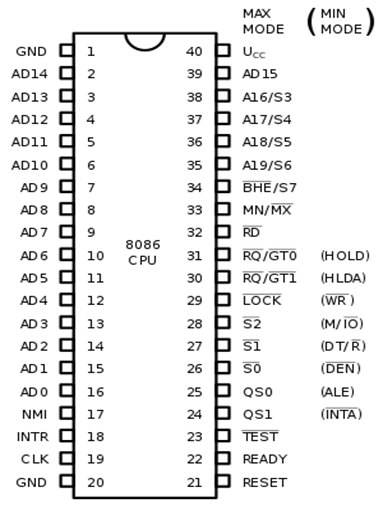
**Microprocessor** is a multipurpose, programmable, clock-driven, register-based electronic device that reads and processes binary instructions.

**Microprocessor 8086** is 40 pin IC having 16 bit processor made by intel in 1976 with VLSI (Very Large Scale Integration where 29000 transistor were fabricated on a single chip) technology, using HMOS (Hybrid metal oxide semiconductor) material.

* **Instruction Set:** Group of commands that microprocessor can understand
* **Bus:** Set of conductors that transmit data, address or information. It can be of three types i.e., data bus, address bus and control bus.
* **IPC (Instructions Per Cycle):** Measure of how many instructions CPU executes in a single clock
* **Clock Speed/Rate:** Number of operations per second (in MHz or GHz)
* **Bandwidth:** Number of bits processed in a single instruction
* **Word Length:** Number of bits processed at a time
* **Interrupt:** Breaks normal sequence of operations/instructions.
* **Interrupt Vector:** Memory location of an interrupt handler (used to prioritize interrupts).

**Microprocessor Architecture:**



* **GND, VCC** (Ground and Voltage pin is used for power supply)
* **CLK** (Clock provides basic timing of operation, generally 5 MHz square wave)
* **NMI** (Non-Maskable positive edge triggered high level Interrupt)
* **INTR** (High level maskable (can be disable) Interrupt Request)
* **AD0-AD7** (carry lower order 16 byte data)
* **AD8-AD15** (carry higher order 16 byte data)
* **A16-A19** (carry higher order 4-bit address for memory operations)
* **BHE** (Bus High Enable, enables data on most significant half of data bus)
* **MN/MX** (Single/ Multiple processors in the system for performing instructions)
* **RD’** (read signal from memory)
* **RQ/GT0’** and **RQ/GT1’** (Request/Grant Pins sends acknowledge signals)
* **LOCK** (indicates that other processes should not ask for CPU system bus)
* **S2/S1/S0** (Status signals indicates operations being done by microprocessor)
* **QS0/QS1** (provides status of instruction queue)
* **HOLD** (issues HLDA signal to DMA controller)
* **HLDA** (It is an active high output signal issued after receiving HOLD signal)
* **WR’** (Write signal from memory)
* **M/IO’** (Distinguish memory from I/O access)
* **DT/R’** (Data Transmit/Receive output signal decides direction of data flow through the transceiver)
* **DEN** (Data Enable Signal used to enable the transceiver for separating data from data bus)
* **ALE** (Address Latch Enable indicates availability of valid address on address lines AD0-AD15)
* **INTA’** (Interrupt Acknowledge, acknowledges device which gives the signal on receive INTR signal)
* **TEST’** (used to test status of math co-processor)
* **READY** (indicates that device is ready to transfer data)
* **RESET** (terminates the current activity)

**Addressing modes:** Rules for interpreting/modifying address field of the instruction

* **Segment Register Addressing Mode:** CS/ DS/ ES/ SS (Code/ Data/ Extra/ Stack segment register used for addressing data/stack segment of memory)
* **Register Addressing Mode:** AX/ BX/ CX/ DX (Accumulator/ Base/ Counter/ Data Register used for logical instructions/ base address/ default counter/ data 16 bit address in memory while AL, BL, CL in 8-bit memory)
* **Index Register Addressing Mode:** SP/ BP/ IP/ SI/ DI (Stack Pointer/ Base Pointer/ Instruction Pointer/ Source Index/ Destination Index contains offset within stack/data/code/source/destination index)
* **Flag Register Addressing Mode:** Each flag indicates the results of computation in ALU
* **Immediate Addressing Mode:** 16-bit (like 3200H, 9A32H) and 8-bit (like 30H, 0A6H)

Memory address is of 5 digits, if 4 digits given then CPU attach one zero by itself.

**Physical address=** Segment address\*10H + offset address

**Assembler Directives:** Provides direction to the assembler for performing specific actions.

* **DB/ DW/ DD/ DQ/ DT** (used to direct assembler to Define Byte/ Word/ Double/ Quad/ Ten Bytes)
* **ASSUME** (Assume Logical Segment Name used to tell assembler the name of logical segment)
* **END** (tells the End of assembly language program)
* **ENDP** (tells the End of procedure/subroutines)
* **ENDS** (tells the End of logical segment)
* **PROC** (indicates start of named Procedure)
* **SEGMENT** (indicates start of logical segment)
* **SEG** (decide the segment address of label, variable or procedure)
* **EVEN** (instructs assembler to increment location of counter to next even address)
* **LENGTH** (determines number of elements in a data item)
* **GROUP** (Form logical group of segments having similar purpose)
* **EQU** (Equate is used to given name to some value)
* **PTR** (Pointer is used to declare type of label, variable or memory)
* **ORG** (Origin specifies the starting point of memory area)
* **NAME** (Give specific name to each assembly module)
* **INCLUDE** (Used to include some source code from named into current source module)
* **OFFSET** (Used to determine offset or displacement of named data item)
* **EXTERN/PUBLIC** (Used to call procedure of another module into current module)
* **SHORT** (Indicates assembler that only one byte is required to code the displacement for the jump)
* **FAR PTR** (Indicates assembler that label following FAR PTR is not available)
* **NEAR PTR** (Indicates assembler that label following NEAR PTR is in the same segment)
* **LOCAL** (Labels, variables, constants, procedures declared by Local are used by that particular module)
* **GLOBAL** (Labels, variables, constants, procedures declared by Local can be used by other modules of the program)
* **LABEL** (Used to assign name to the current content of location counter)
* **TYPE** (Determines number of bytes specified to that variable)

**Instruction Set:** There are 8 types of instruction set i.e. Data transfer, Logical, Branch, Loop, Machine Control, Flag Manipulation, Shift/Rotate and String Instructions.

* **MOV** CX, DX {Moving data from DX to CX}

Transfer 9A32H into DS => MOV AX, 9A32H MOV DS,AX {**Directly moving Immediate data into segment register is not allowed**. Writing MOV DS, 5000H is invalid.

* **LEA** (Load Effective Address) LEA BX, [DI]
* **LDS/LES** (Load memory double word into word register and DS/ES) LDS BX,[4326h]
* **LAHF/SAHF** (Load/ Store AH from lower byte of Flag register)
* **IN** (Input from port into AL or AX)
* **OUT** (Output from AL or AX to port)
* **CTC/STC** (Clear/Set Carry flag)
* **CMC** (Complement Carry flag)
* **STD** (Set Direction flag)
* **STI** (Set Interrupt enable flag)
* **CLI** (Clear Interrupt enable flag)
* **CLD** (Clear Direction flag)
* **NEG** (Negate makes operand negative/ 2’s complement)
* **XLAT** (Replace byte in AL with byte from user table)
* **CWD** (Convert Signed Word (like AX) to Signed Double Word (like DX:AX))
* **CBW** (Convert Signed Byte (like AL) to Signed Word (like AX))
* **MOVSB/ MOVSW** (Copy Byte/ Word at DS:[SI] to ES:[DI])
* **LODSB/ LODSW** (Load Byte/ Word at DS:[SI] into AL/AX)
* **STOSB/STOSW** (Store Byte/ Word in AL/AX into ES:[DI])
* **CMP** (Compare byte or word)
* **CMPSB/ CMPSW** (Compare Byte/ Word ES:[DI] from DS:[SI])
* **SCASB/ SCASW** (Compare Byte/ Word AL/AX from ES:[DI])
* **TEST** (Logical AND between all bits of two operands for flags only)
* **REP** (Repeat MOVSB, MOVSW, LODSB, LODSW, STOSB, STOSW instruction CX times)
* **REPE/REPNE** (Repeat CMPSB, CMPSW, SCASB, SCASW instruction CX times while result is Equal/ Not Equal)
* **REPZ/ REPNZ** (Repeat CMPSB, CMPSW, SCASB, SCASW instruction CX times while result is Zero/ Not Zero)
* **CALL** (Transfer control to procedure and returned IP address is pushed to stack)
* **JMP** (Transfer control to specific destination)
* **JA/JNBE** (Jump Above/ Jump if Not Below or Equal: Jump first operand above second operand)
* **JAE/JNB/JNC** (Jump if Above/ Not Below/ No Carry)
* **JB/JC/JNAE** (Jump if first operand is Below second operand)
* **JBE/JNA** (Jump if first operand is Below or Equal to second operand)
* **JCXZ** (Jump if CX register is 0)
* **JE/JZ** (Jump if first operand is Equal to second operand)
* **JG/JNLE** (Jump if first operand is Greater than second operand)
* **JGE/JNL** (Jump if first operand is Greater or Equal to second operand)
* **JL/JNGE** (Jump if first operand is Less than second operand)
* **JLE/JNG** (Jump if first operand is Less or Equal to second operand)
* **JNE/JNZ** (Jump if first operand is Not Equal to second operand)
* **JO/JNO** (Jump if Overflow/ Not Overflow)
* **JS/JNS** (Jump if Signed/ Not Signed)
* **JNP/JPO** (Jump if No Parity (odd))
* **JP/JPE** (Jump if Parity (even))
* **RET** (Return execution from procedure to calling program)
* **RETF** (Return from FAR Procedure)
* **LOOP** (Decrease CX, jump to label if CX not zero)
* **LOOPE/ LOOPNE** (Decrease CX, jump to label if CX not zero and Equal/ Not Equal)
* **LOOPZ/ LOOPNZ** (Decrease CX, jump to label if CX not zero and ZF=1/ ZF=0)
* **HLT** (Halt the system)
* **NOP** (No Operation)
* **ESP** (Free the bus for an external master)
* **LOCK** (Lock doesn’t allow another master to execute instruction)
* **WAIT** (Holds the operation of processor till TEST pin goes low)
* **IRET** (Interrupt Return)
* **AAA/ DAA** (ASCII/ Decimal Adjust after Addition)
* **IMUL/ IDIV** (Signed Multiplication/ Divide)