MES chapter 2 // 8051 instruction set & programming

2.1 instruction set (data transfer, arithmatic & logic, Branching, machine control, stack operation, boolean)

2.2 Addressing Modes
2.3 Assembly language programming
2.4 8051 programming in C

2.1 >>> instruction set -> • data Transfer
• Arithmatic "
• Logic "
• Branching "
• Machine control "
• Stack operation "
• boolean "

1> Data Transfer - These instruction allow you to move data between the register and memory, load or store data in memory and manipulate the stack
8051 include : mov, mvi, LDA, STA, XCHG, PUSH, POP

1> mov :- move data from one register or memory location to another
2> MVI :- move immediate data to a register or memory location.
3> LDA :- Load accumulator with data from memory
4> STA :- store accumulator in memory
5> LHLD :- load H & L register with data from memory
6> SHLD :- store H & L register in memory
7> XCHG :- Exchang content of H&L with content of D&E register

2. Arithmatic instruction - These instruction allow you to perform basic arithmetic operation such as addition, subtraction, increment & Decrement

- This instruction set allow you to perform more complex operations such as addition with carry & subtraction with borrow
- in That instruction contain following :-

ADD, ADI, ACI, SUB, SUI, INR, DCR, DAD

1) ADD :- Add The contents of a register or memory location to The accumulator

2) ADI - add an immediate value to The accumulator

3) ACI - Add an immediate value to The accumlator along with The carry flag

4) SUB - subtract The contents of a register or memory location from The accumulator

5) SUI - Subtract an Immediate value from the accomulator

6) INR - increment The contents of a register or memory location

7) DCR - Decrement The contents of a register or memory location

3. Logic instruction – Logical instruction are used to perform logical operations like AND, OR, XOR, NOT clear and swap. Logical instruction are performed on Bytes of data on a bit-by-bit basis.

Logical instruction are following –

* ANL, ORL, XRL, CLR, CPL, RL, RLC, RR, RRC, SWAP

ANL - Logical AND

ORL - Logical OR

XRL - Ex-OR

CLR - clear Register

CPL - complement The Register

RL - Rotate a byte to left

RLC - Rotate a byte and carry bit to left

RR - Rotate a byte to Right

RRC - Rotate a byte and carry bit to right

SWAP - exchange lower & higher nibbles in a byte

4. Branching instruction - These instructions control the flow of program logic. The mnemonics of the program branching instruction are following-
• LJMP, AJMP, SJMP, JZ, JNZ, NOP, LCALL, ACALL, RET, JMP

LJMP - Long jump (unconditional)

AJMP - Absolute jump (unconditional)

SJMP - short jump (unconditional)

JZ - jump if a equal to 0

JNZ - jump if not equal to 0

NOP - no operation

LCALL - Long call to subroutine

ACALL - Absolute call to subroutine (unconditional)

RET - return from subroutine

JMP - jump to an Address (unconditional)

5 Machine control - the 8051 microcontroller has a variety of machine control instruction that can be used to control its internal operations
- These include instruction for moving data between registers and memory ~~to perform~~.
example - mov, Add, RET, POP
- These instruction form the basic building blocks of a program running on 8051 microcontroller

* stack operation instruction - The stack is a section of RAM used by the CPU to store information such as data or memory address on temporary basis
- The storing operation of a CPU register in the stack is known as a PUSH and getting the contents from the stack back into a CPU register is called a POP

Push operation :-
```
000h
mov 08h, #21h
mov 09h, #56h
PUSH 00h
PUSH 01h
END
```

POP operation :-
```
000H
mov 00H, #21h
mov 01H, #32h
POP 1FH
POP 0EH
END
```

6. Boolean instruction - Boolean instruction deal with bit variable we know that there is a special bit-addressable area in the RAM and some of the special Function Register are also bit addressable

Boolean instruction are following -
- CLR, SETB, MOV, JC, JNC, JB, JNB, JBC
  ANL, ORL, CPL

CLR - clear a bit (reset to 0)

SETB - set a bit (set to 1)

MOV - move a bit

JC - jump if carry flag is set

JNC - jump if carry flag is not set

JB - jump if specified bit is set

JNB - jump if specified bit is not set

JBC - jump if specified bit is set and also clear the bit

ANL - Bitwise AND

ORL - Bitwise OR

CPL - complement the bit

2.2 >>> Addressing mode

- Immediate Addressing mode
- Register Addressing mode
- Direct Addressing mode
- Register indirect mode
- indexed Addressing mode

1> immediate Addressing mode - in this immediate mode the data is provided in the instruction itself the data is provided immediatly after the oprode. These some example - MOVA, #0AFH;
MOVR3, #45H;
MOVDPTR, #FE00H;

in this # symbol is used for immediate data in the last DPTR instruction stand for data pointer

* in first 0AFH instruction the immediate data is AFH but one 0 is added at the beggining so when the data is starting with A to F the data should be preceded by 0

2> Register Addressing mode - in the register addressing mode the source or destination data should be present in a register

example :- MOVA, R5;
MOVR2, #45H;
MOVR0, A;

3> Direct Addressing mode - in the direct Addressing mode the source or destination address is specified by using 8-bit data in the instruction only the internal data memory can be used in this mode here some of the example -

example - MOV 80H, R6;
MOV R2, 45H; Ram location.
MOV R0, 05H;
↑ Ram location

4> Register indirect Addressing Mode - in this mode the source or destination address is given in the register by using register indirect Adressing mode the internal or external addresses can be acessed

- The R0 & R1 are used for 8-bit address.
- The ~~DPTR~~ DPTR is used for 16-bit address

example - MOV0E5H, @R0;
MOV @R1, 80H

5> indexed Addressing mode - in the indexed Addressing mode the source memory can only be accessed from program memory only. The destination operand is always the register A

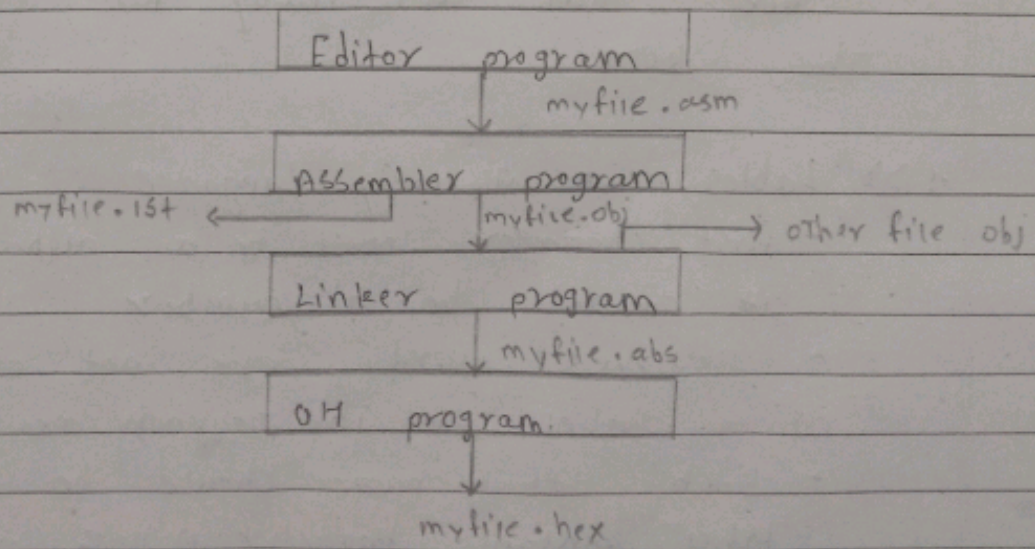
example - MOVCA, @A + PC;
MOVCA, @A + DPTR;

2.3 >>> Assembly language programming

1) Assembly language - Assembly language is a low-level programming language for a computer or other programmable device specific to a particular computer architechre in contrast to must high-level programming language.
- Assembly language is converted into executable machine code by a utility program referred to as an assembler like NASM, MASM
-

2) Advantages of Assembly Language.
- it requires less memory and execution time
- it allows hardware-specific complex jobs in a easier way
- it is suitable for time-critical jobs
- it is most suitable for writing interrupt service routines, and other memory resident programs.

3) Assembling & Running of 8051 Program



1> Editor Program :- At first we use an editor for Type in a program . editors like ms-dos . The editor produces an ASCII file

2> Assembler program :- The " asm" source file contain the code created in step 1 it is transffered to an 8051 assembler
• The assembler is used ~~the~~ for converting The assembly language instruction into machine code & it is produced . obj file & .lst file means object file & list file.

3> Linker program - The linker program is used for gencrating one or more object file & produces an absoloute object file with an extension " abs".

4) OH Program - The OH program fetches The " abs" file & fed it to a program called "OH" . OH is called as object to hex convertor it creates a file with an extension " hex" That is ready for burn in to The ROM

4 >> Lables in Assembly Language.
- first character must be a alphabetical character it cannot be a number
- Reserved words are not allowed to be used as a label in The program example - mov , add
- Each label name should be unique label contain character A TO Z, number 0 TO 9, and @ , ? , - , $ This special symbol

5> High Level VS. Assembly

High level	Assembly
i> more programmer friendly	i> makes low level programming more user friendly
ii> more ISA independent	ii> very ISA - in dependent
iii> each high-level instruction Translate to several instruction in the ISA of the compuer	iii> each instruction specifies a single ISA instruction

6> section of assembly language programming.

- assembly language program divided into 3 section.
  - data section
  - bss section
  - text section

i> data section - The data section is used for declaring initialized data or constants This data cannot change in Run time
syntax :- section.data

ii> bss section - The bss section is used to declaring variables
Syntax - section.bss

iii> Text section - The text section is used for keeping the actual code This section must begin with the declaration global -start which tells the kernal where the program execution begins

```
section .text
        global -start
-start:
```

4) assembly language program Example.

```
segment .text
    global _start
_start:
    mov edx, len
    mov ecx, msg
    mov ebx, msg1
    mov eax, 4
    int 0x80

    mov eax, 1
    int 0x80

segment .data
msg db 'Hellow world', 0xa
len equ $ - msg
```

2.4 >>> 8051 Programming in C

1> what is Embedded C

- embedded C is a set of language extensions ~~of~~ for the C programming language enhaced for different embedded systems
- embedded C uses most of the syntax ~~the~~ and semantics of standard C example - main() function, conditional statement, for-while loop

2> Data type in embedded C

unsigned char	→ 8 bit	→	0 to 255
signed char	→ 8 bit	→	-128 to +127
unsigned int	→ 16-bit	→	0 to 65535
signed int	→ 16-bit	→	-32768 to +32767
sbit	→ 1-bit	→	SFR bit address only
bit	→ 1-bit	→	RAM bit-addressable only
sfr	→ 8-bit	→	RAM addresses 80 FFH only

Data Type | size of Bits | Data Range

3> C Program ~~to~~ for send value of 00-FF to port1

```
#include <reg51.h>
void main()
{
  unsigned char i;
  for (i=0; i<=255; i++)
      P1 = i;
}
```

4> Operator in C

* Logical operators
  AND (&&), OR (||), NOT (!)

* Bit - wise operator
  AND (&), OR (|), EX-OR (^), inverter (~)

* shift right (>>), shift left (<<)

5> Example program

```
# include <reg51.h>
void delay (unsigned int time) {
    unsigned int i, j;
    for (i=0; i< Time; i++);{
     }
    for (j=0; j<1275; j++);
}
void main () {

    while (1) {
      P1 = 0x01;
       delay (1000);

      P1 = 0x00;
       delay (1000);
    }
}
```

* > Program Explanation.

1. #include <reg51.h> :- This line include "reg51.h" header file which is specific to the 8051 architecture and contain definitions for the registers & bit-level operations

2. void delay (unsigned int time) :- This function create delay of a specified time & it takes in a single argument

3. unsigned int i , j ; :- These variable are used in the nested loops That make up the delay function

4. void main () { :- Thise is main functin. our program ~~started~~ executed first paint

5. P1 :- This is Pin 1

6. delay (1000) :- Thise line called delay function with an argument 1000 , which create a delay of 1 second