1. Explain the features of 8086 microprocessor.

Answer:-

The features of 8086 are:

- The 8086 is a 16 bit processor
- The 8086 has a 16 bit data bus
- The 8086 has a 20 bit address bus
- Direct addressing capability 1M byte of memory(2^{20})
- It provides fourteen 16 bit register
- 24 operand addressing modes
- Bit, byte, word and block operations.
- 8 and 16 bit signed and unsigned arithmetic operations including multiply and divide
- Four general purpose 16 bit registers: AX, BX, CX, DX
- Two pointer group registers: stack pointer (SP), Base pointer(BP)
- Two index group registers: source index (SI), destination index (DI)
- Four segment registers: code segment (CS), Data segment (DS), Stack segment (SS), Extra segment(ES)
- 6 Status flag and 3 control flags.
- Memory is byte addressable- each address stores an 8 bit value.
- Address can be up to 32 bit long, resulting in 4GBof memory.
- Range of clock rates: 5MHZ for8086, 8MHZ for8086-2, 10MHZ for8086-1
- Multibus system compatible interface
- Available in 40 pin plastic package and lead cerdip.

2. Explain the internal architecture of 8086 microprocessor

Answer:

The internal functions of 8086 processor are partitioned logically into two processing units

The 8086 CPU is divided into two independent functional parts,

1. **Bus Interface Unit (BIU)**
2. **Execution Unit (EU)**

✓ The BIU and EU function independently.

✓ The BIU interface the 8086 to the outside world. The BIU fetches, reads data from memory and ports, and writes data to memory and I/O ports.

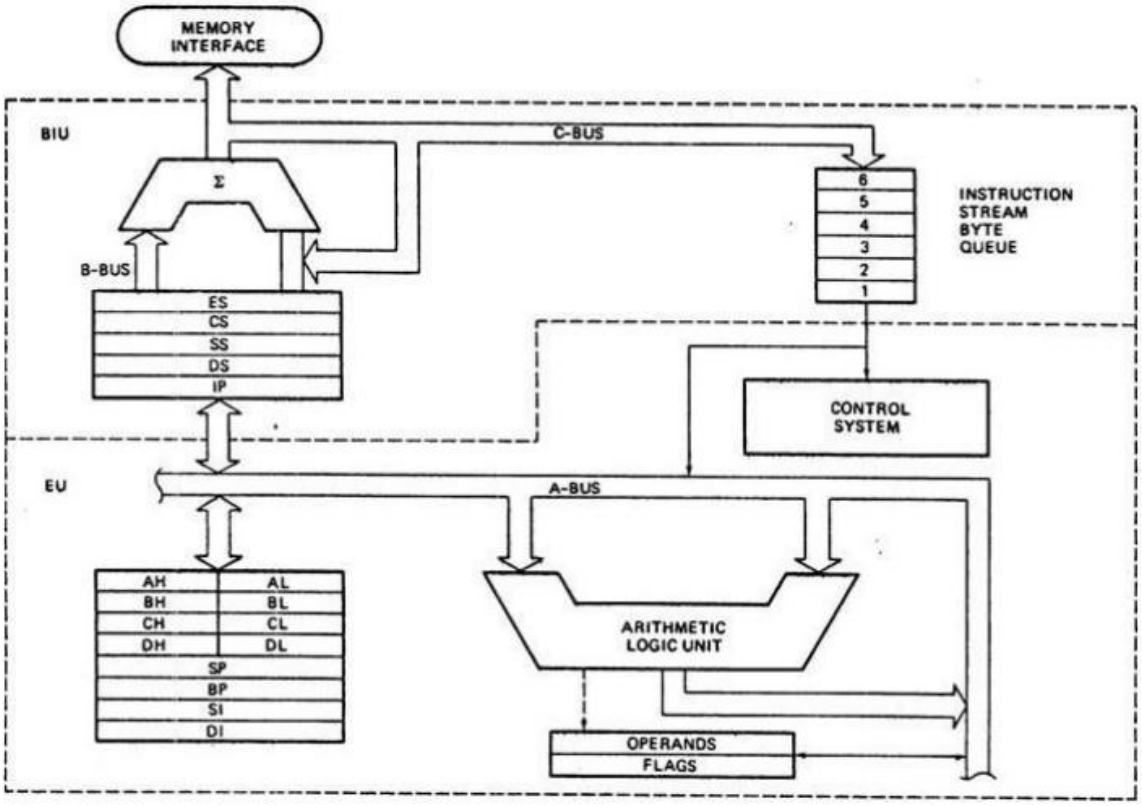
✓ EU receives program instruction codes and data from the BIU, executes these instructions and stores the results either in general registers or output them puts all its data through the BIU.

The BIU contains

1. Segment Registers, 2. Instruction Pointer (IP), 3.Instruction Queue

The EU contains

1. ALU
2. General purpose registers
3. Index registers
4. Pointers
5. Flag register



Architecture of 8086

BUS INTERFACE UNIT (BIU) :

The BIU provides the interface to external world.

- It generates the 20-bit physical address & handles all the data transfers.
- It fetches instruction from memory.
- It reads / writes data from memory / ports.
- It sends the addresses to memory / ports.
- It supports an Instruction QUEUE.

EXECUTION UNIT(EU):

The EU is responsible for the instruction execution & has a control circuit to direct its internal operations.

- It picks up the instructions from the Queue of BIU.
- It decodes the instructions & decides upon the various operations it has to carry out. It then executes these operations.
- It uses ALU to perform 16-bit Arithmetic & logical operations like ADD, SUBTRACT, AND, OR, XOR, INCREMENT, DECREMENT, COMPLEMENT & SHIFT.
- It updates the status of FLAG register (PSW Register).
- It performs BIU from where the next instruction or data has to be read.

DECODE UNIT

- The Decode Unit in the 8086 microprocessor is a component that decodes the instructions that have been fetched from memory.
- The decode unit takes the machine code instructions and translates them into micro-operations that can be executed by the microprocessor's execution unit.
- The Decode Unit works in parallel with the Prefetch Unit, which fetches instructions from memory and stores them in a queue.

- The Decode Unit reads the instructions from the queue and translates them into micro-operations that can be executed by the microprocessor.

CONTROL UNIT

- The Control Unit in the 8086 microprocessor is a component that manages the overall operation of the microprocessor.
- The control unit is responsible for controlling the flow of instructions through the microprocessor and coordinating the activities of the other components, including the Decode Unit, Execution Unit, and Prefetch Unit.
- The Control Unit acts as the central coordinator for the microprocessor, directing the flow of data and instructions and ensuring that the microprocessor operates correctly.
- It also monitors the state of the microprocessor, ensuring that the correct sequence of operations is followed.

ALU: Arithmetic and Logic Unit ALU

It is a 16 bit register. It can add, subtract, increment, decrement, complement, shift numbers and performs AND, OR, XOR operations.

FLAG Register:

The flag register is a 16-bit register in the Intel 8086 microprocessor that contains information about the state of the processor after executing an instruction. It is sometimes referred to as the status register because it contains various status flags that reflect the outcome of the last operation executed by the processor.

General Purpose Registers(AX,BX,CX,DX)

All general registers of the 8086 microprocessor can be used for arithmetic and logic operations. The 16 bit general purpose registers are AX,BX,CX,DX.

Accumulator Register (AX)

- Consists of two 8-bit registers AL and AH, which can be combined together and used as a 16-bit register AX.
- AL in this case contains the low order byte of the word, and AH contains the high-order byte.
- The I/O instructions use the AX or AL for inputting / outputting 16 or 8 bit data to or from an I/O port.
- Multiplication and Division instructions also use the AX or AL.

Base Register (BX)

- Consists of two 8-bit registers BL and BH, which can be combined together and used as a 16-bit register BX.
- BL in this case contains the low-order byte of the word, and BH contains the high-order byte.
- This is the only general purpose register whose contents can be used for addressing the 8086 memory.
- All memory references utilizing this register content for addressing use DS as the default segment register.

Counter Register (CX)

- Consists of two 8-bit registers CL and CH, which can be combined together and used as a 16-bit register CX.
- When combined, CL register contains the low order byte of the word, and CH contains the high-order byte.
- Instructions such as SHIFT, ROTATE and LOOP use the contents of CX as a counter

Data Register (DX)

- Consists of two 8-bit registers DL and DH, which can be combined together and used as a 16-bit register DX.
- When combined, DL register contains the low order byte of the word, and DH contains the high-order byte.
- Used to hold the high 16-bir result in 16×16 multiplication or high 16-bit dividend before a 32÷16 division and 16-bit reminder after division

Stack Pointer (SP):

- Contains an assumed offset value of the top of the stack
- Combined with the SS register form the complete logical address of the top of the stack
- The stack is maintained as a LIFO with its bottom at the start of the stack segment (specified by the SS segment register)
- The top of the stack (the location of the last data in the stack) is specified by the offset stored in the SP register

Base Pointer (BP):

- As it is with the stack pointer, this register can hold an offset from the SS register
- It is usually used by subroutines to locate variables that were passed on the stack by a calling program
- Unlike the SP register, the BP can be used to specify the offset of other program segments

Instruction Pointer (IP or Program counter):

- This is the register used for accessing instructions
- While the CS register contains the segment number of the next instruction, the IP contains the offset of that instruction in the code segment
- The IP register gets updated by the control unit every time an instruction is executed such that it will always point to the next instruction
- Unlike other registers, the IP can not be manipulated by instructions (i.e. it cannot appear as an operand in any instruction)

Source Index (SI):

- Used in conjunction with the DS register to point to data locations in the data segment
- Used with string movement instructions. It would point to the source string

Destination Index (SI):

- Similar function to the SI
- Used in conjunction with the ES register in string operations. In string movement instructions it points to the destination string.

Code Segment (CS) Register

- It is a 16-bit register
- CS contains the base or start of the current code segment; IP contains the distance or offset from this address to the next instruction byte to be fetched.
- BIU computes the 20-bit physical address by logically shifting the contents of CS 4-bits to the left and then adding the 16-bit contents of IP.
- That is, all instructions of a program are relative to the contents of the CS register multiplied by 16 and then offset is added provided by the IP.

Data Segment (DS) Register

- It is a 16-bit register
- Points to the current data segment; operands for most instructions are fetched from this segment.
- The 16-bit contents of the Source Index (SI) or Destination Index (DI) or a 16-bit displacement are used as offset for computing the 20-bit physical address.

Stack Segment (SS) Register

- It is a 16-bit register
- Points to the current stack.
- The 20-bit physical stack address is calculated from the Stack Segment (SS) and the Stack Pointer (SP) for stack instructions such as PUSH and POP.
- In based addressing mode, the 20-bit physical stack address is calculated from the Stack segment (SS) and the Base Pointer (BP).

Extra Segment (ES) Register

- It is a 16-bit register
- Points to the extra segment in which data (in excess of 64K pointed to by the DS) is stored.
- String instructions use the ES and DI to determine the 20-bit physical address for the destination.

Instruction Queue

- The prefetched instruction code (/opcode/machine code) from the memory is stored in the instruction Queue for use by the EU.
- It is 6 byte long
- The use instruction Queue introduce the concept of piplining in Intel microprocessors.

Q3. What are the addressing modes in 8086? Explain with example.

ANSWER:

- Addressing mode indicates a way of locating data or operands.
- The addressing modes describe the types of operands and the way they are accessed for executing an instruction.

According to the flow of instruction execution, the instructions may be categorized as

i) **Sequential control flow instructions** (**Immediate, Direct, Register, Register Indirect, Indexed, Base-Indexed, Register relative, Relative Based Indexed addressing modes**

ii) **Control transfer instructions**(**intersegment and intra-segment addressing modes**

Sequential control flow instructions are the instructions, which after execution, transfer control to the next instruction appearing immediately after it (in the sequence) in the program. For example, the arithmetic, logic, data transfer and processor control instructions are sequential control flow instructions.

The control transfer instructions, on the other hand, transfer control to some predefined address .

Address somehow specified in the instruction, after their execution. For example, INT, CALL, RET and JUMP instructions fall under this category.

The addressing modes for **sequential control transfer instructions** are:

1.Immediate: In this type of addressing, immediate data is a part of instruction and appears in the form of successive byte or bytes.

Ex: MOV AX, 0005H

In the above example, 0005H is the immediate data. The immediate data may be 8-bit or 16-bit in size.

2. Direct: In the direct addressing mode a 16-bit memory address (offset) is directly specified in the instruction as a part of it.

Ex: MOV AX, [5000H]

Here, data resides in a memory location in the data segment, whose effective address may be completed using 5000H

As the offset address and content of DS as segment address. The effective address here, is 10H * DS + 5000H.

3. Register: In register addressing mode, the data is stored in a register and is referred using the particular register. All the registers, except IP, may be used in this mode.
Ex: MOV BX, AX
4. Register Indirect: Sometimes, the address of the memory location, which contains data or operand, is determined in an indirect way, using the offset register. This mode of addressing is known as register indirect mode.
In this addressing mode, the offset address of data is in either BX or SI or DI register. The default segment is either DS or ES.
The data is supposed to be available at the address pointed to by the content of any of the above registers in the default data segment.
Ex: MOV AX, [BX]
Here, data is present in a memory location in DS whose offset address is in BX. The effective address of the data is given as 10H * DS+[BX].
5. Indexed: In this addressing mode, offset of the operand is stored in one of the index registers. DS and ES are the default segments for index registers, SI and DI respectively. This is a special case of register indirect addressing mode.

- Ex: MOV AX, [SI]
- Here, data is available at an offset address stored in SI in DS. The effective address, in this case, is computed as
- 10*DS+[SI].

6. Register Relative: In this addressing mode, the data is available at an effective address formed by adding an 8-bit or 16-bit displacement with the content of any one of the registers BX, BP, SI and DI in the default (either DS or ES) segment.
Ex: MOV AX, 50H[BX]
Here, the effective address is given as 10H *DS+50H+[BX
7. Based Indexed: The effective address of data is formed, in this addressing mode, by adding content of a base register (any one of BX or BP) to the content of an index register (any one of SI or DI). The default segment register

- may be ES or DS.
- Ex: MOV AX, [BX][SI]
- Here, BX is the base register and SI is the index register the effective address is computed as 10H * DS + [BX] + [SI].

8. Relative Based Indexed: The effective address is formed by adding an 8 or 16-bit displacement with the sum of the contents of any one of the base register (BX or BP) and any one of the index register, in a default segment.
Ex: MOV AX, 50H [BX] [SI]

Here, 50H is an immediate displacement, BX is base register and SI is an index register the effective address of data is computed as
10H * DS + [BX] + [SI] + 50H

Basically, there are two addressing modes for the control transfer instructions,
(i)intersegment addressing and
(ii)intrasegment addressing modes.

- If the location to which the control is to be transferred lies in a different segment other than the current one, the mode is called intersegment mode.
- If the destination location lies in the same segment, the mode is called intrasegment mode.

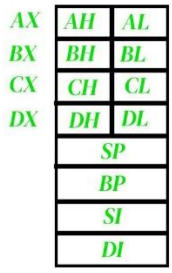
1. Intrasegment Direct Mode: In this mode, the address to which the control is to be transferred lies in the same segment in which the control transfer instruction lies and appears directly in the instruction as an immediate displacement value.
In this addressing mode, the displacement is computed relative to the content of the instruction pointer IP.
2. Intrasegment Indirect Mode: In this mode, the displacement to which the control is to be transferred, is in the same segment in which the control transfer instruction lies, but it is passed to the instruction indirectly.
Here, the branch address is found as the content of a register or a memory location. This addressing mode may be used in unconditional branch instructions
3. Intersegment Direct: In this mode, the address to which the control is to be transferred is in a different segment.
This addressing mode provides a means of branching from one code segment to another code segment. Here, the CS and IP of the destination address are specified directly in the instruction.
4. Intersegment Indirect: In this mode, the address to which the control is to be transferred lies in a different segment and it is passed to the instruction indirectly, i.e contents of a memory block containing four bytes, i.e IP (LSB),
IP(MSB), CS(LSB) and CS (MSB) sequentially. The starting address of the memory block may be referred using any of the addressing modes, except immediate mode.

Q4. Discuss register set of 8086.

Answer:

(i) **General-purpose registers(AX,BX,CX,DX,SI,DI,SP,BP)** **(ii) Special purpose register (CS,SS,DS,ES,IP,FLAG register)**

General-purpose registers are used to store temporary data within the microprocessor. There are 8 general-purpose registers in the 8086 microprocessor.



1. AX: This is the accumulator. It is of 16 bits and is divided into two 8-bit registers AH and AL to also perform 8-bit instructions. It is generally used for arithmetical and logical instructions but in 8086 microprocessor it is not mandatory to have an accumulator as the destination operand. Example:
ADD AX, AX (AX = AX + AX)

2. BX: This is the base register. It is of 16 bits and is divided into two 8-bit registers BH and BL to also perform 8-bit instructions. It is used to store the value of the offset. Example:
MOV BL, [500] (BL = 500H)

3. CX: This is the counter register. It is of 16 bits and is divided into two 8-bit registers CH and CL to also perform 8-bit instructions. It is used in looping and rotation. Example:
MOV CX, 0005

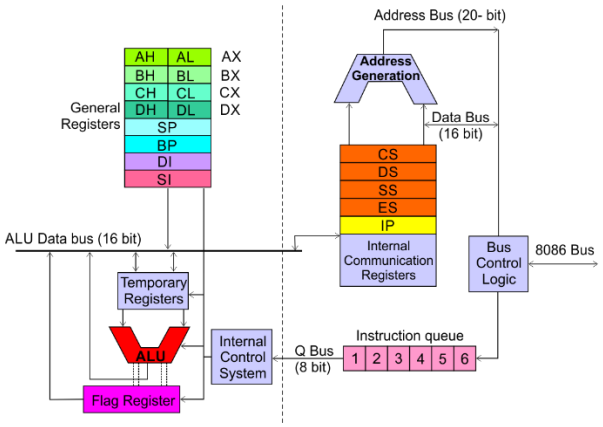
LOOP

4. DX: This is the data register. It is of 16 bits and is divided into two 8-bit registers DH and DL to also perform 8-bit instructions. It is used in the multiplication and input/output port addressing. Example:
MUL BX (DX, AX = AX * BX)

5. SP: This is the stack pointer. It is of 16 bits. It points to the topmost item of the stack. If the stack is empty the stack pointer will be (FFFE)H. Its offset address is relative to the stack segment.
6. BP – This is the base pointer. It is of 16 bits. It is primarily used in accessing parameters passed by the stack. Its offset address is relative to the stack segment.
7. SI – This is the source index register. It is of 16 bits. It is used in the pointer addressing of data and as a source in some string-related operations. Its offset is relative to the data segment.
8. DI – This is the destination index register. It is of 16 bits. It is used in the pointer addressing of data and as a destination in some string-related operations. Its offset is relative to the extra segment.

The 8086 registers are classified into the following types:

- General Data Registers(AX,BX,CX,DX)
- Segment Registers(CS,DS,SS,ES)
- Pointers and Index Registers(SP,BP,DI,SI,IP)
- Flag Register



1. General Data Registers(AX,BX,CX,DX)

- The registers **AX, BX, CX and DX** are the general purpose 16-bit registers.
- AX is used as 16-bit accumulator. The lower 8-bit is designated as AL and higher 8-bit is designated as AH. AL can be used as an 8-bit accumulator for 8-bit operation.
- All data register can be used as either 16 bit or 8 bit.
- BX is a 16 bit register, but BL indicates the lower 8-bit of BX and BH indicates the higher 8-bit of BX. The register BX is used as offset storage for forming physical address in case of certain addressing modes.
- **The register CX** is used default counter in case of string and loop instructions. **DX register is a general purpose** register which may be used as an implicit operand or destination in case of a few instructions

2. Segment Registers

- There are 4 segment registers. They are:
  - **Code Segment Register(CS)**
  - **Data Segment Register(DS)**
  - **Extra Segment Register(ES)**
  - **Stack Segment Register(SS)**

- The 8086 architecture uses the concept of segmented memory. 8086 able to address a memory capacity of 1 megabyte and it is byte organized. This 1 megabyte memory is divided into 16 logical segments. Each segment contains 64 kbytes of memory.
- **Code segment register (CS):** is used for addressing memory location in the code segment of the memory, where the executable program is stored.
- **Data segment register (DS):** points to the data segment of the memory where the data is stored.
- **Extra Segment Register** (ES) : also refers to a segment in the memory which is another data segment in the memory.
- **Stack Segment Register** (SS): is used for addressing stack segment of the memory. The stack segment is that segment of memory which is used to store stack data.

3. Pointers and Index Registers:

- The index and pointer registers are given below:
- **IP—Instruction pointer**-store memory location of next instruction to be executed

- BP—Base pointer
- SP—Stack pointer
- SI—Source index
- DI—Destination index

Pointer Registers

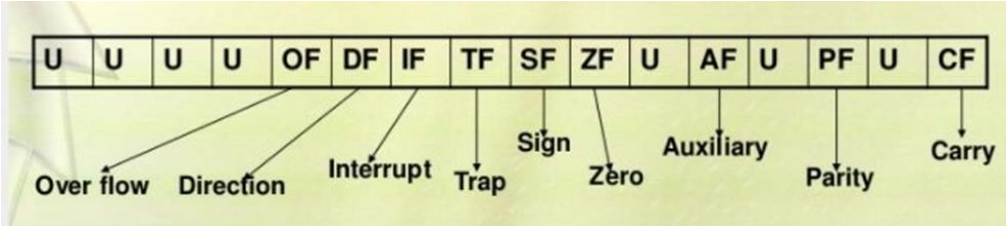
- The pointers registers contain offset within the particular segments.
- The pointer register IP contains offset within the code segment.
- The pointer register BP contains offset within the data segment.
- **Thee pointer register SP contains offset within the stack segment.**

Index registers

- The index registers are used as general purpose registers as well as for offset storage in case of indexed, base indexed and relative base indexed addressing modes.
- The register SI is used to store the offset of source data in data segment.
- The register DI is used to store the offset of destination in data or extra segment.
- The index registers are particularly useful for string manipulation.

FLAG REGISTER

- **The 8086 flag register contents indicate the results of computation in the ALU. It also contains some flag bits to control the CPU operations.**



Flags

- **CF** is set if there is a carry out of the MSB position resulting from an addition operation or if borrow is needed out of the MSB position during subtraction.
- **PF** is set if the lower 8-bits of the result of an operation contains an even number of 1's.
- **AF** is set if there is a carry out of bit 3 resulting from an addition operation or borrow required from bit 4 into bit 3 during subtraction operation.
- **ZF** is set if the result of an arithmetic or logical operation is zero.

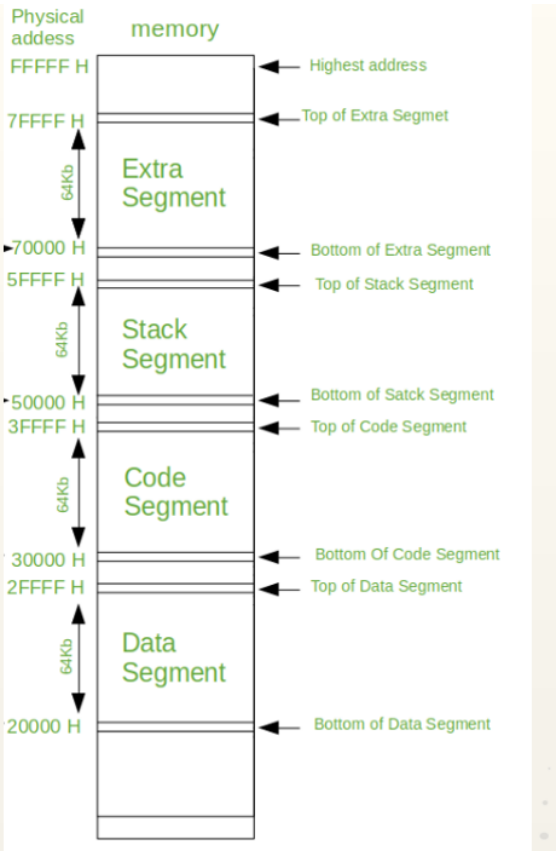
- **SF** is set if the MSB of the result of an operation is 1. SF is used with signed/unsigned numbers.
- **OF** is used only for signed arithmetic operation and is set if the result is too large to be fitted in the number of bits available to accommodate it.
- **TF** is control flag, when it is set (=1), the processor operates in single stepping mode-i.e., pausing after each instruction is executed
- **IF** is control flag, when it is set (=1) the INTR is enabled otherwise disabled
- **DF** is the control flag used in string operations, when it is set (=1) and MOVS instruction is executed, the contents of the index registers SI and DI are automatically decremented. When it is reset (=0) and MOVS instruction is executed, the contents of the index registers SI and DI are automatically incremented.

Q5. Explain memory organization and segmentation in 8086.

Answer:

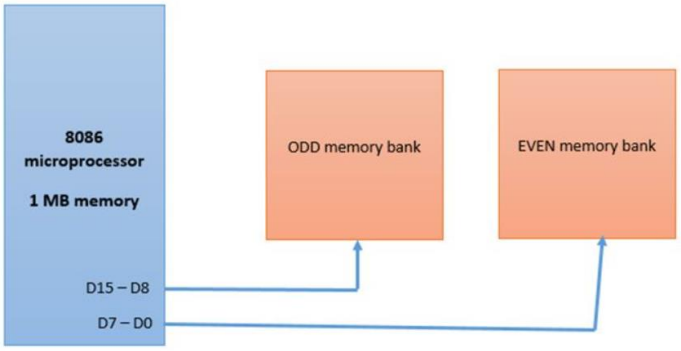
- **Memory Segmentation:**
- The memory in an 8086 based system is organized as segmented memory.
- The CPU 8086 is able to access 1MB of physical memory. The complete 1MB of memory can be divided into 16 segments, each of 64KB size and is addressed by one of the segment register.
- The 16-bit contents of the segment register actually point to the starting location of a particular segment.
- The address of the segments may be assigned as 0000H to F000h respectively.
- To address a specific memory location within a segment, we need an offset address.
- The offset address values are from 0000H to FFFFH so that the physical addresses range from 00000H to FFFFFH

- There are 20 address lines in the 8086 microprocessor. That gives (2^20) 1MB memory units.
- 1MB memory of 8086 is partitioned into 16 segments-each segments is 64kB in length. Out of these segments, only 4 segments can be active at any given instant of time. These are the
- Code Segment Register(CS)
- Data Segment Register(DS)
- Extra Segment Register(ES)
- Stack Segment Register(SS)



Memory Organization and accessing

- There are 20 address lines in the 8086 microprocessor. That gives (2^20) 1MB memory size.
- Even the memory is byte-addressable, yet the 8086 microprocessor an easily handle up to 16 bits of data at a time through its 16 data lines. So, to organize the memory efficiently, the entire memory in 8086 is divided into two memory banks: odd bank and the even bank.



- The way in which data is read or written is decided by the value of BHE, and the last address bit, that is the A0 line. It is done in the following way:
-

BHE'	A0	Operation performed on memory
0	0	16 bits of data will be read or written into the memory
0	1	8 bits of data will be read/written into the odd memory bank
1	0	8 bits of data will be read/written into the even memory bank
1	1	No operation is performed

- To read or write 8 bits of data, it would require only 1 CPU cycle, no matter the data is stored in any of the memory banks, but to read or write 16 bits of data, the BIU of the 8086 may require either 1 or 2 memory cycles depending upon whether the lower byte of word is located at even or odd memory address.
- If the lower byte of the word is stored at even memory bank and the upper byte is stored at odd memory bank then the CPU will require only 1 memory cycle. So, it is better to store data in this way.
- If the lower byte of the word is located at an odd memory address, then the CPU will require 2 memory cycles. The first memory cycle is required for accessing the lower byte of the word through the higher data bus, i.e. D15 to D8, and the second memory cycle is required for accessing the upper byte of the word through the lower data bus, i.e. D7 to D0.

Q5.b Calculation of effective or physical address

Example`The value of Code Segment (CS) Register is 4042H and the value of different offsets is as follows:
BX: 2025H , IP: 0580H , DI: 4247H
Calculate the effective address of the memory location pointed by the CS register.

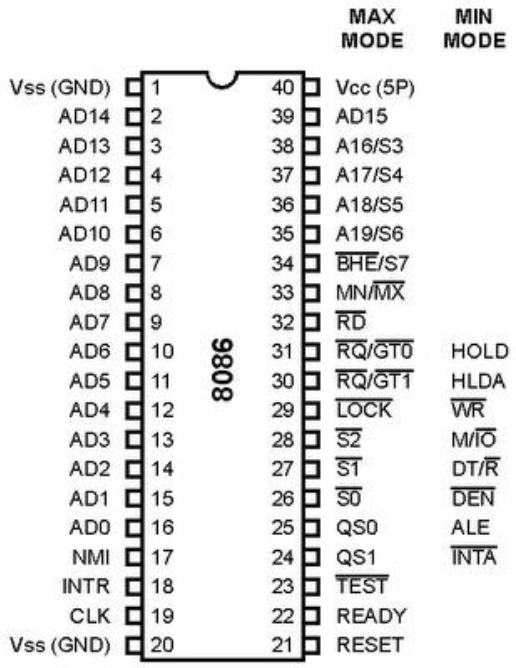
Solution:

- The offset of the CS Register is the IP register.
- Therefore, the effective address of the memory location pointed by the CS register is calculated as follows:
- Effective address= Base address of CS register X 10H + Address of IP
- = 4042H X 10H + 0580H
- = (40420 + 0580)H
- = 41000H

Q6. Explain the PIN DIAGRAM OF 8086 processor

ANSWER:

PIN DIAGRAM



Let us now discuss the signals in detail −

Power supply and frequency signals

It uses 5V DC supply at V_{CC} pin 40, and uses ground at V_{SS} pin 1 and 20 for its operation.

Clock signal

Clock signal is provided through Pin-19. It provides timing to the processor for operations. Its frequency is different for different versions, i.e. 5MHz, 8MHz and 10MHz.

Address/data bus

AD0-AD15. These are 16 address/data bus. AD0-AD7 carries low order byte data and AD8AD15 carries higher order byte data. During the first clock cycle, it carries 16-bit address and after that it carries 16-bit data.

Address/status bus

A16-A19/S3-S6. These are the 4 address/status buses. During the first clock cycle, it carries 4-bit address and later it carries status signals.

S7/BHE

BHE stands for Bus High Enable. It is available at pin 34 and used to indicate the transfer of data using data bus D8-D15. This signal is low during the first clock cycle, thereafter it is active.

Read(*$\overline{RD}$*)

It is available at pin 32 and is used to read signal for Read operation.

Ready

It is available at pin 22. It is an acknowledgement signal from I/O devices that data is transferred. It is an active high signal. When it is high, it indicates that the device is ready to transfer data. When it is low, it indicates wait state.

RESET

It is available at pin 21 and is used to restart the execution. It causes the processor to immediately terminate its present activity. This signal is active high for the first 4 clock cycles to RESET the microprocessor.

INTR

It is available at pin 18. It is an interrupt request signal, which is sampled during the last clock cycle of each instruction to determine if the processor considered this as an interrupt or not.

NMI

It stands for non-maskable interrupt and is available at pin 17. It is an edge triggered input, which causes an interrupt request to the microprocessor.

$\overline{TEST}$

This signal is like wait state and is available at pin 23. When this signal is high, then the processor has to wait for IDLE state, else the execution continues.

MN/*$\overline{MX}$*

It stands for Minimum/Maximum and is available at pin 33. It indicates what mode the processor is to operate in; when it is high, it works in the minimum mode and vice-aversa.

INTA

It is an interrupt acknowledgement signal and id available at pin 24. When the microprocessor receives this signal, it acknowledges the interrupt.

ALE

It stands for address enable latch and is available at pin 25. A positive pulse is generated each time the processor begins any operation. This signal indicates the availability of a valid address on the address/data lines.

DEN

It stands for Data Enable and is available at pin 26. It is used to enable Transreceiver 8286. The transreceiver is a device used to separate data from the address/data bus.

DT/R

It stands for Data Transmit/Receive signal and is available at pin 27. It decides the direction of data flow through the transreceiver. When it is high, data is transmitted out and vice-a-versa.

M/IO

This signal is used to distinguish between memory and I/O operations. When it is high, it indicates I/O operation and when it is low indicates the memory operation. It is available at pin 28.

WR

It stands for write signal and is available at pin 29. It is used to write the data into the memory or the output device depending on the status of M/IO signal.

HLDA

It stands for Hold Acknowledgement signal and is available at pin 30. This signal acknowledges the HOLD signal.

HOLD

This signal indicates to the processor that external devices are requesting to access the address/data buses. It is available at pin 31.

QS_1 and QS_0

These are queue status signals and are available at pin 24 and 25. These signals provide the status of instruction queue. Their conditions are shown in the following table −

QS_0	QS_1	Status

0	0	No operation
0	1	First byte of opcode from the queue
1	0	Empty the queue
1	1	Subsequent byte from the queue

S_0, S_1, S_2

These are the status signals that provide the status of operation, which is used by the Bus Controller 8288 to generate memory & I/O control signals. These are available at pin 26, 27, and 28. Following is the table showing their status –

S_2	S_1	S_0	**Status**
0	0	0	Interrupt acknowledgement
0	0	1	I/O Read
0	1	0	I/O Write
0	1	1	Halt
1	0	0	Opcode fetch
1	0	1	Memory read
1	1	0	Memory write
1	1	1	Passive

LOCK

When this signal is active, it indicates to the other processors not to ask the CPU to leave the system bus. It is activated using the LOCK prefix on any instruction and is available at pin 29.

RQ/GT_1 and RQ/GT_0

These are the Request/Grant signals used by the other processors requesting the CPU to release the system bus. When the signal is received by CPU, then it sends acknowledgment. RQ/GT_0 has a higher priority than RQ/GT_1.

$AD_{15} - AD_0$

- These are time multiplexed address and data lines. They act as address lines during first part of machine cycle and data lines in later part.

$A_{19}/S_6 - A_{16}/S_3$

- These are time multiplexed address and status lines. They act as address lines during first part of machine cycle and status lines in later part.
- These are most significant address lines for memory operations. During I/O operations these lines are low.
- The status signals S_4 and S_3 indicate which segment registers is being used for memory access.
- The status of interrupt enable flag bit will be displayed on S_5.
- The status line S_6 is always low.

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

$\overline{BHE}/S_7$- Bus high enable/ status

- Low signal on $\overline{BHE}$ indicates access to higher order memory banks, otherwise access is to only lower order memory banks.
- $\overline{BHE}$ and A_0 decide the memory bank and type of access.
- S_7 has no function.

$\overline{BHE}$	A_0	Indications
0	0	Both higher and lower order banks for word read/ write
0	1	Higher order bank for byte read/ write
1	0	Lower order bank for byte read/ write
1	1	None

$\overline{RD}$

- Read control signal
- $\overline{RD}$ is low when 8086 is receiving the data from memory or I/O.

READY

- Wait state request signal.
- A HIGH on READY input causes the 8086 to extend the machine cycle by inserting wait states.

$\overline{\text{TEST}}$

- This input is examined by WAIT instruction.
- If the $\overline{\text{TEST}}$ input goes low, execution will continue, else, the processor remains in idle state.

INTR

- This is level triggered input.
- INTR is sampled during the last clock cycle of each instruction to determine the availability of request.
- These interrupts can be masked internally by resetting the interrupt enable flag.

NMI

- NMI (Non-Maskable interrupt) is positive edge triggered non-maskable interrupt request.

CLK

- CLK is clock signal from external crystal oscillator.
- 8086 requires clock signal with 33% duty cycle.

RESET

- System reset signal must be high for atleast 4 clock periods to cause reset.
- Reset operation takes about 10 clock periods.

V_{CC}

- +5V supply with ±5% tolerance.

GND

- Ground for internal circuits.

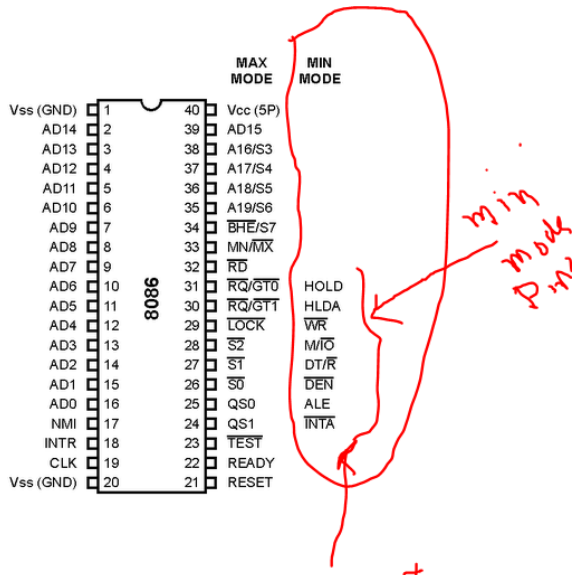
MN/$\overline{MX}$

- High on this pin selects minimum mode and low signal selects maximum mode.

Q7. EXPLAIN 8086 PINS IN MINIMUM MODE.

ANSWER:

PIN DIAGRAM



ALE

- Address latch enable.
- High on this pin indicates valid address on address/data bus.

$\overline{WR}$

- Write control signal.
- $\overline{WR}$ is low when 8086 sends the data to memory or I/O.

$M/\overline{IO}$

- $M/\overline{IO}$ = High, indicates memory access.

- $M/\overline{IO}$ = low, indicates I/O access.

$\overline{INTA}$

- $\overline{INTA}$ is the acknowledgement for the interrupt request on INTR pin.
- It is pulsed low in two consecutive bus cycles.
- First pulse indicates interrupt acknowledgement.
- During second pulse, external logic puts the interrupt type on data bus.

$DT/\overline{R}$

- Data transmit/receive.
- This signal, when high indicates data is being transmitted by 8086.
- The low signal indicates that 8086 is receiving the data.

$\overline{DEN}$

- Data bus enable.
- This signal, when low indicates that the 8086 processors address/data bus is used as data bus.
- It is used to enable data buffers.

HOLD

- HOLD signal when high indicates another master has requested for direct memory access.
- When HOLD becomes low, it indicates that direct memory access is no more required.

HLDA

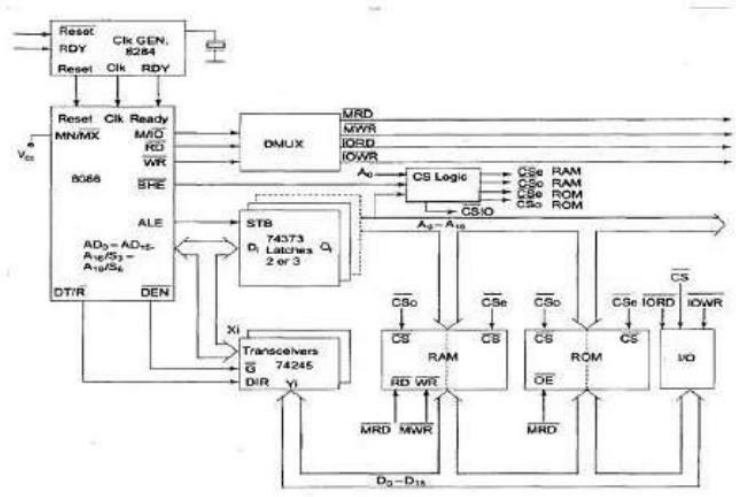
- The microprocessor sends high signal on HLDA to indicate acknowledgement of DMA request. It then tristates the buses and control.
- When HOLD becomes low, the microprocessor makes HLDA low and regains the control of buses.

8.Give the difference between maximum and minimum mode

Minimum mode	Maximum mode
In minimum mode there can be only one processor i.e. 8086.	In maximum mode there can be multiple processors with 8086, like 8087 and 8089.
$MN/\overline{MX}$ is 1 to indicate minimum mode.	$MN/\overline{MX}$ is 0 to indicate maximum mode.
ALE for the latch is given by 8086 as it is the only processor in the circuit.	ALE for the latch is given by 8288 bus controller as there can be multiple processors in the circuit.
$\overline{DEN}$ and $DT/\overline{R}$ for the trans-receivers are given by 8086 itself.	and$DT/\overline{R}$ for the trans-receivers are given by 8288 bus controller.
Direct control signals $M/\overline{IO}$, $\overline{RD}$ and $\overline{WR}$ are given by 8086.	Instead of control signals, each processor generates status signals called $\overline{S_2}$, $\overline{S_1}$ and $\overline{S_0}$.
Control signals $M/\overline{IO}$, $\overline{RD}$ and $\overline{WR}$ are decoded by a 3:8 decoder like 74138.	Status signals $\overline{S_2}$, $\overline{S_1}$ and $\overline{S_0}$ are decoded by a bus controller like 8288 to produce control signals.
$\overline{INTA}$ is given by 8086 in response to an interrupt on INTR line.	$\overline{INTA}$ is given by 8288 bus controller in response to an interrupt on INTR line.
HOLD and HLDA signals are used for bus request with a DMA controller like 8237.	$\overline{RQ}/\overline{GT}$,lines are used for bus requests by other processors like 8087 or 8089.
The circuit is simpler.	The circuit is more complex.
Multiprocessing cannot be performed hence performance is lower.	As multiprocessing can be performed, it can give very high performance.

9. Explain the working of 8086 in Minimum mode with timing diagram

Minimum Mode 8086 System



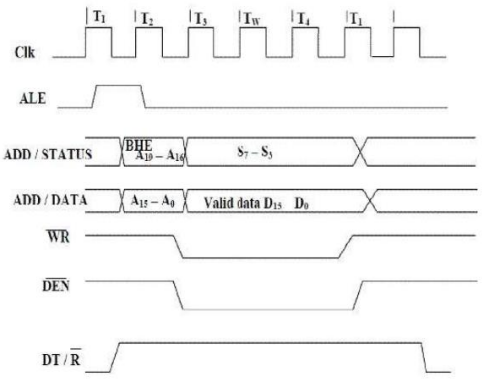
Minimum mode 8086 system

- In a minimum mode 8086 system, the microprocessor 8086 is operated in minimum mode by strapping its MN/MX pin to logic 1.
- 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, transceivers, 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.
- 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.

- Transceivers are the bidirectional buffers and some times they are called as data amplifiers. They are required to separate the valid data from the time multiplexed address/data signals.
- They are controlled by two signals namely, DEN and DT/R.
- The DEN signal indicates the direction of data, i.e. from or to the processor. The system contains memory for the monitor and users program storage.
- Usually, EPROM is used for monitor storage, while RAM for users program storage. A system may contain I/O devices.

Write Cycle Timing Diagram for Minimum Mode

- The working of the minimum mode configuration system can be better described in terms of the timing diagrams rather than qualitatively describing the operations.
- The opcode fetch and read cycles are similar. Hence the timing diagram can be categorized in two parts, the first is the timing diagram for read cycle and the second is the timing diagram for write cycle.



- The read cycle begins in T1 with the assertion of address latch enable (ALE) signal and also M / IO signal. During the negative going edge of this signal, the valid address is latched on the local bus.
- The BHE and A0 signals address low, high or both bytes. From T1 to T4 , the M/IO signal indicates a memory or I/O operation.
- At T2, the address is removed from the local bus and is sent to the output. The bus is then tristated. The read (RD) control signal is also activated in T2.
- The read (RD) signal causes the address device to enable its data bus drivers. After RD goes low, the valid data is available on the data bus.
- The addressed device will drive the READY line high. When the processor returns the read signal to high level, the addressed device will again tristate its bus drivers.
- A write cycle also begins with the assertion of ALE and the emission of the address. The M/IO signal is again asserted to indicate a memory or I/O operation. In T2, after sending the address in T1, the processor sends the data to be written to the addressed location.

processor sends the data to be written to the addressed location.

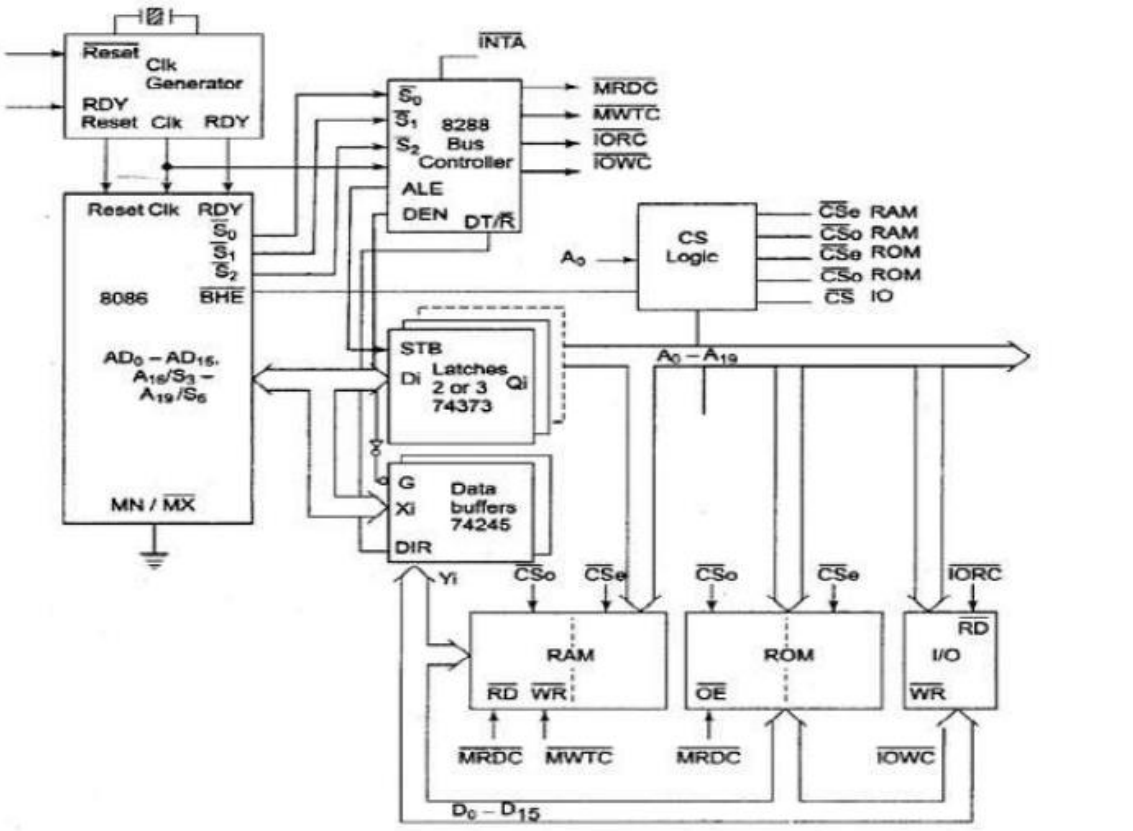
- The data remains on the bus until middle of T4 state. The WR becomes active at the beginning of T2 (unlike RD is somewhat delayed in T2 to provide time for floating).
- The BHE and A0 signals are used to select the proper byte or bytes of memory or I/O word to be read or write.
- The M/IO, RD and WR signals indicate the type of data transfer as specified in table below.

Q10. Explain the working of 8086 in Maximum mode with timing diagram

Answer

Maximum Mode 8086 System

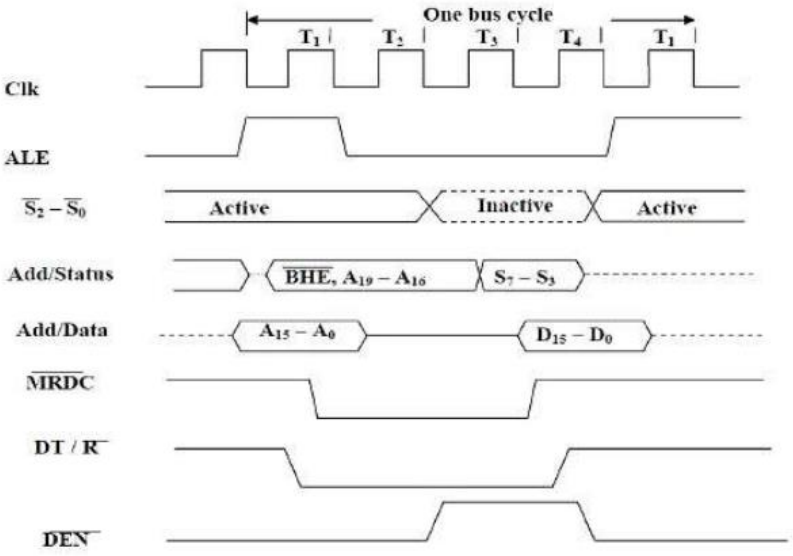
- In the maximum mode, the 8086 is operated by strapping the MN/MX pin to ground.
- In this mode, the processor derives the status signal S2, S1, S0. Another chip called bus controller derives the control signal using this status information.
- In the maximum mode, there may be more than one microprocessor in the system configuration.
- The components in the system are same as in the minimum mode system.
- The basic function 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 by the processor on the status lines.
- The bus controller chip has input lines S2, S1, S0 and CLK. These inputs to 8288 are driven by CPU.



- It derives the outputs ALE, DEN, DT/R, MRDC, MWTC, AMWC, IORC, IOWC and AIOWC. The AEN, IOB and CEN pins are especially useful for multiprocessor systems.
- AEN and IOB are generally grounded. CEN pin is usually tied to +5V. The significance of the MCE/PDEN output depends upon the status of the IOB pin.
- If IOB is grounded, it acts as master cascade enable to control cascade 8259A, else it acts as peripheral data enable used in the multiple bus configurations.
- INTA pin used to issue two interrupt acknowledge pulses to the interrupt controller or to an interrupting device.
- 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 address port.
- The MRDC, MWTC are memory read command and memory write command signals respectively and may be used as memory read or write signals.

- All these command signals instructs 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.
- Here the only difference between in timing diagram between minimum mode and maximum mode is the status signals used and the available control and advanced command signals.
- R0, S1, S2 are set at the beginning of bus cycle.8288 bus controller will output a pulse as on the ALE and apply a required signal to its DT / R pin during T1.
- In T2, 8288 will set DEN=1 thus enabling transceivers, and for an input it will activate MRDC or IORC. These signals are activated until T4. For an output, the AMWC or AIOWC is activated from T2 to T4 and MWTC or IOWC is activated from T3 to T4.
- The status bit S0 to S2 remains active until T3 and become passive during T3 and T4.
- If reader input is not activated before T3, wait state will be inserted between T3 and T4.

Memory Read Timing Diagram in Maximum Mode of 8086



Memory Read Timing in Maximum Mode

Q11,Explain the instruction set of 8086 microprocessor.

a. Give three examples for the following 8086 microprocessor instructions: String Instructions,
b. Process Control Instruction, Program Execution Transfer Instructions and Bit manipulation Instructions
c. Explain the data transfer, arithmetic and branch instructions with examples.

Answer:

Intel 8086 has approximately 117 instructions. These instructions are used to transfer data between registers, register to memory, memory to register or register to I/O ports and other instructions are used for data manipulation.

But in Intel 8086 operations between memory to memory is not permitted. These instructions are classified in to six-groups as follows.

1. Data Transfer Instructions
2. Arithmetic Instructions
3. Bit Manipulation Instructions
4. String Instructions
5. Program Execution Transfer Instructions
6. Processor Control Instructions

1. Data Transfer Instruction

Data Transfer Instructions

General-Purpose	
MOV	Move byte or word
PUSH	Push word onto stack
POP	Pop word off stack
PUSHA	Push registers onto stack
POPA	Pop registers off stack
XCHG	Exchange byte or word
XLAT	Translate byte

Input/Output	
IN	Input byte or word
OUT	Output byte or word
Address Object and Stack Frame	
LEA	Load effective address
LDS	Load pointer using DS
LES	Load pointer using ES
ENTER	Build stack frame
LEAVE	Tear down stack frame
Flag Transfer	
LAHF	Load AH register from flags
SAHF	Store AH register in flags
PUSHF	Push flags from stack
POPF	Pop flags off stack

1. MOV

MOV destination, source

This (Move) instruction transfers a byte or a word from the source operand to the destination operand.

(DEST) ← (SRC), DEST = Destination, SRC = Source

Example:

MOV AX, BX

MOVAX, 2150H

MOV AL, [1135]

MOV [4186], AL

MOV SS, DX

MOV [BX], DS

2. PUSH

PUSH Source

This instruction decrements SP (stack pointer) by 2 and then transfers a word from the source operand to the top of the stack now pointed to by stack pointer.

(SP) ← (SP)-2

((SP) + 1: (SP)) ← (SRC)

Example:

PUSH SI

PUSH BX

3. POP

POP destination

This instruction transfers the word at the current top of stack (pointed to by SP) **to** the destination operand and then increments SP by 2, pointing to the new top of the stack.

(DEST) ← ((SP) + 1:(SP))

(SP) ← (SP) + 2

Example:

POP DX

POP DS

2. Arithmetic And Logical Instructions

A. Arithmetic instructions

1. ADD

ADD destination, source

This **(Add)** instruction adds the two operands (byte or word) and stores the result in destination operand.

(DEST) ← (DEST) + (SRC)

Example:

ADD CX, DX

ADD AX, 1257 H

ADDBX, [CX]

3. SUB

SUB destination, source

This (Subtract) instruction subtracts the source operand from the destination operand and the result is stored in destination operand.

(DEST) ← (DEST) - (SRC)

Example:

SUB AX, 6541 H

SUB BX, AX

SUB SI, 5780 H

13.MUL

MUL source

- This **(Multiply)** instruction multiply AL or AX register by register or memory location contents.
- Both operands are unsigned numbers.
- If the source is a byte (8 bit), then it is multiplied by register AL and the result is stored in AH and AL.
- If the source operand is a word (16 bit), then it is multiplied by register AX and the result is stored in AX and DX registers.

If 8 bit data, (AX) ← (AL) × (SRC)

If 16 bit data, (AX), (DX) ← (AX) × (SRC)

Example:

MUL25

MUL CX .

MULBL

16. DIV

DIV Source

- This (Division) instruction performs an unsigned division of the accumulator by the source operand.
- It allows a 16 bit unsigned number to be divided by an 8 bit unsigned number, or a 32 bit unsigned number to be divided by a 16 bit unsigned number.
- If byte (8-bit) operation is performed, the 8 bit quotient is stored to AL and 8 bit remainder is stored to AH register.

5. CMP

CMP destination, source

This (Compare) instruction subtracts the source from the destination, but does not store the result.

(DEST) - (SRC)

Example:

CMP AX, 18

CMP BX, CX

6. INC

INC destination

This **(Increment)** instruction adds 1 to the destination operand (byte or word).

(DEST) ← (DEST) + 1

Example:

INC BL

INC CX

B. Logical instructions

1. AND

AND destination, source

This **(AND)** instruction performs the logical "AND" of the source operand with the destination operand and the result is stored in destination.

(DEST) ← (DEST) "AND" (SRC)

Example:

AND BL, CL

AND AL, 0011 1100 B

2. OR

OR destination, source

This **(OR)** instruction performs the logical "OR" of the source operand with the destination operand and the result is stored in destination.

(DEST) ← (DEST) "OR" (SRC)

Example:

OR AX, BX

OR AL, 0FH

3. **XOR**

XOR destination, source

This **(Exclusive OR)** instruction performs the logical "XOR" of the two operands and the result is stored in destination operand.

(DEST) ← (DEST) "XOR"(SRC)

4. NOT

NOT destination

This (NOT) instruction inverts the bits (forms the l's complement) of the byte or word.

(DEST) ← 1 's complement of (DEST)

Example:

NOT AX

C. Bit Manipulation Instructions

6. SHL

SHL destination, count

This (Shift Logical Left) instruction performs the shift operation. The number of bits to be shifted is represented by a variable count, either 1 or the number contained in the CL register.

Example

SHL AL, 1

Before execution:

CF	AL							
0	1	1	0	0	1	1	0	0

After execution:

CF	AL							
1	1	0	0	1	1	0	0	0

7. SAL

SAL destination, count

SAL **(Shift Arithmetic Left)** and SHL **(Shift Logical Left)** instructions perform the same operation and are physically the same instruction.

Before execution:

CF	AL							
0	1	1	0	0	1	1	0	0

After execution:

CF	AL							
1	1	0	0	1	1	0	0	0

8. SHR

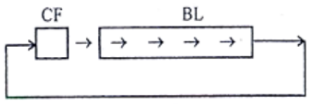
SHR destination, count

This **(Shift Logical Right)** instruction shifts the bits in the destination operand to the right by the number of bits specified by the count operand, either 1 or the number contained in the CL register.

Example

SHR BL, 1

SHR BL, CL



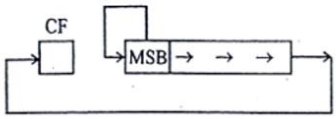
The SHR instruction may be used to divide a number by 2. For example, we can divide 32 by 2,

```
MOV BL, 32  ; 0010 0000  (32)
SHR BL, 1   ; 0001 0000  (16)
SHR BL, 1   ; 0000 1000  (8)
SHR BL, 1   ; 0000 0100  (4)
SHR BL, I   ; 0000 0010  (2)
```

9. SAR

SAR destination, count

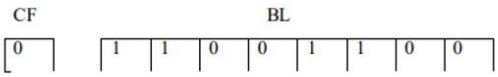
This **(Shift Arithmetic Right)** instruction shifts the bits in the destination operand to the right by the number of bits specified in the count operand. Bits equal to the original high-order (sign) bits are shifted in on the left, thereby preserving the sign of the original value.



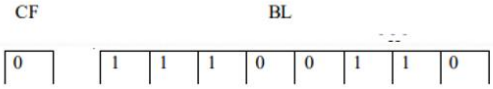
Example :

SAR BL, 1

Before execution:



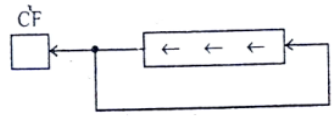
After execution:



10. ROL

ROL destination, count

This **(Rotate Left)** instruction rotates the bits in the byte/word destination operand to the left by the number of bits specified in the count operand.



Example:

ROL AL, 1

Before execution:

CF	AL							
0	1	1	0	0	1	1	0	0

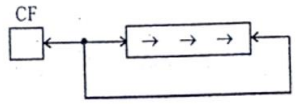
After execution:

CF	AL							
1	1	0	0	1	1	0	0	1

11. ROR

ROR destination, count

This **(Rotate Right)** instruction rotates the bits in the byte/word destination operand to the right by the number of bits specified in the count operand.



Example:

ROR AL, 1

Before execution:

CF	AL							
0	1	1	0	0	1	1	0	0

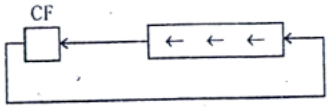
After execution:

CF	AL							
0	0	1	1	0	0	1	1	0

12. RCL

RCL destination, count

This **(Rotate through Carry Left)** instruction rotates the contents left through carry by the specified number of bits in count operand.



Example:

RCL AL, 1

Before execution:

CF	AL							
1	0	0	0	0	1	1	1	1

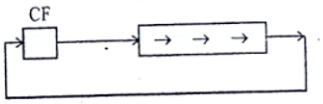
After execution:

CF	AL							
0	0	0	0	1	1	1	1	1

13.RCR

RCR destination, count

This **(Rotate through Carry Right)** instruction rotates the contents right through carry by the specified number of bits in the count operand.



Example:

RCR AL, 1

Before execution:

CF	AL							
1	1	1	0	0	0	0	1	0

After execution:

CF	AL							
0	1	1	1	0	0	0	0	1

3. Branch Instructions

- JMP Des:
  - This instruction is used for unconditional jump from one place to another.
- Jxx Des (Conditional Jump):
  - All the conditional jumps follow some conditional statements or any instruction that affects the flag.

Conditional Jump Table

Mnemonic	Meaning	Jump Condition
JA	Jump if Above	CF = 0 and ZF = 0
JAE	Jump if Above or Equal	CF = 0
JB	Jump if Below	CF = 1
JBE	Jump if Below or Equal	CF = 1 or ZF = 1
JC	Jump if Carry	CF = 1
JE	Jump if Equal	ZF = 1
JNC	Jump if Not Carry	CF = 0
JNE	Jump if Not Equal	ZF = 0
JNZ	Jump if Not Zero	ZF = 0
JPE	Jump if Parity Even	PF = 1
JPO	Jump if Parity Odd	PF = 0
JZ	Jump if Zero	ZF = 1

4. Machine Control Instructions

HLT (Halt) :- It causes the processor to enter in to the halt state. It can be stop by INTR,NMI or RESET pin

NOP (No Opration) :- It causes the processor to enter in to the wait state for 3 Clock cycles.

WAIT :- It causes the processor to enter in to the ideal state. Can be stop by TEST, INTR OR NMI pin

LOCK :- This instruction prevents other processors to take the control of shared resources. For e.g LOCK IN AL,80H

5. Flag Manipulation Instructions

- **STC:**
  - It sets the carry flag to 1.
- **CLC:**
  - It clears the carry flag to 0.
- **CMC:**
  - It complements the carry flag.

6. String Manipulation Instructions

- **String : Sequence of bytes or words**
- **8086 instruction set includes instruction for string movement, comparison, scan, load and store.**
- **REP instruction prefix : used to repeat execution of string instructions**
- **String instructions end with S or SB or SW. S represents string, SB string byte and SW string word.**
- **Offset or effective address of the source operand is stored in SI register and that of the destination operand is stored in DI register.**
- **Depending on the status of DF, SI and DI registers are automatically updated.**
- **DF = 0 Þ SI and DI are incremented by 1 for byte and 2 for word.**
- **DF = 1 Þ SI and DI are decremented by 1 for byte and 2 for word.**

REP

- REPZ/ REPE
  (Repeat CMPS or SCAS until ZF = 0)
- REPNZ/ REPNE

(Repeat CMPS or SCAS until ZF = 1) While CX ¹ 0 and ZF = 1, repeat execution of string instruction and

(CX) ¬ (CX) – 1

While CX ¹ 0 and ZF = 0, repeat execution of string instruction and (CX) ¬ (CX) – 1

Question 1

The value of Code Segment (CS) Register is 4042H and the value of different offsets is as follows:
BX: 2025H , IP: 0580H , DI: 4247H
Calculate the effective address of the memory location pointed by the CS register.

Answer

The offset of the CS Register is the IP register.

Therefore, the effective address of the memory location pointed by the CS register is calculated as follows:

Effective address= Base address of CS register X 10_H + Address of IP

$$= 4042_H \text{ X } 10_H + 0580_H$$
$$= (40420 + 0580)_H$$
$$= 41000_H$$

Question 2

Calculate the effective address for the following register:
SS: 3860H, SP: 1735H, BP: 4826H

Answer

Both SP and BP are the offsets for Stack Register (SS). The address calculated when BP is taken as the offset gives the starting address of the stack. The address when SP is taken as the offset denotes the memory location where the top of the stack lies.

Therefore, the effective address for both these cases is:

```
(SS X 10H) + SP     =   3640H X 10H  +  1735H
                    =   36400H + 1735H
                    =   38135H
(SS X 10H) + BP     =   3640H X 10H  + 4826H
                    =   36400H + 4826H
                    =   41226H
```

Question 3

The value of the DS register is 3032H. And the BX register contains a 16 bit value which is equal to 3032H. 0008H is added to BX.
ADD BX, 0008H
The register AX contains some value which needs to be stored at a location as follows:
MOV [BX], AX
Calculate the address at which the value of the AX will be stored.

Answer

After executing the first instruction, the value of BX Register is as follows:
BX = 3040H

The BX register is an offset of the Data Segment (DS) register. So, the location at which the value of the AX register will be stored is calculated as follows:

```
(DS X 10H) + BX     =   3032H X 10H  +3040H
                    =   30320H + 3040H
                    =   33360H
```

Question 4

You are provided the following values:
DS: 3056H, IP: 1023H, BP: 2322H and SP: 3029H
Can you calculate the effective address of the memory location as per the DS register?

Answer

No, the effective address of the DS register cannot be calculated from the given values because none of the given offset is an offset of the DS Register. This can be done only in the case of segment override prefix, but as it is not mentioned here, we will not follow that.

PROGRAMS

1. 16-BIT ADDITION USING 8086	comments
MOV DX,0000	Explain this line?
MOV AX,[2000H]	Store first value at 2000h? How to store it?what is in AX?
MOV BX,[2002H]	Store second value at 2002H? How to store it?what is in BX?
ADD AX,BX	Explain this line?
JNC L1	Check the flag register and explain this line?
L1:INC DX	Explain this line?

HLT	Explain this line?

2. 16-BIT SUBTRACTION USING 8086

MOV DX,0000	Explain this line?
MOV AX,[2000H]	Store first value at 2000h? How to store it?what is in AX?
MOV BX,[2002H]	Store second value at 2002H? How to store it?what is in BX?
SUB AX,BX	Explain this line?
JNC L1	Check the flag register and explain this line?
INC DX	Explain this line?
L1: MOV [2004],AX	Explain this line?
MOV [2006],DX	Explain this line?
HLT	Explain this line?

3. 16 BIT MULTIPLICATION USING 8086

MOV DX,0000	Explain this line?
MOV SI,2000H	Explain this line?Check and write the value of SI
MOV AX,[SI]	Explain this line?
MOV CX,[2002]	Explain this line?
MUL CX	Explain this line?
MOV SI,2100H	Explain this line?
MOV [SI],AX	Check the location 2100H and write answer?
MOV [2102],DX	Check the location 2102H and write answer? Why some Value is coming in DX?
HLT	

4. 16 BIT DIVISION USING 8086

MOV DX,0000	Explain this line?
MOV SI,2000H	Explain this line?Check and write the value of SI
MOV AX,SI	Explain this line?
MOV CX,[2002H]	Explain this line?
DIV CX	Explain this line?
MOV SI,2100H	Explain this line?
MOV [SI],AX	Check the location 2100H and write answer?

MOV [2102],DX	Check the location 2102H and write answer? Why some Value is coming in DX?
HLT	

(5)ACCENDING ORDER PROGRAM

ORG 0100H	
MOV SI,1200H	Explain this line?
MOV CL,[SI]	Explain this line?
DEC CL	Explain this line?
LOOP2: MOV SI,1200H	Explain this line?
MOV CH,[SI]	Explain this line?
DEC CH	Explain this line?
INC SI	Explain this line?
LOOP1:MOV AL,[SI]	Explain this line?
INC SI	Explain this line?
CMP AL,[SI]	Explain this line?
JC LOOP3	Explain this line?
XCHG AL,[SI]	Explain this line?
XCHG [SI-1],AL	Explain this line?
LOOP3:DEC CH	Explain this line?
JNZ LOOP1	Explain this line?
DEC CL	Explain this line?
JNZ LOOP2	Explain this line?
HLT	Explain this line?

A. What is the purpose of CL and CH load values?
B. What is purpose of LOOP1,LOOP2,LOOP3?

(6) Check the descending order program

```
ORG 0100H

MOV SI,1200H

MOV CL,[SI]

DEC CL

LOOP2: MOV SI,1200H

MOV CH,[SI]

DEC CH

INC SI
```

```
LOOP1:MOV AL,[SI]
INC SI
CMP AL,[SI]
JC LOOP3
XCHG AL,[SI]
XCHG [SI-1],AL
LOOP3:DEC CH
JNZ LOOP1
DEC CL
JNZ LOOP2
HLT
```

(7) EVEN ODD NUMBER PROGRAM

ORG 00H	Comments
MOV SI,5000H	
MOV CL,04H ;	Load COUNT
MOV BL,00H ;	INITIALIZE THE RESULT BECOME 00H (COUNT FOR +VE NO)
MOV BH,00H ;	INITIALIZE THE RESULT BECOME 00H (COUNT FOR -VE NO)
LOOP: MOV AL,[SI]	
RCR AL,01H ;	(ROTATE THE ACCUMULATOR LOWER BYTE (AL) RIGHT WITH CARRY BY 1 BIT POSITION)
JC NEGATIVE ;	cy = 1 then it jumps otherwise it goto next step
INC BL	
JMP REPEAT ;	
NEGATIVE:INC BH ;	
REPEAT:INC SI ;	
DEC CL ;	
JNZ LOOP ;	JUMP IF NO ZERO FOR PREVIOUS INSTRUCTION
HLT	

1. What is the function of RCR AL,01H?
2. What is logic to identify the number is even or odd? (Hint check LSB of Number)
3. What are JC and JNZ commands?
4. What is the functioning of LOOP and REPEAT label?

(7) POSITIVE AND NEGATIVE NUMBER

ORG 100H	What is the role of these four lines?
.DATA	
NUM1 DB 85H,0D2H,50H,32H,0A6H,79H,98H,0F7H	
LEN EQU 08H	
.CODE	
MOV AX,@DATA	
MOV DS,AX	
LEA SI,NUM1	
MOV BX,0000H	
MOV CL,LEN	
AGAIN: MOV AL,[SI]	
RCL AL,01	
JNC PLUS	
INC BL	
JMP NXT	
PLUS: INC BH	
NXT: INC SI	
DEC CL	
JNZ AGAIN	
RET	