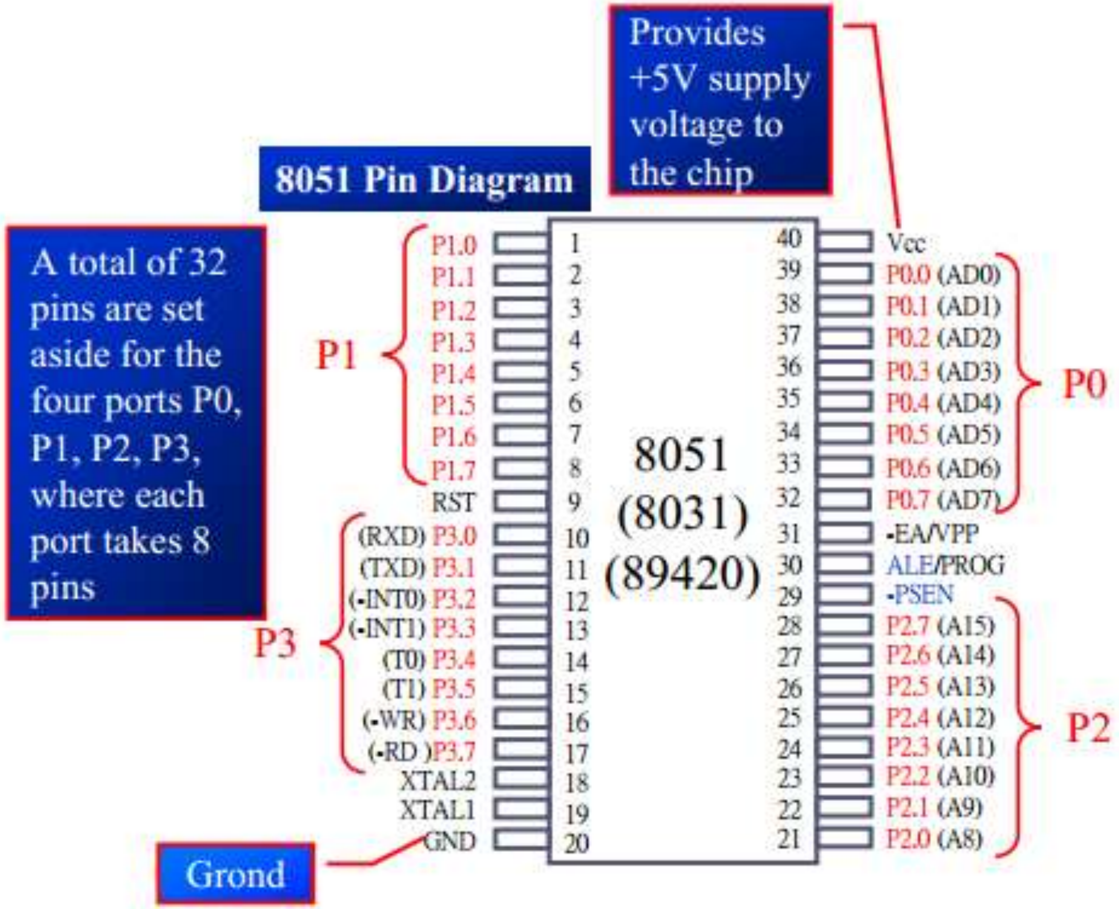
8051
I/O PORTS

8051 I/O PORTS



8051 I/O PORTS

- All 8051 microcontrollers have 4 I/O ports each comprising 8 bits which can be configured as inputs or outputs.
- Pin configuration, i.e. whether it is to be configured as an input (1) or an output (0), depends on its logic state.
- All the ports upon RESET are configured as input, ready to be used as input ports
- In total of 32 input/output pins enabling the microcontroller to be connected to peripheral devices are available for use.

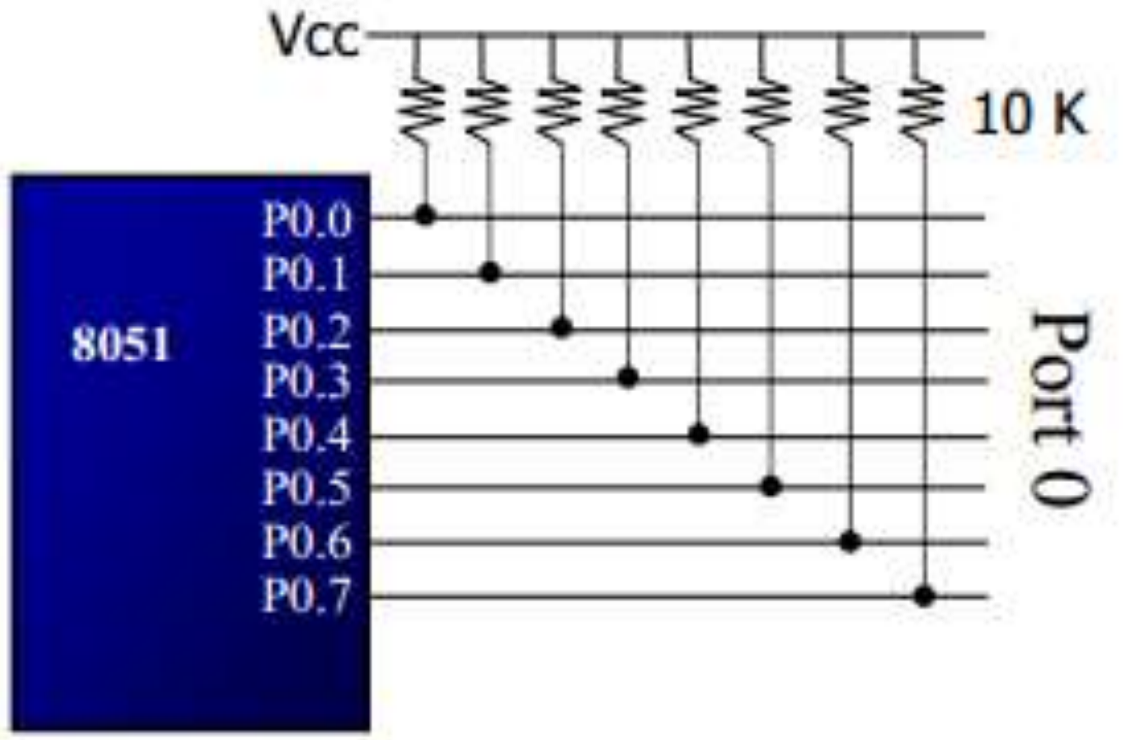
8051 I/O PORTS

PORT - 0

- Port 0 occupies a total of 8 pins (pins 32-39). It can be used for input or output.
- To use the pins of port 0 as both input and output ports, each pin must be connected externally to a 10K ohm pull-up resistor. This is due to the fact that P0 is an open drain, unlike P1, P2, and P3.
- Dual role of port-0: Port 0 is also designated as AD0-AD7, allowing it to be used for both address and data. When ALE = 0, it provides data D0-D7, but when ALE =1 it has address and data with the help of a 74LS373 latch.

8051 I/O PORTS

PORT-0



8051 I/O PORTS

PORT - 1

- Port 1 occupies a total of 8 pins (pins 1 through 8). Port 1 can be used as input or output
- In contrast to port 0, this port does not need any pull-up resistors since it already has pull-up resistors internally
- Upon reset, port 1 is configured as an input port
- To make port 1 an input port, it must be programmed as such by writing 1 to all its bits

8051 I/O PORTS

PORT - 2

- Port 2 occupies a total of 8 pins (pins 21- 28). It can be used as input or output. Upon reset, Port 2 is configured as an input port.
- Just like P1, P2 does not need any pull-up resistors since it already has pull-up resistors internally.
- In many 8051-based system, P2 is used as simple I/O but in 8031-based systems, port 2 must be used along with P0 to provide the 16-bit address for the external memory
- Port 2 is also designated as A8 –A15, indicating its dual function and Port 0 provides the lower 8 bits via A0 –A7.

8051 I/O PORTS

PORT – 3

- Port 3 occupies a total of 8 pins, pins 10 through 17.
- It can be used as input or output. Upon reset, Port 3 is configured as an input port.
- P3 does not need any pull-up resistors, the same as P1 and P2 did not.
- Port 3 has the additional function of providing some extremely important signals such as interrupts.

8051 I/O PORTS

PORT - 3

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

Serial communications

External interrupts

Timers

Read/Write signals of external memories

In systems based on 8751, 89C51 or DS89C4x0, pins 3.6 and 3.7 are used for I/O while the rest of the pins in port 3 are normally used in the alternate function role

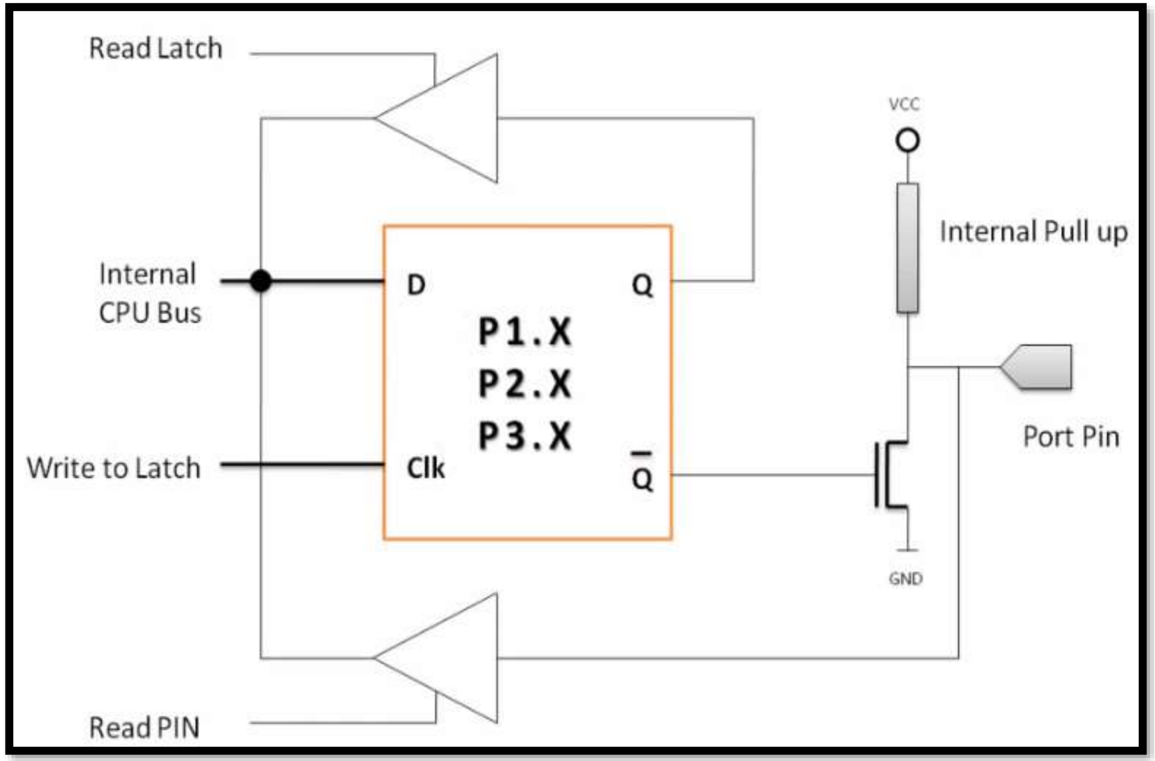
8051 I/O PORTS

Pin's Current limitations

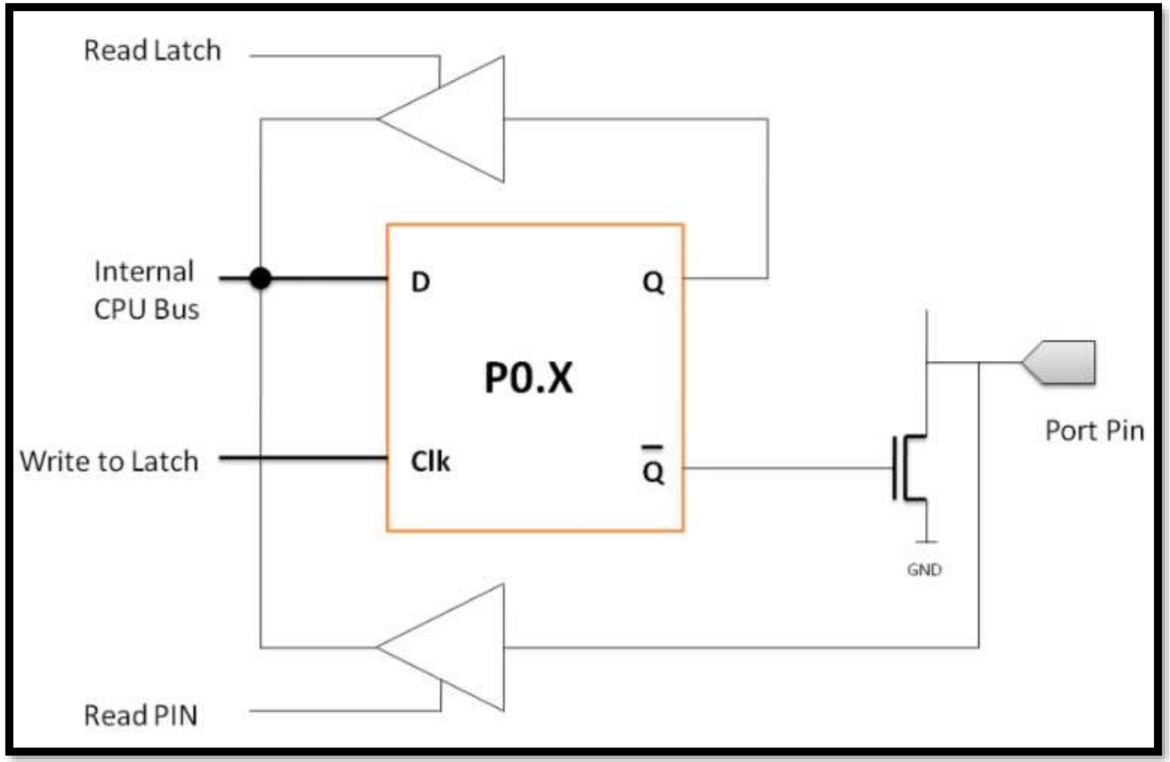
- When configured as outputs (logic zero (0)), single port pins can receive a current of 10mA.
- If all 8 bits of a port are active, a total current must be limited to 15mA (port P0: 26mA).
- If all ports (32 bits) are active, total maximum current must be limited to 71mA.
- When these pins are configured as inputs (logic 1), built-in pull-up resistors provide very weak current, but strong enough to activate up to 4 TTL inputs of LS series.

8051 I/O PORTS

LOGICAL STRUCTURE OF 8051 I/O PORTS



PORT – 1, 2 & 3

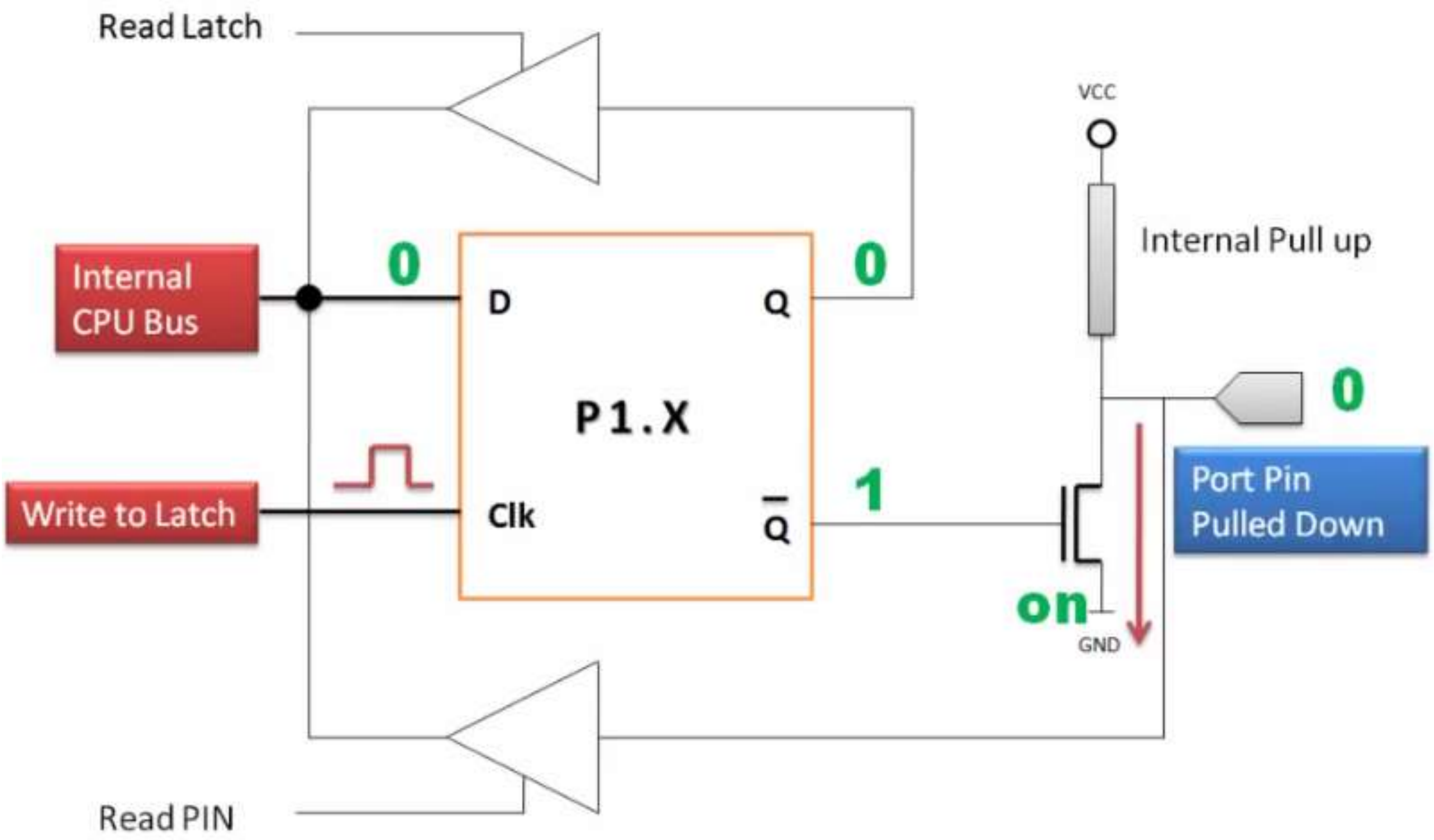


PORT - 0

Reference: https://www.youtube.com/watch?v=qTZaE2IcYTo

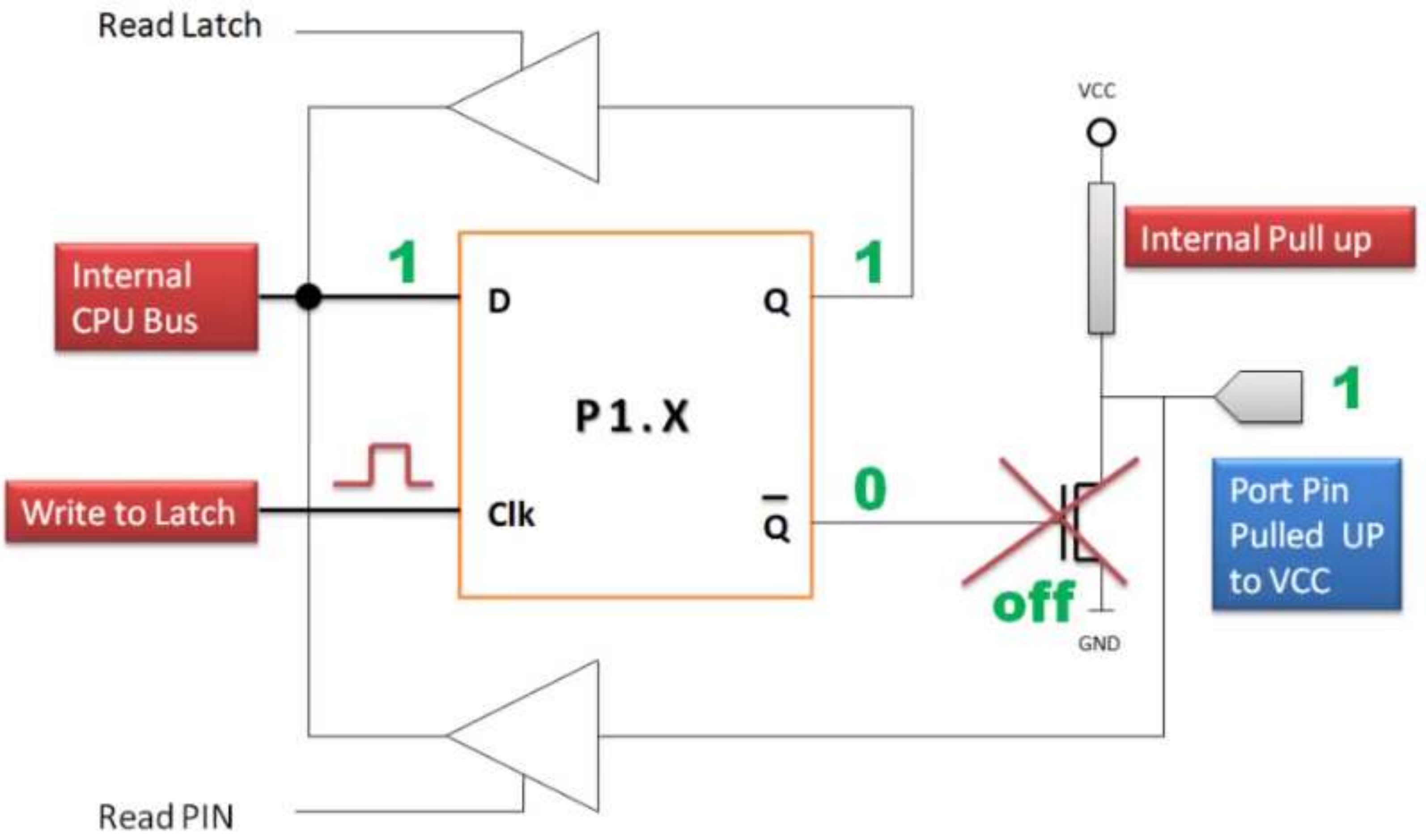
8051 I/O PORTS

Writing "0" to the port (output)

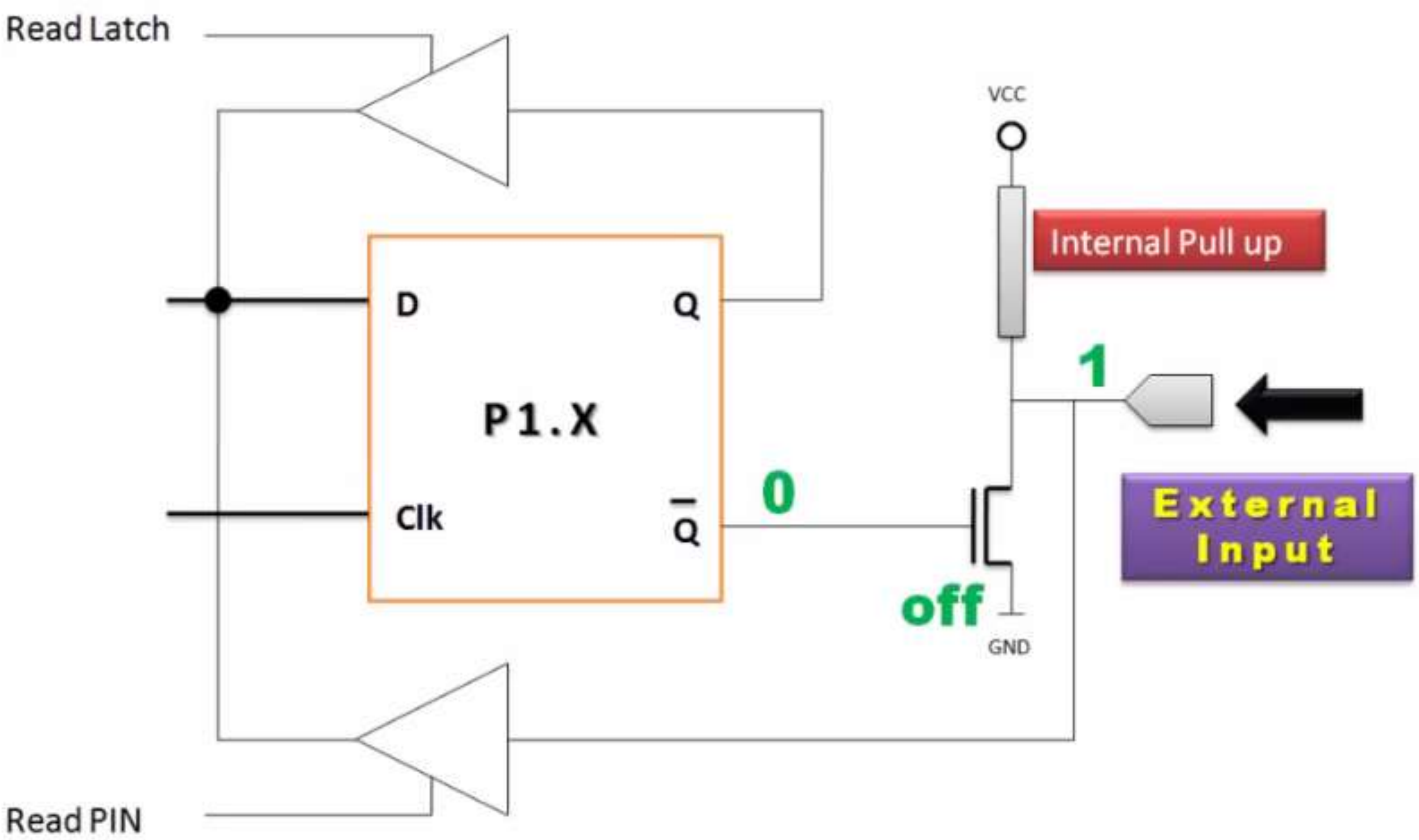


8051 I/O PORTS

Writing “1” to the port (output)

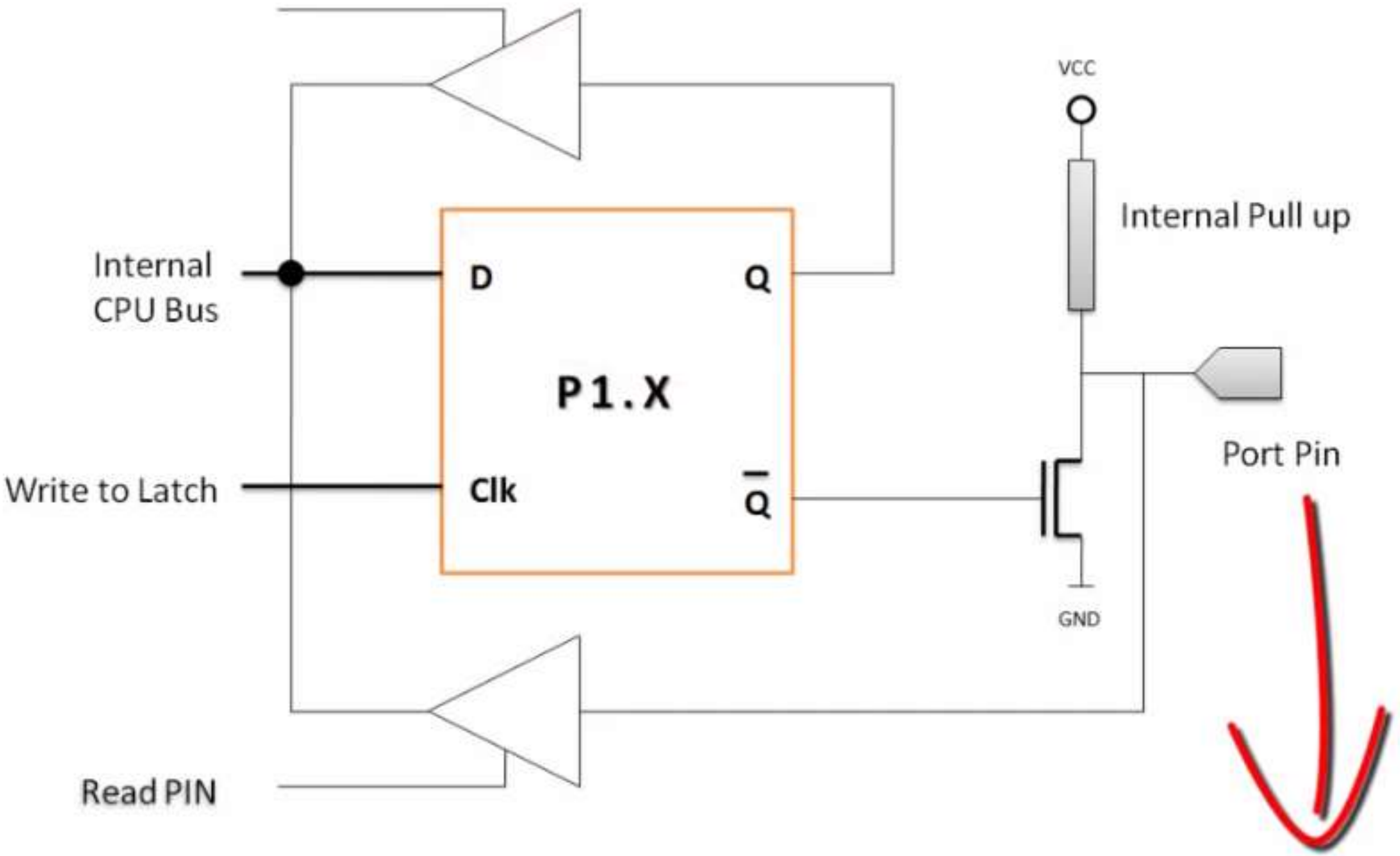


Reading “1” to the port (input)



8051 I/O PORTS

Reading "0" to the port (input)



8051 I/O PORTS

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

- There is a difference between reading a latch and reading the output port pin.
- **Reading a latch:** Usually the instructions that read the latch, read a value, possibly change it, and then rewrite it to the latch. These are called "read-modify-write" instructions. Examples are-

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

- In this the latch value of P2 is read, is modified and is then written back to P2 latch.

8051 I/O PORTS

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

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

"read-modify-write" instructions

8051 I/O PORTS

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

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

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

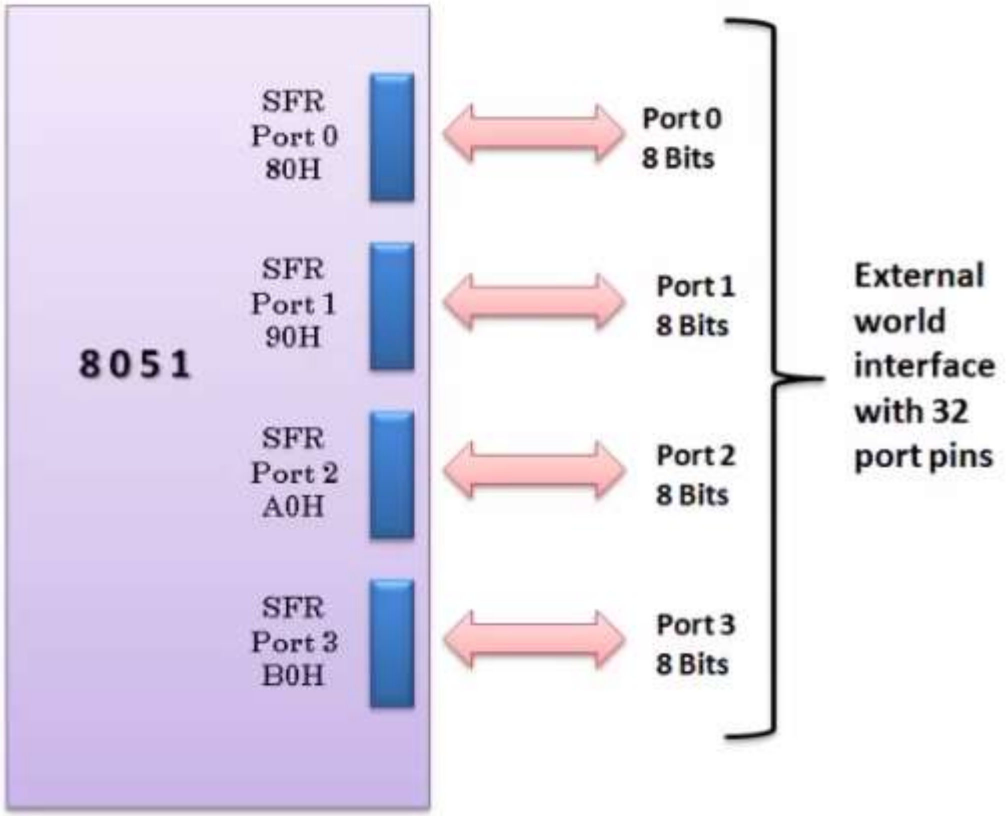
  MOV A, P1; Move port-1 pin values to A

Mnemonic	Example
MOV A , PX	MOV A,P1
JNB PX.Y , ...	JNB P1.2, TARGET
JB PX.Y , ...	JB P1.2, TARRAN
MOV C , PX.Y	MOV C, P1.4
CJNE A,PX	CJNE A, P3

Instructions Reading the Status of Input Port pins

8051 I/O PORTS

8051 I/O PORTS & SFR BIT AND BYTE ADDRESS LOCATIONS



PORT 0 BITS	P0.7	P0.6	P0.5	P0.4	P0.3	P0.2	P0.1	P0.0
BIT ADDRESS	87H	86H	85H	84H	83H	82H	81H	80H

PORT 1 BITS	P1.7	P1.6	P1.5	P1.4	P1.3	P1.2	P1.1	P1.0
BIT ADDRESS	97H	96H	95H	94H	93H	92H	91H	90H

PORT 2 BITS	P2.7	P2.6	P2.5	P2.4	P2.3	P2.2	P2.1	P2.0
BIT ADDRESS	A7H	A6H	A5H	A4H	A3H	A2H	A1H	A0H

PORT 3 BITS	P3.7	P3.6	P3.5	P3.4	P3.3	P3.2	P3.1	P3.0
BIT ADDRESS	B7H	B6H	B5H	B4H	B3H	B2H	B1H	B0H

8051

I/O PORTS PROGRAMS

8051 I/O PORTS PROGRAM - 1

Port 0 is configured first as an input port by writing 1s to it, and then data is received from that port and sent to P1

```
        MOV     A,#0FFH         ;A=FF hex
        MOV     P0,A            ;make P0 an i/p port
                                ;by writing it all 1s
BACK:   MOV     A,P0            ;get data from P0
        MOV     P1,A            ;send it to port 1
        SJMP    BACK            ;keep doing it
```

8051 I/O PORTS PROGRAM - 2

The following code will continuously send out to port 1 the alternating value 55H and AAH.

The entire 8 bits of Port 1 are accessed

```
BACK:   MOV     A,#55H
        MOV     P1,A
        ACALL   DELAY
        MOV     A,#0AAH
        MOV     P1,A
        ACALL   DELAY
        SJMP    BACK
```

8051 I/O PORTS PROGRAM - 2

Rewrite the code in a more efficient manner by accessing the port directly without going through the accumulator

```
BACK:   MOV     P1,#55H
        ACALL   DELAY
        MOV     P1,#0AAH
        ACALL   DELAY
        SJMP    BACK
```

Another way of doing the same thing

```
        MOV     A,#55H
BACK:   MOV     P1,A
        ACALL   DELAY
        CPL     A
        SJMP    BACK
```

8051 I/O PORTS PROGRAM - 3

Write the following programs.

Create a square wave of 50% duty cycle on bit 0 of port 1.

Solution:

The 50% duty cycle means that the "on" and "off" state (or the high and low portion of the pulse) have the same length. Therefore, we toggle P1.0 with a time delay in between each state.

```
HERE:   SETB    P1.0    ;set to high bit 0 of port 1
        LCALL   DELAY   ;call the delay subroutine
        CLR     P1.0    ;P1.0=0
        LCALL   DELAY
        SJMP    HERE    ;keep doing it
```

8051 I/O PORTS PROGRAM - 4

Write a program to perform the following:
(a) Keep monitoring the P1.2 bit until it becomes high
(b) When P1.2 becomes high, write value 45H to port 0
(c) Send a high-to-low (H-to-L) pulse to P2.3

Solution:

```
        SETB P1.2          ;make P1.2 an input
        MOV  A,#45H        ;A=45H
AGAIN:  JNB  P1.2,AGAIN ; get out when P1.2=1
        MOV  P0,A          ;issue A to P0
        SETB P2.3          ;make P2.3 high
        CLR  P2.3          ;make P2.3 low for H-to-L
```

8051 I/O PORTS PROGRAM - 5

Assume that bit P2.3 is an input and represents the condition of an oven. If it goes high, it means that the oven is hot. Monitor the bit continuously. Whenever it goes high, send a high-to-low pulse to port P1.5 to turn on a buzzer.

Solution:

```
HERE:   JNB   P2.3,HERE  ;keep monitoring for high
        SETB  P1.5        ;set bit P1.5=1
        CLR   P1.5        ;make high-to-low
        SJMP  HERE        ;keep repeating
```

8051 I/O PORTS PROGRAM - 6

A switch is connected to pin P1.7. Write a program to check the status of SW and perform the following:
(a) If SW=0, send letter 'N' to P2
(b) If SW=1, send letter 'Y' to P2
Use the carry flag to check the switch status.

Solution:

```
        SETB P1.7       ;make P1.7 an input
AGAIN:  MOV  C,P1.2     ;read SW status into CF
        JC   OVER       ;jump if SW=1
        MOV  P2,#'N'    ;SW=0, issue 'N' to P2
        SJMP AGAIN      ;keep monitoring
OVER:   MOV  P2,#'Y'    ;SW=1, issue 'Y' to P2
        SJMP AGAIN      ;keep monitoring
```

8051 I/O PORTS PROGRAM - 7

A switch is connected to pin P1.0 and an LED to pin P2.7. Write a program to get the status of the switch and send it to the LED

Solution:

```
        SETB P1.7          ;make P1.7 an input
AGAIN:  MOV  C,P1.0        ;read SW status into CF
        MOV  P2.7,C        ;send SW status to LED
        SJMP AGAIN         ;keep repeating
```

The instruction 'MOV P2.7,P1.0' is wrong , since such an instruction does not exist

However 'MOV P2,P1' is a valid instruction

8051

TIMERS/COUNTERS

TIMERS / COUNTERS

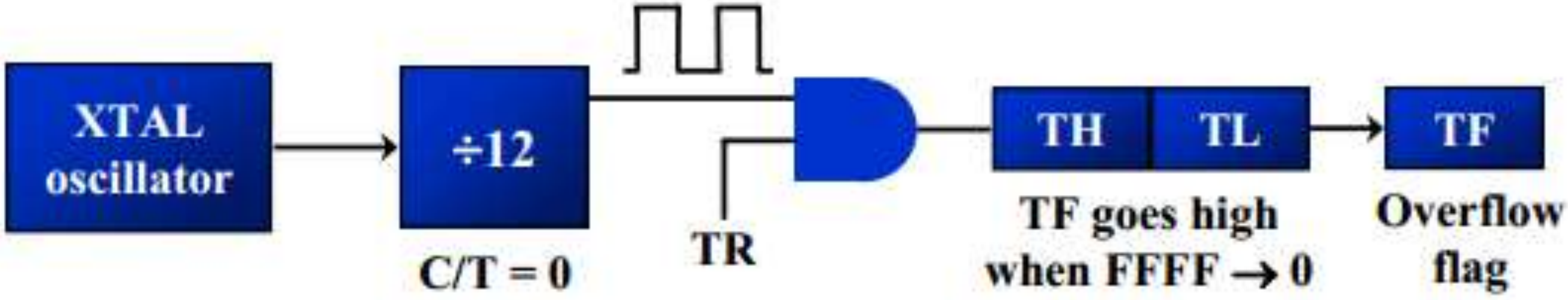
TIMERS

- The 8051 has two timers/counters, they can be used either as
  - Timers to generate a time delay
  - To generate a waveform with specific frequency
  - To generate baud rate signal for serial communication
  - 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

TIMERS / COUNTERS

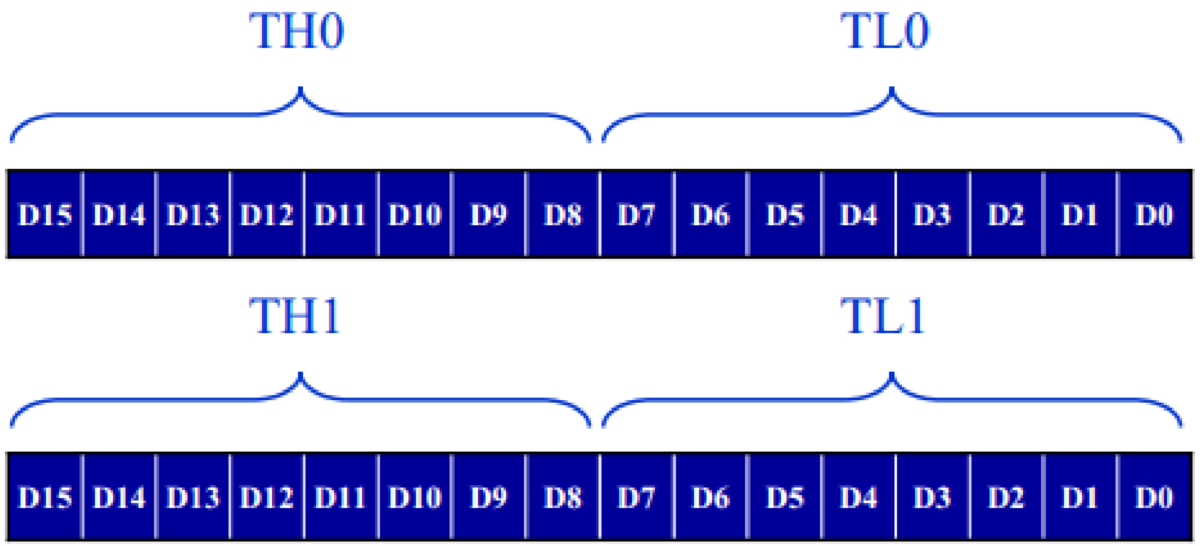
TIMERS

- Register related to work with 8051 timers are:
  - TH & TL Timer/counter register– To hold the value for generating time delay
  - TMOD Register - to define mode of timer operation
  - TCON Register – To control the timer operation



TIMER registers

- Accessed as low byte and high byte
  - The low byte register is called TL0/TL1 and
  - The high byte register is called TH0/TH1



TIMERS / COUNTERS

- Steps to calculate values to be loaded into the TL and TH registers for finding the TH, TL registers' values

  1. Divide the desired time delay by 1.085us (if operating frequency is 11.0592 MHz)

  2. Perform 65536 – n, where n is the decimal value we got in Step1

  3. Convert the result of Step2 to hex value, where yyxx is the initial hex value to be loaded into the timer's register

  4. Set TL = xx and TH = yy To generate a time delay

TIMERS / COUNTERS

- Example: 500us time delay

Step1: 500us/1.085us = 461pulses

Step2: P=65536-461=65075

Step3: 65074 converted by hexa decimal =FE33

Step4: TH1=0xFE; TL1=0x33;

TMOD REGISTER

- 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

TMOD REGISTER

(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)

Prescaler:
Divides down the clock signals used for the timer, giving reduced overflow rates

TIMERS / COUNTERS

TMOD REGISTER

- 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

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

TIMERS / COUNTERS

TCON REGISTER

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0
Timers				Interrupts			

- TF1, TF0 : Overflow flags for Timer 1 and Timer 0. *
- TR1, TR0 : Run control bits for Timer 1 and Timer 0. *
  Set to run, reset to hold.

- IT1, IT0 : Type bit for external interrupts.
  Set for falling edge interrupts, reset for low level interrupts.
- IE1, IE0 : Flag for external interrupts 1 and 0.
  Set by interrupt, cleared when interrupt is processed.

To generate a time delay

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 and 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

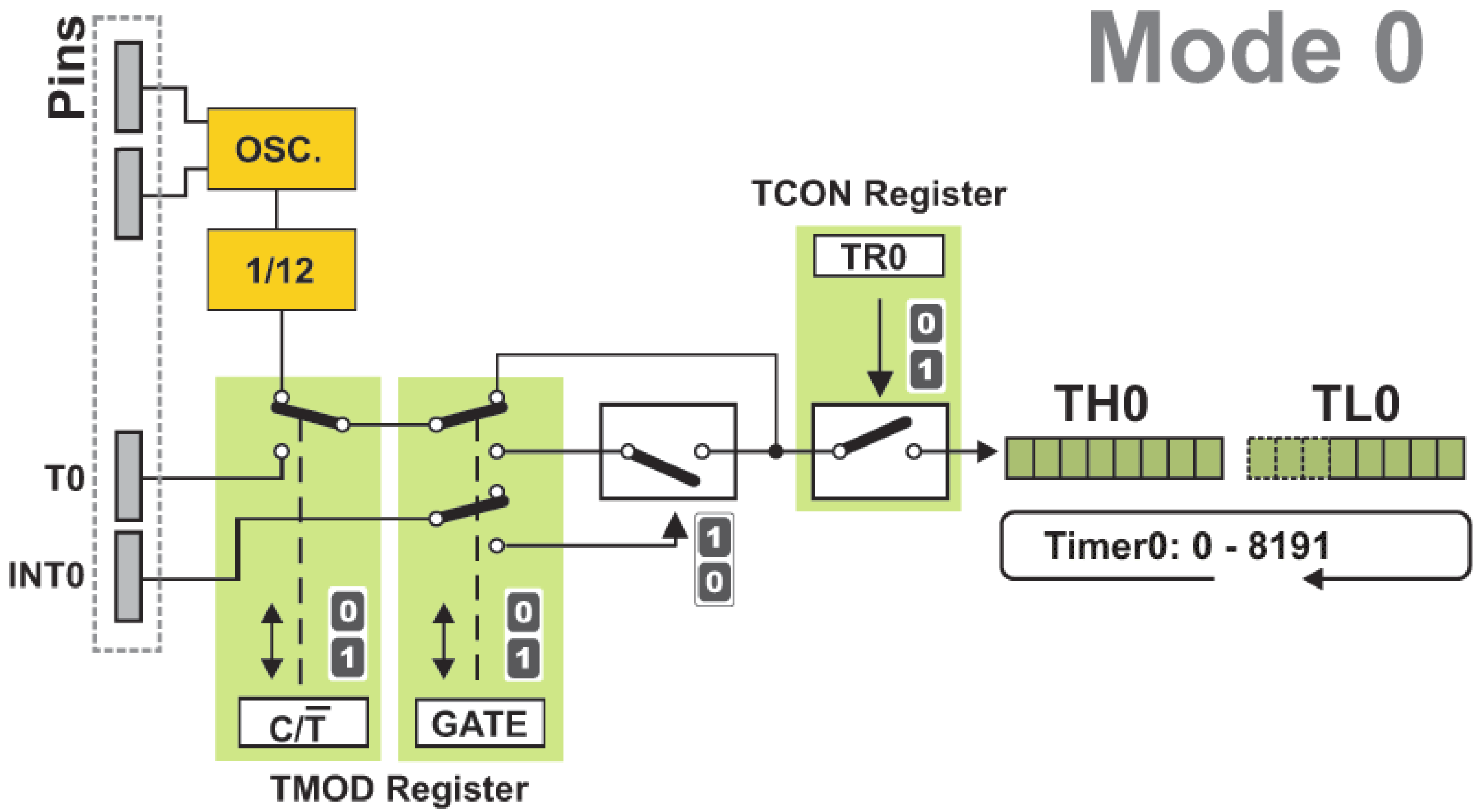
TIMERS

Timer 0 Mode 0 (13-bit timer)

- This is one of the rarities being kept only for the purpose of compatibility with the previous versions of microcontrollers.
- This mode configures timer 0 as a 13-bit timer which consists of all 8-bits of TH0 and the lower 5-bits of TL0.
- How does it operate? Each coming pulse causes the lower register bits to change their states. After receiving 32 pulses, this register is loaded and automatically cleared, while the higher byte (TH0) is incremented by 1.
- This process is repeated until registers count up 8192 pulses. After that, both registers are cleared and counting starts from 0

TIMERS

Timer 0 Mode 0 (13-bit timer)

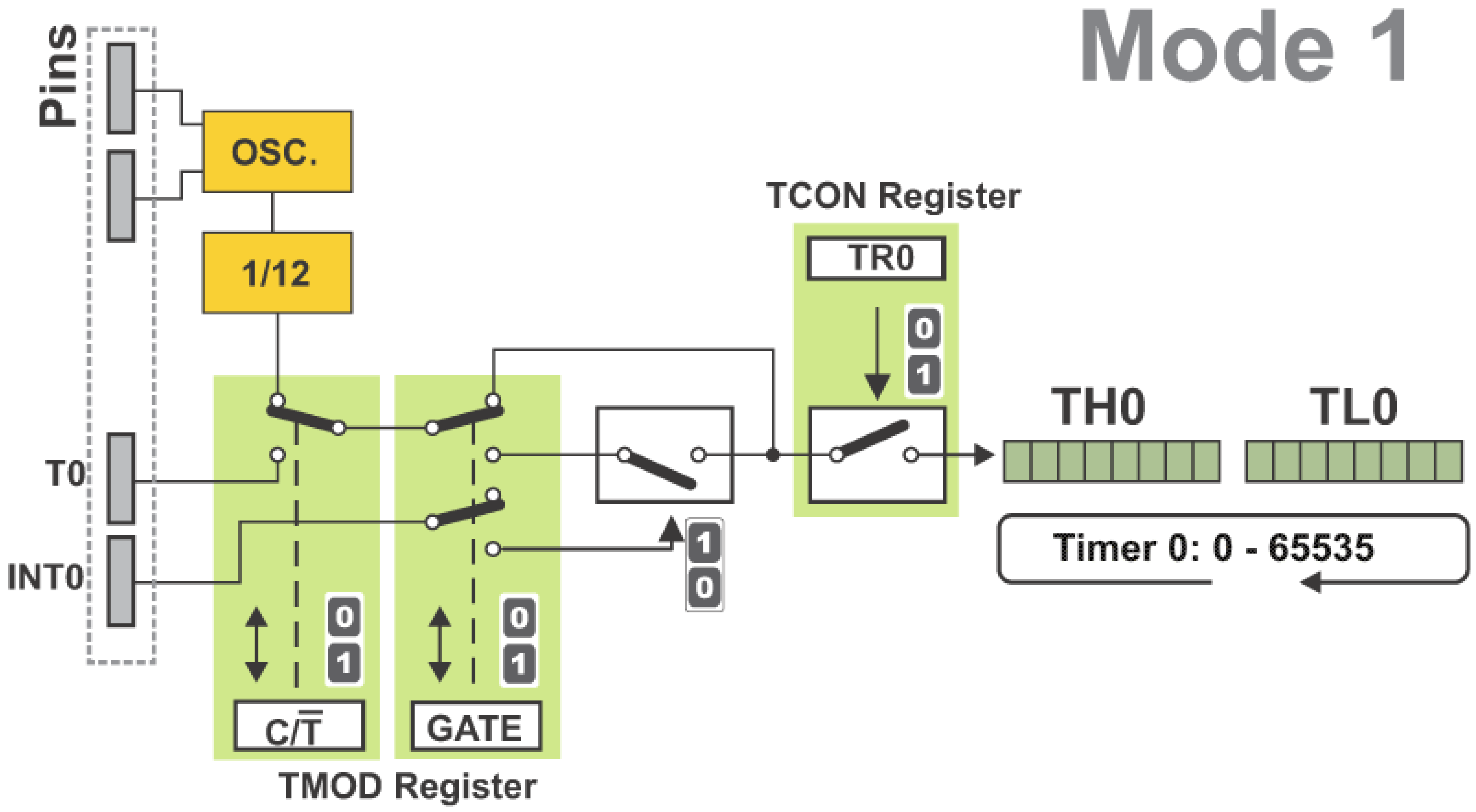


TIMERS

Timer 0 Mode 1 (16-bit timer)

- It is a 16-bit timer; therefore it allows values from 0000 to FFFFH to be loaded into the timer's registers TL and TH.
- After TH and TL are loaded with a 16-bit initial value, the timer must be started.
- After the timer is started. It starts count up until it reaches its limit of FFFFH.
- When it rolls over from FFFF to 0000H, it sets high a flag bit called TF (timer flag). This is one of the most commonly used modes.

Timer 0 Mode 1 (16-bit timer)



TIMERS

Steps to program Timer 0 Mode 1 (16-bit timer)

1. Choose mode 1 timer 0
   – MOV TMOD,#01H
2. Set the original value to TH0 and TL0.
   – MOV TH0,#FFH
   – MOV TL0,#FCH
3. You had better to clear the flag to monitor: TF0=0.
   – CLR TF0
4. Start the timer.
   – SETB TR0
5. The 8051 starts to count up by incrementing the TH0-TL0.
   TH0-TL0= FFFCH,FFFDH,FFFEH,FFFFH,0000H

Timer 0 Mode 1 (16-bit timer)

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

   TH0-TL0= FFFEH, FFFFH, 0000H (Now TF0=1)

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

   ```
   AGAIN:  JNB TF0, AGAIN
   ```

8. Clear TR0 to stop the process.

   ```
   CLR TR0
   ```

9. Clear the TF flag for the next round.

   ```
   CLR TF0
   ```

Timer 0 Mode 1 (16-bit timer)

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

Solution:

1. The period of the square wave = 1 / 50 Hz = 20 ms.
2. The high or low portion of the square wave = 10 ms.
3. 10 ms / 1.085 μs = 9216
4. 65536 – 9216 = 56320 in decimal = DC00H in hex.
5. TL1 = 00H and TH1 = DCH.

Timer 0 Mode 1 (16-bit timer)

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

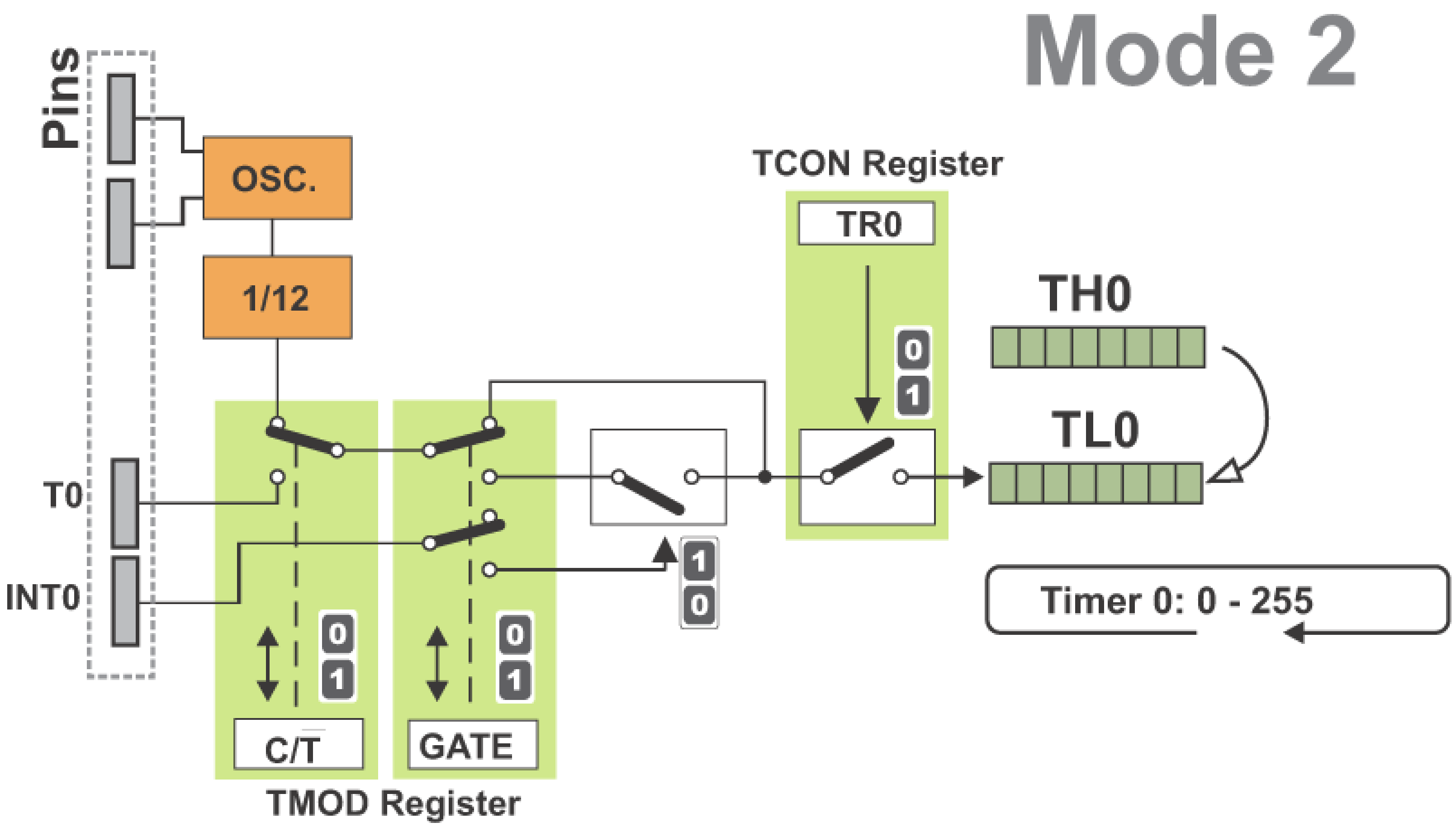
Timer 0 Mode-2 (Auto-reload Timer)

- When a timer is in mode 2, THx holds the "reload value" and TLx is the timer itself.
- Thus, TLx starts counting up. When TLx reaches 255 and is subsequently incremented, instead of resetting to 0 (as in the case of modes 0 and 1), it will be reset to the value stored in THx.
- For example, let's say TH0 holds the value FDh and TL0 holds the value FEh. After every two cycles TL0 reset to the value stored in TH0.
- The auto-reload mode is very commonly used for establishing a baud rate.

Timer 0 Mode-2 (Auto-reload Timer)

- In fact, only the TL0 register operates as a timer, while another TH0 register stores the value from which the counting starts.
- When the TL0 register is loaded, instead of being cleared the contents of TH0 will be reloaded to it.
- Suppose it is necessary to constantly count up 55 pulses generated by the clock.
- In order to register each 55th pulse, the best solution is to write the number 200 to the TH0 register and configure the timer to operate in mode 2.

Timer 0 Mode-2 (Auto-reload Timer)



Timer 0 Mode-2 (Auto-reload Timer)

1. Chose mode 2 timer 0
   MOV TMOD,#02H
2. Set the original value to TH0.
   MOV TH0,#38H
3. Clear the flag to TF0=0.
   CLR TF0
4. After TH0 is loaded with the 8-bit value, the 8051 gives a copy of it to TL0.
   TL0=TH0=38H
5. Start the timer.
   SETB TR0

Timer 0 Mode-2 (Auto-reload Timer)

6. The 8051 starts to count up by incrementing the TL0.
   - TL0= 38H, 39H, 3AH,....
7. When TL0 rolls over from FFH to 00, the 8051 set TF0=1. Also, TL0 is reloaded automatically with the value kept by the TH0.
   - TL0= FEH, FFH, 00H (Now TF0=1)
   - The 8051 auto reload TL0=TH0=38H.
   - Clr TF0
   - Go to Step 6 (i.e., TL0 is incrementing continuously).

- Note that **we must clear TF0** when TL0 rolls over. Thus, we can monitor TF0 in next process.
- Clear TR0 to stop the process.
  - Clr TR0

Timer 0 Mode-3 (Split Timer)

- Mode 3 configures timer 0 so that registers TL0 and TH0 operate as separate 8-bit timers.
- In other words, the 16-bit timer consisting of two registers TH0 and TL0 is split into two independent 8-bit timers.
- This mode is provided for applications requiring an additional 8-bit timer or counter.
- The TL0 timer turns into timer 0, while the TH0 timer turns into timer 1.

Timer 0 Mode-3 (Split Timer)

- In addition, all the control bits of 16-bit Timer 1 (consisting of the TH1 and TL1 register), now control the 8-bit Timer 1.

- Even though the 16-bit Timer 1 can still be configured to operate in any of modes, it is no longer possible to disable it as there is no control bit to do it.

- Thus, its operation is restricted when timer 0 is in mode 3.

Timer 0 Mode-3 (Split Timer)

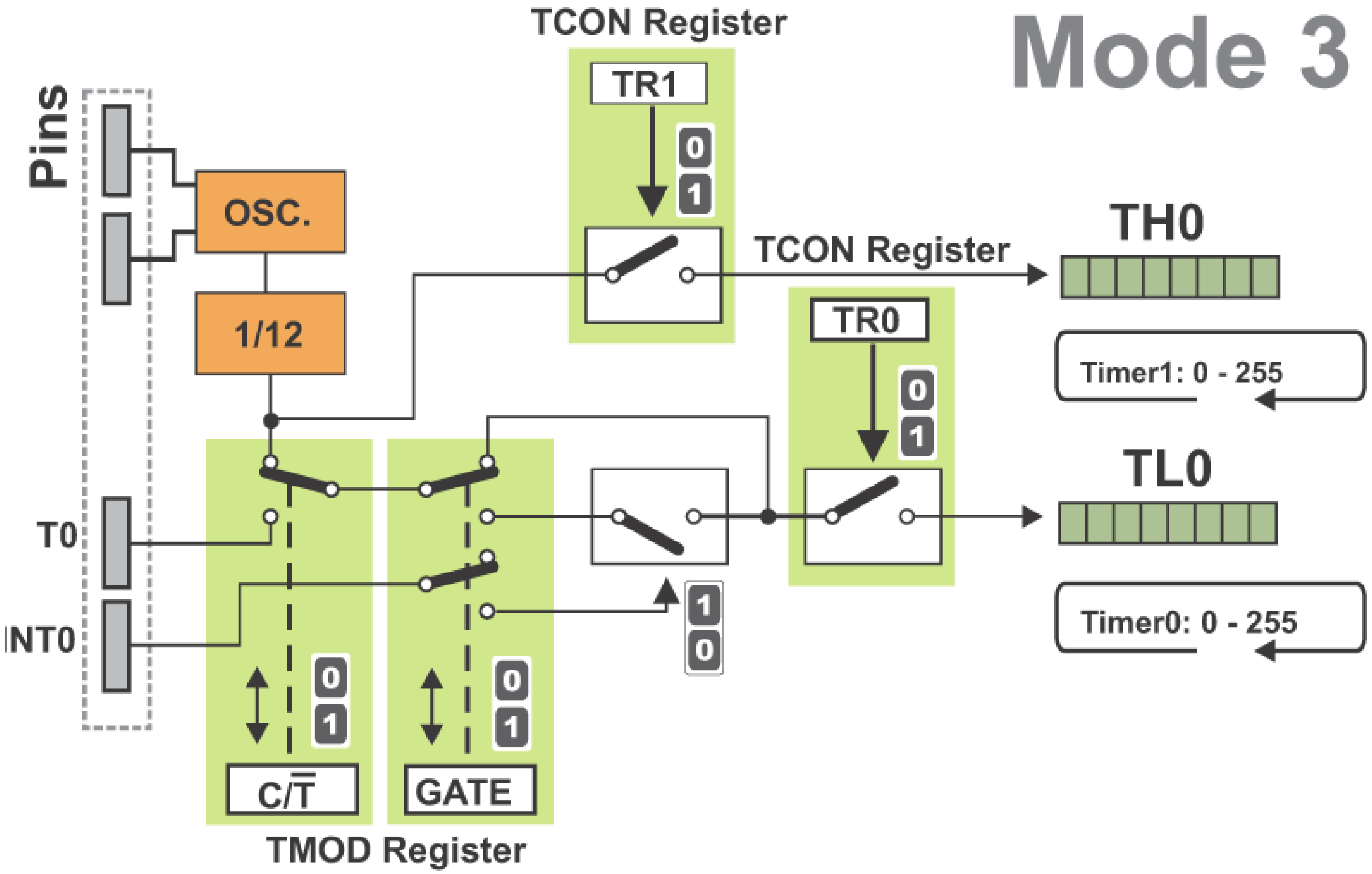
- While Timer 0 is in split mode, you may not start or stop the real timer 1 since the bits that do that are now linked to TH0.

- The only real use by using split timer mode is if you need to have two separate timers and, additionally, a baud rate generator.

- In such case you can use the real Timer 1 as a baud rate generator and use TH0/TL0 as two separate 8-bit timers.

TIMERS

Timer 0 Mode-3 (Split Timer)



TMOD REGISTER

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.

(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.

TIMERS

- **EXAMPLE-1:** In the following program, we create a square wave of 50% duty cycle (with equal portions high and low) on the P1.5bit. Timer 0 is used to generate the time delay. Analyze the program. Assume XTAL = 11.0592 MHz

```
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
```

TIMERS

```
DELAY: SETB TR0           ;start the timer 0
AGAIN: JNB TF0,AGAIN      ;monitor timer flag 0
CLR TR0                   ;stop timer 0
CLR TF0                   ;clear timer 0 flag
RET
```

- 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 ×1.085us = 15.19us for half the pulse.
- For the entire period it is T = 2 ×15.19us = 30.38us as the time delay generated by the timer.

TIMERS

- **EXAPMLE-2:** 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 P2.7

- 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.

TIMERS

```
CLR P2.7                    ;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
CPL P2.7          ;SET high P2.7
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
SJMP HERE
```

TIMERS

- **EXAPMLE-3:** Calculate TL and TH to get the largest time delay possible. Find the delay in ms. In your calculation, exclude the overhead due to the instructions in the loop

Solution:

- To get the largest delay we make TL and TH both 0. This will count up from 0000 to FFFFH and then roll over to zero
- Making TH and TL both zero means that the timer will count from 0000 to FFFF, and then roll over to raise the TF flag.
- As a result, it goes through a total Of 65536 states. Therefore, we have delay $= (65536 - 0) \times 1.085$ us $= 71.1065$ms.

- EXAPMLE-4: Assume that XTAL = 11.0592 MHz, write a program to generate a square wave of 2 kHz frequency on pin P1.7

Solution:

(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 FF1AH.

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

TIMERS

```
MOV TMOD,#10H              ;Timer 1, 16-bitmode
AGAIN: MOV TL1,#1AH        ;TL1=1A, low byte of timer
MOV TH1,#0FFH              ;TH1=FF, the high byte
CPL P1.7                   ;Complement P1.5
SETB TR1                   ;Start timer 1
BACK: JNB TF1,BACK         ;until timer rolls over
CLR TR1                    ;Stop the timer 1
CLR TF1                    ;Clear timer 1 flag
SJMP AGAIN                 ;Reload timer
```

- **EXAPMLE-5:** Generate a square wave with TON of 3ms and TOFF of 10ms on all pins of port 0. Assume an XTAL of 22MHz.

For 22MHz , one timer cycle time is

22MHz / 12 = 1.83MHz

T = 1/1.83M = 0.546 us

For OFF time calculation:

10ms/0.546 us = 18315 cycles

65536-18315 =47221 = B875H

For ON time calculation:

3ms/0.546us = 5494 cycles

65536 – 5494 = 60042=EA8AH

TIMERS

```
MOV TMOD, #01H
BACK: MOV TL0, #75H
MOV TH0, #0B8H
MOV P0, #00H
ACALL DELAY
MOV TL0, #8AH
MOV TH0, #0EAH
MOV P0, #0FFH
ACALL DELAY
SJMP BACK

ORG 300H
DELAY: SETB TR0
AGAIN: JNB TF0, AGAIN
CLR TR0
CLR TF0
RET
END
```

- **EXAPMLE-6:** Assuming XTAL = 22MHz write a program to generate a square pulse of 2 seconds period on pin P2.4. Use timer 1 mode.

Since square wave TON and TOFF values are equal. So 2s/2 = 1 second.

For TON (1 SECOND) time calculation:

- WHEN TL and TH takes value of 0 and the maximum delay is 65536 x 0.546us = 35782 us = 35.78ms.
- If maximum delay is repeated for 28 times we get 28x35.78 ms = 1001 ms = 1 second

TIMERS

```
        MOV TMOD, #10H
REPEAT: MOV R0, #28
        CPL P2.4
BACK:   MOV TL1, #00H
        MOV TH1, #00H
        SETB TR1
AGAIN:  JNB TF1, AGAIN
        CLR TR1
        CLR TF1
        DJNZ R0, BACK
        SJMP REPEAT
```

TIMERS

- EXAPMLE-7: Assuming XTAL = 22MHz write a program to generate a square pulse of frequency 100kHz on port pin P1.2.

- For 100kHz square wave,

(a). T=1/f = 0.01ms = 10uS

(b). TON = TOFF = 10uS/2 = 5uS

(c). 5us/ 0.546 us = 9 cycles

(d). 256 – 9 = 247 = 47H

```
      MOV TMOD, #20H
      MOV TH1, #0F7H
      SETB TR1
BACK: JNB TF1, BACK
      CPL P1.0
      CLR TF1
      SJMP BACK
```

COUNTERS

- 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

COUNTERS

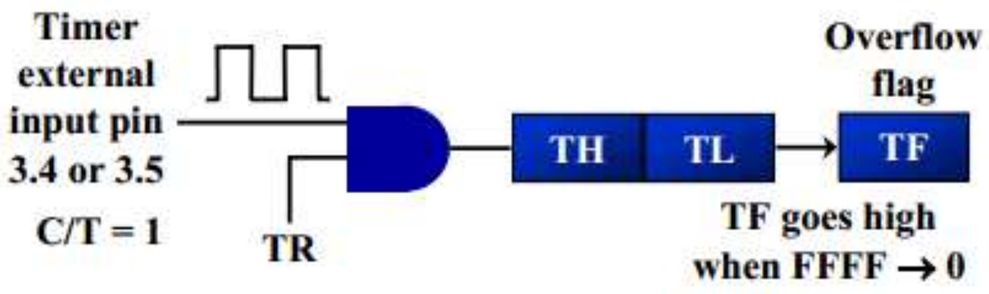
- The C/T bit in the TMOD registers decides the source of the clock for the timer
  - When C/T = 1, the timer is used as a counter and gets its pulses from outside the 8051
  - The counter counts up as pulses are fed from pins 14 and 15, these pins are called T0 (timer 0 input) and T1 (timer 1 input)

Port 3 pins used for Timers 0 and 1

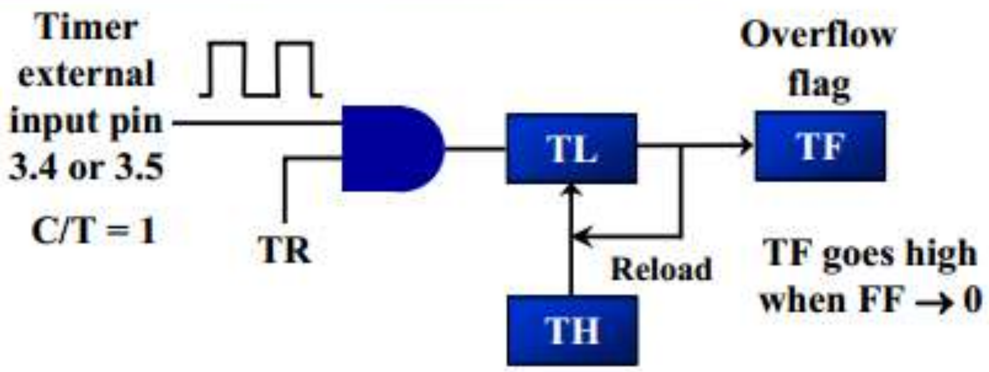
Pin	Port Pin	Function	Description
14	P3.4	T0	Timer/counter 0 external input
15	P3.5	T1	Timer/counter 1 external input

COUNTERS

Timer with external input (Mode 1)



Timer with external input (Mode 2)



COUNTERS

- **EXAPMLE-1:** 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   TMOD,#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
```

COUNTERS

- **EXAPMLE-2:** Write an 8051 assembly language program to implement a counter for counting pulses of an input signals. Assume the crystal frequency as 22 MHz. Configure TIMER 1 to generate a clock pulse for every one seconds at P3.5 and TIMER 0 as a counter which receives input pulses at P3.4 from P3.5 Display final count values in port P1 (TL0) & P2(TH0).

```
ORG 000H
        REPEAT: MOV TMOD, #15H
        SETB P3.4
        MOV TL0, #00
        MOV TH0, #00
        SETB TR0
```

COUNTERS

```
MOV R0,#28
AGAIN: MOV TL1,#00
MOV TH1, #00
SETB TR1
BACK: JNB TF1, BACK
CLR TF1
CLR TR1
DJNZ R0, AGAIN
```

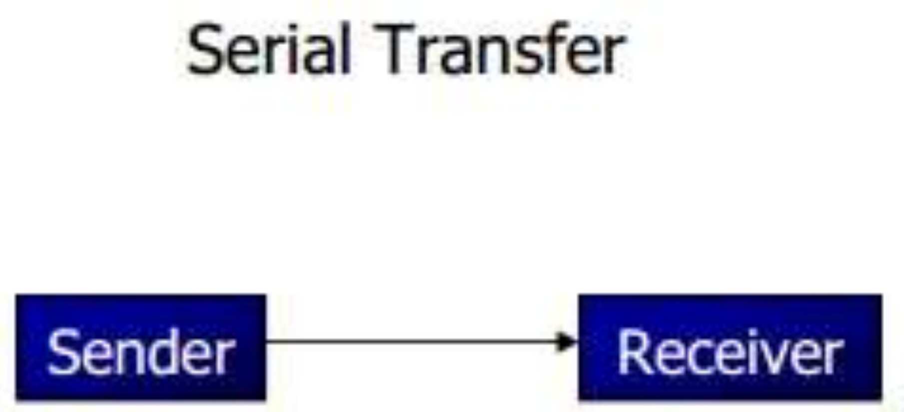
COUNTERS

```
MOV A, TL0
MOV P1,A
MOV A, TH0
MOV P2,A
SJMP REPEAT
END
```

SERIAL COMMUNICATION

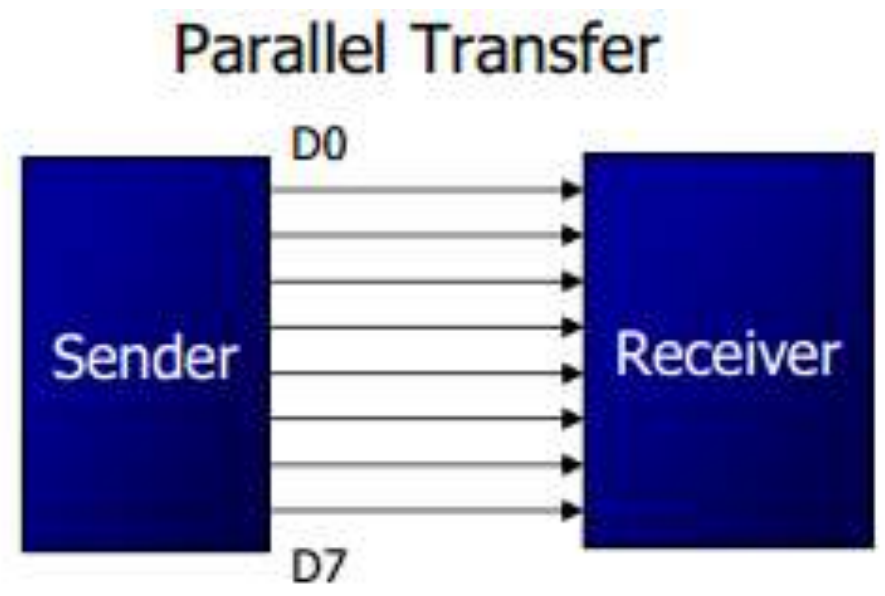
SERIAL COMMUNICATION

- Computers transfer data in two ways:
  - 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 COMMUNICATION

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

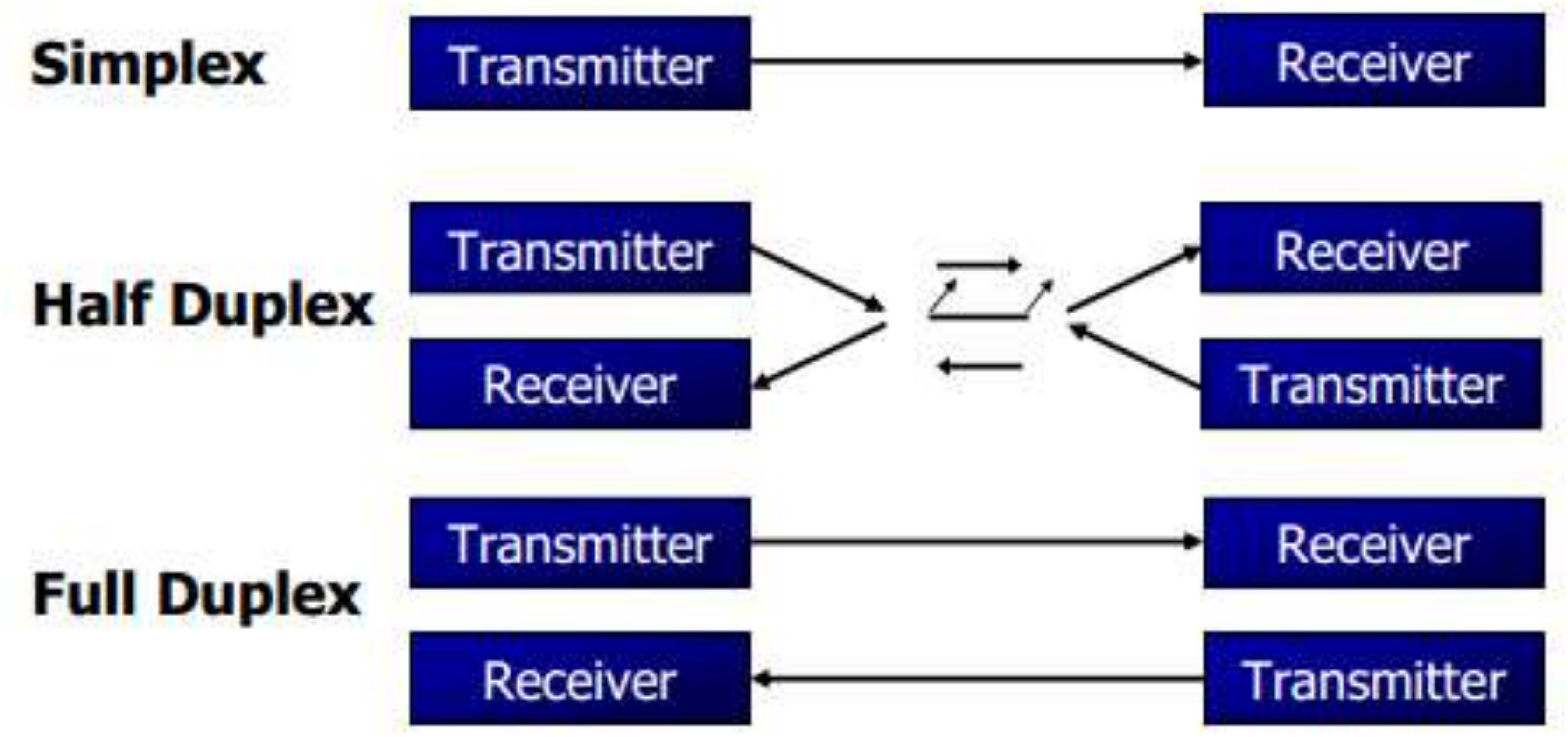


SERIAL COMMUNICATION

- 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
- There are special IC chips made by many manufacturers for serial communications
  - UART (universal asynchronous Receiver-transmitter)
  - USART (universal synchronous-asynchronous Receiver-transmitter)

SERIAL COMMUNICATION

- Data transmission types:



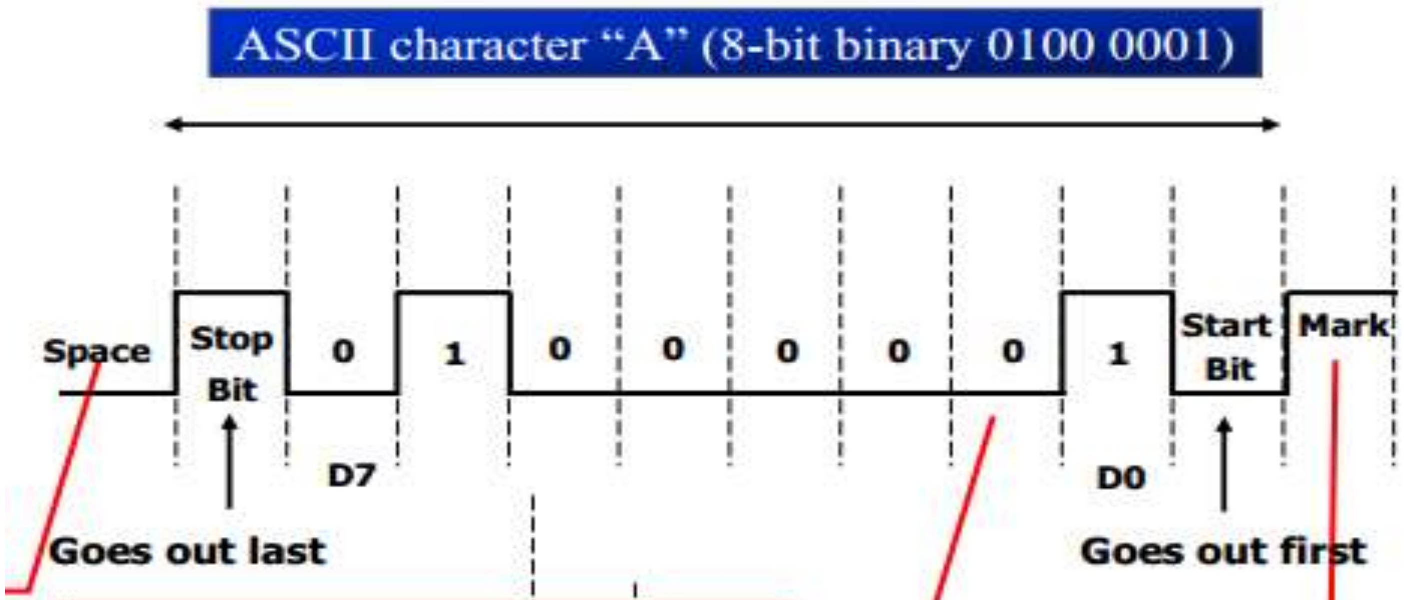
SERIAL COMMUNICATION

- 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 end
- Asynchronous serial data communication is widely used for character-oriented transmissions
  - Each character is placed in b/w start and stop bits, is called framing
  - Block-oriented data transfers use the synchronous method

SERIAL COMMUNICATION

- The start bit is always one bit, but the stop bit can be one or two bits
- The start bit is always a 0 (low) & the stop bit(s) is 1 (high)
- 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

SERIAL COMMUNICATION



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*

SERIAL COMMUNICATION

- 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
- The baud rate and bps are the same, and we use the terms inter changeably

SERIAL COMMUNICATION

- The data transfer rate of given computer system depends on communication ports incorporated into that system
- An interfacing standard RS232 was set by the Electronics Industries Association (EIA) in 1960
- In RS232, a 1 is represented by -3 ~ -25 V, while a 0 bit is +3 ~ +25 V, making -3 to +3 undefined
- The standard was set long before the advent of the TTL logic family, its input and output voltage levels are not TTL compatible

SERIAL COMMUNICATION

- Current terminology classifies data communication equipment as

  - DTE (data terminal equipment) refers to terminal and computers that send and receive data

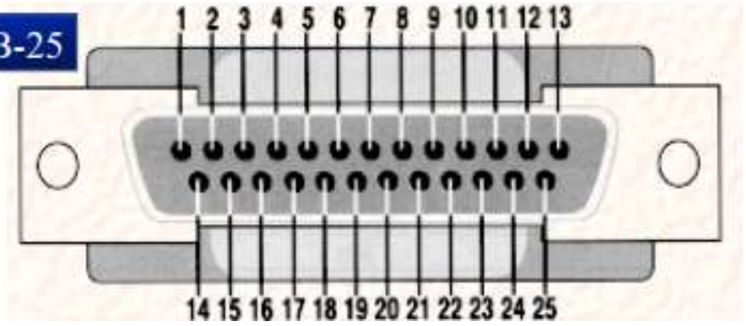
  - DCE (data communication equipment) refers to communication equipment, such as modems

SERIAL COMMUNICATION

RS232 DB-25 Pins

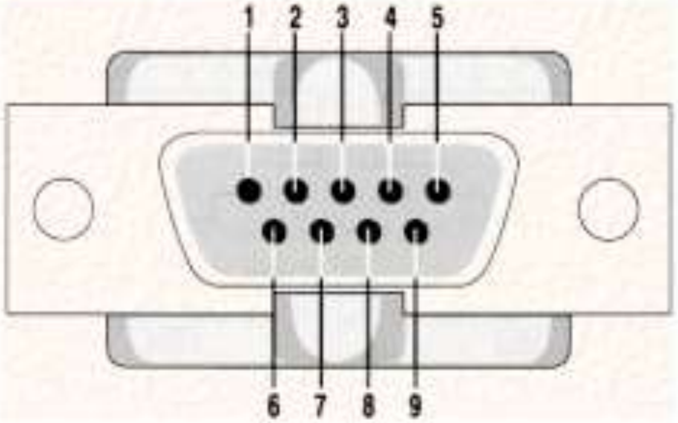
Pin	Description	Pin	Description
1	Protective ground	14	Secondary transmitted data
2	Transmitted data (TxD)	15	Transmitted signal element timing
3	Received data (RxD)	16	Secondary receive data
4	Request to send (-RTS)	17	Receive signal element timing
5	Clear to send (-CTS)	18	Unassigned
6	Data set ready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



SERIAL COMMUNICATION

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)

SERIAL COMMUNICATION

- **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

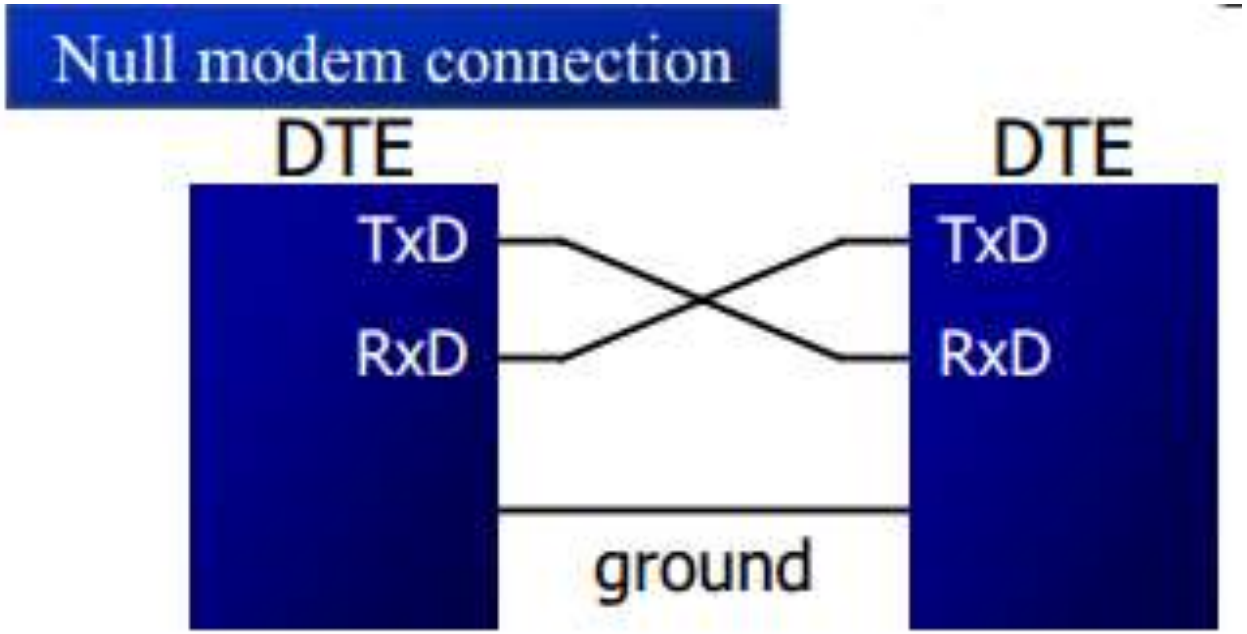
- **RTS (request to send)**
  - When the DTE device has byte to transmit, it assert RTS to signal the modem that it has a byte of data to transmit

SERIAL COMMUNICATION

- **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

SERIAL COMMUNICATION

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



SERIAL COMMUNICATION

- In data transmission, serial communication is the process of sending data one bit at a time, sequentially, over a communication channel or computer bus.

- To reduce the number of pins in a package, many ICs use a serial bus to transfer data when speed is not important.

- Some examples of such low-cost serial buses include SPI, I^2C, DC-BUS, UNI/O, and 1-Wire.

SERIAL COMMUNICATION

- 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

SERIAL COMMUNICATION

- 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

SERIAL COMMUNICATION

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

SERIAL COMMUNICATION

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

SERIAL COMMUNICATION

- 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

  - SBUF holds the byte of data when it is received by 8051 RxD line

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

SERIAL COMMUNICATION

- SCON is an 8-bit the special function register (bit-addressable).
- This register contain not only the mode selection bits but also the 9th data bit for transmit and receive (TB8 and RB8) and the serial port interrupt bits (TI and RI).

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

SERIAL COMMUNICATION

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*

SERIAL COMMUNICATION

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):

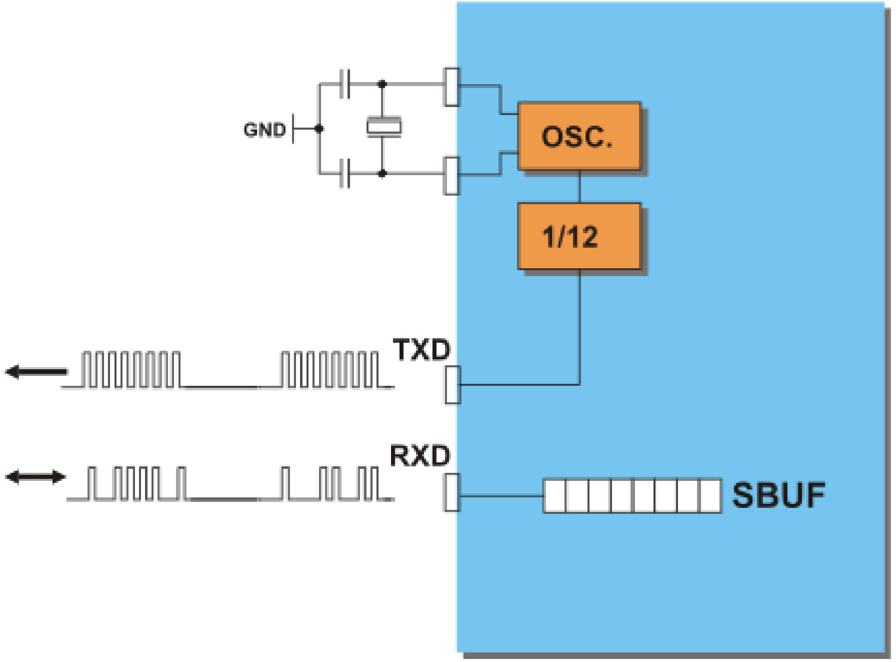
When 8051 finishes the transfer of 8-bit character It raises TI flag to indicate that it is ready to transfer another byte

RI (receive interrupt):

When 8051 receives data serially via RxD, it raises the RI flag bit to indicate that a byte has been received and should be picked up before it is lost

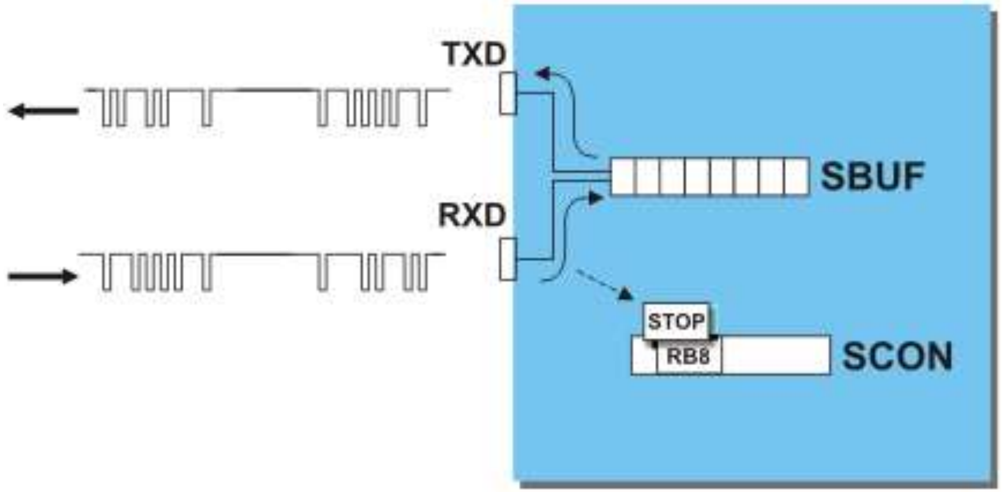
SERIAL COMMUNICATION

- In mode 0, serial data are transmitted and received through the RXD pin, while the TXD pin output clocks. The baud rate is fixed at 1/12 the oscillator frequency. On transmit, the least significant bit (LSB bit) is sent/received first.



SERIAL COMMUNICATION

- In mode 1, 10 bits are transmitted through the TXD pin or received through the RXD pin in the following manner: a START bit (always 0), 8 data bits (LSB first) and a STOP bit (always 1). The START bit is only used to initiate data receive, while the STOP bit is automatically written to the RB8 bit of the SCON register.



SERIAL COMMUNICATION

- In mode 2, 11 bits are transmitted through the TXD pin or received through the RXD pin: a START bit (always 0), 8 data bits (LSB first), a programmable 9th data bit and a STOP bit (1).

- On transmit, the 9th data bit is actually the TB8 bit of the SCON register. This bit usually has a function of parity bit. On receive, the 9th data bit goes into the RB8 bit of the same register (SCON).

- The baud rate is either 1/32 or 1/64 of the oscillator frequency.

- Mode 3 is the same as Mode 2 in all respects except the baud rate. The baud rate in Mode 3 is variable.

SERIAL COMMUNICATION

- Step to program 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

SERIAL COMMUNICATION

4. TR1 is set to 1 to start timer 1

5. The character byte to be transferred serially is written into SBUF register

6. The TI flag bit is monitored with the use of instruction JNB TI, xx to see if the character has been transferred completely

7. TI is cleared by CLR TI instruction

8. To transfer the next byte, go to step 5

SERIAL COMMUNICATION

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
```

SERIAL COMMUNICATION

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
```

SERIAL COMMUNICATION

```
        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
```

SERIAL COMMUNICATION

- 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

SERIAL COMMUNICATION

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

SERIAL COMMUNICATION

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
```

SERIAL COMMUNICATION

- 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

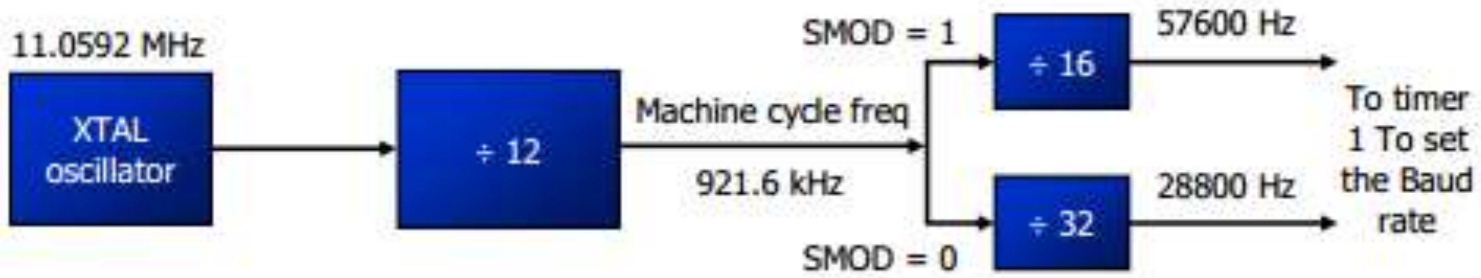
The system crystal is fixed

- 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

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

SERIAL COMMUNICATION



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

SERIAL COMMUNICATION

Assume a switch is connected to pin P1.7. Write a program to monitor its status and send two messages to serial port continuously as follows:

SW=0 send "NO"

SW=1 send "YES"

Assume XTAL = 11.0592 MHz, 9600 baud, 8-bit data, and 1 stop bit.

```
        ORG   0H              ;starting position
MAIN:   MOV   TMOD,#20H
        MOV   TH1,#-3         ;9600 baud rate
        MOV   SCON,#50H
        SETB  TR1             ;start timer
        SETB  P1.7            ;make SW an input
S1:     JB    P1.7,NEXT       ;check SW status
        MOV   DPTR,#MESS1     ;if SW=0 display "NO"
```

SERIAL COMMUNICATION

```
FN:     CLR    A
        MOVC   A,@A+DPTR      ;read the value
        JZ     S1             ;check for end of line
        ACALL  SENDCOM        ;send value to serial port
        INC    DPTR           ;move to next value
        SJMP   FN             ;repeat
NEXT:   MOV    DPTR,#MESS2    ;if SW=1 display "YES"
LN:     CLR    A
        MOVC   A,@A+DPTR      ;read the value
        JZ     S1             ;check for end of line
        ACALL  SENDCOM        ;send value to serial port
        INC    DPTR           ;move to next value
        SJMP   LN             ;repeat
```

SERIAL COMMUNICATION

```
;-------------------
SENDCOM:    MOV    SBUF,A        ;place value in buffer
HERE:       JNB    TI,HERE       ;wait until transmitted
            CLR    TI            ;clear
            RET                  ;return
;-------------------
MESS1:      DB     "NO",0
MESS2:      DB     "YES",0
                   END
```

DB (define byte)

The DB directive is the most widely used data directive in the assembler. It is used to define the 8-bit data. When DB is used to define data, the numbers can be in decimal, binary, hex, or ASCII formats. For decimal, the "D" after the decimal number is optional, but using "B" (binary) and "H" (hexadecimal) for the others is required. Regardless of which is used, the assembler will convert the numbers into hex. To indicate ASCII, simply place the characters in quotation marks ('like this'). The assembler will assign the ASCII code for the numbers or characters automatically. The DB directive is the only directive that can be used to define ASCII strings larger than two characters; therefore, it should be used for all ASCII data definitions. Following are some DB examples:

```
            ORG   500H
DATA1:      DB    28                   ;DECIMAL(1C in hex)
DATA2:      DB    00110101B            ;BINARY (35 in hex)
DATA3:      DB    39H                  ;HEX
            ORG   510H
DATA4:      DB    "2591"               ;ASCII NUMBERS
            ORG   518H
DATA6:      DB    "My name is Joe"     ;ASCII CHARACTERS
```

Either single or double quotes can be used around ASCII strings. This can be useful for strings, which contain a single quote such as "O'Leary". DB is also used to allocate memory in byte-sized chunks.

SERIAL COMMUNICATION

Assume that the 8051 serial port is connected to the COM port of IBM PC, and on the PC, we are using the terminal.exe program to send and receive data serially. P1 and P2 of the 8051 are connected to LEDs and switches, respectively. Write an 8051 program to (a) send to PC the message "We Are Ready", (b) receive any data send by PC and put it on LEDs connected to P1, and (c) get data on switches connected to P2 and send it to PC serially. The program should perform part (a) once, but parts (b) and (c) continuously, use 4800 baud rate.

SERIAL COMMUNICATION

Solution:

```
        ORG   0
        MOV   P2,#0FFH      ;make P2 an input port
        MOV   TMOD,#20H     ;timer 1, mode 2
        MOV   TH1,#0FAH     ;4800 baud rate
        MOV   SCON,#50H     ;8-bit, 1 stop, REN enabled
        SETB  TR1           ;start timer 1
        MOV   DPTR,#MYDATA ;load pointer for message
H_1:    CLR   A
        MOV   A,@A+DPTR     ;get the character
```

SERIAL COMMUNICATION

```
        JZ    B_1           ;if last character get out
        ACALL SEND          ;otherwise call transfer
        INC   DPTR          ;next one
        SJMP  H_1           ;stay in loop
B_1:    MOV   a,P2          ;read data on P2
        ACALL SEND          ;transfer it serially
        ACALL RECV          ;get the serial data
        MOV   P1,A          ;display it on LEDs
        SJMP  B_1           ;stay in loop indefinitely
```

SERIAL COMMUNICATION

```
;----serial data transfer. ACC has the data------
SEND:   MOV   SBUF,A        ;load the data
H_2:    JNB   TI,H_2        ;stay here until last bit
                            ;gone
        CLR   TI            ;get ready for next char
        RET                 ;return to caller
;----Receive data serially in ACC----------------
RECV:   JNB   RI,RECV       ;wait here for char
        MOV   A,SBUF        ;save it in ACC
        CLR   RI            ;get ready for next char
        RET                 ;return to caller
...
```

INTERRUPTS

INTERRUPTS

- A single microcontroller can serve several devices by two ways: (i) Interrupt (ii). Polling
- 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

INTERRUPTS

- 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)

INTERRUPTS

- 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

INTERRUPTS

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

1. It finishes the instruction it is executing and saves the address of the next instruction (PC) on the stack

2. It also saves the current status of all the interrupts internally (i.e: not on the stack)

3. It jumps to a fixed location in memory, called the interrupt vector table, that holds the address of the ISR

INTERRUPTS

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

INTERRUPTS

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.

INTERRUPTS

- 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

- 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. If EA = 1, interrupts are enabled and will be responded to if their corresponding bits in IE are high
  3. If EA = 0, no interrupt will be responded to, even if the associated bit in the IE register is high

INTERRUPTS

IE (Interrupt Enable) Register

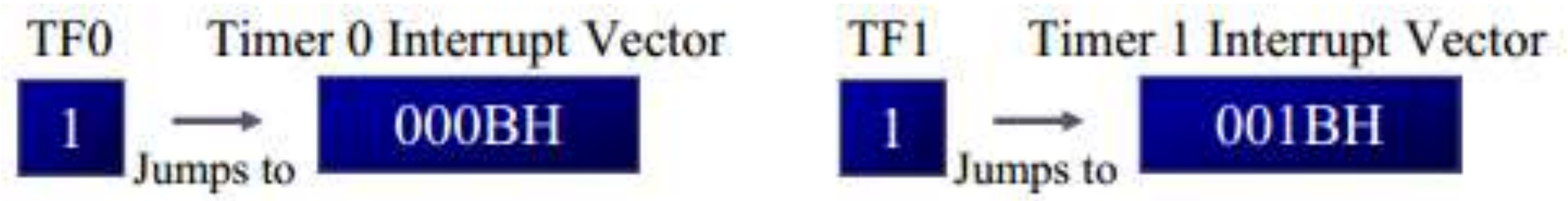
D7							D0
EA	--	ET2	ES	ET1	EX1	ET0	EX0

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

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, it can't do anything else
  - Using interrupts solves this problem and, avoids tying down the controller
  - If the timer interrupt in the IE register



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 us = 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
```

INTERRUPTS

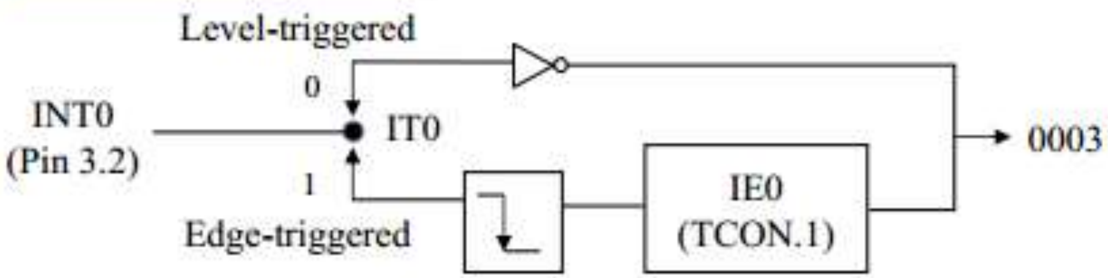
```
;--The main program for initialization
        ORG   0030H      ;after vector table space
MAIN:   MOV   TMOD,#02H  ;Timer 0, mode 2
        MOV   P0,#0FFH   ;make P0 an input port
        MOV   TH0,#-92   ;TH0=A4H for -92
        MOV   IE,#82H    ;IE=10000010 (bin) enable
                         ;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
```

INTERRUPTS

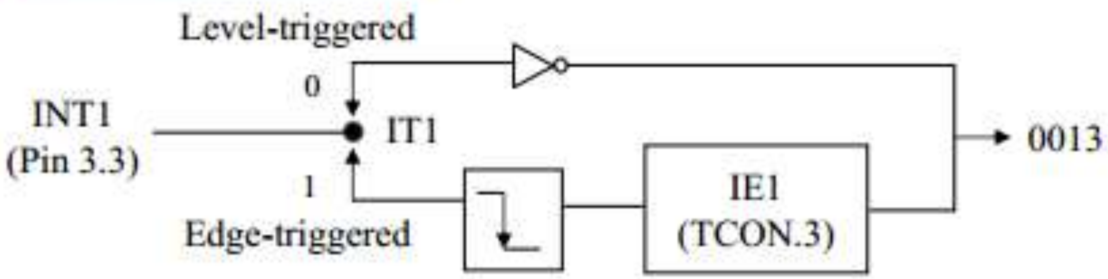
- The 8051 has two external hardware interrupts
  - Pin 12 (P3.2) and pin 13 (P3.3) of the 8051, designated as INT0 and INT1, are used as external hardware interrupts
  - The interrupt vector table locations 0003H and 0013H are set aside for INT0 and INT1
  - There are two activation levels for the external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

INTERRUPTS

Activation of INT0



Activation of INT1

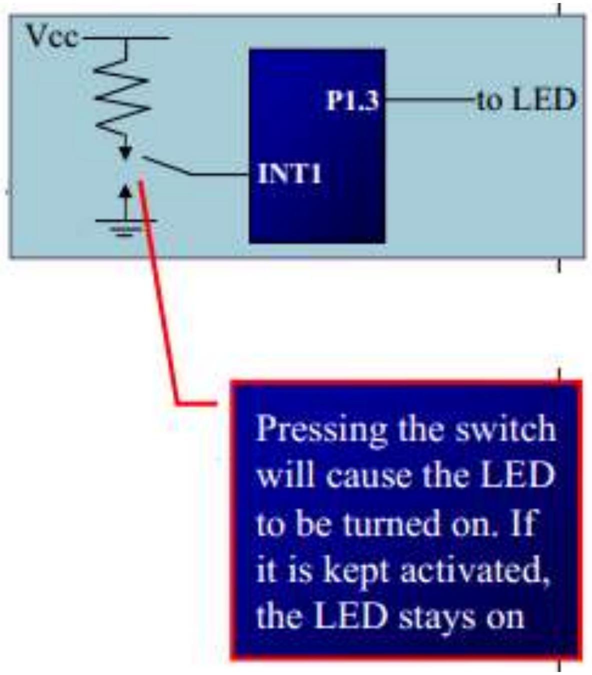


INTERRUPTS

- 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

INTERRUPTS

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. As long as the switch is pressed low, the LED should stay on. Simultaneously perform a toggle operation in P1.5 with the delay of 500ms.



INTERRUPTS

```
ORG 0000H
LJMP main

//ISR for INT1
    ORG 0013H
    SETB P1.3
    MOV R3,#255
Back: DJNZ R3, Back
    CLR P1.3
    RETI
```

INTERRUPTS

```
        ORG 30H
main: MOV IE,#10000100B
Here:  SETB P1.5
        ACALL DELAY
        CLR P1.5
        ACALL DELAY
       SJMP Here
```

```
//Delay of 500ms
DELAY:   MOV R2,#04H        ;LOAD R2 WITH 07 HEX
HERE3:   MOV R1,#0FFH       ;LOAD R1 WITH 0FF HEX
HERE2:   MOV R0,#0FFH       ;LOAD R2 WITH 0FF HEX
HERE1:   DJNZ R0,HERE1      ;DECREMENT R0
         DJNZ R1,HERE2      ;DECREMENT R1
         DJNZ R2,HERE3      ;DECREMENT R2
         RET                ;RETURN
         END
```

INTERRUPTS

- 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

- To make INT0 and INT1 edge-triggered interrupts, we must program the bits of the TCON register

INTERRUPTS

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

INTERRUPTS

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

INTERRUPTS

- 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

INTERRUPTS

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
```

INTERRUPTS

Write a program using interrupts to do the following:

(a) Receive data serially and sent it to P0,

(b) Have P1 port read and transmitted serially, and a copy given to P2,

(c) Make timer 0 generate a square wave of 5kHz frequency on P0.1.

Assume that XTAL-11,0592. Set the baud rate at 4800.

Solution:

```
ORG   0
LJMP  MAIN
ORG   000BH   ;ISR for timer 0
CPL   P0.1    ;toggle P0.1
RETI          ;return from ISR
ORG   23H     ;
LJMP  SERIAL  ;jump to serial interrupt ISR
```

INTERRUPTS

```
         ORG   30H
MAIN:    MOV   P1,#0FFH ;make P1 an input port
         MOV   TMOD,#22H;timer 1,mode 2(auto reload)
         MOV   TH1,#0F6H;4800 baud rate
         MOV   SCON,#50H;8-bit, 1 stop, ren enabled
         MOV   TH0,#-92 ;for 5kHZ wave
         MOV   IE,10010010B ;enable serial int.
         SETB  TR1          ;start timer 1
         SETB  TR0          ;start timer 0
BACK:    MOV   A,P1         ;read data from port 1
         MOV   SBUF,A       ;give a copy to SBUF
         MOV   P2,A         ;send it to P2
         SJMP  BACK         ;stay in loop indefinitely
```

INTERRUPTS

```
;-------------------SERIAL PORT ISR
        ORG  100H
SERIAL:JB    TI,TRANS;jump if TI is high
        MOV  A,SBUF   ;otherwise due to receive
        MOV  P0,A     ;send serial data to P0
        CLR  RI       ;clear RI since CPU doesn't
        RETI          ;return from ISR
TRANS:  CLR  TI       ;clear TI since CPU doesn't
        RETI          ;return from ISR
        END
```

INTERRUPTS

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
Timer 2	TF2	T2CON.7 (AT89C52)
Timer 2	EXF2	T2CON.6 (AT89C52)

INTERRUPTS

- 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)

INTERRUPTS

- Discuss what happens if interrupts INT0, TF0, and INT1 are activated at the same time. Assume priority levels were set by the power-up reset and the external hardware interrupts are edge-triggered.

Solution:

- If these three interrupts are activated at the same time, they are latched and kept internally.
- Then the 8051 checks all five interrupts according to the sequence listed above Table.
- If any is activated, it services it in sequence.
- Therefore, when the above three interrupts are activated, IE0 (external interrupt 0) is serviced first, then timer 0 (TF0), and finally IE1 (external interrupt 1)

INTERRUPTS

- 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)

- To give a higher priority to any of the interrupts, we make the corresponding bit in the IP register high

- When two or more interrupt bits in the IP register are set to high they are serviced according to the sequence of Table

INTERRUPTS

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

INTERRUPTS

- a) Program the IP register to assign the highest priority to INT1(external interrupt 1), then
- (b) discuss what happens if INT0, INT1, and TF0 are activated at the same time. Assume the interrupts are both edge-triggered
- ```
  MOV IP,#00000100B       ;IP.2=1 assign INT1 higher priority.
          (OR)
  SETB IP.2               ;IP.2=1 assign INT1 higher priority.
  ```
- When INT0, INT1, and TF0 interrupts are activated at the same time, the 8051 services INT1 first, then it services INT0, then TF0.
  ```

THANK YOU