KTU NOTES

The learning companion.

KTU STUDY MATERIALS | SYLLABUS | LIVE NOTIFICATIONS | SOLVED QUESTION PAPERS

Website: www.ktunotes.in

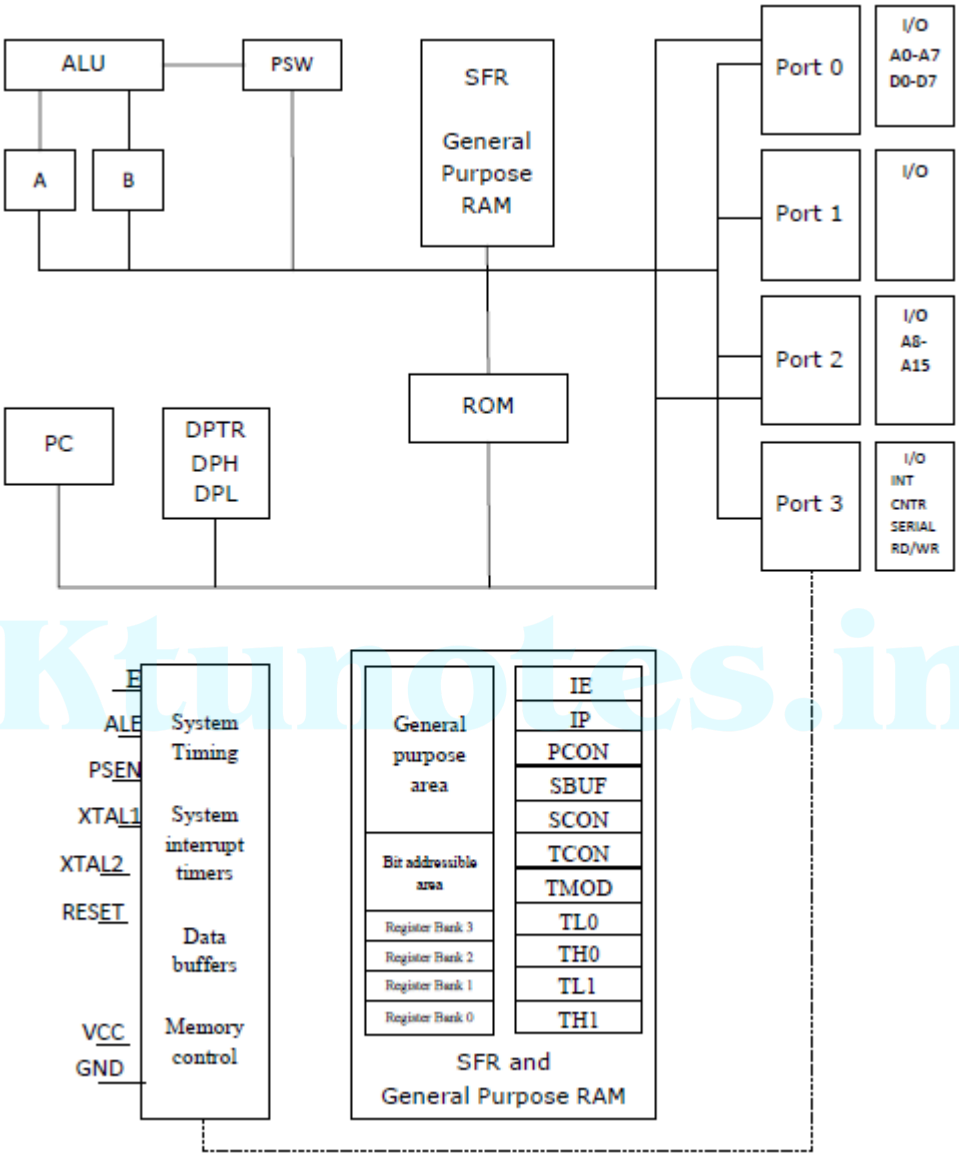
MODULE II
8051 ARCHITECTURE

2.1 INTRODUCTION

Salient features of 8051 microcontroller are given below.

- Eight bit CPU
- On chip clock oscillator
- 4Kbytes of internal program memory (code memory) *[ROM]*
- 128 bytes of internal data memory *[RAM]*
- 64 Kbytes of external program memory address space.
- 64 Kbytes of external data memory address space.
- 32 bi directional I/O lines (can be used as four 8 bit ports or 32 individually addressable I/Olines)
- Two 16 Bit Timer/Counter :T0, T1
- Full Duplex serial data receiver/transmitter
- Four Register banks with 8 registers in each bank.
- Sixteen bit Program counter (PC) and a data pointer (DPTR)
- 8 Bit Program Status Word (PSW)
- 8 Bit Stack Pointer
- Five vector interrupt structure (RESET not considered as an interrupt.)
- 8051 CPU consists of 8 bit ALU with associated registers like accumulator 'A' , B register, PSW, SP, 16 bit program counter, stack pointer.
- ALU can perform arithmetic and logic functions on 8 bit variables.
- 8051 has 128 bytes of internal RAM which is divided into
  - Working registers [00 – 1F]
  - Bit addressable memory area [20 – 2F]
  - General purpose memory area (Scratch pad memory) [30-7F]

2.2 THE 8051 ARCHITECTURE.



- 8051 has 4 K Bytes of internal ROM. The address space is from 0000 to 0FFFh. If the program size is more than 4 K Bytes 8051 will fetch the code automatically from external memory.
- Accumulator is an 8 bit register widely used for all arithmetic and logical operations. Accumulator is also used to transfer data between external memory. B register is used along with Accumulator for multiplication and division. A and B

registers together is also called MATH registers.

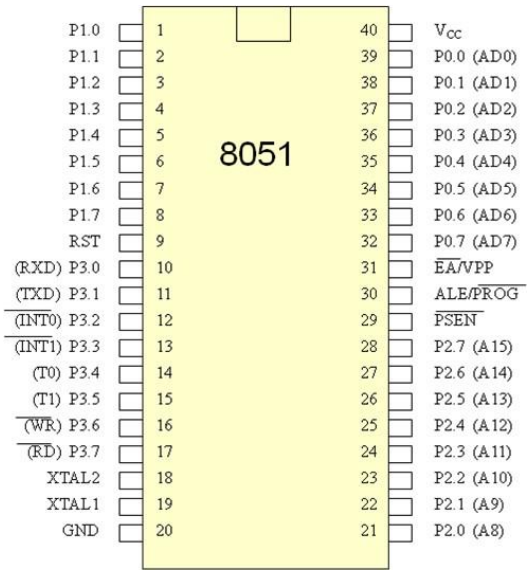
- PSW (Program Status Word). This is an 8 bit register which contains the arithmetic status ofALU and the bank select bits of register banks.

CY	AC	F0	RS1	RS0	OV	-	P

CY - carry flag
AC - auxiliary carry flag
F0 - Available for user for general purpose
RS1,RS0 - register bank select bits
OV - overflow
P - parity

- Stack Pointer (SP) – it contains the address of the data item on the top of the stack. Stack may reside anywhere on the internal RAM. On reset, SP is initialized to 07 so that the defaultstack will start from address 08 onwards.
- Data Pointer (DPTR) – DPH (Data pointer higher byte), DPL (Data pointer lower byte). This is a 16 bit register which is used to furnish address information for internal and external program memory and for external data memory.
- Program Counter (PC) – 16 bit PC contains the address of next instruction to be executed. On reset PC will set to 0000. After fetching every instruction PC will increment by one.

2.3 PIN DIAGRAM



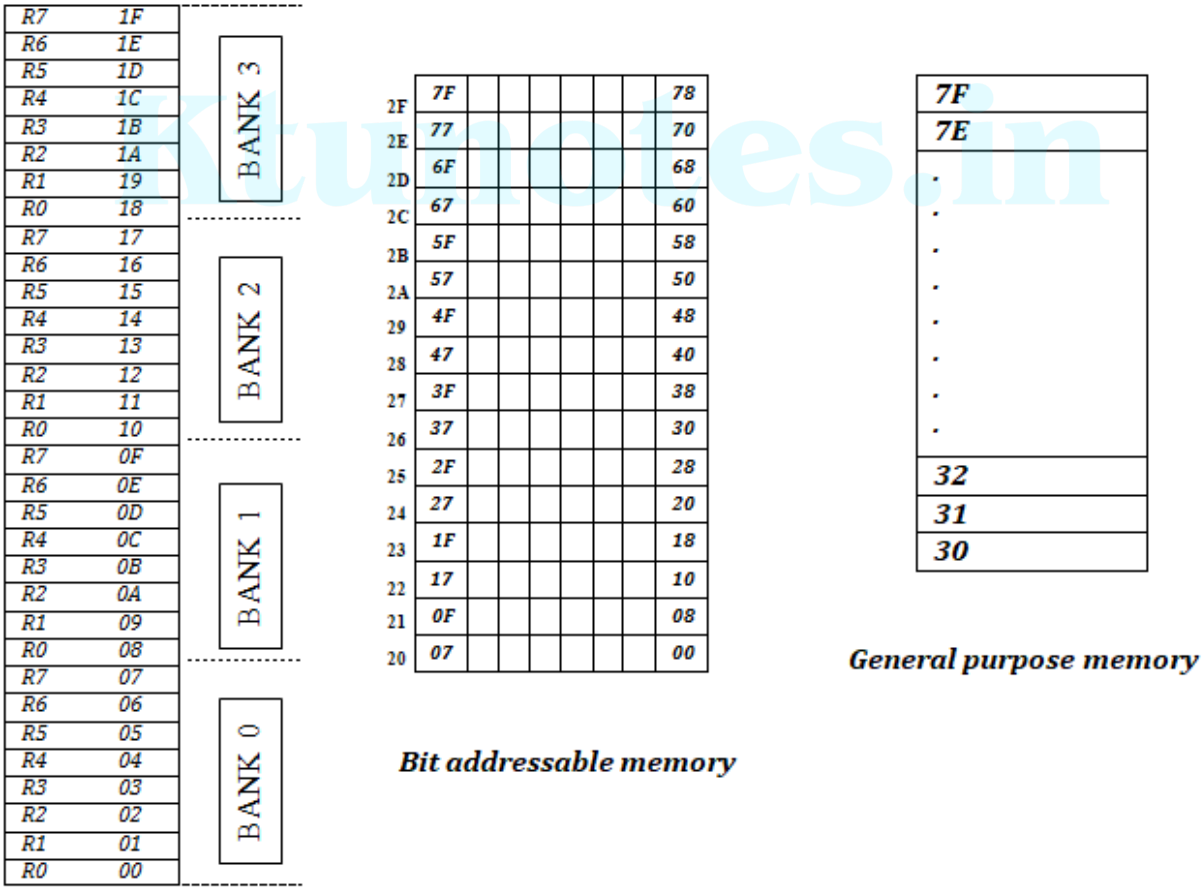
PINOUT DESCRIPTION

Pins 1-8	**PORT 1**. Each of these pins can be configured as an input or an output.
Pin 9	**RESET**. A logic one on this pin disables the microcontroller and clears the contents of most registers. In other words, the positive voltage on this pin resets the microcontroller. By applying logic zero to this pin, the program starts execution from the beginning.
Pins10-17	**PORT 3**. Similar to port 1, each of these pins can serve as general input or output. Besides, all of them have alternative functions
Pin 10	**RXD. Serial asynchronous communication input or Serial synchronous communication output.**
Pin 11	**TXD. Serial asynchronous communication output or Serial synchronous communication clock output.**
Pin 12	**INT0.External Interrupt 0 input**
Pin 13	**INT1. External Interrupt 1 input**
Pin 14	**T0. Counter 0 clock input**
Pin 15	**T1. Counter 1 clock input**
Pin 16	**WR. Write to external (additional) RAM**
Pin 17	**RD. Read from external RAM**
Pin 18, 19	**XTAL2, XTAL1. Internal oscillator input and output. A quartz crystal which specifies operating frequency is usually connected to these pins.**
Pin 20	**GND. Ground.**
Pin 21-28	**Port 2. If there is no intention to use external memory then these port pins are configured as general inputs/outputs. In case external memory is used, the higher address byte, i.e. addresses A8-A15 will appear on this port. Even though memory with capacity of 64Kb is not used, which means that not all eight port bits are used for its addressing, the rest of them are not available as inputs/outputs.**
Pin 29	**PSEN. If external ROM is used for storing program then a logic zero (0) appears on it every time the microcontroller reads a byte from memory.**
Pin 30	**ALE. Prior to reading from external memory, the microcontroller puts the lower address byte (A0-A7) on P0 and activates the ALE output. After receiving signal from the ALE pin, the external latch latches the state of P0 and uses it as a**

	memory chip address. Immediately after that, the ALE pin is returned its previous logic state and P0 is now used as a Data Bus.
Pin 31	**EA. By applying logic zero to this pin, P2 and P3 are used for data and address transmission with no regard to whether there is internal memory or not. It means that even there is a program written to the microcontroller, it will not be executed. Instead, the program written to external ROM will be executed. By applying logic one to the EA pin, the microcontroller will use both memories, first internal then external (if exists).**
Pin 32-39	**PORT 0. Similar to P2, if external memory is not used, these pins can be used as general inputs/outputs. Otherwise, P0 is configured as address output (A0-A7) when the ALE pin is driven high (1) or as data output (Data Bus) when the ALE pin is driven low (0).**
Pin 40	**VCC. +5V power supply.**

2.4 MEMORY ORGANIZATION

Internal RAM organization



Bit addressable memory

General purpose memory

Working Registers

Register Banks: 00h to 1Fh. The 8051 uses 8 general-purpose registers R0 through R7 (R0, R1, R2, R3, R4, R5, R6, and R7). There are four such register banks. Selection of register bank can be done through RS1,RS0 bits of PSW. On reset, the default Register Bank 0 will be selected.

Bit Addressable RAM: 20h to 2Fh . The 8051 supports a special feature which allows access to bit variables. This is where individual memory bits in Internal RAM can be set or cleared. In all there are 128 bits numbered 00h to 7Fh. Being bit variables any one variable can have a value 0 or 1. A bit variable can be set with a command such as SETB and cleared with a command such as CLR.
Example instructions are:
SETB 25h ; sets the bit 25h (becomes 1)
CLR 25h ; clears bit 25h (becomes 0)

Note, bit 25h is actually bit 5 of Internal RAM location 24h.

The Bit Addressable area of the RAM is just 16 bytes of Internal RAM located between 20h and 2Fh.

General Purpose RAM: 30h to 7Fh. Even if 80 bytes of Internal RAM memory are available for general-purpose data storage, user should take care while using the memory location from 00 -2Fh since these locations are also the default register space, stack space, and bit addressable space. It is a good practice to use general purpose memory from 30 – 7Fh. The general purpose RAM can be accessed using direct or indirect addressing modes.

EXTERNAL MEMORY INTERFACING

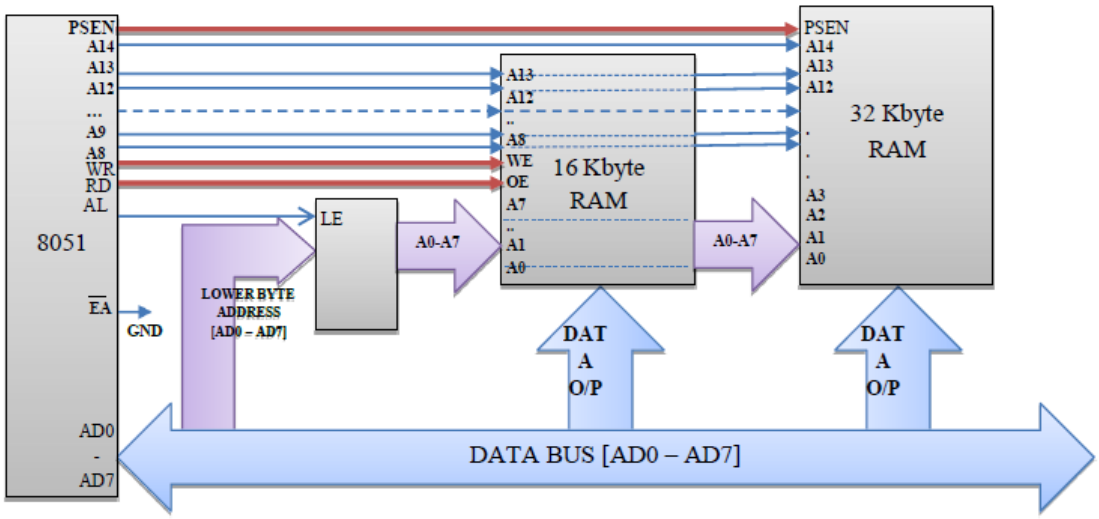
Eg. Interfacing of 16 K Byte of RAM and 32 K Byte of EPROM to 8051

Number of address lines required for ***16 Kbyte memory is 14 lines*** and that ***of 32Kbytes of memory is 15 lines.***

The connections of external memory is shown in figure. The lower order address and data bus are multiplexed. De-multiplexing is done by the latch. Initially the address will appear in the bus and this latched at the output of latch using ALE signal. The output of the latch is directly connected to the lower byte address lines of the memory. Later data will be available in this bus. Still the latch output is address it self. The higher byte of address bus is directly connected to the memory. The number of lines connected depends on the memory size.

The RD and WR (both active low) signals are connected to RAM for reading and writing the data. PSEN of microcontroller is connected to the output enable of the ROM to read the data from the memory.

EA (active low) pin is always grounded if we use only external memory. Otherwise, once the program size exceeds internal memory the microcontroller will automatically switch to external memory.



STACK

A stack is a last in first out memory. In 8051 internal RAM space can be used as stack. The address of the stack is contained in a register called stack pointer. Instructions PUSH and POP are used for stack operations. When a data is to be placed on the stack, the stack pointer increments before storing the data on the stack so that the stack grows up as data is stored (pre-increment). As the data is retrieved from the stack the byte is read from the stack, and then SP decrements to point the next available byte of stored data (post decrement). The stack pointer is set to 07 when the 8051 resets. So that default stack memory starts from address location 08 onwards (to avoid overwriting the default register bank ie., bank 0).

Eg; Show the stack and SP for the following.

```
                [SP]=07       //CONTENT OF SP IS 07 (DEFAULT VALUE)
MOV R6, #25H    [R6]=25H      //CONTENT OF R6 IS 25H
MOV R1, #12H    [R1]=12H      //CONTENT OF R1 IS 12H
MOV R4, #0F3H   [R4]=F3H      //CONTENT OF R4 IS F3H

PUSH 6          [SP]=08       [08]=[06]=25H        //CONTENT OF 08 IS 25H
PUSH 1          [SP]=09       [09]=[01]=12H        //CONTENT OF 09 IS 12H
PUSH 4          [SP]=0A       [0A]=[04]=F3H        //CONTENT OF 0A IS F3H

POP 6           [06]=[0A]=F3H       [SP]=09 //CONTENT OF 06 IS F3H
POP 1           [01]=[09]=12H       [SP]=08 //CONTENT OF 01 IS 12H
POP 4           [04]=[08]=25H       [SP]=07 //CONTENT OF 04 IS 25H
```

2.5 BASICS OF INTERRUPTS.

During program execution if peripheral devices needs service from microcontroller, device will generate interrupt and gets the service from microcontroller. When peripheral device activate the interrupt signal, the processor branches to a program called interrupt service routine. After executing the interrupt service routine the processor returns to the main program.

Steps taken by processor while processing an interrupt:

1. *It completes the execution of the current instruction.*
2. *PSW is pushed to stack.*
3. *PC content is pushed to stack.*
4. *Interrupt flag is reset.*
5. *PC is loaded with ISR address.*

ISR will always ends with RETI instruction. The execution of RETI instruction results in the following.

1. *POP the current stack top to the PC.*
2. *POP the current stack top to PSW.*

Classification of interrupts.

1. ***External and internal interrupts.***
   External interrupts are those initiated by peripheral devices through the external pins of the microcontroller.
   Internal interrupts are those activated by the internal peripherals of the microcontroller like timers, serial controller etc.)
2. ***Maskable and non-maskable interrupts.***
   The category of interrupts which can be disabled by the processor using program is called maskable interrupts.
   Non-maskable interrupts are those category by which the programmer cannot disable it using program.
3. ***Vectored and non-vectored interrupt.***
   Starting address of the ISR is called interrupt vector. In vectored interrupts the starting address is predefined. In non-vectored interrputs, the starting address is provided by the peripheral as follows.
   - Microcontroller receives an interrupt request from external device.
   - Controller sends an acknowledgement (**INTA)** after completing the execution of current instruction.
   - The peripheral device sends the interrupt vector to the microcontroller.

2.6 8051 INTERRUPT STRUCTURE.

8051 has five interrupts. They are maskable and vectored interrupts. Out of these five, two are external interrupt and three are internal interrupts.

Interrupt source	*Type*	*Vector address*	*Priority*
External interrupt 0	External	0003	Highest
Timer 0 interrupt	Internal	000B	
External interrupt 1	External	0013	
Timer 1 interrupt	Internal	001B	
Serial interrupt	Internal	0023	Lowest

8051 makes use of two registers to deal with interrupts.

6. IE Register

This is an 8 bit register used for enabling or disabling the interrupts. The structure of IE register is shown below.

IE : Interrupt Enable Register (Bit Addressable)

If the bit is 0, the corresponding interrupt is disabled. If the bit is 1, the corresponding interrupt is enabled.

EA	-	-	ES	ET1	EX1	ET0	EX0

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

7. IP Register.

This is an 8 bit register used for setting the priority of the interrupts.

IP : Interrupt Priority Register (Bit Addressable)

If the bit is 0, the corresponding interrupt has a lower priority and if the bit is the corresponding interrupt has a higher priority.

-	-	-	PS	PT1	PX1	PT0	PX0

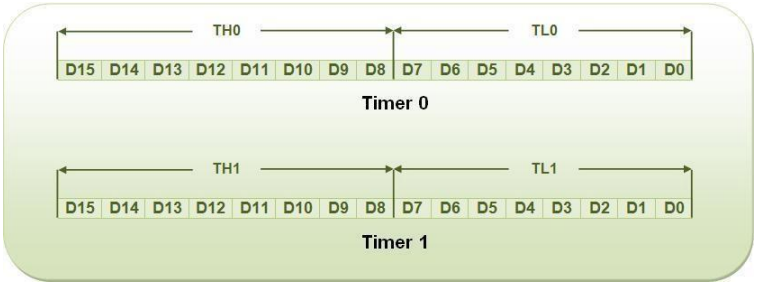
-	IP.7	Not implemented, reserved for future use*.
-	IP.6	Not implemented, reserved for future use*.
-	IP.5	Not implemented, reserved for future use*.
PS	IP.4	Defines the Serial Port interrupt priority level.
PT1	IP.3	Defines the Timer 1 Interrupt priority level.
PX1	IP.2	Defines External Interrupt priority level.
PT0	IP.1	Defines the Timer 0 interrupt priority level.
PX0	IP.0	Defines the External Interrupt 0 priority level.

2.7 TIMERS AND COUNTERS

Timers/Counters are used generally for

- Time reference
- Creating delay
- Wave form properties measurement
- Periodic interrupt generation
- Waveform generation

8051 has two timers, Timer 0 and Timer 1.



Timer in 8051 is used as timer, counter and baud rate generator. Timer always counts up irrespective of whether it is used as timer, counter, or baud rate generator: Timer is always incremented by the microcontroller. The time taken to count one digit up is based on master clock frequency.

If Master CLK=12 MHz,

Timer Clock frequency = Master CLK/12 = 1 MHz
Timer Clock Period = 1micro second

This indicates that one increment in count will take 1 micro second.

The two timers in 8051 share two SFRs (TMOD and TCON) which control the timers, and each timer also has two SFRs dedicated solely to itself (TH0/TL0 and TH1/TL1).

The following are timer related SFRs in 8051.

SFR Name	Description	SFR Address
TH0	*Timer 0 High Byte*	*8Ch*
TL0	*Timer 0 Low Byte*	*8Ah*
TH1	*Timer 1 High Byte*	*8Dh*
TL1	*Timer 1 Low Byte*	*8Bh*
TCON	*Timer Control*	*88h*
TMOD	*Timer Mode*	*89h*

TMOD Register

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

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

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

Note 1 :

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

8051 timers have both software and hardware controls. The start and stop of a timer is controlled by software using the instruction **SETB TR1** and **CLR TR1** for timer 1, and **SETB TR0** and **CLR TR0** for timer 0.

The SETB instruction is used to start it and it is stopped by the CLR instruction. These instructions start and stop the timers as long as GATE = 0 in the TMOD register. Timers can be started and stopped by an external source by making GATE = 1 in the TMOD register.

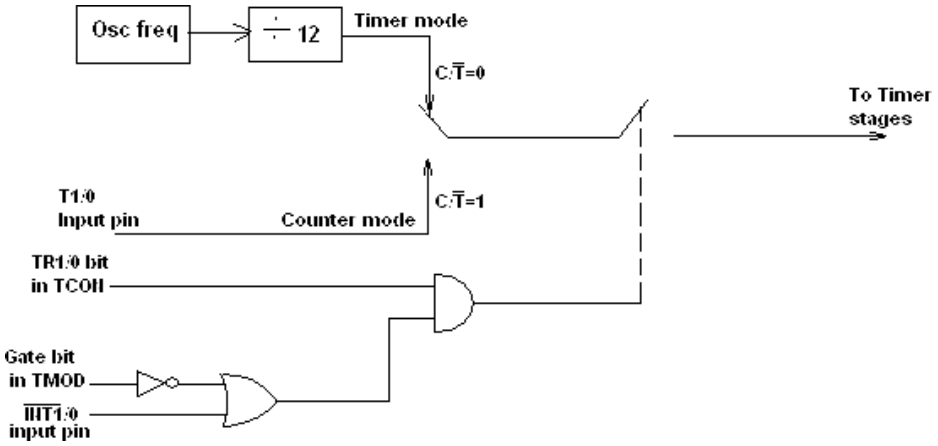
TCON Register

TCON : Timer/Counter Control Register (Bit Addressable)

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

TF1	TCON.7	Timer 1 overflow flag. Set by hardware when the Timer/Counter 1 overflows. Cleared by hardware as processor vectors to the interrupt service routine.
TR1	TCON.6	Timer 1 run control bit. Set/cleared by software to turn Timer/Counter ON/OFF.
TF0	TCON.5	Timer 0 overflow flag. Set by hardware when the Timer/Counter 0 overflows. Cleared by hardware as processor vectors to the service routine.
TR0	TCON.4	Timer 0 run control bit. Set/cleared by software to turn Timer/Counter 0 ON/OFF.
IE1	TCON.3	External Interrupt 1 edge flag. Set by hardware when External interrupt edge is detected. Cleared by hardware when interrupt is processed.
IT1	TCON.2	Interrupt 1 type control bit. Set/cleared by software to specify falling edge/flow level triggered External Interrupt.
IE0	TCON.1	External Interrupt 0 edge flag. Set by hardware when External Interrupt edge detected. Cleared by hardware when interrupt is processed.
IT0	TCON.0	Interrupt 0 type control bit. Set/cleared by software to specify falling edge/low level triggered External Interrupt.

Timer/ Counter Control Logic.



TIMER MODES

Timers can operate in four different modes. They are as follows

Timer Mode-0: In this mode, the timer is used as a 13-bit UP counter as follows.

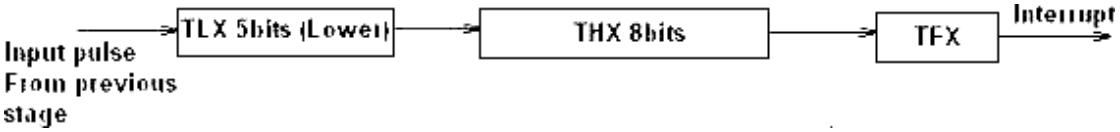


Fig. Operation of Timer on Mode-0

The lower 5 bits of TLX and 8 bits of THX are used for the 13 bit count.Upper 3 bits of TLX are ignored. When the counter rolls over from all 0's to all 1's, TFX flag is set and an interrupt is generated. The input pulse is obtained from the previous stage. If TR1/0 bit is 1 and Gate bit is 0, the counter continues counting up. If TR1/0 bit is 1 and Gate bit is 1, then the operation of the counter is controlled by input. This mode is useful to measure the width of a given pulse fed to input.

Timer Mode-1: This mode is similar to mode-0 except for the fact that the Timer operates in 16-bit mode.

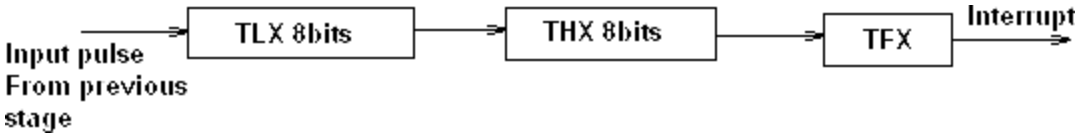


Fig: Operation of Timer in Mode 1

Timer Mode-2: (Auto-Reload Mode): This is a 8 bit counter/timer operation. Counting is performed in TLX while THX stores a constant value. In this mode when the timer overflows i.e. TLX becomes FFH, it is fed with the value stored in THX. For example if we load THX with 50H then the timer in mode 2 will count from 50H to FFH. After that 50H is again reloaded. This mode is useful in applications like fixed time sampling.

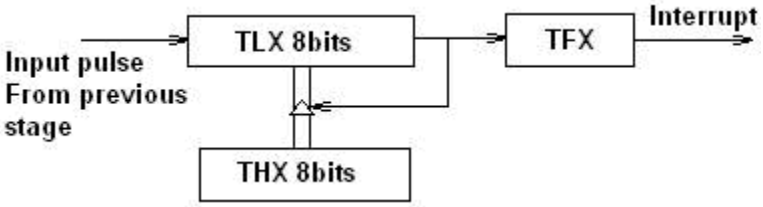


Fig: Operation of Timer in Mode 2

Timer Mode-3: Timer 1 in mode-3 simply holds its count. The effect is same as setting TR1=0. Timer0 in mode-3 establishes TL0 and TH0 as two separate counters.

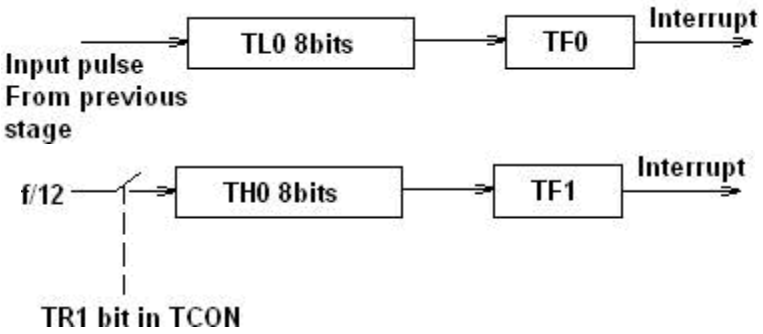


Fig: Operation of Timer in Mode 3

Control bits TR1 and TF1 are used by Timer-0 (higher 8 bits) (TH0) in Mode-3 while TR0 and TF0 are available to Timer-0 lower 8 bits(TL0).

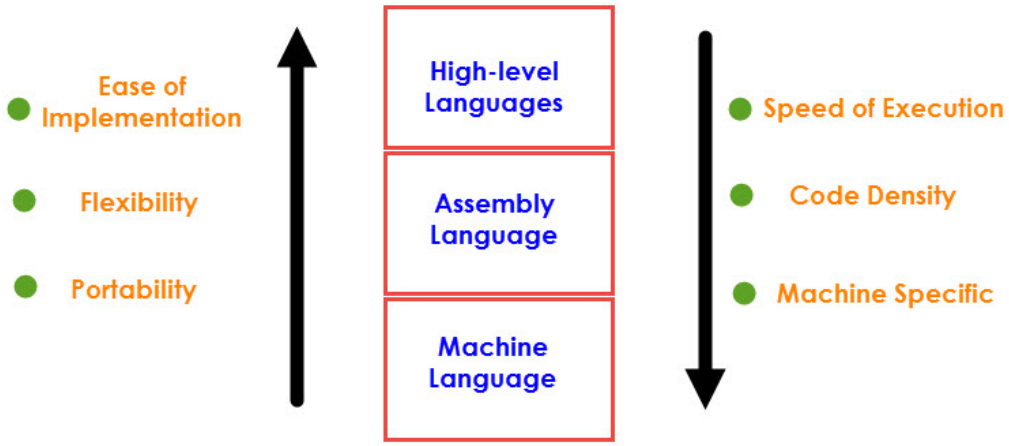
2.8 ASSEMBLY LANGUAGE PROGRAMMING

Programming in the sense of Microcontrollers (or any computer) means writing a sequence of instructions that are executed by the processor in a particular order to perform a predefined task. Programming also involves debugging and troubleshooting of instructions and instruction sequence to make sure that the desired task is performed.

Like any language, Programming Languages have certain words, grammar and rules. There are three types or levels of Programming Languages for 8051 Microcontroller. These levels are based on how closely the statements in the language resemble the operations or tasks performed by the Microcontroller.

The three levels of Programming Languages are:

- Machine Language
- Assembly Language
- High-level Language



MACHINE LANGUAGE

In Machine language or Machine Code, the instructions are written in binary bit patterns i.e. combination of binary digits 1 and 0, which are stored as HIGH and LOW Voltage Levels. This is the lowest level of programming languages and is the language that a Microcontroller or Microprocessor actually understands.

ASSEMBLY LANGUAGE

The next level of Programming Language is the Assembly Language. Since Machine Language or Code involves all the instructions in 1's and 0's, it is very difficult for humans to program using it. Assembly Language is a pseudo-English representation of the Machine Language. The 8051 Microcontroller Assembly Language is a combination of English like words called Mnemonics and Hexadecimal codes.
It is also a low level language and requires extensive understanding of the architecture of the Microcontroller.

HIGH-LEVEL LANGUAGE

The name High-level language means that you need not worry about the architecture or other internal details of a microcontroller and they use words and statements that are easily understood by humans.
Few examples of High-level Languages are BASIC, C Pascal, C++ and Java. A program called Compiler will convert the Programs written in High-level languages to Machine Code.

Why Assembly Language?

Although High-level languages are easy to work with, the following reasons point out the advantage of Assembly Language

- The Programs written in Assembly gets executed faster and they occupy less memory.

- With the help of Assembly Language, you can directly exploit all the features of a Microcontroller.
- Using Assembly Language, you can have direct and accurate control of all the Microcontroller's resources like I/O Ports, RAM, SFRs, etc.
- Compared to High-level Languages, Assembly Language has less rules and restrictions.

STRUCTURE OF THE 8051 MICROCONTROLLER ASSEMBLY LANGUAGE

The Structure or Syntax of the 8051 Microcontroller Assembly Language is discussed here. Each line or statement of the assembly language program of 8051 Microcontroller consists of three fields: Label, Instruction and Comments.

The arrangement of these fields or the order in which they appear is shown below.

```
[Label:]        Instructions        [//Comments]
```

The brackets for Label and Comments mean that these fields are optional and may not be used in all statements in a program.

Before seeing about these three fields, let us first see an example of how a typical statement or line in an 8051 Microcontroller Assembly Language looks like.

```
TESTLABEL:  MOV A, 24H  ; THIS IS A SAMPLE COMMENT
```


In the above statement, the "TESTLABEL" is the name of the Label, "MOV A, 24H" is the Instruction and the "THIS IS A SAMPLE COMMENT" is a Comment.

LABEL

The Label is programmer chosen name for a Memory Location or a statement in a program. The Label part of the statement is optional and if present, the Label must be terminated with a Colon (:). An important point to remember while selecting a name for the Label is that they should reduce the need for documentation.

INSTRUCTION

The Instruction is the main part of the 8051 Microcontroller Assembly Language Programming as it is responsible for the task performed by the Microcontroller. Any Instruction in the Assembly Language consists of two parts: Op-code and Operand(s).

The first part of the Instruction is the Op-code, which is short for Operation Code, specifies the operation to be performed by the Microcontroller. Op-codes in Assembly Language are called as Mnemonics. Op-codes are in binary format (used in Machine Language) while the Mnemonic (which are equivalent to Op-codes) are English like statements.
The second part of the instruction is called the Operand(s) and it represents the Data on which the operation is performed. There are two types of Operands: the Source Operand and the Destination Operand. The Source Operand is the Input of the operation and the Destination Operand is where the result is stored.

COMMENTS

The last part of the Structure of 8051 Assembly Language is the Comments. Comments are statements included by the developer for easier understanding of the code and is used for proper documentation of the Program.
Comments are optional and if used, they must begin with a semicolon (;) or double slash (//) depending on the Assembler.

2.9 ADDRESSING MODES

Various methods of accessing the data are called addressing modes.8051 addressing modes are classified as follows.

1. Immediate addressing.
2. Register addressing.
3. Direct addressing.
4. Indirect addressing.
5. Relative addressing.
6. Absolute addressing.
7. Long addressing.
8. Indexed addressing.
9. Bit inherent addressing.
10. Bit direct addressing.

I. IMMEDIATE ADDRESSING.

In this addressing mode the data is provided as a part of instruction itself. In other wordsdata immediately follows the instruction.

Eg. MOV A,#30H

ADD A, #83 # Symbol indicates the data is immediate.

II. REGISTER ADDRESSING.

In this addressing mode the register will hold the data. One of the eight general registers (R0 to R7) can be used and specified as the operand.

Eg. MOV A,R0
ADD A,R6

R0 – R7 will be selected from the current selection of register bank. The default register bank will be bank 0.

III. DIRECT ADDRESSING

There are two ways to access the internal memory. Using direct address and indirect address. Using direct addressing mode we can not only address the internal memory but SFRs also. In direct addressing, an 8 bit internal data memory address is specified as part of the instruction and hence, it can specify the address only in the range of 00H to FFH. In this addressing mode, data is obtained directly from the memory.

Eg. MOV A,60h
ADD A,30h

IV. INDIRECT ADDRESSING

The indirect addressing mode uses a register to hold the actual address that will be used in data movement. Registers R0 and R1 and DPTR are the only registers that can be used as data pointers. Indirect addressing cannot be used to refer to SFR registers. Both R0 and R1 can hold 8 bit address and DPTR can hold 16 bit address.

Eg. MOV A,@R0
ADD A,@R1
MOVX A,@DPTR

V. INDEXED ADDRESSING.

In indexed addressing, either the program counter (PC), or the data pointer (DTPR)—is used to hold the base address, and the A is used to hold the offset address. Adding the value of the base address to the value of the offset address forms the effective address. Indexed addressing is used with JMP or MOVC instructions. Look up tables are easily implemented with the help of index addressing.

Eg. MOVC A, @A+DPTR // *copies the contents of memory location pointed by the sum of the accumulator A and the DPTR into accumulator A.*

MOVC A, @A+PC // *copies the contents of memory location pointed by the sum of the accumulator A and the program counter into accumulator A.*

VI. RELATIVE ADDRESSING.

Relative addressing is used only with conditional jump instructions. The relative address, (offset), is an 8 bit signed number, which is automatically added to the PC to make the address of the next instruction. The 8 bit signed offset value gives an address range of +127 to —128 locations. The jump destination is usually specified using a label and the assembler calculates the jump offset accordingly. The advantage of relative addressing is that the program code is easy to relocate and the address is relative to position in the memory.

Eg. SJMP LOOP1
JC BACK

VII. ABSOLUTE ADDRESSING

Absolute addressing is used only by the AJMP (Absolute Jump) and ACALL (Absolute Call) instructions. These are 2 bytes instructions. The absolute addressing mode specifies the lowest 11 bit of the memory address as part of the instruction. The upper 5 bit of the destination address are the upper 5 bit of the current program counter. Hence, absolute addressing allows branching onlywithin the current 2 Kbyte page of the program memory.

Eg. AJMP LOOP1
ACALL LOOP2

VIII. LONG ADDRESSING

The long addressing mode is used with the instructions LJMP and LCALL. These are 3 byte instructions. The address specifies a full 16 bit destination address so that a jump or a call can be made to a location within a 64 Kbyte code memory space.

Eg. LJMP FINISH
LCALL DELAY

IX. BIT INHERENT ADDRESSING

In this addressing, the address of the flag which contains the operand, is implied in the opcode of the instruction.

Eg. CLR C ; *Clears the carry flag to 0*

X. BIT DIRECT ADDRESSING

In this addressing mode the direct address of the bit is specified in the instruction. The

RAM space 20H to 2FH and most of the special function registers are bit addressable. Bit address values are between 00H to 7FH.

Eg. CLR 07h ; Clears the bit 7 of 20h RAM space

SETB 07H ; Sets the bit 7 of 20H RAM space.

2.10 INSTRUCTION SET

1. INSTRUCTION TIMINGS

The 8051 internal operations and external read/write operations are controlled by the oscillator clock.

T-state, Machine cycle and Instruction cycle are terms used in instruction timings.

T-state is defined as one subdivision of the operation performed in one clock period. The terms 'T-state' and 'clock period' are often used synonymously.

Machine cycle is defined as 12 oscillator periods. A machine cycle consists of six states and each state lasts for two oscillator periods. An instruction takes one to four machine cycles to execute an instruction. ***Instruction cycle*** is defined as the time required for completing the execution of an instruction. The 8051 instruction cycle consists of one to four machine cycles.

Eg. If 8051 microcontroller is operated with 12 MHz oscillator, find the execution time for the following four instructions.

1. *ADD A, 45H*
2. *SUBB A, #55H*
3. *MOV DPTR, #2000H*
4. *MUL AB*

Since the oscillator frequency is 12 MHz, the clock period is, Clock period = 1/12 MHz = ***0.08333 µS****.*

*Time for 1 machine cycle = 0.08333 µS x 12 =****1 µS****.*

Instruction	*No. of machine cycles*	*Execution time*
1. *ADD A, 45H*	*1*	*1 µs*
2. *SUBB A, #55H*	*2*	*2 µs*
3. *MOV DPTR, #2000H*	*2*	*2 µs*
4. *MUL AB*	*4*	*4 µs*

2. 8051 INSTRUCTIONS

The instructions of 8051 can be broadly classified under the following headings.

- Data transfer instructions
- Arithmetic instructions
- Logical instructions
- Branch instructions
- Subroutine instructions
- Bit manipulation instructions

Data transfer instructions.

In this group, the instructions perform data transfer operations of the following types.

a. Move the contents of a register Rn to A
   i. MOV A,R2
   ii. MOV A,R7

b. Move the contents of a register A to Rn
   i. MOV R4,A
   ii. MOV R1,A

c. Move an immediate 8 bit data to register A or to Rn or to a memory location(direct orindirect)
   i. MOV A, #45H
   ii. MOV R6, #51H
   iii. MOV 30H, #44H

d. Move the contents of a memory location to A or A to a memory location using direct andindirect addressing
   i. MOV A, 65H
   ii. MOV A, @R0
   iii. MOV 45H, A
   iv. MOV @R1, A

e. Move the contents of a memory location to Rn or Rn to a memory location using directaddressing
   i. MOV R3, 65H
   ii. MOV 45H, R2

f. Move the contents of memory location to another memory location using direct andindirect addressing
   i. MOV 47H, 65H
   ii. MOV 45H, @R0

g. Move the contents of an external memory to A or A to an external memory
   i. MOVX A,@R1
   ii. MOVX @R0,A
   iii. MOVX A,@DPTR
   iv. MOVX@DPTR,A

h. Move the contents of program memory to A
   i. MOVC A, @A+PC
   ii. MOVC A, @A+DPTR

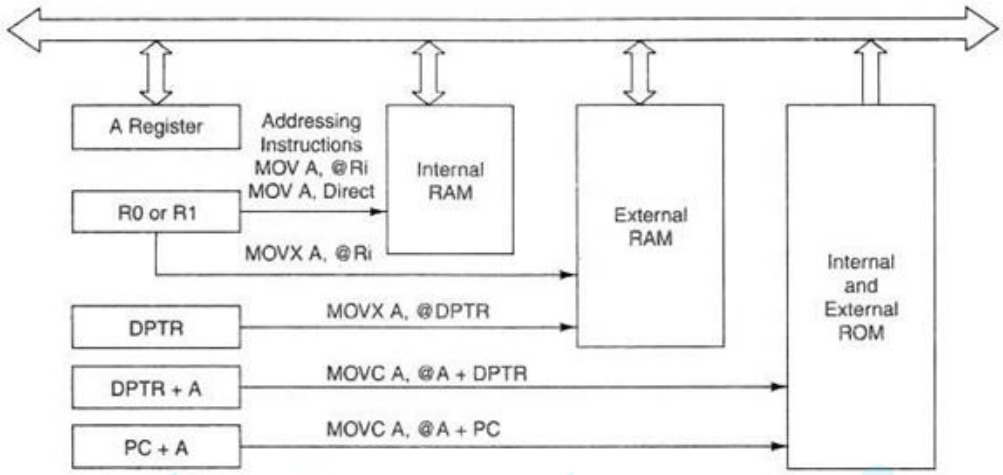


FIG. Addressing Using MOV, MOVX and MOVC

i. Push and Pop instructions

[SP]=07 //CONTENT OF SP IS 07 (DEFAULT VALUE)

MOV R6, #25H [R6]=25H //CONTENT OF R6 IS 25H

MOV R1, #12H [R1]=12H //CONTENT OF R1 IS 12H

MOV R4, #0F3H [R4]=F3H //CONTENT OF R4 IS F3H

PUSH 6 [SP]=08 [08]=[06]=25H //CONTENT OF 08 IS 25H

PUSH 1 [SP]=09 [09]=[01]=12H //CONTENT OF 09 IS 12H PUSH 4
[SP]=0A [0A]=[04]=F3H //CONTENT OF 0A IS F3H

POP 6 [06]=[0A]=F3H [SP]=09 //CONTENT OF 06 IS F3H

POP 1 [01]=[09]=12H [SP]=08 //CONTENT OF 01 IS 12H

POP 4 [04]=[08]=25H [SP]=07 //CONTENT OF 04 IS 25H

j. Exchange instructions

The content of source ie., register, direct memory or indirect memory will be exchangedwith the contents of destination ie., accumulator.

i. XCH A,R3
ii. XCH A,@R1
iii. XCH A,54h

k. Exchange digit. Exchange the lower order nibble of Accumulator (A0-A3) with lower order nibble of the internal RAM location which is indirectly addressed by the register.

i. XCHD A,@R1
ii. XCHD A,@R0

Arithmetic instructions.

The 8051 can perform addition, subtraction. Multiplication and division operations on 8 bitnumbers.

Addition

In this group, we have instructions to

i. Add the contents of A with immediate data with or without carry.
   i. ADD A, #45H
   ii. ADDC A, #OB4H
ii. Add the contents of A with register Rn with or without carry.
   i. ADD A, R5
   ii. ADDC A, R2
iii. Add the contents of A with contents of memory with or without carry using direct andindirect addressing
   i. ADD A, 51H
   ii. ADDC A, 75H
   iii. ADD A, @R1
   iv. ADDC A, @R0

CY AC and OV flags will be affected by this operation

Subtraction

In this group, we have instructions to

i. Subtract the contents of A with immediate data with or without carry.
   i. SUBB A, #45H
   ii. SUBB A, #OB4H
ii. Subtract the contents of A with register Rn with or without borrow.
   i. SUBB A, R5
   ii. SUBB A, R2
iii. Subtract the contents of A with contents of memory with or without carry using direct andindirect addressing
   i. SUBB A, 51H

ii. SUBB A, 75H
iii. SUBB A, @R1
iv. SUBB A, @R0

CY AC and OV flags will be affected by this operation.Multiplication

MUL AB. This instruction multiplies two 8 bit unsigned numbers which are stored in A and B register. After multiplication the lower byte of the result will be stored in accumulator and higher byte of result will be stored in B register.

```
MOV A,#45H      ;[A]=45H
MOV B,#0F5H     ;[B]=F5H
MUL AB          ;[A] x [B] = 45 x F5 = 4209
                ;[A]=09H, [B]=42H
```

Division

DIV AB. This instruction divides the 8 bit unsigned number which is stored in A by the 8 bit unsigned number which is stored in B register. After division the result will be stored in accumulator and remainder will be stored in B register.

```
Eg.  MOV A,#45H     ;[A]=0E8H
     MOV B,#0F5H    ;[B]=1BH
     DIV AB         ;[A] / [B] = E8 /1B = 08 H with remainder 10H
                    ;[A] = 08H, [B]=10H
```

DA A (Decimal Adjust After Addition).

When two BCD numbers are added, the answer is a non-BCD number. To get the result in BCD, weuse DA A instruction after the addition. DA A works as follows.

- If lower nibble is greater than 9 or auxiliary carry is 1, 6 is added to lower nibble.
- If upper nibble is greater than 9 or carry is 1, 6 is added to upper nibble.

```
Eg 1:  MOV A,#23H
       MOV R1,#55H
       ADD A,R1     // [A]=78
       DA A         // [A]=78     no changes in the accumulator after DA A

Eg 2:  MOV A,#53H
       MOV R1,#58H
       ADD A,R1     // [A]=ABh
       DA A         // [A]=11, C=1 . ANSWER IS 111. Accumulator data is
                    changed after DA A
```

Increment: *increments the operand by one.*

INC A INC Rn INC DIRECT INC @RiINC DPTR

INC increments the value of source by 1. If the initial value of register is FFh, incrementing the value will cause it to reset to 0. The Carry Flag is not set when the value "rolls over" from 255 to 0.

In the case of "INC DPTR", the value two-byte unsigned integer value of DPTR is incremented. If the initial value of DPTR is FFFFh, incrementing the value will cause it to reset to 0.

Decrement: *decrements the operand by one.*

DEC A DEC Rn DEC DIRECT DEC @Ri

DEC decrements the value of *source* by 1. If the initial value of is 0, decrementing the value will cause it to reset to FFh. The Carry Flag is not set when the value "rolls over" from 0 to FFh.

Logical Instructions

Logical AND

ANL destination, source: ANL does a bitwise "AND" operation between *source* and *destination*,leaving the resulting value in *destination*. The value in source is not affected. "AND" instruction logically AND the bits of source and destination.

ANL A,#DATA
ANL A, Rn
ANL A,DIRECT
ANL A,@Ri

ANL DIRECT,A ANL DIRECT, #DATA

Logical OR

ORL destination, source:ORL does a bitwise "OR" operation between *source* and *destination*,

leaving the resulting value in *destination*. The value in source is not affected. " OR " instructionlogically OR the bits of source and destination.

ORL A,#DATA
ORL A, Rn
ORL

A,DIRECT
ORL A,@Ri

ORL DIRECT,A ORL DIRECT, #DATA

Logical Ex-OR

XRL destination, source: XRL does a bitwise "EX-OR" operation between *source* and *destination*, leaving the resulting value in *destination*. The value in source is not affected. " XRL "instruction logically EX-OR the bits of source and destination.

XRL
A,#DATA XRL A,Rn XRL A,DIRECT XRL A,@Ri

XRL DIRECT,A XRL DIRECT, #DATA

Logical NOT

CPL complements *operand*, leaving the result in *operand*. If *operand* is a single bit then the state ofthe bit will be reversed. If *operand* is the Accumulator then all the bits in the Accumulator will be reversed.

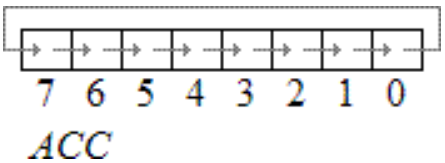
CPL A, CPL C, CPL bit address

SWAP A – Swap the upper nibble and lower nibble of A.

Rotate Instructions

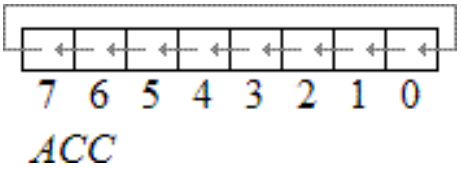
RR A

This instruction is rotate right the accumulator. Its operation is illustrated below. Each bit is shifted one location to the right, with bit 0 going to bit 7.



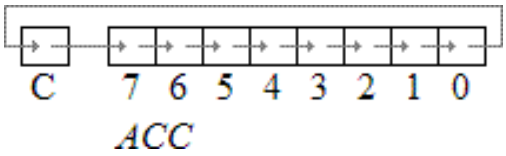
RL A

Rotate left the accumulator. Each bit is shifted one location to the left, with bit 7 going to bit 0



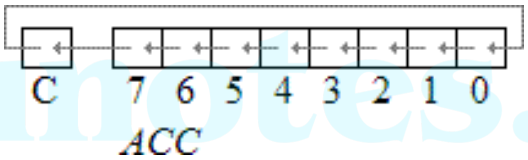
RRC A

Rotate right through the carry. Each bit is shifted one location to the right, with bit going into the carry bit in the PSW, while the carry was at goes into bit 7



RLC A

Rotate left through the carry. Each bit is shifted one location to the left, with bit 7 going into the carry bit in the PSW, while the carry goes into bit 0.



Branch (JUMP) Instructions

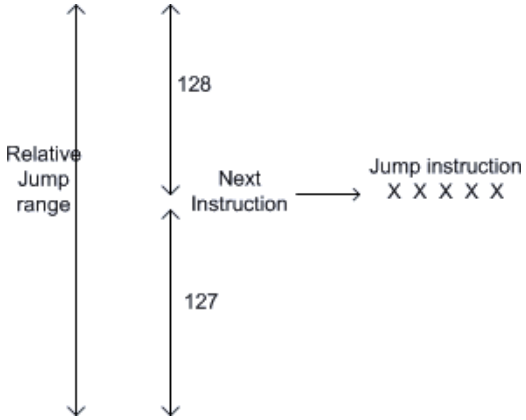
Jump and Call Program Range

There are 3 types of jump instructions. They are:-

1. Relative Jump
2. Short Absolute Jump
3. Long Absolute Jump

Relative Jump

Jump that replaces the PC (program counter) content with a new address that is greater than (the address following the jump instruction by 127 or less) or less than (the address following the jump by -128 or less) is called a relative jump. Schematically, the relative jump can be shown as follows: -



The advantages of the relative jump are as follows:-

1. Only 1 byte of jump address needs to be specified in the 2's complement form, ie. Forjumping ahead, the range is 0 to 127 and for jumping back, the range is -1 to -128.
2. Specifying only one byte reduces the size of the instruction and speeds up programexecution.
3. The program with relative jumps can be relocated without reassembling to generateabsolute jump addresses.

Disadvantages of the absolute jump: -

1. Short jump range (-128 to 127 from the instruction following the jump instruction)

Instructions that use Relative Jump

SJMP <relative address>; *this is unconditional jump*

The remaining relative jumps are conditional jumps

JC <relative address> JNC <relative address>
JB bit, <relative address>
JNB bit, <relative address>
JBC bit, <relative address>
CJNE <destination byte>, <source
byte>, <relative address>
DJNZ <byte>, <relative address>
JZ <relative address> JNZ <relative address>

Short Absolute Jump

In this case only 11bits of the absolute jump address are needed. The absolute jump address iscalculated in the following manner.

In 8051, 64 kbyte of program memory space is divided into 32 pages of 2 kbyte each. The hexadecimal addresses of the pages are given as follows:-

Page (Hex)	*Address (Hex)*
00	*0000 - 07FF*
01	*0800 - 0FFF*
02	*1000 - 17FF*
03	*1800 - 1FFF*
.	
.	
1E	*F000 - F7FF*
1F	*F800 - FFFF*

It can be seen that the upper 5bits of the program counter (PC) hold the page number and the lower11bits of the PC hold the address within that page. Thus, an absolute address is formed by taking page numbers of the instruction (from the program counter) following the jump and attaching the specified 11bits to it to form the 16-bit address.

Advantage: The instruction length becomes 2 bytes.

Example of short absolute jump: -

ACALL <address 11>

AJMP <address 11>

Long Absolute Jump/Call

Applications that need to access the entire program memory from 0000H to FFFFH use long absolute jump. Since the absolute address has to be specified in the op-code, the instruction length is 3 bytes (except for JMP @ A+DPTR). This jump is not re-locatable.

Example: -

LCALL <address 16>
LJMP <address 16>
JMP @A+DPTR

Another classification of jump instructions is

1. Unconditional Jump
2. Conditional Jump

1. **The unconditional jump** is a jump in which control is transferred unconditionally to the target location.
   a. **LJMP** (long jump). This is a 3-byte instruction. First byte is the op-code and second and third bytes represent the 16-bit target address which is any memory location from 0000 to FFFFH
   eg: LJMP 3000H

b. **AJMP:** this causes unconditional branch to the indicated address, by loading the 11 bit address to 0 -10 bits of the program counter. The destination must be therefore within the same 2K blocks.

c. **SJMP** (short jump). This is a 2-byte instruction. First byte is the op-code and second byte is the relative target address, 00 to FFH (forward +127 and backward -128 bytes from the current PC value). To calculate the target address of a short jump, the second byte is added to the PC value which is address of the instruction immediately below the jump.

2. **Conditional Jump instructions.**

JBC	Jump if bit =1 and clear bit
JNB	Jump if bit =0
JB	Jump if bit =1
JNC	Jump if CY =0
JC	Jump if CY =1
CJNE reg,#data	Jump if byte ≠ #data
CJNE A,byte	Jump if A ≠ byte
DJNZ	Decrement and Jump if A ≠ 0
JNZ	Jump if A ≠ 0
JZ	Jump if A =0

All conditional jumps are short jumps.

Bit level jump instructions:

Bit level JUMP instructions will check the conditions of the bit and if condition is true, it jumps to theaddress specified in the instruction. All the bit jumps are relative jumps.

JB bit, rel ; jump if the direct bit is set to the relative address specified.
JNB bit, rel ; jump if the direct bit is clear to the relative address specified.
JBC bit, rel ; jump if the direct bit is set to the relative address specified and then clear the bit.

Subroutine CALL And RETURN Instructions

Subroutines are handled by CALL and RET instructions
There are two types of CALL instructions

1. **LCALL address(16 bit)**
   This is long call instruction which unconditionally calls the subroutine located at the indicated 16 bit address. This is a 3 byte instruction. The LCALL instruction works as follows.

   a. During execution of LCALL, [PC] = [PC]+3; (if address where LCALL resides is say, 0x3254;during execution of this instruction [PC] = 3254h + 3h = 3257h
   b. [SP]=[SP]+1; (if SP contains default value 07, then SP increments and [SP]=08

c. [[SP]] = [PC7-0]; (lower byte of PC content ie., 57 will be stored in memory location 08.
d. [SP]=[SP]+1; (SP increments again and [SP]=09)
e. [[SP]] = [PC15-8]; (higher byte of PC content ie., 32 will be stored in memory location 09.
With these the address (0x3254) which was in PC is stored in stack.
f. [PC]= address (16 bit); the new address of subroutine is loaded to PC. No flags are affected.

2. **ACALL address(11 bit)**

This is absolute call instruction which unconditionally calls the subroutine located at the indicated 11 bit address. This is a 2 byte instruction. The SCALL instruction works as follows.

a. During execution of SCALL, [PC] = [PC]+2; (if address where LCALL resides is say, 0x8549;during execution of this instruction [PC] = 8549h + 2h = 854Bh
b. [SP]=[SP]+1; (if SP contains default value 07, then SP increments and [SP]=08
c. [[SP]] = [PC7-0]; (lower byte of PC content ie., 4B will be stored in memory location 08.
d. [SP]=[SP]+1; (SP increments again and [SP]=09)
e. [[SP]] = [PC15-8]; (higher byte of PC content ie., 85 will be stored in memory location 09.

With these the address (0x854B) which was in PC is stored in stack.

f. [PC10-0]= address (11 bit); the new address of subroutine is loaded to PC. No flags are affected.

RET instruction

RET instruction pops top two contents from the stack and load it to PC.

g. [PC15-8] = [[SP]] ;content of current top of the stack will be moved to higher byte of PC.
h. [SP]=[SP]-1; (SP decrements)
i. [PC7-0] = [[SP]] ;content of bottom of the stack will be moved to lower byte of PC.
j. [SP]=[SP]-1; (SP decrements again)

Bit manipulation instructions.

8051 has 128 bit addressable memory. Bit addressable SFRs and bit addressable PORT pins. It is possible to perform following bit wise operations for these bit addressable locations.

1. LOGICAL AND
   a. ANL C,BIT(BIT ADDRESS) ; 'Logically and' carry and content of bit address, store result in carry
   b. ANL C, /BIT; ; 'Logically and' carry and complement of content of bit address, store result in carry
2. LOGICAL OR
   a. ORL C,BIT(BIT ADDRESS) ; 'Logically or' carry and content of bit address, store result in

carry

b. ORL C, /BIT; ; 'Logically or' carry and complement of content of bit address, store result in carry

3. CLR bit
   a. CLR bit ; Content of bit address specified will be cleared.
   b. CLR C ; Content of carry will be cleared.
4. CPL bit
   a. CPL bit ; Content of bit address specified will be complemented.
   b. CPL C ; Content of carry will be complemente