UNIT-1

8086 Architecture

Introduction to microprocessors

A Microprocessors is a computer processor which incorporates the functions of a computer's central processing unit (CPU) on a single integrated circuit (IC), or at most a few integrated circuits.

The microprocessor is a multipurpose, clock driven, Register based, digital-integrated circuit which accepts binary data as input, processes it according to instructions stored in its memory, and provides results as output. Microprocessors contain both combinational logic and sequential digital logic. Microprocessors operate on numbers and symbols represented in the binary numeral system.

Generation of Microprocessors:-

① Intel 4004 (1971).
- -> 4-bit microprocessor
- -> 4 KB main memory
- -> 45 instructions
- -> PMOS Technology
- -> was first programmable device which was used in calculators.

(2) INTEL 8008 (1972):-

-> 8-bit version of 4004
-> 16 KB main memory
-> 48 instructions
-> pmos technology.
-> slow.

(3) INTEL 8080 (1973):-

-> 8-bit microprocessor
-> 64 kB main memory
-> 2 microseconds clock cycle time
-> 500,000 instructions/sec.
-> 10 X faster than 8008
-> N mos Technology.
-> Drawback was that it needed three power supplies.
-> Small computers (microcomputers) were designed in mid 1970's using 8080 as CPU.

(4) INTEL 8086/8088:-

-> year of introduction 1978 for 8086 and 1979 for 8088.
-> 16-bit Microprocessors.
-> Data bus width of 8086 is 16 bit and 8-bit for 8088
-> 1MB main memory.

→ 400 nanoseconds clock time

→ 6 byte instruction cache for 8086 and 4 byte for 8088

→ In 1981 IBM decided to use 8088 in the personal computer.

⑤ INTEL 80186 (1982)

→ 16-bit microprocessor - upgraded version of 8086

→ 1 MB main memory.

→ contained special h/w like programmable counters, interrupt controller etc.

→ Never used in the PC.

→ But was ideal for systems that required a minimum of hardware.

⑥ INTEL 80286 (1983):

→ INTEL

→ 16-bit high performance microprocessor with memory management & protection.

→ 16 MB main memory

→ Few additional instructions to handle extra 15 MB

→ Instruction execution time is as little as 250 ns.

→ Concentrates on the features needed to implement Multitasking.

→ Intel 80386 (1986)
→ Intel 80486 (1989)
→ Pentium (1993)
→ Pentium pro (1995)
→ Pentium II (1997)
→ Pentium III (1999)
→ Pentium IV (2002)

→ Latest is Intel iq processor.

⇒ 8086 Architecture:-

→ the 8086 is mainly divided into mainly two blocks
① execution Unit (EU)
② Bus Interface (BIU)
→ Dividing the work b/w these two will speedup the processing

① Execution Unit (EU):-

→ the execution unit tells the BIU where to fetch instructions or data from
→ decodes instruction and
→ executes instructions

→ the execution unit contains:-

↓

→ The Execution unit contains:

① control Circuitry
② ALU
③ FLAGS
④ General purpose Registers
⑤ pointer and Index Registers.

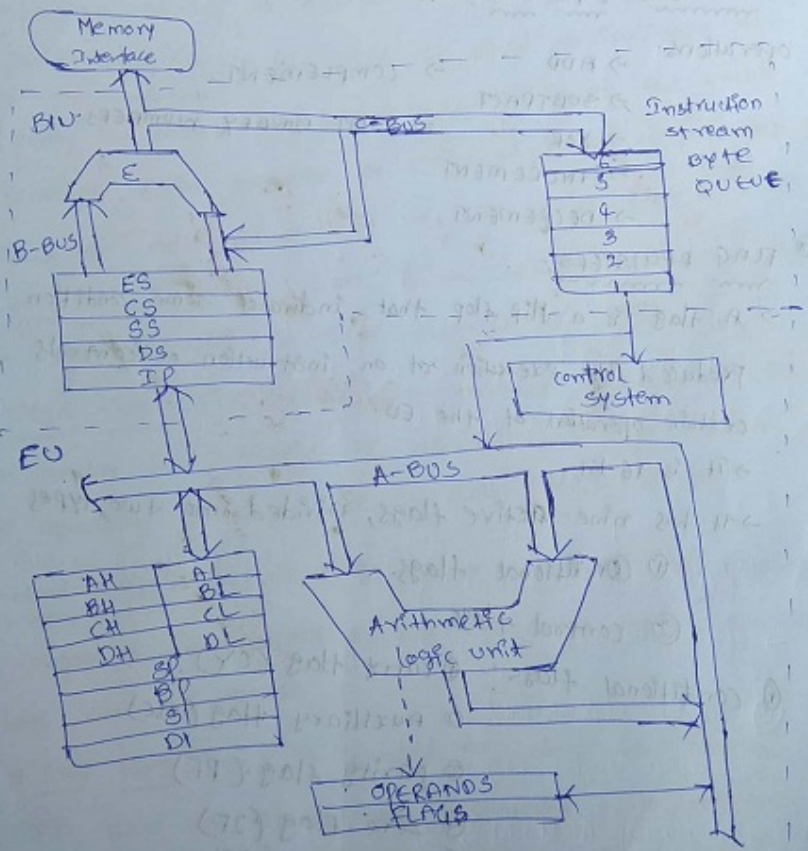


Fig:- Architecture of 8086

① control circuitry :-

→ It directs internal operations.

→ A decoder in the EU translates instruction fetched from memory into series of actions which the EU carries out.

② Arithmetic logic unit :- 16-bit ALU used to carry the operations.

- → ADD
- → SUBTRACT
- → XOR
- → INCREMENT
- → DECREMENT
- → COMPLEMENT
- → SHIT BINARY NUMBERS

③ FLAG REGISTERS :-

→ A flag is a flip flop that indicates some condition produced by execution of an instruction or controls certain operation of the EU.

→ It is 16 bit

→ It has nine active flags, divided into two types

① Conditional flags.

② control flags.

① Conditional flags :- ① carry flag (CY)

② Auxiliary flag (AC)

③ parity flag (PF)

④ zero flag (ZF)

⑤ sign Flag (SF)

① carry flag (CY) :- This flag indicates an overflow condition for unsigned integer arithmetic. It is also used in multiple-precision arithmetic.

Flag Register :-

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	OF	DF	IF	TF	SF	ZF	X	AC	X	PF	X	CY

X = undefined

SF = sign flag → when S=1 if MSB of ALU result = 1

ZF = Zero flag → when Z=1 if ALU result = 00H

PF = parity flag, when Even P=1 if ALU result has even parity

CY = carry flag; CY=1 if carry (or) Borrow occurred during additions & subtractions.

AC = Auxiliary carry → carry from lower to upper nibble (or) Borrow from upper to lower nibble.

Ex:- ① Hex decimal

BC → B 12 | 1011 1100

DE → D 13 | 1101 1110

1 1 1 1 1 1

① 9A ① 1001 1010

26 - 16 = 10

25 - 16 = 9

S = 1

Z = 0

P = 1

CY = 1

AC = 1

Ex:- ① D7 D6 D5 D4 D3 D2 D1 D0

BC 1 0 1 1 1 1 0 0

DE 1 1 0 1 1 1 1 0

1 1 1 1 1 1 1 1

① 9 A ① 1 0 0 1 1 0 1 0

carry 9A

⑤ Auxiliary Flag (AC):- It 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.

⑥ 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.

⑦ Zero Flag (ZF):- It is set; If the result of arithmetic or logical operation is zero else it is reset.

⑧ Sign Flag (SF): In sign magnitude format the sign of number is indicated by MBS bit. If the result of operation is negative, sign flag is set.

② 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.

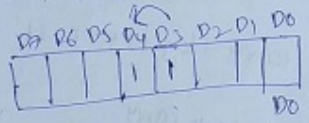
Ex ②

carry flag → If 1 → carry generated
0 → carry Not generated

```
  FFFFH
+ 0 001H
-------
 (1)0000
      Accumulator
```

parity flag → After an operation → 0015H → 0000 0000 0001 0101
PF = 0 → Odd number of 1's
PF = 1 → Even number of 1's

Auxiliary carry flag →



AF = 1, AF = 0

Zero flag → Z = 1 ; content of Accumulator is zero
Z = 0 , content of Accumulator is not zero

Sign flag → 1000 1111 (most significant bit)
MSB LSB

S = 1 , S = 0
(-Ve) (+Ve)

overflow flag :-

```
  7FH   (+127)
  01H   (+1)
 ----   ----
  80H   -128      Over = 1.
```

+Ve 0111 1111
change of
-Ve (number) 1000 0000

(7)

UNIT-1

8086 Architecture

Introduction to microprocessors.

A Microprocessors is a computer processor which incorporates the functions of a computer's central processing unit (CPU) on a single integrated circuit (IC), or at most a few integrated circuits.

The microprocessor is a multipurpose, clock driven, Register based, digital-integrated circuit which accepts binary data as input, processes it according to instructions stored in its memory, and provides results as output.

Microprocessors contain both combinational logic and sequential digital logic. Microprocessors operate on numbers and symbols represented in the binary numeral system.

Generation of Microprocessors:-

1. Intel 4004 (1971).
   - -> 4-bit microprocessor
   - -> 4 KB main memory
   - -> 45 instructions
   - -> PMOS Technology
   - -> was first programmable device which was used in calculators.

② INTEL 8008 (1972) :-

→ 8-bit version of 4004
→ 16 KB main memory
→ 48 instructions
→ pmos technology.
→ slow.

③ INTEL 8080 (1973) :-

→ 8-bit microprocessor
→ 64 kB main memory
→ 2 microseconds clock cycle time
→ 500,000 instructions/sec.
→ 10 X faster than 8008
→ N mos Technology
→ Drawback was that it needed three power supplies.
→ Small computers (micro computers) were designed in mid 1970's using 8080 as CPU.

④ INTEL 8086/8088 :-

→ year of introduction 1978 for 8086 and 1979 for 8088.
→ 16-bit Microprocessors.
→ Data bus width of 8086 is 16 bit and 8-bit for 8088
→ 1MB main memory.

→ 400 nanoseconds clock time

→ 6 byte instruction cache for 8086 and 4 byte for 8088

→ In 1981 IBM decided to use 8088 in its personal computer.

⑤ INTEL 80186 (1982)

→ 16-bit microprocessor - upgraded version of 8086

→ 1 MB main memory.

→ contained special h/w like programmable counters, interrupt controller etc.

→ Never used in the PC.

→ But was ideals for systems that required a minimum of hardware.

⑥ INTEL 80286 (1983):-

→ INTEL

→ 16-bit high performance microprocessor with memory management & protection.

→ 16 MB main memory

→ Few additional instructions to handle extra 15 MB

→ Instruction execution time is as little as 250 ns.

→ Concentrates on the features needed to implement Multitasking.

→ Intel 80386 (1986)

→ Intel 80486 (1989)

→ Pentium (1993)

→ Pentium pro (1995)

→ Pentium ii (1997)

→ Pentium iii (1999)

→ Pentium iv (2002)

→ Latest is Intel iq processor.

⇒ 8086 Architecture:-

→ The 8086 is mainly divided into mainly two blocks

① Execution Unit (EU)

② Bus Interface (BIU)

→ Dividing the work b/w these two will speedup the processing

① Execution Unit (EU):-

→ the Execution unit tells the BIU where to fetch instructions or data from

→ decodes instruction and

→ executes instructions

→ the Execution unit contains:-

↓

→ The Execution unit contains:

① control Circuitry
② ALU
③ FLAGS
④ General purpose Registers
⑤ pointer and Index Registers.

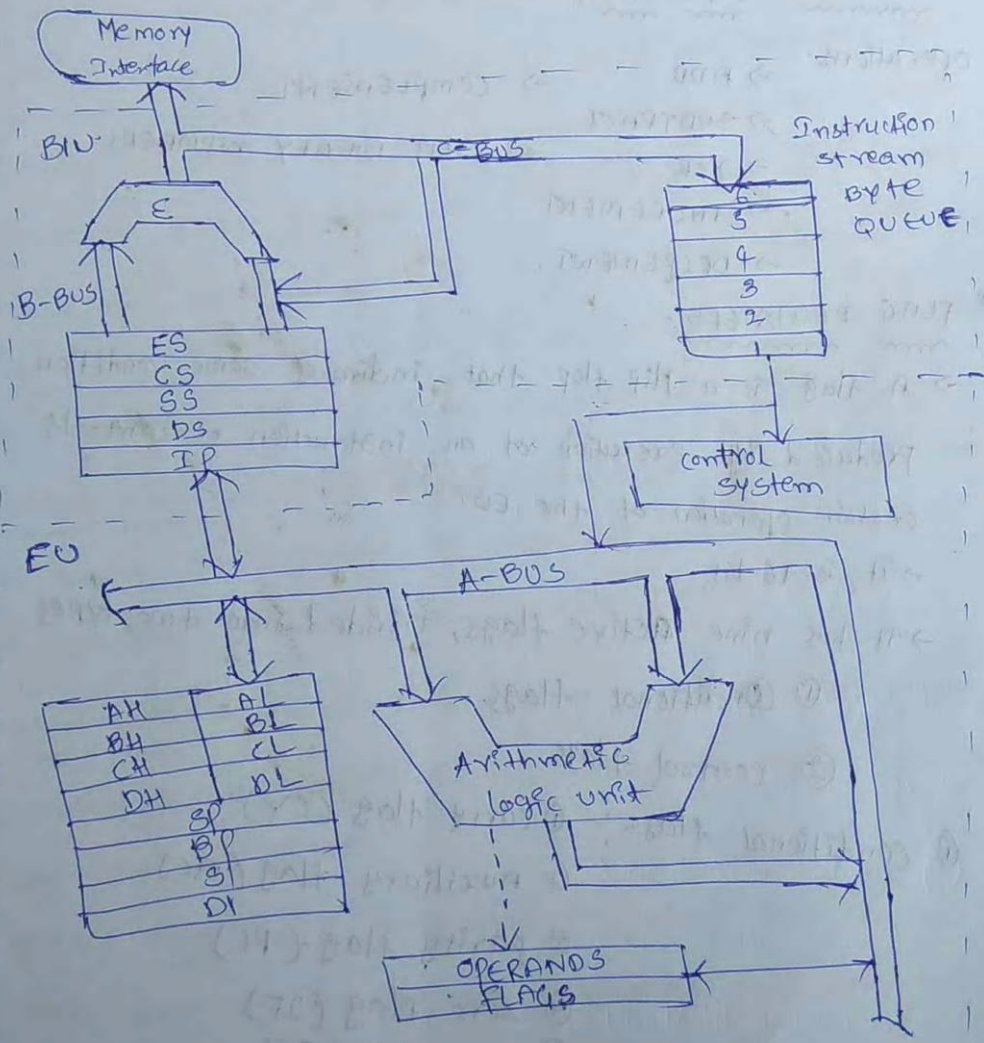


fig:- Architecture of 8086

① Control Circuitry :-

→ It directs internal operations.

→ A decoder in the EU Translates instruction teched from memory into series of actions which the EU carries out.

② Arithmetic Logic Unit :- 16 - bit ALU used to carry the operations.

- → ADD
- → SUBTRACT
- → XOR
- → INCREMENT
- → DECREMENT
- → COMPLEMENT
- → SHIT BINARY NUMBERS

③ FLAG REGISTERS :-

→ A flag is a flip flop that indicates some condition produced by execution of an instruction or controls certain operation of the EU.

→ It is 16 bit

→ It has nine active flags, divided into two types

① Conditional flags.

② control flags.

① Conditional flags :- ① carry flag (CY)

② Auxiliary flag (AC)

③ parity flag (PF)

④ Zero flag (ZF)

⑤ sign Flag (SF)

① carry flag (CY) :- This flag indicates an overflow condition for unsigned integer arithmetic. It is also used in multiple - precision arithmetic.

⑤ Auxiliary Flag (AC):- It 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.

⑥ 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.

⑦ zero Flag (ZF):- It is set; if the result of arithmetic or logical operation is zero else it is reset.

⑧ sign Flag (SF): In sign magnitude format the sign of number is indicated by MBS bit. If the result of operation is negative, sign flag is set.

② 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.

Flag Register :-

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	OF	DF	IF	TF	SF	ZF	X	AC	X	PF	X	CY

X = undefined

SF = sign flag → when S=1 if MSB of ALU result = 1

ZF = Zero flag → when Z=1 if ALU result = 00H

PF = parity flag, when even P=1 if ALU result is even parity

CY = carry flag, CY=1 if carry (or) Barrow occurs during addition & subtraction.

AC = Auxilary carry → carry from lower to upper nibble (or) Barrow from upper to lower nibble.

Ex:- ① Hex decimal

BC → 11 12 → 1011 1100

DE → 13 14 → 1101 1110

1 1 1 1 1 1 1 1

① 9A ① 1001 1010

26 - 16 = 10

25 - 16 = 9

S = 1

Z = 0

P = 1

CY = 1

AC = 1

Ex:- ① BC D7 D6 D5 D4 D3 D2 D1 D0 → 1 0 1 1 1 1 0 0

DE → 1 1 0 1 1 1 1 0

1 1 1 1 1 1 1 1

① 9A ① 1 1 1 1 0 1 0

1 0 0 1 1 0 1 0

carry 9A

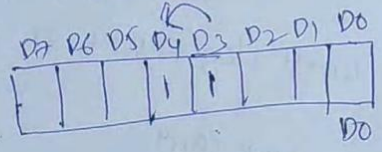
Ex ②

carry flag → If 1 → carry generated
0 → carry Not generated

```
  FFFFH
+ 0 001H
-------
 (1)0000
  Accumulator
```

parity flag → After an operation. 0015 H → 0000 0000 0001 0101
PF = 0 → Odd number of 1's
PF = 1 → Even number of 1's

Auxiliary carry flag →



AF = 1, AF = 0

Zero flag → Z = 1 ; Content of Accumulator is zero
Z = 0 , Content of Accumulator is not zero

Sign flag → 1000 1111 (most significant bit)
MSB LSB
S = 1 , S = 0
(-Ve) (+Ve)

overflow flag :-

```
  7FH   (+127)
  01H   (+1)
 ------
  80H   -128
```

Over = 1.

+ve 0111 1111
change to
+ve (number) 1000 0000

7/2

→① Trap flag → debugging

T = 1 , single step

T = 0 , debugging mode is off

② Interrupt flag → I = 1 , INTR (we want generated Interrupt
But I = 1 else.

I = 0 , INTR

③ Direction flag → SI , DI

D = 1 decrement by 1

D = 0 Incremented by 0

7/3

→ Interrupt flag :- 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 :- 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.

⇒ General Purpose Registers:-

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

① AX Registers : 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 operation, rotate and string Manipulation.

② BX Registers :- this register is mainly used as a base register. It holds the starting base

Segment Registers:-

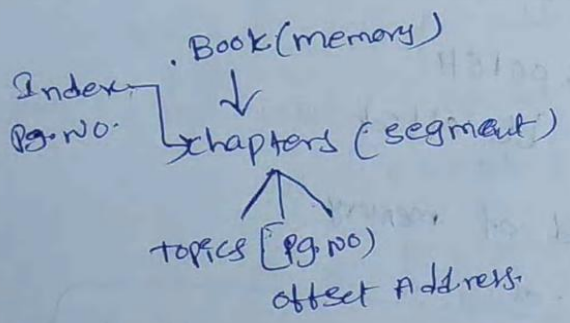
Definition:- generates memory addresses.

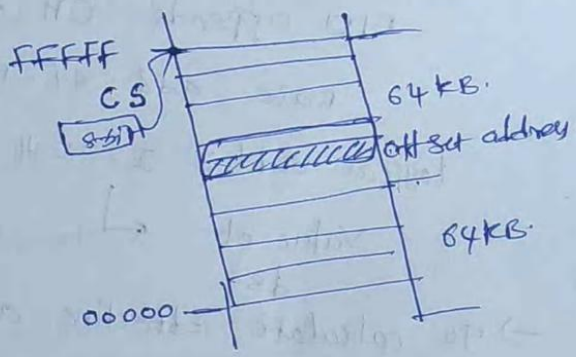
→ CS (Code segment): holds the code (programs & procedures)

→ DS (Data segment): contains most data used by a program

→ ES (Extra segment): Additional Data segment (contents of other Registers.)

→ SS (stack segment): defines area of memory used for the stack.





* PA = Segment × 10H + offset Address.

⇒ Memory segmentation:-

→ total memory size is divided into segments of various size.

↓

Just an area on memory

→ In memory data is stored as bytes.

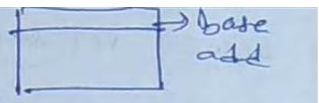
→ 8086 has 20 lines address bus.

data bus = 16
addr bus = 20

→ with 20 add lines, the memory can be addressed as 2^{20} bytes = 1MB.

from 00000H to FFFFFH

Physical Address of 8086:- ↓



→ Each reg stores the base add (Starting Address) because segment segment register cannot store 20 bits.

(*) How is 20 bit address obtained if there are only 16-bit reg. ?

Ex:- DS : 2222H = 16 bit Data.

BIU appends 0H to LSB of add

Base add. of DS = 22220H = 20 bit add.

logical add : 2222H : 0016H

Value of ds ↵ LL offset

→ To calculate Effective add of memory.

BIU uses the formula, as

Effective address = Starting add of segment + offset

→ The content of following registers are

CS = 1111H DS = 1678H ES = 1298H SS = 6784H

IP = 6721H BX = 7865H DI = 1235H, BP = 7821H

Q. CS, DS, ES & SS.

P.A of C.S ⇒ Seg add of CS X 10H + offset add

CS = 1111 X 10H + 6721H

= 17831H.

→ L.A of C.S = CS:IP

L.A ⇒ generate by

= 1111 H : 6721 H

② Physical add of ES = 1298 H × 10 + 1235 H

= 12980 + 1235

= 13BB5 H

12980
1235
13BB5

logical add = ES : DI

= 1298 H : 1235 H.

① Given a physical address and offset address calculate corresponding segment address of ES?

Ans: PA = 13BB5 H offset add = 1235 H

PA = ES × 10h + offset

ES = (PA − offset) / 10h = (13BB5H − 1235H) / 10H

= 12980 / 10 = 1298H

ES = 1298H.

② Calculate the P.A for following registers.

SS : 3864H SP : 1735H , BP : 4826H

⇒ The address calculate when sp is taken as offset gives the starting address of stack

⇒ the addresses when 'sp' is taken as offset denotes the memory location when the top of stack lies.

case ① : SS × 10h + SP

3864 × 10 + 1735

= 37A35H (20 bits)

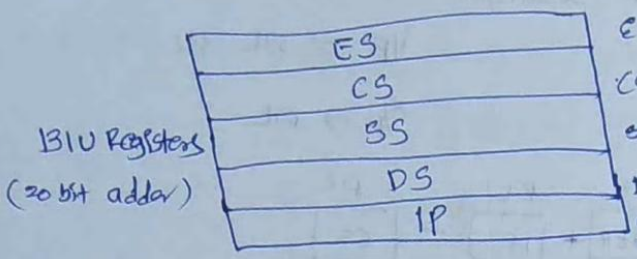
case ② : SS × 10h + BP

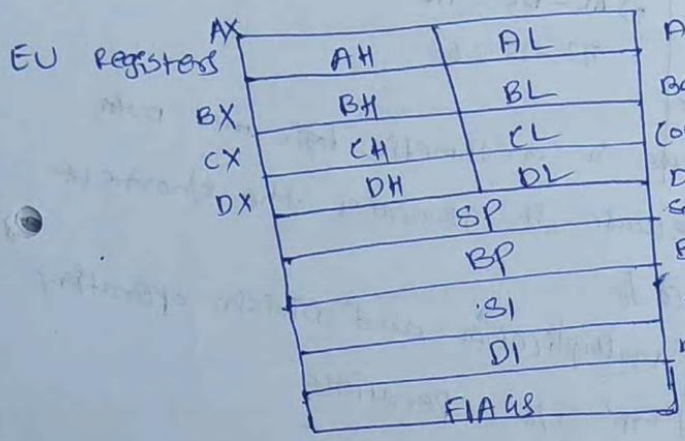
3864 × 10 + 4826

= 41226H. (20-bits)

10/1

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)





Ex:- 1) Normally used for storing temporary results.

2) Each of the registers is 16 bits wide (AX, BX, CX, DX)

3) Can be accessed as either 16 or 8 bits AX, AH, AL.

(1) Aim :- write a execute the an Assembly language programming to find the two 8-bit addition.

I/P → AL, BL

O/P → AL.

```
→ MOV AL, 32H      (0011 0010)
→ MOV BL, 42H
→ ADD AL, BL
  INT 03H
    ↓
  stoping the programing
```

AL	BL	AL
32H	+ 42H	= 74H

⇒ AL ← AL + BL

8. Write a [illegible] to find the two 8-bit substraction.

i/p → AL, BL

o/p → AL

```
MOV AL, 78H
MOV BL, 15H
SUB AL, BL
INT 03H
```

AL		BL		AL
78H	+	15	=	63

⇒ AL − BL = AL

78 − 15 = 63.

→ AX:- Accumulator Register

→ preferred register to use in arithmetic, logic and data transfer instructions because it generates the shortest Machine language code.

→ must be used in multiplication and division operations

→ must also be used in I/O operations;

BX:- Base register

- Also serves as an address register.

CX:→ Count register

→ used as a loop counter

→ used in shift and rotate operations

Ex:

```
    MOV AL, 32H
    MOV BL, 42H
    MOV CL, 02H
L1: ADD AL, BL
    LOOP L1
```

AL = 32H

BL = 42H

CL = 02H → count = 01 → 00

L1 is lable.

→ the condition is satisfied control Jump into (next Instruction,) specified lable (L1)

Instruction set of 8086:-

→ 8086 MPU supports 8 types of Instruction.

1. Data Transfer Instruction
2. Arithmetic Instruction
3. Bit Manipulation Instruction
4. String Instruction
5. program execution Transfer instruction (branch & loop Inst)
6. Processor control instruction.
7. Iteration control Instruction
8. Interrupt Instruction.

① **Data Transfer Instruction**:- These Instruction are used to Transfer the data from the source operand to the destination operands.

→ following are the list of inst under this group.

1. Instruction to Transfer a word

   ex:- MOV, PUSH, POP, XCHG. etc.

2. Instruction to Transfer the address

   Ex:- LEA, LDS, LES.

3. Instruction for input & output port

   Ex:- IN, OUT.

4. Instruction to Transfer flag reg

   Ex:- LAHF, SAHF, PUSHF, POPF

⑦ Iteration control Instruction :- these instructions are used to execute the given instruction for number of times.

① LOOP - used to loop a group of instructions until the condition satisfies i.e $CX = 0$.

② LOOPE / LOOPZ - used to loop a group of instructions till it satisfies $ZF = 1$ & $CX = 0$

③ LOOPNE / LOOPNZ - used to loop a group of instructions till it satisfies $ZF = 0$ & $CX = 0$.

④ JCXZ - used jump to the provided address if $CX = 0$.

⑧ Interrupt Instructions :- these instructions are used to call the interrupt during program execution and calling service specified in the program.

① INT - used to interrupt the program during execution

② INTO :- used to interrupt the program during execution if $OF = 1$.

③ IRET - used to return from interrupt service to the main program.

② Arithmetic Instructions:- There instruction are used to perform arithmetic operation addition, subtraction, Multiplication, division etc.

① Instruction to perform Addition

ex: ADD, ADC, INC, AAA etc

② Instruction to perform multiplication

ex:- MUL, IMUL, AAM.
↳ signed multiplication.

③ Instruction to perform subtraction

ex:- SUB, SBB, DEC, AAS. etc.

④ Instruction to perform division.

ex:- DIV, IDIV, AAD.

③ Bit Manipulation Instructions:- these instruction are used to perform operations where Data bits are involved in operations like logical shift etc.

① Instructions to perform logical operations

ex:- NOT, AND, XOR, TEST.

② Instructions to perform shift operations

ex:- SHL/SAL, SHR, SAR.

③ Instructions to perform rotate operations.

ex: ROL, ROR, RCR, RCL.

④ String Instructions:- String is a group of bytes/words and their memory is always allocated in a sequential order

① REP,
② REPE/REPZ
③ REPNE/REPNZ
④ MOVS/MOVSB/MOVSW
⑤ COMS/COMPSB/COMPSW
⑥ INS/INSB/INSW.
⑦ OUTS/OUTSB/OUTSW.

⑤ Program Execution Transfer Instructions:-

→ these instructions are used to transfer/branch the instructions during an execution.

In includes the following instructions:-

① Instructions to transfer the instruction during an execution without any conditions.

Ex:- CALL, RET, JMP.

② Instructions to transfer the instruction during an exeut with some conditions.

Ex:- JC, JE, JZ, JNC, JS, JO etc.

⑥ Processor control Instructions:- these instructions are used to control the processor action by setting/ resetting the flag values.

STC - used to set carry flag CF to 1.

CLC - used to clear/reset carry flag CF to 0.

CMC - used to put complement of the state of carry flag CF

STD:- used to set the directions flag DF to 1.

CLD:- used to clean/reset the direction flag DF to 0

STI:- used to set the interrupt enable flag to 1, & Enable INTR input.

CLI:- used to clear the interrupt enable flag to 0 & Disable INTR input.

* Hardware Interrupt :- The interrupts initiated by external hardware by sending an appropriate signal to the interrupt pin of the processor is called Hardware Interrupt.

ex:- NMI, INTR

② Vectored and Non-vectored Interrupts

* Vectored Interrupts :- It a program control automatically branches to a specific address when an interrupt signal is accepted by the processor. Such an interrupt is called vectored interrupt.

* Non-vectored Interrupts :- In non-vectored interrupts the interrupting device should supply the address of the ISR to be executed in response to the interrupt.

Maskable and Non-maskable :-

Maskable Interrupt :- The interrupt which can be disable (reject), then it is called Maskable Interrupt.

ex:- INTR pin of 8086

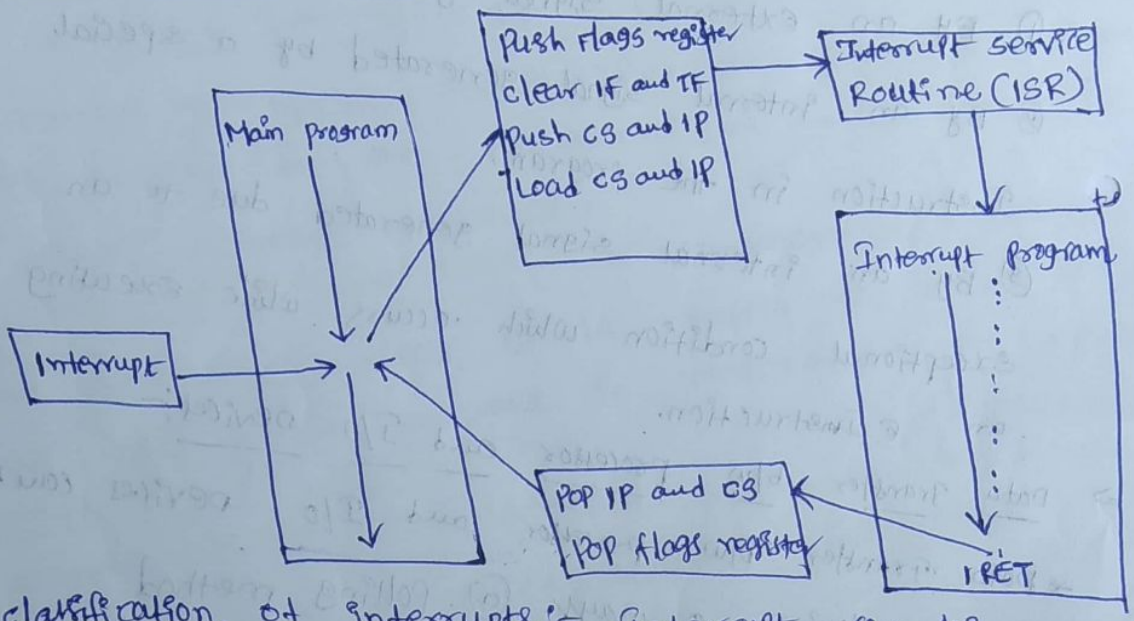
Non-maskable interrupt :- the interrupt which can not be rejected or disable is called non-maskable Interrupt.

ex:- NMI pin of 8086.

→ Upon receiving an interrupt signal, the microprocessor holds whatever it is doing and serves the corresponding device.

→ The program associated with the interrupt is called the interrupt service routine (ISR).

⇒ What happened when Interrupt is occurred.



⇒ Classification of interrupts:- Interrupt can be classified as three types.

① Software and Hardware

② Vectored & Non-Vectored

③ Maskable & Non-maskable.

① Software and Hardware:-

* Software Interrupts:- the interrupts generated by special instruction are called software interrupts and they are used to implement system services/ calls.

Interrupts of 8086 :-

* "the process of Interrupting the Normal execution of the program to carry out the specific Task is called as an Interrupt".

→ Processor can be interrupted by following ways :-

① By an external signal generated by a peripheral

② By an internal signal generated by a special Instruction in the program.

③ By an internal signal generated due to an exceptional condition which occurs while executing an Instruction.

→ Data Transfer b/w processor and I/O devices :-

* Data Transfer b/w processor and I/O devices can be done in following 2 ways ⓐ Polling method

ⓑ Interrupt Method.

ⓐ Polling method :- In Polling Method, the microprocessor continuously monitors the status of a given device, when the status condition is met, it performs the service. After that, it moves on to the next device until each one is serviced.

ⓑ Interrupt Method :- In Interrupt Method, whenever any device needs service from microprocessor, the device notifies to processor by sending signal (called Interrupt).

$\overline{BHE}$	A0	Indication	Data lines used
0	0	whole word	→ D0 – D15
0	1	upper byte from (or) to odd addresses	→ D8 – D15
1	0	lower byte from (or) to Even addresses	→ D7 – D0
1	1	None	→ X

Table of $\overline{BHE}$, A0

→ Read/write a byte from an Even address 10 – D7 D0

→ Read/write a byte from an odd address 01 → D15–D8

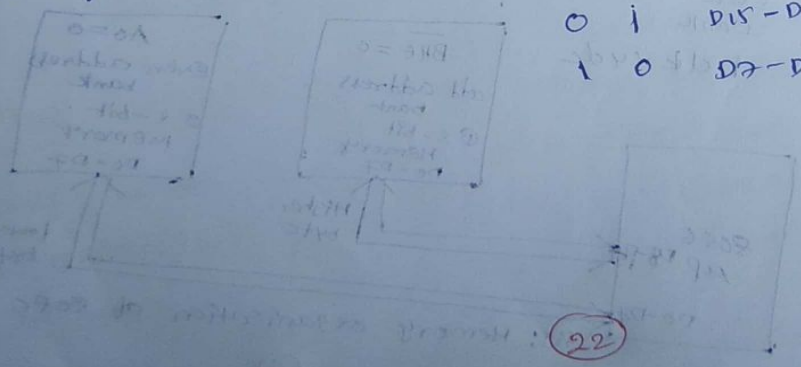
→ Read/write a word from an Even address 00 → D15–D0

→ None — 11 · X

→ Read/write a word from an odd address

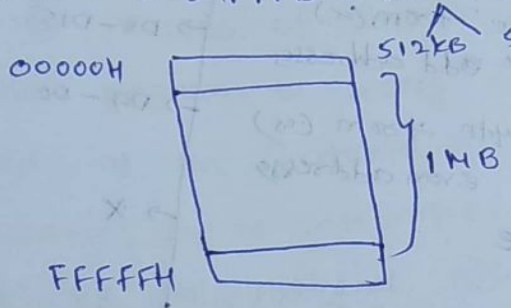
0 1 D15 – D8

1 0 D7 – D0.



Physical Memory Organization

→ 1MB = 1KB × 1KB = 1024 KB



→ 8086 'mp' is 16-bit.

→ word (two bytes).

① Both the bytes may be data

② Both the bytes may be opcodes

③ one of the byte may be Opcode while other may be data

→ 16-bit

→ 2 clk cycles.

→ odd bank / Even bank

→ 2 clk cycle. / 1 clk cycle

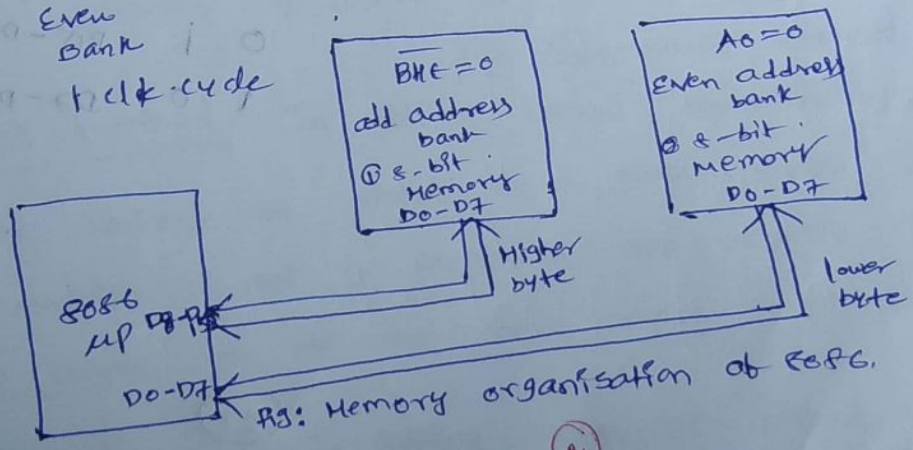


Fig: Memory organisation of 8086.

→ that operand will be moved into AX Register.

⑧ Based Relative Addressing mode:-

MOV AX, 12H [BX]

MOV AX, 1234H [BX]

Here 1234H or 12H is the Displacement and it will be added with BX register content to generate the address.

→ this address will contain the Data (Operand), that will moved in to Destination register AX.

⑨ Indexed Relative Addressing mode:-

MOV AX, 12H [SI]

MOV AX, 1234H [SI].

→ Here 1234H (or) 12H is the Displacement and it will be added with SI register content to generate the address.

→ this address will contain the Data (Operand), that will Moved in to Destination register AX.

⑩ Based Indexed Relative Addressing mode:-

MOV AX, 1234H [BX] [SI]

→ Here 1234H is the 16-bit Displacement, that will be added with BX + SI content to generate the address

→ that address will contain the Data, that Data is moved in to AX Register.

⑤ Based Addressing mode :- In this Addressing mode the offset add of operand is given by the sum of contents of BX/BP reg and 8/16-bit displacement.

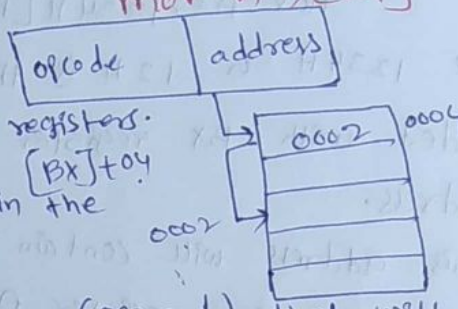
MOV DX, [BX+04]

MOV AX, [BX]

MOV AX, [BP].

→ Here in this addressing mode we will use BX or BP as source registers.

→ the content of BX or BP will contain the [BX]+04 address.



→ In that address we have the Data (operand), that will be moved in to AX (Destination) Register.

⑥ Indexed Addressing mode :-

MOV AX, [SI]

MOV AX, [DI].

→ Here in this addressing mode we will use SI or DI Registers as source registers.

→ the content of SI or DI will contain the address.

→ In that address we have have the Data (operand), that will be moved in to AX Register.

⑦ Based Indexed addressing mode :-

MOV AX, [BX][SI]

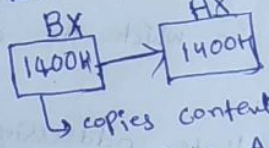
→ In this addressing Mode the content of BX and SI will be added.

→ then that content is a address, which contains the Data (Operand)

(2) Register Addressing mode:- It means that the register is the source of an operand for instruction.

Ex:- MOV AX, BX

MOV AL, BL.

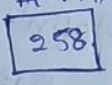


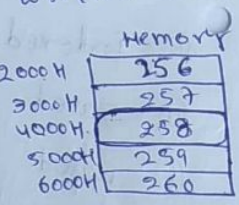
→ Here source Register is BX or BL.
→ Destination Register is AX or AL
→ the BL (8-bit) or BX (16-bit) will contain the operand
→ Instruction is "mov".

(3) Direct Addressing mode:- The addressing mode in which the effective address of memory location is written directly in the instruction.

MOV AX, [4000H]

MOV AL, [4000H]



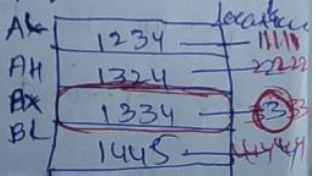


→ Here [4000H] is the Memory location. In that Memory location we have the data (operand).
→ when 4000H is brackets then its become an address.

(4) Indirect Addressing mode:- this addressing mode allow data to be addressed at any memory be through an offset addressing held in any of instruction the following BP, BX, DI & SI.

MOV AX, [BX];

→ Here the content of the BX register is the address
→ that address will contain the data (operand)
→ that operand will be moved in to AX Register.



Addressing Modes of 8086

→ The way in which the operands (Data) are accessed by an Instruction.

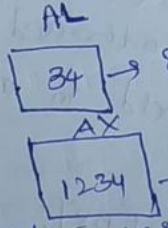
1. Immediate addressing mode
2. Register Addressing mode
3. Direct Addressing mode
4. In Direct Addressing mode.
5. Based Addressing mode
6. Indexed Addressing mode
7. Based Indexed Addressing mode.
8. Based Relative Addressing mode
9. Indexed Relative Addressing mode
10. Based indexed Relative Addressing mode.

① Immediate Addressing mode :-

→ The addressing mode in which the Data operand is part of the Instruction itself.

Ex:- MOV AL, 34H

MOV AX, 1234H



→ Here MOV is the Data Transfer Instruction.

→ AL (or) AX is the Destination Register

→ 1234H is the Hexadecimal Data (Operand)

→ the operand will be moved in to destination Register.

Signal Descriptions of 8086 :-

1. GND :- Ground pin it connects larger currents to the ground.

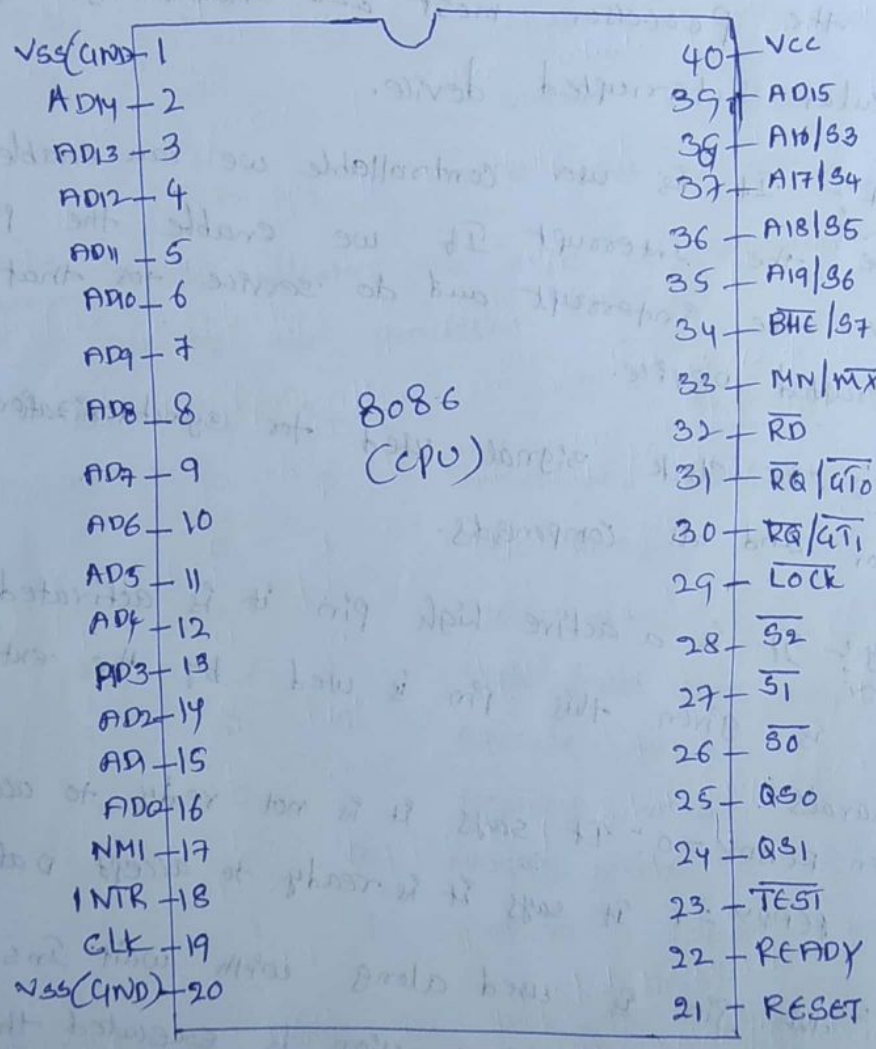


fig:- pin diagram of 8086.

2. AD14 - AD0 } :- These lines are called Multiplexed lines
AD15.

So there are total 16 Multiplexed lines. In this the address is available on the address lines during T_1 and the Data is available on

⇒ GND :- Two ground pins are there, in order to reduce the power dissipation in the processor with in the processor.

⇒ Function of PINS in minimum mode :-

→ $M/\overline{IO}$ ⇒ ① $M/\overline{IO}$ = 1, processor performs the memory read/write operation.

→ $\overline{WR}$

→ $DT/\overline{R}$ ② $M/\overline{IO}$ = 0, processor performs the I/O read/write operation.

→ $\overline{DEN}$

→ ALE

→ $\overline{INTA}$ ⇒ $\overline{WR}$: this indicates the 8086 microprocessor sending the data to memory or I/O devices.

→ HOLD

→ HLDA. ⇒ $DT/\overline{R}$:- This signal indicates that the 8086 Data bus transmitting or receiving the data.

* If it is logic 1 - Transmiting.
* If it is logic 0 - receiving.

→ this signals are used to control the data flow direction in external data bus buffers.

⇒ $\overline{DEN}$:- this signal is used to enable the external Data bus buffer.

* for logic-0 the data is Transferred on data bus.
* for logic-1 no dataflow on the Data bus.

→ ALE :- this signal is high, it indicates that the 8086 multiplexed address/data bus (AD15-AD0) and multiplexed address/status (A19/S6 - A16/S3) contain address, which can be Either address of memory or IO device.

MN/$\overline{MX}$:- this pin is used to select the minimum mode or maximum mode of the 8086mp.

→ MN (+5V) - single processor

→ $\overline{MX}$ (ground) - multi processor

$\overline{RD}$:- whenever the read signal is logic 0 the processor reads the data from the memory or from I/O devices through data lines.

READY :- this input is used to insert the wait state into timing cycle of the 8086.

→ If the ready pin is at logic 1, it has no effect on the operation.

→ If it is logic 0 the 8086 enters into the wait state and remains idle.

→ this pin is used to synchronize the slowly peripheral devices with processor.

RESET :- this input causes the 8086 to RESET

→ It held logic 1 for minimum of 4 clocking periods.

→ when ever 8086 reset, the CS and IP initialized to FFFFH and 0000H.

→ All other register are initialized to 0000H

→ this causes the 8086 has execute the program from the location FFFF0H.

Function of status bits S3 - S4 :-

S4	S3	Segment accessed
0	0	ES
0	1	SS
1	0	CS
1	1	DS

NMI :- → which is a hardware interrupt
→ It can not be disabled by the software
→ It is positive edge triggered interrupt
→ when it occurs type 2 interrupt occurs in the 8086.
→ which is generally is in power failure systems.

INTR :- → this interrupt is level triggered hardware interrupt.
→ which depends upon the status of IF.
→ when IF = 1, INTR = 1 the 8086 gets interrupted
→ when IF = 0, INTR = 1, the INTR is Disabled.

CLK :- the clock signal must have the duty cycle of 33% to provide the proper internal timing for the 8086.
→ Its maximum frequency is 5, 8 and 10 MHz.

Vcc :- this power supply pin provides the +5V DC.
The variation allowed in the power supply input is 10%.

BHE/S7 :- this bus high enable pin is used in the 8086 to Enable the most significant data bus (D15 - D8) during read or write operation.

⇒ Address and Data lines (AD0-AD15)

→ these pins are act as the multiplexed address and Data bus of the Microprocessor.

→ when ever ALE pin is high then pins carry the address, and when ALE pin is low these pins carry the Data.

→ using two external octal latches along with ALE signal, these pins can be demultiplexed into the address bus (A15-A0) and Data bus (D15-D0).

⇒ A19/S6 - A16/S3 (Address / Status) lines.

→ these pins are multiplexed to provide the address signals A19-A16 and the status bits S6-S3.

→ when ALE = 1 these pins carry the address and when ALE = 0, they carry the status lines.

→ using one external latch along with ALE signal these lines can be demultiplexed in to address lines (A19-A16) and the status lines (S6-S3).

→ S3-S4 indicates the which segment is accessed by the 8086mp during the current bus cycle.

→ S5 reflects IF flag register.

→ S6 is always zero.

→ S7 is logic 1

Pin Diagram of 8086 Microprocessor :-

→ Key points :-

- MN/$\overline{MX}$:- MAX mode and min. mode.

→ In Pin 31 to 24 Min mode is : HOLD, HLDA, $\overline{WR}$, M/$\overline{IO}$, DT/$\overline{R}$, $\overline{DEN}$, ALE, $\overline{INTA}$

→ In Pin 24 to 31 MAX mode is $\overline{RQ}/\overline{GT_0}$, $\overline{RQ}/\overline{GT_1}$, $\overline{LOCK}$, $\overline{S_2}$, $\overline{S_1}$, $\overline{S_0}$, QS0, QS1.

→ A system having only one processor without any co-processor is set be configured in Min mode.

→ Pin 33 MN/$\overline{MX}$ is connected Vcc than 8086 enter to Min mode.

→ A system having an option multiple processor with are without co-processor is set be configured.

→ Pin 33 MN/$\overline{MX}$ is connected GND

→ "—" Active low. (0)

→ without — Active High (Logic 1).

Comman Pins of 8086 :-

1. AD15 – AD0
2. A19/S6 – A16/S3
3. NMI
4. INTR
5. CLK
6. Vcc
7. $\overline{BHE}$/S7
8. MN/$\overline{MX}$
9. $\overline{RD}$
10. $\overline{TEST}$.
11. READY.
12. RESET
13. GND

14. A19/S6 - A16/S3 :- address / status Bus (multiplexed)

Memory address A19 - A16 status bits S6 - S3.

S6 :- always remain a logic 0

S5 :- indicate condition of flag bits (IF)

15. S4, S3 :- these two pins represent which segment is currently the processor accessing.

w.k.t there are totally 4 segments.

	S4	S3
① Extra segment	0	0
② stack segment	0	1
③ code (or) No segment	1	0
④ data segment	1	1

16. RD :- Read signal is active low pin these are activated when logic '0' is given.

when it indicates RD = 0, the processor is performing a read operation from memory or I/O device.

17. Vcc :- (power supply) :- we have a supply pin and the maximum voltage we can give at here is +5.0 V.

system can share the Bus.

11. $\overline{R_1}/\overline{GT_1}/\overline{R_0}/\overline{GT_0}$ (Request/Grant):- In case of maximum mode if multiple processors are connected If the other processor wants to access the system Bus then it is going to Request processor which is currently using the system Bus with the help of the pins.

12. MN/$\overline{MX}$:- selecting the minimum mode or maximum mode. If this particular pin is connected to Vcc or programmed to 1 than it means that we are operating the processor in minimum mode If the pin is connected to Max mode then the pin is connected to ground.

13. $\overline{BHE}/S_7$:- Bus high Enable

there two pins representing what type of data processor accessing.

BHE	A0	Indication:-
0	0	whole word
0	1	upper data / from odd Bank
1	0	lower data / from even Bank
1	1	None

8. QS1 & QS0 :- It gives the status of Instruction Queue.

QS1	QS0	Meaning:
0	0	No operation
0	1	first byte of opcode from Queue
1	0	Queue is cleared
1	1	Next byte of opcode from Queue

9. S0, S1, S2 :- this indicates the type of operation carried out by the processor

S_0	S_1	S_2	Function
0	0	0	Interrupted Acknowledge
0	0	1	I/O Read
0	1	0	I/O write
0	1	1	Halt (Not doing anything)
1	0	0	Opcode fetch
1	0	1	Memory read
1	1	0	Memory write
1	1	1	passive (Ideal state).

10. $\overline{Lock}$:- It is going for lock the system bus from preventing others to access the Bus until the Instruction is executed.

If $\overline{Lock} = 0$, then the Bus is locked so that no system can share the bus.

$\overline{Lock} = 1$ than the Bus is not locked so that any

the Data lines during Time T_3, T_W and T_4.

3. NMI :- (Non maskable Interrupt) :- It is not user controlled In this when an Interrupt is generated the processor must and should go for that particular Interrupted device.

4. INTR :- It is user controllable we can Enable or Disable the Interrupt If we Enable the processor receives the Interrupt and do service for that Interrupted device.

5. CLK :- The Clock signal used for synchronization b/w processor and all components.

6. READY :- It is a active high pin it is activated when logic 1 is given this pin is used by the external peripharals.

when READY = 0 : It says it is not ready to accept data

READY = 1 It says it is ready to accept Data

7. Test :- this pin is used along with wait Instruction when ever the wait Instruction is executed the processor ramains at wait state for Idden state and the test pin goess high as soon it goes low and it will continute Its Execution.

INTA :- $\overline{INTA}$ is used to place the interrupt type or vector number in Data bus in response to the INTR interrupt.

HOLD :- The hold input requests a direct memory access (DMA) and is generated by the DMA controller
→ for logic -1 :- the 8086 completes the current execution and places the 3-busses in high impedance state.
→ for logic -0 : Execution is normal.

HLDA :- This indicates that the 8086 microprocessor Entered in to hold state and this pin is connected to the DMA controller.

Function of pins in maximum mode :-

→ $\overline{S_0}$, $\overline{S_1}$, $\overline{S_2}$
→ $\overline{LOCK}$
→ QS_1 and QS_0

$\overline{S_0}$ $\overline{S_1}$ $\overline{S_2}$:- status bits indicates the function of the current Bus cycle.

→ these signal are normally decided by the 8288 bus controller.

$\overline{S_0}$	$\overline{S_1}$	$\overline{S_2}$	Functions :-
0	0	0	Interrupted Acknowledge
0	0	1	I/O Read
0	1	0	I/O write
0	1	1	Halt (not doing anything)
1	0	0	opcode fetch
1	0	1	Memory read
1	1	0	Memory write
1	1	1	passive (Ideal State).

* write an ALP to Arrange the given set of 8-bit numbers into descending order.

30H, 10H, 50H, 20H, 70H.

count - 05H

Ex: Location for count: 2000H - count value - 05H

[3000H - 3004H] - 30H, 10H, 50H, 20H, 70H

Interation-1:-

30H 10 50 20 70 JGE - Jump grater than or Equal.

30H 10 50 20 70

30 50 10 20 70

30 50 20 10 70

30 50 20 70 10

Iteration-2:-

30 50 20 70 10

50 30 20 70 10

50 30 20 70 10

50 30 70 20 10

Iteration-3:-

50 30 70 20 10

50 30 70 20 10

50 70 30 20 10

Iteration-4:-

50 70 30 20 10

70 50 30 20 10

⇒ 3000H - 70H

3001H - 50H

3002H - 30H

3003H - 20H

3004H - 10H

Iteration-3:-

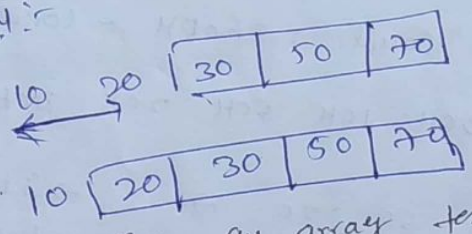
10 20 30 50 70

10 20 30 50 70

10 20 30 50 70

→ 3000H = 10H
→ 3001H = 20H
→ 3002H = 30H
→ 3003H = 50H
→ 3004H = 70H.

Iteration-4:-



⇒ Program for sorting an array for 8086 Microprocessor.

Assembly Language programming of 8086

① Write an ALP to Arrange the given set of 8-bit numbers into Ascending order.

count = 5

30H 10H 50H 20H 70H,

Count = 05H

Eg:- Location for count : 2000H - count value = 05H

[3000H - 3004H] - 30H 10H 50H 20H 70H,

Iteration - 1 :- 30 10 50 20 70 → JLE

Jump less than or Equal,

30 10 50 20 70

10 30 50 20 70 ①

10 30 50 20 70 ②

10 30 20 50 70 ③

10 30H 20 50 70 ④

→ when ever first byte is equal second byte than no need of exchange.

→ when ever first byte is grater than second byte exchange byte.

Iteration - 2 :-

10 30 20 50 70 ①

10 30 20 50 70 ②

10 20 30 50 70 ③

10 20 30 50 70 ④

⇒ Program to sort the given number in ascending order.

```
Data SEGMENT
ARR DB 30H 10H 50H 20H 40H.
Data ENDS.
CODE SEGMENT.
Assume CS: CODE , DS: Data
START:
        XOR AX, AX
        MOV BL, AL
        mov CL, AL
        MOV SI, 2000H
        MOV BL, [SI]
        DEC BL.
L3:     MOV CL, BL
        MOV SI, 3000H
L2:     MOV AL, [SI]
        COM AL, [SI+1]
        JLE L1
        XCHG AL, [SI+1]
        MOV [SI], AL
L1:     INC SI
        LOOP L2
        DEC BL
        JNZ L3
        INT 3.
CODE ENDS
END START
```

→ Multiplication and Division:

MUL reg 8.

MUL BL.

AH.AL = AL * BL

AH = AL * BL

MUL reg 16

MUL BX

DX AX ← AX * BX

MOV SI, 500

MOV DI, 600

mov AL, [500]

AL ← 03

mov BL, [SI+1]

MUL BL

mov [600], AX ⇔ mov [DI], AX

HLT 600 ← AL

601 ← AH

Multiplication

MOV AL, 06H

mov BL, 04H

MUL BL.

INT 3

Result = 18H

16/24

1-8

05	03
501	500

00	0F
601	600

DIV

mov AX, 44444H

mov BX, 0002H

DIV BX

INT 3

Division operation:

DIV BL

AL → Q

AH ← R

AX/BL

64	00	0E
502	501	500

8-bit no 16-bit

000E/04

mov SI, 500

mov AX, [SI]

AX ← 000E

mov BL, [SI+2]

mov DI, 600

DIV BL

mov [DI], AX

HLT.

601	600

AL ← 03 ← Q

DH ← 02 ← R

(6) MUL → TO signed

(7) DIV

(47)

→ Sum of N natural numbers:-

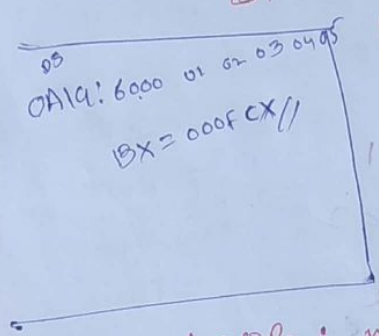
ADD.ASM
Link ADD.OBJ
CV ADD.EXE

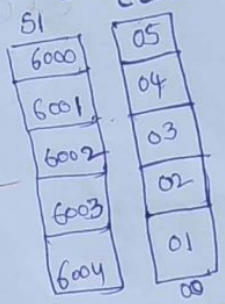
```
Assume DS: Data ; CS: Code
Data Segment
    TABADR  DW  6000H
    COUNT   DB  05H
Data Ends
Code segment
Start:
    MOV AX, Data
    MOV DS, AX
    MOV SI, TABADR
    MOV CL, COUNT
LOOP1: MOV AL, [SI]
    ADD BL, AL
    INC SI
    DEC CL
    JNZ LOOP1
    INT 03H
Code Ends
End start.
```

[Jump if not zero.]
When CL=0 the condition is false.

SI	AL
6000	01
6001	02
6002	03
6003	04
6004	05

0F





AL = 01
BL = 00
BL = 01 → BL+AL
01
BL = 02

MOV
ADD
INC
DEC
JNZ
INT

Mnemonics

Operands

AX, OA1A
DS, AX
SI, WORD PTR(9000)

⇒ Addition program:-

```
Assume cs: code, DS: Data
Data segment
    data1 dw 1234H
    Data2 dw 4321H
Data Ends
code segment
  start:
      MOV ax, Data
      mov DS, ax
      mov ax, data1
      mov bx, data2
      Add ax, bx
      Int 03h
code Ends
Ends start
```

⑤ Alignment directives :- EVEN

→ Align as even memory address. the directive EVEN is used to inform the assembler to increment the location counter to the next even memory address if it is not pointing to even memory location already

```
Data segment
    sum db 10
    even
    items dw 100 dup(?).
data ends.
```

→ ORG : Originate
→ ORG 1000.

④ Program end directive :-

END - END PROGRAM : the END directive is put after the last statement of a program to tell the assembler that this is the end of the program module.

```
code segment
_ _ _ _ _
_ _ _ _ _
ends
dat segment
_ _ _ _ _
_ _ _ _ _
data ends
end
```

⑤ value returning attribute directives :-

LENGTH : The Directive length informs the assembler about the number of elements in a data item such as an array.

Ex :- MOV AX, LENGTH ITEMS.

```
CODE SEGMENT
  -----
  -------
  -------
CODE ENDS.
```

ENDS. [End segment]:- this directive is used with the name of a segment to indicate the end of the logical segment. ENDS is used with the segment directive to bracket a logical segment containing instruction or data.

```
Data segment
  n1 db 05h
  n2 db 04h
  sum db ?
Data Ends.
```

ASSUME:- The assume directive is used to tell the assembler the name of the logical segment it should use for a specified segments. [cs, DS, ss, ES].

the statement:

```
ASSUME CS: code.
```

→ Tells the assembler that the instructions for a program are in a logical segment named code.

```
code segment
Assume CS: code, ds: data
Mov ax, Data
mov ds, ax
  - - - - -
code Ends
Data segment
  - - - -
Data Ends.
```

⑥ Procedure definition directives.

⑦ Macro definition directives.

⑧ Data control directives.

⑨ Header file inclusion directives.

① Data definition and storage allocation directives.

→ Data definition directives are used to define the program variables and allocate a specified amount of Memory to them.

→ they are of type BYTE, WORD, Double word, Quad word and Ten Byte and their size is bytes are 1, 2, 4, 8 and 10 respectively.

→ the Data definition directives are DB, DW, DD, DQ, DT.

DB = Define Byte - 1 byte
DW = Define word - 2 byte
DD = Define Double word - 4 byte
DQ = Define Quard byte - 8-byte
DT = Define Ten bytes - 10 bytes.

② Program Organization Directives:-

→ SEGMENT: the segment Directive is used to indicate the start of a logical segment. Preceding the segment Directive the name you want to give the segment.

↓.

Assembler Directives:- Assembler Directives are instructions entered into the source code along with the assembly language pseudo instructions (Assembly Directives) do not get Translated into object code, but are used as special instructions to the assembler to perform some special functions.

→ "The Directives control the generation of machine code and organizations of the program".

→ Assembly language programs are composed of two types of statements.

① the Instructions which can be translated to Machine code, by the assembler.

② the Directives that directs the assembler during the assembly process for which no machine code is generated.

Types of Assembler Directives:-

① Data definition and storage allocation directives.

② Program organization directives.

③ Alignment directives.

④ Program end directive

⑤ Value returning attribute directives.

QS0 and QS1 :- the queue status bits indicates the status of the internal instruction queue.

QS0	QS1	Function :-
0	0	No operation
0	1	First Byte of opcode
1	0	Queue is Empty.
1	1	Subsequent Byte of opcode

LOCK :- when it is logic low system bus can not be share by other co-processors.

$\overline{RQ}/\overline{GT}$ (Request/grant) :- these signals are as inputs and outputs.

→ other processors send request for the system bus an 8086 microprocessor sends the grant for that request using these signal.

=) program to sort the given number descending order

```
Data SEGMENT
ARR DB 30H 10H 50H 20H 70H
Data Ends.
CODE SEGMENT
    ASSUME CS: CODE
START:
        XOR AX, AX
        MOV BL, AL
        MOV CL, AL
        MOV SI, 2000H
        MOV BL, [SI]
        DEC BL
L3:     MOV CL, BL
        MOV SI, 3000H
L2:     MOV AL, [SI]
        CMP AL, [SI+1]
        JGE L4
        XCHG AL, [SI+1]
        MOV [SI], AL
L4:     INC SI
        LOOP L2
        DEC BL
        JNZ L3
        INT 03
CODE ENDS
END START.
```

Result:

=) 70 50 30 20 10

UNIT-II

Introduction to microcontrollers:-

Microprocessor:- The Microprocessors contain no RAM, no Rom and no I/O ports on the chip itself.

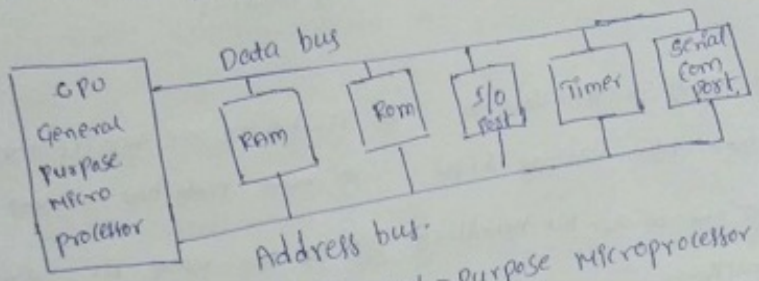


fig:- General-purpose Microprocessor system.

Microcontroller:- A microcontroller has a CPU (a µp) in addition to a fixed amount of Ram, Rom, I/O ports, and a timer all on a single chip.

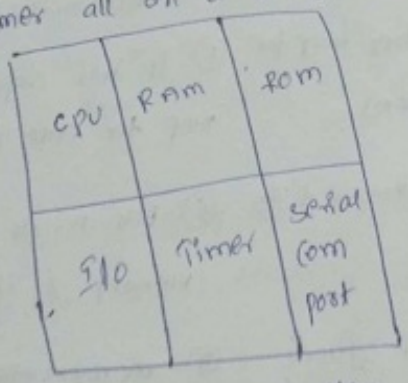
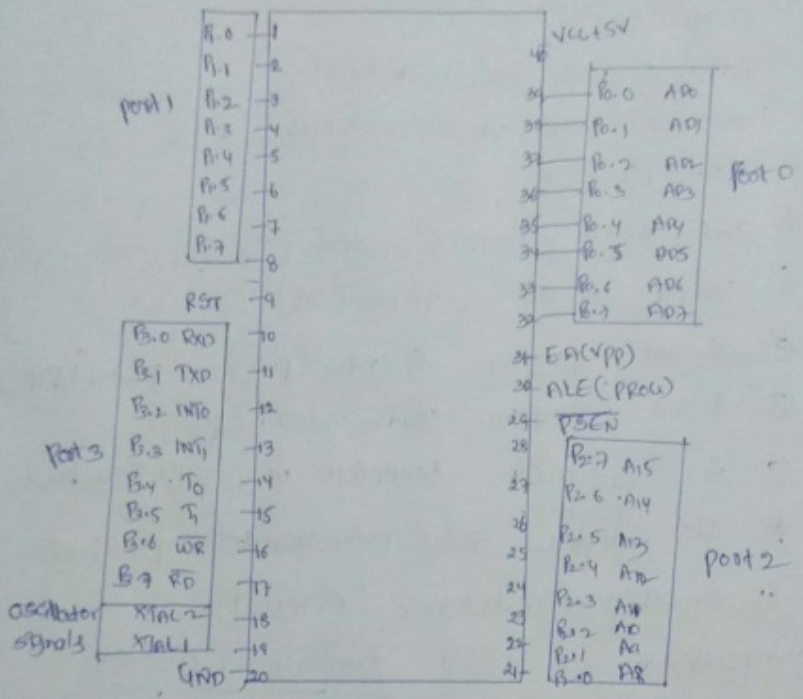


fig:- Microcontroller.

-> In 1981, Intel corporation introduced an 8-bit microcontroller called the 8051. this microcontroller had 128 bytes of RAM, 4K bytes of on chip Rom, two timers, one serial port, and four ports (each 8-bits wide) all on a single chip.

Microprocessor	microcontroller
① Microprocessor contains ALU, General purpose registers, stack, stack pointer, program counter, clock timing ckt & Interrupt ckt.	① microcontroller contains the circuitry of microprocessor & in addition it has built in Rom, Ram, I/O devices, timers & counters.
② It has many instructions to move data b/w Memory & CPU	② It has one or two instructions to move data b/w memory & CPU.
③ It has one or two bit handling instructions.	③ It has many bit handling Instructions. Ex: CLR, SETB P1.0 etc.
④ Less No. of pins are Multifunctioned	④ More No. of pins are Multifunctioned.
⑤ It has single Memory map for Data & code (program)	⑤ It has separate Memory map for Data & code (program)
⑥ Access time for Memory & I/O devices are more.	⑥ Less Access time for built-in Memory & I/O devices.
⑦ Microprocessor based system requires more hardware	⑦ Microcontroller based system requires less hardware reducing PCB size & increasing the reliability.
⑧ Microprocessor based system is more flexible in design point of view.	⑧ less flexible in design point of view.

8051 Pin diagram:



Pins 1-8 :- port 1 :- Each of these pins can be configured as I/p or o/p pins.

Pin 9 : RST (Reset) :- It is a active high signal, when a pulse (square wave) is applied to this pin, microcontroller will terminate all its activities & Reset.

Program counter is loaded with 0000.

* List the specific features of 8051 microcontroller. (3)

-> the salient features of 8051 microcontroller are.

① 8-bit CPU
② Internal ROM of 4-k bytes.
③ Internal RAM of 128 bytes.
④ 32-I/O Pins.
⑤ Two 16-bit Timers/Counters (T_0 & T_1)
⑥ 8-bit stack pointer (SP)
⑦ 8-bit program counter (PC) & Data pointer (DPTR)
⑧ 8-bit program status word (PSW)
⑨ 6 Interrupt sources with priority levels.
⑩ Full duplex serial data Transmitter/Receivers.
⑪ On-chip oscillator circuits.

* comparison of 8051 family members.

Feature	8051	8052	8031
① Rom (on chip program space in bytes.	4K	8K	0K
② Ram (bytes)	128	256	128
③ Timers	2	3	2
④ I/O Pins	32	32	32
⑤ Serial port	01	01	01
⑥ Interrupt sources	06	08	08

(3) 16-bit microcontroller:- * cpu can handle only two bytes at a time.

* designed for high speed / high performance applications.

* these provide large program & data memory spaces for more flexible I/o capabilities, greater speed & less cost than any previous microcontrollers.

Manufacturer	Model	No. of pins	RAM	ROM
Intel	80C196	64	1-kbytes	32-kbytes.
Hitachi	H8/532	84	232-kbytes	8-kbytes.

(4) 32-bit microcontroller:-

* cpu can handle 32-bit data at a time

* designed for applications such as robotics control, highly intelligent instrumentation, image processing & other high end control system.

Ex's Intel 80960

ARM processor.

Overview of 8051 microcontroller

→ Microcontroller survey:-

(1) 4-bit microcontroller:-

* CPU can handle only 4-bit of data at a time.
* these microcontroller are introduced 1st & are still used in very small appliances.

Applications:- In toys.

Eg:-

S.No	manufacturer	model	RAM	ROM
1	Hitachi	HMCS40	32-bytes	512 bytes.
2.	Toshiba	TLCS47	128 bytes	2K-bytes.

(2) 8-bit microcontroller:-

* CPU can handle 8-bits at a time
* 8 bit size of data is proven to be very useful data size because ASCII data is stored in 8-bit formats. this makes 8-bits a good choice for data communication.
* Most of memory ICs are arranged in 8-bit configuration, which can be interfaced easily to data buses of 8-bits.

Applications:- variety of applications that involve limited calculations & simple control operations such as washing machines, TV etc.

Manufacturer	model	No. of pins	RAM	ROM
Intel	8051	40	128	4K-byte
Intel	8052	40	256	8K-byte.

Pins 10-17 : Port 3 :- Each of these pins can be configured as I/P or O/P pins.

Pins 10 & 11 : RxD & TxD :- 8051 has serial data communication circuits that use 'SBUF' register to hold the data & 'SCON' to control the data communication

* the data is transmitted out of 8051 through the TxD Pine.
* the data is received by 8051 through the RxD line.

Pin 12 $\overline{INT0}$:
Pin 13 : $\overline{INT1}$: } Interrupt 0 & Interrupt 1 are two Interrupt Pins that are triggered by External circuits.

Pins 14 & 15 :- T0 & T1 :- the 8051 has two 16-bit Timers/counters.

T0 → Timer0 register (16-bit)

T1 → timer 1 register (16-bit)

* they can be used Either as Timers to generate a time delay or as counters to count events happening outside the microcontroller.

Each 16-bit registers can be accessed as two separate 8-bit registers.

Pins 16 & 17 : $\overline{RD}$ & $\overline{WR}$:- these are active low Pins

when $\overline{RD}$ = 0, microcontroller reads the data from External RAM.

when $\overline{WR}$ = 0, microcontroller writes the data into External RAM.

② Bit-Addressable register:-

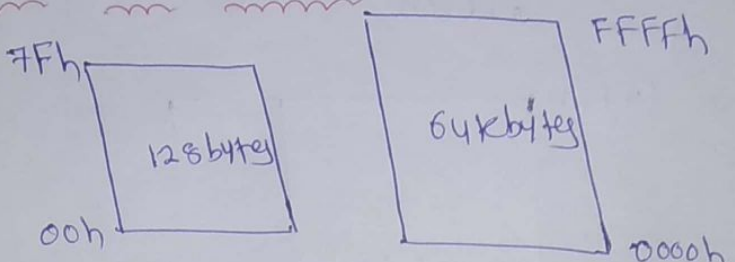
→ the 8051 provides 16-bytes of a bit addressable area. It occupies RAM area from 20h to 2Fh, forming a total of 128 addressable bits (i.e 16 bytes × 8-bits = 128 bits)

→ the addressable bit may be specified by its bit address of 00h to 7Fh.

→ 8-bit may form any byte address from 20h to 2Fh

③ General Purpose register:- the RAM area above bit addressable area from 30h to 7Fh is called General-purpose Register RAM.

(2) External Data memory:-



② Program memory (ROM) or:

→ program memory (ROM) can be classified into two types

① Internal ROM or on-chip ROM

② External ROM or off-chip ROM

① Internal ROM or on-chip ROM:-

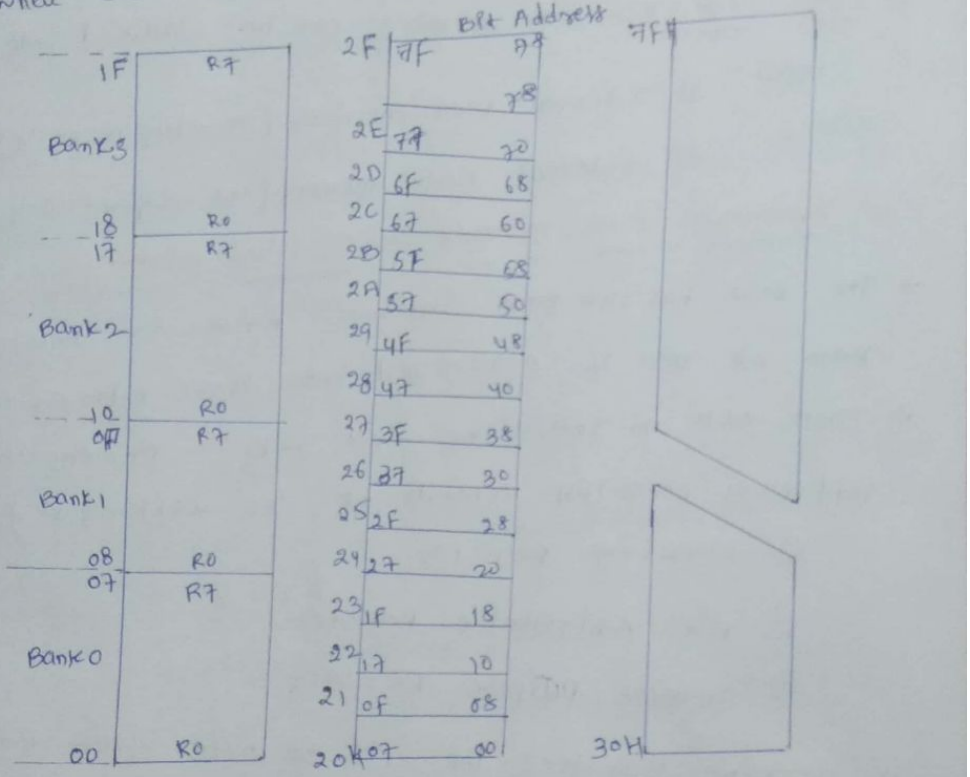
→ the 8051 has 4-Kbytes of Internal ROM with address ranges from 0000 to 0FFFh

→ the 8051 has control pins such as $\overline{PSEN}$ (program store enable) & $\overline{EA}$ (External access)

-> Only one Register bank is in use at a time.

-> You can access the register by name, for this, corresponding bank must be selected. In order to select the bank, we have to use the RS0 and RS1 bits of the program status word (PSW) register.

-> When 8051 is RESET, the Bank 0 is selected.



Working Register (32) | Bit Addressable Register (16) | General Purpose Register (80)

S.NO	Bank	No. of Register	Register Name	Address Range
1	Bank 0	8	R0 - R7	00h to 07h
2	Bank 1	8	R0 - R7	08h to 0Fh
3	Bank 2	8	R0 - R7	10h to 17h
4	Bank 3	8	R0 - R7	18h to 1Fh

Memory Organization of 8051:-

→ The Memory organization of 8051 microcontroller is classified into two types.

① Data memory (RAM)

② program memory (ROM)

① Data memory:- Data memory can be classified into two types. ① Internal Data memory (on-chip memory)

② External Data memory (off-chip memory)

① Internal Data memory (or) on-chip memory:-

→ The 8051 has 128 byte internal RAM, the Internal RAM of 8051 is organized into three different areas.

→ From 00H to 7FH, the first 32B i.e memory from addresses 00H to 1FH consists of 32 working Register.

① working Register.

② Bit Addressable Register

③ General purpose Register

① working Register:- The 1st 32 bytes from address 00h to 1Fh of Internal RAM consists 32 working Register. i.e

→ Register are organized as four Banks with 8 Register in each Bank.

→ Each Register can be addressed by name or by its RAM address.

(5) Branch Instruction set (or) control transfer Instructions:

→ these instructions are used to change the flow of the program by changing the content of PC

Jump:- the permanently changes the program flow.

Jump with out conditions:

* AJmp <address 11 bits> - Absolute Jump
* LJmp <address 16 bits> - long Jump
* SJmp <address> · short Jump.

Jump with conditions:-

JZ address; Jump if Accumulater is zero

JNZ address: Jump if Accumulator is set zero

JC address: Jump if carry flag is set.

JNC address; Jump if CY=0

JB address; Jump of the bit set if set.

CJNE <destination>, <Source>.

compare and Jump if not equal

DJNZ → Decrement and Jump if not zero.

NOP → No operation.

CALL:- It temporary changes the program flow to allow the subroutine program,

· LCALL → long CALL

ACALL → Absolute CALL.

—o—

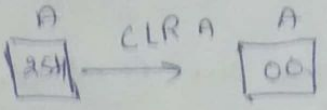
* logical Exclusive-OR operation

Ex: XRL A, #15H

XRL A, 20H

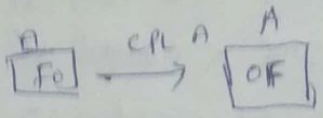
* clear the accumulator:

CLR A;



* Complement the accumulator:

CPL A;



* Rotate Instructions:

RL A: Rotate the accumulator left

RLC A: Rotate the content of 'A' towards left with carry.

RR A: Rotate Right

RRC A: Rotate Right with carry:

* Swap A: swap the nibbles with in the Accumulator

④ level logical Instructions for Boolean Instruction set:

-> CLR -> clear the carry flag / any bit

-> SETB -> set the carry flag / bit

-> CPL -> complement the bit

-> ANL C, <source-bit> logical AND operation for bit variables.

-> ORL C, <source-bit>

-> MOV <destination bit>, <source bit.

.

* Division :-

DIV · AB

[A] ← Quotient of A ÷ B

[B] ← Reminder of A ÷ B

* Increment :- INC A

* Increment the content of A by 1

[A] ← [A] + 1

* Decrement :-

DEC A; [A] ← [A] - 1

Decrement the content of A by 1

* Decimal Arithmetic Instruction.

DA A : Decimal Adjust Accumulator for BCD addition.

③ Logical Instruction :-

→ Byte level [8 bit]

Bit level logical Instructions.

Byte level logical Instructions.

Logical AND Instruction.

ANL <Destination>, <Source>

Ex :- ANL A, #15H

ANL A, R1

Logical OR Instruction :-

ORL <Destination>, <Source>

Ex :- ORL A, #15H

ORL A, R1.

ORL A, 20H

ORL A, @R1 :

→ Data transfer with stack:

PUSH <source> → push onto the stack

POP <destination> → pop from the stack

→ Data external instruction:

XCH XCH A, R1.

XCHD

② Arithmetic Instructions:

① These instruction implement arithmetic and logical operations along with increment, decrement and decimal adjust operations.

② Accumulator is a compulsory destination operand for two-operand instructions.

③ Immediate operand can be only source and not a destination operand.

* Addition:- → ADD A, <source>.

ADD A, #05H → [A] ← [A] + 05.

ADD A, R1.

→ ADDC A, <source> ; Add with carry

[A] ← [A] + [source] + CY

ADDC A, 25H.

* Subtraction:-

SUBB A, <source>

[A] ← [A] - [source] - CY

Ex:-

SUBB A, R0

SUBB A, @R1

SUBB A, #15H

* Multiplication:- [8 bit x 8 bit = 16 bit]

MUL AB

[A] - lower byte of AB

[B] - Higher byte of AB

* 8051 Instruction set :- 8051 Instruction can be categorized in the following categories.

1. Data Transfer Instruction set 28
2. Arithmetic Instruction set 24
3. Logical Instruction set 25
4. Boolean Instruction (or Bit manipulation) 17
5. Control Transfer Instruction. (or Branching Instruction set 17.

(1) Data Transfer Instruction :-

→ These Instruction implement a bit, byte (or) 16-bit Data Transfer operations blw the 'SRC' (source) and 'DST' 'Destination' operands.

→ Both operands can be internal direct Data memory operands.

→ Both cannot be direct and/or Indirect register operands R0 to R7.

→ Immediate operand can be only a source and not a destination.

→ only R0 and R1 are used for Indirect addressing mode.

Ex:- (1) mov <destination>, <source>.

→ used to move the data from the source to destination

MOV A, R1

mov A, #60H

(2) → mov x <destination>, <source>

to access external Data memory (RAM)

mov A, R1

(3) move <destination), <source>

to access External ROM/EPROM [program memory-code]

↳ When R0 and R1 are used as pointers, that is, when they hold the addresses of RAM locations. They must be preceded by the @ sign, as shown below.

MOVA, @R0

MOV @R1, B

⑤ Indexed addressing mode:- only program memory can be accessed using this addressing mode. Basically, this mode of addressing is accomplished in 8051 for look-up table manipulations. Program counter or data pointer are the allowed 16-bit address storage registers, in this mode of addressing.

ex: MOVC A, @A+DPTR; MOV DPTR, #0800H

→ this 16-bit register DPTR and register A are used to form the address of the data element stored in on-chip ROM. Because the data elements are stored in the program (code) space ROM of the 8051, the instruction MOVC is used instead of MOV. the 'C' means code.

—o—

MOV A, R0

MOV R5, A

ADD A, R5

ADD A, R8

MOV R0, A → It should be noted that the source and destination registers must match in size. In other words, coding "mov DPTR, A" will give an Error, since the source is an 8-bit register and the destination is a 16 bit register.

→ Notice that we can move Data b/w the accumulator and Rn (for n=0 to 7) but movement of Data b/w Rn registers is not allowed.

ex:- "mov R4, R7" is invalid.

Direct addressing mode:- In this mode of addressing, the operands are specified using the 8-bit address field in the instruction format. only Internal Data RAM and SFRS can be directly addressed.

ex:- MOV R0, 89H; → save content of RAM location 89H in R0.

MOV 56H, A ; save content of A in RAM location 56H

MOV R4, 7FH ; move contents of RAM location 7FH to R4.

Indirect addressing mode:- In the Register Indirect addressing mode, a register is used as a pointer to the Data. If the Data is inside the CPU, only registers R0 and R1 are used for this purpose.

Addressing modes of 8051:-

→ The way of accessing data, are called addressing modes.

Addressing modes can be classified as

1. Immediate Addressing mode
2. Register Addressing mode
3. Direct Addressing mode
4. In Direct Addressing mode
5. Indexed Addressing mode

① **Immediate Addressing mode:-** In this mode, an immediate data, i.e a constant is specified in the instruction, after the opcode byte. In this addressing mode, the source operand is a constant. Notice that the immediate data must be preceded by the pound sign, "#". this addressing mode can be used to load information into any of the registers, including the DPTR register.

Ex:- MOV A, #25H. → operands
(Register)
mnemonics → mov R4, #62 → constant values

mov DPTR, #4521H. → mov DPL, #21
mov DPH, #45.

② **Register addressing mode:-**

· Register addressing mode involves the use of registers to hold the data to be manipulated. examples of register addressing mode follow.

D3 to D4 bit then AC = 1, otherwise AC = 0.

RS1 & RS0 :- Register Bank selector.

RS1	RS0	Register Bank	address.
0	0	0	00H - 07H
0	1	1	08H - 0FH
1	0	2	10H - 17H
1	1	3	18H - 1FH.

Overflow Flag (OF):-

-> OF flag is set to 1 if either of the following two conditions occurs:

① there is a carry from D6 to D7 but NO carry out of D7 (CY = 0)

② there is a carry out from D7 bit (CY = 1) but NO carry from D6 to D7 - bit.

parity flag (P):- parity flag indicates the number of 1's present in the accumulator.

* If the number of 1's in the accumulator is odd then P = 1.

* If the number of 1's in the accumulator is even then P = 0.

Special Function Registers (SFR):-

the operations of 8051 are done by a group of specific internal registers, each called a special function register (SFR)

* PC is the only register that does not have an Internal address.

I/p - o/p ports (I/o ports):-

→ The 8051 has 32 I/o pins configured as four 8-bit parallel ports i.e port 0, port 1, port 2 and port 3.

* All four ports are bi-directional i.e each pin can be configured as I/p & o/p under software control.

Timers and Counters:-

* 8051 use two 16-bit registers namely T0 & T1, either for Timer or Counter.

* the two timers or counters are divided into two 8-bit registers called Timer low (TL0, TL1) and Timer high (TH0 & TH1).

Program status word (PSW) or Flag register:-

MSB D7	D6	D5	D4	D3	D2	D1	D0 LSB
CY	AC	F0	RS1	RS0	OV	-	P

Carry flag (CY):- After performing arithmetic & logic operation if there is a carry out from the MSB (D7-bit) then CY=1, otherwise CY=0.

Auxiliary carry flag (AC):- After performing arithmetic & logic operation if a carry is generated from

* the stack pointer register is used by the 8051 to hold an internal RAM address that is called the top of the stack.
* When 8051 is RESET, the SP is set to 07h.
* the storing of a CPU register on the stack is called a PUSH.
* loading the contents of the stack back into the CPU register is called a POP.

Data pointer (DPTR):-

* DPTR is a 16-bit register, which holds a 16-bit address.
* DPTR can be splitted into two parts.
  1) DPH → Data pointer high byte having Internal address 83h.
  2) DPL → Data pointer low byte having Internal address 82h.

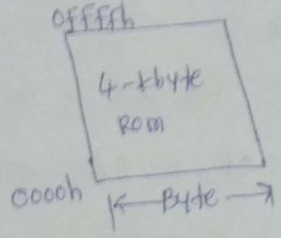
→ the DPTR does not have a single Internal address.

* Program counter (PC):-

* PC is a 16-bit register which holds the address of the next instruction to be executed. the PC is automatically incremented after every instruction byte is fetched.
* the 8051 has 16-bit PC hence it can address upto 2^{16} bytes i.e 2^{16} = 64K bytes of memory.

Internal Rom:-

* the 8051 has 4-kbytes of Internal Rom with address space from 0000h to 0FFFh



* the program address higher than 0FFFh, which exceeds the Internal Rom capacity will cause the 8051 to automatically fetch code bytes from external program memory:

A Register (Accumulator):-

* Accumulator is a 8-bit register it is widely used for many operations like addition, subtraction, multiplication, division & boolean bit manipulations.
* the A-register is also used for all Data transfers b/w the 8051 and any external memory.

B-Register:- the B-register is used with the A-Register for multiplication & division operations. it has no other function other than as a location where data may be stored.

Stack pointer (8-bit):-

* the stack refers to an area of Internal Ram used by the CPU to store & retrieve (take back) data quickly.
* the Register used to access the stack is called the Stack Pointer (SP) Register.

* The 8051 has 128 byte Internal RAM, the Internal RAM of 8051 is organized into three distinct area.

① Working Registers.

② Bit addressable Registers.

③ General Purpose Registers.

* Working Registers:- The 1st 32 bytes from address 00h to 1Fh of Internal RAM constitutes 32 working Registers i.e.

Bank0 → 8 Registers (R0-R7) : 00 to 07h

Bank1 → 8 Registers (R0-R7) : 08 to 0Fh

Bank2 → 8 Registers (R0-R7) : 10h to 17h

Bank3 → 8 Registers (R0-R7) : 18h to 1Fh

* Bits RS0 & RS1 in the PSW determine which bank of registers is currently in use.

* When 8051 is RESET, the Bank 0 is selected.

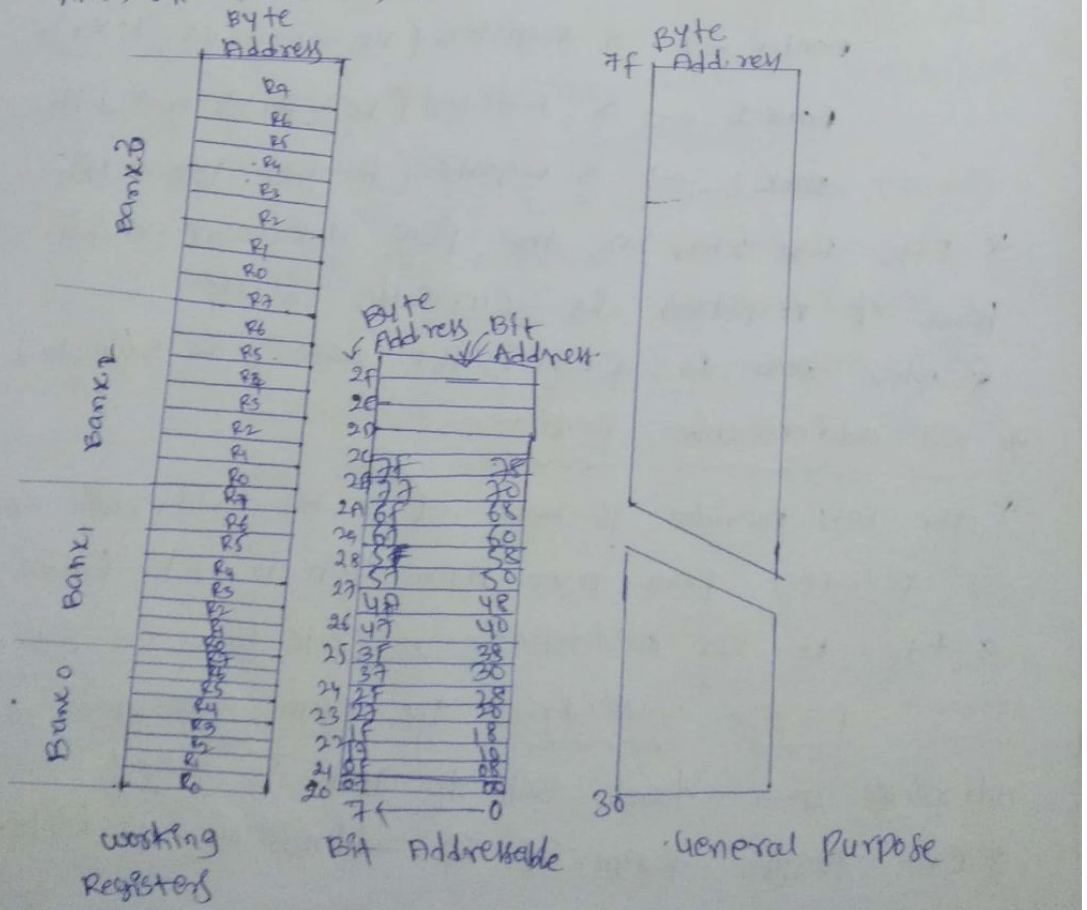
* Bit addressable Registers:-

* The 8051 provides 16-bytes of a bit addressable area. It occupies RAM area from 20h to 2Fh forming a total of 128 addressable bits (16 bytes × 8-bits = 128 bits)

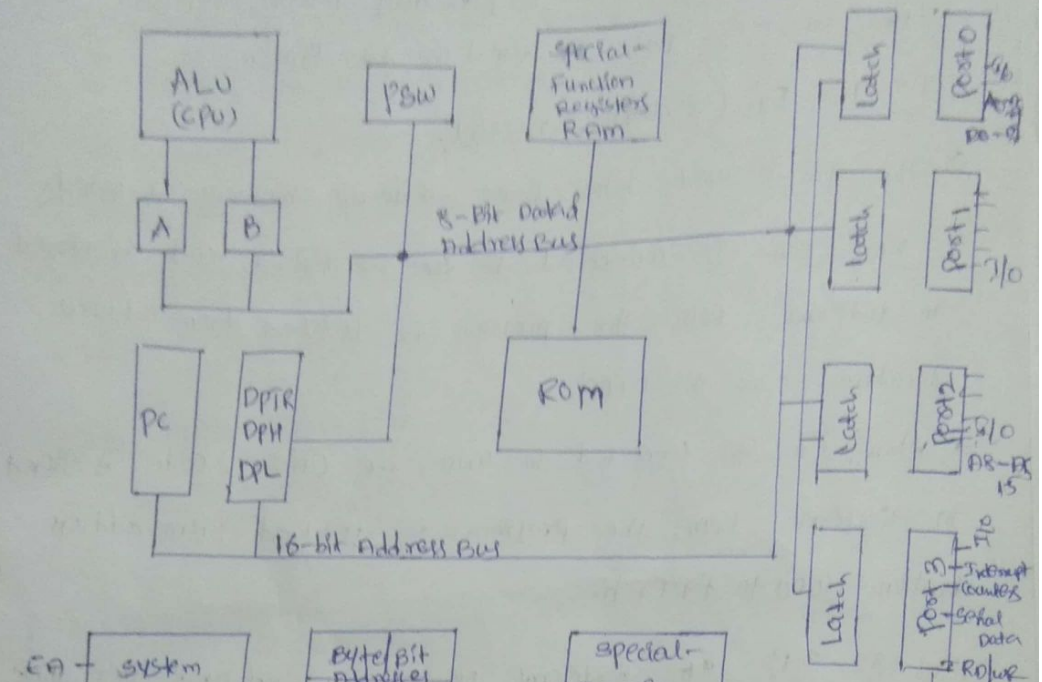
General purpose registers:- the RAM area above bit addressable area from 30h to 7Fh is called general purpose RAM. It is addressable as byte.

Central Processing unit (CPU):-

* the 8051 CPU consists of 8-bit arithmetic & logic unit with associated registers like A, B, PSW, SP, 16-bit program counter & Data pointer registers" (DPTR)
* the 8051's ALU can perform arithmetic & logic functions on 8-bit variables. the arithmetic unit performs addition, subtraction, Multiplication & Division.
* the logic unit can perform logical operations such as AND, OR & EX-OR; as well as rotate, clear & complement.



* Architecture of 8051:-



Internal Ram structure

fig: Architecture of 8051

Pin 30:- ALE (Address Latch Enable):-

→ when ALE = 1, port 0 is providing lower order address (A0-A7)

→ when ALE = 0, port 0 is used as data lines.

Pin 31:- $\overline{EA}$ (External Access):-

* this pin is used when ever external memory is used.
* when $\overline{EA}$ is connected to Vcc i.e $\overline{EA} = 1$, code is stored in internal rom, the program is fetched from address location 0000 to 0FFFh.
* when $\overline{EA}$ is connected to GND i.e $\overline{EA} = 0$, code is stored in external rom, the program is fetched from address location 0000 to FFFFh.

Pins 32-39:- If external memory is not used, then these pins can be used as I/p's or O/p's.

* If external memory is used then the lower address i.e A0-A7 will appear on this port.

Pin 40:- Vcc:- DC power supply +5V is connected to this pin.

Pins 18 & 19:- XTAL2 & XTAL1:-

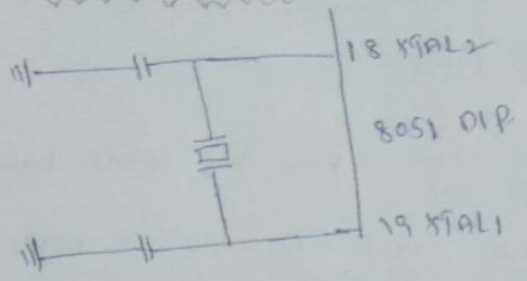


Fig:- Crystal or ceramic Resonator oscillator circuit

* the 8051 has an on-chip oscillator but requires an external clock to run it. A Quartz crystal oscillator is connected to I/p's XTAL1 & XTAL2 with two capacitors having values 30pf.
* If an external frequency (from AFO) have to be applied. then it must be applied b/w XTAL1 & GND. XTAL2 must be left open.
* oscillator frequency may vary from 10MHz to 40MHz.

Pin 20; VSS:- It is a ground pin.

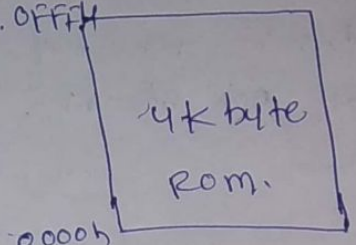
Pins 21-28:- Port 2:- If external memory is not used. these pins can be used as I/p's or o/p's

* If external memory is used then the higher address i.e A8 - A15 will appear on this port.

Pin 29:- $\overline{PSEN}$ (program store Enable):- It the memory access is for the byte of program code in the ROM, then $\overline{PSEN}$ signal goes low & the data byte from the ROM is placed on the data bus.

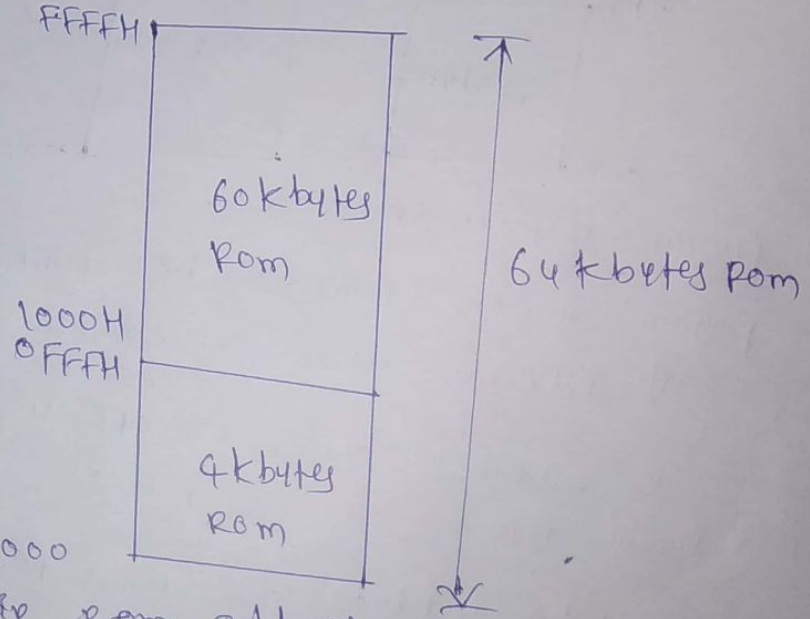
that determines whether external memory is accessed or Internal memory is accessed.

→ If $\overline{EA}$ pin is connected to Vcc (usually +5V) then the program memory accessed by the chip is Internal memory. 0FFFH



(ii) External Memory (or) (Rom):

→ When $\overline{EA}$ pin is connected to Vss (0V & Ground) then the 8051 can access external memory. starting from address 0000h to FFFFh



Note:- On-chip rom address 0000h to 0FFFH
Off-chip Rom address 0000h to FFFFH

—o—