3. Enable the Timer interrupt by passing hex value to IE register or setting required bits of IE register. For example,

   IE = 0x82; enables Timer 0 interrupt

   IE = 0x88; enables Timer 1 interrupt

   or

   EA = 1;

   ET0 = 1; enables Timer 0 interrupt

   IE^7 = 1;

   IE^3 = 1; enables Timer 1 interrupt

4. Start the Timer by setting TRx bit.
5. Write Interrupt Service Routine (ISR) for the Timer interrupt.
6. If the Timer has to be stopped after once the interrupt has occurred, the ISR must contain the statement to stop the Timer.
7. If a routine written for Timer interrupt has to be repeated again and again, the Timer run bit need not be cleared. But it should be kept in mind that Timer will start updating from 0000H and not the initial values in case of mode 0 and 1. So the initial values must be reloaded in the interrupt service routine.

Serial Port Programming: 8051 Serial Communication

One of the 8051's many powerful features -integrated *UART*, known as a serial port to easily read and write values to the serial port instead of turning on and off one of the I/O lines in rapid succession to properly "clock out" each individual bit, including start bits, stop bits and parity bits.

ü **Setting the Serial Port Mode** configures it by specifying 8051 how many data bits

ü **Setting the Serial Port Baud Rate**

ü **Writing to the Serial Port**

ü **Reading the Serial Port**

<u>INTERRUPT PROGRAMMING</u>:

What Events Can Trigger interrupts, and where do they go?

The following events will cause an interrupt:

· Timer 0 Overflow.

· Timer 1 Overflow.

· Reception/Transmission of Serial Character.

· External Event 0.

· External Event 1.

To distinguish between various interrupts and executing different code depending on what interrupt was triggered 8051may be jumping to a fixed address when a given interrupt occurs.

ü **Setting Up Interrupts**

By default at power up, all interrupts are disabled. Even if, for example, the TF0 bit is set, the 8051 will not execute the interrupt. Your program must specifically tell the

8051 that it wishes to enable interrupts and specifically which interrupts it wishes to enable. Your program may enable and disable interrupts by modifying the IE SFR (A8h):

Table 5.6 Interrrupt and address

Table 5.6 Interrrupt and address

Bit	Name Bit	Address	Explanation of Function
7	EA	AFh	Global Interrupt Enable/Disable
6		AEh	Undefined
5		ADh	Undefined
4	ES	ACh	Enable Serial Interrupt
3	ET1	ABh	Enable Timer 1 Interrupt
2	EX1	AAh	Enable External 1 Interrupt
1	ET0	A9h	Enable Timer 0 Interrupt
0	EX0	A8h	Enable External 0 Interrupt

Each of the 8051'sinterrupts has its own bit in the IE SFR. You enable a given interrupt by setting the corresponding bit. For example, if you wish to enable Timer 1 Interrupt, you would execute either:

MOV IE,#08h || SETB ET1

Both of the above instructions set bit 3 of IE, thus enabling Timer 1 Interrupt. Once Timer 1 Interrupt is enabled, whenever the TF1 bit is set, the 8051 will automatically put "on hold" the main program and execute the Timer 1 Interrupt Handler at address 001Bh. However, before Timer 1 Interrupt (or any other interrupt) is truly enabled, you must also set bit 7 of IE.

Bit 7, the Global Interrupt Enable/Disable, enables or disables all interrupts simultaneously. That is to say, if bit 7 is cleared then no interrupts will occur, even if all the other bits of IE are set. Setting bit 7 will enable all the interrupts that have been selected by setting other bits in IE. This is useful in program execution if you have time-critical code that needs to execute. In this case, you may need the code to execute from start to finish without any interrupt getting in the way. To accomplish this you can simply clear bit 7 of IE (CLR EA) and then set it after your time critical code is done.

ü **Polling Sequence**

The 8051 automatically evaluates whether an interrupt should occur after every instruction. When checking for interrupt conditions, it checks them in the following order:

1) External 0 Interrupt

2) Timer 0 Interrupt

3) External 1 Interrupt

4) Timer 1 Interrupt

5) Serial Interrupt

Interrupt Priorities

The 8051 offers two levels of interrupt priority: high and low. By using interrupt priorities you may assign higher priority to certain interrupt conditions. For example, you may have enabled Timer 1 Interrupt which is automatically called every time Timer 1 overflows. Additionally, you may have enabled the Serial Interrupt which is called every time a character is received via the serial port. However, you may consider that receiving a character is much more important than the timer interrupt. In this case, if Timer 1 Interrupt is already executing you may wish that the serial interrupt itself interrupts the Timer 1 Interrupt. When the serial interrupt is complete, search control passes back to Timer 1 Interrupt and finally back to the main program. You may accomplish this by assigning a high priority to the Serial Interrupt and a low priority to the Timer 1 Interrupt.

What Happens When an Interrupt Occurs?

When an interrupt is triggered, the following actions are taken automatically by the microcontroller:

- The current Program Counter is saved on the stack, low-byte first.
- Interrupts of the same and lower priority are blocked.
- In the case of Timer and External interrupts, the corresponding interrupt flag is set.
- Program execution transfers to the corresponding interrupt handler vector address.
- The Interrupt Handler Routine executes. Take special note of the third step: If theinterrupt being handled is a Timer or External interrupt, the microcontroller automatically clearsthe interrupt flag before passing search control to your interrupt handler routine.

What Happens When an Interrupt Ends?

An interrupt ends when your program executes the RETI instruction. When the RETI instruction is executed the following actions are taken by the microcontroller:

- Two bytes are popped off the stack into the Program Counter to restore normal program execution.
- Interrupt status is restored to its pre-interrupt status.

LCD (LIQUID CRYSTAL DISPLAY) INTERFACE

LCDs can display numbers, characters, and graphics. To produce a proper display,the information has to be periodically refreshed. This can be done by the CPU or internally by the LCD device itself. Incorporating a refreshing controller into the LCD, relieves the CPU of this task and hence many LCDs have built-in controllers. These controllers also facilitate flexible programming for characters and graphics. Table 5.1 shows the pin description of an LCD. from Optrex.