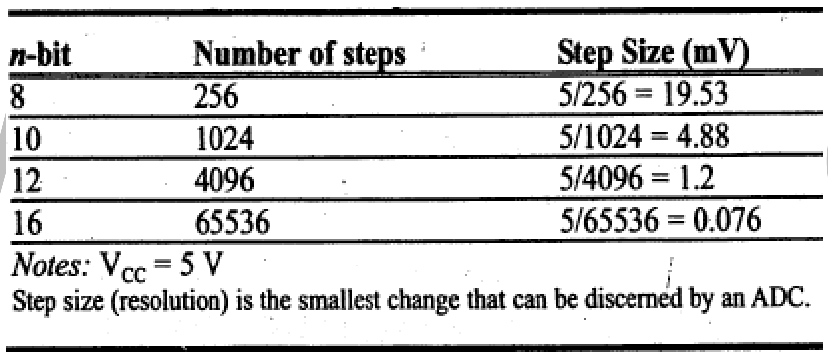
5.5 ADC, DAC AND SENSOR INTERFACING

ANALOG-TO-DIGITAL CONVERTER (ADC) INTERFACING:

- ADCs (Analog-to-Digital Converters) are among the most widely used devices for data acquisition.
- A physical quantity, like temperature, pressure, humidity, and velocity, etc., is converted to electrical (voltage, current) signals using a device called a transducer or sensor
- We need an Analog-to-Digital Converter to translate the analog signals to digital numbers, so microcontroller can read and process them.
- An ADC has n-bit resolution where n can be 8, 10, 12, 16 or even 24 bits.
- The higher-resolution ADC provides a smaller step size, where step size is the smallest change that can be discerned by an ADC. This is shown in Table 5.5.1.



n-bit	Number of steps	Step Size (mV)
8	256	5/256 = 19.53
10	1024	5/1024 = 4.88
12	4096	5/4096 = 1.2
16	65536	5/65536 = 0.076

Notes: $V_{CC} = 5$ V
Step size (resolution) is the smallest change that can be discerned by an ADC.

Table 5.5.1 Resolution vs. Step Size for ADC

[Source: "The 8051Microcontroller and Embedded Systems: Using Assembly and C" by Mohamed Ali Maida, Janice Gillespie Maida, Rollin McKinley]

In addition to resolution, conversion time is another major factor in judging an ADC.

- Conversion time is defined as the time it takes the ADC to convert the analog input to a digital (binary) number.
- In parallel ADC, we have 8 of more pins dedicated to bringing out the binary data, but in serial ADC we have only one pin for data out.

ADC804 CHIP:

ADC804 IC is an 8-bit parallel analog-to-digital converter.

➢ It works with +5 volts and has a resolution of 8bits.

➢ In ADC804 conversion time varies depending on the clocking signals applied to the CLK R and CLK IN pins, but it cannot be faster than 110μs.

➢ Figure 5.5.1 is the Pin out of ADC0804 in free running mode.

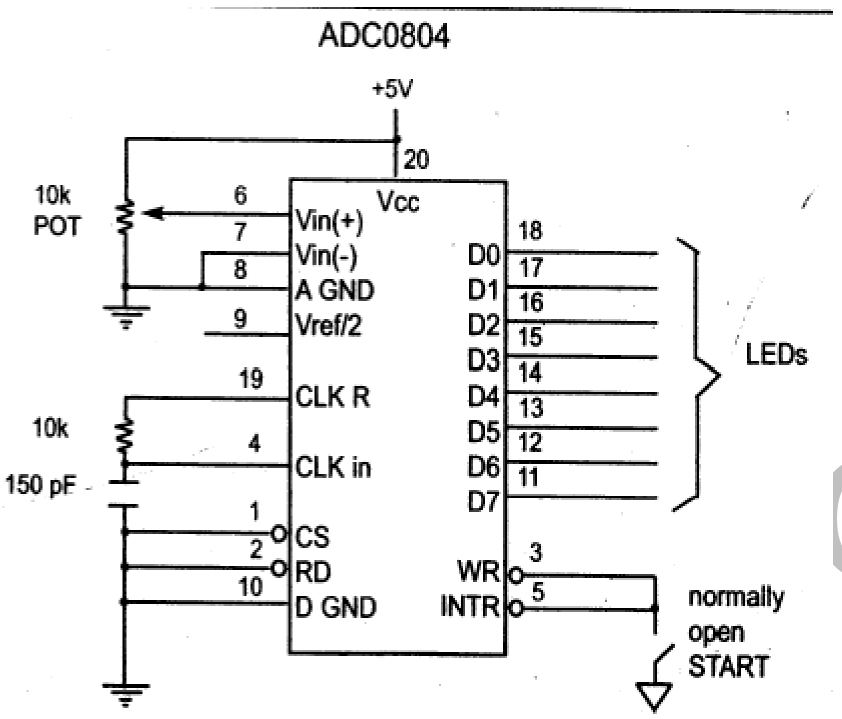


Figure 5.5.1 ADC0804 Chip (Testing ADC0804 in Free Running Mode)

[Source: "The 8051Microcontroller and Embedded Systems: Using Assembly and C" by Mohamed Ali Maida, Janice Gillespie Maida, Rollin McKinley, pg.no.375]

CLK IN is an input pin connected to an external clock source when an external clock is used for timing.

• However, the 0804 has an internal clock generator.

• To use the internal clock generator (also called self-clocking), CLK IN and CLK R pins are connected to a capacitor and a resistor and the clock frequency is determined by:

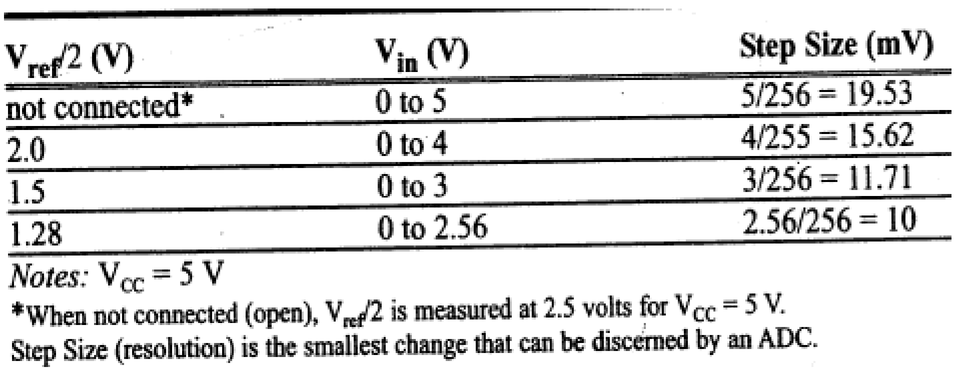
$$f = \frac{1}{1.1\,RC}$$

Verve/2: (Pin 9)

• It is used for the reference voltage.

• If this pin is open (not connected), the analog input voltage is in the range of 0 to 5 volts (the same as the Vic pin).

• If the analog input range needs to be 0 to 4 volts, Verve/2 is connected to 2 volts. Table 5.5.2 shows the VIN range for various Verve/2 inputs.



$V_{ref}/2$ (V)	V_{in} (V)	Step Size (mV)
not connected*	0 to 5	5/256 = 19.53
2.0	0 to 4	4/255 = 15.62
1.5	0 to 3	3/256 = 11.71
1.28	0 to 2.56	2.56/256 = 10

Notes: V_{CC} = 5 V
*When not connected (open), $V_{ref}/2$ is measured at 2.5 volts for V_{CC} = 5 V.
Step Size (resolution) is the smallest change that can be discerned by an ADC.

Table 5.5.2 Verve/2 Relation to VIN Range (ADC0804)

[Source: "The 8051Microcontroller and Embedded Systems: Using Assembly and C" by Mohamed Ali Maida, Janice Gillespie Maida, Rollin McKinley]

D0-D7 are the digital data output pins.

• These are tri-state buffered and the converted data is accessed only when CS =0 and RD is forced low.

• To calculate the output voltage, use the following formula

$$D_{out} = \frac{V_{in}}{step\ size}$$

Where Doubt = digital data output (in decimal), VIN = analog voltage, and Step size (resolution) is the smallest change, which is (2 * Vref/2)/256 for ADC 0804

Analog ground is connected to the ground of the analog Vin and digital ground is connected to the ground of the Vcc pin.

• The reason that to have ground pin is to isolate the analog VIN signal from transient voltages caused by digital switching of the output D0 – D7. This contributes to the accuracy of the digital data output.

- Differential analog inputs where VIN= VIN (+) – VIN (-).

• **VIN (-)** is connected to ground and **VIN (+)** is used as the analog input to be converted.

The ADC converts the analog input to its binary equivalent and holds it in an internal register.

• **RD** is used to get the converted data out of the ADC0804 chip.

• IF "output enable", a high-to-low RD pulse is used to get the 8-bit converted data out of ADC804.

INTR is "end of conversion" .When the conversion is finished; it goes low to signal the CPU that the converted data is ready to be picked up.

WR is an active low input

• It is "start conversion" When WR makes a low-to-high transition, ADC804 starts converting the analog input value of VIN to an 8-bit digital number.

• When the data conversion is complete, the INTR pin is forced low by the ADC0804.

STEPS TO BE FOLLOWED FOR DATA CONVERSION:

The following steps must be followed for data conversion by the ADC804 chip:

- Make CS= 0 and send a L-to-H pulse to pin WR to start conversion.
- Monitor the INTR pin, if high keep polling but if low, conversion is complete, go to next step.
- Make CS= 0 and send a H-to-L pulse to pin RD to get the data out.

The Read and Write Timing for ADC0804 is shown in Figure 5.5.2

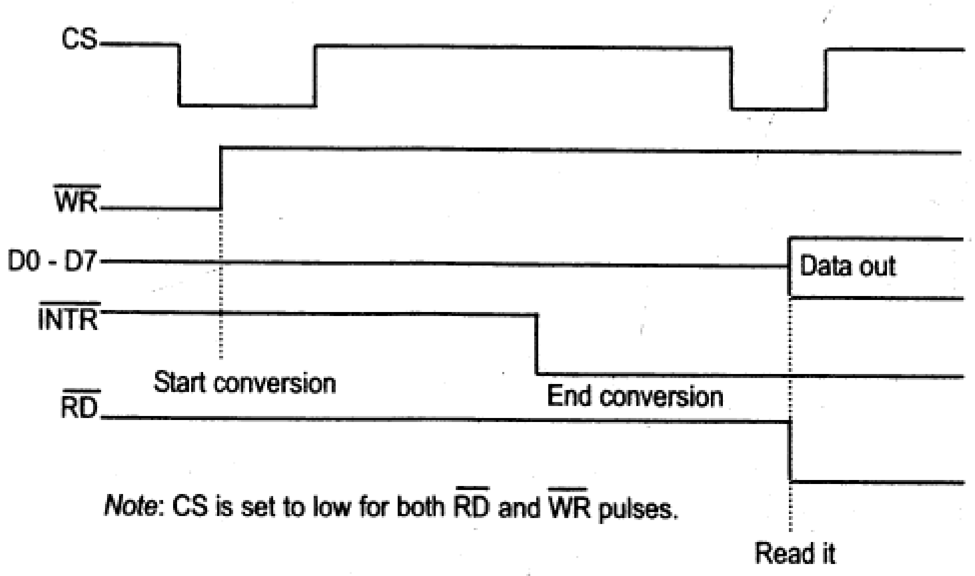


Figure 5.5.2 Read and Write Timing for ADC0804

[Source: "The 8051Microcontroller and Embedded Systems: Using Assembly and C" by Mohamed Ali Maida, Janice Gillespie Maida, Rollin McKinley, pg.no.377]

8051 Connection to ADC0804 with Self-Clocking is shown in Figure 5.5.3.

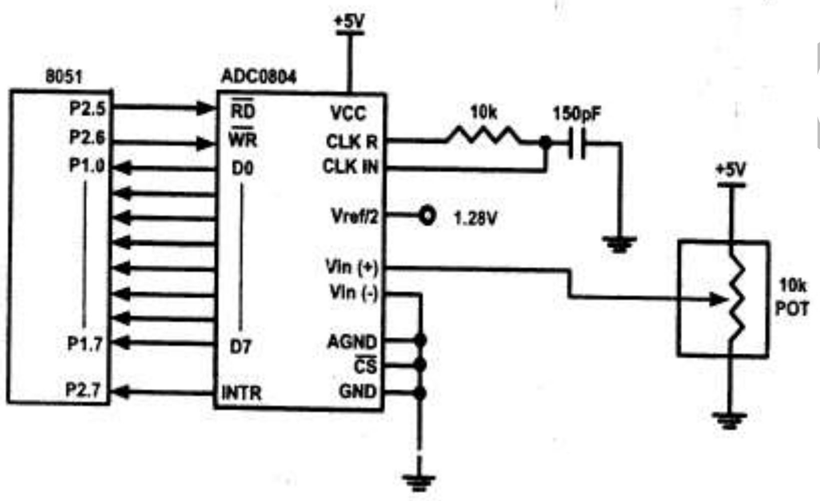


Figure 5.5.3 8051 Connection to ADC0804 with Self-Clocking

[Source: "The 8051Microcontroller and Embedded Systems: Using Assembly and C" by Mohamed Ali Maida, Janice Gillespie Maida, Rollin McKinley, pg.no.378]

Example:

Write a program to monitor the INTR pin and bring an analog input into register A. Then call a hex-to ACSII conversion and data display subroutines. Do this continuously.

; p2.6=WR (start conversion needs to L-to-H pulse)

; p2.7 when low, end-of-conversion)

```
; p2.5=RD (a H-to-L will read the data from ADC chip)
; p1.0 – P1.7= D0 - D7 of the ADC0804

MOV P1, #0FFH                       ; make P1 = input
BACK: CLR P2.6                      ; WR = 0

SETB P2.6                           ; WR = 1 L-to-H to start conversion
HERE: JB P2.7, HERE                 ; wait for end of conversion
CLR P2.5                            ; conversion finished, enable RD
MOV A, P1                           ; read the data
ACALL CONVERSION                    ; hex-to-ASCII
conversion ACALL DATA_DISPLAY          ; display the data
SETB P2.5                           ; make RD=1 for next
round SJMP BACK
```

DIGITAL-TO-ANALOG (DAC) CONVERTER:

The two method of creating a DAC is binary weighted and R/2R ladder.

The Binary Weighted DAC, which contains one resistor or current source for each bit of the DAC connected to a summing point. These precise voltages or currents sum to the correct output value. This is one of the fastest conversion methods but suffers from poor accuracy because of the high precision required for each individual voltage or current. Such high-precision resistors and current-sources are expensive, so this type of converter is usually limited to 8-bit resolution or less.

The R-2R ladder DAC, which is a binary weighted DAC that uses a repeating cascaded structure of resistor values R and 2R. This improves the precision due to the relative ease of producing equal valued matched resistors (or current sources). However, wide converters perform slowly due to increasingly large RC-constants for each added R-2R link.

The first criterion for judging a DAC is its resolution, which is a function of the number of binary inputs. The common ones are 8, 10, and 12 bits. The number of data bit inputs decides the resolution of the DAC since the number of analog output levels is

Equal to 2n, where n is the number of data bit inputs. Therefore, an 8-input DAC such as the DAC0808 provides 256 discrete voltage (or current) levels of output. Similarly, the 12-bit DAC provides 4096 discrete voltage levels. There also 16-bit DACs, but they are more expensive.

DAC 0808:

➢ The digital inputs are converter to current (I out), and by connecting a resistor to the Iout pin, we can convert the result to voltage.

➢ The total current provided by the I out pin is a function of the binary numbers at the D0-D7 inputs of the DAC0808 and the reference current (Iref), and is as follows

$$I_{out} = I_{ref}\left(\frac{D7}{2}+\frac{D6}{4}+\frac{D5}{8}+\frac{D4}{16}+\frac{D3}{32}+\frac{D2}{64}+\frac{D1}{128}+\frac{D0}{256}\right)$$

Usually reference current is 2mA.

➢ Ideally, we connect the output pin to a resistor, convert this current to voltage, and monitor the output on the scope.

➢ But this can cause inaccuracy; hence an opamp is used to convert the output current to voltage.

➢ The 8051 connection to DAC0808is as shown in the below Figure 5.5.4.

➢ Now assuming that Ire = 2mA, if all the inputs to the DAC are high, the maximum output current is 1.99mA.

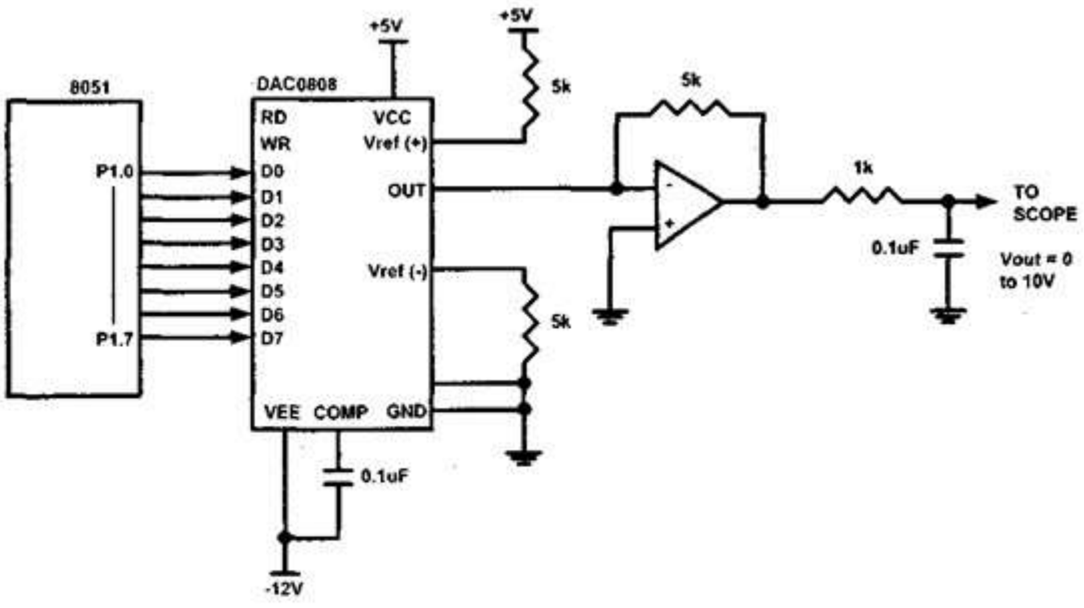


Figure 5.5.4 8051 Connection to DAC808

[Source: "The 8051Microcontroller and Embedded Systems: Using Assembly and C" by Mohamed Ali Maida, Janice Gillespie Maida, Rollin McKinley, pg.no.399]

Example 1:

Assuming that R=5K and Ire=2mA, calculate Vought for the following binary inputs:

(a) 10011001B

(b) 11001000B

Solution:

(a) I out = 2mA(153/256) = 1.195mA and Vought = 1.195mA * 5K

=5.975V (b) I out = 2mA(200/256) = 1.562mA and Vought = 1.562mA *

5K =7.8125V

CONVERTING IOUT TO VOLTAGE IN DAC0808:

➢ Ideally we connect the output pin lout, to a resistor, convert this current to voltage, and monitor the output on the scope.

➢ In real life, however, this can cause inaccuracy since the input resistance of the load where it is connected will also affect the output voltage.

➢ For this reason, the left current output is isolated by connecting it to an op-amp such as the 741 with Raff = 5K ohms for the feedback resistor.

➢ Assuming that R= 5K ohms, by changing the binary input, the output voltage changes as shown in Example 2.

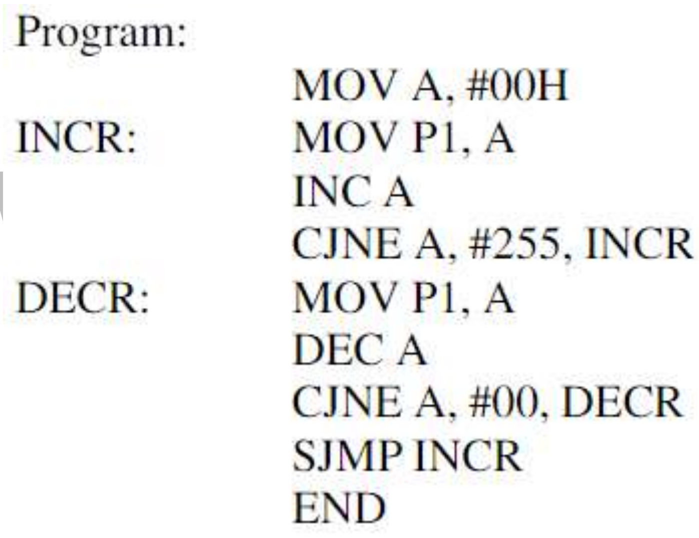
Example 2:

In order to generate a stair-step ramp, set up the circuit in Figure 5.5.4 and connect the output to an oscilloscope. Then write a program to send data to the DAC to generate a stair-step ramp.

```
CLR A
AGAIN: MOV P1, A          ; SEND DATA TO DAC
INC A                     ; COUNT FROM 0 TO FFH
ACALL DELAY               ; LET DAC RECOVER
SJMP AGAIN
```

Example 3:

Write an ALP to generate a triangular waveform.

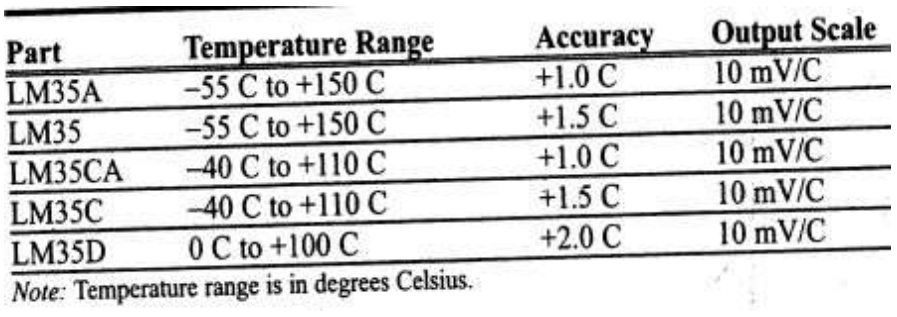


```
Program:
            MOV A, #00H
INCR:       MOV P1, A
            INC A
            CJNE A, #255, INCR
DECR:       MOV P1, A
            DEC A
            CJNE A, #00, DECR
            SJMP INCR
            END
```

SENSOR INTERFACING:

LM35 TEMPERATURE SENSORS:

- The LM35 series sensors are precision integrated-circuit temperature sensors whose output voltage is linearly proportional to the Celsius (centigrade) temperature.
- The LM35 requires no external calibration since it is internally calibrated.
- It outputs 10mV for each degree of centigrade temperature.



Part	Temperature Range	Accuracy	Output Scale
LM35A	−55 C to +150 C	+1.0 C	10 mV/C
LM35	−55 C to +150 C	+1.5 C	10 mV/C
LM35CA	−40 C to +110 C	+1.0 C	10 mV/C
LM35C	−40 C to +110 C	+1.5 C	10 mV/C
LM35D	0 C to +100 C	+2.0 C	10 mV/C

Note: Temperature range is in degrees Celsius.

Table 5.5.3 LM35 Temperature Sensor Series Selection Guide

[Source: "The 8051Microcontroller and Embedded Systems: Using Assembly and C" by Mohamed Ali Maida, Janice Gillespie Maida, Rollin McKinley]

- The sensors of the LM34 series are precision integrated-circuit temperature sensors whose output voltage is linearly proportional to the Fahrenheit temperature.
- It outputs 10mV for each degree Fahrenheit temperature.

SIGNAL CONDITIONING AND INTERFACING THE LM35 TO THE 8051

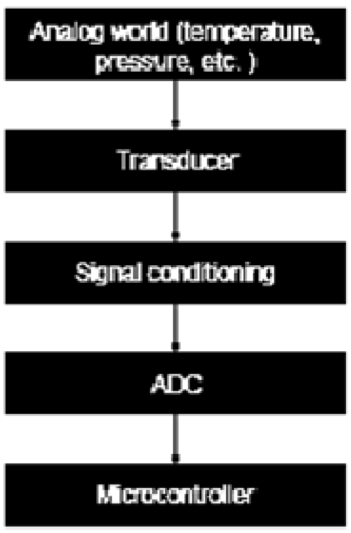


Figure 5.5.5 Getting Data from Analog World

[Source: "The 8051Microcontroller and Embedded Systems: Using Assembly and C" by Mohamed Ali Maida, Janice Gillespie Maida, Rollin McKinley]

- Signal conditioning is widely used in the world of data acquisition.
- The most common transducers produce an output in the form of voltage, current, charge, capacitance, and resistance.
- However, we need to convert these signals to voltage in order to send input to an A-to-D converter.

This conversion (modification) is commonly called signal conditioning.

➢ Signal conditioning can be a current-to-voltage conversion or a signal amplification.

➢ For example, the thermistor changes resistance with temperature.

➢ The change of resistance must be translated into voltages in order to be of any use to an ADC.

➢ Look at the case of connecting an LM35 to an ADC0848.

➢ Since the ADC0848 has 8-bit resolution with a maximum of 256 (28) steps and the LM35 (or LM34) produces l0 mV for every degree of temperature change, we can condition VIN of the ADC0848 to produce a Vought, of 2560 mV (2.56 V) for full-scale output.

➢ Therefore, in order to produce the full-scale Vought of 2.56 V for the ADC0848, we need to set Verve = 2.56.

➢ This makes Vought, of the ADC0848 correspond directly to the temperature as monitored by the LM35. Refer the Table 5.5.4

Temp. (C)	V_{in} (mV)	V_{out} (D7 - D0)
0	0	0000 0000
1	10	0000 0001
2	20	0000 0010
3	30	0000 0011
10	100	0000 1010
30	300	0001 1110

Table 5.5.4 Temperature vs. Vought for ADC0848

[Source: "The 8051Microcontroller and Embedded Systems: Using Assembly and C" by Mohamed Ali Maida, Janice Gillespie Maida, Rollin McKinley]

Figure 5.5.5 shows the connection of a temperature sensor to the ADC0848.

➢ The LM336-2.5 sneer diode to fix the voltage across the 10K pot at 2.5V.

➢ The use of the LM336-2.5 should overcome any fluctuations in the power supply.

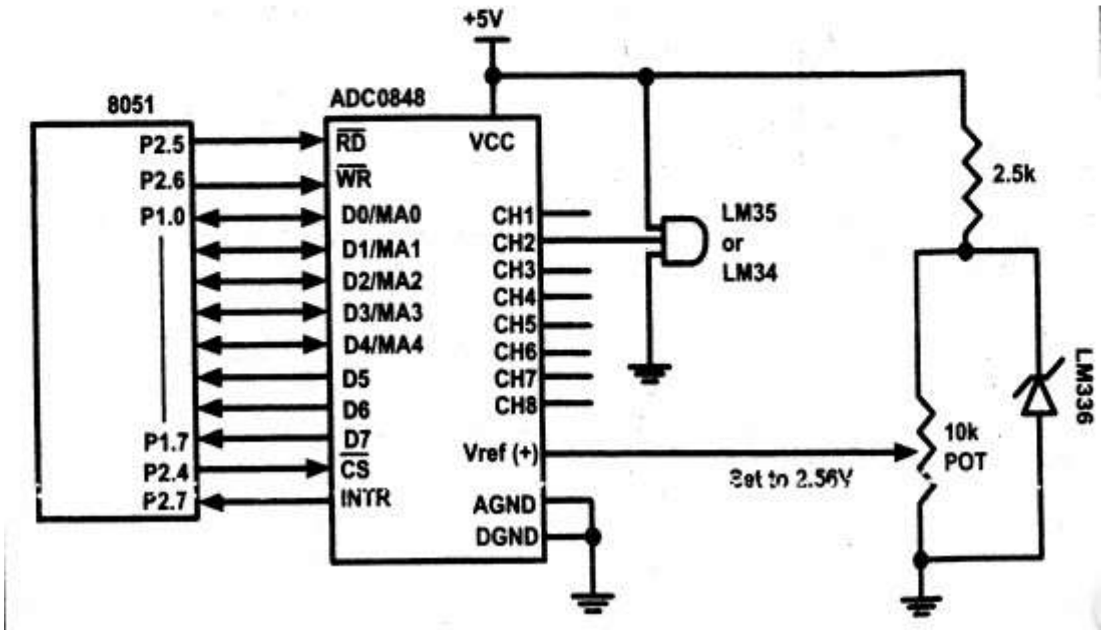


Figure 5.5.6 8051 Connection to ADC0848 and Temperature sensor

[Source: "The 8051Microcontroller and Embedded Systems: Using Assembly and C" by Mohamed Ali Maida, Janice Gillespie Maida, Rollin McKinley, pg.no.405]

Program:

```
RD BIT P2.5                   ; RD
WR BIT P2.6                   ; WR
INTR BIT P2.7                 ; END OF CONVERSION

MYDATA EQU P1; P1.0-P1.7 = D0-D7 OF THE ADC0848
MOV P1, #0FFH                 ; make P1 =
input SETB INTR
BACK: CLR WR                  ; WR = 0
SETB WR                       ; WR = 1 L-to-H to start
conversion HERE: JB INTR, HERE  ; wait for end of conversion
CLR RD                        ; conversion finished, enable RD
MOV A, MYDATA                  ; read the data
ACALL CONVERSION              ; hex-to-ASCII
conversion ACALL DATA_DISPLAY       ; display the
data
SETB RD                       ; make RD=1 for next
round SJMP BACK
```

CONVERSION:

```
MOV B,#10
DIV AB
MOV R7,B
MOV B,#10
DIV AB
MOV R6,B
MOV R5,A
RET
DATA_DISPLAY:
MOV P0, R7
ACALL DELAY
MOV P0, R6
ACALL DELAY
MOV P0, R5
ACALL DELAY
RET
```

5.6 EXTERNAL MEMORY INTERFACE

EXTERNAL ROM (PROGRAM MEMORY) INTERFACING

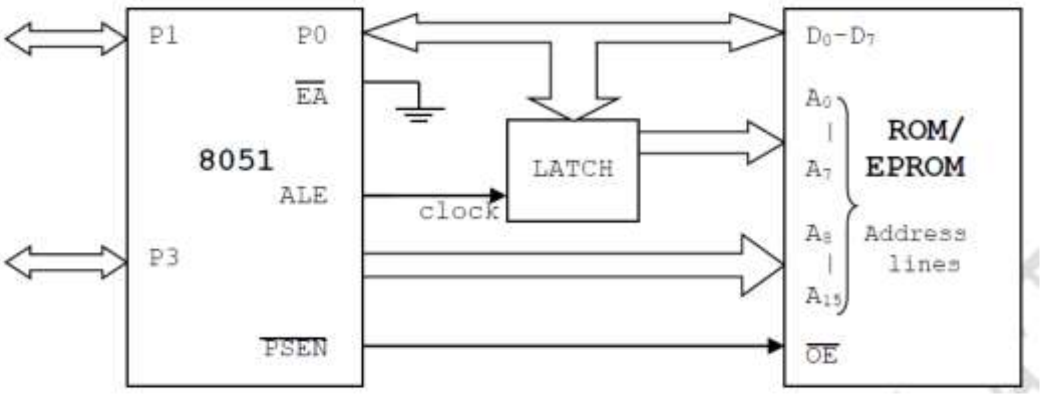


Figure 5.6.1 Interfacing of ROM/EPROM to 8051

[Source: "The 8051Microcontroller and Embedded Systems: Using Assembly and C" by Mohamed Ali Maida, Janice Gillespie Maida, Robin McKinley]

Figure 5.6.1 shows how to access or interface ROM to 8051.

- Port 0 is used as multiplexed data & address lines. It gives lower order (A7-A0) 8 bit address in initial T cycle & higher order (A8-A15) used as data bus.
- 8 bit address is latched using external latch & ALE signal from 8051.
- Port 2 provides higher order (A15-A8) 8 bit address.
- PSEN is used to activate the output enable signal of external ROM/EPROM.

EXTERNAL RAM (DATA MEMORY) INTERFACING

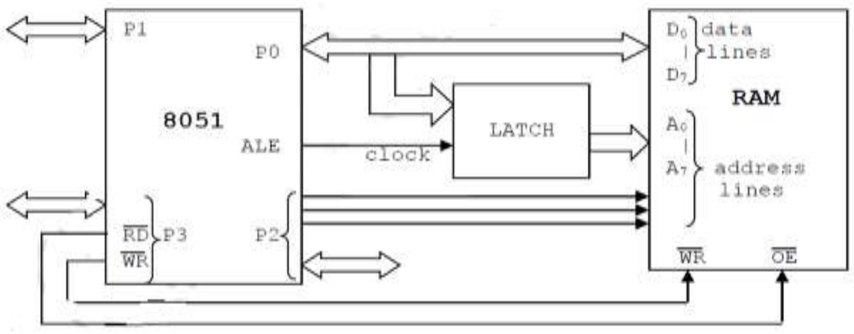


Figure 5.6.2 Interfacing of RAM to 8051

[Source: "The 8051Microcontroller and Embedded Systems: Using Assembly and C" by Mohamed Ali Maida, Janice Gillespie Maida, Robin McKinley]

Figure 5.6.2 shows how to connect or interface external RAM (data memory) to 8051.

- Port 0 is used as multiplexed data & address lines.
- Address lines are decoded using external latch & ALE signal from 8051 to provide lower order (A7-A0) address lines.
- Port 2 gives higher order address lines.
- RD & WR signals from 8051 selects the memory read & memory write operations respectively.

Noted & WR signals: generally P3.6 & P3.7 pins of port 3 are used to generate memory read and memory write signals. Remaining pins of port 3 i.e. P3.0-P3.5 can be used for other functions.

Solved Examples:

Example 1: Design a controller system using 8051 to Interface the external RAM of size 16k x 8.

Solution: Given, Memory size: 16k

Which means, we require 2^n=16k: n address lines

- Here n=14: A0 to A13 address lines are required.
- A14 and A15 are connected through OR gate to CS pin of external RAM.
- When A14 and A15 both are low (logic '0'), external data memory (RAM) is selected.

Address Decoding (Memory Map) For 16k X 8 Ram

Address	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	Hex Addr
Start	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000 H
End	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	3FF FH

Figure 5.6.3 shows interfacing of 16k x 8 RAM to 8051.

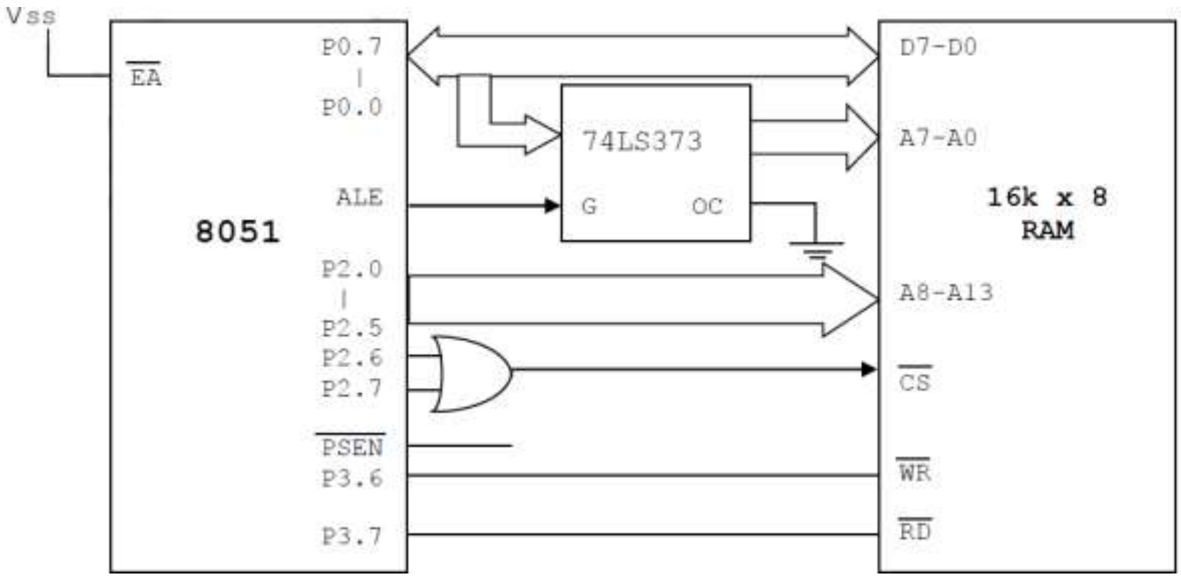


Figure 5.6.3 16Kx8 Memory (RAM) Interfacing with 8051

[Source: "The 8051Microcontroller and Embedded Systems: Using Assembly and C" by Mohamed Ali Maida, Janice Gillespie Maida, Robin McKinley]

Example 2: Design a controller system using 8051 to interface the external ROM of size 4k x 8.

Solution: Given, Memory size: 4k

i.e we require 2^n=4k :: n address lines

- Here n=12: A0 to A11 address lines are required.
- Remaining lines A0, A0, A0, and A0 & PSEN are connected though OR gate to CS & RD of external ROM.
- When A0 to A0 are low (logic '0'), only then external ROM is selected.

Address Decoding (Memory Map) for 4k x 8 RAM

Address	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	Hex Addr
Start	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000 H
End	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	0FF FH

Figure 5.6.4 shows interfacing of 4k x 8 ROM to 8051

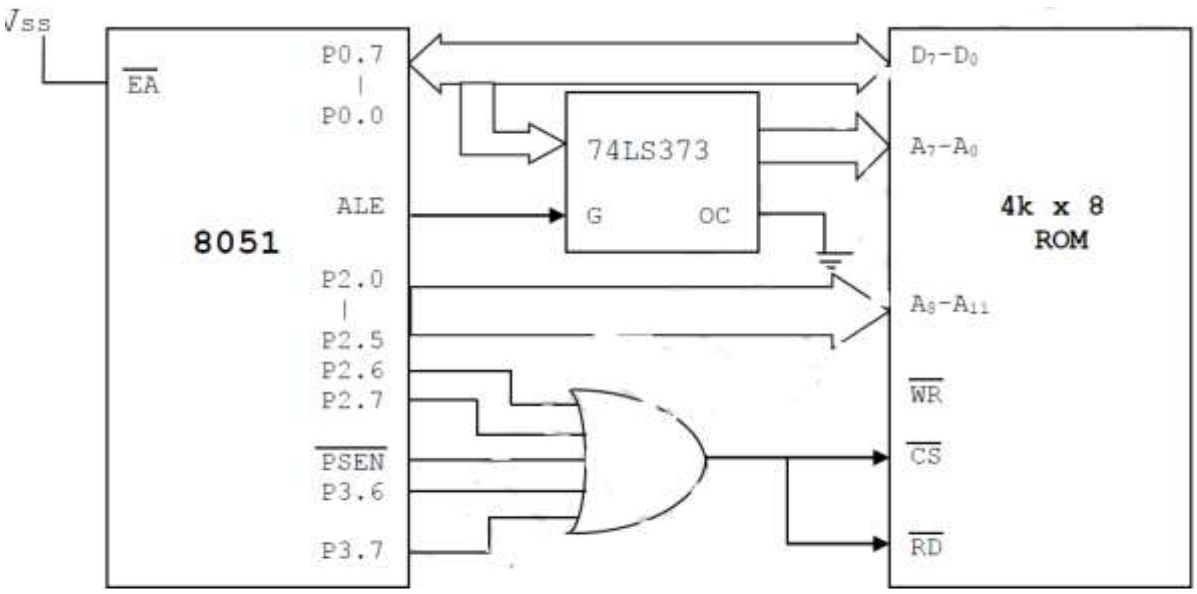


Figure 5.6.4 4Kx8 Memory (ROM) Interfacing with 8051

[Source: "The 8051Microcontroller and Embedded Systems: Using Assembly and C" by Mohamed Ali Maida, Janice Gillespie Maida, Robin McKinley, pg.no.320]

Example 3: Design a controller system using 8051, 16k bytes of ROM & 32k bytes of RAM. Interface the memory such that starting address for ROM is 0000H & RAM is 8000H.

Solution:

Given, Memory size- ROM: 16k

I.e. we require 2^n=16k: n address lines

Here n=14: A0 to A13 address lines are required. A14,

A15, PSEN Oared CS *when low – ROM is selected*

Memory size- RAM: 32k

i.e we require 2^n=32k :: n address lines

Here n=15: A0 to A15 address lines are required.

A15 inverted (NOT Gate) CS when high- RAM is selected. For RAM selection

- PSEN is used as chip select pin ROM.
- RD is used as read control signal pin..
- WR is used as write control signal pin.

Address Decoding (Memory Map) for 16k x 8 ROM.

Address	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	Hex Addr
Start	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000H
End	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	3FFFH

Address Decoding (Memory Map) for 32k x 8 RAM.

Address	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	Hex Addr
Start	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	8000H
End	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	FFFFH

Figure 5.6.5 shows the interfacing of 16Kx8 Memory (ROM) and 32Kx8 RAM with 8051

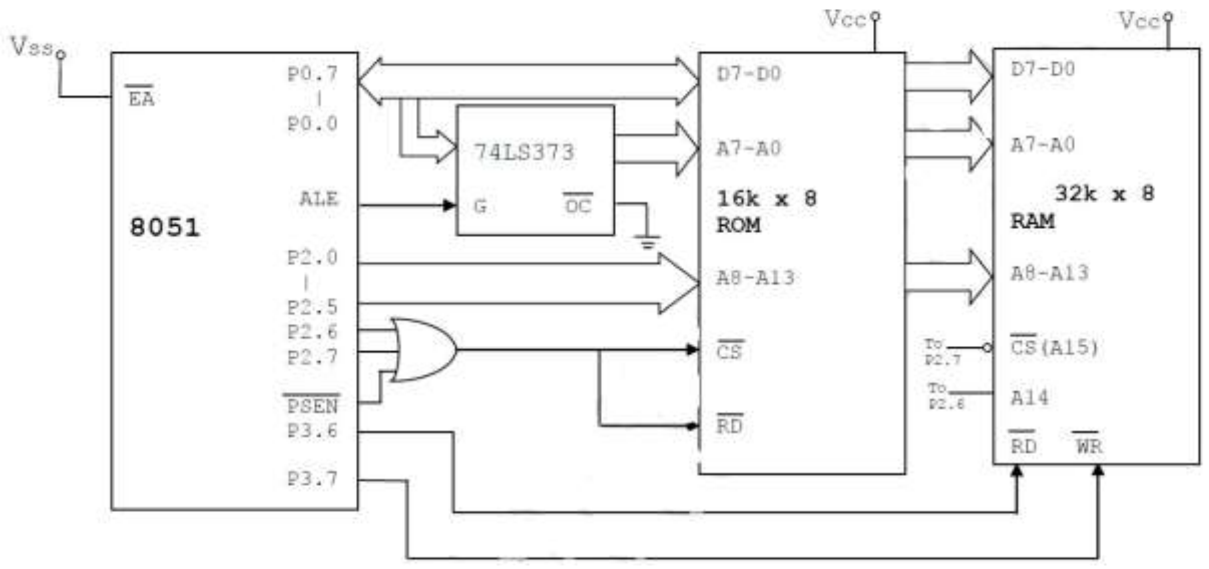


Figure 5.6.5 16Kx8 Memory (ROM) and 32Kx8 RAM Interfacing with 8051

[Source: "The 8051Microcontroller and Embedded Systems: Using Assembly and C" by Mohamed Ali Maida, Janice Gillespie Maida, Robin McKinley, pg.no.320]

5.3 INTERRUPT PROGRAMMING IN 8051

The Microcontroller can serve several devices. The Interrupt is the method to indicate the microcontroller by sending an interrupt signal. After receiving an interrupt, the microcontroller interrupts whatever it is doing and serves the device. The program associated with the interrupt is called the interrupt service routine (ISR). When an Interrupt is invoked, the microcontroller runs the interrupt service routine. For every interrupt, there is a fixed location set aside to hold the addresses of ISRs.

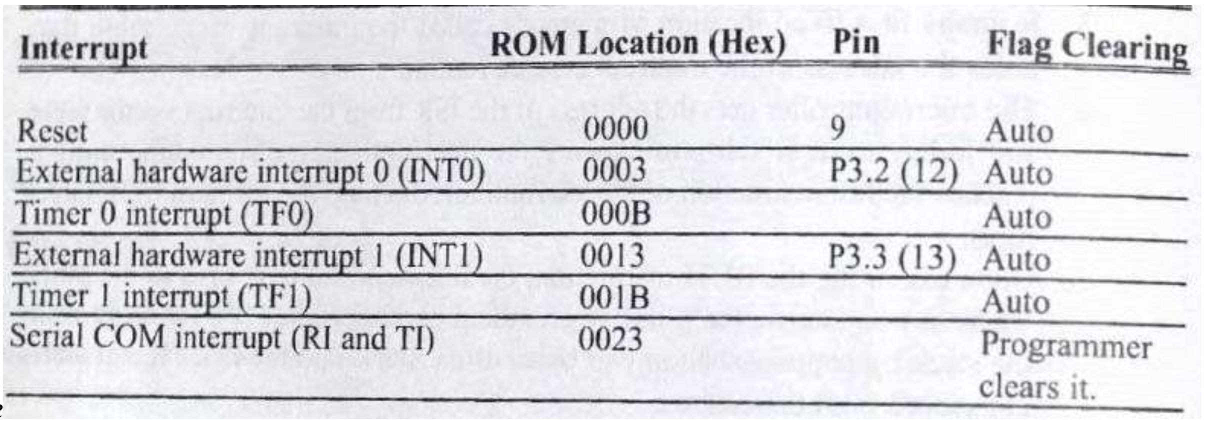
The following events will cause an interrupt:

1. Timer 0 Overflow.
2. Timer 1 Overflow.
3. Reception/Transmission of Serial Character.
4. External Event 0.
5. External Event 1.

To distinguish between various interrupts and executing different code depending on what interrupt was triggered, 8051may be jumping to a fixed address when a given interrupt occurs as shown in Table 5.3.1.

Table 5.3.1 Interrupt Vector Table for 8051

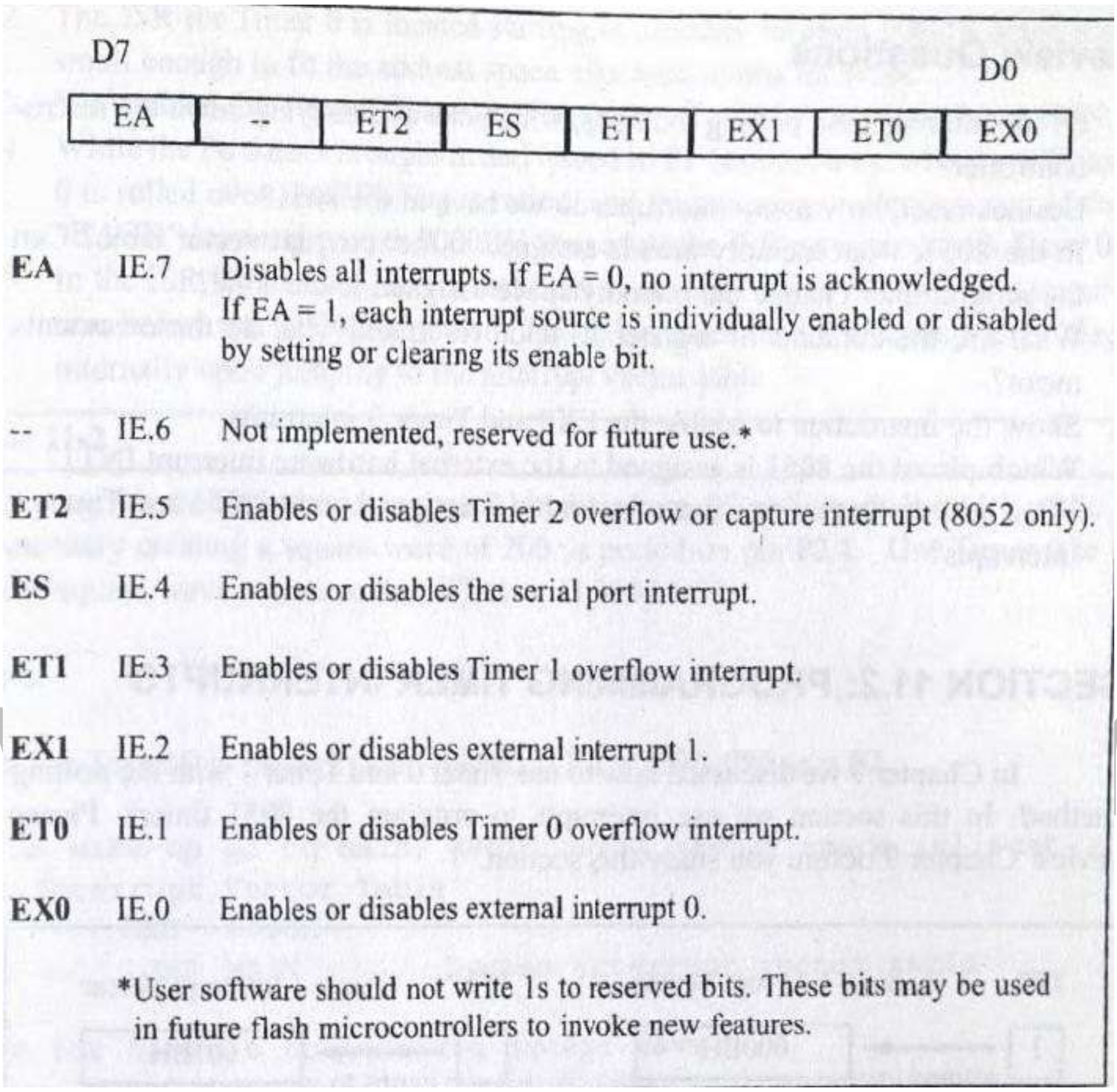
[Source: "The 8051Microcontroller and Embedded Systems: Using Assembly and C" by Mohamed Ali Mazidi, Janice Gillespie Maida, Rollin McKinley, pg.no.320]



Interrupt	ROM Location (Hex)	Pin	Flag Clearing
Reset	0000	9	Auto
External hardware interrupt 0 (INT0)	0003	P3.2 (12)	Auto
Timer 0 interrupt (TF0)	000B		Auto
External hardware interrupt 1 (INT1)	0013	P3.3 (13)	Auto
Timer 1 interrupt (TF1)	001B		Auto
Serial COM interrupt (RI and TI)	0023		Programmer clears it.

ENABLING AND DISABLING AN INTERRUPT

Upon reset all interrupts are disable, meaning that known will be responded to by the microcontroller if they are activated. The Interrupt must be enabled by software in order for microcontroller to respond to them there is a register called IE that is responsible for enabling and disabling the interrupts as shown in Figure 5.3.1



D7							D0
EA	--	ET2	ES	ET1	EX1	ET0	EX0

EA	IE.7	Disables all interrupts. If EA = 0, no interrupt is acknowledged. If EA = 1, each interrupt source is individually enabled or disabled by setting or clearing its enable bit.
--	IE.6	Not implemented, reserved for future use.*
ET2	IE.5	Enables or disables Timer 2 overflow or capture interrupt (8052 only).
ES	IE.4	Enables or disables the serial port interrupt.
ET1	IE.3	Enables or disables Timer 1 overflow interrupt.
EX1	IE.2	Enables or disables external interrupt 1.
ET0	IE.1	Enables or disables Timer 0 overflow interrupt.
EX0	IE.0	Enables or disables external interrupt 0.

*User software should not write 1s to reserved bits. These bits may be used in future flash microcontrollers to invoke new features.

Figure 5.3.1 Interrupt Enable (IE) Register

[Source: "The 8051Microcontroller and Embedded Systems: Using Assembly and C" by Mohamed Ali Maida, Janice Gillespie Maida, Rollin McKinley, pg.no.321]

PROGRAMMING EXTERNAL HARDWARE INTERRUPTS

The 8051 has two external hardware interrupts PIN 12 (P3.2) and Pin 13 (P3.3), designated as INT0 and INT1. Upon activation of these pins, the 8051 finishes the

Execution of current instruction whatever it is executing and jumps to the vector table to perform the interrupt service routine.

TYPES OF INTERRUPT

1) Level-Triggered Interrupt

2) Edge -Triggered Interrupt

LEVEL-TRIGGERED INTERRUPT

In this mode, INT0 and INT1 are normally high and if the low level signal is Applied to them, it triggers the Interrupt. Then the microcontroller stops and jumps to Interrupt vector table to service that interrupt. The low-level signal at the INT pin must Be removed before the execution of the last instruction of the ISR, RETI. Otherwise,

Interrupt the default mode upon reset

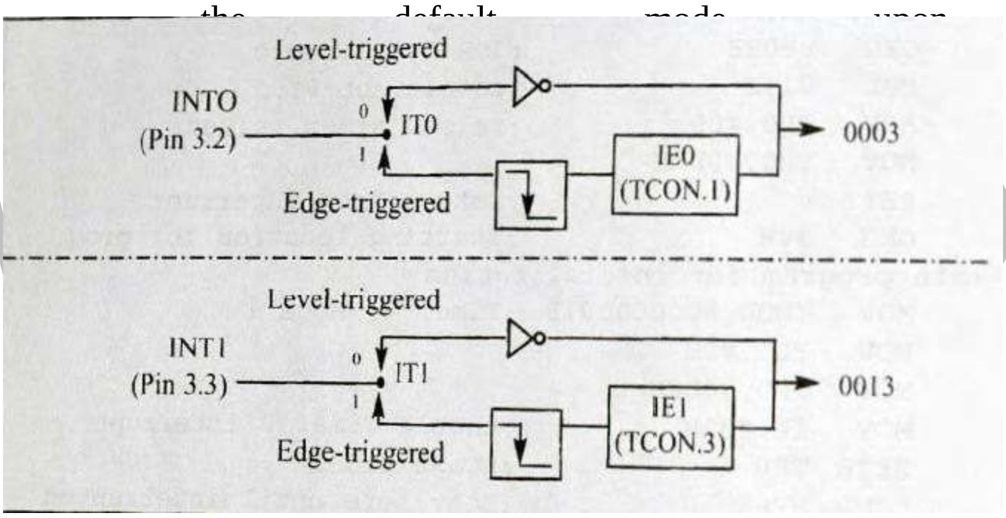
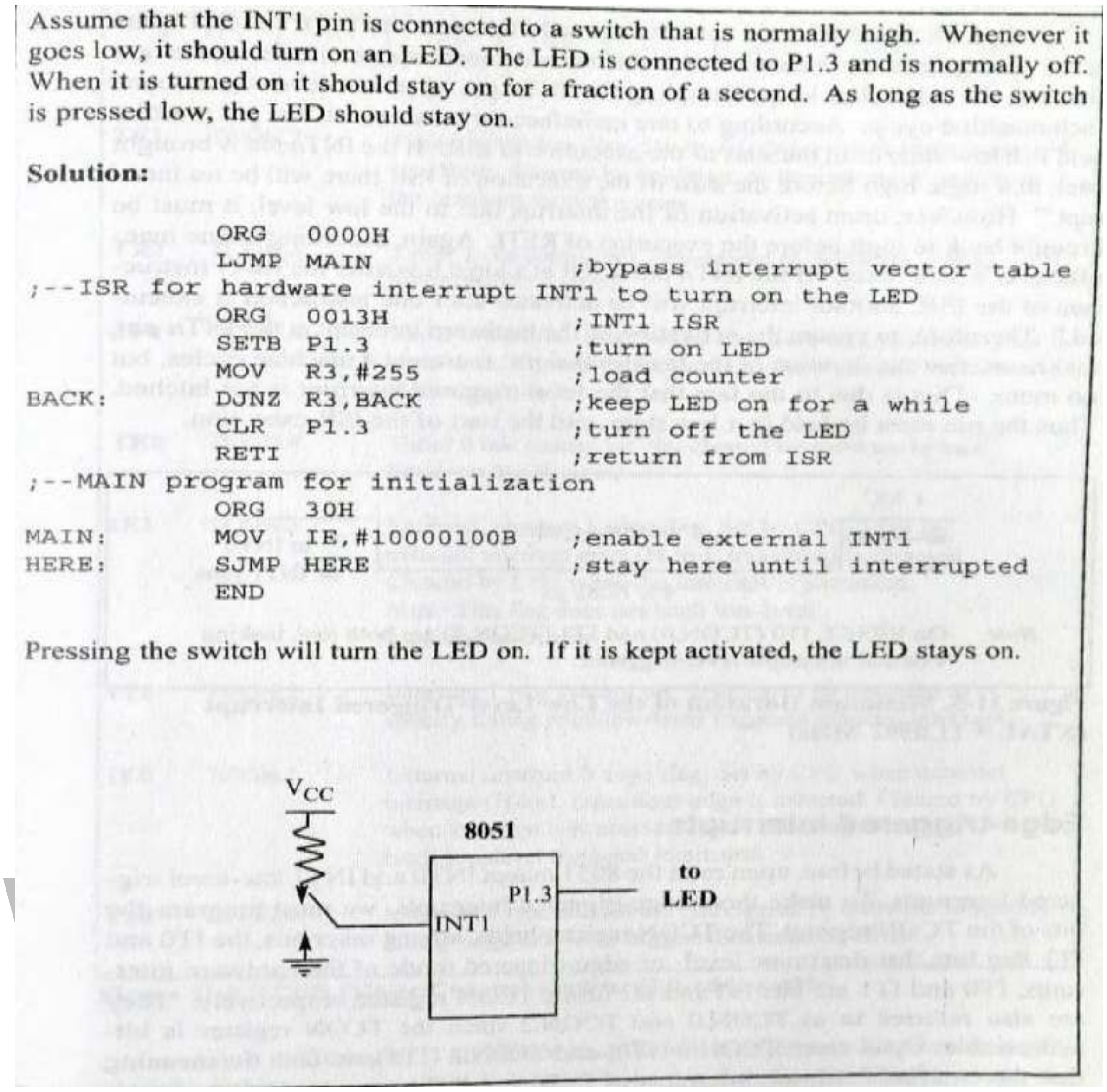


Figure 5.3.2 Activation of INT0 and INT1

[Source: "The 8051Microcontroller and Embedded Systems: Using Assembly and C" by Mohamed Ali Maida, Janice Gillespie Maida, Rollin McKinley, pg.no.326]

EDGE -TRIGGERED INTERRUPT

Upon reset 8051 makes INT0 and INT1 low l Level-Triggered Interrupt. To make them Edge -Triggered Interrupt, we must program the bits of the TCON Register. The TCON register holds among other bits and IT0 and IT1 flags bit the determine level- or edge triggered mode. IT0 and IT1 are bits D0 (TCON.0) and D2 (TCON.2) of the TCON Register respectively.



Assume that the INT1 pin is connected to a switch that is normally high. Whenever it goes low, it should turn on an LED. The LED is connected to P1.3 and is normally off. When it is turned on it should stay on for a fraction of a second. As long as the switch is pressed low, the LED should stay on.

Solution:

```
        ORG   0000H
        LJMP  MAIN             ;bypass interrupt vector table
;--ISR for hardware interrupt INT1 to turn on the LED
        ORG   0013H            ;INT1 ISR
        SETB  P1.3             ;turn on LED
        MOV   R3,#255          ;load counter
BACK:   DJNZ  R3,BACK          ;keep LED on for a while
        CLR   P1.3             ;turn off the LED
        RETI                   ;return from ISR
;--MAIN program for initialization
        ORG   30H
MAIN:   MOV   IE,#10000100B    ;enable external INT1
HERE:   SJMP  HERE             ;stay here until interrupted
        END
```

Pressing the switch will turn the LED on. If it is kept activated, the LED stays on.

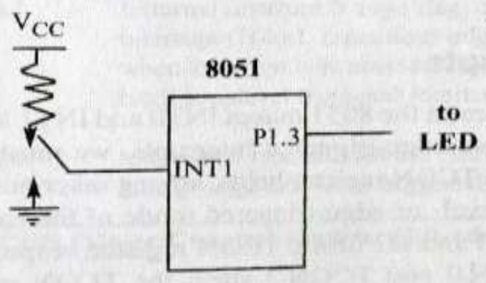


Figure 5.3.3 Example for Level triggered Interrupt

[Source: "The 8051Microcontroller and Embedded Systems: Using Assembly and C" by Mohamed Ali Maida, Janice Gillespie Maida, Rollin McKinley, pg.no.327]

SERIAL COMMUNICATION INTERRUPT

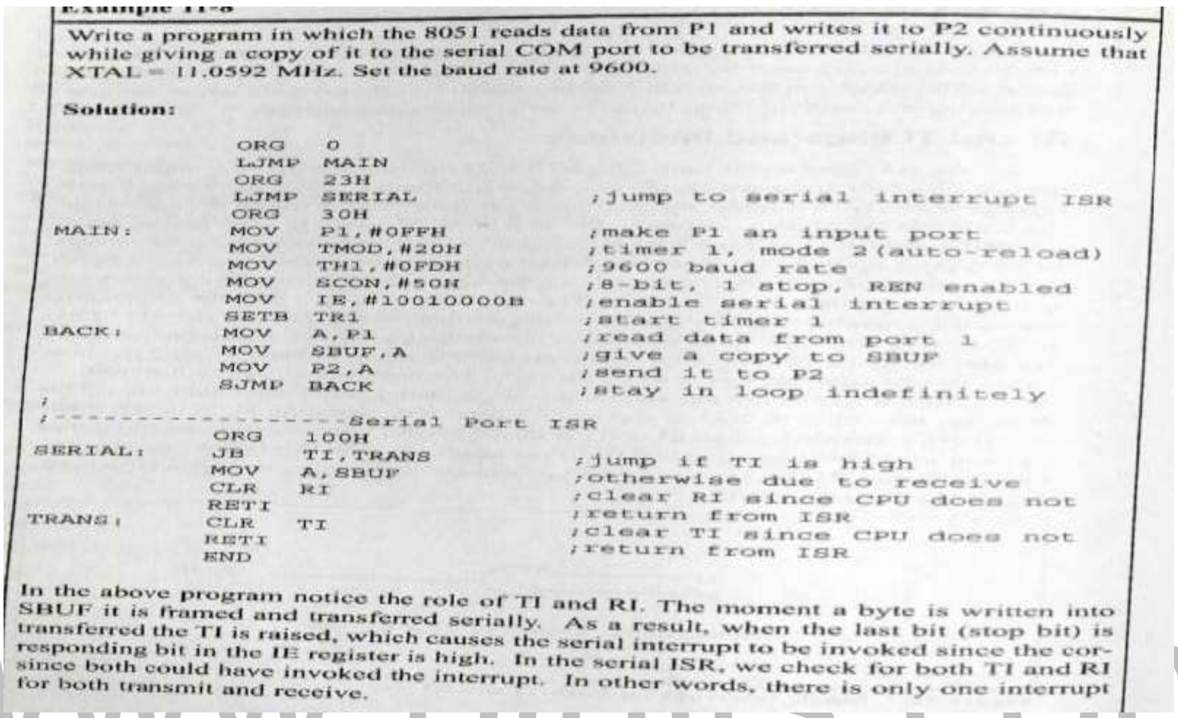
TI (transfer interrupt) is raised when the stop bit is transferred indicating that the SBUF register is ready to transfer the next byte

RI (received interrupt) is raised when the stop bit is received indicating that the received byte needs to be picked up before it is lost (overrun) by new incoming serial data

In the 8051 there is only one interrupt set aside for serial communication, used for both sending and receiving data.

If the interrupt bit in the IE register (IE.4) is enabled, when RI or TI is raised the 8051 gets interrupted and jumps to memory location 0023H to execute the ISR

In that ISR we must examine the TI and RI flags to see which one caused the interrupt and respond accordingly.



Example 11-8

Write a program in which the 8051 reads data from P1 and writes it to P2 continuously while giving a copy of it to the serial COM port to be transferred serially. Assume that XTAL = 11.0592 MHz. Set the baud rate at 9600.

Solution:

```
        ORG   0
        LJMP  MAIN
        ORG   23H
        LJMP  SERIAL            ;jump to serial interrupt ISR
        ORG   30H
MAIN:   MOV   P1,#0FFH          ;make P1 an input port
        MOV   TMOD,#20H         ;timer 1, mode 2(auto-reload)
        MOV   TH1,#0FDH         ;9600 baud rate
        MOV   SCON,#50H         ;8-bit, 1 stop, REN enabled
        MOV   IE,#10010000B     ;enable serial interrupt
        SETB  TR1               ;start timer 1
BACK:   MOV   A,P1              ;read data from port 1
        MOV   SBUF,A            ;give a copy to SBUF
        MOV   P2,A              ;send it to P2
        SJMP  BACK              ;stay in loop indefinitely
;
;-----------------Serial Port ISR
        ORG   100H
SERIAL: JB    TI,TRANS          ;jump if TI is high
        MOV   A,SBUF            ;otherwise due to receive
        CLR   RI                ;clear RI since CPU does not
        RETI                    ;return from ISR
TRANS:  CLR   TI                ;clear TI since CPU does not
        RETI                    ;return from ISR
        END
```

In the above program notice the role of TI and RI. The moment a byte is written into SBUF it is framed and transferred serially. As a result, when the last bit (stop bit) is transferred the TI is raised, which causes the serial interrupt to be invoked since the corresponding bit in the IE register is high. In the serial ISR, we check for both TI and RI since both could have invoked the interrupt. In other words, there is only one interrupt for both transmit and receive.

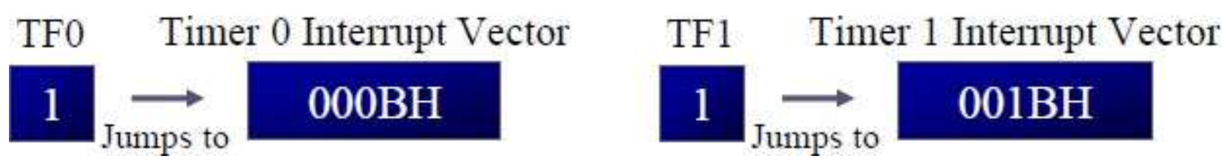
Figure 5.3.4 Example for Serial Communication Interrupt

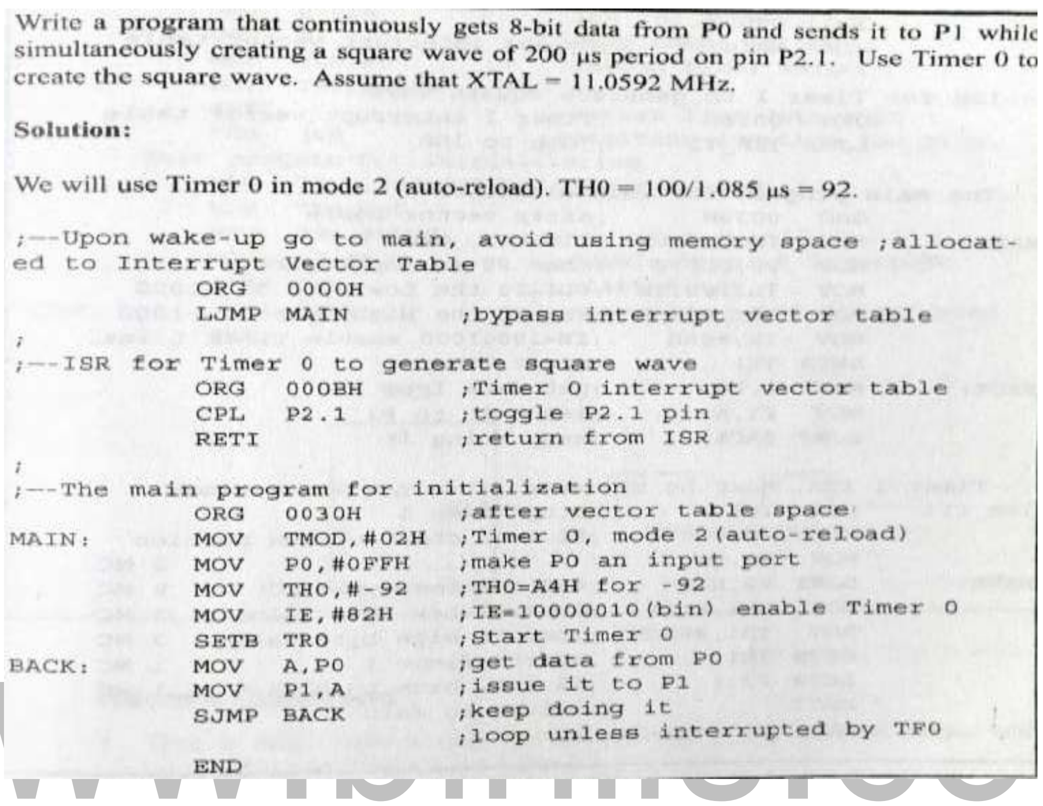
[Source: "The 8051Microcontroller and Embedded Systems: Using Assembly and C" by Mohamed Ali Maida, Janice Gillespie Maida, Rollin McKinley, pg.no.334]

TIMER INTERRUPTS

The timer flag (TF) is raised when the timer rolls over. In polling TF, we have to wait until the TF is raised. The microcontroller is tied down while waiting for TF to be raised, and cannot do anything else. If the timer interrupt in the IE register is enabled, whenever the timer rolls over, TF is raised. This avoids tying down the controller.

The microcontroller is interrupted in whatever it is doing, and jumps to the interrupt vector table to service the Irvin this way, the microcontroller can do other task until it is notified that the timer has rolled over





Write a program that continuously gets 8-bit data from P0 and sends it to P1 while simultaneously creating a square wave of 200 μs period on pin P2.1. Use Timer 0 to create the square wave. Assume that XTAL = 11.0592 MHz.

Solution:

We will use Timer 0 in mode 2 (auto-reload). TH0 = 100/1.085 μs = 92.

```
;--Upon wake-up go to main, avoid using memory space ;allocat-
ed to Interrupt Vector Table
        ORG   0000H
        LJMP  MAIN          ;bypass interrupt vector table
;
;--ISR for Timer 0 to generate square wave
        ORG   000BH         ;Timer 0 interrupt vector table
        CPL   P2.1          ;toggle P2.1 pin
        RETI                ;return from ISR
;
;--The main program for initialization
        ORG   0030H         ;after vector table space
MAIN:   MOV   TMOD,#02H     ;Timer 0, mode 2(auto-reload)
        MOV   P0,#0FFH      ;make P0 an input port
        MOV   TH0,#-92      ;TH0=A4H for -92
        MOV   IE,#82H       ;IE=10000010(bin) enable Timer 0
        SETB  TR0           ;Start Timer 0
BACK:   MOV   A,P0          ;get data from P0
        MOV   P1,A          ;issue it to P1
        SJMP  BACK          ;keep doing it
                            ;loop unless interrupted by TF0
        END
```

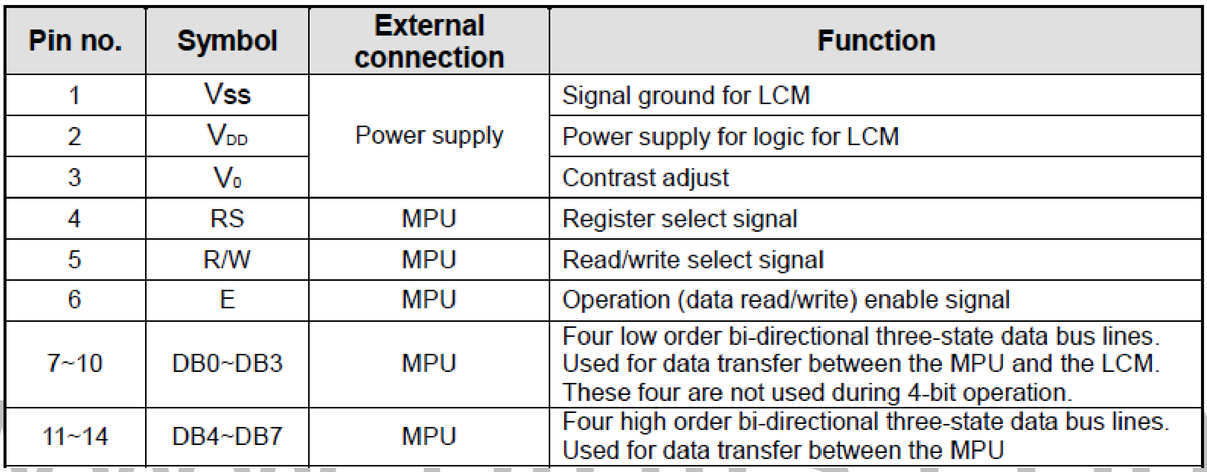
Figure 5.3.5 Example for Timer Interrupt

[Source: "The 8051Microcontroller and Embedded Systems: Using Assembly and C" by Mohamed Ali Maida, Janice Gillespie Maida, Rollin McKinley, pg.no.323]

5.4 LCD AND KEYBOARD INTERFACING

LCD (LIQUID CRYSTAL DISPLAY) INTERFACE

LCDs can display numbers, characters, and graphics. To produce a proper display, the information has to be periodically refreshed. This can be done by the CPU or internally by the LCD device itself. Incorporating a refreshing controller into the LCD, relieves the CPU of this task and hence many LCDs have built-in controllers. These controllers also facilitate flexible programming for characters and graphics. Table 5.1 shows the pin description of an LCD. From Optra.



Pin no.	Symbol	External connection	Function
1	Vss	Power supply	Signal ground for LCM
2	V_{DD}		Power supply for logic for LCM
3	V_0		Contrast adjust
4	RS	MPU	Register select signal
5	R/W	MPU	Read/write select signal
6	E	MPU	Operation (data read/write) enable signal
7~10	DB0~DB3	MPU	Four low order bi-directional three-state data bus lines. Used for data transfer between the MPU and the LCM. These four are not used during 4-bit operation.
11~14	DB4~DB7	MPU	Four high order bi-directional three-state data bus lines. Used for data transfer between the MPU

Table 5.4.1 Pin description of LCD

[Source: "The 8051Microcontroller and Embedded Systems: Using Assembly and C" by Mohamed Ali Maida, Janice Gillespie Maida, Rollin McKinley]

- Voss and VDD provide +5v and ground, V0 is used for controlling LCD contrast.
- If RS=0, the instruction command register is selected, allowing the user to send a command such as clear display, cursor at home, etc.
- If RS=1 the data register is selected, allowing the user to send data to be displayed on the LCD.
- R/W input allows the user to Read/ Write the information to the LCD.
- The enable pin is used by the LCD to latch information presented to its data pins.
- The 8-bit data pins are used to send information to LCD.
- Section 5.1 discusses about command codes for writing the instructions on the LCD register. Section 5.2 gives an example program for displaying a character on the LCD.

LCD COMMAND CODES

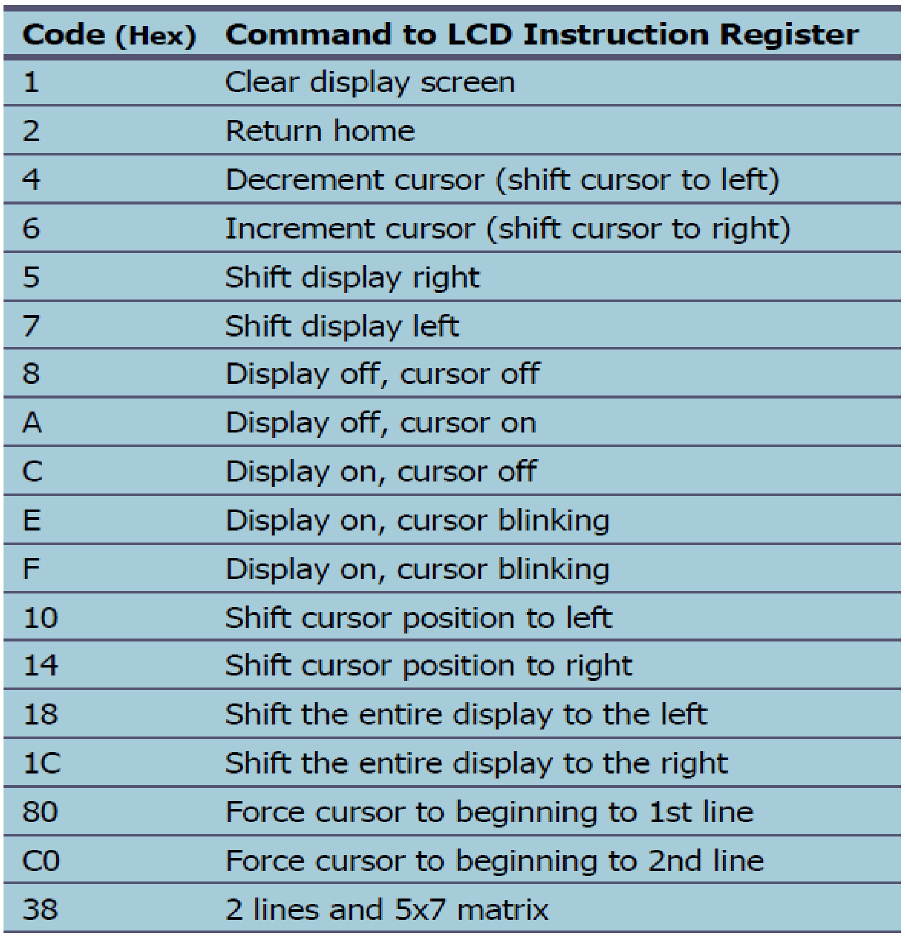
The LCD's internal controller can accept several commands and modify the display accordingly. These commands would be things like:

√ Clear screen

√ Return home

√ Decrement/Increment cursor

After writing to the LCD, it takes some time for it to complete its internal operations. During this time, it will not accept any new commands or data. Figure 5.4.1 shows the command codes of LCD and Figure 5.4.2 shows the LCD interfacing. We need to insert a time delay between any two commands or data sent to LCD.



Code (Hex)	Command to LCD Instruction Register
1	Clear display screen
2	Return home
4	Decrement cursor (shift cursor to left)
6	Increment cursor (shift cursor to right)
5	Shift display right
7	Shift display left
8	Display off, cursor off
A	Display off, cursor on
C	Display on, cursor off
E	Display on, cursor blinking
F	Display on, cursor blinking
10	Shift cursor position to left
14	Shift cursor position to right
18	Shift the entire display to the left
1C	Shift the entire display to the right
80	Force cursor to beginning to 1st line
C0	Force cursor to beginning to 2nd line
38	2 lines and 5x7 matrix

Figure 5.4.1 LCD Command Codes

[Source: "The 8051Microcontroller and Embedded Systems: Using Assembly and C" by Mohamed Ali Maida, Janice Gillespie Maida, robin McKinley, pg.no.353]

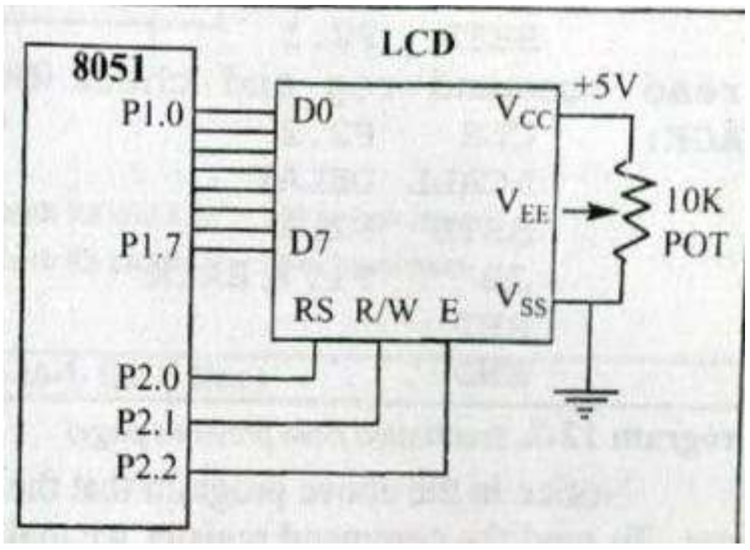


Figure 5.4.2 LCD Connections to 8051

[Source: "The 8051Microcontroller and Embedded Systems: Using Assembly and C" by Mohamed Ali Mazzini, Janice Gillespie Mazzini, robin McKinley, pg.no.355]

PROGRAM TO DISPLAY CHARACTERS ON LCD

To send any of the commands to the LCD, make pin RS=0. For data, make RS=1. Then send a high-to-low pulse to the E pin to enable the internal latch of the LCD. This is shown in the code below.

```
; calls a time delay before sending next data/command
; P1.0-P1.7 are connected to LCD data pins D0-D7
; P2.0 is connected to RS pin of LCD
; P2.1 is connected to R/W pin of LCD
; P2.2 is connected to E pin of LCD
MOV A, #38H                 ; INIT. LCD 2 LINES, 5X7 MATRIX
ACALL COMNWRT               ; call command
subroutine ACALL DELAY      ; give LCD some time
MOV A,#0EH                  ;display on, cursor on
ACALL COMNWRT               ;call command subroutine
ACALL DELAY                 ;give LCD some time
MOV A, #01                  ; clear LCD
ACALL COMNWRT               ; call command
subroutine ACALL DELAY      ; give LCD some time
```

```
MOV A,#06H                      ;shift cursor right
ACALL COMNWRT                   ;call command subroutine
ACALL DELAY                     ;give LCD some time
MOV A, #84H                     ; cursor at line 1, pos. 4
ACALL COMNWRT                   ; call command
subroutine ACALL DELAY          ; give LCD some time
MOV A, #'N'                     ; display letter N
ACALL DATAWRT                   ; call display
subroutine ACALL DELAY          ; give LCD some time
MOV A, #'O'                     ; display letter O
ACALL DATAWRT                   ; call display
subroutine AGAIN: SJMP AGAIN    ; stay here
```

COMNWRT: send command to LCD

```
MOV P1,A                        ;copy rig A to port 1
CLR P2.0                        ; RS=0 for command
CLR P2.1                        ; R/W=0 for write
SETB P2.2                       ; E=1 for high pulse
CLR P2.2                        ; E=0 for H-to-L
pulse RET
```

DATAWRT: write data to LCD

```
MOV P1, A                       ; copy rig A to port 1
SETB P2.0                       ; RS=1 for DATA
CLR P2.1                        ; R/W=0 for write
SETB P2.2                       ; E=1 for high pulse
CLR P2.2                        ; E=0 for H-to-L
pulse RET
```

DELAY: MOV R3, #50   ; 50 or higher for fast

```
CPUs HERE 2: MOV R4, #255       ; R4 = 255
```

HERE: DJNZ R4, ; stay until R4 becomes

HERE DJNZ R3, HERE

2

KEYBOARD INTERFACING WITH 8051

Keys in a keyboard are arranged in a matrix of rows and columns. The controller access both rows and columns through ports. Using two ports, we can connect to an 8x8 or a 4x4 matrix keyboard. When a key is pressed, a row and column make a contact, Otherwise there is no contact. We will look at the details using a 4x4 keyboard.

4X 4 KEYBOARD

Figure 5.4.31 shows a 4 x4 matrix connected to two ports.

- The rows are connected to an output port (Port 1) and the columns are connected to an input port. (Port 2)
- If no key has been pressed, reading the input port will yield 1s for all columns since they are all connected to high (Vic).
- If all the rows are grounded and a key is pressed, one of the columns will have 0 since the key pressed provides the path to ground.
- It is the function of the microcontroller to scan the keyboard continuously to detect and identify the key pressed.

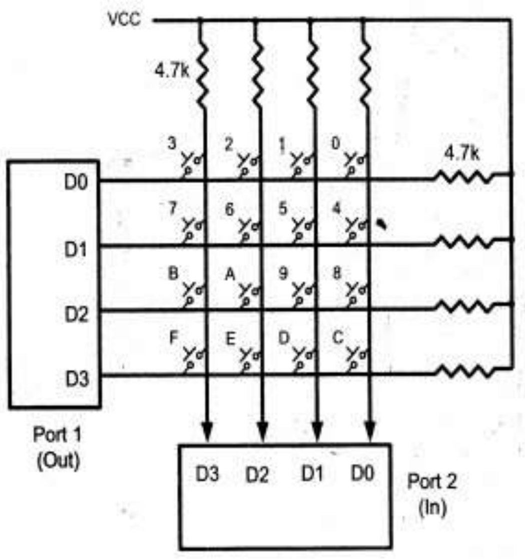


Figure 5.4.3 Matrix Keyboard Connections to Ports

[Source: "The 8051Microcontroller and Embedded Systems: Using Assembly and C" by Mohamed Ali Mazzini, Janice Gillespie Mazzini, robin McKinley]

KEY SCAN

To find out the key pressed, the controller grounds a row by sending a '0' on the corresponding line of the output port. It then reads the data at the columns using the input port. If data from columns is D3-D0=1111, then no key is pressed. If any bit of the column is '0', it indicates that a key is pressed in that column. In this example, the column is identified by the following values:

1110 – Key pressed in column

0 1101 – key pressed in column

1 1011 – key pressed in column

2 0111 – key pressed in column

3

STEPS TO FIND OUT KEY PRESSED

Beginning with the row 0, the microcontroller grounds it by providing a low to row D0 only. It then reads the columns (port2). If the data read is all 1s, then no key in that row is activated and the process is moved to the next row. It then grounds the next row, reads the columns, and checks for any zero. This process continues until a row with a zero is identified. After identification of the row in which the key has been pressed, the column to which the pressed key belongs is identified as discussed above - by looking for a zero in the input values read.

Example:

(a) D3 – D0 = 1101 for the row, D3 – D0 = 1011 for the column, indicate row 1 and column 3 are selected. This indicates that key 6 is pressed.

(b) D3 – D0 = 1011 for the row, D3 – D0 = 0111 for the column, indicate row 2 and column 3 are selected. Then key 'B' is pressed.

PROGRAM:

The program used for detection and identification of the key activated goes through the following stages:

1. To make sure that the preceding key has been released, 0s are output to all rows at once, and the columns are read and checked repeatedly until all the columns are high.

- When all columns are found to be high, the program waits for a short amount of time before it goes to the next stage of waiting for a key to be pressed.

2. To see if any key is pressed, the columns are scanned over and over in an infinite loop until one of them has a 0 on it.

- Remember that the output latch is connected to rows, still have their initial zeros (in stage 1), making them grounded.
- After the key press detection, it waits for 20-ms for the bounce and then scans the columns again.

i) It ensures that the first key press detection was not an erroneous one due to spike noise.

ii) After the 20-ms delay, if the key is still pressed, then it goes to the loop (step 3) to detect the actual key pressed.

3. To detect which row the key pressed belongs to, it grounds one row at a time, reading the columns each time.

• If it finds that all columns are high, this means that the key press does not belong to that row. Therefore, it grounds the next row and continues until it finds the row, that the key pressed belongs to.

• Upon finding the row that the key pressed belongs to, it sets up the starting address for the lookup table holding the scan codes for that row.

4. To identify the key pressed, it rotates the column bits, one bit at a time, into the carry flag and checks to see if it is low.

• Upon finding the zero, it pulls out the ASCII code for that key from the look-up Table.

• Otherwise, it increments the pointer to point to the next element of the look-up Table.

Figure 5.4.4 provides the flowchart for keyboard interfacing Program for scanning and identifying the pressed key.

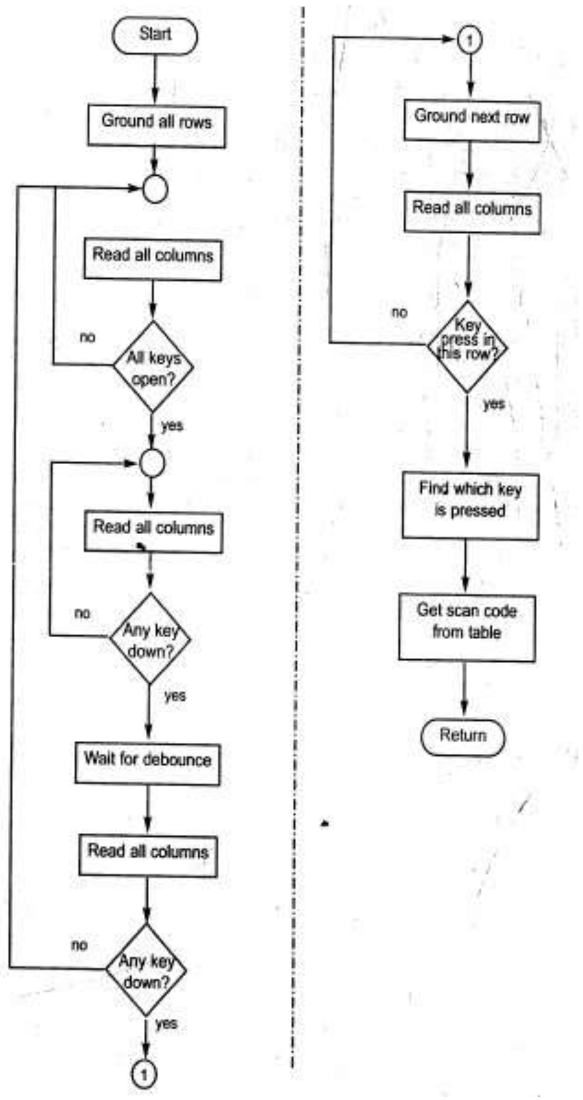


Figure 5.4.4 Flowchart for Keyboard Interfacing

[Source: "The 8051Microcontroller and Embedded Systems: Using Assembly and C" by Mohamed Ali Mazzini, Janice Gillespie Mazzini, robin McKinley, pg.no.365]

PROGRAM:

```
; Keyboard subroutine. This program sends the ASCII code for pressed key to P0.1
; P1.0-P1.3 connected to rows, P2.0-P2.3 to column
MOV P2, #0FFH                  ; make P2 an input port
K1: MOV P1, #0                 ; ground all rows at once
MOV A, P2                      ; read all col
                               ;( ensure keys open)
ANL A, 00001111B               ; masked unused bits
CJNE A, #00001111B, K1         ; till all keys release
```

```
K2: ACALL DELAY                 ; call 20 sec delay
MOV A, P2                       ; see if any key is pressed
ANL A,00001111B                 ;mask unused bits
CJNE A,#00001111B,OVER          ;key pressed, find row
SJMP K2                         ;check till key pressed
OVER: ACALL DELAY               ; wait 20 sec denounce time
MOV A, P2                       ; check key closure
ANL A,00001111B                 ;mask unused bits
CJNE A,#00001111B,OVER1         ;key pressed, find row
SJMP K2                         ;if none, keep polling
OVER1: MOV P1, #11111110B       ;ground row 0
MOV A, P2                       ; read all columns
ANL A,#00001111B                ;mask unused bits
CJNE A,#00001111B,ROW_0         ;key row 0, find col.
MOV P1,#11111101B               ;ground row 1
MOV A,P2                        ;read all columns
ANL A,#00001111B                ;mask unused bits
CJNE A,#00001111B,ROW_1         ;key row 1, find col.
MOV P1,#11111011B               ;ground row 2
MOV A,P2                        ;read all columns
ANL A,#00001111B                ;mask unused bits
CJNE A,#00001111B,ROW_2         ;key row 2, find col.
MOV P1,#11110111B               ;ground row 3
MOV A, P2                       ; read all columns
ANL A,#00001111B                ;mask unused bits
CJNE A,#00001111B,ROW_3         ;key row 3, find col.
LJMP K2                         ;if none, false input,
ROW_0: MOV DPTR, #KCODE0        ; set DPTR=start of row
0 SJMP FIND                     ; find col. Key belongs
to ROW_1: MOV DPTR, #KCODE1     ; set DPTR=start of row
SJMP FIND                       ; find col. Key belong To
```

```
ROW_2: MOV DPTR, #KCODE2; set DPTR=start of row 2
SJMP FIND                        ; find col. Key belongs
to ROW_3: MOV DPTR, #KCODE3; set DPTR=start of row 3
FIND: RRC A                      ; see if any CY bit low
JNC MATCH                        ; if zero, get ASCII code
INC DPTR                          ; point to next col. add
SJMP FIND                        ; keep searching
MATCH: CLR A                     ; set A=0 (match is found)
MOVC A,@A+DPTR                   ; get ASCII from table
MOV P0, A                        ; display pressed
key LJMP K1


        ; ASCII LOOK-UP TABLE FOR EACH ROW
ORG 300H
KCODE0: DB ‘0’,’1’,’2’,’3’       ; ROW 0
KCODE1: DB ‘4’,’5’,’6’,’7’       ; ROW 1
KCODE2: DB ‘8’,’9’,’A’,’B’       ; ROW 2
KCODE3: DB ‘C’,’D’,’E’,’F’       ; ROW 3
```

5.1 PROGRAMMING 8051 TIMERS

8051 TIMERS

8051 has two timers/Counters (TIMER 0 and TIMER 1). They can be used as timers to generate time delays or as event counters.

BASIC REGISTERS OF THE TIMER

1.TIMER REGISTERS

It is a16 bit register and can be accessed as 8 bit registers say Timer High(TX) and Timer Low (TX),These registers can be accessed as any other registers like A,B,R0 etc.,

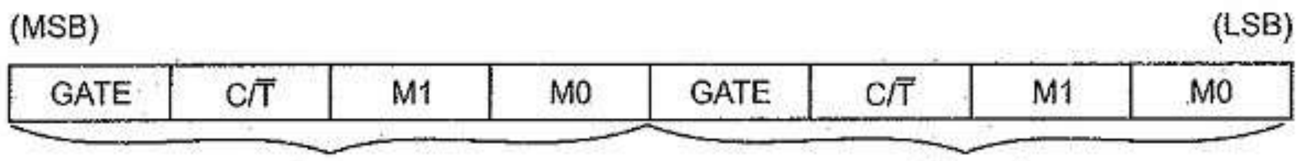
TX								**TX**							
D15	**D14**	**D13**	**D12**	**D11**	**D10**	**D9**	**D8**	**D7**	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**

Figure 5.1.1 Timer Registers

[Source: "The 8051Microcontroller and Embedded Systems: Using Assembly and C" by Mohamed Ali Mazda, Janice Gillespie Mazda, robin McKinley]

2. TMOD (Timer Mode) Register

TMOD Register is an 8-bit register used to control the mode of operation of both timers. The high four bits (bits 4 through 7) relate to Timer 1whereas the low four bits (bits 0 through 3) perform the exact same functions, but for timer 0. In each case, the lower two bits are used to set the timer mode and upper two bits to specify the operation as shown in Figure 5.1.2.



GATE	Gating control when set. Timer/Counter "x" is enabled only while "$\overline{\text{INTx}}$" pin is high and "TRx" control bit is set. When cleared Timer "x" is enabled whenever "TRx" control bit is set.
C/T	Timer or Counter selector cleared for Timer operation (input from internal system clock). Set for Counter operation (input from "Tx" input pin).

M1	M0	Operating Mode
0	0	8-bit Timer/Counter "THx" with "TLx"s 5-bit prescaler.
0	1	16-bit Timer/Counter "THx" with "TLx" are cascaded; there is no prescaler.
1	0	8-bit auto-reload Timer/Counter "THx" holds a value which is to be reloaded into "TLx" each time it overflows.
1	1	(Timer 0) TL0 is an 8-bit Timer/Counter controlled by the standard Timer 0 control bits. TH0 is an 8-bit timer only controlled by Timer 1 control bits.
1	1	(Timer 1) Timer/Counter 1 stopped.

Figure 5.1.2 TMOD Register

[Source: "The 8051Microcontroller and Embedded Systems: Using Assembly and C" by Mohamed Ali Mazda, Janice Gillespie Mazda, robin McKinley, pg.no.241]

In upper or lower 4 bits, first bit is a GATE bit. Every timer has a means of starting and stopping. Some timers do this by software, some by hardware, and some have both software and hardware controls. The hardware way of starting and stopping the timer by an external source is achieved by making GATE=1 in the TMOD register and if GATE=0 then we do start and stop the timers by programming.

The second bit is C/T bit and is used to decide whether a timer is used as a time delay generator or an event counter. If this bit is 0 then it is used as a timer and if it is 1 then it is used as a counter.

In upper or lower 4 bits, the last bits third and fourth are known as M1 and M0 respectively. These are used to select the timer mode.

TIMER'S CLOCK FREQUENCY AND ITS PERIOD

In 8051-based system, the crystal oscillator has a frequency of 11.0592 MHz when C/T bit of TMOD is 0. Each machine cycle is made up of 12 clock cycles.

Hence for a single machine cycle, the frequency becomes 1/12 × 11.0529 MHz = 921.6 KHz. For a single machine cycle, the time taken is T = 1/921.6 KHz = 1.085 us, so the oscillator takes 1.085us for completing a single machine cycle.

MODES OF OPERATION:

MODE 1:

It is a 16-bit timer; therefore it allows values from 0000 to FFFFH to be loaded into the timer's registers TL and TH as shown in Figure 5.1.3. After TH and TL are loaded with a 16-bit initial value, the timer must be started. We can do it by "SETB TR0" for timer 0 and "SETB TR1" for timer 1. After the timer is started, it starts count up until it reaches its limit of FFFFH. When it rolls over from FFFF to 0000H, it sets high a flag bit called TFx (timer flag). This timer flag can be monitored. When this timer flag is raised, one option would be stop the timer with the instructions "CLR TR0" or CLR TR1 for timer 0 and timer 1 respectively. Again, it must be noted that each timer flag TF0 for timer 0 and TF1 for timer1. 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.

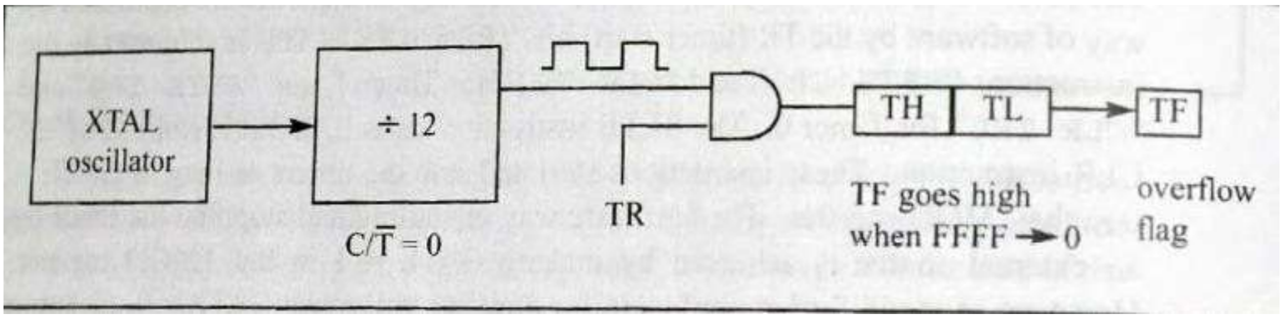


Figure 5.1.3 Timer in Mode 1

[Source: "The 8051Microcontroller and Embedded Systems: Using Assembly and C" by Mohamed Ali Mazda, Janice Gillespie Mazda, robin McKinley, pg.no.244]

MODE 0:

Mode 0 is exactly same like mode 1 except that it is a 13-bit timer instead of 16-bit. The 13- bit counter can hold values between 0000 to 1FFFH in TH-TL.

Therefore, when the timer reaches its maximum of 1FFH, it rolls over to 0000, and TF is raised.

MODE 2:

It is an 8 bit timer that allows only values of 00 to FFH to be loaded into the Timer's register TH as shown in Figure 5.1.4. After THz is loaded with 8 bit value, the 8051 gives a copy of it to telex. Then the timer must be started. It is done by the instruction "SETB TR0" for timer 0 and "SETB TR1" for timer1. This is like mode 1.

After timer is started, it starts to count up by incrementing the telex register. It counts up until it reaches its limit of FFH. When it rolls over from FFH to 00, it sets high the TX (timer flag). If we are using timer 0, TF0 goes high; if using TF1 then TF1 is raised. When TX register rolls from FFH to 00 and TF is set to 1, telex is reloaded automatically with the original value kept by the THz register. To repeat the process, we must simply clear TX and let it go without any need by the programmer to reload the original value. This makes mode 2 auto reload, in contrast in mode 1 in which programmer has to reload THz and TX.

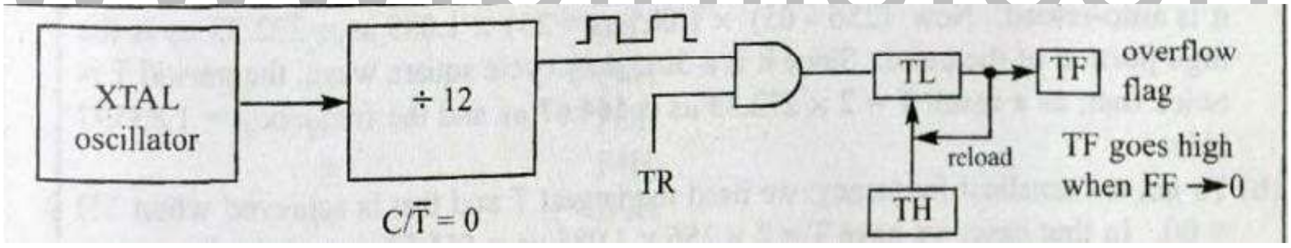


Figure 5.1.4 Timer in Mode 2

[Source: "The 8051Microcontroller and Embedded Systems: Using Assembly and C" by Mohamed Ali Mazda, Janice Gillespie Mazda, robin McKinley, pg.no.251]

MODE 3:

Mode 3 is also known as a split timer mode. Timer 0 and 1 may be Programmed to be in mode 0, 1 and 2 independently of similar mode for other timer. This is not true for mode 3; timers do not operate independently if mode 3 is chosen fortimer

0. Placing timer 1 in mode 3 causes it to stop counting; the control bit TR1 and the timer 1 flag TF1 are then used by timer 0.

TIMER PROGRAMMING

MODE 1 PROGRAMMING:

It is a 16 bit Timer mode.

STEPS TO PROGRAM IN MODE 1:

1. Load the TMOD value register with mode and timer0 or 1.
2. Load registers TX and THz with initial count corresponding to delay.
3. Start the timer.
4. Continuously monitor the timer flag (TX) with the “JNB TX, target” instruction to see if it is raised, if it is raised (TX=1) then get out of the loop.
5. Stop the timer.
6. Clear the TX flag for the next round.
7. Go back to Step 2 to load THz and TX again.

STEPS TO CALCULATE COUNT TO LOAD INTO THX-TLX TO GENERATE DESIRED DELAY

(Assume XTAL = 11.0592 MHz)

Steps for finding the TH, TL registers values

1. Divide the desired time delay by 1.085 us
2. Perform 65536 – n, where n is the decimal value we got in Step 1
3. Convert the result of Step 2 to hex, say we get lynx. Lynx is the initial hex value to be loaded into the timer’s register.
4. Set TL = xx and TH = YY

Example:

Generate Delay =10ms with Clock frequency= 11.0592 MHz, using Timer0 in mode1.

Solution:

Given:

Time delay=10ms

Clock frequency=11.0592 MHz

Step 1: Divide the desired time delay by 1.085 us

Count =10 ms/ 1.085 us=9216

Step 2: Perform 65536 – n

65536-9216=56320= DC00H

Step 3: Set TL = xx and TH = yy

Here xx=DC and YY=00, Hence, TH0=DC and TL0=00.

Following is the assembly code program to generate a delay of 10ms.

```
MOV TMOD, #01                ; Timer 0, mode 1, 16-bitmode
HERE: MOV TL0, #00           ; TL0=0, the low byte
MOV TH0, #0DCH               ; TH0=DC, the high byte
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
```

Program to generate a square wave of 5 kHz frequency on pin P1.0, clock frequency =11.0592 MHz

Given:

Square wave frequency=5 kHz

Clock frequency=11.0592 MHz

Step 1: Calculate the Time delay

T=1/f=1/5 kHz = 0.2 MS

T=0.2 MS which is the period of square

wave T/2 =0.2/2=0.1 MS delay for high and

low

Step 2: Divide the desired time delay by 1.085 us

Count=0.1ms/1.085 us = 921

Step 3: Perform 65536 – n

TH0-TL0= 65536-921=64615=FC67H

Step 4: Set TL = xx and TH = yy

Here xx=FC and yy=67. Hence,TH0=FCh,TL0=67h

```
MOV TMOD,#01                  ;Timer 0, mode 1, 16-bitmode
AGAIN: MOV TL1,#67H           ;TL1=67, low byte of timer
MOV TH1,#0FCH                 ;TH1=FC, the high byte
SETB TR1                      ;Start timer 1 BACK: JNB TF1,BACK ;until
                             Timer rolls over
CPL P1.0                      ; compliment P1.0
CLR TR1                        ; Stop the timer 1
CLR TF1                       ; Clear timer 1 flag
SJMP AGAIN                    ; Reload timer
```

STEPS FOR GENERATING DELAY IN MODE 2

1. Select timer in TMOD register indicating which timer (timer 0 or timer 1) is to be used, and the timer mode (mode 2) to be selected
2. THz register loaded with initial count value
3. Start timer
4. Continuously monitoring the timer flag (TX) with the JNB TX, target instruction to see whether TX is '1' (high) . Get out of the loop when TF goes high
5. Clear the TX flag
6. Go back to Step4, since mode 2 is auto-reload

Toggle LED connected at P1.0 with 5 micro sec delay using timer1 and mode 2

```
MOV TMOD, #20                 ; Timer 1 mode 2, 8-bit auto
reload AGAIN: MOV TH1, #-5; TL1=256-5, low byte of timer
SETB TR1                      ; Start timer 1
BACK: JNB TF1, BACK           ; until timer rolls over
CPL P1.0                      ; compliment P1.0 toggle LED
CLR TF1                       ; Stop the timer 1
SJMP BACK                     ; loop
```

5.2 SERIAL PORT PROGRAMMING

Data transfer between the PC and an 8051 system without any error is possible, if the baud rate of the 8051 system matches the baud rate of the PC's COM port.

BAUD RATE IN THE 8051

The 8051 transfers and receives data serially at many different baud rates. Serial communications of the 8051 is established with PC through the COM port. It must make sure that the baud rate of the 8051 system matches the baud rate of the PC's COM port/ any system to be interfaced. The baud rate in the 8051 is programmable. This is done with the help of Timer. When used for serial port, the frequency of timer is determined by (XTAL/12)/32 and 1 bit is transmitted for each timer period.

The Relationship between the crystal frequency and the baud rate in the 8051 is that the 8051 divides the crystal frequency by 12 to get the machine cycle frequency which is shown in Figure 5.2.1. Here the oscillator is XTAL = 11.0592 MHz, the machine cycle frequency is 921.6 kHz. 8051's UART divides the machine cycle frequency of 921.6 kHz by 32 once more before it is used by Timer 1 to set the baud rate. 921.6 kHz divided by 32 gives 28,800 Hz. Timer 1 in mode 2 is used to set the baud rate.

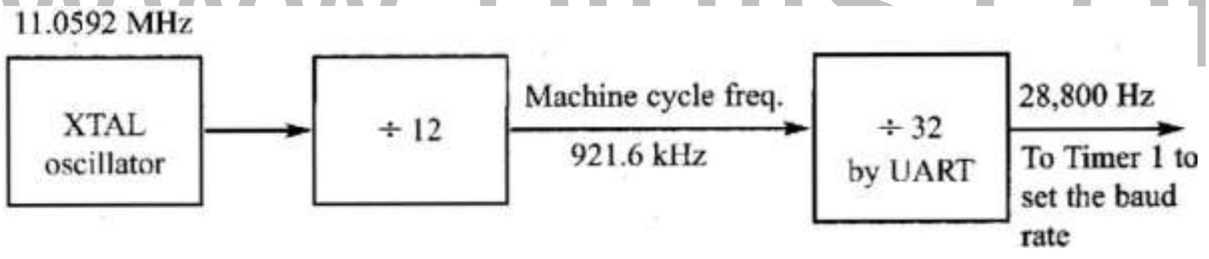


Figure 5.2.1 Frequency required to set the Baud rate

[Source: "The 8051Microcontroller and Embedded Systems: Using Assembly and C" by Mohamed Ali Mazda, Janice Gillespie Mazda, robin McKinley, pg.no.288]

CALCULATION OF BAUD RATE:

In serial communication if data transferred with a baud rate of 9600 and XTAL used is 11.0592 MHz, then following steps to be followed to find the TH1 value to be loaded.

Clock frequency of timer clock: f = (11.0592 MHz / 12)/32 = 28,800Hz

Time period of each clock tick: T0 = 1/f = 1/28800

Duration of timer : n*T0 (n is the number of clock ticks)

9600 baud ->duration of 1 symbol: 1/9600

1/9600 = n*T0 = n*1/28800

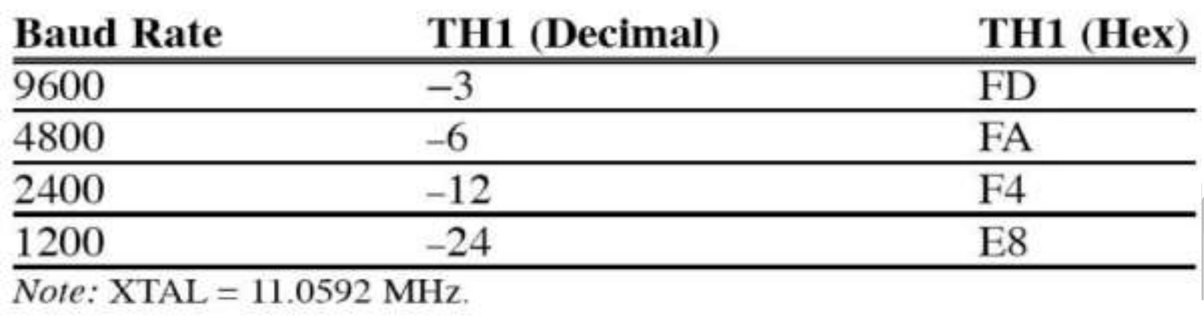
n = f/9600 = 28800/9600 = 3 ->TH1 =-3

Similarly, for baud 2400

n = f/2400 = 12 ->TH1 = -12

BAUD RATE SELECTION

Baud rate is selected by timer1 and when Timer 1 is used to set the baud rate it must be programmed in mode 2 that is 8-bit, auto-reload. To get baud rates compatible with the PC, we must load TH1 with the values shown in Table 5.2.1.



Baud Rate	TH1 (Decimal)	TH1 (Hex)
9600	–3	FD
4800	–6	FA
2400	–12	F4
1200	–24	E8

Note: XTAL = 11.0592 MHz.

Table 5.2.1 Timer 1 THI register values for different baud rates

[Source: "The 8051Microcontroller and Embedded Systems: Using Assembly and C" by Mohamed Ali Mazda, Janice Gillespie Mazda, robin McKinley, pg.no.287]

REGISTERS FOR SERIAL COMMUNICATION

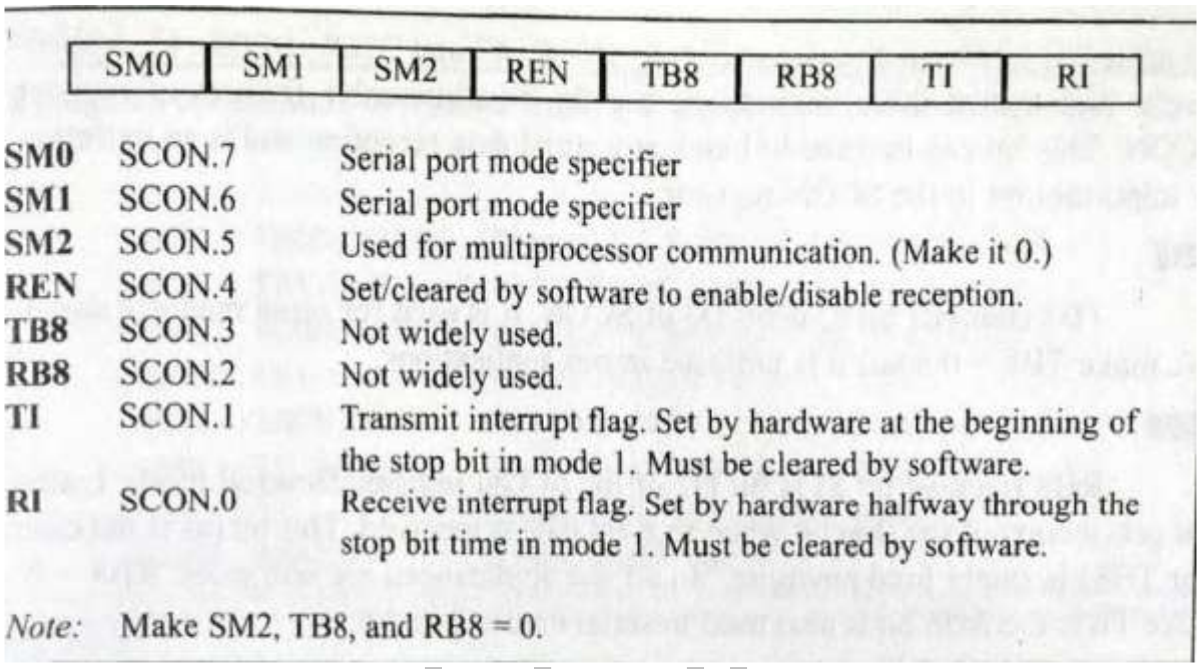
SBUF (SERIAL BUFFER) REGISTER:

It is an 8 bit register used solely for serial communication in the 8051.A byte of data to be transferred via the TX line must be placed in the SBUF register. SBUF holds the byte of data when it is received by the Rd. line. It can be accessed like any other register.

When a byte is written, it is framed with the start and stop bits and transferred serially via the TX pin and when the bits are received serially via Rd., it is defamed by eliminating the stop and start bits, making a byte out of the data received, and then placing it in the SBUF.

SCON (SERIAL CONTROL) REGISTER:

It is an 8 bit register used to program start bit, stop bit, and data bits of data framing, among other things.



SM0	SM1	SM2	REN	TB8	RB8	TI	RI

SM0	SCON.7	Serial port mode specifier
SM1	SCON.6	Serial port mode specifier
SM2	SCON.5	Used for multiprocessor communication. (Make it 0.)
REN	SCON.4	Set/cleared by software to enable/disable reception.
TB8	SCON.3	Not widely used.
RB8	SCON.2	Not widely used.
TI	SCON.1	Transmit interrupt flag. Set by hardware at the beginning of the stop bit in mode 1. Must be cleared by software.
RI	SCON.0	Receive interrupt flag. Set by hardware halfway through the stop bit time in mode 1. Must be cleared by software.

Note: Make SM2, TB8, and RB8 = 0.

Figure 5.2.2 SCON Register

[Source: "The 8051Microcontroller and Embedded Systems: Using Assembly and C" by Mohamed Ali Mazda, Janice Gillespie Mazda, robin McKinley, pg.no.289]

STEPS TO SEND DATA SERIALLY:

1.Set baud rate by loading TMOD register with the value 20H, this indicating timer 1 in mode 2 (8-bit auto-reload) to set baud rate

2. The TH1 is loaded with proper values to set baud rate for serial data transfer

3. The SCON register is loaded with the value 50H, indicating serial mode 1, where an 8- bit data is framed with start and stop bits

4. TR1 is set to 1 to start timer 1

5. TI is cleared by CLR TI instruction

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

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

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

Program to transfer letter "A" serially at 9800baud, continuously:

```
MOV TMOD, #20H               ; timer 1, mode 2(auto
reload) MOV TH1, #-3         ; 9600 baud rate
MOV SCON, #50H               ; 8-bit, 1 stop, REN enabled
SETB TR1                     ; start timer 1
AGAIN: MOV SBUF, #"A"        ; letter "A" to transfer
HERE: JNB TI, HERE           ; wait for the last bit
CLR TI                       ; clear TI for next
char
SJMP AGAIN                   ; keep sending A
```

IMPORTANCE OF THE TI FLAG:

Check the TI flag bit, we know whether or not 8051 is ready to transfer another byte. TI flag bit is raised by the 8051 after transfer of data. TI flag is cleared by the programmer by instruction like "CLR TI". When writing a bite into SBUF, before the TI flag bit is raised, it may lead to loss of a portion of the byte being transferred

STEPS TO RECEIVE DATA SERIALLY:

1. Set baud rate by loading TMOD register with the value 20H, this indicating timer 1 in mode 2 (8-bit auto-reload) to set baud rate.
2. The TH1 is loaded with proper values to set baud rate
3. The SCON register is loaded with the value 50H, indicating serial mode 1, where an 8- bit data is framed with start and stop bits
4. TR1 is set to 1 to start timer 1
5. RI is cleared by CLR RI instruction
6. The RI flag bit is monitored with the use of instruction JNB Roux to see if an entire character has been received yet
7. When RI is raised, SBUF has the byte; its contents are moved into a safe place
8. To receive next character, go to step 5

Program to receive bytes of data serially, and put them in P2, set the baud rate at 9600, 8-bit data, and 1 stop bit:

```
MOV TMOD, #20H              ; timer 1, mode 2(auto
reload) MOV TH1, #-3        ; 9600 baud rate
MOV SCON, #50H              ; 8-bit, 1 stop, REN enabled
SETB TR1                    ; start timer 1
HERE: JNB RI, HERE          ; wait for char to come in
MOV A, SBUF                 ; saving incoming byte in A
MOV P2, A                   ; send to port 1
CLR RI                      ; get ready to receive next byte
SJMP HERE                   ; keep getting data
```

IMPORTANCE OF THE RI FLAG BIT:

It receives the start bit, next bit is the first bit of the character about to be received. When the last bit is received, a byte is formed and placed in SBUF. When stop bit is received, it makes RI = 1 indicating entire character byte has been received and can be read before overwritten by next data. When RI=1, received byte is in the SBUF register, copy SBUF contents to a safe place. After the SBUF contents are copied the RI flag bit must be cleared to 0.