5 Designing Embedded Systems with 8bit Microcontrollers—*8051*

LEARNING OBJECTIVES

- ✓ *Understand the different factors that need to be considered while selecting a microcontroller for an embedded design*
- ✓ *Know why 8051 is the popular choice for low cost low performance embedded system design*
- ✓ *Learn the A to Z of 8051 microcontroller architecture*
- ✓ *Learn the Internals of the 8051 microcontroller*
- ✓ *Learn the program memory and internal data memory organisation of 8051*
- ✓ *Learn the Paged Data memory access and Von-Neumann memory model implementation for 8051*
- ✓ *Learn about the organisation of lower 128 bytes RAM for data memory, upper 128 bytes RAM for SFR and upper 128bytes RAM for Internal Data memory (IRAM)*
- ✓ *Learn about the CPU registers and general purpose registers of 8051*
- ✓ *Learn about the Oscillator unit and speed of execution of 8051*
- ✓ *Learn about the different I/O ports, organisation of the ports, the internal implementation of the port pins, the different registers associated with the ports and the operation of ports*
- ✓ *Learn about interrupts and its significance in embedded applications*
- ✓ *Learn about the Interrupt System of 8051, the interrupts supported by 8051, interrupt priorities, different registers associated with configuring the interrupts, Interrupt Service Routine and their vector address*
- ✓ *Learn about the Timer/Counter units supported by 8051 and configuring the Timer unit for Timer/Counter operation. Learn the different registers associated with timer units and the different modes of operations supported by Timer/Counter units*
- ✓ *Learn about the Serial Port of the 8051, the different control, status and data registers associated with it*
- ✓ *Learn the different modes of operations supported by the Serial port and setting the baudrate for each mode*
- ✓ *Learn about the Power-On Reset circuit implementation for 8051*
- ✓ *Learn about the different power saving modes supported by 8051*
- ✓ *Learn the difference between 8051 and 8052*

A recent survey on the microcontroller industry reveals that 8bit microcontrollers account for more than 40% of the total sales in the microcontroller industry. Among the 8bit microcontrollers, the *8051* family is the most popular, cost effective and versatile device offering extensive support in the embedded application domain. Looking back to the history of microcontrollers you can find that the 8bit microcontroller industry has travelled a lot from its first popular model *8031AH* built by Intel in 1977 to the advanced 8bit microcontroller built by Maxim/Dallas recently, which offers high performance 4 Clock, 75MHz operation *8051* microcontroller core (Remember the original 8031 core was 12 Clock with support for a maximum system clock of 6MHz, so a performance improvement of 3 times on the original version in execution) with extensive support for networking by integrating 10/100 Ethernet MAC with IEEE 802.3 MMI, CAN bus for automotive application support, RS-232 C interface for legacy serial applications, SPI serial interface for board level device inter connect, 1-wire interface for connecting to low cost multi-drop sensors and actuators, and 16MB addressing space for code and data memory.

5.1 FACTORS TO BE CONSIDERED IN SELECTING A CONTROLLER

Selection of a microcontroller for any application depends on some design factors. A good designer finalises his selection based on a comparative study of the design factors. The important factors to be considered in the selection process of a microcontroller are listed below.

5.1.1 Feature Set

The important queries related to the feature set are: Does the microcontroller support all the peripherals required by the application, say serial interface, parallel interface, etc.? Does it satisfy the general I/O port requirements by the application? Does the controller support sufficient number of timers and counters? Does the controller support built-in ADC/DAC hardware in case of signal processing applications? Does the controller provide the required performance?

5.1.2 Speed of Operation

Speed of operation or performance of the controller is another important design factor. The number of clocks required per instruction cycle and the maximum operating clock frequency supported by the processor greatly affects the speed of operation of the controller. The speed of operation of the controller is usually expressed in terms of million instructions per second (MIPS).

5.1.3 Code Memory Space

If the target processor/controller application is written in C or any other high level language, does the controller support sufficient code memory space to hold the compiled hex code (In case of controllers with internal code memory)?

5.1.4 Data Memory Space

Does the controller support sufficient internal data memory (on chip RAM) to hold run time variables and data structures?

5.1.5 Development Support

Development support is another important factor for consideration. It deals with–Does the controller manufacture provide cost-effective development tools? Does the manufacture provide product samples for prototyping and sample development stuffs to alleviate the development pains? Does the controller support third party development tools? Does the manufacture provide technical support if necessary?

5.1.6 Availability

Availability is another important factor that should be taken into account for the selection process. Since the product is entirely dependent on the controller, the product development time and time to market the product solely depends on its availability. By technical terms it is referred to as *Lead time*. Lead time is the time elapsed between the purchase order approval and the supply of the product.

5.1.7 Power Consumption

The power consumption of the controller should be minimal. It is a crucial factor since high power requirement leads to bulky power supply designs. The high power dissipation also demands for cooling fans and it will make the overall system messy and expensive. Controllers should support idle and power down modes of operation to reduce power consumption.

5.1.8 Cost

Last but not least, cost is a big deciding factor in selecting a controller. The cost should be within the reachable limit of the end user and the targeted user should not be *high tech*. Remember the ultimate aim of a product is to *gain marginal benefit*.

5.2 WHY *8051* MICROCONTROLLER

8051 is a very versatile microcontroller featuring powerful Boolean processor which supports bit manipulation instructions for real time industrial control applications. The standard *8051* architecture supports 6 interrupts (2 external interrupts, 2 timer interrupts and 2 serial interrupts), two 16bit timers/counters, 32 I/O lines and a programmable full duplex serial interface. Another fascinating feature of *8051* is the way it handles interrupts. The interrupts have two priority levels and each interrupt is allocated fixed 8 bytes of code memory. This approach is very efficient in real time application. Though *8051* is invented by Intel, today it is available in the market from more than 20 vendors and with more than 100 variants of the original *8051* flavour, supporting CAN, USB, SPI and TCP/IP interfaces, integrated ADC/DAC, LCD Controller and extended number of I/O ports. Another remarkable feature of *8051* is its low cost. The *8051* flash microcontroller (*AT89C51*) from Atmel is available in the market for less than 1US$ per piece. So imagine its cost for high volume purchases.

5.3 DESIGNING WITH *8051*

5.3.1 The *8051* Architecture

The basic *8051* architecture consist of an 8bit CPU with Boolean processing capability, oscillator driver unit, 4K bytes of on-chip program memory, 128 bytes of internal data memory, 128 bytes of special function register memory area, 32 general purpose I/O lines organised into four 8bit bi-directional ports, two 16bit timer units and a full duplex programmable UART for serial data transmission with configurable baudrates. Figure 5.1 illustrates the basic 8051 architecture.

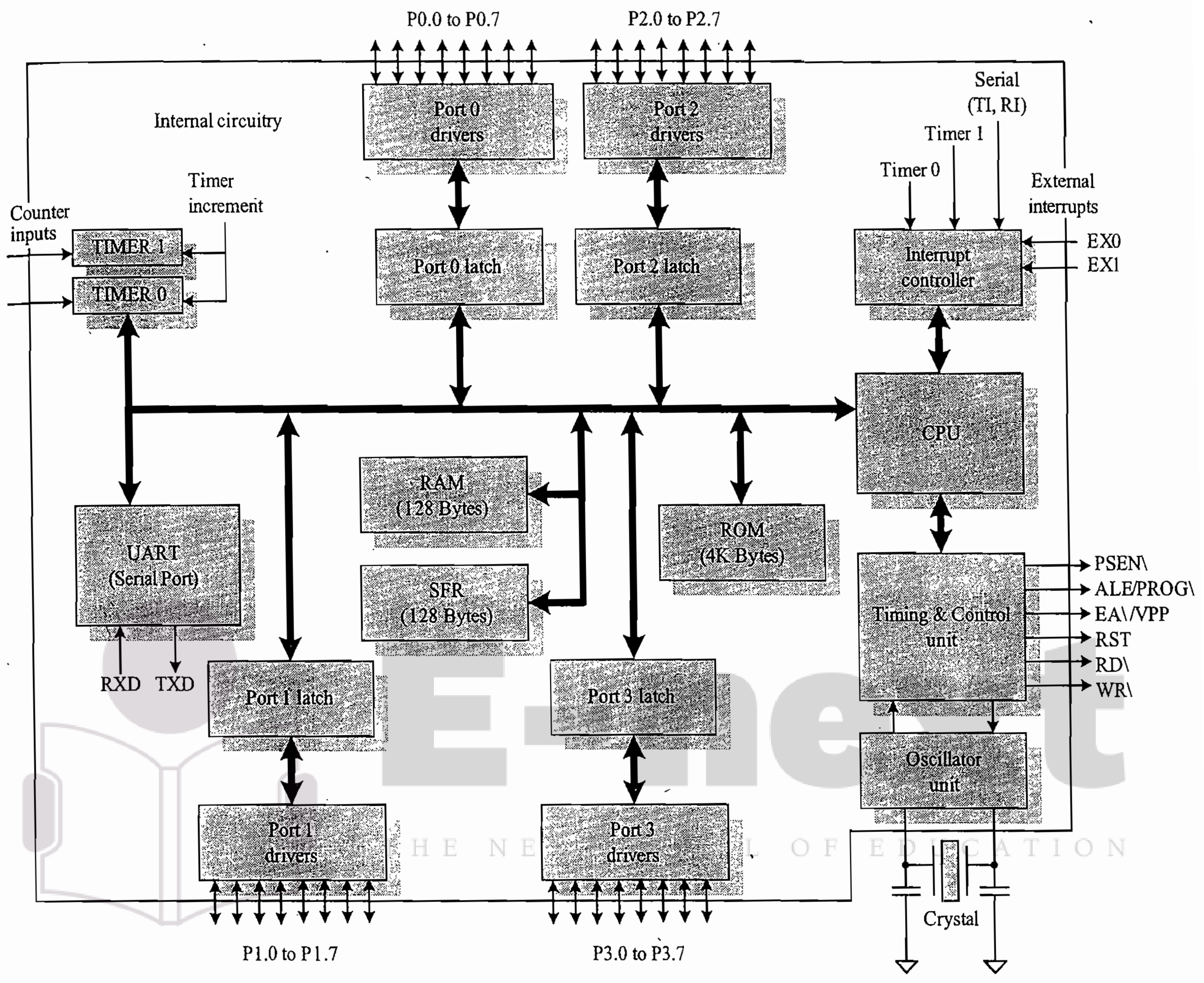


Fig. 5.1 *8051* **Architecture—Block diagram representation**

5.3.2 The Memory Organisation

8051 is built around the *Harvard* processor architecture. The program and data memory of *8051* is logically separated and they physically reside separately. Separate address spaces are assigned for data memory and program memory. *8051*'s address bus is 16bit wide and it can address up to 64KB (2^{16}) memory.

5.3.2.1 The Program (Code) Memory The basic *8051* architecture provides lowest 4K bytes of program memory as on-chip memory (built-in chip memory). In *8031*, the ROMless counterpart of *8051*, all program memory is external to the chip. Switching between the internal program memory and external program memory is accomplished by changing the logic level of the pin External Access (EA\). Tying EA\ pin to logic 1 (V_{CC}), configures the chip to execute instructions from program memory up to 4K (program memory location up to 0FFFH) from internal memory and 4K (program memory location from 1000H) onwards from external memory, while connecting EA\ pin to logic 0 (GND) configures the chip to external program execution mode, where the entire code memory is executed from the external memory. Remember External Access pin is an active low pin (Normally referred as EA\). The control signal for external program memory execution is PSEN\ (Program Strobe Enable). For internal program memory fetches PSEN\ is not activated. For 8031 controller without on-chip memory, the PSEN\ signal is always activated during program memory execution. The External Access pin (EA\) configuration and the corresponding code memory access are illustrated in Fig. 5.2.

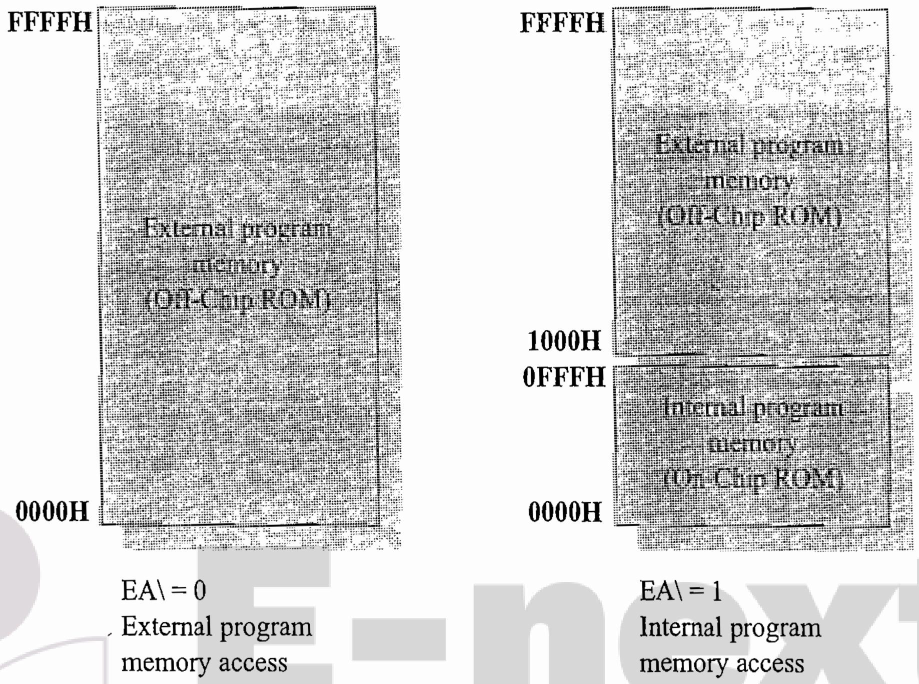

FFFFH
External program memory (Off-Chip ROM)
0000H
EA\ = 0
External program memory access
FFFFH
External program memory (Off-Chip ROM)
1000H
0FFFH
Internal program memory (On-Chip ROM)
0000H
EA\ = 1
Internal program memory access

Fig. 5.2 ***8051* Program memory organisation**

If the program memory is external, 16 I/O lines are used for accessing the external memory. Port 0 and Port 2 are used for external memory accessing. Port 0 serves as multiplexed address/data bus for external program memory access. Similar to the *8085* microprocessor, Port 0 emits the lower order address first. This can be latched to an 8bit external latch with the Address Latch Enable (ALE) signal emitted by *8051*. Once the address outing is over, Port 0 functions as input port for data transfer from the corresponding memory location. The address from which the program instruction to be fetched is supplied by the 16bit register, Program Counter (PC), which is part of the CPU. The Program Counter is a 16bit register made up of two 8bit registers. The lower order byte of program counter register is held by the PCL register and higher order by the PCH register. PCL and PCH in combination serve as a 16bit register. During external program memory fetching, Port 0 emits the contents of PCL and Port 2 emits the contents of PCH register. Port 0 emits the contents of PCL only for a fixed duration allowing the external latch to hold the content on arrival of the ALE signal. Afterwards Port 0 goes into high impedance state, waiting for the arrival of data from the corresponding memory location of external memory. Whereas Port 2 continues emitting the contents of PCH register throughout the external memory fetch. Once the PSEN\ signal is active, data from the program memory is clocked into Port 0. Remember, during external program memory access Port 0 and Port 2 are dedicated for it and cannot be used as general purpose I/O ports. The interfacing of an external program memory chip is illustrated in Fig. 5.3.

5.3.2.2 The Data Memory The basic 8051 architecture supports 128 bytes of internal data memory and 128 bytes of *Special Function Register* memory. Special Function Register memory is not available for the user for general data memory applications. The address range for internal user data memory is 00H to 7FH. Special Function Registers are residing at memory area 80H to FFH. *8051* supports interface for 64 Kbytes of external data memory. The control signals used for external data memory

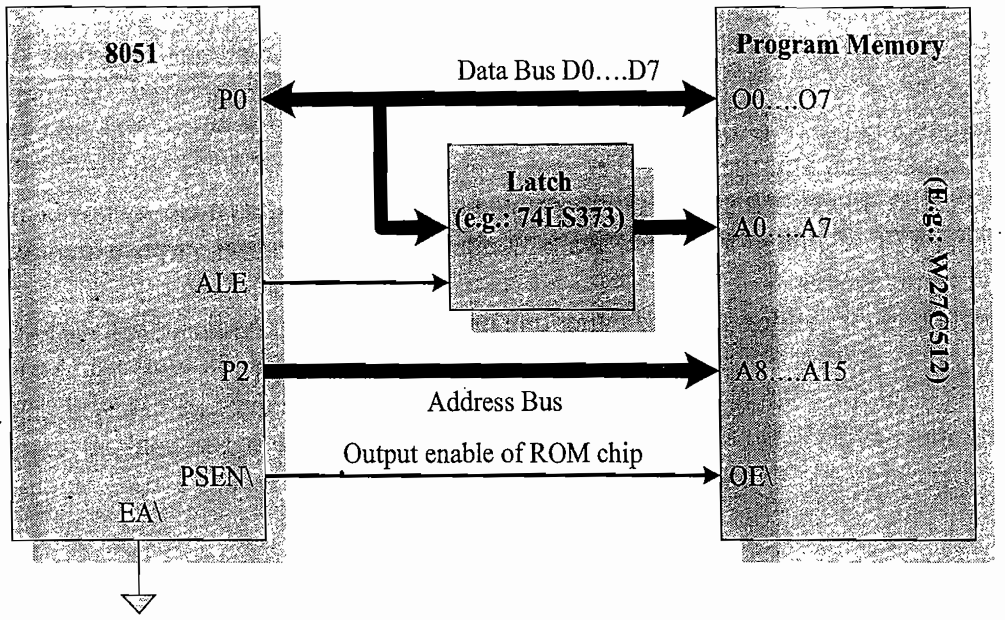


Fig. 5.3 ***8051* External Program Memory chip (ROM) interfacing**

access are RD\ and WR\ and the 16bit register holding the address of external data memory address to be accessed is *Data Pointer* (*DPTR*). Similar to the Program Counter, the Data Pointer is also made up of two 8bit registers, namely, DPL (holding the lower order 8bit) and DPH (holding the higher order 8bit). The program counter is not accessible to the user whereas DPTR is accessible to the user and the contents of DPTR register can be modified. In external data memory operations, Port 0 emits the content of DPL and Port 2 emits the content of DPH. Port 0 is address/data multiplexed in external data memory operations also. The internal and external data memory model of *8051* is diagrammatically represented in Fig. 5.4.

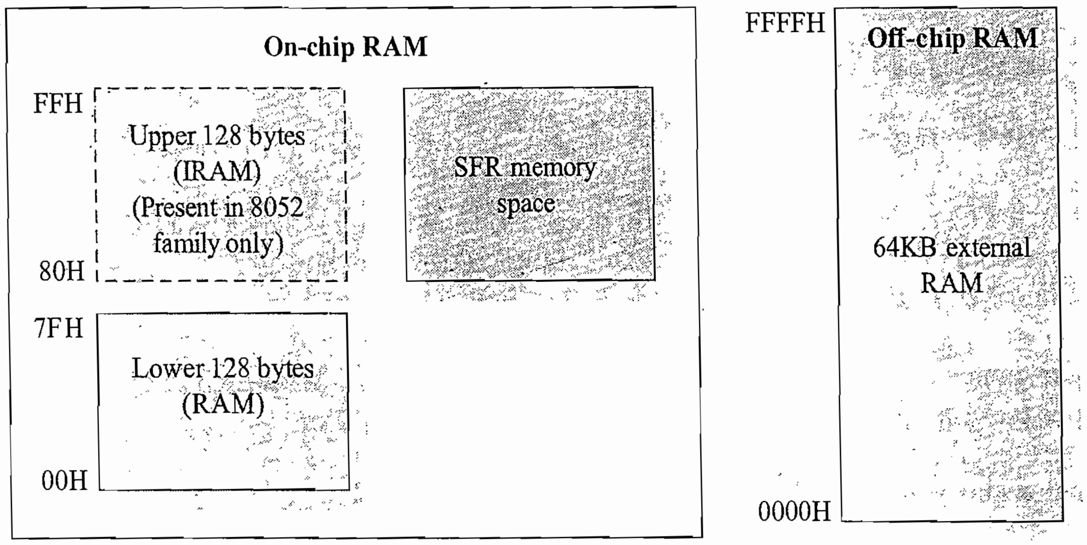


Fig. 5.4 **Data memory map for *8051***

Internal data memory addresses are always one byte long. So it can accommodate up to 256 bytes of internal data memory (Ranging from 0 to 255). However the addressing techniques used in *8051* can accommodate 384 bytes using a simple memory addressing technique. The technique is: Direct addressing of data memory greater than 7FH will access one memory space, namely Special Function Register memory and indirect addressing of memory address greater than 7FH will access another memory space, the upper 128 bytes of data memory (Direct and indirect memory addressing will be discussed in detail in a later section). Remember these techniques will work only if the upper data memory is physically implemented in the chip. The basic version of *8051* does not implement the upper data memory physically. However the *8052* family implements the upper data memory physically in the chip and so the upper 128 byte memory is also available for the user as general purpose memory, if accessed through indirect addressing.

External data memory address can be either one or two bytes long. As described earlier, Port 0 emits the lower order 8bit address and, if the memory address is two bytes and if it ranges up to 64K, the entire bits of Port 2 is used for holding the higher order value of data memory address. If the memory range is 32K, only 7 bits of Port 2 is required for addressing the memory. For 16K, only 6 lines of Port 2 are required for interfacing and so on. Thereby you can save some port pins of Port 2. The interfacing of an external data memory chip is illustrated Fig. 5.5.

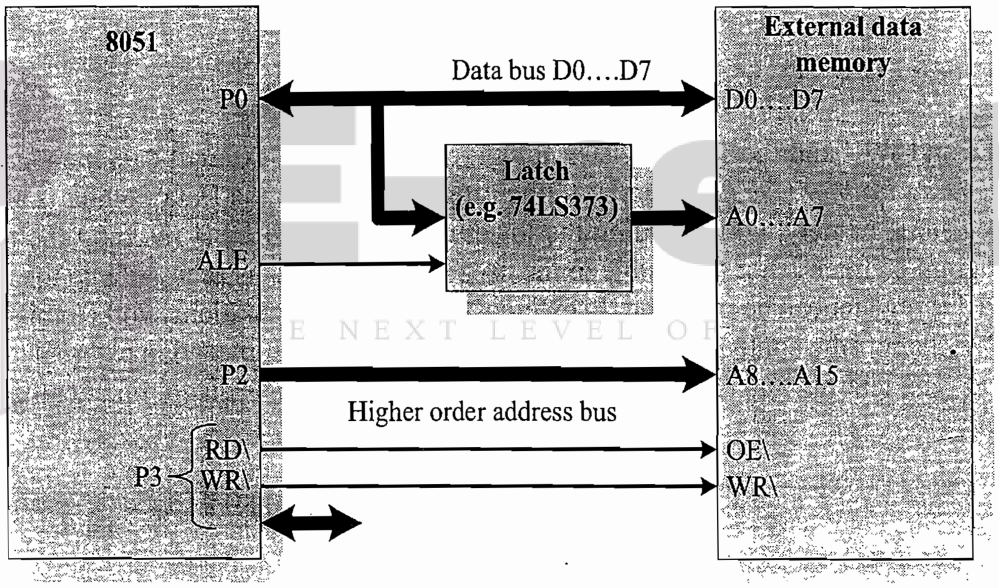


Fig. 5.5 **External Data memory access**

5.3.2.3 Paged Data Memory Access In paged mode addressing, the memory is arranged like the lines of a notebook. The notebook may contain 100 to 200 pages and each page may contain a fixed number of lines. You can access a specific line by knowing its page number and the line number. Memory can also arrange like the lines of a notebook. By using 8bit address, memory up to 256 bytes can be accessed. Imagine the situation where the memory is stacked of 256 bytes each. You can use port pins (High order address rule) to signal the page number and the lower order 8 bits to indicate the memory location corresponding to that page.

For example, take the case where paging is done using the port pin P2.0 and port 0 is used for holding the lower address. The memory range will be

Page selector (P2.0)	Lower order address	Address range
0	00H to FFH	000H to 0FFH
1	00H to FFH	100H to 1FFH

5.3.2.4 The Von-Neumann Memory Model for 8051 The code memory and data memory of *8051* can be combined together to give the Von-Neumann architectural benefit for *8051*. A single memory chip with read/write option can be used for this. The program memory can be allocated to the lower memory space starting from 0000H and data memory can be assigned to some other specific area after the code memory. For program memory fetching and data memory read operations combine the PSEN\ and RD\ signals using an AND gate and connect it to the Output Enable (OE\) signal of the memory chip as shown in Fig. 5.6.

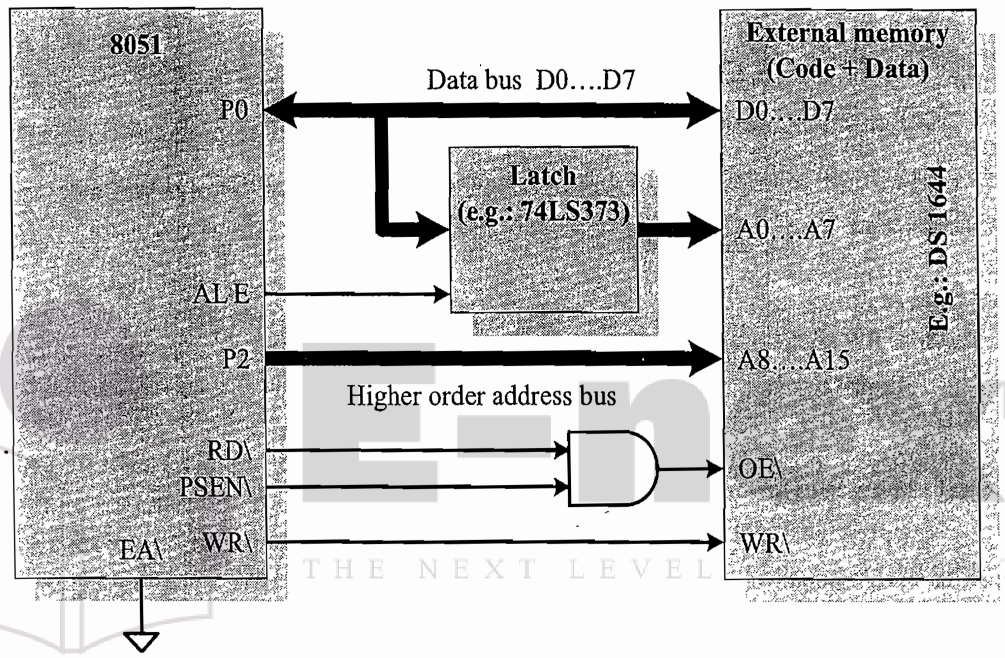


Fig. 5.6 Combining Code memory and Data memory

The Von-Neumann memory model is very helpful in evaluation boards for the controller, which allows modification of code memory on the fly. The major drawbacks of using a single chip for program and data memory are

- Accidental corruption of program memory
- Reduction in total memory space. In separate program and data memory model the total available memory is 128KB (64KB program memory + 64KB Data memory) whereas in the combined model the total available memory is only 64KB

5.3.2.5 Lower 128 Byte Internal Data Memory (RAM) Organisation This memory area is volatile; meaning the contents of these locations are not retained on power lose. On power up these memory locations contain random data. The lowest 32 bytes of RAM (00H to 1FH) are grouped into 4 banks of 8 registers each. These registers are known as R0 to R7 registers which are used as temporary data storage registers during program execution. The effective usage of these registers reduces the code memory requirement since register instructions are shorter than direct memory addressing instructions. The next 16 bytes of RAM with address 20H to 2FH is a bit addressable memory area. It accommodates 128 bits (16 bytes x 8), which can be accessed by direct bit addressing. The address of bits ranges from 00H to 7FH. This is very useful since 8051 is providing extensive support for Boolean operations (Bit Manipulation Operations). Also it saves memory since flag variables can be set up with these bits and there is no need to waste one full byte of memory for setting up a flag variable. These 16 bytes can also be used as byte variables. The context in which these bytes are used as either byte variable or bit variable is determined by the type of instruction. If the instruction is a bit manipulation instruction and the operand is given as a direct address in the range 00H to 7FH, it is treated as a bit variable. The lower 128 bytes of internal RAM for *8051* family members is organised as shown in Fig. 5.7.

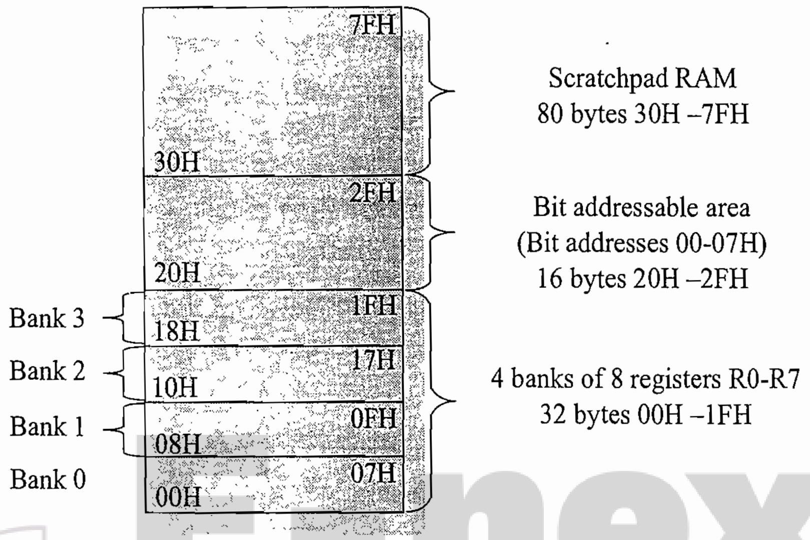


Fig. 5.7 **Internal 128 bytes of lower order data memory organisation**

Remember the byte-wise storage and bitwise storage in the area 20H to 2FH shares a common physical memory area and you cannot use this for both byte storage and bit storage simultaneously. If you do so, depending on the usage, the byte variables and bit variables may get corrupted and it may produce unpredicted results in your application. This is explained more precisely using the following table.

B7	B6	B5	B4	B3	B2	B1	B0	Byte Address
7FH	7EH	7DH	7CH	7BH	7AH	79H	78H	2FH
77H	76H	75H	74H	73H	72H	71H	70H	2EH
6FH	6EH	6DH	6CH	6BH	6AH	69H	68H	2DH
67H	66H	65H	64H	63H	62H	61H	60H	2CH
5FH	5EH	5DH	5CH	5BH	5AH	59H	58H	2BH
57H	56H	55H	54H	53H	52H	51H	50H	2AH
4FH	4EH	4DH	4CH	4BH	4AH	49H	48H	29H
47H	46H	45H	44H	43H	42H	41H	40H	28H
3FH	3EH	3DH	3CH	3BH	3AH	39H	38H	27H
37H	36H	35H	34H	33H	32H	31H	30H	26H
2FH	2EH	2DH	2CH	2BH	2AH	29H	28H	25H
27H	26H	25H	24H	23H	22H	21H	20H	24H
1FH	1EH	1DH	1CH	1BH	1AH	19H	18H	23H

17H	16H	15H	14H	13H	12H	11H	10H	22H
0FH	0EH	0DH	0CH	0BH	0AH	09H	08H	21H
07H	06H	05H	04H	03H	02H	01H	00H	20H

B0, B1, B3 …B7 represents the bit addresses. Now let's have a look at the following piece of Assembly code:

```
ORG 0000H          ; Reset Vector. Assembler directive
LJMP 0050H         ; Jump to location 0050H. Avoid conflicts in ISR Code
ORG 0050H          ; Location 0050H. Assembler directive
MOV 20H, #00H      ; Clear memory location 20H
SETB 01H           ; Store logic 1 in bit address 01H.
MOV 20H, #0F0H     ; Load memory location 20H with F0H
END                ; End of program. Assembler directive
```

This piece of assembly code sets the bit with bit address 01H and loads the memory location 20H with value F0H. Don't worry about the different instructions used here. We will discuss about the 8051 instruction set in a later chapter. The *MOV 20H, #00H* instruction clears the memory location 20H. The *SETB 01H* instruction stores logic 1 in the bit address 01H. In reality the bit address 01H is the bit 1 of the memory location pointed by address 20H. Executing the instruction *SETB 01H* changes the contents of memory location 20H to 02H (00000010b. Only bit 1 is in logic 1 state). The instruction *MOV 20H, #F0H* alters the content of memory location 20H with F0H (11110000b). This overwrites the information held by bit address 01H and leads to data corruption. So be careful while using bitwise storage and byte-wise storage simultaneously.

The next 80 bytes of RAM with address space 30H to 7FH is used as general purpose scratchpad RAM. Though memory spaces 00H to 2FH have specific usage, they can also be used as general purpose scratchpad (Read/Write) RAM. The lower 128 byte RAM can be accessed by either direct addressing or indirect addressing.

5.3.2.6 The Upper 128 bytes RAM (Special Function Registers) The upper 128 bytes of RAM when accessed by direct addressing, accesses the Special Function Registers (SFRs). SFRs include port latches, status and control bits, timer control and value registers, CPU registers, stack pointer, accumulator, etc. Some of the SFR registers are only byte level accessible and some of them are both byte-wise and bit-wise accessible. SFRs with address ends in 0H and 8H are both bit level and byte level accessible. In the standard 8051 architecture, among the 128 bytes, only a few bytes are occupied by the SFR and the rest are left unused and are reserved for future implementations. The table given below explains the SFR implementation for standard *8051* architecture.

Memory Address	SFR Name	Memory Address	SFR Name	Memory Address	SFR Name
80H	Port 0	8AH	TL0	A0H	Port 2
81H	SP	8BH	TL1	A8H	IE
82H	DPL	8CH	TH0	B0H	Port 3
83H	DPH	8DH	TH1	B8H	IP
87H	PCON	90H	Port 1	D0H	PSW
88H	TCON	98H	SCON	E0H	A
89H	TMOD	99H	SBUF	F0H	B

SFR memory is not available to the user for general purpose scratchpad RAM usage. However the user can modify the contents of some of the SFR according to the program requirements. Some of the SFRs are Read Only. Some of the SFR memory spaces are not implemented in the basic 8051 version. They are reserved for future use and users are instructed not to do anything with this reserved SFR space. Reading from the unimplemented SFR memory address returns random data and writing to this memory location will not produce any effect. Each of the SFRs will be discussed in detail in the sections covering their usage.

5.3.2.7 Upper 128 Bytes of Scratchpad RAM (IRAM) Variants of *8051* and the *8052* architecture where the upper 128 bytes of RAM are physically implemented in the chip can be used as general purpose scratchpad RAM by indirect addressing. They are generally known as IRAM. The address of IRAM ranges from 80H to FFH and the access is indirect. Registers R0 and R1 are used for indirect addressing. For example, for accessing the IRAM located at address 80H, load R0 or R1 with 80H and use the indirect memory access instruction. The following piece of assembly code illustrates the same.

```
MOV   R0,#80H  ; Load IRAM address 80H in indirect register
MOV   A,@R0    ; Load Accumulator with IRAM content at address 80H
```

5.3.3 Registers

Registers of *8051* can be broadly classified into CPU Registers and Scratchpad Registers.

5.3.3.1 CPU Registers Accumulator, B register, Program Status Word (PSW), Stack Pointer (SP), Data Pointer (DPTR, Combination of DPL and DPH), and Program Counter (PC) constitute the CPU registers. They are described in detail below.

Accumulator (ACC) (SFR-E0H) It is the most important CPU register which acts as the heart of all CPU related Arithmetic operations. Accumulator is an implicit operand in most of the arithmetic operations. Accumulator is a bit addressable register.

ACC.7	ACC.6	ACC.5	ACC.4	ACC.3	ACC.2	ACC.1	ACC.0

B Register (SFR-F0H) It is a CPU register that acts as an operand in multiply and division operations. It also stores the remainder in division and MSB in multiplication Instruction. B can also be used as a general purpose register for programming.

Program Status Word (PSW) (SFR-D0H) It is an 8-bit, bit addressable Special Function register signalling the status of accumulator related operations and register bank selector for the scratch pad registers R0 to R7. The bit details of PSW register is given below.

PSW.7	PSW.6	PSW.5	PSW.4	PSW.3	PSW.2	PSW.1	PSW.0
CY	AC	F0	RS1	RS0	OV		P

The table given below explains the meaning and use of each bit.

Bit	Name	Explanation
CY	Carry flag	Sets when a carry occurs on the addition of two 8-bit numbers or when a borrow occurs on the subtraction of two 8-bit numbers.
AC	Auxiliary carry flag	Sets when a carry generated out of bit 3 (bit index starts from 0) on addition

F0	Flag 0	General purpose user programmable flag (PSW.5)
OV	Over flow	Sets when overflow occurs. OV is set if there is a carry-out of bit 6 but not out of bit 7, or a carry-out of bit 7 but not bit 6; otherwise OV is cleared. When adding signed integers, OV indicates a negative number produced as the sum of two positive operands, or a positive sum from two negative operands.
P	Parity flag	Set or cleared by hardware each instruction cycle to indicate an odd or even number of 1s in the accumulator. 'P' is set to 1 if the number of 1s in the accumulator content is odd else reset to 0.
PSW.1	General flag	User programmable general purpose bit
RS0 RS1	Register bank selector	The bit status and bank selected is given in the following table

The following table illustrates the possible combinations for the register bank select bits, the corresponding register bank number and the address for the scratchpad registers R0-R7 in the specified register bank.

RS1	RS0	Register Bank	Register Address
0	0	0	00H-07H
0	1	1	08H-0FH
1	0	2	10H-17H
1	1	3	18H-1FH

The power-on reset value for the bits RS1 and RS2 are 0 and the default register bank for scratchpad registers R0 to R7 is 0 and the address range for R0 to R7 is 00H to 07H. A programmer can change the register bank by changing the values of RS1 and RS0. The following piece of assembly code illustrates the selection of bank 2 for the scratchpad registers R0 to R7.

```
CLR RS0      ; Clear bit RS1
SETB RS1     ; Set bit RS1. Bank 2 is selected
```

Data Pointer (DPTR) (DPL: SFR-82H, DPH: SFR-83H) It is a combination of two 8-bit register namely DPL (Lower 8-bit holder of DPTR) and DPH (Higher order 8-bit holder of DPTR). DPTR holds the 16-bit address of the external memory to be read or written in external data memory operations. DPH and DPL can be used as two independent 8-bit general purpose registers for application programming.

Program Counter (PC) It is a 16-bit register holding the address of the code memory to be fetched. It is an integral part of the CPU and it is hidden from the programmer (It is not accessible to the programmer).

Stack Pointer (SP) (SFR-81H) It is an 8-bit register holding the current address of stack memory. Stack memory stores the program counter address, other memory and register values during a sub routine/function call. On power on reset the stack pointer register value is set as 07H. The stack pointer address and bank 0, address of R7 is same when the controller is at reset, so care should be taken for selecting SP address. It is the responsibility of the programmer to assign sufficient stack memory by entering the starting address of stack into the Stack Pointer register. Care should be taken to avoid the overflow of stacks and merging of stack memory with data memory. This will result in un-predicted program flow. The stack grows up in memory.

5.3.3.2 Scratchpad Registers (R0 to R7) The scratchpad registers R0 to R7 is located in the lower 32 bytes of internal RAM. It can be on one of the four banks, which is selected by the register selector bits RS0 and RS1 of the PSW register. On power on reset, by default, RS0 and RS1 are 0 and the default bank selected is bank 0. There are eight scratchpad registers and they are named as R0, R1...R7. The register names and their memory address corresponding to bank 0 are given below.

R7	R6	R5	R4	R3	R2	R1	R0
07H	06H	05H	04H	03H	02H	01H	00H

As the bank number changes the register address also offsets by the memory address bank number multiplied by 8. Though you can select between the register banks 0 and 3, there will be only one active register bank at a time and it depends on the RS0 and RS1 bits of Program Status Word (PSW). Registers R0 to R7 are used as general purpose working registers. R0 and R1 also handle the role of index addressing or indirect addressing register (@R0 and @R1 instructions). R0 and R1 can also be used for external memory access in place of DPTR, if the memory address is 8-bit wide ((MOVX A, @R0) – will be discussed later).

5.3.4 Oscillator Unit

The program execution is dependent on the clock and the oscillator unit is responsible for generating the clock signals. All *8051* family microcontrollers contain an on-chip oscillator. This contains all necessary oscillator driving circuits. The only external component required is a ceramic crystal resonator. The *8051* on chip oscillator circuit provides external interface option through two pins of the microcontroller, namely, XTAL1 and XTAL2.

If you are using a ceramic resonator, you can connect it across the XTAL1 and XTAL2 pins of the chip with two external capacitors. Capacitors with values 15pF, 22pF, 33pF, etc. are used with the crystal resonator. This is the cheapest solution since the total cost for a ceramic resonator and two capacitors is always less than a standalone oscillator module.

If an external stand alone oscillator unit is used, the output signal of the oscillator unit should be connected to the pin XTAL1 of the chip and the pin XTAL2 should be left unconnected for a CMOS* type microcontroller (80C51). For an NMOS** type microcontroller, the oscillator output signal should be connected to the Pin XTAL2 and the pin XTAL1 should be grounded (Fig. 5.8).

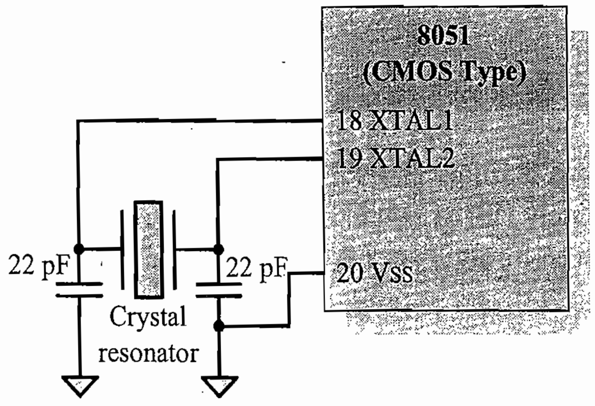


Fig. 5.8 **Circuit configuration for using on-chip oscillator**

* CMOS—Complementary metal–oxide–semiconductor field effect transistor technology for digital circuit design. CMOS features less power consumption and high logic density on an integrated circuit.

** NMOS—*n*-type metal-oxide-semiconductor field effect transistor technology for digital circuit design. It is an old technology and possesses the drawback of noise susceptibility and slow logic transition. In modern designs it is supplanted by CMOS technology.

You may be thinking why an oscillator circuit is required? The answer is–The microcontroller chip is made up of digital combinational and sequential circuits and they require a clock to drive the digital circuitry. The clock is supplied by this oscillator circuit and the operational speed of the chip is dependent on the clock speed.

5.3.4.1 Execution Speed The execution speed of the processor is directly proportional to the oscillator clock frequency. Increasing the clock speed will have direct impact on the speed of program execution. But the internal processor core design will always have certain limitations on the maximum clock frequency on which it can be operated. During program execution the instructions stored in the code memory is fetched, decoded and corresponding action is initiated. Each instruction fetching consists of the number of *machine cycles*. The instruction set of 8051 contains single cycle to four machine cycle instructions.

Each machine cycle is made up of a sequence of states called *T states*. The original 8051 processor's machine cycle consists of 6 T states and is named S1, S2, S3…S6. Each T states in turn consist of two oscillator periods (Clock cycles) and so one machine cycle contains 12 clock cycles. For a one machine cycle instruction to execute, it takes 12 clock cycles. If the system clock frequency is 12MHz, it takes 1microsecond (1µs) time to execute one machine cycle. The machine cycle, T state and clock cycle relationship is illustrated in the following Fig. 5.9.

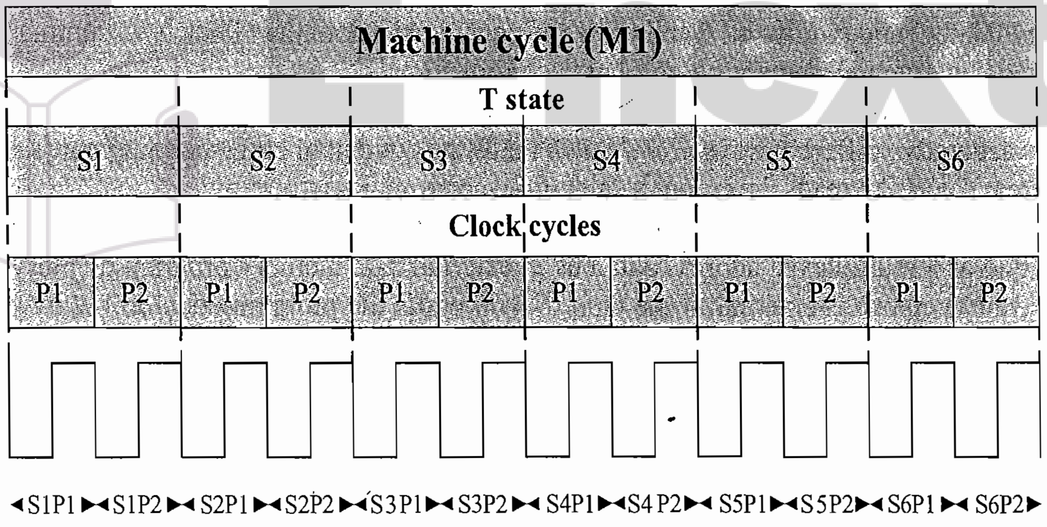


Fig. 5.9 **Machine Cycles, T state & clock periods**

5.3.5 Port

Port is a group of Input/Output (I/O) lines. Each port has its own port control unit, port driver and buffers. The original version of *8051* supports 32 I/O lines grouped into 4 I/O ports, consisting of 8 I/O lines per port. The ports are named as *Port 0*, *Port 1*, *Port 2* and *Port 3*. One output driver and one input buffer is associated with each I/O line. All four ports are bi-directional and an 8bit latch (Special Function Register) is associated with each port.

5.3.5.1 Port 0 PORT 0 is a bi-directional port, which is used as a multiplexed address/data bus in external data memory/program memory operations. Port 0 pin organisation is illustrated in Fig. 5.10.

Each pin of Port 0 possesses a bit latch which is part of the Special Function Register (SFR) for Port 0, P0. The latch is a D flip flop and it clocks in a logic value (either logic 1 or logic 0) from the internal

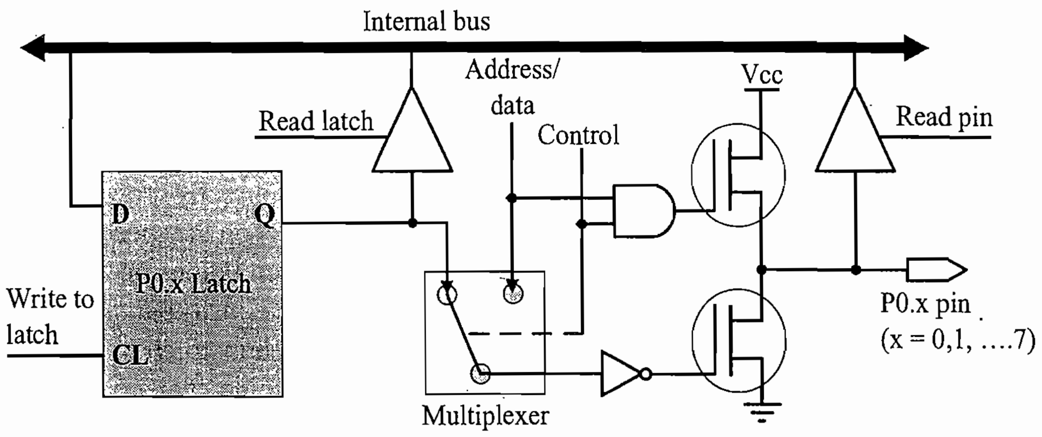


Fig. 5.10 **Port 0 pin organisation**

bus when a write to the corresponding latch signal is issued by the internal control unit. Apart from the write to latch operation you can read the contents of the corresponding Port 0 Pin latch by activating a *Read Latch* signal. The *Read Latch* signal is asserted on executing an instruction with a read from the corresponding 'port latch' instruction (e.g. *ANL P0*, A reads the P0 latch, logical AND it with accumulator and loads the latch with the result. Similarly, the instruction *MOV P0.0,C* reads the Port 0 byte (8 bits of the P0 latch) and modify bit 0 and write the new byte back to the P0 latch. In summary all '*Read-Modify-Write*' instructions generate *Read Latch* control signal).

The *Read Pin* control signal enables reading the status of a port pin. The *Read Latch* and *Read Pin* operations act on two different ways. *Read Latch* reads the content of corresponding port's SFR/SFR bit latch whereas *Read Pin* reads the present state of the corresponding port pin.

Port 0 is designed in a way to operate in different modes. It acts as an I/O port in normal mode of operation and as a multiplexed address data bus in external data memory/program memory operations. If the program memory is external to the chip, *Port 0* emits the program counter low byte in external program memory operation for specific time duration and then acts as an input port to fetch the instruction from the address specified by the program counter. In external data memory operations P0 emits the lower order byte of the DPTR Register (DPL).

If you look back to the Port 0 pin organisation, you can see that during external memory related operations the multiplexer disconnects the port 0 bit output line from its corresponding bit latch and directly connect it to the ADDRESS/DATA line and the output driver circuitry is driven according to the ADDRESS/DATA line and the control.

The output drivers of Port 0 are formed by two FETs, out of which the top FET functions as the internal port pull-up. The pull-up FET driver for Port 0 is active only when the address line is emitting 1s during external memory operations. The pull-up FET will be off on all other conditions and the Port 0 pins which are used as output pins will become open drain (Open Collector for TTL logic). On writing a 1 to the corresponding port bit SFR latch, the bottom FET is turned off and the pin floats and it enters in a high impedance state.

In order to make any Port 0 pin an input pin, a logic 1 should be written to the corresponding Port 0 SFR latch bit and an external pull-up resistor (in the range of Kilo ohms, typically 4.7K) should be connected across the corresponding Port 0 pin and power supply V_{CC}, which will act as a bypass to the internal pull-up FET.

If Port 0 is used for external memory or device interfacing, it should be equipped with external pull-up resistors to provide noise immunity to Port 0 data lines.

When configured as O/p port by writing 1s to the Port 0 SFR, all Port 0 pins floats and it is said to be in a high impedance state. Port 0 is a true bi-directional port.

Port 0 SFR (P0) (SFR-80H) Port 0 SFR is a bit addressable Special Function Register that acts as the bit latch for each Port 0 pins.

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
P0.7	P0.6	P0.5	P0.4	P0.3	P0.2	P0.1	P0.0

During external memory operations P0 SFR gets 1s written into it. The reset value of Port 0 SFR is FFH (All latch bits set to 1)

5.3.5.2 Port 1 PORT 1 is a bi-directional port which is used as a general purpose I/O port. The Port 1 pin organisation is given in Fig. 5.11.

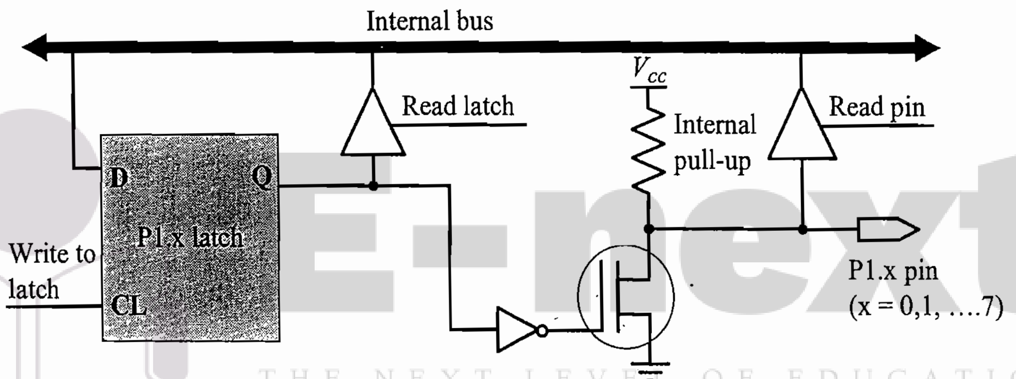


Fig. 5.11 Port 1 pin organisation

As seen in the pin organisation, Port 1 pin contains an internal pull-up resistor. In order to make the Port 1 pins as input line, the corresponding SFR latch bit for Port 1 should be kept as 1. Writing a 1 into any of the P1 SFR bit latch turns off the output driver FET and produces logic high at the corresponding port pin. The internal pull up for Port 1 is fixed and weak. When Port 1 pins are configured as inputs (by writing a 1 to the corresponding Port 1 SFR bit latch) the pins are pulled high and they can source current when an externally connected device pulls the port pin to low, signaling a logic 0 at the corresponding input line and places logic 0 to the internal bus in response to a *Read Pin* command. If the externally connected device forces logic high, the *Read Pin* control signal generated by a *Read Pin* related command (e.g. *MOV A,P1*, *MOV C,P1.0,* etc.) places logic high into the internal bus.

Since Port 1 holds fixed internal pull ups and are capable of sourcing current, it is known as *Quasi Bi-directional*.

Port 1 SFR (P1) (SFR- 90H) It is also a bit addressable Special Function Register that acts as the bit latch for each pin of Port 1. The bit details of Port1 SFR is given below.

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
P1.7	P1.6	P1.5	P1.4	P1.3	P1.2	P1.1	P1.0

The reset value of Port 1 SFR is FFH (All bit latches set to 1).

5.3.5.3 Port 2 Port 2 is designed to operate in two different modes. It acts as general purpose I/O port in normal operational mode and acts as higher order address bus in external data memory/program memory operations. Figure 5.12 illustrates the Port 2 pin organisation.

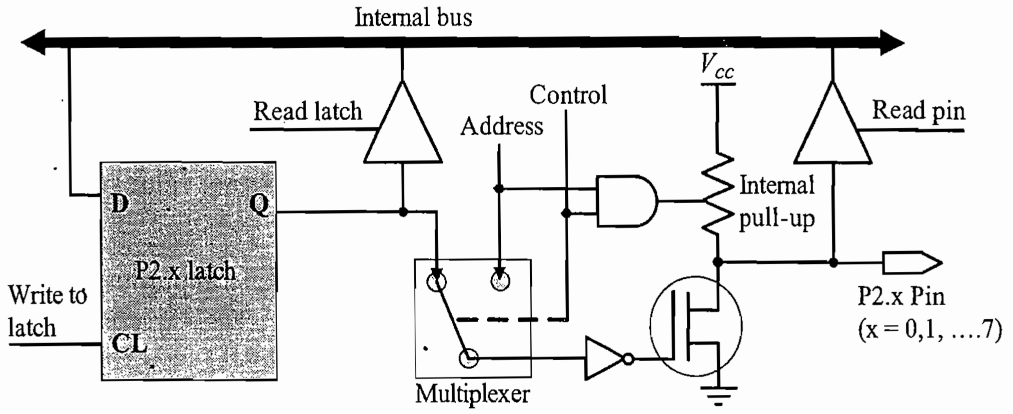


Fig. 5.12 Port 2 pin organisation

Port 2 emits the higher order byte of external memory address if the address is 16 bits wide. As seen in the figure, during 16-bit wide external memory operations the base drive for the O/p driver FET is internally switched to the address line. If the address line is emitting a 1, the O/p driver FET is turned off and the logic 1 is reflected on the O/p pin. If the address line is emitting a 0, the O/p driver FET is turned on and the logic 0 is reflected at the corresponding pin.

The content of Port 2 SFR remains unchanged during external memory access and it holds the previous content as such. It is to be noted that if Port 2 is in external memory operation it cannot be used as general purpose I/O line. When not used for external memory access, Port 2 can be used as general purpose I/O port. During normal operation mode, the internal multiplexer switches the base line (GATE) of O/p FET to the D latch O/p of corresponding SFR bit latch. In normal operation mode when a 1 is written into any of the P2 bit latch, the O/p driver FET is turned off and as in the case of Port 1 this line acts as an I/p line. P2 is a *Quasi bi-directional* port.

Port 2 SFR (P2) (SFR- A0H) It is a bit addressable Special Function Register that acts as the bit latch for each pins of Port 2. The reset value of Port 2 SFR is FFH (All bit latches set to 1).

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
P2.7	P2.6	P2.5	P2.4	P2.3	P2.2	P2.1	P2.0

5.3.5.4 Port 3 Port 3 is a general purpose I/O port which is also configurable for implementing alternative functions. Port 3 Pin configuration is shown in Fig. 5.13.

Port 3 is identical to Port 1 in operation. All the settings that need to be done for configuring Port 1 as I/O port is applicable to Port 3 also. The only difference is that the SFR latch for Port 3 is P3. Port 3 supports alternate I/O functions. The alternate I/O functions supported by Port 3 and the pins used for these alternate functions are tabulated below.

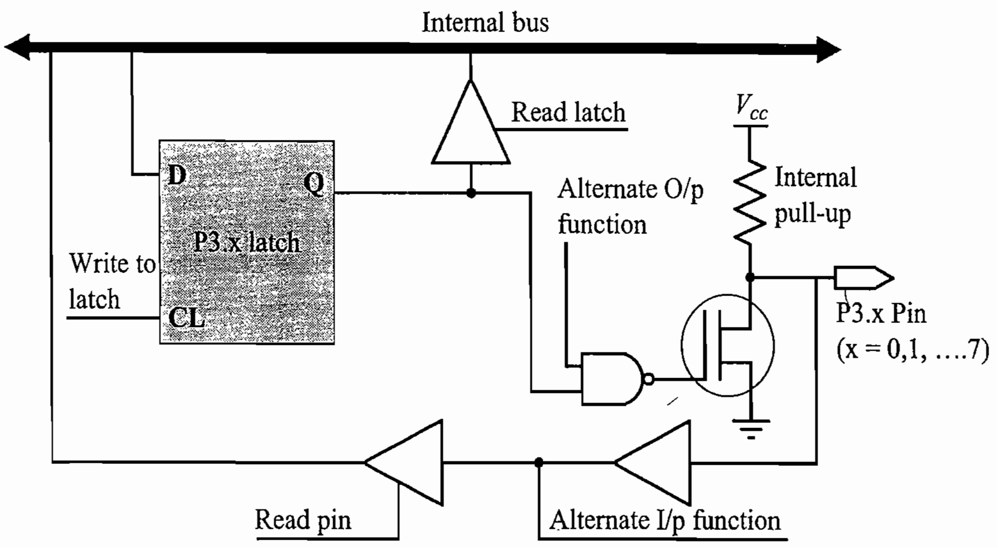


Fig. 5.13 **Port 3 pin organisation**

Port Pin	Alternate I/O Function
P3.0	RXD (Receive Pin – Serial Input Port Pin) – Input line
P3.1	TXD (Transmit Pin – Serial O/p Pin) – Output line
P3.2	INT0\ (External Interrupt 0 line) – Input line
P3.3	INT1\ (External Interrupt 1 line) – Input line
P3.4	T0 (Counter 0 External Input line) – Input line
P3.5	T1 (Counter 1 External Input line) – Input line
P3.6	WR\ (Write signal for External data memory access) – O/p line
P3.7	RD\ (Read signal for External data memory access) – O/p line

It is obvious from the table that all 8 pins of Port 3 are having some alternate I/O function associated with them. From the Port 3 pin configuration it is clear that the alternate I/O functions will come into action only if the corresponding SFR bit latch is set to logic 1. Otherwise the port pin remains at logic 0.

Port 3 SFR (P3) (SFR-B0H) It is a bit addressable Special Function Register that acts as the bit latch for each pin of Port 3. Reset value of Port 3 SFR is FFH (All bit latches set to 1).

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
P3.7	P3.6	P3.5	P3.4	P3.3	P3.2	P3.1	P3.0

5.3.5.5 'Read Port Latch' and 'Read Port Pin' Operations As we discussed earlier, the 'Port Read' operations fall into two categories namely, *Read Latch* and *Read Pin*. The *Read Latch* operation reads the content of the corresponding port latch. The port architecture for all 4 ports contains necessary circuit for reading the *Port Latch* for all port pins. The read *Port Latch* operation is triggered by the control signal *Read Latch*. The *Read Latch* control signal is generated internally on executing an instruction implementing the *Read Latch* operation. The *Read-Modify-Write* instructions which reads the port, modifies it and re-writes it to the port, operates on the port latch instead of port pins. The following *Read-Modify-Write* instructions operate on port latch when the destination operand is a Port or a Port bit.

ANL Px, <source> (x= 0,1,2,3. e.g. *ANL P0, A*)
ORL Px, <source> (x= 0,1,2,3. e.g. *ORL P1, A*)
XRL Px, <source> (x= 0,1,2,3. e.g. *XRL P2, A*)
JBC Px.y, LABEL (x= 0,1,2,3. y = 0 to 7 e.g. *JBC P3.0, REPEAT*)
CPL Px.y (x= 0,1,2,3. y = 0 to 7 e.g. *CPL P0.2*)
INC Px (x= 0,1,2,3. e.g. *INC P0*)
DEC Px (x= 0,1,2,3. e.g. *DEC P1*)
DJNZ Px, LABEL Px (x= 0,1,2,3. e.g. *DJNZ P0, REPEAT*)
MOV Px.y, C (x= 0,1,2,3. y = 0 to 7 e.g. *MOV P3.7, C*)
CLR Px.y (x= 0,1,2,3. y = 0 to 7 e.g. *CLR P1.0*)
SETB Px.y (x= 0,1,2,3. y = 0 to 7 e.g. *SETB P3.6*)

The instructions MOV Px.y, C, CLR Px.y and SETB Px.y, read the Port *x* byte (8 bits of the P*x* latch) and modify bit *y* and write the new byte back to the P*x* latch.

The following assembly code snippet illustrates the *Read Latch* operation.

```
MOV P0, #0FH     ; Configure P0.0 to P0.3 pins as input pins
MOV A, #0FH      ; Load Accumulator with 0FH
ANL P0, A        ; Read P0 latch, logical AND with Accumulator-
                 ; content and load P0 latch with the result
```

Executing the instruction *MOV P0, #0FH* loads the Port 0 latch with 0FH (The latches for port pins P0.0 to P0.3 are set). Now Port pins P0.0 to P0.3 acts as input pins. Executing the instruction *MOV A, #0FH* loads the accumulator with 0FH. The *ANL P0, A* instruction reads the P0 latch and logical AND it with accumulator and rewrites the P0 latch with the ANDed result. The status of port pins configured as input port has no effect on the instruction ANL P0, A. Suppose P0.0 pin (Not P0.0 latch bit) is at logic 0 and pins P0.1 to P0.3 are at logic 1 at the time of executing the instruction *ANL P0, A*, still Port 0 latch is loaded with 0FH and not 0EH.

The *Read Pin* operation reads the status of a port pin when the corresponding port pin is configured as input pin (When the corresponding port latch bit is loaded with logic 1). The port architecture for all 4 ports contains necessary circuit for reading the *Port Pin* for all ports. The read *Port Pin* operation is triggered by the control signal *Read Pin*. The *Read Pin* control signal is generated internally on executing an instruction implementing the *Read Pin* operation. *MOV A, Px, MOV C, Px.y* are examples for *Read Pin* instructions. The following code snippet illustrates the '*Read Pin*' operation.

```
MOV P0, #0FH     ; Configure P0.0 to P0.3 pins as input pins
MOV A, P0        ; Load Accumulator with P0 Port pin status
```

Executing the instruction *MOV P0, #0FH* loads the Port 0 latch with 0FH (The latches for port pins P0.0 to P0.3 are set). Now Port pins P0.0 to P0.3 act as input pins. Executing the instruction *MOV A, P0* loads accumulator with the Pin status of pins P0.0 P0.3. Suppose P0.0 pin is at logic 0 and pins P0.1 to P0.3 are at logic 1 at the time of executing the instruction *MOV A, P0*, the accumulator is loaded with 0EH.

5.3.5.6 Source and Sink Currents for 8051 Ports

Source Current The term *source current* refers to how much current the *8051* port pin can supply to drive an externally connected device. The device can be an LED, a buzzer or a TTL logic device. For TTL family of *8051* devices the source current is defined in terms of TTL logic. TTL logic has two logic levels namely logic 1 (High) and logic 0 (Low). The typical voltage levels for logic *Low* and *High* is given in the following table.

Logic Level	Input signal level		Output signal level		V_{cc}
	Min	Max	Min	Max	
Low	0V	0.8V	0V	0.5V	5V
High	2V	5V	2.7V	5V	5V

The logic levels are defined for a TTL gate acting as input and output. For logic 0 the input voltage level is defined as any voltage below 0.8V and the current is 1.6mA sinking current to ground through a TTL input. According to the *8051* design reference, the maximum current that a port pin (For an LS TTL logic based 8051 devices) can source is 60 µA.

Sink Current It refers to the maximum current that the *8051* port pin can absorb through a device which is connected to an external supply. The device can be an LED, a buzzer or a TTL logic device (For TTL logic based 8051 devices). Pins of Ports P1, P2 and P3 can sink a maximum current of 1.6 mA. Port 0 pins can sink currents up to 3.2 mA. Under steady state the maximum sink current is limited by the criteria: Maximum Sink Current per port pin = 10 mA, Maximum Sink current per 8-bit port for port 0 = 26 mA, Maximum Sink Current per 8-bit port for port 1, 2, & 3 = 15 mA, Maximum total Sink current for all output pin = 71 mA (As per the AT89C51 Datasheet). Figure 5.14 illustrates the circuits for source, sink and ideal port interfacing for *8051* port pins.

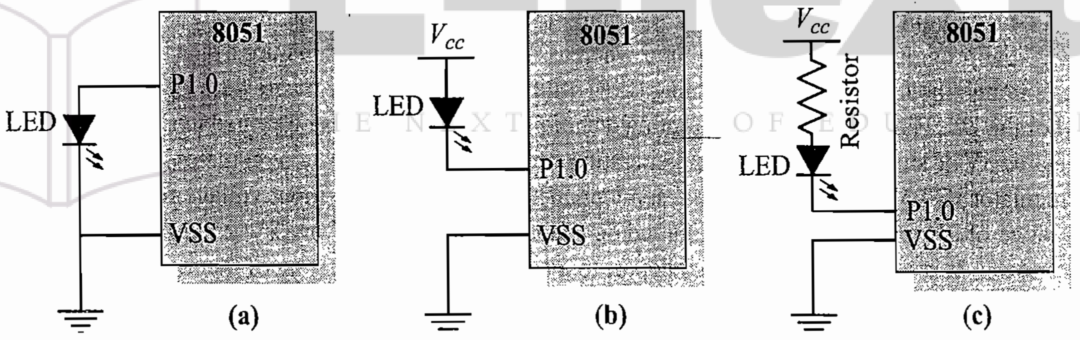


Fig. 5.14 **(a) Current Sourcing, (b) Current Sinking (c) Ideal Port pin interface for *8051***

Figure 5.14(a) illustrates the current sourcing for port pins. Since *8051* port pins are only capable of sourcing less than 1 mA current, the brightness of LED will be very poor. Figure 5.14(b) illustrates the current sinking for port pins. In this configuration, the forward voltage of LED while conducting is approximately 2V and the supply voltage 5V (V_{cc}) is distributed across the LED and the internal TTL circuitry. The extra 3V has to be dropped across the internal TTL circuitry and this will lead to high power dissipation, which in turn will result in the damage of the LED or the port pin. This type of design is not recommended in embedded design. Instead the current through the LED is limited by connecting the LED to the power supply through a resistor as shown in Fig. 5.14(c). In this configuration, the port pin should be at Logic 0 for the LED to conduct. For a 2.2V LED, the drop across Resistor is calculated as Supply voltage – (LED Forward Voltage + TTL Low Voltage) = 5 – (2.2 + 0.8) = 2.0V. The Resistance value is calculated as 2V / (Required LED Current). Refer LED data sheet for LED current. If the resistance value is not properly selected, it may lead to the flow of high current through the LED and may damage the LED.

Example 1

Design an 8051 microcontroller based system for displaying the binary numbers from 0 to 255 using 8 LEDs as per the specifications given below:

1. Use Atmel's AT89C51/52 or AT89S8252 (Flash microcontroller with In System Programming (ISP) support) for designing the system.
2. Use a 12MHz crystal resonator for generating the necessary clock signal for the controller.
3. Use on-chip program memory for storing the program instructions.
4. The 8 LEDs are connected to the port pins P2.0 to P2.7 of the microcontroller and are arranged in a single row with the LED connected to P2.0 at the rightmost position (LSB) and the LED connected to P2.7 at the leftmost position (MSB).
5. The LEDs are connected to the port pins through pull-up resistors of 470 ohms and will conduct only when the corresponding port pin is at logic 0.
6. Each LED represents the corresponding binary bit of a byte and it reflects the logic levels of the bit through turning ON and OFF the LED (The LED is turned on when the bit is at logic 1 and off when the LED is at logic 0).
7. The counting starts from 0 (All LEDs at turned OFF state) and increments by one. The counter is incremented at the rate of 5 seconds.
8. When the counter is at 255 (0FFH, all LEDs are in the turn ON state), the next increment resets the counter to 00H and the counting process is repeated.

The design of this system has two parts. The first part is the design of the microcontroller based hardware circuit. The hardware circuit part can be wired on a breadboard for simplifying the development. The controller for this can be chosen as either AT89C51/52 or AT89S8252. Both of these controllers are from the *8051* family and are pin compatible. Both of them contain built in program memory. The only difference is that for programming the AT89C51/52 device an EEPROM/FLASH programmer device is required whereas AT89S8252 doesn't require a special programmer. It can be programmed through the In System Programming (ISP) utility running on the firmware development PC and through the parallel port of the PC. The In System Programming technique for AT89S8252 is described in OLC. For the controller to work, a regulated 5V dc supply is required. For generating a regulated 5V dc supply, a regulator IC is used. For the current design the regulator IC LM7805 from National semiconductor is selected. The input voltage required for this regulator IC is in the range of 9V to 12V dc. A wall mounted dc adaptor with ratings 9V or 12V, 250mA can be used for supplying the input power. It is better to use a 9V dc adaptor to avoid the excessive heating of the regulator IC. Excessive heat production in the regulator IC leads to the requirement for heat sinks. The circuit details and the components required to implement the counter is shown in Fig. 5.15.

The circuit shows the minimal components and the interconnection among them to make the controller operational. As mentioned earlier, it requires a regulated 5V dc supply for powering the controller. The 12 MHZ crystal resonator in combination with the external 22 picofarad (pF) capacitors drives the on-chip oscillator unit and generates the required clock signal for the controller. The RC circuit connected to the RST pin of the controller provides Power-On reset for the controller. The capacitor and resistor values are selected in such a way that the reset pulse is active (high) for at least 2 machine cycle duration. The diode in the reset circuitry is used as freewheeling diode and it is not mandatory. The 0.1 Microfarad (0.1 MFD) capacitor connected to the power supply line filters the spurious (noise) signals from the power supply line. For proper driving, the LEDs should be connected to the respective port pins through pull-up resistors. The pull-up resistor values are determined by the forward voltage of LEDs and the current rating of the LEDs. The current design uses 470 ohms as the pull up resistor. If you are not sure about the forward voltage and current ratings of the LED, it is better to start with a high value (say 8.2K) for the resistor and replace it with successive low value resistors (4.2 K, 2.7K, 1 K, 870 E, 470 E etc.) till you feel that the brightness of the LED while it is conducting is reasonably good. The controller contains on-chip program memory and it can be used for storing the firmware. In order to use the on-chip program memory, the EA\ pin should be tied to V_{CC}. Pulling the EA\ pin to V_{CC} through a high value resistor ensures that the pin draws very minimal amount of current.

The second part is the design and development of program code (firmware) for implementing the binary counter and displaying the counter content using the LEDs interfaced to Port 2. The firmware requirement can be modelled in the form of a flow chart as given in Fig. 5.16.

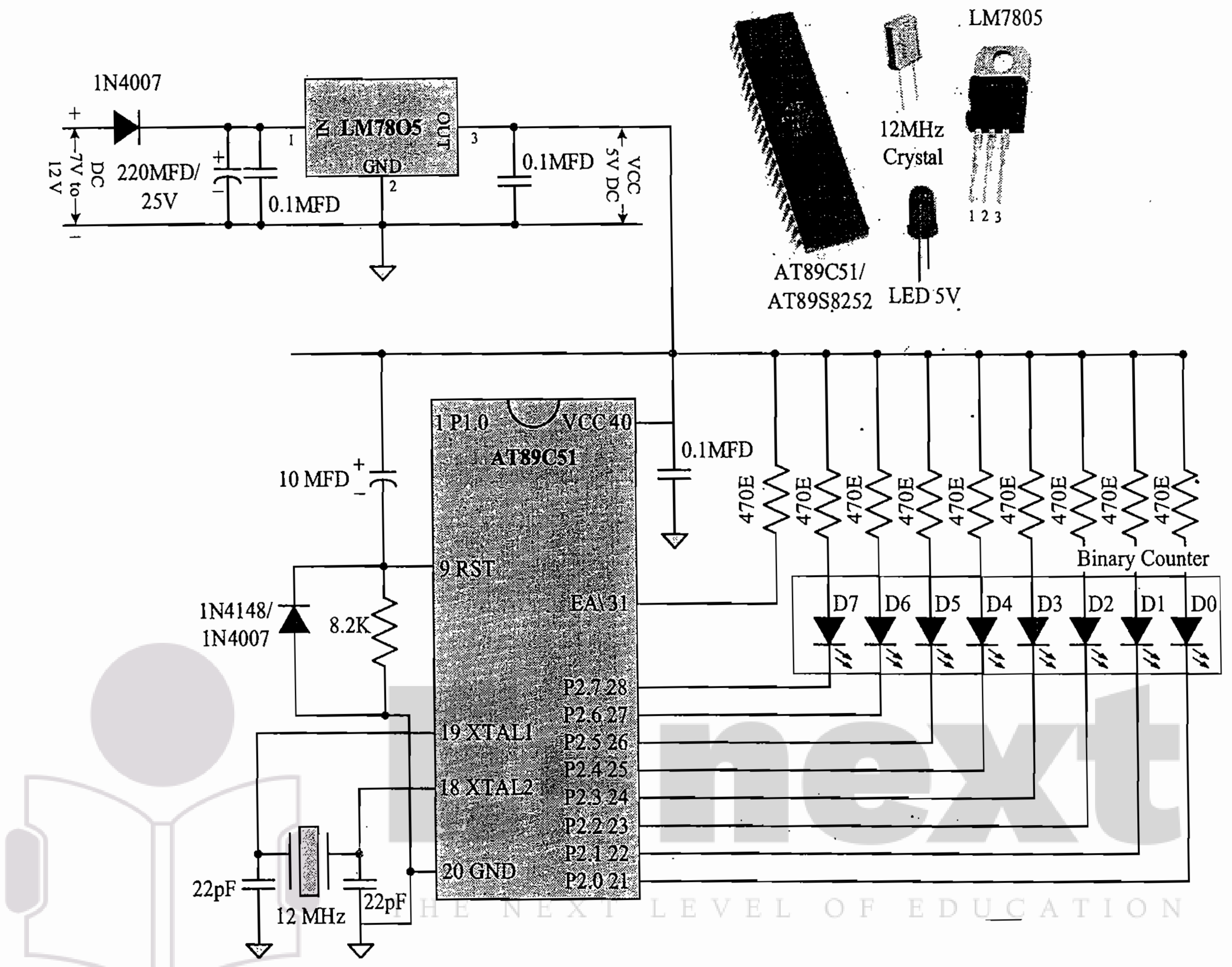
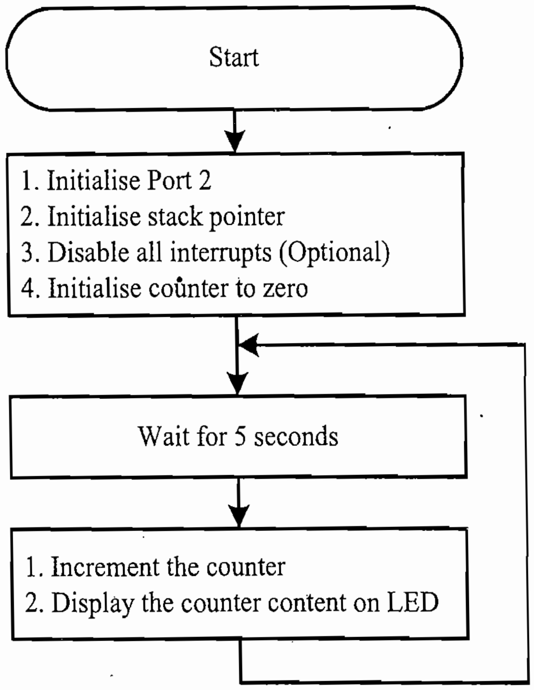


Fig. 5.15 **Binary number display circuit using *8051* and LEDs**



Start
1. Initialise Port 2
2. Initialise stack pointer
3. Disable all interrupts (Optional)
4. Initialise counter to zero
Wait for 5 seconds
1. Increment the counter
2. Display the counter content on LED

Fig. 5.16 **Flow chart for Binary Number Display using LEDs**

Once the firmware requirements are modelled using the flow chart, the next step is implementing it in either processor specific assembly code or high level language. For the time being let us implement the requirements in *8051* specific assembly language. The firmware for implementing the binary counter in *8051* assembly language is given below.

```
;####################################################################
;Binary_Counter.src
;Source code for implementing a binary counter and displaying the
;count through the LEDs connected to P2.0 to P2.7 port pins
;The LED is turned on when the port line is at logic 0
;The counter value should be complemented to display the count
;using the LEDs connected at Port 2. Written by Shibu K V
;Copyright (C) 2008
;####################################################################
ORG 0000H               ; Reset vector
        JMP 0050H       ; Jump to code mem location 0050H
ORG 0003H               ; External Interrupt 0 ISR location
        RETI            ; Simply return. Do nothing
ORG 000BH               ; Timer 0 Interrupt ISR location
        RETI            ; Simply return. Do nothing
ORG 0013H               ; External Interrupt 1 ISR location
        RETI            ; Simply return. Do nothing
ORG 001BH               ; Timer 1 Interrupt ISR location
        RETI            ; Simply return. Do nothing
ORG 0023H               ; Serial Interrupt ISR location
        RETI            ; Simply return. Do nothing
ORG 0050H               ; Start of Program Execution
        MOV P2, #0FFH   ; Turn off all LEDs
        CLR EA          ; Disable All interrupts
        MOV SP, #08H    ; Set stack at memory location 08H
        MOV R7,#00H     ; Set counter Register R7 to zero.
REPEAT: CALL DELAY      ; Wait for 5 seconds
        INC R7          ; Increment binary counter
        MOV A, R7       ;
        CPL A; The LED's are turned on when corresponding bit is 0
        MOV P2, A ; Display the count on LEDs connected at Port 2
        JMP REPEAT      ; Repeat counting
;####################################################################
;Routine for generating 5 seconds delay
;Delay generation is dependent on clock frequency
;This routine assumes a clock frequency of 12.00MHZ
;LOOP1 generates 248 x 2 Machine cycles (496microseconds) delay
;LOOP2 generates 200 x (496+2+1) Machine cycles (99800microseconds)
;delay. LOOP3 generate 50 x (99800+2+1) Machine cycles
;(4990150microseconds) delay. The routine generates a-
;precise delay of 4.99 seconds
;####################################################################
```

```
DELAY:  MOV R2, #50
LOOP1:  MOV R1, #200
LOOP2:  MOV R0, #248
LOOP3:  DJNZ R0, LOOP3
        DJNZ R1, LOOP2
        DJNZ R2, LOOP1
        RET
END ; END of Assembly Program
```

Once the assembly code is written and checked for syntax errors, it is converted into a controller specific machine code (hex file) using an assembler program. The conversion can be done using a freely/commercially available assembler program for *8051* or an IDE based tool (like Keil microvison 3). The final stage is embedding the hex code in the program memory of the controller. If the controller used is AT89C51, the program can be embedded using a FLASH programmer device. For controllers supporting In System Programming (ISP), like AT89S8252, the hex file can be directly loaded into the program memory of the controller using an ISP application running on the development PC.

Example 2

Design an *8051* microcontroller based control system for controlling a 5V, 2-phase 6-wire stepper motor. The system should satisfy the following:

1. Use Atmel's AT89C51/52 or AT89S8252 (Flash microcontroller with In System Programming (ISP) support) for designing the system.
2. Use a 12 MHz crystal resonator for generating the necessary clock signal for the controller.
3. Use on-chip program memory for storing the program instructions.
4. The wires of the stepper motor are marked corresponding to the coils (A, B, C & D) and Ground (2 wires)
5. Use the octal peripheral driver IC ULN2803 from National semiconductors for driving the stepper motor.
6. Step the motor in 'Full step' mode with a delay of 1 sec between the steps.
7. Connect the coil drives to Port 1 in the order Coil A to P1.0, Coil B to P1.1, Coil C to P1.2, and Coil D to P1.3

Refer to the description on stepper motors given in Chapter 2 to get an understanding of unipolar stepper motors and the coil energising sequence for 'Full step' mode.

Figure 5.17 illustrates the interfacing of stepper motor through the driver circuit connected to Port 1 of 8051. The flow chart given in Fig. 5.18 models the firmware requirements for interfacing the stepper motor.

From the pulse sequence for running the stepper motor in 'Full step' it is clear that the pulse sequence for next step is obtained by right shifting the current pulse sequence. The initial pulse sequence required is H, H, L, L at coils A, B, C & D respectively (Please refer to the stepper motor section in Chapter 2). In our case we have only 4 bits to shift and our controller is an 8bit controller. Performing a right shift operation of the accumulator moves the LS bit of accumulator to the MS bit position (Bit position 7 in 0–7 numbering). We want the LS bit to be available at 3rd bit position after each rotation. This can be achieved by some bit manipulation operation. We can also achieve it by loading the MS nibble of accumulator with the same initial sequence HHLL. In this example we are not using Port P1 for any other operation. Hence the values of port pins P1.4 to P1.7 are irrelevant in our case. But in real life scenario it may not be the case always. The firmware implementation for this is given below:

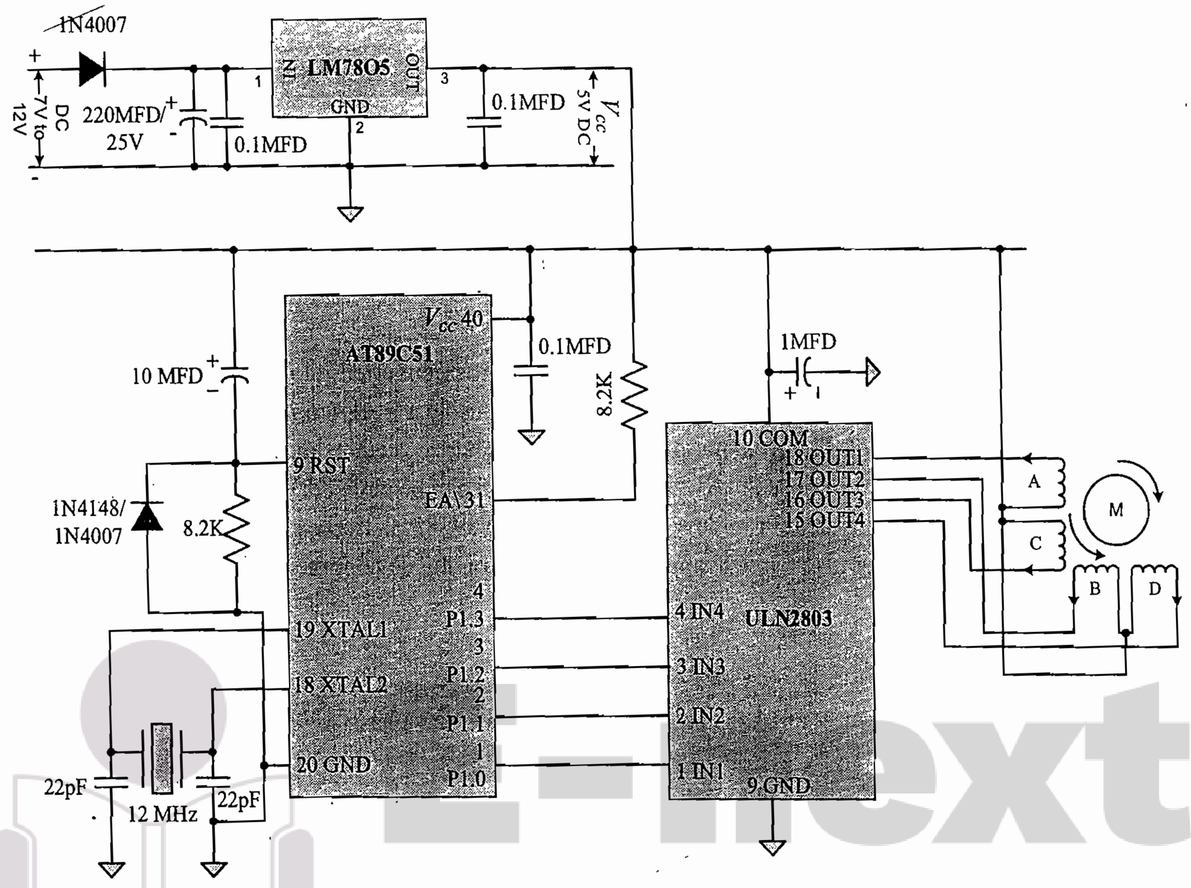


Fig. 5.17 Stepper Motor Interfacing circuit for 8051

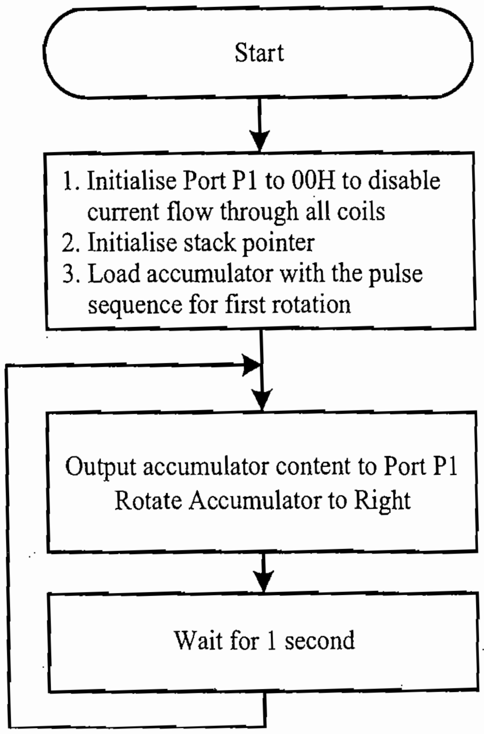

Start
1. Initialise Port P1 to 00H to disable current flow through all coils
2. Initialise stack pointer
3. Load accumulator with the pulse sequence for first rotation
Output accumulator content to Port P1
Rotate Accumulator to Right
Wait for 1 second

Fig. 5.18 Flow chart for Implementing Stepper Motor Interfacing

```
;####################################################################
;Stepper_motor.src. Firmware for Interfacing stepper motor
;The stator coils A, B, C and Dare controlled through Port pins P1.0,
;P1.1, P1.2 and P1.3 respectively.
;Accumulator is used for generating the pulse sequence for 'Fullstep'
;The initial pulse sequence is represented by 0CH
;Written & Compiled for A51 Assembler. Written by Shibu K V
;Copyright (C) 2008
;####################################################################
ORG 0000H               ; Reset vector
        JMP 0100H       ; Jump to code mem location 0100H to start-
                        ; execution
ORG 0003H               ; External Interrupt 0 ISR location
        RETI            ; Simply return. Do nothing
ORG 000BH               ; Timer 0 Interrupt ISR location
        RETI            ; Simply return. Do nothing
ORG 0013H               ; External Interrupt 1 ISR location
        RETI            ; Simply return. Do nothing
ORG 001BH               ; Timer 1 Interrupt ISR location
        RETI            ; Simply return. Do nothing
ORG 0023H               ; Serial Interrupt ISR location
        RETI            ; Simply return. Do nothing
;####################################################################
; Start of main Program
ORG 0100H
        MOV P1, #00H    ; Turn off the drives to all stator coils
        MOV SP, #08H    ; Set stack at memory location 08H
        MOV A, #0CCH    ; Load the initial pulse sequence
REPEAT: MOV P1, A       ; Load Port P1 with pulse sequence
        RR A            ; Rotate Accumulator to right
        CALL DELAY      ; Wait for 1 second
        JMP REPEAT      ; Load Port P1 with new pulse sequence
;####################################################################
;Routine for generating 1 second delay
;Delay generation is dependent on clock frequency
;This routine assumes a clock frequency of 12.00MHz
;LOOP1 generates 248 x 2 Machine cycles (496microseconds) delay
;LOOP2 generates 200 x (496+2+1) Machine cycles (99800microseconds)
;delay. LOOP3 generate 10 x (99800+2+1) Machine cycles
;(998030microseconds) delay. ;The routine generates a-
; precise delay of 0.99 seconds
;####################################################################
DELAY:  MOV R2, #10
LOOP1:  MOV R1, #200
LOOP2:  MOV R0, #248
LOOP3:  DJNZ R0, LOOP3
        DJNZ R1, LOOP2
        DJNZ R2, LOOP1
        RET
END     ;END of Assembly Program
```

Example 3

Design an *8051* microcontroller based system for interfacing the Programmable Peripheral Interface (PPI) device 8255. The system should satisfy the following:

1. Use Atmel's AT89C51/52 or AT89S8252 (Flash microcontroller with In System Programming (ISP) support) for designing the system
2. Use a 12 MHz crystal resonator for generating the necessary clock signal for the controller. Use on-chip program memory for storing the program instructions
3. Use Intersil Corporation's (wwww.intersil.com) 82C55A PPI device
4. Allocate the address space 8000H to FFFFH to 8255. Initialise the Port A, Port B and Port C of *8255* as Output ports in Mode0

Here we are allocating the address space 8000H to FFFFH to 8255. Hence the 8255 is activated when the 15th bit of address line becomes 1. Here we have to use a single *NOT* gate to invert the A15 line before applying it to the Chip Select (CS\) line of *8255*. In this configuration *8255* requires only four address space namely 8000H for Port A, 8001H for Port B, 8002H for Port C and 8003H for the Control Register. Rest of the address space 8004H to FFFFH is left unused. Here we have the luxury of using the entire address range since we don't have any other devices to connect. In real life applications it may not be the case. We may have multiple devices sharing the entire address space 0000H to FFFFH and we need to select each device in their own address space. In such scenarios the address lines A2 to A15 needs to be decoded using a combination of logic gates (NAND, AND, NOT, OR, NOR) and decoders. Figure 5.19 illustrates the interfacing of *8255* with *8051*.

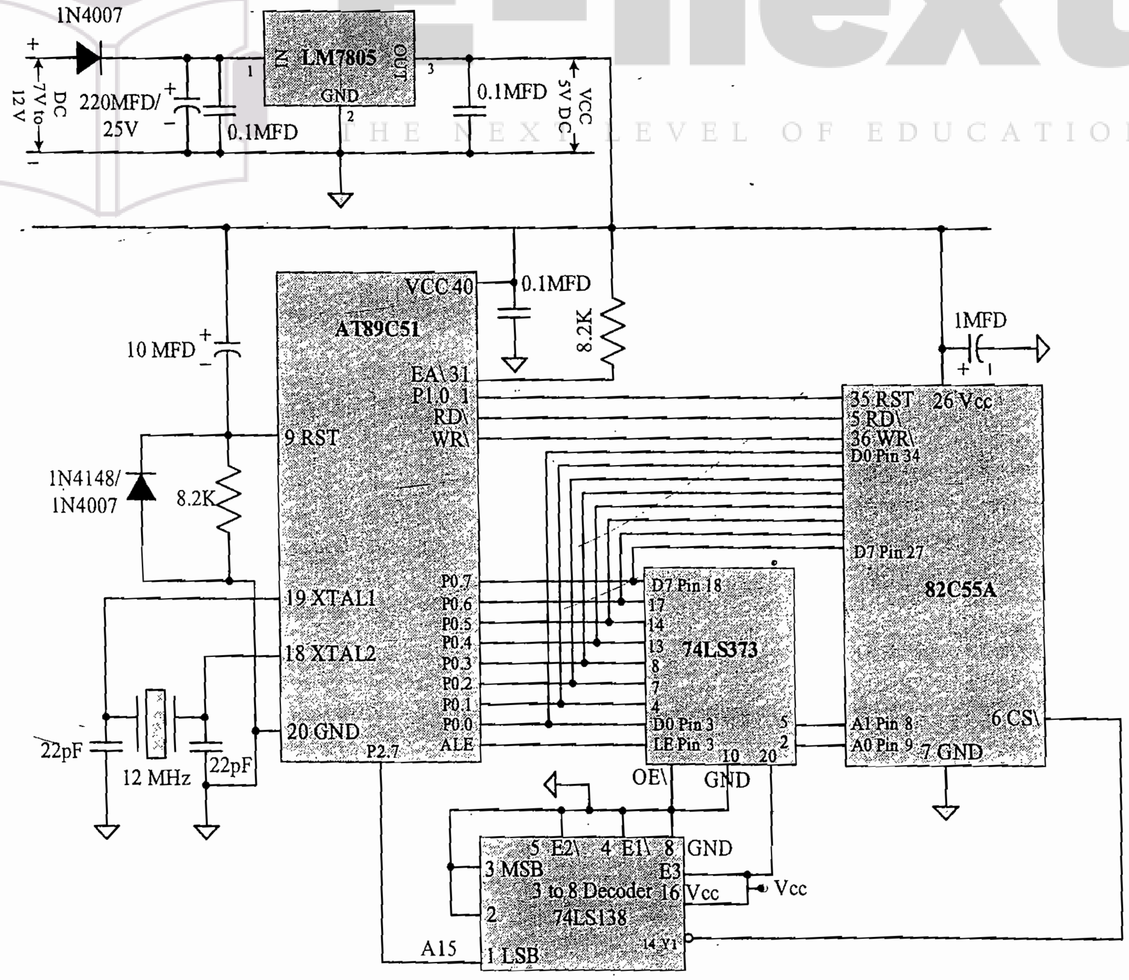


Fig. 5.19 Interfacing *8255* PPI with *8051*

The Octal latch 74LS373 latches the lower order address bus (A0-A7) which is multiplexed with the data bus (D0-D7). A 3 to 8 decoder chip, 74LS138, decodes the address bus to generate the Chip Select (CS\) signal for *8255*. Here we have only one address line (A15) to decode. The rest of the 2 input lines to the decoder (Pins 2 & 3) are grounded. Our intention is to assert the CS\ signal of *8255* when A15 line is 1. The I/p condition corresponds to this is 001. The decoded output for this is Output 1 (Y1). You can replace the decoder with a NOT Logic IC. The reset line of 8255 is controlled through port pin P1.0. The Reset of *8255* is active high and when *8051* is initialised, the port pin P1.0 automatically generates a reset high signal for *8255*.

The flow chart given in Fig. 5.20 models the firmware requirements for interfacing *8255* with *8051* controller.

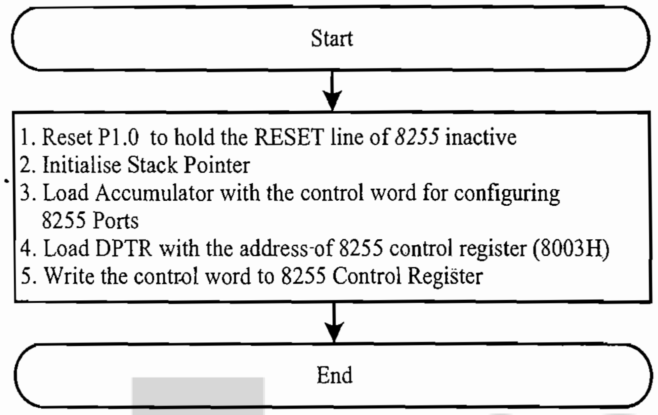

Start
1. Reset P1.0 to hold the RESET line of 8255 inactive
2. Initialise Stack Pointer
3. Load Accumulator with the control word for configuring 8255 Ports
4. Load DPTR with the address of 8255 control register (8003H)
5. Write the control word to 8255 Control Register
End

Fig. 5.20 Flow chart for Interfacing 8255 with *8051*

The control word for configuring all the *8255* ports as output ports in mode 0 is shown below. Please refer to the section on Programmable Peripheral Interface (PPI) given in Chapter 2 for more details on *8255*'s control register.

D7	D6	D5	D4	D3	D2	D1	D0
1	0	0	0	0	0	0	0

The firmware implementation for this is given below.

```
;######################################################################
;8255.src. Firmware for Interfacing 8255A PPI
;8255 is memory mapped at 8000H to FFFFH. The address assignment for
;various port and control register are: Port A : 8000H,
;Port B : 8001H, Port C : 8002H, Control Register : 8003H
;Reset of 8255 is controlled through P1.0 of 8051
;Written & Compiled for A51 Assembler. Written by Shibu K V
;Copyright (C) 2008
;######################################################################
ORG 0000H                ; Reset vector
        JMP MAIN         ; Jump to the address location pointed by
                         ; the Label 'MAIN' to start execution
ORG 0003H                ; External Interrupt 0 ISR location
        RETI             ; Simply return. Do nothing
ORG 000BH                ; Timer 0 Interrupt ISR location
        RETI             ; Simply return. Do nothing
ORG 0013H                ; External Interrupt 1 ISR location
        RETI             ; Simply return. Do nothing
ORG 001BH                ; Timer 1 Interrupt ISR location
        RETI             ; Simply return. Do nothing
```

```
ORG 0023H                 ; Serial Interrupt ISR location
        RETI              ; Simply return. Do nothing
;####################################################################
; Start of main Program
MAIN:   CLR P1.0          ; De-activate 8255 Reset line
        MOV SP, #08H      ; Set stack at memory location 08H
        MOV A, #80H       ; Load the initial Control Word
        MOV DPTR,#8003H; Load DPTR with the address of Control
                         ;Register
        MOVX @DPTR, A     ; Output the Control word to Control Register
        JMP $             ; Simply Loop here
END     ;END of Assembly Program
```

5.3.6 Interrupts

Before going to the details of *8051* interrupt system, let us have a look at interrupts in general and how interrupts work in microprocessor/controller systems.

5.3.6.1 What is Interrupt? As the name indicates, interrupt is something that produces some kind of interruption. In microprocessor and microcontroller systems, an interrupt is defined as a signal that initiates changes in normal program execution flow. The signal that generates changes in normal program execution flow may come from an external device connected to the microprocessor/controller, requesting the system that it needs immediate attention or the interrupt signal may come from some of the internal units of the processor/controller such as timer overflow indication signal. The first type of interrupt signals are referred as external interrupts.

5.3.6.2 Why Interrupts? From a programmer point of view interrupt is a boon. Interrupts are very useful in situations where you need to read or write some data from or to an externally connected device. Without interrupts, the normal procedure adopted is polling the device to get the status. You can write your program in two ways to poll the device. In the first method your program polls the device continuously till the device is ready to send data to the controller or ready to accept data from the controller. This technique achieves the desired objective effectively by sacrificing the processor time for that single task. Also there is a chance for the program hang up and the total system to crash in certain situations where the external device fails or stops functioning. Another approach for implementing the polling technique is to schedule the polling operation on a time slice basis and allocate the total time on a shared basis to rest of the tasks also. This leads to much more effective utilisation of the processor time. The biggest drawback of this approach is that there is a chance for missing some information coming from the device if the total tasks are high in number. Your device polling will get another chance to poll the device only after the other tasks are done at least once.

Here comes the role of interrupts. If the external device supports interrupt, you can connect the interrupt pin of the device to the interrupt line of the controller. Enable the corresponding interrupt in firmware. Write the code to handle the interrupt request service in a separate function and put the other tasks in the main program code. Here the main program is executed normally and when the external device asserts an interrupt, the main program is interrupted and the processor switches the program execution to the interrupt request service. On finishing the execution of the interrupt request service, the program flow is automatically diverted back to the main stream and the main program resumes its execution exactly from the point where it got interrupted.

5.3.6.3 Use of Interrupts In any interrupt based systems, interrupts are mainly used for accomplishing the following tasks:

1. I/O data transfer between peripheral devices and processor/controller
2. Timing applications
3. Handling emergency situations (e.g. switch off the system when the battery status falls below the critical limit in battery operated systems)
4. Context switching/Multitasking/Real-Time application programming
5. Event driven programming

5.3.7 The *8051* Interrupt System

I hope by now you got reasonably good information on interrupts and interrupt handling. Now let us move on to the interrupt system of *8051* microcontroller. The basic *8051* and its ROMless counterpart 8031AH supports five interrupt sources; namely two external interrupts, two timer interrupts and the serial interrupt. The serial interrupt is an ORed combination of the two serial interrupts; Receive Interrupt (RI) and Transmit Interrupt (TI).

5.3.7.1 Enabling Interrupts The interrupt system of *8051* can be enabled or disabled totally under software control. This is achieved by setting or clearing the global interrupt enable bit of the Special Function Register Interrupt Enable (IE). Also, each interrupt can be enabled or disabled individually by setting or clearing the corresponding interrupt enable bit in the SFR Interrupt Enable.

Interrupt Enable (IE) (SFR- A8H) The bit details of the Interrupt Enable Register is given below:

IE.7	IE.6	IE.5	IE.4	IE.3	IE.2	IE.1	IE.0
EA	RSD	RSD	ES	ET1	EX1	ET0	EX0

The table given below explains the meaning and use of each bit.

Bit	Name	Description
EA	Enable All	EA = 0 disable all interrupts. EA = 1 enable all interrupts, which are individually enabled by setting their corresponding enable bit in Interrupt Enable SFR.
RSD	Reserved	Unimplemented. Reserved for future use
ES	Enable Serial	ES = 1 enables Serial Interrupt. ES = 0 disables it
ET1	Enable Timer 1	ET1 = 1 enable Timer1 Interrupt. ET1 = 0 disables it
EX1	Enable External 1	EX1 = 1 enable External Interrupt 1. EX1 = 0 disables it
ET0	Enable Timer 0	ET0=1 enable Timer0 Interrupt. ET0=0 disables it
EX0	Enable External 0	EX0=1 enable External Interrupt 0. EX0 = 0 disables it

The following code snippet illustrates the enabling of Timer 0 interrupt and disabling of all other interrupts.

```
ORL IE, #10000010B    ; Set bits EA & ET0. Preserve other
                      ; bits as such
ANL IE, #11100010B    ; Reset bits ES, ET1, EX1 and EX0.
                      ; Preserve other bits as such
```

The instruction *ORL IE, #10000010B* sets the global interrupt enabler bit *EA(IE.7)* and the Timer 0 Interrupt enabler bit *ET0 (IE.1)*. The status of all other bits in the IE register is preserved. The instruction *ANL IE, #11100010B* preserves the status of the global interrupt enabler bit *EA(IE.7)*, the *RSD (IE.6 & IE.5)* bits and the Timer 0 Interrupt enabler bit *ET0 (IE.1)* and resets the Serial interrupt enabler bit *ES(IE.4)*, Timer 1 Interrupt enabler bit *ET1 (IE.3)*, External Interrupt 1 enabler bit *EX1 (IE.2)* and External Interrupt 0 enabler bit *EX0 (IE.0)*. This ensures that the reserved bits *RSD (IE.6 & IE.5)* are left untouched. The same can also be achieved by individually setting or clearing the corresponding bits of IE register using *SETB* and *CLR* instructions. This requires more number of instructions to achieve the result.

Note: Though the corresponding interrupt bits are 'set' in the IE register, the Interrupts will not be enabled and serviced if the global interrupt enabler, EA bit of the Interrupt Enable Register (IE) is 0.

5.3.7.2 *Setting Interrupt Priorities*

In a Real World application, interrupts can occur at any time (asynchronous behaviour) and different interrupts may occur simultaneously. This may confuse the processor on deciding which interrupt is to be serviced first. This arbitration problem is resolved by setting interrupt priorities. Interrupt priority is configured under software control. The Special Function Register Interrupt Priority (IP) Register is the one holding the interrupt priority settings for each interrupt.

Interrupt Priority Register (IP) (SFR-B8H) The bit details of the Interrupt Priority Register is explained in the table below.

IP.7	IP.6	IP.5	IP.4	IP.3	IP.2	IP.1	IP.0
RSD	RSD	RSD	PS	PT1	PX1	PT0	PX0

The table given below explains the meaning and use of each bit in the IP register.

Bit	Name	Description
RSD	Reserved	Unimplemented, Reserved for future use
PS	Serial interrupt priority	PS = 1 sets priority to Serial Interrupt
PT1	Timer 1 interrupt priority	PT1 = 1 sets priority to Timer1 Interrupt
PX1	External 1 interrupt priority	PX1 = 1 sets priority to External Interrupt 1
PT0	Timer 0 interrupt priority	PT0 = 1 sets priority to Timer 0 interrupt
PX0	External 0 interrupt priority	EX0 = 1 sets priority to External interrupt 0

The interrupt control system of *8051* contains latches to hold the status of each interrupt. The status of each interrupt flags are latched and updated during S5P2 of every machine cycle (Refer to machine cycles for information on S5P2). The latched samples are polled during the following machine cycle. If the flag for an enabled interrupt is found to be set in S5P2 of the previous cycle, the interrupt system transfers the program flow to the corresponding interrupt's service routine in the code memory (Provided none of the conditions described in section "Different conditions blocking an interrupt" blocks the vectoring of the interrupt). The Interrupt Service Routine address for each interrupt in the code memory is listed below.

Interrupt number	Interrupt source	ISR Location in code memory
0	External interrupt 0	0003H
1	Timer 0 interrupt	000BH

2	External interrupt 1	0013H
3	Timer 1 interrupt	001BH
4	Serial interrupt	0023H

It is to be noted that each Interrupt Service Routine (ISR) is allocated 8 bytes of code memory in the code memory space.

From the IP Register architecture it is obvious that each interrupt can be individually programmed to one of two priority levels by setting or clearing the corresponding priority bit in the Interrupt Priority Register. Some general info on *8051* interrupts is given below:

1. If two interrupt requests of different priority levels are received simultaneously, the request of higher priority interrupt is serviced.
2. If interrupt requests of the same priority level are received simultaneously, the order in which the interrupt flags are polled internally is served first. First polled first served. (Also known as internal polling sequence.)
3. A low-priority interrupt can always be interrupted by a high priority interrupt.
4. A low-priority interrupt in progress can never be interrupted by another low priority interrupt.
5. A high priority interrupt in progress cannot be interrupted by a low priority interrupt or an interrupt of equal priority.

5.3.7.3 Different conditions blocking an Interrupt It is not necessary that an interrupt should be serviced immediately on request. The following situations can block an interrupt request or delay the servicing of an interrupt request in *8051* architecture.

1. All interrupts are globally disabled by clearing the Enable All (EA) bit of Interrupt Enable register.
2. The interrupt is individually disabled by clearing its corresponding enable bit in the Interrupt Enable Register (IE).
3. An interrupt of higher priority or equal priority is already in progress.
4. The current polling machine cycle is not the final cycle in the execution of the instruction in progress (To ensure that the instruction in progress will be completed before vectoring to the interrupt service routine. In this state the LCALL generation to the ISR location is postponed till the completion of the current instruction).
5. The instruction in execution is RETI or a write to the IE/IP Register. (Ensures the interrupt related instructions will not make any conflicts).

In the first three conditions the interrupt is not serviced at all whereas conditions 4 and 5 services the interrupt request with a delay.

5.3.7.4 Returning from an Interrupt Service Routine An Interrupt Service Routine should end with an RETI instruction as the last executable instruction for the corresponding ISR. Executing the RETI instruction informs the interrupt system that the service routine for the corresponding interrupt is finished and it clears the corresponding priority-X (X=1 High priority) interrupt in progress flag by clearing the corresponding flip flop. This enables the system to accept any interrupts with low priority or equal priority of the interrupt which was just serviced. Remember an interrupt of higher priority can always interrupt a lower priority even if the corresponding priority's interrupt in progress flag is set. Executing the RETI instruction POPs (retrieves) the Program Counter (PC) content from stack and the program flow is brought back to the point where the interruption occurred.

In operation, RETI is similar to RET where RETI indicates return from an Interrupt Service Routine and RET indicates return from a normal routine. RETI clears the interrupt in progress flag as well as POPs (retrieves) the content of the Program Counter register to bring the program flow back to the point where it got interrupted. RET instruction only POPs the content of the Program Counter register and brings the program flow back to the point where the interruption occurred.

Will a non serviced Interrupt be serviced later? This is a genuine doubt raised by beginners in the *8051* based system design. The answer is '*No*'. Each interrupt polling sequence is a new one and it happens at S5P2 of each machine cycle. If an interrupt is not serviced in a machine cycle for the reason that it occurred simultaneously with another high priority interrupt, will be lost. There is no mechanism in place for holding the non serviced interrupts in queue and marking them as pending interrupts and servicing them later.

5.3.7.5 Priority Levels for 8051 Interrupts By default the *8051* architecture supports two levels of priority which is already explained in the previous sections. The first priority level is determined by the settings of the Interrupt Priority (IP) register. The second level is determined by the internal hardware polling sequence. The internal polling sequence based priority determination comes into action if two or more interrupt requests of equal priority occurs simultaneously. The internal polling based priority within the same level of priority is listed below in the descending order of priority.

Interrupt	Priority
External interrupt 0	HIGHEST
Timer 0 overflow interrupt	
External interrupt 1	
Timer 1 overflow interrupt	↓
Serial interrupt	LOWEST

5.3.7.6 What Happens when an Interrupt Occurs? On identifying the interrupt request number, the following actions are generated by the processor:

1. Complete the execution of instruction in progress.
2. The Program Counter (PC) content which is the address of the next instruction in code memory which will be executed in normal program flow is pushed automatically to the stack. Program Counter Low byte (PCL) is pushed first and Program Counter High (PCH) byte is pushed next.
3. Clear the corresponding interrupt flags if the interrupt is a timer or external interrupt (only for transition activated (edge triggered) configuration).
4. Set interrupt in progress flip flop.
5. Generate a long call (LCALL) to the corresponding Interrupt Service Routine address in the code memory (Known as vectoring of interrupt).

5.3.7.7 Interrupt Latency In the 8051 architecture, the interrupt flags are sampled and latched at S5P2 of each machine cycle. The latched samples are polled during S5P2 of the following machine cycle to find out their state. If the polling process identifies a priority interrupt flag's flip flop as set, an LCALL to its ISR location is generated (If and only if none of the conditions listed under the topic ***"Different conditions blocking an interrupt"*** blocks it). Interrupt latency is the time elapsed between the assertion of the interrupt and the start of the ISR for the same.

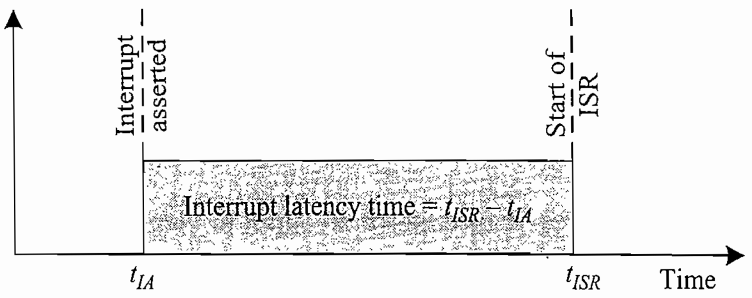


Fig. 5.21 **Interrupt Latency**

Interrupt latency is highly significant in real-time applications and is very crucial in time-critical applications. Interrupt latency can happen due to various reasons. For external interrupts there is no synchronisation with the system (asynchronous in behaviour) and so it can occur at any point of time. But the processor latches each interrupt flag only at S5P2 of each machine cycle. So there is no point even if the interrupt occurs at S1P1 of the machine cycle. It is latched only at S5P2 of the current machine cycle and the latched interrupts flags are polled at S5P2 of the following machine cycle and an LCALL is generated to the corresponding ISR, if no other conditions block the call. So this delay itself contributes a significant part in interrupt latency. Even if the ISR is entered some delay can be happened by the number of register contents to be stored (PUSH instructions) and other actions to be taken before executing the ISR task. The interrupt latency part which contributes the delay in servicing the ISR is the sum of the following time delays.

Time between the interrupt assertion to the start of state S5 of current machine cycle (polling cycle) + (Time for S5 & S6) + Remaining machine cycles for the current instruction in execution (The current execution should not be RETI or instructions accessing registers IE or IP, if so there will be an additional delay of the execution time for the next instruction) + LCALL generation time. The LCALL generation time is 12 T States (2 Machine cycles).

If the current machine cycle (the polling cycle) is the last cycle of the current instruction in execution and the current instruction in execution is other than RETI or instruction accessing IE or IP register, the interrupt vectoring happens at the shortest possible time and it is given as

Time between the interrupt asserted to start of state S5 of current machine cycle (polling cycle) + (S5+S6 T state + 12 clock)

= Time between the Interrupt Asserted to the start of state S5 + $(((1+1+12) \times 2)/f_{osc})$ seconds. (1 T state = 2 clock cycles and 1 clock cycle = $1/f_{osc}$)

The minimum time required to identify an interrupt by the system is one machine cycle, i.e. if an interrupt occurs at S5 of a machine cycle, it is latched and it is identified as an interrupt at state S5 of next machine cycle (Polling cycle). Hence the minimum time from interrupt assertion to S5 of the polling machine cycle is 1 machine cycle (12 clock periods). The maximum time required to identify an interrupt by the system is approximately 2 machine cycles. Assume the interrupt is asserted at state S6 of a machine cycle, it is latched at S5 of next machine cycle and the latched value is polled at S5 of next machine cycle. Hence the minimum time between the 'Interrupt Asserted' to state S5 of the current machine cycle (polling cycle) is 6 T States (1 machine cycle). That means State S6 of previous machine cycle to state S5 of current machine cycle. Hence the minimum acknowledgement time will be 20 T States (Since 1 machine cycle = 6 T states. The approximate response delay will be 3 machine cycles).

3 machine cycles is the minimum response delay for acknowledging an interrupt in a single interrupt system. There may have additional wait times which come as an addition to the minimum response

delay, depending on some other conditions. If you look back to the section "***Different conditions blocking an Interrupt***" you can see that if the current machine cycle when the interrupt asserted (The machine cycle at which the interrupt is latched) is the last machine cycle of the current instruction in progress, the interrupt vectoring will happen only after completing the next instruction. If the instruction in progress is not in its final machine cycle, the maximum additional waiting (waiting time excluding the LCALL generation time) time required to vector the Interrupt cannot be more than 3 machine cycles since the longest instructions MUL AB and DIV AB are 4 cycle instructions. If the instruction in progress is a RETI instruction or any access to IP or IE register then also the vectoring of the interrupt service routine will be delayed till the execution of next instruction. If the next instruction following the RETI or IP/IE register related instruction is MUL AB or DIV AB, the additional wait time will be 5 machine cycles (RETI and IP/IE related instructions are 2 machine cycle instructions).

In brief, the response time for interrupt in 8051 system is always more than 3 machine cycles and less than 9 machine cycles.

5.3.7.8 Configuring External Interrupts 8051 supports two external interrupts, namely, *External interrupt 0* and *External interrupt 1*. These are hardware interrupts. Two port pins of Port 3 serve the purpose of external interrupt input line. External interrupts are usually used for connecting peripheral devices. The external interrupt assertion can be either level triggered or edge triggered depending on the external device connected to the interrupt line. From the *8051* side it is configurable and the configuration is done at the SFR Timer/Counter Control Register (TCON). Bits TCON.0 and TCON.2 of TCON register configures the same for External Interrupt 0 and 1 respectively. TCON.0 is also known as Interrupt 0 type control bit (IT0). Setting this bit to 1 configures the external interrupt 0 as falling edge triggered. Clearing the bit configures the external interrupt 0 as low level triggered. Similarly, TCON.2 is known as Interrupt 1 type control bit (IT1). Setting this bit to 1 configures the external interrupt 1 as falling edge triggered. Clearing this bit configures the external interrupt 1 as low level triggered.

For external interrupts, the interrupt line should be asserted by the externally connected device to a minimum time period of the interval between two consecutive latching, i.e. S6P1 of previous machine cycle to S5P2 of current machine cycle (1 Machine cycle). Otherwise it may not be identified by the processor (Only interrupts which are found asserted during the latching time (S5P2) will be identified).

5.3.7.9 Single Stepping Program with the Help of External Interrupts Single stepping is the process of executing the program instruction by instruction. This can be achieved with the help of the external interrupt 0 or 1 and a small firmware modification. This works on the basic principle that an interrupt request will not be acknowledged if an interrupt of equal priority is in progress and if the instruction in progress when the interrupt is asserted is a RETI instruction, the vectoring will happen only after executing an instruction from the main program, which is interrupted by the interrupt. If the external interrupt is in the asserted state, the interrupt vectoring happens after executing one instruction from the main program. This execution switching between the main program and ISR can be achieved by a simple ISR firmware modification. Connecting a push button switch to the external interrupt line 0 through a resistor is the only hardware change needed for a *single step* operation (Fig. 5.22). Configure INT0 as level triggered in firmware. Activating the push button asserts the INT0 interrupt and the processor starts executing the ISR for interrupt 0. At the end, the ISR waits for a 1 to 0 transition at the INT0 line that asserts the external interrupt 0 again. The next instruction that is going to be executed on asserting the INT0 is *RETI* and according to the general interrupt vectoring principle the processor will go back to the point in the main code where it got interrupted and after completing one instruction it will again come back to the INT0 ISR. This process is repeated.

Single stepping can also be done with external interrupt 1. Make external interrupt 1 as level triggered and connect the hardware set up to pin P3.3 (external interrupt 1 line) and modify the ISR for external interrupt 1 as explained for external interrupt 0 ISR. It will work fine. Single stepping is a very useful method in debugging the application. It gives an insight into the various effects produced by the execution of an instruction, like registers and memory locations modified as a result of the execution of an instruction.

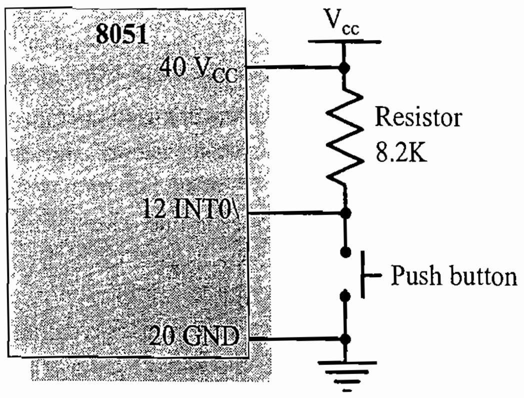


Fig. 5.22 Hardware setup for single stepping program with external interrupt

Example 1

Design an *8051* microcontroller based data acquisition system for interfacing the Analog to Digital Converter ADC, ADC0801 from National semiconductors. The system should satisfy the following:

1. Use Atmel's AT89C51/52 or AT89S8252 (Flash microcontroller with In System Programming (ISP) support) for designing the system. Use a 12MHZ crystal resonator for generating the necessary clock signal for the controller. Use on-chip program memory for storing the program instructions.
2. The data lines of the ADC is interfaced to Port 2 of the microcontroller. The control signals (RD\, WR\, CS\) to the ADC is supplied through Port 3 pins of microcontroller.
3. The Analog voltage range for conversion is from 0V to 5. The varying analog input voltage is generated from the 5V supply voltage (Vcc) using a variable resistor (Potentiometer). The ADC asserts its interrupt line to interrupt the microcontroller when the AD conversion is over and data is available at the ADC data port.
4. The microcontroller reads the data on receiving the interrupt and stores it in the memory. The successive data conversion is initiated after reading the converted data for a sample. Only the most recent 16 samples are stored in the microcontroller memory.

This example is a good starting point for a discussion on data converters and their use in embedded applications. The processing/controlling unit (The core of the embedded system) of every embedded system is made up of digital systems and they work only on digital data. The processor/controller is capable of dealing with binary (digital) data only. As mentioned in the beginning, embedded systems are in constant interaction with the real world through sensors and actuators. In most of the situations, the signals which are handled by embedded systems fall into the category 'analog'. Analog signals are continuous in nature. Most of the embedded systems used in control and instrumentation applications include the monitoring or control of at least one analog signal. The thermocouple-based temperature sensing system used in industrial control is a typical example for this. The thermocouple generates millivoltage (mV) in response to the changes in temperature. This voltage signals are filtered and signal conditioned and then applied to the monitoring system for monitoring and control purpose. Digital systems are not capable of handling analog signals as such for processing. The analog input signal from sensors needs to be digitized (quantized) before inputting to the digital systems. In a similar fashion, the output from digital systems are digital (discrete) in nature and they cannot be applied directly to analog systems which require continuous signals for their operation. The problem of handling the analog and digital data in embedded systems is handled by data converters. Data converters fall into two categories, namely; Analog to Digital Converters (ADC) and Digital to Analog Converters (DAC). Analog to Digital Converter (ADC) converts analog signals to corresponding digital representation, whereas Digital to Analog Converter (DAC) converts digital signals to the corresponding analog representation.

ADCs are used at the input stage of the processor/controller and DACs are used at the output stage of the processor/controller. ADCs and DACs are available in the form of Integrated Circuits (ICs) from different manufacturers. These chips are used for interfacing the processor/controller with sensors/actuators which produces/requires analog signals.

Analog to digital conversion can be accomplished through different conversion techniques. **Successive approximation** and **integrator** are the two commonly adopted analog to digital conversion techniques. In the successive approximation technique, the input analog signal is compared against a reference voltage. The reference voltage is adjusted till it matches the input analog voltage. The integrator technique changes the input voltage to time and compares the new parameters with known values. Discussing each technique in detail is out of the scope of this book (The current scope is limited to how data converters are used in embedded system designs) and readers are advised to refer books dedicated on digital systems/data acquisition systems. The successive approximation technique is faster in data conversion and at the same time less accurate, whereas, integrator technique is highly accurate but the conversion speed is slow compared to successive approximation technique. The successive approximation ADCs are used in embedded systems like control and instrumentation systems, which demands high conversion speed. The Integrator type ADCs are used in systems where the accuracy of conversion required is high. A typical example is a digital multimeter.

ADCs convert analog voltages in a given range to a binary data within the range supported by the ADC register. For example, for an 8bit ADC, the voltage range 0 to 5 V is represented with binary data 00H to FFH. The voltage range (5-0) is split across the 256 (0 to 255) steps. Hence the resolution of the ADC is represented as 1/256, meaning the binary data is incremented by one for a rise in 1/256 V in the input voltage. The resolution offered by the ADC depends on the input voltage range and the register width of the ADC. ADC can be used for the conversion of +ve as well as –ve voltage and range of I/p with offset from 0. It is the responsibility of the firmware developer to interpret all these conditions based on the system design. For example, if the input voltage is in the range 1 to 5V, the ADC represents 1 as 00H and 5 as FFH. The firmware developer should handle this appropriately.

The IC ADC0801ADC from National Semiconductor is an example for successive approximation ADC. The ADC0801 is designed in such a way that its tri-stated output latches can directly drive the processor/controller data bus/port. ADC0801 appears as a memory location or I/O port to the processor/controller and it doesn't require any additional bus interface logic. The logic inputs and outputs of ADC0801 meets both TTL and CMOS voltage level specifications and it can be used with processors/controllers from the CMOS/TTL logic family without using any interface conversion logic. ADC0801 supports input voltage range from 0 to 5V and two inputs, $V_{IN}(+)$ and $V_{IN}(-)$ for differential analog voltages. The input signal is applied to $V_{IN}(+)$ and $V_{IN}(-)$ is grounded for converting single ended positive voltages. For single ended negative voltage conversion, the input voltage is applied to $V_{IN}(-)$ and $V_{IN}(+)$ is grounded. ADC0801 provides an option for adjusting the span (range) of input voltage for conversion. The span adjustment is achieved by applying a voltage, which is half of the required span, at the $V_{REF}/2$ (Pin N0. 9) of ADC0801. For example, if the required span is 3V, apply a voltage 1.5V at pin $V_{REF}/2$. This converts the input voltage range 0V to 3V to digital data 00H to FFH. Keeping the $V_{REF}/2$ pin unconnected sets the span to 5V (The input varies from 0V to 5V). If the span is less than 5 and the range of input signal starts with an offset from 0V, the same can be achieved by applying corresponding voltages at pins $V_{REF}/2$ and $V_{IN}(-)$. As an example consider the requirement where, the span is 3V and range is from 0.5V to 3.5V, a voltage of 1.5V is applied to Pin $V_{REF}/2$ and 0.5V to pin $V_{IN}(-)$.The AD converter requires a clock signal for its operation. The minimal hardware connection required to build the AD converter system is shown in Fig. 5.23.

The AD converter contains an internal clock generator circuit. For putting the chip into operation, either an external clock or the built in clock generator can be used. ADC0801 provides a pin, CLK IN (Pin No. 4), for connecting external clock signal. The external clock signals can be generated using an oscillator circuit or the clock signal available from the CLK OUT pin of the microcontroller can be applied to the CLK IN. The internal clock generator circuit of the ADC can be used for generating the necessary clock signal to the system by connecting an external resistor and capacitor across the pins CLK IN and CLK R as shown in the schematic. The CLK IN pin is internally connected to the input pin of a Schmitt trigger and CLK R is connected to the o/p pin of the Schmitt triggers as shown in Fig. 5.24.

The frequency of the circuit is represented as $f_{CLK} = 1/1.1$ RC. The frequency range for ADC0801 is in the range 100 kHz to 800 kHz

ADC0801 requires three necessary control signals namely; RD\, WR\, CS\ for its operation. The CS\ signal is used for activating the ADC chip. For starting a conversion, the CS\ and WR\ signals should be asserted low. The internal Successive Approximation Register (SAR) of the chip is reset and the output data lines of the ADC enter high impedance state on asserting the WR\ signal. The data conversion begins when the WR\ signal makes a transition from asserted state to high. The ADC asserts the line INTR\ on completion of the conversion. This signal generates an interrupt at the processor. The converted digital data is available on the data bus of the ADC from the moment when the INTR\ line is asserted low. The INTR\ signal is reset when the RD\ signal is asserted low by the processor.

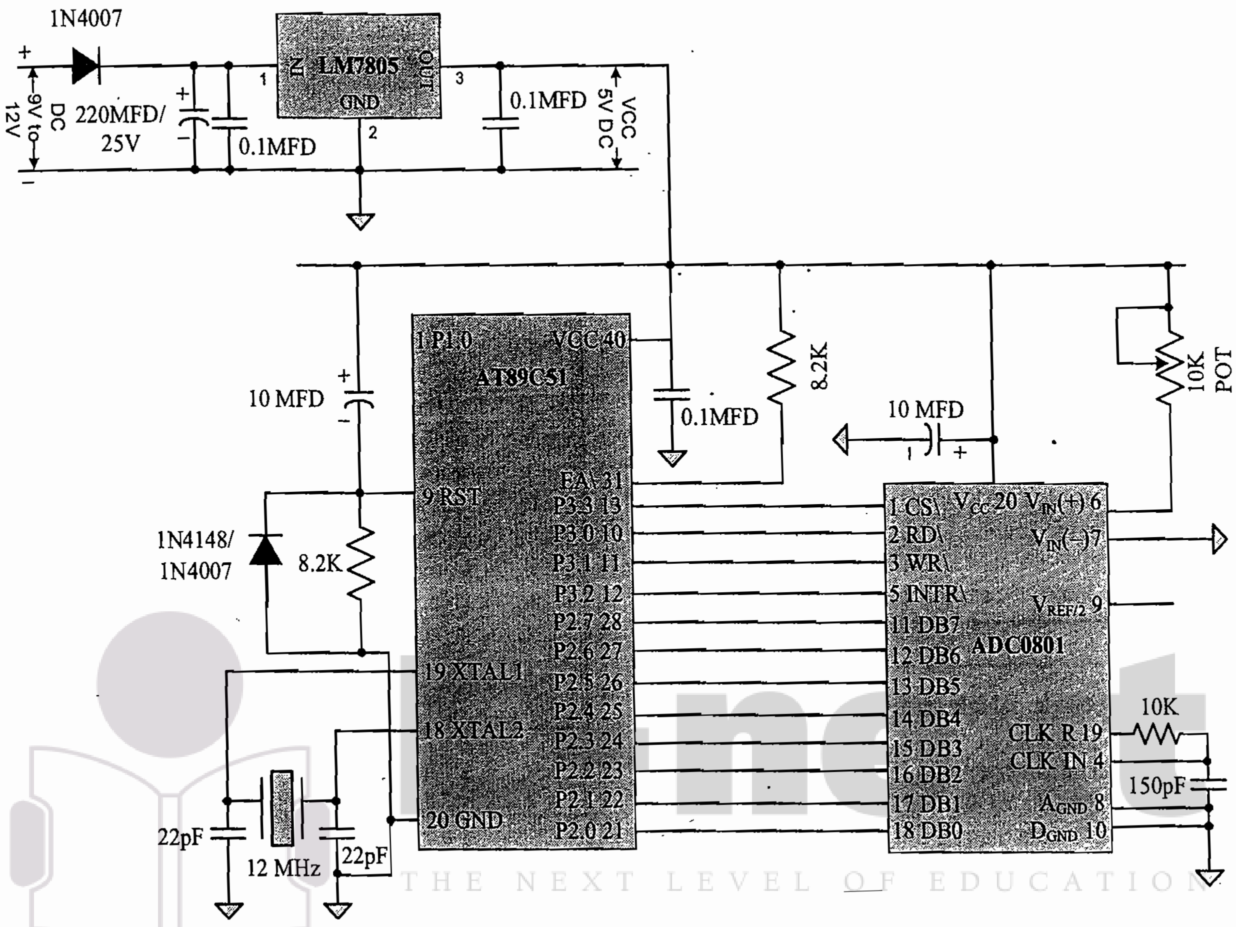


Fig. 5.23 **Interfacing *ADC0801* with *8051***

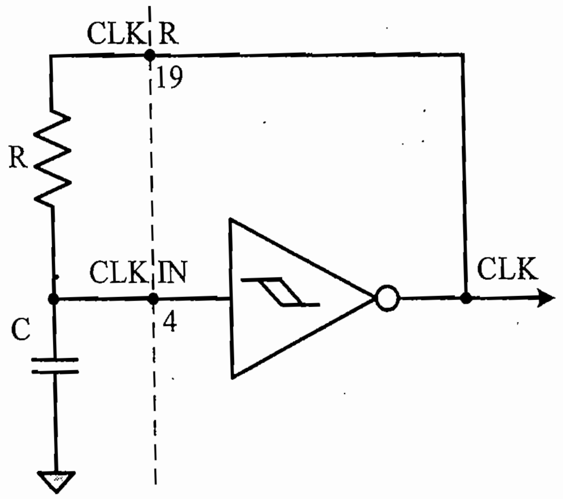
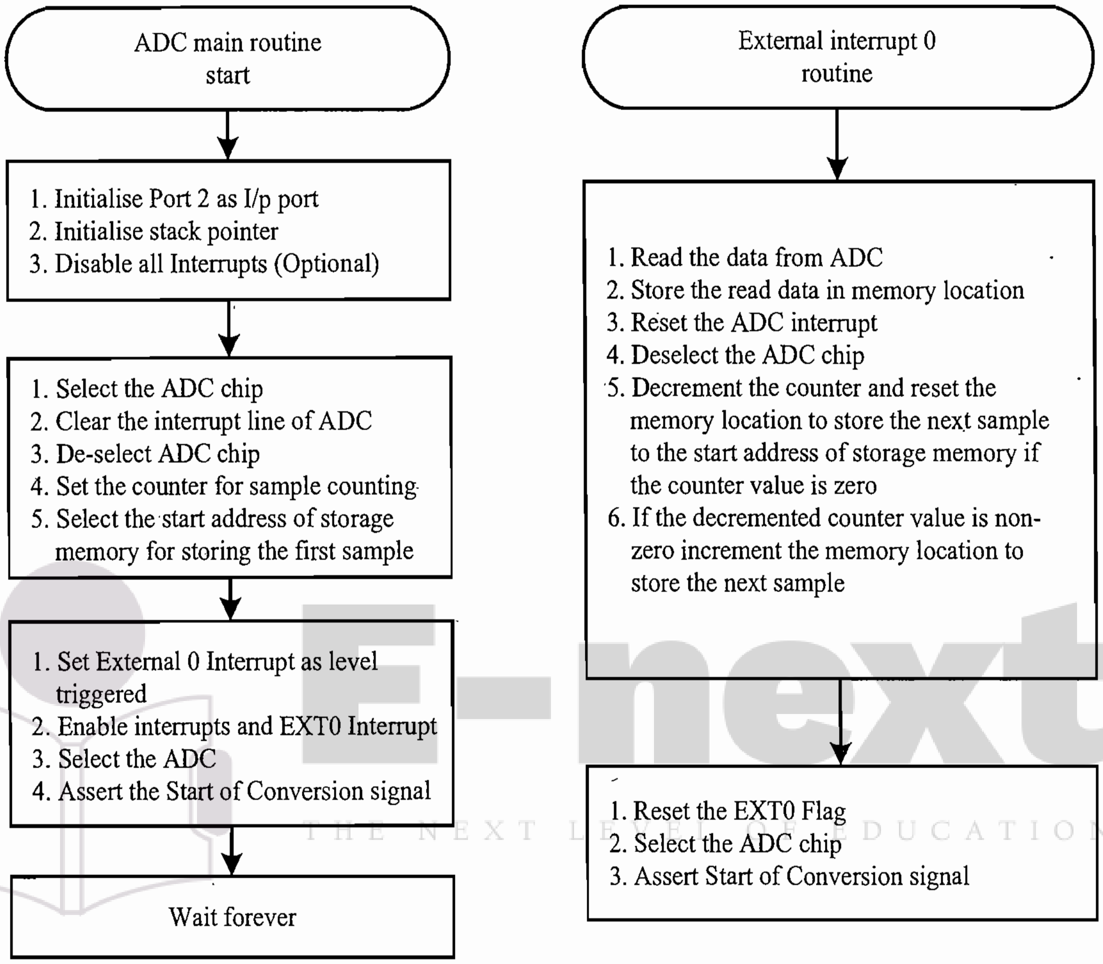


Fig. 5.24 **Schmitt Trigger Circuit**

In the current design, a potentiometer is used for varying the voltage from 0V to 5V. The data lines of the ADC are interfaced to Port 2 of the controller. The control signals to the ADC are supplied through the pins of Port 3. Port Pin P3.3 supplies the Chip Select (CS\) signal and Port Pin P3.0 supplies the RD\ Signal to the ADC. The WR\ signal to ADC is supplied through Port Pin P3.1. The INTR\ signal line of ADC is interfaced to the External Interrupt 0 (INT0\) line (Port Pin P3.2) of *8051*. The flow chart given in Fig. 5.25 illustrates the firmware design for the system.



ADC main routine start
1. Initialise Port 2 as I/p port
2. Initialise stack pointer
3. Disable all Interrupts (Optional)
1. Select the ADC chip
2. Clear the interrupt line of ADC
3. De-select ADC chip
4. Set the counter for sample counting
5. Select the start address of storage memory for storing the first sample
1. Set External 0 Interrupt as level triggered
2. Enable interrupts and EXT0 Interrupt
3. Select the ADC
4. Assert the Start of Conversion signal
Wait forever
External interrupt 0 routine
1. Read the data from ADC
2. Store the read data in memory location
3. Reset the ADC interrupt
4. Deselect the ADC chip
5. Decrement the counter and reset the memory location to store the next sample to the start address of storage memory if the counter value is zero
6. If the decremented counter value is non-zero increment the memory location to store the next sample
1. Reset the EXT0 Flag
2. Select the ADC chip
3. Assert Start of Conversion signal

Fig. 5.25 Flow chart for interfacing ADC0801

The firmware for interfacing the ADC in *8051* Assembly language is given below.

```
;######################################################################
;adc0801_interrupt.src. Firmware for interfacing ADC801 with 8051
;ADC - 8051 Physical Interface details
;ADC Data lines connected to Port P2 of 8051
;CS\ →P3.3; RD\ → P3.0; WR\ → P3.1; INTR\ → P3.2 (INT0\)
;Written by Shibu K V. Copyright (C) 2008
;######################################################################
ORG 0000H               ;Reset vector
        JMP 0050H       ; Jump to start of main program
ORG 0003H               ; External Interrupt 0 ISR location
        ; The ISR will not fit in 8 bytes size.
        ; Hence it is implemented as separate routine
        CALL EXTERNAL_INTR0 ; Function implementing Extnl 0 routine
        RETI            ; Return
```

```
ORG 000BH                  ; Timer 0 Interrupt ISR location
        RETI               ; Simply return. Do nothing
ORG 0013H                  ; External Interrupt 1 ISR location
        RETI               ; Simply return. Do nothing
ORG 001BH                  ; Timer 1 Interrupt ISR location
        RETI               ; Simply return. Do nothing
ORG 0023H                  ; Serial Interrupt ISR location
        RETI               ; Simply return. Do nothing
;####################################################################
ORG 0050H                  ; Start of main Program
        MOV P2, #0FFH      ; Configure Port2 as input Port
        MOV SP, #08H       ; Set stack at memory location 08H
        CLR P3.3           ; Select ADC
        CLR P3.0           ; Clear ADC interrupt line by asserting ADC
                           ; RD\
        SETB P3.3          ; De-select ADC
        MOV R0, #16        ; Set the counter for 16 samples
        MOV R1, #20H       ; Set start of sample storage memloc as 20H
        CLR IT0            ; Set External Interrupt 0 as low-level
                           ; triggered
        MOV IE, #10000001b ; Enable only External 0 interrupt
        CLR P3.3           ; Select ADC
        CLR P3.1           ; Trigger ADC Conversion; Reset ADC SAR
        NOP                ; Hold the WR\Signal low for 2 machine cycles
        NOP
        SETB P3.1          ; Toggle WR\ line from 0 to 1 to initiate-
                           ; conversion
        JMP $              ; Loop for ever
;####################################################################
; Routine for handling External 0 Interrupt
; External 0 Interrupt is asserted when ADC finishes data conversion
;####################################################################
EXTERNAL_INTR0:
        MOV @R1, P2
        CLR P3.0           ; Assert RD\ Signal to clear INTR\ ADC Signal
                           ; line
        NOP
        NOP
        SETB P3.0
        SETB P3.3          ; De-select ADC
        DJNZ R0, RETURN
        ; The 16 samples are taken
        ; overwrite the previous samples with new
        MOV R1, #1FH ; 20H is the mem loc holding the start of sample
        MOV R0, #16        ; Reload Sample counter with 16
RETURN: INC R1             ; Increment mem loc to hold next sample
        CLR IE0            ; Clear the interrupt 0 edge detect flag
        CLR P3.3           ; Select ADC
        CLR P3.1        ; Trigger ADC Conversion; Reset ADC SAR
        NOP             ; Hold the WR\Signal low for 2 machine cycles
        NOP
        SETB P3.1       ; Toggle WR\ line to initiate conversion
        RET
END     ; END of Assembly Program
```

5.3.8 Timer Units

Timers are very essential for generating precise time reference in any system. Timers can be implemented in either software or hardware. Hardware timers are dedicated hardware units and are highly precise in operation. The Standard 8051architecture supports two 16bit hardware timers that can be configured to operate in either timer mode or external event counting mode. The timers are named as *Timer 0* and *Timer 1*. If the timers are configured in timer function mode, the corresponding timer register is incremented once in each machine cycle. Since one machine cycle consists of six T-States (12 clock cycles), the timer increment rate is 1/12th of the oscillator frequency ($f_{osc}/12$).

If the timer unit is configured for counter mode, the timer register is incremented in response to a one to zero transition at the corresponding external port pin for counter mode. The external input pins are sampled during S5P2 of each machine cycle. If the sample shows a high in S5P2 of one machine cycle and a low in any of the next sampling time, the counter register is incremented by one. The count is updated only at S3P1 of the machine cycle following the machine cycle in which the transition is detected. It is obvious that it takes a minimum of two machine cycles to identify a 1 to 0 transition (2 machine cycle corresponds to 24 clock periods). So the maximum count rate for external events is $f_{osc}/24$, where f_{osc} is the oscillator frequency (Clock frequency). Timer 0 and Timer 1 can be configured as timer unit or counter unit by using timer/counter mode select bit of the special function register Timer/counter Mode Control (TMOD). Timer 0 and Timer 1 can be operated in four different modes. The mode selection is done by the timer mode select bits of TMOD register.

Timer/Counter Mode Control Register (TMOD) (SFR-89H)

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
GATE	C/T	M1	M0	GATE	C/T	M1	M0
Timer 1 Configuration				Timer 0 Configuration			

The following table explains the meaning and use of each bit in the TMOD register.

Bit	Name	Description
GATE	Gating control	If this bit is set to 1 in the corresponding timer configuration nibble of TMOD, the corresponding timer/counter will be enabled only when the INT0 /INT1 (INT0 corresponds Timer 0 /Counter 0 & INT1 corresponds Timer 1/Counter 1) line is high and the corresponding timer/counter's run bit TR is enabled in the TCON register. If GATE bit is 0, the timer/counter will run when the corresponding timer/counter's run bit TR is enabled in the TCON register.

C/T	Counter/Timer selector	Counter or timer mode select bit. If this bit is set to 1 in the corresponding timer configuration nibble of TMOD register, the corresponding timer will act as a counter. If the bit is cleared in the corresponding timer configuration nibble of TMOD register, the corresponding timer will function as timer unit.
M1 M0	Mode select bits	The Timer/Counter operation 'mode select' bits. It can be one among the 4 modes.

The values for bits M0 and M1 and the corresponding mode of operation for the timer/counter is explained in the table given below.

Mode Select Bits		Description
M1	**M0**	
0	0	Mode 0. 8bit timer with divided by 32 prescaler.
0	1	Mode 1. 16bit timer.
1	0	Mode 2. Autoreload mode. Configures timer register as 8bit timer/counter with auto reload. The lower byte of timer register only gets incremented and when a roll over from FFH to 00H occurs, the lower byte of register is re-loaded with the higher byte of the corresponding timer register. The higher byte always holds the reload value. Timer *x* can be viewed as two 8bit registers namely (TLx) & (THx) where *x* = timer number (0 or 1)
1	1	Mode 3. Timer 1 in this mode simply holds its count. It is similar to disabling the Timer 1 run control bit TR1. Timer 0 in this mode configures as two 8bit registers TL0 and TH0. TL0 is configured by the Timer 0 configuration nibble and TH0 is configured by the Timer 1 configuration nibble.

Timer/Counter Control Register (TCON) (SFR-88H) It is a bit addressable special function register that holds the timer/counter interrupt flags and run control bits.

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
TF1	TR1	TF0	TR 0	IE1	IT1	IE0	IT 0

The following table explains the meaning and use of each bit in the TCON register.

Bit	Name	Description
TF1	Timer/Counter 1 overflow flag	Set by hardware when timer/counter 1 overflows. The flag is automatically cleared when timer 1 interrupt is vectored.

TR1	Timer/Counter 1 Run control	TR1=1 Start timer/counter1
		TR1=0 Stops timer/counter 1

TF0	Timer/Counter 0 overflow flag	Set by hardware when timer/counter 0 overflows. The flag is automatically cleared when timer 0 interrupt is vectored.

TR0	Timer/Counter 0 Run control	TR0 = 1 Start Timer/Counter 0
		TR0 = 0 Stops Timer/Counter 0

IE1	External Interrupt 1 edge detect flag	Set by hardware when external interrupt 1 edge is detected. Cleared by hardware when interrupt is vectored

IT1	External Interrupt 1 type control	IT1 = 1 Configures the external interrupt 1 to edge triggered (Falling Edge). IT1 = 0 Configures the external interrupt 1 to level triggered (Low level)
IE0	External interrupt 0 edge detect flag	Set by hardware when external interrupt 0 edge is detected. Cleared by hardware when interrupt is vectored
IT0	External interrupt 0 type control	IT0 = 1 Configures the external interrupt 0 to edge triggered (Falling edge) IT0 = 0 Configures the external interrupt 0 to level triggered (Low level)

5.3.8.1 Timer/Counter in Mode 0 Timer/Counter-0 and Timer/Counter-1 in mode 0 acts as a 13bit timer/counter (Fig. 5.26). The 13bit register is formed by all 8 bits of TH0 and the lower 5 bits of TL0 for Timer/Counter-0 (All 8 bits of TH1 and the lower 5 bits of TL1 for Timer/Counter-1). The timer/counter mode selection is done by the bit C/T of the register TMOD. Timer/Counter-0 & Timer/Counter-1 has separate selection bits as described earlier. If Timer/Counter-0 is configured as timer and if the corresponding run control bit for Timer/Counter-0 (TR0 in Timer Control Register (TCON)) is set, registers TL0 & TH0 functions as a 13bit register and starts incrementing from its current value. The timer register is incremented by one on each machine cycle. When the 13bit Timer register rolls over from all 1s to all 0s (FF1FH to 0000H), the corresponding timer overflow flag TF0, present in TCON register is set. If the Timer 0 interrupt is in the enabled state and no other conditions block the Timer 0 interrupt, Timer 0 interrupt is generated and is vectored to its corresponding vector address 000BH. On vectoring the interrupt, the TF0 flag is automatically cleared. Operation of Timer 1 in mode 0 is same as that of Timer 0 except that for Timer 1, the 13bit timer register is formed by the registers TL1 and TH1 and the corresponding Timer run control bit is TR1. The Timer 1 overflow flag is TF1 and the corresponding interrupt vector location for Timer 1 is 001BH. It is advised to set the GATE control bit to 0 for timer operations. If the GATE control is set to 1, the timer register is incremented in accordance with each machine cycle only if the corresponding Interrupt line INTRx is high (INT0 for Timer 0 and INT1 for Timer 1).

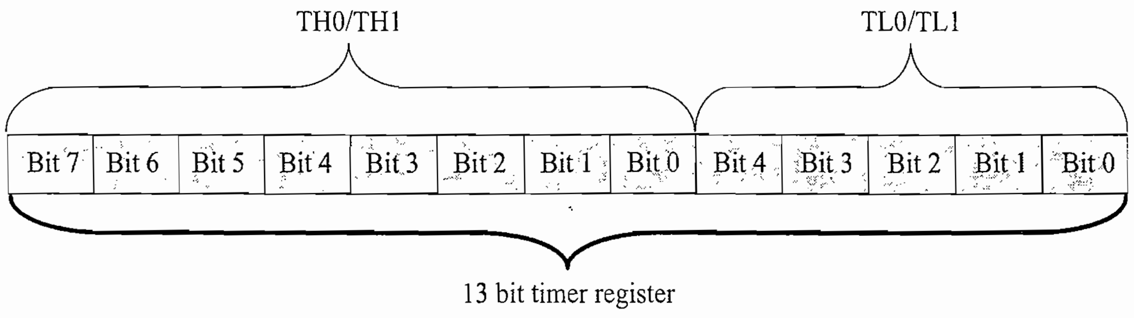


Fig. 5.26 **Timer (Counter) Register for Mode 0**

Figure 5.27 illustrates the operation of Timer/Counter in Mode 0

It is to be noted that for mode 0 operation, the lower 5 bits of TL0/TL1 (Bit 0 to Bit 4) and all 8 bits of TH0/TH1 forms the 13bit register. When the 5 bits of TL0 rolls over from all 1s (IFH) to all 0s (00H), TH0 is incremented by one. The upper 3 bits of TL0/TL1 is indeterminate (Don't care bits). TH0/TH1

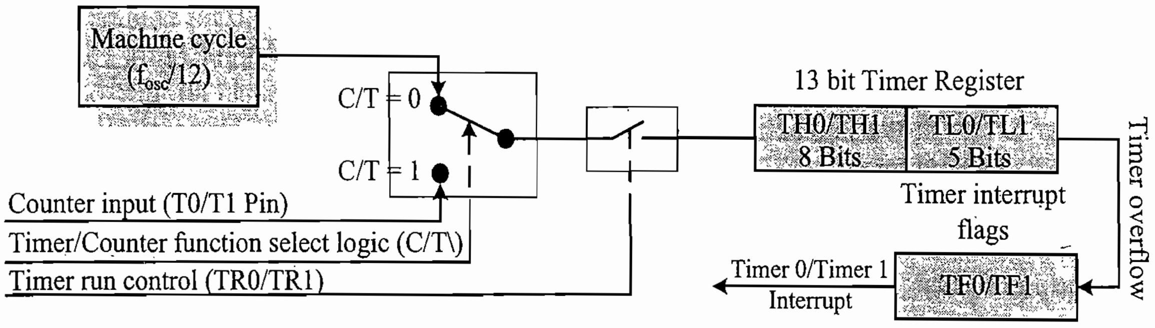

Machine cycle (f_{osc}/12)
C/T = 0
C/T = 1
Counter input (T0/T1 Pin)
Timer/Counter function select logic (C/T\)
Timer run control (TR0/TR1)
13 bit Timer Register
TH0/TH1 8 Bits
TL0/TL1 5 Bits
Timer interrupt flags
Timer 0/Timer 1 Interrupt
TF0/TF1
Timer overflow

Fig. 5.27 **Timer (Counter) 0/ Timer (Counter) 1 in Mode 0**

runs in its full 8bit mode and act as the 8bit counter whereas, only the 5 least significant bits of TL0/TL1 undergoes changes in accordance with machine cycle and so TL0/TL1 is said to be acting as a prescaler 32 for the 8bit counter TH0/TH1. The timer interrupt is generated when the 13bit timer register rolls over from all 1s to all 0s, i.e. when TH0 count is FFH and TL0 count rolls from 1FH to 00H. The count range for mode 0 is 0000H to 1FFFH.

5.3.8.2 Timer/Counter in Mode 1 Mode 1 operation of Timer (Counter) 0/ Timer (Counter) 1 is similar to that of mode 0 except the timer register size (Fig. 5.28). The timer register is 16bit wide in mode 1. The timer/counter mode selection is done by the bit C/T of the register TMOD. Timer (Counter) 0 and Timer (Counter) 1 has separate selection bits as described earlier. If Timer 0 is configured as a timer and if the corresponding run control bit for Timer 0 (TR0 in Timer Control Unit (TCON)) is set, registers TL0 and TH0 functions as 16bit register and starts incrementing from its current value. The timer register is incremented by one on each machine cycle. When the timer register rolls over from all 1s to all 0s (FFFFH to 0000H), the corresponding timer overflow flag TF0, present in TCON register is set. If Timer 0 interrupt is in the enabled state and no other conditions block the Timer 0 interrupt, Timer 0 interrupt is generated and is vectored to its corresponding vector address 000BH. On vectoring the interrupt, the TF0 flag is automatically cleared. Operation of Timer 1 in mode 1 is same as that of Timer 0 except that for Timer 1 the 16bit timer register is formed by the registers TL1 and TH1 and the corresponding Timer run control bit is TR1. The Timer 1 overflow flag is TF1 and the corresponding interrupt vector location for Timer 1 is 001BH.

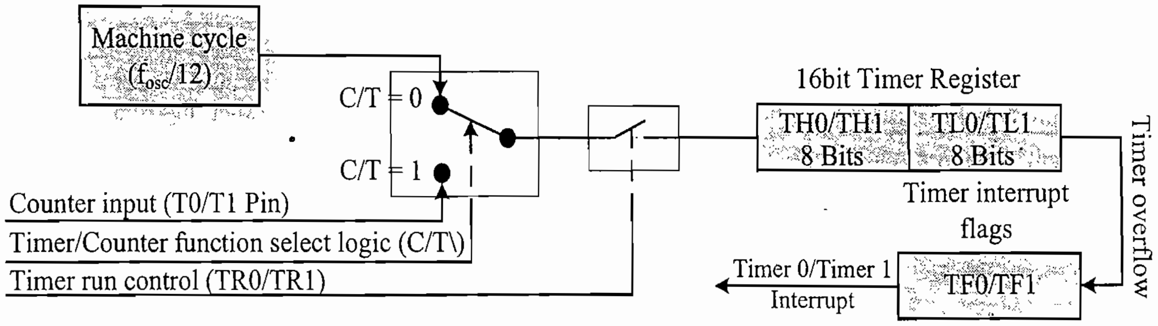

Machine cycle (f_{osc}/12)
C/T = 0
C/T = 1
Counter input (T0/T1 Pin)
Timer/Counter function select logic (C/T\)
Timer run control (TR0/TR1)
16bit Timer Register
TH0/TH1 8 Bits
TL0/TL1 8 Bits
Timer interrupt flags
Timer 0/Timer 1 Interrupt
TF0/TF1
Timer overflow

Fig. 5.28 **Timer (Counter) 0/ Timer (Counter) 1 in Mode 1**

It is advised to set the GATE control bit to 0 for timer operations. If the GATE control is set to 1, the timer register is incremented in accordance with each machine cycle only if the corresponding Interrupt line INTx is high (INT0 for Timer 0 and INT1 for Timer 1).

If the counter mode is set, and the corresponding counter run control bit (TR0 for counter 0 and TR1 for counter 1) is 1, the timer register ((TH0 & TL0) for Counter 0 and (TH1 & TL1) for Counter 1) is incremented by one on each traceable logic transitions at the counter input pins (T0 for Counter 0 and T1 for Counter 1). The above said actions are followed only if the gating control bit GATE is set to 0. If GATE is set to 1 the counter (timer) register is incremented by one for each traceable logic transitions at the counter input pins only when the corresponding Interrupt line INTx is high (INT0 for Counter 0 and INT1 for Counter 1). The overflow process and interrupt vectoring are similar to that for the Timer0/ Timer1 operation.

5.3.8.3 Timer/Counter in Mode 2 (Auto Reload Mode) Timer (Counter) 0/ Timer (Counter) 1 in mode 2 acts as 8bit timer/counter (TL0 for Timer (Counter) 0 and TL1 for Timer (Counter) 1) with auto reload on the timer/counter register overflow (TL0 or TL1) (Fig. 5.29). The timer/counter mode selection is done by the bit C/T of the register TMOD. Timer (Counter) 0 & Timer (Counter) 1 has separate selection bits as described earlier. If Timer 0 is configured as timer and if the corresponding run control bit for Timer 0 (TR0) in Timer Control Unit (TCON)) is set, register TL0 functions as 8bit register and starts incrementing from its current value. The timer register is incremented by one on each machine cycle. When the Timer register rolls over from all 1s to all 0s (FFH to 00H), the corresponding timer overflow flag TF0, present in TCON register is set and TL0 is reloaded with the value from TH0. TH0 remains unchanged. If Timer 0 interrupt is in the enabled state and no other conditions block the Timer 0 interrupt, it is vectored to the corresponding vector address 000BH. On vectoring the interrupt, the TF0 flag is automatically cleared. Operation of Timer 1 in mode 2 is same as that of Timer 0 except that for Timer 1 the 8bit timer register is formed by the register TL1 and the corresponding Timer run control bit is TR1. The Timer 1 overflow flag is TF1 and the corresponding interrupt vector location is 001BH. It is advised to set the GATE control bit to 0 for timer operations. If the GATE control is set to 1, the timer register is incremented in accordance with each machine cycle only if the corresponding Interrupt line INTx is high (INT0 for Timer 0 and INT1 for Timer 1).

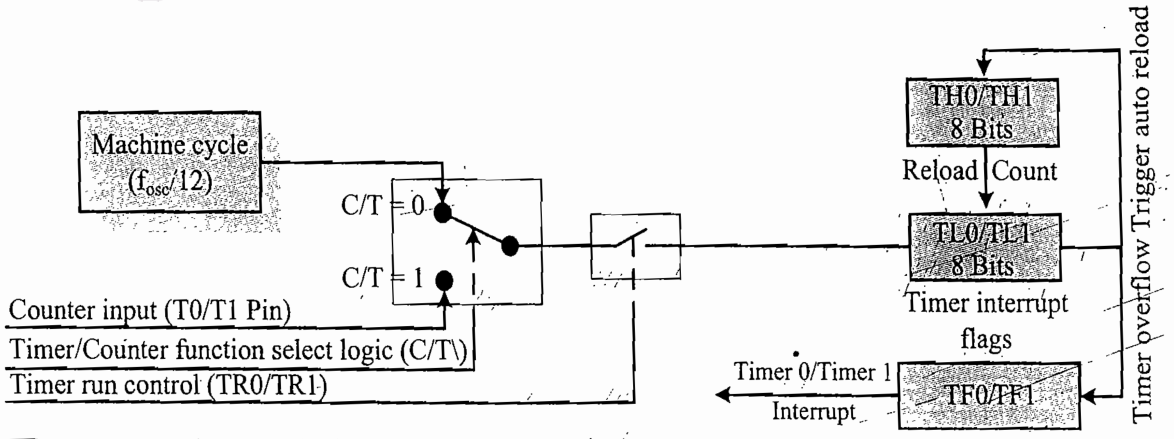

Machine cycle (f_{osc}/12)
C/T = 0
C/T = 1
Counter input (T0/T1 Pin)
Timer/Counter function select logic (C/T\)
Timer run control (TR0/TR1)
TH0/TH1 8 Bits
Reload
Count
TL0/TL1 8 Bits
Timer interrupt flags
TF0/TF1
Timer 0/Timer 1 Interrupt
Timer overflow Trigger auto reload

Fig. 5.29 **Timer (Counter) 0/ Timer (Counter) 1 in Mode 2**

If the counter mode is set, and the corresponding counter run control bit (TR0 for counter 0 and TR1 for counter 1) is 1, the timer register (TL0 for Counter 0 and TL1 for Counter 1) is incremented by one on each traceable logic transitions at the counter input pins (T0 for Counter 0 and T1 for Counter1). The above said actions are followed only when the gating control bit GATE is set to 0. If GATE is set to 1, the timer register is incremented by one for each traceable logic transitions at the counter input pins only if the corresponding Interrupt line INTx (INT0 for Counter 0 and INT1 for Counter 1) is high. The overflow process and interrupt vectoring are similar to that of Timer0/Timer1 operation. Timer 1 in Mode 2 is generally used for baudrate generation in serial data transmission.

5.3.8.4 Timer/Counter in Mode 3 Timer/Counter 0 in mode 3 functions as two separate 8bit Timers/Counters (Fig. 5.30). TL0 acts as the timer/counter register for Timer/Counter 0 and TH0 acts as the timer/counter register for Timer/Counter 1. TL0 uses the Timer 0 control bits namely TR0, GATE, C/T, INT0 and TF0. TL0 can run in either counter/timer mode by setting or clearing the C/T bit. Counter operation is controlled by GATE, INT0 pin, T0 pin and TR0 bit as explained earlier. But the operation is similar to that of Timer 0 in mode 3. In mode 3, TH0 supports only timer function and does not support counter function. TH0 counts only the machine cycles, not external events.

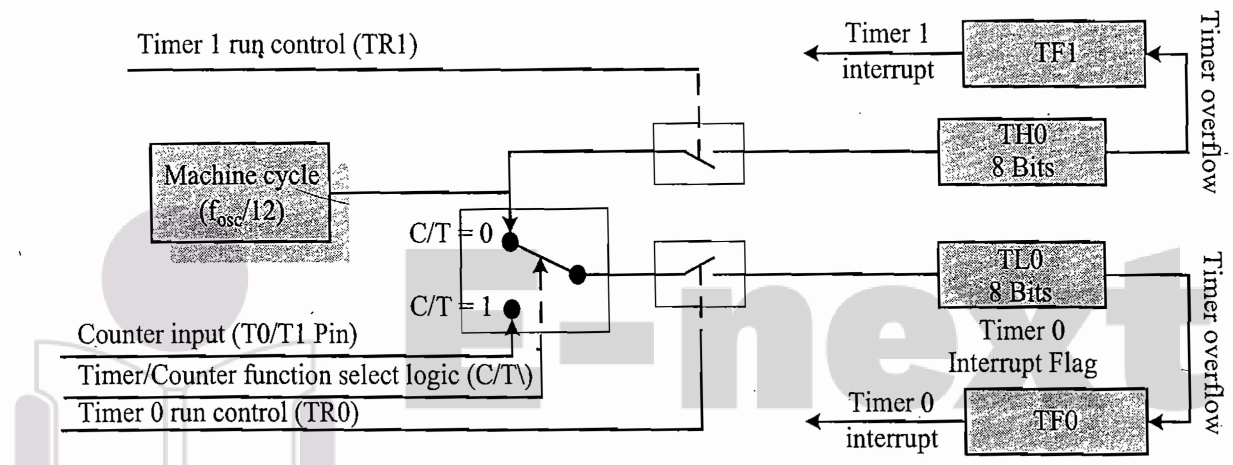

Timer 1 run control (TR1)
Timer 1 interrupt
TF1
Timer overflow
TH0 8 Bits
Machine cycle (f_{osc}/12)
C/T = 0
C/T = 1
TL0 8 Bits
Timer 0 Interrupt Flag
Timer overflow
Counter input (T0/T1 Pin)
Timer/Counter function select logic (C/T\)
Timer 0 run control (TR0)
Timer 0 interrupt
TF0

Fig. 5.30 **Timer (Counter) 0 in Mode 3**

The actual Timer 1 (TH1 TL1), when configured to run in mode 3, by using the Timer 1 configuration bits, stops its functioning and simply holds its count.

Thus mode 3 provides an extra 8bit timer. The original Timer 1 (TH1 TL1) can be turned on and off by switching it out and into mode 3 and it can run in either mode 0, mode 1 or mode 2 apart from mode 3 since the acting Timer 1, TH0 in mode 3 is forced to run as an 8bit timer by default, by putting Timer 0 configuration bits to mode 3 (M0, M1 of Timer 0 Control Register = 1). The only shortcoming is that since TH0 is responsible generating Timer 1 interrupt when Timer 0 is configured in mode 3, no interrupt is generated by the actual Timer 1 (TH1 TL1) when it runs in mode 0, 1 or 2 with Timer 0 in mode 3. But still Timer 1 can use for applications that doesn't require Timer interrupts like baudrate generation.

Example 1

Implement the example 1 given under '*Ports*' section (Binary Counter implementation) using timer interrupts.

In the example given under the '*Ports*' section, the display delay is implemented using a delay generator loop. The delay generation can be easily implemented using a timer and the required operations following the delay can be implemented in the timer interrupt handling routine.

8051 provides two timer units and they can be operated in four different modes. The timer units in mode 1 acts as a 16bit timer unit which counts machine cycles from 0 (0000H) to 65535 (FFFFH) and generates timer interrupt when rolls over from 65535 (FFFFH) to 0 (0000H). If the system clock is 12MHz, the timer interrupt is asserted after every 65536 microseconds (0.0655 seconds), if the counter is started with an initial count 00H. For generating a delay of 1 second, the timer interrupt should be generated multiple times. If one timer interrupt generates 50000 microseconds, 100 such interrupts are required to generate a delay of 5 seconds. For generating a delay of 50000 microseconds, the timer should be loaded with a count 65536 – 50000 = 15536 (3CB0H). The following assembly code illustrates the use of timer and timer interrupts for implementing the binary counter requirement. Timer 0 is used for this.

```
;######################################################################
;binary_counter_timer.src
;Application for implementing binary counter using Timer Interrupt
;The LED is turned on when the port line is at logic 0
;Written by Shibu K V. Copyright (C) 2008
;######################################################################
ORG 0000H                ; Reset vector
        JMP 0050H        ; Jump to main routine
ORG 0003H                ; External Interrupt 0 ISR location
        RETI             ; Simply return. Do nothing
ORG 000BH                ; Timer 0 Interrupt ISR location
        ; The ISR will not fit in 8 bytes size.
        ; Hence it is implemented as separate routine
        JMP TIMER0_INTR ; Function implementing Timer 0 routine
ORG 0013H                ; External Interrupt 1 ISR location
        RETI             ; Simply return. Do nothing
ORG 001BH                ; Timer 1 Interrupt ISR location
        RETI             ; Simply return. Do nothing
ORG 0023H                ; Serial Interrupt ISR location
        RETI             ; Simply return. Do nothing
ORG 0050H                ; Start of Program Execution
        MOV P2, #0FFH    ; Turn off all LEDs
        ORL IE, #10000010B ; Enable Timer 0 Interrupt
        ANL IE, #11100010B ; Disable all interrupts Except Timer 0
        MOV SP, #08H     ; Set stack at memory location 08H
        MOV R7,#00H      ; Set counter Register R7 to zero.
        CLR TR0
        ;Load Timer 0 with count corresponding to 50000 microseconds
        MOV TL0, #0B8H
        MOV TH0, #3CH
        MOV R0, #100     ; Load The multiplier for the time delay
        ; Configure Timer 0 as Timer in mode 1 (16 bit Timer)
        MOV TMOD, #00000001b
        SETB TR0         ; Start Timer 0
        JMP $            ; Loop for ever
;######################################################################
; Routine for handling Timer 0 Interrupt
; Checks whether the timer runs for generating the desired delay
; If so display increment the binary counter and display the count
; Load Timer 0 Register and the multiplier for generating next-
; 5 seconds delay
; If the delay generation is not complete (R0>0) simply return
;######################################################################
```

```
TIMER0_INTR:
        DJNZ R0, RETURN
        INC R7          ; Increment binary counter
        MOV A, R7       ;
        CPL A; The LED's are turned on when corresponding bit is 0
        MOV P2, A ; Display the count on LEDs connected at Port 2
        MOV R0, #100    ; Load the multiplier for the time delay
RETURN: MOV TH0, #3CH
        MOV TL0, #0B8H
        RETI
END     ;END of Assembly Program
```

It should be noted that the Timer 0 ISR involves lot of activities including binary counter increment and display and hence the delay between the successive count display may not be exactly 5 seconds. The timer count can be adjusted to take these operations into account or the displaying of the count can be moved to the main routine and synchronization between main routine and ISR can be implemented through a flag. It is left to the readers as an exercise.

5.3.9 Serial Port

The standard *8051* supports a full duplex (simultaneous data transmission and reception), receive buffered (supports the pipelining system of the reception of second byte before the previously received byte is read from the receive buffer) standard serial interface for serial data transmission. The Special Function Register SBUF provides a common interface to both serial reception and transmission registers. The serial reception and transmission register exist physically as two separate registers but they are accessed by a read/write to the SBUF register. The Serial communication module contains a Transmit control unit and a Receive control unit. The transmit control unit is responsible for handling the serial data transmission and receive control unit is responsible for handling all serial data reception related operations. The Serial Port can be operated in four different modes. The mode selection is configured by setting or clearing the SM0 and SM1 bits of the Special Function Register Serial Port Control SCON.

Serial Port Control Register (SCON) (SFR-98H) SCON is a bit addressable Special Function Register holding the Serial Port Control related bits. The details of SCON bits are given below.

SCON. 7	SCON. 6	SCON. 5	SCON. 4	SCON. 3	SCON. 2	SCON. 1	SCON. 0
SM0	SM1	SM2	REN	TB8	RB8	TI	RI

The following table explains the meaning and use of each bit in the SCON register.

Bit	Name	Description
SM0 SM1	Serial port mode selector	Sets the serial port operation mode

SM2	Multiprocessor communication flag	SM2 = 1 Enable Multiprocessor communication
		SM2 = 0 Disable Multiprocessor communication

REN	Serial data reception enable	REN = 1 Enables reception REN = 0 Disables serial data reception
TB8		9th Data bit that will be transmitted in Modes 2 & 3. Setting/Clearing under software control

RB8		9th Data bit received in Modes 2 & 3. Counterpart for TB8. In Mode 1, if multiprocessor mode is disabled, RB8 will be the stop bit received in serial transmission. RB8 is not used in Mode 0. RB8 is Software controllable
TI	Transmit interrupt	Set by internal circuitry at the end of transmission of the 8th bit in Mode 0. For other modes it is set at the beginning of the stop bit (9th bit). TI should be cleared by firmware
RI	Receive interrupt	Set by internal circuitry at the end of reception of the 8th bit in Mode 0. For other modes it is set on half way of reception of the stop bit (9th bit). RI should be cleared by firmware.

5.3.9.1 Serial Communication Modes As mentioned earlier, *8051* supports four different modes of operation for Serial data communication. The mode is selected by the SM0 and SM1 bits of SCON register. The table given below gives the different combinations of SM0 and SM1 and the serial communication modes corresponding to it.

SM0	SM1	Mode	Type
0	0	0	Shift Register
0	1	1	8 bit UART
1	0	2	9 bit UART
1	1	3	9 bit UART

Mode 0 The Mode 0 operation of serial port is same as that of the operation of a clocked shift register. In Mode 0 operation Pin RXD (Port Pin P3.0) is used for transmitting and receiving serial data and Pin TXD (Port Pin P3.1) outputs the shift clock. 8 data bits are transmitted in this mode with LSB first. The baudrate* is fixed for this mode and it is 1/12 of the oscillator frequency. Mode 0 is half duplex, meaning it supports only unidirectional communication at a time. It can be either transmission or reception. Serial data transmission is initiated by a write access to the SBUF register (Any Instruction that uses SBUF as the destination register. E.g. MOV SBUF, A; ANL SBUF, A; MOV SBUF, #data etc). The write to SBUF loads a 1 to the MSB of the transmit shift register which acts as the stop bit in the serial transmission. The contents of SBUF register is shifted out to the RXD Pin in the order LSB first. The bit shifting occurs in each machine cycle until all bits including the stop bit is shifted out. The bit shift rate is same as the machine cycle rate and hence the transmission rate in mode 0 is 1/12th of the oscillator frequency. The Transmit Interrupt TI is set by the Transmit Control unit when all the 8 bits are shifted out. The TXD pin outputs the transmit shift clock during data transmission. Serial data reception is enabled only when the reception enable bit, REN is set 1 and the Receive Interrupt bit, RI is in the cleared state. A 'write to SCON' (Clear RI/Set REN) initiates the data reception. The receive control unit samples the receive pin, RXD during each machine cycle and places the sample at the LSB of the receive shift register and left shifts the receive shift register. The Receive Interrupt (RI) is set when all the 8 data bits are received. Pin TXD outputs the receive shift clock throughout the reception process.

Mode 1 In Mode 1, serial data pin TXD transmits the serial data and pin RXD receives the serial data. Mode 1 is a full duplex mode, meaning it supports simultaneous reception and transmission of

* Baudrate is defined as the number of bits per second.

serial data. 10 bits are transmitted or received in Mode 1. The 10 bits are formed by the start bit, 8 data bits and one stop bit. The stop bit on reception is moved to RB8 bit of SCON. The baudrate is variable and it can be configured for different bauds. Similar to Mode 0, transmission is triggered by a write access to the SBUF register in Mode 1. A 'write to SBUF' signal loads a 1, which acts as the stop bit, to the 9th bit position of the transmit shift register and informs the transmit control unit that a serial data transmission is requested. Unlike Mode 0 where transmission is synchronized to 'write to SBUF' signal, in Mode 1, the transmission is synchronized to a divide-by-16 counter. The divide-by-16 counter count rate is dependent on the timer 1 overflow rate. The transmission starts with sending the start bit (logic 0), when the divide-by-16 counter rolls over after the 'write to SBUF' signal. The transmit register contents are shifted out to the TXD pin (P3.1 Port pin) at the successive rollover of the divide-by-16 counter. The Transmit Interrupt TI is set by the transmit control unit when all the 8 bits are shifted out. Figure 5.31 illustrates the bit transmission with respect to time in Mode 1.

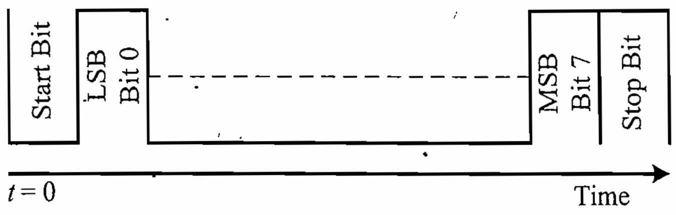


Fig. 5.31 **Bit Transmission with respect to time in Mode 1**

Serial data reception is triggered by a 1 to 0 transition detected at the Receive pin RXD. The RXD pin is sampled at a rate of 16 times the baudrate. On detecting a 1 to 0 transition at the RXD pin, the divided-by-16 counter is reset and 1FFH is loaded into the receive shift register. The divide-by-16 counter divides the bit time of a bit (1/baudrate) into 16 time frames. In order to reduce noise in input data, the RXD pin is sampled at 7th, 8th and 9th frames of the bit time. If the value remains the same for at least two of the samples, it is taken as the data bit. If the value obtained for the first bit time (start bit) is not 0, the receiver circuitry resets and it will look for another 1 to 0 transition as a valid start bit condition. On receiving a valid start bit it is placed in the receive shift register at the rightmost position after shifting the present contents of the receive shift register to the left by one. As data bits enter into the rightmost position, the initially loaded 1s in the receive shift register is shifted out through the left most position. As the reception goes on, when the start bit (the first received 0 bit) reaches at the left most position of the receive shift register, the Receiver unit is informed that only one more bit reception is required and to load the SBUF with the contents of receive shift register after the reception of next bit, move the next receiving bit (stop bit) into RB8 and set the Receive Interrupt RI. The signal to load SBUF and RB8 and set RI is generated only when Receive Interrupt RI = 0 and Bit SM2 in SCON is 0 or received stop bit = 1 at the time of generation of the final shift pulse. If both of these conditions are not met, the received byte is not moved into SBUF else the received byte is moved into SBUF, the received stop bit is placed in the RB8 bit, present in the SCON register and the Receive Interrupt (RI) is activated. After the reception of the stop bit, irrespective of the above-mentioned meeting criteria, the receiver unit looks for the next 1 to 0 transition at the RXD pin for catching the next byte.

Baudrates for Mode 1 As mentioned earlier, the baudrate for Mode 1 is variable. Mode 1 baudrate is dependent on the operating frequency and Timer 1 overflow rate. The bit transmission rate in mode 1 is dependent on the divide-by-16 counter overflow rate. The count rate of divide-by-16 counter is dependent on the timer 1 overflow rate. Depending on the value of the SMOD bit, the divide-by-16 counter is incremented directly on the overflow of Timer 1 or after the two consecutive overflow of the timer 1. If SMOD is 0, the divide-by-16 counter is incremented only after two consecutive overflow of timer 1. If SMOD is 1, the divide-by-16 counter is incremented with the overflow of timer 1. The baudrate in Mode 1 expressed as:

$$\text{Baudrate} = ((2^{SMOD}) \times \text{Timer 1 Overflow rate})/(16 \times 2)$$

For baudrate generation operations, Timer 1 interrupt should be disabled. It is advised to run Timer 1 in the timer mode for baudrate generation operation. Normally Timer 1 is configured in Mode 2 (Auto reload mode) for baudrate generation. In this case the equation will become

$$\text{Baudrate} = 2^{SMOD} \times f_{OSC}/((12 \times [256\text{-TH1}]) \times 16 \times 2)$$

(Timer 1 in mode 2 is incremented at each machine cycle. One machine cycle contains 12 clock periods and so the timer 1 increment rate is f_{OSC} / 12. In Auto reload mode the timer counts from Auto reload value to FF and then rolls over. The machine cycles counted in mode 2 for overflow is expressed as 256 – Reload count = 256 – TH1 (TH1 holds the auto reload count). Hence, Timer 1 overflow rate can be expressed as $f_{OSC}/(12 \times [256\text{-TH1}])$
where

SMOD is the Baudrate multiplier bit present in PCON SFR
f_{OSC} is the oscillator frequency
TH1 is the Timer 1 Auto re-load value.

For achieving low baudrates, Timer 1 can be configured in Mode 1 to run as 16bit timer with interrupt enabled. In Timer 1 interrupt handler write the code to re-load Timer 1 with the count required to get the desired baudrate.

Theoretically the maximum baudrate that can be achieved with Timer 1 running in Mode 1 is given as

$$\text{Max baudrate} = 2^{SMOD} \times f_{OSC}/(12 \times 16 \times 2)$$

Re-load Timer 1 with FFFFH in Timer 1 interrupt handler.

Minimum baudrate that can be achieved with Timer 1 running in Mode 1 is given as Min Baudrate = $2^{SMOD} \times f_{OSC}/(12 \times 16 \times 2 \times 65536)$.

There is no need of re-loading Timer 1

The commonly used baudrates for Mode 1 of serial communication, required oscillator frequency, timer mode and Timer 1 reload value for auto reload mode of Timer 1 are tabulated below. Timer 1 is assumed to be run in the timer mode (Not in counter mode) for all these calculations.

Baudrate	**f_{OSC}(MHz)**	**SMOD**	**Timer 1**	
			Timer Mode	**Reload Value**
9600	11.0592	0	2	FDH
4800	11.0592	0	2	FAH
2400	11.0592	0	2	F4H
1200	11.0592	0	2	E8H
137.5	11.0592	0	2	1DH
2400	11.0592	1	2	E8H
4800	11.0592	1	2	F4H
9600	11.0592	1	2	FAH
19200	11.0592	1	2	FDH
110	6.00	0	2	72H

Mode 1 baudrate 9600 is the most compatible to communicate with PC's COM port.

Modes 2 & 3 11 bits are transmitted in Modes 2 & 3. The 11 bits are formed by the 8 data bits, 1 start bit (0), 1 stop bit (1) and a programmable bit that acts as the 9th data bit. Mode 2 & 3 are also full duplex and serial data is transmitted through the pin TXD and reception is carried out through the pin RXD. TB8 bit of SCON register is transmitted as the 9th bit. TB8 is programmable; it can be set to either 1 or 0. TB8 bit can be used as a parity bit for transmission. The parity bit of PSW reflects the parity of Accumulator content. If the data byte to be sent is the Accumulator content, the parity bit of PSW can be directly moved to TB8 bit of SCON for transmitting it as the 9th data bit, which will give the functionality of a parity enabled serial data transmission. On reception of serial data, the 9th data bit is moved to the RB8 bit of SCON register. If the serial reception is also parity enabled, RB8 can be used for checking the parity of the received byte. The bits of the received byte can be XORed and the resulting bit can be compared to the received 9th bit (which is present in the RB8 bit of SCON) in firmware to check whether the parity bit matches with the parity of the received byte. This ensures error free reception of data. The received stop bit is ignored in Modes 2 & 3. Mode 2 and Mode 3 of Serial communication is similar in all respect except that the Mode 2 and Mode 3 baudrates are different.

Mode 2 baudrate is fixed and is given as $2^{SMOD} \times f_{OSC}/64$

Depending on the SMOD bit, it can be either $f_{OSC}/64$ or $f_{OSC}/32$

Mode 3 baudrate is same as that of Mode 1. Refer back to Mode 1 baudrate for details.

5.3.9.2 Multiprocessor/Controller Communication Support Multiprocessor/controller communication implements communication between multiple processors/controllers connected to the same serial interface bus. Each processor is assigned an address which is set in the firmware.

One of the processor/controller acts as the master controller and the other controllers act as slave controllers (Fig. 5.32).

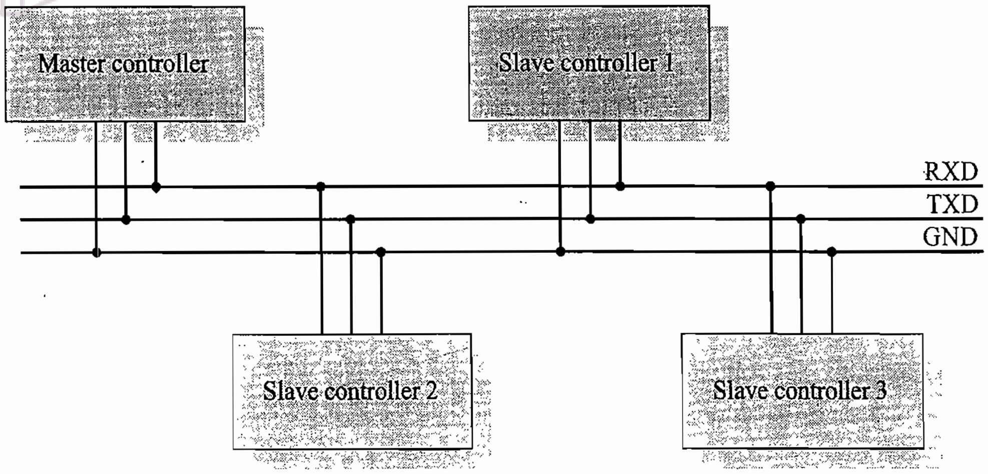


Fig. 5.32 **Multiprocessor communication setup**

Modes 2 & 3 of *8051* serial transmission supports multiprocessor communication if the SM2 bit present in the SCON register is set for each controller. In Modes 2 & 3 operation, 9 data bits and a stop bit are received. The received 9th data bit goes to the RB8 bit of SCON register. With SM2 set, the serial port interrupt is activated on reception of the stop bit only if the received RB8 bit is 1. When the master processor/controller wants to transmit a block of data to any one of the slave controllers, it sends out the address byte of the slave controller, to which it wants to communicate. The 9th data bit that will be sending along with the address byte will be 1. If the multiprocessor mode is enabled in slaves, on receiving the address byte the receive interrupt is activated and each slave controller can use the interrupt handler to examine whether the received address byte is matching to its assigned address (which is stored in the firmware). If a slave finds that the received address byte is matching with its address, it clears its SM2 bit and waits to receive the data byte block from the master. The data bytes are sent along with a 9th data bit which is 0. If the 9th data bit is 1 it implies that the incoming byte is an address byte and if it is 0, it informs the controller that the incoming bytes are data bytes. Once a slave is selected, it clears its SM2 bit and generates interrupt for all incoming data bytes while all other slaves ignore the received bytes since their SM2 bit is still in the set condition. To enable the multiprocessor communication again in the selected slave which is presently involved in the transmission it should be informed the end of reception of data bytes through some mechanism. On receiving the end of data byte reception information, the slave sets the SM2 bit and brings back the slave controller into multiprocessor communication enabled mode.

Setting SM2 bit in Mode 0 will not produce any effect in transmission/reception. If SM2 is set in Mode 1 the validity of the received stop bit is checked. If SM2 = 1 and Mode = 1, receive interrupt RI is generated only if the received stop bit is 1.

Example 1

Design an *8051* microcontroller based system for serial data communication with a PC as per the requirements given below.

1. Use Atmel's AT89C51/52 or AT89S8252 (Flash microcontroller with In System Programming (ISP) support) for designing the system.
2. The baudrate for communication is 9600.
3. The serial data transmission format is 1 Start bit, 8 data bits and 1 Stop bit.
4. The system echoes (sends back) the data byte received from the PC to the PC.
5. On receiving a data byte from the Serial port it is indicated through the flashing of an LED connected to a port pin.
6. Use Maxim's MAX232 RS-232 level shifter IC for converting the TTL serial data signal to RS-232 compatible voltage levels.
7. Use on-chip program memory for storing the program instructions.
8. Use the 'Hyper Terminal' application provided by Windows Operating system for sending and receiving serial data to and from the microcontroller.

The *8051* microcontroller has a built-in serial communication module which is capable of operating in different modes with different data rates (baudrate). The only limitation is that the serial data is transmitted and received in either TTL/CMOS logic. (The voltage levels are the one corresponding to logic 0 and logic 1 in the respective logic family.) The serial communication supported by PC follows the RS-232 Serial communication standard and the voltage levels are entirely different from the TTL/CMOS logic. The voltage levels for RS-232 is +/–12 V (We will discuss about the RS-232 Protocol, voltage level and electrical characteristics in a dedicated book, which is planned under this book series.) For translating the RS-232 voltage levels to the TTL/CMOS compatible voltage level, an RS-232 translator chip is required between the PC interface and the microcontroller interface. The RS-232 translator chip is available from different chip manufacturers and the MAX232 IC from Maxim Dallas Semiconductors is a very popular level translator IC. The Serial data is usually routed to an RS-232 connector (DB Connector). The RS-232 connector comes in two different pin counts namely; DB9 – 9 Pin connector and DB25 – 25-Pin connector. For proper data communication, the Transmit Pin of one device (Here either PC or microcontroller) should be connected to the receive pin of the other device and vice versa. This can be done at the microcontroller board side by connecting the TXDOUT pin of MAX232 to the RX (Receive) pin of DB connector and the RXDIN Pin of MAX232 to the TX (Transmit) pin of DB9 connector. An RS-232 cable is used for connecting the PC COM Port with DB Connector of the microcontroller board. For the above configuration, the RS-232 cable should be 1-1 wired meaning, the RXD and TXD lines of the connectors present at both ends are 1-1 connected. If the Serial lines are not used in the cross over mode in the microcontroller board, a twisted cable (cable with RXD pin connected to TXD pin of the other end connector and TXD pin connected to RXD pin) can be used for establishing the proper communication. The hardware connection for implementing the serial communication is given in Fig. 5.33.

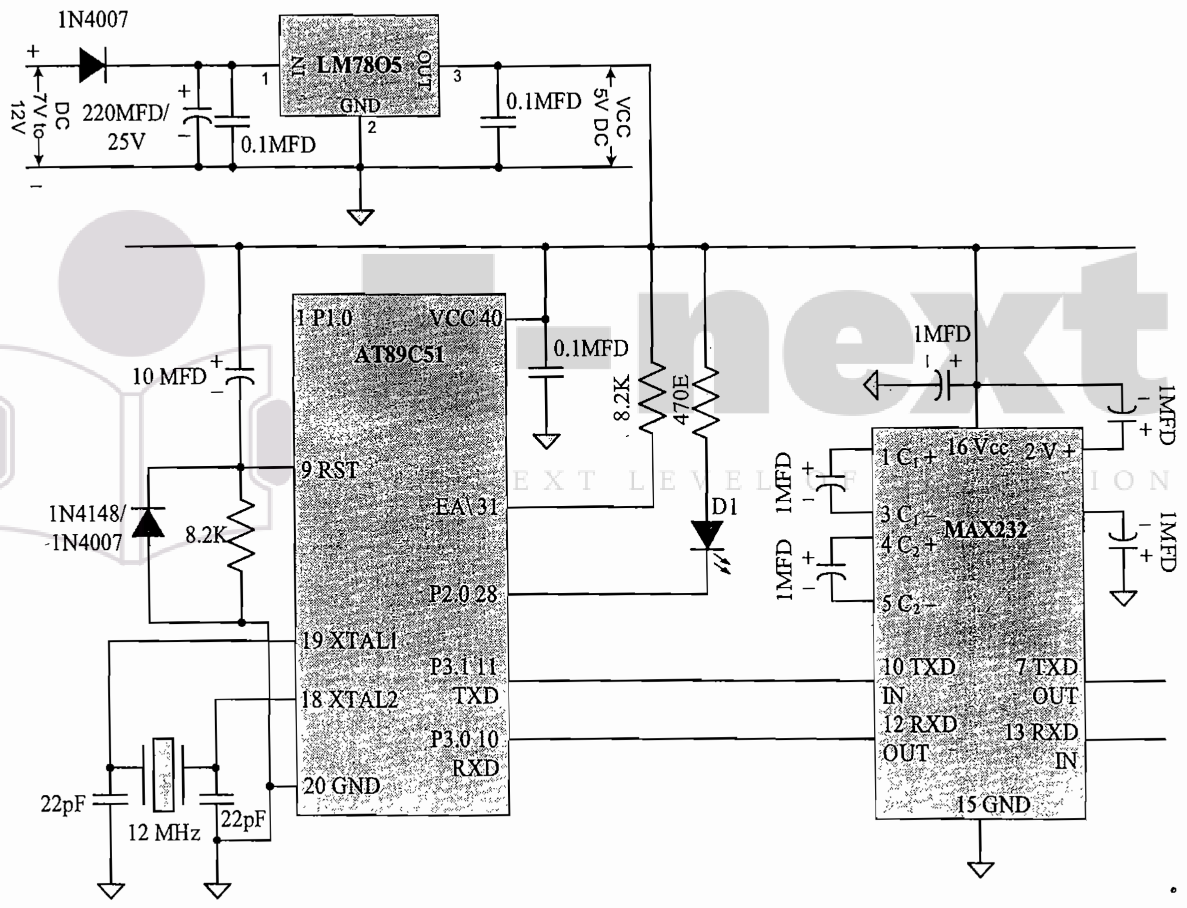


Fig. 5.33 **Serial communication circuit implementation using *8051***

For implementing the serial data communication as 1 Start bit + 8 Data bits + 1 Stop bit with a data rate (baudrate) of 9600, the serial port should be configured in mode 1. Timer 1 should be configured to run in auto reload mode (Mode 2) to generate the required baudrate. With an operating clock frequency of 11.0592 MHz, Timer 1 in mode 2 should be reloaded with a count 0FDH for generating a baudrate of 9600 (Refer section **Baudrates for Mode 1:** for more details on baudrate generation for serial communication in mode 1).

The flow chart given in Fig. 5.34 illustrates the firmware design for implementing serial data communication.

The firmware for Implementing Serial communication in 8051 Assembly language is given below.

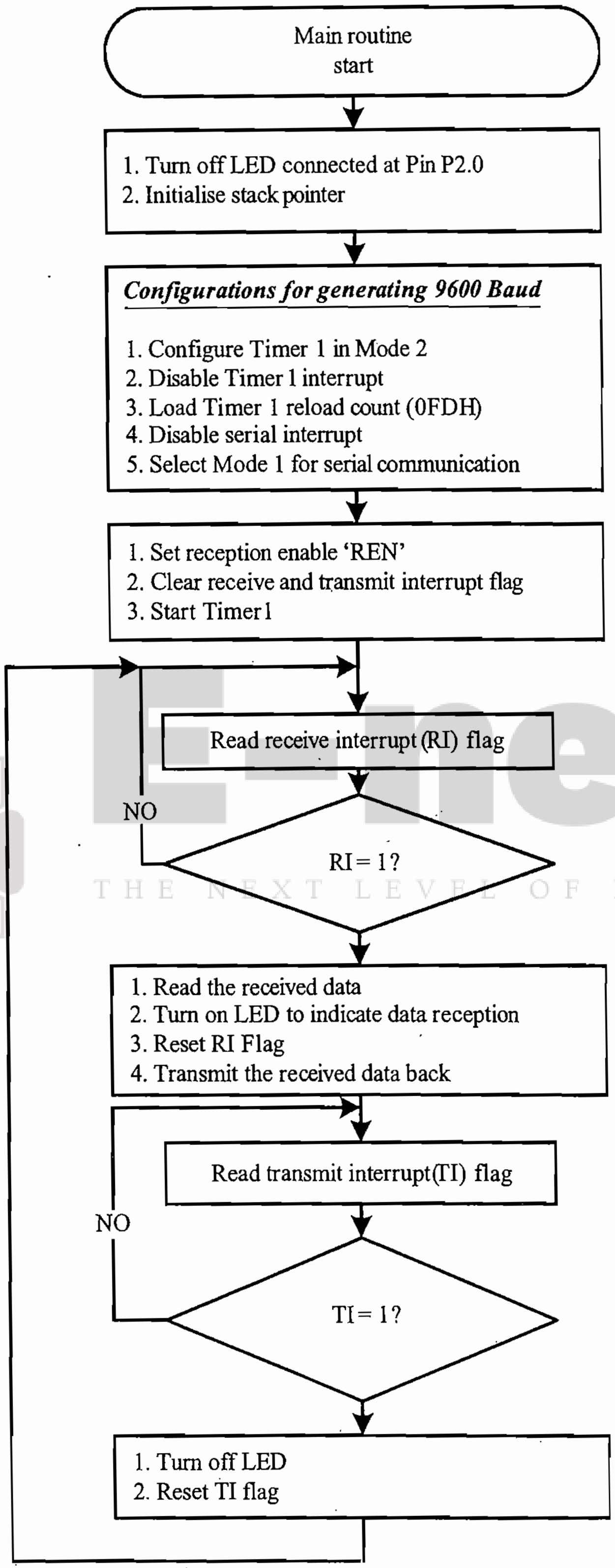

Main routine start
1. Turn off LED connected at Pin P2.0
2. Initialise stack pointer
Configurations for generating 9600 Baud
1. Configure Timer 1 in Mode 2
2. Disable Timer 1 interrupt
3. Load Timer 1 reload count (0FDH)
4. Disable serial interrupt
5. Select Mode 1 for serial communication
1. Set reception enable 'REN'
2. Clear receive and transmit interrupt flag
3. Start Timer1
Read receive interrupt (RI) flag
NO
RI = 1?
1. Read the received data
2. Turn on LED to indicate data reception
3. Reset RI Flag
4. Transmit the received data back
Read transmit interrupt(TI) flag
NO
TI = 1?
1. Turn off LED
2. Reset TI flag

Fig. 5.34 **Flow chart for implementing polling based serial communication**

```
;###################################################################
;serial_polling.src
;Firmware for Implementing Serial Communication with PC
;Written by Shibu K V. Copyright (C) 2008
;###################################################################
ORG 0000H               ; Reset vector
        JMP 0050H       ; Jump to start of main program
ORG 0003H               ; External Interrupt 0 ISR location
        RETI            ; Simply return. Do nothing
ORG 000BH               ; Timer 0 Interrupt ISR location
        RETI            ; Simply return. Do nothing
ORG 0013H               ; External Interrupt 1 ISR location
        RETI            ; Simply return. Do nothing
ORG 001BH               ; Timer 1 Interrupt ISR location
        RETI            ; Simply return. Do nothing
ORG 0023H               ; Serial Interrupt ISR location
        RETI            ; Simply return. Do nothing
;###################################################################
ORG 0050H               ; Start of main program
        SETB P2.0       ; Turn off data reception indicator LED
        MOV SP, #08H    ; Set stack at memory location 08H
        CLR TR1         ; Stop Timer 1
        MOV TMOD, #00100000B ; Set Timer 1 in Mode 2 (Autoreload)
        CLR EA          ; Disable all interrupts
        MOV TH1, #0FDH  ; Reload count for Timer 2
        MOV TL1, #0FDH
        ; Select Mode 1 for serial Communication
        ; SM0 = 0, SM1 = 1, SM2 = 0
        ; Set REN
        ; TB8 = 0, RB8 = 0, TI =0, RI =0
        MOV SCON, #01010000B
        ANL PCON, #01111111B ; Reset baudrate doubler bit
        SETB TR1        ; Start Timer 1
        ; Read Receive Interrupt (RI) flag
        ; Check whether data reception occurred
        ; If not, loop till RI=1
RECEIVE:JNB RI, RECEIVE
        ; Data received
        CLR P2.0        ; Turn ON LED to indicate data reception
        ; Read the received data
        MOV A, SBUF
        CLR RI          ; Reset RI flag
        MOV SBUF, A     ; Transmit back the received data
        ; Read Transmit Interrupt (TI) flag
        ; Check whether data transmission completed
        ; If not, wait till completion (loop till TI=1)
        JNB TI, $
        ; Data Transmitted
        SETB P2.0       ; Turn off Indicator LED
        CLR TI          ; Reset Transmit Interrupt flag
        JMP RECEIVE     ; Wait for next data to arrive
END     ; END of Assembly Program
```

The firmware polls the serial interrupt flags and takes the corresponding actions when either the receive interrupt or transmit interrupt flag is set. This approach is CPU-intensive and the CPU is dedicated for serial data communication. Another approach in which the serial communication is handled is the interrupt based communication approach. The flow chart given in Fig. 5.35 models the interrupt based serial communication.

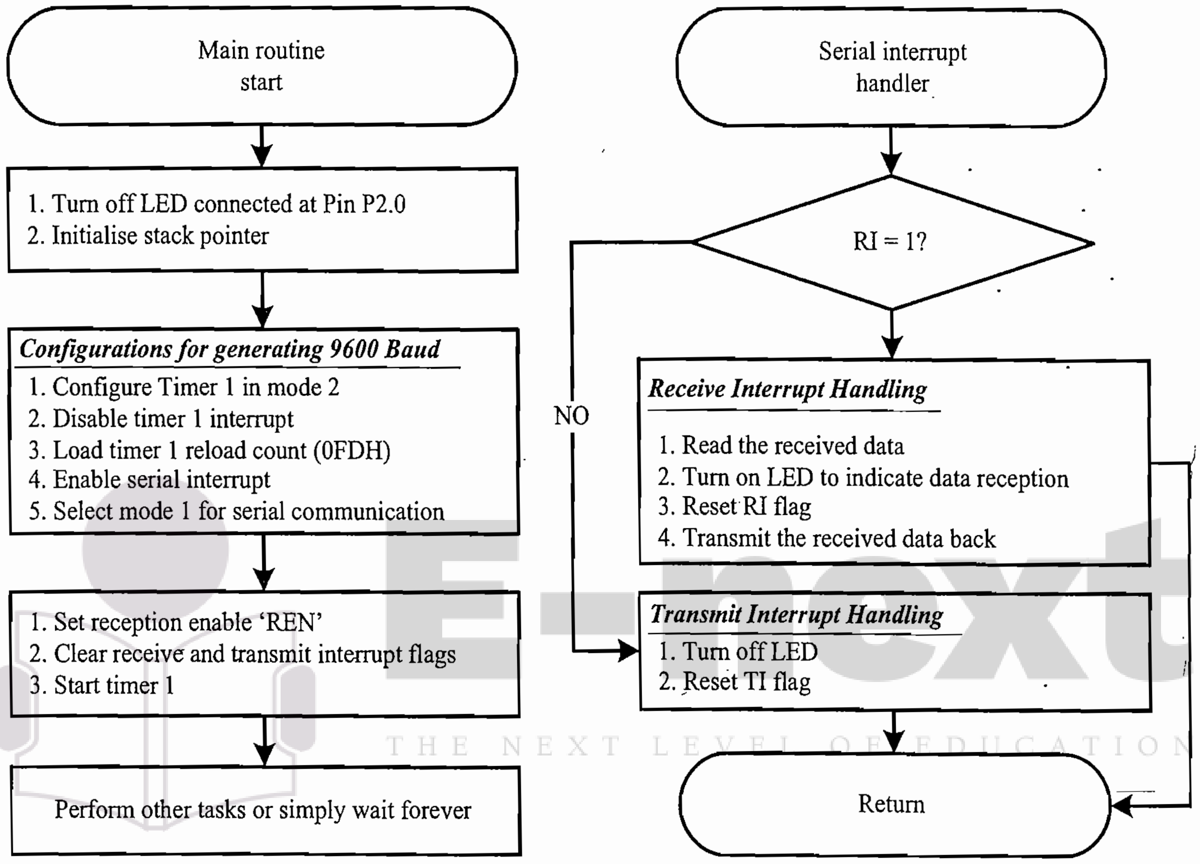

Main routine start
1. Turn off LED connected at Pin P2.0
2. Initialise stack pointer
Configurations for generating 9600 Baud
1. Configure Timer 1 in mode 2
2. Disable timer 1 interrupt
3. Load timer 1 reload count (0FDH)
4. Enable serial interrupt
5. Select mode 1 for serial communication
1. Set reception enable 'REN'
2. Clear receive and transmit interrupt flags
3. Start timer 1
Perform other tasks or simply wait forever
Serial interrupt handler
RI = 1?
NO
Receive Interrupt Handling
1. Read the received data
2. Turn on LED to indicate data reception
3. Reset RI flag
4. Transmit the received data back
Transmit Interrupt Handling
1. Turn off LED
2. Reset TI flag
Return

Fig. 5.35 **Flow chart for implementing interrupt based serial communication**

The firmware implementation for the interrupt based serial data communication is given below.

```
;####################################################################
;serial_interrupt.src
;Firmware for Implementing Serial Communication with PC
;Serial Data reception and transmission handled through Interrupts
;Written by Shibu K V. Copyright (C) 2008
;####################################################################
ORG 0000H           ; Reset vector
        JMP 0050H   ; Jump to start of main routine
ORG 0003H           ; External Interrupt 0 ISR location
        RETI        ; Simply return. Do nothing
ORG 000BH           ; Timer 0 Interrupt ISR location
        RETI        ; Simply return. Do nothing
ORG 0013H           ; External Interrupt 1 ISR location
        RETI        ; Simply return. Do nothing
ORG 001BH           ; Timer 1 Interrupt ISR location
        RETI        ; Simply return. Do nothing
ORG 0023H           ; Serial Interrupt ISR location
                    ; Interrupt handling will not fit in 8 bytes
        ; Call the routine where interrupt is handled
        CALL SERIAL_INTERRUPT
        RETI                ; Return
;#####################################################################
ORG 0050H                   ; Start of main Program
        SETB P2.0           ; Turn off data reception indicator LED
        MOV SP, #08H        ; Set stack at memory location 08H
        CLR TR1             ; Stop Timer 1
        MOV TMOD, #00100000B ; Set Timer 1 in Mode 2 (Autoreload)
        MOV IE, #10010000B ; Enable only serial interrupt
        MOV TH1, #0FDH ; Reload count for Timer 2
        MOV TL1, #0FDH
        ; Select Mode 1 for serial Communication
        ; SM0 = 0, SM1 = 1, SM2 = 0.
        ; REN = 1, TB8 = 0, RB8 = 0, TI = 0, RI = 0
        MOV SCON, #01010000B
        ANL PCON, #01111111B ; Reset baudrate doubler bit
        SETB TR1            ; Start Timer 1
        JMP $               ; Simply Loop here
#####################################################################
;Routine for handling Serial Interrupt (Both RI & TI)
;Only one Interrupt available for Transmission and Reception
;Check whether the interrupt is TI or RI
;Perform the actions corresponding to the Interrupt and then Return
;#####################################################################
SERIAL_INTERRUPT:
        ; Check whether the interrupt is Receive Interrupt (RI)
        JNB RI, CHECK_TI
        ; Receive Interrupt occurred
        ; Data received
        CLR P2.0            ; Turn ON LED to indicate data reception
        MOV A, SBUF         ; Read the received data
        CLR RI              ; Reset RI flag
        MOV SBUF, A         ; Transmit back the received data
        ; Receive Interrupt processing completed
        ; Return
        RET
CHECK_TI:
        ; Transmit Interrupt occurred
        ; Data Transmitted
        SETB P2.0           ; Turn off Indicator LED
        CLR TI              ; Reset Transmit Interrupt flag
        ; Transmit Interrupt processing completed
        ; Return
        RET
END     ;END of Assembly Program
```

This design will not be complete without explaining the PC side application responsible for sending and receiving serial data to and from the *8051* based system connected to the serial port (Communication Port 1, COM 1) of the PC. The *8051* based system should be connected to the serial port using one of the cables (twisted/ one-to-one), as per the design requirement, as explained earlier. Figure 5.36 illustrates the interfacing of the microcontroller board to the COM port of the PC.

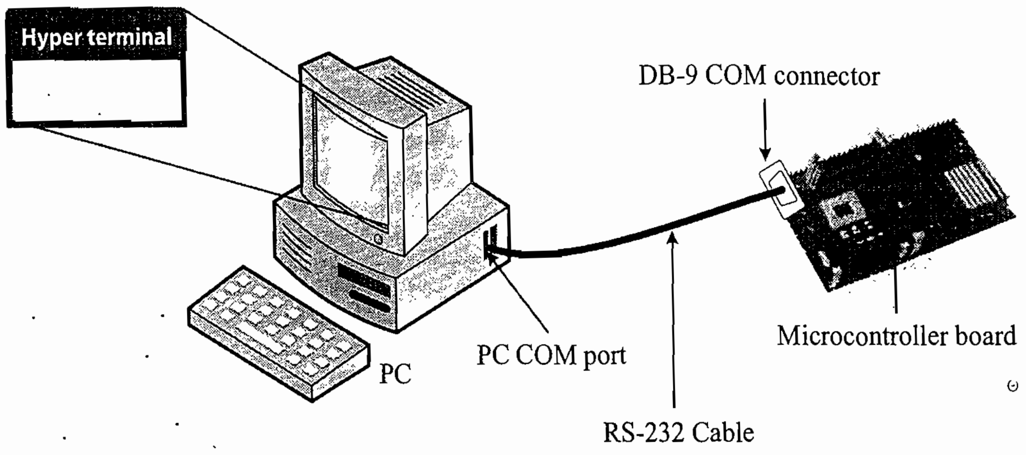


Fig. 5.36 **Serial Port Interfacing with PC**

The Microsoft ® Windows Operating System (Windows 9x/XP is used in our example) provides a built-in serial communication terminal program called '*Hyper terminal*'. The '*Hyper terminal*' application can be launched through *Start* → *All Programs* → *Accessories* → *Communications* → *Hyper terminal*

The *Hyper terminal* application will request for a *Connection Description* while launching a new instance of the application. Provide a connection description name (say *8051*) in the *Connection Description* wizard. Discard the *Connect To* wizard by *Cancel*. Select *Properties* tab from the *File* menu of the *Hyper terminal* application. A new window describing the *Properties* of the *Hyper terminal* connection is displayed. From the *Connect To* tab of the new Window, select the com number to which the 8051 system is connected, say COM1, against the *Connect using:* option. Now select the *Configure..* button. A new configuration window is displayed for configuring the *Port Settings* for the selected COM port. Configure the *Bit per second:* to 9600, *Data bits:* to 8, *Parity:* to *None*, *Stop bits:* to 1 and *Flow control:* to *None*. Save the *Port Settings* through the *Apply* button. Exit the *Port Settings* window by invoking the *OK* button. Start the communication by invoking the *Call* option from the *Call* menu (This can also be achieved by pressing the *Telephone* icon on the hyper terminal window). Now you are ready to communicate with the 8051 system connected to the communication port configured as per the requirement. Start sending data by typing on the hyper terminal application. You can see the data echoed by the 8051 system on the hyper terminal application. The hyper terminal application can be stopped by invoking the *Disconnect* option from the *Call* menu. (This can also be achieved by pressing the *Telephone Hang-up* icon on the hyper terminal window.)

5.3.10 Reset Circuitry

Reset is necessary to bring the different internal register values and associated internal hardware circuitry to a known state. Before putting the 8051 controller into operation a reset should be applied to the controller. As mentioned earlier, applying a reset loads the registers with default known values and loads the Program Counter (PC) with 0000H. This ensures that the program execution starts from the start of the code memory (0000H) and stack will reset to the memory location 07H. Reset can be of two types: hardware and software reset. Hardware reset brings all associated hardware units to a well-defined initial state. *8051* provides a pin named RST for applying the hardware reset.

The reset signal must be kept active until all the three of the following conditions are met

1. The power supply must be in the specified range.
2. The oscillator must reach a minimum oscillation level to ensure a good noise to signal ratio and a correct internal duty cycle generation.
3. The reset pulse width duration must be at least two machine cycles (24 Clock periods) when conditions 1 and 2 are met.

If one of the conditions is not satisfied, the microcontroller may not startup properly. To ensure a good startup, the reset pulse width has to be wide enough to cover the period of time, where the electrical conditions are not met. Please refer to the note given on the 'On line Learning Centre', on the important parameters that should be taken into account for determining the reset pulse width.

The C51 family of microcontrollers support two kinds of reset input. The first one is a pure input which allows an external device or circuitry to reset the microcontroller. The second one is bidirectional, in which the microcontroller is capable of resetting an external device. In actual practice a power on reset (unidirectional input) is given to the *8051* controller by connecting an external resistor and capacitor as shown in Fig. 5.37.

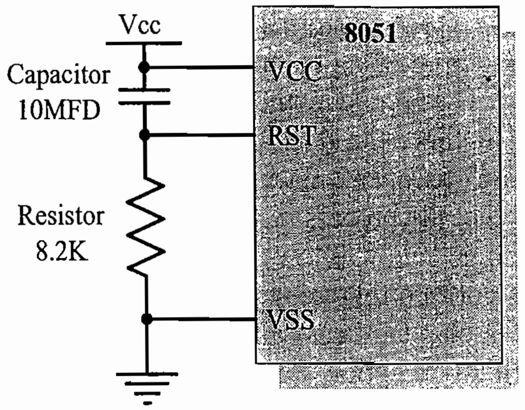


Fig. 5.37 Power on reset circuitry for *8051*

When the power supply is turned on, initially the capacitor acts as short circuit and the full applied voltage appears across the resistor (RST pin becomes at logic 1). The capacitor starts charging gradually and the applied voltage is build across the capacitor (RST pin becomes at logic 0). The time taken by the capacitor to get charged to a voltage V_c, which brings the RST pin to logic LOW, is calculated as

$$V_c = V_f - (V_f - V_i) \times e^{-t/RC}$$

V_c = voltage at which RST pin logic high changes to LOW

V_f = final voltage (VDD =5 V)

V_i = initial voltage (=0 V)

RC = time constant of resistor capacitor circuit $(8.2 \times 10^3 \times 10 \times 10^{-6})$

The reset pulse width 't' can be calculated from the above equation with V_c = The voltage across RST pin which makes the RST pin at logic 0 and V_f = 5V. The pulse width will be of the order of milliseconds to seconds. This ensures that the RST pin remains HIGH for enough time to allow the oscillator to start up (A few milliseconds), the power supply to stabilise plus two machine cycles. It is to be noted that the port pins will be in a random state until the oscillator is started and the internal reset algorithm writes 1 to the corresponding port SFRs.

5.3.11 Power Saving Modes

For applications like battery operated systems where power consumption is critical, the CMOS version of *8051* provides power reduced modes of operation namely *IDLE* and *POWER DOWN* modes.

5.3.11.1 IDLE Mode The *IDLE* mode is activated by setting the bit PCON.0 of the SFR power control register (PCON). The instruction setting the PCON.0 bit pushes the processor into an idle state where the internal clock to the processor is temporarily suspended. Before putting the CPU into idle state, the various CPU statuses like Stack Pointer (SP), Program Counter (PC), Program Status Word (PSW) and Accumulator and all other register values are preserved as such. All port pins will hold the logical states they had at the moment at which the idle mode is activated by setting the PCON.0 bit. ALE and PSEN remains at logic high during Idle Mode. The interrupt, Timer and Serial port sections continue functioning since the clock signal is not withdrawn for the same. The Idle mode is terminated by asserting any enabled interrupt. The interrupt can be any one of the external interrupt (INT0 or INT1) or Serial Interrupt. On asserting the interrupt, the IDL bit (PCON.0) is cleared and the interrupt routine is serviced. After executing the RETI instruction, the next instruction executing will be the instruction which follows immediately the instruction that put the processor in Idle Mode. Two general purpose flag bits GF0 & GF1 present in the PCON register can be used for giving an indication for whether an interrupt occurred in normal mode or Idle Mode. Since PCON register is not a bit addressable register, the instruction which sets the IDLE Mode control bit PCON.0 can also set the flags either GF0 or GF1 or both. When an interrupt occurs its service routine can examine whether the interrupt occurred in the Idle mode or Normal mode by checking the status of GF0 or GF1 bits of PCON (valid only if the Idle mode enabling instruction sets the GF0/GF1 bit along with PCON.0).

The second method for terminating the idle mode is the application of a reset input signal at the RST pin. Since the clock oscillator is already running and it is in the stable state, the RST pin need to be held high for only 2 machine cycles. Resetting the processor clears the IDL bit (PCON.0) directly and at this point the processor resumes program execution from where it left off, that is, at the instruction which immediately follows the one that invoked the idle mode. When the internal circuitry undergoes the reset algorithm following the Reset, two or three machine cycles of program execution may take place. The On-chip hardware inhibits any read/write access to the scratchpad RAM during the internal reset algorithm execution, but access to the port pins are not inhibited. Hence it is advised to put a minimum of 3 NOP instructions which follows the instruction that triggers the idle mode.

Power Control Register (PCON) (SFR-87H) It is the Special Function Register holding the power control bits and baudrate multiplier bit. PCON is not a bit addressable register. The details of PCON bits are given below.

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
SMOD	RSD	RSD	RSD	GF1	GF0	PD	IDL

The following table explains the meaning and use of each bit in the PCON register.

Bit	Name	Description
SMOD	Baudrate multiplier	If Serial port is operating in Modes 1, 2 or 3 and timer 1 is used for baudrate generation, setting SMOD to 1 doubles the baudrate
RSD	Reserved	Unimplemented. Reserved for future use. Users are advised not to modify it
GF1	General purpose flag bit 1	User programmable bit. Can be set or reset under firmware control
GF0	General purpose flag bit 0	User programmable bit. Can be set or reset under firmware control
PD	Power down control bit	PD = 1 Invokes power down mode.
IDL	Idle mode control bit	IDL = Invoke Idle mode

The NMOS version of 8051 devices implement only SMOD bit in PCON register. The other 4 bits are available only in CMOS versions. Reset value of PCON is 0XXX0000 (X denotes don't care. It may be either 0 or 1). PD bit always take precedence over IDL bit and if an instruction sets both IDL and PD bits to 1, action corresponding to PD = 1 is performed and it masks the IDL bits activity.

5.3.11.2 Power Down Mode When the PD bit (PCON.1) is set by an instruction, the Power Down mode is activated. The CPU stops executing instructions; clock signal to all internal hardware including timer unit, interrupt system and CPU is terminated. The contents of internal RAM and SFR are maintained. All ports continue emitting their respective SFR contents. ALE and PSEN signals are held at low. Power Down mode can only be terminated through hardware reset. All SFRs are brought into their reset

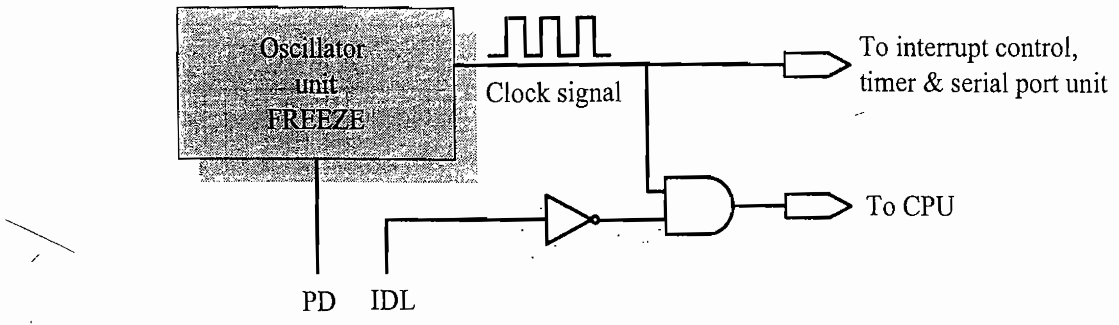


Fig. 5.38 **Power down and IDLE mode operation**

values. However the on-chip RAM retains their contents to that of the values when the Power Down mode is entered. The supply voltage V_{cc} to the controller can be brought down to as low as 2V when the controller is in Power Down mode. However before terminating the Power Down mode with a hardware reset, the supply voltage V_{cc} should be brought into the normal operating level. If the hardware reset is applied to the controller to terminate the processor before bringing the operating voltage to the nominal level, the controller may behave in an unexpected way. Hence reset should not be activated before V_{cc} is brought to its normal operating level, and the reset signal should be held active till the oscillator is re-started and becomes stabilised.

5.3.11.3 Power Consumption V/s Oscillator Frequency Average Power consumption is directly proportional to the operating frequency (Clock frequency). Increasing the clock frequency (subject to the condition that it will not exceed the maximum supported frequency by the system core) increases the execution speed of the controller. But it also has a direct impact on the total power consumption. Figure 5.39 illustrates the typical power curve for a generic *8051* microcontroller.

Generally, the current consumption v/s operating frequency characteristics is linear with a dc offset. The dc quiescent current is generated by static on-chip circuitry such as op amps, comparators, etc. This current is a constant drain current typically in the range of 1mA. From the power curve it is obvious that as the operating speed demand is high, the power consumption also goes high. So there is always a trade-off between processing power and power consumption with the generic *8051* design. Some techniques used to achieve high processing speeds by maintaining the power consumption to the least possible are explained below.

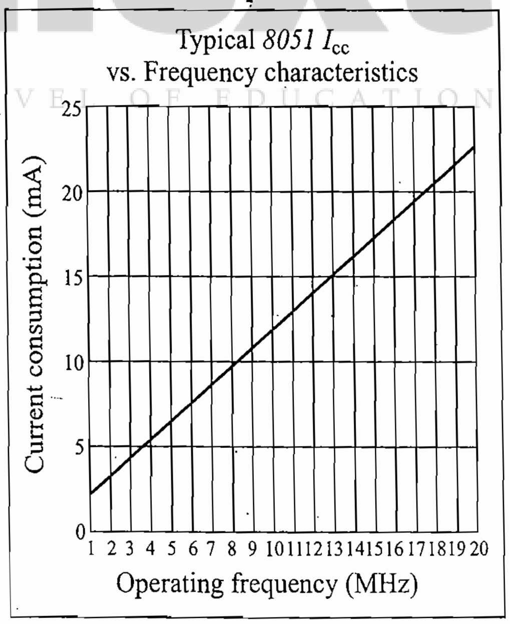


Fig. 5.39 **Typical Power consumption curve for generic *8051***
Copyright Maxim Integrated Products (http://www.maxim-ic.com) used by permission

High-speed core The original *8051* design was based on a 12 clock/machine cycle and two fetches per machine cycle architecture. The high speed core uses 4 clocks or 1 clock per machine cycle design. Comparing this with the original 12 clock/machine cycle for the same operating frequency, the performance will be approximately 3 times for the 4 clock core and 12 times for the 1 clock core preserving the same power consumption. Figure 5.40 gives a comparative measure of power consumption for 80C31 and 80C32 core based on 12 clock/machine cycle and Maxim/Dallas' DS80C320 core based on 4 clock/machine cycle design.

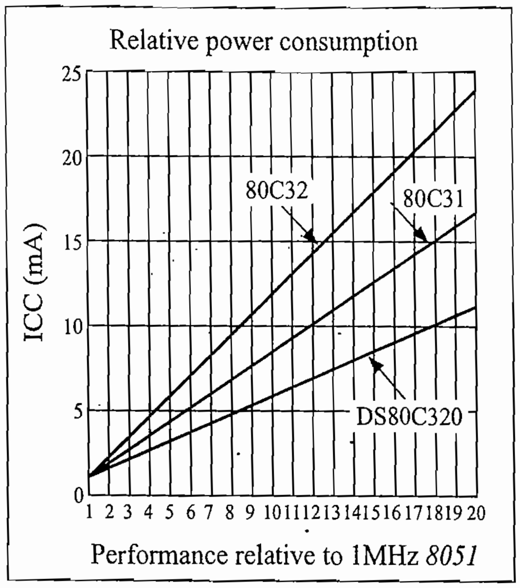


Fig. 5.40 **Power Consumption curve for normal core v/s Reduced Clock cycle**
Copyright Maxim Integrated Products (http://www.maxim-ic.com) used by permission

Use of Single Chip with integrated peripherals A typical *8051* microcontroller based embedded system incorporates a number of peripherals ranging from external UART to reset ICs (e.g. DS1232 from Maxim Integrated Products/Dallas Semiconductor), brown-out circuits and watch dog timers. Using these chips as separate ICs will definitely increase the total system power consumption. An effective way of reducing this kind of power consumption is the use of a single chip that integrates the *8051* controller and the related necessary chips in a single chip. Apart from significant reduction in power consumption, this approach also reduces the space required for placing the components on the PCB and thereby makes the embedded product much more compact one.

Use of On-chip program memory and RAM Use of external Program Memory (EEPROM/FLASH) and RAM requires an additional latch circuit (e.g. 74LS373) to latch the lower order address bus from the multiplexed address/data bus. Adding this chip will again increase the total system power. Nowadays controllers with internal program memory ranging from 4K to 64K are available in the market. Some manufacturers provide extra RAM up to 1 KB as on-chip RAM.

Clock source The standard *8051* design uses either an external quartz crystal resonator to excite the internal on-chip oscillator circuitry or an external standalone crystal oscillator. If an external crystal oscillator is used, the waveform of the clock can affect power consumption. The input stage of the XTAL1 pin, used for driving the external clock signals into the *8051*, typically employs complementary drivers. As the input clock transitions between high and low, the drivers will momentarily both be ON, causing a significant current rush. With a square wave clock signal, the transition between high and low is almost instantaneous and the time in which both drivers are ON is minimised. A waveform with a slower rise and fall time, such as a sine or triangle wave, will take long time to complete the transition and both drivers will be on for a long time. This will increase the current and power consumption. A comparison diagram for power consumption for different clock waves is given in Fig. 5.41.

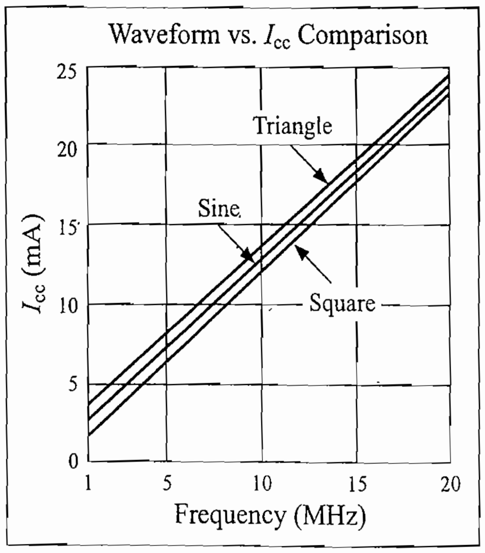


Fig. 5.41 **Power consumption curve for different clock waves**
Copyright Maxim Integrated Products (http://www.maxim-ic.com). used by permission.

5.3.11.4 On Circuit Emulation (ONCE) Mode The On Circuit Emulation (ONCE) Mode helps in testing and debugging the system without removing the microcontroller from the circuit. The ONCE mode is invoked through the following steps:

1. Pulling the ALE pin while the controller is in reset and PSEN is high
2. Holding the ALE pin low as Reset is de-activated

In ONCE mode, Port 0 pins stay in the floating state and other Port pins, ALE and PSEN are pulled high weakly internally. The oscillator circuit remains active. With these conditions an emulator or another controller can be used for driving the circuit. Normal operation of the target controller can be restored by applying a normal reset.

5.4 THE *8052* MICROCONTROLLER

The *8052* microcontroller is a member of the *8051* family and it can be considered as the 'big' brother of *8051*. *8052* is pin to pin compatible with the standard *8051* microcontroller. The only difference between the *8051* and *8052* is that *8052* contains certain additional functionalities. The *8052* architecture implements the upper 128 bytes of user data RAM physically on the chip and application programmers can access this memory through indirect addressing, whereas the standard *8051* architecture doesn't implement the upper 128 bytes of data RAM physically on the chip. *8052* also implements an additional 16bit timer (Timer 2) and thus the total number of timers available in *8052* is three, whereas the standard *8051* architecture implements only two timers. T2CON is the SFR register implemented in *8052* for controlling the timer 2 operations. The 16bit timer register is formed by the register pair TH2 and TL2. Timer 2 supports interrupt and the interrupt vector location for Timer 2 is 002BH. Timer 2 supports a special mode of operation called 'Capture Mode' and when the timer is in capture mode the contents of TH2 and TL2 is captured to the capture SFR RCAP2H and RCAP2L respectively when a 1 to 0 transition is detected at the P1.1 pin of the microcontroller. The timer 2 interrupt is also generated if the timer 2 interrupt is in the enabled state.

5.5 *8051/52* VARIANTS

Lot of variants are available for the basic *8051* architecture from different microcontroller manufactures including Atmel, Maxim/Dallas, etc. We will have a glance through on some of them.

5.5.1 Atmel's AT89C51RD2/ED2

The AT89C51RD2 from Atmel Corporation is a high speed *8052* core compatible microcontroller. It contains six 8bit I/O ports, three 16bit Timers/Counters, 256 bytes of user data RAM, 9 interrupt sources with 4 levels of priority and 64 Kbytes of on-chip In System Programmable FLASH memory for program storage. It also contains 1792 bytes of on-chip expanded RAM (XRAM), SPI port, pulse width modulation (PWM) unit, integrated watch dog timer (WDT), integrated power monitor for Power On reset and power supply fail detect, dual full duplex serial port and dual Data Pointer Register (DPTR). The 89C51RD2 controller can be operated in standard mode and high-speed mode. The standard mode of operation requires 12 clocks per machine cycle, whereas the high-speed mode (Also known as X2 mode) requires only 6 clock periods per machine cycle for instruction fetch and execution. Standard mode supports clock speed of up to 60 MHz and high speed mode supports clock speed up to 30 MHz. It also supports variable length MOVX instruction for synchronising the data communication with slow RAM/peripheral devices. It also contains 2Kbytes of FLASH bootloader containing the low level FLASH memory programming routines and default serial loader. The 89C51ED2 version is similar to the RD2 version in all respect and it supports 2Kbytes of On-Chip EEPROM memory for non-volatile data storage.

5.5.2 Maxim/Dallas' *DS80C320/ DS80C323*

The *DS80C320/DS80C323* microcontrollers from Maxim/Dallas are high speed low power microcontrollers compatible with the standard *80C31/80C32* architecture. It contains four 8bit I/O ports, three 16bit timers/counters, 256 bytes of user data RAM, 13 total interrupt sources with six external, with 3 levels of priority, programmable watch dog timer (WDT), integrated power monitor for Power On reset and power supply fail detect and dual Data Pointer Register (DPTR). The high speed core of *DS80C320/ DS80C323* uses 4 clocks or 1 clock per machine cycle design. Comparing this with the original 12 clock/machine cycle for the same operating frequency, the performance will be approximately 3 times for the 4 clock core and 12 times for the 1 clock core preserving the same power consumption.

Summary

- ✓ Feature set, speed of operation, code memory space, data memory space, development support, availability, power consumption, cost, etc. are the important factors that need to be considered in the selection of a microcontroller for an embedded design
- ✓ The basic *8051* architecture consist of an 8bit CPU with Boolean processing capability, oscillator driver unit, 4K bytes of on-chip program memory, 128 bytes of internal data memory, 128 bytes of special function register memory area, 32 general purpose I/O lines organised into four 8bit bi-directional ports, two 16bit timer units and a full duplex programmable UART for serial data transmission with configurable baudrates
- ✓ *8051* is built around the 'Harvard' processor architecture. The program and data memory of *8051* is logically separated and they physically reside separately. Separate address spaces are assigned for data memory and program memory. *8051*'s address bus is 16bit wide and it can address up to 64KB memory
- ✓ The Program Strobe Enable (PSEN) signal is activated during the external program memory fetching operation, whereas it is not activated if the program memory is internal to the microcontroller. The Program Counter (PC) Register supplies the address from which the program instruction to be fetched.
- ✓ The *8051* architecture implements 8 general purpose registers with names R0 to R7 and they are available in 4 different banks
- ✓ The lower 128 byte RAM of *8051* is organised into register banks, Bit addressable memory and Scratchpad RAM
- ✓ Accumulator, B register, Program Status Word (PSW), Stack pointer (SP), Data pointer (DPTR, combination of DPL and DPH), and Program Counter (PC) constitute the CPU registers of *8051*
- ✓ The instruction fetch operation consists of a number of machine cycles and one machine cycle consists of 12 clock periods for the standard *8051* architecture
- ✓ *8051* supports 4 bi-directional ports. The ports are named as Port 0, 1, 2 and 3. Each port contains 8 bidirectional port pins
- ✓ The 'Read Latch' operation for all port pins return the content of the corresponding pin's latch, whereas the 'Read Pin' operation returns the status of the port pin
- ✓ The basic *8051* and its ROMless counterpart *8031*AH supports five interrupt sources; namely two external interrupts, two timer interrupts and the serial interrupt. The serial interrupt is an ORed combination of the two serial interrupts; Receive Interrupt (RI) and Transmit Interrupt (TI). An interrupt vector is assigned to each interrupts in the program memory and 8 bytes of code memory location is allocated for writing the ISR corresponding to each interrupt

✓ The interrupt system of *8051* can be enabled or disabled totally under software control by setting or clearing the global interrupt enable bit of the Special Function Register Interrupt Enable (IE).

✓ The *8051* architecture supports two levels of interrupt priority. The first priority level is determined by the settings of the Interrupt Priority (IP) register. The second level is determined by the internal hardware polling sequence

✓ An interrupt service routine always ends with the instruction RETI. The RETI instruction informs the interrupt system that the service routine for the corresponding interrupt is finished and it clears the corresponding priority-X interrupt in progress flag by clearing the corresponding flip flop

✓ The basic *8051* architecture supports two timer units namely Timer 0 and Timer 1. The timer units can be configured to operate as either timer or counter. The timers support four modes of operation namely Mode 0, 1, 2 and 3

✓ The standard *8051* supports a full duplex, receive buffered serial interface for serial data transmission

✓ The CMOS version of *8051* supports two power down modes, namely, IDLE and POWERDOWN. They are activated by setting the corresponding bit in Special Function Register PCON

✓ The on circuit emulation (ONCE) mode of *8051* helps in testing and debugging the *8051* microcontroller based system without removing the microcontroller from the circuit

✓ The *8052* microcontroller architecture supports an additional 16bit timer and it supports capture mode for timer operation in addition to the four modes supported by *8051*. It also implements the upper 128bytes of user data RAM physically on the chip

Keywords

MIPS	:	Million instructions per second–A measure of processor speed
Microcontroller	:	A highly integrated chip that contains a CPU, scratchpad RAM, Special and General purpose register Arrays and Integrated peripherals
Code Memory	:	Memory for storing program instructions
Data Memory	:	Memory for holding temporary data during program execution
Register	:	Data holding unit made up of flip-flops
On-Chip Memory	:	Memory internal to the microcontroller
Port	:	An I/O unit which groups a number of I/O pins together and provides a control/status register for controlling the I/O pins and monitoring the pin status
ALE	:	Address latch enable. A control signal generated by the processor/controller for indicating the arrival of valid address signal on the address/data multiplexed bus
Program Counter	:	CPU register which holds the address of the program memory location from which the next instruction is to be fetched (Its width depends on the processor architecture)
Data Pointer Register (DPTR)	:	16bit register which holds the address of external data memory address to be accessed, in 8051 architecture
Special Function Register	:	Register holding the status and control information and data associated with the various configurations, status and data for on-chip peripheral units like Timer, Interrupt Controller, etc.
Accumulator	:	CPU register which holds the results of all CPU related arithmetic operations
B Register	:	CPU register that acts as an operand in multiply and division operations, in *8051* architecture
Program Status Word (PSW)	:	8bit, bit addressable special function register signalling the status of accumulator related operations and register bank selector for the scratchpad registers R0 to R7, in *8051* architecture
Stack Pointer (SP)	:	8bit register holding the current address of stack memory in *8051* architecture

Machine Cycle	: The fundamental unit of instruction execution. One instruction execution may require one or more machine cycles. Under standard *8051* architecture, one machine cycle corresponds to 12 clock periods
Source Current	: The maximum current a port pin can supply to drive an externally connected device
Sink Current	: Themaximumcurrentaportpincanabsorbthroughadevicewhichisconnectedtoanexternal supply
Interrupt	: Signal that initiates changes in normal program execution flow
Interrupt Service Routine (ISR)	: Piece of code representing the actions to be done when an interrupt occurs
Interrupt Vector	: The start address of the program memory where the ISR corresponding to an interrupt is to be located
Interrupt Latency	: The time elapsed between the assertion of the interrupt and the start of the ISR for the same
Timer	: A hardware or software unit which generates time delays. Hardware timers generate more precise time delays
Serial Port	: The I/O unit which transmits and receives data in serial format
Multiprocessor Communication	: A serial communication implementation for communicating between multiple processors on the same serial bus. Only one device acts as the master and rest act as slave at any given point of time
High-Speed Core	: A processor core which requires lesser number of clock periods for instruction fetch, decode and execution

Objective Questions

1. What is the size of internal data memory supported by the standard *8051* architecture
   (a) 64 bytes (b) 128 bytes (c) 256 bytes (d) 1024 bytes
   (e) No internal data memory
2. What is the size of a 'Special Function Register' memory supported by the standard *8051* architecture
   (a) 64 bytes (b) 128 bytes (c) 256 bytes (d) 1024 bytes
   (e) No internal SFR memory
3. What is the size of an internal program memory supported by the standard *8051* architecture
   (a) 128 bytes (b) 1024 bytes (c) 2 Kbytes (d) 4 Kbytes
   (e) No internal program memory
4. What is the number of general purpose I/O lines supported by the standard *8051* architecture
   (a) 8 (b) 16 (c) 32 (d) 64
5. The general purpose I/O lines of a standard *8051* is grouped into
   (a) Two 8-bit bi-directional ports (b) Four 8-bit bi-directional ports
   (c) Two 16-bit bi-directional ports (d) Four 16-bit bi-directional ports
6. What is the number of timer units supported by a standard *8051* architecture
   (a) Two 16-bit timers (b) Three 16-bit timers (c) Two 8-bit timers (d) One 16-bit timer
7. The UART of the standard *8051* controller is
   (a) Half duplex with configurable baudrate (b) Half duplex with fixed baudrate
   (c) Full duplex with configurable baudrate (d) Full duplex with fixed baudrate
8. The standard 8051 controller is built around
   (a) Harvard Architecture (b) Von Neumann Architecture (c) None of these
9. Which of the following is *True* for a *8051* controller?
   (a) The program and data memory of *8051* is logically separated

(b) The program and data memory of *8051* physically resides separately.
(c) Separate address spaces are assigned for data memory and program memory
(d) All of these (e) None of these

10. The address bus of *8051* is
(a) 8-bit wide (b) 16-bit wide (c) 32-bit wide (d) 20-bit wide

11. Which of the following is *True* about external program memory access?
(a) Port 0 acts as the data bus and Port 2 acts as the higher order address bus
(b) Port 2 acts as the data bus and Port 0 acts as the higher order address bus
(c) Port 0 acts as the multiplexed address/data bus and Port 2 acts as the higher order address bus
(d) Port 2 acts as the multiplexed address/data bus and Port 0 acts as the higher order address bus

12. Name the register holding the address of the memory location holding the next instruction to fetch
(a) DPTR (b) PC (c) SP (d) None of these

13. Name the register holding the address of the external data memory to be accessed in 16bit external data memory operation
(a) DPTR (b) PC (c) SP (d) None of these

14. For standard *8051* architecture, the internal data memory address is
(a) 8bit wide (b) 4bit wide (c) 16bit wide (d) None of these

15. The external data memory is 4 Kbytes in size and it is arranged in paged addressing mode where 1 page consists of 256 bytes. Port 2 is used as the page selector. How many port bits are required for implementing the paging?
(a) 1 (b) 2 (c) 3 (d) 4

16. Which of the following conditions should be satisfied to implement the Von-Neumann memory model for *8051*?
(a) The data memory and code memory should be stored in a single memory chip (RAM/NVRAM)
(b) The external access pin should be tied to logic 0
(c) The external access pin should be tied to logic 1
(d) The PSEN\ and RD\ signals of 8051 should be ANDed to generate the output enable (OE\) of the memory chip
(e) (a) (b) and (d) (f) (a) (c) and (d)

17. What is the number of general purpose registers supported by the standard *8051* architecture
(a) 1 (b) 8 (c) 16 (d) 32

18. What is the number of bit variables supported by *8051* in the internal data memory area?
(a) 8 (b) 16 (c) 32 (d) 64
(e) 128

19. Bit address 07H contains logic 1, what is the value of bit 07H after executing the instruction MOV 20H, #01H
(a) 0 (b) 1

20. Memory location 81H contains F0H. What will be the value of Accumulator after executing the instruction *MOV A, 81H*
(a) 81H (b) 00H (c) F0H (d) Random data

21. The target controller is a standard *8051*. Memory location 81H contains F0H. What will be the value of accumulator after executing the instructions
MOV R0, #81H
MOV A, @R0
(a) 81H (b) 00H (c) F0H (e) Random data

22. Name the register signalling the status of accumulator related operations
(a) A (b) B (c) PSW (d) SP
(e) None of these

23. How many user programmable general purpose bits are available in the status register of *8051*?
(a) 0 (b) 1 (c) 2 (d) 3
(e) 4

24. The accumulator content is 02H. What will be the status of the 'parity bit P' of the Status Register?
(a) 1 (b) 0

25. The accumulator contains 0FH. The overflow (OV) carry (C) flag and Auxiliary carry flag (AC) are in the set state. What will be status of these flags after executing the instruction *ADD A, #1*
 (a) OV = 0; C = 0; AC = 0 (b) OV = 0; C = 0; AC = 1
 (c) OV = 1; C = 1; AC = 1 (d) OV = 1; C = 1; AC = 0
26. The register bank selector bits RS0, RS1 are 0 and 1. What is the physical address of register R0?
 (a) 00H (b) 08H (c) 0FH (d) 10H
 (e) 18H
27. The machine cycle for a standard *8051* controller consists of
 (a) 2T States (b) 4T States (c) 6T States (d) 8T States
28. Which port of *8051* is 'true-bidirectional'?
 (a) Port 0 (b) Port 1 (c) Port 2 (d) Port 3
29. For configuring a port pin as input port, the corresponding port pins bit latch should be at
 (a) Logic 0 (b) Logic 1
30. Port 2 latch contains A5H. What will be the value of Port 2 latch after executing the following instructions?

```
MOV DPTR, #0F00H
MOV A, #0FFH
MOVX @DPTR, A
```

 (a) 00H (b) 0FH (c) A5H (d) FFH
31. The alternate I/O function for the pins of Port 3 will come into action only when the corresponding bit latch is
 (a) 1 (b) 0
32. The interrupts Timer 0 and serial interrupt are enabled individually in the interrupt enable register and high priority is given to Timer 0 interrupt by setting the Timer 0 priority selector in the interrupt priority register. It is observed that the serial interrupt is not at all occurring. What could be the reasons for this?
 (a) The global interrupt enable bit (EA) is not in the set state
 (b) The Serial interrupt always occurs with Timer 0 interrupt
 (c) There is no Serial data transmission or reception activity happening in the system
 (d) None of these (e) (a) or (b) or (c)
33. Timer 0 and External 0 interrupts are enabled in the system and are given a priority of 1. Incidentally, the Timer 0 interrupt and External 0 interrupt occurred simultaneously. Which interrupt will be serviced by the *8051* CPU?
 (a) Timer 0 (b) External 0
 (c) External 0 interrupt is serviced first and after completing it Timer 0 interrupt is serviced
 (d) None of them are serviced
34. What is the maximum ISR size allocated for each interrupt in the standard 8051 Architecture
 (a) 1 Byte (b) 4 Byte (c) 8 Byte (d) 16 Byte
35. What is the minimum interrupt acknowledgement latency in a single interrupt system for standard *8051* architecture
 (a) 1 Machine cycle (b) 2 Machine cycle (c) 3 Machine cycle (d) 8 Machine cycle
36. External 0 interrupt is asserted and latched at S5P2 of the first machine cycle of the instruction *MUL AB*. What will be the minimum interrupt acknowledgement latency time?
 (a) 6 Machine cycles (b) 5 Machine cycles (c) 3 Machine cycles (d) 2 Machine cycles
37. What is the minimum duration in which the external interrupt line should be asserted to identify it as a valid interrupt for level triggered configuration?
 (a) 1 Machine cycle (b) 3 T States (c) 2 Machine cycles (d) 3 Machine cycles
38. What is the timer increment rate for timer operation for standard *8051* architecture
 (a) Oscillator Frequency/6 (b) Oscillator Frequency/12
 (c) Same as Oscillator Frequency (d) Oscillator Frequency/24
39. What is the maximum count rate for counting external events for standard *8051* architecture
 (a) Oscillator Frequency/6 (b) Oscillator Frequency/12
 (c) Same as Oscillator Frequency (d) Oscillator Frequency/24
40. Which is the 'Timer' used for baudrate generation for serial communication?
 (a) Timer 0 (b) Timer 1

41. The 'Timer' used for baudrate generation should run in
    (a) Mode 0 (b) Mode 1 (c) Mode 2 (d) Mode 3
42. For 'Mode 0' operation, the 13 bit register is formed by
    (a) 8 bits of TH0/TH1 and least significant 5 bits of TL0/TL1
    (b) 8 bits of TH0/TH1 and most significant 5 bits of TL0/TL1
    (c) 8 bits of TL0/TL1 and least significant 5 bits of TH0/TH1
    (d) 8 bits of TL0/TL1 and most significant 5 bits of TH0/TH1
43. The serial port of the standard *8051* architecture is
    (a) Full duplex (b) Half duplex (c) 'Receive' buffered (d) (a) and (c)
    (e) (b) and (c)
44. Name the 'Register' which acts as the 'Receive' and 'Transmit' buffer in serial communication operation?
    (a) SCON (b) PCON (c) SBUF (d) Accumulator
45. Which of the following is (are) true about 'Mode 0' operation of serial communication
    (a) The baudrate is variable (b) The baudrate is given as oscillator frequency/12
    (c) The baudrate is same as oscillator frequency (d) The baudrate is given as oscillator frequency/2
46. The 'Auto Reload' count for 'Timer 1' is FDH and the operating frequency is 11.0592 MHz and baudrate doubler bit SMOD is 0. What is the baudrate for communication?
    (a) 2400 (b) 4800 (c) 9600 (d) 19200
47. What will be the value of 'Program Counter (PC)' after a proper power on reset?
    (a) FFFFH (b) 0000H (c) Random value (d) 0001H
48. What will be the value of internal RAM after a reset?
    (a) 00H (b) FFH
    (c) The value before reset if the system is resetted during operation
    (d) Random if the system is resetted immediately after Power ON
    (e) (c) or (d)
49. Which of the following is (are) 'True' about 'IDLE' mode?
    (a) The internal clock to the processor is temporarily suspended
    (b) The various CPU status like SP, PC, PSW, Accumulator and all other register values are preserved
    (c) All port pins will retain their logical state
    (d) ALE and PSEN are pulled high
    (e) All of these (f) (a), (b) and (c) only
50. A reset signal is applied to the *8051* processor when it is in the 'Idle' mode. How will the system behave?
    (a) The idle mode setting bit IDL is cleared
    (b) The processor resumes program execution from where it left off
    (c) The processor resumes program execution from 0000H
    (d) (a) and (b) only (e) (a) and (c) only

Review Questions

1. Explain the various factors to be considered while selecting a microcontroller for an embedded system design.
2. Explain the architecture of the *8051* microcontroller with a block diagram.
3. Explain the different operating modes of *8051* (Hint: normal operation mode, power saving mode and ONCE mode)
4. Explain the code memory organisation for *8051* for internal and external program memory access.
5. Explain the data memory organisation for standard *8051* controller.
6. Explain the *Von-Neumann* memory model implementation for *8051* based system. What are the merits and demerits of using a *Von-Neumann* memory model?
7. Explain the memory organisation for lower 128 bytes of internal RAM for standard *8051* architecture.

8. Explain how Port 0 acts as a normal I/O port and multiplexed address data bus for external data/program memory access?
9. What is the difference between *Read Latch* and *Read Pin* operation for port lines?
10. Why is Port 0 known as *true-bidirectional*?
11. Why is Port 1 known as *quasi-bidirectional*?
12. Explain how Port 2 acts as a normal I/O port and higher order address bus for external data/program memory access?
13. Explain how Port 3 acts as a normal I/O port and an alternate I/O function port?
14. What is the difference between RET and RETI instructions for indicating the end of a subroutine call?
15. Explain how a third priority level can be achieved in *8051* interrupt system?
16. What is *interrupt latency*? What is the minimum and maximum interrupt latency time for a single interrupt system in standard *8051* architecture? Explain in detail.
17. Explain the different conditions that block or delay the vectoring of an interrupt.
18. Explain the different actions generated by the processor on identifying an interrupt request number.
19. The external interrupt INT0 is enabled, and set to be level triggered. The Port pin P3.2 is set at logic 0. Explain the behaviour of the system.
20. Explain the operation of Timer 0 in *Mode 0*. Why is *Mode 0* operation known as 8-bit Timer with a divide by 32 prescaler?
21. Explain the *auto reload* mode of operation of a timer. Give an example for the usage of Timer 1 in *auto reload* mode.
22. Explain the *Mode 1* operation of *serial communication*. How is *configurable baudrate* achieved for serial communication in *Mode 1*? What is the *Timer 1* auto reload count for a baudrate of 9600 bits/second for a system working at 11.0592MHz?
23. What is *multiprocessor communication*? Explain the multiprocessor communication for *8051* for serial communication *Mode 2*.
24. How a 'power on reset' can be implemented for an *8051*-based system? Explain the changes that happen to various registers and internal RAM on a proper reset.
25. Explain the two power saving modes *Idle* and *Power Down* modes for *8051*. What is the difference between the two? Explain the methods of terminating each of the power saving modes.
26. Explain the *Power consumption* v/s *Operating Frequency* characteristics for an *8051* based system. Why do the characteristics curve contain an offset from the origin?
27. What are the different techniques that can be adopted for reducing the power consumption of an *8051* based system?
28. What is *On Circuit Emulation (ONCE)* mode? How is *ONCE* mode enabled? What is the use *ONCE* mode in system design?
29. Explain the difference between *8051* and *8052* microcontroller.
30. Only Timer 0 interrupt is enabled in the system and it is assigned a priority of 1. It is observed that the timer interrupt occurred only once and it is not occurring even though the timer roll over happens. The program does not contain any instruction for disabling the global interrupt enable flag IE and Timer 0 interrupt. What could be the reason?

Lab Assignments

1. Write an *8051* assembly language program for transmitting 1 byte data through the serial port with a ninth bit which is used as an even parity bit. The program should configure the serial communication settings as: Baudrate = 9600, 1 start bit, 1 stop bit and also depending on the parity of the data byte to be sent, the parity bit should be set properly.

2. Write an *8051* assembly language program for receiving an even parity enabled 8-bit data through the serial port. The program should configure the serial communication settings as: Baudrate = 9600, 1 start bit, 1 stop bit. Upon receiving the data byte, the parity of it is calculated and it is verified with the parity bit received. Use interrupts for implementing the serial data reception
3. Develop a hardware system to single step the program written for *8051*. The firmware running in the controller sends the register contents A, B, PSW and R0 to R7 on executing each instruction to a program running on PC. Develop a PC application to capture the register details and display it. Use RS-232 UART communication for sending the debug info to PC. Use the following configuration settings for the RS-232 link: Baudrate = 9600, 1 start bit, 1 stop bit, No parity.
4. Develop an *8051*-based system for detecting the keypress of a pushbutton switch connected to the port pin P1.0 of the microcontroller. Implement the key de-bouncing for the push button in firmware (Upon detecting a keypress, the firmware waits for 20 milliseconds and then checks the push button switch again, if the switch remains in the depressed state, the key press is identified as a valid key depression). For a valid keypress, a buzzer connected to port pin P1.1 is activated for a period of 1 second. Use a BC547 transistor-based driver circuit for driving the buzzer.
5. Implement the above requirement using a hardware key de-bounce circuit and connecting the push button switch to the external interrupt 0 pin of the controller. Implement the buzzer control logic in the Interrupt Service Routine for External Interrupt 0.
6. Develop an *8051*-based system for detecting the keypress of an array of 8 pushbutton switches connected to the port pins P1.0 to P1.7 of the microcontroller. Using an AND gate generate an external interrupt signal when any one of the push button switch is depressed. Identify which push button is depressed by scanning the status of the push button connected port pins, in the interrupt service routine for the external interrupt. The port pins are scanned in the order P1.0 to P1.7. Even if multiple push buttons are depressed, the first identified push button switch is recognised as the valid keypress. Use a hardware key de-bounce circuit/IC for implementing the key de-bouncing for all push buttons. An 8 ohm speaker is connected to port pin P2.0. Generate different tones (by varying the frequency (say 100Hz, 200Hz, 300Hz, 500Hz, etc) of the pulse generated at the port pin in which the speaker is connected, corresponding to each push button press. Use BC547 transistor based driver circuit for driving the speaker.
7. Design an *8051* based system for interfacing an RF transceiver ***(e.g. RF Modem from SUNROM Technologies http://www.sunrom.com/index.php?main_page=product_info&cPath=89&products_id=560)*** and a relay driver circuit using an NPN transistor for driving a normally open relay with coil voltage 12V for switching Alternating Current with ratings 5Amps and 50Hz. Connect a 100W/240V bulb through the relay. Turn ON and OFF the bulb in response to the command received by the RF transceiver. The commands are sent by an application running on PC through an RF transceiver.
8. Implement the above system using infrared (IR) remote control and receiver, supporting the RC5 protocol (e.g. TV remote and SFH 506-36 IR receiver unit (Use the reference design http://www.atmel.com/dyn/resources/prod_documents/doc1473.pdf from Atmel for SFH 506 interfacing)), in place of the RF based control implementation.
9. Develop a miniature 'Traffic Light Control' system based on *8051* and LEDs for controlling the 'Red' 'Green' and 'Yellow' signal lights present at a junction where four roads meet. The 'Green' and 'Red' signal will be on for a duration of 30 seconds and 'Yellow' for 5 seconds.
10. Develop a simple electronic calculator using 8051 microcontroller with 16 push button keys arranged in a 4×4 matrix for representing digits 0 to 9, operators +, –, ×, %, = and 'Clear' button. Use two 7-segment LEDs for displaying the result. Only single digit operations are allowed. The system requirements are listed below.
    (a) Upon power on both the 7-Segment LEDs display 0
    (b) When a numeric key is pressed, it is displayed on the rightmost 7-segment LED
    (c) Enter the first operand (digit) by pressing the corresponding push button
    (d) To perform an operation (+, –, ×, %) press the corresponding push button
    (e) Now press the second operand (digit)
    (f) Press the '=' operator for displaying the result of the operation
    (g) Pressing the 'Clear' key at any point clears the display and displays '0' on both LED displays. For division operation, display the quotient as result.