Answers.

Solution 1:

The 8051 has two timers: timer0 and timer1. They can be used either as timers or as counters. Both timers are 16 bits wide. Since the 8051 has an 8-bit architecture, each 16-bit is accessed as two separate registers of low byte and high byte. First we shall discuss about Timer0 registers.

Timer0 registers is a 16 bits register and accessed as low byte and high byte. The low byte is referred as a TL0 and the high byte is referred as TH0. These registers can be accessed like any other registers.

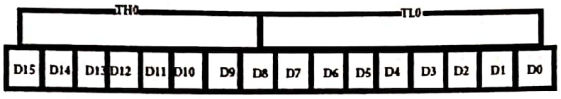


Fig. Timer0

Timer1 registers is also a 16 bits register and is split into two bytes, referred to as TL1 and TH1.

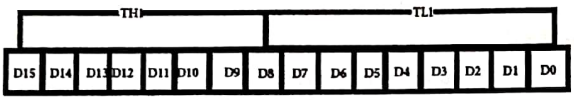


Fig. Timer1

TMOD (timer mode) Register: This is an 8-bit register which is used by both timers 0 and 1 to set the various timer modes. In this TMOD register, lower 4 bits are set aside for timer0 and the upper 4 bits are set aside for timer1. In each case, the lower 2 bits are used to set the timer mode and upper 2 bits to specify the operation.

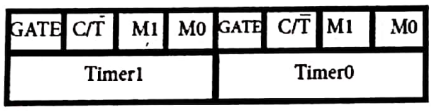


Fig. TMOD Register

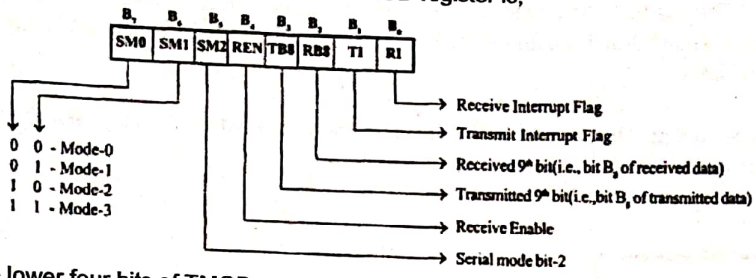
In upper or lower 4 bits, first bit is a GATE bit. Every timer has a means of starting and stopping. Some timers do this by software, some by hardware, and some have both software and hardware controls. The hardware way of starting and stopping the timer by an external source is achieved by making GATE=1 in the TMOD register. And if we change to GATE=0 then we do no need external hardware to start and stop the timers.
The second bit is C/T bit and is used to decide whether a timer is used as a time delay generator or an event counter. If this bit is 0 then it is used as a timer and if it is 1 then it is used as a counter. In upper or lower 4 bits, the last bits third and fourth are known as M1 and M0 respectively. These are used to select the timer mode.

M0	M1	Mode	Operating Mode
0	0	0	13-bit timer mode
0	1	1	16-bit timer mode,
1	0	2	8-bit auto reload mode
1	1	3	Spilt timer mode.

Mode 1 and mode 2 are most widely used.

Solution 2: -
Solution 3:

The TMOD register is used to select the operating mode and the timer/counter operation of the timers. The format of TMOD register is,



The lower four bits of TMOD register is used to control timer-0 and the upper four bits are used to control timer-1.

- The two timers can be independently program to operate in various modes.
- The TMOD register has two separates two-bit field M0 and MI to program the operating mode of timers. The operating modes of timers are mode-0, mode-1, mode-2 and mode-3. In all these operating modes the oscillator clock is divided by 12 and applied as input clock to timer.

MODE-0

- in mode-0 the timer register is configured as 13-bit register.
- for timer-1 the 8 bits of TH1 and lower 5 bits of TL1 are used to form 13-bit register.
- for timer-0 the 8-bit of TH0 and lower 5 bits of TL0 are used to form 13-bit register.
- the upper three bits of TL registers are ignored.
- For every clock input to timer the 13-bit timer register is incremented by one When the timer count rolls over from all 1's to all 0's, (i.e., 1 1111 1111 1111 to 0 0000 0000 0000) the timer interrupt flag in TCON register is set to one.

Mode-1

- The mode-1 is same as mode-0 except the size of the timer register. In mode-1 the TH and TL registers are cascaded to form 16-bit timer register.

MODE-2

- In mode-2, the timers function as 8-bit timer with automatic reload feature. The TL register will function as 8-bit timer count register and the TH register will hold an initial count value.
- When the timer is started, the initial value in TH is loaded to TL and for each clock input to timer the 8-bit timer count register is incremented by one.
- When the timer count rolls over from all 1's to all 0's (i.e., 1111 1111 to 0000 0000), the timer interrupt flag in TCON register is set to one and the content of TH register is reloaded in TL register and the count process starts again from this initial value.

Mode-3

- In mode-3, the timer-0 is configured as two separate 8-bit timers and the timer-1 is stopped.
- In mode-3 the TL0 will function as 8-bit timer controlled by standard timer-0 control bits and the TH0 will function as 8-bit timer controlled by timer-1 control bits.
- While timer-0 is programmed in mode-3, the timer-0 can be programmed in mode-0, 1 or 2 and can be used for an application that does not require an interrupt.
- The C/T(Low) bit of TMOD register is used to program the counter or timer operation of the timer. When C/T bit is set to one, the timer will function as event counter. The C/T(Low) bit is programmed to zero for timer operation.
- The timer will run only if clock input is allowed.
- When GATE = 1, the clock input to timer is allowed only if the signal at pin is high and when GATE =0 the signal at INT (low) pin is ignored.

Solution 4:
Solution 5:

A . XTAL = 11.0592

F=1/12 * XTAL
= 1/12 * 11.0592
= 0.9216

B . Xtal = 16

F=1/12 *XTAL
= 1/12 * 16
= 1.33

Solution 6:
Solution 7:

```
#include <reg51.h>
void T0Delay(void);
void main(void)
  {
   while(1)              //repeat forever
    {
      P1=0x55;           //toggle all bits of P1
      T0Delay();         //delay size unknown
      P1=0xAA;           //toggle all bits of P1
      T0Delay();
    }
  }

void T0Delay()
  {
    TMOD=0x01;           //Timer 0, Mode 1
    TL0=0x00;            //load TL0
    TH0=0x35;            //load TH0
    TR0=1;               //turn on T0
    while(TF0==0);       //wait for TF0 to roll over
    TR0=0;               //turn off T0
    TF0=0;               //clear TF0
  }
```

Solution 8:

```
#include <reg51.h>
void T0M2Delay(void);
sbit mybit=P1^5;
void main(void)
  {
    unsigned char x, y;
    while(1)
      {
        mybit=~mybit;            //toggle P1.5
        for(x=0;x<250;x++)       //due to for loop overhead
          for(y=0;y<36;y++)      //we put 36 and not 40
            T0M2Delay();
      }
  }

void T0M2Delay(void)
  {
    TMOD=0x02;              //Timer 0, mode 2(8-bit auto-reload)
    TH0=-23;                //load TH0(auto-reload value)
    TR0=1;                  //turn on T0
    while(TF0==0);          //wait for TF0 to roll over
    TR0=0;                  //turn off T0
    TF0=0;                  //clear TF0
  }
```

Solution 9:

```
#include <reg51.h>
sbit T1 = P3^5;
void main(void)
  {
    T1=1;                    //make T1 an input
    TMOD=0x60;               //
    TH1=0;                   //set count to 0

    while(1)                 //repeat forever
      {
        do
          {
            TR1=1;           //start timer
            P1=TL1;          //place value on pins
          }
        while(TF1==0);       //wait here
        TR1=0;               //stop timer
        TF1=0;               //clear flag
      }
  }
```

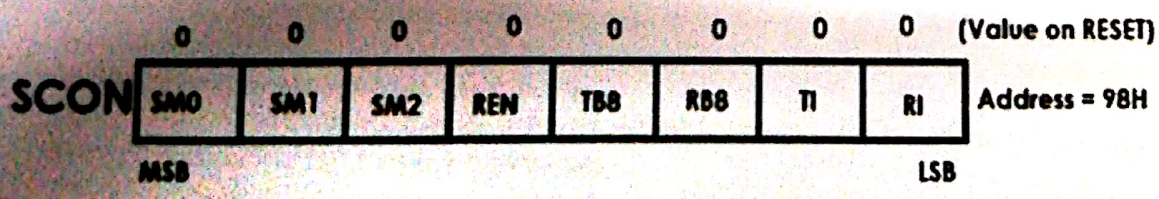
Solution 10:

```
#include <reg51.h>

void main(void)
  {
    T0=1;                    //make T0 an input
    TMOD=0x05;               //
    TL0=0;                   //set count to 0
    TH0=0;                   //set count to 0

    while(1)                 //repeat forever
      {
        do
          {
            TR0=1;           //start timer
            P1=TL0;          //place value on pins
            P2=TH0;          //
          }
        while(TF0==0);       //wait here
        TR0=0;               //stop timer
        TF0=0;
      }
  }
```

Solution 11:

The Serial Control or SCON SFR is used to control the 8051 Microcontroller's Serial Port. It is located as an address of 98H. Using SCON, you can control the Operation Modes of the Serial Port, Baud Rate of the Serial Port and Send or Receive Data using Serial Port. SCON Register also consists of bits that are automatically SET when a byte of data is transmitted or received.

The Serial Port Mode Selection Bits (SM0 and SM1) determine the mode of UART and also the baud rate. The following table gives an overview of how the Serial Port Mode Selection Bits can be used to configure Serial Port (UART) of 8051.



Solution 12:

1. The TMOD register is loaded with the value 20H, indicating the use of Timer 1 in mode 2 (8-bit auto reload) to set the baud rate
2. The TH1 is loaded with one of the values in Table 1 to set the baud rate for serial data transfer (Assuming XTAL = 11.0592 MHz)
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 (TI = 0)
6. The character byte to be transferred serially is written into the SBUF register
7. To transfer the next character, go to step 5

Solution 13:

1. The TMOD register is loaded with the value 20H, indicating the use of Timer 1 in mode 2 (8-bit auto reload) to set the baud rate
2. TH1 is loaded with one of the values in Table 1 to set the baud rate (Assuming XTAL = 11.0592 MHz)
3. The SCON register is loaded with the value 50H, indicating serial mode 1, where 8-bit data is framed with start and stop bits and receive enable is turned on.
4. TR1 is set to 1 to start Timer 1
5. RI is cleared (RI = 0)
6. The RI flag is monitored with [while(RI==0);] 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

Solution 14:

```
#include <reg51.h>
void SerTx(unsigned char);
void main (void)
{
    TMOD = 0x20;      // use Timer1, 8-bit auto reload
    TH1 = 0xFD;       // 9600 baud rate
    SCON = 0x50;
    TR1 = 1;          // start timer
    while (1)
      {
        SerTx ('Y');
        SerTx ('I');
        SerTx ('T');
      }
}
void SerTx (unsigned char x)
{
    SBUF = x;         // place value in buffer
    while (TI == 0);  // wait until transmitted
    TI = 0;
}
```

Solution 15:

```
#include <reg51.h>
void main (void)
{
    unsigned char mybyte;
    TMOD = 0x20;          // use Timer1, 8-bit auto-reload
    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;
    }
```

Solution 16:

Solution 17:

Internet Of Things is Fully Networked and Connected Devices sending analytics data back to cloud or data centre. The definition of Internet of things is that it is the network in which
every object or thing is provided unique identifier and data is transferred through a network without any verbal communication. Scope of IoT is not just limited to just connecting things to the internet, but it allows these things to communicate and exchange data, process them as well as control them while executing applications.

A dynamic global network infrastructure with self- configuring capabilities based on standard and interoperable communication protocols, where physical and virtual "things" have identities, physical attributes, and use intelligent interfaces, and are seamlessly integrated into information
network that communicate data with users and environments.
The Characteristics are:

Dynamic Global network & Self-Adapting: Adapt the changes w.r.t changing contexts

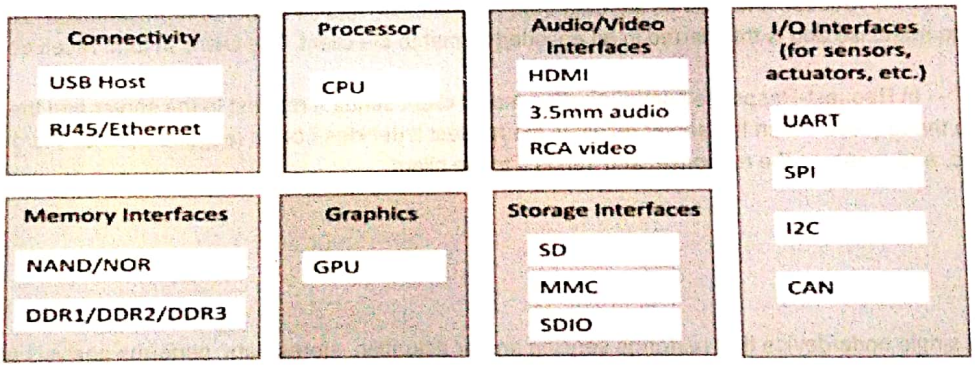
Self-Configuring: Eg. Fetching latest s/w updates without manual intervention.

Interoperable Communication Protocols: Communicate through various protocols

Unique Identity: Such as Unique IP Address or a URI

Integrated into Information Network: This allows to communicate and exchange data with other devices to perform certain analysis.

Solution 18:



Connectivity
Devices like USB hosts and ETHERNET are used for connectivity between the devices and the server.

Processor
A processor like a CPU and other units are used to process the data. these data are further used to improve the decision quality of an IoT system.

Audio/Video Interfaces
An interface like HDMI and RCA devices is used to record audio and videos in a system.

Input/Output Interface
To give input and output signals to sensors, and actuators we use things like UART, SPI, CAN, etc.

Storage Interfaces
Things like SD, MMC, and SDIO are used to store the data generated from an IoT device.
Other things like DDR and GPU are used to control the activity of an IoT system.

Solution 19:

Calculation Example
Consider a temperature sensor measuring outside temperature. People who wants to use it to see outside temperature will want to have sufficiently recent information in order for it to be of interest. In order to use Publish/Subscribe pattern to distribute this temperature, a relatively short time-interval is therefore of interest. Let's assume a 15-minute interval is sufficient. This implies N=96 values are published per day. If $n \cdot E$ is the expected or average number of values that are actually being used by users per day, where n is the number of users and E is the expected number of values that each user uses per day (they might actually only view the application once a day), we see that the number of messages distributed in the network, if only one broker is involved, is $2 \cdot N+n \cdot N$, if notification with payload is used, or $2 \cdot N+n \cdot (N+2 \cdot E)$, if notification without payload is used. However, if Request/Response is used instead, $4 \cdot n \cdot E$ messages are sent (counting messages to and from broker separately). The break-even point between Publish/Subscribe and Request/Response in these examples is therefore (if notification with payload is used) $N \leq 4 \cdot n \cdot E/(2+n) \rightarrow 4 \cdot E$ as $n \rightarrow \infty$. In our example, we would need users to use at least $E=N/4=96/4=24$ temperature values before the Publish/Subscribe pattern is more efficient than the Request/Response pattern. Such is normally the case only if all historical values are used on the receiving end.

Solution 20:

This model follows a client-server architecture.

- The client, when required, requests the information from the server. This request is usually in the encoded format.
- This model is stateless since the data between the requests is not retained and each request is independently handled.
- The server Categories the request, and fetches the data from the database and its resource representation. This data is converted to response and is transferred in an encoded format to the client. The client, in turn, receives the response.
- On the other hand — In Request-Response communication model client sends a request to the server and the server responds to the request. When the server receives the request it decides how to respond, fetches the data retrieves resources, and prepares the response, and sends it to the client.

Solution 21:

IOT Level 1-
A level-1 IoT system has a single node/device that performs sensing and/or actuation, stores data, performs analysis and hosts the application.
Level-1 IoT systems are suitable for modelling low-cost and low-complexity solutions where the data involved is not big and the analysis requirements are not computationally intensive.
e.g. home automation

IOT Level 2-
A level-2 IoT system has a single node that performs sensing and/or actuation and local analysis.
Data is stored in the cloud and the application is usually cloud-based.
Level-2 IoT systems are suitable for solutions where the data involved is big; however, the primary analysis requirement is not computationally intensive and can be done locally.
e.g. smart irrigation

IOT Level 3-
A level-3 IoT system has a single node. Data is stored and analysed in the cloud and the application is cloud-based.

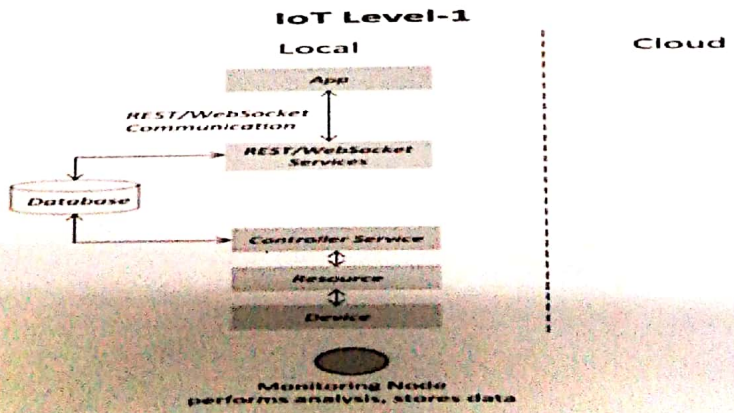
Level-3 IoT systems are suitable for solutions where the data involved is big and the analysis requirements are computationally intensive.
e.g. Tracking Package handling

IOT Level 4-
A level-4 IoT system has multiple nodes that perform local analysis. Data is stored in the cloud and the application is cloud-based.
Level-4 contains local and cloud-based observer nodes which can subscribe to and receive information collected in the cloud from IoT devices.
Level-4 IoT systems are suitable for solutions where multiple nodes are required, the data involved is big and the analysis requirements are computationally intensive.
e.g. noise monitoring

IOT Level 5-
A level-5 IoT system has multiple end nodes and one coordinator node. The end nodes perform sensing and/or actuation.
The coordinator node collects data from the end nodes and sends it to the cloud.
Data is stored and analysed in the cloud and the application is cloud-based.
Level-5 IoT systems are suitable for solutions based on wireless sensor networks, in which the data involved is big and the analysis requirements are computationally intensive.
e.g. Fire detection

IOT Level 6-
A level-6 IoT system has multiple independent end nodes that perform sensing and/or actuation and send data to the cloud.
Data is stored in the cloud and the application is cloud-based.
The analytics component analyses the data and stores the results in the cloud database.
The results are visualized with the cloud-based application.
The centralized controller is aware of the status of all the end nodes and sends control commands to the nodes.
e.g. weather monitoring system

Solution 22:



Solution 23: