PNS SCHOOL OF ENGINEERING & TECHNOLOGY

Nishamani vihar, Marshaghai, Kendrapara

LECTURE NOTES

ON

MICROPROCESSOR AND MICROCONTROLLER

DEPARTMENT OF ELECTRONIC & TELECOMMUNICATION ENGINEERING

4th Semester

PREPARED BY

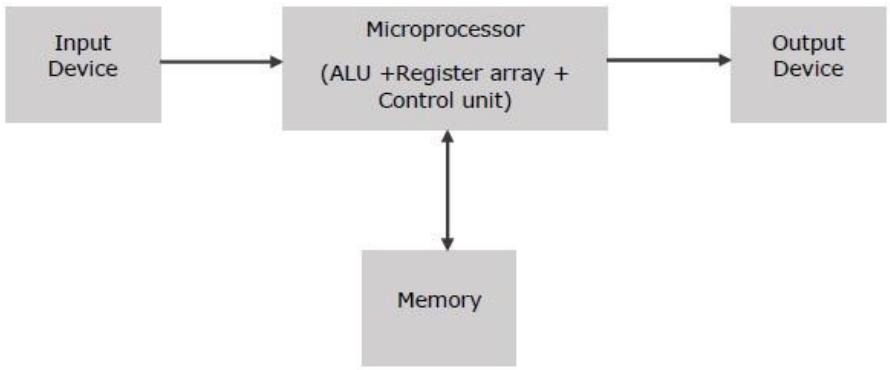
MR. ADITYA NARAYAN JENA

LECTURER IN ELECTRONICS & TELECOMMUNICATION

UNIT-1
8085 Microprocessor

- A Microprocessor is a multipurpose, programmable, clock-driven, register-based electronic device that reads binary instructions from a storage device called memory, accepts binary data as input and processes data according to those instructions and provide results as output.
- Microprocessor is a controlling unit of a micro-computer, fabricated on a small chip capable of performing ALU (Arithmetic Logical Unit) operations and communicating with the other devices connected to it.
- Microprocessor consists of an ALU, register array, and a control unit.
- ALU performs arithmetical and logical operations on the data received from the memory or an input device.
- Register array consists of registers identified by letters like B, C, D, E, H, L and accumulator.
- The control unit controls the flow of data and instructions within the computer.

Block Diagram of a Basic Microcomputer



How does a Microprocessor Work?

The microprocessor follows a sequence: Fetch, Decode, and then Execute.

- Initially, the instructions are stored in the memory in a sequential order.
- The microprocessor fetches those instructions from the memory, then decodes it and executes those instructions till STOP instruction is reached.
- Later, it sends the result in binary to the output port. Between these processes, the register stores the temporarily data and ALU performs the computing functions.

Classification of Microprocessor

Microprocessor is classified into two categories-

RISC & CISC

RISC Processor

- RISC stands for Reduced Instruction Set Computer.
- It is designed to reduce the execution time by simplifying the instruction set of the computer.
- Using RISC processors, each instruction requires only one clock cycle to execute results in uniform execution time.
- This reduces the efficiency as there are more lines of code, hence more RAM is needed to store the instructions.
- The compiler also has to work more to convert high-level language instructions into machine code.

Characteristics of RISC

The major characteristics of a RISC processor are as follows −

- It consists of simple instructions.
- It supports various data-type formats.
- It utilizes simple addressing modes and fixed length instructions for pipelining.
- It supports register to use in any context.
- One cycle execution time.
- “LOAD” and “STORE” instructions are used to access the memory location.
- It consists of larger number of registers.
- It consists of less number of transistors.

CISC Processor

- CISC stands for Complex Instruction Set Computer.
- It is designed to minimize the number of instructions per program, ignoring the number of cycles per instruction.
- The emphasis is on building complex instructions directly into the hardware.
- The compiler has to do very little work to translate a high-level language into assembly level language/machine code because the length of the code is relatively short, so very little RAM is required to store the instructions. Characteristics of CISC
- Variety of addressing modes.
- Larger number of instructions.
- Variable length of instruction formats.
- Several cycles may be required to execute one instruction.
- Instruction-decoding logic is complex.
- One instruction is required to support multiple addressing modes.

8085 Microprocessor

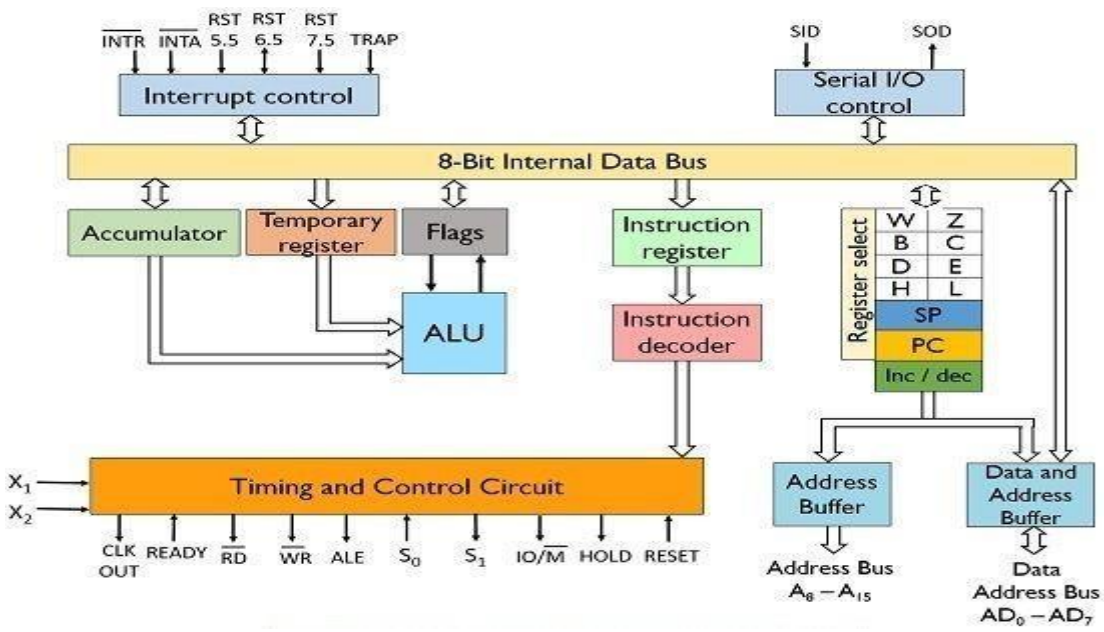
It is an 8-bit microprocessor designed by Intel in 1977 using NMOS technology.

It has the following configuration – □ 8-bit data bus

- 16-bit address bus, which can address upto 64KB
- A 16-bit program counter
- A 16-bit stack pointer
- Six 8-bit registers arranged in pairs: BC, DE, HL
- Requires +5V supply to operate at 3 MHZ single phase clock

It is used in washing machines, microwave ovens, mobile phones, etc.

8085 Microprocessor – Functional Units



8085 consists of the following functional units –

Accumulator

It is an 8-bit register used to perform arithmetic, logical, I/O & LOAD/STORE operations. It is connected to internal data bus & ALU.

Arithmetic and logic unit

As the name suggests, it performs arithmetic and logical operations like Addition, Subtraction, AND, OR, etc. on 8-bit data.

General purpose register

There are 6 general purpose registers in 8085 processor, i.e., B, C, D, E, H & L. Each register can hold 8-bit data.
These registers can work in pair to hold 16-bit data and their pairing combination is like B-C, DE & H-L.

Program counter

It is a 16-bit register used to store the memory address location of the next instruction to be executed. Microprocessor increments the program whenever an instruction is being executed, so that the program counter points to the memory address of the next instruction that is going to be executed.

Stack pointer

It is also a 16-bit register works like stack, which is always incremented/decremented by 2 during push & pop operations.

Temporary register

It is an 8-bit register, which holds the temporary data of arithmetic and logical operations. **Flag register**

It is an 8-bit register having five 1-bit flip-flops, which holds either 0 or 1 depending upon the result stored in the accumulator. These are the set of 5 flip-flops – **Sign (S)-** set to 1 if result is negative. **Zero (Z)-** set to 1 if result is zero.
Auxiliary Carry (AC)- set to 1 if carry arises from 3rd bit to 4th bit.
Parity (P)- set to 1 if result has even no. of 1.
Carry (CS)- set to 1 if carry arises after arithmetic and logical operation.
Its bit position is shown in the following table –

B7	B6	B5	B4	B3	B2	B1	B0
S	Z	X	AC	X	P	X	CS

Instruction register and decoder

It is an 8-bit register. When an instruction is fetched from memory then it is stored in the Instruction register. Instruction decoder decodes the information present in the Instruction register.

Timing and control unit

It provides timing and control signal to the microprocessor to perform operations. Following are
the timing and control signals, which control external and internal circuits – Control Signals: READY, RD', WR', ALE
Status Signals: S0, S1, IO/M'
DMA Signals: HOLD, HLDA
RESET Signals: RESET IN, RESET OUT

Interrupt control

As the name suggests it controls the interrupts during a process. When a microprocessor is executing a main program and whenever an interrupt occurs, the microprocessor shifts the control from the main program to process the incoming request. After the request is completed, the control goes back to the main program.
There are 5 interrupt signals in 8085 Microprocessor: INTR, RST 7.5, RST 6.5, RST 5.5, TRAP.

Serial Input/output control

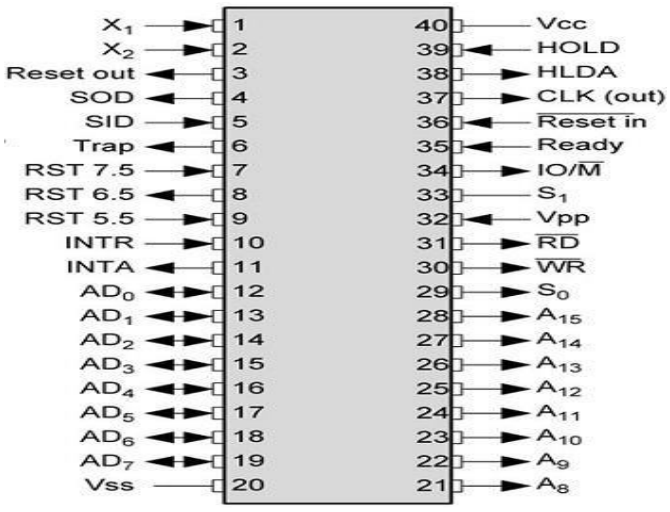
It controls the serial data communication by using these two instructions: SID (Serial input data) and SOD (Serial output data).

Address buffer and address-data buffer

The content stored in the stack pointer and program counter is loaded into the address buffer and address-data buffer to communicate with the CPU. The memory and I/O chips are connected to these buses; the CPU can exchange the desired data with the memory and I/O chips. **Address bus and data bus**

Data bus carries the data to be stored. It is bidirectional, whereas address bus carries the location to where it should be stored and it is unidirectional. It is used to transfer the data & Address I/O devices.

Pin diagram and description



The pins of 8085 microprocessor can be classified into seven groups –

Address bus

A15-A8, it carries the most significant 8-bits of memory/IO address.

Data bus

AD7-AD0, it carries the least significant 8-bit address and data bus.

Control and status signals

These signals are used to identify the nature of operation. There are 3 control signal and 3 status signals.

Three **control signals** are RD, WR & ALE.

RD – This signal indicates that the selected IO or memory device is to be read and is ready for accepting data available on the data bus.

WR – This signal indicates that the data on the data bus is to be written into a selected memory or IO location.

ALE – It is a positive going pulse generated when a new operation is started by the microprocessor. When the pulse goes high, it indicates address. When the pulse goes down it indicates data.

Three **status signals** are IO/M, S0 & S1.

IO/M

This signal is used to differentiate between IO and Memory operations, i.e. when it is high indicates IO operation and when it is low then it indicates memory operation.

S1 & S0

These signals are used to identify the type of current operation.

Power supply

There are 2 power supply signals – VCC & VSS. **VCC** indicates +5v power supply and **VSS** indicates ground signal.

Clock signals

There are 3 **clock signals**, i.e. X1, X2, CLK OUT.

X1, X2 – A crystal (RC, LC N/W) is connected at these two pins and is used to set frequency of the internal clock generator. This frequency is internally divided by 2.

CLK OUT – This signal is used as the system clock for devices connected with the microprocessor.

Interrupts & externally initiated signals

Interrupts are the signals generated by external devices to request the microprocessor to perform a task. There are 5 interrupt signals, i.e., TRAP, RST 7.5, RST 6.5, RST 5.5, and INTR. We will discuss interrupts in detail in interrupts section.

(INTA)' – It is an interrupt acknowledgment signal.

RESET IN – This signal is used to reset the microprocessor by setting the program counter to zero.

RESET OUT – This signal is used to reset all the connected devices when the microprocessor is reset.

READY – This signal indicates that the device is ready to send or receive data. If READY is low, then the CPU has to wait for READY to go high.

HOLD – This signal indicates that another master is requesting the use of the address and data buses.

HLDA (HOLD Acknowledge) – It indicates that the CPU has received the HOLD request and it will relinquish the bus in the next clock cycle. HLDA is set to low after the HOLD signal is removed.

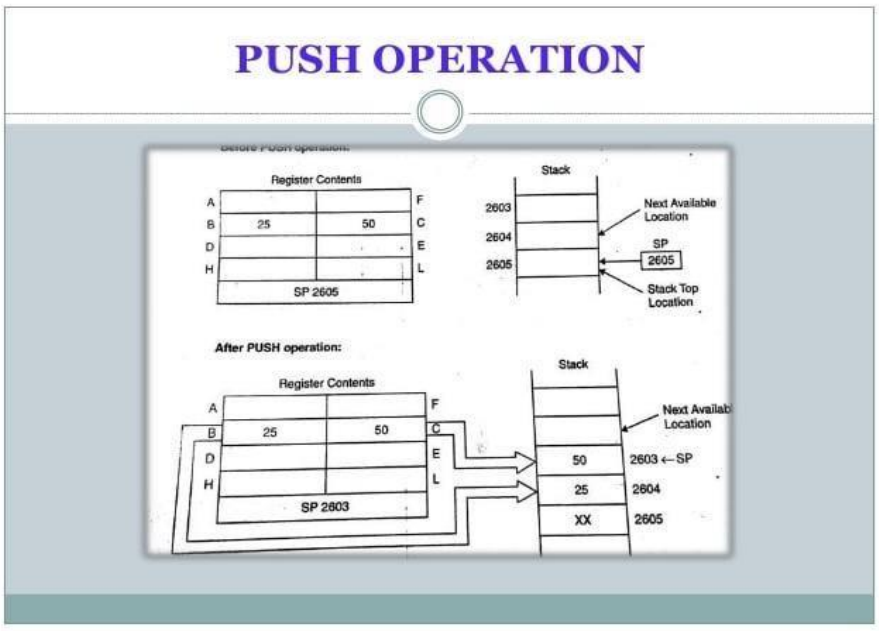
Serial I/O signals

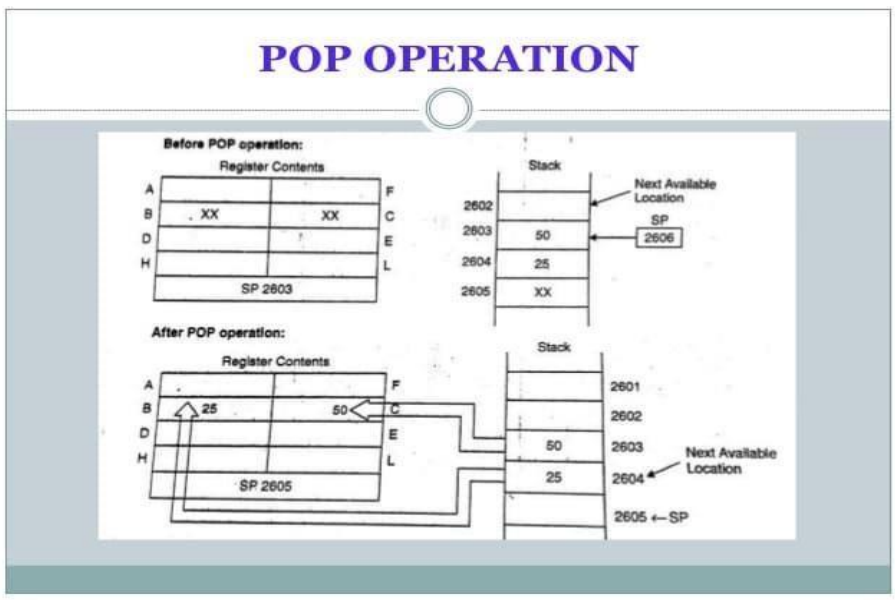
There are 2 serial signals, i.e., SID and SOD and these signals are used for serial communication.

SOD (Serial output data line) – The output SOD is set/reset as specified by the SIM instruction. **SID (Serial input data line)** – The data on this line is loaded into accumulator whenever a RIM instruction is executed.

Stack, stack top and stack pointer

- The stack is a LIFO (last in, first out) data structure implemented in the RAM area and is used to store addresses and data when the microprocessor branches to a subroutine.
- Then the return address used to get pushed on this stack.
- Also, to swap values of two registers and register pairs we use the stack as well.
- The Stack Pointer register will hold the address of the top location of the stack.
- On a stack, we can perform two operations.
- PUSH and POP.
- In case of PUSH operation, the SP register gets decreased by 2 and new data item used to insert on to the top of the stack.
- In case of POP operation, the data item will have to be deleted from the top of the stack and the SP register will get increased by the value of 2.





Interrupts

- When microprocessor receives any interrupt signal from peripheral(s) which are requesting its services, it stops its current execution and program control is transferred to a sub-routine by generating CALL signal and after executing sub-routine by generating RET signal again program control is transferred to main program from where it had stopped.
- When microprocessor receives interrupt signals, it sends an acknowledgement (INTA) to the peripheral which is requesting for its service.

Interrupts can be classified into various categories based on different parameters:

1. Hardware and Software Interrupts –

When microprocessors receive interrupt signals through pins (hardware) of microprocessor, they are known as **Hardware Interrupts**. There are 5 Hardware Interrupts in 8085 Microprocessor. They are – INTR, RST 7.5, RST 6.5, RST 5.5, TRAP.

Software Interrupts are those which are inserted in between the program which means these are mnemonics of microprocessor. There are 8 software interrupts in 8085 Microprocessor. They are – RST 0, RST 1, RST 2, RST 3, RST 4, RST 5, RST 6, RST 7.

2. Vectored and Non-Vectored Interrupts –

Vectored Interrupts are those which have fixed vector address (starting address of sub-routine) and after executing these, program control is transferred to that address.

Vector Addresses are calculated by the formula 8 * TYPE

INTERRUPT	VECTOR ADDRESS
TRAP (RST 4.5)	24 H
RST 5.5	2C H
RST 6.5	34 H
INTERRUPT	VECTOR ADDRESS

RST 7.5	3C H

For Software interrupts vector addresses are given by:

INTERRUPT	VECTOR ADDRESS
RST 0	00 H
RST 1	08 H
RST 2	10 H
RST 3	18 H
RST 4	20 H
RST 5	28 H
RST 6	30 H
RST 7	38 H

Non-Vectored Interrupts are those in which vector address is not predefined. The interrupting device gives the address of sub-routine for these interrupts. INTR is the only non-vectored interrupt in 8085 Microprocessor.

3. Maskable and Non-Maskable Interrupts –

Maskable Interrupts are those which can be disabled or ignored by the microprocessor. These interrupts are either edge-triggered or level-triggered, so they can be disabled. INTR, RST 7.5, RST 6.5, RST 5.5 are maskable interrupts in 8085 Microprocessor.

Non-Maskable Interrupts are those which cannot be disabled or ignored by microprocessor. TRAP is a non-maskable interrupt. It consists of both level as well as edge triggering and is used in critical power failure conditions.

Priority of Interrupts –

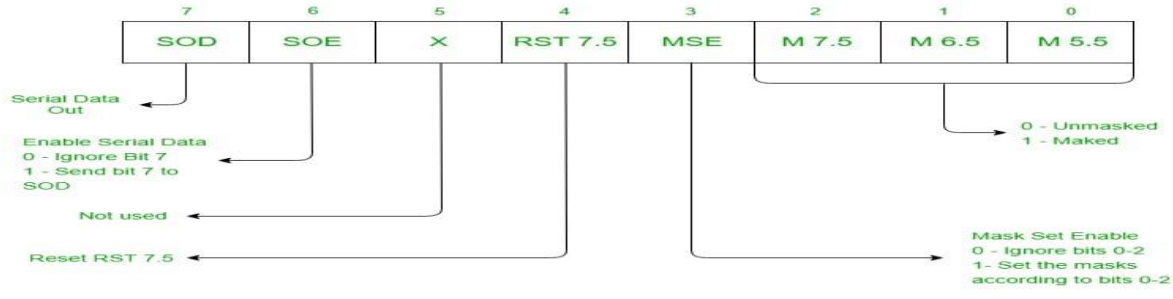
When microprocessor receives multiple interrupt requests simultaneously, it will execute the interrupt service request (ISR) according to the priority of the interrupts.

Instruction for Interrupts –

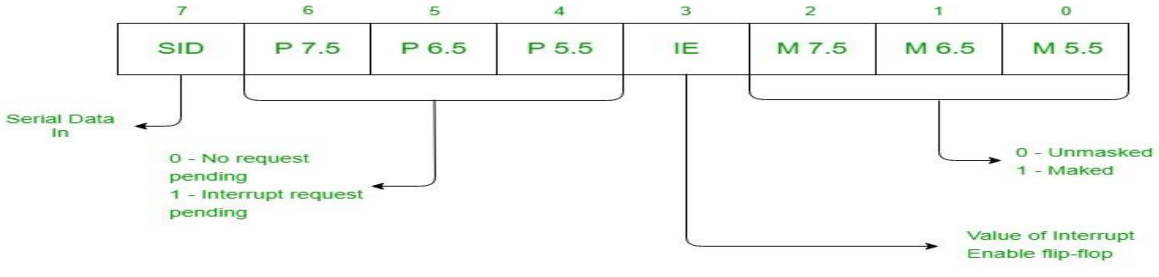
i. **Enable Interrupt (EI) –** The interrupt enable flip-flop is set and all interrupts are enabled following the execution of next instruction followed by EI. No flags are affected. After a system reset, the interrupt enable flip-flop is reset, thus disabling the interrupts. This instruction is necessary to enable the interrupts again (except TRAP).

ii. **Disable Interrupt (DI) –** This instruction is used to reset the value of enable flip-flop hence disabling all the interrupts. No flags are affected by this instruction.

iii. **Set Interrupt Mask (SIM) –** It is used to implement the hardware interrupts (RST 7.5, RST 6.5, RST 5.5) by setting various bits to form masks or generate output data via the Serial Output Data (SOD) line. First the required value is loaded in accumulator then SIM will take the bit pattern from it.



iv. Read Interrupt Mask (RIM) – This instruction is used to read the status of the hardware interrupts (RST 7.5, RST 6.5, RST 5.5) by loading into the A register a byte which defines the condition of the mask bits for the interrupts. It also reads the condition of SID (Serial Input Data) bit on the microprocessor.



UNIT-2 Instruction set and assembly language program

Opcodes and operands

- Instruction is divided into two parts: opcodes and operands.
- The opcode is the instruction that is executed by the CPU and the operand is the data or memory location used to execute that instruction.
- An operand (written using hexadecimal notation) provides the data itself, or the location where the data to be processed is stored.
- Some instructions do not require an operand and some may require more than one operand.

Instruction size

- The 8085 instruction set is classified into 3 categories by considering the length of the instructions.
- Three types of instruction are: 1-byte instruction, 2-byte instruction, and 3-byte instruction.

1. One-byte instructions –
In 1-byte instruction, the opcode and the operand of an instruction are represented in one byte.
Example- MOV A,B

2. Two-byte instructions –
Two-byte instruction is the type of instruction in which the first 8 bits indicates the opcode and the next 8 bits indicates the operand.
Example- MVI A,34H
3. Three-byte instructions –
Three-byte instruction is the type of instruction in which the first 8 bits indicates the opcode and the next two bytes specify the 16-bit address. The low-order address is represented in second byte and the high-order address is represented in the third byte. Example- LDA 2000H

Instruction set of 8085 Microprocessor

8085 instruction set is classified in 5 groups- Data transfer group
Arithmetic group
Logical group
Branch control group
Machine control group

Data transfer group

Data transfer instructions are the instructions which transfers data in the microprocessor. They are also called copy instructions.

Opcode	Operand	Explanation	Example
MOV	R1,R2	Move the data from R2 to R1	MOV A,B
MOV	R,M	Move data from memory location to R	MOV B,M
MVI	R,8-bit data	Move the immediate 8-bit data to R	MVI C,34H
MVI	M,8-bit data	Move the immediate 8-bit data to memory location	MVI M,23H
LDA	16-bit address	Load the data from 16-bit address to ACC	LDA 2000H
STA	16-bit address	Store the data of ACC to 16-bit address	STA 2500H
LHLD	16-bit address	Directly loads at H & L registers	LHLD 2050
SHLD	16-bit address	directly stores from H & L registers	SHLD 2050
LXI	rp, 16-bit data	loads the specified register pair with data	LXI H, 3050
XCHG		exchanges H with D, and L with E	XCHG
PUSH	rp	pushes rp to the stack	PUSH H
POP	rp	pops the stack to rp	POP H
IN	8-bit port address	inputs contents of the specified port to A	IN 01
OUT	8-bit port address	outputs contents of A to the specified port	OUT 02

Arithmetic group

Arithmetic Instructions are the instructions which perform basic arithmetic operations such as addition, subtraction and a few more. In 8085 Microprocessor, the destination operand is generally the accumulator. In 8085 Microprocessor, the destination operand is generally the accumulator.

Opcode	Operand	Explanation	Example
ADD	R	A = A + R	ADD B
ADD	M	A = A + M	ADD M
ADI	8-bit data	A = A + 8-bit data	ADI 50
ADC	R	A = A + R + prev. carry	ADC B
ADC	M	A = A + Mc + prev. carry	ADC M
ACI	8-bit data	A = A + 8-bit data + prev. carry	ACI 50
SUB	R	A = A – R	SUB B
SUB	M	A = A – M	SUB M
SUI	8-bit data	A = A – 8-bit data	SUI 50
SBB	R	A = A – R – prev. carry	SBB B
SBB	M	A = A – M -prev. carry	SBB M
SBI	8-bit data	A = A – 8-bit data – prev. carry	SBI 50
INR	R	R = R + 1	INR B
INR	M	M = M + 1	INR M
INX	r.p.	r.p. = r.p. + 1	INX H
DCR	R	R = R – 1	DCR B
DCR	M	M = M – 1	DCR M
DCX	r.p.	r.p. = r.p. – 1	DCX H
DAD	r.p.	HL = HL + r.p.	DAD H

Logical group

Logical instructions are the instructions which perform basic logical operations such as AND, OR, etc. In 8085 Microprocessor, the destination operand is always the accumulator. Here logical operation works on a bitwise level.

Opcode	Operand	Explanation	Example
ANA	R	A = A AND R	ANA B
ANA	M	A = A AND M	ANA M
ANI	8-bit data	A = A AND 8-bit data	ANI 50
ORA	R	A = A OR R	ORA B
ORA	M	A = A OR M	ORA M
ORI	8-bit data	A = A OR 8-bit data	ORI 50
XRA	R	A = A XOR R	XRA B
XRA	M	A = A XOR M	XRA M
XRI	8-bit data	A = A XOR 8-bit data	XRI 50
CMA		A = 1’s compliment of A	CMA
CMP	R	Compares R with A and triggers the flag register	CMP B
CMP	M	Compares Mc with A and triggers the flag register	CMP M
CPI	8-bit data	Compares 8-bit data with A and triggers the flag register	CPI 50
RRC		Rotate accumulator right without carry	RRC
RLC		Rotate accumulator left without carry	RLC
RAR		Rotate accumulator right with carry	RAR
RAL		Rotate accumulator left with carry	RAR
CMC		Compliments the carry flag	CMC
STC		Sets the carry flag	STC

Branch group

Branching instructions refer to the act of switching execution to a different instruction sequence as a result of executing a branch instruction. The three types of branching instructions are:

1. Jump (unconditional and conditional)
2. Call (unconditional and conditional)
3. Return (unconditional and conditional)

1. Jump Instructions – The jump instruction transfers the program sequence to the memory address given in the operand based on the specified flag. Jump instructions are 2 types: Unconditional Jump Instructions and Conditional Jump Instructions.

(a) Unconditional Jump Instructions: Transfers the program sequence to the described memory address.

JMP 16-bit address Jumps to the address

Example- JMP 2050

(b) Conditional Jump Instructions: Transfers the program sequence to the described memory address only if the condition in satisfied.

Opcode	Operand	Explanation	Example
JC	Address	Jumps to the address if carry flag is 1	JC 2050
JNC	Address	Jumps to the address if carry flag is 0	JNC 2050
JZ	Address	Jumps to the address if zero flag is 1	JZ 2050
JNZ	Address	Jumps to the address if zero flag is 0	JNZ 2050
JPE	Address	Jumps to the address if parity flag is 1	JPE 2050
JPO	Address	Jumps to the address if parity flag is 0	JPO 2050
JM	Address	Jumps to the address if sign flag is 1	JM 2050

JP	Address	Jumps to the address if sign flag 0	JP 2050
JC	Address	Jumps to the address if carry flag is 1	JC 2050
JNC	Address	Jumps to the address if carry flag is 0	JNC 2050
JZ	Address	Jumps to the address if zero flag is 1	JZ 2050
JNZ	Address	Jumps to the address if zero flag is 0	JNZ 2050
JPE	Address	Jumps to the address if parity flag is 1	JPE 2050
JPO	Address	Jumps to the address if parity flag is 0	JPO 2050
JM	Address	Jumps to the address if sign flag is 1	JM 2050
JP	Address	Jumps to the address if sign flag 0	JP 2050

2. Call Instructions – The call instruction transfers the program sequence to the memory address given in the operand. Before transferring, the address of the next instruction after CALL is pushed onto the stack. Call instructions are 2 types: Unconditional Call Instructions and Conditional Call Instructions.

(a) Unconditional Call Instructions: It transfers the program sequence to the memory address given in the operand.
CALL 16-address Unconditionally calls
Example- CALL 2050
(b) Conditional Call Instructions: Only if the condition is satisfied, the instructions executes.

Opcode	Operand	Explanation	Example
CC	Address	Call if carry flag is 1	CC 2050
CNC	Address	Call if carry flag is 0	CNC 2050
CZ	Address	Calls if zero flag is 1	CZ 2050
CNZ	Address	Calls if zero flag is 0	CNZ 2050
CPE	Address	Calls if parity flag is 1	CPE 2050
CPO	Address	Calls if parity flag is 0	CPO 2050
CM	Address	Calls if sign flag is 1	CM 2050
CP	Address	Calls if sign flag is 0	CP 2050

3. Return Instructions – The return instruction transfers the program sequence from the subroutine to the calling program. Return instructions are 2 types: Unconditional Jump Instructions and Conditional Jump Instructions.

(a) Unconditional Return Instruction: The program sequence is transferred unconditionally from the subroutine to the calling program.
RET Return from the subroutine unconditionally
(b) Conditional Return Instruction: The program sequence is transferred unconditionally from the subroutine to the calling program only is the condition is satisfied.

Opcode	Operand	Explanation	Example
RC		Return from the subroutine if carry flag is 1	RC
RNC		Return from the subroutine if carry flag is 0	RNC
RZ		Return from the subroutine if zero flag is 1	RZ
RNZ		Return from the subroutine if zero flag is 0	RNZ
RPE		Return from the subroutine if parity flag is 1	RPE
RPO		Return from the subroutine if parity flag is 0	RPO
RM		Returns from the subroutine if sign flag is 1	RM
RP		Returns from the subroutine if sign flag is 0	RP

Machine control group

Opcode	Operand	Meaning	Explanation
NOP		No operation	No operation is performed, i.e., the instruction is fetched and decoded.
HLT		Halt and enter wait state	The CPU finishes executing the current instruction and stops further execution. An interrupt or reset is necessary to exit from the halt state.
DI		Disable interrupts	The interrupt enable flip-flop is reset and all the interrupts are disabled except TRAP.
EI		Enable interrupts	The interrupt enable flip-flop is set and all the interrupts are enabled.
RIM		Read interrupt mask	This instruction is used to read the status of interrupts 7.5, 6.5, 5.5 and read serial data input bit.
SIM		Set interrupt mask	This instruction is used to implement the interrupts 7.5, 6.5, 5.5, and serial data output.

Addressing modes

The term addressing modes refers to the way in which the operand of an instruction is specified. **Types of addressing modes –**

In 8085 microprocessor there are 5 types of addressing modes:

Immediate Addressing Mode –

In immediate addressing mode the source operand is always data. If the data is 8-bit, then the instruction will be of 2 bytes, if the data is of 16-bit then the instruction will be of 3 bytes.

Examples:

MVI B,45 (move the data 45H immediately to register B)

LXI H,3050 (load the H-L pair with the operand 3050H immediately)

JMP address (jump to the operand address immediately)

Register Addressing Mode –

In register addressing mode, the data to be operated is available inside the register(s) and register(s) is(are) operands. Therefore, the operation is performed within various registers of the microprocessor.

Examples:

MOV A, B (move the contents of register B to register A)

ADD B (add contents of registers A and B and store the result in register A) INR A (increment the contents of register A by one)

Direct Addressing Mode –

In direct addressing mode, the data to be operated is available inside a memory location and that memory location is directly specified as an operand. The operand is directly available in the instruction itself.

Examples:

LDA 2050 (load the contents of memory location into accumulator A)

LHLD address (load contents of 16-bit memory location into H-L register pair)

IN 35 (read the data from port whose address is 35)

Register Indirect Addressing Mode –

In register indirect addressing mode, the data to be operated is available inside a memory location and that memory location is indirectly specified by a register pair.

Examples:

MOV A, M (move the contents of the memory location pointed by the H-L pair to the accumulator)

LDAX B (move contents of B-C register to the accumulator)

LHLD 9570 (load immediate the H-L pair with the data of the location 9570)

Implied/Implicit Addressing Mode –

In implied/implicit addressing mode the operand is hidden and the data to be operated is available in the instruction itself.

Examples:

CMA (finds and stores the 1's complement of the contents of accumulator A in A)

RRC (rotate accumulator A right by one bit)

RLC (rotate accumulator A left by one bit)

UNIT-3 Timing Diagram

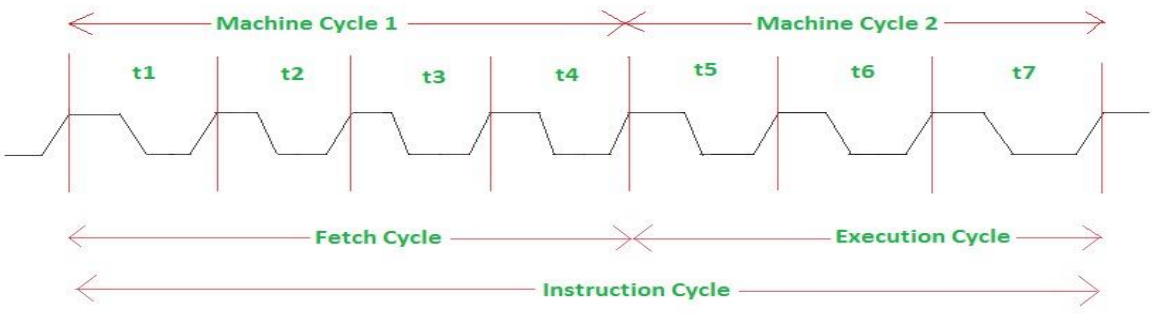
Instruction cycle of 8085 Microprocessor

Time required to execute and fetch an entire instruction is called instruction cycle. It consists: **Fetch cycle –** The next instruction is fetched by the address stored in program counter (PC) and then stored in the instruction register.

Decode instruction – Decoder interprets the encoded instruction from instruction register. **Execution cycle –** consists memory read (MR), memory write (MW), input output read (IOR) and input output write (IOW)

The time required by the microprocessor to complete an operation of accessing memory or input/output devices is called **machine cycle**. One time period of frequency of microprocessor is called **t-state**. A t-state is measured from the falling edge of one clock pulse to the falling edge of the next clock pulse.

Fetch cycle takes four t-states and execution cycle takes three t-states.

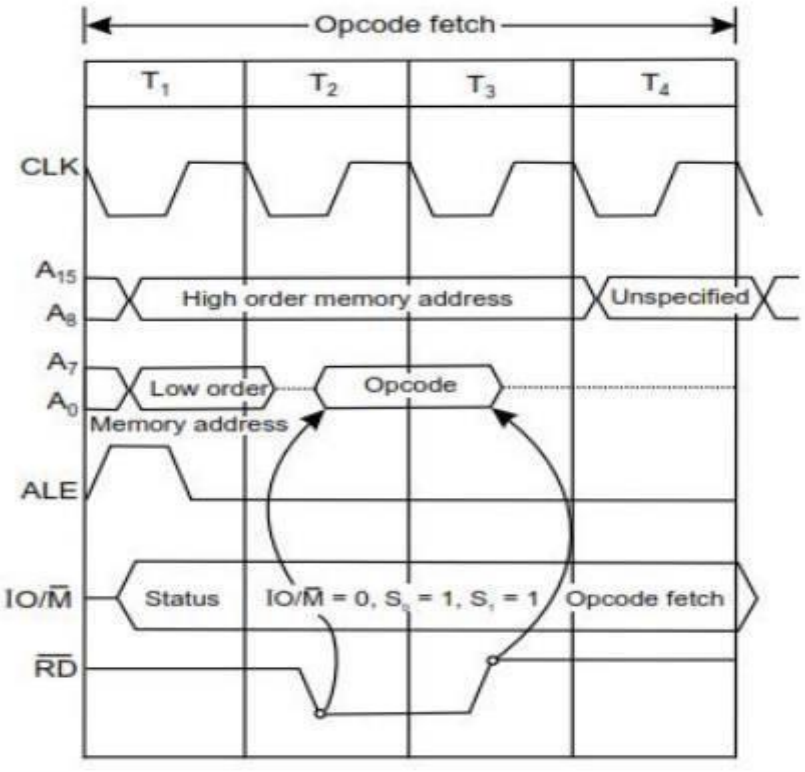


Instruction cycle in 8085 microprocessor

Timing diagram

Timing Diagram is a graphical representation. It represents the execution time taken by each instruction in a graphical format. The execution time is represented in T-states.

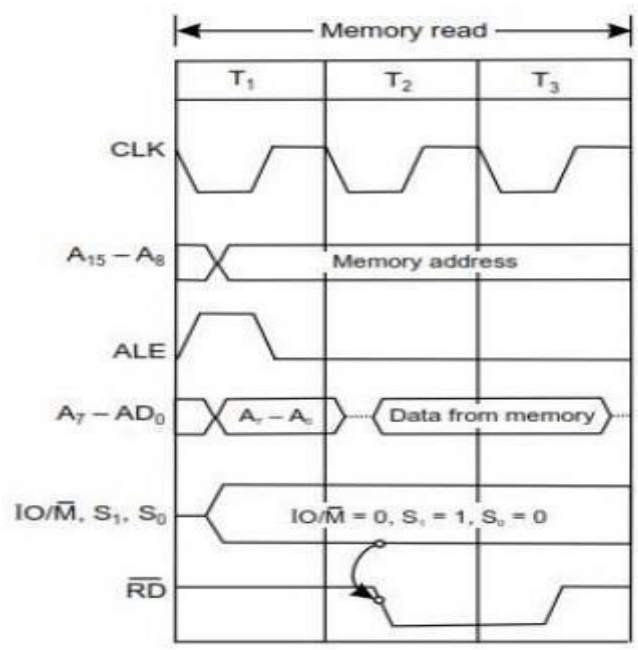
Opcode fetch cycle



- Each instruction of the processor has one byte opcode.
- The opcodes are stored in memory. So, the processor executes the opcode fetch machine cycle to fetch the opcode from memory.
- Hence, every instruction starts with opcode fetch machine cycle.
- The time taken by the processor to execute the opcode fetch cycle is 4T.
- In this time, the first, 3 T-states are used for fetching the opcode from memory and the remaining T-states are used for internal operations by the processor.

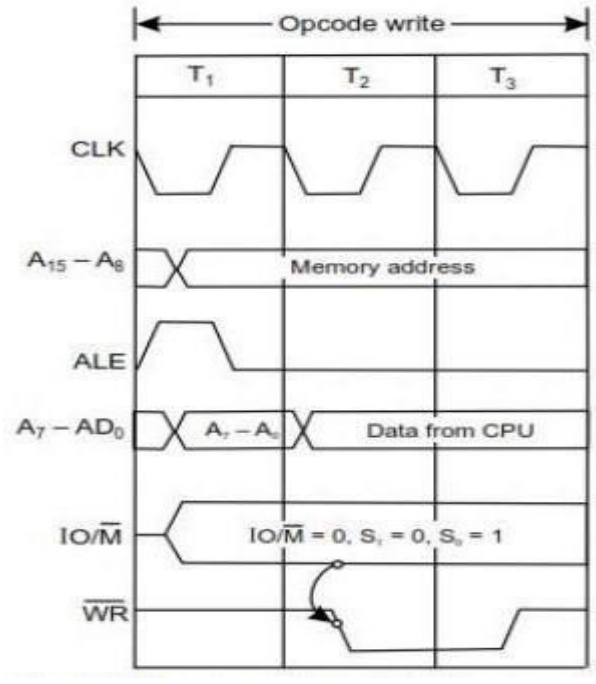
Memory read cycle

- The memory read machine cycle is executed by the processor to read a data byte from memory.
- The processor takes 3T states to execute this cycle.
- The instructions which have more than one byte word size will use the machine cycle after the opcode fetch machine cycle.



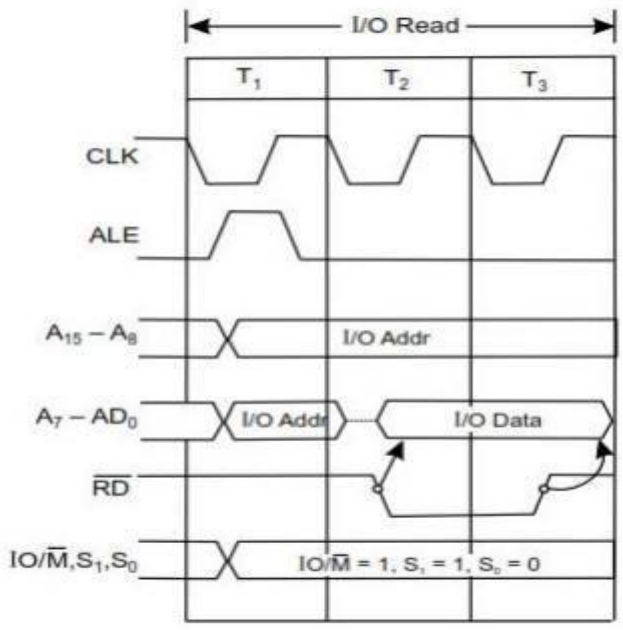
Memory write cycle

- The memory write machine cycle is executed by the processor to write a data byte in a memory location.
- The processor takes, 3T states to execute this machine cycle.



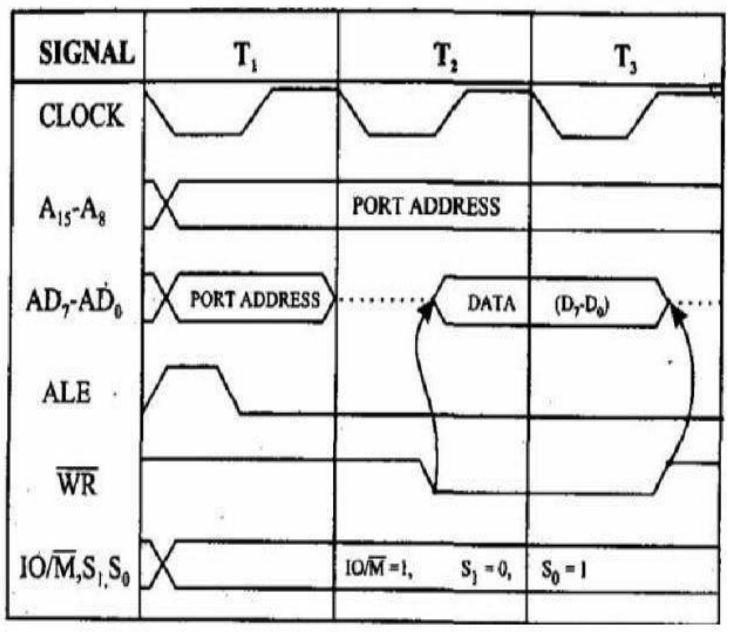
I/O read cycle

- The I/O Read cycle is executed by the processor to read a data byte from I/O port or from the peripheral, which is I/O, mapped in the system.
- The processor takes 3T states to execute this machine cycle.
- The IN instruction uses this machine cycle during the execution.



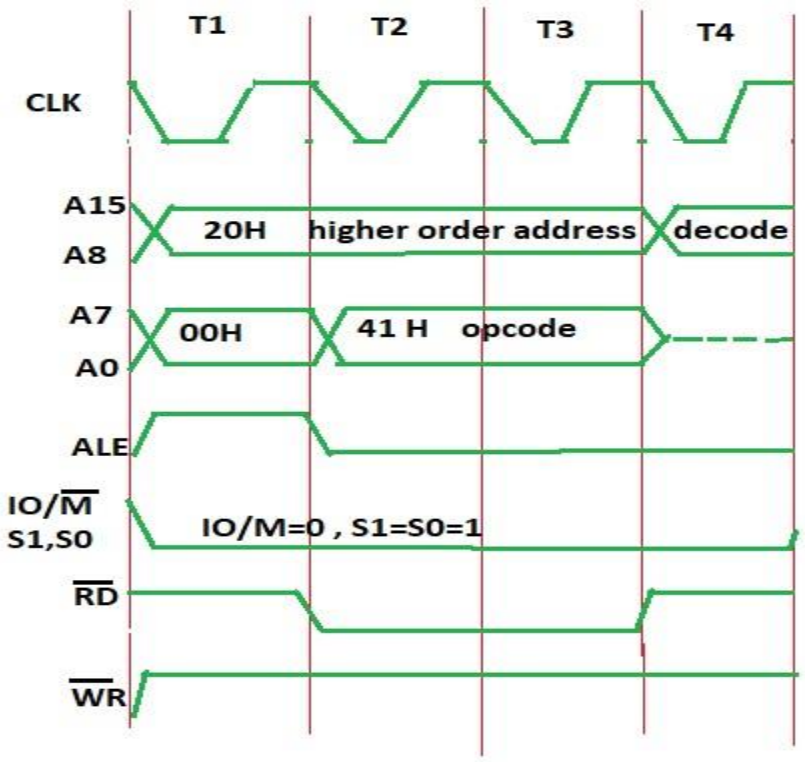
I/O write cycle

- The I/O Read cycle is executed by the processor to write a data byte from system to I/O port or peripheral, which is I/O mapped.
- The processor takes 3T states to execute this machine cycle.
- The OUT instruction uses this machine cycle during the execution.



Example-1

The instruction MOV B, C is of 1 byte; therefore, the complete instruction will be stored in a single memory address. **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.



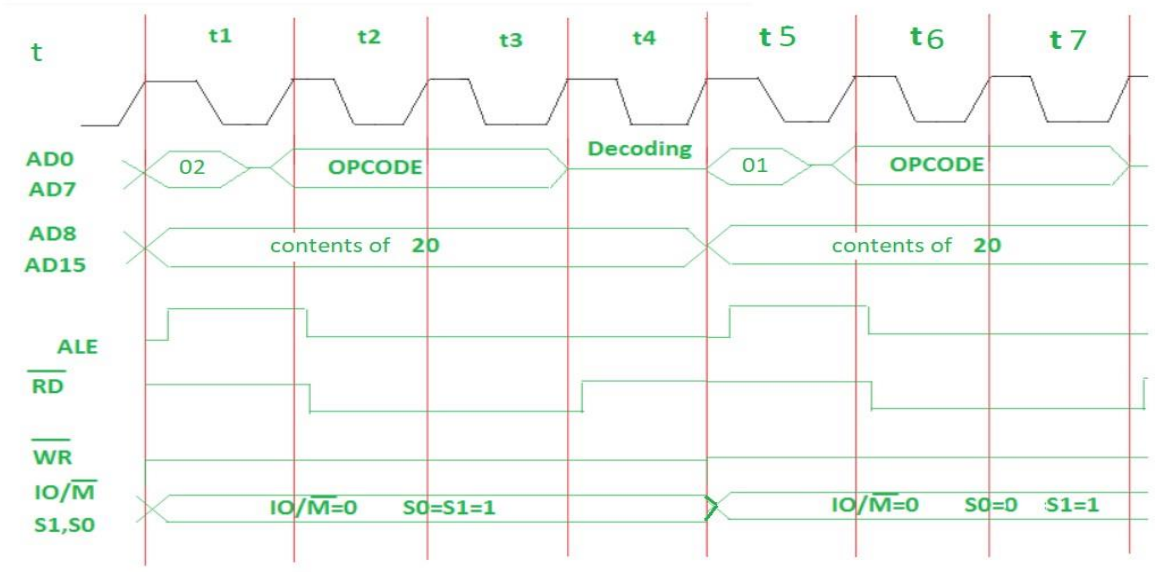
In Opcode fetch (t1-t4 T-states):

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.

Example-2

MVI B, 45
2000: Opcode
2001: 45

- The opcode fetch will be same in all the instructions.
- Only the read instruction of the opcode needs to be added in the successive T states.
- For the opcode fetch the IO/M (low active) = 0, S1 = 1 and S0 = 1. Also, 4 T states will be required to fetch the opcode from memory.
- For the opcode read the IO/M (low active) = 0, S1 = 1 and S0 = 0. Also, only 3 T states will be required to read data from memory.



In Opcode fetch (t1-t4 T-states) –

1. 00 – lower bit of address where opcode is stored.
2. 20 – higher bit of address where opcode is stored.
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, no data is read by microprocessor. Signal is 0 in t2 & t3, 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 0 in throughout, operation is performing on memory.
7. S0 and S1 – Signal is 1 in t1 to t4 states, as to fetch the opcode from the memory.

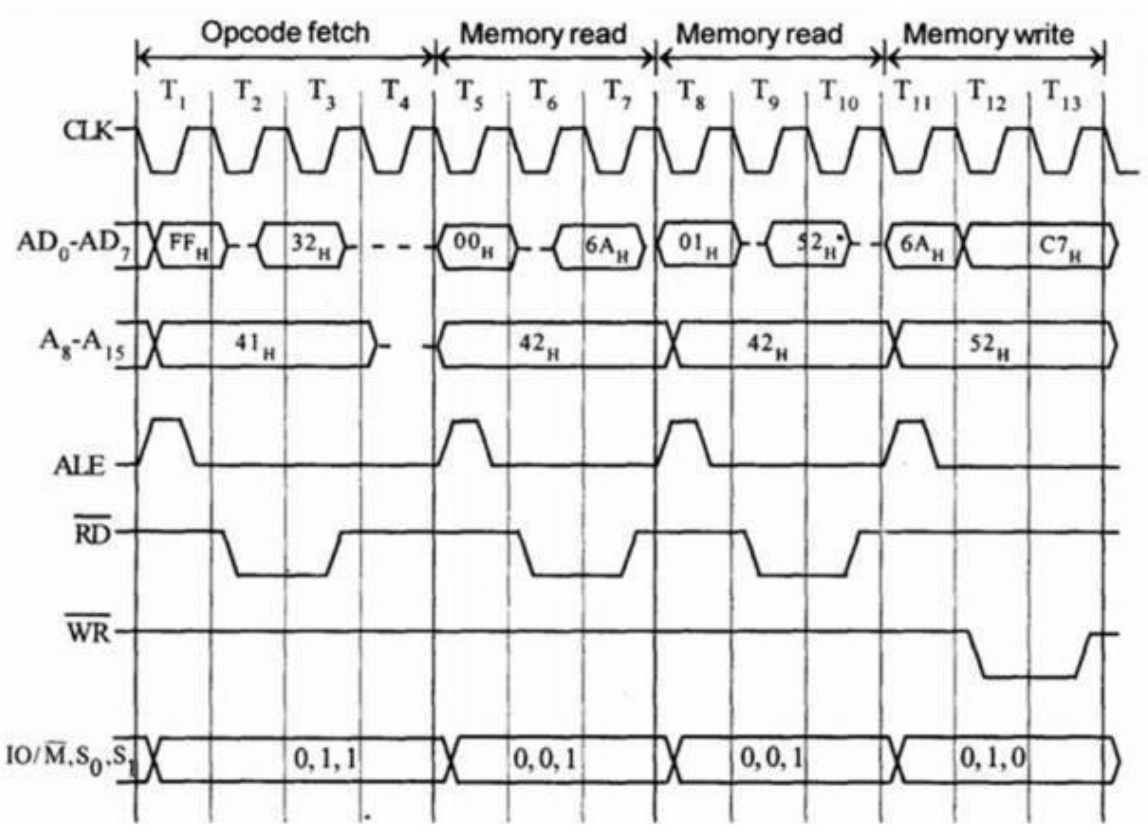
In Opcode read (t5-t7 T-states) –

1. 01 – lower bit of address where data is stored.
2. 320 – higher bit of address where data is stored.
3. ALE – Provides signal for multiplexed address and data bus. Only in t5 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 t5 as no data is read by microprocessor. Signal is 0 in t6 & t7 as 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 0 in throughout, operation is performing on memory.
7. S0 – Signal is 0 in throughout, operation is performing on memory to read data 45.
8. S1 – Signal is 1 throughout, operation is performing on memory to read data 45.

Example-3

41FF STA 526AH

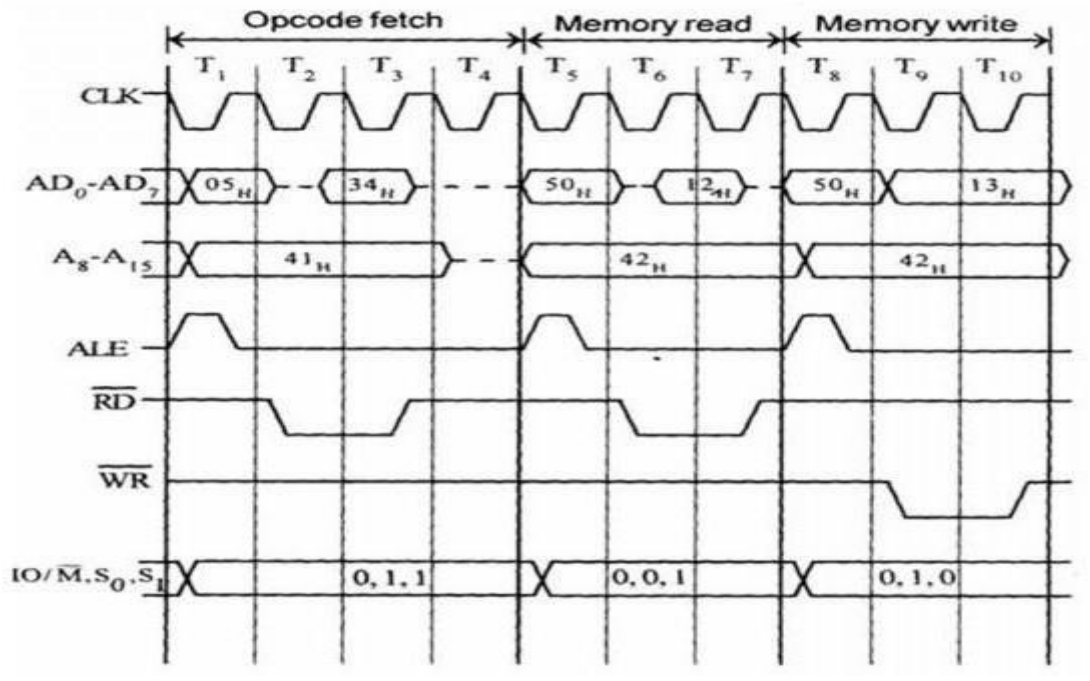


- STA means Store Accumulator -The contents of the accumulator is stored in the specified address (526A).
- The opcode of the STA instruction is said to be 32H. It is fetched from the memory 41FFH
- Then the lower order memory address is read (6A). - Memory Read Machine Cycle
- Read the higher order memory address (52).- Memory Read Machine Cycle
- The combination of both the addresses are considered and the content from accumulator is written in 526A. - Memory Write Machine Cycle
- Assume the memory address for the instruction and let the content of accumulator is C7H. So, C7H from accumulator is now stored in 526A.

Example-4

4105 INR M

- Fetching the Opcode 34H from the memory 4105H. (OF cycle)
- Let the memory address (M) be 4250H. (MR cycle -To read Memory address and data) Let the content of that memory is 12H.
- Increment the memory content from 12H to 13H. (MW machine cycle)



Counter and time delay

When the delay subroutine is executed, the microprocessor does not execute other tasks. For the delay we are using the instruction execution times. executing some instructions in a loop, the delay is generated. There are some methods of generating delays. These methods are as follows.

- Using NOP instructions
- Using 8-bit register as counter
- Using 16-bit register pair as counter.

Using NOT instructions:

- One of the main usage of NOP instruction is in delay generation.
- The NOP instruction is taking four clock pulses to be fetching, decoding and executing.
- In the 8085 MPU the internal clock frequency is 3MHz.
- So, from that we can easily determine that each clock period is 1/3 of a microsecond.
- So, the NOP will be executed in 1/3 * 4 = 1.333μs.

Using 8-bit register as counter:

- Counter is another approach to generate a time delay.
- In this case the program size is smaller.
- So, in this approach we can generate more time delay in less space.
- The following program will demonstrate the time delay using 8-bit counter. MVI B,FFH

LOOP: DCR B

JNZ LOOP RET

- Here the first instruction will be executed once, it will take 7 T-states.
- DCR C instruction takes 4 T-states.
- This will be executed 255 (FF) times.
- The JNZ instruction takes 10 T-states when it jumps (It jumps 254 times), otherwise it will take 7 T-States.
- And the RET instruction takes 10 T-States.
  7 + ((4*255) + (10*254)) + 7 + 10 = 3584.
- So, the time delay will be 3584 * 1/3μs = 1194.66μs.
- So, when we need some small delay, then we can use this technique with some other values in the place of FF.

This technique can also be done using some nested loops to get larger delays. The following code is showing how we can get some delay with one loop into some other loops. MVI B,FFH

L1: MVI C,FFH

L2: DCR C

JNZ L2

DCR B

JNZ L1

RET

From this block, if we calculate the delay, it will be nearly 305μs delay. It extends the time of delay.

Using 16-bit register-pair as counter:

- Instead of using 8-bit counter, we can do that kind of task using 16-bit register pair.
- Using this method more time delay can be generated.
- This method can be used to get more than 0.5 seconds delay.

Program	Time (T-States)

LXI B,FFFFH	10
LOOP: DCX B	6
MOV A,B	4
ORA C	4
JNZ LOOP	10 (For Jump), 7(Skip)
RET	10

From that table, if we calculate the time delay:

10 + (6 + 4 + 4 + 10) * 65535H – 3 + 10 = 17 + 24 * 65535H = 1572857.
So, the time delay will be 1572857 * 1/3μs = 0.52428s. Here we are getting nearly 0.5s delay.

Assembly language program

Example-1

Write an assembly language program to add two 8-bit numbers 45H and 32H in 8085 Microprocessor and store the result in 2050H. The starting address of the program is taken as 2000.

Program address	Mnemonics	Operands	Comments
2000	MVI	A,45	Load 1st data 45H in ACC
2002	MVI	B,32	Load 2nd data 32H in B
2004	ADD	B	A+B=A
2005	STA	2050	Store the result in 2050H
2008	HLT		Stop the program

O/P address **Result**

2050H 77H

Example-2

Write an ALP to add 2 8-bit numbers stored in memory location 2050H and 2051H. Result can be 8/16 bit and store it in 2052H and 2053H.

Program address	**Label**	**Mnemonics**	**Operands**	**Comments**
2000		MVI	C,00	Initialize the carry
2002		LXI	H,2050	Get the 1st data
2005		MOV	A,M	Load 1st data in ACC
2006		INX	H	Get 2nd data
2007		ADD	M	Add both data
2008		JNC	LOOP	If no carry, jump to LOOP
200B		INR	C	If carry, increment register C
200C	LOOP	STA	2052	Store the sum in 2052
2010		MOV	A,C	Move carry to ACC
2011		STA	2053	Store carry in 5053
2014		HLT		Stop the program

Without carry

I/P address **Data**

2050 53

2051 27

O/P address **Result**

2052 7A

2053 00

With carry

I/P address **Data**

2050 D9

2051 62

O/P address **Result**

2053 3B

2053 01

UNIT-4 Microprocessor based system development aids

Interface is the path for communication between two components. Interfacing is of two types, memory interfacing and I/O interfacing.

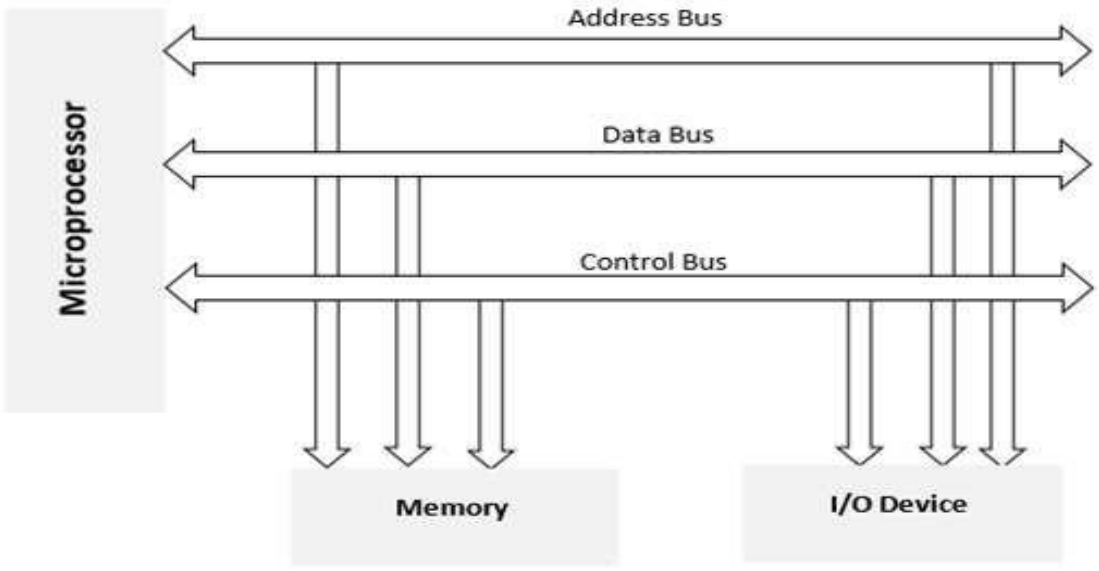
Memory interfacing

- Memory interfacing is used to provide more memory space to accommodate complex programs for more complicated systems.
- Types of memories which are most commonly used to interface with 8085 are RAM, ROM, and EEPROM.
- 8085 can access 64kB of external memory.
- It can be explained as- total number of address lines in 8085 are 16, therefore it can access 2^16 = 65535 locations i.e., 64kB

I/O interfacing

- There are various communication devices like the keyboard, mouse, printer, etc.
- So, we need to interface the keyboard and other devices with the microprocessor by using latches and buffers.
- This type of interfacing is known as I/O interfacing.

Block diagram of memory and I/O interfacing



Memory mapped I/O and I/O mapped I/O

In Memory Mapped Input Output –

- We allocate a memory address to an Input-Output device.
- Any instructions related to memory can be accessed by this Input-Output device.
- The Input-Output device data are also given to the Arithmetic Logical Unit.

Input-Output Mapped Input Output –

- We give an Input-Output address to an Input-Output device Only IN and OUT instructions are accessed by such devices.
- The ALU operations are not directly applicable to such Input-Output data.

So as a summary we can mention that –

- I/O is any general-purpose port used by processor/controller to handle peripherals connected to it.
- I/O mapped I/Os have a separate address space from the memory. So, total addressed capacity is the number of I/Os connected and a memory connected. Separate I/O-related instructions are used to access I/Os. A separate signal is used for addressing an I/O device.
- Memory-mapped I/Os share the memory space with external memory. So, total addressed capacity is memory connected only. This is underutilisation of resources if your processor supports I/O-mapped I/O. In this case, instructions used to access I/Os are the same as that used for memory.
- Let's take an example of the 8085 processor. It has 16 address lines i.e., addressing capacity of 64 KB memory. It supports I/O-mapped I/Os. It can address up to 256 I/Os.
- If we connect I/Os to it an I/O-mapped I/O then, it can address 256 I/Os + 64 KB memory. And special instructions IN and OUT are used to access the peripherals. Here we fully utilize the addressing capacity of the processor.
- If the peripherals are connected in memory mapped fashion, then total devices it can address is only 64K. This is underutilisation of the resource. And only memory-accessing instructions like MVI, MOV, LOAD, SAVE are used to access the I/O devices.

8255 Programable Peripheral Interface (PPI)

- PPI 8255 is a general purpose programmable I/O device designed to interface the CPU with its outside world such as ADC, DAC, keyboard etc.
- We can program it according to the given condition. It can be used with almost any microprocessor.
- It consists of three 8-bit bidirectional I/O ports (24 I/O lines) which can be configured as per the requirement.

Ports of 8255A

8255A has three ports, i.e., PORT A, PORT B, and PORT C.

- Port A (PA0-PA7) contains one 8-bit output latch/buffer and one 8-bit input buffer.
- Port B (PB0-PB7) is similar to PORT A.
- Port C can be split into two parts, i.e., PORT C lower (PC0-PC3) and PORT C upper (PC7PC4) by the control word.

These three ports are further divided into two groups, i.e., Group A includes PORT A and upper PORT C. Group B includes PORT B and lower PORT C. These two groups can be programmed in three different modes, i.e., the first mode is named as mode 0, the second mode is named as Mode 1 and the third mode is named as Mode 2.

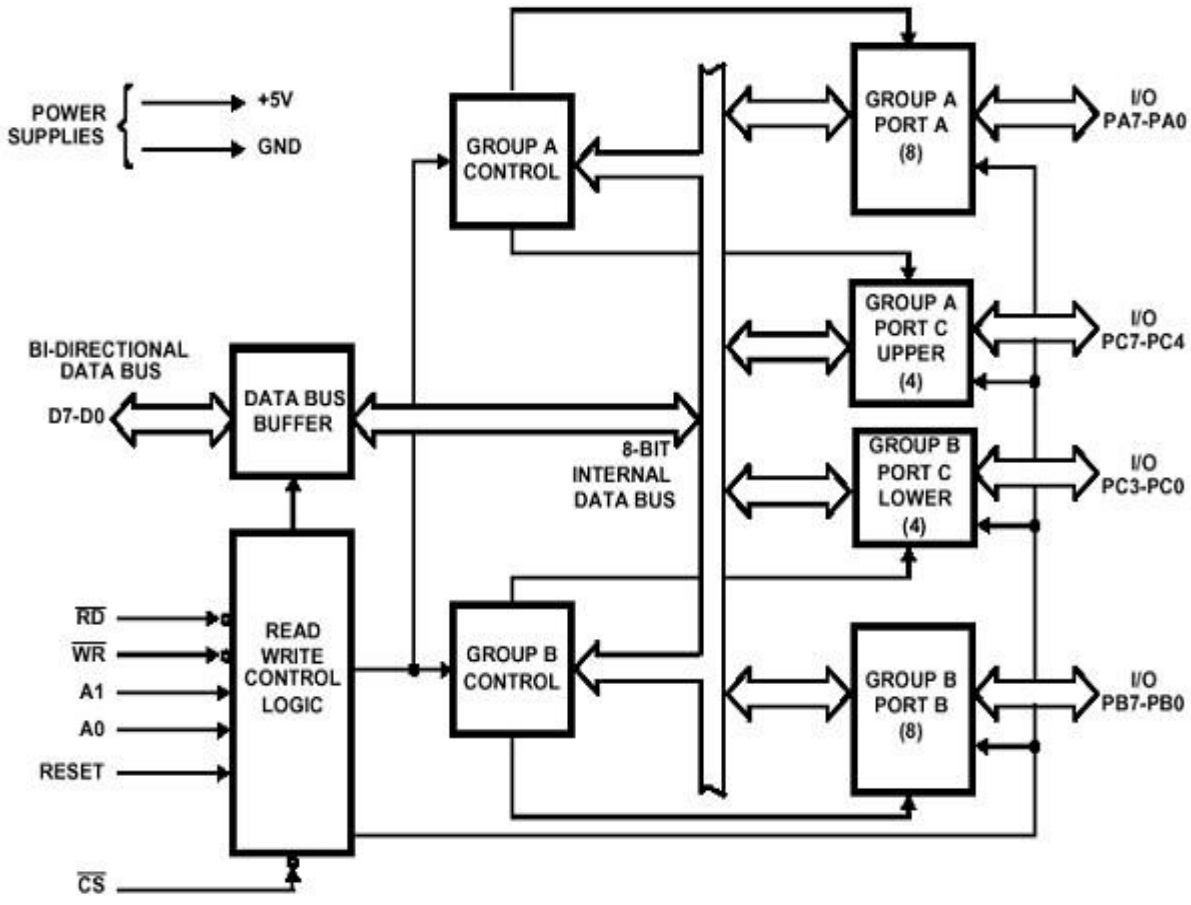
Features of 8255A

The prominent features of 8255A are as follows –

- It consists of 3 8-bit I/O ports i.e., PA, PB, and PC.

- Address/data bus must be externally demultiplexed.
- It is TTL compatible.
- It has improved DC driving capability.

8255 Architecture



Control group A

Control group A consist of port A and port C upper.

Control group B

Control group B consists of port C lower and port B.

Data Bus Buffer

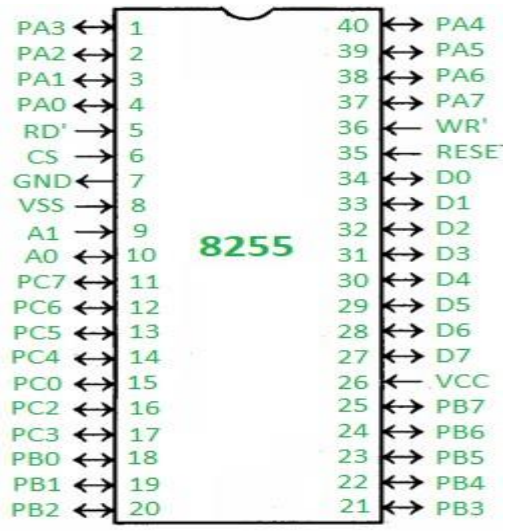
- It is a tri-state 8-bit buffer, which is used to interface the microprocessor to the system data bus.
- Data is transmitted or received by the buffer as per the instructions by the CPU.
- Control words and status information is also transferred using this bus.

Read/Write Control Logic

- This block is responsible for controlling the internal/external transfer of data/control/status word.
- It accepts the input from the CPU address and control buses, and in turn issues command to both the control groups.
- Depending upon the value if CS', A1 and A0 we can select different ports in different modes as input-output function or BSR.
- This is done by writing a suitable word in control register (control word D0-D7).

CS'	A1	A0	Selection
0	0	0	PORT A
0	0	1	PORT B
0	1	0	PORT C
0	1	1	Control Register
1	X	X	No Selection

Pin diagram



CS

It stands for Chip Select. A LOW on this input selects the chip and enables the communication between the 8255A and the CPU. It is connected to the decoded address, and A0 & A1 are connected to the microprocessor address lines.

WR

It stands for write. This control signal enables the write operation. When this signal goes low, the microprocessor writes into a selected I/O port or control register.

RESET

This is an active high signal. It clears the control register and sets all ports in the input mode

RD

It stands for Read. This control signal enables the Read operation. When the signal is low, the microprocessor reads the data from the selected I/O port of the 8255.

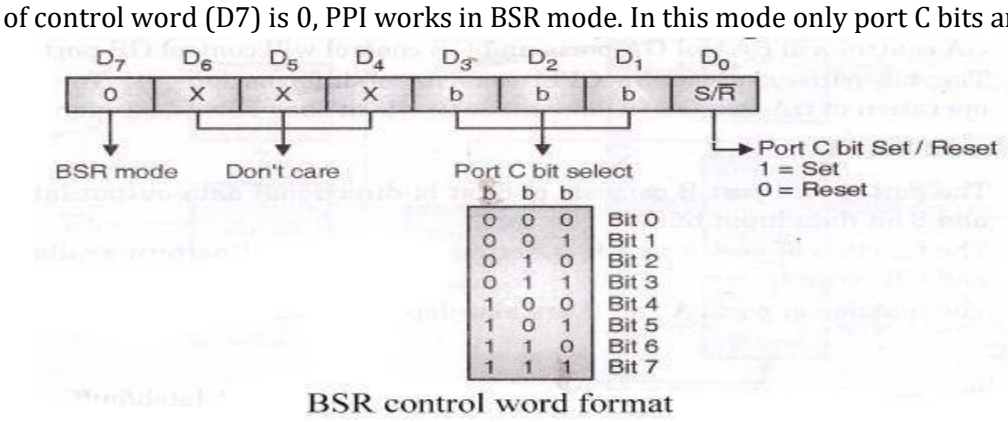
A0 and A1

These input signals work with RD, WR, and one of the control signals. Following is the table showing their various signals with their result.

A_1	A_0	RD	WR	CS	Result
0	0	0	1	0	**Input Operation** PORT A → Data Bus
0	1	0	1	0	PORT B → Data Bus
1	0	0	1	0	PORT C → Data Bus
0	0	1	0	0	**Output Operation** Data Bus → PORT A
0	1	1	0	0	Data Bus → PORT A
1	0	1	0	0	Data Bus → PORT B

Operating Modes 1. BSR (bit set-reset) mode-

If MSB of control word (D7) is 0, PPI works in BSR mode. In this mode only port C bits are used for set or reset.



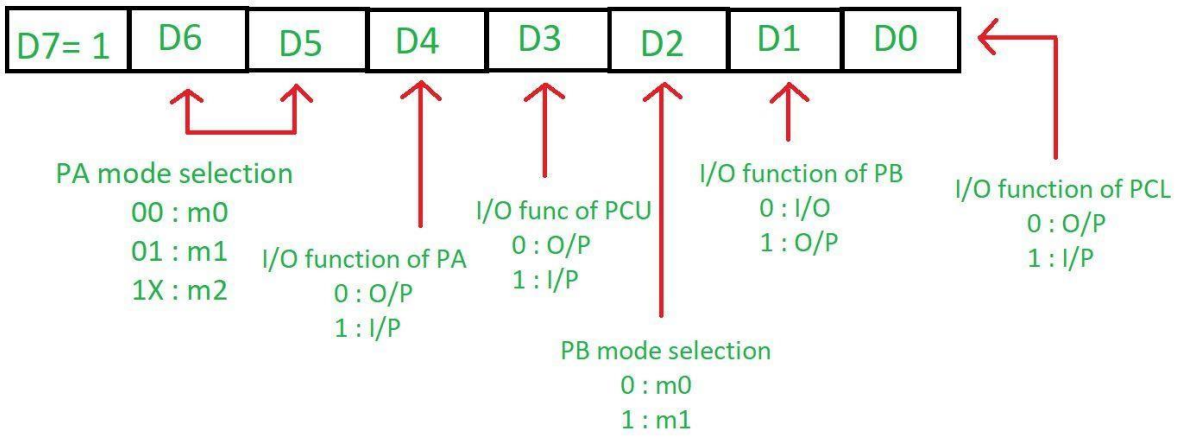
BSR control word format

2. I/O mode-

Mode 0 – In this mode, Port A and B is used as two 8-bit ports and Port C as two 4-bit ports. Each port can be programmed in either input mode or output mode where outputs are latched and inputs are not latched. Ports do not have interrupt capability.

Mode 1 – In this mode, Port A and B is used as 8-bit I/O ports. They can be configured as either input or output ports. Each port uses three lines from port C as handshake signals. Inputs and outputs are latched.

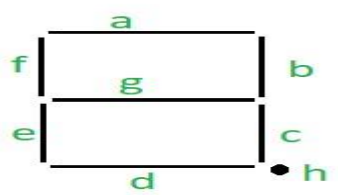
Mode 2 – In this mode, Port A can be configured as the bidirectional port and Port B either in Mode 0 or Mode 1. Port A uses five signals from Port C as handshake signals for data transfer. The remaining three signals from Port C can be used either as simple I/O or as handshake for port B.



Seven segment LED display

A seven-segment LED is a kind of LED (Light Emitting Diode) consisting of 7 small LEDs it usually comes with the microprocessor's as we commonly need to interface them with microprocessors like 8085.

Structure of Seven Segments LED:



- It can be used to represent numbers from 0 to 8 with a decimal point.
- We have eight segments in a Seven Segment LED display consisting of 7 segments which include '.'.
- The seven segments are denoted as "a, b, c, d, e, f, g, h" respectively, and '.' is represented by "h".

Interfacing Seven Segment Display with 8085:

We will see a program to Interfacing Seven Segment Display with 8085 using 8255.

Note logic needed for activation –

Common Anode – 0 will make an LED glow.

Common Cathode – 1 will make an LED glow.

Common Anode Method:

Here we are using a common anode display therefore 0 logic is needed to activate the segment. Suppose to display number 9 at the seven-segment display, therefore the segments F, G, B, A, C, and D have to be activated.

The instructions to execute it is given as,

```
MVI A,99        OUT 00
```

- First, we are storing the 99H in the accumulator i.e., 10010000 by using MVI instruction.
- By OUT instruction we are sending the data stored in the accumulator to the port 00H.

Common Cathode Method:

Here we are using common cathode 1 logic is needed to activate the signal. Suppose to display number 9 at the seven-segment display, therefore the segments F, G, B, A, C, and D have to be activated.

The instructions to execute it is given as,

```
MVI A,6F
OUT 00
```

- First, we are storing the 6FH in the accumulator i.e., 01101111 by using MVI instruction.
- By OUT instruction we are sending the data stored in the accumulator to the port 00H.

Traffic light controller

The traffic lights are interfaced to Microprocessor system through buffer and ports of programmable peripheral Interface 8255. So the traffic lights can be automatically switched ON/OFF in desired sequence. The Interface board has been designed to work with parallel port of Microprocessor system.

Working Program

Design of a microprocessor system to control traffic lights. The traffic should be controlled in the following manner.

1) Allow traffic from W to E and E to W transition for 20 seconds.
2) Give transition period of 5 seconds (Yellow bulbs ON)
3) Allow traffic from N to 5 and 5 to N for 20 seconds
4) Give transition period of 5 seconds (Yellow bulbs ON) 5) Repeat the process.

Source Program:

```
        MVI A, 80H:         Initialize 8255, port A and port B
        OUT 83H (CR):       in output mode
START: MVI A, 09H
        OUT 80H (PA):       Send data on PA to glow R1 and R2
        MVI A, 24H
        OUT 81H (PB):               Send data on PB to glow G3 and G4
        MVI C, 28H:         Load multiplier count (40₁₀) for delay
        CALL DELAY:         Call delay subroutine
        MVI A, 12H
```

```
        OUT (81H) PA:       Send data on Port A to glow Y1 and Y2
        OUT (81H) PB:       Send data on port B to glow Y3 and Y4
        MVI C, 0AH:         Load multiplier count (10₁₀) for delay
        CALL: DELAY:  MVI           Call delay subroutine
        A, 24H
        OUT     (80H)     PA:     Send data on port A to glow G1 and G2
        MVI A, 09H
        OUT (81H) PB:       Send data on port B to glow R3 and R4
        MVI C, 28H:         Load multiplier count (40₁₀) for delay
        CALL DELAY:         Call delay subroutine
        MVI A, 12H
        OUT PA:             Send data on port A to glow Y1 and Y2
        OUT PB:             Send data on port B to glow Y3 and Y4
        MVI C, 0AH:         Load multiplier count (10₁₀) for delay
        CALL DELAY:         Call delay subroutine
        JMP START
Delay Subroutine:
DELAY:  LXI D, Count:               Load count to give 0.5 sec delay
BACK:  DCX D:   MOV A, D            Decrement counter

        ORA E:                      Check whether count is 0
        JNZ BACK:           If not zero, repeat
        DCR C:              Check if multiplier zero, otherwise repeat
        JNZ DELAY
        RET:                Return to main program
```

Square wave generator

- With 00H as i/p to DAC, analog o/p is -5V, and with FFH as i/p, analog o/p is +5V.
- I/P 00H and FFH at regular intervals generate square wave. The frequency can be varied by varying the time delay.

Algorithm

Initialize the control word of 8255 to operate in I/O mode for port A and B & C to operate in o/p mode.

Program

```
        MVI A,80
        OUT CWR      initialize the control word
LOOP: MVI A,00
        OUT PA
        CALL DELAY
        MVI A,FF
        OUT PA
        CALL DELAY
        JMP LOOP
DELAY: MVI C,85
BACK: DCR C
        JNZ BACK
        RET
```

Basic concept of 8257 DMA Controller:

DMA stands for Direct Memory Access. It is designed by Intel to transfer data at the fastest rate. It allows the device to transfer the data directly to/from memory without any interference of the CPU.

Using a DMA controller, the device requests the CPU to hold its data, address and control bus, so the device is free to transfer data directly to/from the memory. The DMA data transfer is initiated only after receiving HLDA signal from the CPU.

How DMA Operations are Performed?

- Initially, when any device has to send data between the device and the memory, the device has to send DMA request (DRQ) to DMA controller.
- The DMA controller sends Hold request (HRQ) to the CPU and waits for the CPU to assert the HLDA.
- Then the microprocessor tri-states all the data bus, address bus, and control bus. The CPU leaves the control over bus and acknowledges the HOLD request through HLDA signal.
- Now the CPU is in HOLD state and the DMA controller has to manage the operations over buses between the CPU, memory, and I/O devices.

Features of 8257:

- It has four channels which can be used over four I/O devices.
- Each channel has 16-bit address and 14-bit counter.

- Each channel can transfer data up to 64kb.
- Each channel can be programmed independently.
- Each channel can perform read transfer, write transfer and verify transfer operations.
- It generates MARK signal to the peripheral device that 128 bytes have been transferred.
- It requires a single-phase clock.
- Its frequency ranges from 250Hz to 3MHz.
- It operates in 2 modes, i.e., Master mode and Slave mode.

8251 USART:

- The 8251 chip is Universal Synchronous Asynchronous Receiver Transmitter (USART). It acts as a mediator between the microprocessor and peripheral devices. It converts serial data to parallel form and vice versa.
- USART provides the computer with the interface necessary for communication with modems and other serial devices.
- USART offers the option of synchronous mode. In program-to-program communication, the synchronous mode requires that each end of an exchange respond in turn without initiating a new communication.
- Asynchronous operation means that a process operates independently of other processes.

Difference between synchronous mode and asynchronous mode:

- Synchronous mode requires both data and a clock. Asynchronous mode requires only data.
- In synchronous mode, the data is transmitted at a fixed rate. In asynchronous mode, the data does not have to be transmitted at a fixed rate.
- Synchronous data is normally transmitted in the form of blocks, while asynchronous data is normally transmitted one byte at a time.
- Synchronous mode allows for a higher DTR (data transfer rate) than asynchronous mode does, if all other factors are held constant.

UNIT-5
8086 Microprocessor

- 8086 Microprocessor is an enhanced version of 8085 Microprocessor 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

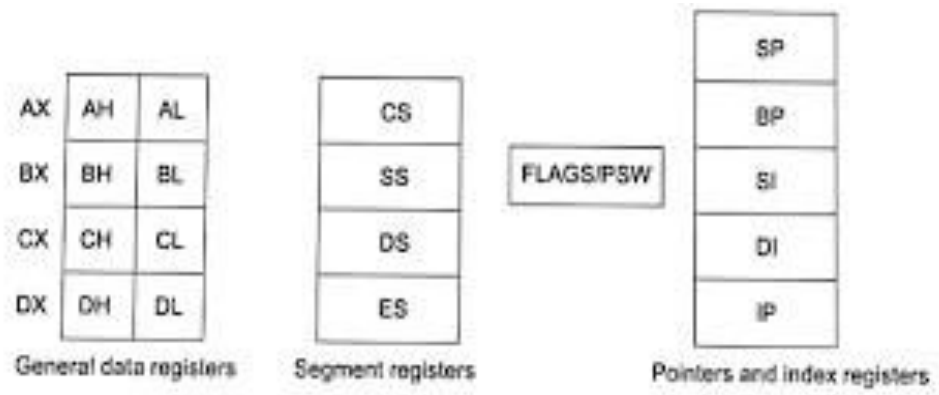
The most prominent features of a 8086 microprocessor are as follows –

- 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
  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.

Register organization of 8086 Microprocessor:



1. General 16-bit registers

The registers AX, BX, CX, and DX are the general 16-bit registers.

AX Register: Accumulator register 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. Accumulator can be used for I/O operations, rotate and string manipulation.

BX Register: This register is mainly used as a base register. It holds the starting base location of a memory region within a data segment. It is used as offset storage for forming physical address in case of certain addressing mode.

CX Register: It is used as default counter or count register in case of string and loop instructions.

DX Register: Data register can be used as a port number in I/O operations and implicit operand or destination in case of few instructions. In integer 32-bit multiply and divide instruction the DX register contains high-order word of the initial or resulting number.

2. Segment registers:

To complete 1Mbyte memory is divided into 16 logical segments. Each segment contains 64Kbyte of memory. There are four segment registers.

Code segment (CS) is a 16-bit register containing address of 64 KB segment with processor instructions. The processor uses CS segment for all accesses to instructions referenced by instruction pointer (IP) register. CS register cannot be changed directly. The CS register is automatically updated during far jump, far call and far return instructions. It is used for addressing a memory location in the code segment of the memory, where the executable program is stored.

Stack segment (SS) is a 16-bit register containing address of 64KB segment with program stack. By default, the processor assumes that all data referenced by the stack pointer (SP) and base pointer (BP) registers is located in the stack segment. SS register can be changed directly using POP instruction. It is used for addressing stack segment of memory. The stack segment is that segment of memory, which is used to store stack data.

Data segment (DS) is a 16-bit register containing address of 64KB segment with program data. By default, the processor assumes that all data referenced by general registers (AX, BX, CX, DX) and index register (SI, DI) is located in the data segment. DS register can be changed directly using POP and LDS instructions. It points to the data segment memory where the data is resided.

Extra segment (ES) is a 16-bit register containing address of 64KB segment, usually with program data. By default, the processor assumes that the DI register references the ES segment in string manipulation instructions. ES register can be changed directly using POP and LES instructions. It also refers to segment which essentially is another data segment of the memory. It also contains data.

3. Pointers and index registers.

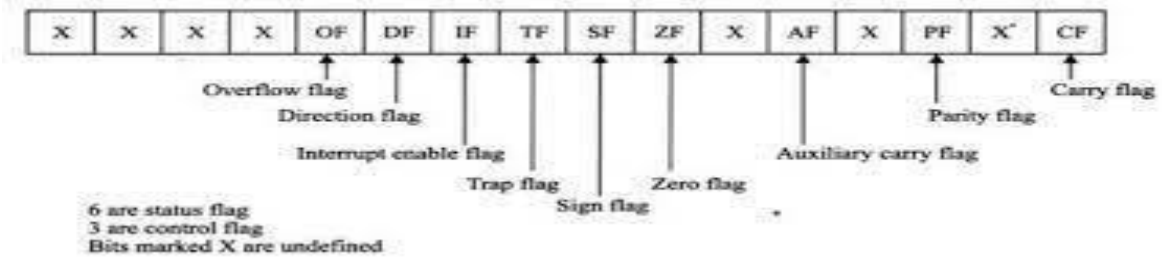
The pointers contain within the particular segments. The pointers IP, BP, SP usually contain offsets within the code, data and stack segments respectively

Stack Pointer (SP) is a 16-bit register pointing to program stack in stack segment.
Base Pointer (BP) is a 16-bit register pointing to data in stack segment. BP register is usually used for based, based indexed or register indirect addressing.

Source Index (SI) is a 16-bit register. SI is used for indexed, based indexed and register indirect addressing, as well as a source data addresses in string manipulation instructions.

Destination Index (DI) is a 16-bit register. DI is used for indexed, based indexed and register indirect addressing, as well as a destination data address in string manipulation instructions.

4. Flags: 8086 Microprocessor has 16-bit flag resister which is divided into two parts: Conditional flag (status flag) and Control flag.



Conditional Flags:
Carry Flag (CY): This flag indicates an overflow condition for unsigned integer arithmetic. It is also used in multiple-precision arithmetic.

Auxiliary Flag (AC): If an operation performed in ALU generates a carry/barrow from lower nibble (i.e., D0 – D3) to upper nibble (i.e., D4 – D7), the AC flag is set i.e., carry given by D3 bit to D4 is AC flag. This is not a general-purpose flag; it is used internally by the Processor to perform Binary to BCD conversion.

Parity Flag (PF): This flag is used to indicate the parity of result. If lower order 8-bits of the result contains even number of 1's, the Parity Flag is set and for odd number of 1's, the Parity flag is reset.

Zero Flag (ZF): It is set; if the result of arithmetic or logical operation is zero else it is reset.

Sign Flag (SF): In sign magnitude format the sign of number is indicated by MSB bit. If the result of operation is negative, sign flag is set.

Control Flags:
Control flags are set or reset deliberately to control the operations of the execution unit. Control flags are as follows:

Trap Flag (TF): It is used for single step control. It allows user to execute one instruction of a program at a time for debugging. When trap flag is set, program can be run in single step mode.

Interrupt Flag (IF): It is an interrupt enable/disable flag. If it is set, the maskable interrupt of 8086 is enabled and if it is reset, the interrupt is disabled. It can be set by executing instruction sit and can be cleared by executing CLI instruction.

Direction Flag (DF): It is used in string operation. If it is set, string bytes are accessed from higher memory address to lower memory address. When it is reset, the string bytes are accessed from lower memory address to higher memory address.

Internal architecture of 8086 Microprocessor:
The internal architecture of Intel 8086 is divided into 2 units: The Bus Interface Unit (BIU), and The Execution Unit (EU).

1. The Bus Interface Unit (BIU):
It provides the interface of 8086 to external memory and I/O devices via the System Bus. It performs various machine cycles such as memory read, I/O read etc. to transfer data between memory and I/O devices.
BIU performs the following functions-

- It generates the 20-bit physical address for memory access.
- It fetches instructions from the memory.
- It transfers data to and from the memory and I/O.
- Maintains the 6-byte prefetch instruction queue (supports pipelining).
- BIU mainly contains the 4 Segment registers, the Instruction Pointer, a prefetch queue and an Address Generation Circuit.

Address of the next instruction is calculated as CS x 10H + IP.
Example:
CS = 4321H IP = 1000H then CS x 10H = 43210H + offset = 44210H This is the address of the instruction.

Address Generation Circuit:
The BIU has a Physical Address Generation Circuit.
It generates the 20-bit physical address using Segment and Offset addresses using the formula:
Physical Address = Segment Address x 10H + Offset Address

6-Byte Pre-fetch Queue:
It is a 6-byte queue (FIFO). Fetching the next instruction (by BIU from CS) while executing the current instruction is called pipelining.

2. The Execution Unit (EU):
The main components of the EU are General purpose registers, the ALU, Special purpose registers, Instruction Register and Instruction Decoder and the Flag/Status Register.

- Fetches instructions from the Queue in BIU, decodes and executes arithmetic and logic operations using the ALU.
- Sends control signals for internal data transfer operations within the microprocessor.
- Sends request signals to the BIU to access the external module.
- It operates with respect to T-states (clock cycles) and not machine cycles.

Arithmetic Logic Unit (16 bit):
Performs 8 and 16 bit arithmetic and logic operations.

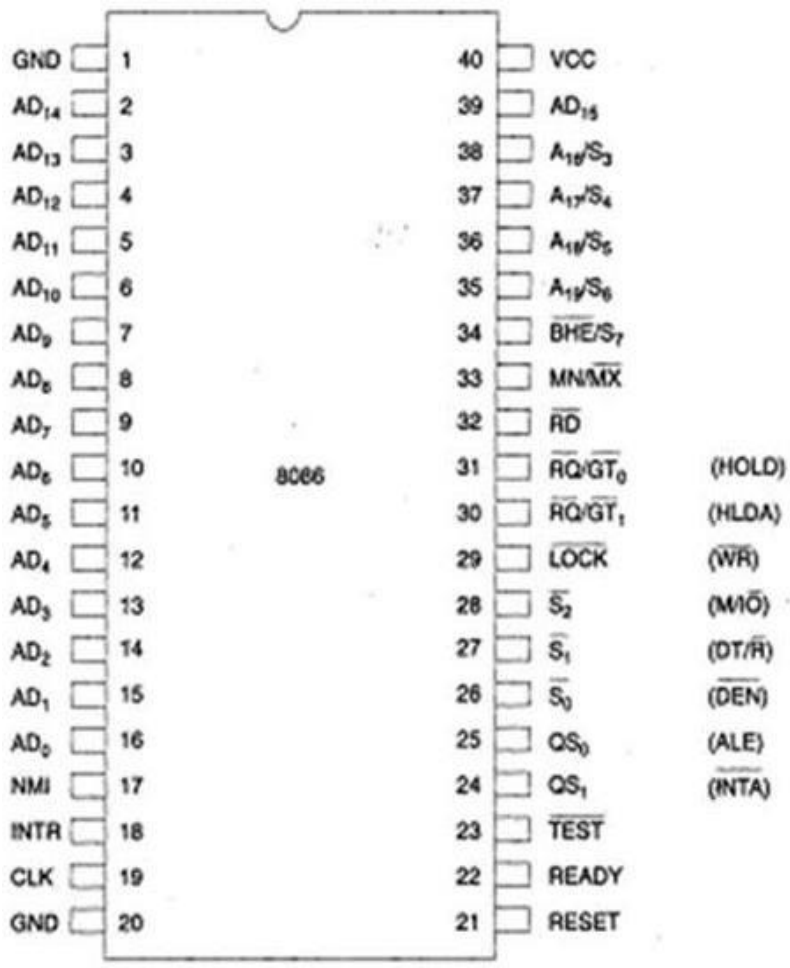
Instruction Register and Instruction Decoder:
The EU fetches an opcode from the queue into the instruction register. The instruction decoder decodes it and sends the information to the control circuit for execution.

Execution of whole 8086 Architecture:

- All instructions are stored in memory hence to fetch any instruction first task is to obtain the Physical address of the instruction is to be fetched. Hence this task is done by Bus Interface Unit (BIU) and by Segment Registers. Suppose the Code segment has a Segment address and the Instruction pointer has some offset address then the physical address calculator circuit calculates the physical address in which our instruction is to be fetched.
- After address calculation instruction is fetched from memory and it passes through C-Bus (Data bus) as shown in the figure, and according to the size of the instruction, the instruction pre-fetch queue fills up. For example MOV AX, BX is 1 Byte instruction so it will take only the 1st block of the queue, and MOV BX,4050H is 3 Byte instruction so it will take 3 blocks of the pre-fetch queue.
- When our instruction is ready for execution, according to the FIFO property of the queue instruction comes into the control system or control circuit which resides in the Execution unit. Here instruction decoding takes place. The decoding control system generates an opcode that tells the microprocessor unit which operation is to be performed. So the control system sends signals all over the microprocessor about what to perform and what to extract from General and special Purpose Registers.
- Hence After decoding microprocessor fetches data from GPR and according to instructions like ADD, SUB, MUL, and DIV data residing in GPRs are fetched and put as ALU's input. and after that addition, multiplication, division, or subtraction whichever calculation is to be carried out.
- According to arithmetic, flag register values change dynamically.
- While Instruction was decoding and executing from step-3 of our algorithm, the Bus interface Unit doesn't remain idle. it continuously fetches an instruction from memory and put it in a pre-fetch queue and gets ready for execution in a FIFO manner whenever the time arrives.
- So, in this way, unlike the 8085 microprocessor, here the Fetch, Decode, and Execution process happens parallelly not sequentially. This is called pipelining, and because of the instruction prefetch queue, all fetching, decoding, and execution process happen side-by-side. Hence there is partitioning in 8086 architecture like Bus Interface Unit and Execution Unit to support Pipelining phenomena.

8086 Pin Configuration:

8086 was the first 16-bit microprocessor available in 40-pin DIP (Dual Inline Package) chip.



The 8086 uses 20-line address bus.
It has a 16-line data bus.
The 20 lines of the address bus operate in multiplexed mode.
The 16-low order address bus lines have been multiplexed with data and 4 high-order address bus lines have been multiplexed with status signals.

AD0-AD15: Address/Data bus. These are low order address bus. They are multiplexed with data. When AD lines are used to transmit memory address the symbol A is used instead of AD, for example A0-A15. When data are transmitted over AD lines the symbol D is used in place of AD, for example D0-D7, D8-D15 or D0-D15.

A16-A19: High order address bus. These are multiplexed with status signals.

S2, S1, S0: Status pins. These pins are active during T4, T1 and T2 states and is returned to passive state (1,1,1 during T3 or Tw (when ready is inactive). These are used by the 8288 bus controller for generating all the memory and I/O operation) access control signals. Any change in S2, S1, S0 during T4 indicates the beginning of a bus cycle.

S2	S1	S0	Characteristics
0	0	0	Interrupt acknowledge
0	0	1	Read I/O port
0	1	0	Write I/O port
0	1	1	Halt
1	0	0	Code access
1	0	1	Read memory
1	1	0	Write memory
1	1	1	Passive

A16/S3, A17/S4, A18/S5, A19/S6: The specified address lines are multiplexed with corresponding status signals.

A17/S4	A16/S3	Function
0	0	Extra segment access
0	1	Stack segment access
1	0	Code segment access
1	1	Data segment access

BHE'/S7: Bus High Enable/Status. During T1 it is low. It is used to enable data onto the most significant half of data bus, D8-D15. 8-bit device connected to upper half of the data bus use BHE (Active Low) signal. It is multiplexed with status signal S7. S7 signal is available during T2, T3 and T4.

RD': This is used for read operation. It is an output signal. It is active when low.

READY: This is the acknowledgement from the memory or slow device that they have completed the data transfer. The signal made available by the devices is synchronized by the 8284A clock generator to provide ready input to the microprocessor. The signal is active high(1).

INTR: Interrupt Request. This is triggered input. This is sampled during the last clock cycles of each instruction for determining the availability of the request. If any interrupt request is found pending, the processor enters the interrupt acknowledge cycle. This can be internally masked after resulting the interrupt enable flag. This signal is active high(1) and has been synchronized internally.

NMI: Non maskable interrupt. This is an edge triggered input which results in a type II interrupt. A subroutine is then vectored through an interrupt vector lookup table which is located in the system memory. NMI is non-maskable internally by software. A transition made from low(0) to high(1) initiates the interrupt at the end of the current instruction. This input has been synchronized internally.

INTA: Interrupt acknowledge. It is active low(0) during T2, T3 and Tw of each interrupt acknowledge cycle.

MN/MX': Minimum/Maximum. This pin signal indicates what mode the processor will operate in.

RQ'/GT1′, RQ'/GT0′: Request/Grant. These pins are used by local bus masters used to force the microprocessor to release the local bus at the end of the microprocessor's current bus cycle. Each of the pin is bi-directional. RQ'/GT0′ have higher priority than RQ'/GT1′.

LOCK': Its an active low pin. It indicates that other system bus masters have not been allowed to gain control of the system bus while LOCK' is active low(0). The LOCK signal will be active until the completion of the next instruction.

TEST': This examined by a 'WAIT' instruction. If the TEST pin goes low(0), execution will continue, else the processor remains in an idle state. The input is internally synchronized during each of the clock cycle on leading edge of the clock.

CLK: Clock Input. The clock input provides the basic timing for processing operation and bus control activity. Its an asymmetric square wave with a 33% duty cycle.

RESET: This pin requires the microprocessor to terminate its present activity immediately. The signal must be active high (1) for at least four clock cycles.

Vcc: Power Supply (+5V D.C.)

GND: Ground

QS1, QS0: Queue Status. These signals indicate the status of the internal 8086 instruction queue according to the table shown below

QS1	QS0	Status
0	0	No operation

0	1	First byte of op code from queue
1	0	Empty the queue
1	1	Subsequent byte from queue

DT/R: Data Transmit/Receive. This pin is required in minimum systems, that want to use an 8286 or 8287 data bus transceiver. The direction of data flow is controlled through the transceiver.

DEN: Data enable. This pin is provided as an output enable for the 8286/8287 in a minimum system which uses transceiver. DEN is active low(0) during each memory and input-output access and for INTA cycles.

HOLD/HLDA: HOLD indicates that another master has been requesting a local bus. This is an active high (1). The microprocessor receiving the HOLD request will issue HLDA (high) as an acknowledgement in the middle of a T4 or T1 clock cycle.

ALE: Address Latch Enable. ALE is provided by the microprocessor to latch the address into the 8282 or 8283 address latch. It is an active high (1) pulse during T1 of any bus cycle. ALE signal is never floated, is always integer.

Memory Segmentation:
Segmentation is the process in which the main memory of the computer is logically divided into different segments and each segment has its own base address. It is basically used to enhance the speed of execution of the computer system, so that the processor is able to fetch and execute the data from the memory easily and fast.

Rules of Segmentation:
Segmentation process follows some rules as follows:

- The starting address of a segment should be such that it can be evenly divided by 16.
- Minimum size of a segment can be 16 bytes and the maximum can be 64 kB.

Segment	Offset Registers	Function
CS	IP	Address of the next instruction
DS	BX, DI, SI	Address of data
SS	SP, BP	Address in the stack
ES	BX, DI, SI	Address of destination data (for string operations)

Maximum mode configuration of 8086:

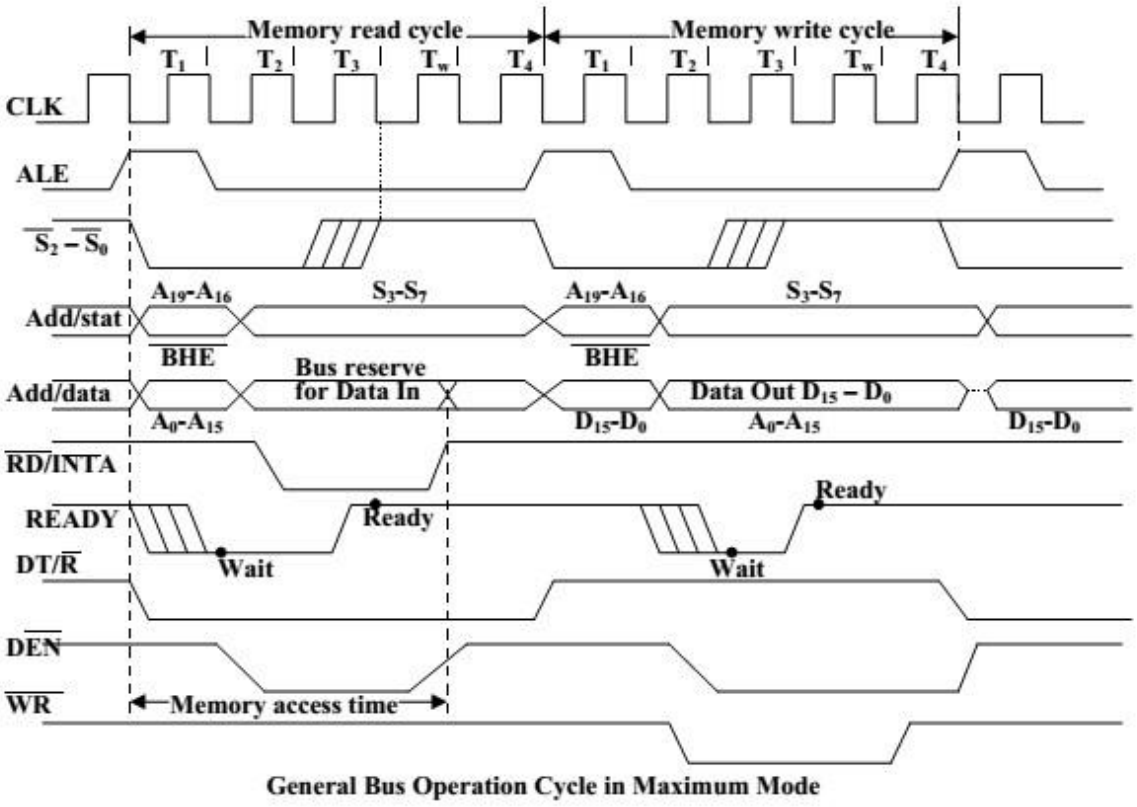
- In this we can connect more processors to 8086 (8087/8089).
- 8086 max mode is basically for implementation of allocation of global resources and passing bus control to other coprocessor (i.e., second processor in the system), because two processors cannot access system bus at same instant.
- All processors execute their own program.
- The resources which are common to all processors are known as global resources.
- The resources which are allocated to a particular processor are known as local or private resources.
- When MN/ MX' = 0 , 8086 works in max mode.
- Clock is provided by 8284 clock generator.
- 8288 bus controller- Address form the address bus is latched into 8282 8-bit latch. Three such latches are required because address bus is 20 bit. The ALE (Address latch enable) is connected to STB(Strobe) of the latch. The ALE for latch is given by 8288 bus controller.
- The data bus is operated through 8286 8-bit transceiver. Two such transceivers are required, because data bus is 16-bit. The transceivers are enabled the DEN signal, while the direction of data is controlled by the DT/R signal. DEN is connected to OE' and DT/ R' is connected to T. Both DEN and DT/ R' are given by 8288 bus controller.
- Bus request is done using RQ' / GT' lines interfaced with 8086. RQ0/GT0 has more priority than RQ1/GT1.
- INTA' is given by 8288, in response to an interrupt on INTR line of 8086.
- In max mode, the advanced write signals get enabled one T-state in advance as compared to normal write signals. This gives slower devices more time to get ready to accept the data, therefore it reduces the number of cycles.

Advantages of max mode of 8086:

- It helps to interface more devices like 8087.This interface is also called a closely coupled co-Processor configuration. In this 8086 is called as the host and 8087 as Co-processor.
- It supports multiprocessing; Therefore, it helps to increase the efficiency.
- The 8087 was the first floating-point coprocessor for the 8086 series of microprocessors. The purpose of the 8087 was to increase calculations for floating point operations, such as add, sub, multiply, divide, and square root.

Disadvantages of max mode over min mode:

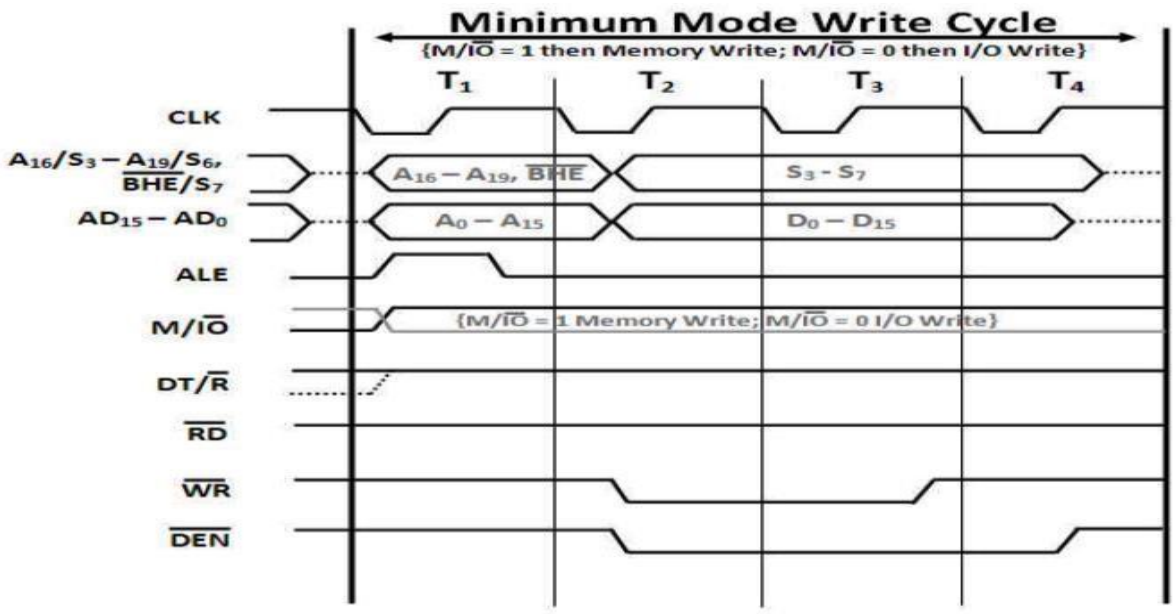
- It has more complex circuit than min mode.



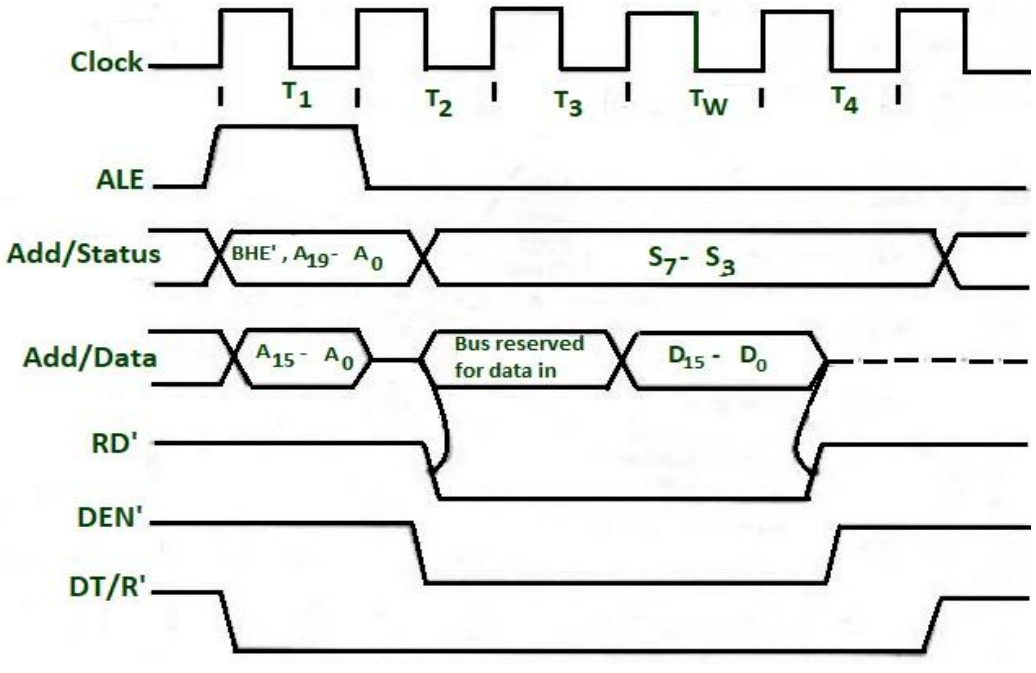
General Bus Operation Cycle in Maximum Mode

Minimum mode configuration of 8086:

- The 8086 microprocessor operates in minimum mode when MN/MX' = 1.
- In minimum mode,8086 is the only processor in the system which provides all the control signals which are needed for memory operations and I/O interfacing.
- Here the circuit is simple but it does not support multiprocessing.
- The other components which are transceivers, latches, 8284 clock generator, 74138 decoder, memory and i/o devices are also present in the system.
- 8282 is 8-bit latch used to separate the valid address from the multiplexed Address/data bus by using the control signal ALE, which is connected to strobe (STB) of 8282.
- 8286 is 8-bit transceivers. They are bidirectional buffers and also known as data amplifiers. They are used to separate the valid data from multiplexed add/data bus.
- 8284 clock generator is used to provide the clock.
- M/IO'= 1, then I/O transfer is performed over the bus. and when M/IO' = 0, then I/O operation is performed.
- The signals RD' and write WR' are used to identify whether a read bus cycle or a write bus cycle is performing. When WR' = 0, then it indicates that valid output data on the data bus.
- RD' indicates that the 8086 is performing a read data or instruction fetch process is occurring. During read operations, one other control signal is also used, which is DEN (data enable) and it indicates the external devices when they should put data on the bus.
- Control signals for all operations are generated by decoding M/IO', RD', WR'. They are decoded by 74138 3:8 decoder.



Memory read cycle:

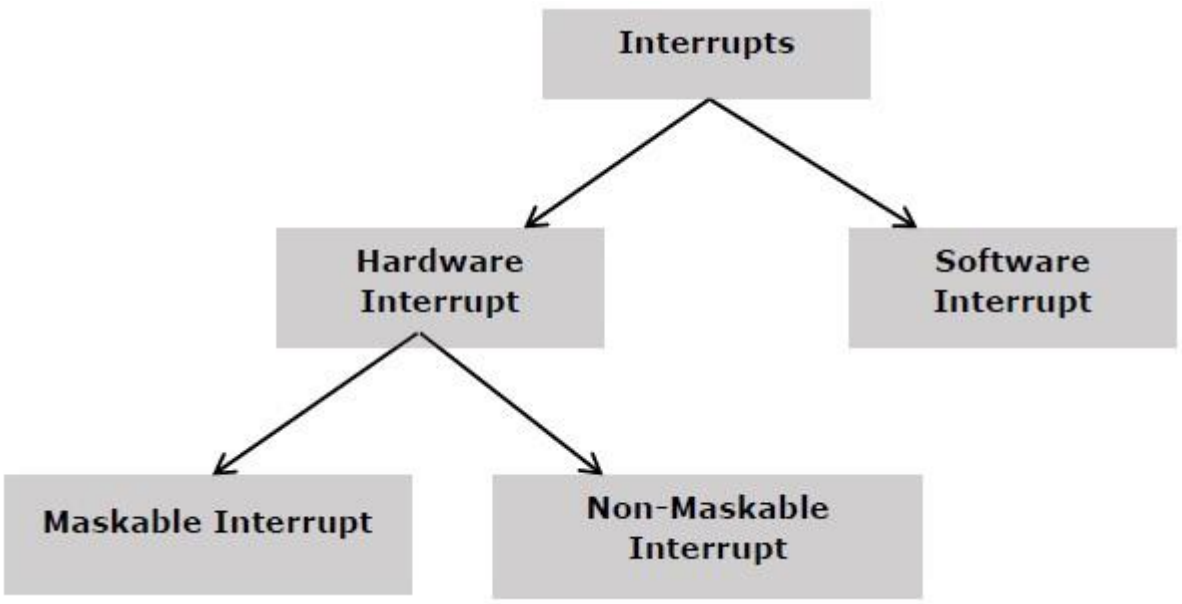


Interrupts in 8086:

- While the CPU is executing a program, an interrupt breaks the normal execution of instructions, diverts its execution to some other program called Interrupt Service Routine (ISR)
- Whenever an interrupt occurs the processor completes the execution of the current instruction and starts the execution of an Interrupt Service Routine (ISR) or Interrupt Handler.
- ISR is a program that tells the processor what to do when the interrupt occurs. At the end of the ISR the last instruction should be IRET.
- After the execution of ISR, control returns back to the main routine where it was interrupted.
- Whenever a number of devices interrupt a CPU at a time, and if the processor is able to handle them properly, it is said to have multiple interrupt processing capability.

Need for Interrupt: Interrupts are particularly useful when interfacing I/O devices that provide or require data at relatively low data transfer rate.

Types of interrupts:



Hardware Interrupts:

Hardware interrupt is caused by any peripheral device by sending a signal through a specified pin to the microprocessor.

The 8086 has two hardware interrupt pins, i.e., NMI and INTR. NMI is a nonmaskable interrupt and INTR is a maskable interrupt having lower priority. One more interrupt pin associated is INTA called interrupt acknowledge.

NMI:

It is a single non-maskable interrupt pin (NMI) having higher priority than the maskable interrupt request pin (INTR)and it is of type 2 interrupt.

When this interrupt is activated, these actions take place – □ Completes the current instruction that is in progress.

- Pushes the Flag register values on to the stack.
- Pushes the CS (code segment) value and IP (instruction pointer) value of the return address on to the stack.

- IP is loaded from the contents of the word location 00008H.
- CS is loaded from the contents of the next word location 0000AH. Interrupt flag and trap flag are reset to 0.

INTR:
The INTR is a maskable interrupt because the microprocessor will be interrupted only if interrupts are enabled using set interrupt flag instruction. It should not be enabled using clear interrupt Flag instruction.

Software Interrupts:
Some instructions are inserted at the desired position into the program to create interrupts. These interrupt instructions can be used to test the working of various interrupt handlers. It includes −
INT- Interrupt instruction with type number. It is 2-byte instruction. First byte provides the op-code and the second byte provides the interrupt type number. There are 256 interrupt types under this group.

Its execution includes the following steps − Flag register value is pushed on to the stack.

- CS value of the return address and IP value of the return address are pushed on to the stack.
- IP is loaded from the contents of the word location 'type number' × 4 CS is loaded from the contents of the next word location.
- Interrupt Flag and Trap Flag are reset to 0

The starting address for type 0 interrupt is 000000H, for type 1 interrupt is 00004H similarly for type2 is 00008H and ……so on. The first five pointers are dedicated interrupt pointers. i.e. −

- TYPE 0 interrupt represents division by zero situation.
- TYPE 1 interrupt represents single-step execution during the debugging of a program.
- TYPE 2 interrupt represents non-maskable NMI interrupt.
- TYPE 3 interrupt represents break-point interrupt.
- TYPE 4 interrupt represents overflow interrupt.
- The interrupts from Type 5 to Type 31 are reserved for other advanced microprocessors, and interrupts from 32 to Type 255 are available for hardware and software interrupts.

Instruction Set of 8086:
Instructions are classified on the basis of functions they perform. They are categorized into the following main types:
Data Transfer instruction:
All the instructions which perform data movement come under this category. The source data may be a register, memory location, port etc. the destination may be a register, memory location or port.

Instruction	Description
MOV	Moves data from register to register, register to memory, memory to register, memory to accumulator, accumulator to memory, etc.
LDS	Loads a word from the specified memory locations into specified register. It also loads a word from the next two memory locations into DS register.
LES	Loads a word from the specified memory locations into the specified register. It also loads a word from next two memory locations into ES register.
LEA	Loads offset address into the specified register.
LAHF	Loads low order 8-bits of the flag register into AH register.
SAHF	Stores the content of AH register into low order bits of the flags register.
XLAT/XLATB	Reads a byte from the lookup table.
XCHG	Exchanges the contents of the 16-bit or 8-bit specified register with the contents of AX register, specified register or memory locations.
PUSH	Pushes (sends, writes or moves) the content of a specified register or memory location(s) onto the top of the stack.
POP	Pops (reads) two bytes from the top of the stack and keeps them in a specified register, or memory location(s).
POPF	Pops (reads) two bytes from the top of the stack and keeps them in the flag register.
IN	Transfers data from a port to the accumulator or AX, DX or AL register.

OUT	Transfers data from accumulator or AL or AX register to an I/O port identified by the second byte of the instruction.

Arithmetic Instructions:
Instructions of this group perform addition, subtraction, multiplication, division, increment, decrement, comparison, ASCII and decimal adjustment etc.

Instruction	Description
ADD	Adds data to the accumulator i.e. AL or AX register or memory locations.
ADC	Adds specified operands and the carry status (i.e. carry of the previous stage).
SUB	Subtract immediate data from accumulator, memory or register.
SBB	Subtract immediate data with borrow from accumulator, memory or register.
MUL	Unsigned 8-bit or 16-bit multiplication.
IMUL	Signed 8-bit or 16-bit multiplication.
DIV	Unsigned 8-bit or 16-bit division.
IDIV	Signed 8-bit or 16-bit division.
INC	Increment Register or memory by 1.
DEC	Decrement register or memory by 1.
DAA	**Decimal Adjust after BCD Addition:** When two BCD numbers are added, the DAA is used after ADD or ADC instruction to get correct answer in BCD.
DAS	**Decimal Adjust after BCD Subtraction:** When two BCD numbers are added, the DAS is used after SUB or SBB instruction to get correct answer in BCD.
AAA	**ASCII Adjust for Addition:** When ASCII codes of two decimal digits are added, the AAA is used after addition to get correct answer in unpacked BCD.
AAD	**Adjust AX Register for Division:** It converts two unpacked BCD digits in AX to the equivalent binary number. This adjustment is done before dividing two unpacked BCD digits in AX by an unpacked BCD byte.
AAM	**Adjust result of BCD Multiplication:** This instruction is used after the multiplication of two unpacked BCD.
AAS	**ASCII Adjust for Subtraction:** This instruction is used to get the correct result in unpacked BCD after the subtraction of the ASCII code of a number from ASCII code another number.
CBW	Convert signed Byte to signed Word.
CWD	Convert signed Word to signed Doubleword.
NEG	Obtains 2's complement (i.e. negative) of the content of an 8-bit or 16-bit specified register or memory location(s).
CMP	Compare Immediate data, register or memory with accumulator, register or memory location(s).

Logical Instructions
Instruction of this group perform logical AND, OR, XOR, NOT and TEST operations.

Instruction	Description
AND	Performs bit by bit logical AND operation of two operands and places the result in the specified destination.
OR	Performs bit by bit logical OR operation of two operands and places the result in the specified destination.
XOR	Performs bit by bit logical XOR operation of two operands and places the result in the specified destination.
NOT	Takes one's complement of the content of a specified register or memory location(s).
TEST	Perform logical AND operation of a specified operand with another specified operand.

Rotate Instructions

Instruction	Description
RCL	Rotate all bits of the operand left by specified number of bits through carry flag.
RCR	Rotate all bits of the operand right by specified number of bits through carry flag.
ROL	Rotate all bits of the operand left by specified number of bits.
ROR	Rotate all bits of the operand right by specified number of bits.

Shift instructions

Instruction	Description
SAL or SHL	Shifts each bit of operand left by specified number of bits and put zero in LSB position.
SAR	Shift each bit of any operand right by specified number of bits. Copy old MSB into new MSB.
SHR	Shift each bit of operand right by specified number of bits and put zero in MSB position.

Branch Instructions:
It is also called program execution transfer instruction. Instructions of this group transfer program execution from the normal sequence of instructions to the specified destination or target.

Instruction	Description
JA or JNBE	Jump if above, not below, or equal i.e. when CF and ZF = 0
JAE/JNB/JNC	Jump if above, not below, equal or no carry i.e. when CF = 0
JB/JNAE/JC	Jump if below, not above, equal or carry i.e. when CF = 0
JBE/JNA	Jump if below, not above, or equal i.e. when CF and ZF = 1

JCXZ	Jump if CX register = 0
JE/JZ	Jump if zero or equal i.e. when ZF = 1
JG/JNLE	Jump if greater, not less or equal i.e. when ZF = 0 and CF = OF
JGE/JNL	Jump if greater, not less or equal i.e. when SF = OF
JL/JNGE	Jump if less, not greater than or equal i.e. when SF ≠ OF
JLE/JNG	Jump if less, equal or not greater i.e. when ZF = 1 and SF ≠ OF
JMP	Causes the program execution to jump unconditionally to the memory address or label given in the instruction.
CALL	Calls a procedure whose address is given in the instruction and saves their return address to the stack.
RET	Returns program execution from a procedure (subroutine) to the next instruction or main program.
IRET	Returns program execution from an interrupt service procedure (subroutine) to the main program.
INT	Used to generate software interrupt at the desired point in a program.
INTO	Software interrupts to indicate overflow after arithmetic operation.
LOOP	Jump to defined label until CX = 0.
LOOPZ/LOOPE	Decrement CX register and jump if CX ≠ 0 and ZF = 1.
LOOPNZ/LOOPNE	Decrement CX register and jump if CX ≠ 0 and ZF = 0.

Flag Manipulation and Processor Control Instructions:
Instructions of this instruction set are related to flag manipulation and machine control.

Instruction	Description
CLC	**Clear Carry Flag:** This instruction resets the carry flag CF to 0.
CLD	**Clear Direction Flag:** This instruction resets the direction flag DF to 0.
CLI	**Clear Interrupt Flag:** This instruction resets the interrupt flag IF to 0.
CMC	This instruction take complement of carry flag CF.
STC	Set carry flag CF to 1.
STD	Set direction flag to 1.
STI	Set interrupt flag IF to 1.
HLT	Halt processing. It stops program execution.
NOP	Performs no operation.
ESC	**Escape:** makes bus free for external master like a coprocessor or peripheral device.

WAIT	When WAIT instruction is executed, the processor enters an idle state in which the processor does no processing.
LOCK	It is a prefix instruction. It makes the LOCK pin low till the execution of the next instruction.

String Instructions:
String is series of bytes or series of words stored in sequential memory locations. The 8086 provides some instructions which handle string operations such as string movement, comparison, scan, load and store.

Instruction	**Description**
MOVS/MOVSB/MOVSW	Moves 8-bit or 16-bit data from the memory location(s) addressed by SI register to the memory location addressed by DI register.
CMPS/CMPSB/CMPSW	Compares the content of memory location addressed by DI register with the content of memory location addressed by SI register.
SCAS/SCASB/SCASW	Compares the content of accumulator with the content of memory location addressed by DI register in the extra segment ES.
LODS/LODSB/LODSW	Loads 8-bit or 16-bit data from memory location addressed by SI register into AL or AX register.
STOS/STOSB/STOSW	Stores 8-bit or 16-bit data from AL or AX register in the memory location addressed by DI register.
REP	Repeats the given instruction until CX ≠ 0
REPE/ REPZ	Repeats the given instruction till CX ≠ 0 and ZF = 1
REPNE/REPNZ	Repeats the given instruction till CX ≠ 0 and ZF = 0

Addressing Mode:
Addressing modes are different ways by which CPU can access data or operands. They determine how to access a specific memory address.

Register addressing mode:
This mode involves the use of registers. These registers hold the operands. This mode is very fast as compared to others because CPU doesn't need to access memory. CPU can directly perform an operation through registers.
MOV AX, BL
MOV AL, BL

Immediate Addressing Mode:
In this mode, there are two operands. One is a register and the other is a constant value.
For example:
The instruction MOV AX, 30H copies hexadecimal value 30H to register AX.
The instructions MOV BX, 255 copies decimal value 255 to register BX.
Immediate addressing mode is not used to load immediate value into segment registers. To move any value into segment registers, first load that value into a general-purpose register then add this value into segment register.

Direct Addressing Mode:
It loads or stores the data from memory to register and vice versa. The instruction consists of a register and an offset address. To compute physical address, shift left the DS register and add the offset address into it.
MOV CX, [481]

Register Indirect Addressing Mode:
The register indirect addressing mode uses the offset address which resides in one of these three registers i.e., BX, SI, DI. The sum of offset address and the DS value shifted by one position generates a physical address.
MOV AL, [SI]

Based Relative Addressing Mode:
This addressing mode uses a base register either BX or BP and a displacement value to calculate physical address.
MOV [BX+5], DX

Indexed Relative Addressing Mode:
This addressing mode is same as the based relative addressing mode. The only difference is it uses DI and SI registers instead of BX and BP registers.
MOV [DI]+12, AL
MOV BX, [SI]+10

Based Indexed Addressing Mode:
The based indexed addressing mode is actually a combination of based relative addressing mode and indexed relative addressing mode. It uses one base register (BX, BP) and one index register (SI, DI).
MOV AX, [BX+SI+20]
Or MOV AX, [SI][BX]+20

Assembly language program:
Addition

```
ORG 0000H
MOV DX, #07H     // move the value 7 to the register AX//
MOV AX, #09H     // move the value 9 to accumulator AX//
ADD AX, DX       // add AX value with DX value and stores the result in AX//
END
```

Multiplication

```
ORG 0000H
MOV DX, #04H     // move the value 4 to the register DX//
MOV AX, #08H     // move the value 8 to accumulator AX//
MUL AX, DX       // Multiplied result is stored in the Accumulator AX //
END
```

Subtraction

```
ORG 0000H
MOV DX, #02H     // move the value 2 to register DX//
MOV AX, #08H     // move the value 8 to accumulator AX//
SUBB AX, DX    // Result value is stored in the Accumulator A X//
END
```

Division

```
ORG 0000H
MOV DX, #08H     // move the value 3 to register DX//
MOV AX, #19H     // move the value 5 to accumulator AX//
DIV AX, DX     // final value is stored in the Accumulator AX //
END
```

Largest number:

```
          MOV SI, 500          //SI<-500//
          MOV CL, [SI]         //CL<-[SI]//
          MOV CH, 00           //CH<-00//
          INC SI               //SI<-SI+1//
          MOV AL, [SI]         //AL<-[SI]//
          DEC CL               //CL<-CL-1//
          INC SI               //SI<-SI+1//
BACK: CMP AL, [SI]             //AL-[SI]//
          JNC HEAD             //JUMP TO LOOP IF CY=0//
          MOV AL, [SI]         //AL<-[SI]//
HEAD: INC SI                   //SI<-SI+1//
          LOOP BACK            //CX<-CX-1 & JUMP TO BACK IF CX NOT 0//
          MOV [600], AL        //AL->[600]//
          HLT
```

Ascending order:

```
          MOV SI, 500          SI<-500
          MOV CL, [SI]         CL<-[SI]
          DEC CL               CL<-CL-1
LEAD: MOV SI, 500              SI<-500
          MOV CH, [SI]         CH<-[SI]
          DEC CH               CH<-CH-1
          INC SI               SI<-SI+1
```

```
BACK: MOV AL, [SI]      AL<-[SI]
      INC SI            SI<-SI+1
      CMP AL, [SI]      AL-[SI]
      JC HEAD           JUMP TO 41C IF CY=1
      XCHG AL, [SI]     SWAP AL AND [SI]
      DEC SI            SI<-SI-1
      XCHG AL, [SI]     SWAP AL AND [SI]
      INC SI            SI<-SI+1
HEAD: DEC CH            CH<-CH-1
      JNZ BACK          JUMP TO 40F IF ZF=0
      DEC CL            CL<-CL-1
      JNZ LEAD          JUMP TO 407 IF ZF=0
      HLT               END
```

8051 Microcontroller

In 1981, Intel introduced an 8-bit microcontroller called the 8051. It was referred as system on a chip because it had 128 bytes of RAM, 4K byte of on-chip ROM, two timers, one serial port, and 4 ports (8-bit wide), all on a single chip.

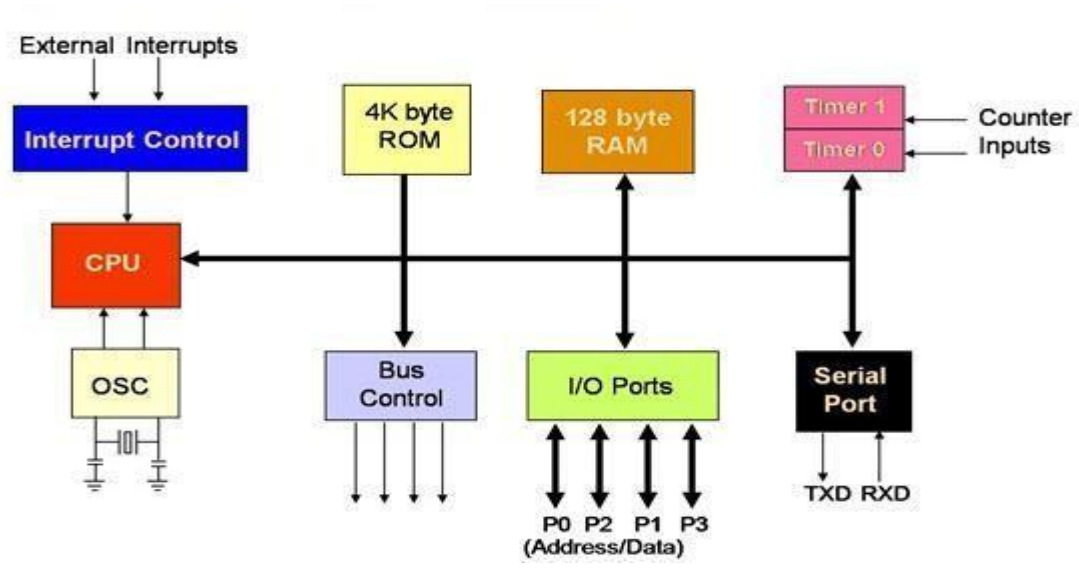
Comparison between 8051 Family Members:
The following table compares the features available in 8051, 8052, and 8031.

Feature	8051	8052	8031
ROM(bytes)	4K	8K	0K
RAM(bytes)	128	256	128
Timers	2	3	2
I/O pins	32	32	32
Serial port	1	1	1
Interrupt sources	6	8	6

Features of 8051 Microcontroller
An 8051 microcontroller comes bundled with the following features – □ 4KB bytes on-chip program memory (ROM)

- 128 bytes on-chip data memory (RAM)
- Four register banks
- 128 user defined software flags
- 8-bit bidirectional data bus
- 16-bit unidirectional address bus
- 32 general purpose registers each of 8-bit
- Two 16-bit Timers
- Three internal and two external Interrupts
- Four 8-bit port
- 16-bit program counter and data pointer

Architecture of 8051:



CPU (Central Processing Unit): CPU act as a mind of any processing machine. It synchronizes and manages all processes that are carried out in microcontroller. User has no power to control the functioning of CPU. It interprets the program stored in ROM and carries out from storage and then performs it projected duty. CPU manage the different types of registers available in 8051 microcontroller.

Interrupts: Interrupts is a sub-routine call that given by the microcontroller when some other program with high priority is request for acquiring the system buses the n interrupts occur in current running program.
Interrupts provide a method to postpone or delay the current process, performs a subroutine task and then restart the standard program again.
Types of interrupt in 8051 Microcontroller:
The five sources of interrupts in 8051 Microcontroller:

Timer 0 overflow interrupt - TF0
Timer 1 overflow interrupt - TF1
External hardware interrupt - INT0
External hardware interrupt - INT1
Serial communication interrupt - RI/TI

Memory: For operation Micro-controller required a program. This program guides the microcontroller to perform the specific tasks. This program installed in microcontroller required some on chip memory for the storage of the program.
Microcontroller also required memory for storage of data and operands for the short duration. In microcontroller 8051 there is code or program memory of 4 KB that is it has 4 KB ROM and it also comprise of data memory (RAM) of 128 bytes.

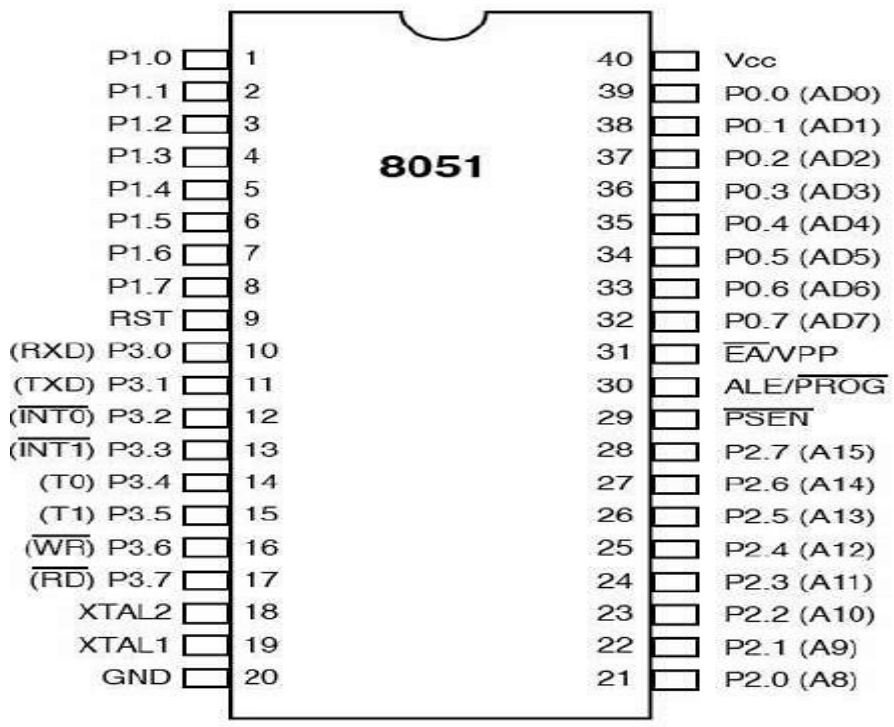
Bus : Bus is a group of wires which uses as a communication canal or acts as means of data transfer. The different bus configuration includes 8, 16 or more cables. Therefore, a bus can bear 8 bits, 16 bits all together.
Types of buses in 8051 Microcontroller:
The two types of bus used in 8051 microcontroller:

Address Bus: 8051 microcontroller consist of 16-bit address bus. It is generally be used for transferring the data from Central Processing Unit to Memory.

Data bus: 8051 microcontroller is consisting of 8 bits data bus. It is generally be used for transferring the data from one peripherals position to other peripherals.

Oscillator: As the microcontroller is digital circuit therefore it needs timer for their operation. To perform timer operation inside microcontroller it required externally connected or on-chip oscillator. Microcontroller is used inside an embedded system for managing the function of devices. Therefore, 8051 uses the two 16-bit counters and timers. For the operation of this timers and counters the oscillator is used inside microcontroller.

Pin configuration of 8051:



Pins 1 to 8 – These pins are known as Port 1. This port doesn't serve any other functions.
It is internally pulled up, bi-directional I/O port.
Pin 9 – It is a RESET pin, which is used to reset the microcontroller to its initial values. **Pins 10 to 17** – These pins are known as Port 3. This port serves some functions like interrupts, timer input, control signals, serial communication signals RxD and TxD, etc. **Pins 18 & 19** – These pins are used for interfacing an external crystal to get the system clock.
Pin 20 – This pin provides the power supply to the circuit.
Pins 21 to 28 – These pins are known as Port 2. It serves as I/O port. Higher order address bus signals are also multiplexed using this port.
Pin 29 – This is PSEN pin which stands for Program Store Enable. It is used to read a signal from the external program memory.
Pin 30 – This is EA pin which stands for External Access input. It is used to enable/disable the external memory interfacing.
Pin 31 – This is ALE pin which stands for Address Latch Enable. It is used to demultiplex the address-data signal of port.
Pins 32 to 39 – These pins are known as Port 0. It serves as I/O port. Lower order address and data bus signals are multiplexed using this port. **Pin 40** – This pin is used to provide power supply to the circuit.

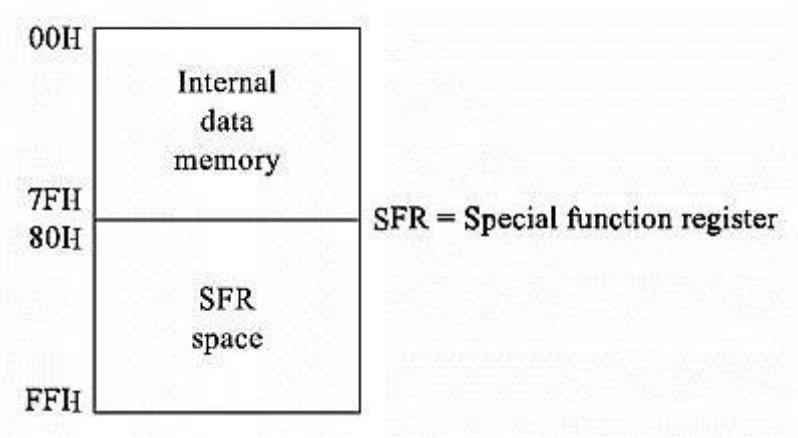
8051 I/O ports:
8051 microcontrollers have 4 I/O ports each of 8-bit, which can be configured as input or output. Hence, total 32 input/output pins allow the microcontroller to be connected with the peripheral devices.

The pins can be configured as 1 for input and 0 for output as per the logic state.

Port 0 – The P0 (zero) port is characterized by two functions –

When the external memory is used then the lower address byte (addresses A0-A7) is applied on it, else all bits of this port are configured as input/output.
When P0 port is configured as an output then other ports consisting of pins with built-in pull-up resistor connected by its end to 5V power supply, the pins of this port have this resistor left out. Port 1
P1 is a true I/O port as it doesn't have any alternative functions as in P0, but this port can be configured as general I/O only. It has a built-in pull-up resistor and is completely compatible with TTL circuits.
Port 2
P2 is similar to P0 when the external memory is used. Pins of this port occupy addresses intended for the external memory chip. This port can be used for higher address byte with addresses A8-A15. When no memory is added then this port can be used as a general input/output port similar to Port 1.
Port 3
In this port, functions are similar to other ports except that the logic 1 must be applied to appropriate bit of the P3 register.

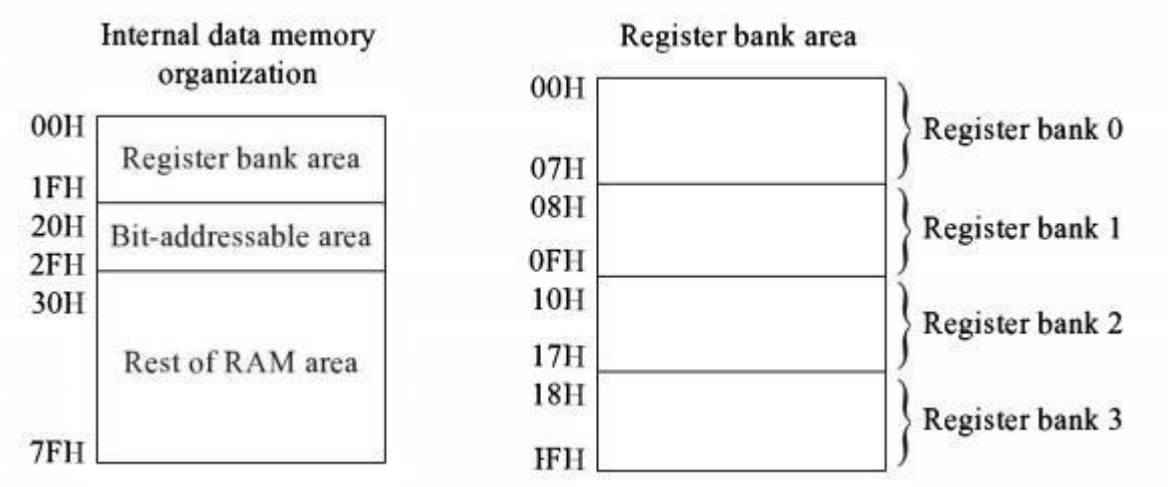
8051 memory organization (RAM organisation):
The 8051 Microcontroller Memory is separated in Program Memory (ROM) and Data Memory (RAM). The Program Memory of the 8051 Microcontroller is used for storing the program to be executed i.e., instructions. The Data Memory on the other hand, is used for storing temporary variable data and intermediate results.
8051 Microcontroller has both Internal ROM and Internal RAM. If the internal memory is inadequate, you can add external memory using suitable circuits.
The internal data memory of 8051 is divided into two groups. These are a set of eight registers, and a scratch pad memory. These eight registers are R0 to R7.
8051 Provides four register bank, but only one register bank can be used at any point of time. To select the register bank, two bits of PSW (Program Status Word) are used.



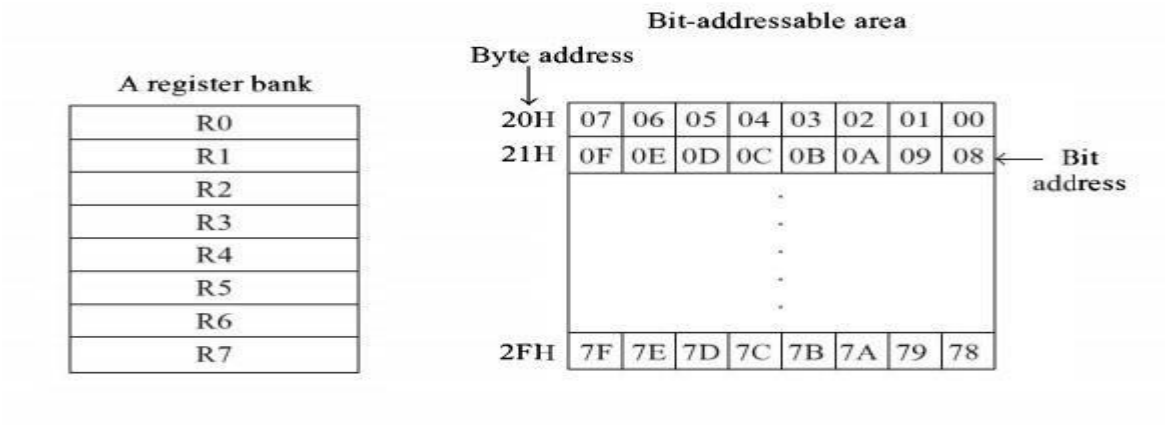
The following addressing can be used to select register banks.

Address Range	Register Bank
00H to 07H	Register Bank 0
08H to 0FH	Register Bank 1
10H to 17H	Register Bank 2
18H to 1FH	Register Bank 3

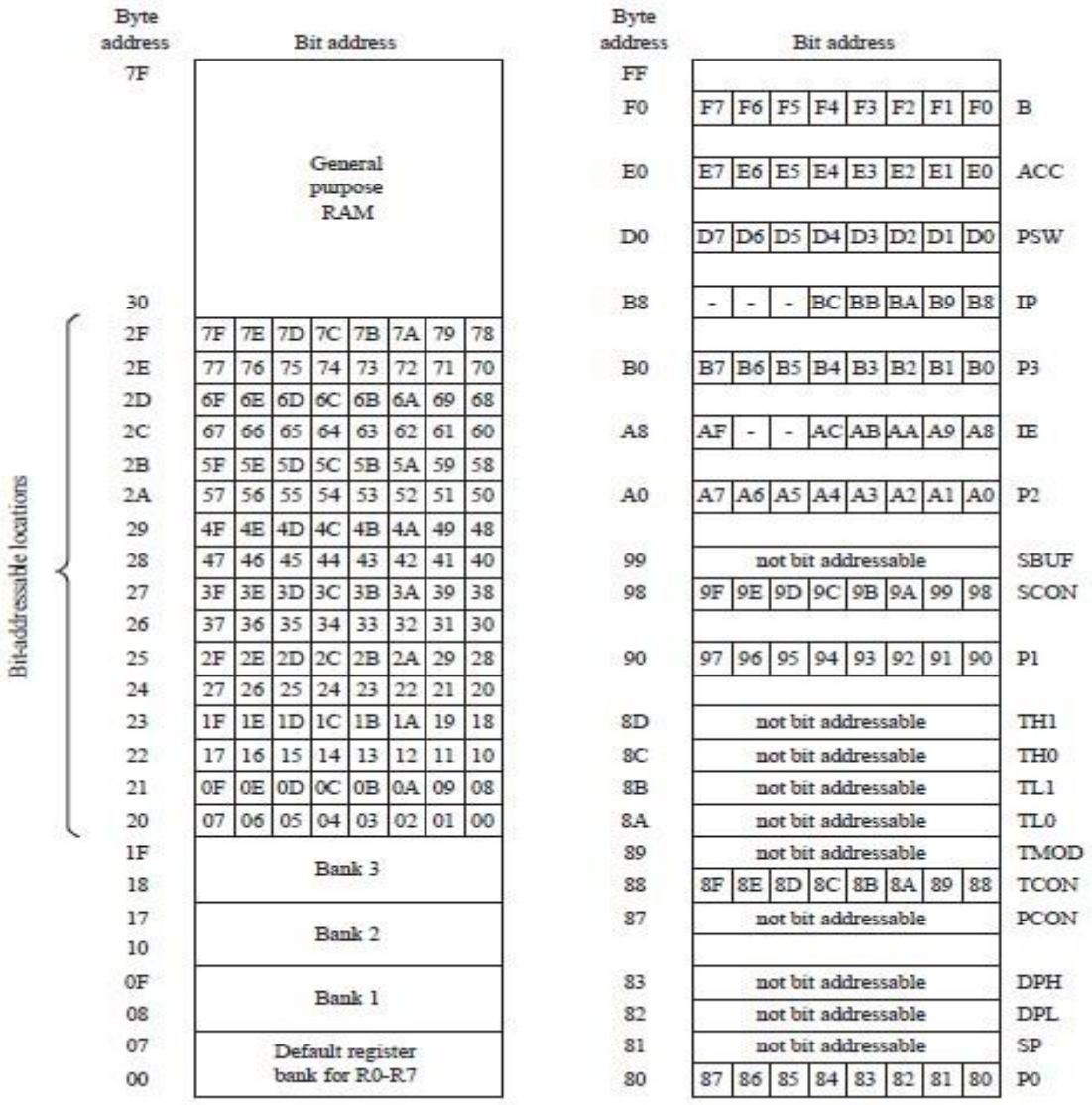
When the 8051 is reset, the Stack Pointer will point to 07H. It means the location 08H to 7FH can be used as a stack.



The scratch pad area will be 20H to 7FH.
From 20H to 2FH (16 bytes or 128 bits) can be used as bit addressable RAM.
For an example the instruction CLR 6FH, using this instruction it clears the location 6FH. The remaining locations (30H to 7EH) of the RAM can be used to store variable data and stack.



Special function registers:
There are 21 Special function registers (SFR) in 8051 micro controller and this includes Register A, Register B, Processor Status Word (PSW), PCON etc.
There are 21 unique locations for these 21 special function registers and each of these register is of 1 byte size.
Some of these special function registers are bit addressable (which means you can access 8 individual bits inside a single byte), while some others are only byte addressable.



Registers of 8051:
Register A/Accumulator:
The most important of all special function registers, is known as ACC or A. The Accumulator holds the result of most of arithmetic and logic operations. ACC is usually accessed by direct addressing and its physical address is E0H. Accumulator is both byte and bit addressable.

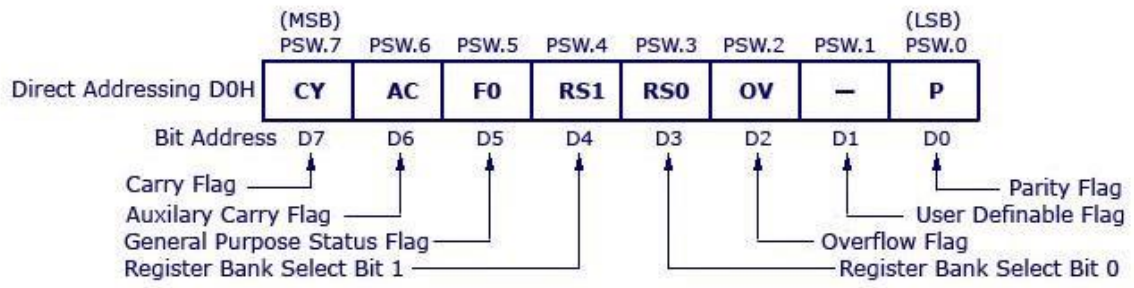
Register B:
The major purpose of this register is in executing multiplication and division. The 8051 micro controller has a single instruction for multiplication (MUL) and division (DIV). Ex: MUL A,B – When this instruction is executed, data inside A and data inside B is multiplied and answer is stored in A.
For MUL and DIV instructions, it is necessary that the two operands must be in A and B. Register B is also byte addressable and bit addressable. To access bit o or to access all 8 bits (as a single byte), physical address F0 is used.

Port Registers:
There are 4 ports named P0, P1, P2 and P3. Data must be written into port registers first to send it out to any other external device through ports. Similarly, any data received through ports must be read from port registers for performing any operation. All 4 port registers are bit as well as byte addressable.
The physical address of port 0 is 80, port 1 is 90, port 2 is A0 and that of port 3 is B0.

Stack Pointer:
Known as SP, stack pointer represents a pointer to the system stack. Stack pointer is an 8-bit register, the direct address of SP is 81H and it is only byte addressable, which means you can't access individual bits of stack pointer. The content of the stack pointer points to the last stored location of system stack. To store something new in system stack, the SP must be incremented by 1 first and then execute the "store" command. Usually after a system reset SP is initialized as 07H and data can be stored to stack from 08H onwards.

Power Management Register (PCON):
As the name indicates, this register is used for efficient power management of 8051 micro controller. Commonly referred to as PCON register, this is a dedicated SFR for power management alone.
Mobile phone automatically going into stand by mode when not used for a couple of seconds or minutes. This is achieved by power management feature of the controller used inside that phone.

Program Status Word (PSW):
Commonly known as the PSW register. This register reflects the status of the operation that is being carried out in the processor. Register banks are selected using PSW register bits – RS1 and RS0. PSW register is both bit and byte addressable. The physical address of PSW starts from D0H.



Bit No	Bit Symbol	Direct Address	Name	Function
0	P	D0	Parity	This bit will be set if ACC has odd number of 1's after an operation. If not, bit will remain cleared.
1	–	D1		User definable bit
2	OV	D2	Overflow	OV flag is set if there is a carry from bit 6 but not from bit 7 of an Arithmetic operation. It's also set if there is a carry from bit 7 (but not from bit 6) of Acc
3	RS0	D3	Register Bank select bit 0	LSB of the register bank select bit. Look for explanation below this table.
4	RS1	D4	Register Bank select bit 1	MSB of the register bank select bits.
5	F0	D5	Flag 0	User defined flag
6	AC	D6	Auxiliary carry	This bit is set if data is coming out from bit 3 to bit 4 of Acc during an Arithmetic operation.
7	CY	D7	Carry	Is set if data is coming out of bit 7 of Acc during an Arithmetic operation.

The Data Pointer:
The Data Pointer (DPTR) is the 8051's only user-accessible 16-bit (2-byte) register. DPTR is meant for pointing to data. It is used by the 8051 to access external memory using the address indicated by DPTR. DPTR is the only 16-bit register available and is often used to store 2-byte values.

The Program Counter:
The Program Counter (PC) is a 2-byte address which tells the 8051 where the next instruction to execute can be found in the memory. PC starts at 0000h when the 8051 initializes and is incremented every time after an instruction is executed. PC is not always incremented by 1. Some instructions may require 2 or 3 bytes; in such cases, the PC will be incremented by 2 or 3.

Interrupts in 8051:
The five sources of interrupts in 8051 Microcontroller:

- Timer 0 overflow interrupt - TF0
- External hardware interrupt - INT0
- Timer 1 overflow interrupt - TF1
- External hardware interrupt - INT1
- Serial communication interrupt - RI/TI

When interrupt occur then the microcontroller executes the interrupt service routine. Therefore, the memory location corresponds to interrupt enables it.

Interrupt Number	Interrupt Description	Address
0	EXTERNAL INT 0	0003h
1	TIMER/COUNTER 0	000Bh
2	EXTERNAL INT 1	0013h
3	TIMER/COUNTER 1	001Bh
4	SERIAL PORT	0023h

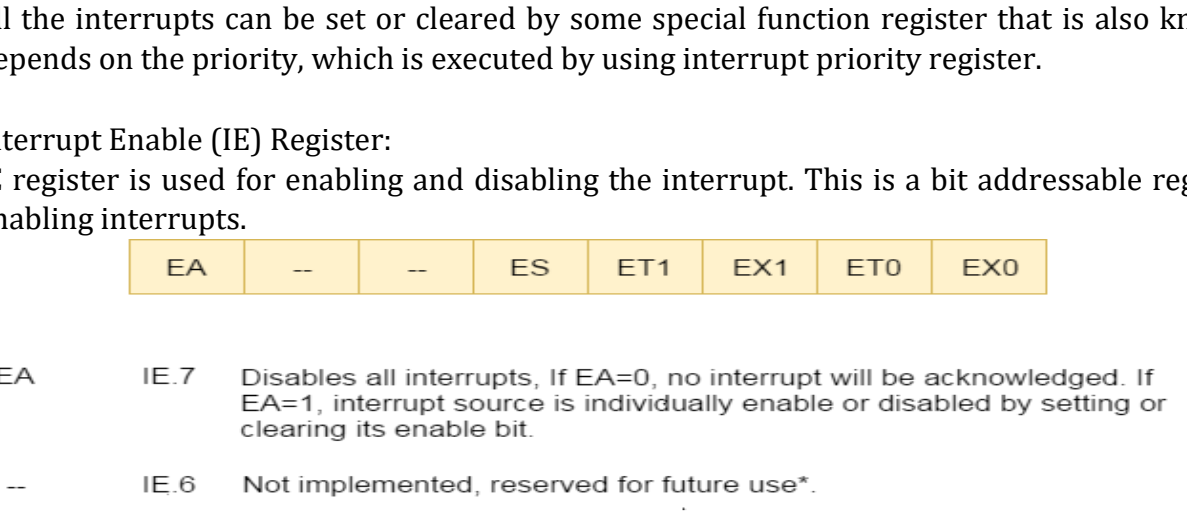
All the interrupts can be set or cleared by some special function register that is also known as interrupt enabled (IE), and it is totally depends on the priority, which is executed by using interrupt priority register.

Interrupt Enable (IE) Register:
IE register is used for enabling and disabling the interrupt. This is a bit addressable register in which EA value must be set to one for enabling interrupts.

EA	--	--	ES	ET1	EX1	ET0	EX0

EA	IE.7	Disables all interrupts, If EA=0, no interrupt will be acknowledged. If EA=1, interrupt source is individually enable or disabled by setting or clearing its enable bit.
--	IE.6	Not implemented, reserved for future use*.
--	IE.5	Not implemented, reserved for future use*.
ES	IE.4	Enable or disable the Serial port interrupt.
ET1	IE.3	Enable or disable the Timer 1 overflow interrupt.
EX1	IE.2	Enable or disable External interrupt 1.
ET0	IE.1	Enable or disable the Timer 0 overflow interrupt.
EX0	IE.0	Enable or disable External interrupt 0.

Interrupt Priority Register (IP)
Using IP register, it is possible to change the priority levels of an interrupts by clearing or setting the individual bit in the Interrupt priority (IP) register as shown in figure. It allows the low priority interrupt can interrupt the high-priority interrupt, but it prohibits the interruption by using another low-priority interrupt. If the priorities of interrupt are not programmed, then microcontroller executes the instruction in a predefined manner and its order are INT0, TF0, INT1, TF1, and SI.



Timers of 8051 and their Associated Registers:
The 8051 has two timers, Timer 0 and Timer 1. They can be used as timers or as event counters. Both Timer 0 and Timer 1 are 16-bit wide. Since the 8051 follows an 8-bit architecture, each 16 bit is accessed as two separate registers of low-byte and high-byte.

Timer 0 Register:
The 16-bit register of Timer 0 is accessed as low- and high-byte. The low-byte register is called TL0 (Timer 0 low byte) and the high-byte register is called TH0 (Timer 0 high byte). These registers can be accessed like any other register.
For example, the instruction MOV TL0, #4H moves the value into the low-byte of Timer #0.

Timer 1 Register:
The 16-bit register of Timer 1 is accessed as low- and high-byte. The low-byte register is called TL1 (Timer 1 low byte) and the high-byte register is called TH1 (Timer 1 high byte). These registers can be accessed like any other register. For example, the instruction MOV TL1, #4H moves the value into the low-byte of Timer 1.

TMOD (Timer Mode) Register:
Both Timer 0 and Timer 1 use the same register to set the various timer operation modes. It is an 8-bit register in which the lower 4 bits are set aside for Timer 0 and the upper four bits for Timers.

TMOD : Timer/Counter Mode Control Register (Not Bit Addressable)

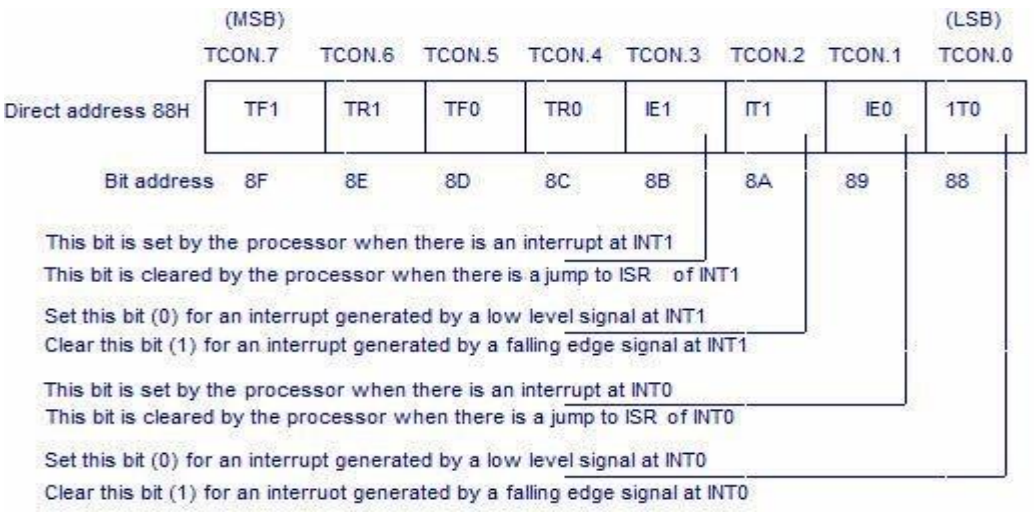
GATE	C/$\overline{T}$	M1	M0	GATE	C/$\overline{T}$	M1	M0
TIMER 1				TIMER 0			

GATE	When TRx (in TCON) is set and GATE = 1, TIMER/COUNTERx will run only while INTx pin is high (hardware control). When GATE = 0, TIMER/COUNTERx will run only while TRx = 1 (software control).
C/$\overline{T}$	Timer or Counter selector. Cleared for Timer operation (input from internal system clock). Set for Counter operation (input from Tx input pin).
M1	Mode selector bit (NOTE 1).
M0	Mode selector bit (NOTE 1).

Note 1 :

M1	M0		OPERATING MODE
0	0	0	13–bit Timer
0	1	1	16–bit Timer/Counter
1	0	2	8–bit Auto–Reload Timer/Counter
1	1	3	(Timer 0) TL0 is an 8–bit Timer/Counter controlled by the standard Timer 0 control bits, TH0 is an 8–bit Timer and is controlled by Timer 1 control bits.
1	1	3	(Timer 1) Timer/Counter 1 stopped.

TCON register:
TCON is another register used to control operations of counter and timers in microcontrollers. It is an 8-bit register wherein four upper bits are responsible for timers and counters and lower bits are responsible for interrupts.



Addressing modes of 8051:
In 8051 There are six types of addressing modes.

- Immediate Addressing Mode
- Register Addressing Mode
- Direct Addressing Mode
- Register Indirect Addressing Mode
- Indexed Addressing Mode
- Implied Addressing Mode

Immediate addressing mode:
In this Immediate Addressing Mode, the data is provided in the instruction itself. The data is provided immediately after the opcode.

```
MOV A, #0AFH;
MOV R3, #45H;
MOV DPTR, #FE00H;
```

symbol is used for immediate data.

Register addressing mode:

In the register addressing mode the source or destination data should be present in a register (R0 to R7).

```
MOV A, R5;
MOV R2, #45H;
MOV R0, A;
```

MOV R0, R1 //Invalid instruction, GPR to GPR not possible//

Direct Addressing Mode:
In the Direct Addressing Mode, the source or destination address is specified by using 8bit data in the instruction. Only the internal data memory can be used in this mode.

```
MOV 80H, R6;
MOV R2, 45H;
MOV R0, 05H;
```

Register indirect addressing Mode:
In this mode, the source or destination address is given in the register. By using register indirect addressing mode, the internal or external addresses can be accessed. The R0 and R1 are used for 8-bit addresses, and DPTR is used for 16-bit addresses, no other registers can be used for addressing purposes.

```
MOV 0E5H, @R0;
MOV @R1, 80H
```

@ symbol is used for register indirect addressing.

```
MOVX A, @R1;
MOV @DPTR, A;
```

X in MOVX indicates the external data memory.

Indexed addressing mode:
In the indexed addressing mode, the source memory can only be accessed from program memory only. The destination operand is always the register A.

```
MOVC A, @A+PC;
MOVC A, @A+DPTR;
```

The C in MOVC instruction refers to code byte. For the first instruction, let us consider A holds 30H. And the PC value is1125H. The contents of program memory location 1155H (30H + 1125H) are moved to register A.

Implied Addressing Mode:
In the implied addressing mode, there will be a single operand. These types of instruction can work on specific registers only. These types of instructions are also known as register specific instruction.

```
RLA;
SWAP A;
```

Instruction set of 8051:
The instructions of 8051 Microcontroller can be classified into five different groups.

- Data Transfer Group
- Arithmetic Group
- Logical Group
- Program Branch Group
- Bit Processing Group also known as Boolean Variable Manipulation.

Like 8085, some instruction has two operands. The first operand is the Destination, and the second operator is Source.
Data transfer group:

Mnemonic	Instruction	Description	Addressing Mode	# of Bytes	# of Cycles
MOV	A, #Data	A ← Data	Immediate	2	1
	A, Rn	A ← Rn	Register	1	1
	A, Direct	A ← (Direct)	Direct	2	1
	A, @Ri	A ← @Ri	Indirect	1	1
	Rn, #Data	Rn ← data	Immediate	2	1
	Rn, A	Rn ← A	Register	1	1
	Rn, Direct	Rn ← (Direct)	Direct	2	2
	Direct, A	(Direct) ← A	Direct	2	1
	Direct, Rn	(Direct) ← Rn	Direct	2	2
	Direct1, Direct2	(Direct1) ← (Direct2)	Direct	3	2
	Direct, @Ri	(Direct) ← @Ri	Indirect	2	2
	Direct, #Data	(Direct) ← #Data	Direct	3	2
	@Ri, A	@Ri ← A	Indirect	1	1
	@Ri, Direct	@Ri ← Direct	Indirect	2	2
	@Ri, #Data	@Ri ← #Data	Indirect	2	1
	DPTR, #Data16	DPTR ← #Data16	Immediate	3	2
MOVC	A, @A+DPTR	A ← Code Pointed by A+DPTR	Indexed	1	2
	A, @A+PC	A ← Code Pointed by A+PC	Indexed	1	2
	A, @Ri	A ← Code Pointed by Ri (8-bit Address)	Indirect	1	2
MOVX	A, @DPTR	A ← External Data Pointed by DPTR	Indirect	1	2
	@Ri, A	@Ri ← A (External Data 8-bit Addr)	Indirect	1	2
	@DPTR, A	@DPTR ← A (External Data 16-bit Addr)	Indirect	1	2
PUSH	Direct	Stack Pointer SP ← (Direct)	Direct	2	2
POP	Direct	(Direct) ← Stack Pointer SP	Direct	2	2
XCH	Rn	Exchange ACC with Rn	Register	1	1
	Direct	Exchange ACC with Direct Byte	Direct	2	1
	@Ri	Exchange ACC with Indirect RAM	Indirect	1	1
XCHD	A, @Ri	Exchange ACC with Lower Order Indirect RAM	Indirect	1	1

Arithmetic group:

Mnemonic	Instruction	Description	Addressing Mode	# of Bytes	# of Cycles
ADD	A, #Data	A ← A + Data	Immediate	2	1
	A, Rn	A ← A + Rn	Register	1	1
	A, Direct	A ← A + (Direct)	Direct	2	1
	A, @Ri	A ← A + @Ri	Indirect	1	1
ADDC	A, #Data	A ← A + Data + C	Immediate	2	1
	A, Rn	A ← A + Rn + C	Register	1	1
	A, Direct	A ← A + (Direct) + C	Direct	2	1
	A, @Ri	A ← A + @Ri + C	Indirect	1	1
SUBB	A, #Data	A ← A – Data – C	Immediate	2	1
	A, Rn	A ← A – Rn – C	Register	1	1
	A, Direct	A ← A – (Direct) – C	Direct	2	1
	A, @Ri	A ← A – @Ri – C	Indirect	1	1
MUL	AB	Multiply A with B (A ← Lower Byte of A*B and B ← Higher Byte of A*B)	--	1	4
DIV	AB	Divide A by B (A ← Quotient and B ← Remainder)	--	1	4
DEC	A	A ← A – 1	Register	1	1
	Rn	Rn ← Rn – 1	Register	1	1
	Direct	(Direct) ← (Direct) – 1	Direct	2	1
	@Ri	@Ri ← @Ri – 1	Indirect	1	1
INC	A	A ← A + 1	Register	1	1
	Rn	Rn ← Rn + 1	Register	1	1
	Direct	(Direct) ← (Direct) + 1	Direct	2	1
	@Ri	@Ri ← @Ri + 1	Indirect	1	1
	DPTR	DPTR ← DPTR + 1	Register	1	2
DA	A	Decimal Adjust Accumulator	--	1	1

Logical group:

Mnemonic	Instruction	Description	Addressing Mode	# of Bytes	# of Cycles
ANL	A, #Data	A ← A AND Data	Immediate	2	1
	A, Rn	A ← A AND Rn	Register	1	1
	A, Direct	A ← A AND (Direct)	Direct	2	1
	A, @Ri	A ← A AND @Ri	Indirect	1	1
	Direct, A	(Direct) ← (Direct) AND A	Direct	2	1
	Direct, #Data	(Direct) ← (Direct) AND #Data	Direct	3	2
ORL	A, #Data	A ← A OR Data	Immediate	2	1
	A, Rn	A ← A OR Rn	Register	1	1
	A, Direct	A ← A OR (Direct)	Direct	2	1
	A, @Ri	A ← A OR @Ri	Indirect	1	1
	Direct, A	(Direct) ← (Direct) OR A	Direct	2	1
	Direct, #Data	(Direct) ← (Direct) OR #Data	Direct	3	2
XRL	A, #Data	A ← A XRL Data	Immediate	2	1
	A, Rn	A ← A XRL Rn	Register	1	1
	A, Direct	A ← A XRL (Direct)	Direct	2	1
	A, @Ri	A ← A XRL @Ri	Indirect	1	1
	Direct, A	(Direct) ← (Direct) XRL A	Direct	2	1
	Direct, #Data	(Direct) ← (Direct) XRL #Data	Direct	3	2
CLR	A	A← 00H	--	1	1
CPL	A	A ← A	--	1	1
RL	A	Rotate ACC Left	--	1	1
RLC	A	Rotate ACC Left through Carry	--	1	1
RR	A	Rotate ACC Right	--	1	1
RRC	A	Rotate ACC Right through Carry	--	1	1
SWAP	A	Swap Nibbles within ACC	--	1	1

Program branch group:

Mnemonic	Instruction	Description	# of Bytes	# of Cycles
ACALL	ADDR11	Absolute Subroutine Call PC + 2 → (SP); ADDR11 → PC	2	2
LCALL	ADDR16	Long Subroutine Call PC + 3 → (SP); ADDR16 → PC	3	2
RET	--	Return from Subroutine (SP) → PC	1	2
RETI	--	Return from Interrupt	1	2
AJMP	ADDR11	Absolute Jump ADDR11 → PC	2	2
LJMP	ADDR16	Long Jump ADDR16 → PC	3	2
SJMP	rel	Short Jump PC + 2 + rel → PC	2	2
JMP	@A + DPTR	A + DPTR → PC	1	2
JZ	rel	If A=0, Jump to PC + rel	2	2
JNZ	rel	If A ≠ 0, Jump to PC + rel		
CJNE	A, Direct, rel	Compare (Direct) with A. Jump to PC + rel if not equal	3	2
	A, #Data, rel	Compare #Data with A. Jump to PC + rel if not equal	3	2
	Rn, #Data, rel	Compare #Data with Rn. Jump to PC + rel if not equal	3	2
	@Ri, #Data, rel	Compare #Data with @Ri. Jump to PC + rel if not equal	3	2
DJNZ	Rn, rel	Decrement Rn. Jump to PC + rel if not zero	2	2
	Direct, rel	Decrement (Direct). Jump to PC + rel if not zero	3	2
NOP		No Operation	1	1

Bit manipulation group:

Mnemonic	Instruction	Description	# of Bytes	# of Cycles
CLR	C	C ← 0 (C = Carry Bit)	1	1
	Bit	Bit ← 0 (Bit = Direct Bit)	2	1
SET	C	C ← 1	1	1
	Bit	Bit ← 1	2	1
CPL	C	C ← $\overline{C}$	1	1
	Bit	Bit ← $\overline{Bit}$	2	1
ANL	C, /Bit	C ← C. $\overline{Bit}$ (AND)	2	1
	C, Bit	C ← C . Bit (AND)	2	1
ORL	C, /Bit	C ← C + $\overline{Bit}$ (OR)	2	1
	C, Bit	C ← C + Bit (OR)	2	1
MOV	C, Bit	C ← Bit	2	1
	Bit, C	Bit ← C	2	2
JC	rel	Jump is Carry (C) is Set	2	2
JNC	rel	Jump is Carry (C) is Not Set	2	2
JB	Bit, rel	Jump is Direct Bit is Set	3	2
JNB	Bit, rel	Jump is Direct Bit is Not Set	3	2
JBC	Bit, rel	Jump is Direct Bit is Set and Clear Bit	3	2

Assembler Directives:

The assembling directives give the directions to the CPU. The 8051 microcontroller consists of various kinds of assembly directives to give the direction to the control unit. The most useful directives are 8051 programming, such as:

ORG
DB
EQU
END

ORG (origin): This directive indicates the start of the program. This is used to set the register address during assembly. For example; ORG 0000H tells the compiler all subsequent code starting at address 0000H.
Syntax: ORG 0000H

DB (define byte): The define byte is used to allow a string of bytes. For example, print the “EDGEFX” wherein each character is taken by the address and finally prints the “string” by the DB directly with double quotes.
Syntax:

```
ORG 0000H
MOV A, #00H
————-
————-
DB ”EDGEFX”
```

EQU (equivalent): The equivalent directive is used to equate address of the variable. Syntax:

```
reg equ,09H
————–
```

———————–
MOV reg,#2H

END: The END directive is used to indicate the end of the program.
Syntax:
reg equ,09H
———————–
———————–
MOV reg, #2H
END

Assembly language program: Addition:
ORG 0000H
MOV R0, #03H // move the value 3 to the register R0//
MOV A, #05H // move the value 5 to accumulator A//
Add A, 00H // add A value with R0 value and stores the result inA// END
Multiplication:
ORG 0000H
MOV R0, #03H // move the value 3 to the register R0//
MOV A, #05H // move the value 5 to accumulator A//
MUL A, 03H // Multiplied result is stored in the Accumulator A //
END

Subtraction:
ORG 0000H
MOV R0, #03H // move the value 3 to register R0//
MOV A, #05H // move the value 5 to accumulator A//
SUBB A, 03H // Result value is stored in the Accumulator A //
END

Division:
ORG 0000H
MOV R0, #03H // move the value 3 to register R0//
MOV A, #15H // move the value 5 to accumulator A//
DIV A, 03H // final value is stored in the Accumulator A //
END

Treat R6-R7 and R4-R5 as two 16-bit registers. Perform subtraction between them. Store the result in 20h (lower byte) and 21h (higher byte).

```
        CLR C              ; clear carry
        MOV A, R4          ; get first lower byte
        SUBB A, R6         ; subtract it with other
        MOV 20H, A         ; store the result
        MOV A, R5          ; get the first higher byte
        SUBB A, R7         ; subtract from other
        MOV 21H, A         ; store the higher byte
         END
```

Jump, Loop and Call program:
Transfer the block of data from 20H to 30H to external location 1020H to 1030H. Solution: – here we have to transfer 10 data bytes from internal to external RAM. So first, we need one counter. Then we need two pointers one for source second for destination.

```
        MOV R7, #0AH       ; initialize counter by 10d
        MOV R0, #20H       ; get initial source location
        MOV DPTR, #1020H   ; get initial destination location
NXT:    MOV A, @R0         ; get first content in acc
        MOVX @DPTR, A      ; move it to external location
        INC R0             ; increment source location
        INC DPTR           ; increase destination location
        DJNZ R7, NXT       ; decrease r7. if zero then over otherwise move next
        END
```

Find out how many equal bytes between two memory blocks 10H to 20H and 20H to 30H. Solution: – here we shall compare each byte one by one from both blocks. Increase the count every time when equal bytes are found

```
        MOV R7, #0AH       ; initialize counter by 10d
        MOV R0, #10H       ; get initial location of block1
        MOV R1, #20H       ; get initial location of block2
        MOV R6, #00H       ; equal byte counter. Starts from zero
```

```
NXT:    MOV A, @R0              ; get content of block 1 in acc
        MOV B, A                ; move it to B
        MOV A, @R1              ; get content of block 2 in acc
        CJNE A, B, NOMATCH      ; compare both if equal
        INC R6                  ; increment the counter
NOMATCH: INC R0                 ; otherwise go for second number
        INC R1
        DJNZ R7, NXT            ; decrease R7. if zero then over otherwise move next
```

Counters and subroutines:
The crystal frequency is given as 12 MHz. Make a subroutine that will generate delay of exact 1 ms. Use this delay to generate square wave of 50 Hz on pin P2.0
Solution: – 50 Hz means 20 ms. And because of square wave 10 ms ontime and 10 ms offtime. So for 10 ms we shall send 1 to port pin and for another 10 ms send 0 in continuous loop.

```
LOOP:   MOV R6, #0AH        ; load 10d in r6
        ACALL DELAY         ; call 1 ms delay ×10 = 10 ms
        CLR P2.0            ; send 0 to port pin
        MOV R6, #0AH        ; load 10d in r6
        ACALL DELAY         ; call 1 ms delay ×10 = 10 ms
        SJMP LOOP           ; continuous loop
 DELAY:                     ; load count 250d
LP2:    MOV R7, #0FAH
LP1:    NOP                 ; 1 cycle
        NOP                 ; 1+1=2 cycles
        DJNZ R7, LP1        ; 1+1+2 = 4 cycles
        DJNZ R6, LP2        ; 4×250 = 1000 cycles = 1000 µs = 1 ms
        RET
```

I/O programming:

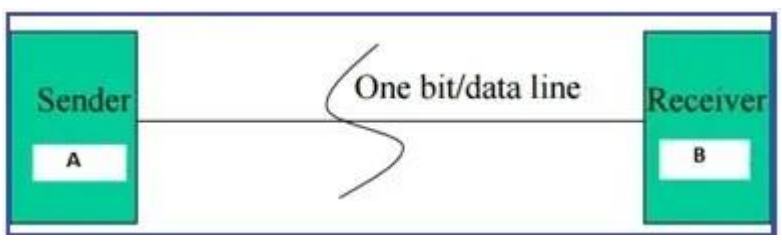
1. Toggle all bits of P2 continuously

```
        MOV A,#55
BACK:   MOV P2,A
        ACALL DELAY
        CPL A               ;complement (invert) reg. A
        SJMP BACK
```

2. Port 1 is configured to be used as an output port, then to use it as an input port again

```
MOV A ,#0FFH        ;A = FF hex
MOV P1,A            ;Make P1 an input port
MOV A,P1            ;get data from P1
MOV R7,A            ;save it in Reg R7
ACALL DELAY         ;wait
```

Serial communication in 8051:
Serial communication may be simplex, half-duplex or full-duplex.
Simplex communication means that data will be transmitted only in one direction while half-duplex means data will be transmitted in both directions but at one time, only one device can transmit, whereas full-duplex means data may be transmitted in both directions at one time, while one device is transmitting, it can also receive data transmitted from other devices at same time.
The transmitter and receiver are configured to communicate at some data transfer rate before communication starts. This data transfer rate or a number of bits transmitted per second is called the baud rate for handling serial communication.
Parallel communication is fast but it is not applicable for long distances (for printers). Moreover, it is also expensive.
Serial is not much fast as parallel communication but it can deal with transmission of data over longer distances (for telephone line, ADC, DAC).
It is also cheaper and requires less physical wires, that's why we use serial communication.
The "Serial Control" (SCON) is the SFR which is used to configure serial port.



METHODS OF SERIAL COMMUNICATION:
There are two methods of Serial Communication:
ASYNCHRONOUS and SYNCHRONOUS
SYNCHRONOUS: Transfer the block of data (characters) between sender and receiver spaced by fixed time interval. This transmission is synchronized by an external clock.

ASYNCHRONOUS: There is no clock involved here and transmission is synchronized by special signals along the transmission medium. It transfers a single byte at a time between sender and receiver along with inserting a start bit before each data character and a stop bit at its termination so that to inform the receiver where the data begins and ends. An example is the interface between a keyboard and a computer. Keyboard is the transmitter and the computer is the receiver.
We use USART and UART for serial communications. USART or UART is a microcontroller peripheral which converts incoming and outgoing bytes of data into a serial bit stream. Both have same work but with different methods which is explained below.

USART:
USART stands for Universal Synchronous/Asynchronous Receiver-Transmitter. USART uses external clock so it needs separate line to carry the clock signal. Sending peripheral generates a clock and the receiving peripheral recover from the data stream without knowing the baud rate ahead of time. By use of external clock, USART's data rate can be much higher (up to rates of 4 Mbps) than that of a standard UART.

UART:
It stands for Universal Asynchronous Receiver-Transmitter. A UART generates its internal data clock to the microcontroller. It synchronizes that clock with the data stream by using the start bit transition. The receiver needs the baud rate to know ahead of time to properly receive the data stream.