COURSE NAME WITH CODE: MICROCONTROLLERS(EC45) **FACULTY NAME: ANURADA K K** **DATE:**	**UNIT-IV**

OBJECTIVES OF THE LECTURE:

- Basics of Interrupts
- Timer introduction
- Different modes of timer operations
- 8051 Serial Communication
- Basics of Serial Communication, RS 232 connections,
- Assembly and C programming on Timers and serial communication
- Applications

REFERENCES:

1. " The 8051 Microcontroller Architecture Programming & Applications", 2e Kenneth J. Ayala
2. " The 8051 Microcontroller and Embedded Systems-using Assembly and C", Muhammad Ali Mazidi, Janice GillespieMazidi and Rllin D. McKinalay; PHI, 2006/Pearson,2006

BASICS OF INTERRUPTS

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

Steps taken by processor while processing an interrupt:

1. It completes the execution of the current instruction.
2. PSW is pushed to stack.
3. PC content is pushed to stack.
4. Interrupt flag is reset.
5. PC is loaded with ISR address.
   ISR will always ends with RETI instruction. The execution of RETI instruction results in the following.

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

Classification of interrupts.

1. External and internal interrupts.
External interrupts are those initiated by peripheral devices through the external pins of the microcontroller.
Internal interrupts are those activated by the internal peripherals of the microcontroller like timers, serial controller etc.)

2. Maskable and non-maskable interrupts.
The category of interrupts which can be disabled by the processor using program is called maskable interrupts.
Non-maskable interrupts are those category by which the programmer cannot disable it using program.

3. Vectored and non-vectored interrupt.
Starting address of the ISR is called interrupt vector. In vectored interrupts the starting address is predefined. In non-vectored interrputs, the starting address is provided by the peripheral as follows.

- Microcontroller receives an interrupt request from external device.
- Controller sends an acknowledgement (**INTA)** after completing the execution of current instruction.
- The peripheral device sends the interrupt vector to the microcontroller.

8051 INTERRUPT STRUCTURE

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

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

8051 makes use of two registers to deal with interrupts.

1. IE Register

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

IE : Interrupt Enable Register (Bit Addressable)

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

EA	-	-	ES	ET1	EX1	ET0	EX0

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

2 IP Register

IP Register.

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

IP : Interrupt Priority Register (Bit Addressable)

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

-	-	-	PS	PT1	PX1	PT0	PX0

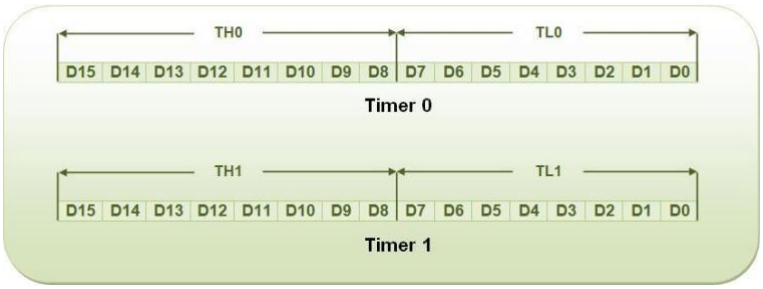
-	IP.7	Not implemented, reserved for future use*.
-	IP.6	Not implemented, reserved for future use*.
-	IP.5	Not implemented, reserved for future use*.
PS	IP.4	Defines the Serial Port interrupt priority level.
PT1	IP.3	Defines the Timer 1 interrupt priority level.
PX1	IP.2	Defines External Interrupt priority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

TIMERS AND COUNTERS

Timers/Counters are used generally for

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

8051 has two timers, Timer 0 and timer 1.



Timer in 8051 is used as timer, counter and baud rate generator. Timer always counts up irrespective of whether it is used as timer, counter, or baud rate generator: Timer is always

incremented by the microcontroller. The time taken to count one digit up is based on master clock frequency.

If Master CLK=12 MHz,
Timer Clock frequency = Master CLK/12 = 1 MHz
Timer Clock Period = 1micro second
This indicates that one increment in count will take 1 micro second.

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

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

TMOD Register

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

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

GATE When TRx (in TCON) is set and GATE = 1, TIMER/COUNTERx will run only while INTx pin is high (hardware control). When GATE = 0, TIMER/COUNTERx will run only while TRx = 1 (software control).

C/$\overline{T}$ Timer or Counter selector. Cleared for Timer operation (input from internal system clock). Set for Counter operation (input from Tx input pin).

M1 Mode selector bit (NOTE 1).

M0 Mode selector bit (NOTE 1).

Note 1 :

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

TCON REGISTER

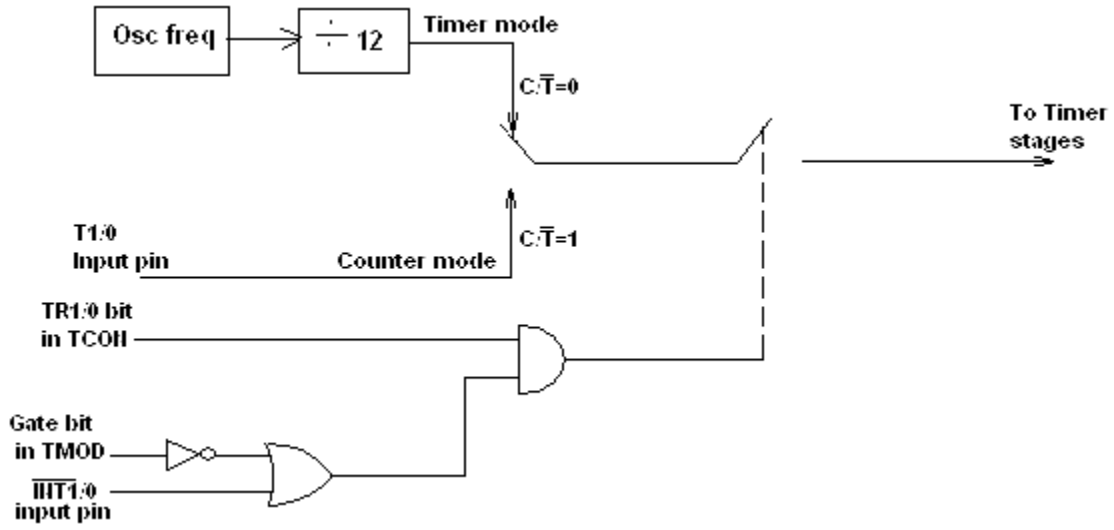
TCON Register

TCON : Timer/Counter Control Register (Bit Addressable)

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

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

Timer/Conter Control Logic



TIMER MODES

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

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

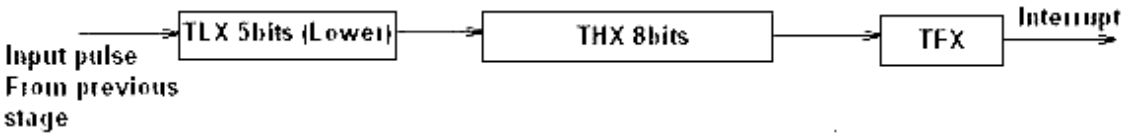


Fig: Operation of Timer on Mode-0

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

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

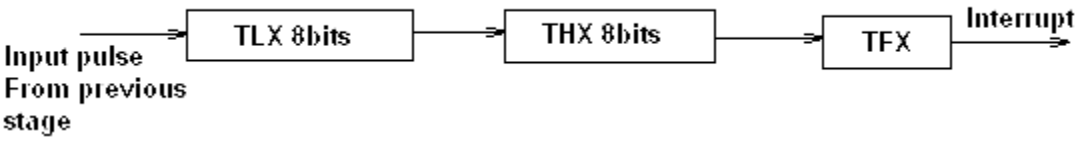


Fig: Operation of Timer in Mode 1

Timer Mode2 (Auto reload Mode): This is a 8 bit counter/timer operation. Counting is performed in TLX while THX store a constant value. In this mode when the timer overflow i,e TLX becomes FFH, it is fed with the value stored in THX. For example if load THX with 50H then the timer in mode 2 will count from 50H to FFH. After that 50H is again reloaded. This mode is useful in applications like fixed time sampling.

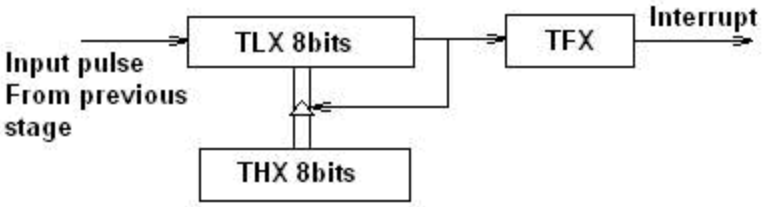


Fig: Operation of Timer in Mode 2

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

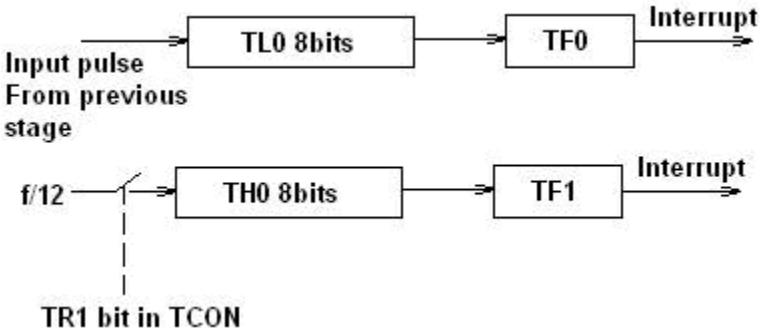


Fig: Operation of Timer in Mode3

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

PROGRAMMING 8051 TIMERS IN ASSEMBLY

In order to program 8051 timers, it is important to know the calculation of initial count value to be stored in the timer register. The calculations are as follows.

In any mode, Timer Clock period = 1/Timer Clock Frequency.

= 1/(Master Clock Frequency/12)

a. Mode 1 (16 bit timer/counter)

Value to be loaded in decimal = 65536 – (Delay Required/Timer clock period) Convert the answer into hexadecimal and load onto THx and TLx register. *(65536D = FFFFH+1)*

b. Mode 0 (13 bit timer/counter)

Value to be loaded in decimal = 8192 – (Delay Required/Timer clock period) Convert the answer into hexadecimal and load onto THx and TLx register. *(8192D = 1FFFH+1)*

c. Mode 2 (8 bit auto reload)

Value to be loaded in decimal = 256 – (Delay Required/Timer clock period)

Convert the answer into hexadecimal and load onto THx register. Upon starting the timer this value from THx will be reloaded to TLx register.

(256D = FFH+1)

Steps for programming timers in 8051

Mode 1:

- Load the TMOD value register indicating which timer (0 or 1) is to be used and which timer mode is selected.
- Load registers TL and TH with initial count values.
- Start the timer by the instruction "SETB TR0" for timer 0 and "SETB TR1" for timer 1.
- Keep monitoring the timer flag (TF) with the "JNB TFx,target" instruction to see if it is raised. Get out of the loop when TF becomes high.
- Stop the timer with the instructions "CLR TR0" or "CLR TR1", for timer 0 and timer 1, respectively.
- Clear the TF flag for the next round with the instruction "CLR TF0" or "CLR TF1", for timer 0 and timer 1, respectively.
- Go back to step 2 to load TH and TL again.

Mode 0:

The programming techniques mentioned here are also applicable to counter/timer mode 0. The only difference is in the number of bits of the initialization value.

Mode 2:

- Load the TMOD value register indicating which timer (0 or 1) is to be used; select timer mode 2.
- Load TH register with the initial count value. As it is an 8-bit timer, the valid range is from 00 to FFH.
- Start the timer.
- Keep monitoring the timer flag (TFx) with the "JNB TFx,target" instruction to see if it Is raised. Get out of the loop when TFx goes high.
- Clear the TFx flag.
- Go back to step 4, since mode 2 is auto-reload.

1. Write a program to continuously generate a square wave of 2 kHz frequency on pin P1.5 using timer 1. Assume the crystal oscillator frequency to be 12 MHz.

The period of the square wave is T = 1/(2 kHz) = 500 µs. Each half pulse = 250 µs. The value n for 250 µs is: 250 µs /1µ s = 250
65536 - 250 = FF06H.
TL = 06H and TH = 0FFH.

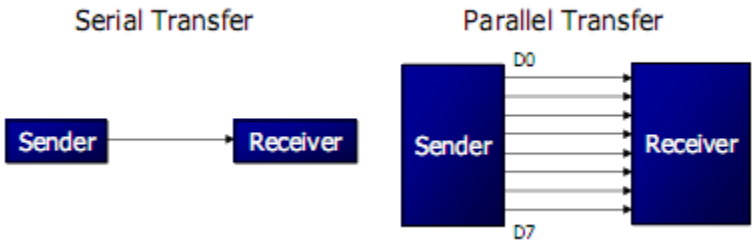
```
        MOV   TMOD,#10    ;Timer 1, mode 1
AGAIN:  MOV   TL1,#06H    ;TL0 = 06H
        MOV   TH1,#0FFH   ;TH0 = FFH
        SETB  TR1         ;Start timer 1
BACK:   JNB   TF1,BACK    ;Stay until timer rolls over
        CLR   TR1         ;Stop timer 1
        CPL   P1.5        ;Complement P1.5 to get Hi, Lo
        CLR   TF1         ;Clear timer flag 1
        SJMP  AGAIN       ;Reload timer
```

Serial communication

Basically there are two ways of communication

Parallel Communication: The data is transmitted over 8 or more lines (wire conductors) which are only a few feet away.

Serial Communication: In serial communication data is sent one bit at a time over single electrical wire.



Communication Links:

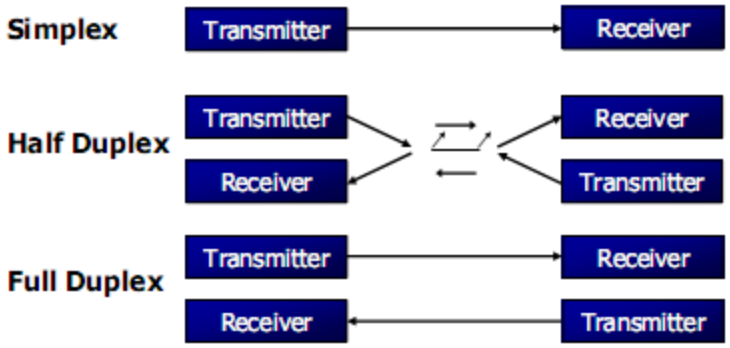
Simplex: This type of link is dedicated only for transmission and data cannot be received through the link.
Half duplex: In half duplex, the communication link can be established for both transmission as well as reception; however the data transfer will be in only one direction at any given time.
Full duplex:

Serial data communication uses two methods

- Synchronous serial data communication
- Asynchronous serial data communication

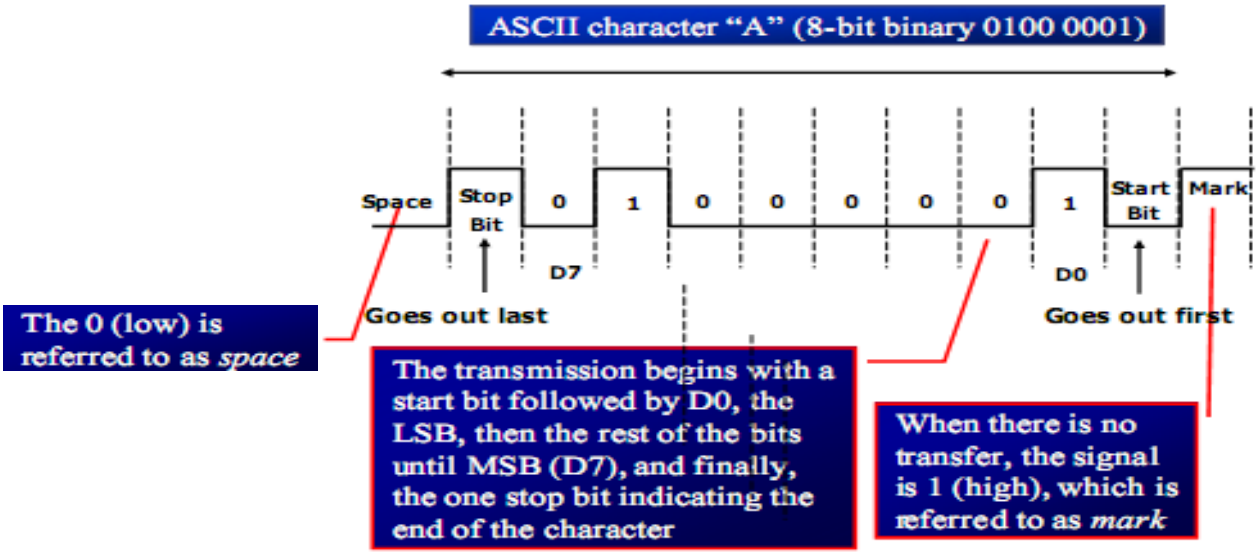
There are special IC chips made by many manufacturers for serial communications

- UART (universal asynchronous Receiver-transmitter)
- USART (universal synchronous-asynchronous Receiver-transmitter)
- If data can be transmitted and received, it is a duplex transmission
- If data transmitted one way a time, it is referred to as half duplex
- If data can go both ways at a time, it is full duplex
- this is contrast to simplex transmission



ASYNCHRONOUS SERIAL COMMUNICATION AND DATA FRAMING

- A protocol is a set of rules agreed by both the sender and receiver on
- How the data is packed
- How many bits constitute a character
- When the data begins and ends
- Asynchronous serial data communication is widely used for character-oriented transmissions
- Each character is placed in between start and stop bits, this is called framing
- Block-oriented data transfers use the synchronous method
- 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)



- Due to the extended ASCII characters, 8-bit ASCII data is common
- In older systems, ASCII characters were 7-bit
- In modern PCs the use of one stop bit is standard
- In 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.
- 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
- This gives 25% overhead, i.e. each 8-bit character with an extra 2 bits
- In some systems in order to maintain data integrity, the parity bit of the character byte is included in the data frame
- UART chips allow programming of the parity bit for odd-, even-, and no-parity options
- 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
- It is modem terminology and is defined as the number of signal changes per second
- In modems, there are occasions when a single change of signal transfers several bits of data
  - As far as the conductor wire is concerned, the baud rate and bps are the same, and we use the terms interchangeably

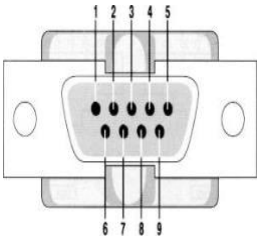
The data transfer rate of given computer system depends on communication ports incorporated into that system

- IBM PC/XT could transfer data at the rate of 100 to 9600 bps
- Pentium-based PCs transfer data at rates as high as 56K bps
- In asynchronous serial data communication, the baud rate is limited to 100K bps

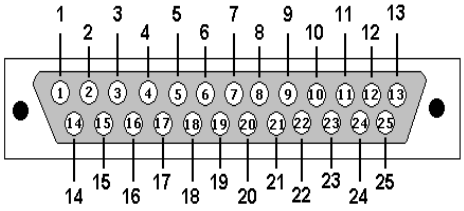
CONNECTIONS TO RS-232

RS-232 standards:

To allow compatibility among data communication equipment made by various manufactures, an interfacing standard called RS232 was set by the Electronics Industries Association (EIA) in 1960. Since the standard was set long before the advent of logic family, its input and output voltage levels are not TTL compatible. In RS232, a logic one (1) is represented by -3 to -25V and referred as MARK while logic zero (0) is represented by +3 to +25V and referred as SPACE. For this reason to connect any RS232 to a microcontroller system we must use voltage converters such as MAX232 to convert the TTL logic level to RS232 voltage levels and vice-versa. MAX232 IC chips are commonly referred as line drivers. In RS232 standard we use two types of connectors. DB9 connector or DB25 connector.



DB9 Male Connector



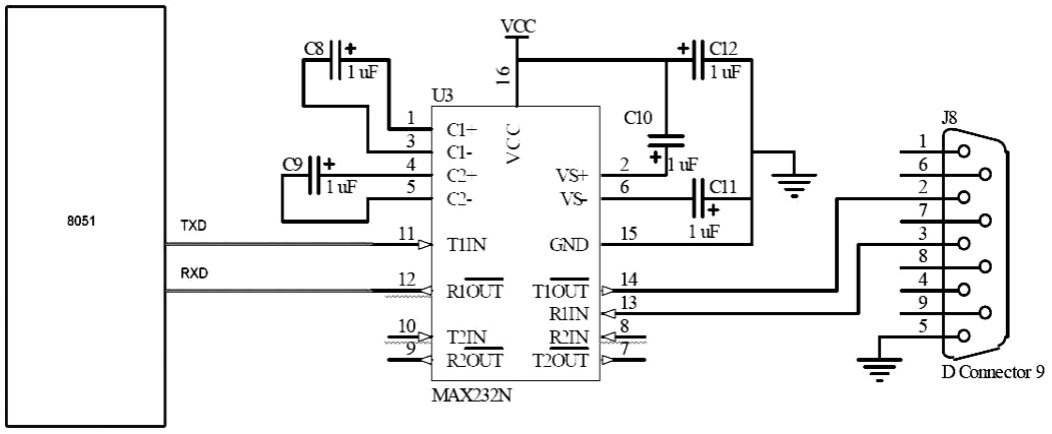
DB25 Male Connector

The pin description of DB9 and DB25 Connectors are as follows:

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

The 8051 connection to MAX232 is as follows.

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



SERIAL COMMUNICATION PROGRAMMING IN ASSEMBLY AND C.

Steps to programming the 8051 to transfer data serially

1. The TMOD register is loaded with the value 20H, indicating the use of the Timer 1 imode 2 (8-bit auto reload) to set the baud rate.
2. The TH1 is loaded with one of the values in table 5.1 to set the baud rate for serialdata transfer.
3. The SCON register is loaded with the value 50H, indicating serial mode 1, where an8-bit data is framed with start and stop bits.
4. TR1 is set to 1 start timer 1.
5. TI is cleared by the "CLR TI" instruction.
6. The character byte to be transferred serially is written into the SBUF register.
7. The TI flag bit is monitored with the use of the instruction JNB TI, target to see if the character has been transferred completely.
8. To transfer the next character, go to step 5.

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

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

Example 2. Write a program for the 8051 to transfer the message 'DSCE1' serially at 9600 baud, 8 bit data, 1 stop bit continuously.

```
ORG 0000H
LJMP START
ORG 0030H
START: MOV TMOD, #20H   ; select timer 1 mode 2
MOV TH1, #0FDH    ; load count to get reqd. baud rate of 9600
MOV SCON, #50H    ; initialise uart in mode 2
                  ; 8 bit data and 1 stop bit
SETB TR1          ; start timer
LOOP: MOV A, #'D'   ; load 1st letter 'D' in a
ACALL LOAD     ; call load subroutine
MOV A, #'S'    ; load 2nd letter 'S' in a
ACALL LOAD     ; call load subroutine
MOV A, #'C'    ; load 3rd letter 'C' in a
ACALL LOAD     ; call load subroutine
MOV A, #'E'    ; load 4th letter 'E' in a
ACALL LOAD     ; call load subroutine
MOV A, #'1'    ; load 4th letter '1' in a
ACALL LOAD    ; call load subroutine
SJMP LOOP     ; repeat steps

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

END
CLR TI   ; Clear transmit interrupt flag
RET
```

b) Write an ALP to implement (display) an eight bit up/down BCD counters by using timer delay.

Algorithm:

1. Set up timer0 in mode 2 operation
2. Load TH1 with 118 to generate an interrupt every 0.05msec.
3. Reset registers a, r1 & r0.
4. Repeat step 4 continuously
5. On interrupt; ISR at 000B location goes to step 6
6. Disable timer0
7. Update r1 & r0
8. Check if 20000 interrupts (=1 sec) over. Yes –increment accumulator a.
9. Enable timer & return from ISR.

Program:

Label	Mnemonic/Operands	Comments
	ORG 0000H	
	SJMP 30H	
	ORG 0BH	
	SJMP ISR	
	ORG 30H	
	MOV A, #00	
	MOV R0,#00	
	MOV R1,#00	
	MOV TMOD, #02H	
	MOV TH0, #118	
	MOV IE, #82H	
	SETB TCON.4	
HERE:	SJMP HERE	
ISR:	CLR TCON.4	
	INC R1	

	CJNE R1,#100,SKIP
	MOV R1,#00
	INC R0
	CJNE R0,#200,SKIP
	MOV R0,#00H
	INC A	

SKIP:	SETB TCON.4 RETI END	

RESULT **:** Accumulator A is incremented in hex from 00, 01,02…09,0A, 0B, …, 0F, 10, 11, …FF every 1 second (for 33MHz clock setting & every 3 seconds for 11.0592MHz)

TMOD FUNCTION REGISTER

7	6	5	4	3	2	1	0
Gate	C/$\overline{T}$	M1	M0	Gate	C/ $\overline{T}$	M1	M0
[TIMER1				][	TIMER0		]

Bit	Symbol	Function
7/3	Gate	OR gate enable bit which controls RUN/STOP of timer 1/0
6/2	C/	Set to 1–by program to make timer 1/0 act as a counter by counting pulses from external input pins 3.5(T1) or 3.4(T0)
5/1	M1	Timer/Counter operating mode select bit 1
4/0	M0	Timer/Counter operating mode select bit 0

M1	M0	MODE
0	0	0-13 bit timer
0	1	1-16 bit timer
1	0	2-8-bit reloadable timer
1	1	3-timer0-2-8bit timers: timer1 Stop

TCON Function Register

7	6	5	4	3	2	1	0
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

Interrupt	Address(Hex)
IE0	0003
TF0	000B
IE1	0013
TF1	001B
Serial	0023

To get 1sec delay

1/0.05msec = 200*100 in the ISR
(Assuming 33 MHz crystal frequency.
For 11 MHz, the calculations change).
Timer delay = 12 * (257-delay)/frequency
Timer delay=0.05 msec
Delay = 256-((timer delay * frequency)/12)
= 256-(0.05*10 $^{-3}$ * 33*10^{6})/12
= 256-137.5
= 118.5 //loaded in TH0

Q)Write a program illustrating serial ASCII data transmission (data-BHARAT). Conduct an experiment to configure 8051 microcontroller to transmit characters (BHARAT) to a PC using the serial port and display on the serial window.

Algorithm:

1. Initialize timer 1 to operate in mode 2 by loading TMOD register.
2. load TH1 with -3 to obtain 9600 baud.
3. Initialize the asynchronous serial communication transmission (SCON) register.
4. Start timer1 to generate the baud rate clock.
5. Transmit the characters "BHARAT" by writing into the SBUF register and waiting for the TI flag.

Program:

Label	Mnemonic/Operands	Comments
	ORG 0000H	
	SJMP 30H	
	ORG 30H	
	MOV R0,#05H	
	MOV DPTR, #300H	
	MOV TMOD,#20H	
	MOV TH1, #-3	//-3=FD loaded into TH1 for 9600 baud, 11.0592MHz.
	MOV SCON, #50H	
	SETB TR1	
AGAIN:	CLR A	
	MOVC A, @A+DPTR	
	JZ BACK	
	ACALL TRANS	
	INC DPTR	

	SJMP AGAIN	
	SJMP BACK	
BACK:	MOV SBUF, A	
TRANS:	JNB TI, HERE	
HERE:		
	CLR TI	
	RET	
	ORG 300H	
MYDATA:	DB “BHARAT”,0	
	END	

Note: To use result of this program, after selecting DEBUG session in the main menu use **View-> serial window #1**. On running & halting the program, the data is seen in the serial window.

RESULT: “BHARAT” is printed on the serial window each time the program is executed.

Theory: In serial transmission as opposed to parallel transmission, one bit at a time is transmitted. In serial asynchronous transmission, the data consists of a Start bit (high), followed by 8 bits of data to be transmitted and finally the stop bit. The byte character to be transmitted is written into the SBUF register. It transmits the start bit. The 8-bit character is transferred one bit at a time. The stop bit is transferred. After the transmission, the TI flag = 1 indicating the completion of transmission. Hence in the subroutine wait until TI is set. Later clear the TI flag and continue with transmission of the next byte by writing into the SBUF register. (The program can also be written in interrupt mode). The speed of the serial transmission is set by the baud rate which is done with the help of timer 1. (Refer Ayala). Timer1 must be programmed in mode 2 (that is, 8-bit, autos reload).

Baud rate Calculation:

Baud Rate = Crystal freq/ (12*32)

= (11.0592MHz)/(12*32)

= 28800

To get 9600, 28800/3 is obtained by loading timer1 with -3 (i.e., FF – 3 = FD) for further clock division. For 2400 baud rate, 28800/12 => -12 = F4 in TH1.

Q) Write an 8051 C program to send values 00 – FF to port P1.

Solution:

```
#include void main(void)
{ unsigned char z;
for (z=0;z<=255;z++) P1=z;
 }
```

Q) Write an 8051 C program to send hex values for ASCII characters of 0, 1, 2, 3, 4, 5, A, B, C, and D to port P1.

Solution:

```
#include void main (void)
{ unsigned char mynum[]="012345ABCD";
unsigned char z;
for (z=0;z<=10;z++) P1=mynum[z];
}
```

Q) Write an 8051 C program to toggle all the bits of P1 continuously.

Solution: //Toggle P1 forever

```
#include void main(void)
```

```
{
   for (;;)
        {
       p1=0x55;
       p1=0xAA;
        }
}
```