SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

SCHOOL OF BIO AND CHEMICAL ENGINNEERING

DEPARTMENT OF BIOMEDICAL ENGINEERING

Fundamentals of Microprocessor and Microcontroller – SBMA1501

SBMA1501- FUNDAMENTALS OF MICROPROCESSOR AND MICROCONTROLLER

COURSE OBJECTIVES

- The prime objective of this course is to introduce to the students the fundamentals of Microprocessor Microcontroller and Arduino.
- The students will be equipped with the basic knowledge of microprocessor, microcontroller and Arduino interfacing and their applications.

UNIT 1 INTRODUCTION TO INTEL 8085 9 Hrs.

Evolution of Microprocessor-Architecture of 8085-Instructionformat-Addressingmodes-Basic timing diagram of opcode fetch, memory read, memory write I/O read and I/Owrite-Interruptsof8085-Software interrupts, Hardware interrupts, Priorities of interrupts 8085 based system design.

UNIT 2 INTEL 8085 INTERFACING 9 Hrs.

Interfacing devices-8255 Programmable Peripherals Interface-Architecture & various modes of operation- 8251USART Architecture and programming features-8237, interfacing with ADC and DAC, LCD, keyboard Interface. Application: Stepper Motor Control, Temperature Control.

UNIT 3 INTRODUCTIONT 8051 9 Hrs.

Introduction to 8-bit Microcontrollers – 8051/8051, Microcontroller Architecture – Internal RAM & Internal ROM, Instruction set, Addressing Modes, Simple programs.

UNIT 4 INTERNAL PERIPERALS OF 8051 9 Hrs.

Modes of Timer/Counter operation – Serial Port operation & Modes – Interrupt Structure of 8051 - Memory Interfacing with 8051 – I/O ports- Input and output devices interfacing with 8051.

UNIT 5 ARDUINO UNO 9 Hrs.

Arduino – Architecture, Pin diagram, Programming Structure, Simple program to blink LED, Subroutine, 16x2 LED display, interfacing with Arduino: LCD, Temperature Sensor, Humidity Sensor and ultrasonic sensor.

Max. 45 Hrs.

COURSE OUTCOMES

On completion of the course, student will be able to

CO1 - Understanding architectural principle of 8085 & 8051.

CO2 - Interpret the various peripherals devices with 8085 & 8051 microprocessor

CO3 - Design and implement programs on 8051 microprocessor.

CO4 - Examine various I/O devices with 8051 microcontroller

CO5 - Apply Arduino code and how to Interface various sensors with the Arduino Board.

CO6 - Implement the interface circuit with various sensor and I/O devices with microprocessor and microcontroller and Arduino Board.

TEXT / REFERENCE BOOKS

1. Ramesh S Gaonkar, Microprocessor Architecture, Programming and application with 8085, 4th Edition, Penram International Publishing, New Delhi, 2000.
2. Kennith J.Ayala, 8051 Microcontroller, Thomson, 2005.
3. Charless M.Gilmore, Microprocessor Principle and Application, McGraw Hill Publication, 1995.
4. Nagoor Kani A., Microprocessor & Microcontroller, Tata McGraw Hill, 3rd Edition, 2012.
5. Ram B., Fundamentals of Microprocessors and Microcomputers, Dhanpat Rai Publications, 2001.
6. Michael Mc Roberts, beginning Arduino, 2nd Edition, 2013.

UNIT 1 - Fundamentals of Microprocessor and Microcontroller – SBMA1501

UNIT 1 INTRODUCTION TO INTEL 8085

Evolution of Microprocessor-Architecture of 8085-Instructionformat-Addressingmodes-Basic timing diagram of pcodefetch, memory read, memory write I/O read and I/Owrite-Interruptsof8085-Software interrupts, Hardware interrupts, Priorities of interrupts 8085 based system design.

1.1 8085 EVOLUTION OF MICROPROCESSOR

It can be classified as following types

First generation of processor: 4-bit Microprocessor

The first microprocessor was introduced in 1971 by Intel Corp. It was named Intel 4004 as it was a 4 bit processor. It was a processor on a single chip. It could perform simple arithmetic and logic operations such as addition, subtraction, Boolean AND & Boolean OR. It had a control unit capable of performing control functions like fetching an instruction from memory, decoding it, and generating control pulses to execute it. It was able to operate on 4 bits of data at a time. This first microprocessor was quite a success in industry. Soon other microprocessors were also introduced. Intel introduced the enhanced version of 4004, the 4040. Some other 4 bit processors are International's PPS4 and Toshiba's T3472.

Second generation of processor: 8-bit Microprocessor

The first 8 bit microprocessor which could perform arithmetic and logic operations on 8 bit words was introduced in 1973 again by Intel. This was Intel 8008 and was later followed by an improved version, Intel 8088. Some other 8 bit processors are Zilog-80 and Motorola M6800.

Third generation of processor: 16-bit Microprocessor

The 8-bit processors were followed by 16 bit processors. They are Intel 8086 and 80286.

Fourth generation of processor: 32-bit Microprocessor

The 32 bit microprocessors were introduced by several companies but the most popular one is Intel 80386.

Fifth generation of processor: Pentium Series

Instead of 80586, Intel came out with a new processor namely Pentium processor. Its performance is closer to RISC performance. Pentium was followed by Pentium Pro CPU. Pentium Pro allows allow multiple CPUs in a single system in order to achieve multiprocessing. The MMX extension was added to Pentium Pro and the result was Pentium

II. The low cost version of Pentium II is Celeron. The Pentium III provided high performance floating point operations for certain types of computations by using the SIMD extensions to the instruction set. These new instructions make the Pentium III faster than high-end RISC CPUs.

Interestingly Pentium IV could not execute code faster than the Pentium III when running at the same clock frequency. So Pentium IV had to speed up by executing at a much higher clock frequency.

1.2 ARCHITECTURE OF 8085 MICROPROCESSOR

The functional block diagram or architecture of 8085 Microprocessor is very important as it gives the complete details about a Microprocessor. Fig1.1 shows the Block diagram of a Microprocessor.

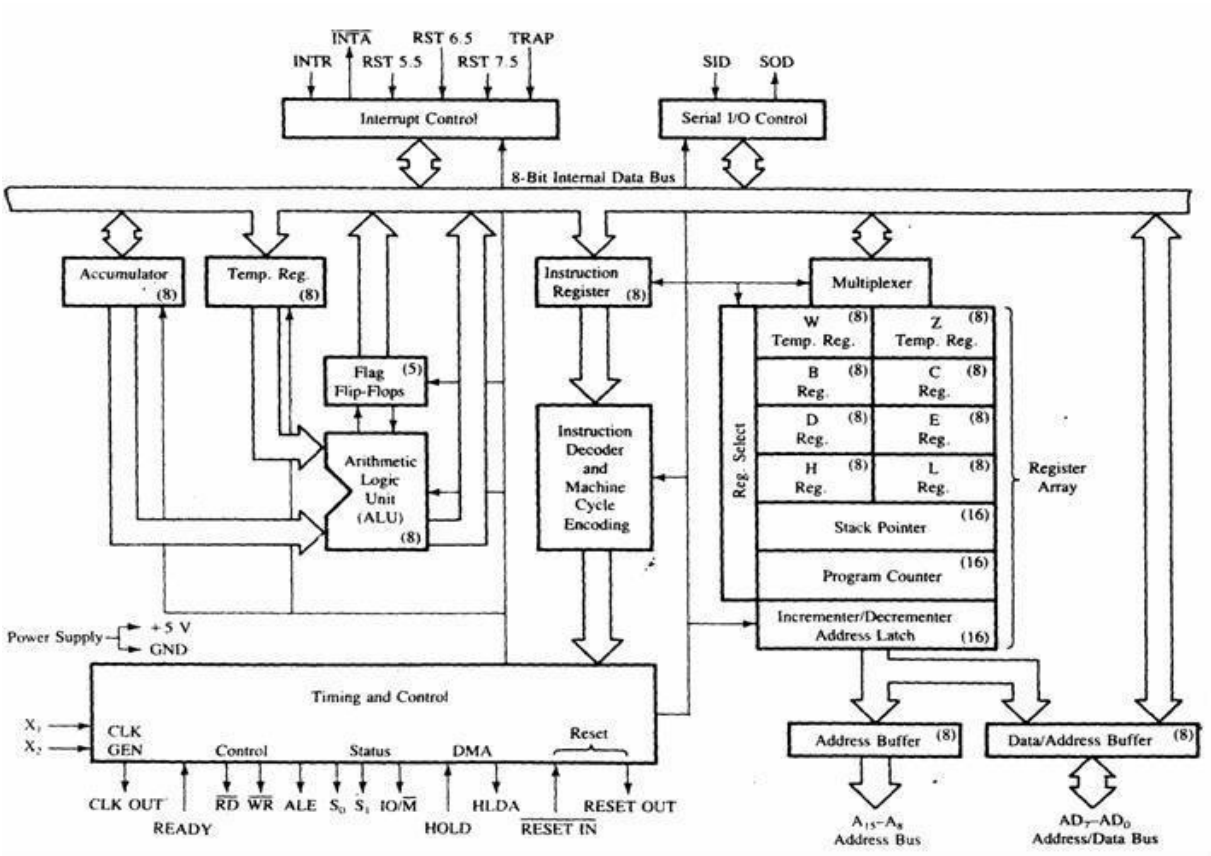


Figure 1.1 8085 Microprocessor Architecture Diagram

8085 consists of the following functional units −

1.2.1 Arithmetic and Logic Unit

As the name suggests, it performs arithmetic and logical operations like Addition, Subtraction, AND, OR, etc. on 8-bit data.

1.2.2 Registers

General purpose register

There are 6 general purpose registers in 8085 processor, i.e. B, C, D, E, H & L. Each register can hold 8-bit data.

These registers can work in pair to hold 16-bit data and their pairing combination is like B-C, D-E & H-L.

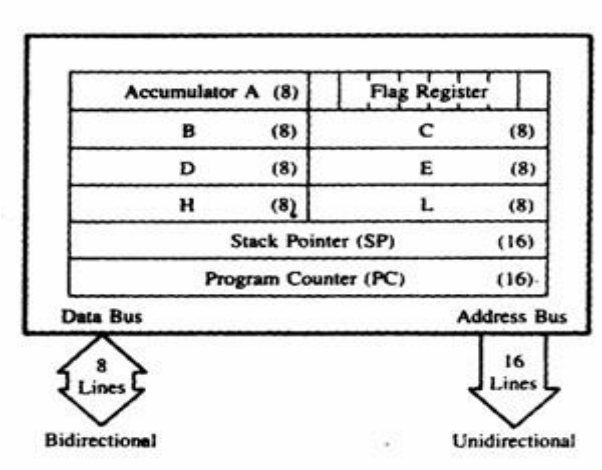


Figure 1.2 8085 Microprocessor Registers set

Accumulator

It is an 8-bit register used to perform arithmetic, logical, I/O & LOAD/STORE operations. It is connected to internal data bus & ALU.

Program counter

It is a 16-bit register used to store the memory address location of the next instruction to be executed. Microprocessor increments the program whenever an instruction is being executed, so that the program counter points to the memory address of the next instruction that is going to be executed.

Stack pointer

It is also a 16-bit register works like stack, which is always incremented/decremented by 2 during push & pop operations.

Temporary register

It is an 8-bit register, which holds the temporary data of arithmetic and logical operations.

Flag register

It is an 8-bit register having five 1-bit flip-flops, which holds either 0 or 1 depending upon the result stored in the accumulator.

These are the set of 5 flip-flops −

- Sign (S)
- Zero (Z)
- Auxiliary Carry (AC)
- Parity (P)
- Carry (C)

Its bit position is shown in the following table –

Table 1.1

D7	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
S	Z		AC		P		CY

Sign Flag (S)

After execution of any arithmetic and logical operation, if D7 of the result is 1, the sign flag is set. Otherwise it is reset. D7 is reserved for indicating the sign; the remaining is the magnitude of number.

If D7 is 1, the number will be viewed as negative number. If D7 is 0, the number will be viewed as positive number.

Zero Flag (z)

If the result of arithmetic and logical operation is zero, then zero flag is set otherwise it is reset.

Auxiliary Carry Flag (AC)

If D3 generates any carry when doing any arithmetic and logical operation, this flag is set. Otherwise it is reset.

Parity Flag (P)

If the result of arithmetic and logical operation contains even number of 1's then this flag will be set and if it is odd number of 1's it will be reset.

Carry flag (CY)

If any arithmetic and logical operation result any carry then carry flag is set otherwise it is reset.

1.2.3 Instruction Register and Decoder

It is an 8-bit register. When an instruction is fetched from memory then it is stored in the Instruction register. Instruction decoder decodes the information present in the Instruction register.

1.2.4 Timing and Control Unit

It provides timing and control signal to the microprocessor to perform operations. Following are the timing and control signals, which control external and internal circuits −

- Control Signals: READY, RD', WR', ALE
- Status Signals: S0, S1, IO/M'
- DMA Signals: HOLD, HLDA
- RESET Signals: RESET IN, RESET OUT
- **ALE** is used for provide control signal to synchronize the components of microprocessor and timing for instruction to perform the operation.

- **RD (Active low) and WR (Active low)** are used to indicate whether the operation is reading the data from memory or writing the data into memory respectively.IO/M(Active low) is used to indicate whether the operation is belongs to the memory or peripherals.

Table 1.2 Read/Write data

IO/M(Active Low)	S1	S2	Data Bus Status(Output)
0	0	0	Halt
0	0	1	Memory WRITE
0	1	0	Memory READ
1	0	1	IO WRITE
1	1	0	IO READ
0	1	1	Opcode fetch
1	1	1	Interrupt acknowledge

8085 System Bus

Typical system uses a number of busses, collection of wires, which transmit binary numbers, one bit per wire. A typical microprocessor communicates with memory and other devices (input and output) using three busses: Address Bus, Data Bus and Control Bus.

Address Bus

The address bus is a group of 16 lines generally identified as A0 to A15.The address bus is unidirectional: bits flow in one direction-from the MPU to peripheral devices. The MPU uses the address bus to perform the first function: identifying a peripheral or a memory location.

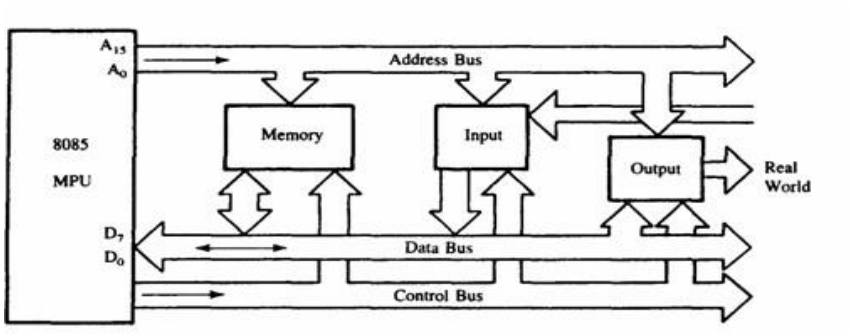


Figure 1.3 Bus interfaced with Microprocessor

Data Bus

The data bus is a group of eight lines used for data flow. These lines are bi-directional - data flow in both directions between the MPU and memory and peripheral devices. The MPU uses the data bus to perform the second function: transferring binary information. The eight data lines enable the MPU to manipulate 8-bit data ranging from 00 to FF (28 = 256 numbers). The largest number that can appear on the data bus is 11111111.

Control Bus

The control bus carries synchronization signals and providing timing signals. The MPU generates specific control signals for every operation it performs. These signals are used to identify a device type with which the MPU wants to communicate.

1.3 8085 PIN DETAILS

The diagram shown in Figure 1.4

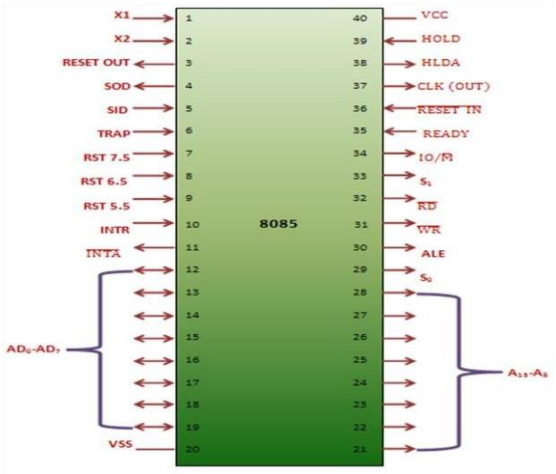


Figure 1.4 Pin Diagram of 8085 Microprocessor

Properties

Single + 5V Supply

4 Vectored Interrupts (One is Non Maskable) Serial In/Serial Out Port

Decimal, Binary, and Double Precision Arithmetic Direct Addressing Capability to 64K bytes of memory

The Intel 8085A is a new generation, complete 8 bit parallel central processing unit (CPU). The 8085A uses a multiplexed data bus. The address is split between the 8bit address bus and the 8bit data bus.

Pin Description

The following describes the function of each pin:

Address bus

A15-A8, it carries the most significant 8-bits of memory/IO address.

Data bus

AD7-AD0, it carries the least significant 8-bit address and data bus.

Control and status signals

These signals are used to identify the nature of operation. There are 3 control signal and 3 status signals.

Three control signals are RD, WR & ALE.

- **RD** − This signal indicates that the selected IO or memory device is to be read and is ready for accepting data available on the data bus.
- **WR** − This signal indicates that the data on the data bus is to be written into a selected memory or IO location.
- **ALE** − It is a positive going pulse generated when a new operation is started by the microprocessor. When the pulse goes high, it indicates address. When the pulse goes down it indicates data.

Three status signals are IO/M, S0 & S1.

IO/M

This signal is used to differentiate between IO and Memory operations, i.e. when it is high indicates IO operation and when it is low then it indicates memory operation.

S1 & S0

These signals are used to identify the type of current operation.

Power supply

There are 2 power supply signals − VCC & VSS. VCC indicates +5v power supply and VSS indicates ground signal.

Clock signals

There are 3 clock signals, i.e. X1, X2, CLK OUT.

- **X1, X2** − A crystal (RC, LC N/W) is connected at these two pins and is used to set frequency of the internal clock generator. This frequency is internally divided by 2.
- **CLK OUT** − This signal is used as the system clock for devices connected with the microprocessor.

Interrupts & externally initiated signals

Interrupts are the signals generated by external devices to request the microprocessor to perform a task. There are 5 interrupt signals, i.e. TRAP, RST 7.5, RST 6.5, RST 5.5, and INTR. We will discuss interrupts in detail in interrupts section.

INTR (Input) **- INTERRUPT REQUEST:** is used as a general purpose interrupt. It is sampled only during the next to the last clock cycle of the instruction. If it is active, the Program Counter (PC) will be inhibited from incrementing and an INTA will be issued. During this cycle a RESTART or CALL instruction can be inserted to jump to the interrupt service routine. The INTR is enabled and disabled by software. It is disabled by Reset and immediately after an interrupt is accepted.

***INTA (Output)*- INTERRUPT ACKNOWLEDGE:** is used instead of (and has the same timing as) RD during the Instruction cycle after an INTR is accepted. It can be used to activate the 8259 Interrupt chip or some other interrupt port.

RST 5.5 RST 6.5 - (Inputs) RST 7.5

RESTART INTERRUPTS; These three inputs have the same timing as INTR except they cause an internal RESTART to be automatically inserted.

RST 7.5 Highest Priority RST 6.5

RST 5.5 Lowest Priority

TRAP (Input)-*Trap* interrupt is a non maskable restart interrupt. It is recognized at the same time as INTR. It is unaffected by any mask or Interrupt Enable. It has the highest priority of any interrupt.

RESET IN − This signal is used to reset the microprocessor by setting the program counter to zero.

- **RESET OUT** − This signal is used to reset all the connected devices when the microprocessor is reset.
- **READY (Input) -** If Ready is high during a read or write cycle, it indicates that the memory or peripheral is ready to send or receive data. If Ready is low, the CPU will wait for Ready to go high before completing the read or write cycle.
- **HOLD (Input) - HOLD :** indicates that another Master is requesting the use of the Address and Data Buses. The CPU, upon receiving the Hold request will relinquish the use of buses as soon as the completion of the current machine cycle. Internal processing can continue. The processor can regain the buses only after the Hold is removed. When the Hold is acknowledged, the Address, Data, RD, WR, and IO/M lines are 3stated.
- **HLDA (Output) - HOLD ACKNOWLEDGE:** indicates that the CPU has received the Hold request and that it will relinquish the buses in the next clock cycle. HLDA goes low after the Hold request is removed. The CPU takes the buses one half clock cycle after HLDA goes low.

Serial I/O signals

There are 2 serial signals, i.e. SID and SOD and these signals are used for serial communication.

- **SOD** (Serial output data line) − The output SOD is set/reset as specified by the SIM instruction.
- **SID** (Serial input data line) − The data on this line is loaded into accumulator whenever a RIM instruction is executed.

Vcc +5 volt supply.

Vss Ground Reference.

1.4 INSTRUCTION FORMAT

An instruction is a command to the microprocessor to perform a given task on a specified data. Each instruction has two parts: one is task to be performed, called the operation code (opcode), and the second is the data to be operated on, called the operand. The operand (or data) can be specified in various ways. It may include 8-bit (or 16-bit) data, an internal register, a memory location, or 8-bit (or 16-bit) address. In some instructions, the operand is implicit.

Instruction format

The 8085 instruction set is classified into the following three groups according to word size:

1. One-word or 1-byte instructions
2. Two-word or 2-byte instructions
3. Three-word or 3-byte instructions

In the 8085, "byte" and "word" are synonymous because it is an 8-bit microprocessor. However, instructions are commonly referred to in terms of bytes rather than words.

One-Byte Instructions

A 1-byte instruction includes the opcode and operand in the same byte. Operand(s) are internal register and are coded into the instruction.

For example:

Table 1.3

Task	Op code	Operand	Binary Code	Hex Code
Copy the contents of the accumulator in the register C.	MOV	C,A	0100 1111	4FH
Add the contents of register B to the contents of the accumulator.	ADD	B	1000 0000	80H
Invert (compliment) each bit in the accumulator.	CMA		0010 1111	2FH

These instructions are 1-byte instructions performing three different tasks. In the first instruction, both operand registers are specified. In the second instruction, the operand B is specified and the accumulator is assumed. Similarly, in the third instruction, the accumulator is assumed to be the implicit operand. These instructions are stored in 8-bit binary format in memory; each requires one memory location.

MOV rd, rs

rd <-- rs copies contents of rs into rd.

Coded as 01 ddd sss where ddd is a code for one of the 7 general registers which is the destination of the data, sss is the code of the source register.

Example: MOV A,B

Coded as 01111000 = 78H = 170 octal (octal was used extensively in instruction design of such processors).

ADD r

A <-- A + r

Two-Byte Instructions

In a two-byte instruction, the first byte specifies the operation code and the second byte specifies the operand. Source operand is a data byte immediately following the opcode. For example:

Table 1.4

Task	Opcode	Operand	Binary Code	Hex Code	
Load an 8-bit data byte in the accumulator.	MVI	A, Data	0011 1110	3E Data	First Byte Second Byte

Assume that the data byte is 32H. The assembly language instruction is written as

Table 1.5

Mnemonics	Hex code
MVI A, 32H	3E 32H

The instruction would require two memory locations to store in memory. MVI r, data r <--data

Example: MVI A,30H coded as 3EH 30H as two contiguous bytes. This is an example of immediate addressing.

ADI data

A <-- A + data

OUT port where port is an 8-bit device address. (Port) <-- A. Since the byte is not the data but points directly to where it is located this is called direct addressing.

Three-Byte Instructions: In a three-byte instruction, the first byte specifies the opcode, and the following two bytes pecify the 16-bit address and the third byte is the high-order address opcode + data byte + data byte

For example:

Table 1.6

Task	Opcode	Operand	Binary code	Hex Code	
Transfer the program sequence to the memory location 2085H.	JMP	2085H		C3	First byte
			1100 0011		
			1000 0101	85	Second Byte
			0010 0000	20	Third Byte

This instruction would require three memory locations to store in memory. Three byte instructions - opcode + data byte + data byte

LXI rp, data16

rp is one of the pairs of registers BC, DE, HL used as 16-bit registers. The two data bytes are 16-bit data in L H order of significance.

rp <-- data16

Example:

LXI H,0520H coded as 21H 20H 50H in three bytes. This is also immediate addressing.

LDA addr

A <-- (addr) Addr is a 16-bit address in L H order. Example: LDA 2134H coded as 3AH 34H 21H. This is also an example of direct addressing.

Classification based on functionality

I. Data transfer operations: This group of instructions copies data from source to destination. The content of the source is not altered.

II. Arithmetic operations: Instructions of this group perform operations like addition, subtraction, increment & decrement. One of the data used in arithmetic operation is stored in accumulator and the result is also stored in accumulator.

III. Logical operations: Logical operations include AND, OR, EXOR, NOT. The operations like AND, OR and EXOR uses two operands, one is stored in accumulator and other can be any register or memory location. The result is stored in accumulator. NOT operation requires single operand, which is stored in accumulator.

IV. Branching operations: Instructions in this group can be used to transfer program sequence from one memory location to another either conditionally or unconditionally.

V. Machine control operations: Instruction in this group control execution of other instructions and control operations like interrupt, halt etc.

1.5 8085 ADDRESSING MODES

The instructions MOV B, A or MVI A, 82H are to copy data from a source into a destination. In these instructions the source can be a register, an input port, or an 8-bit number (00H to FFH). Similarly, a destination can be a register or an output port. The sources and destination are operands. The various formats for specifying operands are called the ADDRESSING MODES. For 8085, they are:

1. Immediate addressing.
2. Register addressing.
3. Direct addressing.
4. Indirect addressing.

Immediate addressing mode

In this mode, the 8/16-bit data is specified in the instruction itself as one of its operand. **For example:** MVI R, 35F: means 35F is copied into register R.

Register addressing mode

In this mode, the data is copied from one register to another. **For example:** MOV K, B: means data in register B is copied to register K.

Direct addressing mode

In this mode, the data is directly copied from the given address to the register. **For example:** LDB 5000K: means the data at address 5000K is copied to register B.

Indirect addressing mode

In this mode, the data is transferred from one register to another by using the address pointed by the register. **For example:** MOV K, B: means data is transferred from the memory address pointed by the register to the register K.

Implied addressing mode

This mode doesn't require any operand; the data is specified by the opcode itself. **For example:** CMP.

1.6 TIMING DIAGRAMS OF 8085

It is one of the best ways to understand to process of micro-processor/controller. With the help of timing diagram we can understand the working of any system, step by step working of each instruction and its execution, etc. It is the graphical representation of process in steps with respect to time. The timing diagram represents the clock cycle and duration, delay, content of address bus and data bus, type of operation ie. Read/write/status signals.

Rules to identify number of machine cycles in an instruction:

1. If an addressing mode is direct, immediate or implicit then No. of machine cycles = No. of bytes.
2. If the addressing mode is indirect then No. of machine cycles = No. of bytes + 1. Add +1 to the No. of machine cycles if it is memory read/write operation.
3. If the operand is 8-bit or 16-bit address then, No. of machine cycles = No. of bytes +1.
4. These rules are applicable to 80% of the instructions of 8085.

Opcode fetch

The microprocessor requires instructions to perform any particular action. In order to perform these actions microprocessor utilizes opcode which is a part of an instruction which provides detail to microprocessor shown in Figure 1.5

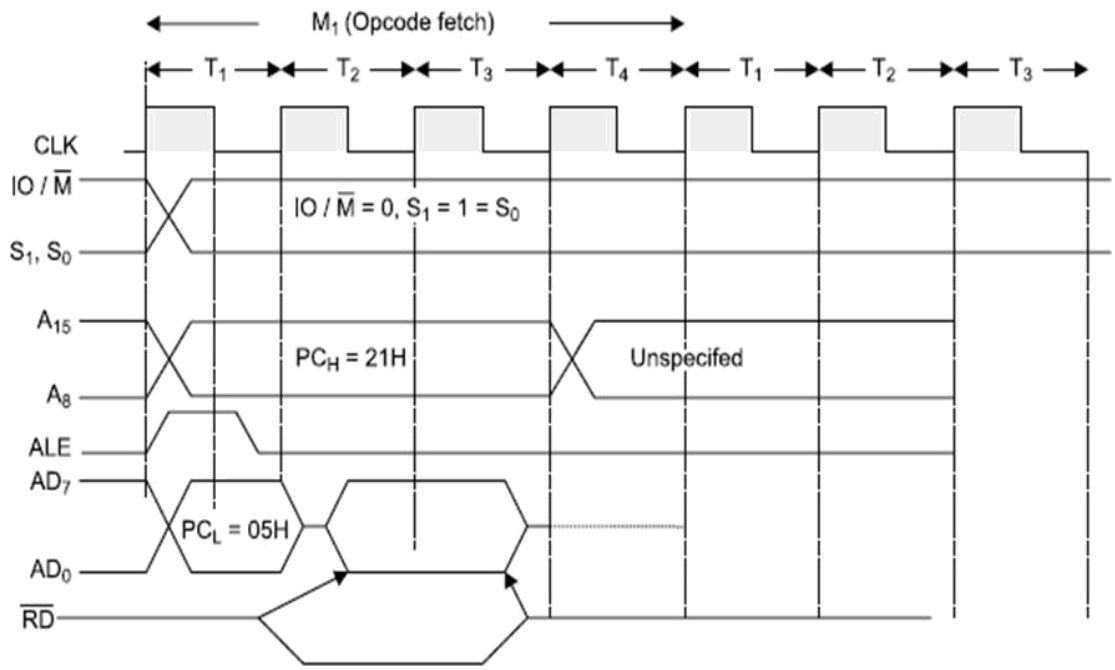


Figure 1. 5 Opcode Fetch Timing Diagram

Opcode Fetch Timing Operation

- During T1 state, microprocessor uses IO/M(bar), S0, S1 signals are used to instruct microprocessor to fetch opcode.
- Thus when IO/M(bar)=0, S0=S1= 1, it indicates opcode fetch operation.
- During this operation 8085 transmits 16-bit address and also uses ALE signal for address latching.
- At T2 state microprocessor uses read signal and make data ready from that memory location to read opcode from memory and at the same time program counter increments by 1 and points next instruction to be fetched.
- In this state microprocessor also checks READY input signal, if this pin is at low logic level ie. '0' then microprocessor adds wait state immediately between T2 and T3.

- At T3, microprocessor reads opcode and store it into instruction register to decode it further. During T4 microprocessor performs internal operation like decoding opcode and providing necessary actions.
- The opcode is decoded to know whether T5 or T6 states are required, if they are not required then the processor performs next operation.

Memory Read Timing Diagram

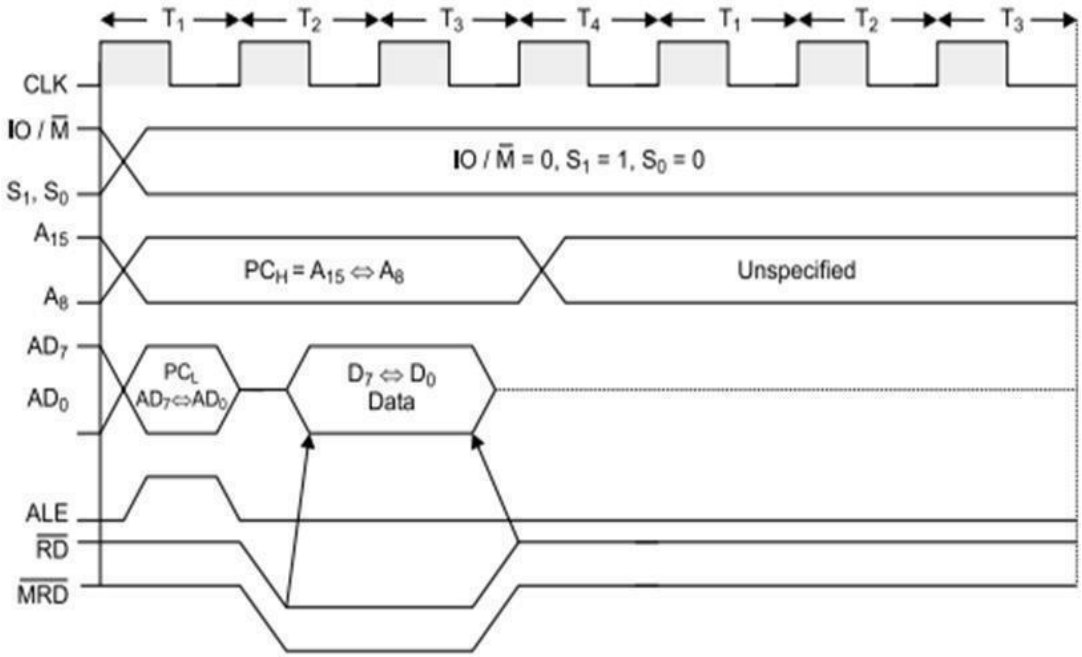


Figure.1.6 Memory Read Timing Diagram

Operation

- It is used to fetch one byte from the memory.
- It requires 3 T-States.
- It can be used to fetch operand or data from the memory.
- During T1, A8-A15 contains higher byte of address. At the same time ALE is high.

  Therefore Lower byte of address A0-A7 is selected from AD0-AD7.

- Since it is memory ready operation, IO/M(bar) goes low.
- During T2 ALE goes low, RD(bar) goes low. Address is removed from AD0-AD7 and data D0-D7 appears on AD0-AD7.
- During T3, Data remains on AD0-AD7 till RD(bar) is at low signal.

Memory Write Timing Diagram

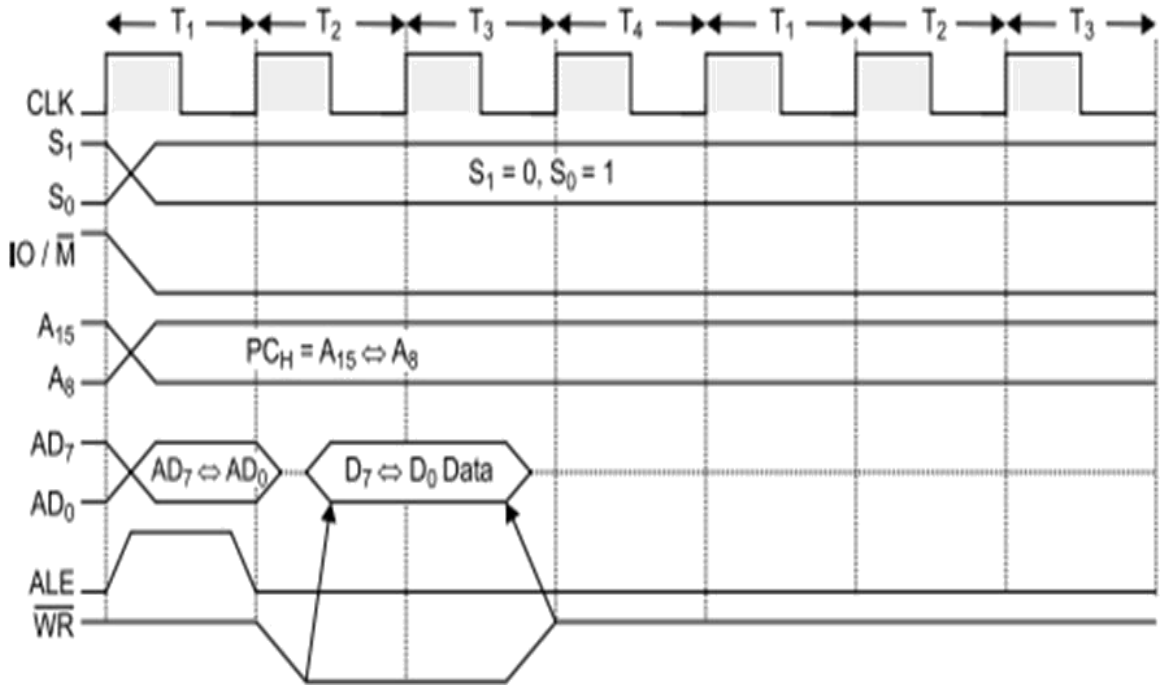


Figure 1.7 Memory Read Write Diagram

Operation

- It is used to send one byte into memory.
- It requires 3 T-States.
- During T1, ALE is high and contains lower address A0-A7 from AD0-AD7.
- A8-A15 contains higher byte of address. As it is memory operation, IO/M goes low.
- During T2, ALE goes low, WR goes low and Address is removed from AD0-AD7 and then data appears on AD0-AD7.
- Data remains on AD0-AD7 till WR is low.

IO Read Timing Diagram

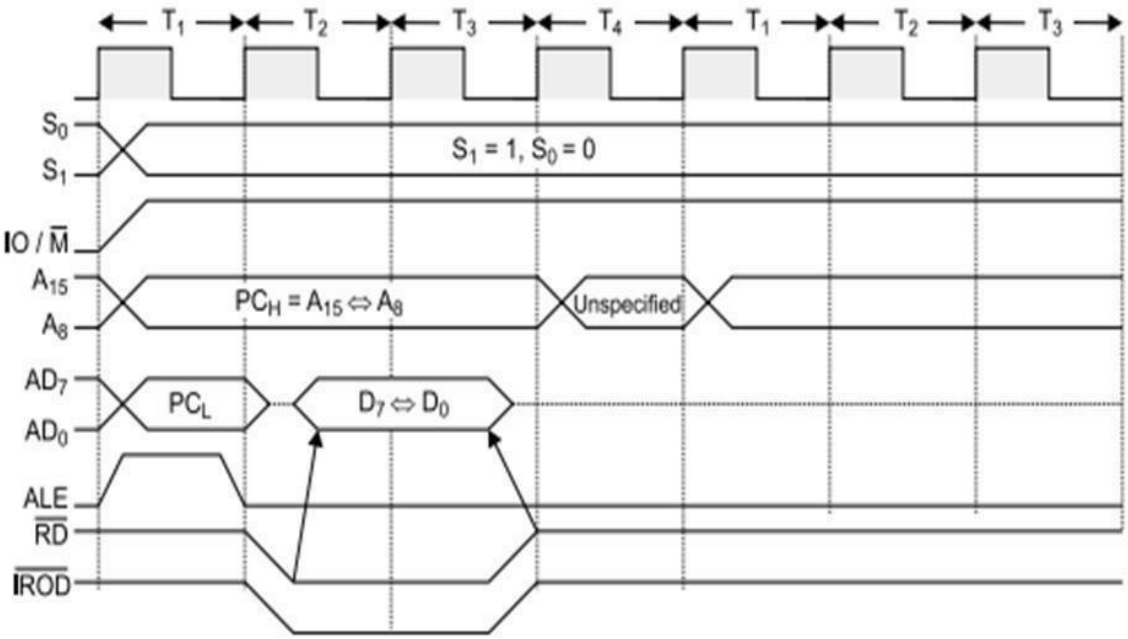


Figure 1.8 I/O Read Timing Diagram I/O Write Timing Diagram

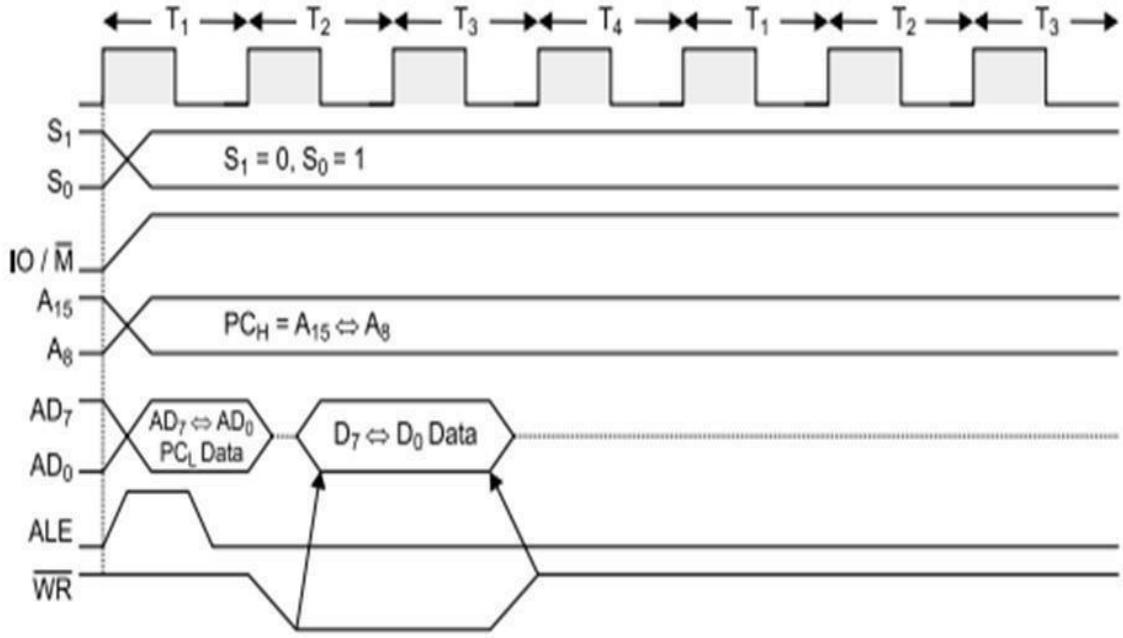


Figure 1.9 I/O Write Timing Diagram

- It is used to writ one byte into IO device.
- It requires 3 T-States.

- During T1, the lower byte of address is duplicated into higher order address bus A8-A15.
- ALE is high and A0-A7 address is selected from AD0- AD7. o As it is an IO operation IO/M goes low.
- During T2, ALE goes low, WR goes low and data appears on AD0-AD7 to write data into IO device.
- During T3, Data remains on AD0-AD7 till WR is low.

1.7 INSTRUCTION SET OF 8085

An Instruction is a command given to the computer to perform a specified operation on given data. The instruction set of a microprocessor is the collection of the instructions that the microprocessor is designed to execute. The instructions described here are of Intel 8085. These instructions are of Intel Corporation. They cannot be used by other microprocessor manufactures. The programmer can write a program in assembly language using these instructions. These instructions have been classified into the following groups:

1. Data Transfer Group
2. Arithmetic Group
3. Logical Group
4. Branch Control Group
5. I/O and Machine Control Group

1.7.1 Data Transfer Instruction

Instructions, which are used to transfer data from one register to another register, from memory to register or register to memory, come under this group. Examples are: MOV, MVI, LXI, LDA, STA etc. When an instruction of data transfer group is executed, data is transferred from the source to the destination without altering the contents of the source. For example, when MOV A, B is executed the content of the register B is copied into the register A, and the content of register B remains unaltered. Similarly, when LDA 2500 is executed the content of the memory location 2500 is loaded into the accumulator. But the content of the memory location 2500 remains unaltered.

1.7.1.1 MOV Rd, Rs - Move Data; Move the content of the from source register to destination register.

1.7.1.2 MOV Rd, M -Move the content of memory register to destination register.

1.7.1.3 MOV M, Rs. -Move the content of register to memory.

1.7.1.4 MVI r, data. -Move immediate data to register.

1.7.1.5 MVI M, data- Move immediate data to memory.

1.7.1.6 LXI rp, data 16- Load register pair immediate.

1.7.1.7 LDA addr- Load Accumulator direct.

1.7.1.8 STA addr- Store accumulator direct.

1.7.1.9 LHLD addr- Load H-L pair direct

1.7.1.10 SHLD addr- Store H-L pair direct

1.7.1.11 LDAX rp. -LOAD accumulator indirect

1.7.1.12 STAX rp- Store accumulator indirect

1.7.1.13 XCHG- Exchange the contents of H-L with D-E pair [H-L] <--> [D-E].

1.7.2 Arithmetic Instructions

The instructions of this group perform arithmetic operations such as addition, subtraction; increment or decrement of the content of a register or memory. Examples are: ADD, SUB, INR, DAD etc.

Logical Group: The Instructions under this group perform logical operation such as AND, OR, compare, rotate etc. Examples are: ANA, XRA, ORA, CMP, and RAL etc.

Branch Control Group: This group includes the instructions for conditional and unconditional jump, subroutine call and return, and restart. Examples are: JMP, JC, JZ, CALL, CZ, RST etc.

I/O and Machine Control Group: This group includes the instructions for input/output ports, stack and machine control. Examples are: IN, OUT, PUSH, POP, and HLT etc.

Intel 8085 Instructions

i. ADD r. (Add register to accumulator) [A] [A] + [r].

ii. ADD M. (Add memory to accumulator) [A] [A] + [[H-L]].

iii. ADC r. (Add register with carry to accumulator). [A] [A] + [r] + [CS].

iv. ADC M. (Add memory with carry to accumulator) [A] [A] + [[H-L]] [CS].

v. ADI data (Add immediate data to accumulator) [A] [A] + data.

vi. ACI data (Add with carry immediate data to accumulator). [A] [A]+data+ [CS].

vii. DAD rp. (Add register paid to H-L pair). [H-L] [H-L] + [rp].

viii. SUB r. (Subtract register from accumulator). [A] [A] – [r].

ix. SUB M. (Subtract memory from accumulator). [A] [A] – [[H-L]].

x. SBB r. (Subtract register from accumulator with borrow). [A] [A] – [r] –[CS].

xi. SBB M. (Subtract memory from accumulator with borrow). [A] [A] – [[H- L]] – [CS].

xii. SUI data. (Subtract immediate data from accumulator) [A] [A] – data.

xiii. SBI data. (Subtract immediate data from accumulator with borrow). [A] – data – [CS].

xiv. INR r (Increment register content) [r][r] +1.

xv. INR M. (Increment memory content) [[H-L]] [[H-L]] + 1.

xvi. DCR r. (Decrement register content). [r] [r] – 1.

xvii. DCR M. (Decrement memory content) [[H-L]] [[H-L]] – 1.

xviii. INX rp. (Increment register pair) [rp] [rp] – 1.

xix DCX rp (Decrement register pair) [rp] [rp] -1.

xx. DAA (Decimal adjust accumulator).

The instruction DAA is used in the program after ADD, ADI, ACI, ADC, etc instructions. After the execution of ADD, ADC, etc instructions the result is in hexadecimal and it is placed in the accumulator. The DAA instruction operates on this result and gives the final result in the decimal system. It uses carry and auxiliary carry for decimal adjustment. 6

is added to 4 LSBs of the content of the accumulator if their value lies in between A and F or the AC flag is set to 1. Similarly, 6 is also added to 4 MSBs of the content of the accumulator if their value lies in between A and F or the CS flag is set to 1. All status flags are affected. When DAA is used data should be in decimal numbers.

1.7.3 Logical Instructions

i. ANA r. (AND register with accumulator) [A] [A] ^ [r].

ii. ANA M. (AND memory with accumulator). [A] [A] ^ [[H-L]].

iii. ANI data. (AND immediate data with accumulator) [A] [A] ^ data.

iv. ORA r. (OR register with accumulator) [A] [A] v [r].

v. ORA M. (OR memory with accumulator) [A] [A] v [[H-L]]

vi. ORI data. (OR immediate data with accumulator) [A] [A] v data.

vii. XRA r. (EXCLUSIVE – OR register with accumulator) [A][A] v [r]

viii. XRA M. (EXCLUSIVE-OR memory with accumulator) [A] [A] v [[H-L]]

ix. XRI data. (EXCLUSIVE-OR immediate data with accumulator) [A] [A] v data.

x. CMA. (Complement the accumulator) [A] [A]

xi. CMC. (Complement the carry status) [CS] [CS]

xii. STC. (Set carry status) [CS] 1.

xiii. CMP r. (Compare register with accumulator) [A] – [r]

xiv. CMP M. (Compare memory with accumulator) [A] – [[H-L]]

xv. CPI data. (Compare immediate data with accumulator) [A] – data.

xvi. RLC (Rotate accumulator left) [An+1] [An], [A0] [A7], [CS] [A7].

The content of the accumulator is rotated left by one bit. The seventh bit of the accumulator is moved to carry bit as well as to the zero bit of the accumulator. Only CS flag is affected.

xvii. RRC. (Rotate accumulator right) [A7] [A0], [CS] [A0], [An] [An+1]. The content of the accumulator is rotated right by one bit. The zero bit of the accumulator is moved to the seventh bit as well as to carry bit. Only CS flag is affected.

xviii. RAL. (Rotate accumulator left through carry) [An+1] [An], [CS] [A7], [A0] [CS].

xix. RAR. (Rotate accumulator right through carry) [An][An+1], [CS] [A0], [A7] [CS]

1.7.4 Branching Instruction

1.7.4.1 JMP addr (label). (Unconditional jump: jump to the instruction specified by the address). [PC] Label.

1.7.4.2 Conditional Jump addr (label): After the execution of the conditional jump instruction the program jumps to the instruction specified by the address (label) if the specified condition is fulfilled. The program proceeds further in the normal sequence if the specified condition is not fulfilled. If the condition is true and program jumps to the specified label, the execution of a conditional jump takes 3 machine cycles: 10 states. If condition is not true, only 2 machine cycles; 7 states are required for the execution of the instruction.

1.7.4.2.1 JZ addr (label). (Jump if the result is zero)

1.7.4.2.2 JNZ addr (label) (Jump if the result is not zero)

1.7.4.2.3 JC addr (label). (Jump if there is a carry)

1.7.4.2.4 JNC addr (label). (Jump if there is no carry)

1.7.4.2.5 JP addr (label). (Jump if the result is plus)

1.7.4.2.6 JM addr (label). (Jump if the result is minus)

1.7.4.2.7 JPE addr (label) (Jump if even parity)

1.7.4.2.8 JPO addr (label) (Jump if odd parity)

1.7.4.3 CALL addr (label) (Unconditional CALL: call the subroutine identified by the operand) CALL instruction is used to call a subroutine. Before the control is transferred to the subroutine, the address of the next instruction of the main program is saved in the stack. The content of the stack pointer is decremented by

two to indicate the new stack top. Then the program jumps to subroutine starting at address specified by the label.

1.7.4.4 RET (Return from subroutine)

1.7.4.5 RST n (Restart) Restart is a one-word CALL instruction. The content of the program counter is saved in the stack. The program jumps to the instruction starting at restart location.

1.7.5 Stack, I/O and Machine Control Instruction

1.7.5.1 IN port-address. (Input to accumulator from I/O port) [A] [Port]

1.7.5.2 OUT port-address (Output from accumulator to I/O port) [Port] [A]

1.7.5.3 PUSH rp (Push the content of register pair to stack)

1.7.5.3.1 PUSH PSW (PUSH Processor Status Word)

1.7.5.4 POP rp (Pop the content of register pair, which was saved, from the stack)

1.7.5.5 POP PSW (Pop Processor Status Word)

1.7.5.6 HLT (Halt)

1.7.5.7 XTHL (Exchange stack-top with H-L)

1.7.5.8 SPHL (Move the contents of H-L pair to stack pointer)

1.7.5.9 EI (Enable Interrupts)

1.7.5.10 DI (Disable Interrupts)

1.7.5.11 SIM (Set Interrupt Masks)

1.7.5.12 RIM (Read Interrupt Masks)

1.7.5.13 NOP (No Operation)

1.8 INTERRUPTS IN 8085 MICROPROCESSOR

Interrupts are the signals generated by the external devices to request the microprocessor to perform a task. There are 5 interrupt signals, i.e. TRAP, RST 7.5, RST 6.5, RST 5.5, and INTR.

Interrupt are classified into following groups based on their parameter −

- **Vector interrupt** − In this type of interrupt, the interrupt address is known to the processor. **For example:** RST7.5, RST6.5, RST5.5, TRAP.
- **Non-Vector interrupt** − In this type of interrupt, the interrupt address is not known to the processor so, the interrupt address needs to be sent externally by the device to perform interrupts. **For example:** INTR.
- **Maskable interrupt** − In this type of interrupt, we can disable the interrupt by writing some instructions into the program. **For example:** RST7.5, RST6.5, RST5.5.
- **Non-Maskable interrupt** − In this type of interrupt, we cannot disable the interrupt by writing some instructions into the program. **For example:** TRAP.
- **Software interrupt** − In this type of interrupt, the programmer has to add the instructions into the program to execute the interrupt. There are 8 software interrupts in 8085, i.e. RST0, RST1, RST2, RST3, RST4, RST5, RST6, and RST7.
- **Hardware interrupt** − There are 5 interrupt pins in 8085 used as hardware interrupts, i.e. TRAP, RST7.5, RST6.5, RST5.5, INTA.

Note − INTA is not an interrupt, it is used by the microprocessor for sending acknowledgement. TRAP has the highest priority, then RST7.5 and so on.

Interrupt Service Routine (ISR)

A small program or a routine that when executed, services the corresponding interrupting source is called an ISR.

TRAP

It is a non-maskable interrupt, having the highest priority among all interrupts. Bydefault, it is enabled until it gets acknowledged. In case of failure, it executes as ISR and sends the data to backup memory. This interrupt transfers the control to the location 0024H.

RST7.5

It is a maskable interrupt, having the second highest priority among all interrupts. When this interrupt is executed, the processor saves the content of the PC register into the stack and branches to 003CH address.

RST 6.5

It is a maskable interrupt, having the third highest priority among all interrupts. When this interrupt is executed, the processor saves the content of the PC register into the stack and branches to 0034H address.

RST 5.5

It is a maskable interrupt. When this interrupt is executed, the processor saves the content of the PC register into the stack and branches to 002CH address.

INTR

It is a maskable interrupt, having the lowest priority among all interrupts. It can be disabled by resetting the microprocessor.

When **INTR signal goes high**, the following events can occur −

- The microprocessor checks the status of INTR signal during the execution of each instruction.
- When the INTR signal is high, then the microprocessor completes its current instruction and sends active low interrupt acknowledge signal.
- When instructions are received, then the microprocessor saves the address of the next instruction on stack and executes the received instruction.

Vectored Interrupts

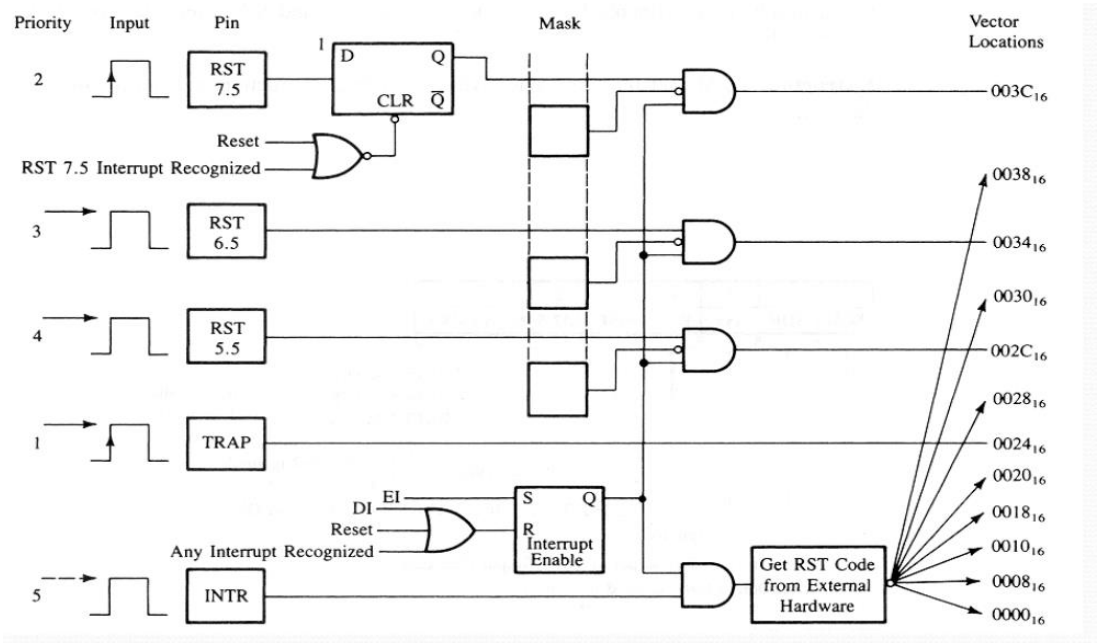


Figure 1.10 Vectored Interrupts

Masking of Interrupts

- These three interrupts are masked at two levels:
  - Through the Interrupt Enable flip flop and the EI/DI instructions.
- The Interrupt Enable flip flop controls the whole maskable interrupt process.
  - Through individual mask flip flops that control the availability of the individual interrupts.
  - These flip flops control the interrupts individually.

1.9 8085 MICROPROCESSOR BASED SYSTEM DESIGN

- The microprocessor is a semiconductor device (Integrated Circuit) manufactured by the VLSI (Very Large Scale Integration) technique. It includes the ALU, register arrays and control circuit on a single chip.
- A system designed using a microprocessor as its CPU is called a microcomputer. The Microprocessor based system (single board microcomputer) consists of microprocessor as CPU, semiconductor memories like EPROM and RAM, input device, output device and interfacing devices.

- The memories, input device, output device and interfacing devices are called peripherals. The popular input devices are keyboard and floppy disk and the output devices are printer, LED/LCD displays, CRT monitor,

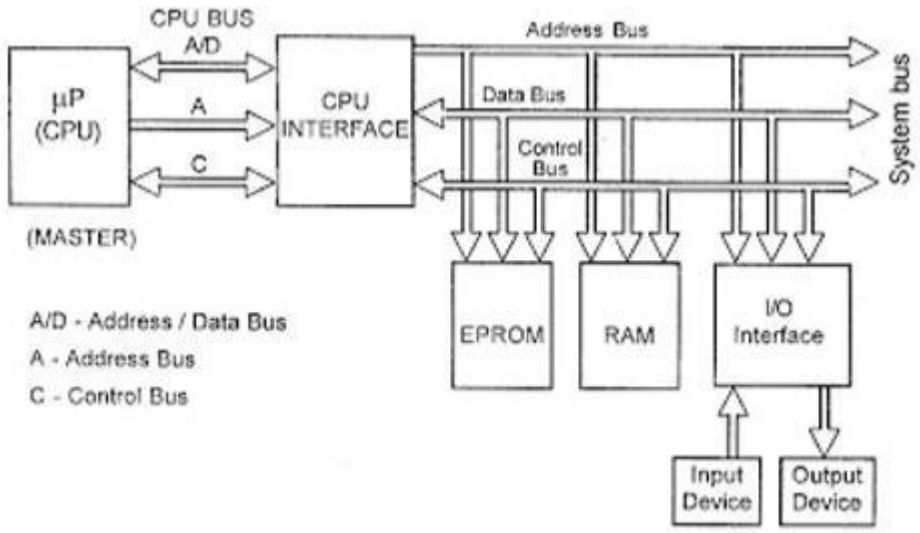


Figure 1.11 Microprocessor interfacing with External Memory

- In the µP based system, the microprocessor is the master and all other peripherals are slaves. The master controls all the peripherals and initiates all operations. The work done by the processor can be classified into the following three groups.
- Work done internal to the processor
- Work done external to the processor
- Operations initiated by the slaves or peripherals.
- The work done internal to the processors are addition, subtraction, logical operations, data transfer operations, etc.
- The work done external to the processor are reading/writing the memory and reading/writing the I/O devices or the peripherals. If the peripheral requires the attention of the master then it can interrupt the master and initiate an operation.
- The microprocessor is the master, which controls all the activities of the system. To perform a specific job or task, the microprocessor has to execute a program stored in memory. The program consists of a set of instructions. It issues address and control signals and fetches the instruction and data from memory.

BUSES

The buses are group of lines that carries data, address or control signals.

- The CPU Bus has multiplexed lines, i.e., same line is used to carry different signals
- The CPU interface is provided to demultiplex, the multiplexed lines, to generate chip select signals and additional control signals.
- The system bus has separate lines for each signal.

All the slaves in the system are connected to the same system bus. At any time instant communication takes place between the master and one of the slaves.

Peripheral Devices

- The EPROM memory is used to store permanent programs and data.
- The RAM memory is used to store temporary programs and data.
- The input device is used to enter the program, data and to operate the system.
- The output device is used for examining the results.

Since the speed of I/O devices does not match with the speed of microprocessor, an interface device is provided between system bus and I/O devices. Generally I/O devices are slow devices.

Reference Books

1. Ramesh S Gaonkar, Microprocessor Architecture, Programming and application with 8085, 4» Edition, Penram International Publishing, New Delhi, 2000
2. Kennith J. Ayala, 8051 Microcontroller, Thomson, 2005.
3. Dougles V. Hall, Microprocessor and Interfacing, Tata MC Graw Hill Publication, 2.‹Edition, 1992.
4. Charless M. Gilmore, “Microprocessor Principle and application, McGraw Hill publication, 1995.
5. A.NagoorKani, Microprocessor & Microcontroller, Tata Mc Graw Hill, 3«Edition, 2012
6. B. Ram, Fundamentals of Microprocessors and Microcomputers, Dhanpat Rai Publications, 2001.

SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

SCHOOL OF BIO AND CHEMICAL ENGINNEERING

DEPARTMENT OF BIOMEDICAL ENGINEERING

UNIT 2 - Fundamentals of Microprocessor and Microcontroller – SBMA1501

UNIT 2 INTEL 8085 INTERFACING

Interfacing devices-8255 Programmable Peripherals Interface-Architecture & various modes of operation- 8251USART Architecture and programming features-8237, interfacing with ADC and DAC, LCD, keyboard Interface. Application: Stepper Motor Control, Temperature Control.

Programmable peripheral Interface 8255 (PPI)

The Intel 8255 (or i8255) Programmable Peripheral Interface (PPI) chip is a peripheral chip, is used to give the CPU access to programmable parallel I/O. It can be programmable to transfer data under various conditions from simple I/O to interrupt I/O. it is flexible versatile and economical (when multiple I/O ports are required) but somewhat complex. It is an important general purpose I/O device that can be used with almost any microprocessor. Functional block of 8255 – Programmable Peripheral Interface (PPI).

The 8255A has 24 I/O pins that can be grouped primarily in two 8-bit parallel ports: A and B with the remaining eight bits as port C. The eight bits of port C can be used as individual bits or be grouped in to 4-bit ports: CUpper (Cu) and CLower (CL) as in Figure. The function of these ports is defined by writing a control word in the control register.

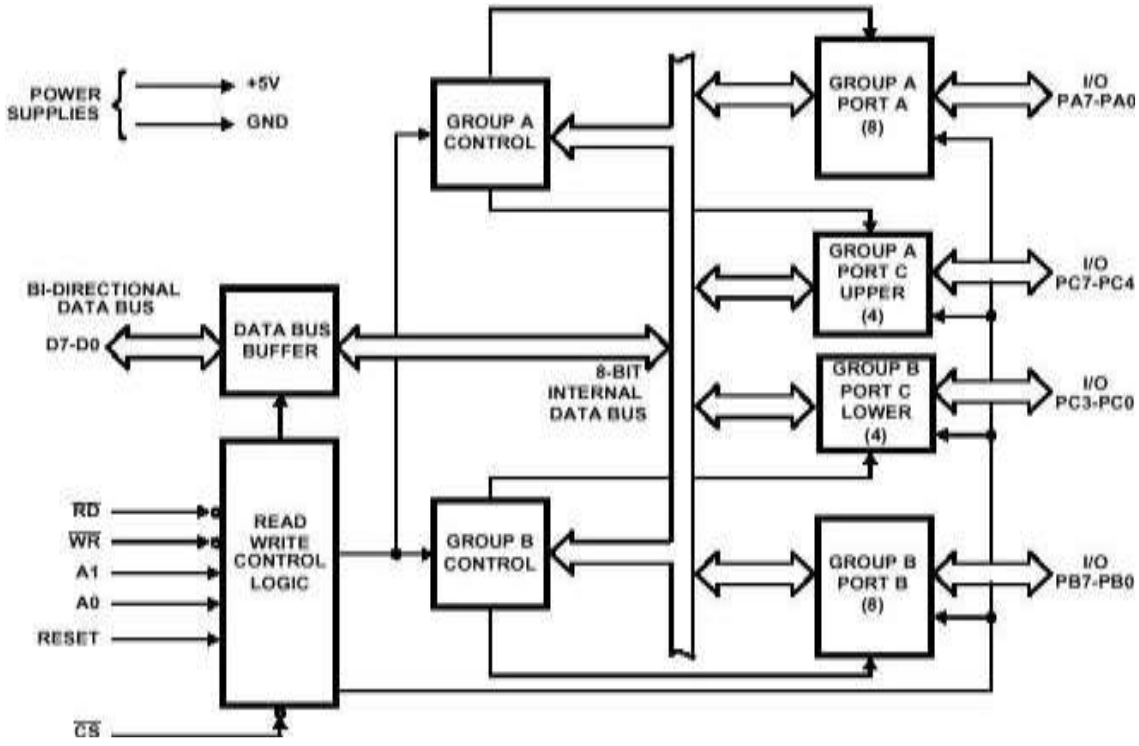


Figure 2.1 Block Diagram of 8255 PPI

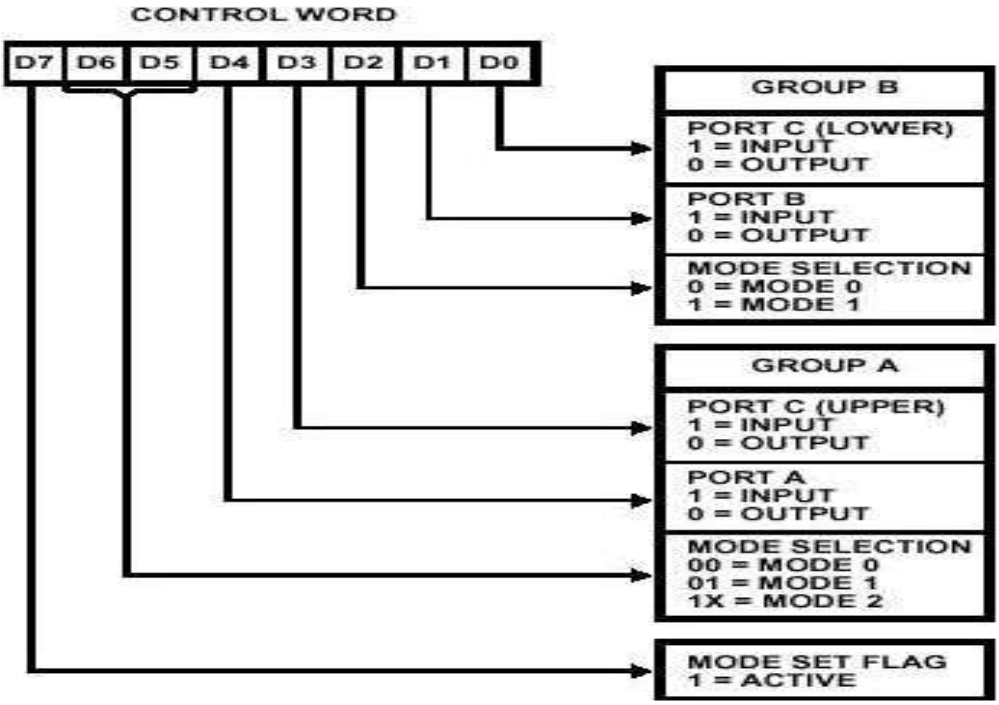


Figure 2.2 Control word Register format

Data Bus Buffer

This three-state bi-directional 8-bit buffer is used to interface the 8255 to the system data bus. Data is transmitted or received by the buffer upon execution of input or output instructions by the CPU. Control words and status information are also transferred through the data bus buffer.

Read/Write and Control Logic

The function of this block is to manage all of the internal and external transfers of both Data and Control or Status words. It accepts inputs from the CPU Address and Control busses and in turn, issues commands to both of the Control Groups.

(CS) Chip Select. A "low" on this input pin enables the communication between the 8255 and the CPU.

(RD) Read. A "low" on this input pin enables 8255 to send the data or status information to the CPU on the data bus. In essence, it allows the CPU to "read from" the 8255.

(**WR) Write.** A "low" on this input pin enables the CPU to write data or control words into the 8255.

(**A0 and A1) Port Select 0 and Port Select 1.** These input signals, in conjunction with the RD and WR inputs, control the selection of one of the three ports or the control word register.

They are normally connected to the least significant bits of the address bus (A0 and A1).

A1	A0	SELECTION
0	0	PORT A
0	1	PORT B
1	0	PORT C
1	1	CONTROL

Figure 2.3 Selection of Ports and Control reg

RESET) Reset. A "high" on this input initializes the control register to 9Bh and all ports (A, B, C) are set to the input mode.

Group A and Group B Controls

The functional configuration of each port is programmed by the systems software. In essence, the CPU "outputs" a control word to the 8255. The control word contains information such as "mode", "bit set", "bit reset", etc., that initializes the functional configuration of the 8255. Each of the Control blocks (Group A and Group B) accepts "commands" from the Read/Write Control logic, receives "control words" from the internal data bus and issues the proper commands to its associated ports.

Ports A, B, and C

The 8255 contains three 8-bit ports (A, B, and C). All can be configured to a wide variety of functional characteristics by the system software but each has its own special features or "personality" to further enhance the power and flexibility of the 8255. Port A One 8-bit data output latch/buffer and one 8-bit data input latch. Both "pull-up" and "pull- down"

bus-hold devices are present on Port A. Port B One 8-bit data input/output latch/buffer and one 8-bit data input buffer.

Port C One 8-bit data output latch/buffer and one 8-bit data input buffer (no latch for input). This port can be divided into two 4-bit ports under the mode control. Each 4-bit port contains a 4-bit latch and it can be used for the control signal output and status signal inputs in conjunction with ports A and B.

I. Operational modes of 8255

There are two basic operational modes of 8255:

1. Bit set/reset Mode (BSR Mode).
2. Input/Output Mode (I/O Mode).

The two modes are selected on the basis of the value present at the D7 bit of the Control Word Register. When D7 = 1, 8255 operates in I/O mode and when D7 = 0, it operates in the BSR mode.

1. Bit set/reset (BSR) mode

The Bit Set/Reset (BSR) mode is applicable to port C only. Each line of port C (PC0 - PC7) can be set/reset by suitably loading the control word register as shown in Figure 4. BSR mode and I/O mode are independent and selection of BSR mode does not affect the operation of other ports in I/O mode. D7 bit is always 0 for BSR mode. Bits D6, D5 and D4 are don't care bits. Bits D3, D2 and D1 are used to select the pin of Port C. Bit D0 is used to set/reset the selected pin of Port C.

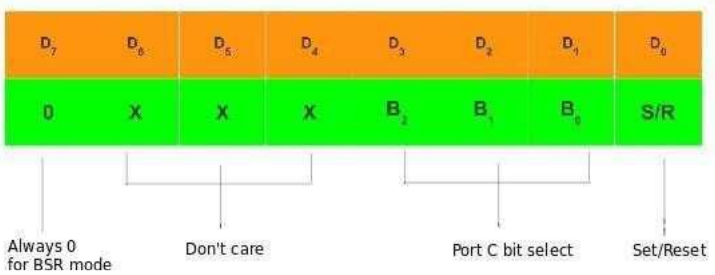


Figure 2.4 8255 Control register format for BSR mode

2. Input / Output mode

This mode is selected when D7 bit of the Control Word Register is 1. There are three I/O modes:

1. Mode 0 - Simple I/O
2. Mode 1 - Strobed I/O
3. Mode 2 - Strobed Bi-directional I/O

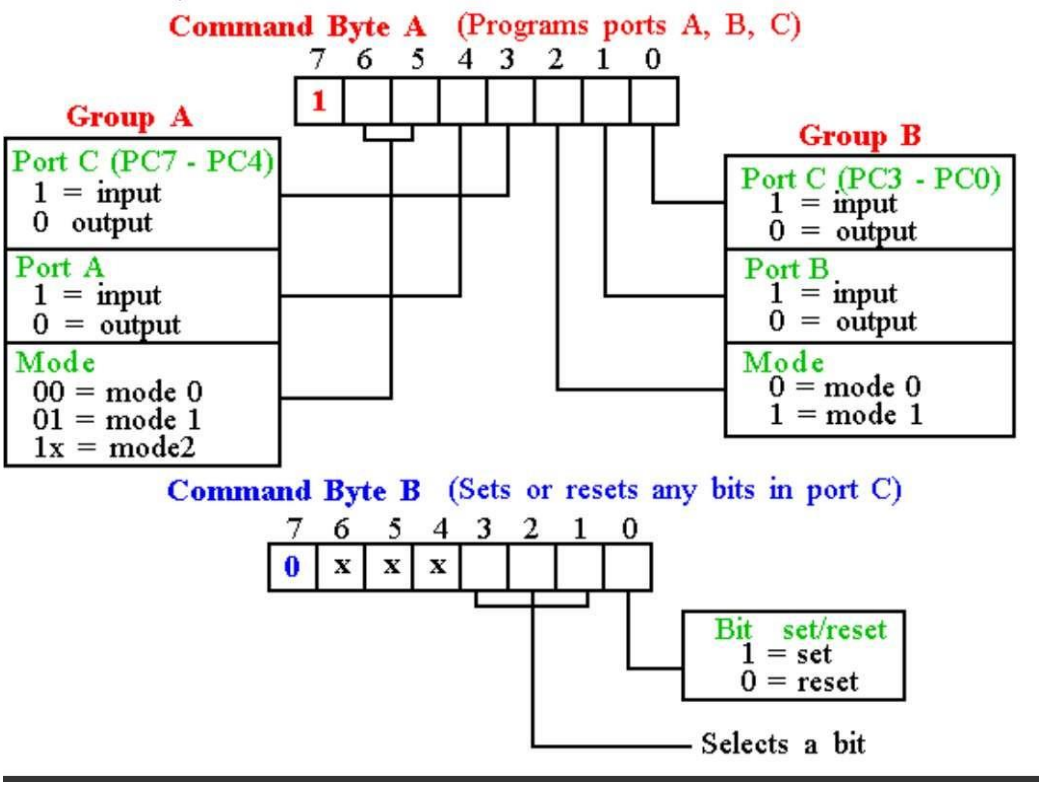


Figure 2.5 8255 Control word for I/O mode

D0, D1, D3, D4 are assigned for lower port C, port B, upper port C and port A respectively. When these bits are 1, the corresponding port acts as an input port. For e.g., if D0 = D4 = 1, then lower port C and port A act as input ports. If these bits are 0, then the corresponding port acts as an output port. For e.g., if D1 = D3 = 0, then port B and upper port C act as output ports .D2 is used for mode selection of Group B (port B and lower port C). When D2 = 0, mode 0 is selected and when D2 = 1, mode 1 is selected.

D5& D6 are used for mode selection of Group A (port A and upper port C). The selection is done as follows:

For example, if port B and upper port C have to be initialized as input ports and lower port C and port A as output ports (all in mode 0):

1. Since it is an I/O mode, D7 = 1.
2. Mode selection bits, D2, D5, D6 are all 0 for mode 0 operation.
3. Port B and upper port C should operate as Input ports, hence, D1 = D3 = 1.
4. Port A and lower port C should operate as Output ports, hence, D4 = D0 = 0.

Hence, for the desired operation, the control word register will have to be loaded with "10001010" = 8A (hex).

Mode 0 - simple I/O

In this mode, the ports can be used for simple input/output operations without handshaking.

- If both port A and B are initialized in mode 0, the two halves of port C can be either used together as an additional 8-bit port, or they can be used as individual 4-bit ports.
- Since the two halves of port C are independent, they may be used such that one-half is initialized as an input port while the other half is initialized as an output port.
- The mode 0 has following features:
- O/p are latched (A latch is a digital IC which holds the data put into it, 1 or 0, until cleared)
- I/p are buffered(A digital buffer (or a voltage buffer) is an electronic circuit element that is used to isolate the input from the output) not latched.
- Port do not have handshake or interrupt capability.

MODE 1- Handshake I/O mode

- To use port A or port B in handshake (strobed) input or output mode, we initialize that port in mode 1.
- For port B in this mode (irrespective of whether is acting as an input port or output port), PC0, PC1 and PC2 pins function as handshake lines.

The mode 1 has following features:

- Two ports i.e. port A and B can be used as 8-bit i/o port.
- Each port uses three lines of port c as handshake signal and remaining two signals can be function as i/o port.
- Interrupt logic is supported.
- Input and Output data are latched.

Mode 2 (Strobed Bidirectional Bus I/O)

In this mode, Port A is used as 8 bits bidirectional bus for data transfer with a peripheral device. This mode is applicable only to Group A, which consists of an 8-bit bidirectional bus (Port A 8-bit) and 5-bit control signals (high order 5 bits of Port C). The bidirectional bus (Port A) has both the internal input and output registers. When group A is set in Mode 2, Group B can be set independently. These are 5 control signals as follows when Group A is used in Mode 2.

- Only group A can be initialized in this mode.
- Port A can be used for *bidirectional handshake* data transfer. This means that data can be input or output on the same eight lines (PA0 - PA7).
- Pins PC3 - PC7 are used as handshake lines for port A.
- The remaining pins of port C (PC0 - PC2) can be used as input/output lines if group B is initialized in mode 0.
- In this mode, the 8255 may be used to extend the system bus to a slave microprocessor.

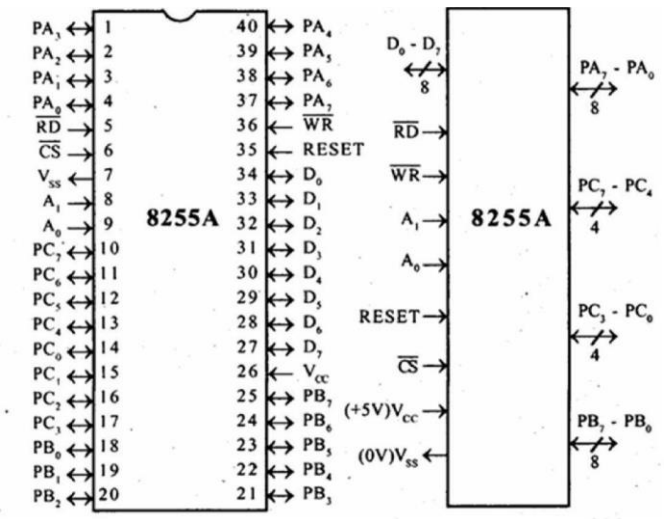


Figure 2.6 Pin Diagram of 8255PPI

USART 8251 (Universal Synchronous/ Asynchronous Receiver Transmitter)

The 8251 is a USART (Universal Synchronous Asynchronous Receiver Transmitter) for serial data communication. As a peripheral device of a microcomputer system, the 8251receives parallel data from the CPU and transmits serial data after conversion. This device also receives serial data from the outside and transmits parallel data to the CPU after conversion.

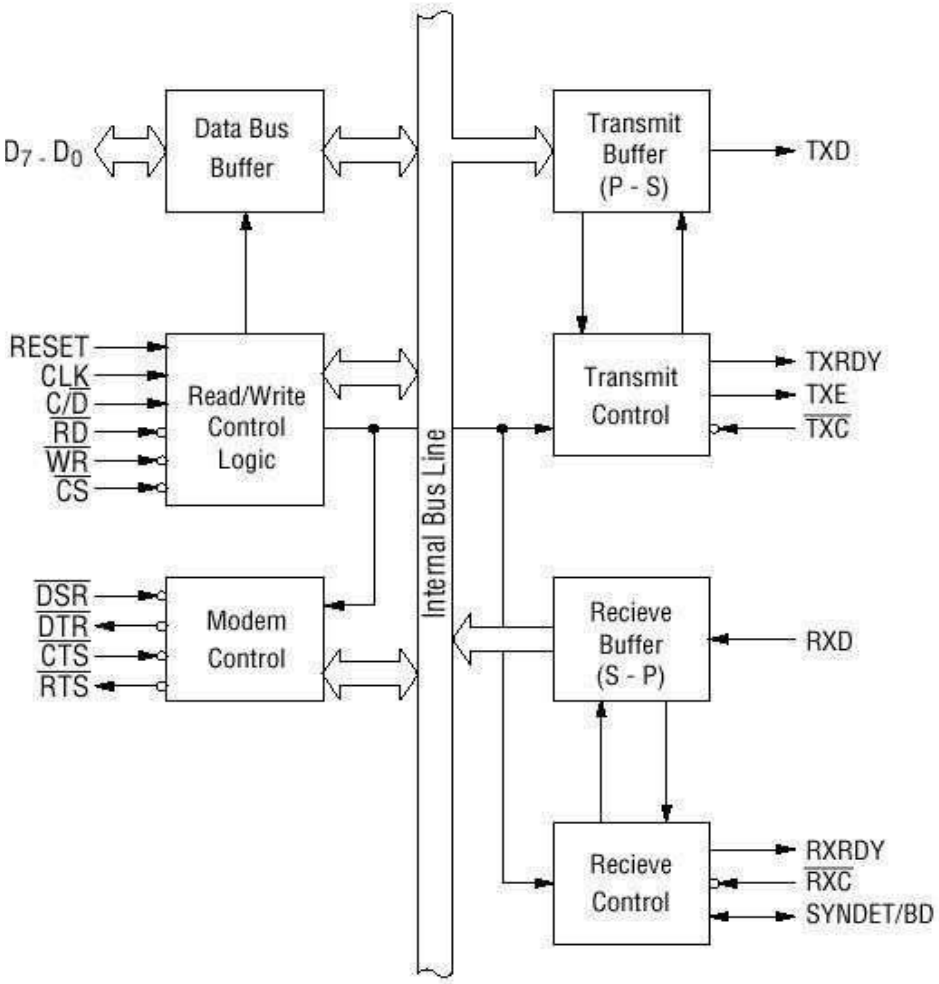


Figure 2.7 Block Diagram of the 8251 USART

Transmitter Section

The transmitter section consists of three blocks—transmitter buffer register, output register and the transmitter control logic block. The CPU deposits (when TXRDY = 1, meaning that the transmitter buffer register is empty) data into the transmitter buffer register, which is subsequently put into the output register (when TXE = 1, meaning that the output buffer is empty). In the output register, the eight bit data is converted into serial form and

comes out via TXD pin. The serial data bits are preceded by START bit and succeeded by STOP bit, which are known as framing bits. But this happens only if transmitter is enabled and the CTS is low. TXC signal is the transmitter clock signal which controls the bit rate on the TXD line (output line). This clock frequency can be 1, 16 or 64 times the baud.

Receiver Section

The receiver section consists of three blocks — receiver buffer register, input register and the receiver control logic block. Serial data from outside world is delivered to the input register via RXD line, which is subsequently put into parallel form and placed in the receiver buffer register. When this register is full, the RXRDY (receiver ready) line becomes high. This line is then used either to interrupt the MPU or to indicate its own status. MPU then accepts the data from the register. RXC line stands for receiver clock. This clock signal controls the rate at which bits are received by the input register. The clock can be set to 1, 16 or 64 times the baud in the asynchronous mode.

RESET (Input terminal)

A "High" on this input forces the 8251 into "reset status." The device waits for the writing of "mode instruction." The min. reset width is six clock inputs during the operating status of CLK.

CLK (Input terminal)

CLK signal is used to generate internal device timing. CLK signal is independent of RXC or TXC. However, the frequency of CLK must be greater than 30 times the RXC and TXC at Synchronous mode and Asynchronous "x1" mode, and must be greater than 5 times at Asynchronous "x16" and "x64" mode.

WR (Input terminal)

This is the "active low" input terminal which receives a signal for writing transmit data and control words from the CPU into the 8251.

RD (Input terminal)

This is the "active low" input terminal which receives a signal for reading receive data and status words from the 8251.

C/D (Input terminal)

This is an input terminal which receives a signal for selecting data or command words and status words when the 8251 is accessed by the CPU. If C/D = low, data will be accessed. If C/D = high, command word or status word will be accessed. CS (Input terminal)

This is the "active low" input terminal which selects the 8251 at low level when the CPU **accesses. Note:** The device won't be in "standby status"; only setting CS = High.

TXD (output terminal)

This is an output terminal for transmitting data from which serial-converted data is sent out. The device is in "mark status" (high level) after resetting or during a status when transmit is disabled. It is also possible to set the device in "break status" (low level) by a command.

TXRDY (output terminal)

This is an output terminal which indicates that the 8251is ready to accept a transmitted data character. But the terminal is always at low level if CTS = high or the device was set in "TX disable status" by a command. Note: TXRDY status word indicates that transmit data character is receivable, regardless of CTS or command. If the CPU writes a data character, TXRDY will be reset by the leading edge or WR signal.

TXEMPTY (Output terminal)

This is an output terminal which indicates that the 8251 has transmitted all the characters and had no data character. In "synchronous mode," the terminal is at high level, if transmit data characters are no longer remaining and sync characters are automatically transmitted. If the CPU writes a data character, TXEMPTY will be reset by the leading edge of WR signal. Note : As the transmitter is disabled by setting CTS "High" or command, data written before disable will be sent out. Then TXD and TXEMPTY will be "High". Even if a data is written after disable, that data is not sent out and TXE will be "High". After the transmitter is enabled, it sent out. (Refer to Timing Chart of Transmitter Control and Flag Timing).

TXC (Input terminal)

This is a clock input signal which determines the transfer speed of transmitted data. In "synchronous mode," the baud rate will be the same as the frequency of TXC. In

"asynchronous mode", it is possible to select the baud rate factor by mode instruction. It can be 1, 1/16 or 1/64 the TXC. The falling edge of TXC sifts the serial data out of the 8251.

RXD (input terminal)

This is a terminal which receives serial data.

RXRDY (Output terminal)

This is a terminal which indicates that the 8251 contains a character that is ready to READ. If the CPU reads a data character, RXRDY will be reset by the leading edge of RD signal. Unless the CPU reads a data character before the next one is received completely, the preceding data will be lost. In such a case, an overrun error flag status word will be set.

RXC (Input terminal)

This is a clock input signal which determines the transfer speed of received data. In "synchronous mode," the baud rate is the same as the frequency of RXC. In "asynchronous mode," it is possible to select the baud rate factor by mode instruction. It can be 1, 1/16, 1/64 the RXC.

SYNDET/BD (Input or output terminal)

This is a terminal whose function changes according to mode. In "internal synchronous mode." this terminal is at high level, if sync characters are received and synchronized. If a status word is read, the terminal will be reset. In "external synchronous mode, "this is an input terminal. A "High" on this input forces the 8251 to start receiving data characters. In "asynchronous mode," this is an output terminal which generates "high level" output upon the detection of a "break" character if receiver data contains a "low-level" space between the stop bits of two continuous characters. The terminal will be reset, if RXD is at high level. After Reset is active, the terminal will be output at low level.

DSR (Input terminal)

This is an input port for MODEM interface. The input status of the terminal can be recognized by the CPU reading status words.

DTR (Output terminal)

This is an output port for MODEM interface. It is possible to set the status of DTR by a command.

CTS (Input terminal)

This is an input terminal for MODEM interface which is used for controlling a transmit circuit. The terminal controls data transmission if the device is set in "TX Enable" status by a command.

Data is transmitable if the terminal is at low level.

RTS (Output terminal)

This is an output port for MODEM interface. It is possible to set the status RTS by a command. The 8251 functional configuration is programmed by software. Operation between the 8251 and a CPU is executed by program control. Table 1 shows the operation between a CPU and the device. Summary of Control Signals for 8251

Summary of Control Signals for 8251

Table 2.1

$\overline{CS}$	$\overline{C/D}$	$\overline{RD}$	$\overline{WR}$	*Function*
0	1	1	0	MPU writes instructions in the control register
0	1	0	1	MPU reads status from the status register
0	0	1	0	MPU outputs data to the Data Buffer
0	0	0	1	MPU accepts data from the Data Buffer
1	X	X	X	USART is not selected

Control Words

There are two types of control word.

1. Mode instruction (setting of function)
2. Command (setting of operation)

1) Mode Instruction

Mode instruction is used for setting the function of the 8251. Mode instruction will be in "wait for write" at either internal reset or external reset. That is, the writing of a control word after resetting will be recognized as a "mode instruction." Items set by mode instruction.

2) Command

Command is used for setting the operation of the 8251. It is possible to write a command whenever necessary after writing a mode instruction and sync characters

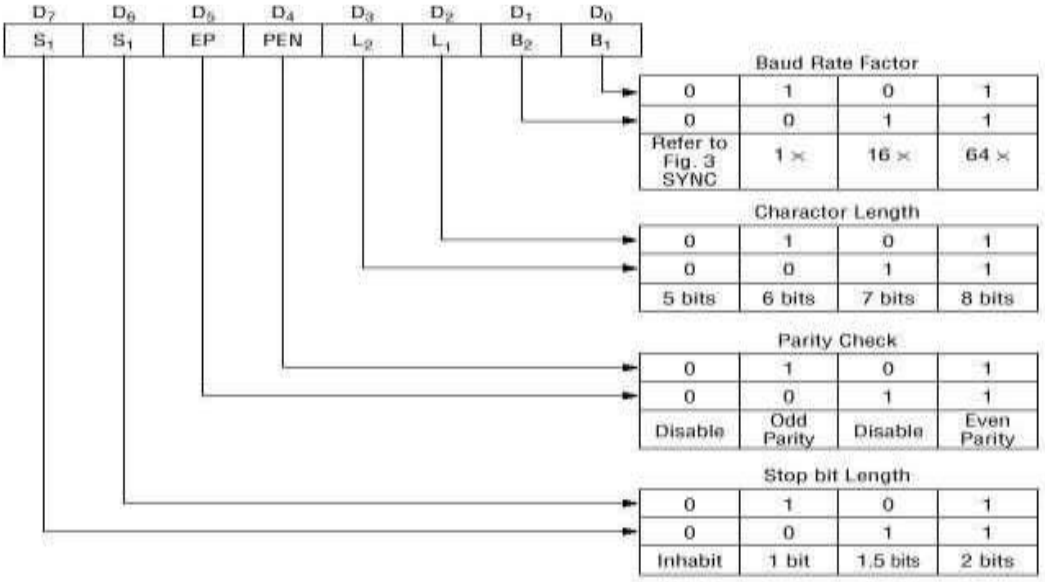


Figure 2.8 Bit Configuration of Mode Instruction (Asynchronous)

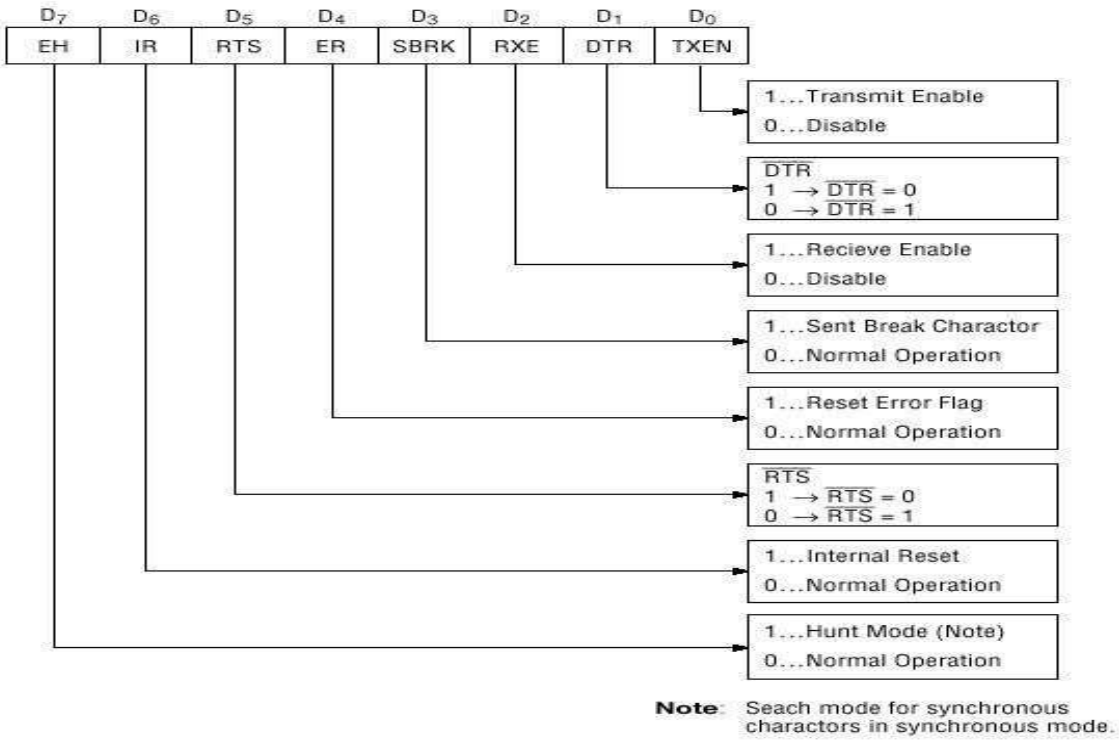


Figure 2.9 Bit Configuration of Command

It is possible to see the internal status of the 8251 by reading a status word. The bit configuration of status word is shown below.

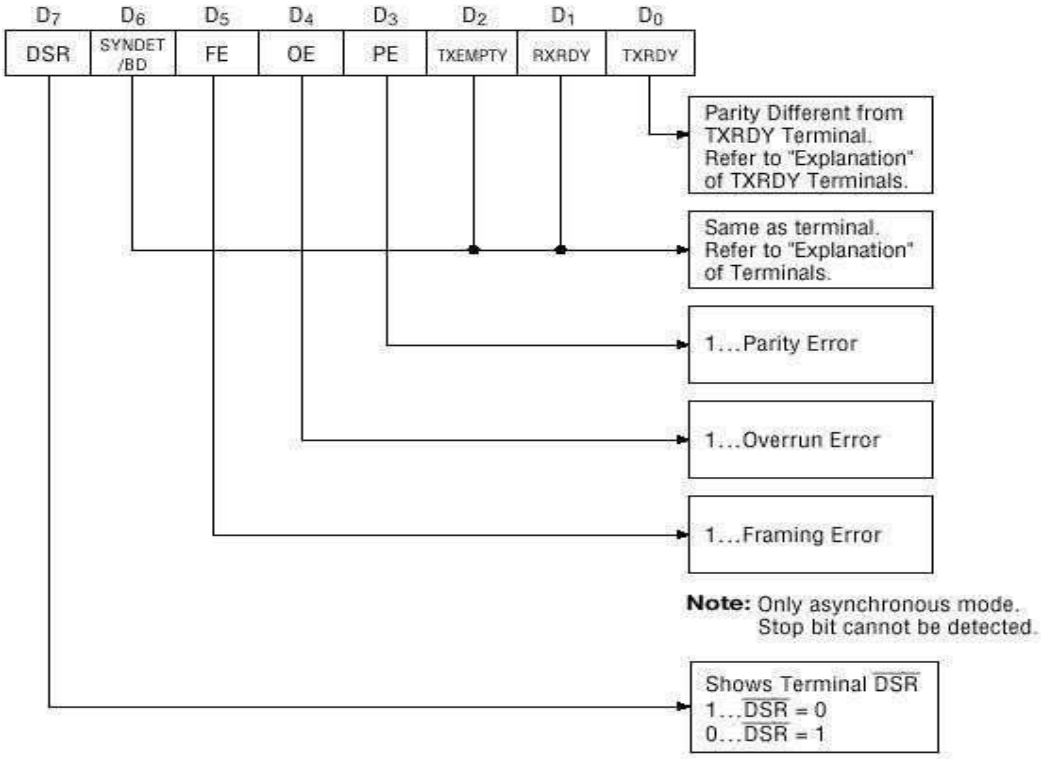


Figure 2.10 Bit Configuration of Status Word

8237 - DMA Controller

- The DMA I/O technique provides direct access to the memory while the microprocessor is temporarily disabled.
- This chapter also explains the operation of disk memory systems and video systems that are often DMA-processed.
- Disk memory includes floppy, fixed, and optical disk storage. Video systems include digital and analog monitors.

Pin Details of DMA (8237)

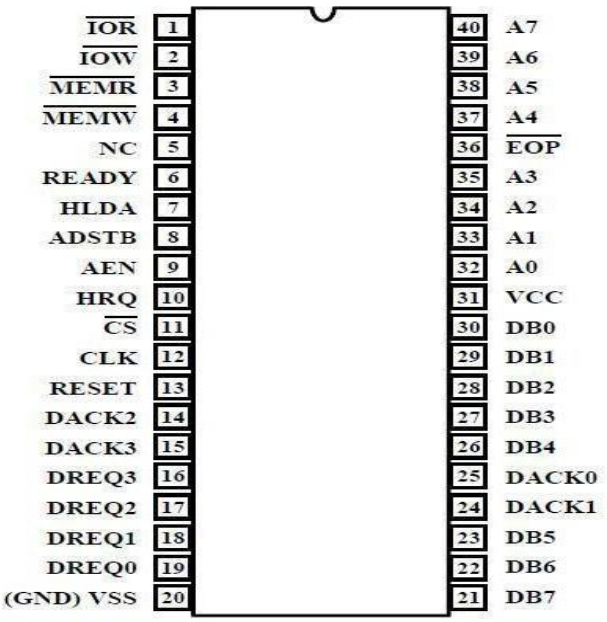


Figure 2.11 Pin Diagram of DMA

A DMA controller is a device, usually peripheral to a CPU that is programmed to perform a sequence of data transfers on behalf of the CPU. A DMA controller can directly access memory and is used to transfer data from one memory location to another, or from an I/O device to memory and vice versa. A DMA controller manages several DMA channels, each of which can be programmed to perform a sequence of these DMA transfers. Devices, usually I/O peripherals, that acquire data that must be read (or devices that must output data and be written to) signal the DMA controller to perform a DMA transfer by asserting a hardware DMA request (DRQ) signal. A DMA request signal for each channel is routed to the DMA controller. This signal is monitored and responded to in much the same way that a processor handles interrupts. When the DMA controller sees a DMA request, it responds by performing one or many data transfers from that I/O device into system memory or vice versa. Channels must be enabled by the processor for the DMA controller to respond to DMA requests. The number of transfers performed, transfer modes used, and memory locations accessed depends on how the DMA channel is programmed. A DMA controller typically shares the system memory and I/O bus with the CPU and has both bus master and slave capability.

In bus master mode, the DMA controller acquires the system bus (address, data, and control lines) from the CPU to perform the DMA transfers. Because the CPU releases the

system bus for the duration of the transfer, the process is sometimes referred to as cycle stealing.

In bus slave mode, the DMA controller is accessed by the CPU, which programs the DMA controller's internal registers to set up DMA transfers. The internal registers consist of source and destination address registers and transfer count registers for each DMA channel, as well as control and status registers for initiating, monitoring, and sustaining the operation of the DMA controller.

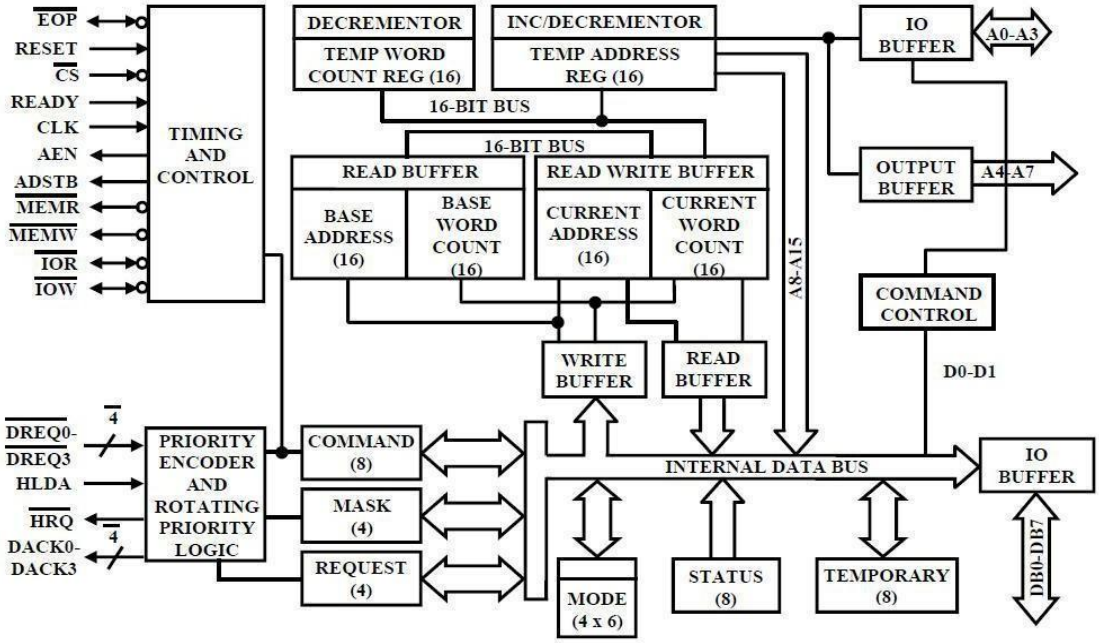


Figure 2.12 Block Diagram

DMA Controller Operation Steps in a Typical DMA cycle Device wishing to perform DMA asserts the processors bus request signal.

Processor completes the current bus cycle and then asserts the bus grant signal to the device.

The device then asserts the bus grant ack signal.

The processor senses in the change in the state of bus grant ack signal and starts listening to the data and address bus for DMA activity.

The DMA device performs the transfer from the source to destination address. During these transfers, the processor monitors the addresses on the bus and checks if any location

modified during DMA operations is cached in the processor. If the processor detects a cached address on the bus, it can take one of the two actions: Processor invalidates the internal cache entry for the address involved in DMA write operation

Processor updates the internal cache when a DMA write is detected Once the DMA operations have been completed, the device releases the bus by asserting the bus release signal.

ADC- Analog to Digital Converter

- An electronic integrated circuit which transforms a signal from analog(continues) to digital(discrete) form
- Analog signals are directly measurable quantities
- Digital signals only have two states for digital computer we refer to binary states, 0 and 1
- The heart of computer-based data acquisition is usually the analog to digital converter
- The Analog to Digital Conversion is a quantizing process. Here the analog signal is represented by equivalent binary states.
- Basically this device is digital volt meter
- Digital Systems require discrète digital data
- Digital computers require signals to be in digital form whereas most instrumentation transducers have an output signal in analogue form.
- ADC conversion is therefore required at the interface between analogue transducers and the digital computer

TYPES OF ADC

The A/D converters can be classified into two groups based on their conversion techniques.

- In the first technique it compares given analog signal with the internally generated equivalent (analog) signal.

- In this technique, it includes successive approximation, counter and flash type converters.
- In another technique it determines the changing of analog signals into time or frequency.
- This process includes integrator-converters and voltage-to-frequency converters. The first process is faster but less accurate, the second one is more accurate. As the first process uses flash type, so it is expensive and difficult to design for high accuracy.

Resolution

Minimum analog value that can be represented in digital. It depends on the number of bits in the ADC The

Conversion Time

The time required to convert the given analog signal in to digital

ADC 0808/0809 Chip

- The ADC 0808/0809 is an 8-bit analog to digital converter. It has 8 channel multiplexer to interface with the microprocessor.
- This device uses successive approximation technique to convert analog signal to digital form.

Features of ADC 0808/0809

- The conversion speed is much higher
- The accuracy is also high
- It has minimal temperature dependence
- Excellent long term accuracy and repeatability
- Less power consumption

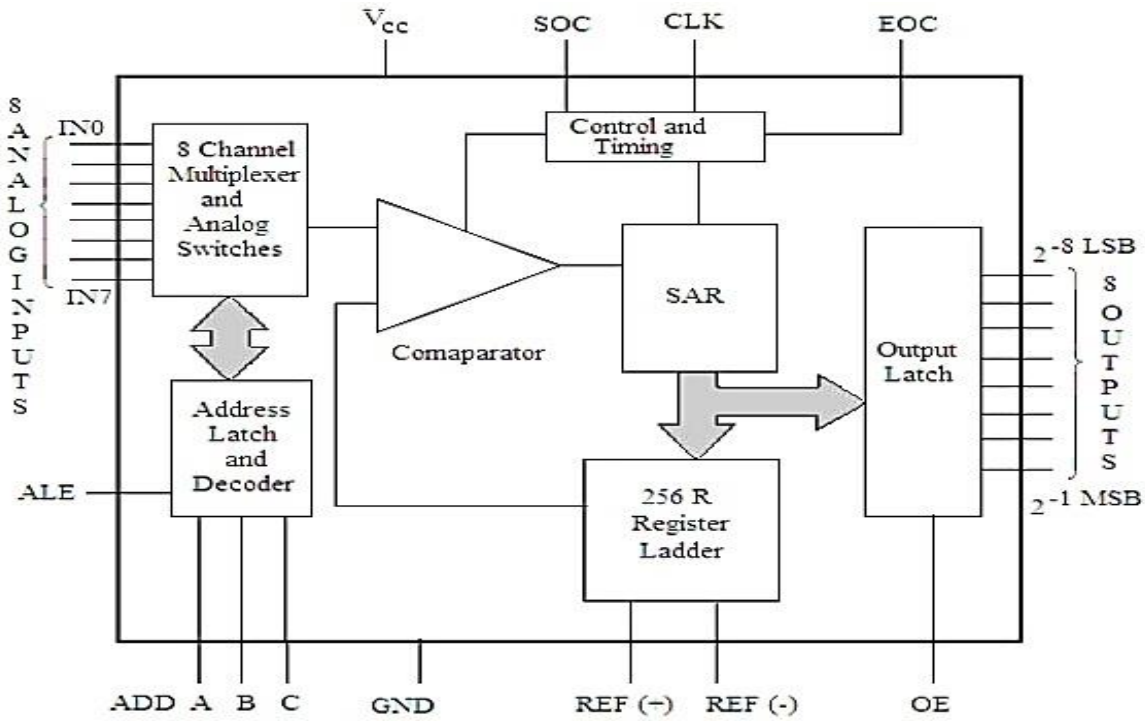


Figure 2.13 Functional Block Diagram of ADC 0808

ADC 0808 data acquisition component is a monolithic CMOS device with 8 channel multiplexer and microprocessor compatible control logic. The 8 channel multiplexer can directly access any of 8 single ended analog signal.

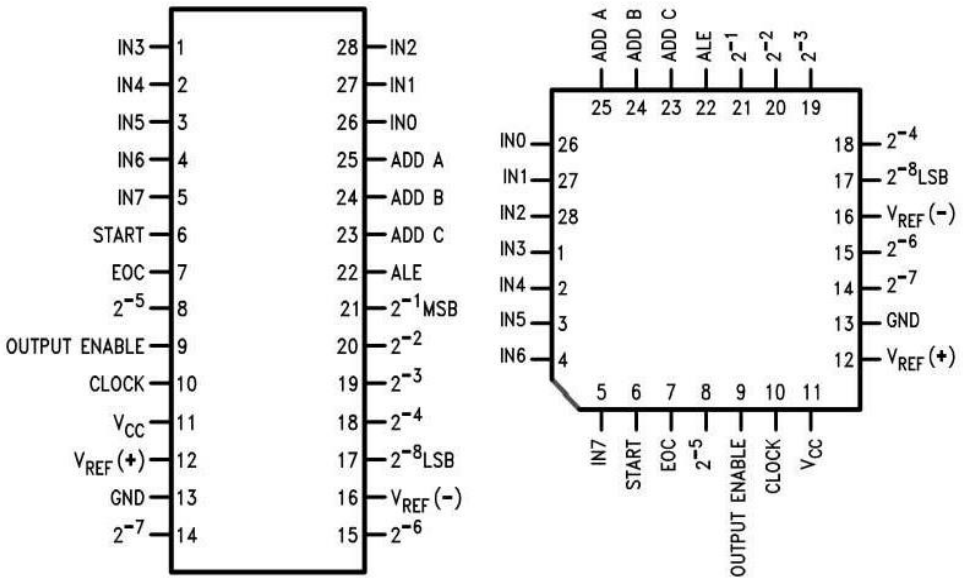


Figure 2.14 Pin Diagram of ADC 0808

ADDRESS SELECT LINE

Table 2.2

Selected Analog Channel	Address Line		
	C	B	A
IN0	L	L	L
IN1	L	L	H
IN2	L	H	L
IN3	L	H	H
IN4	H	L	L
IN5	H	L	H
IN6	H	H	L
IN7	H	H	H

ALE [ADDRESS LATCH ENABLE]

ALE is required to load the selected address lines into the ADC. Once loaded the multiplexer sends the appropriate channel to the converter on the chip. The ALE should be pulsed for at least 100ns in order for the addresses to get loaded properly. As with all control signals it is required to have an input value of Vcc - 1.5 up to 15V for a high and 1.5V down to -0.3V for a low.

CLOCK

The clock signal is required to cycle through the comparator stages to do the conversion. There are 8 clock cycle periods required in order to complete an entire conversion. This means that an entire conversion takes at least 64 clock cycles. The clock should conform to the same range as all other control signals. The maximum frequency of the clock is 1.2MHz

OE [OUTPUT ENABLE] START

ADC stores the data in a tri-state output latch until the next conversion is started The Output Enable signal causes the ADC to actually output the digital values on the output lines.

The, but the data is only output when enabled. In this implementation the OE signal is pulsed high one clock cycle after the EOC signal goes high and remains high until the data is safely stored into the desired register.

EOC [END OF CONVERSION]

The signal goes low once a conversion is initiated by the start signal and remains low until a conversion is complete.

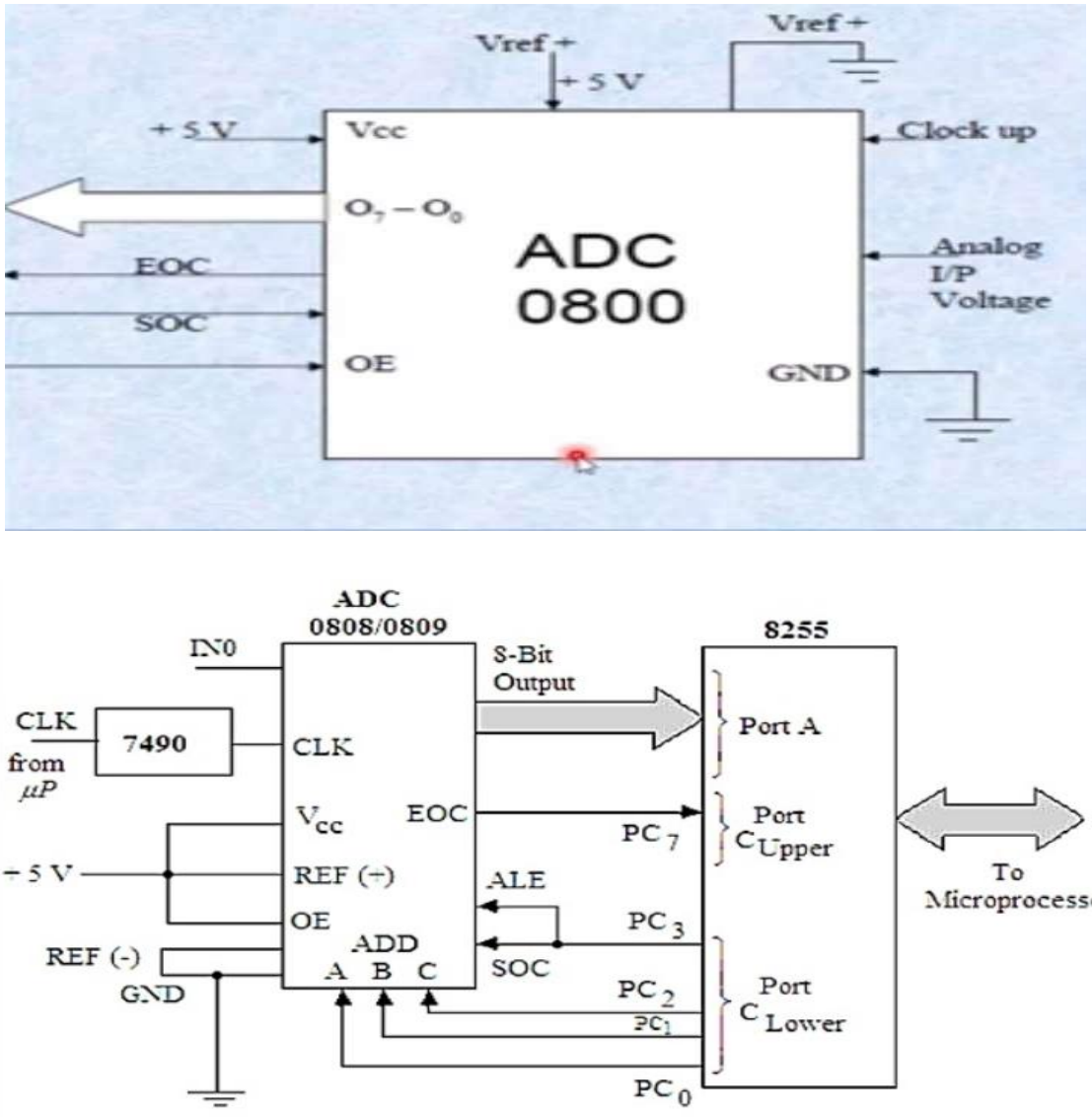


Figure 2.15 Interfacing ADC 0808 with 8085

Control Word Format in I/O Mode

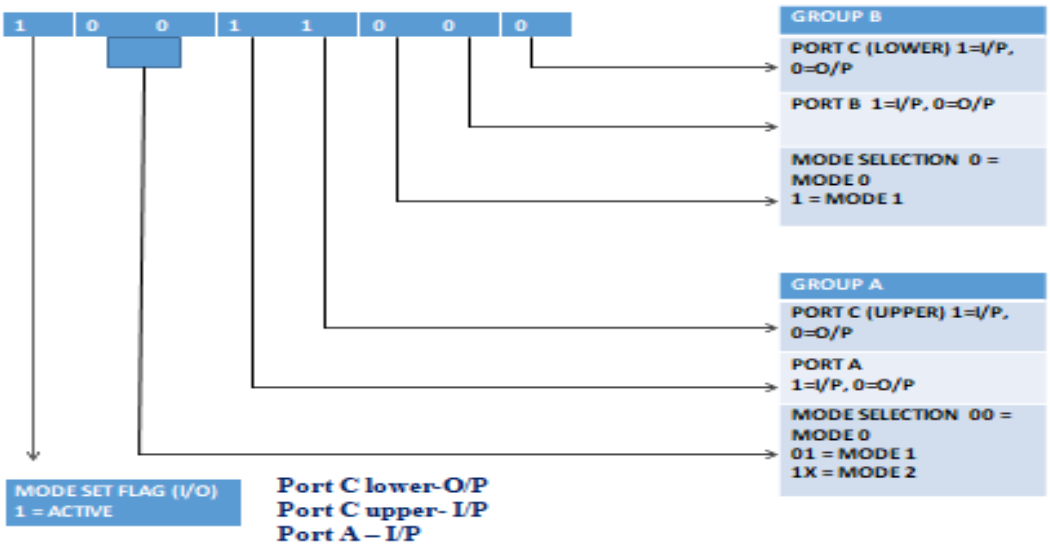


Figure 2.16 Control Word Format in I/O Mode

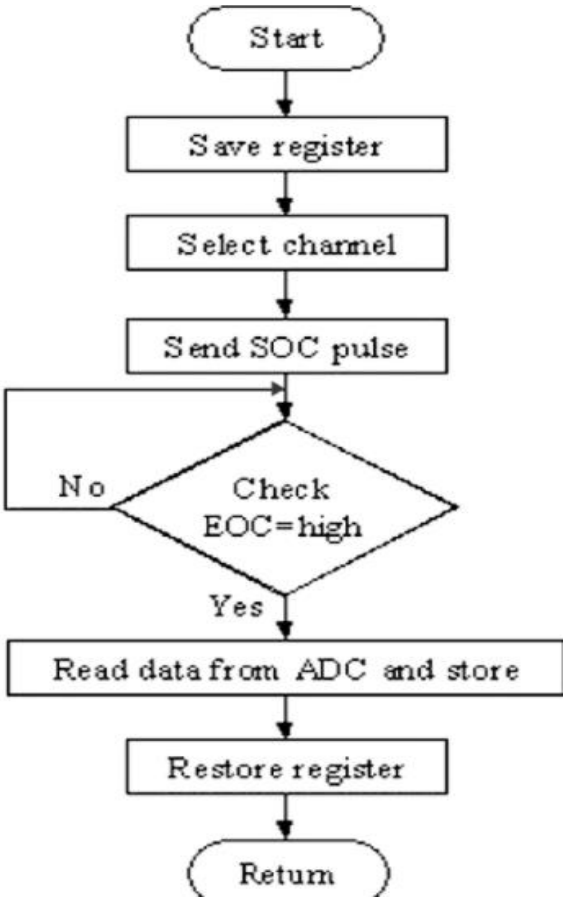


Figure 2.17 Flow Chart

```
PC7-EOC-I/P
PC3-SOC-0/P
PORT A-I/P
MVI A,98  /// 10011000
OUT Control Reg
L2: MVI A, 08(send SOC)
OUT PORT C
L1:IN PORT C(wait for EOC)  A=80   pc7 pc6 pc5 pc4 pc3 pc2 pc1 pc0
CPI 80   1000 0000
JNZ L1:
IN PORT A
STA 9100
JMP L2
```

Program

DAC - Digital to Analog Converter

- In many applications, the microprocessor has to produce analog signals for controlling certain analog devices. Basically the microprocessor system can produce only digital signals.
- In order to convert the digital signal to analog signal a Digital-to-Analog Converter. (DAC) has to be employed.
- The DAC will accept a digital (binary) input and convert to analog voltage or current. Every DAC will have "n" input lines and an analog output.
- The DAC require a reference analog voltage (Vref) or current (Iref) source.
- The smallest possible analog value that can be represented by the n-bit binary code is called resolution.
- The resolution of DAC with n-bit binary input is $1/2^n$ of reference analog value. Every analog output will be a multiple of the resolution
- For example, Consider an 8-bit DAC with reference analog voltage of 5 volts.
- Now the resolution of the DAC is $(1/2^8)$ x 5 volts. The 8-bit digital input can take, 2^8 = 256 different values. The analog values for all possible digital input are as shown in table below.

Table 2.3 Digital to Analog Converter

Digital Input	Analog Output
0000 0000	$\frac{0}{2^8} \times 5$ Volts
0000 0001	$\frac{1}{2^8} \times 5$ Volts
0000 0010	$\frac{2}{2^8} \times 5$ Volts
0000 0011	$\frac{3}{2^8} \times 5$ Volts
⋮	⋮
1111 1111	$\frac{255}{2^8} \times 5$ Volts

DAC0808

The DAC0808 of National Semiconductor Corporation is an example of 8-bit DAC without internal latch and I to V converting amplifier.

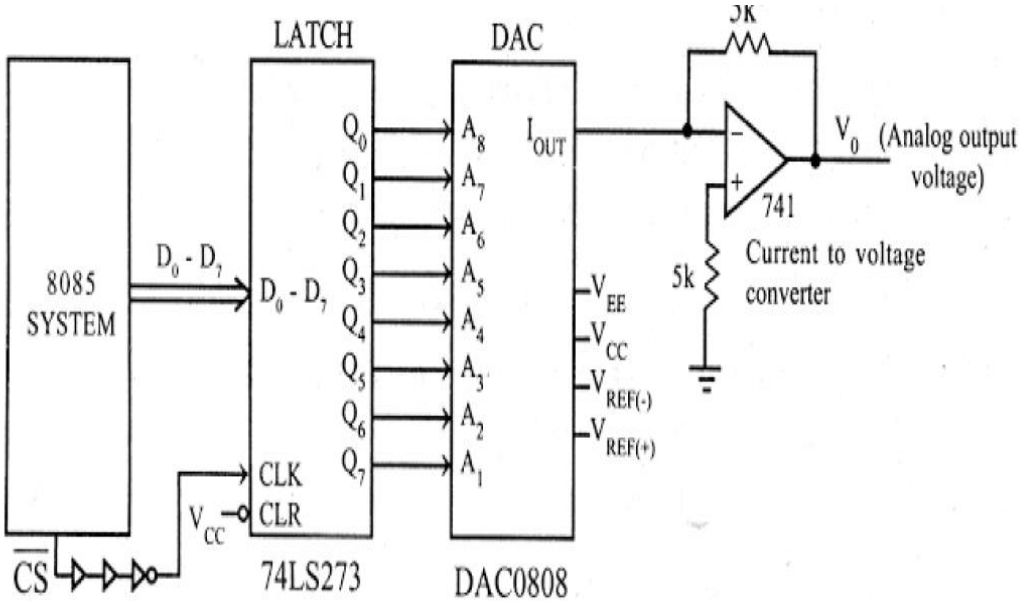


Figure 2.18

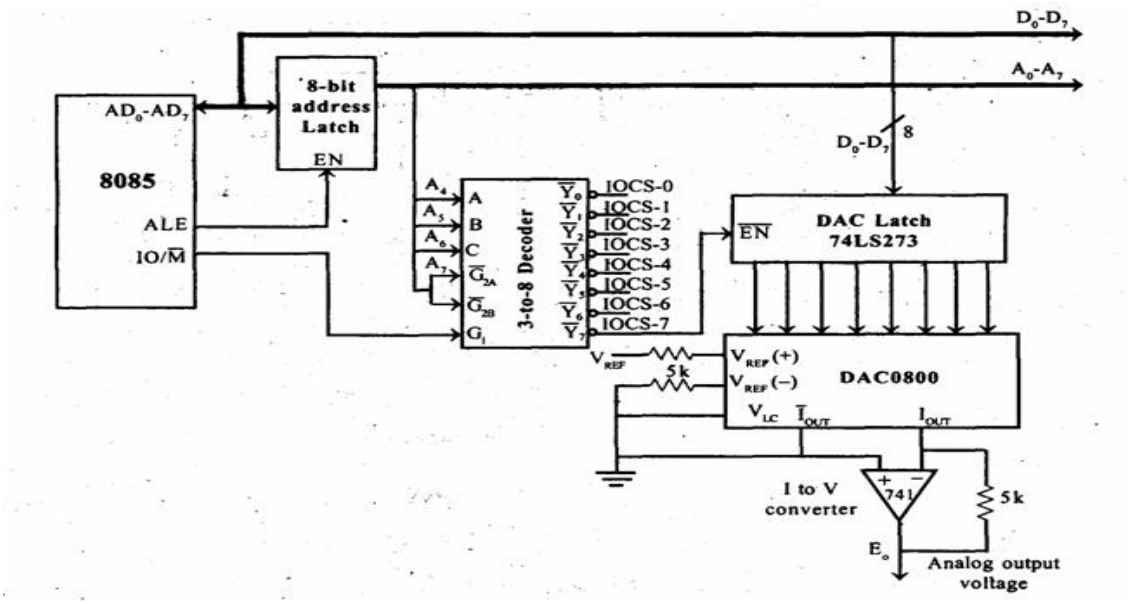


Figure 2.19 Interfacing DAC 0808 with 8085

- The processor sends an address, which is decoded by decoder in the microprocessor system to produce chip select signal.
- Then the processor sends a digital data to latch. The buffer and inverter will produce sufficient delay for CS signal so that, the latch is clocked only after the data is arrived at the input lines of the latch.
- When the latch is clocked the digital data is send to DAC.
- The DAC will produce a corresponding current signal, which is converted to voltage signal by the op-amp 741.
- The typical settling time of DAC0800 is 150nsec. Therefore the processor need not wait for loading next data.
- In this schematic the DAC0800 is interfaced using an 8-bit latch 74LS273 to the system bus.
- The 3-to-8 decoder 74LS 138 is used to generate chip select signals for I/O devices.
- The address lines A4, A5 and A6 are used as input to decoder.
- The address line A7 and the control signal IO/M (low) are used as enable for decoder.
- The decoder will generate eight chip select signals and in this the signal IOCS-7 is used as enable for latch of DAC.
- The I/O address of the DAC is shown in table.

Table 2.4

Device	Binary Address								Hexa Address
	Decoder Input and enable				Unused address lines				
	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	
DAC Latch 74LS273	0	1	1	1	x	x	x	x	70

- In order to convert a digital data to analog value, the processor has to load the data to latch.
- The latch will hold the previous data until next data is loaded.
- The DAC will take definite time to convert the data. The software should take care of loading successive data only after the conversion time.
- The DAC 0800 produces a current output, which is converted to voltage output using Ito V converter

Table 2.5

Device	Binary Address					Hexa Address	Comment
	Decoder input	Unused Address lines					
	$A_{15}A_{14}A_{13}$	A_{12}	$A_{11}A_{10}A_9A_8$	$A_7A_6A_5A_4$	$A_3A_2A_1A_0$		
DAC Latch 74LS273	0 1 1	x	x x x x	x x x x	x x x x	6000	External data memory address space

DAC- WAVEFORM GENERATION

SAW TOOTH

```
L1:MVI A,00
OUT DAC
INR A
JMP L1:
```

TRIANGULAR

```
MVI A,00
L1:OUT DAC
INR A
CPI FF
JNZ L1:
L2:OUT DAC
DCR A
JNZ L2
JMP L1:
```

SQUARE WAVE

```
L1:MVI A,00
OUT DAC
CALL DELAY
MVI A, FF
OUT DAC
CALL DELAY
JMP L1:

DELAY
MVI B,55
L2:DCR B
JNZ L2:
RET
```

Temperature controller using 8085

The microprocessor based temperature control system can be used for automatic control of the temperature of a body. A simplified block diagram of 8085 microprocessor based temperature control system is shown in figure. The system consist of 8085 microprocessor as CPU, ERROM&RAM memory for program & data storage, INTEL 8279 for keyboard and display Interface, ADC, DAC, INTEL 8255 for I/O' ports, Amplifiers, Signal conditioning circuit, Temperature sensor and Supply control circuit. In this system the temperature is controlled by controlling the power input to the heating element.

The temperature of the body is measured using a temperature sensor. The different types of temperature sensor that can be used for temperature measurement are Thermocouple, Thermistors, PN-junctions, IC sensors. This sensor will convert the input temperature to Proportional analog voltage or current. The output signal of the sensor will be a weak signal and so has to be amplified by using high input impedance op-amp. Then the analog signal is scaled to suitable level by the signal conditioning circuit.

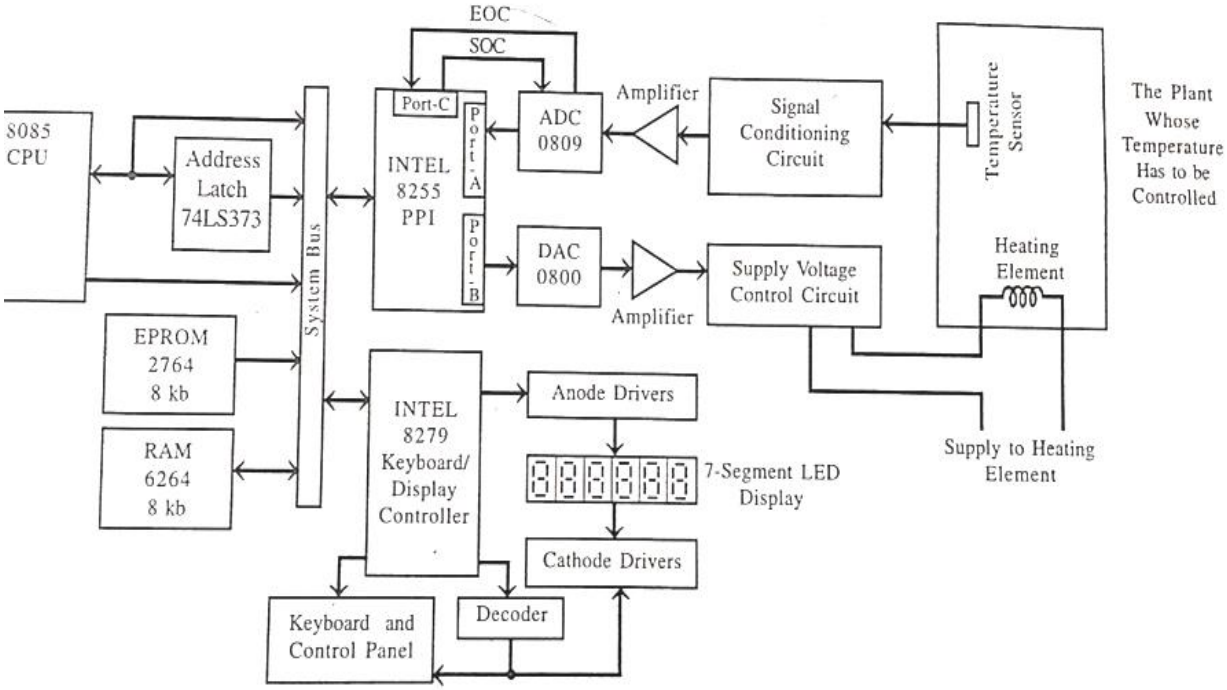


Figure 2.20 8085 Microprocessor based Temperature Control System

The microprocessor can process only digital signals and so the analog signal from signal conditioning circuit cannot be read by the processor directly. The system has an

analog-to digital converter (ADC) to convert the analog signal to proportional digital data. In this system the ADC is interfaced to 8085 processor through port-A of 8255. The 8085 processor send signal to ADC to start conversion and at the end of conversion it read the digital data from the port-A of 8255.

The 8085 processor calculate the actual temperature using the input data and display it on the 7- segment LED. Also, the processor compare the desired temperature with actual temperature (The operator can enter the desired temperature through keyboard) and calculate the error (the difference between actual temperature and desired temperature).

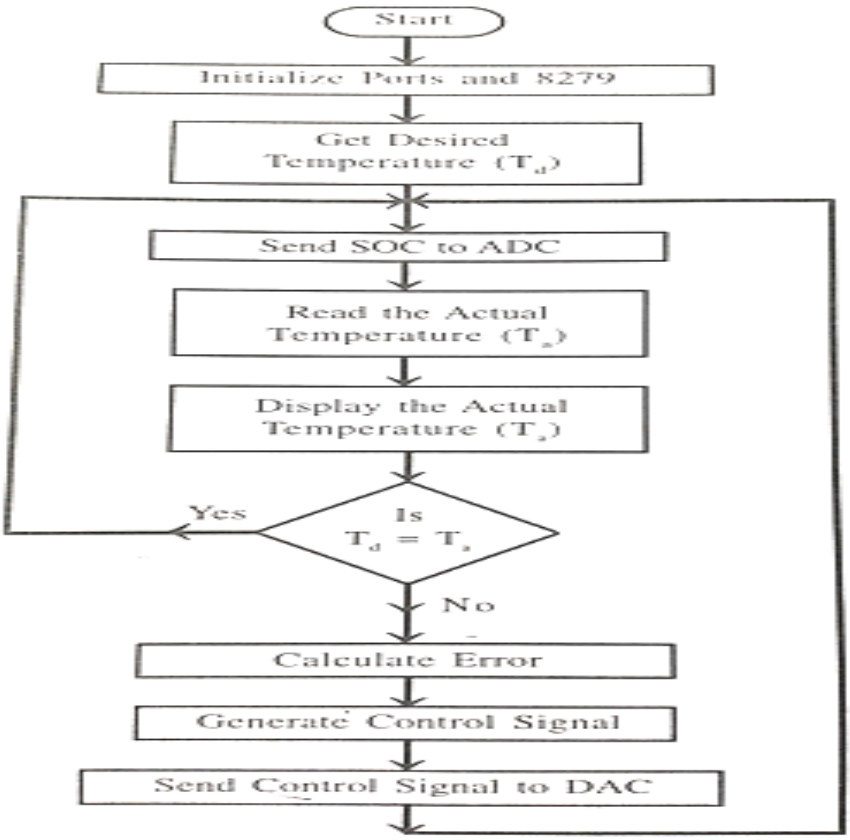


Figure 2.21 Flow Chart for Temperature Control System

The error is used to compute a digital control signal, which is converted to analog control signal by DAC. The DAC is interfaced to the system through port-B of 8255. The analog control signal produced by DAC is used to control the power supply of the heating element of the body.

The digital control signal can be computed by the 8085 processor using different digital control algorithms (PIPI/PID/FUZZY logic control algorithms).

The control circuit for power supply can be either thyristor based circuit or relay. In case of thyristor control circuits the firing angle can be varied by the control signal to control the power input to the heater.

Stepper motor controller system using 8085

The stepper motors are popularly used in computer peripherals, plotters, robots and machine tools for 'precise incremental rotation. In stepper motor, the stator windings are excited by electrical pulses and for each pulse the motor shaft advances by one angular step. (Single the stepper motor can be driven by digital pulses; it is also called digital motor). The step size in the motor is determined by the number of poles in the rotor and the number 'of pairs of stator windings (one pair of stator winding is called one 'phase). The stator windings are also called control windings.

The motor is controlled by switching ON/OFF the control winding. The popular stepper motor used for demonstration in laboratories has a step size of 1.8° (i.e, 200 steps per revolution). This motor consist of four stator winding and require four switching sequence as shown in table 1. The basic step size of the motor is called full-step. By altering the switching sequence, the motor can be made to run with incremental motion of half the full-step value.

A typical stepper motor control system is shown in fig 6. a two-phase or four winding stepper motor is show in fig 6. The system consists of 8085 microprocessor as CPU, EPROM and RAM memory for program &data storage and for stack. Using INTEL 8279, a keyboard and six number of 7-segment LED display has been interfaced in the system. Through the keyboard the operator can issue commands to control the system. The LED display has been provided to display messages to the operator. The windings of stepper motor connected to the collector of Darlington pair transistors. The transistors are switched ON/OFF by the microprocessor through the ports of 8255 and buffer (74LS245).

A freewheeling diode is connected across each winding for fast switching. The flowchart for the operational flow of the stepper motor control system is shown in figure. The processor has to output a switching sequence and wait for l to 5 msec before Sending next switching sequence. (The delay is necessary to allow be motor transients to die-out).

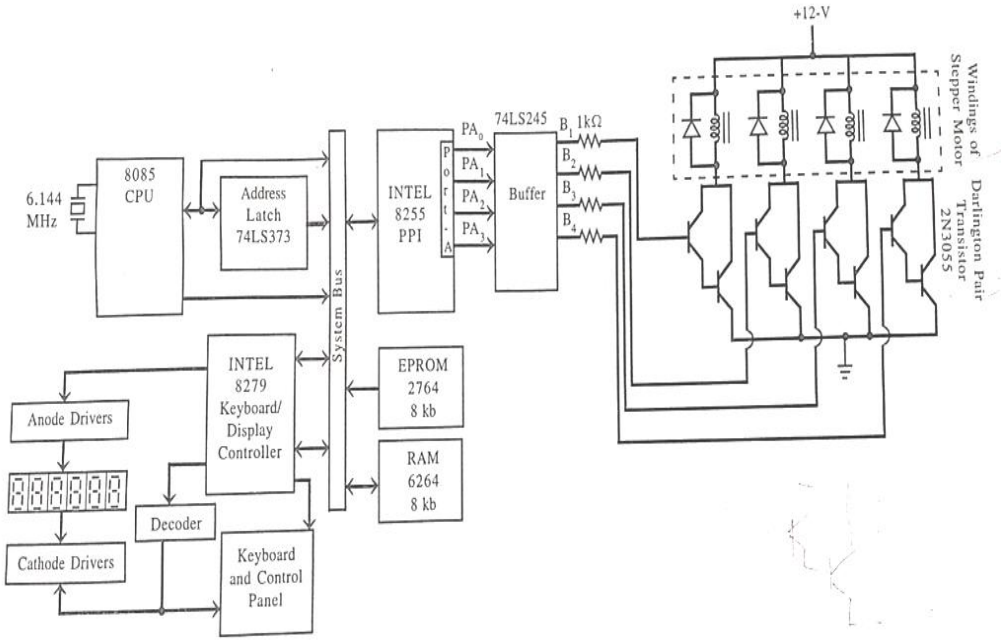


Figure 2.22 8085 Microprocessor based Stepper Motor Control System

Table 2.6 Switching sequence for Full Step Rotation

Switching sequence	Clockwise rotation				Anticlockwise rotation			
	PA_3	PA_2	PA_1	PA_0	PA_3	PA_2	PA_1	PA_0
Sequence-1	1	1	0	0	0	0	1	1
Sequence-2	0	1	1	0	0	1	1	0
Sequence-3	0	0	1	1	1	1	0	0
Sequence-4	1	0	0	1	1	0	0	1

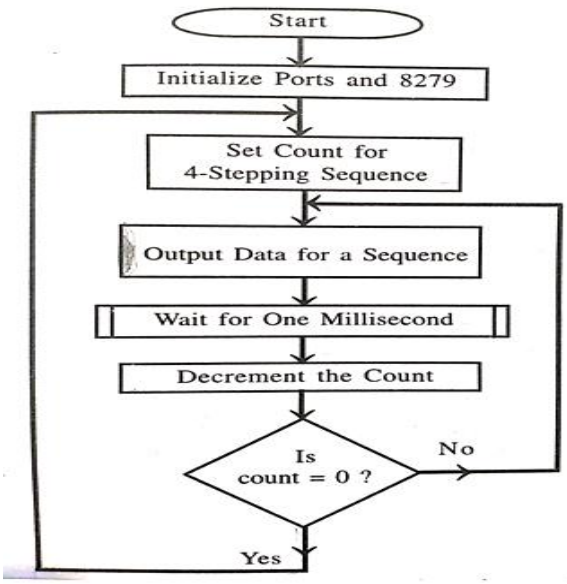


Figure 2.23 Flow Chart for Stepper Motor Control System

ALGORITHM –STEPPER MOTOR

1. Initialize port A of 8255 as output port.
2. Load the step sequence in an array.
3. out the step sequence through port A to the stepper motor
4. Call the delay in between the step sequence.
5. send the 4 step sequence.(reverse the step sequence for anticlock wise)
6. delay decreases speed increases(inversely proportional)
7. repeat unconditionally so that the motor rotate continuously

Full step Mode

In mode1(Full stepping) ,the motor moves by 1.8°, to do this two bits are changed at a time ,the bit pattern is as

X'	Y	X	Y'	HEXA VALUE
0	0	1	1	03
0	1	1	0	06
1	1	0	0	0C
1	0	0	1	09

Program for Half stepping

```
        MVI A,80H
        OUT 83H
BACK:   LXI H,2000H
        MVI C,08H
UP:     MOV A,M
        OUT 82H
        CALL DELAY
        INX H
        DCR C
        JNZ UP
        JMP BACK
```

Address	Data
2000	0A
2001	08
2001	09
2002	01
2003	05
2004	04
2005	06
2006	02

Program for full stepping

```
        MVI A,80H
        OUT 83H
BACK:   LXI H,2000H
        MVI C,04H
UP:     MOV A,M
        OUT 80H
        CALL DELAY
        INX H
        DCR C
        JNZ UP
        JMP BACK
```

Address	Data
2000	03
2001	06
2002	0C
2003	09

STEPPER MOTOR-APPLICATIONS

- The stepper motor is used for precise positioning with a motor like hard disk drives, robotics, telescopes and some toys.
- Industrial Machines-Stepper motors are used in automotive gauge and machine tooling automated production equipments.
- Security-New surveillance products for the security industry
- Medical-Stepper motors are used inside medical scanner , sampler and also found inside digital dental photography.

LCD Interfacing

- Liquid Crystal Displays (LCDs)
- cheap and easy way to display text
- Various configurations (1 line by 20 char upto 8 lines X 80).
- Integrated controller
- The display has two register
  - command register
  - data register
- By RS you can select register
- Data lines (DB7-DB0) used to transfer data and commands

Pinout

- 8 data pins D7:D0
- RS: Data or Command Register Select
- R/W: Read or Write
- E: Enable (Latch data)
- RS – Register Select

- RS = 0 → Command Register
- RS = 1 → Data Register

- R/W = 0 → Write , R/W = 1 → Read
- E – Enable
  - Used to latch the data present on the data pins.
- D0 – D7
  - Bi-directional data/command pins.
  - Alphanumeric characters are sent in ASCII format.

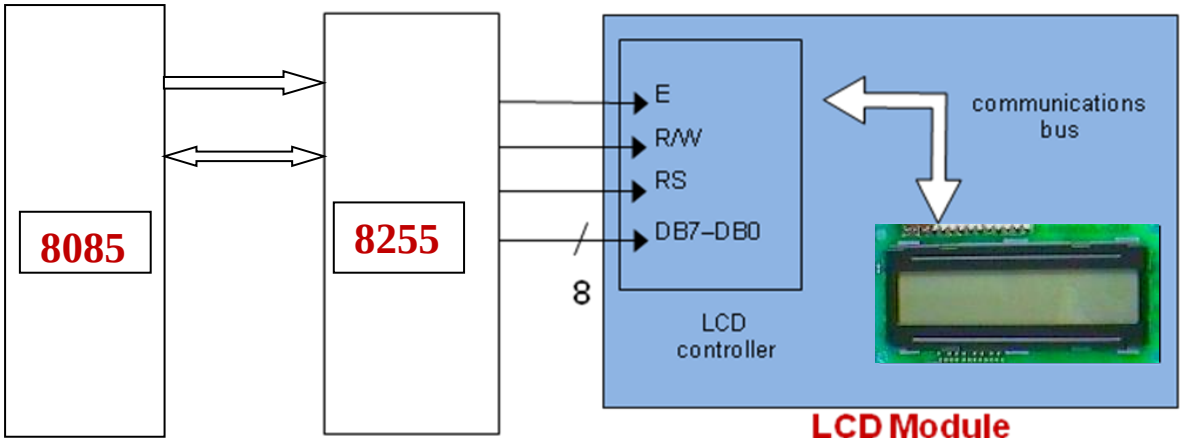


Figure 2.24 LCD Interface with 8085

LCD Commands

- The LCD's internal controller can accept several commands and modify the display accordingly. These commands would be things like:
  - Clear screen
  - Return home
  - Decrement/Increment cursor
- After writing to the LCD, it takes some time for it to complete its internal operations. During this time, it will not accept any new commands or data.
  - We need to insert time delay between any two commands or data sent to LCD

LCD Pin Description

Pin	Symbol	I/O	Description
1	V_{SS}	--	Ground
2	V_{CC}	--	+5V power supply
3	V_{EE}	--	Power supply source to control contrast
4	RS	I	Register select: RS=0 to select instruction command register, RS = 1 to select data register
5	R/$\overline{W}$	I	Read/write: R/W=0 for write, R/W=1 for read
6	E	I	Enable
7	DB0	I/O	The 8-bit data bus
8	DB1	I/O	" "
9	DB2	I/O	" "
10	DB3	I/O	" "
11	DB4	I/O	" "
12	DB5	I/O	" "
13	DB6	I/O	" "
14	DB7	I/O	" "

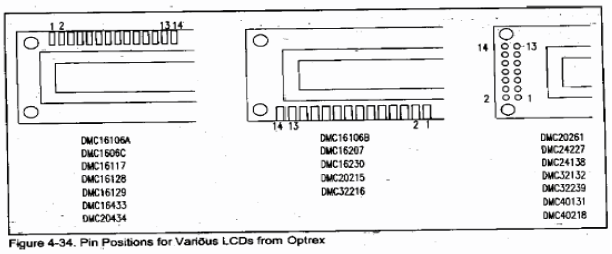


Figure 4-34. Pin Positions for Various LCDs from Optrex

LCD Command codes

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

Note: This table is extracted from Table 4-10.

LCD Addressing

```
16 x 2 LCD
80 81 82 83 84 85 86 through 8F
C0 C1 C2 C3 C4 C5 C6 through CF
20 x 1 LCD
80 81 82 83       through 93
20 x 2 LCD
80 81 82 83       through 93
C0 C1 C2 C3       through D3
20 x 4 LCD
80 81 82 83       through 93
C0 C1 C2 C3       through D3
94 95 96 97       through A7
D4 D5 D6 D7       through E7
40 x 2 LCD
80 81 82 83       through A7
C0 C1 C2 C3       through E7
```

Note: All data is in hex.

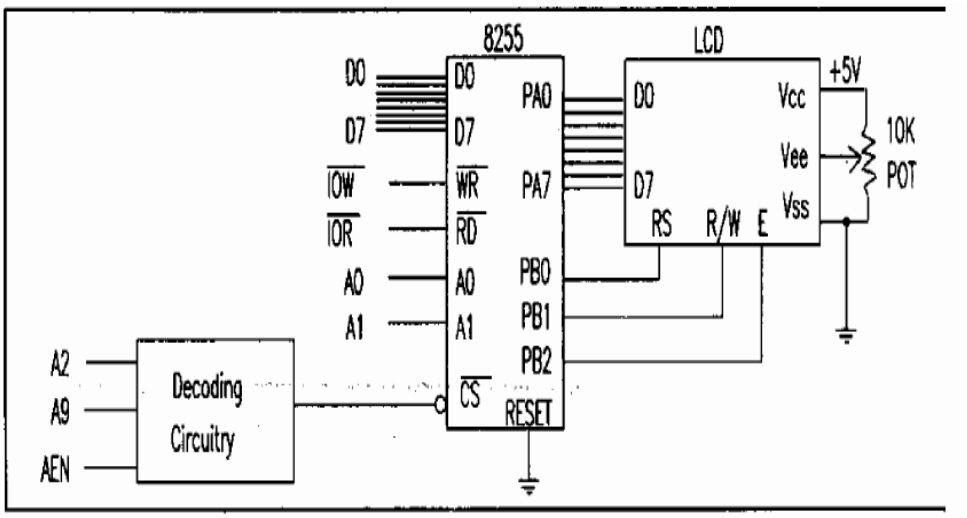


Figure 2.25 8255-to-PC Interface Connection to LCD

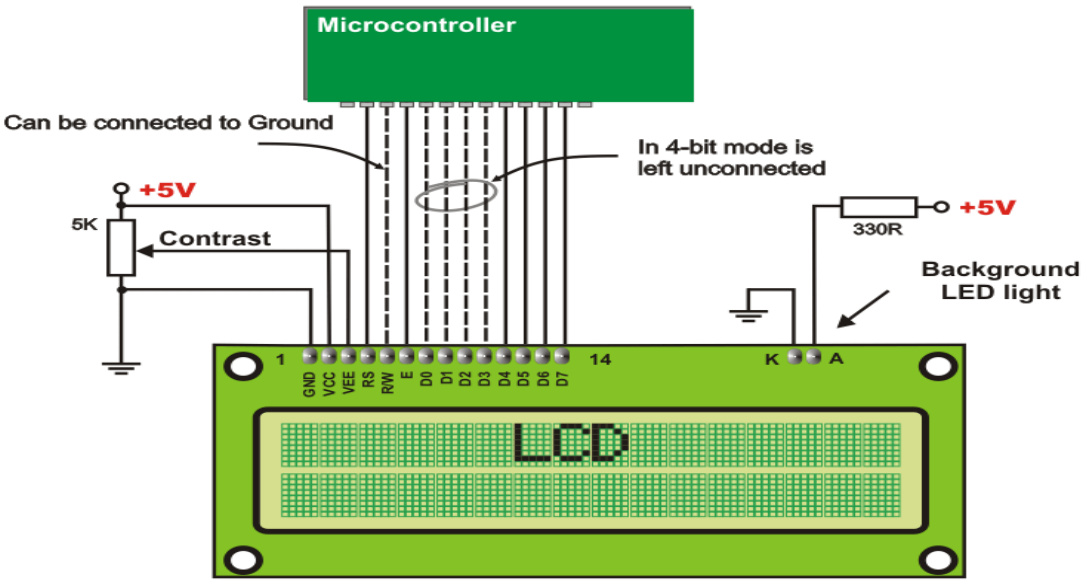


Figure 2.26

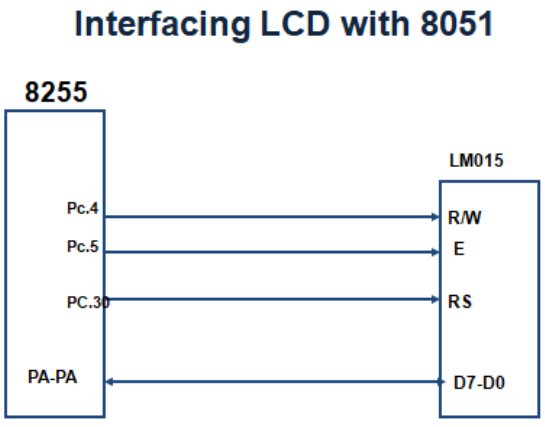


Figure 2.27

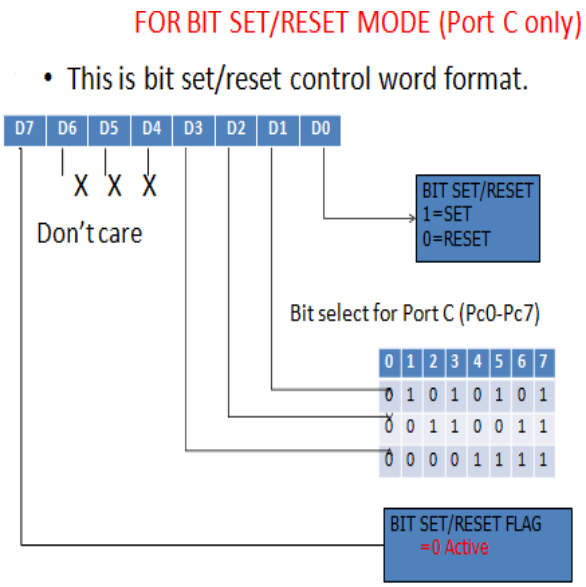


Figure 2.28

Program

```
mov A, command
call cmd
delay
mov A, another_cmd
call cmd
delay
mvi A, 'A'
call data
delay
mvi A, #'B'
call data
delay
....
```

Command and Data Write Routines

```
data:  mov B,A
mov A, 80          ; port A as output port
mov CR, A          ; write the control word into the control reg.
mov A,B
```

```
mov PA,A          ;A has the cmd word
mvi A,06          ; Control word to make PC3 as 0 i.e. Reg. select
mov CR, A
mvi A,09          ;Control word to make PC4 as 1      i.e. Write signal
mov CR, A
mvi  A,0B         ; control word to make enable as high
mov CR, A
mvi  A,05         ;control word to make enable as low
mov CR, A
ret

cmd:  mov B,A
mov A, 80         ; port A as output port
mov CR, A         ; write the control word into the control reg.
mov A,B
mov PA,A          ;A has the cmd word
mvi A,07          ; Control word to make PC3 as 1 i.e. Reg. select
mov CR, A
mvi A,09          ; Control word to make PC4 as 1     i.e. Write signal
mov CR, A
mvi  A,0B         ; control word to make enable as high
mov CR, A
mvi  A,05         ;control word to make enable as low
mov CR, A
ret
```

SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

UNIT 3 - Fundamentals of Microprocessor and Microcontroller – SBMA1501

INTRODUCTION TO 8051

Introduction to 8-bit Microcontrollers – 8051, Microcontroller Architecture – Internal RAM & Internal ROM, Instruction set, Addressing Modes, Simple programs.

ARCHITECTURE OF 8051 MICROCONTROLLER

An 8051 microcontroller has the following 11 major components:

1. ALU (Arithmetic and Logic Unit)
2. PC (Program Counter)
3. Registers
4. Timers and counters
5. Internal RAM and ROM
6. Four general purpose parallel input/output ports
7. Interrupt control logic with five sources of interrupt
8. Serial date communication
9. PSW (Program Status Word)
10. Data Pointer (DPTR)
11. Stack Pointer (SP)

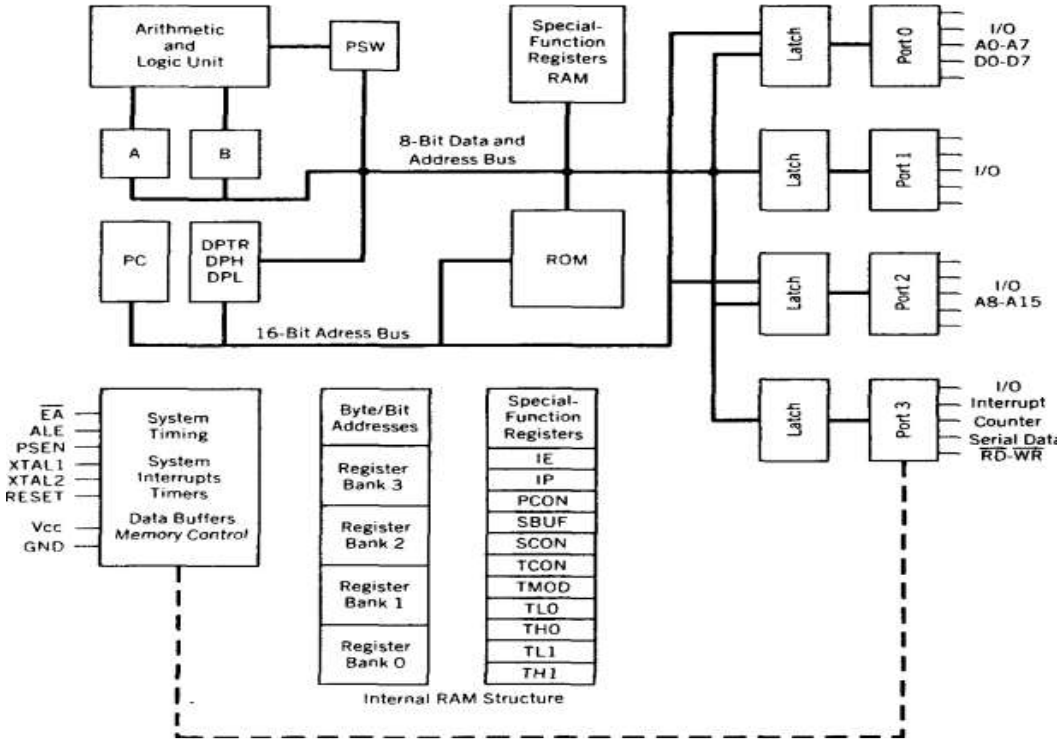


Figure 3.1: 8051- Architecture

Features of 8051

- 8 bit cpu with registers A and B
- 16 bit PC and DPTR(data pointer).

- 8 bit program status word(PSW)
- 8 bit Stack Pointer
- 4K Internal ROM
- 128bytes Internal RAM
  →4 register banks each having 8 registers
  →16 bytes, which may be addressed at the bit level.
  →80 bytes of general purpose data memory
- 32 i/o pins arranged as 4 8 bit ports:P0 to P3
- Two 16 bit timer/counters:T0 and T1
- Full duplex serial data receiver/transmitter
- Control registers: TCON,TMOD,SCON,PCON,IP and IE
- Two external and Three internal interrupt sources.
- Oscillator and Clock Circuits.

ALU

All arithmetic and logical functions are carried out by the ALU. Addition, subtraction with carry, and multiplication come under arithmetic operations. Logical AND, OR and exclusive OR (XOR) come under logical operations.

Program Counter (PC)

A program counter is a 16 - bit register and it has no internal address. The basic function of program counter is to fetch from memory the address of the next instruction to be executed. The PC holds the address of the next instruction residing in memory and when a command is encountered, it produces that instruction. This way the PC increments automatically, holding the **address of the next instruction.**

Registers

Registers are usually known as data storage devices. 8051 microcontroller has 2 registers, namely Register A and Register B. Register A serves as an accumulator while Register B functions as a general purpose register. These registers are used to store the output of mathematical and logical instructions. The operations of addition, subtraction, multiplication and division are carried out by Register A. Register B is usually unused and comes into picture only when multiplication and division functions are carried out by

Register A. Register A also involved in data transfers between the microcontroller and external memory.

8051 microcontroller also has 7 Special Function Registers (SFRs). They are:

- Serial Port Data Buffer (SBUF)
- Timer/Counter Control (TCON)
- Timer/Counter Mode Control (TMOD)
- Serial Port Control (SCON)
- Power Control (PCON)
- Interrupt Priority (IP)
- Interrupt Enable Control (IE)

Timers and Counters

Synchronization among internal operations can be achieved with the help of clock circuits which are responsible for generating clock pulses. During each clock pulse a particular operation will be carried out, thereby, assuring synchronization among operations. For the formation of an oscillator, we are provided with two pins XTAL1 and XTAL2 which are used for connecting a resonant network in 8051 microcontroller device. In addition to this, circuit also consists of four more pins.

Internal operations can be synchronized using clock circuits which produce clock pulses. With each clock pulse, a particular function will be accomplished and hence synchronization is achieved. There are two pins XTAL1 and XTAL2 which form an oscillator circuit which connect to a resonant network in the microcontroller. The circuit also has 4 additional pins

EA: External enable
ALE: Address latch enable
PSEN: Program store enable and
RST: Reset
Quartz crystal is used to generate periodic clock pulses.

Internal RAM and ROM

ROM

A code of 4K memory is incorporated as on-chip ROM in 8051. The 8051 ROM is a non-volatile memory meaning that its contents cannot be altered and hence has a similar range of data and program memory, i.e, they can address program memory as well as a 64K separate block of data memory.

RAM

The 8051 microcontroller is composed of 128 bytes of internal RAM. This is a volatile memory since its contents will be lost if power is switched off. These 128 bytes of internal RAM are divided into 32 working registers which in turn constitute 4 register banks (Bank 0 - Bank 3) with each bank consisting of 8 registers (R0 - R7). There are 128 addressable bits in the internal RAM.

I/O Ports

The 8051 microcontroller has four 8-bit input/output ports.

PORT P0: When there is no external memory present, this port acts as a general purpose input/output port. In the presence of external memory, it functions as a multiplexed address and data bus. It performs a dual role.

PORT P1: This port is used for various interfacing activities. This 8 -bit port is a normal I/O port i.e. it does not perform dual functions.

PORT P2: Similar to PORT P0, this port can be used as a general purpose port when there is no external memory but when external memory is present it works in conjunction with PORT PO as an address bus. This is an 8 -bit port and performs dual functions.

PORT P3: PORT P3 behaves as a dedicated I/O port

PROGRAMMING MODEL OF 8051 (Memory architecture)

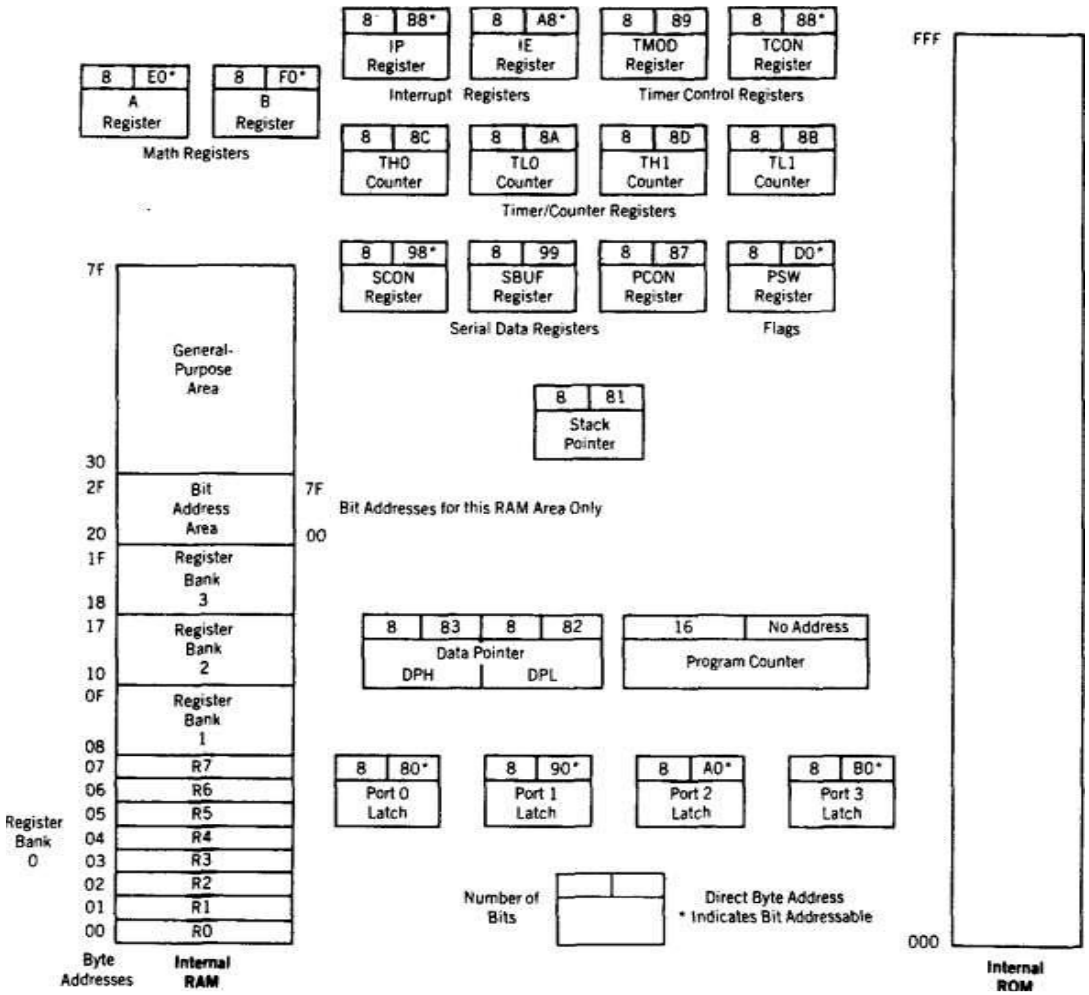


Figure 3.2 Programming Model

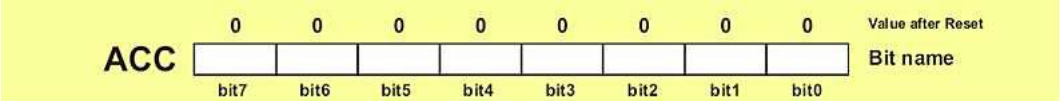
The above diagram shows the programming model of 8051.

Special Function Registers (SFRs)

Special Function Registers (SFRs) are a sort of control table used for running and monitoring the operation of the microcontroller. Each of these registers as well as each bit they include, has its name, address in the scope of RAM and precisely defined purpose such as timer control, interrupt control, serial communication control etc. Even though there are 128 memory locations intended to be occupied by them, the basic core, shared by all types of 8051 microcontrollers, has only 21 such registers. Rest of locations are intentionally left unoccupied in order to enable the manufacturers to further develop microcontrollers keeping them compatible with the previous versions. It also enables programs written a long time ago for microcontrollers which are out of production now to be used today.

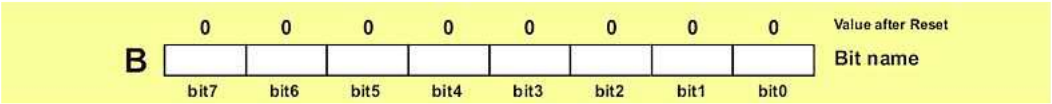
A Register (Accumulator)

A register is a general-purpose register used for storing intermediate results obtained during operation. Prior to executing an instruction upon any number or operand it is necessary to store it in the accumulator first. All results obtained from arithmetical operations performed by the ALU are stored in the accumulator. Data to be moved from one register to another must go through the accumulator. In other words, the A register is the most commonly used register and it is impossible to imagine a microcontroller without it. More than half instructions used by the 8051 microcontroller use somehow the accumulator.



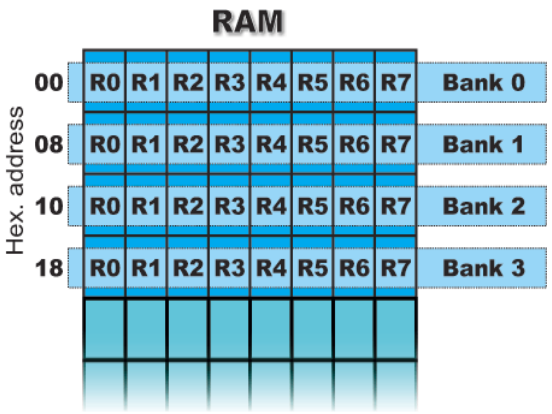
B Register

Multiplication and division can be performed only upon numbers stored in the A and B registers. All other instructions in the program can use this register as a spare accumulator



R Registers (R0-R7)

This is a common name for 8 general-purpose registers (R0, R1, R2 ...R7). Even though they are not true SFRs, they deserve to be discussed here because of their purpose. They occupy 4 banks within RAM. Similar to the accumulator, they are used for temporary storing variables and intermediate results during operation. Which one of these banks is to be active depends on two bits of the PSW Register. Active bank is a bank the registers of which are currently used.



PSW (Program Status Word)

PSW register is one of the most important SFRs. It contains several status bits that reflect the current state of the CPU. Besides, this register contains Carry bit, Auxiliary Carry, two register bank select bits, Overflow flag, parity bit and user-definable status flag.

P - Parity bit. If a number stored in the accumulator is even then this bit will be automatically set (1), otherwise it will be cleared (0). It is mainly used during data transmit and receive via serial communication.

- Bit 1. This bit is intended to be used in the future versions of microcontrollers.

OV Overflow occurs when the result of an arithmetical operation is larger than 255 and cannot be stored in one register. Overflow condition causes the OV bit to be set (1). Otherwise, it will be cleared (0).

RS0, RS1 - Register bank select bits. These two bits are used to select one of four register banks of RAM. By setting and clearing these bits, registers R0-R7 are stored in one of four banks of RAM.

RS1 RS2 Space in RAM

0 0 Bank0 00h-07h

0 1 Bank1 08h-0Fh

1 0 Bank2 10h-17h

1 1 Bank3 18h-1Fh

F0 - Flag 0. This is a general-purpose bit available for use.

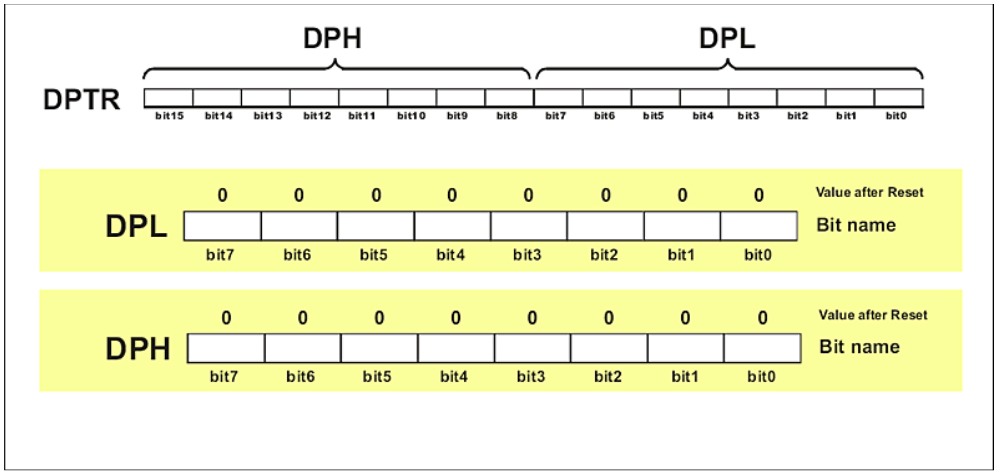
AC - Auxiliary Carry Flag is used for BCD operations only.

CY - Carry Flag is the (ninth) auxiliary bit used for all arithmetical operations and shift instructions.

Data Pointer Register (DPTR)

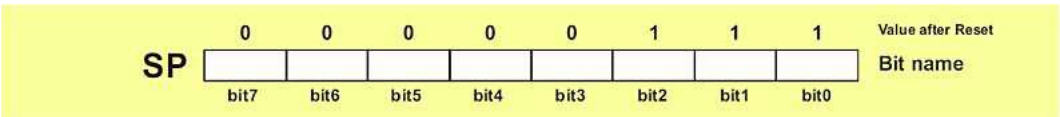
DPTR register is not a true one because it doesn't physically exist. It consists of two separate registers: DPH (Data Pointer High) and (Data Pointer Low). For this reason it may

be treated as a 16-bit register or as two independent 8-bit registers. Their 16 bits are primarly used for external memory addressing. Besides, the DPTR Register is usually used for storing data and intermediate results.



Stack Pointer (SP) Register

A value stored in the Stack Pointer points to the first free stack address and permits stack availability. Stack pushes increment the value in the Stack Pointer by 1. Likewise, stack pops decrement its value by 1. Upon any reset and power-on, the value 7 is stored in the Stack Pointer, which means that the space of RAM reserved for the stack starts at this location. If another value is written to this register, the entire Stack is moved to the new memory location.



P0, P1, P2, P3 – Input / Output Registers

If neither external memory nor serial communication system are used then 4 ports with in total of 32 input/output pins are available for connection to peripheral environment. Each bit within these ports affects the state and performance of appropriate pin of the microcontroller.

Thus, bit logic state is reflected on appropriate pin as a voltage (0 or 5 V) and vice versa, voltage on a pin reflects the state of appropriate port bit.

As mentioned, port bit state affects performance of port pins, i.e. whether they will be configured as inputs or outputs. If a bit is cleared (0), the appropriate pin will be configured as an output, while if it is set (1), the appropriate pin will be configured as an input. Upon reset and power-on, all port bits are set (1), which means that all appropriate pins will be configured as inputs.

Memory Organization

The 8051 has two types of memory and these are Program Memory and Data Memory. Program Memory (ROM) is used to permanently save the program being executed, while Data Memory (RAM) is used for temporarily storing data and intermediate results created and used during the operation of the microcontroller. Depending on the model in use (we are still talking about the 8051 microcontroller family in general) at most a few Kb of ROM and 128 or 256 bytes of RAM is used. However All 8051 microcontrollers have a 16-bit addressing bus and are capable of addressing 64 kb memory. It is neither a mistake nor a big ambition of engineers who were working on basic core development. It is a matter of smart memory organization which makes these microcontrollers a real “programmers’ goody“. Program Memory.

The first models of the 8051 microcontroller family did not have internal program memory. It was added as an external separate chip. These models are recognizable by their label beginning with 803 (for example 8031 or 8032). All later models have a few Kbyte ROM embedded. Even though such an amount of memory is sufficient for writing most of the programs, there are situations when it is necessary to use additional memory as well. A typical example is so called lookup tables. They are used in cases when equations describing some processes are too complicated or when there is no time for solving them. In such cases all necessary estimates and approximates are executed in advance and the final results are put in the tables (similar to logarithmic tables). How does the microcontroller handle external memory depends on the EA pin logic state:

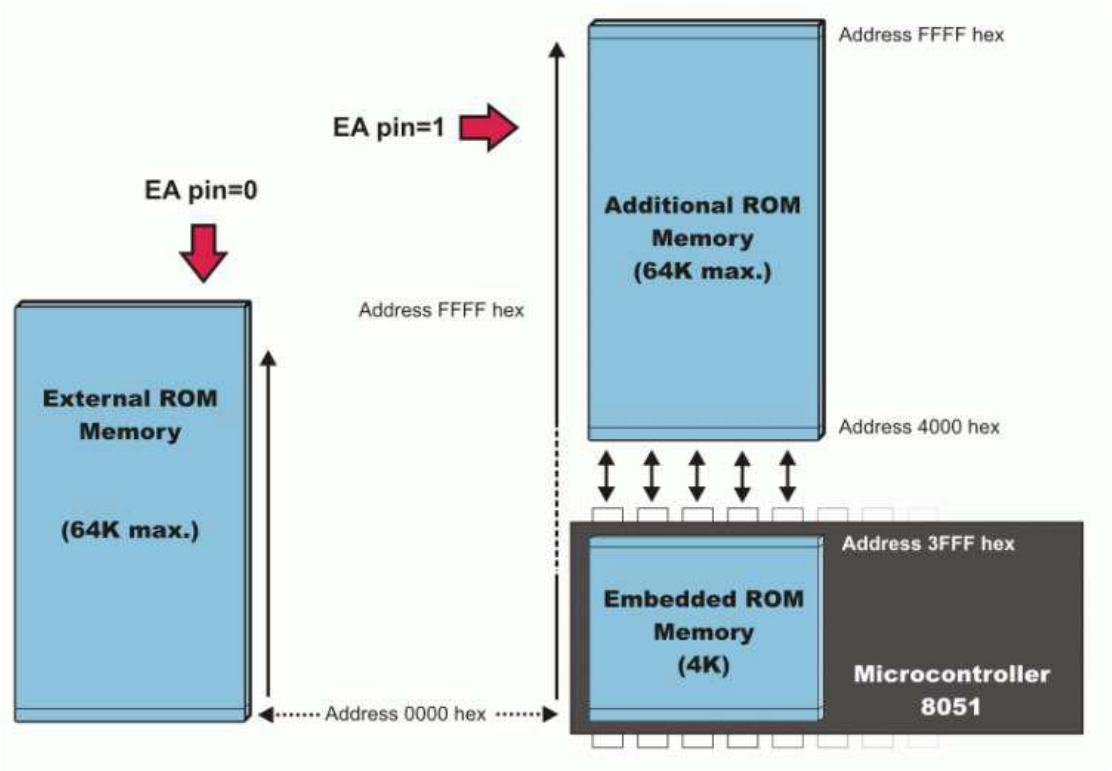


Figure 3.3 External memory EA pin

EA=0 In this case, the microcontroller completely ignores internal program memory and executes only the program stored in external memory.

EA=1 In this case, the microcontroller executes first the program from built-in ROM, then the program stored in external memory.

In both cases, P0 and P2 are not available for use since being used for data and address transmission. Besides, the ALE and PSEN pins are also used.

Data Memory

As already mentioned, Data Memory is used for temporarily storing data and intermediate results created and used during the operation of the microcontroller. Besides, RAM memory built in the 8051 family includes many registers such as hardware counters and timers, input/output ports, serial data buffers etc. The previous models had 256 RAM locations, while for the later models this number was incremented by additional 128 registers. However, the first 256 memory locations (addresses 0-FFh) are the heart of memory common to all the models belonging to the 8051 family. Locations available to the user occupy

memory space with addresses 0-7Fh, i.e. first 128 registers. This part of RAM is divided in several blocks.

The first block consists of 4 banks each including 8 registers denoted by R0-R7. Prior to accessing any of these registers, it is necessary to select the bank containing it. The next memory block (address 20h-2Fh) is bit- addressable, which means that each bit has its own address (0- 7Fh). Since there are 16 such registers, this block contains in total of 128 bits with separate addresses (address of bit 0 of the 20h byte is 0, while address of bit 7 of the 2Fh byte is 7Fh). The third group of registers occupy addresses 2Fh-7Fh, i.e. 80 locations, and does not have any special functions or features. Additional RAM

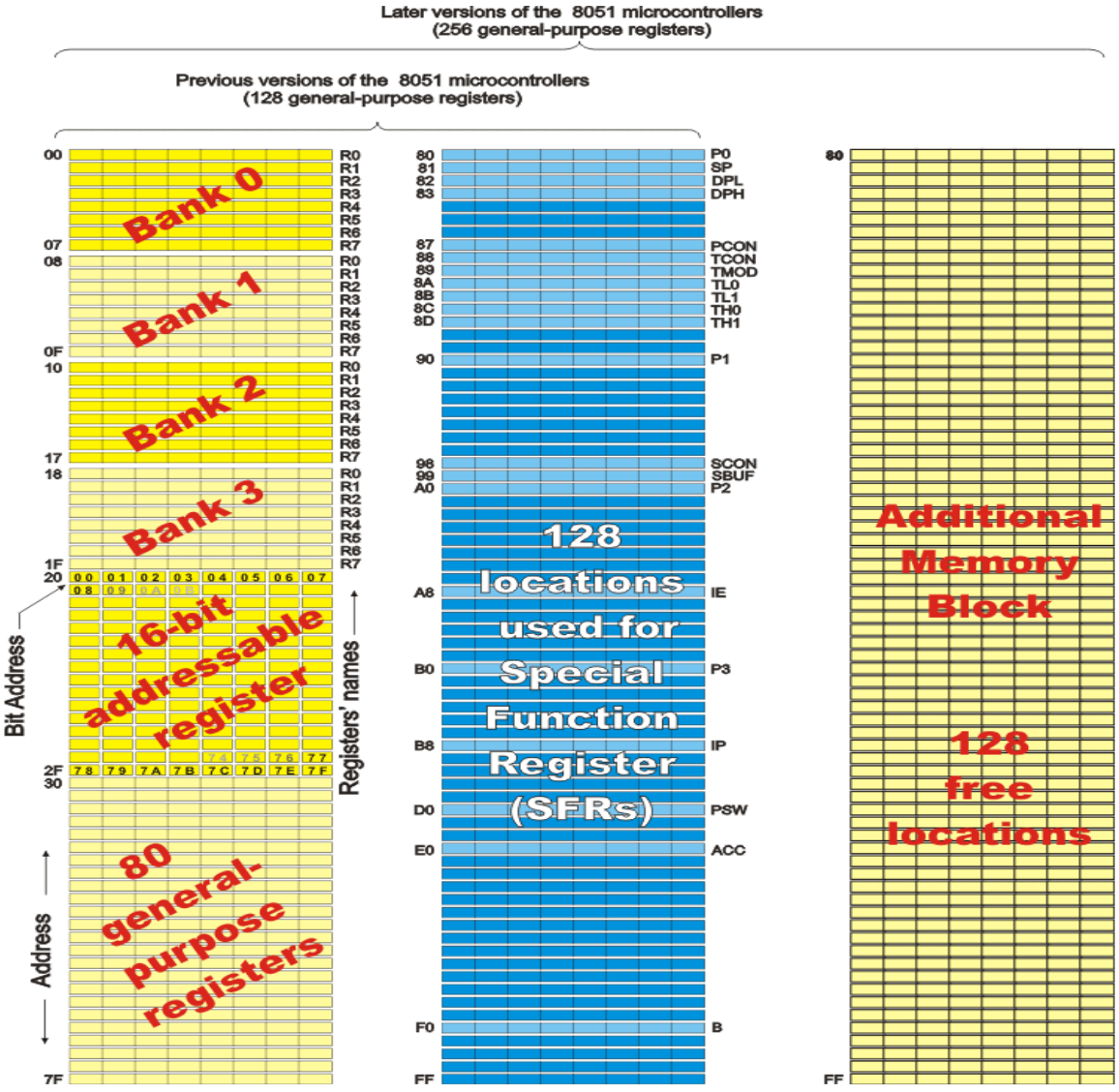


Figure 3.4 Internal RAM

In order to satisfy the programmers'constant hunger for Data Memory, the manufacturers decided to embed an additional memory block of 128 locations into the latest versions of the 8051 microcontrollers. However, it's not as simple as it seems to be… The problem is that electronics performing addressing has 1 byte (8 bits) on disposal and is capable of reaching only the first 256 locations, therefore. In order to keep already existing 8-bit architecture and compatibility with other existing models a small trick was done.

What does it mean? It means that additional memory block shares the same addresses with locations intended for the SFRs (80h- FFh). In order to differentiate between these two physically separated memory spaces, different ways of addressing are used. The SFRs memory locations are accessed by direct addressing, while additional RAM memory locations are accessed by indirect addressing.

Memory expansion

In case memory (RAM or ROM) built in the microcontroller is not sufficient, it is possible to add two external memory chips with capacity of 64Kb each. P2 and P3 I/O ports are used for their addressing and data transmission.

From the user's point of view, everything works quite simply when properly connected because most operations are performed by the microcontroller itself. The 8051 microcontroller has two pins for data read RD#(P3.7) and PSEN#. The first one is used for reading data from external data memory (RAM), while the other is used for reading data from external program memory (ROM). Both pins are active low. A typical example of memory expansion by adding RAM and ROM chips (Hardward architecture), is shown in figure above.

Even though additional memory is rarely used with the latest versions of the microcontrollers, we will describe in short what happens when memory chips are connected according to the previous schematic. The whole process described below is performed automatically.

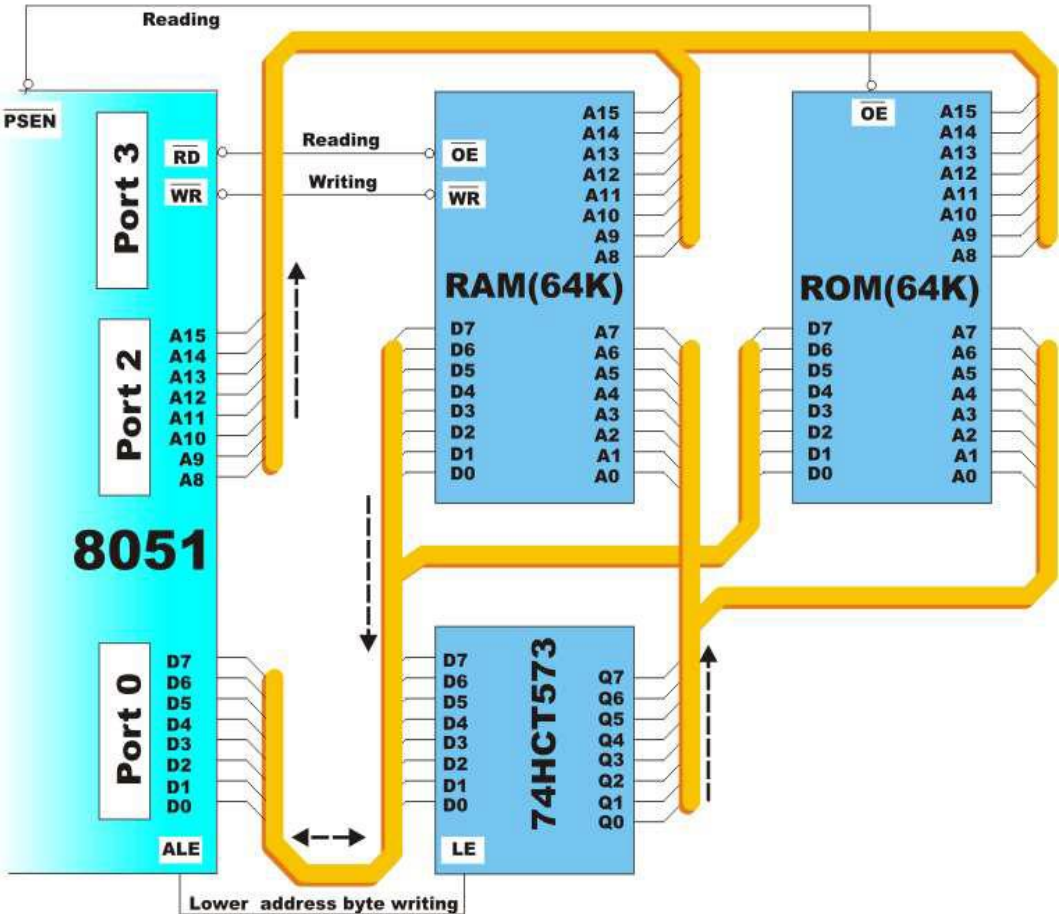


Figure 3.5 External Memory Interfacing

- When the program during execution encounters an instruction which resides in external memory (ROM), the microcontroller will activate its control output ALE and set the first 8 bits of address (A0-A7) on P0. IC circuit 74HCT573 passes the first 8 bits to memory address pins.

- A signal on the ALE pin latches the IC circuit 74HCT573 and immediately afterwards 8 higher bits of address (A8-A15) appear on the port. In this way, a desired location of additional program memory is addressed. It is left over to read its content.

- Port P0 pins are configured as inputs, the PSEN pin is activated and the microcontroller reads from memory chip. Similar occurs when it is necessary to read location from external RAM. Addressing is performed in the same way, while read and write are performed via signals appearing on the control outputs RD (is short for read) or WR (is short for write).

ADDRESSING MODES

An "addressing mode" refers to how you are addressing a given memory location. In summary, the addressing modes are as follows, with an example of each:

- Immediate Addressing MOV A,#20h
- Direct Addressing MOV A,30h
- Indirect Addressing MOV A,@R0
- External Direct MOVX A,@DPTR
- Code Indirect MOVC A,@A+DPTR

Each of these addressing modes provides important flexibility.

Immediate Addressing

Immediate addressing is so-named because the value to be stored in memory immediately follows the operation code in memory. That is to say, the instruction itself dictates what value will be stored in memory.

For example, the instruction:
MOV A,#6Ah

This instruction uses Immediate Addressing because the Accumulator will be loaded with the value that immediately follows; in this case 6A (hexidecimal). Immediate addressing is very fast since the value to be loaded is included in the instruction. However, since the value to be loaded is fixed at compile-time it is not very flexible.

Direct Addressing

Direct addressing is so-named because the value to be stored in memory is obtained by directly retrieving it from another memory location. For example:

MOV A,30h

This instruction will read the data out of Internal RAM address 30 (hexidecimal) and store it in the Accumulator. Direct addressing is generally fast since, although the value to be loaded isnt included in the instruction, it is quickly accessable since it is stored in the 8051s

Internal RAM. It is also much more flexible than Immediate Addressing since the value to be loaded is whatever is found at the given address--which may be variable.

Also, it is important to note that when using direct addressing any instruction which refers to an address between 00h and 7Fh is referring to Internal Memory. Any instruction which refers to an address between 80h and FFh is referring to the SFR control registers that control the 8051 microcontroller itself.

Indirect Addressing

Indirect addressing is a very powerful addressing mode which in many cases provides an exceptional level of flexibility. Indirect addressing is also the only way to access the extra 128 bytes of Internal RAM found on an 8052.

Indirect addressing appears as follows:

```
MOV A,@R0
```

This instruction causes the 8051 to analyze the value of the R0 register. The 8051 will then load the accumulator with the value from Internal RAM which is found at the address indicated by R0.

For example, lets say R0 holds the value 40h and Internal RAM address 40h holds the value 67h. When the above instruction is executed the 8051 will check the value of R0. Since R0 holds 40h the 8051 will get the value out of Internal RAM address 40h (which holds 67h) and store it in the Accumulator. Thus, the Accumulator ends up holding 67h. Indirect addressing always refers to Internal RAM; it never refers to an SFR. Thus, in a prior example we mentioned that SFR 99h can be used to write a value to the serial port. Thus one may think that the following would be a valid solution to write the value 1 to the serial port:

```
MOV R0,#99h ;Load the address of the serial port
MOV @R0,#01h ;Send 01 to the serial port -- WRONG!!
```

This is not valid. Since indirect addressing always refers to Internal RAM these two instructions would write the value 01h to Internal RAM address 99h on an 8052. On an 8051 these two instructions would produce an undefined result since the 8051 only has 128 bytes of Internal RAM.

Direct

External Memory is accessed using a suite of instructions which use what I call "External Direct" addressing. I call it this because it appears to be direct addressing, but it is used to access external memory rather than internal memory.

There are only two commands that use External Direct addressing mode:

MOVX A, @DPT R

MOVX @DPTR, A

Both commands utilize DPTR. In these instructions, DPTR must first be loaded with the address of external memory that you wish to read or write. Once DPTR holds the correct external emory address, the first command will move the contents of that external memory address into the Accumulator. The second command will do the opposite: it will allow you to write the value of the Accumulator to the external memory address pointed to by DPTR.

External Indirect

External memory can also be accessed using a form of indirect addressing which I call External Indirect addressing. This form of addressing is usually only used in relatively small projects that have a very small amount of external RAM. An example of this addressing mode is:

MOVX @R0, A

Once again, the value of R0 is first read and the value of the Accumulator is written to that address in External RAM. Since the value of @R0 can only be 00h through FFh the project would effectively be limited to 256 bytes of External RAM. There are relatively simple hardware/software tricks that can be implemented to access more than 256 bytes of memory using External Indirect addressing.

INSTRUCTION SET

The process of writing program for the microcontroller mainly consists of giving instructions (commands) in the specific order in which they should be executed in order to carry out a specific task. As electronics cannot “understand” what for example an instruction “if the push button is pressed- turn the light on” means, then a certain number of simpler and precisely defined orders that decoder can recognise must be used. All commands are known

as INSTRUCTION SET. All microcontrollers compatibile with the 8051 have in total of 255 instructions, i.e. 255 different words available for program writing.

At first sight, it is imposing number of odd signs that must be known by heart. However, It is not so complicated as it looks like. Many instructions are considered to be “different”, even though they perform the same operation, so there are only 111 truly different commands.

For example: ADD A,R0, ADD A,R1, ... ADD A,R7 are instructions that perform the same Operation (additon of the accumulator and register). Since there are 8 such registers, each instruction is counted separately. Taking into account that all instructions perform only 53 operations (addition, subtraction, copy etc.) and most of them are rarely used in practice, there are actually 20-30 abbreviations to be learned, which is acceptable.

Types of instructions

Depending on operation they perform, all instructions are divided in several groups:

- Arithmetic Instructions
- Branch Instructions
- Data Transfer Instructions
- Logic Instructions
- Bit-oriented Instructions

The first part of each instruction, called MNEMONIC refers to the operation an instruction performs (copy, addition, logic operation etc.). Mnemonics are abbreviations of the name of operation being executed. For example:

- INC R1 - Means: Increment register R1 (increment register R1);
- LJMP LAB5 - Means: Long Jump LAB5 (long jump to the address marked as LAB5);
- JNZ LOOP - Means: Jump if Not Zero LOOP (if the number in the accumulator is not 0, jump to the address marked as LOOP);

The other part of instruction, called OPERAND is separated from mnemonic by at least one whitespace and defines data being processed by instructions. Some of the

instructions have no operand, while some of them have one, two or three. If there is more than one operand in an instruction, they are separated by a comma. For example:

- RET - return from a subroutine;
- JZ TEMP - if the number in the accumulator is not 0, jump to the address marked as TEMP;
- ADD A,R3 - add R3 and accumulator;
- CJNE A,#20,LOOP - compare accumulator with 20. If they are not equal, jump to the address marked as LOOP;

Arithmetic instructions

Arithmetic instructions perform several basic operations such as addition, subtraction, division, multiplication etc. After execution, the result is stored in the first operand. For example:

ADD A,R1 - The result of addition (A+R1) will be stored in the accumulator.

Arithmetic Instructions

Mnemonic Description

ADD A,Rn Adds the register to the accumulator

ADD A,direct Adds the direct byte to the accumulator

ADD A,@Ri Adds the indirect RAM to the accumulator

ADD A,#data Adds the immediate data to the accumulator

ADDC A,Rn Adds the register to the accumulator with a carry flag 1 1

ADDC A,direct Adds the direct byte to the accumulator with a carry flag 2 2

ADDC A,@Ri Adds the indirect RAM to the accumulator with a carry flag 1 2

ADDC A,#data Adds the immediate data to the accumulator with a carry flag 2 2

SUBB A,Rn Subtracts the register from the accumulator with a borrow 1 1

SUBB A,direct Subtracts the direct byte from the accumulator with a borrow 2 2

SUBB A,@Ri Subtracts the indirect RAM from the accumulator with a borrow 1 2

SUBB A,#data Subtracts the immediate data from the accumulator with a borrow 2 2

INC A Increments the accumulator by 1 1 1

INC Rn Increments the register by 1 1 2

INC Rx Increments the direct byte by 1 2 3

INC @Ri Increments the indirect RAM by 1 1 3

DEC A Decrements the accumulator by 1 1 1

DEC Rn Decrements the register by 1 1 1

DEC Rx Decrements the direct byte by 1 1 2

DEC @Ri Decrements the indirect RAM by 1 2 3

INC DPTR Increments the Data Pointer by 1 1 3

MUL AB Multiplies A and B 1 5

DIV AB Divides A by B 1 5

DA A Decimal adjustment of the accumulator according to BCD code 1 1

Branch Instructions

There are two kinds of branch instructions:

Unconditional jump instructions: upon their execution a jump to a new location from where the program continues execution is executed.

Conditional jump instructions: a jump to a new program location is executed only if a specified condition is met. Otherwise, the program normally proceeds with the next instruction.

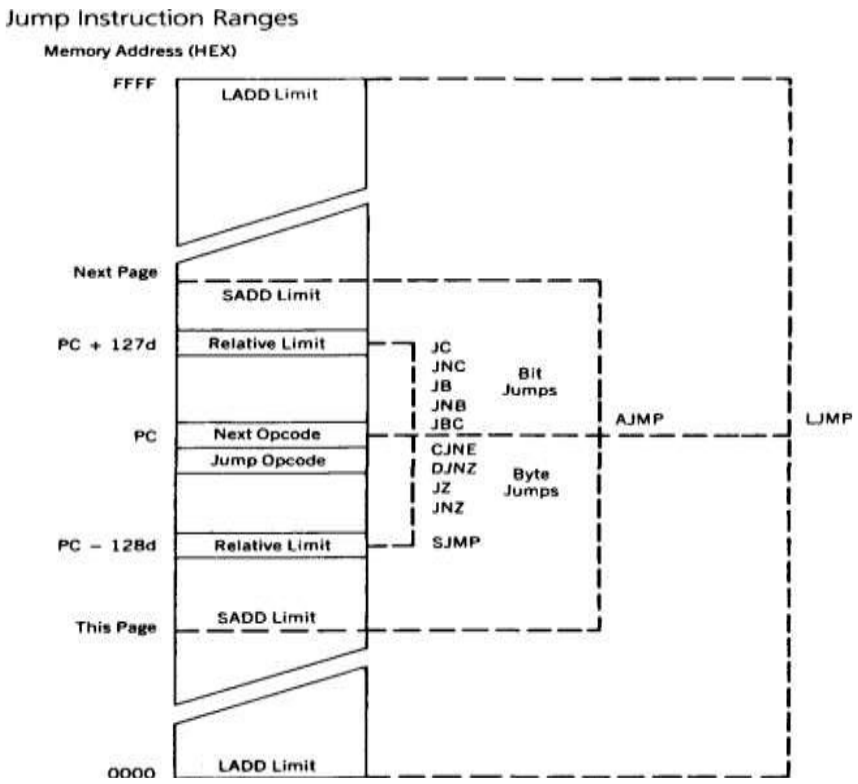


Figure 3.6 Jump Address Range

Branch Instructions

Mnemonic Description Byte Cycle

ACALL addr11 Absolute subroutine call 2 6

LCALL addr16 Long subroutine call 3 6

RET Returns from subroutine 1 4

RETI Returns from interrupt subroutine 1 4

AJMP addr11 Absolute jump 2 3

LJMP addr16 Long jump 3 4

SJMP rel

Short jump (from –128 to +127 locations relative to the following 2 3 instruction)

JC rel Jump if carry flag is set. Short jump. 2 3

JNC rel Jump if carry flag is not set. Short jump. 2 3

JB bit,rel Jump if direct bit is set. Short jump. 3 4

JBC bit,rel Jump if direct bit is set and clears bit. Short jump. 3 4

JMP @A+DPTR Jump indirect relative to the DPTR 1 2

JZ rel Jump if the accumulator is zero. Short jump. 2 3

JNZ rel Jump if the accumulator is not zero. Short jump. 2 3

CJNE A,direct,rel Compares direct byte to the accumulator and jumps if not equal. Short jump.

CJNE A,#data,rel Compares immediate data to the accumulator and jumps if not equal. Short jump.

CJNE Rn,#data,rel

CJNE

DJNZ Rn,rel Decrements register and jumps if not 0. Short jump. 2 3

DJNZ Rx,rel Decrements direct byte and jump if not 0. Short jump. 3 4

NOP No operation

Data Transfer Instructions

Data transfer instructions move the content of one register to another. The register the content of which is moved remains unchanged. If they have the suffix “X” (MOVX), the data is exchanged with external memory.

Mnemonic Description Byte Cycle

MOV A,Rn Moves the register to the accumulator 1 1

MOV A,direct Moves the direct byte to the accumulator 2 2

MOV A,@Ri Moves the indirect RAM to the accumulator 1 2

MOV A,#data Moves the immediate data to the accumulator 2 2

MOV Rn,A Moves the accumulator to the register 1 2

MOV Rn,direct Moves the direct byte to the register 2 4

MOV Rn,#data Moves the immediate data to the register 2 2

MOV direct,A Moves the accumulator to the direct byte 2 3

MOV direct,Rn Moves the register to the direct byte 2 3

MOV direct,direct Moves the direct byte to the direct byte 3 4

MOV direct,@Ri Moves the indirect RAM to the direct byte 2 4

MOV direct,#data Moves the immediate data to the direct byte 3 3

MOV @Ri,A Moves the accumulator to the indirect RAM 1 3

MOV @Ri,direct Moves the direct byte to the indirect RAM 2 5

MOV @Ri,#data Moves the immediate data to the indirect RAM 2 3

MOV DPTR,#data Moves a 16-bit data to the data pointer 3 3

MOVC A,@A+DPTR (address=A+DPTR) Moves the code byte relative to the DPTR to the accumulator 1 3

MOVC A,@A+PC Moves the code byte relative to the PC to the accumulator 1 3 (address=A+PC)

MOVX A,@Ri Moves the external RAM (8-bit address) to the accumulator 1 3-10

OVX A,@DPTR Moves the external RAM (16-bit address) to the accumulator 1 3-10

MOVX @Ri,A Moves the accumulator to the external RAM (8-bit address) 1 4-11

MOVX @DPTR,A

Moves the accumulator to the external RAM (16-bit address) 1 4-11

PUSH direct Pushes the direct byte onto the stack 2 4

POP direct Pops the direct byte from the stack/td> 2 3

XCH A,Rn Exchanges the register with the accumulator 1 2

XCH A,direct Exchanges the direct byte with the accumulator 2 3

XCH A,@Ri Exchanges the indirect RAM with the accumulator 1 3

XCHD A,@Ri Exchanges the low-order nibble indirect RAM with the 1 accumulator

Logic Instructions

Logic instructions perform logic operations upon corresponding bits of
execution, the result is stored in the first operand.
two registers. After

Logic Instructions

Mnemonic Description Byte Cycle

ANL A,Rn AND register to accumulator 1 1

ANL A,direct AND direct byte to accumulator 2 2

ANL A,@Ri AND indirect RAM to accumulator 1 2

ANL A,#data AND immediate data to accumulator 2 2

ANL direct,A AND accumulator to direct byte 2 3

ANL direct,#data AND immediae data to direct register 3 4

ORL A,Rn OR register to accumulator 1 1

ORL A,direct OR direct byte to accumulator 2 2

ORL A,@Ri OR indirect RAM to accumulator 1 2

ORL direct,A OR accumulator to direct byte 2 3

ORL direct,#data OR immediate data to direct byte 3 4

XRL A,Rn Exclusive OR register to accumulator 1 1

XRL A,direct Exclusive OR direct byte to accumulator 2 2

XRL A,@Ri Exclusive OR indirect RAM to accumulator 1 2

XRL A,#data Exclusive OR immediate data to accumulator 2 2

XRL direct,A Exclusive OR accumulator to direct byte 2 3

XORL direct,#data Exclusive OR immediate data to direct byte 3 4

CLR A Clears the accumulator 1 1

CPL A Complements the accumulator (1=0, 0=1) 1 1

SWAP A Swaps nibbles within the accumulator 1 1

RL A Rotates bits in the accumulator left 1 1

RLC A Rotates bits in the accumulator left through carry 1 1

RR A Rotates bits in the accumulator right 1 1

RRC A Rotates bits in the accumulator right through carry 1 1

Bit-oriented Instructions

Similar to logic instructions, bit-oriented instructions perform logic operations. The difference is that these are performed upon single bits.

Bit-oriented Instructions
Mnemonic Description Byte Cycle
CLR C Clears the carry flag 1 1
CLR bit Clears the direct bit 2 3
SETB C Sets the carry flag 1 1
SETB bit Sets the direct bit 2 3
CPL C Complements the carry flag 1 1
CPL bit Complements the direct bit 2 3
ANL C,bit AND direct bit to the carry flag 2 2
ANL C,/bit AND complements of direct bit to the carry flag 2 2
ORL C,bit OR direct bit to the carry flag 2 2
ORL C,/bit OR complements of direct bit to the carry flag 2 2
MOV C,bit Moves the direct bit to the carry flag 2 2
MOV bit,C Moves the carry flag to the direct bit 2 3

SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

SCHOOL OF BIO AND CHEMICAL ENGINEERING

DEPARTMENT OF BIOMEDICAL ENGINEERING

UNIT 4 - Fundamentals of Microprocessor and Microcontroller – SBMA1501

INTERNAL PERIPERALS OF 8051

Modes of Timer/Counter operation – Serial Port operation & Modes – Interrupt Structure of 8051 - Memory Interfacing with 8051 – I/O ports- Input and output devices interfacing with 8051.

4.1 COUNTERS AND TIMERS

The microcontroller oscillator uses quartz crystal for its operation. As the frequency of this oscillator is precisely defined and very stable, pulses it generates are always of the same width, which makes them ideal for time measurement. Such crystals are also used in quartz watches. In order to measure time between two events it is sufficient to count up pulses coming from this oscillator. That is exactly what the timer does. If the timer is properly programmed, the value stored in its register will be incremented (or decremented) with each coming pulse, i.e. once per each machine cycle. A single machine-cycle instruction lasts for 12 quartz oscillator periods, which means that by embedding quartz with oscillator frequency of 12MHz, a number stored in the timer register will be changed million times per second, i.e. each microsecond.

The 8051 microcontroller has 2 timers/counters called T0 and T1. As their names suggest, their main purpose is to measure time and count external events. Besides, they can be used for generating clock pulses to be used in serial communication, so called Baud Rate.

Timer T0

As seen in figure below, the timer T0 consists of two registers – TH0 and TL0 representing a low and a high byte of one 16-digit binary number.

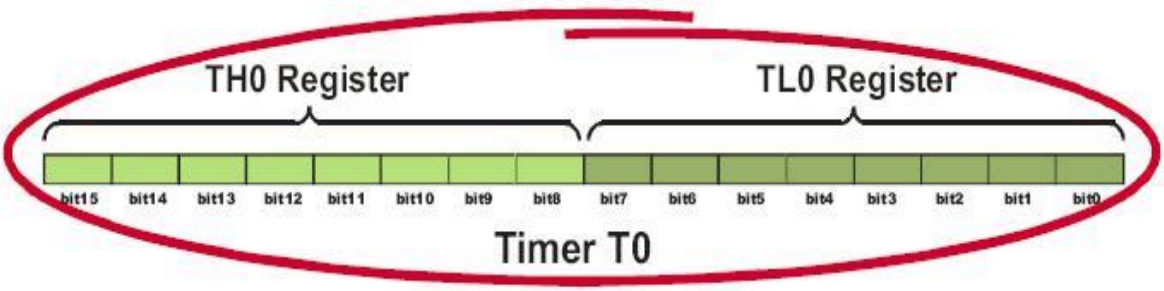


Figure 4.1 Timer T0

Accordingly, if the content of the timer T0 is equal to 0 (T0=0) then both registers it consists of will contain 0. If the timer contains for example number 1000 (decimal), then the TH0 register (high byte) will contain the number 3, while the TL0 register (low byte) will contain decimal number 232.

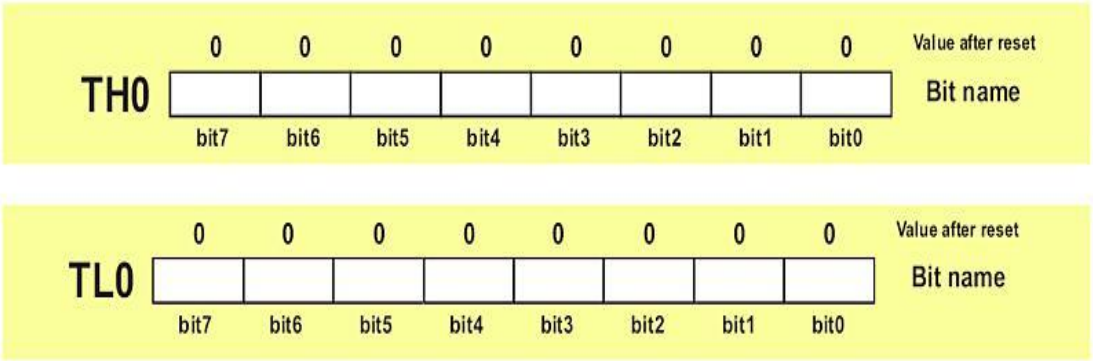


Figure 4.2

Formula used to calculate values in these two registers is very simple: TH0×256+TL0=T
Matching the previous example it would be as follows: 3 × 256 + 232 = 1000

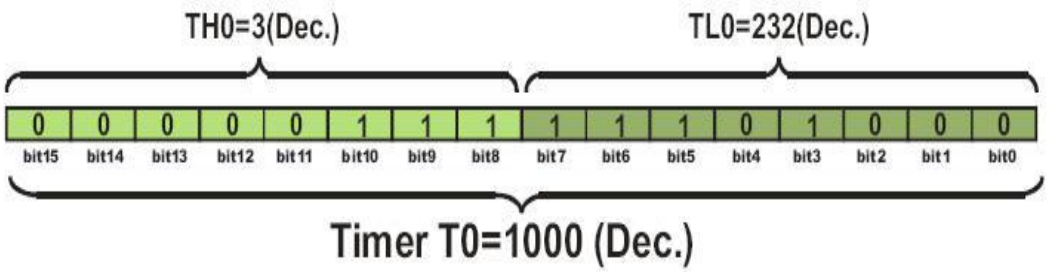


Figure 4.3

Since the timer T0 is virtually 16-bit register, the largest value it can store is 65 535. In case of exceeding this value, the timer will be automatically cleared and counting starts from 0. This condition is called an overflow. Two registers TMOD and TCON are closely connected to this timer and control its operation.

TMOD Register (Timer Mode)

The TMOD register selects the operational mode of the timers T0 and T1. As seen in figure below, the low 4 bits (bit0 - bit3) refer to the timer 0, while the high 4 bits (bit4 - bit7) refer to the timer 1. There are 4 operational modes and each of them is described herein.

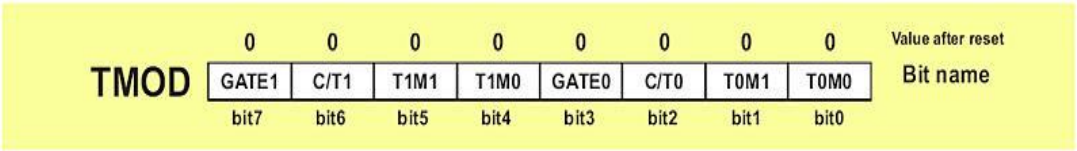


Figure 4.4 TMOD Register (Timer Mode)

Bits of this register have the following function:

GATE1 enables and disables Timer 1 by means of a signal brought to the INT1 pin (P3.3):

1 - Timer 1 operates only if the INT1 bit is set.

0 - Timer 1 operates regardless of the logic state of the INT1 bit.

C/T1 selects pulses to be counted up by the timer/counter 1:

1 - Timer counts pulses brought to the T1 pin (P3.5).

0 - Timer counts pulses from internal oscillator.

- **T1M1, T1M0** These two bits select the operational mode of the Timer 1.

Table 4.1

T1M1	T1M0	MODE	DESCRIPTION
0	0	0	13-bit timer
0	1	1	16-bit timer
1	0	2	8-bit auto-reload
1	1	3	Split mode

GATE0 enables and disables Timer 1 using a signal brought to the INT0 pin (P3.2):

1 - Timer 0 operates only if the INT0 bit is set.

0 - Timer 0 operates regardless of the logic state of the INT0 bit.

C/T0 selects pulses to be counted up by the timer/counter 0:

1 - Timer counts pulses brought to the T0 pin (P3.4).

0 - Timer counts pulses from internal oscillator.

T0M1,T0M0 These two bits select the oprtaional mode of the Timer 0.

Table 4.2

T0M1	T0M0	MODE	DESCRIPTION
0	0	0	13-bit timer
0	1	1	16-bit timer
1	0	2	8-bit auto-reload
1	1	3	Split mode

Timer 0 in mode 0 (13-bit timer)

This is one of the rarities being kept only for the purpose of compatibility with the previous versions of microcontrollers. This mode configures timer 0 as a 13-bit timer which consists of all 8 bits of TH0 and the lower 5 bits of TL0. As a result, the Timer 0 uses only 13 of 16 bits. How does it operate? Each coming pulse causes the lower register bits to change their states. After receiving 32 pulses, this register is loaded and automatically cleared, while the higher byte (TH0) is incremented by 1. This process is repeated until registers count up 8192 pulses. After that, both registers are cleared and counting starts from 0.

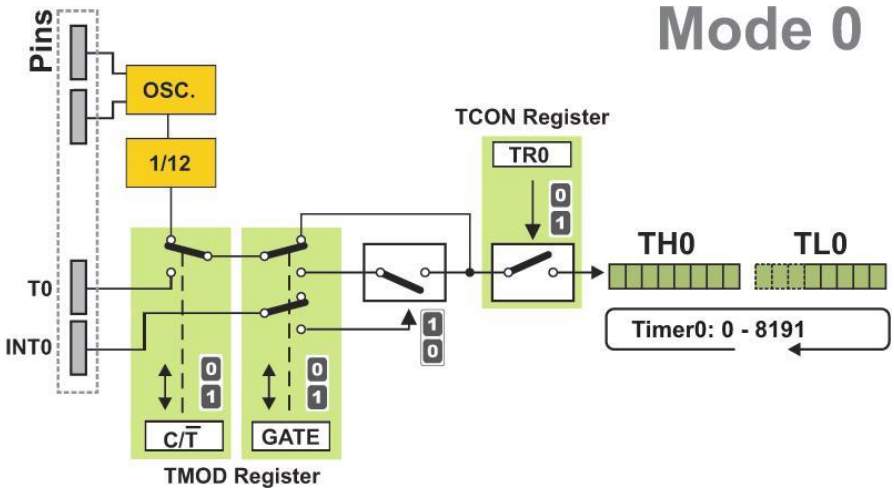


Figure 4.5 Timer 0 in mode 0 (13-bit timer)

Timer 0 in mode 1 (16-bit timer)

Mode 1 configures timer 0 as a 16-bit timer comprising all the bits of both registers TH0 and TL0. That's why this is one of the most commonly used modes. Timer operates in the same way as in mode 0, with difference that the registers count up to 65 536 as allowable by the 16 bits.

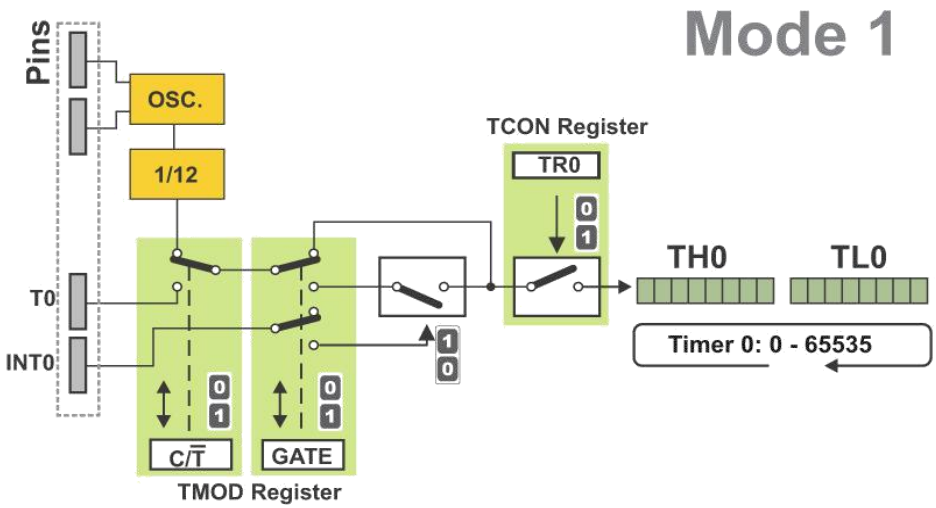


Figure 4.6 Timer 0 in mode 1 (16-bit timer)

Timer 0 in mode 2 (Auto-Reload Timer)

Mode 2 configures timer 0 as an 8-bit timer. Actually, timer 0 uses only one 8-bit register for counting and never counts from 0, but from an arbitrary value (0-255) stored in another (TH0) register.

The following example shows the advantages of this mode. Suppose it is necessary to constantly count up 55 pulses generated by the clock.

If mode 1 or mode 0 is used, It is necessary to write the number 200 to the timer registers and constantly check whether an overflow has occurred, i.e. whether they reached the value 255. When it happens, it is necessary to rewrite the number 200 and repeat the whole procedure. The same procedure is automatically performed by the microcontroller if set in mode 2. In fact, only the TL0 register operates as a timer, while another (TH0) register stores the value from which the counting starts. When the TL0 register is loaded, instead of being cleared, the contents of TH0 will be reloaded to it. Referring to the previous example, in order to register each 55th pulse, the best solution is to write the number 200 to the TH0 register and configure the timer to operate in mode 2.

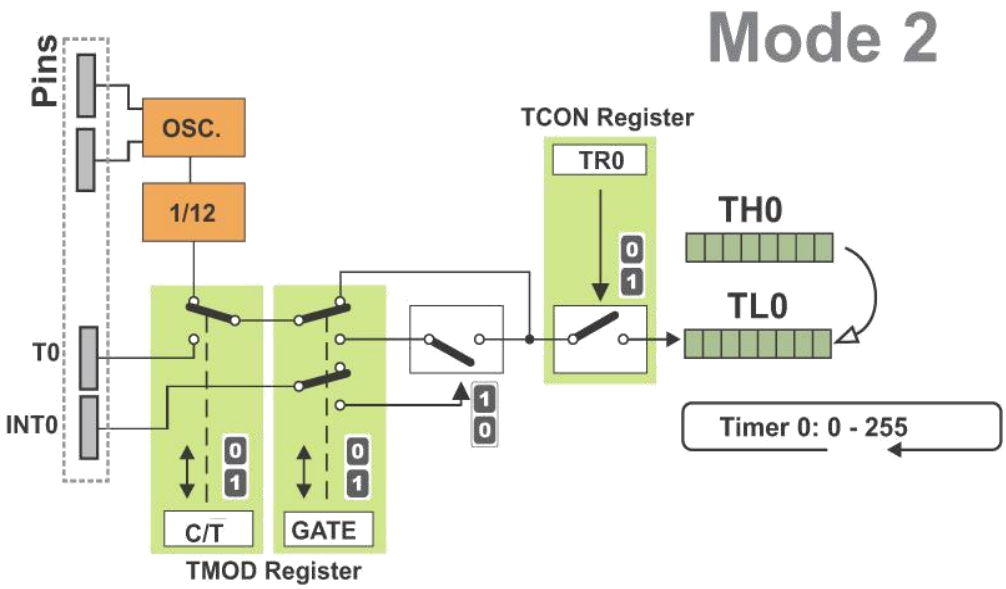


Figure 4.7 Timer 0 in mode 2 (Auto-Reload Timer)

Timer 0 in Mode 3 (Split Timer)

Mode 3 configures timer 0 so that registers TL0 and TH0 operate as separate 8-bit timers. In other words, the 16-bit timer consisting of two registers TH0 and TL0 is split into

two independent 8-bit timers. This mode is provided for applications requiring an additional 8-bit timer or counter. The TL0 timer turns into timer 0, while the TH0 timer turns into timer 1. In addition, all the control bits of 16-bit Timer 1 (consisting of the TH1 and TL1 register), now control the 8-bit Timer 1. Even though the 16-bit Timer 1 can still be configured to operate in any of modes (mode 1, 2 or 3), it is no longer possible to disable it as there is no control bit to do it. Thus, its operation is restricted when timer 0 is in mode 3.

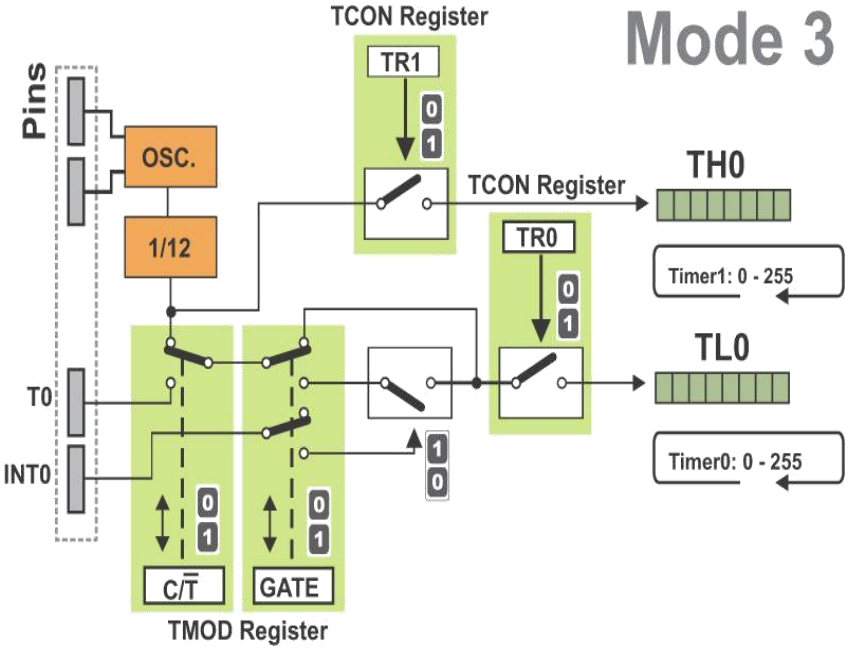


Figure 4.8 Timer 0 in Mode 3 (Split Timer)

The only application of this mode is when two timers are used and the 16-bit Timer 1 the operation of which is out of control is used as a baud rate generator.

Timer Control (TCON) Register

TCON register is also one of the registers whose bits are directly in control of timer operation.

Only 4 bits of this register are used for this purpose, while rest of them is used for interrupt control to be discussed later.

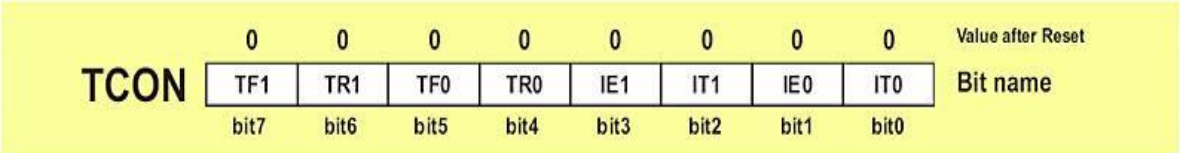


Figure 4.9 Timer Control (TCON) Register

- **TF1** bit is automatically set on the Timer 1 overflow.
- **TR1** bit enables the Timer 1.

1 - Timer 1 is enabled.

0 - Timer 1 is disabled.

- **TF0** bit is automatically set on the Timer 0 overflow.
- **TR0** bit enables the timer 0.

1 - Timer 0 is enabled.

0 - Timer 0 is disabled.

SERIAL I/O - UART (Universal Asynchronous Receiver and Transmitter)

One of the microcontroller features making it so powerful is an integrated UART, better known as a serial port. It is a full-duplex port, thus being able to transmit and receive data simultaneously and at different baud rates. Without it, serial data send and receive would be an enormously complicated part of the program in which the pin state is constantly changed and checked at regular intervals. When using UART, all the programmer has to do is to simply select serial port mode and baud rate. When it's done, serial data transmit is nothing but writing to the SBUF register, while data receive represents reading the same register. The microcontroller takes care of not making any error during data transmission.

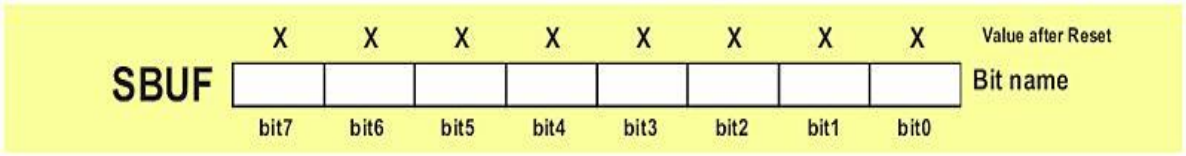


Figure 4.10 SERIAL I/O - UART

Serial port must be configured prior to being used. In other words, it is necessary to determine how many bits is contained in one serial "word", baud rate and synchronization clock source. The whole process is in control of the bits of the SCON register (Serial Control).

Serial Port Control (SCON) Register

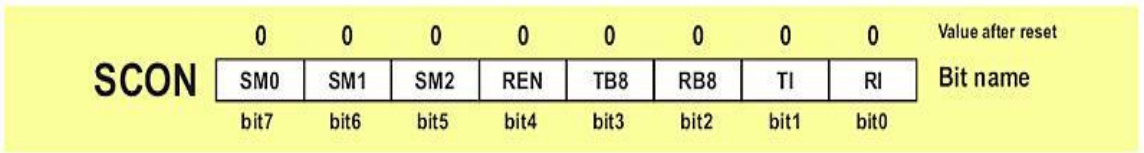


Figure 4.11 Serial Port Control (SCON) Register

- SM0 - Serial port mode bit 0 is used for serial port mode selection.
- SM1 - Serial port mode bit 1.
- SM2 - Serial port mode 2 bit, also known as multiprocessor communication enable bit. When set, it enables multiprocessor communication in mode 2 and 3, and eventually mode 1. It should be cleared in mode 0.
- REN - Reception Enable bit enables serial reception when set. When cleared, serial reception is disabled.
- TB8 - Transmitter bit 8. Since all registers are 8-bit wide, this bit solves the problem of transmitting the 9th bit in modes 2 and 3. It is set to transmit a logic 1 in the 9th bit.
- RB8 - Receiver bit 8 or the 9th bit received in modes 2 and 3. Cleared by hardware if 9th bit received is a logic 0. Set by hardware if 9th bit received is a logic 1.
- TI - Transmit Interrupt flag is automatically set at the moment the last bit of one byte is sent. It's a signal to the processor that the line is available for a new byte transmite. It must be cleared from within the software.
- RI - Receive Interrupt flag is automatically set upon one byte receive. It signals that byte is received and should be read quickly prior to being replaced by a new data. This bit is also cleared from within the software.

  As seen, serial port mode is selected by combining the SM0 and SM2 bits:

Table 4.3

S M 0	S M 1	MODE	DESCRIPTION	BAUD RATE
0	0	0	8-bit Shift Register	1/12 the quartz frequency
0	1	1	8-bit UART	Determined by the timer 1
1	0	2	9-bit UART	1/32 the quartz frequency (1/64 the quartz frequency)
1	1	3	9-bit UART	Determined by the timer 1

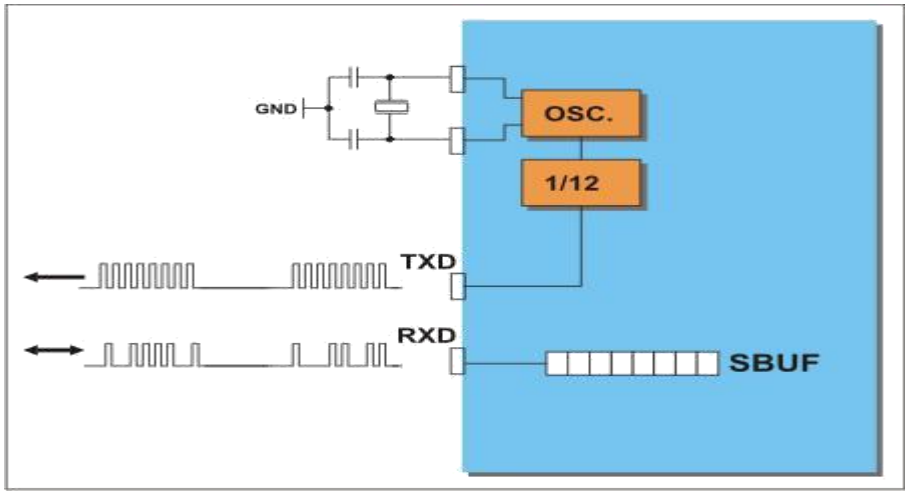


Figure 4.12

Mode o

In mode 0, serial data are transmitted and received through the RXD pin, while the TXD pin output clocks. The bout rate is fixed at 1/12 the oscillator frequency. On transmit, the least significant bit (LSB bit) is sent/received first.

TRANSMIT - Data transmit is initiated by writing data to the SBUF register. In fact, this process starts after any instruction being performed upon this register. When all 8 bits have been sent, the TI bit of the SCON register is automatically set.

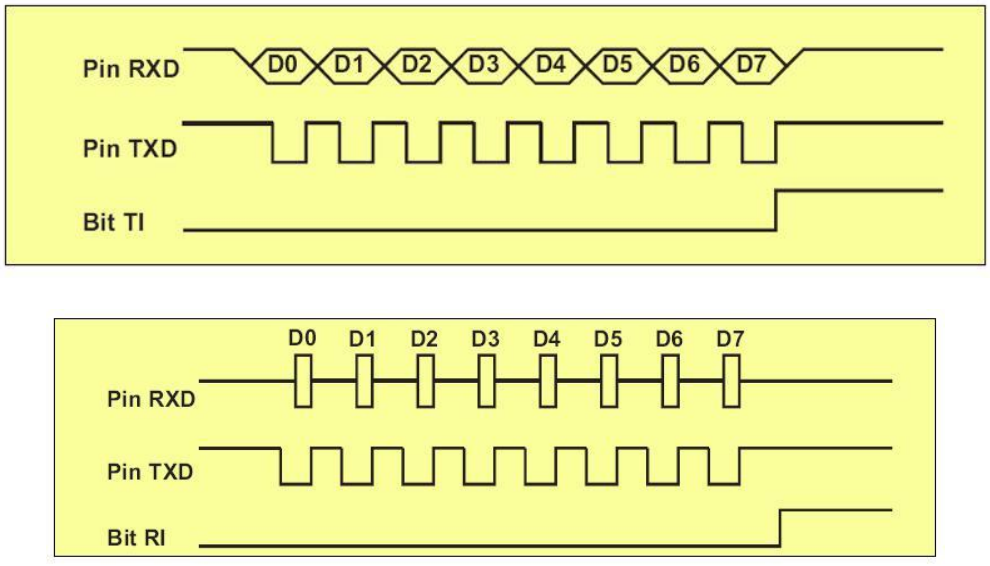


Figure 4.13

RECEIVE - Data receive through the RXD pin starts upon the two following conditions are met: bit REN=1 and RI=0 (both of them are stored in the SCON register).

When all 8 bits have been received, the RI bit of the SCON register is automatically set indicating that one byte receive is complete. Since there are no START and STOP bits or any other bit except data sent from the SBUF register in the pulse sequence, this mode is mainly used when the distance between devices is short, noise is minimized and operating speed is of importance.

Mode 1

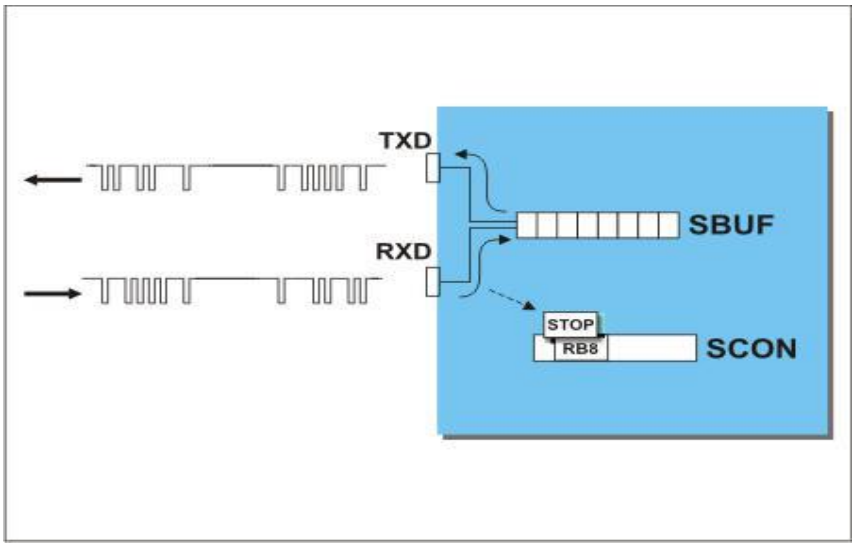


Figure 4.14

In mode 1, 10 bits are transmitted through the TXD pin or received through the RXD pin in the following manner: a START bit (always 0), 8 data bits (LSB first) and a STOP bit (always 1). The START bit is only used to initiate data receive, while the STOP bit is automatically written to the RB8 bit of the SCON register.

TRANSMIT - Data transmit is initiated by writing data to the SBUF register. End of data transmission is indicated by setting the TI bit of the SCON register.

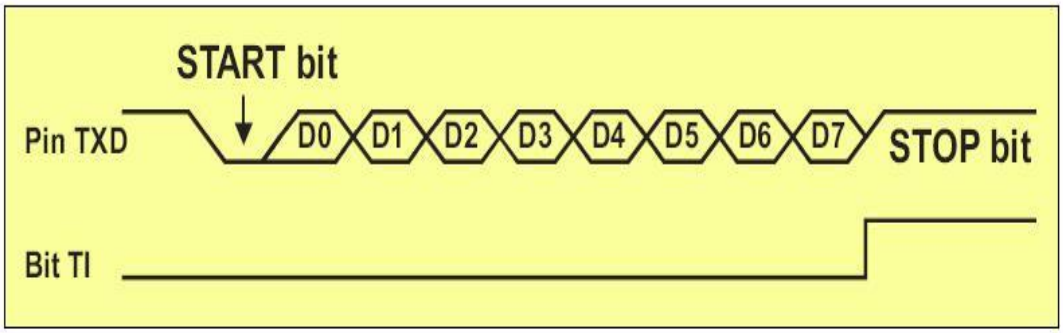


Figure 4.15

RECEIVE - The START bit (logic zero (0)) on the RXD pin initiates data receive. The following two conditions must be met: bit REN=1 and bit RI=0. Both of them are stored in the SCON register. The RI bit is automatically set upon data reception is complete. The Baud rate in this mode is determined by the timer 1 overflow.

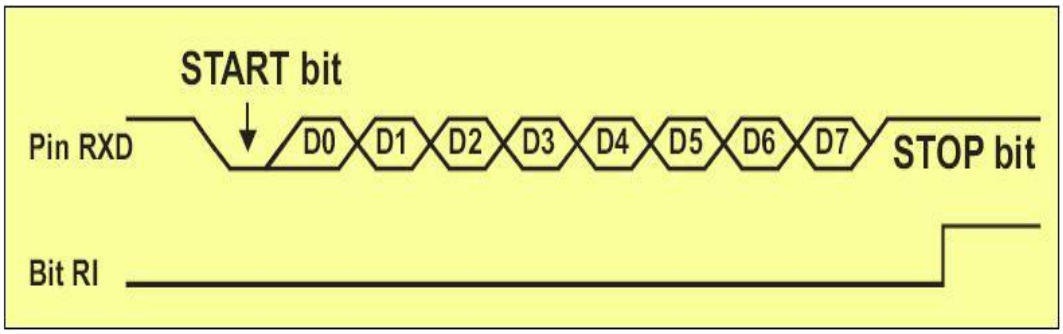


Figure 4.16

Mode 2

In mode 2, 11 bits are transmitted through the TXD pin or received through the RXD pin: a START bit (always 0), 8 data bits (LSB first), a programmable 9th data bit and a STOP bit (always 1). On transmit, the 9th data bit is actually the TB8 bit of the SCON register. This bit usually has a function of parity bit. On receive, the 9th data bit goes into the RB8 bit of the same register (SCON).The baud rate is either 1/32 or 1/64 the oscillator frequency.

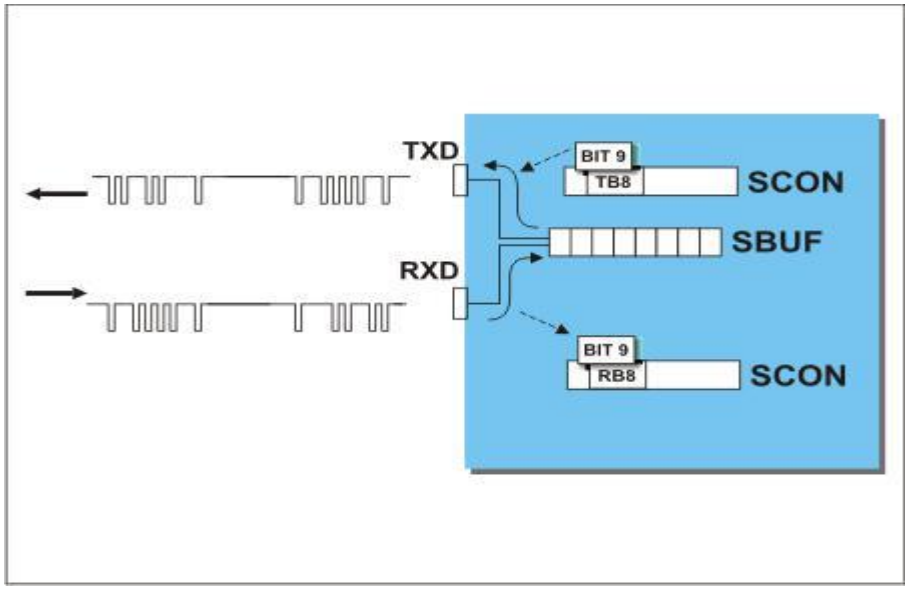


Figure 4.17

TRANSMIT - Data transmit is initiated by writing data to the SBUF register. End of data transmission is indicated by setting the TI bit of the SCON register.

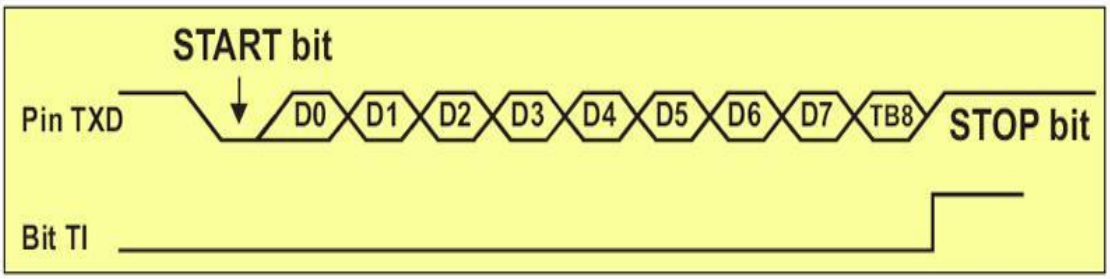


Figure 4.18

RECEIVE - The START bit (logic zero (0)) on the RXD pin initiates data receive. The following two conditions must be met: bit REN=1 and bit RI=0. Both of them are stored in the SCON register. The RI bit is automatically set upon data reception is complete.

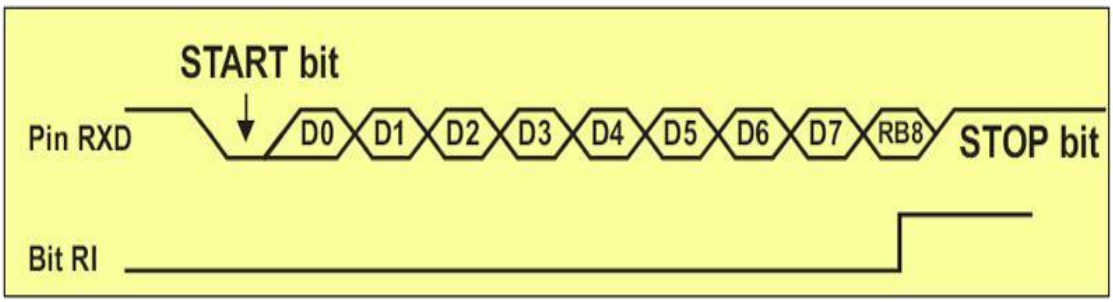


Figure 4.19

Mode 3

Mode 3 is the same as Mode 2 in all respects except the baud rate. The baud rate in Mode 3 is variable

Baud Rate

Baud Rate is a number of sent/received bits per second. In case the UART is used, baud rate depends on: selected mode, oscillator frequency and in some cases on the state of the SMOD bit of the SCON register. All the necessary formulas are specified in the table:

Table 4.4

Mode	BAUD RATE	BITSMOD
Mode 0	**Fosc. / 12**	
Mode 1	**1 Fosc.** **16 12 (256-TH1)**	**BitSMOD**
Mode 2	**Fosc. / 32** **Fosc. / 64**	**1** **0**
Mode 3	**1 Fosc.** **16 12 (256-TH1)**	

Multiprocessor Communication

As you may know, additional 9th data bit is a part of message in mode 2 and 3. It can be used for checking data via parity bit. Another useful application of this bit is in communication between two or more microcontrollers, i.e. multiprocessor communication. This feature is enabled by setting the SM2 bit of the SCON register. As a result, after receiving the STOP bit, indicating end of the message, the serial port interrupt will be generated only if the bit RB8 = 1 (the 9th bit).

Suppose there are several microcontrollers sharing the same interface. Each of them has its own address. An address byte differs from a data byte because it has the 9th bit set (1), while this bit is cleared (0) in a data byte. When the microcontroller A (master) wants to transmit a block of data to one of several slaves, it first sends out an address byte which identifies the target slave. An address byte will generate an interrupt in all slaves so that they can examine the received byte and check whether it matches their address.

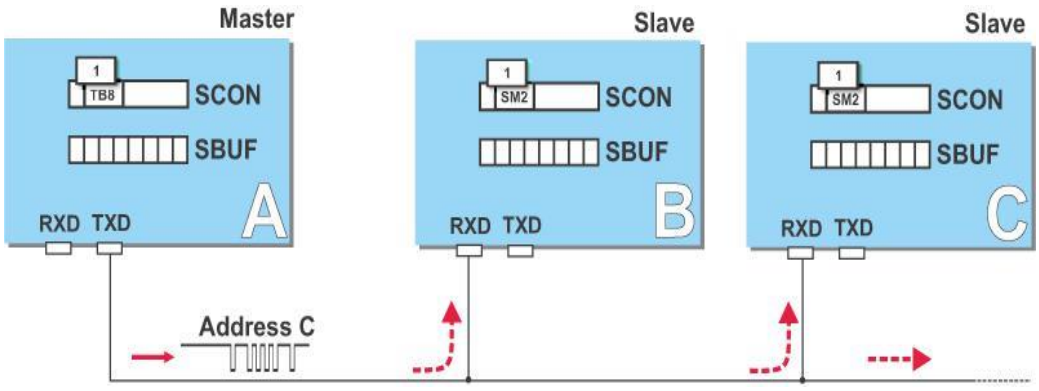


Figure 4.20

Of course, only one of them will match the address and immediately clear the SM2 bit of the SCON register and prepare to receive the data byte to come. Other slaves not being addressed leave their SM2 bit set ignoring the coming data bytes.

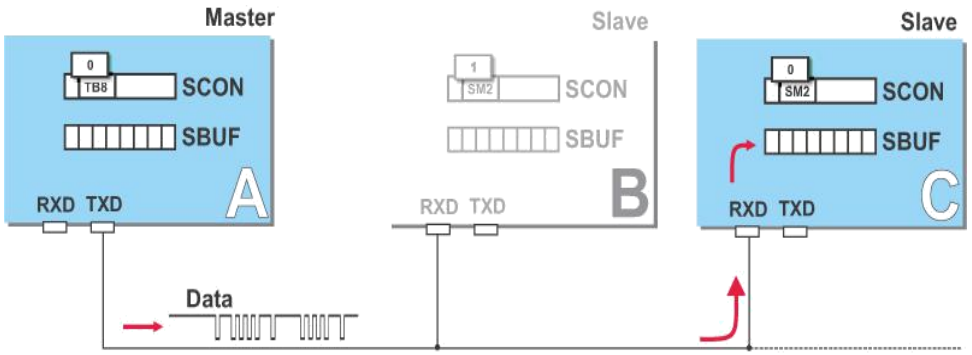


Figure 4.21

8051 Microcontroller Interrupts

There are five interrupt sources for the 8051, which means that they can recognize 5 different events that can interrupt regular program execution. Each interrupt can be enabled or disabled by setting bits of the IE register. Likewise, the whole interrupt system can be disabled by clearing the EA bit of the same register. Refer to figure below.

Now, it is necessary to explain a few details referring to external interrupts- INT0 and INT1. If the IT0 and IT1 bits of the TCON register are set, an interrupt will be generated on high to low transition, i.e. on the falling pulse edge (only in that moment). If these bits are cleared, an interrupt will be continuously executed as far as the pins are held low

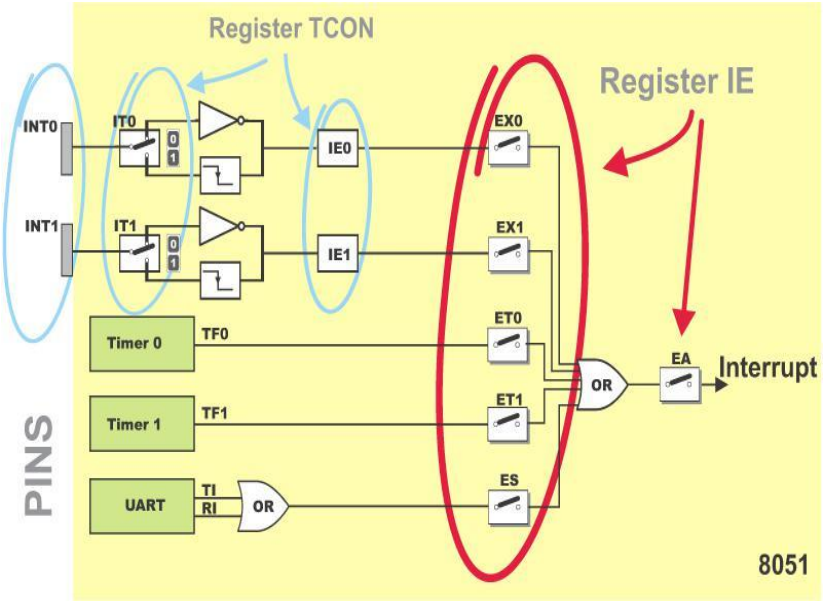


Figure 4.22

IE Register (Interrupt Enable)

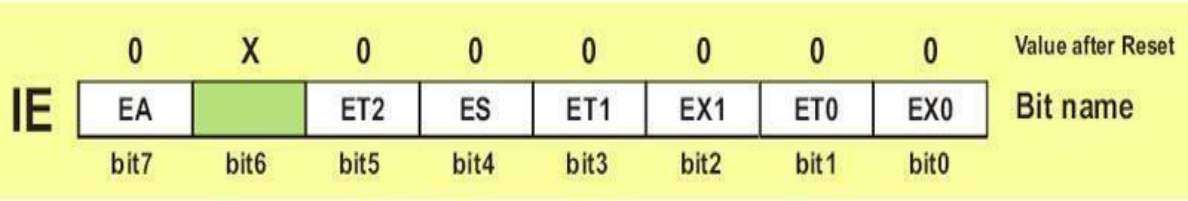


Figure 4.23

- **EA** - global interrupt enable/disable:
  0 - disables all interrupt requests.
  1 - enables all individual interrupt requests.
- **ES** - enables or disables serial interrupt:
  0 - UART system cannot generate an interrupt.
  1 - UART system enables an interrupt.
- **ET1** - bit enables or disables Timer 1 interrupt:
  0 - Timer 1 cannot generate an interrupt.
  1 - Timer 1 enables an interrupt.
- **EX1** - bit enables or disables external 1 interrupt:
  0 - change of the pin INT0 logic state cannot generate an interrupt.
  1 - enables an external interrupt on the pin INT0 state change.
- **ET0** - bit enables or disables timer 0 interrupt:
  0 - Timer 0 cannot generate an interrupt.
  1 - enables timer 0 interrupt.
- **EX0** - bit enables or disables external 0 interrupt:
  0 - change of the INT1 pin logic state cannot generate an interrupt.
  1 - enables an external interrupt on the pin INT1 state change.

Interrupt Priorities

It is not possible to forseen when an interrupt request will arrive. If several interrupts are enabled, it may happen that while one of them is in progress, another one is requested. In order that the microcontroller knows whether to continue operation or meet a new interrupt request, there is a priority list instructing it what to do.

The priority list offers 3 levels of interrupt priority:

1. Reset! The absolute master. When a reset request arrives, everything is stopped and the microcontroller restarts.
2. Interrupt priority 1 can be disabled by Reset only.
3. Interrupt priority 0 can be disabled by both Reset and interrupt priority 1.

The IP Register (Interrupt Priority Register) specifies which one of existing interrupt sources have higher and which one has lower priority. Interrupt priority is usually specified at the beginning of the program. According to that, there are several possibilities:

- If an interrupt of higher priority arrives while an interrupt is in progress, it will be immediately stopped and the higher priority interrupt will be executed first.
- If two interrupt requests, at different priority levels, arrive at the same time then the higher priority interrupt is serviced first.
- If the both interrupt requests, at the same priority level, occur one after another, the one which came later has to wait until routine being in progress ends.
- If two interrupt requests of equal priority arrive at the same time then the interrupt to be serviced is selected according to the following priority list:

    1. External interrupt INT0
    2. Timer 0 interrupt
    3. External Interrupt INT1
    4. Timer 1 interrupt
    5. Serial Communication Interrupt

- If two interrupt requests, at different priority levels, arrive at the same time then the higher priority interrupt is serviced first.
- If the both interrupt requests, at the same priority level, occur one after another, the one which came later has to wait until routine being in progress ends.
- If two interrupt requests of equal priority arrive at the same time then the interrupt to be serviced is selected according to the following priority list:

1. External interrupt INT0
2. Timer 0 interrupt
3. External Interrupt INT1
4. Timer 1 interrupt
5. Serial Communication Interrupt

IP Register (Interrupt Priority)

The IP register bits specify the priority level of each interrupt (high or low priority).

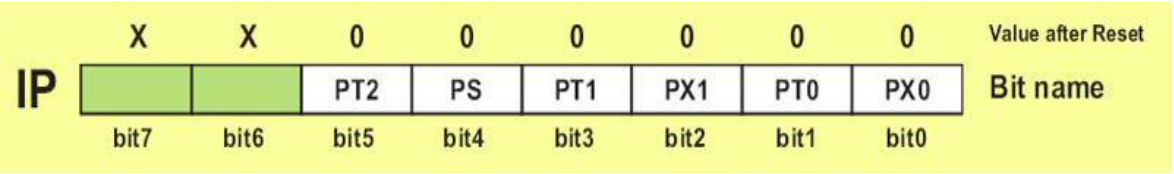


Figure 4.24

- **PS** - Serial Port Interrupt priority bit
  Priority 0
  Priority 1
- **PT1** - Timer 1 interrupt priority
  Priority 0
  Priority 1
- **PX1** - External Interrupt INT1 priority
  Priority 0
  Priority 1
- **PT0** - Timer 0 Interrupt Priority
  Priority 0
  Priority 1
- **PX0** - External Interrupt INT0 Priority
  Priority 0
  Priority 1

Handling Interrupt

When an interrupt request arrives the following occurs:

1. Instruction in progress is ended.
2. The address of the next instruction to execute is pushed on the stack.
3. Depending on which interrupt is requested, one of 5 vectors (addresses) is written to the program counter in accordance to the table below:

Table 4.5

INTERRUPT SOURCE	VECTOR (ADDRESS)
IE0	3 h
TF0	B h
TF1	1B h
RI, TI	23 h

4. These addresses store appropriate subroutines processing interrupts. Instead of them, there are usually jump instructions specifying locations on which these subroutines reside.
5. When an interrupt routine is executed, the address of the next instruction to execute is poped from the stack to the program counter and interrupted program resumes operation from where it left off.

Reset

Reset occurs when the RS pin is supplied with a positive pulse in duration of at least 2 machine cycles (24 clock cycles of crystal oscillator). After that, the microcontroller generates an internal reset signal which clears all SFRs, except SBUF registers, Stack Pointer and ports (the state of the first two ports is not defined, while FF value is written to the ports configuring all their pins as inputs). Depending on surrounding and purpose of device, the RS pin is usually connected to a power-on reset push button or circuit or to both of them. Figure below illustrates one of the simplest circuit providing safe power-on reset.

Basically, everything is very simple: after turning the power on, electrical capacitor is being charged for several milliseconds through a resistor connected to the ground. The pin is driven high during this process. When the capacitor is charged, power supply voltage is already stable

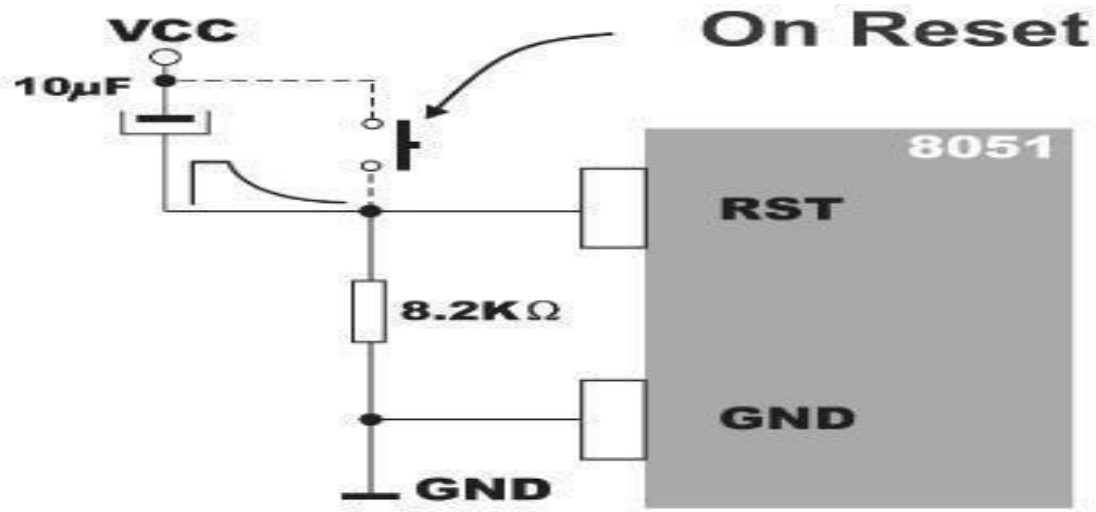


Figure 4.25

Input/Output Ports (I/O Ports)

All 8051 microcontrollers have 4 I/O ports each comprising 8 bits which can be configured as inputs or outputs. Accordingly, in total of 32 input/output pins enabling the microcontroller to be connected to peripheral devices are available for use.

Pin configuration, i.e. whether it is to be configured as an input (1) or an output (0), depends on its logic state. In order to configure a microcontroller pin as an input, it is necessary to apply a logic zero (0) to appropriate I/O port bit. In this case, voltage level on appropriate pin will be 0.

Similarly, in order to configure a microcontroller pin as an input, it is necessary to apply a logic one (1) to appropriate port. In this case, voltage level on appropriate pin will be 5V (as is the case with any TTL input). This may seem confusing but don't loose your patience. It all becomes clear after studying simple electronic circuits connected to an I/O pin.

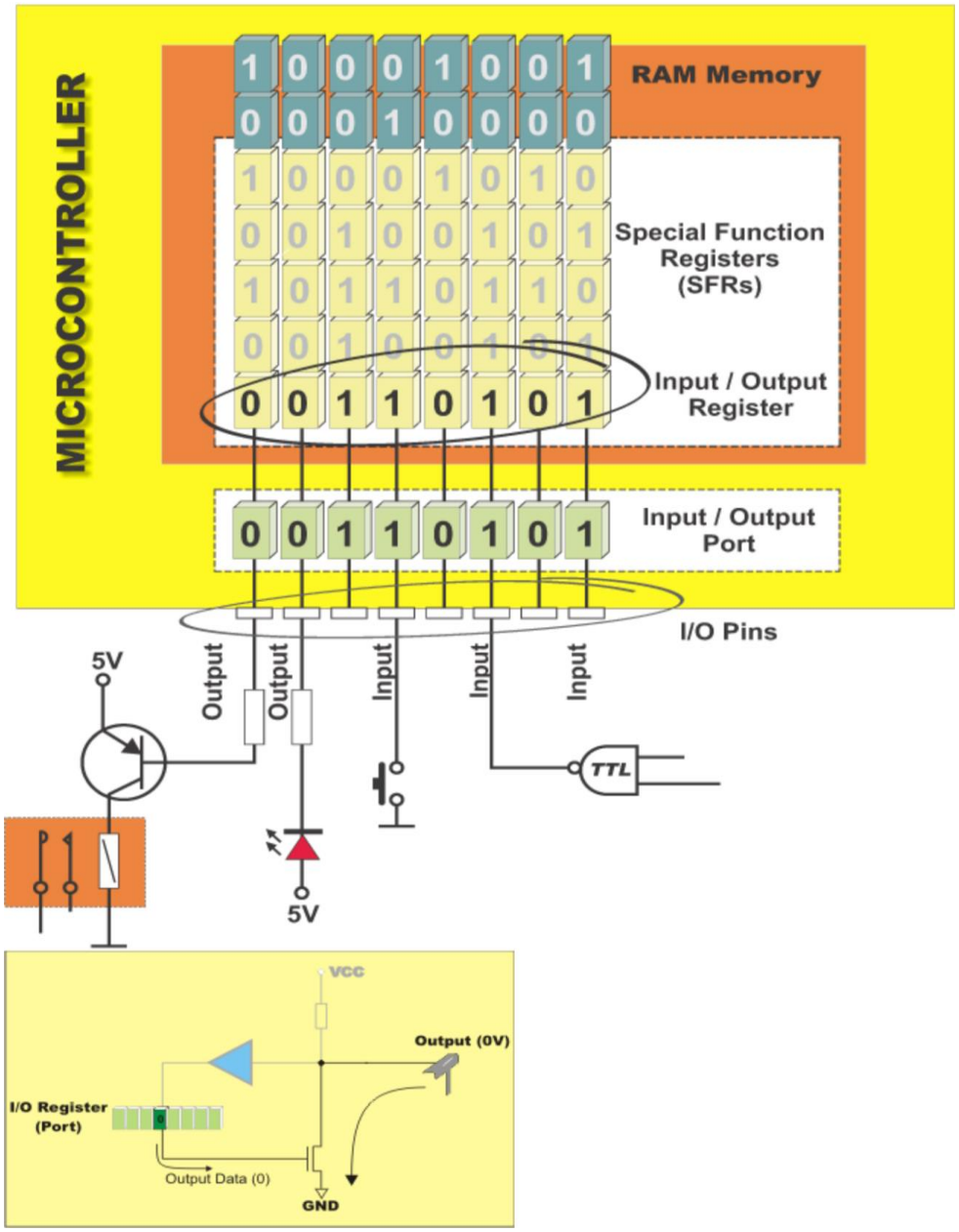


Figure 4.25

Input / Output (I/O) pin

Figure above illustrates a simplified schematic of all circuits within the microcontroller connected to one of its pins. It refers to all the pins except those of the P0 port which do not have pull-up resistors built-in.

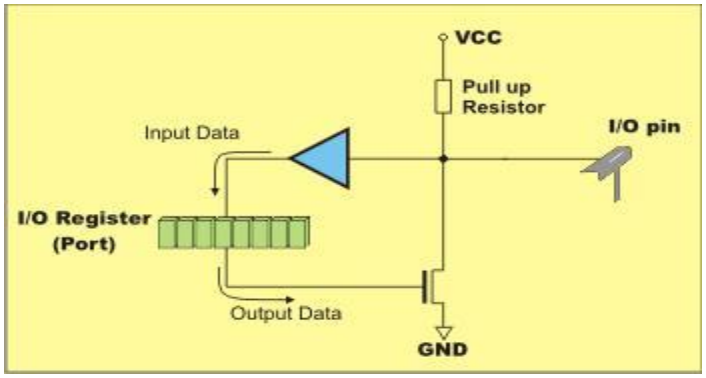


Figure 4.26

Output pin

A logic zero (0) is applied to a bit of the P register. The output FE transistor is turned on, thus connecting the appropriate pin to ground.

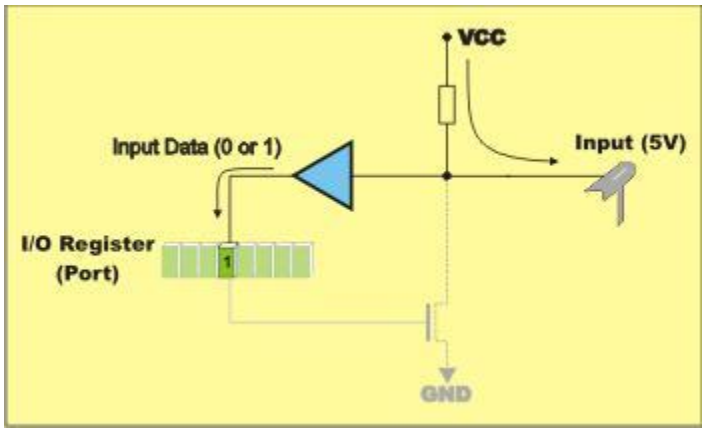


Figure 4.27

Input pin

A logic one (1) is applied to a bit of the P register. The output FE transistor is turned off and the appropriate pin remains connected to the power supply voltage over a pull-up resistor of high resistance.

Port 0

The P0 port is characterized by two functions. If external memory is used then the lower address byte (addresses A0-A7) is applied on it. Otherwise, all bits of this port are configured as inputs/outputs.

The other function is expressed when it is configured as an output. Unlike other ports consisting of pins with built-in pull-up resistor connected by its end to 5 V power supply, pins of this port have this resistor left out. This apparently small difference has its consequences:

Port 1

P1 is a true I/O port, because it doesn't have any alternative functions as is the case with P0, but can be configured as general I/O only. It has a pull-up resistor built-in and is completely compatible with TTL circuits.

Port 2

P2 acts similarly to P0 when external memory is used. Pins of this port occupy addresses intended for external memory chip. This time it is about the higher address byte with addresses A8-A15. When no memory is added, this port can be used as a general input/output port showing features similar to P1.

Port 3

All port pins can be used as general I/O, but they also have an alternative function. In order to use these alternative functions, a logic one (1) must be applied to appropriate bit of the P3 register. In terms of hardware, this port is similar to P0, with the difference that its pins have a pull-up resistor built-in.

Pin's Current limitations

When configured as outputs (logic zero (0)), single port pins can receive a current of 10mA. If all 8 bits of a port are active, a total current must be limited to 15mA (port P0: 26mA). If all ports (32 bits) are active, total maximum current must be limited to 71mA. When these pins are configured as inputs (logic 1), built-in pull-up resistors provide very weak current, but strong enough to activate up to 4 TTL inputs of LS series.

Memory expansion

In case memory (RAM or ROM) built in the microcontroller is not sufficient, it is possible to add two external memory chips with capacity of 64Kb each. P2 and P3 I/O ports are used for their addressing and data transmission.

From the user's point of view, everything works quite simply when properly connected because most operations are performed by the microcontroller itself. The 8051 microcontroller has two pins for data read RD#(P3.7) and PSEN#. The first one is used for reading data from external data memory (RAM), while the other is used for reading data from external program memory (ROM). Both pins are active low. A typical example of memory expansion by adding RAM and ROM chips (Hardward architecture), is shown in figure above.

Even though additional memory is rarely used with the latest versions of the microcontrollers, we will describe in short what happens when memory chips are connected according to the previous schematic. The whole process described below is performed automatically.

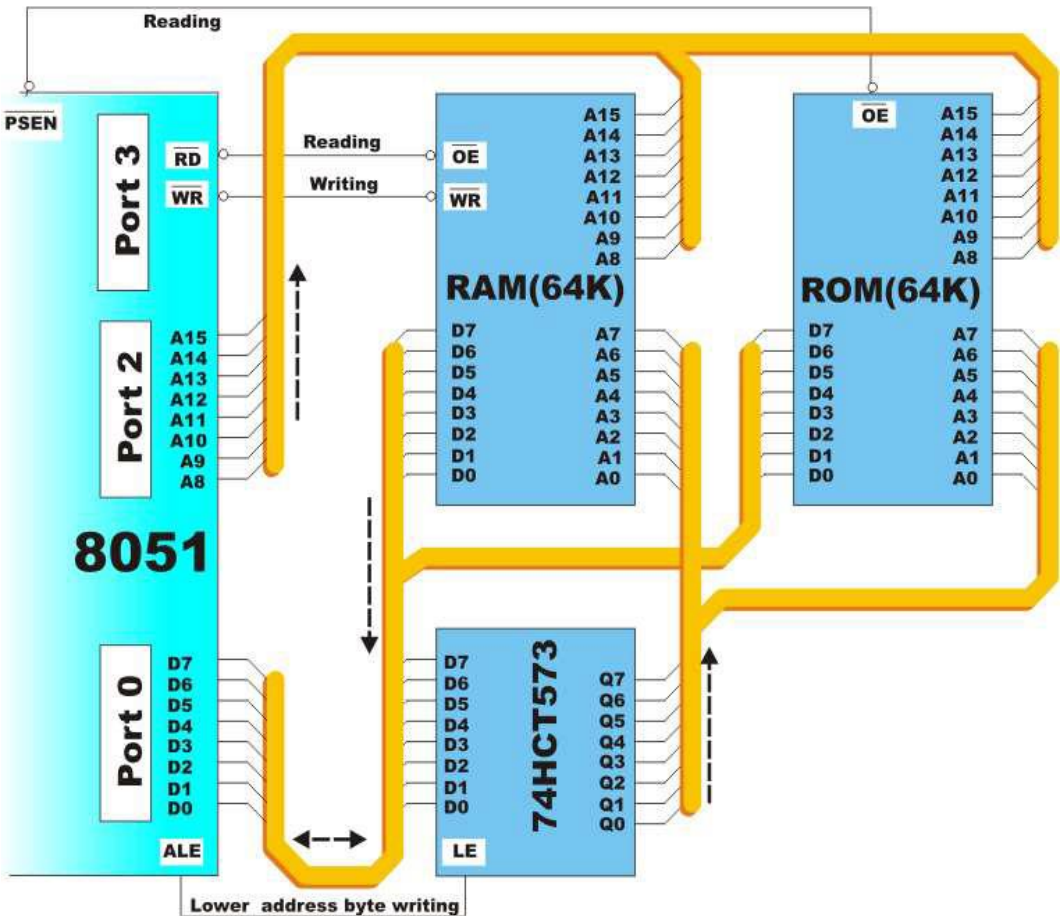


Figure 4.29 External Memory Interfacing

- When the program during execution encounters an instruction which resides in external memory (ROM), the microcontroller will activate its control output ALE and set the first 8 bits of address (A0-A7) on P0. IC circuit 74HCT573 passes the first 8 bits to memory address pins.

- A signal on the ALE pin latches the IC circuit 74HCT573 and immediately afterwards 8 higher bits of address (A8-A15) appear on the port. In this way, a desired location of additional program memory is addressed. It is left over to read its content.

- Port P0 pins are configured as inputs, the PSEN pin is activated and the microcontroller reads from memory chip. Similar occurs when it is necessary to read location from external RAM. Addressing is performed in the same way, while read and write are performed via signals appearing on the control outputs RD (is short for read) or WR (is short for write).

SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

SCHOOL OF BIO AND CHEMICAL ENGINNEERING

DEPARTMENT OF BIOMEDICAL ENGINEERING

UNIT V - Fundamentals of Microprocessor and Microcontroller – SBMA1501

UNIT -5

ARDUINO UNO

Arduino – Architecture, Pin diagram, Programming Structure, Simple program to blink LED, Subroutine, 16x2LED display, Interfacing with Arduino: LCD, Temperature Sensor, Humidity Sensor and ultrasonic sensor.

Arduino is an open source computer hardware and software company, project and user community that designs and manufactures single-board microcontrollers and microcontroller kits for building digital devices and interactive objects that can sense and control objects in the physical and digital world.

Arduino board designs use a variety of microprocessors and controllers. The boards are equipped with sets of digital and analog input/output (I/O) pins that may be interfaced to various expansion boards (shields) or Breadboards (other circuits on them).

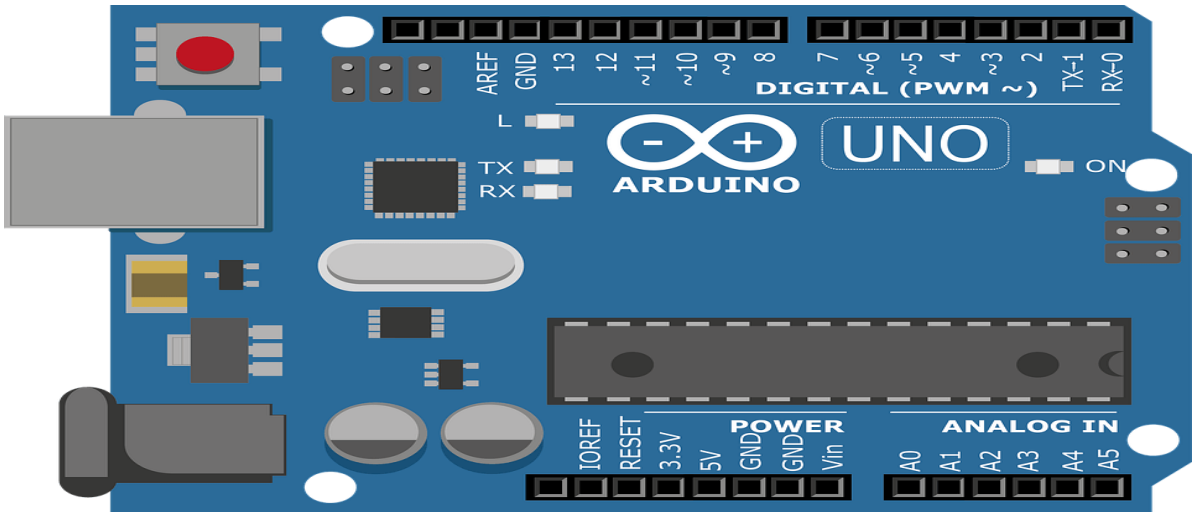


Figure 5.1 ARDUINO UNO

Arduino Uno is a microcontroller board based on the ATmega328P (datasheet). It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz quartz crystal, a USB connection, a power jack, an ICSP header and a reset button. It contains everything needed to support the microcontroller, simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started. You can tinker with your UNO without worrying too much about doing something wrong, worst case scenario you can replace the chip for a few dollars and start over again.

“Uno” means one in Italian and was chosen to mark the release of Arduino Software (IDE) 1.0. The Uno board and version 1.0 of Arduino Software (IDE) were the reference versions of Arduino, now evolved to newer releases. The Uno board is the first in a series of USB Arduino boards, and the reference model for the Arduino platform. (IDE = integrated development environment).

Architecture

Here Architecture is of Arduino or precisely the IC of Arduino (ATmega328p). The ATmega328/P is a low-power CMOS 8-bit microcontroller based on the **AVR®** enhanced **RISC** (reduced instruction set computer) architecture. In Order to maximize performance and parallelism, the AVR uses **Harvard** architecture – with separate memories and buses for program and data. Instruction in the program memory are executed with a single level of pipelining. The clock is controlled by an external **16MHz Crystal Oscillator**.

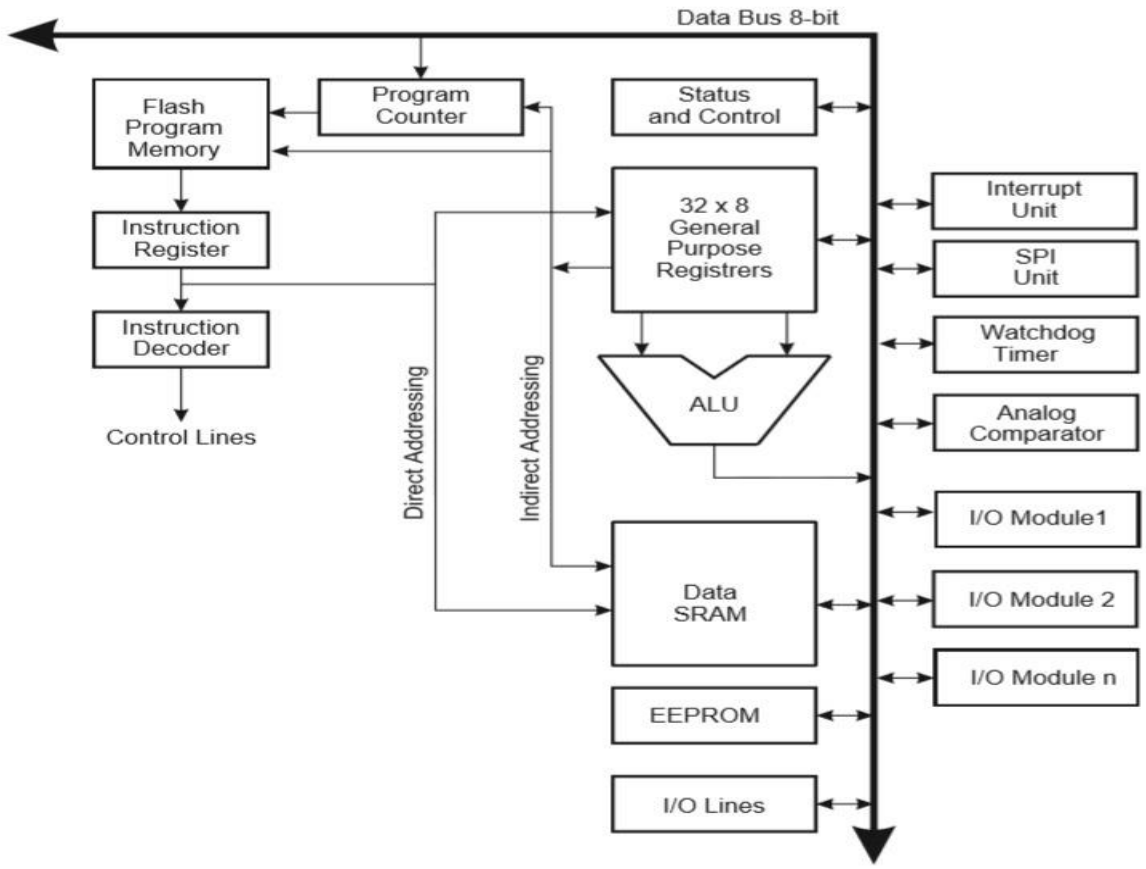


Figure 5.2 Arduino Architecture

The basic working of CPU of ATmega328

1. The data is uploaded in serial via the port (being uploaded from the computer's Arduino IDE). The data is decoded and then the instructions are sent to **instruction register** and it decodes the instructions on the same clock pulse.

2. On the next clock pulse the next set of instructions are loaded in instruction register.

3. **In general purpose registers** the registers are of 8-bit but there are 3 16-bit registers also.

   a. **8-bit** registers are used to store data for normal calculations and results.

   b. **16-bit** registers are used to store data of timer counter in 2 different register. Eg. X-low & X-high. They are fast, and are used to store specific hardware functions.

4. **EEPROM** stores data permanently even if the power is cut out. Programming inside a EEPROM is slow.

5. **Interrupt Unit** checks whether there is an interrupt for the execution of instruction to be executed in ISR (Interrupt Service Routine).

6. **Serial Peripheral Interface (SPI)** is an interface bus commonly used to send data between microcontrollers and small peripherals such as Camera, Display, SD cards, etc. It uses separate clock and data lines, along with a select line to choose the device you wish to talk to.

7. **Watchdog** timer is used to detect and recover from MCU malfunctioning.

8. **Analog comparator** compares the input values on the positive and negative pin, when the value of positive pin is higher the output is set.

9. **Status and control** is used to control the flow of execution of commands by checking other blocks inside the CPU at regular intervals.

10. **ALU (Arithmetic and Logical unit)** The high performance AVR ALU operates in direct connection with all the 32 general purpose working registers. Within a single clock cycle, arithmetic operations b/w general purpose registers are executed. The ALU operations are divided into 3 main categories – arithmetic, logical and bit-function.

11. **I/O pins** The digital inputs and outputs (digital I/O) on the Arduino are what allow you to connect the Arduino sensors, actuators, and other ICs. Learning how to use them will allow you to use the Arduino to do some really useful things, such as reading switch inputs, lighting indicators, and controlling relay outputs.

EXTRA

1. **Automatic (Software) Reset:** Rather then requiring a physical press of the reset button before an upload, the Arduino is designed in a way that allows it to be reset by software running on a connected computer. The Arduino Software (IDE) uses this capability to allow you to upload code by simply pressing the upload button in the Arduino environment. This means that the bootloader can have a shorter timeout, as the lowering of DTR (Data Terminal Ready) can be well-coordinated with the start of the upload.

2. **Firmware:** Firmware is a software program or set of instructions programmed on a hardware device. It provides the necessary instructions for how the device communicates with the other computer hardware. Firmware is held in non-volatile memory devices such as ROM.

3. **To check** whether the firmware is installed in your Arduino or not just press the reset button and if the **inbuilt LED flickers** (**on pin 13)** the firmware is present

Pin Configuration

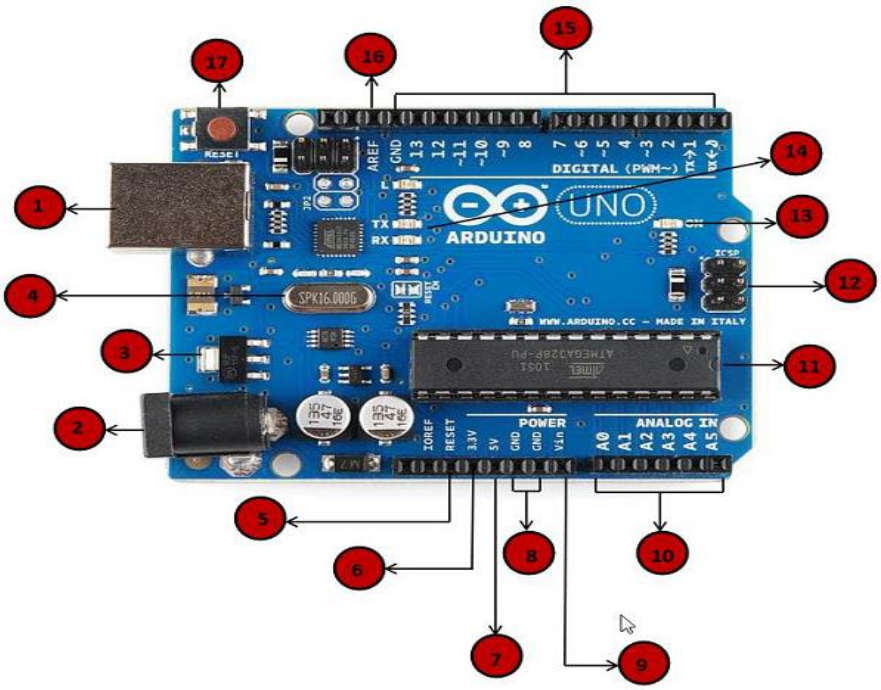


Figure 5.3 Pin Configuration

	**Power USB** Arduino board can be powered by using he USB cable from your computer. All you need to do is connect the USB cable to the USB connection (1).
	**Power (Barrel Jack)** Arduino boards can be powered directly from the AC mains power supply by connecting it to the Barrel Jack (2).
	**Voltage Regulator** The function of the voltage regulator is to control the voltage given to the Arduino board and stabilize the DC voltages used by the processor and other elements.
	**Crystal Oscillator** The crystal oscillator helps Arduino in dealing with time issues. How does Arduino calculate time? The answer is, by using the crystal oscillator. The number printed on top of the Arduino crystal is 16.000H9H. It tells us that the frequency is 16,000,000 Hertz or 16 MHz.
	**Arduino Reset** You can reset your Arduino board, i.e., start your program from the beginning. You can reset the UNO board in two ways. First, by using the reset button (17) on the board. Second, you can connect an external reset button to the Arduino pin labelled RESET (5).
	**Pins (3.3, 5, GND, Vin)** • 3.3V (6) − Supply 3.3 output volt • 5V (7) − Supply 5 output volt • Most of the components used with Arduino board works fine with 3.3 volt and 5 volt. • GND (8)(Ground) − There are several GND pins on the Arduino, any of which can be used to ground your circuit. • Vin (9) − This pin also can be used to power the Arduino board from an external power source, like AC mains power supply.

10	**Analog pins** The Arduino UNO board has six analog input pins A0 through A5. These pins can read the signal from an analog sensor like the humidity sensor or temperature sensor and convert it into a digital value that can be read by the microprocessor.
11	**Main microcontroller** Each Arduino board has its own microcontroller (11). You can assume it as the brain of your board. The main IC (integrated circuit) on the Arduino is slightly different from board to board. The microcontrollers are usually of the ATMEL Company. You must know what IC your board has before loading up a new program from the Arduino IDE. This information is available on the top of the IC. For more details about the IC construction and functions, you can refer to the data sheet.
12	**ICSP pin** Mostly, ICSP (12) is an AVR, a tiny programming header for the Arduino consisting of MOSI, MISO, SCK, RESET, VCC, and GND. It is often referred to as an SPI (Serial Peripheral Interface), which could be considered as an "expansion" of the output. Actually, you are slaving the output device to the master of the SPI bus.
13	**Power LED indicator** This LED should light up when you plug your Arduino into a power source to indicate that your board is powered up correctly. If this light does not turn on, then there is something wrong with the connection.
14	**TX and RX LEDs** On your board, you will find two labels: TX (transmit) and RX (receive). They appear in two places on the Arduino UNO board. First, at the digital pins 0 and 1, to indicate the pins responsible for serial communication. Second, the TX and RX led (13). The TX led flashes with different speed while sending the serial data. The speed of flashing depends on the baud rate used by the board. RX flashes during the receiving process.

	**Digital I/O** The Arduino UNO board has 14 digital I/O pins (15) (of which 6 provide PWM (Pulse Width Modulation) output. These pins can be configured to work as input digital pins to read logic values (0 or 1) or as digital output pins to drive different modules like LEDs, relays, etc. The pins labeled "~" can be used to generate PWM.
	**AREF** AREF stands for Analog Reference. It is sometimes, used to set an external reference voltage (between 0 and 5 Volts) as the upper limit for the analog input pins.

Functions

Functions allow structuring the programs in segments of code to perform individual tasks. The typical case for creating a function is when one needs to perform the same action multiple times in a program.

Standardizing code fragments into functions has several advantages −

Functions help the programmer stay organized. Often this helps to conceptualize the program.

Functions codify one action in one place so that the function only has to be thought about and debugged once.

This also reduces chances for errors in modification, if the code needs to be changed.

Functions make the whole sketch smaller and more compact because sections of code are reused many times.

They make it easier to reuse code in other programs by making it modular, and using functions often makes the code more readable.

There are two required functions in an Arduino sketch or a program i.e. setup () and loop(). Other functions must be created outside the brackets of these two functions.

The most common syntax to define a function is −

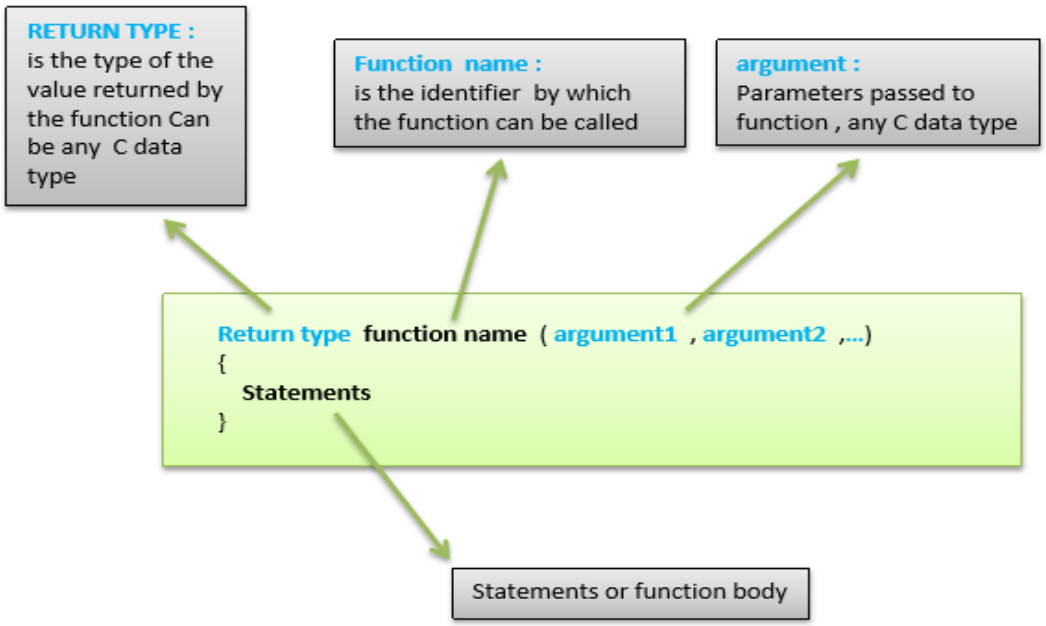


Figure 5.4

Function Declaration

A function is declared outside any other functions, above or below the loop function.

We can declare the function in two different ways −

The first way is just writing the part of the function called a function prototype above the loop function, which consists of −

Function return type

Function name

Function argument type, no need to write the argument name

Function prototype must be followed by a semicolon (;).

The following example shows the demonstration of the function declaration using the first method.

Example

```
int sum_func (int x, int y) // function declaration {
  int z = 0;
  z = x+y ;
```

```
  return z; // return the value
}
void setup () {
  Statements // group of statements
}
Void loop () {
  int result = 0 ;
  result = Sum_func (5,6) ; // function call
}
```

The second part, which is called the function definition or declaration, must be declared below the loop function, which consists of −

Function return type

Function name

Function argument type, here you must add the argument name

The function body (statements inside the function executing when the function is called)

The following example demonstrates the declaration of function using the second method.

Example

```
int sum_func (int , int ) ; // function prototype
void setup () {
  Statements // group of statements
}
Void loop () {
  int result = 0 ;
  result = Sum_func (5,6) ; // function call
}
int sum_func (int x, int y) // function declaration {
  int z = 0;
  z = x+y ;
  return z; // return the value
}
```

Program to blink LED

LEDs are small, powerful lights that are used in many different applications. To start, we will work on blinking an LED, the Hello World of microcontrollers. It is as simple as turning a light on and off. Establishing this important baseline will give you a solid foundation as we work towards experiments that are more complex.

Components Required

You will need the following components −

1 × Breadboard

1 × Arduino Uno R3

1 × LED

1 × 330Ω Resistor

2 × Jumper

Procedure

Follow the circuit diagram and hook up the components on the breadboard as shown in the image given below.

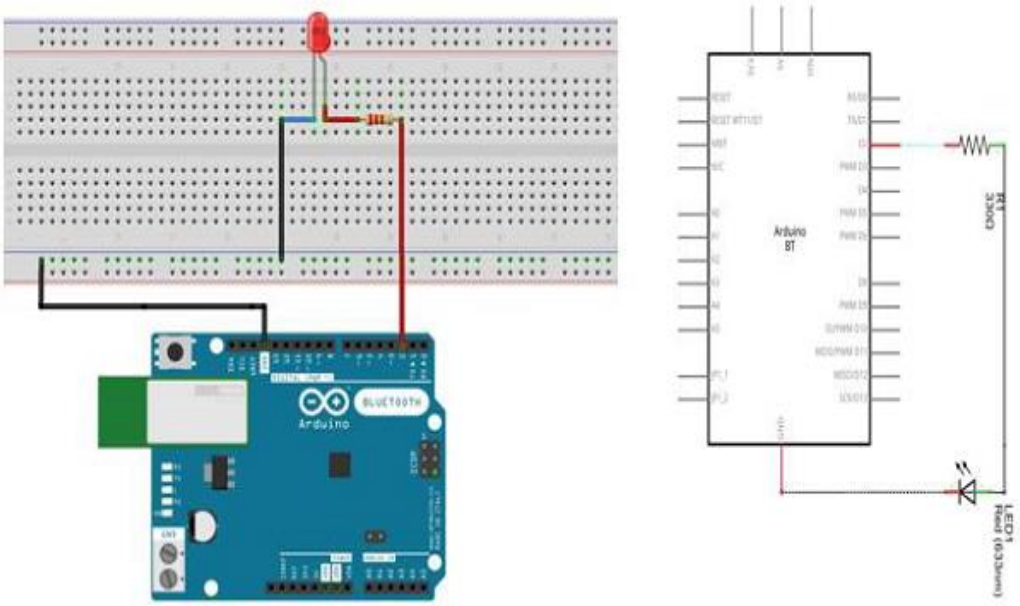


Figure 5.5

Note − To find out the polarity of an LED, look at it closely. The shorter of the two legs, towards the flat edge of the bulb indicates the negative terminal.

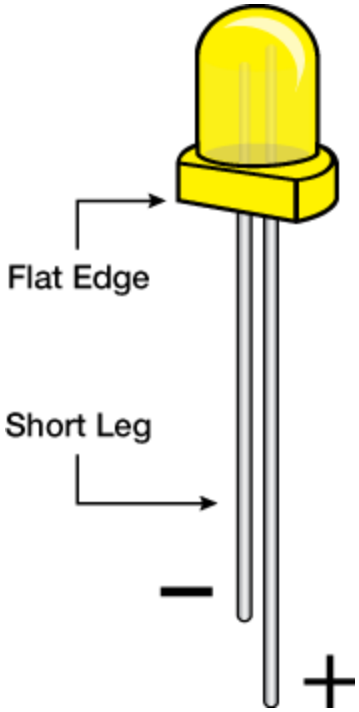


Figure 5.6

LED

Components like resistors need to have their terminals bent into 90° angles in order to fit the breadboard sockets properly. You can also cut the terminals shorter.

Resistors

Sketch

Open the Arduino IDE software on your computer. Coding in the Arduino language will control your circuit. Open the new sketch File by clicking New.

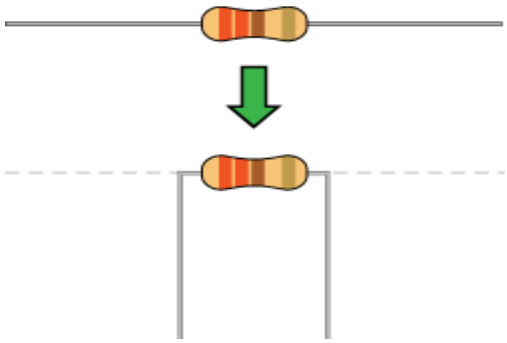


Figure 5.6

Sketch

Open the Arduino IDE software on your computer. Coding in the Arduino language will control your circuit. Open the new sketch File by clicking New.

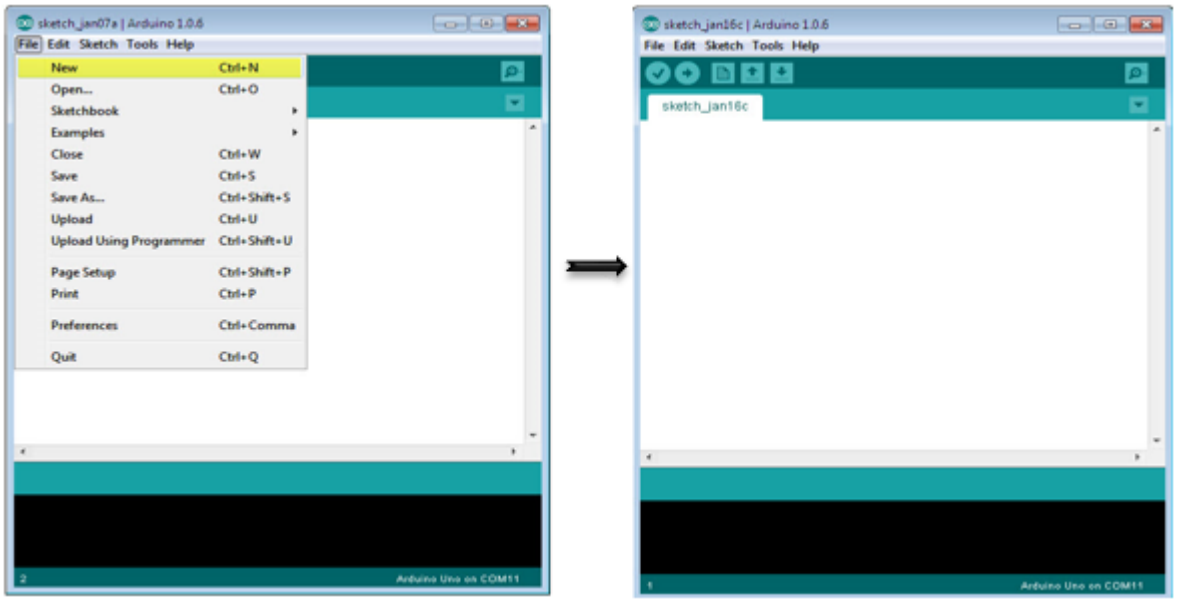


Figure 5.7

Arduino Code

```
/*
  Blink
  Turns on an LED on for one second, then off for one second, repeatedly.
*/
// the setup function runs once when you press reset or power the board
void setup() {  // initialize digital pin 13 as an output.
  pinMode(2, OUTPUT);
}
// the loop function runs over and over again forever
void loop() {
  digitalWrite(2, HIGH); // turn the LED on (HIGH is the voltage level)
  delay(1000); // wait for a second
  digitalWrite(2, LOW); // turn the LED off by making the voltage LOW
  delay(1000); // wait for a second
}
```

Code to Note

pinMode(2, OUTPUT) − Before you can use one of Arduino's pins, you need to tell Arduino Uno R3 whether it is an INPUT or OUTPUT. We use a built-in "function" called pinMode() to do this.

digitalWrite(2, HIGH) − When you are using a pin as an OUTPUT, you can command it to be HIGH (output 5 volts), or LOW (output 0 volts).

Result

You should see your LED turn on and off. If the required output is not seen, make sure you have assembled the circuit correctly, and verified and uploaded the code to your board.

LCD 16x2 Interfacing With Arduino Uno

Introduction

LCDs (Liquid Crystal Displays) are used in embedded system applications for displaying various parameters and status of the system.

LCD 16x2 is a 16-pin device that has 2 rows that can accommodate 16 characters each.

LCD 16x2 can be used in 4-bit mode or 8-bit mode.

It is also possible to create custom characters.

It has 8 data lines and 3 control lines that can be used for control purposes.

For more information about LCD 16x2 and how to use it, refer the topic LCD 16x2 module in the sensors and modules section.

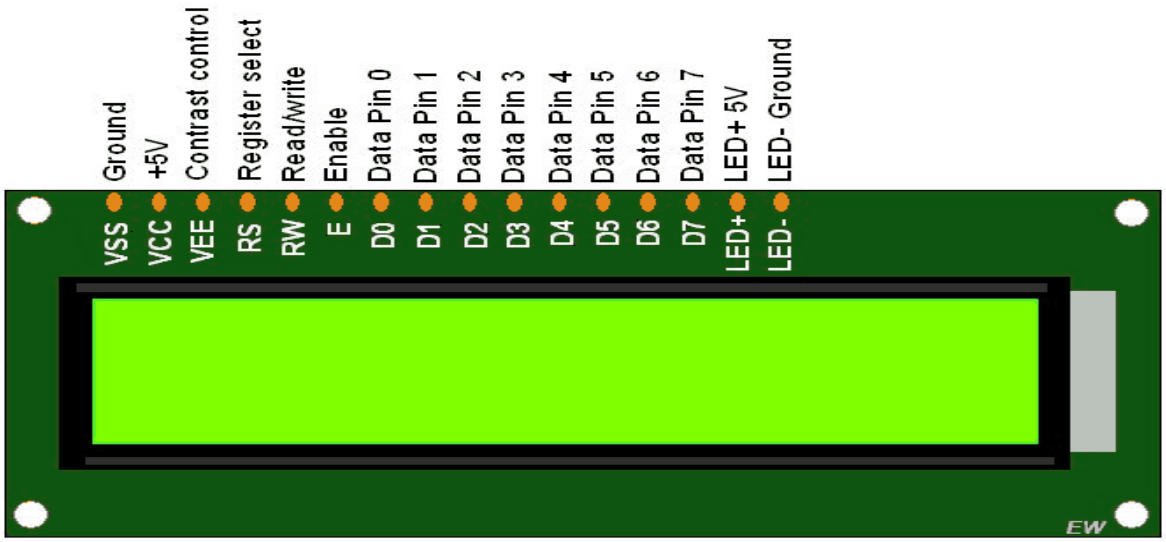


Figure 5.8

Interfacing Diagram

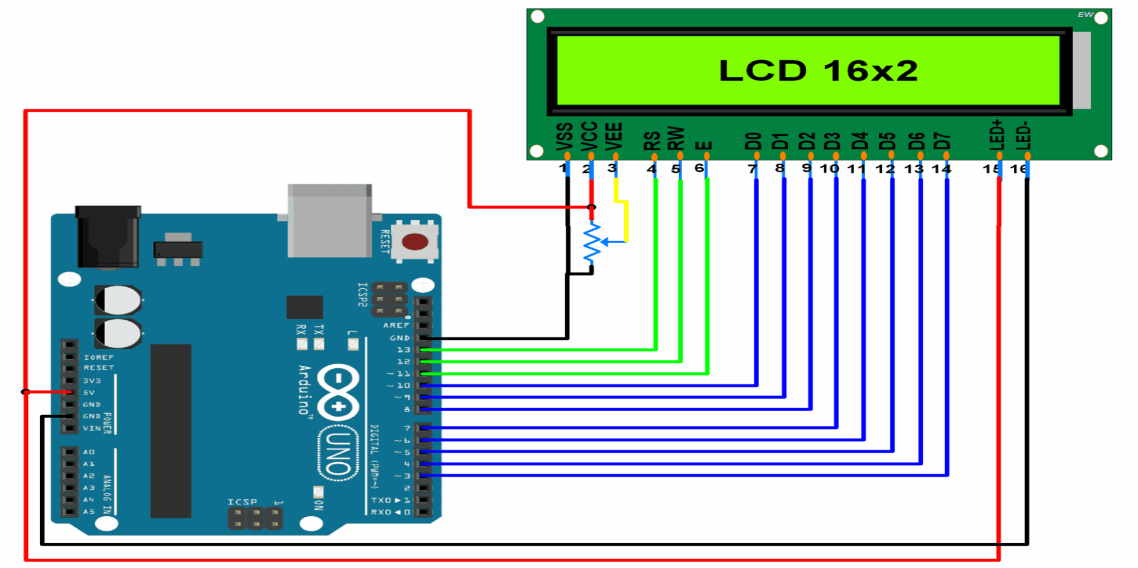


Figure 5.9

Sketch For Displaying Data On LCD (Custom Characters And Regular Characters)

```
#include <LiquidCrystal.h>
/* Create object named lcd of the class LiquidCrystal */
LiquidCrystal lcd(13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3);  /* For 8-bit mode */
//LiquidCrystal lcd(13, 12, 11, 6, 5, 4, 3);      /* For 4-bit mode */
unsigned char Character1[8] = { 0x04, 0x1F, 0x11, 0x11, 0x1F, 0x1F, 0x1F, 0x1F };        /*
Custom Character 1 */
```

```
unsigned char Character2[8] = { 0x01, 0x03, 0x07, 0x1F, 0x1F, 0x07, 0x03, 0x01 };        /*
Custom Character 2 */
void setup() {
  lcd.begin(16,2);        /* Initialize 16x2 LCD */
  lcd.clear();    /* Clear the LCD */
  lcd.createChar(0, Character1);         /* Generate custom character */
  lcd.createChar(1, Character2);
}
void loop() {
  lcd.setCursor(0,0);    /* Set cursor to column 0 row 0 */
  lcd.print("Hello!!!!");          /* Print data on display */
  lcd.setCursor(0,1);
  lcd.write(byte(0));    /* Write a character to display */
  lcd.write(1);
}
```

Functions Used

1. LiquidCrystal object_name(rs,rw,en,d0,d1,d2,d3,d4,d5,d6,d7)

   LiquidCrystal object_name(rs,rw,en,d4,d5,d6,d7)

   This function defines an object named object_name of the class LiquidCrystal.

   rs, rw and en are the pin numbers of the Arduino board that are connected to rs, rw and en of LCD.

   d0, d1, d2, d3, d4, d5, d6 and d7 are the pin numbers of the Arduino board that are connected to data pins D1, D2, D3, D4, D5, D6 and D7 of LCD.

   Example, LiquidCrystal lcd(13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3). This makes use of LCD in 8-bit mode.

   Example, LiquidCrystal lcd(13, 12, 11, 6, 5, 4, 3). This makes use of LCD in 4-bit mode.

2. lcd.begin(cols,rows)

This function is used to define the number of rows and columns the LCD has and to initialize the LCD.

Needs to be called before calling other functions, once the object is defined using the function in point 1.

Example, for 16x2 LCD we write lcd.begin(16,2). lcd is the name of the object of the class LiquidCrystal. 16 is the number of columns and 2 is the number of rows.

3. lcd.setCursor(col,row)

This function positions the cursor of the LCD to a location specified by the row and column parameters.

col is the column number at which the cursor should be at (0 for column 1, 4 for column 5 and so on).

row is the row number at which the cursor should be at (0 for row 1, 1 for row 2).

Example, for setting the cursor at the 5th column in the 2nd row, lcd.setCursor(4,1). lcd is the name of the object of the class LiquidCrystal.

4. lcd.createChar(num,data)

This function is used to create a new custom character for use on the LCD.

num is the CGRAM location (0 to 7) at which the custom character is to be stored.

data is array of eight bytes which represent the custom character.

Custom character can be of 5x8 pixels only.

Each custom character is specified by an array of eight bytes, one for each row. The five least significant bits of each byte determine the pixels in that row.

To display a custom character on the screen, write() function needs to be used. CGRAM location number (0 to 7) of the custom character which is to be displayed on LCD is passed as an argument to the write function.

Note : When referencing custom character "0", you need to cast it as a byte, otherwise the compiler throws an error.

In this section, we will learn how to interface our Arduino board with different sensors. We will discuss the following sensors -

- Humidity sensor (DHT22)
- Temperature sensor (LM35)
- ULTRASONIC SENSOR

Humidity Sensor (DHT22)

The DHT-22 (also named as AM2302) is a digital-output, relative humidity, and temperature sensor. It uses a capacitive humidity sensor and a thermistor to measure the surrounding air, and sends a digital signal on the data pin. In this example, you will learn how to use this sensor with Arduino UNO. The room temperature and humidity will be printed to the serial monitor.

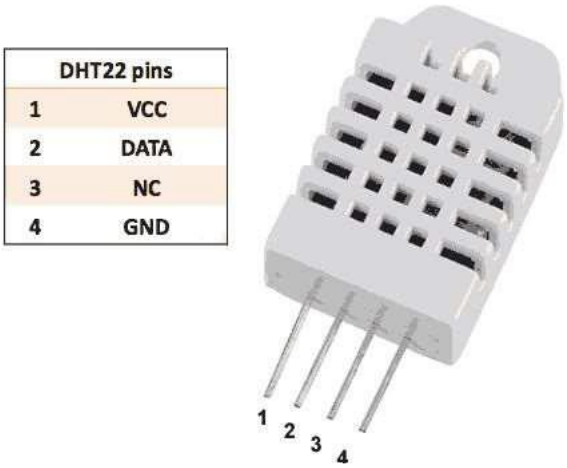


Figure 5.10

The DHT-22 Sensor

The connections are simple. The first pin on the left to 3-5V power, the second pin to the data input pin and the right-most pin to the ground.

Technical Details

- Power: 3-5V
- Max Current: 2.5mA
- Humidity: 0-100%, 2-5% accuracy
- Temperature: -40 to 80°C, ±0.5°C accuracy

Components Required

You will need the following components:

- 1x Breadboard
- 1x Arduino Uno R3
- 1x DHT22
- 1X10K ohm resistor

Procedure

Follow the circuit diagram and hook up the components on the breadboard as shown in the image below.

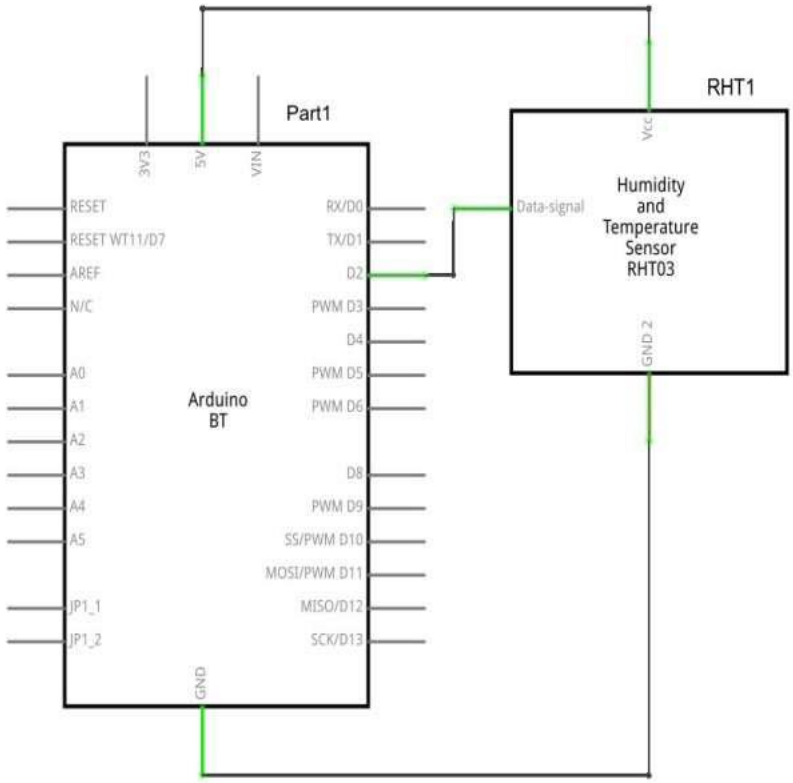


Figure 5.11

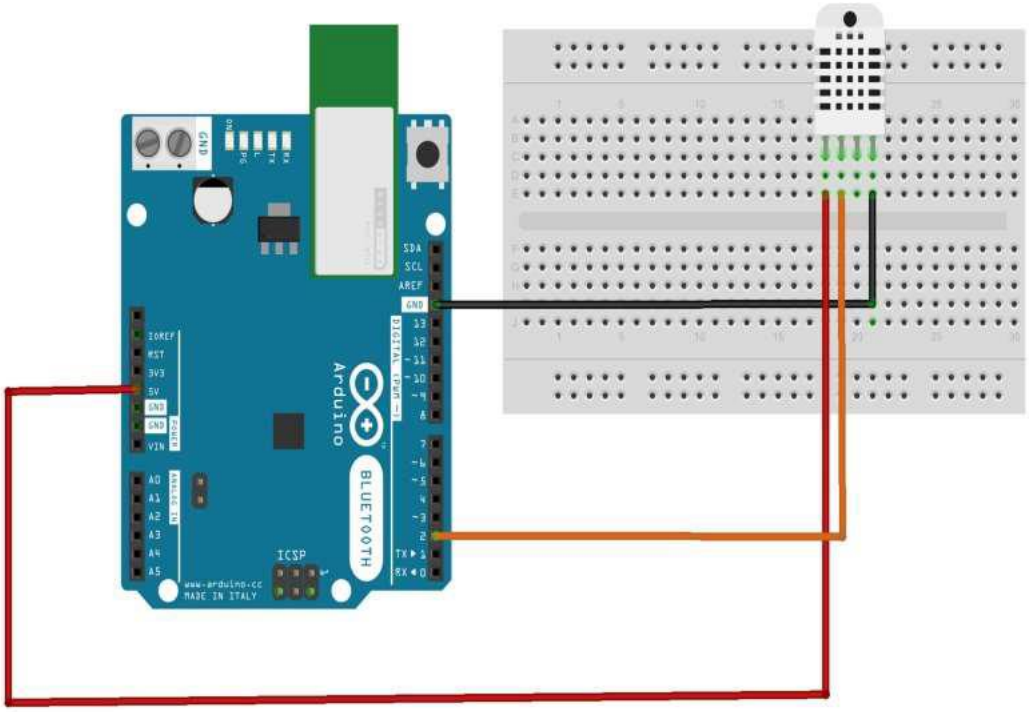


Figure 5.12

Sketch

Open the Arduino IDE software on your computer. Coding in the Arduino language will control your circuit. Open a new sketch File by clicking New.

Arduino Code

```
// Example testing sketch for various DHT humidity/temperature sensors
#include "DHT.h"
#define DHTPIN 2 // what digital pin we're connected to
// Uncomment whatever type you're using!
//#define DHTTYPE DHT11 // DHT 11
#define DHTTYPE DHT22 // DHT 22 (AM2302), AM2321
//#define DHTTYPE DHT21 // DHT 21 (AM2301)
// Connect pin 1 (on the left) of the sensor to +5V
// NOTE: If using a board with 3.3V logic like an Arduino Due connect pin 1
// to 3.3V instead of 5V!
// Connect pin 2 of the sensor to whatever your DHTPIN is
// Connect pin 4 (on the right) of the sensor to GROUND
// Connect a 10K resistor from pin 2 (data) to pin 1 (power) of the sensor
```

```
// Initialize DHT sensor.
// Note that older versions of this library took an optional third parameter to
// tweak the timings for faster processors. This parameter is no longer needed
// as the current DHT reading algorithm adjusts itself to work on faster procs.
DHT dht(DHTPIN, DHTTYPE);
void setup() {
Serial.begin(9600);
Serial.println("DHTxx test!");
dht.begin();
}
void loop() {
delay(2000); // Wait a few seconds between measurements
float h = dht.readHumidity();
// Reading temperature or humidity takes about 250 milliseconds!
float t = dht.readTemperature();
// Read temperature as Celsius (the default)
float f = dht.readTemperature(true);
// Read temperature as Fahrenheit (isFahrenheit = true)
// Check if any reads failed and exit early (to try again).
if (isnan(h) || isnan(t) || isnan(f))
{
Serial.println("Failed to read from DHT sensor!");
return;
Arduino
166
}
// Compute heat index in Fahrenheit (the default)
float hif = dht.computeHeatIndex(f, h);
// Compute heat index in Celsius (isFahreheit = false)
float hic = dht.computeHeatIndex(t, h, false);
Serial.print ("Humidity: ");
Serial.print (h);
Serial.print (" %\t");
Serial.print ("Temperature: ");
```

```
Serial.print (t);
Serial.print (" *C ");
Serial.print (f);
Serial.print (" *F\t");
Serial.print ("Heat index: ");
Serial.print (hic);
Serial.print (" *C ");
Serial.print (hif);
Serial.println (" *F");
}
```

Code to Note

DHT22 sensor has four terminals (Vcc, DATA, NC, GND), which are connected to the board as follows:

- DATA pin to Arduino pin number 2
- Vcc pin to 5 volt of Arduino board
- GND pin to the ground of Arduino board
- We need to connect 10k ohm resistor (pull up resistor) between the DATA and the Vcc pin

Once hardware connections are done, you need to add DHT22 library to your Arduino library file as described earlier.

Temperature Sensor

The Temperature Sensor LM35 series are precision integrated-circuit temperature devices with an output voltage linearly proportional to the Centigrade temperature.

The LM35 device has an advantage over linear temperature sensors calibrated in Kelvin, as the user is not required to subtract a large constant voltage from the output to obtain convenient Centigrade scaling. The LM35 device does not require any external calibration or trimming to provide typical accuracies of ±¼°C at room temperature and ±¾°C over a full −55°C to 150°C temperature range.

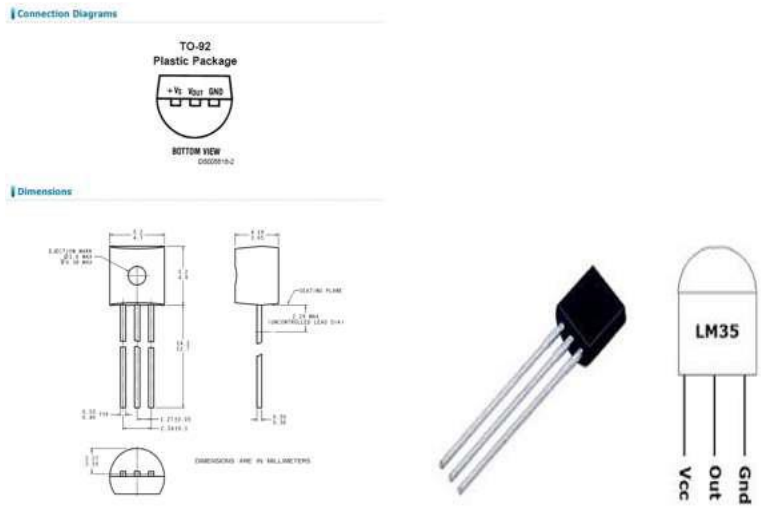


Figure 5.13

Technical Specifications

- Calibrated directly in Celsius (Centigrade)
- Linear + 10-mV/°C scale factor
- 0.5°C ensured accuracy (at 25°C)
- Rated for full −55°C to 150°C range
- Suitable for remote applications

Components Required

You will need the following components:

- 1x Breadboard
- 1x Arduino Uno R3
- 1x LM35 sensor

Procedure

Follow the circuit diagram and hook up the components on the breadboard as shown in the image given below

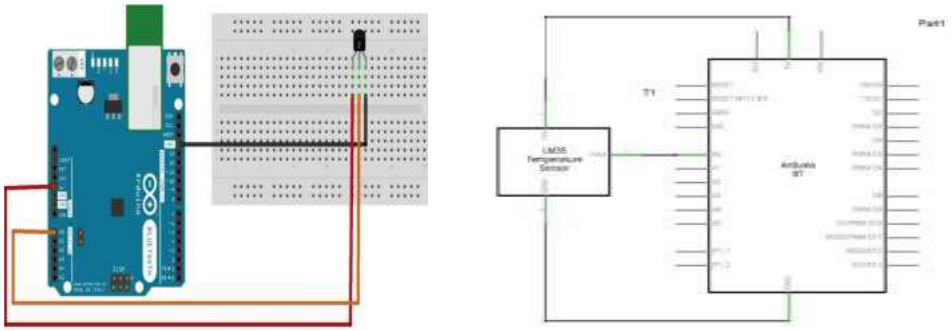


Figure 5.14

Sketch

Open the Arduino IDE software on your computer. Coding in the Arduino language will control your circuit. Open a new sketch File by clicking New.

Arduino Code

```
float temp;
int tempPin = 0;
void setup()
{
Serial.begin(9600);
}
void loop()
{
temp = analogRead(tempPin);
// read analog volt from sensor and save to variable temp
temp = temp * 0.48828125;
// convert the analog volt to its temperature equivalent
Serial.print("TEMPERATURE = ");
Serial.print(temp); // display temperature value
Serial.print("*C");
Serial.println();
delay(1000); // update sensor reading each one second
}
```

Code to Note

LM35 sensor has three terminals - Vs, Vout and GND. We will connect the sensor as follows:

- Connect the +Vs to +5v on your Arduino board.
- Connect Vout to Analog0 or A0 on Arduino board.
- Connect GND with GND on Arduino.

The Analog to Digital Converter (ADC) converts analog values into a digital approximation based on the formula ADC Value = sample * 1024 / reference voltage (+5v). So with a +5 volt reference, the digital approximation will be equal to input voltage * 205.

Ultrasonic Sensor

The HC-SR04 ultrasonic sensor uses SONAR to determine the distance of an object justlike the bats do. It offers excellent non-contact range detection with high accuracy and stable readings in an easy-to-use package from 2 cm to 400 cm or 1" to 13 feet. The operation is not affected by sunlight or black material, although acoustically, soft materials like cloth can be difficult to detect. It comes complete with ultrasonic transmitter and receiver module.

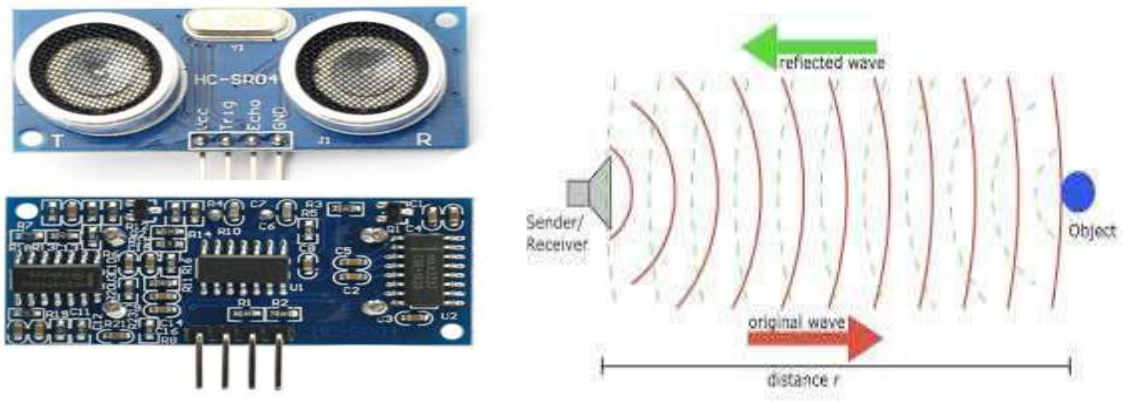


Figure 5.15

Technical Specifications

- Power Supply:+5V DC
- Quiescent Current: <2mA
- Working Current: 15mA

 Effectual Angle: <15°

 Ranging Distance: 2cm – 400 cm/1″ – 13ft

Resolution: 0.3 cm

 Measuring Angle: 30 degree

Components Required

You will need the following components:

 1x Breadboard

 1x Arduino Uno R3

 1x ULTRASONIC Sensor (HC-SR04)

Procedure

Follow the circuit diagram and make the connections as shown in the image given below.

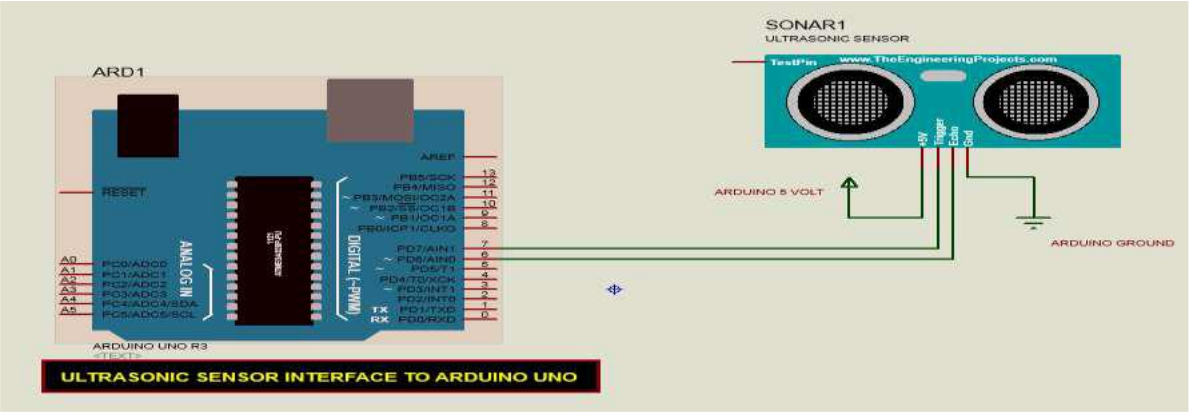


Figure 5.16

Sketch

Open the Arduino IDE software on your computer. Coding in the Arduino language will control your circuit. Open a new sketch File by clicking New.

Arduino Code

```
const int pingPin = 7; // Trigger Pin of Ultrasonic Sensor
const int echoPin = 6; // Echo Pin of Ultrasonic Sensor
void setup()
```

```
{
Serial.begin(9600); // Starting Serial Terminal
}
void loop()
{
long duration, inches, cm;
pinMode(pingPin, OUTPUT);
digitalWrite(pingPin, LOW);
delayMicroseconds(2);
digitalWrite(pingPin, HIGH);
delayMicroseconds(10);
digitalWrite(pingPin, LOW);
pinMode(echoPin, INPUT);
duration = pulseIn(echoPin, HIGH);
inches = microsecondsToInches(duration);
cm = microsecondsToCentimeters(duration);
Serial.print(inches);
Serial.print("in, ");
Serial.print(cm);
Arduino
182
Serial.print("cm");
Serial.println();
delay(100);
}
long microsecondsToInches(long microseconds)
{
return microseconds / 74 / 2;
}
long microsecondsToCentimeters(long microseconds)
{
return microseconds / 29 / 2;
}
```

Code to Note

The Ultrasonic sensor has four terminals - +5V, Trigger, Echo, and GND connected as follows:

- Connect the +5V pin to +5v on your Arduino board.
- Connect Trigger to digital pin 7 on your Arduino board.
- Connect Echo to digital pin 6 on your Arduino board.
- Connect GND with GND on Arduino.

In our program, we have displayed the distance measured by the sensor in inches and cm via the serial port.