SRI G.C.S.R COLLEGE

(Affiliated to Dr. B. R. Ambedkar University, Srikakulam)

GMR Nagar, Rajam – 532127, Srikakulam (Dist.), A.P

T: +91 8941-251336, M: +91 89785 23866, F: +91 8941-251591,

www.srigcsrcollege.org

Department of ELECTRONICS

II B.Sc. IV – Semester

Electronics Paper-5

Microcontrollers & Interfacing

(Study Material)

Name of the Student : _____

Roll Number : _____

Group : _____

Year/ Semester : _____

Prepared by:

K. Venugopala Rao M.Sc.(Tech), M. Tech

Lecturer in Electronics

SRI GCSR COLLEGE, GMR Nagar, Rajam

venugopalarao.k@sgcsrc.edu.in

+91-9603803168, +91-9182061933

Dr. B. R. AMBEDKAR UNIVE RSITY-SRIKAKULAM
B.Sc. ELECTRONICS SYLLABUS
STRUCTURE UNDER CHOICE BASED CREDITS SYSTEM
REVIEWED SYLLUBUS w.e.f. 2021-22

II B.Sc. Semester – IV

Paper – V: Microcontrollers & Interfacing

UNIT – I: INTRODUCTION TO MICROCONTROLLERS

Introduction, comparison of Microprocessor and microcontroller, Evolution of microcontrollers from 4-bit to 32-bit, Development tools for microcontrollers, Assembler-Compiler-Simulator/Debugger.

UNIT – II: MICROCONTROLLER ARCHITECTURE

Overview and block diagram of 8051, Architecture of 8051, program counter and memory organization, Data types and directives, PSW register, Register banks and stack, pin diagram of 8051, Interrupts and timers.

UNIT – III: ADDRESSING MODES, INSTRUCTION SET OF 8051

Addressing modes and accessing memory using various addressing modes, instruction set: Arithmetic, Logical, Simple bit, jump, loop and call instructions.

UNIT – IV: ASSEMBLE LANGUAGE PROGRAMMING

Addition, Multiplication, Subtraction, division, arranging a given set of numbers in largest/smallest order, Timer/Counter Programming

UNIT – V: INTERFACING AND APPLICATION OF MICROCONTROLLER

Interfacing of – PPI 8255, interfacing seven segment displays, displaying information on an LCD, control of a stepper Motor (Uni-Polar).

MICROCONTROLLER AND INTERFACING

UNIT I

Microcontroller Architecture

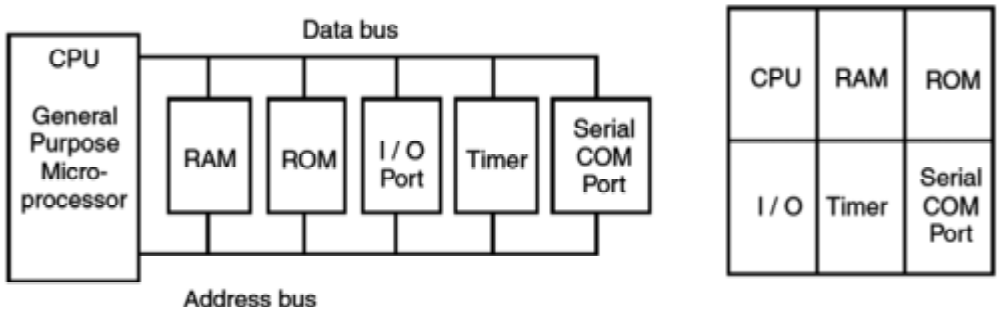
Introduction

A microcontroller is a computer present in a single integrated circuit which is dedicated to perform one task and execute one specific application.

It contains memory, programmable input/output peripherals as well a processor. Microcontrollers are mostly designed for embedded applications and are heavily used in automatically controlled electronic devices such as cell phones, cameras, microwave ovens, washing machines, etc.

Microcontrollers and Microprocessors

A microprocessor is an integrated circuit (IC) and is a computer's central processing unit (CPU on a chip). It is a programmable multipurpose silicon chip, clock driven, register based, accepts binary data as input and provides output after processing it as per the instructions stored in the memory. It consists of Arithmetical and Logical Unit (ALU), Control Unit and Register Array.



General-purpose Microprocessor System

Microcontroller

Microcontroller (MC) may be called computer on chip since it has basic features of microprocessor with internal ROM, RAM, Parallel and serial ports within single chip. Or we can say microprocessor with memory and ports is called as microcontroller. This is widely used in washing machines, vcd player, microwave oven, robotics or in industries.

The 8051 is the first microcontroller of the MCS-51 family introduced by Intel Corporation in the 1981. The 8051 family with its many enhanced members enjoys the largest market share, among the various microcontroller architectures like PIC, AVR, ARM, etc.

Microcontroller can be classified on the basis of their bits processed like 8bit MC, 16bit MC, 32bit MC. 8 bit microcontroller, means it can read, write and process 8 bit data. Ex: 8051 microcontroller. Basically 8 bit specifies the size of data bus. 8 bit microcontroller means 8 bit data can travel on the data bus or we can read, write process 8 bit data.

Microprocessors and Microcontrollers

MICROPROCESSORS	MICROCONTROLLERS
Microprocessor contains ALU, General purpose registers, stack pointer, program counter, clock timing circuit, interrupt circuit	Microcontroller contains the circuitry of microprocessor, and in addition it has built in ROM, RAM, I/O Devices, Timers/Counters etc.
It has many instructions to move data between memory and CPU	It has few instructions to move data between memory and CPU
Few bit handling instruction	It has many bit handling instructions
Less number of pins are multifunctional	More number of pins are multifunctional
Single memory map for data and code (program)	Separate memory map for data and code (program)
Access time for memory and IO are more	Less access time for built in memory and IO.
Microprocessor based system requires additional hardware	It requires less additional hardware
More flexible in the design point of view	Less flexible since the additional circuits which is residing inside the microcontroller is fixed for a particular microcontroller
Large number of instructions with flexible addressing modes	Limited number of instructions with few addressing modes

Evolution of Microcontroller

The last decade has seen an exciting evolution with capabilities of microprocessors. The development of 16 and 32 bit microprocessors contributed to the growth of powerful personal computers. In the evolution of microprocessor capability, instead of focusing upon larger word widths and address spaces. The present emphasis is upon exceedingly fast real time control. The development of microcontrollers has focused upon the integration of the facilities needed to support fast control into a single chip.

Intel has introduced standard 8-bit microcontroller 8048 in 1976. In the year 1980, Intel has introduced the 8051 microcontroller, with higher performance than 8048. With the advantages of 8051, the microcontroller applications took a peak level.

Because of the advanced semiconductor technology, it has become possible to integrate more than 1,00,000 transistors onto a single silicon chip. With this Intel developed a new generation of single chip 16 bit microcontrollers called the MCS-96 (8096 family).

The Motorola Microcontroller family was first introduced to the market in 1978 and is built in the same pattern of the microprocessor 6800.

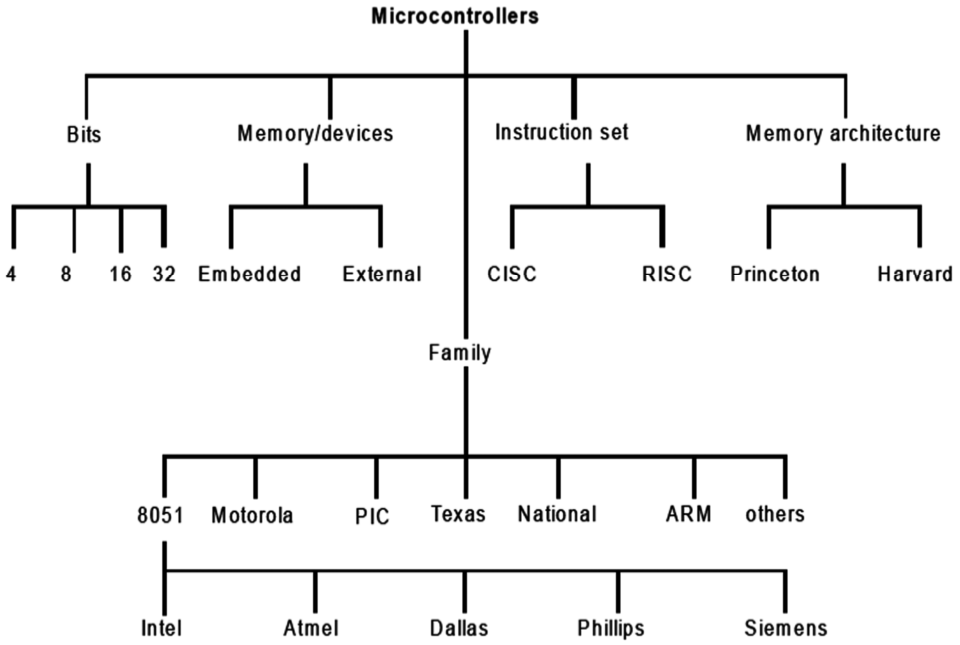
Criteria for selection of a microcontroller in embedded system

Criteria for selection of microcontroller in any embedded system is as following

1. Meeting the computing needs of task at hand efficiently and cost effectively
    - Speed of operation
    - Packing
    - Power consumption
    - Amount of RAM and ROM on chip
    - No. of I/O pins and timers on chip
    - Cost
2. Availability of software development tools such as compiler, assembler and debugger.

Types of microcontrollers

Microcontrollers can be classified on the basis of internal bus width, architecture, memory and instruction set. The various types of microcontrollers are shown below.



The 8, 16 and 32-bit microcontrollers

8-bit microcontroller

When the ALU performs arithmetic and logical operations on a byte (8-bits), the microcontroller is an 8-bit microcontroller. The internal bus width of 8-bit microcontroller is of 8-bit. Examples of 8-bit microcontrollers are Intel 8051 family and Motorola MC68HC11 family.

16-bit microcontroller

When the ALU performs arithmetic and logical operations on a word (16-bits), the microcontroller is an 16-bit microcontroller. The internal bus width of 16-bit microcontroller is of 16-bit. Examples of 16-bit microcontrollers are Intel 8096 family

and Motorola MC68HC12 and MC68332 families. The performance and computing capability of 16 bit microcontrollers are enhanced as compared to the 8-bit microcontrollers.

32-bit microcontroller

When the ALU performs arithmetic and logical operations on a double word (32- bits), the microcontroller is an 32-bit microcontroller. The internal bus width of 32-bit microcontroller is of 32-bit. Examples of 32-bit microcontrollers are Intel 80960 family and Motorola M683xx and Intel/Atmel 251 family. The performance and computing capability of 32 bit microcontrollers are enhanced as compared to the 16-bit microcontrollers.

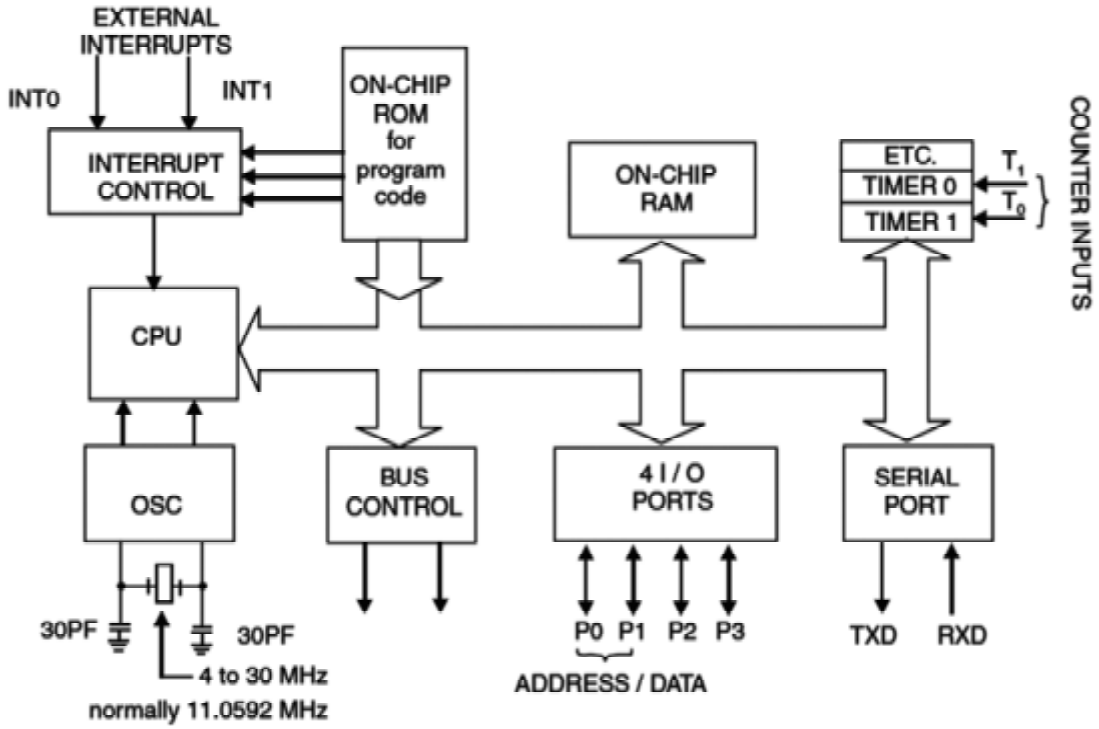
Various features of 8051 microcontroller are given as follows.

- 8-bit CPU
- 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

Some of the microcontrollers of 8051 family are given as follows:

DEVICE	ON-CHIP DATA MEMORY (RAM) (bytes)	ON-CHIP PROGRAM MEMORY (ROM) (bytes)	16-BIT TIMER/COUNTER	NO. OF VECTORED INTERUPTS	Serial I/O
8031	128	None	2	5	1
8032	256	none	2	6	1
8051	128	4k ROM	2	5	1
8052	256	8k ROM	3	6	1
8751	128	4k EPROM	2	5	1
8752	256	8k EPROM	3	6	1
AT89C51	128	4k Flash Memory	2	5	1
AT89C52	256	8k Flash memory	3	6	1

Block diagram of 8051



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:

128 Byte RAM for Data Storage

MC 8051 has 128 byte Random Access memory for data storage. During execution for storing the data the RAM is used. RAM consists of the register banks, stack for temporary data storage. It also consists of some special function register (SFR) which are used for some specific purpose like timer, input output ports etc. Normally microcontroller has 256 byte RAM in which 128 byte (address range 00H to 7FH) is used for user space which is normally Register banks and stack. But other 128 byte (address range 80H to FFH) RAM which consists of SFRs.

4KB ROM

In 8051, 4KB read only memory (ROM) is available for program storage, addressed from 0000H to 0FFFH

Timers and Counters

Timer is which can give the delay of particular time between some events. In MC8051, two 16 bit timers are available, named as T0 (TH0, TL0) and T1 (TH1, TL1),by these timers we can give the delay of particular time. TMOD, TCON registers are used for controlling timer operation.

Serial Port

There are two pins available for serial communication TXD and RXD.

Normally TXD is used for transmitting serial data which is in SBUF register, RXD is used for receiving the serial data.

SCON register is used for controlling the operation.

Input Output Ports

There are four input output ports available P0, P1, P2, P3. Each port is 8 bit wide.

The port 0 can perform dual works. It is also used as Lower order address bus (A0 to A7) multiplexed with 8 bit data bus P0.0 to P0.7 is AD0 to AD7 respectively the address bus and data bus is demultiplex by the ALE signal.

Port 2 can be used as I/O port as well as higher order address bus A8 to A15.

Port 3 also have dual functions it can be worked as I/O as well as each pin of P3 has specific function.

P3.0 – RXD – Serial data IO

P3.1 – TXD – Serial data transmit

P3.2 – INT0 – External Interrupt 0

P3.3 – INT1 – External Interrupt 1

P3.4 – T0 – Clock input for counter 0

P3.5 – T1 – Clock input for counter 1

P3.6 – WR – Signal for writing to external memory

P3.7 – RD – Signal for reading from external memory

When external memory is interfaced with 8051 then P0 and P2 can't be worked as I/O port they works as address bus and data bus, otherwise they can be accessed as I/O ports.

Oscillator

It is used for providing the clock to MC8051 which decides the speed or baud rate of MC.

We use crystal which frequency vary from 4MHz to 30 MHz, normally we use 11.0592 MHz frequency.

Interrupts

Interrupts are defined as requests because they can be refused (masked) if they are not used, that is when an interrupt is acknowledged. A special set of events or routines are followed to handle the interrupts. These special routines are known as or interrupt service routines (ISR).

INT0 and INT1 are the pins for external interrupts.

THE 8051 ARCHITECTURE

The 8051 MC is available in n-channel metal oxide silicon (N-MOS) and Complementary metal oxide silicon (C-MOS) construction in a variety package types. It is housed in a 40 pin DIP. It requires 5V DC for its operation.

Salient features of 8051 microcontroller are given below.

- 8 bit CPU
- 4Kbytes of internal program memory (code memory) [ROM]
- 128 bytes of internal data memory [RAM]
- 64 Kbytes of external program memory address space.
- 64 Kbytes of external data memory address space.
- 32 bi directional I/O lines (can be used as four 8 bit ports or 32 individually addressable I/O lines)
- Two 16 Bit Timer/Counter :T0, T1
- Full Duplex serial data receiver/transmitter
- Four Register banks with 8 registers in each bank.
- Sixteen bit Program counter (PC) and a data pointer (DPTR)
- 8 Bit Program Status Word (PSW)
- 8 Bit Stack Pointer
- Five vector interrupt structure (RESET not considered as an interrupt.)
- 8051 CPU consists of 8 bit ALU with associated registers like accumulator ‘A’ , B register, PSW, SP, 16 bit program counter, stack pointer.
- 8051 has 128 bytes of internal RAM which is divided into

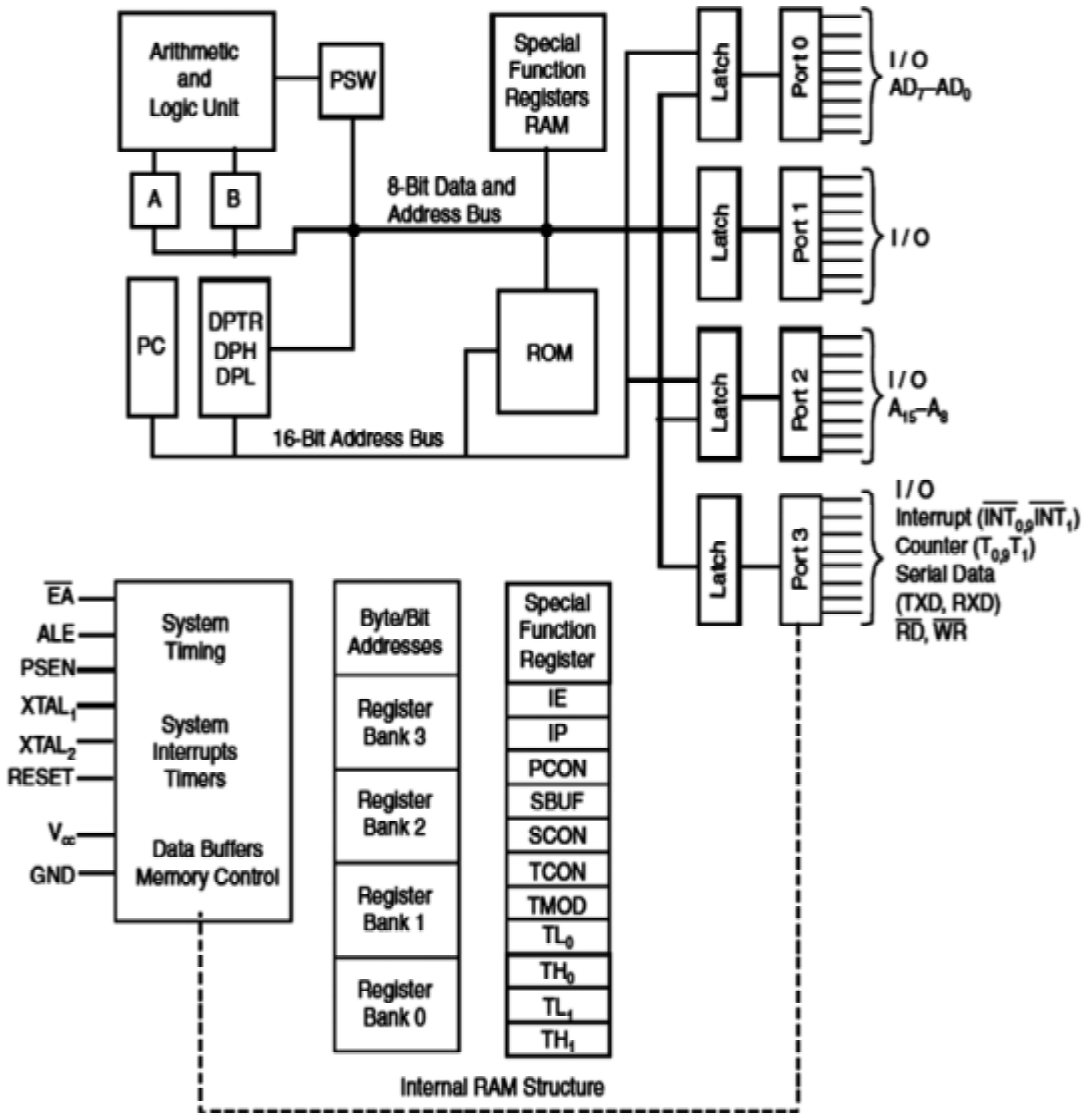
  Working registers [00 – 1F]

  Bit addressable memory area [20 – 2F]

  General purpose memory area (Scratch pad memory) [30-7F]

8051 has 4 K Bytes of internal ROM. The address space is from 0000 to 0FFFh. If the program size is more than 4 K Bytes 8051 will fetch the code automatically from external memory.

Accumulator is an 8 bit register widely used for all arithmetic and logical operations. Accumulator is also used to transfer data between external memory. B register is used along with Accumulator for multiplication and division. A and B registers together is also called MATH registers.



PSW (Program Status Word) - This is an 8 bit register which contains the arithmetic status of ALU and the bank select bits of register banks.

(MSB) PSW.7	PSW.6	PSW.5	PSW.4	PSW.3	PSW.2	PSW.1	(LSB) PSW.0
CY	AC	F0	RS1	RS0	OV	—	P
D7	D6	D5	D4	D3	D2	D1	D0

CY - carry flag

AC - auxiliary carry flag

F0 - available to the user for general purpose

RS1, RS0 - register bank select bits

OV - overflow

P - parity

Stack Pointer (SP)

It contains the address of the data item on the top of the stack. Stack may reside anywhere on the internal RAM. On reset, SP is initialized to 07 so that the default stack will start from address 08 onwards.

Data Pointer (DPTR)

DPH (Data pointer higher byte), DPL (Data pointer lower byte). This is a 16 bit register which is used to furnish address information for internal and external program memory and for external data memory.

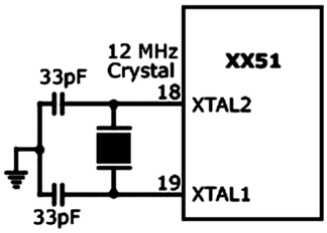
Program Counter (PC)

16 bit PC contains the address of next instruction to be executed. On reset PC will set to 0000. After fetching every instruction PC will increment by one.

System Clock

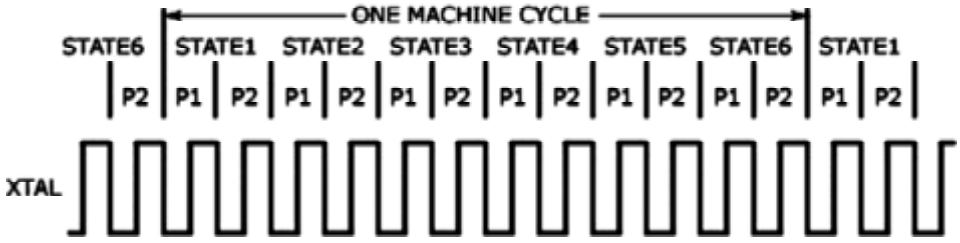
The heart of 8051 is the circuitry that generates the clock pulses by which all internal operations are synchronised. Pins XTAL1 and XTAL2 are provided for connecting an oscillator. The crystal frequency is the basic internal frequency of the microcontroller. 8051 is designed to operate between 1MHz to 16MHz and generally operates with a crystal frequency 11.04962 MHz.

8051 Clock Circuit



The time taken to complete any instruction is called as machine cycle or instruction cycle. In 8051 one instruction cycle consists of 6 states or 12 clock cycles. Two pulses forms a state and six states forms one machine cycle.

System Clock



To calculate time for any instruction will be executed.

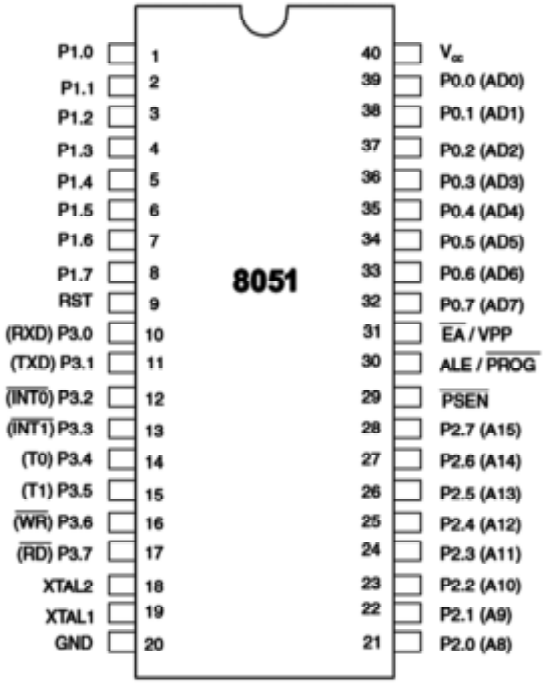
$$T_{inst} = \frac{C\,X12d}{crystal\,frequency}$$

Where, C is the no. of machine cycles.

Ex: for example the crystal frequency is 16 MHz, the time to execute an ADD a, R1 one cycle inst.

$$T_{inst} = \frac{1X12}{16\,X10^{6}} = 0.75\,\mu\,\text{sec}$$

Pin diagram of 8051



Pin diagram of MC 8051

Pins 1-8 : **PORT 1**. Each of these pins can be configured as an input or an output.

Pin 9 : **RESET**. A logic one on this pin disables the microcontroller and clears the contents of most registers. In other words, the positive voltage on this pin resets the microcontroller. By applying logic zero to this pin, the program starts execution from the beginning.

Pins10-17 : **PORT 3**. Similar to port 1, each of these pins can serve as general input or output. Besides, all of them have alternative functions

Pin 10 : **RXD.** Serial asynchronous communication input or Serial synchronous communication output.

Pin 11 : **TXD.** Serial asynchronous communication output or Serial synchronous communication clock output.

Pin 12 : **INT0.** External Interrupt 0 input

Pin 13 : **INT1.** External Interrupt 1 input

Pin 14 : **T0.** Counter 0 clock input

Pin 15 : **T1.** Counter 1 clock input

Pin 16 : **WR.** Write to external (additional) RAM

Pin 17 : **RD.** Read from external RAM

Pin 18, 19 : **XTAL2, XTAL1.** Internal oscillator input and output. A quartz crystal which specifies operating frequency is usually connected to these pins.

Pin 20 : **GND.** Ground.

Pin 21-28 : **Port 2.** If there is no intention to use external memory then these port pins are configured as general inputs/outputs. In case external memory is used, the higher address byte, i.e. addresses A8-A15 will appear on this port. Even though memory with capacity of 64Kb is not used, which means that not all eight port bits are used for its addressing, the rest of them are not available as inputs/outputs.

Pin 29 : **PSEN.** If external ROM is used for storing program then a logic zero (0) appears on it every time the microcontroller reads a byte from memory.

Pin 30 : **ALE.** Prior to reading from external memory, the microcontroller puts the lower address byte (A0-A7) on P0 and activates the ALE output. After receiving signal from the ALE pin, the external latch latches the state of P0 and uses it as a memory chip address. Immediately after that, the ALE pin is returned its previous logic state and P0 is now used as a Data Bus.

Pin 31 : **EA.** By applying logic zero to this pin, P2 and P3 are used for data and address transmission with no regard to whether there is internal memory or not. It means that even there is a program written to the microcontroller, it will not be executed. Instead, the program written to external ROM will be executed. By applying logic one to the EA pin, the microcontroller will use both memories, first internal then external (if exists).

Pin 32-39 : **PORT 0.** Similar to P2, if external memory is not used, these pins can be used as general inputs/outputs. Otherwise, P0 is configured as address output (A0-A7) when the ALE pin is driven high (1) or as data output (Data Bus) when the ALE pin is driven low (0).

Pin 40 : **VCC.** +5V power supply.

SFR Registers

Accumulator

This is an 8 bit register and is used to the hold result of various arithmetic and logic operations.

The accumulator, referred to as ACC or A.

MOV A, #52h - Move immediate the value 52h to the accumulator

MOV 70h, #52h - Move immediate the value 52h to Internal RAM location 70h.

B Register

The B register is an SFR register at addresses F0h which is bit-addressable. The B register is used in two instructions only: i.e. MUL (multiply) and DIV (divide). The B register can also be used as a general-purpose register.

Program Counter

The PC (Program Counter) is a 2 byte (16 bit) register which always contains the memory address of the next instruction to be executed. When the 8051 is reset the PC is always initialised to 0000h.

SFR Registers for the Internal Timer

TCON - The Timer Control register is an SFR at address 88h, which is bit-addressable. TCON is used to configure and monitor the 8051 timers. The TCON SFR also contains some interrupt control bits.

TMOD - The Timer Mode register is an SFR at address 89h and is used to define the operational modes for the timers.

Timer 0 (T0) - TL0 (Timer 0 Low) and TH0 (Timer 0 High) are two SFR registers addressed at 8Ah and 8Bh respectively. The two registers are associated with Timer 0.

Timer 1 (T1) - TL1 (Timer 1 Low) and TH1 (Timer 1 High) are two SFR registers addressed at 8Ch and 8Dh respectively. These two registers are associated with Timer 1.

Power Control Register

PCON (Power Control) - PCON register is an SFR at address 87h. It contains various control bits including a control bit, which allows the 8051 to go to 'sleep' so as to save power when not in immediate use.

Serial Port Registers

SCON (Serial Control) - The SCON (Serial Control) is an SFR register located at addresses 98h, and it is bitaddressable. SCON configures the behaviour of the on-chip serial port, setting up parameters such as the baud rate of the serial port, activating send and/or receive data, and setting up some specific control flags.

SBUF (Serial Buffer) - The SBUF (Serial Buffer) is an SFR register located at address 99h. SBUF is just a single byte deep buffer used for sending and receiving data via the on-chip serial port

Interrupt Registers

IE (Interrupt Enable) - IE (Interrupt Enable) is an SFR register at addresses A8h and is used to enable and disable specific interrupts. The MSB bit (bit 7) is used to disable all interrupts.

IP (Interrupt Priority) - IP (Interrupt Priority) is an SFR register at addresses B8h and it is bit addressable. The IP register specifies the relative priority (high or low priority) of each interrupt.

On the 8051, an interrupt may either be of low (0) priority or high (1) priority. .

Program Counter and Data Pointer

The 8051 contains two 16-bit registers, the program counter (PC) and the data pointer (DPTR). Each is used to hold the address of a byte in memory.

Program Counter

The PC (Program Counter) is a 16 bit register which always contains the memory address of the next instruction to be executed. When the 8051 is reset the PC is always initialised to 0000h. If a instruction is executed the PC is incremented to point to the next instruction to be executed. A jump instruction (e.g. LJMP) has the effect of causing the program to branch to a newly specified location, so the jump instruction causes the PC contents to change to the new address value.

Data Pointer

The Data Pointer, DPTR, is a special 16-bit register used to address the external code or external data memory. Since the SFR registers are just 8-bits wide the DPTR is stored in two SFR registers, where DPL (82h) holds the low byte of the DPTR and DPH (83h) holds the high byte of the DPTR.

For example, for write the value 46h to external data memory location 2500h.

The following instructions are needed.

MOV A, #46h ; Move immediate 8 bit data 46h to A (accumulator)

MOV DPTR, #2500h ; Move immediate 16 bit address value 2500h to DPTR.

Now DPL holds 04h and DPH holds25h.

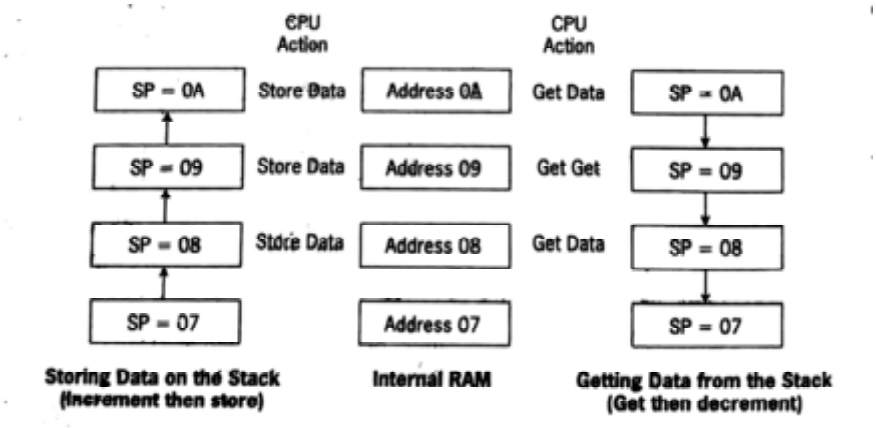
MOVX @DPTR, A ; Move the value in A to external RAM location 2500h.

Note: The **MOVX** (Move X) instruction is used to access external data memory.

The Stack and the Stack Pointer

The stack refers to an area of internal RAM that is used in conjunction with certain opcode to store and retrieve data quickly. The 8-bit Stack Pointer (SP) register is used by the 8051 to hold an internal RAM address that is called the top of the stack. The address held in the SP register is the location in internal RAM where the last byte of data was stored by a stack operation.

When data is to be placed on the stack, the SP is increments before storing data on the stack so that the stack grows up as data is stored. As data is retrieved from the stack, the byte is read from the stack, and then the SP decrements to point to the next available byte of stored data.

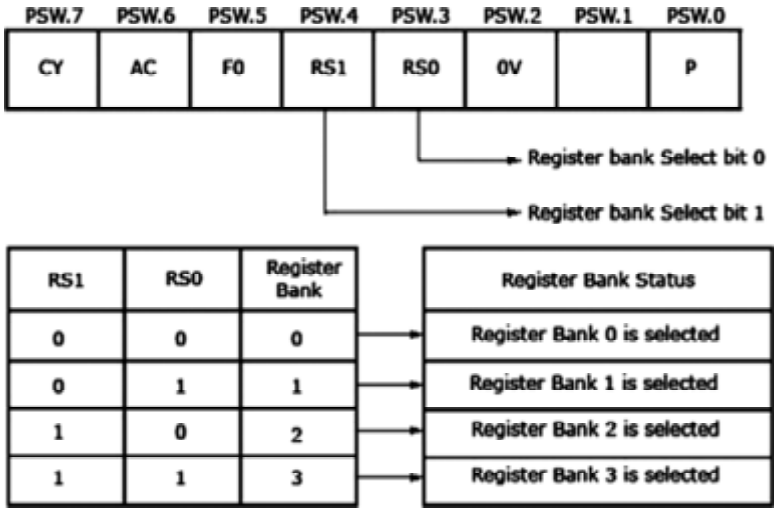


Operation of the stack and the SP is shown in Figure. The SP is set to 07h when the 8051 is reset and can be changed to any internal RAM address by the programmer. The stack is limited in height to the size of the internal RAM. The stack has the potential to over write valuable data in the register banks, bit-addressable RAM, and scratch pad RAM areas. The programmer is responsible for making sure the stack does not grow beyond predefined bounds!

Flags and the Program Status Word (PSW)

The 8051 has four math flags that respond automatically to the outcomes of math operations and three general-purpose user flags that can be set to 1 or cleared to 0 by the programmer as desired. The math flags include Carry (C), Auxiliary Carry (AC), Overflow (OV), and Parity (P). User flags are named FO, GFO, and GF1, they are general-purpose flags that may be used by the programmer to record some event in the program. Note that all of the flags can be set and cleared by the programmer.

The program status word is shown in Figure. The PSW contains the math flags, user program flag FO, and the register select bits that identify which of the four general-purpose register banks is currently in use by the program. The remaining two user flags, GF0 and GF1, are stored in PCON.



CY, the carry flag

- This flag is set whenever there is a carry out from the D7 bit.
- This flag bit is affected after an 8-bit addition or subtraction.
- It can also be set to 1 or 0 directly by an instruction such as "**SETB C" and "CLR.**

AC, the auxiliary carry flag

- If there is a carry from **D3 to D4 during an ADD or SUB** operation, this bit is set; otherwise, it is cleared.
- This flag is used by instructions that perform BCD (binary coded decimal) arithmetic.

P, the parity flag

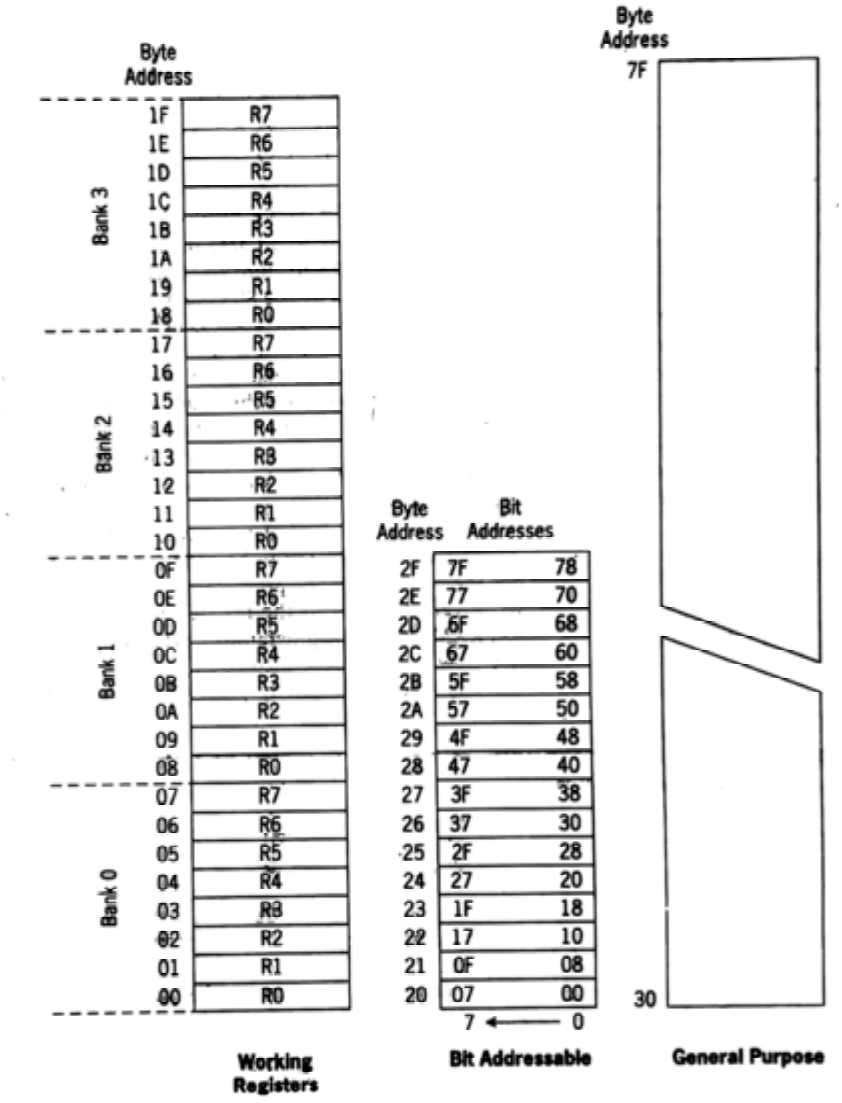
- The parity flag reflects the number of 1 s in the A (accumulator) register only.
- If the A register contains an odd number of Is, then P = 1. Therefore, P = 0 if A has an even number of 1s.

OV, the overflow flag

- This flag is set whenever the result of a signed number operation is too large, causing the high-order bit to overflow into the sign bit.
- In general, the carry flag is used to detect errors in unsigned arithmetic operations.
- The overflow flag is only used to detect errors in signed arithmetic operations

Internal RAM

The 128-byte internal RAM, which is shown detail in Figure and is organized into three distinct areas.



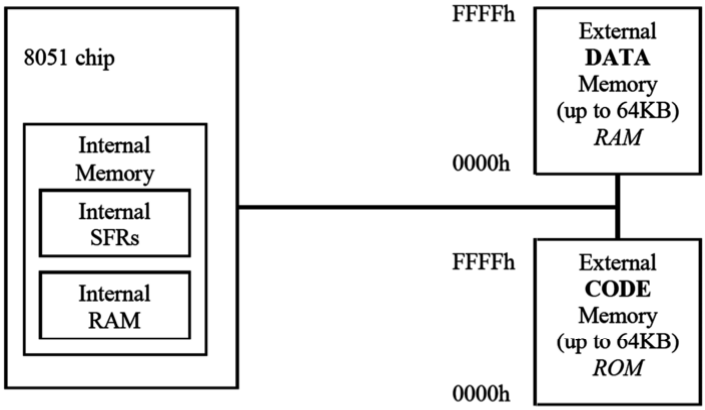
32 bytes from address 00h to 1Fh that make up 32 working registers organized as four banks of 8 registers each. The four register banks are numbered 0 to 3 and are made up of eight registers named R0 to R7. Each register can be addressed by name (when its bank is selected) or by its RAM address. Thus R0 of bank 3 is R0 (if bank 3 is currently selected) or address 18h. Bits RS0 and RS1 in the PSW determine which bank of registers is currently in use. Register banks not selected can be used as general-purpose RAM. Bank 0 is selected on reset.

A bit-addressable area of 16 bytes occupies RAM byte addresses 20h to 2Fh, forming a total of 128 addressable bits. An addressable bit may be specified by its bit address of 00h to 7Fh. for example, bit address 4Fh is a bit 7 of byte address 29h.

A general-purpose RAM area above the bit area, from 30h to 7Fh, addressable as bytes.

Memory and Register Organisation

The 8051 has a separate memory space for code (programs) and data. The on-chip memory and external memory is as shown in figure.



8051 Memory representation

External Code Memory

The executable program code is stored in this code memory. The code memory size is limited to 64KBytes (in a standard 8051). The code memory is read-only in normal operation and is programmed under special conditions e.g. it is a PROM or a Flash RAM type of memory.

External RAM Data Memory

This is read-write memory and is available for storage of data. Up to 64KBytes of external RAM data memory is supported (in a standard 8051).

Internal Memory

The 8051's on-chip memory consists of 256 memory bytes organised as follows:

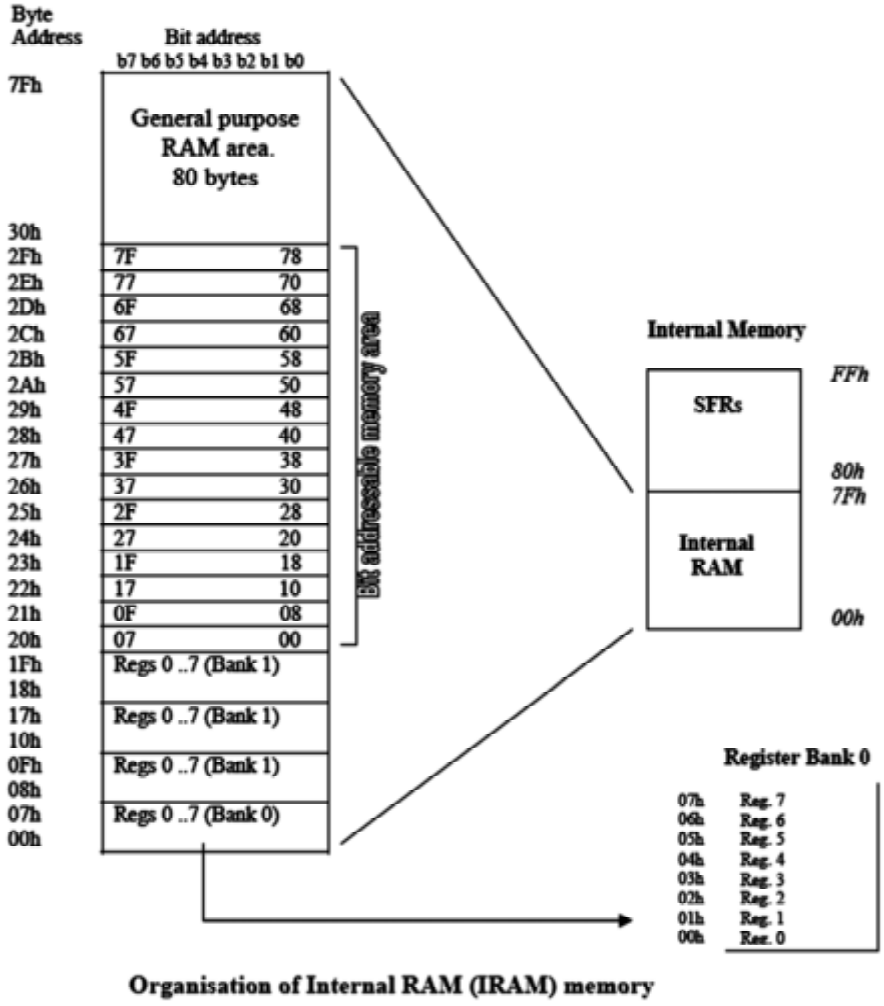
First 128 bytes:	00h to 1Fh Register Banks
	20h to 2Fh Bit Addressable RAM
	30 to 7Fh General Purpose RAM
Next 128 bytes:	80h to FFh Special Function Registers

SFR Registers

The SFR registers are located within the Internal Memory in the address range 80h to FFh. Not all locations within this range are defined. Each SFR has a very specific function. Each SFR has an address (within the range 80h to FFh). Some of the SFR registers are bit addressable. SFRs are accessed just like normal Internal RAM locations.

Internal memory

The first 128 bytes of internal memory is organised as shown in figure, and is referred to as Internal RAM.



Organisation of Internal RAM (IRAM) memory

Register banks RAM:

32 bytes from address 00h to 1Fh that make up 32 working registers organized as four banks of 8 registers each. The four register banks are numbered 0 to 3 and are made up of eight registers named R0 to R7.

The 8051 uses 8 general-purpose registers R0 through R7 (R0, R1, R2, R3, R4, R5, R6, and R7). These registers are used in instructions such as:

ADD A, R2 ; adds the value contained in R2 to the accumulator. i.e., it adds content of 01h location to accumulator.

The following instruction has the same effect as the above instruction.

ADD A, 02h

The selection of register bank depends upon RS1 and RS0 which are in PSW.

Bit Addressable RAM: 20h to 2Fh

The 8051 supports a special feature which allows access to bit variables. That means the individual memory bits in Internal RAM can be set or cleared. The Bit Addressable area of the RAM is just 16 bytes of Internal RAM located between 20h and 2Fh. In all there are

128 bits numbered 00h to 7Fh. A bit variable can be set with a command such as SETB and cleared with a command such as CLR.

SETB 25h ; sets the bit 25h (becomes 1)

CLR 25h ; clears bit 25h (becomes 0)

Note, bit 25h is actually bit b5 of Internal RAM location 24h.

General Purpose RAM: 30h to 7Fh

These 80 bytes of Internal RAM memory are available for general-purpose data storage.

External Memory interfacing

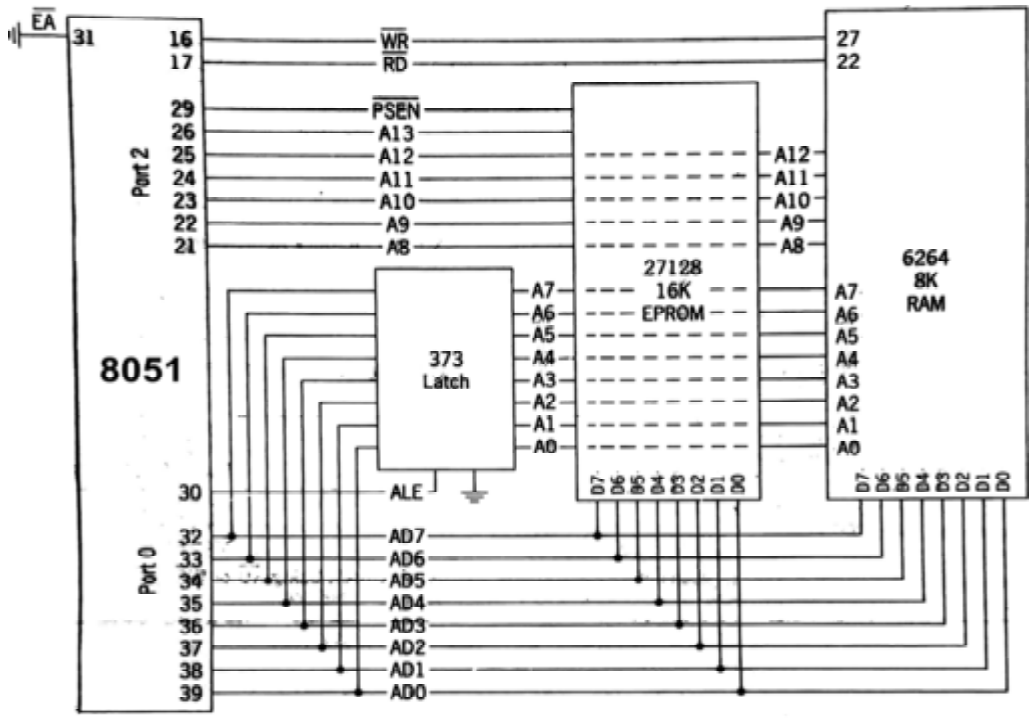


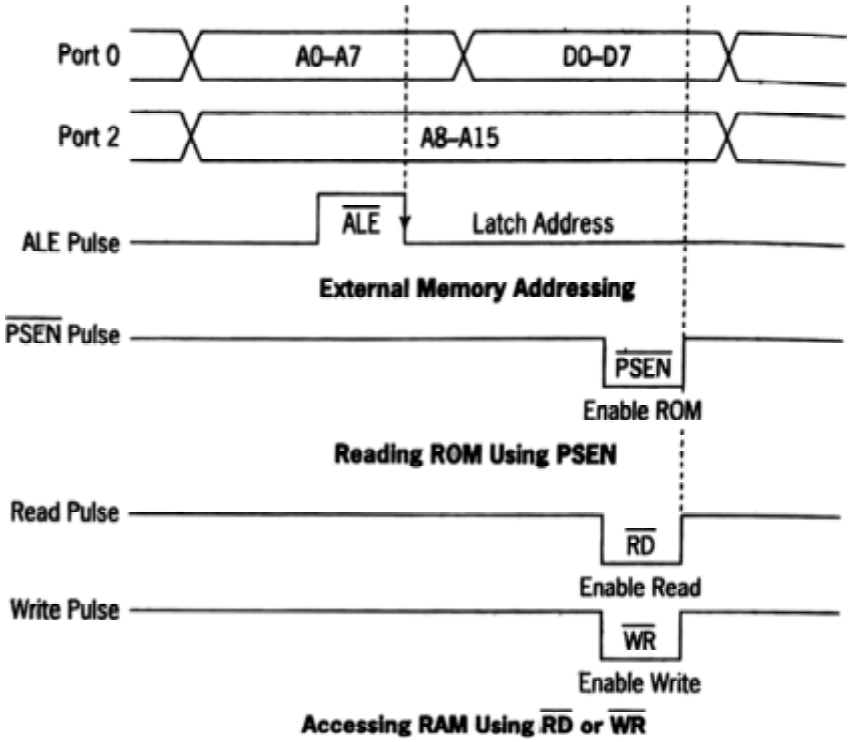
Figure 3.8 shows the connections between an 8051 and an external memory configuration consisting of 16K of EPROM and 8K of RAM. The 8051 accesses external RAM whenever certain program instructions are executed. External ROM is accessed whenever the $\overline{EA}$ (external access) pin is connected to ground or when the PC contains an address higher than the last address in the internal 4K ROM (0FFFh). 8051 designs can thus use internal and external ROM automatically.

The lower address byte from port 0 must be latched into an external register to save the byte with help of ALE clock pulse, that provides the correct timing for the 373 data latch.

If the $\overline{PSEN}$ (program store enable) pin is low, the code byte in the ROM is placed on the data bus. If the access is for a RAM byte, the WR (write)or RD (read) pins will go low, enabling data to flow between the RAM and the data bus.

The ROM may be expanded to 64K by connecting the remaining port 2 upper address lines A14-A15 to the chip.

Fig shows external memory timing diagram.



8051 data type and directives

Data types

The 8051 microcontroller has only one data type. It is 8 bits, and the size of each register is also 8 bits. It is the job of the programmer to break down data larger than 8 bits (00 to FFH) to be processed by the CPU.

DB - Define a Byte - puts a byte (8-bit number) number constant at this memory location

DB 24 - It stores 18h in the memory location.

DB 24h - It stores 24h in the memory location.

DB 101101b - It stores 45 in the memory location.

DW - Define a Word - puts a word (16-bit number) number constant at this memory location

DBIT - Define a Bit, defines a bit constant, which is stored in the bit

Assembler directives

The following are some more widely used directives of the 8051.

ORG (origin)

It defines the starting address for the program in program (code) memory

EQU (equate)

This is used to define a constant without occupying a memory location. This equates a numerical value to symbol.

PR EQU 25h

END directive

The END directive is indicates to the assembler the end of the source (asm) file. The END directive is the last line of an 8051 program, meaning that in the source code anything after the END directive is ignored by the assembler. Some assemblers use "**.** END" instead of "END".

Counters / Timers

Many microcontroller applications require the counting of external events or internal time delays between actions. There are two I6-bit up counters, named T0 and TI, are provided for the general use of the programmer. Each counter may be programmed to count internal clock pulses, acting as a timer, or programmed to count external pulses as a counter.

The counters are divided into two 8-bit registers called the timer low (TLO, TL1) and high (THO, TH1) bytes. All counter action is controlled by bit states in the timer mode control register (TMOD), the timer/counter control register (TCON), and certain program instructions.

TMOD is dedicated solely to the two timers and considered as a two 4-bit registers for 2 timers. TCON has control bits and flags for the timers in the upper nibble, and control bits and flags for the external interrupts in the lower nibble.

Timer control (TCON) special function register

This is an 8bit SFR register shown in figure, upper 4 bits for timer 0 and 1 and lower 4bits are used to control interrupts. It is a bit addressable SFR as TCON.0 to TCON.7

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0
bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0

Timer control bits

TF1 - Timer 1 Overflow flag

Set when timer rolls from all 1s to 0s, Cleared when processor vectors to execute interrupt service routine located at program address 001Bh.

TR1- Timer 1 run control bit

Set to 1 by program to enable timer 1 to count, cleared to O by program to halt timer. Does not reset timer.

TF0 - Timer 0 Overflow flag

Set when timer rolls from all 1s to 0s, Cleared when processor vectors to execute interrupt service routine located at program address 000Bh.

TR0 - Timer 0 run control bit

Set to 1 by program to enable timer 0 to count; cleared to 0 by program to halt timer. Does not reset timer.

Interrupt control bits

IE1- External interrupt 1 Edge flag

Set to 1 when a high-to low edge signal is received on port 3 pin 3.3 (INT1), Cleared to 0 when processor vectors to interrupt service routine located at program address 0013h.

IT1 - External interrupt 1 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 interrupt 1 to generate an interrupt.

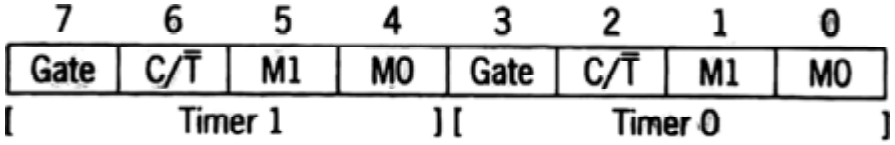
IE0 - External interrupt 0 Edge flag

Set to 1 when a high to low edge signal is received on port 3 pin 3.2 (INTO), Cleared when processor vectors to interrupt service routine located at program address 0003h.

IT0 - External interrupt 0 signal type control bit

Set to 1 by program to enable external interrupt 0 to be triggered by a falling edge signal. Set to 0 by program to enable a low-level signal on external interrupt O to generate an interrupt.

Timer mode control (TMOD) special function register



TMOD is not bit addressable

Gate (Gating control)

Set to 1 by program to enable timer to run if bit TR in TCON is set and signal on external interrupt INT pin is high. Cleared to 0 by program to enable timer to run if bit TR in TCON is set.

C/$\overline{T}$ (counter/ timer)

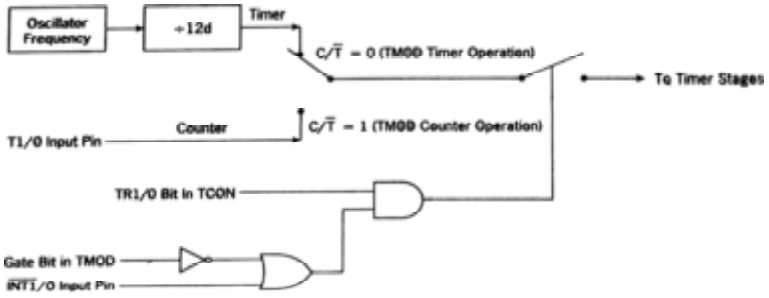
Set to 1 by program to make timer act as a counter by counting pulses from external input pin, Cleared to O by program to make timer act as a timer by counting internal frequency.

M1 M0 (Mode selection bits)

These bits are used to select the timer mode as shown in table.

M1	M0	mode
0	0	Mode 0
0	1	Mode 1
1	0	Mode 2
1	1	Mode 3

Timer/counter logic



The resultant timer clock is gated to the timer by means of the circuit shown in Figure. The C/$\overline{T}$ bit in the TMOD register must be set to 0. Bit TRX in the TCON register must be set to 1 (timer run), and the gate bit in the TMOD register must be 0, or external pin INTX must be a 1.

Timer Modes of operation

The timers may operate in any one of four modes that are determined by the mode bits, M1 and MO, in the TMOD register.

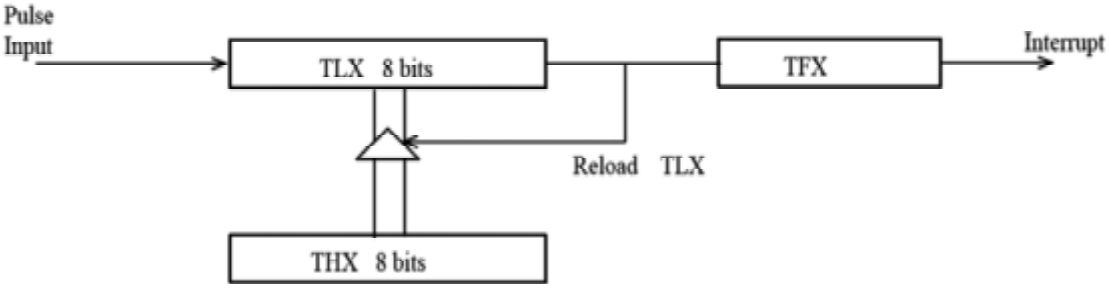
TIMER MODE 0 – 13 bit Timer/Counter

In the mode 0, the timer register is configured as a 13 bit register. The 13 bit register consists of all 8 bits of THX and the lower 5 bits of TLX. It holds value between 0000h to 1FFFh. The upper 3 bits of TLX are indetermined and should be ignored. When timer reaches its maximum value 1FFFh, it comes to initial value 0000h.

TIMER MODE 1 – 16 bit Timer/Counter

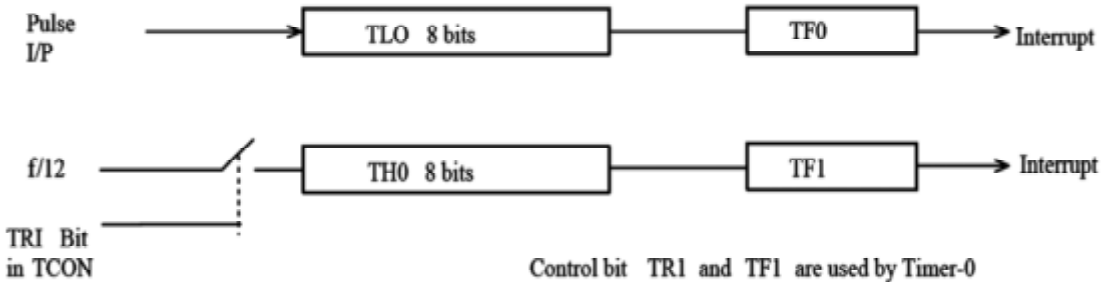
Mode 1 is similar to mode 0 except TLX is configured as a full 8-bit counter. Therefore it holds the value between 0000h to FFFFh. Once it is loaded, the timer must be started. When it moves from FFFFh to 0000h, it sets the timer flag (TF = 1)

TIMER MODE 2 – auto reload of TL from TH



It is an 8 bit timer, therefore it allows only values between 00h to FFh. After TH is loaded with 8bit value, the timer must be started. It counts up by incrementing the TL register. When it moves from FFh to 00h, it sets the timer flag. TL is reloaded automatically with the original value, which is in the TH register.

TIMER MODE 3 – two 8 bit timers using Timer 0



Timer 0 and 1 may be programmed to be in mode 0, 1 or 2 independently. This does not hold good for mode 3. Placing timer 1 in mode 3 causes it to stop counting. The control bit TR! And timer 1 flag bit TF1 are then used by timer 0. Timer0 in mode 3 becomes 2 completely separate 8bit counters, TL0 is controlled by gate arrangement and sets timer flag TF0 whenever it overflows. Tho receives timer clk1 under the control of TR1 and set the TF! When it over flows.

8051 INTERRUPTS

Interrupts and polling

A single microcontroller can serve several devices. There are two ways to do that: interrupts or polling. In the interrupt method, whenever any device needs its service, the device notifies the microcontroller by sending it an interrupt signal. Upon receiving an interrupt signal, the microcontroller interrupts whatever it is doing and serves the device.

In polling, the microcontroller continuously monitors the status of a given device; when the status condition is met, it performs the service. After that, it moves on to monitor the next device until each one is serviced. it is not an efficient use of the microcontroller. The advantage of interrupts is that the microcontroller can serve many devices (not all at the same time, of course)

Interrupts in the 8051

The 8051 has 5 interrupts which are available to user, but including RESET, they are 6.

- Reset. When the reset pin is high, the 8051 jumps to address location 0000.
- Two interrupts from timers, one for Timer 0 and one for Timer 1.
- Two interrupts from external i/ps (INT0 &INT1), Pin numbers 12 (P3.2) and 13 (P3.3) in port 3 are for the external hardware interrupts INTO and INT1, respectively. These external interrupts are also referred to as EX1 and EX2.
- One from Serial communication port.

For every interrupt, there must be an interrupt service routine (ISR). When an interrupt is invoked, the microcontroller runs the interrupt service routine. For every interrupt, there is a fixed location in memory that holds the address of its ISR, is called the interrupt vector table, shown below.

Interrupt Vector Table for the 8051

Interrupt	ROM Location (Hex)	Pin
Reset	0000	9
External interrupt 0 (INT 0)	0003	P3.2 (12)
Timer 0 interrupt (TF0)	000B	--
External interrupt 1 (INT 1)	0013	P3.3 (13)
Timer 1 interrupt (TF1)	0013	--
Serial COM interrupt (RI and TI)	0023	--

Enabling and disabling an Interrupt

Upon reset, all interrupts disabled. that is none will be responded to by the microcontroller even if they are in active. The interrupt must beenabled by the software, a SFR called IE (Interrupt Enable) is responsible for enabling (unmasking) and disabling (masking) the interrupts.

The Interrupt Enable (IE) Special Function Register

It is Bit addressable as IE.0 to IE.7

Set to 1 by program to enable interrupt; cleared to 0 to disable interrupt.

EA	—	ET2	ES	ET1	EX1	ET0	EX0
bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0

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

The Interrupt Priority (IP) Special Function Register

It is a Bit addressable as IP0 to IP7

Interrupt register IP determines, whether an interrupt has highest priority or lowest priority. Bit set to 1, indicates higher priority and bit cleared to 0, and indicates lower priority. Lower priority interrupt continues after the higher is finished.

If two interrupts with same priority occur at same time, then the following level can be used.

Interrupt	Priority
External interrupt 0 (IE0)	Highest
Timer/counter 0 overflow interrupt (TF0)	-
External interrupt 1 (IE1)	-
Timer/counter 1 overflow interrupt (TF1)	-
Serial port interrupt	Lowest

- Bit 7 - Not implemented.
- Bit 6 - Not implemented.
- Bit 5 - PT2 - Reserved for future use.
- Bit 4 – PS - Priority of serial port interrupt. 1= high, 0=low
- Bit 3 - PT1 - Priority of timer 1 overflow interrupt. 1= high, 0=low
- Bit 2 – PXI - Priority of external interrupt 1. 1= high, 0=low
- Bit 1 - PT0 - Priority of timer 0 overflow interrupt. 1= high, 0=low
- Bit 0 - PXO Priority of external interrupt 0. 1= high, 0=low

Steps in executing an interrupt

Upon activation of an interrupt, the microcontroller goes through the following steps.

1. It finishes the present executing instruction and saves the address of the next instruction (PC) on the stack.
2. It jumps to a fixed location in memory address of the interrupt service routine.
3. The microcontroller gets the address of the ISR from the interrupt vector table and jumps to it, It starts to execute the interrupt service subroutine until it reaches the last instruction of the subroutine, which is RETI (return from interrupt).
4. Upon executing the RETI instruction, the microcontroller returns to the place where it was interrupted. First it gets the program counter (PC) address from the stack, then it starts to execute from that address.

UNIT II

ADDRESSING MODES – INSTRUCTION SET

Addressing Modes

The way of data source or destination addresses (operands) are specified in the mnemonic, that means the way of data determines within the instruction is called addressing mode. There are a number of addressing modes available to the 8051 instruction set, as follows:

Immediate Addressing

Register Addressing

Direct Addressing

Indirect Addressing

Relative Addressing

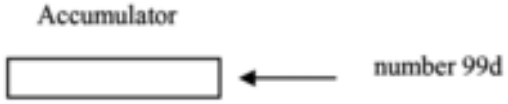
Long Addressing

Indexed Addressing

Immediate Addressing

Immediate addressing means that the operand is the data value to be used.

Ex: MOV A, #99d - Moves the value 99 into the accumulator.

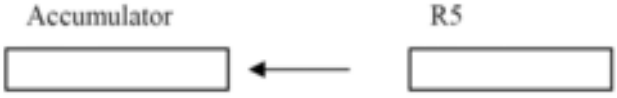


The # symbol tells the assembler that the immediate addressing mode is to be used.

Register Addressing

One of the eight general-registers, R0 to R7, can be specified as the instruction operand.

Ex: ADD A, R5 - the contents of R5 is added to the accumulator.

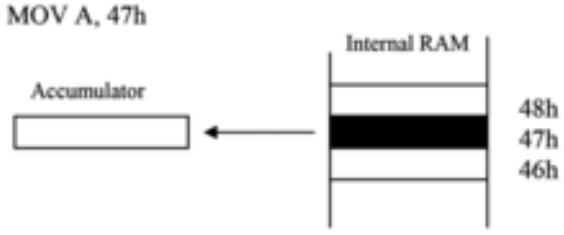


Direct Addressing

Direct addressing means that the operand specifies with RAM address

Ex: MOV A, 47h

The content of RAM address 47h is store in the accumulator.



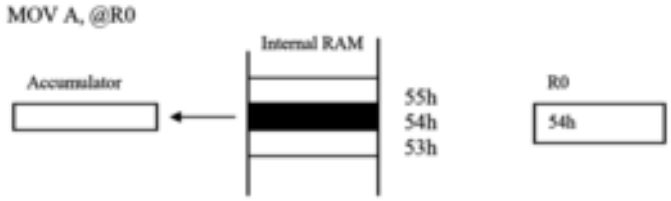
Indirect Addressing

Indirect addressing provides a powerful addressing capability, An example instruction, which uses indirect addressing, is as follows:

MOV A, @R0

Note the @ symbol indicated that the indirect addressing mode is used.

The content of address location which is stored in R0 is moved to the accumulator.



Relative Addressing

This is a special addressing mode used with certain jump instructions. The relative

address, is an 8-bit signed number, which is automatically added to the PC to make the address of the next instruction. The 8-bit signed offset value gives an address range of + 127 to –128 locations.

Ex: SJMP LABEL_X

Long Addressing

The long addressing mode within the 8051 is used with the instructions LJMP and

LCALL. The address specifies a full 16 bit destination address so that a jump or a call

can be made to a location within a 64KByte code memory space.

Ex: LJMP 5000h ; full 16 bit address is specified in operand

Indexed Addressing

With indexed addressing a separate register, either the program counter, PC, or the

data pointer DTPR, is used as a base address and the accumulator is used as an offset

address. The effective address is formed by adding the value from the base address to

the value from the offset address.

Ex: MOVC A, @A+DPTR

MOVC is a move instruction, which moves data from the external code memory

space. The address operand in this example is formed by adding the content of the

DPTR register to the accumulator value.

Number Representation for Different Bases

The following is an example showing the decimal number 46 represented in different number bases:

46d - 46 decimal

2Eh - 2E**h** is 46 decimal represented as a hex number

56o - 56**o** is 46 decimal represented as an octal number

101110b - 101110**b** is 46 decimal represented as a binary number.

Note: A number must be started with digit of a hexadecimal number. For example the hexadecimal number A5h is illegally represented and should be represented as 0A5h.

The Arithmetic Operators : The arithmetic operators are:

+	add
-	subtract
*	multiply
/	divide
MOD	modulo (result is the remainder following division)

The Logical Operators: The logical operators are:

AND	Logical AND
OR	Logical OR
XOR	Logical XOR (exclusive OR)
NOT	Logical NOT

Types of Instructions

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

Data transfer instructions

Arithmetic instructions

Logical instructions

Branch instructions

Subroutine instructions

Data Transfer instructions

Many computer operations are concerned with moving data from one location to another. The 8051 uses five different types of instruction to move data.

- MOV
- MOVX
- MOVC
- PUSH and POP
- XCH

MOV

The MOV instruction is concerned with moving data internally, i.e. between Internal RAM, SFR registers, general registers etc. MOVX and MOVC are used in accessing external memory data.

Syntax - MOV destination source

The instruction copies data from a defined source location to a destination location.

MOV R2, #80h - Move immediate data value 80h to register R2

MOV R4, A - Copy data from accumulator to register R4

MOV DPTR, #0F22Ch - Move immediate value F22Ch to the DPTR register

MOV R2, 80h - Copy data from 80h (Port 0 SFR) to R2

MOV 52h, #52h - Copy immediate data value 52h to RAM location 52h

MOV 52h, 53h - Copy data from RAM location 53h to RAM 52h

MOV A, @R0 - Copy contents of location addressed in R0 to A (indirect addressing)

MOVX

The MOVX instruction is used to access the external data memory (X indicates eXternal memory access). All external moves must work through the A register (accumulator).

MOVX @DPTR, A - Copy data from A to the address specified in DPTR

MOVX A, @DPTR - Copy data from address specified in DPTR to A

MOVC

The MOVC instruction is used to read data from the external code memory (ROM). Like the MOVX instruction the DPTR register is used as the indirect address register.

MOV DPTR, # 2000h - Copy the data value 2000h to the DPTR register

MOV A, #80h - Copy the data value 80h to register A

MOVC A, @A+DPTR - Copy the contents of the address 2080h (2000h + 80h) to register A

Note: For the MOVC the program counter, PC, can also be used to form the address.

PUSH and POP

PUSH and POP instructions are used with the stack only. The SFR register SP contains the current stack address.

PUSH 4Ch - SP is incremented. Contents of RAM location 4Ch is saved to the stack.

PUSH 00h - The content of R0 (which is at 00h in RAM) is saved to the stack.

POP 80h - The data from current SP address is copied to 80h and SP is decremented.

XCH

A special XCH (eXCHange) instruction will swap the data between source and destination. XCH instructions must use register A. XCHD is a special case of the exchange instruction where just the lower nibbles are exchanged.

XCH A, R3 - Exchange bytes between A and R3

XCH A, @R0 - Exchange bytes between A and RAM location whose address is in R0

XCH A, A0h - Exchange bytes between A and RAM location A0h (SFR port 2)

Arithmetic instructions

Some flags within the PSW, i.e. C, AC, OV, P, are utilised in many of the arithmetic instructions.

The arithmetic instructions are

- Addition
- Subtraction
- Increment/decrement
- Multiply/divide
- Decimal adjust

Addition

Register A (the accumulator) is used to hold the result of any addition operation.

The ADDC instruction is used to include the carry bit in the addition process.

ADD A, #25h - Adds the number 25h to A, putting sum in A

ADD A, R3 - Adds the register R3 value to A, putting sum in A

ADDC A, #55h - Add contents of A, the number 55h, the carry bit; and put the sum in A

ADDC A, R4 - Add the contents of A, the register R4, the carry bit; and put the sum in A.

Subtraction

Computer subtraction can be achieved using 2's complement arithmetic. Most computers also provide instructions to directly subtract signed or unsigned numbers. The accumulator A will contain the result (difference) of the subtraction operation. The C (carry) flag is treated as a borrow flag, which is always subtracted from the minuend during a subtraction operation.

SUBB A, #55d - Subtract the number 55 (decimal) and the C flag from A; and put the result in A.

SUBB A, R6 - Subtract R6 the C flag from A; and put the result in A.

SUBB A, 58h - Subtract the number in RAM location 58h and the C flag From A; and put the result in A.

Increment/Decrement

The increment (INC), simply adds a binary 1 to a number while a decrement (DEC) instruction subtracts a binary 1 from a number.

INC R7 - Increment register R7

INC A - Increment A

INC @R1 - Increment the number which is the content of the address in R1

DEC A - Decrement register A

DEC 43h - Decrement the number in RAM address 43h

INC DPTR - Increment the DPTR register

Multiply / Divide

The 8051 supports 8-bit multiplication and division. The arithmetic is relatively fast since multiplication and division are implemented as single instructions. For the MUL or DIV instructions the A and B registers must be used and only unsigned numbers are supported.

Multiplication

The MUL instruction is used as follows,

MUL AB - Multiply A by B.

The resulting product resides in registers A and B, the low-order byte is in A and the high order byte is in B.

Division

The DIV instruction is used as follows,

DIV AB - A is divided by B.

The remainder is put in register B and the integer part of the quotient is put in register A.

Decimal Adjust

The 8051 performs all arithmetic in binary numbers (i.e. it does not support BCD arithmetic). If two BCD numbers are added then the result can be adjusted by using the DA, decimal adjust, instruction.

DA A - Decimal adjust A following the addition of two BCD numbers.

Logical instructions

Boolean Operations

Most control applications implement control logic using Boolean operators to act on the data. Most microcomputers provide a set of Boolean instructions that act on byte level data. However, the 8051 additionally provides Boolean instruction which can operate on bit level data.

The following Boolean operations can operate on byte level or bit level data:

- ANL - Logical AND
- ORL - Logical OR
- CPL - Complement (logical NOT)
- XRL - Logical XOR (exclusive OR)

Logical operations at the BYTE level

ANL A, #55h - AND each bit in A with corresponding bit in number 55h, leaving the result in A.

ANL 42h, R4 - AND each bit in RAM location 42h with corresponding bit in R4, leaving the result in RAM location 42h.

ORL A, @R1 - OR each bit in A with corresponding bit in the number whose address is contained in R1 leaving the result in A.

XRL R4, 80h - XOR each bit in R4 with corresponding bit in RAM location 80h (port 0), leaving result in A.

CPL R0 - Complement each bit in R0

Logical operations at the BIT level

SETB 2Fh - Bit 7 of Internal RAM location 25h is set

CLR C - Clear the carry flag (flag =0)

CPL 20h - Complement bit 0 of Internal RAM location 24h

MOV C, 87h - Move to carry flag the bit 7of Port 0 (SFR at 80h)

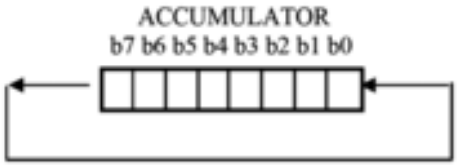
ANL C,90h - AND C with the bit 0 of Port 1 (SFR at 90)

ORL C, 91h - OR C with the bit 1 of Port 1 (SFR at 90)

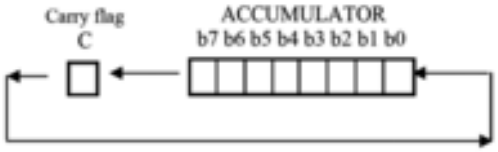
Rotate Instructions

The ability to rotate the A register (accumulator) data is useful to allow examination of individual bits. The options for such rotation are as follows:

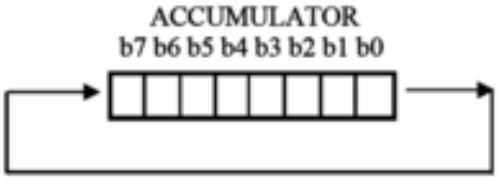
RL A ; Rotate A one bit to the left. Bit 7 rotates to the bit 0 position



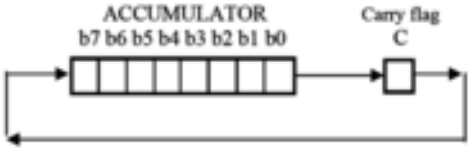
RLC A ; The Carry flag is used as a ninth bit in the rotation loop



RR A ; Rotates A to the right (clockwise)

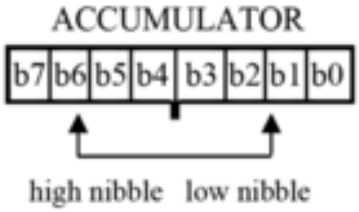


RRC A ; Rotates to the right and includes the carry bit as the 9th bit.



Swap

The Swap instruction swaps the accumulator's high order nibble with the low-order nibble using the instruction SWAP A



Program Control Instructions (Branch control instructions)

Branch control instructions alters the execution sequency.

The 8051 supports three kinds of jump instructions.

- LJMP
- SJMP
- AJMP

LJMP

LJMP (long jump) causes the program to jump to any location within the 64KByte program space.

LJMP 0F200h - Jump to address 0F200h

LJMP @A+DPTR - Jump to address which is the sum of DPTR and Reg. A

SJMP

SJMP (short jump) uses a singe byte address. This address is a signed 8-bit number and allows the program to branch to a distance –128 bytes back from the current PC address or +127 bytes forward from the current PC address.

AJMP

This is a special 8051 jump instruction, which allows a jump with a 2KByte address boundary.

Subroutines and program flow control

A subroutine is called using the LCALL or the ACALL instruction.

LCALL

This instruction is used to call a subroutine at a specified address. The address is 16 bits long so the call can be made to any location within the 64KByte memory space. When a LCALL instruction is executed the current PC content is automatically pushed onto the stack of the PC. When the program returns from the subroutine the PC contents is returned from the stack so that the program can resume operation from the point where the LCALL was made

The return from subroutine is achieved using the RET instruction.

ACALL

The ACALL instruction is logically similar to the LCALL but has a limited address range similar to the AJMP instruction.

Program control using conditional jumps

Most 8051 jump instructions use an 8-bit destination address, based on relative addressing, i.e. addressing within the range –128 to +127 bytes. When using a conditional jump instruction the programmer can simply specify a program label or a full 16-bit address for the conditional jump instruction's destination. The assembler will position the code and work out the correct 8-bit relative address for the instruction.

JZ LABEL - Jump to LABEL, if accumulator is equal to zero

JNZ LABEL - Jump to LABEL if accumulator is not equal to zero

JNC LABEL - Jump to LABEL if the carry flag is not set

DJNZ R2, LABEL - Decrement R2 and jump to LABEL if the resulting value of R2 is not zero.

CJNE R1, #55h , LABEL - Compare the magnitude of R1 and the number 55h and jump to LABEL if the magnitudes are not equal.

Data Transfer Instructions

MOV A, Rn	:	A ← Rn
MOV A, direct	:	A ← (direct)
MOV A, @Ri	:	A ← @Ri
MOV A, #data	:	A ← data
MOV Rn, A	:	Rn ← A
MOV Rn, direct	:	Rn ← (direct)
MOV Rn, #data	:	Rn ← data
MOV direct, A	:	(direct) ← A
MOV direct, Rn	:	(direct) ← Rn
MOV direct1, direct2	:	(direct1) ← (direct2)
MOV direct, @Ri	:	(direct)← @Ri
MOV direct, #data	:	(direct) ← #data
MOV @Ri, A	:	@Ri ← A
MOV @Ri, direct	:	@Ri ← (direct)
MOV @Ri, #data	:	@Ri ← data
MOV DPTR, #data16	:	DPTR ← data16
MOVC A, @A+DPTR	:	A ← Code byte pointed by A + DPTR
MOVC A, @A+PC	:	A ← Code byte pointed by A + PC
MOVC A, @Ri	:	A ← Code byte pointed by Ri 8-bit address)
MOVX A, @DPTR	:	A ← External data pointed by DPTR
MOVX @Ri, A	:	@Ri ← A (External data - 8bit address)
MOVX @DPTR, A	:	@DPTR ← A(External data - 16bit address)
PUSH direct	:	(SP) ← (direct)
POP direct	:	(direct) ← (SP)
XCH Rn	:	Exchange A with Rn
XCH direct	:	Exchange A with direct byte
XCH @Ri	:	Exchange A with indirect RAM
XCHD A, @Ri	:	Exchange least significant nibble of A with that of indirect RAM

Arithmetic Instructions

ADD A, Rn	:	A ← A + Rn
ADD A, direct	:	A ← A + (direct)
ADD A, @Ri	:	A ← A + @Ri
ADD A, #data	:	A ← A + data
ADDC A, Rn	:	A ← A + Rn + C
ADDC A, direct	:	A ← A + (direct) + C
ADDC A, @Ri	:	A ← A + @Ri + C
ADDC A, #data	:	A ← A + data + C
DA A	:	Decimal adjust accumulator
DIV AB	:	Divide A by B
		A ← quotient B ← remainder
DEC A	:	A ← A -1
DEC Rn	:	Rn ← Rn - 1
DEC direct	:	(direct) ← (direct) - 1
DEC @Ri	:	@Ri ← @Ri - 1
INC A	:	A ← A+1
INC Rn	:	Rn ← Rn + 1
INC direct	:	(direct) ← (direct) + 1
INC @Ri	:	@Ri ← @Ri +1
INC DPTR	:	DPTR ← DPTR +1
MUL AB	:	Multiply A by B
		A ← low byte (A*B) B ← high byte (A* B)
SUBB A, Rn	:	A ← A - Rn - C
SUBB A, direct	:	A ← A - (direct) - C
SUBB A, @Ri	:	A ← A - @Ri - C
SUBB A, #data	:	A ← A - data - C

Logical Instructions

ANL A, Rn	:	A ← A AND Rn
ANL A, direct	:	A ← A AND (direct)
ANL A, @Ri	:	A ← A AND @Ri
ANL A, #data	:	A ← A AND data
ANL direct, A	:	(direct) ← (direct) AND A
ANL direct, #data	:	(direct) ← (direct) AND data
CLR A	:	A ← 00H
CPL A	:	A ← A
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 data
ORL direct, A	:	(direct) ← (direct) OR A
ORL direct, #data	:	(direct) ← (direct) OR data
RL A	:	Rotate accumulator left
RLC A	:	Rotate accumulator left through carry
RR A	:	Rotate accumulator right
RRC A	:	Rotate accumulator right through carry
SWAP A	:	Swap nibbles within accumulator
XRL A, Rn	:	A ← A EXOR Rn
XRL A, direct	:	A ← A EXOR (direct)
XRL A, @Ri	:	A ← A EXOR @Ri
XRL A, #data	:	A ← A EXOR data
XRL direct, A	:	(direct) ← (direct) EXOR A
XRL direct, #data	:	(direct) ← (direct) EXOR data

Boolean Variable Instructions

CLR C	:	C-bit ← 0
CLR bit	:	bit ← 0
SET C	:	C ← 1

SET bit	:	bit 1
CPL C	:	C $\overline{C\text{-bit}}$
CPL bit	:	bit $\overline{bit}$
ANL C, /bit	:	C C . $\overline{bit}$
ANL C, bit	:	C C. bit
ORL C, /bit	:	C C + $\overline{bit}$
ORL C, bit	:	C C + bit
MOV C, bit	:	C bit
MOV bit, C	:	bit C

Program Branching Instructions

ACALL addr11	:	PC + 2 (SP) ; addr 11 PC
AJMP addr11	:	Addr11 PC
CJNE A, direct, rel	:	Compare with A, jump (PC + rel) if not equal
CJNE A, #data, rel	:	Compare with A, jump (PC + rel) if not equal
CJNE Rn, #data, rel	:	Compare with Rn, jump (PC + rel) if not equal
CJNE @Ri, #data, rel	:	Compare with @Ri A, jump (PC + rel) if not equal
DJNZ Rn, rel	:	Decrement Rn, jump if not zero
DJNZ direct, rel	:	Decrement (direct), jump if not zero
JC rel	:	Jump (PC + rel) if C bit = 1
JNC rel	:	Jump (PC + rel) if C bit = 0
JB bit, rel	:	Jump (PC + rel) if bit = 1
JNB bit, rel	:	Jump (PC + rel) if bit = 0
JBC bit, rel	:	Jump (PC + rel) if bit = 1
JMP @A+DPTR	:	A+DPTR PC
JZ rel	:	If A=0, jump to PC + rel
JNZ rel	:	If A ≠ 0 , jump to PC + rel
LCALL addr16	:	PC + 3 (SP), addr16 PC
LJMP addr 16	:	Addr16 PC
NOP	:	No operation
RET	:	(SP) PC
RETI	:	(SP) PC, Enable Interrupt

SJMP rel	:	PC + 2 + rel → PC
JMP @A+DPTR	:	A+DPTR → PC
JZ rel	:	If A = 0. jump PC+ rel
JNZ rel	:	If A ≠ 0, jump PC + rel
NOP	:	No operation

Time delay and calculations

The CPU takes a certain number of clock cycles to execute an instruction. In the 8051, these clock cycles are referred to as machine cycles. The length of the machine cycle depends on the frequency of the crystal oscillator connected to the 8051 system. The frequency of the crystal can vary from 4 MHz to 30 MHz, depending on the chip rating and manufacturer.

one machine cycle consists 12 oscillator periods. Therefore, to calculate the machine cycle for the 8051, we take 1/12 of the crystal frequency, then take its inverse, as shown in below example.

- Machine cycle for 8051 with 11.0592 MHz crystal frequency

$$\text{MC frequency} = \frac{11.0592}{12} = 0.9216\,\text{MHz} = 921.6\,\text{KHz}$$

$$\text{MC} = \frac{1}{921.6X10^3} = 0.001085\,X10^3 = 1.085\,\mu\text{sec}$$

- Machine cycle for 8051 with 20 MHz crystal frequency

$$\text{MC frequency} = \frac{20}{12} = 1.6666\,\text{MHz} = 1666.6\,\text{KHz}$$

$$\text{MC} = \frac{1}{1666.6X10^3} = 0.6\,\mu\text{sec}$$

The time to execute some instructions with 11.0592 MHz in 8051 is as follows

The time for one machine cycle for a system of 11.0592 MHz is 1.085 µsec.

Instruction	**Machine cycles**	**Time to execute**
MOV R3,#55	1	1x1.085 µsec =1.085 µsec
DEC R3	1	1x1.085 µsec =1.085 µsec
DJNZ R2,add	2	2x1.085 µsec =2.17 µsec
LJMP	2	2x1.085 µsec =2.17 µsec
SJMP	2	2x1.085 µsec =2.17 µsec
NOP	1	1x1.085 µsec =1.085 µsec
MUL AB	4	4x1.085 µsec =4.34 µsec

Delay calculation for 8051

A delay subroutine consists of two parts: (1) setting a counter, and (2) a loop. Most of the time delay is performed by the body of the loop. We calculate the time delay based on the instructions inside the loop and ignore the clock cycles associated with the instructions outside the loop.

For calculate delay in the program, let us consider following program with crystal frequency 11.0592 MHz.

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


DELAY : MOV R3,#200
HERE : DJNZR3, HERE
 RET
```

Machine cycles for each instruction of the DELAY subroutine

```
MOV R3,#2OO     - 1 mc
DJNZ            - 2 mc
RET             - 2 mc
```

Time for delay subroutine = (1+ (200 x 2) + 2] x 1.085 µsec = 436.255 µsec.

In above example, the largest value the R3 register can take is 255; therefore, one way to increase the delay is to use NOP instructions in the loop. NOP, which stands for "no operation," simply wastes time.

```
DELAY :  MOV R3, #50
HERE  :  NOP
         DJNZR3, HERE
         RET
```

Time for delay subroutine = (1+ ((1+2)50 x 2) + 2] x 1.085 µsec = 166.005 µsec.

Port organization

One major feature of microcontroller is in built IO ports that connect the 8051 to the outside world.

There are four 8bit ports in 8051.

Port 0

Port 0 pins may serve as inputs, outputs, or, when used together, as a bidirectional lower order address and data bus for external memory.

Port 1

Port 1 pins have no dual function; simply it acts as input or output port.

Port 2

Port 2 may be used as an input/output port similar in operation to port l. The alternate use of port 2 is to supply a high-order address byte in conjunction with the port 0 low-order byte to address external memory.

Port 3

Port 3 is an input /output port similar to port 1. The every pin of port 3 has alternate function.

P3.0 - RXD -Serial data input

P3.1-TXD - Serial data output

P3.2 – INT0 - External interrupt 0

P3.3 - INT1 - External interrupt 1

P3.4 –T0 - External timer 0 input

P3.5 - T1 - External timer 1 input

P3.6 – WR - External memory write pulse

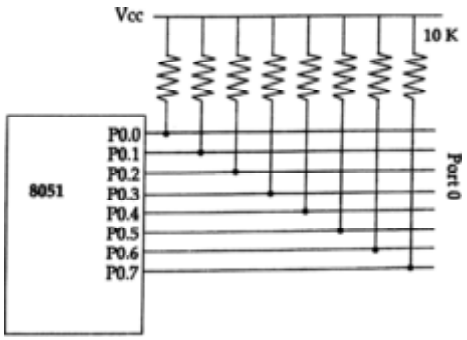
P3.7 – RD - External memory read pulse

IO programming

The 4 ports P0, Pl, P2, and P3 are 8bit ports. All the ports upon RESFT are configured as inputs and ready to be used as input ports. When the first 0 is written to a port, it becomes an output port. To reconfigure as an input, a 1 must be sent to the port. To use any of these ports as an input port, it must be programmed.

Port 0

Port 0 occupies a total of 8 pins (pins 32 - 39) and used for input or output. To use the pins of port 0 as both input and output ports, each pin must be connected externally to a 10K-ohm pull-up resistor. This is due to the fact that P0 is an open drain.



Following code will continuously send out to port 0 the alternating values of 55H and AAH.

```
BACK:   MOV A,#55H
        MOV P0,A
        ACALL DELAY
        MOV A,#0AAH
        MOV PO,A
        ACALL DELAY
        SJMP BACK
```

The above example continuously complements the port0 bits.

Port 0 as input port

In order to make it an input, the port must be programmed by writing 1 to all the bits. In the following code, port 0 is configured first as an input port by writing bits 1 to it, and then data is received from that P0 and sent to Pl.

```
        MOV A, #0FFH
        MOV P0, A  ; P0 as an input port
BACK:   MOV A, P0
        MOV P1, A
        SJMP BACK
```

Dual role of port 0

Port 0 is also designated as ADO - AD7, acts as both address and data bus when connecting an 8051to an external memory.

Port 1

Port1 occupies a total of 8 pins (pin1-8), used as input or output. This port does not need any pull-up resistors since it already has pull-up resistors internally.

The following code will continuously send out to port 1 the alternating values

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

Port 1 as input

If port l has been configured as an output port, to make it an input port again, it must be programmed as such by writing 1 to all its bits.

The following code, port 1 is configured first as an input Port by writing 1s to it, and then data is received from that port and saved in R7, R6, and R5.

```
        MOV A, #0FFH
        MOV P1, A    ; make P1 an input port
        MOV A, P1
        MOV R7, A
        ACALL DELAY
        MOV A, P1
        MOV R6, A
        ACALL DELAY
        MOV A, P1
        MOV R5, A
        ACALL DELAY
HERE:   SJMP HERE
```

Port 2

Port 2 occupies a total of 8 pins (pins 21 -28). It can be used as input or output just like P1. Port 2 does not need any pull-up resistors since it already has pull-up resistors internally. Upon reset, port 2 is configured as an input

The following code will send out continuously to port 2 the alternating values 55H and AAH.

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

Port 2 as input

To make port 2 an input, it must programmed as such by writing 1 to all its bits.

In the following code, port 2 is configured first as an input port by writing 1s to it. Then data is received from that port and is sent to P1 continuously.

```
        MOV A, #0FFH
        MOV P2, A    ; make P2 an input port
BACK:   MOV A, P2
        MOV P1, A
        SJMP BACK
```

Dual role of port 2

P2 is used as simple I / O. port 2 acts as a higher order address bus, along with P0; provide the 16-bit address for external memory.

Port 3

Port 3 occupies a total of 8 pins (10-17). It can be used as input or output. P3 does not need any pull up resistors.

The table shows alternating functions of port3.

P3 Bit	Function	Pin	Description
P3.0	RXD	10	Serial data input
P3.1	TXD	11	Serial data
P3.2	INT 0	12	External interrupt 0
P3.3	INT 1	13	External interrupt 1
P3.4	T0	14	Timer 0 input
P3.5	T1	15	Timer 1 input
P3.6	WR	16	External memory write pulse
P3.7	RD	17	External read write pulse

IO bit-manipulation

Sometimes we need to access only 1 or 2 bits of the port instead of the entire 8 bits. A powerful feature of 8051 I / O ports is their capability to access individual bits of the port without altering the rest of the bits in that port.

When accessing a port in single-bit manner, we use the syntax "SETB X .Y" where X is the port number 0, 1, 2, or 3, and Y is the desired bit number from 0 to 7.

For example SETB P1.5 instruction sets high bit 5 of port 1.

Single bit addressability of ports

P0	**P1**	**P2**	**P3**	**Port Bit**
P0.0	P1 .0	P2.0	P30	D0
P0.1	P1.1	P2.1	P3.1	D1
P0.2	P1.2	P2.2	P3.2	D2
P0.3	P1.3	P2.3	P3.3	D3
P0.4	P1.4	P2.4	P3.4	D4
P0.5	P1.5	P2.5	P3.5	D5
P0.6	P1.6	P2.6	P3.6	D6
P0.7	P1.7	P2.7	P3.7	D7

The following code toggles the bit P1.3

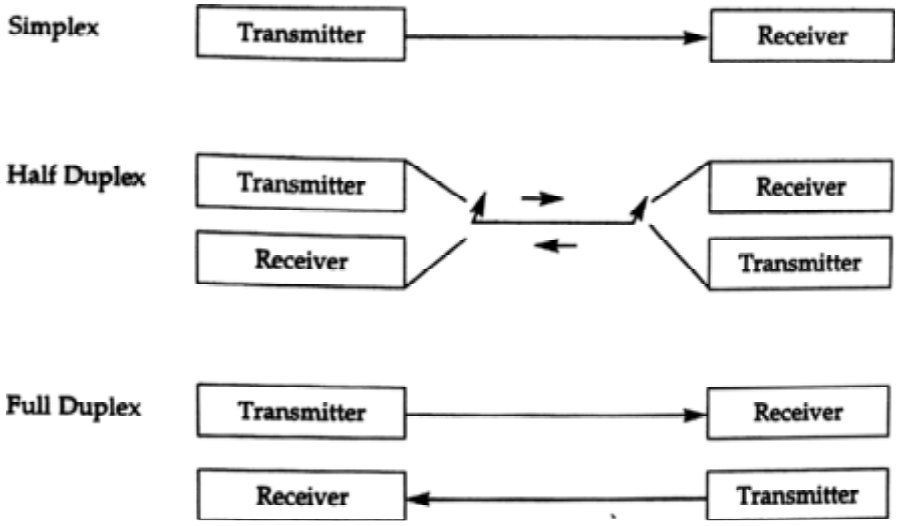
```
        SETB P1.3
BACK:   CPL P1.3
        ACALL DELAY
        SJMP BACK
```

UNT IV

INTERFACING OF PERIPHERALS TO MICROCONTROLLER

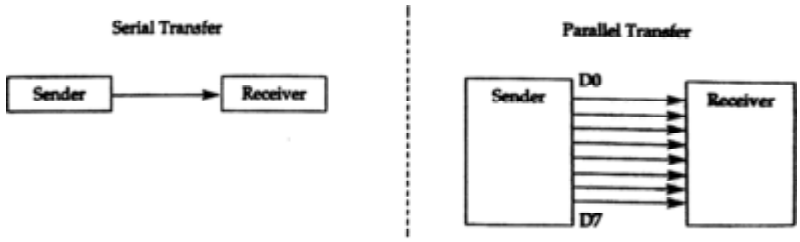
Half- and full-duplex transmission

In data transmission there are two types; one is simplex transmissions, in which the computer only sends data, such as with printers. Second is duplex transmission, in this the data can be transmitted and received. Duplex transmissions can be half or full duplex. If data is transmitted one way at a time, it is referred to as half duplex. If the data can go both ways at the same time, it is full duplex. Of course, full duplex requires two wire conductors for the data lines, one for transmission and one for reception.



Serial communication

When a microprocessor communicates with the outside world, it provides the data in byte-sized chunks. The 8-bit data bus is required to transmit; an 8-bit data path is not safe and expensive, For these reasons, serial communication is used for transferring data between two systems located at distances of hundreds of feet to millions of miles apart. Figure shows serial versus parallel data transfers.



The fact that serial communication uses a single data line instead of the 8-bit data line of parallel communication not only makes it much cheaper but also enables two computers located in two different cities to communicate over the telephone.

For serial data communication, the byte of data must be converted to serial bits using a parallel-in serial-out shift register; then it can be transmitted over a single data line. This also means that at the receiving end there must be a serial-in-

parallel-out shift register to receive the serial data and pack them into a byte. This conversion is performed by a peripheral device called a modem, which stands for “modulator/demodulator.”

Serial data communication uses two methods, asynchronous and synchronous. The synchronous method transfers a block of data (characters) at a time, while the asynchronous method transfers a single byte at a time. The 8051 has a built-in UART (universal asynchronous receiver-transmitter). The 8051 uses serial data communication circuit that uses register SBUF to hold data. Register SCON controls data communication, register PCON controls data rates, and pins RXD and TXD connect to the serial data network.

SBUF is physically two registers. One for transmission and second for reception, but both the registers use address 99h.

There are four programmable modes for serial data communication that are chosen by setting the SM0 and SM1 bits in SCON. Baud rates are determined by the mode selection.

SCON REGISTER

It is bit addressable as SCON.0 to SCON.7.

SM0	SM1	SM2	REN	TB8	RB8	TI	RI
bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0

SM0 and SM1- Serial port mode selection bits

SM0	SM1	Mode	Description
0	0	0	Shift register; baud rate = f/12
0	1	1	8-bit UART; baud rate = variable
1	0	2	9 bit UART; baud rate = 1/32 or 1/64
1	1	3	9 bit UART; baud rate = variable

SM2 - Multiprocessor communications bit. Set/cleared by program to enable multiprocessor communications in modes 2 and 3

REN - Receive enable bit. Set to 1 to enable reception; cleared to 0 to disable reception.

TB8 -Transmitted bit 8. Set/cleared by program in modes 2 and 3.

RB8 - Received bit 8. Bit 8 of received data in modes 2 and 3; stop bit in mode 1. Not used in mode 0.

TI - Transmit Interrupt flag. Set to one at the end of bit 7 time in mode 0, and at the beginning of the stop bit for other modes. Must be cleared by the program.

RI -Receive Interrupt flag. Set to one at the end of bit 7 time in mode O, and halfway through the stop bit for other modes. Must be cleared by the program.

PCON REGISTER

PCON is not bit addressable register.

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

Bit 7 - SMOD - Serial baud rate modify bit. Set to 1 by program to double baud rate using timer 1 for modes 1, 2 and 3 Cleared to 0 by program to use timer 1 baud rate.

Bits 6, 5 and 4 - Not implemented.

Bit3 - GF 1 - General purpose user flag bit 1. Set/cleared by program.

Bit 2 GF0 - General purpose user flag bit 0. Set/cleared by program.

Bit1 - PD - Power down bit. Set to 1 by program to enter power down configuration for CHMOS processors.

Bit 0 - IDL - Idle mode bit. Set to 1 by program to enter idle mode configuration for CHMOS processors.

Serial Data Interrupts

Serial data communication is a relatively slow process, occupying many milliseconds per data byte to accomplish. The Serial Data flags in SCON TI and RI are set whenever data byte is transmitted (TI) or received (RI).

Data Transmission

Transmission of serial data bits begins at any time data is written to SBUF. TI is set to a 1 when the data has been transmitted.

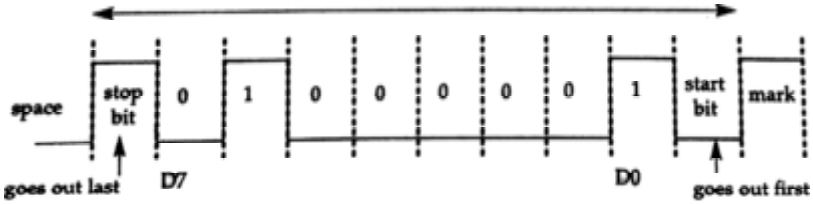
Data Reception

Reception of serial data will begin if the receive enable bit (REN) in SCON is set to 1 for all modes. In addition, for mode 0 only RI must be cleared to 0. Receiver Interrupt flag RI is set after data has been received in all modes.

Start and stop bits

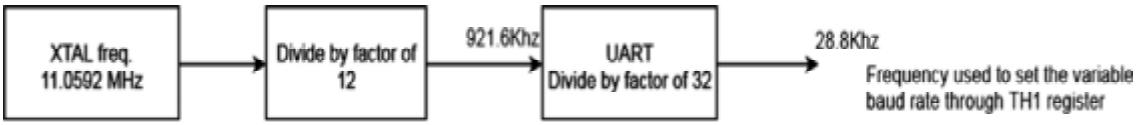
Asynchronous serial data communication is widely used for character-oriented transmissions, while block-oriented data transfers use the synchronous method. In the asynchronous method, each character is placed between start and stop bits. This is called framing. The start bit is always one bit is always a 0 (low), but the stop bit can be one or two bits and the stop bit(s) is 1 (high).

Figure shows framing of 8 bits.



Baud rate in the 8051 (bits per second)

The 8051 transfers and receives data serially at many different baud rates. The baud rate in the 8051 is programmable with the help of Timer 1. The 8051 divides the crystal by 12 to get the machine cycle frequency. In the case of XTAL = 11.0592 MHz, the machine cycle frequency is 921.6 kHz (110592 MHz/12 = 921.6 kHz). The 8051's serial communication UART circuitry divides the machine cycle frequency of 921.6 kHz by 32 once more before it is used by Timer 1 to set the baud rate. Therefore, 921.6 KHz divided by 32 gives 28,800 Hz. When Timer 1 is used to set the baud rate it must be programmed in mode 2 that is 8-bit auto-reload.



To get baud rates compatible with the PC, we must load TH1 with the values shown in Table.

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

With XTAL = 11.0592 MHz.

To get 9600 baud rate, we have to load TH1 with -3 or FDh. That is explained as follows.

Note: XTAL = 1140592 MHz,

$$\text{baud rate} = \frac{2^{SMOD}}{32} X \frac{\text{Crystal frequency}}{12d(256 - TH1)}$$

$$\text{baud rate} = \frac{2^0}{32} X \frac{11.0592}{12(256 - FD)} = \frac{1}{32} X \frac{11.0592}{12(3)} = 9600Hz$$

Serial data transmission modes and port programming

Serial Data Transmission Modes

The 8051 have included four modes of serial data transmission in a variety of ways and a multitude of baud rates. Modes are selected by the programmer by setting the mode bits SM0 and SM1 in SCON. Baud rates are fixed for mode 0 and variable, using timer 1 and the serial baud rate modify bit (SMOD) in PCON, for modes 1, 2, and 3.

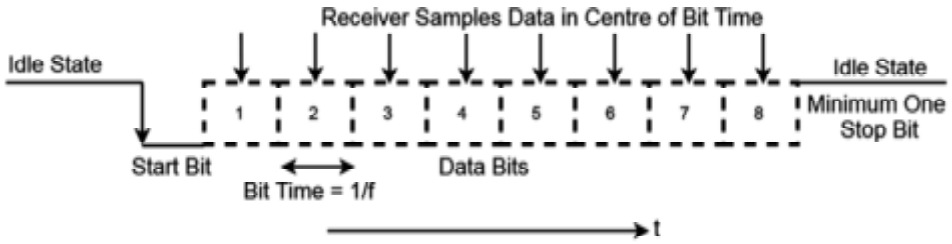
Serial data mode 0 – Shift register mode

Setting bits SM0 and SM1 in SCON to 00, configures ‘SBUF to receive or transmit eight data bits using pin RXD for both functions. Pin TXD is connected to the internal shift frequency pulse source to supply shift pulses to external circuits. The shift frequency, or baud rate, is fixed at 1/12 of the oscillator frequency.

Received data comes in on pin RXD and should be synchronized with the shift clock produced at TXD. The baud rate used in mode 0 will be much higher than standard for any reasonable oscillator frequency.

Serial data mode 1 – Standard UART (8 bit)

When SM0 and SM1 are set to 01, SBUF becomes a 10-bit full-duplex receiver/transmitter that may receive and transmit data at the same time. Pin RXD receives all data, and pin TXD transmits all data. Figure shows the format of a data word.



Transmitted data is sent as a start bit, eight data bits (LSB first), and a stop bit. Interrupt flag TI is set once all ten bits have been sent. Each bit interval is the inverse of the baud rate frequency.

Received data is obtained in the same order; reception is triggered by the falling edge of the start bit. Data bits are shifted into the receiver at the programmed baud rate, and the data word will be loaded to SBUF if the following conditions are true: RI must be 0, and mode bit SM2 is 0 or the stop bit is 1.

Of the original ten bits, the start bit is discarded, the eight data bits go to SBUF, and the stop bit is saved in bit RB8 of SCON. RI is set to 1, indicating a new data byte has been received.

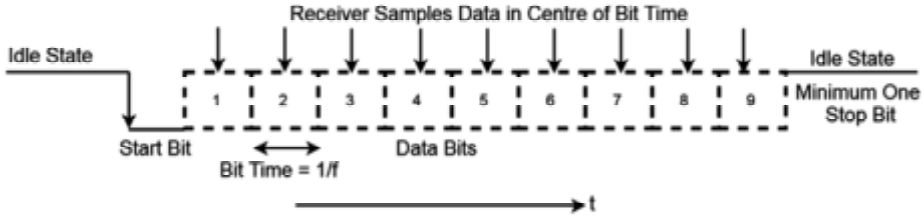
Mode 1 baud rate:

Timer 1 is used to generate the baud rate for mode 1 by using the Overflow flag of the timer. Typically timer 1 is used in timer mode 2 as an autoload-8-bit timer that generates the baud frequency:

$$f_{baud1} = \frac{2^{SMOD}}{32} X \frac{\text{Crystal frequency}}{12d(256 - TH1)}$$

Serial data mode 2 (Multiprocessor mode) : 9bit UART

Mode 2 is similar to mode 1 except 11 bits are transmitted: a start bit, nine data bits, and a stop bit, as shown in Figure.



The ninth data bit is copied from bit TB8 in SCON during transmit and stored in bit RB8 of SCON when data is received. Both the start and stop bits are discarded.

The baud rate is programmed as follows:

$$f_{baud2} = \frac{2^{SMOD}}{32} X \text{ Crystal frequency}$$

Here, as in the case for mode 0, the baud rate is much higher than standard communication fates. This high data rate is needed in many multiprocessor applications.

Serial data mode 3

Mode 3 is identical to mode 2 except that the baud rate is determined exactly as in mode 1, using timer 1 to generate communication frequencies.

Serial Port programming

Programming the 8051 to transfer data serially

In programming the 8051 to transfer character bytes serially, the following steps must be taken.

- The TMOD register is loaded with the value 20H, indicating the use of Timer 1 in mode 2 (8-bit auto-reload) to set the baud rate.
- The TH1 register is loaded with one of the values to set the baud rate for serial data transfer(with XTAL = 11.0592MHZ, we have to load TH1 with FD (-3) to determine 9600Hz baud rate)
- The SCON register is loaded with the value 50H, indicating serial mode 1
- TR1 is set to l to start Timer 1.
- T1 is cleared by the CLR TI.
- The character byte to be transfer serially is written into SBUF register.
- The TI flag is monitored with the use of the instruction JNB TI, XX , to see the character has been transferred completely or not.

To transfer next character, go to step 5.

```
        MOV TMOD, #20H
        MOV TH1, #0FDH
        MOV SCON, #50H
        SETB TRl
        CLR TI
        MOV SBUF, #'Y'
HERE:   JNB TI, HERE
HERE1:  SJMP HERE1
```

Programming the 8051 to receive data serially

In the programming of the 8051 to receive character bytes serially, the following steps must be taken,

- The TMOD register is loaded with the value 20H, indicating the use of Timer 1 in mode 2 (8-bit auto-reload) to set the baud rate.
- The TH1 register is loaded with one of the values to set the baud rate for serial data transfer(with XTAL = 11.0592MHZ, we have to load TH1 with FD (-3) to determine 9600Hz baud rate)
- The SCON register is loaded with the value 50H, indicating serial mode 1 and receive enable bit REN is turned on.
- TR1 is set to l to start Timer 1.

- T1 is cleared by the CLR TI.
- The RI flag bit is monitored with the use of the instruction “JNB RI, xx" to see if an entire character has been received yet.
- When RI is raised, SBUF has the byte. It’s contents are moved into a safe place.

To receive the next character, go to Step 5.

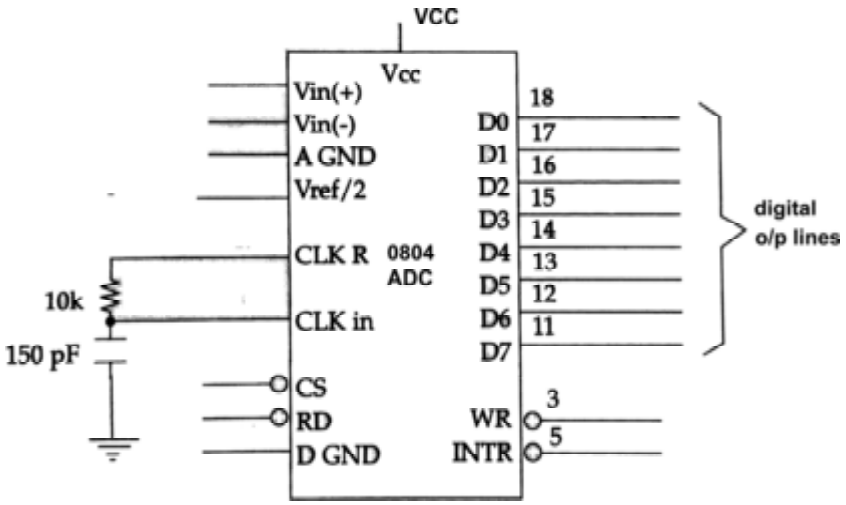
```
        MOV TMOD, #2OH
        MOV TH1, #0FDH
        MOV SCON, #50H
        CLR TR1
HERE:   JNB RI  HERE
        MOV A,SBUF
        MOV 6F, A
HERE1:  SJMP HERE1
```

Analog -to-digital converters (ADC)

Analog-to-digital converters are widely used devices for data acquisition. Digital computers use binary (discrete) values, but in the physical world everything is analog (continuous) such as temperature, pressure (wind or liquid), humidity, velocity, .. etc.

A physical quantity is converted to electrical signals using a device called a transducer. Transducers are also referred to as sensors. Therefore, we need an ADC to translate the analog signals to digital numbers so that the microcontroller can read and process them. We have parallel ADCs and serial ADCs.

Parallel ADC 0804



The ADC0804 IC is an 8-bit parallel ADC in the family of the ADC0800 series from National Semiconductors. It works with +5 volts and has a resolution of 8 bits. In the ADC0804, the conversion time varies depending on the clocking signals applied to the CLK IN pin,

CS - Chip select is an active low input used to activate the ADC0804 chip. To access the ADC0804, this pin must be low.

RD (read) - This is an input signal and is active low. The ADC converts the analog input to its binary equivalent and holds it in an internal register. RD is used to get the converted data out of the ADC0804 chip. When CS = 0, if a high-to-low pulse is applied to the RD pin, the 8-bit digital output at the D0 - D7 data pins. The RD pin is also referred to as output enable (OE).

WR (write) – start conversion

If CS = 0, when WR makes a low to high transition, the ADC0804 starts converting the analog input value of V_{in} to an 8-bit digital number

CLK IN and CLKR

CLK IN is an input pin connected to an external clock source when an external clock is used for timing. However, the 0804 has an internal clock generator. To use the internal clock of the ADC0804, the CLK IN and CLK R pins are connected to a capacitor and a resistor, as shown in Figure. In this case the clock frequency is given by

$$f = \frac{1}{1.1\,R\,C}$$

Ex: Typical values are R = 10 Kohms and C = 150 pF. Substituting in the above equation, we get f = 606 kHz. In this case, the conversion time is 110 us.

INTR (end of conversion)

This is an output pin and is active low. It is a normally high pin and when the conversion is finished it goes low to signal the CPU that the converted data is ready to be picked up.

After INTR goes low, we make CS = 0 and send a high-to-low pulse to the RD pin to get the data out of the ADC0804 chip.

V_{in} (+) and V_{in} (-)

These are the differential analog inputs where $V_{in} = V_{in}$ (+) – V_{in} (-). Often the V_{in} (-) pin is connected to ground and the V_{in} (+) pin is used as the analog input.

VCC

This is the +5 volt power supply. It is also used as a reference voltage when the $V_{ref}/2$ is open.

$V_{ref}/2$

This pin is used for the reference voltage. If this pin is open, the analog input voltage for the ADC0804 is in the range of 0 to 5V. V_{ref} / 2 is used to implement analog input voltages other than 0 to 5 V. For example, if the analog input range needs to be 0 to 4 volts, V_{ref}/ 2 is connected to 2V.

D0-D7

D0 - D7 are the digital data output pins of a parallel ADC0804. The converted data is accessed only when CS = O and RD is forced low.

Analog ground and digital ground

These are the input pins providing the ground for both the analog signal and the digital signal.

The following steps must be followed for data conversion by the ADC0804.

- Make CS = 0 and send a low-to-high pulse to pin WR to start the conversion.
- Keep monitoring the INTR pin.
- If INTR is low, the conversion is finished.
- After the INTR has become low, we make CS = 0 and send high -to-low to RD to get the data out of ADC0804.

The timing for this process is shown in figure.

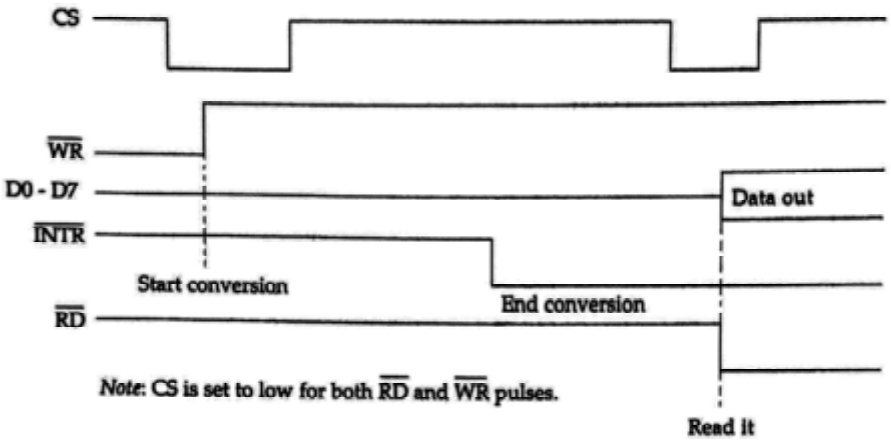
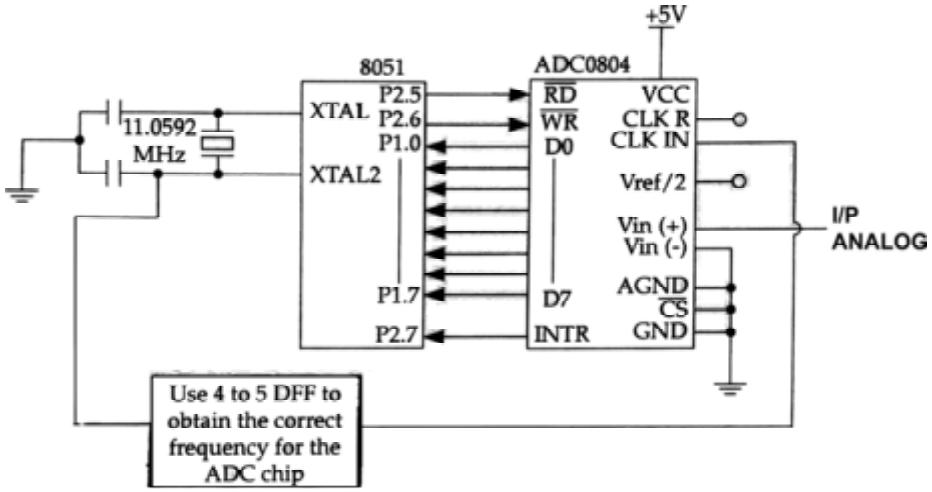


Fig shows interfacing connections between ADC0804 and 8051. Here CLKIN connected to the external CLK at 8051's crystal oscillator.

The following program monitors the INTR pin and brings an analog input into register A. It then calls hex-to-ASCII conversion and data display subroutines. Out data pins (D0-D7) of ADC0804 are connected to port 1 of 8051. RD of ADC0804 connected to pin P2.5 of 8051, WR of ADC0804 connected to pin P2.6 of 8051 and INTR of ADC0804 connected to pin P2.7 of 8051.

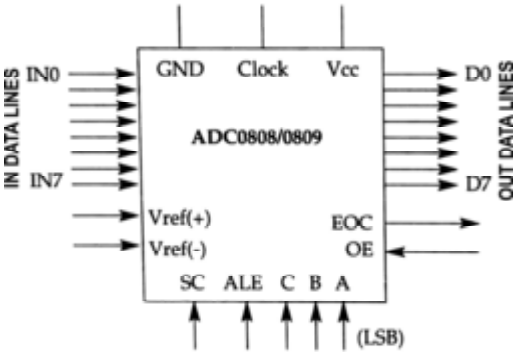


Program:

```
        RD BIT P2.5
        WR BIT P2.6
        INTR BIT P2.7
        MYDATA EQU P1
        MOV P1, #0FFH
BACK:   SETB INTR
        CLR WR
        SETB WR
HERE:   JB INTR, HERE
        CLR RD
        MOV A, MYDATA
        ACALL CONVERSION
        ACALL DISPLAY
        SETB RD
        SJMP BACK
```

Another useful chip is the ADC0808/0809 has 8 analog inputs. This chip allows 8 different analog inputs by using selection ckt and has 8 output data lines just like ADC0804.

Fig. shows pin configuration of ADC 0808/09 and table shows selection of analog input channel.



Selected i/p channel	C	B	A
IN0	0	0	0
IN1	0	0	1
IN2	0	1	0
IN3	0	1	1
IN4	1	0	0
IN5	1	0	1
IN6	1	1	0
IN7	1	1	1

In many applications, space is critical issue, using large no. of pins for data is not feasible. For this reason serial ADCs are becoming widely used than parallel ADC. MAX 1112 is a serial ADC chip from MAXIM Corporation.

DAC interfacing

Digital-to-analog (DAC) converter

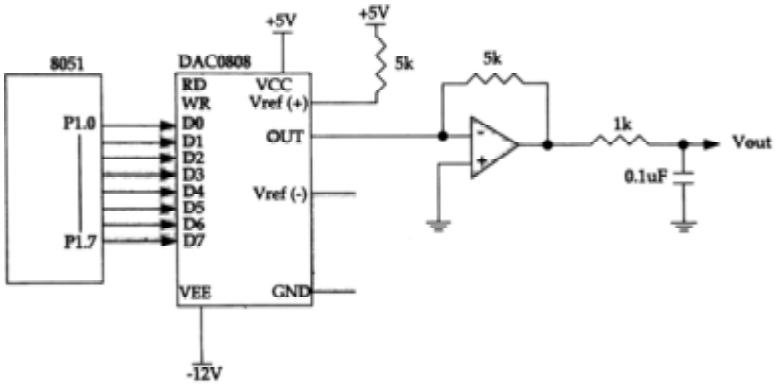
The digital-to-analog converter (DAC) is a device widely used to convert digital pulses to analog signals. Basically from digital electronics, we have 2 methods of designing a DAC.

1. Binary weighted 2. R/2R ladder network

The MC1408 (DAC0808) uses the R/2R method. It is an 8bit DAC converts 8bit digital data into analog signal. The no. of data inputs decides the resolution of the DAC. So, the number of analog output levels is equal to 2^n, where n is the number of data bit inputs. Therefore, an 8-input DAC (0808) provides 256 discrete voltage or current levels of output. In the DAC0808, the digital inputs are converted to current (I_{out}), and by connecting a resistor to the I_{out} pin, we convert the result to voltage.

The total current provided by the I_{out} pin is a function of the binary numbers D0 - D7 inputs of the DAC0808 and the reference current (I_{ref}), and is given by:

$$I_{out} = I_{ref}\left(\frac{D_7}{2}+\frac{D_6}{4}+\frac{D_5}{8}+\frac{D_4}{16}+\frac{D_3}{32}+\frac{D_2}{64}+\frac{D_1}{128}+\frac{D_0}{256}\right)$$



Where D0 is the LSB, D7 is the MSB for the inputs, and I_{ref} is the input current that must be applied. If I_{ref} = 2 mA, that all inputs to the DAC are high. The maximum current is 1.99 mA, which is as follows.

Digital i/p = FFh = 11111111b

$$I_{out} = 2\left(\frac{1}{2}+\frac{1}{4}+\frac{1}{8}+\frac{1}{16}+\frac{1}{32}+\frac{1}{64}+\frac{1}{128}+\frac{1}{256}\right) = 2\left(\frac{255}{256}\right) = 2(0.996) = 1.99\,mA$$

Ex: Program for generation of saw tooth wave form by using above interfacing ckt:

```
        MOV A, #00H
HERE:   MOVE P1, A
        INC A
        SJMP HERE
```

Ex: The following program is to set maximum required peak value in saw tooth wave.

```
        MOV R0, #7FH
BACK:   MOV A, #00H
HERE:   MOVE P1, A
        INC A
        CJNE A, R0 HERE
        SJMP BACK
```

PROGRAMMING THE 8255

8255 features

The 8255 is a 40-pin DIP chip. It has three separately accessible ports. The ports are each 8-bit, and are named A, B, and C. The individual ports of the 8255 can be programmed to be input or output.

Port A (PA0 - PA7) - The 8-bit port A can be programmed as all input, or as all output, or all bits as bidirectional input/ output.

Port B (PB0 - PB7) - The 8-bit port B can be programmed as all input or as all output. Port B cannot be used as a bidirectional port.

Port C (PC0 - PC7) - This 8-bit port C can be all input or all output. It can also be split into two parts, CU (upper bits PC4 - PC7) and CL (lower bits PC0 - PC3). Each can be used for input or output.

RD and WR – these two are active-low control pins. The RD and WR signals from the 8051 are connected to these inputs.

D0 - D7 (data bus) – These pins are used for data transmission.

RESET - This is an active-high signal input into the 8255 used to clear the control register.

A0 and A1 – these two pins are used to access the A,B, C and CER as follows.

CS	A1	A0	Port/CWR
0	0	0	A
0	0	1	B
0	1	0	C
0	1	0	CWR
1	x	x	8051 not selected

Mode selection of 8255

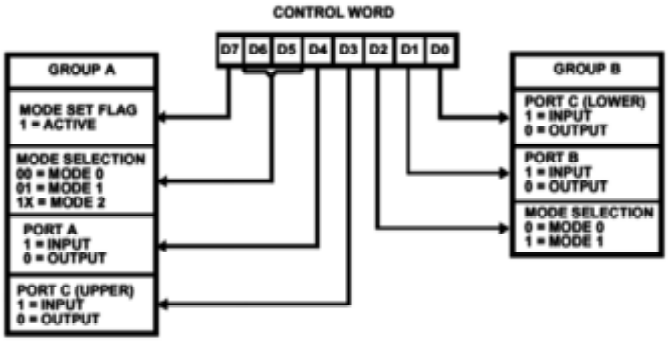
These are three modes. The ports of 8255 can be programmed in any of the following modes by CWR register.

Mode 0 : simple IO mode, in this any of the ports A, B, CL, and CU can be programmed as input or output

Mode 1: in this mode, ports A and B can be used as in put or output ports with handshaking capabilities. Handshaking signals provided by the bits of port C.

Mode 2 : In this mode, port A can be used as a bidirectional I/O port with handshaking capabilities whose signals are provided by port C.

Control word of PPI 8255A



Ex: Obtain the control word for the following configuration of the ports of 8255A.

Port A – as input port,

Mode for Port A – mode 1

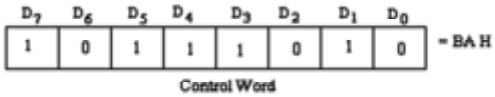
Port B – input port,

Mode for Port B – mode 0,

Port CLower – output port

The remaining pins PC6 and PC7 of port C are to be used input pins.

Sol: The control word for this case is given as



Ex: Obtain the control word for the following configuration of the ports of 8255A.

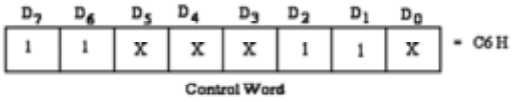
Port A – as bidirectional

Mode for Port A – mode 2

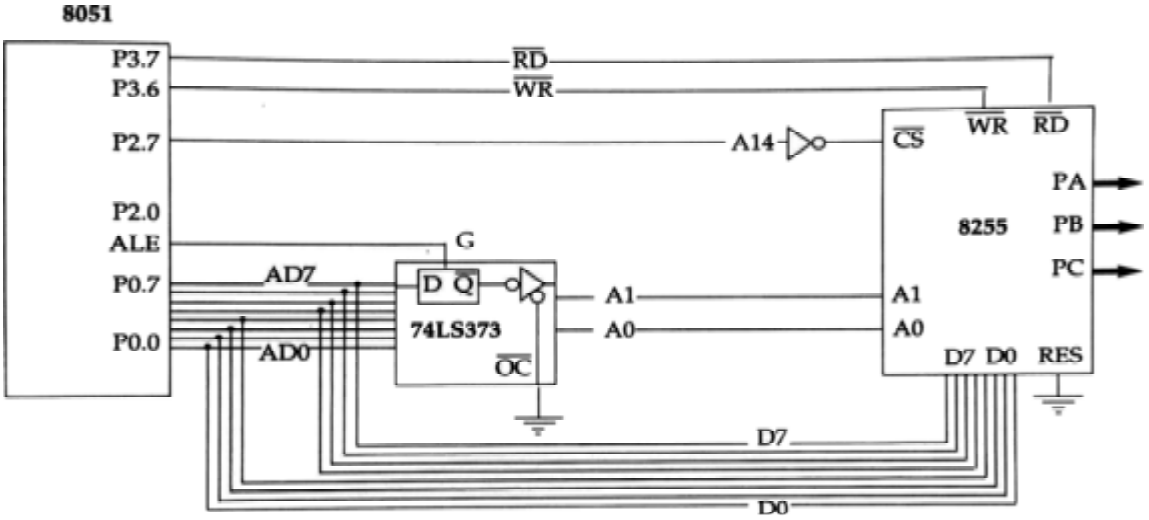
Port B – input port

Mode for Port B – mode 1

Sol: The control word for this case is given as



Interfacing of 8255 to 8051



The 8255 chip is programmed in any of the 3 modes mentioned earlier, by sending a byte to the control register of the 8255. We must first find the port addresses assigned to each of ports A, B. C, and CWR. This is called mapping the IO port. The 8255 is connected to an 8051as shown in fig, this method of connecting IO chip to CPU is called memory mapped IO since it is mapped into memory space. For this reason we use instruction such as MOVX to access the 8255.

Programming example:

4 switches connected to the upper 4 bits of port A (PA4 – PA7). Write a program to transfer status of switches to LEDs which connected to the lower 4 bits of port B (PB0 – PB4)

CWR for above example – 99H

Address	Port
4000H	Port A
4001H	Port B
4002 H	port C
4003 H	CWR

```
        MOV A, 99H
        MOV DPTR, #4003H
        MOVX A, @DPTR
BACK:   MOV DPTR, #4000H
        MOVX A, @DPTR
        SWAP A
MOV DPTR, #4001H
        MOV @DPTR, A
        SJMP BACK
```

UNIT - V

APPLICATIONS

Temperature measurement

Temperature sensor interfacing

Transducers convert physical data such as temperature, light intensity, flow, and speed to electrical signals. Depending on the transducer, the output produced is in the form of voltage, current, resistance, or capacitance. Temperature is converted to electrical signals using a transducer called a thermistor, it responds to temperature change by changing resistance, but its response is not linear.

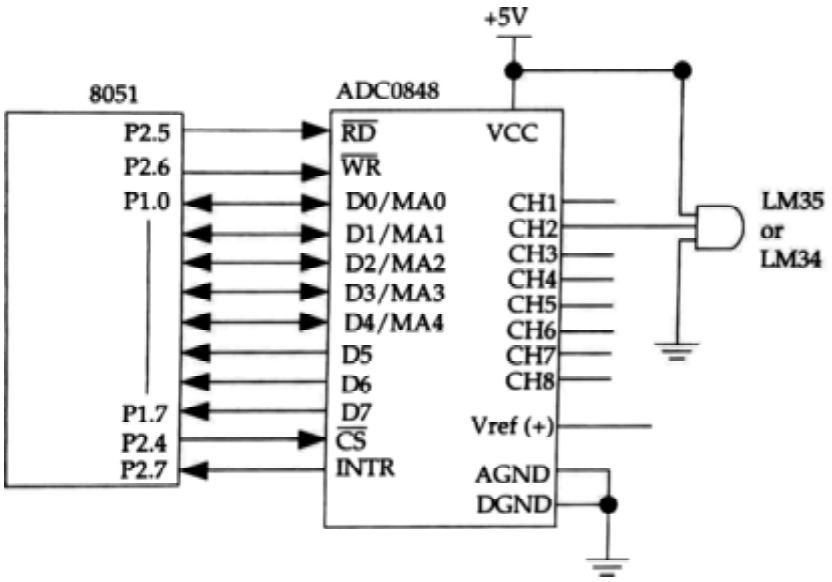
The complexity associated with writing software for such nonlinear devices. That's why manufacturers introduced linear temperature sensor. Simple and widely used linear temperature sensors are the LM34 and LM35.

LM34 and LM35 temperature sensors

The sensor LM34 is a temperature sensor whose output voltage is linearly proportional to the Fahrenheit temperature. The LM34 requires no external calibration since it is internally calibrated. It outputs 10 mV for each degree of Fahrenheit temperature.

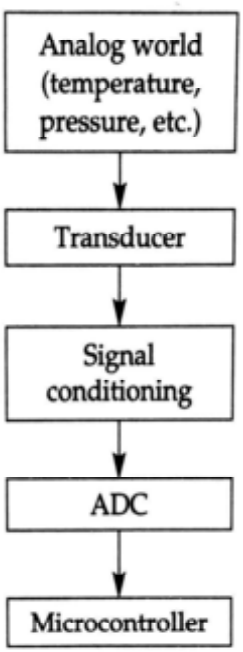
The LM35 sensor is a temperature sensor whose output voltage is linearly proportional to the Celsius (centigrade) temperature. The LM35 requires no external calibration since it is internally calibrated. It outputs 10 mV for each degree of centigrade temperature.

Interfacing LM 35 to 8051



Signal conditioning

The most common transducers produce an output in the form of voltage, current, charge, capacitance, and resistance. However, we need to convert these signals to voltage in order to send input to an A-to-D converter. This conversion (modification) is called signal conditioning.



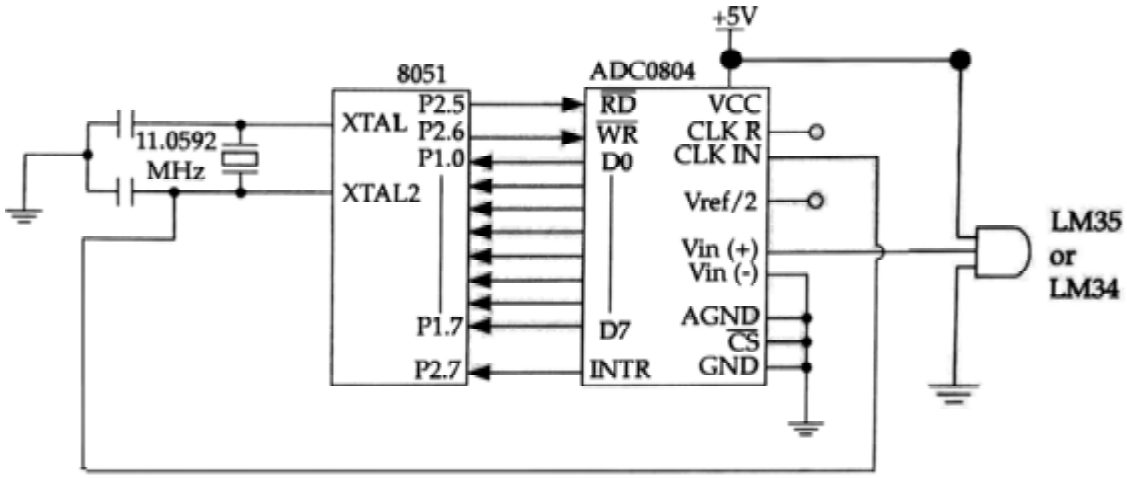
The thermistor changes resistance with temperature. The change of resistance must be translated into voltages by using signal conditioning process to give ADC.

Figure shows connections of an LM35 to an ADCO848 which is interfaced to 8051. The ADC0848 has 8-bit resolution with a maximum of 256 steps. The LM35 (or LM34) produces 10 mV for every degree of temperature change, that means it produces maximum of 2560 mV (2.56 V) for full-scale output and is applied to ADC0848. Therefore, in order to produce the full-scale V_{out} of 2.56 V for the ADC0848, we need to set V_{ref} = 2.56V. in this case ADC0848 outputs directly to the temperature as monitored by the LM35.

The following program is for displaying temperature.

```
        RD PIN P2.6
        WR PIN P2.7
        INTR PIN P2.5
        CS PIN P2.4
        MYDATA EQU P1
        CLR CS
        SETB INTR
BACK:   CLR WR
        SETB WR
HERE:   JB INTR HERE
        CLR RD
        MOV A, MYDATA
        ACALL DIPLAY
        SJMP BACK
```

The interfacing ckt for temperature sensor to the microcontroller via ADC0804 as shown in fig.

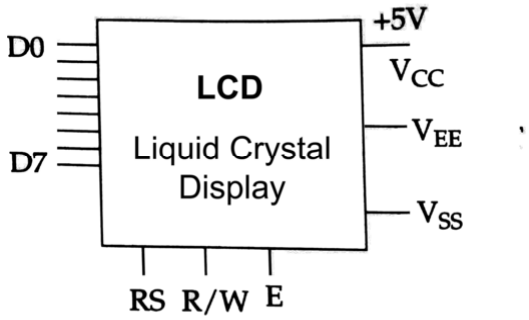


LCD Interfacing and Displaying information on LCD

In recent years the LCD is finding widespread use replacing LEDs. This is due to the following reasons:

- The ability to display numbers, characters. and graphics. In LEDs, which are limited to numbers and a few characters.
- LCD itself, it has refreshing controller, in case of LED must be refreshed by the CPU.
- Ease of programming for characters and graphics.

LCD pin description



V_{CC} V_{SS} :

VCC and VSS provide +5V and ground respectively.

V_E : it is used for controlling LCD contrast.

RS - register select

There are two very important registers inside the LCD. The RS pin is used for their selection as follows.

If RS = 0, the instruction command code register is selected, allowing the user to send a command such as clear display, cursor at home, etc. If RS = 1 the data register is selected, allowing the user to send data to be displayed on the LCD.

R/W - read/write

R/W input allows the user to write information to the LCD or read information from it.

R/W =1 when reading;

R/W = 0 when writing.

E - Enable

The enable pin is used by the LCD to latch information presented to its data pins. When data is supplied to data pins, a high to low pulse must be applied to this pin.

D0 - D7

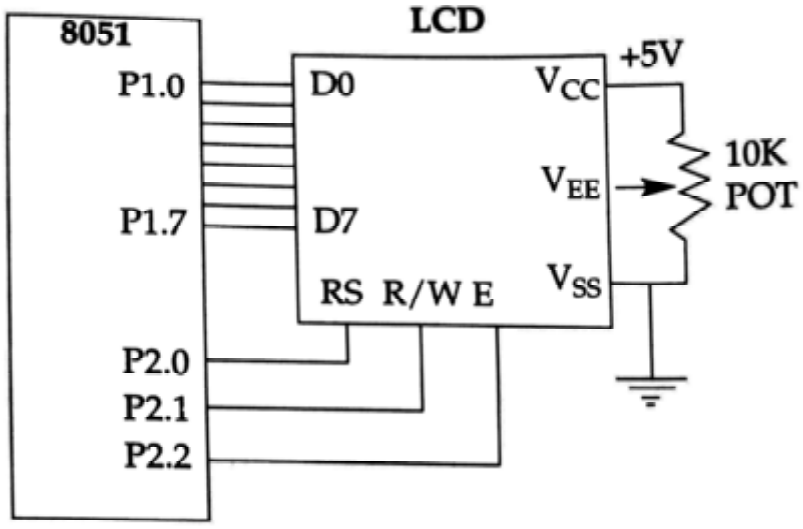
The 8-bit data pins, D0 - D7, are used to send information to The LCD or read the contents of the LCD's internal registers.

To display letters and numbers, we send ASCII codes for while making RS=1. There are also instruction commands to the LCD by making RS=0.

We also use RS=0 to check the busy flag bit (D7) to see the LCD is ready to receive information or not. The busy flag D7 can be read when R/W = 1 and RS = 0. When D7 is 0 the LCD is ready to receive new information.

Code	Command to LCD Instruction Register
01	Clear display screen
02	Return home
08	Display off, cursor off
0E	Display on, cursor blinking
38	2 lines and 5x7 matrix

Figure shows the interfacing connections of LCD to 8051



Program to send command and data to the LCD

```
        R/W BIT P2.0
        RS BIT P2.1
        E BIT P2.2
        MOV A, #38H
        ACALL COMD WRT
        ACALL DELAY
        MOV A, #0EH
        ACALL COMD WRT
        ACALL DELAY
        MOV A, #01H
        ACALL COMD WRT
        ACALL DELAY
        MOV A, #'P'
        ACALL DATA WRT
        ACALL DELAY
        MOV A, #'R'
        ACALL DATA WRT
        ACALL DELAY
        MOV A, #'S'
        ACALL DATA WRT
        ACALL DELAY
AGAIN:  SJMP AGAIN
```

```
DELAY:          MOV R0, #50H
BACK:  MOVR1, #255D
HERE:  DJNZ R1 HERE
        DJNZ R0 BACK
        RET
```

```
COMD WRT:
        CLR RS
        CLR R/W
        SETB E
        MOV P1, A
        CLR E
        RET
```

```
DATA WRT:
        SETB RS
        CLR R/W
        SETB E
        MOV P1, A
        CLR E
        RET
```

Command write:

COMD WRT subroutine is used to send command to the LCD. In this, command placed on port1, RS=0 by CLR RS, write operation selected (R/W) by CLR R/W. high to low signal presented at enable pin E by SETB E and CLR E. with this, the command goes to command register which is located in LCD.

Data write:

DATA WRT subroutine is used to display on the LCD. RS=1 by SETB RS, write operation selected (R/W) by CLR R/W, the data displays on LCD which is on data bus when high to low signal presents at enable pin E.

Send data and command to LCD by checking busy flag.

```
        R/W BIT P2.0
        RS BIT P2.1
        E BIT P2.2
        MOV A, #38H
        ACALL COMD WRT
        ACALL READY
        MOV A, #0EH
        ACALL COMD WRT
        ACALL READY
        MOV A, #01H
        ACALL COMD WRT
        ACALL READY
        MOV A, #'P'
        ACALL DATA WRT
        ACALL READY
        MOV A, #'R'
        ACALL DATA WRT
        ACALL READY
        MOV A, #'S'
        ACALL DATA WRT
        ACALL READY
AGAIN:  SJMP AGAIN
```

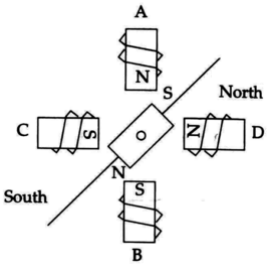
```
READY:          SETB D7
                CLR RS
                SETB R/W
                CLR E
HERE:   JB P1.7  HERE
                SETB E
                RET
```

```
COMD WRT:       CLR RS
                CLR R/W
                SETB E
                MOV P1, A
                CLR E
                RET
```

```
DATA WRT:       SETB RS
                CLR R/W
                SETB E
                MOV P1, A
                CLR E
                RET
```

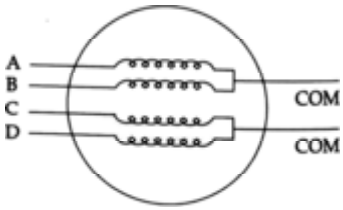
Stepper motor interfacing to 8051

Stepper motors

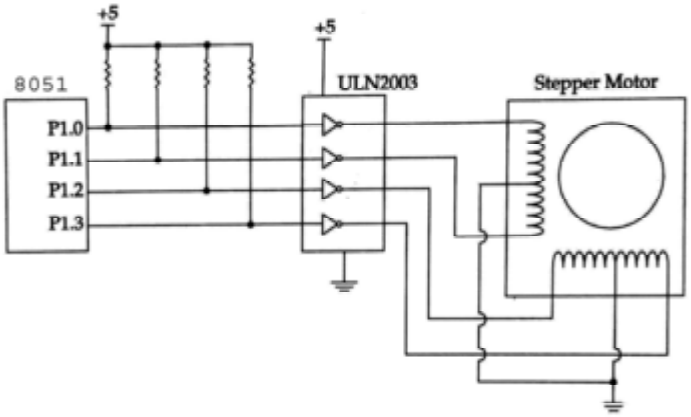


A stepper motor is a widely used device that translates electrical pulses into mechanical movement. In applications such as disk drives, dot matrix printers, and robotics, the stepper motor is used for position control. Stepper motors commonly have a permanent magnet rotor (shaft) surrounded by a stator sown in figure. There are also other steppers called variable reluctance stepper motors that do not have permanent rotor.

The most common stepper motors have four stator windings that are paired with a center tapped common as shown in figure.



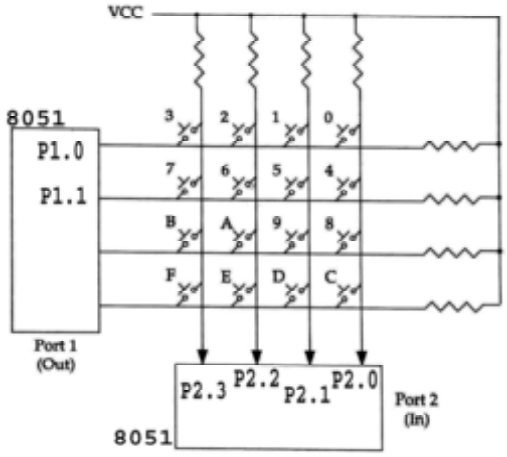
This type of stepper motor is commonly referred to as a four-phase stepper motor. The center tap allows a change of current direction in each of two coils when a winding is grounded, thereby resulting in a polarity change of the stator. The stepper motor rotor moves in a fixed repeatable increment, which allows one to move it to an accurate position. The stator poles are determined by the current sent through the wire coils. As the direction of the current is changed, the polarity is also changed causing the reverse motion of the rotor. Here the stepper motor has a total of 6 leads: 4 leads representing the four stator windings and 2 commons for the center-tapped leads. As the sequence of power is applied to each stator winding, the rotor will rotate.



Keyboard Interface

Once you add a keyboard for your system, you should allow the user to input information to the microcontroller in real time.

In general keyboards are organized in a matrix of rows and columns. Figure shows a 4 x 4 matrix connected to two ports, two sides of matrix are connected to VCC through resisters while 3rd side is connected to 8051 port 2 which is configured as input port, and last side connected to port 1 which is configured as an output port.



Microcontroller keeps scanning the keyboard, if all inputs are high, that means no key is pressed. If one bit is low that means there is a pressed key. System designer setup a look up table contained ASCII code for each key. In this keyboard we have 16 keys to represent the hexa number 0 to F arranged as shown in fig.

To detect a pressed key

The microcontroller grounds all rows by providing 0 to the P1, and then it reads the columns. If data is 1111, no key has been pressed and the process continues until a key press is detected. If one of the column bits has a zero, this means that a key press has occurred.

To identify exact key pressed

Microcontroller Start with the top row by grounding it, then it reads the columns. If the data read is all 1s, no key in that row is activated and the process is moved to the next row until reach the row that has a pressed key. At this stage, microcontroller identifies the row that has a pressed key and can setup the starting address in the look-up table for that row. The last step is finding the column that has a pressed key by rotating the column bits by using carry flag and RRC instruction, when 0 bit found in carry, microcontroller pulls the corresponding code from the look-up table.

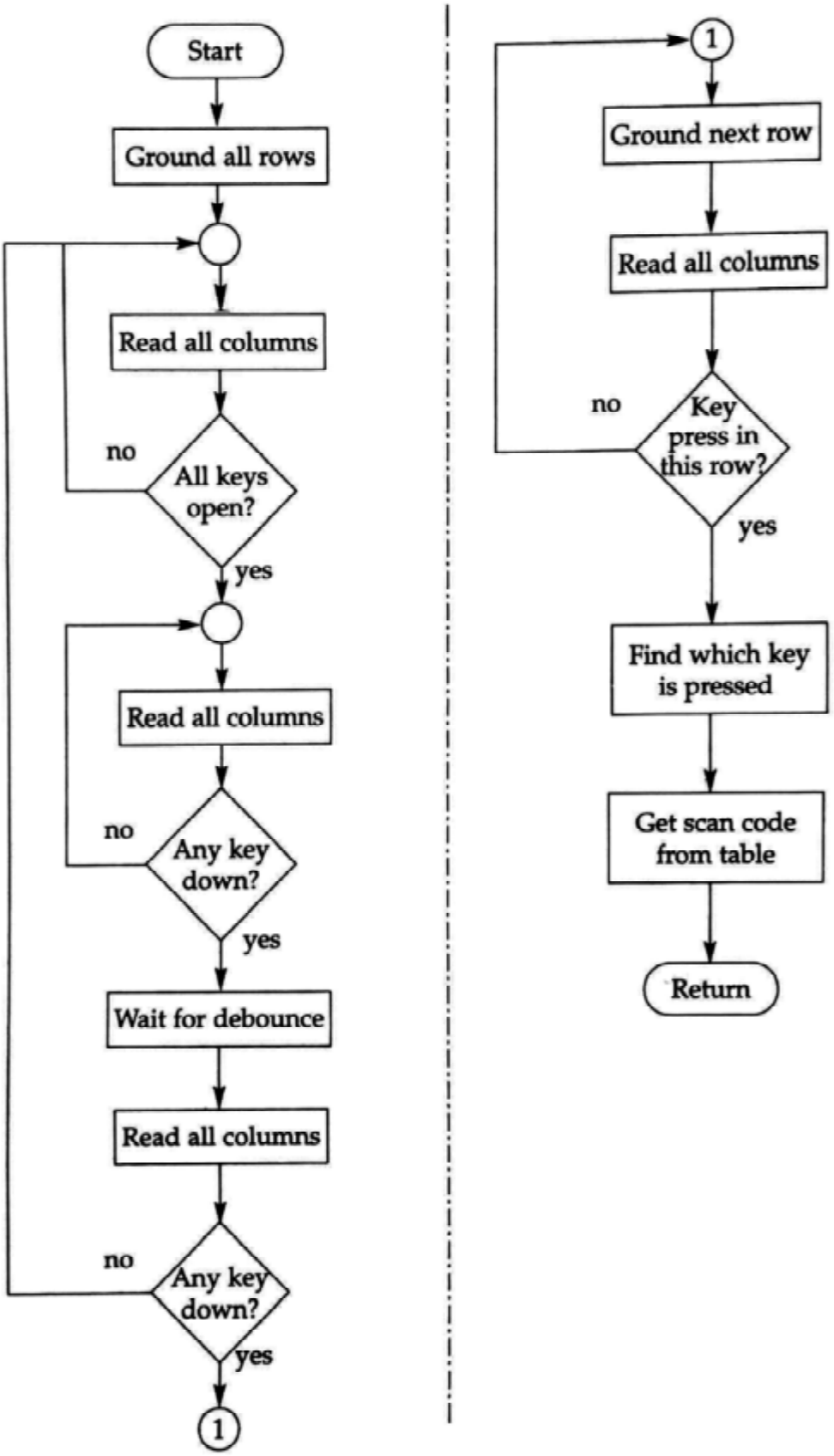
The following code is for scanning the 4 X 4 keyboard.

```
        MOV P2, #0FFH     - P2 as a i/p port
BACK:   MOV P1, #00H      - P1 as a o/p port
GO:     MOV A, P2
        ANL A, #000D1111B
```

```
        CJNE A, #00001111B  GO
GO1:    ACALL DELAY
        MOV A, P2
        ANL A, #0000111lB
        CJNE A, #00001111B, PRS
        SJMP GO1
PRS:    ACALL DELAY
        MOV A, P2
        ANL A, #000011l1B
        CJNE A, #00001111B, OVER
        SJMP GO1
OVER:   MOV P1, #11111110B
        MOV A, P2
        ANL A, #00001111B
        CJNE A, #O00O1111B, ROW0
        MOV P1, #11111101B
        MOV A,P2
        ANL A,#000O1111B
        CJNE A,#D00Oll1lB,ROW_1
        MOV Pl,#11111011B
        MOV A,P2
        ANL A,#00001111B
        CJNE A,#00001111B,ROW_2
        MOV P1,#11110111B
        MOV A,P2
        ANL A,#00001111B
        CJNE A,#00001111B,ROW_3
        LJMP K2
        MOV DPTR,#KCODEO
        SJMP FIND
        MOV DPTR,#KCODE1
        SJMP FIND
        MOV DPTR,#KCODE2
        SJMP FIND
        MOV DPTR,#KCODE3
        RRC A
```

```
        JNC MATCH
        INC DPTR
        SJMP FIND
        CLR A
        MOVC A,OA+DPTR
        MOV P0/A
        LJMP K1

LOOK-UP TABLE POR EACH ROW
        ORG 0000H
        DB ‘0’,’1’,’2’,’3’
        DB ‘4’,’5’,’6’,’7’
        DB ’8’,’9’,’A’,’B’
        DB ‘C’,’D’,’3’,’F’
        END
```


Start
Ground all rows
Read all columns
no
All keys open?
yes
Read all columns
no
Any key down?
yes
Wait for debounce
Read all columns
no
Any key down?
yes
1
1
Ground next row
Read all columns
no
Key press in this row?
yes
Find which key is pressed
Get scan code from table
Return