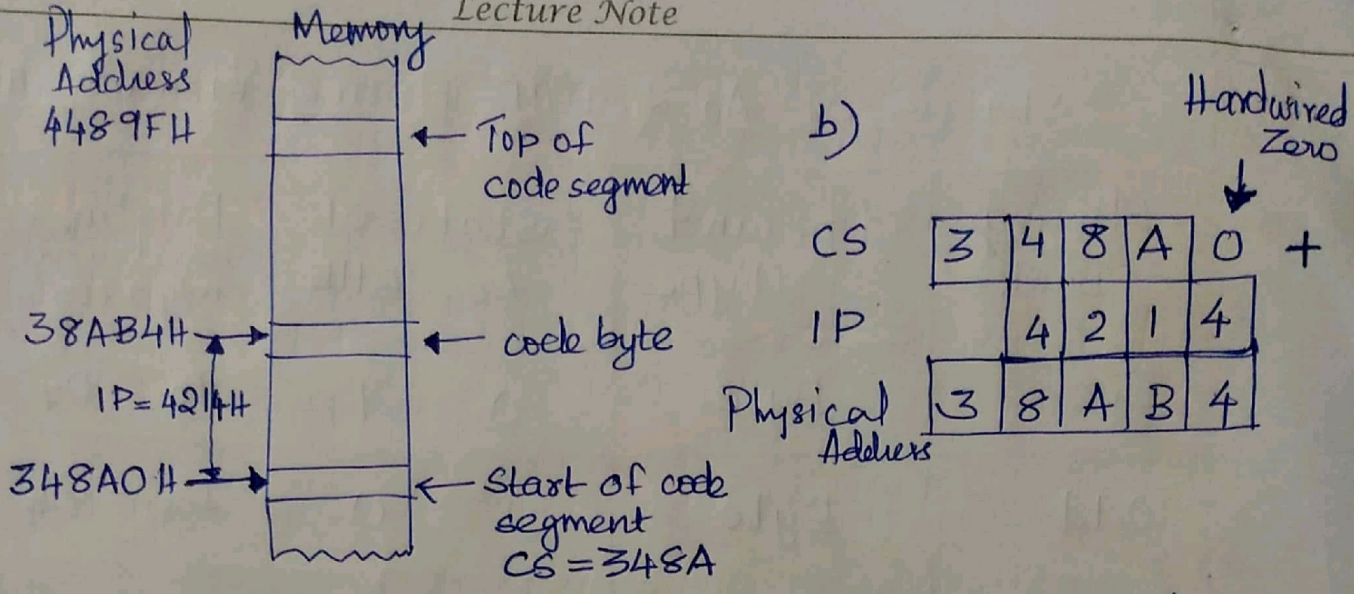


figure 1.7 Addition of IP to CS to produce the physical address of the code byte.
a) Diagram b) Computation

Physical Memory Address = Base address $\times 10_H$ + offset

Case-I	(CS)	$\times 10_H$ + IP
Case-II	(DS)	$\times 10_H$ + offset specified by the instruction.
Case-III	(ES)	$\times 10_H$ + offset specified by the instr.
Case-IV	(SS)	$\times 10_H$ + SP

1. d) Bus Control logic Unit

Generate all the bus control signals such as m/y read, m/y 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 mfy as the address generated from AGU 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
* 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.

✓ G.P.R can be used for data storage. When they are not involved in any, special functions assigned to them.

Register	Special 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 m/y 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 m/y (TOS).
BP	Base Pointer: hold the base value in base addressing mode using the stack segment. (offset)
SI (For string instruction)	Source Index: hold the index value of the source operand.
DI (For string instruction)	Destination Index: hold the index value of the destination operand.

2.d) Flag Register (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) → 1 (high); If there is a carry from the addition or borrow from the subtraction.
→ 0; no carry

② AF (Auxiliary Carry Flag) → 1; 0: no carry; If there is a carry from low nibble to high nibble of a 8 bit no.

③ OF (Overflow Flag) → 1; 0: no overflow; If there is an arithmetic overflow ie. If the size of the result exceeds the capacity of the destination location.

④ PF (Parity Flag) → 1; even parity.
→ 0; odd parity.

⑤ ZF (Zero Flag) → 1; result is zero.
→ 0; result is not zero.

⑥ SF (Sign Flag) → 1; result is negative.
→ 0; result is positive.

⑦ DF (Direction Flag) {used for string data accessing} → 1; autodecrement SI & DI mly pointer
→ 0; autoincrement SI & DI mly pointer

⑧ IF (Interrupt Flag) → 1; { to enable interrupt. (during external interrupt request)
→ 0; to disable interrupt.

⑨ TF (Trap Flag) → 1; Step by step execution (8086 generate an internal interrupt after execution of each instruction)
→ 0; Otherwise.

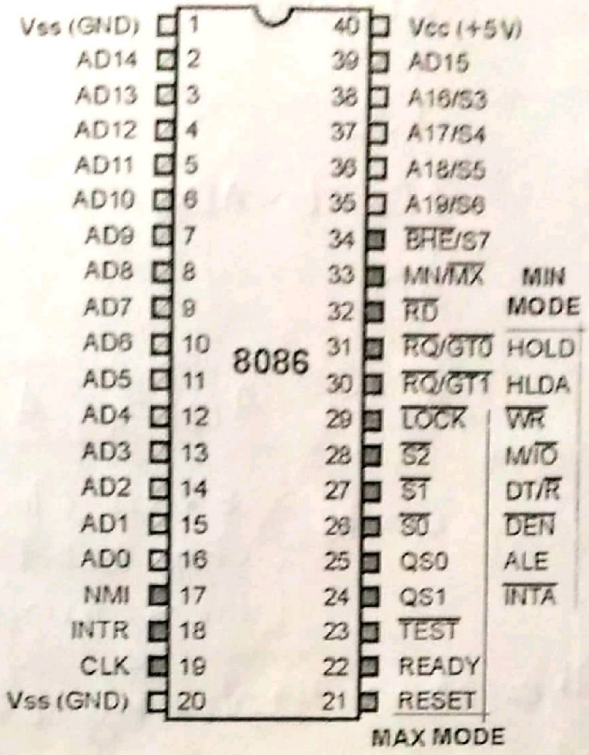
Questions

1) Compute the physical address
   a) CS: F272 & IP: 0F2F b) 2F72 : 2971
2) Draw and Explain the flag register of 8086.
3) With a neat diagram, explain the architecture of 8086.
4) Explain the physical memory organisation of 8086.
5) Explain the register organisation of 8086.

References

1. A. Nagoorkani, Microprocessors and Microcontrollers Second Edition Page No: 1.29 - 1.32
2. Douglas V. Hall, SSSP Rao, Microprocessors and Interfacing, 3e.

Title: 8086 Pinout and Signals



✓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 → 8284 clock generator generates max Internal clock of 5MHz.

✓ In addition to 20 address lines, 8086 uses a separate 16 bit address; so it generate 64kb I/O addresses.

To differentiate
→ M/y addresses
→ I/O addresses
} using M/$\overline{IO}$ → high
M/$\overline{IO}$ → low

Pin	Function
(1) & (20)	Ground
(40)	V_{CC}
(16) to (2)	AD_0 to AD_{14}
(39)	AD_{15}
(38) to (35)	A_{16}/S_3, A_{17}/S_4, A_{18}/S_5, A_{19}/S_6
(19)	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 2 Nos of 8-bit 74LS373 D-Latch. When ALE become high, two latches contain address A_0 to A_7 and A_8 to A_{15} and third latch contain address A_{16} to A_{19}. Then the address given out through the 20 bit address bus.

When ALE become low, two bidirectional buffer (8 bit) contain data D_0 to D_7 & D_8 to D_{16} and data given out through the 16 bit data bus.

Third latch contain status signals S_3 to S_6. & 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 → Indicates the status of an 8086 interrupt enable flag.

S_6 → low ; 8086 is the bus master
→ high ; local bus master gain the bus control from the processor.

(34) BHE/S7 → 0 ; select odd address bank
Bus High Enable

(16) A_0 (address line 0) → 0 ; select even address bank

(17) & (18) → Interrupt pins
17 → NMI (Non markable Interrupt)
18 → Interrupt Request

(33) MN/$\overline{MX}$ → high ; 8086 in minimum mode
→ low ; 8086 in maximum mode

(32) $\overline{RD}$ → low ; m/y or I/O read operation
Read s/g → high ; no read

(23) $\overline{TEST}$ → low; stop execution of WAIT instruction.

Used to synchronize an external activity to the processor internal operation.
8086 will enter a WAIT state after execution of the WAIT instruction. It will resume execution only when $\overline{TEST}$ is made low.

(22) READY → normally 1 (high)
→ low; Introduce WAIT states
OR
Used by the m/y 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 register are cleared & queue is emptied.

8086 operates in 2 Modes

Min mode

* MN/$\overline{MX}$ → high
* S/m → uniprocessor s/m
* 8086 itself generate bus control signals.

Max mode

* MN/$\overline{MX}$ → low
* multiprocessor s/m
* 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 s/g by the processor to the bus master after giving bus control.
(29)	$\overline{WR}$ → low	; to write data to memory/ I/O port.
(28)	M/$\overline{IO}$ → high	; memory access.
	→ low	; I/O address (address) access.
(27)	DT/$\overline{R}$ → high	; to transmit data.
	→ low	; to receive data.
(26)	$\overline{DEN}$ → low	; to enable the external bi-direction data buffers.
(25)	ALE → high	; address lines
	→ low	; data lines/status lines

(used to demultiplex address & data lines)

(24) $\overline{INTA}$ → an acknowledgement signal produced by the processor, when the interrupt request is accepted.

Maximum Mode

31 $\overline{RQ}/\overline{GT0}$
30 $\overline{RQ}/\overline{GT1}$

(Bus Request and Bus Grant)
↳ Request used by the local bus masters to force the processor inorder to release the local bus at the end of the processor's current bus cycle.

$\overline{GT0} \xrightarrow{\text{priority}} \overline{GT1}$

(29) $\overline{LOCK}$ ⟹ While executing an instruction prefixed by LOCK, this pin is tied low inorder 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 inorder 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 m/c
1	1	0	Write m/c
1	1	1	Passive/Inactive

(25) & (24) ⇒ Instruction Queue Status

QS0, QS1 — Used by external device, to track the internal status of the queue in 8086.

QS1	QS0	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 Nagoorkani, Microprocessors and Microcontrollers Second edition page No: 1.24 to 1.29
2. A.K Ray, Advanced Microprocessors and peripherals. page No: 8–12

MINIMUM MODE 8086 SYSTEM AND TIMINGS

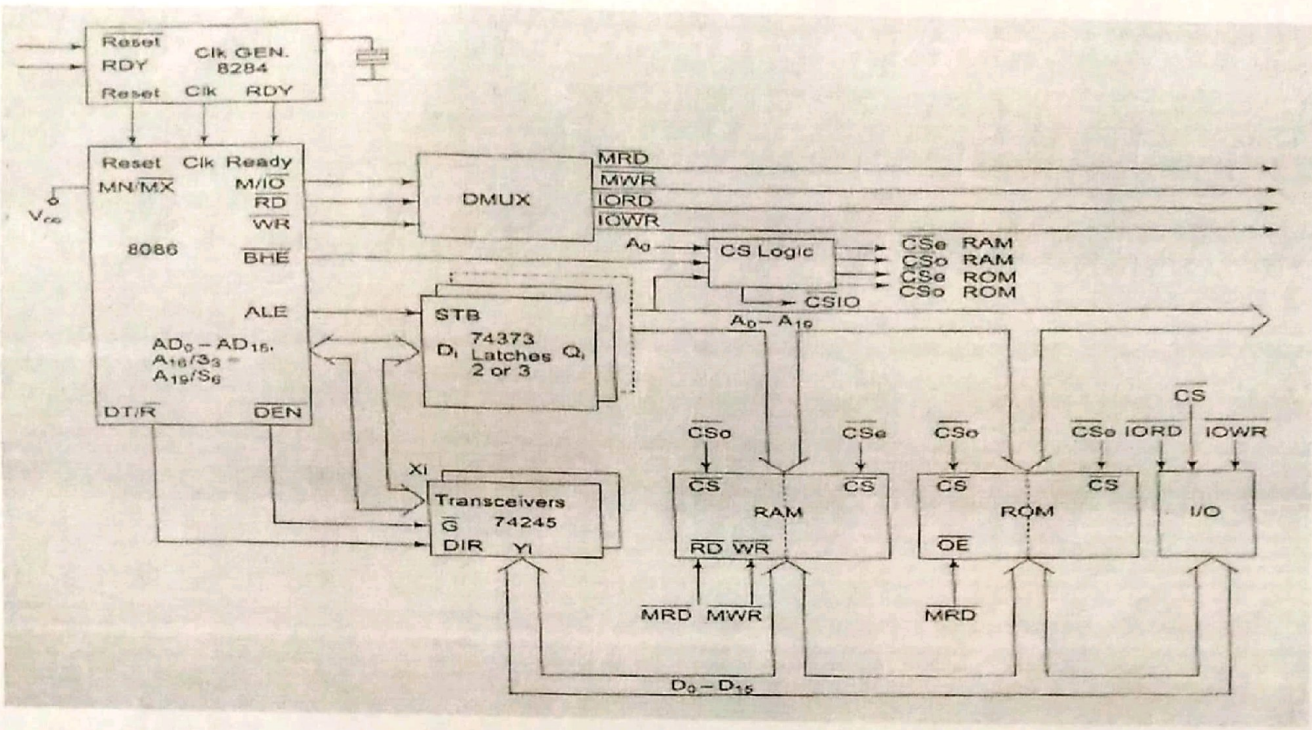


Fig 1.1. Minimum Mode 8086 System

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.

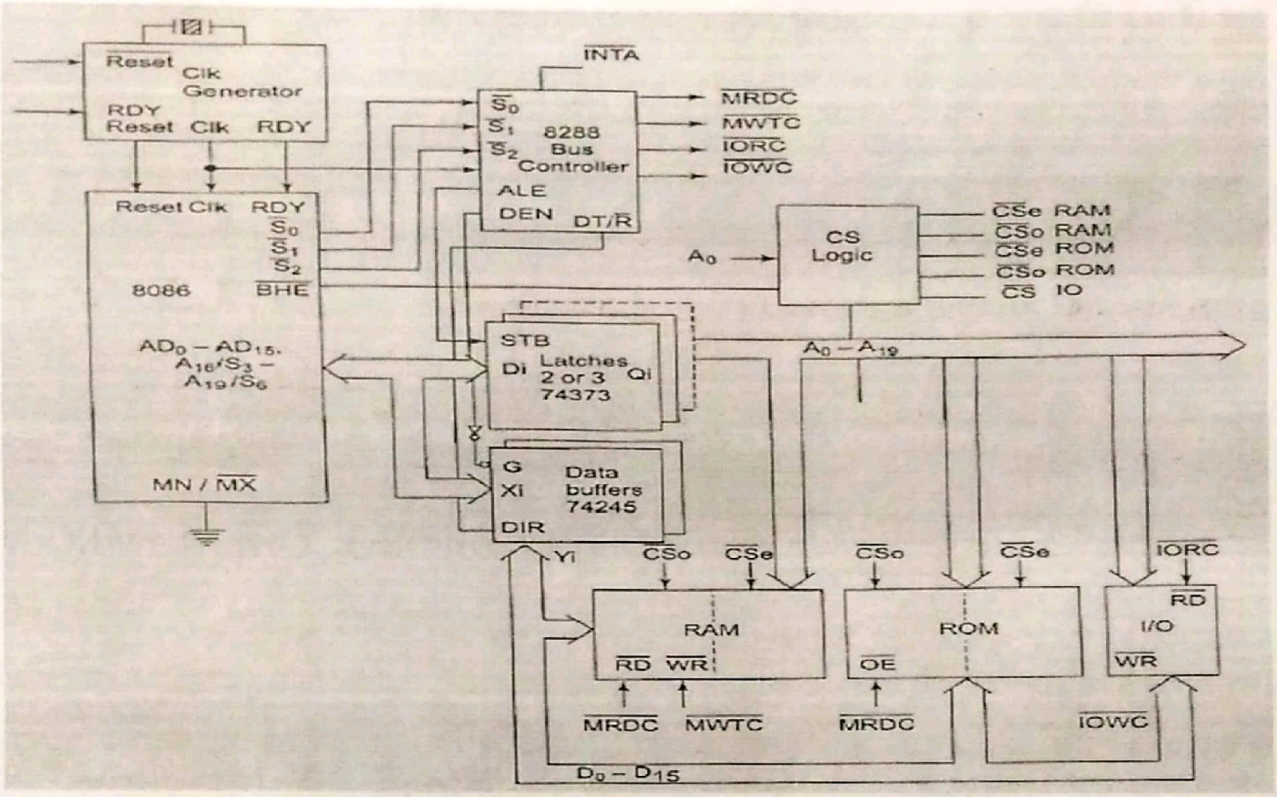
Transreceivers are the bidirectional buffers and sometimes they are called as data amplifiers. They are required to separate the valid data from the time multiplexed address/data signal. They are controlled by two signals, namely, DEN* and DT/R*. The DEN* signal indicates that the valid data is available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



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.

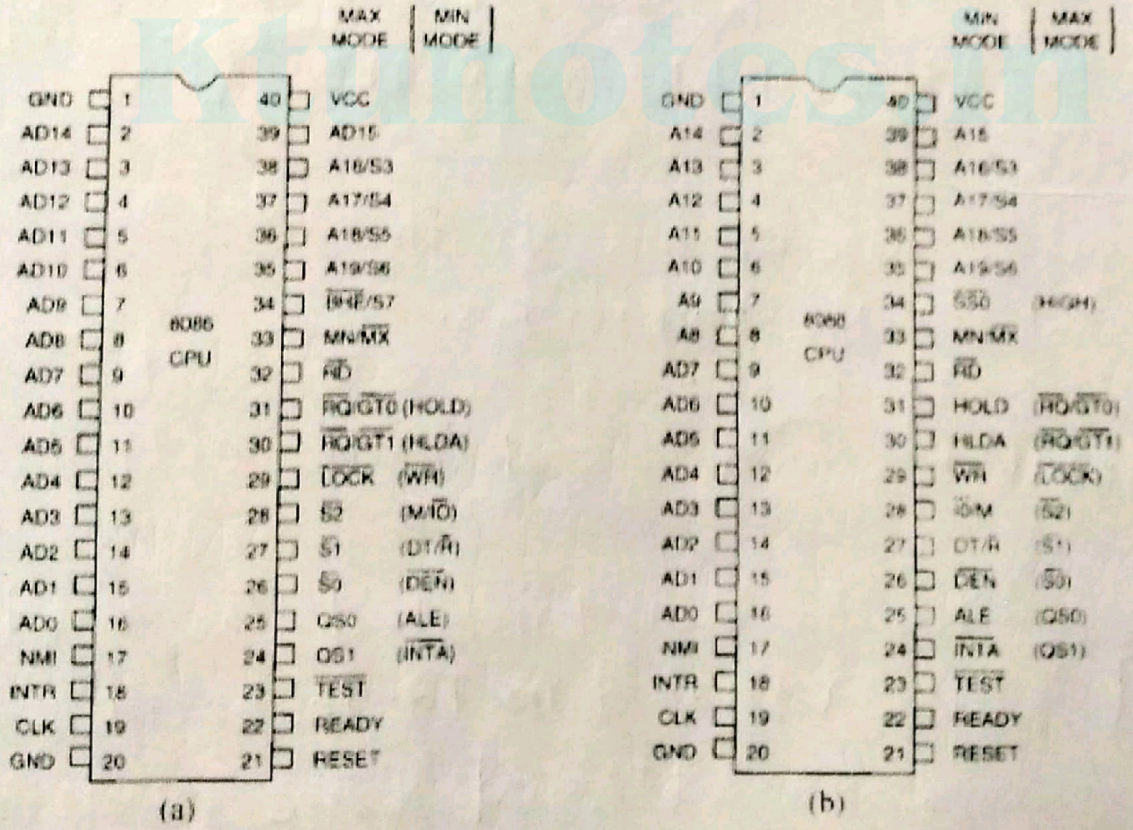
The basic functions of the bus controller chip IC8288, is to derive control signals like RD* and WR* (for memory and I/O devices), DEN*, DT/R*, ALE, etc. using the information made available by the processor on the status lines. The bus controller chip has input lines S2*, S1* and S0* and CLK. These 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.

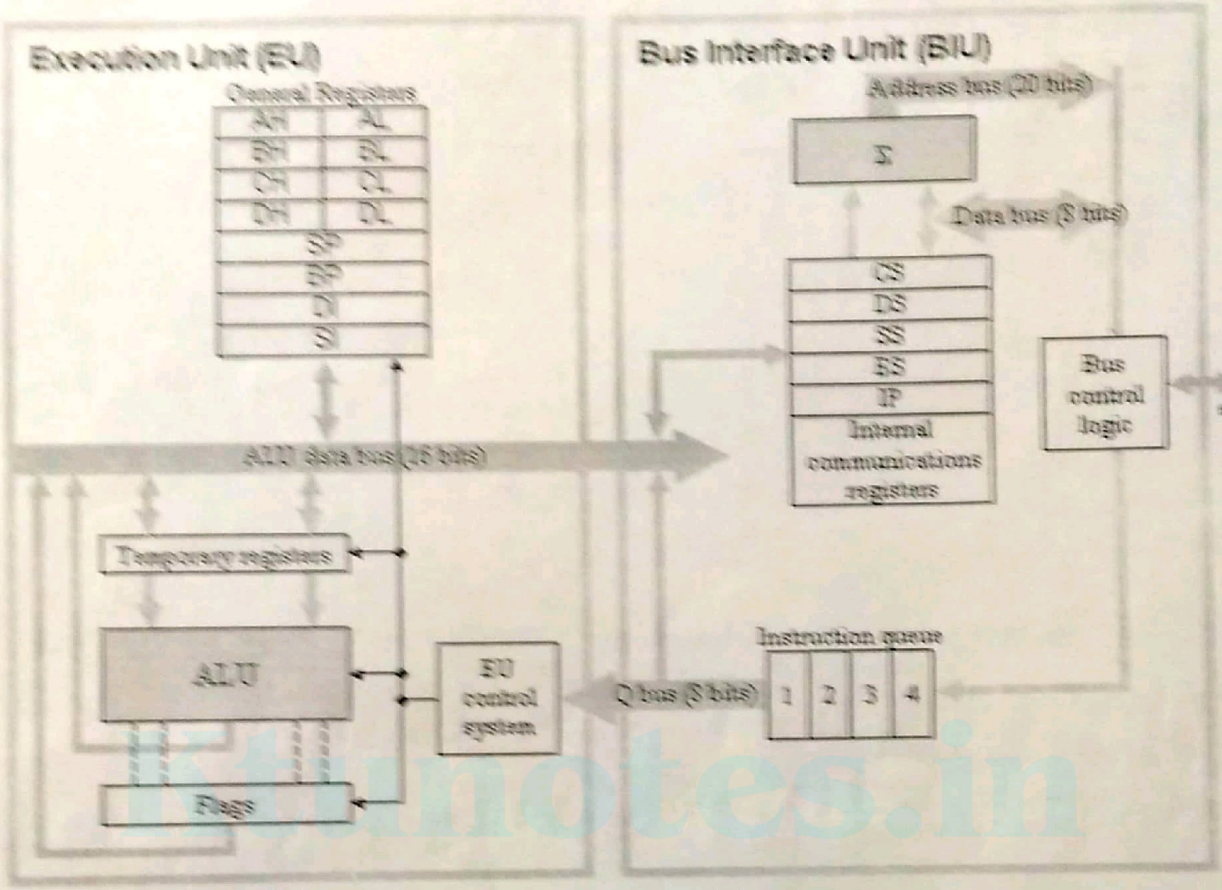
IORC*, IOWC* are I/O read command and I/O write command signals respectively. These signals enable an IO interface to read or write the data from or to the addressed port. The MRDC*, MWTC* are memory read command and memory write command signals respectively and may be used as memory read and write signals. 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
* $\overline{BHE}$ signal to access higher byte	* Data bus is 8bit wide, it does not have $\overline{BHE}$ signal.
* 6 byte instruction queue.	*4 byte instruction queue.
* $(M/\overline{IO})$ for memory or I/O access.	* $IO/\overline{M}$ for memory or I/O access.
*Pin no:34 is $\overline{BHE}/S_7$. 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 $\overline{SS0}$. It acts as $\overline{SO}$ in the minimum mode of 8088. This signal is combined with $IO/\overline{M}$ and $(DT/\overline{R})$ to decode the function of the current bus cycle.

8086/8088 Pin Configuration





Architecture of 8088 processor

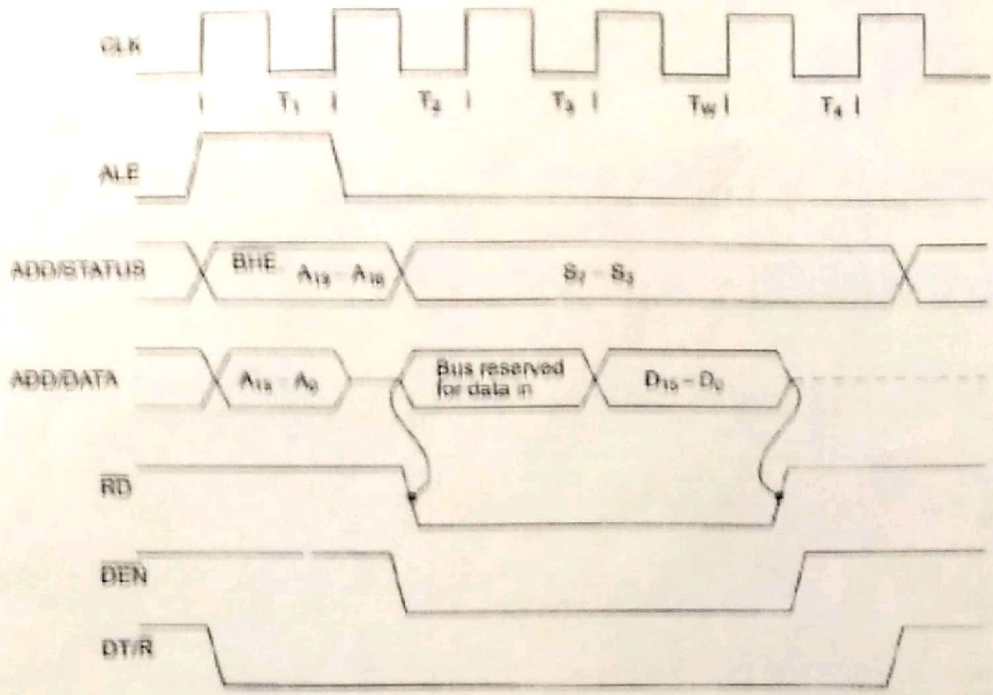
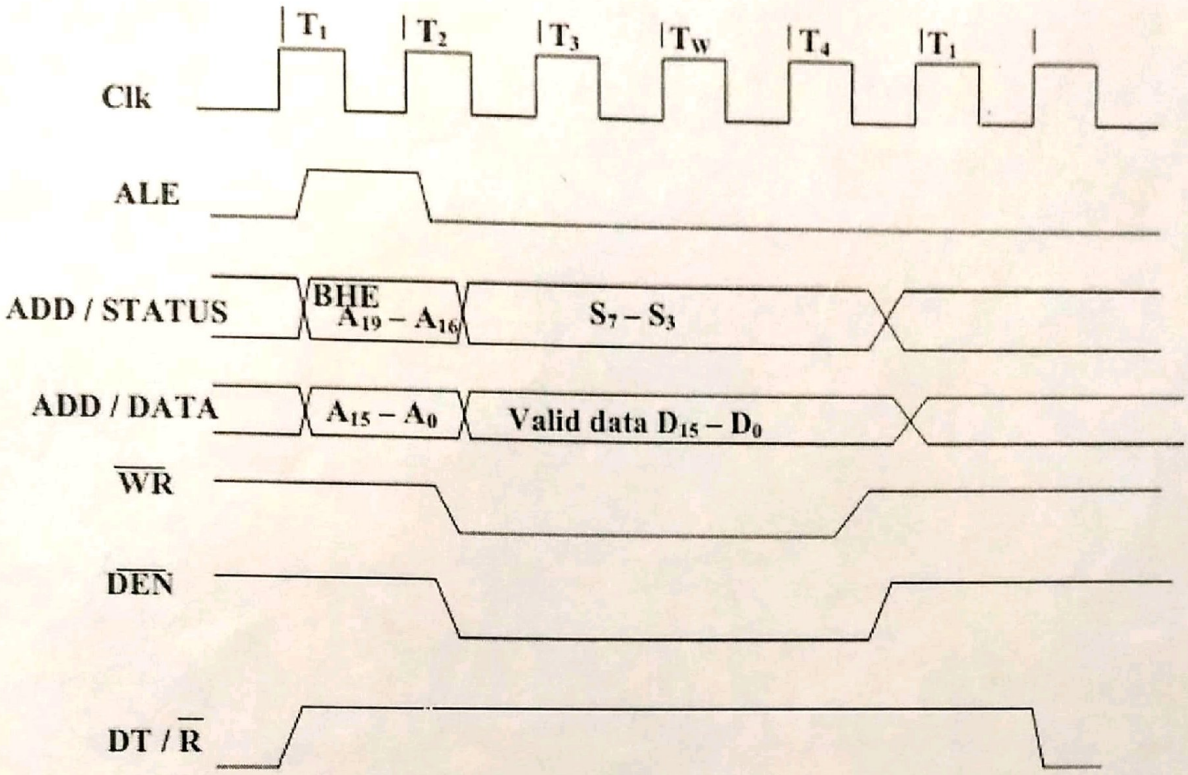


Fig. 2.4 Read cycle timing diagram for minimum mode

The read cycle begins in T1 with the assertion of the Address latch Enable(ALE) high and M/$\overline{IO}$ signal (M/$\overline{IO}$ ↗ high ; m/y address, ↘ low ; I/O address). During the negative edge of this s/g, 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($\overline{RD}$) signal is also activated in T2. ie. become low that causes the addressed device to enable its data bus driven. After $\overline{RD}$ goes low, the valid data is available on the data bus.



Write Cycle Timing Diagram for Minimum Mode

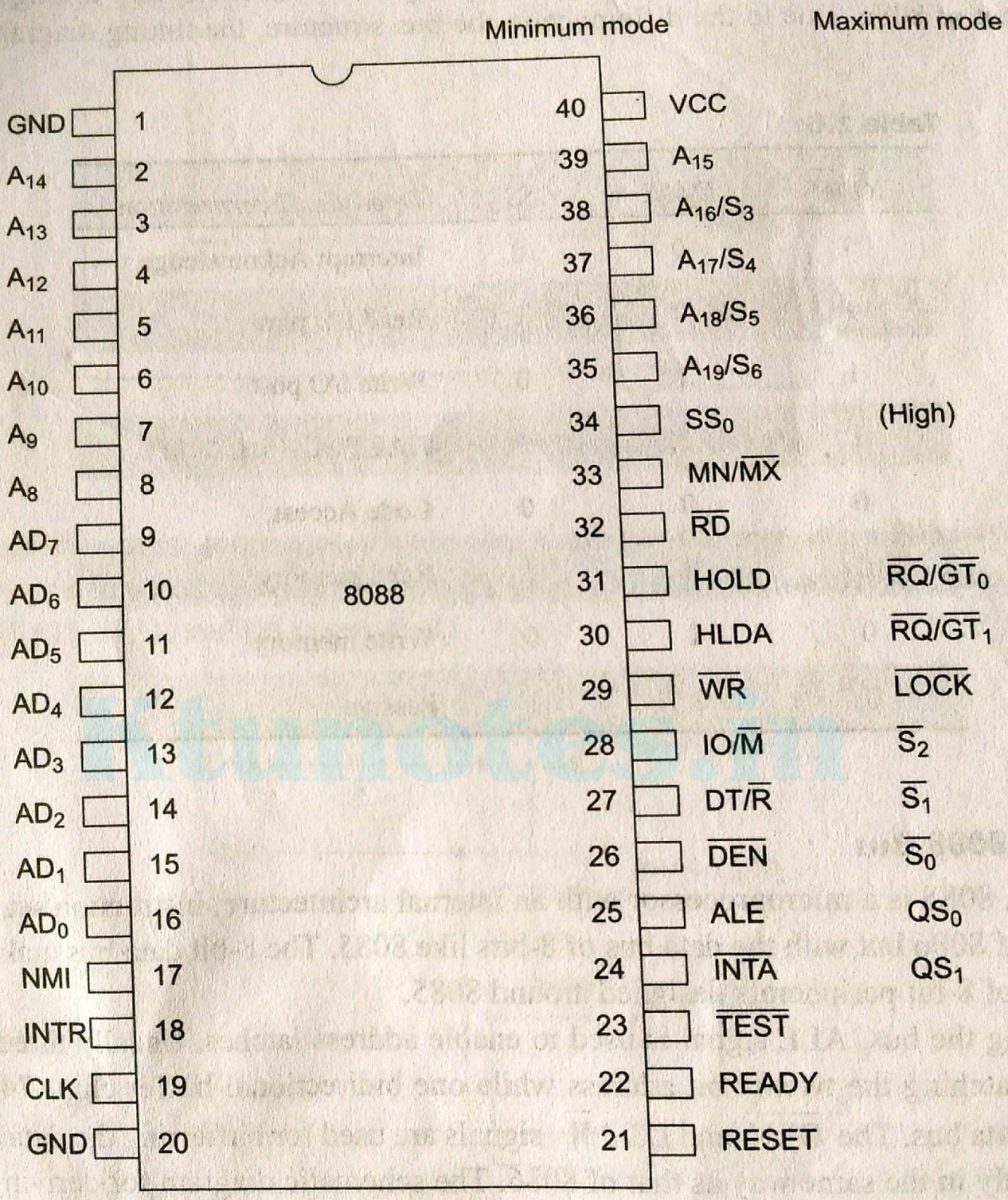


Fig. 1.18 *Pin Diagram of 8088*

Table 1.6

$IO/\overline{M}$	$DT/\overline{R}$	$\overline{SS_0}$	*Operation/Interpretation*
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) ← (DH)

MOV BX, DX ; (BX) ← (DX)

② Immediate A·M

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

eg: MOV DL, 08H ; (DL) ← 08H

MOV AX, 0A9FH ; (AX) ← 0A9FH

③ Direct Addressing

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

Displacement i.e. ⇒ effective address (E·A)

8-bit displacement

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

Memory Address (20-bit physical addr. of m/y) = BA + EA

{ CS , IP
DS , SI
ES , DI
SS , SP }

eg: MOV DX, [08H]

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

BA = (DS) × 16_{10} ; MA = BA + EA

(Base address) × 10_H (My address)

(DX) ← (MA)

eg: MOV AX, [089DH]

EA = $089D_H$; BA = (DS) × 10_H

MA = BA + EA

(AX) ← (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) × 10_H

MA = BA + EA

(CX) ← (MA)

MOV AX, [SI]

EA = (SI)

BA = (DS) × 10_H

MA = BA + EA

(AX) ← (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 ← sign extend 08H

EA = (BX) + 0008H

BA = (DS) × 10H

MA = BA + EA

(AX) ← (MA)

Segment

BX → DS

BP → SS

b) Indexed Addressing Mode

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

eg: MOV CX, [SI + 08H]

0008H ← sign extend 08H ; EA = (SI) + 0008H

BA = (DS) × 10H ∴ MA = BA + EA

(CX) ← (MA)

Based Indexed Addressing Mode

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

eg: MOV DX, [BX + SI + 0AH]

000AH ← sign extend 0AH ; EA = (BX) + (SI) + 000AH

BA = (DS) × 16_{10} ; MA = BA + EA

(DX) ← (MA)

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

In this A.M, the E.A of Src data stored in → SI reg & the E.A of Dest. Data stored in → DI reg.

⇒ Seg. register is used for calculating base address.

for src data → DS

for dest data → ES.

eg: MOVS BYTE REP> MOVSB

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

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

(MAE) ← (MA)

If DF=1, (SI) ← (SI) - 1 & (DI) ← (DI) - 1

If DF=0, (SI) ← (SI) + 1 & (DI) ← (DI) + 1

⑨ Direct I/O port Addressing Mode

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

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

eg: IN AL, [09H]

Port addr = 09H

(AL) ← (Port addr)

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

eg: Port addr = (DX) ; (PORT) ← (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 0AH

0000A ← (sign extend) 0AH

If ZF=1, then (IP) ← (IP) + 000AH

(EA) = (IP) + 000AH

(BA) = (CS) * 16_{16}

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

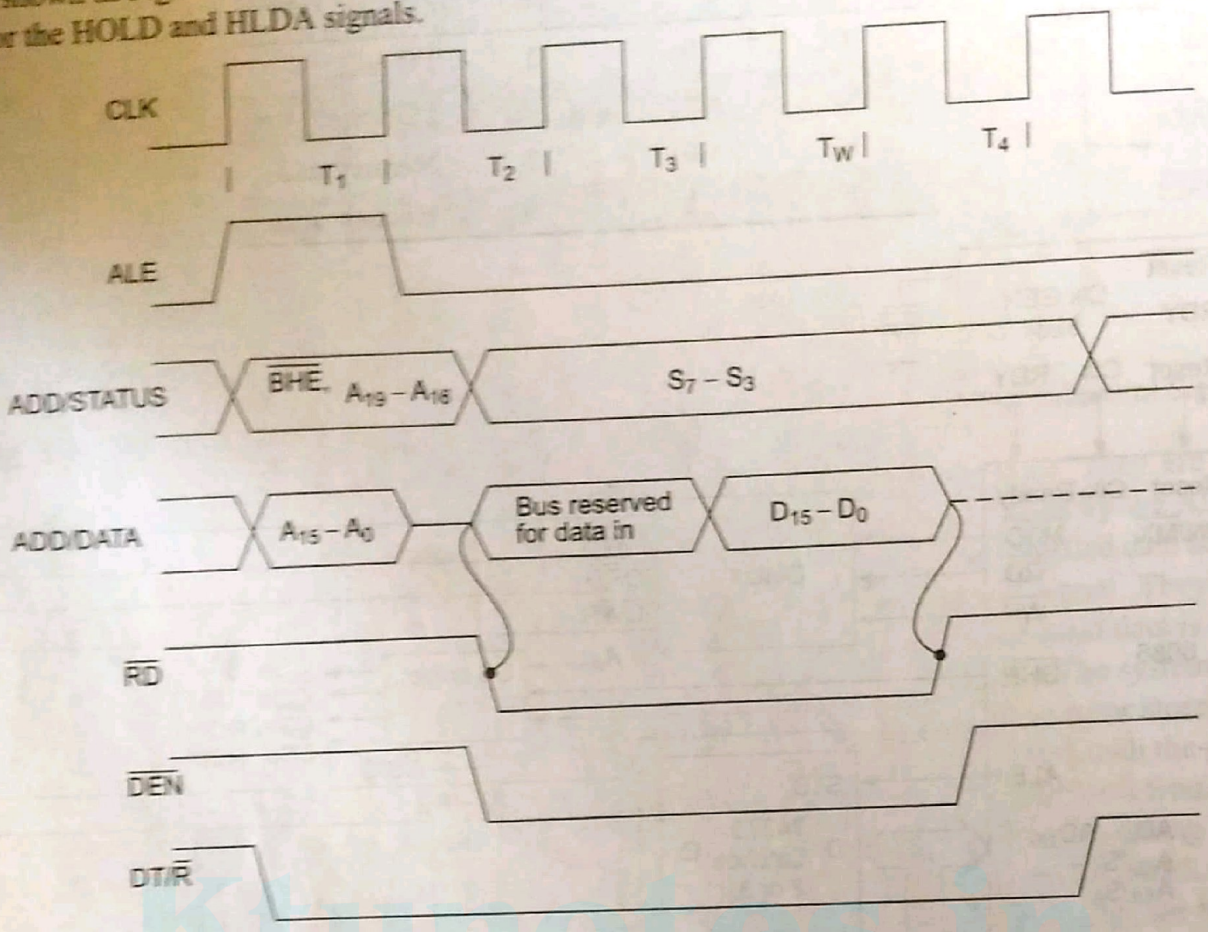


Fig. 1.14(a) *Read Cycle Timing Diagram for Minimum Mode*

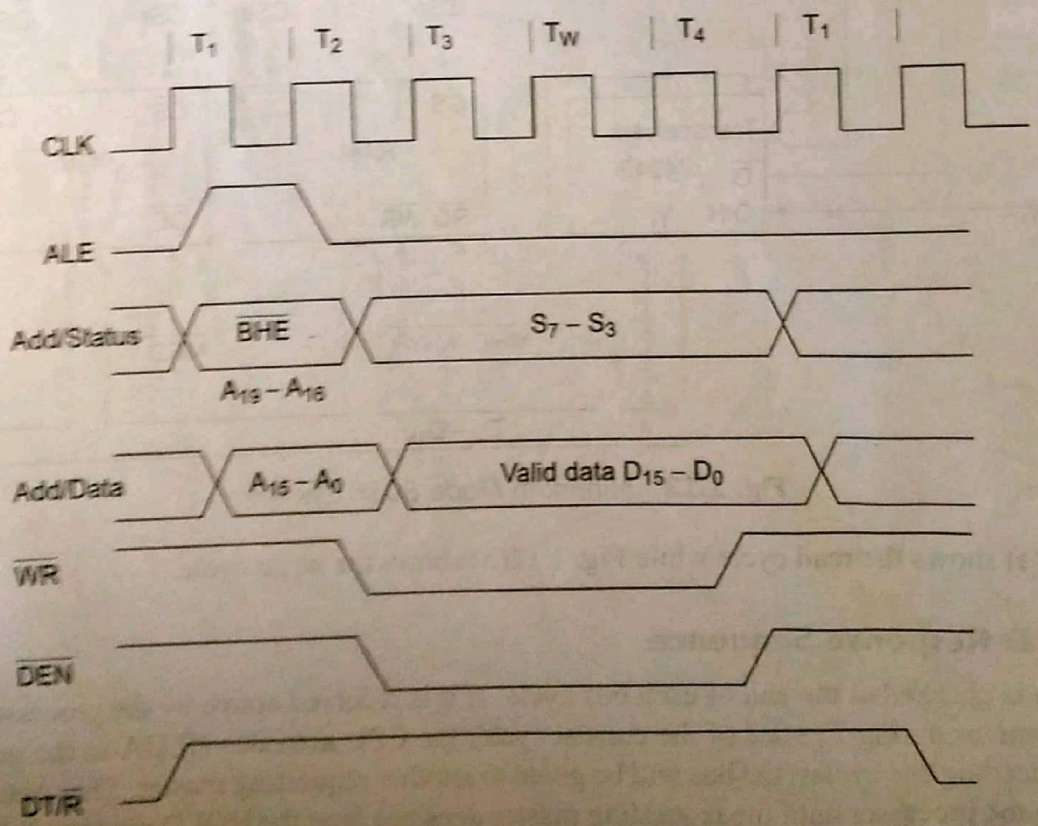


Fig. 1.14(b) *Write Cycle Timing Diagram for Minimum Mode Operation*

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
   - - - -
   - - - -      } pgm code 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 0F2H ; the value of variable PORT1 is F2H.

(9) PROC, FAR, NEAR, ENDP

- PROC → beginning of procedure
- FAR → inter segment
- NEAR → intrasegment call
- ENDP → indicate end of a procedure

General form:

```
procname PROC [NEAR/FAR]
        .....
        .....   } pgm stats in procedure
        RET
procname ENDP
```

(10) MACRO AND ENDM

Indicate the beginning of a macro → end of a macro

General form:

```
macroname MACRO [Arg1, Arg2 ...]
        .....   } definitions declaration Pgm statements
macroname ENDM
```

Question

Explain the assembler directives of 8086 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

Title Sample Pgms (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
                          if result is -ve.
L1:     MOV [1300], AX
        MOV [1302], CX
        HLT
```

NUM1 : [1200, 1201] = 1101 H

NUM2 : [1202, 1203] = 5678 H

Difference: [1300, 1301, 1302, 1303] = 0001 4577H

[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
       END START
```

(4) 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' on the display mon...

```
DATA1  SEGMENT
       MSG DB 'Welcome to MBCET, Thiruvananthapuram$
                ; here $ fonctions as a string terminator
DATA1  END

CODE1  SEGMENT
       ASSUME CS: CODE1, DS: DATA1
START: MOV AX, DATA1
       MOV DS, AX

       MOV AH, 09H      } DOS fonction call for
       MOV DX, OFFSET MSG  displaying a string
       INT 21H             on the string on the
                           screen.

       MOV AH, 4CH      } DOS fn call to ter-
       INT 21H             minate the execution
                           of the user pgm and 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
        END START
```

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

```
DATA SEGMENT
    LIST DB 25H, 23H, 12H, 0FFH, 0EFH, 86H
    COUNT EQU 06H
    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, 0000H
    MOV SI, OFFSET LIST
L2: ADD AL, [SI]
    JNC L1
    INC AH
L1: INC SI
    LOOP L2
    MOV SUM, AL
    MOV SUM+1, AH
    MOV AH, 4CH
    INT 21H
CODE ENDS
    END START
```

(8) To find the largest signed number

```
DATA SEGMENT
    LIST DB 7FH, 0F3H, 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]
    JGE L1 ; for largest
             JBE L1 for smallest
    MOV AL, [SI+1]
L1: INC SI
    LOOP L2
    MOV LARGEST, AL
    MOV AH, 4CH
    INT 21H
CODE ENDS
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 CX
    JNZ L2
    DEC BX
    JNZ L3
    MOV AH, 4CH
    INT 21H
CODE ENDS
```

(10) Addition of two 3x3 mateices

```
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]
       JNC L1
       INC AH
```

```
L1:   MOV  [BX], AX
      INC  SI
      INC  DI
      ADD  BX, 02
      DEC  CX      } Loop NEXT
      JNZ  NEXT    }
      MOV  AH, 4CH
      INT  21H
CODE  ENDS
      END  START
```

(11) BCD to Binary conversion

```
      MOV  BX, 1100        → MSD x 0A + LSD
      MOV  AL, [BX]
      MOV  DL, AL
      AND  DL, 0F      ; Now DL contain LSD
      AND  AL, 0F0
      MOV  CL, 04
      ROR  AL, CL      ; Now AL contain MSD
      MOV  DH, 0A
      MUL  DH          ; (AX) ← (AL) x (DH)
      ADD  AL, DL
      MOV  [BX+ ], AL
      HLT
```

Title: MACRO & PROCEDURES

Macro 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 MSG
          MOV AH, 09H
          MOV DX, OFFSET MSG
          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
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_SEG SEGMENT STACK          ; setup stack segment
        DW 100 DUP(0)            ; set aside 100 word locations as stack
TOP_STACK LABEL WORD             ; label directive is used to assign a name to the current value in loc counter. i.e. 101 word (200th byte)
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 BIN_VALUE, AL
        MOV AH, 4CH
        INT 21H
```

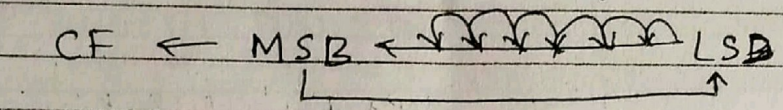
```
; Procedure starts
BCD_BIN  PROC  NEAR
         PUSHF
; does the conversion now
         MOV BL, AL
         AND BL, 0FH
         AND AL, 0F0H
         MOV CL, 04H
         ROR AL, CL
         MOV BH, 0AH
         MUL BH
         ADD AL, BL
         POPF
         RET
BCD_BIN  ENDP
CODE     ENDS
         END START
```

(C) Rotate Instruction

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

ROL Destination, Count

mem loc / reg

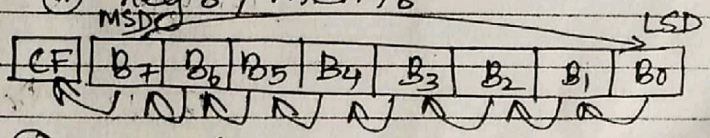


Note:

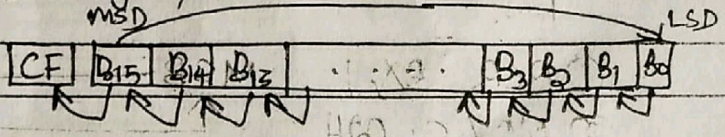
① ROL AX, 1

② MOV CL, 04H
ROL BL, CL

(a) reg 8 / mem 8



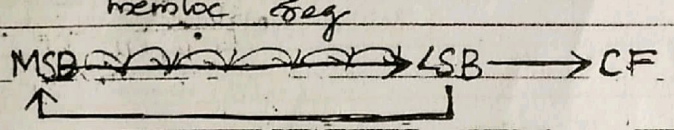
(b) reg 16 / mem 16



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

ROR Destination, Count

mem loc / reg

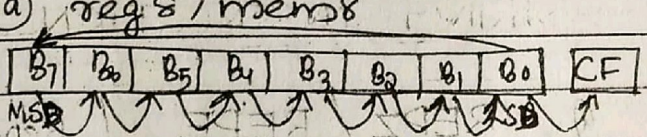


Note:

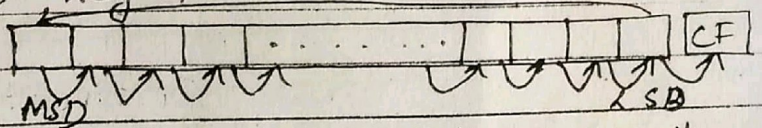
① ROR BH, 1

② MOV CL, 08H
ROR AL, CL

(a) reg 8 / mem 8



(b) reg 16 / mem 16



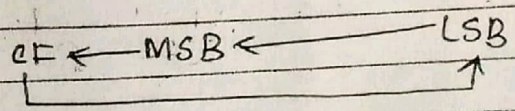
(C.3) RCL (Rotate Operand Around to the left through CF)

RCL Destination, Count

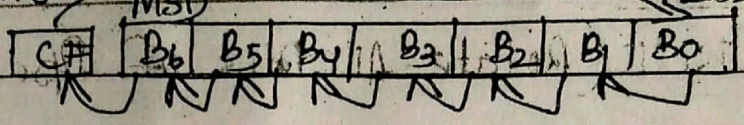
mem loc / reg

Note: ① RCL DX, 1

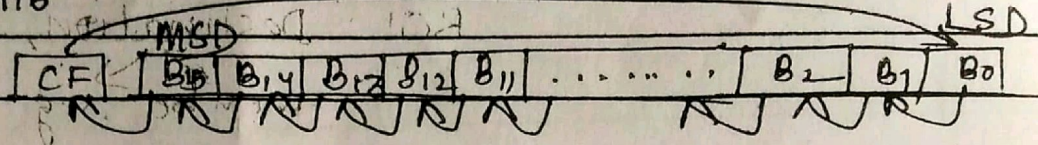
② MOV CL, 4
RCL AX, CL



ⓐ reg8/mem8



ⓑ reg16/mem16



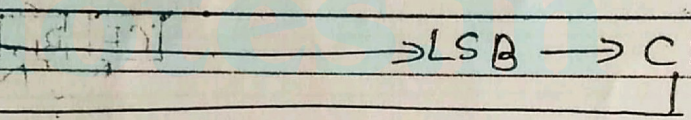
(C4) RCR (Rotate Operand Around to the Right through CF

RCR Destination, Count

mem/reg

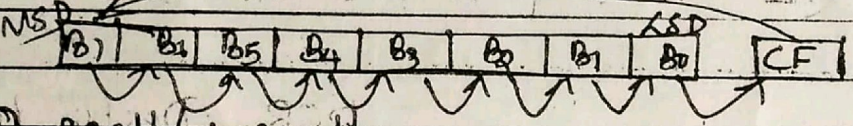
Note :

① RCR BX, 1

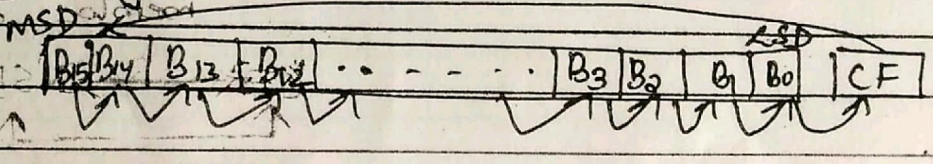


② MOV CL, 02H
RCR BX, CL

ⓐ reg8/mem8



ⓑ reg16/mem16



(4) STRING INSTRUCTION

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

(12) 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 its works

Processor receives an interrupt signal

↓

Stops execution of current normal pgm

↓

Save the status of the IP, CS, flag regs in stack

↓

then Processor executes a subroutine/proceed inorder 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

↓

Processor Restart the Normal pgm execution.

execute response as Interrupt Service Subroutine (ISR)

SOURCES OF INTERRUPT IN 8086

- ✓ From an external signal applied to NMI or INTR pin of the processor
- ✓ Execution of the interrupt instruction "INT n" n → type number (0-25
- ✓ From conditional interrupts produced in the 8086 by the execution of an instruc

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.

INTR NMI

(a) INTR

from external device

High signal of the type 0 to type 255 can be applied to INTR pin of the processor

To get the type number of the interrupt. ← send Ack signal

(b) NMI

Rising edge (low to high) transition of the signal can be applied to NMI Pin (The processor will execute type 2 interrupt)

Software Interrupt

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.

(a) Vectored Interrupt

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

(b) 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

(a) Maskable Interrupts

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

eg: Interrupt initiated through INTR pin.

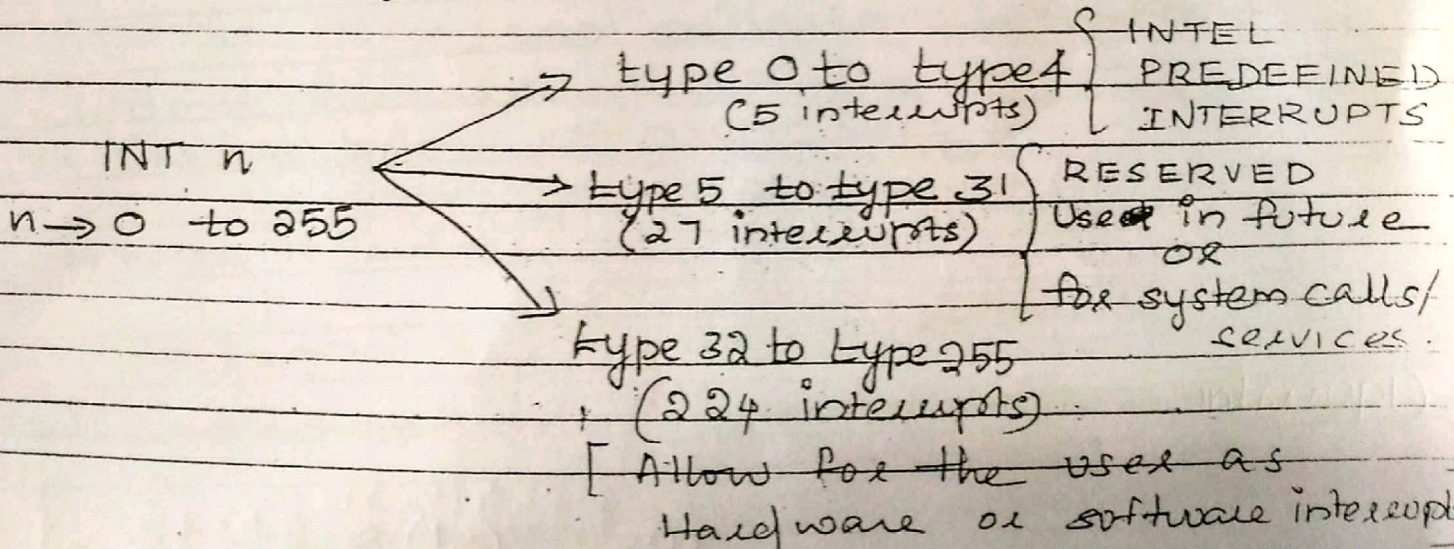
(b) Non-Maskable Interrupts.

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

eg: Interrupt initiated through NMI pin and all software interrupts "INT n"
n → 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 first 1kb m/y space [addr range 00000_H to $003FF_H$]

vector address → Base address + offset address

SERVICING AN INTERRUPT BY 8086

8086 Processor receives an interrupt signal

↓

Stop Execution of current normal pgm

↓

Save the status of the IP, CS, flag regs in stack

Push operation:

(SP) ← (SP) − 2 ; TOS ← (flag reg)

(SP) ← (SP) − 2 ; TOS ← (CS)

(SP) ← (SP) − 2 ; TOS ← (IP)

IF ← 0 & TF ← 0

↓

then processor execute a subroutine/procedure inorder to perform the specific task by the interrupt.

$\therefore M.A_{procedure}^{(ISR)}$ = new IP value + new CS value × 10/16

New IP value (offset addr)

Multiplying the type number specified in the instruction by 4:

eg: INT 8

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

$\therefore$ New IP value will be read from addr 00020H (vector address)

New CS value (Base addr)

Multiplying the type number by 4 and plus 2.

eg: INT 8

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

$\therefore$ New CS value will be read from addr 00022H (vector addr)

↓

At the end of ISR, the stored status of Regs in stack are restored to respective regs (pop operations)

(IP) ← TOS ; (SP) ← (SP) + 2

(CS) ← TOS ; (SP) ← (SP) + 2

(flag reg) ← TOS ; (SP) ← (SP) + 2

Processor Restart Normal pgm execution

lecture No: 15, 16

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 allows one by one to the processor. INTR pin.

→ 8259 used in cascade mode in a system to expand the interrupts upto 64. 8086 + 8259 (master) + 8259 (slave) ... (8259)

→ 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 priorities

① Automatic Rotation

IR_x → lowest priority (Int request being serviced)
IR_{x+1} → highest priority

eg: IR_3 – being serviced (lowest priority)

IR_0	IR_1	IR_2	IR_3	IR_4	IR_5	IR_6	IR_7
4	5	6	7	0	1	2	3

② Fully Nested Mode

IR_x → highest priority (any IR can be assigned the highest priority)

IR_{x-1} → lowest priority

$IR_x \rightarrow IR_3$

eg:

IR_0	IR_1	IR_2	IR_3	IR_4	IR_5	IR_6	IR_7
5	6	7	0	1	2	3	4

③ Specific Rotation Mode

Similar to the automatic rotation mode except the user can select any IR for lowest priority, thus fixing all other priorities.

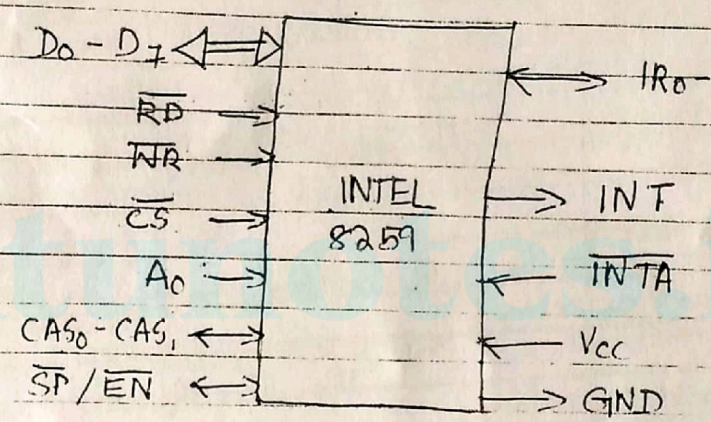
$IR_0 - IR_7$ (INTERRUPT REQUEST)

⇒ 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 interrupts, masked interrupts and interrupt being serviced.

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



D0 - D7 (8259)	Databus	(D0 - D7) Databus of 8086 connected to D0 - D7 (8259)

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 3-to-8 decoder)

3-to-8 decoder

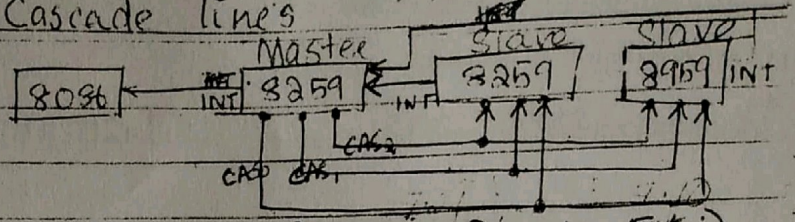
I/P address lines → A5, A6, A7

Decoder enable signals → A0, M/IO

A0 → 0, 1 — Internal 2 Address lines. [A1 (8086) connected to A0 (8259)]

fig No: 5.4 page No: 182

$CAS_0 - CAS_1$ ⟶ Cascade lines



(Page No: ~~185~~, fig No: ~~5.6~~)

$\overline{SP}/EN$ ⟶ Slave Program / Enable buffer

(i/p or o/p signal)

Act as

i/p signal — In ~~non buffered Mode~~

o/p signal — ~~Buffered Mode~~

$\overline{SP}/EN$ → 1 (master)

→ 0 (slave)

Is used to disable the data buffer while data is transferred from 8259 to the CPU.

Page No: 5.4 Fig 5.4 ~~explanation~~

Introduction (command words) Operation (command words)

~~Working of 8259 with 8086 Processor~~

(Processing of ~~Interrupt by~~ 8259)

First- 8259, ~~programmed~~ by sending ICW and OCW

↓

Receive an interrupt through any one of the interrupt line $IR_0 - IR_7$

↓

Ref 1 page No: 249 Fig No: 6.21 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 (1)

To get type number of Interrupt ← INTA (2)

$D_0 - D_7$ data bus: typenumber → get type number in 8086

↓

Using ~~type number~~, Calculate vector address of the interrupt & Memory address of the ISR and execute ISR

↓

8086 —~~command~~ word→ 8259 (to inform 8259 abt the end of Interrupt)

AEOI, OCW2

$IR_0 - IR_7$ (INTERRUPT REQUEST)

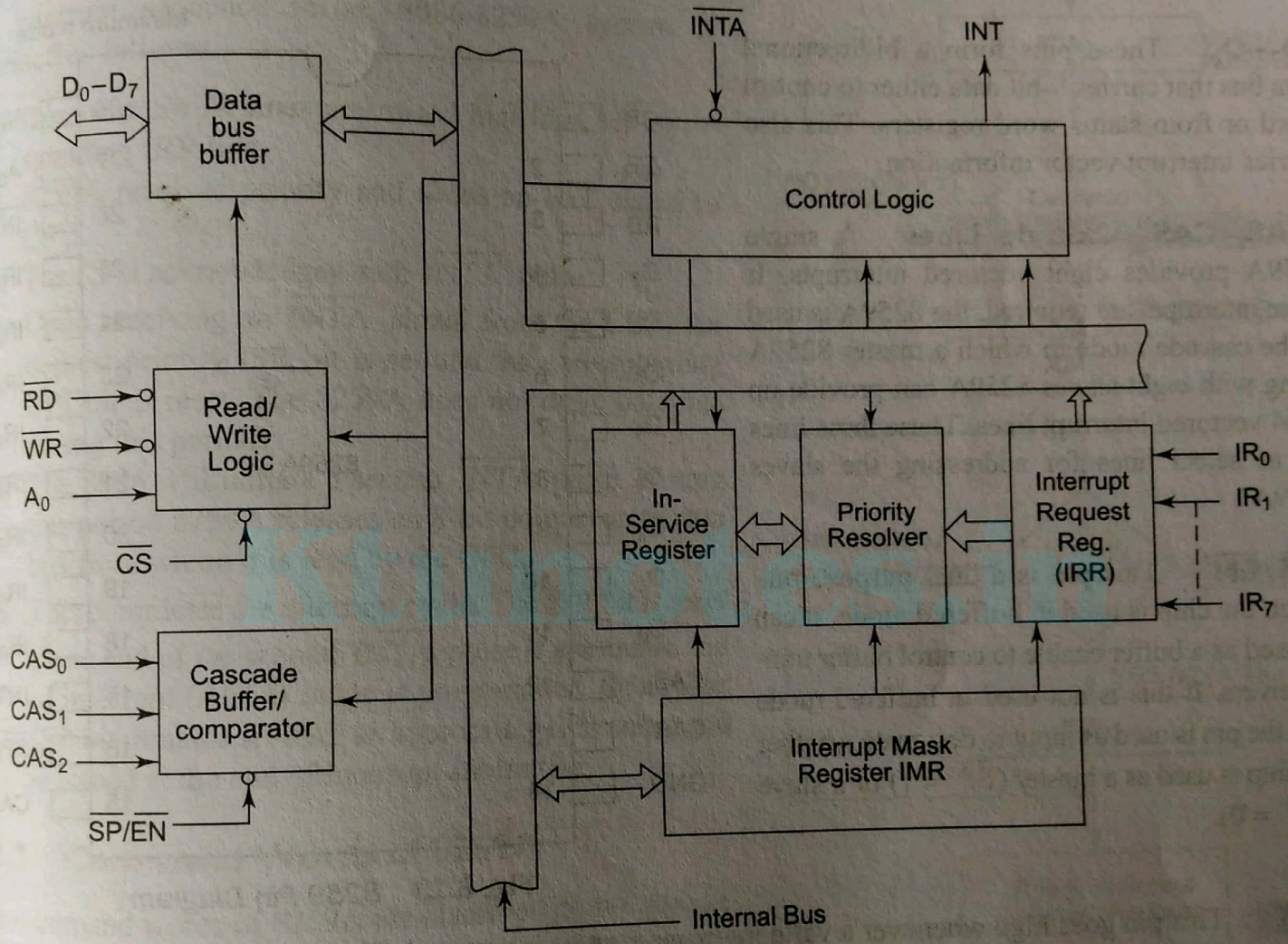


Fig. 6.12 *8259A Block Diagram*

Functional Block Diagram of 8259

{Page No. 184}
{fig: 8259}

8 Functional blocks.

(1) DATA BUS BUFFER

a) Processor send control word (command) to data bus buffer through D_0-D_7.

b) Processor read status word from data bus buffer through D_0-D_7.

(c) From the data bus buffer, the 8259 send type number (vector number) of the interrupt to the processor through D_0-D_7.

(2) Read / Write Logic

(internal address line)

→ The processor uses $\overline{RD}$, $\overline{WR}$ and A_0=0, to write a command word or read a status word.

→ 8259 is selected by $\overline{CS}$.

(3) Cascade Buffer / Comparator

→ Used to expand the interrupts of 8259.

→ In Cascade connection,

8259 directly interrupting 8086

↓

master 8259 SP/$\overline{EN}$ → high.

8259s interrupting the master 8259

↓

slave 8259s. SP/$\overline{EN}$ → low.

→ Cascade pins (CAS_0, CAS_1, CAS_2) (output pins) from the master are connected to the corresponding pins of the slave. (input 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 corresponding bit in ISR is set.

⑥ Interrupt Mask Register.

→ Used to disable (mask) (1) or enable (unmask) (0) 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~~ Marking Bits
ISR ~~checked for~~ interrupt being served.

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

⑧ Control Logic

8.L ~~INT signal~~ → INTR (8086)
8259 ← ~~INTA~~

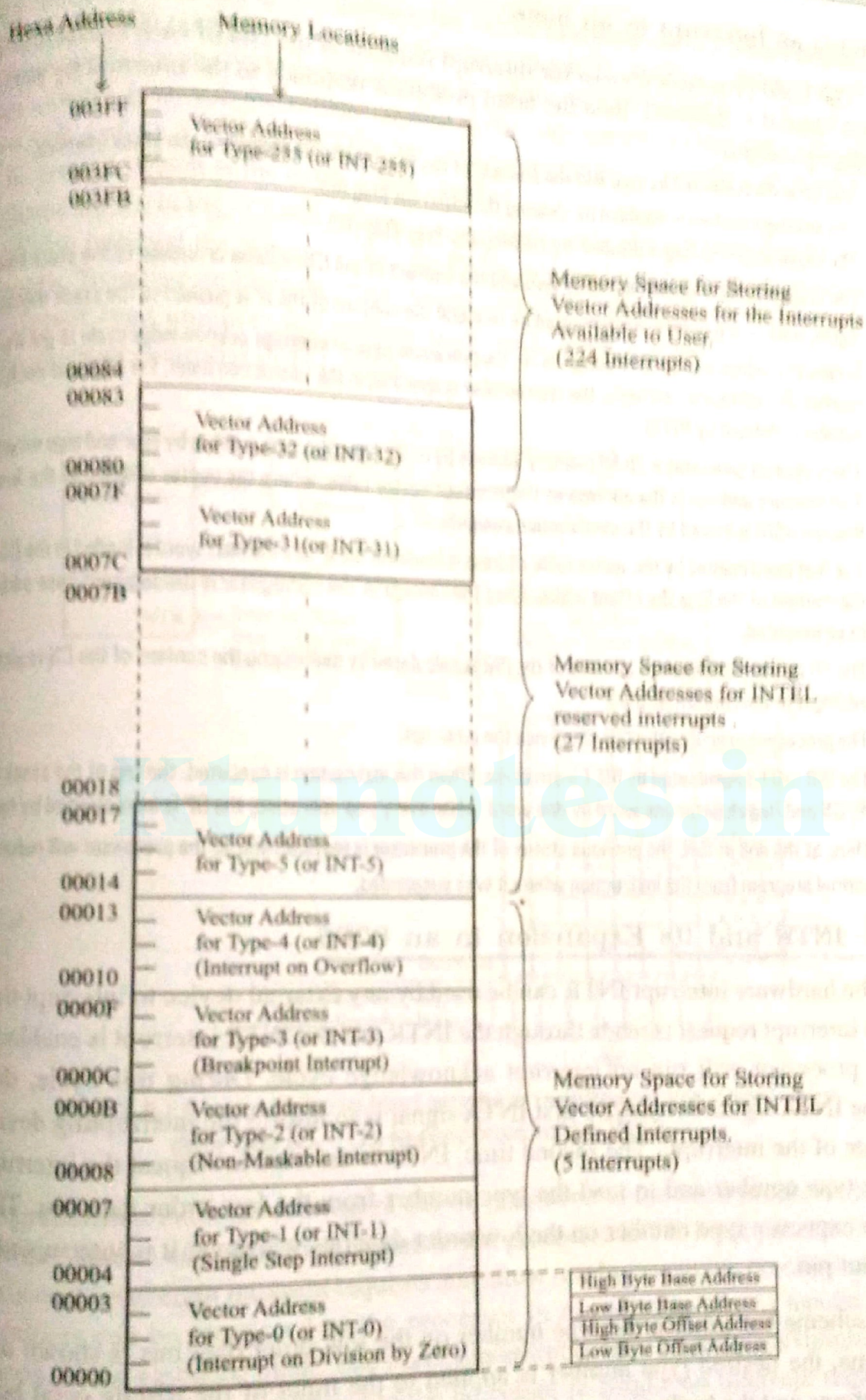


Fig. 7.4 : Organization of an interrupt vector table in 8086.

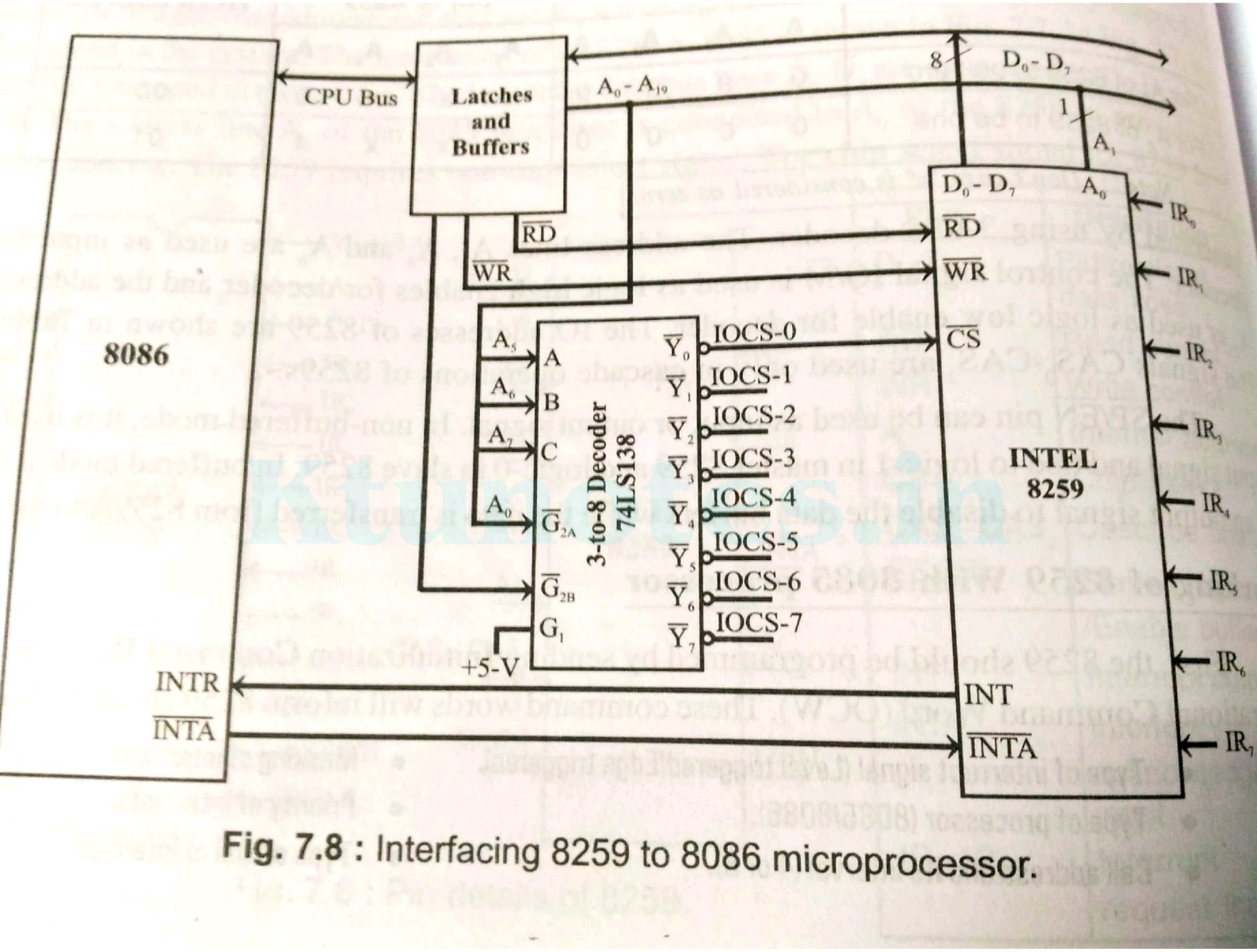
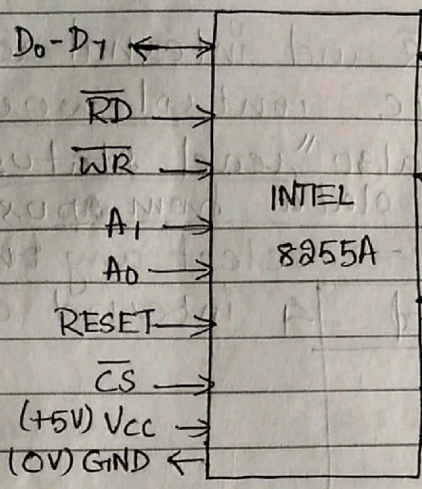


Fig. 7.8 : Interfacing 8259 to 8086 microprocessor.

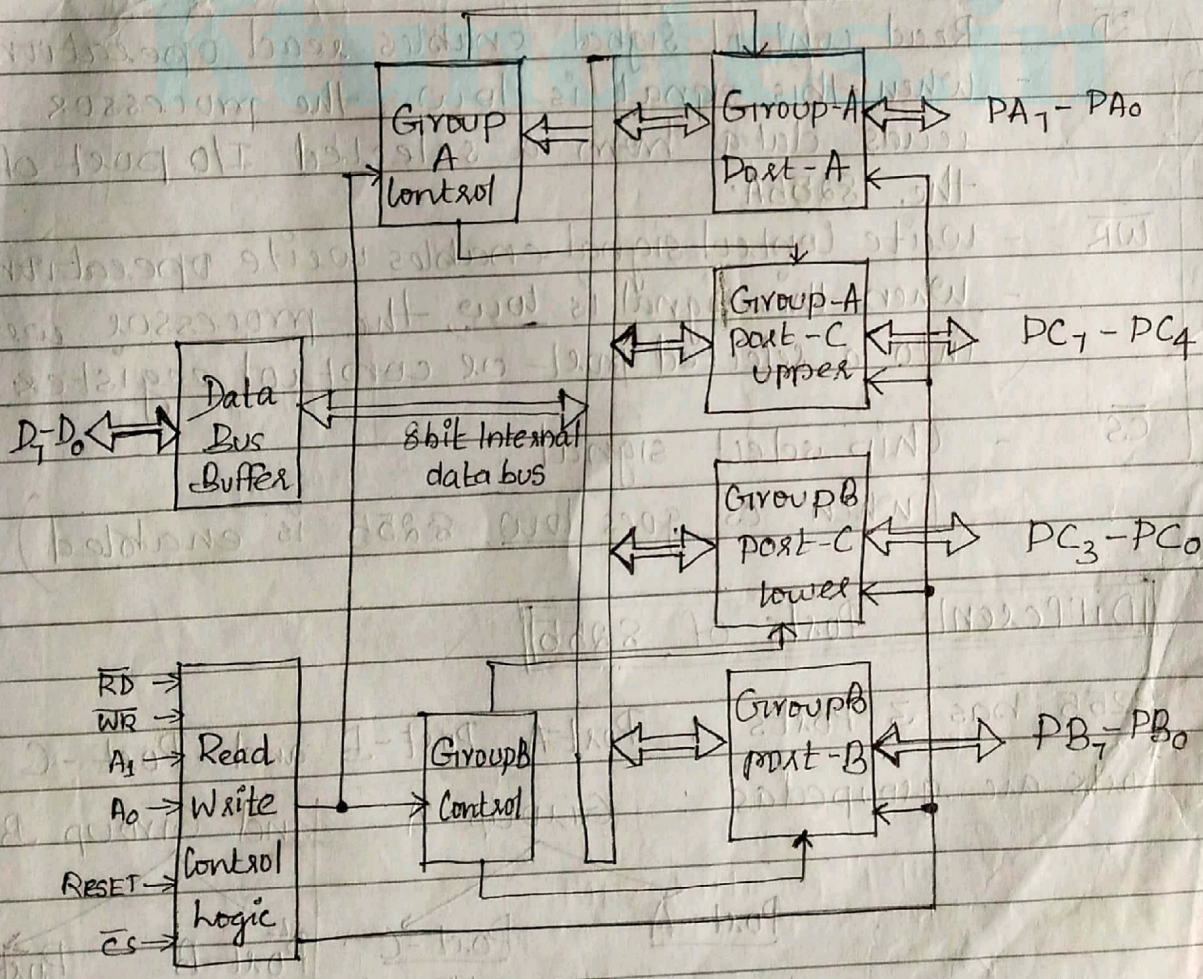
PROGRAMMABLE PERIPHERAL INTERFACE - INTEL 8255

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
$D_0 - D_7$	Datalines
RESET	Reset input
$\overline{CS}$	Chip select
$\overline{RD}$	Read Control
$\overline{WR}$	Write Control
A_0, A_1	Internal address
$PA_7 - PA_0$	Port-A pins
$PB_7 - PB_0$	Port-B pins
$PC_7 - PC_0$	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 $\overline{RD}$, $\overline{WR}$ 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 → select any one of the 4 internal devices

A_1	A_0	Device Selected
0	0	Port A
0	1	Port B
1	0	Port C
1	1	Control Register

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

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

$\overline{CS}$ - Chip select signal
(when $\overline{CS}$ 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

Group A → Port-A ($PA_0 - PA_7$) and Port-C upper ($PC_7 - PC_4$) and its control circuit.

Group B → Port-B ($PB_0 - PB_7$) and Port-C lower ($PC_3 - PC_0$) and its control circuit.

✓ 3 Operating Modes: Mode-0 → Simple I/O port.
Mode-1 → Handshake I/O port.
Mode-2 → Bidirectional I/O port.

	Port A	Port B	Port C
Act as	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 or b) 2 nos. of 4 bit i/p or o/p parallel port or c) individual pins can be set or reset for various applications
Programmed to work in	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
1. As 8 bit i/p or o/p parallel port in Mode-0.
2. As two nos. of 4-bit i/p or o/p parallel port in Mode0
3. Individual pins of port-C can be set or reset for various control applications.

✓ If Port-A is programmed in Mode-1/mode-2 & Port-B in Mode-1,
then ✓ some of the pins of port-C —used for→ handshake signals
✓ Remaining pins —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)

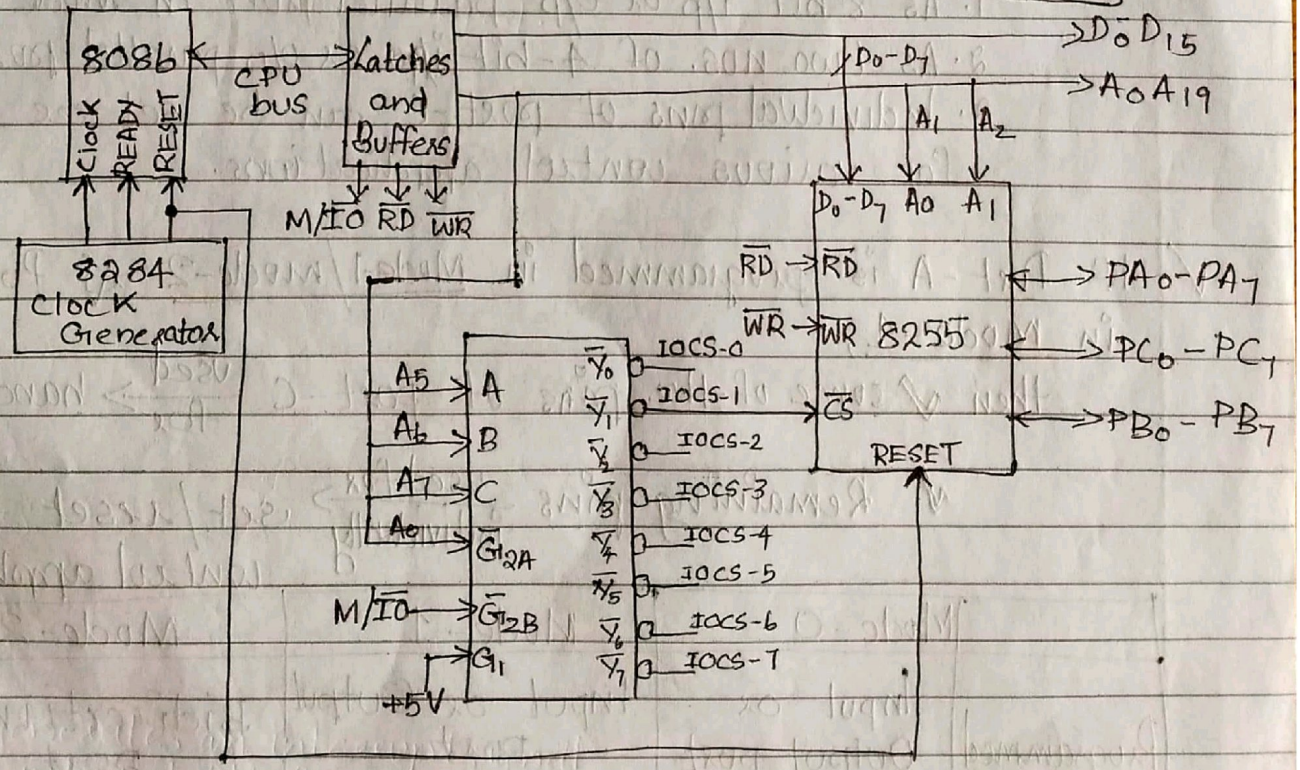
* Bidirectional port → The processor can perform both read & write operations with an I/O device connected to a port.

	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, Hexakeypad, LEDs and 7-segment LEDs to the Processor.	Handshake* (port-C) & Interrupt** Capability	→ Handshake capability (5 pins of PortC) → 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

Decoder input lines → A_5 A_6 A_7 address lines of 8086.

A_0 and M/$\overline{IO}$ → Used to enable 3-to-8 decoder.

Decoder Ouput lines (IOCS-1, ... IOCS-7) → Chip select signal ($\overline{CS}$) (IOCS-1) o/p line "used to select 8255."

Internal Device	Binary Address							
	A_7	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

→ A_1 and A_2 of 8086 are connected to A_0 and A_1 of 8255 respectively ⟹ to provide Internal Address.

→ Dataline D_0-D_7 of 8086 are connected to D_0-D_7 of 8255.
$\overline{RD}$ and $\overline{WR}$ signals of 8255 are connected to $\overline{RD}$ and $\overline{WR}$ 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

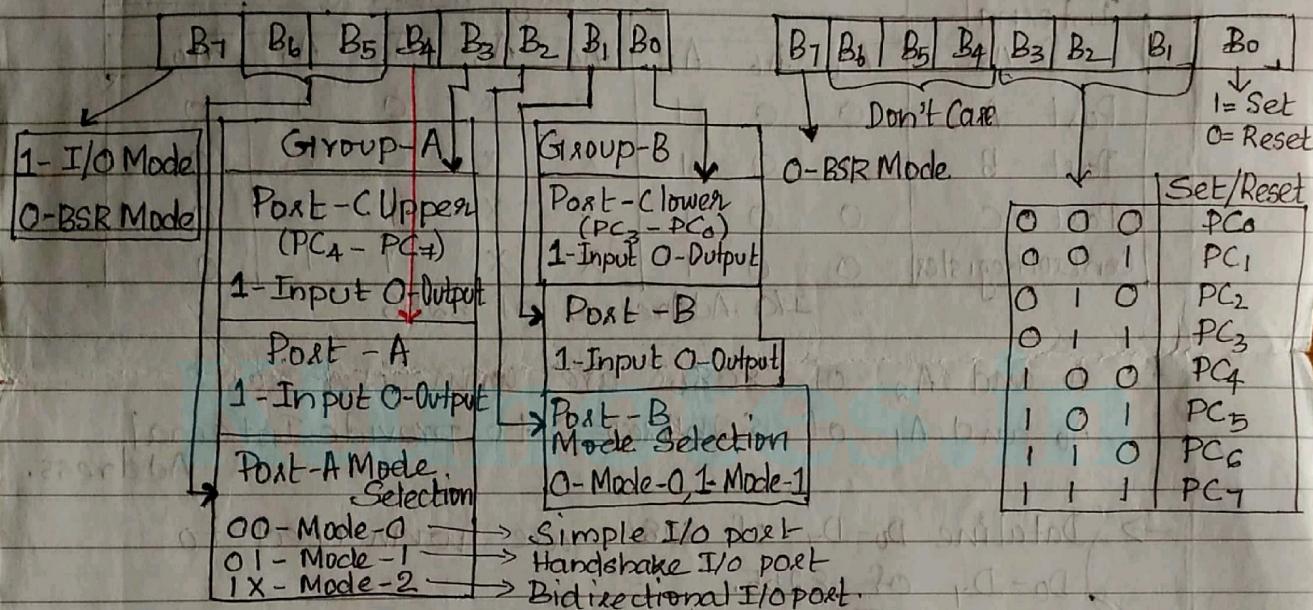
- Interrupt driven data transfer Scheme
- Checking the status (Status word) of 8255.

[Interrupt Controller 8259 has to be interfaced to s/m & 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 high

When the port is ready for data transfer

- 8255 generates an "interrupt signal" if the internal flip flops ($INTE_A$ & $INTE_B$) are enabled using BSR control word. ⇒ it interrupts the 8086 through NMI/INTR pin for read/write operation.
- the processor executes a read/write cycle.

⇒ 8255 has 2 Internal flip flops

$INTE_A$ — Interrupt enables for Port-A
- * Enabled by setting PC_4 to high using BSR control word.
- * Disabled by resetting to Zero.

$INTE_B$ — Interrupt enables for Port-B
- * Enabled by Setting PC_2 to high using BSR control word.
- * 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_7	B_6	B_5	B_4	B_3
I/O	I/O	IBF_A	$INTR_{AL}$	$INTR_A$

Status of Port-A (output port)

$\overline{OBF_A}$	$INTE_A$	I/O	I/O	$INTR_A$

Port A Status

Port-B status

Status of Port-B (input port)

B_2	B_1	B_0
$INTE_B$	IBF_B	$INTR_B$

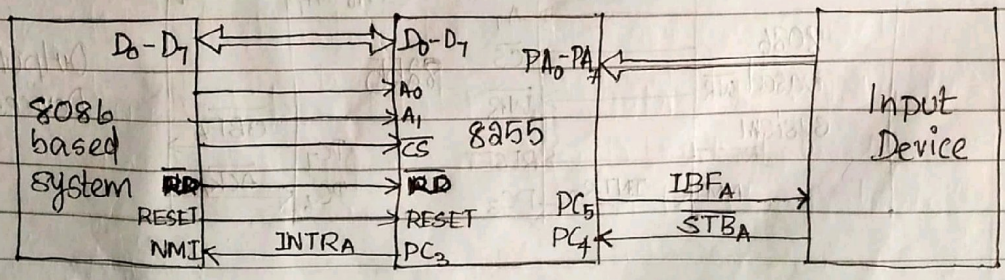
Status of Port-B (output port)

$INTE_B$	$\overline{OBF_B}$	$INTR_B$

* **INTERRUPT METHOD**

⇒ @ 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"



$\overline{STB}$ - Strobe
IBF - Input Buffer Full
OBF - Output Buffer Full
INTR - Interrupt Request

Working

1. Input device checks IBF_A signal, if it is low then the i/p device places the data on the port lines PA_0-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 $INTR_A$ is set high.

⇩

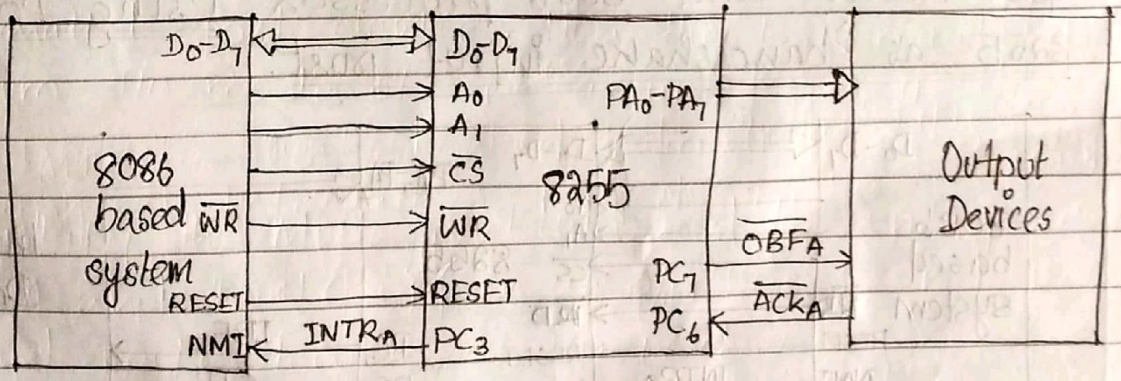
3. When $INTR_A$ goes high, the processor is interrupted through NMI pin to execute a subroutine, for reading the data from the port.

⇩

4. For a read operation, the processor asserts $\overline{RD}$ low and the $INTR_A$ is resetted to low.
After read operation, the $\overline{RD}$ become high. Then the IBF is asserted low and the input device can send the next data.

⇒ (b) 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 $\overline{WR}$ low. At the rising edge of $\overline{WR}$ after write operation to the port, both $INTR_A$ and $\overline{OBF_A}$ are asserted low by the 8255.

⇩

③ When the $\overline{OBF_A}$ 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 $\overline{ACK_A}$ low.

⇩

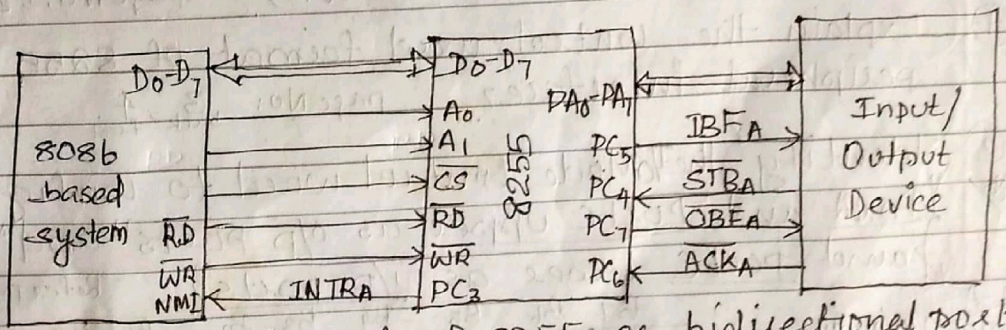
⑤ When $\overline{ACK_A}$ is low, $\overline{OBF_A}$ is asserted high by the 8255.
After reached the $\overline{ACK_A}$ in the 8255, it asserted $\overline{ACK_A}$ high, then the $INTR_A$ is set (asserted high).

⇩

⑥ When $INTR_A$ goes high, the processor is interrupted through NMI input pin "to enewte a Interrupt service routine" inorder to load next data in the output port.

ⓒ 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 I/O 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

4 marks

Port-A
- Simple I/O output
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