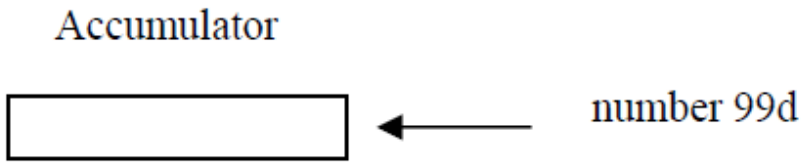
Unit-II
8051 Instruction Set and programming

2.1 Addressing modes
2.2 Instruction set (Data transfer, Logical, Arithmatic, Branching, Machine control, Stack operation, Boolean)
2.3 Assembly language programming (ALP)
2.4 Software development cycle: editor , assembler , cross- compiler, linker,locator,compiler
2.5 Assembler Directives: ORG , DB , EQU , END,CODE,DATA

2.1 ADDRESSING MODES

There are a number of addressing modes available to the 8051 instruction set, as follows:
Immediate Addressing
Register Addressing
Direct Addressing
Indirect Addressing
Indexed Addressing

Immediate Addressing
Immediate addressing simply means that the operand (which immediately follows the instruction op. code) is the data value to be used. For example the instruction:
MOV A, #99d

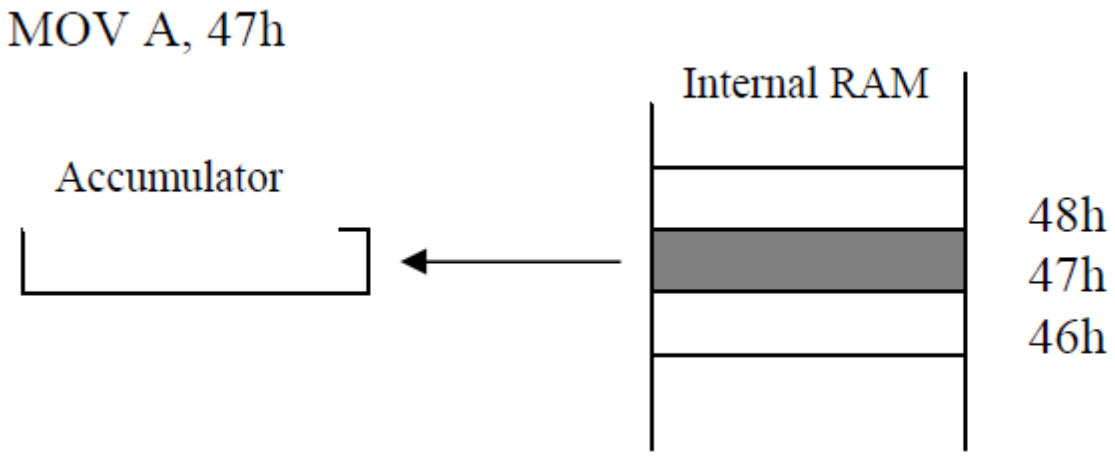


Moves the value 99 into the accumulator (note this is 99 decimal since we used 99d). The # symbol tells the assembler that the immediate addressing mode is to be used.

Register Addressing
One of the eight general-registers, R0 to R7, can be specified as the instruction operand. The assembly language documentation refers to a register generically as *Rn*.
An example instruction using register addressing is :
MOV A, R5 ; Moves register R5 to A (accumulator)

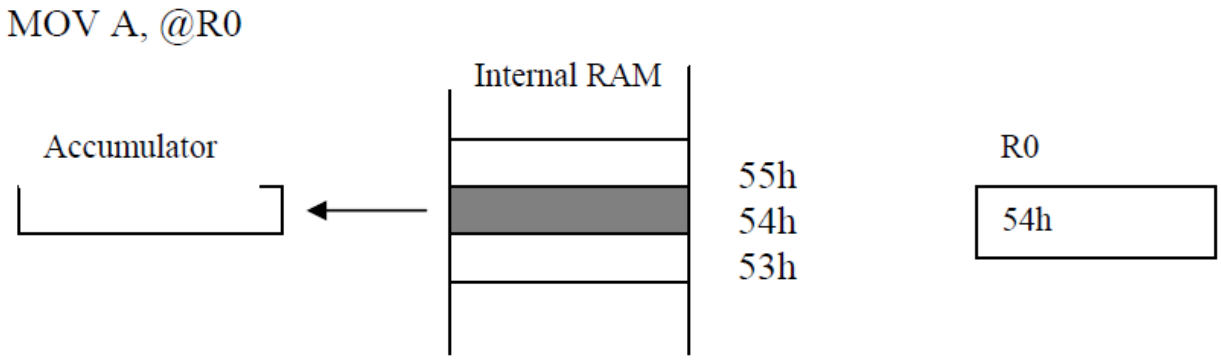
Here the contents of R5 is added to the accumulator. One advantage of register addressing is that the instructions tend to be short, single byte instructions.

Direct Addressing
Direct addressing means that the data value is obtained directly from the memory location specified in the operand. For example consider the instruction:
MOV A, 47h



The instruction reads the data from Internal RAM address 47h and stores this in the accumulator. Direct addressing can be used to access Internal RAM , including the SFR registers.

Indirect Addressing
Indirect addressing provides a powerful addressing capability, which needs to be appreciated. An example instruction, which uses indirect addressing, is as follows:
MOV A, @R0



The @ symbol indicated that the indirect addressing mode is used. R0 contains a value, for example 54h, which is to be used as the address of the internal RAM

An advantage of relative addressing is that the program code is easy to relocate in memory in that the addressing is relative to the position in memory.

Indexed Addressing
With indexed addressing a separate register, either the program counter, PC, or the data pointer DTPR, is used as a base address and the accumulator is used as an offset address. The effective address is formed by adding the value from the base address to the value from the offset address. Indexed addressing in the 8051 is used with the JMP or MOVC instructions. Look up tables are easy to implemented with the help of index addressing. Consider the example instruction:
MOVC A, @A+DPTR
MOVC is a move instruction, which moves data from the external code memory space. The address operand in this example is formed by adding the content of the DPTR register to the accumulator value. Here the DPTR value is referred to as the *base address* and the accumulator value us referred to as the *index address*.

2.2 Instruction Set

Writing a Program for any Microcontroller consists of giving commands to the Microcontroller in a particular order in which they must be executed in order to perform a specific task. The commands to the Microcontroller are known as a Microcontroller's Instruction Set.

Just as our sentences are made of words, a Microcontroller's (for that matter, any computer) program is made of Instructions. Instructions written in a program tell the Microcontroller which operation to carry out.

Types of instructions

Depending on operation they perform, all instructions are divided in several groups:

- Arithmetic Instructions
- Branch Instructions
- Data Transfer Instructions including Stack operations
- Logic Instructions
- Bit-oriented Instructions

The first part of each instruction, called MNEMONIC refers to the operation an instruction performs (copy, addition, logic operation etc.). Mnemonics are abbreviations of the name of operation being executed. For example:

The other part of instruction, called OPERAND is separated from mnemonic by at least one whitespace and defines data being processed by instructions. Some of the instructions have no operand, while some of them have one, two or three. If there is more than one operand in an instruction, they are separated by a comma. For example:in MOV A,R0, MOV is the mnemonic or opcode while A and R0 are operands.The following is the summary if all instructions available in 8051.

DATA TRANSFER	ARITHMETIC	LOGICAL	BOOLEAN	PROGRAM BRANCHING	Machine control
MOV	ADD	ANL	CLR	LJMP	NOP
MOVC	ADDC	ORL	SETB	AJMP	
MOVX	SUBB	XRL	MOV	SJMP	
PUSH (stack)	INC	CLR	JC	JZ	
POP(stack)	DEC	CPL	JNC	JNZ	
XCH	MUL	RL	JB	CJNE	
XCHD	DIV	RLC	JNB	DJNZ	
	DA A	RR	JBC	NOP	
		RRC	ANL	LCALL	
		SWAP	ORL	ACALL	
			CPL	RET	
				RETI	
				JMP	

Arithmetic instructions

Arithmetic instructions perform several basic operations such as addition, subtraction, division, multiplication etc. After execution, the result is stored in the first operand. For example:

ADD A,R1 - The result of addition (A+R1) will be stored in the accumulator.

ARITHMETIC INSTRUCTIONS			
Mnemonic	**Description**	**Byte**	**Cycle**
ADD A,Rn	Adds the register to the accumulator	1	1
ADD A,direct	Adds the direct byte to the accumulator	2	2
ADD A,@Ri	Adds the indirect RAM to the accumulator	1	2
ADD A,#data	Adds the immediate data to the accumulator	2	2
ADDC A,Rn	Adds the register to the accumulator with a carry flag	1	1
ADDC A,direct	Adds the direct byte to the accumulator with a carry flag	2	2
ADDC A,@Ri	Adds the indirect RAM to the accumulator with a carry flag	1	2
ADDC A,#data	Adds the immediate data to the accumulator with a carry flag	2	2
SUBB A,Rn	Subtracts the register from the accumulator with a borrow	1	1
SUBB A,direct	Subtracts the direct byte from the accumulator with a borrow	2	2
SUBB A,@Ri	Subtracts the indirect RAM from the accumulator with a borrow	1	2
SUBB A,#data	Subtracts the immediate data from the accumulator with a borrow	2	2
INC A	Increments the accumulator by 1	1	1
INC Rn	Increments the register by 1	1	2
INC Rx	Increments the direct byte by 1	2	3
INC @Ri	Increments the indirect RAM by 1	1	3
DEC A	Decrements the accumulator by 1	1	1
DEC Rn	Decrements the register by 1	1	1
DEC Rx	Decrements the direct byte by 1	1	2
DEC @Ri	Decrements the indirect RAM by 1	2	3
INC DPTR	Increments the Data Pointer by 1	1	3
MUL AB	Multiplies A and B	1	5
DIV AB	Divides A by B	1	5
DA A	Decimal adjustment of the accumulator according to BCD code	1	1

Branch Instructions

There are two kinds of branch instructions:

Unconditional jump instructions: upon their execution a jump to a new location from where the program continues execution is executed.

Conditional jump instructions: a jump to a new program location is executed only if a specified condition is met. Otherwise, the program normally proceeds with the next instruction.

BRANCH INSTRUCTIONS			
Mnemonic	**Description**	**Byte**	**Cycle**
ACALL addr11	Absolute subroutine call	2	6
LCALL addr16	Long subroutine call	3	6
RET	Returns from subroutine	1	4
RETI	Returns from interrupt subroutine	1	4
AJMP addr11	Absolute jump	2	3
LJMP addr16	Long jump	3	4
SJMP rel	Short jump (from –128 to +127 locations relative to the following instruction)	2	3
JC rel	Jump if carry flag is set. Short jump.	2	3
JNC rel	Jump if carry flag is not set. Short jump.	2	3
JB bit,rel	Jump if direct bit is set. Short jump.	3	4
JBC bit,rel	Jump if direct bit is set and clears bit. Short jump.	3	4
JMP @A+DPTR	Jump indirect relative to the DPTR	1	2
JZ rel	Jump if the accumulator is zero. Short jump.	2	3
JNZ rel	Jump if the accumulator is not zero. Short jump.	2	3
CJNE A,direct,rel	Compares direct byte to the accumulator and jumps if not equal. Short jump.	3	4
CJNE A,#data,rel	Compares immediate data to the accumulator and jumps if not equal. Short jump.	3	4
CJNE Rn,#data,rel	Compares immediate data to the register and jumps if not equal. Short jump.	3	4
CJNE @Ri,#data,rel	Compares immediate data to indirect register and jumps if not equal. Short jump.	3	4

DJNZ Rn,rel	Decrements register and jumps if not 0. Short jump.	2	3
DJNZ Rx,rel	Decrements direct byte and jump if not 0. Short jump.	3	4
NOP	No operation	1	1

Data Transfer Instructions

Data transfer instructions move the content of one register to another. The register the content of which is moved remains unchanged. If they have the suffix “X” (MOVX), the data is exchanged with external memory.

DATA TRANSFER INSTRUCTIONS			
Mnemonic	**Description**	**Byte**	**Cycle**
MOV A,Rn	Moves the register to the accumulator	1	1
MOV A,direct	Moves the direct byte to the accumulator	2	2
MOV A,@Ri	Moves the indirect RAM to the accumulator	1	2
MOV A,#data	Moves the immediate data to the accumulator	2	2
MOV Rn,A	Moves the accumulator to the register	1	2
MOV Rn,direct	Moves the direct byte to the register	2	4
MOV Rn,#data	Moves the immediate data to the register	2	2
MOV direct,A	Moves the accumulator to the direct byte	2	3
MOV direct,Rn	Moves the register to the direct byte	2	3
MOV direct,direct	Moves the direct byte to the direct byte	3	4
MOV direct,@Ri	Moves the indirect RAM to the direct byte	2	4
MOV direct,#data	Moves the immediate data to the direct byte	3	3
MOV @Ri,A	Moves the accumulator to the indirect RAM	1	3
MOV @Ri,direct	Moves the direct byte to the indirect RAM	2	5
MOV @Ri,#data	Moves the immediate data to the indirect RAM	2	3
MOV DPTR,#data	Moves a 16-bit data to the data pointer	3	3
MOVC A,@A+DPTR	Moves the code byte relative to the DPTR to the accumulator (address=A+DPTR)	1	3
MOVC A,@A+PC	Moves the code byte relative to the PC to the accumulator	1	3

	(address=A+PC)		
MOVX A,@Ri	Moves the external RAM (8-bit address) to the accumulator	1	3-10
MOVX A,@DPTR	Moves the external RAM (16-bit address) to the accumulator	1	3-10
MOVX @Ri,A	Moves the accumulator to the external RAM (8-bit address)	1	4-11
MOVX @DPTR,A	Moves the accumulator to the external RAM (16-bit address)	1	4-11
PUSH direct	Pushes the direct byte onto the stack	2	4
POP direct	Pops the direct byte from the stack/td>	2	3
XCH A,Rn	Exchanges the register with the accumulator	1	2
XCH A,direct	Exchanges the direct byte with the accumulator	2	3
XCH A,@Ri	Exchanges the indirect RAM with the accumulator	1	3
XCHD A,@Ri	Exchanges the low-order nibble indirect RAM with the accumulator	1	3

Logical Instructions

Logic instructions perform logic operations upon corresponding bits of two registers. After execution, the result is stored in the first operand.

LOGIC INSTRUCTIONS			
Mnemonic	**Description**	**Byte**	**Cycle**
ANL A,Rn	AND register to accumulator	1	1
ANL A,direct	AND direct byte to accumulator	2	2
ANL A,@Ri	AND indirect RAM to accumulator	1	2
ANL A,#data	AND immediate data to accumulator	2	2
ANL direct,A	AND accumulator to direct byte	2	3
ANL direct,#data	AND immediae data to direct register	3	4
ORL A,Rn	OR register to accumulator	1	1
ORL A,direct	OR direct byte to accumulator	2	2
ORL A,@Ri	OR indirect RAM to accumulator	1	2
ORL direct,A	OR accumulator to direct byte	2	3
ORL direct,#data	OR immediate data to direct byte	3	4

XRL A,Rn	Exclusive OR register to accumulator	1	1
XRL A,direct	Exclusive OR direct byte to accumulator	2	2
XRL A,@Ri	Exclusive OR indirect RAM to accumulator	1	2
XRL A,#data	Exclusive OR immediate data to accumulator	2	2
XRL direct,A	Exclusive OR accumulator to direct byte	2	3
XORL direct,#data	Exclusive OR immediate data to direct byte	3	4
CLR A	Clears the accumulator	1	1
CPL A	Complements the accumulator (1=0, 0=1)	1	1
SWAP A	Swaps nibbles within the accumulator	1	1
RL A	Rotates bits in the accumulator left	1	1
RLC A	Rotates bits in the accumulator left through carry	1	1
RR A	Rotates bits in the accumulator right	1	1
RRC A	Rotates bits in the accumulator right through carry	1	1

Bit-oriented Instructions or Boolean instructions

Similar to logic instructions, bit-oriented instructions perform logic operations. The difference is that these are performed upon single bits.

BIT-ORIENTED INSTRUCTIONS			
Mnemonic	**Description**	**Byte**	**Cycle**
CLR C	Clears the carry flag	1	1
CLR bit	Clears the direct bit	2	3
SETB C	Sets the carry flag	1	1
SETB bit	Sets the direct bit	2	3
CPL C	Complements the carry flag	1	1
CPL bit	Complements the direct bit	2	3
ANL C,bit	AND direct bit to the carry flag	2	2
ANL C,/bit	AND complements of direct bit to the carry flag	2	2
ORL C,bit	OR direct bit to the carry flag	2	2

ORL C,/bit	OR complements of direct bit to the carry flag	2	2
MOV C,bit	Moves the direct bit to the carry flag	2	2
MOV bit,C	Moves the carry flag to the direct bit	2	3

2.4 Software development cycle: editor , assembler , cross- compiler, linker, locator, compiler

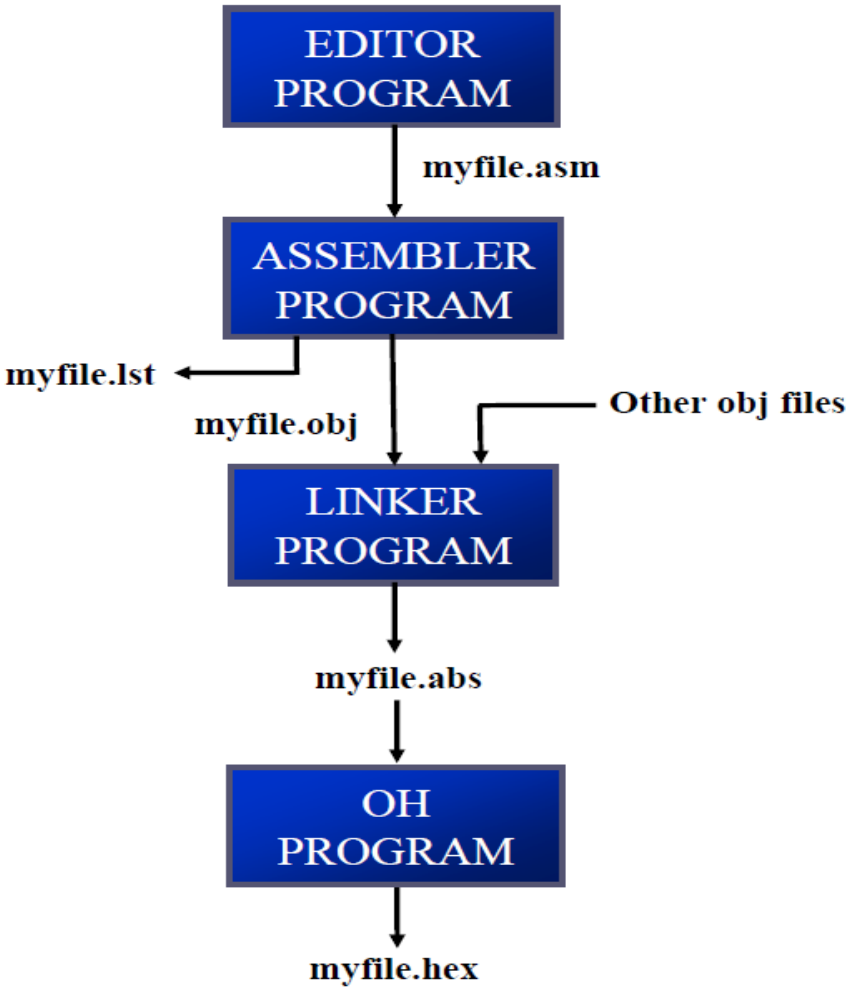


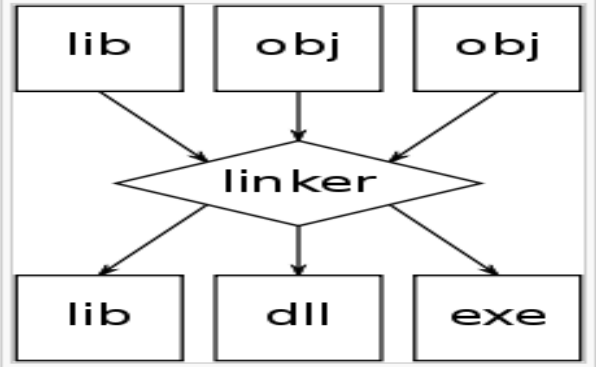
Fig : Software development cycle

Editor: An Editor is a program which allows us to create a file containing the assembly language statements for the program.. As we type the program the editor stores the ACSII codes for the letters and numbers in successive RAM locations. If any typing mistake is done editor will alert

us to correct it.. After typing all the program we have to save the program . This we call it as source file. The source file has the extension “asm“ or “src”, depending on which assembly you are using The next step is to process the source file with an assembler.

Assembler: An Assembler is used to translate the assembly language mnemonics into machine language(i.e binary codes). The “asm” source file containing the program code created in step 1 is fed to an 8051 assembler. The assembler converts these instructions into machine codes. The assembler will produce an object file and a list file. The extension for the object file is “obj” while the extension for the list file is “lst” . Object files contain a combination of machine instructions, data, and information needed to place instructions properly in memory. The lst (list) file, which is optional, lists all the opcodes and addresses as well as errors that the assembler detected.

Linker: A linker is special program that combines the object files, generated by assembler, and other pieces of codes to originate an executable file. In the object file, linker searches and append all libraries needed for execution of file. It regulates the memory space that will hold the code from each module. The linker program takes one or more object code files and produce an absolute object file with the extension “abs”.



An illustration of the linking process.

OH Program: Next the “abs” file is fed into a program called “OH” (object to hex converter) which creates a file with extension “hex” that is ready to burn into ROM.

Locator: Locator decides where in the memory the program will be and fix up all the addresses.

Compiler : A compiler is a computer program that translates a program in a high level language like C or C++ (source language) into an equivalent program in a machine language or an intermediate code (target language).

Cross Compiler: A cross compiler is a compiler capable of creating executable code for a platform other than the one on which the compiler is running.

For example, a compiler that runs on a Windows 7 PC but generates code that runs on Android smartphone is a cross compiler

2.5 Assembler Directives: ORG , DB , EQU , END, CODE, DATA

Directives give messages or instructions to the assembler. They are not part of instruction set and do not generate any machine codes. They can be used to define symbols, reserve memory space for data or program etc.

ORG : **(origin)**
The ORG directive is used to indicate the starting address. It can be used only when the program counter needs to be changed. The number that comes after ORG can be either in hex or in decimal.
Eg: ORG 0000H ;Set PC to 0000.

DB: The DB directive is used to define an 8 bit data. DB directive initializes memory with 8 bit values. The numbers can be in decimal, binary, hex or in ASCII formats. For decimal, the 'D' after the decimal number is optional, but for binary and hexadecimal, 'B' and 'H' are required. For ASCII, the number is written in quotation marks ('LIKE This).

```
ORG 500H
DATA1: DB 28                    ;DECIMAL (1C in Hex)
DATA2: DB 00110101B             ;BINARY (35 in Hex)
DATA3: DB 39H                   ;HEX
ORG 518H
DATA6: DB "My name is Joe"      ;ASCII CHARACTERS
```

EQU: This is used to define a constant without occupying a memory location. The EQU directive does not set aside storage for a data item but associates a constant value with a data label. When the label appears in the program, its constant value will be substituted for the label.

Example:-

```
COUNT EQU 25
MOV R3, # COUNT
```

END: This indicates to the assembler the end of the source (asm) file. It informs the assembler where to stop assembling the program. The END directive is the last line of an 8051 program. That is, in the code, anything after the END directive is ignored by the assembler. Assembler will generate error message if END is not written.

CODE : This directive assigns a code address value to a symbol. Range of code address value is 0000H to FFFF H.

Eg. MEM1 CODE 0030.

DATA : This directive assigns a data address value to the symbol . IDATA assigns internal data memory address (Range 00H to FFH). XDATA assigns external data memory address.(Range 0000 H to FFFF H).