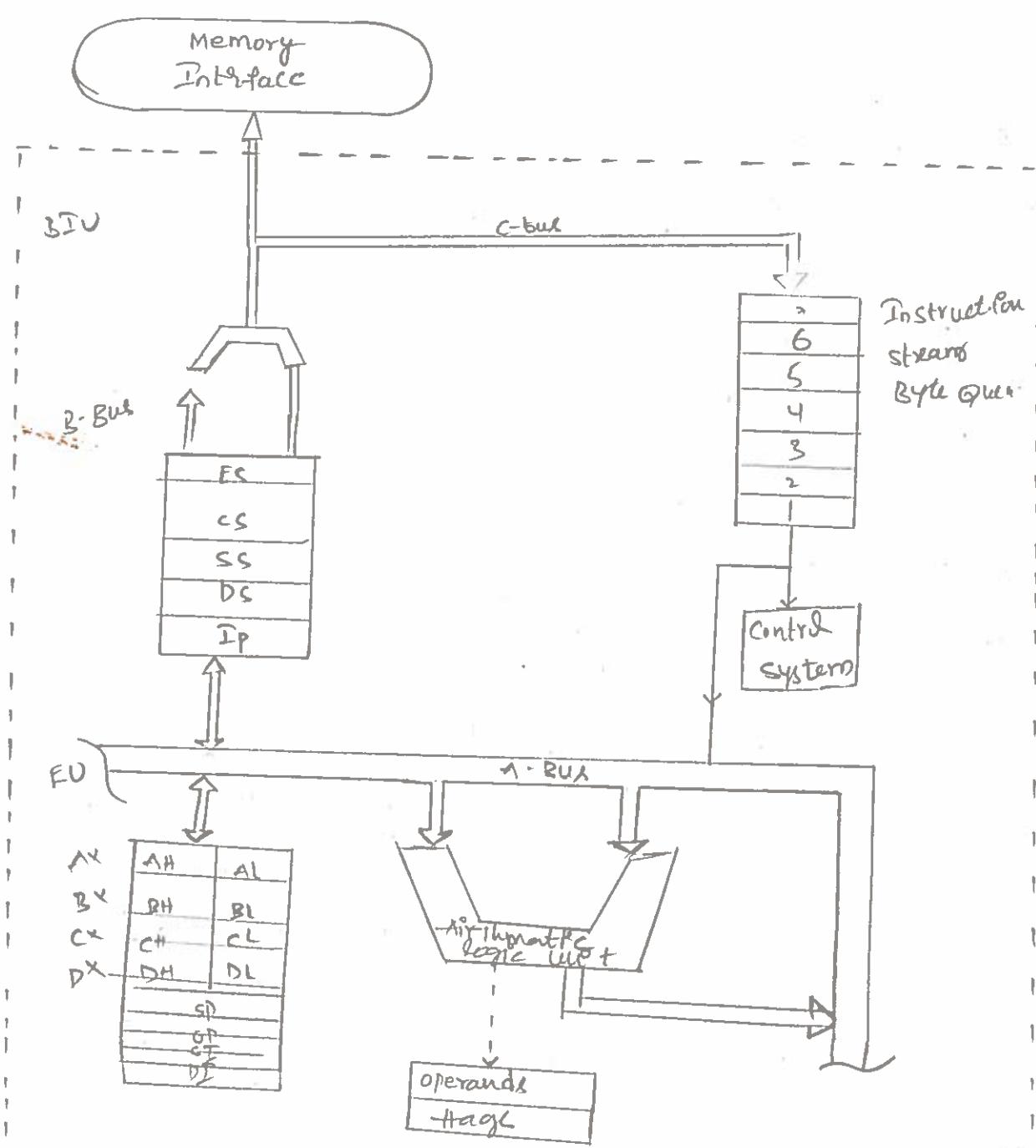


8086 Architecture8086 Architecture - Function diagram :-

The block diagram of 8086 internal architecture is shown in below. It is internally divided into two separate functional units these are

- * Bus interface unit (BIU)
- * Execution unit (EU)

These two functional units can work simultaneously to increase system speed and hence the throughput. Throughput is a measure of number of instructions executed per unit time.



BUS Interface unit:-

The bus interface unit is the 8086's interface to the outside world. It provides a full 16-bit bi-directional data bus and 20-bit address bus. The bus interface unit is responsible for performing all external bus operations.

functions of Bus Interface unit:-

- * It sends address of the memory.
- * It fetches instruction from memory.
- * It reads data from memory.
- * It writes data into memory.
- * It supports instruction queuing.
- * It provides the address relocation facility

To implement these functions the BIU contains the instruction queue, segment registers, instruction pointer, address summer and bus control logic.

Execution Unit [EU]:-

The execution unit of 8086 tells the BIU from where to fetch instructions or data, decodes instructions and executes instructions. It contains

- It
 - * Control circuitry
 - * Instruction decoder
 - * Arithmetic logic unit
 - * Flag Registers
 - * General purpose Registers
 - * Pointers and Index Registers.

Control system, instruction Decoder , ALU:-

The control system in the EU directs the internal operations. A decoder in the EU translates the instructions fetched from memory into a series of action which the execution unit performs. ALU is 16-bit. It can add, subtract, AND, OR, XOR, increment, decrements, complement and shift binary numbers.

Flag Register:-

A flag is a flip-flop which indicates some condition produced by the execution of an instruction operations of the execution unit. The flag register contains nine active flags.

General purpose Registers:-

The 8086 has four 16 bit general purpose registers labeled AX, BX, CX, DX. Each 16-bit general purpose register can be split into two 8-bit registers (the letters L and H specify the lower and higher bytes of a particular register).

Pointers and index registers:-

All segment registers are 16-bit. But it is necessary to put 32-bit address (physical address) on the address bus. The pointers and index group consists of instruction pointer, stack pointer, base pointer, source index and destination index.

-o-

Registers Organization:-

The 8086 has set of powerful registers there are

- * General purpose Registers
- * Segment registers.
- * Flag registers
- * Pointers and index registers.

General purpose Registers:-

	15	8 7	0
AX	AH	AL	
BX	BH	BL	
CX	CH	CL	
DX	DH	DL	

(a) General purpose registers.

General purpose registers are as they are,

- (1) Accumulator register (AX)
- (2) Base register (BX)

(3) Counter register (Cx)

(4) Data Register (Dx)

Each 16-bit general purpose register can be split into two 8-bit registers. The letters L and H specify lower and higher bytes of a particular register. The letter X is used to specify the complete 16-bit register.

The general purpose registers are either used for holding data, variables and intermediate results temporarily. They can also be used for storing offset address for some particular addressing modes.

These registers Ax is used as 16-bit accumulator. The register Bx is also used as offset storage for generating physical address in case of certain addressing modes. The register Cx is also used as a default counter in case of string and loop instruction.

Segment Registers:-

1M byte Physical Memory	<table border="1"><tr><td>CS</td><td>code segment</td></tr><tr><td>DS</td><td>Data segment</td></tr><tr><td>ES</td><td>Extra segment</td></tr><tr><td>SS</td><td>Stack segment</td></tr></table>	CS	code segment	DS	Data segment	ES	Extra segment	SS	Stack segment
CS	code segment								
DS	Data segment								
ES	Extra segment								
SS	Stack segment								

The physical address of the 8086 is 20 bits wide to access 1M byte memory location. 1M byte of memory is divided into 4 segments, with a maximum size of a segment as 64k bytes. The 4 active segment registers are provided by the bus interface unit of the 8086. These 4 registers are code segments, data segment, extra segment & stack segment registers.

These are used to hold the upper 16-bits of the starting addresses of the 4 memory segments. Unit-1, 4th

on which 8086 works at a particular time.

Functions of segment Register:-

1. The CS register holds the upper 16-bits of the starting address of the segment from which the BIU is currently fetching the instruction code byte.
 2. The SS register is used for the upper 16-bits of the starting address for the program stack.
 3. ES register and DS register are used to hold the upper 16-bits of the starting address of the two memory segments which are used for data.

Flag Registers:-

A flag is a flip-flop which indicates some condition produced by the execution of instruction operations on execution unit. The flag register contains nine active flags.

six of them are used to indicate some condition produced by instruction. They are

(1) Carry flag (CF) :-

In case of addition this flag is set if there is a carry out from the MSB. The carry flag also serves as a borrow flag for subtraction. In case of subtraction it is set when borrow is needed.

(2) Parity flag (PF) :-

Q) Parity flag(PE):
It is set to 1 if the result of byte operation contains an even number of ones, otherwise it is zero.

(3) Ambilay flag (AF) :-

the flag is set if there is an overflow out of bit 3 i.e carry from D3 bit to D4 bit.

This flag is used for BCD operations and it is not available for programmer.

D₇ D₆ D₅ D₄ D₃ D₂ D₁ D₀

4) Zero flag (ZF) :-

The zero flag sets if the results of operation in ALU is zero and flag resets if the result is non-zero.

5) Sign flag (SF) :-

After the execution of arithmetic operations if the MSB of the result is 1, the sign bit is set. Sign bit 1 indicates the result is negative otherwise it is positive.

6. Overflow flag :-

This flag is set if result is out of range. For addition this flag is set when there is a carry into the MSB and no carry out of the MSB or vice-versa. For subtraction, it is set when the MSB needs a borrow and there is no borrow from the MSB or vice-versa.

The 3 remaining flags are used to control certain operations of the processor.

① Trap flag (TF) :-

One way to debug a program is to run the program one instruction at a time and see the contents of used registers and memory variables after execution of every instruction. This process is called "single stepping" through a program. Trap flag is used for single stepping through a program. If set, a trap is executed after execution of each instruction. Thus programmers can easily identify and correct errors in the program.

② Interrupt flag (IF):-

(4)

It is used to allow the interruption of a program if set, a certain type of interrupt can be recognized by the 8086; otherwise these interrupts are ignored.

③ Direction flag (DF):-

It is used with string instruction. If $DF = 0$, the string is processed from its beginning with the first element having the lowest address. Otherwise, the string is processed from the high address towards the low address.

Pointers and index Registers:-

The pointer registers are

- * Instruction pointer
- * Base pointer
- * Stack pointer

These pointer registers are associated with code data and stack segments. The index registers DI and SI are used as a general purpose registers as well as for offset storage in case of indexed, based indexed and relative base indexed addressing modes.

Stack pointer:-

The stack pointer register contains the 16-bit offset from the start of the segment to the top of stack. For stack operation, physical address is produced by adding the contents of stack pointer register to the segment base address in SS .

Base pointer:-

We can use the BP register instead of SP for accessing the stack using the based addressing mode.

Source index:-

SI can be used to hold the offset of a data word in the data segment.

Destination index:- The ES register points to the extra segment in which data is stored.

Memory Segmentation:-

Two types of memory organisations are commonly used. These are linear addressing and segmented addressing. In linear addressing the entire memory space is available to the processor in one linear array. The available memory space is divided into "chunks" called segments. Such memory is known as segmented memory.

Advantages of Memory Segmentation:-

- (1) It allows the memory addressing capacity to be 1M byte even though the address associated with individual instruction is only 16-bit.
- (2) It allows instruction code, stack, and portion of programs to be more than 64KB long by using more than one code, data, stack segment and extra segment.
- (3) It facilitates use of separate memory areas for program, data, and stack.
- (4) It permits a program or its data to be put in different areas of memory each time the program is executed.

Programming Model:-

The program is a set of instructions written in a systematic order to perform a specified task. The program developed using assembly language instruction set of a processor is called as assembly language program.

Procedure for executing an assembly language program using MASM.

1. Invoke the MS DOS (start → run → type cmd → click on ok)
2. Type the path of the directly (folder in which MASM loaded at command prompt).
3. Invoke the editor by typing word 'edit' at the folder in which MASM is loaded.
4. Program editor window will appear now. Type the assembly language source program in the editor window and then save it with appropriate name with .asm extension.

5. Exit the program editor you will come back to the directory(**D:**) folder in which the MASM is located.
6. Type MASM and then press the enter key. Then it asks for source filename.
7. Type the source filename with **.asm** extension which you have developed using editor and then press enter key. Then it asks for object file name.
8. Type the same file name with **.obj** extension and then press enter key.
9. Press the enter key for **.lst** and **crf** requests. Now you will come back to the directory in which the MASM is located.
10. Now assembler displays the list of errors and warnings. correct the program if any errors detected. Repeat the steps 3 through 13 until all the errors and warnings indicated by the assembler are cleared.
11. Type link and then press enter key. Then it asks for object file name which is to be linked with library functions and other modules.
12. Type the object filename **.obj** extension which is obtained in step 8 and then press enter key.
13. To execute the program and check the results type **debug** file name with extension and then press enter key. use different debug commands at debug command prompt (-) for different purpose.

Memory address physical memory organization:-

A logical address in the 8086 system is described by a segment and offset. Both the segment and offset are 16-bit size. This is because all registers and memory locations are 16-bit size.

Physical address calculation:-

The source of the base address is always the code segment register and the source of the offset is always the instruction pointer.

- The content of the code segment register is first shifted 4 bit left or content of segment register is multiplied with 10H.
- Then content of the instruction pointer (IP) is added to the content of the code segment register.
- Physical address $CS * 10H + (IP)$

(1) To calculate the physical address of the bus interfacing unit of code segment, Instruction pointers are 142CH & 31H.

$$\begin{aligned}
 \text{Physical address} &= \text{segment Register} \times 10H + \text{offset Register} \\
 &= CS \times 10H + IP \\
 &= 142CH \times 10H + 31H \\
 &= 142C0H + 31H \\
 \text{Physical address} &= 142F1H
 \end{aligned}$$

(2) Calculate the physical address of Bus interfacing unit of stack segment and stack pointers are 800H & 0001H.

$$\begin{aligned}
 \text{Physical address} &= \text{segment Register} \times 10H + \text{offset register} \\
 &= 800H \times 10H + 0001H \\
 &= 8000H + 0001H \\
 &= 80001H
 \end{aligned}$$

(3) What would be value of offset Register the physical address is 101H and segment Register is 132CH.

$$\begin{aligned}
 \text{Physical address} &= \text{segment Register} \times 10H + \text{offset Register} \\
 \text{Offset Register} &= \text{physical address} - SR \times 10H
 \end{aligned}$$

$$= 132CH - 101 \times 10H$$

$$= 132CH - 1010H$$

Offset Register	= 31CH
-----------------	--------

Signal Description of 8086:-

(6)

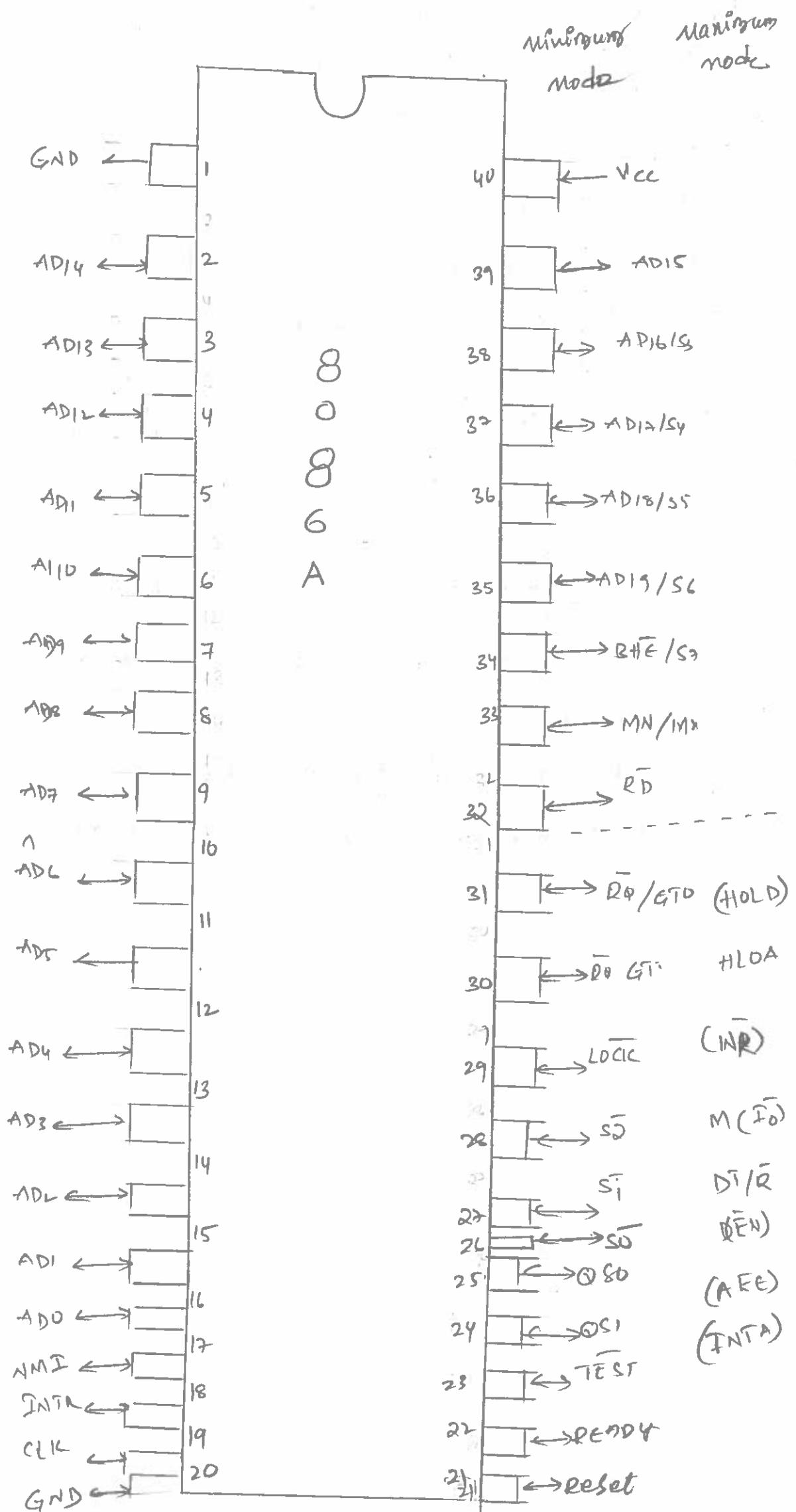
In order to implement many situations in the micro-computer system the 8086 and 8088 has been designed to work in two operating modes.

1. Minimum mode.
2. Maximum mode.

The minimum mode is used for a small systems with a single processor and maximum mode is for medium size to large systems, which often include two or more processors. The 8086 is a 16-bit microprocessor with a 16-bit data bus, and the 8086 pin connections AD₀-AD₁₅ as shown in below.

The 8086 signals can be categorised in three groups

- * signals having common functions in both minimum and maximum modes.
- * signals having special functions for minimum mode.
- * signals having special functions for maximum mode.



signals having common functions in both minimum and max. modes (T)

1) AD₁₅ - AD₀:-

Act as address bus during the first part of machine cycle and data bus for the remaining part of the machine cycle.

2) A₁₉/S₆ - A₁₆/S₃:-

pin no. 35 to 38 has dual function i.e., the pins are used as address pins as well as status pins.

3. BHE /s₇:- Bus High Enable

The pin is low during the first part of the machine cycle indicates that atleast byte of the current transfer is to be higher order byte A₁₅ - A_{D8} otherwise the transfer is lower order byte (AD₀ - AD₇). The status pin s₇ is output during the later part of the machine cycle.

4. NMI:-

It is abbreviated as non Maskable interrupt. It is a positive edged trigger. Non maskable interrupt request. The pin no is 17.

5) INTR:-

It is a level trigger maskable interrupt request. It is sampled during the last clock cycle of each instruction. The pin no is 18.

6. CLK:- Clock is a square wave is used to coordinate the different internal units of the processor. The 8086 have external clock signal is applied b/w pin 19 & pin 20.

7. RESET:-

Pin 21 is Reset pin when this pin is goes high it will restart the system with the period of 4 clock pulses. In this high state the execution of instructions are started from FFFF0H.

8. READY:-

If this signal is low the 8086 micro processor enters into wait state. This signal is used primarily to synchronize the slower peripherals of microprocessor pin no is 22. unit-1, 13/34

9. TEST:- This signal is only used for weight instruction. The 8086 enters into a weight stage after execution of weight instruction until a low signal on the TEST pin.

The TEST signal is synchronized internally during each clock cycle during leading edge of the clock cycle.

10. RD (Output):-

It is a control pin which controls the read operation when it goes low the selected memory or I/O device is read.

11. MN ($\overline{M_X}$ (input))

The 8086 can be configured into either minimum mode or maximum mode using the pins. The pin is high for minimum mode, low for maximum mode.

* Signals having special functions for minimum mode:-

INTA:- Interrupt Acknowledgement.

It is used as o/p pin. This pin indicates recognition of an interrupt request. It consists two -ve going pulses. In two consecutive bus cycles, the first pulse informs the interface and the second pulse informs the interface is to send the interrupt type to the processor.

ALE:- Address latch enable

This pin provides demultiplexing operation of 8086 MP. i.e ADO to AD₁₅.

DEN:- Data enable pin.

It is used as o/p pin. This pin indicates the transmitting & Rx of the data in the CPU i.e, ready to send or ready to receive.

DT/R:-

It is abbreviated as data transmit or Receive. It is used as o/p pin. This pin is used to control the data flow direction. This pin is high, it indicates transmitting the data when the pin is low, it indicates the receiving the data.

M/I/O:-

It is used to distinguish memory data transfer, when this pin is high memory data is transferred, when this pin is low So data is transferred.

WR:-

It is also a control signal which controls the WRITE operation when it is goes low. The data on the bus is return into the memory or So device pin no is 89.

HOLD(C1P), HLDA(C0P):-

A high on hold pin indicates that another master DMA is requesting to take over the system bus on receiving hold signal processor o/p's HLDA signal is high is acknowledgement At the same time the processor tristate the system bus. A hold pin is low the system bus controls the processor.

* Signals having special functions for maximum mode.

Q_{S1}, Q_{S0}:-

It is used for o/p signal .the two o/p signals reflects the status of the instruction Queue .This state indicate the activity in the queue during the clock cycle.

Q _{S1}	Q _{S0}	Status
0	0	No operation
0	1	first byte of an opcode
1	0	Queue is empty
1	1	subsequent byte of an opca

$\bar{S}_0, \bar{S}_1, \bar{S}_2$:-

This 3 status signals indicate the type of transfer to be taken to place during the current bus cycle.

S_2	S_1	S_0	Machine cycle
0	0	0	Interrupt . Ack
0	0	1	I/O Read
0	1	0	I/O write.
0	1	1	Halt
1	0	0	Instruction fetch
1	0	1	Memory read
1	1	0	Memory write
1	1	1	Inactive.

LOCK:-

This signal indicates an instruction with a lock prefix is being executed and the bus is not to be used by another processor.

$\overline{RQ}[G_I]$, $\overline{RQ}[G_O]$:- [Bus Request | Bus Grant]

In the minimum mode the hold & HLDA pins are replaced by $\overline{RQ}[G_O]$ and $\overline{RQ}[G_I]$ signals.

Timing Diagrams:-

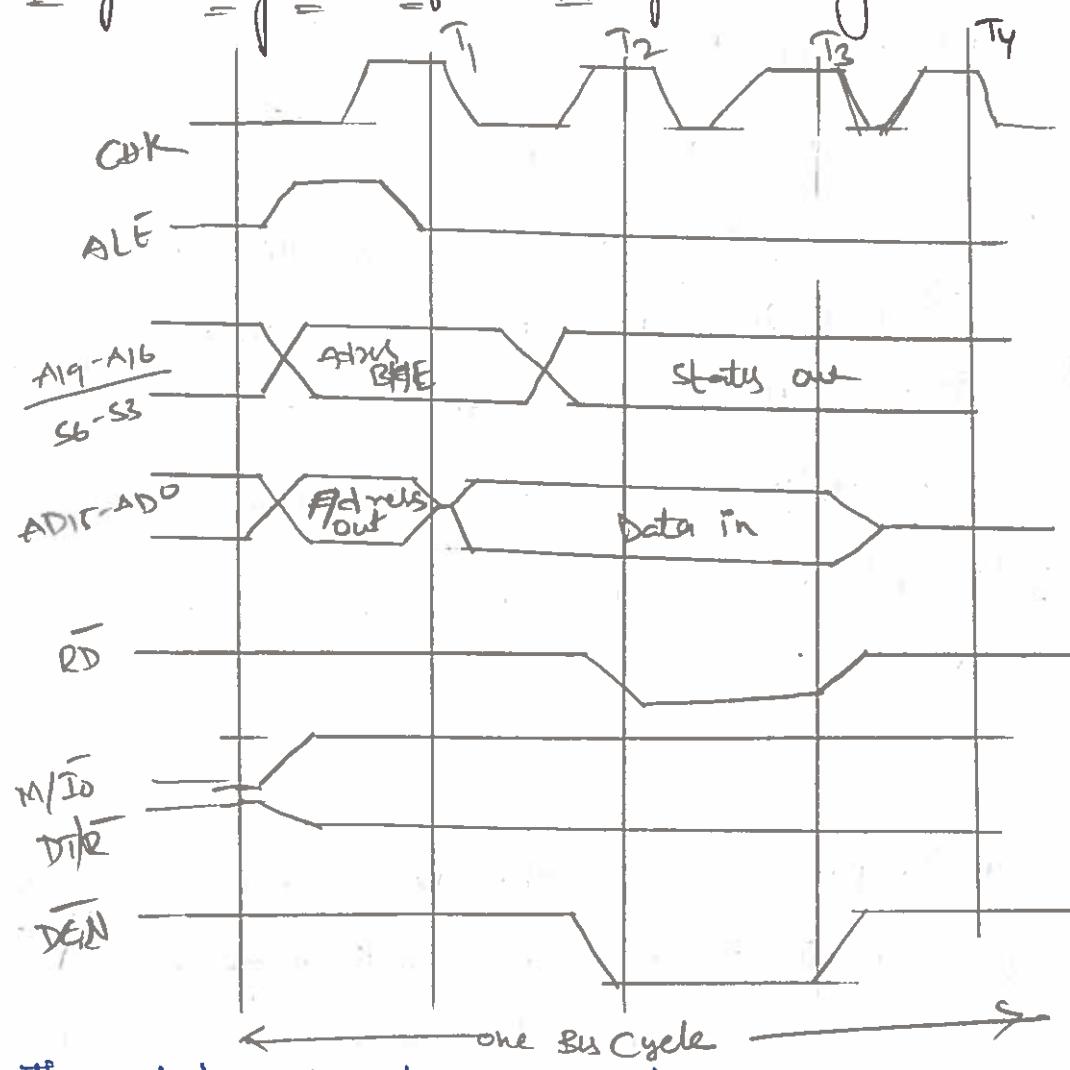
Bus cycle:- The basic operations such as fetching of code and data decoding of instructions and execution performed by the CPU are called bus cycle.

The bus cycles are classified into 5 types

- (1) Memory read cycle.
- (2) Memory write cycle
- (3) I/O Read cycle
- (4) I/O write cycle
- (5) Interrupt Acknowledgement cycle.

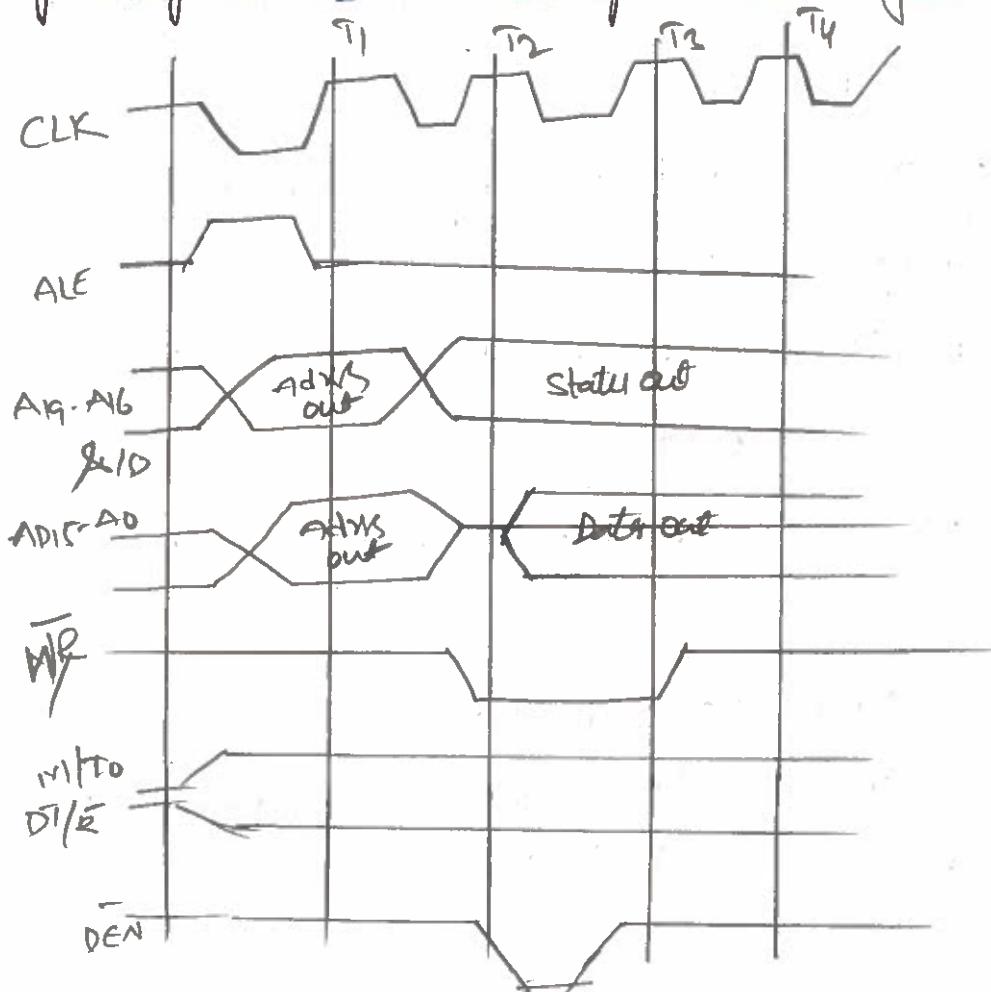
The graphical representation of these cycles are called as timing diagram. (9)

Timing diagram of Memory Read cycle:-



* The data transfer takes place during T3 and T4 clock periods.

Timing diagram of memory write cycle:-



Interrupt of 8086 :-

Interrupt is a signal given by the external or internal devices to the microprocessor so to request the microprocessor to perform a particular task.

Whenever a external device needs some service from the microprocessor it places a request in the form of interrupt. On receiving interrupt the microprocessor temporarily stops its work and accepts the request and external device for execution is called interrupt service subroutine (ISS) after completing interrupt service the microprocessor returns to its original work.

Interrupt Response in 8086 :-

The 8086 microprocessor checks the status of interrupt at the end of each instruction cycle. If any interrupt request is detected. The processor immediately response from the following steps.

- 1) The stack pointer decremented by two and flag registers pushed to stack memory.
- 2) Interrupt system is desirable by clearing interrupt flag.
- 3) The trap flag is resetted.
- 4) The stack pointer is decremented by 2 & the base address is pushed to stack memory.
- 5) The stack pointer is decremented by 2 & offset address is pushed to stack memory.
- 6) The physical address of interrupt service Routine is calculate by multiplying the content of code segment register by 10H and adding to the content of instruction pointer.
- 7) The processor execute the ISR to service the interrupt.

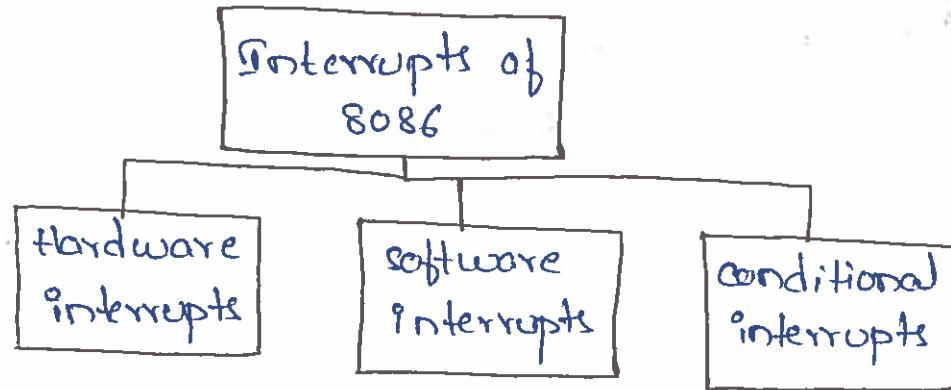
8) After completion of the task, the ISR terminated by ⑩ `IRET` instruction. whenever this instruction is executed, the stack is popped to instruction pointer and code segment register and flag register one word by one word. After completion of pop operation, the `SP` is incremented by two.

9) After completing the interrupt service the microprocessor returns back where from it has left.

Types of interrupt of 8086:-

Interrupts of 8086 are classified in three ways

- (1) · hardware interrupts
- (2) · software interrupts
- (3) · conditional interrupts (automatic interrupts)



Hardware interrupt

The 8086 has two interrupt pins `INTR` (18th) and `NMI`

(17th) the interrupt initiated by external hardware by sending a signal to the interrupt pin (17th or 18th) of the 8086 is called hardware interrupt.

Software interrupt:-

The interrupt initiated by inserting appropriated instructions at desired locations in a program. While executing program, if a software interrupt instruction is encountered the processor immediately initiates the interrupt. The 8086 has 256 software interrupt instructions. The interrupt can be initiated either by executing `INT n` instruction where `n` is the type number (0 to 255).

Conditional Interrupts:-

The conditional interrupts also referred as automatic interrupts or dedicated interrupts. An interrupt caused from some condition produced in the 8086 by the execution of an instruction. Example is divide by zero interrupt the processor initiate the interrupt if the result of a division operation is too large to fit in the destination register and this interrupt is non-maskable interrupt some other examples of automatic interrupts are :-

- (i) Single step interrupt
- (ii) Break point interrupt
- (iii) Overflow interrupt etc.

* UNIT - I *

(1)

Instruction Set and Assembly language programming of 8086

Instruction formats:-

Proper way of representing the instruction is called instruction format. The format of an instruction is usually represented in a rectangular box.

The most common fields found in instruction formats are

- (i) Operation code field.
- (ii) Address field
- (iii) Operand (effective address)

opcode field	address field	operand
--------------	---------------	---------

Types of instruction formats:-

The instruction formats are 4 types they are,

- (i) Zero - addressing instruction format.
- (ii) One - address instruction format.
- (iii) Two - address instruction format.
- (iv) Three - address instruction format.

Zero address instruction:-

→ If an instruction contains no address field is known as zero address instruction.

→ In this type of instructions, all the operand defined in the stack related instructions are zero address instructions.

Example:-

PUSH and POP instructions.

opcode	address field	operand
PUSH A	-	-

One address Instruction:-

- If an instruction contains one address field is known as one address instruction.
- In this type of instructions accumulator used as implied register for all data manipulation operations.
- All the operations are performed between accumulator and a memory operand.

Example:-

ADD B $AC \leftarrow AC + M(B)$

MULT $AC \leftarrow AC \times M(T)$

Two address Instructions:-

- If an instruction contains two address fields is known as two address instruction
- In this type of instruction each address field represents either processor register or a memory.
- General purpose or register organisation purpose this type of instruction format is used.

Example:-

MOV R₃, A $R_3 \leftarrow M(A)$

ADD R₂, B $R_2 \leftarrow R_2 + M(B)$

Three address instruction:-

- If an instruction contains three address fields is known as three address instruction.
- The address field may be a processor register or a memory operand, first address field represents destination field and remaining address fields represents the source fields.
- It occupies more space, this is the disadvantage of this instruction.

SUB R₁, A, B $R_1 \leftarrow M(A) - M(B)$

MUL X, R₁, R₂ $M(X) \leftarrow R_1 \times R_2$

Addressing mode:-

Instruction consisting two parts i.e, opcode and operands

The way in which an operand specified in the instruction is called its addressing mode.

The various types of addressing modes of 8086 are discussed below.

Immediate addressing:-

In this addressing mode 8-bit or 16-bit data is specified as a part of instruction

instruction
Data

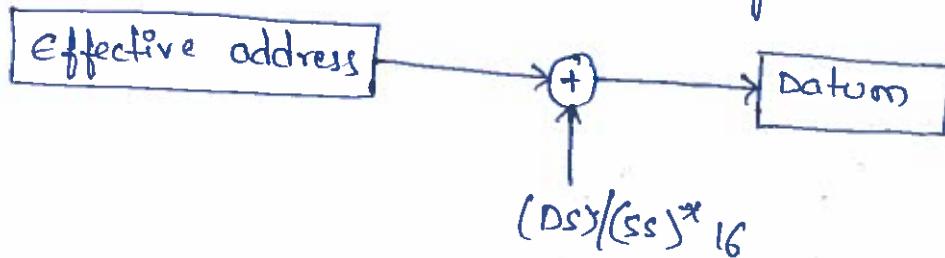
Register Addressing:-

In this addressing mode name of a register is a part of the instruction. The register will hold the data.



Direct Addressing:-

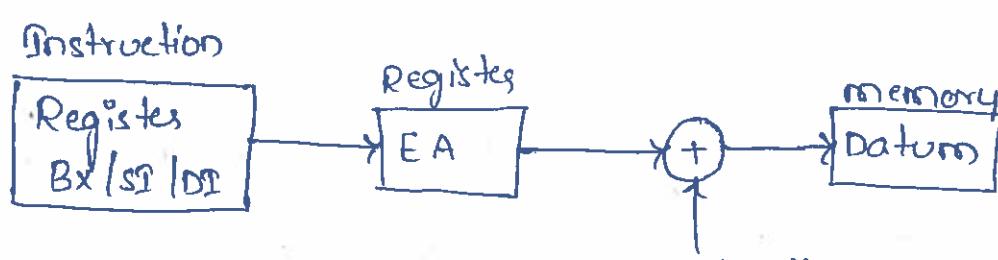
In this addressing mode the effective address of the operand is specified in this construction. The physical address of the operand is evaluated as given below.



Register Indirect Addressing:-

In this addressing mode name of the register which holds the effective address of the operand will be specified in the instruction. BX, SI & DI registers are used in this address mode to hold the effective address of the operand.

The register name is specified within the square brackets to indicate the indirect addressing. The evaluation of the memory address in this addressing mode is as follows.

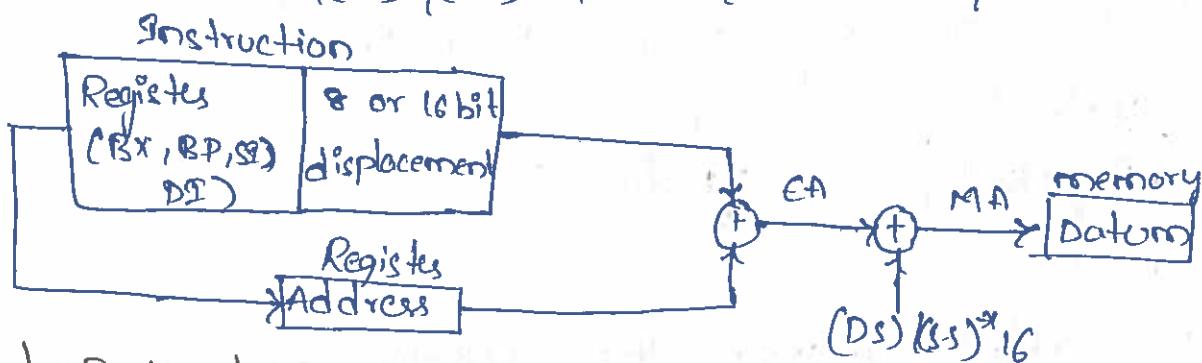


Register Relative Addressing:-

In this addressing mode the effective address of operand is specified in the instruction through a register and an 8 or 16-bit displacement.

The effective address of operand is equal to sum of the content of registers and the displacement. The registers used in this addressing are BX, BP, SP and DI.

$$EA = (BX) / (BP) / (SP) / (DI) + 8\text{ bit} / 16\text{-bit displacement.}$$

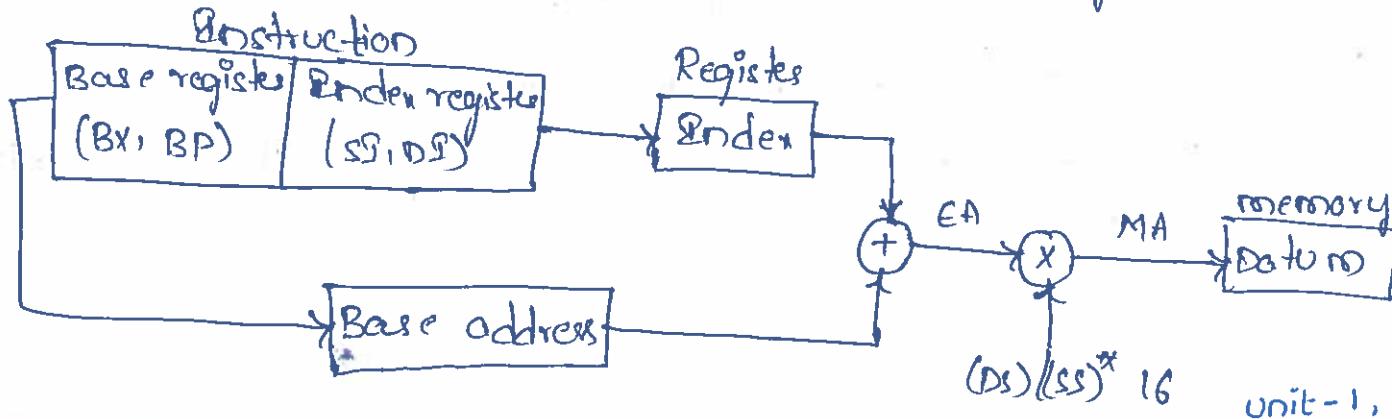


Based Indexed Addressing:-

In this addressing mode the effective address of the operand is specified through a base register (BX or BP) and an index register (SP or DI). The effective address is the sum of content of base register and index register i.e.,

$$EA = \boxed{(BX)} \quad \boxed{(SP)} + \boxed{(BP)} \quad \boxed{(DI)}$$

If BX holds the base of the effective address data segment register is used to evaluate the 20 bit physical address of the operand. If BP holds the base of the effective address, stack segment register is used to evaluate the 20 bit physical address of operand.



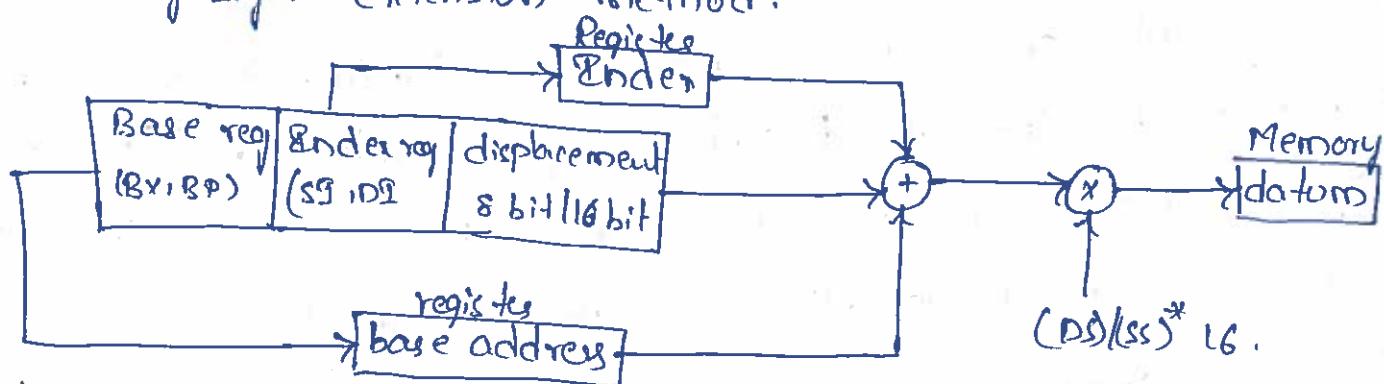
Relative Based Indexed Addressing:-

In this addressing mode of the effective address of the operand is specified in the instruction through a base register, index register and an 8-bit or 16-bit displacement. The effective address of the operand is the sum of the 8-bit or 16-bit displacement content of base registers and the content of index registers. The BS or BP can be used as base register. The SS or DS can be used as indexed registers.

$$EA = \boxed{(BX) \\ (BP)} + \boxed{(SI) \\ (DI)} + \boxed{8\text{-bit} / 16\text{-bit} \\ \text{displacement}}$$

If BX holds the base of the effective address data segment register is used to evaluate the 20-bit physical address of the operand.

If BP holds the base of the effective address stack segment register is used to evaluate the 20-bit physical address of the operand. If the displacement is 8-bit it will be converted into 16-bit by sign extension method.



Direct I/O port addressing:-

This addressing mode is used when the data from standard I/O mapped devices or ports are accessed. In the direct I/O port addressing the I/O port address is directly specified in the instruction.

Indirect I/O port addressing:-

This addressing mode is also used to access data from the standard I/O mapped devices or ports. However here the instruction specifies the register name (px) which holds the address of the I/O port.

Branch Addressing / Relative Addressing:-

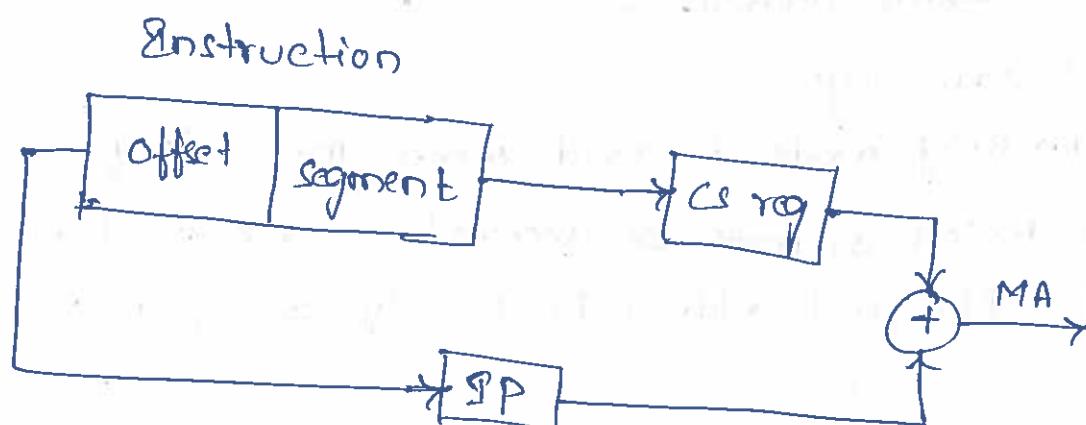
This addressing mode is used in which branch instructions. In these instructions the program control may be transferred to some other instruction of the same program depending on the type of instruction. 8-bit or 16-bit displacement will be specified in the instruction. The effective branch address is the sum of the displacement & the current contents of the instruction pointer (IP).

Intersegment Branch Addressing:-

If the instruction causes to jump of program to another code segment then it is called as intersegment branching. In this addressing mode the instruction specifies the base address of the destination code segment & the offset address in the segment.

The offset address may be specified directly or through any one of the addressing modes except immediate and register modes.

The code segment register is loaded with the base address of the segment specified in the instruction. The instruction pointer is loaded with the offset address. Offset address may be specified directly in the instruction, or it may be evaluated through the addressing mode specified in the instruction.



Implied addressing mode:-

In this addressing mode the operand is implied i.e., the instruction itself specifies the data to be operated.

Instruction Set:-

Assembly language of 8086 consists of 2 parts

- (i) Instructions and
- (ii) Assembler directives.

Instructions:-

Instructions are translated into machine codes by the assembler. These machine instructions are executed by the microprocessors. Assembler directives give directions to the assembler during the assembly process. These are not executed by the processor and they are only directions to the assembler.

Assembler directives are called as pseudo instructions.

General format of Assembler instruction:-

- Label: Mnemonic operand, operand; comment.

Label:- It is an optional identifier. Address of the 1st byte to the instruction is assigned to the label.

Mnemonic:- All instructions must contain a mnemonic. Mnemonic in the instruction indicates the type of instruction.

Operands:-

- Operands are data on which operation specified by the instruction is performed. The number of operands in the instruction depends on the type of instruction.

If an instruction contains two operands the destination operand appears first and the second operand appears next.

Comment:-

Comments in the program are optional. A comment should be preceded with semicolon. In this field programmer can comment on the program which is helpful for understanding program.

AGAIN : ADD AX,[1534H] ; AX \leftarrow AX + 1534H

↓
label

↓
Mnemonic

↓
destination
Operant

↓
source operand

↓
comment.
unit - 1, 27/34

Classification of 8086 Instruction Set:-

The instructions of 8086 can be classified into following groups.

- (1) Data transfer instruction.
- (2) Arithmetic instruction.
- (3) Logical instruction.
- (4) Program control transfer instruction.
- (5) String Manipulation instruction.
- (6) Process control instructions.

The data transfer group includes instructions for moving data between registers, register and memory, accumulator and I/O devices etc.

Data Transfer Instructions:-

- Copy the content of a register to another register.
- Copy the content of a register to a memory location or vice-versa.
- Copy the content of a register/memory location to a segment register.
- Load a register or a memory location with an immediate data.
- Exchange the content of two registers or register and memory location.
- Load effective address into a register.
- Load effective address into segment register.
- Transferring data onto stack and fetching data from the stack.
- Transferring data from accumulator to I/O ports or vice-versa.

Arithmetic Instructions:-

- signed and unsigned binary addition, subtraction, multiplication and division.
- Packed BCD addition and subtraction.
- Unpacked BCD addition, subtraction, multiplication and division.
- Increment and decrement operations.
- Comparison operation.

Logical Instructions:-

This group includes instructions to perform the logical operations such as complement, OR, AND, EX-OR and shift and rotate operations.

- * In double operand logical instructions.
- * the destination operand should not be immediate.
- * the source and destination operands must be of same size.
- * Unless the source operand is immediate, one of the operands must be a register.

Program control transfer instruction:-

This group of instructions transfer the program control to a new address from the normal sequence of instructions. The program control transfer is done either by changing the content of instruction pointer (or) by changing the content of both IP & CS.

If both IP and CS are modified, then the program control transfers to a memory location in another segment. The following instructions are discussed under this group.

- Branch instructions.
- Loop instructions.
- CALL & RET instructions.
- Software interrupt instructions.

The program control transfer instructions do not affect the flags.

Branch instructions:-

Branch instructions can be classified as conditional and unconditional.

Loop instructions:-

Loop instructions are used to execute a group of instructions no. of times as specified by the CX register. How many instructions are to be grouped is specified in the instruction by an 8-bit displacement. The 8-bit displacement must be specified using a label in the instruction. The 8-bit displacement is a signed number so it can be a positive or negative.

CALL & RET Instructions:-

CALL instruction transfers the program control to subprogram or procedure after saving the return address on the stack. If the called subprogram resides on the same code segment in which the calling main program resides, then the call is known as near call. If the called subprogram resides on another code segment, then the call is known as far call.

While executing the near call only the content of instruction pointer is pushed onto top of the stack. While executing the far call the contents of both IP & CS registers are pushed onto the stack.

Software interrupt Instructions:-

These instructions are used to call the subprograms known as interrupt service routines [ISR]. The format of software interrupt is

INT n

n = 0, 1, 2, ..., 255 is known as interrupt type number.

∴ The 8086 processor has 256 software interrupts. Each software interrupt is associated with a subprogram called as interrupt service routine (ISR).

Process Control Instructions:-

This group includes instructions to set or reset different flags and certain instructions to control the processor operation.

String Manipulation Instructions:-

A sequence of characters is called as a string. Each character is represented with an 8-bit ASCII code. The string manipulation instruction includes instructions to move, compare, scan, load and store the strings. This group also includes a REP instruction which is used as prefix to the string instructions.

Assembler directives:-

Assembly language of 8086 consists of two parts.

(i) Executable instructions.

(ii) Non-executable instructions.

→ The executable instructions are translated into machine codes by the assembler and are executed by the processor.

→ The non executable instructions are merely directions to the assembler called as pseudo codes. These non-executable instructions are called as assembler directives, not translated into machine codes and not executed by the processor.

Variable :- Variables are group of characters used in program to represent variables data or address

→ Variable should starts with Alphabet.

→ Alphabet can starts with : a to Z

A to Z

0 to 9

→ The length of string ie below 32 bits.

→ the characters should be in either uppercase /lower case.

Constants:- The data / address can be represented in binary, decimal or hexadecimal numbers. These numbers which cannot be varied are called as constants.

1. Data Definition Directive:-

This directive is used to pre-assign data (or) reserve memory locations to a variable. The format for this directive is variable-name mnemonic operand, operand, ... operand.

variable-name is the name of the variable to which the data or storage is associated.

Mnemonic determines the length of the each operand specified

in the directive. Different used are described below.

DB - each operand datum is of 1 byte length.

DW - each operand datum is of 2 word length.

DO - each operand datum is of 4 word length

DD - each operand datum is of 2 word length

DT - each operand datum is of 10 bytes length.

Operand Overrided Directive:-

The PTR directive instructs the assembler to override the default size given to an operand. If we want to override the default word size operand with byte, then we have to place the directive Byte PTR in front of the variable. If we want to override the default byte size operand with word, then we have to place the directive WORD PTR in front of the variable which is also with the address of byte size operand.

Segment and Ends directives:-

It is already learnt that the 1MB memory in 8086 is utilized in the form of segments. Different segments carry different types of data.

The SEGMENT and ENDS directives gives the information about the structure of segment to assembler.

The directive SEGMENT indicates the beginning of a segment.
The general form of code, data, stack or extra segment:
segment-name SEGMENT

Program code | data definitions | directives etc.)

segment - name ENDS

Structure of data segment:-

MYDATA - SEG SEGMENT

- Data defining statements
- And related directives.

MYDATA - SEG ENDS

Assume directive:- Every segment will have a name. In the above

example name of the data segment is MYDATA - SEG. The programmer has to inform the assembler the segment name and to which segment it is associated. This can be done through the ASSUME directive.

The form of the assume directive is

ASSUME CS : code segment name, DS : data segment name.....

ORG Directive:-

This directive is used to assign the starting address for a data segment or code segment.

END directive:-

This directive indicates the end of the assembly language code. The general form of this directive is
END label.

The label should also appear in the first instruction that is supposed to be executed.

Program -1:- 8 bit Addition

```

data segment
num1 db 10h
num2 db 20h
res db ?
data ends
code segment
assume : cs: code ,ds : data
start : mov ax,data
        mov ds,ax
        mov al,num1
        mov bl,num2
        add al,bl
        res al
        int 03
        code ends
end start

```

Program -3 8 bit multiplication

```

data segment
num1 db 10h
num2 db 03h
res db ?
data ends
code segment
assume : cs: code ,ds : data
start : mov ax,data
        mov ds,ax
        mov al,num1
        mov bl,num2
        mul bl
        res al
        int 03
        code ends
end start

```

```

assume : cs: code ,ds : data
start : mov ax,data
        mov ds,ax
        mov al,num1
        mov bl,num2
        mul bl
        res al
        int 03
        code ends
end start

```

Program 2:- 8-bit subtraction.

```

data segment
num1 db 20h
num2 db 10h
res db ?
data ends
code segment
assume : cs: code ,ds : data
start : mov ax,data
        mov ds,ax
        mov al,num1
        mov bl,num2
        sub al,bl
        mov res,al
        int 03
        code ends
end start

```

program -4 :- 8 bit division.

```

data segment
num1 db 20h
num2 db 02h
res db ?
data ends
code segment
assume : cs: code ,ds : data
start : mov ax,data
        mov ds,ax
        mov al,num1
        mov bl,num2
        div bl
        res al
        int 03
        code ends
end start

```

```

assume : cs: code ,ds : data
start : mov ax,data
        mov ds,ax
        mov al,num1
        mov bl,num2
        div bl
        res al
        int 03
        code ends
end start

```

Program -5:- 16 bit addition

```

data segment
num1 dw 0020h
num2 dw 0010h
res dw ?
data ends
code segment
assume:cs:code,ds:data
start : mov ax,data
        mov ds,ax
        mov AX,num1
        mov BX,num2
        ADD AX,BX
        mov res,Ax
        Int 03
        code ends
end start

```

Program -7:- 16-bit Multiplication

```

data segment
num1 dw 0020h
num2 dw 0003h
res dw ?
data ends
code segment
assume:cs:code,ds:data
start : mov ax,data
        mov ds,ax
        mov AX,num1
        mov BX,num2
        MUL BX
        mov res,Ax
        Int 03
        code ends
end start

```

Program -6:- 16-bit subtraction

```

data segment
num1 dw 0020h
num2 dw 0010h
res dw ?
data ends
code segment
assume:cs:code,ds:data
start : mov ax,data
        mov ds,ax
        mov AX,num1
        mov BX,num2
        Sub AX,BX
        mov res,Ax
        Int 03
        code ends
end start

```

Program -8:- 16-bit division.

```

data segment
num1 dw 0020h
num2 dw 0002h
res dw ?
data ends
code segment
assume:cs:code,ds:data
start : mov ax,data
        mov ds,ax
        mov AX,num1
        mov BX,num2
        DIV BX
        mov res,Ax
        Int 03
        code ends
end start

```