Microprocessor - 8086 Overview

8086 Microprocessor is an enhanced version of 8085Microprocessor that was designed by Intel in 1976. It is a 16-bit Microprocessor having 20 address lines and16 data lines that provides up to 1MB storage. It consists of powerful instruction set, which provides operations like multiplication and division easily.

It supports two modes of operation, i.e. Maximum mode and Minimum mode. Maximum mode is suitable for system having multiple processors and Minimum mode is suitable for system having a single processor.

Features of 8086

The most prominent features of a 8086 microprocessor are as follows −

- It has an instruction queue, which is capable of storing six instruction bytes from the memory resulting in faster processing.
- It was the first 16-bit processor having 16-bit ALU, 16-bit registers, internal data bus, and 16-bit external data bus resulting in faster processing.
- It is available in 3 versions based on the frequency of operation −
    - 8086 → 5MHz
    - 8086-2 → 8MHz
    - (c)8086-1 → 10 MHz
- It uses two stages of pipelining, i.e. Fetch Stage and Execute Stage, which improves performance.
- Fetch stage can prefetch up to 6 bytes of instructions and stores them in the queue.
- Execute stage executes these instructions.
- It has 256 vectored interrupts.
- It consists of 29,000 transistors.

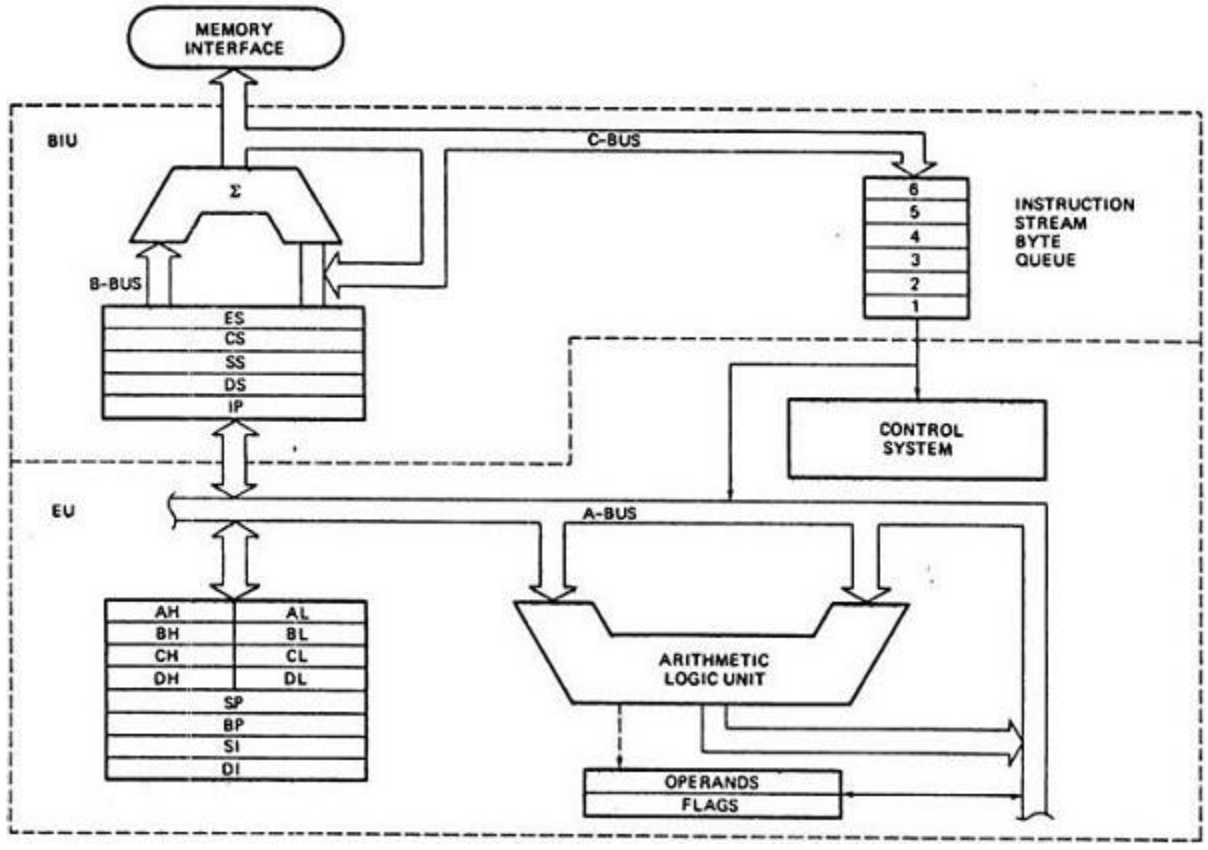
Comparison between 8085 & 8086 Microprocessor

- **Size** − 8085 is 8-bit microprocessor, whereas 8086 is 16-bit microprocessor.
- **Address Bus** − 8085 has 16-bit address bus while 8086 has 20-bit address bus.
- **Memory** − 8085 can access up to 64Kb, whereas 8086 can access up to 1 Mb of memory.

- **Instruction** − 8085 doesn’t have an instruction queue, whereas 8086 has an instruction queue.
- **Pipelining** − 8085 doesn’t support a pipelined architecture while 8086 supports a pipelined architecture.
- **I/O** − 8085 can address 2^8 = 256 I/O's, whereas 8086 can access 2^16 = 65,536 I/O's.
- **Cost** − The cost of 8085 is low whereas that of 8086 is high.

Architecture of 8086

The following diagram depicts the architecture of a 8086 Microprocessor −

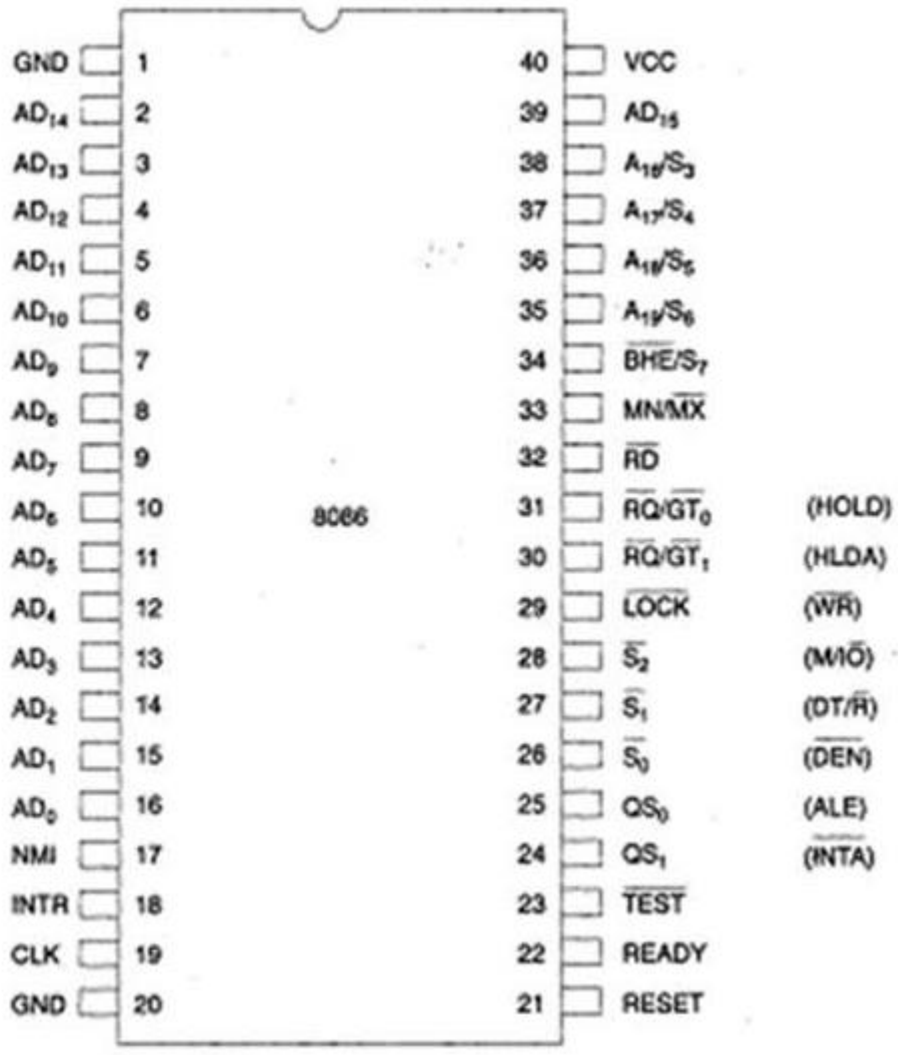


Microprocessor - 8086 Pin Configuration

8086 was the first 16-bit microprocessor available in 40-pin DIP (Dual Inline Package) chip. Let us now discuss in detail the pin configuration of a 8086 Microprocessor.

8086 Pin Diagram

Here is the pin diagram of 8086 microprocessor −



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.

MINIMUM MODE MAXIMUM 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/MX¯¯¯¯¯¯¯¯MN/MX¯ is 1 to indicate minimum mode.	MN/MX¯¯¯¯¯¯¯¯MN/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.
DEN¯¯¯¯¯¯¯¯¯DEN¯and DT/R¯¯¯DT/R¯ for the trans-receivers are given by 8086 itself.	andDT/R¯¯¯DT/R¯ for the trans-receivers are given by 8288 bus controller.
Direct control signals M/IO¯¯¯¯¯M/IO¯, RD¯¯¯¯¯¯RD¯and WR¯¯¯¯¯¯¯WR¯ are given by 8086.	Instead of control signals, each processor generates status signals called S_2¯¯¯¯S2¯, S_1¯¯¯¯S1¯ and S_0¯¯¯¯S0¯.
Control signals M/IO¯¯¯¯¯M/IO¯, RD¯¯¯¯¯¯RD¯ and WR¯¯¯¯¯¯¯WR¯ are decoded by a 3:8 decoder like 74138.	Status signals S_2¯¯¯¯S2¯, S_1¯¯¯¯S1¯ and S_0¯¯¯¯S0¯ are decoded by a bus controller like 8288 to produce control signals.
INTA¯¯¯¯¯¯¯¯¯¯¯INTA¯ is given by 8086 in response to an interrupt on INTR line.	INTA¯¯¯¯¯¯¯¯¯¯¯INTA¯ is given by 8288 bus controller in response to

Minimum mode	**Maximum mode**
	an interrupt on INTR line.
HOLD and HLDA signals are used for bus request with a DMA controller like 8237.	$\overline{RQ}$/$\overline{GT}$RQ¯/GT¯,line s 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.

INTERNAL OPERATION

It requires single phase clock with 33% duty cycle to provide **internal** timing.**8086** is designed to **operate** in two modes, Minimum and Maximum. It can prefetches upto 6 instruction bytes from memory and queues them in order to speed up instruction execution. It requires +5V power supply.

Microprocessor - 8086 Instruction Sets

The 8086 microprocessor supports 8 types of instructions −

- Data Transfer Instructions
- Arithmetic Instructions
- Bit Manipulation Instructions
- String Instructions
- Program Execution Transfer Instructions (Branch & Loop Instructions)
- Processor Control Instructions
- Iteration Control Instructions
- Interrupt Instructions

Let us now discuss these instruction sets in detail.

Data Transfer Instructions

These instructions are used to transfer the data from the source operand to the destination operand. Following are the list of instructions under this group −

Instruction to transfer a word

- **MOV** − Used to copy the byte or word from the provided source to the provided destination.
- **PPUSH** − Used to put a word at the top of the stack.
- **POP** − Used to get a word from the top of the stack to the provided location.
- **PUSHA** − Used to put all the registers into the stack.
- **POPA** − Used to get words from the stack to all registers.
- **XCHG** − Used to exchange the data from two locations.
- **XLAT** − Used to translate a byte in AL using a table in the memory.

Instructions for input and output port transfer

- **IN** − Used to read a byte or word from the provided port to the accumulator.
- **OUT** − Used to send out a byte or word from the accumulator to the provided port.

Instructions to transfer the address

- **LEA** − Used to load the address of operand into the provided register.
- **LDS** − Used to load DS register and other provided register from the memory
- **LES** − Used to load ES register and other provided register from the memory.

Instructions to transfer flag registers

- **LAHF** − Used to load AH with the low byte of the flag register.
- **SAHF** − Used to store AH register to low byte of the flag register.
- **PUSHF** − Used to copy the flag register at the top of the stack.
- **POPF** − Used to copy a word at the top of the stack to the flag register.

Arithmetic Instructions

These instructions are used to perform arithmetic operations like addition, subtraction, multiplication, division, etc.

Following is the list of instructions under this group −

Instructions to perform addition

- **ADD** − Used to add the provided byte to byte/word to word.
- **ADC** − Used to add with carry.
- **INC** − Used to increment the provided byte/word by 1.
- **AAA** − Used to adjust ASCII after addition.
- **DAA** − Used to adjust the decimal after the addition/subtraction operation.

Instructions to perform subtraction

- **SUB** − Used to subtract the byte from byte/word from word.
- **SBB** − Used to perform subtraction with borrow.
- **DEC** − Used to decrement the provided byte/word by 1.
- **NPG** − Used to negate each bit of the provided byte/word and add 1/2's complement.
- **CMP** − Used to compare 2 provided byte/word.
- **AAS** − Used to adjust ASCII codes after subtraction.
- **DAS** − Used to adjust decimal after subtraction.

Instruction to perform multiplication

- **MUL** − Used to multiply unsigned byte by byte/word by word.
- **IMUL** − Used to multiply signed byte by byte/word by word.
- **AAM** − Used to adjust ASCII codes after multiplication.

Instructions to perform division

- **DIV** − Used to divide the unsigned word by byte or unsigned double word by word.
- **IDIV** − Used to divide the signed word by byte or signed double word by word.
- **AAD** − Used to adjust ASCII codes after division.
- **CBW** − Used to fill the upper byte of the word with the copies of sign bit of the lower byte.
- **CWD** − Used to fill the upper word of the double word with the sign bit of the lower word.

Bit Manipulation Instructions

These instructions are used to perform operations where data bits are involved, i.e. operations like logical, shift, etc.

Following is the list of instructions under this group −

Instructions to perform logical operation

- **NOT** − Used to invert each bit of a byte or word.
- **AND** − Used for adding each bit in a byte/word with the corresponding bit in another byte/word.
- **OR** − Used to multiply each bit in a byte/word with the corresponding bit in another byte/word.
- **XOR** − Used to perform Exclusive-OR operation over each bit in a byte/word with the corresponding bit in another byte/word.
- **TEST** − Used to add operands to update flags, without affecting operands.

Instructions to perform shift operations

- **SHL/SAL** − Used to shift bits of a byte/word towards left and put zero(S) in LSBs.
- **SHR** − Used to shift bits of a byte/word towards the right and put zero(S) in MSBs.
- **SAR** − Used to shift bits of a byte/word towards the right and copy the old MSB into the new MSB.

Instructions to perform rotate operations

- **ROL** − Used to rotate bits of byte/word towards the left, i.e. MSB to LSB and to Carry Flag [CF].
- **ROR** − Used to rotate bits of byte/word towards the right, i.e. LSB to MSB and to Carry Flag [CF].
- **RCR** − Used to rotate bits of byte/word towards the right, i.e. LSB to CF and CF to MSB.
- **RCL** − Used to rotate bits of byte/word towards the left, i.e. MSB to CF and CF to LSB.

String Instructions

String is a group of bytes/words and their memory is always allocated in a sequential order.

Following is the list of instructions under this group −

- **REP** − Used to repeat the given instruction till $CX \neq 0$.
- **REPE/REPZ** − Used to repeat the given instruction until $CX = 0$ or zero flag $ZF = 1$.
- **REPNE/REPNZ** − Used to repeat the given instruction until $CX = 0$ or zero flag $ZF = 1$.
- **MOVS/MOVSB/MOVSW** − Used to move the byte/word from one string to another.
- **COMS/COMPSB/COMPSW** − Used to compare two string bytes/words.
- **INS/INSB/INSW** − Used as an input string/byte/word from the I/O port to the provided memory location.

- **OUTS/OUTSB/OUTSW** − Used as an output string/byte/word from the provided memory location to the I/O port.
- **SCAS/SCASB/SCASW** − Used to scan a string and compare its byte with a byte in AL or string word with a word in AX.
- **LODS/LODSB/LODSW** − Used to store the string byte into AL or string word into AX.

Program Execution Transfer Instructions (Branch and Loop Instructions)

These instructions are used to transfer/branch the instructions during an execution. It includes the following instructions −

Instructions to transfer the instruction during an execution without any condition −

- **CALL** − Used to call a procedure and save their return address to the stack.
- **RET** − Used to return from the procedure to the main program.
- **JMP** − Used to jump to the provided address to proceed to the next instruction.

Instructions to transfer the instruction during an execution with some conditions −

- **JA/JNBE** − Used to jump if above/not below/equal instruction satisfies.
- **JAE/JNB** − Used to jump if above/not below instruction satisfies.
- **JBE/JNA** − Used to jump if below/equal/ not above instruction satisfies.
- **JC** − Used to jump if carry flag CF = 1
- **JE/JZ** − Used to jump if equal/zero flag ZF = 1
- **JG/JNLE** − Used to jump if greater/not less than/equal instruction satisfies.
- **JGE/JNL** − Used to jump if greater than/equal/not less than instruction satisfies.
- **JL/JNGE** − Used to jump if less than/not greater than/equal instruction satisfies.
- **JLE/JNG** − Used to jump if less than/equal/if not greater than instruction satisfies.
- **JNC** − Used to jump if no carry flag (CF = 0)
- **JNE/JNZ** − Used to jump if not equal/zero flag ZF = 0
- **JNO** − Used to jump if no overflow flag OF = 0
- **JNP/JPO** − Used to jump if not parity/parity odd PF = 0
- **JNS** − Used to jump if not sign SF = 0
- **JO** − Used to jump if overflow flag OF = 1
- **JP/JPE** − Used to jump if parity/parity even PF = 1
- **JS** − Used to jump if sign flag SF = 1

Processor Control Instructions

These instructions are used to control the processor action by setting/resetting the flag values.

Following are the instructions under this group −

- **STC** − Used to set carry flag CF to 1
- **CLC** − Used to clear/reset carry flag CF to 0
- **CMC** − Used to put complement at the state of carry flag CF.
- **STD** − Used to set the direction flag DF to 1
- **CLD** − Used to clear/reset the direction flag DF to 0
- **STI** − Used to set the interrupt enable flag to 1, i.e., enable INTR input.
- **CLI** − Used to clear the interrupt enable flag to 0, i.e., disable INTR input.

Iteration Control Instructions

These instructions are used to execute the given instructions for number of times. Following is the list of instructions under this group −

- **LOOP** − Used to loop a group of instructions until the condition satisfies, i.e., CX = 0
- **LOOPE/LOOPZ** − Used to loop a group of instructions till it satisfies ZF = 1 & CX = 0
- **LOOPNE/LOOPNZ** − Used to loop a group of instructions till it satisfies ZF = 0 & CX = 0
- **JCXZ** − Used to jump to the provided address if CX = 0

Interrupt Instructions

These instructions are used to call the interrupt during program execution.

- **INT** − Used to interrupt the program during execution and calling service specified.
- **INTO** − Used to interrupt the program during execution if OF = 1
- **IRET** − Used to return from interrupt service to the main program.

Timing diagram of MOV Instruction

```
Operand: B and C
```

Bis the destination register and C is the source register whose contents need to be transferred to the destination register.

Algorithm –
The instruction MOV B, C is of 1 byte; therefore the complete instruction will be stored in a single memory address. For example:

Problem – Draw the timing diagram of the given instruction in 8085,

```
MOV B, C
```

Given instruction copies the contents of the source register into the destination register and the contents of the source register are not altered.

Example:

```
MOV B, C

Opcode: MOV
```

```
2000: MOV B, C
```

Only opcode fetching is required for this instruction and thus we need 4 T states for the timing diagram. For the opcode fetch the IO/M (low active) = 0, S1 = 1 and S0 = 1.

In Opcode fetch (t1-t4 T states):

1. **00 –** lower bit of address where opcode is stored, i.e., 00
2. **20 –** higher bit of address where opcode is stored, i.e., 20.
3. **ALE –** provides signal for multiplexed address and data bus. Only in t1 it used as address bus to fetch lower bit of address otherwise it will be used as data bus.
4. **RD (low active) –** signal is 1 in t1 & t4 as no data is read by microprocessor. Signal is 0 in t2 & t3 because here the data is read by microprocessor.
5. **WR (low active) –** signal is 1 throughout, no data is written by microprocessor.

6. **IO/M (low active) –** signal is 1 in throughout because the operation is performing on memory.
7. **S0 and S1 –** both are 1 in case of opcode fetching.

ASSEMBLER INSTRUCTION FORMAT

For every **instruction that is executed in the 8086 microprocessor**, an **instruction format** is available that is the binary representation of that instruction.

This instruction format can be coded from 1 to 6 bytes depending upon the addressing modes used for instructions.

The general Instruction format that most of the **instructions of the 8086 microprocessor**follow is:

- The Opcode stands for Operation Code. Every Instruction has a unique 6-bit opcode. For example, the opcode for **MOV** is 100010.
- **D** stands for direction
  If **D=0**, then the direction is from the register
  If **D=1**, then the direction is to the register
- **W** stands for word
  If **W=0**, then only a byte is being transferred, i.e. 8 bits
  If **W=1**, them a whole word is being transferred, i.e. 16 bits
- The **MOD** and **R/M** together is calculated based upon the addressing mode and register being used in it. This is calculated as follows:

R/M	**0 0 (Memory Mode with no displacement)**	**0 1 (Memory mode with 8 bit displacement)**	**1 0 (Memory Mode with 16 bit displacement)**	**1 1 (Register Mode)**
000	[BX] + [SI]	[BX] + [SI] + d8	[BX] + [SI] + d16	AL AX

001	[BX] + [DI]	[BX] + [DI] + d8	[BX] + [DI] + d16	CL CX
010	[BP] + [SI]	[BP] + [SI] + d8	[BP] + [SI] + d16	DL DX
011	[BP] + [DI]	[BP] + [DI] + d8	[BP] + [DI] + d16	BL BX
100	[SI]	[SI] + d8	[SI] + d16	AH SP
101	[DI]	[DI] + d8	[DI] + d16	CH BP
110	d16 (direct)	[BP] + d8	[BP] + d16	DH SI
111	[BX]	[BX] + d8	[BX] + d16	BH DI

- REG stands for register selected. It is a 3-bit code which is calculated as follows:

REG Code	**Register Selected**
0 0 0	AL AX
0 0 1	CL CX
0 1 0	DL DX
0 1 1	BL BX
1 0 0	AH SP
1 0 1	CH BP
1 1 0	DH SI
1 1 1	BH DI

- The low order displacement and high order displacement are optional and the instruction format contains them only if there exists any displacement in the instruction. If the displacement is of 8 bits, then only the cell of low order displacement infilled and if the displacement is of 16 bits, then both the cells od low order and high order are filled, with the exact bits that the displacement number represents.