SERIAL COMMUNICATION.

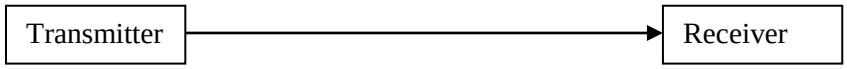
DATA COMMUNICATION

The 8051 microcontroller is parallel device that transfers eight bits of data simultaneously over eight data lines to parallel I/O devices. Parallel data transfer over a long is very expensive. Hence, a serial communication is widely used in long distance communication. In serial data communication, 8-bit data is converted to serial bits using a parallel in serial out shift register and then it is transmitted over a single data line. The data byte is always transmitted with least significant bit first.

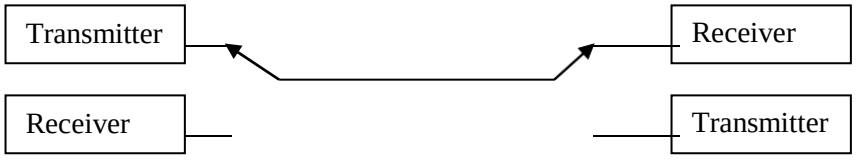
BASICS OF SERIAL DATA COMMUNICATION,

Communication Links

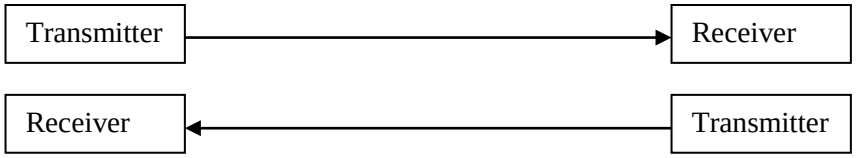
1. Simplex communication link: In simplex transmission, the line is dedicated for transmission. The transmitter sends and the receiver receives the data.



2. Half duplex communication link: In half duplex, the communication link can be used for either transmission or reception. Data is transmitted in only one direction at a time.



3. Full duplex communication link: If the data is transmitted in both ways at the same time, it is a full duplex i.e. transmission and reception can proceed simultaneously. This communication link requires two wires for data, one for transmission and one for reception.

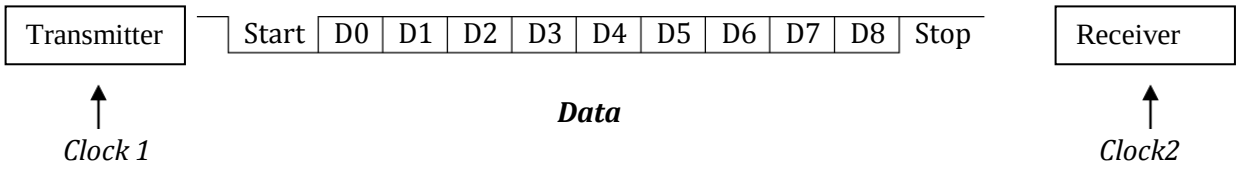


Types of Serial communication:

Serial data communication uses two types of communication.

1. Synchronous serial data communication: In this transmitter and receiver are synchronized. It uses a common clock to synchronize the receiver and the transmitter. First the synch character is sent and then the data is transmitted. This format is generally used for high speed transmission. In Synchronous serial data communication a block of data is transmitted at a time.

2. Asynchronous Serial data transmission: In this, different clock sources are used for transmitter and receiver. In this mode, data is transmitted with start and stop bits. A transmission begins with start bit, followed by data and then stop bit. For error checking purpose parity bit is included just prior to stop bit. In Asynchronous serial data communication a single byte is transmitted at a time.



Baud rate:

The rate at which the data is transmitted is called baud or transfer rate. The baud rate is the reciprocal of the time to send one bit. In asynchronous transmission, baud rate is not equal to number of bits per second. This is because; each byte is preceded by a start bit and followed by parity and stop bit. For example, in synchronous transmission, if data is transmitted with 9600 baud, it means that 9600 bits are transmitted in one second. For bit transmission time = 1 second/ 9600 = 0.104 ms.

Over the years, dozens of serial protocols have been crafted to meet particular needs of embedded systems.

1. USB (universal serial bus), and Ethernet, are a couple of the well- known computing serial interfaces.
2. Other very common serial interfaces include SPI, I2C, RS-232 and so on.
3. Each of these serial interfaces can be sorted into one of two groups: synchronous or asynchronous.
4. A Synchronous serial interface always pairs its data line(s) with a clock signal, so all devices on a synchronous serial bus share a common clock. This often allows faster serial transfer, but it also requires at least one extra wire between communicating devices. Examples of synchronous interfaces include SPI, and I2C. Asynchronous means that data is transferred without support from an external clock signal. Minimizes the required wires and I/O pins, but we need to put some extra effort into reliably transferring and receiving data.
5. The Universal Asynchronous Receiver/Transmitter (UART) controller is the key component of the serial communications subsystem. UART is also a common integrated feature in most microcontrollers. The UART takes bytes of data and transmits the individual bits in a sequential fashion. At the destination, a second UART re-assembles the bits into complete bytes. Communication can be “full duplex” (both send and receive at the same time) or “half duplex” (devices take turns transmitting and receiving).
6. The Universal Synchronous/Asynchronous Receiver/Transmitter (USART) controller is another implementation of the serial port. UART supports only asynchronous mode, whereas USART supports both asynchronous and synchronous modes. Unlike Ethernet, Firewire etc., there is no specific port for UART/USART. They are commonly used in conjugation with protocols like RS- 232, RS-434 etc.

Rules for UART

The asynchronous serial protocol has a number of built-in rules - mechanisms that help ensure robust and error-free data transfers. These mechanisms compensate the non-existence of the common clock signal:

Baud Rate: The baud rate specifies how fast data is sent over a serial line.

- It's usually expressed in units of bits-per-second (bps).
- This value determines how long the transmitter holds a serial line high/low or at what period the receiving device samples it's line.
- Baud rates can be just about any value within reason. The only requirement is that both devices operate at the same rate.
- Common baud rate is 9600 bps. Other "standard" baud are 1200, 2400, 4800, 19200, 38400, 57600, and 115200.

Framing the data

- Each block (usually a byte) of data transmitted is actually sent in a packet or frame of bits.
- Frames are created by appending synchronization and parity bits to our data.
- Some symbols in the frame have configurable bit sizes.
- Data chunk :The amount of data in each packet can be set to anything from 5 to 9 bits. Certainly, the standard data size is your basic 8-bit byte, but other sizes have their uses. A 7-bit data chunk can be more efficient than 8, especially if you're just transferring 7-bit ASCII characters. After agreeing on a character-length, both serial devices also have to agree on the endianness of their data. Is data sent most-significant bit (msb) to least, or vice-versa? If it's not otherwise stated, you can usually assume that data is transferred least-significant bit (lsb) first.

- Synchronization bits: The synchronization bits are two or three special bits transferred with each chunk of data. They are the start bit and the stop bit(s), and mark the beginning and end of a packet. There is always only one start bit, but the number of stop bits is configurable to either one or two (though it's commonly left at one). The start bit is always indicated by an idle data line going from 1 to 0, while the stop bit(s) will transition back to the idle state by holding the line at 1.
- Parity bits: Parity is optional, and not very widely used. It can be helpful for transmitting across noisy mediums, but it'll also slow down data transfer a bit and requires both sender and receiver to implement error-handling (usually, received

data that fails must be re-sent).

A variety of communication protocols have been developed based on serial communication in the past few decades. Some of them are:

- SPI – Serial Peripheral Interface: It is a three-wire based communication system. One wire each for Master to slave and Vice-versa, and one for clock pulses. There is an additional SS (Slave Select) line, which is mostly used when we want to send/receive data between multiple ICs.
- I2C – Inter-Integrated Circuit : Pronounced eye-two-see or eye-square-see, this is an advanced form of USART. The transmission speeds can be as high as 400KHz. The I2C bus has two wires – one for clock, and the other is the data line, which is bi-directional – this being the reason it is also sometimes (not always – there are a few conditions) called Two Wire Interface (TWI). It is a new and revolutionary technology invented by Philips.
- FireWire – Developed by Apple. High-speed buses capable of audio/video transmission. The bus contains a number of wires depending upon the port, which can be either a 4-pin one, or a 6-pin one, or an 8-pin one.
- Ethernet : Used mostly in LAN connections, the bus consists of 8 lines, or 4 Tx/Rx pairs.
- Universal serial bus (USB). Most popular of all serial interfaces. Is used for virtually all type of connections. The bus has 4 lines: VCC, Ground, Data+, and Data-.

8051 SERIAL COMMUNICATION

The 8051 supports a full duplex serial port.
Three special function registers support serial communication.

1. SBUF Register: Serial Buffer (SBUF) register is an 8-bit register. It has separate SBUF registers for data transmission and for data reception. For a byte of data to be transferred via the TXD line, it must be placed in SBUF register. Similarly, SBUF holds the 8-bit data received by the RXD pin and read to accept the received data.
2. SCON register: The contents of the Serial Control (SCON) register are shown below. This register contains mode selection bits, serial port interrupt bit (TI and RI) and also the ninth data bit for transmission and reception (TB8 and RB8).

Serial Port Control (SCON) Register							
D7	D6	D5	D4	D3	D2	D1	D0
SM0	**SM1**	**SM2**	**REN**	**TB8**	**RB8**	**TI**	**RI**

- SM0 (SCON.7) : Serial communication mode selection bit
- SM1 (SCON.6) : Serial communication mode selection bit

SM0	SM1	Mode	Description	Baud rate
0	0	Mode 0	8-bit shift register mode	Fosc / 12
0	1	Mode 1	8-bit UART	Variable (set by timer 1)
1	0	Mode 2	9-bit UART	Fosc/ 32 or Fosc/64
1	1	Mode 3	9-bit UART	Variable (set by timer 1)

- SM2 (SCON.5) : Multiprocessor communication bit. In modes 2 and 3, if set this will enable multiprocessor communication.
- REN (SCON.4) : Enable serial reception
- TB8 (SCON.3) : This is 9th bit that is transmitted in mode 2 & 3.
- RB8 (SCON.2) : 9th data bit is received in modes 2 & 3.
- TI (SCON.1) : Transmit interrupt flag, set by hardware must be cleared by software.
- RI (SCON.0) : Receive interrupt flag, set by hardware must be cleared by software.

3. PCON register: The SMOD bit (bit 7) of PCON register controls the baud rate in asynchronous mode transmission.

Power mode Control (PCON) Register							
D7	D6	D5	D4	D3	D2	D1	D0
SMOD	**--**	**--**	**--**	**GF1**	**GF0**	**PD**	**IDL**

- SMD (PCON.7) : Serial rate modify bit. Set to 1 by program to double baud rate using timer 1 for modes 1, 2, and 3. cleared by program to use timer 1 baud rate.
- GF1 (PCON.3) : General Purpose user flag bit.
- GF0 (PCON.2) : General Purpose user flag bit.
- PD (PCON.1) : Power down bit. Set to 1 by program to enter power down configuration for CHMOS processors.
- IDL (PCON.0) : Idle mode bit. Set to 1 by program to enter idle mode configuration for CHMOS processors.

SERIAL COMMUNICATION MODES

Mode 0

In this mode serial port runs in synchronous mode. The data is transmitted and received through RXD pin and TXD is used for clock output. In this mode the baud rate is 1/12 of clock frequency.

Mode 1

In this mode SBUF becomes a 10 bit full duplex transceiver. The ten bits are 1 start bit, 8 data bit and 1 stop bit. The interrupt flag TI/RI will be set once transmission or reception is over. In this mode the baud rate is variable and is determined by the timer 1 overflow rate.

Baud rate = $[2^{smod}/32]$ x Timer 1 overflow Rate

= $[2^{smod}/32]$ x [Oscillator Clock Frequency] / [12 x [256 – [TH1]]]

Mode 2

This is similar to mode 1 except 11 bits are transmitted or received. The 11 bits are, 1 start bit, 8 data bit, a programmable 9th data bit, 1 stop bit.

Baud rate = $[2^{smod}/64]$ x Oscillator Clock Frequency

Mode 3

This is similar to mode 2 except baud rate is calculated as in mode 1

CONNECTIONS TO RS-232

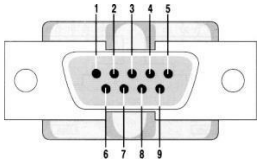
RS-232 standards

To allow compatibility among data communication equipment made by various manufactures, an interfacing standard called RS232 was set by the Electronics Industries Association (EIA) in 1960. Since the standard was set long before the advent of logic family, its input and output voltage levels are not TTL compatible.

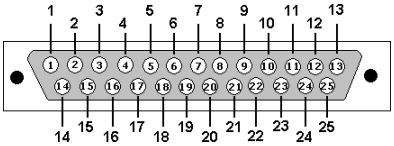
Microcontrollers and other low-level ICs communicate serially at (Transistor-Transistor-Logic) levels which exist between a microcontroller's voltage supply range - usually 0V to 3.3V or 5V.

In RS232, a logic one (1) is represented by -3 to -25V and referred as MARK while logic zero (0) is represented by +3 to +25V and referred as SPACE. For this reason to connect any RS232 to a microcontroller system we must use voltage converters such as MAX232 to convert the TTL logic level to RS232 voltage levels and vice-versa. MAX232 IC chips are commonly referred as line drivers.

In RS232 standard we use two types of connectors. DB9 connector or DB25 connector.



DB9 Male Connector



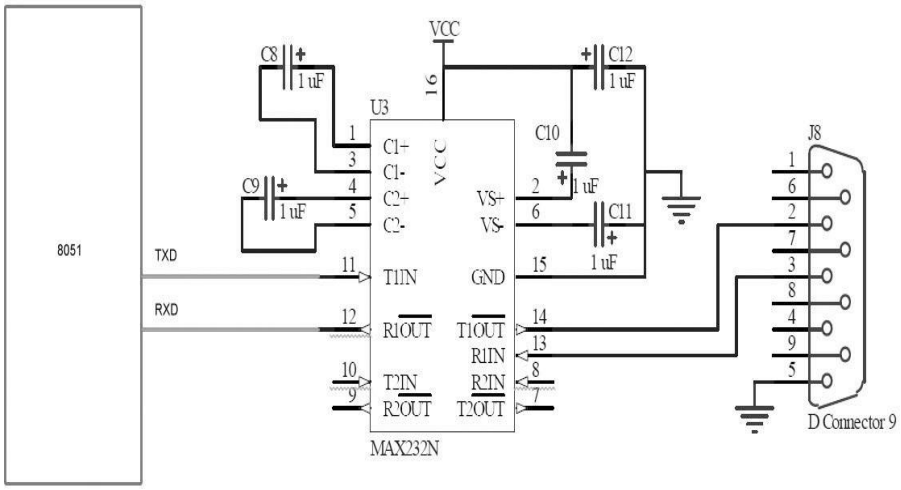
DB25 Male Connector

The pin description of DB9 and DB25 Connectors are as follows

DB-25 Pin No.	DB-9 Pin No.	Abbreviation	Full Name
Pin 2	Pin 3	TD	Transmit Data
Pin 3	Pin 2	RD	Receive Data
Pin 4	Pin 7	RTS	Request To Send
Pin 5	Pin 8	CTS	Clear To Send
Pin 6	Pin 6	DSR	Data Set Ready
Pin 7	Pin 5	SG	Signal Ground
Pin 8	Pin 1	CD	Carrier Detect
Pin 20	Pin 4	DTR	Data Terminal Ready
Pin 22	Pin 9	RI	Ring Indicator

The 8051 connection to MAX232 is as follows.

The 8051 has two pins that are used specifically for transferring and receiving data serially. These two pins are called TXD, RXD. Pin 11 of the 8051 (P3.1) assigned to TXD and pin 10 (P3.0) is designated as RXD. These pins TTL compatible; therefore they require line driver (MAX 232) to make them RS232 compatible. MAX 232 converts RS232 voltage levels to TTL voltage levels and vice versa. One advantage of the MAX232 is that it uses a +5V power source which is the same as the source voltage for the 8051. The typical connection diagram between MAX 232 and 8051 is shown below.



SERIAL COMMUNICATION PROGRAMMING IN ASSEMBLY AND C.

Steps to programming the 8051 to transfer data serially

4. The TMOD register is loaded with the value 20H, indicating the use of the Timer 1 in mode 2 (8-bit auto reload) to set the baud rate.
5. The TH1 is loaded with one of the values in table 5.1 to set the baud rate for serial data transfer.
6. The SCON register is loaded with the value 50H, indicating serial mode 1, where an 8-bit data is framed with start and stop bits.
7. TR1 is set to 1 start timer 1.
8. TI is cleared by the "CLR TI" instruction.
9. The character byte to be transferred serially is written into the SBUF register.
10. The TI flag bit is monitored with the use of the instruction JNB TI, target to see if the character has been transferred completely.
11. To transfer the next character, go to step 5.

Example 1. Write a program for the 8051 to transfer letter 'A' serially at 4800- baud rate, 8 bit data, 1 stop bit continuously.

```
ORG 0000H
LJMP START
ORG 0030H
START: MOV TMOD, #20H    ; select timer 1 mode 2
MOV TH1, #0FAH           ; load count to get baud rate of 4800
MOV SCON, #50H           ; initialize UART in mode 2
                         ; 8 bit data and 1 stop bit
SETB TR1                 ; start timer
AGAIN: MOV SBUF, #'A'    ; load char 'A' in SBUF
BACK: JNB TI, BACK ; Check for transmit interrupt flag
CLR TI                   ; Clear transmit interrupt flag
SJMP AGAIN
```

```
END
```

Example 2. Write a program for the 8051 to transfer the message 'EARTH' serially at 9600 baud, 8 bit data, 1 stop bit continuously.

```
ORG 0000H
LJMP START
ORG 0030H
START: MOV TMOD, #20H          ; select timer 1 mode 2
MOV TH1, #0FDH                 ; load count to get reqd. baud rate of 9600
MOV SCON, #50H                 ; initialise uart in mode 2
                               ; 8 bit data and 1 stop bit
SETB TR1                       ; start timer
LOOP: MOV A, #'E'          ; load 1st letter 'E' in a
ACALL LOAD                     ; call load subroutine
MOV A, #'A'                ; load 2nd letter 'A' in a
ACALL LOAD                     ; call load subroutine
MOV A, #'R'                ; load 3rd letter 'R' in a
ACALL LOAD                     ; call load subroutine
MOV A, #'T'                ; load 4th letter 'T' in a
ACALL LOAD                     ; call load subroutine
MOV A, #'H'                ; load 4th letter 'H' in a
ACALL LOAD                     ; call load subroutine
SJMP LOOP                      ; repeat steps

LOAD: MOV SBUF, A
HERE: JNB TI, HERE ; Check for transmit interrupt flag
        CLR TI         ; Clear transmit interrupt flag
        RET

END
```

8051 Interrupts

An interrupt is an external or internal event that interrupts the microcontroller to inform it that a device/process needs its service. A single microcontroller can serve several devices by two ways:

1. Polling : The microcontroller continuously monitors the status of a given device. When the conditions are met, it performs the service. After that, it moves on to monitor the next device until each one is serviced. The polling method is not efficient, since it wastes much of the microcontroller's time by polling devices that may not need service.
2. Interrupts : Whenever any device needs its service, the device notifies the microcontroller by generating an interrupt signal. Upon receiving an interrupt signal, the microcontroller saves whatever it is doing and services the device by executing a specific program for the device.

The program which is associated with the interrupt is called the interrupt service routine (ISR) or interrupt handler. The advantage of interrupts is that the microcontroller can serve many devices (not all at the same

time but depending upon the requirement). A device can get the attention of the microcontroller based on the assigned priority. For the polling method, it is not possible to assign priority since it checks all devices in a round-robin fashion.

For every interrupt, there must be an interrupt service routine (ISR), or interrupt handler. For every interrupt, there is a fixed location in memory that holds the address of its ISR. The group of memory locations set aside to hold the addresses of ISRs is called interrupt vector table. Towards the end of each machine cycle, the 8051 microcontroller checks each of the corresponding interrupt flags.

Upon activation of an interrupt (interrupt flag being set), the microcontroller goes through the following steps.

- It finishes the instruction it is executing and saves the address of the next instruction (PC) on the stack.
- It jumps to a fixed location in memory, called the interrupt vector table, which holds an LCALL to the address of the ISR.

Upon execution of the LCALL, it starts to execute the interrupt service subroutine until it reaches the last instruction of the subroutine which is RETI (return from interrupt).

Upon executing the RETI instruction, the microcontroller clears the corresponding interrupt flag and returns to the place where it was interrupted.

- First, it gets the program counter (PC) address from the stack by popping the top two bytes of the stack into the PC.
- Then it starts to execute from that address.

8051 has a total of 5 (five) interrupts allocated.

- Two interrupts are set aside for hardware/external interrupts.

  P3.2 and P3.3 are for the external/hardware interrupts INT0 and INT1.
- Two interrupts are set aside for the timers.

  One for timer 0 and one for timer 1.
- Serial communication has a single interrupt that belongs to both receive and transfer.

NOTE: The reset pin of 8051 may also be considered as an external interrupt source. Functionality-to set the PC to 0000H.

Interrupt vector table

Interrupt	ROM Location (hex)	Pin
Reset	0000	9
External HW (INT0)	0003	P3.2 (12)
Timer 0 (TF0)	000B	
External HW (INT1)	0013	P3.3 (13)
Timer 1 (TF1)	001B	
Serial COM (RI and TI)	0023	

The **External Interrupts INT0 and INT1** can each be either level- activated or transition-activated, depending on bits IT0 and IT1 in Register TCON. The flags that actually generate these interrupts are bits IE0 and IE1 in TCON. When an external interrupt is generated, the flag that generated it is cleared by the hardware when the service routine is vectored to, only if the interrupt was transition-activated. If the interrupt was level-activated, then the external requesting source is what controls the request flag, rather than the on-chip hardware.

In the level-triggered mode, INT0 and INT1 pins are normally high when the interrupt is not generated. If a low-level signal is applied to them, it triggers the interrupt. Then the microcontroller stops whatever it is doing and jumps to the interrupt vector table to service that interrupt. The low-level signal at the INT pin must be removed before the execution of the last instruction of the ISR, RETI; otherwise, another interrupt will be generated. The level-triggered mode is the default mode upon reset of the 8051.

In edge-triggered interrupts the external source must be held high for at least one machine cycle, and then held low for at least one machine cycle. The falling edge of pins INT0 and INT1 are latched by the 8051 and are held by the TCON.1 and TCON.3 bits of TCON Register. Function as interrupt-in-service flags. It indicates that the interrupt is being serviced now and on this INTn pin, and no new interrupt will be responded to until this service is finished.

The **Timer 0 and Timer 1** Interrupts are generated by TF0 and TF1, which are set by a rollover in their respective Timer/Counter registers (except in Mode 3).

In polling TF, we have to wait until the TF is raised. The problem here is that the microcontroller is tied down while waiting for TF to be raised, and cannot do anything else.

Using interrupts solves this problem and, avoids tying down the controller. If the timer interrupt in the IE register is enabled, whenever the timer rolls over, TF is raised, and the microcontroller is interrupted in whatever it is doing, and jumps to the interrupt vector table to service the ISR. When a timer interrupt is generated, the flag that generated it is cleared by the on-chip hardware when the service routine is vectored to.

The **Serial Port Interrupt** is generated by the logical OR of RI and TI. Neither of these flags is cleared by hardware when the service routine is vectored to. In fact, the service routine will normally have to determine whether it was RI or TI that generated the interrupt, and the bit will have to be cleared in software. TI (transfer interrupt) is raised when the last bit of the framed data, the stop bit, is transferred; indicating that the SBUF register is ready to transfer the next byte. RI (received interrupt) is raised when the entire frame of data, including the stop bit, is received. When the SBUF register has a byte, RI is raised to indicate that the received byte needs to be picked up before it is lost (overrun) by new incoming serial data. In the 8051 there is only one interrupt set aside for serial communication.

All of the bits that generate interrupts can be set or cleared by software, with the same result as though it had been set or cleared by hardware. That is, interrupts can be generated or pending interrupts can be canceled in software. Interrupt sources can be individually enabled or disabled by setting or clearing a bit in Special Function Register IE (Interrupt Enable). IE also contains a global disable bit, EA, which disables all interrupts at once. The interrupts must be enabled by software for response (On reset all are disabled). The IE register is bit-addressable with the following function :

Bit	Function	Cleared to 0	Set as 1
Bit7-EA	global interrupt enable/disable	disables all interrupt requests	enables all individual interrupt requests
Bit6-X	-	-	-
Bit5-ET2	Enables Timer 2 interrupt (only in some variants)	-	-
Bit4-ES	enables or disables serial interrupt	UART system cannot generate an interrupt	UART system enables an interrupt
Bit3-ET1	bit enables or disables Timer 1 interrupt	Timer 1 cannot generate an interrupt	Timer 1 enables an interrupt
Bit2-EX1	enables or disables external 1 interrupt	change of the pin INT0 logic state cannot generate an interrupt	enables an external interrupt on the pin INT0 state change
Bit1-ET0	enables or disables timer 0 interrupt	Timer 0 cannot generate an interrupt	enables timer 0 interrupt
Bit0-EX0	enables or disables external 0 interrupt	change of the INT1 pin logic state cannot generate an interrupt	enables an external interrupt on the pin INT1 state change

Interrupt Flag Bits

Interrupt	Flag	SFR Register Bit
External 0	IE0	TCON.1
External 1	IE1	TCON.3
Timer 0	TF0	TCON.5
Timer 1	TF1	TCON.7
Serial Port	T1	SCON.1

Interrupt Priority

The priority scheme of 8051 is an internal polling sequence according to which it polls the five interrupts and responds accordingly.

When the 8051 is powered up (or reset), the priorities are assigned according to the following.

Interrupt Priority Upon Reset

Highest To Lowest Priority	
External Interrupt 0	(INT0)
Timer Interrupt 0	(TF0)
External Interrupt 1	(INT1)
Timer Interrupt 1	(TF1)
Serial Communication	(RI + TI)

It is not possible to foresee when an interrupt request will arrive. If several interrupts are enabled, it may happen that while one of them is in progress, another one is raised. For the microcontroller to know whether to continue operation or meet a new interrupt request, there is a priority list instructing it what to do. The priority list offers 3 levels of interrupt priority:

- Reset! The absolute master. When a reset request arrives, everything is stopped and the microcontroller restarts.
- Interrupt priority 1-can be disabled by Reset only.
- Interrupt priority 0-can be disabled by both Reset and interrupt priority 1.

We can alter the sequence of interrupt priority by assigning a higher priority to any one of the interrupts by programming a register called IP (interrupt priority). To give a higher priority to any of the interrupts, we make the corresponding bit in the IP register high.

MSB							LSB
X	X	X	PS	PT1	PX1	PT0	PX0

BIT	SYMBOL	FUNCTION
IP.7	—	Reserved.
IP.6	—	Reserved.
IP.5	—	Reserved.
IP.4	PS	Defines the Serial Port interrupt priority level. PS=1 programs it to the higher priority level.
IP.3	PT1	Defines the Timer 1 interrupt priority level. PT1=1 programs it to the higher priority level.
IP.2	PX1	Defines the External Interrupt 1 priority level. PX1=1 programs it to the higher priority level.
IP.1	PT0	Enables or disables the Timer 0 interrupt priority level. PT0=1 programs it to the higher priority level.
IP.0	PX0	Defines the External Interrupt 0 priority level. PX0=1 programs it to the higher priority level.

SU00545

A low-priority interrupt can itself be interrupted by a high- priority interrupt, but not by another low-priority interrupt. A high-priority interrupt can't be interrupted by any other interrupt source. If two requests of different priority levels are received simultaneously, the request of higher priority level is serviced. When two or more interrupt bits in the IP register are set to high, interrupts with equal priority are internally serviced according to the default sequence given in the previous table.