

Module-3.

8051 Stack, I/O Port Interfacing and Programming:

Why Program 8051 in C?

- Compilers produce hex files that is downloaded to ROM of microcontroller. The size of hex file is the main concern
 - microcontrollers have limited on-chip ROM
 - Code Space for 8051 is limited on 64k bytes.
- C programming is less time consuming, but has larger hex file size.
- The reasons for writing programs in C
 - It is easier and less time consuming to write in C than Assembly.
 - C is easier to modify and update.
 - You can use code available in function libraries.
 - C code is portable to other microcontroller with little or no modification.

Data Types

- A good understanding of C data types for 8051 can help programmers to create smaller hex files.
 - Unsigned char
 - Signed char
 - Unsigned int
 - Signed int
 - Sbit (single bit)
 - Bit and Sfr

UNSIGNED CHAR

- The character data type is the most natural choice.

• 8051 is an 8-bit microcontroller

- Unsigned char is an 8-bit data type in the range of 0-255 (0-FF₁₆)

• One of the most widely used data types for the 8051 Counter value & ASCII characters.

- C compilers use the signed char as the default if we do not put the keyword unsigned

① Write an 8051 C program to send values 00-FF to port P1.

```
#include <reg51.h>
void main(void)
{
    unsigned char z;
    for (z=0; z<=255; z++)
        P1=z;
}
```

② 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.

```
#include <reg51.h>
void main(void)
{
    unsigned char mynum[] = "012345ABCD";
    unsigned char z;
    for (z=0; z<=10; z++)
        P1=mynum[z];
}
```


③ write an 8051 C program to toggle all the bits of P1 continuously.

```
// Toggle P1 forever
```

```
#include <reg51.h>
```

```
void main (void)
```

```
{ for ( ; ; )
```

```
{
```

```
    P1 = 0x55;
```

```
    P1 = 0xAA;
```

```
}
```

```
}
```

SIGNED CHAR

- The signed char is an 8-bit data type

- Use the MSB D7 to represent - or +
- give us values from -128 to +127

- we should stick with the unsigned char unless the data needs to be represented as signed numbers.

① write an 8051 C program to send values of -4 to +4 to port P1.

```
// Signed numbers
```

```
#include <reg51.h>
```

```
void main (void)
```

```
{
```

```
    char mynum[] = {+1, -1, +2, -2, +3, -3, +4, -4};
```

```
    unsigned char z;
```

```
    for (z = 0; z <= 8; z++)
```

```
        P1 = mynum[z];
```

```
}
```


UNSIGNED & SIGNED INT

- The unsigned int is a 16 bit data type.

- Takes a value in the range of 0 to 65535 (0000 - FFFF H).
- Define 16-bit variables such as memory addresses.
- Set counter values of more than 256.
- Since registers and memory accesses are in 8-bit chunks, the misuse of int variables will result in a larger hex file.

- Signed int is a 16 bit data type

- Use the MSB bit to represent - or +
- We have 15 bits for the magnitude of the number from - 32768 to + 32767

SBIT → SINGLE BIT

① Write an 8051 C program to toggle bit D0 of the port P1 (P1.0) 50,000 times.

```
#include <reg51.h>
```

```
sbit MYBIT = P1^0;
```

```
void main(void)
```

```
{
```

```
    unsigned int z;
```

```
    for (z = 0; z <= 50000; z++)
```

```
    {
```

```
        MYBIT = 0;
```

```
        MYBIT = 1;
```

```
    }
```

```
}
```


BIT & SFR

- The BIT data type allows access to single bits of bit addressable memory spaces 20-2FH (internal RAM)
- The SFR data type allows to access the byte-size SFR registers.

Data Type	Size in Bits	Data Range / Usage
unsigned char	8-bit	0 to 255
(Signed) char	8-bit	-128 to +127
unsigned int	16-bit	0 to 65535
(Signed) int	16-bit	-32768 to +32767
sbit	1-bit	SFR bit addressable only
bit	1-bit	RAM bit addressable only
sfr	8-bit	RAM addresses 80-FFH only

TIME DELAY

- There are two ways to create a time delay in 8051 C.
 - using the 8051 timer
 - using a simple for loop
- There are three factors that can affect the accuracy of the delay. They are:
 - ① The 8051 design
 - The number of machine cycle.
 - The number of clock periods per machine cycle.
 - ② The crystal frequency connected to the X1-X2 input pins.
 - ③ Compiler choice.
 - C compiler converts the C statements & functions to assembly language instructions.
 - different compilers produce different code.

① Write an 8051 C program to toggle bits of P1 continuously forever with some delay.

// Toggle P1 forever with some delay in between
// "on" and "off"

```
#include <reg51.h>
```

```
void main(void)
```

```
{
```

```
    unsigned int X;
```

```
    for(;;)
```

```
        // repeat forever
```

```
    {
```

```
        P1 = 0x55;
```

```
        for(X=0; X<40000; X++); // delay size
```

```
        // unknown.
```

```
        P1 = 0xAA;
```

```
        for(X=0; X<40000; X++);
```

```
    }
```

```
}
```

② Write an 8051 C program to toggle bits of P1 ports continuously with a 250 ms.

```
#include <reg51.h>
```

```
void MSDelay(unsigned int);
```

```
void main(void)
```

```
{
```

```
    while(1)
```

```
        // repeat forever.
```

```
    {
```

```
        P1 = 0x55;
```

```
        MSDelay(250);
```

```
        P1 = 0xAA;
```

```
        MSDelay(250);
```

```
    }
```

```
}
```

```
void MSDelay(unsigned int itime)
```

```
{
    unsigned int i, j;
    for (i=0; i<itime; i++)
        for (j=0; j<1275; j++);
}
```

BYTE SIZE I/O PROGRAMMING

① LED's are connected to bits P1 and P2. Write an 8051 C program that shows the count from 0 to FFH (0000 0000 to 1111 1111) in binary on the LED's.

```
#include <reg51.h>
#define LED P2
void main(void)
{
    P1=00; //clear P1
    LED=0; //clear P2
    for (;;) //repeat forever
    {
        P1++; //increment P1
        LED++; //increment P2
    }
}
```

② Write an 8051 C program to get a byte of data from P1, wait 1/2 second and then send it to P2.

```
#include <reg51.h>
void MSDelay(unsigned int);
void main(void)
{
    unsigned char mybyte;
    P1=0xFF; //make P1 input port
    while (1)
    {
        mybyte = P1; //get a byte from P1
        MSDelay(500);
    }
}
```


P2 = mybyte ; // send it to P2

}

}

③ write an 8051 C program to get a byte of data from P0.

If it is less than 100, send it to P2.

#include <reg51.h>

void main(void)

{

unsigned char mybyte;

P0 = 0xFF;

// make P0 input port

while (1)

{

mybyte = P0; // get a byte from P0

if (mybyte < 100)

P1 = mybyte; // send it to P1

else

P2 = mybyte; // send it to P2

}

}

BIT ADDRESSABLE I/O PROGRAMMING

- ① Write an 8051 C program to toggle only bit P2.4 continuously without disturbing the rest of the bits of P2.

// Toggling an individual bit

```
#include <reg51.h>
```

```
sbit mybit = P2^4;
```

```
void main(void)
```

```
{
```

```
while(1)
```

```
{
```

```
mybit = 1;
```

```
mybit = 0;
```

```
}
```

```
}
```

- ② Write an 8051 C program to monitor bit P1.5. If it is high, send 55H to P0; otherwise, send AAH to P2.

```
#include <reg51.h>
```

```
sbit mybit = P1^5;
```

```
void main(void)
```

```
{
```

```
mybit = 1; // make mybit an input
```

```
while(1)
```

```
{
```

```
if(mybit == 1)
```

```
P0 = 0x55;
```

```
else
```

```
P2 = 0xAA;
```

```
}
```

```
}
```


③ A door sensor is connected to the P1.1 pin, and a buzzer is connected to P1.7. Write a 8051 C program to monitor the door sensor, and when it opens, sound the buzzer. You can sound the buzzer by sending a square wave of a few hundred Hz.

```
#include <reg51.h>
void MSDelay (unsigned int);
```

```
Sbit Dsensor = P1^1;
```

```
Sbit Buzzer = P1^7;
```

```
void main(void)
```

```
{
```

```
    Dsensor = 1;
```

```
    while(1)
```

```
    {
```

```
        while(Dsensor == 0) // while it opens
```

```
        {
```

```
            Buzzer = 0;
```

```
            MSDelay(200);
```

```
            Buzzer = 1;
```

```
            MSDelay(200);
```

```
        }
```

```
    }
```

```
}
```


I/O PROGRAMMING USING SFR REGISTERS.

① write an 8051 C program to toggle all the bits of P0, P1, & P2 continuously with a 250 ms delay. Use the sfr keyword to declare the port addresses.

// accessing ports as SFRs using sfr data type

```
sfr P0 = 0x80;  
sfr P1 = 0x90;  
sfr P2 = 0xA0;  
void Msdelay(unsigned int);  
void main(void)
```

```
{  
    while (1)  
    {  
        P0 = 0x55;  
        P1 = 0x55;  
        P2 = 0x55;  
        Msdelay(250);  
        P0 = 0xAA;  
        P1 = 0xAA;  
        P2 = 0xAA;  
        Msdelay(250);  
    }  
}
```

② write an 8051 C program to turn bit P1.5 on and off 50,000 times

```
sbit MYBIT = 0x95;  
void main(void)
```

```
{  
    unsigned int z;  
    for (z = 0; z < 50000; z++)  
    {  
        MYBIT = 1;  
        MYBIT = 0;  
    }  
}
```


3) Write an 8051 C program to get status of bit P1.0, save it and send it to P2.7 continuously.

```
#include <reg51.h>
sbit inbit = P1^0;
sbit outbit = P2^7;
bit membit;

// use bit to declare
// bit-addressable memory

void main(void)
{
    while(1)
    {
        membit = inbit; // get a bit from P1.0
        outbit = membit; // send it to P2.7
    }
}
```

LOGIC OPERATORS IN C

- Logical operators (Byte level) in C are

- AND (&)
- OR (|)
- NOT (!)

LOGICAL BITWISE OPERATORS IN C

- Bit-wise operators

AND (&), OR (|), EXOR (^), Inverter (~), Shift Right (>>) & shift left (<<)

• These operators are widely used in software engineering for embedded systems and control.

A	B	AND A & B	OR A B	EX-OR A ^ B	Inverter ~ B
0	0	0	0	0	1
0	1	0	1	1	0
1	0	0	1	1	0
1	1	1	1	0	1

SUBROUTINE

(call & rer instructions)

What is Subroutine?

- Subroutine (also called as procedure or function) a reusable program module.
 - write it once.
 - Store it once.
 - invoke it from several places in the main program.
 - when finished it returns to the main program.
- A sequence of instructions invoked so that it looks like a single instruction in the main source-code.
- Syntax:
`CALL SubroutineName.`
- A Program module used multiple time during the execution of a program.
- It can be imbedded repeatedly in the main program listing but it is more efficiently called by the main program.
- Subroutines increase our programming productivity.

Subroutines

A Subroutine is a program that may be used many times in the execution of a larger program. The subroutine could be written in to the body of the main program everywhere it is needed, resulting in the fastest possible code execution.

calls and the Stack

- A call, whether hardware or software initiated, causes a jump to the address where the called subroutine is located. At the end of the subroutine the program resumes operation at the opcode address immediately following the call. As calls can be located anywhere in the program address space and used many times, there must be an automatic means of storing the address of the instruction following the call so that program execution can continue after the subroutine has executed.
- The stack area of internal RAM is used to automatically store the address, called the return address, of the instruction found immediately after the call. The stack pointer register holds the address of the last space used on the stack. It stores the return address above this space, adjusting itself upward as the return address is stored. The terms "stack" and "stack pointer" are often used interchangeably to designate the top of the stack area in RAM that is pointed to by the stack pointer.

Calls and Returns

Calls use short or long range addressing; returns have no addressing mode specified but are always long range. The following table shows examples of call opcodes

Mnemonic	Operation
ACALL Saddr	Call the subroutine located on the same page as the address of the instruction immediately after the call on the stack.
LCALL Laddr	Call the subroutine located anywhere in the program memory space; push the address of the instruction immediately following the call on the stack.
RET	POP two bytes from the stack into the program counter.

How to use Subroutines?

To use a subroutine.

- There must be a mechanism for transferring control to the beginning of the subroutine and
- Then returning to execute what would have been the next instruction after the subroutine is done.
- This should work no matter where in memory the subroutine is called from.
- It should also work even if the subroutine is called from inside another subroutine.
- It turns out that the stack is the perfect mechanism for keeping track of return address.
- It is also very helpful in passing parameters to and from subroutines and in defining variables local to subroutines.

CALL and RETURN

- Calling A Subroutine (CALL instruction)
 - main program "transfers control" to the subroutine.
 - main program "jumps" to subroutine.
 - PC = address of (first instruction in subroutine)
- Returning from A Subroutine (RET instruction)
 - transfers control back to the main program.
 - "jumps" back into main program.
 - PC = address of (first instruction after the calling instruction)

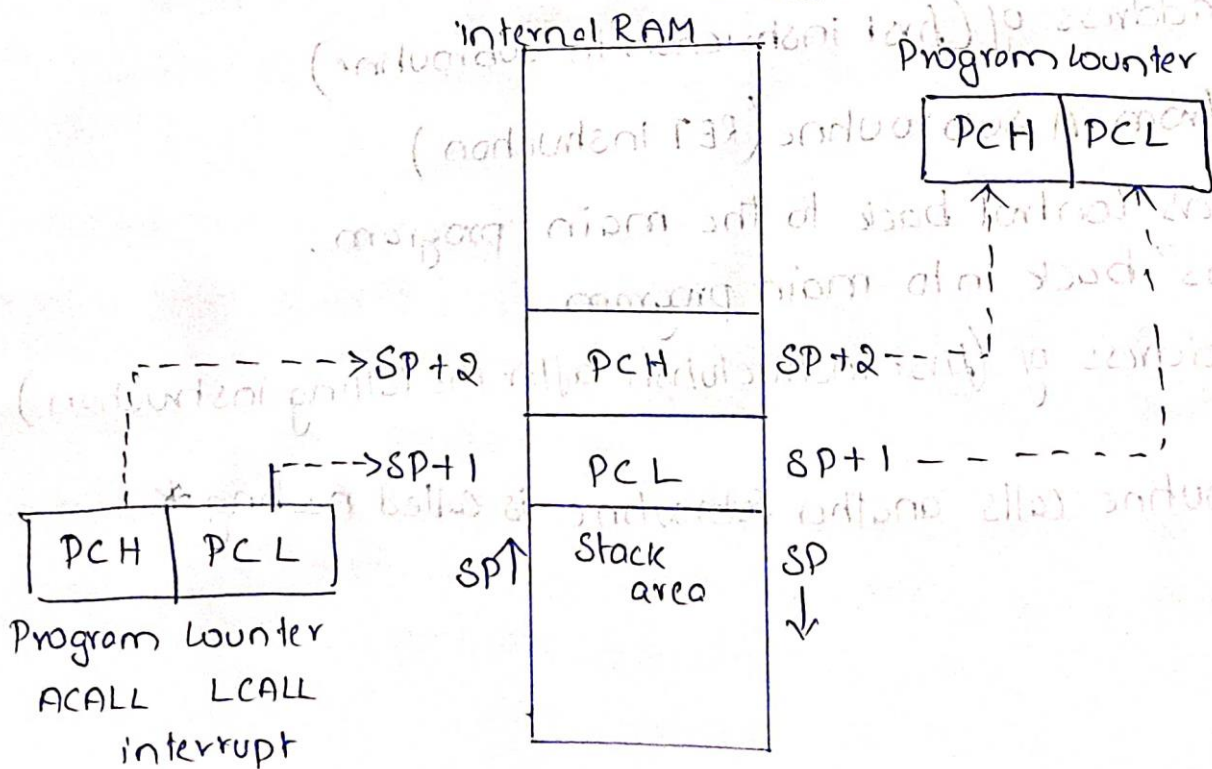
* One Subroutine calls another Subroutine is called nesting *

Steps for calling subroutine & returning back.

- ① A call opcode occurs in the program software, or an interrupt is generated in the hardware circuitry.
- ② The return address of the next instruction after the call instruction or interrupt is found in the program counter.
- ③ The return address bytes are pushed on the stack, low byte first.
- ④ The stack pointer is incremented for each push on the stack.
- ⑤ The subroutine address is placed in the program counter.
- ⑥ The subroutine is executed.
- ⑦ A RET (return) opcode is encountered at the end of the subroutine.
- ⑧ Two pop operations restore the return address to the PC from the stack area in internal RAM.
- ⑨ The stack pointer is decremented for each address byte pop.

* All of these steps are automatically handled by the 8051 hardware. It is the responsibility of the programmer to ensure that the subroutine ends in a RET instruction and that the stack does not grow up into data areas that are used by the program.*

Storing and Retrieving the Return Address



Example

; MAIN program calling subroutines

```
                ORG 0  
MAIN:           LCALL SUBR-1  
                LCALL SUBR-2  
                LCALL SUBR-3
```

```
HERE:          SJMP  HERE  
; ————— end of MAIN  
;
```

```
SUBR-1:         . . .  
                . . .  
                RET  
; ————— end of subroutine 1  
;
```

```
SUBR-2:         . . .  
                . . .  
                RET  
; ————— end of subroutine 2
```

```
SUBR-3:         . . .  
                . . .  
                RET  
; ————— end of subroutine 3
```

```
                END ; end of the asm file.
```


Module – 3

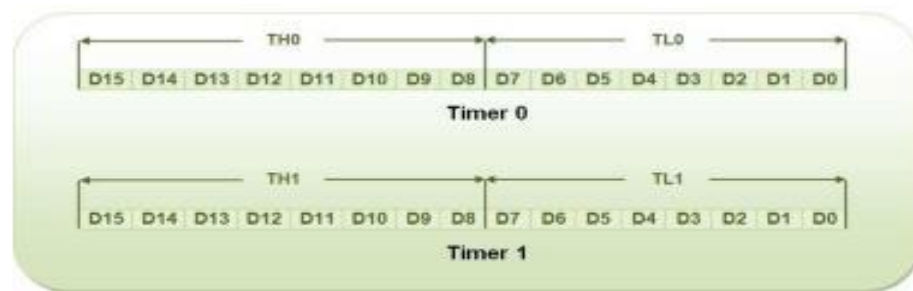
8051 Timers and Serial Port

Timers/Counters are used generally for

- ☐ Time reference
- ☐ Creating delay
- ☐ Wave form properties measurement
- ☐ Periodic interrupt generation
- ☐ The 8051 has two timers/counters, they can be used either as Timers to generate a time delay or as Event counters to count events happening outside the microcontroller

8051 has two timers, Timer 0 and Timer 1.

- ☐ Timer 0 and Timer 1 are 16 bits.
- ☐ 8051 has an 8-bit architecture, each 16-bits timer is accessed as two separate registers of low byte and high byte.
- ☐ The low byte register is called TL0/TL1 and The high byte register is called TH0/TH1.
- ☐ Accessed like any other register.



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 microsecond.

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

- Both timers 0 and 1 use the same register, called TMOD (timer mode), to set the various timer operation modes
- TMOD is an 8-bit register
- The lower 4 bits are for Timer 0, the upper 4 bits are for Timer 1
- In each case, the lower 2 bits are used to set the timer mode, the upper 2 bits to specify the operation.

GATE	C/T	M1	M0	GATE	C/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/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 (timer control) register

- TCON (timer control) register is an 8bit 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/low 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 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.

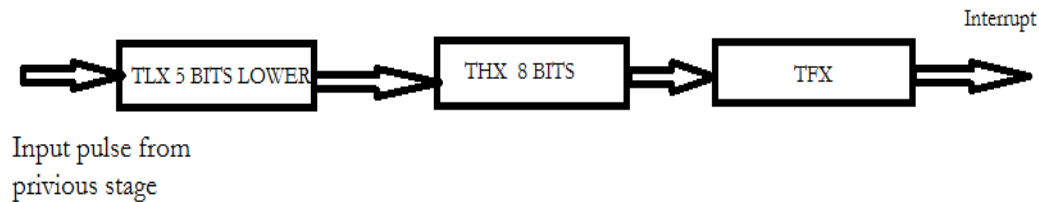


Fig. Operation of Timer on Mode-0

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.

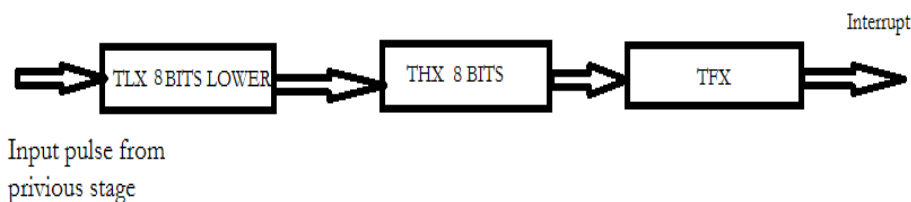


Fig: Operation of Timer in Mode 1

Timer Mode-2: (Auto-Reload Mode): This is an 8 bit counter/timer operation. Counting is performed in TLX while THX stores a constant value. In this mode when the timer overflows i.e. TLX becomes FFH, it is fed with the value stored in THX. For example, if we 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.

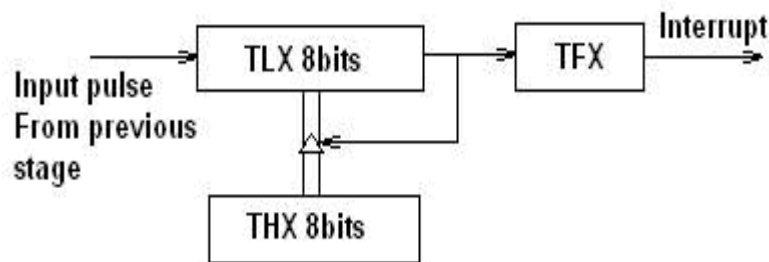


Fig: Operation of Timer in Mode 2

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.

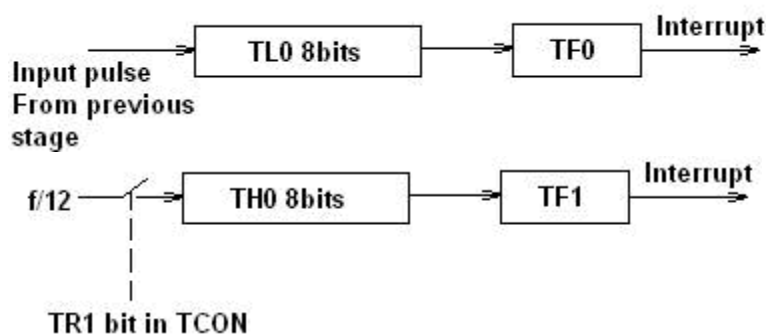


Fig: Operation of Timer in Mode 3

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 = $1/\text{Timer Clock Frequency}$.
 $= 1/(\text{Master Clock Frequency}/12)$

1. Mode 1 (16 bit timer/counter)

Value to be loaded in decimal = $65536 - (\text{Delay required}/\text{Timer clock period})$

Convert the answer into hexadecimal and load onto THx and TLx register.

$(65536D = FFFFH+1)$

2. Mode 0 (13 bit timer/counter)

Value to be loaded in decimal = $8192 - (\text{Delay required}/\text{Timer clock period})$

Convert the answer into hexadecimal and load onto THx and TLx register.

$(8192D = 1FFFH+1)$

3. Mode 2 (8 bit auto reload)

Value to be loaded in decimal = $256 - (\text{Delay required} / \text{Timer clock period})$

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.

Example 4-1

Indicate which mode and which timer are selected for each of the following.

(a) MOV TMOD, #01H (b) MOV TMOD, #20H (c) MOV TMOD, #12H

Solution:

We convert the value from hex to binary. From Figure 9-3 we have:

(a) TMOD = 00000001, mode 1 of timer 0 is selected.

(b) TMOD = 00100000, mode 2 of timer 1 is selected.

(c) TMOD = 00010010, mode 2 of timer 0, and mode 1 of timer 1 are selected

Example 4-2

Find the timer's clock frequency and its period for various 8051-based system, with the crystal frequency 11.0592 MHz when C/T bit of TMOD is 0.



$$1/12 \times 11.0529 \text{ MHz} = 921.6 \text{ MHz};$$

$$T = 1/921.6 \text{ kHz} = 1.085 \text{ us}$$

Example 4-3

In the following program, we create a square wave of 50% duty cycle (with equal portions high and low) on the P1.5 bit. Timer 0 is used to generate the time delay. Analyze the program

```

HERE:    MOV TMOD,#01    ;Timer 0, mode 1(16-bit mode)
        MOV TL0,#0F2H    ;TL0=F2H, the low byte
        MOV TH0,#0FFH    ;TH0=FFH, the high byte
        CPL P1.5         ;toggle P1.5
        ACALL DELAY
        SJMP HERE

DELAY:   SETB TR0        ;start the timer 0
AGAIN:   JNB TF0,AGAIN    ;monitor timer flag 0 ;until it rolls over
        CLR TR0         ;stop timer 0
        CLR TF0         ;clear timer 0 flag
        RET

```

In the above program notice the following step.

1. TMOD is loaded.
2. FFF2H is loaded into TH0-TL0.
3. P1.5 is toggled for the high and low portions of the pulse.
4. The DELAY subroutine using the timer is called.
5. In the DELAY subroutine, timer 0 is started by the SETB TR0 instruction.
6. Timer 0 counts up with the passing of each clock, which is provided by the crystal oscillator. As the timer counts up, it goes through the states of FFF3, FFF4, FFF5, FFF6, FFF7, FFF8, FFF9, FFFA, FFFB, and so on until it reaches FFFFH. One more clock rolls it to 0, raising the timer flag (TF0=1). At that point, the JNB instruction falls through.



7. Timer 0 is stopped by the instruction CLR TR0. The DELAY subroutine ends, and the process is repeated.

Notice that to repeat the process, we must reload the TL and TH registers, and start the process is repeated

Example 4-4

In Example 9-4, calculate the amount of time delay in the DELAY subroutine generated by the timer.

Assume XTAL = 11.0592 MHz.

Solution:

The timer works with a clock frequency of 1/12 of the XTAL frequency; therefore, we have $11.0592 \text{ MHz} / 12 = 921.6 \text{ kHz}$ as the timer frequency. As a result, each clock has a period of $T = 1/921.6 \text{ kHz} = 1.085 \mu\text{s}$. In other words, Timer 0 counts up each 1.085 μs resulting in delay = number of counts \times 1.085 μs .

The number of counts for the roll over is $\text{FFFFH} - \text{FFF2H} = 0\text{DH}$ (13decimal). However, we add one to 13 because of the extra clock needed when it rolls over from FFFF to 0 and raise the TF flag. This gives $14 \times 1.085 \mu\text{s} = 15.19 \mu\text{s}$ for half the pulse. For the entire period it is $T = 2 \times 15.19 \mu\text{s} = 30.38 \mu\text{s}$ as the time delay generated by the timer.

Example 4-5

The following program generates a square wave on 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,#10    ;Timer 1, mod 1 (16-bitmode)
AGAIN:          MOV  TL1,#34H    ;TL1=34H, low byte of timer
                MOV  TH1,#76H    ;TH1=76H, high byte timer
                SETB TR1          ;start the timer 1
BACK:           JNB  TF1,BACK    ;till timer rolls over
                CLR  TR1         ;stop the timer 1
                CPL  P1.5        ;comp. p1. to get hi, lo
                CLR  TF1         ;clear timer flag 1
                SJMP AGAIN       ;is not auto-reload

```

Solution:

Since $\text{FFFFH} - 7634\text{H} = 89\text{CBH} + 1 = 89\text{CCH}$ and $89\text{CCH} = 35276$ clock count and $35276 \times 1.085 \mu\text{s} = 38.274 \text{ ms}$ for half of the square wave. The frequency = 13.064Hz. Also notice that the high portion and low portion of the square wave pulse are equal. In the above calculation, the overhead due to all the instruction in the loop is not included.

Example 4-6

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 $T = 1/(2 \text{ kHz}) = 500 \mu\text{s}$. Each half pulse = $250 \mu\text{s}$.

The value n for $250 \mu\text{s}$ is: $250 \mu\text{s} / 1 \mu\text{s} = 250 \times 65536 - 250 = \text{FF06H}$. $\text{TL} = 06\text{H}$ and $\text{TH} = 0\text{FFH}$.

```

                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

```

Example4-6

Write a program segment that uses timer 1 in mode 2 to toggle P1.0 once whenever the counter reaches a count of 100. Assume the timer clock is taken from external source P3.5 (T1).

The TMOD value is 60H The initialization value to be loaded into TH1 is $256 - 100 = 156 = 9\text{CH}$

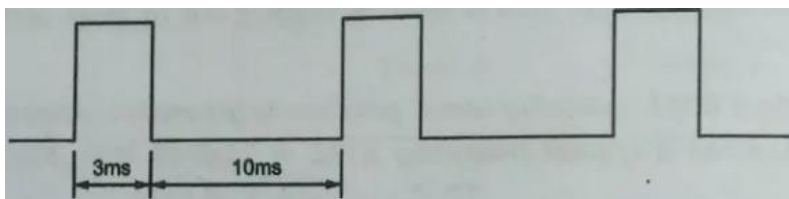
```

                MOV TMOD, #60h ;Counter1, mode 2, C/T'= 1
                MOV TH1, #9Ch  ;Counting 100 pulses
                SETB P3.5      ;Make T1 input
                SETB TR1       ;Start timer 1
BACK:           JNB TF1, BACK  ;Keep doing it if TF = 0
                CPL P1.0       ;Toggle port bit
                CLR TF1        ;Clear timer overflow flag
                SJMP BACK      ;Keep doing it

```

Example4-7

Generate a square wave with an ON time of 3ms and an OFF time of 10ms on all pins of port 0. Assume an XTAL of 22 MHz.



Solution:

Timer Clock period = $1/\text{Timer Clock Frequency}$.
 $= 1/(\text{Master Clock Frequency}/12)$

Tclk = $12/22\text{MHZ} = 0.546\mu\text{s}$

For OFF Time calculation:

$10\text{ms}/0.546\mu\text{s} = 18,315 \text{ cycle}$

$65536 - 18,315 = 47,221 = \text{B875H}$

For ON Time calculation:

$3\text{ms}/0.546\mu\text{s} = 5,494 \text{ cycle}$

$65536 - 5,494 = 60,042 = \text{EA8AH}$

Tested for an AT89C51 with a crystal frequency of 22 MHz.

Let us use Timer 0 in Mode 1.

MOV TMOD , #01H ; Timer 0 in mode 1

BACK: MOV TL0 , # 075H ; to generate the OFF time , load TL0

MOV TH0 , # 0B8H ; load OFF time value in TH0

MOV P0 , # 00H ; make port bits low

ACALL DELAY ; call delay routine

MOV TL0 , # 8AH ; to generate the ON time , load TL0

MOV TH0 , # 0EAH ; load ON time value in TH0

MOV P0 , # 0FFH ; make port bits high

ACALL DELAY ; call delay

SJMP BACK ; repeat for reloading counters to get a continuous

;square wave

ORG 300H

DELAY: SET TR0 ; start the counter

AGAIN: JNB TF0,AGAIN ; check timer overflow

CLR TR0 ; when TF0 is set , stop the timer

CLR TF0 ; clear timer flag

RET

END ; end of file

Example4-8

Examine the following program and find the time delay in seconds. Exclude the overhead due to the instructions in the loop.

```
        MOV    TMOD,#10H    ;Timer 1, mod 1
        MOV    R3,#200      ;cnter for multiple delay
AGAIN:  MOV    TL1,#08H      ;TL1=08,low byte of timer
        MOV    TH1,#01H      ;TH1=01,high byte
        SETB   TR1           ;Start timer 1
BACK:   JNB    TF1,BACK      ;until timer rolls over
        CLR    TR1           ;Stop the timer 1
        CLR    TF1           ;clear Timer 1 flag
        DJNZ   R3,AGAIN      ;if R3 not zero then
                                ;reload timer
```

Solution:

- Timer 1, mod 1= 16bit counter
- **No. of counts Roll over the timer=Maximum Bits timer can Hold – Count loaded in the Timer register.**
- $65536 - 264 = 65272$. (TH-TL = 0108H = 264 in decimal)
- **Time Delay = Roll over Count – Clock period**
- Time Delay = $65272 \times 1.085 \mu\text{s} = 70.820 \text{ ms}$, and
- for 200 of them we have $200 \times 70.820 \text{ ms} = 14.164024 \text{ seconds}$.

SERIAL COMMUNICATION

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

Basics of serial data communication

Communication Links

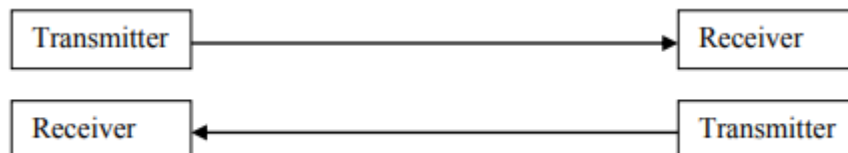
1. **Simplex communication link:** In simplex transmission, the line is dedicated for transmission. The transmitter sends and the receiver receives the data.



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



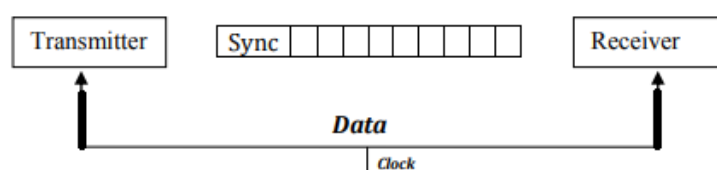
3. **Full duplex communication link:** If the data is transmitted in both ways at the same time, it is a full duplex i.e. transmission and reception can proceed simultaneously. This communication link requires two wires for data, one for transmission and one for reception.



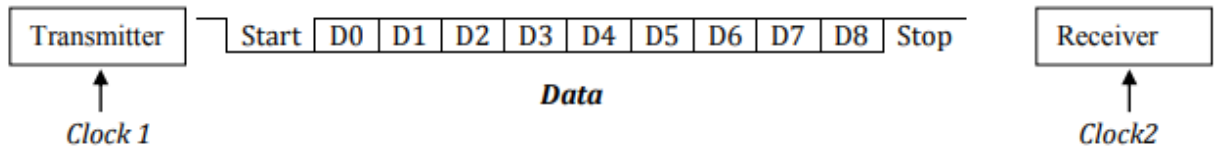
Types of Serial communication:

Serial data communication uses two types of communication.

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



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



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

8051 SERIAL COMMUNICATION

Three special function registers support serial communication.

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

SM0	SM1	SM2	REN	TB8	RB8	TI	RI
-----	-----	-----	-----	-----	-----	----	----

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

Note: Make SM2, TB8, and RB8 =0

SM0, SM1: They determine the framing of data by specifying the number of bits per character, and the start and stop bits

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

SM2: This enables the multiprocessing capability of the 8051

REN (receive enable) : It is a bit-addressable register f . When it is high, it allows 8051 to receive data on RxD pin f . If low, the receiver is disabled.

TI (transmit interrupt) : When 8051 finishes the transfer of 8-bit character f , it raises 'TI' flag to indicate that it is ready to transfer another byte f . 'TI' bit is raised at the beginning of the stop bit %0.

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

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

PCON : Power Control Register (Not Bit Addressable)

SMOD	-	-	-	GF1	GF0	PD	IDL
------	---	---	---	-----	-----	----	-----

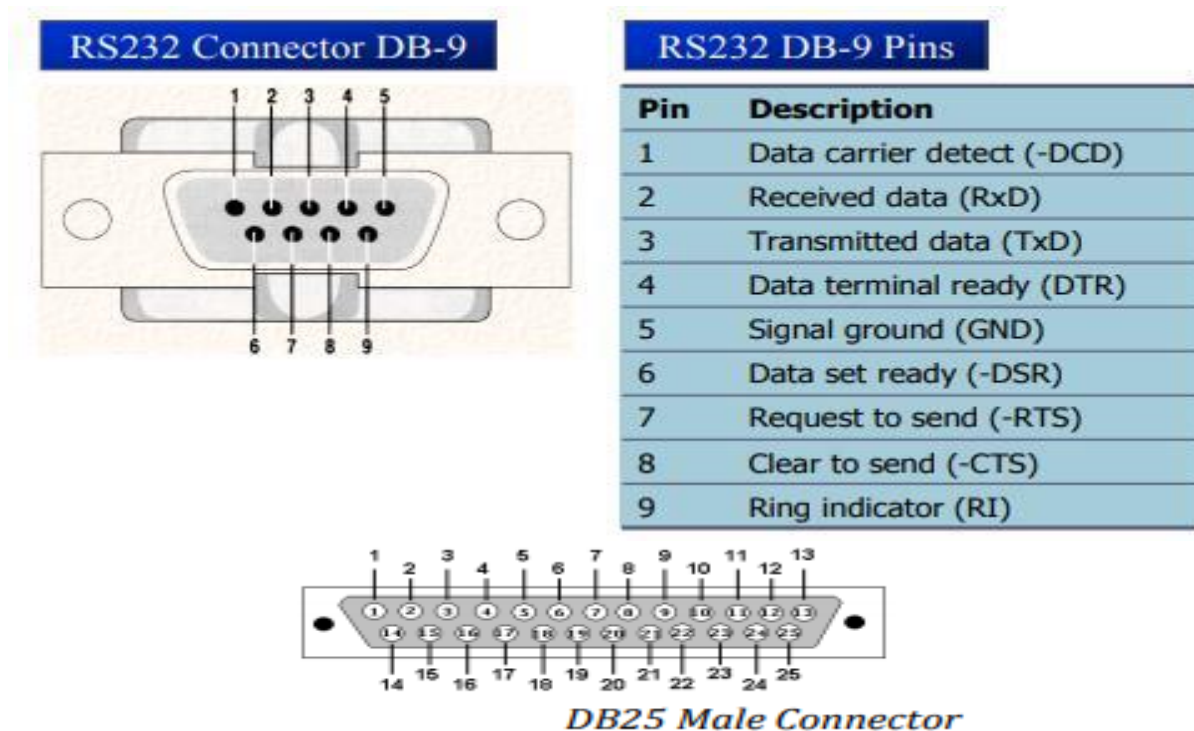
SMOD	PCON.7	Double baud rate bit. If SMOD = 1, the baud rate is doubled when the serial port is used in mode 1, 2 and 3.
-	PCON.6	Not implemented, reserved for future use*
-	PCON.5	Not implemented, reserved for future use*
-	PCON.4	Not implemented, reserved for future use*
GF1	PCON.3	General purpose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

SERIAL COMMUNICATION MODES

1. **Mode 0:** In this mode serial port runs in synchronous mode. The data is transmitted and received through RXD pin and TXD is used for clock output. In this mode the baud rate is 1/12 of clock frequency.
2. **Mode 1:** In this mode SBUF becomes a 10 bit full duplex transceiver. The ten bits are 1 start bit, 8 data bit and 1 stop bit. The interrupt flag TI/RI will be set once transmission or reception is over. In this mode the baud rate is variable and is determined by the timer 1 overflow rate. Baud rate = $[2\text{smod}/32] \times \text{Timer 1 overflow Rate} = [2\text{smod}/32] \times [\text{Oscillator Clock Frequency}] / [12 \times [256 - [\text{TH1}]]]$
3. **Mode 2:** This is similar to mode 1 except 11 bits are transmitted or received. The 11 bits are, 1 start bit, 8 data bit, a programmable 9th data bit, 1 stop bit. Baud rate = $[2\text{smod}/64] \times \text{Oscillator Clock Frequency}$.
4. **Mode 3:** This is similar to mode 2 except baud rate is calculated as in mode 1.

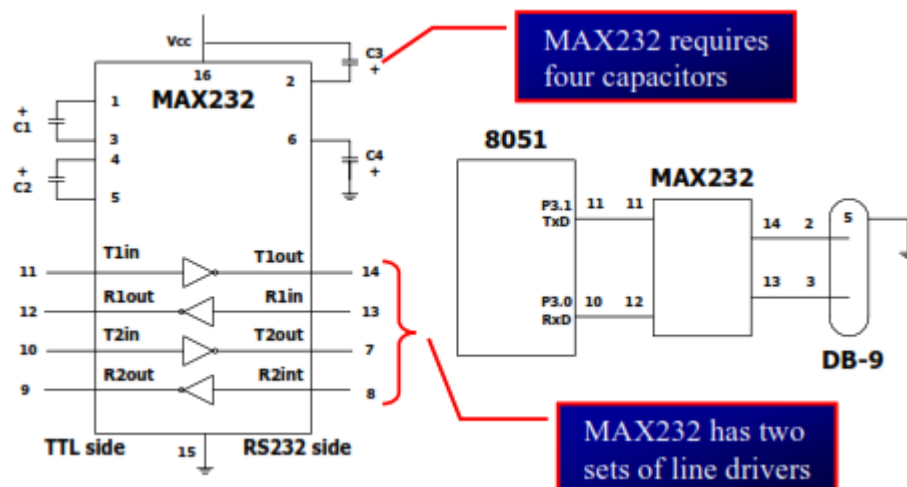
RS-232 standards: To allow compatibility among data communication equipment made by various manufacturers, an interfacing standard called RS232 was set by the Electronics Industries Association (EIA) in 1960. Since the standard was set long before the advent of logic family, its input and output voltage 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.



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 in mode 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 serial data transfer.
3. The SCON register is loaded with the value 50H, indicating serial mode 1, where an 8-bit data is framed with start and stop bits.
4. TR1 is set to 1 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 4.7

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

Solution:

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

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

Example 4.8

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

Solution:

```

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

```

Example 4.9

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

```

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

```

Example 4.10

Write a C program for 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, mode 2
    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, mode 2
    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;
}
```