BASICS OF INTERRUPTS.

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

Steps taken by processor while processing an interrupt:

1. *It completes the execution of the current instruction.*
2. *PSW is pushed to stack.*
3. *PC content is pushed to stack.*
4. *Interrupt flag is reset.*
5. *PC is loaded with ISR address.*

ISR will always ends with RETI instruction. The execution of RETI instruction results in the following.

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

Classification of interrupts.

1. ***External and internal interrupts.***
   External interrupts are those initiated by peripheral devices through the external pins of the microcontroller.
   Internal interrupts are those activated by the internal peripherals of the microcontroller like timers, serial controller etc.)
2. ***Maskable and non-maskable interrupts.***
   The category of interrupts which can be disabled by the processor using program is called maskable interrupts.
   Non-maskable interrupts are those category by which the programmer cannot disable it using program.
3. ***Vectored and non-vectored interrupt.***
   Starting address of the ISR is called interrupt vector. In vectored interrupts the starting address is predefined. In non-vectored interrputs, the starting address is provided by the peripheral as follows.
   - Microcontroller receives an interrupt request from external device.
   - Controller sends an acknowledgement (**INTA)** after completing the execution of current instruction.
   - The peripheral device sends the interrupt vector to the microcontroller.

8051 INTERRUPT STRUCTURE.

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

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

8051 makes use of two registers to deal with interrupts.

6. **IE Register**

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

IE : Interrupt Enable Register (Bit Addressable)

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

EA	–	–	ES	ET1	EX1	ET0	EX0

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

7. **IP Register.**

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

IP : Interrupt Priority Register (Bit Addressable)

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

–	–	–	PS	PT1	PX1	PT0	PX0

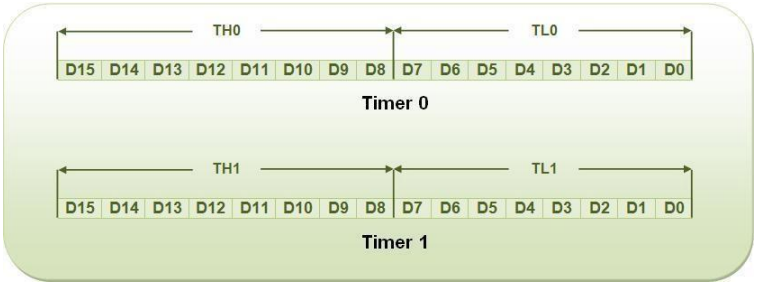
-	IP.7	Not implemented, reserved for future use*.
-	IP.6	Not implemented, reserved for future use*.
-	IP.5	Not implemented, reserved for future use*.
PS	IP.4	Defines the Serial Port interrupt priority level.
PT1	IP.3	Defines the Timer 1 Interrupt priority level.
PX1	IP.2	Defines External Interrupt priority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

TIMERS AND COUNTERS

Timers/Counters are used generally for

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

8051 has two timers, Timer 0 and Timer 1.



Timer in 8051 is used as timer, counter and baud rate generator. Timer always counts up irrespective of whether it is used as timer, counter, or baud rate generator: Timer is always incremented by the microcontroller. The time taken to count one digit up is based on master clock frequency.

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

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

The following are timer related SFRs in 8051.

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

TMOD Register

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

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

GATE	When TRx (in TCON) is set and GATE = 1, TIMER/COUNTERx will run only while INTx pin is high (hardware control). When GATE = 0, TIMER/COUNTERx will run only while TRx = 1 (software control).
C/$\overline{T}$	Timer or Counter selector. Cleared for Timer operation (input from internal system clock). Set for Counter operation (input from Tx input pin).
M1	Mode selector bit (NOTE 1).
M0	Mode selector bit (NOTE 1).

Note 1 :

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

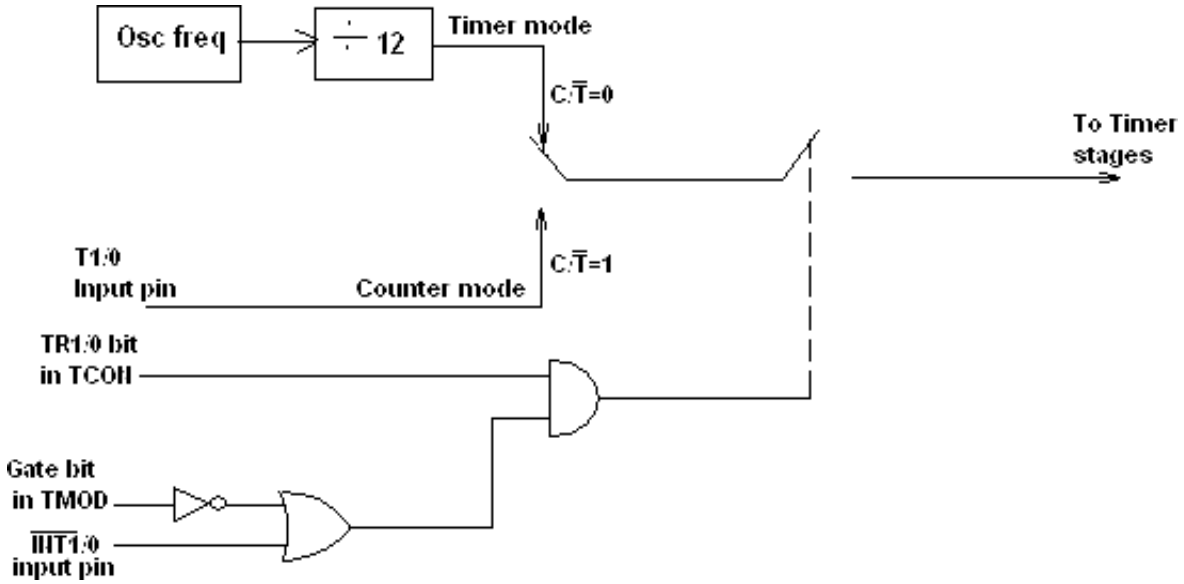
TCON Register

TCON : Timer/Counter Control Register (Bit Addressable)

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

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

Timer/ Counter Control Logic.



TIMER MODES

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

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

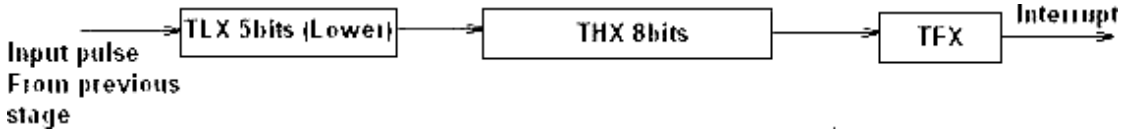


Fig. Operation of Timer on Mode-0

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

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

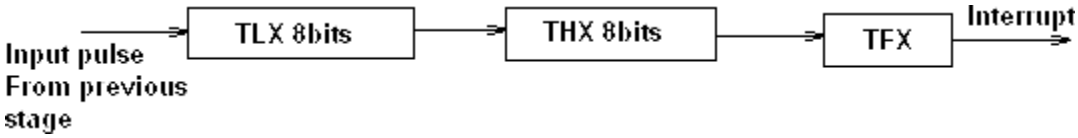


Fig: Operation of Timer in Mode 1

Timer Mode-2: (Auto-Reload Mode): This is a 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.

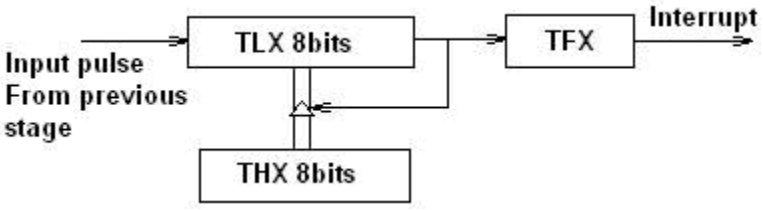


Fig: Operation of Timer in Mode 2

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.

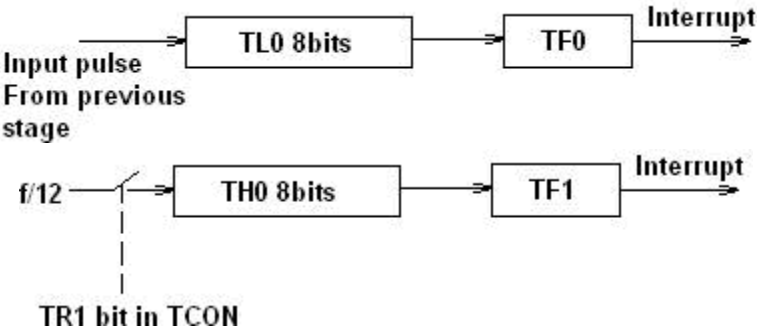


Fig: Operation of Timer in Mode 3

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/Timer Clock Frequency.
= 1/(Master Clock Frequency/12)

a. Mode 1 (16 bit timer/counter)
   Value to be loaded in decimal = 65536 – (Delay Required/Timer clock period)
   Convert the answer into hexadecimal and load onto THx and TLx register.
   ($65536_D = FFFF_H+1$)
b. Mode 0 (13 bit timer/counter)
   Value to be loaded in decimal = 8192 – (Delay Required/Timer clock period)
   Convert the answer into hexadecimal and load onto THx and TLx register.
   ($8192_D = 1FFF_H+1$)
c. Mode 2 (8 bit auto reload)
   Value to be loaded in decimal = 256 – (Delay Required/Timer clock period)
   Convert the answer into hexadecimal and load onto THx register. Upon starting the timer this value from THx will be reloaded to TLx register.
   ($256_D = FF_H+1$)

Steps for programming timers in 8051

Mode 1:

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

Mode 0:

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

Mode 2:

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

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

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

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

2. ***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 = 9CH

```
            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
```

Watchdog Timer

A watchdog timer can be thought of as having the inverse functionality than that of a regular timer. We configure a watchdog timer with a real-time value, just as with a regular timer. However, instead of the timer generating a signal for us every X time units, we must generate a signal for the timer every X time units. If we fail to generate this signal in time, then the timer generates a signal indicating that we failed. We often connect this signal to the reset or interrupt signal of a general-purpose processor. Thus, a watchdog timer provides a mechanism of ensuring that our software is working properly; every so often in the software, we include a statement that generates a signal to the watchdog timer (in particular, that resets the timer). If something undesired happens in the software (e.g., we enter an undesired infinite loop, we wait for an input signal that never arrives, a part fails, etc.), the watchdog generates a signal that we can use to restart or test parts of the system. Using an interrupt service routine, we may record information as to the number of failures and the causes of each, so that a service technician may later evaluate this information to determine if a particular part requires replacement. Note that an embedded system often must recover from failures whenever possible, as the user may not have the means to reboot the system in the same manner that he/she might reboot a desktop system.

Real-Time Clock (RTC)

Much like a digital wristwatch, a real-time clock (RTC) keeps the time and date in an embedded system. Read-time clocks are typically composed of a crystal-controlled oscillator, numerous cascaded counters, and a battery backup. The crystal-controlled oscillator generates a very consistent high-frequency digital pulse that feed the cascaded counters. The first counter, typically, counts these pulses up to the oscillator frequency, which corresponds to exactly one second. At this point, it generates a pulse that feeds the next counter. This counter counts up to 59, at which point it generates a pulse feeding the minute counter. The hour, date, month and year counters work in similar fashion. In addition, real-time clocks adjust for leap years. The rechargeable back-up battery is used to keep the real-time clock running while the system is powered off.

From the micro-controller’s point of view, the content of these counters can be set to a desired value, (this corresponds to setting the clock), and retrieved. Communication between the micro-

controller and a real-time clock is accomplished through a serial bus, such as I^2C. It should be noted that, given a timer peripheral, it is possible to implement a real-time clock in software running on a processor. In fact, many systems use this approach to maintain the time. However, the drawback of such systems is that when the processor is shut down or reset, the time is lost.