ISC 502
Applications of Microcontroller

Mrs. Kanchan Chavan

PROGRAMMING TIMERS

SFR RAM Address (Byte and Bit)

Bit addresses 80 – F7H belong to SFR of P0, TCON, P1, SCON, P2, etc

Byte address	Bit address	
FF		
F0	F7 F6 F5 F4 F3 F2 F1 F0	**B**
E0	E7 E6 E5 E4 E3 E2 E1 E0	**ACC**
D0	D7 D6 D5 D4 D3 D2 D1 D0	**PSW**
B8	-- -- -- BC BB BA B9 B8	**IP**
B0	B7 B6 B5 B4 B3 B2 B1 B0	**P3**
A8	AF AE AD AC AB AA A9 A8	**IE**
A0	A7 A6 A5 A4 A3 A2 A1 A0	**P2**
99	not bit addressable	**SBUF**

Byte address	Bit address	
98	9F 9E 9D 9C 9B 9A 99 98	**SCON**
90	97 96 95 94 93 92 91 90	**P1**
8D	not bit addressable	**TH1**
8C	not bit addressable	**TH0**
8B	not bit addressable	**TL1**
8A	not bit addressable	**TL0**
89	not bit addressable	**TMOD**
88	8F 8E 8D 8C 8B 8A 89 88	**TCON**
87	not bit addressable	**PCON**
83	not bit addressable	**DPH**
82	not bit addressable	**DPL**
81	not bit addressable	**SP**
80	87 86 85 84 83 82 81 80	**P0**

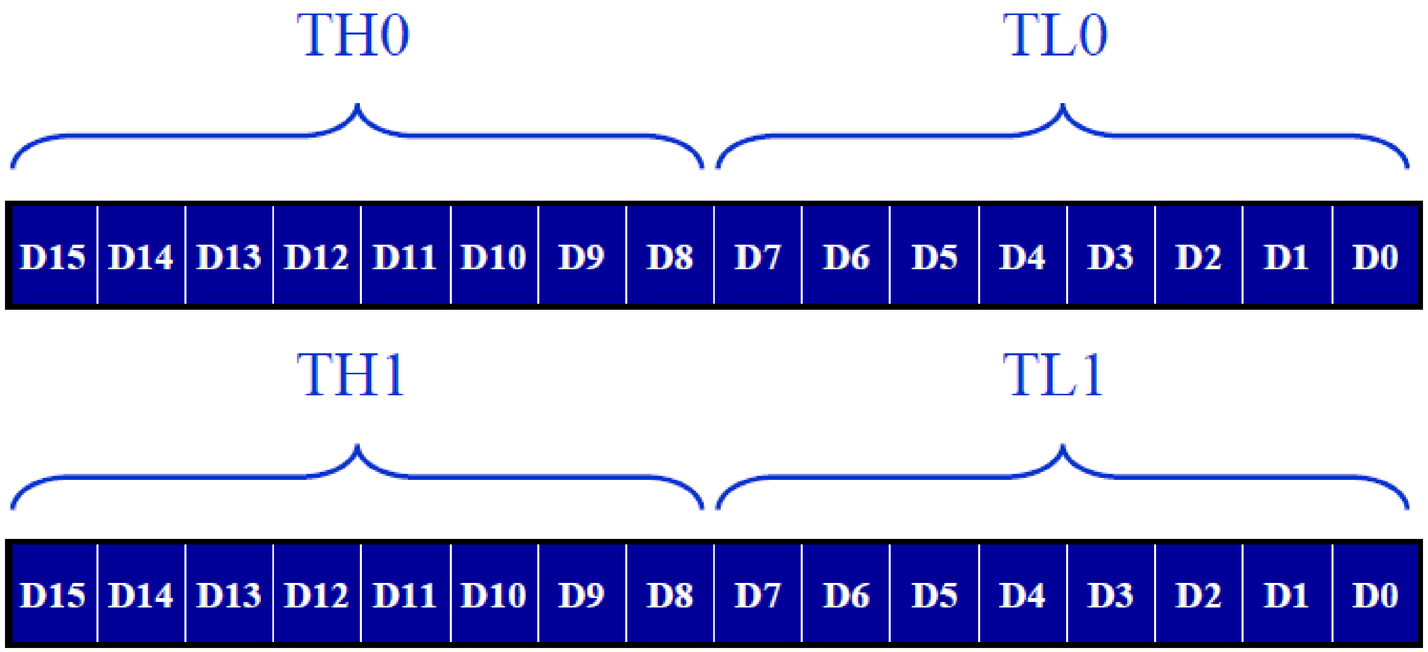
Special Function Register

Only registers A, B, PSW, IP, IE, ACC, SCON, and TCON are bit-addressable

- While all I/O ports are bit-addressable

- The 8051 has two timers/counters, 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-bits timer is accessed as two separate registers of low byte and high byte

- Accessed as low byte and high byte
  - The low byte register is called TL0/TL1 and
  - The high byte register is called TH0/TH1
  - Accessed like any other register
    - `MOV TL0,#4FH`
    - `MOV R5,TH0`



- 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

(MSB) (LSB)

GATE	C/T	M1	M0	GATE	C/T	M1	M0
Timer1				Timer0			

PROGRAMMING TIMERS

TMOD Register (cont’)

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

M1	M0	Mode	Operating Mode
0	0	0	**13-bit timer mode** 8-bit timer/counter THx with TLx as 5-bit prescaler
0	1	1	**16-bit timer mode** 16-bit timer/counter THx and TLx are cascaded; there is no prescaler
1	0	2	**8-bit auto reload** 8-bit auto reload timer/counter; THx holds a value which is to be reloaded TLx each time it overfolws
1	1	3	**Split timer mode**

Gating control when set. Timer/counter is enable only while the INTx pin is high and the TRx control pin is set
When cleared, the timer is enabled whenever the TRx control bit is set

Timer or counter selected
Cleared for timer operation (input from internal system clock)
Set for counter operation (input from Tx input pin)

Example 9-1

Indicate which mode and which timer are selected for each of the following.
(a) MOV TMOD, #01H (b) MOV TMOD, #20H (c) MOV TMOD, #12H

Solution:

We convert the value from hex to binary. From Figure 9-3 we have:

(a) TMOD = 00000001, mode 1 of timer 0 is selected.
(b) TMOD = 00100000, mode 2 of timer 1 is selected.
(c) TMOD = 00010010, mode 2 of timer 0, and mode 1 of timer 1 are selected.

If C/T = 0, it is used as a timer for time delay generation. The clock source for the time delay is the crystal frequency of the 8051

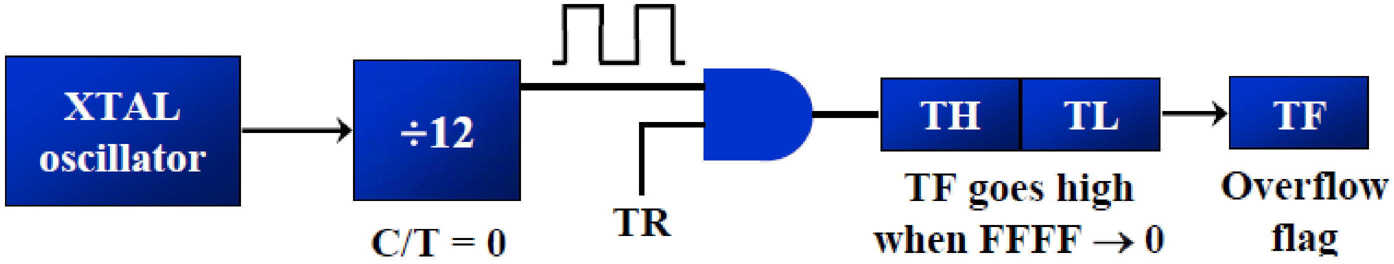
Example 9-2

Find the timer's clock frequency and its period for various 8051-based system, with the crystal frequency 11.0592 MHz when C/T bit of TMOD is 0.

Solution:

$1/12 \times 11.0529$ MHz = 921.6 MHz;
T = 1/921.6 kHz = 1.085 us

- 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

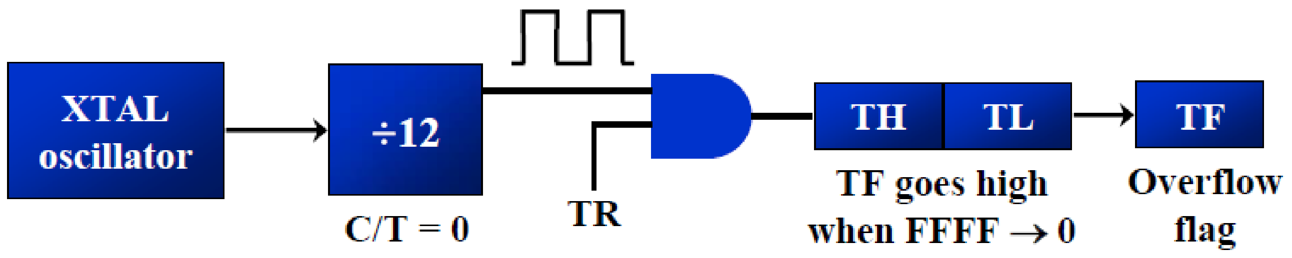


3. (cont')

- 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



TCON Register:

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

- TF1: Timer 1 overflow flag.
- TR1: Timer 1 run control bit.
- TF0: Timer 0 overflag.
- TR0: Timer 0 run control bit.
- IE1: External interrupt 1 edge flag.
- IT1: External interrupt 1 type flag.
- IE0: External interrupt 0 edge flag.
- IT0: External interrupt 0 type flag.

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
```

```
                                          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
```

$T = 2 \times 28 \times 1.085$ us = 60.76 us and F = 16458.2 Hz

Example 9-9

The following program generates a square wave on P1.5 continuously using timer 1 for a time delay. Find the frequency of the square wave if XTAL = 11.0592 MHz. In your calculation do not include the overhead due to Instructions in the loop.

```
       MOV   TMOD,#10;Timer 1, mod 1 (16-bitmode)
AGAIN: MOV   TL1,#34H ;TL1=34H, low byte of timer
       MOV   TH1,#76H ;TH1=76H, high byte timer
       SETB  TR1      ;start the timer 1
BACK:  JNB   TF1,BACK ;till timer rolls over
       CLR   TR1      ;stop the timer 1
       CPL   P1.5     ;comp. p1. to get hi, lo
       CLR   TF1      ;clear timer flag 1
       SJMP  AGAIN    ;is not auto-reload
```

Solution:

Since FFFFH – 7634H = 89CBH + 1 = 89CCH and 89CCH = 35276 clock count and 35276 $\times$ 1.085 us = 38.274 ms for half of the square wave. The frequency = 13.064Hz.

Also notice that the high portion and low portion of the square wave pulse are equal. In the above calculation, the overhead due to all the instruction in the loop is not included.

- 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 9-10

Assume that XTAL = 11.0592 MHz. What value do we need to load the timer's register if we want to have a time delay of 5 ms (milliseconds)? Show the program for timer 0 to create a pulse width of 5 ms on P2.3.

Solution:

Since XTAL = 11.0592 MHz, the counter counts up every 1.085 us. This means that out of many 1.085 us intervals we must make a 5 ms pulse. To get that, we divide one by the other. We need 5 ms / 1.085 us = 4608 clocks. To Achieve that we need to load into TL and TH the value 65536 – 4608 = EE00H. Therefore, we have TH = EE and TL = 00.

```
       CLR  P2.3      ;Clear P2.3
       MOV  TMOD,#01 ;Timer 0, 16-bitmode
HERE:  MOV  TL0,#0    ;TL0=0, the low byte
       MOV  TH0,#0EEH ;TH0=EE, the high byte
       SETB P2.3      ;SET high P2.3
       SETB TR0       ;Start timer 0
AGAIN: JNB  TF0,AGAIN ;Monitor timer flag 0
       CLR  TR0       ;Stop the timer 0
       CLR  TF0       ;Clear timer 0 flag
```

65536 - 4608 = 60928

60928 (d) = EE00 (H)

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.

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:

This is similar to Example 9-10, except that we must toggle the bit to generate the square wave. Look at the following steps.

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

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

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

(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
       CLR  P1.5     ;Clear timer flag 1
       CLR  TF1      ;Clear timer 1 flag
       SJMP AGAIN    ;Reload timer
```

Mode 2 Programming

- To generate a time delay
  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 hi, lo
        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 $(256 - 05) \times 1.085$ us $= 251 \times 1.085$ us $= 272.33$ us 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 $T = 2 \times 272.33$ us $= 544.67$ us and the frequency = 1.83597 kHz

Example 9-15

Find the frequency of a square wave generated on pin P1.0.

Solution:

```
       MOV    TMOD,#2H   ;Timer 0, mod 2
                          ;(8-bit, auto reload)
       MOV    TH0,#0
AGAIN: MOV    R5,#250     ;multiple delay count
       ACALL  DELAY
       CPL    P1.0
       SJMP   AGAIN

DELAY: SETB   TR0         ;start the timer 0
BACK:  JNB    TF0,BACK   ;stay timer rolls over
       CLR    TR0         ;stop timer
       CLR    TF0         ;clear TF for next round
       DJNZ   R5,DELAY
       RET
```

Example 9-15

Find the frequency of a square wave generated on pin P1.0.

Solution:

```
        MOV    TMOD,#2H   ;Timer 0, mod 2
                           ;(8-bit, auto reload)
        MOV    TH0,#0
AGAIN:  MOV    R5,#250     ;multiple delay count
        ACALL  DELAY
        CPL    P1.0
        SJMP   AGAIN

DELAY:  SETB   TR0         ;start the timer 0
BACK:   JNB    TF0,BACK   ;stay timer rolls over
        CLR    TR0         ;stop timer
        CLR    TF0         ;clear TF for next round
        DJNZ   R5,DELAY
        RET
```

$T = 2\ (\ 250 \times 256 \times 1.085\ \text{us}\ ) = 138.88\text{ms}$, and frequency = 72 Hz

(a) `MOV TH1,#-200` (b) `MOV TH0,#-60`

(c) `MOV TH1,#-3` (d) `MOV TH1,#-12`

(e) `MOV TH0,#-48`

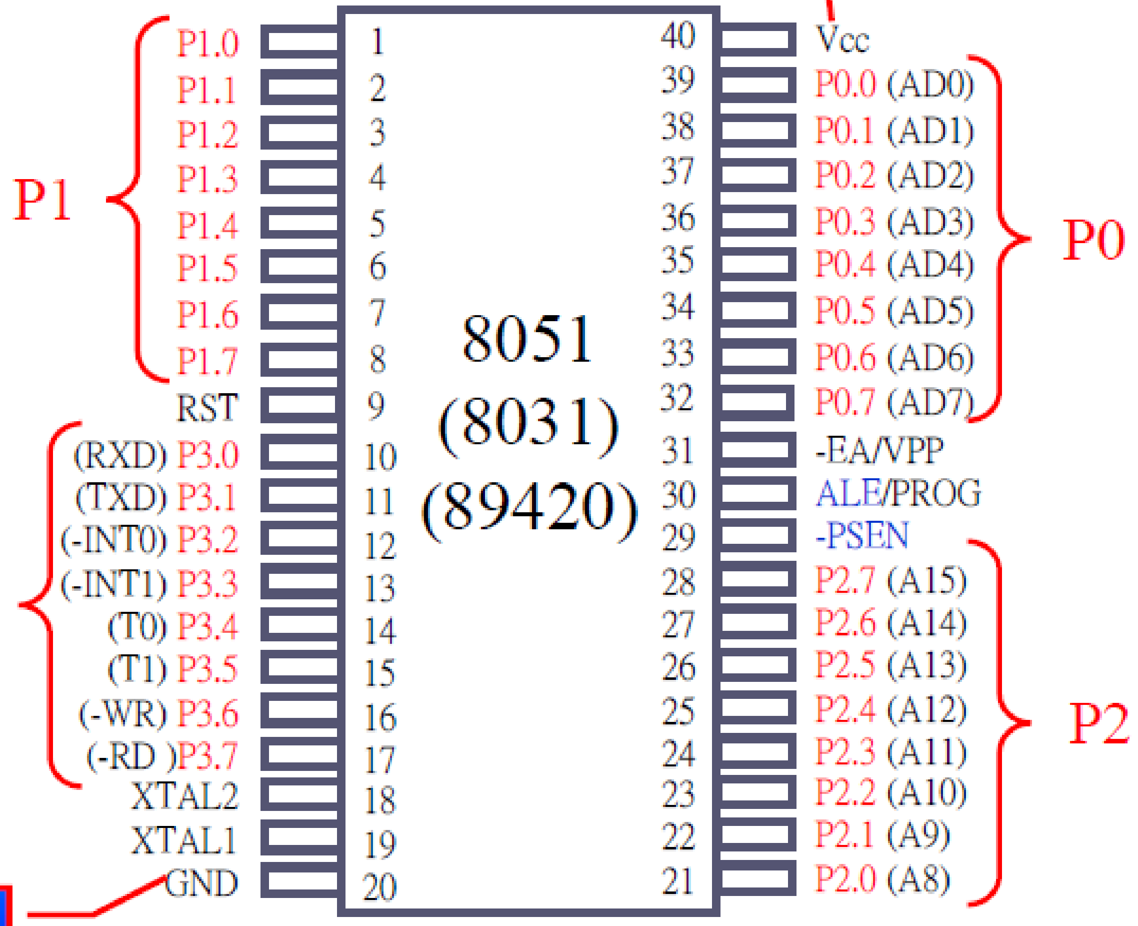
COUNTER PROGRAMMING

- Timers can also be used as counters counting events happening outside the 8051
  - When it is used as a counter, it is a pulse outside of the 8051 that increments the TH, TL registers
  - TMOD and TH, TL registers are the same as for the timer discussed previously
- Programming the timer in the last section also applies to programming it as a counter
  - Except the source of the frequency

8051 Pin Diagram

A total of 32 pins are set aside for the four ports P0, P1, P2, P3, where each port takes 8 pins

Provides +5V supply voltage to the chip



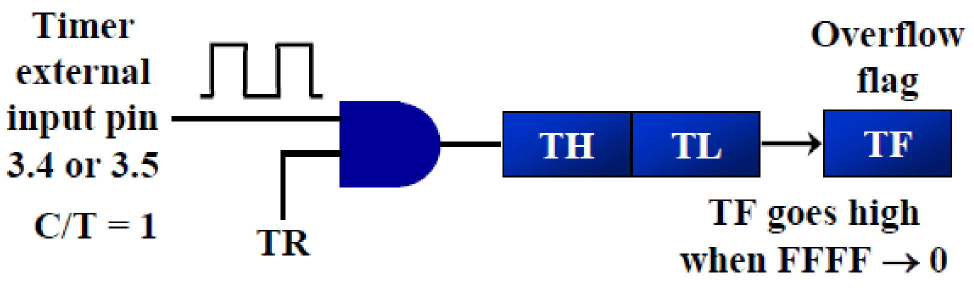
Grond

- The C/T bit in the TMOD registers decides the source of the clock for the timer
  - When C/T = 1, the timer is used as a counter and gets its pulses from outside the 8051
    - The counter counts up as pulses are fed from pins 14 and 15, these pins are called T0 (timer 0 input) and T1 (timer 1 input)

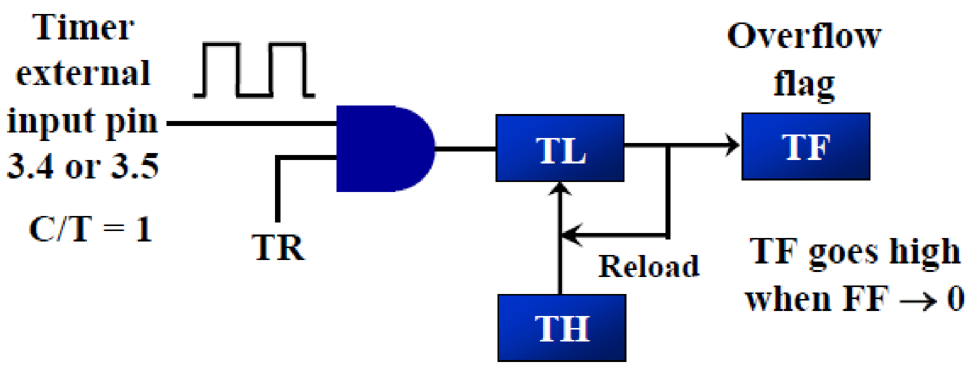
Port 3 pins used for Timers 0 and 1

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

Timer with external input (Mode 1)



Timer with external input (Mode 2)



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.

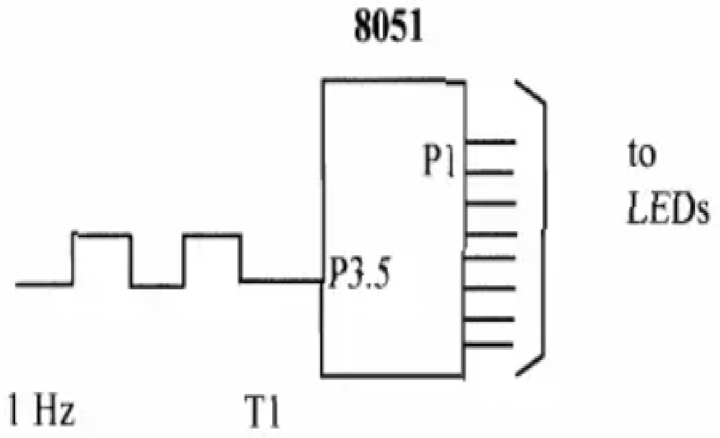
Example 9-18

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
```

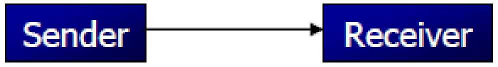
P1 is connected to 8 LEDs.
T1 (P3.5) is connected to a
1-Hz external clock.



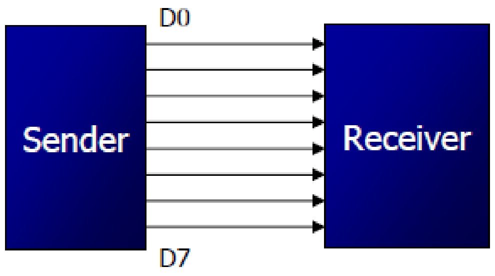
BASICS OF SERIAL COMMUNICATION

- 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

Serial Transfer

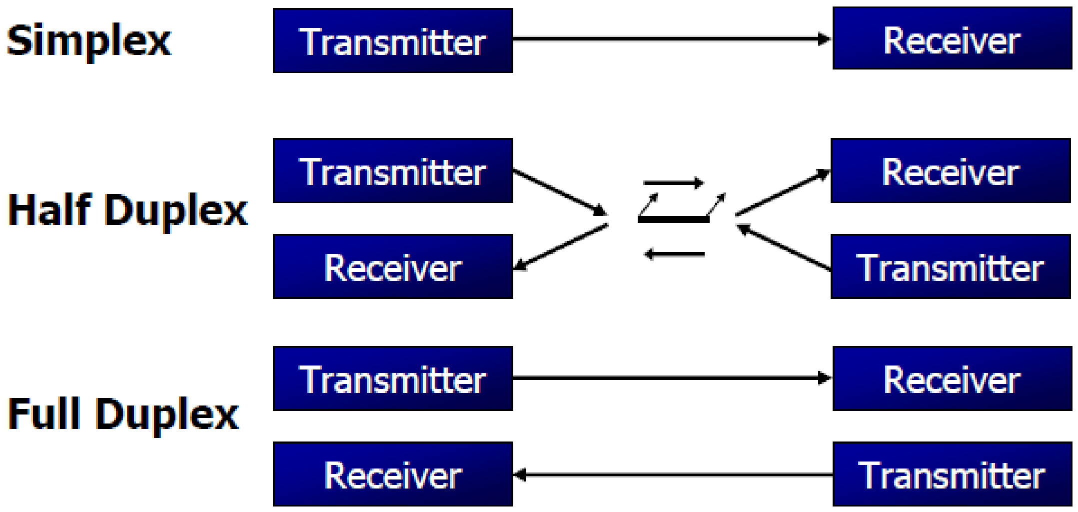


Parallel Transfer



- 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)

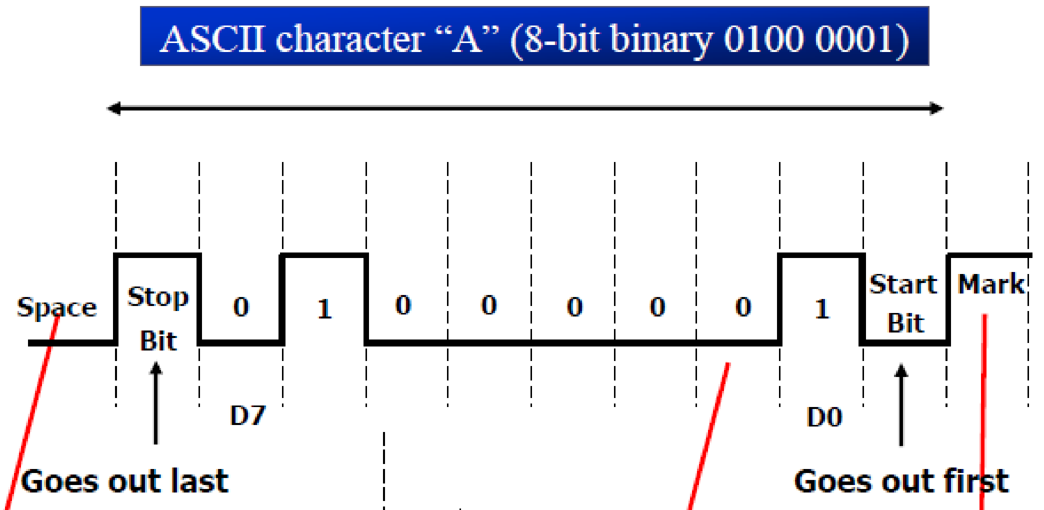
- If data can be transmitted and received, it is a *duplex* transmission
  - If data transmitted one way a time, it is referred to as *half duplex*
  - If data can go both ways at a time, it is *full duplex*
- This is contrast to *simplex* transmission



BASICS OF SERIAL COMMUNICATION

Start and Stop Bits (cont')

- The start bit is always a 0 (low) and the stop bit(s) is 1 (high)



The 0 (low) is referred to as *space*

The transmission begins with a start bit followed by D0, the LSB, then the rest of the bits until MSB (D7), and finally, the one stop bit indicating the end of the character

When there is no transfer, the signal is 1 (high), which is referred to as *mark*

- 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

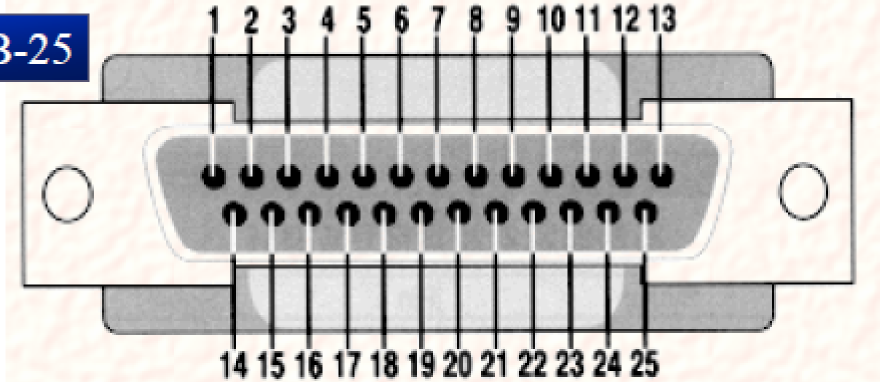
- The rate of data transfer in serial data communication is stated in *bps* (bits per second)
- Another widely used terminology for bps is *baud rate*
  - It is modem terminology and is defined as the number of signal changes per second
  - In modems, there are occasions when a single change of signal 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

- 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

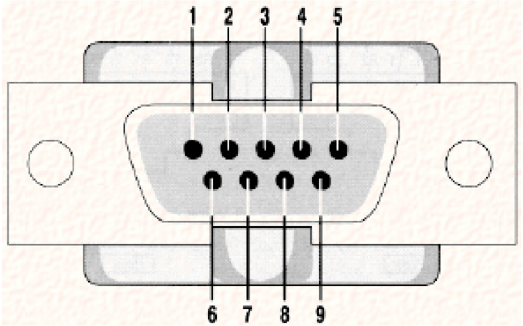
Pin	Description	Pin	Description
1	Protective ground	14	Secondary transmitted data
2	Transmitted data (TxD)	15	Transmitted signal element timing
3	Received data (RxD)	16	Secondary receive data
4	Request to send (-RTS)	17	Receive signal element timing
5	Clear to send (-CTS)	18	Unassigned
6	Data set ready (-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)

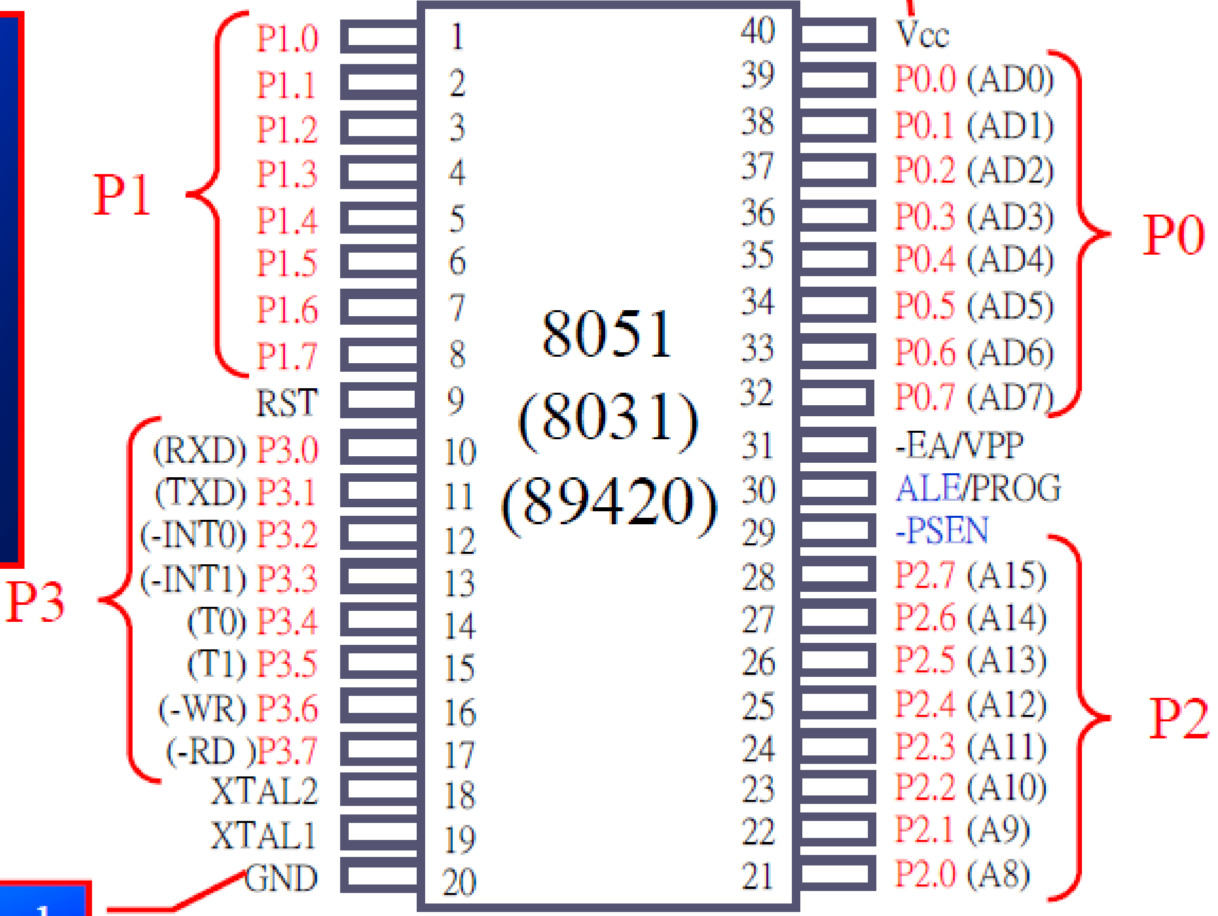
- A line driver such as the MAX232 chip is required to convert RS232 voltage levels to TTL levels, and vice versa
- 8051 has two pins that are used specifically for transferring and receiving data serially
  - These two pins are called TxD and RxD and are part of the port 3 group (P3.0 and P3.1)
  - These pins are TTL compatible; therefore, they require a line driver to make them RS232 compatible

8051 Pin Diagram

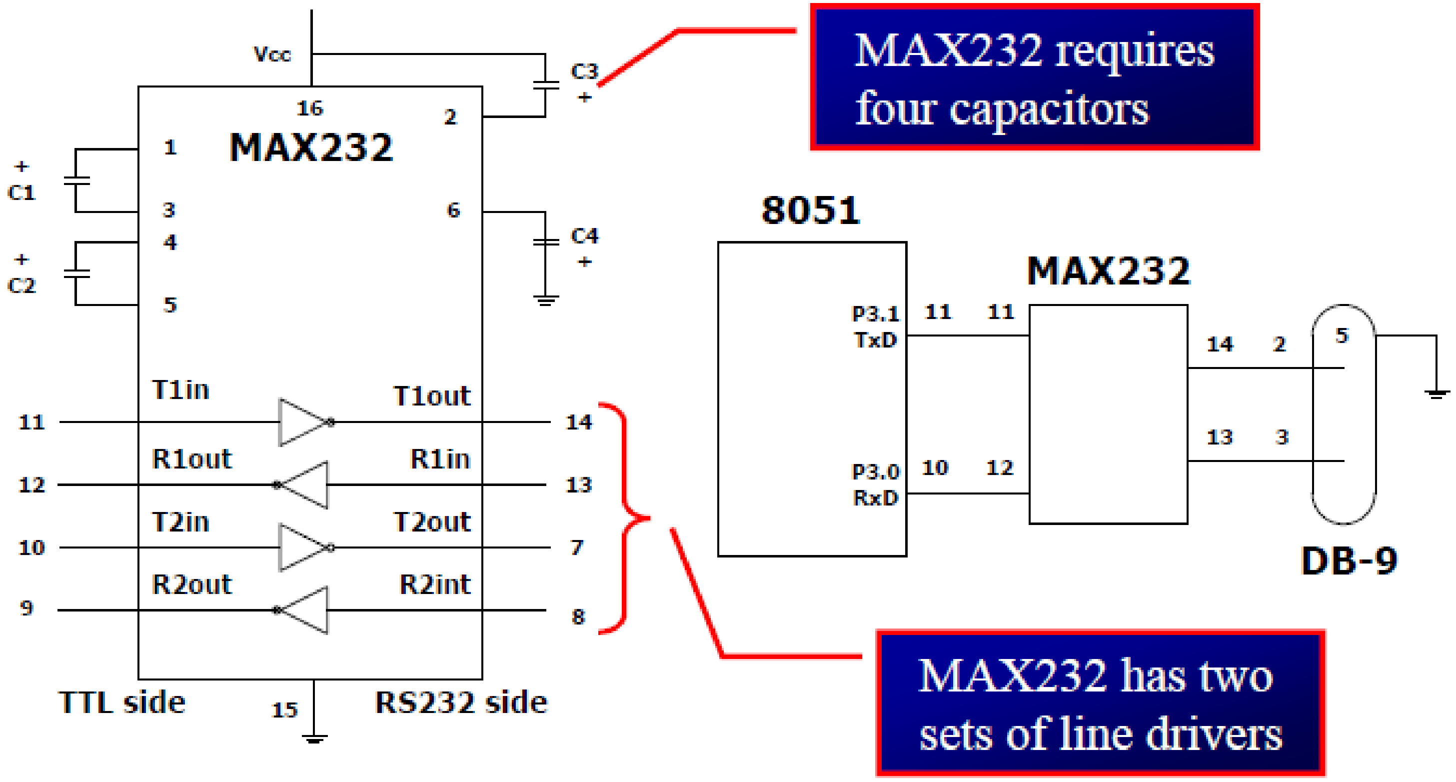
A total of 32 pins are set aside for the four ports P0, P1, P2, P3, where each port takes 8 pins

Provides +5V supply voltage to the chip

Grond



- 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



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:

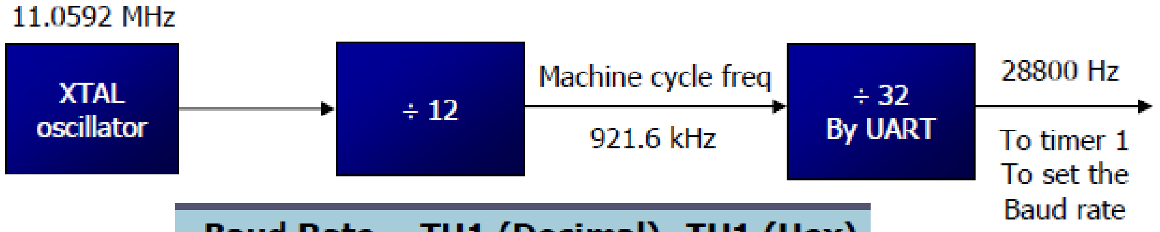
The machine cycle frequency of 8051 = 11.0592 / 12 = 921.6 kHz, and 921.6 kHz / 32 = 28,800 Hz is frequency by UART to timer 1 to set baud rate.

(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.



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

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

- SBUF is an 8-bit register used solely for serial communication
  - For a byte data to be transferred via the TxD line, it must be placed in the SBUF register
    - The moment a byte is written into SBUF, it is framed with the start and stop bits and transferred serially via the TxD line
  - SBUF holds the byte of data when it is received by 8051 RxD line
    - When the bits are received serially via RxD, the 8051 deframes it by eliminating the stop and start bits, making a byte out of the data received, and then placing it in SBUF

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

SFR RAM Address (Byte and Bit)

Bit addresses 80 – F7H belong to SFR of P0, TCON, P1, SCON, P2, etc

Byte address	Bit address	
FF		
F0	F7 F6 F5 F4 F3 F2 F1 F0	B
E0	E7 E6 E5 E4 E3 E2 E1 E0	ACC
D0	D7 D6 D5 D4 D3 D2 D1 D0	PSW
B8	-- -- -- BC BB BA B9 B8	IP
B0	B7 B6 B5 B4 B3 B2 B1 B0	P3
A8	AF AE AD AC AB AA A9 A8	IE
A0	A7 A6 A5 A4 A3 A2 A1 A0	P2
99	not bit addressable	SBUF

Byte address	Bit address	
98	9F 9E 9D 9C 9B 9A 99 98	SCON
90	97 96 95 94 93 92 91 90	P1
8D	not bit addressable	TH1
8C	not bit addressable	TH0
8B	not bit addressable	TL1
8A	not bit addressable	TL0
89	not bit addressable	TMOD
88	8F 8E 8D 8C 8B 8A 89 88	TCON
87	not bit addressable	PCON
83	not bit addressable	DPH
82	not bit addressable	DPL
81	not bit addressable	SP
80	87 86 85 84 83 82 81 80	P0

Special Function Register

- SCON is an 8-bit register used to program the start bit, stop bit, and data bits of data framing, among other things

SM0	SM1	SM2	REN	TB8	RB8	TI	RI

SM0	SCON.7	Serial port mode specifier
SM1	SCON.6	Serial port mode specifier
SM2	SCON.5	Used for multiprocessor communication
REN	SCON.4	Set/cleared by software to enable/disable reception
TB8	SCON.3	Not widely used
RB8	SCON.2	Not widely used
TI	SCON.1	Transmit interrupt flag. Set by HW at the begin of the stop bit mode 1. And cleared by SW
RI	SCON.0	Receive interrupt flag. Set by HW at the begin of the stop bit mode 1. And cleared by SW

Note: Make SM2, TB8, and RB8 =0

- 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)
  - It is a bit-adressable register
    - 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

- 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

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
```

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
```

- 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
```

8051 PROGRAMMING IN C

DATA TYPES

- A good understanding of C data types for 8051 can help programmers to create smaller hex files
  - Unsigned char
  - Signed char
  - Unsigned int
  - Signed int
  - Sbit (single bit)
  - Bit and sfr

- The character data type is the most natural choice
  - 8051 is an 8-bit microcontroller
- Unsigned char is an 8-bit data type in the range of 0 – 255 (00 – FFH)
  - One of the most widely used data types for the 8051
    - Counter value
    - ASCII characters
- C compilers use the signed char as the default if we do not put the keyword *unsigned*

Write an 8051 C program to toggle all the bits of P1 continuously.

Solution:

```
//Toggle P1 forever
#include <reg51.h>
void main(void)
  {
    for (;;)
      {
        p1=0x55;
        p1=0xAA;
      }
  }
```

The following code will continuously send out to port 0 the alternating value 55H and AAH

```
BACK:  MOV    A,#55H
       MOV    P0,A
       ACALL  DELAY
       MOV    A,#0AAH
       MOV    P0,A
       ACALL  DELAY
       SJMP   BACK
```

```
/*  BYTE Register  */
sfr at 0x80 P0   ;
sfr at 0x90 P1   ;
sfr at 0xA0 P2   ;
sfr at 0xB0 P3   ;
sfr at 0xD0 PSW  ;
sfr at 0xE0 ACC  ;
sfr at 0xF0 B    ;
sfr at 0x81 SP   ;
sfr at 0x82 DPL  ;
sfr at 0x83 DPH  ;
sfr at 0x87 PCON ;
sfr at 0x88 TCON ;
```

Write an 8051 C program to send values 00 – FF to port P1.

Solution:

```
#include <reg51.h>
void main(void)
  {
    unsigned char z;
    for (z=0;z<=255;z++)
       P1=z;
  }
```

1. Pay careful attention to the size of the data
2. Try to use unsigned *char* instead of *int* if possible

Write an 8051 C program to send hex values for ASCII characters of 0, 1, 2, 3, 4, 5, A, B, C, and D to port P1.

Solution:

```
#include <reg51.h>
void main(void)
  {
    unsigned char mynum[]="012345ABCD";
    unsigned char z;
    for (z=0;z<=10;z++)
       P1=mynum[z];
  }
```

Write an 8051 C program to send values of –4 to +4 to port P1.

Solution:

```
//Singed numbers
#include <reg51.h>
void main(void)
  {
    char mynum[]={+1,-1,+2,-2,+3,-3,+4,-4};
    unsigned char z;
    for (z=0;z<=8;z++)
       P1=mynum[z];
  }
```

- The unsigned int is a 16-bit data type
  - Takes a value in the range of 0 to 65535 (0000 – FFFFH)
  - Define 16-bit variables such as memory addresses
  - Set counter values of more than 256
  - Since registers and memory accesses are in 8-bit chunks, the misuse of int variables will result in a larger hex file
- Signed int is a 16-bit data type
  - Use the MSB D15 to represent – or +
  - We have 15 bits for the magnitude of the number from –32768 to +32767

Write an 8051 C program to toggle bit D0 of the port P1 (P1.0) 50,000 times.

Solution:

sbit keyword allows access to the single bits of the SFR registers

```
#include <reg51.h>
sbit MYBIT=P1^0;

void main(void)
  {
    unsigned int z;
    for (z=0;z<=50000;z++)
       {
         MYBIT=0;
         MYBIT=1;
       }
  }
```

- The bit data type allows access to single bits of bit-addressable memory spaces 20 – 2FH
- To access the byte-size SFR registers, we use the sfr data type

Data Type	Size in Bits	Data Range/Usage
unsigned char	8-bit	0 to 255
(signed) char	8-bit	-128 to +127
unsigned int	16-bit	0 to 65535
(signed) int	16-bit	-32768 to +32767
sbit	1-bit	SFR bit-addressable only
bit	1-bit	RAM bit-addressable only
sfr	8-bit	RAM addresses 80 – FFH only

TIME DELAY

Write an 8051 C program to toggle bits of P1 continuously forever with some delay.

Solution:

```
//Toggle P1 forever with some delay in between
//“on” and “off”
#include <reg51.h>
void main(void)
  {
    unsigned int x;
    for (;;)                  //repeat forever
      {
        p1=0x55;
        for (x=0;x<40000;x++); //delay size
                               //unknown
        p1=0xAA;
        for (x=0;x<40000;x++);
      }
  }
```

We must use the oscilloscope to measure the exact duration

Write an 8051 C program to toggle bits of P1 ports continuously with a 250 ms.

Solution:

```
#include <reg51.h>
void MSDelay(unsigned int);
void main(void)
  {
    while (1)                    //repeat forever
       {
         p1=0x55;
         MSDelay(250);
         p1=0xAA;
         MSDelay(250);
       }
  }

void MSDelay(unsigned int itime)
  {
    unsigned int i,j;
    for (i=0;i<itime;i++)
       for (j=0;j<1275;j++);
  }
```

Write an 8051 C program to get a byte of data form P1, wait 1/2 second, and then send it to P2.

Solution:

```
#include <reg51.h>
void MSDelay(unsigned int);

void main(void)
  {
    unsigned char mybyte;
    P1=0xFF;                    //make P1 input port
    while (1)
       {
         mybyte=P1;             //get a byte from P1
         MSDelay(500);
         P2=mybyte;             //send it to P2
       }
  }
```

A door sensor is connected to the P1.1 pin, and a buzzer is connected to P1.7. Write an 8051 C program to monitor the door sensor, and when it opens, sound the buzzer. You can sound the buzzer by sending a square wave of a few hundred Hz.

A door sensor is connected to the P1.1 pin, and a buzzer is connected to P1.7. Write an 8051 C program to monitor the door sensor, and when it opens, sound the buzzer. You can sound the buzzer by sending a square wave of a few hundred Hz.

Solution:

```
#include <reg51.h>
void MSDelay(unsigned int);
sbit Dsensor=P1^1;
sbit Buzzer=P1^7;

void main(void)
  {
    Dsensor=1;                //make P1.1 an input
    while (1)
      {
        while (Dsensor==1)//while it opens
          {
            Buzzer=0;
            MSDelay(200);
            Buzzer=1;
            MSDelay(200);
          }
      }
  }
```

Example 9-20

Write an 8051 C program to toggle all the bits of port P1 continuously with some delay in between. Use Timer 0, 16-bit mode to generate the delay.

Solution:

```
#include <reg51.h>
void T0Delay(void);
void main(void){
  while (1) {
    P1=0x55;
    T0Delay();
    P1=0xAA;
    T0Delay();
  }
}
void T0Delay(){
  TMOD=0x01;
  TL0=0x00;
  TH0=0x35;
  TR0=1;
  while (TF0==0);
  TR0=0;
  TF0=0;
}
```

FFFFH – 3500H = CAFFH

= 51967 + 1 = 51968

51968 × 1.085 μs = 56.384 ms is the approximate delay

Example 9-22

Write an 8051 C program to toggle all bits of P2 continuously every 500 ms. Use Timer 1, mode 1 to create the delay.

Example 9-22

Write an 8051 C program to toggle all bits of P2 continuously every 500 ms. Use Timer 1, mode 1 to create the delay.

Solution:

```
//tested for DS89C420, XTAL = 11.0592 MHz
#include <reg51.h>
void T1M1Delay(void);
void main(void){
  unsigned char x;
  P2=0x55;
  while (1) {
    P2=~P2;
    for (x=0;x<20;x++)
        T1M1Delay();
  }
}
void T1M1Delay(void){
  TMOD=0x10;
  TL1=0xFE;
  TH1=0xA5;
  TR1=1;
  while (TF1==0);
  TR1=0;
  TF1=0;
}
```

A5FEH = 42494 in decimal

65536 – 42494 = 23042

23042 × 1.085 μs = 25 ms and

20 × 25 ms = 500 ms

SERIAL PORT PROGRAMMING IN C

Example 10-15

Write a C program for 8051 to transfer the letter "A" serially at 4800 baud continuously. Use 8-bit data and 1 stop bit.

Solution:

```
#include <reg51.h>
void main(void){
  TMOD=0x20;          //use Timer 1, mode 2
  TH1=0xFA;           //4800 baud rate
  SCON=0x50;
  TR1=1;
  while (1) {
    SBUF='A';         //place value in buffer
    while (TI==0);
    TI=0;
  }
}
```

Example 10-16

Write an 8051 C program to transfer the message "YES" serially at 9600 baud, 8-bit data, 1 stop bit. Do this continuously.

Example 10-16

Write an 8051 C program to transfer the message “YES” serially at 9600 baud, 8-bit data, 1 stop bit. Do this continuously.

Solution:

```
#include <reg51.h>
void SerTx(unsigned char);
void main(void){
  TMOD=0x20;          //use Timer 1, mode 2
  TH1=0xFD;           //9600 baud rate
  SCON=0x50;
  TR1=1;              //start timer
  while (1) {
    SerTx('Y');
    SerTx('E');
    SerTx('S');
  }
}
void SerTx(unsigned char x){
  SBUF=x;             //place value in buffer
  while (TI==0);      //wait until transmitted
  TI=0;
}
```

Example 10-17

Program the 8051 in C to receive bytes of data serially and put them in P1. Set the baud rate at 4800, 8-bit data, and 1 stop bit.

Example 10-17

Program the 8051 in C 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:

```
#include <reg51.h>
void main(void){
  unsigned char mybyte;
  TMOD=0x20;          //use Timer 1, mode 2
  TH1=0xFA;           //4800 baud rate
  SCON=0x50;
  TR1=1;              //start timer
  while (1) {         //repeat forever
    while (RI==0);    //wait to receive
    mybyte=SBUF;      //save value
    P1=mybyte;        //write value to port
    RI=0;
  }
}
```

INTERRUPTS

INTERRUPTS

Interrupts vs. Polling (cont')

- (cont')
  - 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`

- For every interrupt, there must be an interrupt service routine (ISR), or interrupt handler
  - When an interrupt is invoked, the micro-controller 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

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	

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)

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

Priority bit=0 assigns low priority

- 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

IE (Interrupt Enable) Register

D7							D0
EA	--	ET2	ES	ET1	EX1	ET0	EX0

EA (enable all) must be set to 1 in order for rest of the register to take effect

EA	IE.7	Disables all interrupts
--	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

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
```

Table 11-4: 8051/52 Interrupt Numbers in C

Interrupt	Name	Numbers used by 8051 C
External Interrupt 0	(INT0)	0
Timer Interrupt 0	(TF0)	1
External Interrupt 1	(INT1)	2
Timer Interrupt 1	(TF1)	3
Serial Communication	(RI + TI)	4
Timer 2 (8052 only)	(TF2)	5

Example 11-14

Write a C program that continuously gets a single bit of data from P1.7 and sends it to P1.0, while simultaneously creating a square wave of 200 μs period on pin P2.5. Use Timer 0 to create the square wave. Assume that XTAL = 11.0592 MHz.

Solution:

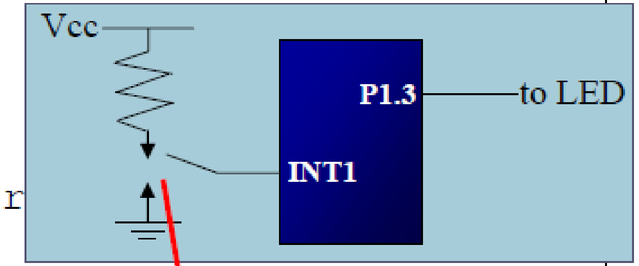
We will use timer 0 mode 2 (auto-reload). One half of the period is 100 μs. 100/1.085 μs = 92, and TH0 = 256 - 92 = 164 or A4H

```
#include <reg51.h>
sbit SW   =P1^7;
sbit IND  =P1^0;
sbit WAVE =P2^5;
void timer0(void) interrupt 1 {
  WAVE=~WAVE;  //toggle pin
}
void main() {
  SW=1;          //make switch input
  TMOD=0x02;
  TH0=0xA4;      //TH0=-92
  IE=0x82;       //enable interrupt for timer 0
  while (1) {
    IND=SW;      //send switch to LED
  }
}
```

- 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

Example 11-5

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



Solution:

```
      ORG  0000H
      LJMP MAIN ;by-pass inter
                ;vector table
;--ISR for INT1 to turn on LED
      ORG  0013H      ;INT1 ISR
      SETB P1.3       ;turn on LED
      MOV  R3,#255
BACK: DJNZ R3,BACK    ;keep LED on for a
      CLR  P1.3       ;turn off the LED
      RETI            ;return from ISR

;--MAIN program for initialization
      ORG  30H
MAIN: MOV  IE,#10000100B ;enable external INT 1
HERE: SJMP HERE      ;stay here until get interrupted
      END
```

Pressing the switch will cause the LED to be turned on. If it is kept activated, the LED stays on

- 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

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

EXTERNAL HARDWARE INTERRUPTS

Edge-Triggered Interrupt (cont')

Assume that 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, which is connected to an LED (or buzzer). In other words, the LED is turned on and off at the same rate as the pulses are applied to the INT1 pin.

When the falling edge of the signal is applied to pin INT1, the LED will be turned on momentarily.

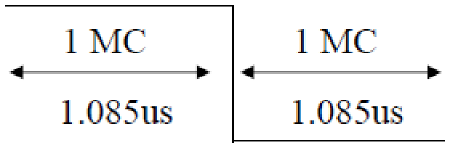
Solution:

```
        ORG   0000H
        LJMP MAIN
;--ISR for hardware interrupt INT1 to turn on LED
        ORG   0013H   ;INT1 ISR
        SETB P1.3     ;turn on LED
        MOV   R3,#255
BACK:   DJNZ R3,BACK ;keep the buzzer on for a while
        CLR   P1.3    ;turn off the buzzer
        RETI          ;return from ISR
;------MAIN program for initialization
        ORG   30H
MAIN:   SETB TCON.2 ;make INT1 edge-triggered int.
        MOV   IE,#10000100B ;enable External INT 1
HERE:   SJMP HERE     ;stay here until get interrupted
        END
```

The on-state duration depends on the time delay inside the ISR for INT1

- 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



Self study – Interrupt for serial communication TI, RI Flags