Instruction set of 8086:

The ins We have some instructions to create the assembly language programming of 8086 with the help of addressing mode

1) Data copy / transfer instructions:-

These instructions are used for copy / transfer the data one register to another register

Mov instruction:-

It moves the content from one register to another register and also it copies the content into the register

Eg:- Mov AX, BX

push instruction:-

It insert the content into the register by using a stack pointer as it increases by 1 that is SP+1

Eg:- push Ax

pop instruction:-

It is used to delete the data from AX register by using stack pointer as decrement by 1 that is SP-1 Eg:- pop AX.

Xchange instruction:-

It XCHG the content of BX with AX content

Eg: XCHG AX, BX

2) Arithmatic & logic instructions:-

These instructions are used to perform arithmatic and logical operations as follows

i) ADD AX, BX:-

This instruction adds the content of BX into AX and final result will be stored in AX.

ii) SUB AX, BX:-

This instruction Subtracts the content of Bx with AX and final result will be stored in AX.

iii) ADC AX, BX

This instruction adds the content of AX and BX and then if carry is executed within the addition then carry is added to the result and the final result is stored in AX

iv) SBB AX, BX :-

This instruction subtract the content of BX from AX and then if borrow exhibit Subtract the borrow from the result

v) MUL BX

It multiplies the content of BX those content are unsigned.

vi) DIV BX

It divides the content of AX by BX.

vii) IMUL BX :-

It multiplies the content of AX, BX and the content are signed numbers the MSB is 1.

viii) IDIV BX.

It divides the content of AX by BX and the content are signed numbers the MSB is 1

ix) INC AX

It increments the content of AX by 1 final result will be stored in AX.

x) Dec AX

It decrements the content of AX by 1 final result will be stored in AX.

Logical instructions:-

These instructions are used to perform logical operations between two registers

1) AND AX, BX :-

It is used to Logical AND between the contents of AX and BX and final result is stored in AX. When both inputs are logical high then the result is also high. Remaining inputs are '0'

2) NAND AX, BX :-
It is used to Logical NAND between the contents of AX and Bx and final result is stored in AX. When both inputs are logical high then the result is low remaining inputs the result is high.

3) OR AX, BX :-
It is used to Logical OR between the content of AX and Bx and final result is stored in AX. when both both inputs are low then the result is low remaining inputs the result is high.

4) NOR AX, BX :-
It is used to Logical NOR between the content of AX and Bx and final result is stored in AX: when both inputs are low then the result is high remaining inputs the result is low.

5) XOR AX, BX :-
It is used to Logical XOR between the content of AX and Bx and final result is stored in AX. When two identical inputs are then the result is low remaining inputs are result is high

6) XNOR AX, BX :-
It is used to Logical XNOR between the content of AX and Bx and final result is stored in AX. When we give two identical inputs then the result is high remaining inputs are result is low.

7) NOT AX, BX :-
It is used to Logical NOT between the content of AX and Bx and final result is stored in AX. When one input is high and the result is low. similarly when one input is low and the result is high.

Control transfer instruction :-
These instructions are used to control the machine code data and then converted into Shift left or right / rotate right or left. We can transfer the control by using the value of count register.

i) SHL AL, CL :-

This instruction shift the content of AX in the value of count register and final result will be stored in AX.

Eg:- 76H CL=02

0111 0110
1101 1000
1101 1000
D 8

ii) SHR AL, CL :-

This instruction shift the content of AX. In the value of count register and final result will be stored in AX.

Eg:- 76H

0111 0110 CL=02
0011 1011
0001 1101
1D

iii) ROL AL, CL :-

This instruction is used to rotate the content of AL in the count register and final result will be stored AL

Eg:- 76 H.

0111 0110 CL=02
1110 1100
1101 1001
D9

iv) ROR AL, CL

This instruction is used To rotate the content of AL in the count register and final result will be stored AL

Eg:- 76 H

0111 0110 CL=02
0011 1011
1001 1101
9D

4) String manipulation instruction:-
String is a group of bytes or words and their memory is always allocated in a sequential order. These instructions are used to manipulate the strings like movement of strings, comparison of string, length of the string, Reverse of the string

MOV SB or MOV SW:-
It is used for moving the string byte or string word from one register to another register

CMP SB or CMPSW:-
It is used to compare the string byte or string word then each and every character is same or not

REP/REPE/REPNE/REPNZ (Repeat)
Repeat the instruction until the condition becomes false.

LODS/LODSB/LODSW (load)
Copies a byte from word from a string location pointed to SI to AL or word from location pointed to by source index (SI) to a string location pointed to by AX.

SCASB/SCASW (scan)
It is used to scanning or searching each and every character is given to strings and finding it gets the designed character given by the user

STOSB/STOSW (store)
It is used to store the string in the memory block that is data segment, does not effect any flag

OUTS/OUTSB/OUTSW:-
Output string byte/string word from the memory location to the IO port

Jump instruction:-
Jump instructions are also known as branch instructions. It is divided into 2 types.

If jump occurs only when condition was satisfy jumping from one label to another label it is called conditional jump instruction.

If jump occurs with no condition is called unconditional jump instruction.

JE (Jump equal):-

Jump occurs when result is equal

JNE (Jump NOT equal):

Jump occurs when result was not equal

JZ

Jump if zero.

JNZ

Jump if result is not zero.

JC

Jump if carry

JNC:- Jump if result has no carry.

JS:- Jump if result was signed number

JNS:- Jump if result was not a signed number

Loop instruction:-

Loop z

It is used when the Loop result was zero

Loop NZ:- It is used for when the result of loop is not zero.

flag manipulation instruction:-

STC:- (set carry)

It is used to carry exhibits which is unable whenever carry are generated by addition or borrow in the Subtraction.

STI:- (set Interrupt)

It is used whenever interrupt arises while executing an application program.

CLC (clear carry)

It is used to clear the carry flag bit.

CLI (clear interrupt)

It is used to clear the interrupt flag whenever interrupt was vanished by ISR (Interrupt Service routine)

Assembler directives:-

Assembler directives are the special instruction used to indicate the assembler how a program to be executed assembled and executed in a proper way. Assembler directives are the standard libraries which are used in 8086 microprocessor assembly language programming.

It is used to indicate the memory allocations, data movement into the segment register.

• model small :-

It is one of the assembler directive which assigns 64 KB for each and every segment register. under this library we can use simple 8086 microprocessor assembly language programming like ALU, data transfer that is read and write

Note:- • model medium, • model large which are used for advanced programming in 8086.

Assume:-

It is one of the assembler directive which can be used to assume the segment register has CS, DS, SS and ES.

Assign:-

It is one of the assembler directive which is used to assign define byte (DB), DW (define word).

DB:- It is used to assign a single byte to the given data by the user.

Ex:- assign DB 56H

DW:- It is used to assign a single word that is 16 bit data to the given data by the user

Ex:- assign DW 56H

DD:- (Define double word):-

It is used to assign two words that is 32 bit to the given data.

Ex:- assign DD 56H

DQ:- (Define quad)

It is used to assign 4 words that is 64 bits to the given data.

DT :-
It is used to assign 10 bytes that is 80 bits to the given data.

DUP (?) (uninitialize)
It is used to create the duplicate copy of given string and question mark indicate the uninitialized point.

DUP(0)
It is also used to create the duplicate copy of given string which is initialize at '0'.

- Stack :- It is used to store the main program termination address and it reads the top of the stack (64 KB)
- code :- It is used to store the programming instructions given by the user it is also 64 KB
- Data :- It is used to store the data items input or output it is also 64 KB.

Start :- It is used to start the main program.
End start :- It is used to End the program
- Title :- It is used to assign the file name to our program with extension .ASM
  Eg :- Sindhu.ASM
- page :- It is used to know the number of characters per instruction and maximum no. of instructions given by the user
  Eg :- [Length] [width]
  [20 instructions per page] [200 characters per instruction]

Macros and procedures :-
It is said to be Subroutine or Subprogram which are written in main program.

```
Main program
  =
Call proc1
Procedure 1
  =
RET proc 1
  =
end
```

It uses two instructions call - to call procedure from main program. RET - Return back to main program.

There are two types of procedure, 1) near procedure, 2) far procedure.

For calling near procedure we use near call that is within the code segment (same segment register) for main program that is in code segment and program is in different segment (ES) for call a far procedure by using far call.

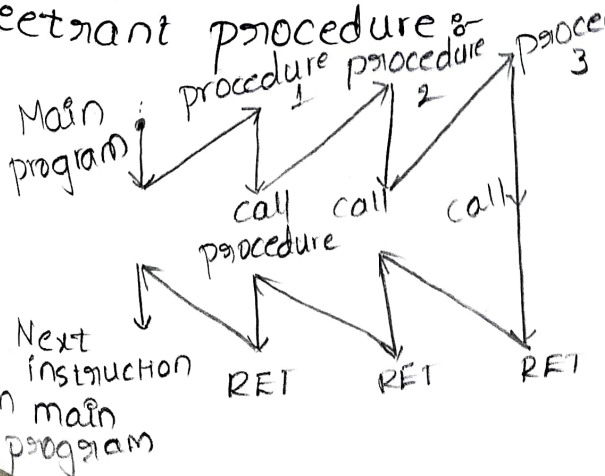
It is a set of instructions used in the main program to call a procedure by using call proc which gives the particular procedure. After execution of the procedure the instruction pointer get back to main program. RET proc. These are the lengthy number of instructions used in the main program with the help of call and RET functions.

procedures are of two types 1) Reetriant procedure 2) Recursive procedure.

Recursive procedure:- This procedure which are repeatedly calling the same procedure with several times in the main program.

Main program
=
call proc 1
-
RET proc 1
=
-
Call proc 1
=
=
RET proc 1

Reetriant procedure:-



In this method the flow of program execution reenters the procedure 1 from procedure 2 and then main program.

It is also called as nested p...

Macros :-

It is a single executable program designed by macro assembler.

Ex:- Macro label 1

.
.
.
.

END Macro

It does not have call and return functions

It is completely machine instructed code by assembler, macros are assigned with a label of name macro label. The assembler generate a code in the program each time when macro is called.

Macro sequence execute faster than procedure because of no call and return function

More memory is required in the macros.

It is a group of instructions which is a single executable program designed by the macro assembler.

Differences b/w the macros and procedure

Macros	procedure
=> Requires more memory	=> Requires less memory
=> Does not require CALL and RET instructions	=> Requires CALL and RET instructions.
=> Machine code is generated each time the macro is called	=> Machine code generates only once.
=> Parameters are passed in registers, memory locations or stack	=> Parameters passed as part of statment which calls macro.
=> Macro executes faster than a procedure	=> procedure executes slower than a macro.

8051 Microcontroller :-

=> 8051 employes Hardvard architecture. It has periferal's such as 32 bit digital I/O's, Timer's & serial I/O ports.

=> the basic Arcture of 8051 is given in below figure.

=> i) 8 bit CPU

ii) 16 bit program Counter

iii) 8 bit processor status word.

=> iv) 8 bit stack pointer

=> Internal RAM of 128 bytes.

=> on chip rom is 4 KB

=> 32 I/O pins arranged as 4-8 bit pot's (P_0, P_1, P_2, P_3).

⇒ Two 16 bit Timer/counter's i.e T_0 & T_1

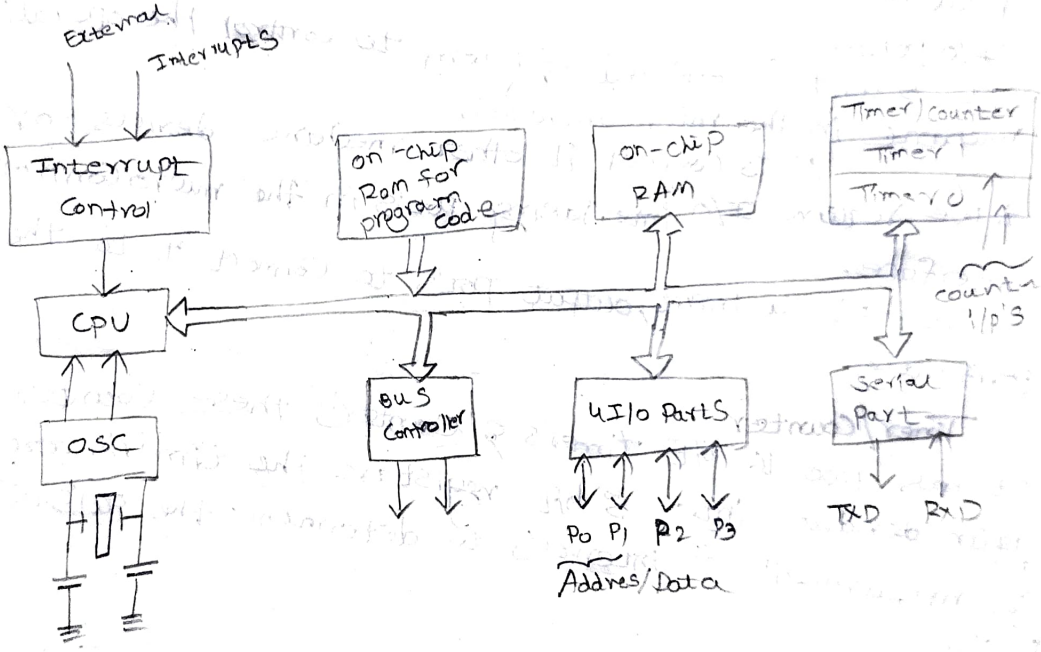
⇒ Two external & 3 internal Vectored interrupts.

⇒ one full duplex serial I/o Communication.

CPU:- (central processing unit);-

It is the brain of any prosseing derice of the microcontroller it monitor's & control's are all operations that are performed on the microcontroller unit,

⇒ the user has no control over the work of the CPU directly. It reads the program return in Rom memory & execute's them and do the expected task of that application.



Interrupt ;-

Interrupt is a Subrotaine Call that interrupts of the microcontroller main operation or work & causes it to execute any other program which is more important at the time of operation.

It is very useful as it help's in case of emergny operations.

⇒ An interrupts gives us a machanism to put on hold the on going operations, execute a subrotaine & then again resumes to another type of operations.

Bus :-

It is a collection of wire's which work has a Communication chanel or medium for transfer of data these are two types.

Address bus:-

It is a 16 bit address bus for transfering the data it is used to address memory locations & to transfer the address bus. CPU to memory of the micro Controller.

Data bus :-

It is a 8 bit's of a data bus it is used to Carry data of Particular application.

Oscilator :-

It requires clock Pulse's for it's operation of microController application.

I/O pot :-

It is used in embeded system to Control the operation of mechane in the microcontroller.

∴ To Connect it other mechane devices or Periferals & we require I/O interfasing pot's in the micro Controller interface.

It has 4 input, output pot's to Connect it to the other periferals.

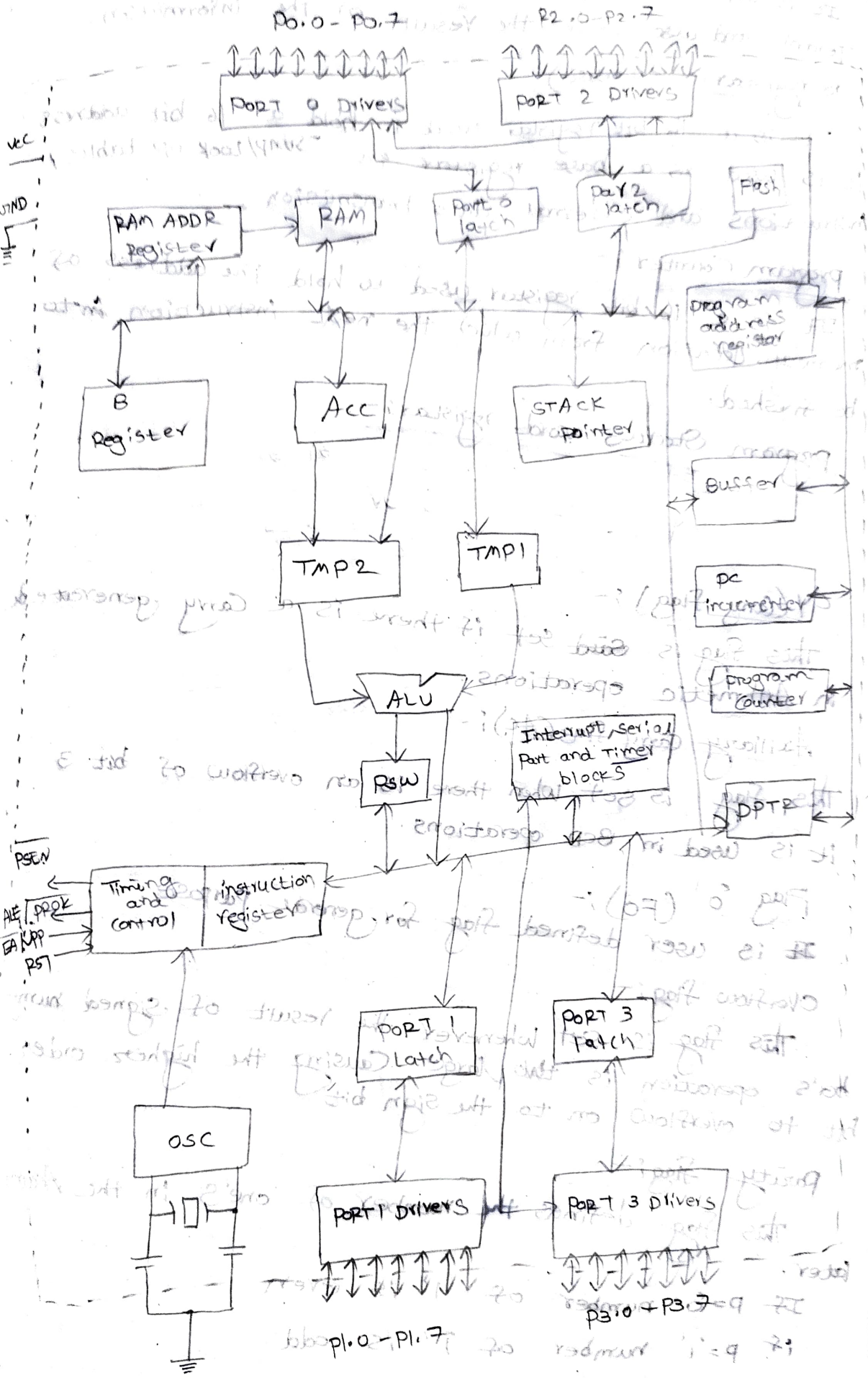
Timer/Counter

It has two 16 bit timer's & Counter's these Counter are again devided into 8 bit registar. the timer's are used for measurment of intervals to determine the pul's width of Pul's.

Memory :-

⇒ It is a Collection of instructions,
⇒ this program tells microController to do specific task
⇒ The memory which is used to store the program of the microcontroller is known as code memory or program memory it is known as ROM memory of microController.
⇒ It is used to store data temperary for operatios is knows as RAM
⇒ It has 4K of Code memory i.e 4KB ROM & also 128 bytes of data memory of RAM.

Internal Architecture of 8051 Microcontroller



Accumulator :-

It is an 8 bit register It is used to hold the source operant and also resive the result of the information.

B registar :- (B → Base)

It is a 16 bit register used to hold a 16 bit address it is used as a base registar in Jump/Look up table, instructions and external data transmision.

program Counter :-

It is a 16 bit registar used to hold the address of memory location from which the next instruction into be freshed.

program Status word registar :-

B7	B6	B5	B4	B3	B2	B1	B0
CY	AC	F0	RS1	RS0	OV	-	P

CY (Carry flag) :-

This flag is set if there is a Carry generated in Arthmetic operations.

Auciliary Carry flag (AC) :-

This flag is set when there is an overflow of bit 3 it is used in BCD operations.

Flag '0' (F0) :-

It is user defined flag for genera purpose.

Overflow flag :-

This flag is set whenever the result of signed number's operation is too large causing the higher order bit to overflow on to the sign bit

parity flag :-

this flag defines the number of one's in the Accumulater.

If p='0' number of '1' is even

if p='1' number of '1' is odd.

RS0 & RS1 :- Registar bank selection.

RS1	RS0	Bank Selection.
0	0	Bank 0 [00H to 07H]
0	1	Bank 1 [08H to 0FH]
1	0	Bank 2 [10H to 17H]
1	1	Bank 3 [18H to 1FH]

IO port :-
8051 has 32 bidirectional IO pins which are organized as 4 parall 8 bit IO ports.

port 0 :-
It can be used as 8 bit IO port & multiplexed as AD0 - AD7.

port 1 :-
It can be used as 8 bit IO port.

port 2 :-
It can be used as 8 bit IO ports as well as higher order address bus A8 - A15.

port 3 :-
It can be used as 8 bit IO ports & multifunctional unit.

Stack pointer :-
It points out to the top of the stack & it indicates the next data to be accessed stack pointer can be accessed using push or pop, call and return instructions.

It is an 8 bit registar.

When writing a new data the stack pointer is automatically incremented by '1' the new data is written at new an address SP+1.

When reading the data from stack the data is retrived from the address from the stack pointer after that stack pointer is decreased by 1