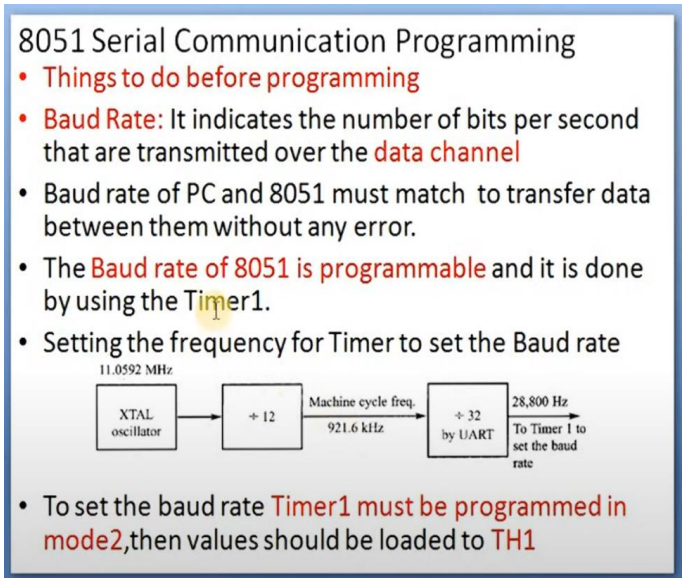
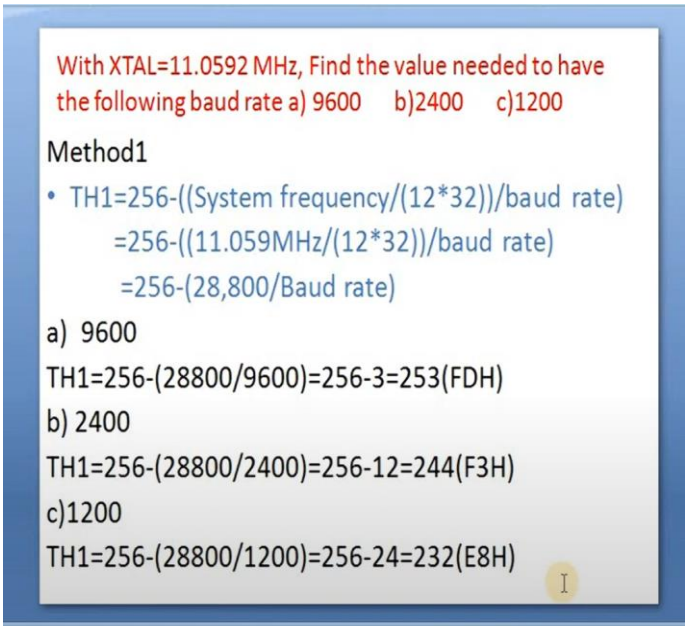
Youtube Playlist to be follow for understanding serial communication:

(1241) Serial communication of 8051 microcontroller - YouTube

Also follow the below Youtube video:

(1241) 8051 serial port programming using assembly and C - YouTube





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

Method 2

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

(a) 28,800 / 3 = 9600 where -3 = FD (hex) is loaded into TH1

(b) 28,800 / 12 = 2400 where -12 = F4 (hex) is loaded into TH1

(c) 28,800 / 24 = 1200 where -24 = E8 (hex) is loaded into TH1

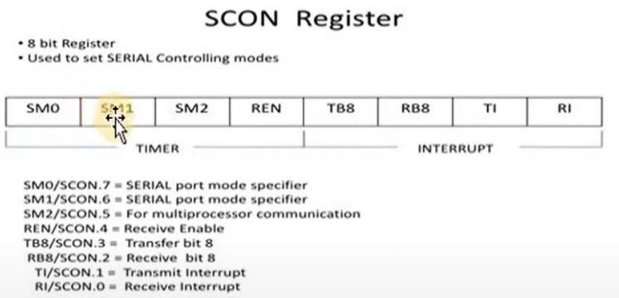
SBUF register

- SBUF is an single 8- bit register used 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.
- The Serial Port receive and transmit register are both accessed at Serial

```
MOV SBUF, # 'H' ; (The Moment a byte is written in to SBUF, it is
                   framed with the start and stop bits before
transmitting similarly it eliminated while it is received through Rx pin
)
MOV  SBUF, A ;
MOV A, SBUF ;
```

SCON(serial control) REGISTER

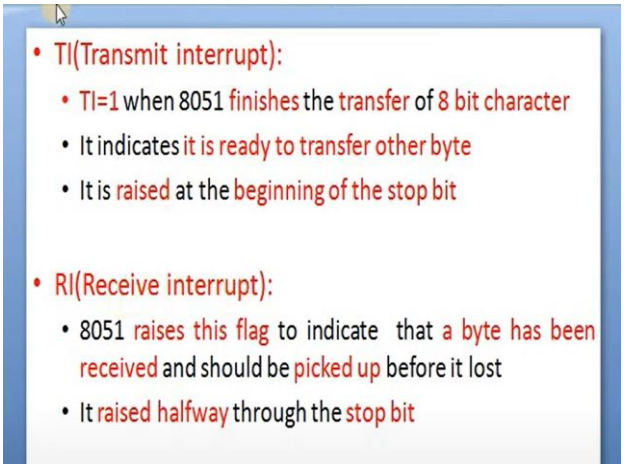
- It is an 8 bit register used to Program the start bit, stop bit and the data bits of data framing among other things.
-



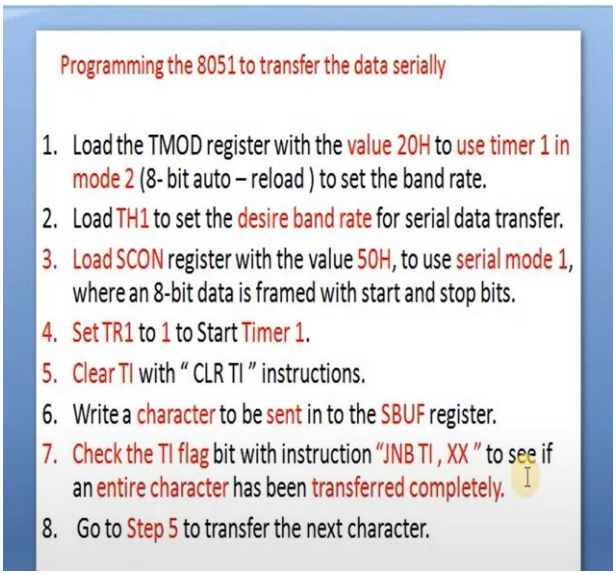
- SM0,SM1:Determine the framing of data by specifying the number of bits per character, and start and stop bits.

SM0	SM1	Mode
0	0	Serial Mode 0
0	1	Serial mode1,8 bit data,1 start bit.1 stop bit
1	0	Serial Mode 2
1	1	Serial Mode 3

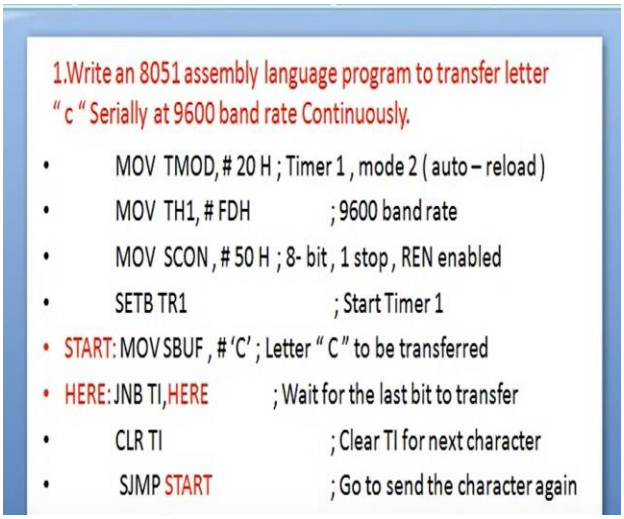
- SM2:Enables multiprocessing capability. Since we are not using multiprocessor we will make SM2=0
- REN(Receive Enable):REN=1 enables 8051 to receive the data and Receiver is disabled by making REN=0
- TB8(Transfer bit 8):It is used for serial mode2 and 3.Since we will be using only mode1,we make TB8=0
- RB8(Receive bit 8):This bit gets a copy of the stop bit when an 8 bit data is received. In our discussion we make RB8=0



Steps in programming 8051 to <u>TRANSFER</u> data Serially:



Programing in Assembly language:



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
```

Steps in programming 8051 to <u>Recieve</u> data Serially:

Steps In programming the 8051 to receive character bytes serially

1. TMOD register is loaded with the value 20H, indicating the use of timer 1 in mode 2 (8-bit auto-reload) to set baud rate
2. TH1 is loaded to set baud rate
3. The SCON register is loaded with the value 50H, indicating serial mode 1, where an 8- bit data is framed with start and stop bits
4. TR1 is set to 1 to start timer 1
5. RI is cleared by CLR RI instruction
6. The RI flag bit is monitored with the use of instruction JNB RI, xx to see if an entire character has been received yet
7. When RI is raised, SBUF has the byte, its contents are moved into a safe place
8. To receive the next character, go to step 5

Programing in Assembly Language:

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

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

Now Programming in C language:

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

```
#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='C';               //place value in buffer
While (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

```
#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;
}
```

Task_15_02:

Write a program to send data to PC from the microcontroller using serial port at 9600 baud rate.
Send any message like "Hello world".
Use \n\r to see how it takes the cursor to a new line on the Hercules software e.g "Hello world\n\r"

```
//To write a program that sends data from a microcontroller to a PC using the serial port at 9600 baud rate,
//you can use the following C code. This program is designed for the 8051 microcontroller (using the Keil compiler),
//and it will send the message "Hello world" followed by a newline and carriage return ("\n\r") to a PC.
//The delay function ensures there is enough time for each character to be transmitted.
*/

#include <reg51.h>

void delay(unsigned int itime) {
    int i, j;
    for (i = 0; i < itime; i++) {
        for (j = 0; j < 114; j++);
    }
}

unsigned int u = 0;

void main() {
    unsigned char Mystring[] = "Hello world\n\r";  // The message to be sent

    TMOD = 0x20;  // Timer1 in Mode2 (8-bit auto-reload)
    TH1 = 0xFD;   // Baud rate 9600 for 11.0592 MHz
    SCON = 0x50;  // Serial mode 1, 8-data bit, 1 stop bit, REN enabled
    TR1 = 1;      // Start Timer1

    for (u = 0; u < 14; u++) {
        SBUF = Mystring[u];  // Load the character to SBUF
        while (TI == 0);     // Wait until the transmission is complete
        TI = 0;              // Clear the TI flag
    }

    delay(300);  // Wait for a while to ensure transmission is complete
}



//-------------------------------------------------------------------------------------------------------------------------
```

Explanation of the above Code:

Understanding the Code Block

The code block you provided is used to create and display a custom character on an LCD screen connected to a microcontroller. Let's break down each part:

CGRAM (Character Generator RAM)

1. **CGRAM** stands for **Character Generator RAM**. This is a special area of memory inside the LCD where you can create custom characters.
2. The LCD has a set of predefined characters (like letters and numbers), but sometimes you want to show a character that isn't built-in (like a smiley face). CGRAM allows you to design and store these custom characters.

DDRAM (Display Data RAM)

1. **DDRAM** stands for **Display Data RAM**. This is where the characters that are actually displayed on the LCD screen are stored.
2. When you write data to DDRAM, you're telling the LCD what characters to show and where to show them on the screen.

The Code Block Explained

Let's go through the code line by line:

1. `cmd(64);`
    - This sets the **CGRAM address** to 0x40 (which is the starting address for storing the first custom character). It's like telling the LCD, "I'm going to give you the design for a new character, and I want you to start storing it at this address."
2. `dat(0);`

- This sends the first row of pixel data for the custom character to the LCD. Here, `0` means that the first row of the character will be all blank (no pixels turned on).

3. `dat(10);`
    - This sends the second row of pixel data. The value `10` (in binary: `00001010`) turns on some pixels in the second row, creating part of the custom character.
4. `dat(21);`
    - This sends the third row of pixel data. The value `21` (in binary: `00010101`) continues to shape the custom character.
5. `dat(17);`
    - This sends the fourth row of pixel data. The value `17` (in binary: `00010001`) adds more detail to the custom character.
6. `dat(10);`
    - This sends the fifth row of pixel data. Again, the value `10` turns on specific pixels.
7. `dat(4);`
    - This sends the sixth row of pixel data. The value `4` (in binary: `00000100`) turns on a single pixel in the sixth row.
8. `dat(0);`
    - This sends the seventh row of pixel data. The value `0` means that the seventh row will be blank.
9. `dat(0);`
    - This sends the eighth (and final) row of pixel data. The value `0` means that the eighth row will also be blank.

Moving to DDRAM

10. `cmd(0xc0);`
    - This sets the DDRAM address to 0xC0, which is the starting address for the second line of the display. It's like saying, "I want to start showing characters on the second line of the screen."
11. `dat(0);`
    - This sends the code for the custom character stored at location 0 in CGRAM to DDRAM. It tells the LCD to display this custom character at the current position on the screen.
12. `lcd_delay();`
    - This introduces a delay to give the LCD time to process the commands and data. It's like saying, "Take a moment to get everything set up."

Summary

- **CGRAM** allows you to create custom characters by specifying the pixel data for each row.
- **DDRAM** is where the display data is stored, determining what characters appear on the screen and where.
- This code block creates a custom character by writing pixel data to CGRAM and then displays this character by writing its code to DDRAM.

Purpose

The purpose of this code is to:

1. Create a custom character that isn't available in the LCD's built-in character set.
2. Display this custom character on the LCD screen, specifically on the second line.

By using CGRAM and DDRAM effectively, you can make your LCD show any custom-designed characters you need for your project.