h

MODULE 4: 8051 Serial Port Programming in Assembly and C

Structure

4.1 Basics of serial communication

4.2 8051 connection to RS242

4.3 8051 serial port programming in assembly

4.4 Serial port programming in 8051 C

8051 Interrupt programming in assembly and C:

4.5 8051 interrupts

4.6 Programming timer

4.7 External hardware

4.8serial communication interrupt

4.9 Interrupt priority in 8051/52

4.10 Interrupt programming in C

Objectives

- To explain in detail the execution of 8051 Assembly language instructions and data types
- To explain develop 8051 programs for time delay and Interrupts

4.1 Basics of serial communication

1. When a microprocessor communicates with the outside world, it provides the data in byte-sized chunks. In some cases, such as printers, the information is simply grabbed from the 8-bit data bus and presented to the 8-bit data bus of the printer.
2. This can work only if the cable is not too long, since long cables diminish and even distort signals. Furthermore, an 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 of miles apart. Figure 4-1 diagrams serial versus parallel data transfers.
3. The fact that serial communication uses a single data line instead of the 8-bit data line of parallel communication not only makes it much cheaper but also enables two computers located in two different cities to communicate over the telephone.
4. 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. This also means that 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. Of course, if data is to be transferred on the telephone line, it must be converted from Os and Is to audio tones, which are sinusoidal-shaped signals. This conversion is performed by a peripheral device called a *modem,* which stands for "modulator/demodulator."
5. $_{t}$, When the distance is short, the digital signal can be transferred as it is on a simple wire and requires no modulation. This is how IBM PC 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* (convert from Os and 1 s to audio tones) and *demodulate* (converting from audio tones to Os and 1 s).
6. Serial data communication uses two methods, asynchronous and synchronous. 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 1C chips made by many manufacturers for serial data communications. These chips are commonly referred to as UART (universal asynchronous receiver-transmitter) and USART (universal synchronous-asynchronous receiver-transmitter).

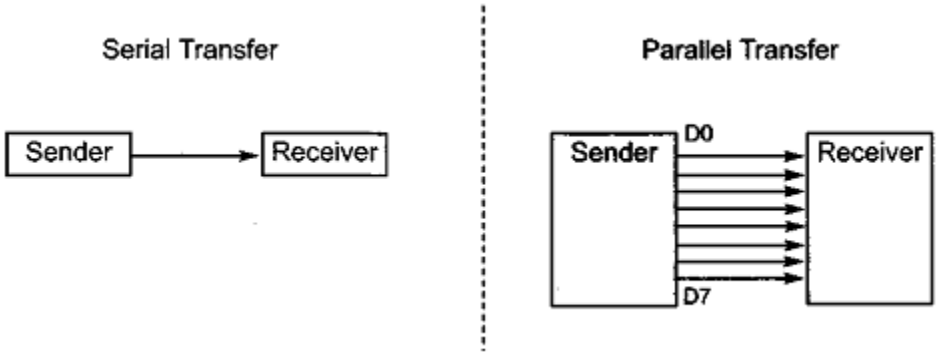


Figure 4-1. Serial versus Parallel Data Transfer

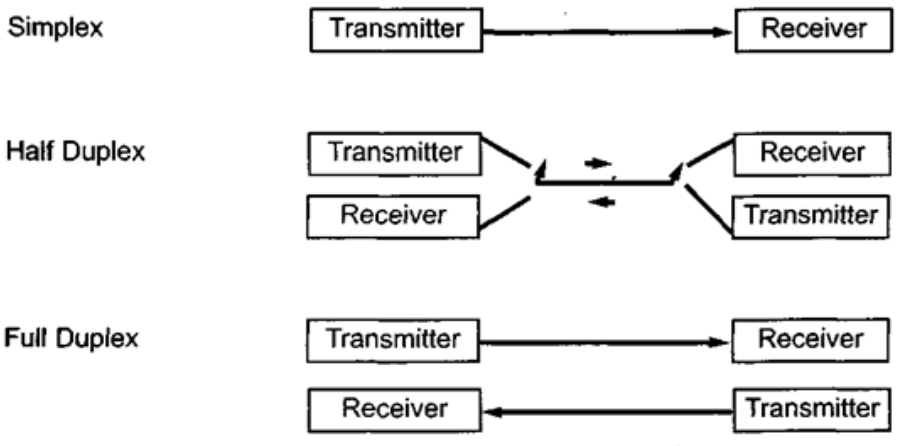


Figure 4.2: Simplex, Half-, and Full-Duplex Transfers

a) Half- and full-duplex transmission

In data transmission if the 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 transmissions 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.

b)Asynchronous serial communication and data framing

1. The data coming in at the receiving end of the data line in a serial data transfer is all Os and 1s;
2. 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

1. Asynchronous serial data communication is widely used for character-oriented transmissions, while block-oriented data transfers use the synchronous method.
2. 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.
3. 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).
4. For example, look at Figure 10-3 in which the ASCII character "A" (8-bit binary 0100 0001) is framed between the start bit and a single stop bit. Notice that the LSB is sent out first.

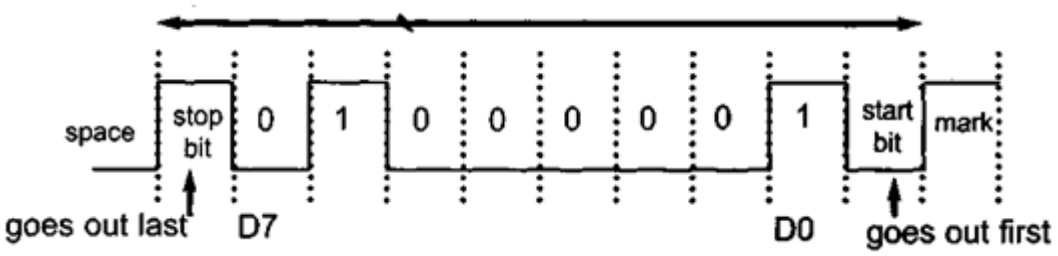


Figure 4.3: Framing ASCII "A" (41H)

5. Notice in Figure 4.3that when there is no transfer, the signal is 1 (high), which is referred to as *mark.* The 0 (low) is referred to as *space.* Notice that the transmission begins with a start bit followed by DO, 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".
6. In asynchronous serial communications, peripheral chips and modems can be programmed for data that is 7 or 8 bits wide. This is in addition to the number of stop bits, 1 or 2. While in older systems ASCII characters were 7-bit, in recent years, due to the extended ASCII characters, 8-bit data has become common. In some older systems, due to the slowness of the 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 PCs however, the use of one stop bit is standard.

7. Assuming that we are transferring a text file of ASCII characters using 1 stop bit, we have a total of 10 bits for each character: 8 bits for the ASCII code, and 1 bit each for the start and stop bits. Therefore, for each 8-bit character there are an extra 2 bits, which gives 20% overhead.
8. In some systems, the parity bit of the character byte is included in the data frame in order to maintain data integrity. This means that for each character (7- or 8-bit, depending on the system) we have a single parity bit in addition to start and stop bits. The parity bit is odd or even. In the case of an odd-parity bit the number of data bits, including the parity bit, has an odd number of Is. Similarly, in an even-parity bit system the total number of bits, including the parity bit, is even.
9. For example, the ASCII character "A", binary 0100 0001, has 0 for the even-parity bit. UART chips allow programming of the parity bit for odd-, even-, and no-parity options.

c) Data transfer rate

1. The rate of data transfer in serial data communication is stated in *bps* (bits per second). Another widely used terminology for bps is ***baud rate.***
2. However, the baud and bps rates are not necessarily equal. This is due to the fact that 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. As far as the conductor wire is concerned, the baud rate and bps are the same, and for this reason in this book we use the terms bps and baud interchangeably.
3. The data transfer rate of a given computer system depends on communication ports incorporated into that system. For example, the early IBM PC/XT could transfer data at the rate of 100 to **9600 bps.**
4. In recent years, however, Pentium-based PCs transfer data at rates as high as 56K bps. It must be noted that in asynchronous serial data communication, the baud rate is generally limited to 100,000 bps.

4.1.1 RS232 standards

1. To allow compatibility among data communication equipment made by various manufacturers, an interfacing standard called RS232 was set by the Electronics Industries Association (EIA) in 1960. In 1963 it was modified and called RS232A.
2. RS232B and RS232C were issued in 1965 and 1969, respectively. In this book we refer to it simply as RS232.
3. Today, RS232 is the most widely used serial I/O interfacing standard. This standard is used in PCs and numerous types of equipment. However, since the standard was set long before the advent of the TTL logic family, its input and output voltage levels are not TTL compatible. In RS232, a 1 is represented by -3 to -25 V, while a 0 bit is +3 to +25 V, making -3 to +3 undefined.
4. For this reason, to connect any RS232 to a microcontroller system we must use voltage converters such as MAX232 to convert the TTL logic levels to the RS232 voltage levels, and vice versa. MAX232 1C chips are commonly referred to as line drivers.

RS232 pins

Table 4.1 provides the pins and their labels for the RS232 cable, commonly referred to as the DB-25 connector. In labelling, DB-25P refers to the plug connector (male) and DB-25S is for the socket connector (female).

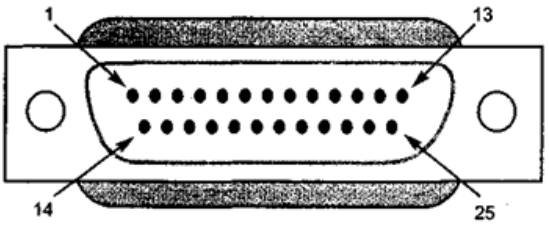


Figure 4.4: RS232 Connector DB-25

Since not all the pins are used in PC cables, IBM introduced the DB-9 version of the serial I/O standard, which uses 9 pins only, as shown in Table 4.2.

a) Data communication classification

Table 4.1:DB25 Pin connection

Pin	Description
1	Protective ground
2	Transmitted data (TxD)
3	Received data (RxD)
4	Request to send ($\overline{\text{RTS}}$)
5	Clear to send ($\overline{\text{CTS}}$)
6	Data set ready ($\overline{\text{DSR}}$)
7	Signal ground (GND)
8	Data carrier detect ($\overline{\text{DCD}}$)
9/10	Reserved for data testing
11	Unassigned
12	Secondary data carrier detect
13	Secondary clear to send
14	Secondary transmitted data
15	Transmit signal element timing
16	Secondary received data
17	Receive signal element timing
18	Unassigned
19	Secondary request to send
20	Data terminal ready ($\overline{\text{DTR}}$)
21	Signal quality detector
22	Ring indicator
23	Data signal rate select
24	Transmit signal element timing
25	Unassigned

Current terminology classifies data communication equipment as DTE (data terminal equipment) or DCE (data communication equipment). DTE refers to terminals and computers that send and receive data, while DCE refers to communication equipment, such as modems, that are responsible for transferring the data.

b) Examining RS232 handshaking signals

To ensure fast and reliable data transmission between two devices, the data transfer must be coordinated. Just as in the case of the printer, because the receiving device in serial data communication may have no room for the data, there must be a way to inform the sender to stop sending data. Many of the pins of the RS-232 connector are used for handshaking signals. Their descriptions are provided below only as a reference and they can be bypassed since they are not supported by the 8051 UARTchip.

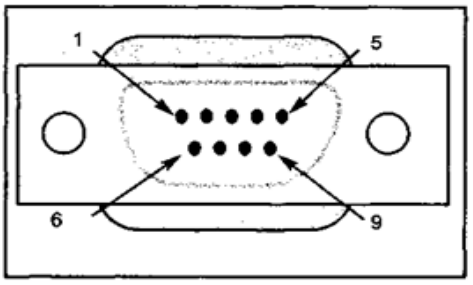


Figure 4.5: RS232 Connector DB-9

Table 4.2:DB-9 Pin connection

Pin	Description
1	Data carrier detect (DCD)
2	Received data (RxD)
3	Transmitted data (TxD)
4	Data terminal ready (DTR)
5	Signal ground (GND)
6	Data set ready (DSR)
7	Request to send (RTS)
8	Clear to send (CTS)
9	Ring indicator (RI)

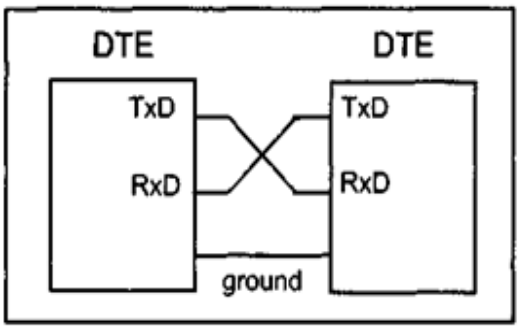


Figure 4.6: Null Modem connection

1. DTR (data terminal ready). When a terminal (or a PC COM port) is turned on, after going through a self-test, it sends out signal DTR to indicate that it is ready for communication. If there is something wrong with the COM port, this signal will not be activated. This is an active-low signal and can be used to inform the •modem that the computer is alive and kicking. This is an output pin from DTE (PC COM port) and an input to the modem.

2. DSR (data set ready). When DCE (modem) is turned on and has gone through the self-test, it asserts DSR to indicate that it is ready to communicate. Thus, it is an output from the modem (DCE) and input to the PC (DTE). This is an active- low signal. If for any reason the modem cannot make a connection to the telephone, this signal remains inactive, indicating to the PC (or terminal) that it cannot accept or send data.
3. RTS (request to send). When the DTE device (such as a PC) has a byte to transmit, it asserts RTS to signal the modem that it has a byte of data to transmit. RTS is an active-low output from the DTE and an input to the modem.
4. CTS (clear to send). In response to RTS, when the modem has room for storing the data it is to receive, it sends out signal CTS to the DTE (PC) to indicate that it can receive the data now. This input signal to the DTE is used by the DTE to start transmission.
5. DCD (carrier detect, or DCD, data carrier detect). The modem asserts signal DCD to inform the DTE (PC) that a valid carrier has been detected and that contact between it and the other modem is established. Therefore, DCD is an output from the modern and an input to the PC (DTE).
6. RI (ring indicator). An output from the modem (DCE) and an input to a PC (DTE) indicates that the telephone is ringing. It goes on and off in synchronization with the ringing sound. Of the six handshake signals, this is the least often used, due to the fact that modems take care of answering the phone. However, if the PC is in charge of answering the phone, this signal can be used.

4.2 8051 connection to RS242

a) RxD and TxD pins in the 8051

1. The 8051 has two pins that are used specifically for transferring and receiving data serially.
2. These two pins are called TxD and RxD and are part of the port 3 group (P3.0 and P3.1). Pin 11 of the 8051 (P3.1) is assigned to TxD and pin 10 (P3.0) is designated as RxD.

3. These pins are TTL compatible; therefore, they require a line driver to make them RS232 compatible. One such line driver is the MAX232 chip.

b)MAX232

1. Since the RS232 is not compatible with today's microprocessors and microcontrollers, we need a line driver (voltage converter) to convert the RS232′s signals to TTL voltage levels that will be acceptable to the 8051 's TxD and RxD pins.
2. One example of such a converter is MAX232 from Maxim Corp. (www.maxim-ic.com). The MAX232 converts from RS232 voltage levels to TTL voltage levels, and vice versa. One advantage of the MAX232 chip is that it uses a +5 V power source which, is the same as the source voltage for the 8051.
3. In other words, with a single +5 V power supply we can power both the 8051 and MAX232, with no need for the dual power supplies that are common in many older systems.
4. The MAX232 has two sets of line drivers for transferring and receiving data, as shown in

   Figure 4.7. The line drivers used for TxD are called Tl and T2, while the line drivers for RxD are designated as Rl and R2. In many applications only one of each is used. For example, Tl and Rl are used together for TxD and RxD of the 8051, and the second set is left unused. Notice in MAX232 that the Tl line driver has a designation of Tlin and Tlout on pin numbers 11 and 14, respectively.
5. The Tlin pin is the TTL side and is connected to TxD of the microcontroller, while Tlout is the RS232 side that is connected to the RxD pin of the RS232 DB connector. The Rl line driver has a designation of Rlin and Rlout on pin numbers 13 and 12, respectively. The Rlin (pin 13) is the RS232 side that is connected to the TxD pin of the RS232 DB connector, and Rlout (pin 12) is the TTL side that is connected to the RxD pin of the microcontroller.
6. MAX232 requires four capacitors ranging from 1 to 22 nF. The most widely used value for these capacitors is 22 nF.

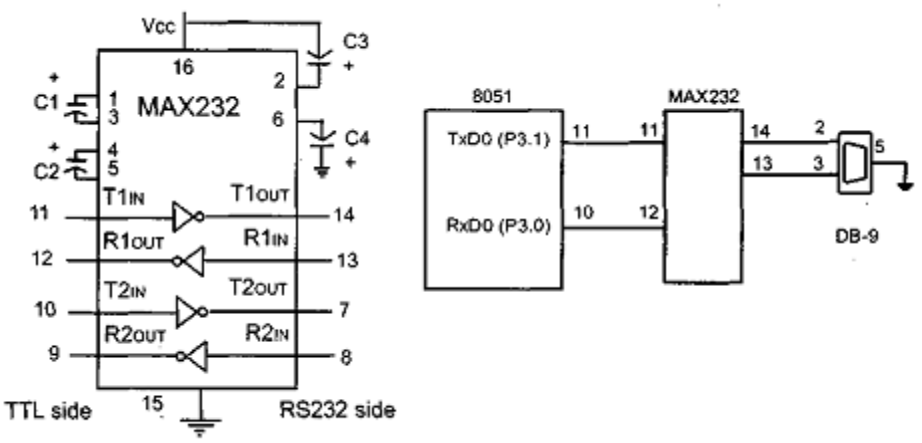


Figure 4.7. (a) Inside MAX232 and (b) its Connection to the 8051 (Null Modem)

4.3 8051 serial port programming in assembly

Baud rate in the 8051

1. The 8051 transfers and receives data serially at many different baud rates. The baud rate in the 8051 is programmable. This is done with the help of Timer 1. Before we discuss how to do that, we will look at the relationship between the crystal frequency and the baud rate in the 8051.
2. The 8051 divides the crystal frequency by 12 to get the machine cycle frequency. In the case of XTAL = 11.0592 MHz, the machine cycle frequency is 921.6 kHz (11.0592 MHz / 12 = 921.6 kHz). The 8051 ‘s serial communication UART circuitry divides the machine cycle frequency of 921.6 kHz by 32 once more before it is used by Timer 1 to set the baud rate. Therefore, 921.6 kHz divided by 32 gives 28,800 Hz.
3. This is the number we will use throughout this section to find the Timer 1 value to set the baud rate. When Timer 1 is used to set the baud rate it must be programmed in mode 2, that is 8-bit, auto-reload.

Table 4.3: Baud rate of TH1

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

Example 4-1

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

Solution:

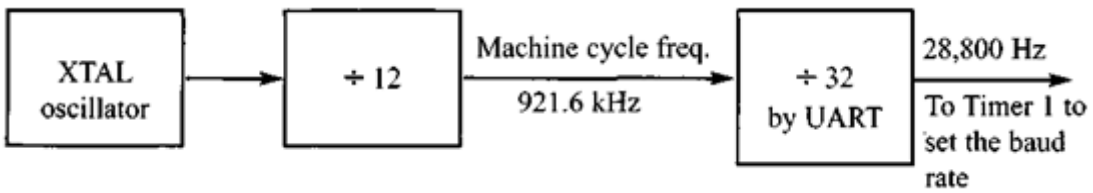
With XTAL = 11.0592 MHz, we have:

The machine cycle frequency of the 8051 = 11.0592 MHz / 12 = 921.6 kHz, and 921.6 kHz / 32 = 28,800 Hz is the frequency provided 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 1/12th of the crystal frequency divided by 32 is the default value upon activation of the 8051 RESET pin. We can change this default setting.

11.0592MHz



4.3.1 SBUF register

SBUF is an 8-bit register used solely for serial communication in the 8051. For a byte of data to be transferred via the TxD line, it must be placed in the SBUF register. Similarly, SBUF holds the byte of data when it is received by the 8051 ‘s RxD line. SBUF can be accessed like any other register in the 8051. Look at the following examples of how this register is accessed:

```
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 pin. 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 the SBUF.

4.3.2 SCON register

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

The following describes various bits of the SCON register.

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. (Make it 0.)
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 hardware at the beginning of the stop bit in mode 1. Must be cleared by software.
RI	SCON.0	Receive interrupt flag. Set by hardware halfway through the stop bit time in mode 1. Must be cleared by software.

Note: Make SM2, TB8, and RB8 = 0.

Figure 4.8:. SCON Serial Port Control Register (Bit-Addressable)

a) SMO, *SM1*

SMO and SMI are D7 and D6 of the SCON register, respectively. These two bits determine the framing of data by specifying the number of bits per character, and the start and stop bits. They take the following combinations.

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

In the SCON register, when serial mode 1 is chosen, the data framing is 8 bits, 1 stop bit, and 1 start bit, which makes it compatible with the COM port of IBM/compatible PCs. More importantly, serial mode 1 allows the baud rate to be variable and is set by Timer 1 of the 8051. In serial mode 1, 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 finally 1 stop bit.

b) SM2

SM2 is the D5 bit of the SCON register. This bit enables the multiprocessing capability of the 8051 and is beyond the discussion of this chapter. For our applications, we will make SM2 = 0 since we are not using the 8051 in a multiprocessor environment.

c) REN

The REN (receive enable), bit is D4 of the SCON register. The REN bit is also referred to as SCON.4 since SCON is a bit-addressable register. When the REN bit is high, it allows the 8051 to receive data on the RxD pin of the 8051. As a result if we want the 8051 to both transfer and receive data, REN must be set to 1. By making REN = 0, the receiver is disabled. Making REN — 1 or REN = 0 can

be achieved by the instructions "SETB SCON. 4″ and "CLR SCON. 4″, respectively. Notice that these instructions use the bit-addressable features of register SCON. This bit can be used to block any serial data reception and is an extremely important bit in the SCON register.

d) TBS

TBS (transfer bit 8) is bit D3 of SCON. It is used for serial modes 2 and 3. We make TBS = 0 since it is not used in our applications.

e) RB8

RB8 (receive bit 8) is bit D2 of the SCON register. In serial mode 1, this bit gets a copy of the stop bit when an 8-bit data is received. This bit (as is the case for TBS) is rarely used anymore. In all our applications we will make RB8 = 0. Like TB8, the RB8 bit is also used in serial modes 2 and 3.

f) Tl

TI (transmit interrupt) is bit Dl of the SCON register. This is an extremely important flag bit in the SCON register. When the 8051 finishes the transfer of the 8-bit character, it raises the TI flag to indicate that it is ready to transfer another byte. The TI bit is raised at the beginning of the stop bit. We will discuss its role further when programming examples of data transmission are given.

g) Rl

RI (receive interrupt) is the DO bit of the SCON register. This is another extremely important flag bit in the SCON register. 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 flag bit to indicate that a byte has been received and should be picked up before it is lost. RI is raised

halfway through the stop bit, and we will soon see how this bit is used in programs for receiving data serially.

4.3.3 Programming the 8051 to transfer data serially

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

1. The TMOD register is loaded with the value 20H, indicating the use of Timer 1 in mode 2 (8-bit auto-reload) to set the baud rate.
2. The TH1 is loaded with one of the values in Table 10-4 to set the 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.

1. TR1 is set to 1 to start Timer 1.
2. TI is cleared by the "CLR TI" instruction.
3. The character byte to be transferred serially is written into the SBUF register.
    1. The TI flag bit is monitored with the use of the instruction " JNB TI, xx" to see if the character has been transferred completely.
4. To transfer the next character, go to Step 5.

Example 4-2

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 be transferred
HERE:   JNB   TI,HERE    ;wait for the last bit
        CLR   TI         ;clear TI for next char
        SJMP  AGAIN      ;keep sending A
```

Example 4-3

Write a program to transfer the message "YES" serially at 9600 baud, 8-bit data, 1 stop bit. Do this continuously.

Solution:

```
            MOV    TMOD,#20H ;Timer 1, mode 2
            MOV    TH1,#-3   ;9600 baud
            MOV    SCON,#50H ;8-bit, 1 stop bit, 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 last bit to transfer
            CLR    TI        ;get ready for next byte
            RET
```

4.3.4 Importance of the Tl flag

To understand the importance of the role of TI, look at the following sequence of steps that the 8051 goes through 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 one bit at a time.

   The stop bit is transferred. It is during the transfer of the stop bit that the 8051 raises the TI flag (TI =1), indicating that the last character was transmitted and it is ready to transfer the next character.

   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. In other words, when the 8051 finishes
4. After SBUF is loaded with a new byte, the TI flag bit must be forced to 0 by the "CLR TI" instruction in order for this new byte to be transferred.

Example 4-4

Program the 8051 to receive bytes of data serially, and put them in PI. Set the baud rate at 4800, 8-bit data, and 1 stop bit.

Solution:

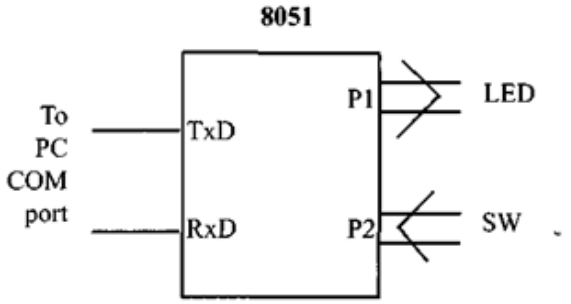
```
        MOV   TMOD,#20H  ;Timer 1, mode 2(auto-reload)
        MOV   TH1,#-6    ;4800 baud
        MOV   SCON,#50H  ;8-bit, 1 stop, REN enabled
        SETB  TR1        ;start Timer 1
HERE:   JNB   RI,HERE    ;wait for char to come in
        MOV   A,SBUF     ;save incoming byte in A
        MOV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

Example 4-5

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

Solution:

```
            ORG    0
            MOV    P2,#0FFH          ;make P2 an input port
            MOV    TMOD,#20H         ;Timer 1, mode 2(auto-reload)
            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
            MOVC   A,@A+DPTR         ;get the character
            JZ     B_1               ;if last character get 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 indefinitly
;--------------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                      ;return to caller
;-----------------The message
MYDATA:     DB "We Are Ready",0
            END
```



4.3.5 Importance of RI flag

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

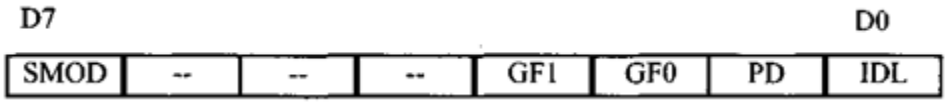
1. It receives the start bit indicating that the next bit is the first bit of the character byte it is about to receive.
2. The 8-bit character is received one bit at time. When the last bit is received, a byte is formed and placed in SBUF.

3. The stop bit is received. When receiving the stop bit the 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 the "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.

4.3.6 Doubling the baud rate in the 8051

There are two ways to increase the baud rate of data transfer in the 8051.

1. Use a higher-frequency crystal.
2. Change a bit in the PCON register, shown below.



Option 1 is not feasible in many situations since the system crystal is fixed. More importantly, it is not feasible because the new crystal may not be compatible with the IBM PC serial COM port's baud rate. Therefore, we will explore option 2. There is a software way to double the baud rate of the 8051 while the crystal frequency is fixed. This is done with the register called PCON (power control). The PCON register is an 8-bit register. Of the 8 bits, some are unused, and some are used for the power control capability of the 8051. The bit that is used for the serial communication is D7, the SMOD (serial mode) bit. When the 8051 is powered up, D7 (SMOD bit) of the PCON register is zero. We can set it to high by software and thereby double the baud rate. The following sequence of instructions must be used to set high D7 of PCON, since it is not a bit-addressable register:

```
MOV  A,PCON        ;place a copy of PCON in ACC
SETB ACC.7         ;make D7=1
MOV  PCON,A        ;now SMOD=1 without
                   ;changing any other bits
```

To see how the baud rate is doubled with this method, we show the role of the SMOD bit (D7 bit of the PCON register), which can be 0 or 1.

We discuss each case.

Baud rates for SMOD = 0

1. When SMOD = 0, the 8051 divides 1/12 of the crystal frequency by 32 and uses that frequency for Timer 1 to set the baud rate. In the case of XTAL = 11.0592 MHz we have:

Machine cycle freq. = 11.0592 MHz / 12 = 921.6 kHz

And 921.6 kHz / 32 = 28,800 Hz since SMOD = 0

This is the frequency used by Timer 1 to set the baud rate. This has been the basis of all the examples so far since it is the default when the 8051 is powered up.

Baud rates for SMOD = 1

With the fixed crystal frequency, we can double the baud rate by making SMOD – 1. When the SMOD bit (D7 of the PCON register) is set to 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. In the case of XTAL = 11.0592 MHz, we have:

Machine cycle freq. = 11.0592 MHz / 12 = 921.6 kHz

And 921.6 kHz / 16 = 57,600 Hz since SMOD = 1

This is the frequency used by Timer 1 to set the baud rate.

Table 4.4: Baud Rate Comparison for SMOD = 0 and SMOD = 1

TH1	(Decimal)	(Hex)	SMOD = 0	SMOD = 1
	-3	FD	9,600	19,200
	-6	FA	4,800	9,600
	-12	F4	2,400	4,800
	-24	E8	1,200	2,400

Note: XTAL = 11.0592 **MHz.**

Example 4-6

Assuming 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
        SETB  ACC.7       ;make D7 = 1
        MOV   PCON,A      ;SMOD = 1, double baud rate
                          ;with same XTAL freq.
        MOV   TMOD,#20H   ;Timer 1, mode 2(auto-reload)
        MOV   TH1,-3      ;19200 (57,600 / 3 = 19200 baud rate
                          ;since SMOD=1)
        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 and again
```

Solution:

1. This program transfers ASCII letter B (01000010 binary) continuously.
2. With XTAL = 11.0592 MHz and SMOD = 1 in the above program, we have:

11.0592 MHz / 12 = 921.6 kHz machine cycle frequency

921.6 kHz 716 = 57,600 Hz frequency used by Timer 1 to set the baud rate

57,600 Hz / 3 = 19,200 baud rate

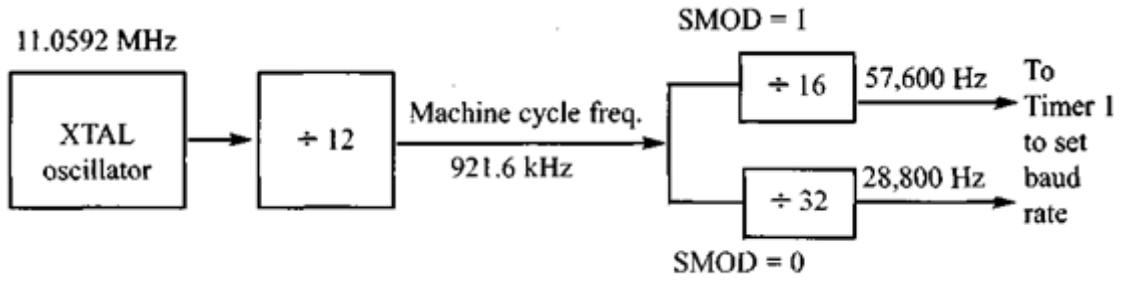
Example 10-7

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 MHz and SMOD = 1, we have Tinier 1 frequency = 57,600 Hz.

3. 57,600 / 9600 = 6; therefore, TH1 = -6 or TH1 = FAH.
4. 57,600 / 4800 = 12; therefore, TH1 = -12 or TH1 = F4H.



Example 4-8

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 MHz and SMOD = 1, we have Timer 1 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.

Example 4-9

Assume a switch is connected to pin PL7. Write a program to monitor its status and send two messages to serial port continuously as follows:

SW=0 send “NO”

SW=1 send “YES”

Assume XTAL = 11.0592 MHz, 9600 baud, 8-bit data, and 1 stop bit.

Solution:

```
            SW1     EQU P1.7
            ORG     0H              ;starting position
MAIN:       MOV     TMOD,#20H
            MOV     TH1,#-3         ;9600 baud rate
            MOV     SCON,#50H
            SETB    TR1             ;start timer
            SETB    SW1             ;make SW an input
S1:         JB      P2.1,NEXT       ;check SW status
            MOV     DPTR,#MESS1     ;if SW=0 display "NO"
FN:         CLR     A
            MOVC    A,@A+DPTR       ;read the value
            JZ      S1              ;check for end of line
            ACALL   SENDCOM         ;send value to serial port
            INC     DPTR            ;move to next value
            SJMP    FN              ;repeat
NEXT:       MOV     DPTR,#MESS2     ;if SW=1 display "YES"
LN:         CLR     A
            MOVC    A,@A+DPTR       ;read the value
            JZ      S1              ;check for end of line
            ACALL   SENDCOM         ;send value to serial port
            INC     DPTR            ;move to next value
            SJMP    LN              ;repeat
;-------------------
SENDCOM:    MOV     SBUF,A          ;place value in buffer
HERE:       JNB     TI,HERE         ;wait until transmitted
            CLR     TI              ;clear
            RET                     ;return
;-------------------
MESS1:      DB      "NO",0
MESS2:      DB      "YES",0
            END
```

4.4 Serial port programming in 8051 C

Transmitting and receiving data in 8051 C

Example 4.10

Write a C program for the 8051 to transfer the letter "A" serially at 4800 baud continuously. Use 8-bit data and 1 stop bit.

Solution:

```
#include <reg51.h>
void main(void)
  {
    TMOD=0x20;              //use Timer 1,8-BIT auto-reload
    TH1=0xFA;               //4800 baud rate
    SCON=0x50;
    TR1=1;
    while(1)
      {
        SBUF='A';           //place value in buffer
        while(TI==0);
        TI=0;
      }
  }
```

Example 4.11

Write an 8051 C program to transfer the message "YES" serially at 9600 baud, 8-bit data, 1 stop bit. Do this continuously.

Solution:

```
#include <reg51.h>
void SerTx(unsigned char);
void main(void)
  {
    TMOD=0x20;          //use Timer 1,8-BIT auto-reload
    TH1=0xFD;           //9600 baud rate
    SCON=0x50;
    TR1=1;              //start timer
    while(1)
      {
        SerTx('Y');
        SerTx('E');
        SerTx('S');
      }
  }
void SerTx(unsigned char x)
  {
    SBUF=x;             //place value in buffer
    while(TI==0);       //wait until transmitted
    TI=0;
  }
```

Example 4.12

Program the 8051 in C 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:

```
#include <reg51.h>
void main (void)
  {
    unsigned char mybyte;
    TMOD=0x20;                  //use Timer 1,8-BIT auto-reload
    TH1=0xFA;                   //4800 baud rate
    SCON=0x50;
    TR1=1;                      //start timer
    while(1)                    //repeat forever
      {
        while(RI==0);           //wait to receive
        mybyte=SBUF;            //save value
        P1=mybyte;              //write value to port
        RI=0;
      }
  }
```

Example 4.13

Write an 8051 C program to send two different strings to the serial port. Assuming that SW is connected to pin P2.0, monitor its status and make a decision as follows:

SW = 0: send your first name

SW = 1: send your last name

Assume XTAL = 11.0592 MHz, baud rate of 9600, 8-bit data, 1 stop bit.

Solution:

```
#include <reg51.h>
sbit MYSW=P2^0;                    //input switch
void main(void)
  {
    unsigned char z;
    unsigned char fname[]="ALI";
    unsigned char lname[]="SMITH";
    TMOD=0x20;                     //use Timer 1,8-BIT auto-reload
    TH1=0xFD;                      //9600 baud rate
    SCON=0x50;
    TR1=1;                         //start timer
    if(MYSW==0)                    //check switch
      {
        for(z=0;z<3;z++)           //write name
          {
            SBUF=fname[z];         //place value in buffer
            while(TI==0);          //wait for transmit
            TI=0;
          }
      }
    else
      {
        for(z=0;z<5;z++)           //write name
          {
            SBUF=lname[z];         //place value in buffer
            while(TI==0);          //wait for transmit
            TI=0;
          }
      }
  }
```

Example 4.14

Write an 8051 C program to send the two messages “Normal Speed” and “High Speed” to the serial port. Assuming that SW is connected to pin P2.0, monitor its status and set the baud rate as follows:

SW = 0 28,800 baud rate

SW = 1 56K baud rate

Assume that XTAL = 11.0592 MHz for both cases.

Solution:

```
#include <reg51.h>
sbit MYSW=P2^0;                  //input switch
void main(void)
  {
    unsigned char z;
    unsigned char Mess1[]="Normal Speed";
    unsigned char Mess2[]="High Speed";
    TMOD=0x20;                   //use Timer 1,8-BIT auto-reload
    TH1=0xFF;                    //28,800 for normal speed
    SCON=0x50;
    TR1=1;                       //start timer
    if(MYSW==0)
      {
        for(z=0;z<12;z++)
          {
            SBUF=Mess1[z];       //place value in buffer
            while(TI==0);        //wait for transmit
            TI=0;
          }
      }
    else
      {
        PCON=PCON|0x80;          //for high speed of 56K
        for(z=0;z<10;z++)
          {
            SBUF=Mess2[z];       //place value in buffer
            while(TI==0);        //wait for transmit
            TI=0;
          }
      }
  }
```

4.4.1 8051 C compilers and the second serial port

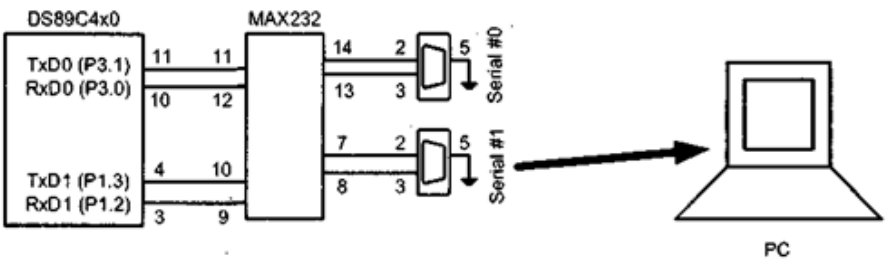
Since many C compilers do not support the second serial port of the DS89C4xO chip, we have to declare the byte addresses of the new SFR registers using the sfr keyword. Table 10-6 and Figure 10-12 provide the SFR byte and bit addresses for the DS89C4xO chip.

Example 4-15

Write a C program for the DS89C4xO to transfer letter "A" serially at 4800 baud continuously. Use the second serial port with 8-bit data and 1 stop bit. We can only use Timer 1 to set the baud rate.

Solution:

```
#include <reg51.h>
sfr SBUF1=0xC1;
sfr SCON1=0xC0;
sbit TI1=0xC1;
void main(void)
  {
    TMOD=0x20;          //use Timer 1 for 2nd serial port
    TH1=0xFA;           //4800 baud rate
    SCON1=0x50;         //use 2nd serial port SCON1 register
    TR1=1;              //start timer
    while(1)
      {
        SBUF1='A';      //use 2nd serial port SBUF1 register
        while(TI1==0); //wait for transmit
        TI1=0;
      }
  }
```



Example 4-16

Program the DS89C4xO in C to receive bytes of data serially via the second serial port and put them in PI. Set the baud rate at 9600, 8-bit data, and 1 stop bit. Use Timer 1 for baud rate generation.

Solution:

```
#include <reg51.h>
sfr SBUF1=0xC1;
sfr SCON1=0xC0;
sbit RI1=0xC0;
void main(void)
  {
    unsigned char mybyte;
    TMOD=0x20;              //use Timer 1,8-BIT auto-reload
    TH1=0xFD;               //9600
    SCON1=0x50;             //use SCON1 of 2nd serial port
    TR1=1;
    while(1)
      {
        while(RI1==0);      //monitor RI1 of 2nd serial port
        mybyte=SBUF1;       //use SBUF1 of 2nd serial port
        P2=mybyte;          //place value on port
        RI1=0;
      }
  }
```

4.5 8051 interrupts

Interrupts vs. polling

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

b) 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 associated with the interrupt is called **the *interrupt service routine* (ISR) or *interrupt handler***.

c) In ***polling,*** the microcontroller continuously monitors the status of a given device; when the status condition is met, it performs the service. After that, it moves on to monitor the next device until each one is serviced. Although polling can monitor the status of several devices and serve each of them as certain conditions are met, it is not an efficient use of the microcontroller.

d) The **advantage** of interrupts is that the microcontroller can serve many devices (not all at the same time, of course); each device can get the attention of the microcontroller based on the priority assigned to it. The polling method cannot assign priority since it checks all devices in a round-robin fashion. More importantly, in the interrupt method the microcontroller can also ignore (mask) a device request for service. This is again not possible with the polling method.

e)The most important reason that the interrupt method is preferable is that the polling method wastes much of the microcontroller's time by polling devices that do not need service. So in order to avoid tying down the microcontroller, interrupts are used. For example, in discussing timers in Chapter 9 we used the instruction "JNB TF, target", and waited until the timer rolled over, and while we were waiting we could not do anything else.

f)That is a waste of the microcontroller's time that could have been used to perform some useful tasks. In the case of the timer, if we use the interrupt method, the microcontroller can go about doing other tasks, and when the TF flag is raised the timer will interrupt the microcontroller in whatever it is doing.

Interrupt service routine

When an interrupt is invoked, the microcontroller runs the interrupt service routine. 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 the **interrupt vector table.**

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 interrupt service routine.
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. Then it starts to execute from that address.

4.5.1 Six interrupts in the 8051

In reality, only five interrupts are available to the user in the 8051, but many manufacturers' data sheets state that there are six interrupts since they include reset. The six interrupts in the 8051 are allocated as follows.

1. Reset. When the reset pin is activated, the 8051 jumps to address location 0000. This is the power-up reset.
2. Two interrupts are set aside for the timers: one for Timer 0 and one for Timer
3. Memory locations OOOBH and 001BH in the interrupt vector table belong to Timer 0 and Tinier 1, respectively.
4. Two interrupts are set aside for hardware external hardware interrupts. Pin numbers 12 (P3.2) and 13 (P3.3) in port 3 are for the external hardware interrupts INTO and INT1, respectively. These external interrupts are also referred to as EX1 and EX2. Memory locations 0003H and 0013H in the interrupt vector table are assigned to INTO and INT1, respectively.
5. Serial communication has a single interrupt that belongs to both receive and transmit. The interrupt vector table location 0023H belongs to this interrupt.

From Table 4.5 , also notice that only three bytes of ROM space are assigned to the reset pin. They are ROM address locations 0, 1, and 2. Address location 3 belongs to external hardware interrupt 0. For this reason, in our program we put the LJMP as the first instruction and redirect the processor away from the interrupt vector table

Table 4.5; Interrupt Vector Table for the 8051

Interrupt	ROM Location (Hex)	Pin	Flag Clearing
Reset	0000	9	Auto
External hardware interrupt 0 (INT0)	0003	P3.2 (12)	Auto
Timer 0 interrupt (TF0)	000B		Auto
External hardware interrupt 1 (INT1)	0013	P3.3 (13)	Auto
Timer 1 interrupt (TF1)	001B		Auto
Serial COM interrupt (RI and TI)	0023		Programmer clears it.

```
        ORG  0    ;wake-up ROM reset location
        LJMP MAIN ;bypass interrupt vector table

;---- the wake-up program
        ORG  30H
MAIN:
        ....
        END
```

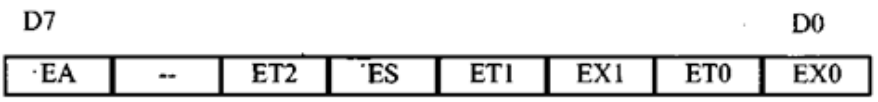
Figure 4.9. Redirecting the 8051 from the Interrupt Vector Table at Power-up

4.5.2 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. Note that IE is a bit-addressable register.

Steps in enabling an interrupt

1. To enable an interrupt, we take the following steps: .1. Bit D7 of the IE register (EA) must be set to high to allow the rest of register to take effect.
2. If EA = 1, interrupts are enabled and will be responded to if their corresponding bits in IE are high. If EA = 0, no interrupt will be responded to, even if the associated bit in the IE register is high.



EA	IE.7	Disables all interrupts. If EA = 0, no interrupt is acknowledged. If EA = 1, each interrupt source is individually enabled or disabled by setting or clearing its enable bit.
--	IE.6	Not implemented, reserved for future use.*
ET2	IE.5	Enables or disables Timer 2 overflow or capture interrupt (8052 only).
ES	IE.4	Enables or disables the serial port interrupt.
ET1	IE.3	Enables or disables Timer 1 overflow interrupt.
EX1	IE.2	Enables or disables external interrupt 1.
ET0	IE.1	Enables or disables Timer 0 overflow interrupt.
EX0	IE.0	Enables or disables external interrupt 0.

*User software should not write 1s to reserved bits. These bits may be used in future flash microcontrollers to invoke new features.

Figure 4.10. IE (Interrupt Enable) Register

Example 4-16

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
```

Since IE is a bit-addressable register, we can use the following instructions to access individual bits of the register.

```
(b)   CLR IE.1        ;mask(disable) Timer 0 interrupt only
(c)   CLR IE.7        ;disable all interrupts
```

Another way to perform the "MOV IE,#10010110B" instruction is by using single-bit instructions as shown below.

```
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
```

4.6 Programming timer

.

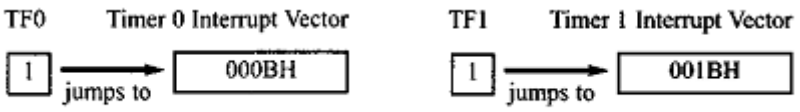


Figure 4.11. TF Interrupt

Roll-over timer flag and interrupt

1. Timer flag (TF) is raised when the timer rolls over. In that chapter, we also showed how to monitor TF with the instruction "JNB TF, target". In polling TF, we have to wait until the TF is raised.
2. The problem with this method is that the microcontroller is tied down while waiting for TF to be raised, and cannot do any thing else. Using interrupts solves this problem and avoids tying down the controller.
3. 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.

4. In this way, the microcontroller can do other things until it is notified that the timer has rolled over

Example 4-17

Write a program that continuously gets 8-bit data from PO and sends it to PI while simultaneously creating a square wave of 200 (as 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 μs = 92.

```
;--Upon wake-up go to main, avoid using memory space ;allocat-
ed to Interrupt Vector Table
            ORG   0000H
            LJMP  MAIN        ;bypass 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              ;return from ISR
;
;--The main program for initialization
            ORG   0030H       ;after vector table space
MAIN:       MOV   TMOD,#02H   ;Timer 0, mode 2(auto-reload)
            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
```

Example 4-18

Write a program o create a square wave that has a high portion of 1085 us and a low portion of 15 us. Assume XTAL = 11.0592 MHz. 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 space
;--allocated to Interrupt Vector Table
          ORG   0000H
          LJMP  MAIN        ;bypass interrupt vector table
;
;--ISR for Timer 1 to generate square wave
          ORG   001BH       ;Timer 1 interrupt vector table
          LJMP  ISR_T1      ;jump to ISR
;
;--The main program for initialization
          ORG   0030H       ;after vector table
MAIN:     MOV   TMOD,#10H   ;Timer 1, mode 1
          MOV   P0,#0FFH    ;make P0 an input port
          MOV   TL1,#018H   ;TL1=18 the Low byte of -1000
          MOV   TH1,#0FCH   ;TH1=FC the High byte of -1000
          MOV   IE,#88H     ;IE=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 since not auto-reload
ISR_T1:   CLR   TR1         ;stop Timer 1
          CLR   P2.1        ;P2.1=0, start of low portion
          MOV   R2,#4       ;                              2 MC
HERE:     DJNZ  R2,HERE     ;4x2 machine cycle(MC)         8 MC
          MOV   TL1,#18H    ;load T1 Low byte value        2 MC
          MOV   TH1,#0FCH   ;load T1 High byte value       2 MC
          SETB  TR1         ;starts Timer 1                1 MC
          SETB  P2.1        ;P2.1=1, back to high          1 MC
          RETI              ;return to main
          END
```

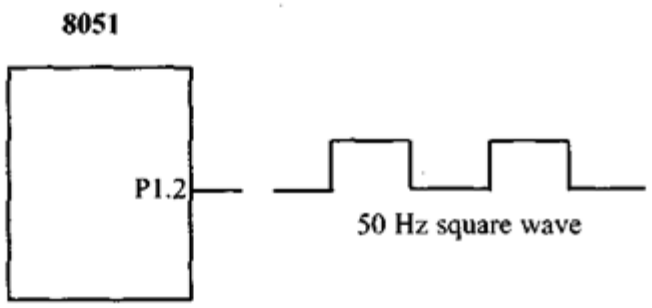
Notice that the low portion of the pulse is created by the 14 MC (machine cycles) where each MC = 1.085 us and 14 x 1.085 us = 15.19 us.

Example 4-19

Write a program to generate a square wave of 50 Hz frequency on pin PI .2. UseTimer 0. Assume that XTAL = 11.0592MHz.

Solution:

```
            ORG   0
            LJMP  MAIN
            ORG   000BH             ;ISR for Timer 0
            CPL   P1.2              ;complement P1.2
            MOV   TL0,#00           ;reload timer values
            MOV   TH0,#0DCH
            RETI                    ;return from interrupt
            ORG   30H               ;starting location for prog.
;------main program for initialization
MAIN:       MOV   TMOD,#00000001B   ;Timer 0, Mode 1
            MOV   TL0,#00
            MOV   TH0,#0DCH
            MOV   IE,#82H           ;enable Timer 0 interrupt
            SETB  TR0               ;start timer
HERE:       SJMP  HERE              ;stay here until interrupted
            END
```



4.7 External hardware interrupts

The 8051 has two external hardware interrupts. Piri 12 (P3.2) and pin 13 (P3.3) of the 8051, designated as INTO 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.

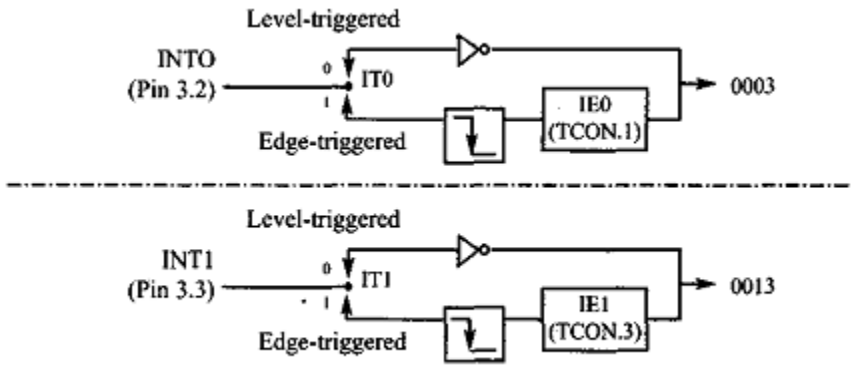


Figure 4.12: Activation of INTO and INT1

a) External interrupts INTO and INT1

There are only two external hardware interrupts in the 8051: INTO and INT1. They are located on pins P3.2 and P3.3 of port 3, respectively. The interrupt vector table locations 0003H and 0013H are set aside for INTO and INT1, respectively. As mentioned in Section 11.1, they are enabled and disabled using the IE register. How are they activated? There are two types of activation for the external hardware interrupts: (1) level triggered, and (2) edge triggered. Let's look at each one. First, we see how the level-triggered interrupt works.

4.7.1 Level-triggered interrupt

In the level-triggered mode, INTO and INT1 pins are normally high (just like all I/O port pins) and 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 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 interrupt service routine, 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.

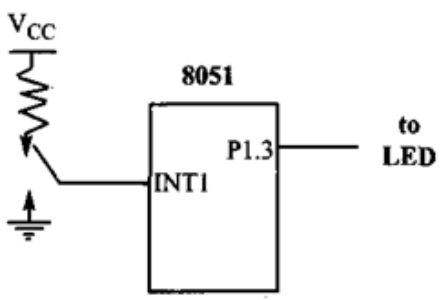
Example 4.20

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 PI .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              ;bypass interrupt vector table
;--ISR for hardware interrupt INT1 to turn on the LED
            ORG   0013H             ;INT1 ISR
            SETB  P1.3              ;turn on LED
            MOV   R3,#255           ;load counter
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 INT1
HERE:       SJMP  HERE              ;stay here until interrupted
            END
```

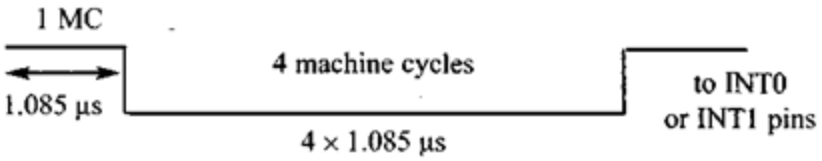
Pressing the switch will turn the LED on. If it is kept activated, the LED stays on.



4.7.2 Sampling the low level-triggered interrupt

1. Pins P3.2 and P3.3 are used for normal I/O unless the INTO and INT1 bits in the IE registers are enabled. After the hardware interrupts in the IE register are enabled, the controller keeps sampling the INT« 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.
2. If the INTn pin is brought back to a 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. Again, according to one manufacturer's data sheet, "If the INTw 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 INTw pin, make sure that the duration of the low-level signal is around 4 machine cycles,

but no more. This is due to the 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.



Note: On RESET, IT0 (TCON.0) and IT1 (TCON.2) are both low, making external interrupts level-triggered.

Figure 4.13:. Minimum Duration of the Low Level-Triggered Interrupt (XTAL = 11.0592 MHz)

4.7.3 Edge-triggered interrupts

As stated before, upon reset the 8051 makes INTO and INT1 low-level triggered interrupts. To make them edge-triggered interrupts, we must program the bits of the TCON register. The TCON register holds, among other bits, the ITO and IT1 flag bits that determine level- or edge-triggered mode of the hardware interrupts. ITO and IT1 are bits DO and D2 of the TCON register, respectively. They are also referred to as TCON.O and TCON.2 since the TCON register is bit-addressable. Upon reset, TCON.O (ITO) and TCON.2 (III) are both Os, meaning that the external hardware interrupts of INTO and INT1 pins are low-level triggered. By making the TCON.O and TCON.2 bits high with instructions such as "SETB TCON. 0″ and "SETB TCON. 2″, the external hardware interrupts of INTO and INT1 become edge-triggered. For example, the instruction "SETB CON. 2″ makes INT1 what is called an *edge-triggered interrupt*, in which, when a high-to-low signal is applied to pin P3.3, in this case, the controller will be interrupted and forced to jump to location 0013H in the vector table to service the ISR (assuming that the interrupt bit is enabled in the IE register).

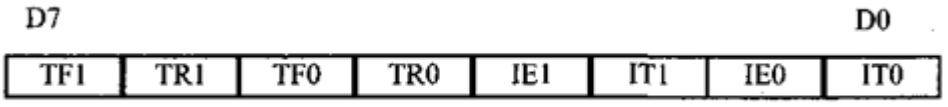


Figure 4.14:. TCON (Timer/Counter) Register (Bit-addressable)

TF1 TCON.7 Timer 1 overflow flag. Set by hardware when timer/counter 1 overflows. Cleared by hardware as the processor vectors to the interrupt service routine.

TR1 TCON.6 Timer 1 run control bit. Set/cleared by software to turn timer/counter 1 on/off.

TF0 TCON.5 Timer 0 overflow flag. Set by hardware when timer/counter 0 overflows. Cleared by hardware as the 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 CPU when the external interrupt edge (H-to-L transition) is detected. Cleared by CPU when the interrupt is processed. *Note:* This flag does not latch low-level triggered interrupts.

IT1 TCON.2 Interrupt 1 type control bit. Set/cleared by software to specify falling edge/low-level triggered external interrupt.

IE0 TCON.1 External interrupt 0 edge flag. Set by CPU when external interrupt (H-to-L transition) edge is detected. Cleared by CPU when interrupt is processed. *Note:* This flag does not latch low-level triggered interrupts.

IT0 TCON.0 Interrupt 0 type control bit. Set/cleared by software to specify falling edge/low-level triggered external interrupt.

Example 4-21

Assuming 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 PI.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. This is an edge-triggered version of Example 11-5.

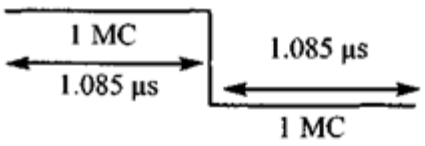
Solution:

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

4.7.4 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 = 11.0592 MHz



1. 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 INTO and INT1, respectively. TCON.l and TCON.3 are also called IEO and IE1, respectively, as shown in Figure 11-6. They function as interrupt-in-service flags.
2. 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 INTw 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 ITO and IT1 bits in the TCON register, the following two points must be emphasized.

Example 4-22

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.

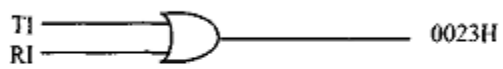
Solution:

Both perform the same actions of popping off the top two bytes of the stack into the program counter, and making 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 TFO, TF1, TCON.l, and TCON.3, they are cleared by the execution of RETI.

4.8 Serial communication interrupt

Rl and Tl flags and interrupts

1. 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. In other words, 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.
2. As far as serial communication is concerned, 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 polling method, we wait for the flag (TI or RI) to be raised; while we wait we cannot do anything else. In the interrupt method, we are notified when the 8051 has received a byte, or is ready to send the next byte; we can do other things while the serial communication needs are served.
3. 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

Figure 4.15:. Single Interrupt for Both TI and RI

Example 4-23

Write a program in which the 8051 reads data from PI 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 MHz. Set the baud rate at 9600.

Solution:

```
            ORG   0
            LJMP  MAIN
            ORG   23H
            LJMP  SERIAL          ;jump to serial interrupt ISR
            ORG   30H
MAIN:       MOV   P1,#0FFH        ;make P1 an input port
            MOV   TMOD,#20H       ;timer 1, mode 2(auto-reload)
            MOV   TH1,#0FDH       ;9600 baud rate
            MOV   SCON,#50H       ;8-bit, 1 stop, REN enabled
            MOV   IE,#10010000B   ;enable serial interrupt
            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 does not
            RETI                  ;return from ISR
TRANS:      CLR   TI              ;clear TI since CPU does not
            RETI                  ;return from ISR
            END
```

Example 4-24

Write a program in which the 8051 gets data from PI and sends it to P2 continuously while incoming data from the serial port is sent to PO. Assume that XTAL = 11.0592 MHz. Set the baud rate at 9600.

Solution:

```
            ORG   0
            LJMP  MAIN
            ORG   23H
            LJMP  SERIAL          ;jump to serial ISR
            ORG   30H
MAIN:       MOV   P1,#0FFH        ;make P1 an input port
            MOV   TMOD,#20H       ;timer 1, mode 2(auto-reload)
            MOV   TH1,#0FDH       ;9600 baud rate
            MOV   SCON,#50H       ;8-bit,1 stop, REN enabled
            MOV   IE,#10010000B   ;enable serial interrupt
            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
```

4.9 Interrupt priority in 8051/52

Interrupt priority upon reset

When the 8051 is powered up, the priorities are assigned according to Table 4.6.

Table 4.6: Interrupt priority

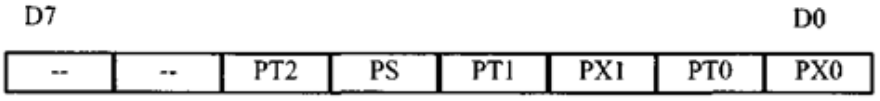
Highest to Lowest Priority	
External Interrupt 0	(INT0)
Timer Interrupt 0	(TF0)
External Interrupt 1	(INT1)
Timer Interrupt 1	(TF1)
Serial Communication	(RI + TI)
Timer 2 (8052 only)	**TF2**

Example 4-26

Discuss what happens if interrupts INTO, TFO, and INT1 are activated at the same time. Assume priority levels were set by the power-up reset and that 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 4.6.If any is activated, it services it in sequence. Therefore, when the above three interrupts are activated, IEO (external interrupt 0) is serviced first, then Timer 0 (TFO), and finally IE1 (external interrupt 1).



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

--	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

User software should never write 1s to unimplemented bits, since they may be used in future products.

Figure 4.16:. Interrupt Priority Register (Bit-addressable)

Example 4-27

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

Solution:

1. MOV IP,#000001006 ;IP.2 = 1 to assign INT1 higher priority
   The instruction "SETB IP.2″ also will do the same thing as the above line since IP is bit-addressable.
2. The instruction in Step (a) assigned a higher priority to INT1 than the others; therefore, when INTO, INT1, and TFO interrupts are activated at the same time, the 8051services INT1 first, then it services INTO, then TFO. 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 INTO and TFO bits in the IP register 0.

Example 4-28

Assume that after reset, the interrupt priority is set by the instruction "MOV IP, 400001100B". Discuss the sequence in which the interrupts are serviced.

Solution:

The instruction "MOV IP, #0 0 0 0110 OB" (B is for binary) sets the external interrupt 1 (INT1) and Timer 1 (TF1) to a higher priority level compared with the rest of the interrupts. However, since they are polled according to Table 11-3, 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)

4.10 Interrupt programming in C

The 8051 C compilers have extensive support for the 8051 interrupts with two major features as follows:

1. They assign a unique number to each of the 8051 interrupts, as shown in Table 11-4.
2. It can also assign a register bank to an ISR. This avoids code overhead due to the pushes and pops of the RO – R7 registers.

Table 4.7: 8051/52 Interrupt Numbers in C

Interrupt	Name	Numbers used by 8051 C
External Interrupt 0	(INT0)	0
Timer Interrupt 0	(TF0)	1
External Interrupt 1	(INT1)	2
Timer Interrupt 1	(TF1)	3
Serial Communication	(RI + TI)	4
Timer 2 (8052 only)	(TF2)	5

Example 4-29

Write a C program that continuously gets a single bit of data from PI. 7 and sends it to Pl.0, while simultaneously creating a square wave of 200 (as period on pin P2.5. Use timer 0 to create the square wave. Assume that XTAL = 11.0592 MHz.

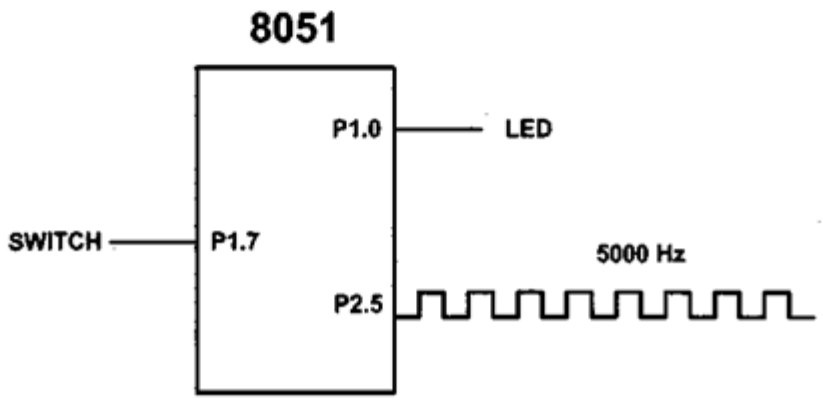
Solution:

We will use timer 0 in mode 2 (auto-reload). One half of the period is 100 µs.
100 /1. 085 µs = 92, and TH0 = 256 – 92 = 164 or A4H

```
#include <reg51.h>

sbit SW     = P1^7;
sbit IND    = P1^0;
sbit WAVE   = P2^5;

void timer0(void) interrupt 1
  {
    WAVE = ~WAVE;     //toggle pin
  }

void main()
  {
    SW = 1;           //make switch input
    TMOD = 0x02;
    TH0 = 0xA4;       //TH0 = -92
    IE = 0x82;        //enable interrupts for timer 0
    while(1)
      {
        IND = SW;     //send switch to LED
      }
  }
```

200 µs / 2 = 100 µs

100 µs / 1.085 µs = 92



Example 4-30

Write a C program that continuously gets a single bit of data from PI. 7 and sends it to Pl.0 in the main, while simultaneously (a) creating a square wave of 200 us period on pin P2.5, and

(b) sending letter 'A' to the serial port. Use Timer 0 to create the square wave. Assume that XTAL = 11.0592 MHz. Use the 9600 baud rate.

Solution:

We will use Timer 0 in mode 2 (auto-reload). TH0 = 100/1.085 μs = –92, which is A4H

```
#include <reg51.h>

sbit SW    = P1^7;
sbit IND   = P1^0;
sbit WAVE  = P2^5;

void timer0(void) interrupt 1
  {
    WAVE = ~WAVE;     //toggle pin
  }

void serial0() interrupt 4
  {
    if(TI == 1)
      {
        SBUF = 'A';  //send A to serial port
        TI = 0;      //clear interrupt
      }
    else
      {
        RI = 0;      //clear interrupt
      }
  }

void main()
  {
    SW = 1;          //make switch input
    TH1 = -3;        //9600 baud
    TMOD = 0x22;     //mode 2 for both timers
    TH0 = 0xA4;      //-92=A4H for timer 0
    SCON = 0x50;
    TR0 = 1;
    TR1 = 1;               //start timer
    IE = 0x92;       //enable interrupt for T0
    while(1)         //stay here
      {
        IND = SW;    //send switch to LED
      }
  }
```

Example 4-31

Write a C program using interrupts to do the following:

1. Receive data serially and send it to P0,
2. Read port PI, transmit data serially, and give a copy to P2,
3. Make timer 0 generate a square wave of 5 kHz frequency on P0.l.

Assume that XTAL = 11.0592 MHz. Set the baud rate at 4800.

Solution:

```
#include <reg51.h>
sbit WAVE = P0^1;

void timer0() interrupt 1
  {
    WAVE = ~WAVE;          //toggle pin
  }

void serial0() interrupt 4
  {
    if(TI == 1)
      {
        TI = 0;            //clear interrupt
      }
    else
      {
        P0 = SBUF;         //put value on pins
        RI = 0;            //clear interrupt
      }
  }

void main()
  {
    unsigned char x;
    P1 = 0xFF;             //make P1 an input
    TMOD = 0x22;
    TH1 = 0xF6;            //4800 baud rate
    SCON = 0x50;
    TH0 = 0xA4;            //5 kHz has T = 200 µs
    IE = 0x92;             //enable interrupts
    TR1 = 1;               //start timer 1
    TR0 = 1;               //start timer 0
    while(1)
      {
        x = P1;            //read value from pins
        SBUF = x;          //put value in buffer
        P2 = x;            //write value to pins
      }
  }
```

Example 4-32

Write a C program using interrupts to do the following:

1. Generate a 10000 Hz frequency on P2.1 using TO 8-bit auto-reload,
2. Use timer 1 as an event counter to count up a 1-Hz pulse and display it on PO. The pulse is connected to EX1.

Assume that XTAL = 11.0592 MHz. Set the baud rate at 9600.

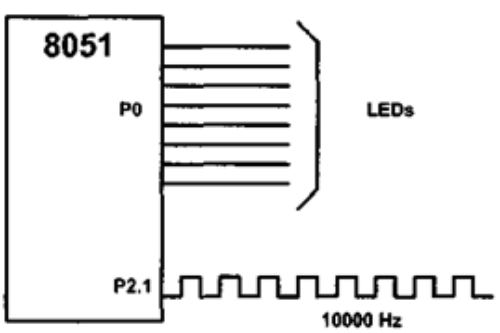
Solution:

```
#include <reg51.h>

sbit WAVE = P2^1;
unsigned char cnt;

void timer0() interrupt 1
  {
    WAVE = ~WAVE;          //toggle pin
  }
void timer1() interrupt 3
  {
    cnt++;                 //increment counter
    P0 = cnt;              //display value on pins
  }

void main()
  {
    cnt = 0;               //set counter to zero
    TMOD = 0x42;
    TH0 = 0x-46;           //10000 Hz
    IE = 0x86;             //enable interrupts
    TR0 = 1;               //start timer 0
    TR1 = 1;               //start timer 1
    while(1);              //wait until interrupted
  }
```

1 / 10000 Hz = 100 µs

100 µs / 2 = 50 µs

50 µs / 1.085 µs = 46



Outcomes

At the end of the module, students will be able

CO4: Analyse different I/O devices (Serial), interrupts and develop programs to configure 8051 Microcontroller. **[L4, MODULE 4]**