Addressing Modes:

- Various methods of accessing the data are called addressing modes.

8051 addressing modes are classified as follows

1. Immediate addressing
2. Register addressing
3. Direct addressing
4. Indirect addressing
5. Relative addressing
6. Absolute addressing
7. Long addressing
8. Indexed addressing
9. Bit inherent addressing
10. Bit direct addressing.

1. Immediate addressing:

In this addressing mode the data is provided as a part of instruction itself. In other words data immediately follows the instruction.

Eg: MOV A, #30H
ADD A, #83 #Symbol indicates the data is immediate

2. Register addressing:

In this addressing mode the register will hold the data. One of the eight general registers (R0 to R7) can be used and specified as the operand.

Ex: MOV A, R0
MOV A, R_0

R_0-R_7 will be selected from the current selection

of register blank. The default register bank will be bank 0.

3. Direct Addressing:

There are two ways to access the internal memory. Using direct addressing mode and indirect address.
Using direct addressing mode we can not only address the internal memory but SFRs also. In direct addressing an 8 bit internal data memory address is specified as part of the instruction and hence, it can specify the address only in the range of 00H to FFH. In this addressing mode data is obtained directly from the memory.

Ex: MOV A, 60h
ADD A, 30h

4. Indirect Addressing:

The indirect addressing using a register to hold the actual address that will be used in data movement. Registers R_0 and R_1 and DPTR are the only registers that can be used as data pointers. Indirect addressing cannot be used to refer to SFR registers. Both R_0 and R_1 can hold 8 bit address and DPTR can hold 16 bit address.

Ex: MOV A, @R_0
ADD A, @R_1
MOVX A, @DPTR

5. Indexed Addressing:

In indexed addressing either the program counter (PC), or the data pointer (DTPR) is used to hold the base address, and the A is used to hold the offset address. Adding the value of the base address to the value of offset address forms the effective address. Indexed addressing is used with JMP or MOVC instructions. Look up tables are implemented with the help of index addressing.

Ex: MOVC A, @A+DPTR // copies the contents of memory location, pointed by sum of accumulator A and the DPTR into accumulator.

MOVC A, @A+PC

6. Relative addressing:

Relative addressing is used only with unconditional jump instructions. The relative address, (offset), is an 8 bit signed number, which is automatically added to the PC to make the address of the next instruction. The 8 bit signed offset value gives an address range of +127 to -128 locations.

The jump destination is usually specified using a label and the assembler calculates the jump offset accordingly. The advantage of relative addressing is that the program code is easy to relocate and address is relative to position memory.

Ex: SJMPLOOP1

JC BACK

7. Absolute addressing:

Absolute addressing is used only by the AJMP (Absolute Jump) and ACALL (Absolute call) instructions. These are 2 bytes instructions (the absolute addressing mode specifies the low)

Ex: AJMP LOOP1
ACALL LOOP2

8. Long Addressing:

The long addressing mode is used with the instructions LJMP and LCALL. These are 3 bytes instructions. The address specifies a full 16 bit destination address so that a jump or call can be made to be location within 64 Kbyte code memory space.

Ex: LJMP FINISH
LCALL DELAY

10. Bit Direct addressing:

In this addressing mode the direct address of bit is specified in the instruction. The RAM space 20H to 2FH and most of the special function registers are bit addressable. Bit address values are between 00H to 7FH.

Ex: CLR 07H : clears the bit 7 of 20h RAM space
SETB 07H : Sets the bit 7 of 20H of RAM space.

9. Bit Inherent addressing:

In this addressing the address of the flag which contains the operand, is implied in the opcode of instruction

Ex: CLR C ; clears the carry flag to 0

Instruction Set of 8051 Microcontroller:

The instructions of 8051 can be broadly classified under the following headings.

1. Data transfer instructions
2. Arithmetic instructions
3. Logical instructions
4. Branch instructions
5. Subroutine instructions
6. Bit manipulation instructions

1. Data Transfer Instructions:

The data transfer instructions perform data transfer operations from source to destination.

```
MOV A, R2   // contents of register R2 will move to (or) copied
               into register A.
MOV R4, A   // copies the contents of register A to register R4.
MOV A, 65H  // move the contents of memory location of 65H
               to the register A
```

2. Arthmetic Instructions:

The 8051 can perform addition, subtraction, Multiplication and division operations on 8 bit numbers

i) Addition

The 8051 can perform addition operation by using the following

```
ADD A, #45H   // Add the contents of 45H without carry
ADDC A, #0B4H // Add with carry
```

ii) Subtraction.

The 8051 can perform subtraction operation using the following

Instructions i) SUB A, #21H

SBB A, @R1

SBB A, @R4

iii) Multiplication:

MUL AB :

This instruction multiplies two 8 bit unsigned numbers which are stored in A and B register. After multiplication the lower byte of the result is showed in accumulator and higher byte of result is stored in B register.

```
Ex: MOV A, #45H        ;[A]=45H
    MOV B, #0F5H       ;[B] = F5H
    MUL AB             ; [A] x [B] = 45 x F5 = 4209
                       ;[A] = 09H, [B] = 42H
```

Division:

DIV AB : This instruction divides the 8 bit unsigned numbers which is stored in A by 8 bit unsigned number which is stored in B register. After division the result will be stored in accumulator and remainder will be stored in B register.

```
Ex: MOV A, #45H        ;[A]=45H
    MOV B, #0F5H       ;[B]= 1BH
    DIV AB             ; [A]/[B] = E8/1B = 08 H with
                           remainder 10H
                       ;[A] = 08H, [B] = 10H .
```

Increment Ex: - INC A

Decrement Ex: DEC A

3 Logical Instructions :

Logical AND

ANL destination, source : ANL does a bitwise "AND" operation between source and destination, leaving the resulting value in destination. The value in source is not affected "AND" instruction logically AND the bits of source and destination

ANL A, #DATA	ANL A, Rn
ANL A, DIRECT	ANL A, @Ri
ANL DIRECT, A	ANL DIRECT, #DATA

Logical OR:

ORL destination source : ORL does a a bitwise "OR" operation between source and destination, leaving the results value in destination. The value in source is not affected. "OR" instruction logically OR the bits of source and destination

ORL A, #DATA	ORL A, Rn
ORL A, DIRECT	ORL A, @Ri
ORL DIRET, A	ORL DIRECT, #DATA

Logical Ex-OR :

XRL A, #DATA	XRL A, Rn
XRL A, DIRECT	XRL A, @Ri
XRL DIRECT, A	XRL DIRECT, #DATA

Logical NOT :

CPL complements operand, leaving the result in operand. If operand is a single bit then the state of the bit will be reversed. If operand is the Accumulator then all the bits of in Acumulator will be reversed.

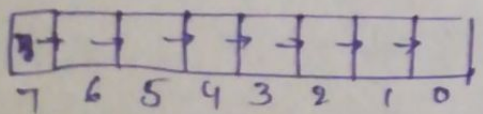
CPLA , CPLC, CPL bit address

SWAP A - Swap the upper nibble and lower nibble of A.

Rotate Instruction:

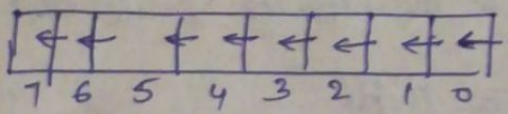
RRA - Rotate Right

This instruction is rotate right the accumulator. Its operation is illustrated below. Each bit is shifted one location to the right with bit 0 going to bit 7.



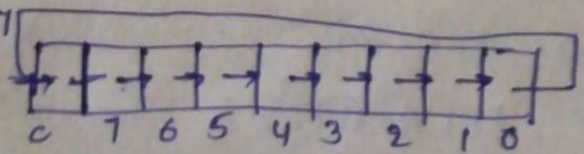
RLA - Rotate Left

Rotate left the accumulator. Each bit is signed shifted one location to the left, with bit.



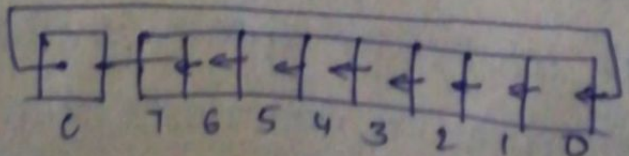
RRCA — Rotate right through carry

Each bit is shifted one location to the right, with bit 0 going into the carry bit in the PSW, while the carry was to goes into bit 7



RLCA -

Each bit is shifted one location to the left, with bit 7 going to the carry bit in the PSW, while the carry goes into bit 0



4. Branch Instructions:

Jump and Call Program Range

There are 3 types of Jump instructions. They are

1. Relative Jump
2. Short Absolute Jump
3. Long Absolute Jump

5. Subroutine CALL and RETURN Instructions:

Subroutines are handled by CALL and RETURN instructions.

* There are two types of CALL Instructions

1. LCALL address (16 bit)

Long call instruction calls the subroutine located within 64 KB of program memory.

2. Absolute CALL (ACALL address) 11 bit:

Absolute CALL instruction calls the subroutine located within 2 KB of program memory.

RET Instruction

RETURN (RET) instruction will return the program from subroutine

6. Bit Manipulation Instructions:

8051 has 128 bit addressable memory. Bit addressable SFRs and bit addressable PORT pins.

By using Bit Manipulation Instructions it is possible to perform to perform bitwise operation.

1. Logical AND
   a. ANL C, BIT

ANL C,/BIT

2) Logical OR

a) ORL C, BIT

b. ORL C,/BIT

3. CLR BIT

a. CLR bit

b. CLR C

4. CPL bit

a. CPL bit

b. CPL C

Timer Counting Programming :

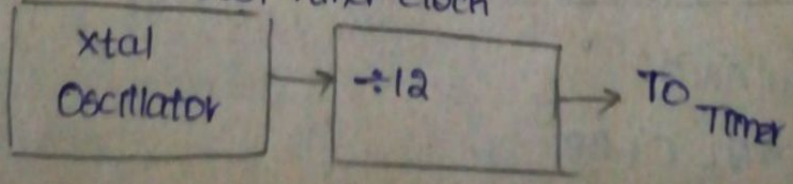
8051 Timers:

8051 ~~Microprogramming~~ microcontrollers have two timers/counters which work on the clock frequency. Time/Counter can be used for time delay generation, counting external events, etc--

Clock:

Every Timer needs a clock to work, and 8051 provides it from an external crystal oscillator which is the main clock source for Timer. The internal circuitry in the 8051 microcontroller provides a clock source to the timers which is 1/12th of the frequency of crystal attached to the microcontroller, also called Machine Cycle Frequency.

8051 Timer Clock



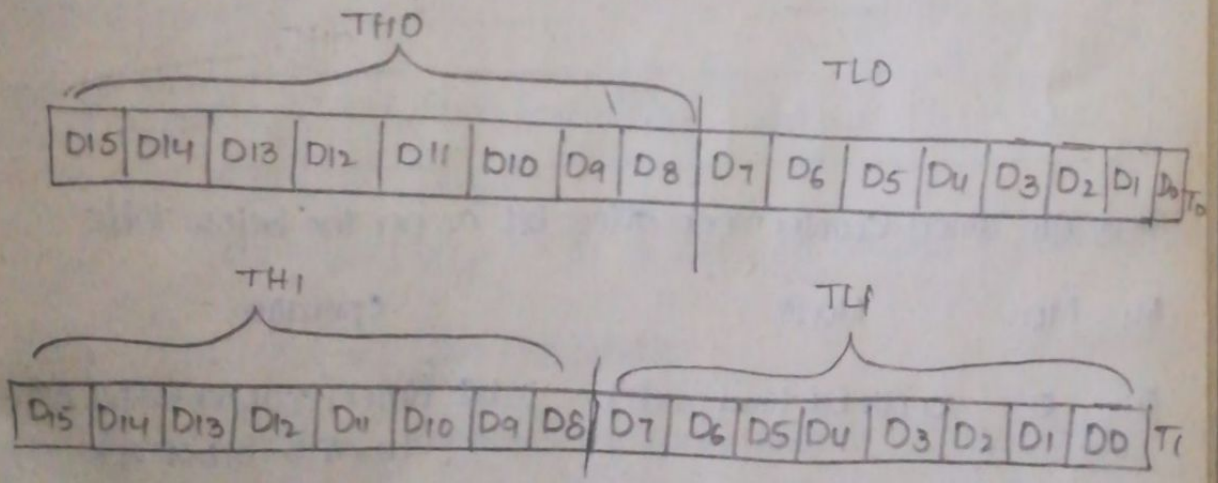
Example: Suppose we have a crystal frequency of 11.0592 MHz then the microcontroller will provide 1/12th i.e

Timer Clock Frequency = (Xtal Osc frequency)/12 = (11.0592 MHz)/12 = 921.6kHz

period T = 1/(921.6kHz) = 1.085 μs

Timer:

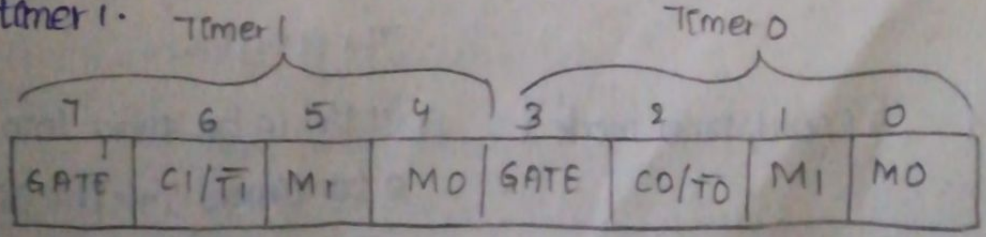
8051 has two timers Timer 0 (T0) and Timer 1 (~~Timer~~ T1), both are 16-bit wide. Since 8051 has 8-bit architecture, each of the these is accessed by two seperate 8-bit registers as shown in the figure below. These registers are used to load timer count



8051 has a Timer Mode Register and Timer Countrol Register for selecteding a mode of operation and controlling purpose.

TMOD Register:

TMOD is an 8-bit register used to set timer mode of timer 0 and timer 1.



Its lower 4 bits are used for Timer 0 and the upper 4 bits

are used for Timer 1

BIT 7,3 - GATE :

1 = Enable Timer/Counter only when the INT0/INT1 pin is high and TR0/TR1 is set.

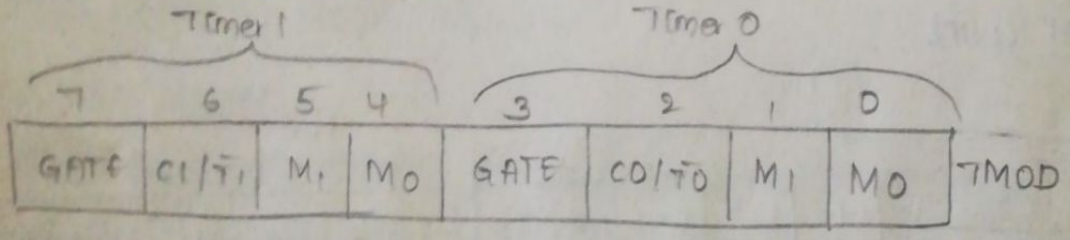
0 = Enable Timer/Counter when TR0/TR1 is reset.

BIT 6,2 - C/$\bar{T}$ (Counter/Timer) : ~~Timer or consumer~~

Timer or counter select bit

1 = Use as Counter ; 0 = Use as Timer

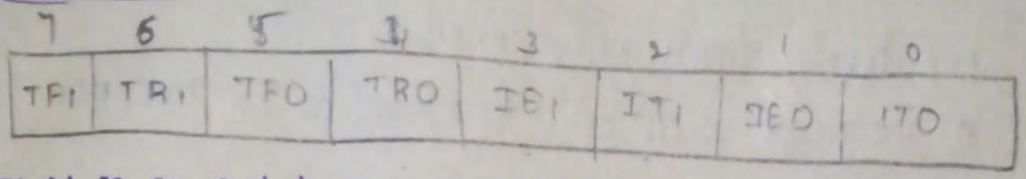
Bit 5:4 & 1:0 - M1: M0 : Timer/counter mode select bit



These are Timers/Counter mode select bit as per the below table

M1	M0	Mode	Operation
0	0	0 (13-bit timer mode)	13 bit timer/counter, 8 bit of THx & 5-bit of TLx
0	1	1 (16bit timer mode)	16 bit timer/counter, THx cascaded with TLx.
1	0	2 (8-bit auto-reload mode)	8 bit timer/counter (auto-reload mode), TLx related/reload with the value held by THx each time TLx overflow.
1	1	3 (split timer mode)	Split the 16 bit timer into two 8 bit timers i.e THx and TLx like two 8 bit timer

TCON Register:



TCON is an 8-bit control register and contains a timer and interrupt flags.

Bit 7 – TF1 : Timer 1 Overflow flag

1 = Timer 1 Overflow occurred (i.e Timer 1 goes to its max and roll over back to zero).

0 = Timer 1 overflow not occured.

It is cleared through software. In the Timer 1 overflow interrupt service routine, this bit will get cleared automatically while exiting from ISR.

Bit 6 – TR1 : Timer 1 Run Control Bit

1 = Timer 1 start

0 = Timer 1 stop It is set and cleared by software.

Bit 5 – TF0 : Timer 0 Overflow flag

1 = Timer 0 Overflow occurred (i.e Timer goes to its max and roll over back to zero.)

0 = Timer 0 overflow not occurred.

It is cleared through software. In the Timer 0 overflow interrupt service routine, this bit will get cleared automatically while exiting from ISR.

Bit 4 – TR0 : Timer 0 Run Control Bit

1 = Timer 0 start

0 = Timer 0 stop

It is, cleared by software.
set &

Bit 3 - IE1 : External Interrupt 1 Edge Flag

1 = External Interrupt 1 occurred

0 = Interrupt occur on a low level at the INT1 pin

It is set and cleared by software.

Bit 2 - IT1 : External Interrupt 1 Trigger Type Select Bit

1 = Interrupt occurs on falling edge at INT1 pin

0 = Interrupt occur on a lowlevel at the INT1 pin.

Bit 1 - IE0: External Interrupt 0 Edge flag.

1 = External Interrupt 0 occurred.

0 = External Interrupt 0 Proccessed

It is set and cleared by hardware.

Bit 0 - IT0: External Interrupt 0 Trigger Type Select Bit

1 = Interrupt occurs on falling edge at INT0 pin

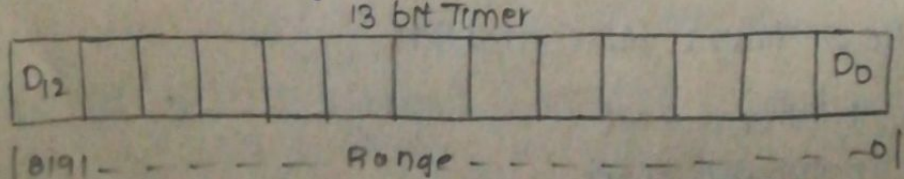
0 = Interrupt occur on a lowlevel at INT0 pin

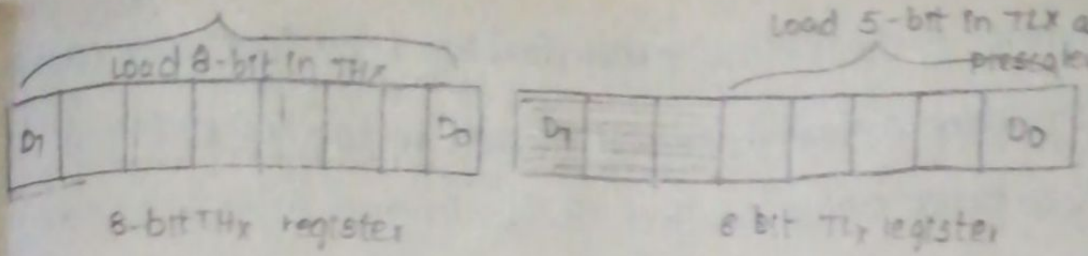
Modes of Operations:

Timer Modes : Timers have their operation modes which are selected in the TMOD register using M0 & M1 bit combinations.

Mode 0 (13-bit timer mode):

Mode 0 is a 13-bit timer mode for which 8-bit of THx and 5-bit of TLx are used. It is mostly used for interfacing possible with old MCS-48 family controllers.

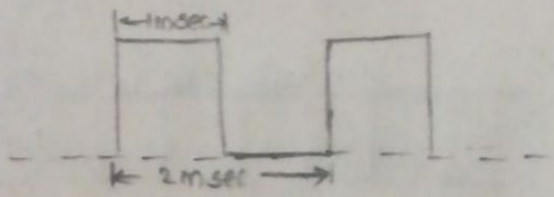




As shown in the figure, 8-bit of THx and lower 5-bit of TLx used to perform a total 13-bit timer. Higher 3 bits of TLx should be written as zero while using timer mode 0, or it will affect the result.

Example:

Let's generate a square wave of 2mSec period using an AT89C51 microcontroller with timer 0 a in mode 0 on the P1.0pin of port 1. Assume xtal oscillator frequency of 11.0592 MHz



As the xtal oscillator frequency is 11.0592 MHz we have a machine cycle of 1.085usec. Hence required

Timer clock frequency = (xtal osc. frequency) / 12

$= (11.0592\ MHz)/12 = 921.6\ KHz$

period $T = 1/(921.6\ kHz) = 1.085\ \mu s$.

Hence required count to generate a delay of 1 msec is

$$Count = (1 \times 10^{-3})/(1.085 \times 10^{-6}) \approx 921$$

The maximum count of Mode 0 is 2^13 (0-8191) and the Timer 0 can count will increment from 0 - 8191. So we need to load value which is 921 less from its maximum count i.e 8191.

Also, here in the below program, we need to an additional 13MC (machine cycles) from call to RET return of delay function. Hence value needed to be loaded is

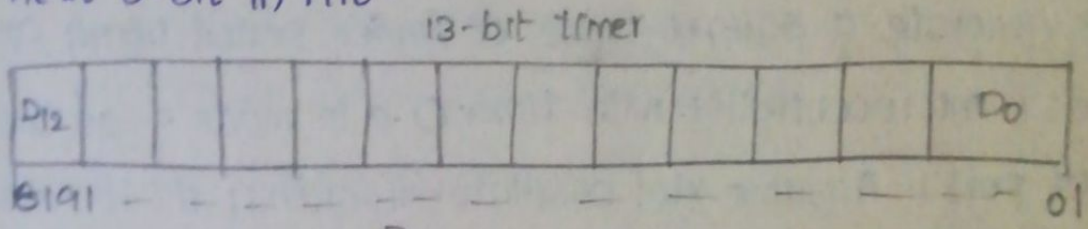
Value = (8191 - Count) + function_Mcycles + 1

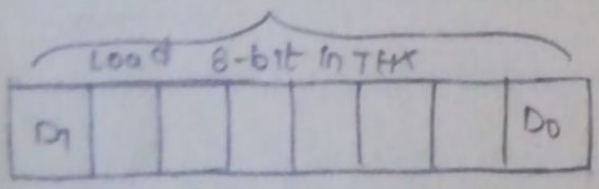
= 7284

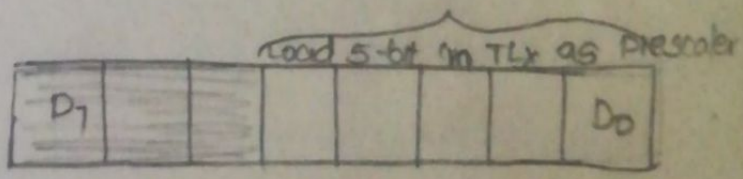
= 0x1C74

So we need to load 0x1C74 value in Timer 0.

1C74 = 0001 1100 0111 0100 b, now load lower 5-bit in TL0 and next 8-bit in TH0.







So here we get,

TL0 = 0001 0100 = 0x14 and TH0 = 1110 0011 = 0xE3.

TL0	0	0	0	1	0	1	0	0	14H
TH0	1	1	1	0	0	0	1	1	E3H

Programming Steps for delayfunction:

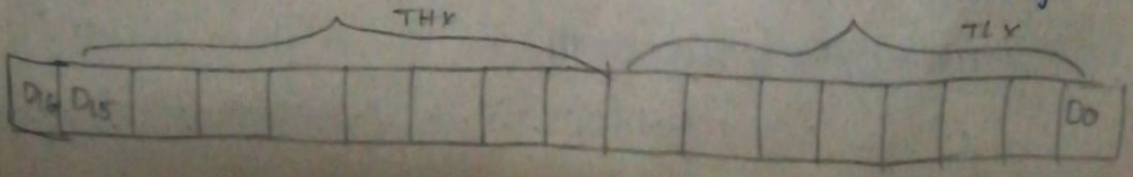
1. Load Timer register value i.e TMOD = 0x00 for Timer 0/1 mode 0 (13 bit timer mode).
2. Load calculated THx value i.e here TH0 = 0xE3.
3. Load calculated TLx value i.e here TL0 = 0x14
4. Start the timer by setting a TRx bit i.e here TR0 = 1
5. Poll TFx flag till it does not get set.
6. Stop the timer by clearing TRx bit i.e here TR0 = 0.
7. Clear timer flag TFx bit i.e here TF0 = 0
8. Repeat from step 1 to 7 for the delay again.

Program:

```
      MOV  TMOD , #00
AGAIN: MOV  TL0 , #14H
       MOV  TH0, #0E3H
       SETB TR0
BACK:  JNB  TF0,BACK
       CLR  TR0
       //CPL P1.5
       CLR  TF1
       SJMP AGAIN
```

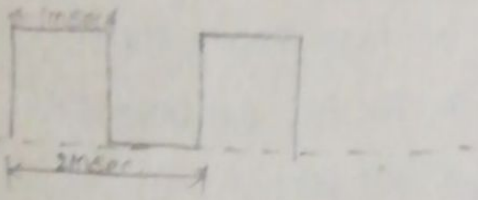
Mode 1 (16-bit timer mode):

Mode 1 is a 16 bit timer mode used to generate a delay, it uses 8-bit THx and 8 bit of TLx to form a total 16-bit register.



Example:

Let's generate a square wave of 2msec time period using an AT89C51 microcontroller with timer 0 in mode1 on the P1.0 pin of port 1. Assume xtal oscillator frequency of 11.0592MHz.



Timer clock frequency = (xtal osc. frequency)/12

= (11.0592 MHz)/12 = 921.6kHz

period T = 1/921.6kHz = 1.085 μs.

As the xtal oscillator frequency is 11.0592MHz we have a machine cycle of 1.085 μsec, Hence required count to generate a 1 msec delay is

$$\text{count} = (1\times10^{-3})/(1.085\times10^{-6}) \cong 921$$

The maximum count of mode 1 is 2^{16} (0 - 65535) & the timer 0 count will increment from 0-65535. So we need to load value which is 921 less from it max count i.e 65535 - 921.

Also here in the below program, we need an additional 13MC (machine cycles) from call to return of delay function. Hence value needed to be loaded is

value = (65535 - 921) + function-M cycles + 1

= 65535 - 921 + 13 + 1 = 64615 = (FC74)Hex

So we need to load FC74 Hex value higher byte in TH0 and lower byte in TL0 as FC74 = 1111 1100 0111 0100b

TH0 = 0XFC & TL0 = 0X74

TH0	1	1	1	1	1	1	0	0	FC
TL0	0	1	1	1	0	1	0	0	74

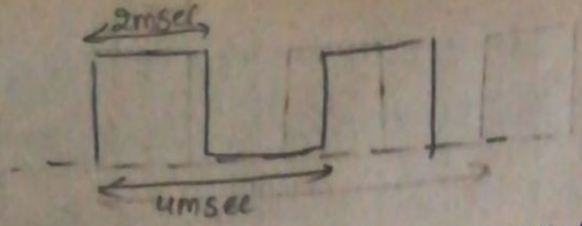
Programming Steps for delayfunction:

1. Load Tmod register value i.e TMOD=0x01 for Timer 0 mode 1 (16-bit timer mode)
2. Load calculated THx value i.e here TH0=0XFC
3. Load calculated TLx value i.e here TL0=0X74
4. Start the timer by setting a TRx bit i.e here TR0=1
5. Poll TFx flag till it does not get set.
6. Stop the Timer by clearing TRx bit i.e here TR0=0
7. Clear Timer flag TFx bit i.e here TF0=0
8. Repeat from step 1 to step 7 for the delay again.

Program:

```
        MOV TMOD , #01
AGAIN : MOV TL0 , #74H
        MOV TH0 , 0FCH
        SETB TR0
BACK :  JNB TF0, BACK
        CLR TR0
        CLR TF0
        SJMP AGAIN
```

* Generate a square wave of 1msec timer period using Timer 1 mode 0 of 8051 microcontroller. Assume xtal Oscillator frequency of 11.0592MHz.



Time clock frequency = xtal oscillator frequency / 12

= 11.0592 / 12 = 921.6 kHz

period = 1/921.6 kHz = 1.085 μs

count = $(2 \times 10^{-3}) / 1.085 \times 10^{-6} \approx$ ~~1843~~ 1842

Value = max count − Count required generate delay + Machine cycle required + 1

= 8191 − 1842 + 13 + 1

= 6363 = 18DB (Hex)

8191
14
8205
1842
6363

18DB hex value — ~~0001 1000 1101 1011~~

0001 1000 1101 1011

TL = 0001 1011 and TH = 1000 1001

= 1B = C6

TL_1	0	0	0	1	1	0	1	1	1B
TH_1	1	0	0	0	1	1	0	1	C6

```
    MOV TMOD, #10
AGAIN: MOV TL1, #1BH
       MOV TH1, #0C6H
       SETB TR1
BACK:  JNB TF1, BACK
       CLR TR1
       SJMP AGAIN
```

Generate a square wave of 200Hz frequency by using Timer 1 and mode 1 operation of 8051 microcontroller. Assume xtal(crystal) frequency of 16MHz.

$f = 200Hz$ → square frequency xtal $f = 16MHz$

delay $T = \frac{1}{f} = \frac{1}{200} = 0.05$

~~count~~

Given data freq = 200Hz

xtal fre = 16MHz

Timer 1, mode 1 ,

TL_1 , TH_1 = ?

Count value = max count - count required to generate delay + machine cyle required + 1

max count ⇒ de Mode 1 is a 16 bit mode

max count = $2^{16} = 65536 \rightarrow 65535$

count required to generate delay = $\frac{\text{delay time}}{\text{machine cycle time}}$

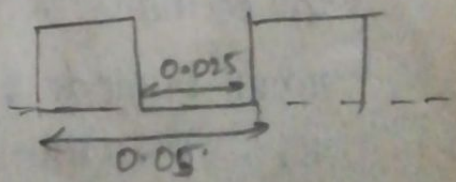
xtal = 16MHz $f = \frac{16}{12} = 1.3MHz$

machine time = $\frac{1}{f} = \frac{1}{1.3 \times 10^6}$

time = 0.76 Msec.

Total time period = $\frac{1}{f} = \frac{1}{200Hz} = 0.05$ sec.

count = $\frac{0.025}{0.76 \times 10^{-6}} \cong 32895$.



Total count = 65535 − 32895 + 1 + 1

= 32654 = 7F8E (Hex)

7F8E = 0111 1111 1000 1110

TL_1 = 8EH

TH_1 = 7FH

Programs:

```
        MOV  TMOD, #11
AGAIN:  MOV  TL1, #8EH
        MOV  TH1, 7FH
        SETB TR1
BACK:   JNB  TF1, BACK
        CLR  TR1
        CLR  TF1
        SJMP AGAIN
```

Mode 2 (8-bit auto reload timer mode):

Mode 2 is an 8-bit auto-reload timer mode. In this mode, we have to load the THx-8 bit value only. when the Timer gets started, THx value gets automatically loaded into the TLx and THx

* Generate a square wave with 200 μsec. Using timer 1 mode 2 operation assume xtal oscilator frequency 11.0592 MHz.

Given data Total time = 200 μsec

x-tal oscillator freq = 11.0592 MHz

timer 1, mode 2

TLx = ?, THx = ?

delay time = 100 μsec.

Time freq = Xtal/12 = $\frac{11.0592}{12}$ = 1.085 x 10^-6

= 921

Time period = $\frac{1}{921.6} = 1.085 \times 10^{-6}$

delay count = $\frac{\text{delay time}}{\text{m/c time}} = \frac{100 \times 10^{-6}}{1.085 \times 10^{-6}}$

$= 92 (D) = 5C (Hex)$

Count = 255 − 92H0 + 1 = 164 (D) = A4H

TL1 = A4H

TH1 = A4H

Program:

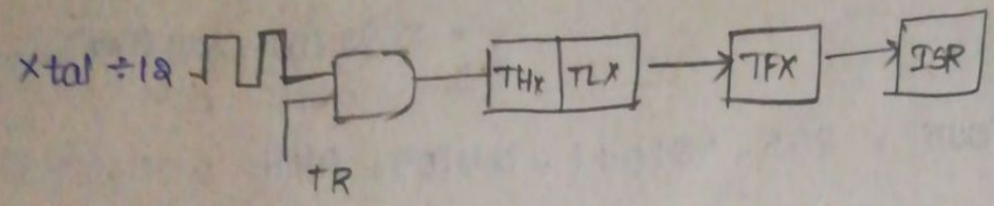
```
      MOV TMOD , #12
AGAIN: MOV TL1 , #A4H
       MOV TH1 , #A4H
       SETB TR1
BACK:  JNB TF1 , BACK
       CLR TR1
       CLR TF1
       SJMP AGAIN
```

Timer 0 Interrupts in 8051 (Interrupt Programming)

8051 has two timer interrupts assigned with different vector address. When Timer count rolls over from its max value to 0, it sets the timer flag TFx. This will interrupt the 8051 microcontroller to serve ISR (interrupt service routine) if global and timer interrupt is enabled.

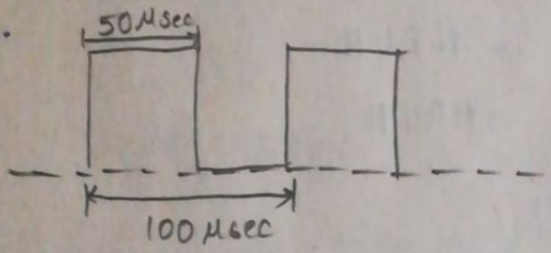
Interrupt source	Vector address
Timer 0 overflow (TF0)	000BH
Timer 1 overflow (TF1)	001BH

The 8051 microcontroller jumps directly to the vector address on the occurrence of a corresponding interrupt.



Example: wave

Generate a square of 10kHz on PORT1.0 using Timer 0 interrupt. We will use Timer 0 in mode 1 with 11.0592MHz oscillator frequency.



Time frequency = $\frac{11.0592}{12} = 921$

Time period = $\frac{1}{921} = 1.085 \times 10^{-6}$

delay count = $\frac{\text{delay time}}{\text{m/c time}} = \frac{50 \times 10^{-6}}{1.085 \times 10^{-6}}$

$= 46$

count = 65535 - 46 + 1 = 65535 - 47 + 0 + 1

= 65489 (D) = FFD1 (Hex)

FFD1 = 1111 1111 1101 0001

TL_0 = 1101 0001 (D1)

TH_0 = 1111 1111 (FF)

92
100×2
$\frac{92}{2} = 46$

65535
47
65489

Program:

```
      MOV TMOD , #01
AGAIN: MOV TL0 , #01H
       MOV TH0 , #0FFH
       SETB TR1
BACK : JNB TF1 , BACK
       CLR TR1
       CLR TF1
       SJMP AGAIN
```

* Generate a square wave of frequency 100Hz using timer 1 interrupts. We will use Timer1 mode 0 of 8051 microcontroller with external oscillator frequency of 11.0592 MHz.

Sol:- Given freq = 100Hz

oscillator frequency = 11.0592 MHz

$$\text{Time freq} = \frac{11.0592\text{MHz}}{12} = 921$$

$$\text{Time period} = \frac{1}{921} = 1.085 \times 10^{-6}$$

$$\text{Delay count} = \frac{\text{delay time}}{\text{m/c time}} = \frac{5\text{ msec}}{1.085 \times 10^{-6}}$$

$$= 4608$$

Max count = 65535 − 4608 + 0 + 1 = 8191 − 4608 + 0 + 1

= 65536 − 4608 = 8192 − 4608

= 3584

65536
4600

8192
4608

3584

Program:

```
      MOV TMOD,#10
AGAIN: MOV TL, #
       MOV TH ,#
       SETB TR
BACK : JNB TF, BACK
       CLR TR1
       CLR TF1
       SJMP AGAIN
```

I/O Port Programming:

In 8051 there are total 40 pins in that 32 pins are set asisged for four ports P_0, P_1, P_2 and P_3 where each port takes 8 pins.

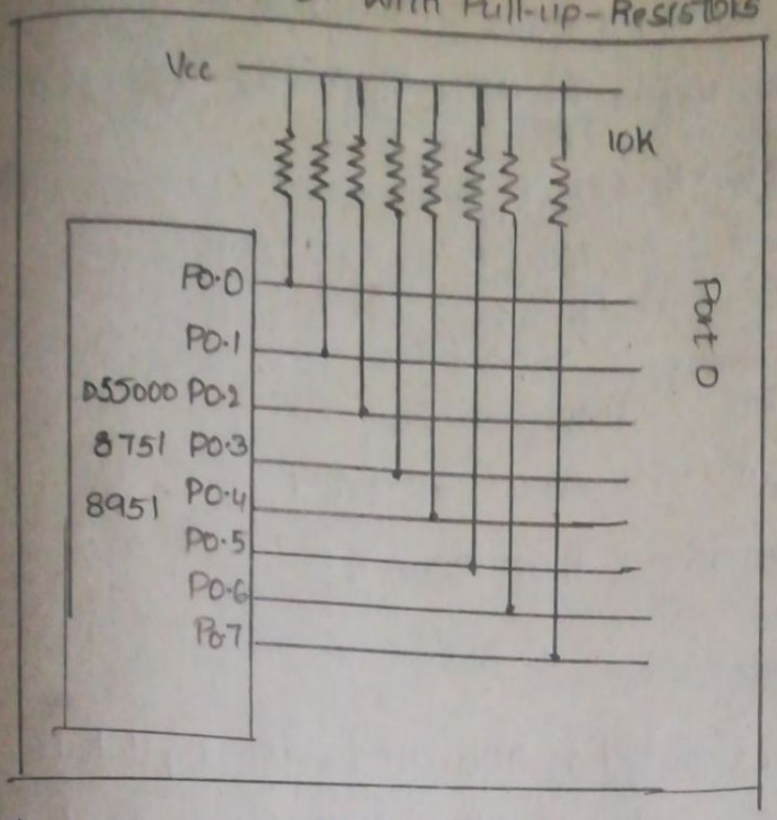
I/o port pins and their functions:

The four ports P_0, P_1, P_2 and P_3 each use 8 pins, making them as 8-bit ports. All the ports upon reset can be configured as output, ready to be used as output ports. To use any of these ports as an input port, it must be programmed.

Port 0:

Port 0 occupies a total of 8 pins (pins 32-39). It can be used for input or output. To use pins of port 0 as both input and output ports, each pin must be connected externally to a 10k ohm pull-up resistor. This is due to the fact that P_0 is an open drain, unlike P_1, P_2 and P_3.

Port 0 with Pull-up-Resistors



The pull-up-resistors are connected to the Port 0 asshown in above figure. In this way we take advantage of port 0 for both input and output ports

* With external pull-up-resistor connected upon RESET (reset) configured as an output port.

Example Send out to P0 the alternating values of 55H and AAH

```
        MOV  A,#55H  ;  move 55H to A
BACK :  MOV  P0,A    ;  move A to P0.
        ACALL DELAY  ;
        CPL   A      ;
        SJMP  BACK
```

Port 0 as input:

With resistors connected to port 0, in order to make it an input, port must be programmed by writing 1 to all the bits.

Ex: Write a down a small program to configure port 0 as input port and then receive the data then send it to port 1

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

Port 0 as dual role:

- Port 0 is also designated as AD0 - AD7, allowing it to be used for both address and data. When connecting an 8051 to an external memory, port 0 provides both address and data.

Port 1 :

Port 1 occupies a total of 8 pins (same as port 0). Port 1 can be used as input or output port. In constrat to port 0, this port does not need any pull-up resistors since it ~~has~~ already has pull-up-resistors internally. Upon reset port 1 is configured as an output port.

Ex: Write a program to continuosly send out to port 1 the alternating values of 55H and AAH.

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

Port 1 as Input Port:

To make port 1 as input port, it must programmed as such by writing 1 to all its bits.

Ex: Write a program to configure port 1 as an input port then receive data and send that data to R_7, R_6 and R_5 register.

```
MOV A, #0FFH
MOV P1, A
MOV A, P1      ; get data from P1
MOV R7, A      ; save it in reg R7
MOV R6,
ACALL DELAY    ; wait
MOV A, P1      ; get another data from P1
MOV R6, A      ; save it in R6
ACALL DELAY    ; wait
MOV A, P1      ; get from P1
MOV R5, A      ; save it in R5
```

Port 2:

Port 2 occupies of a total of 8 pins. It can be used as input port or output port. Just like Port 1, Port 2 does not need any pull-up resistors since it already has pull-up-resistors internally. Upon port 2 is configured as an output port.

Ex: Write a program to continuesly send out to port 2. The alternating values of 55H and AAH.

```
      MOV A, #55H
BACK: MOV P2, A
      ACALL DELAY
      CPL A
      SJMP BACK
```

Port 2 as Input Port:

To make port 2 as input port, it must programmed as such by writing 1 to all its bits. In the following code, port 2 is configured first as an input port

Write a program to configure port 2 as input port and receive the data & send that data to Port 1 continuously.

```
        MOV A, #FFH
BACK:   MOV P2, A

BACK:   MOV A, P2
        MOV P1, A
        SJMP BACK
```

Dual role of Port 2:

In system based on the 8051, 80sh, 89C51 and DS5000 P2 is used as simple I/O. However in 8051-based systems, port 2 must be used along with P0 to provide the 16 bit address for the external memory.

* Port 2 is also designated as A8-A15, indicating its dual function. Since 8051 is capable of accessing 64K bytes of external memory, it needs a path for the 16 bits of the address
* while the P0 provides the lower 8 bits via A0-A7, it is the job of P2 to provide bits A18-A15 of the address location of memory

Port 3 :

For 8051 Microcontroller we have 3 ports P_0, P_1 and P_2 for I/O port operations. These 3 ports should be enough for most microcontroller applications. That leaves port 3 for interrupts as well as other signals, ~~as we w~~

* Port 3 occupies a total of 8 pins, pins 10 through 17. It can be used as input or output. P_3 does not need any pull-up-resistors, the same as P_1 and P_2 did not.
* Although port 3 is configured as an output port upon reset, this is not the way it is most commonly used.
* Port 3 has the addition-function of providing some extremely important signals such as interrupts, read/write signals, data receiving and transmitting signals & Timer control signals.

Port 3 : Alternate Functions

P_3 bit	Function	Pin
$P_{3.0}$	RxD	10
$P_{3.1}$	TxD	11
$P_{3.2}$	$\overline{INT0}$	12
$P_{3.3}$	$\overline{INT1}$	13
$P_{3.4}$	T0	14
$P_{3.5}$	T1	15
$P_{3.6}$	$\overline{WR}$	16
$P_{3.7}$	$\overline{RD}$	17