

Example

Find the value for TMOD if we want to program timer 0 in mode 2, use 8051 XTAL for the clock source, and use instructions to start and stop the timer.

Solution:

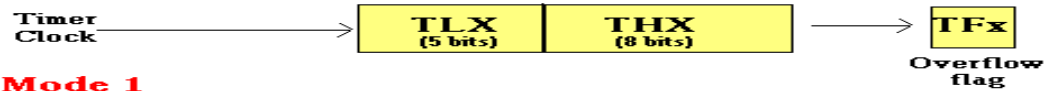
timer 1 timer 0

TMOD= 0000 0010

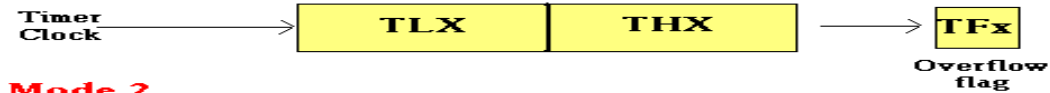
Timer 1 is not used.
Timer 0, **mode 2**,
C/T = 0 to use XTAL clock source (timer)
gate = 0 to use internal (**software**)
start and stop method.

Timer modes

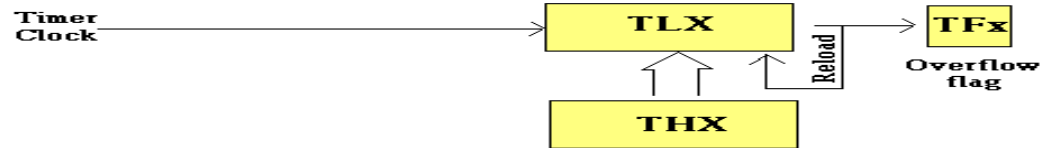
Mode 0



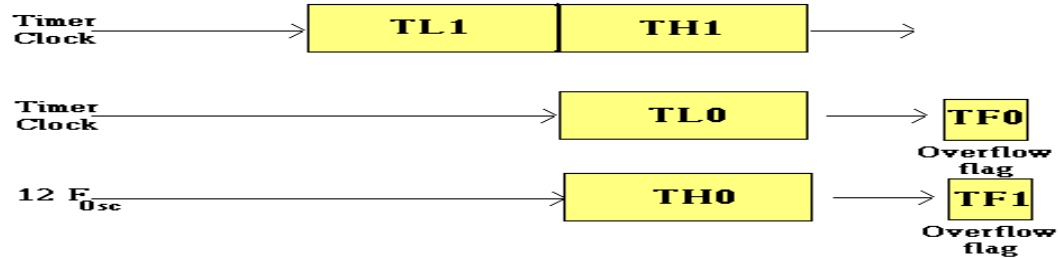
Mode 1



Mode 2



Mode 3



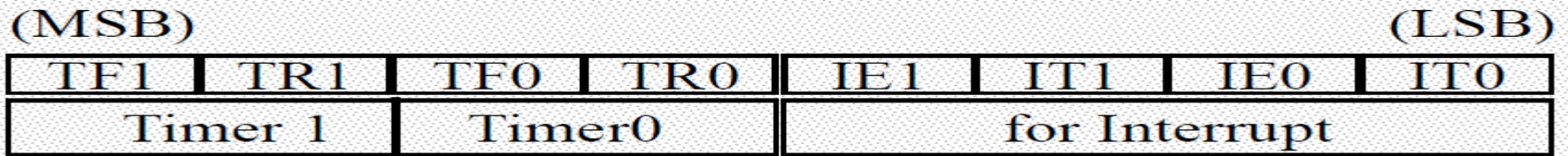
TCON Register (1/2)

- Timer control register: **TMOD**
 - Upper nibble for timer/counter, lower nibble for interrupts
- **TR** (run control bit)
 - TR0 for Timer/counter 0; TR1 for Timer/counter 1.
 - TR is set by programmer to turn timer/counter on/off.
 - TR=0: off (stop)
 - TR=1: on (start)

(MSB)				(LSB)			
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0
Timer 1		Timer0		for Interrupt			

TCON Register (2/2)

- **TF** (timer flag, control flag)
 - TF0 for timer/counter 0; TF1 for timer/counter 1.
 - TF is like a carry. Originally, TF=0. When TH-TL roll over to 0000 from FFFFH, the TF is set to 1.
 - TF=0 : not reach
 - TF=1: reach
 - If we enable interrupt, TF=1 will trigger ISR.



Equivalent Instructions for the Timer Control Register

For timer 0

SETB TR0	=	SETB TCON.4
----------	---	-------------

CLR TR0	=	CLR TCON.4
---------	---	------------

SETB TF0	=	SETB TCON.5
----------	---	-------------

CLR TF0	=	CLR TCON.5
---------	---	------------

For timer 1

SETB TR1	=	SETB TCON.6
----------	---	-------------

CLR TR1	=	CLR TCON.6
---------	---	------------

SETB TF1	=	SETB TCON.7
----------	---	-------------

CLR TF1	=	CLR TCON.7
---------	---	------------

TCON: Timer/Counter Control Register

TF1

TR1

TF0

TR0

IE1

IT1

IE0

IT0

Timer Mode 1

- In following, we all use timer 0 as an example.
- **16-bit** timer (TH0 and TL0)
- TH0-TL0 is incremented continuously when TR0 is set to 1. And the 8051 stops to increment TH0-TL0 when TR0 is cleared.
- The timer works with the internal system clock. In other words, the timer counts up each machine cycle.
- When the timer (TH0-TL0) reaches its maximum of FFFFH, it rolls over to 0000, and TF0 is raised.
- Programmer should check TF0 and stop the timer 0.

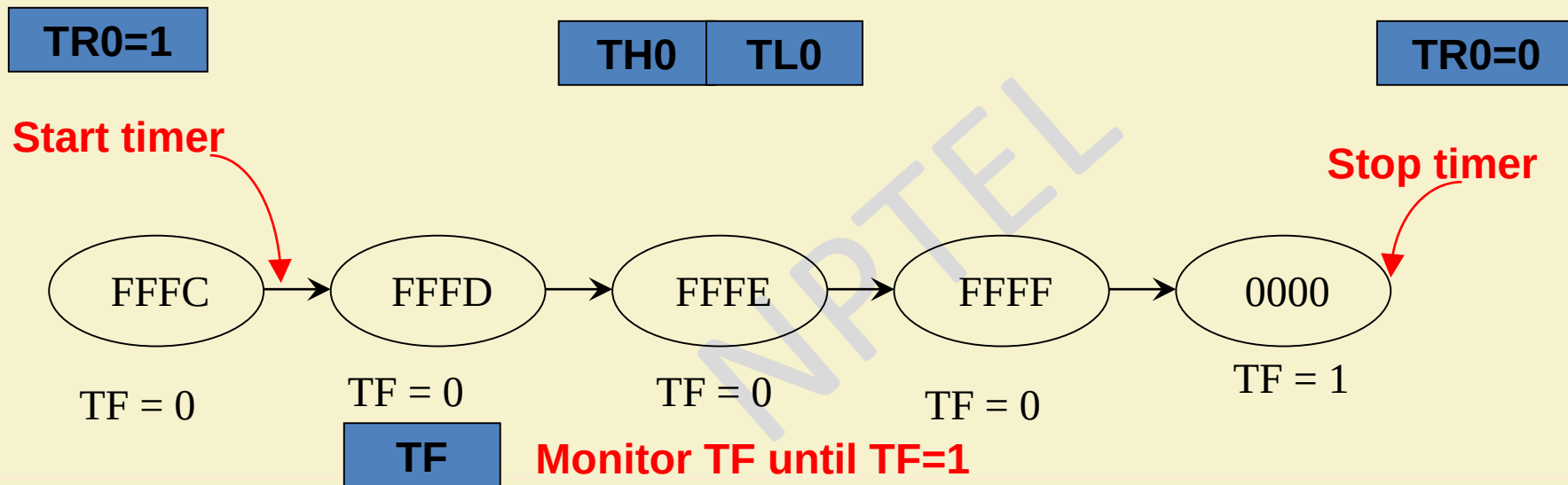
Steps of Mode 1 (1/3)

1. Choose mode 1 timer 0
 - **MOV TMOD, #01H**
2. Set the original value to TH0 and TL0.
 - **MOV TH0, #FFH**
 - **MOV TL0, #FCH**
3. You had better to clear the flag to monitor: TF0=0.
 - **CLR TF0**
4. Start the timer.
 - **SETB TR0**

Steps of Mode 1 (2/3)

5. The 8051 starts to count up by incrementing the TH0-TL0.

– TH0-TL0= FFCH,FFFDH,FFFEH,FFFFH,0000H



Steps of Mode 1 (3/3)

6. When TH0–TL0 rolls over from FFFFH to 0000, the 8051 set TF0=1.

TH0-TL0= FFFE H, FFFF H, 0000 H (Now TF0=1)

7. Keep monitoring the timer flag (TF) to see if it is raised.

AGAIN: JNB TF0, AGAIN

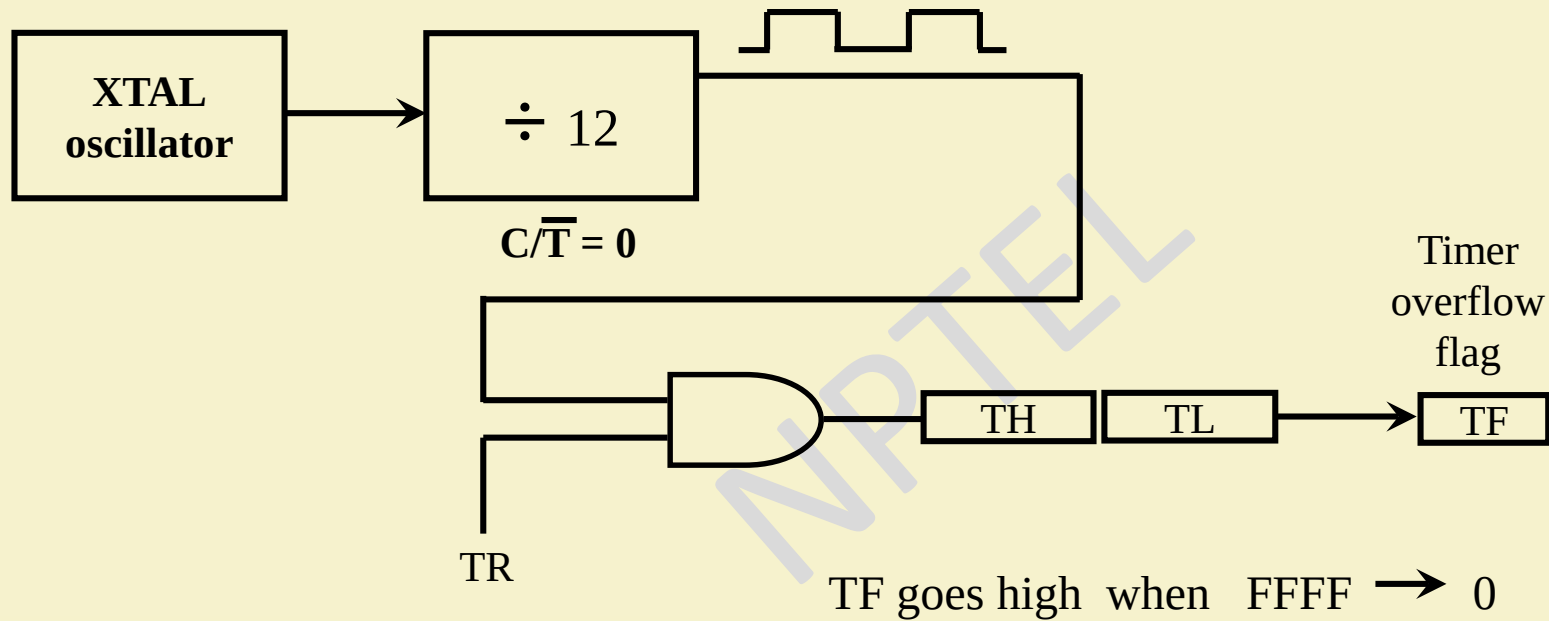
8. Clear TR0 to stop the process.

CLR TR0

9. Clear the TF flag for the next round.

CLR TF0

Mode 1 Programming



Timer Delay Calculation for XTAL = 11.0592 MHz

(a) in hex

- $(FFFF - YYXX + 1) \times 1.085 \mu s$
- where YYXX are TH, TL initial values respectively.
- Notice that values YYXX are in hex.

(b) in decimal

- Convert YYXX values of the TH, TL register to decimal to get a NNNNN decimal number
- then $(65536 - NNNNN) \times 1.085 \mu s$

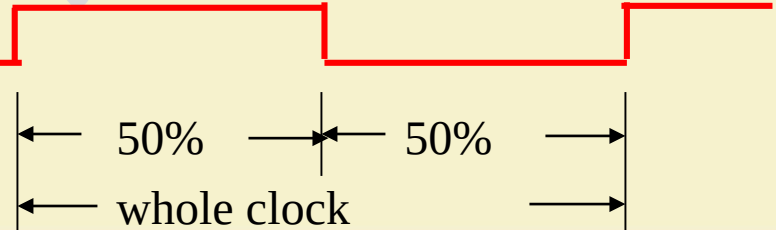
Example

- square wave of 50% duty on P1.5
- Timer 0 is used

;each loop is a half clock

```
MOV TMOD,#01    ;Timer 0,mode 1(16-bit)
HERE: MOV TL0,#0F2H ;Timer value = FFF2H
MOV TH0,#0FFH
CPL P1.5
ACALL DELAY
SJMP HERE
```

P1.5



Example

;generate delay using timer 0

DELAY:

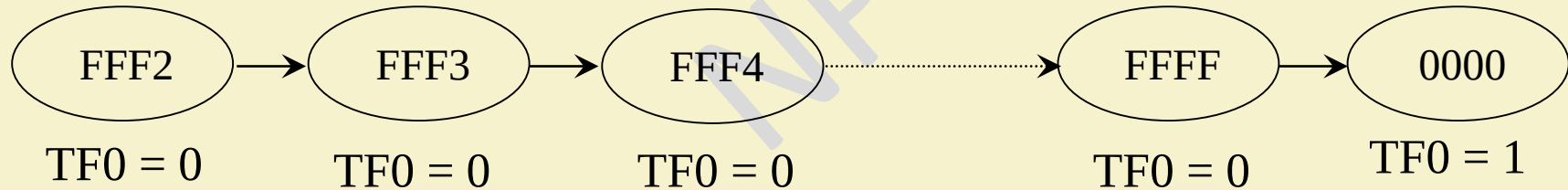
SETB TR0 ;start the timer 0

AGAIN: JNB TF0, AGAIN

CLR TR0 ;stop timer 0

CLR TF0 ;clear timer 0 flag

RET



Example

- This program generates a square wave on pin P1.5 Using timer 1
- Find the frequency.(dont include the overhead of instruction delay)
- XTAL = 11.0592 MHz

```
      MOV    TMOD, #10H      ;timer 1, mode 1
AGAIN: MOV    TL1, #34H      ;timer value=3476H
      MOV    TH1, #76H
      SETB   TR1             ;start
BACK:  JNB    TF1, BACK
      CLR    TR1             ;stop
      CPL    P1.5            ;next half clock
      CLR    TF1             ;clear timer flag 1
      SJMP   AGAIN           ;reload timer1
```

Example

Solution:

$$\text{FFFFH} - 7634\text{H} + 1 = 89\text{CCH} = 35276 \text{ clock count}$$

$$\text{Half period} = 35276 \times 1.085 \mu\text{s} = 38.274 \text{ ms}$$

$$\text{Whole period} = 2 \times 38.274 \text{ ms} = 76.548 \text{ ms}$$

$$\text{Frequency} = 1 / 76.548 \text{ ms} = 13.064 \text{ Hz.}$$

Note

Mode 1 is not auto reload then the program must reload the TH1, TL1 register every timer overflow if we want to have a continuous wave.

Find Timer Values

- Assume that XTAL = 11.0592 MHz .
- And we know desired **delay**
- how to find the values for the TH,TL ?
 1. Divide the **delay** by $1.085 \mu\text{s}$ and get n.
 2. Perform **65536 - n**
 3. Convert the result of Step 2 to hex (**yyxx**)
 4. Set **TH = yy** and **TL = xx**.

Example

- Assuming XTAL = 11.0592 MHz,
- write a program to generate a square wave of 50 Hz frequency on pin P2.3.

Solution:

1. The period of the square wave = $1 / 50 \text{ Hz} = 20 \text{ ms}$.
2. The high or low portion of the square wave = **10 ms**.
3. **$10 \text{ ms} / 1.085 \mu\text{s} = 9216$**
4. **$65536 - 9216 = 56320$** in decimal = **DC00H** in hex.
5. TL1 = 00H and TH1 = DCH.

Example

```
      MOV    TMOD,#10H      ;timer 1, mode 1
AGAIN: MOV    TL1,#00        ;Timer value = DC00H
      MOV    TH1,#0DCH
      SETB   TR1            ;start
BACK:  JNB    TF1,BACK
      CLR    TR1            ;stop
      CPL    P2.3
      CLR    TF1            ;clear timer flag 1
      SJMP   AGAIN          ;reload timer since
                              ;mode 1 is not
                              ;auto-reload
```

Timer Mode 0

- Mode 0 is exactly like mode 1 except that it is a **13-bit** timer instead of 16-bit.
 - 8-bit TH0
 - 5-bit TL0
- The counter can hold values between 0000 to 1FFF in TH0-TL0.
 - $2^{13}-1 = 2000H-1 = 1FFFH$
- We set the initial values TH0-TL0 to **count up**.
- When the timer reaches its maximum of 1FFFH, it rolls over to 0000, and TF0 is raised.

Timer Mode 2

- 8-bit timer.
 - It allows only values of 00 to FFH to be loaded into TH0.
- Auto-reloading
- TL0 is incremented continuously when TR0=1.

Steps of Mode 2

1. Chose mode 2 timer 0

MOV TMOD, #02H

2. Set the original value to TH0.

MOV TH0, #38H

3. Clear the flag to TF0=0.

CLR TF0

4. After TH0 is loaded with the 8-bit value, the 8051 gives a copy of it to TL0.

TL0=TH0=38H

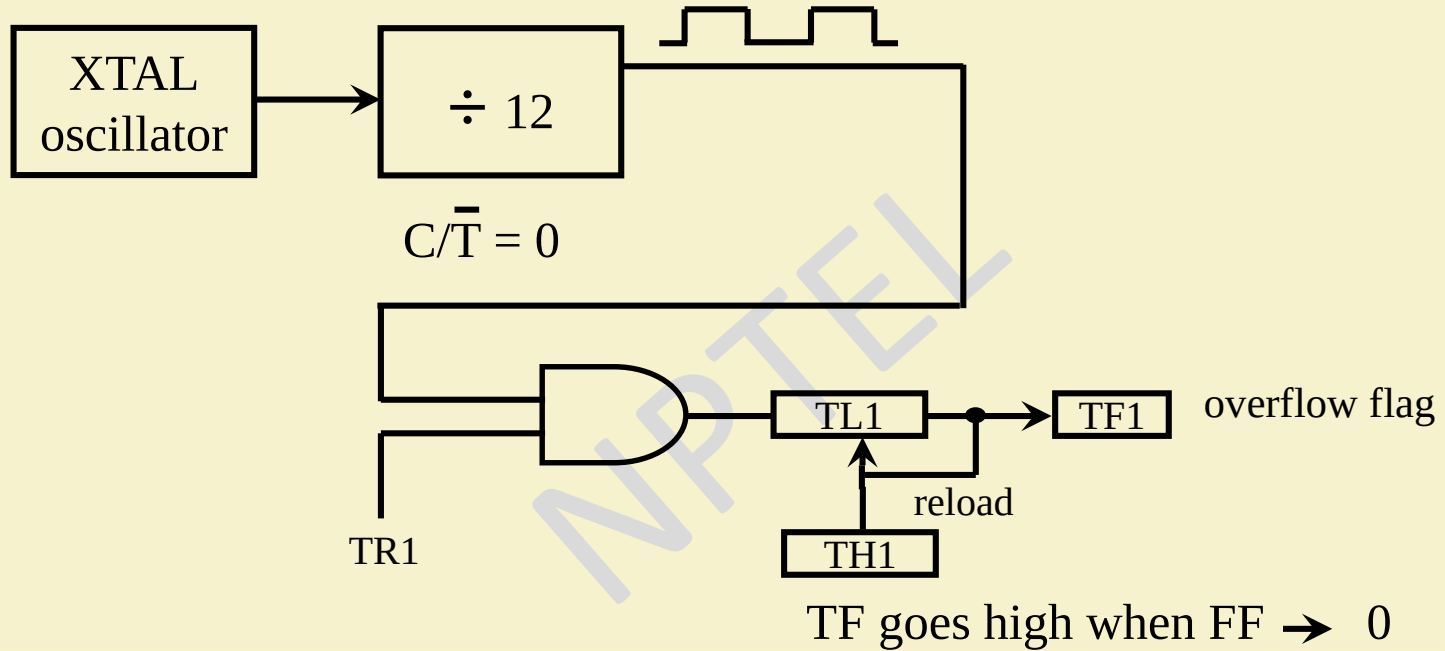
5. Start the timer.

SETB TR0

Steps of Mode 2

6. The 8051 starts to count up by incrementing the TL0.
 - **TL0= 38H, 39H, 3AH,**
7. When TL0 rolls over from FFH to 00H, the 8051 set TF0=1. Also, TL0 is reloaded automatically with the value kept by the TH0.
 - **TL0= FEH, FFH, 00H (Now TF0=1)**
 - The 8051 auto reload **TL0=TH0=38H.**
 - **Clr TF0**
 - Go to Step 6 (i.e., TL0 is incrementing continuously).
- Note that **we must clear TF0** when TL0 rolls over. Thus, we can monitor TF0 in next process.
- Clear TR0 to stop the process.
 - **Clr TR0**

Timer 1 Mode 2 with internal Input



Counter

- These timers can also be used as counters **counting events** happening outside the 8051.
- When the timer is used as a counter, it is a pulse outside of the 8051 that increments the TH, TL.
- When $C/T=1$, the counter counts up as pulses are fed from
 - T0: timer 0 input (Pin 14, P3.4)
 - T1: timer 1 input (Pin 15, P3.5)

Port 3 Pins Used For Timers 0 and 1

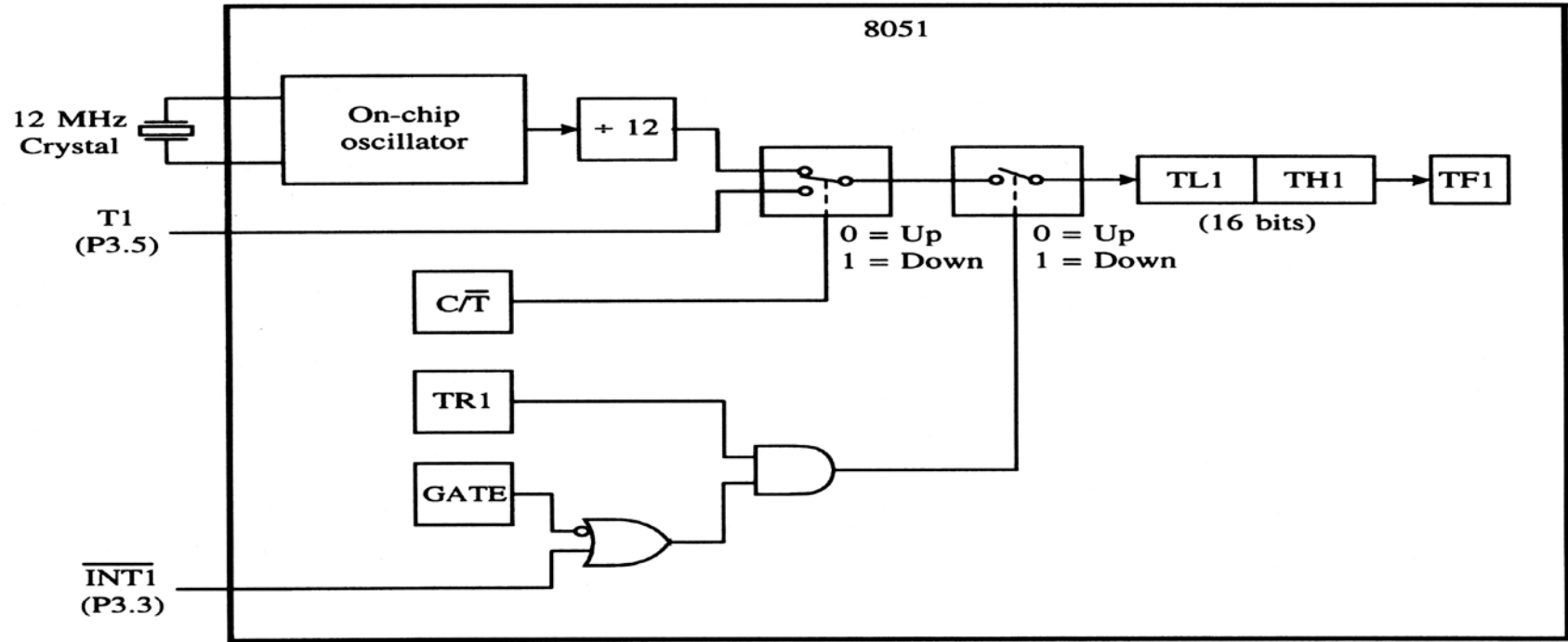
Pin	Port Pin	Function	Description
14	P3.4	T0	Timer/Counter 0 external input
15	P3.5	T1	Timer/Counter 1 external input

(MSB)

(LSB)

GATE	C/T=1	M1	M0	GATE	C/T=1	M1	M0
Timer 1				Timer 0			

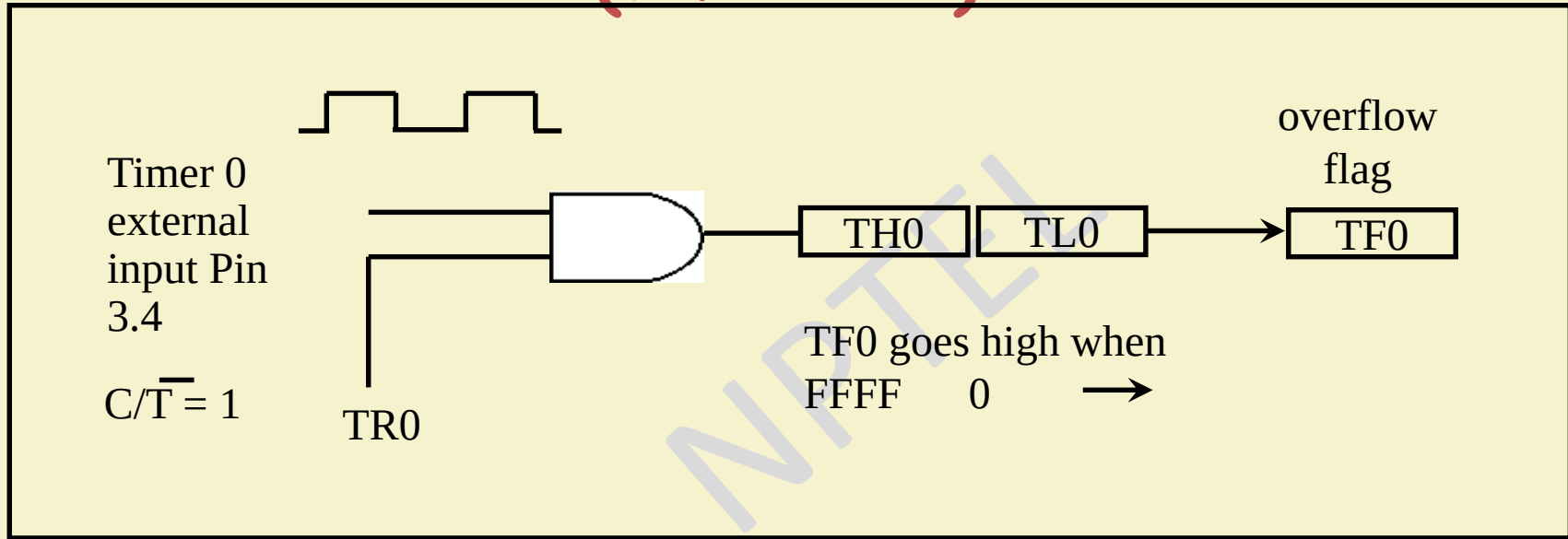
Timer/Counter selection



Counter Mode 1

- **16-bit** counter (TH0 and TL0)
- TH0-TL0 is incremented when TR0 is set to 1 **and** an external pulse (in T0) occurs.
- When the counter (TH0-TL0) reaches its maximum of FFFFH, it rolls over to 0000, and TF0 is raised.
- Programmers should monitor TF0 continuously and stop the counter 0.
- Programmers can set the initial value of TH0-TL0 and let TF0=1 as an indicator to show a special condition. (ex: 100 people have come).

Timer 0 with External Input (Mode 1)



Counter Mode 2

- 8-bit **counter**.
 - It allows only values of 00 to FFH to be loaded into TH0.
- Auto-reloading
 - TL0 is incremented if TR0=1 **and** external pulse occurs.

Example

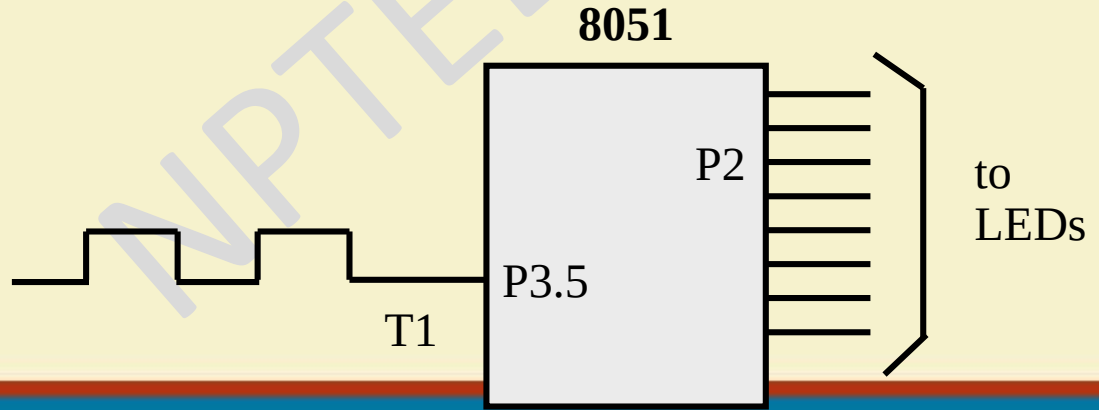
Assuming that clock pulses are fed into pin T1, write a program for counter 1 in mode 2 to count the pulses and display the state of the TL 1 count on P2.

```
MOV    TMOD,#01100000B    ;mode 2, counter 1
MOV    TH1,#0
SETB   P3.5                ;make T1 input port
AGAIN: SETB TR1             ;start
BACK:  MOV    A,TL1
MOV    P2,A                ;display in P2
JNB    TF1,Back            ;overflow
CLR    TR1                 ;stop
CLR    TF1                 ;make TF=0
SJMP   AGAIN               ;keep doing it
```

Example

- Timer 1 as an event counter fed into pin3.5.
- “SETB P3.5” make P3.5 an input port by making it high

P2 is connected to 8 LEDs
and input T1 to pulse.

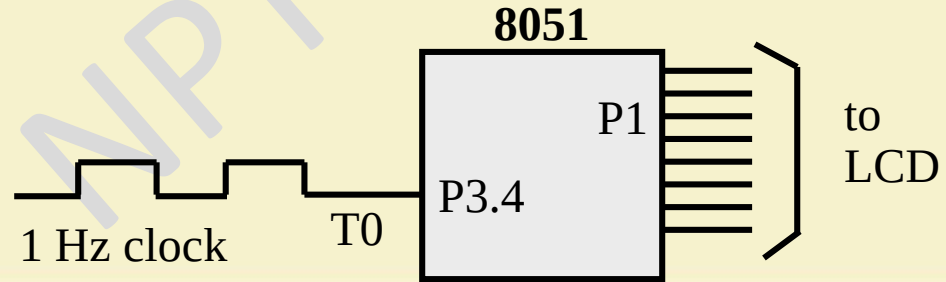


Example

Assume that a **1-Hz frequency pulse** is connected to input pin 3.4. Write a program to display counter 0 on an LCD. Set the initial value of TH0 to -60.

Solution:

Note that on the first round, it starts from 0 and counts 256 events, since on RESET, TL0=0. To solve this problem, load TH0 with -60 at the beginning of the program.



Example

```
ACALL LCD_SET_UP      ;initialize the LCD
MOV  TMOD,#00000110B  ;Counter 0,mode2
MOV  TH0,#-60
SETB P3.4             ;make T0 as input
AGAIN:SETB TR0         ;starts the counter
BACK: MOV  A,TL0       ;every 60 events
      ACALL CONV       ;convert in R2,R3,R4
      ACALL DISPLY     ;display on LCD
      JNB  TF0,BACK    ;loop if TF0=0
      CLR  TR0         ;stop
      CLR  TF0
      SJMP AGAIN
```

GATE=1 in TMOD

- All discuss so far has assumed that GATE=0.
 - The timer is started with instructions “**SETB TR0**” and “**SETB TR1**” for timers 0 and 1, respectively.
- If GATE=1, we can use hardware to control the start and stop of the timers.
 - INT0 (P3.2, pin 12) starts and stops timer 0
 - INT1 (P3.3, pin 13) starts and stops timer 1
 - This allows us to start or stop the timer externally at any time via a simple switch.

GATE (external control)

- Timer 0 must be turned on by “**SETB TR0**”
- If GATE=1 count up if
 - INT0 input is high
 - TR0=1
- If GATE=0 count up if
 - TR0=1

Timer Special Function Register

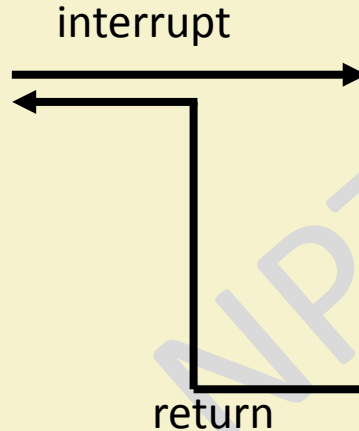
TIMER SFR	PURPOSE	ADDRESS	BIT-ADDRESSABLE
TCON	Control	88H	Yes
TMOD	Mode	89H	No
TL0	Timer 0 low-byte	8AH	No
TL1	Timer 1 low-byte	8BH	No
TH0	Timer 0 high-byte	8CH	No
TH1	Timer 1 high-byte	8DH	No
T2CON*	Timer 2 control	C8H	Yes
RCAP2L*	Timer 2 low-byte capture	CAH	No
RCAP2H*	Timer 2 high-byte capture	CBH	No
TL2*	Timer 2 low-byte	CCH	No
TH2*	Timer 2 high-byte	CDH	No

*8032/8052 only

Interrupts

Program Execution

```
...  
mov a, #2  
mov b, #16  
mul ab  
mov R0, a  
mov R1, b  
mov a, #12  
mov b, #20  
mul ab  
add a, R0  
mov R0, a  
mov a, R1  
addc a, b  
mov R1, a  
end
```



```
ISR: inc r7  
      mov a, r7  
      jnz NEXT  
      cpl P1.6  
NEXT: reti
```



Interrupt Sources

- Original 8051 has 5 sources of interrupts
 - Timer 0 overflow
 - Timer 1 overflow
 - External Interrupt 0
 - External Interrupt 1
 - Serial Port events (buffer full, buffer empty, etc)
- Enhanced version has 22 sources
 - More timers, programmable counter array, ADC, more external interrupts, another serial port (UART)

Interrupt Process

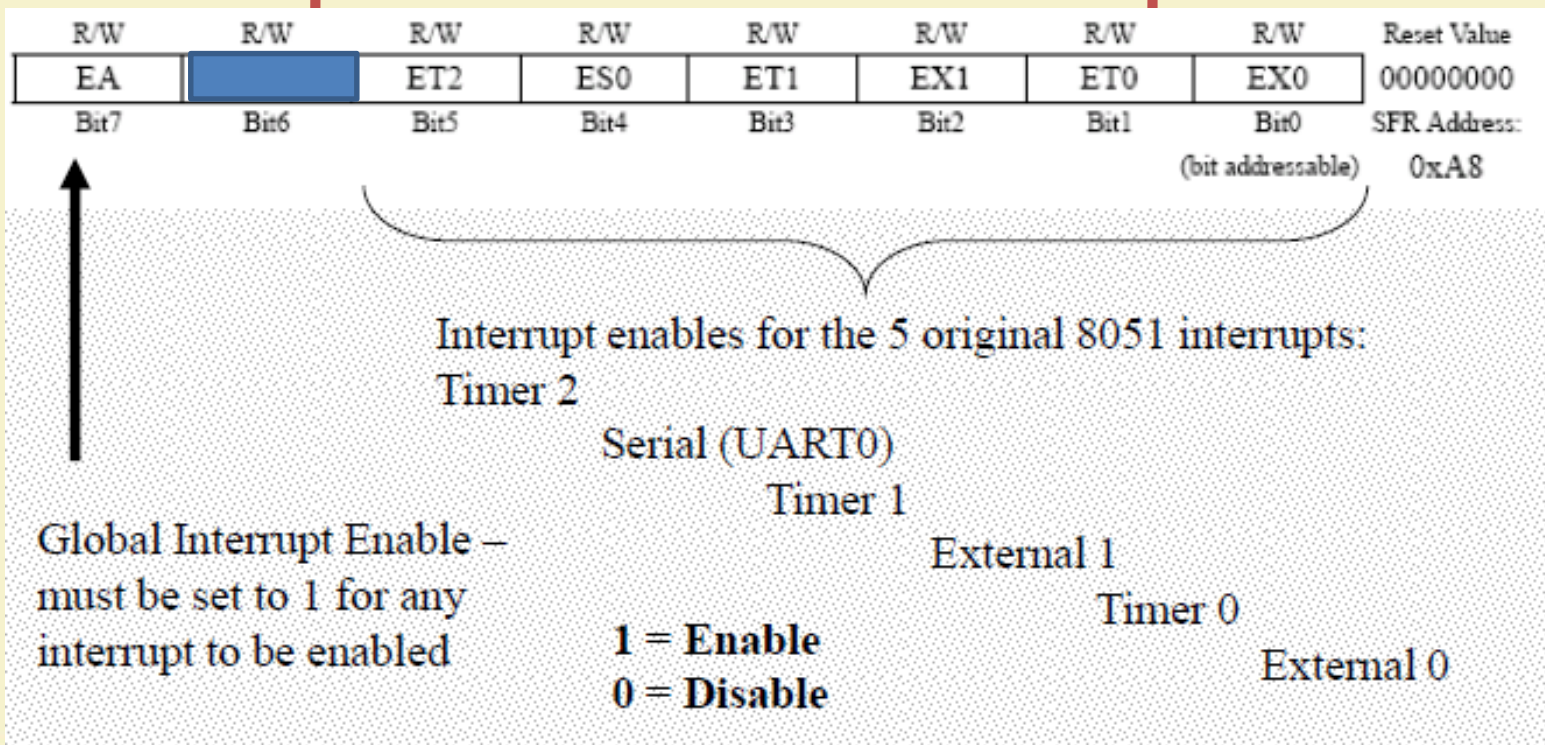
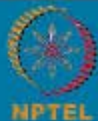
If interrupt event occurs AND interrupt flag for that event is enabled, AND interrupts are enabled, then:

1. Current PC is pushed on stack.
2. Program execution continues at the interrupt vector address for that interrupt.
3. When a RETI instruction is encountered, the PC is popped from the stack and program execution resumes where it left off.

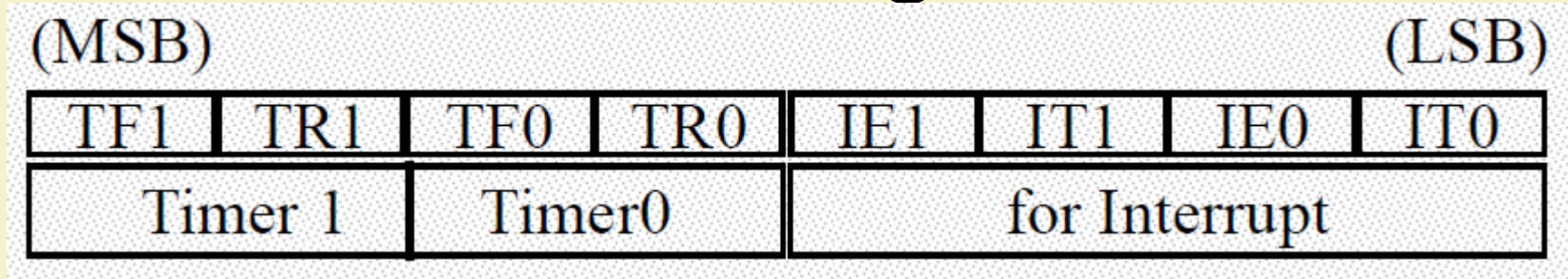
Interrupt Priorities

- What if **two** interrupt sources interrupt at the **same time**?
- The interrupt with the highest PRIORITY gets serviced first.
- All interrupts have a default priority order.
- Priority can also be set to “high” or “low”.

Interrupt SFR - Interrupt Enable



TCON Register



IE1: Set by CPU when H-L transition detected on external interrupt 1. Cleared when processed.

IT1: Interrupt 1 type control. Set/cleared by software to specify falling edge/low-level triggered.

Similarly IE0 and IT0.

Interrupt Priorities

- Default priority: $INT0 > TF0 > INT1 > TF1 > RI + TI$
- To alter priority set IP register

--	--	PT2	PS	PT1	PX1	PT0	PX0
----	----	-----	----	-----	-----	-----	-----

- PT2: Priority of timer 2 interrupt
- PS: Priority for serial port interrupt, and so on.
- “MOV IP, #00000100B” sets INT1 to have the highest priority
- “MOV IP, #00001100B” sets priorities as $INT1 > TF1 > INT0 > TF0 > RI + TI$

Interrupt Vectors

Each interrupt has a **specific** place in **code** memory where program execution (interrupt service routine) begins.

Reset:	0000h
External Interrupt 0:	0003h
Timer 0 overflow:	000Bh
External Interrupt 1:	0013h
Timer 1 overflow:	001Bh
Serial :	0023h
Timer 2 overflow(8052+)	002bh

Note: that there are only 8 memory locations between vectors.

Interrupt Vectors

To avoid **overlapping** Interrupt Service routines, it is common to put **JUMP** instructions at the vector address. This is similar to the reset vector.

```
                org 000B                ; at EX7 vector
                ljmp EX7ISR
                org 0x100                ; at Main program
Main:           ...                     ; Main program
                ...
EX7ISR:         ...                     ; Interrupt service routine
                ...                     ; Can go after main program
                reti                    ; and subroutines.
```

Example Interrupt Service Routine

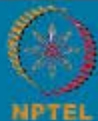
Pin 3.3 (INT1) is connected to a pulse generator. Write a program in which the falling edge of the pulse will send a high to P1.3, connected to a LED.

```
        org 0000h
        ljmp main
; ISR for hardware interrupt INT1
        org 0013h
        setb p1.3
        mov r3, #255
back:    djnz r3, back
        clr P1.3
        reti
; Main program for initialization
        org 30h
main:    setb tcon.2      ; make INT1 edge triggered
        mov ie, #10000100B ; enable int1
here:    sjmp here
```

Serial Communication



IIT KHARAGPUR



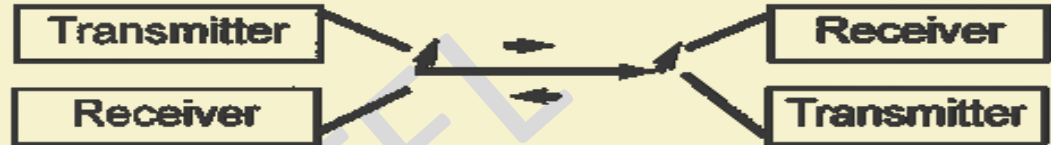
NPTEL ONLINE
CERTIFICATION COURSES

Basics of serial communication

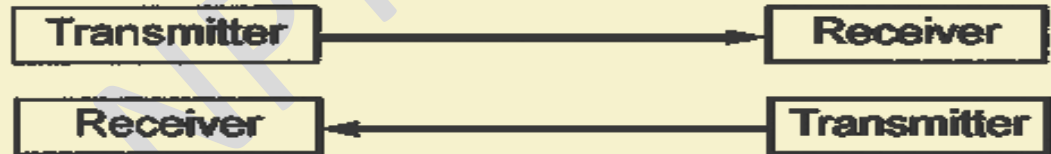
Simplex



Half Duplex



Full Duplex



Start and stop bits

□ When there is **no** transfer the signal is **high**

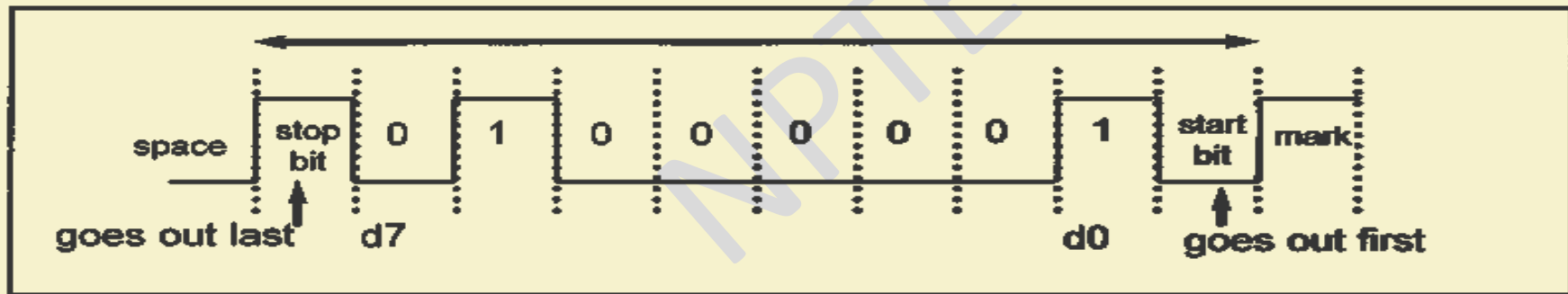
Transmission begins with a start (**low**) bit

LSB first

Finally 1 stop bit (**high**)

Data transfer rate (baud rate) is stated in **bps**

bps: bit per second



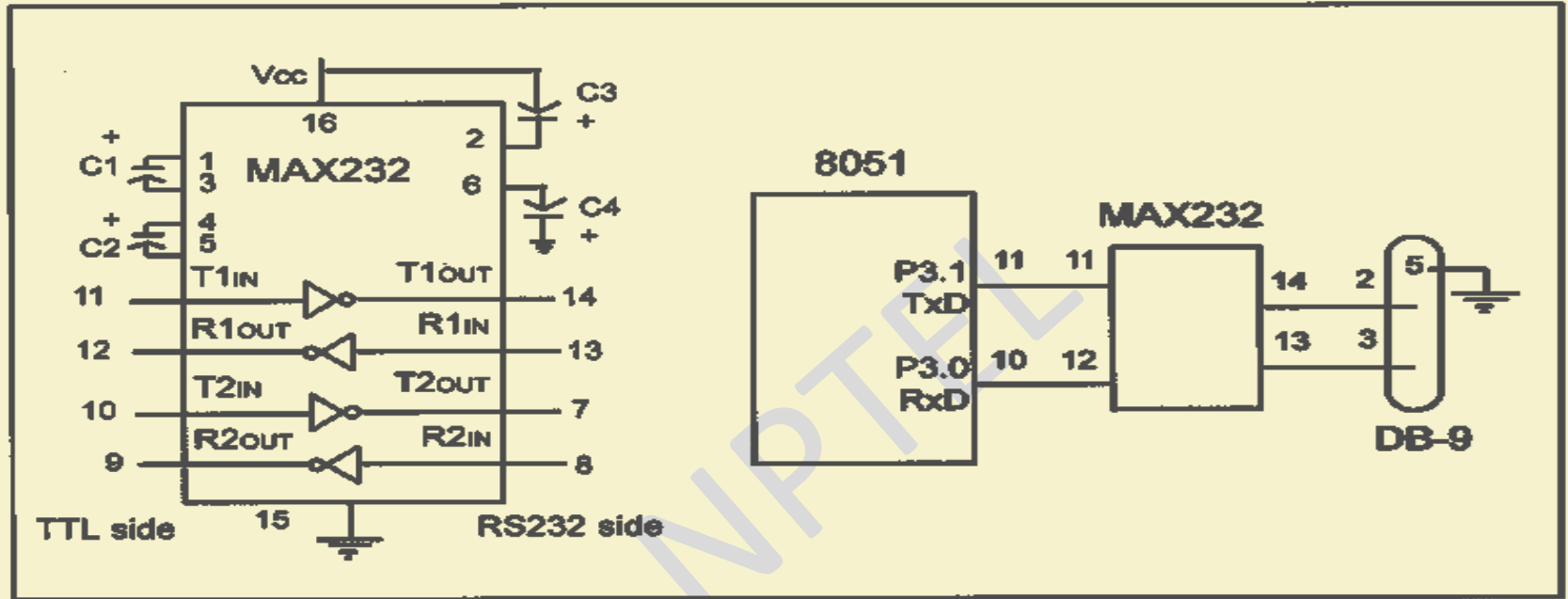
RxD and TxD pins in the 8051

- TxD pin 11 of the 8051 (P3.1)
- RxD pin 10 of the 8051 (P3.0)

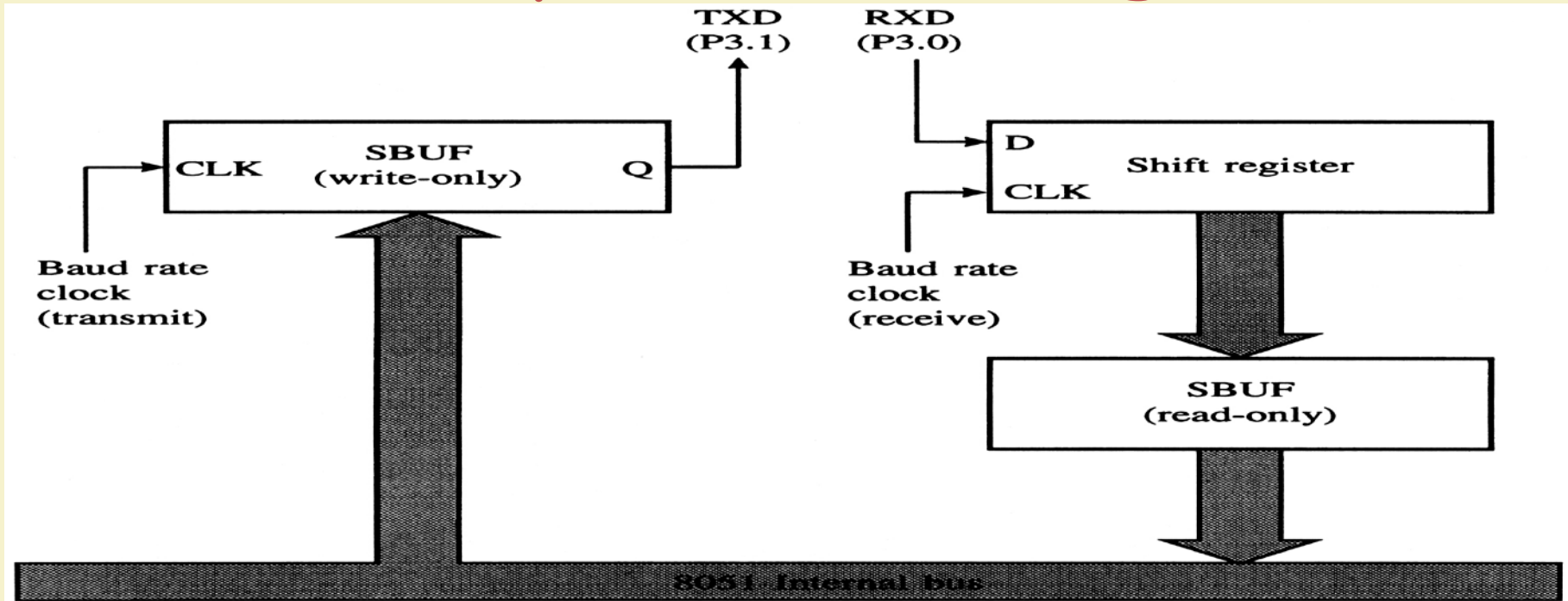
SBUF register

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

MAX232



Serial port block diagram



Baud Rate	TH1 (Decimal)	TH1 (Hex)
9600	-3	FD
4800	-6	FA
2400	-12	F4
1200	-24	E8

Note: XTAL = 11.0592 MHz.

Serial control (SCON) Register

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

SM0 (SCON.7) : mode specifier

SM1 (SCON.6) : mode specifier

SM2 (SCON.5) : used for multi processor communication

REN (SCON.4) : receive enable (by software enable/disable)

TB8 (SCON.3) : transmit bit8

RB8 (SCON.2) : receive bit 8

TI (SCON.1) : transmit interrupt flag set by HW clear by SW

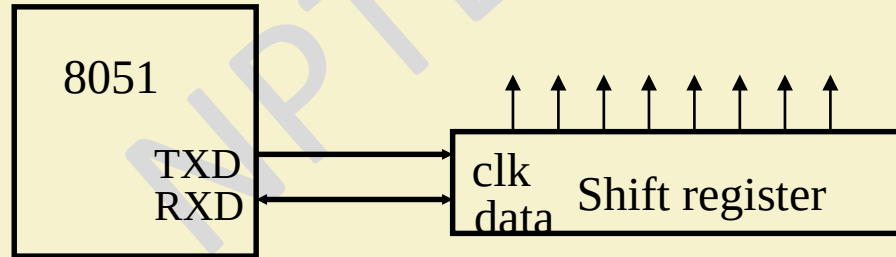
RI (SCON.0) : receive interrupt flag set by HW clear by SW

Mode of operation

SM0	SM1	MODE	operation	transmit rate
0	0	0	shift register	fixed ($x_{tal}/12$)
0	1	1	8 bit UART	variable (timer1)
1	0	2	9 bit UART	fixed ($x_{tal}/32$ or $x_{tal}/64$)
1	1	3	9 bit UART	variable (timer1)

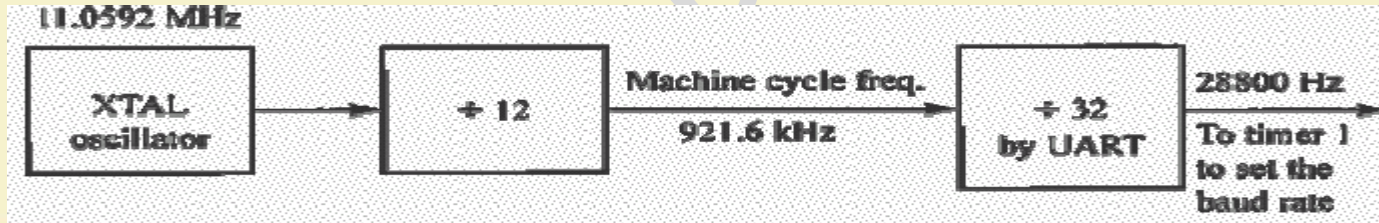
Mode of operation

- Mode 0 :
 - Serial **data** enters and exits through **RxD**
 - **TxD** outputs the shift **clock**.
 - 8 bits are transmitted/received (LSB first)
 - The baud rate is fixed at $1/12$ the oscillator frequency.
- Application
 - Port expansion

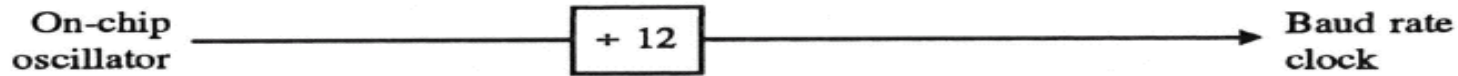


Mode of operation

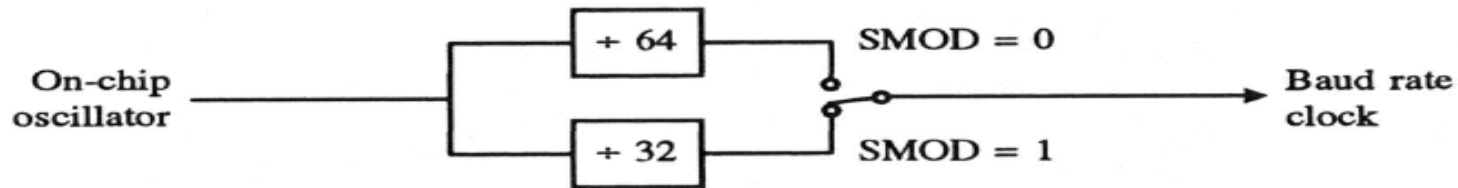
- Mode 1
 - Ten bits are transmitted (through TxD) or received (through RxD)
 - A start bit (0), 8 data bits (LSB first), and a stop bit (1)
 - On receive, the stop bit goes into RB8 in SCON
 - the baud rate is determined by the Timer 1 overflow rate.
 - Timer1 clock is $1/32$ machine cycle ($MC=1/12$ XTAL)
 - Timer clock can be programmed as $1/16$ of machine cycle
 - Transmission is initiated by any instruction that uses SBUF as a destination register.



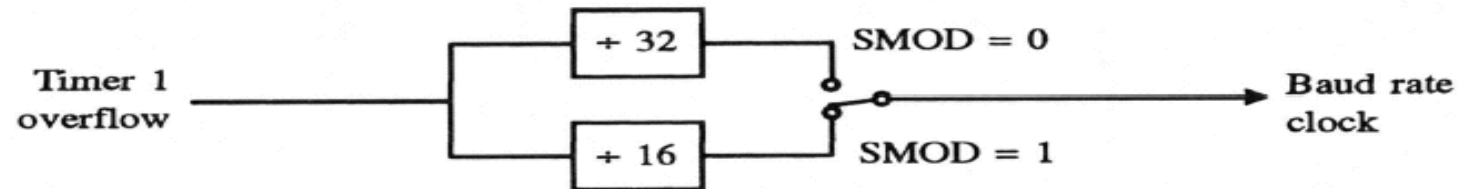
Mode of operation



(a) Mode 0



(b) Mode 2



(c) Modes 1 and 3



Mode of operation

- Mode 2 :

- Eleven bits are transmitted (through TxD), received (through RxD)
 - A start bit (0)
 - 8 data bits (LSB first)
 - A programmable 9th data bit
 - and a stop bit (1)
- On transmit, the 9th bit (TB8) can be assigned 0 or 1.
- On receive, the 9th data bit goes into RB8 in SCON.
- the 9th can be parity bit
- The baud rate is programmable to 1/32 or 1/64 the oscillator frequency in Mode 2 by SMOD bit in PCON register.

- Mode 3

- Same as mode 2
- But may have a variable baud rate generated from Timer 1.

What is SMOD

- ❑ Bit 7 of PCON register
- ❑ If SMOD=1 **double** baud rate
- ❑ PCON is not bit addressable
- ❑ How to set SMOD

Mov a, pcon

Setb acc.7

Mov pcon, a

MSB

LSB

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

Power control register

MSB				LSB			
SMOD	—	—	—	GF1	GF0	PD	IDL

BIT	SYMBOL	FUNCTION
PCON.7	SMOD	Double Baud rate bit. When set to a 1 and Timer 1 is used to generate baud rate, and the Serial Port is used in modes 1, 2, or 3.
PCON.6	—	Reserved.
PCON.5	—	Reserved.
PCON.4	—	Reserved.
PCON.3	GF1	General-purpose flag bit.
PCON.2	GF0	General-purpose flag bit.
PCON.1	PD	Power-Down bit. Setting this bit activates power-down operation.
PCON.0	IDL	Idle mode bit. Setting this bit activate idle mode operation.

Power control

- A standard for applications where power consumption is critical
- two power reducing modes
 - Idle
 - Power down

Idle mode

- An instruction that sets PCON.0 causes Idle mode
 - Last instruction executed before going into the Idle mode
 - the internal CPU clock is gated off
 - Interrupt, Timer, and Serial Port functions act normally.
 - All of registers , ports and internal RAM maintain their data during Idle
 - ALE and PSEN hold at logic high levels
- Any interrupt
 - will cause PCON.0 to be cleared by HW (terminate Idle mode)
 - then execute ISR
 - with RETI return and execute next instruction after Idle instruction.
- RST signal clears the IDL bit directly

Power-Down Mode

- An instruction that sets PCON.1 causes power down mode
- Last instruction executed before going into the power down mode
- the on-chip oscillator is stopped.
- all functions are stopped, the contents of the on-chip RAM and Special Function Registers are maintained.
- The ALE and PSEN output are held low
- The **reset** that terminates Power Down

Power control example

```
Org 0000h  
Ljmp main
```

```
Org 0003h  
Orl pcon, #02h      ;power down mode  
Reti
```

```
Main:  
.....  
.....  
.....  
Orl pcon, #01h      ;Idle mode  
end
```

8051 Programming Examples

Santanu Chattopadhyay

Electronics and Electrical Communication Engineering

Example

Assume 10 bytes of data are stored from RAM location 50H. Compute the average of these numbers and store the result at memory location 40H.

For the code fragment, compute the size and the time needed assuming 11.0582MHz crystal

Example

Assume 10 bytes of data are stored from location 3000H in the external memory. Sort them in ascending order.