MAR GREGORIOS COLLEGE OF ARTS & SCIENCE

Block No.8, College Road, Mogappair West, Chennai – 37

Affiliated to the University of Madras
Approved by the Government of Tamil Nadu
An ISO 9001:2015 Certified Institution

DEPARTMENT OF ELECTRONICS & COMMUNICATION SCIENCE

SUBJECT NAME: MICRO CONTROLLER

SUBJET CODE: TAG6A

SEMESTER: VI

PREPARED BY: PROF.M.SATHIYA

UNIT I8051 ARCHITECTURE

Introduction to Microcontroller – Comparison of Microcontroller & Microprocessor – 8051 Microcontroller – Block diagram – I/O pins, ports, and circuits – External memory – Counter and timers' serial data I/O – Interrupts.

UNIT II 8051 INSTRUCTION SET

Addressing modes – Logical operation: Byte level – Bit level – Rotate and Swap operation.

ARITHEMTIC OPERATIONS

Instruction affecting flags – Incrementing and Decrementing – Addition – Subtraction – Multiplication and Division – Example Program.

UNIT III JUMP and CALL INSTRUCTION

Introduction – The Jump and Call program Range – Jumps: Bit – Byte Unconditional: Calls and Subroutine – Interrupts and Returns – Example program.

UNIT IV INTERFACING

Keyboard – Displays – Stepper motor – ADC & DAC.

UNIT V INTRODUCTION TO MICROCONTROLLERS

6509 – PIC Controllers – 6525 series – Introduction to Embedded Systems.

REFERENCES:

1. Kenneth J. Ayala. "The 8051 Microcontroller, Architecture, Program and Applications", Pen ram International.
2. Muhammed Ali Mazidi, Janice GillispieMazidi. "The 8051 Microcontroller and Embedded Systems", Low Price Edition.
3. Microcontrollers: Theo & App by Ajay V. Deshmuk Tata McGraw-Hill Education 2005.

CONTENT

UNIT I

8051 ARCHITECTURE

1.0 Introduction

1.1 Comparison of Microcontroller & Microprocessor

1.2 8051 Microcontroller

1.2.1 Features

1.3 Block diagram

1.3.1 Oscillator and clock generator

1.3.2 ALU

1.3.3Serial data register

1.3.4 Interrupt register

1.3.5 Power control register

1.3.6 Memory (RAM and ROM)

1.4 I/O pins, ports, and circuits

1.4.1 Pin configuration

1.4.2 Ports

1.4.3 Internal memory

1.4.4 Internal data memory and Special function register (SFR) map

1.5 External memory

1.6 Counter and timer's serial data I/O

1.6.1 TMOD&TCON

1.6.2 Serial Communication

1.6.3 Types of serial communication

1.6.4 Baud rate

1.6.5 Serial communication registers

1.6.6 Serial communication modes

1.7 Interrupts

1.7.1 Classification of interrupts

1.7.2 Interrupt structure

1.7.3 Interrupt registers

UNIT II

8051 INSTRUCTION SET

2.0 Introduction

2.1 Addressing modes

- 2.1.1 Immediate addressing mode
- 2.1.2 Register addressing mode
- 2.1.3 Direct addressing mode
- 2.1.4 Register indirect addressing mode
- 2.1.5 Indexed addressing mode

2.2 Notation descriptions

2.3 Logical operations

2.4 Boolean Instructions

2.5 Rotate and Swap instruction

2.6 Data transfer instructions

2.7 Arithmetic Operations

UNIT III

JUMP and CALL INSTRUCTION

3.0 Introduction

3.1 Jump instruction

3.2 Jump: Byte Unconditional

3.3 Calls and Subroutine

3.4 Interrupts and Returns

3.5 NOP operation

UNIT IV

INTERFACING

4.0 Introduction

4.1 LED interfacing

- 4.1.1 Components
- 4.1.2 Circuit diagram
- 4.1.3 Connection

4.1.4 Source code
4.1.5 Applications
4.2 7-Segment Display
4.2.1 Circuit diagram
4.2.2 Source code
4.2.3 Applications
4.3 LCD interfacing
4.3.1 Circuit diagram
4.3.2 Commands
4.3.3 Source code
4.3.4 Applications
4.4 Matric Keypad interface
4.4.1 Circuit diagram
4.4.2 Description
4.4.3 Source code
4.4.4 Applications
4.5 Stepper motor
4.5.1 Circuit diagram
4.5.2 Components and description
4.5.3 Description
4.5.4 Source code
4.5.5Applications
4.6 ADC interfacing
4.6.1 Components
4.6.2Circuit diagram
4.6.3 Description
4.6.4 Source code
4.6.5 Applications
4.7 DAC interfacing
4.7.1 Components
4.7.2 Circuit diagram

4.7.3 Source code

4.7.4 Applications

UNIT V

INTRODUCTION TO MICROCONTROLLERS

5.0 Introduction

5.1 NI 6509

5.1.1 PCI Interface

5.1.2 General Operation Registers

5.1.3 Recurring Port Registers

5.1.4 Non-Recurring Port Registers

5.1.5 Watchdog Timer Registers

5.1.6 RTSI Configuration registers

5.1.7 Applications

5.2 PIC Controllers

5.3 Architecture

5.3.1 Memory structure

5.3.2 Status register

5.3.3 File Selection Register

5.3.4 EEPROM

5.3.5 I/O Ports

5.3.6 Timers

5.3.7 A/D Converters

5.3.8 Oscillators

5.3.9 CCP module

5.3.10 Advantages

5.46525 series

5.4.1 Peripheral Features

5.4.2 Pin configuration

5.4.3 Advantages

5.5Introduction to Embedded Systems

5.5.1 Basic structure

5.5.2 Advantages

5.5.3 Disadvantages

5.5.4 Applications

5.5.5 Future trends

UNIT I

8051 ARCHITECTURE

1.0INTRODUCTION

A microcontroller is a computer with most of the necessary support chips onboard. All computers have several things in common, namely

- ✓ A central processing unit (CPU) that 'executes' programs.
- ✓ Some random-access memory (RAM) where it can store data that is variable.
- ✓ Some read only memory (ROM) where programs to be executed can be stored.
- ✓ Input and output (I/O) devices that enable communication to be established with the outside world i.e. connection to devices such as keyboard, mouse, monitors and other peripherals.
- ✓ There are several other common characteristics that define microcontrollers. If a computer matches most of these characteristics, then it can be classified as a 'microcontroller'. Microcontrollers may be:
- ✓ 'Embedded' inside some other device (often a consumer product) so that they can control the features or actions of the product. Another name for a microcontroller is therefore an 'embedded controller'.
- ✓ Dedicated to one task and run one specific program. The program is stored in ROM and generally does not change.
- ✓ A low-power device. A battery-operated microcontroller might consume as little as 50 milliwatts.

A microcontroller may take an input from the device it is controlling and controls the device by sending signals to different components in the device. A microcontroller is often small and low cost. The components may be chosen to minimize size and to be as inexpensive as possible. The actual processor used to implement a microcontroller can vary widely. In many products, such as

microwave ovens, the demand on the CPU is low and price is an important consideration. In these cases, manufacturers turn to dedicated microcontroller chips – devices that were originally designed to be low-cost, small, low-power, embedded CPUs. The Motorola 6811 and Intel 8051 are both good examples of such chips. A typical low-end microcontroller chip might have 1000 bytes of ROM and 20 bytes of RAM on the chip, along with eight I/O pins. In large quantities, the cost of these chips can sometimes be just a few pence. In this book the authors will introduce the reader to some of the Philips' 8051 family of microcontrollers, and show their working, with applications, throughout the book. The programming of these devices is the same and, depending on type of device chosen, functionality of each device is determined by the hardware devices onboard the chosen device.

1.1 COMPARISON OF MICROCONTROLLER & MICROPROCESSOR

Microprocessor	**Microcontroller**
A microprocessor requires an external memory for program/data storage	A microcontroller has required on-chip memory with associated peripherals
Brain of the computer	Computer on chip
Instruction execution requires movement of data from the external memory to the microprocessor or vice versa	A microcontroller does not require much additional interfacing ICs for operation and it functions as a standalone system
General purpose digital computers	Special purpose digital controllers
Does not contain RAM, ROM, and I/O ports on chip	Contains a microprocessor, RAM, ROM, I/O ports, and timer on a single chip
Instruction execution requires movement of data from the external memory to the microprocessor or vice versa. Slow response	Fast response
E.g., 8086, Core 2 Duo, …	E.g., 8051, PIC, AVR, …

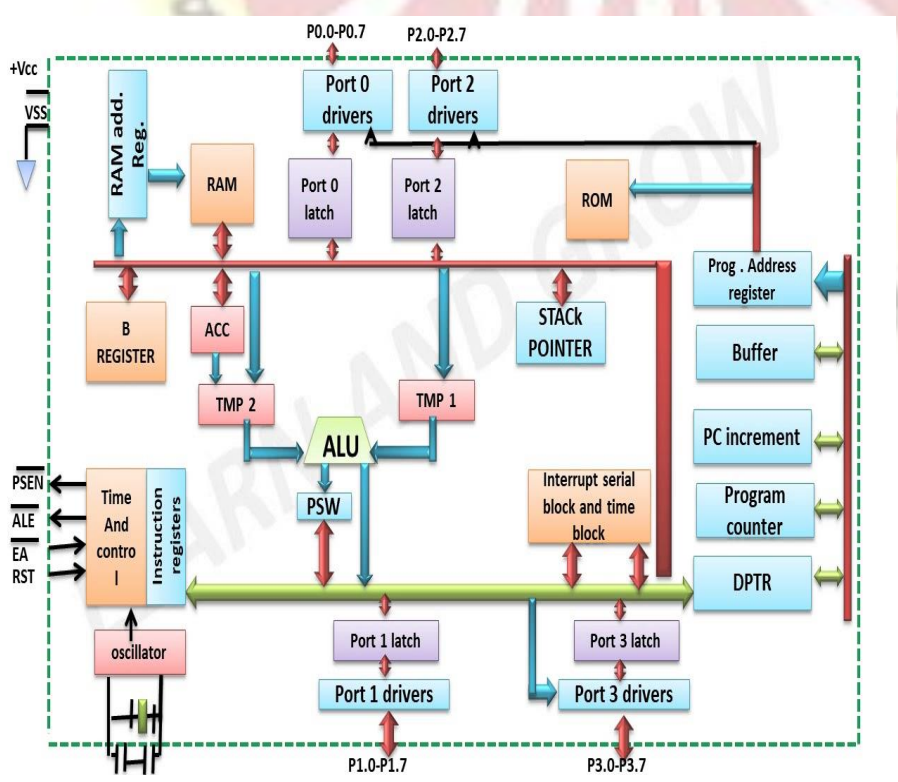
1.2 8051 MICROCONTROLLERS

Various features of 8051 microcontroller are given as follows:

- ✓ 16-bit Program Counter
- ✓ 8-bit Processor Status Word (PSW)
- ✓ 8-bit Stack Pointer
- ✓ Internal RAM of 128bytes
- ✓ Special Function Registers (SFRs) of 128 bytes
- ✓ 32 I/O pins arranged as four 8-bit ports (P0 - P3)
- ✓ Two 16-bit timer/counters: T0 and T1
- ✓ Two external and three internal vectored interrupts
- ✓ One full duplex serial I/O

1.3 BLOCK DIAGRAM

It is 8-bit microcontroller, means MC 8051 can Read, Write and Process 8-bit data. This is mostly used microcontroller in the robotics, home appliances like mp3 player, washing machines, electronic iron, and industries. Mostly used blocks in the architecture of 8051 are as follows:



1.3.1 Oscillator and clock generator

All operations in a microcontroller are synchronized by the help of an oscillator clock. The oscillator clock generates the clock pulses by which all internal operations are synchronized. A resonant network connected through pins XTAL1 and XTAL2 forms up an oscillator. For this purpose, a quartz crystal and capacitors are employed. The crystal run at specified maximum and minimum frequencies typically at 1 MHz to 16 MHz.

1.3.2 ALU

It is 8-bit unit. It performs arithmetic operation as addition, subtraction, multiplication, division, increment and decrement. It performs logical operations like AND, OR and EX-OR. It manipulates 8-bit and 16-bit data. It calculates address of jump locations in relative branch instruction. It performs compare, rotate, and compliment operations. It consists of Boolean processor which performs bit, set, test, clear and compliment. 8051 micro controller contains 34 general purpose registers or working registers.2 of them are called math registers A & B and 32 are bank of registers.

- ✓ **Accumulator(A-reg):** It is 8-bit register. Its address is E0H and it is bit and byte accessible. Result of arithmetic & logic operations performed by ALU is accumulated by this register. Therefore, it is called accumulator register. It is used to store 8-bit data and to hold one of operand of ALU units during arithmetical and logical operations. Most of the instructions are carried out on accumulator data. It is most versatile of 2 CPU registers.
- ✓ **B-register:** It is special 8-bit math register. It is bit and byte accessible. It is used in conjunction with A register as I/P operand for ALU. It is used as general-purpose register to store 8-bit data.
- ✓ **PSW:** It is 8-bit register. Its address is D0H and It is bit and byte accessible. It has 4 conditional flags or math flags which sets or resets according to condition of result. It has 3 control flags, by setting or resetting bit required operation or function can be achieved. The format of flag register is as shown below:

  Carry flag (CY): During addition and subtraction any carry or borrow is generated then carry flag is set otherwise carry flag resets. It is used in arithmetic, logical, jump, rotate and Boolean operations.

Auxiliary carry flag (AC): If during addition and subtraction any carry or borrow is generated from lower 4 bit to higher 4 bit then AC sets else it resets. It is used in BCD arithmetic operations.

Overflow flag (OV): If in signed arithmetic operations result exceeds more than 7 bit than OV flag sets else resets. It is used in signed arithmetic operations only.

Parity flag(P): If in result, even no. Of ones "1" are present than it is called even parity and parity flag sets. In result odd no. Of ones "1"are present than it is called odd parity and parity flag resets.

FO: It is user defined flag. The user defines the function of this flag. The user can set, test n clears this flag through software.

RS1 and RS0: These flags are used to select bank of register by resetting those flags which are as shown intable:

- ✓ **Program counter (PC):** The Program Counter (PC) is a 2-byte address which tells the 8051 where the next instruction to execute is found in memory. It is used to hold 16-bit address of internal RAM, external RAM, or external ROM locations. When the 8051 is initialized PC always starts at 0000h and is incremented each time an instruction is executed. It is important to note that PC isn't always incremented by one and never decremented.
- ✓ **Data pointer register (DTPR):** It is a 16-bit register used to hold address of external or internal RAM where data is stored, or result is to be stored. It is used to store 16-bit data. It is divided into2- 8bit registers, DPH-data pointer higher order (83H) and DPL-data pointer lower order (82H). Each register can be used as general-purpose register to store 8-bit data and can also be used as memory location. DPTR does not have single internal address. It functions as Base register in base relative addressing mode and in-direct jump.
- ✓ **Stack pointer (SP):** It is 8-bit register. It is byte addressable. Its address is 81H. It is used to hold the internal RAM memory location addresses which are used as stack memory. When the data is to be placed on stack by push instruction, the content of stack pointer is incremented by 1, and when data is retrieved from stack, content of stack of stack pointer is decremented by 1
- ✓ **Special function Registers (SFR):**The 8051 microcontroller has 11 SFR divided in 4 groups:

Timer/Counter register: 8051 microcontroller has 2-16-bit Timer/counter registers called Timer-reg-T0 And Timer/counter Reg-T1.Each register is 16-bit register divide into lower and higher byte register as shown below: These register are used to hold initial no. of count. All of the 4 register are byte addressable. 1.

Timer control register: 8051 microcontroller has two 8-bit timer control register i.e. TMOD and TCON register. TMOD Register: it is 8-bit register. Its address is 89H. It is byte addressable. It used to select mode and control operation of time by writing control word.

TCON register: It is 8-bit register. Its address is 88H. It is byte addressable. Its MSB 4-bit are used to control operation of timer/ counter and LSB 4-bit are used for external interrupt control.

1.3.3 Serial data register

8051 micro controller has 2 serial data register viz. SBUF and SCON.

- ✓ **Serial buffer register (SBUF):** it is 8-bit register. It is byte addressable. Its address is 99H. It is used to hold data which is to be transferred serially.
- ✓ **Serial control register (SCON)**: it is 8-bit register. It is bit/byte addressable. Its address is 98H. The 8-bit loaded into this register controls the operation of serial communication.

1.3.4 Interrupt register

8051 µC has 2 8-bit interrupt register.

- ✓ **Interrupt enable register (IE):** It is 8-bit register. It is bit/byte addressable. Its address is A8H.it is used to enable and disable function of interrupt.
- ✓ **Interrupt priority register (IP):** It is 8-bit register. It is bit/byte addressable. Its address is B8H. it is used to select low- or high-level priority of each individual interrupts.

1.3.5 Power control register (PCON)

It is 8-bit register. It is byte addressable. Its address is 87H. Its bits are used to control mode of power saving circuit, either idle or power down mode and also one bit is used to modify baud rate of serial communication.

1.3.6 Memory

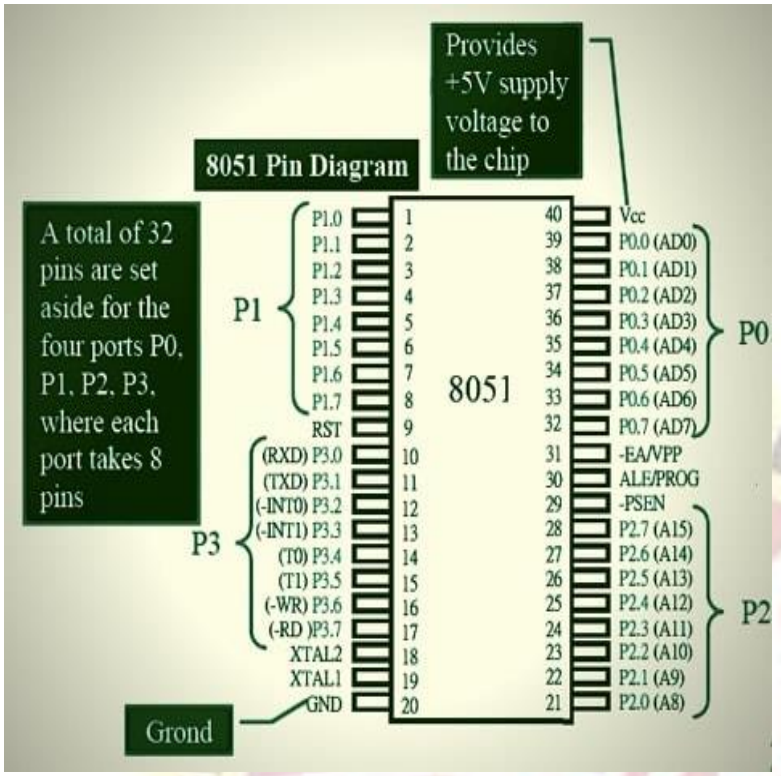
Internal RAM has memory 128-byteInternal RAM is organized into three distinct areas: 32 bytes working registers from address 00h to 1Fh 16 bytes bit addressable occupies RAM byte address 20h to 2Fh, altogether 128 addressable bits General purpose RAM from 30h to 7Fh.

Internal **ROM** data memory and program code memory both are in different physical memory, but both have the same addresses. An internal ROM occupied addresses from 0000h to 0FFFh. PC addresses program codes from 0000h to 0FFFh. Program addresses higher than 0FFFh that exceed the internal ROM capacity will cause 8051 architecture to fetch codes bytes from external program memory.

1.4 I/O PINS, PORTS AND CIRCUITS

- ✓ 40-pin dual in line package
- ✓ Ports (P0-P3)
  - 8-bit bidirectional bit addressable I/O port
  - Allotted address in special function register (SFR)
- ✓ Port 0 (P0.0-P0.7)
  - Access low 8-bit address lines from (A0-A7) when ALE-high and EA-low
- ✓ Port 1 (P1.0-P1.7)-only data lines
- ✓ Port 2 (P2.0-P2.7)
  - Access high 8-bit address lines from (A8-A15) when ALE-high and EA-low
- ✓ Port 3 (P3.0-P3.7)

1.4.1 Pinconfiguration of 8051

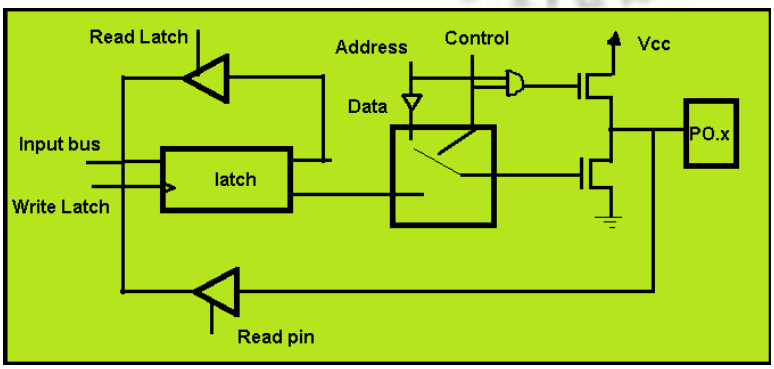


1.4.2 Ports:

Pins may serve as inputs, outputs, or, when used together, as a bidirectional low order address and data bus for external memory.

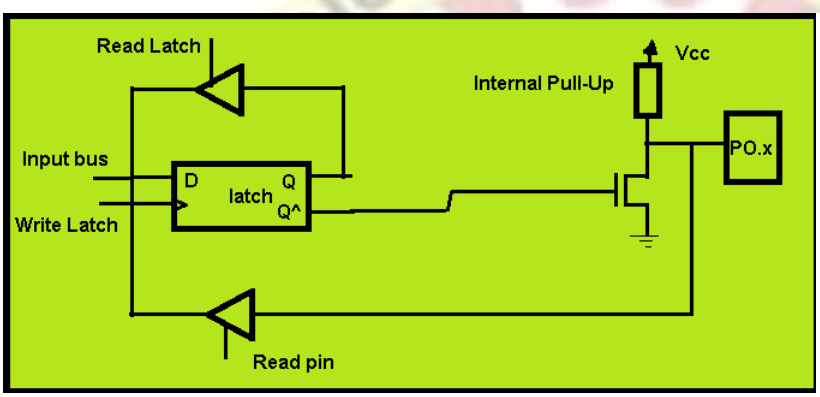
Port 0

Port-0 can be used as a normal bidirectional I/O port or it can be used for address/data interfacing for accessing external memory. When control is '1', the port is used for address/data interfacing. When the control is '0', the port can be used as a bidirectional I/O port.



Port1

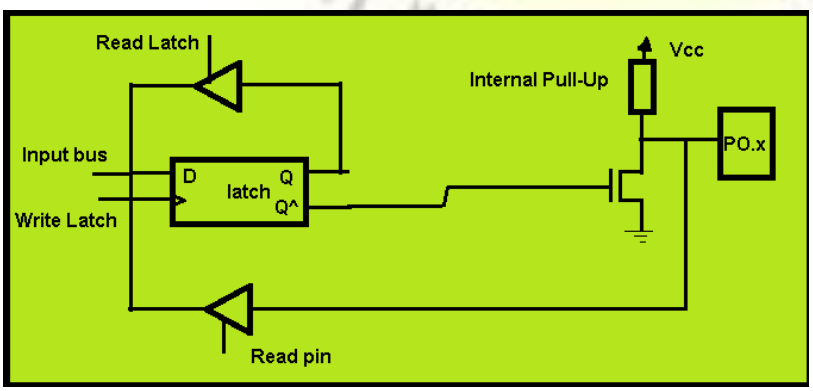
Port-1 dedicated only for I/O interfacing. When used as output port, not needed to connect additional pull-up resistor like port 0. It has provided internally pull-up resistor as shown in fig. below. The pin is pulled up or down through internal pull-up when we want to initialize as an output port. To use port-1 as input port, '1' must be written to the latch. In this input mode when '1' is written to the pin by the external device then it read fine. But when '0' is written to the pin by the external device then the external source must sink current due to internal pull-up. If the external device is not able to sink the current the pin voltage may rise, leading to a possible wrong reading.



Port2

We use for higher external address byte or a normal input/output port. The I/O operation is like Port-1. Port-2 latch remains stable when Port-2 pin are used for external memory access.

Here again due to internal pull-up there is limited current driving capability.

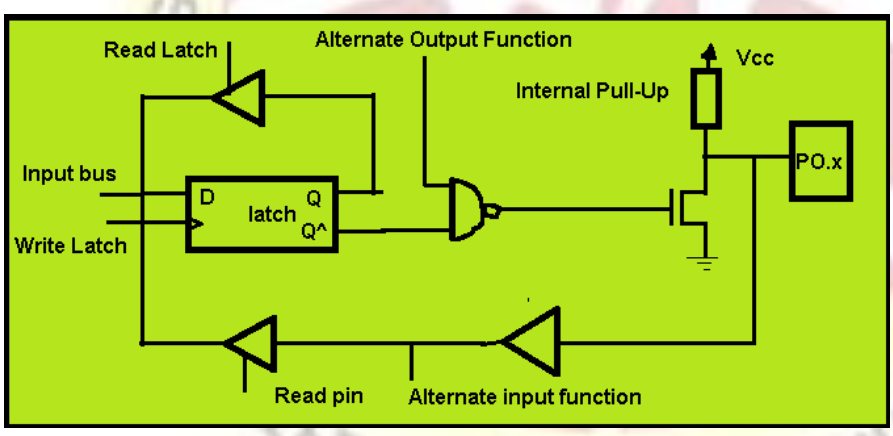


Port 3

Following are the alternate functions of port 3:

- ✓ P3.0—RXD
- ✓ P3.1— TXD
- ✓ P3.2— INT0 BAR
- ✓ P3.3— INT1 BAR
- ✓ P3.4— T0
- ✓ P3.5— T1
- ✓ P3.6— WR BAR
- ✓ P3.7— RD BAR

It works as an IO port same as Port 2 as well as it can do lots of alternate work which are discuss above. Only alternate function of port 3 makesits architecturedifferent than other ports.



1.4.3 Internal Memory:

A functioning computer must have memory for program code bytes, commonlyin ROM, and RAM memory for variable data that can be altered as the programruns. The 8051 has internal RAM and ROM memory for these functions.Additional memory can be added externally using suitable circuits.Unlike microcontrollers with Von Neumann architectures, which can use a singlememory address for either program code or data, but not for both, the 8051 has aHarvard architecture, which uses the same address, in different memories, forcode and data. Internal circuitry accesses the correct memory based upon thenature of the operation in progress

Internal RAM

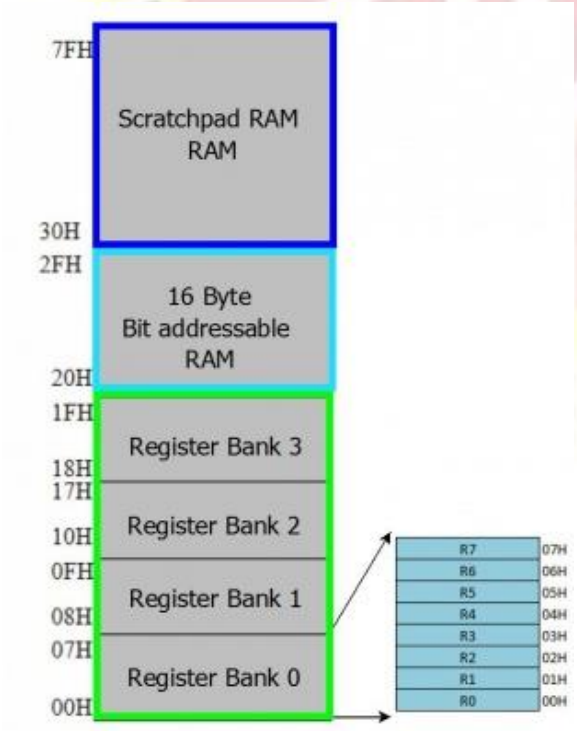
Internal RAM has memory 128-byte. See 8051 hardware for further internal RAM design.Internal RAM is organized into three distinct areas: 32 bytes working registers from address 00hto 1Fh 16 bytes bit addressable occupies RAM byte address 20h to 2Fh, altogether 128addressable bits General purpose RAM from 30h to 7Fh.

Internal ROM

Data memory and program code memory both are in different physical memory, butboth have the same addresses. An internal ROM occupied addresses from 0000h to 0FFFh. PCaddresses program codes from 0000h to 0FFFh. Program addresses higher than 0FFFh thatexceed the internal ROM capacity will cause 8051 architecture to fetch codes bytes fromexternal program memory.

28 bytes of Internal RAM Structure (lower address space)

The lower 32 bytes are divided into 4 separate banks. Each register bank has 8 registers of one byte each.



Aregister bank is selected depending upon two banks select bits in the PSW register. Next 16bytes are bitaddressable. In total, 128bits (16X8) are available in bitaddressable area. Each bit can be accessed andmodified by suitable instructions. The bit addresses are from 00H (LSB of the first byte in 20H) to 7FH (MSBof the last byte in 2FH). Remaining 80bytes of RAM are available for general purpose.

1.4.4 Internal Data Memory and Special Function Register (SFR) Map

Fig 4.6 : Internal Data Memory Map

The special function registers (SFRs) are mapped in the upper 128 bytes of internal data memory address.Hence there is an address overlap between the upper 128 bytes of data RAM and SFRs. Please note that theupper 128 bytes of data RAM are present only in the 8052 family. The lower128 bytes of RAM (00H - 7FH)can be accessed both by direct and indirect addressing while the upper 128 bytes of RAM (80H - FFH) areaccessed by indirect addressing.The SFRs (80H - FFH) are accessed by direct addressing only. This featuredistinguishes the upper 128 bytes of memory from the SFRs, as shown in fig 4.6

SFR Map

The set of Special Function Registers (SFRs) contains important registers such as Accumulator, Register B, I/OPort latch registers, Stack pointer, Data Pointer, Processor Status Word (PSW) and various control registers.

Some of these registers are bit addressable. The detailedmap of various registers is shown in the following figure.

	SFR MEMORY MAP								
F8									FF
F0	B								F7
E8									EF
E0	ACC								E7
D8									DF
D0	PSW								D7
C8	T2CON		RCAP2L	RCAP2H	TL2	TH2			CF
C0									C7
B8	IP								BF
B0	P3								B7
A8	IE								AF
A0	P2								A7
98	SCON	SBUF							9F
90	P1								97
88	TCON	TMOD	TL0	TL1	THO	TH1			8F
80	P0	SP	DPL	DPH				PCON	87
MEMORY LOCATIONS	← 8 BYTES →								MEMORY LOCATIONS

It should be noted that all registers appearing in the first column are bit addressable. The bit address of a bit in the register is calculated as follows.

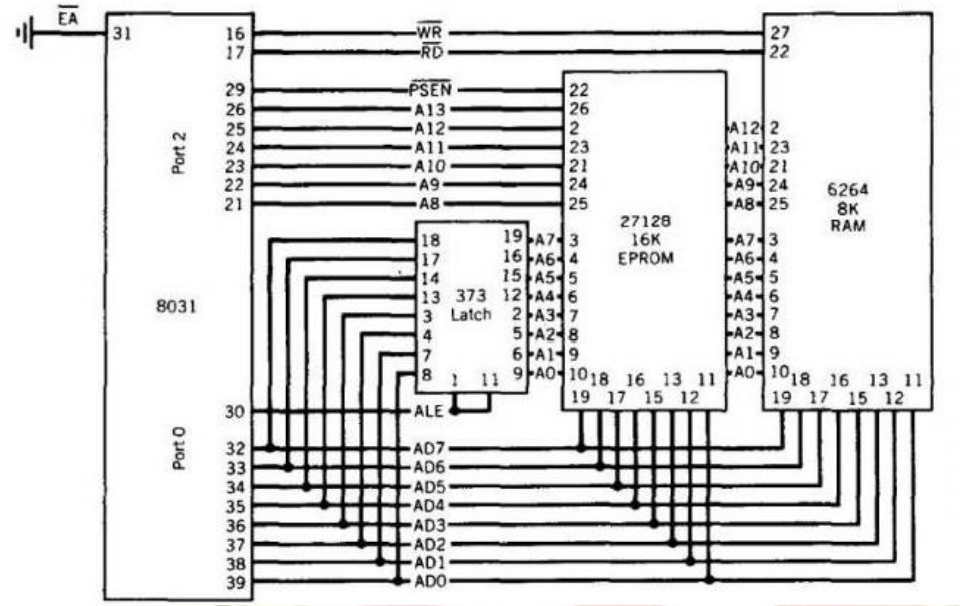
Bit address of 'b' bit of register 'R' is Address of register 'R' + b, where $0 \leq b \leq 7$.

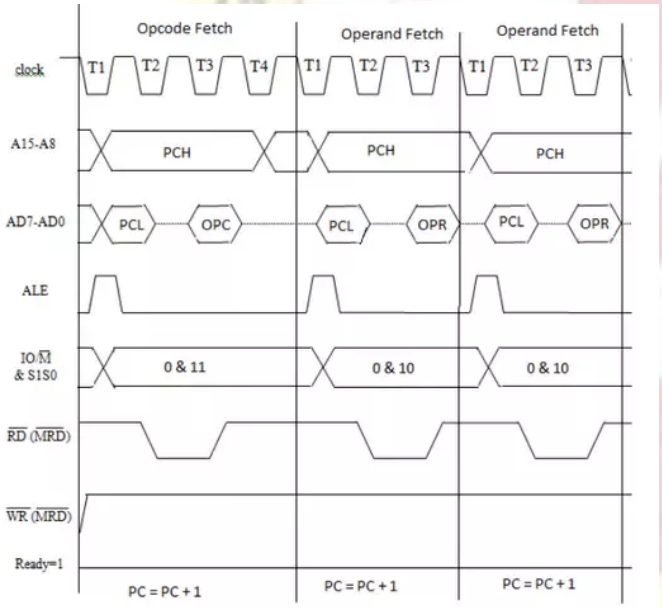
1.5 EXTERNAL MEMORY

The system designer is not limited by the amount of internal RAM and ROMavailable on chip. Two separate external memory spaces are made available bythe 16-bit PC and DPTR and by different control pins for enabling external ROMand RAM chips. Internal control circuitry accesses the correct physical memory,depending upon the machine cycle state and the op code being executed.There are several reasons for adding external memory, particularly programmemory, when applying the 8051 in a system. When the project is in theprototype stage, the expense—in time and money—of having a masked internalROM made for each program "try" is prohibitive.To alleviate this problem, the manufacturers make available an EPROM version,the 8751, which has 4K of on-chip EPROM that may be programmed and erasedas needed as the program is developed. The resulting circuit board layout will beidentical to one that uses a factory-programmed 8051. The only drawbacks to the8751 are the specialized EPROM programmers that must be used to program thenon-standard 40-pin part, and the limit of "only" 4096 bytes of program code.The 8751 solution works well if the program will fit into 4K bytes.Unfortunately, many times, particularly if the program is written in a high-levellanguage,

the program size exceeds 4K bytes, and an external program memory isneeded. Again, the manufacturers provide a version for the job, the ROMIess8031. The EA pin is grounded when using the 8031, and all program code iscontained in an external EPROM that may be as large as 64K bytes and that canbe programmed using standard EPROM programmers.

External RAM, which is accessed by the DPTR, may also be needed when 128bytes of internal data storage are not sufficient. External RAM, up to 64K bytes,may also be added to any chip in the 8051 family.Connecting External MemoryFigure 2.8 shows the connections between an 8031 and an external memoryconfiguration consisting of I6K bytes of EPROM and 8K bytes of static RAM.The 8051 accesses external RAM whenever certain program instructions areexecuted. External ROM is accessed whenever the EA (external access) pin isconnected to ground or when the PC contains an address higher than the lastaddress in the internal 4K bytes ROM (OFFFh). 8051 designs can thus useinternal and external ROM automatically; the 8031, having no internal ROM,must have EA grounded.Figure shows the timing associated with an external memory access cycle.During any memory access cycle, port 0 is time multiplexed. That is, it firstprovides the lower byte of the 16-bit memory address, then acts as a bidirectionaldata bus to write or read a byte of memory data. Port 2 provides the high byte ofthe memory address during the entire memory read/write cycle.The lower address byte from port 0 must be latched into an external register tosavethe byte. Address byte save is accomplished by the ALE clock pulse that providesthe correct timing for the '373 type data latch. The port 0 pins then become free toserve as a data bus.If the memory access is for a byte of program code in the ROM, the PSEN(program store enables) pin will go low to enable the ROM to place a byte ofprogram code on the data bus. If the access is for a RAM byte, the WR (write) orRD (read) pins will go low, enabling data to flow between the RAM and the databus.



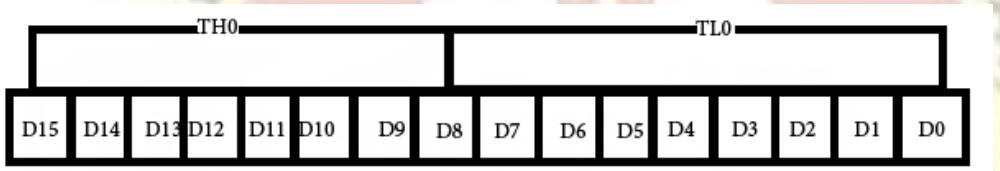


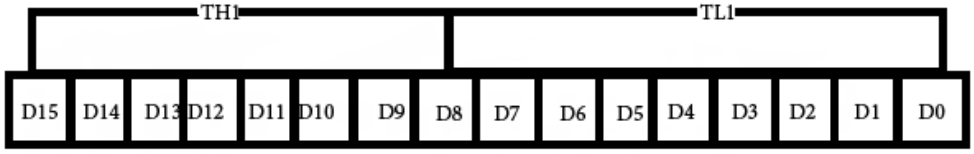
The ROM may be expanded to 64K by using a 27512 type EPROM andconnecting the remaining port 2 upper address lines AI4-A15 to the chip.At this time the largest static RAMs available are 32K in size; RAM can beexpanded to 64K by using two 32K RAMs that are connected through addressA14 of port 2 first 32K RAM (OOOOh-7FFFh) can then be enabled when AI5 of port 2 is low, and the second 32K RAM (0000h-FFFFh) when A15 is high, by using an inverter. Note that the WR and RD signals are alternate uses for port 3 pins 16 and 17. Also, port 0 is used for

the lower address byte and data; port 2 is used for upper address bits. The use of external memory consumes many of the port pins, leaving only port 1 and parts of port 3 for general I/O.

1.6 COUNTER AND TIMERS SERIAL DATA I/O

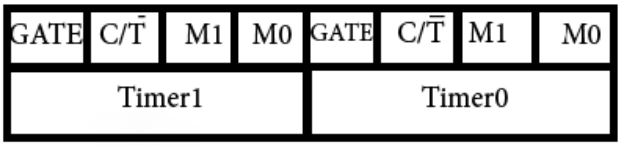
In this article, we focus on Timers/Counters of the 8051-micro controller. The 8051 has two counters/timers which can be used either as timer to generate a time delay or as counter to count events happening outside the microcontroller. The 8051 has two timers: timer0 and timer1. They can be used either as timers or as counters. Both timers are 16-bits wide. Since the 8051 has an 8-bit architecture, each 16-bit is accessed as two separate registers of low byte and high byte. First, we shall discuss about Timer0 registers. Timer0 registers is a 16-bits register and accessed as low byte and high byte. The low byte is referred as a TL0 and the high byte is referred as TH0. These registers can be accessed like any other registers





1.6.1 TMOD & TCON

TMOD (timer mode) Register: This is an 8-bit register which is used by both timers 0 and 1 to set the various timer modes. In this TMOD register, lower 4 bits are set aside for timer0 and the upper 4 bits are set aside for timer1. In each case, the lower 2 bits are used to set the timer mode and upper 2 bits to specify the operation.



TMOD

In upper or lower 4 bits, first bit is a GATE bit. Every timer has a means of starting and stopping. Some timers do this by software, some by hardware, and some have both software and hardware controls. The hardware way of starting and stopping the timer by an external source is achieved by making GATE=1 in the TMOD register. And if we change to GATE=0 then we do no need external hardware to start and stop the timers. The second bit is C/T bit and is used to decide whether a timer is used as a time delay generator or an event counter. If this bit is 0 then it is used as a timer and if it is 1 then it is used as a counter. In upper or lower 4 bits, the last bits third and fourth are known as M1 and M0 respectively. These are used to select the timer mode. M0 M1 Mode Operating Mode 0 0 0 13-bit timer mode, 8-bit timer/counter THx and TLx as 5-bit prescalar.0 1 1 16-bit timer mode, 16-bit timer/counters THx and TLx are cascaded; There are no prescalar. 1 0 2 8-bit auto reload mode, 8-bit auto reload timer/counter; THx holds a value which is to be reloaded into TLx each time it overflows. 1 1 3 Spilt timer mode.

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

Mode 0 is exactly same as mode 1 except that it is a 13-bit timer instead of 16-bit. The 13- bit counter can hold values between 0000 to 1FFFH in TH-TL. Therefore, when the timer reaches its maximum of 1FFH, it rolls over to 0000, and TF is raised. Mode 2- It is an 8-bit timer that allows only values of 00 to FFH to be loaded into the timer's register TH. After TH is loaded with 8-bit value, the 8051 gives a copy of it to TL. Then the timer must be started. It is done by the instruction "SETB TR0" for timer 0 and "SETB TR1" for timer1. This is like mode 1. After timer is started, it starts to count by incrementing the TL register. It counts until it reaches its

limit of FFH. When it rolls over from FFH to 00. It sets high the TF (timer flag). If we are using timer 0, TF0 goes high, if using TF1 then TF1 is raised. When Tl register rolls from FFH to 00 and TF is set to 1, TL is reloaded automatically with the original value kept by the TH register. To repeat the process, we must simply clear TF and let it go without any need by the programmer to reload the original value. This makes mode 2 auto reload, in contrast in mode 1 in which programmer must reload TH and TL. Mode3- Mode 3 is also known as a split timer mode. Timer 0 and 1 may be programmed to be in mode 0, 1 and 2 independently of similar mode for another timer. This is not true for mode 3; timers do not operate independently if mode 3 is chosen for timer 0. Placing timer 1 in mode 3 causes it to stop counting; the control bit TR1 and the timer 1 flag TF1 are then used by timer0. TCON register- Bits and symbol and functions of every bits of TCON are as follows:

TCON

BIT symbol functions 7 TF1 Timer1 overflow flag. Set when timer rolls from all 1s to 0. Cleared When the processor vectors to execute interrupt service routine Located at program address 001Bh. 6 TR1 Timer 1 run control bit. Set to 1 by programmer to enable timer to count; Cleared to 0 by program to halt timer. 5 TF0 Timer 0 overflow flag. Same as TF1. 4 TR0 Timer 0 run control bit. Same as TR1. 3 IE1 External interrupt 1 Edge flag. Not related to timer operations. 2 IT1 External interrupt1 signal type control bit. Set to 1 by program to enable external interrupt 1 to be triggered by a falling edge signal. Set To 0 by program to enable a low-level signal on external interrupt1 to generate an interrupt. 1 IE0 External interrupt 0 Edge flag. Not related to timer operations. 0 IT0 External interrupt 0 signal type control bit. Same as IT0.

1.6.2 Serial communication

Data communication

The 8051 microcontroller is parallel device that transfers eight bits of data simultaneously over eight data lines to parallel I/O devices. Parallel data transfer over a long is very expensive. Hence, a serial communication is widely used in long distance communication. In serial data communication, 8-bit data is converted to serial bits using a parallel in serial out shift register

and then it is transmitted over a single data line. The data byte is always transmitted with least significant bit first.

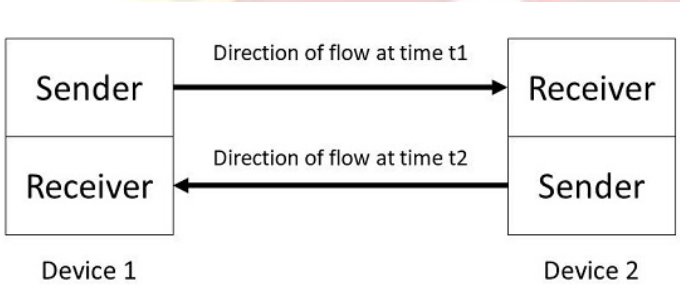
Basics of serial data communication:

1. Simplex communicationlink:

In simplex transmission, the line is dedicated for transmission. The transmitter sends and the receiver receives the data

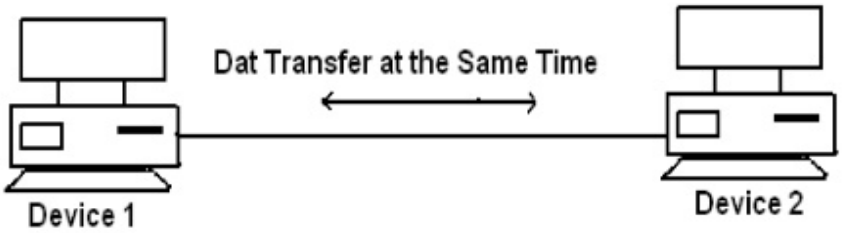
2. Half duplex communication link

In half duplex, the communication link can be used for either transmission or reception. Data is transmitted in only one direction at a time



3. Full duplex communication link

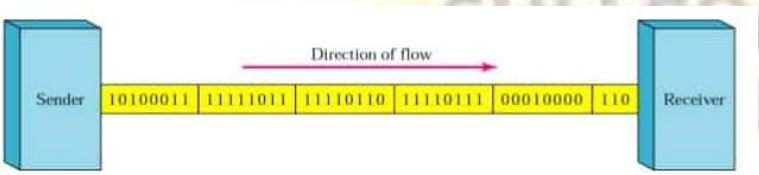
If the data is transmitted in both ways at the same time, it is a full duplex i.e. transmission and reception can proceed simultaneously. This communication link requires two wires for data, one for transmission and one for reception.



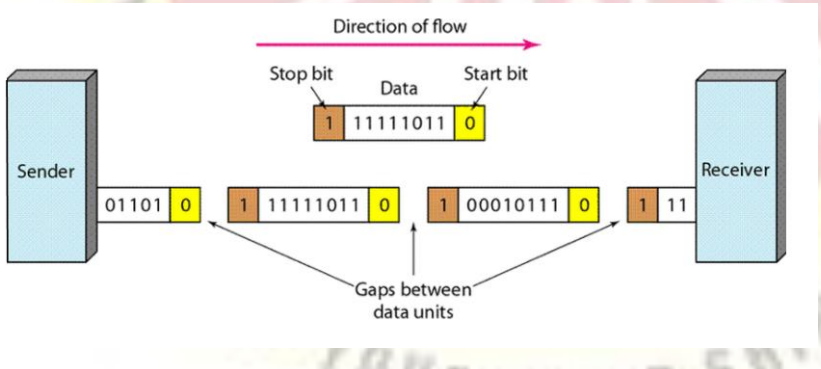
1.6.3 Types of Serial communication

Serial data communication uses two types of communication

Synchronous serial data communication: In this transmitter and receiver are synchronized. It uses a common clock to synchronize the receiver and the transmitter. First the synch character is sent and then the data is transmitted. This format is generally used for high speed transmission. In Synchronous serial data communication, a block of data is transmitted at a time.



Asynchronous Serial data transmission: In this, different clock sources are used for transmitter and receiver. In this mode, data is transmitted with start and stop bits. A transmission begins with start bit, followed by data, and then stop bit. For error checking purpose parity bit is included just prior to stop bit. In Asynchronous serial data communication, a single byte is transmitted at a time.



1.6.4 Baud rate

The rate at which the data is transmitted is called baud or transfer rate. The baud rate is the reciprocal of the time to send one bit. In asynchronous transmission, baud rate is not equal to number of bits per second. This is because; each byte is preceded by a start bit and followed by parity and stop bit. For example, in synchronous transmission, if data is transmitted with 9600 baud, it means that 9600 bits are transmitted in one second. For bit transmission time = 1 second/9600 = 0.104 ms.

1.6.5 Serial communicationregisters

The 8051 supports a full duplex serial port. Three special function registers support serial communication.

SBUF Register: Serial Buffer (SBUF) register is an 8-bit register. It has separate SBUF registers for data transmission and for data reception. For a byte of data to be transferred via the TXD line, it must be placed in SBUF register. Similarly, SBUF holds the 8-bit data received by the RXD pin and read to accept the received data.

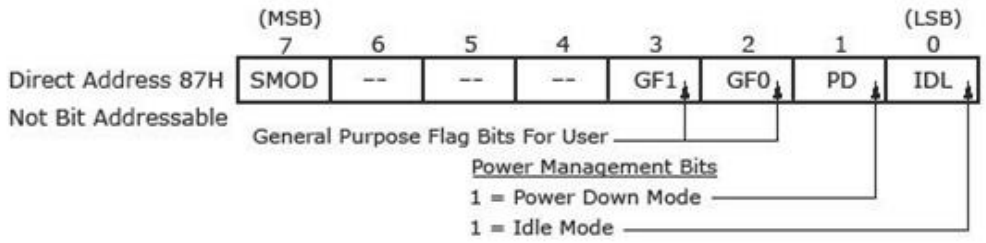
SCON register: The contents of the Serial Control (SCON) register is shown below. This register contains mode selection bits, serial port interrupt bit (TI and RI) and the ninth data bit for transmission and reception (TB8 and RB8).

7	6	5	4	3	2	1	0
SM0	SM1	SM2	REN	TB8	RB8	T1	R1

Bit	Name	Bit Address	Explanation of Function
7	SM0	9Fh	Serial port mode bit 0
6	SM1	9Eh	Serial port mode bit 1.
5	SM2	9Dh	Mutli processor Communications Enable
4	REN	9Ch	Receiver Enable. This bit must be set in order to receive Characters.
3	TB8	9Bh	Transmit bit 8. The 9th bit to transmit in mode 2 and 3.
2	RB8	9AH	Receive bit 8. The 9th bit received in mode 2 and 3.
1	T1	99h	Transmit Flag. Set when a byte has been completely Transmitted.
0	RI	98h	Receive Flag. Set when a byte has been completely Received.

SM0	SM1	Serial Mode	Explanation Baud Rate
0	0	0	0 8-bit Shift Register Oscillator / 12
0	1	1	8-bit UART Set by Timer 1 (*)
1	0	2	9-bit UART Oscillator / 32 (*)
1	1	3	9-bit UART Set by Timer 1 (*)

PCON register: The SMOD bit (bit 7) of PCON register controls the baud rate in asynchronous mode transmission.



1.6.6Serial communication modes

Mode 0

In this mode serial port runs in synchronous mode. The data is transmitted and received through RXD pin and TXD is used for clock output. In this mode the baud rate is 1/12 of clock frequency.

Mode 1

In this mode SBUF becomes a 10-bit full duplex transceiver. The ten bits are 1 start bit, 8 data bit and 1 stop bit. The interrupt flag TI/RI will be set once transmission or reception is over. In this mode the baud rate is variable and is determined by the timer 1 overflow rate.

Baud rate = smod/32] x Timer 1 overflow Rate = [2smod/32] x [Oscillator Clock Frequency] / [12 x [256 – [TH1]]]

Mode 2

This is like mode 1 except 11 bits are transmitted or received. The 11 bits are, 1 start bit, 8 data bits, a programmable 9th data bit, 1 stop bit. Baud rate = [2smod/64] x Oscillator Clock Frequency

1.7INTERRUPTS

During program execution if peripheral devices needs service from microcontroller, device will generate interrupt and gets the service from microcontroller. When peripheral device activates the interrupt signal, the processor branches to a program called interrupt service routine. After executing the interrupt service routine, the processor returns to the main program.

Steps taken by processor while processing an interrupt:

- ✓ It completes the execution of the current instruction.
- ✓ PSW is pushed to stack.
- ✓ PC content is pushed to stack.
- ✓ Interrupt flag is reset.
- ✓ PC is loaded with ISR address.

ISR will always ends with RETI instruction. The execution of RETI instruction results in the following:

- ✓ POP the current stack top to the PC.
- ✓ POP the current stack top to PSW.

1.7.1 Classification of interrupts

External and internal interrupts:External interrupts are those initiated by peripheral devices through the external pins of the microcontroller. Internal interrupts are those activated by the internal peripherals of the microcontroller like timers, serial controller etc.)

Maskable and non-maskable interrupts: The category of interrupts which can be disabled by the processor using program is called maskable interrupts. Non-maskable interrupts are those categories by which the programmer cannot disable it using program.

Vectored and non-vectored interrupt:Starting address of the ISR is called interrupt vector. In vectored interrupts the starting address is predefined. In non-vectored interrupts, the starting address is provided by the peripheral as follows. Microcontroller receives an interrupt request from external device.

Controller sends an acknowledgement (INTA) after completing the execution ofcurrent instruction. The peripheral device sends the interrupt vector to the microcontroller.

8051 has five interrupts. They are maskable and vectored interrupts. Out of these five, two are external interrupt and three are internal interrupts.

1.7.2Interrupt structure

8051 has five interrupts. They are maskable and vectored interrupts. Out of these five, two are external interrupt and three are internal interrupts.

Interrupt Types	ROM Location	Pin
Reset	0000	9
External HW (INT0)	0003	P3.2 (12)
Timer 0 (TF0)	000B	-
External HW (INT1)	0013	P3.3 (13)
Timer 1 (TF1)	001B	-
Serial Com (RI,TI)	0023	-

1.7.3 Interrupt registers

Interrupt registers of 8051 are classified into two registers:

- ✓ Interrupt enable register
- ✓ Interrupt parity register

Interrupt enable register (IE)

This is an 8-bit register used for enabling or disabling the interrupts. The structure of IE register is shown below

IE : Interrupt Enable Register (Bit Addressable)

If the bit is 0, the corresponding interrupt is disabled. If the bit is 1, the corresponding interrupt is enabled.

EA	–	–	ES	ET1	EX1	ET0	EX0

EA	IE.7	Disables all interrupts. If EA = 0, no interrupt will be acknowledged. If EA = 1, interrupt source is individually enable or disabled by setting or clearing its enable bit.
-	IE.6	Not implemented, reserved for future use*.
-	IE.5	Not implemented, reserved for future use*.
ES	IE.4	Enable or disable the Serial port interrupt.
ET1	IE.3	Enable or disable the Timer 1 overflow interrupt.
EX1	IE.2	Enable or disable External interrupt 1.
ET0	IE.1	Enable or disable the Timer 0 overflow interrupt.
EX0	IE.0	Enable or disable External Interrupt 0.

Interrupt parityregister (IP)

This is an 8-bit register used for setting the priority of the interrupts.

Interrupt Priorities (IP) Register

---	---	PT2	PS	PT1	PX1	PT0	PX0

IP.7: **reserved**
IP.6: **reserved**
IP.5: **timer 2 interrupt priority bit(8052 only)**
IP.4: **serial port interrupt priority bit**
IP.3: **timer 1 interrupt priority bit**
IP.2: **external interrupt 1 priority bit**
IP.1: **timer 0 interrupt priority bit**
IP.0: **external interrupt 0 priority bit**

UNIT II

8051 INSTRUCTION SET

2.0 INTRODUCTION

The 8051 instruction sets are classified in to five functional groups. They are:

- ✓ Arithmetic Instruction
- ✓ Logical Instruction
- ✓ Data transfer Instruction
- ✓ Boolean Instruction
- ✓ Branching Instruction

2.1 ADDRESSING MODES OF 8051

The CPU can access data from different places like registers, the memory or immediate values given by the programmer. The method (or channels) used by the CPU to access data to perform an operation is known as an addressing mode.They are classified into five types:

- ✓ Immediate addressing mode
- ✓ Register addressing mode
- ✓ Direct addressing mode
- ✓ Register indirect addressing mode
- ✓ Indexed addressing mode

2.1.1 Immediate addressing mode

In this addressing mode, the source operand is a constant and it is specified immediately after the opcode (i.e. the source data is specified in the instruction itself. The immediate data must be preceded by the hash (#) sign.

It is used to load information into any of the registers including the DPTR register.

e.g.) MOV A, #52H; Load 52H into A

2.1.2 Register addressing mode

In this addressing mode, the data is placed in one of the CPU registers. Registers A, Rn (n = 0 to 7) are used to hold data registers (R0 to R7) selected from register bank select bit RS0, RS1 of the PSW. We can move data between accumulator and Rn (n = 0 to 7) but the movement of data between Rn registers is not permitted.

e.g.) MOV A, R0; Load the contents of R0 into A

2.1.3 Direct addressing mode

The operand 8-bit address is given in the instruction itself. The data is available in the internal RAM, only data RAM and SFR is addressed.

There are 128 bytes of RAM in 8051. The RAM has been assigned addresses 00 to 7FH.

e.g.) MOV R0, 60H; Save contents of RAM 60H into R0

2.1.4 Register indirect addressing mode

In the register indirect addressing mode, a register is used as a pointer to the data. The address of the operand is specified by one of the CPU register content. The address register for 8-bit address can be R0 or R1 of register bank.

The address register for 16-bit address can only be the 16-bit address can only be the 16-bit data register (DPTR). 8-bit address is used to access the data from RAM location.

When R0 and R1 are used as pointers i.e. When they hold the addresses of RAM location. They must be preceded by the @sign.

e.g.) MOV A, @R0; Move the contents of RAM location whose address is held by R0 into A.

2.1.5 Indexed addressing mode

Indexed addressing mode is widely used in accessing data elements of lookup table entries located in the program ROM space of the 8051. The instruction used for the purpose is MOV C A, @ A + DPTR, the 16-bit register DPTR and register A are used to form the address of the data element stored in on-chip ROM, because the data elements are stored in the program memory.

The instruction MOV C is used instead of MOV with C denoting program code. In this instruction the contents of A are added to the 16-bit register DPTR to form the 16-bit address.

2.2 NOTATION DESCRIPTIONS

Rn	Register R0-R7 of the currently selected register bank
Direct	8-bit internal data location's address. This could be internal data RAM or SFR
@Ri	8-bit internal data RAM location addressed indirectly through register R0-R7
# data	8-bit constant
# data16	16-bit constant
Addr 16	16-bit destination address used by LCALL and AJMP
Addr 11	11-bit destination address used by ACALL and AJMP
rel	Signed (two's complement) 8-bit offset byte used by SJMP
Bit	Direct addressed bit in internal RAM or SFR

2.3 LOGICAL OPERATIONS

Logical operations perform Boolean operations (AND, OR, NOR, and NOT) on data bytes on a bit-by-bit basis. Other logical operations are clear accumulator, rotate accumulator left and right, and swap nibbles in accumulator.

Mnemonic	Description
ANL A, Rn	A = A & [Rn]
ANL A, direct	A = A & [direct memory]
ANL A,@Ri	A = A & [memory pointed to by Ri]
ANL A,#data	A= A & immediate data
ANL direct,A	[direct] = [direct] & A
ANL direct,#data	[direct] = [direct] & immediate data
ORL A, Rn	A = A OR [Rn]
ORL A, direct	A = A OR [direct]
ORL A,@Ri	A = A OR [@RI]
ORL A,#data	A = A OR immediate data
ORL direct,A	[direct] = [direct] OR A
ORL direct,#data	[direct] = [direct] OR immediate data
XRL A, Rn	A = A XOR [Rn]
XRL A, direct	A = A XOR [direct memory]
XRL A,@Ri	A = A XOR [@Ri]
XRL A,#data	A = A XOR immediate data
XRL direct,A	[direct] = [direct] XOR A
XRL direct,#data	[direct] = [direct] XOR immediate data
CLR A	Clear A
CPL A	Complement A
RL A	Rotate A left
RLC A	Rotate A left (through C)
RR A	Rotate A right
RRC A	Rotate A right (through C)
SWAP A	Swap nibbles

ANL <dest_byte>, <source_byte>

- ✓ It performs logical AND operations.
- ✓ Operation on the source and destination operands and stores the result in the destination variable.
- ✓ No flags are affected.

e.g., ANL A, R2

If ACC = D3H (11010011)

R2 = 75H (01110101)

The result of the instruction is ACC = 51_H (01010001)

e.g., ANL PI, #10111001B

ORL <dest_byte>, <source_bytes>

- ✓ This instruction performs the logical OR operation on the source and destination operands and stores the result in the destination variable.
- ✓ No flags are affected.

e.g., ORL A, R2

If ACC = D3H (11010011)

R2 = 75H (01110101)

The result of the instruction is ACC = $F7_H$ (11110111)

ORL PI, #110000IOB

This instruction sets bits 7, 6 and 1 of output port 1.

XRL <dest_byte>, <source_bytes>

- ✓ This instruction performs logical XOR operation on the source and destination operands and stores the result in the destination variable.
- ✓ No flags are affected.

e.g., XRL A, R0

if ACC = C3H (11000011) and

R0 = AAH (10101010)

Then the instruction in ACC = 69H (01101001)

e.g., XRL P1, #00110001

This instruction complements bits 5, 4, and 0 of output port 1.

CRA

- ✓ It clears the accumulator.
- ✓ No flags are affected.

If ACC = C3H, then the instruction results in ACC = 00H.

CPL A

- ✓ This instruction logically complements each bit of the accumulator (1's complement).
- ✓ No flags are affected.

If ACC = C3H (11000011), then the instruction results in ACC = 3CH (00111100).

2.4 BOOLEAN INSTRUCTIONS

The 8051 can perform single bit operations. The operations include set, clear, AND, OR, and complement instructions. It also includes bit-level moves or conditional jump instructions. All bit accesses use direct addressing.

Mnemonic	Description
CLR C	Clear C
CLR bit	Clear direct bit
SETB C	Set C
SETB bit	Set direct bit
CPL C	Complement c
CPL bit	Complement direct bit
ANL C,bit	AND bit with C
ANL C,/bit	AND NOT bit with C
ORL C,bit	OR bit with C
ORL C,/bit	OR NOT bit with C
MOV C,bit	MOV bit to C
MOV bit,C	MOV C to bit
JC rel	Jump if C set
JNC rel	Jump if C not set
JB bit,rel	Jump if specified bit set
JNB bit,rel	Jump if specified bit not set
JBC bit,rel	if specified bit set then clear it and jump

CLR <bit>

- ✓ This operation clears the specified bit indicated in the instruction.
- ✓ No other flags are affected.
- ✓ CLR instruction can operate on the carry flag or and directly addressable bit.

e.g.) CLR P2.7

If port 2 has been written with DCH (11011100), then this operation leaves the port set to 5CH (01011100).

SET B <bit>

- ✓ This operation sets the specified bit to 1.
- ✓ SET B instruction can operate on the carry flag or any directly addressable bit.
- ✓ No other flags are affected.

e.g.) SET B C

SET B P2.0

If the carry flag is cleared and the output Port 2 has the value of 24H (00100100), then the result of the instruction sets the carry flag to 1 and changes the Port 2 value to 25H (00100101).

CPL <bit>

- ✓ This operation complements the bit indicated by the operand.
- ✓ No other flags are affected.
- ✓ CPL instruction can operate on the carry flag or any directly addressable bit.

e.g.) CPL P2.2

If Port 2 has the value of 53H (01010011) before the start of the instruction, then after the execution of the instruction it leaves the port set to 55H (01010101).

ANL C, <source_bit>

- ✓ This instruction ANDs the bit addressed with the carry bit and stores the result in the carry bit itself.
- ✓ If the source bit is a logical 0, then the instruction clears the carry flag else it is left with its original value.

- ✓ If the slash (/) is used in the source operand bit, it means that the logical component of the addressed source bit is used, but the source bit itself is not affected.
- ✓ No other flags are affected

eg) MOV C, P2.0 ; Load C with input pin state of P2.0
ANL C, P2.7 ; AND carry flag with bit 7 of P2

ORL C <source_bit>

- ✓ This instruction OR, the bit is addressed with the carry bit and stores the result in the carry bit itself.
- ✓ It sets the carry flag if the source bit is a logical 1, else the carry is left in its original value.
- ✓ If a slash (/) is used in the source operand bit it means that the logical complement of the addressed source bit is used, but the source bit itself is not affected.
- ✓ No other flags are affected

e.g.) MOV C, P2.0; Load C with input pin of P2.0
ORL C, P2.7; OR carry flag with bit 7 of P2

MOV <dest_bit>, <source_bit>

- ✓ The instruction loads the value of source operand bit into the destination operand bit.
- ✓ One of the addressable operands must be the carry flag, the other may be any directly addressable bit.
- ✓ No other flags are affected.

e.g.) MOV P2.3, C; Load the value of C to P2.3

JC rel

- ✓ This instruction branches to the address, indicated by the label, if the carry flag is set, otherwise the program continues to the next instruction.
- ✓ No flags are affected.

e.g.) CLR C
SUBB A, R0
JC ARRAY 1

MOV A, #20H

The carry flag is cleared initially. After the SUBB instruction, if the value of A is smaller than R0, then the instruction sets the carry flag and causes program execution to branch to ARRAY 1 address, otherwise it continues to the MOV instruction.

JNC rel

- ✓ This instruction branches to the address, indicated by the label, if the carry flag is not set, otherwise the program continues to the next instruction.
- ✓ No flags are affected.
- ✓ The carry flag is not modified.

e.g.) CLR C

SUBB A, R0

JNC ARRAY 2

MOV A, #20H

The above sequence of instructions will cause to jump to be taken if the value of A is greater than or equal to R0. Otherwise the program will continue to the MOV instruction.

JB <bit>, rel

- ✓ This instruction jumps to the address indicated if the instruction bit is 1, otherwise the program continues to the next instruction.
- ✓ No flags are affected. The bit tested is not modified.

e.g.) JB ACC.7, ARRAY 1

JB P1.2, ARRAY 2

If the accumulator values is 01001010 and Port 1 = 57H (01010111), then the above instruction sequence will cause the program to branch to the instruction at ARRAY 2.

JNB <bit>rel

- ✓ This instruction jumps to the address indicated if the destination bit is 0, otherwise the program continues to the next instruction.
- ✓ No flags are affected.

e.g.) JNB ACC.6, ARRAY 1

JNB P1.3, ARRAY 2

If the accumulator value is 01001010 and Port 1 = 57H (01010111), then the above instruction sequence will cause the program to branch to the instruction at ARRAY 2.

JBC <bit>, rel

- ✓ If the source bit is 1, this instruction cleans it and branches to the address indicated, else it proceeds with the next instruction.
- ✓ The bit is not cleared if it is already a 0.
- ✓ No flag is affected.

e.g.) JBC P1.3, ARRAY 1

JBC P1.2, ARRAY 2

If P1 = 56H (01010110) the above instruction sequence will cause the program to branch to the instruction at array 2, modifying P1 to 52H (01010010).

2.5 ROTATE AND SWAP INSTRUCTIONS

RotateLeft Accumulator (RLA):

- ✓ The 8-bits in the accumulator are rotated one bit to left. Bit 7 is rotated into bit 0 position.
- ✓ No flags are affected.

If ACC = C3H (11000011), then the instruction results in ACC = 87H (10000111) with the carry unaffected.

Rotate Left Accumulator with Carry (RLC A)

- ✓ The instruction rotates the accumulator contents one bit to the left through the carry flag.
- ✓ Bit 7 of the accumulator will move into carry flag and the original value of the carry flag will move into the bit 0 position.
- ✓ No flags are affected.

If ACC = C3H (11000011) and the carry flag is 1. The instruction results in ACC = 87H (10000111) with the carry flag set.

Rotate Right Accumulator (RRA)

- ✓ The instruction rotates the accumulator contents one bit to right. Bit 0 is rotated into the bit 7 position.
- ✓ No flags are affected.

If ACC = C3H (11000011), then the instruction results in ACC = E1H (11100001) with the carry unaffected.

Rotate Right Accumulator with Carry (RRC A)

- ✓ This instruction rotates the accumulator contents one bit to the right through the carry flag.
- ✓ The original value of carry flag will move into bit 7 of the accumulator and bit 0 rotated into carry flag
- ✓ No flags are affected.

If ACC = C3H (11000011) and the carry flag is 0. The instruction results in ACC = 61H (01100001) with the carry flag set.

SWAP

- ✓ This instruction interchanges the low order 4-bit nibbles (A3-A0) with the high order 4-bit nibbles (A7-A4) of the accumulator.
- ✓ The operation can also be thought of as a 4-bit rotate instruction.
- ✓ No flags are affected.

If ACC = C3H (11000011), then the instruction leaves ACC = 3CH (00111100).

2.6 DATA TRANSFER INSTRUCTIONS

Data transfer instructions are used to transfer data between an internal RAM location and SFR location without going through the accumulator. Data can also be transferred between the internal and external RAM by using indirect addressing. The upper 128 bytes of data RAM are accessed only by indirect addressing and the SFRs are accessed only by direct addressing.

The Data transfer instructions are move, push, pop, and exchange.

Mnemonic	Description
MOV @Ri, direct	[@Ri] = [direct]
MOV @Ri, #data	[@Ri] = immediate data
MOV DPTR, #data 16	[DPTR] = immediate data
MOVC A,@A+DPTR	A = Code byte from [@A+DPTR]
MOVC A,@A+PC	A = Code byte from [@A+PC]
MOVX A,@Ri	A = Data byte from external ram [@Ri]
MOVX A,@DPTR	A = Data byte from external ram [@DPTR]
MOVX @Ri, A	External[@Ri] = A
MOVX @DPTR,A	External[@DPTR] = A
PUSH direct	Push into stack
POP direct	Pop from stack
XCH A,Rn	A = [Rn], [Rn] = A
XCH A, direct	A = [direct], [direct] = A
XCH A, @Ri	A = [@Rn], [@Rn] = A
XCHD A,@Ri	Exchange low order digits

MOV <dest-byte>, <source-byte>

- ✓ This instruction moves the source byte into the destination location.
- ✓ The source byte is not affected, neither are any other registers or flags.

Example:

MOV R1, #60; R1=60H

MOV A, @R1; A=[60H]

MOV R2, #61; R2=61H

ADD A, @R2; A=A+[61H]

MOV R7, A; R7=A

If internal RAM locations 60H=10H, and 61H=20H, then after theoperations of the above instructions R7=A=30H.

The data contents of memory locations 60H and 61H remain intact.

MOV DPTR, #data 16

- ✓ This instruction loads the data pointer with the 16-bitconstant and no flags are affected.

Example:

MOV DPTR, #1032

This instruction loads the value 1032H into the data pointer,
i.e. DPH=10H and DPL=32H.

MOVC A, @A + <base-reg>

- ✓ This instruction moves a code byte from program memory into ACC.
- ✓ The effective address of the byte fetched is formed by adding the original 8-bitaccumulator contentsand the contents of the base register, which is either thedata pointer (DPTR) or program counter (PC).
- ✓ 16-bit addition is performed, and no flags are affected.
- ✓ The instruction is useful in reading the look-up tables in the program memory.
- ✓ If the PC is used, it is incremented to the address of the following instructionbefore being added to theACC.

Example:

```
CLR A
LOC1: INC A
MOVC A, @A + PC
RET
Look_up DB 10H
DB 20H
DB 30H
DB 40H
```

The subroutine takes the value in the accumulator to 1 of 4 valuesdefined by the DB (define byte) directive. After the operation of the subroutine it returns ACC=20H.

MOVX <dest-byte>, <source-byte>

- ✓ This instruction transfers data between ACC and a byte of external datamemory.
- ✓ There are two forms of this instruction, the only difference between themis whether to use an 8-bit or 16-bit indirect addressing mode to accessthe external data RAM.

- ✓ The 8-bit form of the MOVX instruction uses the EMI0CN SFR todetermine the upper 8-bits of theeffective address to be accessed andthe contents of R0 or R1 to determine the lower 8-bits of the effectiveaddress to be accessed.

Example:

MOV EMI0CN, #10H; Load high byte of address into EMI0CN.

MOV R0, #34H; Load low byte of address into R0(or R1).

MOVX A, @R0; Load contents of 1034H into ACC.

MOVX <dest-byte>, <source-byte>

- ✓ The 16-bit form of the MOVX instruction accesses the memory locationpointed to by the contents of the DPTR register.

Example:

MOV DPTR, #1034H; Load DPTR with 16-bit address to read (1034H).

MOVX A, @DPTR; Load contents of 1034H into ACC.

The above example uses the 16-bit immediate MOV DPTR instruction toset the contents of DPTR.

Alternately, the DPTR can be accessed through the SFR registers DPH,which contains the upper 8-bits ofDPTR, and DPL, which contains thelower 8-bits of DPTR.

PUSH Direct

- ✓ This instruction increments the stack pointer (SP) by 1.
- ✓ The contents of Direct, which is an internal memory location or a SFR,
- ✓ are copied into the internal RAM location addressed by the stack pointer.
- ✓ No flags are affected.

Example:

PUSH 22H

PUSH 23H

Initially the SP points to memory location 4FH and the contents of memory locations 22H and 23H are 11H and 12H respectively. After the

above instructions, SP=51H, and the internal RAM locations 50H and 51H will store 11H and 12H respectively.

POP Direct

- ✓ This instruction reads the contents of the internal RAM locationaddressed by the stack pointer (SP) and decrements the stack pointerby 1. The data read is then transferred to the Direct address which is aninternal memory or an SFR. No flags are affected.

Example:

POP DPH

POP DPL

If SP=51H originally and internal RAM locations 4FH, 50H and 51H contain the values 30H, 11H and 12H respectively, the instructions above leave SP=4FH and DPTR=1211H

POP SP

If the above line of instruction follows, then SP=30H. In this case, SP is decremented to 4EH before being loaded with the value popped (30H).

XCH A, <byte>

- ✓ This instruction swaps the contents of ACC with thecontents of the indicated data byte.

Example:

XCH A, @R0

Suppose R0=2EH, ACC=F3H (11110011) and internal RAM location 2EH=76H (01110110). The result of the above instruction leaves RAM location 2EH=F3H and ACC=76H.

XCHD A,@Ri

- ✓ This instruction exchanges the low order nibble of ACC (bits0-3), with that of the internal RAM location pointed to by RiRegister.

- ✓ The high order nibbles (bits 7-4) of both the registers remainthe same.
- ✓ No flags are affected.

Example:

XCHD A,@R0

If R0=2EH, ACC=76H (01110110) and internal RAM location 2EH=F3H (11110011), the result of the instruction leaves RAM location 2EH=F6H (11110110) and ACC=73H (01110011).

2.7 ARITHMETIC OPERATIONS

Arithmetic operation performs the operations of addition, subtraction, multiplication, division, increment, and decrement the value present in the accumulator.

The value of the status bits in the PSW flag are set when specific conditions are met.

Mnemonic	Description
ADD A, Rn	A = A + [Rn]
ADD A, direct	A = A + [direct memory]
ADD A,@Ri	A = A + [memory pointed to by Ri]
ADD A,#data	A = A + immediate data
ADDC A,Rn	A = A + [Rn] + CY
ADDC A, direct	A = A + [direct memory] + CY
ADDC A,@Ri	A = A + [memory pointed to by Ri] + CY
ADDC A,#data	A = A + immediate data + CY
SUBB A,Rn	A = A - [Rn] - CY
SUBB A, direct	A = A - [direct memory] - CY
SUBB A,@Ri	A = A - [@Ri] - CY
SUBB A,#data	A = A - immediate data - CY
INC A	A = A + 1
INC Rn	[Rn] = [Rn] + 1
INC direct	[direct] = [direct] + 1
INC @Ri	[@Ri] = [@Ri] + 1
DEC A	A = A - 1
DEC Rn	[Rn] = [Rn] - 1
DEC direct	[direct] = [direct] - 1
DEC @Ri	[@Ri] = [@Ri] - 1
MUL AB	Multiply A & B
DIV AB	Divide A by B
DA A	Decimal adjust A

ADD A, <source-byte> and ADDC A,<source-byte>

- ✓ ADD adds the data byte specified by the source operand tothe accumulator, leaving the result in the accumulator.
- ✓ ADDC adds the data byte specified by the source operand,the carry flag, and the accumulator contents, leaving theresult in the accumulator.
- ✓ Operation of both the instructions, ADD and ADDC, canaffect the carry flag (CY), auxiliary carry flag (AC) and the overflow flag (OV).

CY=1 If there is a carryout from bit 7; cleared otherwise.

AC =1 If there is a carryout from the lower 4-bit of A i.e., from bit 3;cleared otherwise.

OV=1 If the signed result cannot be expressed within the numberof bits in the destination operand, cleared otherwise.

SUBB A, <source-byte>

- ✓ SUBB subtracts the specified data byte and the carry flag together fromthe accumulator, leaving the result in the accumulator.
- ✓ CY=1 If a borrow is needed for bit 7; cleared otherwise.
- ✓ AC =1 If a borrow is needed for bit 3, cleared otherwise.
- ✓ OV=1 If a borrow is needed into bit 6, but not into bit 7, or into bit 7,but not into bit 6.

Example:

The accumulator holds 0C1H (11000001B), Register1 holds 40H(01000000B) and the CY=1.The instruction,SUBB A, R1gives the value 70H (01110000B) in the accumulator, with the CY=0 andAC=0 but OV=1.

INC <byte>

- ✓ Increments the data variable by 1. The instruction is used in register,direct or register direct addressing modes.

Example:

MOV R1, #5E

INC R1

INC @R1

If R1=5E (01011110) and internal RAM location 5FH contains 20H, theinstructions will result in R1=5FH and internal RAM location 5FH toincrement by one to 21H.

DEC <byte>

- ✓ The data variable is decremented by 1.
- ✓ The instruction is used in accumulator, register, direct orregister direct addressing modes.
- ✓ A data of value 00H underflows to FFH after the operation.
- ✓ No flags are affected.

INC DPTR

- ✓ Increments the 16-bit data pointer by 1.
- ✓ DPTR is the only 16-bit register that can be incremented.
- ✓ The instruction adds one to the contents of DPTR directly.

MUL AB

- ✓ Multiplies A & B and the 16-bit result stored in [B15-B8], [A7-A0] multiplies the unsigned 8-bit integers in the accumulator and the B register.
- ✓ The Low order byte of the 16-bit product will go to the accumulatorand the High order byte will go to the B register.

If the product is greater than 255 (FFH), the overflow flag is set, otherwise it is cleared. The carry flag is always cleared.

If ACC=85 (55H) and B=23 (17H), the instruction gives the product1955 (07A3H), so B is now 07H and the accumulator is A3H. Theoverflow flag is set, and the carry flag is cleared.

DIV AB

- ✓ Divides A by B. The integer part of the quotient is stored in A and theremainder goes to the B register.
- ✓ If ACC=90 (5AH) and B=05(05H), the instruction leaves 18(12H) in ACC and the value 00 (00H) in B, since 90/5 = 18(quotient) and 00 (remainder).

- ✓ Carry and OV are both cleared.

If B contains 00H before the division operation (divide byzero), then the values stored in ACC and B are undefinedand an overflow flag is set. The carry flag is cleared.

Decimal adjust accumulator (DA A)

- ✓ This is a decimal adjust instruction.
- ✓ It adjusts the 8-bit value in ACC resulting from operationslike ADD or ADDC and produces two 4-bit digits (in packedBinary Coded Decimal (BCD) format).
- ✓ Effectively, this instruction performs the decimal conversionby adding 00H, 06H, 60H or 66H to the accumulator,depending on the initial value of ACC and PSW.
- ✓ If ACC bits A3-0 are greater than 9 (xxxx1010-xxxx1111), orif AC=1, then a value 6 is added to the accumulator toproduce a correct BCD digit in the lower order nibble.
- ✓ If CY=1, because the high order bits A7-4 is now exceeding9 (1010xxxx-1111xxxx), then these high order bits will beincreased by 6 to produce a correct proper BCD in the highorder nibble but not clear the carry.

UNIT III

JUMP and CALL instructions

3.0 INTRODUCTION

Program branching instructions are used to control the flow of actions in a program. Some instructions provide decision making capabilities and transfer control to other parts of the program e.g. conditional and unconditional branches.

Mnemonic	Description
ACALL addr11	Absolute subroutine call
LCALL addr16	Long subroutine call
RET	Return from subroutine
RETI	Return from interrupt
AJMP addr11	Absolute jump
LJMP addr16	Long jump
SJMP rel	Short jump
JMP @A+DPTR	Jump indirect
JZ rel	Jump if A=0
JNZ rel	Jump if A NOT=0
CJNE A,direct,rel	Compare and Jump if Not Equal
CJNE A,#data,rel	
CJNE Rn,#data,rel	
CJNE @Ri,#data,rel	
DJNZ Rn,rel	Decrement and Jump if Not Zero
DJNZ direct,rel	
NOP	No Operation

3.1 JUMP INSTRUCTIONS

JMP @A + DPTR

- ✓ This instruction adds the 8-bit unsigned value of the ACC to the 16-bit data pointer and the resulting sum is returned to the PC.
- ✓ Neither ACC nor DPTR is altered.
- ✓ No flags are affected.

Example:

```
MOV DPTR, #LOOK_TBL
JMP @A + DPTR
LOOK_TBL: AJMP LOC0
AJMP LOC1
AJMP LOC2
```

If the ACC=02H, execution jumps to LOC1 AJMP is a two-byte instruction.

JZ rel

- ✓ This instruction branches to the destination address if ACC=0; else the program continues to the next instruction.
- ✓ The ACC is not modified, and no flags are affected.

Example:

SUBB A, #20H

JZ LABEL1

DEC A

If ACC originally holds 20H and CY=0, then the SUBB instruction changes ACC to 00H and causes the program execution to continue at the instruction identified by LABEL1; otherwise the program continues to the DEC Instruction.

JNZ rel

- ✓ This instruction branches to the destination address if any bit of ACC is a 1; else the program continues to the next instruction.
- ✓ The ACC is not modified, and no flags are affected.

Example:

DEC A

JNZ LABEL2

MOV RO, A

If ACC originally holds 00H, then the instructions change ACC to FFH and cause the program execution to continue at the instruction identified by LABEL2; otherwise the program continues to MOV instruction.

- ✓ **CJNE <dest-byte>, <source-byte>, rel**

  This instruction compares the magnitude of the dest-byte and the source-byte and branches if their values are not equal.
- ✓ The carry flag is set if the unsigned dest-byte is less than the unsigned integer source-byte; otherwise, the carry flag is cleared.
- ✓ Neither operand is affected.

Example:

CJNE R3, #50H,NEQU

… … ;R3 = 50H

NEQU: JC LOC1; If R3 < 50H

… …; R3 > 50H

LOC1: … …; R3 < 50H

DJNZ <byte>, <rel-addr>

- ✓ This instruction is" decrement jump not zero".
- ✓ It decrements the contents of the destination location and if the resulting value is not 0, branches to the address indicated by the source operand.
- ✓ An original value of 00H underflows to FFH.
- ✓ No flags are affected.

Example:

DJNZ 20H, LOC1

DJNZ 30H, LOC2

DJNZ 40H, LOC3

If internal RAM locations 20H, 30H and 40H contain the values 01H, 5FH and 16H respectively, the above instruction sequence will cause a jump to the instruction at LOC2, with the values 00H, 5EH, and 15H in

the 3 RAM locations.

Note, the first instruction will not branch to LOC1 because the [20H] = 00H, hence the program continues to the second instruction.

Only after the execution of the second instruction (where the location [30H] = 5FH), then the branching takes place.

3.2 JUMPS: BYTE UNCONDITIONAL

AJMP addr11

- ✓ The AJMP instruction transfers program execution to the destination address which is located at the absolute short-range distance (short range means 11-bit address).

- ✓ The destination must therefore be within the same 2 kB block of program memory.

Example:

AJMP NEAR

If the label NEAR is at program memory location 0120H, the AJMP instruction at location 0234H loads the PC with 0120H.

LJMP addr16

- ✓ The LJMP instruction transfers program execution to the destination address which is located at the absolute long-range distance (long range means 16-bit address).
- ✓ The destination may therefore be anywhere in the full 64 kB program memory address space.
- ✓ No flags are affected.

Example:

LJMP FAR_ADR

If the label FAR_ADR is at program memory location 3456H, the LJMP instruction at location 0120H loads the PC with 3456H.

SJMP rel

- ✓ This is a short jump instruction, which increments the PC by 2 and then adds the relative value 'rel' (signed 8-bit) to the PC.
- ✓ This will be the new address where the program would branch to unconditionally.
- ✓ Therefore, the range of destination allowed is from -128 to +127 bytes from the instruction.

Example:

SJMP RELSRT

If the label RELSRT is at program memory location 0120H and the SJMP instruction is located at address 0100H, after executing the instruction, PC=0120H.

3.3 CALLS AND SUBROUTINE

ACALL addr11

- ✓ This instruction unconditionally calls a subroutine indicated by theaddress.
- ✓ The operation will cause the PC to increase by 2, then it pushes the 16-bit PC value onto the stack (low order byte first) and increments thestack pointer twice.
- ✓ The PC is now loaded with the value addr11 and the program executioncontinues from this new location.
- ✓ The subroutine called must therefore start within the same 2 kB block ofthe program memory.
- ✓ No flags are affected.

Example:

ACALL LOC_SUB

If SP=07H initially and the label "LOC_SUB" is at program memorylocation 0567H, then executing the instruction at location 0230H,SP=09H, internal RAM locations 08H and 09H will contain 32Hand 02H respectively and PC=0567H.

LCALL addr16

- ✓ This instruction calls a subroutine located at the indicated address.
- ✓ The operation will cause the PC to increase by 3, then it pushes the 16-bit PC value onto the stack (low order byte first) and increments thestack pointer twice.
- ✓ The PC is then loaded with the value addr16 and the program executioncontinues from this new location.
- ✓ Since it is a Long call, the subroutine may therefore begin anywhere inthe full 64 kB program memory address space.
- ✓ No flags are affected.

Example:

LCALL LOC_SUB

Initially, SP=07H and the label "LOC_SUB" is at program memorylocation 2034H. Executing theinstruction at location 0230H,SP=09H, internal RAM locations 08H and 09H contain 33H

and 02H respectively and PC=2034H.

3.4 INTERRUPTS AND RETURNS

RET

- ✓ This instruction returns the program from a subroutine.
- ✓ RET pops the high byte and low byte address of PC fromthe stack and decrements the SP by 2.
- ✓ The execution of the instruction will result in the program toresume from the location just after the "call" instruction.
- ✓ No flags are affected.
- ✓ Suppose SP=0BH originally and internal RAM locations 0AHand 0BH contain the values 30H and 02H respectively. Theinstruction leaves SP=09H and program execution willcontinue at location 0230H.

RETI

- ✓ This instruction returns the program from an interrupt Subroutine.
- ✓ RETI pops the high byte and low byte address of PC fromthe stack and restores the interrupt logic to accept additionalinterrupts.
- ✓ SP decrements by 2 and no other registers are affected.However, the PSW is not automatically restoredto its pre-interrupt status.
- ✓ After the RETI, program execution will resume immediatelyafter the point at which the interrupt isdetected.
- ✓ Suppose SP=0BH originally and an interrupt is detectedduring the instruction ending at location 0213H
- ✓ Internal RAM locations 0AH and 0BH contain the values 14H and02H respectively.
- ✓ The RETI instruction leaves SP=0BH and returnsprogram execution to location 0214H.

3.5 NOP

- ✓ This is the no operation instruction.
- ✓ The instruction takes one machine cycle operation time.
- ✓ Hence it is useful to time the ON/OFF bit of an output port.

Example:

```
CLR P1.2
NOP
NOP
NOP
NOP
SETB P1.2
```

The above sequence of instructions outputs a low-going output pulse onbit 2 of Port 1 lasting exactly 5 cycles.

Note a simple SETB/CLR generates a 1 cycle pulse, so four additionalcycles must be inserted to have a 5-clockpulse width.

UNIT-IV

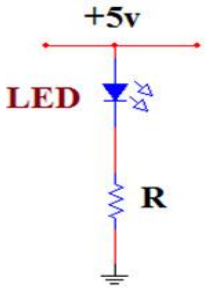
INTERFACING

INTRODUCTION

Interfacing is one of the important concepts in microcontroller 8051 because the microcontroller is a CPU that can perform some operation on a data and gives the output. Interfacing is the process of connecting devices together so that they can exchange the information and that proves to be easier to write the programs. There are different type of input and output devices as for our requirement such as LEDs, LCDs, 7segment, keypad, motors, and other devices.

4.1 LED Interfacing to Microcontroller

LEDs are most used in many applications for indicating the output. They find huge range of applications as indicators during test to check the validity of results at different stages. They are very cheap and easily available in a variety of shape, color, and size.The principle of operation of LEDs is very easy. A simple LEDs also serves as a basic display device, it On and OFF state express meaning full information about a device. The common available LEDs have a 1.7v voltage drop that means when we apply above 1.7V, the diode conducts. The diode needs 10mA current to glow with full intensity.



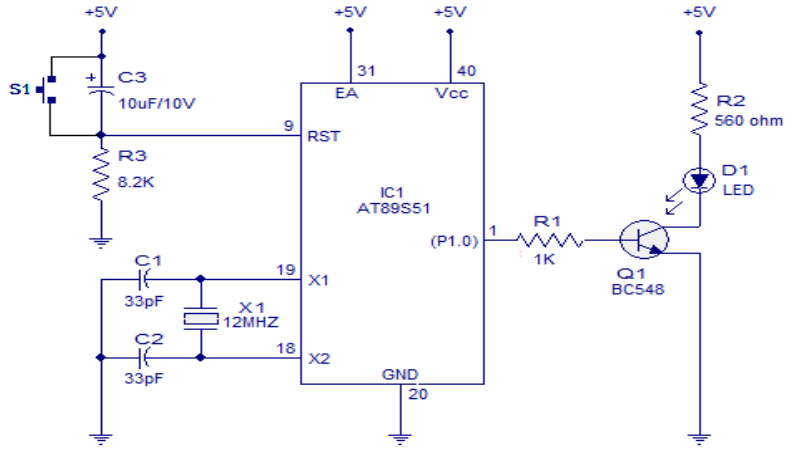
4.1.1 Components Required

- ✓ AT89C51 (8051 Microcontroller)
- ✓ 1 LEDs
- ✓ 1 Resistors – 1KΩ
- ✓ Crystal oscillator – 11.0592MHz
- ✓ Capacitors – 33pF
- ✓ Resistors – 10KΩ
- ✓ 1 Capacitor – 10µF
- ✓ 1 Push Button
- ✓ 8051 Programmer
- ✓ 5V Power Supply

4.1.2 Circuitconnection

LEDs can be interfaced to the microcontroller in either common anode or common cathode configuration. Here the LEDs are connected in common anode configuration because the common cathode configuration consumes more power.

4.1.3Circuit Diagram



4.1.4Source code

```
START: CPL P1.0
       ACALL WAIT
       SJMP START

WAIT:  MOV R4,#05H
WAIT1: MOV R3,#00H
WAIT2: MOV R2,#00H
WAIT3: DJNZ R2,WAIT3
       DJNZ R3,WAIT2
       DJNZ R4,WAIT1
       RET
```

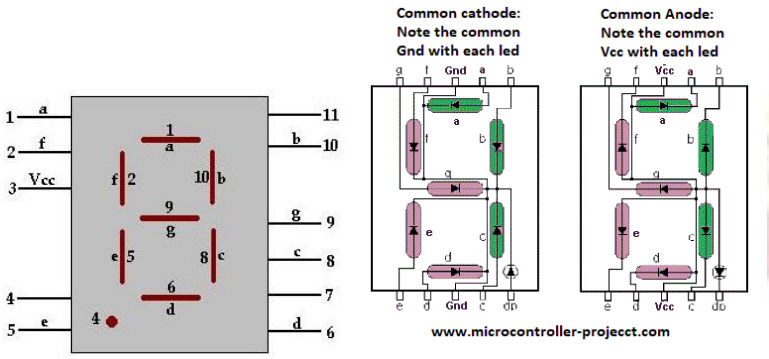
4.1.5Applications

- ✓ LEDs are widely used in many applications like in seven segments.
- ✓ They are used in dot matrix displays.
- ✓ They can be used for streetlights.
- ✓ They are used as indicators.
- ✓ They can be used in traffic lights.
- ✓ They are used in emergency lights
- ✓ They can used to make electronic designs.

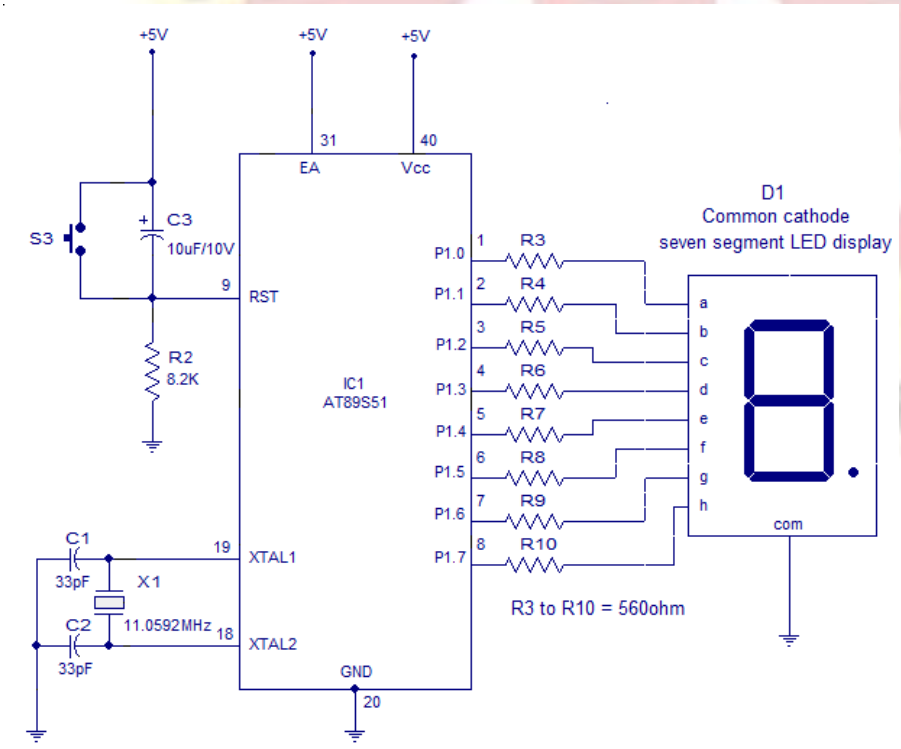
4.2 7-Segment Display interfacing circuit

A Seven segment display is the most basic electronic display. It consists of eight LEDs which are associated in a sequence manner to display digits from 0 to 9 when proper combinations of LEDs are switched on. A 7-segment display uses seven LEDs to display digits from 0 to 9 and the 8th LED is used for dot. A typical seven segment looks likes as shown in figure below. The 7-segment displays are used in several systems to display the numeric information. They can display one digit at a time. Thus, the number of segments used depends on the number of digits to display. Here the digits 0 to 9 are displayed continuously at a predefined time delay.

The 7-segment displays are available in two configurations which are common anode and common cathode. Here common anode configuration is used because output current of the microcontroller is not sufficient to drive the LEDs. The 7-segment display works on negative logic, we must provide logic 0 to the corresponding pin to make on LED glow.



4.2.1 Circuit Diagram



4.2.2 Source Code

```
ORG 000H //initial starting address
START: MOV A, #00001001B // initial value of accumulator
MOV B, A
MOV R0, #0AH //Register R0 initialized as counter which counts from 10 to 0
LABEL: MOV A, B
INC A
MOV B, A
MOVC A, @A+PC // adds the byte in A to the program counters address
MOV P1, A
ACALL DELAY // calls the delay of the timer
DEC R0//Counter R0 decremented by 1
MOV A, R0 // R0 moved to accumulator to check if it is zero in next instruction.
JZ START //Checks accumulator for zero and jumps to START. Done to check if counting has been finished.
SJMP LABEL
DB 3FH // digit drive pattern for 0
DB 06H // digit drive pattern for 1
DB 5BH // digit drive pattern for 2
DB 4FH // digit drive pattern for 3
DB 66H // digit drive pattern for 4
DB 6DH // digit drive pattern for 5
DB 7DH // digit drive pattern for 6
DB 07H // digit drive pattern for 7
DB 7FH // digit drive pattern for 8
DB 6FH // digit drive pattern for 9
DELAY: MOV R4, #05H // subroutine for delay
WAIT1: MOV R3, #00H
WAIT2: MOV R2, #00H
WAIT3: DJNZ R2, WAIT3
DJNZ R3, WAIT2
```

```
DJNZ R4, WAIT1
RET
END
```

4.2.3 Applications

- ✓ The applications of seven segments are mostly in digital calculators, electronic meters, digital clocks, odometers, digital clocks, clock radios, etc.
- ✓ Today most of the 7 segment applications are using LCDs, because of low current consumption.

4.3 LCD Interfacing to Microcontroller

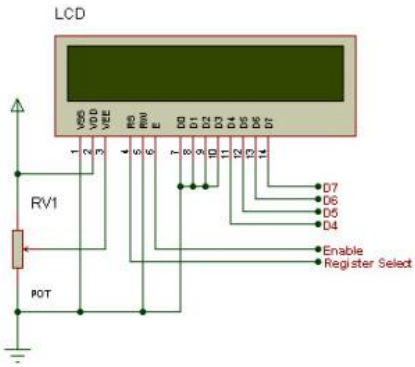
LCD stands for liquid crystal display which can display the characters per line. Here 16 by 2 LCD display can display 16 characters per line and there are 2 lines. In this LCD each character is displayed in 5*7-pixel matrix.

LCD is very important device which is used for almost all automated devices such as washing machines, an autonomous robot, power control systems and other devices. This is achieved by displaying their status on small display modules like 7-seven segment displays, multi segment LEDs etc. The reasons being LCDs are reasonably priced, easily programmable and they have a no limitations of displaying special characters.It consists of two registers such as command/instruction register and data register.

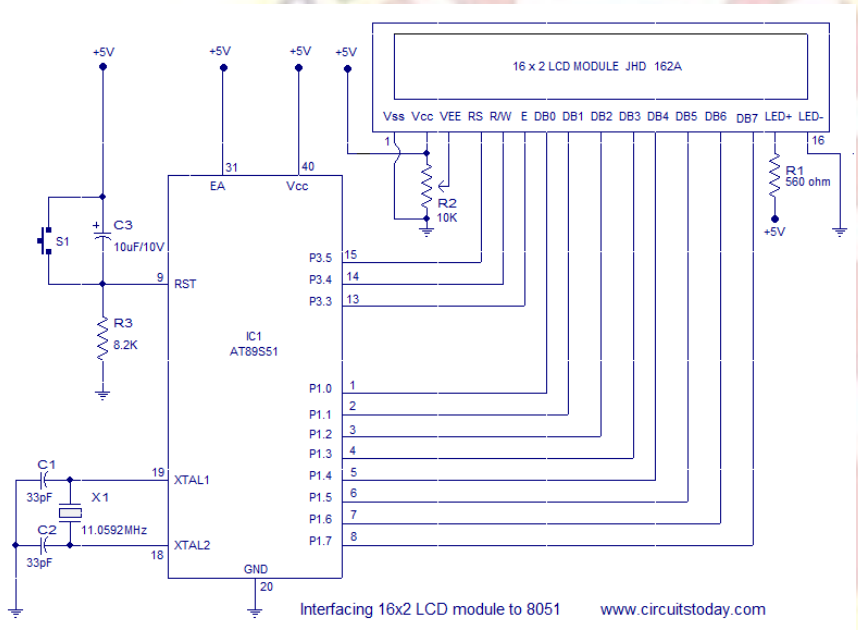
The command/instruction register stores the command instructions given to the LCD. A command is an instruction which is given to the LCD that perform a set of predefined tasks like initializing, clearing the screen, setting the cursor posing, controlling display etc.

The data register stores the data to be displayed on LCD. The data is an ASCII value of the characters to be displayed on the LCD.

Operation of LCD is controlled by two commands. When RS=0, R/W=1 it reads the data and when RS=1, R/W=0, it writes (print) the data.



4.3.1 Circuit Diagram



4.3.2 Commands

Hex Rode	Command to LCD Instruction Register
1	clear screen display
2	Return home
4	decrement cursor
6	increment cursor
E	display ON, cursor ON
80	force the cursor to the begining of the 1st line
c0	force cursor to the begining of the 2nd line
38	Use 2 lines and 5*7 matrics

4.3.3 Source code

```
MOV A, #38H // Use 2 lines and 5x7 matrix
ACALL CMND
MOV A, #0FH // LCD ON, cursor ON, cursor blinking ON
ACALL CMND
MOV A, #01H //Clear screen
ACALL CMND
MOV A, #06H //Increment cursor
ACALL CMND
MOV A, #82H //Cursor line one, position 2
ACALL CMND
MOV A, #3CH //Activate second line
ACALL CMND
MOV A, #49D
ACALL DISP
MOV A, #54D
ACALL DISP
MOV A, #88D
ACALL DISP
MOV A, #50D
ACALL DISP
MOV A, #32D
ACALL DISP
MOV A, #76D
ACALL DISP
MOV A, #67D
ACALL DISP
MOV A, #68D
ACALL DISP

MOV A, #0C1H //Jump to second line, position 1
```

```
ACALL CMND

MOV A, #67D
ACALL DISP
MOV A, #73D
ACALL DISP
MOV A, #82D
ACALL DISP
MOV A, #67D
ACALL DISP
MOV A, #85D
ACALL DISP
MOV A, #73D
ACALL DISP
MOV A, #84D
ACALL DISP
MOV A, #83D
ACALL DISP
MOV A, #84D
ACALL DISP
MOV A, #79D
ACALL DISP
MOV A, #68D
ACALL DISP
MOV A, #65D
ACALL DISP
MOV A, #89D
ACALL DISP

HERE: SJMP HERE
```

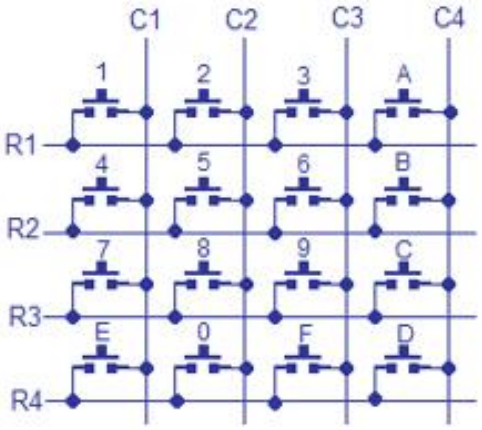
```
CMND: MOV P1, A
CLR P3.5
CLR P3.4
SETB P3.3
CLR P3.3
ACALL DELY
RET

DISP:MOV P1, A
SETB P3.5
CLR P3.4
SETB P3.3
CLR P3.3
ACALL DELY
RET

DELY: CLR P3.3
CLR P3.5
SETB P3.4
MOV P1, #0FFh
SETB P3.3
MOV A, P1
JB ACC.7, DELY

CLR P3.3
CLR P3.4
RET

END
```

4.3.4 Applications

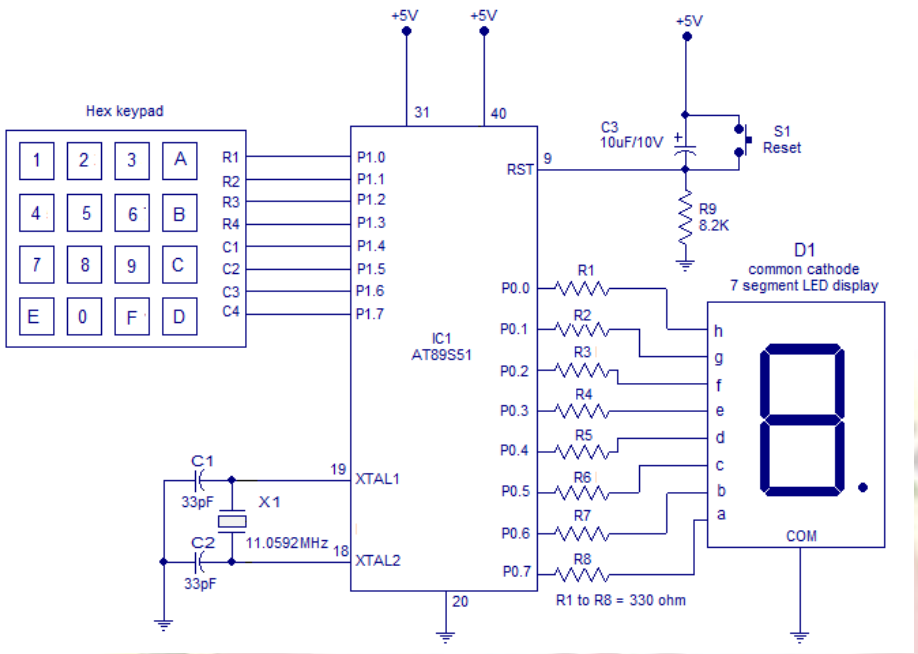
- ✓ The LCDs are commonly used in all the digital wrist watches for displaying time.
- ✓ The LCD (liquid crystal display) is used in aircraft cockpit displays.
- ✓ It is used for displaying images used in digital cameras.
- ✓ It is used in instruments panel where all the lab instruments use LCD screens for display.

4.4 Matrix keypad interfacing to 8051

Keypad is a widely used input device with lot of applications such as telephone, computer, ATM, electronic lock etc. A keypad is used to take input from the user for further processing. Here a 4 by 3 matrix keypad consisting of switches arranged in rows and columns is interfaced to the microcontroller. A 16 by 2 LCD is also interfaced for displaying the output. The interfacing concept of keypad is very simple. Every number of keypads is assigned two unique parameters that are row and column (R, C). Hence every time a key is pressed the number is identifying by detecting the row and column numbers of keypad.



4.4.1 Circuit Diagram



4.4.2 Description

Hex keypad is essentially a collection of 16 keys arranged in the form of a 4×4 matrix. Hex keypad usually has keys representing numeric 0 to 9 and characters A to F. The simplified diagram of a typical hex keypad is shown in the figure below.The hex keypad has 8 communication lines namely R1, R2, R3, R4, C1, C2, C3 and C4. R1 to R4 represents the four rows and C1 to C4 represents the four columns. When a particular key is pressed the corresponding row and column to which the terminals of the key are connected gets shorted. For example, if key 1 is pressed row R1 and column C1 gets shorted and so on. The program identifies which key is pressed by a method known as column scanning. In this method a particular row is kept low (other rows are kept high) and the columns are checked for low. If a particular column is found low, then that means that the key connected between that column and the corresponding row (the row that is kept low) is been pressed. For example,if row R1 is initially kept low and column C1 is found low during scanning, that means key 1 is pressed.

4.4.3 Source Code

```
ORG 00H
MOV DPTR, #LUT // moves starting address of LUT to DPTR
```

```
MOV A, #11111111B // loads A with all 1's
MOV P0, #00000000B // initializes P0 as output port

BACK:MOV P1, #11111111B // loads P1 with all 1's
    CLR P1.0  // makes row 1 low
    JB P1.4, NEXT1  // checks whether column 1 is low and jumps to NEXT1 if not low
    MOV A, #0D   // loads a with 0D if column is low (that means key 1 is pressed)
    ACALL DISPLAY  // calls DISPLAY subroutine
NEXT1: JB P1.5, NEXT2 // checks whether column 2 is low and so on...
     MOV A, #1D
     ACALL DISPLAY
NEXT2: JB P1.6, NEXT3
     MOV A, #2D
     ACALL DISPLAY
NEXT3: JB P1.7, NEXT4
     MOV A, #3D
     ACALL DISPLAY
NEXT4: SETB P1.0
     CLR P1.1
     JB P1.4, NEXT5
     MOV A, #4D
     ACALL DISPLAY
NEXT5: JB P1.5, NEXT6
     MOV A, #5D
     ACALL DISPLAY
NEXT6: JB P1.6, NEXT7
     MOV A, #6D
     ACALL DISPLAY
NEXT7: JB P1.7, NEXT8
     MOV A, #7D
     ACALL DISPLAY
```

```
NEXT8: SETB P1.1
       CLR P1.2
       JB P1.4, NEXT9
       MOV A, #8D
       ACALL DISPLAY
NEXT9: JB P1.5, NEXT10
       MOV A, #9D
       ACALL DISPLAY
NEXT10: JB P1.6, NEXT11
        MOV A, #10D
        ACALL DISPLAY
NEXT11: JB P1.7, NEXT12
        MOV A, #11D
        ACALL DISPLAY
NEXT12: SETB P1.2
        CLR P1.3
        JB P1.4, NEXT13
        MOV A, #12D
        ACALL DISPLAY
NEXT13: JB P1.5, NEXT14
        MOV A, #13D
        ACALL DISPLAY
NEXT14: JB P1.6, NEXT15
        MOV A, #14D
        ACALL DISPLAY
NEXT15: JB P1.7, BACK
        MOV A, #15D
        ACALL DISPLAY
        LJMP BACK
```

```
DISPLAY: MOVC A, @A+DPTR // gets digit drive pattern for the current key from LUT
        MOV P0, A       // puts corresponding digit drive pattern into P0
        RET

LUT: DB 01100000B // Look up table starts here
    DB 11011010B
    DB 11110010B
    DB 11101110B
    DB 01100110B
    DB 10110110B
    DB 10111110B
    DB 00111110B
    DB 11100000B
    DB 11111110B
    DB 11110110B
    DB 10011100B
    DB 10011110B
    DB 11111100B
    DB 10001110B
    DB 01111010B
    END
```

4.4.4 Applications

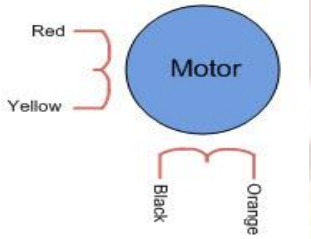
- ✓ Keypads are found on devices which require mainly numeric input such as calculators, television remotes, push-button telephones, vending machines, ATMs, Point of Sale devices, combination locks, and digital door locks.

4.5 Stepper motor interfacing circuit

A stepper motor is one of the most used motor for precise angular movement. The advantage of using a stepper motor is that the angular position of the motor can be controlled without

any feedback mechanism. The stepper motors are widely used in industrial and commercial applications. They are also commonly used as in drive systems such as robots, washing machines etc. Stepper motors can be unipolar or bipolar and here we are using unipolar stepper motor. The unipolar stepper motor consists of six wires out of which four are connected to coil of the motor and two are common wires. Each common wire is connected to a voltage source and remaining wires are connected to the microcontroller. The project is to implement a Stepper Motor Control using 8051 Microcontroller and ULN2003. Since the ULN2003 Transistor Array consists of 7 outputs, you can control both the Unipolar and Bipolar Stepper Motors.

Stepper motor is a brush less motor which converts electrical pulses into mechanical rotation. As the name indicates it rotates in steps according to the input pulses. A stepper motor usually has several field coils (phases) and a toothed rotor. The step size of the motor is determined by the number of phases and the number of teeth on the rotor. Step size is the angular displacement of the rotor in one step. If a stepper motor has 4 phases and 50 teeth, it takes 50×4=200 steps to make one complete rotation. Sostep angle will be 360/200=1.8°.

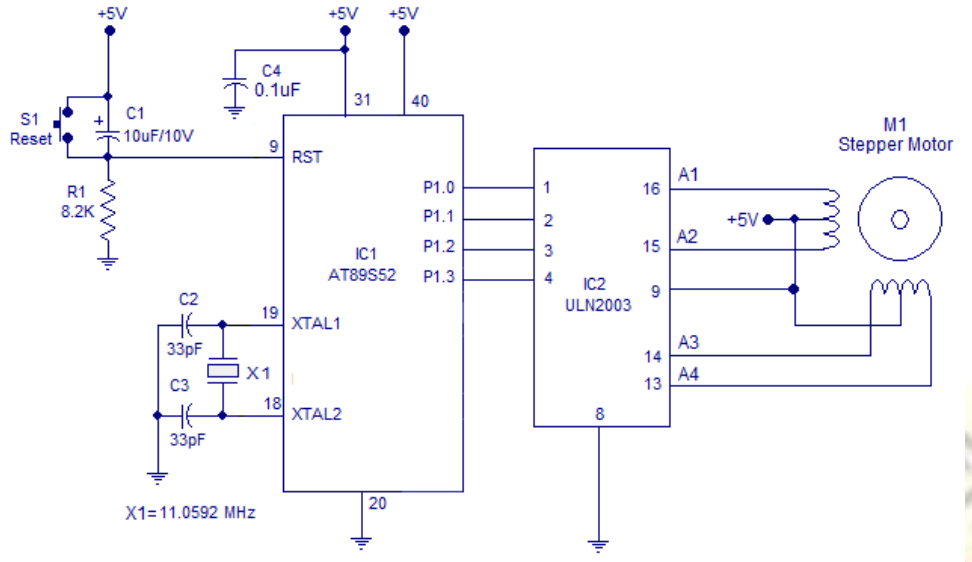


Bipolar Stepper Motor

4.5.1 Circuit Components

- ✓ AT89C51 Microcontroller
- ✓ ULN2003A
- ✓ Stepper Motor
- ✓ Crystal
- ✓ Resistor
- ✓ Capacitor

4.5.2 Circuit Diagram



4.5.3 Description

The circuit consists of AT89C51 microcontroller, ULN2003A, Motor. AT89c51 is low power, high-performance, CMOS 8bit, 8051 family microcontrollers. It has 32 programmable I/O lines. It has 4K bytes of Flash programmable and erasable memory. An external crystal oscillator is connected at the 18 and 19 pins of the microcontroller. Motor is connected to the port2 of the microcontroller through a driver IC.

The ULN2003A is a current driver IC. It is used to drive the current of the stepper motor as it requires more than 60mA of current. It is an array of Darlington pairs. It consists of seven pairs of Darlington arrays with common emitter. The IC consists of 16 pins in which 7 are input pins, 7 are output pins and remaining are VCC and Ground. The first four input pins are connected to the microcontroller. In the same way, four output pins are connected to the stepper motor.

Stepper motor has 6 pins. In these six pins, 2 pins are connected to the supply of 12V and the remaining are connected to the output of the stepper motor. Stepper rotates at a given step angle. Each step-in rotation is a fraction of full cycle. This depends on the mechanical parts and the driving method.

Like all the motors, stepper motors will have stator and rotor. Rotor has permanent magnet and stator has coil. The basic stepper motor has 4 coils with 90 degrees rotation step. These four coils are activated in the cyclic order.

Full step drive

STEP	A	B	C	D
1	1	1	0	0
2	0	1	1	0
3	0	0	1	1
4	1	0	0	1

Half step angles

STEP	A	B	C	D
1	1	0	0	0
2	1	1	0	0
3	0	1	0	0
4	0	1	1	0
5	0	0	1	0
6	0	0	1	1
7	0	0	0	1
8	1	0	0	1

Half Step Mode Clockwise				HEX 16-bits
PB15	PB14	PB13	PB12	
4 Orange	3 Yellow	2 Pink	1 Blue	
1	0	0	1	0x9000
0	0	0	1	0x1000
0	0	1	1	0x3000
0	0	1	0	0x2000
0	1	1	0	0x6000
0	1	0	0	0x4000
1	1	0	0	0xC000
1	0	0	0	0x8000

4.5.4 Source code

```
A1 EQU P1.0
A2 EQU P1.1
A3 EQU P1.2
A4 EQU P1.3
ORG 00H
MOV TMOD, #00000001B

MAIN:
CLR A1
```

```
ACALL DELAY
SETB A1

CLR A2
ACALL DELAY
SETB A2

CLR A3
ACALL DELAY
SETB A3

CLR A4
ACALL DELAY
SETB A4
SJMP MAIN


DELAY:MOV R6, #1D
BACK: MOV TH0, #00000000B
    MOV TL0, #00000000B
    SETB TR0
HERE2: JNB TF0, HERE2
    CLR TR0
    CLR TF0
    DJNZ R6, BACK
    RET
END
```

4.5.5Applications

- ✓ This circuit can be used in the robotic applications.
- ✓ This can also be used in mechatronics applications.

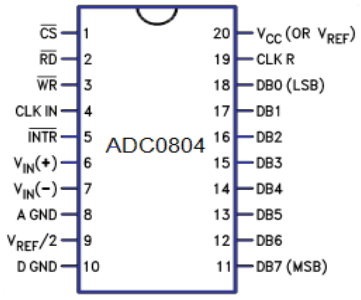
- ✓ The stepper motors can be used in disk drives, matrix printers, etc.

4.6 ADC interfacing

ADC (Analog to digital converter) forms a very essential part in many embedded projects and this article is about interfacing an ADC to 8051 embedded controllers. ADC 0804 is the ADC used here and before going through the interfacing procedure, we must neatly understand how the ADC 0804 works. ADC 0804.

ADC0804 is an 8-bit successive approximation analogue to digital converter from National semiconductors. The features of ADC0804 are differential analogue voltage inputs, 0-5V input voltage range, no zero adjustment, built in clock generator, reference voltage can be externally adjusted to convert smaller analogue voltage span to 8-bit resolution etc.

ADC is the Analog to Digital converter, which converts analog data into digital format; usually it is used to convert analog voltage into digital format. Analog signal has infinite no of values like a sine wave or our speech, ADC converts them into levels or states, which can be measured in numbers as a physical quantity. Instead of continuous conversion, ADC converts data periodically, which is usually known as sampling rate. Telephone modem is one of the examples of ADC, which is used for internet, it converts analog data into digital data, so that computer can understand, because computer can only understand Digital data. The major advantage, of using ADC is that, we noise can be efficiently eliminated from the original signal and digital signal can travel more efficiently than analog one.

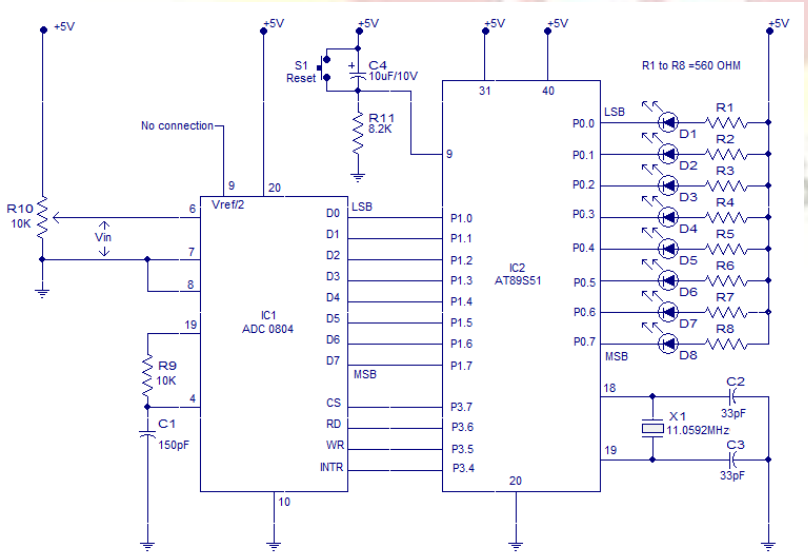


The voltage at Vref/2 (pin9) of ADC0804 can be externally adjusted to convert smaller input voltage spans to full 8-bit resolution. Vref/2 (pin9) left open means input voltage span is 0-5V and step size is 5/255=19.6V. Have a look at the table below for different Vref/2 voltages and corresponding analogue input voltage spans.

4.6.1 Components

- ✓ 8051 Microcontroller (AT89S52)
- ✓ ADC0808/0809
- ✓ 16x2 LCD
- ✓ Resistor (1k,10k)
- ✓ POT(10k x4)
- ✓ Capacitor(10uf,1000uf)
- ✓ Red led
- ✓ Bread board or PCB
- ✓ 7805
- ✓ 11.0592 MHz Crystal
- ✓ Power
- ✓ Connecting wires

4.6.2 Circuit diagram



4.6.3 Description

The figure above shows the schematic for interfacing ADC0804 to 8051. The circuit initiates the ADC to convert a given analogue input, then accepts the corresponding digital data and displays it on the LED array connected at P0. For example, if the analogue input voltage Vin is 5V then all LEDs will glow indicating 11111111 in binary which is the equivalent of 255 in decimal. AT89s51 is the microcontroller used here. Data out pins (D0 to D7) of the ADC0804 are connected to the port pins P1.0 to P1.7 respectively. LEDs D1 to D8 are connected to the port pins P0.0 to P0.7 respectively. Resistors R1 to R8 are current limiting resistors. In simple words P1 of the microcontroller is the input port and P0 is the output port. Control signals for the ADC (INTR, WR, RD and CS) are available at port pins P3.4 to P3.7 respectively. Resistor R9 and capacitor C1 are associated with the internal clock circuitry of the ADC. Preset resistor R10 forms a voltage divider which can be used to apply a particular input analogue voltage to the ADC. Push button S1, resistor R11 and capacitor C4 forms a debouncing reset mechanism. Crystal X1 and capacitors C2,C3 are associated with the clock circuitry of the microcontroller.

ADC0808 gives ratio metric conversion output at its output pins. And the formula for radiometric conversion is given by:

Vin/(Vfs-Vz)= Dx/(Dmax-Dmin)

Where,

Vin is input voltage for conversion

Vfs is full scale Voltage

Vz is zero voltage

Dx is data point being measure

Dmax is Maximum data limit

Dmin is Minimum data limit

4.6.4Source code

```
ORG 00H
MOV P1, #11111111B // initiates P1 as the input port
```

```
MAIN: CLR P3.7 // makes CS=0
      SETB P3.6 // makes RD high
      CLR P3.5 // makes WR low
      SETB P3.5 // low to high pulse to WR for starting conversion
WAIT: JB P3.4, WAIT // polls until INTR=0
      CLR P3.7 // ensures CS=0
      CLR P3.6 // high to low pulse to RD for reading the data from ADC
      MOV A, P1 // moves the digital data to accumulator
      CPL A // complements the digital data (*see the notes)
      MOV P0, A // outputs the data to P0 for the LEDs
      SJMP MAIN // jumps back to the MAIN program
      END
```

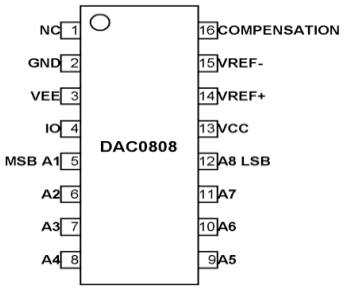
4.6.5Applications

- ✓ Used together with the transducer.
- ✓ Used in computer to convert the analog signal to digital signal.
- ✓ Used in cell phones.
- ✓ Used in microcontrollers.
- ✓ Used in digital signal processing.
- ✓ Used in digital storage oscilloscopes.
- ✓ Used in scientific instruments.
- ✓ Used in music reproduction technology etc.

4.7 DAC interfacing

The digital-to-analog converter (DAC) is a device widely used to convert digital pulses to analog signals. There are two methods to create a DAC: binary weighted and R/2R ladder. Most of the integrated circuit DACs, including the MCl408 (DAC0808) used in this section, use the R/2R method because a much higher degree of precision can be achieved by it. The criterion for judging a DAC is its resolution, which is a function of the number of binary inputs. The common ones are 8, 10, and 12 bits. The number of data bit inputs decides the resolution of the DAC because the number of analog output levels is equal to 2n, where n is

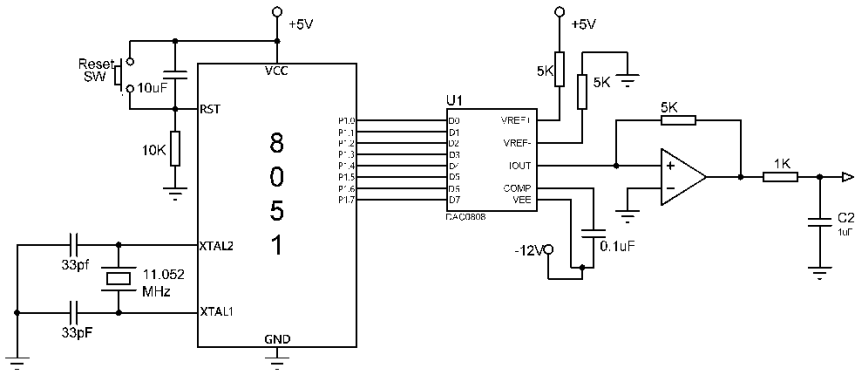
the number of data bit inputs. Therefore, an 8-input DAC such as the DAC0808 provides 256 discrete voltage (or current) levels of output.



4.7.1 Components required

- ✓ MCU (AT89C51)
- ✓ DAC0808
- ✓ Counter (IC 79LS43)
- ✓ Resistors (10k, 8.2k)
- ✓ Capacitors (33pF, 10uF)
- ✓ Op-amp
- ✓ Signal Generator
- ✓ Oscilloscope
- ✓ Power Supply

4.7.2 Circuit Diagram



The value of the sine function varies from -1.0 to +1.0.

Therefore, the values 128,192,238,255,238,192,128,64,17,0,17, and 64 are integer numbers representing the voltage magnitudes for the sine of theta. This method ensures that only integer numbers are output to the DAC by the MCU. To generate the sine wave, the output of DAC is assumed to be 10V. Full scale output of the DAC is achieved when all the data inputs of the DAC are high.

4.7.3 Sourcecode

Tringular Wave Generation using DAC

```
ORG 0000h
mov P1, #00H
repeat: ACALL triwave; generate triangular wave
SJMP repeat
triwave: mov A, #00H
INCR: mov P1, A
INC A
CJNE A, #0FFH, INCR
DECR: mov P1, A
DEC A
CJNE A, #00H, DECR
ret
END
```

4.7.4 Applications

- ✓ DACs are commonly used in music players to convert digital data streams into analog audio signals.
- ✓ They are also used in televisions and mobile phones to convert digital video data into analog video signals.

UNITV

INTRODUCTION TO MICROCONTROLLERS

INTRODUCTION

A micro-controller can be comparable to a little stand-alone computer. A single microcontroller can be enough to manage a small mobile robot, an automatic washer machine or a security system. Several microcontrollers contain a memory to store the program to be executed, and a lot of input/output lines that can be a used to act jointly with other devices, like reading the state of a sensor or controlling a motor.

5.1 NI 6509

The NI 6509 is a 96-bit, high-drive digital input/output (DIO) device for PCI, PXI, and Compact PCI chassis. The NI 6509 features 96 TTL/CMOS-compatible digital I/O lines, 24 mA high-drive output, and the industrial DIO feature set. 6509 is a 96 Channel, 5 V, TTL/CMOS Digital I/O device.

5.1.1 PCIinterface

The NI 6509/651x/6520/6521/6528 use the PCI MITE Application-Specific Integrated Circuit (ASIC) to communicate with the PCI or PXI bus. National Instruments designed this ASIC specifically for data acquisition.

5.1.2 General operation registers

The general operation registers include the IO Port Data registers and corresponding IO Selectregisters for reading and writing data. There are also registers for controlling digital filtering, change detection, the watchdog timer, RTSI output, and PXI synchronization. With the final set of registers, you can read and reset the status of your device. The general operation registers are organized into two groups—recurring and non-recurring registers. Each port has a sequence of recurring registers for Data, IO Select, and other port-specific features. This same set of registers is repeated for each port but incremented 0x10 times the port number above the base address. The non-recurring registers affect theentire board and have set addresses. All NI 6509/651x/6520/6521/6528 devices have the same addresses for common registers.

5.1.3 Recurring port registers

Register Name	Short Name	Offset (Hex)	Type	Size
IO Port Data	IOPort(*N*)Data	0x40 + 0x*N*0	Read-write	8-bit
IO Select	IOSelect(*N*)	0x41 + 0x*N*0	Read-write	8-bit
Rising Edge Sensitivity Configuration	RiseEdgeEnable(*N*)	0x42 + 0x*N*0	Read-write	8-bit
Falling Edge Sensitivity Configuration	FallEdgeEnable(*N*)	0x43 + 0x*N*0	Read-write	8-bit
Filter Enable	FilterEnable(*N*)	0x44 + 0x*N*0	Read-write	8-bit
Watchdog Timer High Impedance	WatchdogHighImp(*N*)	0x46 + 0x*N*0	Read-write	8-bit
Watchdog Timer Enable	WatchdogEnable(*N*)	0x47 + 0x*N*0	Read-write	8-bit
Watchdog Timer High or Low	WatchdogHighLow(*N*)	0x48 + 0x*N*0	Read-write	8-bit
RTSI Enable	RTSI_En(*N*)	0x49 + 0x*N*0	Read-write	8-bit

Note: *N* is the port number in hexidecimal. Ports can range from 0 to 11 (0x0 to 0xB), depending on your device. For each port, you must add an additional offset equal to 0x10 times the port number in hex.

Examples:
1. Offset of Port 6 Data Register (IOPort6Data) = 0x40 + 0x60 = 0xA0
2. Offset of Port 11 Filter Enable Register (FilterEn11) = 0x44 + 0xB0 = 0xF4

5.1.4 Non-recurring port registers

Register Name	Offset (Hex)	Type	Size
ID Register	0x00	Read	8-bit
Clear Register	0x01	Write strobe	8-bit
Change Status Register	0x02	Read	8-bit
Master Interrupt Control Register	0x03	Read-write	8-bit
Revision Register	0x04	Read	32-bit
Filter Interval 32-bit Register	0x08	Read-write	32-bit
Automatic Clock Selection Register	0x14	Bit 0: Write Bit 1: Read	8-bit

5.1.5 Watchdog timer registers

Register Name	Offset (Hex)	Type	Size
Watchdog Timer Software Timeout Enable	0x15	Read-write	8-bit
Watchdog Timer Expire Status	0x17	Read	8-bit
Watchdog Timer Timeout Interval	0x18	Read-write	32-bit

5.1.6 RTSI configuration registers

Register Name	Offset (Hex)	Type	Size
RTSI Input Route	0x0C	Read-write	16-bit
RTSI Pulse when Edge Detected	0x0E	Read-write	16-bit
RTSI Pulse when Watchdog Timer Expires	0x10	Read-write	16-bit
RTSI Trigger for Watchdog Timer	0x12	Read-write	16-bit
RTSI Edge Detection Configuration Register	0x16	Read-write	8-bit

5.1.7 Applications

These devices offer superior features and high value for industrialcontrol and manufacturing test applications such as factory automation,embedded machine control, and production line verification.

Theyhave been designed to incorporate the latest hardware technologiesand provide innovative features for applications requiring ease of use,high reliability, andperformance.
NI 6509 devices takeadvantage ofNI-DAQmx measurement services software (7.1 or later) to speed upapplication development with many helpful features including the DAQAssistant, automatic code generation, and high-performance multithreadedstreaming technology.

5.2 PIC CONTROLLERS

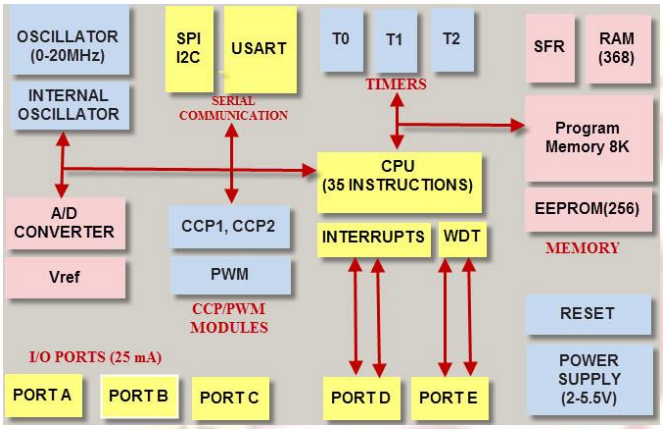
PIC is a Peripheral Interface Microcontroller which was developed in the year 1993 by the General Instruments Microcontrollers. It is controlled by software and programmed in such a way that it performs different tasks and controls a generation line. PIC microcontrollers are used in different new applications such as smartphones, audio accessories, and advanced medical devices.

There are many PICs available in the market ranging from PIC16F84 to PIC16C84. These types of PICs are affordable flash PICs. Microchip has recently introduced flash chips with different types, such as 16F628, 16F877, and 18F452. The 16F877 costs twice the price of the old 16F84, but it is eight times more than the code size, with more RAM and much more I/O pins, a UART, A/D converter and a lot more features. PIC mostly used to modify Harvard architecture and also supports RISC (Reduced Instruction Set Computer) by the above

requirement RISC and Harvard we can simply that PIC is faster than the 8051 based controllers which is prepared up of Von-Newman architecture.

5.3 ARCHITECTURE

The PIC microcontroller is based on RISC architecture. Its memory architecture follows the Harvard pattern of separate memories for program and data, with separate buses.



5.3.1 Memory structure

Program memory

This is a 4Kx14 memory space. It is used to store 13-bit instructions or the program code. The program memory data is accessed by the program counter register that holds the address of the program memory. The address 0000H is used as reset memory space and 0004H is used as interrupt memory space.

Data memory

The data memory consists of the 368 bytes of RAM and 256 bytes of EEPROM. The 368 bytes of RAM consists of multiple banks. Each bank consists of general-purpose registers and special function registers.The general-purpose registers consist of registers that are used to store temporary data and processing results of the data. These general-purpose registers are each 8-bit registers.

Working register

It consists of a memory space that stores the operands for each instruction. It also stores the results of each execution.

5.3.2 Status register

The bits of the status register denote the status of the ALU (arithmetic logic unit) after every execution of the instruction. It is also used to select any one of the 4 banks of the RAM.

5.3.3 File selection register

It acts as a pointer to any other general-purpose register. It consists of a register file address, and it is used in indirect addressing.Another general-purpose register is the program counter register, which is a 13-bit register. The 5 upper bits are used as PCLATH (Program Counter Latch) to independently function as any other register, and the lower 8-bits are used as the program counter bits. The program counter acts as a pointer to the instructions stored in the program memory.

5.3.4 EEPROM

It consists of 256 bytes of memory space. It is a permanent memory like ROM, but its contents can be erased and changed during the operation of the microcontroller. The contents into EEPROM can be read from or written to, using special function registers like EECON1, EECON, etc.

5.3.5 I/O Ports

PIC16 series consists of five ports, such as Port A, Port B, Port C, Port D, and Port E.

Port A: It is a 16-bit port, which can be used as an input or output port based on the status of the TRISA register.

Port B: It is an 8-bit port, which can be used as both an input and output port. 4 of its bits, when used as input, can be changed upon interrupt signals.

Port C: It is an 8-bit port whose operation (input or output) is determined by the status of the TRISC register.

Port D: It is an 8-bit port, which apart from being an I/O port, acts as a slave port for connection to the microprocessor bus.

Port E: It is a 3-bit port that serves the additional function of the control signals to the A/D converter.

5.3.6 Timers

PIC microcontrollers consist of 3 timers, out of which the Timer 0 and Timer 2 are 8-bit timers and the Time-1 is a 16-bit timer, which can also be used as a counter.

5.3.7 A/D Converter

The PIC Microcontroller consists of 8-channels, 10-bit Analog to Digital Converter. The operation of the A/D converter is controlled by these special function registers: ADCON0 and ADCON1. The lower bits of the converter are stored in ADRESL (8-bits), and the upper bits are stored in the ADRESH register. It requires an analog reference voltage of 5V for its operation.

5.3.8 Oscillators

Oscillators are used for timing generation. PIC microcontrollers consist of external oscillators like crystals or RC oscillators. In the case of crystal oscillators, the crystal is connected between two oscillator pins, and the value of the capacitor connected to each pin determines the mode of operation of the oscillator. The different modes are low-power mode, crystal mode, and the high- speed mode. In the case of RC oscillators, the value of the Resistor and Capacitor determines the clock frequency. The clock frequency ranges from 30 kHz to 4 MHz.

5.3.9 CCP module:

A CCP module works in the following three modes:

Capture Mode: This mode captures the time of arrival of a signal, or in other words, captures the value of the Timer1 when the CCP pin goes high.

Compare Mode: It acts as an analog comparator that generates an output when the timer1 value reaches a certain reference value.

PWM Mode: It provides pulse width modulated output with a 10-bit resolution and programmable duty cycle.

Other special peripherals include a Watchdog timer that resets the microcontroller in case of any software malfunction and a Brownout reset that resets the microcontroller in case of any power fluctuation and others. For a better understanding of this PIC microcontroller, we are giving one practical project which uses this controller for its operation.

5.3.10 Advantages

- ✓ Small instruction set to learn
- ✓ RISC architecture
- ✓ Built-in oscillator with selectable speeds
- ✓ Easy entry level, in-circuit programming plus in-circuit debugging PICkit units available for less than $50
- ✓ Inexpensive microcontrollers
- ✓ Wide range of interfaces including I²C, SPI, USB, USART, A/D, programmable comparators, PWM, LIN, CAN, PSP, and Ethernet
- ✓ Availability of processors in DIL package make them easy to handle for hobby use.

5.4 6525 SERIES

- ✓ 64–80-Pin High-Performance
- ✓ 64-Kbyte Enhanced Flash

Microcontrollers with A/D:

Features	PIC18F6525	PIC18F6621	PIC18F8525	PIC18F8621
Operating Frequency	DC – 40 MHz	DC – 40 MHz	DC – 40 MHz	DC – 40 MHz
Program Memory (Bytes)	48K	64K	48K	64K
Program Memory (Instructions)	24576	32768	24576	32768
Data Memory (Bytes)	3840	3840	3840	3840
Data EEPROM Memory (Bytes)	1024	1024	1024	1024
External Memory Interface	No	No	Yes	Yes
Interrupt Sources	17	17	17	17
I/O Ports	Ports A, B, C, D, E, F, G	Ports A, B, C, D, E, F, G	Ports A, B, C, D, E, F, G, H, J	Ports A, B, C, D, E, F, G, H, J
Timers	5	5	5	5
Capture/Compare/PWM Modules	2	2	2	2
Enhanced Capture/Compare/ PWM Module	3	3	3	3
Serial Communications	MSSP, Addressable EUSART (2)	MSSP, Addressable EUSART (2)	MSSP, Addressable EUSART (2)	MSSP, Addressable EUSART (2)
Parallel Communications	PSP	PSP	PSP	PSP
10-bit Analog-to-Digital Module	12 input channels	12 input channels	16 input channels	16 input channels
Resets (and Delays)	POR, BOR, RESET Instruction, Stack Full, Stack Underflow (PWRT, OST)	POR, BOR, RESET Instruction, Stack Full, Stack Underflow (PWRT, OST)	POR, BOR, RESET Instruction, Stack Full, Stack Underflow (PWRT, OST)	POR, BOR, RESET Instruction, Stack Full, Stack Underflow (PWRT, OST)
Programmable Low-Voltage Detect	Yes	Yes	Yes	Yes
Programmable Brown-out Reset	Yes	Yes	Yes	Yes
Instruction Set	77 Instructions	77 Instructions	77 Instructions	77 Instructions
Package	64-pin TQFP	64-pin TQFP	80-pin TQFP	80-pin TQFP

5.4.1Features

- ✓ High current sink/source 25 mA/25 mA
- ✓ Four external interrupt pins
- ✓ Timer0 module: 8-bit/16-bit timer/counter
- ✓ Timer1 module: 16-bit timer/counter
- ✓ Timer2 module: 8-bit timer/counter
- ✓ Timer3 module: 16-bit timer/counter
- ✓ Timer4 module: 8-bit timer/counter
- ✓ Secondary oscillator clock option – Timer1/Timer3
- ✓ Two Capture/Compare/PWM (CCP) modules:
  - Capture is 16-bit, max. resolution 6.25 ns (TCY/16)
  - Compare is 16-bit, max. resolution 100 ns (TCY)
  - PWM output: 1 to 10-bit PWM resolution
- ✓ Three Enhanced Capture/Compare/PWM (ECCP)modules:
  - Same Capture/Compare features as CCP
  - One, two or four PWM outputs

- Selectable polarity
- Programmable dead time
- Auto-Shutdown on external event
- Auto-Restart

✓ Master Synchronous Serial Port (MSSP) modulewith two modes of operation:

- 2/3/4-wire SPI (supports all 4 SPI modes)
- I2C™ Master and Slave mode

✓ Two Enhanced USART modules:

- Supports RS-485, RS-232 and LIN 1.2
- Auto-Wake-up on Start bit
- Auto-Baud Rate Detect

✓ Parallel Slave Port (PSP) module

✓ 10-bit, up to 16-channel Analog-to-DigitalConverter (A/D):

- Auto-Acquisition
- Conversion available during Sleep

✓ Programmable 16-level Low-Voltage Detection(LVD) module:

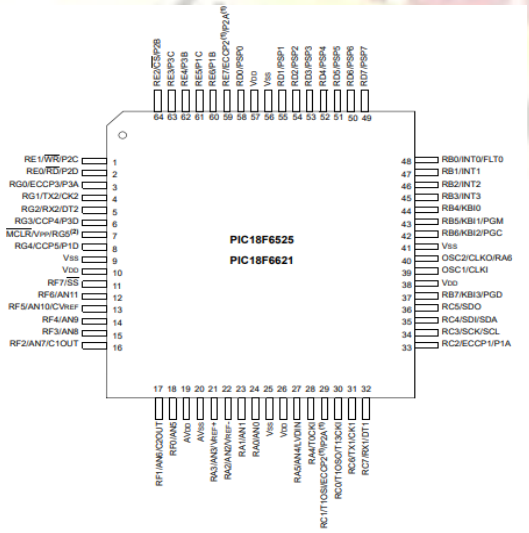
- Supports interrupt on Low-Voltage Detection

✓ Programmable Brown-out Reset (BOR)

✓ Dual analog comparators:

- Programmable input/output configuration

5.4.2 Pin Configuration



5.4.3 Advantages

- ✓ 6509 is a low-cost microprocessor capable of solving a broad range of small-systems and memory management problems at minimum cost to the user.
- ✓ A memory management system allows for up to One Mega-Byte of memory for ease in downloading languages, operating systems, or other data.

5.5 INTRODUCTION TO EMBEDDED SYSTEMS

An embedded system is a microprocessor- or microcontroller-based system of hardware and software designed to perform dedicated functions within a larger mechanical or electrical system.

Complexities range from a single microcontroller to a suite of processors with connected peripherals and networks, from no user interface to complex graphical user interfaces. The complexity of an embedded system varies significantly depending on the task for which it is designed.

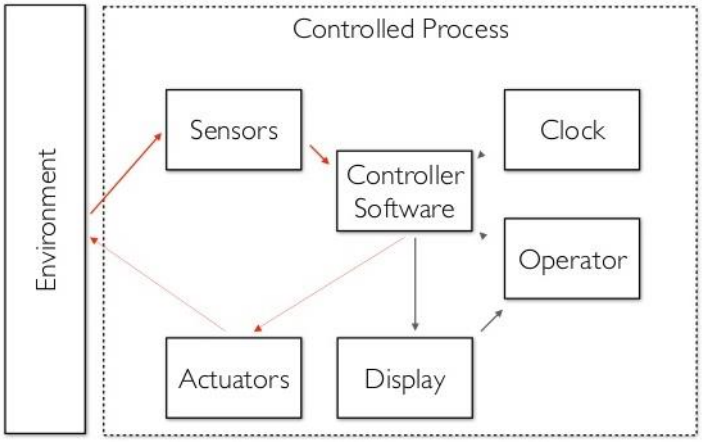
Embedded system applications range from digital watches and microwaves to hybrid vehicles and avionics. As much as 98 percent of all microprocessors manufactured are used in embedded systems.

Embedded systems are managed by microcontrollers or digital signal processors (DSP), application-specific integrated circuits (ASIC), field-programmable gate arrays (FPGA), GPU technology, and gate arrays. These processing systems are integrated with components dedicated to handling electric and/or mechanical interfacing.

Embedded systems programming instructions, referred to as firmware, are stored in read-only memory or flash memory chips, running with limited computer hardware resources. Embedded systems connect with the outside world through peripherals, linking input and output devices.

5.5.1Basic structure of an embedded system

The basic structure of an embedded system includes the following components:



Sensor: The sensor measures and converts the physical quantity to an electrical signal, which can then be read by an embedded system engineer or any electronic instrument. A sensor stores the measured quantity to the memory.

A-D Converter: An analog-to-digital converter converts the analog signal sent by the sensor into a digital signal.

Processor & ASICs: Processors assess the data to measure the output and store it to the memory.

D-A Converter: A digital-to-analog converter changes the digital data fed by the processor to analog data

Actuator: An actuator compares the output given by the D-A Converter to the actual output stored and stores the approved output.

5.5.2 Advantages

- ✓ Easily Customizable
- ✓ Low power consumption
- ✓ Low cost
- ✓ Enhanced performance

5.5.3 Disadvantages

- ✓ High development effort
- ✓ Larger time to market

5.5.4 Applications

- ✓ Smart homes – Home security system, Digital camera, Microwave oven
- ✓ Office – Router, modem
- ✓ Industrial automation

5.5.5Future trends

The industry for embedded systems is expected to continue growing rapidly, driven by the continued development of Artificial Intelligence (AI), Virtual Reality (VR) and Augmented Reality (AR), machine learning, deep learning, and the Internet of Things (IoT). The cognitive embedded system will be at the heart of such trends as: reduced energy consumption, improved security for embedded devices, cloud connectivity and mesh networking, deep learning applications, and visualization tools with real time data.

*****