UNIT – 4: INTERRUPT PROGRAMMING & SERIAL COMMUNICATION WITH 8051:

Definition of an interrupt, types of interrupts, Timers and Counter programming with interrupts in assembly. 8051 Serial Communication: Data communication, Basics of Serial Data Communication, 8051 Serial Communication.

Interrupt Programming:

Interrupts vs. Polling Method:

A single microcontroller can serve several devices. There are two ways to do that: interrupts and polling.

In the interrupt method, whenever any device needs its service, the device notifies the microcontroller by sending it an interrupt signal. Upon receiving an interrupt signal, the microcontroller interrupts whatever it is doing and serves the device. The program which is associated with the interrupt is called the interrupt service routine (ISR) or interrupt handler.

In polling method, the microcontroller continuously monitors the status of a given device; when the status conditions met, it performs the service. After that, it moves on to monitor the next device until each one is serviced.

In polling method there is no efficient use of microcontroller. Though in interrupt method not all devices can be serviced at the same time, each device can get the attention of microcontroller based on priority assigned to it. In polling method priority cannot be assigned since it checks all devices in a round-robin fashion. In interrupt method, microcontroller can also ignore (mask) a device request for service which is not possible in polling method. In polling method, microcontroller wastes its time by polling devices that do not need service. To save the time, interrupt method is employed.

For example, in timer programming, microcontroller waits till the TF flag is set to 1. In interrupt method, microcontroller will perform some useful task while the timer is running. It does not wait till the TF is set to 1. Once the TF flag is set to 1, timer generates interrupt.

Interrupt service routine:

For every interrupt, there must be an interrupt service routine (ISR) or interrupt handler. When an interrupt is invoked, the microcontroller runs ISR. For ever 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 the interrupt vector table.

Interrupt Vector Table

Interrupt	ISR ROM Address (Hex)	Pin	Flag Clearing
Reset	0000	9	Auto
External Hardware interrupt 0 (INT0)	0003	P3.2	Auto
Timer 0 Interrupt (TF0)	000B		Auto
External Hardware interrupt 1 (INT1)	0013	P3.3	Auto
Timer 1 Interrupt (TF1)	001B		Auto
Serial COM Interrupt (RI and TI)	0023		Programmer clears it.

Steps in executing an interrupt:

Upon activation of an interrupt, the microcontroller goes through the following steps:
1. It finishes the instruction it is executing and saves the address of the next instruction (PC) on the stack.
2. It also saves the current status of all the interrupts internally (i.e.: not on the stack)
3. It jumps to a fixed location in memory, called the interrupt vector table that holds the address of the ISR.
4. The microcontroller gets the address of the ISR from the interrupt vector table and jumps to it. It starts to execute the interrupt service subroutine until it reaches the last instruction of the subroutine which is RETI (return from interrupt).
5. Upon executing the RETI instruction, the microcontroller 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.

Types of Interrupts:

There are two types of interrupts:
a. External interrupts
b. Internal interrupts

8051 has three external interrupts and three internal interrupts. They are:
i. Reset: When the reset pin is activated, the 8051 jumps to address location 0000h.

ii. Timer 0 and Timer 1 interrupt: Two interrupts are set aside for the timers: one for timer 0 and one for timer 1. Memory locations 000BH and 001BH in the interrupt vector table belongs to timer 0 and timer 1 respectively.

iii. INT0 and INT1: Two interrupts are set aside for external hardware interrupts. Pin numbers 12 (P3.2) and 13 (P3.3) of Port 3 are INT0 and INT1 respectively. They are also referred as EX1 and EX2. Memory locations 0003H is assigned to INT0 and 0013H is assigned to INT1 in the interrupt vector table.

iv. Serial Communication: It has single interrupt that belongs to both receive and transmit. Memory location 0023H belongs to this interrupt.

From interrupt vector table, it is evident that limited number of bytes is set aside for each interrupt. For Reset interrupt only 3 bytes of location is allocated. For example, a total of 8 bytes from location 0003H to 000AH is set aside for INT0, 8 bytes from location 000BH to 0012H for Timer 0, 8 bytes from location 0013H to 001AH for INT1, 8 bytes from location 001BH to 0022H for timer 1. If the service routine for a given interrupt is short enough to fit in the memory space allocated to it, it is placed in the vector table; otherwise an LJMP instruction is placed in the vector table to point to the address of the ISR and rest of the bytes allocated to that interrupt are unused.

```
        ORG 0         ; wake-up ROM reset location
        LJMP MAIN     ; by-pass int. vector table
        ; ---- the wake-up program
        ORG 30H
MAIN:
        ....
        END
```

Enabling and disabling an interrupt:

Upon reset, all interrupts are disabled (masked), meaning that none will be responded to by the microcontroller if they are activated. The interrupts must be enabled by software in order for the microcontroller to respond to them. There is a register called IE (interrupt enable) that is responsible for enabling (unmasking) and disabling (masking) the interrupts.

Interrupt Enable (IE) register:

It is a bit addressable register.

EA: If 0, disables all interrupts, no interrupt is acknowledged. If 1, each interrupt source is individually enabled or disabled by setting or clearing individual bit.
--: Not implemented, reserved for future use.
ET2: Enables or disables timer 2 overflows or capture interrupt (8052) only.
ES: Enables or disables the serial port interrupt.
ET1: Enables or disables timer 1 overflow interrupt.
EX1: Enables or disables external interrupt 1 (INT1).
ET0: Enables or disables timer 0 overflow interrupt.
ET0: Enables or disables external interrupt 0 (INT0).

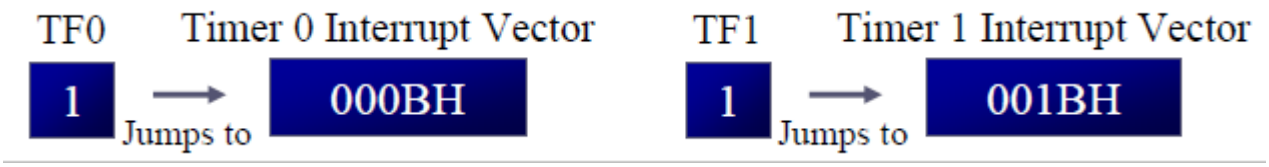
Show the instructions to (a) enable the serial interrupt, timer 0 interrupt, and external hardware interrupt 1 (EX1),and (b) disable (mask) the timer 0 interrupt, then (c) show how to disable all the interrupts with a single instruction.

Solution:
(a) MOV IE, #10010110B; enable serial, timer 0, EX1
(b) CLR IE.1; mask (disable) timer 0 interrupt only
(c) CLR IE.7; disable all interrupts.
Another way to perform the same manipulation is
SETB IE.7; EA=1, global enable
SETB IE.4; enable serial interrupt
SETB IE.1; enable Timer 0 interrupt
SETB IE.2; enable EX1

Programming Timer Interrupts:

Roll-over timer flag and interrupt:

The timer flag (TF) is raised when the timer rolls over. In polling TF, we have to wait until the TF is raised. The problem with this method 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. In this way, the microcontroller can do other things until it is notified that the timer has rolled over.



Write a program that continuously get 8-bit data from P0 and sends it to P1 while simultaneously creating a square wave of 200 µs period on pin P2.1. Use timer 0 to create the square wave. Assume that XTAL = 11.0592 MHz.

Solution:
We will use timer 0 in mode 2 (auto reload). TH0 = 100/1.085 us = 92

```
;--upon wake-up go to main, avoid using memory allocated to Interrupt Vector Table
            ORG 0000H
            LJMP MAIN               ; by-pass interrupt vector table
;--ISR for timer 0 to generate square wave
            ORG 000BH               ; Timer 0 interrupt vector table
            CPL P2.1                ; toggle P2.1 pin
            RETI
;--The main program for initialization
            ORG 0030H               ; after vector table space
      MAIN: MOV TMOD, #02H          ; Timer 0, mode 2
            MOV P0, #0FFH           ; make P0 an input port
            MOV TH0, #-92           ; TH0=A4H for -92
            MOV IE, #82H            ; IE=10000010 (bin) enable Timer 0
```

```
        SETB TR0            ; Start Timer 0
        BACK: MOV A, P0     ; get data from P0
        MOV P1, A           ; issue it to P1
        SJMP BACK           ; keep doing it loop unless interrupted by TF0
        END
```

Notice the following points in the above program:

1. Memory space allocated to the interrupt vector table should be avoided. Place all the initialization codes in memory starting at 30H. The LJMP instruction is the first instruction that the 8051 executes when it is powered up. LJMP redirects the controller away from the interrupt vector table.

2. The ISR for timer 0 is located starting at memory location 000BH since it is small enough to fit the address space allocated to this interrupt.

3. Enable timer 0 interrupt.

4. While the P0 data is brought in and issued to P1 continuously, whenever timer 0 is rolled over, the TF0 flag is raised and the microcontroller gets out of the BACK loop and goes to 000BH to execute the ISR associated with timer 0.

5. In the ISR for timer 0, notice that there is no need for a CLR TF0 instruction before the RETI instruction. This is because the 8051 clears the TF flag internally upon jumping to the interrupt vector table.

Write an assembly language program to create a square wave that has a high portion of 1085 us and a low portion of 15 us. Assume XTAL=11.0592MHz. Use timer 1.

Solution:

Since 1085 μs is 1000 × 1.085 we need to use mode 1 of timer 1.

```
;--upon wake-up go to main, avoid using memory allocated to Interrupt Vector Table
        ORG 0000H
        LJMP MAIN           ; by-pass int. vector table
;--ISR for timer 1 to generate square wave
        ORG 001BH           ; Timer 1 int. vector table
        LJMP ISR_T1         ; jump to ISR
;--The main program for initialization
        ORG 0030H           ; after vector table space
    MAIN: MOV TMOD, #10H    ; Timer 1, mode 1
        MOV P0, #0FFH       ; make P0 an input port
        MOV TL1, #018H      ; TL1=18 low byte of -1000
        MOV TH1, #0FCH      ; TH1=FC high byte of -1000
        MOV IE, #88H        ; 10001000 enable Timer 1 int
        SETB TR1            ; Start Timer 1
        BACK: MOV A, P0     ; get data from P0
        MOV P1, A           ; issue it to P1
        SJMP BACK           ; keep doing it
; Timer 1 ISR. Must be reloaded, not auto-reload
        ISR_T1: CLR TR1     ; stop Timer 1
        MOV R2, #4          ; 2MC
```

```
CLR P2.1                ; P2.1=0, start of low portion
HERE: DJNZ R2, HERE     ; 4x2 machine cycle 8MC
MOV TL1, #18H           ; load T1 low byte value 2MC
MOV TH1, #0FCH          ; load T1 high byte value 2MC
SETB TR1                ; starts timer1 1MC
SETB P2.1               ; P2.1=1, back to high 1MC
RETI                    ; return to main
END
```

In the above program the low portion of the pulse is created by the 14 machine cycles (MC) where each MC = 1.085 μs and 14 x 1.085 μs = 15.19 μs.

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.**

Programming external hardware interrupts:

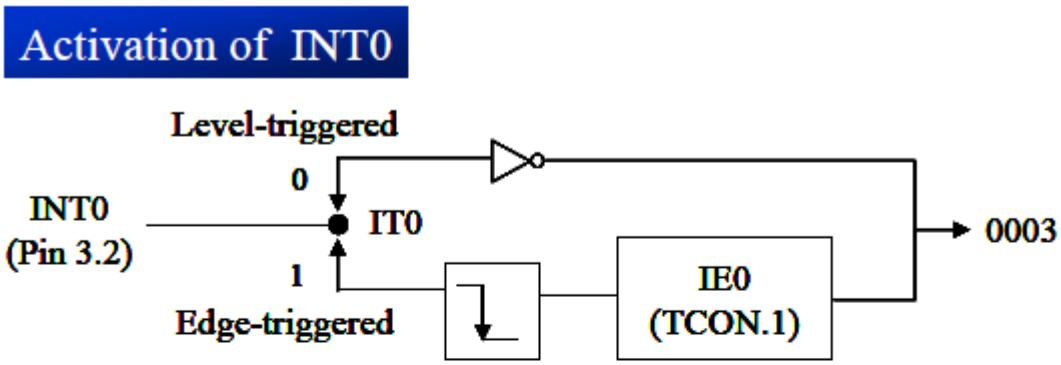
The 8051 has two external hardware interrupts. Pin 12 (P3.2) and pin 13 (P3.3) of the 8051, designated as INT0 and INT1, are used as external hardware interrupts. Upon activation of these pins, the 8051 gets interrupted in whatever it is doing and jumps to the vector table to perform the interrupt service routine.

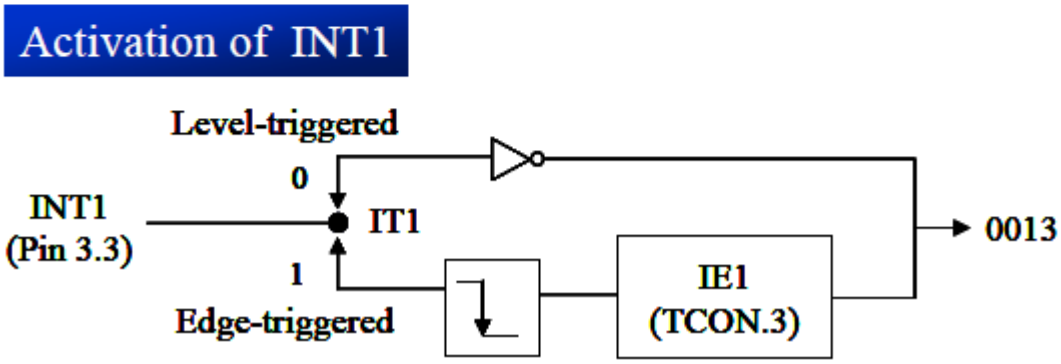
External interrupts INT0 and INT1:

There are two external hardware interrupts; INT0 and INT1. They are located on pins P3.2 and P3.3 of port 3 respectively. The IVT locations 0003H and 0013H are set aside for INTO0 and INT1 respectively. They are enabled and disabled using IE register.

They are activated using two methods:
1. Level triggered
2. Edge triggered.

Level triggered interrupt:





In the level-triggered mode, INT0 and INT1 pins are normally high. 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. This is called as a level-triggered or level-activated interrupt and is the default mode upon reset of the 8051. 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. In other words, if the low level interrupt signal is not removed before the ISR is finished it is interpreted as another interrupt and the 8051 jumps to the vector table to execute the ISR again.

Assume that the INT1 pin is connected to a switch that is normally high. Whenever it goes low, it should turn on an LED. The LED is connected to P1.3 and is normally off. When it is turned on it should stay on for a fraction of a second. As long as the switch is pressed low, the LED should stay on.

Solution:

```
            ORG 0000H
            LJMP MAIN               ; by-pass interrupt vector table
;--ISR for INT1 to turn on LED
            ORG 0013H               ; INT1 ISR
            SETB P1.3               ; turn on LED
            MOV R3, #255
            BACK: DJNZ R3, BACK     ; keep LED on for a while
            CLR P1.3                ; turn off the LED
            RETI                    ; return from ISR
```

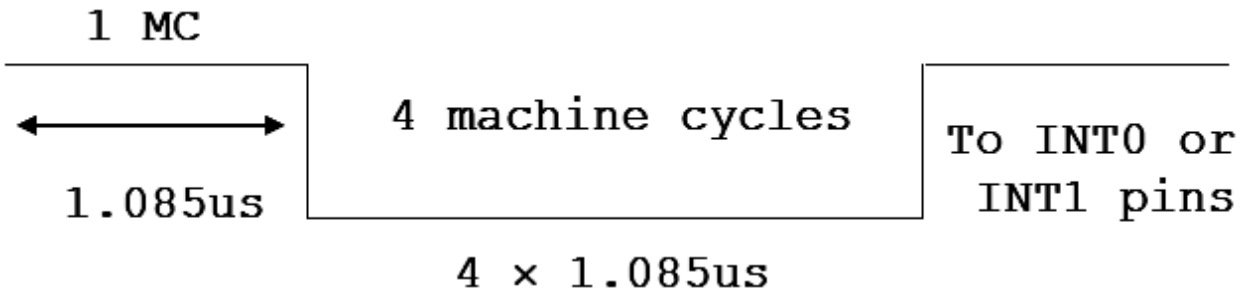
```
;--MAIN program for initialization
        ORG 30H
MAIN: MOV IE, #10000100B      ; enable external INT 1
      HERE: SJMP HERE         ; stay here until get interrupted
        END
```

In the above program, the microcontroller is looping continuously in the HERE loop. Whenever the switch on INT1 is activated, the 8051 gets out of the loop and jumps to vector location 0013H. The ISR for INT1 turns on the LED, keeps it on for a while, and turns it off before it returns. If by the time it executes the RETI instruction, the INT1 pin is still low, the microcontroller initiates the interrupt again. Therefore, to end this problem, the INT1 pin must be brought back to high by the time RETI is executed.

Sampling the low level-triggered interrupt:

Pins P3.2 and P3.3 are used for normal I/O unless the INT0 and INT1 bits in the IE register are enabled. After the hardware interrupts in the IE register are enabled, the controller keeps sampling the INT0 or INT1 pin for a low-level signal once each machine cycle. According to one manufacturer's data sheet, the pin must be held in a low state until the start of the execution of ISR. If the INT0 or INT1 pin is brought back to logic high before the start of the execution of ISR there will be no interrupt.

However, upon activation of the interrupt due to the low level, it must be brought back to high before the execution of RETI. According to one manufacturer's data sheet, if the INT0 or INT1 pin is left at a logic low after the RETI instruction of the ISR, another interrupt will be activated after one instruction is executed. Therefore to ensure the activation of the hardware interrupt at the INT0 or INT1 pin, make sure that the duration of the low-level signal is around 4 machine cycles but no more. This is due to fact that the level-triggered interrupt is not latched. Thus the pin must be held in a low state until the start of the ISR execution.



On reset, both IT0 (TCON.0) and IT1 (TCON.2) are low, making external interrupt level triggered.

Edge-triggered interrupts:

To make INT0 and INT1 edge triggered interrupts, we must program the bits of the TCON register. The TCON register holds, among other bits, the IT0 and IT1 flag bits that determine level- or edge-triggered mode of the hardware interrupts. IT0 and IT1 are bits D0 and D2 of the TCON register, respectively. They are also referred to as TCON.0 and TCON.2 since the TCON register is bit addressable. On reset, both IT0 (TCON.0) and IT1 (TCON.2) are low, making external interrupt level triggered. By making the TCON.0 and TCON.2 bits high with instructions such as SETB TCON.0 and SETB TCON.2, the external hardware interrupts of INT0 and INT1 become edge-triggered. For example, the instruction SETB TCON.2 makes INT1 edge-triggered. When a high-to-low signal is applied to pin P3.3, 8051 is interrupted and forced to jump to location 0013H in the vector table to service the ISR, assuming interrupt due to INT1 is enabled.

Generate from all pins of Port 0, a square wave which is half the frequency of the signal applied at INT0.

Solution:

```
            ORG 0000H
            LJMP MAIN
; ISR for hardware interrupt INT0
            ORG 0003H
            CPL P0
            RETI
            ORG 0030H
      MAIN:
            SETB TCON.0          ; make INT0 an edge-triggered interrupt
            MOV IE, #81H         ; enable hardware interrupt INT0
      HERE: SJMP HERE
            END
```

Assume that pin 3.3 (INT1) is connected to a pulse generator, write a program in which the falling edge of the pulse will send a high to P1.3, which is connected to an LED (or buzzer). In other words, the LED is turned on and off at the same rate as the pulses are applied to the INT1 pin.

Solution:

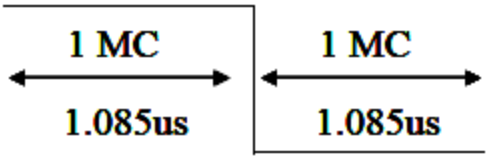
```
            ORG 0000H
            LJMP MAIN
;--ISR for hardware interrupt INT1 to turn on LED
            ORG 0013H            ; INT1 ISR
            SETB P1.3            ; turn on LED
            MOV R3, #255
            BACK: DJNZ R3, BACK  ; keep the buzzer on for a while
            CLR P1.3             ; turn off the buzzer
            RETI                 ; return from ISR
; ------MAIN program for initialization
            ORG 30H
            MAIN: SETB TCON.2    ; make INT1 edge-triggered int.
            MOV IE, #10000100B   ; enable External INT 1
            HERE: SJMP HERE      ; stay here until get interrupted
            END
```

In the above program when falling edge of the signal is applied to pin INT1, the LED will be turned on momentarily. The LED's on-state duration depends on the time delay inside the ISR for INT1. To turn on the LED again, another high-to-low pulse must be applied to pin P3.3.

Sampling the edge-triggered interrupt:

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 to ensure that the transition is seen by the microcontroller.

Minimum pulse duration to detect edge-triggered interrupts XTAL=11.0592MHz



The falling edge is latched by the 8051 and is held by the TCON register. The TCON.1 and TCON.3 bits hold the latched falling edge of pins INT0 and INT1, respectively. TCON.1 and TCON.3 are also called IE0 and IE1, respectively. They function as interrupt-in-service flags. When an interrupt-in-service flag is raised, it indicates to the external world that the interrupt is being serviced and no new interrupt on this INT0 or INT1 pin will be responded to until this service is finished. This is just like the busy signal you get if calling a telephone number that is in use.

Regarding the IT0 and IT1 bits in the TCON register, the following two points must be emphasized.
1. When the ISRs are finished (that is, upon execution of RETI), these bits (TCON.1 and TCON.3) are cleared, indicating that the interrupt is finished and the 8051 is ready to respond to another interrupt on that pin. For another interrupt to be recognized, the pin must go back to a logic high state and be brought back low to be considered an edge-triggered interrupt.

2. While the ISR is being executed, the INT0 or INT1 pin is ignored, no matter how many times it makes a high-to-low transition. In reality one of the functions of the RETI instruction is to clear the corresponding bit in the TCON register (TCON.1 or TCON.3). This informs us that the service routine is no longer in progress and has finished being serviced. For this reason TCON.1 and TCON.3 in the TCON register are called interrupt-in-service flags. The interrupt-in-service flag goes high whenever a falling edge is detected at the INT pin, and stays high during the entire execution of the ISR. It is only cleared by RETI, the last instruction of the ISR. Because of this, there is no need for an instruction such as CLR TCON.1 (INT0) or CLR TCON.3 (INT1) before the RETI in the ISR associated with the hardware interrupt INT0 or INT1.

What is the difference between the RET and RETI instructions? Explain why we cannot use RET instead of RETI as the last instruction of an ISR.

Both perform the same actions of popping off the top two bytes of the stack into the program counter, and marking the 8051 return to where it left off. However, RETI also performs an additional task of clearing the interrupt-in-service flag, indicating that the servicing of the interrupt is over and the 8051 now can accept a new interrupt on that pin. If you use RET instead of RETI as the last instruction of the interrupt service routine, you simply block any new interrupt on that pin after the first interrupt, since the pin status would indicate that the interrupt is still being serviced. In the cases of TF0, TF1, TCON.1, and TCON.3, they are cleared due to the execution of RETI.

More about the TCON register:

IT0 and IT1:
These two bits set the low-level or edge-triggered modes of the external hardware interrupts of the INT0 and INT1 pins. They are both 0 upon reset, which makes them low-level triggered. The programmer can make either of them high to make the external hardware interrupt edge-triggered. Once they are set to 0 or 1 they will not be altered again since the designer has fixed the interrupt as wither edge or level triggered.

IE0 and IE1:

These bits are used to keep track of the edge-triggered interrupt only. In other words, if the IT0 and IT1 are 0, meaning that the hardware interrupts are low level triggered, IE0 and IE1 are not used at all. The IE0 and IE1 bits are used to latch the high-to-low edge transition on the INT0 and INT1 pins. Upon the edge transition pulse on the INT0 or INT1 pin, the 8051 sets the IE0 or IE1 bit in the TCON register, jumps to the vector in the interrupt vector table and starts to execute the ISR. While it is executing the ISR, no high-to-low pulse transition on the INT0 or INT1 is recognized, thereby preventing any interrupt inside the interrupt. Only the execution of the RETI instruction at the end of the ISR will clear the IE0 or IE1 bit, indicating that the new high-to-low pulse will activate the interrupt again. Hence IE0 and IE1 bits are used internally by the 8051 to indicate whether or not an interrupt is in use and programmer is not concerned with these bits since they are solely for internal use.

TR0 and TR1:

These bits are used to start and stop the timer. These bits can be enabled and disabled using the instructions SETB or CLR instruction and SETB TCON.4 and CLR TCON.4.

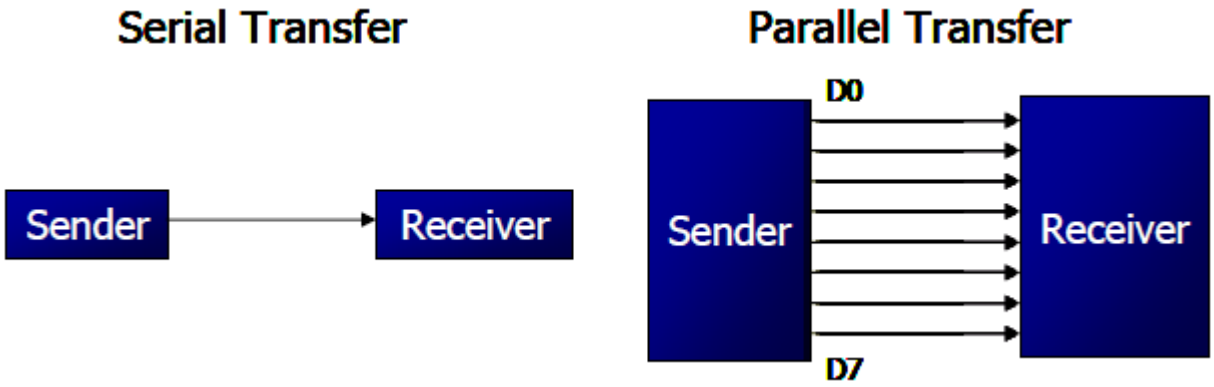
TF0 and TF1:

These bits are timer overflow flag bits. They are set when the corresponding timer overflows. They are monitored using JNB TFx, TARGET instruction and cleared using CLR instruction. They can also be monitored using JNB TCON.5, TARGET and cleared using CLR TCON.5 instructions.

Basics of Serial Communication:

Computers transfer data in two ways; parallel and serial. In parallel data transfers, 8 or more lines (wire conductors) are used to transfer data to a device that is only a few feet away. Example, printers and hard disks. Lot of data can be transferred in a short amount of time by using many wires in parallel.

To transfer to a device located many meters away, the serial method is used. Here the data is sent one bit at a time, 8051 has serial communication capability built into it, thereby making possible fast data transfer using only a few wires.



When a microprocessor communicates with the outside world, it provides the data in byte-sized chunks. In printers, the information is grabbed from the 8-bit data bus and presented to the 8-bit data bus of the printer. This can work only if the cable is not too long, since long cables diminish and even distort signals. Also 8-bit data path is expensive. For these reasons serial communication is used for transferring data between two systems located at distances of hundreds of feet to millions off miles apart. It is much cheaper and enables two two computers located in two different cities to communicate over the telephone.

For serial data communication to work the byte of data must be converted to serial bits using a parallel-in-serial-out shift register; then it can be transmitted over a single data line. At the receiving end there must be a serial-in-parallel-out shift register to receive the serial data and pack them into a byte. If data is to be transferred on the telephone line, it must be converted from 0s and 1s to audio tones, which are sinusoidal-shaped signals. This conversion is performed by a peripheral device called a modem.

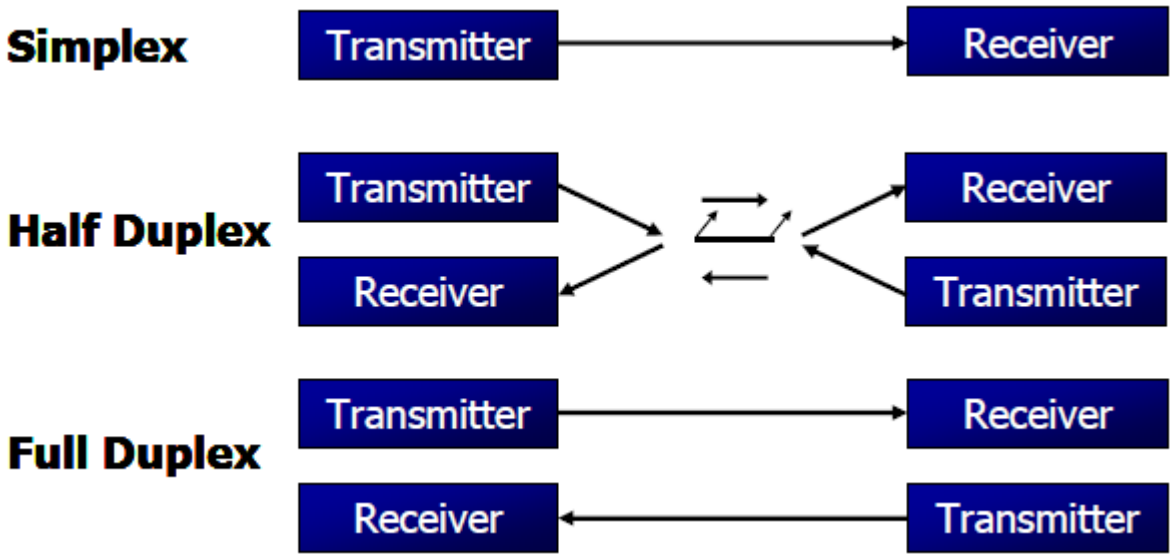
When the distance is short, the digital signal can be transferred as it is on a simple wire and requires no modulation. Example keyboards transfer data to the motherboard. However, for long distance data transfers using communication lines such as a telephone, serial data communication requires a modem to modulate and demodulate.

Serial data communication uses two methods synchronous and asynchronous. The Synchronous method transfers a block of data (characters) at a time while the Asynchronous method transfers a single byte at a time. It is possible to write software to use either of these methods, but the programs can be tedious and long. For this reason, there are special IC chips made by many manufacturers for serial data communications. They are commonly referred to as UART (Universal Asynchronous Receiver Transmitter) and USART (Universal Synchronous Asynchronous Receiver Transmitter). The 8051 has built-in UART.

Half and full-duplex transmission:

In data transmission if data can be transmitted and received, it is a duplex transmission. This is in contrast to simplex transmissions such as with printers, in which the computer only sends data. Duplex transmission can be half or full duplex, depending on whether or not the data transfer can be simultaneous. If data is transmitted

one way at a time, it is referred to as half duplex. If the data can go both ways at the same time, it is full duplex. Of course, full duplex requires two wire conductors for the data lines (in addition to the signal ground), one for transmission and one for reception, in order to transfer and receive data simultaneously.

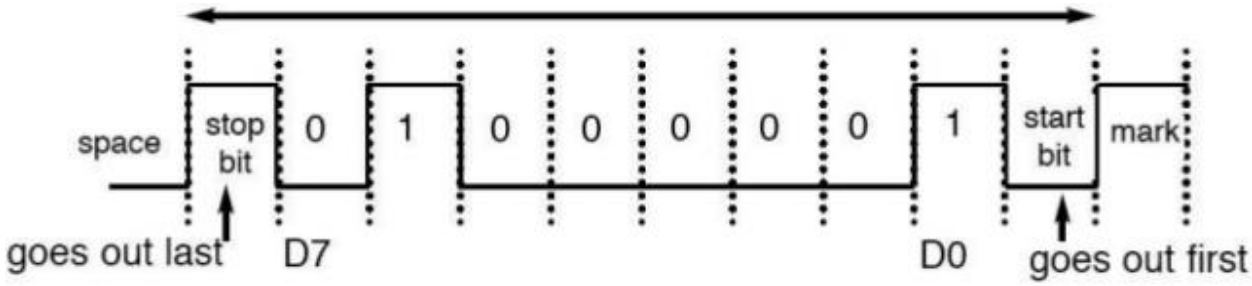


Asynchronous serial communication and data framing:

The data coming in at the receiving end of the data line in a serial data transfer is all 0s and 1s. It is difficult to make sense of the data unless the sender and receiver agree on a set of rules, a protocol, on how the data is packed, how many bits constitute a character, and when the data begins and ends.

Start and stop bits:

Asynchronous serial data communication is widely used for character-oriented transmissions while block-oriented data transfers use the synchronous method. In the asynchronous method, each character is placed between start and stop bits. This is called framing. In data framing for asynchronous communications, the data, such as ASCII characters, are packed between a start bit and a stop bit. The start bit is always one bit, but the stop bit can be one or two bits. The start bit is always a 0 (low) and the stop bit(s) is 1 (high).



For example, in the above figure an ASCII character ‘A’ (41H = 01000001) is framed between the start and stop bit. The LSB is sent out first. When there is no transfer, the signal is 1, which is referred to as mask. The 0 is referred to as space. The transmission begins with a start bit followed by D0, which is the LSB, then the rest of the bits until the MSB (D7), and finally, the one stop bit indicating the end of the character ‘A’.

In asynchronous serial communication, peripheral chips and modems can be programmed for data that is 7 or 8 bits wide in addition to 1 start bit and 1 or 2 stop bits. In older systems due to slowness of receiving mechanical device, two stop bits were used to give the device sufficient time to organize itself before transmission of the

next byte. In modern systems only one stop bit is used. For each 8 bit data, 1 start bit and 1 stop bit are used. Hence total bits transmitted are 10 bits.

Data transfer rate:

The rate of data transfer is stated in bits per second (bps). Another widely used terminology for bps is baud rate. However both are not necessarily equal. This is because baud rate is the modem terminology and is defined as the number of signal changes per second. In modems a single change of signal, sometimes transfers several bits of data. For conductor wire which transfers the data, baud rate and bps are same.

The data transfer rate of a given computer system depends on communication ports incorporated into that system. For example early IBM PC/XT could transfer data at the rate of 100 to 9600 bps. However Pentium based PCs transfer data at rates as high as 56 Kbps. In asynchronous serial data communication, the baud rate is generally limited to 100,000 bps.

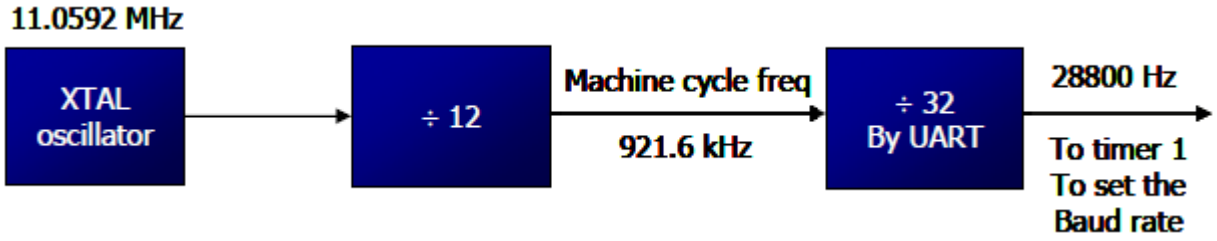
8051 serial port programming in assembly language:

To allow data transfer between the PC and 8051 without any error, baud rate of both 8051 and baud rate of com ort of PC should match. Some of the baud rates supported by PC is listed below.

PC Baud Rates
110
150
300
600
1200
2400
4800
9600
19200

Baud rate in 8051:

The 8051 transfers and receives data serially at many different baud rates. The baud rate in the 8051 is programmable using timer 1 in mode 2. 8051 divides the crystal frequency by 12 to get the machine cycle frequency. For XTAL = 11.0592 MHz, the machine cycle frequency is (11.0592 MHz / 12 = 921.6 kHz). The UART circuitry divides the machine cycle frequency of 921.6 kHz by 32 and feeds it to timer 1 to set the baud rate. The frequency of the signal applied to timer 1 is (921.6 kHz / 32 = 28800).



The timer 1 should be used in mode 2 operation to generate the baud rate. To generate baud rate initial value is loaded to TH1. Initial value to be loaded to TH1 to generate various baud rate is shown below.

Baud Rate	TH1 (Decimal)	TH1 (Hex)
9600	–3	FD
4800	–6	FA
2400	–12	F4
1200	–24	E8

Note: XTAL = 11.0592 MHz.

With XTAL = 11.0592 MHz, find the TH1 value needed to have the following baud rates. (a) 9600, (b) 2400 and (c) 1200.

Solution:
The machine cycle frequency of 8051 = 11.0592 / 12 = 921.6 kHz, and 921.6 kHz / 32 = 28,800 Hz is frequency by UART to timer 1 to set baud rate.

(a) 28,800 / 3 = 9600 where -3 = FD (hex) is loaded into TH1
(b) 28,800 / 12 = 2400 where -12 = F4 (hex) is loaded into TH1
(c) 28,800 / 24 = 1200 where -24 = E8 (hex) is loaded into TH1

Notice that dividing 1/12 of the crystal frequency by 32 is the default value upon activation of the 8051 RESET pin.

SBUF register:
It is an 8-bit SFR used solely for serial communication. For a byte of data to be transferred via the TxD line, it must be placed in the SBUF register. Also SBUF holds the byte of data when it is received on RxD line. SBUF can be accessed like any other register.

```
Example:    MOV SBUF, #'D'    ; load SBUF=44h, ASCII for 'D'
            MOV SBUF, A       ; copy accumulator into SBUF
            MOV A, SBUF       ; copy SBUF into accumulator
```

The moment a byte is written into SBUF, it is framed with the start and stop bits and transferred serially via the TxD line. Similarly, When the bits are received serially via RxD, the 8051 deframes it by eliminating the stop and start bits, making a byte out of the data received, and then placing it in SBUF.

SCON (serial control) register:

SCON is an 8-bit register used to program the start bit, stop bit, and data bits of data framing, among other things.

SM0	SM1	SM2	REN	TB8	RB8	TI	RI

SM0	SCON.7	Serial port mode specifier
SM1	SCON.6	Serial port mode specifier
SM2	SCON.5	Used for multiprocessor communication
REN	SCON.4	Set/cleared by software to enable/disable reception
TB8	SCON.3	Not widely used
RB8	SCON.2	Not widely used
TI	SCON.1	Transmit interrupt flag. Set by HW at the begin of the stop bit mode 1. And cleared by SW
RI	SCON.0	Receive interrupt flag. Set by HW at the begin of the stop bit mode 1. And cleared by SW

Note: Make SM2, TB8, and RB8 =0

SMO, SM1:

They determine the framing of data by specifying the number of bits per character, and the start and stop bits. There are four modes of operation out of which only mode 1 is of interest to us.

SM0	SM1	
0	0	Serial Mode 0
0	1	Serial Mode 1, 8 bit data, 1 stop bit, 1 start bit
1	0	Serial Mode 2
1	1	Serial Mode 3

When mode 1 is chosen, the data framing is 8 bits, 1 start bit and 1 stop bit which makes it compatible with the com ports of PC. This mode allows baud rate to be variable and is set by timer 1 in mode 2. In this mode for each character a total of 10 bits are transferred where the first bit is the start bit, followed by 8 bits of data and 1 stop bit.

SM2:

This enables the multiprocessing capability of the 8051. For our application this bit is set to 0 since we are not using the 8051 in multiprocessor environment.

REN:

The REN (receive enable) bit is also referred to as SCON.4. When this bit is high, it allows the 8051 to receive data on the RxD pin of the 8051. For transferring and receiving data serially this bit should be set to 1. If the bit is reset to 0, the receiver is disabled. It is used to block any serial data reception. It can be set using SETB SCON.4 and cleared using CLR SCON.4.

TB8:
TB8 (transfer bit 8) is used for serial modes 2 and 3. Since we use only mode 1, this bit is reset to 0.

RB8:
RB8 (receive bit 8) gets a copy of stop bit when an 8 bit data is received in mode 1 operation. It is used in modes 2 and 3 hence it is reset to 0.

TI:
When the 8051 finishes the transfer of 8 bit character, it raises the TI (transmit interrupt) flag to indicate that it is ready to transmit another byte. It is raised at the beginning of the stop bit.

RI:
When the 8051 receives data serially via RxD, it gets rid of the start and stop bits and places the byte in the SBUF register. Then it raises the RI (receive interrupt) flag bit to indicate that a byte has been received and should be picked up before it is lost. It is raised halfway through the stop bit.

Programming the 8051 to transfer data serially:
In programming the 8051 to transfer character bytes serially, the following steps must be taken.

1. TMOD register is loaded with the value 20H, indicating the use of timer 1 in mode 2 (8-bit auto-reload) to set baud rate.
2. The TH1 is loaded with one of the initial values to set baud rate for serial data transfer (assuming XTAL = 11.0592 MHz).
3. 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
4. TR1 is set to 1 to start timer 1.
5. TI is cleared by CLR TI instruction.
6. The character byte to be transferred serially is written into SBUF register.
7. The TI flag bit is monitored with the use of instruction JNB TI,xy to see if the character has been transferred completely.
8. To transfer the next byte, go to step 5.

Write a program for the 8051 to transfer letter "A" serially at 4800 baud, continuously.

Solution:

```
        MOV TMOD, #20H      ; timer 1, mode 2(auto reload)
        MOV TH1, #-6        ; 4800 baud rate
        MOV SCON, #50H      ; 8-bit, 1 stop, REN enabled
        SETB TR1            ; start timer 1
AGAIN:  MOV SBUF, #"A"      ; letter "A" to transfer
HERE:   JNB TI, HERE        ; wait for the last bit
        CLR TI              ; clear TI for next char
        SJMP AGAIN          ; keep sending A
```

Write a program for the 8051 to transfer "YES" serially at 9600 baud, 8-bit data, 1 stop bit, do this continuously

Solution:

```
                MOV TMOD, #20H          ; timer 1,mode 2(auto reload)
                MOV TH1, #-3            ; 9600 baud rate
                MOV SCON, #50H          ; 8-bit, 1 stop, REN enabled
                SETB TR1                ; start timer 1
                AGAIN: MOV A, #"Y"      ; transfer "Y"
                ACALL TRANS
                MOV A, #"E"             ; transfer "E"
                ACALL TRANS
                MOV A, #"S"             ; transfer "S"
                ACALL TRANS
                SJMP AGAIN              ; keep doing it
; serial data transfer subroutine
TRANS:  MOV SBUF, A             ; load SBUF
HERE:   JNB TI, HERE            ; wait for the last bit
                CLR TI                  ; get ready for next byte
                RET
```

Importance of the TI flag:

The 8051 goes through the following steps in transmitting a character via TxD.

1. The byte character to be transmitted is written into the SBUF register.
2. The start bit is transferred.
3. The 8-bit character is transferred on bit at a time.
4. The stop bit is transferred. It is during the transfer of the stop bit that 8051 raises the TI flag (TI = 1), indicating that the last character was transmitted and it is ready to transfer the next character.
5. By monitoring the TI flag, we make sure that we are not overloading the SBUF register. If we write another byte into the SBUF register before TI is raised, the untransmitted portion of the previous byte will be lost. That is when 8051 finishes transferring a byte, it raises the TI flag to indicate it is ready for the next character.
6. After SBUF is loaded with a new byte, the TI flag bit must be forced to 0 by CLR TI in order for this new byte to be transferred.

Hence by checking the TI flag bit, we know whether or not the 8051 is ready to transfer another byte. It must be noted that TI flag bit is raised by 8051 itself when it finishes data transfer, whereas it must be cleared by the programmer with an instruction such as CLR TI. Also, If we write a byte into SBUF before the TI flag bit is raised, we risk the loss of a portion of the byte being transferred. The TI flag bit can be checked by the instruction JNB TI, XY or use an interrupt.

Programming the 8051 to receive data serially:

In programming the 8051 to receive character bytes serially, the following steps must be taken.

1. TMOD register is loaded with the value 20H, indicating the use of timer 1 in mode 2 (8-bit auto-reload) to set baud rate.
2. TH1 is loaded with initial value to set baud rate (assuming XTAL = 11.0592 MHz).
3. 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 and receive enable is turned on.
4. TR1 is set to 1 to start timer 1.
5. RI is cleared by CLR RI instruction.

6. The RI flag bit is monitored with the use of instruction JNB RI,xy to see if an entire character has been received yet.
7. When RI is raised, SBUF has the byte, its contents are moved into a safe place.
8. To receive the next character, go to step 5.

Write a program for the 8051 to receive bytes of data serially, and put them in P1, set the baud rate at 4800, 8-bit data, and 1 stop bit.

Solution:

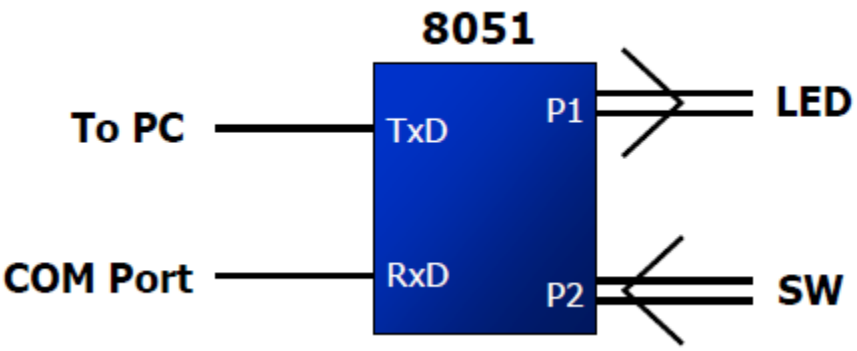
```
        MOV TMOD, #20H      ; timer 1, mode 2(auto reload)
        MOV TH1, #-6        ; 4800 baud rate
        MOV SCON, #50H      ; 8-bit, 1 stop, REN enabled
        SETB TR1            ; start timer 1
HERE:   JNB RI, HERE        ; wait for char to come in
        MOV A, SBUF         ; saving incoming byte in A
        MOV P1, A           ; send to port 1
        CLR RI              ; get ready to receive next byte
        SJMP HERE
```

Assume that the 8051 serial port is connected to the COM port of IBM PC, and on the PC, we are using the terminal.exe program to send and receive data serially. P1 and P2 of the 8051 are connected to LEDs and switches, respectively. Write an 8051 program to (a) send to PC the message "We Are Ready", (b) receive any data send by PC and put it on LEDs connected to P1, and (c) get data on switches connected to P2 and send it to PC serially. The program should perform part (a) once, but parts (b) and (c) continuously, use 4800 baud rate.

Solution:

```
        ORG 0
        MOV P2, #0FFH       ; make P2 an input port
        MOV TMOD, #20H      ; timer 1, mode 2
        MOV TH1, #0FAH      ; 4800 baud rate
        MOV SCON, #50H      ; 8-bit, 1 stop, REN enabled
        SETB TR1            ; start timer 1
        MOV DPTR, #MYDATA   ; load pointer for message
H_1:    CLR A
        MOV A, @A+DPTR      ; get the character
        JZ B_1              ; if last character gets out
        ACALL SEND          ; otherwise call transfer
        INC DPTR            ; next one
        SJMP H_1            ; stay in loop
B_1:    MOV a, P2           ; read data on P2
        ACALL SEND          ; transfer it serially
        ACALL RECV          ; get the serial data
        MOV P1, A           ; display it on LEDs
        SJMP B_1            ; stay in loop indefinitely
; ----serial data transfer. ACC has the data------
SEND:   MOV SBUF, A         ; load the data
H_2:    JNB TI, H_2         ; stay here until last bit gone
```

```
            CLR TI                  ; get ready for next char
            RET                     ; return to caller
; ----Receive data serially in ACC----------
RECV:       JNB RI, RECV            ; wait here for char
            MOV A, SBUF             ; save it in ACC
            CLR RI                  ; get ready for next char
            RET
; -----The message---------------
MYDATA:     DB "We Are Ready", 0
            END
```



Importance of RI flag bit:

In receiving bit via its RxD pin, 8051 goes through the following steps.

1. It receives the start bit indicating that the next bit is the first bit of the character byte it is about to receive.
2. The 8-bit character is received one bit at a time. When the last bit is received, a byte is formed and placed in SBUF.
3. The stop bit is received when receiving the stop bit 8051 makes RI = 1, indicating that an entire character byte has been received and must be picked up before it gets overwritten by an incoming character.
4. By checking the RI flag bit when it is raised, we know that a character has been received and is sitting in the SBUF register. We copy the SBUF contents to a safe place in some other register or memory before it is lost.
5. After the SBUF contents are copied into a safe place, the RI flag bit must be forced to 0 by CLR RI instruction in order to allow the next received character byte to be placed in SBUF. Failure to do this causes loss of the received character.

From the above discussion we can conclude that by checking the RI flag bit, we know whether or not the 8051 received a character byte. If we failed to copy SBUF into a safe place, we risk the loss of the received byte. It must be noted that RI flag bit is raised by 8051 when it finish receive data, it must be cleared by the programmer with instruction CLR RI. If we copy SBUF into a safe place before the RI flag bit is raised, we risk copying garbage. The RI flag bit can be checked by the instruction JNB RI, XY or by using the interrupt.

Doubling the baud rate in the 8051:

There are two ways to increase the baud rate of data transfer in 8051.
1. Use a higher frequency crystal.
2. Change a bit in the PCON register shown below.

PCON : Power Control Register (Not Bit Addressable)

SMOD	–	–	–	GF1	GF0	PD	IDL

SMOD	PCON.7	Double baud rate bit. If SMOD = 1, the baud rate is doubled when the serial part is used in mode 1, 2 and 3.
-	PCON.6	Not implemented, reserved for futur used*
-	PCON.5	Not implemented, reserved for futur used*
-	PCON.4	Not implemented, reserved for futur used*
GF1	PCON.3	General purpose bit.
GF0	PCON.2	General purpose bit.
PD	PCON.1	Power Down bit. If set, the oscillator is stopped. A reset or an interrupt (83C154 and 83C154D only) can cancel this mode (Note 1).
IDL	PCON.0	IDLE bit. If set the activity CPU is stopped. A reset or an interrupt can cancel this mode (See Note 1).

Option 1 is not feasible in many situations since the system crystal is fixed. New crystal may not be compatible with the PC serial COM ports baud rate. There is a software way to double the baud rate of 8051 while the crystal frequency is fixed. This is achieved with a SFR called PCON (power control). It is only byte addressable. In this register some bits are unused and some are used for the power control capability. The bit that is used for serial communication is SMOD (serial mode) bit. When the 8051 is powered up, SMOD bit is zero. This bit can be set to high by software or programming method and thereby double the baud rate. The following sequence of instructions must be used to set high SMOD bit, since PCON is not bit-addressable.

```
MOV A, PCON        ; place a copy of PCON in ACC
SETB ACC.7         ; make D7=1
MOV PCON, A        ; changing any other bits
```

Baud rates for SMOD = 0:

When SMOD=0, the 8051 divides $1/12^{th}$ of the crystal frequency by 32 and uses that frequency for timer 1 to set the baud rate. For XTAL = 11.0592,

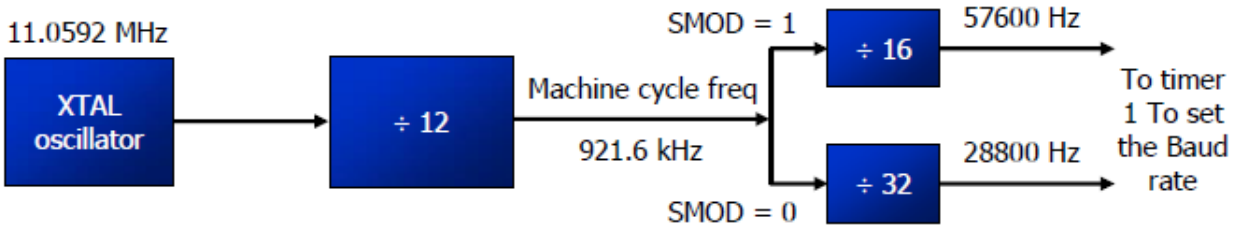
Machine cycle frequency = (11.0592 MHz / 12) = 921.6 kHz and (921.6 kHz / 32) = 28800 Hz is the frequency used by the timer 1 to set the baud rate.

Baud rates for SMOD = 1:

When SMOD=1, (1/12) of XTAL is divided by 16 (instead of 32) and that is the frequency used by timer 1 to set the baud rate. For XTAL = 11.0592,

Machine cycle frequency = (11.0592 MHz / 12) = 921.6 kHz and (921.6 kHz / 16) = 57600 Hz is the frequency used by the timer 1 to set the baud rate.

The initial value loaded into TH1 are the same for both cases; however, the baud rates are doubled when SMOD = 1 as shown in the table below.



Baud Rate comparison for SMOD=0 and SMOD=1

TH1	(Decimal)	(Hex)	SMOD=0	SMOD=1
	-3	FD	9600	19200
	-6	FA	4800	9600
	-12	F4	2400	4800
	-24	E8	1200	2400

Assume that XTAL = 11.0592 MHz for the following program, state (a) what this program does, (b) compute the frequency used by timer 1 to set the baud rate, and (c) find the baud rate of the data transfer.

```
        MOV A, PCON          ; A=PCON
        MOV ACC.7            ; make D7=1
        MOV PCON, A          ; SMOD=1, double baud rate with same XTAL freq.
        MOV TMOD, #20H       ; timer 1, mode 2
        MOV TH1, -3          ; 19200 (57600/3 =19200)
        MOV SCON, #50H       ; 8-bit data, 1 stop bit, RI enabled
        SETB TR1             ; start timer 1
        MOV A, #"B"          ; transfer letter B
A_1:    CLR TI               ; make sure TI=0
        MOV SBUF, A          ; transfer it
H_1:    JNB TI, H_1          ; stay here until the last bit is gone
        SJMP A_1             ; keep sending "B" again
```

Solution:

(a) This program transfers ASCII letter B (01000010 binary) continuously

(b) With XTAL = 11.0592 MHz and SMOD = 1 in the above program, we have: 11.0592 / 12 = 921.6 kHz machine cycle frequency. 921.6 / 16 = 57,600 Hz frequency used by timer 1 to set the baud rate. 57600 / 3 = 19,200, the baud rate.

Find the TH1 value (in both decimal and hex) to set the baud rate to each of the following. (a) 9600 (b) 4800 if SMOD=1. Assume that XTAL 11.0592 MHz.

Solution:

With XTAL = 11.0592 and SMOD = 1, we have timer frequency = 57,600 Hz.

(a) 57600 / 9600 = 6; so TH1 = -6 or TH1 = FAH

(b) 57600 / 4800 = 12; so TH1 = -12 or TH1 = F4H

Find the baud rate if TH1 = -2, SMOD = 1, and XTAL = 11.0592 MHz. Is this baud rate supported by IBM compatible PCs?

Solution:
With XTAL = 11.0592 and SMOD = 1, we have timer frequency = 57,600 Hz. The baud rate is 57,600/2 = 28,800. This baud rate is not supported by the BIOS of the PCs; however, the PC can be programmed to do data transfer at such a speed. Also, HyperTerminal in Windows supports this and other baud rates.

Write a program to send the message "The Earth is but One Country" to serial port. Assume a SW is connected to pin P1.2. Monitor its status and set the baud rate as follows:
SW = 0, 4800 baud rate
SW = 1, 9600 baud rate
Assume XTAL = 11.0592 MHz, 8-bit data, and 1 stop bit.

Solution:

```
        SW BIT P1.2
        ORG 0H                  ; starting position
MAIN:
        MOV TMOD, #20H
        MOV TH1, #-6            ; 4800 baud rate (default)
        MOV SCON, #50H
        SETB TR1
        SETB SW                 ; make SW an input
S1:     JNB SW, SLOWSP          ; check SW status
        MOV A, PCON             ; read PCON
        SETB ACC.7              ; set SMOD high for 9600
        MOV PCON, A             ; write PCON
        SJMP OVER               ; send message
SLOWSP:
        MOV A, PCON             ; read PCON
        SETB ACC.7              ; set SMOD low for 4800
        MOV PCON, A             ; write PCON
OVER:   MOV DPTR, #MESS1        ; load address to message
FN:     CLR A
        MOVC A, @A+DPTR         ; read value
        JZ S1                   ; check for end of line
        ACALL SENDCOM           ; send value to serial port
        INC DPTR                ; move to next value
        SJMP FN                 ; repeat
SENDCOM:
        MOV SBUF, A             ; place value in buffer
HERE:   JNB TI, HERE            ; wait until transmitted
        CLR TI                  ; clear
        RET                     ; return
MESS1:  DB "The Earth is but One Country",0
        END
```

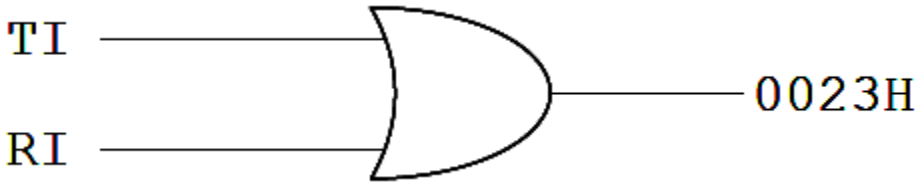
Programming the serial communication interrupt:

In the earlier topic, when data is transmitted or received serially, TI or RI flag was set which was monitored using JNB instruction. This method is known as polling method. Instead of this if interrupt is enabled due to serial communication, TI or RI flag need not be monitored. Once these flags are set, microcontroller will automatically generate interrupt.

RI and TI flags and interrupts:

TI 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 is raised when the entire frame of data, including the stop bit is received. That is 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. All the above concepts apply equally when using either polling or an interrupt. The only difference is in how the serial communication needs are served.

In the 8051 only one interrupt is set aside for serial communication. This interrupt is used to both send and receive data. If the interrupt bit in the IE register (IE.4) is enabled when RI or TI is raised, the 8051 gets interrupted and jumps to memory address location 0023H to execute the ISR. In that ISR we must examine the TI and RI flags to see which one caused the interrupt and respond accordingly.



Serial interrupt is invoked by TI or RI flags

Write a program in which the 8051 reads data from P1 and writes it to P2 continuously while giving a copy of it to the serial COM port to be transferred serially. Assume that XTAL=11.0592. Set the baud rate at 9600.

Solution:

```
            ORG 0000H
            LJMP MAIN
            ORG 23H
            LJMP SERIAL          ; jump to serial int ISR
            ORG 30H
MAIN:       MOV P1, #0FFH        ; make P1 an input port
            MOV TMOD, #20H       ; timer 1, auto reload
            MOV TH1, #0FDH       ; 9600 baud rate
            MOV SCON, #50H       ; 8-bit,1 stop, ren enabled
            MOV IE, #10010000B   ; enable serial int.
            SETB TR1             ; start timer 1
BACK:       MOV A, P1            ; read data from port 1
            MOV SBUF,  A         ; give a copy to SBUF
            MOV P2, A            ; send it to P2
            SJMP BACK            ; stay in loop indefinitely
; ----------------SERIAL PORT ISR
            ORG 100H
SERIAL:     JB TI, TRANS         ; jump if TI is high
            MOV A, SBUF          ; otherwise due to receive
```

```
        CLR RI              ; clear RI since CPU doesn't
        RETI                ; return from ISR
TRANS:  CLR TI              ; clear TI since CPU doesn't
        RETI                ; return from ISR
        END
```

The moment a byte is written into SBUF it is framed and transferred serially. As a result, when the last bit (stop bit) is transferred the TI is raised, and that causes the serial interrupt to be invoked since the corresponding bit in the IE register is high. In the serial ISR, we check for both TI and RI since both could have invoked interrupt.

Use of serial COM in the 8051:

In majority applications, the serial interrupt is used mainly for receiving data and is never used for sending data serially. This is like receiving a telephone call, where we need a ring to be notified. If we need to make a phone call there are other ways to remind ourselves and so no need for ringing. In receiving call, we must respond immediately. Similarly we use the serial interrupt to receive incoming data so that it is not lost.

Write a program in which the 8051 gets data from P1 and sends it toP2 continuously while incoming data from the serial port is sent to P0. Assume that XTAL=11.0592. Set the baud rata at 9600.

Solution:

```
        ORG 0000H
        LJMP MAIN
        ORG 23H
        LJMP SERIAL         ; jump to serial int ISR
        ORG 30H
MAIN:   MOV P1, #0FFH       ; make P1 an input port
        MOV TMOD, #20H      ; timer 1, auto reload
        MOV TH1, #0FDH      ; 9600 baud rate
        MOV SCON, #50H      ; 8-bit,1 stop, ren enabled
        MOV IE, #10010000B  ; enable serial int.
        SETB TR1            ; start timer 1
BACK:   MOV A, P1           ; read data from port 1
        MOV P2, A           ; send it to P2
        SJMP BACK           ; stay in loop indefinitely
; ----------------SERIAL PORT ISR
        ORG 100H
SERIAL: JB TI, TRANS        ; jump if TI is high
        MOV A, SBUF         ; otherwise due to receive
        MOV P0, A           ; send incoming data to P0
        CLR RI              ; clear RI since CPU doesn't
        RETI                ; return from ISR
TRANS:  CLR TI              ; clear TI since CPU doesn't
        RETI                ; return from ISR
        END
```

Write a program using interrupts to do the following:
(a) Receive data serially and sent it to P0,
(b) Have P1 port read and transmitted serially, and a copy given to P2,
(c) Make timer 0 generate a square wave of 5kHz frequency on P0.1.
Assume that XTAL-11,0592. Set the baud rate at 4800.

Solution:

```
                ORG 0
                LJMP MAIN
                ORG 000BH               ; ISR for timer 0
                CPL P0.1                ; toggle P0.1
                RETI                    ; return from ISR
                ORG 23H
                LJMP SERIAL             ; jump to serial interrupt ISR
                ORG 30H
MAIN:           MOV P1, #0FFH           ; make P1 an input port
                MOV TMOD, #22H          ; timer 1,mode 2(auto reload)
                MOV TH1, #0F6H          ; 4800 baud rate
                MOV SCON, #50H          ; 8-bit, 1 stop, ren enabled
                MOV TH0, #-92           ; for 5kHZ wave
                MOV IE, #10010010B      ; enable serial int.
                SETB TR1                ; start timer 1
                SETB TR0                ; start timer 0
BACK:           MOV A, P1               ; read data from port 1
                MOV SBUF, A             ; give a copy to SBUF
                MOV P2, A               ; send it to P2
                SJMP BACK               ; stay in loop indefinitely
; ----------------SERIAL PORT ISR
                ORG 100H
SERIAL:         JB TI, TRANS            ; jump if TI is high
                MOV A, SBUF             ; otherwise due to receive
                MOV P0, A               ; send serial data to P0
                CLR RI                  ; clear RI since CPU doesn't
                RETI                    ; return from ISR
TRANS:          CLR TI                  ; clear TI since CPU doesn't
                RETI                    ; return from ISR
                END
```

Clearing RI and TI before the RETI instruction:

In the above program RI and TI is cleared in the ISR before RETI instruction. This is necessary since there is only one interrupt for both receive and transmit, and the 8051 does not know who generated it. Hence programmer has to clear the flag in the ISR. Whereas if the interrupt is due to timers or external hardware interrupt, 8051 will clear the flag. The TCON register holds the four of the interrupt flags and SCON register has the RI and TI flags.

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:

When the 8051 is powered up, the priorities are assigned according to the following table.

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)

If INT0 and INT1 are activated at the same time, INT0 is responded first because it has higher priority over INT1.

Setting interrupt priority with the IP register:

We can alter the sequence of interrupt priority by assigning a higher priority to any one of the interrupts. This is done by programming a register called IP (interrupt priority).

Interrupt Priority Register (Bit-addressable)

D7							D0
--	--	PT2	PS	PT1	PX1	PT0	PX0

--	IP.7	Reserved
--	IP.6	Reserved
PT2	IP.5	Timer 2 interrupt priority bit (8052 only)
PS	IP.4	Serial port interrupt priority bit
PT1	IP.3	Timer 1 interrupt priority bit
PX1	IP.2	External interrupt 1 priority bit
PT0	IP.1	Timer 0 interrupt priority bit
PX0	IP.0	External interrupt 0 priority bit

Priority bit=1 assigns high priority
Priority bit=0 assigns low priority

Discuss what happens if interrupts INT0, TF0, and INT1 are activated at the same time. Assume priority levels were set by the power-up reset and the external hardware interrupts are edge triggered.

Solution:
If these three interrupts are activated at the same time, they are latched and kept internally. Then the 8051 checks all five interrupts according to the sequence listed in Table 11-3. If any is activated, it services it in sequence. Therefore, when the above three interrupts are activated, IE0 (external interrupt 0) is serviced first, then timer 0 (TF0), and finally IE1 (external interrupt 1).

When two or more interrupt bits in the IP register are set to high, while these interrupts have a higher priority than others, they are serviced according to the normal priority sequence.

Interrupt inside an interrupt:
When 8051 is executing an ISR belonging to an interrupt and another interrupt is activated, then a high priority interrupt can interrupt a low priority interrupt. This is an interrupt inside an interrupt. Low priority interrupt can be interrupted by a higher priority interrupt, but not by another low priority interrupt. Although all the interrupts are latched and kept internally, no low priority interrupt can get the immediate attention of the CPU until 8051 has finished servicing the high priority interrupts.

Triggering the interrupt by software:
To test an ISR by way of simulation, set the interrupts high and thereby cause the 8051 to jump to the interrupt vector table. For example, if the IE bit for timer 1 is set, an instruction such as SETB TF1 will interrupt the

8051 in whatever it is doing and force it to jump to the interrupt vector table. That is we need not wait for timer 1 to roll over to to have an interrupt. We can cause an interrupt with an instruction that raises the interrupt flag.

(a) Program the IP register to assign the highest priority to INT1(external interrupt 1), then
(b) Discuss what happens if INT0, INT1, and TF0 are activated at the same time. Assume the interrupts are both edge-triggered.

Solution:
(a) MOV IP,#00000100B ;IP.2=1 assign INT1 higher priority. The instruction SETB IP.2 also will do the same thing as the above line since IP is bit-addressable.
(b) The instruction in Step (a) assigned a higher priority to INT1 than the others; therefore, when INT0, INT1, and TF0 interrupts are activated at the same time, the 8051 services INT1 first, then it services INT0, then TF0. This is due to the fact that INT1 has a higher priority than the other two because of the instruction in Step (a). The instruction in Step (a) makes both the INT0 and TF0 bits in the IP register 0. As a result, the normal priority sequence is followed which gives a higher priority to INT0 over TF0.

Assume that after reset, the interrupt priority is set the instruction MOV IP,#00001100B. Discuss the sequence in which the interrupts are serviced.

Solution:
The instruction "MOV IP #00001100B" (B is for binary) and timer 1 (TF1)to a higher priority level compared with the reset of the interrupts. However, since they are polled according to normal priority sequence, they will have the following priority.

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