

BEE403:MICROCONTROLLER

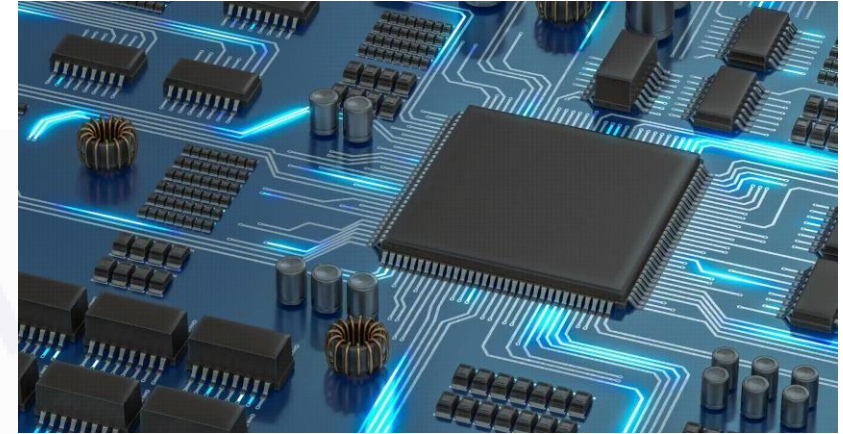
MODULE – 4:

8051 Serial Communication_ 8051 Interrupt programming in assembly and C



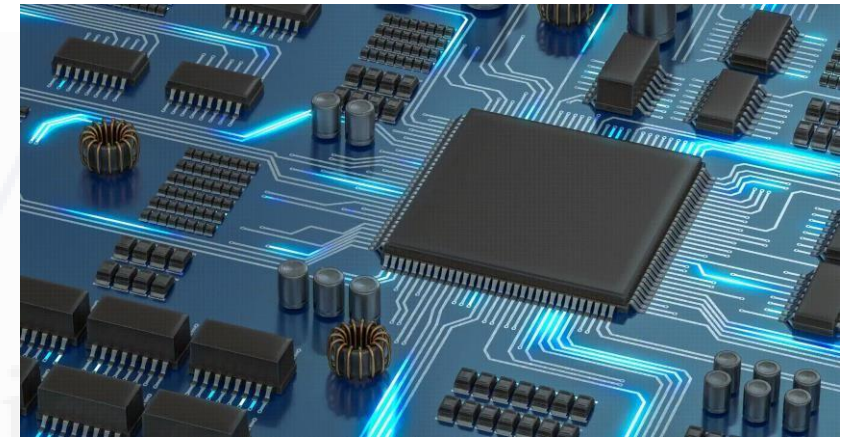
OUTLINE

- 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



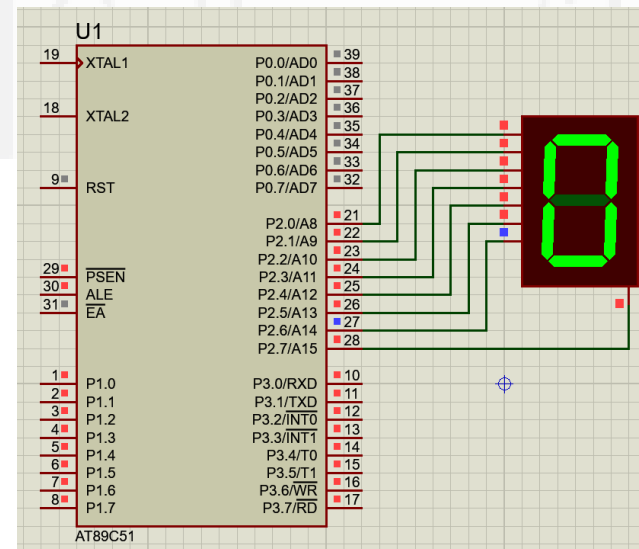
OUTLINE

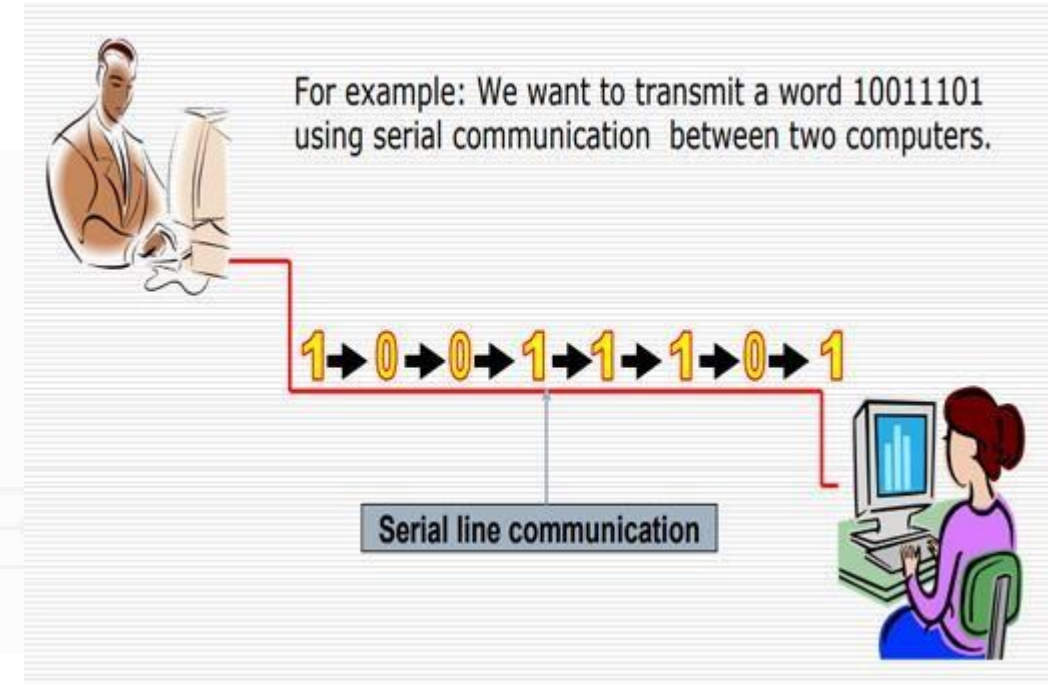
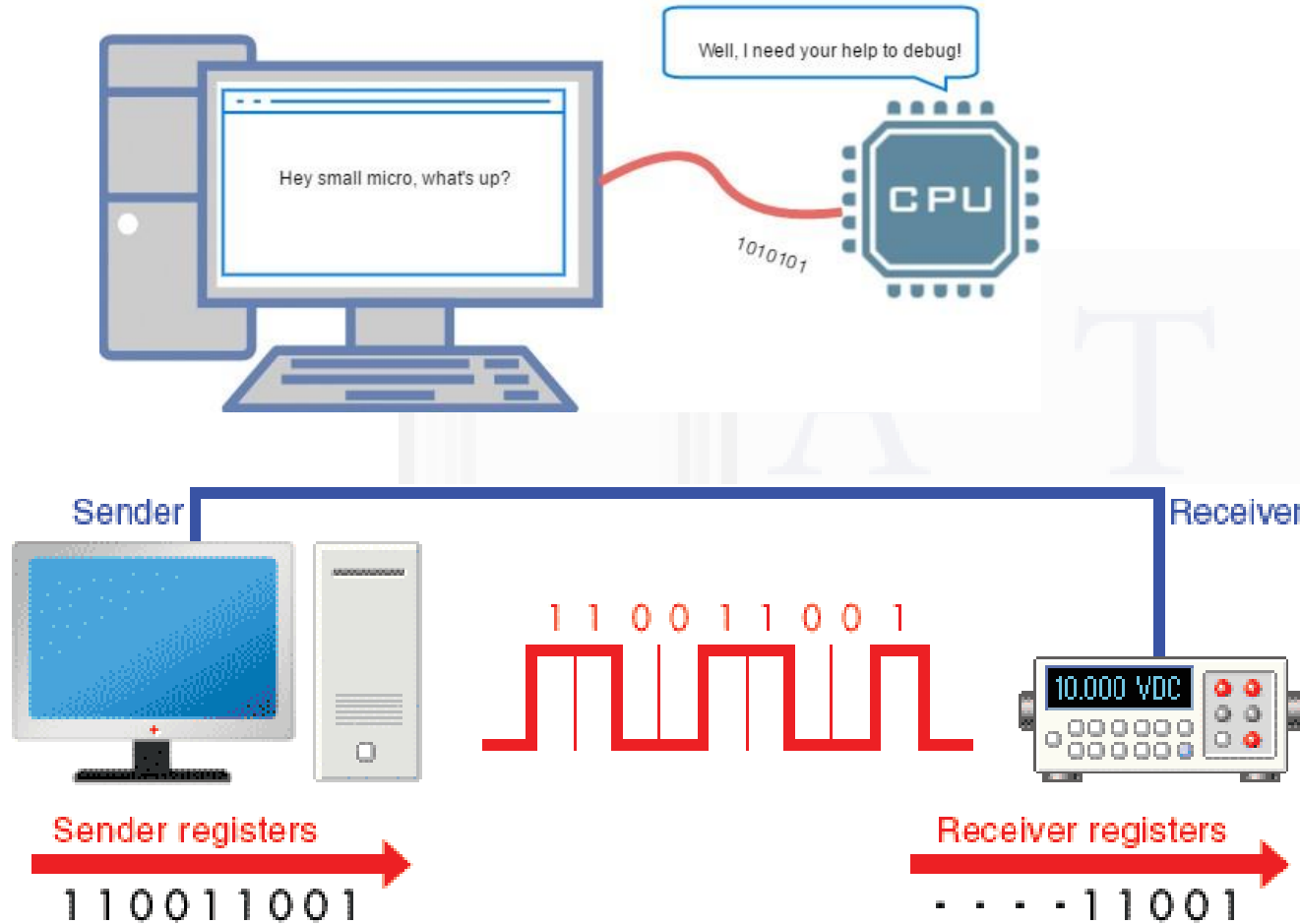
- **8051 Interrupt programming in assembly and C:**
- 4.5 8051 interrupts
- 4.6 Programming timer
- 4.7 External hardware
- 4.8 serial communication interrupt
- 4.9 Interrupt priority in 8051/52
- 4.10 Interrupt programming in C





4.1





4.1 Basics of serial communication

- 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.

4.1 Basics of serial communication

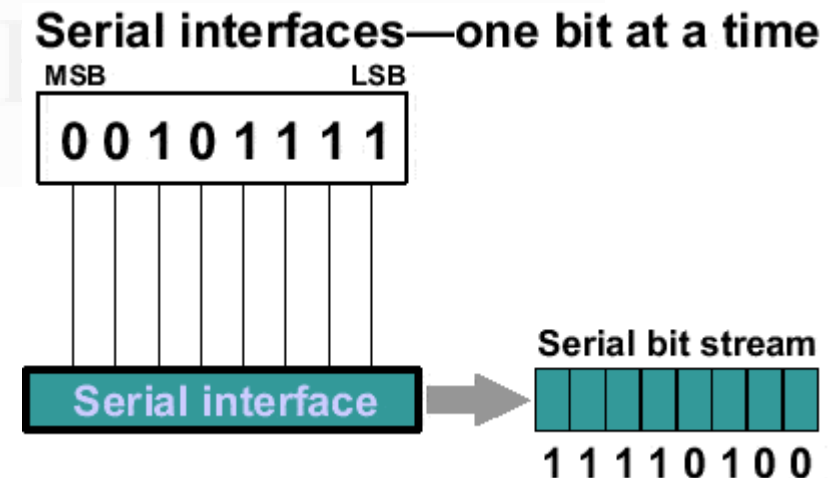
- This can work only if the cable is not too long, since long cables diminish and even distort signals.
- Furthermore, an 8-bit data path is **expensive**. For these reasons, serial communication is used for transferring data between two systems located at **distances of hundreds of feet to millions of miles apart.**

4.1 Basics of serial communication

- 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.

4.1 Basics of serial communication

- 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



4.1 Basics of serial communication

- 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)**.

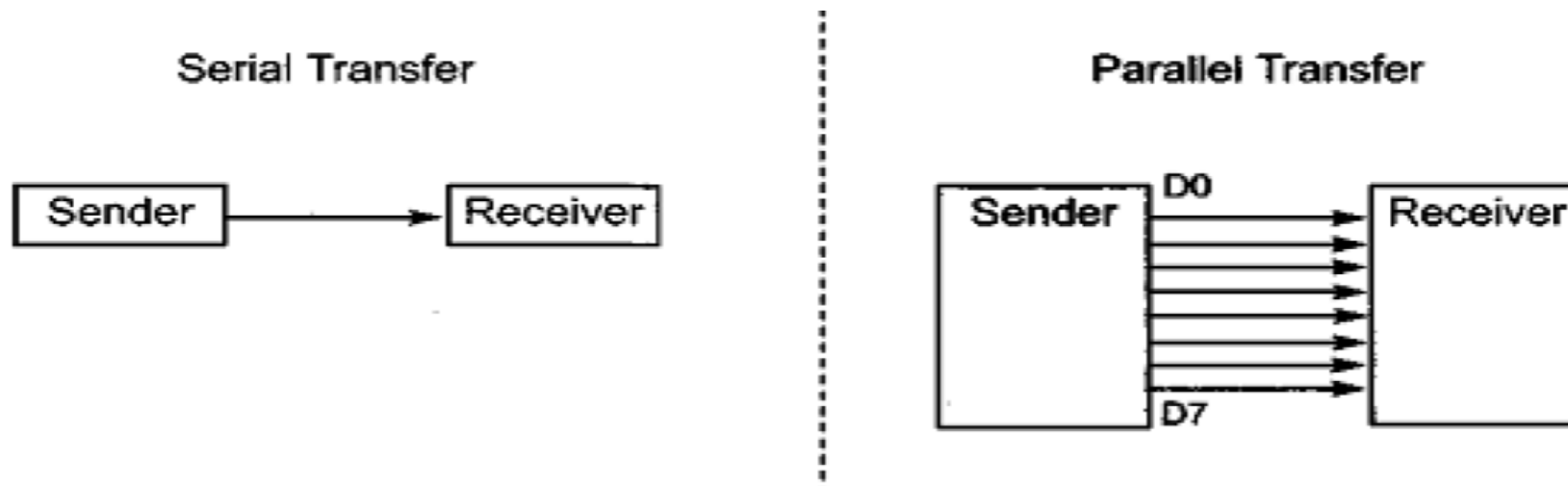


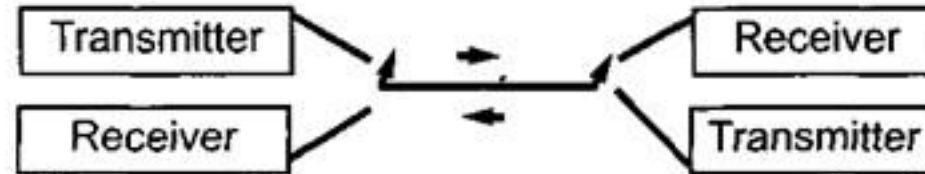
Figure 4-1. Serial versus Parallel Data Transfer

Serial versus Parallel Data Transfer

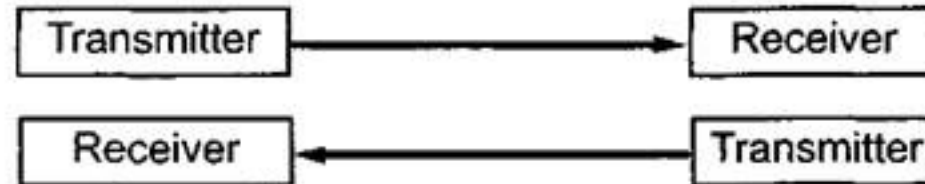
Simplex



Half Duplex



Full Duplex



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*

b) Asynchronous serial communication and data framing

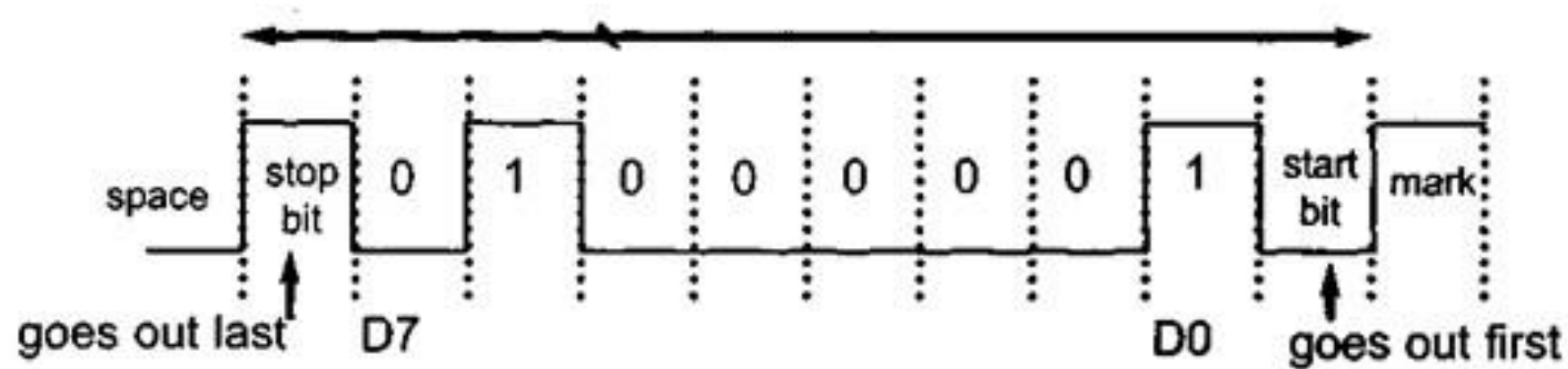
- 1. The data coming in at the receiving end of the data line in a serial data transfer is all **0s and 1s**;
- 2. It is difficult to make sense of the data unless the sender and receiver agree on a set of rules, a ***protocol***, on how the data is packed, how many bits constitute a character, and when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

Framing ASCII “A” (41H)



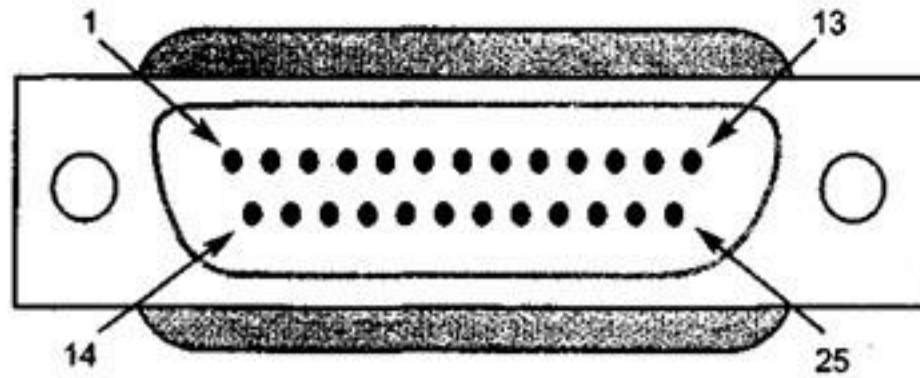
The rate of data transfer in serial data communication is stated in *bps* (bits per second). Another widely used terminology for bps is *baud rate*

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.

- 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.

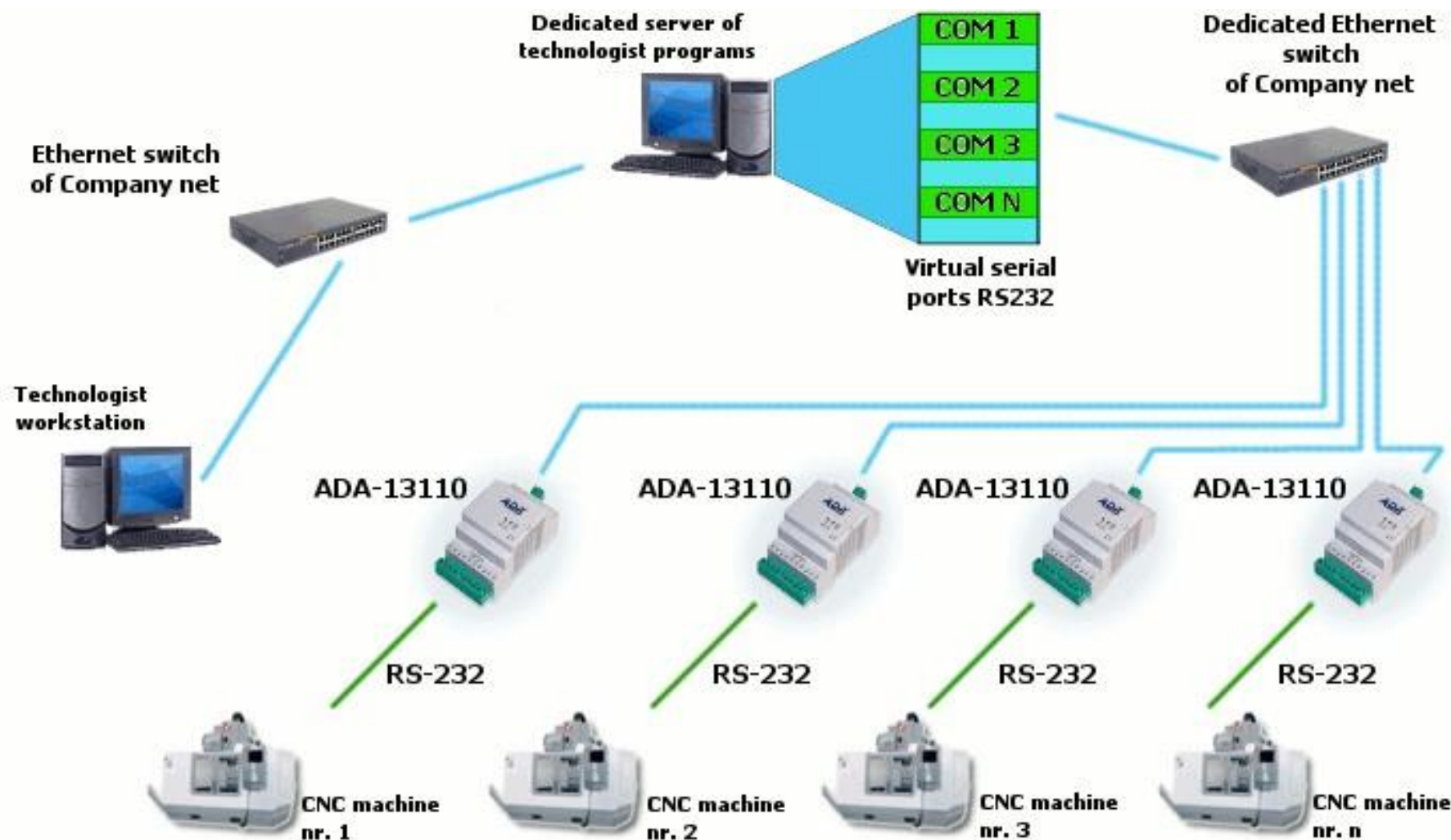
RS232 pins

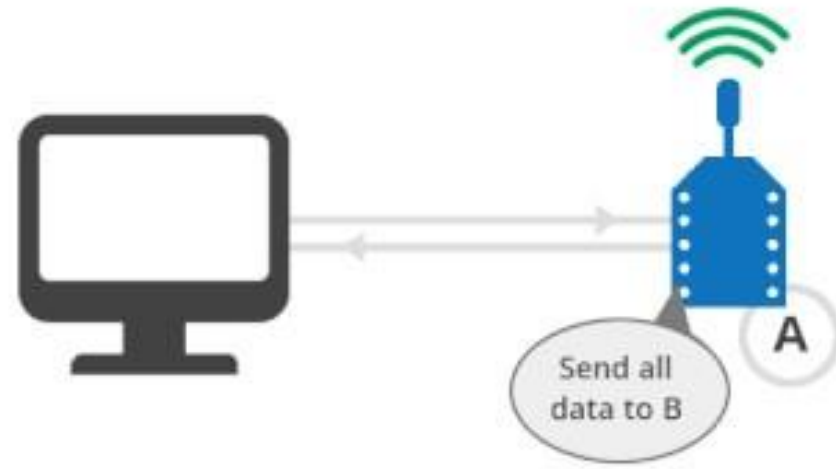


On to the leading edge
www.atme.in

Pin	Description
1	Protective ground
2	Transmitted data (TxD)
3	Received data (RxD)
4	Request to send (RTS)
5	Clear to send (CTS)
6	Data set ready (DSR)
7	Signal ground (GND)
8	Data carrier detect (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 (DTR)
21	Signal quality detector
22	Ring indicator
23	Data signal rate select
24	Transmit signal element timing
25	Unassigned

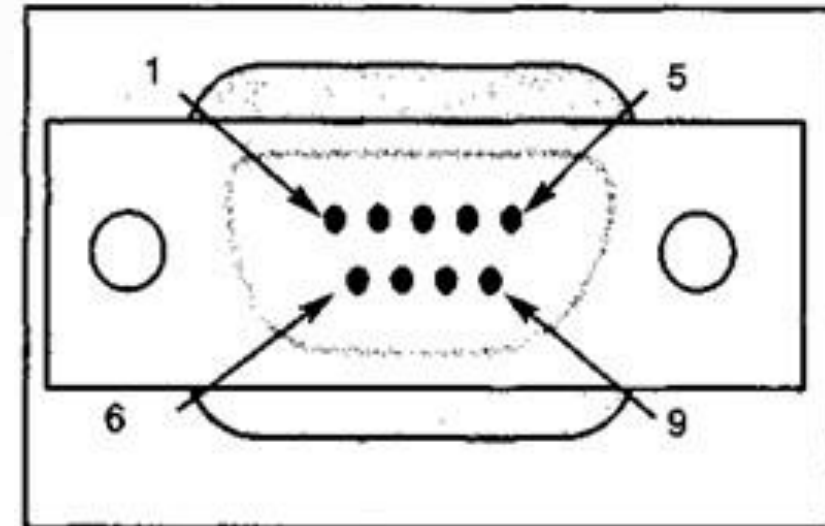
RS232



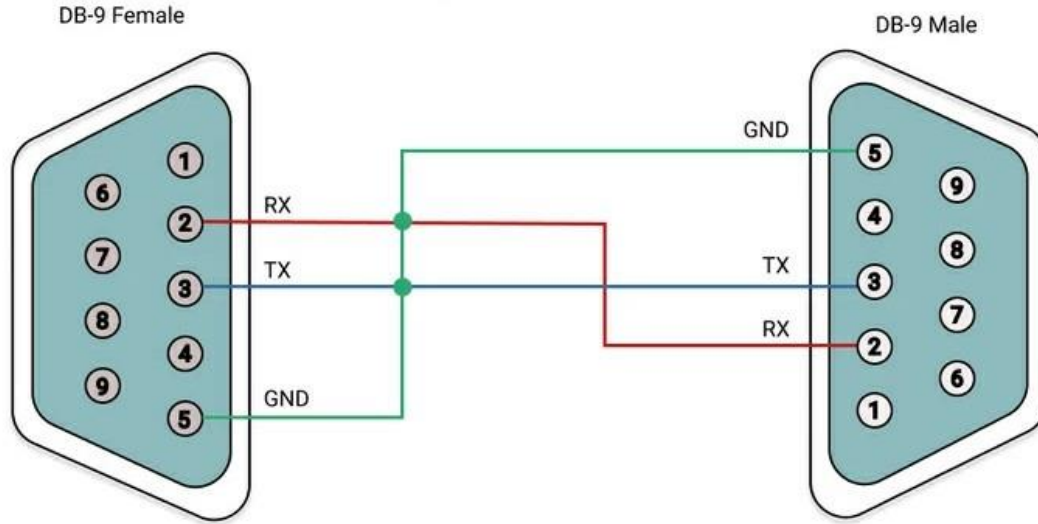


DB-9 Pin connection

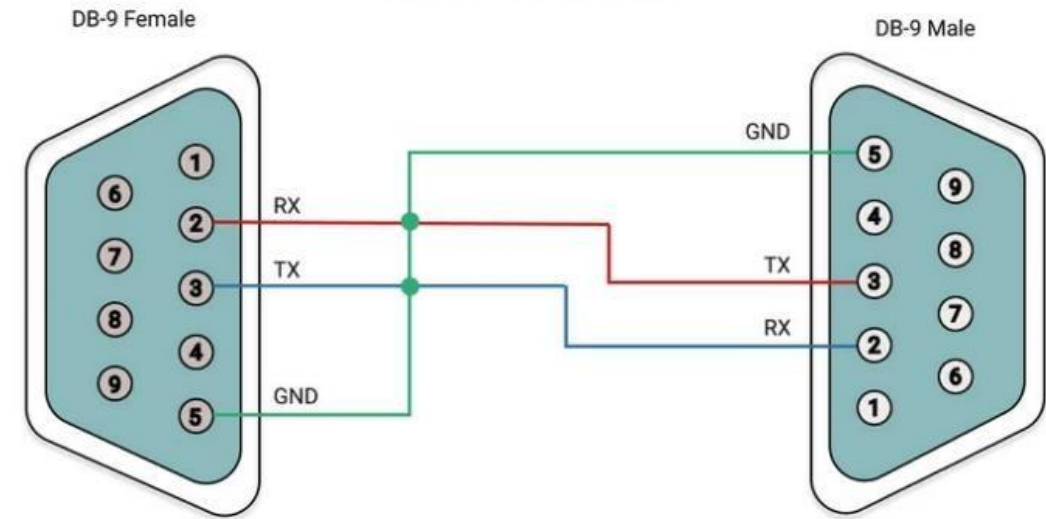
Pin	Description
1	Data carrier detect ($\overline{\text{DCD}}$)
2	Received data (Rx D)
3	Transmitted data (Tx D)
4	Data terminal ready (DTR)
5	Signal ground (GND)
6	Data set ready ($\overline{\text{DSR}}$)
7	Request to send (RTS)
8	Clear to send ($\overline{\text{CTS}}$)
9	Ring indicator (RI)

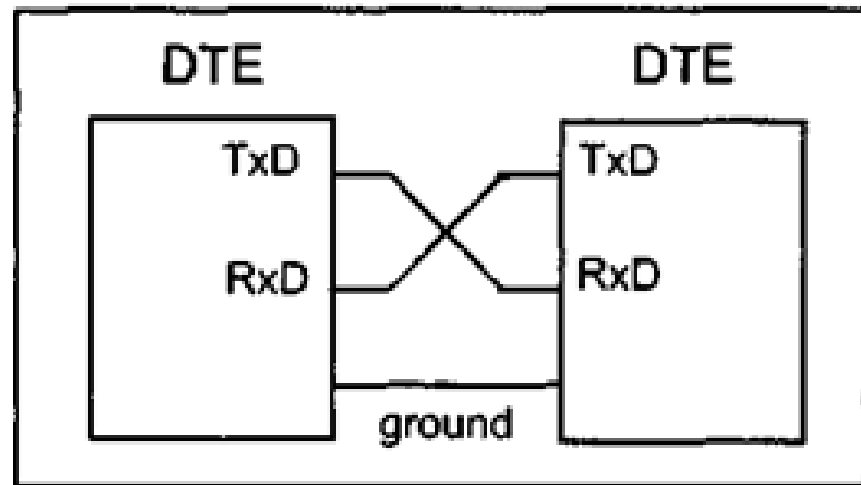
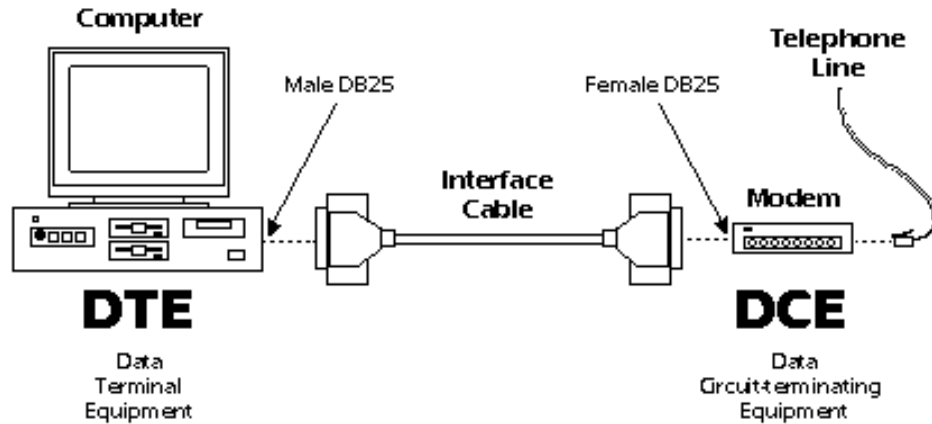


Straight cable connection



Null modem connection





Step 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.

Step 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.**

Step 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.

Step 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.

Step 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 modem and an input to the PC (DTE).

Step 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.

8051 connection to RS232

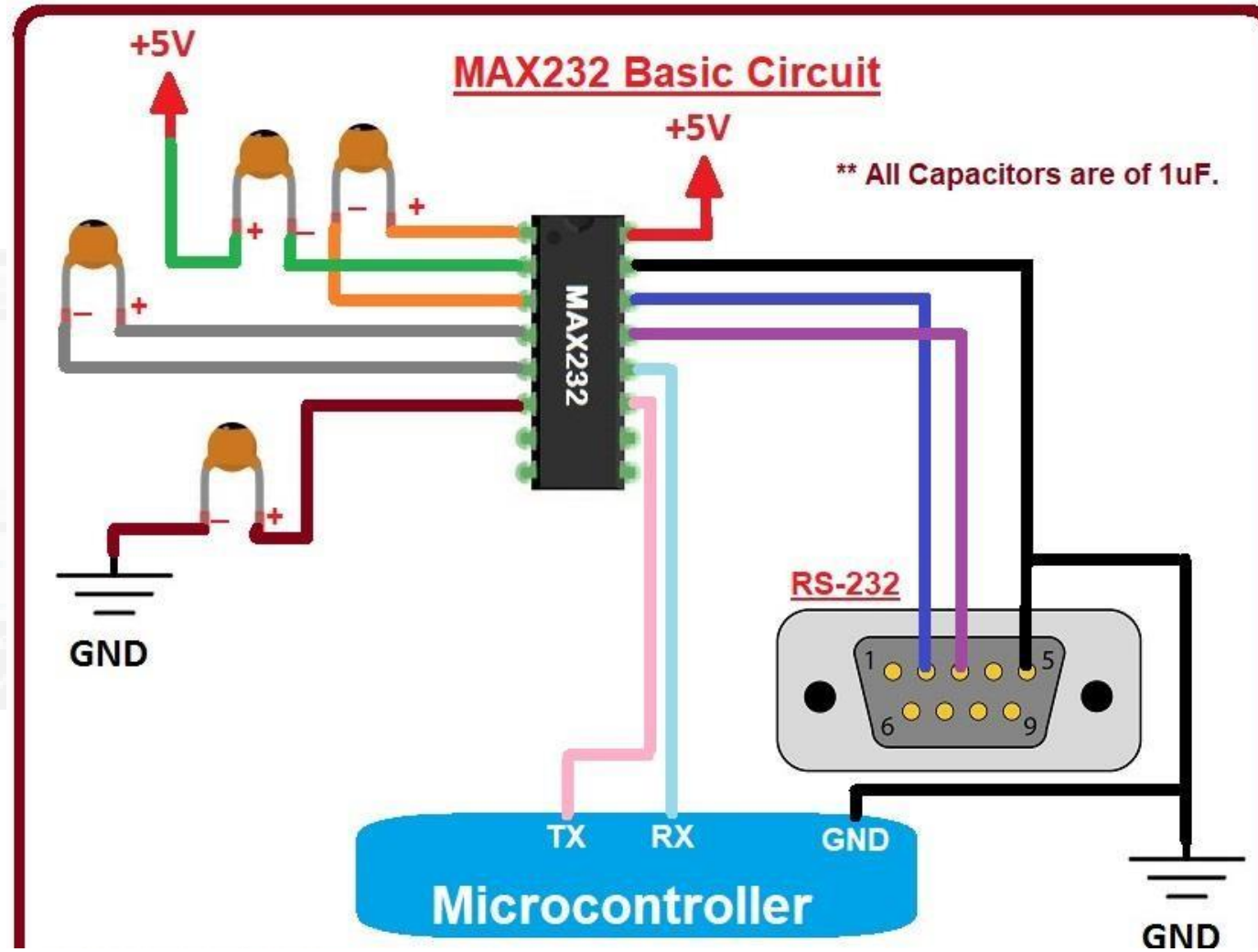
- **a) RxD and TxD pins in the 8051**
- 1. The 8051 has two pins that are used specifically for transferring and receiving data serially.
- 2. These two pins are called TxD and RxD and are part of the port 3 group (P3.0 and P3.1).
- Pin 11 of the 8051 (P3.1) is assigned to TxD and pin 10 (P3.0) is designated as RxD.
- These pins are TTL compatible; therefore, they require a line driver to make them RS232 compatible. One such line driver is the MAX232 chip.

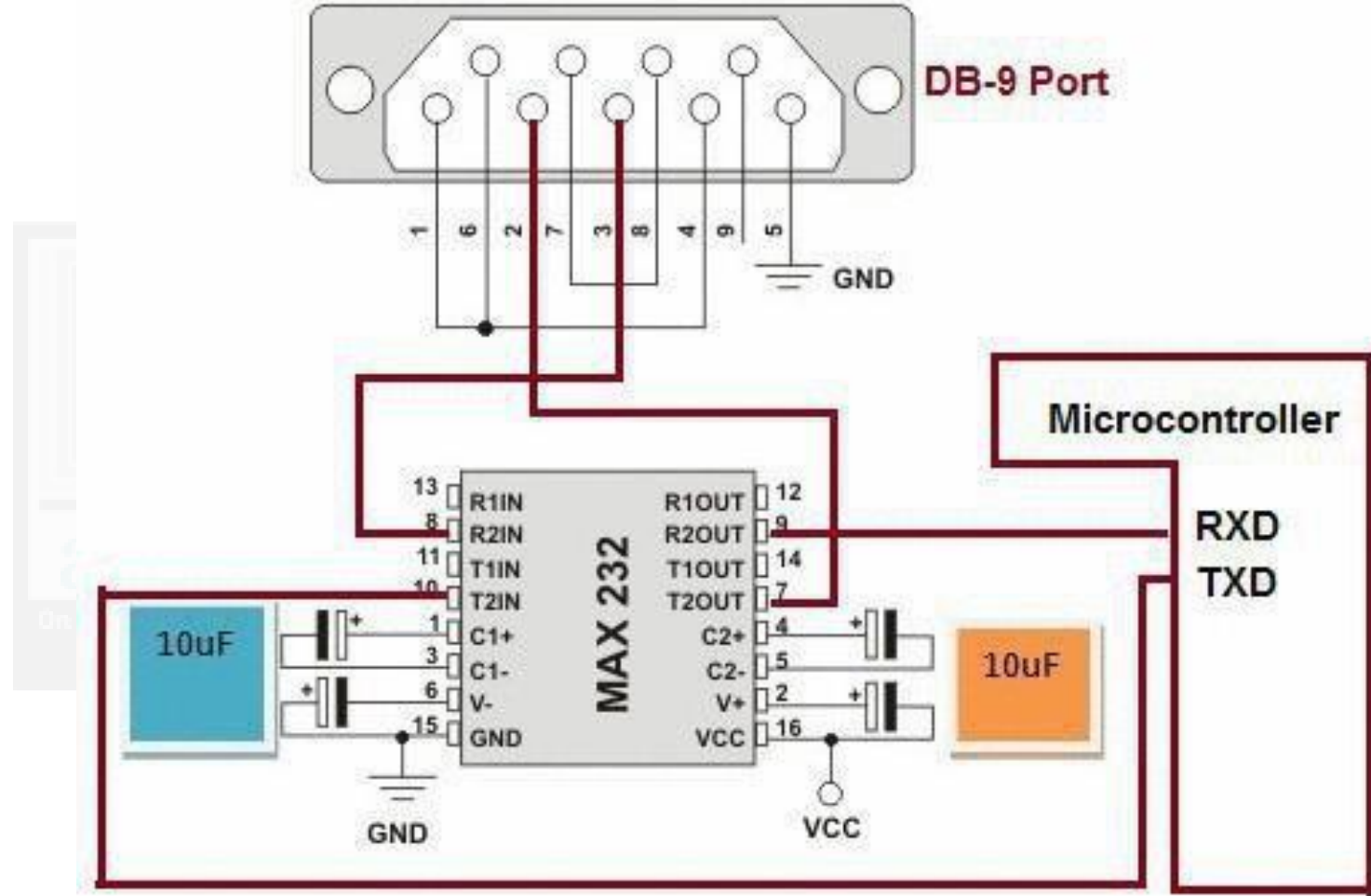
MAX232

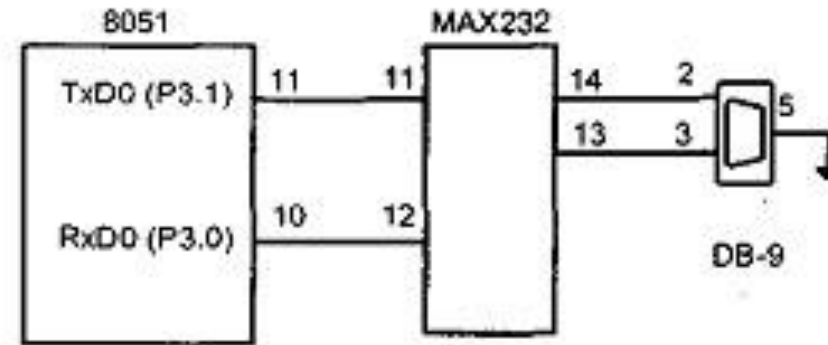
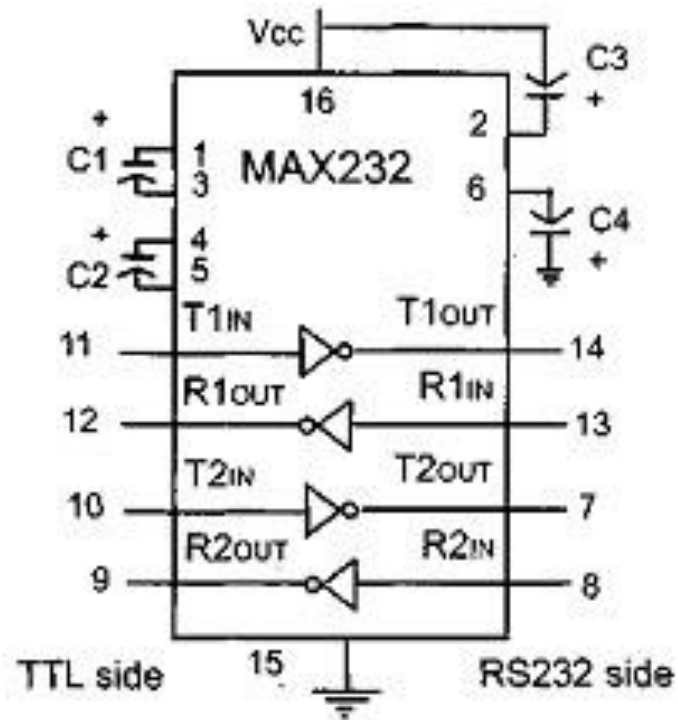
- Since the **RS232** is not compatible with today's microprocessors and microcontrollers, we need a line driver (voltage converter) to convert the RS232's signals to TTL voltage levels that will be acceptable to the 8051 's TxD and RxD pins.

MAX232

- 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.







(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
- 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 ($11.0592 \text{ MHz} / 12 = 921.6 \text{ kHz}$).
- The 8051 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.

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

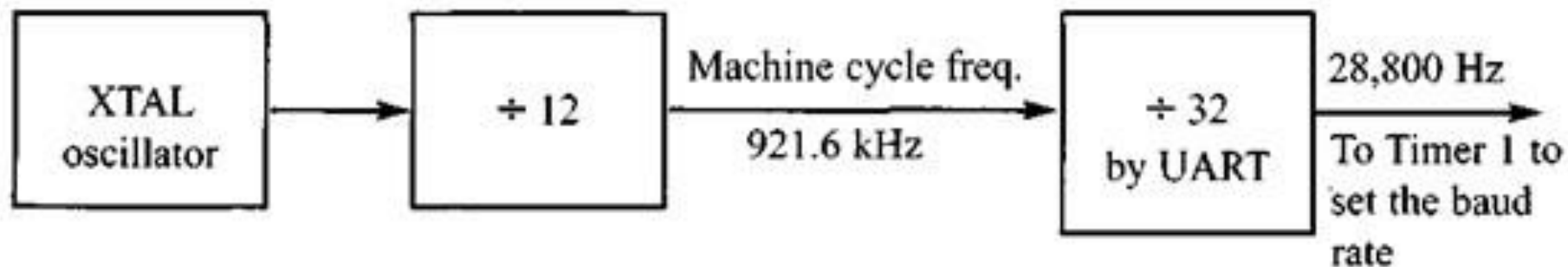
With XTAL = 11.0592 MHz, find the TH1 value needed to have the following baud rates.

(a) 9600 (b) 2400 (c) 1200

Solution:

With XTAL = 11.0592 MHz, we have:

The machine cycle frequency of the 8051 = $11.0592 \text{ MHz} / 12 = 921.6 \text{ kHz}$, and $921.6 \text{ kHz} / 32 = 28,800 \text{ Hz}$ is the frequency provided by UART to Timer 1 to set baud rate.



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.

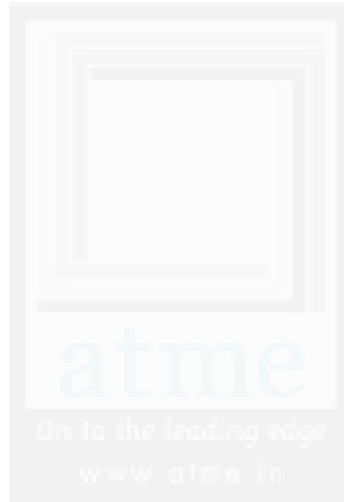
```
MOV SBUF, #'D'      ;load SBUF=44H, ASCII for 'D'
MOV SBUF, A          ;copy accumulator into SBUF
MOV A, SBUF          ;copy SBUF into accumulator
```



A T M E
College of Engineering



SCON register



A T M E
College of Engineering

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

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.

- **a) SM0, SM1**
- SM0 and SM1 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.

<i>SM0</i>	<i>SM1</i>	
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

- *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.**

REN — 1 or REN = 0 can
be achieved by the instructions **“SETB SCON. 4”** and **“CLR SCON. 4”**,
respectively

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**

- *f) TI*
- **TI (transmit interrupt)** is bit D1 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**

- *g) RI*

- RI (receive interrupt) is the D0 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.

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 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.

4.3.3 Programming the 8051 to transfer data serially

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**.
4. The TI flag bit is monitored with the use of the instruction **JNB TI, xx** to see if the character has been transferred completely.
5. To transfer the next character, Repeat

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
    
```

SM0	SM1		REN			TI	RI
0	1		1			0	0

- STEP 1: START TIMER (TIMER 1) SETB TR1
- STEP 2: **BACK:** MOV SBUF,# 'A'
- STEP 3: HERE: JNB TI, HERE
- STEP 4: CLR TI
- STEP 5: SJMP BACK



Write a program to transfer the message “YES” serially at 9600 baud, 8-bit data, 1 Start bit and 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
  
```

Program 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
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
```

8051 serial port programming in assembly

Importance of the TI 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**.
4. 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.
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. In other words, when the 8051 finishes
6. 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.

Program 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
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
  
```

0	1	0	0	0	0	0	1
---	---	---	---	---	---	---	---

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 **SBUE.**

- 3. The stop bit is received. When receiving the stop bit the 8051 makes **RI** $\equiv 1$, 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.

Doubling the baud rate in the 8051



```
MOV  A,PCON
SETB ACC.7
MOV  PCON,A
```

```
;place a copy of PCON in ACC
;make D7=1
;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.
- ***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. = **$11.0592 \text{ MHz} / 12 = 921.6 \text{ kHz}$**
- **And $921.6 \text{ kHz} / 32 = 28,800 \text{ Hz}$ 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. = $11.0592 \text{ MHz} / 12 = 921.6 \text{ kHz}$**
- **And $921.6 \text{ kHz} / 16 = 57,600 \text{ Hz}$ since SMOD = 1**
- This is the frequency used by Timer 1 to set the baud rate

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

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.

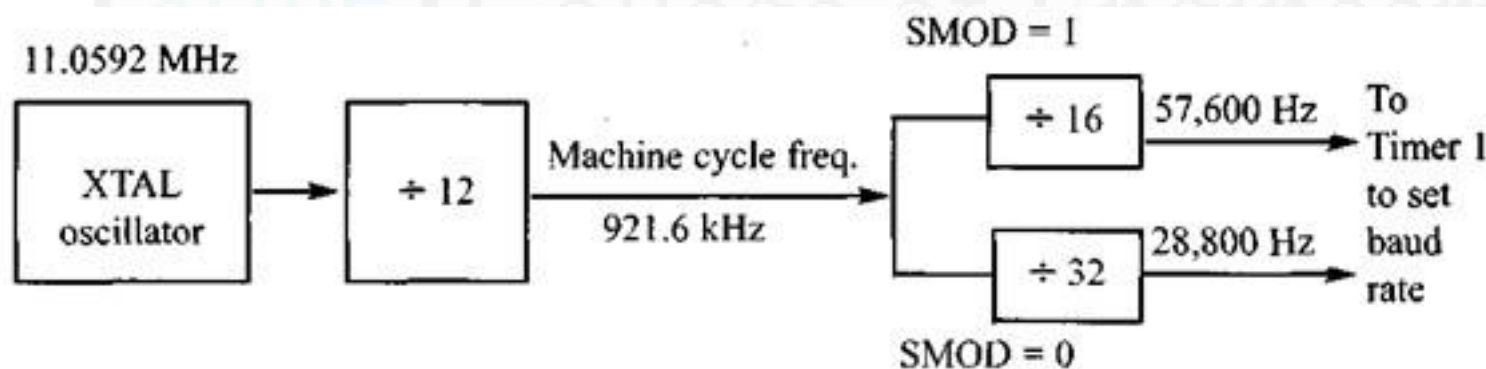
```
MOV    A,PCON        ;A = PCON
SETB   ACC.7          ;make D7 = 1
MOV    PCON,A         ;SMOD = 1, double baud rate
                        ;with same XTAL freq.
MOV    TMOD,#20H      ;Timer 1, mode 2(auto-reload)
MOV    TH1,-3         ;19200 (57,600 / 3 = 19200 baud rate
                        ;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:
 $11.0592 \text{ MHz} / 12 = 921.6 \text{ kHz}$ machine cycle frequency
 $921.6 \text{ kHz} / 16 = 57,600 \text{ Hz}$ frequency used by Timer 1 to set the baud rate
 $57,600 \text{ Hz} / 3 = 19,200$ 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.

- With XTAL = 11.0592 MHz and SMOD = 1, we have Timer 1 frequency = 57,600 Hz.
- $57,600 / 9600 = 6$; therefore, TH1 = -6 or TH1 = FAH.
- $57,600 / 4800 = 12$; therefore, TH1 = -12 or TH1 = F4H.



Serial port programming in 8051 C

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>
```

Header File

```
void main(void)
```

Main Function

```
{
```

```
    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==1;

```
        TI=0;
```

TI=0;

```
    }
```

```
}
```

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;
}
```

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
    SCCN=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;
    }
}
```

C Programs continued

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;
}
```



#include<reg51.h>

void SerTx(Unsigned char);

Void Main()

```
{  
    TMOD=0X20;  
    TH1=0XFD;  
    SCON=0X50;
```

TR1=1;

While(1)

```
{  
    SerTx('Y');  
    SerTx('E');  
    SerTx('S');  
}
```

Void SerTx (unsigned char x)

```
{  
    SBUF=x;  
    While(TI==0);  
    TI=0;  
}
```



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
    SCCN=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;
    }
}
```

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;
void main(void)
{
    unsigned char z;
    unsigned char fname[]="ALI";
    unsigned char lname[]="SMITH";
    TMOD=0x20;
    TH1=0xFD;
    SCON=0x50;
    TR1=1;
    if (MYSW==0)
    {
        for(z=0;z<3;z++)
        {
            SBUF=fname[z];
            while(TI==0);
            TI=0;
        }
    }
}
```

//input switch

//use Timer 1, 8-BIT auto-reload

//9600 baud rate

//start timer

//check switch

//write name

//place value in buffer

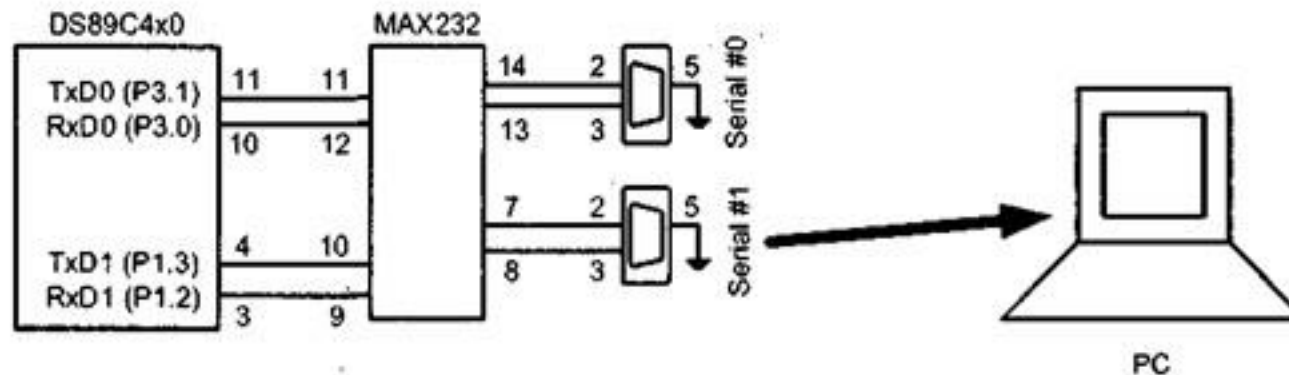
//wait for transmit

- For(z=0; z<3;z++) z++ ++z
- Fname[]= 'ALI'
- Step 1: Z=1 SBUF=Z
- Step 2: Z=2 SBUF=Z
- Step 3: Z=3 SBUF=Z

```
else                                     ←
{                                       ←
    for(z=0;z<5;z++)                  //write name ←
    {                                  ←
        SBUF=lname[z];                //place value in buffer ←
        while(TI==0);                 //wait for transmit ←
        TI=0;                          ←
    }                                  ←
}                                       ←
}                                     ←
```

8051 C compilers and the second serial port

- Since many C compilers do not support the second serial port of the DS89C4x0 chip, we have to declare the byte addresses of the new **SFR registers using the sfr keyword.**



- Write a C program for the **DS89C4x0** 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.

```
#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;
    }
}
```

- Program the DS89C4x0 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;
    }
}
```

8051 interrupts

- **Interrupts vs. 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 associated with the interrupt is called **the interrupt service routine (ISR) or interrupt handler.**



- 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 **served.**



- 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.**

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.**

Six interrupts in 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.

HERE: JNB **TF0**, HERE ; CLR TF0

SMO

SM1

SM2

REN

TB8

RB8

TI

RI

Enabling and disabling an interrupt

- Steps in enabling an interrupt
 1. To enable an interrupt, we take the following steps: . 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.

D7							D0
EA	--	ET2	ES	ET1	EX1	ET0	EX0

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.

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`

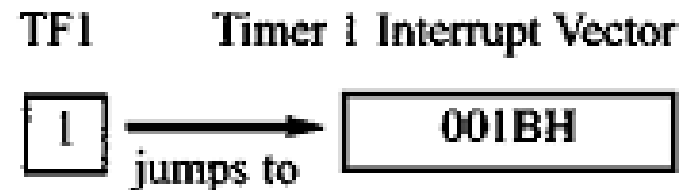
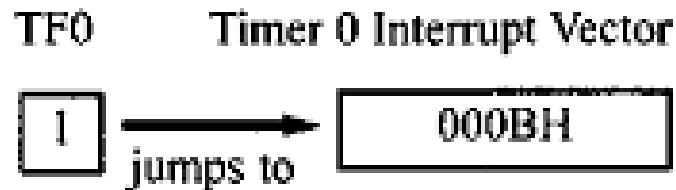
`SETB IE.4 ;enable serial interrupt`

`SETB IE.1 ;enable Timer 0 interrupt`

`SETB IE.2 ;enable EX1`

Programming timer

- **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.**



- 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.

A We will use Timer 0 in mode 2 (auto-reload). $TH0 = 100/1.085 \mu s = 92$.

Col

;--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 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



- The following program generates a square wave on pin P1.5 continuously using timer 1 for a time delay. Find the frequency of the square wave if XTAL = 11.0592 MHz. In your calculation do not include the overhead due to instructions in the loop

MOV TMOD, #10H

AGAIN: MOV TL1, #34H

MOV TH1, #76H

SETB TR1

BACK: JNB TF1, BACK

CLR TR1

CPL P1.5

CLR TF1

SJMP AGAIN

1. $(FFFF-7634+1) = 89CC \text{ (hex)} = 35276 \text{ (dec)}$

2. $35276 * 1.085 = 38.274\text{ms}$

3. $1/38.274\text{ms} = 26.127 \text{ Hz}$

TL1,#05H

TH1,# FDH

Assume that XTAL=11.0592MHz, write a program to generate a square wave of 2kHz frequency on pin P1.5

$$T=1/f=1/2\text{KHz}=500\mu\text{s}$$

$$500\mu\text{s}/2=250\mu\text{s}$$

$$250\mu\text{s}/1.085\mu\text{s}=230=n \quad 65536-230=65306=\text{FF1A (hex)}$$

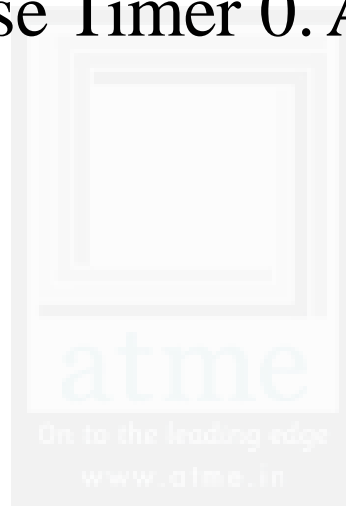
$$\text{TH}=\text{FF}, \text{TL}=1\text{A}$$



A T M E
College of Engineering



- Write a program to generate a square wave of 50 Hz frequency on pin P1 .2. Use Timer 0. Assume that XTAL = 11.0592MHz

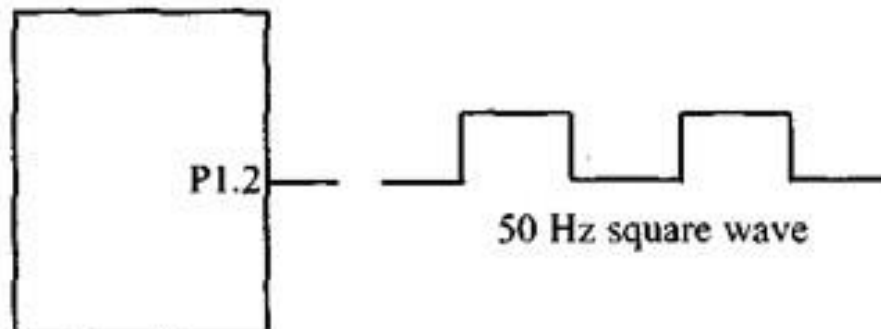


A T M E
College of Engineering

```

    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
  
```

8051



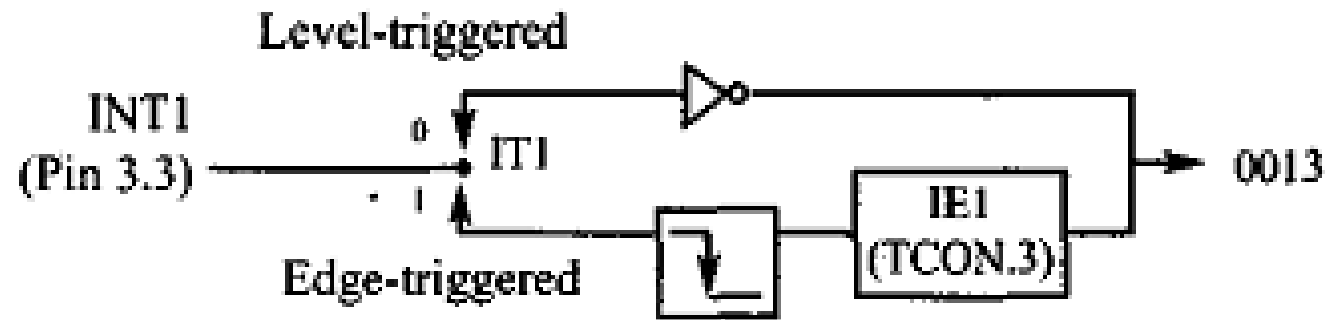
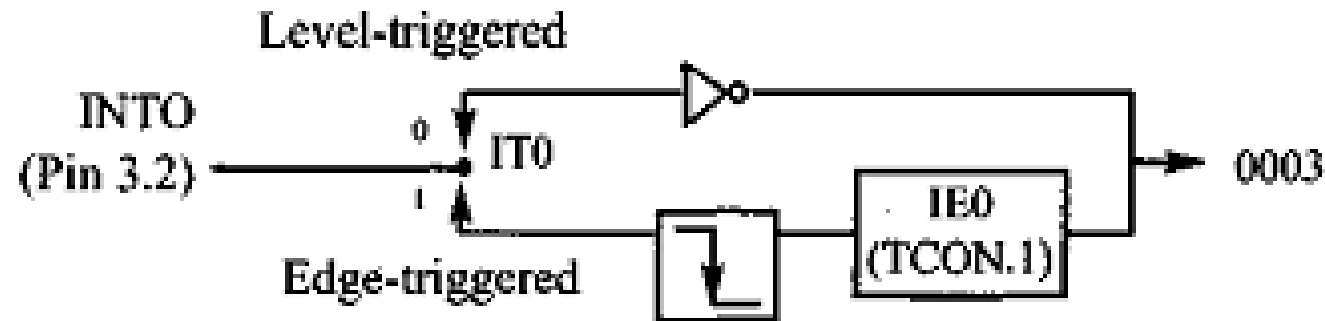
E
ring



External hardware interrupts

- The 8051 has two external hardware interrupts. Pins 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.

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.



A T M E
College of Engineering



- There are two types of activation for the external hardware interrupts:
- (1) level triggered, and
- (2) edge triggered



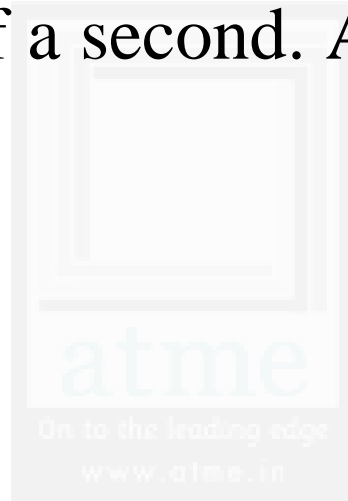
A T M E
College of Engineering

- **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.



- 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.

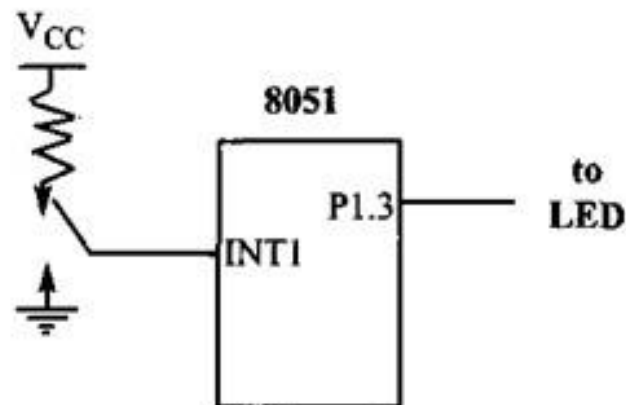


```

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

```

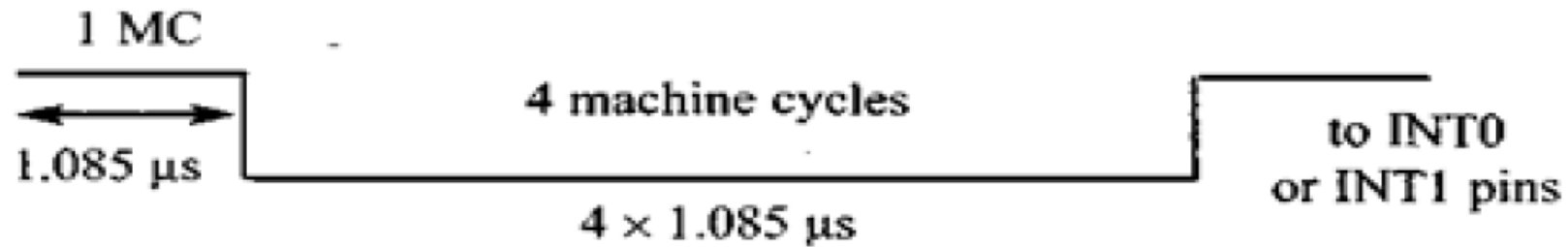
Pressing the switch will turn the LED on. If it is kept activated, the LED stays on.



Engineering

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 registers are enabled.**
- After the hardware interrupts in the IE register are enabled, the controller keeps sampling the INT \llcorner 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."



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

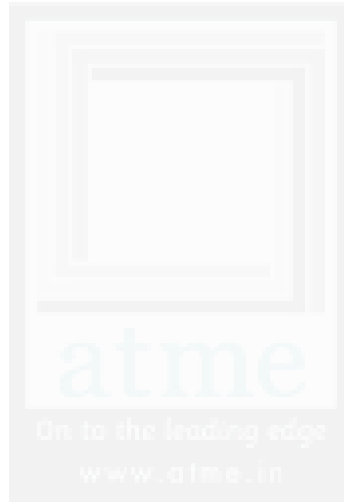
Minimum Duration of the Low Level-Triggered Interrupt (XTAL = 11.0592 MHz)

Write a program to (a) load the accumulator with the value 55H, and (b) complement the ACC 700 times.

```
MOV    A, #55H
MOV    R3, #10
NEXT:  MOV    R2, #70
AGAIN: CPL    A
        DJNZ  R2, AGAIN
        DJNZ  R3, NEXT
```



A T M E
College of Engineering



Thank You
A T M E
College of Engineering