

# BEE403:MICROCONTROLLER

## MODULE – 3:

8051 programming in C\_ 8051 Timer programming in Assembly and C

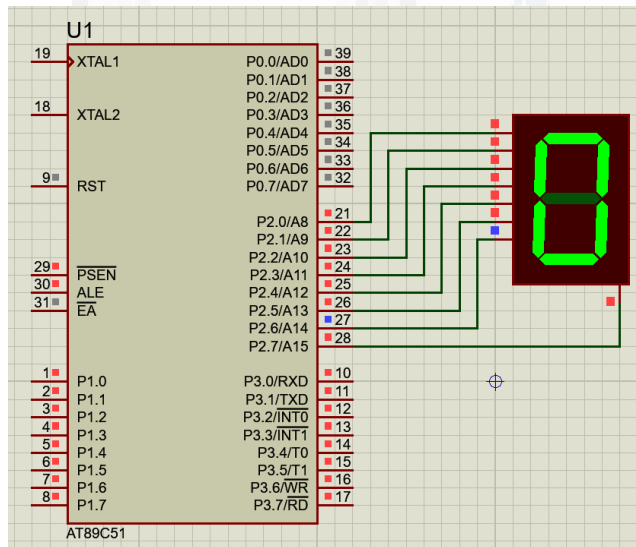


# Syllabus



- **8051 programming in C:** Data types and time delay in 8051C, IO programming in 8051C, Logic operations in 8051 C, Data conversion program in 8051 C, Accessing code ROM space in 8051C, Data serialization using 8051C
- **8051 Timer programming in Assembly and C:** Programming 8051 timers, Counter programming, Programming timers 0 and 1 in 8051 C.
- L2 – Understanding, L3 – Applying, L4 – Analyzing, L5 – Evaluating.

**CO5:** Evaluate software delays, timer delays and timer programming using both Assembly and C language. [L5, MODULE 3 ]



## 3.1 Data types in 8051C

### Why program the 8051 in C?

Compilers produce hex files that we download into the ROM of the microcontroller. The size of the hex file produced by the compiler is one of the main concerns of microcontroller programmers, for two reasons:

1. **Microcontrollers have limited on-chip ROM.**
2. **The code space for the 8051 is limited to 64K bytes.**

Following are some of the major reasons for writing programs in C instead of Assembly:

1. It is easier and less time consuming to write in C than Assembly.
2. C is easier to modify and update.
3. You can use code available in function libraries.
4. C code is portable to other microcontrollers with little or no modification

# C data types for the 8051

## 1. Unsigned char

**A.** Since the 8051 is an 8-bit microcontroller, the character data type is the most natural choice for many applications.

The unsigned char is an **8-bit data type** that takes a value in the range of **0 – 255 (00 – FFH)**.

It is one of the most widely used data types for the 8051. In many situations, such as setting a counter value.

B. Where there is **no need for signed data we should use the unsigned char instead of the signed char.**

Remember that C compilers use the signed char as the default if we do not put the keyword **unsigned in front of the char**

- We can also use the unsigned char data type for a string of ASCII characters



C. In declaring variables, we must pay careful attention to the size of the data and try to use **unsigned char instead of int if possible.**

- Because the 8051 has a limited number of registers and data RAM locations, using the int in place of the char data type can lead to a larger size hex file.



# Example 1-1

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;
}
```

Run the above program on your simulator to see how P1 displays values 00 - FFH in binary.

## Example 1-2

- Write an 8051 C program to send hex values 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[ ] = “0,1,2,3,4,5,A,B,C,D”;
    Unsigned char z;
    For(z=0;z<=10;z++)
    P1=mynum(z);
}
```

## 2. Signed char

- The signed char is an **8-bit data type** that uses the most significant bit (D7 of D7 – D0) to represent the – or + value. As a result, we have only 7 bits for the magnitude of the signed number, giving us values from **-128 to +127**.
- In situations where + and – are needed to represent a given quantity such as **temperature**, the use of the signed char data type is a must.
- **Again notice that if we do not use the keyword *unsigned*, the default is the signed value.** For that reason we should stick with the unsigned char unless the data needs to be represented as signed numbers.

## Example 1-4

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

### Solution:

```
//sign 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];
}
```

## 3. Unsigned int

- 1.The unsigned int is a 16-bit data type that takes a value in the range of 0000 to 65535 (0000 – FFFFH). In the 8051, unsigned int is used to define 16-bit variables such as memory addresses. It is also used to set counter values of more than 256.
- 2.Since the 8051 is an 8-bit microcontroller and the int data type takes two bytes of RAM, we must not use the int data type unless we have to.

3. Since registers and memory accesses are in 8-bit chunks, the misuse of int variables will result in a larger hex file. Such misuse is not a big deal in PCs with 256 megabytes of memory, 32-bit Pentium registers and memory accesses, and a bus speed of 133 MHz.

4. However, for 8051 programming do not use unsigned int in places where unsigned char will do the job. Of course the compiler will not generate an error for this misuse, but the overhead in hex file size is noticeable.



## 4. Signed int

Signed int is a 16-bit data type that uses the most significant bit (D15 of D15 – D0) to represent the – or + value.

As a result, we have only 15 bits for the magnitude of the number, or values from -32,768 to +32,767.

## 5. Sbit (single bit)

The sbit keyword is a widely used 8051 C data type designed specifically to access single-bit addressable registers. It allows access to the single bits of the SFR registers.

# Example 1-5

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

## Solution:

```
#include <reg51.h>
sbit MYBIT = P1^0;    //notice that sbit is
                      //declared outside of main

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

# Some Widely Used Data Types for 8051 C

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	-32,768 to +32,767
sbit	1-bit	SFR bit-addressable only
bit	1-bit	RAM bit-addressable only
sfr	8-bit	RAM addresses 80 - FFH only

# Time delay in C

# Time delay in C

There are two ways to create a time delay in 8051 C:

**1. Using a simple for loop**

**2. Using the 8051 timers**

In either case, when we write a time delay we must use the oscilloscope to measure the duration of our time delay.



## In creating a time delay using a for loop, we must be mindful of three factors that can affect the accuracy of the delay.

1. The 8051 design. Since the original 8051 was designed in 1980, both the fields of IC technology and microprocessor architectural design have seen great advancements. **The number of machine cycles and the number of clock periods per machine cycle vary among different versions of the 8051/52 microcontroller.**

2. While the original 8051/52 design used **12 clock periods per machine cycle**, many of the newer generations of the 8051 use fewer clocks per machine cycle.

For example, the DS5000 uses 4 clock periods per machine cycle, while the DS89C420 uses only one clock per machine cycle.

3. The crystal frequency connected to the XI – X2 input pins. The duration of the clock period for the machine cycle is a function of this crystal frequency.

**4. Compiler choice:** The third factor that affects the time delay is the compiler used to compile the C program.

When we program in Assembly language, we can control the exact **instructions and their sequences** used in the delay sub routine.

In the case of C programs, it is the C compiler that converts the C statements and functions to Assembly language instructions. As a result, different compilers produce different code.

In other words, if we compile a given 8051 C programs with different compilers, each compiler produces different hex code.

## Example 1-6

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++);
```

```
    }
```

```
}
```

## Example 1-7

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

**Solution:**



**A T M E**  
College of Engineering

```
#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 a 8051 C program to toggle all the bits of P0 and P2 continuously with a 250 ms delay.

**Solution:**

```
//This program is tested for the DS89C420 with XTAL = 11.0592 MHz
#include <reg51.h>
void MSDelay(unsigned int);
void main(void)
{
    while(1)    //another way to do it forever
    {
        P0=0x55;
        P2=0x55;
        MSDelay(250);
        P0=0xAA;
        P2=0xAA;
        MSDelay(250);
    }
}
void MSDelay(unsigned int itime)
{
    unsigned int i, j;
    for(i=0;i<itime;i++)
        for(j=0;j<1275;j++);
}
```



## 3.2 IO programming in 8051C

### Byte size I/O

ports P0 – P3 are byte-accessible. We use the P0 – P3 labels as defined in the 8051/52 C header file.

## Example 1-9

LEDs are connected to bits P1 and P2. Write an 8051 C program that shows the count from 0 to FFH (0000 0000 to 1111 1111 in binary) on the LEDs.

### Solution:

```
#include <reg51.h>
#define LED P2           //notice how we can define P2
void main(void)
{
    P1=00;               //clear P1
    LED=0;                //clear P2
    for(;;)               //repeat forever
    {
        P1++;             //increment P1
        LED++;            //increment P2
    }
}
```

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

**Solution:**



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

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 from PO. If it is less than 100, send it to P1; otherwise, send it to P2.

**Solution:**



```
#include <reg51.h>
void main(void)
{
    unsigned char mybyte;
    P0=0xFF;           //make P0 an input port
    while(1)
    {
        mybyte=P0;     //get a byte from P0
        if(mybyte<100)
            P1=mybyte;  //send it to P1 if less than 100
        else
            P2=mybyte;  //send it to P2 if more than 100
    }
}
```

# Logic operations in 8051 C

## Bit-wise operators in C

1. While every C programmer is familiar with the logical operators **AND** (&&), **OR** (||), and **NOT** (!), many C programmers are less familiar with the bitwise operators **AND** (&), **OR** (|), **EX-OR** (^), **Inverter** (~), **Shift Right** (»), and **Shift Left** («).

2. These bit-wise operators are widely used in **software engineering for embedded systems and control**; consequently, understanding and mastery of them are critical in microprocessor-based system design and interfacing.



# Table : Bit-wise Logic Operators for C

AND			OR	EX-OR	Inverter
A	B	A&B	A B	A^B	Y=~B
0	0	0	0	0	1
0	1	0	1	1	0
1	0	0	1	1	
1	1	1	1	0	

The following shows some examples using the C logical operators.

1.  $0x35 \& 0x0F = 0x05$  /\* ANDing \*/
2.  $0x04 | 0x68 = 0x6C$  /\* ORing: \*/
3.  $0x54 \wedge 0x78 = 0x2C$  /\* XORing \*/
4.  $\sim 0x55 = 0xAA$  /\* Inverting 55H \*/

# Run the following program on your simulator and examine the results

**Solution:**

```
#include <reg51.h>
```

```
void main (void)
```

```
{
```

```
P0 = 0x35 & 0x0F; //ANDing
```

```
P1 = 0x04 | 0x68; //ORing
```

```
P2= 0x54 * 0x78; //XORing
```

```
P0= ~0x55; //inversing
```

```
P1= 0x9A » 3; //shifting right 3 times
```

```
P2= 0x77 » 4; //shifting right 4 times
```

```
P0= 0x6 « 4; //shifting left 4 times
```

```
}
```

0	1	0	1	0	1	0	0
0	1	1	1	1	0	0	0
0	0	1	0	1	1	0	0

1	1	0	0	1	0	0	0
0	1	1	0	0	1	0	0

**Write an 8051 C program to toggle all the bits of P0 and P2 continuously with a 250 ms delay. Use the **inverting operator**.**

**Solution:**

**The program below is tested for the DS89C420 with XTAL = 11.0592 MHz.**

```
#include <reg51.h>
void MSDelay(unsigned int);
void main(void)
{
    P0=0x55;
    P2=0x55;
    while(1)
    {
        P0=~P0;
        P2=~P2;
        MSDelay(250);
    }
}

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

# Bit-wise shift operation in C

1. There are two bit-wise shift operators in C:

(1) **shift right** (  $\gg$  ), and (2) **shift left** (  $\ll$  ).

Their format in C is as follows:

2. data  $\gg$  number of bits to be shifted right

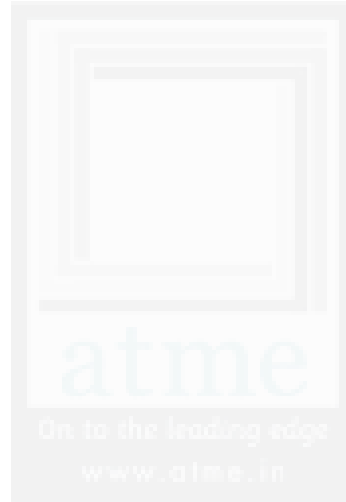
3. data  $\ll$  number of bits to be shifted left

The following shows some examples of shift operators in C.

- |                        |   |
|------------------------|---|
| 1. $0x9A \gg 3 = 0x13$ | <code>/* shifting right 3 times */</code> |
| 2. $0x77 \gg 4 = 0x07$ | <code>/* shifting right 4 times */</code> |
| 3. $0x6 \ll 4 = 0x60$  | <code>/* shifting left 4 times */</code>  |

Write an 8051 C program to toggle all the bits of P0, P1, and P2 continuously with a 250 ms delay. Use the Ex-OR operator.

**Solution:**



**A T M E**  
College of Engineering

The program below is tested for the DS89C420 with XTAL = 11.0592 MHz.

```
#include <reg51.h>
void MSDelay(unsigned int);
void main(void)
{
    P0=0x55;
    P1=0x55;
    P2=0x55;
    while(1)
    {
        P0=P0^0xFF;
        P1=P1^0xFF;
        P2=P2^0xFF;
        MSDelay(250);
    }
}

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

E  
ing



**Write an 8051 C program to get bit P1.0 and send it to P2.7 after inverting it.**

```
#include <reg51.h>
sbit inbit=P1^0;
sbit outbit=P2^7;    //sbit is used declare port (SFR) bits
bit membit;          //notice this is bit-addressable memory
void main(void)
{
    while(1)
    {
        membit=inbit;    //get a bit from P1.0
        outbit=~membit;  //invert it and send it to P2.7
    }
}
```



**Write an 8051 C program to read the P1.0 and P1.1 bits and issue an ASCII character to P0 according to the following table.**

P1.1	P1.0	
0	0	send '0' to P0
0	1	send '1' to P0
1	0	send '2' to P0
1	1	send '3' to P0

**Solution:**

```
#include <reg51.h>
void main(void)
{
    unsigned char z;
    z=P1;
    z=z&0x3;
    switch(z)
    {
        case(0):
        {
            P0='0';
            break;
        }
        //read P1
        //mask the unused bits
        //make decision
        //issue ASCII 0
    }
```

```
case (1) :  
{  
    P0='1';           //issue ASCII 1  
    break;  
}  
case (2) :  
{  
    P0='2';           //issue ASCII 2  
    break;  
}  
case (3) :  
{  
    P0='3';           //issue ASCII 3  
    break;  
}  
}  
}
```

# Data conversion program in 8051 C

# Data conversion program in 8051 C

## ASCII Code for Digits 0 – 9

Key	ASCII (hex)	Binary	BCD (unpacked)
0	30	011 0000	0000 0000
1	31	011 0001	0000 0001
2	32	011 0010	0000 0010
3	33	011 0011	0000 0011
4	34	011 0100	0000 0100
5	35	011 0101	0000 0101
6	36	011 0110	0000 0110
7	37	011 0111	0000 0111
8	38	011 1000	0000 1000
9	39	011 1001	0000 1001

# Packed BCD to ASCII conversion

Packed BCD	Unpacked BCD	ASCII
0x29	0x02, 0x09	0x32, 0x39
00101001	00000010, 00001001	00110010, 00111001

## ASCII to packed BCD conversion

Key	ASCII	Unpacked BCD	Packed BCD
4	34	00000100	
7	37	00000111	01000111 or 47H

Write an 8051 C program to convert packed BCD 0x29 to ASCII and display the bytes on P1 and P2.

### Solution:

```
#include <reg51.h>
void main(void)
{
    unsigned char x, y, z;
    unsigned char mybyte = 0x29;
    x = mybyte & 0x0F;           //mask lower 4 bits
    P1 = x | 0x30;               //make it ASCII
    y = mybyte & 0xF0;           //mask upper 4 bits
    y = y >> 4;                  //shift it to lower 4 bits
    P2 = y | 0x30;               //make it ASCII
}
```

0	0	1	0	1	0	0	1
0	0	0	0	1	1	1	1
0	0	0	0	1	0	0	1
0	0	1	1	0	0	0	0
0	0	1	1	1	0	0	1

0	0	1	0	1	0	0	1
1	1	1	1	0	0	0	0
0	0	1	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	0	0	1	0	0	0
0	0	0	0	0	1	0	0
0	0	0	0	0	0	1	0
0	0	1	1	0	0	0	0
0	0	1	1	0	0	1	0

Engineering

Write an 8051 C program to convert ASCII digits of '4' and '7' to packed BCD and display them on P1.

### Solution:

```
#include <reg51.h>
void main(void)
{
    unsigned char bcdbyte;
    unsigned char w='4';
    unsigned char z='7';
    w = w & 0x0F;      //mask 3
    w = w << 4;        //shift left to make upper BCD digit
    z = z & 0x0F;      //mask 3
    bcdbyte = w | z;   //combine to make packed BCD
    P1 = bcdbyte;
}
```



# Checksum byte in ROM

1. To ensure the integrity of ROM contents, every system must perform the **checksum calculation**.
2. The process of checksum will **detect any corruption of the contents of ROM**. One of the causes of ROM corruption is current surge, either when the system is turned on or during operation.
3. To ensure data integrity in ROM, the checksum process uses what is called a ***checksum byte***.

4. The **checksum byte is an extra byte** that is tagged to the end of a series of bytes of data. To calculate the checksum byte of a series of bytes of data, the following steps can be taken.

1. Add the bytes together and drop the carries.
2. Take the 2's complement of the total sum. This is the checksum byte, which becomes the last byte of the series.
3. To perform the checksum operation, add all the bytes, including the checksum byte. The result must be zero. If it is not zero, one or more bytes of data have been changed (corrupted).

(a) Find the checksum byte.

$$\begin{array}{r} 25H \\ + 62H \\ + 3FH \\ + \underline{52H} \end{array}$$

118H (Dropping carry of 1 and taking the 2's complement, we get E8H.)

(b) Perform the checksum operation to ensure data integrity.

$$\begin{array}{r} 25H \\ + 62H \\ + 3FH \\ + 52H \\ + \underline{E8H} \end{array}$$

200H (Dropping the carries we get 00, which means data is not corrupted.)

(c) If the second byte 62H has been changed to 22H, show how checksum detects the error.

$$\begin{array}{r} 25H \\ + 22H \\ + 3FH \\ + 52H \\ + \underline{E8H} \end{array}$$

1C0H (Dropping the carry, we get C0H, which means data is corrupted.)

Write an 8051 C program to calculate the checksum byte for the data given

```
#include <reg51.h>
void main(void)
{
    unsigned char mydata[] = {0x25,0x62,0x3F,0x52};
    unsigned char sum=0;
    unsigned char x;
    unsigned char checksumbyte;
    for(x=0;x<4;x++)
    {
        P2=mydata[x];           //issue each byte to P2
        sum=sum+mydata[x];      //add them together
        P1=sum;                 //issue the sum to P1
    }
    checksumbyte=~sum+1;        //make 2's complement
    P1=checksumbyte;            //show the checksum byte
}
```

# Binary (hex) to decimal and ASCII conversion in 8051 C

- Write an 8051 C program to convert 11111101 (FD hex) to decimal and display the digits on P0, P1, and P2.

```
#include <reg51.h>
void main(void)
{
    unsigned char x, binbyte, d1, d2, d3;
    binbyte = 0xFD;           //binary(hex) byte
    x = binbyte / 10;         //divide by 10
    d1 = binbyte % 10;        //find remainder (LSD)
    d2 = x % 10;              //middle digit
    d3 = x / 10;              //most significant digit (MSD)
    P0 = d1;
    P1 = d2;
    P2 = d3;
}
```

www.atme.in

	<u>Quotient</u>	<u>Remainder</u>
FD/0A	19	3 (low digit) LSD
19/0A	2	5 (middle digit)
		2 (high digit) (MSD)

E  
eering



# Accessing code ROM space in 8051C

## RAM data space v. code data space

In the 8051 we have three spaces in which to store data. They are as follows:

- 1.The 128 bytes of RAM space with address range 00 – 7FH. (In the 8052, it is 256 bytes.) We can read (from) or write (into) this RAM space directly or indirectly using the R0 and R1 registers.
- 2.The 64K bytes of code (program) space with addresses of 0000 – FFFFH. This 64K bytes of on-chip ROM space is used for storing programs (opcodes) and therefore is directly under the control of the program counter (PC).

# There are two problems with using this code space for data.

- a) First, since it is ROM memory, we can burn our predefined data and tables into it. But we cannot write into it during the execution of the program.
- b) The second problem is that the more of this code space we use for data, the less is left for our program code.

For example, if we have an 8051 chip such as DS89C420 with only 16K bytes of on-chip ROM, and we use 4K bytes of it to store some look-up table, only 12K bytes is left for the code program. For some applications this can be a problem. For this reason Intel created another memory space called *external memory* especially for data.



Compile and single-step the following program on your 8051 simulator. Examine the contents of the 128-byte RAM space to locate the ASCII values.

**Solution:**

```
#include <reg51.h>
void main(void)
{
    unsigned char mynum[] = "ABCDEF"; //This uses RAM space
                                        //to store data

    unsigned char z;
    for(z=0; z<=6; z++)
        P1=mynum [z];
}
```

Compare and contrast the following programs and discuss the advantages and disadvantages of each one.

(a)

```
#include <reg51.h>
void main(void)
{
    P1='H';
    P1='E';
    P1='L';
    P1='L';
    P1='O';
}
```

(b)

```
#include <reg51.h>
void main(void)
{
    unsigned char mydata[]="HELLO";
    unsigned char z;
    for(z=0;z<=5;z++)
        P1=mydata[z];
}
```

(c)

```
#include <reg51.h>
void main(void)
{
    //Notice Keyword code
    code unsigned char mydata[]="HELLO";
    unsigned char z;
    for(z=0;z<=5;z++)
        P1=mydata[z];
}
```

- The first one is short and simple, but the individual characters are embedded into the program. If we change the characters, the whole program changes. It also mixes the code and data together.
- The second one uses the RAM data space to store array elements, therefore the size of the array is limited.
- The third one uses a separate area of the code space for data. This allows the size of the array to be as long as you want if you have the on-chip ROM.

# Data serialization using 8051C

1. **Using the serial port.** When using the serial port, the programmer has very limited control over the sequence of data transfer.
2. **The second method of serializing data is to transfer data one bit a time and control the sequence of data** and spaces in between them.

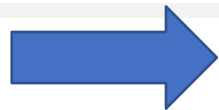
In many new generations of devices such as LCD, ADC, and ROM the serial versions are becoming popular since they take less space on a printed circuit board.

Write a C program to send out the value 44H serially one bit at a time via P1.0. The LSB should go out first

### Solution:

```
//SERIALIZING DATA VIA P1.0 (SHIFTING RIGHT)
#include <reg51.h>
sbit P1b0 = P1^0;
sbit regALSB = ACC^0;
void main(void)
{
    unsigned char conbyte = 0x44;
    unsigned char x;
    ACC = conbyte;
    for(x=0; x<8; x++)
    {
        P1b0 = regALSB;
        ACC = ACC >> 1;
    }
}
```

Conbyte=0x44



RegA= Accumulator



ACC.0



P1.0



Write a C program to send out the value 44H serially one bit at a time via P1.O. The MSB should go out first.



## Solution:

//SERIALIZING DATA VIA P1.0 (SHIFTING LEFT)

```
#include <reg51.h>
```

```
sbit P1b0 = P1^0;
```

```
sbit regAMSB = ACC^7;
```

```
void main(void)
```

```
{
```

```
    unsigned char conbyte = 0x44;
```

```
    unsigned char x;
```

```
    ACC = conbyte;
```

```
    for(x=0; x<8; x++)
```

```
    {
```

```
        P1b0 = regAMSB;
```

```
        ACC = ACC << 1;
```

```
    }
```

```
}
```

ACC.7 → P1.0

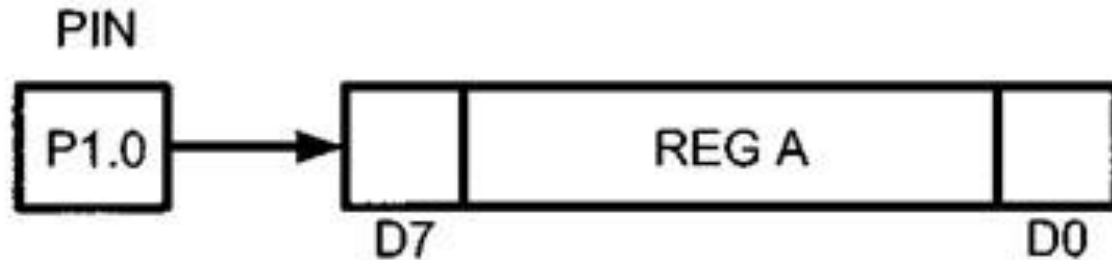


Write a C program to bring in a byte of data serially one bit at a time via P1 .0. The LSB should come in first.

## Solution:

//BRINGING IN DATA VIA P1.0 (SHIFTING RIGHT)

```
#include <reg51.h>
sbit P1b0 = P1^0;
sbit ACCMSB = ACC^7;
void main(void)
{
    unsigned char conbyte = 0x44;
    unsigned char x;
    for(x=0; x<8; x++)
    {
        ACCMSB = P1b0;
        ACC = ACC >> 1;
    }
    P2=ACC;
}
```



# Programming 8051 timers

1. The 8051 has two timers: Timer 0 and Timer 1. They can be used either as **timers or as event counters**.

## 2. Basic registers of the timer

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





# TMOD (timer mode) register

1. Both timers 0 and 1 use the same register, called TMOD, to set the various timer operation modes.
2. TMOD is an 8-bit register in which the lower 4 bits are set aside for Timer 0 and the upper 4 bits for Timer 1.
3. In each case, the lower 2 bits are used to set the timer mode and the upper 2 bits to specify the operation.



(MSB)

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

(LSB)

**GATE** Gating control when set. The timer/counter is enabled 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.

**C/T** Timer or counter selected cleared for timer operation (input from internal system clock). Set for counter operation (input from Tx input pin).

**M1** Mode bit 1

**M0** Mode bit 0

<u>M1</u>	<u>M0</u>	<u>Mode</u>	<u>Operating Mode</u>
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/counters 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 that is to be reloaded into TLx each time it overflows.
1	1	3	Split timer mode

# Example 1-37

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:

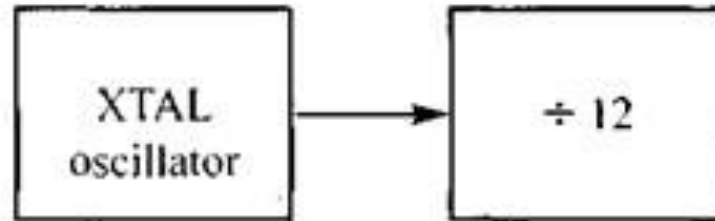
We convert the values from hex to binary.

1. TMOD = 00000001, mode 1 of Timer 0 *is* selected.
2. TMOD = 00100000, mode 2 of Timer 1 is selected.
3. TMOD = 00010010, mode 2 of Timer 0, and mode 1 of Timer 1 are selected.

Find the timer's clock frequency and its period for various 8051-based systems, with the following crystal frequencies.

- (a) 12 MHz
- (b) 16 MHz
- (c) 11.0592 MHz

**Solution:**



(a)  $1/12 \times 12 \text{ MHz} = 1 \text{ MHz}$  and  $T = 1/1 \text{ MHz} = 1 \mu\text{s}$

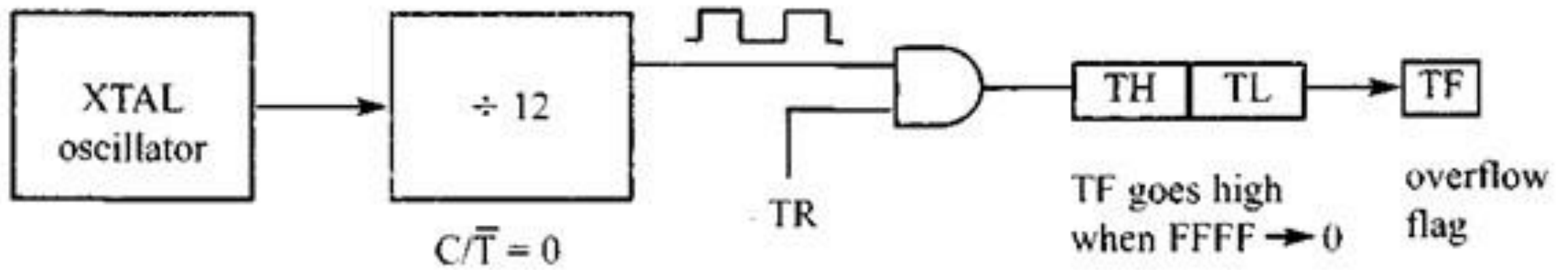
(b)  $1/12 \times 16 \text{ MHz} = 1.333 \text{ MHz}$  and  $T = 1/1.333 \text{ MHz} = .75 \mu\text{s}$

(c)  $1/12 \times 11.0592 \text{ MHz} = 921.6 \text{ kHz};$   
 $T = 1/921.6 \text{ kHz} = 1.085 \mu\text{s}$

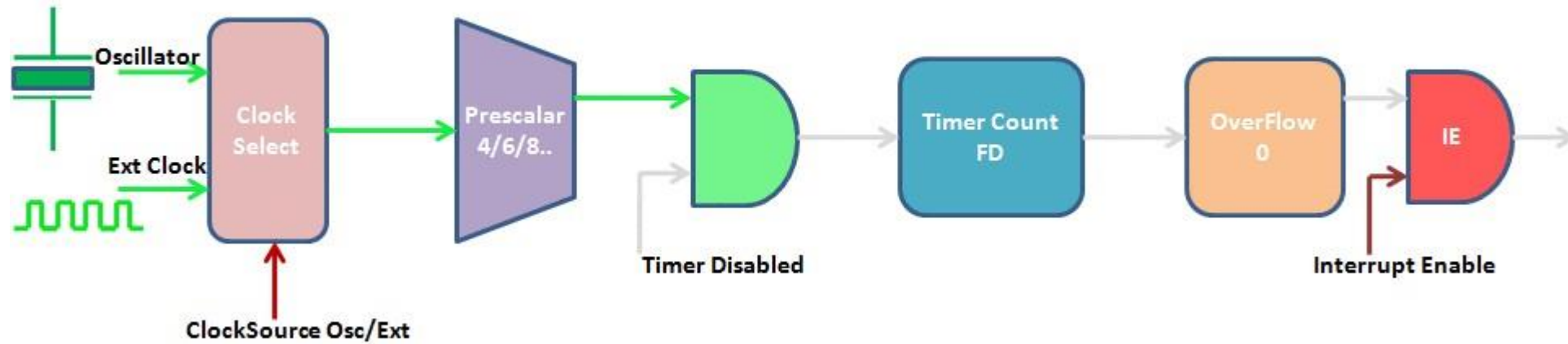
# Mode 1 programming

The following are the characteristics and operations of mode 1:

1. It is a 16-bit timer; therefore, it allows values of **0000 to FFFFH to be loaded into the timer's registers TH and TL.**
2. After TH and TL are loaded with a 16-bit initial value, the timer must be started. This is done by **“SETB TRO”** for Timer 0 and “SETB TR1” for Timer 1.
3. After the timer is started, it starts to count up. It counts up until it reaches its limit of FFFFH. When it rolls over from FFFFH to 0000, it sets high a flag bit called TF (timer flag). This timer flag can be monitored. When this timer flag is raised, one option would be to stop the timer with the instructions “CLR TRO” or “CLR TR1”, for Timer 0 and Timer 1, respectively. Again, it must be noted that each timer has its own timer flag: TFO for Timer 0, and TF1 for Timer 1.
4. After the timer reaches its limit and rolls over, in order to repeat the process the registers TH and TL must be reloaded with the original value, and TF must be reset to 0.



Timer Block Diagram



ExploreEmbedded

# Steps to program in mode 1

To generate a time delay, using the timer's mode 1, the following steps are taken.

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 values.
3. Start the timer.
4. Keep monitoring the timer flag (TF) with the “JNB TFX, target” instruction to see if it is raised. Get out of the loop when TF becomes high.
5. Stop the timer.
6. Clear the TF flag for the next round.
7. Go back to Step 2 to load TH and TL again.



To calculate the exact time delay and the square wave frequency generated on pin P1 .5, we need to know the XTAL frequency.

**(a) in hex**

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

**(b) in decimal**

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

**Timer Delay Calculation for XTAL = 11.0592 MHz**



## Example

In the following program, we are creating 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               ;load TH, TL again

;-----delay using Timer 0
DELAY:
                SETB   TR0                ;start Timer 0
AGAIN:         JNB     TF0,AGAIN          ;monitor Timer 0 flag until
                                                ;it rolls over
                CLR     TR0               ;stop Timer 0
                CLR     TF0               ;clear Timer 0 flag
                RET
```

## **Solution:**

In the above program notice the following steps.

1. TMOD is loaded.
2. FFF2H is loaded into TH0 – TLO.
3. P1.5 is toggled for the high and low portions of the pulse.
4. The DELAY subroutine using the timer is called.
5. In the DELAY subroutine, Timer 0 is started by the “SETB TRO” instruction.

# Programs

**Calculate the amount of time delay in the DELAY subroutine generated by the timer. Assume that XTAL = 11.0592 MHz.**

**Solution:**

1. The timer works with a clock frequency of  $1/12$  of the XTAL frequency; therefore, we have  $11.0592 \text{ MHz} / 12 = 921.6 \text{ kHz}$  as the timer frequency.
2. As a result, each clock has a period of  $T = 1 / 921.6 \text{ kHz} = 1.085 \text{ us}$  (is. In other words, Timer 0 counts up each 1.085 us resulting in delay = number of counts x 1.085 us)

1. The number of counts for the rollover is  $\text{FFFFH} - \text{FFF2H} = \text{ODH}$  (13 decimal).
2. However, we add one to 13 because of the extra clock needed when it rolls over from FFFF to 0 and raises the TF flag.
3. This gives  $14 \times 1.085 \text{ us} = 15.19 \text{ }\mu\text{s}$  for half the pulse.
4. For the entire period  $T = 2 \times 15.19$  (as  $= 30.38 \text{ }\mu\text{s}$  gives us the time delay generated by the timer.

Find the delay generated by Timer 0 in the following code, using both of the methods. Do not include the overhead due to instructions.

	CLR	P2.3	;clear P2.3
	MOV	TMOD,#01	;Timer 0, mode 1(16-bit mode)
HERE:	MOV	TL0,#3EH	;TL0 = 3EH, Low byte
	MOV	TH0,#0B8H	;TH0 = B8H, High byte
	SETB	P2.3	;SET high P2.3
	SETB	TR0	;start Timer 0
AGAIN:	JNB	TF0,AGAIN	;monitor Timer 0 flag
	CLR	TR0	;stop Timer 0
	CLR	TF0	;clear Timer 0 flag for
			;next round
	CLR	P2.3	

**Solution:**



1.  $(\text{FFFF-B83E} + 1) = 47\text{C2H} = 18370$  in decimal and  $18370 \times 1.085 \mu\text{s} = 19.93145\text{ms}$ .

2. Since  $\text{TH} - \text{TL} = \text{B83EH} = 47166$  (in decimal) we have  $65536 - 47166 = 18370$ . This means that the timer counts from B83EH to FFFFH.. This plus rolling over to 0 goes through a total of 18370 clock cycles, where each clock is  $1.085\mu\text{s}$  in duration. Therefore, we have  $18370 \times 1.085 \mu\text{s} = 19.93145 \text{ ms}$  as the width of the pulse.

- The following program generates a square wave on pin 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.

```
AGAIN:    MOV    TMOD, #10H        ;Timer 1, mode 1(16-bit)
          MOV    TL1, #34H         ;TL1 = 34H, Low byte
          MOV    TH1, #76H         ;TH1 = 76H, High byte
                                   ;(7634H = timer value)

          SETB   TR1               ;start Timer 1

BACK:     JNB    TF1, BACK          ;stay until timer rolls over
          CLR    TR1               ;stop Timer 1
          CPL    P1.5              ;comp. P1.5 to get hi, lo
          CLR    TF1               ;clear Timer 1 flag
          SJMP   AGAIN             ;reload timer since Mode 1
                                   ;is not auto-reload
```

**Solution:**

1. In the above program notice the target of SJMP. In mode 1, the program must reload the TH, TL register every time if we want to have a continuous wave.
2. Now the calculation. Since  $FFFFH - 7634H = 89CBH + 1 = 89CCH$  and  $89CCH = 35276$  clock count.  **$35276 \times 1.085 \mu s = 38.274 ms$**  for half of the square wave.
3. The entire square wave length is  $38.274 \times 2 = 76.548 ms$  and has a frequency = **13.064 Hz**.

Assume that XTAL = 11.0592 MHz. What value do we need to load into the timer's registers 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:**

1. Since XTAL = 11.0592 MHz, the counter counts up every 1.085 us.
2. This means that out of many 1.085 us intervals we must make a 5 ms pulse.
3. To get that, we divide one by the other. We need  $5 \text{ ms} / 1.085 \text{ us} = 4608$  clocks.
4. To achieve that we need to load into TL and TH the value  $65536 - 4608 = 60928 = \text{EEOOH}$ . Therefore, we have TH = EE and tt = 00

```
                CLR    P2.3                ;clear P2.3
                MOV    TMOD,#01            ;Timer 0, mode 1 (16-bit mode)
HERE:           MOV    TL0,#0              ;TL0 = 0, Low byte
                MOV    TH0,#0EEH           ;TH0 = EE( hex), High byte
                SETB   P2.3                ;SET P2.3 high
                SETB   TR0                 ;start Timer 0
AGAIN:          JNB    TF0,AGAIN            ;monitor Timer 0 flag
                                                ;until it rolls over

                CLR    P2.3                ;clear P2.3
                CLR    TR0                 ;stop Timer 0
                CLR    TF0                 ;clear Timer 0 flag
```

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

### Solution:

Look at the following steps.

1.  $T = 1 / f = 1 / 2 \text{ kHz} = 500 \mu\text{s}$  the period of the square wave.
2. 1/2 of it for the high and low portions of the pulse is 250  $\mu\text{s}$ .
3.  $250 \text{ us} / 1.085 \text{ us} = 230$  and  $65536 - 230 = 65306$ . which in hex is FF1AH.
4. TL = 1AH and TH = FFH. all in hex.



```
                MOV    TMOD,#10H        ;Timer 1, mode 1(16-bit)
AGAIN:          MOV    TL1,#1AH         ;TL1=1AH, Low byte
                MOV    TH1,#0FFH        ;TH1=FFH, High byte
                SETB   TR1               ;start Timer 1
BACK:           JNB    TF1,BACK          ;stay until timer rolls over
                CLR    TR1              ;stop Timer 1
                CPL    P1.5              ;complement P1.5 to get hi, lo
                CLR    TF1               ;clear Timer 1 flag
                SJMP   AGAIN              ;reload timer since mode 1
                ;is not auto-reload
```



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

### **Solution:**

Look at the following steps.

1.  $T = 1 / 50 \text{ Hz} = 20 \text{ ms}$ , the period of the square wave.
2.  $1/2$  of it for the high and low portions of the pulse = 10 ms
3.  $10 \text{ ms} / 1.085 \text{ us} = 9216$  and  $65536 - 9216 = 56320$  in decimal, and in hex it is DC00H.
4. TL = 00 and TH = DC (hex)

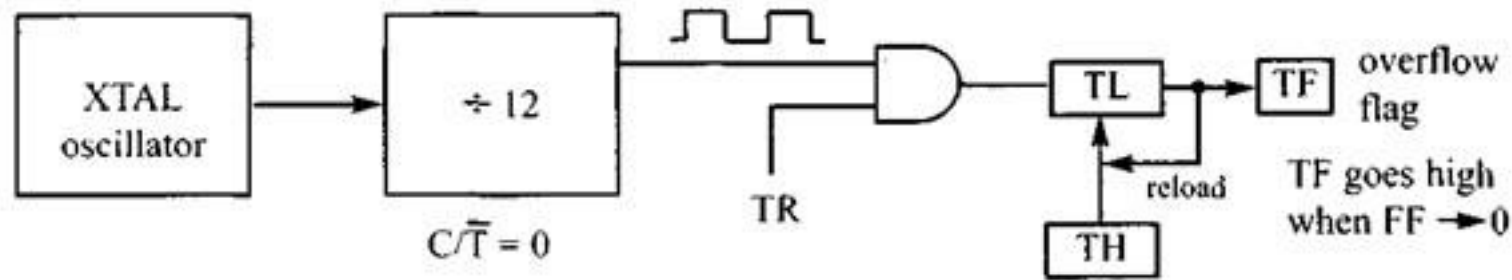
Assuming XTAL = 11.0592 MHz, write a program to generate a square wave of 5kHz frequency on pin P2.3.

- 1. Solve steps**
- 2. Write the program**

The program follows.

	MOV	TMOD, #10H	;Timer 1, mode 1 (16-bit)
AGAIN:	MOV	TL1, #00	;TL1 = 00, Low byte
	MOV	TH1, #0DCH	;TH1 = DCH, High byte
	SETB	TR1	;start Timer 1
BACK:	JNB	TF1, BACK	;stay until timer rolls over
	CLR	TR1	;stop Timer 1
	CPL	P2.3	;comp. P2.3 to get hi, lo
	CLR	TF1	;clear Timer 1 flag
	SJMP	AGAIN	;reload timer since mode 1 ;is not auto-reload

# Mode 2 programming



## Steps to program in mode 2

To generate a time delay using the timer's mode 2, take the following steps.

1. Load the TMOD value register indicating which timer (Timer 0 or Timer 1) is to be used, and select the timer mode (mode 2).
2. Load the TH registers with the initial count value.
3. Start the 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 Step 4, since mode 2 is auto-reload.

Assuming that XTAL = 11.0592 MHz. find (a) the frequency of the square wave generated on pin P 1.0 in the following program, and (b) the smallest frequency achievable in this program, and the TH value to do that.

```
                MOV    TMOD, #20H           ;T1/mode 2/8-bit/auto-reload
                MOV    TH1, #5              ;TH1 = 5
                SETB   TR1                  ;start Timer 1
BACK:           JNB    TF1, BACK             ;stay until timer rolls over
                CPL    P1.0                ;comp. P1.0 to get 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  $(256 - 05) \times 1.085 \mu\text{s} = 251 \times 1.085 \mu\text{s} = 272.33 \mu\text{s}$  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  $T = 2 \times 272.33 \mu\text{s} = 544.67 \mu\text{s}$  and the frequency = 1.83597 kHz.

1.To get the smallest frequency, we need the largest T and that is achieved when TH = 00.

2.In that case, we have  $T = 2 \times 256 \times 1.085 \mu\text{s} = 555.52 \mu\text{s}$  and the frequency = 1.8kHz.



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

**Solution:**

```
                MOV    TMOD,#2H           ;Timer 0, mode 2
                                           ;(8-bit, auto-reload)
                MOV    TH0,#0             ;TH0=0
AGAIN:          MOV    R5,#250            ;count for multiple delay
                ACALL  DELAY
                CPL     P1.0              ;toggle P1.0
                SJMP   AGAIN              ;repeat
DELAY:          SETB   TR0                ;start Timer 0
BACK:           JNB    TF0,BACK            ;stay until timer rolls over
                CLR    TR0                ;stop Timer 0
                CLR    TF0                ;clear TF for next round
                DJNZ   R5,DELAY
                RET
```

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



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

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

**Solution:**

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

# Counter programming

## C/T bit in TMOD register

1. Recall from the last section that the C/T bit in the TMOD register decides the **source of the clock for the timer.**

If **C/T = 0**, the timer gets pulses from the crystal.

2. In contrast, when  $C/T = 1$ , the timer is used as a counter and gets its pulses from outside the 8051.

Therefore, when **C/T = 1**, the counter counts up as pulses are fed from pins 14 and 15.

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

(MSB)

(LSB)

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



0 0 MODE 0 13 BIT TH TL

0 1 MODE 1 16 BIT TH TL

1 0 **MODE 2** 8 BIT AR **TH0= -60**

1 1 MODE 3 8 BIT TH0

GATE	C/T	M1	MO	GATE	C/T	M1	MO
0	0	0	0	0	1	1	0

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**.

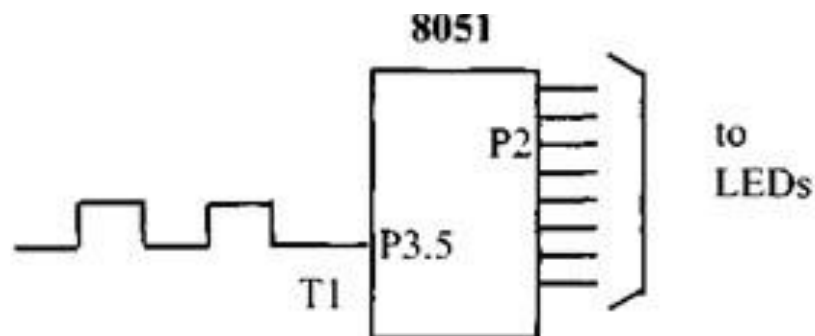
```

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 count TL1
        MOV    P2,A         ;display it on port 2
        JNB    TF1,BACK     ;keep doing it if TF=0
        CLR    TR1          ;stop the counter 1
        CLR    TF1          ;make TF=0
        SJMP   AGAIN        ;keep doing it

```

Notice in the above program the role of the instruction “SETB P3.5”. Since ports are set up for output when the 8051 is powered up, we make P3.5 an input port by making it high. In other words, we must configure (set high) the T1 pin (pin P3.5) to allow pulses to be fed into it.

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



Assume that a 1-Hz frequency pulse is connected to **input pin 3.4**.

Write a program to display counter 0 on an LCD. Set the initial value of **TH0 to -60**

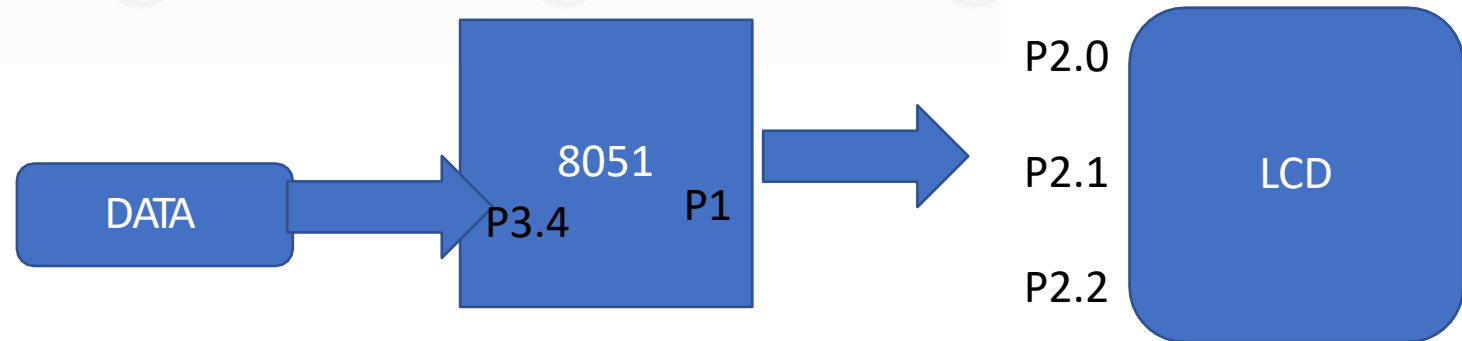
### **Solution:**

To display the TL count on an LCD, we must convert 8-bit binary data to ASCII



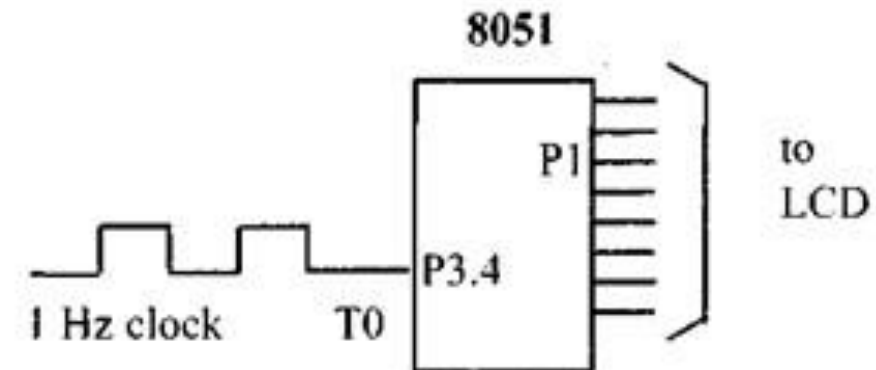
AGAIN:  
BACK:

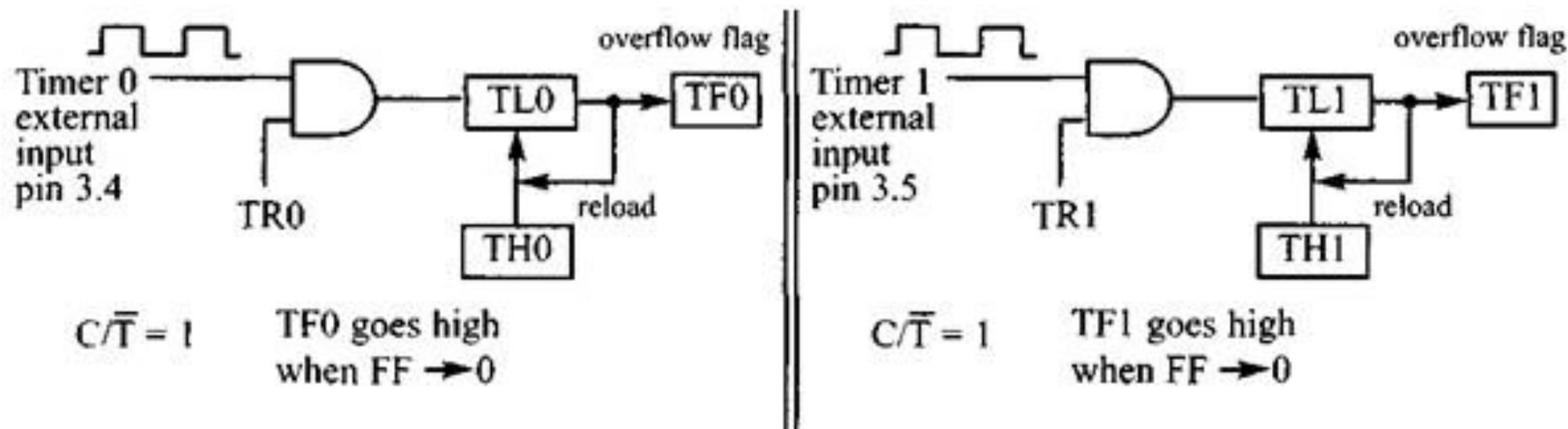
```
MOV    TMOD, #00000110B ;counter 0, mode 2, C/T=1
MOV    TH0, #-60         ;counting 60 pulses
SETB   P3.4              ;make T0 as input
SETB   TR0               ;starts the counter
MOV    A, TL0             ;get copy of count TL0
ACALL  CONV              ;convert in R2, R3, R4
ACALL  DISPLAY           ;display on LCD
JNB    TF0, BACK         ;loop if TF0=0
CLR    TR0               ;stop the counter 0
CLR    TF0               ;make TF0=0
SJMP   AGAIN             ;keep doing it
```



```
CONV:      MOV    B,#10           ;divide by 10
           DIV    AB
           MOV    R2,B           ;save low digit
           MOV    B,#10           ;divide by 10 once more
           DIV    AB
           ORL    A,#30H          ;make it ASCII
           MOV    R4,A           ;save MSD
           MOV    A,B
           ORL    A,#30H          ;make 2nd digit an ASCII
           MOV    R3,A           ;save it
           MOV    A,R2
           ORL    A,#30H          ;make 3rd digit an ASCII
           MOV    R2,A           ;save the ASCII
           RET
```

[www.atme.in](http://www.atme.in)





# Equivalent Instructions for the Timer Control Register (TCON)

## For Timer 0

SETB TR0	=	SETB TCON.4
CLR TR0	=	CLR TCON.4
SETB TF0	=	SETB TCON.5
CLR TF0	=	CLR TCON.5

## For Timer 1

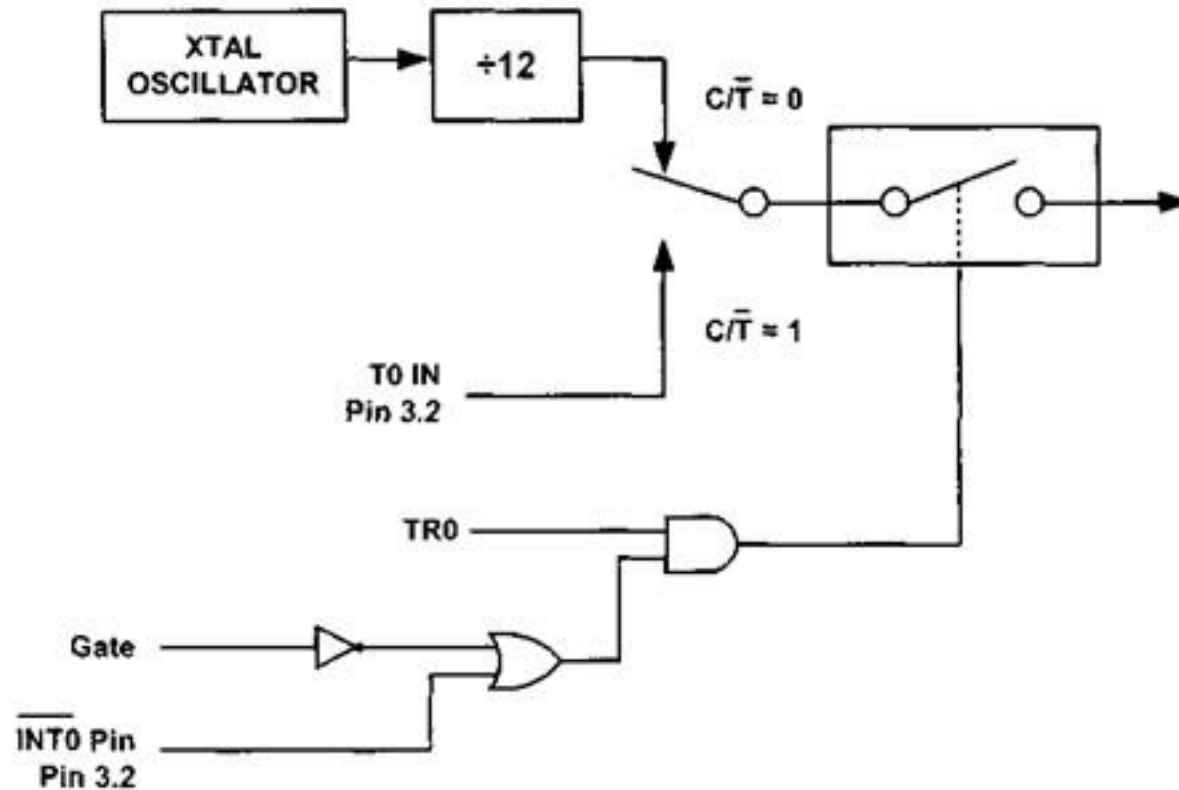
SETB TR1	=	SETB TCON.6
CLR TR1	=	CLR TCON.6
SETB TF1	=	SETB TCON.7
CLR TF1	=	CLR TCON.7

TCON: Timer/Counter Control Register

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0
-----	-----	-----	-----	-----	-----	-----	-----

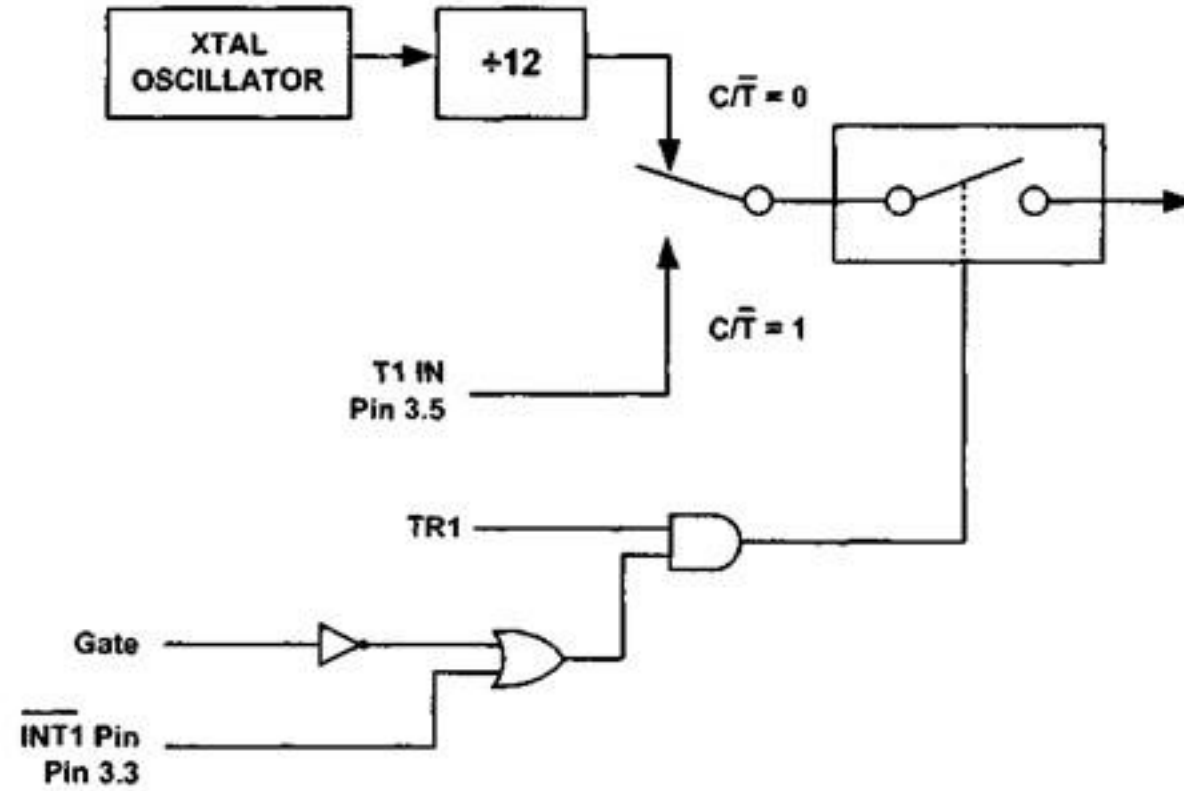
# TCON register

The case of GATE = 1 in TMOD



Timer/Counter 0

1. All discussion so far has assumed that  $GATE = 0$ . When  $GATE = 0$ , the timer is started with instructions “SETB TRO” and “SETB TR1”, for Timers 0 and 1, respectively.
2. What happens if the GATE bit in TMOD is set to 1. if  $GATE = 1$ , the start and stop of the timer are done externally through pins P3.2 and P3.3 for Timers 0 and 1, respectively.



**Timer/Counter 1**



# Programming timers 0 and 1 in 8051 C

## Accessing timer registers in C

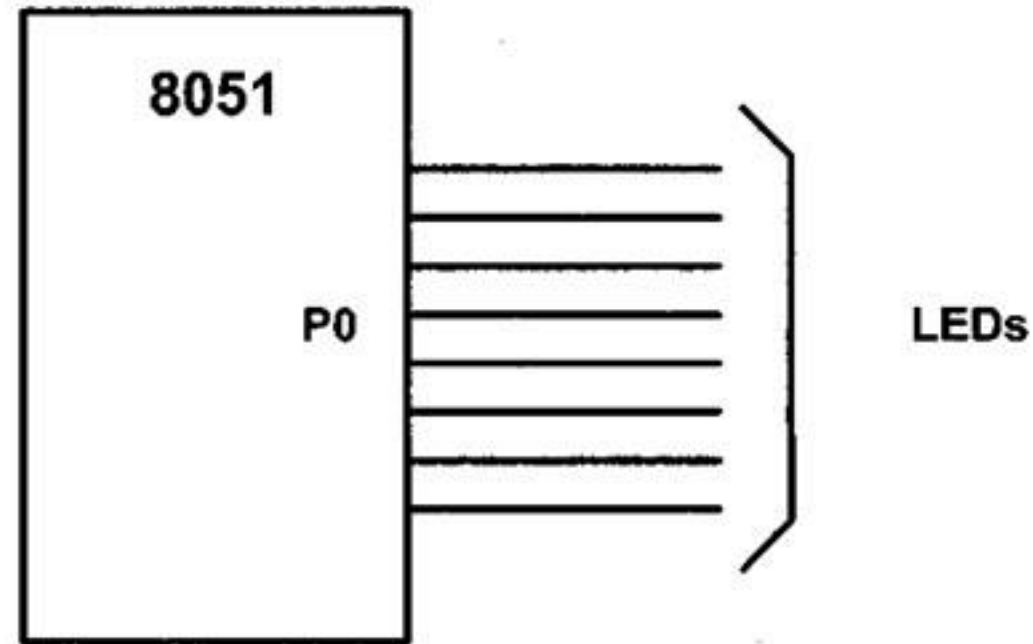
In 8051 C we can access the timer registers TH, TL, and TMOD directly using the reg51 .h header file.

### Example 1-56

Write a 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.

$$\text{FFFFH} - \text{3500H} = \text{CAFFH} = 51967 + 1 = 51968$$

$51968 \times 1.085 \mu\text{s} = 56.384 \text{ ms}$  is the approximate delay.



```
#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
}
```

Write an 8051 C program to toggle only bit P1.5 continuously every 50 ms.

Use Timer 0, mode 1 (16-bit) to create the delay.

Test the program (a) on the AT89C51 and (b) on the DS89C420.

**Solution:**

```
#include <reg51.h>
void TOM1Delay(void);
sbit mybit=P1^5;
void main(void)
{
    while(1)
    {
        mybit=~mybit; //toggle P1.5
        TOM1Delay(); //Timer 0, mode 1(16-bit)
    }
}
```

(a) Tested for AT89C51, XTAL=11.0592 MHz, using the Proview32 compiler

```
void TOM1Delay(void)
{
    TMOD=0x01; //Timer 0, mode 1(16-bit)
    TL0=0xFD; //load TL0
    TH0=0x4B; //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
}
```

E  
ring

(b) Tested for DS89C420, XTAL=11.0592 MHz, using the Proview32 compiler

```
void T0M1Delay(void)
{
    TMOD=0x01;           //Timer 0, mode 1(16-bit)
    TL0=0xFD;             //load TL0
    TH0=0x4B;             //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
}
```

$$\text{FFFFH} - 4\text{BFDH} = \text{B402H} = 46082 + 1 = 46083$$

$$\text{Timer delay} = 46083 \times 1.085 \mu\text{s} = 50 \text{ ms}$$



A T M E  
College of Engineering



# Programs Continued



A T M E  
College of Engineering



Write an 8051 C program to toggle only bit P1.5 continuously every 50 ms. Use Timer 0, mode 1 (16-bit) to create the delay.

Test the program

- (a) on the AT89C51 and
- (b) on the DS89C420.

**Solution:**





# ATME

College of Engineering



```
#include <reg51.h>
void TOM1Delay(void);
sbit mybit=P1^5;
void main(void)
{
    while(1)
    {
        mybit=~mybit;    //toggle P1.5
        TOM1Delay();    //Timer 0, mode 1(16-bit)
    }
}
```

(a) Tested for AT89C51, XTAL=11.0592 MHz, using the Proview32 compiler

```
void T0M1Delay(void)
{
    TMOD=0x01;           //Timer 0, mode 1(16-bit)
    TL0=0xFD;            //load TL0
    TH0=0x4B;            //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
}
```

(b) Tested for DS89C420, XTAL=11.0592 MHz, using the Proview32 compiler

```
void TOM1Delay(void)
{
    TMOD=0x01;           //Timer 0, mode 1(16-bit)
    TL0=0xFD;             //load TL0
    TH0=0x4B;             //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
}
```

$$\text{FFFFH} - 4\text{BFDH} = \text{B402H} = 46082 + 1 = 46083$$

$$\text{Timer delay} = 46083 \times 1.085 \mu\text{s} = 50 \text{ ms}$$

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:**

A5FEH = 42494 in decimal

$65536 - 42494 = 23042$

$23042 \times 1.085 \mu\text{s} = 25 \text{ ms}$  and  $20 \times 25 \text{ ms} = 500 \text{ ms}$

//tested for DS89C420, XTAL = 11.0592 MHz, using the Proview32 compiler

```
#include <reg51.h>
void T1M1Delay(void);
void main(void)
{
    unsigned char x;
    P2=0x55;
    while(1)
    {
        P2=~P2;        //toggle all bits of P2
        for(x=0;x<20;x++)
            T1M1Delay();
    }
}
```



```
void T1M1Delay(void)
{
    TMOD=0x10;           //Timer 1, mode 1(16-bit)
    TL1=0xFE;            //load TL1
    TH1=0xA5;            //load TH1
    TR1=1;               //turn on T1
    while(TF1==0);       //wait for TF1 to roll over
    TR1=0;               //turn off T1
    TF1=0;               //clear TF1
}
```

A5FEH = 42494 in decimal

$65536 - 42494 = 23042$

$23042 \times 1.085 \mu\text{s} = 25 \text{ ms}$  and  $20 \times 25 \text{ ms} = 500 \text{ ms}$



A switch is connected to pin P1.7.

Write an 8051 C program to monitor SW and create the following frequencies on pin P1.5:

SW=0: 500 Hz

SW=1: 750 Hz

Use Timer 0, mode 1 for both of them.

**Solution:**

$$FC67H = 64615$$

$$65536 - 64615 = 921$$

$$921 \times 1.085 \mu s = 999.285 \mu s$$

$$1 / (999.285 \mu s \times 2) = 500 \text{ Hz}$$

//tested for AT89C51/52, XTAL = 11.0592 MHz, using the Proview32 compiler

```
#include <reg51.h>
```

```
sbit mybit=P1^5;
```

```
sbit SW=P1^7;
```

```
void TOM1Delay(unsigned char);
```

```
void main(void)
```

```
{
```

```
    SW=1;
```

```
    //make P1.7 an input
```

```
    while(1)
```

```
    {
```

```
        mybit=~mybit;
```

```
        //toggle P1.5
```

```
        if(SW==0)
```

```
        //check switch
```

```
            TOM1Delay(0);
```

```
        else
```

```
            TOM1Delay(1);
```

```
    }
```

```
}
```

```
void T0M1Delay(unsigned char c)
{
    TMOD=0x01;
    if(c==0)
    {
        TL0=0x67;           //FC67
        TH0=0xFC;
    }
    else
    {
        TL0=0x9A;           //FD9A
        TH0=0xFD;
    }
    TR0=1;
    while(TF0==0);
    TR0=0;
    TF0=0;
}
```

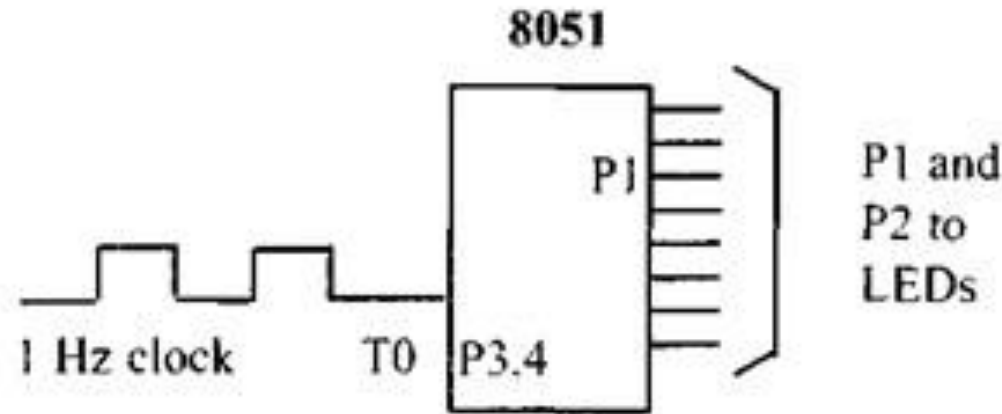
# C Programming of timers 0 and 1 as counters

A timer can be used as a counter if we provide pulses from outside the chip instead of using the frequency of the crystal oscillator as the clock source.

By feeding pulses to the T0 (P3.4) and T1 (P3.5) pins, we turn Timer 0 and Timer 1 into counter 0 and counter 1, respectively

Assume that a **1-Hz external clock** is being fed into pin T0 (P3.4). Write a C program for counter 0 in mode -1 (16-bit) to count the pulses and display the TH0 and TLO registers on P2 and P1, respectively.

**Solution:**



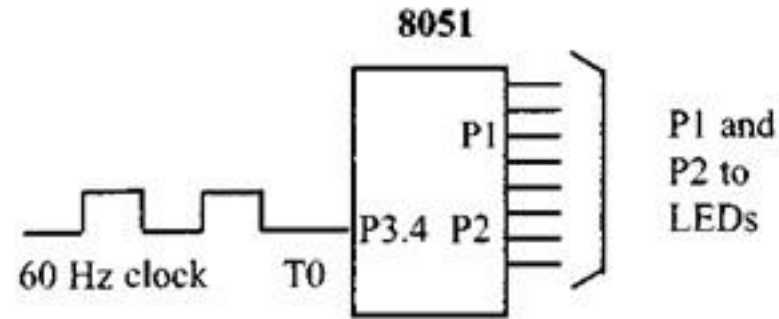
```
{
    T0=1;           //make T0 an input
    TMOD=0x05;      //
    TL0=0;          //set count to 0
    TH0=0;          //set count to 0

    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;
    }
}
```

E  
ing

Assume that a 60-Hz external clock is being fed into pin T0 (P3.4). Write a C program for counter 0 in mode 2 (8-bit auto-reload) to display the seconds and minutes on P1 and P2, respectively.

**Solution:**



By using 60 Hz, we can generate seconds, minutes, hours.



```
#include <reg51.h>
void ToTime(unsigned char);
void main()
{
    unsigned char val;
    T0=1;
    TMOD=0x06;                //T0, mode 2, counter
    TH0=-60;                  //sec = 60 pulses
    while(1)
    {
        do
        {
            TR0=1;
            sec=TL0;
            ToTime(val);
        }
        while (TF0==0);
        TR0=0;
        TF0=0;
    }
}
```

```
void ToTime(unsigned char val)
{
    unsigned char sec, min;
    min = value / 60;
    sec = value % 60;
    P1 = sec;
    P2 = min;
}
```

