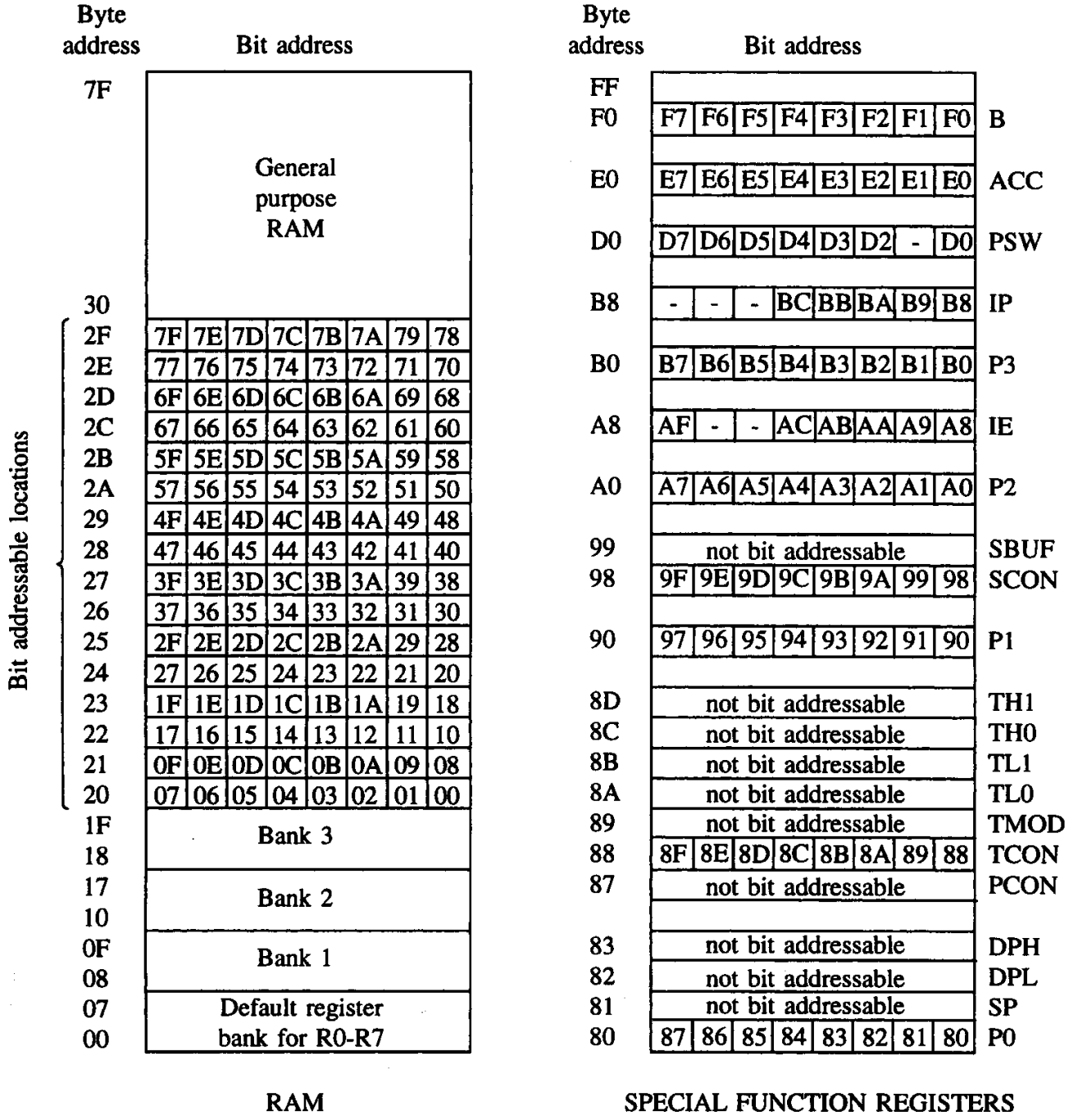
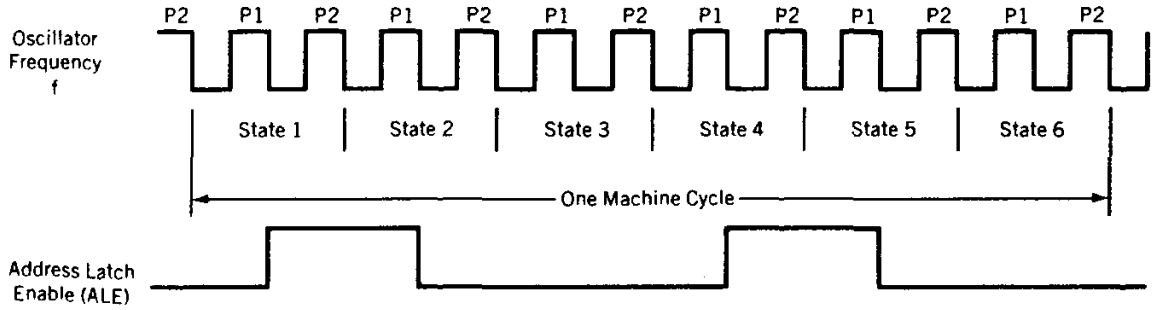
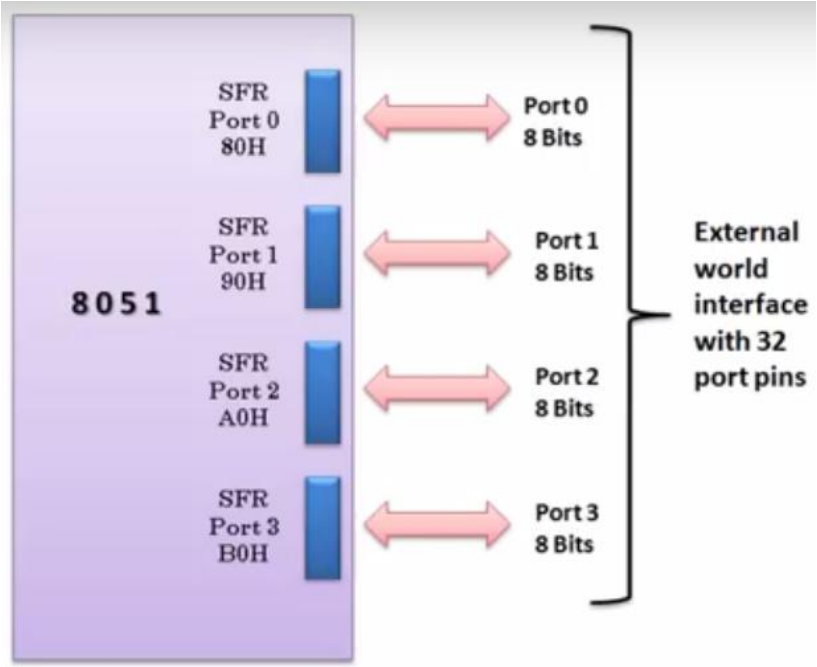
UNIT V - 8051 Microcontroller Hardware




Oscillator Frequency f
P2 P1 P2 P1 P2 P1 P2 P1 P2 P1 P2 P1 P2
State 1
State 2
State 3
State 4
State 5
State 6
One Machine Cycle
Address Latch Enable (ALE)

8051 Timing

Parallel Ports in 8051



 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
- When ALE=1, it has address A0-A7

 It can be used for input or output, each pin must be connected externally to a 10K ohm pull-up resistor
 In 8051-based systems with no external memory connection both P0 and P2 are used as simple I/O.
 In 8031/51-based systems with external memory connections Port 2 must be used along with P0 to provide the 16-bit address for the external memory.

- P0 provides the lower 8 bits via A0 – A7.
- P2 is used for the upper 8 bits of the 16-bit address, designated as A8 – A15, and it cannot be used for I/O.

 Port 3 can be used as input or output.

- Port 3 does not need any pull-up resistors.
- Port 3 has the additional function of providing some extremely important signals.

P3 Bit	Function	Pin
P3.0	RxD	10
P3.1	TxD	11
P3.2	$\overline{\text{INT0}}$	12
P3.3	$\overline{\text{INT1}}$	13
P3.4	T0	14
P3.5	T1	15
P3.6	$\overline{\text{WR}}$	16
P3.7	$\overline{\text{RD}}$	17

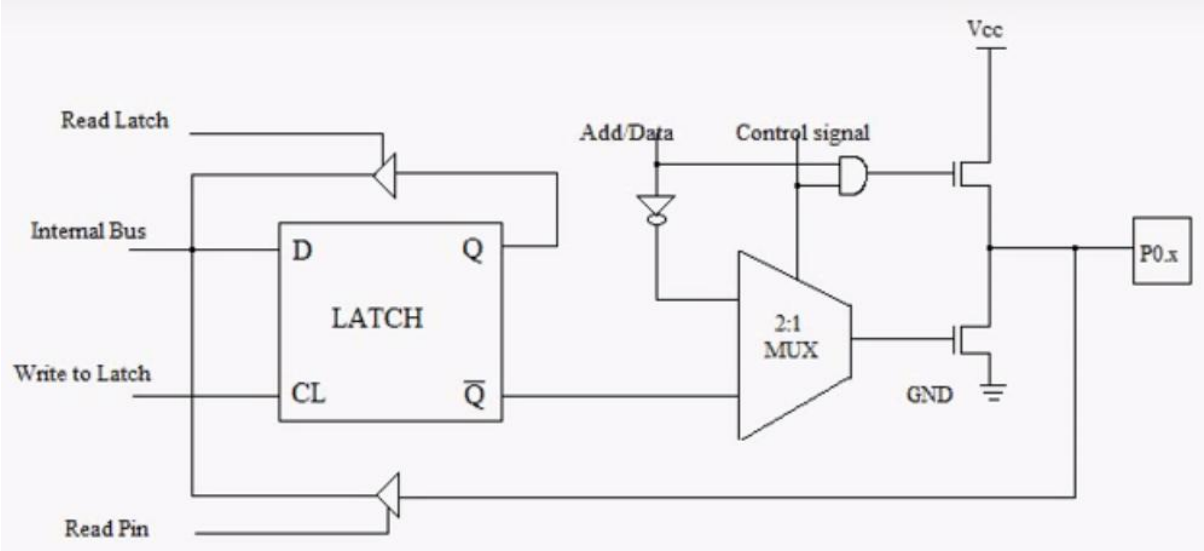
Serial communications

External interrupts

Timers

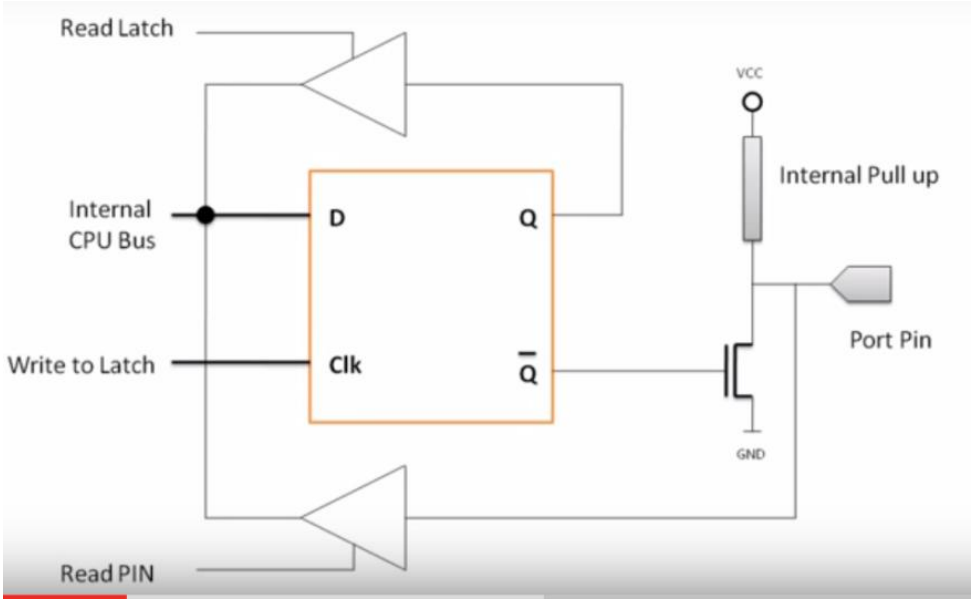
Read/Write signals of external memories

Port-0 Pin configuration

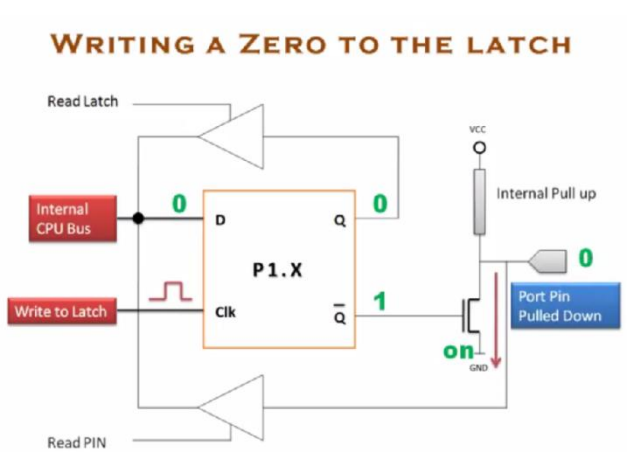


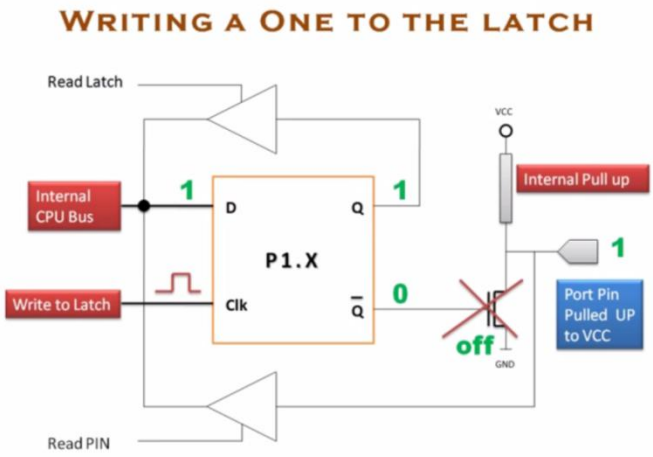
- When a pin is to be used as an input, a 1 must be written to the corresponding port 0 latch by the program, thus turning both of the output transistors off, which in turn causes the pin to "float" in a high-impedance state, and the pin is essentially connected to the input buffer.
- When used as an output, the pin latches that are programmed to a 0 will turn on the lower FET, grounding the pin. All latches that are programmed to a 1 still float; thus, external pullup resistors will be needed to supply logic high when using port 0 as an output.

Port-1 Pin Configuration



- Port 1 pins have no dual functions. Therefore, the output latch is connected directly to the gate of the lower FET, which has an FET circuit labeled "Internal FET Pullup" as an active pullup load.
- Used as an input, a 1 is written to the latch, turning the lower FET off; the pin and the input to the pin buffer are pulled high by the FET load.
- An external circuit can overcome the high impedance pullup and drive the pin low to input a 0 or leave the input high for a 1.
- If used as an output, the latches containing a 1 can drive the input of an external circuit high through the pullup. If a 0 is written to the latch, the lower FET is on, the pullup is off, and the pin can drive the input of the external circuit low.





Instruction which Read The Status of Input Port

Serial Number	Mnemonic	Example
1	MOV A , PX	MOV A,P1
2	JNB PX.Y , ...	JNB P1.2, TARGET
3	JB PX.Y , ...	JB P1.2, TARRAN
4	MOV C , PX.Y	MOV C, P1.4
5	CJNE A,PX	CJNE A, P3

Read-Modify-Write Instruction

Serial Number	Mnemonic	Example
1	ANL	ANL P1, A
2	ORL	ORL P3, A
3	XRL	XRL P2,A
4	CPL	CPL P2.1
5	INC	INC P3
6	DJNZ	DJNZ P1, Target

External Memory interfacing with 8051

The number of bits that a semiconductor memory chip can store is called chip capacity
 It can be in units of Kbits (kilobits), Mbits (megabits), and so on
 A memory chip contain 2X location, where x is the number of address pins.
 Each location contains Y bits, where y is the number of data pins on the chip.
 The entire chip will contain 2X × Y bits.

Example
A given memory chip has 12 address pins and 4 data pins. Find: (a) The organization, and (b) the capacity.
Solution:
(a) This memory chip has 4096 locations (212 = 4096), and each location can hold 4 bits of data. This gives an organization of 4096 × 4, often represented as 4K × 4.
(b) The capacity is equal to 16K bits since there is a total of 4K locations and each location can hold 4 bits of data.

Example
A 512K memory chip has 8 pins for data. Find: (a) The organization, and (b) the number of address pins for this memory chip.
Solution:
(a) A memory chip with 8 data pins means that each location within the chip can hold 8 bits of data. To find the number of locations within this memory chip, divide the capacity by the number of data pins. 512K/8 = 64K; therefore, the organization for this memory chip is 64K × 8
(b) The chip has 16 address lines since 216 = 64K

 ROM is a type of memory that does not lose its contents when the power is turned off.
 ROM is also called nonvolatile memory.
 There are different types of read-only memory

- PROM
- EPROM
- EEPROM
- Flash EPROM
- Mask ROM

 PROM is also referred to as OTP (one-time programmable)
 Programming ROM, also called burning ROM, requires special equipment called a ROM burner or ROM programmer
 RAM memory is called volatile memory since cutting off the power to the IC will result in the loss of data.
 Sometimes RAM is also referred to as RAWM (read and write memory), in contrast to ROM, which cannot be written to.

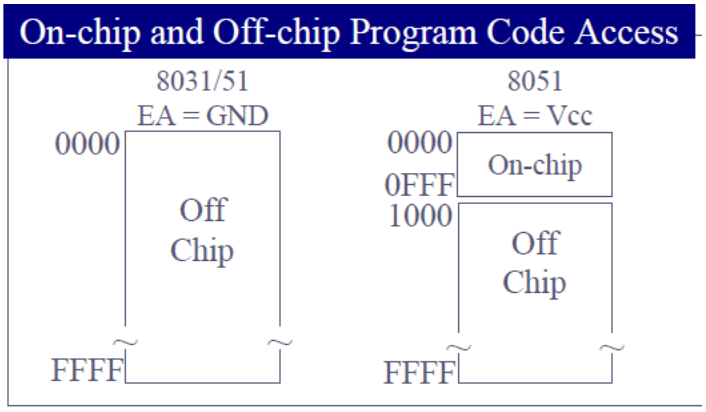
 There are three types of RAM

- Static RAM (SRAM)
- NV-RAM (nonvolatile RAM)
- Dynamic RAM (DRAM)

 Storage cells in static RAM memory are made of flip-flops and therefore do not require refreshing in order to keep their data.
 NV-RAM combines the best of RAM and ROM.
 The read and write ability of RAM, plus the nonvolatility of ROM.
 Dynamic RAM uses a capacitor to store each bit.
 In the ROM, note that all of them have 8 pins for data.
 This is not the case for DRAM memory chips, which can have any of the x1, x4, x8, x16 organizations
 The CPU provides the address of the data desired, but it is the job of the decoding circuitry to locate the selected memory block
 Memory chips have one or more pins called CS (chip select), which must be activated for the memory's contents to be accessed
 Sometimes the chip select is also referred to as chip enable (CE).
 In connecting a memory chip to the CPU, note the following points

- The data bus of the CPU is connected directly to the data pins of the memory chip
- Control signals RD (read) and WR (memory write) from the CPU are connected to the OE (output enable) and WE (write enable) pins of the memory chip
- In the case of the address buses, while the lower bits of the address from the CPU go directly to the memory chip address pins, the upper ones are used to activate the CS pin of the memory chip

 The 8031 chip is a ROMless version of the 8051.
 It is exactly like any member of the 8051 family as far as executing the instructions and features are concerned, but it has no on-chip ROM.
 To make the 8031 execute 8051 code, it must be connected to external ROM memory containing the program code.
 8031 is ideal for many systems where the on-chip ROM of 8051 is not sufficient, since it allows the program size to be as large as 64K bytes.
 For 8051-based system, we connected the EA pin to Vcc to indicate that the program code is stored in the microcontroller's on-chip ROM. To indicate that the program code is stored in external ROM, this pin must be connected to GND.

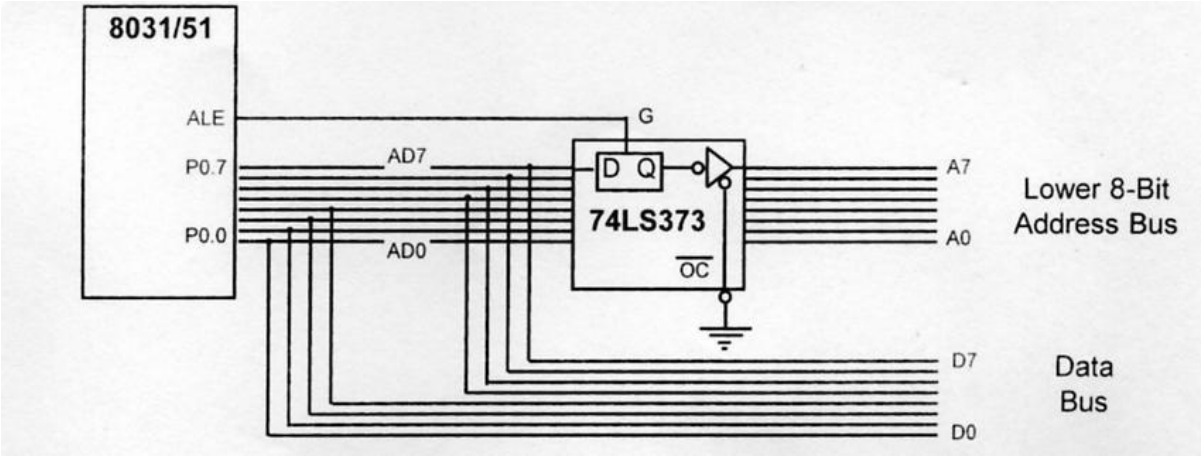


 Since the PC (program counter) of the 8031/51 is 16-bit, it is capable of accessing up to 64K bytes of program code
 In the 8031/51, port 0 and port 2 provide the 16-bit address to access external memory
 P0 provides the lower 8 bit address A0 – A7, and P2 provides the upper 8 bit address A8 – A15
 P0 is also used to provide the 8-bit data bus D0 – D7
 P0.0 – P0.7 are used for both the address and data paths.
 ALE (address latch enable) pin is an output pin for 8031/51

- ALE = 0, P0 is used for data path
- ALE = 1, P0 is used for address path

 To extract the address from the P0 pins we connect P0 to a 74LS373 and use the ALE pin to latch the address.

Address/Data De-Multiplexing

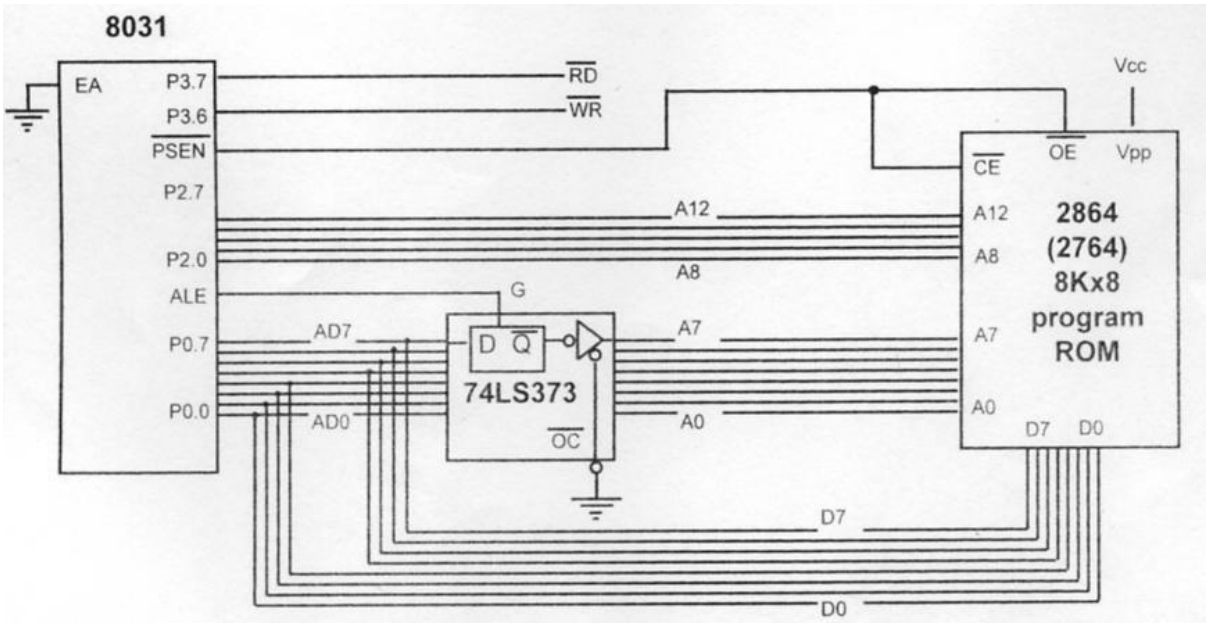


 When the EA pin is connected to GND, the 8031/51 fetches opcode from external ROM by using PSEN
 PSEN (program store enable) signal is an output signal for the 8031/51 microcontroller and must be connected to the OE pin of a ROM containing the program code.
 The 8051 has 128K bytes of address space.
 64K bytes are set aside for program code.
 Program space is accessed using the program counter (PC) to locate and fetch instructions.

 In some example we placed data in the code space and used the instruction MOVC A,@A+DPTR to get data, where C stands for code.
 The other 64K bytes are set aside for data.
 The data memory space is accessed using the DPTR register and an instruction called MOVX, where X stands for external.

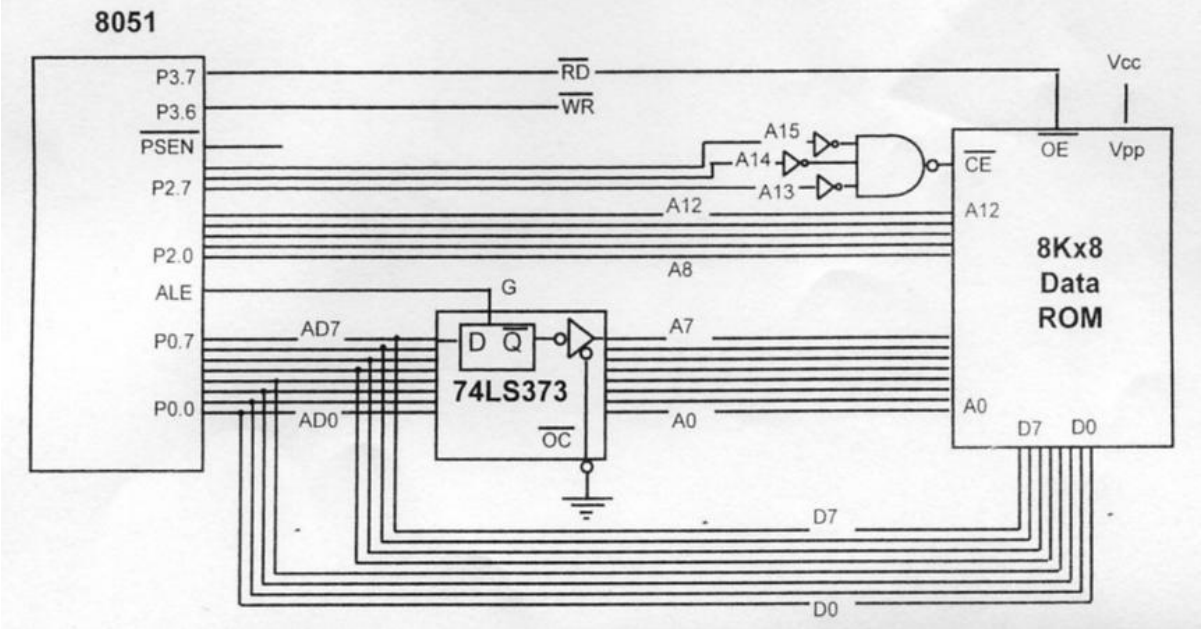
- If the memory access is for a byte of program code in the ROM, the PSEN (program store enable) pin will go low to enable the ROM to place a byte of program code on the data bus. If the access is for a RAM byte, the WR (write) or RD (read) pins will go low, enabling data to flow between the RAM and the data bus.

Connection to External Program ROM

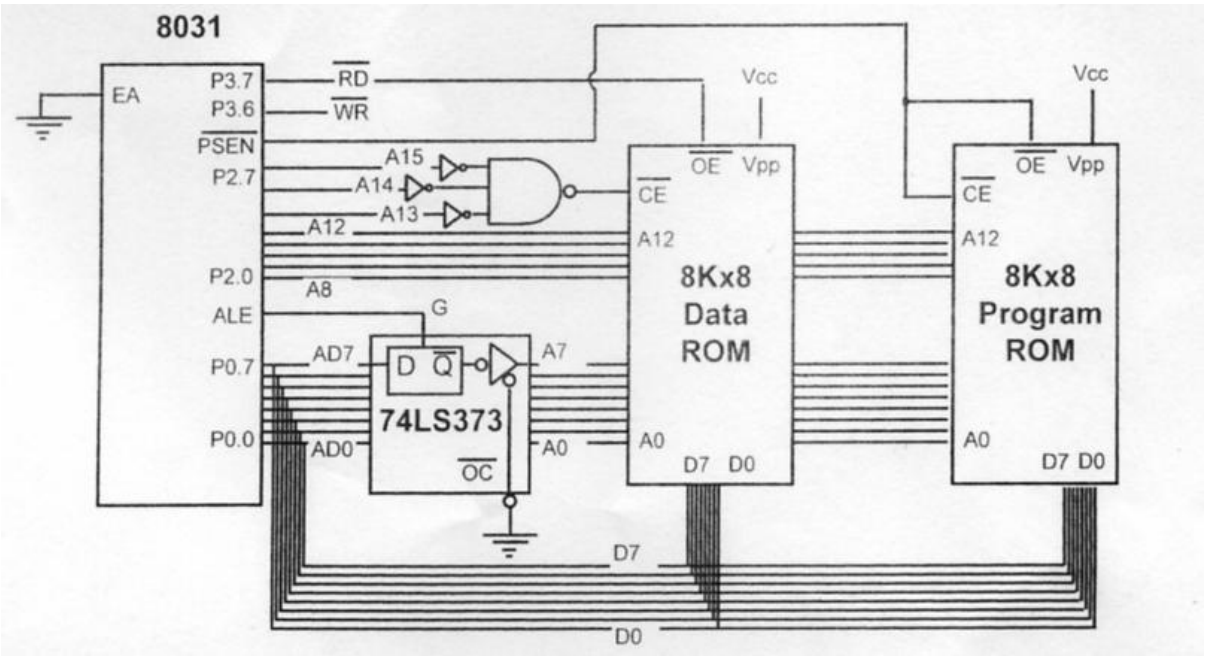


 We use RD to connect the 8031/51 to external ROM containing data.
 For the ROM containing the program code, PSEN is used to fetch the code.

8051 Connection to External Data ROM

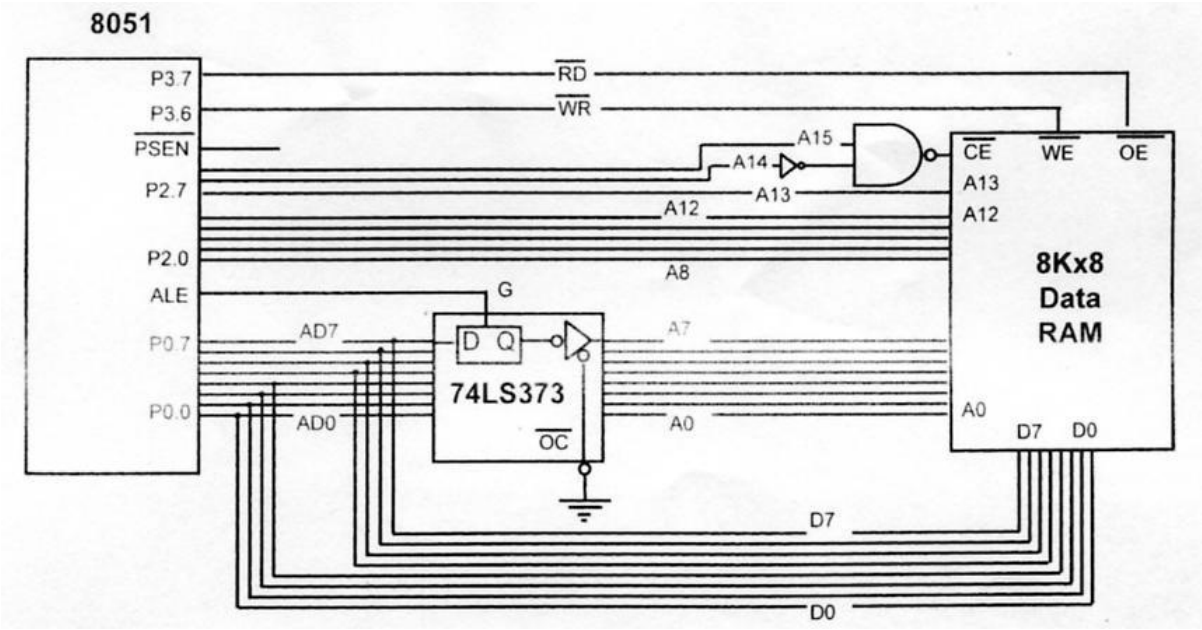


8031 Connection to External Data ROM and External Program ROM



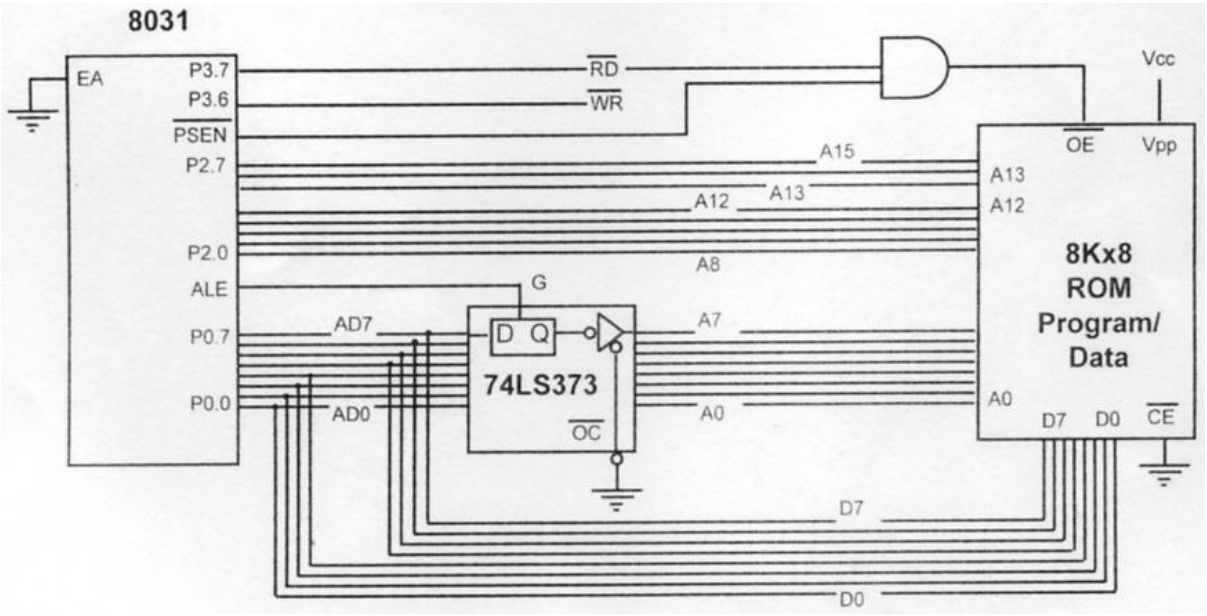
 MOVX is a widely used instruction allowing access to external data memory space
 To bring externally stored data into the CPU, we use the instruction
MOVX A,@DPTR
 To connect the 8051 to an external SRAM, we must use both RD (P3.7) and WR (P3.6).

8051 Connection to External Data RAM



 In writing data to external data RAM, we use the instruction MOVX @DPTR,A
 To allow a single ROM chip to provide both program code space and data space, we use an AND gate to signal the OE pin of the ROM chip

A Single ROM for both Program and Data



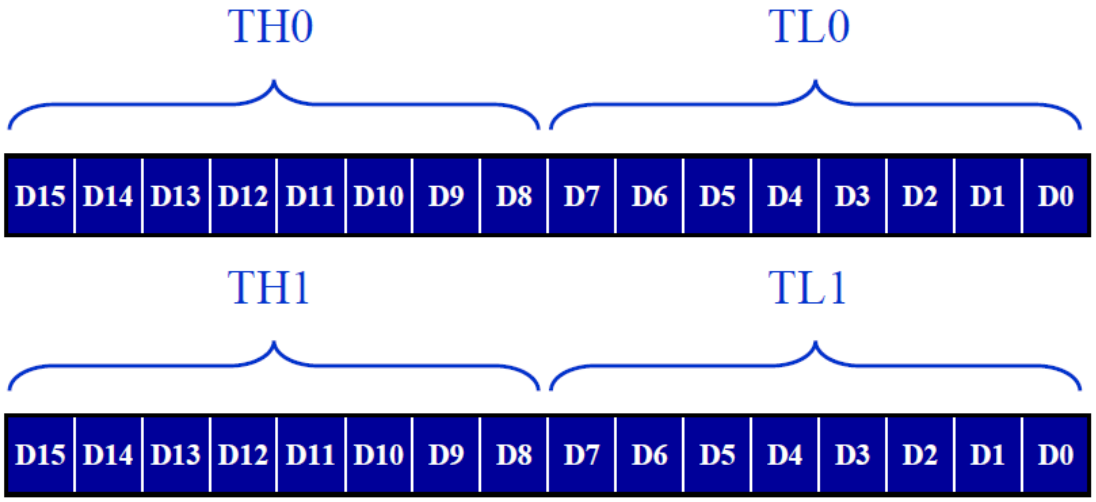
8051 Timers

The 8051 has two timers/counters, Timer 0 and Timer 1. They can be used either as

- Timers to generate a time delay or as
- Event counters to count events happening outside the microcontroller

Both Timer 0 and Timer 1 are 16 bits wide. Since 8051 has an 8-bit architecture, each 16-bit timer is accessed as two separate registers of low byte and high byte.

- 16-bit Register is accessed as low byte and high byte
- The low byte register is called TL0/TL1 and
- The high byte register is called TH0/TH1



They are accessed like any other register

- MOV TL0,#4FH
- MOV R5,TH0

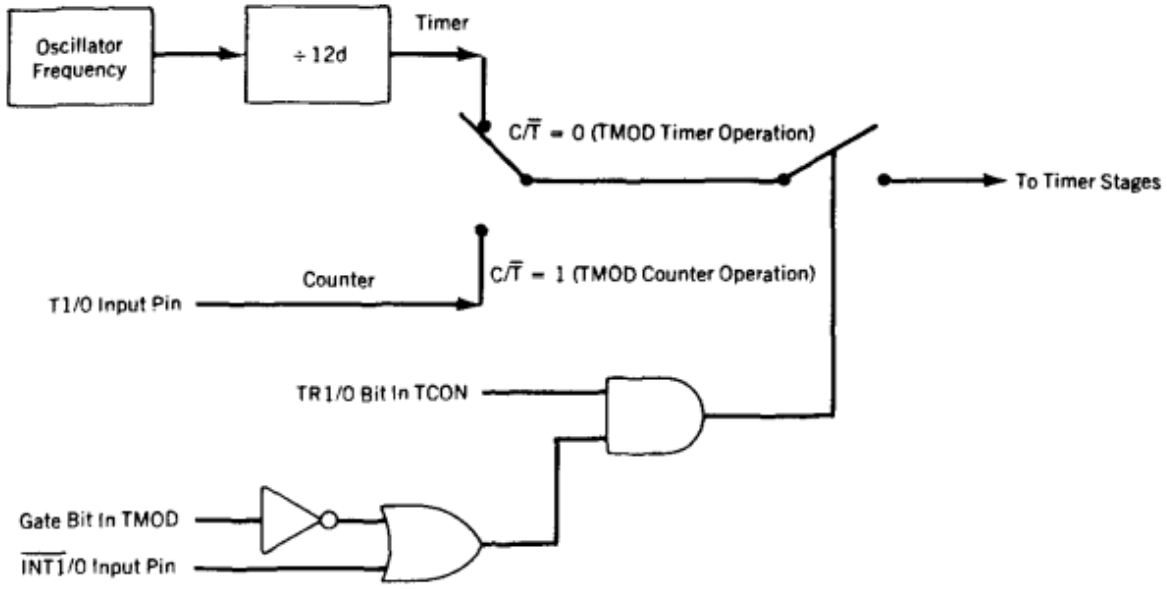
All counter action is controlled by bit states in the timer mode control register (TMOD), the timer/counter control register (TCON), and certain program instructions.

TMOD is dedicated solely to the two timers and can be considered to be two duplicate 4-bit registers, each of which controls the action of one of the timers.

TCON has control bits and flags for the timers in the upper nibble, and control bits and flags for the external interrupts in the lower nibble.

Timer special function registers

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



Timer/Counter Control Logic

TMOD register

Both timers 0 and 1 use the same register, called TMOD (timer mode), to set the various timer operation modes

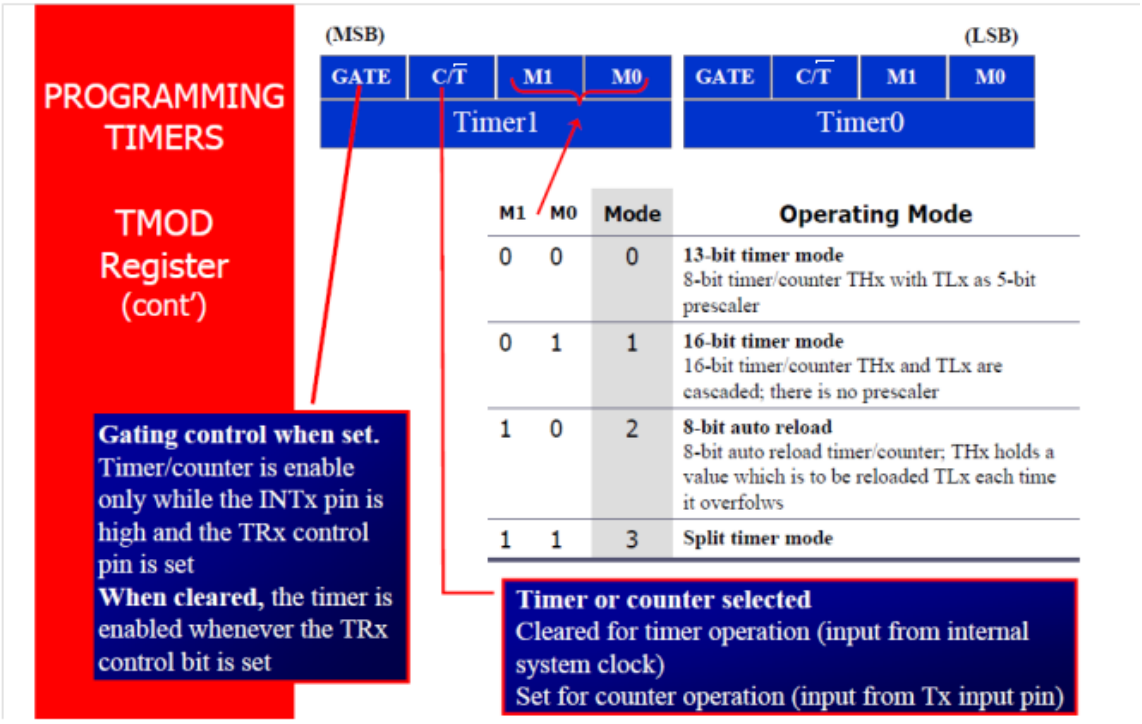
- TMOD is a 8-bit register
- The lower 4 bits are for Timer 0
- The upper 4 bits are for Timer 1

In each case,

- The lower 2 bits are used to set the timer mode
- The upper 2 bits to specify the operation

SFR Addresss for TMOD register = 89H

(MSB)							(LSB)
GATE	C/T	M1	M0	GATE	C/T	M1	M0
Timer1				Timer0			



Timers of 8051 do starting and stopping by either software or hardware control

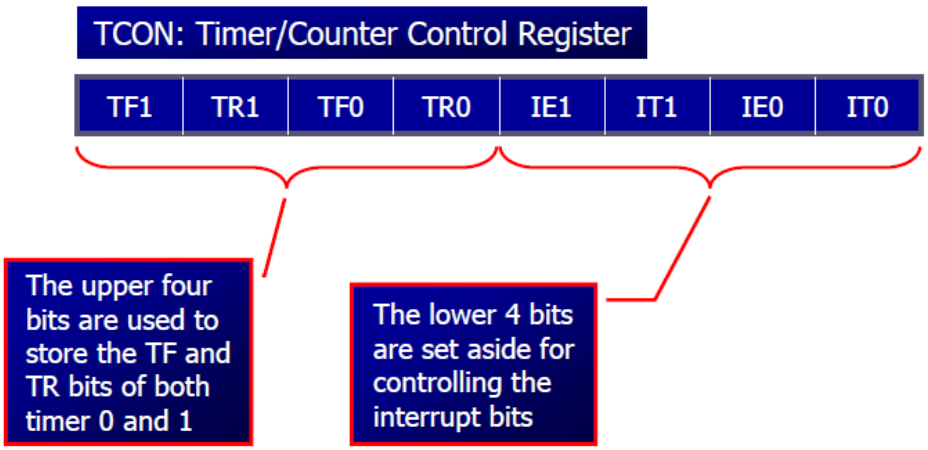
- In using software to start and stop the timer where GATE=0
- The start and stop of the timer are controlled by way of software by the TR (timer start) bits TR0 and TR1

– The SETB instruction starts it, and it is stopped by the CLR instruction

– These instructions start and stop the timers as long as GATE=0 in the TMOD register

– The hardware way of starting and stopping the timer by an external source is achieved by making GATE=1 in the TMOD register.

- TCON (timer control) register is an 8-bit register



7	6	5	4	3	2	1	0
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

THE TIMER CONTROL (TCON) SPECIAL FUNCTION REGISTER

Bit	Symbol	Function
7	TF1	Timer 1 Overflow flag. Set when timer rolls from all ones to zero. Cleared when processor vectors to execute interrupt service routine located at program address 001Bh.
6	TR1	Timer 1 run control bit. Set to 1 by program to enable timer to count; cleared to 0 by program to halt timer. Does not reset timer.
5	TF0	Timer 0 Overflow flag. Set when timer rolls from all ones to zero. Cleared when processor vectors to execute interrupt service routine located at program address 000Bh.
4	TR0	Timer 0 run control bit. Set to 1 by program to enable timer to count; cleared to 0 by program to halt timer. Does not reset timer.
3	IE1	External interrupt 1 edge flag. Set to 1 when a high to low edge signal is received on port 3 pin 3.3 ($\overline{INT1}$). Cleared when processor vectors to interrupt service routine located at program address 0013h. Not related to timer operations.
2	IT1	External interrupt 1 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 interrupt 1 to generate an interrupt.
1	IE0	External interrupt 0 edge flag. Set to 1 when a high to low edge signal is received on port 3 pin 3.2 ($\overline{INT0}$). Cleared when processor vectors to interrupt service routine located at program address 0003h. Not related to timer operations.
0	IT0	External interrupt 0 signal type control bit. Set to 1 by program to enable external interrupt 0 to be triggered by a falling edge signal. Set to 0 by program to enable a low level signal on external interrupt 0 to generate an interrupt.

Bit addressable as TCON.0 to TCON.7

- If GATE = 1, the start and stop of the timer are done externally through pins P3.2 and P3.3 for timers 0 and 1, respectively
- This hardware way allows to start or stop the timer externally at any time via a simple switch

13-Bit Timer Mode (Mode 0)

Mode 0 is a 13-bit timer mode that provides compatibility with the 8051's predecessor, the 8048. It is not generally used in new designs. (See Figure) The timer high-byte (THx) is cascaded with the five least-significant bits of the timer low-byte (TLx) to form a 13-bit timer. The upper three bits of TLx are not used.

16-Bit Timer Mode (Mode 1)

Mode 1 is a 16-bit timer mode and is the same as mode 0, except the timer is operating as a full 16-bit timer.

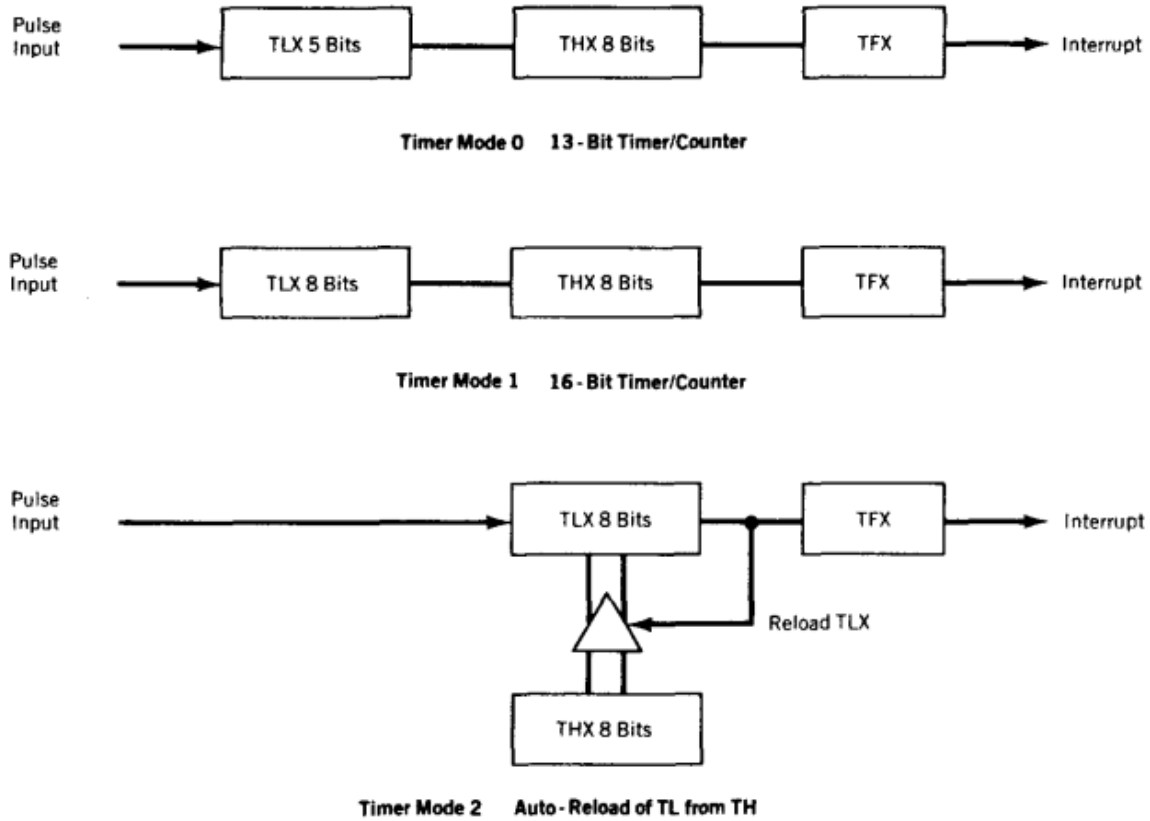
The clock is applied to the combined high and low timer registers (TLx/THx).

As clock pulses are received, the timer counts up: 0000H, 0001H, 0002H, etc.
An overflow occurs on the 0FFFFH-to-0000H transition of the count and sets the timer overflow flag. The timer continues to count.
The overflow flag is the TFx bit in TCON that is read or written by software.
The most-significant bit (MSB) of the value in the timer registers is THx bit 7, and the least-significant bit (LSB) is TLx bit 0.
The LSB toggles at the input clock frequency divided by 2, while the MSB toggles at the input clock frequency divided by 65,536 (i.e., 2^{16}.) .
The timer registers (TLx/THx) may be read or written at any time by software.

8-Bit Auto-Reload Mode (Mode 2)

Mode 2 is 8-bit auto-reload mode. The timer low-byte (TLx) operates as an 8-bit timer while the timer high-byte (THx) holds a reload value.
When the count overflows from 0FFH, not only is the timer flag set, but the value in THx is loaded into TLx; counting continues from this value up to the next 0FFH overflow, and so on. If TLx contains 4FH, for example, the time counts continuously from 4FH to 0FFH.

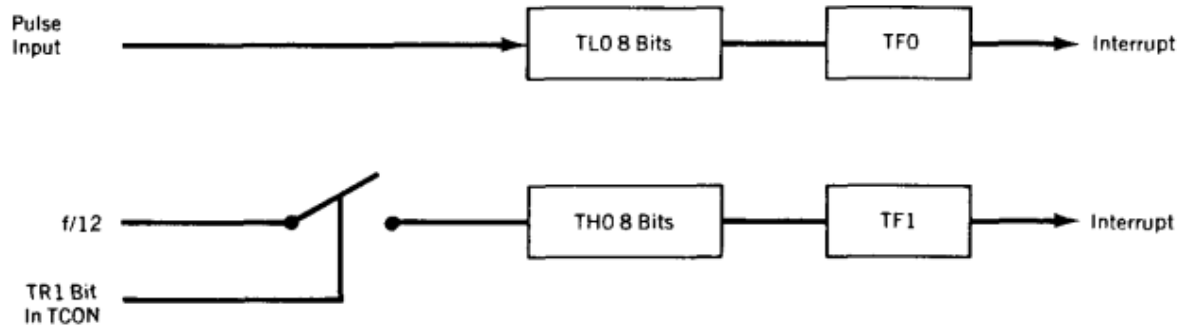
FIGURE Timer 1 and Timer 0 Operation Modes



Split Timer Mode (Mode 3)

Mode 3 is the split timer mode and is different for each timer.
Timer 0 in mode 3 is split into two 8-bit timers. TL0 and TH0 act as separate timers with overflows setting the TF0 and TF1 bits, respectively.

Timer 1 is stopped in mode 3 but can be started by switching it into one of the other modes. The only limitation is that the usual Timer 1 overflow flag, TF1, is not affected by Timer 1 overflows, since it is connected to TH0.



Timer Mode 3 **Two 8 - Bit Timers Using Timer 0**

Mode 1 Programming

The following are the characteristics and operations of mode1:

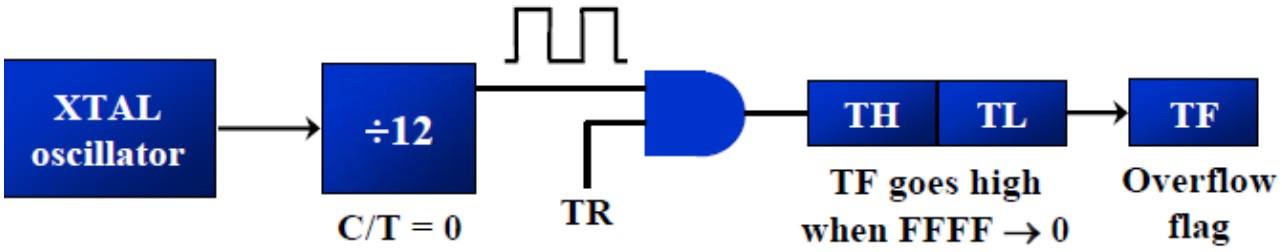
1. It is a 16-bit timer; therefore, it allows value of 0000 to FFFFH to be loaded into the timer's register TL and TH.
2. After TH and TL are loaded with a 16-bit initial value, the timer must be started.
   - This is done by SETB TR0 for timer 0 and SETB TR1 for timer 1
3. After the timer is started, 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

   - 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



To generate a time delay, using the Timer's mode-1

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.

To calculate the values to be loaded into the TL and TH registers, look at the following example

- Assume XTAL = 11.0592 MHz, we can use the following steps for finding the TH, TL registers' values

1. Divide the desired time delay by 1.085 us
2. Perform 65536 – n, where n is the decimal value we got in Step1
3. Convert the result of Step2 to hex, where yyxx is the initial hex value to be loaded into the timer's register
4. Set TL = xx and TH = yy

Example 9-4

In the following program, we create a square wave of 50% duty cycle (with equal portions high and low) on the P1.5 bit. Timer 0 is used to generate the time delay. Analyze the program.

```
        MOV TMOD,#01        ;Timer 0, mode 1(16-bit mode)
HERE:   MOV TL0,#0F2H       ;TL0=F2H, the low byte
        MOV TH0,#0FFH       ;TH0=FFH, the high byte
        CPL P1.5            ;toggle P1.5
        ACALL DELAY
        SJMP HERE
DELAY:
        SETB TR0            ;start the timer 0
AGAIN:  JNB TF0,AGAIN       ;monitor timer flag 0 until it rolls over
        CLR TR0             ;stop timer 0
        CLR TF0             ;clear timer 0 flag
        RET
```

In the above program notice the following step.
1. TMOD is loaded.
2. FFF2H is loaded into TH0-TL0.
3. P1.5 is toggled for the high and low portions of the pulse.

4. The DELAY subroutine using the timer is ''called''.
5. In the DELAY subroutine, timer 0 is started by the SETB TR0 instruction.
6. Timer 0 counts up with the passing of each clock, which is provided by the crystal oscillator. As the timer counts up, it goes through the states of FFF3,FFF4, FFF5, FFF6, FFF7, FFF8, FFF9, FFFA, FFFB, and so on until it reaches FFFFH. One more clock rolls it to 0, raising the timer flag (TF0=1).
At that point, the JNB instruction falls through.
7. Timer 0 is stopped by the instruction CLR TR0. The DELAY subroutine ends, and the process is repeated.
Notice that to repeat the process, we must reload the TL and TH registers, and start the process is repeated.

Example 9-5

In Example 9-4, calculate the amount of time delay in the DELAY subroutine generated by the timer. Assume XTAL = 11.0592 MHz.

Solution:

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 1.085us resulting in delay = number of counts × 1.085us.
The number of counts for the roll over is FFFFH – FFF2H = 0DH (13 decimal). However, we add one to 13 because of the extra clock needed when it rolls over from FFFF to 0 and raise the TF flag. This gives 14 × 1.085 µs = 15.19 µs for half the pulse. For the entire period it is T = 2 × 15.19 µs = 30.38 µs as the time delay generated by the timer.

Example 9-6

In Example 9-5, calculate the frequency of the square wave generated on pin P1.5.

Solution:

In the timer delay calculation of Example 9-5, we did not include the overhead due to instruction in the loop. To get a more accurate timing, we need to add clock cycles due to this instruction in the loop. To do that, we use the machine cycle, as shown below.

		Cycles
HERE:	MOV TL0,#0F2H	2
	MOV TH0,#0FFH	2
	CPL P1.5	1
	ACALL DELAY	2
	SJMP HERE	2
DELAY:		
	SETB TR0	1
AGAIN:	JNB TF0,AGAIN	14
	CLR TR0	1

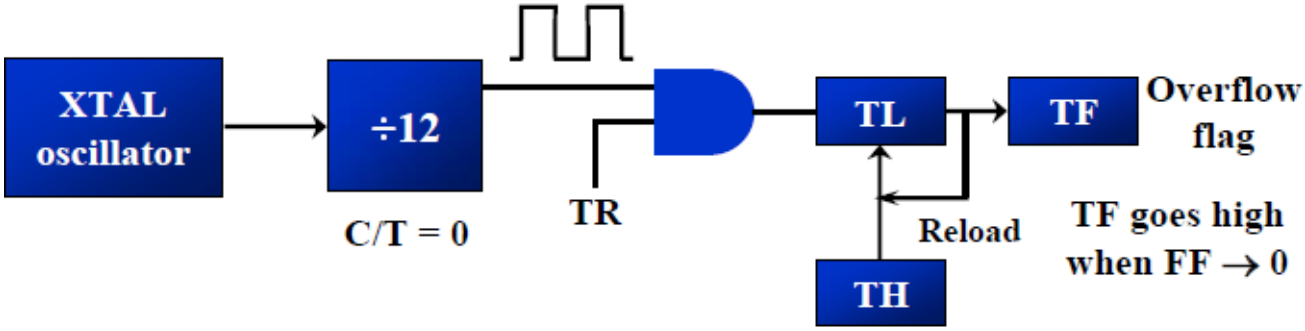
CLR TF0	1
RET	2
Total 28	

T = 2 × 28 × 1.085 µs = 60.76µs and F=16458.2 Hz.

Mode 2 Programming

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 and let it go without any need by the programmer to reload the original value
   - This makes mode 2 an auto-reload, in contrast with mode 1 in which the programmer has to reload TH and TL



To generate a time delay, using the Timer's mode-2

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 auto-reload

Example 9-14

Assume XTAL = 11.0592 MHz, find the frequency of the square wave generated on pin P1.0 in the following program.

```
        MOV TMOD,#20H       ;T1/8-bit/auto reload
        MOV TH1,#5          ;TH1 = 5
        SETB TR1            ;start the timer 1
BACK:   JNB TF1,BACK        ;till timer rolls over
        CPL P1.0            ;P1.0 to high, low
        CLR TF1             ;clear Timer 1 flag
        SJMP BACK           ;mode 2 is auto-reload
```

Solution:

First notice the target address of SJMP. In mode 2 we do not need to reload TH since it is auto-reload. Now (256 - 05) × 1.085 µs = 251 × 1.085 µs = 272.33 µs is the high portion of the pulse. Since it is a 50% duty cycle square wave, the period T is twice that; as a result T = 2 × 272.33 µs = 544.67 µs and the frequency = 1.83597 kHz

Example 9-16

Assuming that we are programming the timers for mode 2, find the value (in hex) loaded into TH for each of the following cases.

(a) MOV TH1,#-200 (b) MOV TH0,#-60 (c) MOV TH1,#-3 (d) MOV TH1,#-12 (e) MOV TH0,#-48

Solution:

You can use the Windows scientific calculator to verify the result provided by the assembler. In Windows calculator, select decimal and enter 200. Then select hex, then +/- to get the TH value. Remember that we only use the right two digits and ignore the rest since our data is an 8-bit data.

Decimal	**2's complement (TH value)**
-3	FDH
-12	F4H
-48	D0H
-60	C4H
-200	38H

Counter

The only difference between counting and timing is the source of the clock pulses to the counters.

When used as a timer, the clock pulses are sourced from the oscillator through the divide-by-12d circuit.

When used as a counter, pin T0 (P3.4) supplies pulses to counter 0, and pin T1 (P3.5) to counter 1.

The C/T bit in TMOD must be set to 1 to enable pulses from the TX pin to reach the control circuit.

The input pulse on T_X is sampled during P2 of state 5 every machine cycle. A change on the input from high to low between samples will increment the counter. Each high and low state of the input pulse must thus be held constant for at least one machine cycle to ensure reliable counting.

Since this takes 24 pulses, the maximum input frequency that can be accurately counted is the oscillator frequency divided by 24. For 6 megahertz crystal, the calculation yields a maximum external frequency of 250 kilohertz.

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

Example 9-18

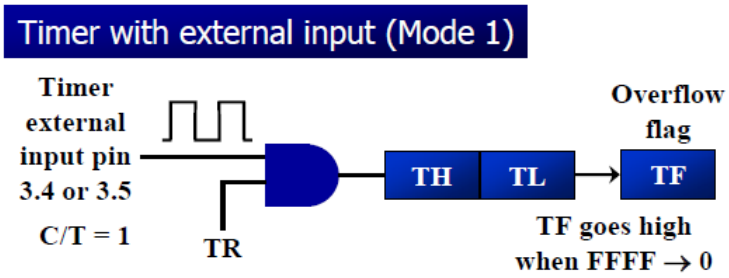
Assuming that clock pulses are fed into pin T1, write a program for counter 1 in mode 2 to count the pulses and display the state of the TL1 count on P2, which connects to 8 LEDs.

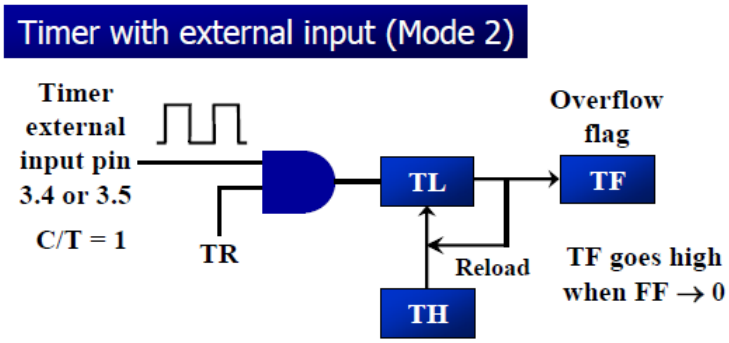
Solution:

```
        MOV TM0D,#01100000B     ;counter 1, mode 2, C/T=1 external
    pulses
        MOV TH1,#0              ;clear TH1
        SETB P3.5               ;make T1 input
AGAIN:  SETB TR1                ;start the counter
BACK:   MOV A,TL1               ;get copy of TL
        MOV P2,A                ;display it on port 2
        JNB TF1,Back            ;keep doing, if TF = 0
        CLR TR1                 ;stop the counter 1
        CLR TF1                 ;make TF=0
        SJMP AGAIN              ;keep doing it
```

Notice in the above program the role of the instruction SETB P3.5. Since ports are set up for output when the 8051 is powered up, we make P3.5 an input port by making it high. In

other words, we must configure (set high) the T1 pin (pin P3.5) to allow pulses to be fed into it.





To speed up the 8051, many recent versions of the 8051 have reduced the number of clocks per machine cycle from 12 to four, or even one

 The frequency for the timer is always 1/12th the frequency of the crystal attached to the 8051, regardless of the 8051 version.

Example 9-11

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

Solution:

Look at the following steps.

(a) T = 1 / f = 1 / 2 kHz = 500 µs the period of square wave.

(b) 1 / 2 of it for the high and low portion of the pulse is 250 µs.

(c) 250 µs / 1.085 µs = 230 and 65536 – 230 = 65306 which in hex is FF1AH.

(d) TL = 1A and TH = FF, all in hex. The program is as follow.

```
        MOV TMOD,#01          ;Timer 0, 16-bitmode
AGAIN:  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
        CPL P1.5              ;Complement port1.5 pin
```

```
CLR TF1                 ;Clear timer 1 flag
SJMP AGAIN              ;Reload timer
```

8051 Serial ports

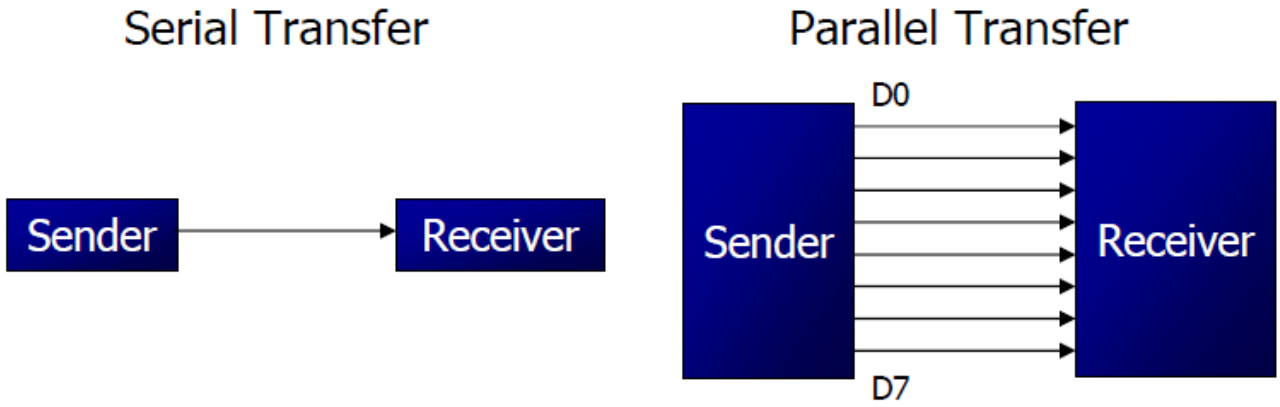
Computers transfer data in two ways:

 Parallel

- Often 8 or more lines (wire conductors) are used to transfer data to a device that is only a few feet away

 Serial

- To transfer to a device located many meters away, the serial method is used
- The data is sent one bit at a time



 At the transmitting end, the byte of data must be converted to serial bits using parallel-in-serial-out shift register

 At the receiving end, there is a serial in-parallel-out shift register to receive the serial data and pack them into byte

 When the distance is short, the digital signal can be transferred as it is on a simple wire and requires no modulation

 If data is to be transferred on the telephone line, it must be converted from 0s and 1s to audio tones

 This conversion is performed by a device called a modem, “Modulator/demodulator”

 Serial data communication uses two methods

- Synchronous method transfers a block of data at a time
- Asynchronous method transfers a single byte at a time

 It is possible to write software to use either of these methods, but the programs can be tedious and long

 There are special IC chips made by many manufacturers for serial communications

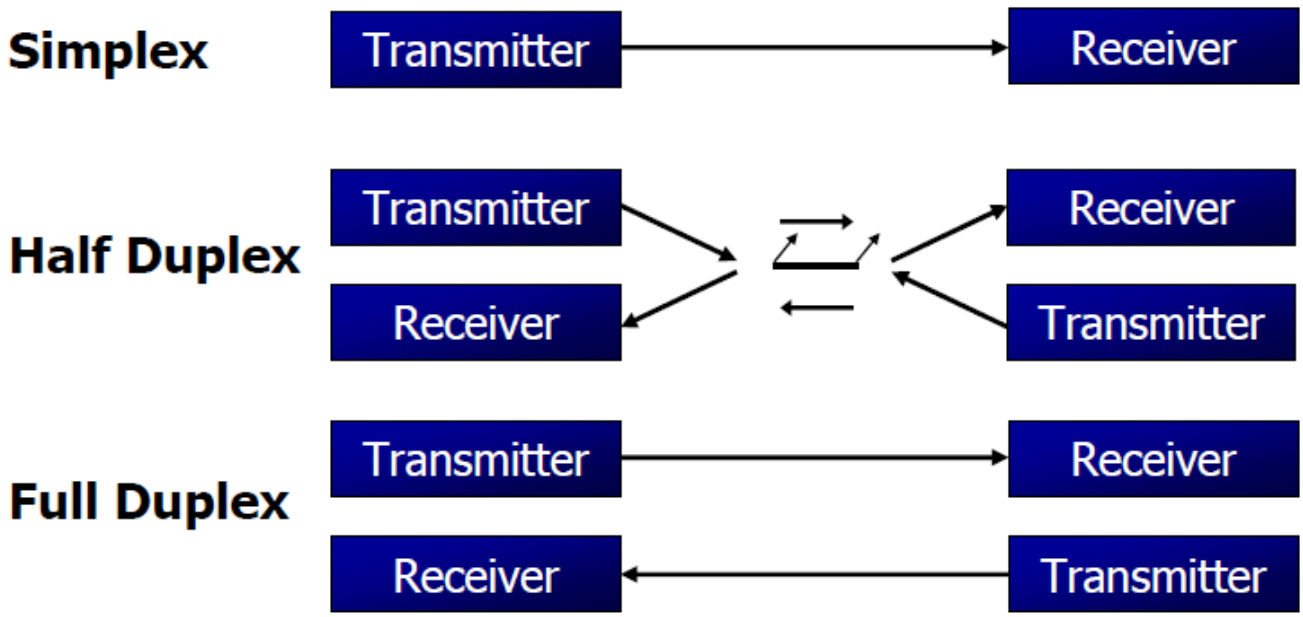
- UART (universal asynchronous Receiver transmitter)
- USART (universal synchronous-asynchronous Receiver-transmitter)

 If data can be transmitted and received, it is a duplex transmission

 If data transmitted one way a time, it is referred to as half duplex

 If data can go both ways at a time, it is full duplex

 This is contrast to simplex transmission



 A protocol is a set of rules agreed by both the sender and receiver on

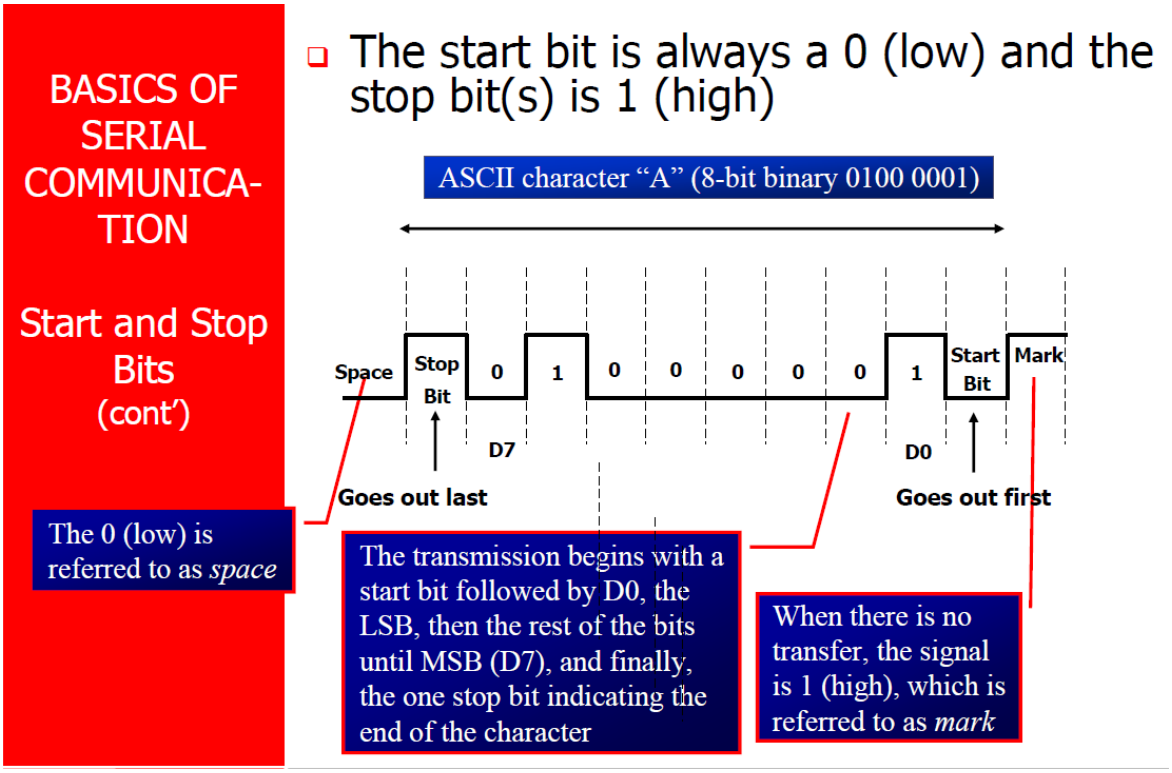
- How the data is packed
- How many bits constitute a character
- When the data begins and ends

 Asynchronous serial data communication is widely used for character-oriented transmissions.

 Each character is placed in between start and stop bits, this is called framing.

 The start bit is always one bit, but the stop bit can be one or two bits.

 Block-oriented data transfers use the synchronous method.



 Due to the extended ASCII characters, 8-bit ASCII data is common.

 In older systems, ASCII characters were 7-bit.

 In modern PCs the use of one stop bit is standard.

 In older systems, due to the slowness of the receiving mechanical device, two stop bits were used to give the device sufficient time to organize itself before transmission of the next byte.

 Assuming that we are transferring a text file of ASCII characters using 1 stop bit, we have a total of 10 bits for each character. This gives 25% overhead, i.e. each 8-bit character with an extra 2 bits.

 In some systems in order to maintain data integrity, the parity bit of the character byte is included in the data frame.

 UART chips allow programming of the parity bit for odd-, even-, and no-parity options.

 The rate of data transfer in serial data communication is stated in bps (bits per second).

 Another widely used terminology for bps is baud rate. It is modem terminology and is defined as the number of signal changes per second.

 In modems, there are occasions when a single change of signal transfers several bits of data.

 As far as the conductor wire is concerned, the baud rate and bps are the same, and we use the terms.

- Interchangeably The data transfer rate of given computer system depends on communication ports incorporated into that system

 IBM PC/XT could transfer data at the rate of 100 to 9600 bps

 Pentium-based PCs transfer data at rates as high as 56K bps

 In asynchronous serial data communication, the baud rate is limited to 100K bps

 An interfacing standard RS232 was set by the Electronics Industries Association (EIA) in 1960

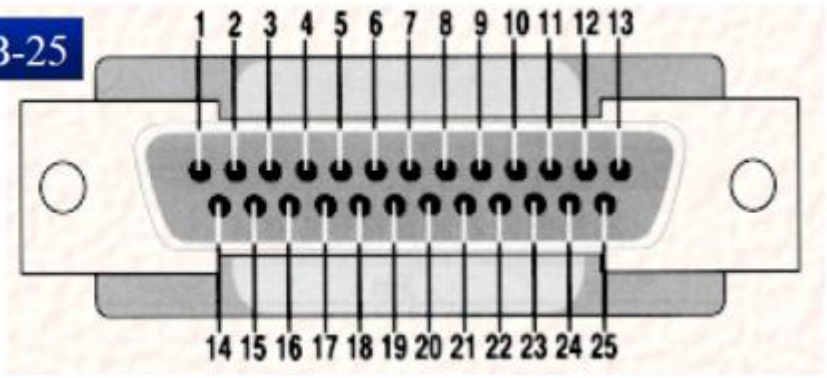
 The standard was set long before the advent of the TTL logic family, its input and output voltage levels are not TTL compatible

 In RS232, a 1 is represented by -3 ~ -25 V, while a 0 bit is +3 ~ +25 V, making -3 to +3 undefined

RS232 DB-25 Pins

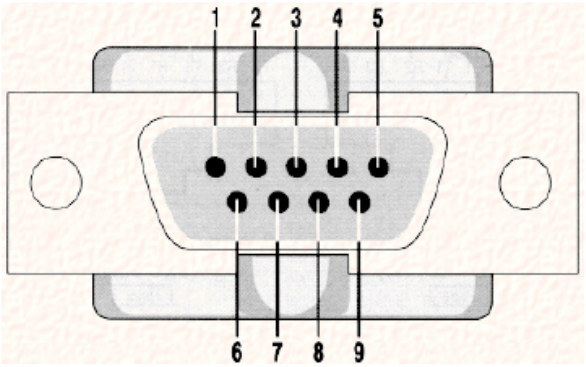
Pin	Description	Pin	Description
1	Protective ground	14	Secondary transmitted data
2	Transmitted data (TxD)	15	Transmitted signal element timing
3	Received data (RxD)	16	Secondary receive data
4	Request to send (-RTS)	17	Receive signal element timing
5	Clear to send (-CTS)	18	Unassigned
6	Data set ready (-DSR)	19	Secondary receive data
7	Signal ground (GND)	20	Data terminal ready (-DTR)
8	Data carrier detect (-DCD)	21	Signal quality detector
9/10	Reserved for data testing	22	Ring indicator (RI)
11	Unassigned	23	Data signal rate select
12	Secondary data carrier detect	24	Transmit signal element timing
13	Secondary clear to send	25	Unassigned

RS232 Connector DB-25



- Since not all pins are used in PC cables, IBM introduced the DB-9 version of the serial I/O standard

RS232 Connector DB-9



RS232 DB-9 Pins

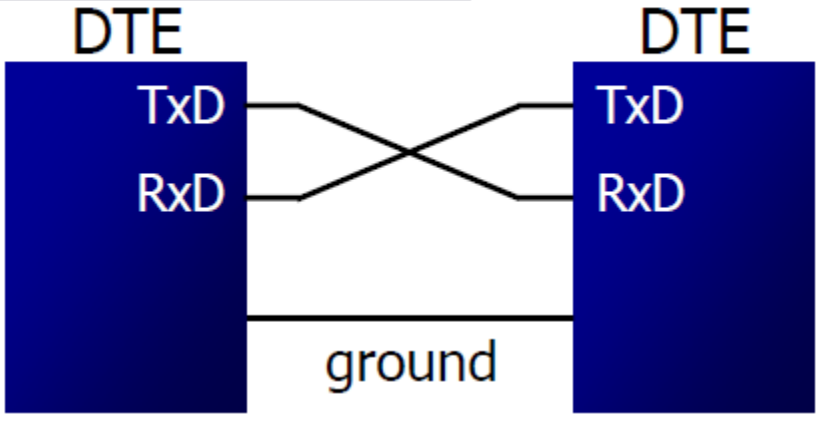
Pin	Description
1	Data carrier detect (-DCD)
2	Received data (RxD)
3	Transmitted data (TxD)
4	Data terminal ready (DTR)
5	Signal ground (GND)
6	Data set ready (-DSR)
7	Request to send (-RTS)
8	Clear to send (-CTS)
9	Ring indicator (RI)

 Current terminology classifies data communication equipment as

- DTE (data terminal equipment) refers to terminal and computers that send and receive data.
- DCE (data communication equipment) refers to communication equipment, such as modems.

 The simplest connection between a PC and microcontroller requires a minimum of three pins, TxD, RxD, and ground.

Null modem connection



 DTR (data terminal ready)

- When terminal is turned on, it sends out signal DTR to indicate that it is ready for communication

 DSR (data set ready)

- When DCE is turned on and has gone through the self-test, it assert DSR to indicate that it is ready to communicate.

 RTS (request to send)

- When the DTE device has byte to transmit, it assert RTS to signal the modem that it has a byte of data to transmit.

 CTS (clear to send)

- When the modem has room for storing the data it is to receive, it sends out signal CTS to DTE to indicate that it can receive the data now.

 DCD (data carrier detect)

- The modem asserts signal DCD to inform the DTE that a valid carrier has been detected and that contact between it and the other modem is established.

 RI (ring indicator)

- An output from the modem and an input to a PC indicates that the telephone is ringing
- It goes on and off in synchronous with the ringing sound.

A line driver such as the MAX232 chip is required to convert RS232 voltage levels to TTL levels, and vice versa

 8051 has two pins that are used specifically for transferring and receiving data serially

 These two pins are called TxD and RxD and are part of the port 3 group (P3.0 and P3.1)

 These pins are TTL compatible; therefore, they require a line driver to make them RS232 compatible

 We need a line driver (voltage converter) to convert the R232's signals to TTL voltage levels that will be acceptable to 8051's TxD and RxD pins.

 To allow data transfer between the PC and an 8051 system without any error, we must make sure that the baud rate of 8051 system matches the baud rate of the PC's COM port

PC Baud Rates
110
150
300
600
1200
2400
4800
9600
19200

Baud rates supported by 486/Pentium IBM PC BIOS

SERIAL COMMUNICA-TION PROGRAMMING (cont')

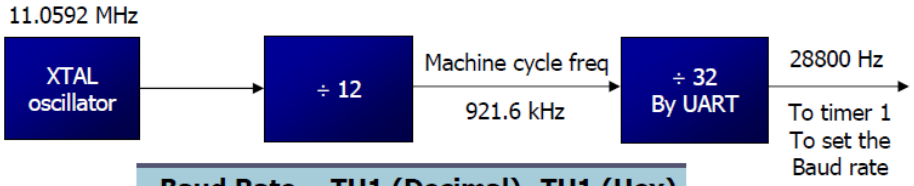
With XTAL = 11.0592 MHz, find the TH1 value needed to have the following baud rates. (a) 9600 (b) 2400 (c) 1200

Solution:

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

(a) 28,800 / 3 = 9600 where -3 = FD (hex) is loaded into TH1
(b) 28,800 / 12 = 2400 where -12 = F4 (hex) is loaded into TH1
(c) 28,800 / 24 = 1200 where -24 = E8 (hex) is loaded into TH1

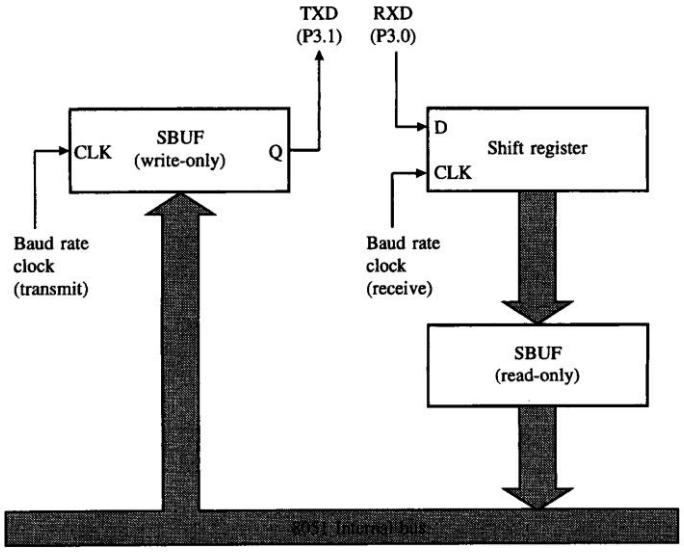
Notice that dividing 1/12 of the crystal frequency by 32 is the default value upon activation of the 8051 RESET pin.



TF is set to 1 every 12 ticks, so it functions as a frequency divider

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

- The 8051 has a serial data communication circuit that uses register SBUF to hold data. Register SCON controls data communication, register PCON controls data rates, and pins RXD (P3.0) and TXD (P3.1) connect to the serial data network.
- The serial port buffer register (SBUF) at address 99H 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.



- There are four programmable modes for serial data communication that are chosen by setting the SMX bits in SCON.

 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.

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

- The serial port control register (SCON) at address 98H is a bit-addressable register containing status bits and control bits.
- Status bits indicate the end of a character transmission or reception and are tested in software or programmed to cause an interrupt. Meanwhile, writing to the control bits would set the operating mode for the 8051 serial port.

 SM0, SM1

- They determine the framing of data by specifying the number of bits per character, and the start and stop bits

SM0	SM1	
0	0	Serial Mode 0
0	**1**	**Serial Mode 1, 8-bit data, 1 stop bit, 1 start bit**
1	0	Serial Mode 2
1	1	Serial Mode 3

Only mode 1 is of interest to us

SM2

- This enables the multiprocessing capability of the 8051

REN (receive enable)

- When it is high, it allows 8051 to receive data on RxD pin
- If low, the receiver is disable

TI (transmit interrupt)

- When 8051 finishes the transfer of 8-bit character, It raises TI flag to indicate that it is ready to transfer another byte
- TI bit is raised at the beginning of the stop bit

RI (receive interrupt)

- When 8051 receives data serially via RxD, it gets rid of the start and stop bits and places the byte in SBUF register
- It raises the RI flag bit to indicate that a byte has been received and should be picked up before it is lost
- RI is raised halfway through the stop bit.

7	6	5	4	3	2	1	0
SM0	SM1	SM2	REN	TB8	RB8	TI	RI

THE SERIAL PORT CONTROL (SCON) SPECIAL FUNCTION REGISTER

Bit	Symbol	Function
7	SM0	Serial port mode bit 0. Set/cleared by program to select mode.
6	SM1	Serial port mode bit 1. Set/cleared by program to select mode.

SM0	SM1	Mode	Description
0	0	0	Shift register; baud = f/12
0	1	1	8-bit UART; baud = variable
1	0	2	9-bit UART; baud = f/32 or f/64
1	1	3	9-bit UART; baud = variable

5	SM2	Multiprocessor communications bit. Set/cleared by program to enable multiprocessor communications in modes 2 and 3. When set to 1 an interrupt is generated if bit 9 of the received data is a 1; no interrupt is generated if bit 9 is a 0. If set to 1 for mode 1, no interrupt will be generated unless a valid stop bit is received. Clear to 0 if mode 0 is in use.
4	REN	Receive enable bit. Set to 1 to enable reception; cleared to 0 to dissable reception.
3	TB8	Transmitted bit 8. Set/cleared by program in modes 2 and 3.
2	RB8	Received bit 8. Bit 8 of received data in modes 2 and 3; stop bit in mode 1. Not used in mode 0.
1	TI	Transmit interrupt flag. Set to one at the end of bit 7 time in mode 0, and at the beginning of the stop bit for other modes. Must be cleared by the program.
0	RI	Receive interrupt flag. Set to one at the end of bit 7 time in mode 0, and halfway through the stop bit for other modes. Must be cleared by the program.

Bit addressable as SCON.0 to SCON.7

Serial data Transmission modes

Serial Data Mode 0—Shift Register Mode

Mode 0, selected by writing 0s into bits SM1 and SM0 of SCON, puts the serial port into 8-bit shift register mode. Serial data enter and exit through RXD, and TXD outputs the shift clock. Eight bits are transmitted or received with the least-significant (LSB) first.
The baud rate is fixed at 1/12th the on-chip oscillator frequency. The terms "RXD" and "TXD" are misleading in this mode. The RXD line is used for both data input and output, and the TXD line serves as the clock.

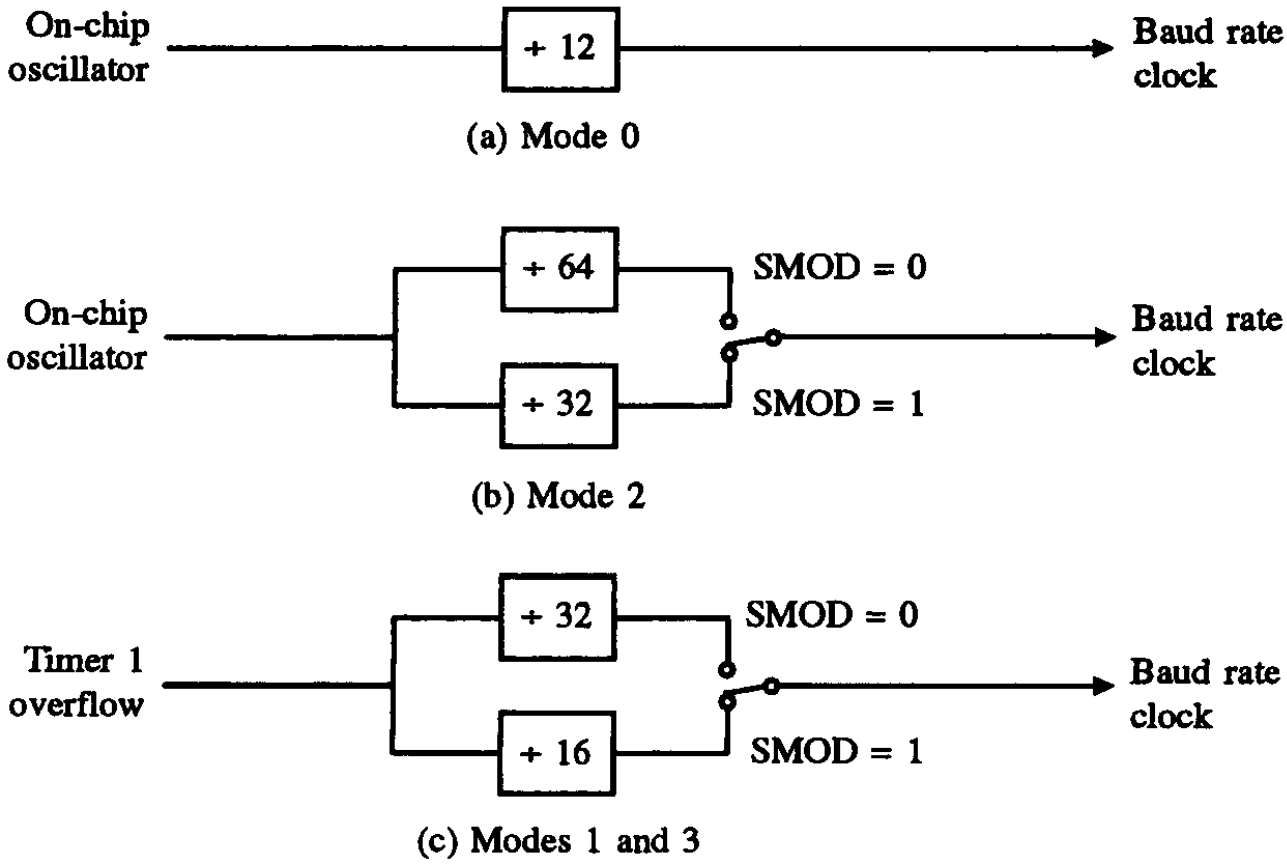


FIGURE
Serial port clocking sources (a) Mode 0 (b) Mode 2 (c) Modes 1 and 3

Serial Data Mode 1—Standard UART

When SM0 and SM1 are set to 01, SBUF becomes a 10-bit full-duplex receiver/ transmitter that may receive and transmit data at the same time. Pin RXD receives all data, and pin TXD transmits all data.

Transmitted data is sent as a start bit, eight data bits (Least Significant Bit, LSB, first), and a stop bit.

Interrupt flag TI is set once all ten bits have been sent. Each bit interval is the inverse of the baud rate frequency, and each bit is maintained high or low over that interval. Received data is obtained in the same order.

Data bits are shifted into the receiver at the programmed baud rate, and the data word will be loaded to SBUF if the following conditions are true: RI must be 0, and mode bit SM2 is 0 or the stop bit is 1 (the normal state of stop bits).

Of the original ten bits, the start bit is discarded, the eight data bits go to SBUF, and the stop bit is saved in bit RB8 of SCON. RI is set to 1, indicating a new data byte has been received.

If RI is found to be set at the end of the reception, indicating that the previously received data byte has not been read by the program, or if the other conditions listed are not true, the new data will not be loaded and will be lost.

RI set to 0 implies that the program has read the previous data byte and is ready to receive the next; a normal stop bit will then complete the transfer of data to SBUF regardless of the state of SM2.
SM2 set to 0 enables the reception of a byte with any stop bit state, a condition which is of limited use in this mode, but very useful in modes 2 and 3.
SM2 set to 1 forces reception of only "good" stop bits, an anti-noise safeguard.

Mode 1 Baud Rates

Timer 1 is used to generate the baud rate for mode 1 by using the overflow flag of the timer to determine the baud frequency.
Typically, timer 1 is used in timer mode 2 as an autoload 8-bit timer that generates the baud frequency:

$$f_{baud} = \frac{2^{SMOD}}{32d} \times \frac{\text{oscillator frequency}}{12d \times [256d - (TH1)]}$$

SMOD is the control bit in PCON and can be 0 or 1, which raises the 2 in the equation to a value of 1 or 2.
If timer 1 is not run in timer mode 2, then the baud rate is

$$f_{baud} = \frac{2^{SMOD}}{32d} \times (\text{timer 1 overflow frequency})$$

and timer 1 can be run using the internal clock or as a counter that receives clock pulses from any external source via pin T1.
The oscillator frequency is chosen to help generate both standard and nonstandard baud rates. If standard baud rates are desired, then an 11.0592 megahertz crystal could be selected.
To get a standard rate of 9600 hertz then, the setting of TH1 may be found as follows:

$$TH1 = 256d - \left(\frac{2^0}{32d} \times \frac{11.0592 \times 10^6}{12 \times 9600d}\right) = 253.0000d = 0FDh$$

if SMOD is cleared to 0.

Serial Data Mode 2—Multiprocessor Mode

Mode 2 is similar to mode I except 11 bits are transmitted: a start bit, nine data bits, and a stop bit.
The ninth data bit is gotten from bit TB8 in SCON during transmit and stored in bit RB8 of SCON when data is received. Both the start and stop bits are discarded.
The baud rate is programmed as follows:

$$f_{baud2} = \frac{2^{SMOD}}{64d} \times \text{oscillator frequency}$$

The conditions for setting RI for mode 2 are similar to mode 1: RI must be 0 before the last bit is received, and SM2 must be 0 or the ninth data bit must be a 1.

Serial Data Mode 3

Mode 3 is identical to mode 2 except that the baud rate is determined exactly as in mode 1, using Timer 1 to generate communication frequencies.

SERIAL COMMUNICA-TION PROGRAMMING

Doubling Baud Rate

- There are two ways to increase the baud rate of data transfer
  - To use a higher frequency crystal
  - To change a bit in the PCON register

The system crystal is fixed

- PCON register is an 8-bit register
  - When 8051 is powered up, SMOD is zero
  - We can set it to high by software and thereby double the baud rate

SMOD	--	--	--	GF1	GF0	PD	IDL

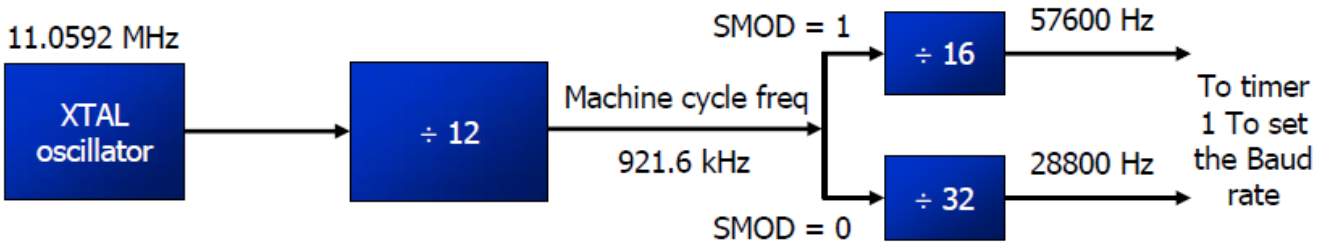
It is not a bit-addressable register

```
MOV  A,PCON     ;place a copy of PCON in ACC
SETB ACC.7      ;make D7=1
MOV  PCON,A     ;changing any other bits
```

7	6	5	4	3	2	1	0
SMOD	—	—	—	GF1	GF0	PD	IDL

THE POWER MODE CONTROL (PCON) SPECIAL FUNCTION REGISTER

Bit	Symbol	Function
7	SMOD	Serial baud rate modify bit. Set to 1 by program to double baud rate using timer 1 for modes 1, 2, and 3. Cleared to 0 by program to use timer 1 baud rate.
6-4	—	Not implemented.
3	GF1	General purpose user flag bit 1. Set/cleared by program.
2	GF0	General purpose user flag bit 0. Set/cleared by program.
1	PD	Power down bit. Set to 1 by program to enter power down configuration for CHMOS processors.
0	IDL	Idle mode bit. Set to 1 by program to enter idle mode configuration for CHMOS processors. PCON is not bit addressable.



Baud Rate comparison for SMOD=0 and SMOD=1

TH1	(Decimal)	(Hex)	SMOD=0	SMOD=1
	-3	FD	9600	19200
	-6	FA	4800	9600
	-12	F4	2400	4800
	-24	E8	1200	2400

TABLE 5-3
Baud rate summary

Baud Rate	Crystal Frequency	SMOD	TH1 Reload Value	Actual Baud Rate	Error
9600	12.000 MHz	1	-7 (0F9H)	8923	7%
2400	12.000 MHz	0	-13 (0F3H)	2404	0.16%
1200	12.000 MHz	0	-26 (0E6H)	1202	0.16%
19200	11.059 MHz	1	-3 (0FDH)	19200	0
9600	11.059 MHz	0	-3 (0FDH)	9600	0
2400	11.059 MHz	0	-12 (0F4H)	2400	0
1200	11.059 MHz	0	-24 (0E8H)	1200	0

- In programming the 8051 to transfer 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. 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

1. Write a program for the 8051 to transfer letter "A" serially at 4800 baud, continuously.

Solution:

```
        MOV TMOD,#20H       ;timer 1,mode 2(auto reload)
        MOV TH1,#-6         ;4800 baud rate
        MOV SCON,#50H       ;8-bit, 1 stop, REN enabled
        SETB TR1            ;start timer 1
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:

```
        MOV TMOD,#20H       ;timer 1,mode 2(auto reload)
        MOV TH1,#-3         ;9600 baud rate
        MOV SCON,#50H       ;8-bit, 1 stop, REN enabled
        SETB TR1            ;start timer 1
AGAIN:  MOV A,#"Y"          ;transfer "Y"
```

```
        ACALL TRANS
        MOV A,#"E"          ;transfer "E"
        ACALL TRANS
        MOV A,#"S"          ;transfer "S"
        ACALL TRANS
        SJMP AGAIN          ;keep doing it

;serial data transfer subroutine
TRANS:  MOV SBUF,A          ;load SBUF
HERE:   JNB TI,HERE         ;wait for the last bit
        CLR TI              ;get ready for next byte
        RET
```

- The steps that 8051 goes through in transmitting a character via TxD

1. The byte character to be transmitted is written into the SBUF register
2. The start bit is transferred
3. The 8-bit character is transferred on bit at a time
4. The stop bit is transferred
    - It is during the transfer of the stop bit that 8051 raises the TI flag, indicating that the last character was transmitted
5. By monitoring the TI flag, we make sure that we are not overloading the SBUF
    - If we write another byte into the SBUF before TI is raised, the untransmitted portion of the previous byte will be lost
6. After SBUF is loaded with a new byte, the TI flag bit must be forced to 0 by CLR TI in order for this new byte to be transferred.

- By checking the TI flag bit, we know whether or not the 8051 is ready to transfer another byte
    - It must be noted that TI flag bit is raised by 8051 itself when it finishes data transfer
    - It must be cleared by the programmer with instruction CLR TI
    - If we write a byte into SBUF before the TI flag bit is raised, we risk the loss of a portion of the byte being transferred
- The TI bit can be checked by
    - The instruction JNB TI,xx
    - Using an interrupt

- 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

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:

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

- In receiving bit via its RxD pin, 8051 goes through the following steps

1. It receives the start bit
   - Indicating that the next bit is the first bit of the character byte it is about to receive
2. The 8-bit character is received one bit at time
3. The stop bit is received
   - When receiving the stop bit 8051 makes RI = 1, indicating that an entire character byte has

been received and must be picked up before it gets overwritten by an incoming character

4. By checking the RI flag bit when it is raised, we know that a character has been received and is sitting in the SBUF register
   - We copy the SBUF contents to a safe place in some other register or memory before it is lost
5. After the SBUF contents are copied into a safe place, the RI flag bit must be forced to 0 by CLR RI in order to allow the next received character byte to be placed in SBUF
   - Failure to do this causes loss of the received Character
   - By checking the RI flag bit, we know whether or not the 8051 received a character byte
   - If we failed to copy SBUF into a safe place, we risk the loss of the received byte
   - It must be noted that RI flag bit is raised by 8051 when it finish receive data It must be cleared by the programmer with instruction CLR RI
   - If we copy SBUF into a safe place before the RI flag bit is raised, we risk copying garbage

The RI bit can be checked by

- The instruction JNB RI,xx
- Using an interrupt

8051 Interrupts

 An interrupt is an external or internal event that interrupts the microcontroller to inform it that a device needs its service.

 A single microcontroller can serve several devices by two ways

 Interrupts

- Whenever any device needs its service, the device notifies the microcontroller by sending it an interrupt signal
- Upon receiving an interrupt signal, the microcontroller interrupts whatever it is doing and serves the device
- The program which is associated with the interrupt is called the interrupt service routine (ISR) or interrupt handler

 Polling

- The microcontroller continuously monitors the status of a given device
- When the conditions met, it performs the service
- After that, it moves on to monitor the next device until every one is serviced
- Polling can monitor the status of several devices and serve each of them as certain conditions are met
- The polling method is not efficient, since it wastes much of the microcontroller's time by polling devices that do not need service

  ex. JNB TF,target

 The advantage of interrupts is that the microcontroller can serve many devices (not all at the same time). Each devices can get the attention of the microcontroller based on the assigned priority

 For the polling method, it is not possible to assign priority since it checks all devices in a round-robin fashion

 The microcontroller can also ignore (mask) a device request for service. This is not possible for the polling method.

- Software techniques use up processor time that could be devoted to other tasks; interrupts take processor time only when action by the program is needed. Most applications of microcontrollers involve responding to events quickly enough to control the environment that generates the events (generically termed "real-time programming"). Interrupts are often the only way in which real-time programming can be done successfully.
- For every interrupt, there must be an interrupt service routine (ISR), or interrupt handler. When an interrupt is invoked, the microcontroller runs the interrupt service routine. For every interrupt, there is a fixed location in memory that holds the address of its ISR. The group of memory locations set aside to hold the addresses of ISRs is called interrupt vector table.

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.

Six interrupts are allocated as follows

- Three of these are generated automatically by internal operations: timer flag 0, timer flag 1, and the serial port interrupt (RI or TI).
- Two interrupts are triggered by external signals provided by circuitry that is connected to pins INT0 and INT1 (port pins P3.2 and P3.3).
- Reset – power-up reset

Interrupt vector table

Interrupt	ROM Location (hex)	Pin
Reset	0000	9
External HW (INT0)	0003	P3.2 (12)
Timer 0 (TF0)	000B	
External HW (INT1)	0013	P3.3 (13)
Timer 1 (TF1)	001B	
Serial COM (RI and TI)	0023	

```
        ORG   0      ;wake-up ROM reset location
        LJMP MAIN    ;by-pass int. vector table
;---- the wake-up program
        ORG   30H
MAIN:
        ....
        END
```

Only three bytes of ROM space assigned to the reset pin. We put the LJMP as the first instruction and redirect the processor away from the interrupt vector table.

Reset

A reset can be considered to be the ultimate interrupt because the program may not block the action of the voltage on the RST pin. This type of interrupt is often called "non-maskable," since no combination of bits in any register can stop, or mask the reset action.
Whenever a high level is applied to the RST pin, the 8051 enters a reset condition.

- In order for the RESET input to be effective, it must have a minimum duration of 2 machine cycles
- In other words, the high pulse must be high for a minimum of 2 machine cycles before it is allowed to go low

After the RST pin is brought low, the internal registers will have the values shown in the following table:

REGISTER	**VALUE(HEX)**
PC	0
DPTR	0
A	0
B	0
SP	7
PSW	0
P0-3	FF
IP	XXX00000b
IE	0XX00000b
TCON	0
TMOD	0
TH0	0
TL0	0
TH1	0
TL1	0
SCON	0
SBUF	XX
PCON	0XXXXXXXb

- Internal RAM is not changed by a reset; however, the states of the internal RAM when power is first applied to the 8051 are random.
- Register bank 0 is selected upon reset as all Bits in PSW are 0.

Timer Flag Interrupt

When a timer/counter overflows, the corresponding timer flag, TF0 or TF1, is set to 1. The flag is cleared to 0 when the resulting interrupt generates a program call to the appropriate timer subroutine in memory.

Serial Port Interrupt

If a data byte is received, an interrupt bit, RI, is set to 1 in the SCON register. When a data byte has been transmitted an interrupt bit, TI, is set in SCON. These are ORed together to provide a single interrupt to the processor: the serial port interrupt. These bits are not cleared when the interrupt-generated program call is made by the processor. The program that handles serial data communication must reset RI or TI to 0 to enable the next data communication operation.

External Interrupts

Pins INT0 and INT1 are used by external circuitry. Inputs on these pins can set the interrupt flags IE0 and IE1 in the TCON register to 1 when a high-to-low transition takes place on the INTX pin.

Bits IT0 and IT1 in TCON program the INTX pins for low-level interrupt when set to 0 and program the INTX pins for transition interrupt when set to 1.

Flags IEX will be reset when a transition-generated interrupt is accepted by the processor and the interrupt subroutine is accessed.

It is the responsibility of the system designer and programmer to reset any level-generated external interrupts when they are serviced by the program. The external circuit must remove the low level before an RETI is executed. Failure to remove the low will result in an immediate interrupt after RETI, from the same source.

 Upon reset, all interrupts are disabled (masked), meaning that none will be responded to by the microcontroller if they are activated.

 The interrupts must be enabled by software in order for the microcontroller to respond to them.

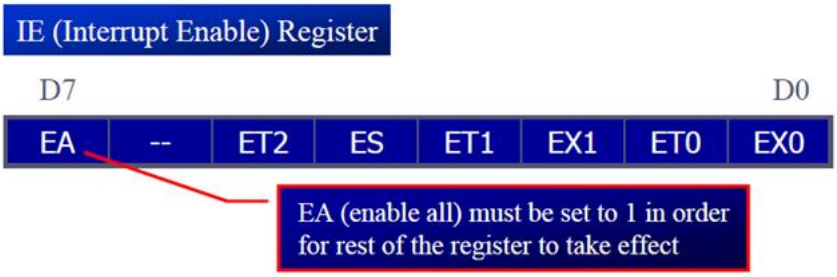
 There is a register called IE (interrupt enable) that is responsible for enabling (unmasking) and disabling (masking) the Interrupts.

Interrupt Control

The program must be able, at critical times, to inhibit the action of some or all of the interrupts so that crucial operations can be finished.

The IE register holds the programmable bits that can enable or disable all the interrupts as a group, or if the group is enabled, each individual interrupt source can be enabled or disabled. Often, it is desirable to be able to set priorities among competing interrupts that may conceivably occur simultaneously. The IP register bits may be set by the program to assign priorities among the various interrupt sources so that more important interrupts can be serviced first should two or more interrupts occur at the same time.

Interrupt Enable/Disable Bits in the El register are set to 1 if the corresponding interrupt source is to be enabled and set to 0 to disable the interrupt source. Bit EA is a master, or "global," bit that can enable or disable all of the interrupts.



EA	IE.7	Disables all interrupts when the bit =0
--	IE.6	Not implemented, reserved for future use
ET2	IE.5	Enables or disables timer 2 overflow or capture interrupt (8952)
ES	IE.4	Enables or disables the serial port interrupt
ET1	IE.3	Enables or disables timer 1 overflow interrupt
EX1	IE.2	Enables or disables external interrupt 1
ET0	IE.1	Enables or disables timer 0 overflow interrupt
EX0	IE.0	Enables or disables external interrupt 0

 To enable an interrupt, we take the following steps:

1. Bit D7 of the IE register (EA) must be set to high to allow the rest of register to take effect
2. The value of EA

 If EA = 1, interrupts are enabled and will be responded to if their corresponding bits in IE are high

 If EA = 0, no interrupt will be responded to, even if the associated bit in the IE register is High.

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:

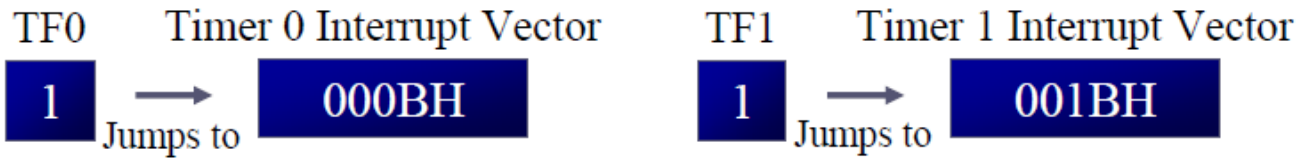
```
(a)     MOV IE,#10010110B       ;enable serial, ;timer 0, EX1
```

Another way to perform the same manipulation is

```
        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
(b)     CLR IE.1                ;mask (disable) timer 0 ;interrupt only
```

(c) CLR IE.7 ;disable all interrupts

The timer flag (TF) is raised when the timer rolls over
 In polling TF, we have to wait until the TF is raised
 The problem with this method is that the microcontroller is tied down while waiting for TF to be raised, and cannot do anything else
 Using interrupts solves this problem and, avoids tying down the controller
 If the timer interrupt in the IE register is enabled, whenever the timer rolls over, TF is raised, and the microcontroller is interrupted in whatever it is doing, and jumps to the interrupt vector table to service the ISR
 In this way, the microcontroller can do other until it is notified that the timer has rolled over.



Example 11-2

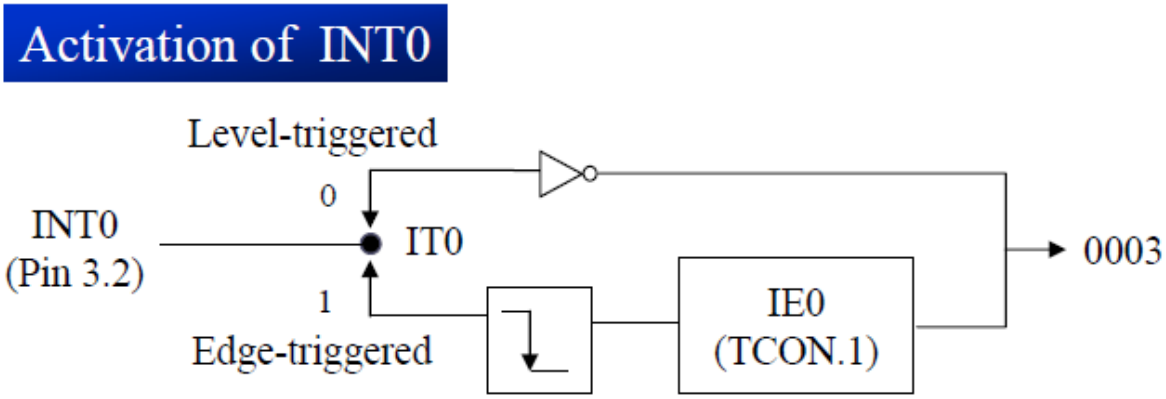
Write a program that continuously get 8-bit data from P0 and sends it to P1 while simultaneously creating a square wave of 200 µs period on pin P2.1. Use timer 0 to create the square wave. Assume that XTAL = 11.0592 MHz.
Solution:
We will use timer 0 in mode 2 (auto reload). TH0 = 100/1.085 µs = 92

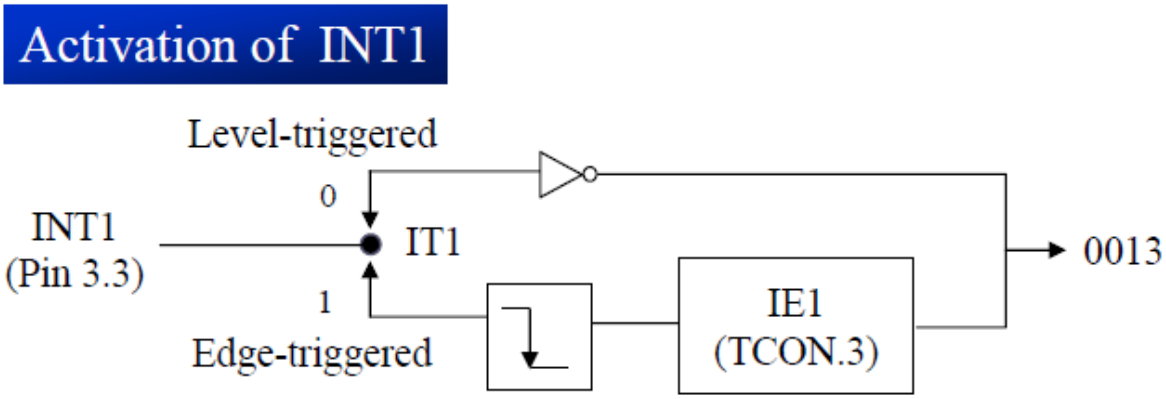
```
                                ;--upon wake-up go to main, avoid using
                                ;memory allocated to Interrupt Vector Table

ORG 0000H
LJMP MAIN                       ;by-pass interrupt vector table
                                ;--ISR for timer 0 to generate square wave
ORG 000BH                       ;Timer 0 interrupt vector table
CPL P2.1                        ;toggle P2.1 pin
RETI                            ;return from ISR
                                ;--The main program for initialization
ORG 0030H                       ;after vector table space
MAIN:  MOV TMOD,#02H            ;Timer 0, mode 2
       MOV P0,#0FFH             ;make P0 an input port
       MOV TH0,#-92             ;TH0=A4H for -92
       MOV IE,#82H              ;IE=10000010 (bin) enable ;Timer 0
       SETB TR0                 ;Start Timer 0
BACK:  MOV A,P0                 ;get data from P0
       MOV P1,A                 ;issue it to P1
```

```
        SJMP BACK           ;keep doing it loop unless interrupted by TF0
        END
```

 The 8051 has two external hardware interrupts
 Pin 12 (P3.2) and pin 13 (P3.3) of the 8051, designated as INT0 and INT1, are used as external hardware interrupts
 The interrupt vector table locations 0003H and 0013H are set aside for INT0 and INT1
 There are two activation levels for the external hardware interrupts

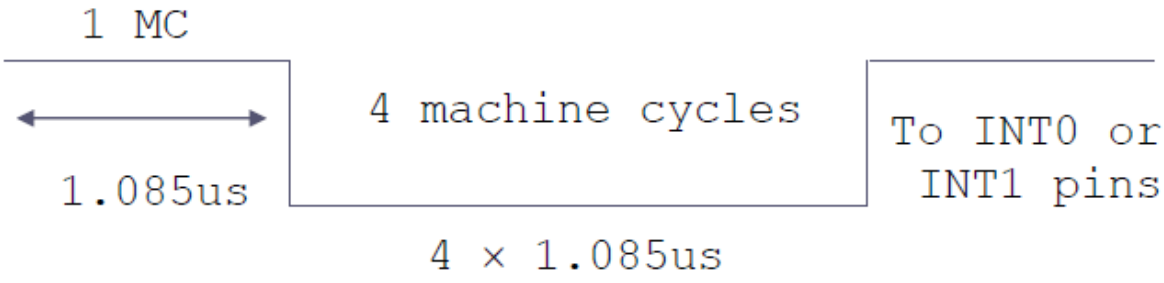
- Level trigged
- Edge trigged





 In the level-triggered mode, INT0 and INT1 pins are normally high
 If a low-level signal is applied to them, it triggers the interrupt
 Then the microcontroller stops whatever it is doing and jumps to the interrupt vector table to service that interrupt
 The low-level signal at the INT pin must be removed before the execution of the last instruction of the ISR, RETI; otherwise, another interrupt will be generated
 This is called a level-triggered or level activated interrupt and is the default mode upon reset of the 8051

 Pins P3.2 and P3.3 are used for normal I/O unless the INT0 and INT1 bits in the IE register are enabled.
 After the hardware interrupts in the IE register are enabled, the controller keeps sampling the INTn pin for a low-level signal once each machine cycle.
 According to one manufacturer's data sheet, The pin must be held in a low state until the start of the execution of ISR.
 If the INTn pin is brought back to a logic high before the start of the execution of ISR there will be no interrupt.
 If INTn pin is left at a logic low after the RETI instruction of the ISR, another interrupt will be activated after one instruction is executed.
 To ensure the activation of the hardware interrupt at the INTn pin, make sure that the duration of the low-level signal is around 4 machine cycles, but no more.
 This is due to the fact that the level-triggered interrupt is not latched.
 Thus the pin must be held in a low state until the start of the ISR execution.



note: On reset, IT0 (TCON.0) and IT1 (TCON.2) are both low, making external interrupt level-triggered

 To make INT0 and INT1 edge triggered interrupts, we must program the bits of the TCON register
 The TCON register holds, among other bits, the IT0 and IT1 flag bits that determine level- or edge-triggered mode of the hardware interrupt
 IT0 and IT1 are bits D0 and D2 of the TCON register.
 They are also referred to as TCON.0 and TCON.2 since the TCON register is bit addressable.

TCON (Timer/Counter) Register (Bit-addressable)

D7 D0

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

TF1	TCON.7	Timer 1 overflow flag. Set by hardware when timer/counter 1 overflows. Cleared by hardware as the processor vectors to the interrupt service routine
TR1	TCON.6	Timer 1 run control bit. Set/cleared by software to turn timer/counter 1 on/off
TF0	TCON.5	Timer 0 overflow flag. Set by hardware when timer/counter 0 overflows. Cleared by hardware as the processor vectors to the interrupt service routine
TR0	TCON.4	Timer 0 run control bit. Set/cleared by software to turn timer/counter 0 on/off

TCON (Timer/Counter) Register (Bit-addressable) (cont')

IE1	TCON.3	External interrupt 1 edge flag. Set by CPU when the external interrupt edge (H-to-L transition) is detected. Cleared by CPU when the interrupt is processed
IT1	TCON.2	Interrupt 1 type control bit. Set/cleared by software to specify falling edge/low-level triggered external interrupt
IE0	TCON.1	External interrupt 0 edge flag. Set by CPU when the external interrupt edge (H-to-L transition) is detected. Cleared by CPU when the interrupt is processed
IT0	TCON.0	Interrupt 0 type control bit. Set/cleared by software to specify falling edge/low-level triggered external interrupt

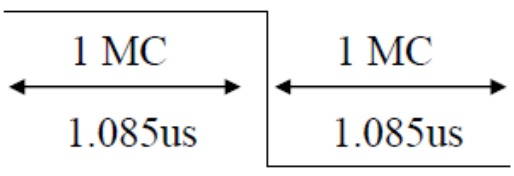
 In edge-triggered interrupts The external source must be held high for at least one machine cycle, and then held low for at least one machine cycle

 The falling edge of pins INT0 and INT1 are latched by the 8051 and are held by the TCON.1 and TCON.3 bits of TCON register.

 Function as interrupt-in-service flags

 It indicates that the interrupt is being serviced now and on this INTn pin, and no new interrupt will be responded to until this service is finished.

Minimum pulse duration to detect edge-triggered interrupts XTAL=11.0592MHz

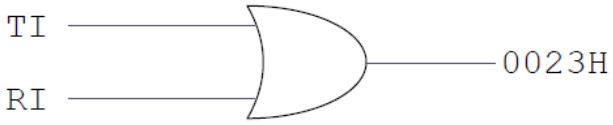


 Regarding the IT0 and IT1 bits in the TCON register, the following two points must be emphasized

 When the ISRs are finished (that is, upon execution of RETI), these bits (TCON.1 and TCON.3) are cleared, indicating that the interrupt is finished and the 8051 is ready to respond to another interrupt on that pin

 During the time that the interrupt service routine is being executed, the INTn pin is ignored, no matter how many times it makes a high-to-low transition
 RETI clears the corresponding bit in TCON register (TCON.1 or TCON.3)
 There is no need for instruction CLR TCON.1 before RETI in the ISR associated with INT0.

 TI (transfer interrupt) is raised when the last bit of the framed data, the stop bit, is transferred; indicating that the SBUF register is ready to transfer the next byte.
 RI (received interrupt) is raised when the entire frame of data, including the stop bit, is received
 In other words, when the SBUF register has a byte, RI is raised to indicate that the received byte needs to be picked up before it is lost (overrun) by new incoming serial 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



Serial interrupt is invoked by TI or RI flags

 The serial interrupt is used mainly for receiving data and is never used for sending data serially
 This is like getting a telephone call in which we need a ring to be notified
 If we need to make a phone call there are other ways to remind ourselves and there is no need for ringing
 However in receiving the phone call, we must respond immediately no matter what we are doing or we will miss the call
 The TCON register holds four of the interrupt flags, in the 8051 the SCON register has the RI and TI flags

Interrupt Flag Bits

Interrupt	Flag	SFR Register Bit
External 0	IE0	TCON.1
External 1	IE1	TCON.3
Timer 0	TF0	TCON.5
Timer 1	TF1	TCON.7
Serial Port	T1	SCON.1

 When the 8051 is powered up, the priorities are assigned according to the following
 In reality, the priority scheme is nothing but an internal polling sequence in which the 8051 polls the interrupts in the sequence listed and responds accordingly

Interrupt Priority Upon Reset

Highest To Lowest Priority	
External Interrupt 0	(INT0)
Timer Interrupt 0	(TF0)
External Interrupt 1	(INT1)
Timer Interrupt 1	(TF1)
Serial Communication	(RI + TI)

 In the 8051 a low-priority interrupt can be interrupted by a higher-priority interrupt but not by another low priority interrupt.
 Although all the interrupts are latched and kept internally, no low-priority interrupt can get the immediate attention of the CPU until the 8051 has finished servicing the high-priority interrupts.
 To test an ISR by way of simulation can be done with simple instructions to set the interrupts high and thereby cause the 8051 to jump to the interrupt vector table.
ex. If the IE bit for timer 1 is set, an instruction such as SETB TF1 will interrupt the 8051 in whatever it is doing and will force it to jump to the interrupt vector table. We do not need to wait for timer 1 go roll over to have an interrupt.

Interrupt Priority Register (Bit-addressable)

D7							D0
--	--	PT2	PS	PT1	PX1	PT0	PX0

--	IP.7	Reserved
--	IP.6	Reserved
PT2	IP.5	Timer 2 interrupt priority bit (8052 only)
PS	IP.4	Serial port interrupt priority bit
PT1	IP.3	Timer 1 interrupt priority bit
PX1	IP.2	External interrupt 1 priority bit
PT0	IP.1	Timer 0 interrupt priority bit
PX0	IP.0	External interrupt 0 priority bit

Priority bit=1 assigns high priority

 We can alter the sequence of interrupt priority by assigning a higher priority to any one of the interrupts by programming a register called IP (interrupt priority)

 Bits set to 1 give the accompanying interrupt a high priority while a 0 assigns a low priority. Interrupts with a high priority can interrupt another interrupt with a lower priority: the low priority interrupt continues after the higher is finished.

 If two interrupts with the same priority occur at the same time, then they have the following ranking:

1. IE0
2. TFO
3. 1E1
4. TF1
5. Serial = RI or TI

- The serial interrupt could be given the highest priority by setting the PS bit in IP to 1 , and all others to 0.