Chapter 3

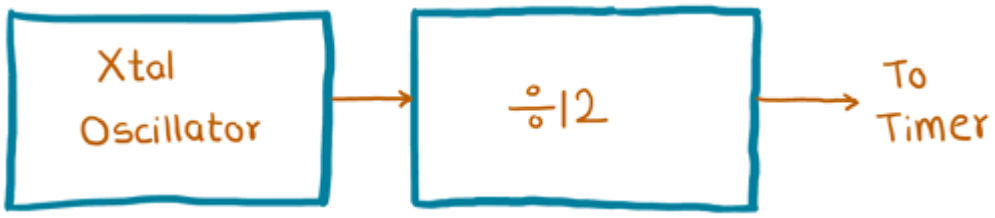
3.1 Timer/Counters :SFRs: TMOD, TCON, Timer/Counter - Logic and modes, Simple programs on timer to generate time delay
3.2 Interrupts-SFRs:- IE, IP , Simple programs on interrupts
3.3 Serial communication - SFRs: SCON , SBUF , PCON, Modes of serial communication. Simple programs on serial communication
3.4 I/O port structure and configuration - P0 , PI , P2 ,P3

3.1 Counters and Timers

As the frequency of the oscillator is precisely defined and very stable, pulses it generates are always of the same width, which makes them ideal for time measurement. Such crystals are also used in quartz watches. In order to measure time between two events it is sufficient to count up pulses coming from this oscillator. That is exactly what the timer does. If the timer is properly programmed, the value stored in its register will be incremented (or decremented) with each coming pulse, i.e. once per each machine cycle. A single machine-cycle instruction lasts for 12 quartz oscillator periods, which means that by embedding quartz with oscillator frequency of 12MHz, a number stored in the timer register will be changed million times per second, i.e. each microsecond.
The 8051 microcontroller has 2 timers/counters called Timer0 and Timer1. As their names suggest, their main purpose is to measure time and count external events. Besides, they can be used for generating clock pulses to be used in serial communication, so called Baud Rate.

a. Clock

Every Timer needs a clock to work, and 8051 provides it from an external crystal which is the main clock source for Timer. The internal circuitry in the 8051 microcontrollers provides a clock source to the timers which is 1/12th of the frequency of crystal attached to the microcontroller, also called as Machine cycle frequency.



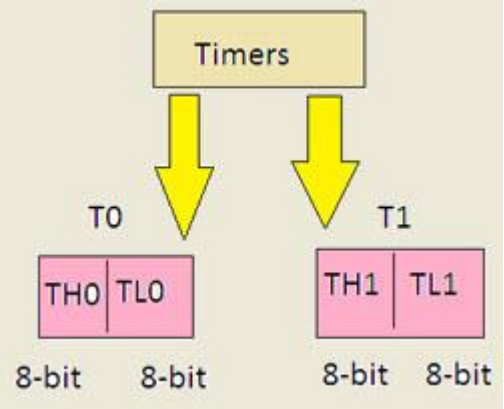
8051 Timer Clock

For example, suppose we have crystal frequency of 11.0592 MHz then microcontroller will provide 1/12th i.e.
Timer clock frequency= (Xtal Osc.frequency)/12 = (11.0592 MHz)/12 = 921.6 KHz
period T= 1/(921.6 KHz)=1.085 µS

b. Counter Mode

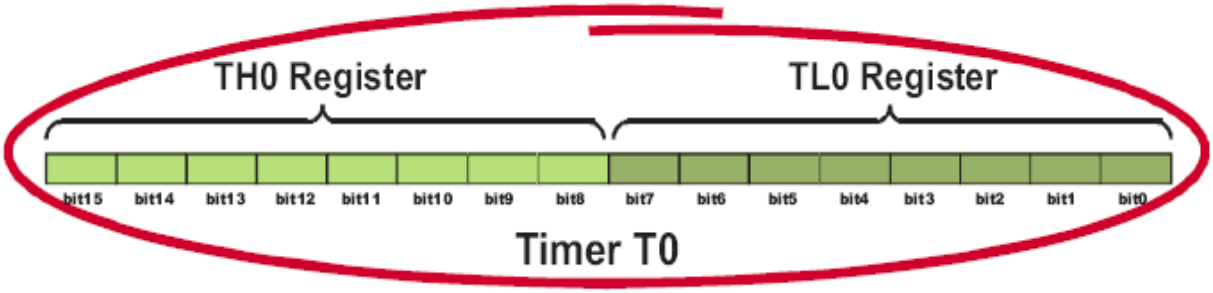
In the counter mode, the external events are counted. In this mode, the timer register is incremented for each 1 to 0 transition of the external input pin. This type of transitions is treated as events. The external input pins are sampled once in each machine cycle, and to determine the 1or 0 transitions, another machine cycle will be needed. So in this mode, at least two machine cycles are needed. When the frequency is12MHz, then the maximum count frequency will be 12MHz/24 = 500KHz. So for event counting the time duration is 2 µs.

There are two 16-bit timers and counters in 8051 microcontroller: timer 0 and timer 1. Both timers consist of 16-bit register in which the lower byte is stored in TL and the higher byte is stored in TH.

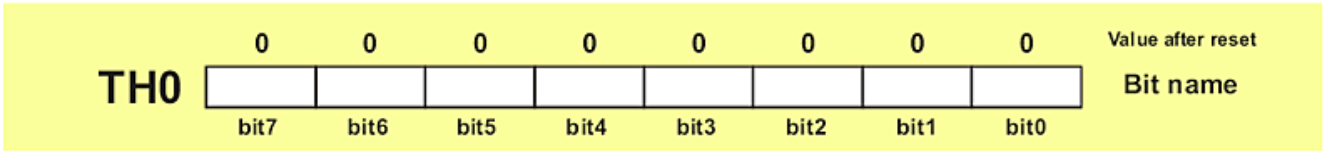


Timer T0

As seen in figure below, the timer T0 consists of two registers – TH0 and TL0 representing a low and a high byte of one 16-digit binary number.



Accordingly, if the content of the timer T0 is equal to 0 (T0=0) then both registers it consists of will contain 0. If the timer contains for example number 1000 (decimal), then the TH0 register (high byte) will contain the number 3, while the TL0 register (low byte) will contain decimal number 232.

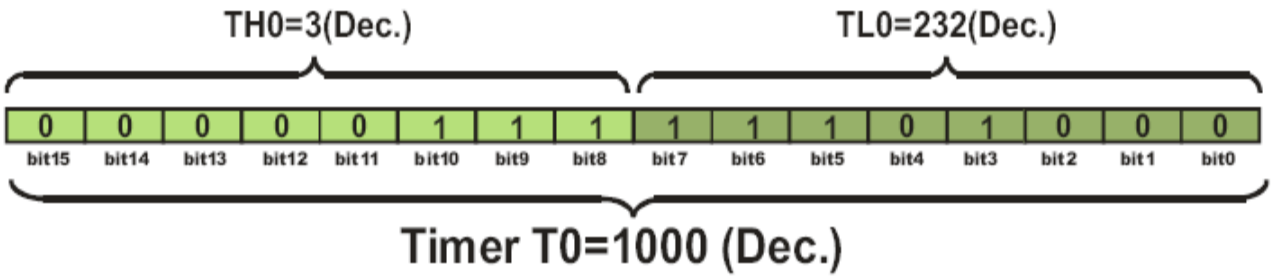


Formula used to calculate values in these two registers is very simple:

TH0 × 256 + TL0 = T

Matching the previous example it would be as follows:

3 × 256 + 232 = 1000



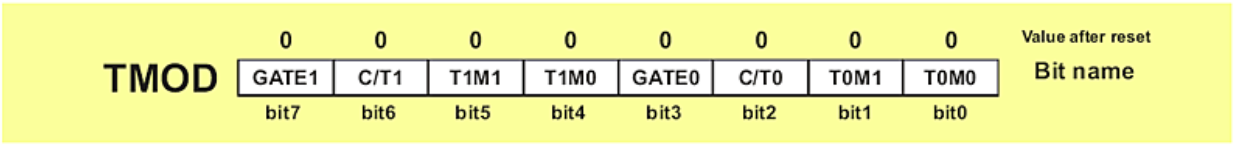
Since the timer T0 is virtually 16-bit register, the largest value it can store is 65 535. In case of exceeding this value, the timer will be automatically cleared and counting starts from 0. This condition is called an overflow. Two registers TMOD and TCON are closely connected to this timer and control its operation.

Timer 1 :

Similarly TL1 and TH1 registers are available to load the count values for Timer 1.

c. TMOD Register (Timer Mode)

The TMOD register selects the operational mode of the timers T0 and T1. As seen in figure below, the low 4 bits (bit0 - bit3) refer to the timer 0, while the high 4 bits (bit4 - bit7) refer to the timer 1. There are 4 operational modes and each of them is described herein.



Bits of this register have the following function:

- GATE1 enables and disables Timer 1 by means of a signal brought to the INT1 pin (P3.3):
  - 1 - Timer 1 operates only if the INT1 bit is set.
  - 0 - Timer 1 operates regardless of the logic state of the INT1 bit.
- C/T1 selects pulses to be counted up by the timer/counter 1:
  - 1 - Timer counts pulses brought to the T1 pin (P3.5).
  - 0 - Timer counts pulses from internal oscillator.
- T1M1,T1M0 These two bits select the operational mode of the Timer 1.
-

T1M1	T1M0	MODE	DESCRIPTION
0	0	0	13-bit timer
0	1	1	16-bit timer
1	0	2	8-bit auto-reload
1	1	3	Split mode

- GATE0 enables and disables Timer 1 using a signal brought to the INT0 pin (P3.2):
  - 1 - Timer 0 operates only if the INT0 bit is set.
  - 0 - Timer 0 operates regardless of the logic state of the INT0 bit.
- C/T0 selects pulses to be counted up by the timer/counter 0:
  - 1 - Timer counts pulses brought to the T0 pin (P3.4).
  - 0 - Timer counts pulses from internal oscillator.
- T0M1,T0M0 These two bits select the opertaional mode of the Timer 0.

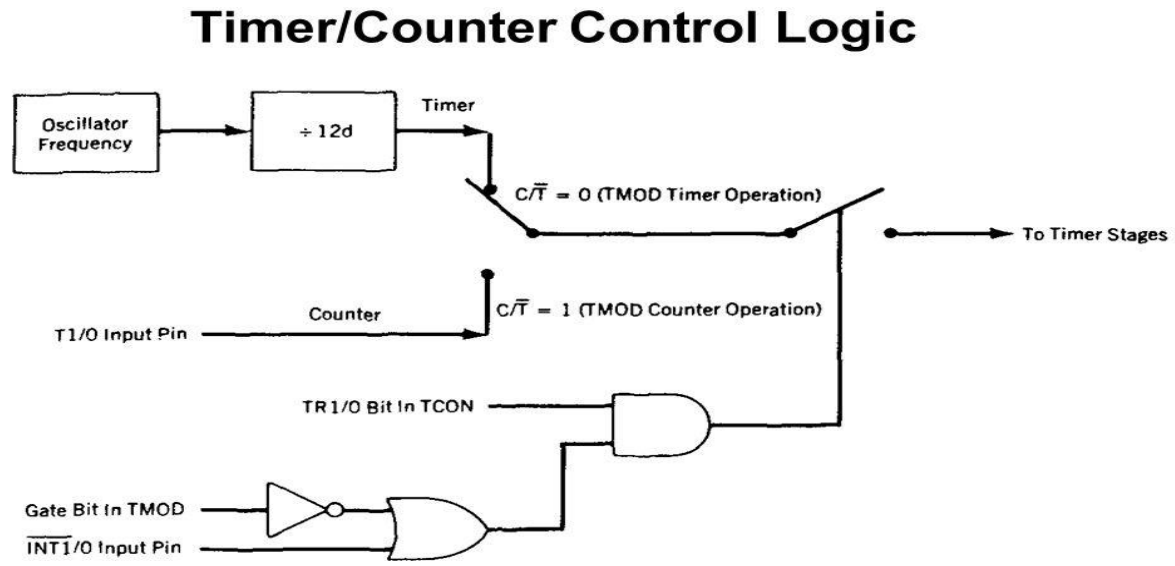
d. TCON Register (Timer Control)

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

Fig. TCON Register

BIT	Symbol	Functions
7	TF1	Timer1 over flow flag. Set when timer rolls from all 1s to 0. Cleared when the processor vectors to execute interrupt service routine Located at program address 001Bh.
6	TR1	Timer 1 run control bit. Set to 1 by programmer to enable timer to count; Cleared to 0 by program to halt timer.
5	TF0	Timer 0 over flow flag. Same as TF1.
4	TR0	Timer 0 run control bit. Same as TR1.
3	IE1	External interrupt 1 Edge flag.
2	IT1	External interrupt1 signal type control bit. Set to 1 by program to Enable external interrupt 1 to be triggered by a falling edge signal. Set To 0 by program to enable a low level signal on external interrupt1 to generate an interrupt.
1	IE0	External interrupt 0 Edge flag.
0	IT0	External interrupt 0 signal type control bit. Same as IT0.

e. Timer Control logic



If we want to timer as a timer , for delay generation, C/T is made 0 and it will count the on chip clock frequency of 8051 oscillator divided by 12. The resultant timer clock is gated to the timer by means of the circuit shown in Figure above. If C/T is made 1, the timer is clocked externally through T0 or T1 pin.

If GATE bit of TMOD register is at logic 0, we must set Bit TRX in the TCON register to '1' to run the timer. If GATE bit of TMOD register is at logic 1 , we must set Bit TRX in the TCON register to '1' to run the timer and also external pin (INTX) must be a 1 . In short, the timer is configured as a timer, then the timer pulses are gated to the timer by the run bit and the gate bit or the external input bits (INTX).

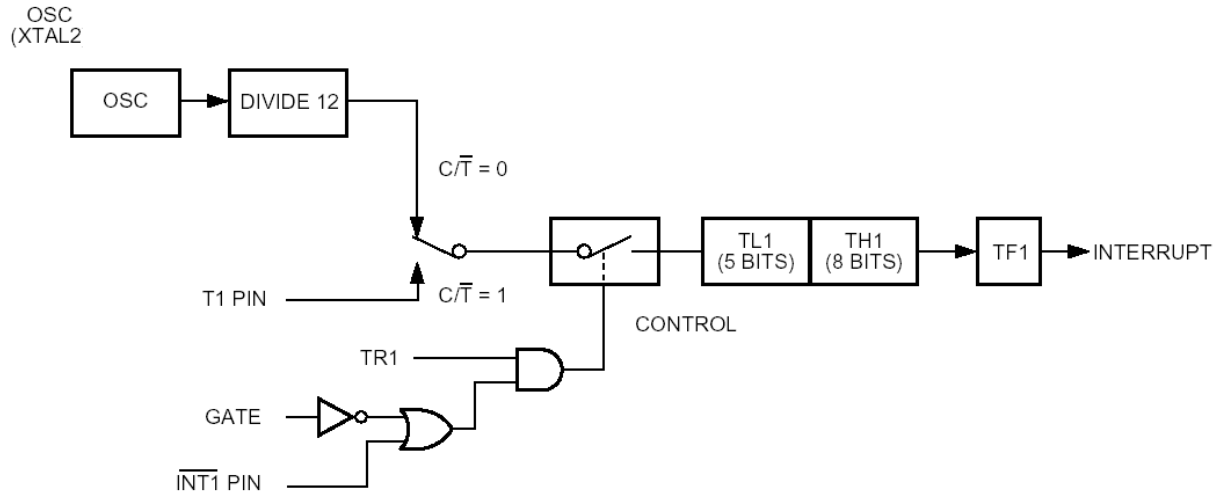
In the following table, we will see the bit details and their different operations for high or low value.

Bit Details	High Value(1)	Low Value(0)
C/**T**	Configure for the Counter operations	Configure for the Timer operations
Gate (G)	Timer0 or Timer1 will be in RunMode when TRX bit of TCON register is high.	Timer0 or Timer1 will be in RunMode when TRX bit of TCON register is high and INT0 or INT1 is high.

f. Modes of Timer:

13-bit Time Mode (mode 0)

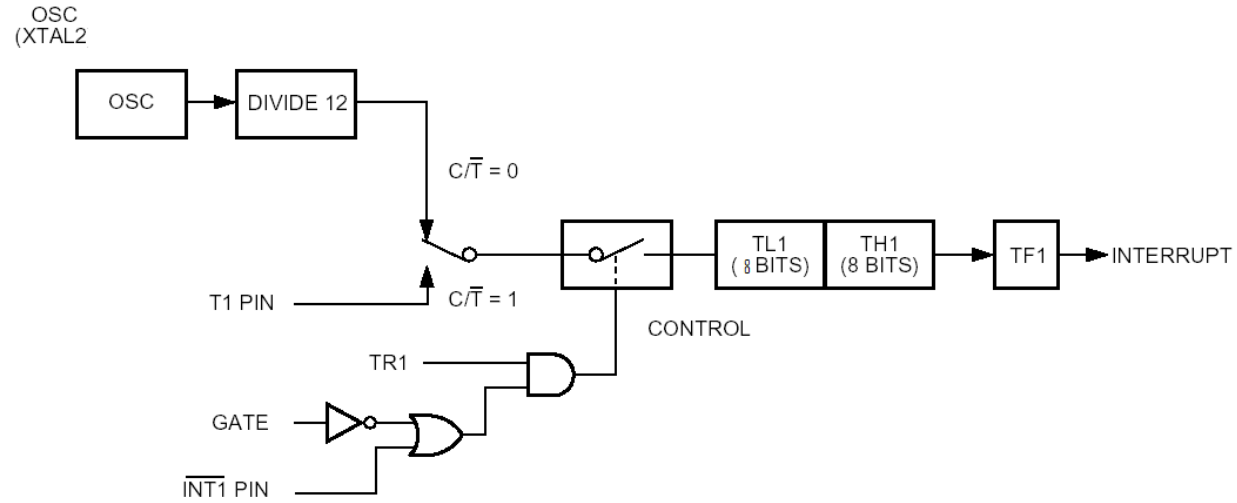
Timer mode "0" is a 13-bit timer. This is a relic that was kept around in the 8051 to maintain compatability with its predecesor, the 8048. Generally the 13-bit timer mode is not used in new development.
When the timer is in 13-bit mode, TLx will count from 0 to 31. When TLx is incremented from 31, it will "reset" to 0 and increment THx. Thus, effectively, only 13 bits of the two timer bytes are being used: bits 0-4 of TLx and bits 0-7 of THx. This also means, in essence, the timer can only contain 8192 values. If you set a 13-bit timer to 0, it will overflow back to zero 8192 machine cycles later.



16-bit Time Mode (mode 1)

Timer mode "1" is a 16-bit timer. This is a very commonly used mode. It functions just like 13-bit mode except that all 16 bits are used.

TLx is incremented from 0 to 255. When TLx is incremented from 255, it resets to 0 and causes THx to be incremented by 1. Since this is a full 16-bit timer, the timer may contain up to 65536 distinct values. If you set a 16-bit timer to 0, it will overflow back to 0 after 65,536 machine cycles.



The following are the characteristics and operations of mode1:

1. It is a 16-bit timer; therefore, it allows value of 0000 to FFFFH to be loaded into the timer's register TL and TH 2. After TH and TL are loaded with a 16-bit initial value, the timer must be started. This is done by SETB TR0 for timer 0 and SETB TR1 for timer 1.
2. After the timer is started, it starts to count up. It counts up until it reaches its limit of FFFFH. When it rolls over from FFFFH to 0000, it sets high a flag bit called TF (timer flag). Each timer has its own timer flag: TF0 for timer 0, and TF1 for timer 1.This timer flag can be monitored.
3. When this timer flag is raised, one option would be to stop the timer with the instructions CLR TR0 or CLR TR1, for timer 0 and timer 1, respectively.
4. After the timer reaches its limit and rolls over, in order to repeat the process TH and TL must be reloaded with the original value, and TF must be reloaded to 0

Steps to generate a time delay in mode 1 (polling method)
1. Load the TMOD value register indicating which timer (timer 0 or timer 1) is to be used and which timer mode (0 or 1) is selected
2. Load registers TL and TH with initial count value
3. Start the timer
4. 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
5. Stop the timer
6. Clear the TF flag for the next round
7. Go back to Step 2 to load TH and TL again

Delay Calculation:

The timer works with a clock frequency of 1/12 of the XTAL frequency; therefore, we have 11.0592 MHz / 12 = 921.6 kHz as the timer frequency.
As a result, each clock has a period of T = 1/921.6kHz = 1.085us. In other words, Timer 0 counts up each 1.085 us resulting in delay = number of counts × 1.085us.
(a) in hex
(FFFF – YYXX + 1) × 1.085 us, where YYXX are TH, TL initial values respectively. Notice that value YYXX are in hex.
(b) in decimal

Convert YYXX values of the TH, TL register to decimal to get a NNNNN decimal, then (65536 - NNNN) × 1.085 us

MODE 1 Example Programs:

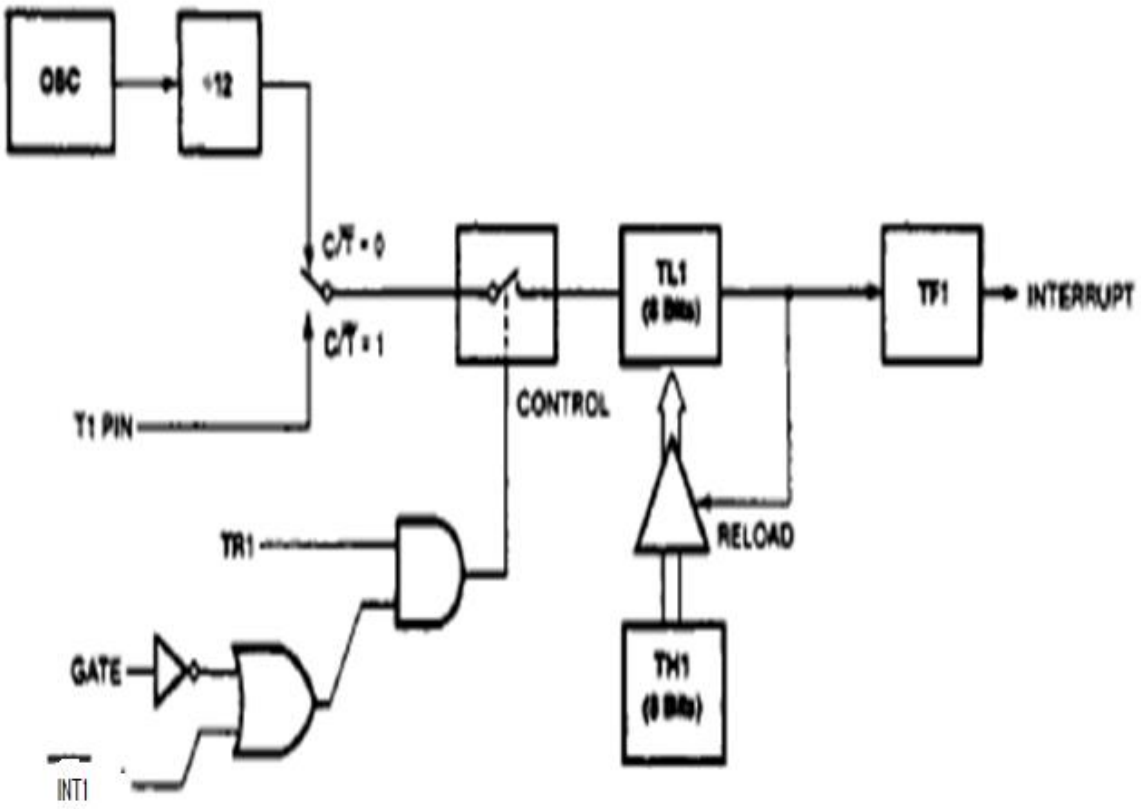
1.Assume that XTAL = 11.0592 MHz, write a program to generate a square wave of 2 kHz frequency on pin P1.5.

Solution:
Timer 0 is used for delay generation.
(a) T = 1 / f = 1 / 2 kHz = 500 us the period of square wave.
(b) 1 / 2 of it for the high and low portion of the pulse is 250 us.
(c) 250 us / 1.085 us = 230 and 65536 – 230 = 65306 which in hex is FF1AH. (write full steps as done in the lecture)
(d) TL = 1A and TH = FF, all in hex.
The program is as follow.

```
MOV TMOD,#01              ;Timer 0, 16-bitmode
AGAIN: MOV TL0,#1AH       ;TL0=1A, low byte of timer
MOV TH0,#0FFH             ;TH0=FF, the high byte
SETB TR0                  ;Start timer 0
BACK: JNB TF0,BACK        ;until timer rolls over
CLR TR0                   ;Stop the timer 0
CPL  P1.5                 ;toggle for square wave
CLR TF0                   ;Clear timer 0 flag
SJMP AGAIN                ;Reload timer
```

8-bit Time Mode (mode 2)

Timer mode "2" is an 8-bit auto-reload mode. When a timer is in mode 2, THx holds the "reload value" and TLx is the timer itself. Thus, TLx starts counting up. When TLx reaches 255 and is subsequently incremented, instead of resetting to 0 (as in the case of modes 0 and 1), it will be reset to the value stored in THx.



The following are the characteristics and operations of mode 2:

1. It is an 8-bit timer; therefore, it allows only values of 00 to FFH to be loaded into the timer's register TH
2. After TH is loaded with the 8-bit value, the 8051 gives a copy of it to TL. Then the timer must be started. This is done by the instruction SETB TR0 for timer 0 and SETB TR1 for timer 1
3. After the timer is started, it starts to count up by incrementing the TL register. It counts up until it reaches its limit of FFH. When it rolls over from FFH to 00, it sets high the TF (timer flag).
4. When the TL register rolls from FFH to 0 and TF is set to 1, TL is reloaded automatically with the original value kept by the TH register. To repeat the process, we must simply clear TF. This makes mode 2 an auto-reload, in contrast with mode 1 in which the programmer has to reload TH and TL

Steps to program in Mode 2. (polling method)

1. Load the TMOD value register indicating which timer (timer 0 or timer 1) is to be used, and the timer mode (mode 2) is selected.
2. Load the TH registers with the initial count value.
3. Start timer.
4. Keep monitoring the timer flag (TF) with the JNB TFx,target instruction to see whether it is raised. Get out of the loop when TF goes high.
5. Clear the TF flag.
6. Go back to Step4, since mode 2 is autoreload.

Delay Calculation:

The timer works with a clock frequency of 1/12 of the XTAL frequency; therefore, we have 11.0592 MHz / 12 = 921.6 kHz as the timer frequency.

As a result, each clock has a period of T = 1/921.6kHz = 1.085us. In other words, Timer 0 counts up each 1.085 us resulting in delay = number of counts × 1.085us.

(a) in hex

(FF – XX + 1) × 1.085 us, where XX are TH, TL initial values . Notice that value XX is in hex.
(b) in decimal
Convert XX values of the TH, TL register to decimal to get a NNN decimal, then (256-NNN) × 1.085 us

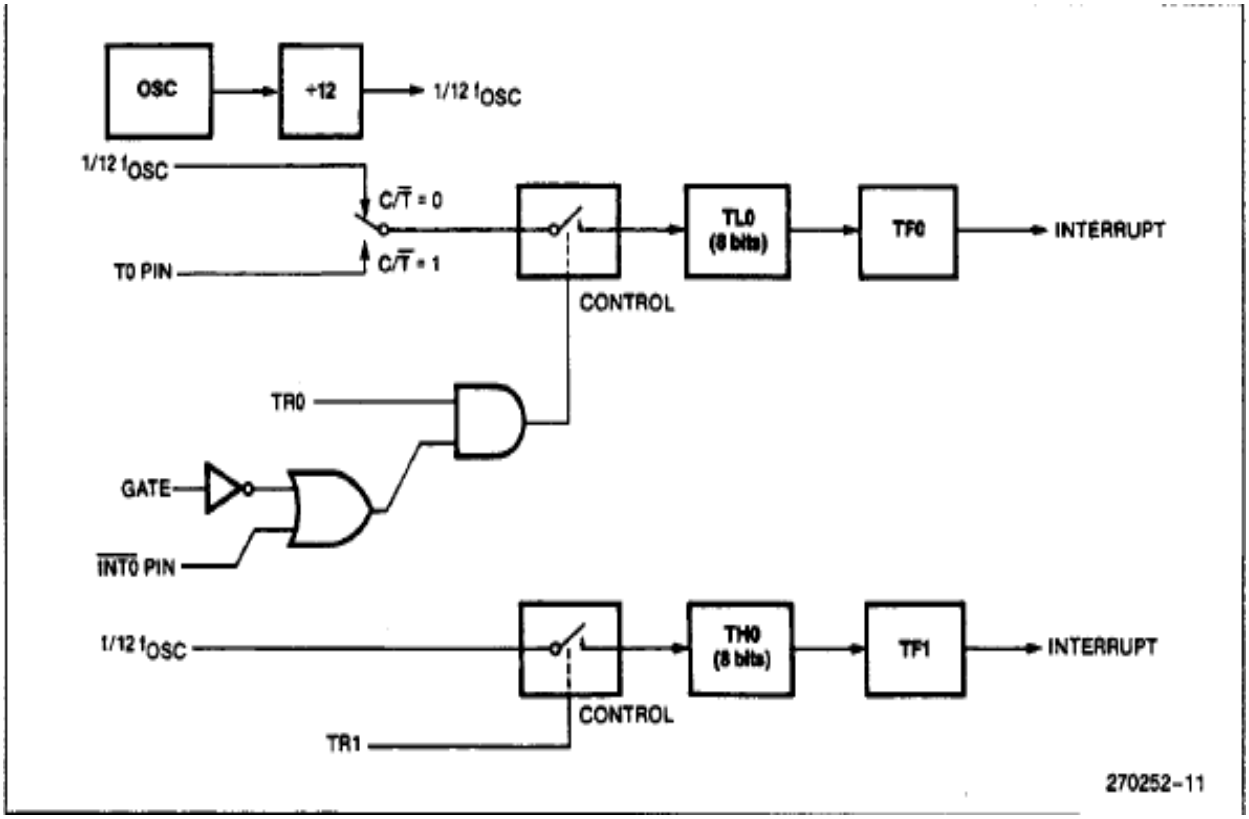
Example program in Mode 2

Write a program togenerate a square wave of 200 μs period on pin P2.1. Use timer 0 in mode 2 to create the square wave. Assume that XTAL = 11.0592 MHz.
Solution:
We will use timer 0 in mode 2 (auto reload).
Ton = Toff = 200/2 = 100 μs
TH0 = 256 –(100/1.085 us) = 164D = A4H

```
MOV TMOD,#02H          ;T0/8-bit/auto reload
MOV TH0,#0A4           ; TH0 = count value
MOV TH1,#0A4           ; TH1 = count value
SETB TR0               ;start the timer 0
BACK: JNB TF0,BACK     ;till timer rolls over
CPL P2.1               ;toggle p2.1
CLR TF0                ;clear Timer 0 flag
SJMP BACK              ;mode 2 is auto-reload
```

Split Timer Mode (mode 3)

Timer mode "3" is a split-timer mode. When Timer 0 is placed in mode 3, it essentially becomes two separate 8-bit timers. Timer 0 is TL0 and Timer 1 is TH0. Both timers count from 0 to 255 and overflow back to 0. All the bits that are related to Timer 1 will now be tied to TH0.
While Timer 0 is in split mode, the real Timer 1 (i.e. TH1 and TL1) can be put into modes 0, 1 or 2 normally--however, you may not start or stop the real timer 1 since the bits that do that are now linked to TH0. The real timer 1, in this case, will be incremented every machine cycle no matter what.
In such case you can use the real Timer 1 as a baud rate generator and use TH0/TL0 as two separate timers.



g. Example Programs:

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

T = 1 / f = 1 / 2 kHz = 500 us the period of square wave.

1 / 2 of it for the high and low portion of the pulse is 250 us.

(65535-count+1) X 1.085 us = 250 us
65536-count = 250 us / 1.085 us = 230
Count = 65536 – 230 = 65306 which in hex is FF1AH.

```
ORG 0000H
AGAIN: SETB P1.5
ACALL T1M1DELAY
CLR P1.5
ACALL T1M1DELAY
SJMP AGAIN
T1M1DELAY:
MOV TMOD,#10H              ; Timer 0, 16-bitmode
MOV TL1,#1AH    ;  TL1=1A, low byte of  timer
MOV TH1,#0FFH              ;   TH1=FF, the 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 flag 1
RET
END
```

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

T = 1 / f = 1 / 5 kHz = 200 us the period of square wave.

1 / 2 of it for the high and low portion of the pulse is 100 us.

(65535-count+1) X 1.085 us = 100 us
65536-count = 100 us / 1.085 us = 92
Count = 65536 – 92 = 65444 which in hex is FFA4H.

```
ORG 0000H
AGAIN: SETB P1.5
ACALL T1M1DELAY
CLR P1.5
ACALL T1M1DELAY
SJMP AGAIN
T1M1DELAY:
MOV TMOD,#10H              ; Timer 0, 16-bitmode
MOV TL1,#A4H    ;  TL1=1A, low byte of  timer
MOV TH1,#0FFH              ;   TH1=FF, the 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 flag 1
RET
END
```

3. **Assume that XTAL = 11.0592 MHz, write a program to generate a square wave of 1kHz frequency on pin P2.0.**

T = 1 / f = 1 /1 kHz = 1000 us the period of square wave.

1 / 2 of it for the high and low portion of the pulse is 500 us.

(65535-count+1) X 1.085 us = 500 us
65536-count = 500 us / 1.085 us = 461
Count = 65536 – 461 = 65075 which in hex is FE33H.

```
ORG 0000H
AGAIN: SETB P2.0
ACALL T0M1DELAY
CLR P2.0
ACALL T0M1DELAY
SJMP AGAIN
T0M1DELAY:
MOV TMOD,#01h              ; Timer 0, 16-bitmode
MOV TL0,#A4H    ;  TL1=1A, low byte of  timer
MOV TH0,#0FFH              ;   TH1=FF, the high byte
SETB TR0                   ;  Start timer 1
```

```
BACK: JNB TF0,BACK       ;  until timer rolls over
CLR TR0                  ;  Stop the timer 1
CLR TF0                  ;   Clear timer flag 1
RET
END
```

4. **Assume that XTAL = 12 MHz, write a program to generate a square wave of 5kHz frequency on pin P2.0. Use Timer0 in Mode1**

T = 1 / f = 1 /5 kHz = 200 us the period of square wave.

1 / 2 of it for the high and low portion of the pulse is 100 us.

(65535-count+1) X 1 us = 100 us
65536-count = 100 us / 1 us = 100
Count = 65536 – 100 = 65436 which in hex is FF9CH.

```
ORG 0000H
AGAIN: SETB P2.0
ACALL T0M1DELAY
CLR P2.0
ACALL T0M1DELAY
SJMP AGAIN
T0M1DELAY:
MOV TMOD,#01h            ; Timer 0, 16-bitmode
MOV TL0,#9CH     ;  TL1=1A, low byte of  timer
MOV TH0,#0FFH            ;   TH1=FF, the high byte
SETB TR0                 ;  Start timer 1
BACK: JNB TF0,BACK       ;  until timer rolls over
CLR TR0                  ;  Stop the timer 1
CLR TF0                  ;   Clear timer flag 1
RET
END
```

3.2 Interrupts

Interrupts are the events that temporarily suspend the main program, pass the control to the external sources and execute their task.
The interrupts refer to a notification, communicated to the controller, by a hardware device or software, on receipt of which controller momentarily stops and responds to the interrupt. Whenever an interrupt occurs the controller completes the execution of the current instruction and starts the execution of an **Interrupt Service Routine** (ISR) or Interrupt Handler. **ISR** is a piece of code that tells the processor or controller what to do when the interrupt occurs. After the execution of ISR, controller returns back to the instruction it has jumped from (before the interrupt was received). The interrupts can be either **hardware interrupts** or **software interrupts**.

a. Interrupt Sources

The 8051 architecture can handle interrupts from 5 sources. These are: the two external interrupt lines, two timers and the serial interface. Each one of these is assigned an interrupt vector address.

Timer interrupts – Each Timer is associated with a Timer interrupt. A timer interrupt notifies the microcontroller that the corresponding Timer has finished counting.

External interrupts – There are two external interrupts EX0 and EX1 to serve external devices. Both these interrupts are active low. In 8051, P3.2 (INT0) and P3.3 (INT1) pins are available for external interrupts 0 and 1 respectively. An external interrupt notifies the microcontroller that an external device needs its service.

Serial interrupt – This interrupt is used for serial communication. When enabled, it notifies the controller whether a byte has been received or transmitted.

b. How is an interrupt serviced?

Every interrupt is assigned a fixed memory area inside the processor/controller. The Interrupt Vector Table (IVT) holds the starting address of the memory area assigned to it (corresponding to every interrupt).

The interrupt vector table (IVT) for AT89C51 interrupts is as follows :

Interrupt	**Vector Location (Hex)**	**Default priority**	**Source**	**SFR location**
Reset	0000			
External interrupt 0	0003	1	IE0	TCON.1
Timer interrupt 0	000B	2	TF0	TCON.3
External interrupt 1	0013	3	IE1	TCON.5
Timer interrupt 1	001B	4	TF1	TCON.7
Serial COM interrupt	0023	5	RI OR TI	SCON.1,2

When an interrupt is received, the controller stops after executing the current instruction. It transfers the content of program counter into stack. It also stores the current status of the interrupts internally but not on stack. After this, it jumps to the memory location specified by **Interrupt Vector Table** (IVT). After that the code written on that memory area gets executed. This code is known as the Interrupt Service Routine (ISR) or interrupt handler. ISR is a code written by the programmer to handle or service the interrupt.

c. Steps executed by Microcontroller on activation of Interrupts.

Upon activation of an interrupt, the microcontroller goes through the following steps

1. It finishes the instruction it is executing and saves the address of the next instruction (PC) on the stack.
2. It also saves the current status of all the interrupts internally (i.e: not on the stack).

3. It jumps to a fixed location in memory, called the interrupt vector table, that holds the address of the ISR.
4. The microcontroller gets the address of the ISR from the interrupt vector table and jumps to it. It starts to execute the interrupt service subroutine until it reaches the last instruction of the subroutine which is RETI (return from interrupt)
5. Upon executing the RETI instruction, the microcontroller returns to the place where it was interrupted. First, it gets the program counter (PC) address from the stack by popping the top two bytes of the stack into the PC. Then it starts to execute from that address.

Programming Interrupts and SFRs associated with interrupts

At power-up, all interrupts are disabled. Suppose Timer 0 is started. When it times out, TF0 in the special function register TCON will be set. However, this will not cause an interrupt. To enable interrupts, a number of steps need to be taken. There is an interrupt enable special function register IE at byte address A8H. This register is bit addressable. (The assembler gives special mnemonics to each bit address.)

d. IE (Interrupt Enable) Register

This register is responsible for enabling and disabling the interrupt. EA register is set to one for enabling interrupts and set to 0 for disabling the interrupts. Its bit sequence and their meanings are shown in the following figure.

EA	IE.7	It disables all interrupts. When EA = 0 no interrupt will be acknowledged and EA = 1 enables the interrupt individually.
-	IE.6	Reserved for future use.
-	IE.5	Reserved for future use.
ES	IE.4	Enables/disables serial port interrupt.
ET1	IE.3	Enables/disables timer1 overflow interrupt.
EX1	IE.2	Enables/disables external interrupt1.
ET0	IE.1	Enables/disables timer0 overflow interrupt.
EX0	IE.0	Enables/disables external interrupt0.

Example 11-1

Show the instructions to (a) enable the serial interrupt, Timer 0 interrupt, and external hardware interrupt 1 (EX1), and (b) disable (mask) the Timer 0 interrupt, then (c) show how to disable all the interrupts with a single instruction.

Solution:

```
(a)   MOV IE,#10010110B   ;enable serial, Timer 0, EX1
```

Since IE is a bit-addressable register, we can use the following instructions to access individual bits of the register.

```
(b)   CLR IE.1        ;mask(disable) Timer 0 interrupt only
(c)   CLR IE.7        ;disable all interrupts
```

Another way to perform the "MOV IE,#10010110B" instruction is by using single-bit instructions as shown below.

```
SETB IE.7       ;EA=1, Global enable
SETB IE.4       ;enable serial interrupt
SETB IE.1       ;enable Timer 0 interrupt
SETB IE.2       ;enable EX1
```

e. IP (Interrupt Priority) Register

The default priority of interrupts are shown in the interrupt vector table. We can change the priority levels of the interrupts by changing the corresponding bit in the Interrupt Priority (IP) register as shown in the following figure.

- A low priority interrupt can only be interrupted by the high priority interrupt, but not interrupted by another low priority interrupt.
- If two interrupts of different priority levels are received simultaneously, the request of higher priority level is served.
- If the requests of the same priority levels are received simultaneously, then the internal polling sequence determines which request is to be serviced.

-	-	PT2	PS	PT1	PX1	PT0	PX0
bit7	bit6	bit5	bit4	bit3	bit2	bit1	

-	IP.6	Reserved for future use.
-	IP.5	Reserved for future use.
PS	IP.4	It defines the serial port interrupt priority level.
PT1	IP.3	It defines the timer interrupt of 1 priority.
PX1	IP.2	It defines the external interrupt priority level.
PT0	IP.1	It defines the timer0 interrupt priority level.
PX0	IP.0	It defines the external interrupt of 0 priority level.

For example, IP = 0x08; will make Timer1 priority higher. So the interrupt priority order will change as follows (in descending order):

IP = 0x08	
Default Priority	**Changed Priority**
EX0	ET1
ET0	EX0
EX1	ET0
ET1	EX1
ES	ES
ET2	ET2

More than one bit in IP register can also be set. In such a case, the higher priority interrupts will follow the sequence as they follow in default case.

For example, IP = 0x0A; will make Timer0 and Timer1 priorities higher. So the interrupt priority order will change as follows (in descending order):

IP = 0x0A	
Default Priority	**Changed Priority**
EX0	ET0
ET0	ET1
EX1	EX0
ET1	EX1
ES	ES
ET2	ET2

Note that the Timer 0 and 1 have been assigned higher priorities in the same sequence as they follow in default case.

TCON register and External interrupts in 8051

- 8051 has two external interrupt INT0 and INT1.
- 8051 controller can be interrupted by external Interrupt, by providing level or edge on external interrupt pins PORT3.2, PORT3.3.
- External peripherals can interrupt the microcontroller through these external interrupts if global and external interrupts are enabled.

- Then the microcontroller will execute current instruction and jump to the Interrupt Service Routine (ISR) to serve to interrupt.
- In polling method microcontroller has to continuously check for a pulse by monitoring pin, whereas, in interrupt method, the microcontroller does not need to poll. Whenever an interrupt occurs microcontroller serves the interrupt request.

External interrupt has two types of activation level

1. Edge triggered (Interrupt occur on rising/falling edge detection)
2. Level triggered (Interrupt occur on high/low-level detection)

In 8051, two types of activation level are used. These are,

Low level triggered

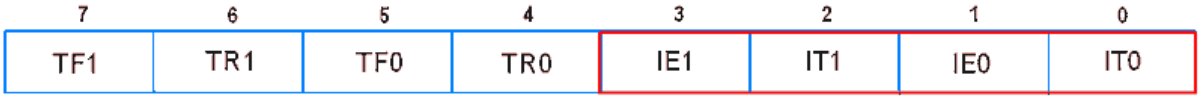
Whenever a low level is detected on the INT0/INT1 pin while global and external interrupts are enabled, the controller jumps to interrupt service routine (ISR) to serve interrupt.

Falling edge triggered

Whenever falling edge is detected on the INT0/INT1 pin while global and ext. interrupts are enabled, the controller jumps to interrupt service routine (ISR) to serve interrupt.

There are lower four flag bits in **TCON register** required to select and monitor the external interrupt type and ISR status.

f. TCON: Timer/ counter Register



Bit 3- IE1:

External Interrupt 1 edge flag, set by hardware when interrupt on INT1 pin occurred and cleared by hardware when interrupt get processed.

Bit 2- IT1:

This bit selects external interrupt event type on INT1 pin,

1= sets interrupt on falling edge

0= sets interrupt on low level

Bit 1- IE0:

Interrupt0 edge flag, set by hardware when interrupt on INT0 pin occurred and cleared by hardware when an interrupt is processed.

Bit 0 - IT0:

This bit selects external interrupt event type on INT0 pin.

1= sets interrupt on falling edge

0= sets interrupt on low level

g. Example Programs for External interrupts

1. **Write a program to clear P2.7 whenever INT0 occurs**

```
ORG 0000H
LJMP MAIN
ORG 0003H
CLR P2.7
RETI
ORG 0030H
MAIN: MOV IE,#81H
L1: SJMP L1
END
```

2. **CLEAR P2.7 WHEN INTERRUPT0 OCCURS AND SET P2.7 WHEN INTERRUPT1 OCCURS**

- ORG 0000H
- LJMP MAIN
- ORG 0003H
- CLR P2.7
- RETI
- ORG 0013H
- SETB P2.7

RETI

- ORG 0030H
- MOV IE,#85H
- L1. SJMP L1
- END

3. **Assume that the INT1 pin is connected to a switch that is normally high. Whenever it goes low, it should turn on an LED. The LED is connected to P1.3 and is normally off. When it is turned on it should stay on for a fraction of a second. As long as the switch is pressed low, the LED should stay on.**

```
01  ORG 0000H
02  LJMP MAIN                       ;bypass interrupt vector table
03
04  ;ISR for hardware interrupt INT1 to turn on the LED
05  ORG 0013H                       ;INT1 ISR
06  SETB P1.3                       ;turn on LED
07  MOV R3,#255                     ;load counter
08  BACK: DJNZ R3,BACK              ;keep LED on for a while
09  CLR P1.3                        ;turn off the LED
10  RETI                            ;return from ISR
11
12  ;MAIN program for initialization
13  ORG 30H
14  MAIN: MOV IE,#10000100B         ;enable external INTI
15  HERE: SJMP HERE                 ;stay here until interrupted
16
17  END
```

4. **Write a program to switch on an LED connected to P3.0 for 500ms whenever INT0 occurs**

   500 ms delay count calculation

   Since 500ms is not possible with 16bit count, we will generate a delay of 50ms and call it 10 times
   For 50ms

   (65535-count+1) X 1.085 us = 50ms
   65536-count = 50000/1.085 = 46082.9 = 19453 = 48FD H

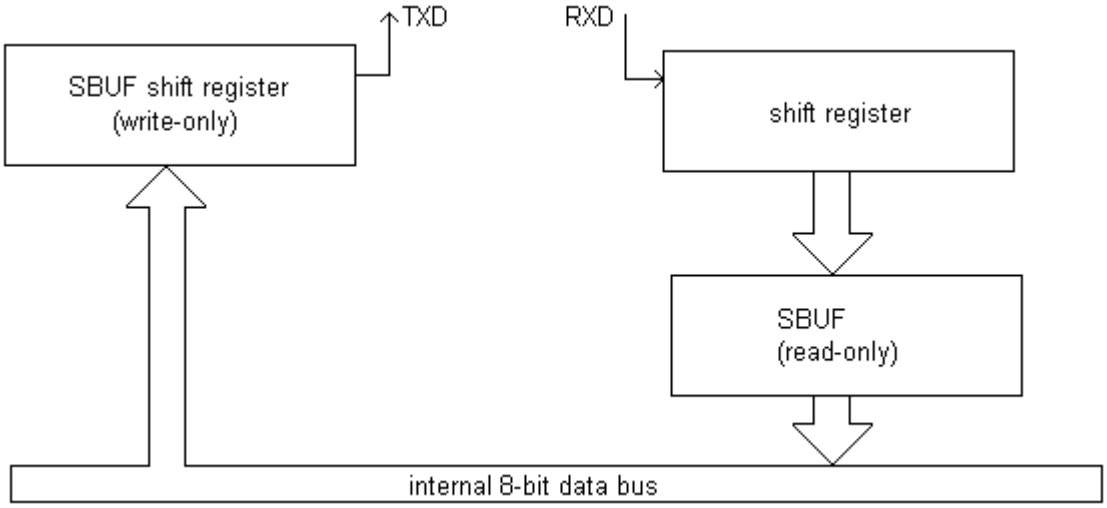
```
ORG 0000H
SJMP MAIN
ORG 0003H
SETB P3.0
MOV R7,#0AH
L1: ACALL DELAY
DJNZ R7,L1
CLR P3.0
RETI
DELAY: MOV TMOD,#01H
MOV TL0,#0FDH
MOV TH0,#48H
SETB TR0
H1: JNB TF0,H1
CLR TR0
CLR TF0
RET
ORG 0030H
MAIN: MOV IE,#82H
L2: SJMP L2
END
```

3.3 Serial communication - SFRs: SCON , SBUF , PCON, Modes of serial communication. Simple programs on serial communication

a. Serial Port

One of the 8051's many powerful features is it's integrated *UART*, otherwise known as a serial port. The fact that the 8051 has an integrated serial port means that you may very easily read and write values to the serial port. We simply need to configure the serial port's operation mode and baud rate. Once configured, all we have to do is write to an SFR to write a value to the serial port or read the same SFR to read a value from the serial port. The 8051 will automatically let us know when it has finished sending the character we wrote and will also let us know whenever it has received a byte so that we can process it. We do not have to worry about transmission at the bit level--which saves us quite a bit of coding and processing time.

b. SBUF register:



SBUF is an **8-bit register** used for serial communication.

- For a byte data to be transferred via the **TxD line**, it must be placed in the **SBUF register**.
- The moment a byte is written into SBUF, it is framed with the start and stop bits and transferred serially via the TxD line.
- SBUF holds the byte of data when it is received by 8051 **RxD** line.
- When the bits are received serially via RxD, the **8051 deframes** it by eliminating the stop and start bits, making a byte out of the data received, and then placing it in SBUF.
- SBUF is physically two registers . one is write only and is used to hold data to be transmitted out of the 8051 via TXD. The other is read only and holds received data from external sources via RXD. Both mutually exclusive registers use address 99h.

For a byte data to be transferred via the TxD line, it must be placed in the SBUF register using instructions

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

A serially received data can be read from SBUF register using the instruction

```
MOV A,SBUF           ;copy SBUF into accumulator.
```

c. PCON register and serial communication

- **PCON register** is an 8-bit SFR.
- It is **byte addressable** register.

SMOD: double baud rate bit. When 8051 is powered up, SMOD bit is at zero value. To double the baud rate SMOD to be set to 1.

Structure of PCON Register

7	6	5	4	3	2	1	0
SMOD	-	-	-	GF1	GF0	PD	IDLE

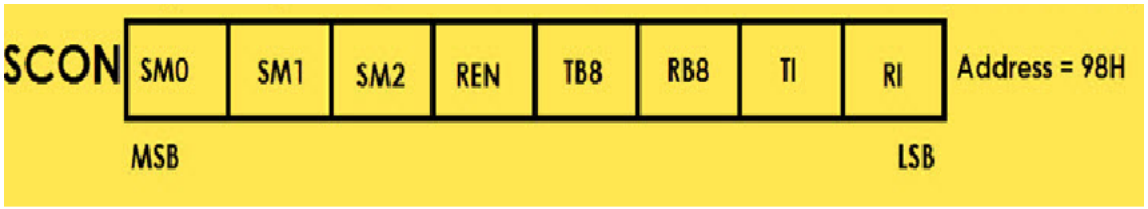
Description

Bit	Symbol	Function
7	SMOD	Serial baud rate modify bit, Set SMOD=1; Baud rate will be doubled SMOD=0; Baud rate will remain as it is
6-4	-	Not implemented
3	GF1	General purpose user flag bit 1
2	GF0	General purpose user flag bit 0
1	PD	PD=1;Microcontroller enters into power down mode
0	IDLE	IDLE=1;Microcontroller enters into IDLE power saving mode

d. Setting the Serial Port Mode : SCON Register

SCON register is used to configure the serial port of 8051.This register is used to configure the data bits we want, the baud rate we will be using, and how the baud rate will be determined.

Format of SCON register:



Bit	Name	Bit Addres	Explanation of Function
7	SM0	9Fh	Serial port mode bit 0, as per table below
6	SM1	9Eh	Serial port mode bit 1, as per table below
5	SM2	9Dh	Mutliprocessor Communications Enable
4	REN	9Ch	Receiver Enable. This bit must be set in order to receive characters.
3	TB8	9Bh	Transmit bit 8. The 9th bit to transmit in mode 2 and 3.
2	RB8	9Ah	Receive bit 8. The 9th bit received in mode 2 and 3.
1	TI	99h	Transmit Flag. Set when a byte has been completely transmitted.
0	RI	98h	Receive Flag. Set when a byte has been completely received.

SM0 and SM1:

SM0	SM1	Mode	Operation	Baud Rate
0	0	Mode 0	Half Duplex Synchronous Operation. Data is sent and received (not simultaneously) using the RxD pin. The TxD pin carries “the shift clock” during both receiving and transmitting.	Fosc/12
0	1	Mode 1	10 bits are transmitted on TxD or received on RxD. Start bit, 8 data bits, 1 stop bit	Variable, set by TIMER 1 overflow bit
1	0	Mode 2	11 bits are transmitted: The start bit, The 8 data bits from SBUF, A 9th data bit from TB8, The stop bit	Fosc/32 if SMOD = 1 Fosc/64 if SMOD = 0
1	1	Mode 3	11 bits are transmitted: The start bit, The 8 data bits from SBUF, A 9th data bit from TB8, The stop bit	Variable, set by TIMER 1 overflow bit

Note: The baud rate indicated in this table is doubled if PCON.7 (SMOD) is set.

Explanation of SCON

The first four bits (bits 4 through 7) are configuration bits.

Bits **SM0** and **SM1** let us set the *serial mode* to a value between 0 and 3, as in table above. Selecting the Serial mode also determines how the baud rate will be calculated. In modes 0 and 2 the baud rate is fixed based on the oscillator’s frequency. In modes 1 and 3 the baud rate is variable based on how often Timer 1 overflows.

The next bit, **SM2**, is a flag for "Multiprocessor communication." Setting this bit enables multi-processor communication.
In multiprocessor communication the serial port of a number of microcontrollers can be connected to a common serial bus. One controller will act as a master and all other controller will act as slave. •A unique 8-bit address is assigned to each slave and the SM2 bit in all the slaves is set to 1.

The next bit, **REN**, is "Receiver Enable." This bit is set to receive data via the serial port, set this bit.

The last four bits (bits 0 through 3) are operational bits. They are used when actually sending and receiving data--they are not used to configure the serial port.

The **TB8** bit is used in modes 2 and 3. In modes 2 and 3, a total of nine data bits are transmitted. The first 8 data bits are the 8 bits of the main value, and the ninth bit is taken from TB8.

The **RB8** also operates in modes 2 and 3 and functions essentially the same way as TB8, but on the reception side. When a byte is received in modes 2 or 3, a total of nine bits are received. In this case, the first eight bits received are placed in SBUF and the value of the ninth bit received will be placed in RB8.

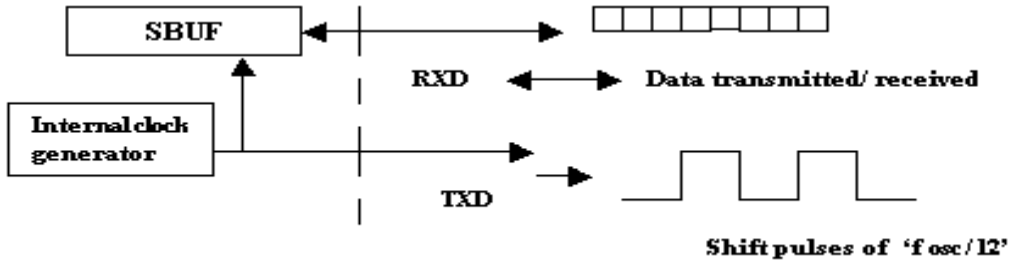
TI means "Transmit Interrupt." When a program writes a value to the serial port, a certain amount of time will pass before the individual bits of the byte are "clocked out" the serial port. If the program were to write another byte to the serial port before the first byte was completely output, the data being sent would be garbled. Thus, the 8051 lets the program know that it has "clocked out" the last byte by setting the TI bit. When the TI bit is set, the program may assume that the serial port is "free" and ready to send the next byte.

Finally, the **RI** bit means "Receive Interrupt." It functions similarly to the "TI" bit, but it indicates that a byte has been received. That is to say, whenever the 8051 has received a complete byte it will trigger the RI bit to let the program know that it needs to read the value quickly, before another byte is read.

Serial Data Transmission Modes:

Mode-0:

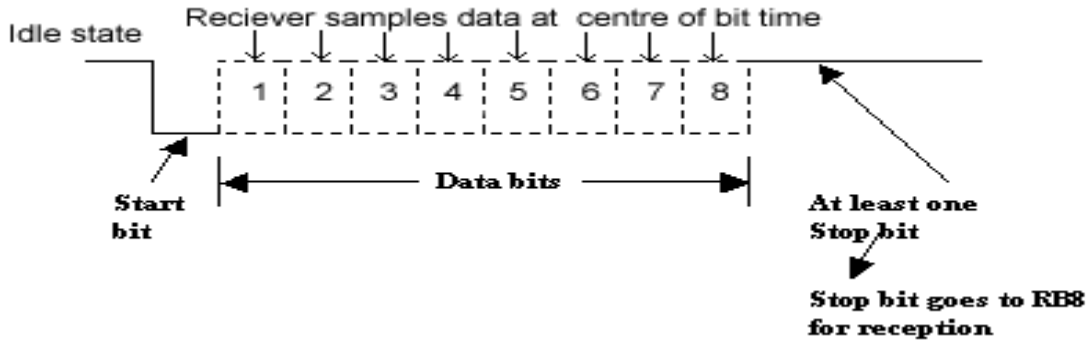
- This mode is configured when SM0 and SM1 bits in SCON register are 0 and 0.
- In this mode, the serial port works like a shift register and the data transmission works synchronously with a clock frequency of fosc /12.
- Serial data is received and transmitted through RXD. 8 bits are transmitted/ received at a time.
- Pin TXD outputs the shift clock pulses of frequency fosc /12, which is connected to the external circuitry for synchronization.
- The shift frequency or baud rate is always 1/12 of the oscillator frequency.



Mode-1 (standard UART mode) :

- This mode is configured when SM0 and SM1 bits in SCON register are 0 and 1.
- In mode-1, the serial port functions as a standard Universal Asynchronous Receiver Transmitter (UART) mode.
- 10 bits are transmitted through TXD or received through RXD. The 10 bits consist of one start bit (which is usually '0'), 8 data bits (LSB is sent first/received first), and a stop bit (which is usually '1').
- TI flag is set after transmission of one byte.
- The baud rate is variable.

The following figure shows the way the bits are transmitted/ received.



Bit time= $1/f_{baud}$

- In receiving mode, data bits are shifted into the receiver at the programmed baud rate. The data word (8-bits) will be loaded to SBUF if the following conditions are true.
- RI must be zero. (i.e., the previously received byte has been cleared from SBUF)
- After the data is received and the data byte has been loaded into SBUF, RI becomes one.

:**Mode-1 baud rate generation**

Timer-1 is used to generate baud rate for mode-1 serial communication by using overflow flag of the timer to determine the baud frequency. Timer-1 is used in timer mode-2 as an auto-reload 8-bit timer. The data rate is generated by timer-1 using the following formula.

$$f_{baud} = \frac{2^{SMOD}}{32} X \frac{fosc}{12 X [256-(TH1)]}$$

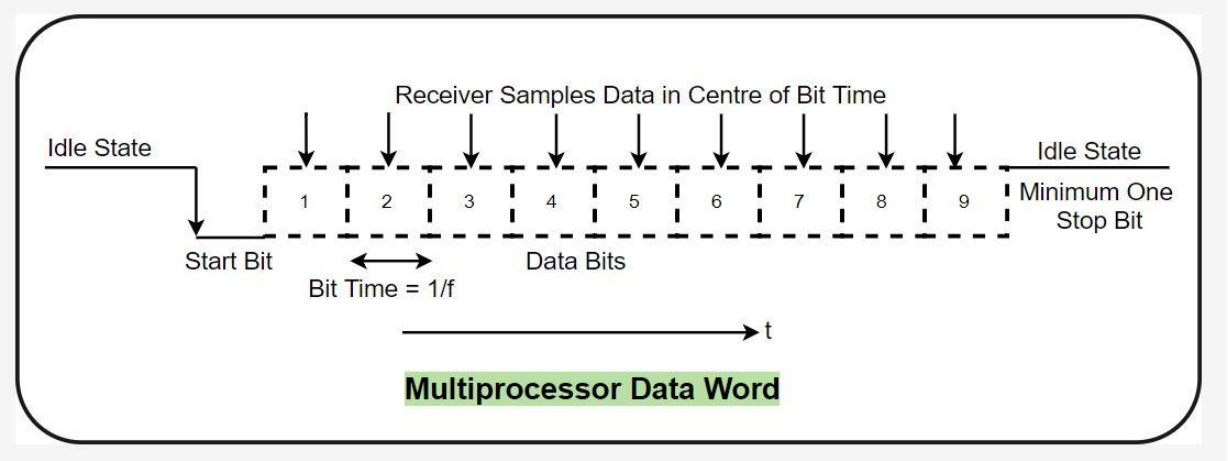
Where,

SMOD is the 7th bit of PCON register
fosc is the crystal oscillator frequency of the microcontroller

It can be noted that fosc/ (12 X [256- (TH1)]) is the timer overflow frequency in timer mode-2, which is the auto-reload mode.

Mode-2 –9 Bit UART with fixed frequency :

- This mode is configured when SM0 and SM1 bits in SCON register are 1 and 0.
- In this mode 11 bits are transmitted through TXD or received through RXD.
- The various bits are as follows: a start bit (usually '0'), 8 data bits (LSB first), a programmable 9 th (TB8 or RB8)bit and a stop bit (usually '1').
- While transmitting, the 9 th data bit (TB8 in SCON) can be assigned the value '0' or '1'. For example, if the information of parity is to be transmitted, the parity bit (P) in PSW could be moved into TB8.
- TI flag is set after transmission of one data byte
- On reception of the data, the 9 th bit goes into RB8 in 'SCON', while the stop bit is ignored.
- RI flag is set , once a data byte is received
- The baud rate is programmable to either 1/32 or 1/64 of the oscillator frequency. $f_{baud} = (2^{SMOD}/64)$ fosc.



Mode-3 –9 bit UART mode with variable baud rate :

- This mode is configured when SM0 and SM1 bits in SCON register are 1 and 1.
- In this mode 11 bits are transmitted through TXD or received through RXD.
- The various bits are as follows: a start bit (usually '0'), 8 data bits (LSB first), a programmable 9 th (TB8 or RB8)bit and a stop bit (usually '1').
- While transmitting, the 9 th data bit (TB8 in SCON) can be assigned the value '0' or '1'. For example, if the information of parity is to be transmitted, the parity bit (P) in PSW could be moved into TB8.
- TI flag is set after transmission of one data byte
- On reception of the data, the 9 th bit goes into RB8 in 'SCON', while the stop bit is ignored.
- RI flag is set , once a data byte is received
- Mode-3 is same as mode-2, except the fact that the baud rate in mode-3 is variable (i.e., just as in mode-1).
- $f_{baud} = (2^{SMOD}/32) * (fosc / 12 (256-TH1))$. This baud rate holds when Timer-1 is programmed in Mode-3.

e. Programming:

Writing to the Serial Port (only for understanding)

Once the Serial Port has been propertly configured as explained above, the serial port is ready to be used to send data and receive data. To write a byte to the serial port one must simply write the value to the **SBUF** (99h) SFR. For example, if you wanted to send the letter "A" to the serial port, it could be accomplished as easily as:

MOV SBUF,#'A'

Upon execution of the above instruction the 8051 will begin transmitting the character via the serial port. Obviously transmission is not instantaneous--it takes a measureable amount of time to transmit. And since the 8051 does not have a serial output buffer we need to be sure that a character is completely transmitted before we try to transmit the next character.

The 8051 lets us know when it is done transmitting a character by setting the **TI** bit in SCON. When this bit is set we know that the last character has been transmitted and that we may send the next character, if any. Consider the following code segment:

CLR TI ;Be sure the bit is initially clear
MOV SBUF,#'A' ;Send the letter 'A' to the serial port
JNB TI,$;Pause until the RI bit is set.

The above three instructions will successfully transmit a character and wait for the TI bit to be set before continuing. The last instruction says "Jump if the TI bit is not set to $"--$, in most assemblers, means "the same address of the current instruction." Thus the 8051 will pause on the JNB instruction until the TI bit is set by the 8051 upon successful transmission of the character.

Reading the Serial Port (only for understanding)

Reading data received by the serial port is equally easy. To read a byte from the serial port one just needs to read the value stored in the **SBUF** (99h) SFR after the 8051 has automatically set the **RI** flag in SCON.

For example, if your program wants to wait for a character to be received and subsequently read it into the Accumulator, the following code segment may be used:

JNB RI,$;Wait for the 8051 to set the RI flag
MOV A,SBUF ;Read the character from the serial port

The first line of the above code segment waits for the 8051 to set the RI flag; again, the 8051 sets the RI flag automatically when it receives a character via the serial port. So as long as the bit is not set the program repeats the "JNB" instruction continuously.

Once the RI bit is set upon character reception the above condition automatically fails and program flow falls through to the "MOV" instruction which reads the value.

Steps to Transmit a byte

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. The TH1 is loaded with one of the values to set 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 to start timer 1
5. TI is cleared by CLR TI instruction
6. The character byte to be transferred serially is written into SBUF register
7. The TI flag bit is monitored with the use of instruction JNB TI,xx to see if the character has been transferred completely
8. To transfer the next byte, go to step 5

f. Example Programs:

1. Write a program for the 8051 to transfer letter "A" serially at 4800 baud, continuously.
Count calculation for 4800 baud
The final formula for baud rate is as below. Baudrate = Fosc/(32 * 12 * (256-count))
Now with Fosc = 11.0592Mhz, TH1 value for 4800 baudrate will be
$4800 = 11.0592 \text{ X } 10^6/(32 * 12 * (256\text{-count}))$
$256\text{-count} = 11.0592 \text{ X } 10^6/(32 * 12 * 4800) = 6$
Count =256-6 = 250 = FA H
Solution:

```
MOV TMOD,#20H          ;timer 1,mode 2(auto reload)
MOV TH1,#FAH           ;4800 baud rate
MOV TL1,#FAH
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
```

2. Write a program for the 8051 to transfer "YES" serially at 9600 baud, 8-bit data, 1 stop bit, do this continuously
Solution:
Count calculation for 9600 baud
The final formula for baud rate is as below. Baudrate = Fosc/(32 * 12 * (256-count))
Now with Fosc = 11.0592Mhz, count value for 9600 baudrate will be
$9600 = 11.0592 \text{ X } 10^6/(32 * 12 * (256\text{-count}))$
$256\text{-count} = 11.0592 \text{ X } 10^6/(32 * 12 * 4800) = 3$
Count =256-3 = 253 = FD H

```
MOV TMOD,#20H          ;timer 1,mode 2(auto reload)
MOV TH1,#0FDH          ;9600 baud rate
MOV TL1,#0FDH
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
```

Receiving:
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

3. 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

Solution:

Count calculation for 4800 baud

The final formula for baud rate is as below. Baudrate = Fosc/(32 * 12 * (256-count))
Now with Fosc = 11.0592Mhz, TH1 value for 4800 baudrate will be
$4800 = 11.0592 \text{ X } 10^6/(32 * 12 * (256\text{-count}))$
$256\text{-count} = 11.0592 \text{ X } 10^6/(32 * 12 * 4800) = 6$
Count =256-6 = 250 = FA H

```
MOV TMOD,#20H       ;timer 1,mode 2(auto reload)
MOV TH1,#0FAH           ;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
```

In the 8051 there is only one interrupt set aside for serial communication. This interrupt is used to both send and receive data. If the interrupt bit in the IE register (IE.4) is enabled, when RI or TI is raised the 8051 gets interrupted and jumps to memory location 0023H to execute the ISR. In that ISR we must examine the TI and RI flags to see which one caused the interrupt and respond accordingly.

Programming with interrupts:

Write a program in which the 8051 reads data from P1 and writes it to
P2 continuously while giving a copy of it to the serial COM port to be transferred serially. Assume that XTAL=11.0592. Set the baud rate at
9600.

Solution:

```
ORG 0000H
LJMP MAIN
ORG 0023H
LJMP SERIAL       ;jump to serial int ISR
ORG 0030H
MAIN: MOV P1,#0FFH ;make P1 an input port
MOV TMOD,#20H     ;timer 1, auto reload
MOV TH1,#0FDH     ;9600 baud rate
MOV SCON,#50H    ;8-bit,1 stop, ren enabled
MOV IE,10010000B   ;enable serial int.
SETB TR1          ;start timer 1
BACK: MOV A,P1     ;read data from port 1
```

```
MOV SBUF,A      ;give a copy to SBUF
MOV P2,A        ;send it to P2
SJMP BACK       ;stay in loop indefinitely
```

4.1 Parallel Ports of 8051

The basic 8051 microcontroller has four 8-bit ports, P0, P1, P2 and P3. The addresses of these ports are 80H, 90H, A0H and B0H respectively. All of the ports, except P1, have alternate functions.

All pins on all the ports (32 pins) can be used as either inputs or outputs. These ports are bit addressable. Each port has a D-type output latch for each pin. The SFR for each port is made up of these eight latches, which can be addressed at the SFR address for that port. For instance, the eight latches for port 0 are addressed at location 80h; port 0 pin 3 is bit 2 of the PO SFR. The port latches should not be confused with the port pins; the data on the latches does *not* have to be the same as that on the pins.

Port Structure

Each port of 8051 has bidirectional capability. Port 0 is called 'true bidirectional port' as it floats (tristated) when configured as input. Port-1, 2, 3 are called 'quasi bidirectional port'.

a. Port-0 Pin Structure

Port0 has 8 pins (P0.0-P0.7).
The structure of a Port-0 pin is shown in fig below

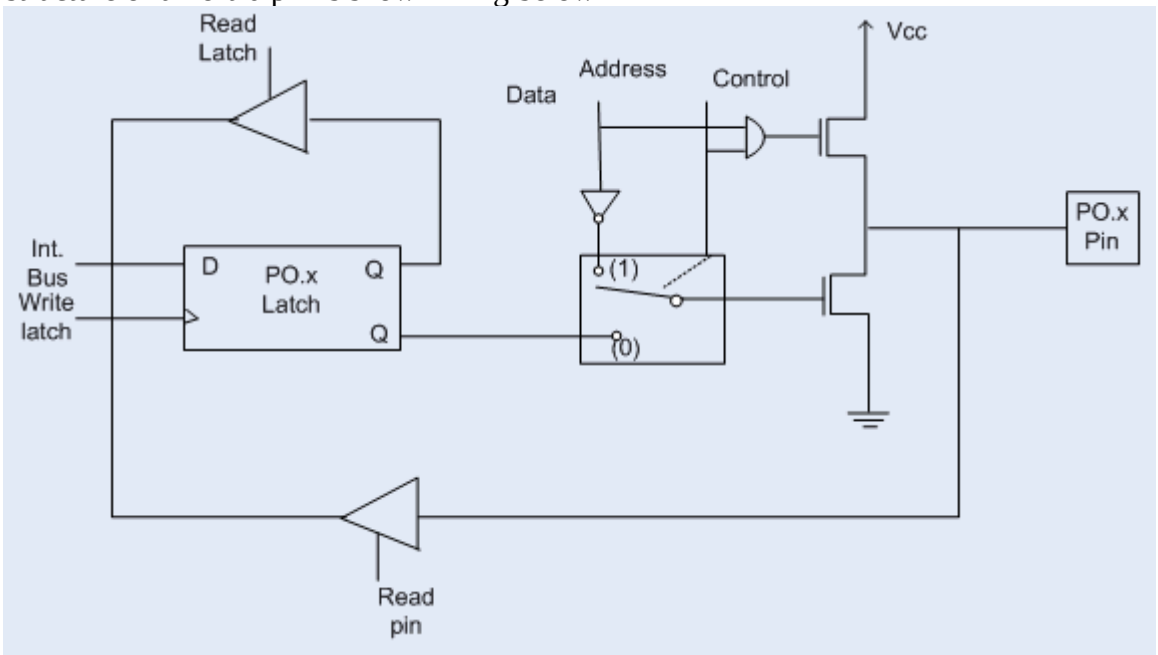


Fig 1: Port-0 Structure

Port 0: Port 0 occupies a total of 8 pins (pins 32-39) .It can be used for input or output. To use the pins of port 0 as both input and output ports, each pin must be connected externally to a 10K ohm pull-up resistor. This is due to the fact that P0 is an open drain, unlike P1, P2, and P3.Open drain is a term used for MOS chips in the same way that open collector is used for TTL chips. With external pull-up resistors connected upon reset, port 0 is configured as an output port. For

example, the following code will continuously send out to port 0 the alternating values 55H and AAH

```
MOV A,#55H
BACK:  MOV P0,A
ACALL DELAY
CPL A
SJMP BACK
```

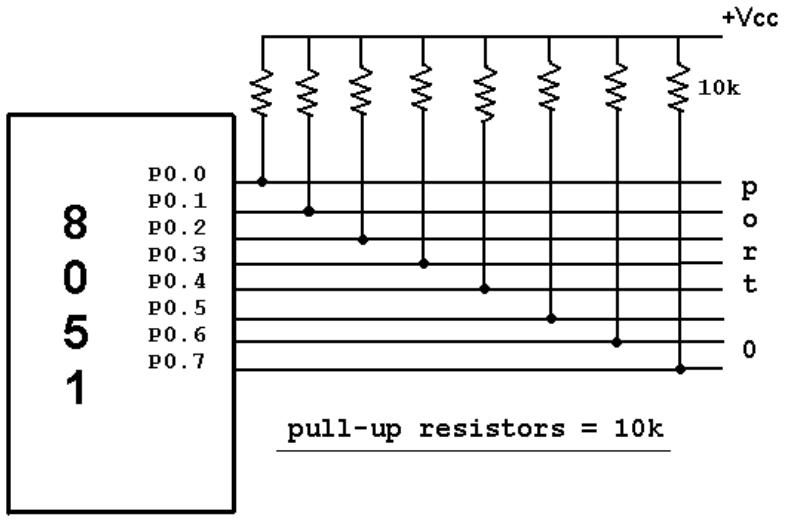


Fig 2: External Pull up

Port 0 as Input : With resistors connected to port 0, in order to make it an input, the port must be programmed by writing 1 to all the bits. In the following code, port 0 is configured first as an input port by writing 1′s to it, and then data is received from the port and sent to P1.

```
MOV A,#0FFH       ; A = FF hex
MOV P0,A          ; make P0 an input port
BACK: MOV A,P0          ;get data from P0
MOV P1,A          ;send it to port 1
SJMP BACK
```

Dual role of port 0: Port 0 is also designated as AD0-AD7, allowing it to be used for both address and data. When connecting an 8051/31 to an external memory, port 0 provides both address and data. The 8051 multiplexes address and data through port 0 to save pins. ALE indicates if P0 has address or data. When ALE = 0, it provides data D0-D7, but when ALE =1 it has address and data with the help of a 74LS373 latch

As shown in Fig 1, When control is '1', the port is used for address/data interfacing. When the control is '0', the port can be used as a normal bidirectional I/O port.

Let us assume that control is '0'. When the port is used as an input port, '1' is written to the latch. In this situation both the output MOSFETs are 'off'. Hence the output pin floats. This high impedance pin can be pulled up or low by an external source. When the port is used as an output port, a '1' written to the latch again turns 'off' both the output MOSFETs and causes the output pin to float. An

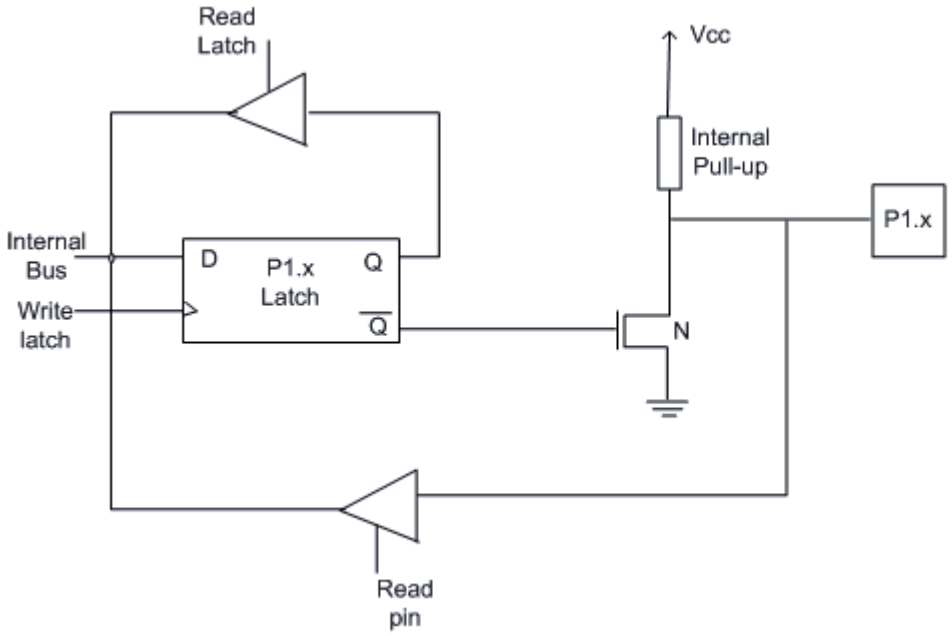
external pull-up is required to output a '1'. But when '0' is written to the latch, the pin is pulled down by the lower MOSFET. Hence the output becomes zero.

When the control is '1', address/data bus controls the output driver MOSFETs. If the address/data bus (internal) is '0', the upper MOSFET is 'off' and the lower MOSFET is 'on'. The output becomes '0'. If the address/data bus is '1', the upper transistor is 'on' and the lower transistor is 'off'. Hence the output is '1'. Hence for normal address/data interfacing (for external memory access) no pull-up resistors are required.

Port-0 latch is written to with 1's when used for external memory access.

b. Port-1 Pin Structure

Port-1 has 8 pins (P1.1-P1.7) .The structure of a port-1 pin is shown in figure below.



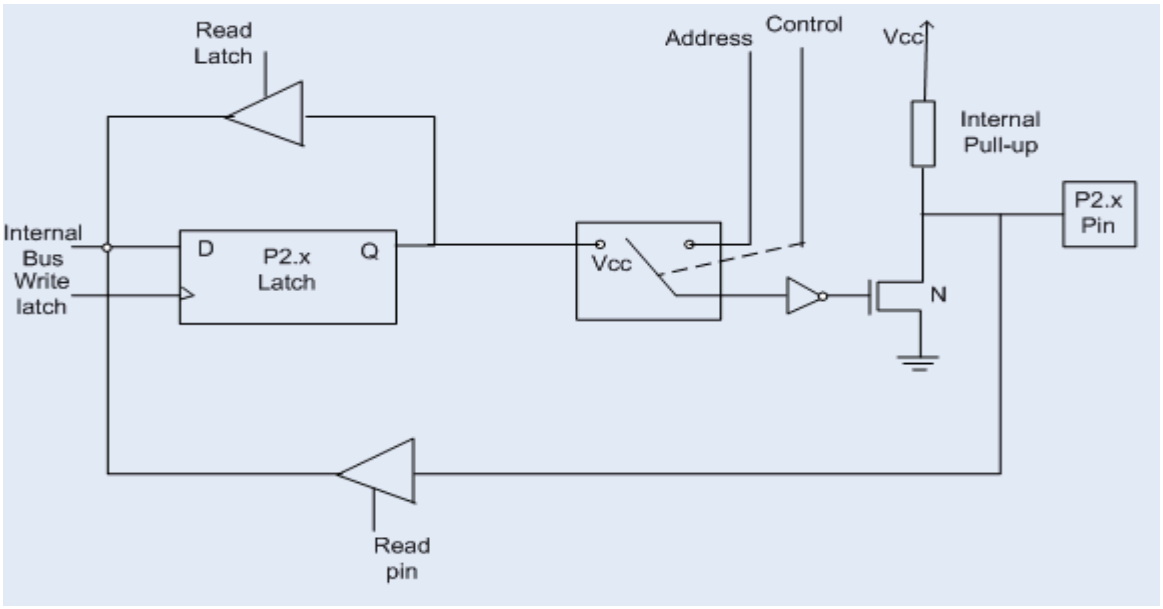
Port-1 does not have any alternate function i.e. it is dedicated solely for I/O interfacing.

To use port-1 as input port, '1' has to be written to the latch. When '0' is written to the pin by the external device then the external source must sink current due to internal pull-up. If the external device is not able to sink the current the pin voltage may rise, leading to a possible wrong reading.

Suppose we want to write 1 on pin of Port 1, a '1' written to the latch which turns 'off' the FET . Due to Vcc connection through internal pull up resistor, the port pin will be at logic 1.When a 0 is written to the latch, the FET will be turned ON and pin will be at Logic 0.

c. PORT 2 Pin Structure

Port-2 has 8-pins (P2.0-P2.7) . The structure of a port-2 pin is shown in fig below.



Port-2 is used for higher external address byte or a normal input/output port.

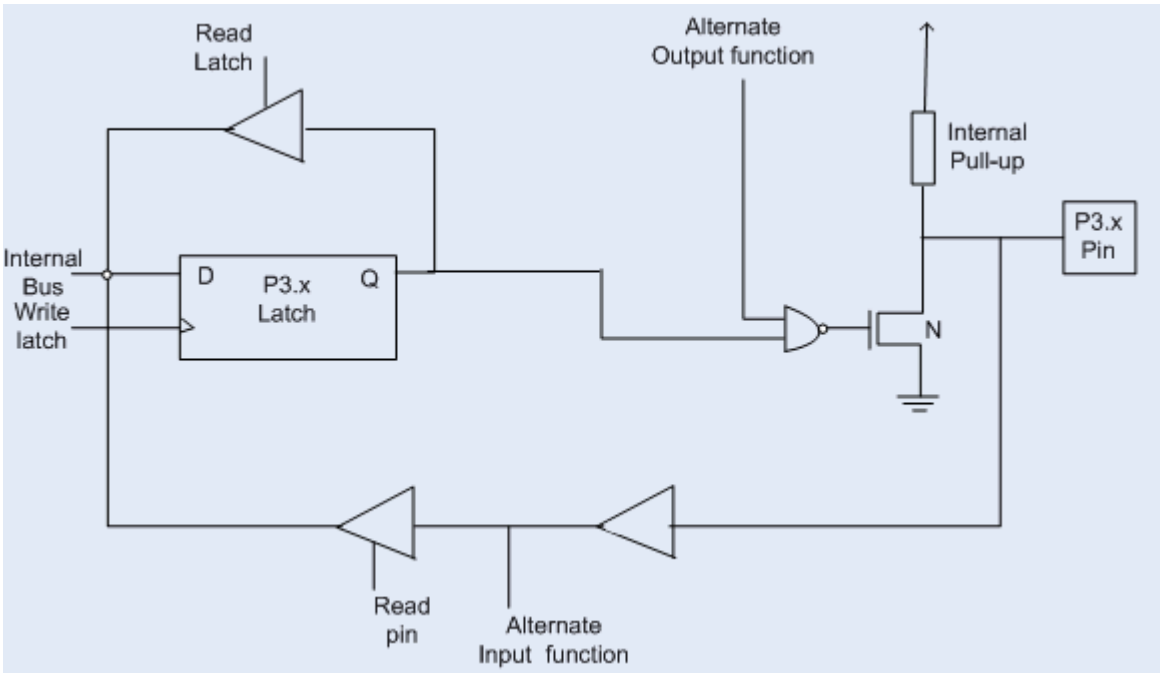
When control bit is logic 0, I/O operation takes place. The I/O operation is similar to Port-1. To use port-2 as input port, '1' has to be written to the latch. When '0' is written to the pin by the external device then the external source must sink current due to internal pull-up. If the external device is not able to sink the current the pin voltage may rise, leading to a possible wrong reading.

Suppose we want to write 1 on pin of Port 2, a '1' written to the latch which turns 'off' the FET . Due to Vcc connection through internal pull up resistor, the port pin will be at logic 1.When a 0 is written to the latch, the FET will be turned ON and pin will be at Logic 0.

When external memory is connected, the control bit will be at logic 1 and port bit is used as higher order address line. Port-2 latch remains stable when Port-2 pin are used for external memory access. Here again due to internal pull-up there is limited current driving capability.

d. PORT 3 Pin Structure

Port-3 has 8 pin (P3.0-P3.7) . Port-3 pins have alternate functions. The structure of a port-3 pin is shown in fig below. Each pin of Port-3 can be individually programmed for I/O operation or for alternate function.



- When used for I/O operation, the alternate output function would be at logic high.
- If you write a logic 0 to the port pin, this Q is logic 0, the output of NAND gate will be logic 1 and this turns on the FET gate. It makes the port pin connected to ground (logic 0).
- If you write a logic 1 is written to the port pin, then Q is 1 , the output of NAND gate will be at logic 0,and this turns off the FET gate. Therefore the pin is at logic 1 because it is connected to high.

Similarly, the alternate function can be activated only if the corresponding latch has been written to '1'.

To use the port as input port, '1' should be written to the latch. This port also has internal pull-up and limited current driving capability.

Alternate functions of Port-3 pins are –

P3.0	RxD
P3.1	TxD
P3.2	INT0
P3.3	INT1
P3.4	T0
P3.5	T1
P3.6	$\overline{WR}$
P3.7	$\overline{RD}$

In addition to acting as a normal I/O port,

- P3.0 can be used for serial receive input pin(RXD)

- P3.1 can be used for serial transmit output pin(TXD) in a serial port,
- P3.2 and P3.3 can be used as external interrupt pins(INT0' and INT1'),
- P3.4 and P3.5 are used for external counter input pins(T0 and T1),
- P3.6 and P3.7 can be used as external data memory write and read control signal pins(WR' and RD')read and write pins for memory access**.**

Initializing Port pins(only for understanding)

Because of the way the 8051 port pin circuitry is designed, to use a port pin as an output requires no initialisation. You simply write to the port. For example, *SETB P1.5* will send a logic 1 to P1 bit 5. *MOV P0, A* will send the data in the accumulator to P0.

To initialise a port pin as an input we must first write a logic 1 to that pin. For example, to set P3 pin 2 as an input we could execute the code: *SETB P3.2*

This code is exactly the same as if you were writing a logic 1 to an output pin. Therefore, simply by examining a line of code, say, *MOV P2, #0FFH*, there is no way of telling if this is sending all 1s to P2 because P2 is an output port, or sending all 1s to P2 to make all of P2's pins inputs. Only by examining what is connected to P2 would we be able to decipher this piece of code correctly. For example, if 8 LEDs are connected to P2, then this is an output port (you never read the value of an LED, you either turn it on or off by writing 1s and 0s to it) and the code *MOV P2, #0FFH* is turning on or off (depending on the way the LEDs are connected) the LEDs.

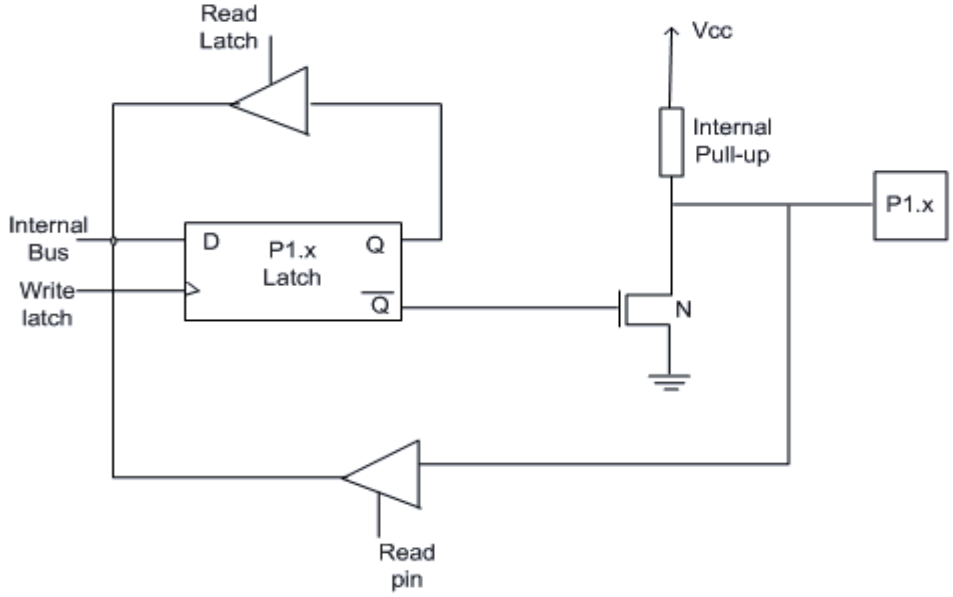
If, on the other hand, 8 switches were connected to P2, then these pins must be used as inputs (because we read the value of a switch) and the code *MOV P2, #0FFH* is initialising each pin of P2 as an input.

It is very important when initialising a port pin on the 8051 to write a comment stating this, otherwise it is difficult to know whether data is being sent out to the port pin, or the port pin is being initialised for input. Taking the above example, the code should be written something like:

MOV P2, #0FFH ; initialising P2, all pins, as inputs

Why must a logic 1 be written to the port pin to make it an input pin?

To understand why writing a 1 to a port pin is necessary if the pin is to be used as an input, take a look at the schematic for an 8051 port pin.



When data is written to the port pin, it first appears on the latch input (D) and is then passed through to the output (Q) and through an inverter to the FET. If a logic 0 is written to the port pin, this 0 on Q is inverted to a logic 1, which turns on the FET. There is now a direct path from the port pin to ground (therefore it is at logic 0, as desired).
However, if this pin is to be used as an input and there is a logic 0 on the latch, then the pin will always be at 0 because it is connected directly to ground through the switched on FET. No matter what voltage is applied to the port pin it will not rise above 0V.
Therefore, to use the port pin as an input we first write a logic 1 to the latch. This logic 1 is then inverted and the FET is switched off. Now there is no connection between the port pin and ground and the voltage applied to the pin can be read through the READ pin buffer.

Reading and Writing to Port Pins (Programming)

Writing to a port pin is very simple, as shown below:

```
SETB P3.5 ; set pin 5 of port 3
MOV P1, #4AH ; sending data 4AH to port 1 - the binary pattern on the port will be 0100 1010
MOV P2, A ; send whatever data is in the accumulator to port 2
```

Reading a port pin is also very simple, once the pin has been initialised for input:

```
SETB P1.0 ; initialise pin 0 of port 1 as an input pin
MOV P2, #FFH ; set all pins of port 2 as inputs
MOV C, P1.0 ; move value on pin 0 of port 1 to the carry
MOV R3, P2 ; move data on port 2 into R3
```

Reading a port (port-pins) versus reading a latch

There is a subtle difference between reading a latch and reading the output port pin.

The status of the output port pin is sometimes dependant on the connected load. For instance if a port is configured as an output port and a '1' is written to the latch, the output pin should also show '1'. If the output is used to drive the base of a transistor, the transistor turns 'on'.

If the port pin is read, the value will be '0' which is corresponding to the base-emitter voltage of the transistor.

Reading a latch: Usually the instructions that read the latch, read a value, possibly change it, and then rewrite it to the latch. These are called "read-modify-write" instructions. Examples of a few instructions are-

ORL P2, A; P2 <-- P2 or A

MOV P2.1, C; Move carry bit to PX.Y bit.

In this the latch value of P2 is read, is modified such that P2.1 is the same as Carry and is then written back to P2 latch.

Reading a Pin: Examples of a few instructions that read port pin, are-

MOV A, P0 ; Move port-0 pin values to A

MOV A, P1; Move port-1 pin values to A