UNIT – 3: EMBEDDED C PROGRAMMING & TIMER OPERATION

3.1 Embedded C Programming:

Why C:

Compilers produce hex files that are downloaded into ROM of microcontroller. The assembly language produces a hex file that is much smaller than C. Assembly language programming is tedious and time consuming. C is less time consuming and much easier to write.

Advantages of C programming:

a. Easier and less time consuming to write in C than assembly programming.
b. C is easier to modify and update
c. You can use code available in function libraries
d. C code is portable to other microcontroller with little or no modification

Data Types of 8051 C:

A good understanding of C data types for 8051 can help programmers to create smaller hex files.

Various data types of C:

i. Unsigned char
ii. Signed char
iii. Unsigned int
iv. Signed int
v. Sbit (single bit)
vi. Bit and sfr

i. Unsigned char:

Since 8051 is an 8-bit microcontroller, the character data type is the most natural choice for many applications and most widely used data types for 8051. It is an 8-bit data type that takes a value in the range of 0 – 255(00H – FFH). It occupies one byte location in internal RAM. In setting a counter value, to store string of ASCII characters, where there is no need for signed data, unsigned char should be used instead of signed char. Default is signed char. ‘unsigned’ keyword is used to declare unsigned char.

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;
}
```

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

Solution:

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

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

ii. Signed char:

It is an 8-bit data type that uses MSB to represent the – (negative) or + (positive) values. Lower 7 bits are used to represent magnitude of the signed number from -128 to +127. It is used to represent quantity like temperature.

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

```
#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];
}
```

iii. Unsigned int:

It is a 16-bit data type that takes value in the range of 0 to 65535 (0000H – FFFFH). It is used to define 16-bit variables like memory address and to set counter values of more than 256. It occupies two bytes of location in internal RAM. If one byte location is sufficient, char data type should be used instead of int because the size of hex file will be smaller with the use of char data type. 'unsigned' keyword is used to declare unsigned int otherwise signed int is the default type of int.

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

```
#include <reg51.h>
sbit MYBIT=P1^0;
void main(void)
{
unsigned int z;
for (z=0;z<=50000;z++)
{
MYBIT=0;
MYBIT=1;
}
}
```

iv. Signed int:
It is a 16-bit data type that uses the MSB to represent – (negative) or + (positive) values. Lower 15 bits are used to represent the magnitude of the number from -32768 to +32767.

v. Sbit (Single bit):
'sbit' keyword is used to access single-bit addressable Special Function Registers including P0 to P3.

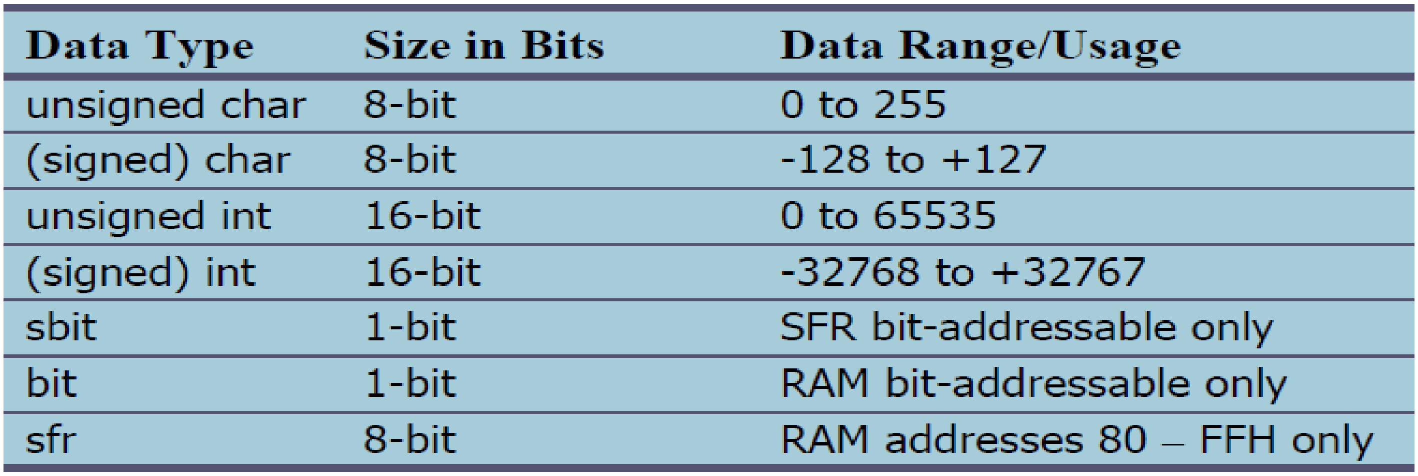
Write an 8051 C program to toggle only bit P2.4 continuously without disturbing the rest of the bits of P2.
Solution:

```
#include <reg51.h>
sbit mybit=P2^4;
void main(void)
{
while (1)
{
mybit=1; //turn on P2.4
mybit=0; //turn off P2.4
}
}
```

vi. Bit and sfr:
'bit' data type allows access to single bits of bit-addressable memory of internal RAM from 20H to 2FH. 'sfr' data type is used to access byte-size SFR registers.

Write an 8051 C program to get the status of bit P1.0, save it, and send it to P2.7 continuously.
Solution:

```
#include <reg51.h>
sbit inbit=P1^0;
sbit outbit=P2^7;
bit membit; //use bit to declare bit- addressable memory
void main(void)
{
while (1)
{
membit=inbit; //get a bit from P1.0
outbit=membit; //send it to P2.7
}
}
```



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

Time Delay:

There are two ways to create delay:

a. Using simple for loop
b. Using timers

Factors which affect delay using for loop:

1. 8051 uses 12 clock periods per machine cycle. Newer generation of 8051 like DS5000 uses 4 clock periods per machine cycle, DS89C420 uses only one clock periods per machine cycle. Lesser the number of clock periods per machine cycle, faster the operation of microcontroller, lesser the delay.

2. The crystal frequency connected to XTAL1 and XTAL2 input pins. The duration of clock period for machine cycle is a function of crystal frequency.

3. C compiler converts C program to assembly language instructions. Different compilers produce different code and hence produce different hex code.

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

Solution:

```
#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++);
}
}
```

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

```
#include <reg51.h>
void MSDelay(unsigned int);
void main(void)
{
while (1) //repeat forever
{
P1=0x55;
MSDelay(250);
p1=0xAA;
MSDelay(250);
}
}
void MSDelay(unsigned int itime)
{
unsigned int i,j;
for (i=0;i<itime;i++)
for (j=0;j<1275;j++);
}
```

Logic operations in 8051 C:
One of the most important and powerful features of the C language is its ability to perform bit manipulation.

Byte-wise operators in C:
a. AND (&&),
b. OR (||),
c. NOT (!)

Bit-wise operators in C:
a. AND (&),
b. OR (|),
c. EX-OR (^),
d. Inverter (~),
e. Shift Right (>>),
f. Shift Left (<<)

Bit-wise shift operation in C:
a. Shift Right (>>),
b. Shift Left (<<)

Format: data >> number of bits to be shifted right
data << number of bits to be shifted left.

Example: 0x9A >> 3 = 0x13.

These operators are widely used in software engineering for embedded systems and control.

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
P2=0x77 >> 4; //shifting right 4
P0=0x6 << 4; //shifting left 4
}
```

Write an 8051 C program to toggle all the bits of P0 and P2 continuously with a 250 ms delay. Using the inverting and Ex-OR operators, respectively.
Solution:

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

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;
P1=x|0x30;
y=mybyte&0xF0;
y=y>>4;
P2=y|0x30;
}
```

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;
w=w<<4;
z=z&0x0F;
bcdbyte=w|z;
P1=bcdbyte;
}
```

In 8051 data can be stored in three types of memory:

1. The 128 bytes of RAM space with address range 00H to 7FH.

2. The 64 kbytes of on-chip code or program memory with address range 0000H to FFFFH. Here both program as well data can be stored and accessed using program counter (PC). To read data from this memory "movc a, @a+dptr" instruction is used in assembly language program. To store data into this memory DB (define byte) assembler directive is used. If more data is stored in this memory, less space is left for program. This space is used to store predefined data and tables. But data cannot be written to it during execution of program.

3. The 64 kbytes of external memory can be used as both RAM and ROM. MOVX instruction is used to access external memory.

RAM data space usage by 8051 C compiler:

C compiler allocates RAM locations in the following order:
1. Bank 0 – address 00H to 07H
2. Individual variables – addresses 08H and beyond
3. Array elements – addresses right after variables
4. Stack – addresses right after array elements.

Write, compile and single-step the following program on your 8051simulator. Examine the contents of the code space to locate the values.
Solution:

```
#include <reg51.h>
void main(void)
{
unsigned char mydata[100]; //RAM space
unsigned char x,z=0;
for (x=0;x<100;x++)
{
z--;
mydata[x]=z;
P1=z;
}
}
```

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

Accessing Code ROM Space in 8051 C:

To store data in code space or memory instead of RAM memory, keyword *code* is used in front of the variable declaration

```
Ex:     code unsigned char num[] = “1234”;
        code unsigned char weekdays=7, month=0x12;
```

Compile and single-step the following program on your 8051 simulator. Examine the contents of the code space to locate the ASCII values.
Solution:

```
#include <reg51.h>
void main(void)
{
code unsigned char mynum[]=“ABCDEF”;
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’;
}
```

Short and simple, but the individual characters are embedded into the program and it mixes the code and data together.

(b)

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

Use the RAM data space to store array elements, therefore the size of the array is limited.

(c)

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

Use 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. However, the more code space you use for data, the less space is left for your program code.

Sample C Programs:

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

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

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>
#defind LED 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 form P0. 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 input port
while (1)
{
mybyte=P0; //get a byte from P0
if (mybyte<100)
P1=mybyte; //send it to P1
else
P2=mybyte; //send it to P2
}
}
```

Write an 8051 C program to monitor bit P1.5. If it is high, send 55H to P0; otherwise, send AAH to P2.
Solution:

```
#include <reg51.h>
sbit mybit=P1^5;
void main(void)
{
mybit=1; //make mybit an input
while (1)
{
if (mybit==1)
P0=0x55;
else
P2=0xAA;
}
}
```

Write an 8051 C program to turn bit P1.5 on and off 50,000 times.
Solution:

```
sbit MYBIT=0x95;
```

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

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

```
#include <reg51.h>
sbit inbit=P1^0;
sbit outbit=P2^7;
bit membit;
void main(void)
{
while (1)
{
membit=inbit; //get a bit from P1.0
outbit=~membit; //invert it and send
//it to P2.7
}}
```

3.2 TIMERS AND COUNTERS

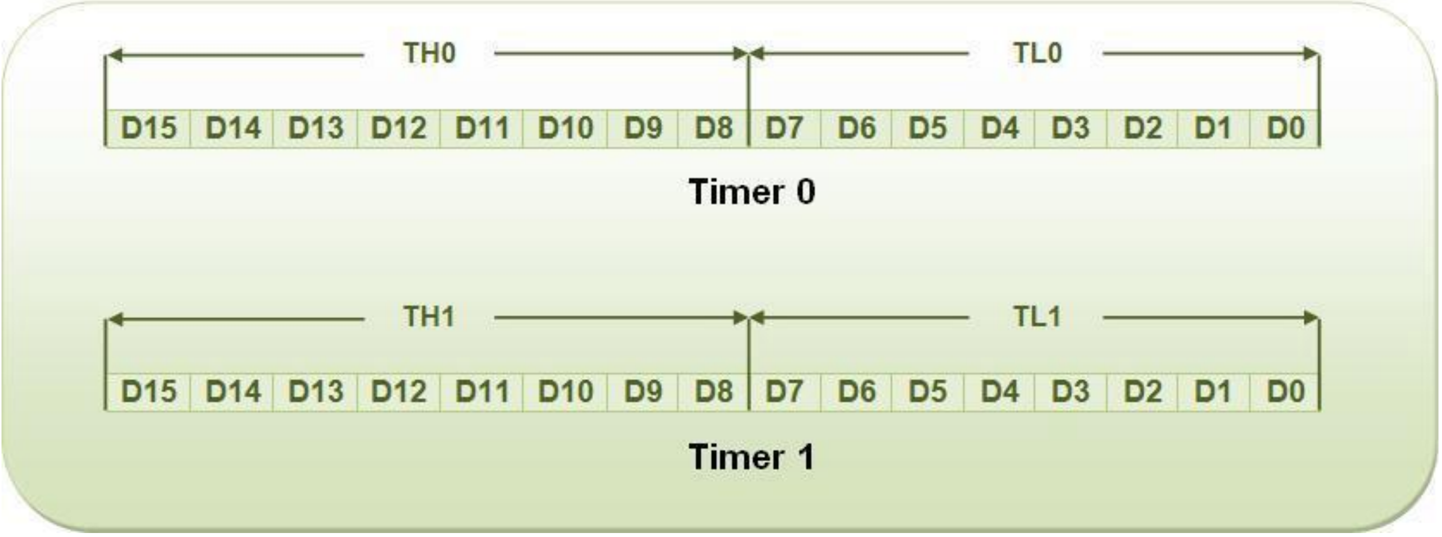
The 8051 has two timers/counters, Timer 0 and Timer 1. They can be used as either timers to generate a time delay or as counters to count events happening outside the microcontroller. Both Timer 0 and Timer 1 are 16 bits wide. They are accessed as two separate registers of low byte and high byte.

Timer 0 (T0) Register:

Low byte register of T0 is called TL0 and high byte is called as TH0. These registers can be accessed like any other register. For example, MOV TL0, #3EH, will transfer data 3EH to TL0, and MOV A, TH0 will transfer content of TH0 to accumulator.

Timer 1 (T1) Register:

Low byte register of T1 is called TL1 and high byte is called as TH1. These registers are accessible in the same way as registers of Timer 0.


TH0
TL0
D15 D14 D13 D12 D11 D10 D9 D8 D7 D6 D5 D4 D3 D2 D1 D0
Timer 0
TH1
TL1
D15 D14 D13 D12 D11 D10 D9 D8 D7 D6 D5 D4 D3 D2 D1 D0
Timer 1

The following are timer related SFRs in 8051.

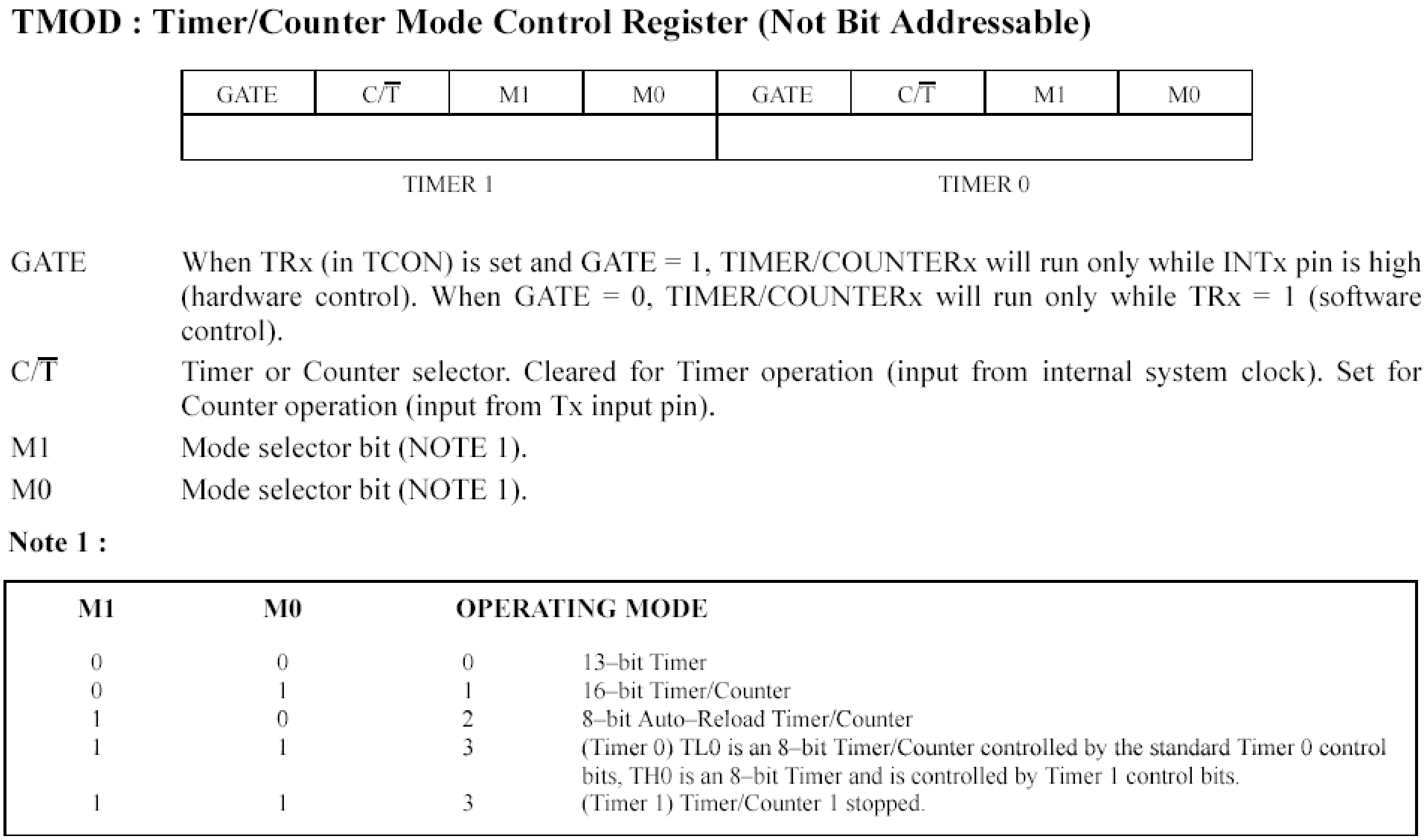
SFR Name	Description	SFR Address
TH0	*Timer0 High Byte*	*8Ch*
TL0	*Timer0 Low Byte*	*8Ah*
TH1	*Timer1 High Byte*	*8Dh*
TL1	*Timer1 Low Byte*	*8Bh*
TCON	*Timer Control*	*88h*
TMOD	*Timer Mode*	*89h*

Timers/Counters are used generally for:

a. Time reference
b. Creating delay
c. Wave form properties measurement
d. Periodic interrupt generation
e. Waveform generation

TMOD (Timer Mode) Register:

It is used to various timer operation modes of T0 and T1. It is an 8 bit SFR in which lower 4 bits are used for T0 and upper 4 bits are used for T1.



TMOD : Timer/Counter Mode Control Register (Not Bit Addressable)

GATE	C/$\overline{T}$	M1	M0	GATE	C/$\overline{T}$	M1	M0
TIMER 1				TIMER 0			

GATE — When TRx (in TCON) is set and GATE = 1, TIMER/COUNTERx will run only while INTx pin is high (hardware control). When GATE = 0, TIMER/COUNTERx will run only while TRx = 1 (software control).

C/$\overline{T}$ — Timer or Counter selector. Cleared for Timer operation (input from internal system clock). Set for Counter operation (input from Tx input pin).

M1 — Mode selector bit (NOTE 1).

M0 — Mode selector bit (NOTE 1).

Note 1 :

M1	M0	OPERATING MODE	
0	0	0	13–bit Timer
0	1	1	16–bit Timer/Counter
1	0	2	8–bit Auto–Reload Timer/Counter
1	1	3	(Timer 0) TL0 is an 8–bit Timer/Counter controlled by the standard Timer 0 control bits, TH0 is an 8–bit Timer and is controlled by Timer 1 control bits.
1	1	3	(Timer 1) Timer/Counter 1 stopped.

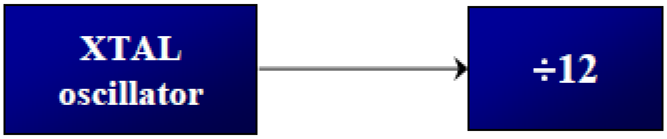
M1, M0:

They are used to select timer mode. There are four modes of operation.

1. Mode 0 – 13-bit timer
2. Mode 1 – 16-bit timer/counter
3. Mode 2 – 8-bit auto-reload timer/counter
4. Mode 3 – Split timer.

C/T (clock/timer):

This bit is used to select timer hardware as a timer or counter. If this bit is set to 0, timer hardware act as a timer to generate time delay by counting internal clock pulse generated by the crystal oscillator. If the bit is set to 1, the hardware acts as a counter to count external clock pulse which represents an event.



Timer in 8051 is used as timer, counter and baud rate generator. Timer always counts up irrespective of whether it is used as timer, counter, or baud rate generator: Timer is always incremented by the microcontroller. The time taken to count one digit up is based on master clock frequency.
If Master CLK=11.0592 MHz,
Timer Clock frequency = Master CLK/12 = 921.6 kHz
Timer Clock Period = 1.085 micro second
This indicates that one increment in count will take 1.085 micro second.

The two timers in 8051 share two SFRs (TMOD and TCON) which control the timers, and each timer also has two SFRs dedicated solely to itself (TH0/TL0 and TH1/TL1).

Indicate which mode and which timer is 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.

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:

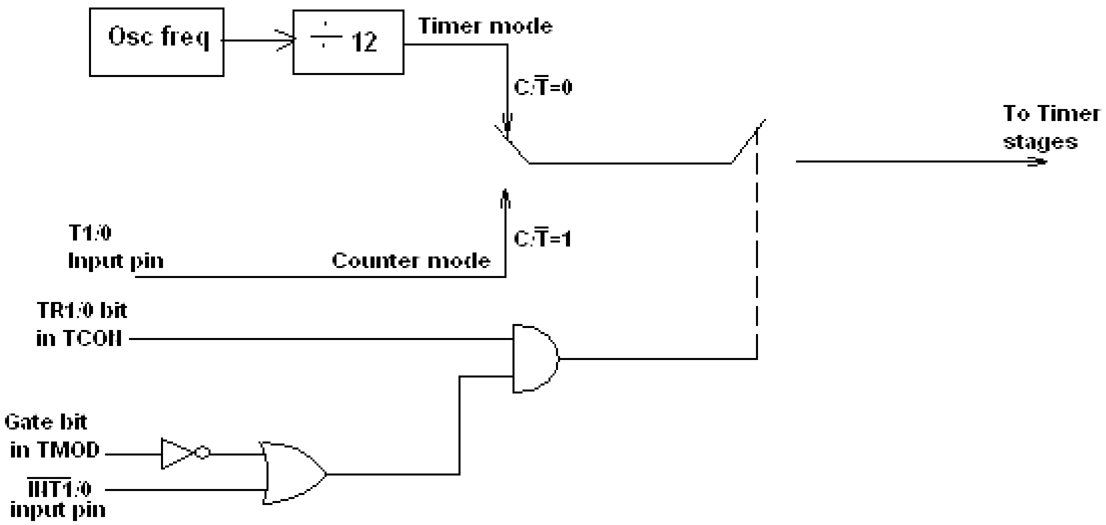
a. (1/12) * 12 MHz = 1MHz and T=1/1 MHz = 1µs.
b. (1/12) * 16 MHz = 1.333 MHz and T=1/1.333 MHz = 0.75µs.
c. (1/12) * 11.0592 MHz = 921.6 kHz and T=1/921.6 kHz MHz = 1.085µs.

Clock source for timer:

Crystal frequency attached to 8051 is the source of the clock for the timer. The size of crystal frequency decides the speed at which the timer ticks. The frequency for the timer is always (1/12)th the frequency of the crystal oscillator. The oscillator frequency ranges from 10 MHz to 40 MHz. Though frequency 11.0592 MHz is an odd number, it is preferred for generating baud rate for serial communication. It allows 8051 to communicate with the IBM PC with no errors.

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 timers in 8051 have both. The start and stop of the timer are controlled by way of software by the TR (timer start) bits TR0 and TR1. This is achieved by the instructions "SETB TR1" and "CLR TR1" for Timer 1 and 'SETB TR0" and "CLR TR0" for Timer 0. The SETB instruction starts and CLR instruction stops the timer. This method works when gate bit of TMOD is 0. When gate bit is 1, external hardware is used to start and stop the timer. The signal from external hardware is applied to external interrupt pins INT0' (P3.2) and INT1' (P3.3).



Finding values to be loaded into the timer:

If time delay to be generated is known, then value to be loaded to timer is calculated using two methods: Decimal and Hexadecimal method.

a. Hexadecimal method:

- Divide the desired time delay by 1.085 µs which gives number of clock pulses to be counted to generate the required delay.
- Convert the number of clock pulses to hexadecimal value (YYXX).
- Perform (FFFF – YYXX + 1) gives value to be loaded to TH and TL.

b. Decimal method:

- Divide the desired time delay by 1.085 µs which gives number of clock pulses to be counted to generate the required delay (n).
- Perform 65536 – n.
- Convert the decimal value got in previous step to hexadecimal which is the value to be loaded to TH and TL.

Time Delay Calculation for XTAL = 11.0592 MHz:

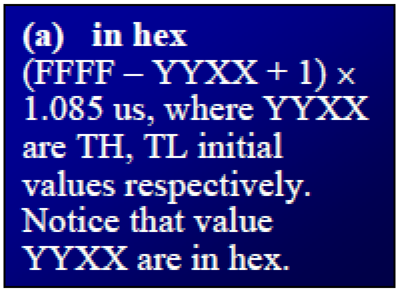
If initial value (YYXX, TH=YY, TL=XX) is known, time delay can be calculated using two methods: Decimal and Hexadecimal method.

a. Hexadecimal method:

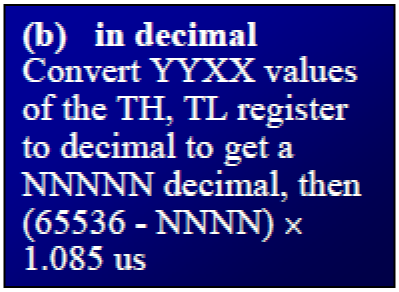
- Perform (FFFF – YYXX +1) * 1.085 µs which gives the value of time delay.

b. Decimal method:

- Convert YYXX to decimal value NNNN.
- Perform (65536 – NNNN) * 1.085 µs which gives the value of time delay.

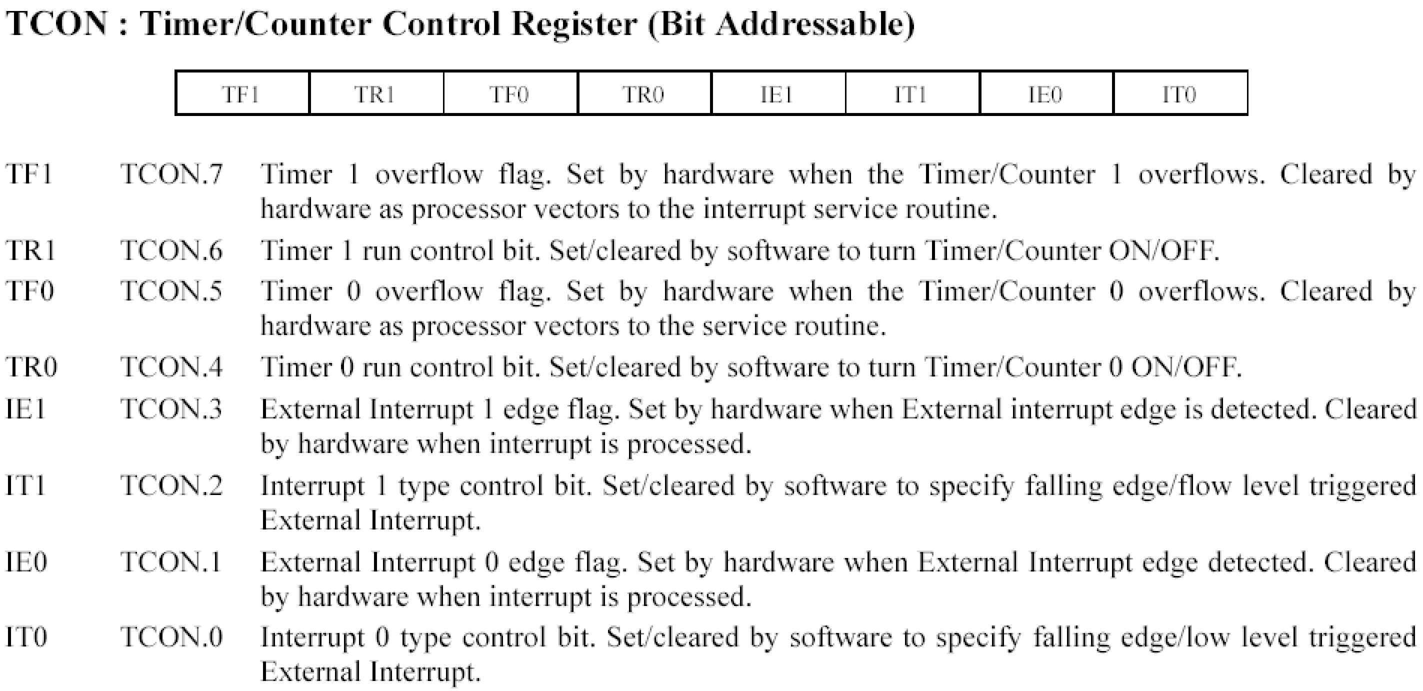


(a) in hex
(FFFF – YYXX + 1) × 1.085 us, where YYXX are TH, TL initial values respectively. Notice that value YYXX are in hex.



(b) in decimal
Convert YYXX values of the TH, TL register to decimal to get a NNNNN decimal, then (65536 - NNNN) × 1.085 us

TCON Register:



TCON : Timer/Counter Control Register (Bit Addressable)

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

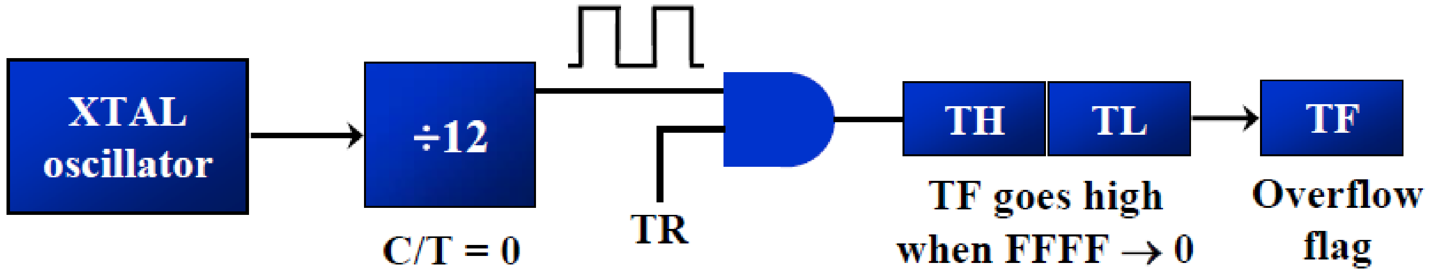
TF1	TCON.7	Timer 1 overflow flag. Set by hardware when the Timer/Counter 1 overflows. Cleared by hardware as processor vectors to the interrupt service routine.
TR1	TCON.6	Timer 1 run control bit. Set/cleared by software to turn Timer/Counter ON/OFF.
TF0	TCON.5	Timer 0 overflow flag. Set by hardware when the Timer/Counter 0 overflows. Cleared by hardware as processor vectors to the service routine.
TR0	TCON.4	Timer 0 run control bit. Set/cleared by software to turn Timer/Counter 0 ON/OFF.
IE1	TCON.3	External Interrupt 1 edge flag. Set by hardware when External interrupt edge is detected. Cleared by hardware when interrupt is processed.
IT1	TCON.2	Interrupt 1 type control bit. Set/cleared by software to specify falling edge/flow level triggered External Interrupt.
IE0	TCON.1	External Interrupt 0 edge flag. Set by hardware when External Interrupt edge detected. Cleared by hardware when interrupt is processed.
IT0	TCON.0	Interrupt 0 type control bit. Set/cleared by software to specify falling edge/low level triggered External Interrupt.

TCON is an 8-bit, bit accessible special function register. The upper four bits consists of timer run bit and timer overflow flag bit for both T0 and T1. The lower four bits are used for interrupt operation. TR0 and TR1 bits stands for timer run bit and are used to start and stop the TR0 and TR1 respectively. TF0 and TF1 bits stands for timer overflow flag for T0 and T1 respectively. These two flags are set to 1 when the timer resets to initial value from FFFFH.

Mode 1 Programming:

Following are the characteristics and operations of mode 1:

1. It is a 16-bit timer, hence allows values from 0000H to FFFFH to be loaded into the timers registers TL and TH.
2. Initial value is loaded to timer and timer started. Timer starts to count up until it reaches FFFFH.
3. When timer rolls over from FFFFH to 0000H, overflow flag bit corresponding to the timer is set to 1. This flag is monitored continuously using JNB instruction. Once the flag is set to 1, the timer can be stopped using CLR instruction.
4. To repeat the operation, overflag is reset to 0, load the initial value to TL and TH and start the timer.



Steps to program in mode 1:

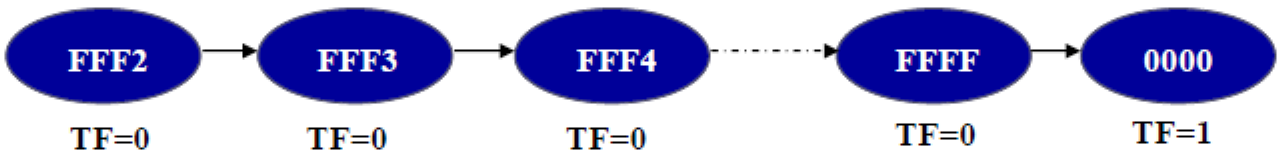
1. Load the TMOD value register indicating which timer (timer 0 or timer 1) is to be used and which timer mode (0 or 1) is selected
2. Load registers TL and TH with initial count value
3. Start the timer
4. Keep monitoring the timer flag (TF) with the JNB TFx, target instruction to see if it is raised. Get out of the loop when TF becomes high
5. Stop the timer
6. Clear the TF flag for the next round
7. Go back to Step 2 to load TH and TL again

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

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

In the above program notice the following step.

1. TMOD is loaded.
2. FFF2H is loaded into TH0-TL0.
3. P1.5 is toggled for the high and low portions of the pulse.
4. The DELAY subroutine using the timer is called.
5. In the DELAY subroutine, timer 0 is started by the SETB TR0 instruction.
6. Timer 0 counts up with the passing of each clock, which is provided by the crystal oscillator. As the timer counts up, it goes through the states of FFF3, FFF4, FFF5, FFF6, FFF7, FFF8, FFF9, FFFA, FFFB, and so on until it reaches FFFFH. One more clock rolls it to 0, raising the timer flag (TF0=1). At that point, the JNB instruction falls through.



7. Timer 0 is stopped by the instruction CLR TR0. The DELAY subroutine ends, and the process is repeated. Notice that to repeat the process, we must reload the TL and TH registers, and start the process is repeated.

Find the delay generated by timer 0 in the following code, using both decimal and hexadecimal method. Do not include the overhead due to instruction.

```
CLR P2.3                  ;Clear P2.3
MOV TMOD, #01             ;Timer 0, 16-bitmode
HERE: MOV TL0, #3EH       ;TL0=3Eh, the low byte
MOV TH0, #0B8H            ;TH0=B8H, the high byte
SETB P2.3                 ;SET high timer 0
SETB TR0                  ;Start the timer 0
AGAIN: JNB TF0, AGAIN     ;Monitor timer flag 0
CLR TR0                   ;Stop the timer 0
CLR TF0                   ;Clear TF0 for next round
CLR P2.3
```

Solution:

(a) Hexadecimal Method: (FFFFH – B83E + 1) = 47C2H = 18370 in decimal and 18370 × 1.085 us = 19.93145 ms

(b) Decimal Method: Since TH – TL = B83EH = 47166 (in decimal) we have 65536 – 47166 = 18370. This means that the timer counts from B38EH to FFFF. This plus Rolling over to 0 goes through a total of 18370 clock cycles, where each clock is 1.085 us in duration. Therefore, we have 18370 × 1.085 us = 19.93145 ms as the width of the pulse.

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

Solution:

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

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

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

(d) TL = 1A and TH = FF, all in hex.

```
MOV TMOD, #01             ;Timer 0, 16-bitmode
AGAIN: MOV TL1, #1AH      ;TL1=1A, low byte of timer
MOV TH1, #0FFH            ;TH1=FF, the high byte
SETB TR1                  ;Start timer 1
BACK: JNB TF1,BACK        ;until timer rolls over
CLR TR1                   ;Stop the timer 1
CLR P1.5                  ;Clear timer flag 1
CLR TF1                   ;Clear timer 1 flag
SJMP AGAIN                ;Reload timer
```

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

Solution:

(a) T = 1 / 50 = 20 ms, the period of square wave.
(b) 1 / 2 of it for the high and low portion of the pulse is 10 ms.
(c) 10 ms / 1.085 µs = 9216 and 65536 – 9216 = 56320 in decimal, and in hex it is DC00H.
(d) TL = 00 and TH = DC (hex).

```
MOV TMOD, #10H          ;Timer 1, mod 1
AGAIN: MOV TL1, #00     ;TL1=00,low byte of timer
MOV TH1, #0DCH          ;TH1=DC, the high byte
SETB TR1                ;Start timer 1
BACK: JNB TF1,BACK      ;until timer rolls over
CLR TR1                 ;Stop the timer 1
CLR P2.3                ;Comp. p2.3 to get hi, lo
SJMP AGAIN              ;Reload timer mode 1 isn't auto-reload
```

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

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

Solution:

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

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

Examine the following program and find the time delay in seconds. Exclude the overhead due to the instructions in the loop.

```
MOV TMOD, #10H               ;Timer 1, mod 1
MOV R3, #200                 ;cnter for multiple delay
AGAIN: MOV TL1,   #08H       ;TL1=08,low byte of timer
MOV TH1,     #01H            ;TH1=01,high byte
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
DJNZ R3, AGAIN               ;if R3 not zero then reload timer
```

Solution:

TH-TL = 0108H = 264 in decimal and 65536 – 264 = 65272. Now 65272 × 1.085 μs = 70.820 ms, and for 200 of them we have 200 ×70.820 ms = 14.164024 seconds.

Generating a large time delay:

Time delay depends on crystal frequency and initial value loaded to the timer. In mode 1, the maximum delay is generated when content of registers are 0, that is TH=00H and TL=00H. The value of delay is
(65536-0000) * 1.085 μs = 71.1 ms. If this delay is not sufficient, then repeat delay subroutine n number of times till required delay is obtained.

For example, if delay required is 1 second, then (1second/71.1 ms) = 14, that is number of times delay subroutine is to be repeated is 14 times.

Examine the following program and find the time delay in seconds. Exclude the overhead due to the instructions in the loop.

```
MOV TMOD, #10H              ;Timer 1, mod 1
MOV R3, #200                ;cnter for multiple delay
AGAIN: MOV TL1, #08H        ;TL1=08,low byte of timer
MOV TH1, #01H               ;TH1=01,high byte
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
DJNZ R3, AGAIN              ;if R3 not zero then reload timer
```

Solution:

TH-TL = 0108H = 264 in decimal and 65536 – 264 = 65272. Now 65272 × 1.085 μs = 70.820 ms, and for 200 of them we have 200 ×70.820 ms = 14.164024 seconds.

In the following program calculate the frequency of the square wave generated on pin P1.5. Consider the overhead due to instruction.

Solution:

To get a more accurate timing, we need to add clock cycles due to these instructions in the loop. To do that, we use the machine cycle of each instruction in the program.

Program	Machine Cycle
HERE: MOV TL0, #0F2H	2
MOV TH0, #0FFH	2
CPL P1.5	1
ACALL DELAY	2
SJMP HERE	2
DELAY: SETB TR0	1
AGAIN: JNB TF0, AGAIN	14
CLR TR0	1
CLR TF0	1
RET	2
	Total 28

T = 2 × 28 × 1.085 us = 60.76 us and F = 16458.2 Hz

Write an ALP to generate maximum delay in mode 1 operation of timer. In your calculation, exclude the overhead due to the instructions in the loop.
Solution:

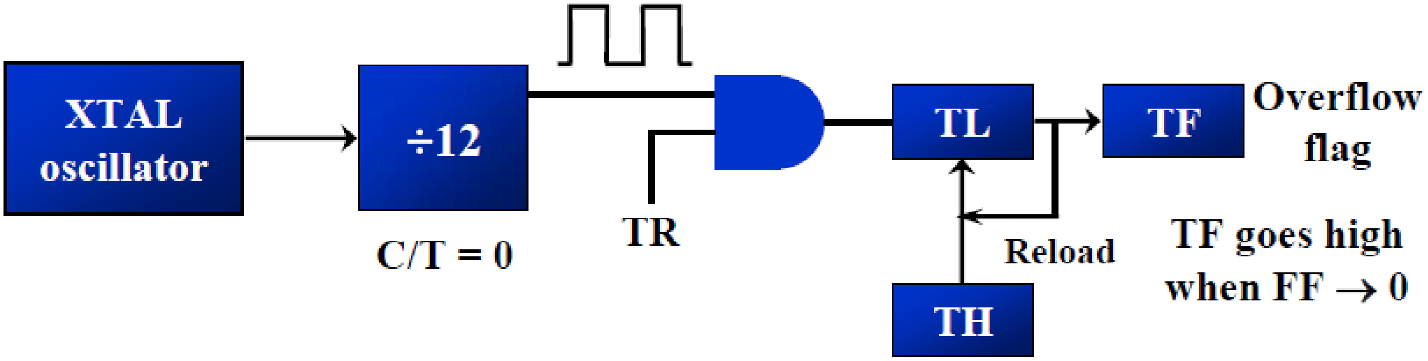
To get the largest delay we make TL and TH both 0. This will count up from 0000 to FFFFH and then roll over to zero.

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

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) × 1.085 us = 71.1065ms.

Mode 0:
In this mode timer is used as a 13 bit timer. The lower 5 bits of TLX and 8 bits of THX are used for the 13 bit count. Upper 3 bits of TLX are ignored. It can hold values between 0000H to 1FFFH. Once it reaches final count of 1FFFH, on the application of clock pulse, the timer will roll back to 0000h. The maximum delay that can be generated in this mode is 8.88 ms. Because of lesser delay, mode 1 is preferred instead of mode 0.

Mode 2:
The following are the characteristics and operations of mode 2.
1. It is an 8-bit timer; therefore, it allows only values of 00 to FFH to be loaded into the timer’s register TH.
2. After TH is loaded with the 8-bit value, the 8051 gives a copy of it to TL. Then the timer must be started. This is done by the instruction SETB TR0 for timer 0 and SETB TR1 for timer 1.
3. After the timer is started, it starts to count up by incrementing the TL register. It counts up until it reaches its limit of FFH. When it rolls over from FFH to 00, it sets high the TF (timer flag).
4. When the TL register rolls from FFH to 00H and TF is set to 1, TL is reloaded automatically with the original value kept by the TH register. To repeat the process, we must simply clear TF and let it go without any need by the programmer to reload the original value. This makes mode 2 an auto-reload, in contrast with mode 1 in which the programmer has to reload TH and TL.
Mode 2 is used in generating baud rate for serial communication.



Steps to program timer in mode 2:

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) is selected.
2. Load the TH registers with the initial count value.
3. Start timer.
4. Keep monitoring the timer flag (TF) with the JNB TFx, target instruction to see whether it is raised. Get out of the loop when TF goes high.
5. Clear the TF flag.
6. Go back to Step4, since mode 2 is auto reload.

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

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

Solution:

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

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

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

T = 2 (250 × 256 × 1.085 μs) = 138.88 ms, and frequency = 72 Hz.

Assemblers and negative values:

To find the value to be loaded to the timer, we have used decimal method and hexadecimal method. Alternate method is to enter value into the assembler. For example, if 100 number of clock pulses is to be counted to generate a delay, then 'MOV TH1, #-100' will load 2's complement of 100 into TH1 which is 9CH. Same result is obtained in both decimal as well as hexadecimal method.

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

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

Solution:

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

Decimal 2's complement (TH value)

-3 = FDH
-12 = F4H
-48 = D0H
-60 = C4H
-200 = 38H

Counter Programming:

If the timer is used to external event happening outside 8051, timer is known as a counter also known as event counter. External event in the form of clock pulse is applied to either pin number 14 (P3.4, T0 input) or 15(P3.5, T1 input) which increments the content of the TH and TL registers.

C/T' bit in TMOD register:

This bit decides the source of clock for the timer. If bit is set to 0, timer circuit receives clock pulses from the crystal and if it is 1, timer receives clock pulses from external source. If external clock pulse applied to P3.4, T0 increments, if applied to P3.5, T1 increments.

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

Solution:

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

```
JNB TF1, Back                          ;keep doing, if TF = 0
CLR TR1                                ;stop the counter 1
CLR TF1                                ;make TF=0
SJMP AGAIN                             ;keep doing it
```

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

Design a counter for counting the pulses of an input signal. The pulses to be counted are fed to pin P3.4. Assume XTAL = 11.0592 MHz.
Solution:

Here timer 1 is used as time base for timing 1 second. During this 1 second, timer 0 is run as a counter with input pulses fed into P3.4. At the end of 1 second, the values in TL0 and TH0 give the number of pulses received at P3.4. This gives the frequency of unknown signal.

```
            ORG 0000H
RPT:        MOV TMOD, #15H
            SETB P3.4
            MOV TL0, #00
            MOV TH0, #00
            SETB TR0
            MOV R0, #14
AGAIN:      MOV TL1, #00H
            MOV TH1, #00H
            SETB TR1
BACK:               JNB TF1, BACK
            CLR TF1
            CLR TR1
            DJNZ R0, AGAIN
            MOV A, TL0
            MOV P2, A
            MOV A, TH0
            MOV P1, A
            SJMP RPT
            END
```

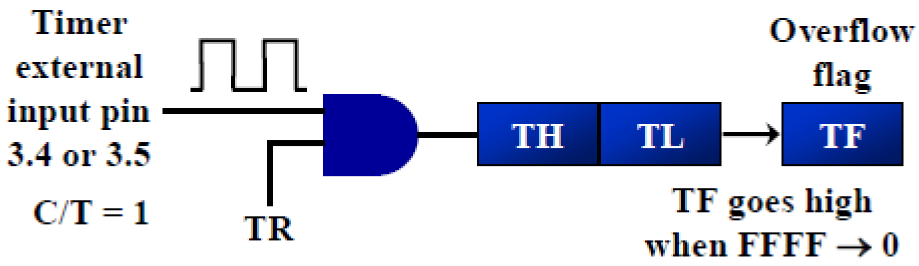
As the frequency varies, the values obtained at P1 and P2 vary. The values obtained for 10 Hz, 25 Hz, 100 Hz, are respectively 0AH, 19H, 64H., etc.,.

In the above program, timer 1 is used as an event counter where it counts up as clock pulses are fed into P3.5. These clock pulses could represent the number of people passing through an entrance or the number of wheel rotations or any other event that can be converted to pulses.

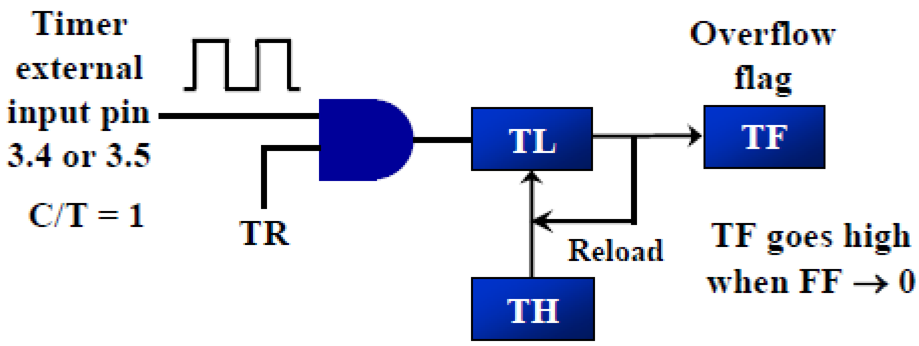
<u>Modes of Operation of Counter:</u>
Modes of operation of counter is similar to timer. The only difference is the source of clock pulse. For timer, the source of clock pulse is crystal where as the source of clock pulse for counter is external clock pulse applied

to P3.4 for timer 0 and P3.5 for timer 1. Figure below shows mode 1 which is 16 bit counter and mode 2 which is 8-bit auto reload mode. For counter operation C/T' bit should be set to 1.

Timer with external input (Mode 1)

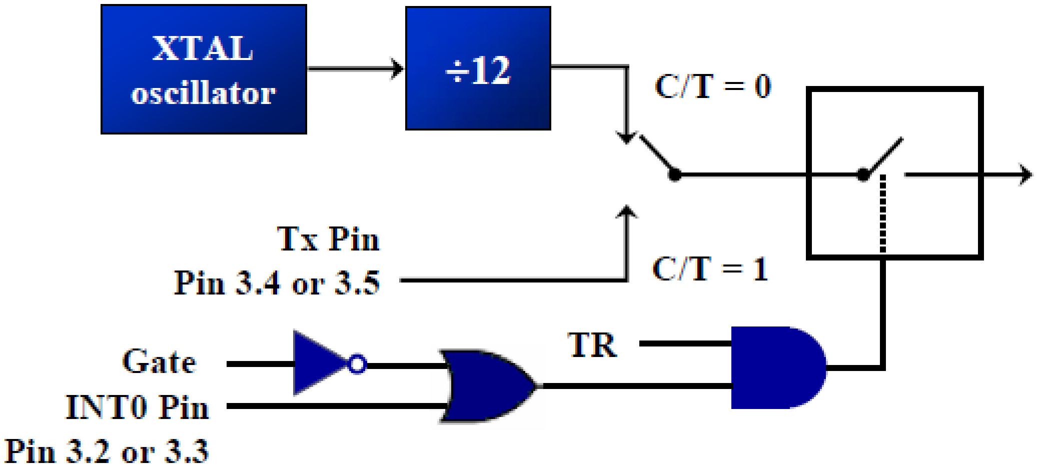


Timer with external input (Mode 2)



When GATE bit =1:

If GATE bit is set, the start and stop of the timer are done externally through pins P3.2 and P3.3 for timer 0 and 1 respectively. This is in spite of the fact that TRx is turned on by the SETB TRX. This will allow start and stop of timer externally via a simple switch. Application includes turning ON and OFF an alarm.


XTAL
oscillator
÷12
C/T = 0
Tx Pin
Pin 3.4 or 3.5
C/T = 1
TR
Gate
INT0 Pin
Pin 3.2 or 3.3